intel®

IA-32 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

下巻：
システム・プログラミング・ガイド

注記：
『IA-32 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』は、次の 4 巻から構成されています。

上巻：基本アーキテクチャ　（資料番号 253665-013J)
中巻 A：命令セット・リファレンス A-M　（資料番号 253666-013J)
中巻 B：命令セット・リファレンス N-Z　（資料番号 253667-013J)
下巻：システム・プログラミング・ガイド　（資料番号 253668-013J)

設計する際は、これら 4 巻すべてを参照してください。

2004 年

【輸出規制に関する告知と注意事項】

本資料に掲載されている製品のうち、外国為替および外国為替管理法に定める戦略物資等または役務に該当するものについては、輸出または再輸出する場合、同法に基づく日本政府の輸出許可が必要です。また、米国産品である当社製品は日本からの輸出または再輸出に際し、原則として米国政府の事前許可が必要です。

【資料内容に関する注意事項】

- 本ドキュメントの内容を予告なしに変更することがあります。
- インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。
- インテルでは、インテル製品の内部回路以外の使用にて責任を負いません。また、外部回路の特許についても関知いたしません。
- 本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。

  インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。
- いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複写することは禁じられています。

IA-32 アーキテクチャ・プロセッサ （インテル® Pentium® 4 プロセッサ、インテル® Pentium® III プロセッサなど）、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。

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

図目次

表目次

1

本書について

第 1 章
本書について

1

『IA-32 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、下巻：システム・プログラミング・ガイド』（資料番号 253668-013J）は、IA-32 インテル® プロセッサ全般のアーキテクチャとプログラミング環境を説明している全3巻のうちの1巻である。他の3巻を次に示す。

- 『IA-32 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』（資料番号 253665-013J）
- 『IA-32 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、中巻 A：命令セット・リファレンス A-M』（資料番号 253666-013J）
- 『IA-32 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、中巻 B：命令セット・リファレンス N-Z』（資料番号 253667-013J）

『IA-32 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』の「上巻：基本アーキテクチャ」は、IA-32 プロセッサの基本的なアーキテクチャとプログラミング環境について説明している。「中巻 A、B：命令セット・リファレンス・マニュアル」は、プロセッサの命令セットとオペコードの構造について説明している。上巻と中巻は、既存のオペレーティング・システムやエグゼクティブの下で実行するプログラムを開発しているアプリケーション・プログラマを対象としている。「下巻：システム・プログラミング・ガイド」は、IA-32 プロセッサのオペレーティング・システム・サポート環境について説明している。これには、メモリ管理、保護、タスク管理、割り込み / 例外処理、およびシステム管理モードの説明が含まれる。また、IA-32 プロセッサの互換性に関する情報も掲載している。下巻が対象とするのは、オペレーティング・システムや BIOS の開発者である。

1.1. 本書の対象となる IA-32 プロセッサ

本書には、主に最近の IA-32 プロセッサに関する情報が記載されている。これには、インテル® Pentium® プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサが含まれる。P6 ファミリ・プロセッサとは、P6 ファミリ・マイクロアーキテクチャに基づく IA-32 プロセッサである。P6 ファミリには、インテル® Pentium® Pro プロセッサ、インテル® Pentium® II プロセッサ、インテル® Pentium® III プロセッサが含まれる。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサは、Intel NetBurst® マイクロアーキテクチャに基づいている。

1.2. 『IA-32 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、下巻：システム・プログラミング・ガイド』の概要

本書は以下の内容で構成されている。

第 1 章 ー 本書について。『IA-32 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』の 3 巻それぞれの内容を簡単に説明する。また、これらのマニュアルで使用されている表記法について説明すると共に、インテルのマニュアルやドキュメントのなかでプログラマやハードウェア設計者に関係する関連資料を併記している。

第 2 章 ー システム・アーキテクチャの概要。IA-32 プロセッサの動作モード、オペレーティング・システムやエグゼクティブをサポートするため IA-32 アーキテクチャに用意されているメカニズムについて説明する。これには、システム関連のレジスタとデータ構造の説明や、システム関連の命令の説明が含まれる。また、実アドレスモードと保護モード間の切り替えに必要なステップについても説明する。

第 3 章 ー 保護モードにおけるメモリ管理。セグメンテーションとページングをサポートするデータ構造、レジスタ、命令について説明する。それらを使用して「フラット」（セグメント化されていない）メモリモデルまたはセグメント化されたメモリモデルを使用する方法を説明する。

第 4 章 ー 保護。IA-32 アーキテクチャのページ保護とセグメント保護のサポートについて説明する。この章はまた、特権規則、スタックの切り替え、ポインタの検証、ユーザモードとスーパーバイザ・モードの使用方法を説明する。

第 5 章 ー 割り込みと例外の処理。IA-32 アーキテクチャで定義されている基本的な割り込みメカニズムについて説明し、割り込みや例外と保護との関係を示し、アーキテクチャが各例外タイプをどのように処理するかについて説明する。章の最後には、それぞれの IA-32 の例外に対する参照情報を掲載している。

第 6 章 ー タスク管理。マルチタスキングやタスク間保護を提供するため IA-32 アーキテクチャに用意されているメカニズムについて説明する。

第 7 章 ー マルチプロセッサ管理。共有メモリ、メモリ・オーダリング、ハイパー・スレッディング・テクノロジを使用してマルチプロセッサをサポートする命令やフラグについて説明する。

第 8 章 ー APIC（Advanced Programmable Interrupt Controller）。ローカル APIC のプログラミング・インターフェイスについて説明し、ローカル APIC と I/O APIC 間のインターフェイスの概要を示す。

第9章 ー プロセッサの管理と初期化。リセット初期化後のIA-32プロセッサ、浮動小数点ユニットおよびSIMD浮動小数点ユニットのステートを定義する。この章ではさらに、実アドレスモード動作や保護モード動作向けのセットアップ手順、両モード間の切り替え手順を説明する。

第10章 ー メモリ・キャッシュ制御。キャッシングの一般的な概念と、IA-32アーキテクチャがサポートするキャッシュ・メカニズムについて説明する。この章ではさらに、メモリ・タイプ・レンジ・レジスタ（MTRR）や、これらのレジスタを使用して物理メモリの各メモリタイプをマッピングする方法についても説明する。また、インテル® Pentium® IIIプロセッサ、インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサで新たに導入されたキャッシュ制御命令とメモリ・ストリーミング命令の使用方法についても説明する。

第11章 ー インテル® MMX®テクノロジ・システム・プログラミング。システム・プログラミング・レベルで処理し、かつ配慮しなければならないインテルMMXテクノロジの各局面について説明する。これには、タスク・スイッチング、例外処理、既存システム環境との互換性の説明が含まれる。

第12章 ー SSE、SSE2、SSE3システム・プログラミング。システム・プログラミング・レベルで取り扱い、考慮に入れる必要がある、ストリーミングSIMD拡張命令の機能について説明する。これには、タスク・スイッチング、例外処理、既存のシステム環境との互換性の説明も含む。

第13章 ー システム管理。IA-32アーキテクチャのシステム管理モード（SMM）と温度モニタ機能について説明する。

第14章 ー マシン・チェック・アーキテクチャ。インテル® Pentium® プロセッサでインテル・アーキテクチャに導入されたマシン・チェック・アーキテクチャについて説明する。

第15章 ー デバッグと性能モニタリング。インテル・アーキテクチャに用意されているデバッグレジスタおよびその他のデバッグ・メカニズムについて説明する。この章ではまた、タイムスタンプ・カウンタと性能モニタリング・カウンタについても説明する。

第16章 ー 8086エミュレーション。IA-32アーキテクチャの実アドレスモードと仮想8086モードについて説明する。

第17章 ー 16ビット・コードと32ビット・コードの混在。同一プログラムまたは同一タスク内で、16ビット・コードと32ビット・コードのモジュールを混在させる方法について説明する。

第 18 章 ー IA-32 の互換性。各種の IA-32 プロセッサの互換性について説明する。これには、インテル® 286 プロセッサ、Intel386™ プロセッサ、Intel486™ プロセッサ、インテル Pentium プロセッサ、P6 ファミリ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサが含まれる。32 ビットの IA-32 プロセッサ（Intel386 プロセッサ、Intel486 プロセッサ、インテル Pentium プロセッサ、P6 ファミリ・プロセッサ）間の違いは、『IA-32 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』の全 4 巻を通じてアーキテクチャの特定の機能に関連する箇所で説明する。この章では、すべての IA-32 プロセッサに当てはまる互換情報を提供し、16 ビットの IA-32 プロセッサ（インテル® 8086 プロセッサと 286 プロセッサ）に関する基本的な違いについても説明する。

付録 A ー 性能モニタリング・イベント。性能モニタリング・カウンタでカウント可能なイベントを併記すると共に、これらのイベントを選択する際に使用できるコードを併記する。インテル Pentium プロセッサおよび P6 ファミリ・プロセッサ両方のイベントについて説明する。

付録 B ー モデル固有レジスタ（MSR）。インテル Pentium プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサで使用可能な MSR と、それらの機能の一覧を示す。

付録 C ー P6 ファミリ・プロセッサの MP 初期化。MP システムで P6 ファミリ・プロセッサをブートするための MP プロトコルの使用法の例を示す。

付録 D ー LINT0 および LINT1 入力のプログラミング。特定の割り込みベクタに対する LINT0 および LINT1 ピンのプログラミング方法の例を示す。

付録 E ー マシン・チェック・エラー・コードの解釈。P6 ファミリ・プロセッサ上で発生したマシン・チェック・エラーのエラー・コードの解釈方法の例を示す。

付録 F ー APIC バス・メッセージの形式。P6 ファミリおよびインテル Pentium プロセッサの APIC バス上に送信されるメッセージの形式について説明する。

1.3. 表記法

本書では、データ構造フォーマット、命令のシンボリック表現、16 進数、2 進数に対して特別な表記法を使用している。この表記法を理解しておけば、本書を理解しやすくなる。

1.3.1. ビット・オーダとバイト・オーダ

メモリ内のデータ構造図では、小さい方のアドレスが図の下の方に示され、上に行くほど大きくなる。ビット位置は、右から左に番号が付けられている。セットされたビットの数値は、2をビット位置を表す数で累乗した値に等しくなる。インテル・アーキテクチャ・プロセッサは「リトル・エンディアン」マシンであり、ワードのバイトは最下位バイトから順に番号が付けられている。図1-1.にこれらの規則を示す。

1.3.2. 予約ビットとソフトウェアの互換性

レジスタやメモリのレイアウトの説明で、特定のビットが**「予約済み」**と記されていることがある。ビットが予約済みとして記されている場合は、将来のプロセッサとの互換性を維持するため、これらのビットが将来的に何らかの機能を持つものとみなした上で、ソフトウェア上でこれらのビットを取り扱わなければならない。予約ビットの動作は、未定義としてだけではなく、予測不可能とみなさなければならない。予約ビットを処理する場合は、ソフトウェア上で、次に示すガイドラインに従わなければならない。

- 予約ビットを含むレジスタの値をテストするときは、予約ビットのステートに依存してはならない。テストする前に、予約ビットをマスクアウトする。
- メモリまたはレジスタに格納するときは、予約ビットのステートに依存してはならない。
- 予約ビットに書き込まれた情報が保存されるものとみなしてはならない。
- レジスタにロードするときは、マニュアル上で予約ビットに対して値を指定している場合には、その値を予約ビットにロードしなければならない。マニュアルになければ、同じレジスタから前に読まれた値を再ロードする。

注記

ソフトウェアを、インテル・アーキテクチャ・レジスタの予約ビットのステートに依存させることは絶対に避けること。予約ビットの値に依存すると、プロセッサが予約ビットを処理する方法が決定されていないにもかかわらず、その未決定の方法にソフトウェアが依存することになる。予約ビットの値に依存したプログラムは、将来のプロセッサとの互換性を損なう危険がある。

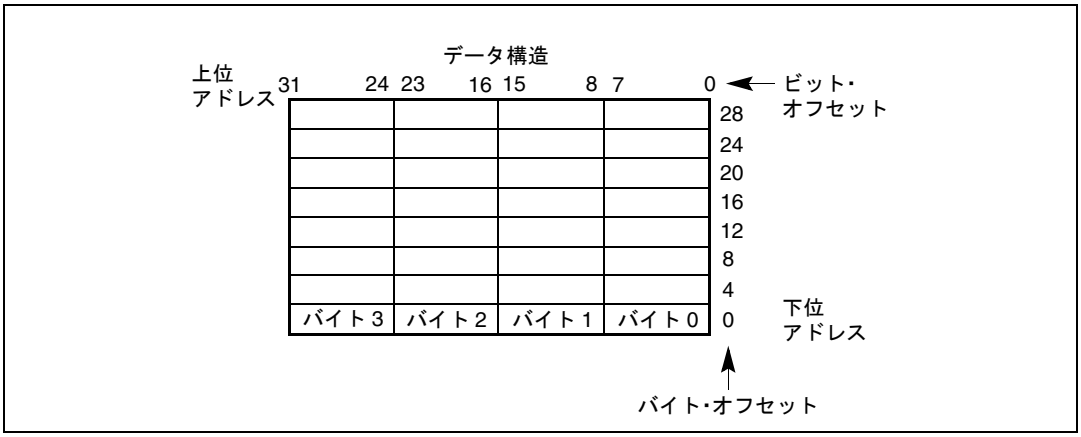


図 1-1. ビット・オーダとバイト・オーダ

1.3.3. 命令のオペランド

命令をシンボルで表現する場合は、インテル・アーキテクチャのアセンブリ言語のサブセットを使用する。このサブセットでは、命令は次の形式をとる。

```
label: mnemonic argument1, argument2, argument3
```

上記の形式において :

- label は識別子で、後にコロンが続く。
- mnemonic は、同じ機能を持つ命令オペコードの予約語である。
- オペランド argument1、argument2、argument3 はオプションである。オペコードに応じて、0 ～ 3 つのオペランドを使用する。オペランドを使用する場合、オペランドはリテラルかデータ項目の識別子のいずれかの形式をとる。オペランド識別子は、レジスタの予約語であるか、またはプログラムの別の箇所（例には示されていないことがある）で宣言されたデータ項目に割り当てられているものとみなされる。

演算命令や論理命令にオペランドが 2 つある場合は、右側のオペランドがソースであり、左側がデスティネーションになる。

例 :

```
LOADREG: MOV EAX, SUBTOTAL
```

この例では、LOADREG はラベル、MOV はオペコードのニーモニック識別子、EAX はデスティネーション・オペランド、SUBTOTAL はソース・オペランドになる。アセンブリ言語によっては、ソースとデスティネーションの順序が逆になることがある。

1.3.4. 16 進と 2 進数

16 をベースとする数（16 進数）は、末尾に文字 H を付けた 16 進数字の文字列で表す（例えば、F82EH）。16 進数字は、0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F のいずれかである。

ベースを 2 とする数（2 進数）は、1 と 0 の文字列で表し、場合によって末尾に文字 B を付ける（例えば、1010B）。「B」を付けるのは、数値のタイプに混乱が生じるような場合に限られる。

1.3.5. セグメント化アドレス指定

インテル・アーキテクチャ・プロセッサでは、バイトによるアドレス指定を採用している。つまり、メモリはバイトの連続として構成されアクセスされる。1 バイトをアクセスするのか複数バイトをアクセスするのかにかかわらず、そのバイトを格納しているメモリへのアクセスには、1 つのバイトアドレスを使用する。アドレス指定が可能なメモリの範囲を、**アドレス空間**と呼ぶ。

プロセッサは、セグメント化アドレス指定もサポートしている。これは、プログラムが**セグメント**と呼ばれる多数の独立したアドレス空間を持つ場合のアドレス指定の一形式である。例えば、プログラムはコード（命令）とスタックを別々のセグメントに保持できる。これにより、コードアドレスは常にコード空間を、スタックアドレスは常にスタック空間を参照することが可能になる。セグメント内のバイトアドレスを指定するには、次の表記法を使用する。

Segment-register:Byte-address

例えば、次のセグメント・アドレスは、DS レジスタがポイントするセグメント内のアドレス FF79H にあるバイトを指す。

```
DS:FF79H
```

また、次のセグメント・アドレスは、コード・セグメントの命令アドレスを指す。CS レジスタはコード・セグメントをポイントし、EIP レジスタは命令のアドレスを格納する。

```
CS:EIP
```

1.3.6. 例外

例外とは、命令がエラーを引き起こした場合に一般的に発生するイベントである。例えば、0 で除算しようとすると例外が発生する。ただし、ブレークポイントのように、エラー以外の条件で発生する例外もある。例外によっては、エラーコードを提示するものもある。エラーコードによって、エラーに関する追加情報が示される。例外とエラーコードを示すために使用する表記例を次に示す。

```
#PF(fault code)
```

この例が示すのは、フォルトのタイプを指すエラーコードが報告される条件でのページフォルト例外である。ある種の条件では、エラーコードが発生する例外でも、正確なコードを報告できない場合がある。このような場合、一般保護例外の例が次に示すように、エラーコードは 0 になる。

```
#GP(0)
```

1.4. 参考文献

インテル・プロセッサに関連する資料の一覧は、以下のリンクに記載されている。

http://www.intel.co.jp/jp/developer/hardwaredesign/processors.htm（日本語）

http://developer.intel.com/design/processor/（英語）

この Web サイトに記載されている資料には、オンラインで表示できるものと、注文できるものがある。入手可能な資料は、まずインテル・プロセッサ別に、次に資料のタイプ（アプリケーション・ノート、データシート、マニュアル、論文、仕様のアップデート）別に記載されている。

以下の資料も参照のこと。

- 『インテル® アーキテクチャ・コンピュータ・ベース・トレーニング』(CD)（資料番号 020005-001）
- 『Intel Pentium® II Processor Specification Update』（資料番号 243337-010）
- 『Intel Pentium® Pro Processor Specification Update』（資料番号 242689-031）
- 『Intel Pentium® Processor Specification Update』（資料番号 242480）
- 『AP-485, Intel Processor Identification and the CPUID Instruction』（資料番号 241618）
- 『AP-485、インテル® プロセッサの識別と CPUID 命令』（資料番号 241618J）
- 『AP-578, Software and Hardware Considerations for FPU Exception Handlers for Intel Architecture Processors』（資料番号 243291）
- 『Pentium® Pro Processor Data Book』（資料番号 242690）

- 『Pentium® Pro BIOS Writer's Guide』
- 『Pentium® Processor Data Book』（資料番号 241428）
- 『82496 Cache Controller and 82491 Cache SRAM Data Book For Use With the Pentium® Processor』（資料番号 241429）
- 『Intel486™ Microprocessor Data Book』（資料番号 240440）
- 『Intel486™ SX CPU/Intel487™ SX Math Coprocessor Data Book』（資料番号 240950）
- 『Intel486™ DX2 Microprocessor Data Book』（資料番号 241245）
- 『Intel486™ Microprocessor Product Brief Book』（資料番号 240459）
- 『Intel386™ Processor Hardware Reference Manual』（資料番号 231732）
- 『Intel386™ Processor System Software Writer's Guide』（資料番号 231499）
- 『Intel386™ High-Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management』（資料番号 231630）
- 『376 Embedded Processor Programmer's Reference Manual』（資料番号 240314）
- 『80387 DX User's Manual Programmer's Reference』（資料番号 231917）
- 『376 High-Performance 32-Bit Embedded Processor』（資料番号 240182）
- 『Intel386™ SX Microprocessor』（資料番号 240187）
- 『Intel® Architecture Optimization Reference Manual』（資料番号 730795）
- 『インテル® アーキテクチャ最適化リファレンス・マニュアル』（資料番号 730795J-001）
- 『Intel® Pentium® 4 and Intel® Xeon™ Processor Optimization Reference Manual』（資料番号 248966）
- 『インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ最適化リファレンス・マニュアル』（資料番号 248966J）

1.5. 参考 URL

- http://developer.intel.com/sites/developer/（英語）
- http://www.intel.co.jp/jp/developer/（日本語）

2

システム・アーキテクチャの概要

第 2 章
システム・アーキテクチャの概要

2

IA-32アーキテクチャ（Intel386™プロセッサ・ファミリから始まる）は、広範なオペレーティング・システムとシステム開発ソフトウェアをサポートしている。このサポートは、IA-32 システム・レベル・アーキテクチャの一部であり、次の動作を支援する機能が含まれる。

- メモリ管理
- ソフトウェア・モジュールの保護
- マルチタスキング
- 例外/割り込み処理
- マルチプロセシング
- キャッシュ管理
- ハードウェア・リソースおよびパワー管理
- デバッグおよび性能モニタリング

本章では、IA-32アーキテクチャの各部分の詳細について説明する。また本章では、システムレベルでのプロセッサのセットアップと制御に使用するシステムレジスタについても説明し、プロセッサのシステムレベル（オペレーティング・システム）の命令について簡単に概要を述べる。

IA-32 システム・レベル・アーキテクチャ機能の多くは、システム・プログラマだけが使用するものである。ただし、アプリケーション・プログラマも、本章および以降の章を読み、アプリケーション・プログラムに対して信頼できる安全な環境を作り出す必要がある。

注記

この概要と本書の以降の章の大部分は、IA-32アーキテクチャのネイティブ、つまり保護モードの動作に焦点を当てている。第9章「プロセッサの管理と初期化」で説明しているように、すべてのIA-32プロセッサは、電源投入またはリセット後に実アドレスモードに入る。ソフトウェアは、その後、実アドレスモードから保護モードへの切り替えを行わなければならない。

2.1. システム・レベル・アーキテクチャの概要

IA-32 システム・レベル・アーキテクチャは、メモリ管理、割り込み/例外処理、タスク管理、複数プロセッサの制御などの基本的なシステムレベルの動作をサポートするために設計された一連のレジスタ、データ構造、命令で構成されている。図 2-1. に、システムレジスタとデータ構造を一般化してまとめたものを示す。

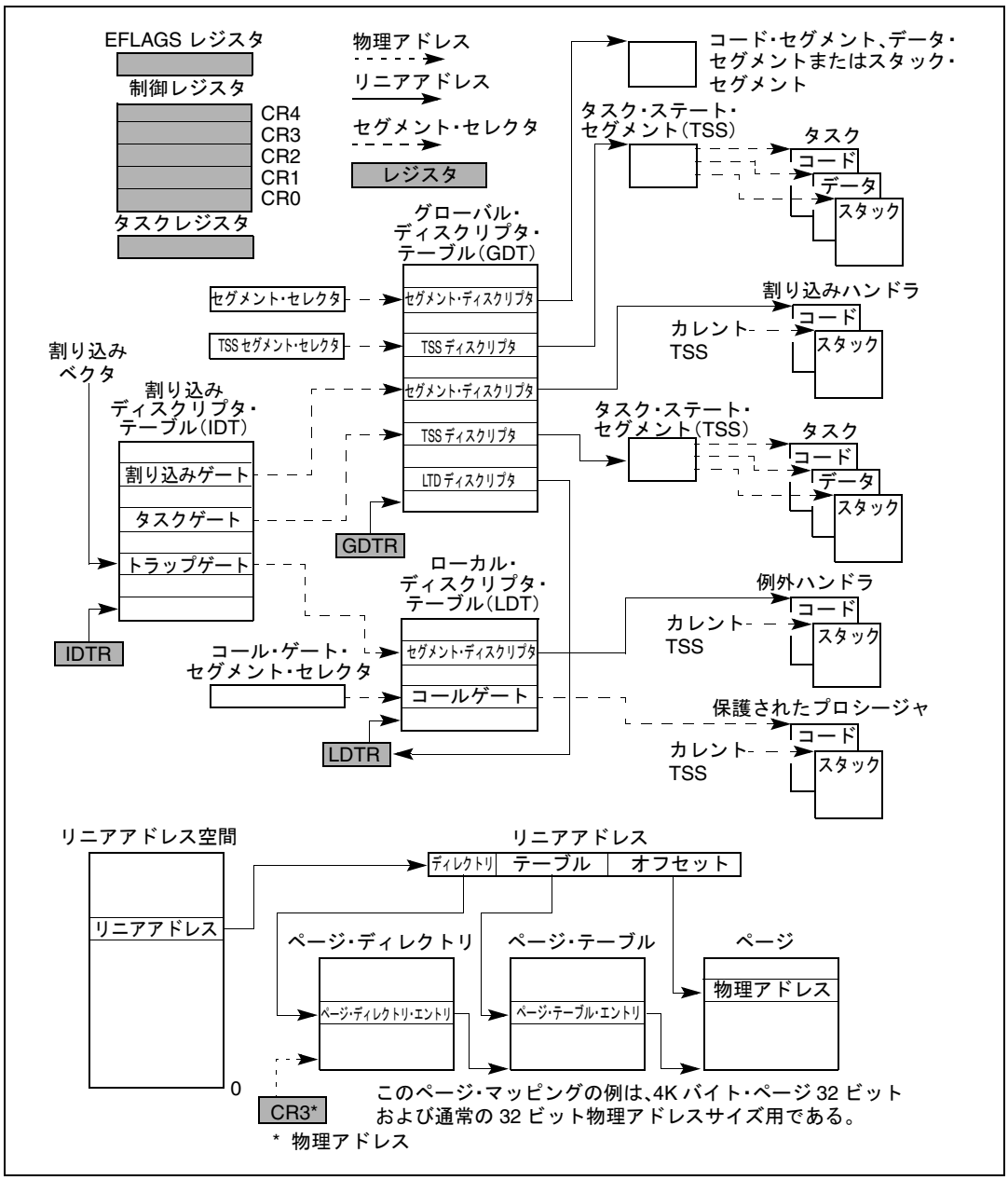


図 2-1. IA-32 システムレベルのレジスタおよびデータ構造

2.1.1. グローバル・ディスクリプタ・テーブルとローカル・ディスクリプタ・テーブル

保護モードで動作しているときは、すべてのメモリアクセスは、図2-1.に示すようにグローバル・ディスクリプタ・テーブル（GDT）または（オプションの）ローカル・ディスクリプタ・テーブル（LDT）を通る。これらのテーブルには、セグメント・ディスクリプタというエントリが入っている。セグメント・ディスクリプタは、セグメントのベースアドレスとアクセス権、タイプ、使用法に関する情報を提供する。各セグメント・ディスクリプタは、それに関連付けられたセグメント・セレクタを持つ。セグメント・セレクタは、（その関連するセグメント・ディスクリプタへの）GDT またはLDT へのインデックス、（セグメント・セレクタがGDT またはLDT のいずれを指しているかを決定する）グローバル/ローカルフラグ、アクセス権情報を提供する。

セグメント内のバイトにアクセスするには、セグメント・セレクタとオフセットを指定しなければならない。セグメント・セレクタによって、（GDT または LDT にある）セグメントのセグメント・ディスクリプタにアクセスする。プロセッサは、セグメント・ディスクリプタからリニアアドレス空間にあるセグメントのベースアドレスを受け取る。オフセットは、ベースアドレスを基準とするバイト位置の相対位置を示す。このメカニズムを使用して、プロセッサが動作している現行特権レベル（CPL）からセグメントにアクセスできれば、GDT または LDT にある任意の有効なコード、データ、またはスタックの各セグメントにアクセスできる。CPL は、その時点で実行しているコード・セグメントの保護レベルとして定義される。

図2-1.で、実線の矢印はリニアアドレスを表し、破線はセグメント・セレクタを表し、点線の矢印は物理アドレスを表している。簡単にするため、セグメント・セレクタの多くをセグメントへのダイレクト・ポインタとして示している。しかし、セグメント・セレクタからその関連するセグメントへの実際のパスは、常に GDT または LDT を通る。

GDT のベースのリニアアドレスはGDT レジスタ（GDTR）に含まれ、LDT のリニアアドレスはLDT レジスタ（LDTR）に含まれている。

2.1.2. システム・セグメント、セグメント・ディスクリプタ、ゲート

システム・アーキテクチャでは、プログラムまたはプロシージャの実行環境を構成するコード、データ、スタックの各セグメント以外に、2 つのシステム・セグメント、すなわち、タスク・ステート・セグメント（TSS）と LDT も定義している。（GDT は、セグメント・セレクタとセグメント・ディスクリプタを使用してアクセスされないのでセグメントと見なされない。）これらのセグメント・タイプのそれぞれに、対応するセグメント・ディスクリプタが定義されている。

IA-32 システム・アーキテクチャでは、ゲートの一連の特別なディスクリプタ（コールゲート、割り込みゲート、トラップゲート、タスクゲート）を定義している。これらのゲートは、アプリケーション・プログラムや大部分のプロシージャとは異なる特権レベルで動作するシステム・プロシージャやシステムハンドラに至る、保護されたゲートウェイとなる。例えば、コールゲートに対してCALL命令を実行すると、現在のコード・セグメントと同じかまたは数値として小さい特権レベル（高い特権）のコード・セグメントにあるプロシージャへのアクセスが実現する。コールゲートを通じてプロシージャにアクセスするには、コール側プロシージャ[1]は、コールゲートのセレクタを指定しなければならない。プロセッサは、その後、コールゲートに対してアクセス権チェックを実行し、CPLをコールゲートとコールゲートによって指されているデスティネーション・コード・セグメントの特権レベルと比較する。デスティネーション・コード・セグメントへのアクセスが許可されると、プロセッサは、デスティネーション・コード・セグメントのセグメント・セレクタとそのコード・セグメントへのオフセットをコールゲートから受け取る。コールが特権レベルの変更を必要とする場合、プロセッサは、その特権レベルに対応するスタックへの切り替えも行う。(新しいスタックのセグメント・セレクタは、現在実行しているタスクのTSSから得られる。)ゲートは、16ビットと32ビットのコード・セグメント間の移行も容易にする。

2.1.3. タスク・ステート・セグメントとタスクゲート

TSS（図 2-1. を参照）は、タスクの実行環境の状態を定義する。これには、汎用レジスタ、セグメント・レジスタ、EFLAGSレジスタ、EIPレジスタ、および3つのスタック・セグメント（特権レベル0、1、2に対してそれぞれ1つのスタック）のセグメント・セレクタとスタックポインタの状態が含まれる。また、タスクに関連するLDTのセグメント・セレクタとページテーブルのベースアドレスも含まれる。

保護モードでのすべてのプログラム実行は、現行タスクというタスクのコンテキスト内で行われる。現行タスクのTSSに対応するセグメント・セレクタは、タスクレジスタにストアされている。タスクを切り替える最も簡単な方法は、タスクのコールまたはタスクへのジャンプを行うことである。この場合には、新しいタスクのTSSに対応するセグメント・セレクタをCALL命令またはJMP命令に指定する。タスクを切り替える際に、プロセッサは、次の操作を実行する。

1. 現行タスクの状態を現在のTSSにストアする。
2. 新しいタスクのセグメント・セレクタをタスクレジスタにロードする。
3. GDTにあるセグメント・ディスクリプタによって新しいTSSにアクセスする。

1. 本書で「プロシージャ」とは、論理ユニットまたはコードのブロック（プログラム、プロシージャ、関数、ルーチンなど）を全般的に指す用語として一般的に使用している。

4. 新しいタスクの状態を新しい TSS から汎用レジスタ、セグメント・レジスタ、LDTR、制御レジスタ CR3（ページテーブルのベースアドレス）、EFLAGS レジスタ、EIP レジスタにロードする。
5. 新しいタスクの実行を開始する。

タスクは、タスクゲートを使用してもアクセスできる。タスクゲートは、コールゲートに類似しているが、コード・セグメントではなく（セグメント・セレクタを通じて）TSS へのアクセスを提供する点が異なる。

2.1.4. 割り込み / 例外処理

外部割り込み、ソフトウェア割り込み、例外は、割り込みディスクリプタ・テーブル（IDT）を通じて処理される（図 2-1. を参照）。IDT には、割り込みハンドラと例外ハンドラへのアクセスを提供するゲート・ディスクリプタが集められている。GDT と同様に、IDT はセグメントではない。IDT のベースのリニアアドレスは、IDT レジスタ（IDTR）に入っている。

IDT にあるゲート・ディスクリプタのタイプは、割り込みゲート、トラップゲート、またはタスクゲートのいずれかである。割り込みハンドラまたは例外ハンドラにアクセスするには、プロセッサは、最初に割り込みベクタ（割り込み番号）を受け取らなければならない。これは、内部ハードウェアである外部割り込みコントローラから、あるいは INT、INTO、INT 3、または BOUND の各命令によってソフトウェアから受け取る。割り込みベクタは、IDT 内のゲート・ディスクリプタを指示するインデックスの働きをする。選択されたゲート・ディスクリプタが割り込みゲートまたはトラップゲートである場合は、コールゲートを通じたプロシージャのコールに非常によく似た方法で、関連するハンドラ・プロシージャにアクセスする。ディスクリプタがタスクゲートである場合は、タスクスイッチによってハンドラにアクセスする。

2.1.5. メモリ管理

システム・アーキテクチャは、メモリの直接的な物理アドレス指定か、または（ページングによる）仮想メモリをサポートしている。物理アドレス指定を使用すると、リニアアドレスは物理アドレスとして扱われる。ページングを使用すると、コード・セグメント、データ・セグメント、スタック・セグメント、システム・セグメントのすべて、および GDT と IDT をページングすることができ、直前にアクセスされたページだけが物理メモリに保持される。

物理メモリ内のページ（または IA-32 アーキテクチャではページフレームということもある）の位置は、2 つのタイプのシステムデータ構造（ページ・ディレクトリと一組のページテーブル）に含まれており、その両方とも物理メモリ内に常駐する（図 2-1. を参照）。ページ・ディレクトリのエントリには、ページテーブルのベースの物理ア

ドレス、アクセス権、メモリ管理情報が入っている。ページテーブルのエントリには、ページフレームの物理アドレス、アクセス権、およびメモリ管理情報が入っている。ページ・ディレクトリのベース物理アドレスは、制御レジスタ CR3 に入っている。

このページング・メカニズムを使用するには、ページ・ディレクトリ、ページテーブル、およびページフレームへの別個のオフセットを与える 3 つの部分にリニアアドレスを分割する。

システムは、1 つだけまたは複数のページ・ディレクトリを持つことができる。例えば、各タスクは、専用のページ・ディレクトリを持つことができる。

2.1.6. システムレジスタ

プロセッサの初期化とシステム動作の制御を支援するため、システム・アーキテクチャでは、EFLAGS レジスタと複数のシステムレジスタにシステムフラグを備えている。

- EFLAGS レジスタのシステムフラグと IOPL フィールドは、タスクとモードの切り替え、割り込み処理、命令トレース、アクセス権を制御する。これらのフラグの説明については、2.3.節「EFLAGS レジスタのシステムフラグとフィールド」を参照。
- 制御レジスタ（CR0、CR2、CR3、CR4）には、システムレベルの動作を制御するためのさまざまなフラグとデータ・フィールドが入っている。これらのレジスタの他のフラグは、オペレーティング・システムがプロセッサ固有の機能をサポートしているかどうかを示すために使用される。これらのフラグの説明については、2.5. 節「制御レジスタ」を参照。
- デバッグレジスタ（図 2-1. には示していない）によって、プログラムとシステム・ソフトウェアのデバッグで使用するブレークポイントを設定できる。これらのレジスタの説明については、第 15 章「デバッグと性能モニタリング」を参照。
- GDTR、LDTR、IDTR の各レジスタには、それぞれのテーブルのリニアアドレスとサイズ（リミット）が入っている。これらのレジスタの説明については、2.4. 節「メモリ管理レジスタ」を参照。
- タスクレジスタには、現行タスクの TSS のリニアアドレスとサイズが入っている。このレジスタの説明については、2.4. 節「メモリ管理レジスタ」を参照。
- モデル固有レジスタ（図 2-1. には示していない）。

モデル固有レジスタ（MSR）とは、オペレーティング・システムまたはエグゼクティブ・プロシージャ（特権レベル 0 で実行されるコード）で主に使用できるレジスタのグループである。これらのレジスタは、デバッグ拡張、性能モニタリング・カウンタ、マシン・チェック・アーキテクチャ、メモリ・タイプ・レンジ（MTRR）などのアイテムを制御する。

これらのレジスタの数と機能は、IA-32 プロセッサ・ファミリのメンバー間で変わることがある。MSR の詳細については、9.4. 節「モデル固有レジスタ（MSR）」を、MSR のすべてのリストについては、付録 B「モデル固有レジスタ（MSR）」をそれぞれ参照。

大部分のシステムでは、アプリケーション・プログラムによって（EFLAGS レジスタ以外の）システムレジスタへのアクセスを制限している。ただし、すべてのプログラムとプロシージャを最高の特権レベル（特権レベル 0）で実行するシステムを設計することは可能であり、その場合には、アプリケーション・プログラムでもシステムレジスタを修正できる。

2.1.7. その他のシステムリソース

前の項で説明したシステムレジスタとデータ構造に加えて、システム・アーキテクチャでは、次の追加リソースを備えている。

- オペレーティング・システム命令（2.6. 節「システム命令のまとめ」を参照）
- 性能モニタリング・カウンタ（図 2-1. に示していない）
- 内部キャッシュと内部バッファ（図 2-1. に示していない）

性能モニタリング・カウンタは、デコードされた命令数、受け取った割り込み数、キャッシュ・ロード回数などのプロセッサ・イベントをカウントするようにプログラムできるイベントカウンタである。これらのカウンタの詳細については、15.8. 節「性能モニタリングの概要」を参照。

プロセッサは、複数の内部キャッシュと内部バッファを備えている。キャッシュは、データと命令両方のストアに使用される。バッファは、システム・セグメントとアプリケーション・セグメントへのデコードされたアドレスや、実行を待機中の書き込み操作などのストアに使用される。プロセッサのキャッシュとバッファの詳細については、第 10 章「メモリ・キャッシュ制御」を参照。

2.2. 動作モード

IA-32 アーキテクチャは、3 つの動作モードと 1 つの疑似動作モードをサポートしている。

- **保護モード。**これは、プロセッサの本来の動作モードである。このモードでは、すべての命令とアーキテクチャ機能を使用することができ、最高の性能と能力を発揮する。これは、新規のすべてのアプリケーションとオペレーティング・システムに推奨するモードである。

- **実アドレスモード。**この動作モードは、インテル® 8086 プロセッサのプログラミング環境に（保護モードまたはシステム管理モードに切り替える機能など）いくつかの拡張機能を加えて提供する。
- **システム管理モード（SMM）。**システム管理モード（SMM）は、Intel386™ SL プロセッサをはじめ、すべての IA-32 プロセッサにある標準アーキテクチャ機能である。このモードは、オペレーティング・システムまたはエグゼクティブにパワー管理機能と OEM 差別化機能を採用するための透過的なメカニズムを提供する。SMM に入るには、外部システム割り込みピン（SMI#）をアクティブにする。こうすると、システム管理割り込み（SMI）が生成される。SMM では、プロセッサは、現在実行しているプログラムまたはタスクのコンテキストをセーブしながら、個別のアドレス空間に切り替わる。その場合に、SMM 固有コードを透過的に実行できる。SMM から戻ると、プロセッサは、SMI 生成前の状態に戻される。
- **仮想 8086 モード。**保護モードでは、プロセッサは、**仮想 8086 モード**という疑似動作モードをサポートしている。このモードによって、プロセッサは、8086 ソフトウェアを保護されたマルチタスキング環境で実行できる。

図 2-2. に、プロセッサがこれらの動作モード間をどのように移行するかを示す。

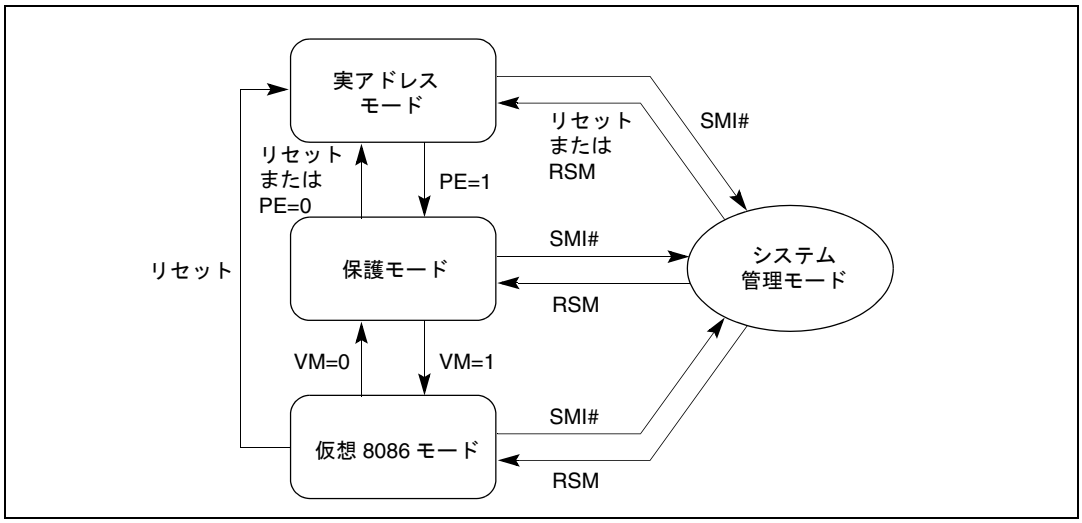


図 2-2. プロセッサの動作モード間の移行

プロセッサは、電源投入またはリセット後は実アドレスモードになっている。その後、プロセッサが実アドレスモードまたは保護モードのどちらで動作するかは、制御レジスタ CR0 の PE フラグによって制御される（2.5. 節「制御レジスタ」を参照）。実アドレスモードと保護モードとの切り替えの詳細については、9.9. 節「モード切り替え」を参照。

EFLAGS レジスタの VM フラグは、プロセッサが保護モードまたは仮想 8086 モードのどちらで動作するかを決定する。保護モードと仮想 8086 モードとの間の切り替えは、一般的にはタスクスイッチあるいは割り込みハンドラまたは例外ハンドラからの戻りの一部として行われる（16.2.5. 項「仮想 8086 モードへの移行」を参照）。

プロセッサは、実アドレスモード、保護モード、または仮想 8086 モードにある間に SMI を受け取ると SMM に切り替わる。プロセッサは、RSM 命令を実行すると、SMI が発生した時点のモードに常に戻る。

2.3. EFLAGS レジスタのシステムフラグとフィールド

EFLAGS レジスタ内のシステムフラグと IOPL フィールドは、I/O、マスク可能ハードウェア割り込み、デバッグ、タスク・スイッチング、仮想 8086 モードを制御する（図 2-3. を参照）。特権を持つコード（通常はオペレーティング・システムまたはエグゼクティブ・コード）だけがこれらのビットの修正を許可されなければならない。

システムフラグと IOPL の機能は、次のとおりである。

TF　**トラップ（ビット 8）。**このフラグがセットされると、デバッグのためのシングル・ステップ・モードがイネーブルになる。クリアされると、シングル・ステップ・モードがディスエーブルになる。シングル・ステップ・モードでは、プロセッサは、各命令の後にデバッグ例外を生成するので、各命令後にプログラムの実行状態を調べることができる。アプリケーション・プログラムが POPF、POPFD、または IRET 命令を使用して TF フラグをセットすると、POPF、POPFD、または IRET 命令の後に続く命令後に、デバッグ例外が発生する。

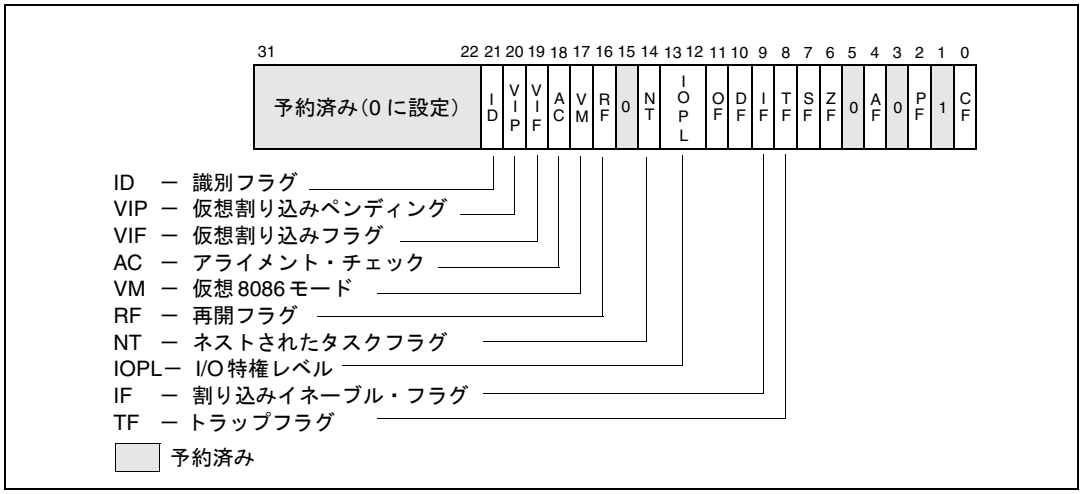


図 2-3. EFLAGS レジスタのシステムフラグ

IF　**割り込みイネーブル（ビット 9）。**マスク可能ハードウェア割り込み要求（5.3.2. 項「マスク可能ハードウェア割り込み」を参照）に対するプロセッサの応答を制御する。このフラグがセットされていると、マスク可能ハードウェア割り込みに応答する。クリアされていると、マスク可能ハードウェア割り込みを禁止する。IF フラグは、例外またはマスク不可能な割り込み（NMI 割り込み）の生成に影響を与えない。CPL、IOPL、および制御レジスタ CR4 の VME フラグの状態によって、IF フラグを CLI、STI、POPF、POPFD、IRET 命令によって修正できるかどうかが決定される。

IOPL　**I/O 特権レベル・フィールド（ビット 12 および 13）。**現在実行しているプログラムまたはタスクの I/O 特権レベル（IOPL）を表す。現在実行しているプログラムまたはタスクの CPL は、I/O アドレス空間にアクセスするには IOPL に等しいかそれより下位でなければならない。このフィールドは、CPL 0 で動作しているときに POPF および IRET 命令によってだけ修正することができる。IOPL と I/O 操作との関係の詳細については、『IA-32 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』の第 13 章「入出力」を参照。

IOPL は、仮想モード拡張が有効である（制御レジスタ CR4 の VME フラグがセットされている）ときに仮想 8086 モードでの IF フラグの修正と割り込みの処理を制御するメカニズムの 1 つでもある。

NT　**ネストされたタスク（ビット 14）。**割り込まれたタスクとコールされたタスクの連鎖を制御する。プロセッサは、CALL 命令、割り込み、または例外によって開始されたタスクへのコール時にこのフラグをセットする。プロセッサは、IRET 命令によって開始されたタスクからの戻り時にこのフラグを調べて修正する。フラグは、POPF/POPFD 命令で明示的にセットまたはクリアできるが、このフラグの状態を変更すると、アプリケーション・プログラムに予測不可能な例外が発生する場合もある。

ネストされたタスクの詳細については、6.4. 節「タスクのリンク」を参照。

RF　**再開（ビット 16）。**命令ブレークポイント条件に対するプロセッサの応答を制御する。このフラグがセットされていると、命令ブレークポイントに対して一時的にデバッグ例外（#DE）が生成されないようになる。ただし、他の例外条件では、例外を生成できる。クリアされていると、命令ブレークポイントはデバッグ例外を生成する。

RF フラグの第一の機能は、命令ブレークポイント条件によって生成されたデバッグ例外の後に続く命令を再開させることである。この場合に、デバッグ・ソフトウェアは、割り込まれたプログラムに IRETD 命令で戻る直前にスタック上の EFLAGS イメージのこのフラグをセットして、命令ブレークポイントがもう 1 つのデバッグ例外を発生させないようにしなければならない。プロセッサは、その後、戻った命令が正常に実行されるとこのフラグを自動的にクリアして、命令ブレークポイント・フォルトを再びイネーブルにする。

このフラグの使用法の詳細については、15.3.1.1. 項「命令ブレークポイント例外条件」を参照。

VM **仮想 8086 モード（ビット 17）。**このフラグがセットされると、仮想 8086 モードがイネーブルになる。クリアされると、保護モードに戻る。仮想 8086 モードに切り替えるためのこのフラグの使用法の詳細については、16.2.1. 項「仮想 8086 モードの有効化」を参照。

AC **アライメント・チェック（ビット 18）。**このフラグと制御レジスタ CR0 の AM フラグがセットされると、メモリ参照のアライメント・チェックがイネーブルになる。AC フラグまたは AM フラグ、あるいはその両方がクリアされると、アライメント・チェックがディスエーブルになる。奇数バイトアドレスにあるワードや 4 の整数倍でないアドレスにあるダブルワードなどアライメントの合っていないオペランドが参照されると、アライメント・チェック例外が発生する。アライメント・チェック例外は、ユーザモード（特権レベル 3）だけで発生する。セグメント・ディスクリプタのロードなど特権レベル 0 をデフォルトとするメモリ参照は、ユーザモードで実行された命令によって発生するときでもこの例外を生成しない。

アライメント・チェック例外を使用して、データのアライメントをチェックすることができる。これは、すべてのデータのアライメントを必要とするプロセッサとのデータ交換の場合に役立つ。アライメント・チェック例外は、インタプリタが使用して、ポインタのアライメントが合っていない場合にそのポインタを特殊としてフラグ表示することもできる。こうすると、各ポインタをチェックするオーバーヘッドが除去され、特殊ポインタが使用するときだけにそれを処理するだけで済む。

VIF **仮想割り込み（ビット 19）。**IF フラグの仮想イメージを含む。このフラグは、VIP フラグと連係して使用される。プロセッサは、制御レジスタ CR4 の VME フラグまたは PVI フラグがセットされており、IOPL が 3 未満であるときだけに VIF フラグを認識する。（VME フラグは仮想 8086 モード拡張をイネーブルにし、PVI フラグは保護モード仮想割り込みをイネーブルにする。）

このフラグの使用法の詳細については、16.3.3.5. 項「方法 6: ソフトウェア割り込みの処理」と 16.4. 節「保護モード仮想割り込み」を参照。

VIP **仮想割り込みペンディング（ビット 20）。**ソフトウェアがこのフラグをセットした場合は、割り込みがペンディングであることを表す。クリアした場合は、ペンディングの割り込みがないことを表す。このフラグは、VIF フラグと連係して使用される。プロセッサは、このフラグを読み取るが、変更することはない。プロセッサは、制御レジスタ CR4 の VME フラグまたは PVI フラグがセットされており、IOPL が 3 未満であるときだけに VIP フラグを認識する（VME フラグは仮想 8086 モード拡張をイネーブルにし、PVI フラグは保護モード仮想割り込みをイネーブルにする）。

このフラグの使用法の詳細については、16.3.3.5. 項「方法 6: ソフトウェア割り込みの処理」と 16.4. 節「保護モード仮想割り込み」を参照。

ID　**識別（ビット 21）。**このフラグをセットまたはクリアできるプログラムまたはプロシージャの機能は、CPUID 命令をサポートする。

2.4. メモリ管理レジスタ

プロセッサは、セグメント化されたメモリ管理を制御するデータ構造の位置を指定する 4 つのメモリ管理レジスタ（GDTR、LDTR、IDTR、TR）を備えている（図 2-4. を参照）。これらのレジスタをロードおよびストアするための特殊な命令が用意されている。

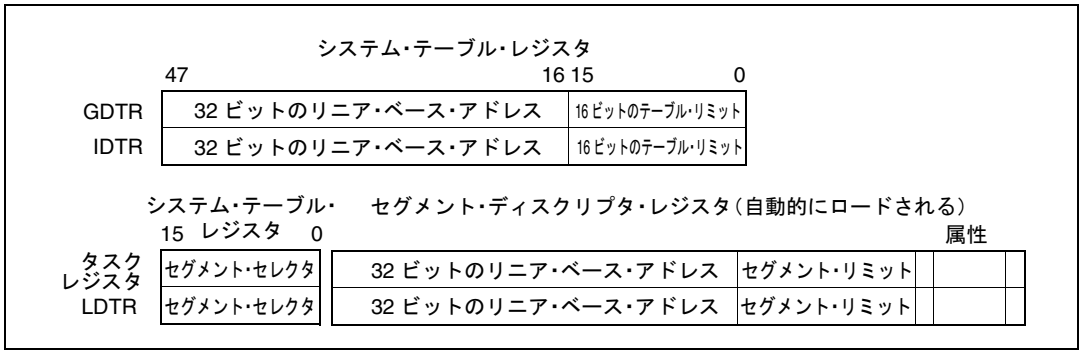


図 2-4. メモリ管理レジスタ

2.4.1. グローバル・ディスクリプタ・テーブル・レジスタ（GDTR）

GDTR レジスタには、GDT の 32 ビット・ベース・アドレスと 16 ビット・テーブル・リミットが入っている。ベースアドレスは GDT のバイト 0 のリニアアドレスを指定し、テーブルリミットはテーブルのバイト数を指定する。LGDT 命令と SGDT 命令は、それぞれ GDTR レジスタをロードおよびストアする。プロセッサの電源投入時またはリセット時に、ベースアドレスはデフォルト値 0 に設定され、リミットは FFFFH に設定される。新しいベースアドレスは、保護モード動作のためのプロセッサ初期化プロセスの一部として GDTR にロードされなければならない。ベース・アドレス・フィールドとリミット・フィールドの詳細については、3.5.1. 項「セグメント・ディスクリプタ・テーブル」を参照。

2.4.2. ローカル・ディスクリプタ・テーブル・レジスタ（LDTR）

LDTR レジスタには、LDT の 16 ビットのセグメント・セレクタ、32 ビットのベースアドレス、16 ビットのセグメント・リミット、ディスクリプタ属性が入っている。ベースアドレスは LDT セグメントのバイト 0 のリニアアドレスを指定し、セグメント・リミットはセグメントのバイト数を指定する。ベース・アドレス・フィールドとリミット・フィールドの詳細については、3.5.1. 項「セグメント・ディスクリプタ・テーブル」を参照。

LLDT 命令と SLDT 命令は、それぞれ LDTR レジスタのセグメント・セレクタ部分をロードおよびストアする。LDT を含むセグメントは、GDT にセグメント・ディスクリプタがなければならない。LLDT 命令がセグメント・セレクタを LDTR にロードすると、LDT ディスクリプタからのベースアドレス、リミット、ディスクリプタ属性が LDTR に自動的にロードされる。

タスクスイッチが発生すると、新しいタスクの LDT に対応するセグメント・セレクタとディスクリプタが LDTR に自動的にロードされる。LDTR の内容は、新しい LDT 情報がレジスタに書き込まれる前に自動的にはセーブされない。

プロセッサの電源投入時またはリセット時に、セグメント・セレクタとベースアドレスはデフォルト値 0 に設定され、リミットは FFFFH に設定される。

2.4.3. 割り込みディスクリプタ・テーブル・レジスタ（IDTR）

IDTR レジスタには、IDT の 32 ビットのベースアドレスと 16 ビットのテーブルリミットが入っている。ベースアドレスは IDT のバイト 0 のリニアアドレスを指定し、テーブルリミットはテーブルのバイト数を指定する。LIDT 命令と SIDT 命令は、それぞれ IDTR レジスタをロードおよびストアする。プロセッサの電源投入時またはリセット時に、ベースアドレスはデフォルト値 0 に設定され、リミットは FFFFH に設定される。レジスタのベースアドレスとリミットは、その後、プロセッサ初期化プロセスの一部として変更できる。ベース・アドレス・フィールドとリミット・フィールドの詳細については、5.10. 節「割り込みディスクリプタ・テーブル（IDT: Interrupt Descriptor Table）」を参照。

2.4.4. タスクレジスタ（TR）

タスクレジスタには、現行タスクの TSS の 16 ビットのセグメント・セレクタ、32 ビットのベースアドレス、16 ビットのセグメント・リミット、ディスクリプタ属性が入っている。このレジスタは、GDT にある TSS ディスクリプタを参照する。ベースアドレスは TSS のバイト 0 のリニアアドレスを指定し、セグメント・リミットは TSS のバイト数を指定する。（タスクレジスタの詳細については、6.2.3. 項「タスクレジスタ」を参照。）

LTR 命令と STR 命令は、それぞれタスクレジスタのセグメント・セレクタ部分をロードおよびストアする。LTR 命令がセグメント・セレクタをタスクレジスタにロードすると、TSS ディスクリプタからのベースアドレス、リミット、ディスクリプタ属性がタスクレジスタに自動的にロードされる。プロセッサの電源投入時またはリセット時に、ベースアドレスはデフォルト値 0 に設定され、リミットは FFFFH に設定される。

タスクスイッチが発生すると、新しいタスクの TSS のセグメント・セレクタとディスクリプタがタスクレジスタに自動的にロードされる。タスクレジスタの内容は、新しい TSS 情報がレジスタに書き込まれる前に自動的にはセーブされない。

2.5. 制御レジスタ

制御レジスタ（CR0、CR1、CR2、CR3、CR4、図 2-5. を参照）は、下図に示すようにプロセッサの動作モードと現在実行しているタスクの特性を決定する。

- CR0 ― プロセッサの動作モードと状態を制御するシステム制御フラグが入っている。
- CR1 ― 予約済み
- CR2 ― ページ・フォルト・リニア・アドレス（ページフォルトを発生させたリニアアドレス）が入っている。
- CR3 ― ページ・ディレクトリのベースの物理アドレスと 2 つのフラグ（PCD と PWT）が入っている。このレジスタは、ページ・ディレクトリ・ベース・レジスタ（PDBR）ともいう。ページ・ディレクトリのベースアドレスの上位 20 ビットだけが指定され、アドレスの下位 12 ビットは 0 であると見なされる。したがって、ページ・ディレクトリはページ（4K バイト）境界にアライメントを合わせなければならない。PCD フラグと PWT フラグは、プロセッサの内部データ・キャッシュにあるページ・ディレクトリのキャッシングを制御する（これらのフラグはページ・ディレクトリ情報の TLB キャッシングを制御しない）。

  物理アドレス拡張を使用すると、レジスタ CR3 には、ページ・ディレクトリ・ポインタ・テーブルのベースアドレスが入る（3.8. 節「PAE ページング・メカニズムを使用した 36 ビット 物理アドレス指定」を参照）。
- CR4 ― このレジスタ内の一連のフラグを使用して、アーキテクチャ上の各種の拡張機能をイネーブルにできる。また、オペレーティング・システムおよびプロセッサ固有の機能エグゼクティブ・サポートを示す。制御レジスタは、MOV 命令による制御レジスタへの移動または制御レジスタからの移動によって読み取りまたはロードを行うことができる。保護モードでは、MOV 命令によって、制御レジスタの読み取りまたはロード（特権レベル 0 だけで）を行うことができる。この制約は、（特権レベル 1、2、または 3 で実行している）アプリケーション・プログラム

またはオペレーティング・システム・プロシージャは制御レジスタに読み取りまたはロードできないことを意味している。

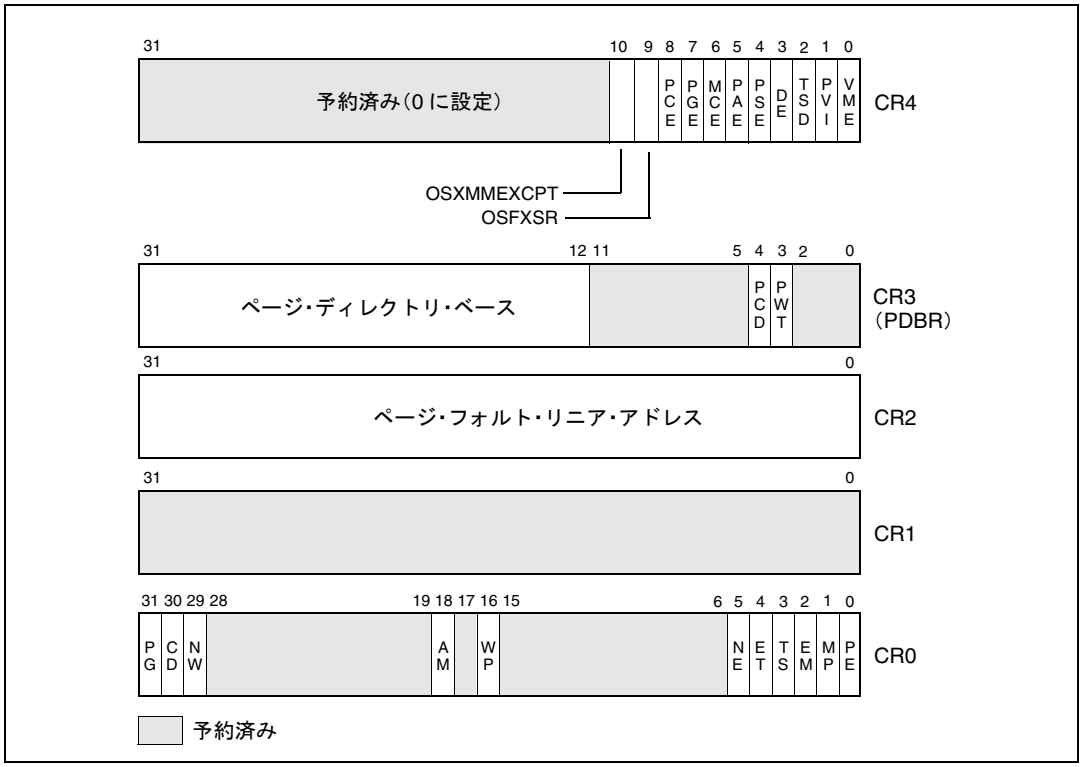


図 2-5. 制御レジスタ

制御レジスタをロードするときは、予約ビットは常に以前に読み取られた値にセットしなければならない。制御レジスタにあるフラグの機能は、次のとおりである。

PG　**ページング（CR0 のビット 31）。**このフラグがセットされると、ページングがイネーブルになる。クリアされると、ページングがディスエーブルになる。ページングがディスエーブルになっていると、すべてのリニアアドレスは物理アドレスとして扱われる。PE フラグ（レジスタ CR0 のビット 0）がセットされていないと、PG フラグは効力を持たない。PE フラグがクリアされているときに PG フラグをセットすると、一般保護例外（#GP）が発生する。プロセッサのページング・メカニズムの詳細については、3.6. 節「ページング（仮想メモリ）の概要」を参照。

CD　**キャッシュ・ディスエーブル（CR0 のビット 30）。**CD フラグと NW フラグがクリアされると、プロセッサの内部（および外部）キャッシュにある物理メモリ全体のメモリ位置のキャッシングがイネーブルになる。CD フラグがセットされると、キャッシングは、表 10-5. で説明しているように制約される。プロセッサ

がそのキャッシュをアクセスしたり更新したりしないようにするには、CD フラグをセットしなければならず、キャッシュ・ヒットが起こらないようにキャッシュを無効化しなければならない（10.5.3. 項「キャッシングの防止」を参照）。選択したページまたはメモリ領域のキャッシングに課すことのできる追加の制約の詳細については、10.5. 節「キャッシュの制御」を参照。

NW　**ノット・ライトスルー（CR0 のビット 29）。**NW フラグと CD フラグがクリアされると、キャッシュをヒットする書き込みに対してライトバック（インテル ® Pentium® 4 プロセッサ、インテル ® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル ® Pentium® プロセッサの場合）またはライトスルー（Intel486™ プロセッサの場合）がイネーブルになり、無効化サイクルがイネーブルになる。CD フラグと NW フラグのその他の設定におけるキャッシングへの NW フラグの影響の詳細については、表 10-5. を参照。

AM　**アライメント・マスク（CR0 のビット 18）。**このフラグがセットされると、自動アライメント・チェックがイネーブルになる。クリアされると、アライメント・チェックがディスエーブルになる。アライメント・チェックは、AM フラグがセットされ、EFLAGS レジスタの AC フラグがセットされ、CPL が 3 であり、プロセッサが保護モードまたは仮想 8086 モードで動作しているときだけに実行される。

WP　**書き込み保護（CR0 のビット 16）。**このフラグがセットされると、スーパーバイザ・レベルのプロシージャがユーザレベル読み取り専用ページに書き込むことを禁止する。クリアされると、スーパーバイザ・レベルのプロシージャがユーザレベル読み取り専用ページに書き込むことを許可する。このフラグは、UNIX* などのオペレーティング・システムによって使用され、コピー・オン・ライト方式による新しいプロセスを生成する場合に（フォークする）設計を容易にする。

NE　**数値演算エラー（CR0 のビット 5）。**このフラグがセットされると、x87 FPU エラーをレポートするための標準メカニズムがイネーブルになる。クリアされると、PC スタイルの x87 FPU エラー・レポート・メカニズムがイネーブルになる。NE フラグがクリアされており、IGNNE# 入力がアサートされていると、x87 FPU エラーは無視される。NE フラグがクリアされており、IGNNE# 入力がディアサートされていると、マスクされていない x87 FPU エラーによって、プロセッサは FERR# ピンをアサートして外部割り込みを生成し、命令実行を直ちに停止してから次に待機している浮動小数点命令または WAIT/FWAIT 命令を実行する。FERR# ピンは、外部割り込みコントローラへの入力をドライブするためのピンである（FERR# ピンは、インテル ® 287 プロセッサおよびインテル ® 387 DX マス・コプロセッサをエミュレートする）。NE フラグ、IGNNE# ピン、FERR# ピンは、PC スタイルのエラーレポートを参照するために外部ロジックと一緒に使用される。（x87 FPU エラーレポート、プロセッサに依存する FERR# ピンをアサートする時期の詳細については、『IA-32 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』の第 8 章にある「ソフトウェア例外ハンドラ」と付録 D を参照。）

ET　**拡張タイプ（CR0 のビット 4）。**インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサで予約されている。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、このフラグは 1 に固定されている。Intel386™ プロセッサと Intel486 プロセッサでは、このフラグがセットされていると 387 DX マス・コプロセッサ命令のサポートを示す。

TS　**タスクスイッチ（CR0 のビット 3）。**このフラグによって、x87 FPU、MMX® 命令、SSE 命令、SSE2 命令、または SSE3 命令が新しいタスクによって実際に実行されるまで、タスクスイッチ時の x87 FPU、MMX 命令、SSE、SSE2、SSE3 コンテキストのセーブを遅延できる。プロセッサは、タスクスイッチのたびにこのフラグをセットし、x87 FPU、MMX 命令、SSE 命令、SSE2 命令、SSE3 命令を実行するときにこのフラグをテストする。

- TS フラグがセットされ、EM フラグ（CR0 のビット 2）がクリアされていると、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH 命令を除く x87 FPU、MMX 命令、SSE 命令、SSE2 命令、または SSE3 命令の実行前に、デバイス使用不可能例外（#NM）が発生する。（WAIT/FWAIT 命令の特殊な場合については、下記の段落を参照。）
- TS フラグがセットされ、MP フラグ（CR0 のビット 1）と EM フラグがクリアされていると、x87 FPU WAIT/FWAIT 命令の実行前に、#NM 例外は発生しない。
- EM フラグがセットされている場合、TS フラグの設定は x87 FPU、MMX 命令、SSE 命令、SSE2 命令、SSE3 命令の実行に影響を及ぼさない。

表 2-1. に、TS フラグ、EM フラグ、MP フラグの設定に基づいて、プロセッサが x87 FPU 命令を見つけたときに取られる措置を示す。表 11-1. および 12-1. に、プロセッサが MMX 命令、SSE 命令、SSE2 命令、SSE3 命令をそれぞれ見つけたときに取られる措置を示す。

プロセッサは、タスクスイッチ時に x87 FPU、XMM、MXCSR レジスタのコンテキストを自動的にセーブしない。代わりに、TS フラグをセットし、これによって、プロセッサは、新しいタスクの命令ストリームに x87 FPU、MMX 命令、SSE 命令、SSE2 命令、または SSE3 命令を見付けると #NM 例外を発生させる（上記の命令の例外）。

その場合には、#NM 例外のフォルトハンドラを使用して、（CLTS 命令で）TS フラグをクリアし、x87 FPU、XMM、MXCSR レジスタのコンテキストをセーブできる。タスクに x87 FPU、MMX 命令、SSE 命令、SSE2 命令、または SSE3 命令がなければ、x87 FPU、MMX 命令、SSE、SSE2、SSE3 コンテキストはセーブされない。

表 2-1. EM、MP、TS のさまざまな組み合わせに対して x87 FPU 命令によって取られる措置

CR0 フラグ			x87 FPU 命令タイプ	
EM	MP	TS	浮動小数点	WAIT/FWAIT
0	0	0	実行	実行
0	0	1	#NM 例外	実行
0	1	0	実行	実行
0	1	1	#NM 例外	#NM 例外
1	0	0	#NM 例外	実行
1	0	1	#NM 例外	実行
1	1	0	#NM 例外	実行
1	1	1	#NM 例外	#NM 例外

EM　**エミュレーション（CR0 のビット 2）。**このフラグがセットされていると、プロセッサは内部または外部の x87 FPU を持っていないことを表す。クリアされていると、x87 FPU が存在することを表す。このフラグはまた、MMX 命令、SSE 命令、SSE2 命令、SSE3 命令の実行に影響を及ぼす。

EM フラグがセットされていると、x87 FPU 命令の実行によってデバイス使用不可能例外（#NM）が発生する。プロセッサが内部 x87 FPU を持っていないかまたは外部のマス・コプロセッサに接続されていない場合は、このフラグをセットしなければならない。このフラグをセットすると、すべての浮動小数点命令はソフトウェア・エミュレーションによって強制的に処理されることになる。表 9-2. に、システムで IA-32 プロセッサと x87 FPU かマス・コプロセッサのどちらが存在するかに応じたこのフラグの推奨の設定を示す。表 2-1. に、EM フラグ、MP フラグ、TS フラグの相互作用を示す。

また、EM フラグがセットされていると、MMX 命令の実行によって無効なオペコード例外（#UD）が発生する（表 11-1. を参照）。そのため、IA-32 プロセッサが MMX テクノロジを組み込んでいる場合は、EM フラグを 0 に設定して MMX 命令の実行をイネーブルにしなければならない。

同じように、EM フラグがセットされているときにほとんどの SSE、SSE2、SSE3 を実行すると、無効オペコード例外（#UD）が発生する（表 12-1. を参照）。したがって、IA-32 プロセッサが SSE、SSE2、SSE3 を搭載している場合は、EM フラグを 0 に設定して、これらの命令の実行をイネーブルにしなければならない。ただし、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH の SSE、SSE2、SSE3 命令は、EM フラグの影響を受けない。

MP　**モニタ・コプロセッサ（CR0 のビット 1）。**WAIT（または FWAIT）命令と TS フラグ（CR0 のビット 3）との相互作用を制御する。MP フラグがセットされていると、WAIT 命令は、TS フラグがセットされていればデバイス使用不可能例外（#NM）を生成する。MP フラグがクリアされていると、WAIT 命令は TS フラグの設定を無視する。表 9-2. に、システムで IA-32 プロセッサと x87 FPU かマス・

コプロセッサのどちらが存在するかに応じたこのフラグの推奨の設定を示す。表 2-1. に、MP フラグ、EM フラグ、TS フラグの相互作用を示す。

PE **保護イネーブル（CR0 のビット 0）。**このフラグがセットされると保護モードがイネーブルになり、クリアされると実アドレスモードがイネーブルになる。このフラグは、ページングを直接にイネーブルにすることはせず、システムレベルの保護をイネーブルにするだけである。ページングをイネーブルにするには、PE フラグと PG フラグの両方ともセットしなければならない。実アドレスモードと保護モードとを切り替えるための PE フラグの使用法の詳細については、9.9. 節「モード切り替え」を参照。

PCD **ページ・レベル・キャッシュ・ディスエーブル（CR3 のビット 4）。**現在のページ・ディレクトリのキャッシングを制御する。PCD フラグがセットされていると、ページ・ディレクトリのキャッシングが抑制され、このフラグがクリアされていると、ページ・ディレクトリをキャッシュすることができる。このフラグは、プロセッサの内部キャッシュ（存在しているときには、L1 および L2 の両方）だけに影響を与える。ページングが使用されていない（レジスタ CR0 の PG フラグがクリアされている）かまたは CR0 の CD（キャッシュ・ディスエーブル）フラグがセットされていると、プロセッサはこのフラグを無視する。このフラグの使用法の詳細については、第 10 章「メモリ・キャッシュ制御」を参照。ページ・ディレクトリ・エントリとページ・テーブル・エントリにある同類の PCD フラグの説明については、3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」を参照。

PWT **ページレベル書き込み透過（CR3 のビット 3）。**現在のページ・ディレクトリのライトスルーまたはライトバック・キャッシング・ポリシーを制御する。PWT フラグがセットされると、ライトスルー・キャッシングがイネーブルになり、このフラグがクリアされると、ライトバック・キャッシングがイネーブルになる。このフラグは、内部キャッシュ（存在しているときは、L1 と L2 の両方）だけに影響を与える。ページングが使用されていない（レジスタ CR0 の PG フラグがクリアされている）かまたは CR0 の CD（キャッシュ・ディスエーブル）フラグがセットされていると、プロセッサはこのフラグを無視する。このフラグの使用法の詳細については、10.5. 節「キャッシュの制御」を参照。ページ・ディレクトリ・エントリとページ・テーブル・エントリにある同類の PCD フラグの説明については、3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」を参照。

VME **仮想 8086 モード拡張（CR4 のビット 0）。**このフラグがセットされると、仮想 8086 モードでの割り込み / 例外処理拡張がイネーブルになる。クリアされると、拡張がディスエーブルになる。仮想モード拡張を使用すると、仮想 8086 アプリケーションの性能を向上できる。8086 プログラムを実行している間に発生する割り込みと例外を処理する仮想 8086 モニタをコールするオーバーヘッドを除去して、代わりに、割り込みと例外を 8086 プログラムのハンドラに返すようにしているためである。この拡張は、8086 プログラムがマルチタスキング環境および複

数プロセッサ環境で実行する信頼性を向上させるための仮想割り込みフラグ（VIF）のハードウェア・サポートも提供している。この機能の使用法の詳細については、16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照。

PVI　**保護モード仮想割り込み（CR4 のビット 1）。**このフラグがセットされると、保護モードで仮想割り込みフラグ（VIF）のハードウェア・サポートがイネーブルになる。クリアされると、保護モードで VIF フラグがディスエーブルになる。この機能の使用法の詳細については、16.4. 節「保護モード仮想割り込み」を参照。

TSD　**タイムスタンプ・ディスエーブル（CR4 のビット 2）。**このフラグがセットされると、RDTSC 命令が特権レベル 0 で実行しているプロシージャだけで実行するように限定される。クリアされると、RDTSC 命令がどの特権レベルでも実行できる。

DE　**デバッグ拡張（CR4 のビット 3）。**このフラグがセットされていると、デバッグレジスタ DR4 と DR5 への参照によって未定義オペコード（#UD）例外が生成される。クリアされていると、早期の IA-32 プロセッサで実行するように作成されているソフトウェアとの互換性のため、プロセッサは、レジスタ DR4 と DR5 の参照にエイリアスを割り当てる。このフラグの機能の詳細については、15.2.2. 項「デバッグレジスタ DR4 および DR5」を参照。

PSE　**ページサイズ拡張（CR4 のビット 4）。**このフラグがセットされると、4M バイト・ページがイネーブルになる。クリアされると、ページは 4K バイトに制限される。このフラグの使用法の詳細については、3.6.1. 項「ページングのオプション」を参照。

PAE　**物理アドレス拡張（CR4 のビット 5）。**このフラグがセットされると、36 ビット物理アドレスを参照するページング・メカニズムがイネーブルになる。クリアされると、物理アドレスは 32 ビットに限定される。物理アドレス拡張の詳細については、3.8. 節「PAE ページング・メカニズムを使用した 36 ビット 物理アドレス指定」を参照。

MCE　**マシン・チェック・イネーブル（CR4 のビット 6）。**このフラグがセットされると、マシンチェック例外がイネーブルになる。クリアされると、マシンチェック例外がディスエーブルになる。マシンチェック例外とマシン・チェック・アーキテクチャの詳細については、第 14 章「マシン・チェック・アーキテクチャ」を参照。

PGE　**ページ・グローバル・イネーブル（CR4 のビット 7）。**（P6 ファミリ・プロセッサで導入された。）このフラグがセットされると、グローバル・ページ機能がイネーブルになる。クリアされると、グローバル・ページ機能がディスエーブルになる。グローバル・ページ機能は、頻繁に使用されるページまたは共用されるページに、すべてのユーザに対してグローバルというマークが付くようにする（ページ・ディレクトリ・エントリまたはページ・テーブル・エントリのグローバル・フラグ（ビット 8）によって行われる）。グローバル・ページは、タスクス

イッチ時またはレジスタ CR3 への書き込み時にトランスレーション・ルックアサイド・バッファ（TLB）からフラッシュされない。

グローバル・ページ機能がイネーブルのときは、PGE フラグをセットする前に、(制御レジスタ CR0 の PG フラグをセットすることによって)ページングをイネーブルにしなければならない。この順序を逆にすると、プログラムの整合性が影響を受けて、プロセッサの性能が低下する。このビットの使用法の詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB)」を参照。

PCE **性能モニタリング・カウンタ・イネーブル（CR4 のビット 8)。**このフラグがセットされていると、任意の保護レベルで実行しているプログラムまたはプロシージャに対して RDPMC 命令を実行することができる。クリアされていると、RDPMC 命令は保護レベル 0 でしか実行することはできない。

OSFXSR

オペレーティング・システムによるFXSAVE/FXRSTOR命令のサポート（CR4 のビット 9)。このフラグがセットされると、次の機能が実行される。(1）オペレーティング・システムにおいて FXSAVE 命令と FXRSTOR 命令の使用がサポートされることをソフトウェアに示す。(2）FXSAVE 命令と FXRSTOR 命令を使用して、x87 FPU レジスタおよび MMX テクノロジ・レジスタの内容と一緒に XMM レジスタと MXCSR レジスタの内容も保存および復元できるようにする。(3）プロセッサで SSE 命令、SSE2 命令、SSE3 命令を実行できるようにする（ただし、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH は除く)。

このフラグがクリアされると、FXSAVE 命令と FXRSTOR 命令により x87 FPU レジスタおよび MMX テクノロジ・レジスタの内容は保存 / 復元されるが、XMM レジスタおよび MXCSR レジスタの内容は保存 / 復元されないことがある。

また、このフラグがクリアされると、プロセッサは、SSE 命令、SSE2 命令、SSE3 命令を実行しようとするたびに、無効オペコード例外（#UD）を生成する（ただし、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH は除く)。オペレーティング・システムやエグゼクティブでは、このフラグを明示的にセットする必要がある。

注記

CPUID 機能フラグの FXSR、SSE、SSE2、SSE3 はそれぞれ、FXSAVE/FXRESTOR 命令、SSE 拡張命令、SSE2 拡張命令、SSE3 拡張命令が特定の IA-32 プロセッサ上で使用できるかどうかを示す。OSFXSR ビットにより、オペレーティング・システム・ソフトウェア上でこれらの機能をイネーブルに設定できる。また、オペレーティング・システムがこれらの機能をサポートしているのかも示すこともできる。

OSXMMEXCPT

オペレーティング・システムによるマスクされていない SIMD 浮動小数点例外のサポート（CR4 のビット 10）。SIMD 浮動小数点例外（#XF）の発生時に起動される例外ハンドラを通じて、マスクされていない SIMD 浮動小数点例外の処理をオペレーティング・システムがサポートすることを示す。SIMD 浮動小数点例外を生成するのは、SSE、SSE2、SSE3 の SIMDSIMD 浮動小数点命令だけである。オペレーティング・システムやエグゼクティブでは、このフラグを明示的にセットする必要がある。このフラグをセットしないと、プロセッサは、マスクされていない SIMD 浮動小数点例外を検出するたびに、無効オペコード例外（#UD）を生成する。

2.5.1. 制御レジスタフラグの CPUID 確認

制御レジスタ CR4 の VME、PVI、TSD、DE、PSE、PAE、MCE、PGE、PCE、OSFXSR、および OSXMMCEPT の各フラグは、モデル固有である。（PCE フラグを除く）これらのフラグのすべてを CPUID 命令で確認して、それらのフラグを使用する前にプロセッサで使用できるかを判定できる。

2.6. システム命令のまとめ

システム命令は、システムレジスタのロード、キャッシュの管理、割り込みの管理、またはデバッグレジスタのセットアップなどシステムレベルの機能を処理する。これらの命令の多くは、オペレーティング・システムまたはエグゼクティブ・プロシージャ（つまり、特権レベル 0 で実行するプロシージャ）でしか実行できない。その他の命令は、任意の特権レベルで実行することができ、したがって、アプリケーション・プログラムで使用できる。

表 2-2. に、システム命令を一覧し、アプリケーション・プログラムで有効に利用できるかどうかを示す。これらの命令については、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」および『IA-32 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、中巻 B』の第 4 章「命令セット・リファレンス N-Z」で詳細に説明されている。

表 2-2. システム命令のまとめ

命令	説明	アプリケーションに有用	アプリケーションからの保護
LLDT	レジスタをロードする	いいえ	はい
SLDT	レジスタをストアする	いいえ	いいえ
LGDT	レジスタをロードする	いいえ	はい

表 2-2. システム命令のまとめ（続き）

命令	説明	アプリケーションに有用	アプリケーションからの保護
SGDT	レジスタをストアする	いいえ	いいえ
LTR	タスクレジスタをロードする	いいえ	はい
STR	タスクレジスタをストアする	いいえ	いいえ
LIDT	レジスタをロードする	いいえ	はい
SIDT	レジスタをストアする	いいえ	いいえ
MOV CR*n*	制御レジスタをロードおよびストアする	いいえ	はい
SMSW	MSW をストアする	はい	いいえ
LMSW	MSW をロードする	いいえ	はい
CLTS	CR0 の TS フラグをクリアする	いいえ	はい
ARPL	RPL を調整する	はい[1]	いいえ
LAR	アクセス権をロードする	はい	いいえ
LSL	セグメント・リミットをロードする	はい	いいえ
VERR	読み取りのために検証する	はい	いいえ
VERW	書き込みのために検証する	はい	いいえ
MOV DB*n*	デバッグレジスタをロードおよびストアする	いいえ	はい
INVD	キャッシュを無効化する（ライトバック無）	いいえ	はい
WBINVD	キャッシュを無効化する（ライトバック有）	いいえ	はい
INVLPG	TLB エントリを無効化する	いいえ	はい
HLT	プロセッサを停止する	いいえ	はい
LOCK（プリフィックス）	バスロック	はい	いいえ
RSM	システム管理モードから戻る	いいえ	はい
RDMSR[3]	モデル固有レジスタを読み取る	いいえ	はい
WRMSR[3]	モデル固有レジスタに書き込む	いいえ	はい
RDPMC[4]	性能モニタリング・カウンタを読み取る	はい	はい[2]
RDTSC[3]	タイムスタンプ・カウンタを読み取る	はい	はい[2]

注：

1. CPL 1 または 2 で実行するアプリケーション・プログラムに役立つ。
2. 制御レジスタ CR4 の TSD フラグと PCE フラグは、CPL 3 で実行するアプリケーション・プログラムによるこれらの命令へのアクセスを制御する。
3. これらの命令はインテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。
4. この命令は、インテル® Pentium® Pro プロセッサと MMX® テクノロジ Pentium® プロセッサでインテル・アーキテクチャに導入された。

2.6.1. システムレジスタのロードとストア

GDTR レジスタ、LDTR レジスタ、IDTR レジスタ、TR レジスタには、それぞれレジスタへのデータのロードとレジスタ内のデータストアのためのロード命令とストア命令が与えられている。

LGDT（GDTR レジスタのロード）
: GDT のベースアドレスとリミットをメモリから GDTR レジスタにロードする。

SGDT（GDTR レジスタのストア）
: GDT のベースアドレスとリミットを GDTR レジスタからメモリにストアする。

LIDT（IDTR レジスタのロード）
: IDT のベースアドレスとリミットをメモリから IDTR レジスタにロードする。

SIDT（IDTR レジスタのストア）
: IDT のベースアドレスとリミットを IDTR レジスタからメモリにストアする。

LLDT（LDT レジスタのロード）
: LDT のセグメント・セレクタとセグメント・ディスクリプタをメモリから LDTR にロードする。（セグメント・セレクタ・オペランドは、汎用レジスタに置くこともできる。）

SLDT（LDT レジスタのストア）
: LDT のセグメント・セレクタを LDTR レジスタからメモリまたは汎用レジスタにストアする。

LTR（タスクレジスタのロード）
: TSS のセグメント・セレクタとセグメント・ディスクリプタをメモリからタスクレジスタにロードする。（セグメント・セレクタ・オペランドは、汎用レジスタに置くこともできる。）

STR（タスクレジスタのストア）
: 現行タスク TSS のセグメント・セレクタをタスクレジスタからメモリまたは汎用レジスタにストアする。

LMSW（マシン・ステータス・ワードのロード）命令と SMSW（マシン・ステータス・ワードのストア）命令は、制御レジスタ CR0 のビット 0 からビット 15 までを操作する。これらの命令は、16 ビットのインテル® 286 プロセッサとの互換性を保つために備えられている。32 ビットの IA-32 プロセッサで実行するように作成されたプログラムでは、これらの命令を使用してはならない。その代わりに、それらのプログラムは、MOV 命令を使用して制御レジスタ CR0 にアクセスする必要がある。

CLTS（CR0のTSフラグのクリア）命令は、TSフラグがセットされているときにプロセッサが浮動小数点命令を実行しようとすると発生するデバイス使用不可能例外（#NM）の処理に使用するために備えられている。この命令を使用して、x87 FPUコンテキストをセーブした後にTSフラグをクリアして、それ以上の#NM例外の発生を防ぐことができる。TSフラグの詳細については、2.5.節「制御レジスタ」を参照。

制御レジスタ（CR0、CR1、CR2、CR3、CR4）には、MOV命令を使用してロードする。この命令は、汎用レジスタからの制御レジスタのロードまたは汎用レジスタへの制御レジスタの内容のストアを行うことができる。

2.6.2. アクセス特権の確認

プロセッサは、セグメントへのアクセスを認めるかどうかを判定するため、そのセグメントに対応するセグメント・セレクタとセグメント・ディスクリプタを調べるための複数の命令を用意している。これらの命令は、プロセッサによって行われる自動アクセス権チェックとタイプチェックの一部を模倣して行っているため、オペレーティング・システムまたはエグゼクティブ・ソフトウェアは、例外が発生しないようにできる。

ARPL（RPLの調整）命令は、セグメント・セレクタのRPL（要求される特権レベル）を、そのセグメント・セレクタを指定したプログラムまたはプロシージャのRPLと一致するように調整する。この命令の機能と使用法の詳細については、4.10.4.項「呼び出し側のアクセス特権のチェック（ARPL命令）」を参照。

LAR（アクセス権のロード）命令は、指定されたセグメントのアクセス可能性を検証し、アクセス権情報をセグメントのセグメント・ディスクリプタから汎用レジスタにロードする。ソフトウェアは、その後、アクセス権を調べてセグメント・タイプが意図された用途に適合しているかを判定できる。この命令の機能と使用法の詳細については、4.10.1.項「アクセス権のチェック（LAR命令）」を参照。

LSL（セグメント・リミットのロード）命令は、指定されたセグメントのアクセス可能性を検証し、セグメント・リミットをセグメントのセグメント・ディスクリプタから汎用レジスタにロードする。ソフトウェアは、その後、セグメント・リミットをセグメントへのオフセットと比較してオフセットがセグメント内にあるかどうかを判定できる。この命令の機能と使用法の詳細については、4.10.3.項「ポインタ・オフセットがリミット内にあるかどうかのチェック（LSL命令）」を参照。

VERR（読み取りのための検証）命令とVERW（書き込みのための検証）命令は、選択されているセグメントが指定されたCPLでそれぞれ読み取り可能かまたは書き込み可能かを検証する。この命令の機能と使用法の詳細については、4.10.2.項「読み取り/書き込み権のチェック（VERR命令とVERW命令）」を参照。

2.6.3. デバッグレジスタのロードとストア

プロセッサの内部デバッグ機能は、8 個のデバッグレジスタ（DR0 ～ DR7）の設定によって制御される。MOV 命令によって、セットアップ・データをこれらのレジスタにロードし、これらのレジスタからストアできる。

2.6.4. キャッシュと TLB の無効化

プロセッサは、キャッシュと TLB エントリを明示的に無効化する際に使用するための複数の命令を備えている。INVD（ライトバックなしのキャッシュの無効化）命令は、内部キャッシュにあるすべてのデータエントリと命令エントリを無効化し、外部キャッシュも無効化しなければならないことを指示する信号を外部キャッシュに送る。

WBINVD（ライトバックありのキャッシュの無効化）命令は、INVD 命令と同じ機能を実行するが、キャッシュを無効化する前に内部キャッシュに修正された行がある場合には、それらをメモリにライトバックする点が異なる。WBINVD 命令は、内部キャッシュを無効化した後、修正されたデータをライトバックして、その後外部キャッシュの内容を無効化する信号を外部キャッシュに送る。

INVLPG（TLB エントリの無効化）命令は、指定されたページの TLB エントリを無効化する（フラッシュする）。

2.6.5. プロセッサの制御

HLT（プロセッサの停止）命令は、イネーブルになっている割り込み（NMI や SMI など、これらは通常はイネーブルになっている）、デバッグ例外 BINIT# 信号、INIT# 信号、または RESET# 信号を受け取るまでプロセッサを停止する。プロセッサは、停止モードに入っていることを示す特別なバスサイクルを生成する。

ハードウェアは、この信号にいくつかの方法で応答できる。つまり、フロントパネルにあるインジケータ・ライトを点灯させることもある。診断情報を記録するための NMI 割り込みを生成することもある。リセット初期化を呼び出すこともある。BINIT# ピンはインテル® Pentium® Pro プロセッサで導入されたことに注意が必要である。シャットダウン時に非ウェークイベントが保留されていた場合、それらのイベントは、シャットダウンからのウェークイベント（例えば、A20M# 割り込み）が処理された後で処理される。

LOCK プリフィックスは、メモリ・オペランドを修正するときにロックされた（アトミック）読み取り - 修正 - 書き込み操作を起動する。このメカニズムは、以下に説明するように、マルチプロセッサ・システムのプロセッサ間で信頼できる通信が可能になるように使用される。

- インテル® Pentium® プロセッサとそれ以前の IA-32 プロセッサでは、LOCK プリフィックスによって、プロセッサは、命令中に LOCK# 信号をアサートし、これは常に明示的なバスロックを発生させる。
- インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、ロック操作は、キャッシュ・ロックまたはバスロックで扱われる。メモリアクセスがキャッシュ可能であり、1 つのキャッシュ・ラインにしか影響を与えない場合は、キャッシュ・ロックが起動されるが、システムバスとシステムメモリ内の実際のメモリ位置は、操作の間にロックされない。この場合に、バス上の他のインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサは、修正されたデータがあればそれをライトバックし、必要に応じてそれらのキャッシュを無効にしてシステムメモリのコヒーレンシを維持する。メモリアクセスがキャッシュ可能でない場合やキャッシュ・ライン境界を越える場合は、プロセッサの LOCK# 信号がアサートされ、プロセッサは、ロックされた操作の間はバス制御の要求に応答しない。

RSM（SMM からの戻り）命令は、システム管理モード（SMM）割り込み前にあった状態にプロセッサを（コンテキスト・ダンプから）リストアする。

2.6.6. 性能モニタリング・カウンタとタイムスタンプ・カウンタの読み取り

性能モニタリング・カウンタの読み取り（RDPMC: Read Performance-monitoring Counter）命令とタイムスタンプ・カウンタの読み取り（RDTSC: Read Time-stamp Counter）命令によって、アプリケーション・プログラムは、それぞれプロセッサの性能モニタリング・カウンタとタイムスタンプ・カウンタを読み取ることができる。インテル® Pentium® 4 プロセッサとインテル® Xeon™ プロセッサには 18 個の 40 ビットの性能モニタリング・カウンタがあり、P6 ファミリ・プロセッサには 2 つの 40 ビットの性能カウンタがある。

これらのカウンタは、イベントの発生または持続期間を記録するのに使用される。モニタリングできるイベントはモデル固有であり、デコードされた命令数、受け取った割り込み、キャッシュ・ロードの回数が含まれる。さまざまなイベントをモニタリングするように個別のカウンタをセットアップできる。システム命令 WRMSR を使用して、45 個の ESCR の 1 つおよび 18 個の CCCR MSR の 1 つ（インテル Pentium 4 プロセッサとインテル Xeon プロセッサの場合）もしくは、PerfEvtSel0 または PerfEvtSel1 MSR（P6 ファミリ・プロセッサの場合）の値をセットアップする。RDPMC 命令は、選択したカウンタの現在のカウントを EDX:EAX レジスタにロードする。

タイムスタンプ・カウンタはモデル固有の 64 ビット・カウンタであり、プロセッサがリセットされるたびにゼロにリセットされる。リセットされない場合には、プロセッサが 200MHz のクロックレートで動作しているときは、カウンタは 1 年に～6.3×10^{15} 回インクリメントする。このクロック周波数では、カウンタが一巡するには 2000 年以

上を要する。RDTSC 命令は、タイムスタンプ・カウンタの現在のカウントを EDX:EAX レジスタにロードする。

性能モニタリング・カウンタとタイムスタンプ・カウンタの詳細については、15.7. 節「タイムスタンプ・カウンタ」と 15.8. 節「性能モニタリングの概要」を参照。

RDTSC 命令は、インテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。RDPMC 命令は、インテル® Pentium® Pro プロセッサと MMX® テクノロジ Pentium® プロセッサで IA-32 アーキテクチャに導入された。それ以前のインテル Pentium プロセッサには、2 つの性能モニタリング・カウンタがあるが、それらのカウンタは RDMSC 命令でのみ、特権レベル 0 でのみ読み取ることができる。

2.6.7. モデル固有レジスタの読み取りと書き込み

モデル固有レジスタの読み取り（RDMSR: Read Performance-monitoring Counter）命令とモデル固有レジスタへの書き込み（CORMSR: Corite Model-specific Register）命令によって、プロセッサの 64 ビットのモデル固有レジスタの読み取りと書き込みをそれぞれ行うことができる。読み取るまたは書き込む MSR は、ECX レジスタの値によって指定される。

RDMSR 命令は、指定された MSR から EDX:EAX レジスタに値を読み取る。WRMSR 命令は、EDX:EAX レジスタの値を指定された MSR に書き込む。RDMSR 命令と WRMSR 命令は、インテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。

MSR の詳細については、9.4. 節「モデル固有レジスタ（MSR）」を参照。

3

保護モードにおけるメモリ管理

第 3 章
保護モードにおける
メモリ管理

3

本章では、物理メモリの要件、セグメンテーション・メカニズム、ページング・メカニズムをはじめとする IA-32 アーキテクチャの保護モードにおけるメモリ管理について説明する。プロセッサの保護メカニズムの説明については、第 4 章「保護」を参照。実アドレスモードと仮想 8086 モードでのメモリアドレス指定保護の説明については、第 16 章「8086 エミュレーション」を参照。

3.1. メモリ管理の概要

IA-32 アーキテクチャのメモリ管理機能は、セグメンテーションとページングという 2 つの部分に分かれている。セグメンテーションは、複数のプログラム（またはタスク）が相互に干渉しないで同じプロセッサ上で実行できるように、個別のコード・モジュール、データ・モジュール、スタック・モジュールを分離するメカニズムである。ページングは、プログラムの実行環境のセクションが必要に応じて物理メモリにマッピングされる従来のデマンドページの仮想メモリシステムを使用するためのメカニズムである。ページングは、複数のタスク間の分離にも使用できる。保護モードで動作しているときは、なんらかの形式のセグメンテーションを使用しなければならない。**セグメンテーションをディスエーブルにするモードビットはない。**ただし、ページングの使用はオプションである。

これら 2 つのメカニズム（セグメンテーションとページング）は、単純なシングルプログラム（またはシングルタスク）システム、マルチタスキング・システム、または共有メモリを使用する複数プロセッサ・システムをサポートするように構成できる。

図 3-1. に示すように、セグメンテーションは、プロセッサのアドレス指定可能なメモリ空間（**リニアアドレス空間**という）を**セグメント**のより小さい保護されたアドレス空間に分割するメカニズムである。セグメントを使用して、プログラムのコード、データ、スタックを保持したり、（TSS や LDT などの）システムデータ構造を保持したりできる。複数のプログラム（またはタスク）が 1 つのプロセッサ上で動作している場合には、各プログラムに専用のセグメント・セットを割り当てることができる。プロセッサは、その場合には、これらのセグメント間に境界を設け、1 つのプログラムが他のプログラムのセグメントに書き込むことによって別のプログラムの実行に干渉しないように保証する。セグメンテーション・メカニズムによってセグメントをタイプ分類し、特定のタイプのセグメントで実行できる操作も限定できる。

1 つのシステム内のすべてのセグメントは、プロセッサのリニアアドレス空間内に入っている。特定のセグメントにあるバイト位置を指定するには、**論理アドレス**（far ポインタともいう）を指定しなければならない。論理アドレスは、セグメント・セレクタとオフセットで構成される。セグメント・セレクタは、セグメントに固有の識別子である。セグメント・セレクタは、まず第一に、セグメント・ディスクリプタというデータ構造へのディスクリプタ・テーブル（グローバル・ディスクリプタ・テーブル GDT など）内へのオフセットを提供する。各セグメントは、セグメント・ディスクリプタを持ち、これがセグメントのサイズ、セグメントのアクセス権と特権レベル、セグメント・タイプ、リニアアドレス空間内のセグメントの先頭バイトの位置（セグメントのベースアドレスという）を指定する。論理アドレスのオフセット部分は、セグメントのベースアドレスに加えられてセグメント内のバイト位置を指定する。したがって、ベースアドレスとオフセットを加えたものがプロセッサのリニアアドレス空間内の**リニアアドレス**となる。

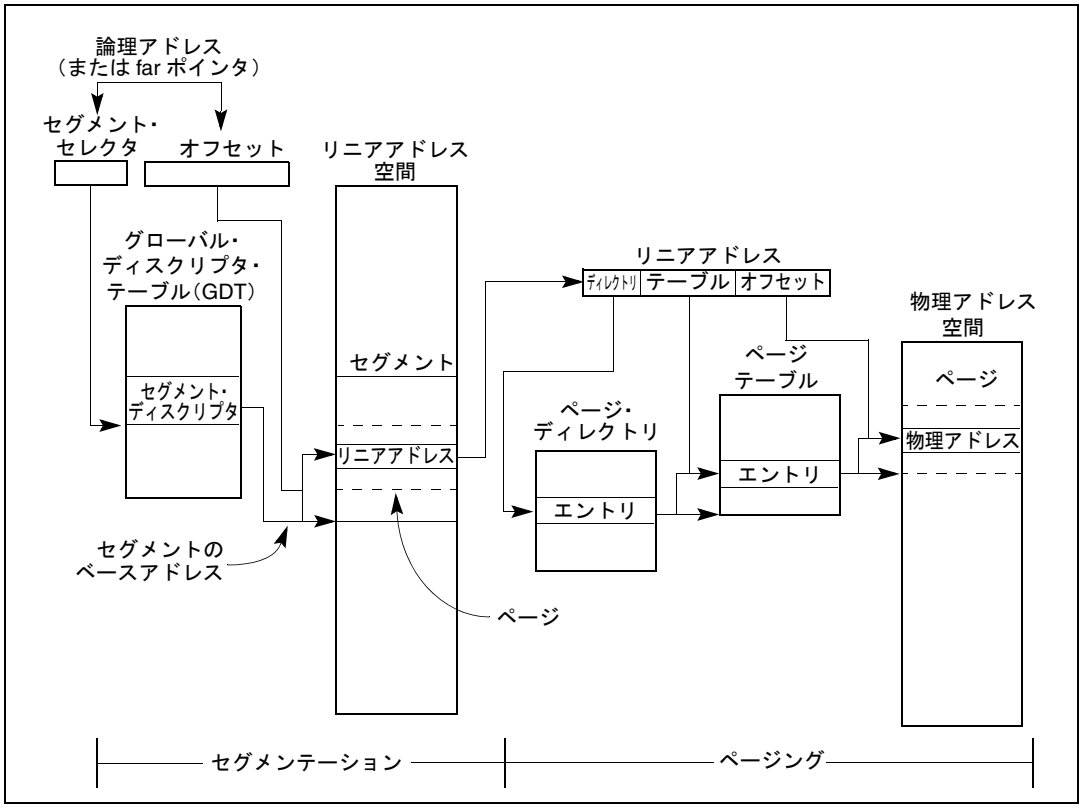


図 3-1. セグメンテーションとページング

ページングを使用しないと、プロセッサのリニアアドレス空間は、プロセッサの物理アドレス空間に直接にマッピングされる。物理アドレス空間は、プロセッサがそのアドレスバス上に生成できるアドレスの範囲として定義される。

マルチタスキング・コンピューティング・システムでは、一般的には、物理メモリに入りきらない、はるかに大きいリニアアドレス空間を定義するので、リニアアドレス空間を「仮想化する」何らかの方式が必要になる。このリニアアドレス空間の仮想化は、プロセッサのページング・メカニズムによって処理される。

ページングは、小量の物理メモリ（RAM と ROM）と若干のディスク記憶領域を使用して大きなリニアアドレス空間をシミュレートする、「仮想メモリ」環境をサポートする。ページングを使用すると、各セグメントは、ページ（通常はそれぞれ 4K バイトのサイズ）に分割され、ページが物理メモリまたはディスクにストアされる。オペレーティング・システムまたはエグゼクティブは、ページ・ディレクトリと一組のページテーブルを維持してページの動きを追跡する。プログラム（またはタスク）がリニアアドレス空間のアドレス位置にアクセスしようとすると、プロセッサは、ページ・ディレクトリとページテーブルを使用して、リニアアドレスを物理アドレスに変換してから要求された操作（読み取りまたは書き込み）をメモリ位置に実行する。

アクセスされるページがその時点で物理メモリにないと、プロセッサは、(ページフォルト例外を生成することによって) プログラムの実行に割り込む。オペレーティング・システムまたはエグゼクティブは、その後、ページをディスクから物理メモリに読み込み、プログラムの実行を続ける。

ページングがオペレーティング・システムまたはエグゼクティブに適切に採用されていると、物理メモリとディスクとの間のページのスワップは、プログラムが正しく実行されていればそこから透過的になる。16 ビットの IA-32 プロセッサ用に作成されたプログラムでも、仮想 8086 モードで実行されるときは（透過的に）ページングができる。

3.2. セグメントの使用法

IA-32 アーキテクチャによってサポートされるセグメンテーション・メカニズムを使用して、幅広い多様なシステム設計を実現することができる。これらの設計は、プログラムを保護するためにセグメンテーションを最小限にしか利用しないフラットなモデルから、複数のプログラムとタスクを確実に実行できる頑健なオペレーティング環境を生成するためにセグメンテーションを採用するマルチセグメント化したモデルまで広範囲に及ぶ。

以降の項では、システムでセグメンテーションを使用してメモリ管理の性能と信頼性を向上させる方法を示す例をいくつか挙げる。

3.2.1. 基本フラットモデル

システムの最も単純なメモリモデルは、基本的な「フラットモデル」であり、このモデルでは、オペレーティング・システムとアプリケーション・プログラムは、連続したセグメント化されていないアドレス空間にアクセスする。最大限に可能な範囲まで、この基本フラットモデルでは、アーキテクチャのセグメンテーション・メカニズムをシステム設計者とアプリケーション・プログラマの両方から見えないようにしている。

IA-32 アーキテクチャで基本フラット・メモリ・モデルを使用するには、少なくとも 2 つのセグメント・ディスクリプタを作成しなければならならない。1 つはコード・セグメント参照用で、もう 1 つはデータ・セグメント参照用である（図 3-2. を参照）。ただし、これらのセグメントの両方ともリニアアドレス空間全体にマッピングされる。つまり、両方のセグメント・ディスクリプタとも同じ 0 のベースアドレス値で同じ 4G バイトのセグメント・リミットである。セグメント・リミットを 4G バイトに設定することによって、特定のアドレスに常駐する物理メモリがない場合でも、セグメンテーション・メカニズムは、リミット外のメモリ参照に対して例外を生成することはない。プロセッサは FFFF_FFF0H で実行を開始するので、ROM（EPROM）は、一般的には物理アドレス空間の一番上に置かれる。リセット初期化後の DS データ・セグメントの初期ベースアドレスは 0 であるので、RAM（DRAM）はアドレス空間の一番下に置かれる。

3.2.2. 保護されたフラットモデル

保護されたフラットモデルは、基本フラットモデルと同様であるが、物理メモリが実際に存在しているアドレス範囲だけを含むようにセグメント・リミットが設定される点が異なる（図 3-3. を参照）。この場合には、存在していないメモリをアクセスしようとすると、一般保護例外（#GP）が発生する。このモデルは、ある種のプログラム・バグに対する最小限のレベルのハードウェア保護を備えている。

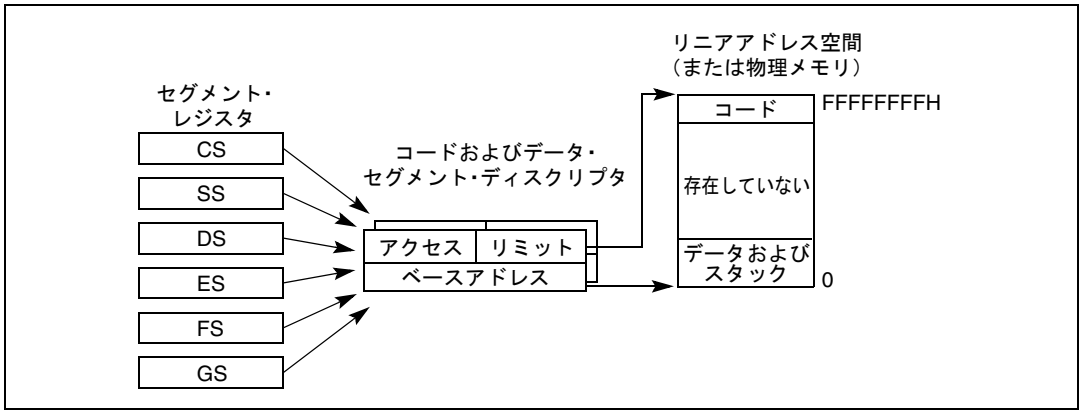


図 3-2. フラットモデル

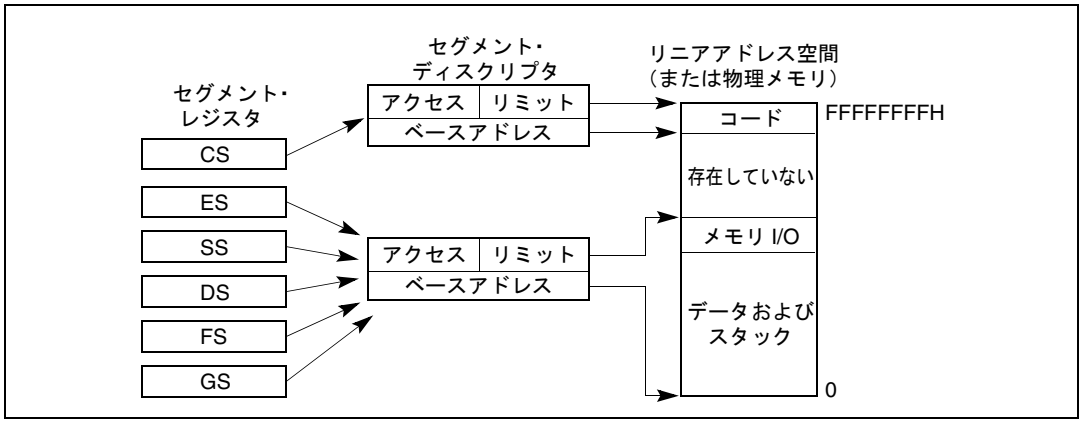


図 3-3. 保護されたフラットモデル

さらに保護するため、この保護されたフラットモデルをさらに複雑にすることができる。例えば、ページング・メカニズムでユーザとスーパーバイザのコードとデータを分離できるようにするには、4 つのセグメントを定義する必要がある。ユーザ用に特権レベル 3 でのコード・セグメントとデータ・セグメント、スーパーバイザ用に特権レベル 0 でのコード・セグメントとデータ・セグメントである。通常、これらのセグメントはすべて相互にオーバレイし、リニアアドレス空間のアドレス 0 で始まる。このフラットなセグメンテーション・モデルは、単純なページング構造と共に、オペレーティング・システムをアプリケーションから保護ができ、タスクまたはプロセスごとに別個のページング構造を加えることによって、アプリケーションを相互にも保護ができる。同様の設計は、複数のよく知られたマルチタスキング・オペレーティング・システムで使用されている。

3.2.3. マルチセグメント・モデル

マルチセグメント・モデル（図3-4.に示すモデルなど）は、セグメンテーション・メカニズムの機能すべてを利用して、コード、データ構造およびプログラムとタスクの保護をハードウェアに行わせている。この場合には、各プログラム（またはタスク）は、セグメント・ディスクリプタの専用のテーブルと専用のセグメントを与えられる。セグメントは、割り当てられたプログラムだけの専用にすることも、複数のプログラム間での共有にすることもできる。すべてのセグメントへのアクセスとシステムで実行される個別プログラムの実行環境へのアクセスは、ハードウェアによって制御される。

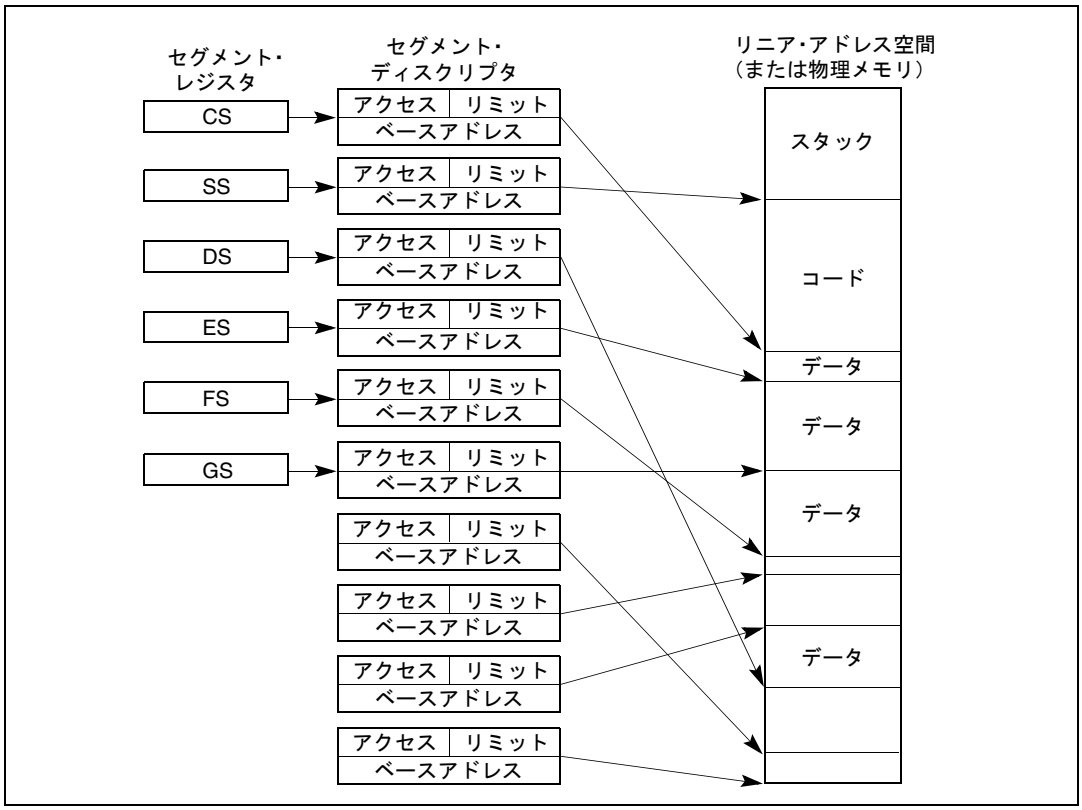


図 3-4. マルチセグメント・モデル

アクセスチェックを使用して、セグメントのリミット外のアドレスへの参照に対してだけではなく、一定のセグメントには許可されていない操作の実行に対しても保護することができる。例えば、コード・セグメントは読み取り専用セグメントとして指定されるので、ハードウェアを使用してコード・セグメントへの書き込みを防止できる。セグメントに対して作成されるアクセス権情報を使用して、保護リングまたは保護レベルもセットアップできる。保護レベルを使用して、アプリケーション・プログラム

がオペレーティング・システム・プロシージャを無許可でアクセスしないように保護できる。

3.2.4. ページングとセグメンテーション

ページングは、図3-2.、図3-3.、および図3-4. で説明したセグメンテーション・モデルのいずれとも一緒に使用できる。プロセッサのページング・メカニズムは、（セグメントがマッピングされる）リニアアドレス空間を（図3-1. に示すように）ページに分割する。これらのリニアアドレス空間ページは、その後、物理アドレス空間のページにマッピングされる。ページング・メカニズムは、セグメント保護機能と一緒にまたはその代わりに使用できる複数のページレベル保護機能を備えている。例えば、ページ単位で読み取り / 書き込み保護を行うことができる。ページング・メカニズムは、ページ単位でも指定できる2 レベルのユーザ - スーパーバイザ保護も備えている。

3.3. 物理アドレス空間

保護モードでは、IA-32 アーキテクチャは、4G バイト（2^{32} バイト）の通常物理アドレス空間を提供する。これは、プロセッサがそのアドレスバス上でアドレス指定できるアドレス空間である。このアドレス空間は、フラットであり（セグメント化されておらず）、アドレスは 0 から FFFFFFFFH までの連続した範囲である。この物理アドレス空間は、読み取り / 書き込みメモリ、読み取り専用メモリ、メモリマップド I/O にマッピングできる。本章で説明するメモリ・マッピング機能を利用して、この物理メモリをセグメントまたはページ、あるいはその両方に分割できる。

インテル® Pentium® Pro プロセッサ以降のプロセッサで、IA-32 アーキテクチャは、2^{36} バイト（64G バイト）への物理アドレス空間の拡張もサポートしており、最大物理アドレスは FFFFFFFFFH になる。この拡張は、次の 2 つの方法のいずれかで起動される。

- 制御レジスタ CR4 のビット 5 の物理アドレス拡張（PAE）フラグを使用する。
- インテル® Pentium® III プロセッサで導入された 36 ビット・ページ・サイズ拡張（PSE-36）機能を使用する。

36 ビットの物理アドレス指定の詳細については、3.8. 節「PAE ページング・メカニズムを使用した 36 ビット 物理アドレス指定」および 3.9. 節「PSE-36 ページング・メカニズムを使用した 36 ビット物理アドレス指定」を参照のこと。

3.4. 論理アドレスとリニアアドレス

保護モードのシステム・アーキテクチャ・レベルでは、プロセッサは、論理アドレス変換とリニアアドレス空間ページングという 2 段階のアドレス変換を使用して物理アドレスに到達する。

セグメントを最小限にしか利用していない場合でも、プロセッサのアドレス空間にあるすべてのバイトは、論理アドレスによってアクセスされる。論理アドレスは、16 ビットのセグメント・セレクタと 32 ビットのオフセットで構成される（図 3-5. を参照）。セグメント・セレクタはバイトが入っているセグメントを識別し、オフセットはセグメントのベースアドレスに相対的なセグメント内のバイトの位置を指定する。

プロセッサは、すべての論理アドレスをリニアアドレスに変換する。リニアアドレスは、プロセッサのリニアアドレス空間内の 32 ビットのアドレスである。物理アドレス空間と同様に、リニアアドレス空間は、フラットな（セグメント化されない）2^{32} バイトのアドレス空間であり、アドレスは 0 から FFFFFFFH の範囲に及ぶ。リニアアドレス空間には、システムについて定義されているすべてのセグメントとシステムテーブルが入っている。

論理アドレスをリニアアドレスに変換するため、プロセッサは次の操作を行う。

1. セグメント・セレクタのオフセットを使用して、GDT または LDT にあるセグメントのセグメント・ディスクリプタの位置を見つけ、それをプロセッサに読み込む。（このステップは、新しいセグメント・セレクタがセグメント・レジスタにロードされるときだけに必要である。）
2. セグメント・ディスクリプタを調べてセグメントのアクセス権と範囲をチェックし、セグメントがアクセス可能であることとオフセットがセグメントのリミット内にあることを確認する。
3. セグメント・ディスクリプタからのセグメントのベースアドレスをオフセットに加算してリニアアドレスを得る。

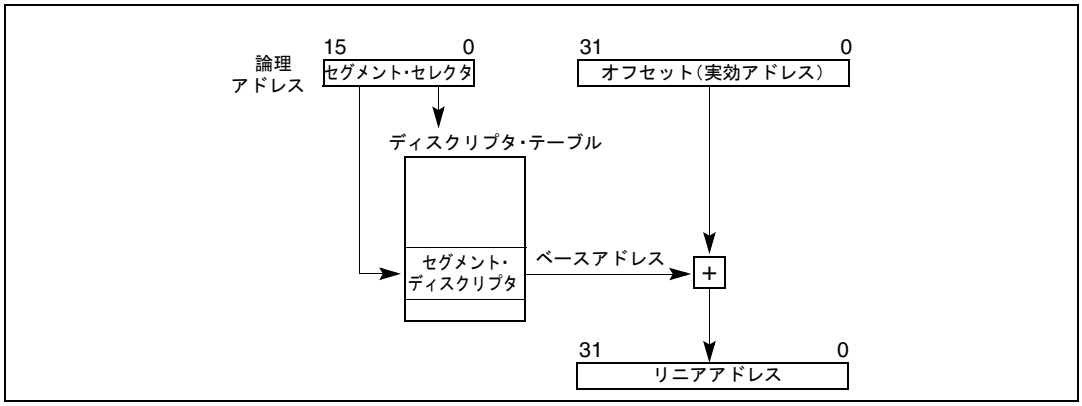


図 3-5. 論理アドレスからリニアアドレスへの変換

ページングを使用しない場合、プロセッサはリニアアドレスを物理アドレスに直接にマッピングする（つまり、リニアアドレスがプロセッサのアドレスバス上に移行する）。リニアアドレス空間がページングされている場合は、第二レベルのアドレス変換が使用されてリニアアドレスを物理アドレスに変換する。ページ変換は、3.6. 節「ページング（仮想メモリ）の概要」に説明されている。

3.4.1. セグメント・セレクタ

セグメント・セレクタは、16 ビットのセグメント識別子である（図 3-6. を参照）。これは、セグメントを直接に指すのではなく、その代わりにセグメントを定義するセグメント・ディスクリプタを指す。セグメント・セレクタには、次のアイテムが入っている。

インデックス （ビット 3 ～ 15）。GDT または LDT にある 8192 個のディスクリプタのうちの 1 つを選択する。プロセッサは、インデックス値に 8（セグメント・ディスクリプタのバイト数）を掛け、結果を（それぞれ GDTR レジスタまたは LDTR レジスタからの）GDT または LDT のベースアドレスに加算する。

TI（テーブル・インジケータ）
フラグ（ビット 2）。使用するディスクリプタ・テーブルを指定する。このフラグをクリアすると GDT が選択され、セットすると現在の LDT が選択される。

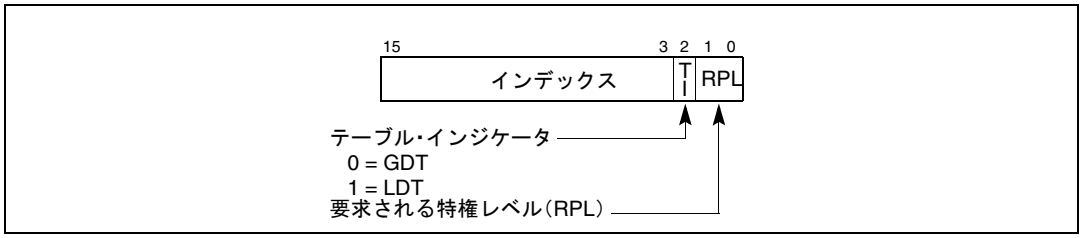


図 3-6. セグメント・セレクタ

要求される特権レベル（RPL）

（ビット 0 と 1）。セレクタの特権レベルを指定する。特権レベルは 0 ～ 3 の範囲が可能であり、0 が最も特権レベルが高い。RPL と実行しているプログラム（またはタスク）の CPL との関係およびセグメント・セレクタが指しているディスクリプタのディスクリプタ特権レベル（DPL）の説明については、4.5. 節「特権レベル」を参照。

プロセッサは GDT の最初のエントリを使用しない。GDT のこのエントリを指すセグメント・セレクタ（つまり、インデックスが 0 であり、TI フラグが 0 に設定されたセグメント・セレクタ）は、「ヌル・セグメント・セレクタ」として使用される。（CS レジスタまたは SS レジスタ以外の）セグメント・レジスタにヌル・セレクタがロードされても、プロセッサは例外を生成しない。ただし、ヌル・セレクタの入っているセグメント・レジスタがメモリのアクセスに使用されると、プロセッサは例外を生成する。ヌル・セレクタを使用して、未使用のセグメント・レジスタを初期化できる。CS レジスタまたは SS レジスタにヌル・セグメント・セレクタをロードすると、一般保護例外（#GP）が発生する。

セグメント・セレクタは、ポインタ変数の一部としてアプリケーション・プログラムから見えるが、セレクタの値は、通常はアプリケーション・プログラムではなくリンクエディタまたはリンキング・ローダによって割り当てられたり修正される。

3.4.2. セグメント・レジスタ

アドレス変換の時間を短縮し、コーディングの複雑さを低減するため、プロセッサは 6 つまでのセグメント・セレクタを保持できるレジスタを備えている（図 3-7. を参照）。これらのセグメント・レジスタはそれぞれ、特定の種類のメモリ参照（コード、スタック、またはデータ）をサポートする。どのような種類のプログラム実行も行うには、少なくともコード・セグメント（CS）、データ・セグメント（DS）、スタック・セグメント（SS）の各レジスタに有効なセグメント・セレクタをロードしなければならない。プロセッサは、その他に 3 つのデータ・セグメント・レジスタ（ES、FS、GS）も備えており、これらのレジスタを使用して、現在実行しているプログラム（またはタスク）に使用できるデータ・セグメントを追加できる。

プログラムがセグメントにアクセスするには、そのセグメントのセグメント・セレクタがセグメント・レジスタの1つにロードされていなければならない。そのため、システムは数千のセグメントを定義できるが、即時使用可能の状態にできるのは6個だけである。その他のセグメントは、プログラム実行中にそれらのセグメント・セレクタをこれらのレジスタにロードすれば使用できるようになる。

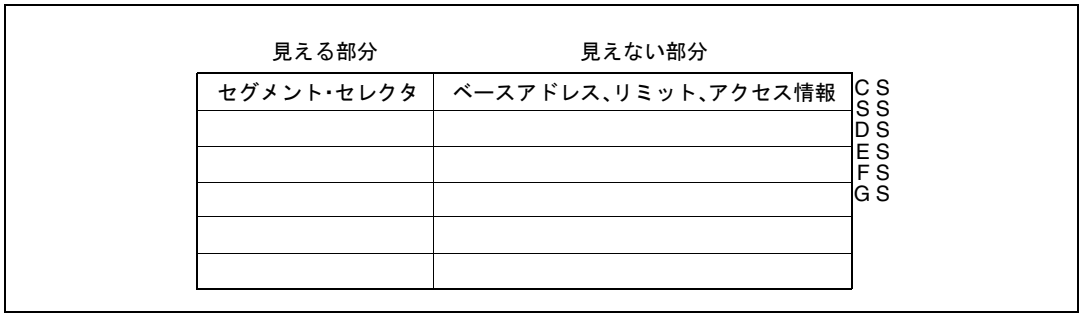


図 3-7. セグメント・レジスタ

すべてのセグメント・レジスタには、「見える」部分と「見えない」部分がある（見えない部分を「ディスクリプタ・キャッシュ」または「シャドーレジスタ」ということもある）。セグメント・セレクタがセグメント・レジスタの見える部分にロードされると、プロセッサは、セグメント・レジスタの見えない部分にもベースアドレス、セグメント・リミット、アクセス制御情報をセグメント・セレクタによって指されているセグメント・ディスクリプタからロードする。（見えるおよび見えない）セグメント・レジスタにキャッシュされた情報によって、プロセッサは、ベースアドレスとリミットをセグメント・ディスクリプタから読み取る余分なバスサイクルを行わずにアドレスを変換できる。複数のプロセッサが同じディスクリプタ・テーブルにアクセスするシステムでは、ディスクリプタ・テーブルが修正されたときにセグメント・レジスタを再ロードすることは、ソフトウェアが実行しなければならない。これが行われないと、メモリに常駐しているディスクリプタが修正された後でも、セグメント・レジスタにキャッシュされている古いセグメント・ディスクリプタが使用されるおそれがある。

2種類のロード命令がセグメント・レジスタをロードするために用意されている。

1. MOV、POP、LDS、LES、LSS、LGS、LFS 命令などの直接的なロード命令。これらの命令は、セグメント・レジスタを明示的に参照する。
2. CALL、JMP、RET 命令の far ポインタ・バージョン、SYSENTER および SYSEXIT 命令、および IRET、INTn、INTO、INT3 命令などの暗黙のロード命令。これらの命令は、命令の機能の中で、CS レジスタ（場合によっては他のセグメント・レジスタ）の内容を変更する。

MOV 命令は、セグメント・レジスタの見える部分を汎用レジスタにストアするためにも使用できる。

3.4.3. セグメント・ディスクリプタ

セグメント・ディスクリプタは、GDT または LDT にあるデータ構造で、セグメントのサイズと位置、およびアクセス制御情報とステータス情報をプロセッサに提供する。セグメント・ディスクリプタは、一般的にはコンパイラ、リンカ、ローダ、またはオペレーティング・システムやエグゼクティブによって作成され、アプリケーション・プログラムによっては作成されない。図 3-8. に、すべてのタイプのセグメント・ディスクリプタに使用される一般的なディスクリプタのフォーマットを示す。

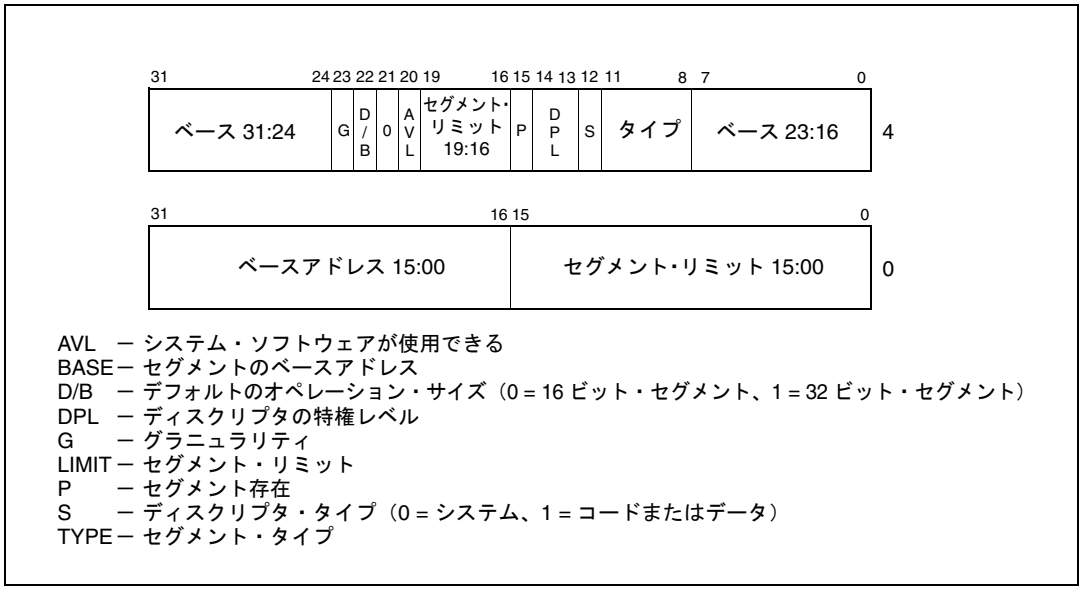


図 3-8. セグメント・ディスクリプタ

セグメント・ディスクリプタにあるフラグとフィールドは、次のとおりである。

セグメント・リミット・フィールド

セグメントのサイズを指定する。プロセッサは、2 つのセグメント・リミット・フィールドを一緒にして 20 ビット値を形成する。プロセッサは、G（グラニュラリティ）フラグの設定に応じて、次の 2 つの方法のうちの 1 つでセグメント・リミットを解釈する。

- グラニュラリティ・フラグがクリアされていると、セグメント・サイズは、1 バイト単位で 1 バイトから 1M バイトまでの範囲に及ぶ。

- グラニュラリティ・フラグがセットされていると、セグメント・サイズは、4K バイト単位で 4K バイトから 4G バイトまでの範囲に及ぶ。

プロセッサは、セグメントがエクスパンドアップ・セグメントであるかまたはエクスパンドダウン・セグメントであるかに応じて、2 つの異なる方法でセグメント・リミットを使用する。セグメント・タイプの詳細については、3.4.3.1. 項「セグメント・ディスクリプタのタイプ - コードとデータ」を参照。エクスパンドアップ・セグメントでは、論理アドレスのオフセットは、0 からセグメント・リミットまでの範囲で指定できる。オフセットをセグメント・リミットより大きくすると、一般保護例外（#GP）が発生する。エクスパンドダウン・セグメントでは、セグメント・リミットは反対の機能を持ち、オフセットは、B フラグの設定に応じて、セグメント・リミットから FFFFFFFFH または FFFFH までの範囲になる。オフセットをセグメント・リミットより小さくすると、一般保護例外が発生する。エクスパンドダウン・セグメントのセグメント・リミット・フィールドの値を小さくすると、新しいメモリがセグメントのアドレス空間の先頭ではなく最後に割り当てられる。IA-32 アーキテクチャのスタックは常に下方に伸長する。このメカニズムを使用すると、拡張可能なスタックにとって便利である。

ベース・アドレス・フィールド

4G バイトのリニアアドレス空間内のセグメントのバイト 0 の位置を定義する。プロセッサは、3 つのベース・アドレス・フィールドを一緒にして 1 つの 32 ビット値を形成する。セグメント・ベース・アドレスは、16 バイト境界にアライメントを合わせなければならない。16 バイトの境界にアライメントを合わせることは必須ではないが、プログラムは、コードとデータを16バイト境界にアライメントを合わせることによって最大の処理能力を出せる。

タイプ・フィールド

セグメントまたはゲートのタイプを指示し、セグメントに行うことができるアクセスの種類と伸長の方向を指定する。このフィールドの解釈は、ディスクリプタ・タイプ・フラグがアプリケーション（コードまたはデータ）ディスクリプタかシステム・ディスクリプタのどちらを指定しているかによって変わる。タイプ・フィールドのエンコーディングは、コード・ディスクリプタ、データ・ディスクリプタ、およびシステム・ディスクリプタに応じて異なる（図 4-1. を参照）。コードとデータのセグメント・タイプを指定するためにこのフィールドをどのように指定するかの説明については、3.4.3.1. 項「セグメント・ディスクリプタのタイプ - コードとデータ」を参照。

S（ディスクリプタ・タイプ）フラグ

セグメント・ディスクリプタがシステム・セグメント用（S フラグがクリア）か、コードまたはデータ・セグメント用（S フラグがセット）かを指定する。

DPL（ディスクリプタ特権レベル）フィールド

セグメントの特権レベルを指定する。特権レベルは 0 ～ 3 の範囲で指定でき、0 が最高の特権レベルである。DPL は、セグメントへのアクセスの制御に使用される。DPL と実行しているコード・セグメントの CPL との関係およびセグメント・セレクタの RPL の説明については、4.5. 節「特権レベル」を参照。

P（セグメント存在）フラグ

セグメントがメモリ内に存在しているか（セット）または存在していないか（クリア）を示す。このフラグがクリアされていると、セグメント・ディスクリプタを指しているセグメント・セレクタがセグメント・レジスタにロードされると、プロセッサはセグメント不在例外（#NP）を生成する。メモリ管理ソフトウェアは、このフラグを使用して、所与の時点でどのセグメントを実際に物理メモリにロードするかを制御できる。これは、仮想メモリを管理するためのページングに加えた制御を提供する。

図 3-9. に、セグメント存在フラグがクリアされているときのセグメント・ディスクリプタのフォーマットを示す。このフラグがクリアされていると、オペレーティング・システムまたはエグゼクティブは、使われなくなったセグメントがある位置に関する情報など独自のデータをストアするために、「使用可能」とマークされている位置を自由に使用できる。

D/B（デフォルトのオペレーション・サイズ / デフォルトのスタック・ポインタ・サイズまたは上限あるいはその両方）フラグ

セグメント・ディスクリプタが実行可能なコード・セグメントであるか、エクスパンドダウン・データ・セグメントであるか、またはスタック・セグメントであるかに応じて、異なる機能を実行する。（このフラグは、常に 32 ビットのコード・セグメントとデータ・セグメントに対しては 1 に設定し、16 ビットのコード・セグメントとデータ・セグメントに対しては 0 に設定する必要がある。）

- **実行可能なコード・セグメント。**このフラグは、D フラグといい、セグメント内の命令によって参照される実効アドレスとオペランドのデフォルトの長さを示す。このフラグがセットされていると、32 ビットのアドレスと 32 ビットまたは 8 ビットのオペランドが想定される。クリアされていると、16 ビットのアドレスと 16 ビットまたは 8 ビットのオペランドが想定される。

命令プリフィックス 66H を使用して、デフォルト以外のオペランド・サイズを選択でき、プリフィックス 67H を使用して、デフォルト以外のアドレスサイズを選択できる。

- **スタック・セグメント（SS レジスタによって指されているデータ・セグメント）。**このフラグは、B（ビッグ）フラグといい、（プッシュ、ポップ、コールなど）暗黙のスタック操作に使用されるスタックポインタのサイズを指定する。このフラグがセットされていると、32 ビットの ESP レジスタにストアされる 32 ビットのスタックポインタが使用される。クリアされていると、16 ビットの SP レジスタにストアされる 16 ビットのスタックポインタが使用される。スタック・セグメントが（次のパラグラフで説明する）エクスパンドダウン・データ・セグメントであるようにセットアップされていると、B フラグはスタック・セグメントの上限も指定する。
- **エクスパンドダウン・データ・セグメント。**このフラグは、B フラグといい、セグメントの上限を指定する。このフラグがセットされていると、上限は FFFFFFFFH（4G バイト）である。クリアされていると、上限は FFFFH（64K バイト）である。

図 3-9. セグメント存在フラグがクリアされているときのセグメント・ディスクリプタ

G（グラニュラリティ）フラグ

セグメント・リミット・フィールドのスケーリングを決定する。グラニュラリティ・フラグがクリアされていると、セグメント・リミットはバイト単位で解釈される。セットされていると、セグメント・リミットは 4K バイト単位で解釈される（このフラグは、常にバイト・グラニュラであるベースアドレスのグラニュラリティには影響を与えない）。グラニュラリティ・フラグがセットされていると、オフセットをセグメント・リミットに対してチェックするときに、オフセットの下位 12 ビットはテストされない。例えば、グラニュラリティ・フラグがセットされている場合にリミットを 0 にすると、有効なオフセットは 0 から 4095 までとなる。

使用可能ビットと予約ビット

セグメント・ディスクリプタの第二ダブルワードのビット 20 は、システ

ム・ソフトウェアで使用できる。ビット 21 は、予約されており、常に 0 に設定する必要がある。

3.4.3.1. セグメント・ディスクリプタのタイプ - コードとデータ

セグメント・ディスクリプタの S（ディスクリプタ・タイプ）フラグがセットされていると、ディスクリプタはコード・セグメント用またはデータ・セグメント用である。その場合には、タイプ・フィールドの最上位ビット（セグメント・ディスクリプタの第二ダブルワードのビット 11）の状態によって、ディスクリプタがデータ・セグメント用（クリア）であるかまたはコード・セグメント用（セット）であるかが決定される。

データ・セグメント用の場合には、タイプ・フィールドの下位 3 ビット（ビット 8、9、10）は、アクセス済み（A）、書き込み可能（W）、伸長方向（E）として解釈される。コード・セグメントとデータ・セグメントのタイプ・フィールドにあるビットのエンコーディングの説明については、表 3-1. を参照。データ・セグメントは、書き込み可能ビットの設定に応じて、読み取り専用セグメントまたは読み取り / 書き込みセグメントのどちらにでもできる。

表 3-1. コードタイプとデータタイプのセグメント

タイプ・フィールド					ディスクリプタ・タイプ	説明
10 進	11	10 E	9 W	8 A		
0	0	0	0	0	データ	読み取り専用
1	0	0	0	1	データ	読み取り専用、アクセス
2	0	0	1	0	データ	読み取り / 書き込み
3	0	0	1	1	データ	読み取り / 書き込み、アクセス
4	0	1	0	0	データ	読み取り専用、エクスパンドダウン
5	0	1	0	1	データ	読み取り専用、エクスパンドダウン、アクセス
6	0	1	1	0	データ	読み取り / 書き込み、エクスパンドダウン
7	0	1	1	1	データ	読み取り / 書き込み、エクスパンドダウン、アクセス
		C	R	A		
8	1	0	0	0	コード	実行専用
9	1	0	0	1	コード	実行専用、アクセス
10	1	0	1	0	コード	実行 / 読み取り
11	1	0	1	1	コード	実行 / 読み取り、アクセス
12	1	1	0	0	コード	実行専用、コンフォーミング
13	1	1	0	1	コード	実行専用、コンフォーミング、アクセス
14	1	1	1	0	コード	実行 / 読み取り専用、コンフォーミング
15	1	1	1	1	コード	実行 / 読み取り専用、コンフォーミング、アクセス

スタック・セグメントは、読み取り / 書き込みセグメントでなければならないデータ・セグメントである。書き込み可能ではないデータ・セグメントのセグメント・セレクタを SS レジスタにロードすると、一般保護例外（#GP）が発生する。スタック・セグメントのサイズを動的に変更する必要がある場合には、スタック・セグメントをエクスパンドダウン・データ・セグメント（伸長方向フラグをセット）にできる。この場合に、セグメント・リミットを動的に変更すると、スタックの一番下にスタック空間

が追加される。スタック・セグメントのサイズを静的なままにする場合は、スタック・セグメントは、エクスパンドアップ・タイプまたはエクスパンドダウン・タイプのどちらであってもよい。

アクセス済みビットは、オペレーティング・システムまたはエグゼクティブが最後にこのビットをクリアして以降に、セグメントがアクセスされたかどうかを示す。プロセッサは、セグメントのセグメント・セレクタをセグメント・レジスタにロードすると、セグメント・ディスクリプタを含むメモリのタイプがプロセッサの書き込みをサポートすると仮定して、このビットをセットする。このビットは、明示的にクリアされるまでセットされたままである。このビットは、仮想メモリ管理とデバッグの両方に使用できる。

コード・セグメント用の場合には、タイプ・フィールドの下位3ビットは、アクセス済み（A）、読み取り可能（R）、コンフォーミング（C）とし解釈される。コード・セグメントは、読み取り可能ビットの設定に応じて、実行専用または実行 / 読み取りのどちらにでもできる。定数またはその他の静的データがROMに命令コードと一緒に置かれている場合は、実行 / 読み取りセグメントを使用することができる。この場合には、CSオーバーライド・プリフィックスを付けて命令を使用するか、またはデータ・セグメント・レジスタ（DS、ES、FS、またはGSの各レジスタ）にコード・セグメントのセグメント・セレクタをロードして、データをコード・セグメントから読み取ることができる。保護モードでは、コード・セグメントは書き込み可能ではない。

コード・セグメントは、コンフォーミングまたは非コンフォーミングのどちらであってもよい。より特権レベルの高いコンフォーミング・セグメントへ実行権を譲渡しても、実行は現在の特権レベルで継続される。異なる特権レベルの非コンフォーミング・セグメントへの譲渡は、コールゲートまたはタスクゲートが使用されていない限り一般保護例外（#GP）となる（コンフォーミングおよび非コンフォーミングのコード・セグメントの詳細については、4.8.1. 項「コード・セグメントへの直接呼び出しまたはジャンプ」を参照）。保護機能と一部のタイプの例外（除算エラーやオーバーフローなど）のハンドラにアクセスしないシステム・ユーティリティは、コンフォーミング・コード・セグメントにロードすることができる。より低い特権レベルのプログラムやプロシージャから保護する必要があるユーティリティは、非コンフォーミング・コード・セグメントに置く必要がある。

注記

ターゲット・セグメントがコンフォーミングまたは非コンフォーミングのどちらのコード・セグメントであるかに関係なく、コールまたはジャンプによってより低い特権レベルの（数値としてより大きい特権レベルの）コード・セグメントに実行権を譲渡できない。そのような実行権譲渡を行おうとすると、一般保護例外が発生する。

すべてのデータ・セグメントは非コンフォーミングであり、これは、より低い特権レベルのプログラムまたはプロシージャ（数値としてより大きい特権レベルで実行しているコード）はそれらのセグメントをアクセスできないことを意味している。ただし、コード・セグメントとは異なり、データ・セグメントは、特別なアクセスゲートを使用しなくても、より高い特権レベルのプログラムまたはプロシージャ（数値としてより小さい特権レベルで実行しているコード）によってアクセスできる。

GDT または LDT 内のセグメント・ディスクリプタが ROM に格納される場合、ソフトウェアまたはプロセッサが ROM ベースのセグメント・ディスクリプタに対して更新（書き込み）しようとすると、プロセッサが無限ループに入ることがある。この問題を回避するには、ROM に格納されているすべてのセグメント・ディスクリプタのアクセスビットをセットし、ROM 内のセグメント・ディスクリプタを変更しようとするオペレーティング・システムやエグゼクティブのコードを除去すればよい。

3.5. システム・ディスクリプタ・タイプ

セグメント・ディスクリプタの S（ディスクリプタ・タイプ）フラグがクリアされていると、ディスクリプタ・タイプはシステム・ディスクリプタである。プロセッサは、次のタイプのシステム・ディスクリプタを認識する。

- ローカル・ディスクリプタ・テーブル（LDT）セグメント・ディスクリプタ
- タスク・ステート・セグメント（TSS）ディスクリプタ
- コール・ゲート・ディスクリプタ
- 割り込みゲート・ディスクリプタ
- トラップ・ゲート・ディスクリプタ
- タスク・ゲート・ディスクリプタ

これらのディスクリプタは、システム・セグメント・ディスクリプタとゲート・ディスクリプタという 2 つのカテゴリのどちらかに当てはまる。システム・セグメント・ディスクリプタは、システム・セグメント（LDT セグメントと TSS セグメント）を指す。ゲート・ディスクリプタは、それ自体が「ゲート」であり、コード・セグメントのプロシージャ・エントリ・ポイントへのポインタを保持する（コールゲート、割り込みゲート、トラップゲート）か、または TSS のセグメント・セレクタを保持する（タスクゲート）。表 3-2. に、システム・セグメント・ディスクリプタとゲート・ディスクリプタのタイプ・フィールドのエンコーディングを示す。

表 3-2. システム・セグメント・タイプとゲート・ディスクリプタ・タイプ

タイプ・フィールド					説明
10 進	11	10	9	8	
0	0	0	0	0	予約済み
1	0	0	0	1	16 ビット TSS（使用可能）
2	0	0	1	0	LDT
3	0	0	1	1	16 ビット TSS（ビジー）
4	0	1	0	0	16 ビット・コール・ゲート
5	0	1	0	1	タスクゲート
6	0	1	1	0	16 ビット割り込みゲート
7	0	1	1	1	16 ビット・トラップ・ゲート
8	1	0	0	0	予約済み
9	1	0	0	1	32 ビット TSS（使用可能）
10	1	0	1	0	予約済み
11	1	0	1	1	32 ビット TSS（ビジー）
12	1	1	0	0	32 ビット・コール・ゲート
13	1	1	0	1	予約済み
14	1	1	1	0	32 ビット割り込みゲート
15	1	1	1	1	32 ビット・トラップ・ゲート

システム・セグメント・ディスクリプタの詳細については、3.5.1. 項「セグメント・ディスクリプタ・テーブル」と 6.2.2. 項「TSS ディスクリプタ」を参照。ゲート・ディスクリプタの詳細については、4.8.3. 項「コールゲート」、5.11. 節「IDT ディスクリプタ」、6.2.4. 項「タスク・ゲート・ディスクリプタ」を参照。

3.5.1. セグメント・ディスクリプタ・テーブル

セグメント・ディスクリプタ・テーブルは、セグメント・ディスクリプタの配列である（図 3-10. を参照）。ディスクリプタ・テーブルは、可変長であり、8192（2^{13}）までの 8 バイトのディスクリプタを入れることができる。ディスクリプタ・テーブルには次の 2 種類がある。

- グローバル・ディスクリプタ・テーブル（GDT）
- ローカル・ディスクリプタ・テーブル（LDT）

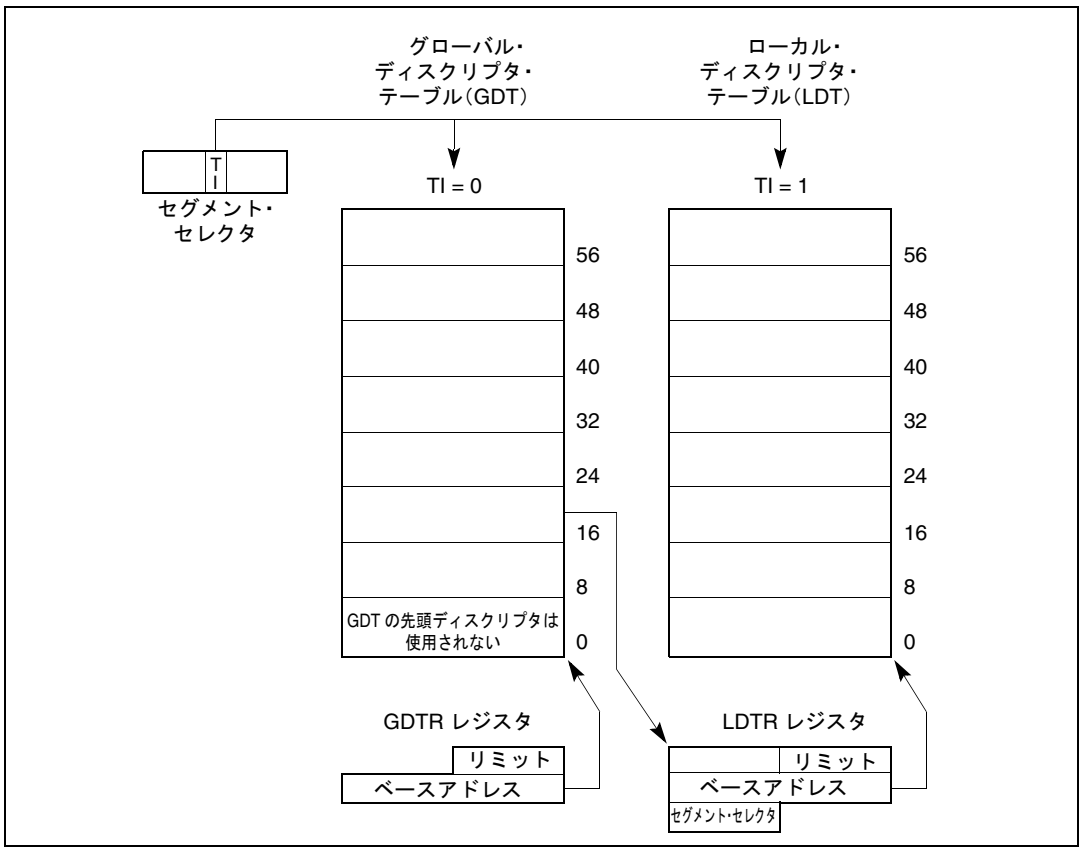


図 3-10. グローバル・ディスクリプタ・テーブルとローカル・ディスクリプタ・テーブル

各システムには1つのGDTが定義されていなければならず、これは、システムにあるすべてのプログラムとタスクに使用できる。オプションで、1つ以上のLDTを定義できる。例えば、実行する個別のタスクごとにLDTを定義できる。あるいは、一部またはすべてのタスクが同じLDTを共有できる。

GDTは、セグメントそのものではなく、リニアアドレス空間にあるデータ構造である。GDTのベース・リニア・アドレスとリミットは、GDTRレジスタにロードしなければならない（2.4.節「メモリ管理レジスタ」を参照）。最高のプロセッサ処理能力を引き出すには、GDTのリニアアドレスのアライメントを8バイト境界に合わせる必要がある。GDTのリミット値は、バイト単位で表される。セグメントの場合と同じように、リミット値をベースアドレスに加算すると最後の有効バイトのアドレスが得られる。リミットが0の場合には、有効なバイトは1つしかない。セグメント・ディスクリプタは常に8バイト長であるため、GDTリミットは常に8の整数倍より1だけ小さい値（つまり、8N-1）でなければならない。

プロセッサは、GDTの先頭ディスクリプタを使用しない。この「ヌル・ディスクリプタ」へのセグメント・セレクタは、データ・セグメント・レジスタ（DS、ES、FS、またはGS）にロードされたときに例外を生成しないが、そのディスクリプタを使用してメモリにアクセスしようとすると、常にこのセレクタは一般保護例外（#GP）を生成する。このセグメント・セレクタでセグメント・レジスタを初期化すると、未使用のセグメント・レジスタを誤って参照すると確実に例外が発生するようにできる。

LDTは、LDTタイプのシステム・セグメント内にある。GDTには、LDTセグメントのセグメント・ディスクリプタが入っていなければならない。システムが複数のLDTをサポートしている場合は、各LDTの個別のセグメント・セレクタとセグメント・ディスクリプタがGDTの中に入っていなければならない。LDTのセグメント・ディスクリプタは、GDTの任意の場所に置くことができる。LDTセグメント・ディスクリプタ・タイプの詳細については、3.5.節「システム・ディスクリプタ・タイプ」を参照。

LDTは、そのセグメント・セレクタを使用してアクセスされる。LDTにアクセスするときにアドレス変換の必要をなくすため、LDTのセグメント・セレクタ、ベース・リニア・アドレス、リミット、アクセス権はLDTRレジスタにストアされる（2.4.節「メモリ管理レジスタ」を参照）。

GDTRを（SGDT命令を使用して）ストアすると、48ビットの「疑似ディスクリプタ」がメモリにストアされる（図3-11.を参照）。ユーザモード（特権レベル3）でのアライメント・チェック・フォルトを避けるには、疑似ディスクリプタを奇数ワードアドレス（つまり、アドレスMOD 4が2に等しい）に置く必要がある。こうすると、プロセッサは、アライメントの合ったワード、その後にアライメントの合ったダブルワードをストアする。ユーザモードのプログラムは、通常は疑似ディスクリプタをストアしないが、このようにして疑似ディスクリプタのアライメントを合わせると、アライメント・チェック・フォルトが発生する可能性を回避できる。SIDT命令を使用してIDTRレジスタをストアするときにも、同じアライメントを使用する必要がある。（それぞれSLTR命令またはSTR命令を使用して）LDTRレジスタまたはタスクレジスタをストアするときは、疑似ディスクリプタをダブルワード・アドレス（つまり、アドレスMOD 4が0に等しい）に置く必要がある。

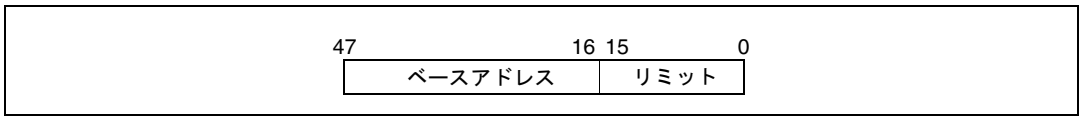


図 3-11. 疑似ディスクリプタのフォーマット

3.6. ページング（仮想メモリ）の概要

IA-32 アーキテクチャは、保護モードで動作するときは、リニアアドレス空間を大きい物理メモリ（例えば、4G バイトの RAM）に直接にマッピングもでき、またより小さい物理メモリとディスク記憶領域に（ページングを使用して）間接的にマッピングもできる。リニアアドレス空間をマッピングする後者の方法を一般的には仮想メモリまたはデマンドページ仮想メモリという。

ページングを使用すると、プロセッサは、物理メモリやディスク記憶領域にマッピングできる固定サイズのページ（4K バイト、2M バイト、または 4M バイトの長さ）にリニアアドレス空間を分割する。プログラム（またはタスク）がメモリ内の論理アドレスを参照すると、プロセッサは、そのアドレスをリニアアドレスに変換してから、そのページング・メカニズムを使用してリニアアドレスを対応する物理アドレスに変換する。

リニアアドレスを含むページがその時点で物理メモリにないと、プロセッサはページフォルト例外（#PF）を生成する。ページフォルト例外の例外ハンドラは、通常はそのページをディスク記憶領域から物理メモリにロードするようにオペレーティング・システムまたはエグゼクティブに指示する（恐らくそのプロセスで別のページを物理メモリからディスクに書き出す）。ページが物理メモリにロードされると、例外ハンドラから戻って、例外を発生させた命令が再開される。プロセッサがリニアアドレスを物理アドレス空間にマッピングするために、また（必要なときには）ページフォルト例外を生成するために使用する情報は、メモリにストアされているページ・ディスクリプタとページテーブルに入っている。

ページングは、固定サイズのページを使用する点でセグメンテーションとは異なる。通常は、それらが保持するコードまたはデータ構造と同じサイズであるセグメントとは異なり、ページのサイズは固定されている。アドレス変換の形態としてセグメンテーションしか使用しない場合は、物理メモリに存在するデータ構造はメモリ内にそのすべての部分があることになる。ページングを使用する場合は、データ構造は、一部はメモリ、一部はディスク記憶領域に入れることもできる。

アドレス変換に必要なバスサイクル数を最小にするため、直前にアクセスされたページ・ディレクトリとページテーブルのエントリを、トランスレーション・ルックアサイド・バッファ（TLB）というプロセッサにあるデバイス内にキャッシュする。TLB は、バスサイクルを要求することなく現在のページ・ディレクトリとページテーブルを読み取るためのほとんどの要求を満たす。TLB にページ・テーブル・エントリが入っていないときだけに余分のバスサイクルが発生する。これは、通常はページが長い間アクセスされていないときに起こる。TLB の詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照。

3.6.1. ページングのオプション

ページングは、プロセッサの制御レジスタの次の3つのフラグによって制御される。

- **ページング（PG: Paging）・フラグ。**CR0のビット31（Intel386™プロセッサ以降のすべてのIA-32プロセッサで使用可能）
- **ページサイズ拡張（PSE: Paging Size Extensions）フラグ。**CR4のビット4（インテル® Pentium® プロセッサで導入された）
- **物理アドレス拡張（PAE: Physical Address Extention）フラグ。**CR4のビット5（インテル® Pentium® Proプロセッサで導入された）

PGフラグは、ページ変換メカニズムをイネーブルにする。オペレーティング・システムまたはエグゼクティブは、通常はプロセッサ初期化中にこのフラグをセットする。プロセッサのページ変換メカニズムを使用してデマンドページ仮想メモリシステムを使用する場合、または複数のプログラム（またはタスク）を仮想8086モードで実行するようにオペレーティング・システムが設計されている場合は、PGフラグをセットしなければならない。

PSEフラグは4Mバイトのページサイズをイネーブルにする（PAEフラグをセットすると2Mバイト・ページとなる）。PSEフラグがクリアされていると、4Kバイトのより一般的なページ長が使用される。PSEフラグの使用法の詳細については、3.7.2. 項「リニアアドレス変換（4Mバイト・ページ）」、3.8.2. 項「PAEがイネーブルの場合のリニアアドレス変換（2Mバイト・ページ）」、3.9. 節「PSE-36ページング・メカニズムを使用した36ビット物理アドレス指定」を参照。

PAEフラグによって、物理アドレスを36ビットに拡張する方法が提供される。この物理アドレス拡張は、ページングがイネーブルになっているときだけに使用できる。これは、ページ・ディレクトリとページテーブルを使用してFFFFFFFFHより上の物理アドレスを参照する追加のページ・ディレクトリ・ポインタ・テーブルによる。PAEフラグを使用した物理アドレス拡張の詳細については、3.8. 節「PAEページング・メカニズムを使用した36ビット 物理アドレス指定」を参照。

36ビット・ページ・サイズ拡張（PSE-36）機能を使用すると、別の方法で物理アドレス指定を36ビットに拡張できる。このページング・メカニズムでは、ページサイズ拡張モード（PSEフラグでイネーブルにされる）と変更ページ・ディレクトリ・エントリを使用して、FFFFFFFFHより上の物理アドレスを参照する。PSE-36機能フラグ（ソース・オペランドに1をセットしてCPUID命令を実行したときの、EDXレジスタのビット17）は、このアドレス指定メカニズムが使用できるかどうかを示す。PSE-36物理アドレス拡張とページサイズ拡張メカニズムの詳細については、3.9. 節「PSE-36ページング・メカニズムを使用した36ビット物理アドレス指定」を参照のこと。

3.6.2. ページテーブルとページ・ディレクトリ

プロセッサが（ページングがイネーブルになっているときに）リニアアドレスを物理アドレスに変換するために使用する情報は、次の4つのデータ構造に入っている。

- ページ・ディレクトリ ― 4Kバイト・ページに入っている32ビットのページ・ディレクトリ・エントリ（PDE）の配列。1024個までのページ・ディレクトリ・エントリをページ・ディレクトリに保持できる。
- ページテーブル ― 4Kバイト・ページに入っている32ビットのページ・テーブル・エントリ（PTE）の配列。1024個までのページ・テーブル・エントリをページテーブルに保持することができる。（ページテーブルは、2Mバイト・ページまたは4Mバイト・ページには使用されない。これらのページサイズは、1つ以上のページ・ディレクトリ・エントリから直接にマッピングされる。）
- ページ ― 4Kバイト、2Mバイト、または4Mバイトのフラットなアドレス空間
- ページ・ディレクトリ・ポインタ・テーブル ― 4つの64ビット・エントリの配列。そのおのおのがページ・ディレクトリを指す。このデータ構造は、物理アドレス拡張がイネーブルになっているときだけに使用される（3.8.節「PAEページング・メカニズムを使用した36ビット 物理アドレス指定」を参照）。

これらのテーブルによって、通常の32ビット物理アドレス指定が使用されているときには4Kバイトまたは4Mバイトのページ、また拡張（36ビット）物理アドレス指定が使用されているときにのみ4Kバイト、2Mバイトまたは4Mバイトのページにアクセスできるようになる。表3-3.に、ページング制御フラグとPSE-36 CPUID機能フラグのさまざまな設定から得られるページサイズと物理アドレスサイズを示す。各ページ・ディレクトリ・エントリには、PS（ページサイズ）フラグが含まれる。PSフラグは、エントリがページテーブルを指し、テーブルのエントリが4Kバイト・ページを指すか（PSを0に設定）、またはページ・ディレクトリ・エントリが4Mバイト（PSEおよびPSを1にセット）または2Mバイトのページを直接に指すか（PAEおよびPSを1にセット）を指定する。

3.7. 32ビット物理アドレス指定を使用したページ変換

以降の各項では、32ビット物理アドレスと4Gバイトの最大物理アドレス空間を使用する場合の、IA-32アーキテクチャのページ変換メカニズムを説明する。3.8.節「PAEページング・メカニズムを使用した36ビット 物理アドレス指定」および3.9.節「PSE-36ページング・メカニズムを使用した36ビット物理アドレス指定」では、このページ変換メカニズムの拡張機能について説明している。この拡張機能により、36ビットの物理アドレスと64Gバイトの最大物理アドレス空間をサポートできる。

表 3-3. ページサイズと物理アドレスサイズ

PG フラグ、CR0	PAE フラグ、CR4	PSE フラグ、CR4	PS フラグ、PDE	PSE-36 CPUID 機能フラグ	ページサイズ	物理アドレスサイズ
0	X	X	X	X	—	ページングはディスエーブル
1	0	0	X	X	4K バイト	32 ビット
1	0	1	0	X	4K バイト	32 ビット
1	0	1	1	0	4M バイト	32 ビット
1	0	1	1	1	4M バイト	36 ビット
1	1	X	0	X	4K バイト	36 ビット
1	1	X	1	X	2M バイト	36 ビット

3.7.1. リニアアドレス変換（4K バイト・ページ）

図 3-12. に、リニアアドレスを 4K バイト・ページにマッピングするときのページ・ディレクトリとページテーブルの階層を示す。ページ・ディレクトリのエントリはページテーブルを指し、ページテーブルのエントリは物理メモリのページを指している。このページング方式を使用して、2^{20} ページまでアドレス指定でき、これは 2^{32} バイト（4G バイト）のリニアアドレス空間を占める。

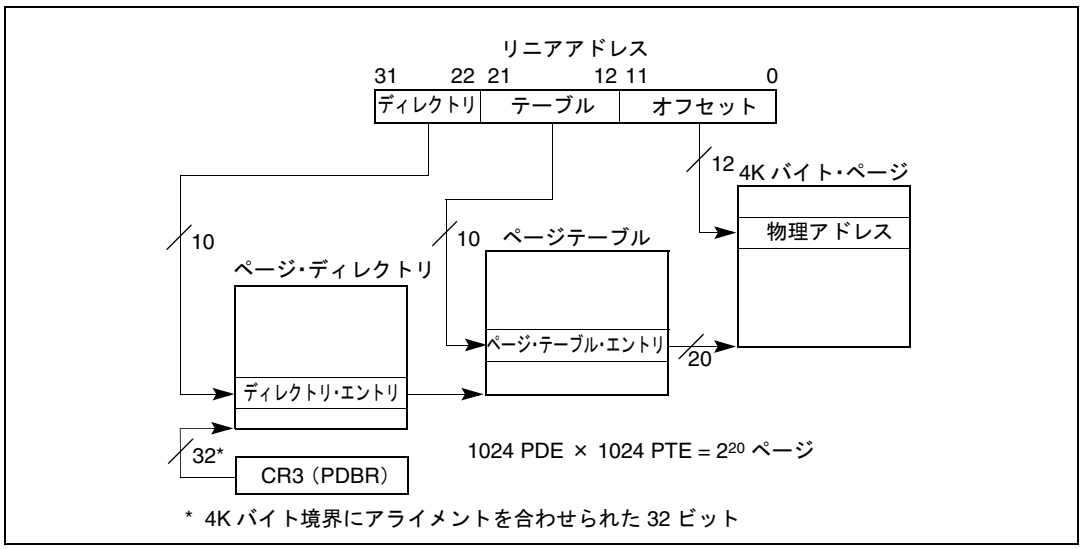


図 3-12. リニアアドレス変換（4K バイト・ページ）

さまざまなテーブルエントリを選択するため、リニアアドレスは次の 3 つのセクションに分割される。

- ページ・ディレクトリ・エントリ ー ビット 22 ～ 31 は、ページ・ディレクトリにあるエントリへのオフセットを示す。選択されているエントリは、ページテーブルのベース物理アドレスを示す。
- ページ・テーブル・エントリ ー ビット 12 ～ 21 は、選択されているページテーブルにあるエントリへのオフセットを示す。このエントリは、物理メモリにあるページのベース物理アドレスを示す。
- ページ・オフセット ー ビット 0 ～ 11 は、そのページにある物理アドレスへのオフセットを示す。

メモリ管理ソフトウェアには、すべてのプログラムとタスクに 1 つのページ・ディレクトリを使用するか、タスクごとに 1 つのページ・ディレクトリを使用するか、または 2 つを組み合わせて使用するオプションがある。

3.7.2. リニアアドレス変換（4M バイト・ページ）

図 3-12. に、ページ・ディレクトリを使用してリニアアドレスを 4M バイト・ページにどのようにマッピングできるかを示す。ページ・ディレクトリのエントリは、物理メモリ内の 4M バイト・ページを指す。このページング方式を使用して 1024 までのページを 4G バイト・リニア・アドレス空間にマッピングできる。

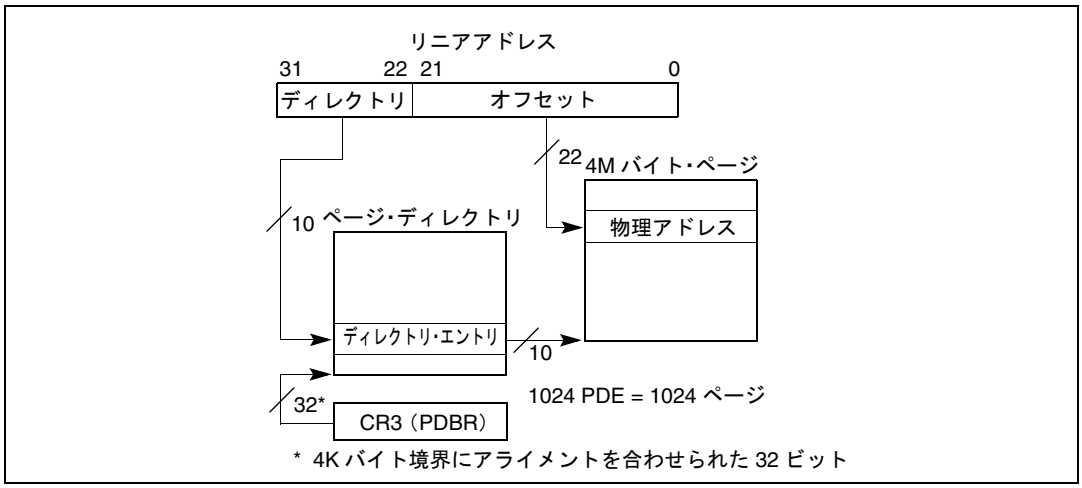


図 3-13. リニアアドレス変換（4M バイト・ページ）

4M バイトのページサイズを選択するには、制御レジスタ CR4 の PSE フラグをセットし、ページ・ディレクトリ・エントリのページサイズ（PS）フラグをセットする（図 3-14. を参照）。これらのフラグをセットすると、リニアアドレスは次の 2 つのセクションに分割される。

- ページ・ディレクトリ・エントリ ー ビット 22 ～ 31 は、ページ・ディレクトリにあるエントリへのオフセットを示す。選択されているエントリは、4M バイト・ページのベース物理アドレスを示す。
- ページ・オフセット ー ビット 0 ～ 21 は、そのページにある物理アドレスへのオフセットを示す。

注記

(インテル® Pentium® プロセッサに対してのみ) 大きいページサイズをイネーブルまたはディスエーブルにするときは、制御レジスタ CR4 の PSE フラグをセットまたはクリアした後に、TLB を無効化（フラッシュ）しなければならない。そうしないと、プロセッサが TLB に記憶されている古くなったページ変換情報を使用するために、間違ったページ変換が行われるおそれがある。TLB を無効化する方法の詳細については、10.9. 節「トランスレーション・ルックアサイド・バッファ（TLB）の無効化」を参照。

3.7.3. 4K バイト・ページと 4M バイト・ページの混在

CR4 の PSE フラグをセットすると、4M バイト・ページと 4K バイト・ページのページテーブルの両方を同じページ・ディレクトリからアクセスすることができる。PSE フラグをクリアすると、(ページ・ディレクトリ・エントリ内の PS フラグの設定に関係なく）4K バイト・ページのページテーブルしかアクセスできない。

4K バイト・ページと 4M バイト・ページを混在させる代表的な例として、オペレーティング・システムまたはエグゼクティブのカーネルを大きい 1 ページに置いて、TLB ミスを減少させ、したがってシステムの処理能力全体を向上させる方法がある。

プロセッサは、4M バイト・ページ・エントリと 4K バイト・ページ・エントリを別個の TLB に維持する。そのため、カーネルなど頻繁に使用されるコードを大きい 1 ページに置き、4K バイト・ページ TLB エントリをアプリケーション・プログラムとタスク向けに解放する。

3.7.4. メモリ別名定義

IA-32 アーキテクチャでは、共通のページ・テーブル・エントリを 2 つのページ・ディレクトリ・エントリが指せるようにすれば、メモリ別名定義を可能にしている。この方法でメモリ別名定義を実装するソフトウェアでは、ページ・ディレクトリ・エントリとページ・テーブル・エントリのアクセスビットとダーティビットの整合性を管理する必要がある。2 つのページ・ディレクトリ・エントリに対してアクセスビットとダーティビットの整合性が保たれないと、プロセッサがデッドロックすることがある。

3.7.5. ページ・ディレクトリのベースアドレス

現在のページ・ディレクトリの物理アドレスは、レジスタ CR3（ページ・ディレクトリ・ベース・レジスタまたは PDBR ともいう）にストアされている。（PDBR の詳細については、図 2-5. と 2.5. 節「制御レジスタ」を参照。）ページングを使用する場合は、（ページングをイネーブルにする前に）プロセッサ初期化プロセスの一部として、PDBR をロードしなければならない。その後は、MOV 命令で新しい値を CR3 にロードして明示的に、またはタスクスイッチの一部として暗黙のうちに、PDBR を変更することができる。（タスクにレジスタ CR3 の内容を設定する方法の説明については、6.2.1. 項「タスク・ステート・セグメント（TSS）」を参照）。

ページ・ディレクトリのための存在フラグは PDBR にはない。関連するタスクが中断されている間は、ページ・ディレクトリは存在していなくても（物理メモリからページアウトされていても）よいが、オペレーティング・システムは、タスクがディスパッチされる前に、そのタスクの TSS にある PDBR イメージによって示されるページ・ディレクトリが物理メモリに存在していることを保証しなければならない。また、ページ・ディレクトリは、タスクがアクティブである限りメモリに残っていなければならない。

3.7.6. ページ・ディレクトリ・エントリとページ・テーブル・エントリ

図 3-14. に、4K バイト・ページと 32 ビット物理アドレスを使用しているときのページ・ディレクトリ・エントリとページ・テーブル・エントリのフォーマットを示す。図 3-15. には、4M バイト・ページと 32 ビット物理アドレスを使用しているときのページ・ディレクトリ・エントリのフォーマットを示す。図 3-14. と図 3-15. のエントリにあるフラグとフィールドの機能は、次のとおりである。

ページのベースアドレス、ビット 12 ～ 32

（4K バイト・ページ用のページ・テーブル・エントリ）4K バイト・ページの先頭バイトの物理アドレスを指定する。このフィールドのビットは、物理アドレスの上位 20 ビットと解釈され、それがページのアライメントを強制的に 4K バイト境界に合わせる。

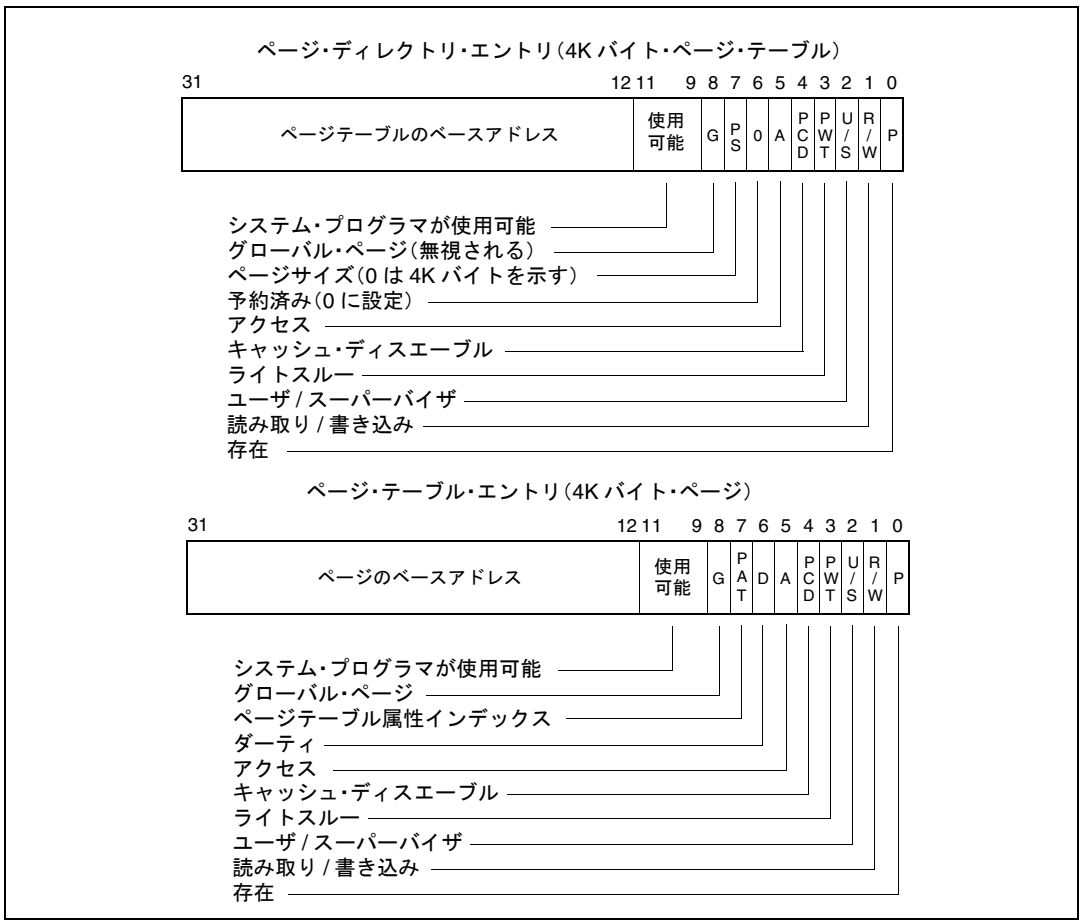


図 3-14. 4K バイト・ページと 32 ビット物理アドレスを使用する場合の
ページ・ディレクトリ・エントリとページ・テーブル・エントリのフォーマット

（4K バイト・ページ・テーブル用のページ・ディレクトリ・エントリ）ページテーブルの先頭バイトの物理アドレスを指定する。このフィールドのビットは、物理アドレスの上位 20 ビットと解釈され、それがページテーブルのアライメントを強制的に 4K バイト境界に合わせる。

（4M バイト・ページ用のページ・ディレクトリ・エントリ）4M バイト・ページの先頭バイトの物理アドレスを指定する。このフィールドのビット 22 ～ 31 だけが使用される（インテル® Pentium® II プロセッサまでの IA-32 プロセッサでは、ビット 12 ～ 21 は予約されており、0 に設定しなければならない）。ベース・アドレス・ビットは、物理アドレスの上位 10 ビットと解釈され、これが 4M バイト・ページのアライメントを強制的に 4M バイト境界に合わせる。

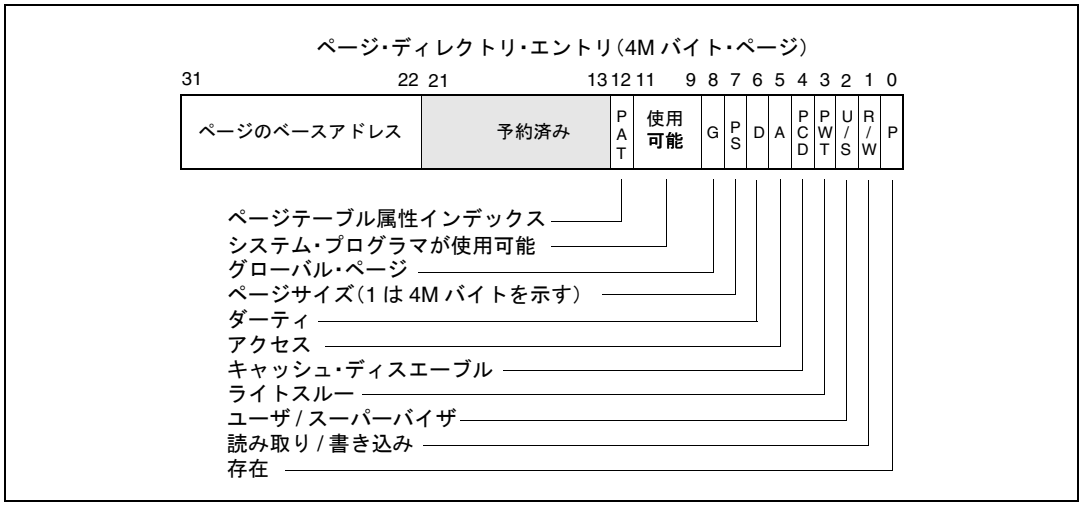


図 3-15. 4M バイト・ページと 32 ビット・アドレスを使用する場合の
ページ・ディレクトリ・エントリのフォーマット

存在（P）フラグ、ビット 0

エントリによって指されているページまたはページテーブルが物理メモリに現在ロードされているかどうかを示す。このフラグがセットされていると、ページは物理メモリにあり、アドレス変換が行われる。クリアされていると、ページはメモリにはなく、プロセッサがページにアクセスしようとすると、ページフォルト例外（#PF）が発生する。

プロセッサは、このフラグをセットしたりクリアしたりしない。フラグの状態を維持することは、オペレーティング・システムまたはエグゼクティブに任されている。

プロセッサがページフォルト例外を生成する場合は、オペレーティング・システムは、一般的に次の操作を実行する必要がある。

1. ページをディスク記憶領域から物理メモリにコピーする。
2. ページアドレスをページ・テーブル・エントリまたはページ・ディレクトリ・エントリにロードし、その存在フラグをセットする。ダーティフラグやアクセス済みフラグなどその他のビットもこの時点でセットすることがある。
3. TLB にある現在のページ・テーブル・エントリを無効化する（TLB およびそれらを無効化する方法の説明については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照）。
4. ページ・フォルト・ハンドラから戻って、割り込まれたプログラムまたはタスクを再開する。

読み取り / 書き込み（R/W）フラグ、ビット 1

ページまたは（ページテーブルを指しているページ・ディレクトリ・エントリの場合には）ページのグループの読み取り / 書き込み特権を指定する。このフラグがクリアされていると、ページは読み取り専用であり、セットされていると、ページの読み取りと書き込みを行うことができる。このフラグは、レジスタ CR0 の U/S フラグおよび WP フラグと相互作用する。これらのフラグの使用法の詳細については、4.11. 節「ページレベルの保護」と表 4-2. を参照。

ユーザ / スーパーバイザ（U/S）フラグ、ビット 2

ページまたは（ページテーブルを指しているページ・ディレクトリ・エントリの場合には）ページのグループのユーザ / スーパーバイザ特権を指定する。このフラグがクリアされていると、ページにはスーパーバイザ特権レベルが割り当てられている。セットされていると、ページにはユーザ特権レベルが割り当てられている。このフラグは、レジスタ CR0 の R/W フラグおよび WP フラグと相互作用する。これらのフラグの使用法の詳細については、4.11. 節「ページレベルの保護」と表 4-2. を参照。

ページ・レベル・ライトスルー（PWT）フラグ、ビット 3

個別ページまたはページテーブルのキャッシュ方式をライトスルーにするかライトバックにするかを制御する。PWT フラグがセットされていると、ライトスルー・キャッシングが関連するページまたはページテーブルに対してイネーブルになり、クリアされていると、ライトバック・キャッシングが関連するページまたはページテーブルに対してイネーブルになる。CR0 の CD（キャッシュ・ディスエーブル）フラグがセットされていると、プロセッサはこのフラグを無視する。このフラグの使用法の詳細については、10.5. 節「キャッシュの制御」を参照。制御レジスタ CR3 にある同類の PWT フラグの説明については、2.5. 節「制御レジスタ」を参照。

ページ・レベル・キャッシュ・ディスエーブル（PCD）フラグ、ビット 4

個別ページまたはページテーブルのキャッシングを制御する。PCD フラグがセットされていると、関連するページまたはページテーブルのキャッシングは禁止される。クリアされていると、ページまたはページテーブルをキャッシュできる。このフラグは、メモリマップド I/O ポートを含むページまたはキャッシングによる性能上の利益がないページに対してキャッシングをディスエーブルにできる。CR0 の CD（キャッシュ・ディスエーブル）フラグがセットされていると、プロセッサはこのフラグを無視する（セットされているとみなす）。このフラグの使用法の詳細については、第 10 章「メモリ・キャッシュ制御」を参照。制御レジスタ CR3 にあるコンパニオン PCD フラグの説明については、2.5. 節「制御レジスタ」を参照。

アクセス済み（A）フラグ、ビット 5

セットされていると、ページまたはページテーブルがすでにアクセス（読み取りまたは書き込み）されたかどうかを示す。メモリ管理ソフトウェアは、一般的にページまたはページテーブルが最初に物理メモリにロードされたときにこのフラグをクリアする。その後、プロセッサは、ページまたはページテーブルが初めてアクセスされたときにこのフラグをセットする。

このフラグは、「スティッキー」なフラグであり、これは、プロセッサはセットしてしまうと暗黙のうちにはクリアしないことを意味している。ソフトウェアだけがこのフラグをクリアできる。アクセス済みフラグとダーティフラグは、メモリ管理ソフトウェアが物理メモリとの間のページとページテーブルの転送の管理に使用するために備えられている。

注：プロセッサがこのビットをセットするアクセス操作は、プロセッサの自己修正コード検出ロジックによって検出されることも、検出されないこともある。プロセッサが、ページテーブル構造に使用されているのと同じメモリ領域からコードを実行している場合、このビットをセットすると、実行中のコード・ストリームに対する変更が直ちに起こることも、起こらないこともある。

ダーティ（D）フラグ、ビット 6

セットされていると、ページが書き込まれたかどうかを示す。（このフラグは、ページテーブルを指しているページ・ディレクトリ・エントリでは使用されない。）メモリ管理ソフトウェアは、一般的にページが最初に物理メモリにロードされたときにこのフラグをクリアする。その後、プロセッサは、ページが初めて書き込み操作のためにアクセスされたときにこのフラグをセットする。

このフラグは、「スティッキー」であり、これは、プロセッサはセットしてしまうと暗黙のうちにはクリアしないことを意味している。ソフトウェアだけがこのフラグをクリアすることができる。ダーティフラグとアクセス済みフラグは、メモリ管理ソフトウェアが物理メモリとの間のページとページテーブルの転送の管理に使用するために備えられている。

注：プロセッサがこのビットをセットするアクセス操作は、プロセッサの自己修正コード検出ロジックによって検出されることも、検出されないこともある。プロセッサが、ページテーブル構造に使用されているのと同じメモリ領域からコードを実行している場合、このビットをセットすると、実行中のコード・ストリームに対する変更が直ちに起こることも、起こらないこともある。

ページサイズ（PS）フラグ、ビット 7、4K バイト・ページのページ・ディレクトリ・エントリ

ページサイズを決定する。このフラグがクリアされていると、ページサイズは 4K バイトであり、ページ・ディレクトリ・エントリはページテーブルを指す。セットされていると、ページサイズは、通常の 32 ビットのアドレス指定では 4M バイト（および拡張物理アドレス指定がイネーブルになっている場合は 2M バイト）であり、ページ・ディレクトリ・エントリはページを指す。ページ・ディレクトリ・エントリがページテーブルを指している場合は、そのページテーブルに関連するすべてのページは、4K バイト・ページになる。

ページ属性テーブル・インデックス（PAT）フラグ、4K バイト・ページの場合のページ・テーブル・エントリのビット 7、4M バイト・ページの場合のページ・ディレクトリ・エントリのビット 12

（これは、インテル® Pentium® III プロセッサで導入された。）PAT エントリを選択する。ページ属性テーブル（PAT）をサポートするプロセッサの場合、このフラグは、PCD フラグおよび PWT フラグと共に PAT 内のエントリを選択するために使用される。これにより、そのページのメモリタイプが選択される（10.12. 節「ページ属性テーブル（PAT）」を参照）。PAT をサポートしないプロセッサの場合、このビットは予約されており、0 をセットする必要がある。

グローバル（G）フラグ、ビット 8

（インテル® Pentium® Pro プロセッサで導入された。）セットされていると、グローバル・ページを示す。ページがグローバルとマークされ、レジスタ CR4 のページ・グローバル・イネーブル（PGE）フラグがセットされていると、そのページのページ・テーブル・エントリまたはページ・ディレクトリ・エントリは、レジスタ CR3 がロードされるかまたはタスクスイッチが発生しても TLB で無効化されない。このフラグは、（カーネルやその他のオペレーティング・システムまたはエグゼクティブのコードを含むページなど）頻繁に使用されるページが TLB からフラッシュされるのを防ぐために用意されたものである。ソフトウェアだけがこのフラグをセットまたはクリアすることができる。ページテーブルを指しているページ・ディレクトリ・エントリの場合には、このフラグは無視され、ページのグローバル特性がページ・テーブル・エントリに設定される。このフラグの使用法の詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照。（このビットは、インテル® Pentium® プロセッサおよびそれ以前の IA-32 プロセッサでは予約されている。）

予約ビットとソフトウェア使用可能ビット

すべての IA-32 プロセッサでは、ビット 9、10、11 は、ソフトウェアで使用することができる。(存在ビットがクリアされていると、ビット 1 ～ 31 をソフトウェアで使用することができる。図 3-16. を参照。) ページテーブルを指しているページ・ディレクトリ・エントリでは、ビット 6 が予約されており、0 に設定する必要がある。制御レジスタ CR4 の PSE フラグと PAE フラグがセットされていると、プロセッサは、予約ビットが 0 に設定されていなければページフォルトを生成する。

インテル® Pentium® II プロセッサとそれ以前のプロセッサでは、ページ・テーブル・エントリのビット 7 が予約されており、0 に設定する必要がある。4M バイト・ページ用のページ・ディレクトリ・エントリの場合には、ビット 12 ～ 21 が予約されており、0 に設定する必要がある。

インテル Pentium III プロセッサとそれ以降のプロセッサでは、4M バイト・ページ用のページ・ディレクトリ・エントリの場合には、ビット 13 ～ 21 が予約されており、0 に設定する必要がある。

3.7.7. 存在していないページ・ディレクトリ・エントリとページ・テーブル・エントリ

ページ・テーブル・エントリまたはページ・ディレクトリ・エントリの存在フラグがクリアされているときは、オペレーティング・システムまたはエグゼクティブは、エントリの残りの部分を使用してディスク記憶領域システムにあるページの位置などの情報をストアすることがある（図3-16. を参照)。

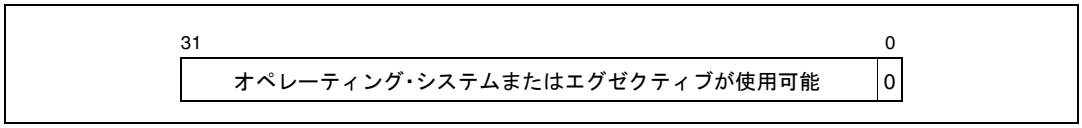


図 3-16. 存在していないページのページ・テーブル・エントリまたは
ページ・ディレクトリ・エントリのフォーマット

3.8. PAE ページング・メカニズムを使用した 36 ビット物理アドレス指定

PAE ページング・メカニズムと 36 ビット物理アドレス指定のサポートは、インテル® Pentium® Pro プロセッサで IA-32 アーキテクチャに導入された。IA-32 プロセッサにこの機能が実装されているかどうかは、CPUID 機能フラグの PAE（ソース・オペランドに 2 をセットして CPUID 命令を実行したときの、EDX レジスタのビット 6）で示される。レジスタ CR4 の物理アドレス拡張（PAE）フラグは、PAE メカニズムをイネーブルにすると共に、物理アドレスを 32 ビットから 36 ビットに拡張する。プロセッサには、追加のアドレスビットを収容するための 4 本の追加アドレス・ライン・ピンが備えられている。このオプションを使用するには、以下のフラグをセットする必要がある。

- 制御レジスタ CR0 の PG フラグ（ビット 31）－ ページングをイネーブルにする。
- 制御レジスタ CR4 の PAE フラグ（ビット 5）－ PAE ページング・メカニズムをイネーブルにする。

PAE ページング・メカニズムがイネーブルになっていると、プロセッサは、2 つのサイズのページ、つまり、4K バイトおよび 2M バイトを許容する。32 ビットのアドレス指定の場合と同じように、両方のページサイズとも同じページング・テーブルのセット内でアドレス指定ができる（つまり、ページ・ディレクトリ・エントリは、2M バイト・ページを指すことも、4K バイト・ページを指しているページテーブルを指すこともできる。）36 ビットの物理アドレスをサポートするため、ページング・データ構造は次のように変更される。

- 36 ビットのベース物理アドレスを収容するため、ページング・テーブル・エントリを 64 ビットに増やす。
- ページ・ディレクトリ・ポインタ・テーブルという新しいテーブルをリニアアドレス変換階層に追加する。このテーブルは、それぞれが 64 ビットの 4 個のエントリを持ち、階層ではページ・ディレクトリの上に置かれる。物理アドレス拡張メカニズムがイネーブルになっていると、プロセッサは、4 つまでのページ・ディレクトリをサポートする。
- レジスタ CR3 の 20 ビットのページ・ディレクトリ・ベース・アドレス・フィールド（PDPR）を 27 ビットのページ・ディレクトリ・ポインタ・テーブル・ベース・アドレス・フィールド（図 3-17. を参照）に置き換える（この場合には、レジスタ CR3 を PDPTR という）。このフィールドは、ページ・ディレクトリ・ポインタ・テーブルの先頭バイトの物理アドレスのうち上位 27 ビットを提供し、これによってテーブルは強制的に 32 バイト境界に置かれる。
- 32 ビットのリニアアドレスをより大きい物理アドレス空間にマッピングできるように、リニアアドレス変換を変更する。

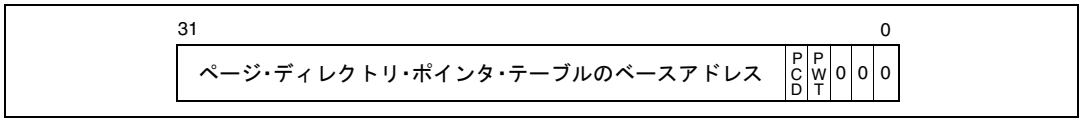


図 3-17. 物理アドレス拡張がイネーブルの場合のレジスタ CR3 のフォーマット

3.8.1. PAE がイネーブルの場合のリニアアドレス変換（4K バイト・ページ）

図 3-18. に、PAE ページング・メカニズムをイネーブルにしてリニアアドレスを 4K バイト・ページにマッピングするときのページ・ディレクトリ・ポインタ、ページ・ディレクトリ、ページテーブルの階層を示す。このページング方式を使用して、2^{20} までのページをアドレス指定でき、これは、2^{32} バイト（4G バイト）のリニアアドレス空間を占める。

さまざまなテーブルエントリを選択するため、リニアアドレスは次の 3 つのセクションに分割される。

- ページ・ディレクトリ・ポインタ・テーブル・エントリ ー ビット 30 と 31 は、ページ・ディレクトリ・ポインタ・テーブルにある 4 エントリのうちの 1 つへのオフセットを示す。選択されているエントリは、ページ・ディレクトリのベース物理アドレスを示す。
- ページ・ディレクトリ・エントリ ー ビット 21 ～ 29 は、選択されているページ・ディレクトリにあるエントリへのオフセットを示す。選択されているエントリは、ページテーブルのベース物理アドレスを示す。
- ページ・テーブル・エントリ ー ビット 12 ～ 20 は、選択されているページテーブルにあるエントリへのオフセットを示す。このエントリは、物理メモリにあるページのベース物理アドレスを示す。
- ページ・オフセット ー ビット 0 ～ 11 は、そのページにある物理アドレスへのオフセットを示す。

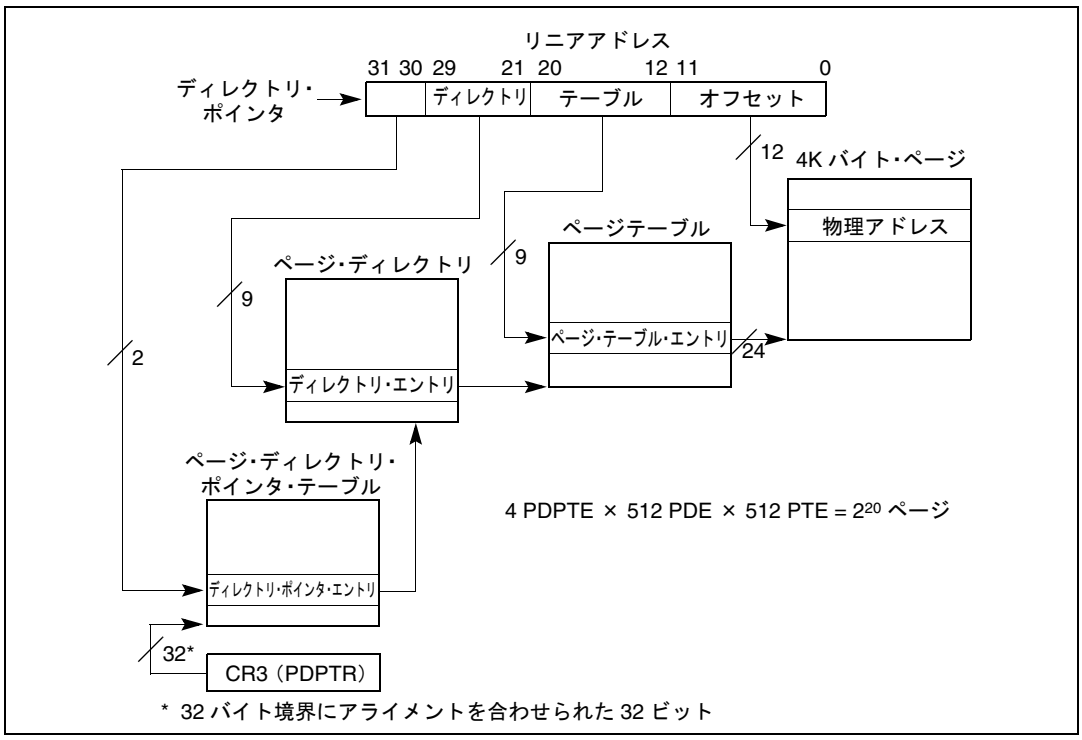


図 3-18. PAE がイネーブルの場合のリニアアドレス変換（4K バイト・ページ）

3.8.2. PAE がイネーブルの場合のリニアアドレス変換（2M バイト・ページ）

図 3-19. に、PAE ページング・メカニズムがイネーブルのときに、ページ・ディレクトリ・ポインタ・テーブルとページ・ディレクトリを使用してリニアアドレスを 2M バイト・ページにマッピングする方法を示す。このページング方式を使用して、2048（4 ページ・ディレクトリ・ポインタ・テーブル・エントリ × 512 ページ・ディレクトリ・エントリ）までのページを 4G バイトのリニアアドレス空間にマッピングできる。

PAE がイネーブルのとき、2M バイト・ページのページサイズを選択するには、ページ・ディレクトリ・エントリのページサイズ（PS）フラグをセットする（図 3-14. を参照）。（表 3-3. に示すように、PAE がイネーブルのときは、制御レジスタ CR4 内の PSE フラグはページサイズに影響しない。）PS フラグをセットすると、リニアアドレスは次の 3 つのセクションに分割される。

- ページ・ディレクトリ・ポインタ・テーブル・エントリ ― ビット 30 と 31 は、ページ・ディレクトリ・ポインタ・テーブルにあるエントリへのオフセットを示す。選択されているエントリは、ページ・ディレクトリのベース物理アドレスを示す。

- ページ・ディレクトリ・エントリ ー ビット 21 ～ 29 は、ページ・ディレクトリにあるエントリへのオフセットを示す。選択されているエントリは、2M バイト・ページのベース物理アドレスを示す。
- ページ・オフセット ー ビット 0 ～ 20 は、そのページにある物理アドレスへのオフセットを示す。

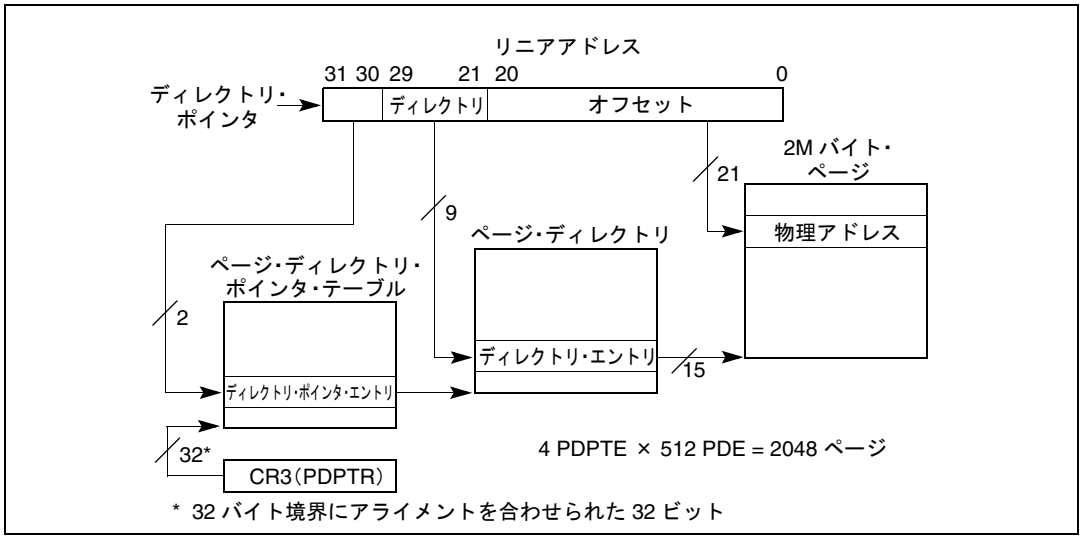


図 3-19. PAE がイネーブルの場合のリニアアドレス変換
（2M バイト・ページまたは 4M バイト・ページ）

3.8.3. 拡張ページテーブル構造を使用した拡張物理アドレス空間全体へのアクセス

前の 2 つの項で説明したページテーブル構造を使用して、64G バイト拡張物理アドレス空間のうちの 4G バイトまでを一度にアドレス指定できる。物理メモリの追加の 4G バイト部分は、次の 2 つの方法のいずれかでアドレス指定できる。

- 別のページ・ディレクトリ・ポインタ・テーブルを指すようにレジスタ CR3 のポインタを変更する。ポインタテーブルは、別のページ・ディレクトリとページテーブルのセットを指している。
- 別のページ・ディレクトリを指すようにページ・ディレクトリ・ポインタ・テーブルのエントリを変更する。ページ・ディレクトリは、別のページテーブルのセットを指している。

3.8.4. 拡張アドレス指定がイネーブルの場合のページ・ディレクトリ・エントリとページ・テーブル・エントリ

図 3-20. に、4K バイト・ページと 36 ビット拡張物理アドレスを使用している場合のページ・ディレクトリ・ポインタ・テーブル・エントリ、ページ・ディレクトリ・エントリ、ページ・テーブル・エントリのフォーマットを示す。図 3-21. には、2M バイト・ページと 36 ビット拡張物理アドレスを使用している場合のページ・ディレクトリ・ポインタ・テーブル・エントリとページ・ディレクトリ・エントリのフォーマットを示す。これらのエントリにあるフラグの機能は、3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」で説明したものと同じである。これらのエントリの主な相違点は、次のとおりである。

- ページ・ディレクトリ・ポインタ・テーブル・エントリが追加されている。
- エントリのサイズが 32 ビットから 64 ビットに増えている。
- ページ・ディレクトリまたはページ・テーブルにあるエントリの最大数は 512 である。
- 各エントリのベース物理アドレス・フィールドが 24 ビットに拡張されている。

注記

PAE メカニズムを実装している現在の IA-32 プロセッサでは、ページ・ディレクトリ・ポインタ・テーブル・エントリのロード時に、キャッシュなしのアクセスを使用する。この動作はモデル固有のもので、アーキテクチャに基づくものではない。今後の IA-32 プロセッサでは、ページ・ディレクトリ・ポインタ・テーブル・エントリをキャッシュする可能性もある。

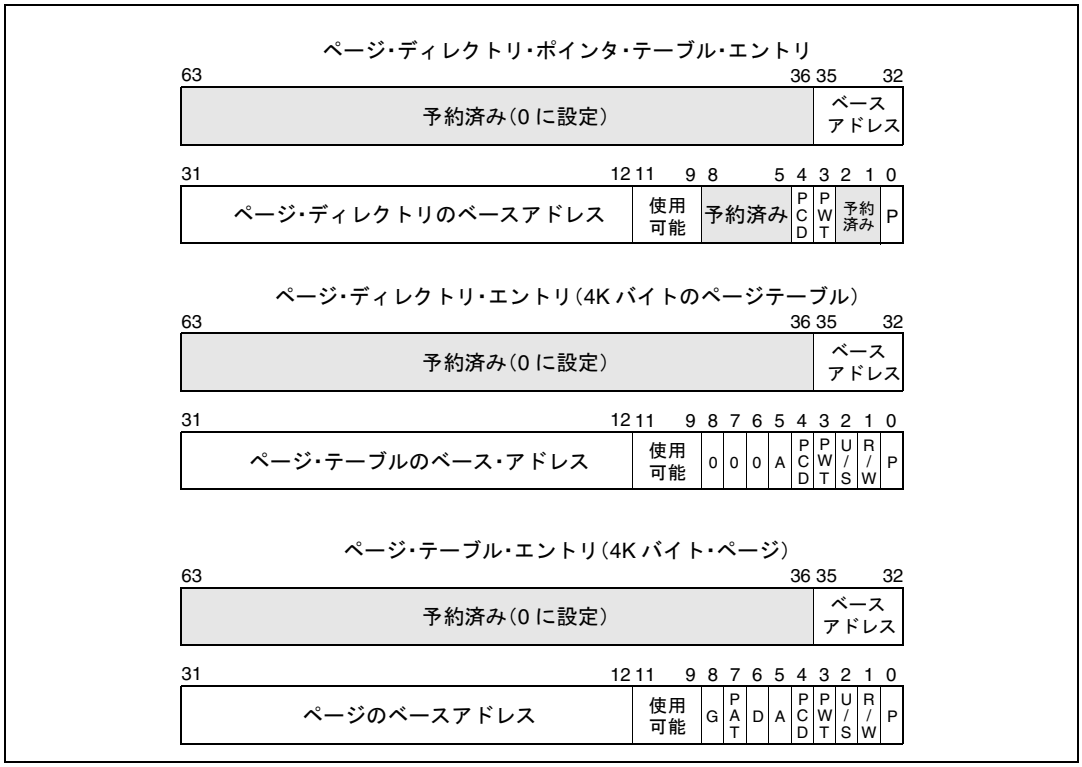


図 3-20. PAE がイネーブルの 4K バイト・ページを使用する場合のページ・ディレクトリ・ポインタ・テーブル・エントリ、ページ・ディレクトリ・エントリ、ページ・テーブル・エントリのフォーマット

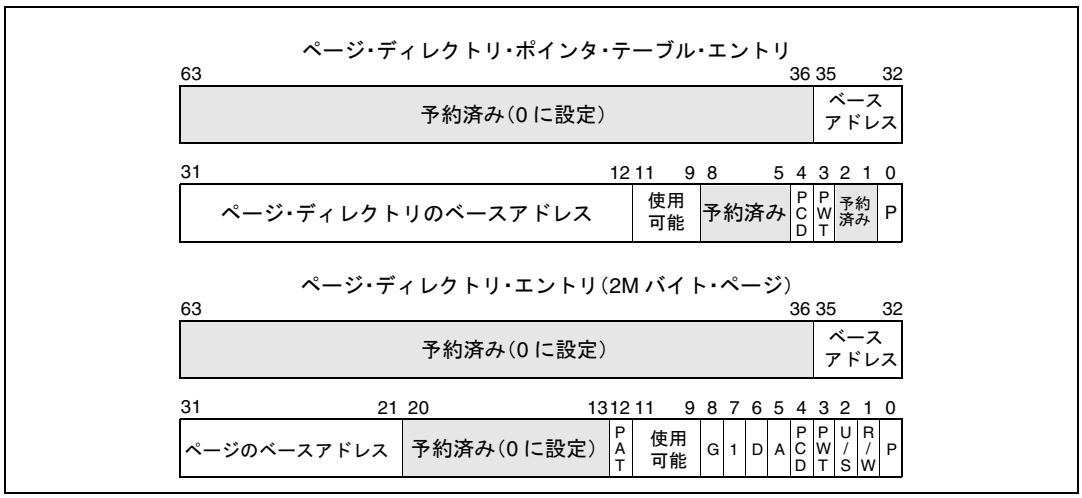


図 3-21. PAE がイネーブルの 2M バイト・ページを使用する場合のページ・ディレクトリ・ポインタ・テーブル・エントリとページ・ディレクトリ・エントリのフォーマット

エントリのベース物理アドレスは、エントリのタイプに応じて次の内容を指定する。

- ページ・ディレクトリ・ポインタ・テーブル・エントリ － 4K バイト・ページ・ディレクトリの先頭バイトの物理アドレス
- ページ・ディレクトリ・エントリ － 4K バイト・ページ・テーブルまたは 2M バイト・ページの先頭バイトの物理アドレス
- ページ・テーブル・エントリ － 4K バイト・ページの先頭バイトの物理アドレス

（2M バイト・ページを指すページ・ディレクトリ・エントリを除く）すべてのテーブル・エントリでは、ページ・ベース・アドレスのビットは、36 ビット物理アドレスの上位 24 ビットと解釈され、これがページテーブルとページのアライメントを強制的に 4K バイト境界に合わせる。ページ・ディレクトリ・エントリが 2M バイト・ページを指しているときは、ベースアドレスは 36 ビット物理アドレスの上位 15 ビットと解釈され、これがページのアライメントを強制的に 2M バイト境界に合わせる。

ページ・ディレクトリ・ポインタ・テーブル・エントリ内の存在フラグ（ビット 0）は、0 または 1 に設定できる。存在フラグがクリアされている場合、ページ・ディレクトリ・ポインタ・テーブル・エントリ内のその他のビットは、オペレーティング・システムが利用できる。存在フラグがセットされている場合のページ・ディレクトリ・ポインタ・テーブル・エントリの各フィールドの定義を、図 3-20.（4KB ページの場合）と図 3-21.（2MB ページの場合）に示す。

ページ・ディレクトリ・エントリのページサイズ（PS）フラグ（ビット 7）は、そのエントリがページテーブルか 2M バイト・ページのいずれを指しているかを決定する。このフラグがクリアされていると、エントリはページテーブルを指し、セットされていると、エントリは 2M バイト・ページを指している。このフラグによって、4K バイト・ページと 2M バイト・ページを 1 つのページング・テーブルのセット内で混在できる。

アクセス済み（A）フラグとダーティ（D）フラグ（ビット 5 および 6）は、ページを指すテーブルエントリのために用意されている。

物理アドレス拡張のためのすべてのテーブルエントリのビット 9、10、11 は、ソフトウェアが使用するためのものである。（存在フラグがクリアされていると、ビット 1 ～ 63 がソフトウェアで使用できる。）予約済みまたは 0 とマークされている図 3-14. にあるすべてのビットは、ソフトウェアによって 0 に設定される必要があり、ソフトウェアがアクセスしてはならない。制御レジスタ CR4 の PSE フラグまたは PAE フラグあるいはその両方がセットされていると、プロセッサは、ページ・ディレクトリ・エントリとページ・テーブル・エントリの予約ビットが 0 に設定されていなければページフォルト（#PF）を生成し、ページ・ディレクトリ・ポインタ・テーブル・エントリの予約ビットが 0 に設定されていなければ一般保護例外（#GP）を生成する。

3.9. PSE-36 ページング・メカニズムを使用した 36 ビット物理アドレス指定

PSE-36 ページング・メカニズムを使用すると、PAE メカニズムとは別の方法で、物理メモリのアドレス指定を 36 ビットに拡張できる。このメカニズムでは、ページサイズ拡張（PSE）モードおよび変更ページ・ディレクトリ・テーブルを使用して、4M バイトのページを 64G バイトの物理アドレス空間へマップする。PAE メカニズムと同様、プロセッサには、追加のアドレスビットを収容するための 4 本の追加アドレス・ライン・ピンが備えられている。

PSE-36 メカニズムは、インテル® Pentium® III プロセッサで IA-32 アーキテクチャに導入された。この機能を使用できるかどうかは、PSE-36 機能ビット（ソース・オペランドに 1 をセットして CPUID 命令を実行したときの、EDX レジスタのビット 17）で示される。

PSE-36 ページング・メカニズムをイネーブルにするには、表 3-3. に示すように、以下のフラグをセットまたはクリアする必要がある。

- PSE-36 CPUID 機能フラグ ― このフラグがセットされていると、CPUID 命令を実行する IA-32 プロセッサ上で PSE-36 ページング・メカニズムを使用できる。
- レジスタ CR0 の PG フラグ（ビット 31）― ページングをイネーブルにするには、1 にセットする。
- 制御レジスタ CR4 の PAE フラグ（ビット 5）― PAE ページング・メカニズムをディスエーブルにするには、0 にクリアする。
- 制御レジスタ CR4 の PSE フラグ（ビット 4）および PDE の PS フラグ― 4M バイト・ページでのページサイズ拡張をイネーブルにするには、1 にセットする。
- 制御レジスタ CR4 の PSE フラグ（ビット 4）および PDE の PS フラグ（ビット 7）― 32 ビットのアドレス指定を行って 4K バイト・ページをイネーブルにするには、PSE フラグを 1 にセットし、PS フラグを 0 にクリアする。

図 3-22. は、拡張ページ・ディレクトリ・エントリを使用して、32 ビットのリニアアドレスを 36 ビットの物理アドレスにマップする方法を示している。このリニアアドレスは、次の 2 つのセクションに分割される。

- ページ・ディレクトリ・エントリ ― ビット 22 ～ 35 は、ページ・ディレクトリ内のエントリへのオフセットを表す。選択されたエントリは、36 ビット・アドレスの上位 14 ビットを表し、これによって 4M バイト・ページのベース物理アドレスが特定される。
- ページ・オフセット ― ビット 0 ～ 21 は、ページ内の物理アドレスへのオフセットを表す。

このページング方式を使用すると、64Gバイトの物理アドレス空間に最大1024ページをマップができる。

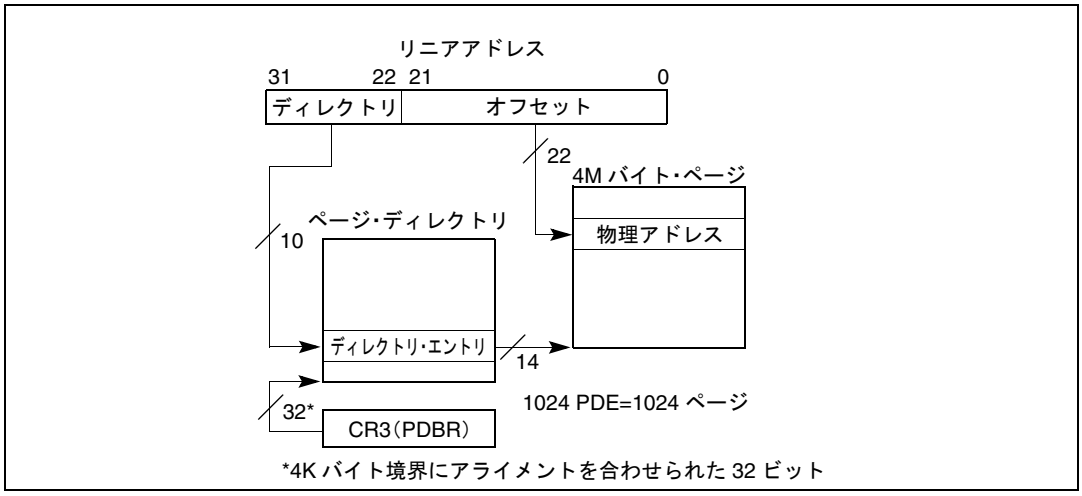


図 3-22. リニアアドレス変換（4M バイト・ページ）

図3-23.は、4Mバイト・ページおよび36ビット物理アドレスを使用する場合の、ページ・ディレクトリ・エントリのフォーマットを示している。3.7.6.項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」では、ビット0～11のフィールドとフラグの機能を説明している。

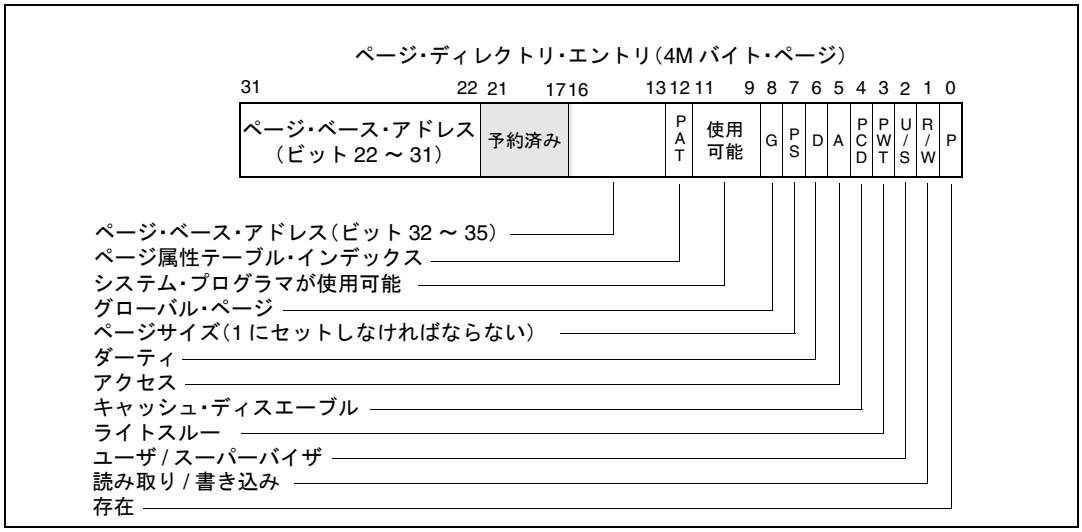


図 3-23. 4M バイト・ページと 36 ビット物理アドレスを使用する場合の
ページ・ディレクトリ・エントリのフォーマット

3.10. ページへのセグメントのマッピング

IA-32 アーキテクチャに備えられているセグメンテーション・メカニズムとページング・メカニズムは、メモリ管理への幅広い多様なアプローチをサポートする。セグメンテーションとページングを組み合わせると、セグメントをページに複数の方法でマッピングできる。例えば、フラットな（セグメント化されていない）アドレス指定環境を実現するため、すべてのコード・モジュール、データ・モジュール、スタック・モジュールを、リニアアドレスの同じ範囲を共有する 1 つ以上の大きいセグメント（4G バイトまで）にマッピングできる（図 3-2. を参照）。この場合には、セグメントは必然的にアプリケーションとオペレーティング・システムまたはエグゼクティブからは見れない。ページングを使用すると、ページング・メカニズムでは、（1 つのセグメントだけに含まれている）1 つのリニアアドレス空間だけを仮想メモリにマッピングできる。あるいは、各プログラム（またはタスク）は、専用の大きいリニアアドレス空間（専用のセグメントに含まれている）を持つことができ、それが専用のページ・ディレクトリとページテーブルのセットを通じて仮想メモリにマッピングされる。

セグメントをページのサイズより小さくできる。これらのセグメントの 1 つを別のセグメントと共有していないページに置くと、残りのメモリは無駄になる。例えば、1 バイト・セマフォなどの小さいデータ構造をそれだけで 1 つのページに入れると、それで 4K バイトを占有する。多くのセマフォが使用されている場合は、それらを 1 つのページにまとめて入れるほうが効率的である。

IA-32 アーキテクチャは、ページとセグメントの境界間の対応付けを強制していない。1 つのページ内に 1 つのセグメントの終了部分と別のセグメントの開始部分が入っていてもよい。同様に、1 つのセグメント内に 1 つのページの終了部分と別のページの開始部分が入っていてもよい。

メモリ管理ソフトウェアは、ページ境界とセグメント境界間に一定のアライメント合わせを実施すると、簡単で効率的になる場合がある。例えば、1 つのページに入りきるセグメントを 2 つのページに入れると、そのセグメントへのアクセスをサポートするために最大で 2 倍のページング・オーバーヘッドがかかる。

ページングとセグメンテーションを組み合わせてメモリ管理ソフトウェアを単純化する 1 つのアプローチとして、図 3-24. に示すように、各セグメントに専用のページテーブルを与える方法がある。この規約では、セグメントにページ・ディレクトリの 1 つのエントリを与え、そのエントリがセグメント全体をページングするためのアクセス制御情報を提供する。

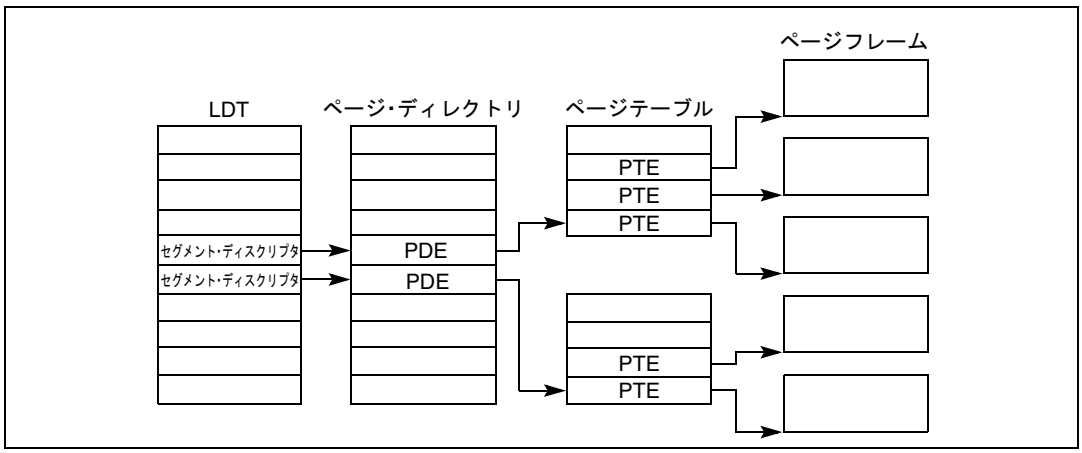


図 3-24. ページテーブルを各セグメントに割り当てるメモリ管理規約

3.11. トランスレーション・ルックアサイド・バッファ（TLB）

プロセッサは、直前に使用されたページ・ディレクトリ・エントリとページ・テーブル・エントリをトランスレーション・ルックアサイド・バッファ（TLB）というオンチップのキャッシュに記憶する。P6ファミリ・プロセッサとインテル® Pentium® プロセッサには、データ・キャッシュと命令キャッシュに別々の TLB がある。また、P6ファミリ・プロセッサには、4K バイトと 4M バイトのページサイズに別々の TLB がある。CPUID 命令を使用して、P6 ファミリ・プロセッサとインテル Pentium プロセッサに備えられている TLB のサイズを判定できる。

ほとんどのページングは、TLB の内容を使用して行われる。メモリ内のページ・ディレクトリとページテーブルへのバスサイクルは、要求されたページの変換情報が TLB に入っていないときだけに行われる。

TLB は、アプリケーション・プログラムとタスク（0 より大きい特権レベル）からアクセスすることはできない。つまり、それらが TLB を無効化できない。0 の特権レベルで実行しているオペレーティング・システムまたはエグゼクティブ・プロシージャだけが TLB または TLB 内の指定したエントリを無効化できる。ページ・ディレクトリ・エントリまたはページ・テーブル・エントリが（存在フラグがゼロに設定されているときを含めて）変更されたときには、オペレーティング・システムは、TLB の対応するエントリが次に参照されるときに更新できるように、そのエントリを直ちに無効化しなければならない。

（グローバルでない）すべての TLB は、（本節で後述するように、ページまたはページ・テーブル・エントリの G フラグがセットされていない限り）レジスタ CR3 がロー

ドされると常に自動的に無効化される。レジスタ CR3 は、次の 2 つの方法のいずれかでロードされる。

- MOV 命令を使用して明示的に。

  ```
  MOV CR3, EAX
  ```

  この場合に、EAX レジスタには適当なページ・ディレクトリのベースアドレスが入っている。
- タスクスイッチを実行することによって暗黙のうちに、レジスタ CR3 の内容が自動的に変わる。

TLB の特定のページ・テーブル・エントリを無効化するため、INVLPG 命令が用意されている。通常、この命令は個別の TLB エントリだけを無効化するが、指定されたエントリ以外のエントリを無効化する場合があり、すべての TLB を無効化する場合もある。この命令は、ページ・ディレクトリ・エントリまたはページ・テーブル・エントリの G フラグの設定を無視する（次のパラグラフを参照）。

（インテル® Pentium® Pro プロセッサで導入された。）レジスタ CR4 のページ・グローバル・イネーブル（PGE）フラグ、およびページ・ディレクトリ・エントリまたはページ・テーブル・エントリのグローバル（G）フラグ（ビット 8）を使用して、頻繁に使用されるページがタスクスイッチ時またはレジスタ CR3 のロード時に TLB で自動的に無効化されないようにできる。（グローバル・フラグの詳細については、3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」を参照。）プロセッサがグローバル・ページのページ・ディレクトリ・エントリまたはページ・テーブル・エントリを TLB にロードすると、そのエントリは、いつまでも TLB に残る。グローバル・ページ・エントリを完全に無効化する唯一の方法は、以下のとおりである。

- PGE フラグをクリアしてから TLB を無効化する。
- INVLPG 命令を実行して、TLB にある個々のページ・ディレクトリ・エントリまたはページ・テーブル・エントリを無効化する。

TLB3 の無効化の詳細については、10.9. 節「トランスレーション・ルックアサイド・バッファ（TLB）の無効化」を参照。

4

保護

第 4 章
保護

4

IA-32 アーキテクテクチャは保護モードにおいて、セグメント・レベルとページレベルの両方で働く保護メカニズムを提供している。この保護メカニズムは、特権レベル（セグメントに4つの特権レベル、ページに2つの特権レベル）に基づいて特定のセグメントまたはページへのアクセスを制限できる。例えば、オペレーティング・システムの重要なコードとデータを、アプリケーション・コードを含むセグメントより高い特権レベルのセグメントに置くことによって、それらのコードとデータを保護できる。このようにして、プロセッサの保護メカニズムは、アプリケーション・コードが、制御され定義された以外の方法でオペレーティング・システムのコードとデータにアクセスしないようにする。

セグメントとページの保護をソフトウェア開発のすべての段階で使用して、設計上の問題やバグの局所化と検出を容易にできる。また、保護を最終製品に組み込んで、オペレーティング・システム、ユーティリティ・ソフトウェア、アプリケーション・ソフトウェアをさらに頑健にできる。

保護メカニズムを使用すると、各メモリ参照がチェックされ、さまざまな保護チェックを満たしているか検証される。すべてのチェックは、メモリサイクルが開始される前に実行され、違反があれば例外が発生する。チェックはアドレス変換と並行して実行されるので、処理能力が損なわれることはない。実行される保護チェックは、次のカテゴリに分類される。

- リミットチェック
- タイプチェック
- 特権レベルチェック
- アドレス指定可能なドメインの制限
- プロシージャ・エントリ・ポイントの制限
- 命令セットの制限

どのような保護違反が起こっても、例外が発生することになる。例外メカニズムの説明については、第5章「割り込みと例外の処理」を参照。本章では、保護メカニズムおよび例外となる違反について説明する。

以降の節では、保護モードで利用できる保護メカニズムについて説明する。実アドレスモードと仮想8086モードでの保護の詳細については、第16章「8086エミュレーション」を参照。

4.1. セグメントとページの保護の有効化 / 無効化

レジスタ CR0 の PE フラグをセットすると、プロセッサは保護モードに切り替わり、セグメント保護メカニズムがイネーブルになる。いったん保護モードになると、保護メカニズムをオンまたはオフにする制御ビットはない。まだ保護モードにいる間に（最も特権レベルの高い）0 の特権レベルをすべてのセグメント・セレクタとセグメント・ディスクリプタに割り当てると、特権レベルに基づいているセグメント保護メカニズムの一部を本質的にディスエーブルにできる。この処置によって、セグメント間の特権レベル保護バリアはディスエーブルになるが、リミットチェックやタイプチェックなどその他の保護チェックは引き続き実行される。

（レジスタ CR0 の PG フラグをセットして）ページングをイネーブルにすると、ページレベル保護が自動的にイネーブルになる。この場合にも、いったんページングがイネーブルになると、ページレベル保護をオフにするモードビットはない。ただし、次の操作を実行すると、ページレベル保護をディスエーブルにできる。

- 制御レジスタ CR0 の WP フラグをクリアする。
- ページ・ディレクトリ・エントリとページ・テーブル・エントリごとに読み取り / 書き込み（R/W）フラグとユーザ / スーパーバイザ（U/S）フラグをセットする。

この処置によって、各ページは書き込み可能なユーザページとなり、実質的にページレベル保護がディスエーブルにされたことになる。

4.2. セグメント・レベルとページレベルの保護に使用されるフィールドとフラグ

プロセッサの保護メカニズムでは、システムデータ構造内にある次のフィールドとフラグを使用して、セグメントとページへのアクセスを制御する。

- ディスクリプタ・タイプ（S）フラグ ー（セグメント・ディスクリプタの第二ダブルワードのビット 12）セグメント・ディスクリプタがシステム・セグメント用であるかあるいはコード・セグメントまたはデータ・セグメント用であるかを決定する。
- タイプ・フィールド ー（セグメント・ディスクリプタの第二ダブルワードのビット 8 ～ 11）コード、データ、またはシステム・セグメントのタイプを決定する。
- リミット・フィールド ー（セグメント・ディスクリプタの第一ダブルワードのビット 0 ～ 15 と第二ダブルワードのビット 16 ～ 19）G フラグと E フラグ（データ・セグメントの場合）と共に、セグメントのサイズを決定する。

- G フラグ　－（セグメント・ディスクリプタの第二ダブルワードのビット 23）リミット・フィールドと E フラグ（データ・セグメントの場合）と共に、セグメントのサイズを決定する。
- E フラグ －（データ・セグメント・ディスクリプタの第二ダブルワードのビット 10）リミット・フィールドと G フラグと共に、セグメントのサイズを決定する。
- ディスクリプタ特権レベル（DPL）フィールド －（セグメント・ディスクリプタの第二ダブルワードのビット 13 と 14）セグメントの特権レベルを決定する。
- 要求される特権レベル（RPL）フィールド －（任意のセグメント・セレクタのビット 0 と 1）セグメント・セレクタの要求される特権レベルを指定する。
- 現行特権レベル（CPL）フィールド －（CS セグメント・レジスタのビット 0 と 1）現在実行しているプログラムまたはプロシージャの特権レベルを示す。現行特権レベル（CPL）という用語は、このフィールドの設定を表す。
- ユーザ / スーパーバイザ（U/S）フラグ －（ページ・ディレクトリ・エントリまたはページ・テーブル・エントリのビット 2）ページのタイプ（ユーザまたはスーパーバイザ）を決定する。
- 読み取り / 書き込み（R/W）フラグ　－（ページ・ディレクトリ・エントリまたはページ・テーブル・エントリのビット 1）ページに対して許可されるアクセスのタイプ（読み取り専用または読み取り / 書き込み）を決定する。

図 4-1. に、データ、コード、システムのセグメント・ディスクリプタにあるさまざまなフィールドとフラグの位置を示す。図 3-6. には、セグメント・セレクタ（または CS レジスタ）にある RPL（または CPL）フィールドの位置が示されている。図 3-14. には、ページ・ディレクトリ・エントリとページ・テーブル・エントリにある U/S フラグと R/W フラグの位置が示されている。

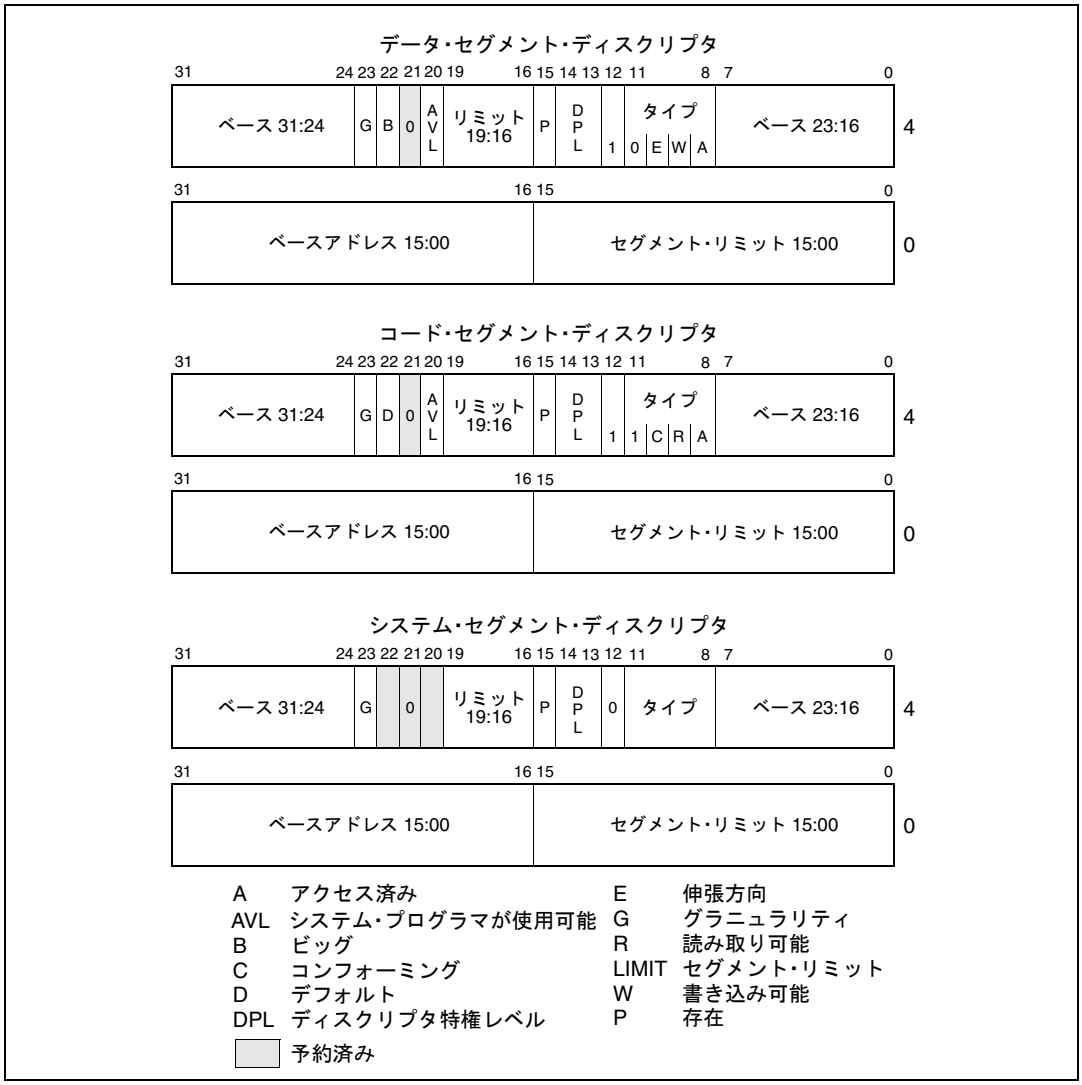


図 4-1. 保護のために使用されるディスクリプタのフィールド

これらのフィールドとフラグを利用して、多くの異なるスタイルの保護スキームを使用できる。オペレーティング・システムは、ディスクリプタを作成するときに、オペレーティング・システムまたはエグゼクティブ用に選択されている特定の保護スタイルに合わせてこれらのフィールドとフラグに値を設定する。アプリケーション・プログラムは、一般的にはこれらのフィールドとフラグにアクセスしたり、これらを修正することはしない。

以降の節では、プロセッサがこれらのフィールドとフラグを使用して、この章の冒頭で説明したさまざまなカテゴリのチェックをどのように実行するかについて説明する。

4.3. リミットチェック

セグメント・ディスクリプタのリミット・フィールドは、プログラムまたはプロシージャがセグメント外のメモリ位置をアドレス指定しないようにする。リミットの有効値は、G（グラニュラリティ）フラグの設定によって変わる（図 4-1. を参照）。データ・セグメントでは、リミット値は、E（伸張方向）フラグと B（デフォルトのスタック・ポインタ・サイズまたは上限あるいはその両方）フラグによっても変わる。セグメント・ディスクリプタがデータタイプのセグメント用である場合は、E フラグはタイプ・フィールドにあるビットの 1 つである。

G フラグがクリアされていると（バイト・グラニュラリティ）、有効リミットは、セグメント・ディスクリプタにある 20 ビットのリミット・フィールドの値となる。この場合には、リミットの範囲は、0 ～ FFFFFH（1M バイト）になる。G フラグがセットされていると（4K バイト・ページ・グラニュラリティ）、プロセッサは、リミット・フィールドにある値を 2^{12}（4K バイト）のファクタでスケーリングする。この場合には、有効リミットの範囲は、FFFH（4K バイト）～ FFFFFFFFH（4G バイト）になる。スケーリングが使用されると（G フラグがセットされていると）、セグメント・オフセット（アドレス）の下位 12 ビットはリミットに対してチェックされないことに注意が必要である。例えば、セグメント・リミットが 0 である場合は、オフセット 0 ～ FFFH がなお有効であることに注意する。

エクスパンドダウン・データ・セグメントを除いたすべてのタイプのセグメントでは、有効リミットは、セグメント内でアクセスを許可される最終アドレスであり、セグメントのバイト単位のサイズより 1 だけ小さい。プロセッサは、セグメント内の次のアドレスへのアクセスが試みられると、一般保護例外を生成する。

- オフセットが有効リミットより大きいバイト
- オフセットが（有効リミット - 1）より大きいワード
- オフセットが（有効リミット - 3）より大きいダブルワード
- オフセットが（有効リミット - 7）より大きいクワッドワード

エクスパンドダウン・データ・セグメントでは、セグメント・リミットは他の場合と同じ機能を果たすが、異なって解釈される。この場合には、有効リミットは、セグメント内でアクセスを許可されていない最終アドレスを指定する。有効なオフセットの範囲は、B フラグがセットされている場合には（有効リミット + 1）～ FFFFFFFFH であり、B フラグがクリアされている場合は（有効リミット + 1）～ FFFFH である。エクスパンドダウン・セグメントは、セグメント・リミットが 0 であるときに最大サイズになる。

リミットチェックは、暴走コードや暴走サブスクリプト、無効なポインタ計算などのプログラミング・エラーを検出する。これらのエラーは、それらが発生したときに検

出されるため、原因の特定が容易である。リミットチェックを実行しないと、これらのエラーによって別のセグメントにあるコードまたはデータが上書きされるおそれがある。

セグメント・リミットのチェックに加えて、プロセッサは、ディスクリプタ・テーブルのリミットもチェックする。GDTR レジスタと IDTR レジスタには、16 ビットのリミット値が入っており、プロセッサはこれらを使用して、プログラムがそれぞれのディスクリプタ・テーブル外にあるセグメント・ディスクリプタを選択するのを防ぐ。LDTR レジスタとタスクレジスタには、（それぞれ現在の LDT と TSS のセグメント・ディスクリプタから読み取られた）32 ビットのセグメント・リミット値が入っている。プロセッサは、これらのセグメント・リミットを使用して、現在の LDT と TSS の境界を超えるアクセスがないようにする。GDT と LDT のリミット・フィールドの詳細については、3.5.1. 項「セグメント・ディスクリプタ・テーブル」を参照。IDT のリミット・フィールドの詳細については、5.10. 節「割り込みディスクリプタ・テーブル（IDT: Interrupt Descriptor Table）」を参照。TSS セグメントのリミット・フィールドの詳細については、6.2.3. 項「タスクレジスタ」を参照。

4.4. タイプチェック

セグメント・ディスクリプタには、次の 2 つの場所にタイプ情報が入っている。

- S（ディスクリプタ・タイプ）フラグ
- タイプ・フィールド

プロセッサは、これらの情報を使用して、間違ったまたは意図されていない方法でセグメントまたはゲートを使用する結果になるプログラミング・エラーを検出する。

S フラグは、ディスクリプタがシステムタイプであるか、あるいはコードまたはデータタイプであるかを示す。タイプ・フィールドには、コード、データ、システム・ディスクリプタのさまざまなタイプを定義する際に使用される追加の 4 ビットが入っている。表 3-1. には、コード・ディスクリプタとデータ・ディスクリプタのタイプ・フィールドのエンコーディングが示され、表 3-2. には、システム・ディスクリプタのタイプ・フィールドのエンコーディングが示されている。

プロセッサは、セグメント・セレクタとセグメント・ディスクリプタを操作するとき、さまざまな時点でタイプ情報を調べる。次のリストは、タイプチェックが実行される代表的な操作の例を示している。ただし、このリストはすべてを網羅しているわけではない。

- **セグメント・セレクタがセグメント・レジスタにロードされるとき。**特定のセグメント・レジスタには、特定のディスクリプタ・タイプだけを入れられる。例えば、
  - CS レジスタには、コード・セグメントのセレクタだけをロードできる。

- データ・セグメント・レジスタ（DS、ES、FS、GS）には、読み取り不可能なコード・セグメントまたはシステム・セグメントのセグメント・セレクタをロードできない。
- SS レジスタには、書き込み可能なデータ・セグメントのセグメント・セレクタだけをロードできる。

- **セグメント・セレクタがLDTR レジスタまたはタスクレジスタにロードされるとき**
  - LDTR には、LDT のセレクタだけをロードできる。
  - タスクレジスタには、TSS のセグメント・セレクタだけをロードできる。
- **ディスクリプタがセグメント・レジスタにすでにロードされているセグメントに、命令がアクセスするとき。**あらかじめ定義されている特定の方法だけで、命令が特定のセグメントを使用できる。例えば、
  - 命令は、実行可能セグメントに書き込むことはできない。
  - 命令は、データ・セグメントが書き込み可能でない場合にはそのセグメントに書き込むことはできない。
  - 命令は、読み取り可能フラグがセットされていない限り、実行可能セグメントを読み取ることはできない。
- **命令オペランドにセグメント・セレクタが入っているとき。**特定の命令が、特定のタイプのセグメントまたはゲートだけにアクセスできる。例えば、
  - far CALL 命令または far JMP 命令は、コンフォーミング・コード・セグメント、非コンフォーミング・コード・セグメント、コールゲート、タスクゲートまたは TSS のセグメント・ディスクリプタだけにアクセスできる。
  - LLDT 命令は、LDT のセグメント・ディスクリプタを参照しなければならない。
  - LTR 命令は、TSS のセグメント・ディスクリプタを参照しなければならない。
  - LAR 命令は、LDT、TSS、コールゲート、タスクゲート、コード・セグメント、またはデータ・セグメントのセグメント・ディスクリプタまたはゲート・ディスクリプタを参照しなければならない。
  - LSL 命令は、LDT、TSS、コード・セグメント、またはデータ・セグメントのセグメント・ディスクリプタを参照しなければならない。
  - IDT エントリは、割り込みゲート、トラップゲート、またはタスクゲートでなければならない。
- **例えば次のような内部操作の最中**
  - （far CALL 命令または far JMP 命令で実行される）far コールまたは far ジャンプでは、プロセッサは、CALL 命令または JMP 命令のオペランドとして指定されているセグメント（またはゲート）セレクタによって指されているセグメント（またはゲート）ディスクリプタのタイプ・フィールドをチェックして、実行する制御移行のタイプ（別のコード・セグメントへの呼び出しまたはジャンプ、ゲートを通じた

呼び出しまたはジャンプ、あるいはタスクスイッチ）を決定する。ディスクリプタ・タイプがコード・セグメントまたはコールゲート用である場合は、別のコード・セグメントへの呼び出しまたはジャンプが示される。ディスクリプタ・タイプが TSS またはタスクゲート用である場合は、タスクスイッチが示される。

- コールゲートを通じたコールまたはジャンプでは（あるいはトラップゲートまたは割り込みゲートを通じた割り込みまたは例外のハンドラのコールでは）、プロセッサは、ゲートによって指されているセグメント・ディスクリプタがコード・セグメント用であることを自動的にチェックする。
- タスクゲートを通じた新しいタスクへのコールまたはジャンプでは（あるいはタスクゲートを通じた新しいタスクへの割り込みまたは例外のハンドラのコールでは）、プロセッサは、タスクゲートによって指されているセグメント・ディスクリプタが TSS 用であることを自動的にチェックする。
- TSS の直接参照による新しいタスクへのコールまたはジャンプでは、プロセッサは、CALL 命令または JMP 命令によって指されているセグメント・ディスクリプタが TSS 用であることを自動的にチェックする。
- （IRET 命令によって開始された）ネストされたタスクからの戻り時に、プロセッサは、現在の TSS にある前のタスク・リンク・フィールドが TSS を指していることをチェックする。

4.4.1. ヌル・セグメント・セレクタ・チェック

ヌル・セグメント・セレクタ（3.4.1.項「セグメント・セレクタ」を参照）を CS または SS セグメント・レジスタにロードしようとすると、一般保護例外（#GP）が発生する。ヌル・セグメント・セレクタは、DS、ES、FS、または GS レジスタにロードできるが、これらのレジスタにヌル・セグメント・セレクタがロードされているときにそれらの 1 つを使用してセグメントにアクセスしようとすると、#GP 例外が発生する。未使用のデータ・セグメント・レジスタにヌル・セグメント・セレクタをロードする方法は、未使用のセグメント・レジスタへのアクセスを検出したり、データ・セグメントへの望ましくないアクセスを防止するのに役立つ。

4.5. 特権レベル

プロセッサのセグメント保護メカニズムでは、番号が 0 から 3 までの 4 つの特権レベルを認識する。番号が大きくなると、特権は低くなる。図 4-2. に、これらの特権レベルをリング状の保護として解釈する方法を示す。

リングの中心は、(最も特権レベルの高いコード、データ、スタックに予約されている) 通常はオペレーティング・システムのカーネルである重要なソフトウェアを含むセグメントに使用される。外側のリングは、あまり重要でないソフトウェアに使用される。(4 つの可能な特権レベルのうちの 2 つしか使用しないシステムでは、レベル 0 と 3 を使用するべきである。)

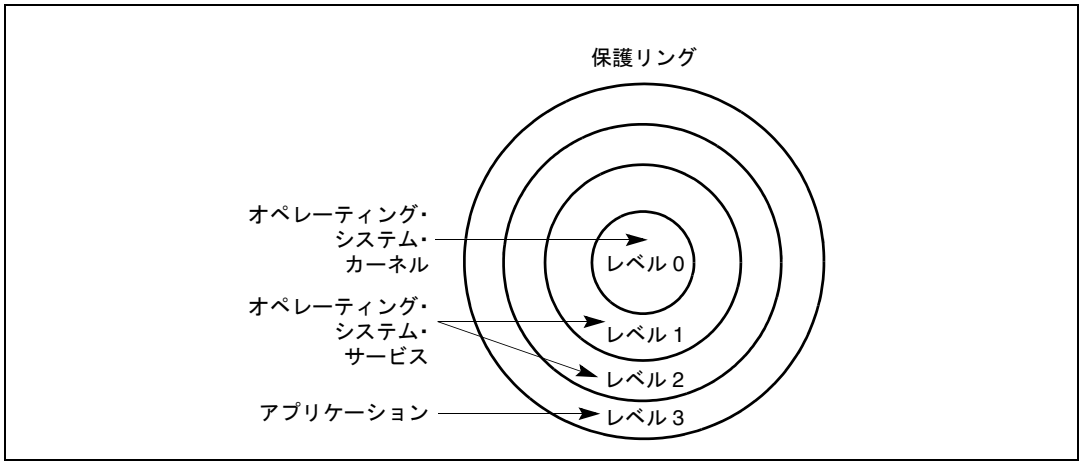


図 4-2. 保護リング

制御された状況以外では、プロセッサは特権レベルを使用して、低い特権レベルで動作しているプログラムまたはタスクが高い特権レベルのセグメントにアクセスしないようにする。プロセッサは、特権レベル違反を検出すると、一般保護例外 (#GP) を生成する。

コード・セグメントとデータ・セグメント間の特権レベルチェックを実行するため、プロセッサは、次の 3 つのタイプの特権レベルを認識する。

- **現行特権レベル** (CPL: Current Privilege Level)。CPL とは、現在実行しているプログラムまたはタスクの特権レベルである。CPL は、CS と SS セグメント・レジスタのビット 0 とビット 1 にストアされている。通常、CPL は、命令をフェッチした元のコード・セグメントの特権レベルに等しい。プログラム制御が異なる特権レベルのコード・セグメントに転送されると、プロセッサは CPL を変更する。コンフォーミング・コード・セグメントにアクセスしているときは、CPL の取り扱いかたが少し異なる。コンフォーミング・コード・セグメントには、その DPL と等

しいかまたは数値として大きい（特権レベルが低い）任意の特権レベルからアクセスできる。また、プロセッサが CPL とは異なる特権レベルを持つコンフォーミング・コード・セグメントにアクセスしても、CPL は変更されない。

- **ディスクリプタ特権レベル**（DPL: Description Privilege Level）。DPL とは、セグメントまたはゲートの特権レベルである。DPL は、セグメントのセグメント・ディスクリプタまたはゲートのゲート・ディスクリプタ内にある DPL フィールドにストアされている。現在実行しているコード・セグメントがセグメントまたはゲートにアクセスしようとすると、そのセグメントまたはゲートの DPL が（本節で後述するように）CPL、およびセグメント・セレクタまたはゲートセレクタの RPL と比較される。DPL は、アクセスされるセグメントまたはゲートのタイプに応じて異なって解釈される。
  - **データ・セグメント。**DPL は、セグメントへのアクセスを許可されるためにプログラムまたはタスクが持つ必要がある数値として最も大きい特権レベルを示す。例えば、データ・セグメントの DPL が 1 である場合は、CPL 0 または 1 で実行しているプログラムだけがそのセグメントにアクセスできる。
  - **（コールゲートを使用していない）非コンフォーミング・コード・セグメント。**DPL は、プログラムまたはタスクがそのセグメントにアクセスするためにそこにいなければならない特権レベルを示す。例えば、非コンフォーミング・コード・セグメントの DPL が 0 である場合は、CPL 0 で実行しているプログラムだけがそのセグメントにアクセスできる。
  - **コールゲート。**DPL は、現在実行しているプログラムまたはタスクがそのレベルであることができ、なおコールゲートにアクセスすることができる数値として最も大きい特権レベルを示す。（これは、データ・セグメントに対するのと同じアクセス規則である。）
  - **コンフォーミング・コード・セグメントとコールゲートを通じてアクセスされる非コンフォーミング・コード・セグメント。**DPL は、プログラムまたはタスクがそのセグメントへのアクセスを許可されるためにそのレベルである必要がある数値として最も小さい特権レベルを示す。例えば、コンフォーミング・コード・セグメントの DPL が 2 である場合は、CPL 0 または 1 で実行しているプログラムは、そのセグメントにアクセスできない。
  - **TSS。**DPL は、現在実行しているプログラムまたはタスクがそのレベルであることができ、なお TSS にアクセスできる数値として最も大きい特権レベルを示す。（これは、データ・セグメントに対するのと同じアクセス規則である。）
- **要求される特権レベル**（RPL: Requested Privilege Level）。RPL とは、セグメント・セレクタに割り当てられるオーバーライド特権レベルである。これは、セグメント・セレクタのビット 0 とビット 1 にストアされている。プロセッサは、RPL を CPL と共にチェックして、セグメントへのアクセスが許可されるかどうか判断する。セグメントへのアクセスを要求しているプログラムまたはタスクがそのセグメントにアクセスするのに十分な特権を持っている場合でも、RPL が十分な特

権レベルではないとアクセスは拒否される。つまり、セグメント・セレクタのRPLがCPLより数値として大きい場合は、RPLがCPLに優先し、その逆も成立する。RPLを使用して、アプリケーション・プログラム自体がそのセグメントに対するアクセス特権を持っていない限り、特権レベルの高いコードがアプリケーション・プログラムのためにセグメントにアクセスしないことを保証できる。RPLの目的と一般的な使用法の詳細については、4.10.4. 項「呼び出し側のアクセス特権のチェック（ARPL命令）」を参照。

セグメント・ディスクリプタのセグメント・セレクタがセグメント・レジスタにロードされると、特権レベルがチェックされる。データアクセスに使用されるチェックは、コード・セグメント間のプログラム制御の移行に使用されるチェックとは異なるため、2種類のアクセスを以降の節で別々に検討する。

4.6. データ・セグメントにアクセスする際の特権レベルチェック

データ・セグメントにあるオペランドにアクセスするには、データ・セグメントのセグメント・セレクタをデータ・セグメント・レジスタ（DS、ES、FS、GS）またはスタック・セグメント・レジスタ（SS）にロードしなければならない。（MOV、POP、LDS、LES、LFS、LGS、LSS命令を使用してセグメント・レジスタにロードすることができる。）プロセッサは、セグメント・セレクタをセグメント・レジスタにロードする前に、現在実行しているプログラムまたはタスクの特権レベル（CPL）、セグメント・セレクタのRPL、およびセグメントのセグメント・ディスクリプタのDPLを比較して、特権チェックを実行する（図4-3. を参照）。DPLがCPLとRPLより数値として大きいかまたは等しい場合は、プロセッサはセグメント・セレクタをセグメント・レジスタにロードする。それ以外の場合は、一般保護フォルトが発生し、セグメント・レジスタにはロードされない。

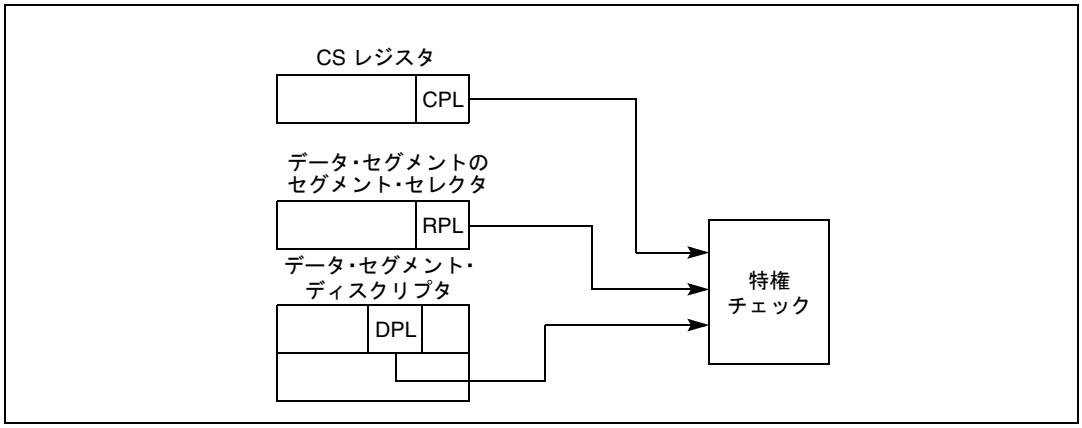


図 4-3. データアクセスのための特権チェック

図4-4.に、（コード・セグメントA、B、C、Dにある）4つのプロシージャを示す。それぞれは異なる特権レベルで実行し、それぞれが同じデータ・セグメントにアクセスしようとしている。

- コード・セグメントAのCPLとセグメント・セレクタE1のRPLがデータ・セグメントEのDPLに等しいので、コード・セグメントAにあるプロシージャは、セグメント・セレクタE1を使用してデータ・セグメントEにアクセスできる。
- コード・セグメントBのCPLとセグメント・セレクタE2のRPLが両方ともデータ・セグメントEのDPLより数値として小さい（特権レベルが高い）ので、コード・セグメントBにあるプロシージャは、セグメント・セレクタE2を使用してデータ・セグメントEにアクセスできる。コード・セグメントBのプロシージャも、セグメント・セレクタE1を使用してデータ・セグメントEにアクセスできる。
- コード・セグメントCのCPLとセグメント・セレクタE3のRPLが両方ともデータ・セグメントEのDPLより数値として大きい（特権レベルが低い）ので、コード・セグメントCにあるプロシージャは、セグメント・セレクタE3（点線）を使用してデータ・セグメントEにアクセスできない。RPLが受け入れられるようにコード・セグメントCのプロシージャがセグメント・セレクタE1またはE2を使用したとしても、そのCPLに十分な特権がないので、なおデータ・セグメントEにアクセスできない。
- コード・セグメントDのCPLはデータ・セグメントEのDPLより数値として小さいので、コード・セグメントDにあるプロシージャは、データ・セグメントEにアクセスできるはずである。しかし、（コード・セグメントDのプロシージャがデータ・セグメントEへのアクセスに使用している）セグメント・セレクタE3のRPLは、データ・セグメントEのDPLより数値として大きいため、アクセスは許可されない。コード・セグメントDのプロシージャがデータ・セグメントへのアクセスにセグメント・セレクタE1またはE2を使用すると、アクセスは許可される。

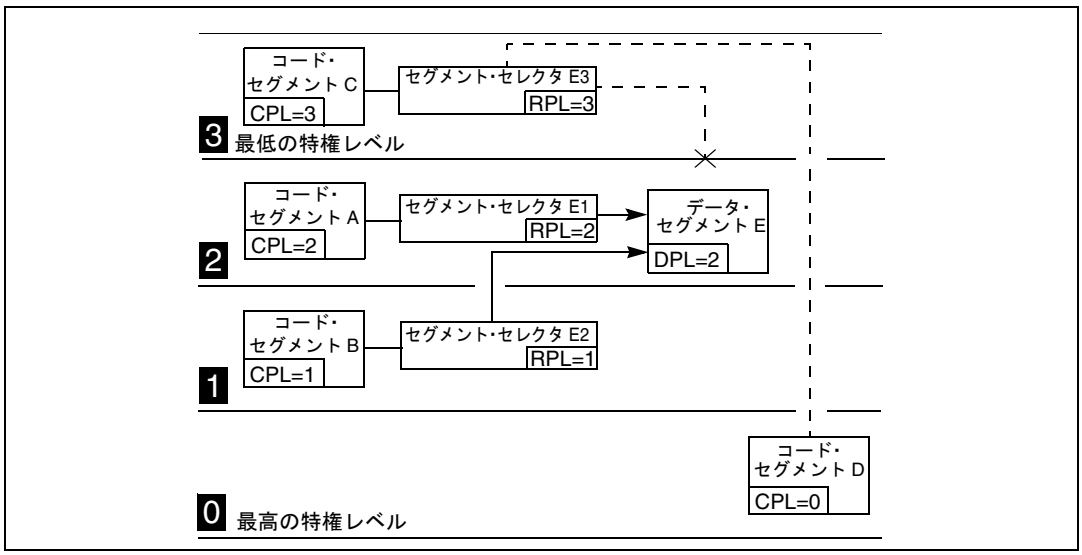


図 4-4. さまざまな特権レベルからデータ・セグメントにアクセスする例

前の例に示されているように、そのCPLが変わると、プログラムまたはタスクのアドレス指定可能なドメインは変わる。CPLが0であるときは、すべての特権レベルにあるデータ・セグメントにアクセスできる。CPLが1であるときは、特権レベル1～3にあるデータ・セグメントだけにアクセスする。CPLが3であるときは、特権レベル3にあるデータ・セグメントだけにアクセスする。

セグメント・セレクタのRPLは、常にプログラムまたはタスクのアドレス指定可能なドメインをオーバーライドできる。適切に使用されていると、RPLは、特権レベルの低いプログラムまたはプロシージャが特権レベルの高いデータ・セグメントのセグメント・セレクタを間違って（または意図的に）使用することによって生じる問題を防ぐことができる。

データ・セグメントのセグメント・セレクタのRPLは、ソフトウェアによって制御されるのに注意することは重要である。例えば、CPL 3で実行しているアプリケーション・プログラムは、データ・セグメント・セレクタのRPLを0に設定できる。RPLが0に設定されていると、RPLチェックではなくCPLチェックによってのみ、データ・セグメントの特権レベル・セキュリティに違反する意図的で直接的な試みに対する保護が可能になる。これらのタイプの特権レベルチェック違反を防ぐため、プログラムまたはプロシージャは、別のプロシージャからデータ・セグメント・セレクタを受け取ったときにアクセス特権をチェックできる（4.10.4.項「呼び出し側のアクセス特権のチェック（ARPL命令）」を参照）。

4.6.1. コード・セグメント内のデータへのアクセス

コード・セグメントに含まれているデータ構造にアクセスすることが望ましい場合もある。コード・セグメントにあるデータにアクセスするには、次の方法を使用できる。

- データ・セグメント・レジスタに非コンフォーミングの読み取り可能なコード・セグメントのセグメント・セレクタをロードする。
- データ・セグメント・レジスタにコンフォーミングの読み取り可能なコード・セグメントのセグメント・セレクタをロードする。
- コード・セグメント・オーバーライド・プリフィックス（CS）を使用して、対応するセレクタがすでにCS レジスタにロードされている読み取り可能なコード・セグメントを読み取る。

データ・セグメントをアクセスする場合と同じ規則が方法1に適用される。方法2は、そのDPLに関係なくコンフォーミング・コード・セグメントの特権レベルがCPLと実効的に同じであるために、常に有効である。方法3は、CS レジスタによって選択されたコード・セグメントのDPLがCPLと同じであるために、常に有効である。

4.7. SS レジスタにロードする際の特権レベルチェック

SS レジスタにスタック・セグメントのセグメント・セレクタをロードする際にも、特権レベルチェックが実行される。この場合には、そのスタック・セグメントに関係するすべての特権レベルは、CPL と一致していなければならない。つまり、CPL、スタック・セグメント・セレクタのRPL、およびスタック・セグメント・ディスクリプタのDPLが等しくなければならない。RPL とDPLがCPLに等しくない場合は、一般保護例外（#GP）が発生する。

4.8. コード・セグメント間でプログラム制御を移行する際の特権レベルチェック

プログラム制御を1つのコード・セグメントから別のコード・セグメントに移行するには、デスティネーション・コード・セグメントのセグメント・セレクタがコード・セグメント・レジスタ（CS）にロードされていなければならない。このロードプロセスの一部として、プロセッサは、デスティネーション・コード・セグメントのセグメント・ディスクリプタを調べて、さまざまなリミット、タイプ、および特権のチェックを実行する。これらのチェックにパスすると、CS レジスタにロードされ、プログラム制御が新しいコード・セグメントに移行され、プログラム実行がEIP レジスタによって指されている命令で開始される。

プログラム制御の移行は、JMP、CALL、RET、SYSENTER、SYSEXIT、INT *n*、IRET 命令によって実行され、さらに例外メカニズムと割り込みメカニズムによっても実行される。例外、割り込み、IRET 命令は特殊なケースであり、これらについては第5章「割り込みと例外の処理」で説明している。本章では、JMP、CALL、RET、SYSENTER、SYSEXIT 命令についてだけ説明する。

JMP 命令または CALL 命令は、次の 4 つの方法のいずれかで、別のコード・セグメントを参照できる。

- ターゲット・オペランドがターゲット・コード・セグメントのセグメント・セレクタを含んでいる。
- ターゲット・オペランドがコール・ゲート・ディスクリプタを指し、このディスクリプタがターゲット・コード・セグメントのセグメント・セレクタを含んでいる。
- ターゲット・オペランドが TSS を指し、この TSS がターゲット・コード・セグメントのセグメント・セレクタを含んでいる。
- ターゲット・オペランドがタスク・ゲートを指し、このゲートが TSS を指し、この TSS がターゲット・コード・セグメントのセグメント・セレクタを含んでいる。

以降の項では、最初の 2 つのタイプの参照について説明する。タスクゲートや TSS を通じたプログラム制御移行の詳細については、6.3. 節「タスク・スイッチング」を参照。

SYSENTER 命令と SYSEXIT 命令は、オペレーティング・システムやエグゼクティブのプロシージャに対して、高速呼び出しおよび高速リターンを行うための特別な命令である。これらの命令については、4.8.7. 項「SYSENTER 命令と SYSEXIT 命令によるシステム・プロシージャへの 高速呼び出しの実行」で簡単に説明している。

4.8.1. コード・セグメントへの直接呼び出しまたはジャンプ

near 型の JMP、CALL、RET 命令は、現在のコード・セグメント内でプログラム制御を移行するので、特権レベルチェックは実行されない。far 型の JMP、CALL、RET 命令は、別のコード・セグメントに制御を移行するので、プロセッサは特権レベルチェックを実行する。

コールゲートを通さないでプログラム制御を別のコード・セグメントに移行するときは、プロセッサは 4 種類の特権レベルとタイプ情報を調べる（図 4-5. を参照）。

- CPL（この場合には、CPL は呼び出し側コード・セグメント、つまり、呼び出しまたはジャンプを実行しているプロシージャを含むコード・セグメントの特権レベルである。）

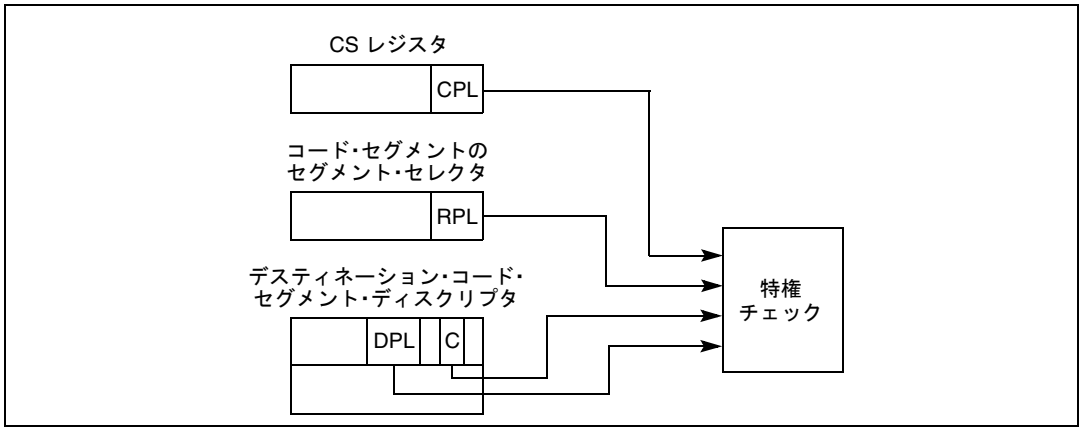


図 4-5. ゲートを使用しない制御移行の場合の特権チェック

- 呼び出し先プロシージャを含むデスティネーション・コード・セグメントのセグメント・ディスクリプタの DPL
- デスティネーション・コード・セグメントのセグメント・セレクタの RPL
- デスティネーション・コード・セグメントのセグメント・ディスクリプタ内にあるコンフォーミング（C）フラグ。このフラグは、セグメントがコンフォーミング・コード・セグメントであるか（C フラグがセット）、または非コンフォーミング・コード・セグメントであるか（C フラグがクリア）を決定する。（このフラグの詳細については、3.4.3.1. 項「セグメント・ディスクリプタのタイプ - コードとデータ」を参照。）

プロセッサが CPL、RPL、DPL のチェックに使用する規則は、以降の項で説明するように、C フラグの設定によって変わる。

4.8.1.1. 非コンフォーミング・コード・セグメントへのアクセス

非コンフォーミング・コード・セグメントにアクセスするときは、呼び出し側プロシージャの CPL は、デスティネーション・コード・セグメントの DPL に等しくなければならない。そうでない場合は、プロセッサは一般保護例外（#GP）を生成する。

例えば、図 4-6. では、コード・セグメント C は非コンフォーミング・コード・セグメントである。したがって、コード・セグメント A にあるプロシージャは、（セグメント・セレクタ C1 を使用して）コード・セグメント C にあるプロシージャを呼び出すことができる。それらが同じ特権レベルにある（コード・セグメント A の CPL はコード・セグメント C の DPL に等しい）ためである。しかし、コード・セグメント B にあるプロシージャは、（セグメント・セレクタ C2 または C1 を使用して）コード・セグメント C にあるプロシージャを呼び出すことはできない。2 つのコード・セグメントが異なる特権レベルにあるためである。

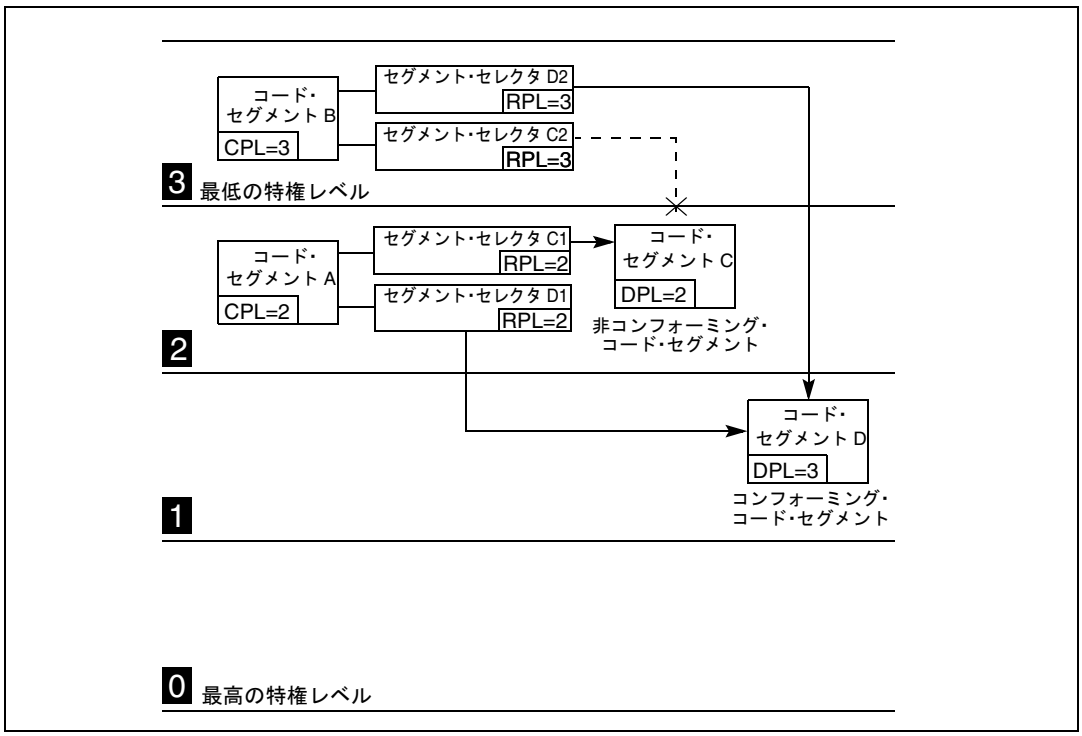


図 4-6. さまざまな特権レベルからコンフォーミングと非コンフォーミングの
コード・セグメントにアクセスする例

非コンフォーミング・コード・セグメントを指しているセグメント・セレクタの RPL は、特権チェックを実行しても限られた効果しか持たない。制御移行が正常に実行されるには、RPL は呼び出し側プロシージャの CPL より数値として小さいかまたは等しくなければならない。そのため、図 4-6. の例では、セグメント・セレクタ C1 と C2 の RPL が 0、1、または 2 に設定されていれば条件は満たされるが、3 に設定されていれば無効である。

非コンフォーミング・コード・セグメントのセグメント・セレクタが CS レジスタにロードされても、特権レベル・フィールドは変化しない。つまり、（呼び出し側プロシージャの特権レベルである）CPL のままである。セグメント・セレクタの RPL が CPL と異なっている場合でも、同じことがいえる。

4.8.1.2. コンフォーミング・コード・セグメントへのアクセス

コンフォーミング・コード・セグメントにアクセスするときは、呼び出し側プロシージャのCPLは、デスティネーション・コード・セグメントのDPLに数値として等しいかまたは大きく（特権レベルが低く）てもよい。CPLがDPLより小さい場合だけに、プロセッサは一般保護例外（#GP）を生成する。（セグメントがコンフォーミング・コード・セグメントである場合は、デスティネーション・コード・セグメントのセグメント・セレクタRPLはチェックされない。）

図4-6.の例では、コード・セグメントDはコンフォーミング・コード・セグメントである。したがって、両方のコード・セグメントAとBにある呼び出し側プロシージャは、（それぞれセグメント・セレクタD1またはD2を使用して）コード・セグメントDにアクセスできる。両方ともコンフォーミング・コード・セグメントのDPL以上であるCPLを持っているためである。**コンフォーミング・コード・セグメントの場合には、DPLは、コード・セグメントへの呼び出しを正常に実行するために呼び出し側プロシージャに許される、数値として最も小さい特権レベルを表している。**

（セグメント・セレクタD1とD2は、それぞれのRPLを除いて同じであることに注意が必要である。ただし、コンフォーミング・コード・セグメントにアクセスするときはRPLはチェックされないので、2つのセグメント・セレクタは本質的に相互交換可能である。）

プログラム制御をコンフォーミング・コード・セグメントに移行するとき、デスティネーション・コード・セグメントのDPLがCPLより小さい場合でも、CPLは変化しない。CPLが現在のコード・セグメントのDPLと異なってもよいのは、この状況だけである。また、CPLが変化しないので、スタックスイッチは実行されない。

コンフォーミング・セグメントは、マス・ライブラリや例外ハンドラなどのコード・モジュールに使用され、これらのモジュールはアプリケーションをサポートするが、保護されたシステム機能へのアクセスを必要としない。これらのモジュールは、オペレーティング・システムまたはエグゼクティブ・ソフトウェアの一部であるが、数値として大きい特権レベル（低い特権レベル）で実行する。コンフォーミング・コード・セグメントに切り替えるときにCPLを呼び出し側コード・セグメントのレベルに保つと、アプリケーション・プログラムがコンフォーミング・コード・セグメントの特権レベル（DPL）にある間に非コンフォーミング・コード・セグメントにアクセスできず、したがって、より高い特権レベルのデータにもアクセスできない。

ほとんどのコード・セグメントは非コンフォーミングである。これらのセグメントでは、以降の項で説明するように、移行がコールゲートを通じて実行されない限り、プログラム制御は同じ特権レベルのコード・セグメントだけに移行できる。

4.8.2. ゲート・ディスクリプタ

異なる特権レベルのコード・セグメントに制御されたアクセスを行うため、プロセッサは、ゲート・ディスクリプタという特殊な一組のディスクリプタを備えている。ゲート・ディスクリプタには次の4種類がある。

- コールゲート
- トラップゲート
- 割り込みゲート
- タスクゲート

タスクゲートは、タスク・スイッチングに使用され、第6章「タスク管理」で説明している。トラップゲートと割り込みゲートは、例外ハンドラと割り込みハンドラの呼び出しに使用される特殊なコールゲートである。これらについては、第5章「割り込みと例外の処理」で説明している。本章では、コールゲートについてだけ説明する。

4.8.3. コールゲート

コールゲートによって、異なる特権レベル間でのプログラム制御の制御された移行が容易になる。このゲートは、一般的には特権レベル保護メカニズムを使用するオペレーティング・システムまたはエグゼクティブだけで使用される。コールゲートは、16ビットと32ビットのコード・セグメント間のプログラム制御の移行にも役立つ。これらについては、17.4.節「異なるサイズのコード・セグメント間での制御の 移行」で説明している。

図4-7.に、コール・ゲート・ディスクリプタのフォーマットを示す。コール・ゲート・ディスクリプタは、GDT または LDT に常駐させてもよいが、割り込みディスクリプタ・テーブル（IDT）には常駐させることはできない。コール・ゲート・ディスクリプタは、次の6つの機能を実行する。

- アクセスされるコード・セグメントを指定する。
- 指定されたコード・セグメントにあるプロシージャ・エントリ・ポイントを定義する。
- 呼び出し側がプロシージャのアクセスを試みるために必要な特権レベルを指定する。

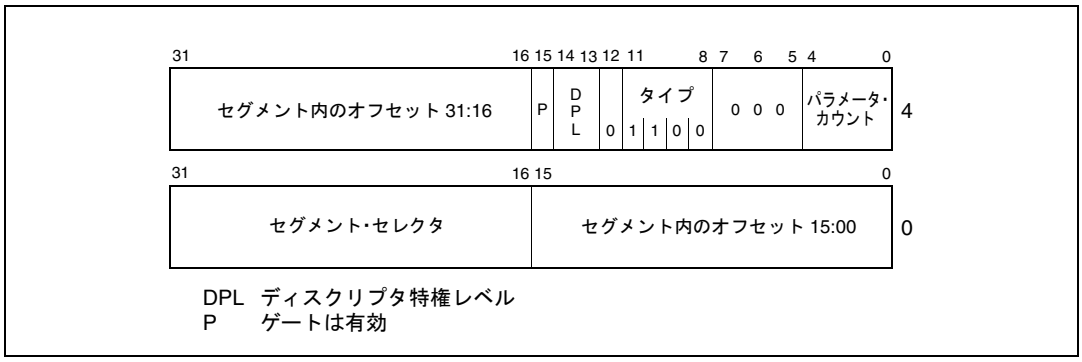


図 4-7. コール・ゲート・ディスクリプタ

- スタックスイッチが実行される場合は、スタック間でコピーされるオプションのパラメータの数を指定する。
- ターゲット・スタックにプッシュされる値のサイズを定義する。16 ビット・ゲートは16ビット・プッシュを強制し、32ビット・ゲートは32ビット・プッシュを強制する。
- コール・ゲート・ディスクリプタが有効であるかどうかを指定する。

コールゲート内のセグメント・セレクタ・フィールドでは、アクセスされるコード・セグメントを指定する。オフセット・フィールドでは、コード・セグメント内のエントリポイントを指定する。このエントリポイントは、一般的には特定のプロシージャの最初の命令に対するものである。DPLフィールドは、コールゲートの特権レベルを示し、これがゲートを通じて選択されているプロシージャにアクセスするために必要な特権レベルである。P フラグは、コール・ゲート・ディスクリプタが有効であるかどうかを示す。（ゲートが指しているコード・セグメントの存在は、コード・セグメントのディスクリプタにある P フラグによって示される。）パラメータ・カウント・フィールドは、スタックスイッチが実行される場合に呼び出し側プロシージャのスタックから新しいスタックにコピーするパラメータの数を示す（4.8.5.項「スタック・スイッチング」を参照）。パラメータ・カウントは、16ビットのコールゲートではワード数を指定し、32ビットのコールゲートではダブルワード数を指定する。

ゲート・ディスクリプタ内の P フラグは、通常は常に 1 にセットすることに注意しなければならない。0 に設定すると、プログラムがディスクリプタにアクセスしようとすると、不在（#NP）例外が発生する。オペレーティング・システムは、特殊な目的のために P フラグを使用できる。例えば、ゲートが使用された回数の記録に使用する。この場合には、P フラグは最初に 0 に設定され、不在例外ハンドラへのトラップを発生させる。例外ハンドラは、その後、カウンタをインクリメントし、P フラグを 1 にセットするので、ハンドラからの戻り時に、ゲート・ディスクリプタは有効になっている。

4.8.4. コールゲートを通じたコード・セグメントへのアクセス

コールゲートにアクセスするため、ゲートへのfarポインタがCALL命令またはJMP命令のターゲット・オペランドとして用意されている。このポインタからのセグメント・セレクタがコールゲートを識別する（図4-8.を参照）。ポインタからのオフセットは必須であるが、プロセッサによって使用またはチェックされない。（オフセットを任意の値に設定できる。）

プロセッサは、コールゲートにアクセスした後、コールゲートからのセグメント・セレクタを使用して、デスティネーション・コード・セグメントのセグメント・ディスクリプタの位置を確認する。（このセグメント・ディスクリプタは、GDTまたはLDTのどちらにあってもよい。）その後、コード・セグメント・ディスクリプタからのベースアドレスをコールゲートからのオフセットと組み合わせて、コード・セグメントにあるプロシージャ・エントリ・ポイントのリニアアドレスを形成する。

図4-9.に示すように、コールゲートを通じたプログラム制御移行の妥当性チェックには、次の4つの異なる特権レベルが使用される。

- CPL（現行特権レベル）
- コールゲートのセレクタのRPL（リクエスタ特権レベル）
- コール・ゲート・ディスクリプタのDPL（ディスクリプタ特権レベル）
- デスティネーション・コード・セグメントのセグメント・ディスクリプタのDPL

デスティネーション・コード・セグメントのセグメント・ディスクリプタ内のC（コンフォーミング）フラグもチェックされる。

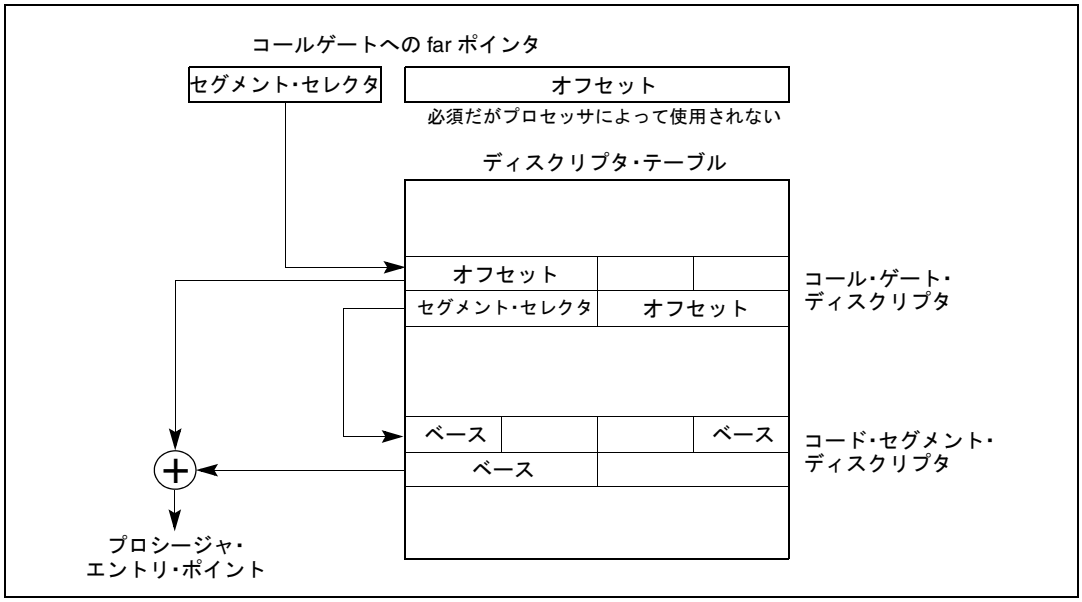


図 4-8. コールゲートのメカニズム

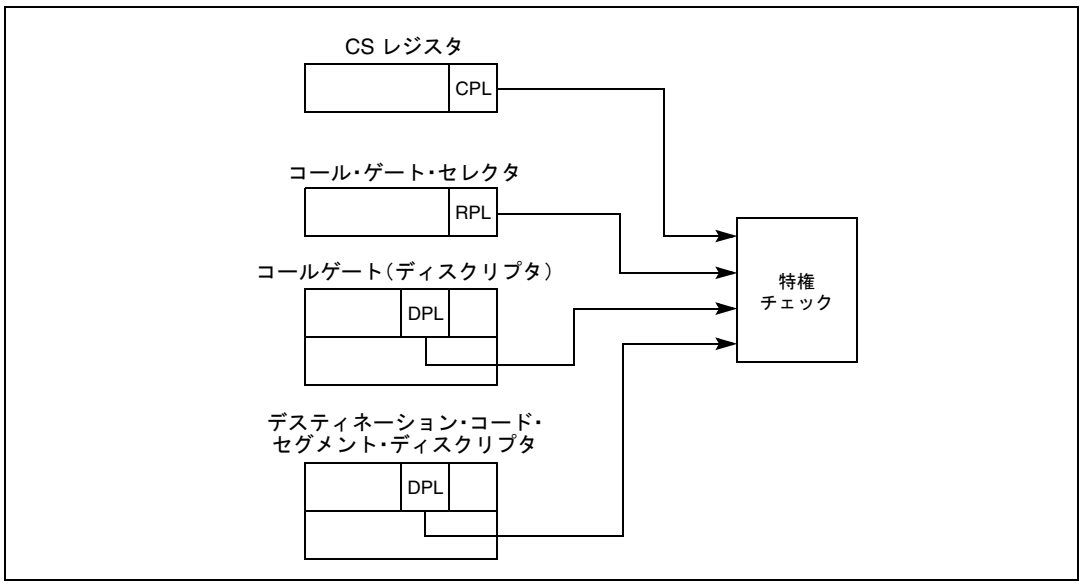


図 4-9. コールゲートを使用した制御移行の場合の特権チェック

特権チェック規則は、表 4-1. に示すように、制御移行が CALL 命令または JMP 命令のいずれによって開始されたかに応じて異なる。

表 4-1. コールゲートを使用した制御移行の場合の特権チェック

命令	特権チェック規則
CALL	CPL ≦ コールゲート DPL、RPL ≦ コールゲート DPL デスティネーション・コンフォーミング・コード・セグメント DPL ≦ CPL デスティネーション非コンフォーミング・コード・セグメント DPL ≦ CPL
JMP	CPL ≦ コールゲート DPL、RPL ≦ コールゲート DPL デスティネーション・コンフォーミング・コード・セグメント DPL ≦ CPL デスティネーション非コンフォーミング・コード・セグメント DPL = CPL

コール・ゲート・ディスクリプタの DPL フィールドは、呼び出し側プロシージャがコールゲートにアクセスできるための数値として最も大きい特権レベルを指定する。つまり、コールゲートにアクセスするには、呼び出し側プロシージャの CPL は、そのコールゲートの DPL 以下でなければならない。例えば、図 4-12. では、コールゲート A の DPL は 3 である。したがって、すべての CPL（0 ～ 3）の呼び出し側プロシージャがこのコールゲートにアクセスできる。これには、コード・セグメント A、B、C にある呼び出し側プロシージャが含まれる。コールゲート B の DPL は 2 であるので、CPL が 0、1、または 2 の呼び出し側プロシージャだけがコールゲート B にアクセスできる。これには、コード・セグメント B と C にある呼び出し側プロシージャが含まれる。点線は、コード・セグメント A にある呼び出し側プロシージャはコールゲート B にアクセスできないことを示している。

コールゲートへのセグメント・セレクタの RPL は、呼び出し側プロシージャの CPL と同じテストを満たしていなければならない。つまり、この RPL は、コールゲートの DPL 以下でなければならない。図 4-12. の例では、コード・セグメント C にある呼び出し側プロシージャは、ゲートセレクタ B2 または B1 を使用して、コールゲート B にアクセスできるが、ゲートセレクタ B3 を使用して、コールゲート B にアクセスはできない。

呼び出し側プロシージャとコールゲートとの間の特権チェックが正常である場合は、プロセッサは、コード・セグメント・ディスクリプタの DPL を呼び出し側プロシージャの CPL に対してチェックする。この場合に、特権チェック規則は、CALL 命令と JMP 命令とでは異なる。CALL 命令だけがコールゲートを使用して、より特権レベルの高い（数値として小さい特権レベル）非コンフォーミング・コード・セグメント、つまり、CPL より小さい DPL の非コンフォーミング・コード・セグメントにプログラム制御を移行できる。JMP 命令は、コールゲートを使用して、DPL が CPL に等しい非コンフォーミング・コード・セグメントだけにプログラム制御を移行できる。CALL 命令と JMP 命令は、両方ともより特権レベルの高いコンフォーミング・コード・セグメント、つまり、DPL が CPL 以下であるコンフォーミング・コード・セグメントにプログラム制御を移行できる。

呼び出しがより特権レベルの高い（数値として小さい特権レベル）非コンフォーミング・デスティネーション・コード・セグメントに実行されると、CPL は、デスティネーション・コード・セグメントの DPL に下げられ、スタックスイッチが実行される（4.8.5. 項「スタック・スイッチング」を参照）。呼び出しまたはジャンプがより特権レベルの高いコンフォーミング・デスティネーション・コード・セグメントに実行されると、CPL は変更されず、スタックスイッチは実行されない。

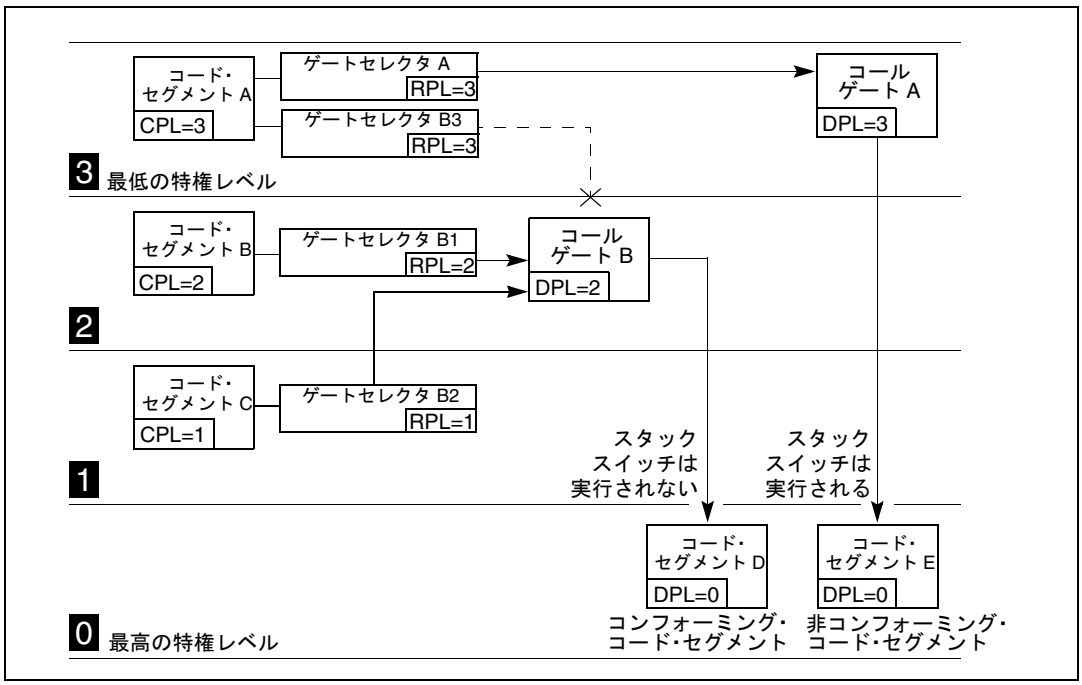


図 4-10. さまざまな特権レベルでコールゲートにアクセスする例

コールゲートは、1 つのコード・セグメントだけが異なる特権レベルでアクセスできるプロシージャを持てるようにする。例えば、コード・セグメントにあるオペレーティング・システムは、（キャラクタ I/O 処理のプロシージャなど）オペレーティング・システムとアプリケーション・ソフトウェアの両方が利用するためのサービスを持つことがある。これらのプロシージャのコールゲートは、すべての特権レベル（0 ～ 3）でのアクセスを認めるようにセットアップできる。その場合には、（デバイスドライバを初期化するプロシージャなど）オペレーティング・システムだけが利用するように意図されたその他のオペレーティング・システム・サービスに、より特権レベルの高いコールゲート（DPL が 0 または 1）をセットアップできる。

4.8.5. スタック・スイッチング

コールゲートを使用して、プログラム制御をより特権レベルの高い非コンフォーミング・コード・セグメントに移行する場合（つまり、非コンフォーミング・デスティネーション・コード・セグメントのDPLがCPLより小さい場合）は必ず、プロセッサはデスティネーション・コード・セグメントの特権レベルに対応するスタックに自動的に切り替える。このスタックスイッチは、より高い特権レベルのプロシージャがスタック空間の不足のためにクラッシュしないように実行される。また、低い特権レベルのプロシージャが共有スタックを通じて高い特権レベルのプロシージャに（間違ってまたは意図的に）干渉しないようにもしている。

各タスクは、4つまでのスタックを定義しなければならない。1つは（特権レベル3で動作する）アプリケーション・コード用であり、残りは使用される特権レベル2、1、0にそれぞれ1つずつである。（2つの特権レベル[3と0]しか使用しない場合は、2つのスタックだけを定義しなければならない。）これらの各スタックは、別々のセグメントに置かれ、セグメント・セレクタとスタック・セグメントへのオフセット（スタックポインタ）で識別される。

特権レベル3のスタックのセグメント・セレクタとスタックポインタは、特権レベル3のコードが実行されているときはそれぞれSSレジスタとESPレジスタに置かれ、スタックスイッチが実行されると呼び出し先プロシージャのスタックに自動的にストアされる。

特権レベル0、1、2のスタックへのポインタは、現在実行しているタスクのTSSにストアされる（図6-2.を参照）。これらの各ポインタは、セグメント・セレクタと（ESPレジスタにロードされた）スタックポインタで構成される。これらの初期ポインタは、厳密に読み取り専用値である。プロセッサは、タスクの実行中はこれらのポインタを変更しない。これらのポインタは、呼び出しがより高い特権レベル（数値として小さい特権レベル）に実行されるときに新しいスタックを作成するためだけに使用される。これらのスタックは、呼び出し先プロシージャからの戻りが実行されるときに処分される。プロシージャが次に呼び出されると、初期スタックポインタを使用して新しいスタックが作成される。（戻り時を除いて、CPL 0、1、または2で実行しているプロシージャからCPL 3で実行しているプロシージャへのプログラム制御移行をプロセッサは認めないので、TSSは特権レベル3のスタックを指定しない。）

オペレーティング・システムには、使用されるすべての特権レベルにスタックとスタック・セグメント・ディスクリプタを作成し、これらのスタックの初期ポインタをTSSにロードする責任がある。各スタックは、（そのセグメント・ディスクリプタのタイプ・フィールドに指定されるように）読み取り/書き込みアクセス可能でなければならず、次の項目を保持するのに十分な（リミット・フィールドに指定された）空間を持っていなければならない。

- 呼び出し側プロシージャのSS、ESP、CS、EIPレジスタの内容

- 呼び出し先プロシージャが必要とするパラメータと一時変数
- 暗黙の呼び出しが例外ハンドラまたは割り込みハンドラに実行されるとき、EFLAGS レジスタとエラーコード

プロシージャは他のプロシージャを呼び出すことがよくあり、オペレーティング・システムは複数の割り込みのネスティングをサポートすることもあるため、スタックは、これらの項目のスタックフレームを持つのに十分な空間を要求する必要がある。各スタックは、その特権レベルにおける最悪のネスティング状況に対処するのに十分な空間を持っていなければならない。

(オペレーティング・システムは、プロセッサのマルチタスキング・メカニズムを使用しない場合でも、このスタック関連の目的のために少なくとも 1 つの TSS を作成しなければならない。)

コールゲートを通じたプロシージャ呼び出しによって特権レベルが変わると、プロセッサは、次の手順を実行して、スタックを切り替え、新しい特権レベルで呼び出し先プロシージャの実行を開始する。

1. デスティネーション・コード・セグメントの DPL（新しい CPL）を使用して、TSS から新しいスタックへのポインタ（セグメント・セレクタとスタックポインタ）を選択する。
2. 現在の TSS から切り替わる先のスタックのセグメント・セレクタとスタックポインタを読み取る。スタック・セグメント・セレクタ、スタックポインタ、またはスタック・セグメント・ディスクリプタを読み取る間にリミット違反が検出されると、無効 TSS（#TS）例外を生成する。
3. 特権とタイプが適切であるかスタック・セグメント・ディスクリプタをチェックし、違反が検出されると無効 TSS（#TS）例外を生成する。
4. SS レジスタと ESP レジスタの現在の値を一時的にセーブする。
5. 新しいスタックのセグメント・セレクタとスタックポインタを SS レジスタと ESP レジスタにロードする。
6. （呼び出し側プロシージャの）SS レジスタと ESP レジスタの一時的にセーブした値を新しいスタックにプッシュする（図 4-11. を参照）。
7. コールゲートのパラメータ・カウント・フィールドに指定されているパラメータの数を呼び出し側プロシージャのスタックから新しいスタックにコピーする。カウントが 0 である場合は、パラメータはコピーされない。
8. 戻り命令ポインタ（CS レジスタと EIP レジスタの現在の内容）を新しいスタックにプッシュする。

9. 新しいコード・セグメントのセグメント・セレクタと新しい命令ポインタをコールゲートからそれぞれ CS レジスタと EIP レジスタにロードし、呼び出し先プロシージャの実行を開始する。

プロセッサがコールゲートを通じた far コール時に実行する特権レベルチェックとその他の保護チェックの詳細については、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」にある CALL 命令の説明を参照。

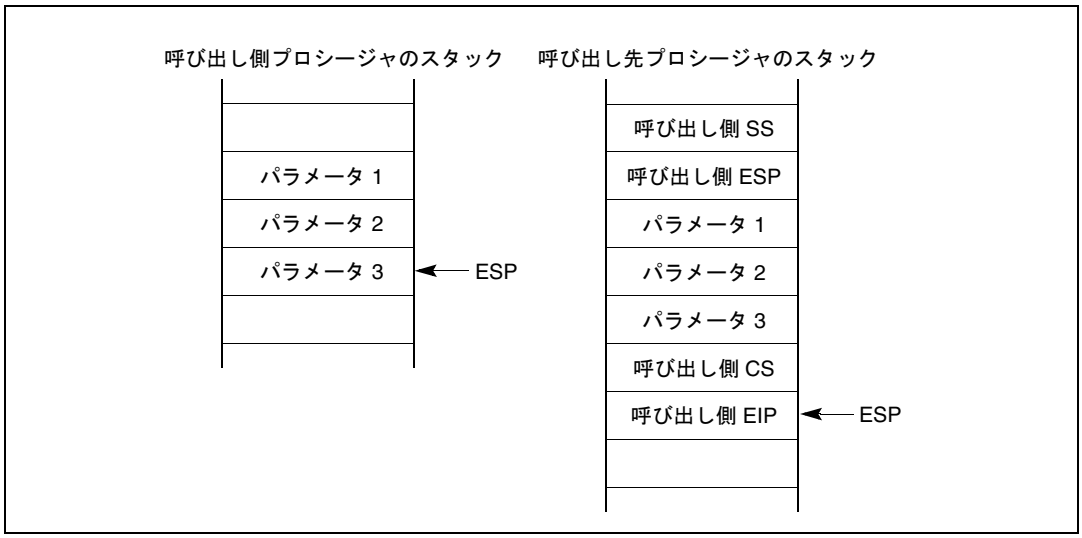


図 4-11. 特権レベル間呼び出しの間のスタック・スイッチング

コールゲート内のパラメータ・カウント・フィールドでは、プロセッサが呼び出し側プロシージャのスタックから呼び出し先プロシージャのスタックにコピーする必要があるデータ項目数（最大 31）を指定する。32 以上のデータ項目を呼び出し先プロシージャに渡す必要がある場合は、パラメータの 1 つをデータ構造へのポインタにできる。または、SS レジスタと ESP レジスタのセーブされた内容を使用して、古いスタック空間にあるパラメータにアクセスしてもよい。呼び出し先プロシージャに渡されるデータ項目のサイズは、4.8.3. 項「コールゲート」で説明しているように、コールゲートのサイズによって異なる。

4.8.6. 呼び出し先プロシージャからの戻り

RET 命令を使用して、同じ特権レベルで near リターン、far リターンを実行でき、異なる特権レベルで far リターンを実行できる。この命令は、CALL 命令で呼び出し先プロシージャからの戻りを実行するためのものである。JMP 命令では戻り命令ポインタをスタックにセーブしないので、RET 命令は JMP 命令からの戻りをサポートしていない。

near リターンは現在のコード・セグメント内でプログラム制御を移行するだけである。したがって、プロセッサは、リミットチェックだけを実行する。プロセッサは、戻り命令ポインタをスタックから EIP レジスタにポップするとき、ポインタが現在のコード・セグメントのリミットを超えていないかチェックする。

同じ特権レベルでの far リターンでは、プロセッサは、戻る先のコード・セグメントのセグメント・セレクタと戻り命令ポインタの両方をスタックからポップする。通常の状態では、これらのポインタは、CALL 命令によってスタックにプッシュされたものなので有効なはずである。ただし、プロセッサは、現在のプロシージャがポインタを変更したかまたはスタックを適切に維持しなかった状況を検出するために特権チェックを実行する。

特権レベル変更を必要とする far リターンは、低い特権のレベルに戻る（つまり、戻りコード・セグメントの DPL が CPL より数値として大きい）ときだけに認められる。プロセッサは、呼び出し側プロシージャのためにセーブされた CS レジスタ値（図 4-11. を参照）からの RPL フィールドを使用して、数値として大きい特権レベルへの戻りが必要かを判断する。RPL が CPL より数値として大きい（特権レベルが低い）場合に、特権レベルをまたがる戻りが実行される。

プロセッサは、呼び出し側プロシージャへの far リターンを実行するときに、次の手順を実行する（戻り前と戻り後のスタックの内容の図については、『IA-32 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』の図 6-2. と図 6-4. を参照）。

1. セーブされた CS レジスタ値の RPL フィールドをチェックして、特権レベルの変更が戻り時に必要か判断する。
2. CS レジスタと EIP レジスタに呼び出し先プロシージャのスタックにある値をロードする。（タイプチェックと特権レベルチェックがコード・セグメント・ディスクリプタとコード・セグメント・セレクタの RPL に対して実行される。）
3. （RET 命令がパラメータ・カウント・オペランドを含んでおり、戻りが特権レベル変更を要求する場合）（RET 命令から得られるバイトにある）パラメータ・カウントを（CS 値と EIP 値をポップした後の）現在の ESP レジスタ値に加え、呼び出し先プロシージャのスタックにあるパラメータを飛び越える。ESP レジスタにある結果の値は、呼び出し側プロシージャのスタックのセーブされた SS 値と ESP 値を指す。（RET 命令にあるバイトカウントを選択して、パラメータのサイズによって乗算された呼び出し側プロ

シージャがオリジナルの呼び出しを実行したときに参照したコールゲートにあるパラメータ・カウントに一致させなければならないことに注意が必要である。）

4. （戻りが特権レベル変更を要求する場合）SS レジスタと ESP レジスタにセーブされた SS 値と ESP 値をロードし、呼び出し側プロシージャのスタックに切り替え直す。呼び出し先プロシージャのスタックの SS 値と ESP 値は廃棄される。スタック・セグメント・セレクタまたはスタックポインタをロードしている間にリミット違反が検出されると、一般保護例外（#GP）が発生する。新しいスタック・セグメント・ディスクリプタは、タイプと特権違反がないかもチェックされる。

5. （RET 命令がパラメータ・カウント・オペランドを含む場合）（RET 命令から得られるバイトにある）パラメータ・カウントを現在の ESP レジスタ値に加え、呼び出し側プロシージャのスタックにあるパラメータを飛び越える。ESP レジスタにある結果の値は、スタック・セグメントのリミットに対してチェックされない。ESP 値がリミットを超えていても、その事実は次のスタック操作まで認識されない。

6. （戻りが特権レベル変更を要求する場合）DS、ES、FS、GS の各セグメント・レジスタの内容をチェックする。（コンフォーミング・コード・セグメントを除いて）DPL が新しい CPL より小さいセグメントをこれらのレジスタのいずれかが参照していると、セグメント・レジスタにヌル・セグメント・セレクタがロードされる。

プロセッサが far リターン時に実行する特権レベルチェックとその他の保護チェックの詳細については、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」にある RET 命令の説明を参照。

4.8.7. SYSENTER 命令と SYSEXIT 命令によるシステム・プロシージャへの高速呼び出しの実行

SYSENTER 命令と SYSEXIT 命令は、インテル® Pentium® II プロセッサで IA-32 アーキテクチャに導入された。オペレーティング・システムやエグゼクティブのプロシージャを呼び出す際の高速な（オーバーヘッドの低い）メカニズムを提供することが目的である。SYSENTER 命令は、特権レベル 3 で実行中のユーザコードで使用して、特権レベル 0 で実行中のオペレーティング・システムやエグゼクティブのプロシージャにアクセスすることが目的である。SYSEXIT プロシージャは、特権レベル 0 のオペレーティング・システムやエグゼクティブのプロシージャが、特権レベル 3 のユーザコードへ高速リターンするために使用する。SYSENTER 命令は特権レベル 3、2、1 から実行できるが、SYSEXIT 命令は特権レベル 0 からしか実行できない。

SYSENTER 命令と SYSEXIT 命令はコンパニオン命令であるが、SYSENTER 命令は SYSEXIT命令がリターン時に使用するステート情報を保存しないため、コール/リターンのペアは形成されない。

これらの命令のターゲット命令とスタックポインタは、命令オペランドによって指定されるのではなく、いくつかの MSR と汎用レジスタに入力されたパラメータにより指定される。SYSENTER 命令の場合、プロセッサは、以下のソースから特権レベル 0 のターゲット命令とスタックポインタを取得する。

- ターゲット・コード・セグメント － SYSENTER_CS_MSR から読み取る。
- ターゲット命令 － SYSENTER_EIP_MSR から読み取る。
- スタック・セグメント － SYSENTER_CS_MSR の値に 8 を加算して算出する。
- スタックポインタ － SYSENTER_ESP_MSR から読み取る。

SYSEXIT 命令の場合、特権レベル 3 のターゲット命令とスタックポインタは、以下のようにして指定される。

- ターゲット・コード・セグメント － SYSENTER_CS_MSR の値に 16 を加算して算出する。
- ターゲット命令 － EDX レジスタから読み取る。
- スタック・セグメント － SYSENTER_CS_MSR の値に 24 を加算して算出する。
- スタックポインタ － ECX レジスタから読み取る。

SYSENTER 命令と SYSEXIT 命令は、「高速」な呼び出しおよびリターンを実行する。なぜなら、これらの命令によりプロセッサは、SYSENTER 命令の実行時は事前定義された特権レベル 0 の状態にされ、SYSEXIT 命令の実行時は事前定義された特権レベル 3 の状態にされるからである。事前定義された整合性のあるプロセッサ状態にすることにより、他の特権レベルに対する far コールを実行するために通常は必要とされる特権チェックの数が大幅に削減される。また、MSR および汎用レジスタにターゲットのコンテキストの状態を事前に定義しておくと、ターゲット・コードをフェッチする場合を除いて、すべてのメモリアクセスが不要になる。

呼び出し側プロシージャへのリターンが可能となるために保存する必要のある追加の状態情報については、呼び出し側プロシージャが明示的に保存するか、または、プログラミング規則を通じて事前定義する必要がある。

4.9. 特権命令

一部のシステム命令（「特権命令」という）は、アプリケーション・プログラムに使用されないように保護されている。特権命令は、（システムレジスタのロードなどの）システム機能を制御する。これらの命令は、CPL が（最も高い特権レベルの）0 であるときだけに実行できる。CPL が 0 でないときにこれらの命令のいずれかが実行されると、一般保護例外（#GP）が発生する。次のシステム命令は特権命令である。

- LGDT － GDT レジスタをロードする
- LLDT － LDT レジスタをロードする
- LTR － タスクレジスタをロードする
- LIDT － IDT レジスタをロードする
- MOV（制御レジスタ） － 制御レジスタをロードおよびストアする
- LMSW － マシン・ステータス・ワードをロードする
- CLTS － レジスタ CR0 のタスク・スイッチ・フラグをクリアする
- MOV（デバッグレジスタ） － デバッグレジスタをロードおよびストアする
- INVD － ライトバックなしでキャッシュを無効にする
- WBINVD － ライトバックありでキャッシュを無効にする
- INVLPG － TLB エントリを無効にする
- HLT － プロセッサを停止する
- RDMSR － モデル固有レジスタを読み取る
- WRMSR － モデル固有レジスタに書き込む
- RDPMC － 性能モニタリング・カウンタを読み取る
- RDTSC － タイムスタンプ・カウンタを読み取る

一部の特権命令は、IA-32 プロセッサの中のより新しいファミリでしか使用できない（18.10. 節「インテル® Pentium® プロセッサ以降の IA-32 プロセッサの新しい命令」を参照）。

レジスタ CR4 の PCE フラグと TSD フラグ（それぞれビット 4 と 2）は、それぞれ RDPMC 命令と RDTSC 命令を任意の CPL で実行できるようにする。

4.10. ポインタの妥当性チェック

プロセッサは、保護モードで動作しているときは、すべてのポインタについて妥当性チェックを実行し、セグメント間に保護を強制し、特権レベル間の分離を維持する。ポインタの妥当性チェックは、次のチェックで構成されている。

1. セグメント・タイプがその用途に適合しているかを判断するアクセス権のチェック
2. 読み取り / 書き込み権のチェック
3. ポインタ・オフセットがセグメント・リミットを超えていないかのチェック
4. ポインタのサプライアがセグメントへのアクセスを認められているかのチェック
5. オフセット・アライメントのチェック

プロセッサは、命令実行の間に 1、2、3 のチェックを自動的に実行する。ソフトウェアは、ARPL 命令を発行して 4 のチェックを明示的に要求しなければならない。5 のチェック（オフセット・アライメント）は、アライメント・チェックがオンになっていると特権レベル 3 で自動的に実行される。オフセット・アライメントは、特権レベルの分離には影響を与えない。

4.10.1. アクセス権のチェック（LAR 命令）

プロセッサは、far ポインタを使用してセグメントにアクセスするときに、far ポインタによって指されているセグメント・ディスクリプタにアクセス権チェックを実行する。このチェックは、セグメント・ディスクリプタのタイプと特権レベル（DPL）が実行される操作に適合しているかどうかを判断するために行われる。例えば、保護モードで far コールを実行するときは、セグメント・ディスクリプタ・タイプは、コンフォーミングまたは非コンフォーミング・コード・セグメント、コールゲート、タスクゲート、または TSS に対応するものでなければならない。その場合に、呼び出しが非コンフォーミング・コード・セグメントに対するものである場合は、コード・セグメントの DPL は CPL に等しくなければならず、コード・セグメントのセグメント・セレクタの RPL は DPL 以下でなければならない。タイプまたは特権レベルが適合していないことが判った場合には、それに該当する例外が発生する。

タイプ不適合例外の発生を防ぐため、ソフトウェアはアクセス権のロード（LAR: Load Access Rights）命令を使用して、セグメント・ディスクリプタのアクセス権をチェックすることができる。LAR 命令は、アクセス権をチェックするセグメント・ディスクリプタのセグメント・セレクタ、およびデスティネーション・レジスタを指定する。この命令は、次の操作を実行する。

1. セグメント・セレクタがヌルでないことをチェックする。

2. セグメント・セレクタがディスクリプタ・テーブル・リミット（GDT または LDT）内にあるセグメント・ディスクリプタを指していることをチェックする。
3. セグメント・ディスクリプタがコード、データ、LDT、コールゲート、タスクゲート、または TSS のセグメント・ディスクリプタ・タイプであることをチェックする。
4. セグメントがコンフォーミング・コード・セグメントでない場合は、セグメント・ディスクリプタが CPL で見えるかどうか（つまり、CPL とセグメント・セレクタの RPL が DPL 以下であるかどうか）をチェックする。
5. 特権レベルチェックとタイプチェックに合格すると、セグメント・ディスクリプタの第二ダブルワードを（値 00FXFF00H（X は対応する 4 ビットが未定義であることを示す）でマスクして）デスティネーション・レジスタにロードし、EFLAGS レジスタの ZF フラグをセットする。セグメント・セレクタが現行特権レベルで見えないか、または LAR 命令に対して無効なタイプである場合は、命令は、デスティネーション・レジスタを修正せず、ZF フラグをクリアする。

デスティネーション・レジスタにロードされると、ソフトウェアはアクセス権情報についてさらにチェックを実行することができる。

4.10.2. 読み取り / 書き込み権のチェック（VERR 命令と VERW 命令）

プロセッサは、コード・セグメントまたはデータ・セグメントにアクセスするときは、セグメントに割り当てられている読み取り / 書き込み特権をチェックして、意図された読み取りまたは書き込み操作が許可されることを確認する。ソフトウェアは、VERR（読み取りのための検証）命令と VERW（書き込みのための検証）命令を使用して、読み取り / 書き込み権をチェックできる。これらの命令は、両方ともチェックするセグメントのセグメント・セレクタを指定する。これらの命令は、次の操作を実行する。

1. セグメント・セレクタがヌルでないことをチェックする。
2. セグメント・セレクタがディスクリプタ・テーブル・リミット（GDT または LDT）内にあるセグメント・ディスクリプタを指していることをチェックする。
3. セグメント・ディスクリプタがコードまたはデータのセグメント・ディスクリプタ・タイプであることをチェックする。
4. セグメントがコンフォーミング・コード・セグメントでない場合は、セグメント・ディスクリプタが CPL で見えるかどうか（つまり、CPL とセグメント・セレクタの RPL が DPL 以下であるかどうか）をチェックする。
5. セグメントが読み取り可能（VERR 命令の場合）または書き込み可能（VERW 命令の場合）であることをチェックする。

VERR 命令は、セグメントが CPL で見えて読み取り可能であれば EFLAGS レジスタの ZF フラグをセットする。VERW 命令は、セグメントが見えて書き込み可能であれば

ZF フラグをセットする（コード・セグメントが書き込み可能であることはない）。これらのチェックのいずれかに合格しないと、ZF フラグはクリアされる。

4.10.3. ポインタ・オフセットがリミット内にあるかどうかのチェック（LSL 命令）

プロセッサは、任意のセグメントにアクセスするときに、リミットチェックを実行して、オフセットがセグメントのリミット内にあるようにする。ソフトウェアは、LSL（セグメント・リミットのロード）命令を使用して、このリミットチェックを実行できる。LAR 命令と同様に、LSL 命令は、リミットをチェックするセグメント・ディスクリプタのセグメント・セレクタ、およびデスティネーション・レジスタを指定する。この命令は、次の操作を実行する。

1. セグメント・セレクタがヌルでないことをチェックする。
2. セグメント・セレクタがディスクリプタ・テーブル・リミット（GDT または LDT）内にあるセグメント・ディスクリプタを指していることをチェックする。
3. セグメント・ディスクリプタがコード、データ、LDT、または TSS のセグメント・ディスクリプタ・タイプであることをチェックする。
4. セグメントがコンフォーミング・コード・セグメントでない場合は、セグメント・セレクタが CPL で見えるかどうか（つまり、CPL とセグメント・セレクタの RPL が DPL 以下であるかどうか）をチェックする。
5. 特権レベルチェックとタイプチェックに合格すると、スクランブルされていないリミット（セグメント・ディスクリプタの G フラグの設定にしたがってスケーリングされたリミット）をデスティネーション・レジスタにロードし、EFLAGS レジスタの ZF フラグをセットする。セグメント・セレクタが現行特権レベルで見えないか、または LSL 命令に対して無効なタイプである場合は、命令は、デスティネーション・レジスタを変更せず、ZF フラグをクリアする。

デスティネーション・レジスタにロードされると、ソフトウェアはセグメント・リミットをポインタのオフセットと比較できる。

4.10.4. 呼び出し側のアクセス特権のチェック（ARPL 命令）

セグメント・セレクタのリクエスタ特権レベル（RPL）フィールドは、呼び出し側プロシージャの特権レベル（呼び出し側プロシージャの CPL）を呼び出し先プロシージャに移行するためのものである。呼び出し先プロシージャは、RPL を使用してセグメントへのアクセスが認められるかどうか判断する。RPL は、呼び出し先プロシージャの特権レベルを RPL のレベルに「弱める」といわれる。

オペレーティング・システム・プロシージャは、一般的にはRPLを使用して、低い特権レベルのアプリケーション・プログラムがより高い特権レベルのセグメントにあるデータにアクセスしないようにする。オペレーティング・システム・プロシージャ（呼び出し先プロシージャ）は、セグメント・セレクタをアプリケーション・プログラム（呼び出し側プロシージャ）から受け取ると、セグメント・セレクタのRPLを呼び出し側プロシージャの特権レベルに設定する。その後、オペレーティング・システムがそのセグメント・セレクタを使用して関連するセグメントにアクセスすると、プロセッサは、オペレーティング・システム・プロシージャの数値として小さい特権レベル（CPL）ではなく、（RPLにストアされている）呼び出し側プロシージャの特権レベルを使用して特権チェックを実行する。したがって、RPLは、アプリケーション・プログラム自体がそのセグメントへのアクセス権を持っていない限り、オペレーティング・システムがアプリケーション・プログラムのためにセグメントにアクセスしないことを保証する。

図4-12.に、プロセッサがRPLフィールドを使用する方法を示す。この例では、（コード・セグメントAにある）アプリケーション・プログラムは、高い特権レベルのデータ構造（つまり、特権レベル0のデータ・セグメントDにあるデータ構造）を指しているセグメント・セレクタ（セグメント・セレクタD1）を持っている。

アプリケーション・プログラムは、十分な特権を持っていないのでデータ・セグメントDにアクセスできないが、（コード・セグメントCにある）オペレーティング・システムはアクセスできる。そのため、アプリケーション・プログラムは、データ・セグメントDにアクセスしようとして、オペレーティング・システムへの呼び出しを実行し、セグメント・セレクタD1をスタック上のパラメータとしてオペレーティング・システムに渡す。（よくできた）アプリケーション・プログラムは、セグメント・セレクタを渡す前に、セグメント・セレクタのRPLをその現行特権レベル（この例では3）に設定する。オペレーティング・システムがセグメント・セレクタD1を使用してデータ・セグメントDにアクセスしようとすると、プロセッサはCPL（呼び出し後の今では0)、セグメント・セレクタD1のRPL、データ・セグメントDのDPL（0である）を比較する。RPLがDPLより大きいので、データ・セグメントDへのアクセスは拒否される。したがって、（セグメント・セレクタBのRPLによって提示された）アプリケーション・プログラムの特権レベルがデータ・セグメントDのDPLより大きいので、プロセッサの保護メカニズムは、データ・セグメントDがオペレーティング・システムによってアクセスされないように保護する。

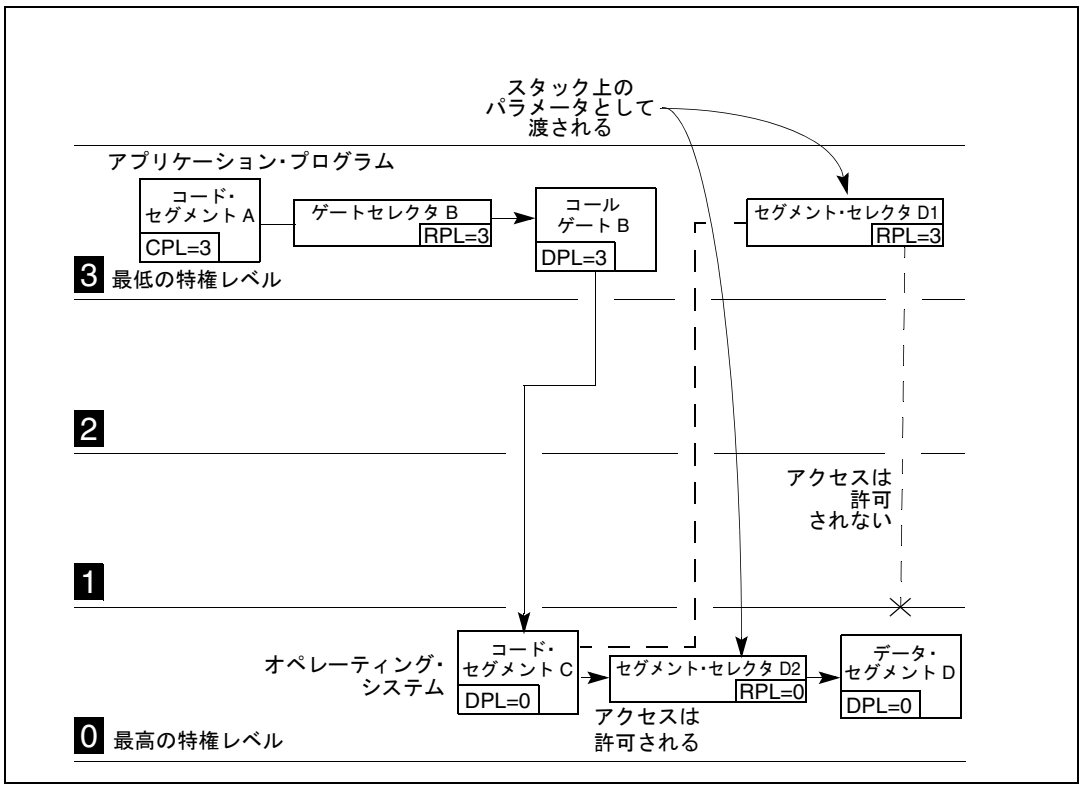


図 4-12. RPL を使用して呼び出し先プロシージャの特権レベルを弱める方法

今度は、アプリケーション・プログラムは、セグメント・セレクタの RPL を 3 に設定する代わりに、RPL を 0（セグメント・セレクタ D2）に設定すると想定する。今度はオペレーティング・システムの CPL とセグメント・セレクタ D2 の RPL が両方ともデータ・セグメント D の DPL に等しいので、オペレーティング・システムはデータ・セグメント D にアクセスできる。

アプリケーション・プログラムは、セグメント・セレクタの RPL を任意の値に変更できるので、数値として小さい特権レベルで動作するプロシージャを使用して、保護されたデータ構造にアクセスする能力を潜在的に持つことになる。セグメント・セレクタの RPL を小さくするこの能力は、プロセッサの保護メカニズムを侵害する。

呼び出し先プロシージャは、呼び出し側プロシージャに依存して RPL を正しく設定することができないので、数値として大きい特権レベルのプロシージャからセグメント・セレクタを受け取る（数値として小さい特権レベルで実行している）オペレーティング・システム・プロシージャは、セグメント・セレクタの RPL をテストして、それが適切なレベルにあるかどうかを判断する必要がある。ARPL（要求される特権レベルの調整）命令がこの目的のために用意されている。この命令は、1 つのセグメ

ント・セレクタのRPLがもう1つのセグメント・セレクタのRPLに一致するように調整する。

図4-12.の例では、ARPL命令がどのように使用されるものかを示す。オペレーティング・システムは、セグメント・セレクタD2をアプリケーション・プログラムから受け取ると、ARPL命令を使用して、セグメント・セレクタのRPLを（スタック上にプッシュされているコード・セグメント・セレクタによって提示される）アプリケーション・プログラムの特権レベルと比較する。RPLがアプリケーション・プログラムの特権レベルより小さいと、ARPL命令はアプリケーション・プログラム（セグメント・セレクタD1）の特権レベルと一致するようにセグメント・セレクタのRPLを変更する。したがって、この命令を使用すると、セグメント・セレクタのRPLを下げれば、数値として大きい特権レベルで実行しているプロシージャが数値として小さい特権レベル（高い特権レベル）のセグメントをアクセスすることを防げる。

アプリケーション・プログラムの特権レベルは、アプリケーション・プログラムのコード・セグメントに対応するセグメント・セレクタのRPLフィールドを読み取ると判定できることに注意が必要である。このセグメント・セレクタは、オペレーティング・システムへの呼び出しの一部としてスタックにストアされる。オペレーティング・システムは、セグメント・セレクタをスタックからレジスタにコピーして、ARPL命令のオペランドとして使用できる。

4.10.5. アライメント・チェック

CPLが3であるときは、レジスタCR0のAMフラグとEFLAGSレジスタのACフラグをセットして、メモリ参照のアライメントをチェックできる。アライメントの合っていないメモリ参照があると、アライメント例外（#AC）が発生する。プロセッサが特権レベル0、1、または2で動作しているときは、アライメント例外は発生しない。アライメント・チェックがイネーブルになっているときのアライメント要件については、表5-7.を参照。

4.11. ページレベルの保護

ページレベル保護は、単独で使用するか、またはセグメントに適用できる。ページレベル保護をフラットなメモリモデルに使用すると、スーパーバイザのコードとデータ（オペレーティング・システムまたはエグゼクティブ）をユーザのコードとデータ（アプリケーション・プログラム）から保護できる。また、コードを含むページを書き込み保護にできる。セグメント・レベル保護とページレベル保護を組み合わせると、ページレベルの読み取り / 書き込み保護によって、セグメント内の保護グラニュラリティが高まる。

ページレベル保護を使用すると、（セグメント・レベル保護の場合と同じように）各メモリ参照がチェックされて、保護チェックを満たしているか検証される。すべてのチェックはメモリサイクルが開始する前に実行され、違反があれば、サイクルは開始されず、ページフォルト例外が発生する。チェックはアドレス変換と並行して実行されるので、処理能力が損なわれることはない。

プロセッサは、次の2つのページレベル保護チェックを実行する。

- アドレス指定可能な所有者制限（スーパーバイザ・モードとユーザモード）
- ページタイプ（読み取り専用または読み取り/書き込み）

これらのチェックのいずれかに違反があると、ページフォルト例外が発生する。ページフォルト例外メカニズムの説明については、第5章の「割り込み14 ーページフォルト例外（#PF）」を参照。本章では、ページフォルト例外となる保護違反について説明する。

4.11.1. ページ保護フラグ

ページの保護情報は、ページ・ディレクトリ・エントリまたはページ・テーブル・エントリにある2つのフラグ（読み取り/書き込みフラグ（ビット1）とユーザ/スーパーバイザ・フラグ（ビット2））に入っている（図3-14.を参照）。保護チェックは、第一レベルと第二レベル両方のページテーブル（つまり、ページ・ディレクトリとページテーブル）に適用される。

4.11.2. アドレス指定可能ドメインの制限

ページレベル保護メカニズムによって、次の2つの特権レベルに基づいてページへのアクセスを制限することができる。

- スーパーバイザ・モード（U/Sフラグが0）ー（最も特権レベルが高い）オペレーティング・システムまたはエグゼクティブ、（デバイスドライバなど）その他のシステム・ソフトウェア、（ページテーブルなど）保護されたシステムデータに対してするモード
- ユーザモード（U/Sフラグが1）ー（最も特権レベルが低い）アプリケーション・コードとデータに対するモード

セグメント特権レベルは、次のようにページ特権レベルにマッピングされる。プロセッサがCPL 0、1、または2で現在動作している場合は、スーパーバイザ・モードになっている。CPL 3で動作している場合は、ユーザモードになっている。プロセッサは、スーパーバイザ・モードにあるときは、すべてのページにアクセスできる。ユーザモードにあるときは、ユーザレベルのページだけにアクセスする。（4.11.3.項「ペー

ジタイプ」で説明しているように、制御レジスタ CR0 の WP フラグはスーパーバイザ許可を変更することに注意が必要である。）

ページレベル保護メカニズムを使用するには、コード・セグメントとデータ・セグメントを少なくとも 2 つのセグメント・ベースの特権レベルに対してセットアップしなければならず、スーパーバイザのコード・セグメントとデータ・セグメントに対してはレベル 0、ユーザのコード・セグメントとデータ・セグメントに対してはレベル 3 であることに注意が必要である。（このモデルでは、スタックはデータ・セグメントに置かれる。）セグメントの使用を最小限に抑えるために、フラット・メモリ・モデルを使用できる（3.2.1. 項「基本フラットモデル」を参照）。

この場合には、ユーザとスーパーバイザのコード・セグメントとデータ・セグメントは、すべてリニアアドレス空間のアドレス 0 で始まり、相互にオーバレイする。この配置では、（スーパーバイザ・レベルで動作する）オペレーティング・システム・コードと（ユーザレベルで動作する）アプリケーション・コードは、セグメントがないかのように実行できる。オペレーティング・システムとアプリケーションのコードとデータ間の保護は、プロセッサのページレベル保護メカニズムによって提供される。

4.11.3. ページタイプ

ページレベル保護メカニズムでは、次の 2 つのページタイプを認識する。

- 読み取り専用アクセス（R/W フラグが 0）
- 読み取り / 書き込みアクセス（R/W フラグが 1）

プロセッサがスーパーバイザ・モードにあり、レジスタ CR0 の WP フラグがクリアされていると（リセット初期化後の状態）、すべてのページは、読み取り可能かつ書き込み可能である（書き込み保護は無視される）。プロセッサは、ユーザモードにあるときは、読み取り / 書き込みアクセス可能であるユーザ・モード・ページだけに書き込むことができる。読み取り / 書き込みまたは読み取り専用であるユーザ・モード・ページは、読み取り可能である。スーパーバイザ・モード・ページは、ユーザモードから読み取ることも書き込むこともできない。この保護規則に違反する試みが行われると、ページフォルト例外が発生する。

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは、ユーザ・モード・ページをスーパーバイザ・モード・アクセスに対して書き込み保護にできる。レジスタ CR0 の WP フラグを 1 にセットすると、ユーザモードの書き込み保護されたページに対するスーパーバイザ・モード・センシティビティがイネーブルになる。このスーパーバイザ書き込み保護機能は、タスク作成（フォークまたは生成ともいう）のために UNIX* など一部のオペレーティング・システムで使用されている「コピー・オン・ライト」法の実現に役立つ。新しいタスクを作成するときに、親タスクのアドレス空間全体をコピーできる。こうすると、子タスクは、親タスクのセグ

メントとページを完全に複写したセットとなる。代替のコピー・オン・ライト法では、子タスクのセグメントとページを親タスクが使用している同じセグメントとページにマッピングすれば、メモリ空間と時間が節約できる。ページの個別コピーは、タスクの 1 つがページに書き込むときだけに作成される。WP フラグを使用し、共有ページを読み取り専用としてマークすると、スーパーバイザは、ユーザ・レベル・ページへの書き込みの試みを検出でき、そのページをその時点でコピーできる。

4.11.4. ページテーブルの 2 つのレベルの保護の組み合わせ

任意の 1 ページに対して、そのページ・ディレクトリ・エントリ（第一レベルのページテーブル）の保護属性は、そのページ・テーブル・エントリ（第二レベルのページテーブル）の保護属性と異なってもよい。プロセッサは、ページに対する保護をページ・ディレクトリ・エントリとページ・テーブル・エントリの両方でチェックする。表 4-2. に、WP フラグがクリアされているときの保護属性のあり得る組み合わせによって与えられる保護を示す。

4.11.5. ページ保護のオーバーライド

次のタイプのメモリアクセスは、プロセッサが現在動作している CPL に関係なく、それらが特権レベル 0 アクセスの場合と同じようにチェックされる。

- GDT、LDT、または IDT にあるセグメント・ディスクリプタへのアクセス
- 特権レベルの変更が行われるときの、特権レベル間呼び出しまたは例外 / 割り込みハンドラへの呼び出しの間の特権レベル間スタックへのアクセス

4.12. ページ保護とセグメント保護の組み合わせ

ページングがイネーブルになっていると、プロセッサは、まずセグメント保護を評価し、次にページ保護を評価する。プロセッサがセグメント・レベルまたはページレベルで保護違反を検出すると、メモリアクセスは行われず、例外が生成される。例外がセグメンテーションによって生成されると、ページング例外は生成されない。

ページレベル保護をセグメント・レベル保護に優先して使用はできない。例えば、コード・セグメントは、定義によって書き込み不可能である。コード・セグメントがページングされる場合に、ページの R/W フラグを読み取り / 書き込みに設定しても、ページは書き込み可能にならない。ページに書き込もうとすると、セグメント・レベル保護チェックによって阻止される。

ページレベル保護を使用して、セグメント・レベル保護を強化することができる。例えば、大きい読み取り / 書き込みデータ・セグメントがページングされた場合、ページ保護メカニズムを使用して個々のページを書き込み保護にできる。

表 4-2. ページ・ディレクトリとページテーブルの保護の組み合わせ

ページ・ディレクトリ・エントリ		ページ・テーブル・エントリ		組み合わせた結果	
特権	アクセスタイプ	特権	アクセスタイプ	特権	アクセスタイプ
ユーザ	読み取り専用	ユーザ	読み取り専用	ユーザ	読み取り専用
ユーザ	読み取り専用	ユーザ	読み取り / 書き込み	ユーザ	読み取り専用
ユーザ	読み取り / 書き込み	ユーザ	読み取り専用	ユーザ	読み取り専用
ユーザ	読み取り / 書き込み	ユーザ	読み取り / 書き込み	ユーザ	読み取り / 書き込み
ユーザ	読み取り専用	スーパーバイザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
ユーザ	読み取り専用	スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り/書き込み*
ユーザ	読み取り / 書き込み	スーパーバイザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
ユーザ	読み取り / 書き込み	スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り / 書き込み
スーパーバイザ	読み取り専用	ユーザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り専用	ユーザ	読み取り / 書き込み	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り / 書き込み	ユーザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り / 書き込み	ユーザ	読み取り / 書き込み	スーパーバイザ	読み取り / 書き込み
スーパーバイザ	読み取り専用	スーパーバイザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り専用	スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り専用	スーパーバイザ	読み取り/書き込み*
スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り / 書き込み	スーパーバイザ	読み取り / 書き込み

注：

* CR0 の WP フラグがセットされている場合は、アクセスタイプはページ・ディレクトリ・エントリとページ・テーブル・エントリの R/W フラグによって決定される。

5

割り込みと例外の処理

第 5 章
割り込みと例外の処理

5

本章では、保護モードで動作中に使用される、プロセッサの割り込み / 例外の処理機構について説明する。本章で提供する情報の大部分は、実アドレスモードや仮想 8086 モードで使用される割り込み / 例外機構も当てはまる。実アドレスモードや仮想 8086 モードで使用する割り込み / 例外機構の相違点については、第 15 章「デバッグと性能モニタリング」を参照。

5.1. 割り込みと例外の概要

割り込みと**例外**は、システム、プロセッサ、または現在実行されているプログラムまたはタスクのどこかに、プロセッサの処置を必要とする状態が発生したことを示すイベントである。割り込みまたは例外が発生すると、通常は、現在実行されているプログラムまたはタスクから、**割り込みハンドラ**または**例外ハンドラ**と呼ばれる特殊なソフトウェア・ルーチンまたはタスクに、実行が強制的に移される。割り込みまたは例外に対してプロセッサがとる処置は、割り込みまたは例外の**サービス**または**処理**と呼ばれる。

割り込みは、通常は、ハードウェアからの信号に応えて、プログラムの実行中の任意の時間に発生する。システム・ハードウェアは、割り込みを使用して、周辺機器のサービスの要求など、プロセッサの外部のイベントを処理する。ソフトウェアも、INT *n* 命令を生成すれば、割り込みを生成できる。

例外は、プロセッサが命令を実行中にゼロ除算などのエラー状態を検出したときに発生する。プロセッサは、保護違反、ページフォルト、内部マシンフォルトなど、各種のエラー状態を検出する。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサの**マシンチェック・アーキテクチャ**により、内部ハードウェア・エラーおよびバスエラーが検出されたときにも、マシンチェック例外が生成される。

IA-32 アーキテクチャの割り込み処理と例外処理の機構は、アプリケーション・プログラムおよびオペレーティング・システムまたはエグゼクティブに対して透過的に、割り込みと例外を処理できる。割り込みを受信するか、例外が検出されると、現在実行中のプロシージャまたはタスクは自動的に保留状態になり、プロセッサが割り込みハンドラまたは例外ハンドラを実行している間はその状態が続く。ハンドラの実行が完了すると、プロセッサは、割り込みをかけられたプロシージャやタスクの実行を再開する。割り込みをかけられたプロシージャやタスクは、プログラムの連続性を保っ

た状態で再開される。ただし、例外からの回復が不可能な場合や、割り込みによって実行中のプログラムが終了した場合は、プログラムの連続性は失われる。

本章では、保護モードでのプロセッサの割り込み処理と例外処理の機構を説明する。各例外とその発生条件については、本章の最後に詳しく説明する。実アドレスモードと仮想 8086 モードでの割り込みと例外の処理については、第 16 章「8086 エミュレーション」を参照のこと。

5.2. 例外ベクタと割り込みベクタ

例外と割り込みを処理しやすいように、IA-32 アーキテクチャで定義されている、プロセッサの特殊な処理を必要とする各例外条件および割り込み条件には、ベクタと呼ばれる個別の識別番号が割り当てられている。プロセッサは、例外または割り込みに割り当てられた**ベクタ**を、プロセッサの割り込みディスクリプタ・テーブル（IDT）へのインデックスとして使用し、例外ハンドラまたは割り込みハンドラの入口点を見付ける（5.10. 節「割り込みディスクリプタ・テーブル（IDT: Interrupt Descriptor Table）」を参照）。

ベクタ番号の許容範囲は 0 ～ 255 である。0 ～ 31 の範囲のベクタは、アーキテクチャ上で定義される例外と割り込みのために、IA-32 アーキテクチャによって予約されている。この範囲内のベクタの中には、現時点では機能が定義されていないものもある。この範囲内の割り当てられていないベクタは、将来に備えて予約されている。**予約済みベクタを使用してはならない。**

32 ～ 255 の範囲のベクタは、ユーザ定義の割り込みとして指定されており、IA-32 アーキテクチャによって予約されていない。これらの割り込みは、通常は外部 I/O デバイスに割り当てられる。これにより外部 I/O デバイスは、5.3. 節「割り込みのソース」で説明する外部ハードウェア割り込み機構を使用して、プロセッサに割り込みを送信できる。

表 5-1. は、アーキテクチャ上で定義された例外および NMI 例外に対するベクタの割り当てを示している。この表は、各例外について、例外のタイプ（5.5. 節「例外の分類」を参照）とスタック上にエラーコードが保存されるかどうかを示す。また、それぞれの定義済み例外と NMI 例外のソースも示す。

5.3. 割り込みのソース

プロセッサは 2 つのソースから割り込みを受信する。

- 外部（ハードウェア生成）割り込み
- ソフトウェア生成割り込み

5.3.1. 外部割り込み

外部割り込みは、プロセッサ上のピンまたはローカルAPICを介して受信される。インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサ上の主な割り込みピンはLINT[1:0] ピンで、ローカルAPICに接続される（第8章「アドバンスド・プログラマブル 割り込みコントローラ（APIC)」を参照)。ローカルAPICが有効にされている場合は、プロセッサの例外ベクタまたは割り込みベクタに関連付けられるAPICのローカル・ベクタ・テーブルによって、LINT[1:0] ピンをプログラムできる。

ローカルAPICがディスエーブルされている場合は、これらのピンはそれぞれINTRおよびNMIピンとして構成される。INTRピンをアサートすると、外部割り込みが発生したという信号がプロセッサに送信され、プロセッサは8259Aなどの外部割り込みコントローラが指定した割り込みベクタ番号をシステムバスから読み取る（5.2. 節「例外ベクタと割り込みベクタ」を参照)。NMIピンをアサートすると、割り込みベクタ2に割り当てられるマスク不可能割り込み（NMI）の信号が送信される。

表 5-1. 保護モードにおける例外と割り込み

ベクタ番号	ニーモニック	説明	タイプ	エラーコード	ソース
0	#DE	除算エラー	フォルト	なし	DIV および IDIV 命令。
1	#DB	予約済み	フォルト/トラップ	なし	インテルのみが使用。
2	—	NMI 割り込み	割り込み	なし	マスク不可能外部割り込み。
3	#BP	ブレークポイント	トラップ	なし	INT 3 命令。
4	#OF	オーバーフロー	トラップ	なし	INTO 命令。
5	#BR	BOUND の範囲外	フォルト	なし	BOUND 命令。
6	#UD	無効オペコード（未定義オペコード)	フォルト	なし	UD2命令または予約オペコード。[1]
7	#NM	デバイス使用不可（マス・コプロセッサがない)	フォルト	なし	浮動小数点命令またはWAIT/FWAIT 命令。
8	#DF	ダブルフォルト	アボート	あり（0)	例外、NMI または INTR を生成できる任意の命令。
9		コプロセッサ・セグメント・オーバーラン（予約済み)	フォルト	なし	浮動小数点命令。[2]
10	#TS	無効 TSS	フォルト	あり	タスクスイッチまたはTSSアクセス。
11	#NP	セグメント不在	フォルト	あり	セグメント・レジスタのロードまたはシステム・セグメントへのアクセス。
12	#SS	スタック・セグメント・フォルト	フォルト	あり	スタック操作および SS レジスタのロード。
13	#GP	一般保護	フォルト	あり	任意のメモリ参照およびその他の保護チェック。

表 5-1. 保護モードにおける例外と割り込み（続き）

ベクタ番号	ニーモニック	説明	タイプ	エラーコード	ソース
14	#PF	ページフォルト	フォルト	あり	任意のメモリ参照。
15	—	（インテルがすでに予約済み。使用禁止）		なし	
16	#MF	x87 FPU 浮動小数点エラー（マスフォルト）	フォルト	なし	x87 FPU 浮動小数点命令または WAIT/FWAIT 命令。
17	#AC	アライメント・チェック	フォルト	あり（0）	メモリ内の任意のデータ参照。[3]
18	#MC	マシンチェック	アボート	なし	エラーコード（存在する場合）とソースはモデルに依存。[4]
19	#XF	SIMD 浮動小数点例外	フォルト	なし	SSE、SSE2、SSE3 浮動小数点命令。[5]
20-31	—	インテルがすでに予約済み。使用禁止			
32-255	—	ユーザー定義（非予約）割り込み	割り込み		外部割り込みまたは INT *n* 命令。

注 :

1. UD2 命令はインテル® Pentium® Pro プロセッサで導入された。
2. Intel386™ プロセッサ以降の IA-32 プロセッサはこの例外を生成しない。
3. この例外は Intel486™ プロセッサで導入された。
4. この例外はインテル Pentium プロセッサで導入され、P6 ファミリ・プロセッサで拡張された。
5. この例外はインテル® Pentium® III プロセッサで導入された。

通常、プロセッサのローカル APIC は、システムベースの I/O APIC に接続できる。この場合、I/O APIC のピンで受信された外部割り込みを、システムバス（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）または APIC シリアルバス（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサ）を介してローカル APIC にダイレクトできる。I/O APIC は割り込みのベクタ番号を特定し、この番号をローカル APIC に送信する。システムに複数のプロセッサが含まれる場合は、システムバス（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）または APIC シリアルバス（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサ）を介してプロセッサは互いに割り込みも送信できる。

LINT[1:0] ピンは、Intel486™ プロセッサおよびオンチップ・ローカル APIC を持たない初期のインテル Pentium プロセッサでは使用できない。これらのプロセッサには、この代わりに専用の NMI および INTR ピンがある。これらのプロセッサでは、外部割り込みは通常システムベースの割り込みコントローラ（8259A）で生成され、INTR ピンで受信される。

プロセッサには他にもプロセッサに割り込みを発生させるピンがあるが、これらの割り込みは本章で説明する割り込み / 例外機構では処理されない。このようなピンには、RESET#、FLUSH#、STPCLK#、SMI#、R/S#、INIT# ピンがある。特定の IA-32 プロ

セッサに、これらのうちのどのピンが含まれるかはプロセッサに依存する。これらのピンの機能については、各プロセッサのデータブックで説明されている。SMI# ピンについては、第13章「システム管理」でも説明されている。

5.3.2. マスク可能ハードウェア割り込み

INTR#ピンまたはローカルAPICを介してプロセッサに送信された外部割り込みを、**マスク可能ハードウェア割り込み**と呼ぶ。INTRピンを介して送信できるマスク可能ハードウェア割り込みには、IA-32アーキテクチャで定義されている0から255までのすべての割り込みベクタが含まれる。ローカルAPICを介して送信できる割り込みには、16から255までの割り込みベクタが含まれる。

EFLAGSレジスタ内のIFフラグによって、すべてのマスク可能ハードウェア割り込みをグループとしてマスクすることが許可される（5.8.1. 項「マスク可能ハードウェア割り込みのマスキング」を参照）。0から15までの割り込みがローカルAPICを介して送信された場合、APICは無効なベクタの受信を示す。

5.3.3. ソフトウェア生成割り込み

INT *n* 命令を使用すると、割り込みベクタ番号をオペランドとして指定することにより、ソフトウェア内部から割り込みを生成できる。例えば、INT 35命令を実行すると、割り込み35の割り込みハンドラが暗黙に呼び出される。

この命令では、0から255までの任意の割り込みベクタをパラメータとして使用できる。ただし、プロセッサで事前に定義されているNMIベクタを使用する場合、プロセッサの応答は通常の方法で生成されたNMI割り込みの場合とは異なる。この命令でベクタ番号2（NMIベクタ）を使用すると、NMI割り込みハンドラは呼び出されるが、プロセッサのNMI処理ハードウェアはアクティブ化されない。

注記

INT *n* 命令を使用してソフトウェアで生成された割り込みは、EFLAGSレジスタのIFフラグではマスクできない。

5.4. 例外のソース

プロセッサは3つのソースから例外を受信する。

- プロセッサが検出したプログラム・エラー例外
- ソフトウェア生成例外
- マシンチェック例外

5.4.1. プログラム・エラー例外

プロセッサは、アプリケーション・プログラム、オペレーティング・システム、またはエグゼクティブの実行中にプログラム・エラーを検出すると、例外を生成する。IA-32 アーキテクチャでは、プロセッサで検出可能な各例外についてベクタ番号を定義している。例外は、**フォルト**、**トラップ**、**アボート**に分類される（5.5. 節「例外の分類」を参照）。

5.4.2. ソフトウェア生成例外

INTO、INT 3、BOUND 命令を使用すると、ソフトウェアで例外を生成できる。これらの命令によって、命令ストリームの特定の時点で、特定の例外条件が実行されるかどうかをチェックできる。例えば、INT 3 命令を実行すると、ブレークポイント例外が生成される。

INT *n* 命令を使用して、ソフトウェアで特定の例外をエミュレートできるが、制限事項が1つある。INT *n* 命令の *n* オペランドに、IA-32 アーキテクチャのいずれかの例外のベクタが含まれている場合、プロセッサはそのベクタへの割り込みを生成した後、そのベクタに対応付けられた例外ハンドラを呼び出す。しかし、これは実際には割り込みであるため、そのベクタのハードウェア生成例外が通常エラーコードを生成する場合でも、プロセッサはエラーコードをスタックにプッシュしない。エラーコードを生成する例外については、例外ハンドラは例外の処理中にスタックからエラーコードをポップしようとする。INT *n* 命令を使用して例外の生成をエミュレートした場合、ハンドラは（存在しないエラーコードの代わりに）EIP をポップして破棄し、誤った場所へのリターンを送信する。

5.4.3. マシンチェック例外

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサは、チップ内部のハードウェアおよびバス・トランザクションの動作をチェックするための内部および外部マシンチェック機構を提供する。この機構は（プロセッサに依存する）拡張例外機構

を構成する。マシン・チェック・エラーを検出すると、プロセッサはマシンチェック例外（ベクタ 18）の信号を出し、エラーコードを返す。

マシンチェック機構の詳細については、本章の「割り込み 18 ーマシンチェック例外（#MC）」および第 14 章「マシン・チェック・アーキテクチャ」を参照。

5.5. 例外の分類

例外は、それが報告される形式、および例外の原因となった命令をプログラムやタスクの連続性を損なうことなく再スタートできるかどうかに応じて、**フォルト**、**トラップ**、**アボート**に分類される。

フォルト　フォルトとは、通常では修正可能であり、かつ修正された後にプログラムをその連続性を損なわずに再スタートできるタイプの例外をいう。フォルトが報告されると、プロセッサは、マシンのステートを、フォルトを起こした命令の実行開始点の手前のステートにリストアする。フォルトハンドラの戻りアドレス（レジスタ CS と EIP にセーブされている内容）は、フォルトを起こした命令の次の命令でなく、フォルトを起こした命令そのものを指す。

トラップ　トラップとは、それを起こした命令の実行直後に報告される例外をいう。トラップの場合、プログラム連続性を損なわずにプログラムやタスクの実行を続行できる。トラップハンドラの戻りアドレスは、トラップを起こした命令の後に実行される命令を指す。

アボート　アボートとは、例外の原因となった命令の正確な位置が必ずしも報告されない、またそれを起こしたプログラムやタスクを再スタートできない例外をいう。アボートは、ハードウェア・エラーや、システムテーブルに一致しないまたは不当な値が含まれているといった、重大なエラーを報告するために使用される。

注記

通常フォルトとして報告される例外の一部は、リスタート不能になり、プロセッサ・ステートの一部が失われる。この例として、POPAD 命令の実行時に、スタックフレームがスタック・セグメントの終端を超えた場合、このようなフォルトが報告される。ここで、例外ハンドラは、あたかも POPAD 命令が実行されなかったかのように、命令ポインタ CS:EIP がリストアされたことを認識するが、内部プロセッサ・ステート（特に、汎用レジスタ）は修正されている。これらの特殊な場合は、プログラミング・エラーと見なされる。このクラスの例外を発生させたアプリケーションは、通常はオペレーティング・システムによって強制終了される。

5.6. プログラムまたはタスクの再スタート

例外や割り込みの処理のすぐ後にプログラムやタスクを再スタートできるように、アボート以外のすべての例外は、正確に命令境界で報告されるように保証され、すべての割り込みは命令境界で受け取られるように保証される。

フォルトクラスの例外の場合、プロセッサがそれを生成したときにセーブするリターン命令ポインタは、フォルトを起こした命令を指す。それゆえ、フォルト処理後にプログラムやタスクが再スタートされると、フォルトを起こした命令も再スタート（再実行）される。この、フォルトを起こした命令を再スタートする方法は、オペランドへのアクセスがブロックされたときに生成される例外の処理に使用されることが多い。最もよく起こるフォルト例は、プログラムやタスクがメモリ内に存在しないページのオペランドを参照したときに発生する、ページフォルト例外（#PF）である。ページフォルト例外が発生すると、例外ハンドラは、フォルトを起こした命令を再スタートすると、ページをメモリ内へロードし、プログラムやタスクの実行を再開できる。この命令再スタートが現在実行中のプログラムやタスクに透過な形で処理されるようにするために、プロセッサは、必要なレジスタとスタックポインタをセーブし、フォルトを起こした命令を実行する手前のステートに自己をリストアできるようにする。

トラップクラスの例外の場合、リターン命令ポインタは、トラップを起こした命令の次の命令を指す。実行を移行するタイプの命令の途中でトラップが検出された場合は、リターン命令ポインタはその移行内容を反映する。例えばJMP命令の実行中にトラップが検出されたとすると、リターン命令ポインタは、JMP命令の次のアドレスではなくJMP命令のデスティネーションを指す。トラップ例外では必ず、プログラムやタスクを連続性を損なわずに再スタートできる。例えば、オーバーフロー例外はトラップを起こす例外である。この場合リターン命令ポインタは、EFLAGSレジスタのOF（オーバーフロー）フラグをテストしたINTO命令の次の命令を指す。この場合のトラップハンドラは、オーバーフロー条件を解消する。トラップハンドラから復帰すると、INTO命令の次の命令からプログラムまたはタスクの実行が続行される。

アボートクラスの例外は、プログラムやタスクの再スタートを確実にはサポートしない。一般にアボートハンドラは、アボート例外が起こった時点のプロセッサのステートに関する診断情報を収集し、アプリケーションやシステムを可能な限り既定の手順にしたがってシャットダウンするように設計されている。

割り込みは、連続性を損なわずに割り込まれたプログラムやタスクを再スタートすることを厳格にサポートする。ある割り込みについてセーブされるリターン命令ポインタは、次の命令、つまりプロセッサが割り込みを受け取った命令境界から実行される命令を指す。実行中であった命令がリピート・プリフィックス付きである場合、割り込みは、現在の繰り返しが終了した時点で、レジスタが次の繰り返しを実行するように設定された状態で受け取られることになる。

P6 ファミリのプロセッサが推論的に命令を実行できることは、プロセッサの割り込み受け取りには影響しない。割り込みは、命令実行のリタイアメント・フェーズに配置されている命令境界で受け取られる。それゆえ割り込みは常に「インオーダー」命令ストリームの中で受け取られる。P6 ファミリ・プロセッサのマイクロアーキテクチャとアウトオブオーダー命令の実行の詳細については、『IA-32 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』の第 2 章「IA-32 インテル® アーキテクチャの概説」を参照。

インテル® Pentium® プロセッサおよび以前の IA-32 プロセッサでは、さまざまな命令のプリフェッチおよびプリデコードも実行されるが、この場合も例外や割り込みの信号は、命令が実際に「インオーダー」実行されるまで送信されない。あるコードサンプルについて、例外の信号が送信されるのは、IA-32 のどのファミリのプロセッサでもコードが実行されたときである（ただし、新しい例外または新しいオペコードが定義された場合は除く）。

5.7. マスク不可能割り込み（NMI: Nonmaskable Interrupt）

マスク不可能割り込み（NMI）は、次の 2 つの方法のいずれかで生成される。

- 外部ハードウェアが NMI ピンをアサートする。
- プロセッサがシステムバス（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）または APIC シリアルバス（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサ）上で送信モードが NMI のメッセージを受信する。

プロセッサは、これらのソースのいずれかから NMI を受信すると、割り込みベクタ番号 2 によって示された NMI ハンドラを呼び出して直ちに NMI を処理する。また、プロセッサは特定のハードウェア条件を呼び出して、NMI ハンドラが実行を完了するまで、NMI 割り込みを含む他の割り込みが受信されないようにする（5.7.1. 項「複数の NMI の処理」を参照）。

また、上記のいずれかのソースから NMI を受信した場合は、EFLAGS レジスタの IF フラグで NMI をマスクできない。

（INTR ピンを使用して）ベクタ 2 へのマスク可能ハードウェア割り込みを生成して NMI 割り込みハンドラを呼び出せるが、この割り込みは実際には NMI 割り込みではない。プロセッサの NMI 処理ハードウェアをアクティブ化する実際の NMI 割り込みは、上記のいずれかの機構によってのみ送信できる。

5.7.1. 複数の NMI の処理

NMI 割り込みハンドラの実行中、プロセッサは次の IRET 命令が実行されるまで NMI ハンドラへの再度の呼び出しをディスエーブルにする。こうして以降の NMI をブロックすれば、NMI ハンドラへの呼び出しが積重なることを防ぐ。マスク可能ハードウェア割り込みをディスエーブルにする場合は、割り込みゲートから NMI 割り込みハンドラにアクセスすることを推奨する（5.8.1. 項「マスク可能ハードウェア割り込みのマスキング」を参照）。NMI ハンドラが IOPL が 3 未満の仮想 8086 タスクである場合、このハンドラから発行される IRET 命令は一般保護例外を生成する（16.2.7. 項「センシティブな命令」を参照）。この場合、NMI は、一般保護例外ハンドラが起動される前にマスクはされない。

5.8. 割り込みのイネーブル / ディスエーブル

プロセッサのステートおよびEFLAGS レジスタのIF と RF フラグのステートに応じて、プロセッサはある種の割り込みの生成を禁止する。これについて次の項で説明する。

5.8.1. マスク可能ハードウェア割り込みのマスキング

IF フラグは、プロセッサの INTR ピンまたはローカル APIC から受け取ったマスク可能ハードウェア割り込みの処理をディスエーブルにできる（5.3.2. 項「マスク可能ハードウェア割り込み」を参照）。IF フラグがクリアされている場合、プロセッサは INTR ピンまたはローカル APIC から受け取った割り込みが内部割り込み要求を生成するのを禁止する。IF フラグがセットされている場合、INTR ピンまたはローカル APIC から受け取った割り込みは通常の外部割り込みとして処理される。

IF フラグは、NMI ピンに送信されたマスク不可能割り込み（NMI）や、ローカル APIC から送信された送信モードが NMI のメッセージには影響せず、プロセッサが生成する例外にも影響しない。EFLAGS レジスタの他のフラグの場合と同様に、プロセッサはハードウェア・リセットに応答して IF フラグをクリアする。

マスク可能ハードウェア割り込みのグループに、予約されている割り込みと例外のベクタ 0 から 32 が含まれていることが混乱の原因となる場合がある。アーキテクチャでは、IF フラグがセットされている場合、0 から 32 の任意のベクタの割り込みを INTR ピンを介してプロセッサに送信したり、16 から 32 の任意のベクタの割り込みをローカル APIC を介して送信できる。プロセッサは割り込みを生成し、そのベクタ番号によって示される割り込みまたは例外ハンドラを呼び出す。したがって、例えば、INTR ピンを使用して（ベクタ 14 によって）ページ・フォルト・ハンドラを呼び出せるが、これは実際にはページフォルト例外ではなく、割り込みである。INT *n* 命令の場合と同様に（5.4.2. 項「ソフトウェア生成例外」を参照）、INTR ピンを介して例外ベクタに

割り込みが生成された場合、プロセッサはエラーコードをスタックにプッシュしないので、例外ハンドラは正常に機能しない。

IF フラグは、STI（割り込みイネーブル・フラグ・セット）命令、CLI（割り込みイネーブル・フラグ・クリア）命令によってそれぞれセット、クリアされる。これらの命令は、CPL が IOPL に等しいか下位である場合にだけ実行できる。CPL が IOPL より上位であるときにこれらの命令を実行すると、一般保護例外（#GP）が発生する。（制御レジスタ CR4 の VME フラグをセットすることにより仮想モード拡張がイネーブルされている場合は、IOPL のこれらの命令に対する影響は少し変わる。16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照。動作も PVI フラグの影響を受ける。16.4. 節「保護モード仮想割り込み」を参照。）

IF フラグは以下の操作によっても影響を受ける。

- PUSHF 命令はすべてのフラグをスタック上にストアする。ここではフラグを検査したり変更できる。POPF 命令を使用すると、変更されたフラグを元の EFLAGS レジスタへロードできる。
- タスクスイッチおよび POPF と IRET 命令は、EFLAGS レジスタをロードする。したがって、これらを使用して IF フラグの設定値を変更できる。
- 割り込みが割り込みゲートを介して処理される場合、IF フラグは自動的にクリアされ、したがってマスク可能ハードウェア割り込みはディスエーブルされる。（割り込みがトラップゲートを介して処理される場合は、IF フラグはクリアされない。）

これらの命令が IF フラグに対して行うことのできる操作の詳細については、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」の、CLI、STI、PUSHF、POPF、IRET 命令の項を参照。

5.8.2. 命令ブレークポイントのマスキング

EFLAGS レジスタの RF（再開）フラグは、命令ブレークポイント条件に対するプロセッサの応答を制御する（2.3. 節「EFLAGS レジスタのシステムフラグとフィールド」の RF フラグの説明を参照）。

このフラグがセットされている場合、命令ブレークポイントがデバッグ例外（DB#）を生成するのを防止する。このフラグがクリアされている場合、命令ブレークポイントはデバッグ例外を生成する。RF フラグの主要な機能は、プロセッサが命令ブレークポイントでデバッグ例外のループに入るのを防止することである。このフラグの使用についての詳細は、15.3.1.1. 項「命令ブレークポイント例外条件」を参照。

5.8.3. スタック切り替え時に行われる例外と割り込みのマスキング

ソフトウェアは別のスタック・セグメントへ切り替えるために、多くの場合次の例のように1対の命令を使用する。

```
MOV SS, AX
MOV ESP, StackTop
```

セグメント・セレクタがSSレジスタにロードされた後、ESPレジスタにロードされるまでの間に割り込みや例外が発生した場合、割り込み/例外ハンドラの実行中、スタック空間に入れられた論理アドレスのこの2つの部分は一致しなくなる。

こうした状況を回避するために、プロセッサはSSに対するMOV命令もしくはSSに対するPOP命令のどちらかを実行する場合、次の命令の後に来る命令境界に到達するまでの間、割り込み、デバッグ例外、シングル・ステップ・トラップ例外を禁止する。この場合でもその他のフォルトは発生することがある。LSS命令を使用してSSレジスタの内容を変更（このレジスタを変更する場合に推奨される方法）するようにすれば、この問題は起こらない。

5.9. 同時に発生した例外や割り込みの間の優先順位

1つの命令境界で複数の例外または割り込みが保留になっている場合、プロセッサはこれらを予測可能な順序で処理する。表5-2.に、例外や割り込みソースをクラスに分けてその間の優先順位を示す。

表 5-2. 例外や割り込みが同時に発生した場合の優先順位

優先順位	説明
1（最も高い）	ハードウェア・リセットおよびマシンチェック - RESET - マシンチェック
2	タスクスイッチ時のトラップ - TSS レジスタの T フラグがセットされる
3	外部ハードウェア割り込み - FLUSH - STOPCLK - SMI - INIT
4	前の命令時のトラップ - ブレークポイント - デバッグトラップ例外（TF フラグがセットされる、またはデータ /I/O ブレークポイント）
5	外部割り込み - NMI 割り込み - マスク可能ハードウェア割り込み
6	コード・ブレークポイント・フォルト

表 5-2. 例外や割り込みが同時に発生した場合の優先順位（続き）

優先順位	説明
7	次の命令フェッチからのフォルト - コード・セグメント・リミット違反 - コード・ページ・フォルト
8	次の命令デコーディングからのフォルト - 命令長さ >15 バイト - 不正なオペコード - コプロセッサが使用できない
9（最も低い）	命令実行時のフォルト - オーバーフロー - 境界エラー - TSS が無効 - セグメント不在 - スタックフォルト - 一般保護 - データ・ページ・フォルト - アライメント・チェック - x87 FPU 浮動小数点例外 - SIMD 浮動小数点例外

表 5-2. に一覧表示したこれらのクラス間の優先順位はアーキテクチャで一定しているが、各クラス中の例外の優先順位はプロセッサに依存し、プロセッサによって異なることがある。プロセッサは優先順位の最も高いクラスから保留の例外または割り込みを選んでこれを第一に処理し、ハンドラの最初の命令へ実行を移行する。これより優先順位の低い例外は廃棄され、これより優先順位の低い割り込みは保留状態で残される。割り込みハンドラが、プログラムまたはタスク内の例外や割り込みが発生したポイントへ実行を戻すと、廃棄された例外が再度発生する。

5.10. 割り込みディスクリプタ・テーブル（IDT: Interrupt Descriptor Table）

割り込みディスクリプタ・テーブル（IDT）は、各例外 / 割り込みのベクタを、その例外 / 割り込みを処理するために使用されるプロシージャやタスクのゲート・ディスクリプタに対応付ける。IDT は、GDT や LDT と同じように 8 バイトのゲート・ディスクリプタの配列になっている（保護モードで）。GDT と異なる点として、IDT の最初のエントリにはディスクリプタを入れてもよい。IDT へのインデックスを形成するために、プロセッサは例外 / 割り込みベクタを 8（ゲート・ディスクリプタに入っているバイト数）でスケーリングする。例外 / 割り込みベクタは 256 個しかないので、IDT にはディスクリプタを 256 より多く入れる必要はないが、256 未満にすることはできる。これは、ディスクリプタは発生する可能性のある割り込みおよび例外のベクタについてだけ必要になるからである。IDT の空のディスクリプタ・スロットには、0 にセットされたディスクリプタのセグメント存在フラグがなければならない。

キャッシュ・ライン・フィルの処理能力を最大限にするためには、IDT のベースアドレスのアライメントを 8 バイト境界に合わせなければならない。リミット値はバイト単位で表され、これをベースアドレスに加算して最後の有効バイトのアドレスを得る。リミット値を 0 にすると有効バイトがちょうど 1 になる。IDT エントリは常に 8 バイトであるから、リミットは常に 8 の整数倍より 1 だけ少なく（つまり 8N-1）する必要がある。

IDT はリニアアドレス空間のどこに常駐してもかまわない。図 5-1. に示すように、プロセッサは IDTR レジスタを使用して IDT の位置を確認する。このレジスタには IDT 用に 32 ビットのベースアドレスと 16 ビット・リミットの両方が入っている。

LIDT（IDT レジスタのロード）命令と SIDT（IDT レジスタのストア）命令は、レジスタ IDTR の内容をそれぞれロード、ストアする。LIDT 命令は、メモリ・オペランドに入っているベースアドレスとリミットをレジスタ IDTR にロードする。この命令は CPL が 0 のときにだけ実行できるもので、通常は IDT 作成時にオペレーティング・システムの初期化コードによって使用される。また IDT を別の IDT に変更するために、オペレーティング・システムによって使用されることもある。SIDT 命令は、IDTR にストアされているベース値とリミット値をメモリにコピーする。この命令はどの特権レベルでも使用できる。

ベクタがIDP境界外のディスクリプタを参照すると一般保護例外(#GP)が生成される。

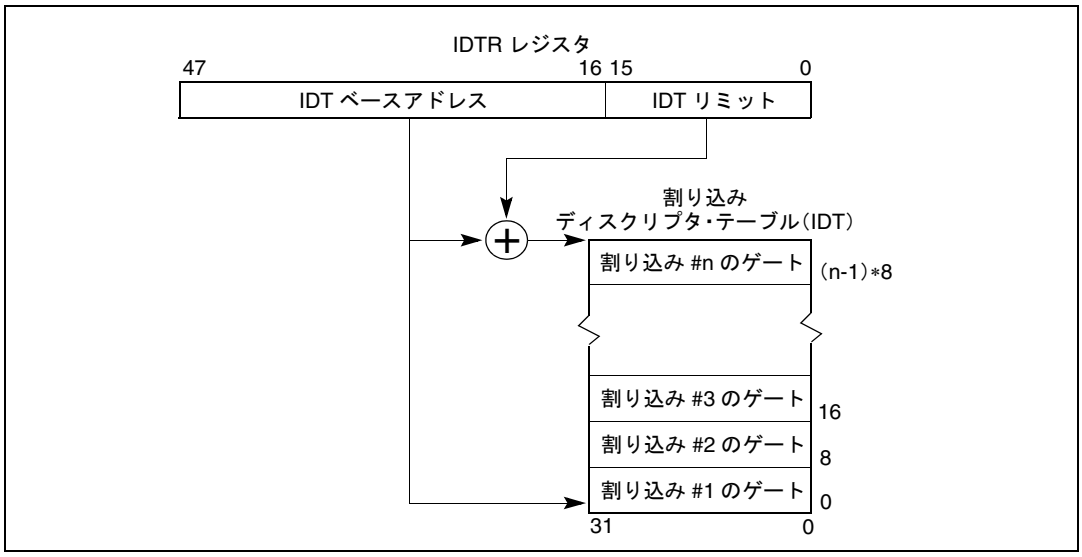


図 5-1. IDTR と IDT の関係

5.11. IDT ディスクリプタ

IDT には次の 3 種類のゲート・ディスクリプタのいずれかが入る。

- タスク・ゲート・ディスクリプタ
- 割り込みゲート・ディスクリプタ
- トラップ・ゲート・ディスクリプタ

図 5-2. に、タスクゲート、割り込みゲート、トラップゲートの各ディスクリプタのフォーマットを示す。IDT 内のタスクゲートは GDT や LDT に入っているタスクゲートとフォーマットが同じである（6.2.4. 項「タスク・ゲート・ディスクリプタ」を参照）。タスクゲートには、例外 / 割り込みハンドラのタスクのために、TSS のセグメント・セレクタが含まれる。

割り込みゲートおよびトラップゲートはコールゲートによく似ている（4.8.3. 項「コールゲート」を参照）。これらのゲートには、プロセッサが例外 / 割り込みハンドラのコード・セグメント内のハンドラ・プロシージャにプログラムの実行を移行するために使用する far ポインタ（セグメント・セレクタおよびオフセット）が含まれている。これらのゲートは、プロセッサが EFLAGS レジスタ内の IF フラグを使用する方法が異なる（5.12.1.2. 項「例外 / 割り込みハンドラ・プロシージャによるフラグの使用法」を参照）。

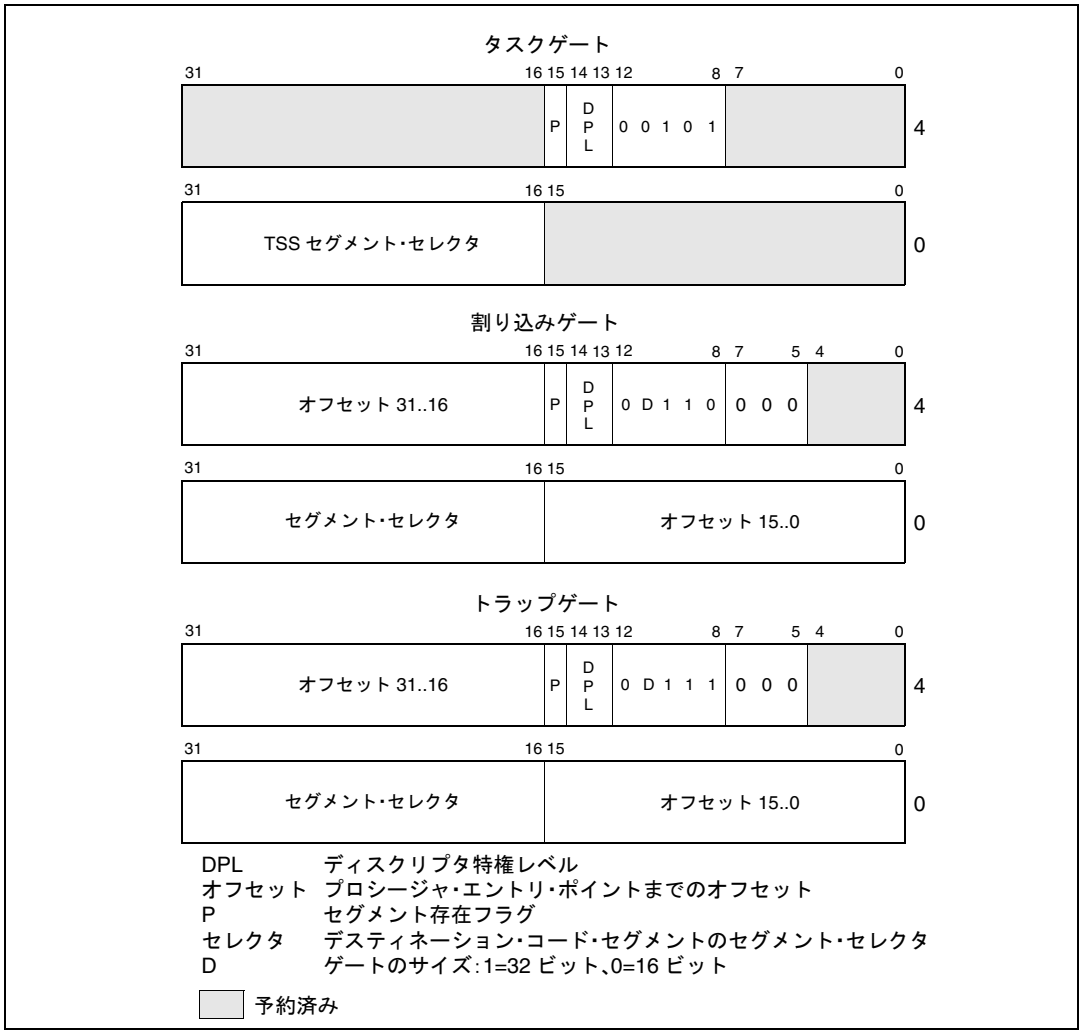


図 5-2. IDT のゲート・ディスクリプタ

5.12. 例外処理と割り込み処理

プロセッサは、例外/割り込みハンドラへの呼び出しの処理を、CALL 命令によってプロシージャやタスクへの呼び出しを処理する場合と同じ方法で行う。例外 / 割り込みに応答するとき、プロセッサは例外 / 割り込みベクタを IDT 内のディスクリプタへのインデックスとして使用する。インデックスが割り込みゲートまたはトラップゲートを指している場合、プロセッサはCALL 命令によるコールゲート呼び出しと同じような方法で例外/割り込みハンドラを呼び出す（4.8.2. 項「ゲート・ディスクリプタ」から 4.8.6. 項「呼び出し先プロシージャからの戻り」までを参照）。インデックスがタスクゲートを指している場合は、CALL 命令によるタスクゲート呼び出しと同じような

方法で、例外/割り込みハンドラのタスクへのタスクスイッチを実行する（6.3.節「タスク・スイッチング」を参照)。

5.12.1. 例外 / 割り込みハンドラ・プロシージャ

割り込みゲートやトラップゲートは、現在実行中のタスクのコンテキスト内で動作する例外/割り込みハンドラ・プロシージャを参照する（図5-3.を参照)。ゲートのセグメント・セレクタは、GDTまたは現行LDTのどちらかに入っている、実行可能なコード・セグメントのセグメント・ディスクリプタを指す。ゲート・ディスクリプタのオフセット・フィールドは、この例外/割り込みハンドラ・プロシージャの始点を指す。

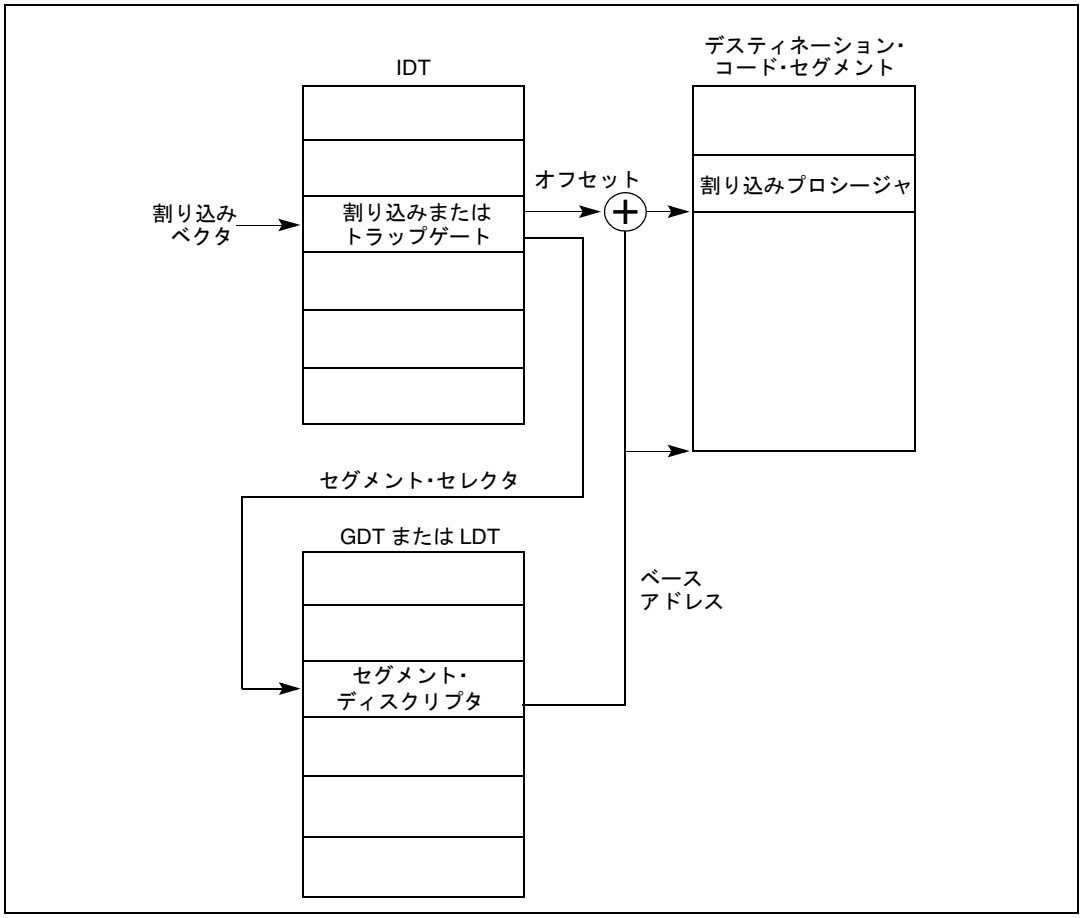


図 5-3. 割り込みプロシージャ呼び出し

プロセッサが例外/割り込みハンドラ・プロシージャへの呼び出しを行うとき、

- ハンドラ・プロシージャが、割り込まれたプロシージャより数値として小さい特権レベル（すなわち、高い特権レベル）で実行される場合は、スタックスイッチが発生する。スタックスイッチが発生すると、以下の操作が行われる。
  a. ハンドラが使用するスタックのセグメント・セレクタとスタックポインタが、現在実行中のタスクの TSS から取得される。この新しいスタック上に、プロセッサは割り込まれたプロシージャのスタック・セグメント・セレクタとスタックポインタをプッシュする。
  b. 次に、プロセッサは、EFLAGS、CS、EIP の各レジスタの現在の状態を新しいスタック上にセーブする（図 5-4. を参照）。
  c. 例外が発生したためにエラーコードがセーブされる場合は、エラーコードは新しいスタック上の EIP 値の後にプッシュされる。
- ハンドラ・プロシージャが、割り込まれたプロシージャと同じ特権レベルで実行される場合は、以下の操作が行われる。
  a. プロセッサは、EFLAGS、CS、EIP の各レジスタの現在の状態を現在のスタック上にセーブする（図 5-4. を参照）。
  b. 例外が発生したためにエラーコードがセーブされる場合は、エラーコードは現在のスタック上の EIP 値の後にプッシュされる。

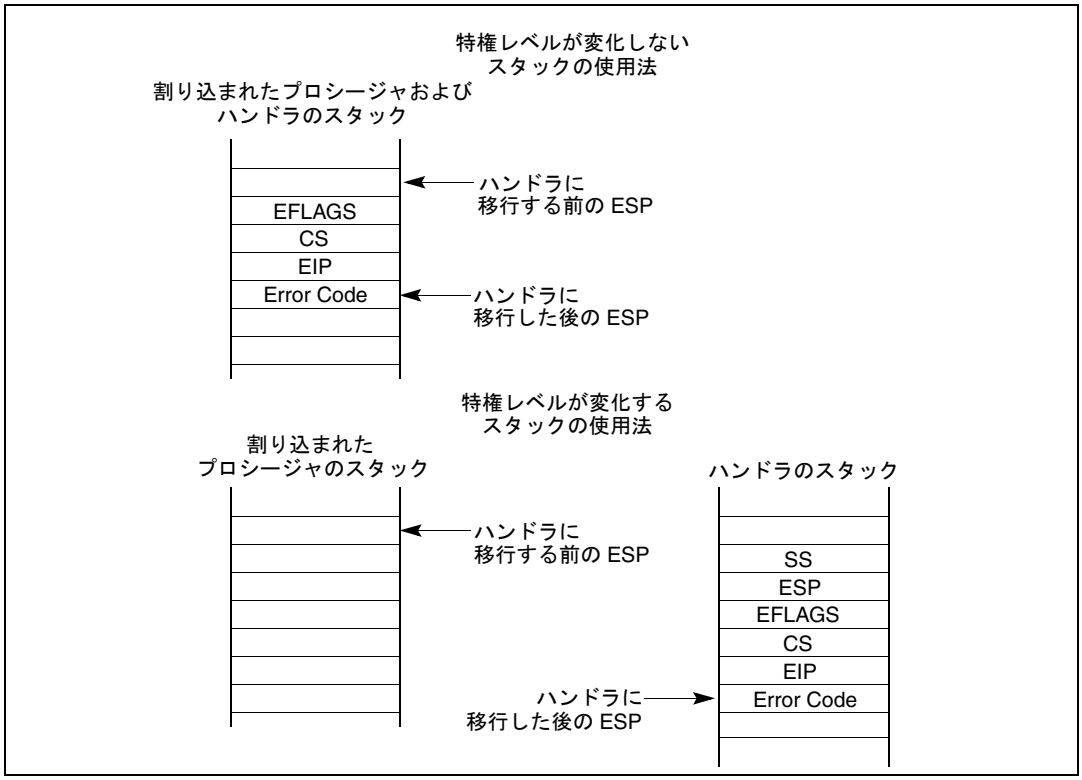


図 5-4. 割り込み / 例外処理ルーチンへの移行時のスタック使用法

例外/割り込みハンドラ・プロシージャから復帰するためには、そのハンドラがIRET（またはIRETD）命令を使用しなければならない。IRET命令はRET命令と類似しているが、セーブされているフラグをEFLAGSレジスタにリストアする点が異なる。EFLAGSレジスタのIOPLフィールドはCPLが0の場合にだけリストアされる。またIFフラグは、CPLがIOPLに等しいか下位である場合にだけ変更される。IRET命令が行う操作の全体については、『IA-32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、中巻A』の第3章の「IRET/IRETD—Interrupt Return」を参照のこと。

ハンドラ・プロシージャ呼び出し中にタスクスイッチが起こった場合、復帰する時点でIRET命令は割り込まれたプロシージャのスタックへスイッチバックする。

5.12.1.1. 例外 / 割り込みハンドラ・プロシージャの保護

例外 / 割り込みハンドラ・プロシージャの特権レベル保護は、コールゲートを介した通常のプロシージャ呼び出しの場合と類似している（4.8.4. 項「コールゲートを通じたコード・セグメントへのアクセス」を参照）。プロセッサは、CPLよりも特権の低い（数値の大きい特権レベル）コード・セグメントにある例外 / 割り込みハンドラ・プロシージャに実行が移行することを許さない。

この規則に違反した試みが行われると一般保護例外（#GP）が発生する。例外/割り込みハンドラ・プロシージャの保護機構は次の点が異なる。

- 割り込みと例外のベクタにはRPLがないので、例外 / 割り込みハンドラへの暗黙呼び出しではRPLチェックが行われない。
- プロセッサは、例外/割り込みがINT *n*、INT 3、またはINTO命令により生成された場合にのみ割り込み/トラップゲートのDPLをチェックする。この場合、CPLはゲートのDPLに等しいか下位でなければならない。この制限によって、ページ・フォルト・ハンドラなどのクリティカルな例外ハンドラが上位の特権コード・セグメントに配置されている（特権レベルの数値が小さい）場合に、特権レベル3で実行されているアプリケーション・プログラムまたはプロシージャが、ソフトウェア割り込みを使用してこのような例外ハンドラにアクセスするのを防止する。ハードウェア生成割り込みおよびプロセッサが検出する例外の場合は、プロセッサは割り込み/トラップゲートのDPLを無視する。

割り込みと例外は一般に予測できる時点で起こるものではないので、これらの特権レベル規則により、例外 / 割り込みハンドラ・プロシージャを実行できる特権レベルが効果的に限定される。次の手法のどちらかを使用すると、特権レベル違反を防止できる。

- 例外/割り込みハンドラをコンフォーミング・コード・セグメントへ入れることができる。この手法は、ハンドラがスタック上のデータにアクセスするだけでよい

場合（例えば除算エラー例外の場合）に利用できる。ハンドラがデータ・セグメントからデータを抽出する必要がある場合は、そのデータ・セグメントは特権レベル3からアクセス可能である必要があり、それではこのセグメントは保護されない。

- ハンドラを特権レベル 0 の非準拠コード・セグメントへ入れることができる。こうすればこのハンドラは、割り込まれたプログラムやタスクが動作していた CPL に関係なく、必ず実行される。

5.12.1.2. 例外 / 割り込みハンドラ・プロシージャによるフラグの使用法

プロセッサが割り込みゲートかトラップゲートのどちらかを通じて割り込み / 例外ハンドラへアクセスするとき、プロセッサはEFLAGS レジスタの内容をスタックにセーブした後で、このレジスタの TF フラグをクリアする。（プロセッサは割り込み / 例外ハンドラへの呼び出し時に、EFLAGS レジスタの VM、RF、NT フラグもクリアする。）TF フラグをクリアすると、命令トレースが割り込み応答に影響を与えることがなくなる。その後に実行される IRET 命令によって、TF フラグ（および VM、RF、NT フラグ）はEFLAGS レジスタの内容の一部としてスタックにセーブされていた値にリストアされる。

割り込みゲートとトラップゲートの唯一の相違点は、プロセッサがEFLAGS レジスタの IF フラグを処理する方法にある。割り込みゲートを通じて割り込み / 例外ハンドラ・プロシージャへアクセスするとき、プロセッサはIF フラグをクリアし、他の割り込みが現行の割り込みハンドラに干渉することを防ぐ。その後に実行される IRET 命令により、IF フラグはEFLAGS レジスタ内容の一部としてスタックにセーブされていた値にリストアされる。トラップゲートを通じてハンドラ・プロシージャへアクセスする場合は、IF フラグには影響が及ばない。

5.12.2. 割り込みタスク

例外 / 割り込みハンドラへのアクセスを IDT 内のタスクゲートを通じて行うと、タスクスイッチが起こる。例外や割り込みを独立のタスクで処理する方法には、次のような2つの利点がある。

- 割り込まれたプログラム / タスクのコンテキスト全体が自動的にセーブされる。
- 新しいTSS はハンドラが例外 / 割り込みを処理するときに新しい特権レベル0のスタックを使用することを許可する。現在の特権レベル0のスタックが破壊されたときに例外 / 割り込みが発生した場合は、タスクゲートを通じてハンドラにアクセスすることにより、ハンドラに新しい特権レベル0のスタックが与えられるので、システムがクラッシュするのを防止できる。
- 当該ハンドラに独立のアドレス空間を与えると、それを他のタスクから隔離できる。これは独立のLDT を与えることにより行う。

独立のタスクで割り込みを処理することの欠点は、タスクスイッチ時にセーブしなければならないマシンステートの量が増加するため、割り込みゲートを使用する場合よりも処理が遅くなり、割り込みの待ち時間が長くなることである。

IDT 内のタスクゲートは、GDT 内の TSS ディスクリプタを参照する（図 5-5. を参照）。ハンドラタスクへの切り替えは通常のタスクスイッチと同じ方法で処理される（6.3. 節「タスク・スイッチング」を参照）。割り込まれたタスクへのリンクは、ハンドラタスクの TSS の以前のタスク・リンク・フィールドにストアされる。例外によりエラーコードが生成された場合、このエラーコードは新しいタスクのスタックにコピーされる。

オペレーティング・システム内部で例外/割り込みハンドラタスクを使用する場合、タスクのディスパッチに使用できる機構が実際に 2 つ存在する。それはソフトウェア・スケジューラ（オペレーティング・システムの一部）とハードウェア・スケジューラ（プロセッサの割り込み機構の一部）である。ソフトウェア・スケジューラは、割り込みがイネーブルなときにディスパッチされる可能性のある割り込みタスクに対応できるものでなければならない。

注記

IA-32 アーキテクチャのタスクは再入可能ではないため、割り込みハンドラタスクは、割り込み処理を完了してから IRET 命令を実行するまでの間、割り込みをディスエーブルにする必要がある。この処置により、割り込みタスクの TSS がまだビジーであるとマークされている間に別の割り込みが発生するのを防ぐことができる。別の割り込みが発生すると、一般保護（#GP）例外が生成される原因となる。

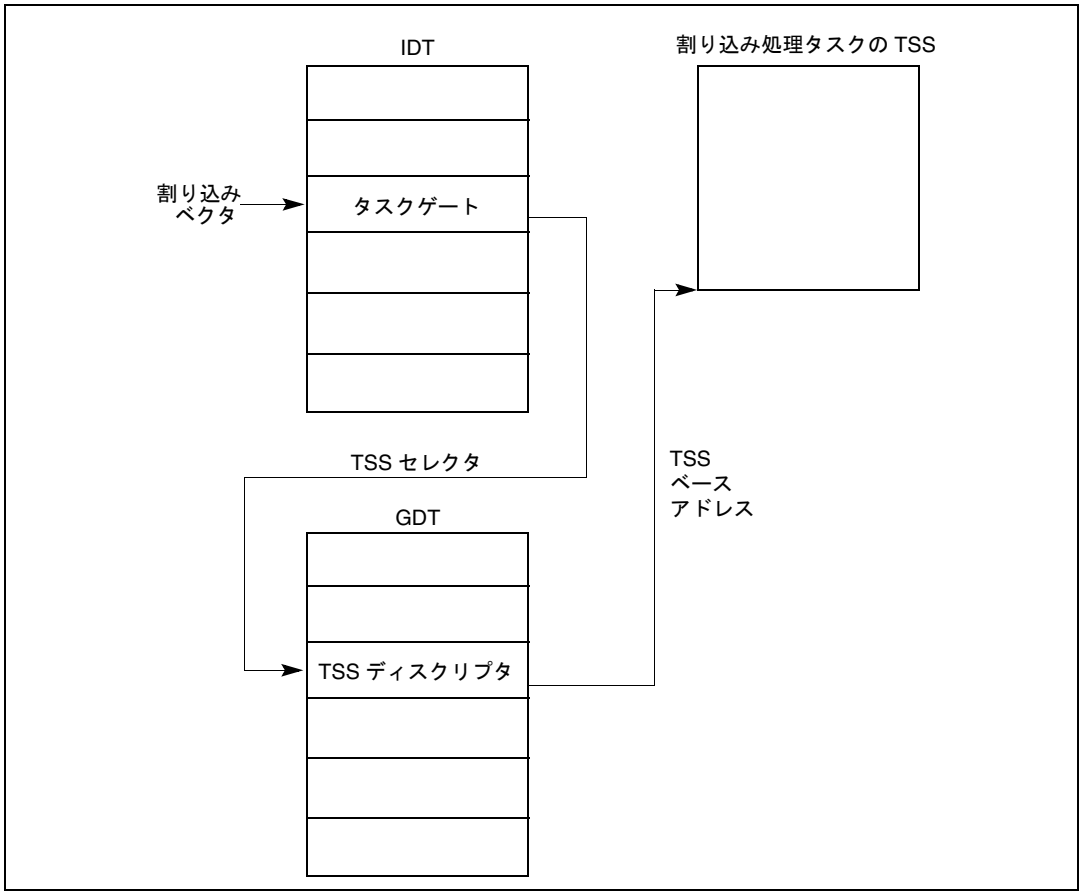


図 5-5. 割り込みのタスクスイッチ

5.13. エラーコード

例外条件が特定のセグメントに関連している場合、プロセッサは例外ハンドラ（それがプロシージャかタスクかにかかわらず）のスタックへエラーコードをプッシュする。エラーコードは図 5-6. に示すような形式を持つ。エラーコードはセグメント・セレクタに形式が似ているが、エラーコードの場合は RPL フィールドと TI フラグでなく、次の 3 つのフラグが入っている。

EXT　**外部イベントフラグ（ビット 0）。**これがセットされている場合、プログラムにとって外部的なイベント（ハードウェア割り込みなど）によって例外が発生したことを示す。

IDT　**ディスクリプタ位置フラグ（ビット 1）。**これがセットされている場合、エラーコードのインデックス部が IDT 内のゲート・ディスクリプタを参照することを表し、これがクリアされている場合は、インデックスが GDT または現行 LDT 内のディスクリプタを参照することを表している。

TI　**GDT/LDT フラグ（ビット 2）。**IDT フラグがクリアされている場合にだけ使用される。TI フラグがセットされている場合、エラーコードのインデックス部が LDT 内のセグメント・ディスクリプタまたはゲート・ディスクリプタを参照することを表し、これがクリアされている場合は、インデックスが現行 GDT 内のディスクリプタを参照することを表す。

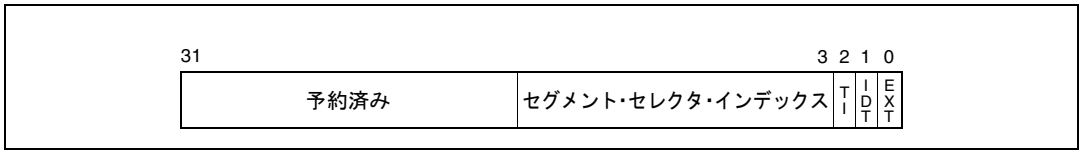


図 5-6. エラーコード

セグメント・セレクタ・インデックス・フィールドは、IDT、GDT または現行 LDT の中に、エラーコードが参照するセグメント・セレクタまたはゲートセレクタへのインデックスを提供する。エラーコードがヌルである場合は、エラーが特定のセグメントへの参照によって発生したのではないこと、または操作の中でヌル・セグメント・ディスクリプタが参照されたことを示す。

ページフォルト例外（#PF）の場合エラーコードの形式は異なる。これについては本章の「割り込み 14 ーページフォルト例外（#PF）」の項を参照。

エラーコードはダブルワードまたはワードとして（割り込み、トラップまたはタスクゲートのデフォルト・サイズに依存）スタック上にプッシュされる。ダブルワード・プッシュに備えてスタックのアライメントを合わせておくために、エラーコードの上位半分は予約済みになっている。例外ハンドラから復帰するために IRET 命令が実行

される場合はエラーコードがポップされないので、ハンドラは復帰を実行する前にエラーコードを削除しなければならないことに注意する。

外部で（INTR# または LINT[1:0] ピンを使用して）生成された例外または INT *n* 命令で生成された例外の場合は、たとえ通常どおりその例外についてエラーコードが生成されたとしても、エラーコードのスタックへのプッシュは行われない。

5.14. 例外と割り込みのリファレンス

以降の各項で、例外と割り込みを生成するさまざまな条件を記述する。条件はベクタ番号順に配列されており、各項には次の情報が提供されている。

例外クラス　例外クラスがフォルト、トラップ、アボートタイプのどれであるかを示す。例外によっては、エラー条件が検出された時点に応じてフォルトタイプになる場合と、トラップタイプになる場合がある。（この項は割り込みには適用されない。）

説明　そのタイプの例外や割り込みの目的を一般的に述べる。プロセッサがその例外や割り込みを処理する方法も説明する。

例外エラーコード
その例外についてエラーコードがセーブされるかどうかを示す。セーブされる場合はエラーコードの内容も説明する。（この項は割り込みには適用されない。）

セーブされている命令ポインタ
セーブされている（つまりリターン）命令ポインタがどの命令を指すかを示す。そのポインタを使用してフォルトを起こした命令を再スタートできるかどうかについても示す。

プログラム・ステートの変化
その例外や割り込みが現在実行中のプログラムまたはタスクのステートに及ぼす影響、およびそのプログラム / タスクを連続性を損なわずに再スタートできるかどうかについて説明する。

割り込み 0 ー除算エラー例外（#DE）

例外クラス　フォルト

説明

DIV または IDIV 命令の除数オペランドが 0 であること、またはデスティネーション・オペランド用に指定されたビット数では除算結果を表せないことを示す。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、その例外を生成した命令を指す。

プログラム・ステートの変化

除算エラー例外はフォルトを起こした命令が実行される前に起こるので、除算エラーにはプログラム・ステートの変化は付随しない。

割り込み 1 ーデバッグ例外（#DB）

例外クラス　トラップまたはフォルト。例外ハンドラは、レジスタ DR6 やその他のデバッグレジスタの内容を調べれば、トラップかフォルトかを判別できる。

説明

複数のデバッグ例外条件の 1 つ以上が検出されたことを表す。この例外がトラップであるかフォルトであるかは、起こった条件に依存する（表 5-3. を参照）。デバッグ例外の詳細については、第 15 章「デバッグと性能モニタリング」を参照。

表 5-3. デバッグ例外条件と対応する例外クラス

例外条件	例外クラス
命令フェッチ・ブレークポイント	フォルト
データ読み取りまたは書き込みブレークポイント	トラップ
I/O 読み取りまたは書き込みブレークポイント	トラップ
一般的な検出条件（インサーキット・エミュレーションと結合）	フォルト
シングルステップ	トラップ
タスクスイッチ	トラップ

例外エラーコード

なし。例外ハンドラはデバッグレジスタを調べ、どの条件がその例外を引き起こしたか特定できる。

セーブされている命令ポインタ

フォルトの場合ーレジスタ CS と EIP にセーブされている内容は、その例外を生成した命令を指す。

トラップの場合ーレジスタ CS と EIP にセーブされている内容は、その例外を生成した命令の次の命令を指す。

プログラム・ステートの変化

フォルトの場合ーデバッグ例外はフォルトを起こした命令が実行される前に発生するので、デバッグ例外にはプログラム・ステートの変化は付随しない。プログラムは、デバッグ例外ハンドラから復帰した時点で、通常の実行を再開できる。

トラップの場合ーデバッグ例外が生成される前に命令やタスクの切り替えを完了できるので、デバッグ例外にはプログラム・ステートの変化が付随する。しかし、プログラムの新しいステートは破壊されず、安全にプログラムの実行を続行できる。

割り込み 2 ― NMI 割り込み

例外クラス　該当せず。

説明

マスク不可能割り込み（NMI）は、プロセッサの NMI# ピンをアサートするか、I/O APIC によってセットされたローカル APIC への NMI 要求を介して外部的に生成される。この割り込みが起こると NMI 割り込みハンドラが呼び出される。

例外エラーコード

該当せず。

セーブされている命令ポインタ

プロセッサは常に命令境界で NMI 割り込みを受け取る。レジスタ CS と EIP にセーブされている内容は、割り込みが受け取られたポイントから実行される次の命令を指す。プロセッサが NMI 割り込みを受け取る時点の詳細については、5.5. 節「例外の分類」を参照。

プログラム・ステートの変化

NMI 割り込みが受け取られた時点で実行中の命令は、NMI が生成される前に終了する。したがって、割り込みハンドラから復帰した時点で、連続性を損なわずにプログラムまたはタスクを再スタートできる。ただしこれは、割り込みハンドラが割り込みを処理する前にプロセッサのステートをセーブしておき、復帰を行う前にそのプロセッサのステートをリストアすることが前提である。

割り込み 3 ーブレークポイント例外（#BP）

例外クラス　　トラップ

説明

ブレークポイント命令（INT 3）が実行され、そのためにブレークポイント・トラップが発生したことを示す。通常デバッガは、ある命令の最初のオペコード・バイトの代りに INT 3 命令のオペコードを置き換えることにより、ブレークポイントを設定する。（INT 3 命令の長さは 1 バイトなので、RAM のコード・セグメント内のオペコードをブレークポイント・オペコードで置き換えるのは簡単である。）オペレーティング・システムやデバッグツールは、コード・セグメントと同じ物理アドレス空間にマッピングされたデータ・セグメントを使用して、デバッガを呼び出す必要のある場所に INT 3 命令を配置できる。

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™プロセッサでは、デバッグレジスタでブレークポイントを設定する方が便利である。（ブレークポイント例外の詳細については、15.3.2. 項「ブレークポイント例外 ー（#BP）割り込みベクタ 3」を参照。）デバッグレジスタで設定できる数よりも多くのブレークポイントが必要な場合は、INT 3 命令を使用できる。

オペランドを 3 として INT *n* 命令を実行してもブレークポイント例外（#BP）を生成できる。この命令（INT 3）の動作は、INT 3 命令の動作とはわずかに異なる（『IA-32 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、中巻 A』の第 3 章の「INT*n*/INTO/INT 3—Call to Interrupt Procedure」を参照のこと）。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、INT 3 命令の次の命令を指す。

プログラム・ステートの変化

EIP がブレークポイント命令の次の命令を指しても、プログラム・ステートは本質的に無変化である。これは、INT 3 命令はレジスタやメモリの位置には全く影響しないからである。このようにしてデバッガは、ブレークポイントを引き起こした INT 3 命令を元のオペコードで置き換え、EIP レジスタのセーブ内容を減らすことで、保留のプログラムを再開できる。デバッガから復帰すると、置き換わった命令からプログラム実行が再開される。

割り込み 4 ーオーバーフロー例外（#OF）

例外クラス　　トラップ

説明

INTO 命令が実行された時点でオーバーフロー・トラップが発生したことを示す。INTO 命令は EFLAGS レジスタの OF フラグのステートをチェックする。OF フラグがセットされていればオーバーフロー・トラップが発生する。

演算命令には、符号付きと符号なし演算を両方行うものがある（ADD、SUB など）。これらの命令では、符号付きオーバーフロー、符号なしオーバーフローを表すのに、EFLAGS レジスタの OF フラグ、CF フラグをそれぞれセットする。符号付きオペランドで演算を実行した場合は、直接に OF フラグをテストしても、INTO 命令を使用してもよい。INTO 命令使用のメリットは、オーバーフロー例外が検出された場合は自動的に例外ハンドラを呼び出してオーバーフロー条件を処理できる点にある。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、INTO 命令の次の命令を指す。

プログラム・ステートの変化

EIP が INTO 命令の次の命令を指しても、プログラム・ステートは本質的に無変化である。これは、INTO 命令はレジスタやメモリの位置には全く影響しないからである。このようにして、オーバーフロー例外ハンドラから復帰した時点で、プログラムの実行を再開できる。

割り込み 5 ー BOUND 範囲超過例外（#BR）

例外クラス　　フォルト

説明

BOUND 命令が実行された時点で BOUND 範囲超過フォルトが発生したことを示す。BOUND 命令は、符号付きの配列インデックスを、配列の上下バウンドの符号付きリミットを基準にチェックする。配列インデックスが配列のバウンド内部にない場合は、BOUND 範囲超過フォルトが発生する。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、例外を生成した BOUND 命令を指す。

プログラム・ステートの変化

BOUND 命令のオペランドは変更されないので、バウンド・チェック・フォルトにはプログラム・ステートの変化は付随しない。BOUND 範囲超過例外ハンドラから復帰すると BOUND 命令が再スタートされる。

割り込み 6 ー無効オペコード例外（#UD）

例外クラス　フォルト

説明

プロセッサが次のどれかを行ったことを示す。

- 無効なオペコードまたは予約されているオペコードを実行しようとした。
- 付随するオペコードに対して無効なオペランド・タイプを指定して命令を実行しようとした。例えば、LES 命令のソース・オペランドはメモリ位置ではない。
- MMX® テクノロジ、SSE、SSE2、SSE3 の拡張命令をサポートしない IA-32 プロセッサ上で、MMX 命令、SSE、SSE2、SSE3 の命令を実行しようとした。CPUID 機能フラグの MMX 命令（ビット 23）、SSE（ビット 25）、SSE2（ビット 26）、SSE3（ECX、ビット 0）は、これらの拡張機能のサポートの有無を示す。
- 制御レジスタ CR0 の EM フラグが 1 にセットされているときに、MMX 命令、または SSE、SSE2、SSE3 の SIMD 命令を実行しようとした（ただし、MOVNTI、PAUSE、PREFETCH*h*、SFENCE、LFENCE、MFENCE、CLFLUSH は除く）。
- 制御レジスタ CR4 の OSFXSR ビットが 0 にクリアされているときに、SSE 命令、SSE2 命令または SSE3 命令を実行しようとした。ただし、これには、次の SSE 命令、SSE2 命令、SSE3 命令は含まれない。MASKMOVQ、MOVNTQ、MOVNTI、PREFETCH*h*、SFENCE、LFENCE、MFENCE、CLFLUSH、または、64 ビット版の PAVGB、PAVGW、PEXTRW、PINSRW、PMAXSW、PMAXUB、PMINSW、PMINUB、PMOVMSKB、PMULHUW、PSADBW、PSHUFW、PADDQ、PSUBQ。
- 制御レジスタ CR4 の OSXMMEXCPT ビットが 0 にクリアされているときに、SIMD 浮動小数点例外を発生させる SSE 命令、SSE2 命令、または SSE3 命令を IA-32 プロセッサ上で実行しようとした。
- UD2 命令を実行した。無効オペコード例外を発生させる UD2 命令が実行されても、保存された命令ポインタは UD2 命令を指していることに注意する。
- ロックされない可能性がある、またはロックできるがデスティネーション・オペランドがメモリ位置でないような命令の前に、LOCK プリフィックスが付いているのを検出した。
- 実アドレスモードや仮想 8086 モードで操作中に、LLDT、SLDT、LTR、STR、LSL、LAR、VERR、VERW または ARPL 命令を実行しようとした。
- SMM モードでないときに RSM 命令を実行しようとした。

割り込み 6 ー無効オペコード例外（#UD）（続き）

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、この例外は、無効な命令の実行の結果をリタイアメントする試みがなされた後で発生する。つまり、無効なオペコードをデコードして推論的に実行しようとした場合、この例外は発生しない。同様に、インテル® Pentium® プロセッサおよび以前の IA-32 プロセッサでは、無効な命令のプリフェッチおよび事前デコードの結果として、この例外が発生することはない。（割り込みと例外の受け取りに関する一般規則については、5.5. 節「例外の分類」を参照。）

オペコード D6 と F1 は、IA-32 アーキテクチャが予約している未定義のオペコードである。これらのオペコードは未定義ではあるが、これを使用しても無効オペコード例外は発生しない。

UD2 命令は、必ず無効オペコード例外を発生するように保証されている。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、例外を生成した命令を指す。

プログラム・ステートの変化

無効な命令は結局実行されないので、無効オペコード・フォルトにはプログラム・ステートの変化は付随しない。

割り込み 7 ーデバイス使用不可例外（#NM）

例外クラス　フォルト

説明

次の事柄のどれかを示す。

デバイス使用不可能例外は、次の 3 つの条件のいずれかによって発生する。

- 制御レジスタ CR0 の EM フラグがセットされた状況で、プロセッサが x87 FPU 浮動小数点命令を実行した。WAIT/FWAIT 命令の特殊なケースについては、下のパラグラフを参照。
- レジスタ CR0 の MP フラグと TS フラグがセットされた状態で（EM フラグの設定には関わらない）、プロセッサが WAIT/FWAIT 命令を実行した。
- 制御レジスタ CR0 の TS フラグをセットし、EM フラグがクリアされた状態で、プロセッサが x87 FPU、MMX® 命令、SSE、SSE2、SSE3 の命令を実行した（ただし、MOVNTI、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、CLFLUSH は除く）。

EM フラグは、プロセッサが内部 x87 FPU 浮動小数点ユニットを持たない場合にセットされる。この場合 x87 FPU 浮動小数点命令が検出されるたびにデバイス使用不可例外が生成され、例外ハンドラで浮動小数点命令エミュレーション・ルーチンを呼び出すことが可能になる。

TS フラグは、最後に x87 浮動小数点命令、MMX 命令、SSE、SSE2、または SSE3 を実行した後に、コンテキスト・スイッチ（タスクスイッチ）が発生したが、x87 FPU、XMM、MXCSR レジスタのコンテキストはセーブされなかったことを示す。TS フラグがセットされていて、EM フラグがクリアされている場合、プロセッサは、x87 浮動小数点命令、MMX 命令、SSE、SSE2、または SSE3 を（上記の命令の例外を伴って）検出するたびに、デバイス使用不可例外を生成する。すると、その命令が実行される前に例外ハンドラが x87 FPU、XMM、MXCSR レジスタのコンテキストをセーブできる。TS フラグの詳細については、2.5. 節「制御レジスタ」を参照。

制御レジスタ CR0 の MP フラグは、TS フラグとあわせて使用され、WAIT または FWAIT 命令がデバイス使用不可例外を生成すべきかどうかを決定する。このことにより TS フラグの機能は WAIT および FWAIT 命令にまで拡張され、それによって WAIT または FWAIT 命令が実行される前に x87 FPU のコンテキストをセーブする機会が例外ハンドラに与えられる。MP フラグは主にインテル® 286 プロセッサと Intel386™ DX プロセッサで使用するように用意されたものである。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサや Intel486™ DX プロセッサ、またはインテル® 487 SX コプロセッサで実行するプログラムでは、常に MP フラグをセットする必要がある。Intel486 SX プロセッサで実行するプログラムでは、MP フラグをクリアしておく必要がある。

割り込み 7 ーデバイス使用不可例外（#NM）（続き）

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、例外を生成した浮動小数点命令または WAIT/FWAIT 命令を指す。

プログラム・ステートの変化

例外を生成した命令は結局実行されないので、デバイス使用不可能フォルトにはプログラム・ステートの変化は付随しない。

EM フラグがセットされている場合、例外ハンドラは EIP が指している浮動小数点命令を読み取って適切なエミュレーション・ルーチンを呼び出せる。

MP と TS フラグの両方、または TS フラグだけがセットされている場合、例外ハンドラは、x87 FPU のコンテキストをセーブし、TS フラグをクリアしてから、浮動小数点命令または WAIT/FWAIT 命令が割り込まれたポイントから実行を続行できる。

割り込み 8 ーダブルフォルト例外（#DF）

例外クラス　　アボート

説明

プロセッサが、ある例外の例外ハンドラを呼び出している最中に次の例外を検出したことを表す。通常では、プロセッサが例外ハンドラの呼び出し中に次の例外を検出した場合は、2 つの例外を順番に処理することができる。しかしプロセッサがこれらを逐次処理できない場合、このダブルフォルト例外の信号を発する。2 つのフォルトをダブルフォルトとして信号を送らなければならない場合を判断するため、プロセッサは例外を、良性例外（Benign）、寄与的例外（Contributory）、ページフォルトという 3 クラスに分類する（表 5-4. を参照）。

表 5-4. 割り込み / 例外のクラス

クラス	ベクタ番号	説明
良性例外 / 割り込み	1 2 3 4 5 6 7 9 16 17 18 19 すべて すべて	デバッグ NMI 割り込み ブレークポイント オーバーフロー BOUND 範囲超過 無効オペコード デバイス使用不可能 コプロセッサ・セグメント・オーバーラン 浮動小数点エラー アライメント・チェック マシンチェック SIMD 浮動小数点 INT *n* INTR
寄与的例外	0 10 11 12 13	除算エラー 無効 TSS セグメント不在 スタックフォルト 一般保護
ページフォルト	14	ページフォルト

表 5-5. に、ダブルフォルトが生成される原因となる例外クラスのさまざまな組み合わせを示す。ダブルフォルト例外自体は、アボートクラスの例外にあたる。つまりあとでプログラムやタスクを再スタートまたは再開できない。この場合ダブル・フォルト・ハンドラを使用して、マシンのステートに関する診断情報を収集したり、可能な場合はアプリケーションやシステムを既定の手順にしたがってシャットダウンしたり、システムを再起動できる。

割り込み 8 ーダブルフォルト例外（#DF）（続き）

命令をプリフェッチしている最中に検出されるセグメント・フォルトやページフォルトは、表 5-5. の範囲外である。プロセッサが適切なフォルトハンドラに制御を移行しようとしたときにさらにフォルトが発生した場合も、ダブルフォルトが生成される。

表 5-5. ダブルフォルト生成の条件

	後で発生する例外		
最初に発生する例外	良性例外	寄与的例外	ページフォルト
良性例外	例外を逐次処理する	例外を逐次処理する	例外を逐次処理する
寄与的例外	例外を逐次処理する	ダブルフォルトを生成する	例外を逐次処理する
ページフォルト	例外を逐次処理する	ダブルフォルトを生成する	ダブルフォルトを生成する

ダブル・フォルト・ハンドラの呼び出しを試みている最中に別の例外が起こった場合、プロセッサはシャットダウン・モードに入る。このモードは、HLT 命令実行直後のステートに類似している。このモードではプロセッサは、NMI 割り込み、SMI 割り込み、ハードウェア・リセット、INIT# のどれかを受け取るまで命令の実行を停止する。プロセッサは特殊なバスサイクルを生成し、自分がシャットダウン・モードに入ったことを知らせる。ソフトウェア設計者は、ソフトウェアがシャットダウン・モードに入った場合のハードウェアの反応も熟知しておく必要がある。例えば、ハードウェアによって、フロントパネルのインジケータ・ランプを点灯する、NMI 割り込みを生成して診断情報を記録する、リセット初期化を起動する、INIT 初期化を生成する、または SMI を生成することができる。シャットダウン中にいずれかのイベントが保留されている場合、そのイベントはシャットダウンからのウェイクイベントが処理された後に処理される（例えば、A20M# 割り込み）。

プロセッサが NMI 割り込みハンドラを実行中にシャットダウンが起こった場合は、ハードウェア・リセットを行う以外にプロセッサを再スタートできない。同様に、SMM で実行中にシャットダウンが起こった場合は、プロセッサを再スタートさせるためにハードウェア・リセットを行う必要がある。

例外エラーコード

ゼロ。プロセッサはダブル・フォルト・ハンドラのスタックには常にエラーコード 0 をプッシュする。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は未定義である。

割り込み 8 ーダブルフォルト例外（#DF）（続き）

プログラム・ステートの変化

ダブルフォルト例外発生後のプログラム・ステートは未定義である。プログラムやタスクは再開も再スタートもできない。ダブルフォルト例外ハンドラに許される動作は、診断時に使用できるすべてのコンテキスト情報を収集した後、アプリケーションを終了し、プロセッサをシャットダウンまたはリセットするだけである。

例外処理マシンステートのいずれかの部分が壊されたときにダブルフォルトが発生した場合は、ハンドラが起動することはできず、プロセッサをリセットしなければならない。

割り込み 9 ーコプロセッサ・セグメント・オーバーラン

例外クラス　アボート。（**インテルにより予約済み。使用禁止。現在の IA-32 プロセッサはこの例外を生成しない。**）

説明

Intel386™ プロセッサの CPU をベースにし、インテル® 387 マス・コプロセッサを備えたシステムが、387 マス・コプロセッサのオペランドの中央部分を転送中に、ページまたはセグメント違反を検出したことを表す。P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサではこの例外は生成されないが、代りに割り込み 13 の一般保護例外（#GP）でこの条件が検出される。

例外エラーコード

なし。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、例外を生成した命令を指す。

プログラム・ステートの変化

コプロセッサ・セグメント・オーバーラン例外発生後のプログラム・ステートは未定義である。プログラムやタスクは再開も再スタートもできない。例外ハンドラに許される動作は、命令ポインタをセーブし、FNINIT 命令を使用して x87 FPU を再初期化することだけである。

割り込み 10 －無効 TSS 例外（#TS）

例外クラス　フォルト

説明

タスクスイッチを実行しようとし、ターゲット・タスクのTSS内に無効な情報が検出されたことを示す。表5-6.に、無効TSS例外を生成させる条件を示す。一般的に、これらの無効条件は、TSSディスクリプタ、TSSディスクリプタによって示されるLDT、あるいはTSSが参照したスタック、コードまたはデータ・セグメントに対応する保護違反の結果生じるものである。

表 5-6. 無効 TSS 例外の条件

エラー・コード・インデックス	無効条件
TSS セグメント・セレクタ・インデックス	TSS セグメントのリミットが 32 ビット TSS で 67H より小さいか、16 ビット TSS で 2CH より小さい
TSS セグメント・セレクタ・インデックス	IRET タスクスイッチ中に、TSS セグメント・セレクタ内の TI フラグが LDT を示す
TSS セグメント・セレクタ・インデックス	IRET タスクスイッチ中に、TSS セグメント・セレクタがディスクリプタ・テーブル・リミットを超過
TSS セグメント・セレクタ・インデックス	IRET タスクスイッチ中に、TSS ディスクリプタ内のビジーフラグが非アクティブなタスクを示す
LDT セグメント・セレクタ・インデックス	LDT が無効または不在
スタック・セグメント・セレクタ・インデックス	スタック・セグメント・セレクタがディスクリプタ・テーブル・リミットを超過
スタック・セグメント・セレクタ・インデックス	スタック・セグメントが書き込み不可能
スタック・セグメント・セレクタ・インデックス	スタック・セグメントの DPL が CPL に不適合
スタック・セグメント・セレクタ・インデックス	スタック・セグメント・セレクタの RPL が CPL に不適合
コード・セグメント・セレクタ・インデックス	コード・セグメント・セレクタがディスクリプタ・テーブル・リミットを超過
コード・セグメント・セレクタ・インデックス	コード・セグメントが実行不可能
コード・セグメント・セレクタ・インデックス	非準拠コード・セグメントの DPL が CPL に不適合
コード・セグメント・セレクタ・インデックス	コンフォーミング・コード・セグメントの DPL が CPL より高い
データ・セグメント・セレクタ・インデックス	データ・セグメント・セレクタがディスクリプタ・テーブル・リミットを超過
データ・セグメント・セレクタ・インデックス	データ・セグメントが読み取り不可能

割り込み 10 ー無効 TSS 例外（#TS）（続き）

この例外は元のタスクのコンテキスト内でも新しいタスクのコンテキスト内でも発生しうる（6.3. 節「タスク・スイッチング」を参照）。プロセッサが完全に新しいTSSの存在を確認し終わるまでは、例外は元のタスクのコンテキスト内で生成される。いったん新しいTSSの存在が確認されたら、タスクスイッチが完了したと見なされる。このポイント以降に無効TSS条件が検出されるとそれは新しいタスクのコンテキスト内で処理される。（タスクスイッチは、タスクレジスタに新しい TSS のセグメント・セレクタがロードされた時点で完了したと見なされ、その切り替えがプロシージャ呼び出しまたは割り込みによるものであれば、新しいTSSのリンク・フィールドは古いTSSを参照する。）

無効TSSハンドラは、タスクゲートを使用して呼び出されたタスクでなければならない。フォルトが起こっているTSSコンテキスト内でこの例外を処理することは、プロセッサ・ステートに一貫性がない場合があるので推奨しない。

例外エラーコード

違反を発生させたセグメント・ディスクリプタに対応するセグメント・セレクタのインデックスを含むエラーコードが、例外ハンドラのスタックにプッシュされる。EXT フラグがセットされている場合、それはこの例外が現在実行中のプログラムにとって外部的なイベントによって起こったことを示す（例えば、タスクゲートを使用する外部割り込みハンドラが、無効なTSSへタスクスイッチを試みた）。

セーブされている命令ポインタ

タスクスイッチが実行される前に例外条件が検出された場合、レジスタ CS と EIP にセーブされている内容は、タスクスイッチを起動した命令を指す。タスクスイッチが実行された後に例外条件が検出された場合は、レジスタCSとEIPにセーブされている内容は、新しいタスクで最初に実行される命令を指す。

プログラム・ステートの変化

無効TSSハンドラがフォルトから回復する可能性は、フォルトの原因となるエラー条件にかかっている。タスク・スイッチプロセスと考えられる回復措置については、6.3. 節「タスク・スイッチング」を参照。

割り込み 10 －無効 TSS 例外（#TS）（続き）

タスクスイッチ中に無効TSS例外が起こる場合、それは「新しいタスクに切り替える」ポイントの前でも後でも起こる可能性がある。切り替えポイントの手前で起こった場合は、プログラム・ステートの変化は生じない。切り替えポイントの後（つまり新しいセグメント・セレクタのセグメント・ディスクリプタ情報がすでにセグメント・レジスタにロードされている）で起こった場合は、プロセッサは新しいTSSからすべてのステート情報をロードしてから、例外を生成する。タスクスイッチ中、プロセッサはまずすべてのセグメント・レジスタにTSSからのセグメント・セレクタをロードし、次いでその内容の妥当性をチェックする。無効TSS例外が発見された場合、残りのセグメント・レジスタはロードされるが妥当性のチェックは行われず、したがってこのデータはメモリ参照には有効でない。無効TSSハンドラは、他の例外を発生させずに、CS、SS、DS、ES、FS、GSレジスタの中で見つかったセグメント・セレクタを使用できることを前提としてはならない。例外ハンドラは新しいタスクの再開を試みる前にすべてのセグメント・レジスタをロードする必要がある。そうしないと、後で、もっと診断しにくい条件で一般保護例外（#GP）が起こる可能性がある。インテルが推奨するこの状況の対処方法は、無効TSS例外ハンドラ用のタスクを使用することである。無効TSS例外ハンドラのタスクから割り込まれたタスクに戻るタスクスイッチが行われると、プロセッサはTSSからレジスタをロードするときにレジスタをチェックする。

割り込み 11 ーセグメント不在例外（#NP）

例外クラス　　フォルト

説明

セグメントまたはゲートのディスクリプタの存在フラグが、クリアされていることを示す。プロセッサは、次のどの操作の最中でもこの例外を生成できる。

- CS、DS、ES、FS または GS レジスタをロードしようとしているとき。[SS レジスタをロードしている最中に不在セグメントを検出すると、スタックフォルト例外（#SS）が生成される。] この状況はタスクスイッチ実行中も起こる場合がある。
- LLDT 命令を使用して LDTR をロードしようとしているとき。タスクスイッチ操作中に LDTR をロードしていて不在 LDT を検出すると、無効 TSS 例外（#TS）が生成される。
- LTR 命令を実行中であって TSS が不在とマークされている場合。
- セグメント不在とマークされ、それ以外は有効なゲート・ディスクリプタまたは TSS を使用しようとしているとき。

オペレーティング・システムは通常、セグメント不在例外を使用してセグメント・レベルの仮想メモリを実現する。例外ハンドラがセグメントをロードしてから復帰した場合、割り込まれたプログラムまたはタスクは、実行を再開する。

しかし、ゲート・ディスクリプタの中にある不在の指標は、セグメントが存在しないことを示さない（ゲートがセグメントに対応していないため）。オペレーティング・システムはゲート・ディスクリプタに対応する存在フラグを使用して、オペレーティング・システムにとって特別な重要性を持つ例外をトリガする可能性がある。

そのような重要な例外またはページフォルトがその後に不在セグメントを参照すると、#NP ではなくダブルフォルト（#DF）が発生する。

例外エラーコード

違反を発生させたセグメント・ディスクリプタに対応するセグメント・セレクタのインデックスを含むエラーコードが、例外ハンドラのスタックにプッシュされる。EXT フラグがセットされている場合は、例外が次のいずれかの原因で発生したことを示す。

- 外部イベント（NMI または INTR）によって割り込みが発生し、その割り込みがその後に不在セグメントを参照した。
- 害のない例外がその後に不在セグメントを参照した。

割り込み 11 ーセグメント不在例外（#NP）（続き）

エラーコードがIDTエントリを参照する場合は、IDTフラグがセットされる。この状態は、処理されている割り込みのIDTエントリが不在ゲートを参照するときに発生する。このようなイベントは、INT命令またはハードウェア割り込みによって発生する。

セーブされている命令ポインタ

レジスタCSとEIPにセーブされている内容は、通常は例外を生成した命令を指す。新しいTSS内のセグメント・セレクタに対応するセグメント・ディスクリプタをロードしている最中に例外が起こった場合、レジスタCSとEIPは、新しいタスクの最初の命令を指す。ゲート・ディスクリプタにアクセス中に例外が起こった場合、レジスタCSとEIPは、アクセスを起動した命令（例えばコールゲートを参照したCALL命令）を指す。

プログラム・ステートの変化

レジスタ（CS、DS、SS、ES、FS、GSまたはLDTR）をロードした結果セグメント不在例外が起こった場合、レジスタはロードされていないので、この例外にはプログラム・ステートの変化は付随しない。不在になっているセグメントをメモリにロードし、そのセグメント・ディスクリプタ内の存在フラグをセットするだけで、この例外からの回復は可能である。

ゲート・ディスクリプタにアクセス中にセグメント不在例外が起こった場合、この例外にはプログラム・ステートの変化は付随しない。そのゲート・ディスクリプタ内の存在フラグをセットするだけで、この例外からの回復は可能である。

タスクスイッチ中にセグメント不在例外が起こる場合、それは「新しいタスクに切り替える」ポイントの前でも後でも起こる可能性がある（6.3. 節「タスク・スイッチング」を参照）。切り替えポイントの手前で起こった場合は、プログラム・ステートの変化は生じない。切り替えポイントの後で起こった場合は、プロセッサは新しいTSSからすべてのステート情報を（追加のリミット、存在、タイプチェックを一切行わずに）ロードしてから、例外を生成する。セグメント不在例外ハンドラは、他の例外を発生させずに、CS、SS、DS、ES、FS、GSレジスタの中にあるセグメント・セレクタを使用できることを前提としてはならない。（この状況を処理する方法の詳細については、本章の「割り込み 10 ー無効 TSS 例外（#TS）」の「プログラム・ステートの変化」の説明を参照。）

割り込み 12 ースタックフォルト例外（#SS）

例外クラス　　フォルト

説明

次のスタック関連の条件の 1 つが検出されたことを表す。

- SS レジスタを参照する操作中にリミット違反が検出された。リミット違反を引き起こす可能性のある操作には、POP、PUSH、CALL、RET、IRET、ENTER、LEAVE のようなスタック操作を伴う命令や、SS レジスタを暗黙または明示的に使用するその他のメモリ参照命令（例えば MOV AX,[BP+6] または MOV AX,SS:[EAX+6]）などがある。ENTER 命令の場合、ローカル変数を割り当てるだけのスタック空間がないときにこの例外を生成する。
- SS レジスタをロードしようとしているときに不在スタック・セグメントが検出された。この違反は、タスクスイッチの実行中、異なる特権レベルへの CALL 命令、異なる特権レベルへの戻り、LSS 命令、または SS レジスタに対する MOV や POP 命令の実行中に起こる可能性がある。

このフォルトから回復するには、スタック・セグメントのリミットを拡張する（リミット違反の場合）か不在のスタック・セグメントをメモリにロードする（不在違反の場合）のどちらかを行えばよい。

例外エラーコード

スタック・セグメントの不在、または特権レベル間呼び出し中に新しいスタックがオーバーフローしたためにこの例外が引き起こされた場合、エラーコードには、例外を引き起こしたセグメントを指すセグメント・セレクタが入る。この場合例外ハンドラは、セグメント・セレクタが指すセグメント・ディスクリプタの存在フラグをテストし、例外の原因を判断する。通常のリミット違反（すでに使用中のスタック・セグメントに関して起こった違反）の場合、エラーコードは 0 に設定される。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、通常は例外を生成した命令を指す。しかし、この例外がタスクスイッチ中に不在スタック・セグメントをロードしようと試みたことによるものである場合、レジスタ CS と EIP は、新しいタスクの最初の命令を指す。

プログラム・ステートの変化

フォルトを生成した命令は結局実行されないので、一般にスタックフォルト例外にはプログラム・ステートの変化は付随しない。この場合は、例外ハンドラがスタックフォルト条件を修正し終った後、命令を再スタートできる。

割り込み 12 ースタックフォルト例外（#SS）（続き）

タスクスイッチ中にスタックフォルトが起こる場合、それは「新しいタスクに切り替える」ポイントの後で発生する（6.3. 節「タスク・スイッチング」を参照）。この場合、プロセッサは新しい TSS からすべてのステート情報を（追加のリミット、存在、タイプチェックを一切行わずに）ロードしてから、この例外を生成する。スタック・フォルト・ハンドラは、他の例外を発生させずに、CS、SS、DS、ES、FS、GS レジスタの中で見つかったセグメント・セレクタを使用できることを前提としてはならない。例外ハンドラは、新しいタスクの再開を試みる前にすべてのセグメント・レジスタをチェックする必要がある。そうしないと、後で、もっと診断しにくい条件で一般保護フォルトが起こる可能性がある。(この状況を処理する方法の詳細については、本章の「割り込み 10 ー無効 TSS 例外（#TS)」の「プログラム・ステートの変化」の説明を参照。)

割り込み 13 ー一般保護例外（#GP）

例外クラス　　フォルト

説明

プロセッサが、「一般保護違反」と呼ばれるクラスの保護違反の 1 つを検出したことを表す。この例外を発生させる条件は、保護違反のうち他の例外（無効 TSS、セグメント不在、スタックフォルトまたはページフォルト例外）を発生させないものすべてで構成される。次の条件があると一般保護例外が生成される。

- CS、DS、ES、FS または GS セグメントにアクセスするときにセグメント・リミットを超過する。
- ディスクリプタ・テーブルを参照するときにセグメント・リミットを超過する（タスクスイッチまたはスタック切り替えの場合を除く）。
- 実行を実行可能でないセグメントに転送する。
- コード・セグメントまたは読み取り専用のデータ・セグメントに書き込みをする。
- 実行専用のコード・セグメントから読み取りをする。
- SS レジスタに読み取り専用のセグメントのセグメント・セレクタをロードする（そのセレクタがタスクスイッチ中に TSS から抽出したものである場合は除く。この場合は無効 TSS 例外が起こる）。
- SS、DS、ES、FS または GS レジスタに、システム・セグメントのセグメント・セレクタをロードする。
- DS、ES、FS または GS レジスタに、実行専用コード・セグメントのセグメント・セレクタをロードする。
- SS レジスタに、実行可能セグメントのセグメント・セレクタまたはヌル・セグメント・セレクタをロードする。
- CS レジスタに、データ・セグメントのセグメント・セレクタまたはヌル・セグメント・セレクタをロードする。
- ヌル・セグメント・セレクタが入っている状態の DS、ES、FS または GS レジスタを使用して、メモリにアクセスする。
- TSS への呼び出しまたはジャンプ中に、ビジーなタスクへ切り替える。
- IRET 命令でないタスクスイッチ時に、現行 LDT 内の TSS ディスクリプタを指すセグメント・セレクタを使用する。TSS ディスクリプタは GDT 内にしかない。この条件によって、IRET タスクスイッチの間に #TS 例外が発生する。
- 第 4 章「保護」に記述されている特権規則のどれかに違反する。
- 15 バイトという命令長のリミットを超過する（これは命令の前に冗長プリフィックスが付けられる場合にだけ起こる）。

割り込み 13 ーー一般保護例外（#GP）（続き）

- レジスタ CR0 に、セット状態の PG フラグ（ページング・イネーブル）とクリア状態の PE フラグ（保護ディスエーブル）をロードする。
- レジスタ CR0 に、セット状態の NW フラグとクリア状態の CD フラグをロードする。
- （割り込み / 例外発生後に）IDT 内の割り込み、トラップまたはタスクゲートでないエントリを参照する。
- 割り込み / 例外ハンドラのコード・セグメント DPL が 0 より大きいときに、仮想 8086 モードから割り込みまたはトラップゲートを通じてそのハンドラにアクセスしようとする。
- レジスタ CR4 内の予約ビットに 1 を書き込もうとする。
- CPL が 0 に等しくないときに特権命令を実行しようとする（特権命令のリストについては 4.9. 節「特権命令」を参照）。
- MSR 内の予約ビットに書き込みを行う。
- ヌル・セグメント・セレクタの入ったゲートにアクセスする。
- CPL が、参照される割り込み、トラップまたはタスクゲートの DPL より高いときに INT *n* 命令を実行する。
- 呼び出し、割り込みまたはトラップゲート内のセグメント・セレクタがコード・セグメントを指さない。
- LLDT 命令内のセグメント・セレクタ・オペランドが、ローカルタイプである（TI フラグがセットされている）か、LDT タイプのセグメント・ディスクリプタを指さない。
- LTR 命令内のセグメント・セレクタ・オペランドがローカルタイプであるか、使用可能でない TSS を指す。
- 呼び出し、ジャンプまたは復帰のターゲット・コード・セグメント・セレクタがヌルである。
- 制御レジスタ CR4 内の PAE フラグや PSE フラグがセットされており、ページ・ディレクトリ・ポインタ・テーブルのエントリ内の予約ビットのどれかが 1 に設定されているのをプロセッサが検出する。これらのビットは、制御レジスタ CR0、CR3 または CR4 への書き込み中にチェックされる。これらの書き込みはページ・ディレクトリ・ポインタ・テーブルのエントリの再ロードを引き起こす。
- MXCSR レジスタの予約ビットに 0 以外の値を書き込もうとする。
- 16 バイトのアライメントを必要とする SSE 命令、SSE2 命令、または SSE3 命令において、16 バイト境界にアライメントが合っていない 128 ビット・メモリ位置へアクセスしようとする命令を実行する。この条件は、スタック・セグメントにも適用される。

割り込み 13 ー一般保護例外（#GP）（続き）

一般保護例外の後はプログラムやタスクを再スタートできる。割り込みハンドラへの呼び出しを試みている間に例外が起こった場合、割り込まれたプログラムは再スタート可能であるが、割り込みの方は失われることがある。

例外エラーコード

プロセッサは例外ハンドラのスタックにエラーコードをプッシュする。セグメント・ディスクリプタへのロード中にフォルト条件が検出された場合、エラーコードには、そのディスクリプタを指すセグメント・セレクタが入り、そうでない場合エラーコードは0になる。エラーコードに入るセレクタのソースは、次のうちのどれでもよい。

- 命令のオペランド
- 命令のオペランドとなるゲートから取ったセレクタ。
- タスクスイッチにかかわったTSSから取ったセレクタ。
- IDTベクタ番号。

セーブされている命令ポインタ

レジスタCSとEIPにセーブされている内容は、例外を生成した命令を指す。

プログラム・ステートの変化

一般に、一般保護例外では無効な命令や動作は実行されないので、プログラム・ステートの変化は付随しない。例外ハンドラは、一般保護例外を引き起こしたすべての条件を修正した後、プログラムの連続性を損なわずにプログラムやタスクを再スタートするように設計できる。

タスクスイッチ中に一般保護例外が起こる場合、それは「新しいタスクに切り替える」ポイントの前でも後でも起こる可能性がある（6.3. 節「タスク・スイッチング」を参照）。切り替えポイントの手前で起こる場合は、プログラム・ステートの変化は生じない。切り替えポイントの後で起こる場合、プロセッサは新しいTSSからすべてのステート情報を（追加のリミット、存在、タイプチェックを一切行わずに）ロードしてから、この例外を生成することになる。一般保護例外ハンドラは、他の例外を発生させずに、CS、SS、DS、ES、FS、GSレジスタの中で見つかったセグメント・セレクタを使用できることを前提としてはならない。（この状況を処理する方法の詳細については、本章の「割り込み 10 ー無効 TSS 例外（#TS）」の「プログラム・ステートの変化」の説明を参照。）

割り込み 14 ーページフォルト例外（#PF）

例外クラス　フォルト

説明

ページングがイネーブル（レジスタ CR0 の PG フラグがセットされている）の状態でページ変換機構を使用してリニアアドレスを物理アドレスに変換している最中に、プロセッサが次の条件の 1 つを検出したことを表す。

- アドレス変換のために必要とされる、ページ・ディレクトリまたはページテーブルのエントリの P（存在）フラグがクリアされている、つまり物理メモリ内にページテーブルもオペランドの入っているページも存在しないことを表す。
- プロシージャが指示されたページにアクセスするのに十分な特権レベルを持たない（例えばユーザモードで実行中のプロシージャがスーパーバイザ・モード・ページへアクセスしようとしている場合）。
- ユーザモードで実行中のコードが読み取り専用ページに書き込もうとしている。Intel486™ プロセッサ以降のプロセッサでは、CR0 の WP フラグがセットされている場合、スーパーバイザ・モードで実行中のコードが読み取り専用のユーザ・モード・ページに書き込みしようとした場合にもページフォルトがトリガされる。
- ページ・ディレクトリ・エントリの 1 つまたは複数の予約ビットが 1 に設定されている。RSVD エラー・コード・フラグについての下記の説明を参照のこと。

例外ハンドラは、ページ不在条件から回復し、プログラムやタスクをその連続性を損なうことなく再スタートできる。ハンドラは特権違反の後にもプログラムやタスクを再スタートできるが、特権違反を引き起こした問題そのものは修正不可能なことがある。

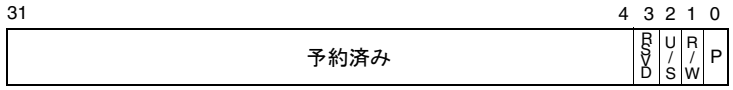
例外エラーコード

あり（特殊形式）。プロセッサは、例外の診断とそれからの回復を支援するために、ページ・フォルト・ハンドラに 2 つの項目で情報を提供する。

- スタック上のエラーコード。ページフォルトを表すエラーコードの形式は、他の例外のエラーコードとは異なる（図 5-7. を参照）。このエラーコードは例外ハンドラに次のことを知らせる。
  - P フラグにより、例外の原因が不在ページ（0）であったか、あるいはアクセス権違反か予約ビット使用（1）であったかを示す。
  - W/R フラグにより、例外の原因となったメモリアクセスが読み取り（0）、書き込み（1）のどちらであったかを示す。
  - U/S フラグにより、例外発生時点でプロセッサがユーザモード（1）かスーパーバイザ・モード（0）のどちらで実行中であったかを示す。

割り込み 14 ーページフォルト例外（#PF）（続き）

– 制御レジスタ CR4 の PSE または PAE フラグが 1 にセットされている場合、RSVD フラグにより、プロセッサがページ・ディレクトリの予約ビットの中に 1 を検出したことを示す。(PSE フラグはインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサでのみ使用可能であり、PAE フラグはインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでのみ使用可能である。以前の IA-32 プロセッサでは、RSVD フラグのビットの位置は予約されている。)



P	0	フォルトの原因が不在ページであった。
	1	フォルトの原因がページレベル保護違反であった。
W/R	0	フォルトの原因となったアクセスが読み取りであった。
	1	フォルトの原因となったアクセスが書き込みであった。
U/S	0	フォルトの原因となったアクセスはプロセッサがスーパーバイザ・モードで実行中に行われた。
	1	フォルトの原因となったアクセスはプロセッサがユーザモードで実行中に行われた。
RSVD	0	フォルトの原因は予約ビット違反ではなかった。
	1	フォルトの原因はページ・ディレクトリで予約ビットが 1 にセットされたことであった。

図 5-7. ページフォルトのエラーコード

- レジスタ CR2 の内容。プロセッサはレジスタ CR2 に、例外を生成した 32 ビット・リニア・アドレスをロードする。ページ・フォルト・ハンドラはこのアドレスを使用して、対応するページ・ディレクトリとページテーブルのエントリの位置を確認できる。ページ・フォルト・ハンドラの実行中に別のページフォルトが起こる可能性がある。ハンドラは 2 番目のページフォルトが発生する前にレジスタ CR2 の内容をセーブしなければならない。[1] ページフォルトの原因がページレベル保護違反である場合、そのフォルトが起こるとページ・ディレクトリ・エントリのアクセスフラグがセットされる。対応するページ・テーブル・エントリのアクセスフラグに関する IA-32 プロセッサの動作は、モデルに固有で、アーキテクチャでは定義されていない。

1. ページフォルトが検出された場合、プロセッサは必ず CR2 を更新する。最初のページフォルトを処理している最中に 2 番目のページフォルトが発生した場合は、2 番目のフォルトを起こしたリニアアドレスが CR2 の内容を上書きする（以前のアドレスを置き換える）。ページフォルトのためにダブルフォルトが発生した場合や、ダブルフォルトの処理中にページフォルトが発生した場合にも、これらの CR2 の更新が行われる。

割り込み 14 ーページフォルト例外（#PF）（続き）

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、通常は例外を生成した命令を指す。ページフォルト例外がタスクスイッチ中に起こった場合、レジスタ CS と EIP は新しいタスクの最初の命令を指すこともある（後の「プログラム・ステートの変化」の項を参照）。

プログラム・ステートの変化

例外を発生させる命令は結局実行されないので、通常はページフォルト例外にはプログラム・ステートの変化は付随しない。ページフォルト例外ハンドラが違反を修正し終った（例えば不在ページをメモリにロードし終った）後、プログラムまたはタスクの実行を再開できる。

タスクスイッチ中にページフォルト例外が起こる場合、プログラム・ステートは次のように変わることがある。タスクスイッチの間に、次のどの動作中でもページフォルト例外は起こり得る。

- 元のタスクのステートをそのタスクの TSS に書き込んでいるとき。
- GDT を読み取り、新しいタスクの TSS ディスクリプタの位置を確認しているとき。
- 新しいタスクの TSS を読み取っているとき。
- 新しいタスクのセグメント・セレクタに対応付けられたセグメント・ディスクリプタを読み取っているとき。
- 新しいタスクの LDT を読み取り、新しい TSS にストアされたセグメント・レジスタを確認しているとき。

後の 2 つのケースでは、例外は新しいタスクのコンテキスト内で発生する。命令ポインタは、タスクスイッチを引き起こした命令ではなく、新しいタスクの最初の命令（割り込みの場合は最後に実行される命令）を参照する。オペレーティング・システムの設計上タスクスイッチ中にページフォルトが起こり得るようになっている場合、ページ・フォルト・ハンドラはタスクゲートを通じて呼び出す必要がある。

タスクスイッチ中にページフォルトが起こる場合、プロセッサは新しい TSS からすべてのステート情報を（追加のリミット、存在、タイプチェックを一切行わずに）ロードしてから、この例外を生成する。したがって、ページ・フォルト・ハンドラは、他の例外を発生させずに、CS、SS、DS、ES、FS、GS レジスタの中で見つかったセグメント・セレクタを使用できることを前提としてはならない。（この状況を処理する方法の詳細については、本章の「割り込み 10 ー無効 TSS 例外（#TS）」の「プログラム・ステートの変化」の説明を参照。）

割り込み 14 ーページフォルト例外（#PF）（続き）

その他の例外処理情報

明示的なスタック切り替え中に発生した例外のためにプロセッサが無効なスタックポインタ（SS:ESP）を使用することのないように保証するために、特別の配慮が必要である。16 ビット IA-32 プロセッサ用に書かれたソフトウェアは、新しいスタックへ切り替わるために、多くの場合次の例のように 1 対の命令を使用する。

```
MOV SS, AX
MOV ESP, StackTop
```

このコードを 32 ビット IA-32 プロセッサの 1 つで実行している場合、セグメント・セレクタが SS レジスタにロードされてから ESP レジスタがロードされるまでの間に、ページフォルト、一般保護フォルト（#GP）、またはアライメント・チェック・フォルト（#AC）が発生することがある。この時点で、スタックポインタの 2 つの部分（SS と ESP）は一致しなくなる。つまり新しいスタック・セグメントが古いスタックポインタと一緒に使用されている。

例外ハンドラがよく定義されたスタックに切り替わる（すなわちハンドラがタスクまたはより特権の高いプロシージャである）場合、プロセッサは一致しないスタックポインタを使用しない。しかし、もし例外ハンドラが同じ特権レベルで、かつ同じタスクから呼び出された場合は、プロセッサはその一致しないスタックポインタを使用しようと試みる。

フォルトを起こしたタスクの範囲内で、（トラップまたは割り込みゲートを使用して）ページフォルト、一般保護、またはアライメント・チェック例外を処理するシステムでは、例外ハンドラと同じ特権レベルで実行されるソフトウェアは、上記の MOV 命令のペアでなく LSS 命令を使用して新しいスタックを初期化する必要がある。例外ハンドラが特権レベル 0 で実行している場合（正常なケース）、問題が起こってもそれは特権レベル 0 で実行中のプロシージャまたはタスク（通常はオペレーティング・システムのカーネル）に限定されることになる。

割り込み 16 ― x87 FPU 浮動小数点エラー（#MF）

例外クラス　フォルト

説明

x87 FPUが浮動小数点エラーを検出したことを示す。割り込み16（浮動小数点エラー例外）が生成されるためには、レジスタCR0のNEフラグがセットされていなければならない。（NEフラグの詳細については2.5.節「制御レジスタ」を参照。）

注記

SIMD浮動小数点例外（#XF）は割り込み19で通知される。

x87 FPU命令の実行中に、x87 FPUは次の6タイプの浮動小数点エラー条件を検出、報告する。

- 無効動作（#I）
  - スタック・オーバーフロー/アンダーフロー（#IS）
  - 無効算術演算（#IA）
- ゼロ除算（#Z）
- デノーマル・オペランド（#D）
- 数値オーバーフロー（#O）
- 数値アンダーフロー（#U）
- 不正確結果（精度）（#P）

これらのエラー条件はそれぞれx87 FPU例外のタイプを表し、x87 FPUはこれらの例外の各タイプについて、x87 FPUステータス・レジスタにフラグを、x87 FPU制御レジスタにマスクビットをそれぞれ1つずつ用意している。x87 FPUが浮動小数点エラーを検出したときに、その例外タイプのマスクビットがセットされている場合、x87 FPUは自動的に事前定義された（デフォルトの）応答を生成することにより例外を処理し、プログラム実行を続行する。デフォルトの応答は、大部分の浮動小数点アプリケーションが応答できる結果を提供するように設計されたものである。

その例外のマスクビットがクリアされ、レジスタCR0のNEフラグがセットされている場合、x87 FPUは次の動作を行う。

1. FPUステータス・レジスタ内の必要なフラグをセットする。
2. プログラムの命令ストリームで次に「待機中の」x87 FPU命令またはWAIT/FWAIT命令が検出されるまで待機する。

割り込み 16 ー x87 FPU 浮動小数点エラー（#MF）（続き）

3. プロセッサに浮動小数点例外（#MF）を生成させる内部エラー信号を生成する。

待機中の x87 FPU 命令または WAIT/FWAIT 命令を実行する前に、x87 FPU 命令は、保留中の x87 FPU 浮動小数点例外がないかどうかをチェックする（前記ステップ 2 で説明したとおり）。「待機中ではない」x87 FPU 命令の場合（FNINIT、FNCLEX、FNSTSW、FNSTSW AX、FNSTCW、FNSTENV、FNSAVE を含む）、保留中の x87 FPU 浮動小数点例外は無視される。ステート管理命令の FXSAVE と FXRSTOR を実行するときは、保留中の x87 FPU 例外も無視される。

どの x87 FPU 浮動小数点エラー条件も、そこから回復可能である。x87 FPU 浮動小数点エラー例外ハンドラは、x87 FPU ステータス・ワードのフラグの設定値から、例外を引き起こしたエラー条件を特定することができる。x87 FPU 浮動小数点例外の処理に関する詳細については、『IA-32 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』の付録 E.2. の「ソフトウェアによる例外処理」を参照。

例外エラーコード

なし。x87 FPU は独自のエラー情報を提供する。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、浮動小数点エラー例外が生成された時点で実行されようとしていた浮動小数点命令または WAIT/FWAIT 命令を指す。これは、エラー条件が検出された命令、つまりフォルトの原因となった命令ではない。フォルト原因の命令のアドレスは、x87 FPU 命令ポインタレジスタに入っている。浮動小数点エラー例外処理での FPU セーブの詳細については、『IA-32 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』の第 8 章の「x87 FPU 命令とオペランド（データ）ポインタ」を参照。

プログラム・ステートの変化

例外の処理は、フォルト原因の命令の次に待機中の x87 FPU 浮動小数点命令または WAIT/FWAIT 命令が検出されるまで遅延されるので、一般に x87 FPU 浮動小数点例外にはプログラム・ステートの変化が付随する。ただしエラーからの回復、および必要があればフォルト原因の命令の再実行が可能なように、x87 FPU はエラー条件に関する十分な情報をセーブしておく。

割り込み 16 ー x87 FPU 浮動小数点エラー（#MF）（続き）

非x87 FPU浮動小数点命令がx87 FPU浮動小数点命令の結果に依存するような状況では、依存する命令の前にWAITまたはFWAIT命令を挿入し、これによって依存する命令を実行する前に強制的に保留のx87 FPU浮動小数点例外が処理されるようにできる。x87浮動小数点エラー例外の同期化の詳細については、『IA-32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』の第8章の「x87 FPU例外の同期」を参照。

割り込み 17 ーアライメント・チェック例外（#AC）

例外クラス　　フォルト

説明

アライメント・チェックがイネーブルなときにプロセッサがアライメントの合っていないメモリ・オペランドを検出したことを表す。アライメント・チェックはデータ（またはスタック）セグメントでだけ行われる（コードやシステム・セグメントでは行われない）。アライメント・チェック違反の例として、ワードが奇数バイトアドレスにストアされていたり、ダブルワードが 4 の整数倍でないアドレスにストアされている場合がある。表 5-7. に、プロセッサが認識するデータタイプごとにアライメント条件を示す。

表 5-7. データタイプ別のアライメント条件

データタイプ	アドレスは下記の数字で割り切れること
ワード	2
ダブルワード	4
単精度浮動小数点（32 ビット）	4
倍精度浮動小数点（64 ビット）	8
拡張倍精度浮動小数点（80 ビット）	8
クワッドワード	8
ダブル・クワッドワード	16
セグメント・セレクタ	2
32 ビット far ポインタ	2
48 ビット far ポインタ	4
32 ビット・ポインタ	4
GDTR、IDTR、LDTR またはタスクレジスタの内容	4
FSTENV/FLDENV 命令のセーブエリア	オペランド・サイズに応じて 4 または 2
FSAVE/FRSTOR 命令のセーブエリア	オペランド・サイズに応じて 4 または 2
ビット・ストリング	オペランド・サイズ属性に応じて 2 または 4

アライメント・チェック例外（#AC）は、ワード、ダブルワード、クワッドワードの境界にアライメントを合わせる必要のあるデータタイプに対してのみ生成される。一般保護例外（#GP）は、16 バイト境界にアライメントが合っていない 128 ビット・データ・タイプに対して生成される。

割り込み 17 ーアライメント・チェック例外（#AC）（続き）

アライメント・チェックをイネーブルにするためには、次の条件が成立しなければならない。

- レジスタ CR0 の AM フラグがセットされていること。
- EFLAGS レジスタの AC フラグがセットされていること。
- CPL が 3 であること（保護モードまたは仮想 8086 モード）。

アライメント・チェック例外（#AC）は、特権レベル 3（ユーザモード）で実行中にだけ生成される。セグメント・ディスクリプタのロードなど、デフォルト時特権レベル 0 と設定されているメモリ参照は、たとえ特権レベル 3 で実行されてもアライメント・チェック例外は生成されない。

特権レベル 3 で実行中に GDTR、IDTR、LDTR またはタスクレジスタの内容をメモリにストアすると、アライメント・チェック例外が生成されることがある。通常ではアプリケーション・プログラムがこれらのレジスタをストアしないが、（ストアする場合は）ストアする情報のアライメントを奇数のワードアドレスに合わせれば、フォルトを回避できる。

FXSAVE 命令と FXRSTOR 命令は 512 バイトのデータ構造を保存および復元するが、その先頭のバイトは 16 バイト境界にアライメントを合わせる必要がある。これらの命令を実行する際にアライメント・チェック例外（#AC）がイネーブル（および CPL が 3）になっている場合は、メモリ・オペランドのアライメントが合っていないと、IA-32 プロセッサによっては、アライメント・チェック例外または一般保護例外（#GP）が発生することがある（『IA-32 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、中巻 A』の第 3 章の「FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State」および「FXRSTOR—Restore x87 FPU、MMX、SSE、and SSE2 State」を参照）。

アライメントの合っていない 128 ビット・データのロードまたはストアを行う MOVUPS 命令と MOVUPD 命令では、オペランドのアライメントが 16 バイト境界に合っていない場合、一般保護例外（#GP）を生成しない。ただし、（前述したように）アライメント・チェックがイネーブルの場合は、2、4、8 バイトのミスアライメントが検出され、アライメント・チェック例外が生成される。

FSAVE と FRSTOR 命令がアライメントの合っていないメモリ参照を生成し、それがアライメント・チェック・フォルトを引き起こすことがある。ただしアプリケーション・プログラムでは、これらの命令を必要とすることはほとんどない。

例外エラーコード

あり（常に 0）。

割り込み 17 ーアライメント・チェック例外（#AC）（続き）

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、例外を生成した命令を指す。

プログラム・ステートの変化

命令は結局実行されないので、アライメント・チェック・フォルトにはプログラム・ステートの変化は付随しない。

割り込み 18 ーマシンチェック例外（#MC）

例外クラス　アボート

説明

プロセッサが内部マシンエラーまたはバスエラーを検出したか、外部エージェントがバスエラーを検出したことを示す。マシンチェック例外はモデル固有の例外で、インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサにだけ用意されている。この例外はインテル Pentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサ、インテル Pentium プロセッサでは異なり、将来の IA-32 プロセッサとは互換性がない可能性がある。（この機能が存在するかどうかを判断するにはCPUID命令を使用する。）

外部エージェントによってバスエラーが検出されると、プロセッサの専用のピンに信号が送信される。このピンは、インテル Pentium 4プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは BINIT# ピンと MCERR# ピンで、インテル Pentium プロセッサではBUSCHK# ピンである。これらのピンのいずれかがイネーブルである場合にピンをアサートすると、エラー情報がマシン・チェック・レジスタにロードされ、マシンチェック例外が生成される。

マシンチェック例外とマシン・チェック・アーキテクチャについては、第14章「マシン・チェック・アーキテクチャ」で詳細に説明する。プロセッサ固有のハードウェア情報については、個々のプロセッサのデータブックを参照。

例外エラーコード

なし。エラー情報はマシンチェック MSR で提供される。

セーブされている命令ポインタ

インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの場合、拡張マシン・チェック・ステート・レジスタにセーブされている内容は、マシンチェック例外を発生させたエラーに直接に対応したものとなる（14.3.1.3. 項「IA32_MCG_STATUS MSR」および14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照）。

P6ファミリ・プロセッサでは、レジスタ MCG_STATUS MSR の EIPV フラグがセットされている場合、レジスタ CS と EIP にセーブされている内容は、マシンチェック例外を発生させたエラーに直接に対応したものである。フラグがクリアされている場合、セーブされている命令ポインタがエラーに対応しないことがある。（14.3.1.3. 項「IA32_MCG_STATUS MSR」を参照。）

インテル Pentium プロセッサでは、レジスタ CS と EIP の内容がエラーに対応しない場合がある。

割り込み 18 ーマシンチェック例外（#MC）（続き）

プログラム・ステートの変化

マシン・チェック・メカニズムは、制御レジスタ CR4 の MCE フラグをセットするとイネーブルにされる。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサの場合、プログラム・ステートが変化すると必ずマシンチェック例外が発生し、アボートクラス例外が生成される。アボート例外が起きた場合、この例外に関する情報をマシンチェック MSR から収集できるが、プログラムを再スタートすることは通常はできない。

マシンチェック機構がイネーブルでない（制御レジスタ CR4 の MCE フラグがクリアされている）場合、マシンチェック例外が起こるとプロセッサはシャットダウン・ステートに入る。

割り込み 19 ― SIMD 浮動小数点例外（#XF）

例外クラス　フォルト

説明

プロセッサがSSE、SSE2、またはSSE3 SIMD浮動小数点例外を検出したことを示す。この割り込みが生成されるためには、MXCSR レジスタの適切なステータス・フラグがセットされ、特定の例外がアンマスクされていなければならない。

SSE、SSE2、またはSSE3 SIMD浮動小数点命令の実行時に発生する数値例外条件には、次の6つのクラスがある。

- 無効操作（#I）
- ゼロ除算（#Z）
- デノーマル・オペランド（#D）
- 数値オーバーフロー（#O）
- 数値アンダーフロー（#U）
- 不正確結果（精度）（#P）

無効操作例外、ゼロ除算例外、デノーマル・オペランド例外は、計算前型の例外であり、算術演算が実行される前に検出される。数値アンダーフロー例外、数値オーバーフロー例外、不正確結果例外は、計算後型の例外である。

SIMD浮動小数点例外クラスの詳細については、『IA-32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』の第11章の「SIMD浮動小数点例外」を参照のこと。

SIMD浮動小数点例外が発生すると、プロセッサは、次のいずれかの処置をとる。

- プロセッサは自動的に例外を処理する。この場合は、最も妥当な結果を求め、プログラムの実行をそのまま続ける。これは、マスクされた例外に対する応答である。
- SIMD浮動小数点例外を生成する。この例外により、ソフトウェア例外ハンドラが起動される。これは、マスクされていない例外に対する応答である。

割り込み 19 ー SIMD 浮動小数点例外（#XF）（続き）

MXCSR レジスタには、上記の 6 つの SIMD 浮動小数点例外条件のそれぞれに対応するフラグビットとマスクビットが入っている。例外がマスクされている場合（MXCSR レジスタの対応するマスクビットがセットされている）は、プロセッサは自動的に適切なデフォルトの処置を実行し、計算を続行する。例外がマスクされてなく（対応するマスクビットがクリアされている）、オペレーティング・システムが SIMD 浮動小数点例外をサポートしている場合（制御レジスタ CR4 の OSXMMEXCPT フラグがセットされている）は、SIMD 浮動小数点例外により、ソフトウェア例外ハンドラが起動される。例外がマスクされてなく、OSXMMEXCPT ビットがクリアされている（オペレーティング・システムがマスクされていない SIMD 浮動小数点例外をサポートしていないことを示す）場合は、SIMD 浮動小数点例外ではなく、無効オペコード例外（#UD）が通知される。

SIMD 浮動小数点例外は正確であり、直ちに処理される。このため、x87 浮動小数点命令、WAIT/FWAIT 命令、または他の SSE、SSE2、または SSE3 が、保留中のマスクされていない SIMD 浮動小数点例外を検出する状況は起こらない。

SIMD 浮動小数点例外がマスクされている（対応する例外フラグがセットされている）ときに SIMD 浮動小数点例外が発生し、その後で SIMD 浮動小数点例外がアンマスクされた場合は、例外がアンマスクされたときに例外は生成されない。

SSE、SSE2、SSE3 の SIMD 浮動小数点命令が（2 つまたは 4 つのサブオペランドからなる）パックド・オペランドを操作する場合、複数の SIMD 浮動小数点例外条件が検出されることがある。1 つまたは複数のサブオペランド・セットに対して 1 つの例外条件が検出される場合、検出される例外条件ごとに例外フラグがセットされる。例えば、1 つのサブオペランドについて検出された無効例外によって、別のサブオペランドについてのゼロ除算例外の報告は妨げられない。ただし、1 つのサブオペランドについて 2 つ以上の例外条件が生成された場合は、表 5-8. の優先度にしたがって 1 つだけ例外条件が報告される。この例外の優先度では優先度の高い例外条件だけが報告され、優先度の低い例外条件は無視されるときがある。

割り込み 19 — SIMD 浮動小数点例外（#XF）（続き）

表 5-8. SIMD 浮動小数点例外の優先順位

優先順位	説明
1（最も高い）	SNaN オペランド（max 操作、min 操作、または特定の比較 / 変換操作では任意の NaN オペランド）による無効操作例外
2	QNaN オペランド[1]
3	上記以外の任意の無効操作例外またはゼロ除算例外[2]
4	デノーマル・オペランド例外[2]
5	数値オーバーフロー / アンダーフロー例外（不正確結果例外と組み合わされている可能性あり）[2]
6（最も低い）	不正確結果例外

注記：

1. QNaN は例外ではないが、QNaN オペランドの処理は、より優先順位の低い例外に優先する。例えば、QNaN をゼロで除算した結果は、ゼロ除算例外ではなく、QNaN になる。
2. 例外がマスクされている場合は、命令の実行が続行されるため、より優先順位の低い例外が発生することがある。

例外エラーコード

なし。

セーブされている命令ポインタ

CS レジスタと EIP レジスタにセーブされている内容は、SIMD 浮動小数点例外の発生時に実行されていた SSE、SSE2、または SSE3 を指す。これはフォルトの原因になった命令であり、この命令でエラー状態が検出された。

プログラム・ステートの変化

特定の例外がマスクされている場合を除いて、SIMD 浮動小数点例外は直ちに処理される。このため、SIMD 浮動小数点例外が発生すると、通常はプログラムのステートが変化する。利用可能なステート情報が十分にあるので、必要に応じてエラーから回復し、フォルトの原因になった命令を再実行できる。

割り込み 32 ～ 255 ーユーザ定義の割り込み

例外クラス　　該当せず。

説明

プロセッサが次のどれかを行ったことを表す。

- 32 ～ 255 の範囲のベクタ番号のどれかを命令オペランドとする INT *n* 命令を実行した。
- 要求に対応付けられた割り込みベクタ番号が32～255のいずれかである場合、INTR ピンまたはローカル APIC からの割り込み要求に応答した。

例外エラーコード

該当せず。

セーブされている命令ポインタ

レジスタ CS と EIP にセーブされている内容は、INT *n* 命令の後の命令、または INTR 信号が発生したときの命令の後の命令を指す。

プログラム・ステートの変化

INT *n* 命令または INTR 信号によって生成された割り込みには、プログラム・ステートの変化は付随しない。INT *n* 命令は命令ストリームの内部で割り込みを生成する。プロセッサは INTR 信号を受け取ると、それ以前のすべての命令についてのすべてのステート変化を切り替えてから、割り込みに応答する。したがって割り込みハンドラから戻った時点でプログラム実行を再開できる。

6

タスク管理

第 6 章
タスク管理

6

本章ではIA-32アーキテクチャのタスク管理機能について説明する。これらの機能は、プロセッサが保護モードで実行しているときにだけ使用できる。

6.1. タスク管理の概要

タスクとは、プロセッサがディスパッチ、実行、保留できる1作業単位である。このタスクを使用してプログラム、タスク/プロセス、オペレーティング・システム・サービス・ユーティリティ、割り込み/例外ハンドラ、またはカーネル/エグゼクティブ・ユーティリティを実行できる。

IA-32アーキテクチャは、タスクのステートをセーブしたり、タスクを実行用にディスパッチしたり、あるタスクから別のタスクへ切り替えたりするための機構を備えている。保護モードでの操作中、プロセッサの実行はすべてタスク内部で起こる。単純なシステムでも最低1つはタスクを定義しなければならない。もっと複雑なシステムでは、プロセッサのタスク管理機能を使用してマルチタスク・アプリケーションをサポートできる。

6.1.1. タスクの構造

1つのタスクは、タスク実行空間とタスク・ステート・セグメント（TSS: Task-state Segment）という2つの部分で構成される。タスク実行空間は、コード・セグメント、スタック・セグメント、および1つまたは複数のデータ・セグメントから成る（図6-1.を参照）。オペレーティング・システムやエグゼクティブがプロセッサの特権レベル保護機構を使用する場合、タスク実行空間にも各特権レベルについて別々のスタックが用意される。

TSSは、タスク実行空間を構成するセグメントを指定し、タスクステート情報の記憶場所を提供する。マルチタスキング・システムでは、TSSはタスクをリンクするための機構も提供する。

注記

本章では、主に32ビットのタスクと32ビットのTSSの構造について説明する。16ビットのタスクと16ビットのTSSの構造については、6.6.節「16ビットのタスク・ステート・セグメント（TSS）」を参照。

タスクはそのTSSのセグメント・セレクタによって識別される。タスクが実行のためにプロセッサにロードされると、そのTSSのセグメント・セレクタ、ベースアドレス、リミットおよびセグメント・ディスクリプタ属性がタスクレジスタにロードされる（2.4.4.項「タスクレジスタ（TR）」を参照）。

ページングがタスク用に使用されている場合、タスクが使用するページ・ディレクトリのベースアドレスが、制御レジスタCR3にロードされる。

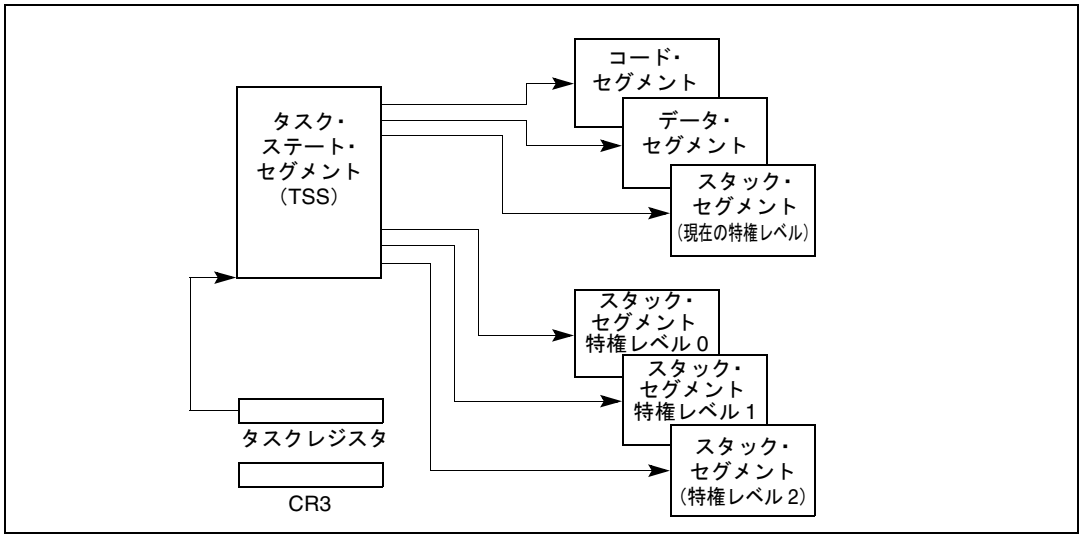


図 6-1. タスクの構造

6.1.2. タスクのステート

次の項目によって、現在実行中のタスクのステートが定義される。

- タスクの現行実行空間。セグメント・レジスタ（CS、DS、SS、ES、FS、GS）の中にあるセグメント・セレクタによって定義される。
- 汎用レジスタのステート
- EFLAGSレジスタのステート
- EIPレジスタのステート
- 制御レジスタCR3のステート

- タスクレジスタのステート
- LDTR レジスタのステート
- I/O マップ・ベース・アドレスおよびI/O マップ（TSS に入っている）
- 特権レベル0、1、2 のスタックに対するスタックポインタ（TSS に入っている）
- 前回実行されたタスクへのリンク（TSS に入っている）

タスクのディスパッチを行う前は、上記の項目のうちタスクレジスタのステートを除くすべての項目が、そのタスクの TSS に入っている。また、LDTR レジスタについては、その全内容ではなく、LDT のセグメント・セレクタだけが TSS に入っている。

6.1.3. タスクの実行

ソフトウェアやプロセッサは、次の方法のいずれか 1 つを使用してタスクを実行用にディスパッチできる。

- CALL 命令によるタスクへの明示的呼び出し
- JMP 命令によるタスクへの明示的ジャンプ
- 割り込みハンドラタスクへの暗黙の呼び出し（プロセッサが行う）
- 例外ハンドラタスクへの暗黙の呼び出し
- EFLAGS レジスタの NT フラグがセットされている場合のリターン（IRET 命令によって行われる）

これらのタスク・ディスパッチ法ではすべて、タスクのタスクゲートか TSS のどちらかを指すセグメント・セレクタによって、ディスパッチするタスクを特定する。CALL または JMP 命令によってタスクをディスパッチする場合は、命令内のセレクタが直接 TSS を選択するか、TSS のセレクタを保持するタスクゲートを選択する。割り込みや例外を処理するためにタスクをディスパッチする場合は、割り込みや例外の IDT エントリに、割り込み / 例外ハンドラの TSS のセレクタを保持するタスクゲートが含まれていなければならない。

あるタスクがディスパッチされると、現在実行中のタスクからディスパッチされたタスクへ、自動的にタスクスイッチが起こる。タスクスイッチ中に、現在実行中のタスクの実行環境（そのタスクのステートまたは**コンテキスト**と呼ばれる）がそのタスクの TSS にセーブされ、タスクの実行は保留状態になる。その後ディスパッチされたタスクのコンテキストがプロセッサにロードされ、新しくロードされた EIP レジスタが指す命令によって、そのタスクの実行が開始される。システムが最後に初期化された後、タスクが実行されていない場合は、EIP はタスクのコードの最初の命令を指す。それ以外の場合は、タスクが最後にアクティブであったときに実行していた命令の次の命令を指す。

現在実行中のタスク（呼び出しタスク）がディスパッチされるタスク（呼び出されるタスク）を呼び出した場合、呼び出しタスクのTSSセグメント・セレクタが呼び出されたタスクのTSSにストアされ、呼び出しタスクへのリンクバックができるようになっている。

IA-32プロセッサの場合、タスクはリカーシブでない。タスクはそれ自体に対しては呼び出しやジャンプができない。

割り込みや例外を、ハンドラタスクへのタスクスイッチを使用して処理できる。この場合、プロセッサはタスクスイッチを行って割り込みや例外を処理できるだけでなく、割り込み / 例外ハンドラタスクから戻る時点で、自動的に割り込まれたタスクにスイッチバックもできる。この機構では、割り込みタスク中に発生する割り込みを処理できる。

タスクスイッチの一部として、プロセッサは別のLDTへ切り替わり、各タスクがLDTベースのセグメントに対し別々の論理 - 物理アドレス・マッピングを持つようにできる。タスクスイッチ時にはページ・ディレクトリ・ベース・レジスタ（CR3）も再ロードされ、各タスクが専用のページ・テーブル・セットを持てるようにもできる。このような保護機能は、孤立したタスクを助け、それらが互いに干渉しあうのを防ぐ。これらの保護機構の一方または両方を使用していない場合は、プロセッサによるタスク間の保護は行われない。これは、複数の特権レベルを使用して保護を行うオペレーティング・システムについてもあてはまる。この場合、特権レベル3で実行されるタスクが、他の特権レベル3のタスクと同じLDTおよびページテーブルを使用し、他のタスクのコードにアクセスして、そのデータやスタックを破壊することがある。

オプションで、タスク管理機能をマルチタスク・アプリケーションの処理に使用できる。ソフトウェアでも、ソフトウェア定義の各タスクがIA-32プロセッサの1つのタスクのコンテキスト内で実行される形で、マルチタスキングを処理できる。

6.2. タスク管理用データ構造

プロセッサはタスク関連アクティビティを処理するために次の 5 つのデータ構造を定義する。

- タスク・ステート・セグメント（TSS）
- タスク・ゲート・ディスクリプタ
- TSS ディスクリプタ
- タスクレジスタ
- EFLAGS レジスタ内の NT フラグ

保護モードで操作しているとき、最低 1 つのタスクについて TSS と TSS ディスクリプタを作成し、その TSS のセグメント・セレクタを（LTR 命令を使用して）タスクレジスタにロードしなければならない。

6.2.1. タスク・ステート・セグメント（TSS）

タスクをリストアするために必要となるプロセッサのステート情報は、タスク・ステート・セグメント（TSS）と呼ばれるシステム・セグメントにセーブされる。図 6-2. に、32 ビット CPU 用に設計されたタスクの TSS のフォーマットを示す。（16 ビットのインテル® 286 プロセッサ用のタスクとの互換性は別のタイプの TSS により提供される。図 6-9. を参照。）TSS のフィールドは、動的フィールドと静的フィールドという 2 つの主カテゴリに分類される。

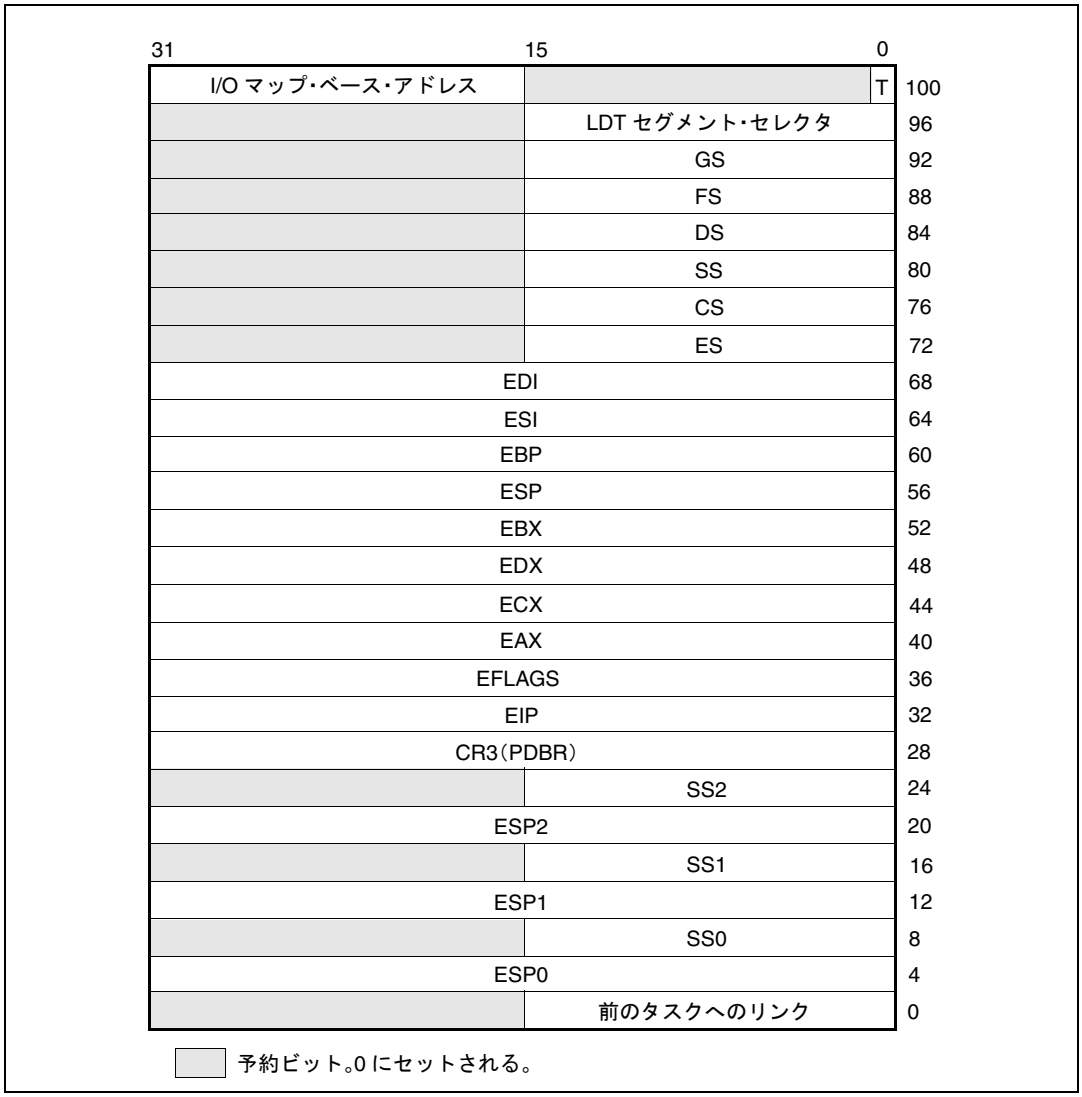


図 6-2. 32 ビット・タスク・ステート・セグメント（TSS）

プロセッサは、タスクスイッチ中タスクが保留状態のときに動的フィールドを更新する。動的フィールドは次のとおりである。

汎用レジスタ・フィールド

タスクスイッチ前の EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI レジスタのステート。

セグメント・セレクタ・フィールド

タスクスイッチ前の ES、CS、SS、DS、FS、GS レジスタにストアされたセグメント・セレクタ。

EFLAGS レジスタ・フィールド
: タスクスイッチ前の EFLAGS レジスタのステート。

EIP（命令ポインタ）フィールド
: タスクスイッチ前の EIP レジスタのステート。

前のタスクへのリンク・フィールド
: 前のタスクの TSS に対応するセグメント・セレクタが入る（呼び出し、割り込み、または例外によるタスクスイッチ時に更新される）。このフィールド（バック・リンク・フィールドとも呼ばれる）により、IRET 命令で前のタスクへのタスクスイッチが可能になる。

プロセッサは、静的フィールドを読み取るが通常は変更しない。静的フィールドはタスク作成時にセットアップされる。静的フィールドは次のとおりである。

LDT セグメント・セレクタ・フィールド
: タスクの LDT に対応するセグメント・レジスタが入る。

CR3 制御レジスタ・フィールド
: タスクが使用する予定のページ・ディレクトリのベース物理アドレスが入る。制御レジスタ CR3 もページ・ディレクトリ・ベース・レジスタ（PDBR）である。

特権レベル 0、1 および 2 のスタック・ポインタ・フィールド
: これらのスタックポインタはそれぞれ、スタック・セグメントのセグメント・セレクタ（SS0、SS1、SS2）とスタック内のオフセット（ESP0、ESP1、ESP2）で構成される 1 つの論理アドレスから成る。これらのフィールドの値は特定のタスクに対して静的である。一方、SS および ESP の値は、タスク内でスタック切り替えが発生すると変更される。

T（デバッグトラップ）フラグ（バイト 100、ビット 0）
: T フラグがセットされている場合、このタスクへのタスクスイッチが起こるとプロセッサはデバッグ例外を発生させる（15.3.1.5. 項「タスクスイッチ例外条件」を参照）。

I/O マップ・ベース・アドレス・フィールド
: TSS のベースから I/O 許可ビットマップまでと割り込みリダイレクション・ビット・マップまでの 16 ビット・オフセットが入る。これらのマップは存在するとすれば TSS 内の相対的に上位のアドレスにストアされている。I/O マップ・ベース・アドレスは、I/O 許可ビットマップの初めと割り込みリダイレクション・ビット・マップの終りを指す。I/O 許可ビットマップの詳細については、『IA-32 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』の第 13 章「入出力」を参照。また、割り込みリダイレクション・ビット・マップの詳細については 16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照。

ページングを使用する場合は、タスクスイッチ中にプロセッサが読み取る TSS 部分（先頭の 104 バイト）の内部にページ境界が来ないように注意する必要がある。TSS のこの部分の内部にページ境界が来る場合、境界のいずれの側にもページが同時に存在し、物理アドレスで連続していなければならない。

この制限の理由は、タスクスイッチ中に TSS にアクセスするとき、プロセッサは、TSS の最初のバイトの物理アドレスから始まる、連続する物理アドレスから各 TSS の最初の 104 バイトを読み書きするからである。この領域にページ境界がある場合、ページ境界でアドレス変換は行われない。したがって、TSS へのアクセスが開始された後、104 バイトの一部が存在せず、物理的に連続していない場合、プロセッサはページフォルト例外を発生させずに、不正な TSS 情報にアクセスする。この不正な情報の読み取りが行われると、一般的に、後のタスクスイッチ時に回復不能な例外が発生する。

また、ページングを使用する場合、前のタスクの TSS、現行タスクの TSS にそれぞれ対応するページ、およびそれぞれに対応するディスクリプタ・テーブル・エントリが、読み取り / 書き込み可能とマークされていなければならない。タスクスイッチが開始される前に、このような構造を含むページがメモリ内にも存在している場合、高速にタスクスイッチを実行できる。

6.2.2. TSS ディスクリプタ

すべての他のセグメントと同様、TSS もセグメント・ディスクリプタによって定義される。図 6-3. に TSS ディスクリプタのフォーマットを示す。TSS ディスクリプタは GDT にだけ入れることができ、LDT や IDT には入れられない。

TI フラグがセットされている状態（現行 LDT が示す）でセグメント・セレクタを使用して TSS にアクセスを試みると、CALL 命令および JMP 命令の間に一般保護例外（#GP）が生成され、IRET 命令の間に無効 TSS 例外（#TS）が発生する。ある TSS のセグメント・セレクタをセグメント・レジスタにロードしようとした場合も、一般保護例外が生成される。

タイプ・フィールドのビジー（B）フラグは、タスクがビジーであるかどうかを示す。ビジーなタスクとは、現在実行中、または実行を待機中のタスクのことである。タイプ・フィールドの値が 1001B の場合はタスクが非アクティブであることを示し、フィールドの値が 1011B の場合はタスクがビジーであることを示す。タスクはリカーシブではない。プロセッサはビジーフラグを使用して、実行中に割り込まれたタスクを呼び出そうとする試みを検出する。1 つのタスクに対応するビジーフラグは必ず 1 つであるようにするため、各 TSS を指す TSS ディスクリプタも 1 つにしなければならない。

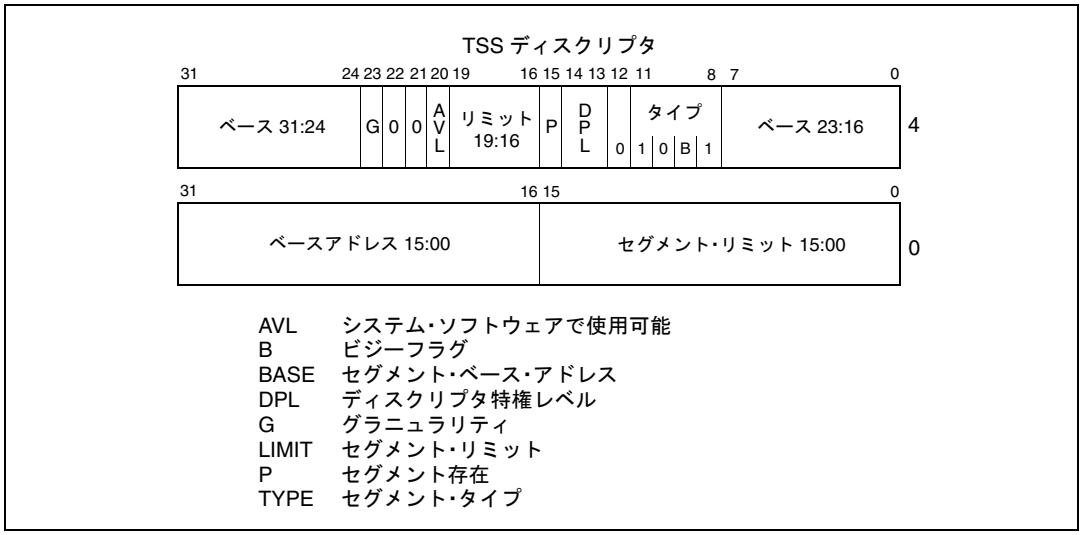


図 6-3. TSS ディスクリプタ

ベース、リミット、DPL フィールド、およびグラニュラリティと存在フラグの機能は、データ・セグメント・ディスクリプタの中で使用される場合（3.4.3.項「セグメント・ディスクリプタ」を参照）とよく似ている。32 ビット TSS 用の TSS ディスクリプタの中で G フラグが 0 のとき、リミット・フィールドの値は 67H（TSS の最小サイズより 1 バイトだけ小さい）以上でなければならない。TSS ディスクリプタのリミットが 67H より小さいようなタスクに切り替えようと試みると、無効 TSS 例外（#TS）が生成される。TSS の中に I/O 許可ビットマップが存在する場合はそれより大きいリミットが必要になる。また、オペレーティング・システムが TSS 内にさらに追加のデータをストアする場合、それよりさらに大きいリミットが必要になる。プロセッサは、タスクスイッチに際してはリミットが 67H を越えるかどうかのチェックは行わないが、I/O 許可ビットマップまたは割り込みリダイレクション・ビット・マップにアクセスする際にはこのチェックを行う。

TSS ディスクリプタへアクセスするプログラムやプロシージャ（つまりその CPL が TSS ディスクリプタの DPL 以下であるもの）はいずれも、呼び出しまたはジャンプでタスクをディスパッチできる。

大部分のシステムでは、TSS ディスクリプタの DPL を 2 以下の値に設定し、特権ソフトウェアだけがタスクスイッチを実行できるようになっているはずである。ただしマルチタスク・アプリケーションでは、TSS ディスクリプタによっては DPL を 3 に設定できるものがあり、この場合はアプリケーション（またはユーザ）特権レベルでタスクスイッチを行える。

6.2.3. タスクレジスタ

タスクレジスタは、現行タスクのTSSに対応する16ビット・セグメント・セレクタおよびセグメント・ディスクリプタ全体（32ビット・ベース・アドレス、16ビット・セグメント・リミットおよびディスクリプタ属性）をストアしている（図2-4.を参照）。このレジスタはGDT内のTSSディスクリプタを参照する。図6-4.に、プロセッサがタスクレジスタ内の情報を用いてTSSにアクセスする場合の経路を示す。

タスクレジスタには見える部分（ソフトウェアが読み取ったり変更できる）と見えない部分（プロセッサが保持しており、ソフトウェアからはアクセス不可能）の両方がある。見える部分にあるセグメント・セレクタは、GDT内のTSSディスクリプタを指す。プロセッサはタスクレジスタの見えない部分を使用して、TSSのセグメント・ディスクリプタをキャッシングする。これらの値を1つのレジスタ内にキャッシングしておくと、プロセッサが現行タスクのTSSを参照する場合これらの値をメモリからフェッチする必要がなくなるので、タスクの実行がより効率的になる。

LTR（タスクレジスタのロード）とSTR（タスクレジスタのストア）命令は、タスクレジスタの見える部分のロード、読み取りを行う。LTR命令はセグメント・セレクタ（ソース・オペランド）を、GDTのTSSディスクリプタを指すタスクレジスタ内へロードし、ついでタスクレジスタの見えない部分に、TSSディスクリプタから情報をロードする。この命令は特権命令、すなわちCPLが0のときにだけ実行できる命令である。LTR命令は一般に、システム初期化時に初期値をタスクレジスタに入れるために使用する。その後、タスクが切り替わると、タスクレジスタの内容は暗黙的に変更される。

STR（タスクレジスタのストア）命令は、タスクレジスタの見える部分を汎用レジスタまたはメモリにストアする。この命令は、任意の特権レベルで実行されるコードによって、現在実行中のタスクを識別するために実行することができる。しかし、通常、この命令を使用するのはオペレーティング・システム・ソフトウェアだけである。

プロセッサの電源投入またはリセットの時点で、セグメント・セレクタとベースアドレスがデフォルト値0に、リミットはFFFFHに設定される。

6.2.4. タスク・ゲート・ディスクリプタ

タスク・ゲート・ディスクリプタは、タスクへの間接的な保護された参照を提供する。図6-5.にタスク・ゲート・ディスクリプタのフォーマットを示す。タスク・ゲート・ディスクリプタはGDT、LDTまたはIDTに入れられる。

タスク・ゲート・ディスクリプタのTSSセグメント・セレクタ・フィールドは、GDT内のTSSディスクリプタを指す。このセグメント・セレクタのRPLは使用されない。

タスク・ゲート・ディスクリプタのDPLは、タスクスイッチ中にTSSディスクリプタへのアクセスを制御する。プログラムまたはプロシージャがタスクゲートを介してタ

スクへの呼び出しまたはジャンプを行うとき、タスクゲートを指しているゲートセレクタのCPLとRPLはタスク・ゲート・ディスクリプタのDPL以下でなければならない。(タスクゲートを使用する場合はデスティネーションのTSSディスクリプタのDPLを使用しないことに注意。)

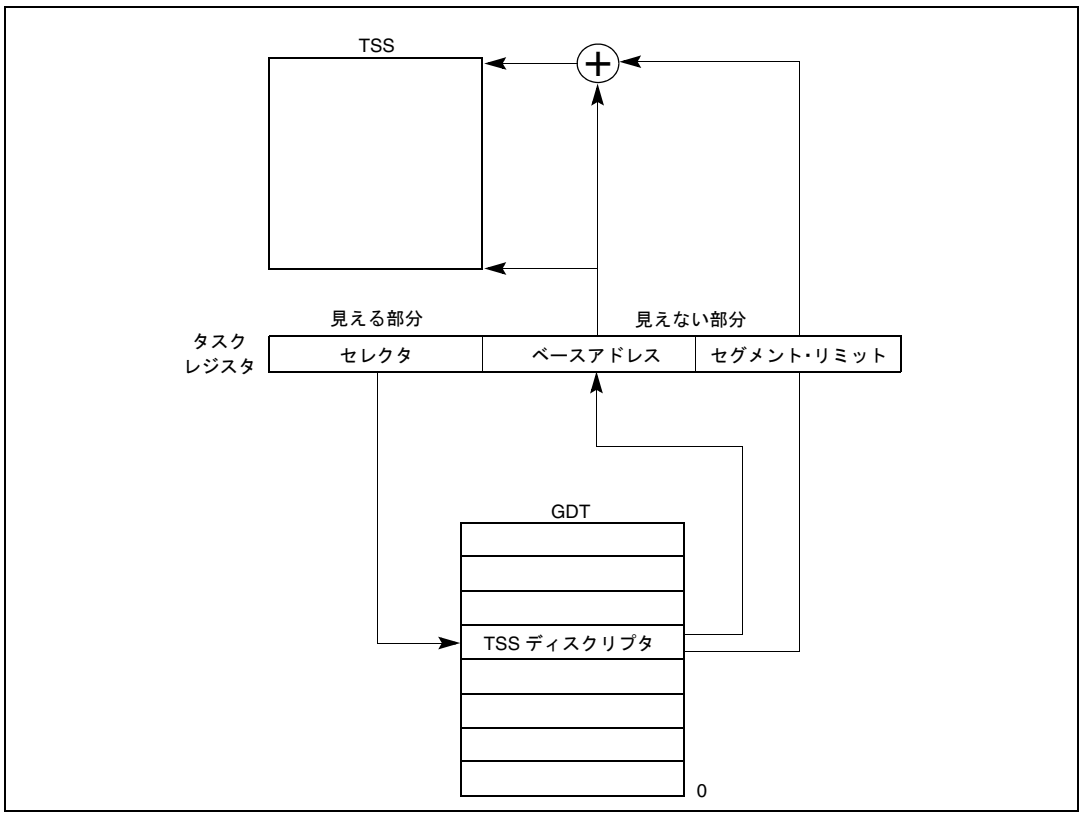


図 6-4. タスクレジスタ

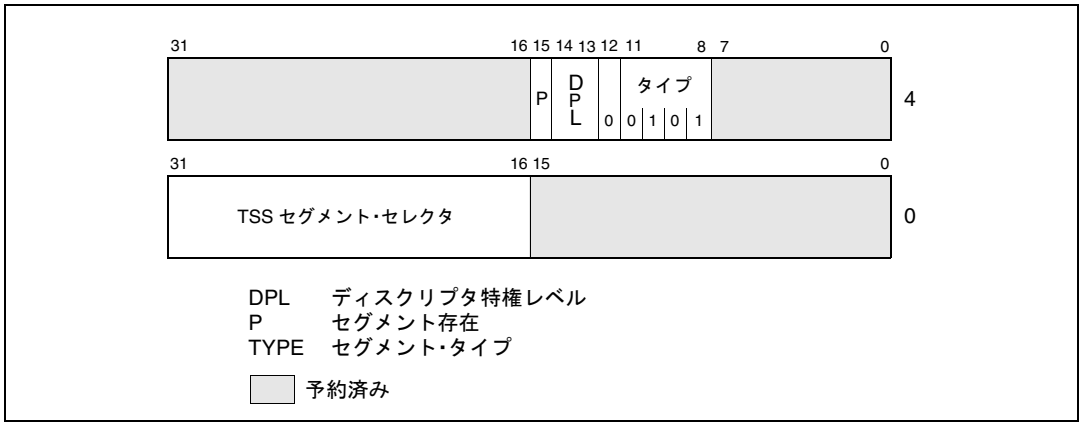


図 6-5. タスク・ゲート・ディスクリプタ

タスクには、タスク・ゲート・ディスクリプタかTSSディスクリプタのどちらかを介してアクセスできる。これらのデータ構造は両方とも、次の必要性を満たすために用意されたものである。

- タスクがビジーフラグを１つだけ持つ必要性。タスクのビジーフラグは TSS ディスクリプタ内にストアされているので、各タスクはTSSディスクリプタを１つだけ持つはずである。しかし複数のタスクゲートが同一の TSS ディスクリプタを参照することもある。
- タスクへの選別的アクセスを提供する必要性。タスクゲートは LDT に常駐でき、TSSディスクリプタのDPLと異なるDPLを持てるので、この必要性を満たす。GDT内のあるタスクに対応するTSSディスクリプタ（通常はDPL0）にアクセスできるだけの特権を持たないプログラムやプロシージャは、それより高い DPL を持つタスクゲートを介してタスクへのアクセスを許されることもある。タスクゲートを使用すると、オペレーティング・システムは各タスクへのアクセス制限を緩和できる。
- 割り込み / 例外を独立のタスクで処理できるようにする必要性。タスクゲートはIDTにも常駐でき、この場合は例外/割り込みをハンドラタスクによって処理できる。割り込み/例外ベクタがタスクゲートを指している場合、プロセッサは指定されたタスクへ切り替わる。

図6-6.に、どのようにすればLDT内のタスクゲートとGDT内のタスクゲートとIDT内のタスクゲートがすべて同じタスクを指せるかを示す。

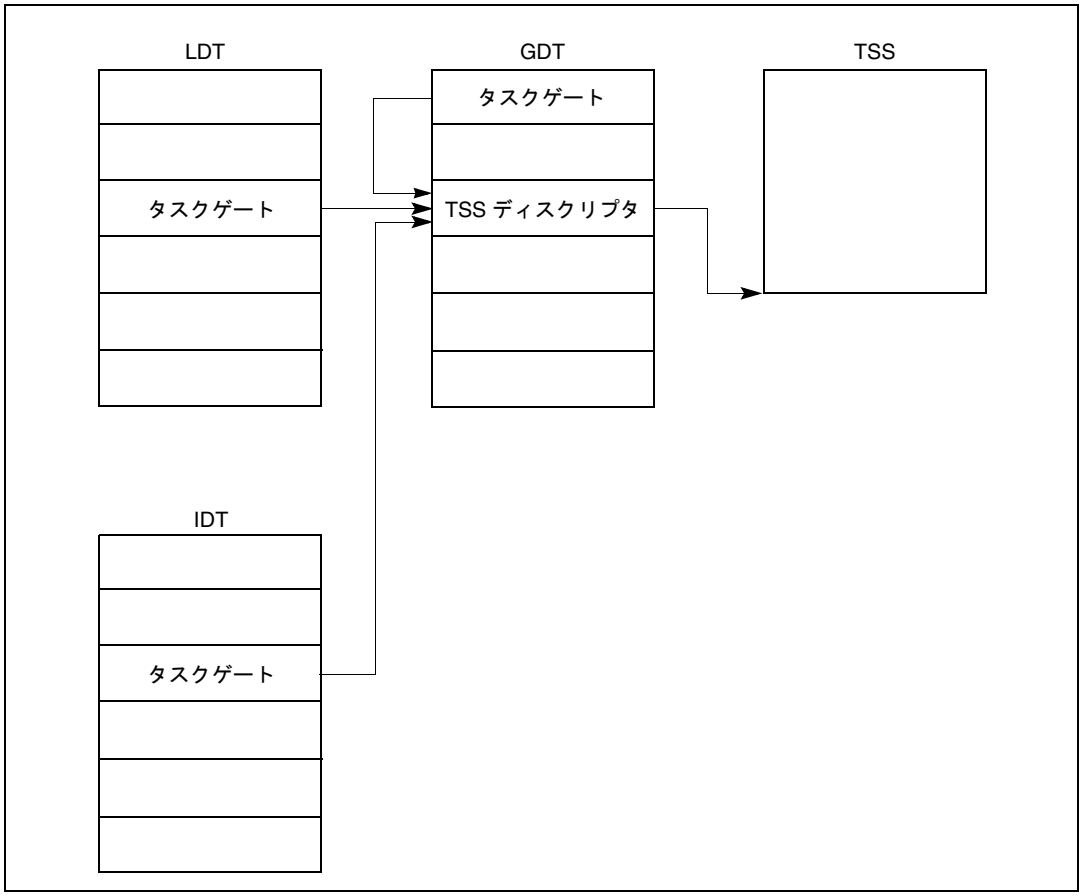


図 6-6. 複数のタスクゲートが同じタスクを参照する場合

6.3. タスク・スイッチング

プロセッサは次の4つのどれかが起こった場合に他のタスクへ実行を転送する。

- 現行プログラム、タスクまたはプロシージャが、GDT 内にある TSS ディスクリプタに対して JMP または CALL 命令を実行した。
- 現行プログラム、タスクまたはプロシージャが、GDT または現行 LDT 内にあるタスク・ゲート・ディスクリプタに対して JMP または CALL 命令を実行した。
- 割り込み/例外ベクタが IDT 内にあるタスク・ゲート・ディスクリプタを指している。
- EFLAGS レジスタの NT フラグがセットされているときに現行タスクが IRET を実行した。

JMP、CALL、IRET 命令は割り込みや例外と同様、すべてプログラムをリダイレクトするための汎用機構である。TSS ディスクリプタまたはタスクゲートの参照（タスクを呼び出したりそこへジャンプする場合）、あるいは NT フラグのステート（IRET 命令を実行する場合）によって、タスクスイッチが起こるかどうかが判断される。

プロセッサは新しいタスクへ切り替わるとき、次の動作を行う。

1. タスクゲートまたは前のタスクへのリンク・フィールド（IRET 命令でタスクスイッチを行う場合）から、JMP または CALL 命令のオペランドとして新しいタスクの TSS セグメント・セレクタを取得する。
2. 現行（古い）タスクが新しいタスクへの切り替えを許されているかどうかチェックする。JMP、CALL 命令にはデータアクセス特権規則が適用される。現行（古い）タスクの CPL と新しいタスクのセグメント・セレクタの RPL が、参照される TSS ディスクリプタまたはタスクゲートの DPL 以下でなければならない。例外、割り込み（INT *n* 命令によって生成される割り込みを除く）および IRET 命令は、デスティネーションのタスクゲートや TSS ディスクリプタの DPL に関係なく、タスクの切り替えを許される。INT *n* 命令によって生成された割り込みについては、DPL がチェックされる。
3. 新しいタスクの TSS ディスクリプタが存在とマークされており、かつそのリミットが有効（67H 以上）であるかをチェックする。新しいタスクが使用可能であるか（呼び出し、ジャンプ、例外、割り込みの場合）、ビジーであるか（IRET の戻りの場合）をチェックする。
4. 新しいタスクが使用可能であるか（呼び出し、ジャンプ、例外、割り込みの場合）、ビジーであるか（IRET の戻りの場合）をチェックする。
5. 現行（古い）TSS、新しい　TSS、タスクスイッチに使用されたすべてのセグメント・ディスクリプタがシステムメモリにページングされているかチェックする。
6. タスクスイッチが JMP または IRET 命令によるものである場合、プロセッサは現行（古い）タスクの TSS ディスクリプタのビジー（B）フラグをクリアする。CALL 命令、例外、または割り込みによるものである場合は、ビジー（B）フラグはセットされたままである（表 6-2. を参照）。
7. タスクスイッチが IRET 命令によるものである場合、プロセッサは EFLAGS レジスタの一時的にセーブされているイメージ内の NT フラグをクリアする。CALL、JMP 命令、例外、または割り込みによるものである場合は、セーブされた EFLAGS イメージ内の NT フラグは変更されない。
8. 現行タスクの TSS に現行（古い）タスクのステートをセーブする。プロセッサはタスクレジスタの中で現行 TSS のベースアドレスを見つけた後、現行 TSS に以下のレジスタのステートをコピーする。すべての汎用レジスタ、セグメント・レジスタのセグメント・セレクタ、EFLAGS レジスタの一時的にセーブされたイメージ、命令ポインタレジスタ（EIP）。

9. タスクスイッチが CALL 命令、例外、または割り込みによるものである場合、プロセッサは新しいタスクからロードされた EFLAGS 内の NT フラグをセットする。IRET 命令または JMP 命令によるものである場合は、NT フラグは新しいタスクからロードされた EFLAGS 内のステートを表している（表 6-2. を参照）。
10. タスクスイッチが CALL 命令、JMP 命令、例外、または割り込みによるものである場合、プロセッサは新しいタスクの TSS ディスクリプタのビジー（B）フラグをセットする。IRET 命令によるものである場合は、ビジー（B）フラグはセットされたままになる。
11. タスクレジスタに新しいタスクの TSS のセグメント・セレクタおよびセグメント・ディスクリプタをロードする。
12. TSS ステートがプロセッサにロードされる。これには、LDTR レジスタ、PDBR（制御レジスタ CR3）、EFLAGS レジスタ、EIP レジスタ、汎用レジスタ、セグメント・セレクタが含まれる。なお、このステートのロードの実行中にフォルトが発生すると、アーキテクチャ・ステートが壊れることがある。
13. セグメント・セレクタに関連付けられるディスクリプタがロードされ、検証される。このロードと検証に関連するすべてのエラーは、新しいタスクのコンテキスト内で発生する。

注記

この時点で、すべてのチェックとセーブが正常に完了している場合、プロセッサはタスクスイッチを切り替える。上記ステップ 1～11 で回復不能なエラーが起こった場合、プロセッサはタスクスイッチを完了せずに、タスクスイッチを開始した命令の実行前のステートに戻る。ステップ 12 で回復不可能なエラーが起こった場合、アーキテクチャのステートは壊されるが、以前の実行環境でエラーを処理しようと試みる。切り替えポイント後（ステップ 13）に回復不能なエラーが起こった場合、プロセッサはタスクスイッチを完了し（追加のアクセスチェックやセグメント使用可能チェックを行わずに）、新しいタスクの開始前に該当する例外を生成する。例外が切り替えポイント後に発生した場合は、例外ハンドラが、プロセッサにタスク実行を許可する前に、タスクスイッチそのものを終了しなければならない。例外がタスクスイッチの切り替えポイント後に発生した場合に例外がタスクに及ぼす作用の詳細については、第 5 章の「割り込み 10 － 無効 TSS 例外（#TS）」を参照。

---

14. 新しいタスクの実行を開始する。（例外ハンドラからは、新しいタスクの最初の命令は実行されなかったように見える。）

タスクスイッチが成功した場合、必ず現在実行中のタスクのステートがセーブされる。タスクが再開される場合、EIP レジスタにセーブされた値が指す命令から実行が始まり、すべてのレジスタにはタスクが保留になった時点に入っていた値がリストアされる。

タスクを切り替える場合、新しいタスクは保留になったタスクからその特権レベルを引き継ぐわけではない。新しいタスクは CS レジスタの CPL フィールドで指定される特権レベル（TSS からロードされる）で実行を開始する。タスク同士は独立したアドレス空間と TSS によって隔離されているので、また特権規則により TSS へのアクセスが制御されているので、ソフトウェアはタスクスイッチ時に明示的な特権チェックを行う必要がない。

表 6-1. に、タスクスイッチ時にプロセッサがその有無をチェックする例外条件を示す。また各チェックについて、エラーが検出された場合に生成される例外と、そのエラーコードが参照するセグメントも併せて示す。（表中のチェックの順序は、P6 ファミリ・プロセッサでそれらが行われる順序の通りである。この順序はモデル固有であり、他の IA-32 プロセッサではこれと異なっていることもある。）これらの例外を処理するように設計されている例外ハンドラは、その例外を生成したセグメント・セレクタを再ロードしようと試みるとリカーシブ呼び出しを受けることがある。セレクタを再ロードする前に、例外の原因（原因が複数の場合は最初の原因）を修正しなくてはならない。

表 6-1. タスクスイッチ時にチェックされる例外条件

チェックされる条件	例 [1]	エラーコードが参照するもの [2]
TSS ディスクリプタのセグメント・セレクタが GDT を参照しており、そのテーブルのリミット内にある。	#GP #TS（IRET の場合）	新しいタスクの TSS
TSS ディスクリプタはメモリに存在する。	#NP	新しいタスクの TSS
TSS ディスクリプタはビジーでない（呼び出し、割り込み、例外によるタスクスイッチの場合）。	#GP（JMP、CALL、INT の場合）	タスクのバックリンク TSS
TSS ディスクリプタはビジーでない（IRET 命令によるタスクスイッチの場合）。	#TS （IRET の場合）	新しいタスクの TSS
TSS セグメント・リミットが 108 以上（32 ビット TSS の場合）または 44 以上（16 ビット TSS の場合）である。	#TS	新しいタスクの TSS
レジスタに TSS 内の値をロードした。		
新しいタスクの LDT セグメント・セレクタは有効である。[3]	#TS	新しいタスクの LDT
コード・セグメントの DPL がセグメント・セレクタの RPL と一致する。	#TS	新しいコード・セグメント
SS セグメント・セレクタが有効である。[2]	#TS	新しいスタック・セグメント
スタック・セグメントはメモリに存在する。	#SF	新しいスタック・セグメント
スタック・セグメントの DPL が CPL に一致する。	#TS	新しいスタック・セグメント
新しいタスクの LDT がメモリに存在する。	#TS	新しいタスクの LDT
CS セグメント・セレクタは有効である。[3]	#TS	新しいコード・セグメント
コード・セグメントがメモリに存在する。	#NP	新しいコード・セグメント
スタック・セグメントの DPL がセレクタの RPL に一致する。	#TS	新しいスタック・セグメント

表 6-1. タスクスイッチ時にチェックされる例外条件（続き）

チェックされる条件	例[1]	エラーコードが参照するもの[2]
DS、ES、FS、GS セグメント・セレクタは有効である。[3]	#TS	新しいデータ・セグメント
DS、ES、FS、GS セグメントは読み取り可能である。	#TS	新しいデータ・セグメント
DS、ES、FS、GS セグメントはメモリに存在する。	#NP	新しいデータ・セグメント
DS、ES、FS、GS セグメント（コンフォーミング・セグメントでない場合）の DPL が CPL 以上である。	#TS	新しいデータ・セグメント

注：

1. #NP はセグメント不在例外、#GP は一般保護例外、#TS は無効 TSS 例外、#SF はスタックフォルト例外を表す。
2. エラーコードにはこの欄で参照されるセグメント・ディスクリプタへのインデックスが入る。
3. セグメント・セレクタが適合するタイプのテーブル（GDT または LDT）に含まれ、テーブルのセグメント・リミット内の 1 アドレスを占め、適合するタイプのディスクリプタを参照する場合に、そのセグメント・セレクタは有効となる（例えば CS レジスタ内のセグメント・セレクタは、それがコード・セグメント・ディスクリプタを指す場合に限り有効である）。

制御レジスタ CR0 の TS（タスクスイッチ完了）フラグは、タスクスイッチが起こるたびにセットされる。システム・ソフトウェアは、TS フラグを使用して、プロセッサの残りの部分について浮動小数点例外を生成するときに浮動小数点ユニットの演算を調整する。TS フラグは、浮動小数点ユニットのコンテキストが現行タスクのそれと異なっている可能性があることを表す。TS フラグの機能および使用の詳細については、2.5. 節「制御レジスタ」を参照。

6.4. タスクのリンク

TSS の前のタスクへのリンク・フィールド（「バックリンク」とも呼ばれる）と EFLAGS レジスタの NT フラグを使用して、実行を前のタスクに戻すことができる。NT フラグは、現在実行中のタスクが別のタスクの内部にネストされているかどうかを示し、現行タスクの TSS のリンク・フィールドには、ネスト階層でそれより上位のタスクがあればその TSS セレクタが入っている（図 6-7. を参照）。

CALL 命令、割り込みまたは例外がタスクスイッチを引き起こした場合、プロセッサは現在実行中の TSS のセグメント・セレクタを、新しいタスクの TSS のリンク・フィールドへコピーした後、EFLAGS レジスタの NT フラグをセットする。NT フラグは、セーブされている TSS セグメント・セレクタが TSS のリンク・フィールドにロードされたことを示す。ソフトウェアが IRET 命令を使用して新しいタスクを保留にした場合、プロセッサはリンク・フィールドに入っている値と NT フラグを使用して前のタスクに戻る、つまり、NT フラグがセットされている場合、プロセッサは、リンク・フィールドに指定されたタスクへのタスクスイッチを実行することになる。

注記

JMP 命令がタスクスイッチを引き起こした場合、新しいタスクはネストされない。つまり、NT フラグは 0 にセットされ、前のタスクへのリンク・フィールドは使用されない。JMP 命令は、ネストが適切ではない場合に新しいタスクをディスパッチするために使用される。

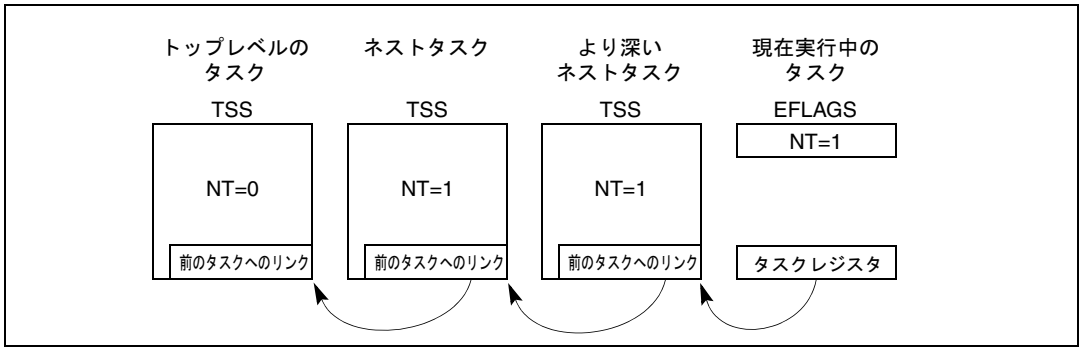


図 6-7. ネストタスク

表 6-2. に、タスクスイッチ時におけるビジーフラグ（TSS セグメント・ディスクリプタの中にある）、NT フラグ、リンク・フィールドおよび TS フラグ（制御レジスタ CR0 の中にある）の使用法をまとめて示す。NT フラグはどの特権レベルで実行中のソフトウェアからでも変更できるのに注意する。プログラムでその NT フラグをセットして IRET 命令の実行も可能である。この場合 IRET 命令は、現行タスクの TSS のリンク・フィールドに指定されているタスクを起動する効果を果たす。疑似タスクスイッチが実行されないようにしておくためには、オペレーティング・システムは TSS を作成するごとにリンク・フィールドを 0 に初期化する必要がある。

表 6-2. ビジーフラグ、NT フラグ、リンク・フィールドおよび TS フラグに対するタスクスイッチの影響

フラグまたは フィールド	JMP 命令の影響	CALL 命令または 割り込みの影響	IRET 命令の影響
新しいタスクのビジー（B）フラグ	フラグがセットされる。その前はクリアされていなければならない。	フラグがセットされる。その前はクリアされていなければならない。	変更なし。セットされていなければならない。
古いタスクのビジーフラグ	フラグがクリアされる。	変更なし。フラグは現在セットされている。	フラグがクリアされる。
新しいタスクの NT フラグ	新しいタスクの TSS の値にセットされる。	フラグがセットされる。	新しいタスクの TSS の値にセットされる。
古いタスクの NT フラグ	変更なし。	変更なし。	フラグがクリアされる。
新しいタスクのリンク・フィールド	変更なし。	ここに古いタスクの TSS のセレクタがロードされる。	変更なし。

表 6-2. ビジーフラグ、NT フラグ、リンク・フィールドおよび TS フラグに対するタスクスイッチの影響（続き）

フラグまたはフィールド	JMP 命令の影響	CALL 命令または割り込みの影響	IRET 命令の影響
古いタスクのリンク・フィールド	変更なし。	変更なし。	変更なし。
制御レジスタ CR0 の TS フラグ	フラグがセットされる。	フラグがセットされる。	フラグがセットされる。

6.4.1. ビジーフラグを使用したリカーシブ・タスク・スイッチの防止

TSS には 1 つのタスクのコンテキストしかセーブできない。したがって、タスクが呼び出される（ディスパッチされる）と、そのタスクへのリカーシブ（または再入可能な）呼び出しによってタスクの現在のステートが失われる場合がある。TSS セグメント・ディスクリプタのビジーフラグを使用すると、再入可能なタスクスイッチ、およびそれに続きタスクのステート情報が失われることを防止できる。プロセッサはビジーフラグを次のように管理する。

1. タスクをディスパッチするとき、プロセッサは新しいタスクのビジーフラグをセットする。
2. タスクスイッチ中に、現在のタスクがネストされたチェーンに配置された（タスクスイッチが CALL 命令、割り込み、または例外によって発生した）場合、現在のタスクのビジーフラグはセット状態のままになる。
3. あるタスクへ切り替えるとき（CALL 命令、割り込み、または例外による場合）、新しいタスクのビジーフラグがすでにセットされている場合、プロセッサは一般保護例外（#GP）を生成する。（タスクスイッチが IRET 命令によるものである場合は、プロセッサはビジーフラグがセットされていることを前提としているので例外を生成しない。）
4. タスクが新しいタスクへのジャンプ（タスクコード内の JMP 命令による場合）またはタスクコード内の IRET 命令によって終了したときに、プロセッサはビジーフラグをクリアし、タスクを「ビジーではない」ステートに戻す。

このようにして、プロセッサはタスクがそれ自体に、またはタスクのネストチェーンに属する任意のタスクに切り替わるのを防げば、リカーシブなタスクスイッチを防止する。複数の呼び出し、割り込み、または例外によって、ネストされた保留中のタスクのチェーンが長くなることがある。ビジーフラグを使用すると、タスクがこのチェーンの中にあるときに呼び出されるのを防止できる。

ビジーフラグはマルチプロセッサ構成でも使用できる。プロセッサはビジーフラグをセットまたはクリアするときに（バスまたはキャッシュにおいて）LOCK プロトコルに従うからである。このロックが、2 つのプロセッサが同時に同一のタスクを起動することを防ぐ。（マルチプロセッサ・アプリケーションでビジーフラグをセットする場合の詳細については、7.1.2.1. 項「自動バスロック」を参照。）

6.4.2. タスク・リンケージの変更

単一プロセッサ・システムで、リンクされたタスクのチェーンからあるタスクを除去する必要がある状況では、次の手順によってタスクを除去する。

1. 割り込みをディスエーブルにする。
2. 割り込んだタスク（つまり除去されるタスクを保留にしたタスク）の TSS にあるリンク・フィールドを変更する。この場合割り込んだタスクは、チェーンの中で除去されるタスクの次のタスク（より新しいタスク）であると見なされる。このとき、リンク・フィールドは、チェーンの中の次に古いタスクの TSS、またはそれより古いタスクを指すように変更する。
3. チェーンから除去するタスクに対応する、TSS セグメント・ディスクリプタのビジー（B）フラグをクリアする。チェーンから複数のタスクを除去しようとする場合は、除去するタスクごとにビジーフラグをクリアしなければならない。
4. 割り込みをイネーブルにする。

マルチプロセッサ・システムの場合は、この手順の他に同期化とシリアル化の処理を行い、リンク・フィールドが変更され、ビジーフラグがクリアされたときに、TSS とそのセグメント・ディスクリプタが両方ともロックされるようにしなければならない。

6.5. タスクアドレス空間

1 つのタスクのアドレス空間は、そのタスクがアクセスできるセグメントで構成される。このようなセグメントとしては、TSS 内で参照されるコード、データ、スタックおよびシステム・セグメント、その他タスクコードによってアクセスされる任意のセグメントがある。これらのセグメントは、まずプロセッサのリニアアドレス空間内にマッピングされ、さらに（直接にまたはページングによって）プロセッサの物理アドレス空間内へマッピングされる。

TSS の LDT セグメント・フィールドを使用して、タスクごとに専用の LDT を与えられる。タスクに専用の LDT を与えると、そのタスクに対応するすべてのセグメントのセグメント・ディスクリプタをそのタスクの LDT に入れることにより、そのタスクのアドレス空間を他のタスクから隔離できる。

また、複数のタスクが同一の LDT を使用することも可能である。これは、システム全体の保護バリアを壊すことなく、複数のタスクが互いに通信したり制御しあえるようにする、単純でメモリ効率の良い方法である。

すべてのタスクはGDTへのアクセスを持つので、このテーブルに入っているセグメント・ディスクリプタを介してアクセスできる共有のセグメントを作成することも可能である。

ページングがイネーブルされている場合、TSSのレジスタCR3（PDBR）フィールドで、リニアアドレスから物理アドレスへマッピングするための専用のページ・テーブル・セットを各タスクが持てるようにできる。また、複数のタスクが同じページ・テーブル・セットも共有できる。

6.5.1. タスクのリニアー物理アドレス空間へのマッピング

次の2通りの方法のどちらかによって、タスクをリニアアドレス空間と物理アドレス空間へマッピングできる。

- 1つのリニアー物理アドレス空間マッピングがすべてのタスクに共有されている。ページングがイネーブルされていない場合はこれだけが選択可能である。この場合はページングを使用することなく、すべてのリニアアドレスが同一の物理アドレスへマッピングされる。ページングがイネーブルされている場合は、すべてのタスクについて1つのページ・ディレクトリを使用することにより、この形式のリニアー物理アドレス空間マッピングが得られる。デマンドページ式仮想メモリがサポートされている場合は、リニアアドレス空間が使用可能な物理アドレス空間より大きくてもよい。
- 各タスクが専用のリニアアドレス空間を持ち、それが物理アドレス空間にマッピングされる。この形式のマッピングは、タスクごとに別々のページ・ディレクトリを使用すれば実現する。タスクスイッチごとにPDBR（制御レジスタCR3）にロードが行われるので、各タスクでページ・ディレクトリが異なることも起こる。

複数の異なるタスクのリニアアドレス空間が、互いに完全に離れた物理アドレスへマッピングされることもある。異なるページ・ディレクトリのエントリが異なるページテーブルを指し、かつそのページテーブルが物理メモリの異なるページを指す場合、タスク同士は物理アドレスを全く共有しない。

どちらかのタスク・リニア・アドレス空間マッピング方式を使用して、すべてのタスクのTSSが、すべてのタスクからアクセス可能な物理空間の共有領域に入っているようにしなければならない。このマッピングでは、タスクスイッチ時にプロセッサがTSSの読み取りや更新を行っている間、TSSアドレスのマッピングが変化しないことが要求される。GDTによってマッピングされたリニアアドレス空間も、物理空間の共有領域にマッピングされる必要がある。そうしないと、GDTの目的に反することになる。図6-8.に、ページテーブルを共有して、2つのタスクのリニアアドレス空間を物理空間内でオーバラップさせる方法を示す。

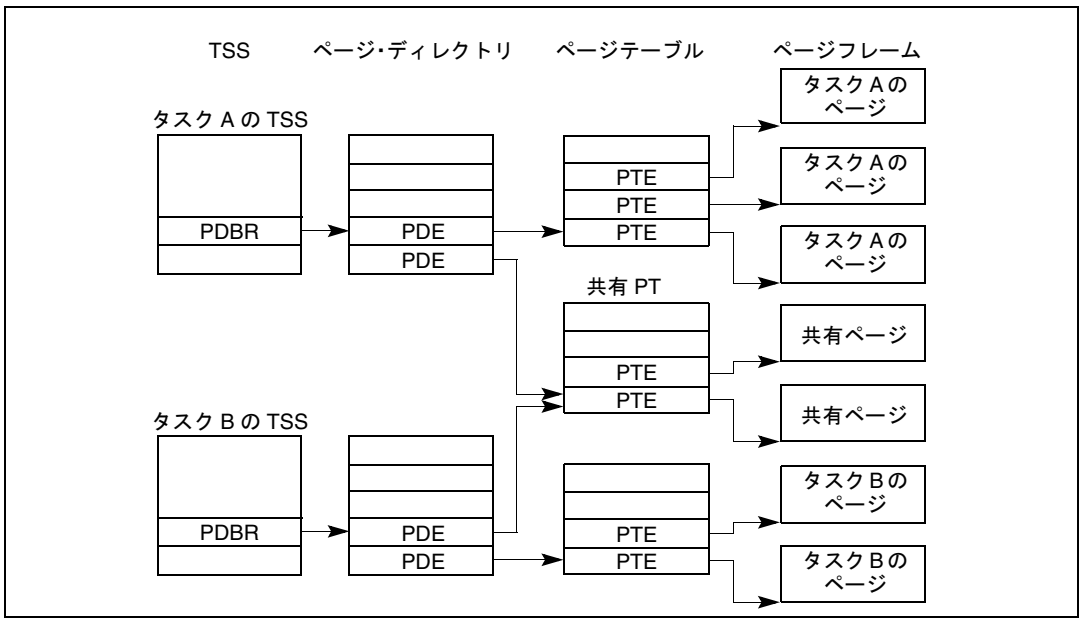


図 6-8. リニアー物理マッピングのオーバラップ

6.5.2. タスクの論理アドレス空間

複数のタスクでデータの共有を可能にするには、以下のいずれかの方法で、データ・セグメント用の共有の論理ー物理アドレス空間マッピングを作成する。

- GDT 内のセグメント・ディスクリプタを使用する方法。すべてのタスクは GDT 内のセグメント・ディスクリプタへのアクセスを持たなければならない。GDT 内のセグメント・ディスクリプタの一部が、物理アドレス空間の中のすべてのタスクに共通の 1 領域にマッピングされる、リニアアドレス空間内にある複数のセグメントを指す場合、すべてのタスクがこれらのセグメント内のデータとコードを共有できる。
- 1 つの共有 LDT を使用する方法。2 つ以上のタスクの TSS の LDT フィールドが同じ LDT を指す場合、それらのタスクは同一の LDT を使用できる。共有の LDT 内のセグメント・ディスクリプタの一部が、物理アドレス空間内の共通の 1 領域にマッピングされる複数のセグメントを指す場合、これらのセグメント内のデータとコードを、LDT を共有するタスクで共有することができる。この共有方法は、共有を特定のタスクに限定できるので、GDT を使用した場合よりも選択性が高い。システム内の他のタスクは、それらに共有セグメントへのアクセスを与えない別の LDT を持つ可能性がある。
- リニアアドレス空間内の共通のアドレスにマッピングされる、複数の別々の LDT にあるセグメント・ディスクリプタを使用する方法。各タスクについてこのリニアアドレス空間内の共通領域が、物理アドレス空間の同一の領域にマッピングさ

れる場合、これらのセグメント・ディスクリプタによってタスクはセグメントを共有できる。このようなセグメント・ディスクリプタを一般にエイリアスと呼ぶ。この共有方法は上記の 2 つ方法よりもさらに選択性が高い。この場合は LDT 内の他のセグメント・ディスクリプタが、共有されない独立のリニア・アドレスを指す可能性があるからである。

6.6. 16 ビットのタスク・ステート・セグメント（TSS）

32 ビット IA-32 プロセッサでは、インテル® 286 プロセッサで使用されているような 16 ビットの TSS 形式も認識する（図 6-9. を参照）。このフォーマットは、以前の IA-32 プロセッサで実行するように書かれたソフトウェアとの互換性を保つためにサポートされている。

16 ビットの TSS については、以下の点に注意が必要である。

- 16 ビットの TSS を使用して仮想 8086 タスクを使用しないこと。
- 16 ビットの TSS の有効セグメント・リミットは 2CH である。
- 16 ビットの TSS には、制御レジスタ CR3 にロードされる、ページ・ディレクトリのベースアドレス用のフィールドがない。したがって、16 ビットのタスクでは、各タスクについて個別のページテーブルのセットはサポートされない。16 ビットのタスクがディスパッチされると、前のタスクのページテーブル構造が使用される。
- 16 ビットの TSS には I/O ベースアドレスは含まれないので、I/O マップの機能はサポートされない。
- タスクステートが 16 ビットの TSS にセーブされると、EFLAGS レジスタと EIP レジスタの上位 16 ビットが失われる。
- 16 ビット TSS から汎用レジスタがロードまたはセーブされると、レジスタの上位 16 ビットが変更され、保持されない。

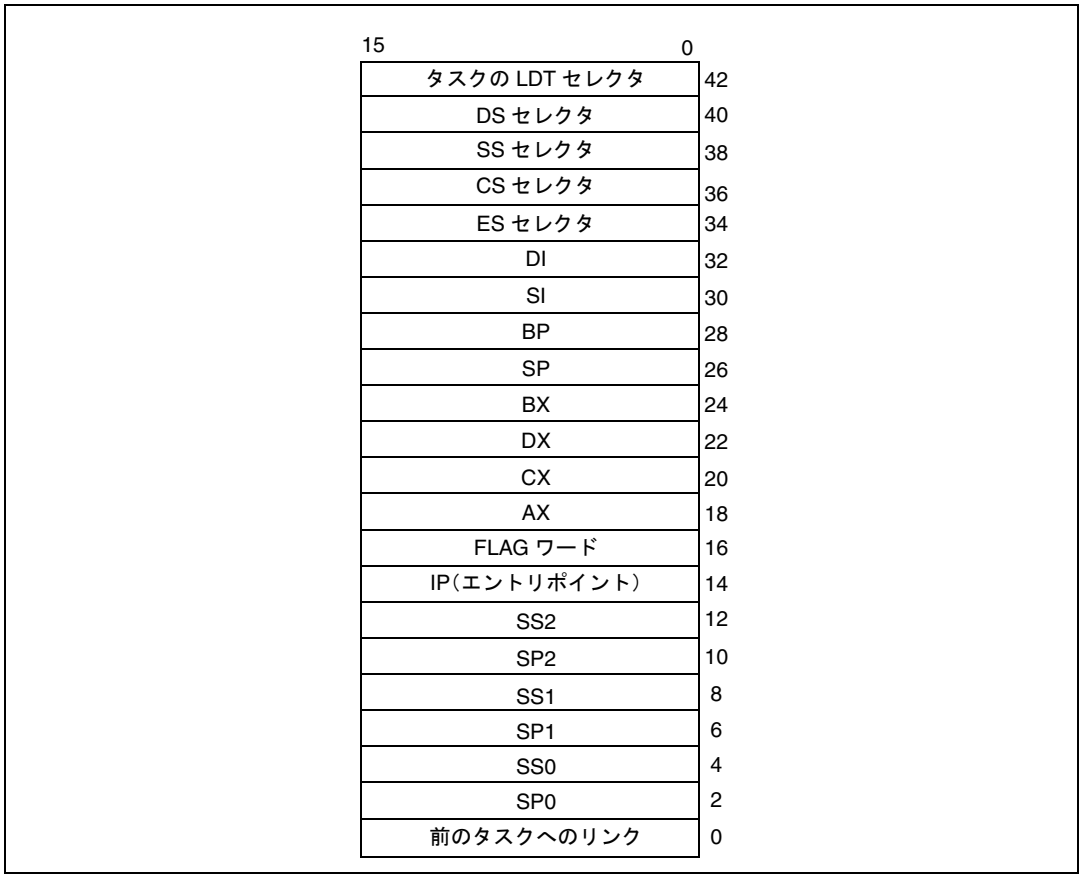


図 6-9. 16 ビットの TSS 形式

7

マルチプロセッサ管理

第 7 章
マルチプロセッサ管理

7

IA-32 アーキテクチャは、同一のシステムバスに接続された複数のプロセッサを管理し処理能力を向上させる、いくつかの機構を備えている。これには次のようなものがある。

- バスロック / キャッシュ・コヒーレンシ管理。システムメモリに対するアトミック操作実行用。
- シリアル化命令。（これらの命令はインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサでのみ使用可能。）
- プロセッサ・チップ上に配置されているアドバンスド・プログラマブル割り込みコントローラ（APIC: Advance Programmable Interrupt Controller）（第 8 章「アドバンスド・プログラマブル 割り込みコントローラ（APIC）」を参照）。APIC アーキテクチャはインテル Pentium プロセッサ以降の IA-32 プロセッサに導入されている。
- 第 2 レベル・キャッシュ（レベル 2、L2）。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサの場合は、L2 キャッシュはプロセッサ・パッケージに封入され、プロセッサと密接に結合されている。インテル Pentium プロセッサおよび Intel486™ プロセッサの場合は、外部 L2 キャッシュをサポートするピンが用意されている。
- 3 次キャッシュ（レベル 3、L3）。インテル Xeon プロセッサでは、L3 キャッシュはプロセッサ・パッケージに封入され、プロセッサと密接に結合されている。
- ハイパー・スレッディング・テクノロジ。IA-32 アーキテクチャを拡張する技術であり、1 つのプロセッサ・コアが 2 つ以上の実行スレッドを同時に実行できるようにする（7.6. 節「ハイパー・スレッディング・テクノロジ」を参照）。

これらの機構は、とりわけ対称的マルチプロセシング（SMP）システムで利用すると便利であるが、IA-32 プロセッサと専用プロセッサ（通信、グラフィクス、ビデオ・プロセッサなど）がシステムバスを共有するようなアプリケーションでも使用できる。

これらのマルチプロセシング機構の目的は次のとおりである。

- システムメモリのコヒーレンシの維持 ― 複数のプロセッサが同時にシステムメモリ内の同一アドレスへアクセスを試みる場合、データ・コヒーレンシを向上させ、場合によっては一方のプロセッサが一時的にメモリ位置をロックできるよう

に、何らかの通信機構またはメモリ・アクセス・プロトコルが使用可能になっていなければならない。

- キャッシュ・コンシステンシの維持 － あるプロセッサにキャッシュされているデータに別のプロセッサがアクセスした場合に、後のプロセッサは誤ったデータを受け取らないようになっていなければならない。後者のプロセッサがデータを変更した場合、そのデータにアクセスする他のすべてのプロセッサも変更されたデータを受け取るようになっていなければならない。
- メモリへの書き込みは予測可能なオーダリングであること － 状況によっては、メモリ書き込みが正確にプログラムされたとおりの順序で行われているかを外部から観察することが重要になる。
- 割り込み処理をあるグループのプロセッサ間に分散すること － 1 つのシステム内で複数のプロセッサが並行して動作している場合、割り込みを受け取って使用可能なプロセッサに分散して処理させる、集中型機構があると便利である。
- 最新のオペレーティング・システム、アプリケーションのマルチスレッド、マルチプロセス機能を活用して、システム・パフォーマンスを向上させること。

IA-32 アーキテクチャのキャッシュ機構およびキャッシュ・コンシステンシについては、第 10 章「メモリ・キャッシュ制御」で説明する。APIC アーキテクチャについては、第 8 章「アドバンスド・プログラマブル 割り込みコントローラ（APIC）」で説明する。本章では、バス、メモリのロック、シリアル化命令、メモリ・オーダリングおよびハイパ・スレッド・テクノロジについて説明する。

7.1. ロックされたアトミック操作

32 ビット IA-32 プロセッサは、システムメモリ内の位置を対象とした、ロックされたアトミック操作をサポートしている。これらの動作は一般に、共有のデータ構造（セマフォ、セグメント・ディスクリプタ、システム・セグメント、ページテーブルなど）を管理するために使用される。このようなデータ構造では、複数のプロセッサが同時に同一のフィールドやフラグの変更を試みる事態が起こる可能性がある。プロセッサは、次の 3 つの相互依存の機構を使用してロックされたアトミック操作を実行する。

- 保証付きのアトミック操作。
- バスロック。LOCK# 信号と LOCK 命令プリフィックスを使用する。
- キャッシュ・コヒーレンシ・プロトコル。キャッシュされたデータ構造にアトミック操作が実行されるように保証するもの。この機構は、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサに存在する。

これらの機構は次のように相互に依存している。すなわち、ある種の基本的なメモリ・トランザクション（システムメモリ内の 1 バイト読み取り / 書き込みなど）は、常にアトミック処理されることが保証されている。つまり、いったん動作が開始されると、

プロセッサは、別のプロセッサやバス・エージェントがそのメモリ位置へアクセス可能になるまでにその動作が完了するように、保証する。プロセッサはまた、選択されたメモリ操作（例えばメモリの共有領域におけるリード・モディファイ・ライト操作）を実行するためのバスロックもサポートする。選択されたメモリ操作とは、通常アトミック処理が必要ではあるが、自動的にそう処理されるようにはなっていない操作である。頻繁に使用されるメモリ位置は、プロセッサのL1またはL2キャッシュに入れられることが多いので、アトミック操作は、プロセッサのキャッシュ内部ではバスロックをアサートしなくても実行できることが多い。この場合、プロセッサのキャッシュ・コヒーレンシ・プロトコルにより、キャッシュされたメモリ位置にアトミック操作が行われている間は、同一メモリ位置をキャッシュしようとする他のプロセッサが適切に管理されるように、保証される。

ロックされたアトミック操作を処理する機構は、IA-32プロセッサが複雑になるにつれて発展してきた。したがって、最新のIA-32プロセッサ（インテルPentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサなど）では、以前のIA-32プロセッサに比べて高度なロック機構が用意されている。詳細については以下に説明する。

7.1.1. 保証付きアトミック操作

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™プロセッサは、次の基本的なメモリ操作が常にアトミックに実行されるよう保証する。

- 1バイトの読み取りまたは書き込み。
- 16ビット境界にアライメント合わせされた1ワードの読み取りまたは書き込み。
- 32 ビット境界にアライメントが調整された 1 ダブルワードの読み取りまたは書き込み。

インテルPentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサ、インテルPentiumプロセッサは、これに加えて、次のメモリ操作が常にアトミックに実行されるよう保証する。

- 64 ビット境界にアライメントが調整された 1 クワッドワードの読み取りまたは書き込み。
- 32ビット・データ・バス内に納まる、キャッシュされていないメモリ位置への16ビット・アクセス。

P6 ファミリ・プロセッサでは、これらに加えて、次のメモリ操作が常にアトミックに実行されることが保証されている。

- 32 バイト・キャッシュ・ライン内に納まる、キャッシュされたメモリ位置へのアライメントが調整されていない 16、32、64 ビット・アクセス。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサでは、キャッシュ可能メモリへのアクセスで、バス幅、キャッシュ・ライン、ページ境界をまたがっているものは、アトミック操作が保証されない。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサは、外部メモリ・サブシステムによってこのようなスプリット・アクセスをアトミック操作にできるようにするバス制御信号を備えているが、アライメントの合っていないデータアクセスを行うとプロセッサの処理能力に重大な影響を及ぼすため、使用を避けるべきである。

7.1.2. バスのロック

IA-32 プロセッサは LOCK# の信号を備えている。ある種の重要なメモリ操作の際、プロセッサは自動的にこの信号をアサートしてシステムバスをロックさせる。この出力信号がアサートされている間は、他のプロセッサやバス・エージェントがバス制御要求を出しても無視される。ソフトウェアは、命令の前に LOCK プリフィックスを付加すると、上記以外の状況で LOCK# 信号がアサートされる機会を指定できる。

Intel386™ プロセッサ、Intel486™ プロセッサ、インテル® Pentium® プロセッサの場合は、命令を明示的にロックすると、LOCK# 信号がアサートされる。システム・ハードウェアで LOCK# 信号を使用可能にし、プロセッサ間でメモリアクセスを制御するようにするのは、ハードウェア設計者の責任である。

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、アクセスされるメモリ領域がキャッシュに入っている場合は一般に LOCK# 信号はアサートされず、代わりにプロセッサのキャッシュにロックが適用されるだけである（7.1.4. 項「プロセッサ内部キャッシュ上のロック操作」を参照）。

7.1.2.1. 自動バスロック

次の操作時には、プロセッサは自動的に LOCK# 信号をアサートする。

- **メモリを参照する XCHG 命令の実行中。**
- **TSS ディスクリプタの B（ビジー）フラグをセットするとき。**プロセッサはタスクに切り替えるとき、TSS ディスクリプタのタイプ・フィールドのビジーフラグをテストし、セットする。2 つのプロセッサが同時に同一タスクに切り替えないよう

にするために、プロセッサはこのフラグをテストしてセットする間LOCK#信号をアサートしておく。

- **セグメント・ディスクリプタを更新するとき。**プロセッサはセグメント・ディスクリプタをロードするとき、セグメント・ディスクリプタ内のアクセスド・フラグがクリアされている場合はそれをセットする。この動作の間、プロセッサはLOCK#信号をアサートし、ディスクリプタの更新中に他のプロセッサがこれを変更しないようにする。この措置が有効になるためには、オペレーティング・システムのディスクリプタを更新するプロシージャで次のステップを行う必要がある。
  - ロックされた操作を使用して、セグメント・ディスクリプタが不在を表すようにアクセス権バイトを変更してから、ディスクリプタが更新中を示すようにタイプ・フィールドの値を指定する。
  - セグメント・ディスクリプタの各フィールドを更新する。（この動作はいくつかのメモリアクセスを必要とする場合があるので、ロックされた操作は使用できない。）
  - ロックされた操作を使用して、セグメント・ディスクリプタが存在しかつ有効なことを表すように、アクセス権バイトを変更する。
- Intel386™プロセッサでは、セグメント・ディスクリプタのアクセスド・フラグがクリアされているかどうかにかかわらず、必ずこれを更新することに注意する。インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™プロセッサの場合は、このフラグがまだセットされていない場合にのみ更新する。
- **ページ・ディレクトリとページテーブルのエントリを更新するとき。**ページ・ディレクトリとページテーブルのエントリを更新するとき、プロセッサはロックサイクルを使用して、ページ・ディレクトリとページテーブルのエントリの中の、アクセスされたダーティフラグをセットする。
- **割り込みの確認。**割り込み要求の後、割り込みコントローラはデータバスを使用してプロセッサに割り込みの割り込みベクタを送信する場合がある。割り込みベクタの送信中にほかのデータがデータバス上に存在しないようにするために、プロセッサはLOCK#信号をアサートする。

7.1.2.2. ソフトウェア制御されたバスロック

明示的にバスをロックするために、ソフトウェアは次に示すメモリ位置を変更する命令に先だってLOCKプリフィックスを付加できる。これら以外の命令にLOCKプリフィックスを使用した場合、またはメモリに書き込み操作を行わない場合（つまりデスティネーション・オペランドがレジスタに入っている場合）は、無効オペコード例外（#UD）が生成される。

- ビットテスト/変更命令（BTS、BTR、BTC）。

- 交換命令（XADD、CMPXCHG、CMPXCHG8B）。
- XCHG 命令には自動的に LOCK プリフィックスが付くものと想定されている。
- 次の 1 オペランド算術演算 / 論理演算命令 : INC、DEC、NOT、NEG。
- 次の 2 オペランド算術演算 / 論理演算命令 : ADD、ADC、SUB、SBB、AND、OR、XOR。

ロックされた命令は、デスティネーション・オペランドが定義するメモリ領域だけをロックするように保証されているが、システムの側でこれをもっと大きいメモリ領域のロックと解釈することもある。

ソフトウェアがセマフォ（複数のプロセッサ間で信号を出し合うのに使用する共有メモリ）にアクセスする場合、同じアドレスと同じ長さのオペランドを使用する必要がある。例えばあるプロセッサがワードアクセスによってセマフォにアクセスした場合、他のプロセッサはバイトアクセスでこのセマフォにアクセスしてはならない。

バスロックのメカニズムは、メモリ・フィールドのアライメントに影響されない。オペランド全体を更新するのに必要な回数のバスサイクルについて、LOCK# 信号がアサートされる。ただし、システムの処理能力を上げるためには、ロックされたアクセスはその境界にアライメントを合わせることを推奨する。

- 8 ビット・アクセス（ロックされたものでもそれ以外でも）の場合はどのような境界でもよい。
- ロックされたワードアクセスの場合は 16 ビット境界。
- ロックされたダブルワード・アクセスの場合は 32 ビット境界。
- ロックされたクワッドワード・アクセスの場合は 64 ビット境界。

ロックされた操作は、他のすべてのメモリ操作および外部から認識可能なすべてのイベントに関して、アトミックである。ロックされた命令を渡すことができるのは、命令フェッチとページ・テーブル・アクセスだけである。ロックされた命令を使用して、1 つのプロセッサが書き込み、別のプロセッサが読み取るようなデータを同期化できる。

P6 ファミリ・プロセッサの場合、ロックされた操作によりすべての未実行のロードおよびストア操作がシリアル化される（つまり、前にあるすべての操作が終了するまで待機する）。この規則はインテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサにも当てはまるが、1 つ例外があり、順序設定の緩いメモリタイプ（WC メモリタイプなど）を参照するロード操作がシリアル化されない場合がある。

ロックされた命令を使用して、書き込まれたデータが命令としてフェッチされるように保証してはならない。

注記

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium 4 プロセッサ、Inetl486™ プロセッサの現在のバージョンでは、ロックされた命令によって、書き込まれたデータを命令としてフェッチできる。ただし、自己修正コードを使う必要がある場合は、次の項で説明するように他の同期化機構を使用することを推奨する。

7.1.3. 自己修正コードおよびクロス修正コードの処理

プロセッサが、データをコードとして実行する目的で、現在実行中のコード・セグメントにデータを書き込むようなコードを**自己修正コード**と呼ぶ。IA-32 プロセッサでは、現在の実行ポインタからどの程度離れたコードを修正したかによって、自己修正コードを実行するときにモデル固有の動作をする。プロセッサのアーキテクチャが複雑になり、(インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、および P6 ファミリ・プロセッサのように) リタイアメント・ポイントよりも前に推論によるコードの実行を開始するので、修正の前または後に、どのコードを実行すべきであるかに関する規則はわかりにくくなっている。自己修正コードを作成し、現在および将来の IA-32 アーキテクチャのバージョンに準拠することを保証するには、次のいずれかのコード化オプションを選択しなければならない。

(* オプション 1 *)
コード・セグメントへ変更したコードをストア ;
新たなコードか中間コードへ分岐 ;
新しいコードを実行する ;

(* オプション 2 *)
コード・セグメントへ変更したコードをストア ;
シリアル化命令を実行 ; (* CPUID 等 *)
新しいコードを実行する ;

(インテル® Pentium® プロセッサまたは Intel486™ プロセッサ上で実行するように設計されたプログラムでは、これらのオプションを使用する必要はないが、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサとの互換性を保証するためにいずれかを使用することを推奨する。)

自己修正コードを実行する場合、自己修正コード以外の (通常の) コードに比べて、パフォーマンスのレベルは低くなる。パフォーマンス低下の程度は、修正の頻度とコードの特性によって異なる。

あるプロセッサが、2番目のプロセッサにデータをコードとして実行させる目的で、2番目のプロセッサの現在実行中のコード・セグメントにデータを書き込むことを**クロス修正コード**と呼ぶ。自己修正コードの場合と同様に、IA-32 プロセッサでは、実行中のプロセッサの現在の実行ポインタからどの程度離れたコードを修正したかによって、クロス修正コードを実行するときにモデル固有の動作をする。クロス修正コードを作成し、現在および将来の IA-32 アーキテクチャのバージョンに準拠することを保証するには、以下のプロセッサ同期化アルゴリズムを考慮しなければならない。

```
; プロセッサ変更アクション
Memory_Flag ← 0; （* Memory_Flag を 1 以外の値にセット *）
コード・セグメントへ変更したコードをストア ;
Memory_Flag ← 1;


; プロセッサ実行アクション
WHILE （Memory_Flag ≠ 1）
    コード更新待ち ;
ELIHW;
シリアル化命令を実行 ; （*CPUID 等 *）
変更したコードの実行開始 ;
```

（Intel486 プロセッサ上で実行するように設計されたプログラムでは、このオプションを使用する必要はないが、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサとの互換性を保証するために使用することを推奨する。）

自己修正コードの場合と同様に、クロス修正コードを実行する場合、クロス修正コード以外の（通常の）コードに比べて、パフォーマンスは低くなる。パフォーマンス低下の程度は、修正の頻度とコードの特性によって異なる。

7.1.4. プロセッサ内部キャッシュ上のロック操作

Intel486™ プロセッサおよびインテル® Pentium® プロセッサについては、ロックされているメモリ領域のプロセッサにキャッシュされている場合でも、LOCK 操作中はバス上で常に LOCK# 信号がアサートされる。

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサについては、LOCK 操作中ロックされているメモリ領域を、LOCK 操作を行っているプロセッサがライトバック・メモリとしてキャッシュし、その領域が 1 つのキャッシュ・ラインに完全に内包されている場合、プロセッサはバス上の LOCK# 信号をアサートしないことがある。この場合プロセッサは代わりにメモリ位置を内部的に変更し、そのキャッシュ・コヒーレンシ機構によってその動作がアトミックに実行されることを保証する。この操作を「キャッシュ・ロック」と呼ぶ。キャッシュ・コ

ヒーレンシ機構は自動的に、同一メモリ領域をキャッシュした複数のプロセッサが、その領域内のデータを同時に変更することを防止する。

7.2. メモリ・オーダリング

メモリ・オーダリングとは、プロセッサがシステムメモリへのシステムバスを介して読み取り（ロード）および書き込み（ストア）命令を発行する順序である。IA-32アーキテクチャでは、プロセッサによって、いくつかのメモリ・オーダリング・モデルをサポートしている。例えば、Intel386™プロセッサでは**プログラム・オーダリング**（通常、**ストロング・オーダリング**と呼ばれる）を使用しており、すべての状況において、命令ストリームでの順序でシステムバス上に読み取りおよび書き込みが発行される。

命令の実行を最適化するために、IA-32 インテル・アーキテクチャは、インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサでは、ストロング・オーダリングの代わりに**プロセッサ・オーダリング**と呼ばれるモデルが使用されている。この**プロセッサ・オーダリング**では、バッファリングされた書き込みよりも前に読み取りを実行するなど、操作のパフォーマンスを向上させることができる。このオーダリング・モデルの目的は、命令の実行速度を向上させる一方で、マルチプロセッサ・システムでのメモリ・コヒーレンシを維持することである。

以下の項では、Intel486™ プロセッサ、インテル® Pentium® プロセッサ、インテル Pentium 4プロセッサ、インテル Xeonプロセッサ、P6ファミリ・プロセッサで使用されているメモリ・オーダリング・モデルについて説明する。

7.2.1. インテル® Pentium® プロセッサおよび Intel486™ プロセッサでのメモリ・オーダリング

インテル® Pentium® プロセッサおよびIntel486™プロセッサは、「プロセッサ・オーダリング」と定義されるメモリ・オーダリング・モデルを使用するが、ほとんどの状況ではストロング・オーダリング・プロセッサとして動作する。システムバスでは、読み取りと書き込みは常にプログラムされた順序で実行されるが、次のような場合には、プロセッサ・オーダリングが行われる。バッファリングされた書き込みがすべてキャッシュ・ヒットで、読み取りミスによってアクセスされるアドレスに指定されない場合は、システムバス上でバッファリングされている書き込みよりも先に読み取りミスを実行できる。

I/O操作の場合は、読み取りと書き込みは常にプログラムされた順序で行われる。

プロセッサ・オーダリング・プロセッサ（インテル® Pentium® 4プロセッサ、インテル® Xeon™ プロセッサ、およびP6ファミリ・プロセッサなど）で正しく動作するように設計されたソフトウェアは、インテルPentiumプロセッサまたはIntel486プロセッサのストロング・オーダリングに依存してはならはない。この代わりに、プロセッサ間の並列実

行を制御するための共有変数へのアクセスが、適切なロックまたはシリアル化操作を使用してプログラム・オーダリングに明示的に従う必要があることを保証しなければならない（7.2.4.項「メモリ・オーダリング・モデルのストロング・オーダリング/ウィーク・オーダリング」を参照）。

7.2.2. インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでのメモリ・オーダリング

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサも、「プロセッサ・オーダリング」と定義されるメモリ・オーダリング・モデルを使用する。このプロセッサ・オーダリングは、「ストアバッファ・フォワーディングによる書き込みオーダリング」と定義できる。このオーダリングには次のような特徴がある。

ライトバック・キャッシュ可能と定義されるメモリ領域に対応するシングル・プロセッサ・システムでは、次のオーダリング規則が適用される。

1. 読み取りは推論により、どのような順序ででも実行できる。
2. 読み取りはバッファリングされた書き込みをパスできるが、プロセッサはプログラムが正しく実行されるように保証する。
3. メモリへの書き込みは、常にプログラムされた順序で行われる。ただし、CLFLUSH 命令で実行される書き込み、および非テンポラルな移動命令（MOVNTI、MOVNTQ、MOVNTDQ、MOVNTPS、MOVNTPD）で実行されるストリーミング・ストア（書き込み）は除く。
4. 書き込みはバッファに入れることができる。
5. 推論による書き込みは行われない。書き込みは実際に実行された命令にしたがってだけ行われる。
6. バッファリングされた書き込みのデータはプロセッサ内部で待機中の読み取りよりも先に実行できる。
7. 読み取りまたは書き込みは I/O 命令、ロックされた命令またはシリアル化命令をパスする（それらの命令より先に実行される）ことはできない。
8. 読み取りは、LFENCE 命令と MFENCE 命令をパスすることはできない。
9. 書き込みは、SFENCE 命令をパスすることはできない。

上記の2番目の規則により、読み取りは書き込みをパスすることができる。ただし書き込みが読み取りと同じメモリ位置を対象としている場合は、プロセッサの内部「スヌープ」機構が矛盾を検出し、プロセッサがその値を使用する命令を実行する前に、すでにキャッシュされている読み取りを更新してしまう。

6 番目の規則は、上記が成立しない場合の例外的書き込みオーダリング・モデルにあたる。

「ストアバッファ・フォワーディングによる書き込みオーダリング」（この項の最初で説明）という用語は、規則 2 と 6 を組み合わせた効果を指す。

マルチプロセッサ・システムでは、次のオーダリング規則が適用される。

- 個々のプロセッサは、シングル・プロセッサ・システムの場合と同じオーダリング規則を使用する。
- 単一のプロセッサによる書き込みは、全プロセッサによる書き込みと同じオーダリングにしたがって行われる。
- システムバス上にある各プロセッサからの書き込みは、別の順序にしたがってもよい。

図 7-1. の例は、後者（マルチプロセッサ・システムの場合）の規則を図示したものである。ここでは 3 つのプロセッサがすべて、指定された 3 つのメモリ位置（A、B、C）のそれぞれに書き込みを行っている。個別に見ると、各プロセッサはそれぞれ同じプログラム順序で書き込みを行っているが、バス・アービトレーションその他のメモリアクセス機構のために、3 つのプロセッサが個々のメモリ位置に書き込む順序は、それぞれのコード・シーケンスがプロセッサに対して実行されるごとに異なってもよい。したがって、メモリ位置 A、B、C の最終的な値は、書き込みシーケンスが実行されるたびに異なる可能性がある。

本項で解説しているプロセッサ・オーダリング・モデルは、事実上はインテル® Pentium® プロセッサや Intel486™ プロセッサで使用されているものと同一である。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサで強化された点は、次のとおりである。

- 推論による読み取りのサポートの追加。
- 読み取りが同じメモリ位置への書き込みをパスする場合の、ストアバッファ順方向付け。
- ロング・ストリング・ストアからのアウト・オブ・オーダー・ストアおよびストリング移動操作（7.2.3. 項「インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでのストリングからの アウト・オブ・オーダー・ストア操作」を参照）。

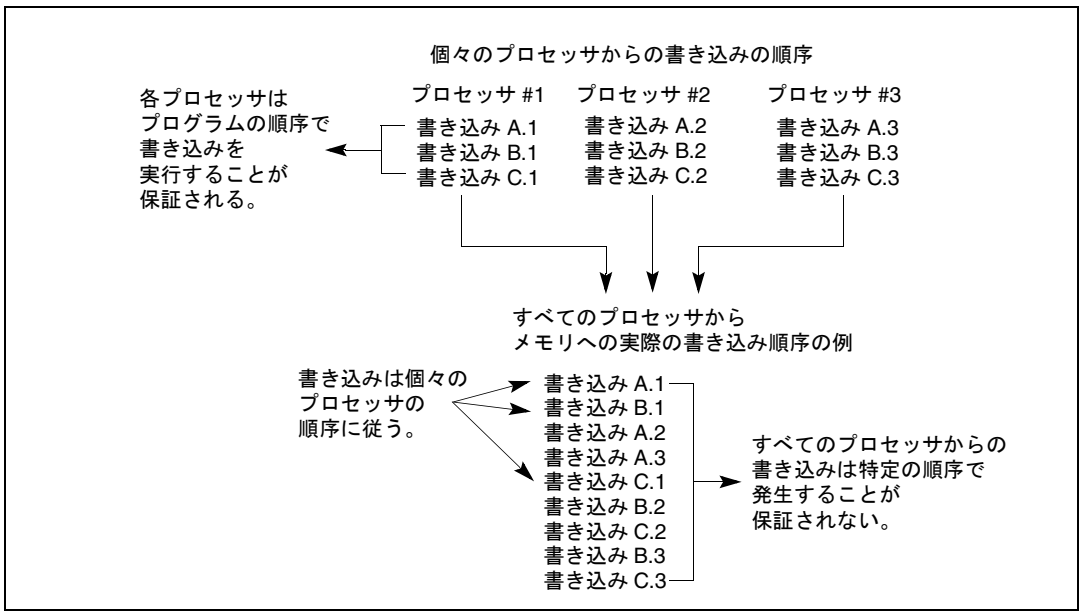


図 7-1. マルチプロセッサ・システムの書き込みオーダリング

7.2.3. インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでのストリングからのアウト・オブ・オーダー・ストア操作

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサは、（MOVS および STOS 命令による）ストリング・ストア操作中に、プロセッサの動作を変更し、パフォーマンスを最大化する。「高速ストリング」操作の初期条件（以下を参照）が満たされると、プロセッサは基本的にストリングをキャッシュ・ラインごとのモードで操作する。この結果、プロセッサはソースアドレスに対してキャッシュ・ライン読み取りを発行し、外部バスのデスティネーション・アドレスで違反となるループに入り、そのストリングの長さのデスティネーション・キャッシュ・ラインのすべてのバイトが変更される。このモードでは、割り込みはキャッシュ・ライン境界上でのみ、プロセッサに受け入れられる。したがって、このモードでは、デスティネーション・ライン違反およびストアが、外部バス上にアウト・オブ・オーダーで発行される可能性がある。

逐次ストアに依存するコードは、ストアするデータ構造全体に対してストリング操作を使用してはならない。データとセマフォを分離する必要がある。順序に依存するコードは、すべてのプロセッサでデータが正しい順序になるようにストリング操作を行った後、個別のセマフォを使用して個々にストアしなければならない。

「高速ストリング」操作の初期状態は次のとおりである。

- インテル® Pentium® III プロセッサでは、EDI と ESI は 8 バイト境界にアライメントされていなければならない。インテル Pentium 4 プロセッサでは、EDI は 8 バイト境界にアライメントされていなければならない。
- ストリング操作をアドレスの昇順に実行しなければならない。
- 初期操作カウンタ（ECX）が 64 以上でなければならない。
- ソースおよびデスティネーションがキャッシュ・ライン（64 バイトインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ、32 バイト P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサ）未満でオーバーラップしてはならない。
- ソースおよびデスティネーション・アドレスのメモリのタイプが WB または WC でなければならない。

7.2.4. メモリ・オーダリング・モデルのストロング・オーダリング / ウィーク・オーダリング

IA-32 アーキテクチャは、特殊なプログラミング状況を処理するために、メモリ・オーダリング・モデルを強めたり（ストロング）弱めたり（ウィーク）する機構を備えている。これには次のようなものがある。

- I/O 命令、ロックする命令、LOCK プリフィックス、シリアル化命令はプロセッサにストロング・オーダリングを強制する。
- SFENCE 命令（インテル® Pentium® III プロセッサで IA-32 アーキテクチャに導入された）と、LFENCE 命令および MFENCE 命令（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで導入された）は、特定のタイプのメモリ操作に対してメモリ・オーダリングとシリアル化の機能を提供する。
- メモリ・タイプ・レンジ・レジスタ（MTRR）を使用して、物理メモリの特定の領域についてメモリ・オーダリングを強めたり弱めたりできる（10.11. 節「メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）」を参照）。MTRR はインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでのみ使用できる。
- ページ属性テーブル（PAT）を使用して、特定のページまたはページグループのメモリ・オーダリングを強められる（10.12. 節「ページ属性テーブル（PAT）」を参照）。PAT は、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、インテル Pentium III プロセッサでのみ使用できる。

これらの機構は次のように利用できる。

メモリ・マップド・デバイスやその他のバス上の I/O デバイスは、その I/O バッファへの書き込みの順序にセンシティブであることが多い。I/O 命令（IN 命令と OUT 命令）

を使用して次のようなアクセスに対し書き込みのストロング・オーダリングを行うことができる。I/O 命令を実行するに先立ち、プロセッサはプログラムでそれより前にあるすべての命令が完了し、バッファに入っているすべての書き込みがメモリに排出されるまで待機する。命令フェッチとページ・テーブル・ウォークだけが、I/O 命令をパスできる。I/O 命令が完了したとプロセッサが判断するまで、その後の命令の実行は始まらない。

マルチプロセッサ・システム内の同期機構がストロング・メモリ・オーダリング・モデルに依存することがある。この場合、プログラムはXCHG命令やLOCKプリフィックスなどのロックする命令を使用して、メモリに対するリード・モディファイ・ライト操作がアトミックに実行されるように保証する。ロックする操作は通常、それより前にあるすべての命令が完了し、バッファに入っているすべての書き込みがメモリに排出されるまで待機する点で、I/O操作と似ている（7.1.2.項「バスのロック」を参照）。

プログラム同期もシリアル化命令を使用して実行される（7.4. 節「シリアル化命令」を参照）。プロシージャやタスクの重要な境界では一般にこの命令を使用し、新しいコード・セクションへジャンプする前や、コンテキスト切替えが起こる前にすべての前の命令が完了するように強制する。ここでもI/O命令やロックする命令と同様、プロセッサは、それより前にあるすべての命令が完了し、バッファに入っているすべての書き込みがメモリに排出されるまで待ってから、シリアル化命令を実行する。

SFENCE、LFENCE、MFENCE の各命令は、順序設定の緩い結果を生成するルーチンとそのデータを使用するルーチンとの間で、効率のよい方法でロードおよびストア時のメモリ・オーダリングがなされることを保証している。これらの命令の機能を以下に示す。

- SFENCE ― プログラム命令ストリーム内のSFENCE命令の前に発生したすべてのストア（書き込み）操作をシリアル化する。ただし、ロード操作には影響を与えない。
- LFENCE ― プログラム命令ストリーム内のLFENCE命令の前に発生したすべてのロード（読み取り）操作をシリアル化する。ただし、ストア操作には影響を与えない。
- MFENCE ― プログラム命令ストリーム内のMFENCE命令の前に発生したすべてのストア操作およびロード操作をシリアル化する。

SFENCE、LFENCE、MFENCEの各命令は、CPUID命令よりも効率的な方法でメモリ・オーダリングを制御できる。

MTRRは、物理メモリの指定された領域に対するキャッシュ特性を定義するために、P6ファミリ・プロセッサで導入された。次に2つの例を挙げ、インテルPentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサでMTRRによってセッ

トアップされたメモリタイプを使用してストロング・メモリ・オーダリングにしたり、ウィーク・メモリ・オーダリングにする様子を示す。

- ストロング・キャッシュされていない（UC）メモリタイプは、メモリアクセスに対してストロング・オーダリング・モデルを強制する。この場合、UC メモリ領域へのすべての読み取りと書き込みは、バス上で行われ、順序に従わないアクセスや推論によるアクセスは実行されない。このメモリタイプをメモリマップド I/O デバイスに専用のアドレス範囲に適用すると、ストロング・メモリ・オーダリングを強制できる。
- ウィーク・オーダリングが許容されるメモリ領域では、ライトバック（WB）メモリタイプを選べる。この場合、読み取りは推論によって実行でき、書き込みはバッファに入れたり組み合わせたりできる。このタイプのメモリの場合、キャッシュ・ラインにまたがらないアトミック（ロックされる）動作に対してキャッシュ・ロックが行われ、これは通常の同期命令（例えば、リード・モディファイ・ライト操作全体の間バスをロックする XCHG のような命令）を使用する場合に付きものの、処理能力のペナルティを低減するのに役立つ。WB メモリタイプの場合、メモリアクセスが 1 キャッシュ・ラインの内部に入っている場合 XCHG 命令はバスでなくキャッシュをロックする。

PAT は、ページまたはページグループに割り当て可能なキャッシュ特性を拡張するために、インテル Pentium III プロセッサで導入された。PAT 機構は、通常は、MTRR によって設定されたキャッシュ特性に対して、ページレベルでキャッシュ特性を強めるために使用される。表 10-7. に、PAT と MTRR の相互作用を示す。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサで実行するように書かれたソフトウェアにはプロセッサ・オーダリング・モデルまたはそれより弱いメモリ・オーダリング・モデルを想定することを推奨する。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、UC メモリタイプを使用する場合以外は、ストロング・メモリ・オーダリング・モデルをサポートしない。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサはプロセッサ・オーダリングをサポートしているという事実にもかかわらず、インテルは将来のプロセッサがこのモデルをサポートすることは保証しない。ソフトウェアを将来のプロセッサに移植できるようにするためには、オペレーティング・システムがクリティカル領域やリソース制御構成体を提供し、I/O 命令やロックする命令やシリアル化命令に基づいたアプリケーション・プログラム・インターフェイス（API）を使用して、マルチプロセッサ・システムの共有メモリ領域へのアクセスを同期化することを推奨する。また、システム・ハードウェアがこのメモリ・オーダリング・モデルをサポートしていないような場合は、ソフトウェアもプロセッサ・オーダリングに依存してはならない。

7.3. 複数のプロセッサに対するページテーブルおよびページ・ディレクトリ・エントリの変更の伝搬

マルチプロセッサ・システムでは、1 つのプロセッサがページテーブルまたはページ・ディレクトリ・エントリを変更した場合、その変更を他のすべてのプロセッサに伝搬しなければならない。このプロセスは、「TLB シュートダウン」と呼ばれる。ページテーブルまたはページ・ディレクトリ・エントリの変更の伝搬は、メモリベースのセマフォ、または複数のプロセッサ間のプロセッサ間割り込み（IPI）、あるいはその両方によって行われる。IA-32 プロセッサの TLB シュートダウン・シーケンスの一例を以下に示す。このシーケンスは、単純ではあるが、アルゴリズムとしては適切である。

1. バリアを開始する。1 つを除いて、すべてのプロセッサを停止する。つまり、1 つを除いて、すべてのプロセッサを HALT 状態にするか、スピンループ内で停止させる。
2. アクティブなプロセッサに必要な PTE または PDE、あるいはその両方を変更させる。
3. すべてのプロセッサに、各プロセッサの TLB 内で、修正された PTE、PDE を無効化させる。
4. バリアを終了する。すべてのプロセッサを再開する。通常の処理を再開する。

パフォーマンスを最適化した、別の TLB シュートダウン・アルゴリズムも開発できる。ただし、開発者は、以下のいずれかの条件が満たされるように保証しなければならない。

- アップデート・プロセス中に、異なる TLB マッピングが異なるプロセッサ上で使用されないようにする。
- アップデート・プロセス中に、プロセッサが古いマッピングを使用した場合は、オペレーティング・システムが対処できるようにする。

7.4. シリアル化命令

IA-32 アーキテクチャでは、いくつかの**シリアル化命令**を定義している。これらの命令は、前の命令がフラグ、レジスタ、メモリに対して行う変更のすべてが完了し、バッファに入っているすべての書き込みがメモリに排出されてから、はじめて次の命令をフェッチして実行することをプロセッサに強制する。例えば、制御レジスタ命令に対してMOVを使用し、制御レジスタCR0に新しい値がロードされて保護モードがイネーブルにされると、プロセッサは常に、保護モードに入る前にシリアル化命令を実行しなければならない。このシリアル化操作によって、プロセッサが実アドレスモードだったときに開始されたすべての操作が完了してから、保護モードへの切り替えが行われることが保証される。

シリアル化命令の概念は、並列命令実行をサポートするために、インテル® Pentium® プロセッサ以降の IA-32 アーキテクチャに導入された。シリアル化命令は、並列命令実行をサポートしない Intel486™ プロセッサ以前のプロセッサでは意味を持たない。

推論により実行された命令の結果は廃棄されるので、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでのシリアル化命令の実行が推論による実行に制限を加えるようになることに注意する。

次の命令はシリアル化命令である。

- 特権シリアル化命令 － MOV（制御レジスタへの）、MOV（デバッグレジスタへの）、WRMSR、INVD、INVLPG、WBINVD、LGDT、LLDT、LIDT、LTR。
- 非特権シリアル化命令 － CPUID、IRET、RSM。
- 非特権メモリ・オーダリング命令 － SFENCE、LFENCE、MFENCE。

プロセッサはある命令の実行をシリアル化するとき、保留中のメモリ・トランザクションのすべて（ストアバッファにストアされた書き込みも含めて）が完了した後に次の命令を実行するように保証する。シリアル化命令をパスできるものは何もなく、シリアル化命令は他の命令（読み取り、書き込み、命令フェッチ、I/O 命令）をいっさいパスできない。

CPUID 命令をどの特権レベルで実行しても、プログラムの流れに影響を与えることなく命令実行をシリアル化できる。ただし EAX、EBX、ECX、EDX レジスタが変更されている場合は除く。

SFENCE、LFENCE、MFENCE の各命令は、メモリのロードおよびストアのシリアル化を制御する際に、よりきめ細かなグラニュラティを提供する（7.2.4. 項「メモリ・オーダリング・モデルのストロング・オーダリング / ウィーク・オーダリング」を参照）。

次の追加情報はシリアル化命令に関しては意味を持たない。

- プロセッサは、命令実行をシリアル化するとき、データ・キャッシュの中にある変更されたデータの内容を外部メモリにライトバックしない。ソフトウェアは、シリアル化命令である WBINVD 命令を実行すれば、変更されたデータがライトバックされるように強制できる。しかし、WBINVD 命令を頻繁に使用するとシステムの処理能力が非常に低下することに注意する。
- ページングをイネーブル / ディスエーブルにする命令を実行する（つまり制御レジスタ CR0 の PG フラグが変更される）場合、その命令の後にジャンプ命令を実行する必要がある。ジャンプ命令のターゲットになる命令は PG フラグの新しい設定値（つまりページングがイネーブルまたはディスエーブル）でフェッチされるが、ジャンプ命令そのものは前の設定値でフェッチされる。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサはレジスタ CR0 への移動の後にジャンプ動作を必要としない（インテル Pentium 4 プロセッサ、インテル Xeon プ

ロセッサ、P6 ファミリ・プロセッサ内の MOV 命令を使用した CR0 への書き込みはいずれも完全にシリアル化であるため）。しかし、他の IA-32 プロセッサ上で実行するように書かれたコードとの下位および上位方向の互換性を維持するためには、ジャンプ動作を実行することを推奨する。

- ページングがイネーブルのときに CR3 の内容を変更する命令を実行すると必ず、次の命令は CR3 の新しい値に対応する変換テーブルを使用してフェッチされる。したがって、次の命令およびシーケンスでその後に続く命令は、CR3 の新しい値に基づいたマッピングを持つはずである。（TLB のグローバル・エントリは無効化されない。10.9. 節「トランスレーション・ルックアサイド・バッファ（TLB）の無効化」を参照。）
- インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサでは分岐予測手法を使用して、分岐命令が実行される前に分岐命令のデスティネーションをプリフェッチすることで処理能力を向上させる。それゆえ、分岐命令が実行されるときは、命令実行は必ずしもシリアル化されない。

7.5. マルチプロセッサ（MP）初期化

IA-32 アーキテクチャ（P6 ファミリ・プロセッサから始まる）は、マルチプロセッサ仕様バージョン 1.4 と呼ばれる、マルチプロセッサ（MP）初期化プロトコルを定義している。この仕様は、マルチプロセッサ・システム内の IA-32 プロセッサが使用するブート・プロトコルを定義している（**マルチプロセッサ**は、2 個以上のプロセッサとして定義される）。MP 初期化プロトコルには、次のような重要な特徴がある。

- 専用のシステム・ハードウェアを使用せずに、複数のプロセッサの制御式ブートに対応する。
- 専用の信号や事前に定義されたブート・プロセッサを使用せずに、ハードウェアがシステムのブートを開始できる。
- すべての IA-32 プロセッサ（ハイパー・スレッディング・テクノロジ対応プロセッサを含む）を同じ方法でブートできる。

MP 初期化プロトコルの実行機構は、IA-32 プロセッサ・ファミリに応じて、次のように異なる。

- P6 ファミリ・プロセッサの場合 ― BSP と AP（7.5.1. 項「BSP プロセッサと AP プロセッサ」を参照）の選択は、BIPI メッセージと FIPI メッセージを使用して、APIC バス上でのアービトレーションによって処理される。P6 ファミリ・プロセッサの MP 初期化の詳細については、付録 C を参照のこと。
- インテル® Xeon™ プロセッサ（F09H までのファミリ、モデル、ステッピング ID）の場合 ― BSP と AP（7.5.1. 項「BSP プロセッサと AP プロセッサ」を参照）の選択

は、BIPI メッセージと FIPI メッセージを使用して、システムバス上でのアービトレーションによって処理される。インテル Xeon プロセッサの MP 初期化の詳細については、7.5.3. 項「インテル® Xeon™ プロセッサの MP 初期化プロトコルのアルゴリズム」を参照のこと。

- インテル Xeon プロセッサ（F0AH およびそれ以上のファミリ、モデル、およびステッピング ID）の場合 － BSP と AP の選択は、BIPI メッセージと FIPI メッセージによるアービトレーションを使用せずに、特殊なシステム・バス・サイクルによって処理される。この選択方法については、7.5.3. 項「インテル® Xeon™ プロセッサの MP 初期化プロトコルのアルゴリズム」でも説明する。

プロセッサのファミリ、モデル、ステッピング ID は、EAX レジスタの値を 1 にして CPUID 命令を実行すると、EAX レジスタに返される。

7.5.1. BSP プロセッサと AP プロセッサ

MP 初期化プロトコルは、ブートストラップ・プロセッサ（BSP）とアプリケーション・プロセッサ（AP）の 2 クラスのプロセッサを定義している。MP システムの電源投入またはリセット後、システム・ハードウェアは、システムバス上のプロセッサのうち 1 つを BSP として動的に選択する。その他のプロセッサは、AP として指定される。

BSP 選択機構の一部として、BSP の IA32_APIC_BASE MSR（図 8-5. を参照）内で BSP フラグがセットされ、このプロセッサが BSP であることを示す。他のすべてのプロセッサでは、このフラグはクリアされる。

BSP は、BIOS のブートストラップ・コードを実行して、APIC 環境の設定、システム全体のデータ構造のセットアップ、AP の起動と初期化を行う。BSP と AP の初期化が完了すると、BSP は、オペレーティング・システムの初期化コードの実行を開始する。

電源投入またはリセット後、AP は最小限の自己設定を完了し、BSP プロセッサからのスタートアップ信号（SIPI メッセージ）を待機する。AP は、SIPI メッセージを受信すると、BIOS の AP 設定コードを実行する。このプロセスが完了すると、AP はホルト状態に置かれる。

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサでは、MP 初期化プロトコルは、システムバス上の個々の論理プロセッサを（個別の APIC ID を持つ）別々のプロセッサとして扱う。ブートアップ時に、論理プロセッサのうち 1 つが BSP として選択され、その他の論理プロセッサは AP として指定される。

7.5.2. インテル® Xeon™ プロセッサの MP 初期化プロトコルの必要条件および制約

MP 初期化プロトコルはシステムに対して次の必要条件と制約をかける。

- MP プロトコルは、電源投入またはリセット後にのみ実行される。MP プロトコルが完了し BSP が選択されると、これ以降に（特定のプロセッサに対して、またはシステム全体に対して）INIT が発行されても、MP プロトコルは繰り返し実行されない。この場合、各プロセッサは、（IA32_APIC_BASE MSR 内の）自分の BSP フラグを調べて、BIOS のブートストラップ・コードを実行するか（そのプロセッサが BSP の場合）、SIPI 待機状態に移行するか（AP の場合）を判断する。
- システム内のすべてのデバイスは、MP 初期化プロトコルが完了するまで、プロセッサに対する割り込みを生成できない。この割り込み禁止期間には、BSP が AP に対して INIT-SIPI-SIPI シーケンスを発行してから、AP がシーケンス内の最後の SIPI に応答する時点までの時間幅が含まれる。

7.5.3. インテル® Xeon™ プロセッサの MP 初期化プロトコルのアルゴリズム

MP システムの電源投入またはリセット後、システム内のインテル® Xeon™ プロセッサは、MP 初期化プロトコルのアルゴリズムを実行し、システムバス上の各プロセッサを初期化する。このアルゴリズムの実行の過程で、以下のブートアップ操作と初期化操作が実行される。

1. システム構成に基づいて、システムバス上の各プロセッサに個別の 8 ビット APIC ID が割り当てられる（7.5.5. 項「MP システム内のプロセッサの識別」を参照）。この ID は、各プロセッサのローカル APIC ID レジスタに書き込まれる。
2. APIC ID に基づいて、各プロセッサに個別のアービトレーション優先度が割り当てられる。
3. 各プロセッサが、システムバス上の他のプロセッサと同時に、自分の内部 BIST を実行する。
4. BIST が完了すると、プロセッサは、ハードウェア上で定義された選択機構を使用して、システムバス上の使用可能なプロセッサの中から、BSP と AP を選択する。BSP 選択機構は、プロセッサのファミリ、モデル、ステッピング ID に応じて、次のように異なる。
   - F0AH およびそれ以上のファミリ、モデル、ステッピング ID
     - プロセッサは、トグルする BNR# 信号の監視を開始する。BNR# ピンがトグルを停止すると、各プロセッサは、システムバス上に NOP 特殊サイクルを発行しようとする。
     - 最高のアービトレーション優先度を持つプロセッサが、NOP 特殊サイクルの発行に成功し、BSP として指定される。このプロセッサは、自分の IA32_APIC_BASE MSR 内の BSP フラグをセットし、リセットベクタ（物理ア

ドレス FFFF FFF0H）を始点とする BIOS のブートストラップ・コードをフェッチして実行を開始する。

- その他のプロセッサ（NOP 特殊サイクルを発行できなかったプロセッサ）は、AP として指定される。これらのプロセッサは、BSP フラグをクリア状態のままにして、「SIPI 待機状態」に移行する。

— F09H までのファミリ、モデル、ステッピング ID

- 各プロセッサは、「自分自身を含むすべてのプロセッサ」に BIPI をブロードキャストする。最初に BIPI をブロードキャストしたプロセッサ（したがって、自分の BIPI ベクタを受信したプロセッサ）が、自分自身を BSP として選択し、IA32_APIC_BASE MSR 内の BSP フラグをセットする（BIPI、FIPI、SIPI メッセージについては、C.1. 節「P6 ファミリ・プロセッサの MP 初期化プロセスの 概要」を参照）。
- その他のプロセッサ（BSP として選択されなかったプロセッサ）は、AP として指定される。これらのプロセッサは、BSP フラグをクリア状態のままにして、「SIPI 待機状態」に移行する。
- 新たに選択された BSP は、「自分自身を含むすべてのプロセッサ」に FIPI メッセージをブロードキャストする。BSP と AP は、このメッセージを、MP 初期化の終了を示す信号として扱う。BSP フラグがセットされているプロセッサだけが、この FIPI メッセージに応答する。このプロセッサは、メッセージへの応答として、リセットベクタ（物理アドレス FFFF FFF0H）を始点とする BIOS のブートストラップ・コードをフェッチして実行する。

5. ブートストラップ・コードの一部として BSP は、ACPI テーブルと MP テーブルを作成し、自分の初期 APIC ID をこれらのテーブルに追加する。
6. ブートストラップ・プロシージャの最後に、BSP はプロセッサ・カウンタを 1 に設定し、システム内のすべての AP に SIPI メッセージをブロードキャストする。この SIPI メッセージには、000VV000H の位置にある BIOS の AP 初期化コードへのベクタが含まれる（VV は SIPI メッセージに含まれるベクタ）。
7. AP 初期化コードの最初の動作は、BIOS の初期化セマフォに対する（AP 間の）競争をセットアップすることである。最初にセマフォにアクセスした AP が、初期化コードの実行を開始する（セマフォの使用の詳細については、7.5.4. 項「MP 初期化の例」を参照）。AP 初期化プロシージャの一部として、この AP は、自分の APIC ID 番号を ACPI テーブルと MP テーブルに追加し、プロセッサ・カウンタを 1 だけインクリメントする。この初期化プロシージャが完了すると、この AP は CLI 命令を実行し、ホルト状態に移行する。
8. すべての AP がセマフォにアクセスし、AP 初期化コードを実行すると、BSP は、システムバスに接続されているプロセッサの数に合わせてカウントを設定し、BIOS のブートストラップ・コードの実行を完了する。次に、BSP は、オペレーティング・システムのブートストラップおよびスタートアップ・コードの実行を開始する。

9. BSP がオペレーティング・システムのブートストラップおよびスタートアップ・コードを実行している間、AP はホルト状態のままになる。ホルト状態の AP は、INIT、NMI、SMI にのみ応答する。また、これらのプロセッサは、スヌープと STPCLK# ピンのアサートにも応答する。

次の項では、MP 構成で動作する複数のインテル Xeon プロセッサの MP 初期化プロトコルの例（コードを含む）を示す。

付録 B「モデル固有レジスタ（MSR）」では、MP 設定の完了後にプロセッサのローカル APIC の LINT[0:1] ピンをプログラムする方法について説明する。

7.5.4. MP 初期化の例

次の例は、BSP と AP が選択された後に MP システム内の IA-32 プロセッサを初期化する、MP 初期化プロトコルの使用方法を示している。このコードは、MP 初期化プロトコルを使用するすべての IA-32 プロセッサ上で正常に動作する。これらのプロセッサには、P6 ファミリ・プロセッサとインテル Xeon プロセッサ（ハイパー・スレッディング・テクノロジ対応プロセッサと非対応プロセッサの両方）が含まれる。

記載されているコード例では、以下の定数とデータ定義を使用する。これらは、表 8-1. で定義される APIC レジスタのアドレスに基づいている。

```
ICR_LOW         EQU 0FEE00300H
SVR             EQU 0FEE000F0H
APIC_ID         EQU 0FEE00020H
LVT3            EQU 0FEE00370H
APIC_ENABLED    EQU 0100H
BOOT_ID         DD ?
COUNT           EQU 00H
VACANT          EQU 00H
```

7.5.4.1. 一般的な BSP 初期化シーケンス

ハードウェア・プロトコルによって BSP と AP が選択された後（7.5.3. 項「インテル® Xeon™ プロセッサの MP 初期化プロトコルのアルゴリズム」を参照）、BSP は、通常の IA-32 アーキテクチャの開始アドレス（FFFFFFF0H）から、BIOS のブートストラップ・コード（POST）の実行を開始する。ブートストラップ・コードは、通常は以下の操作を実行する。

1. メモリを初期化する。
2. マイクロコード・アップデートをプロセッサにロードする。

3. MTRR を初期化する。
4. キャッシュを有効にする。
5. EAX レジスタの値を 0H にして CPUID 命令を実行した後、EBX、ECX、EDX レジスタを読み出し、BSP が "GenuineIntel" かどうかを確認する。
6. EAX レジスタの値を 1H にして CPUID 命令を実行した後、EBX、ECX、EDX レジスタの値を、後で使用するために RAM 内のシステム設定空間に保存する。
7. メモリの下位 1M バイト内の 4K バイト・ページ内で実行される、AP のスタートアップ・コードをロードする。
8. 保護モードに切り替えて、APIC のアドレス空間がストロング・キャッシュ不可（UC）メモリタイプにマッピングされていることを確認する。
9. ローカル APIC ID レジスタから、BSP の APIC ID を確認する（デフォルトでは 0）。

```
MOV ESI, APIC_ID       ; address of local APIC ID register
MOV EAX, [ESI]
AND EAX, 0FF000000H    ; zero out all other bits except APIC ID
MOV BOOT_ID, EAX       ; save in memory
```

   この APIC ID を ACPI テーブルと MP テーブルに保存する。オプションにより、RAM 内のシステム設定空間にも保存する。

10. AP のブートアップ・コード用の 4K バイト・ページのベースアドレスを、8 ビット・ベクタに変換する。この 8 ビット・ベクタは、実アドレスモードのアドレス空間（1M バイト空間）内の 4K バイト・ページのアドレスを定義する。例えば、値が 0BDH のベクタは、000BD000H のスタートアップ・メモリ・アドレスを指定する。
11. APIC スプリアス・ベクタ・レジスタ（SVR）のビット 8 をセットして、ローカル APIC を有効にする。

```
MOV ESI, SVR           ; address of SVR
MOV EAX, [ESI]
OR  EAX, APIC_ENABLED ; set bit 8 to enable (0 on reset)
MOV [ESI], EAX
```

12. APIC エラーハンドラの 8 ビット・ベクタを設定して、LVT のエラー処理エントリをセットアップする。

```
MOV ESI, LVT3
MOV EAX, [ESI]
AND EAX, FFFFFF00H     ; clear out previous vector
OR EAX, 000000xxH      ; xx is the 8-bit vector the APIC error
                       ; handler.
MOV [ESI], EAX
```

13. ロックセマフォ変数 VACANT を 00H に初期化する。AP は、このセマフォを使用して、BIOS の AP 初期化コードを実行する順番を決める。
14. 以下の操作を実行して、システム内の AP の存在とプロセッサ数を検出するように、BSP をセットアップする。

- COUNT 変数の値を 1 に設定する。
- （約 100msec の間隔に設定された）タイマをスタートする。BIOS の AP 初期化コード内で、各 AP は、COUNT 変数をインクリメントして自分が存在することを示す。タイマの設定時間が経過すると、BSP は COUNT 変数の値をチェックする。タイマの設定時間が経過したときに COUNT 変数がインクリメントされていない場合は、AP が存在しないか、何らかのエラーが発生している。

15. INIT-SIPI-SIPI IPI シーケンスを AP にブロードキャストして、AP を起動し、初期化する。

```
MOV ESI, ICR_LOW      ; load address of ICR low dword into ESI
MOV EAX, 000C4500H    ; load ICR encoding for broadcast INIT IPI
                      ; to all APs into EAX
MOV [ESI], EAX        ; broadcast INIT IPI to all APs
                      ; 10-millisecond delay loop
MOV EAX, 000C46XXH    ; load ICR encoding for broadcast SIPI IPI
                      ; to all APs into EAX, where xx is the
                      ; vector value computed in step 8.
MOV [ESI], EAX        ; broadcast SIPI IPI to all APs
                      ; 200-microsecond delay loop
MOV [ESI], EAX        ; broadcast second SIPI IPI to all APs
                      ; 200-microsecond delay loop
```

16. タイマの割り込みを待機する。
17. COUNT 変数を読み出して評価し、プロセッサ数を設定する。
18. 必要に応じて、APIC を再設定し、その他のシステム診断に進む。

7.5.4.2. 一般的な AP 初期化シーケンス

AP は、SIPI を受信すると、SIPI 内にコード化されたベクタの位置にある、BIOS の AP 初期化コードの実行を開始する。AP 初期化コードは、通常は以下の操作を実行する。

1. BIOS の初期化ロックセマフォを待機する。セマフォの制御が得られたら、初期化を続行する。
2. マイクロコード・アップデートをプロセッサにロードする。
3. MTRR を初期化する（BSP に使用したのと同じマッピングを使用する）。
4. キャッシュを有効にする。
5. EAX レジスタの値を 0H にして CPUID 命令を実行した後、EBX、ECX、EDX レジスタを読み出し、AP が "GenuineIntel" かどうかを確認する。
6. EAX レジスタの値を 1H にして CPUID 命令を実行した後、EBX、ECX、EDX レジスタの値を、後で使用するために RAM 内のシステム設定空間に保存する。
7. 保護モードに切り替えて、APIC のアドレス空間がストロング・キャッシュ不可（UC）メモリタイプにマッピングされていることを確認する。

8. ローカル APIC ID レジスタから、この AP の APIC ID を確認し、その ID を MP テーブルと ACPI テーブルに保存する。オプションにより、RAM 内のシステム設定空間にも保存する。
9. SVR レジスタのビット 8 をセットし、エラー処理用の LVT3（エラー LVT）をセットアップすることにより、ローカル APIC の初期化と設定を行う（7.5.4.1. 項「一般的な BSP 初期化シーケンス」の手順 9 と手順 10 を参照）。
10. AP の SMI 実行環境を設定する（各 AP と BSP は、異なる SMBASE アドレスを持っていなければならない）。
11. COUNT 変数を 1 だけインクリメントする。
12. セマフォを解放する。
13. CLI 命令と HLT 命令を実行する。
14. INIT IPI を待機する。

7.5.5. MP システム内のプロセッサの識別

BIOS が MP 初期化プロトコルを完了すると、各プロセッサは、それぞれのローカル APIC ID によって個別に識別可能になる。ソフトウェアは、次のいずれかの方法で、これらの APIC ID にアクセスできる。

- **ローカル APIC の APIC ID を読み出す。**プロセッサ上で実行されるコードは、MOV 命令を実行して、そのプロセッサのローカル APIC ID レジスタの内容を読み出せる（8.4.6. 項「ローカル APIC ID」を参照）。
- **ACPI テーブルまたは MP テーブルを読み出す。**MP 初期化プロトコルの一部として、BIOS は、ACPI テーブルと MP テーブルを作成する。これらのテーブルは、マルチプロセッサ仕様バージョン 1.4 で定義されており、システム内のプロセッサのリストと各プロセッサのローカル APIC ID をソフトウェアに提供する。ACPI テーブルの形式は、ACPI 仕様に基づいている。ACPI 仕様は、業界で広く使われている、MP システムのパワー・マネージメントおよびプラットフォーム構成の仕様である。

インテル® Xeon™ プロセッサでは、電源投入および初期化の際にプロセッサに割り当てられる APIC ID は 8 ビットである（図 7-2. を参照）。この場合、ビット 1 とビット 2 は、2 ビットのプロセッサ識別子になる（これはソケット識別子とも見なせる）。プロセッサをクラスタ構成にしているシステムでは、ビット 3 とビット 4 が 2 ビットのクラスタ ID になる。ビット 0 は、インテル Xeon プロセッサ MP 内で、プロセッサ・パッケージ内の 2 つの論理プロセッサの識別に使用される（7.7.5. 項「MP システム内の論理プロセッサの識別」を参照）。ハイパー・スレッディング・テクノロジ非対応のインテル Xeon プロセッサでは、ビット 0 は常に 0 に設定される。ハイパー・スレッ

ディング・テクノロジ対応インテル Xeon プロセッサでは、ビット 0 は、インテル Xeon プロセッサ MP の場合と同じように機能する。

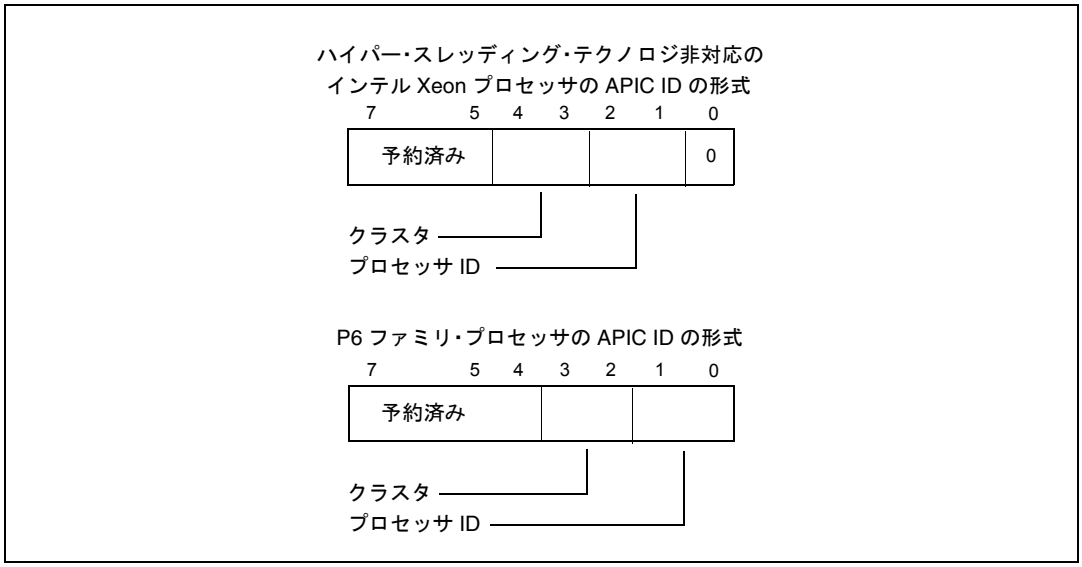


図 7-2. MP システム内の APIC ID の解釈

P6 ファミリ・プロセッサでは、電源投入および初期化の際にプロセッサに割り当てられる APIC ID は 4 ビットである（図 7-2. を参照）。この場合、ビット 0 とビット 1 は 2 ビットのプロセッサ（またはソケット）識別子になり、ビット 2 とビット 3 は 2 ビットのクラスタ ID になる。

7.6. ハイパー・スレッディング・テクノロジ

ハイパー・スレッディング（HT）テクノロジは、インテル® Xeon™ プロセッサ MP とインテル Xeon プロセッサの最近のステッピングで、IA-32 アーキテクチャに導入された。ハイパー・スレッディング・テクノロジは、ハイパー・スレッディング・テクノロジ対応のすべてのインテル® Pentium® 4 プロセッサでもサポートされている。すべてのハイパー・スレッディング・テクノロジ構成には、ハイパー・スレッディング・テクノロジを活用するチップセットおよび BIOS と、ハイパー・スレッディング・テクノロジ向けの最適化を組み込んだオペレーティング・システムが必要である。詳細については、http://www.intel.co.jp/jp/info/hyperthreading/ を参照のこと。また、上巻の 2.2.4. 項「ハイパー・スレッディング・テクノロジ」も参照のこと。

インテルでは、ソフトウェアがプロセッサ上のハイパー・スレッディング・テクノロジのサポートの有無を確認する場合、IA-32 プロセッサの名称に依存しないことを推

奨する。ソフトウェアは、CPUID 命令を使用しなければならない（7.6.3. 項「ハイパー・スレッディング・テクノロジの検出」を参照）。

ハイパー・スレッディング・テクノロジは、IA-32 アーキテクチャを拡張する技術であり、1 つの物理プロセッサが（スレッドと呼ばれる）2 つ以上の別々のコード・ストリームを同時に実行できるようにする。以下の各項では、IA-32 プロセッサ内でのその機能の実装方法について説明する。

7.6.1. ハイパー・スレッディング・テクノロジのアーキテクチャ

図 7-3. は、インテル® Xeon™ プロセッサ MP を例として、ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサの一般的な図を示している。ここでは、ハイパー・スレッディング・テクノロジは、（それぞれ別々の IA-32 アーキテクチャ・ステートによって表される）2 つの論理プロセッサによって実現される。これらの論理プロセッサは、物理プロセッサの実行エンジンとバス・インターフェイスを共有する。各論理プロセッサは、それぞれのアドバンスト・プログラマブル割り込みコントローラ（APIC）を持つ。

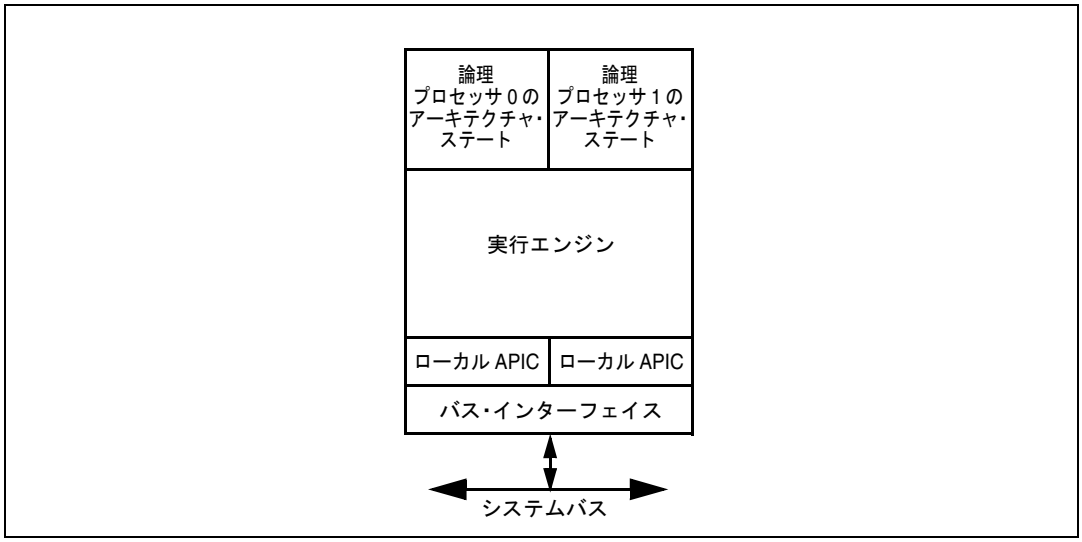


図 7-3. 2 つの論理プロセッサを使用する、ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサ

7.6.1.1. 論理プロセッサのステート

以下の機能は、ハイパー・スレッディング・テクノロジをサポートするIA-32プロセッサでの論理プロセッサのアーキテクチャ・ステートの一部である。これらの機能は、以下の3つのグループに分けられる。

- 各論理プロセッサに対して複製される機能
- 物理プロセッサ内の論理プロセッサによって共有される機能
- プロセッサに応じて、共有されたり複製されたりする機能

以下の機能は、各論理プロセッサに対して複製される。

- 汎用レジスタ（EAX、EBX、ECX、EDX、ESI、EDI、ESP、EBP）
- セグメント・レジスタ（CS、DS、SS、ES、FS、GS）
- EFLAGS レジスタと EIP レジスタ。各論理プロセッサの CS レジスタと EIP レジスタは、その論理プロセッサが実行しているスレッドの命令ストリームを指す。
- x87 FPU レジスタ（ST0 ～ ST7、ステータス・ワード、制御ワード、タグワード、データ・オペランド・ポインタ、命令ポインタ）
- MMX命令レジスタ（MM0 ～ MM7）
- XMM レジスタ（XMM0 ～ XMM7）と MXCSR レジスタ
- 制御レジスタ（CR0、CR2、CR3、CR4）とシステム・テーブル・ポインタ・レジスタ（GDTR、LDTR、IDTR、タスクレジスタ）
- デバッグレジスタ（DR0、DR1、DR2、DR3、DR6、DR7）とデバッグ制御　MSR（IA32_DEBUGCTL）
- マシン・チェック・グローバル・ステータス（IA32_MCG_STATUS）MSR とマシンチェック機能（IA32_MCG_CAP）MSR
- 温度クロック変調MSR と ACPI パワー・マネージメント制御MSR
- タイムスタンプ・カウンタ MSR
- その他のMSR レジスタの大部分（ページ属性テーブル（PAT）を含む）。以下の例外事項を参照のこと。
- ローカルAPIC レジスタ

以下の機能は、論理プロセッサによって共有される。

- IA32_MISC_ENABLE MSR（MSR アドレス 1A0H）
- メモリタイプ範囲レジスタ（MTRR）

以下の機能が複製されるか共有されるかは、プロセッサによって異なる。

- マシン・チェック・アーキテクチャ（MCA）MSR（IA32_MCG_STATUS MSR と IA32_MCG_CAP MSR を除く）
- 性能モニタリング制御 MSR と性能モニタリング・カウンタ MSR

7.6.1.2. APIC の機能

ハイパー・スレッディング・テクノロジ対応プロセッサの初期化の際に、各論理プロセッサにローカル APIC ID が割り当てられる（表 8-1. を参照）。このローカル APIC ID は、各論理プロセッサの ID とし、その論理プロセッサの APIC ID レジスタに格納される。デュアルプロセッサ（DP）または MP システム内に、ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサが 2 個以上ある場合は、システムバス上の各論理プロセッサに、個別のローカル APIC ID が割り当てられる（7.7.5. 項「MP システム内の論理プロセッサの識別」を参照）。

ソフトウェアは、APIC のプロセッサ間割り込み（IPI）メッセージング機能を使用して、ローカル・プロセッサと通信する。ハイパー・スレッディング・テクノロジ対応プロセッサ内のローカル APIC のセットアップとプログラミングは、ハイパー・スレッディング・テクノロジ非対応の IA-32 プロセッサの場合と同じである。詳細については、第 8 章「アドバンスド・プログラマブル 割り込みコントローラ（APIC）」を参照のこと。

7.6.1.3. メモリタイプ範囲レジスタ（MTRR）

ハイパー・スレッディング・テクノロジ対応プロセッサ内の MTRR は、論理プロセッサによって共有される。1 つの論理プロセッサが MTRR の設定を更新した場合、その設定は、同じ物理パッケージ内の他の論理プロセッサと自動的に共有される。

IA-32 アーキテクチャでは、IA-32 プロセッサ（論理プロセッサを含む）に基づくすべての MP システムが同じ MTRR メモリマップを使用しなければならない。これにより、ソフトウェアは、どのプロセッサ上で実行されているかに関係なく、同じようにメモリを認識できる。MTRR の設定については、10.11. 節「メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）」を参照のこと。

7.6.1.4. ページ属性テーブル（PAT）

各論理プロセッサは、それぞれの PAT MSR（IA32_CR_PAT）を持つ。ただし、10.12. 節「ページ属性テーブル（PAT）」で説明するように、PAT MSR の設定は、（論理プロセッサを含む）システム内のすべてのプロセッサで同一でなければならない。

7.6.1.5. マシン・チェック・アーキテクチャ

HT テクノロジ・コンテキスト内では、すべてのマシン・チェック・アーキテクチャ（MCA）MSR（IA32_MCG_STATUS MSR や IA32_MCG_CAP MSR など）は、各論理プロセッサに対して複製される。これにより、同じ物理プロセッサ内の論理プロセッサが、マシンチェック例外の初期化、設定、クエリ、処理を同時に実行できる。この設計は、第 14 章「マシン・チェック・アーキテクチャ」で説明するガイドラインに従ったマシンチェック例外ハンドラと適合性がある。

IA32_MCG_STATUS MSR が各論理プロセッサに対して複製されるため、この MSR 内の「進行中のマシンチェック（MCIP）」ビット・フィールドを使用して、MCA ハンドラ側の再帰を検出できる。さらに、MSR により、各論理プロセッサは、同じ物理パッケージ内のもう 1 つの論理プロセッサの動作とは無関係に、マシンチェック例外が進行中かどうかを確認できる。

1 つの物理パッケージ内の 2 つの論理プロセッサは、ハードウェア・リソースの共有によって密接に結合されている。このため、特定の物理プロセッサ内で発生したマシン・チェック・エラーは、両方の論理プロセッサに通知される。マシンチェック例外が有効になっているときに致命的エラーが報告されると、物理パッケージ内のすべての論理プロセッサの状態は、マシンチェック例外ハンドラに送られる。マシンチェック例外が無効になっている場合は、論理プロセッサはシャットダウン状態に移行し、IERR#信号をアサートする。

マシンチェック例外を有効にすると、各論理プロセッサの制御レジスタ CR4 の MCE フラグがセットされる。

7.6.1.6. デバッグレジスタと拡張機能

各論理プロセッサは、一連のデバッグレジスタ（DR0、DR1、DR2、DR3、DR6、DR7）とデバッグ制御 MSR（IA32_DEBUGCTL）を持つ。これらのレジスタを設定して、各論理プロセッサのデバッグ情報を個別に記録し、デバッグ方法を個別に制御できる。各論理プロセッサは、それぞれの最終分岐レコード（LBR）スタックを持つ。

7.6.1.7. 性能モニタリング・カウンタ

性能カウンタとそれに関連する制御 MSR は、物理プロセッサ内の論理プロセッサの間で共有される。したがって、ソフトウェアがこれらのリソースの使用を管理しなければならない。性能カウンタの割り込み、イベント、正確なイベント監視機能は、スレッドごと（論理プロセッサごと）に設定し、割り当てられる。

インテル® Xeon™ プロセッサ MP の性能モニタリングについては、15.10. 節「性能モニタリングとハイパー・スレッディング・テクノロジ」を参照のこと。

7.6.1.8. IA32_MISC_ENABLE MSR

IA32_MISC_ENABLE MSR（MSRアドレス1A0H）は、HTテクノロジ対応IA-32プロセッサ内の論理プロセッサの間で共有される。したがって、このレジスタが制御するアーキテクチャ機能は、同じ物理パッケージ内のすべての論理プロセッサに対して同じ内容に設定される。

7.6.1.9. メモリ・オーダリング

メモリ・オーダリングについては、HTテクノロジ対応IA-32プロセッサ内の論理プロセッサは、HTテクノロジ非対応のIA-32プロセッサの場合と同じ規則に従う（7.2.節「メモリ・オーダリング」を参照）。各論理プロセッサは、「ストア・バッファ・フォワーディングによるライト・オーダード」として定義されるプロセッサ・オーダード・メモリ・モデルを使用する。特殊なプログラミング条件を扱うためにメモリ・オーダリング・モデルを強めるかまたは弱めるすべての機構は、各論理プロセッサに適用される。

7.6.1.10. シリアル化命令

一般的な規則として、HTテクノロジ対応IA-32プロセッサ内の1つの論理プロセッサがシリアル化命令を実行する場合、その論理プロセッサだけがその操作の影響を受ける。ただし、WBINVD、INVD、WRMSR命令を実行する場合と、制御レジスタCR0のCDフラグの状態が変更されたときにMOV CR命令を実行する場合は、この規則は適用されず、両方の論理プロセッサがシリアル化される。

7.6.1.11. マイクロコード・アップデートのリソース

HTテクノロジ対応IA-32プロセッサ内では、マイクロコード・アップデートの機能は論理プロセッサの間で共有され、いずれの論理プロセッサもアップデートを開始できる。各論理プロセッサは、それぞれのBIOSシグネチャMSR IA32_BIOS_SIGN_ID（MSRアドレス8BH）を持つ。1つの論理プロセッサが物理プロセッサのアップデートを実行すると、存在する論理プロセッサのIA32_BIOS_SIGN_ID MSRが同じ情報で更新される。論理プロセッサが同時にアップデートを開始した場合は、アップデートが一度に1つだけ実行されるように、プロセッサ・コアが必要な同期をとる。

オペレーティング・システムのマイクロコード・アップデート・ドライバは、インテルのアップデート・ガイドラインに適合していれば、HTテクノロジ対応IA-32プロセッサ上で、修正の必要なしに実行できる。

7.6.1.12. 自己修正コード

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサは、自己修正コードをサポートしている。自己修正コードでは、データの書き込みによって、キャッシュに蓄えられた命令または現在実行中の命令が修正される。これらのプロセッサは、クロス修正コードもサポートしている。クロス修正コードでは、MP システム上で、1 つのプロセッサが生成した書き込みによって、別のプロセッサ上でキャッシュに蓄えられた命令または現在実行中の命令が修正される。IA-32 プロセッサ内の自己修正コードとクロス修正コードの必要条件については、7.1.3. 項「自己修正コードおよびクロス修正コードの処理」を参照のこと。

7.6.2. プロセッサ固有の HT テクノロジ機能

HT テクノロジ対応 IA-32 プロセッサでは、以下の非アーキテクチャ機能は、プロセッサによって異なる。

- キャッシュ
- トランスレーション・ルックアサイド・バッファ（TLB）
- 温度モニタリング機能

以下の各項では、インテル® Xeon™ プロセッサ MP に導入された機能について説明する。

7.6.2.1. プロセッサ・キャッシュ

インテル® Xeon™ プロセッサ MP の場合、キャッシュは共有される。1 つの論理プロセッサ上で実行されるすべてのキャッシュ操作命令は、以下に注意する。物理プロセッサのキャッシュ階層にグローバルな影響を与える。

- **WBINVD 命令。**修正されたデータをメモリに書き戻した後、キャッシュ階層全体を無効化する。書き戻しと無効化の操作が完了するまで、すべての論理プロセッサの実行は停止される。特殊なバスサイクルが、すべてのキャッシュ・エージェントに送信される。
- **INVD 命令。**修正されたデータをメモリに書き戻さずに、キャッシュ階層全体を無効化する。無効化操作が完了するまで、すべての論理プロセッサの実行は停止される。特殊なバスサイクルが、すべてのキャッシュ・エージェントに送信される。
- **CLFLUSH　命令。**修正されたデータをメモリに書き戻し、すべてのキャッシュ・エージェントにバスサイクルを送信した後、（どの論理プロセッサによってキャッシュ・ラインが充填されたかに関係なく）キャッシュ階層内の指定したキャッシュ・ラインを無効化する。

- **制御レジスタ CR0 内の CD フラグ。**各論理プロセッサは、それぞれの CR0 制御レジスタを（したがって、CR0 内の CD フラグを）持つ。2 つの論理プロセッサの CD フラグは、OR 演算される。これにより、いずれかの論理プロセッサが自分の CD フラグをセットすると、キャッシュ全体が名目上無効にされる。

7.6.2.2. プロセッサのトランスレーション・ルックアサイド・バッファ（TLB）

インテル® Xeon™ プロセッサ MP では、データ・キャッシュの TLB は共有される。命令キャッシュの TLB は各論理プロセッサ内で複製される。

TLB 内のエントリには、変換を開始した論理プロセッサを示す ID のタグが付けられる。このタグは、メモリ・ページングにページ・グローバル機能を使用する、グローバルとしてマークされた変換に対しても適用される。

1 つの論理プロセッサが TLB 無効化操作を実行すると、その論理プロセッサのタグが付けられた TLB エントリだけがフラッシュされる。この規則は、すべての TLB 無効化操作に適用される。これには、制御レジスタ CR3 および CR4 への書き込みと INVLPG 命令の使用が含まれる。

7.6.2.3. 温度モニタ

インテル® Xeon™ プロセッサ MP 内では、各論理プロセッサは、致命的シャットダウン・ディテクタと自動温度監視機構を共有する（13.16. 節「温度監視と保護」を参照）。共有の結果、以下の動作が行われる。

- プロセッサ・コアの温度が、設定された致命的シャットダウン温度を超えた場合は、プロセッサ・コアは実行を停止する。その結果、両方の論理プロセッサも実行を停止する。
- プロセッサ・コアの温度が、設定された自動温度モニタトリップ温度を超えた場合は、プロセッサ・コアのクロックスピードが自動的に変調される。その結果、両方の論理プロセッサの実行速度も影響を受ける。

ソフトウェアで制御されるクロック変調では、各論理プロセッサはそれぞれの IA32_CLOCK_MODULATION_MSR を持つため論理プロセッサごとに、クロック変調を有効または無効に設定できる。通常は、ソフトウェアで制御されるクロック変調を使用する場合は、物理プロセッサ内のすべての論理プロセッサのクロック変調を有効にし、各論理プロセッサの変調デューティ・サイクルを同じ値に設定しなければならない。論理プロセッサの間でデューティ・サイクルの値が異なる場合は、プロセッサのクロックは、選択されたデューティ・サイクルの中で最も高い値に合わせて変調される。

7.6.2.4. 外部信号の適合性

この項では、インテル® Xeon™ プロセッサ MP のピンを介して受信する外部信号に関する制約と、これらの信号を論理プロセッサの間で共有する方法について説明する。

- **STPCLK#。**インテル Xeon プロセッサ MP の物理パッケージには、1 つの STPCLK# ピンが用意されている。外部の制御ロジックは、システムのパワー・マネージメント用にこのピンを使用する。STPCLK# 信号がアサートされると、プロセッサ・コアはストップグラント状態に移行する。この状態では、命令の実行は停止されるが、プロセッサ・コアはスヌープ・トランザクションへの応答を続ける。STPCLK# 信号がアサートされた時点で論理プロセッサがアクティブであるかホルト状態であるかに関係なく、両方の論理プロセッサで実行が停止され、割り込みにも応答しなくなる。
  MP システム内では、すべての物理プロセッサ上の STPCLK# ピンは、通常は互いに結合されている。その結果、この信号は、システム内のすべての論理プロセッサに同時に影響を与える。
- **LINT0 ピンと LINT1 ピン。**インテル Xeon プロセッサ MP は、1 組の LINT0 ピンと LINT1 ピンを持ち、それを論理プロセッサの間で共有する。これらのピンのうち 1 つがアサートされると、一方または両方の論理プロセッサの APIC ローカル・ベクタ・テーブル内でそのピンがマスクされていない限り、両方の論理プロセッサが応答する。
  MP システムでは、論理プロセッサに割り込みを送信する場合、通常は LINT0 ピンと LINT1 ピンを使用しない。すべての割り込みは、I/O APIC を介して論理プロセッサに伝えられる。
- **A20M# ピン。**IA-32 プロセッサ上には、インテル® 286 プロセッサとの互換性を保つために、通常は A20M# ピンが用意されている。このピンをアサートすると、すべての外部バス・メモリ・アクセスについて、物理アドレスのビット 20 がマスクされる（強制的に 0 に設定される）。インテル Xeon プロセッサ MP は、1 つの A20M# ピンを搭載している。このピンは、物理プロセッサ内の両方の論理プロセッサの動作に影響を与える。この構成は、IA-32 アーキテクチャと互換性がある。

7.6.3. ハイパー・スレッディング・テクノロジの検出

ソフトウェアは、CPUID 命令を使用して、IA-32 プロセッサ上でのハイパー・スレッディング・テクノロジのサポートの有無と、テクノロジの構成を検出できる。EAX レジスタの入力値を 1 にして CPUID 命令を実行すると、次の 2 つの項目によって、ハイパー・スレッディング・テクノロジの可用性が報告される。

- ハイパー・スレッディング・テクノロジ機能フラグ（EDX レジスタのビット 28）がセットされている場合は、その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できる。

- EBX レジスタのビット 16 ～ 23 は、物理パッケージ内でサポートされる論理プロセッサの数を示す。

パッケージ内で使用可能な論理プロセッサが 1 つしかないときに、CPUID 命令で機能フラグがセットされていて、プロセッサがハイパー・スレッディング・テクノロジをサポートできることを示すときがある。この場合は、EBX レジスタのビット 16 ～ 23 の値が 1 になっている。

7.6.3.1. MONITOR/MWAIT 命令のサポートの検出

ストリーミング SIMD 拡張命令 3 では、マルチスレッド・ソフトウェアでスレッドの同期化を向上するのに役立つ 2 つの命令（MONITOR と MWAIT）が導入された。最初の実装では、ソフトウェアはリング 0 で MONITOR と MWAIT を利用できる。これらの命令は、0 より大きいレベルでは条件付きで利用可能である。以下の手順にしたがって、MONITOR と MWAIT が利用可能かどうかを検出できる。

- CPUID の機能 1H を使用して、MONITOR ビット（ECX[3]）をクエリする。このビットの値が 1 の場合は、MONITOR と MWAIT はリング 0 でサポートされている。
- CPUID の機能 1H が ECX[3]=1 を報告した場合は、TRY/EXCEPT 例外ハンドラ内で MONITOR を実行し、例外をトラップする。例外が発生した場合は、MONITOR と MWAIT は 0 より大きい特権レベルではサポートされていない。例 7-1. を参照のこと。

例 7-1. MONITOR/MWAIT のサポートの確認

```
boolean MONITOR_MWAIT_works = TRUE;
try {
        _asm {
           xor ecx, ecx
           xor edx, edx
           mov eax, MemArea
           monitor
           }
        // Use monitor
} except (UNWIND) {
        // if we get here, MONITOR/MWAIT is not supported
   MONITOR_MWAIT_works = FALSE;
}
```

7.6.4. ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサの初期化

ハイパー・スレッディング・テクノロジをサポートする IA-32 プロセッサを使用する MP システムの初期化プロセスは、従来の MP システムについて説明したプロセスと同

じである（7.5. 節「マルチプロセッサ（MP）初期化」を参照）。システム内の論理プロセッサのうち 1 つが BSP として選択され、その他のプロセッサ（すなわち論理プロセッサ）は AP として指定される。初期化プロセスは、7.5.3. 項「インテル® Xeon™ プロセッサの MP 初期化プロトコルのアルゴリズム」と 7.5.4. 項「MP 初期化の例」で説明したプロセスと同じである。

この初期化プロシージャの一部として、各論理プロセッサに、APIC ID が自動的に割り当てられる。この ID は、各論理プロセッサのローカル APIC ID レジスタに格納される。ハイパー・スレッディング・テクノロジ対応プロセッサがシステム内に 2 個以上ある場合は、システムバス上の各論理プロセッサに、個別の ID が割り当てられる（7.7.5. 項「MP システム内の論理プロセッサの識別」を参照）。

論理プロセッサが APIC ID を一度持つと、ソフトウェアは、APIC IPI メッセージを送信することによって、論理プロセッサと通信できる。

7.6.5. ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサ上の複数のスレッドの実行

オペレーティング・システムのブートアップ・プロシージャが完了すると、ブートストラップ・プロセッサ（BSP）はオペレーティング・システム・コードの実行を続けるが、システム内の他の論理プロセッサはホルト状態に移行する。ホルト状態の論理プロセッサのうち 1 つがコード・ストリーム（スレッド）を実行するには、オペレーティング・システムが、その論理プロセッサに対してプロセッサ間割り込み（IPI）を発行しなければならない。IPI に応答して、ホルト状態の論理プロセッサが起動し、IPI の一部として受信した割り込みベクタで識別されるスレッドの実行を開始する。ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサ内のすべての論理プロセッサがスレッドを実行している場合は、コア実行エンジンがアクティブ・スレッドに対する命令ストリームを同時に実行する。共有される実行リソースは「必要に応じて」アクティブな論理プロセッサに割り当てられる。

論理プロセッサ上で複数のスレッドが実行される場合、それを管理するために、オペレーティング・システムは、従来のシンメトリック・マルチプロセシング（SMP）技法を使用できる。例えば、オペレーティング・システムは、タイムスライスなどのロードバランス機構を使用して、アクティブな論理プロセッサのそれぞれに定期的に割り込みをかけられる。オペレーティング・システムは、論理プロセッサに割り込みをかけると、実行キュー内に実行待ちのスレッドがないかどうかチェックし、割り込みをかけられた論理プロセッサにそのスレッドを発行する。この方法で、MP に対応するオペレーティング・システムは、従来の MP システム内のプロセッサと同じ方法で、論理プロセッサ上でのスレッドの実行をスケジューリングできる。

7.6.6. ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサ上の割り込みの処理

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサでは、割り込みは、従来の MP システムと同じ方法で処理される。I/O APIC が外部割り込みを受信し、個々の論理プロセッサに対して割り込みメッセージとして分散する（図 7-4. を参照）。また、各論理プロセッサは、自分のローカル APIC の ICR レジスタに書き込めば、他の論理プロセッサに IPI を送信できる（8.6. 節「プロセッサ間割り込みの発行」を参照）。

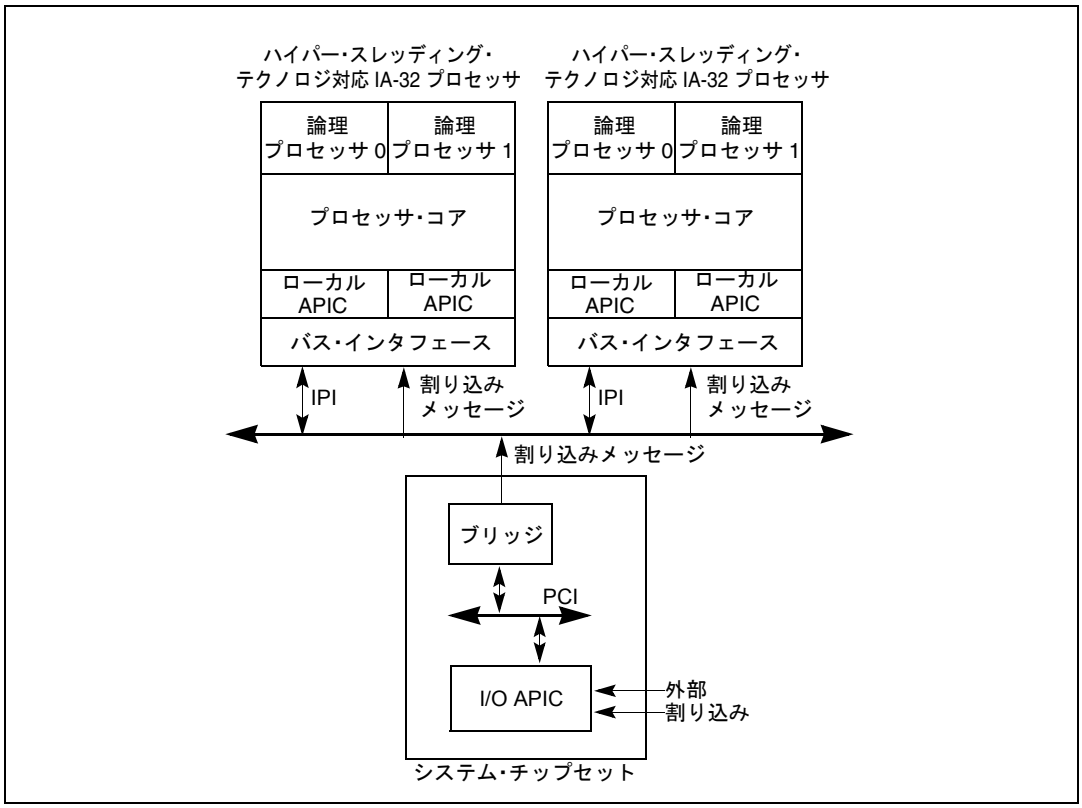


図 7-4. ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサを MP システム内で使用する場合のローカル APIC と I/O APIC

7.7. アイドル状態とブロック状態の管理

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサの実行中に、各論理プロセッサがアクティブにスレッドを実行している場合、論理プロセッサは、必要に応じて共有プロセッサ・リソース（キャッシュ・ライン、TLB エントリ、バスアクセス）を使用する。1 つの論理プロセッサがアイドルになるか（実行する作業がない状態）、ブロックされたときは（ロックまたはセマフォ上）、HLT（ホルト）命令、PAUSE 命令、または MONITOR/MWAIT 命令を使用してコア実行エンジンのリソースを管理できる。

7.7.1. HLT 命令

HLT 命令は、HLT 命令を実行する論理プロセッサの動作を停止し、次の指示があるまで、その論理プロセッサをホルト状態にする（『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」の HLT 命令の説明を参照）。論理プロセッサがホルトにされても、アクティブな論理プロセッサは、物理パッケージ内の共有リソースにフルにアクセスできる。この場合、ホルトにされた論理プロセッサが使用していた共有リソースを、アクティブな論理プロセッサが使用できるようになり、アクティブな論理プロセッサの実行効率が向上する。ホルトにされた論理プロセッサが実行を再開すると、共有リソースは、すべてのアクティブな論理プロセッサの間で再び共有される（ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサでの HLT 命令の使用については、7.7.6.2. 項「アイドル状態の論理プロセッサのホルト」を参照）。

7.7.2. PAUSE 命令

PAUSE 命令は、HT テクノロジ対応 IA-32 プロセッサが、「spin-wait ループ」や、1 つのスレッドが密なポーリング・ループ内で共有ロックまたはセマフォにアクセスするルーチンを実行するときのパフォーマンスを向上させる。プロセッサは、spin-wait ループを実行するとき、起こり得るメモリアクセス順序の違反を検出し、コア・プロセッサのパイプラインをフラッシュするため、ループの終了時に性能上の大きなペナルティが生じる。PAUSE 命令は、コード・シーケンスが spin-wait ループであるというヒントをプロセッサに与える。プロセッサは、このヒントを使用して、メモリアクセス順序の違反を回避し、パイプラインのフラッシュを防ぐ。さらに、PAUSE 命令は、spin-wait ループをパイプラインから排除し、ループが実行リソースを必要以上に使用することを防ぐ（ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサでの PAUSE 命令の使用方法については、7.7.6.1. 項「spin-wait ループ内での PAUSE 命令の使用」を参照）。

7.7.3. MONITOR/MWAIT 命令

オペレーティング・システムは、通常はアイドルループを使用してスレッドの同期化を処理する。一般的なアイドルループ条件では、複数の「ビジーループ」が存在し、それらが一連のメモリ上の位置を使用する。影響を受けるプロセッサは、1 つのループ内で待機し、メモリ上の位置をポーリングして、実行できる作業の有無を確認する。作業のポスティングは、通常はメモリ（待機中のプロセッサの作業キュー）への書き込みになる。作業要求の開始とスケジューリングにかかる時間は、おおよそ数バスサイクルである。

リソース共有（実行リソースを共有する論理プロセッサ）の観点から見ると、OS のアイドルループ内で HLT 命令を使用することが望ましいが、これは他の条件に影響を与える。アイドル状態の論理プロセッサ上で HLT 命令を実行すると、ターゲット・プロセッサが非実行状態に移行する。この場合、他のプロセッサが（ホルト状態のプロセッサに対して作業をポスティングする際に）プロセッサ間割り込みを使用してホルト状態のプロセッサをウェークアップする必要がある。このような割り込みのポスティングと処理のために、新しい作業要求の処理に遅延が生じる。

共有メモリ構成では、ビジーループの終了は、通常は、特定のメモリ上の位置に適用される状態の変化が原因で起こる。このような状態の変化は、多くの場合、他のエージェント（通常はプロセッサ）がそのメモリ上の位置に書き込むことによってトリガされる。

MONITOR/MWAIT は、HLT と PAUSE の使用を補助し、物理リソースを共有する論理プロセッサ間で共有リソースの効率的な分割と分割解除を可能にする。MONITOR は、メモリへの書き込み動作がモニタされる実効アドレス範囲を設定する。MWAIT は、モニタされるアドレス範囲への書き込みが行われるまで、プロセッサを最適化された状態にする（この状態はプロセッサによって異なる）。

MONITOR と MWAIT は、最初の実装では、CPL=0 でのみ利用可能である。

いずれの命令の動作も、プロセッサのモニタ・ハードウェアの状態に依存する。モニタ・ハードウェアは、（MONITOR 命令の実行によって）準備状態になっているか、または（モニタされるメモリ領域へのストアなどの各種のイベントによって）トリガ状態になっている。MWAIT の実行時にモニタ・ハードウェアがトリガ状態になっていた場合は、MWAIT は NOP として動作し、実行は実行ストリーム内の次の命令で再開される。モニタ・ハードウェアの状態をアーキテクチャ的に認識する方法は、MWAIT の動作を介して知る以外にない。

MWAIT を実行したプロセッサは、トリガとなるアドレス範囲への書き込み以外に、各種のイベントによってウェークアップされる可能性がある。これらのイベントには、自発的または非自発的コンテキスト・スイッチを発生させる、次のようなイベントが含まれる。

- 外部割り込み（NMI、SMI、INIT、BINIT、MCERR、A20M# など）
- フォルト、アボート（マシンチェックを含む）
- （モニタの設定から MWAIT の発行までの間に起こる）アーキテクチャ的な TLB の無効化（CR0、CR3、CR4 への書き込みや MSR への特定の書き込み、LMSW の実行など）
- （モニタの設定から MWAIT の発行までの間に起こる）高速システムコールや far コールによる自発的な移行

パワー・マネージメントに関連するイベント（温度モニタ 2 やチップセットによる STPCLK# のアサートなど）では、モニタ・イベント・ペンディング・フラグはクリアされない。フォルトでは、モニタ・イベント・ペンディング・フラグはクリアされない。

ソフトウェアは、命令フロー内で MONITOR と MWAIT の間の自発的コンテキスト・スイッチを許容してはならない。MWAIT を実行しても、モニタ・ハードウェアは再び準備状態にならない。つまり、MONITOR/MWAIT は 1 つのループ内で実行する必要がある。また、トリガとなるアドレスへの書き込み以外の原因で、MWAIT 状態が終了する可能性がある。ソフトウェアは、トリガとなるデータ位置を明示的にチェックして、書き込みの有無を確認する必要がある。また、ソフトウェアは、モニタ命令の実行に続いて（MWAIT 命令を実行する前に）トリガとなるアドレスの値をチェックする必要がある。このチェックの目的は、MONITOR の実行の最中に発生した、トリガとなるアドレスへの書き込みを特定することである。

MONITOR 命令で指定されるアドレス範囲は、ライトバック・キャッシュ・タイプでなければならない。モニタ・ハードウェアがトリガされるのは、モニタされるアドレス範囲に対して、ライトバック・メモリ・タイプのストアが行われた場合に限られる。このアドレス範囲がライトバック・タイプのメモリになっていない場合は、アドレス・モニタ・ハードウェアが正しく設定されないか、モニタ・ハードウェアが準備状態にならないことがある。また、ソフトウェアは、以下のことを保証する責任を負う。

- ビジーループを終了させることを意図しない書き込みは、モニタ・ハードウェアによってモニタされているアドレス領域内の位置には書き込まれない。
- ビジーループを終了させることを意図した書き込みは、モニタされるアドレス領域内の位置に書き込まれる。

これらの条件を保証しないと、偽りのウェークアップ（意図したデータ位置への書き込みによるものではない MWAIT 状態の終了）の数が増える。このことは、パフォーマンスに悪影響を与える。ソフトウェアは、パディングを使用して偽りのウェークアップを防がなければならない場合がある。CPUID は、モニタリング用のデータ位置のサイズを確認する機能と、パッドのサイズを決めるための機能を備えている。

7.7.4. Monitor/Mwait のアドレス範囲の決定

MONITOR/MWAIT 命令を使用するには、ソフトウェアは、MONITOR/MWAIT 命令によってモニタされる領域の大きさと、マルチプロセッサ・システム内のキャッシュ・スヌープ・トラフィック用のコヒーレンシ・ライン・サイズを知っている必要がある。この情報は、CPUID のモニタリーフ機能（EAX=05H）を使用してクエリできる。最小および最大モニタ・ライン・サイズは、次の条件を満たす必要がある。

- ウェークアップの失敗を避けるために、書き込みのモニタに使用されるデータ構造が最小モニタ・ライン・サイズ内に収まるように保証する。そうなっていないと、MWAIT の終了をトリガするはずの書き込みの後、プロセッサがウェークアップされない場合がある。
- 偽りのウェークアップを避けるために、最大モニタ・ライン・サイズを使用して、書き込みのモニタに使用されるデータ構造をパディングする。ソフトウェアは、MWAIT のトリガリング領域内に、このデータ構造を超える無関係のデータ変数が存在しないことを保証しなければならない。この状況を避けるために、パッドが必要になる場合がある。

上記の 2 つの値は、システム内のキャッシュ・ライン・サイズとは無関係である。ソフトウェアは、これらの値とキャッシュ・ライン・サイズの関係について何も前提にしてはならない。シングルクラスタ・システムでは、2 つのパラメータはデフォルトにより同じ値になる（モニタ・トリガリング領域のサイズは、システム・コヒーレンシ・ライン・サイズと同じ値になる）。

CPUID で返されるモニタ・ライン・サイズに基づいて、OS は、適切なパディングを使用してデータ構造を動的に割り当てる必要がある。OS が静的なデータ構造を使用する必要がある場合は、スレッドの同期化にデータ構造を適応させて、動的に割り当てられたデータバッファを使用する。この手法を使用できない場合は、静的なデータ構造の使用時には MONITOR/MWAIT を使用しないことを検討する。

マルチクラスタ・システム上で MONITOR/MWAIT 用のデータ構造を正しく設定するには、プロセッサ、チップセット、BIOS 間の相互作用が必要である（システム・コヒーレンシ・ライン・サイズは、システムに使用されるチップセットによって異なる。このサイズは、プロセッサのモニタ・トリガリング領域のサイズとは異なる可能性がある）。BIOS は、IA32_MONITOR_FILTER_LINE_SIZE MSR を使用して、システム・コヒーレンシ・ライン・サイズを正しい値に設定する役割を受け持つ。IA32_MONITOR_FILTER_LINE_SIZE MSR に書き込まれた値とモニタ・トリガリング領域のサイズの相対的な大きさに基づいて、2 つのパラメータのうち小さい方の値が、最小モニタ・ライン・サイズとして報告される。2 つのパラメータのうち大きい方の値が、最大モニタ・ライン・サイズとして報告される。

7.7.5. MP システム内の論理プロセッサの識別

すべての IA-32 プロセッサで、システム・ハードウェアは、電源投入時またはリセット時にそのプロセッサの初期 APIC ID を設定する（7.6.4. 項「ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサの初期化」を参照）。HT テクノロジ対応 IA-32 プロセッサでは、システム・ハードウェアは、システムバス上の各論理プロセッサに個別の APIC ID を割り当てる。

論理プロセッサの APIC ID は、論理プロセッサ ID、物理パッケージ ID、クラスタ ID の 3 つのフィールドで構成される。図 7-5. は、これらのフィールドのレイアウトを示している。ここで、ビット 0 は 1 ビットの論理プロセッサ ID、ビット 1 とビット 2 は 2 ビットのパッケージ ID、ビット 3 とビット 4 は 2 ビットのクラスタ ID になる。ビット 0 を使用して、パッケージ内の 2 つの論理プロセッサを識別する。

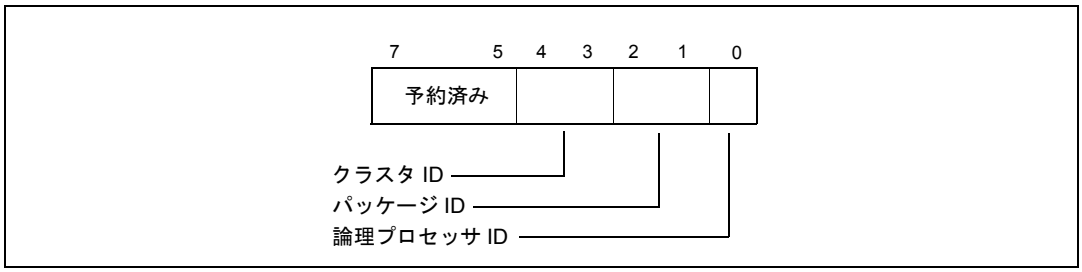


図 7-5. APIC ID の解釈

表 7-1. は、4 個の MP 型インテル® Xeon™ プロセッサ（合計 8 個の論理プロセッサ）を搭載したシステム内の論理プロセッサに対して生成される APIC ID を示している。インテル Xeon プロセッサ MP 内の 2 つの論理プロセッサのうち、論理プロセッサ 0 は「プライマリ論理プロセッサ」、論理プロセッサ 1 は「セカンダリ論理プロセッサ」とも呼ばれる。

表 7-1. 4 個のハイパー・スレッディング・テクノロジ対応 MP 型インテル® Xeon™ プロセッサを搭載したシステム内の論理プロセッサの初期 APIC ID

論理プロセッサの初期 APIC ID	物理プロセッサ ID	論理プロセッサ ID
0H	0H	0H
1H	0H	1H
2H	1H	0H
3H	1H	1H
4H	2H	0H
5H	2H	1H
6H	3H	0H
7H	3H	1H

ソフトウェアは、7.5.5. 項「MP システム内のプロセッサの識別」で説明した 2 つの方法で、システム内の論理プロセッサの APIC ID を確認できる。ただし、MP テーブルには、各物理パッケージ内のプライマリ論理プロセッサの APIC ID だけが記録される。システム内のすべての論理プロセッサは、ACPI テーブルに記録される。ACPI テーブルの上部にプライマリ論理プロセッサがリストされ、それに続いてセカンダリ論理プロセッサがリストされる。

1 つの物理プロセッサ当たり 3 つ以上の論理プロセッサをサポートする、将来の HT テクノロジ対応 IA-32 プロセッサでは、図 7-5. に示した論理プロセッサ・ビットは、各論理プロセッサを識別できるように、2 ビットまたは 3 ビットのフィールドに拡張される。パッケージ ID フィールドとクラスタ ID フィールドは左にシフトする。また、パッケージ ID が 3 ビット以上に拡張され、クラスタ ID フィールドがさらに左にシフトされる可能性もある。

オペレーティング・システムとアプリケーション・ソフトウェアは、EAX レジスタのパラメータを 1 に設定して CPUID 命令を実行したとき、EBX レジスタに返される論理プロセッサ数フィールドとローカル APIC 物理 ID フィールドを解釈して、特定のプロセッサの APIC ID のレイアウトを確認できる。

HT テクノロジ非対応の IA-32 プロセッサでは、ソフトウェアは、ローカル APIC ID レジスタに値を書き込むことにより、論理プロセッサに割り当てられた APIC ID を変更できる。ただし、この場合も、CPUID 命令を実行すると、そのプロセッサの初期 APIC ID（電源投入時またはリセット時に割り当てられた値）が返される。

図 7-5. は、現在の HT テクノロジ対応プロセッサについて、APIC ID のクラスタ ID、パッケージ ID、論理プロセッサ ID のビット・フィールドのレイアウトを示している（1 パッケージ当たり 2 つの論理プロセッサ）。一般的に、1 パッケージ当たりの論理プロセッサ数が限られているパッケージ内の論理プロセッサの APIC ID（クラスタ ID を除く）の内容は、次の式で求められる。

((パッケージ ID << (1+((int)(log(2)(max(Logical_Per_Package-1,1)))) || 論理プロセッサ ID)

この公式を使用して、今後の HT テクノロジ対応プロセッサについても、論理プロセッサと物理パッケージの関係を判定できる。以下の疑似コード（例 7-1 と例 7-2）は、論理プロセッサと物理プロセッサの関係を判定するためのアルゴリズムを示している。このアルゴリズムには、1 パッケージ当たりの論理プロセッサ数の制限はない。このアルゴリズムは、システム内の各論理プロセッサ上で実行され、オペレーティング・システム固有のアフィニティを使用して結合を行う。アルゴリズムの実行後、同じ物理パッケージ内の論理プロセッサは、同じプロセッサ ID を持つ。システム内のすべてのプロセッサについて、1 つの物理プロセッサでサポートされる論理プロセッサの数は一致していなければならない。

HT テクノロジのサポートを検出し、論理プロセッサとそれに対応する物理プロセッサの関係を判定するアルゴリズムは、次の 5 つの手順で構成される。

1. プロセッサ内の HT テクノロジのサポートを検出する。
2. 物理プロセッサ・パッケージ内で使用可能な論理プロセッサの数を計算する。
3. このプロセッサの初期 APIC ID を抽出する。
4. マスク値とビットシフト値を計算する。
5. 論理プロセッサ ID と物理プロセッサ・パッケージ ID を計算する。

例 7-2. ハイパー・スレッディング・テクノロジに対応した物理プロセッサの ID を抽出する一般的なアルゴリズム

```
1.  Pseudo-code to detect support for Hyper-Threading Technology
    in a processor.

// Returns non-zero if Hyper-Threading Technology is supported on
// the processors and zero if not. This does not mean that
// Hyper-Threading Technology is necessarily enabled.

unsigned int HTSupported(void)
{
    try { // verify cpuid instruction is supported
       execute cpuid with eax = 0 to get vendor string
       execute cpuid with eax = 1 to get feature flag and
       signature
    }
    except (EXCEPTION_EXECUTE_HANDLER) {
    returen 0 ; // CPUID is not supported and so Hyper-Threading
       // Technology is not supported
    }

    // Check to see if this a a Genuine Intel Processor
    // a member of the Pentium 4 processor family
    // and supporting Hyper-Threading Technology

if (vendor string NEQ GenuineIntel)
       if (family signature NEQ Pentium4Family)
           return (feature_flag_edx & HTT_BIT);
    return 0;
}

2.  Pseudo-code to identify the number of logical processors per
    physical processor package.

#define NUM_LOGICAL_BITS 0x00FF0000 // EBX[23:16] indicate number
                                    // of logical processor per
                                    // package

// Returns the number of logical processors per physical
processor.
```

```
unsigned char LogicalProcessorsPerPackage(void)
{
    if (!HTSupported()) return (unsigned char) 1;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned char) ((reg_ebx & NUM_LOGICAL_BITS) >> 16);
}
```

例 7-3. マスクの効率的な計算による論理プロセッサ番号の取得

```
3.  Pseudo-code to extract the initial APIC ID of a processor

#define INITIAL_APIC_ID_BITS 0xFF000000 // EBX[31:24] initial
                                        // APIC ID

// Returns the 8-bit unique initial APIC ID for the processor this
// code is actually running on. The default value returned is
// 0xFF if Hyper-Threading Technology is not supported.

unsigned char GetAPIC_ID (void)
{
    unsigned int reg_ebx = 0;
    if (!HTSupported()) return (unsigned char) -1;
    execute cpuid with eax = 1
    store returned value of ebx
    return (unsigned char) ((reg_ebx & INITIAL_APEIC_ID_BITS)
    >> 24;
}

4.  Sample code to compute a mask value and a bit-shift value, the
    logical processor ID and physical processor package ID.

    unsigned char i = 1;
    unsigned char PHY_ID_MASK = 0xFF;
    unsigned char PHY_ID_SHIFT = 0;
    unsigned char APIC_ID;
    unsigned char LOG_ID, PHY_ID;

    Logical_Per_Package = LogicalProcessorsPerPackage();
    While (i < Logical_Per_Package){
       i *= 2;
       PHY_ID_MASK <<= 1;
       PHY_ID_SHIFT++;
    }
    // Assume this thread is running on the logical processor from
    // which we extract the logical processor ID and its physical
    // processor package ID. If not, use the OS-specific affinity
    // service (See example 7-3) to bind this thread to the target
    // logical processor
    APIC_ID = GetAPIC_ID();
    LOT_ID = APIC_ID & ~PHY_ID_MASK;
```

```
PHY_ID = APIC_ID >> PHY_ID_SHIFT;
```

例 7-4. OS 固有のアフィニティ・サービスを使用した、
MP システム内の論理プロセッサ ID の計算

```
5.  Compute the logical processor ID and physical processor
    package ID.

// The OS may limit the processor that this process may run on.

    hCurrentProcessHandle = GetCurrentProcess();
    GetProcessAffinityMask(hCureentPorcessHandle,
       &dwProcessAffinity, &dwSystemAffinity);

    // If the available process affinity mask does not equal the
    // available system affinity mask, then determining if
    // Hyper-Threading Technology is enabled may not be possible.

    if (dwProcessAffinity != dwSystemAffinity)
       printf ("This process can not utilize all processors. \n"),

    dwAffinityMask = 1;
    while (dwAffinityMask != 0 &&
         dwAffinityMask <= dwProcessAffinity) {
       // Check to make sure we can utilize this processor first.
       if (dwAffinityMask & dwProcessAffinity){
          if (SetProcessAffinityMask(hCurrentProcessHandle,
             dwAffinityMask)) {

          Sleep(0);  // May not be running on the logical processor
                     // on the affinity just set. Sleep gives the
                     // OS a chance to switch to the desired
                     // logical processor.

          // Retrieve APIC_ID for this logical processor
          // Extract logical processor ID and physical processor
          // package ID

          }
       }
    }
```

7.7.6. 必要なオペレーティング・システムのサポート

この項では、HTテクノロジ対応IA-32プロセッサ上でオペレーティング・システムを実行するために必要な変更について説明する。また、オペレーティング・システムが物理パッケージ内の論理プロセッサをより効率的に使用できる、最適化手法についても説明する。必要な変更と推奨する最適化は、HTテクノロジ対応IA-32プロセッサをサポートするためにWindows* XPおよびLinux*カーネル2.4.0オペレーティング・シス

テム内に現れる変更のタイプの代表的な例である。HTテクノロジ対応IA-32プロセッサのその他の最適化手法については、『インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサ最適化リファレンス・マニュアル』に記載されている（資料番号については、1.4. 節「参考文献」を参照）。

7.7.6.1. spin-wait ループ内での PAUSE 命令の使用

インテルでは、インテル® Xeon™ プロセッサまたはインテル® Pentium® 4プロセッサ、あるいはその両方で実行されるすべてのspin-waitループ内にPAUSE命令を配置するように推奨している

spin-wait ループを使用するソフトウェア・ルーチンには、マルチプロセッサ同期化プリミティブ（スピンロック、セマフォ、ミューテックス変数）とアイドルループが含まれる。このようなルーチンでは、プロセッサ・コアが load-compare-branch ループの実行でビジーの間、スレッドはリソースが利用可能になるのを待つ。このようなループの中に PAUSE 命令を配置すると、効率は大幅に向上する（7.7.2. 項「PAUSE 命令」を参照）。次のルーチンは、PAUSE 命令を使用する spin-wait ループの例を示している。

```
Spin_Lock:
       CMP lockvar, 0;Check if lock is free
       JE Get_Lock
       PAUSE ;  Short delay
       JMP Spin_Lock
Get_Lock:
       MOV EAX, 1
       XCHG EAX, lockvar ; Try to get lock
       CMP EAX, 0 ; Test if successful
       JNE Spin_Lock
Critical_Section:
       <critical section code>
       MOV lockvar, 0
       ...
Continue:
```

上の spin-wait ループは、"test, test-and-set" 手法を使用して、同期化変数が使用できるかどうか確認する。spin-wait ループを作成するときは、この手法の使用を推奨する。

インテル Pentium 4 プロセッサより以前の世代の IA-32 プロセッサでは、PAUSE 命令は NOP 命令として扱われる。

C0 アイドルループ内での MONITOR/MWAIT の可能な使用例

オペレーティング・システムは、各種のアイドル状態に対する各種のハンドラを実装できる。例 7-5. に、ACPI 互換 OS 上の一般的な OS アイドルループを示す。

例 7-5. 一般的な OS アイドルループ

```
// WorkQueue is a memory location indicating there is a thread
// ready to run.  A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The idle loop is entered with interrupts disabled.
WHILE (1) {
     IF (WorkQueue) THEN {
     // Schedule work at WorkQueue
}    ELSE {
// No work to do - wait in appropriate C-state handler depending
// on Idle time accumulated
     IF (IdleTime >= IdleTimeThreshhold) THEN {
     // Call appropriate C1, C2, C3 state handler, C1 handler
     // shown below
        }
     }
}
// C1 handler uses a Halt instruction
VOID C1Handler()
{    STI
     HLT
}
```

MONITOR 命令と MWAIT 命令をサポートしている場合は、C0 アイドル状態ループ内での MONITOR 命令と MWAIT 命令の使用を検討できる。

例 7-6. C0 アイドルループ内で MONITOR/MWAIT を使用した OS アイドルループ

```
// WorkQueue is a memory location indicating there is a thread
// ready to run.  A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The following example assumes that the necessary padding has
been
// added surrounding WorkQueue to eliminate false wakeups
// The idle loop is entered with interrupts disabled.
WHILE (1) {
     IF (WorkQueue) THEN {
     // Schedule work at WorkQueue
}    ELSE {
// No work to do - wait in appropriate C-state handler depending
// on Idle time accumulated
     IF (IdleTime >= IdleTimeThreshhold) THEN {
        // Call appropriate C1, C2, C3 state handler, C1
        // handler shown below
        MONITOR WorkQueue // Setup of eax with WorkQueue
LinearAddress,
                       // ECX, EDX = 0
        IF (WorkQueue != 0) THEN {
             MWAIT
          }
```

```
        }
    }
}
// C1 handler uses a Halt instruction
VOID C1Handler()
{   STI
    HLT
}
```

7.7.6.2. アイドル状態の論理プロセッサのホルト

2 つの論理プロセッサのうち 1 つがアイドル状態または長時間の spin-wait ループにある場合は、HLT 命令によってその論理プロセッサを直接にホルトにする。

MP システムでは、オペレーティング・システムは、アイドル状態のプロセッサに、実行キュー内に実行可能なソフトウェア・タスクがないかどうか常にチェックするループを実行できる。アイドルループを実行している論理プロセッサは、（物理パッケージ内の他の論理プロセッサが使用できたはずの）コアの実行リソースを大量に使用する。この理由で、アイドル状態の論理プロセッサをホルトにすると、パフォーマンスが最適化される。[1] 物理パッケージ内のすべての論理プロセッサがホルトにされると、その物理プロセッサは省電力状態に移行する。

C1 アイドルループ内での MONITOR/MWAIT の可能な使用例

オペレーティング・システムは、OS の C1 アイドルループ内で HLT を MONITOR/MWAIT で置き換えることも検討できる。例 7-7. に例を示す。

例 7-7. C1 アイドルループ内で MONITOR/MWAIT を使用した OS アイドルループ

```
// WorkQueue is a memory location indicating there is a thread
// ready to run.  A non-zero value for WorkQueue is assumed to
// indicate the presence of work to be scheduled on the processor.
// The following example assumes that the necessary padding has been
// added surrounding WorkQueue to eliminate false wakeups
// The idle loop is entered with interrupts disabled.
WHILE (1) {
    IF (WorkQueue) THEN {
    // Schedule work at WorkQueue
}   ELSE {
// No work to do - wait in appropriate C-state handler depending
// on Idle time accumulated
    IF (IdleTime >= IdleTimeThreshhold) THEN {
       // Call appropriate C1, C2, C3 state handler, C1
       // handler shown below
       }
```

1. ホルト状態を必要以上に利用すると、性能上のペナルティが生じるときがある。オペレーティング・システムは、そのオペレーティング・システム上でのパフォーマンスのトレードオフを評価する必要がある。

```
    }
}
// C1 handler uses a Halt instruction
VOID C1Handler()
{
       MONITOR WorkQueue // Setup of eax with WorkQueue LinearAddress,
                         // ECX, EDX = 0
       IF (WorkQueue != 0) THEN {
            STI
            MWAIT      // EAX, ECX = 0
          }

}
```

7.7.6.3. 複数の論理プロセッサ上のスレッドのスケジューリングのガイドライン

論理プロセッサでは、スレッドが実行のために論理プロセッサに発行される順番は、システムの全体的な効率に影響を与える。以下のガイドラインにしたがってスレッドの実行をスケジューリングすることを推奨する。

- 各物理パッケージ内の 1 つの論理プロセッサにスレッドを発行してから、利用可能な物理パッケージ内の残りの論理プロセッサにスレッドを発行する。HTテクノロジ対応 IA-32 プロセッサを 2 個以上使用する MP システムでは、スレッドの処理を 1 つまたは 2 つの物理プロセッサに集中するのではなく、すべての物理パッケージに分散するようにする。
- プロセッサのアフィニティを使用して、スレッドを特定の物理プロセッサに割り当てる。この手法により、スレッドが保留された後、実行のために再び発行されるとき、そのスレッドのコードとデータの一部がプロセッサのキャッシュに残っている可能性が高くなる。各論理プロセッサは物理プロセッサのキャッシュを共有しているため、物理パッケージ内のどの論理プロセッサにスレッドを発行してもよい。

7.7.6.4. 実行ベースのタイミング・ループの除去

プロセッサの実行速度に依存するタイミング・ループを使用して、時間を測定するのはお勧めできない。これには次のような理由がある。

- 特定のクロックスピードで動作する IA-32 プロセッサ上でタイミング・ループのキャリブレーションを行ってから、異なるクロックスピードで動作するプロセッサ上でそのタイミング・ループを実行すると、問題が発生する。
- 実行ベースのタイミング・ループをキャリブレーションするためのルーチンを、HT テクノロジ対応 IA-32 プロセッサ上で実行すると、予測不可能な結果が発生す

る。これは、物理パッケージ内の論理プロセッサの間で実行リソースが共有されるためである。

上記の問題を回避するために、タイミング・ループ・ルーチンは、システム内の論理プロセッサの実行速度に依存しないタイミング機構を使用する必要がある。一般的に、以下のソースを使用できる。

- 高分解能システムタイマ（例えば、Intel 8254）
- プロセッサ内の高分解能タイマ（ローカルAPICタイマやタイムスタンプ・カウンタなど）

詳細については、『インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサ最適化リファレンス・マニュアル』を参照のこと（資料番号は、1.4.節「参考文献」を参照）。

7.7.6.5. アライメントされた128バイト・メモリ・ブロック内へのロックとセマフォの配置

ソフトウェアが、ロックまたはセマフォを使用して、プロセス、スレッド、または他のコード・セクションの同期をとる場合、インテルでは、1つのキャッシュ・ライン内のロックまたはセマフォは1つだけにすることを推奨する。幅128バイトのキャッシュ・ラインを持つインテル® Xeon™プロセッサMPの場合、この推奨事項に従うのは、各ロックまたはセマフォが128バイト境界を始点とする128バイト・メモリ・ブロック内に含まれる意味である。この手法により、ロックの処理に必要なバス・トラフィックが最小限に抑えられる。

8

アドバンスド・プログラマブル割り込みコントローラ（APIC）

第 8 章
アドバンスド・プログラマブル割り込みコントローラ（APIC）

8

アドバンスト・プログラマブル割り込みコントローラ（APIC）（以下の各節ではローカル APIC と呼ばれる）は、インテル® Pentium® プロセッサで IA-32 プロセッサに導入され（18.24. 節「アドバンスト・プログラマブル割り込み コントローラ（APIC）」を参照）、現在はインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサに搭載されている（8.4.2. 項「ローカル APIC の有無」を参照）。**ローカル APIC** は、プロセッサのために次の 2 つの主な機能を実行する。

- プロセッサの割り込みピン、内部ソース、または外部 I/O APIC（または他の外部割り込みコントローラ）からの割り込みを受信し、処理のためにプロセッサ・コアに送信する。
- マルチプロセッサ（MP）システムでは、システムバス上の他の IA-32 プロセッサとの間でプロセッサ間割り込み（IPI）メッセージの送信と受信を行う。これらの IPI メッセージを使用して、システム内のプロセッサの間で割り込みを分散したり、システム全体にわたる機能（例えば、プロセッサのブートアップや、プロセッサのグループ内での作業の分散）を実行できる。

外部 **I/O APIC** は、インテルのシステム・チップセットの一部である。外部 I/O APIC の主な機能は、システムとそれに関連する I/O デバイスからの外部割り込みイベントを受信し、割り込みメッセージとしてローカル APIC に中継することである。MP システム内では、外部 I/O APIC は、システムバス上の選択したプロセッサまたはプロセッサ・グループのローカル APIC に対して外部割り込みを分散する機構も提供する。

本章では、ローカル APIC とそのプログラミング・インタフェースについて詳しく説明し、ローカル APIC と I/O APIC の間のインタフェースの概要も示す。I/O APIC の詳細は、インテルまでお問い合わせのこと。

ローカル APIC が、関連するプロセッサ・コアに対して、処理のために割り込みを送信すると、そのプロセッサは、第 5 章「割り込みと例外の処理」で説明した例外処理機構を使用して割り込みを処理する。5.1. 節「割り込みと例外の概要」に、IA-32 アーキテクチャ内の割り込みと例外の処理の概要が記載されている。以下の各項以外に 5.1. 節も読み、IA-32 APIC アーキテクチャとその機能についてよく理解することを推奨する。

8.1. ローカル APIC と I/O APIC の概要

各ローカル APIC は、一連の APIC レジスタ（表 8-1. を参照）とそれに関連するハードウェアで構成される。このハードウェアは、プロセッサ・コアに対する割り込みの伝達と IPI メッセージの生成を制御する。APIC レジスタは、メモリにマップされ、MOV 命令を使用して読み出しと書き込みができる。

ローカル APIC は、以下のソースからの割り込みを受信できる。

- **ローカル接続された I/O デバイス。**これらの割り込みは、プロセッサのローカル割り込みピン（LINT0 と LINT1）に直接接続された I/O デバイスによってアサートされるエッジまたはレベルとして発信される。これらの I/O デバイスは、いずれかのローカル割り込みピンを介してプロセッサに接続される、8259 タイプの割り込みコントローラに接続されていてもよい。
- **外部接続された I/O デバイス。**これらの割り込みは、I/O APIC の割り込み入力ピンに接続された I/O デバイスによってアサートされるエッジまたはレベルとして発信される。これらの割り込みは、I/O APIC からシステム内の 1 つ以上の IA-32 プロセッサに I/O 割り込みメッセージとして送信される。
- **プロセッサ間割り込み（IPI）。**IA-32 プロセッサは、IPI 機構を使用して、システムバス上の他のプロセッサまたはプロセッサのグループに割り込みをかけることができる。IPI は、ソフトウェアの自己割り込み、割り込みの転送、またはプリエンティティブ・スケジューリングなどの目的で使用される。
- **APIC タイマが生成する割り込み。**ローカル APIC タイマが設定したカウントに達したとき、関連するプロセッサにローカル割り込みを送信するように、APIC タイマをプログラムできる（8.5.4. 項「APIC タイマ」を参照）。
- **性能モニタリング・カウンタ割り込み。**インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサは、性能モニタリング・カウンタのオーバーフローが発生したとき、関連するプロセッサに割り込みを送信する機能を持っている（15.9.6.9. 項「オーバーフロー時の割り込みの生成」を参照）。
- **温度センサ割り込み。**インテル Pentium 4 プロセッサとインテル Xeon プロセッサは、内部温度センサがトリップしたとき、自分に割り込みを送信する機能を持っている（13.16.2. 項「温度モニタ」を参照）。
- **APIC 内部エラー割り込み。**ローカル APIC 内でエラー状態が検出されたとき（サポートしていないレジスタにアクセスしようとした場合など）、関連するプロセッサに割り込みを送信するように、APIC をプログラムできる（8.5.3. 項「エラー処理」を参照）。

これらの割り込みソースのうち、プロセッサの LINT0 ピンと LINT1 ピン、APIC タイマ、性能モニタリング・カウンタ、温度センサ、内部 APIC エラー・ディテクタは、**ローカル割り込みソース**と呼ばれる。ローカル割り込みソースからの信号を受信する

と、ローカルAPICは、**ローカル・ベクタ・テーブル**またはLVTと呼ばれるAPICレジスタグループによって設定された割り込み伝達プロトコルを使用して、その割り込みをプロセッサ・コアに伝達する（8.5.1.項「ローカル・ベクタ・テーブル」を参照）。ローカル・ベクタ・テーブル内には、ローカル割り込みソースごとに別々のエントリが用意されている。これにより、各ソースに対して個別に割り込み伝達プロトコルを設定できる。例えば、LINT1ピンをNMIピンとして使用する場合、ベクタ番号2の割り込み（NMI割り込み）をプロセッサ・コアに伝達するように、ローカル・ベクタ・テーブル内のLINT1のエントリを設定できる。

ローカルAPICは、他の2つの割り込みソース（外部接続されたI/Oデバイスと IPI）からの割り込みを、ローカルAPICのIPIメッセージ処理機能を使用して処理する。

各プロセッサは、自分のローカルAPIC内の割り込みコマンドレジスタ（ICR）をプログラムして、IPIを生成できる（8.6.1.項「割り込みコマンドレジスタ（ICR）」を参照）。ICRへの書き込みが行われると、システムバス上（インテルPentium 4プロセッサおよびインテルXeonプロセッサの場合）またはAPICバス上（インテル® Pentium®プロセッサおよびP6ファミリ・プロセッサの場合）でIPIメッセージが生成され、発行される（8.2.節「システムバスとAPICバス」を参照）。

IPIは、システム内の他のIA-32プロセッサに送信されることも、発信元プロセッサに送信されることもある（自己割り込み）。送信先のプロセッサがIPIメッセージを受信すると、そのプロセッサのローカルAPICが（ベクタ番号やトリガモードなど、メッセージ内の情報を使用して）メッセージを自動的に処理し、サービスのためにプロセッサ・コアに伝達する。ローカルAPICのIPIメッセージの伝達/受け入れ機構についての詳細は、8.6.節「プロセッサ間割り込みの発行」を参照のこと。

ローカルAPICは、I/O APICを介して、外部接続されたデバイスからの割り込みを受信することもできる（図8-1.を参照）。I/O APICは、システム・ハードウェアおよびI/Oデバイスが生成した割り込みを受信し、割り込みメッセージとしてローカルAPICに転送する役割を受け持つ。

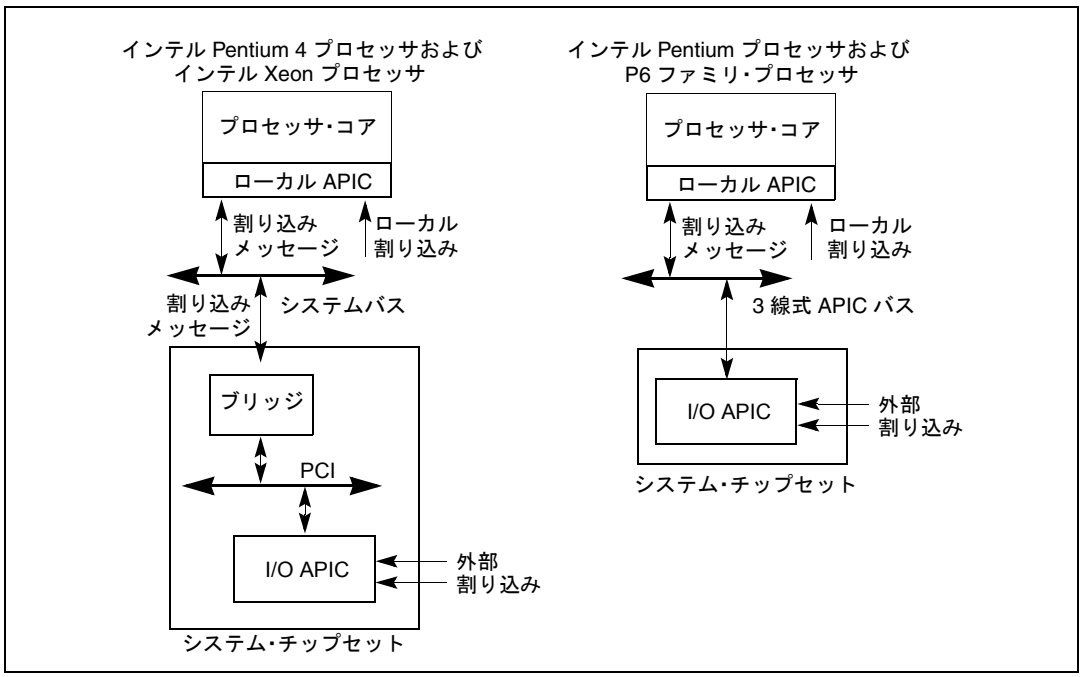


図 8-1. シングルプロセッサ・システム内のローカル APIC と I/O APIC の関係

I/O APIC 上の個々のピンは、アサートされたときに特定の割り込みベクタを生成するようにプログラムできる。また、I/O APIC は、「仮想ワイヤモード」を使用して、標準的な 8259A タイプの外部割り込みコントローラと通信できる。

ローカル APIC を無効にして（8.4.3. 項「ローカル APIC の有効化または無効化」を参照）、関連するプロセッサ・コアが、8259A 割り込みコントローラから直接に割り込みを受信することもできる。

ローカル APIC と I/O APIC は、いずれも MP システム内で動作するように設計されている（図 8-2. と図 8-3. を参照）。各ローカル APIC は、I/O APIC から割り込みメッセージとして受信した外部生成割り込みと、システムバス上の他のプロセッサおよびそのプロセッサ自身からの IPI の両方を処理する。（ローカル割り込みピンを介して個々のプロセッサに割り込みも伝達できるが、この機構は通常は MP システムでは使用されない。）

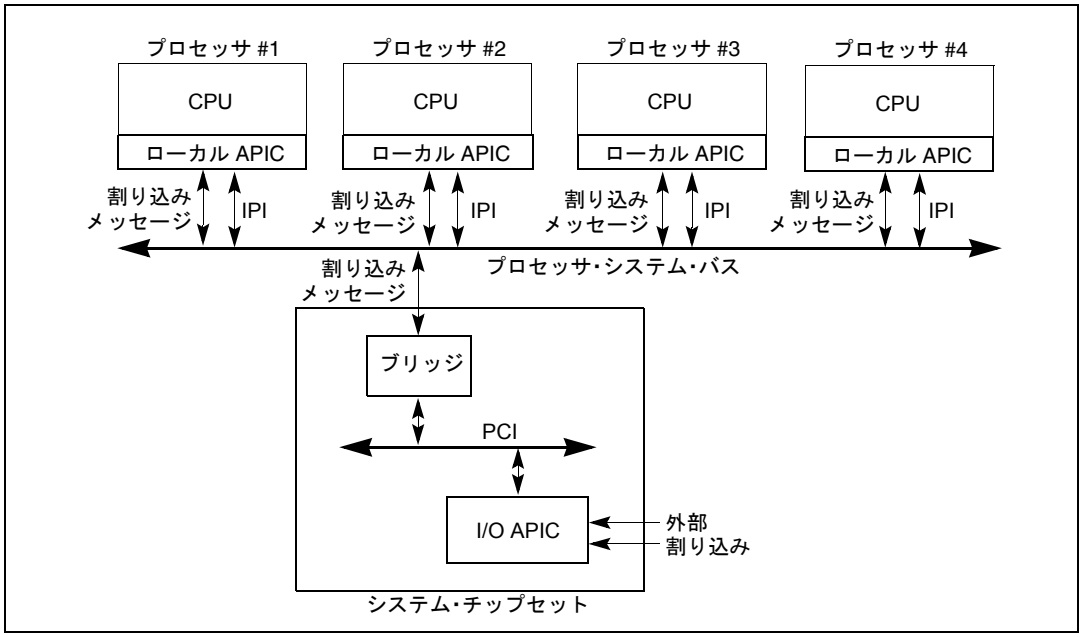


図 8-2. マルチプロセッサ・システム内でインテル® Xeon™ プロセッサを使用する場合の
ローカル APIC と I/O APIC

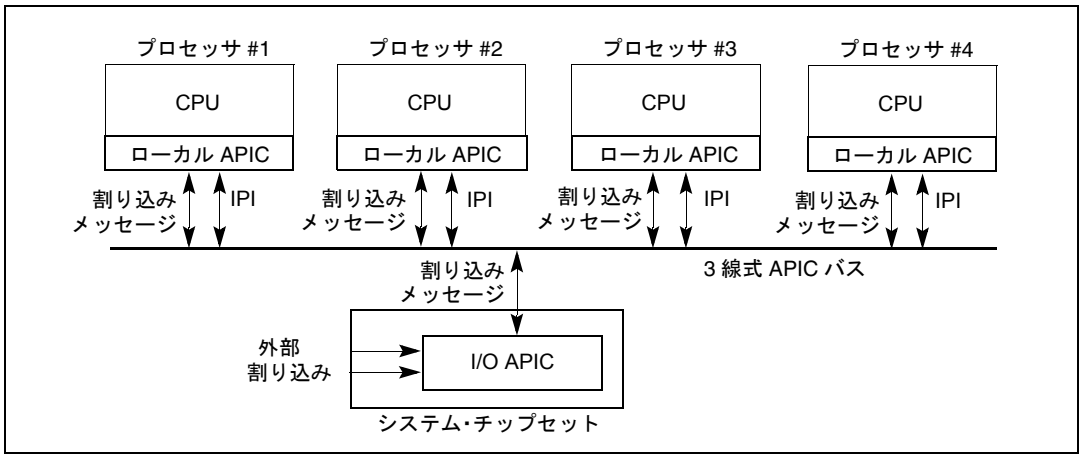


図 8-3. マルチプロセッサ・システム内で P6 ファミリ・プロセッサを使用する場合の
ローカル APIC と I/O APIC

IPI機構は、通常は、MPシステム内でシステムバス上の他のプロセッサまたは発信元プロセッサ自身に対して固定割り込み（特定のベクタ番号の割り込み）および特殊目的の割り込みを送信するために使用される。例えば、1つのローカルAPICはIPIを使用して、固定割り込みをサービスのために他のプロセッサに送信できる。また、シス

テムバス上の 1 つ以上のプロセッサは、特殊目的の IPI（NMI、INIT、SMI、SIPI IPI など）を使用して、システム全体のブートアップおよび制御機能を実行できる。

以下の各節では、ローカル APIC に焦点を合わせて、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのローカル APIC の実装方法について説明する。以下の説明で、「ローカル APIC」および「I/O APIC」の一般的な用語は、P6 ファミリ・プロセッサではローカル APIC と I/O APIC を指し、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではローカル APIC と I/O xAPIC を指す（8.3. 節「インテル® 82489DX 外部 APIC、APIC、xAPIC の関係」を参照）。

8.2. システムバスと APIC バス

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、I/O APIC とローカル APIC は、3 線式 APIC 間バスを介して通信する（図 8-3. を参照）。また、ローカル APIC は、APIC バスを使用して IPI の送信と受信を行う。APIC バスと APIC バス上のメッセージは、ソフトウェアからは認識されず、アーキテクチャ機能として分類されない。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサから、（xAPIC アーキテクチャを使用する）I/O APIC とローカル APIC は、システムバスを介して通信するようになった（図 8-2. を参照）。この場合、I/O APIC は、インテル・チップセットの一部であるブリッジ・ハードウェアを介して、システムバス上のプロセッサに対して割り込み要求を送信する。このブリッジ・ハードウェアは、ローカル APIC に送信される実際の割り込みメッセージを生成する。ローカル APIC 間の IPI は、システムバス上に直接送信される。

8.3. インテル® 82489DX 外部 APIC、APIC、xAPIC の関係

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサのローカル APIC は、アーキテクチャ的にはインテル® 82489DX 外部 APIC の一部である。ローカル APIC と外部 APIC の相違点については、18.24.1. 項「ローカル APIC と 82489DX のソフトウェア的に認識可能な相違点」を参照のこと。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサに使用される APIC アーキテクチャ（xAPIC アーキテクチャと呼ばれる）は、P6 ファミリ・プロセッサの APIC アーキテクチャを拡張したものである。APIC アーキテクチャと xAPIC アーキテクチャの主な相違点は、xAPIC アーキテクチャでは、ローカル APIC と I/O APIC はシステムバスを介して互いに通信するが、APIC アーキテクチャでは、APIC バスを介して通信することである（8.2. 節「システムバスと APIC バス」を参照）。また、APIC アーキテクチャ機能の一部は、xAPIC アーキテクチャでは拡張または変更されている。これらの拡張と変更については、以下の各節で説明する。

8.4. ローカル APIC

以下の各項では、ローカル APIC のアーキテクチャと、ローカル APIC の検出、識別、ステータスの確認の方法について説明する。ローカル APIC のプログラム方法については、8.5.1. 項「ローカル・ベクタ・テーブル」と 8.6.1. 項「割り込みコマンドレジスタ（ICR）」に記載されている。

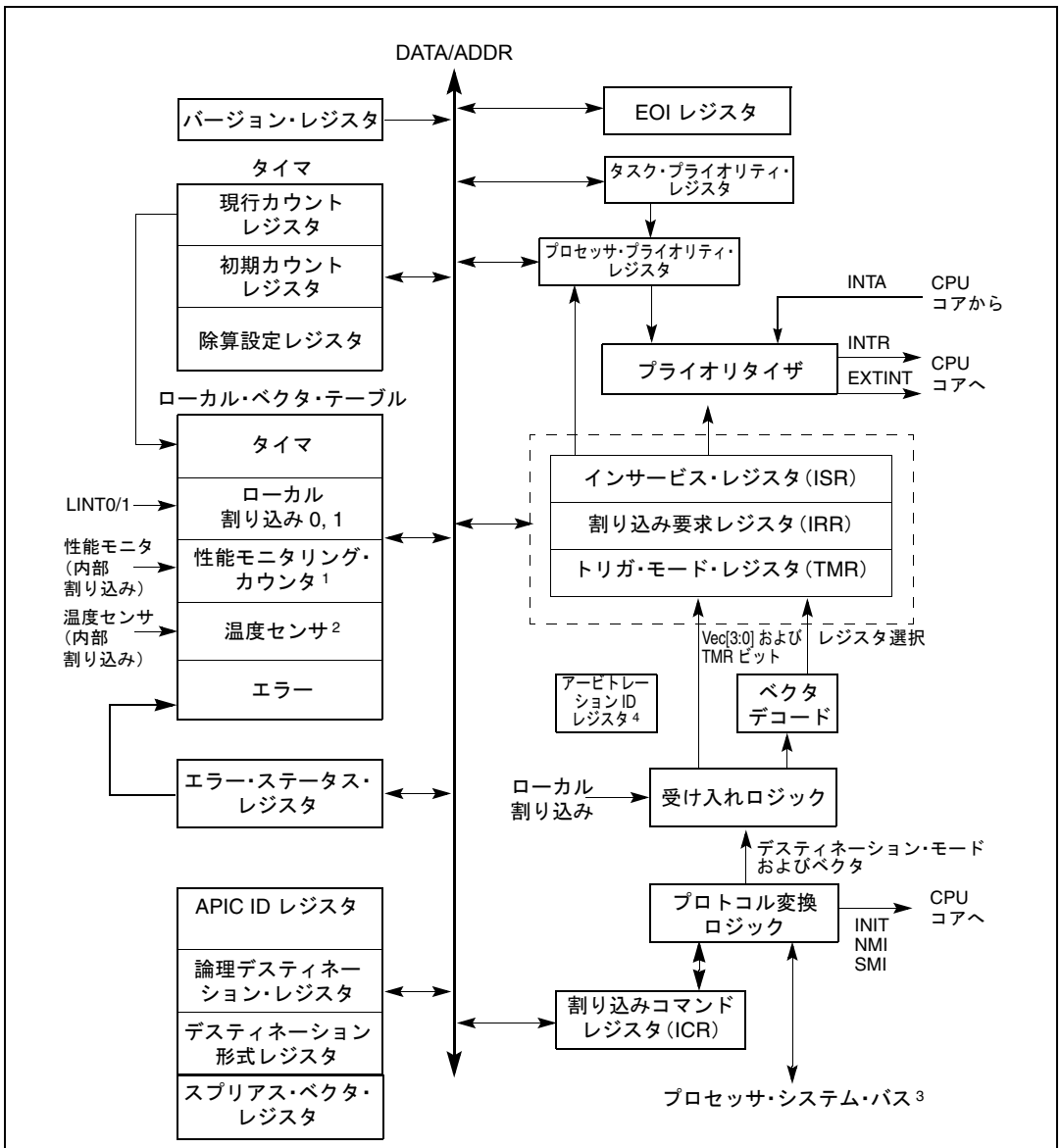
8.4.1. ローカル APIC のブロック図

図 8-4. に、ローカル APIC の機能ブロック図を示す。ソフトウェアは、ローカル APIC レジスタの読み出しと書き込みを行うと、ローカル APIC と対話する。APIC レジスタは、プロセッサの物理アドレス空間の 4K バイト領域にメモリマップされ、初期開始アドレスは FEE00000H である。APIC が正常に動作するためには、このアドレス空間が、ストロング・キャッシュ不可（UC）メモリとして指定されたメモリ領域にマッピングされる必要がある。10.3. 節「有効なキャッシングの方法」を参照のこと。

MP システム構成では、システムバス上のすべての IA-32 プロセッサの APIC レジスタは、最初は物理アドレス空間の同じ 4K バイト領域にマッピングされる。ソフトウェアは、この初期マッピングを変更して、すべてのローカル APIC を他の 4K バイト領域に割り当てたり、各ローカル APIC の APIC レジスタをそれぞれの 4K バイト領域にマッピングすることができる。特定のプロセッサの APIC レジスタのベースアドレスの再配置については、8.4.5. 項「ローカル APIC レジスタの再配置」を参照のこと。

注記

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、APIC は、4K バイトの APIC レジスタ空間内のアドレスに対するすべてのメモリアクセスを内部で処理するため、外部バスサイクルは発生しない。しかし、オンチップ APIC を使用するインテル® Pentium® プロセッサでは、APIC レジスタ空間へのアクセスによってバスサイクルが発生する。したがって、ソフトウェアをインテル Pentium プロセッサ上で実行する場合は、システム・ソフトウェアは、APIC レジスタ空間を通常のシステムメモリに明示的にマッピングしてはならない。この注意に従わないと、無効オペコード例外（#UD）が発生したり、実行の結果が予測不可能になる。



1. P6 ファミリ・プロセッサで導入された。
2. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサで導入された。
3. P6 ファミリ・プロセッサおよびインテル Pentium プロセッサの 3 線式 APIC バス。
4. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではサポートしていない。

図 8-4. ローカル APIC の構造

表 8-1. は、4K バイトの APIC レジスタ空間に対する APIC レジスタのマッピングを示している。すべてのレジスタは、32 ビット幅、64 ビット幅、または 256 ビット幅で、すべて 128 ビット境界にアライメントされている。すべての 32 ビット・レジスタへのアクセスには、128 ビットにアライメントされた 32 ビット・ロードまたはストアを使

用しなければならない。より幅の広いレジスタ（64 ビットまたは 256 ビット）へのアクセスには、複数の 32 ビット・ロードまたはストアを使用しなければならない（最初のアクセスは、128 ビットにアライメントされていなければならない）。APIC アドレス空間にアクセスする MOV 命令と LOCK プリフィックスを組み合わせて使用した場合、LOCK プリフィックスは無視される。つまり、ロック操作は行われない。表 8-1. に示すすべてのレジスタについて、本章の以下の各項で説明する。

表 8-1. ローカル APIC レジスタのアドレスマップ

アドレス	レジスタ名	ソフトウェア読み出し / 書き込み
FEE0 0000H	予約済み	
FEE0 0010H	予約済み	
FEE0 0020H	ローカル APIC ID レジスタ	読み出し / 書き込み
FEE0 0030H	ローカル APIC バージョン・レジスタ	読み出し専用
FEE0 0040H	予約済み	
FEE0 0050H	予約済み	
FEE0 0060H	予約済み	
FEE0 0070H	予約済み	
FEE0 0080H	タスク・プライオリティ・レジスタ（TPR）	読み出し / 書き込み
FEE0 0090H	アービトレーション・プライオリティ・レジスタ[1]（APR）	読み出し専用
FEE0 00A0H	プロセッサ・プライオリティ・レジスタ（PPR）	読み出し専用
FEE0 00B0H	EOI レジスタ	書き込み専用
FEE0 00C0H	予約済み	
FEE0 00D0H	論理デスティネーション・レジスタ	読み出し / 書き込み
FEE0 00E0H	デスティネーション形式レジスタ	ビット 0-27 は読み出し専用、ビット 28 ～ 31 は読み出し / 書き込み
FEE0 00F0H	スプリアス割り込みベクタレジスタ	ビット 0 ～ 8 は読み出し / 書き込み、ビット 9 ～ 31 は読み出し専用
FEE0 0100H ～ FEE0 0170H	インサービス・レジスタ（ISR）	読み出し専用
FEE0 0180H ～ FEE0 01F0H	トリガ・モード・レジスタ（TMR）	読み出し専用
FEE0 0200H ～ FEE0 0270H	割り込み要求レジスタ（IRR）	読み出し専用
FEE0 0280H	エラー・ステータス・レジスタ	読み出し専用
FEE0 0290H ～ FEE0 02F0H	予約済み	
FEE0 0300H	割り込みコマンドレジスタ（ICR）[0-31]	読み出し / 書き込み
FEE0 0310H	割り込みコマンドレジスタ（ICR）[32-63]	読み出し / 書き込み

表 8-1. ローカル APIC レジスタのアドレスマップ（続き）

アドレス	レジスタ名	ソフトウェア読み出し / 書き込み
FEE0 0320H	LVT タイマレジスタ	読み出し / 書き込み
FEE0 0330H	LVT 温度センサレジスタ [2]	読み出し / 書き込み
FEE0 0340H	LVT 性能モニタリング・カウンタ・レジスタ [3]	読み出し / 書き込み
FEE0 0350H	LVT LINT0 レジスタ	読み出し / 書き込み
FEE0 0360H	LVT LINT1 レジスタ	読み出し / 書き込み
FEE0 0370H	LVT エラーレジスタ	読み出し / 書き込み
FEE0 0380H	初期カウントレジスタ（タイマ用）	読み出し / 書き込み
FEE0 0390H	現行カウントレジスタ（タイマ用）	読み出し専用
FEE0 03A0H ～ FEE0 03D0H	予約済み	
FEE0 03E0H	除算設定レジスタ（タイマ用）	読み出し / 書き込み
FEE0 03F0H	予約済み	

注 :

1. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではサポートしていない。
2. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサで導入された。この APIC レジスタとそれに関連する機能は、プロセッサによって異なり、今後の IA-32 プロセッサではサポートされない可能性がある。
3. インテル Pentium Pro プロセッサで導入された。この APIC レジスタとそれに関連する機能は、プロセッサによって異なり、今後の IA-32 プロセッサではサポートされない可能性がある。

注記

表 8-1. に示したローカル APIC レジスタは、MSR ではない。ローカル APIC のプログラミングに関連する MSR は、IA32_APIC_BASE MSR だけである（8.4.3. 項「ローカル APIC の有効化または無効化」を参照）。

8.4.2. ローカル APIC の有無

P6 ファミリ・プロセッサから、オンチップ・ローカル APIC の有無を、CPUID 命令を使用して検出できるようになった。EAX レジスタのソース・オペランドを 1 にして CPUID 命令を実行したとき、EDX レジスタに返される CPUID 機能フラグのビット 9 に、ローカル APIC の有無が示される（ローカル APIC があればセットされ、ローカル APIC がなければクリアされる）。

8.4.3. ローカル APIC の有効化または無効化

ローカルAPICは、次のいずれかの方法で、有効または無効にできる。

- IA32_APIC_BASE MSR内のAPICグローバル有効/無効フラグの使用（図8-5.を参照）
- スプリアス割り込みベクタレジスタ内の APIC ソフトウェア有効 / 無効フラグの使用（図 8-22. を参照）

IA32_APIC_BASE MSR 内の APIC グローバル有効 / 無効フラグを使用して、ローカル APIC を永続的に無効にできる。電源投入またはリセット後、このフラグがセットされ、ローカルAPICが有効になる。次の電源投入またはリセットまでローカル APIC を永続的に無効にするには、ソフトウェアがこのフラグをクリアする。

このフラグがクリアされると、プロセッサは、オンチップ APIC を持たない IA-32 プロセッサ（例えば、Intel486™ プロセッサ）と機能的に同等になる。この状態では、APIC の CPUID 機能フラグ（EDX レジスタのビット 9）は 0 にクリアされる（8.4.2. 項「ローカル APIC の有無」を参照）。また、IA32_APIC_BASE MSR 内の APIC グローバル有効 / 無効フラグがクリアされた場合、このフラグを再びセットするには、電源投入またはリセット操作を行わなければならない。

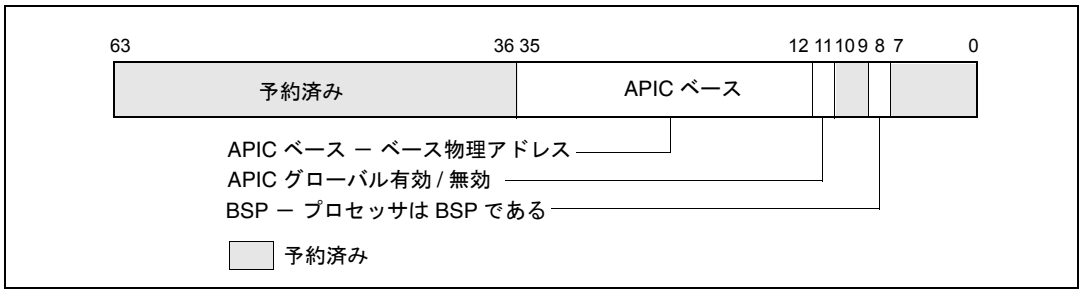


図 8-5. IA32_APIC_BASE MSR

インテル® Pentium® プロセッサでは、電源投入時またはリセット時に APICEN ピン（PICD1 ピンと共有される）を使用して、ローカル APIC を無効にできる。

IA32_APIC_BASE MSR 内の APIC グローバル有効 / 無効フラグがクリアされていない場合、ソフトウェアは、スプリアス割り込みベクタレジスタ内の APIC ソフトウェア有効 / 無効フラグをクリアすれば、いつでも一時的にローカル APIC を無効にできる（図 8-22. を参照）。ソフトウェア的に無効にされたローカル APIC の状態については、8.4.7.2. 項「ソフトウェア的に無効にされたローカル APIC の状態」を参照のこと。ローカル APIC がソフトウェア的に無効にされた場合、APIC ソフトウェア有効 / 無効フラグを 1 にセットすると、いつでもローカル APIC を再び有効にできる。

LVT 内の各エントリは、マスクビットを持つ。このビットを使用して、選択したローカル割り込みソース（LINT0 ピンと LINT1 ピン、APIC タイマ、性能モニタリング・カ

ウンタ、温度センサ、または内部 APIC エラー・ディテクタ）からの割り込みがプロセッサに伝達されないように設定できる。

8.4.4. ローカル APIC のステータスと位置

ローカル APIC のステータスと位置は、IA32_APIC_BASE MSR（P6 ファミリ・プロセッサでは、APIC_BASE_MSR）に格納される。この MSR は、MSR アドレス 27 にある。図 8-5. は、この MSR 内のビットのエンコーディングを示している。これらのビットの機能は、次のとおりである。

BSP フラグ、ビット 8
: ブートストラップ・プロセッサ（BSP）かどうかを示す（7.5. 節「マルチプロセッサ（MP）初期化」を参照）。電源投入またはリセット後、BSP として選択されているプロセッサでは、このフラグは 1 にセットされる。その他の各アプリケーション・プロセッサ（AP）では、このフラグは 0 にクリアされる。

APIC グローバル有効フラグ、ビット 11
: ローカル APIC を有効（1）または無効（0）にする（8.4.3. 項「ローカル APIC の有効化または無効化」を参照）。このフラグは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサで利用可能である。今後の IA-32 プロセッサでこのフラグを利用できるかどうか、またフラグの位置が同じかどうかは保証されない。

APIC ベース・フィールド、ビット 12 ～ 35
: APIC レジスタのベースアドレスを指定する。この 24 ビット値が下位側で 12 ビット拡張されて、ベースアドレスが生成される。これによって、アドレスは自動的に 4K バイト境界にアライメントされる。電源投入またはリセット後、このフィールドは FEE00000H に設定される。

IA32_APIC_BASE MSR のビット 0 ～ 7、ビット 9 と 10、ビット 36 ～ 63 は予約済みである。

8.4.5. ローカル APIC レジスタの再配置

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、IA32_APIC_BASE MSR の 24 ビットのベース・アドレス・フィールドの値を変更すると、APIC レジスタの開始アドレスを FEE00000H から他の物理アドレスに変更できる。APIC アーキテクチャがこのように拡張されたため、既存のシステムのメモリマップとの競合を解消できる。また、MP システム内の個々のプロセッサが、それぞれの APIC レジスタを、物理メモリ内の異なる位置にマップできる。

8.4.6. ローカル APIC ID

電源投入時に、システム・ハードウェアは、システムバス上（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの場合）または APIC バス上（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサの場合）の各ローカル APIC に対して、個別の APIC ID を割り当てる。ハードウェアによって割り当てられる APIC ID は、システム構成に基づいて指定され、ソケット位置とクラスタ情報のエンコーディングを含む（図 7-2. を参照）。

MP システム内では、ローカル APIC ID は、BIOS とオペレーティング・システムによってプロセッサ ID としても使用される。

プロセッサは、ピン A11# および A12# と、ピン BR0# ～ BR3#（インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサの場合）または BE0# ～ BE3#（インテル Pentium プロセッサの場合）のサンプリングにより、ハードウェアによって割り当てられた APIC ID を受信する。これらのポートからラッチされた APIC ID は、ローカル APIC ID レジスタの APIC ID フィールドに格納され（図 8-6. を参照）、プロセッサの初期 APIC ID として使用される。この値は、EAX レジスタのソース・オペランドの値を 1 にして CPUID 命令を実行したとき、EBX レジスタに返される。

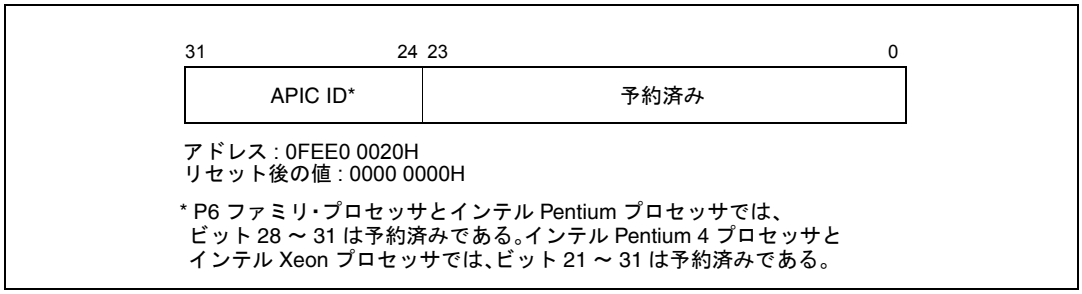


図 8-6. ローカル APIC ID レジスタ

P6 ファミリ・プロセッサおよびインテル Pentium プロセッサでは、ローカル APIC ID レジスタ内のローカル APIC ID フィールドは 4 ビットであり、エンコーディング 0H ～ EH を使用して、APIC バスに接続された 15 個の異なるプロセッサを個別に識別できる。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、xAPIC 仕様によってローカル APIC ID フィールドが 8 ビットに拡張され、システム内の最大 255 個までのプロセッサを識別できる。

電源投入またはハードウェア・リセット後、ソフトウェア（通常は BIOS ソフトウェア）は、システム内の各プロセッサのローカル APIC ID レジスタ内の APIC ID フィールドを変更できる。APIC ID を変更する場合は、ソフトウェアが、システム全体で各ローカル APIC が固有の APIC ID を持つように保証しなければならない。

8.4.7. ローカル APIC の状態

以下の各項では、電源投入またはリセット後、ローカル APIC がソフトウェア的に無効にされた後、INIT リセット後、INIT デアサート・メッセージ後の、ローカル APIC およびローカル APIC レジスタの状態について説明する。

8.4.7.1. 電源投入またはリセット後のローカル APIC の状態

プロセッサの電源投入またはリセット後、ローカル APIC およびローカル APIC レジスタの状態は次のようになる。

- IRR、ISR、TMR、ICR、LDR、TPR レジスタ、タイマ初期カウントレジスタとタイマ現行カウントレジスタ、除算設定レジスタは、すべて 0 にリセットされる。
- DFR レジスタは、すべて 1 にリセットされる。
- LVT レジスタのエントリはすべて 0 にリセットされるが、マスクビットだけは 1 にセットされる。
- ローカル APIC バージョン・レジスタは影響を受けない。
- ローカル APIC ID レジスタは、固有の APIC ID に設定される（インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサのみ）。Arb ID レジスタは、APIC ID レジスタ内の値に設定される。
- スプリアス割り込みベクタレジスタは、0000 00FFH に初期化される。ビット 8 を 0 に設定すると、ソフトウェアがローカル APIC を無効にする。
- プロセッサがシステム内に 1 個しかない場合や、MP システム内で BSP として指定されている場合は（7.5.1. 項「BSP プロセッサと AP プロセッサ」を参照）、ローカル APIC は、INIT および NMI メッセージと INIT# および STPCLK# 信号に対して正常に応答する。プロセッサが MP システム内で AP として指定されている場合は、ローカル APIC は、BSP の場合と同じメッセージと信号に応答する以外に、SIPI メッセージにも応答する。ただし、P6 ファミリ・プロセッサの場合に限り、AP は STPCLK# 信号に応答しない。

8.4.7.2. ソフトウェア的に無効にされたローカル APIC の状態

スプリアス割り込みベクタレジスタ内の APIC ソフトウェア有効 / 無効フラグが（電源投入時またはリセット時にクリアされるのではなく）明示的にクリアされた場合、ローカル APIC は一時的に無効にされる（8.4.3. 項「ローカル APIC の有効化または無効化」を参照）。ソフトウェア的に無効にされた状態では、ローカル APIC の動作と応答は次のようになる。

- ローカル APIC は、INIT、NMI、SMI、SIPI メッセージに対して正常に応答する。

- IRR および ISR レジスタ内の未処理の割り込みは保留され、CPU によるマスキングまたは処理を必要とする。
- この状態でも、ローカル APIC は IPI を発行できる。IPI 機構によって割り込みを送信したくない場合は、ソフトウェア側で、IPI 機構と ICR レジスタによって IPI を発行しないようにする必要がある。
- IPI の受理または伝送の途中でローカル APIC が無効にされた場合は、その操作が完了してから、ローカル APIC はソフトウェア無効状態に移行する。
- すべての LVT エントリのマスクビットがセットされる。これらのビットをリセットしようとしても無視される。
- （インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサ）ローカル APIC は、自分のアービトレーション ID とシステム内の他のプロセッサの間の同期を保つために、すべてのバス・メッセージの監視を続ける。

8.4.7.3. INIT リセット後のローカル APIC の状態（"wait-for-SIPI" 状態）

プロセッサの INIT リセットは、次のいずれかの方法で開始できる。

- プロセッサの INIT# ピンをアサートする
- プロセッサに INIT IPI を送信する（伝達モードを INIT に設定して、プロセッサに IPI を送信する）

プロセッサは、上のいずれかの機構によって INIT を受け取ると、それに反応してプロセッサ・コアとローカル APIC の初期化プロセスを開始する。INIT リセット後のローカル APIC の状態は、APIC ID レジスタとアービトレーション ID レジスタが影響を受けないことを除いて、電源投入またはハードウェア・リセット後の状態と同じである。この状態は、"wait-for-SIPI" 状態とも呼ばれる。MP システム内で電源投入またはリセット後に INIT を発行した場合の影響については、7.5.2. 項「インテル® Xeon™ プロセッサの MP 初期化プロトコルの必要条件および制約」を参照のこと。

8.4.7.4. INIT デアサート IPI 受信後のローカル APIC の状態

（INIT デアサート IPI は、インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサでのみサポートされる。）INIT デアサート IPI は、アービトレーション ID レジスタに APIC ID レジスタの値が再ロードされることを除いて、APIC の状態に影響を与えない。

8.4.8. ローカル APIC バージョン・レジスタ

ローカル APIC には、ハードワイヤード・バージョン・レジスタが含まれている。ソフトウェアは、このレジスタを使用して、APIC のバージョンを識別できる（図 8-7. を参照）。また、このレジスタは、特定のプロセッサのローカル・ベクタ・テーブル（LVT）内のエントリ数を指定する。ローカル APIC バージョン・レジスタ内のフィールドは、次のとおりである。

バージョン（Version）ローカル APIC のバージョン番号。

1XH	ローカル APIC。インテル ® Pentium® 4 プロセッサおよびインテル ® Xeon™ プロセッサでは、14H が返される。
0XH	82489DX 外部 APIC。
20H ～ FFH	予約済み。

最大 LVT エントリ（Max LVT Entry）
LVT エントリの数から 1 を引いた値を表示する。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ（LVT エントリの数は 6 個）では、Max LVT フィールドの戻り値は 5 である。P6 ファミリ・プロセッサ（LVT エントリの数は 5 個）では、戻り値は 4 である。インテル ® Pentium® プロセッサ（LVT エントリの数は 4 個）では、戻り値は 3 である。

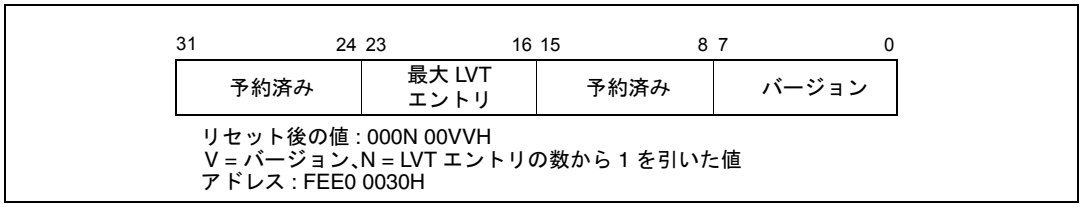


図 8-7. ローカル APIC バージョン・レジスタ

8.5. ローカル割り込みの処理

以下の各項では、ローカル割り込みの処理のためにローカル APIC 内に用意されている機能について説明する。ローカル割り込みのソースは、プロセッサの LINT0 ピンと LINT1 ピン、APIC タイマ、性能モニタリング・カウンタ、温度センサ、内部 APIC エラー・ディテクタである。ローカル割り込み処理機能には、LVT、エラー・ステータス・レジスタ（ESR）、除算設定レジスタ（DCR）、および初期カウントレジスタと現行カウントレジスタがある。

8.5.1. ローカル・ベクタ・テーブル

ローカル・ベクタ・テーブル（LVT）により、ソフトウェアは、ローカル割り込みをプロセッサ・コアに伝達する方法を指定できる。LVT は、次の 5 つの 32 ビット APIC レジスタで構成される（図 8-8. を参照）。各レジスタが各ローカル割り込みに対応する。

- LVT タイマレジスタ（FEE0 0320H）－ APIC タイマが割り込みを通知したときの割り込みの伝達方法を指定する（8.5.4. 項「APIC タイマ」を参照）。
- LVT 温度モニタレジスタ（FEE0 0330H）－ 温度センサが割り込みを生成したときの割り込みの伝達方法を指定する（13.16.2. 項「温度モニタ」を参照）。この LVT エントリはアーキテクチャ機能ではなく、プロセッサによって異なる。サポートされている場合、このエントリは常にベースアドレス FEE0 0330H に置かれる。
- LVT 性能カウンタレジスタ（FEE0 0340H）－ 性能カウンタがオーバーフロー時に割り込みを生成したときの割り込みの伝達方法を指定する（15.9.6.9. 項「オーバーフロー時の割り込みの生成」を参照）。この LVT エントリはアーキテクチャ機能ではなく、プロセッサによって異なる。サポートされている場合でも、ベースアドレス FEE0 0340H に置かれるかどうかは保証されない。
- LVT LINT0 レジスタ（FEE0 0350H）－ LINT0 ピンで割り込みが発生したときの割り込みの伝達方法を指定する。
- LVT LINT1 レジスタ（FEE0 0360H）－ LINT1 ピンで割り込みが発生したときの割り込みの伝達方法を指定する。
- LVT エラーレジスタ（FEE0 0370H）－ APIC が内部エラーを検出したときの割り込みの伝達方法を指定する（8.5.3. 項「エラー処理」を参照）。

注記

LVT 性能カウンタレジスタとそれに関連する割り込みは、P6 プロセッサで導入され、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでもサポートされている。LVT 温度モニタレジスタとそれに関連する割り込みは、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサで導入された。

ただし、図 8-8. に示すように、いくつかのエントリでは一部のフィールドおよびフラグが予約済みになっており、利用できない。

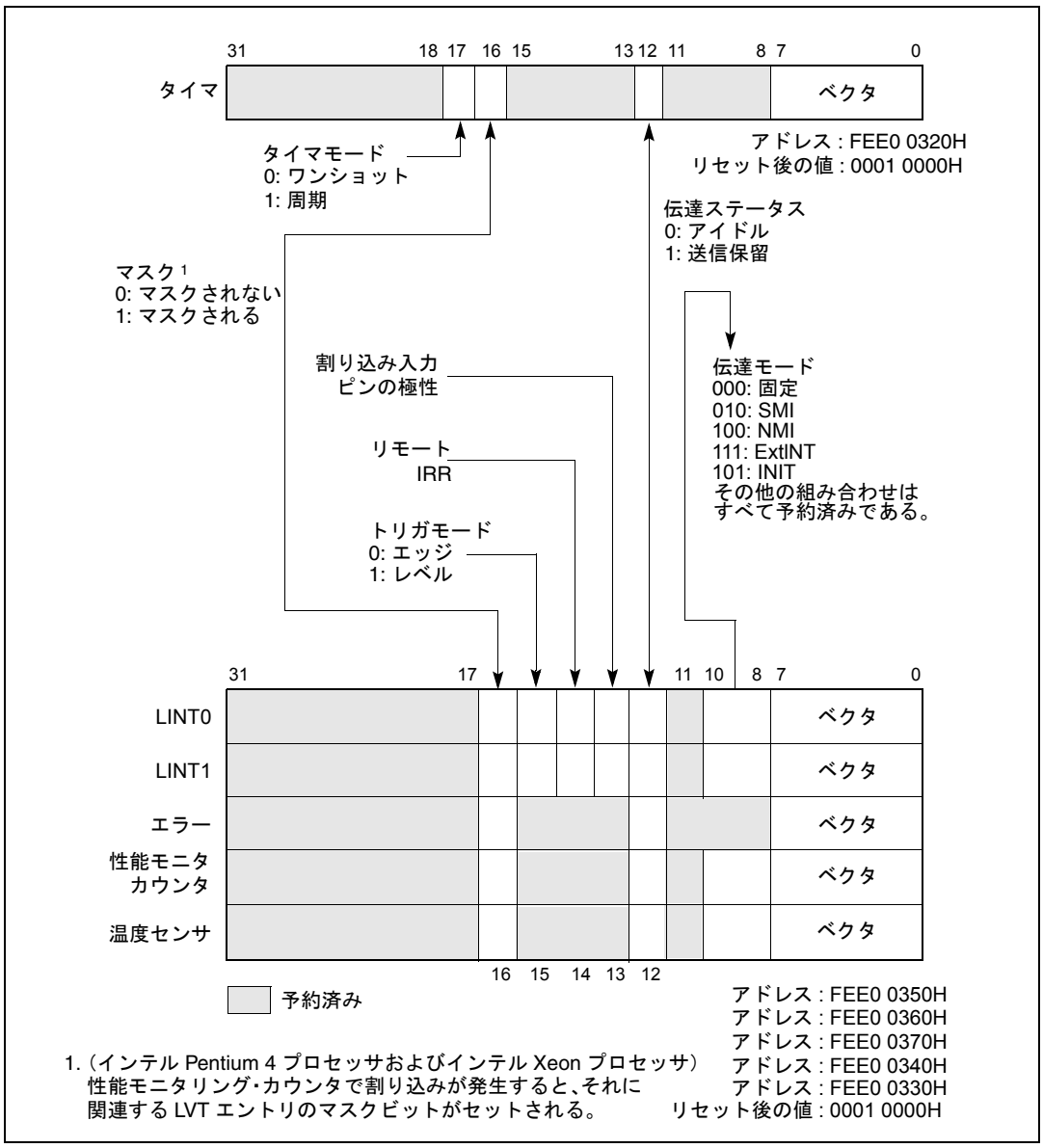


図 8-8. ローカル・ベクタ・テーブル（LVT）

LVTテーブルのレジスタ内で指定できる設定情報は以下のとおりである。

ベクタ　割り込みのベクタ番号。

伝達モード　プロセッサに送信される割り込みのタイプを指定する。ただし、一部の伝達モードは、特定のトリガモードと組み合わせて使用した場合にのみ正常に機能する。指定できる伝達モードは次のとおりである。

- **000（固定）**　ベクタ・フィールドで指定された割り込みを伝達する。
- **010（SMI）**　プロセッサのローカル SMI 信号パスを介して、SMI 割り込みをプロセッサ・コアに伝達する。この伝達モードを使用する場合は、将来の互換性を保証するために、ベクタ・フィールドを 00H に設定する必要がある。
- **100（NMI）**　NMI 割り込みをプロセッサに伝達する。ベクタ情報は無視される。
- **101（INIT）**　INIT 要求をプロセッサ・コアに伝達し、プロセッサに INIT を実行させる。この伝達モードを使用する場合は、将来の互換性を保証するために、ベクタ・フィールドを 00H に設定する必要がある。
- **111（ExtINT）**　プロセッサは、外部接続された（8259A 互換）割り込みコントローラから割り込みが発信されたかのように、割り込みに応答する。ExtINT に対応する特殊な INTA バスサイクルが、外部コントローラに転送される。ベクタ情報は、外部コントローラが供給するものとする。APIC アーキテクチャは、システム内の ExtINT ソースを 1 つしかサポートしない。このソースは、通常は互換ブリッジに組み込まれる。

伝達ステータス（読み出し専用）
割り込み伝達ステータスを、次のように表示する。

- **0（アイドル）**　この割り込みソースは現在割り込みを発信していない。または、このソースから発信された以前の割り込みは、すでにプロセッサ・コアに伝達され、受け入れられた。
- **1（送信保留）**　このソースから発信された割り込みはプロセッサ・コアに伝達されたが、まだ受け入れられていない（8.5.5. 項「ローカル割り込みの受け入れ」を参照）。

割り込み入力ピンの極性
対応する割り込みピンの極性を、アクティブ・ハイ（0）またはアクティブ・ロー（1）として指定する。

リモート IRR フラグ（読み出し専用）
: 固定モードのレベルトリガ割り込みの場合、このフラグは、ローカル APIC がサービスのために割り込みを受け入れた時点でセットされ、プロセッサから EOI コマンドを受信した時点でリセットされる。エッジトリガ割り込みの場合や、固定モード以外の伝達モードの場合は、このフラグの意味は未定義である。

トリガモード
: ローカル LINT0 ピンと LINT1 ピンのトリガモードを、エッジ・センシティブ（0）またはレベル・センシティブ（1）として選択する。このフラグは、伝達モードが固定（Fixed）のときにのみ使用される。伝達モードが NMI、SMI、または INIT のときは、トリガモードは常にエッジ・センシティブになる。伝達モードが ExtINT のときは、トリガモードは常にレベル・センシティブになる。タイマ割り込みとエラー割り込みは、常にエッジ・センシティブとして処理される。

: ローカル APIC が I/O APIC と組み合わされていない場合、固定伝達モードが選択されているときは、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサは、トリガモードの選択とは無関係に、常にレベル・センシティブなトリガを使用する。

マスク
: 割り込みマスク。(0) の場合は割り込みの受け入れが有効になり、(1) の場合は割り込みの受け入れが行われない。ローカル APIC は、性能モニタリング・カウンタ割り込みを処理するとき、対応する LVT エントリのマスクフラグを自動的にセットする。このフラグは、ソフトウェアがクリアするまでセットされたままになる。

タイマモード
: タイマモードをワンショット・モード（0）または周期モード（1）として選択する（8.5.4. 項「APIC タイマ」を参照）。

8.5.2. 有効な割り込みベクタ

IA-32 アーキテクチャでは、0 ～ 255 の範囲で 256 個のベクタ番号が定義されている（5.2. 節「例外ベクタと割り込みベクタ」を参照）。ローカル APIC と I/O APIC は、これらのベクタのうち、（16 ～ 255 の範囲の）240 個を有効な割り込みとしてサポートしている。

0 ～ 15 の範囲の割り込みベクタがローカル APIC によって送信または受信されると、ローカル APIC はエラー・ステータス・レジスタ内に無効ベクタを表示する（8.5.3. 項「エラー処理」を参照）。また、IA-32 アーキテクチャは、ベクタ 16 ～ 31 を、事前に定義された割り込み、例外、およびインテルで予約済みのエンコーディング用に確保している（表 5-1. を参照）。しかし、ローカル APIC は、この範囲内のベクタを無効ベクタとしては扱わない。

伝達モードが固定モード（ビット 8 ～ 11=0）のとき、無効なベクタ値（0 ～ 15）が LVT エントリに書き込まれると、APIC は、マスクビットがセットされているか、また割

り込みが入力上で実際に検出されたかに関係なく、無効ベクタエラーを報告する場合がある。

8.5.3. エラー処理

ローカル APIC は、エラー・ステータス・レジスタ（ESR）を使用して、割り込みの処理中に検出したエラーを記録する（図 8-9. を参照）。ローカル APIC が ESR 内のエラービットのうち 1 つをセットすると、APIC エラー割り込みが発生する。LVT エラーレジスタを使用して、APIC エラーが検出されたときにプロセッサ・コアに伝達される割り込みベクタを選択できる。また、LVT エラーレジスタを使用して、APIC エラー割り込みをマスクすることもできる。

ESR フラグの機能は次のとおりである。

送信チェックサム・エラー
（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサのみ）ローカル APIC が APIC バス上に送信したメッセージに、チェックサム・エラーが検出されたときにセットされる。

受信チェックサム・エラー
（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサのみ）ローカル APIC が APIC バス上で受信したメッセージに、チェックサム・エラーが検出されたときにセットされる。

送信受け入れエラー
（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサのみ）ローカル APIC が送信したメッセージが、APIC バス上のどの APIC にも受け入れられなかったときにセットされる。

受信受け入れエラー
（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサのみ）ローカル APIC が受信したメッセージが、そのローカル APIC を含む APIC バス上のどの APIC にも受け入れられなかったときにセットされる。

送信無効ベクタ
ローカル APIC が送信しているメッセージ内で無効ベクタが検出されたときにセットされる。

受信無効ベクタ
ローカル APIC が受信したメッセージ内で無効ベクタが検出されたときにセットされる（ローカル・ベクタ・テーブル割り込みまたは自己割り込み内で無効ベクタコードが検出された場合を含む）。

無効レジスタアドレス
（インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのみ）プロセッサが、そのプロセッサのローカル APIC レジスタアドレス空間（つまり、IA32_APIC_BASE MSR 内で指定される、APIC レジスタのベースアドレス + 4K バイトのアドレス範囲）内に存在しないレジスタにアクセスしようとしたときにセットされる。

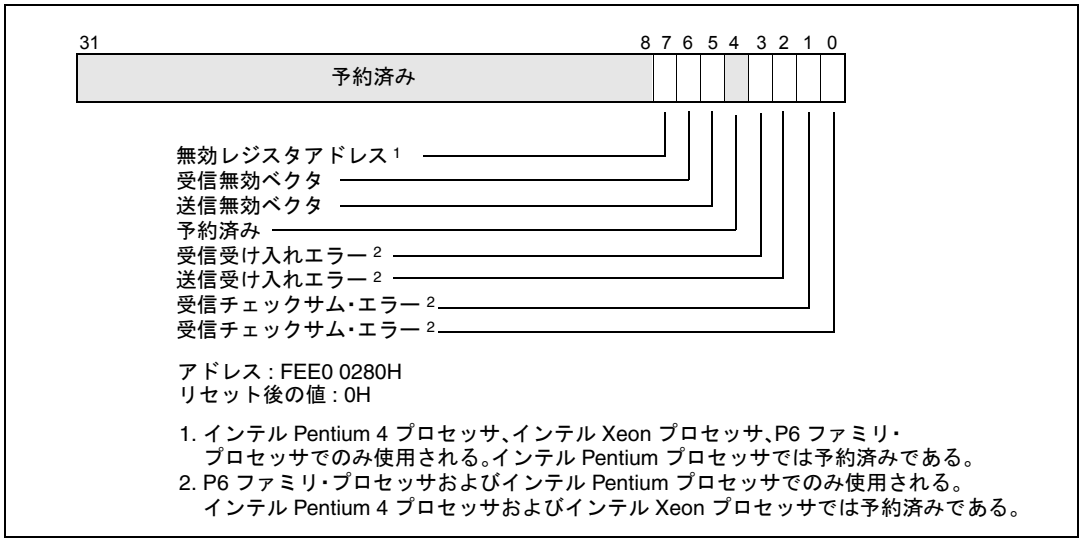


図 8-9. エラー・ステータス・レジスタ（ESR）

ESR は、読み出し / 書き込みレジスタである。このレジスタを更新するには、ESR の読み出しの直前に、任意の値を ESR に書き込む必要がある。この 1 回目の書き込みによって、ESR の内容は最新のエラー・ステータスで更新される。書き込みを連続して実行すると、ESR レジスタはクリアされる。

ESR レジスタ内でエラービットがセットされると、レジスタがクリアされるまで、エラービットはセットされたままになる。LVT エラーレジスタのマスクビットをセットすると、エラーが ESR に記録されなくなる。ただし、マスクビットをセットする前の ESR の状態は保持される。

8.5.4. APIC タイマ

ローカル APIC ユニットには、32 ビット・プログラマブル・タイマが組み込まれている。ソフトウェアは、このタイマを使用して、イベントまたは操作の時間を計測できる。このタイマは、除算設定レジスタ（図 8-10. を参照）、初期カウントレジスタと現行カウントレジスタ（図 8-11. を参照）、および LVT タイマレジスタ（図 8-8. を参照）の 4 つのレジスタをプログラムすることによって設定される。

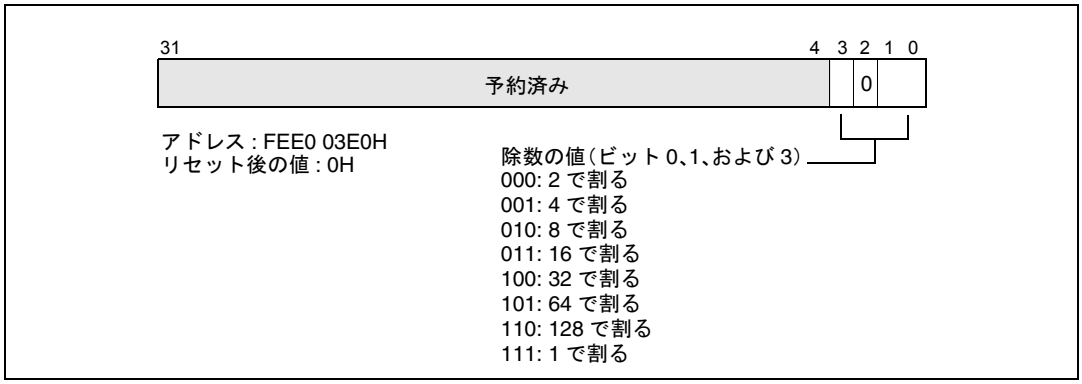


図 8-10. 除算設定レジスタ

図 8-11. 初期カウントレジスタと現行カウントレジスタ

APIC タイマのタイムベースは、除算設定レジスタ内で指定された値でプロセッサのバスクロックを割ることによって得られる。

APIC タイマは、LVT のタイマエントリで、ワンショット動作モードまたは周期動作モードに設定できる。ワンショット・モードでは、初期カウントレジスタがプログラムされると、タイマがスタートする。初期カウントの値が現行カウントレジスタにコピーされ、カウントダウンが開始される。タイマが 0 になると、タイマ割り込みが発生する。再びプログラムされるまで、タイマの値は 0 のままになる。

周期モードでは、カウントが 0 になってタイマ割り込みが発生すると、現行カウントレジスタの値が初期カウントレジスタから自動的に再ロードされ、カウントダウンが繰り返される。カウントダウン・プロセスの途中で初期カウントレジスタが設定されると、新しい初期カウント値にしたがってカウントが再び開始される。初期カウントレジスタは読み出し / 書き込みレジスタである。現行カウントレジスタは読み出し専用である。

LVT のタイマレジスタは、タイマカウントが 0 になったときに発生するタイマ割り込みでプロセッサに伝達されるベクタ番号を指定する。LVT タイマレジスタ内のマスクフラグを使用して、タイマ割り込みをマスクできる。

8.5.5. ローカル割り込みの受け入れ

ローカル割り込みがプロセッサ・コアに送信されると、図 8-17. の割り込み受け入れフローチャートで指定された受け入れ基準が、割り込みに対して適用される。受け入れられた割り込みは、IRR レジスタに記録され、優先度にしたがってプロセッサによって処理される（8.8.4. 項「固定割り込みの受け入れ」を参照）。受け入れられなかった割り込みは、ローカル APIC に返送され、再試行される。

8.6. プロセッサ間割り込みの発行

以下の各項では、ソフトウェアからプロセッサ間割り込み（IPI）を発行するために用意されている、ローカル APIC の機能について説明する。IPI を発行するための主なローカル APIC の機能は、割り込みコマンドレジスタ（ICR）である。ICR は、以下の目的で使用される。

- 割り込みを他のプロセッサに送信する。
- プロセッサが受信したが処理しなかった割り込みを、サービスのために他のプロセッサに転送する。
- 自分自身に割り込みをかける（自己割り込みを実行する）ようにプロセッサに指示する。
- スタートアップ IPI（SIPI）メッセージなどの特殊な IPI を、他のプロセッサまたは当該プロセッサに伝達する。

この機能によって発行された割り込みは、システムバス（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの場合）または APIC バス（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサの場合）を介して、システム内の他のプロセッサに伝達される。

8.6.1. 割り込みコマンドレジスタ（ICR）

割り込みコマンドレジスタ（ICR）は、64 ビットのローカル APIC レジスタである（図 8-12. を参照）。プロセッサ上で実行されるソフトウェアは、このレジスタを使用して、プロセッサ間割り込み（IPI）を指定し、システム内の他の IA-32 プロセッサに送信する。

IPI を送信するには、ソフトウェアは、ICR を設定し、送信される IPI メッセージのタイプとデスティネーション・プロセッサを指定しなければならない（ICR のすべてのフィールドは、ソフトウェアによって読み出し / 書き込み可能である。ただし、伝達ステータス・フィールドだけは読み出し専用である）。ICR の下位ダブルワードへの書き込みが行われると、IPI が送信される。

ICR は、以下のフィールドで構成される。

ベクタ　　送信される割り込みのベクタ番号。

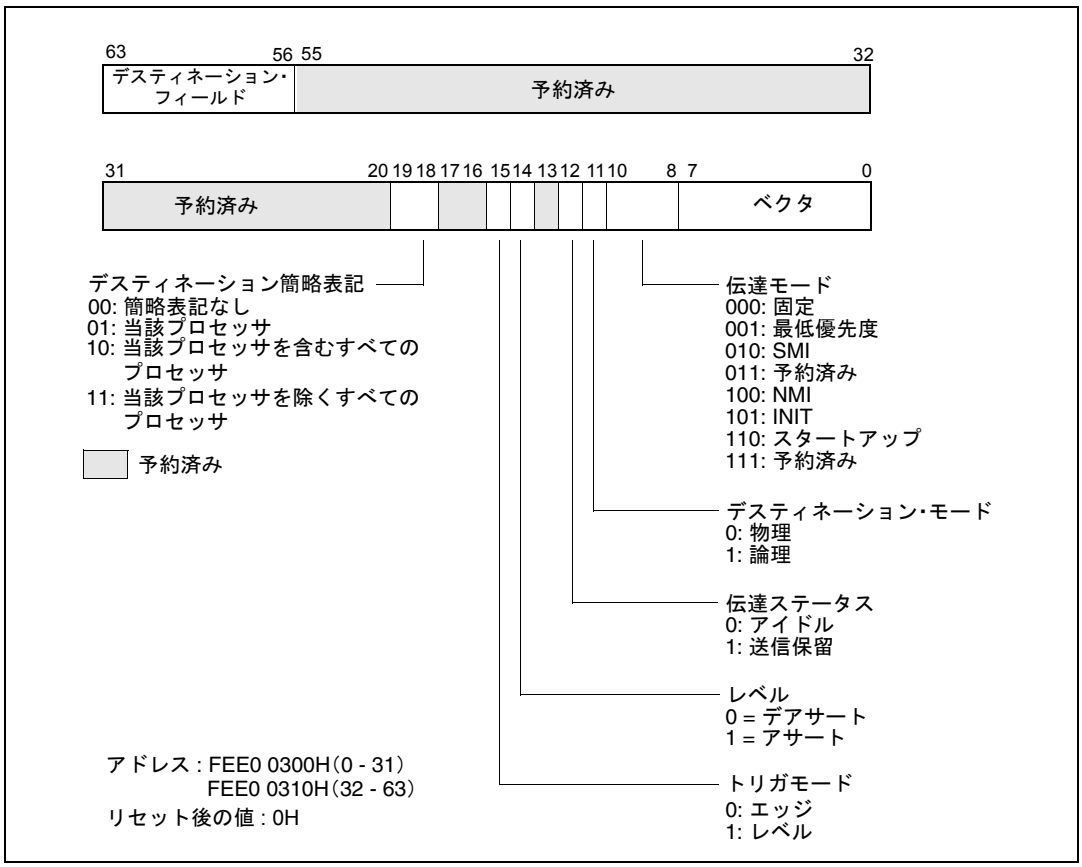


図 8-12. 割り込みコマンドレジスタ（ICR）

伝達モード　　送信される IPI のタイプを指定する。このフィールドは、IPI メッセージ・タイプ・フィールドとも呼ばれる。

000（固定）　　ベクタ・フィールドで指定された割り込みを送信先プロセッサに伝達する。

001（最低優先度）

固定モードと同じであるが、デスティネーション・フィールドで指定された一連のプロセッサのうち、最低の優先度で動作しているプロセッサに割り込みが伝達される点だけが異なる（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、この伝達モードの使用は推奨できない。複数の IPI が送信されるため、パフォーマンスが低下する場合がある）。

010（SMI）　SMI 割り込みを送信先プロセッサに伝達する。将来の互換性を保証するために、ベクタ・フィールドは 00H にプログラムしなければならない。

011（予約済み）

100（NMI）　NMI 割り込みを送信先プロセッサに伝達する。ベクタ情報は無視される。

101（INIT）　INIT 要求を送信先プロセッサに伝達し、INIT を実行させる。この IPI メッセージを送信すると、すべての送信先プロセッサが INIT を実行する。将来の互換性を保証するために、ベクタ・フィールドは 00H にプログラムしなければならない。

101（INIT レベル・デアサート）

（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではサポートしていない。）システム内のすべてのローカル APIC に同期メッセージを送信し、各 APIC の Arb ID レジスタに格納されているアービトレーション ID を、APIC ID の値に設定する（8.7. 節「システムバスと APIC バスのアービトレーション」を参照）。この伝達モードでは、レベルフラグは 0、トリガ・モード・フラグは 1 に設定されていなければならない。この IPI は、デスティネーション・フィールドやデスティネーション簡略表記フィールドの値に関係なく、すべてのプロセッサに送信される。ただし、ソフトウェアは、「当該プロセッサを含むすべてのプロセッサ」の簡略表記を指定する必要がある。

110（スタートアップ）

特殊な「スタートアップ」IPI（SIPI と呼ばれる）を送信先プロセッサに送信する。ベクタは、通常は、BIOS ブートストラップ・コードの一部であるスタートアップ・ルーチンを指す（7.5. 節「マルチプロセッサ（MP）初期化」を参照）。ただし、この伝達モードで送信される IPI は、発信元の APIC がこの IPI を伝達できなかった場合でも、自動的に再試行されない。SIPI が正常に伝達されたかどうかを確認し、必要に応じて SIPI を再発行するのは、ソフトウェアの役割である。

デスティネーション・モード

物理（0）デスティネーションまたは論理（1）デスティネーションを選択する（8.6.2. 項「IPI のデスティネーションの指定」を参照）。

伝達ステータス（読み出し専用）
IPI の伝達ステータスを次のように表示する。

0（アイドル） このローカル APIC は現在 IPI の処理を行っていない。または、このローカル APIC から送信された以前の IPI は、すでに送信先プロセッサに伝達され、受け入れられた。

1（送信保留） このローカル APIC から最後に送信された IPI は、まだ送信先プロセッサによって受け入れられていない。

レベル INIT レベル・デアサート伝達モードでは、このフラグは 0 に設定されていなければならない。その他のすべての伝達モードでは、このフラグは 1 に設定されていなければならない（このフラグは、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは無意味であり、常に 1 として発行される）。

トリガモード INIT レベル・デアサート伝達モードの場合は、エッジ（0）またはレベル（1）のトリガモードを選択する。その他のすべての伝達モードでは、このフラグは無視される（このフラグは、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは無意味であり、常に 1 として発行される）。

デスティネーション簡略表記
簡略表記を使用して割り込みのデスティネーションを指定するかどうか、簡略表記を使用する場合は、どの簡略表記を使用するかを指定する。デスティネーション簡略表記は、8 ビットのデスティネーション・フィールドの代わりに使用される。ソフトウェアは、ICR の下位ダブルワードに 1 回書き込むと、デスティネーション簡略表記を送信できる。簡略表記は、ソフトウェア自己割り込み、送信元プロセッサを含むシステム内のすべてのプロセッサに対する IPI、送信元プロセッサを除くシステム内のすべてのプロセッサに対する IPI について定義されている。

00:（簡略表記なし）
デスティネーションはデスティネーション・フィールドで指定される。

01:（当該プロセッサ）
発行元の APIC が、IPI の唯一のデスティネーションになる。ソフトウェアは、このデスティネーション簡略表記を使用して、そのソフトウェアを実行しているプロセッサに割り込みをかけられる。APIC は、他の IPI メッセージと同様に、自己割り込みメッセージを内部で伝達することも、自己割り込みメッセージをバスに対して発行して「スヌープ」を実行することもできる。

10:（当該プロセッサを含むすべてのプロセッサ）
この IPI は、IPI を送信しているプロセッサを含む、システム内のすべてのプロセッサに送信される。APIC は、インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサではデスティネーション・フィールドを FH に設定し、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではデスティネーション・フィールドを FFH に設定して、IPI メッセージをブロードキャストする。

11:（当該プロセッサを除くすべてのプロセッサ）
この IPI は、IPI を送信しているプロセッサを除く、システム内のすべてのプロセッサに送信される。APIC は、物理デスティネーション・モードを使用して、インテル Pentium プロセッサおよび P6 ファミリ・プロセッサではデスティネーション・フィールドを FH に設定し、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではデスティネーション・フィールドを FFH に設定して、メッセージをブロードキャストする。（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、このデスティネーション簡略表記と最低優先度伝達モードを組み合わせて使用した場合、IPI は拒否されて発行元プロセッサに返送される。）

デスティネーション
送信先プロセッサを指定する。このフィールドは、デスティネーション簡略表記フィールドが 00B に設定されている場合にのみ使用される。デスティネーション・モードが物理モードに設定されている場合は、インテル Pentium プロセッサおよび P6 ファミリ・プロセッサではビット 56 ～ 59 で送信先プロセッサの APIC ID を指定し、
インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサではビット 56 ～ 63 で送信先プロセッサの APIC ID を指定する。デスティネーション・モードが論理モードに設定されている場合は、8 ビットのデスティネーション・フィールドの解釈は、システム内のすべてのプロセッサのローカル APIC の DFR レジスタと LDR レジスタの設定によって決まる（8.6.2. 項「IPI のデスティネーションの指定」を参照）。

ただし、ICR の設定の組み合わせの中には、有効でないものもある。表 8-2. は、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの ICR のフィールド設定の有効な組み合わせを示している。表 8-3. は、P6 ファミリ・プロセッサの ICR のフィールド設定の有効な組み合わせを示している。

表 8-2. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
ローカル xAPIC の割り込みコマンドレジスタの有効な組み合わせ

デスティネーション簡略表記	有効 / 無効	トリガモード	伝達モード	デスティネーション・モード
簡略表記なし	有効	エッジ	すべてのモード	物理または論理
簡略表記なし	無効[1]	レベル	すべてのモード	物理または論理
当該プロセッサ	有効	エッジ	固定	X[2]
当該プロセッサ	無効[1]	レベル	固定	X
当該プロセッサ	無効	X	最低優先度、NMI、INIT、SMI、スタートアップ	X
当該プロセッサを含むすべてのプロセッサ	有効	エッジ	固定	X
当該プロセッサを含むすべてのプロセッサ	無効[1]	レベル	固定	X
当該プロセッサを含むすべてのプロセッサ	無効	X	最低優先度、NMI、INIT、SMI、スタートアップ	X
当該プロセッサを除くすべてのプロセッサ	有効	エッジ	固定、最低優先度[3]、NMI、INIT、SMI、スタートアップ	X
当該プロセッサを除くすべてのプロセッサ	無効[1]	レベル	固定、最低優先度[3]、NMI、INIT、SMI、スタートアップ	X

注記：

1. これらの割り込みでは、トリガ・モード・ビットが 1（レベル）に設定されていても、ローカル xAPIC はこのビット設定を無効にして、エッジトリガ割り込みとして割り込みを発行する。
2. X ― 無関係。
3.「最低優先度」伝達モードと「当該プロセッサを除くすべてのプロセッサ」のデスティネーションを使用すると、IPI が発行元の APIC に再転送されるときがある。この動作は、「当該プロセッサを含むすべてのプロセッサ」デスティネーション・モードと基本的に同じである。

表 8-3. P6 ファミリ・プロセッサのローカル APIC の割り込みコマンドレジスタの
有効な組み合わせ

デスティネーション簡略表記	有効 / 無効	トリガモード	伝達モード	デスティネーション・モード
簡略表記なし	有効	エッジ	すべてのモード	物理または論理
簡略表記なし	有効[1]	レベル	固定、最低優先度、NMI	物理または論理
簡略表記なし	有効[2]	レベル	INIT	物理または論理
当該プロセッサ	有効	エッジ	固定	X[3]
当該プロセッサ	1	レベル	固定	X
当該プロセッサ	無効[4]	X	最低優先度、NMI、INIT、SMI、スタートアップ	X
当該プロセッサを含むすべてのプロセッサ	有効	エッジ	固定	X
当該プロセッサを含むすべてのプロセッサ	有効[1]	レベル	固定	X

表 8-3. P6 ファミリ・プロセッサのローカル APIC の割り込みコマンドレジスタの有効な組み合わせ（続き）

デスティネーション簡略表記	有効 / 無効	トリガモード	伝達モード	デスティネーション・モード
当該プロセッサを含むすべてのプロセッサ	無効[4]	X	最低優先度、NMI、INIT、SMI、スタートアップ	X
当該プロセッサを含むすべてのプロセッサ	有効	エッジ	すべてのモード	X
当該プロセッサを含むすべてのプロセッサ	有効[1]	レベル	固定、最低優先度、NMI	X
当該プロセッサを含むすべてのプロセッサ	無効[4]	レベル	SMI、スタートアップ	X
当該プロセッサを含むすべてのプロセッサ	有効[2]	レベル	INIT	X
X	無効[4]	レベル	SMI、スタートアップ	X

注記 :

1. レベルビットが 1 に設定されている場合はエッジトリガ割り込みとして扱われ、それ以外の場合は無視される。
2. レベルビットが 1 に設定されている場合はエッジトリガ割り込みとして扱われ、レベルビットが 0（デアサート）に設定されている場合は「INIT レベル・デアサート」メッセージとして扱われる。レベルビットを 0 に設定してよいのは、INIT レベル・デアサート・メッセージの場合だけである。その他のすべてのメッセージでは、レベルビットは 1 に設定しなければならない。
3. X ー 無関係。
4. APIC の動作は未定義である。

8.6.2. IPI のデスティネーションの指定

IPI のデスティネーションは、システムバス上のプロセッサのうち 1 つ、すべて、または一部（グループ）である。IPI の送信元は、以下の APIC レジスタおよびレジスタ内のフィールドを使用して、IPI のデスティネーションを指定する。

- ICR レジスタ ー ICR レジスタ内の以下のフィールドを使用して、IPI のデスティネーションを指定する。
  - ー デスティネーション・モード ー 2 つのデスティネーション・モード（物理または論理）のうち 1 つを選択する。
  - ー デスティネーション・フィールド ー 物理デスティネーション・モードでは、デスティネーション・プロセッサの APIC ID を指定する。論理デスティネーション・モードでは、メッセージ・デスティネーション・アドレス（MDA）を指定する。このアドレスを使用して、クラスタ内の特定のプロセッサを選択できる。
  - ー デスティネーション簡略表記 ー すべてのプロセッサ、当該プロセッサを除くすべてのプロセッサ、または当該プロセッサを、デスティネーションとして素早く指定できる。

— 伝達モード、最低優先度 — 最低優先度アービトレーション機構を使用して、指定したプロセッサ・グループの中からデスティネーション・レジスタを選択するように指定する。

- ローカル・デスティネーション・レジスタ（LDR）— 論理デスティネーション・モードおよびMDAと組み合わされて、デスティネーション・プロセッサの選択に使用される。
- デスティネーション・フォーマット・レジスタ（DFR）— 論理デスティネーション・モードおよびMDAと組み合わされて、デスティネーション・プロセッサの選択に使用される。

ICR、LDR、DFRを使用してIPIのデスティネーションを選択する方法は、使用するデスティネーション・モード（物理、論理、ブロードキャスト / 自己、または最低優先度伝達モード）によって異なる。これらのデスティネーション・モードについて、以下の各項で説明する。

8.6.2.1. 物理デスティネーション・モード

物理デスティネーション・モードでは、プロセッサのローカルAPIC IDによってデスティネーション・プロセッサを指定する（8.4.6.項「ローカルAPIC ID」を参照）。インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサでは、物理デスティネーション・モードで、単一のデスティネーション（ローカルAPIC IDは00H～FEH）またはすべてのAPICに対するブロードキャスト（APIC IDはFFH）を指定できる。このAPIC ID機構は、1つのシステムバス上で最大255個のローカルAPICを個別にアドレス指定できる。

P6ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、物理デスティネーション・モードで、0H～0EHのローカルAPIC IDを使用して単一のデスティネーションを指定できる。この方法により、APICバス上で最大15個のローカルAPICをアドレス指定できる。すべてのローカルAPICに対するブロードキャストは、0FHで指定できる。

注記

システムバス上で実際にアドレス指定できるローカルAPICの数は、ハードウェアによって制限される。

8.6.2.2. 論理デスティネーション・モード

論理デスティネーション・モードでは、ICR のデスティネーション・フィールドに入力される、8 ビットのメッセージ・デスティネーション・アドレス（MDA）を使用して IPI のデスティネーションを指定する。ローカル APIC は、論理デスティネーション・モードで送信された IPI メッセージを受信すると、メッセージ内の MDA と自分の LDR および DFR 内の値を比較して、IPI を受け入れて処理するかどうかを決める。

図 8-13. は、論理デスティネーション・レジスタ（LDR）のレイアウトを示している。このレジスタ内の 8 ビットの論理 APIC ID フィールドを使用して、MDA と比較される識別子が作成される。

注記

論理 APIC ID と、ローカル APIC ID レジスタ内のローカル APIC ID を混同してはならない。

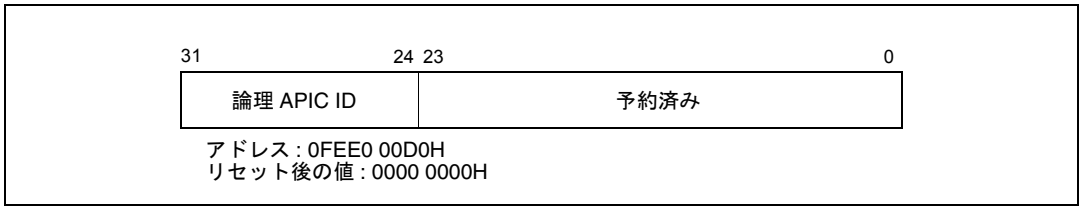


図 8-13. 論理デスティネーション・レジスタ（LDR）

図 8-14. は、デスティネーション・フォーマット・レジスタ（DFR）のレイアウトを示している。このレジスタ内の 4 ビットのモデル・フィールドは、論理デスティネーション・モードの使用時に、フラットモデルとクラスタモデルの 2 つのモデルのうちどちらを使用して MDA を解釈するかを選択する。

図 8-14. デスティネーション・フォーマット・レジスタ（DFR）

2 つのモデルによる MDA の解釈について、以下に説明する。

フラットモデル。このモデルを選択するには、DFR のビット 28 ～ 31 を 1111 にプログラムする。各ローカル APIC に対して LDR の論理 APIC ID フィールド内の異なるビットをセットすると、最大 8 つのローカル APIC に対して個別の論理 APIC ID を設定できる。次に、MDA 内の 1 ビット以上をセットすれば、ローカル APIC のグループを選択できる。

各ローカル APIC は、MDA と自分の論理 APIC ID の間で、ビット単位の AND 演算を実行する。真の条件が検出された場合は、ローカル APIC は IPI メッセージを受け入れる。すべての APIC に対するブロードキャストを実行するには、MDA をすべて 1 に設定する。

クラスタモデル。このモデルを選択するには、DFR のビット 28 ～ 31 を 0000 に設定する。このモデルは、フラットクラスタと階層クラスタと呼ばれる、2 つの基本的なデスティネーション指定方式をサポートしている。

フラット・クラスタ・デスティネーション・モデルは、P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでのみサポートされる。このモデルを使用する場合は、すべての APIC が APIC バスを介して接続されているものとする。MDA のビット 28 ～ 31 にはデスティネーション・クラスタのコード化されたアドレスが入り、ビット 24 ～ 27 はクラスタ内の最大 4 つのローカル APIC を指定する（フラット接続モデルと同様に、各ビットがクラスタ内の 1 つのローカル APIC に割り当てられる）。1 つ以上のローカル APIC を特定するには、MDA のビット 28 ～ 31 と LDR のビット 28 ～ 31 を比較して、ローカル APIC がそのクラスタに所属しているかどうかを判断する。MDA のビット 24 ～ 27 と LDR のビット 24 ～ 27 を比較して、クラスタ内のローカル APIC を特定する。

クラスタ内の一連のプロセッサを指定するには、MDA のビット 28 ～ 31 に送信先クラスタのアドレスを書き込み、MDA のビット 24 ～ 27 内で（選択されたクラスタメンバに対応する）選択されたビットをセットする。このモードでは、メッセージ内で、それぞれ 4 つのローカル APIC を含む 15 個のクラスタ（クラスタアドレス 0 ～ 14）を指定できる。しかし、P6 ファミリ・プロセッサおよびインテル Pentium プロセッサのローカル APIC の場合、APIC アービトレーション ID は、APIC エージェントを 15 個しかサポートしていない。したがって、このモードでサポートされるプロセッサおよびそのローカル APIC の総数は、15 までに制限される。すべてのローカル APIC に対するブロードキャストを実行するには、すべてのデスティネーション・ビットを 1 に設定する。これにより、すべてのクラスタが送信先としてマッチし、各クラスタ内のすべての APIC が選択される。

階層クラスタ・デスティネーション・モデルは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、またはインテル Pentium プロセッサで使用できる。このモデルでは、独立した複数のシステムバスまたは APIC

バスを介して異なるフラットクラスタを接続すれば、階層ネットワークを作成できる。この方式では、各クラスタ内に1つのクラスタ・マネージャが必要である。クラスタ・マネージャは、システムバスまたはAPICバス間のメッセージの受け渡しを処理する役割を受け持つ。1つのクラスタは最大4つのエージェントを使用できる。したがって、15のクラスタ・マネージャがそれぞれ4つのエージェントを使用すると、最大 60 個の APIC エージェントで構成されるネットワークを作成できる。階層的な APIC ネットワークには、ローカル APIC ユニットにも I/O APIC ユニットにも所属しない、特殊なクラスタ・マネージャ・デバイスが必要である。

8.6.2.3. ブロードキャスト / 自己伝達モード

ICR のデスティネーション簡略表記フィールドによって、伝達モードを無視し、システムバス上のすべてのプロセッサに対して IPI をブロードキャストしたり、発信元プロセッサに IPI を返送できる（8.6.1. 項「割り込みコマンドレジスタ（ICR）」を参照）。3 つのデスティネーション簡略表記（当該プロセッサ、当該プロセッサを除くすべてのプロセッサ、当該プロセッサを含むすべてのプロセッサ）がサポートされている。デスティネーション簡略表記を使用する場合は、デスティネーション・モードは無視される。

8.6.2.4. 最低優先度伝達モード

最低優先度伝達モードでは、ICR は、論理デスティネーションまたは簡略表記デスティネーション機構を使用して送信先プロセッサを選択し、システムバス上の複数のプロセッサに IPI を送信するようにプログラムされる。IPI が送信されると、システムバスまたは APIC バス上の選択されたプロセッサの間でアービトレーションが行われ、優先度が最も低いプロセッサが IPI を受け入れる。

インテル® Xeon™ プロセッサ・ベースのシステムでは、チップセットのバス・コントローラが、システム内の I/O APIC エージェントからのメッセージを受け入れ、システムバス上のプロセッサに割り込みを転送する。最低優先度伝達モードを使用する場合、チップセットは、一連の送信先候補の中から割り込みを受信するプロセッサを選択する。インテル® Pentium® 4 プロセッサは、システム内の各論理プロセッサの現在のタスク優先度をチップセットに通知する、特殊なバスサイクルをシステムバス上に発生させる。チップセットは、この情報を保存しておき、割り込みを受信したとき、この情報を使用して最低優先度のプロセッサを選択する。

P6 ファミリ・プロセッサ・ベースのシステムでは、最低優先度アービトレーションで使用されるプロセッサ優先度は、各ローカル APIC のアービトレーション・プライオリティ・レジスタ（APR）に格納されている。図 8-15. は、APR のレイアウトを示している。

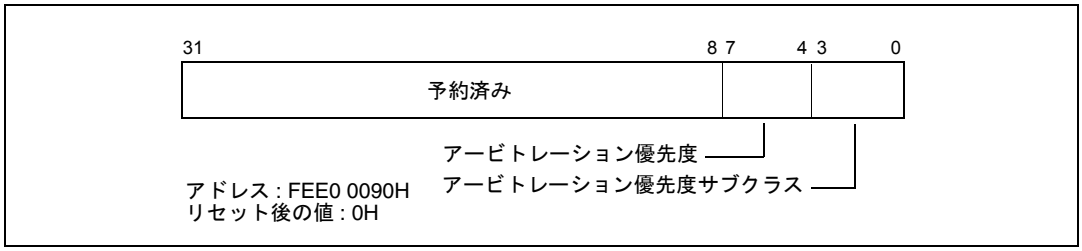


図 8-15. アービトレーション・プライオリティ・レジスタ（APR）

APR の値は、次のように計算される。

```
IF (TPR[7:4] ≥ IRRV[7:4]) AND (TPR[7:4] > ISRV[7:4])
    THEN
        APR[7:0] ← TPR[7:0]
    ELSE
        APR[7:4] ← max(TPR[7:4] AND ISRV[7:4], IRRV[7:4])
        APR[3:0] ← 0.
```

ここで、TPR の値は、TPR 内のタスク優先度の値である（図 8-18. を参照）、IRRV の値は、IRR 内でセットされているうちで最高の優先度を持つビットのベクタ番号（図 8-20. を参照）か、00H（IRR ビットがセットされていない場合）である。ISRV の値は、ISR 内でセットされているうちで最高の優先度を持つビットのベクタ番号（図 8-20. を参照）である。デスティネーション・プロセッサの間のアービトレーションの後、APR 内の値が最も小さいプロセッサが IPI を処理し、その他のプロセッサは IPI を無視する。

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、**フォーカス・プロセッサ**が存在する場合、優先度に関係なく、そのプロセッサが割り込みを受け入れる。あるプロセッサが現在割り込みを処理しているか、または割り込みに対する要求を保留している場合、そのプロセッサはその割り込みのフォーカスと呼ばれる。インテル Xeon プロセッサは、フォーカス・プロセッサの概念をサポートしていない。

最低優先度伝達モードを使用するオペレーティング・システムが TPR を更新しない場合、チップセットに保存された TPR 情報のために、割り込みが常に論理セット内の同じプロセッサに伝達される可能性がある。この動作は、P6 ファミリ・プロセッサに対して機能面で下方互換性があるが、パフォーマンスに予想不可能な影響を与えるときがある。

8.6.3. IPI の伝達と受け入れ

ICR の下位ダブルワードへの書き込みが行われると、ローカル APIC は、ICR に格納された情報に基づいて IPI メッセージを作成し、システムバス上（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）または APIC バス上（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサ）にそのメッセージを送信する。発行された IPI がどのように処理されるかについては、8.8. 節「割り込みの処理」で説明する。

8.7. システムバスと APIC バスのアービトレーション

複数のローカル APIC と I/O APIC がシステムバス（または APIC バス）上に IPI メッセージおよび割り込みメッセージを送信する場合、メッセージが送信されて処理される順番は、バス・アービトレーションによって決定される。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、ローカル APIC と I/O APIC は、システムバス用に定義されたアービトレーション機構を使用して、IPI を処理する順番を決定する。この機構はアーキテクチャ機能ではなく、ソフトウェアでは制御できない。

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、ローカル APIC と I/O APIC は、APIC に基づくアービトレーション機構を使用して、IPI を処理する順番を決定する。この場合、各ローカル APIC には、0 ～ 15 のアービトレーション優先度が与えられる。I/O APIC は、アービトレーションの際にこの優先度を使用して、どのローカル APIC が APIC バスにアクセスできるかを決定する。最高のアービトレーション優先度を持つローカル APIC が、常にバスへのアクセスに成功する。1 つのアービトレーション・ラウンドが完了するたびに、アクセスに成功したローカル APIC のアービトレーション優先度は 0 に下がり、アクセスできなかった各ローカル APIC のアービトレーション優先度は 1 ずつ上がる。

各ローカル APIC の現在のアービトレーション優先度は、ソフトウェアに透過的な 4 ビットのアービトレーション ID（Arb ID）レジスタに格納される。リセット時には、このレジスタは、（ローカル APIC ID レジスタに格納された）APIC ID 番号に初期化される。ICR コマンドによって発行される INIT レベル・デアサート IPI を使用して、各エージェントの Arb ID レジスタを現在の APIC ID の値にリセットすれば、ローカル APIC のアービトレーション優先度を再同期化できる。（インテル Pentium 4 プロセッサやインテル Xeon プロセッサは、Arb ID レジスタをサポートしていない。）

APIC バスのアービトレーション・プロトコルとバス・メッセージの形式については、8.10. 節「APIC バス・メッセージの受け渡し機構およびプロトコル（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサのみ）」で説明している。INIT レベル・

デアサート IPI メッセージについては、8.6.1. 項「割り込みコマンドレジスタ（ICR）」で説明している。

SIPI IPI（8.6.1. 項「割り込みコマンドレジスタ（ICR）」を参照）を除いて、指定されたデスティネーションに伝達されなかったすべてのバス・メッセージは、自動的に再試行される。したがって、ソフトウェアは、無効なローカル APIC または存在しないローカルAPICにIPIが送信されたためにメッセージが繰り返し再送信される状態が起こらないようにする必要がある。

8.8. 割り込みの処理

ローカルソースからの割り込み、I/O APIC からの割り込みメッセージ、または IPI を受信したとき、ローカル APIC がどのようにメッセージを処理するかは、以下の各項で説明するように、プロセッサによって異なる。

8.8.1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの割り込み処理

インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサでは、ローカルAPICは、受信したローカル割り込み、割り込みメッセージ、およびIPIを次のように処理する。

1. そのローカル APIC が指定されたデスティネーションかどうかを判断する（図 8-16. を参照）。指定されたデスティネーションであれば、メッセージを受け入れる。指定されたデスティネーションでなければ、メッセージを廃棄する。

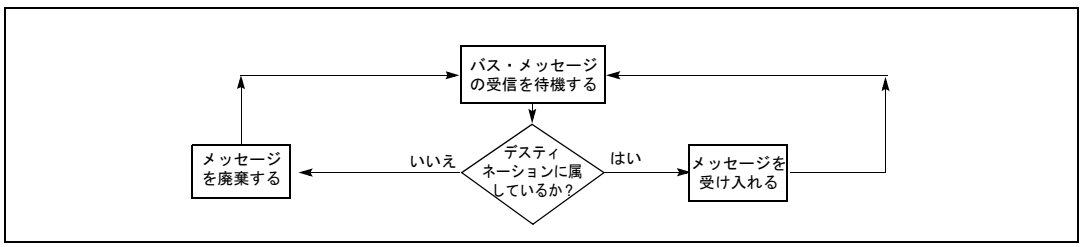


図 8-16. ローカル APIC の割り込み受け入れのフローチャート（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

2. ローカル APIC が割り込みのデスティネーションとして指定されており、割り込み要求が NMI、SMI、INIT、ExtINT、または SIPI である場合は、割り込みは処理のために直接プロセッサ・コアに送られる。

3. ローカル APIC が割り込みのデスティネーションとして指定されているが、割り込み要求が手順 2 に示した割り込みではない場合は、ローカル APIC は、IRR 内の適切なビットをセットする。
4. 割り込みが IRR および ISR レジスタ内で保留されているときは、ローカル APIC は、それらの割り込みの優先度、TPR 内の現在のタスク優先度、PPR 内の現在のプロセッサ優先度に基づいて、プロセッサ・コアに対して一度に 1 つずつ割り込みを発行する（8.8.3.1. 項「タスク優先度とプロセッサ優先度」を参照）。
5. 固定割り込みが処理のためにプロセッサ・コアに対して発行されると、ローカル APIC 内の割り込み終了（EOI）レジスタへの書き込みを行う命令ハンドラコード内の命令によって、ハンドラルーチンの完了が通知される（8.8.5. 項「割り込みサービス完了の通知」を参照）。EOI レジスタへの書き込みが行われると、ローカル APIC は ISR キューから割り込みを削除し、（レベルトリガ割り込みの場合は）割り込み処理が完了したことを示すメッセージをバス上に送信する。（NMI、SMI、INIT、ExtINT、または SIPI の場合は、ハンドラルーチン内に EOI レジスタへの書き込みが含まれていてはならない。）

8.8.2. P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサの割り込み処理

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、ローカル APIC は、受信したローカル割り込み、割り込みメッセージ、および IPI を次のように処理する。

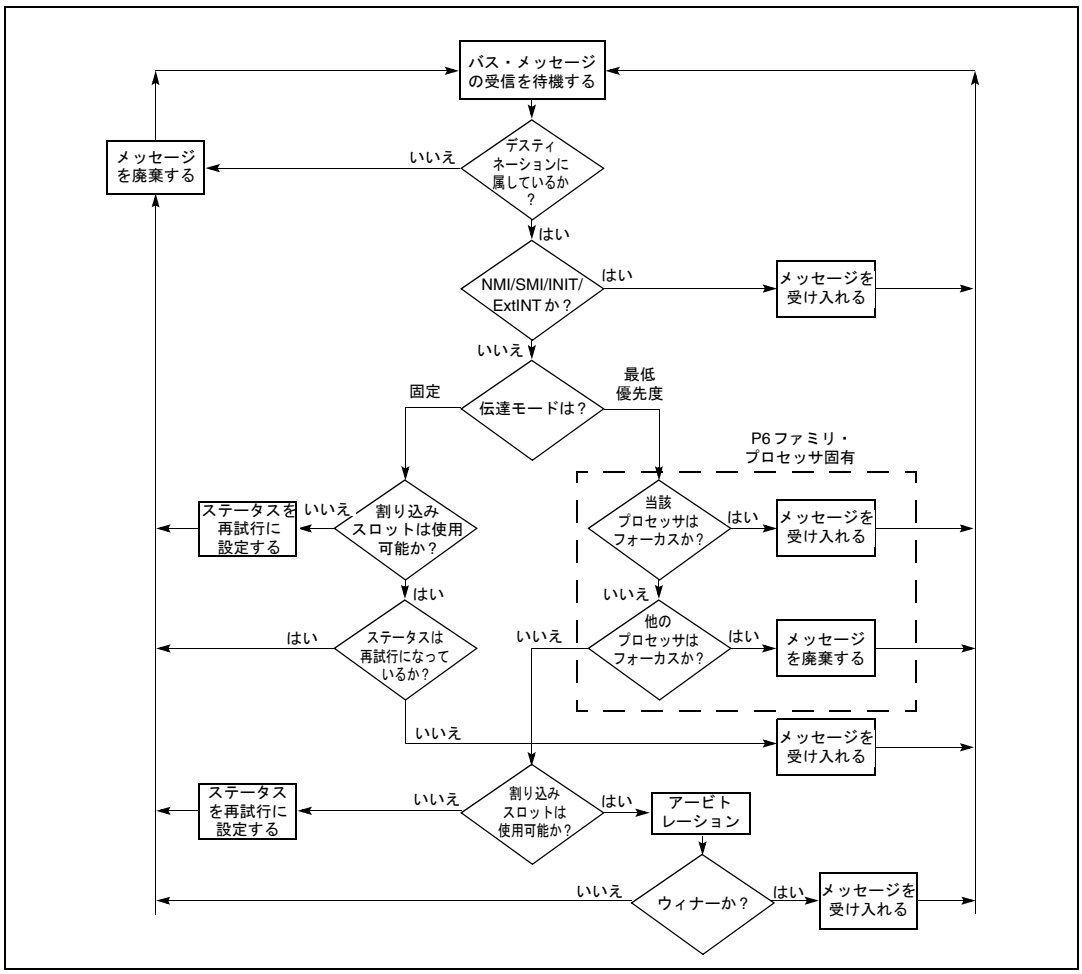


図 8-17. ローカル APIC の割り込み受け入れフローチャート（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサ）

1. （IPI のみ）ローカル APIC は、8.6.2. 項「IPI のデスティネーションの指定」で説明した方法で IPI メッセージを調べて、そのローカル APIC が指定されたデスティネーションかどうかを判断する。指定されたデスティネーションであれば、受け入れプロシージャを続行する。デスティネーションでなければ、IPI メッセージを廃棄する。メッセージが最低優先度伝達モードを指定している場合は、そのローカル APIC と、IPI メッセージの受信先として指定されている他のプロセッサの間でアービトレーションが行われる（8.6.2.4. 項「最低優先度伝達モード」を参照）。
2. ローカル APIC が割り込みのデスティネーションとして指定されており、割り込み要求が NMI、SMI、INIT、ExtINT、または INIT デアサート割り込み、あるいは MP プロトコル IPI メッセージ（BIPI、FIPI、SIPI）である場合は、割り込みは処理のために直接プロセッサ・コアに送られる。

3. ローカル APIC が割り込みのデスティネーションとして指定されているが、割り込み要求が手順 2 に示した割り込みではない場合は、ローカル APIC は、IRR レジスタと ISR レジスタの 2 つの未処理割り込みキューのうち 1 つで空きスロットを探す（図 8-20. を参照）。使用可能なスロットがあれば（8.8.4. 項「固定割り込みの受け入れ」を参照）、割り込みをそのスロットに入れる。使用可能なスロットがなければ、ローカル APIC は割り込み要求を拒否し、再試行メッセージと一緒に送信元に返送する。
4. 割り込みが IRR および ISR レジスタ内で保留されているときは、ローカル APIC は、それらの割り込みの優先度、TPR 内の現在のタスク優先度、PPR 内の現在のプロセッサ優先度に基づいて、プロセッサ・コアに対して一度に 1 つずつ割り込みを発行する（8.8.3.1. 項「タスク優先度とプロセッサ優先度」を参照）。
5. 固定割り込みが処理のためにプロセッサ・コアに対して発行されると、ローカル APIC 内の割り込み終了（EOI）レジスタへの書き込みを行う命令ハンドラコード内の命令によって、ハンドラルーチンの完了が通知される（8.8.5. 項「割り込みサービス完了の通知」を参照）。EOI レジスタへの書き込みが行われると、ローカル APIC はキューから割り込みを削除し、（レベルトリガ割り込みの場合は）割り込み処理が完了したことを示すメッセージをバス上に送信する。（NMI、SMI、INIT、ExtINT、または SIPI の場合は、ハンドラルーチンに EOI レジスタへの書き込みが含まれていてはならない。）

以下の各項では、ローカル APIC およびプロセッサによる割り込みの受け入れと処理について、さらに詳しく説明する。

8.8.3. 割り込み、タスク、プロセッサの優先度

割り込みがローカル APIC を介してプロセッサに伝達される場合、各割り込みには、ベクタ番号に基づいて暗黙的な優先度が与えられる。ローカル APIC は、この優先度を使用して、（他の割り込みの処理を含む）プロセッサの他の活動に対して、どの時点で割り込みを処理するかを決定する。

割り込みベクタが 16 ～ 255 の範囲内の場合、割り込みの優先度は、次の式で計算される。

優先度 = ベクタ /16

商は直近の整数値に合わせて切り捨てられ、優先度が求められる。1 が最低の優先度、15 が最高の優先度である。ベクタ 0 ～ 31 は IA-32 アーキテクチャ専用として予約済みであるため、ユーザ定義の割り込みの優先度は 2 ～ 15 の範囲になる。

それぞれの割り込み優先度レベル（ソフトウェアによって割り込み優先度クラスとして解釈される）には、16 のベクタが含まれる。特定の優先度レベル内の割り込みの優先度は、ベクタ番号によって決まる。ベクタ番号が大きいほど、優先度レベル内での優先度は高くなる。ベクタの優先度を計算し、優先度グループ内でのベクタの順位を

決めるとき、ベクタ番号は通常は2つの部分に分けられる。ベクタの上位4ビットはベクタの優先度を示し、下位4ビットは優先度グループ内での順位を示す。

8.8.3.1. タスク優先度とプロセッサ優先度

ローカルAPICは、割り込み処理の順番の決定に使用されるタスク優先度とプロセッサ優先度も定義している。タスク優先度は、タスク・プライオリティ・レジスタ（TPR）に書き込まれる、ソフトウェアによって選択された0～15の範囲の値である（図8-18.を参照）。TPRは、読み出し/書き込みレジスタである。

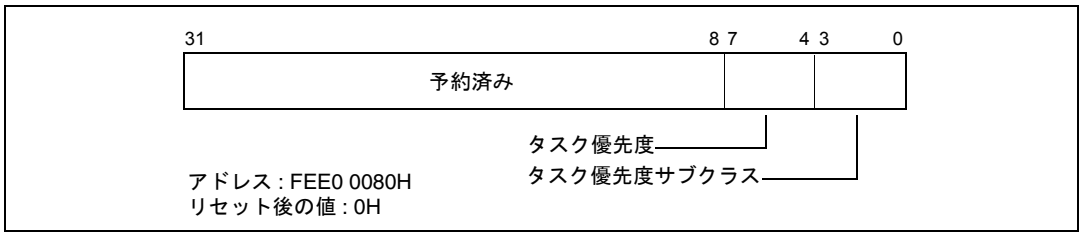


図 8-18. タスク・プライオリティ・レジスタ（TPR）

注記

この説明では、「タスク」の用語は、プロセッサ上で実行するためにオペレーティング・システムによって発行される、ソフトウェアで定義されたタスク、プロセス、スレッド、プログラム、またはルーチンを指す。これは、第6章「タスク管理」で説明した、IA-32アーキテクチャ上で定義されるタスクはない。

ソフトウェアは、タスク優先度を利用して、プロセッサに割り込みをかけられる**優先度のしきい値**を設定できる。プロセッサは、TPR内で指定された優先度より高い優先度を持つ割り込みだけを処理する。ソフトウェアがTPR内のタスク優先度を0に設定すると、プロセッサはすべての割り込みを処理する。ソフトウェアがTPR内のタスク優先度を15に設定すると、NMI、SMI、INIT、ExtINT、INITデアサート、スタートアップ伝達モードで伝達された割り込みを除く、すべての割り込みの処理が抑制される。この機構によって、オペレーティング・システムは、プロセッサが実行している優先度の高い作業を妨げないように、特定の割り込み（通常は優先度の低い割り込み）を一時的にブロックできる。

タスク優先度は、ローカル・プロセッサのアービトレーション優先度の決定にも使用される（8.6.2.4.項「最低優先度伝達モード」を参照）。

プロセッサ優先度は、プロセッサ・プライオリティ・レジスタ（PPR）に書き込まれる、プロセッサによって設定される0～15の範囲の値である（図8-19.を参照）。PPRは読み出し専用レジスタである。プロセッサ優先度は、プロセッサが現在どれだけの

優先度で実行されているかを表す。プロセッサ優先度は、未処理の割り込みをそのプロセッサに対して発行できるかどうかの判断に使用される。

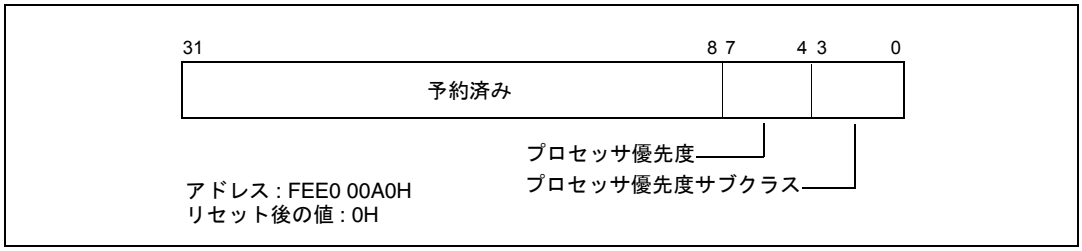


図 8-19. プロセッサ・プライオリティ・レジスタ（PPR）

PPR 内のプロセッサ優先度の値は、次のように計算される。

```
IF TPR[7:4] ≥ ISRV[7:4]
    THEN
        PPR[7:0] ← TPR[7:0]
    ELSE
        PPR[7:4] ← ISRV[7:4]
        PPR[3:0] ← 0
```

ISRV の値は、セットされているうちで最高の優先度を持つ ISR ビットのベクタ番号か、00H（ISR ビットがセットされていない場合）である。基本的に、プロセッサ優先度は、ISR 内の未処理の割り込みのうち最高の優先度を持つものと、現在のタスク優先度を比較して、優先度の高い方に合わせて設定される。

8.8.4. 固定割り込みの受け入れ

ローカル APIC は、受け入れた固定割り込みを、割り込み要求レジスタ（IRR）とインサービス・レジスタ（ISR）の 2 つの割り込み保留レジスタのうち 1 つのキューに入れる。図 8-20. は、これらの 2 つの 256 ビット読み出し専用レジスタを示している。これらのレジスタ内の 256 ビットは、サポートされる 256 個のベクタ（ベクタ 0 ～ 15 は APIC によって予約済み）を表す（8.5.2. 項も参照）。

注記

NMI、SMI、INIT、ExtINT、スタートアップ、または INIT デアサート伝達モードを使用するすべての割り込みは、IRR レジスタと ISR レジスタを無視して、サービスのために直接プロセッサ・コアに送られる。

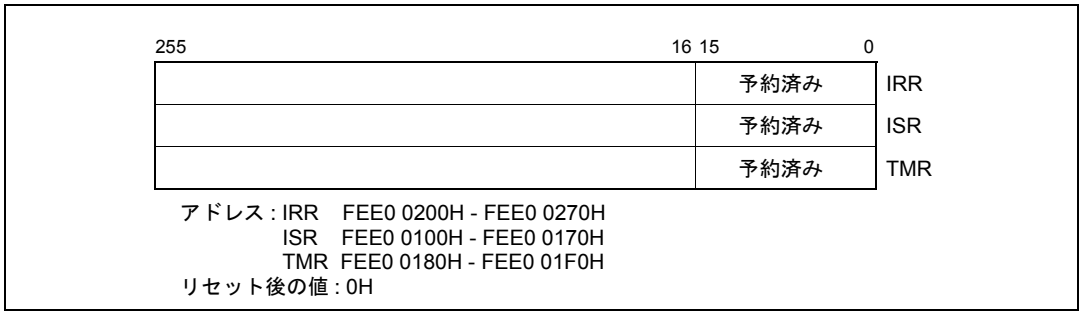


図 8-20. IRR、ISR、TMR レジスタ

IRR には、すでに受け入れられたが、まだサービスのためにプロセッサに対して発行されていない、アクティブな割り込み要求が格納される。ローカル APIC は、割り込みを受け入れると、受け入れられた割り込みのベクタに対応する IRR 内のビットをセットする。プロセッサ・コアが次の割り込みを処理する準備ができると、ローカル APIC は、セットされているうちで最高の優先度を持つ IRR ビットをクリアし、それに対応する ISR ビットをセットする。これで、ISR 内でセットされているうちで最高の優先度を持つビットのベクタが、サービスのためにプロセッサ・コアに対して発行される。

プロセッサが最高の優先度を持つ割り込みを処理している間、ローカル APIC は、IRR 内のビットをセットすることにより、追加の固定割り込みを送信できる。割り込みサービスルーチンが EOI レジスタへの書き込みを発行すると（8.8.5.項「割り込みサービス完了の通知」を参照）、ローカル APIC は、それに応答して、セットされているうちで最高の優先度を持つ ISR ビットをクリアする。次に、ローカル APIC は、IRR 内で最高の優先度を持つビットをクリアして ISR 内でそれに対応するビットをセットするプロセスを繰り返す。プロセッサ・コアは、ISR 内でセットされているうちで最高の優先度を持つビットに対するサービスルーチンの実行を開始する。

同じベクタ番号を持つ割り込みが 2 つ以上発生した場合は、ローカル APIC は、IRR と ISR の両方で、そのベクタに対応するビットをセットできる。つまり、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、IRR と ISR は、各割り込みベクタにつき 2 つの割り込みをキューに入れることができる（1 つは IRR、もう 1 つは ISR に入れられる）。同じ割り込みベクタで追加の割り込みが発行された場合は、それらはすべて IRR 内の 1 ビットに圧縮される。

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサでは、IRR および ISR レジスタがキューに入れられる割り込みは、各優先度レベル当たり最大 2 つまでに制限される。同じ優先度レベルで受信したその他の割り込みは、受け入れを拒否される。

プロセッサ・コア内で割り込みが有効になっているときに、現在処理中の割り込みより高い優先度を持つ割り込みをローカル APIC が受信した場合は、ローカル APIC は、

優先度が高い方の割り込みを(EOI レジスタへの書き込みを待たずに)直ちにプロセッサに対して発行する。最高の優先度を持つ割り込みを処理するために、現在実行中の割り込みハンドラは中断される。最高の優先度を持つ割り込みの処理が完了すると、中断された割り込みの処理が再開される。

トリガ・モード・レジスタ（TMR）は、割り込みのトリガモードを示す（図 8-20. を参照）。割り込みが IRR に受け入れられると、エッジトリガ割り込みの場合はそれに対応する TMR ビットがクリアされ、レベルトリガ割り込みの場合は TMR ビットがセットされる。割り込みベクタに対する EOI サイクルが発生したとき、それに対応する TMR ビットがセットされた場合は、すべての I/O APIC に EOI メッセージが送信される。

8.8.5. 割り込みサービス完了の通知

NMI、SMI、INIT、ExtINT、スタートアップ、または INIT デアサート伝達モードで伝達される割り込み以外のすべての割り込みでは、割り込みハンドラに、割り込み終了（EOI）レジスタ（図 8-21. を参照）への書き込みが含まれていなければならない。この書き込みは、ハンドラルーチンの終了時に、IRET 命令の少し前に実行されなければならない。この動作は、現在の割り込みのサービスが完了し、ローカル APIC が ISR から次の割り込みを発行できることを通知する。

図 8-21. EOI レジスタ

EOI を受信すると、ローカル APIC は、ISR 内で最高の優先度を持つビットをクリアし、1 つ下の優先度を持つ割り込みをプロセッサに対して発行する。終了した割り込みがレベルトリガ割り込みである場合は、ローカル APIC は、すべての I/O APIC に割り込み終了（EOI）メッセージを送信する。

将来の互換性を保証するために、ソフトウェアは EOI レジスタに 0H の値を書き込むと、割り込み終了（EOI）コマンドを発行する必要がある。

8.9. スプリアス割り込み

プロセッサが、プロセッサのタスク優先度を、プロセッサの INTR 信号が現在アサートされている割り込みのレベルと同じかそれ以上に上げると、特殊な状況が発生するときがある。INTA サイクルが発行された時点で、プロセッサに対して発行されるはずの割り込みが（ソフトウェアによってプログラムされて）マスクされていた場合、ローカル APIC は、スプリアス割り込みベクタを伝達する。プロセッサに対してスプリアス割り込みベクタを発行しても、ISR には影響を与えない。したがって、このベクタのハンドラは、EOI なしでリターンする必要がある。

スプリアス割り込みベクタのベクタ番号は、スプリアス割り込みベクタレジスタで指定される（図 8-22. を参照）。このレジスタの各フィールドの機能は次のとおりである。

スプリアス・ベクタ

ローカル APIC がスプリアス・ベクタを生成したときにプロセッサに伝達されるベクタ番号を指定する。

（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）このフィールドのビット 0 ～ 7 は、ソフトウェアによってプログラム可能である。

（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサ）このフィールドのビット 4 ～ 7 は、ソフトウェアによってプログラム可能である。ビット 0 ～ 3 は、論理的な 1 にハードワイヤードされる。ソフトウェアがビット 0 ～ 3 に書き込んでも、何も影響はない。

APIC のソフトウェアによる有効 / 無効

ソフトウェアがローカル APIC を一時的に有効（1）または無効（0）にする（8.4.3.「ローカル APIC の有効化または無効化」を参照）。

フォーカス・プロセッサ・チェック

最低優先度伝達モードを使用する場合のフォーカス・プロセッサ・チェックを有効（0）または無効（1）にする。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、このビットは予約済みであり、0 にクリアされていなければならない。

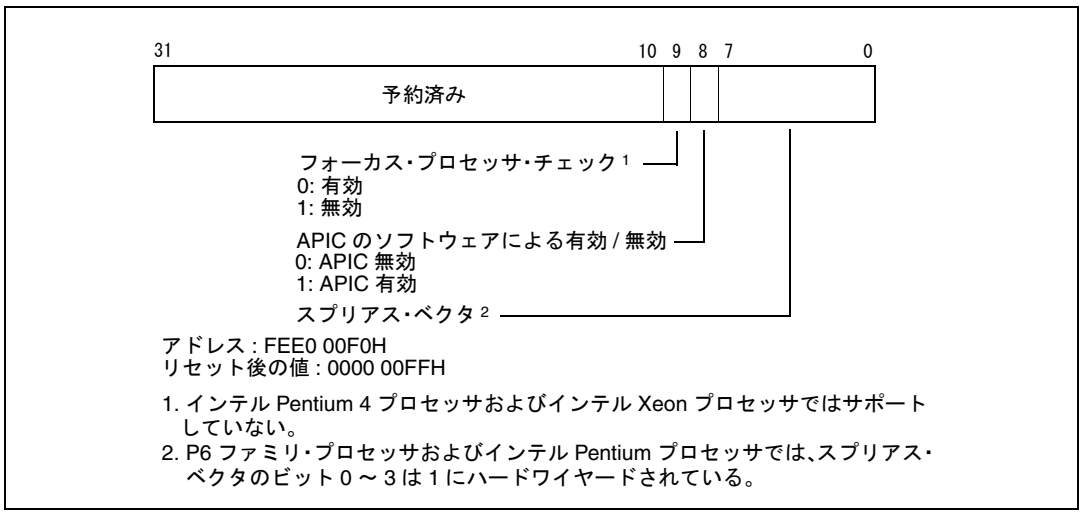


図 8-22. スプリアス割り込みベクタレジスタ（SVR）

8.10. APIC バス・メッセージの受け渡し機構およびプロトコル（P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサのみ）

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサは、システム・バス・メッセージ受け渡し機構およびプロトコルを使用して、システムバス上のローカル APIC と I/O APIC の間でメッセージを受け渡す。

P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサは、シリアル APIC バス上のローカル APIC と I/O APIC の間で、次のような方法でメッセージを受け渡す。APIC バス上には一度に 1 つのメッセージしか送信できないため、I/O APIC とローカル APIC は、「循環優先度」アービトレーション・プロトコルを使用して、APIC バス上にメッセージを送信する許可を得る。1 つ以上の APIC が、メッセージの送信を同時に開始できる。各メッセージの始めに、各 APIC は、送信中のメッセージのタイプと APIC バス上での現在のアービトレーション優先度を示す。この情報がアービトレーションに使用される。（1 つのアービトレーション・ラウンドの中で）アービトレーション・サイクルが 1 つ完了するたびに、ウィナーの候補だけが残ってバスをドライブし続ける。すべてのアービトレーション・サイクルが完了するまでに、バスをドライブしている APIC は 1 つだけになる。ウィナーが選択されると、その APIC はバスの独占的な使用を許され、バスをドライブし続けて実際のメッセージを送信する。

メッセージが正常に送信されるたびに、すべての APIC のアービトレーション優先度が 1 ずつ上がる。前回のウィナー（つまり、最後にメッセージを正常に送信した APIC）の優先度は 0（最低）になる。アービトレーション中にアービトレーション優先度が

15（最高）だったにもかかわらずメッセージを送信しなかったエージェントは、前回のウィナーのアービトレーション優先度を受け継ぎ、それを1だけ上げる。

ただし、APICのうち1つが特殊な割り込み終了（EOI）メッセージを発行した場合は、上記のアービトレーション・プロトコルは多少異なるものになる。この優先度の高いメッセージは、送信元のアービトレーション優先度に関係なく、バスの使用許可を与えられる。ただし、2つ以上のAPICが同時にEOIメッセージを発行した場合は、EOIメッセージを送信しているAPICのアービトレーション優先度に基づいて、それらのAPICのアービトレーションが行われる。

APICが「最低優先度」アービトレーションを使用するように設定されている場合（8.6.2.4. 項「最低優先度伝達モード」を参照）、最低の優先度（APRレジスタ内の値）で実行中のAPICが複数あるときは、その中からアービトレーション優先度（Arb IDレジスタ内の固有の値）に基づいてAPICが選択される。APRの8ビットすべてが、最低優先度アービトレーションに使用される。

8.10.1. バス・メッセージの形式

シリアルAPICバス上でのメッセージの送信に使用されるバス・メッセージの形式については、付録F「APICバス・メッセージの形式」を参照のこと。

8.11. メッセージ・シグナル割り込み

『PCI Local Bus Specification, Rev 2.2』（www.pcisig.com）では、メッセージ・シグナル割り込みの概念が導入された。この機能を持つインテル・プロセッサおよびチップセットには、現在のところインテル® Pentium® 4プロセッサとインテル® Xeon™プロセッサが含まれる。この仕様は、次のように規定している。

「メッセージ・シグナル割り込み（MSI）は、PCIデバイスが、システムが指定するメッセージをシステムが指定するアドレスに書き込むこと（PCI DWORDメモリ書き込みトランザクション）によってサービスを要求できる、オプションの機能である。トランザクション・アドレスはメッセージの送信先を指定し、トランザクション・データはメッセージを指定する。システム・ソフトウェアは、デバイスの構成時にメッセージの送信先とメッセージを初期化し、それぞれのMSI対応機能に1つ以上の非共有メッセージを割り当てる役割を受け持つ。」

PCIローカルバス仕様に規定されたMSI機構を使用して、MSI対応PCIデバイスを特定し、設定できる。なかでも、MSI構造には、メッセージ・データ・レジスタとメッセージ・アドレス・レジスタが含まれる。サービスを要求するには、PCIデバイスの機能が、メッセージ・データ・レジスタの内容を、メッセージ・アドレス・レジスタ（64ビット・メッセージ・アドレスの場合はメッセージ上位アドレスレジスタ）に格納されたアドレスに書き込む。

8.11.1. 項と 8.11.2. 項では、メッセージ・アドレス・レジスタとメッセージ・データ・レジスタのレイアウトを詳しく説明する。PCI デバイスによって発行される操作は、メッセージ・データ・レジスタの内容を使用した、メッセージ・アドレス・レジスタに対する PCI 書き込みコマンドである。この操作は、PCI 書き込み操作について定義されたセマンティクスの規則に従う。この操作はDWORD操作である。

8.11.1. メッセージ・アドレス・レジスタの形式

図 8-23. に、メッセージ・アドレス・レジスタ（下位 32 ビット）の形式を示す。

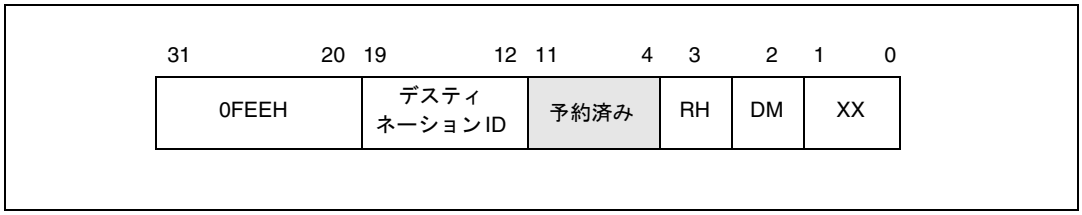


図 8-23. MSI メッセージ・アドレス・レジスタのレイアウト

メッセージ・アドレス・レジスタ内の各フィールドは、次のとおりである。

1. ビット 31 ～ 20: これらのビットは、割り込みメッセージを表す固定値（0FEEH）を格納する。この値は、4G ～ 18M のベースアドレスを持つ 1MB 領域での割り込みの位置を示す。この領域へのすべてのアクセスは、割り込みメッセージとして転送される。他のデバイスがこの領域を I/O 空間として要求しないように保証する必要がある。
2. デスティネーション ID: このフィールドは、8 ビットのデスティネーション ID を格納する。デスティネーション ID は、メッセージの送信先のプロセッサを指定する。IOAPIC を使用して送信先プロセッサに対して割り込みを発行する場合、デスティネーション ID は、I/O APIC リダイレクション・テーブル・エントリのビット 63:56 に対応する。
3. リダイレクション・ヒント指標（RH）: このビットは、割り込みを受信できるプロセッサの中で最低の割り込み優先度のプロセッサにメッセージを転送するかどうかを指定する。
   - RH が 0 の場合、割り込みは、デスティネーション ID フィールドにリストされたプロセッサに転送される。
   - RH が 1 で物理デスティネーション・モードが使用されている場合には、デスティネーション ID フィールドは 0xFF にセットされていてはいけない。デスティネーション ID フィールドは存在し割り込みを受け付けるプロセッサを指していなければならない。
   - RH が 1 でフラットなアドレッシング・モデルを採用したシステムにおいて論理デスティネーションモードがアクティブの場合には、デスティネーション ID フィー

ルドは、存在し割り込みを受け付けるプロセッサを示すように 1 にセットされなければならない。

- RH が 1 でクラスタ・アドレッシング・モデルを採用したシステムにおいて論理デスティネーション・モードがアクティブの場合、デスティネーション ID フィールドは 0xFF にセットされなければならない。このフィールドで識別されたプロセッサは存在し割り込みを受け付けなければならない。

4. デスティネーション・モード（DM）：このビットは、最低優先度のプロセッサに割り込みを伝達する場合、デスティネーション ID フィールドを論理 APIC ID として解釈するか、物理 APIC ID として解釈するかを指定する。RH が 1 で DM が 0 の場合、デスティネーション ID フィールドは物理デスティネーション・モードになり、一致する APIC ID を持つシステム内のプロセッサだけが割り込みの伝達先と見なされる（つまり、リダイレクションは行われない）。RH が 1 で DM が 1 の場合、デスティネーション ID フィールドは論理デスティネーション・モードとして解釈される。この場合、リダイレクションは、プロセッサの論理 APIC ID とメッセージ内のデスティネーション ID フィールドに基づいたプロセッサの論理グループに含まれるプロセッサだけに制限される。この論理グループを構成するプロセッサは、8 ビットのデスティネーション ID と、各ローカル APIC 内のデスティネーション・フォーマット・レジスタおよび論理デスティネーション・レジスタによって指定される論理デスティネーションを照合することによって特定される。詳細は、8.6.2. 項「IPI のデスティネーションの指定」の説明と同様である。RH が 0 の場合は、DM ビットは無視され、メッセージは、物理デスティネーション・モードと論理デスティネーション・モードのどちらが使用されているかとは関係なく送信される。

8.11.2. メッセージ・データ・レジスタの形式

図 8-24. に、メッセージ・データ・レジスタのレイアウトを示す。

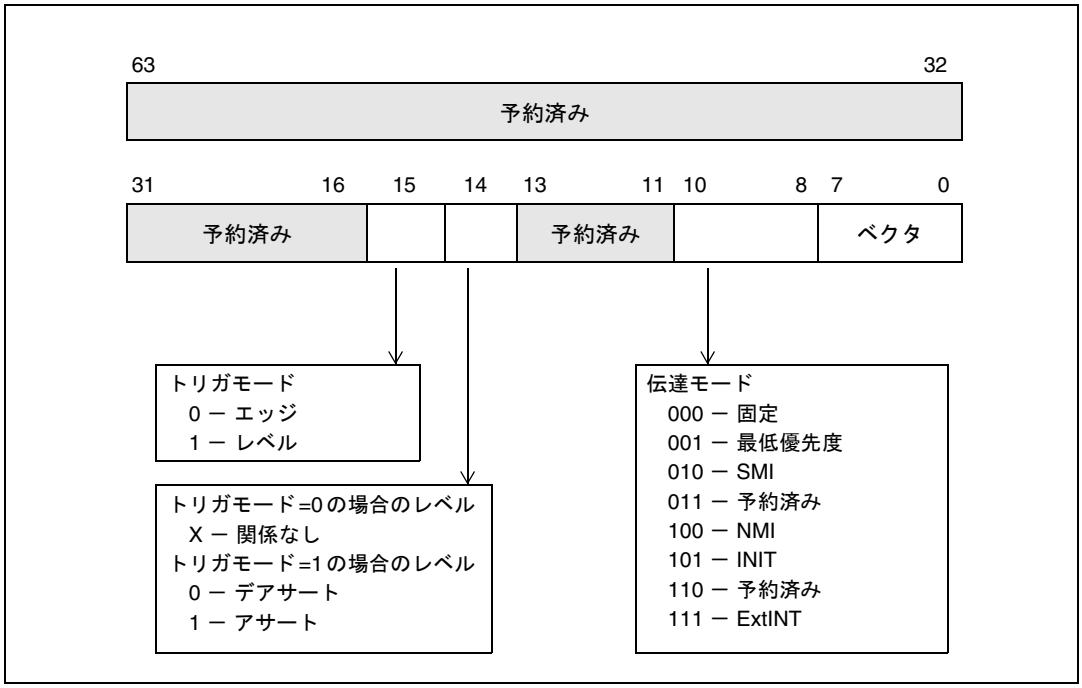


図 8-24. MSI メッセージ・データ・レジスタのレイアウト

予約済みのフィールドの値を前提にしてはならない。これらのフィールドに書き込みがあった場合、ソフトウェアはフィールドの内容を維持しなければならない。メッセージ・データ・レジスタ内のその他のフィールドについて以下に説明する。

1. ベクタ : この 8 ビット・フィールドは、メッセージに関連付けられる割り込みベクタを格納する。値の範囲は 010H ～ 0FEH である。ソフトウェアは、このフィールドにベクタ 00H ～ 0FH が書き込まれないように保証しなければならない。
2. 伝達モード : この 3 ビット・フィールドは、割り込みの受け入れ方法を指定する。伝達モードは、指定されたトリガモードと合わせて機能する。正しいトリガモードは、ソフトウェアが保証しなければならない。制限について以下に説明する。
   a. 000B（固定モード）ー デスティネーション・フィールドにリストされたすべてのエージェントに信号を伝達する。固定伝達モードで有効なトリガモードは、エッジまたはレベルである。
   b. 001B（最低優先度）ー デスティネーション・フィールドにリストされたすべてのエージェントのうち、最低優先度で動作しているエージェントに信号を伝達する。トリガモードは、エッジまたはレベルである。

c. 010B（システム管理割り込み、SMI）－ 伝達モードはエッジのみである。SMI セマンティクスに依存するシステムでは、ベクタ・フィールドは無視されるが、将来の互換性を保証するためにすべてゼロを書き込まなければならない。

d. 100B（NMI）－ デスティネーション・フィールドにリストされたすべてのエージェントに信号を伝達する。ベクタ情報は無視される。NMI は、トリガモードの設定に関係なく、エッジトリガ割り込みになる。

e. 101B（INIT）－ デスティネーション・フィールドにリストされたすべてのエージェントにこの信号を伝達する。ベクタ情報は無視される。INIT は、トリガモードの設定に関係なく、エッジトリガ割り込みになる。

f. 111B（ExtINT）－ デスティネーション・フィールド内のすべてのエージェントの INTR 信号に（8259A 割り込みコントローラまたはその互換デバイスから発信された割り込みとして）信号を伝達する。ベクタは、ExtINT の活性化によって発行される INTA サイクルによって指定される。ExtINT はエッジトリガ割り込みである。

3. レベル : エッジトリガ割り込みメッセージは、常にアサート・メッセージとして解釈される。エッジトリガ割り込みの場合、このフィールドは使用されない。レベルトリガ割り込みの場合、このビットは割り込み入力の状態を反映する。

4. トリガモード : このフィールドは、メッセージのトリガとなる信号タイプを示す。

a. 0 － エッジ・センシティブを示す。

b. 1 － レベル・センシティブを示す。

9

プロセッサの管理と初期化

第 9 章
プロセッサの管理と初期化

9

本章では、プロセッサ全体にわたるさまざまな機能を管理したり、プロセッサを初期化するための便利な方法について説明する。本章で扱っているトピックは以下のとおりである。1）プロセッサの初期化、2）x87 FPU の初期化、3）プロセッサのコンフィギュレーション、4）他の機能の決定、5）モードの切り替え、6）インテル® Pentium® プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ内のモデル固有レジスタ（MSR: Model-Specific Register）、7）P6 ファミリ・プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ内のメモリタイプ範囲レジスタ（MTRR: Memory Type Range Register）

9.1. 初期化の概要

電源投入の後、または RESET# ピンのアサーションの後、システムバス上の各プロセッサは、それぞれのハードウェアを初期化し（これをハードウェア・リセットと呼ぶ）、オプションのビルトイン・セルフ・テスト（BIST: Built-In Self Test）を実行する。ハードウェア・リセットによって、各プロセッサのレジスタは定められた状態に設定され、プロセッサは実アドレスモードに設定される。またこのリセットにより、内部キャッシュ、トランスレーション・ルックアサイド・バッファ（TLB: Translation Lookaside Buffer）、ブランチ・ターゲット・バッファ（BTB: Branch Target Buffer）も無効になる。この時点では、動作内容は以下に示すようにプロセッサ・ファミリによって決まる。

- インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ ー（ユニ・プロセッサ・システム内の単一プロセッサも含め）システムバス上にあるプロセッサのすべてが、マルチプロセッサ（MP）の初期化プロトコルを実行する。次に、このプロトコルによってブートストラップ・プロセッサ（BSP）として選択されているプロセッサは、EIP レジスタのオフセットから始まる現行のコード・セグメント内で、ソフトウェア初期化コードの実行をすぐに開始する。（ブートストラップ・プロセッサ以外の）AP プロセッサは、ブートストラップ・プロセッサ（BSP）が初期化コードを実行している間スタートアップ IPI（SIPI）待機状態になる。詳細については、7.5. 節「マルチプロセッサ（MP）初期化」を参照。ユニ・プロセッサのシステムでは、単一のインテル Pentium 4 プロセッサまたはインテル Xeon プロセッサが自動的に BSP となることに注意しなければならない。
- P6 ファミリ・プロセッサ ー 実行される処理は、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの場合（前のパラグラフで説明）と同じである。

- インテル® Pentium® プロセッサ ー 単一（single）プロセッサまたは二重（dual）プロセッサのどちらかのシステムでは、単一のインテル Pentium プロセッサが、常に第一次プロセッサとしてあらかじめ指定されている。単一プロセッサと二重プロセッサの両方のシステム内で、リセットの後に第一次プロセッサが動作する内容は以下のようになる。つまり第一次プロセッサは、二重プロセッサ（DP）初期化準備プロトコルを使用して、すぐに EIP レジスタ内のオフセットから始まる現行のコード・セグメント内にあるソフトウェア初期化コードの実行を開始する。（第二次プロセッサがある場合は）その第二次プロセッサは停止状態になる。
- Intel486™ プロセッサ ー 直ちに第一次プロセッサ（またはユニ・プロセッサ・システム内の単一プロセッサ）が、EIP レジスタ内のオフセットから始まる現行のコード・セグメント内にあるソフトウェア初期化コードの実行を開始する。（Intel486 は、どのプロセッサが第一次プロセッサであるかを確定するために、DP 初期化プロトコルまたは MP 初期化プロトコルを自動的には実行しない。）

ソフトウェア初期化コードは、ブートストラップ・プロセッサ（BSP）または第一次プロセッサの全システム固有の初期化と、システム論理を実行する。

この時点で、マルチプロセッサ（または二重プロセッサ）のシステムでは、ブートストラップ・プロセッサ（または第一次プロセッサ）が、それぞれの AP（または第二次）プロセッサを起動し、各プロセッサがセルフ・コンフィギュレーション・コードを実行できるようにする。

プロセッサの初期化、コンフィギュレーション、同期化がすべて終わると、BSP または第一次プロセッサが、オペレーティング・システムまたはエグゼクティブの最初のタスクの実行を開始する。

ハードウェア・リセットの実行時には、x87FPU も、定められている状態に初期化される。次に、x87 FPU ソフトウェア初期化コードを実行して x87 FPU 精度の設定や例外マスクの設定などを行うことができる。動作モードの切り替えには、特に x87 FPU を初期化する必要はない。

プロセッサ上の INIT# ピンをアサートすると、ハードウェア・リセットに類似した応答が呼び出される。大きな違いは、INIT 時に内部キャッシュ、MSR、MTRR、x87 FPU の状態が変更されないことである。（ハードウェア・リセットの場合と同じように、TLB と BTB は無効になる）。INIT を使用すると、内部キャッシュの内容を保持しながら、保護モードから実アドレスモードに切替えることができる。

9.1.1. リセット後のプロセッサの状態

表 9-1. では、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサに電源を投入した後のフラグと他のレジスタの状態を示している。制御レジスタ CR0 の状態は 60000010H になる（図 9-1. を参照）。この状態では、プロセッサは実アドレスモードに設定され、ページングがディスエーブルになる。

9.1.2. プロセッサ・ビルトイン・セルフ・テスト（BIST: Processor Built-in Self-test）

ハードウェアは、電源投入時に BIST の実行を要求できる。EAX レジスタは、プロセッサが BIST をパスした場合にクリアされる（0H）。BIST 後に、EAX レジスタ内にゼロ以外の値がストアされている場合は、プロセッサの不良が検出されたことを表している。BIST が要求されなかったときは、ハードウェア・リセットの後 EAX レジスタの内容は 0H になる。

BIST を実行するオーバーヘッドは、プロセッサ・ファミリ間で異なる。例えば BIST は、約 3000 万のプロセッサ・クロック周期でインテル® Pentium® 4 プロセッサを実行する。（このクロックカウントはモデル固有のものである。またインテルは、IA-32 プロセッサに使用する正確な周期数を通知することなしに変更する権利を保有する）。

表 9-1. 電源投入、リセットまたは INIT 後の 32 ビット IA-32 プロセッサの状態

レジスタ	インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ	P6 ファミリ・プロセッサ	インテル Pentium プロセッサ
EFLAGS[1]	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H[2]	60000010H[2]	60000010H[2]
CR2, CR3, CR4	00000000H	00000000H	00000000H
CS	セレクタ = F000H ベース = FFFF0000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み	セレクタ = F000H ベース = FFFF0000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み	セレクタ = F000H ベース = FFFF0000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み
SS、DS、ES、FS、GS	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み、アクセス済み
EDX	00000FxxH	000006xxH	000005xxH
EAX	0[3]	0[3]	0[3]
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H

表 9-1. 電源投入、リセットまたは INIT 後の 32 ビット IA-32 プロセッサの状態（続き）

レジスタ	インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ	P6 ファミリ・プロセッサ	インテル Pentium プロセッサ
ST0 ～ ST7[4]	電源投入またはリセット: +0.0 FINIT/FNINIT: 変更なし	電源投入またはリセット: +0.0 FINIT/FNINIT: 変更なし	電源投入またはリセット: +0.0 FINIT/FNINIT: 変更なし
x87 FPU 制御ワード[4]	電源投入またはリセット: 0040H FINIT/FNINIT: 037FH	電源投入またはリセット: 0040H FINIT/FNINIT: 037FH	電源投入またはリセット: 0040H FINIT/FNINIT: 037FH
x87 FPU 状態ワード[4]	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H
x87 FPU タグワード[4]	電源投入またはリセット: 5555H FINIT/FNINIT: FFFFH	電源投入またはリセット: 5555H FINIT/FNINIT: FFFFH	電源投入またはリセット: 5555H FINIT/FNINIT: FFFFH
x87 FPU データ・オペランドと CS セグメント・セレクタ[4]	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H	電源投入またはリセット: 0000H FINIT/FNINIT: 0000H
x87 FPU データ・オペランドと命令ポインタ[4]	電源投入またはリセット: 00000000H FINIT/FNINIT: 00000000H	電源投入またはリセット: 00000000H FINIT/FNINIT: 00000000H	電源投入またはリセット: 00000000H FINIT/FNINIT: 00000000H
MM0 ～ MM7[4]	電源投入またはリセット: 0000000000000000H INIT または FINIT/FNINIT: 変更なし	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサのみー 電源投入またはリセット: 0000000000000000H INIT または FINIT/FNINIT: 変更なし	MMX テクノロジ搭載のインテル Pentium プロセッサのみー 電源投入またはリセット: 0000000000000000H INITまたはFINIT/FNINIT: 変更なし
MM0 ～ MM7	電源投入またはリセット: 0000000000000000H INIT: 変更なし	インテル Pentium III プロセッサのみー 電源投入またはリセット : 0000000000000000H INIT: 変更なし	NA
MXCSR	電源投入またはリセット: 1F80H INIT: 変更なし	インテル Pentium III プロセッサのみー 電源投入またはリセット: 1F80H INIT: 変更なし	NA
GDTR、IDTR	ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み	ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み	ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み
LDTR、タスクレジスタ	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み	セレクタ = 0000H ベース = 00000000H リミット = FFFFH AR = 存在、読み取り / 書き込み
DR0、DR1、DR2、DR3	00000000H	00000000H	00000000H

表 9-1. 電源投入、リセットまたは INIT 後の 32 ビット IA-32 プロセッサの状態（続き）

レジスタ	インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ	P6 ファミリ・プロセッサ	インテル Pentium プロセッサ
DR6	FFFF0FF0H	FFFF0FF0H	FFFF1FF0H
DR7	00000400H	00000400H	00000000H
タイムスタンプ・カウンタ	電源投入またはリセット: 0H INIT: 変更なし	電源投入またはリセット: 0H INIT: 変更なし	電源投入またはリセット: 0H INIT: 変更なし
性能カウンタとイベント選択	電源投入またはリセット: 0H INIT: 変更なし	電源投入またはリセット: 0H INIT: 変更なし	電源投入またはリセット: 0H INIT: 変更なし
他の MSR すべて	電源投入またはリセット: 未定義 INIT: 変更なし	電源投入またはリセット: 未定義 INIT: 変更なし	電源投入またはリセット: 未定義 INIT: 変更なし
TLB のデータとコード・キャッシュ	無効	無効	無効
固定 MTRR	電源投入またはリセット: ディスエーブル INIT: 変更なし	電源投入またはリセット: ディスエーブル INIT: 変更なし	未実行
可変 MTRR	電源投入またはリセット: ディスエーブル INIT: 変更なし	電源投入またはリセット: ディスエーブル INIT: 変更なし	未実行
マシン・チェック・アーキテクチャ	電源投入またはリセット: 未定義 INIT: 変更なし	電源投入またはリセット: 未定義 INIT: 変更なし	未実行
APIC	電源投入またはリセット: イネーブル INIT: 変更なし	電源投入またはリセット: イネーブル INIT: 変更なし	電源投入またはリセット: イネーブル INIT: 変更なし

注：

1. EFLAGS レジスタの最上位 10 ビットは、リセット後も未定義のままである。ソフトウェアは、これらのいずれのビットの状態にも影響されない。
2. CD フラグと NW フラグは変更されない。またビット 4 は 1 にセットされ、他のビットはすべてクリアされる。
3. 電源投入またはリセット時にビルトイン・セルフ・テスト（BIST）が呼び出された場合、すべてのテストが合格したときにだけ EAX が 0 になる。（INIT 中に BIST を呼び出すことはできない。）
4. x87 FPU と MMX テクノロジ・レジスタの状態は、INIT の実行によって変更されない。

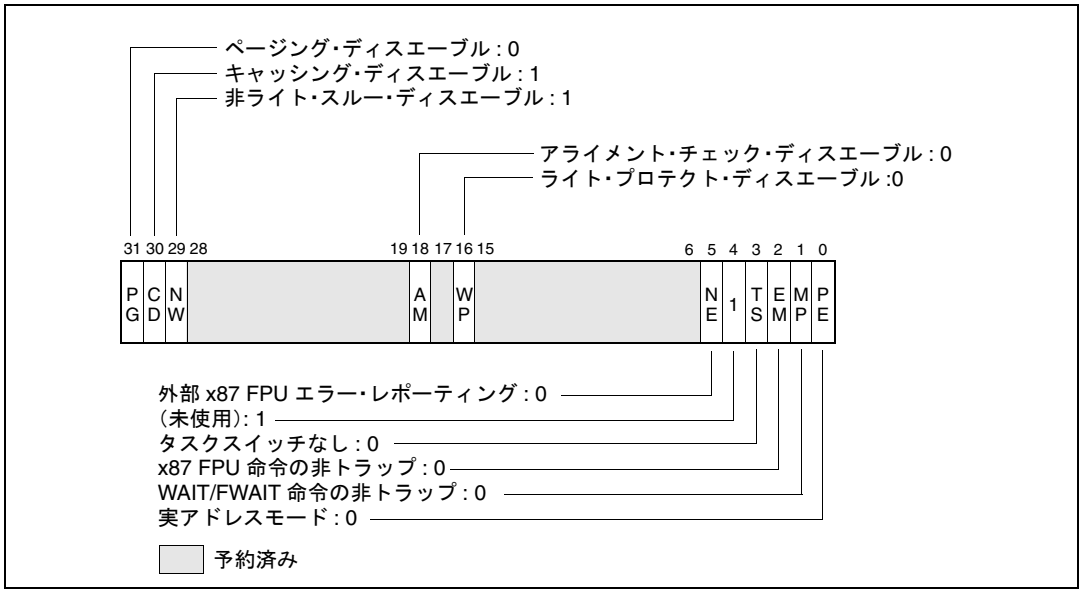


図 9-1. リセット後の CR0 レジスタの内容

9.1.3. モデルとステッピングに関する情報

ハードウェア・リセットの後、EDX レジスタ内に、コンポーネント識別と改訂情報がストアされる（図 9-2. を参照）。例えば、インテル® Pentium® 4 プロセッサ・ファミリの最初のプロセッサについては、モデル（0000B）、ファミリ（1111B）、プロセッサ・タイプ（00B）が返される。

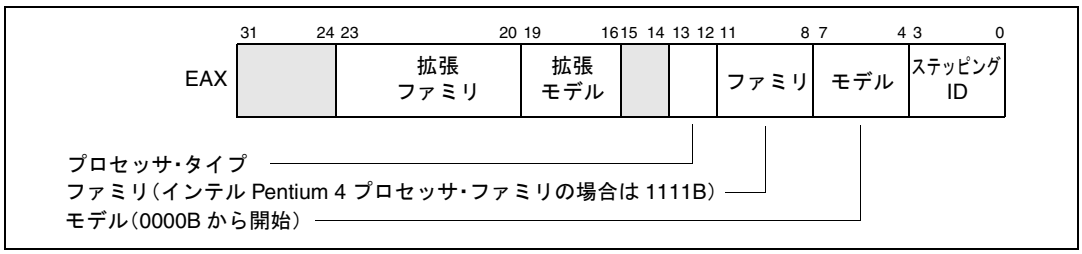


図 9-2. リセット後に EDX レジスタにストアされるバージョン情報

ステッピング ID フィールドには、プロセッサのステッピング ID または改訂レベルに使用する一意の識別子がストアされる。拡張ファミリおよび拡張モデル・フィールドは、インテル Pentium 4 プロセッサで IA-32 アーキテクチャに追加された。

9.1.4. 最初に実行される命令

ハードウェア・リセット後にフェッチされてから実行される最初の命令は、物理アドレス FFFFFFF0H に配置される。このアドレスは、プロセッサの最上位にある物理アドレスの 16 バイト下に位置している。ソフトウェア初期化コードをストアしている EPROM は、このアドレスに配置されなければならない。

プロセッサが実アドレスモードに設定されている間、アドレス FFFFFFF0H は、プロセッサのアドレス可能範囲である 1M バイトを越えている。プロセッサは、次のようにしてこの開始アドレスに初期化される。つまり CS レジスタには、表示されるセグメント・セレクタと非表示のベースアドレスという 2 つの部分があるが、実アドレスモードでは、通常、16 ビットのセグメント・セレクタ値を 4 ビット分左にシフトして 20 ビットのベースアドレスを作成すると、ベースアドレスが形成される。ただし、ハードウェアのリセット中には、CS レジスタ内のセグメント・セレクタに F000H がロードされ、ベースアドレスに FFFF0000H がロードされる。このように、開始アドレスは EIP レジスタ内の値にベースアドレスを追加することによって形成される（つまり、FFFF0000 + FFF0H = FFFFFFF0H である）。

ハードウェアのリセット後、最初に CS レジスタに新しい値がロードされると、プロセッサは、実アドレスモードでアドレス変換に適用される通常の規則に従う（つまり、[CS ベースアドレス = CS セグメント・セレクタ *16]）。EPROM ベースのソフトウェア初期化コードが完了するまで CS レジスタ内のベースアドレスが変更されないようにするため、コードに far ジャンプまたは far コールを入れてはならない。あるいは割り込みを発生させてはならない。（これらの動作によって CS セレクタの値が変更されてしまうからである。）

9.2. x87 FPU の初期化

ソフトウェア初期化コードは、プロセッサに x87 FPU が組み込まれているかどうかを確定できる。このときには CPUID 命令を使用する。確定後、コードは x87 FPU を初期化して、制御レジスタ CR0 内のフラグを設定して x87 FPU 環境の状態を反映させなければならない。

ハードウェア・リセット後、x87 FPU は、表 9-1. に示されている状態になる。この状態は、FINIT 命令または FNINIT 命令を実行したときに x87 FPU が置かれる状態（表 9-1. に示す）とは異なっている。x87 FPU を使用する場合は、ハードウェア・リセット後にソフトウェア初期化コードが FINIT/FNINIT 命令を実行しなければならない。これらの命令は、すべてのデータレジスタを空としてタグ付けし、すべての例外マスクをクリアする。そしてスタックの TOP 値を 0 にセットし、さらにデフォルトの丸め制御と精度制御の設定（最近接値と 64 ビット精度への丸め）を選択する。

INIT# ピンのアサーションによってプロセッサがリセットされた場合には、x87 FPU の状態は変わらない。

9.2.1. x87 FPU 環境の構成

初期化コードは、適切な値を制御レジスタ CR0 の MP、EM、NE の各フラグにロードする必要がある。これらのビットは、プロセッサのハードウェア・リセット時にクリアされる。各フラグに推奨する設定内容を、表 9-2. に示している。各設定は、初期化される IA-32 プロセッサによって異なる。初期化コードは、各フラグの設定またはクリア前に、存在するプロセッサのタイプをテストできる。

表 9-2. IA-32 プロセッサの EM フラグと MP フラグの推奨設定値

EM	MP	NE	IA-32 プロセッサ
1	0	1	Intel486™ SX プロセッサ、Intel386™ DX プロセッサ、Intel386 SX プロセッサのみ。マス・コプロセッサは存在しない。
0	1	1 または 0*	インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486 DX プロセッサ、インテル® 487 SX プロセッサ。マス・コプロセッサが付随している場合の Intel386 DX プロセッサと Intel386 SX プロセッサ。

注 :

*** NE フラグの設定値は、使用しているオペレーティング・システムによって異なる。**

EM フラグは、浮動小数点命令を x87 FPU が実行する（EM はクリアされる）のか、または浮動小数点命令によってデバイス使用不可の例外（#NM）が生成されるのかを決定する。後者の場合、例外ハンドラが浮動小数点演算をエミュレートできるようになる（EM = 1）。通常、x87 FPU かマス・コプロセッサが存在していれば EM フラグはクリアされ、存在していない場合は EM フラグが設定される。EM フラグが設定されていても、x87 FPU もマス・コプロセッサも浮動小数点エミュレータも存在していないときは、浮動小数点命令が実行された時点でプロセッサがハングする。

MP フラグは、WAIT/FWAIT 命令が TS フラグの設定に反応するかどうかを決定する。MP フラグがクリアされている場合は、WAIT/FWAIT 命令は TS フラグの設定を無視する。MP フラグが設定されていて TS フラグも設定されている場合は、WAIT/FWAIT 命令はデバイス使用付加の例外（#NM）を生成する。一般的には、x87 FPU が組み込まれているプロセッサには MP フラグが設定されていなければならない。また x87 FPU が組み込まれてなく、またマス・コプロセッサも存在しないプロセッサに対してはクリアされる必要がある。ただしオペレーティング・システムは、コンテキストが切り替えられるたびに浮動小数点コンテキストをセーブするように選択できる。この場合、MP ビットを設定する必要はない。

表 2-1. では、EM、MP、TS の各フラグの設定値に基づき、浮動小数点と WAIT/FWAIT 命令に対して実行される動作を示している。

NE フラグは、マスクされていない浮動小数点例外の処理を、浮動小数点エラー例外（NE が設定されているネイティブ・モード）を内部に生成して行うか、または外部割り込み（NE はクリア）によって行うかを決定する。外部割り込みコントローラを使用して数値例外ハンドラを呼び出すシステム（例えばMS-DOS* ベースのシステム）では、NE ビットはクリアされる必要がある。

9.2.2. x87 FPU ソフトウェア・エミュレーションに対するプロセッサの設定

EM フラグを設定すると、浮動小数点命令に遭遇するたびにプロセッサは、デバイス使用不可例外（#NM）を生成し、ソフトウェア例外ハンドラにトラップされる。（表 9-2. では、このフラグをいつ使用するのが適切かを示している。）EM フラグを設定すると、以下の 2 つの機能が得られる。

- x87 FPU が組み込まれてなく、また外部マス・コプロセッサにも接続されていない IA-32 プロセッサ上で、浮動小数点エミュレータを使用して x87 FPU コードを実行する機能。
- 特殊なアプリケーション用に選択された特別な（または標準外の）浮動小数点エミュレータを使用して浮動小数点コードを実行する機能。この場合、x87 FPU またはマス・コプロセッサが存在しているかどうかは関係ない。

浮動小数点命令をエミュレートするには、制御レジスタ CR0 内の EM、MP、NE の各フラグを、表 9-3. に示す値に設定しなければならない。

表 9-3. EM、MP、および NE フラグのソフトウェア・エミュレーションの設定値

CR0 ビット	値
EM	1
MP	0
NE	1

EM ビットの値に関係なく、Intel486™ SX プロセッサは、浮動小数点命令に遭遇した時点でデバイス使用不可例外（#NM）を生成する。

9.3. キャッシュのイネーブル

（Intel486™ プロセッサを始めとする）IA-32 プロセッサには、内部命令と複数のデータ・キャッシュがストアされている。各キャッシュは制御レジスタ CR0 内の CD フラグと NW フラグ（両方ともハードウェア・リセット時に設定される）をクリアするとイネーブルになる。リセット初期化後にはすべての内部キャッシュ・ラインが無効になるため、キャッシングをイネーブルにする前にキャッシュを無効にする必要はない。どの外部キャッシュも、システム固有の初期化と無効化のコード・シーケンスを使用して初期化や無効化を行うことが必要となる場合がある。

ハードウェアと、オペレーティング・システムまたはエグゼクティブの必要条件によっては、プロセッサのキャッシング機能のコンフィギュレーションが別途必要になることがある。ページレベルのキャッシングは、まず Intel486 プロセッサでは、ページ・ディレクトリとページテーブルの各エントリ内にある PCD フラグと PWT フラグを使用して制御できる。P6 ファミリ・プロセッサを始めとして、メモリタイプ範囲レジスタ（MTRR）によって、物理メモリの各領域のキャッシング特性が制御される。（Intel486 プロセッサとインテル® Pentium® プロセッサでは、外部ハードウェアを使用して物理メモリの各領域のキャッシング特性を制御できる。）インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサとシステムメモリにおけるキャッシング機能のコンフィギュレーションの詳細については、第 10 章「メモリ・キャッシュ制御」を参照。

9.4. モデル固有レジスタ（MSR）

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサには、モデル固有のレジスタ（MSR: Model-Specific Register）が複数装備されている。MSR は、その名が示すとおりプロセッサ固有のものである。言い換えると、これらのレジスタが将来の IA-32 プロセッサでサポートされる保証はないし、また同じ機能を持つ保証もない。MSR は、以下に示すハードウェアとソフトウェアに関連したいろいろな機能を制御するものとして提供されている。

- 性能モニタリング・カウンタ（15.8. 節「性能モニタリングの概要」を参照）。
- （インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのみ。）デバッグの拡張（15.4. 節「最新の分岐記録の概要」を参照）。
- （インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのみ。）マシンチェック例外処理機能と、それに随伴したマシン・チェック・アーキテクチャ（第 14 章「マシン・チェック・アーキテクチャ」を参照）。
- （インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのみ。）MTRR（10.11. 節「メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）」を参照）。

MSR の読み取りと書き込み処理は、それぞれ RDMSR 命令と WRMSR 命令を使用すると実行できる。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ、またはインテル Pentium プロセッサのソフトウェア初期化を実行する場合は、MSR の多くを初期化して性能モニタリング・イベント、ランタイム・マシン・チェック、物理メモリのメモリタイプなどをセットアップする必要がある。

インテルPentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサ、Pentiumプロセッサに使用できる性能モニタリング・カウンタのリストは、付録A「性能モニタリング・イベント」に記載されている。またインテルPentium 4プロセッサ、インテルXeonプロセッサ、P6ファミリ・プロセッサ、インテルPentiumプロセッサに使用できるMSRのリストは、付録B「モデル固有レジスタ（MSR）」に記載されている。MSRのいろいろなグループの機能を本書のどの箇所で説明しているかについては、この項でこれまでに記載したリファレンスを参照。

9.5. メモリタイプ範囲レジスタ（MTRR）

メモリタイプ範囲レジスタ（MTRR）は、インテル® Pentium® Proプロセッサによって IA-32 アーキテクチャに導入された。各メモリタイプ範囲レジスタを使用すると、キャッシュする方式（またはキャッシュしない方式）を、選択した物理アドレス範囲に対してシステムメモリ内で指定できる。またRAM、ROM、フレーム・バッファ・メモリ、メモリマップドI/Oデバイスなどのさまざまなタイプのメモリに対してメモリのアクセスを最適化できる。

一般的に、MTRRの初期化を処理するのはソフトウェア初期化コードまたはBIOSであり、オペレーティング・システムまたはエグゼクティブの機能ではない。少なくとも、すべてのMTRRSは0にクリアされなければならない。アンキャッシュド（UC: uncached）メモリタイプがこれによって選択される。MTRRの詳細については、10.11.節「メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）」を参照。

9.6. SSE、SSE2、SSE3の拡張命令の初期化

SSE、SSE2、SSE3拡張命令を搭載したプロセッサの場合、これらの命令を実行可能にするには、初期化の際にいくつかのステップを実行する必要がある。

- CPUID機能フラグをチェックし、SSE、SSE2、SSE3の拡張命令の有無（それぞれEDXのビット25、26とECXのビット0）、FXSAVE命令とFXRSTOR命令のサポートの有無（EDXのビット24）を調べる。また、CLFLUSH命令のサポートの有無（EDXのビット19）もチェックする必要がある。CPUID機能フラグは、EAXレジスタに1をセットしてCPUID命令を実行すると、EDXレジスタとECXレジスタにロードされる。
- OSFXSR フラグ（制御レジスタ CR4 のビット 9）をセットする。このフラグは、FXSAVE命令とFXRSTOR命令でSSE、SSE2、SSE3の実行環境（XMMレジスタおよびMXCSRレジスタ）の保存と復元がサポートされることを示す。OSFXSRフラグの説明については、2.5.節「制御レジスタ」を参照のこと。
- OSXMMEXCPTフラグ（制御レジスタCR4のビット10）をセットする。このフラグは、SSE、SSE2、SSE3のSIMD浮動小数点例外（#XF）の処理がサポートされる

ことを示す。OSXMMEXCPT フラグの説明については、2.5. 節「制御レジスタ」を参照のこと。

- SSE、SSE2、SSE3 の SIMD 浮動小数点命令で必要とされる動作モードに応じて、MXCSR レジスタにマスクビットとフラグをセットする。MXCSR レジスタのビットおよびフラグについては、『IA-32 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』の第 10 章の「MXCSR 制御 / ステータス・レジスタ」を参照のこと。

9.7. 実アドレスモード動作に対するソフトウェア初期化

（電源投入または RESET# ピンのアサートのどちらかを介した）ハードウェア・リセットの後、プロセッサは実アドレスモードに設定されて、物理アドレス FFFFFFF0H からソフトウェア初期化コードの実行を開始する。基本的なシステム機能（割り込みや例外を処理する実モード IDT など）を扱うために必要なデータ構造が、ソフトウェア初期化コードによって最初にセットアップされなければならない。プロセッサを実アドレスモードの状態にしておく場合は、ソフトウェアによってオペレーティング・システムまたはエグゼクティブの追加のコード・モジュールやデータ構造をロードし、アプリケーション・プログラムが実アドレスモードで確実に実行されるようにする必要がある。

プロセッサを保護モードで動作させる場合は、ソフトウェアによって保護モードでの動作に必要なデータ構造をロードしてから保護モードに切り替えなければならない。ロードする必要のある保護モードのデータ構造については、9.8. 節「保護モード動作に対するソフトウェア初期化」で説明している。

9.7.1. 実アドレスモード IDT

実アドレスモードでは、メモリにロードしなければならない唯一のシステムデータ構造が IDT（「割り込みベクタテーブル」とも呼ばれる）である。デフォルトでは、IDT のベースのアドレスは物理アドレス 0H である。このアドレスの変更は、LIDT 命令を使用して IDTR 内のベースアドレスを変更して行う。割り込みがイネーブルになる前に、ソフトウェア初期化コードによって割り込みハンドラポインタと例外ハンドラポインタを IDT にロードする必要がある。

実際の割り込みハンドラコードと例外ハンドラコードは、EPROM か RAM のどちらかにストアできる。ただしコードは、実アドレスモードにあるプロセッサの 1M バイトのアドレス可能な範囲内に配置しなければならない。ハンドラコードを RAM にストアする場合は、コードを IDT と一緒にロードする必要がある。

9.7.2. NMI 割り込みの処理

NMI 割り込みは、（複数の NMI がネストされている場合を除いて）常にイネーブルである。IDT と NMI 割り込みハンドラを RAM にロードしなければならない場合、ハードウェア・リセットの後に一定の時間、NMI 割り込みを処理できなくなる。この時間内にハードウェアは、IDT と必要な NMI ハンドラ・ソフトウェアがロードされるまで、NMI 割り込みによってコードの実行が妨げられないようにするメカニズムを備えている必要がある。以下に、プロセッサが初期化された最初の状態で NMI を処理できる 2 つの例を示す。

- 簡単な IDT と NMI 割り込みハンドラを EPROM 内で使用して、リセット初期化のすぐ後に NMI 割り込みを処理できる。
- I/O ポート内のフラグが制御する AND ゲートを介して NMI# 信号を渡すと、システム・ハードウェアによって NMI をイネーブルにもディスエーブルにもできる。プロセッサがリセットされる時点で、ハードウェアでフラグをクリアできる。また NMI 割り込み処理の準備ができた時点で、ソフトウェアでフラグを設定できる。

9.8. 保護モード動作に対するソフトウェア初期化

ハードウェア・リセットの後にプロセッサは実アドレスモードになる。初期化プロセスのこの時点で、さらに続くプロセッサの初期化をサポートするために、一部の基本データ構造やコード・モジュールを物理メモリにロードしなければならない。詳細については、9.7. 節「実アドレスモード動作に対するソフトウェア初期化」で説明している。プロセッサを保護モードに切り替えるために、ソフトウェア初期化コードは、最低限の数の保護モードデータ構造とコード・モジュールをメモリにロードして、保護モードにあるプロセッサの確実な動作をサポートする必要がある。これらのデータ構造には以下のものが含まれる。

- 保護モード IDT。
- GDT。
- TSS。
- LDT（オプション）。
- ページングを使用する場合は、少なくとも 1 つのページ・ディレクトリと 1 つのページテーブル。
- プロセッサが保護モードに切り替わるときに実行されるコードをストアしているコード・セグメント。
- 必要な割り込みと例外ハンドラをストアしているコード・モジュール（1 つまたは複数）。

またソフトウェア初期化コードは、プロセッサを保護モードに切り替える前に、以下の各システムレジスタを初期化する必要がある。

- GDTR。
- IDTR（オプション）。保護モードへの切り替えの直後、割り込みをイネーブルにする前に、このレジスタも初期化できる。
- 制御レジスタ CR1 ～ CR4。
- （インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのみ。）メモリタイプ範囲レジスタ（MTRR）。

上記のデータ構造、コード・モジュール、システムレジスタが初期化された状態で、PE フラグ（ビット 0）を設定している値を制御レジスタ CR0 にロードすると、プロセッサを保護モードに切り替えられる。

9.8.1. 保護モードシステムのデータ構造

ソフトウェア初期化実行時にメモリにロードされる保護モードシステムのデータ構造の内容は、保護モードにあるオペレーティング・システムまたはエグゼクティブがサポートしようとするメモリ管理の種類（フラット、ページング付きフラット、セグメント、ページング付きセグメント）によって大部分が決定される。

ページングのないフラットなメモリモデルを実現する場合、ソフトウェア初期化コードは少なくとも 1 つのコードと 1 つのデータ・セグメントのディスクリプタを GDT にロードしなければならない。また最初の GDT エントリ内にもヌルのディスクリプタが必要になる。スタックは通常の読み取り / 書き込みデータ・セグメントに配置できるため、スタック専用のディスクリプタを必要としない。またページング付きのフラットなメモリモデルも、1 つのページ・ディレクトリと、少なくとも 1 つのページテーブルを必要とする（ただし、すべてのページが 4 M バイトである場合を除く。この場合は 1 つのページ・ディレクトリだけが必須である）。9.8.3. 項「ページングの初期化」を参照。

GDT を使用できるようにするには、LGDT 命令を使用して、GDT のベースアドレスとリミットを GDTR レジスタにロードしなければならない。

マルチセグメントのモデルでは、各アプリケーション・プログラムのセグメントと LDT だけでなく、オペレーティング・システム用の追加セグメントが必要になることがある。LDT は GDT 内にセグメント・ディスクリプタを必要とする。一部のオペレーティング・システムは、必要に応じて新しいセグメントと LDT を割り当てる。これによって、動的なプログラミング環境を扱うための最大限の柔軟性が得られる。しかし大多数のオペレーティング・システムは、すべてのタスクに単一の LDT を使用し、GDT のエントリをあらかじめ割り当てている。プロセス・コントローラなどの埋め込

みシステムが、固定数のアプリケーション・プログラムに固定数のセグメントとLDTをあらかじめ割り当てる場合がある。この割り当てによって、リアルタイム・システムのソフトウェア環境が簡単にまた能率的に構築される。

9.8.2. 保護モードにおける例外と割り込みの初期化

ソフトウェア初期化コードは、プロセッサが生成できるそれぞれの例外ベクタに対して、少なくとも1つの保護モードIDTにゲート・ディスクリプタをロードしなければならない。割り込みゲートまたはトラップゲートを使用する場合、ゲート・ディスクリプタはすべて、必要な例外ハンドラをストアしている同一のコード・セグメントを指すことができる。タスクゲートを使用する場合は、1つのTSSと、それに付随しているコード、データ、タスク・セグメントが、タスクゲートで呼び出される例外ハンドラのそれぞれに必要となる。

ハードウェアが割り込みの生成を許可する場合は、1つまたは複数の割り込みハンドラに対してIDT内でゲート・ディスクリプタを提供しなければならない。

IDTを使用できるようにするには、IDTのベースアドレスとリミットを、LIDT命令を使用してIDTRレジスタにロードしなければならない。この操作は、一般的には保護モードへの切り替え直後に実行する。

9.8.3. ページングの初期化

ページングは、制御レジスタCR0内のPGフラグが制御する。このフラグがクリア状態（ハードウェア・リセット後の状態）のときは、ページング機構がオフになる。フラグが設定されている場合、ページングはイネーブルになる。PGフラグを設定する前に、以下のデータ構造とレジスタを初期化しなければならない。

- ソフトウェアは、少なくとも1つのページ・ディレクトリと、1つのページテーブルを物理メモリにロードしなければならない。ページ・ディレクトリにそれ自体を指すディレクトリ・エントリがストアされている場合（つまりページ・ディレクトリとページテーブルが同じページにある場合）、または4Mバイトだけのページが使用されている場合、ページテーブルを削除できる。
- 制御レジスタCR3（PDBRレジスタとも呼ばれる）に、ページ・ディレクトリの物理ベースアドレスがロードされる。
- （オプション）ソフトウェアは、スーパーバイザ・モードに対して、コードおよびデータ・ディスクリプタの1セットをGDTまたはLDT内で提供できる。またユーザモードに対して他のセットを提供できる。

このページング初期化が完了すると、PGフラグとPEフラグが設定されている制御レジスタCR0にイメージがロードされ、プロセッサが保護モードに切り替えられると同

時にページングがイネーブルになる。(プロセッサが保護モードに切り替えられるまで、ページング機能は使用できない。)

9.8.4. マルチタスキングの初期化

マルチタスキング機構を使用しないときや、特権レベルの変更が許可されない場合は、TSS をメモリにロードしたりタスクレジスタを初期化する必要はない。

マルチタスキング機構を使用するか、または特権レベルの変更が許可されている場合(あるいはその両方の場合)、ソフトウェア初期化コードは、少なくとも 1 つの TSS と付随している TSS ディスクリプタをロードしなければならない。(TSS は、特権レベルの変更には必須である。なぜなら、特権レベル 0、1、2 の各スタック・セグメントに対するポインタと、これらのスタックに対するスタックポインタが TSS から獲得されるためである。)TSS ディスクリプタを作成するときにビジーとマーク付けしてはならない。ビジーのマーク付けは、プロセッサによって、タスクスイッチを実行するときの副作用としてだけ行われなければならない。LDT のディスクリプタの場合と同じように、TSS ディスクリプタは GDT 内に存在する。

プロセッサが保護モードに切り替えられた後には、LTR 命令を使用して、TSS ディスクリプタのセグメント・セレクタをタスクレジスタにロードすることができる。この命令は TSS ディスクリプタをビジーとしてマーク付けするが、タスクスイッチは実行しない。ただしプロセッサは、TSS を使用して、特権レベル 0、1、2 の各スタックへのポインタを探すことができる。TSS のセグメント・セレクタは、ソフトウェアが保護モードで最初のタスクスイッチを実行する前にロードされなければならない。なぜなら、タスクスイッチにより、現行のタスク状態が TSS にコピーされるためである。

LTR 命令の実行が完了すると、タスクレジスタに対してさらに継続して操作が実行される。他のセグメントや LDT の場合と同じように、TSS と TSS ディスクリプタは、事前に割り当てるか、あるいは必要に応じて割り当てられる。

9.9. モード切り替え

プロセッサを保護モードで使用するには、モードの切り替えを実アドレスモードから実行しなければならない。いったん保護モードに移行した後は、通常の場合、ソフトウェアが実アドレスモードに戻る必要はない。実アドレスモード(8086 モード)で実行するように記述されたソフトウェアを実行するには、実アドレスモードに再度切り替えるよりも、ソフトウェアを仮想 8086 モードで実行するほうが一般的には便利である。

9.9.1. 保護モードへの切り替え

保護モードに切り替える前に、システムデータ構造とコード・モジュールの必要最小限のセットをメモリにロードしなければならない。詳細については、9.8.節「保護モード動作に対するソフトウェア初期化」で説明している。各テーブルが作成された後は、ソフトウェア初期化コードを保護モードに切り替えられる。

保護モードに入るには、CR0 レジスタ内の PE フラグを設定する MOV CR0 命令を実行する。（同じ命令で、レジスタ CR0 内の PG フラグを設定してページングをイネーブルにできる。）保護モードでの実行は 0 の CPL から開始する。

32 ビット IA-32 プロセッサでは、保護モードへの切り替えに必要な条件が少し異なる。すべての 32 ビット IA-32 プロセッサ間でコードの上位互換性と下位互換性を確保するためには、以下の操作手順を実行する。

1. 割り込みをディスエーブルにする。CLI 命令によってマスク可能ハードウェア割り込みがディスエーブルになる。NMI 割り込みは、外部回路によってディスエーブルにできる。（ソフトウェアは、モード切り替えの動作中に例外または割り込みが発生しないように保証する必要がある。）
2. LGDT 命令を実行して、GDTR レジスタに GDT のベースアドレスをロードする。
3. 制御レジスタ CR0 の PE フラグ（またオプションで PG フラグ）を設定する MOV CR0 命令を実行する。
4. MOV CR0 命令のすぐ後で、far JMP 命令または far CALL 命令を実行する。（この操作は通常、命令ストリーム内の次の命令への far ジャンプまたは far コールになる。）
5. MOV CR0 命令の直後に JMP 命令または CALL 命令を実行すると、実行の流れが変化し、プロセッサが順次処理される。
6. ページングがイネーブルな場合、MOV CR0 命令のコードと、JMP 命令または CALL 命令のコードは、アイデンティティ・マッピングされたページから得たものでなければならない（つまり、ジャンプ前のリニアアドレスが、ページングと保護モードがイネーブルになった後の物理アドレスと同一でなければならない）。JMP 命令または CALL 命令のターゲット命令は、アイデンティティ・マッピングされる必要はない。
7. ローカル・ディスクリプタ・テーブルを使用する場合には、LLDT 命令を実行して LDTR レジスタに LDT のセグメント・セレクタをロードする。
8. LTR 命令を実行して、セグメント・セレクタ付きタスクレジスタを初期保護モード・タスクへロードする。またはタスクスイッチに関する TSS 情報を保存できるメモリの書き込み可能領域へロードする。
9. 保護モードに移行した後、セグメント・レジスタは実アドレスモードで保持していた内容をそのまま保持し続ける。操作手順 4 の JMP 命令または CALL 命令が CS レジス

タをリセットする。以下のどれかを実行して、残りのセグメント・レジスタの内容を更新する。

- DS、SS、ES、FS、GS の各セグメント・レジスタを再ロードする。ES、FS、GS の各レジスタのなかで使用しないものがある場合、そのレジスタにヌルセレクタをロードする。
- JMP 命令または CALL 命令を新しいタスクに対して実行する。これによって、セグメント・レジスタの値がリセットされ、新しいコード・セグメントに分岐する。

10. LIDT 命令を実行して、IDTR レジスタに保護モード IDT のアドレスとリミットをロードする。
11. STI 命令を実行して、マスク可能ハードウェア割り込みをイネーブルにし、また必要なハードウェア動作を実行して NMI 割り込みをイネーブルにする。

上記の手順 3 と手順 4 の間に他の命令が存在すると、不規則な障害が発生することがある。システム管理モードで、メモリを参照する命令が手順 3 と手順 4 の間に挿入されたときのような状況では、障害は直ちに認識される。

9.9.2. 実アドレスモードへの再切り替え

ソフトウェアが MOV CR0 命令によって CRO レジスタ内の PE ビットをクリアした場合、プロセッサは実アドレスモードに切り替えられる。実アドレスモードに再移行するときは、以下の操作手順を実行する必要がある。

1. 割り込みをディスエーブルにする。CLI 命令によってマスク可能ハードウェア割り込みがディスエーブルになる。NMI 割り込みは、外部回路によってディスエーブルにできる。
2. ページングがイネーブルである場合は、以下の各操作を行う。
   - プログラム制御を、物理アドレスにアイデンティティ・マッピングされている（つまり物理アドレスと同一の）リニアアドレスに転送する。
   - GDT と IDT がアイデンティティ・マッピングされたページにあるようにする。
   - CR0 レジスタ内の PG ビットをクリアする。
   - 0H を CR3 レジスタに移動して TLB をフラッシュする。
3. プログラム制御を、64K バイト（FFFH）のリミットを持つ読み取り可能セグメントに転送する。この動作によって、実アドレスモードで必要になるセグメント・リミットが CS レジスタにロードされる。
4. SS、DS、ES、FS、GS の各セグメント・レジスタに、以下の値（実アドレスモードに適した値）をストアしているディスクリプタのセレクタをロードする。
   - リミット = 64 K バイト（0FFFFH）

- バイト・グラニュラリティ（G = O）
- 上方拡張（E = O）
- 書き込み可能（W = 1）
- 存在（P = 1）
- ベース = 任意の値

5. セグメント・レジスタには、ヌル以外のセグメント・セレクタがロードされなければならない。そうしないと、実アドレスモードでセグメント・レジスタを使用できなくなる。セグメント・レジスタが再ロードされない場合は、保護モード時にロードされたディスクリプタの属性によって実行が継続されることに注意しなければならない。
6. LIDT 命令を実行して、実アドレスモードの割り込みテーブルを指す。これは 1M バイトの実アドレスモードのアドレス範囲内にある。
7. CR0 レジスタ内の PE フラグをクリアして、実アドレスモードに切り替える。
8. far JMP 命令を実行して実アドレスモードのプログラムにジャンプする。この動作によって、命令キューがフラッシュされ、適切なベース値とアクセス権値が CS レジスタにロードされる。
9. 実アドレスモードのコードに必要であれば、SS、DS、ES、FS、GS の各レジスタをロードする。これらのレジスタのどれかを実アドレスモードで使用しない場合は、そのレジスタに O を書き込む。
10. STI 命令を実行して、マスク可能ハードウェア割り込みをイネーブルにする。また必要なハードウェア動作を実行して NMI 割り込みをイネーブルにする。

注記

操作手順 1 ～ 9 で実行されるコードはすべて、単一のページ内になければならない。またそのページ内にあるリニアアドレスは、物理アドレスにアイデンティティ・マッピングされなければならない。

9.10. 初期化とモード切り替えの例

本節では、アプリケーションに組み込むことができる初期化とモード切り替えの例を扱う。このコードは、本来は Intel386™ プロセッサを初期化するために記述されたものであるが、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでも問題なく実行できる。本例に示すコードは、EPROM 内に常駐して、プロセッサのハードウェア・リセット後に実行されることを意図しているものである。コードの機能は以下の各動作を実行する。

- 基本的な実アドレスモード動作環境を設定する。
- 必要な保護モードシステムのデータ構造をRAMにロードする。
- 保護モード動作用に、データ構造に対する必要なポインタと適切なフラグ設定をシステムレジスタにロードする。
- プロセッサを保護モードに切り替える。

図9-3.では、ハードウェア・リセット後のプロセッサに対する物理メモリのレイアウトを示している。またこの例の開始点も示している。初期化コードをストアしているEPROMは、プロセッサの物理メモリアドレス範囲の上限に常駐していて、アドレスFFFFFFFFHから始まりそこから下方に降りていく。実行される最初の命令のアドレスはFFFFFFF0Hにある。これは、ハードウェア・リセット後のプロセッサに対するデフォルトの開始アドレスである。

表9-4.では、この例で実行される主な手順を要約している。また例9-1.では、この例のソースリスト（ファイル名「STARTUP.ASM」）を示している。表9-4.に示す行番号は、ソースリストに対応している。

この例に関する追加注記を以下に記述する。

- プロセッサが保護モードに切り替えられると、FFFF0000Hのオリジナル・コード・セグメントの（CSレジスタの隠された部分に置かれている）ベースアドレス値が保持され、EIPレジスタ内の現行オフセットから実行が継続される。プロセッサはこのようにして、farジャンプまたはfarコールが新しいコード・セグメントに対して行われるまで、EPROM内のコードを実行し続ける。ジャンプか呼び出しが行われた時点で、CSレジスタ内のベースアドレスが変更される。
- マスク可能ハードウェアの割り込みは、ハードウェア・リセットの後にディスエーブルになる。この状態は、必要な割り込みハンドラがインストールされるまで続く。NMI割り込みはリセットの後にディスエーブルにならない。したがって、NMIハンドラがロードされてプロセッサが利用できるようになるまで、NMI#ピンのアサートを禁止する必要がある。
- 一時GDTを使用すると、EPROMからRAM領域のどこへでもテーブルを簡易転送できる。1つのGDTエントリは、アドレスOを指すベースと4Gバイトのリミットによって構築される。DSレジスタとESレジスタにこのディスクリプタがロードされると、一時GDTが不要となり、アプリケーションGDTと置き換えられる。
- このコードは1つのTSSをロードするけれども、LDTはロードしない。複数のTSSがアプリケーション内に存在する場合は、RAMにロードする必要がある。LDTが存在する場合は、これらも同様にロードできる。

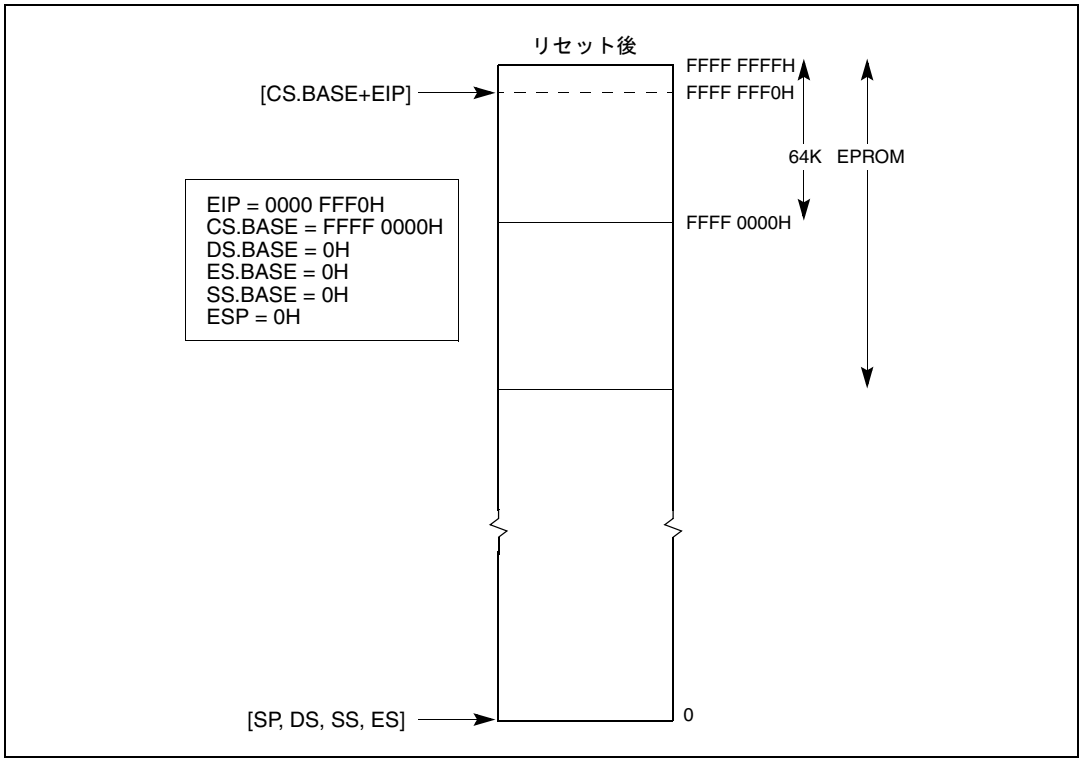


図 9-3. リセット後のプロセッサ状態

表 9-4. STARTUP.ASM ソースリストの主な初期化手順

STARTUP.ASM 行番号		説明
開始行	終了行	
157	157	EPROM 内のエントリコードにジャンプ（ショート）する
162	169	1 つのエントリを使用して RAM 内に一時 GDT を構築する 0 ー ヌル 1 ー R/W データ・セグメント、ベース = 0、リミット = 4GB
171	172	GDTR をロードして一時 GDT を指す
174	177	設定された PE フラグを CR0 にロードして、保護モードに切り替える
179	181	NEAR ジャンプをして、実モード命令キューをクリアする
184	186	GDT[1] ディスクリプタを DS レジスタと ES レジスタにロードし、両レジスタが物理メモリ空間全体を指すようにする
188	195	新しい保護モードによって課される特定のボード初期化を実行する
196	218	アプリケーションの GDT を、ROM から RAM にコピーする
220	238	アプリケーションの IDT を、ROM から RAM にコピーする
241	243	アプリケーションの GDTR をロードする
244	245	アプリケーションの IDTR をロードする

表 9-4. STARTUP.ASM ソースリストの主な初期化手順（続き）

STARTUP.ASM 行番号		説明
開始行	終了行	
247	261	アプリケーションの TSS を、ROM から RAM にコピーする
263	267	TSS ディスクリプタと、GDT 内にある他の別名（GDT の別名、または IDT の別名）を更新する
277	277	LTR 命令を使用してタスクレジスタをロードする（タスクスイッチは除く）
282	286	SS、ESP に、アプリケーションの TSS 内で検出された値をロードする
287	287	アプリケーションの TSS 内で検出された EFLAGS 値をプッシュする
288	288	アプリケーションの TSS 内で検出された CS 値をプッシュする
289	289	アプリケーションの TSS 内で検出された EIP 値をプッシュする
290	293	DS、ES に、アプリケーションの TSS 内で検出された値をロードする
296	296	IRET を実行する。上記の値をポップし、アプリケーション・コードを入力する

9.10.1. アセンブラの使用

この例では、インテル・アセンブラ ASM386 と作成ツール BLD386 を使用して、初期化コード・モジュールのアセンブルと作成を行っている。インテル ASM386 と BLD386 ツールを使用するときには、以下のことが当然行われるものと想定されている。

- ASM386 は、コード・セグメント属性に従った正しいオペランド・サイズのオペコードを生成する。属性は、ASM386 の呼び出し制御、またはコード・セグメントの定義内のどちらかによって割り当てられる。
- 実アドレスモードで実行する予定のコード・セグメントを定義する場合は、USE16 属性に設定しなければならない。このコード・セグメントの命令（例えば、MOV EAX、EBX）内で 32 ビットのオペランドが使用される場合、アセンブラは、命令に対するオペランド・プリフィックスを自動的に生成し、デフォルトのコード・セグメント属性が 16 ビットであっても、32 ビットの操作を実行するようプロセッサに強制する。
- インテルの ASM386 アセンブラによって、16 ビット命令または 32 ビット命令（例えば、LGDTW、LGDTD、IRETD）を特定して使用することができる。一般命令の LGDT を使用する場合は、デフォルトのセグメント属性を使用して正しいオペコードを生成する。

9.10.2. STARTUP.ASM リスト

例 9-1. では、プロセッサを保護モードに移行させるための高度なコード例を記載している。このリストには、オペコードやオフセット情報は含まれていない。

例 9-1. STARTUP.ASM

```
MS-DOS* 5.0(045-N) 386(TM) MACRO ASSEMBLER STARTUP  09:44:51  08/19/92
PAGE 1

MS-DOS 5.0(045-N) 386(TM) MACRO ASSEMBLER V4.0, ASSEMBLY OF MODULE STARTUP
OBJECT MODULE PLACED IN startup.obj
ASSEMBLER INVOKED BY: f:\386tools\ASM386.EXE startup.a58 pw (132 )

LINE     SOURCE

       NAME    STARTUP

             ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;
 ;   ASSUMPTIONS:
 ;
 ;     1.  The bottom 64K of memory is ram, and can be used for
 ;         scratch space by this module.
 ;
 ;     2.  The system has sufficient free usable ram to copy the
 ;         initial GDT, IDT, and TSS
 ;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 ; configuration data - must match with build definition

 CS_BASE       EQU     0FFFF0000H

  ; CS_BASE is the linear address of the segment STARTUP_CODE
  ; - this is specified in the build language file

 RAM_START     EQU     400H

 ; RAM_START  is the start of free, usable ram in the linear
 ; memory  space.   The GDT,  IDT, and  initial TSS  will be
 ; copied above this space, and a small data segment will be
 ; discarded at  this linear  address.   The 32-bit  word at
 ; RAM_START will contain  the linear  address of  the first
 ; free byte above the copied tables - this may be useful if
 ; a memory manager is used.

 TSS_INDEX    EQU     10

 ; TSS_INDEX is the  index of the  TSS of the  first task to
```

```
35  ; run after startup
36
37
38   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
39
40  ; ---------------------------- STRUCTURES and EQU ---------------
41  ; structures for system data
42
43  ; TSS structure
44  TASK_STATE  STRUC
45      link                    DW ?
46      link_h                  DW ?
47      ESP0                    DD ?
48      SS0                     DW ?
49      SS0_h                   DW ?
50      ESP1                    DD ?
51      SS1                     DW ?
52      SS1_h                   DW ?
53      ESP2                    DD ?
54      SS2                     DW ?
55      SS2_h                   DW ?
56      CR3_reg                 DD ?
57      EIP_reg                 DD ?
58      EFLAGS_reg              DD ?
59      EAX_reg                 DD ?
60      ECX_reg                 DD ?
61      EDX_reg                 DD ?
62      EBX_reg                 DD ?
63      ESP_reg                 DD ?
64      EBP_reg                 DD ?
65      ESI_reg                 DD ?
66      EDI_reg                 DD ?
67      ES_reg                  DW ?
68      ES_h                    DW ?
69      CS_reg                  DW ?
70      CS_h                    DW ?
71      SS_reg                  DW ?
72      SS_h                    DW ?
73      DS_reg                  DW ?
74      DS_h                    DW ?
75      FS_reg                  DW ?
76      FS_h                    DW ?
77      GS_reg                  DW ?
78      GS_h                    DW ?
79      LDT_reg                 DW ?
80      LDT_h                   DW ?
81      TRAP_reg                DW ?
82      IO_map_base             DW ?
83  TASK_STATE  ENDS
84
85  ; basic structure of a descriptor
86  DESC    STRUC
```

```
 87      lim_0_15                DW ?
 88      bas_0_15                DW ?
 89      bas_16_23               DB ?
 90      access                  DB ?
 91      gran                    DB ?
 92      bas_24_31               DB ?
 93  DESC    ENDS
 94
 95  ; structure for use with LGDT and LIDT instructions
 96  TABLE_REG   STRUC
 97      table_lim               DW ?
 98      table_linear            DD ?
 99  TABLE_REG   ENDS
100
101  ; offset of GDT and IDT descriptors in builder generated GDT
102  GDT_DESC_OFF    EQU 1*SIZE(DESC)
103  IDT_DESC_OFF    EQU 2*SIZE(DESC)
104
105  ; equates for building temporary GDT in RAM
106  LINEAR_SEL          EQU     1*SIZE (DESC)
107  LINEAR_PROTO_LO     EQU     00000FFFFH  ; LINEAR_ALIAS
108  LINEAR_PROTO_HI     EQU     000CF9200H
109
110  ; Protection Enable Bit in CR0
111  PE_BIT  EQU 1B
112
113  ; -------------------------------------------------------------
114
115  ; ----------------------- DATA SEGMENT----------------------
116
117  ; Initially, this  data segment starts at linear 0, according
118  ; to the processor's power-up state.
119
120  STARTUP_DATA    SEGMENT RW
121
122  free_mem_linear_base    LABEL   DWORD
123  TEMP_GDT                LABEL   BYTE  ; must be first in segment
124  TEMP_GDT_NULL_DESC   DESC    <>
125  TEMP_GDT_LINEAR_DESC DESC    <>
126
127  ; scratch areas for LGDT and LIDT instructions
128  TEMP_GDT_SCRATCH TABLE_REG   <>
129  APP_GDT_RAM     TABLE_REG    <>
130  APP_IDT_RAM     TABLE_REG    <>
131          ; align end_data
132  fill    DW      ?
133
134  ; last thing in this segment - should be on a dword boundary
135  end_data    LABEL   BYTE
136
137  STARTUP_DATA    ENDS
138  ; -------------------------------------------------------------
```

```
139
140
141  ; ----------------------- CODE SEGMENT----------------------
142  STARTUP_CODE SEGMENT ER PUBLIC USE16
143
144  ; filled in by builder
145      PUBLIC  GDT_EPROM
146  GDT_EPROM   TABLE_REG   <>
147
148  ; filled in by builder
149      PUBLIC  IDT_EPROM
150  IDT_EPROM   TABLE_REG   <>
151
152  ; entry point into startup code - the bootstrap will vector
153  ; here  with a  near JMP  generated by  the builder.   This
154  ; label must be in the top 64K of linear memory.
155
156      PUBLIC  STARTUP
157  STARTUP:
158
159  ; DS,ES address the bottom 64K of flat linear memory
160      ASSUME  DS:STARTUP_DATA, ES:STARTUP_DATA
161  ; See 図 9-4.
162  ; load GDTR with temporary GDT
163          LEA     EBX,TEMP_GDT  ; build the TEMP_GDT in low ram,
164          MOV     DWORD PTR [EBX],0   ; where we can address
165          MOV     DWORD PTR [EBX]+4,0
166          MOV     DWORD PTR [EBX]+8, LINEAR_PROTO_LO
167          MOV     DWORD PTR [EBX]+12, LINEAR_PROTO_HI
168          MOV     TEMP_GDT_scratch.table_linear,EBX
169          MOV     TEMP_GDT_scratch.table_lim,15
170
171                  DB      66H         ; execute a 32 bit LGDT
172          LGDT    TEMP_GDT_scratch
173
174  ; enter protected mode
175          MOV     EBX,CR0
176          OR      EBX,PE_BIT
177          MOV     CR0,EBX
178

179   ; clear prefetch queue
180          JMP     CLEAR_LABEL
181  CLEAR_LABEL:
182
183   ; make DS and ES address 4G of linear memory
184          MOV     CX,LINEAR_SEL
185          MOV     DS,CX
186          MOV     ES,CX
187
188    ; do board specific initialization
189    ;
```

```
190                 ;
191                 ; ......
192                 ;
193
194
195         ; See 図 9-5.
196         ; copy EPROM GDT to ram at:
197         ;                 RAM_START + size (STARTUP_DATA)
198         MOV     EAX,RAM_START
199         ADD     EAX,OFFSET (end_data)
200         MOV     EBX,RAM_START
201         MOV     ECX, CS_BASE
202         ADD     ECX, OFFSET (GDT_EPROM)
203         MOV     ESI, [ECX].table_linear
204         MOV     EDI,EAX
205         MOVZX   ECX, [ECX].table_lim
206         MOV     APP_GDT_ram[EBX].table_lim,CX
207         INC     ECX
208         MOV     EDX,EAX
209         MOV     APP_GDT_ram[EBX].table_linear,EAX
210         ADD     EAX,ECX
211     REP MOVS    BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
212
213         ; fixup GDT base in descriptor
214         MOV     ECX,EDX
215         MOV     [EDX].bas_0_15+GDT_DESC_OFF,CX
216         ROR     ECX,16
217         MOV     [EDX].bas_16_23+GDT_DESC_OFF,CL
218         MOV     [EDX].bas_24_31+GDT_DESC_OFF,CH
219
220         ; copy EPROM IDT to ram at:
221         ; RAM_START+size(STARTUP_DATA)+SIZE (EPROM GDT)
222         MOV     ECX, CS_BASE
223         ADD     ECX, OFFSET (IDT_EPROM)
224         MOV     ESI, [ECX].table_linear
225         MOV     EDI,EAX
226         MOVZX   ECX, [ECX].table_lim
227         MOV     APP_IDT_ram[EBX].table_lim,CX
228         INC     ECX
229         MOV     APP_IDT_ram[EBX].table_linear,EAX
230         MOV     EBX,EAX
231         ADD     EAX,ECX
232     REP MOVS    BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
233
234                 ; fixup IDT pointer in GDT
235         MOV     [EDX].bas_0_15+IDT_DESC_OFF,BX
236         ROR     EBX,16
237         MOV     [EDX].bas_16_23+IDT_DESC_OFF,BL
238         MOV     [EDX].bas_24_31+IDT_DESC_OFF,BH
239
240                 ; load GDTR and IDTR
241         MOV     EBX,RAM_START
```

```
242                 DB      66H         ; execute a 32 bit LGDT
243         LGDT    APP_GDT_ram[EBX]
244                 DB      66H         ; execute a 32 bit LIDT
245         LIDT    APP_IDT_ram[EBX]
246
247                 ; move the TSS
248         MOV     EDI,EAX
249         MOV     EBX,TSS_INDEX*SIZE(DESC)
250         MOV     ECX,GDT_DESC_OFF ;build linear address for TSS
251         MOV     GS,CX
252         MOV     DH,GS:[EBX].bas_24_31
253         MOV     DL,GS:[EBX].bas_16_23
254         ROL     EDX,16
255         MOV     DX,GS:[EBX].bas_0_15
256         MOV     ESI,EDX
257         LSL     ECX,EBX
258         INC     ECX
259         MOV     EDX,EAX
260         ADD     EAX,ECX
261     REP MOVS    BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
262
263                 ; fixup TSS pointer
264         MOV     GS:[EBX].bas_0_15,DX
265         ROL     EDX,16
266         MOV     GS:[EBX].bas_24_31,DH
267         MOV     GS:[EBX].bas_16_23,DL
268         ROL     EDX,16
269     ;save start of free ram at linear location RAMSTART
270         MOV     free_mem_linear_base+RAM_START,EAX
271
272     ;assume no  LDT used in  the initial task  - if necessary,
273     ;code  to move the LDT could be added, and should resemble
274     ;that used to move the TSS
275
276     ; load task register
277         LTR     BX   ; No task switch, only descriptor loading
278     ; See 図 9-6.
279     ; load minimal set of registers necessary to simulate task
280     ; switch
281
282
283         MOV     AX,[EDX].SS_reg     ; start loading registers
284         MOV     EDI,[EDX].ESP_reg
285         MOV     SS,AX
286         MOV     ESP,EDI             ; stack now valid
287         PUSH    DWORD PTR [EDX].EFLAGS_reg
288         PUSH    DWORD PTR [EDX].CS_reg
289         PUSH    DWORD PTR [EDX].EIP_reg
290         MOV     AX,[EDX].DS_reg
291         MOV     BX,[EDX].ES_reg
292         MOV     DS,AX     ; DS and ES no longer linear memory
293         MOV     ES,BX
```

```
 294
 295          ; simulate far jump to initial task
 296          IRETD
 297
 298  STARTUP_CODE  ENDS
*** WARNING #377 IN 298, (PASS 2) SEGMENT CONTAINS PRIVILEGED
INSTRUCTION(S)
 299
 300  END STARTUP, DS:STARTUP_DATA, SS:STARTUP_DATA
 301
 302

ASSEMBLY COMPLETE,   1 WARNING,   NO ERRORS.
```

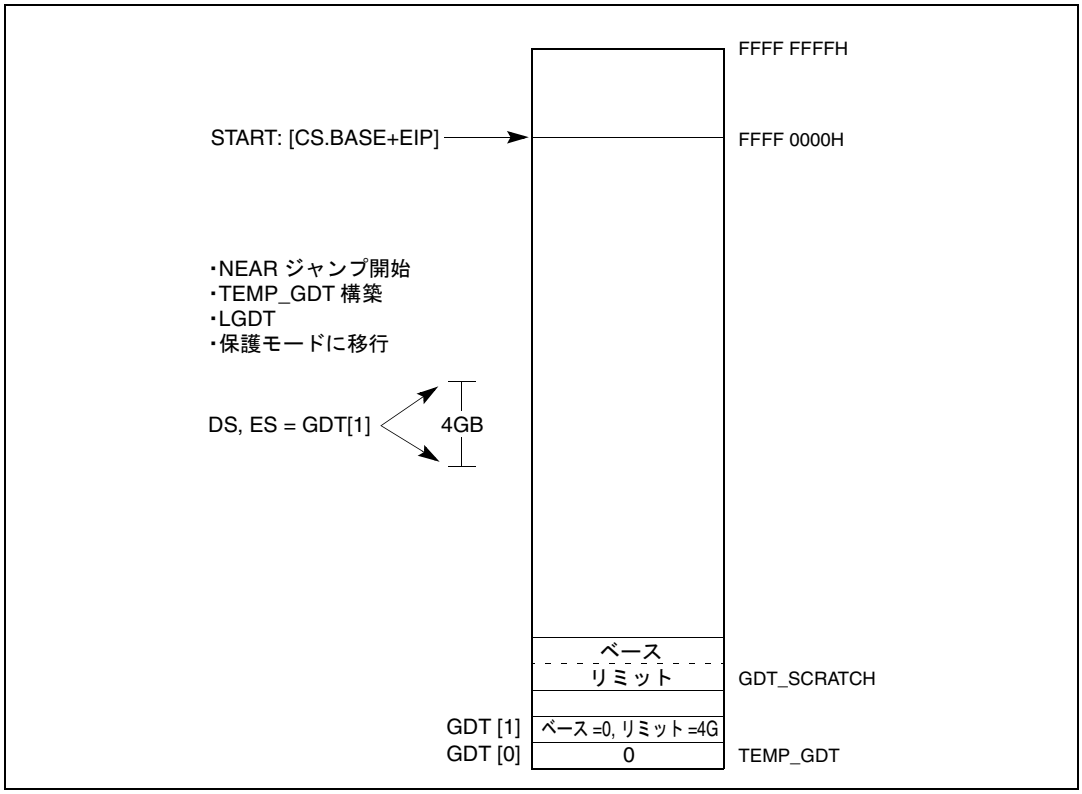


図 9-4. 一時 GDT を構築し、保護モードに切り替える（リストファイルの 162 ～ 172 行）

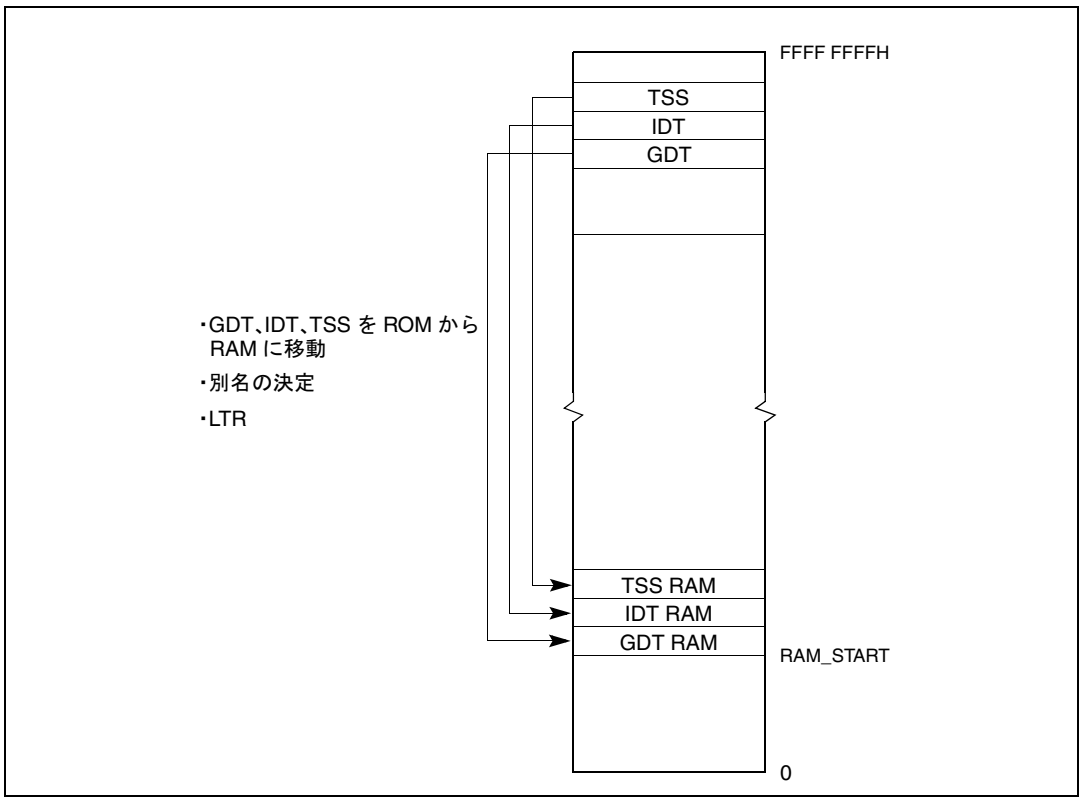


図 9-5. GDT、IDT、TSS を ROM から RAM に移動（リストファイルの 196 ～ 261 行）

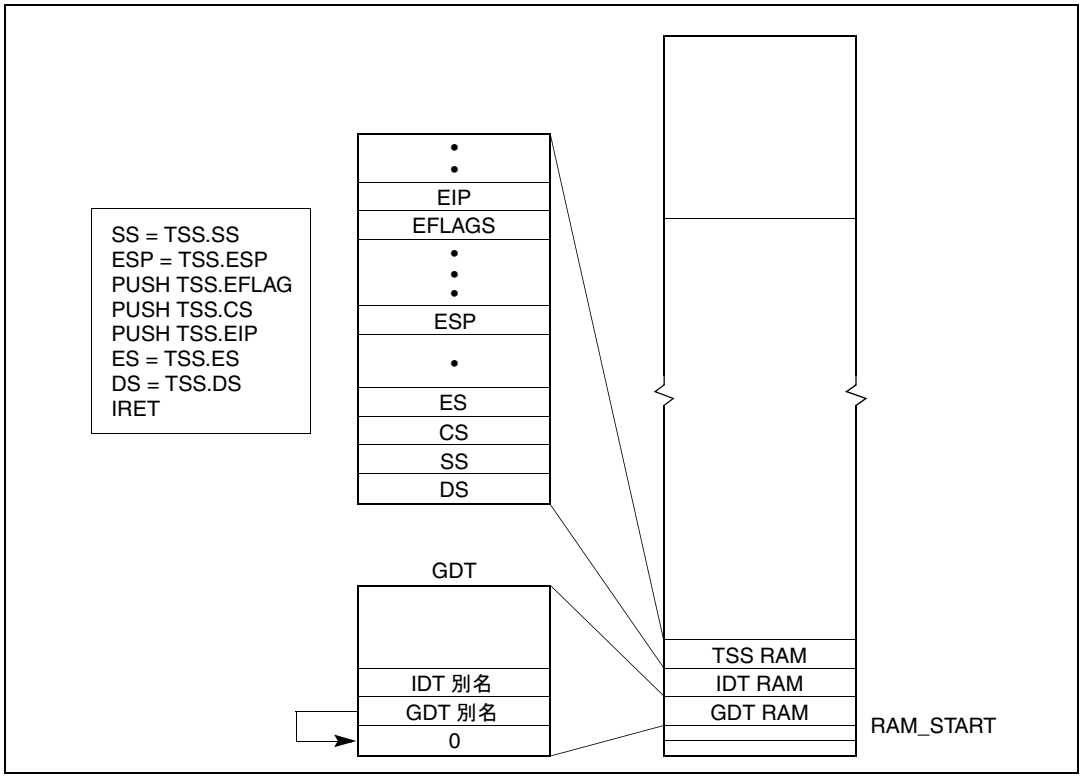


図 9-6. タスク・スイッチング（リストファイルの 282 ～ 296 行）

9.10.3. MAIN.ASM ソースコード

例 9-2. に示すファイル MAIN.ASM は、このアプリケーションのデータ・セグメントとスタック・セグメントを定義する。このファイルは、STARTUP.ASM が実行した IRET 命令によって起動されるメイン・モジュール・タスク（高級言語で記述されたタスク）と置き換えられる。

例 9-2. MAIN.ASM

```
NAME    main_module
data    SEGMENT RW
    dw 1000 dup(?)
DATA    ENDS
stack stackseg 800
CODE SEGMENT ER  use32 PUBLIC
main_start:
    nop
    nop
    nop
CODE  ENDS
END main_start, ds:data, sss:stack
```

9.10.4. サポートするファイル

例 9-3. に示すバッチファイルを使用して、STARTUP.ASM と MAIN.ASM の 2 つのソース・コード・ファイルをアセンブルし、最終的なアプリケーションを作成できる。

例 9-3. アプリケーションのアセンブルと作成に使用するバッチファイル

```
ASM386 STARTUP.ASM
ASM386 MAIN.ASM
BLD386 STARTUP.OBJ, MAIN.OBJ buildfile(EPROM.BLD)
bootstrap(STARTUP) Bootload
```

この例では、BLD386 は以下の各機能を実行する。

- セグメントとテーブルに物理メモリ位置を割り当てる。
- 作成ファイルと入力ファイルを使用してテーブルを生成する。
- オブジェクト・ファイルをリンクし、参照を解決する。
- EPROM へプログラムされるブートロード可能なファイルを生成する。

上記の各機能を実行する BLD386 への入力として使用される作成ファイルを例 9-4. に示す。

例 9-4. 作成ファイル

```
INIT_BLD_EXAMPLE;

SEGMENT
        *SEGMENTS(DPL = 0)
    ,   startup.startup_code(BASE = 0FFFF0000H)
    ;

TASK
        BOOT_TASK(OBJECT = startup, INITIAL,DPL = 0,
                                NOT INTENABLED)
,       PROTECTED_MODE_TASK(OBJECT = main_module,DPL = 0,
                                NOT INTENABLED)
    ;

TABLE
    GDT (
        LOCATION = GDT_EPROM
    ,   ENTRY = (
            10:   PROTECTED_MODE_TASK
    ,               startup.startup_code
    ,       startup.startup_data
    ,       main_module.data
    ,       main_module.code
    ,       main_module.stack
```

```
                )
        ),

    IDT (
        LOCATION = IDT_EPROM
        );

MEMORY
    (
        RESERVE = (0..3FFFH
                    -- Area for the GDT, IDT, TSS copied from ROM
    ,                60000H..0FFFEFFFFH)
    ,   RANGE = (ROM_AREA = ROM (0FFFF0000H..0FFFFFFFFH))
                    -- Eprom size 64K
    ,   RANGE = (RAM_AREA = RAM (4000H..05FFFFH))
    );

END
```

表 9-5. に、それぞれの作成項目と ASM ソースファイルの関係を示す。

表 9-5. BLD 項目と ASM ソースファイルの関係

項目	ASM386 と Startup.A58	BLD386 Controls と BLD ファイル	影響
ブートストラップ	public startup startup:	bootstrap start(startup)	0FFFFFFF0H から NEAR ジャンプしてスタート。
GDT 位置	public GDT_EPROM GDT_EPROM TABLE_REG <>	TABLE GDT(location = GDT_EPROM)	GDT の位置が GDT_EPROM の位置にプログラムされる。
IDT 位置	public IDT_EPROM IDT_EPROM TABLE_REG <>	TABLE IDT(location = IDT_EPROM	GDT の位置が GDT_EPROM の位置にプログラムされる。
RAM スタート	RAM_START equ 400H	memory (reserve = (0..3FFFH))	テーブルを移動するための RAM デスティネーションとして、RAM_START が使用される。これは、アプリケーションのセグメント領域からは除外しなければならない。
アプリケーション TSS の GDT 内の位置	TSS_INDEX EQU 10	TABLE GDT(ENTRY=(10: PROTECTED_MODE_ TASK))	アプリケーション TSS の記述を GDT エントリ 10 に入れる
EPROM のサイズと位置	初期化コードのサイズと位置	SEGMENT startup.code (base= 0FFFF0000H) ...memory (RANGE (ROM_AREA = ROM(x..y)))	初期化コードのサイズは 64K 未満でなければならない。また 4G バイト・メモリ空間の最上位の 64K になければならない。

9.11. マイクロコード・アップデート機能

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサは、インテルが提供するデータブロックをプロセッサにロードすることによって、誤りを修正する機能を持っている。このデータブロックは、マイクロコード・アップデートと呼ばれている。システムの初期化時にこの機能を利用するためには、BIOSが機構を用意する必要がある。この節では、この機能について説明する。また、今後のアップデートをシステムBIOSに組み込むための仕様についても説明する。

インテルでは、プロセッサのリビジョンに対応したマイクロコード・アップデートのリリースはプロセッサのステッピングと同等と考えていて、マイクロコード・アップデートのリリースに対してフルステッピング・レベルの検証を行っている。

マイクロコード・アップデートを使用して、プロセッサのエラッタを修正できる。アップデート・ローダを持つBIOSが、システムの初期化時にプロセッサ上にアップデートをロードする役割を受け持つ（図9-7.を参照）。このプロセスには、2つの段階がある。第1の段階では、必要なアップデート・データ・ブロックをBIOSに組み込む。第2の段階では、アップデート・データ・ブロックをプロセッサにロードする。

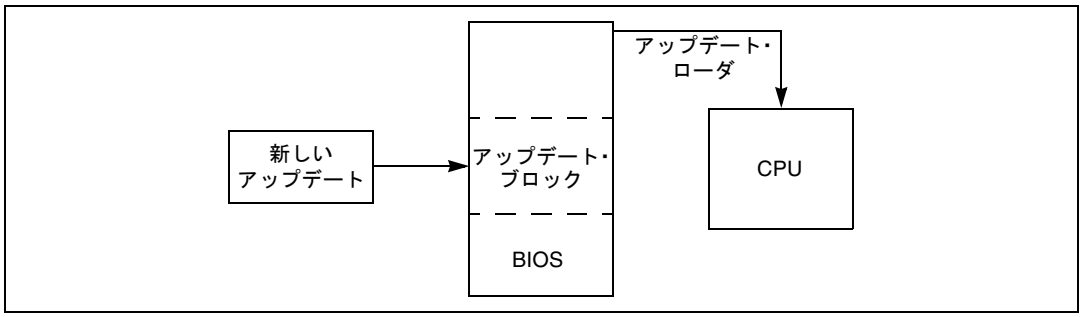


図 9-7. マイクロコード・アップデートの適用

9.11.1. マイクロコード・アップデート

マイクロコード・アップデートは、インテルが提供する、記述的なヘッダとデータを含むバイナリコードである。アップデートには実行可能コードは含まれていない。各マイクロコード・アップデートは、特定のプロセッサ・シグネチャのリストに合わせて開発されている。プロセッサのシグネチャとアップデート内のシグネチャが一致しない場合は、ロードは失敗する。プロセッサ・シグネチャには、プロセッサの拡張ファミリ、拡張モデル、タイプ、ファミリ、モデル、ステッピングが含まれている（プロセッサ・ファミリ0fH、モデル03Hからは、特定のマイクロコード・アップデートが複数のプロセッサ・シグネチャのうち1つに関連付けられるときがある。詳細は、9.11.2.項を参照）。

マイクロコード・アップデートは、マルチバイト・ヘッダ、暗号化されたデータ、オプションの拡張シグネチャ・テーブルで構成される。表9-6.は、各フィールドの定義を示している。表9-7.は、アップデートのフォーマットを示している。

ヘッダは48バイトである。ヘッダの最初の4バイトは、ヘッダ・バージョンを格納する。ソフトウェアは、ヘッダ・バージョンに基づいて、アップデートのヘッダと予約済みフィールドを解釈する。エンコード方式によって改ざんを防止し、特定のアップデートの認証手段を提供する。マイクロコード・アップデートのデータ・サイズ・フィールドの値が00000000Hである場合は、マイクロコード・アップデートのサイズは2048バイトである。最初の48バイトは、マイクロコード・アップデート・ヘッダを格納する。それ以外の2000バイトは、暗号化されたデータを格納する。

マイクロコード・アップデートのデータ・サイズ・フィールドの値が00000000Hでない場合は、合計サイズのフィールドがマイクロコード・アップデートのサイズを指定する。アップデートの最初の48バイトは、マイクロコード・アップデート・ヘッダを格納する。マイクロコード・アップデートの2番目の部分は、暗号化されたデータである。マイクロコード・アップデート・ヘッダのデータ・サイズ・フィールドは、暗号化されたデータのサイズを指定する。この値は、DWORDのサイズの倍数でなければならない。オプションの拡張シグネチャ・テーブルは（存在する場合）、暗号化されたデータの後に続く。このテーブルのサイズは、（Total Size – (Data Size + 48)）によって計算される。

注記

オプションの拡張シグネチャ・テーブルは、プロセッサ・ファミリ 0FH、モデル03Hからサポートされる。

表 9-6. マイクロコード・アップデートのエンコーディング・フォーマット

フィールド名	オフセット（バイト単位）	データ長（バイト単位）	説明
Header Version	0	4	アップデート・ヘッダのバージョン番号。
Update Revision	4	4	アップデートの固有のバージョン番号。プロセッサが提供する、プロセッサ内の現在のアップデートの機能を示すアップデート・シグネチャの基礎になる。BIOS は、このフィールドの値を使用してアップデートを認証し、プロセッサが正常にロードしたことを確認する。このフィールドの値を、プロセッサのステッピングの識別だけに使用することはできない。これは、符号付き 32 ビット数である。
Date	8	4	アップデートの作成日（バイナリ・フォーマット）: mmddyyyy（例えば、07/18/98 は、07181998H になる）。
Processor Signature	12	4	このアップデート・リビジョンを必要とする、プロセッサの拡張ファミリ、拡張モデル、タイプ、ファミリ、モデル、ステッピング（例えば、00000650H）。各マイクロコード・アップデートは、プロセッサの特定のタイプ、ファミリ、モデル、ステッピング向けに設計されている。BIOS は、プロセッサ・シグニチャ・フィールドの値と CPUID 命令の戻り値を照合して、アップデートとロード先のプロセッサが適合するかどうかを判断する。このフィールドにコード化される情報は、CPUID 命令によって返されるビット表現に正確に対応している。
Checksum	16	4	アップデート・データとヘッダのチェックサム。これを使用して、アップデート・ヘッダとデータの状態を確認する。マイクロコード・アップデートを含む DWORD の合計が 00000000H になれば、チェックサムは正常である。
Loader Revision	20	4	このアップデートを正しくロードするのに必要なローダ・プログラムのバージョン番号。最初のバージョンは 00000001H である。
Processor Flags	24	4	プラットフォーム・タイプの情報は、この 4 バイト・フィールドの最下位 8 ビットにコード化される。各ビットが、CPUID の特定のプラットフォーム・タイプを表す。BIOS は、プロセッサ・フラグ・フィールドと MSR（17H）のプラットフォーム ID ビットを照合して、アップデートとロード先のプロセッサが適合するかどうかを判断する。複数のプラットフォーム ID のサポートを表す複数のビットがセットされることがある。
Data Size	28	4	暗号化されたデータのサイズをバイト単位で指定する。この値は、DWORD の倍数でなければならない。この値が 00000000H である場合は、マイクロコード・アップデートの暗号化されたデータは 2000 バイト（すなわち、500DWORD）である。
Total Size	32	4	マイクロコード・アップデートの合計サイズをバイト単位で指定する。この値は、ヘッダのサイズ、暗号化されたデータのサイズ、オプションの拡張シグネチャ・テーブルのサイズを合計した値になる。

表 9-6. マイクロコード・アップデートのエンコーディング・フォーマット（続き）

フィールド名	オフセット（バイト単位）	データ長（バイト単位）	説明
Reserved	36	12	今後の拡張のために予約されたフィールド。
Update Data	48	データサイズまたは 2000	アップデート・データ。
Extended Signature Count	Data Size + 48	4	このマイクロコード・アップデート内に存在する、拡張シグネチャ構造の数（プロセッサ・シグネチャ [n]、プロセッサ・フラグ [n]、チェックサム [n]）を指定する。
Extended Checksum	Data Size + 52	4	アップデートの拡張プロセッサ・シグネチャ・テーブルのチェックサム。拡張プロセッサ・シグネチャ・テーブルの保全性の検証に使用される。拡張プロセッサ・シグネチャ・テーブルを構成する DWORD を合計した結果が 00000000H になれば、チェックサムは正常である。
Reserved	Data Size + 56	12	予約済みのフィールド
Processor Signature[n]	Data Size + 68 + (n * 12)	4	このアップデート・リビジョン（例えば、00000650H）を要求するプロセッサの拡張ファミリ、拡張モデル、タイプ、ファミリ、モデル、ステッピング。各マイクロコード・アップデートは、特定の拡張ファミリ、拡張モデル、タイプ、ファミリ、モデル、ステッピングのプロセッサ専用に設計されている。BIOS は、プロセッサ・シグネチャ・フィールドと CPUID 命令を組み合わせて使用し、アップデートをプロセッサにロードしてよいかを判断する。このフィールド内にコード化される情報は、CPUID 命令によって返されるビット表現に完全に対応する。
Processor Flags[n]	Data Size + 72 + (n * 12)	4	プラットフォーム・タイプの情報は、この 4 バイト・フィールドの下位 8 ビットにコード化される。各ビットは、特定の CPUID 向けの特定のプラットフォーム・タイプを表す。BIOS は、プロセッサ・フラグ・フィールドと MSR (17H) 内のプロセッサ ID ビットを組み合わせて使用し、アップデートをプロセッサにロードしてよいかを判断する。複数のビットをセットして、複数のプラットフォーム ID のサポートを表現できる。
Checksum[n]	Data Size + 76 + (n * 12)	4	ユーティリティ・ソフトウェアが 1 つのマイクロコード・アップデートを複数のマイクロコード・アップデートに分割するのに使用する。それぞれの新しいマイクロコード・アップデートは、オプションの拡張プロセッサ・シグネチャ・テーブルなしで作成される。

表 9-7. マイクロコード・アップデートのフォーマット

31 24 16 8 0	バイト
ヘッダ・バージョン	0
アップデート・リビジョン	4
月 : 8 / 日 : 8 / 年 : 16	8
プロセッサ・シグネチャ（CPUID）	12
予約済み : 4 / 拡張ファミリ : 8 / 拡張モード : 4 / 予約済み : 2 / タイプ : 2 / ファミリ : 4 / モデル : 4 / ステッピング : 4	
チェックサム	16
ローダ・リビジョン	20
プロセッサ・フラグ	24
予約済み（24 ビット） / P7 P6 P5 P4 P3 P2 P1 P0	
データサイズ	28
合計サイズ	32
予約済み（12 バイト）	36
アップデート・データ（Data Size バイト、または Data Size=00000000H の場合は 2000 バイト）	48
拡張シグネチャ数 ‘n’	データサイズ + 48
拡張プロセッサ・シグネチャ・テーブル・チェックサム	データサイズ + 52
予約済み（12 バイト）	データサイズ + 56
プロセッサ・シグネチャ [n]	データサイズ + 68 + （n * 12）
プロセッサ・フラグ [n]	データサイズ + 72 + （n * 12）
チェックサム [n]	データサイズ + 76 + （n * 12）

9.11.2. オプションの拡張シグネチャ・テーブル

拡張シグネチャ・テーブルは、暗号化されたデータがただ1つのプロセッサ・シグネチャをサポートしている場合は、暗号化されたデータの最後に付加できる構造である（オプションの場合）。暗号化されたデータが複数のプロセッサ・ステッピングまたはモデルをサポートしている場合は、拡張シグネチャ・テーブルは常に存在する（必須の場合）。

拡張シグネチャ・テーブルは、20バイトの拡張シグネチャ・ヘッダ構造（拡張シグネチャ数を格納する）、拡張プロセッサ・シグネチャ・テーブル・チェックサム、予約済みの12バイトで構成される（図9-8.）。拡張シグネチャ・テーブルは、拡張シグネチャ・ヘッダ構造の後に続けて、0 ～ n 個の拡張プロセッサ・シグネチャ構造を格納する。

各プロセッサ・シグネチャ構造は、プロセッサ・シグネチャ、プロセッサ・フラグ、チェックサムで構成される（表9-9.）。

拡張シグネチャ・ヘッダ構造内の拡張シグネチャ数は、拡張シグネチャ・テーブル内に存在するプロセッサ・シグネチャ構造の数を示す。

拡張プロセッサ・シグネチャ・テーブル・チェックサムは、拡張シグネチャ・テーブルを構成するすべてのDWORDのチェックサムである。これには、拡張シグネチャ数、拡張プロセッサ・シグネチャ・テーブル・チェックサム、予約済みの12バイト、n個のプロセッサ・シグネチャ構造が含まれる。DWORDチェックサムの結果が00000000Hになる場合は、有効な拡張シグネチャ・テーブルが存在する。

表 9-8. 拡張プロセッサ・シグネチャ・テーブル・ヘッダ構造

拡張シグネチャ数 ‘n’	データサイズ + 48
拡張プロセッサ・シグネチャ・テーブル・チェックサム	データサイズ + 52
予約済み（12 バイト）	データサイズ + 56

表 9-9. プロセッサ・シグネチャ構造

プロセッサ・シグネチャ [n]	データサイズ + 68 +（n * 12）
プロセッサ・フラグ [n]	データサイズ + 72 +（n * 12）
チェックサム [n]	データサイズ + 76 +（n * 12）

9.11.3. プロセッサの識別

各マイクロコード・アップデートは、特定の 1 種類のプロセッサまたは一連のプロセッサ向けに設計されている。ロードする適切なマイクロコード・アップデートを決めるために、ソフトウェアは、マイクロコード・アップデートに埋め込まれたプロセッサ・シグネチャのうち 1 つが、ターゲット・プロセッサが EAX=1 で実行した CPUID 命令で返される 32 ビット・プロセッサ・シグネチャに一致することを保証しなければならない。マイクロコード・アップデートに埋め込まれたプロセッサ・シグネチャと CPUID 命令で返されるプロセッサ・シグネチャが一致しない場合、そのマイクロコード・アップデートをロードしようとすると、プロセッサはそのアップデートを拒否する。

例 9-5. は、プロセッサとマイクロコード・アップデートの間でプロセッサ・シグネチャが一致するかをチェックする方法を示している。

例 9-5. プロセッサ・シグネチャを検証する疑似コード

```
ProcessorSignature ← CPUID(1):EAX

If (Update.HeaderVersion == 00000001h)
{
       // first check the ProcessorSignature field
    If (ProcessorSignature == Update.ProcessorSignature)
       Success

       // if extended signature is present
    Else If (Update.TotalSize > (Update.DataSize + 48))
    {

       //
       // Assume the Data Size has been used to calculate the
       // location of Update.ProcessorSignature[0].
       //

       For (N ← 0; ((N < Update.ExtendedSignatureCount) AND
          (ProcessorSignature != Update.ProcessorSignature[N]));
N++);

          // if the loops ended when the iteration count is
          // less than the number of processor signatures in
          // the table, we have a match
       If (N < Update.ExtendedSignatureCount)
          Success
       Else
          Fail
    }
    Else
       Fail
Else
    Fail
```

9.11.4. プラットフォームの識別

適切なマイクロコード・アップデートを選択するには、プロセッサ・シグネチャの検証以外に、使用するプロセッサのプラットフォームのタイプを確認する必要がある。使用するプロセッサのプラットフォームのタイプを確認するには、IA32_PLATFORM_ID レジスタ（MSR 17H）を読み取る必要がある。この 64 ビット・レジスタは、RDMSR 命令を使用して読み取られる。

3 つのプラットフォーム ID ビットを 2 進化 10 進数（BCD）として読み取ると、インストールされたプロセッサに対応する、マイクロコード・アップデート・ヘッダのプロセッサ・フラグ・フィールド内のビット位置が示される。48 バイト・ヘッダ内のプロセッサ・フラグと、拡張プロセッサ・シグネチャ構造に関連するプロセッサ・フラグ・フィールドは、複数のビットをセットできる。セットされた各ビットは、アップデートがサポートする異なるプラットフォーム ID を表す。

レジスタ名： **IA32_PLATFORM_ID**

MSR アドレス： 017H
アクセス： **読み取り専用**

IA32_PLATFORM_ID は、RDMSR 命令によって Qword として参照された場合にのみアクセスされる 64 ビット・レジスタである。

表 9-10. プロセッサ・フラグ

ビット	説明
63:53	予約済み
52:50	プラットフォーム ID ビット（RO）。このフィールドは、プロセッサに対応するプラットフォームに関する情報を示す。表 9-7. も参照。 52 51 50 0 0 0 プロセッサ・フラグ 0 0 0 1 プロセッサ・フラグ 1 0 1 0 プロセッサ・フラグ 2 0 1 1 プロセッサ・フラグ 3 1 0 0 プロセッサ・フラグ 4 1 0 1 プロセッサ・フラグ 5 1 1 0 プロセッサ・フラグ 6 1 1 1 プロセッサ・フラグ 7
49:0	予約済み

ソフトウェアは、例 9-6. のアルゴリズムと同様のアルゴリズムを使用して、プラットフォーム情報を検証できる。

例 9-6. プロセッサ・フラグ・テストの疑似コードの例

```
Flag ← 1 << IA32_PLATFORM_ID[52:50]

If (Update.HeaderVersion == 00000001h)
{
     If (Update.ProcessorFlags & Flag)
     {
        Load Update
     }
     Else
     {

        //
        // Assume the Data Size has been used to calculate the
        // location of Update.ProcessorSignature[N] and a match
        // on Update.ProcessorSignature[N] has already succeeded
        //

        If (Update.ProcessorFlags[n] & Flag)
        {
           Load Update
        }
     }
}
```

9.11.5. マイクロコード・アップデートのチェックサム

各マイクロコード・アップデートのアップデート・ヘッダには、1 つの DWORD チェックサムが含まれている。マイクロコード・アップデートが壊れていないことを保証するのは、ソフトウェアの役割である。ソフトウェアは、マイクロコード・アップデートが壊れていないかをチェックするために、マイクロコード・アップデートの符号なし DWORD（32 ビット）チェックサムを実行しなければならない。いくつかのフィールドが符号付きの場合でも、チェックサム・プロシージャは、すべての DWORD を符号なしとして扱う。マイクロコード・アップデートのヘッダ・バージョンが 00000001H である場合は、マイクロコード・アップデートを構成するすべての DWORD を合計する必要がある。その結果、00000000H の値が得られれば、チェックサム・チェックは正常である。それ以外のすべての値は、マイクロコード・アップデートが壊れており、ロードしてはならないことを示す。

例 9-7. の疑似コードに示したチェックサム・アルゴリズムは、マイクロコード・アップデートを符号なし DWORD の配列として扱う。バイト・オフセット 32 のデータ・サイズ DWORD フィールドの値が 00000000H である場合は、暗号化されたデータのサイズは 2000 バイト（すなわち、500DWORD）になる。データ・サイズ・フィールドの値が 00000000H でない場合は、DWORD 単位のマイクロコード・アップデートのサイズは、（Total Size / 4）になる。

例 9-7. チェックサム・テストの疑似コードの例

```
N ← 512

If (Update.DataSize != 00000000H)
    N ← Update.TotalSize / 4

ChkSum ← 0
For (I ← 0; I < N; I++)
{
    ChkSum ← ChkSum + MicrocodeUpdate[I]
}

If (ChkSum == 00000000H)
    Success
Else
    Fail
```

9.11.6. マイクロコード・アップデート・ローダ

この項では、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサへのアップデートのロードに使用される、アップデート・ローダについて説明する。また、正常にロードするための BIOS の必要条件についても説明する。これから説明するアップデート・ローダには、アップデートのロードに必要な最小限の命令が含まれている。アップデートのロードに必要な特定の命令シーケンスは、アップデート・ヘッダ内のローダ・リビジョン・フィールドの値によって異なる。リビジョンは、通常は新しいプロセッサ・モデルが開発されたときにのみ変更されるため、次に変更されるまでの間隔は長い。

以下の例 9-8. は、ローダ・リビジョン 00000001H のアップデート・ローダを示している。マイクロコード・アップデートは、16 バイト境界にアライメントが合っていなければならないことに注意する。

例 9-8. 簡単なマイクロコード・アップデート・ローダのアセンブリ・コード例

```
mov ecx,79h        ; MSR to read in ECX
xor  eax,eax       ; clear EAX
xor  ebx,ebx       ; clear EBX
mov  ax,cs         ; Segment of microcode update
shl  eax,4
mov  bx,offset Update; Offset of microcode update
add  eax,ebx       ; Linear Address of Update in EAX
add  eax,48d       ; Offset of the Update Data within the Update
xor  edx,edx       ; Zero in EDX
WRMSR              ; microcode update trigger
```

例 9-8. で示したローダでは、update は、BIOS のコード・セグメントに埋め込まれたマイクロコード・アップデート（ヘッダとデータ）のアドレスである。また、16 バイト境界にアライメントを合わせて、プロセッサはリアルモードで動作しているものとする。アップデートのデータは、プロセッサが現在の動作モード（リアルモードまたは保護モード）でアクセスできるメモリ内のどこにあってもかまわない。

BIOS がマイクロコード・アップデート・トリガ（WRMSR）命令を実行するには、以下の条件を満たしていなければならない。

- EAX にアップデート・データの始点のリニアアドレスが入っている。
- EDX にゼロが入っている。
- ECX に 79H が入っている（IA32_BIOS_UPDT_TRIG のアドレス）。

その他の必要条件には、以下のものがある。

- マイクロコード・アップデートは、BIOS POST の段階で、また必ずプロセッサの L2 キャッシュ・コントローラの初期化より前に、プロセッサにロードしなければならない。
- プロセッサがリアルモードで動作しているときにアップデートをロードする場合は、アップデート・データがセグメント範囲を超えることはできない。
- ページングがイネーブルになっている場合は、現在存在するページにアップデート・データをマッピングしなければならない。
- マイクロコード・アップデート・データは、16 バイト境界にアライメントが合っている必要がある。

9.11.6.1. アップデートのロードに対するハードリセットの影響

ハードリセット時には、ロードされたアップデートの影響はプロセッサからクリアされる。したがって、BIOS POST 中にハードリセットがアサートされるたびに、リセットされたすべてのプロセッサ上にアップデートを再ロードしなければならない。ただし、プロセッサ INIT の前後で、ロードされたアップデートの影響は保持される。アップデートをプロセッサに複数回ロードしても、特に問題はない。

9.11.6.2. マルチプロセッサ・システム内のアップデート

マルチプロセッサ（MP）システムでは、それぞれのプロセッサに、そのプロセッサの CPUID とプラットフォーム ID ビットに適合するアップデート・データをロードする必要がある。BIOS は、この必要条件が満たされているかどうか、またシステム内のすべてのプロセッサが実行するモジュール内にローダがあるかどうかを確認する役割を受け持つ。異なるステッピングのインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサが共存できるシステム設計の場合は、

BIOS は、個々のプロセッサとアップデート・ヘッダ情報の適合性を確認して、アップデートが正常にロードされるように保証しなければならない。これらの条件を考えると、マルチプロセッサの初期化時にアップデートをロードするのが最も現実的である。

9.11.6.3. HT テクノロジ対応システム内のアップデート

ハイパー・スレッディング・テクノロジ（HTテクノロジ）は、マイクロコード・アップデートのロードに影響を与える。アップデートは、物理プロセッサ内の各コアにロードしなければならない。したがって、HT テクノロジ対応プロセッサでは、マイクロコード・アップデートをロードする論理プロセッサは、コア当たり 1 つだけである。個々の論理プロセッサは、互いに独立してアップデートをロードできる。ただし、MP 初期化機能は、何らかの機構（例えば、ソフトウェア・セマフォ）によって、マイクロコード・アップデートのロードをシリアル化し、同じコアに対して同時に複数のロードが行われることを防ぐ必要がある。

9.11.6.4. アップデート・ローダの拡張

9.11.6. 項「マイクロコード・アップデート・ローダ」で示したアップデート・ローダは、最小限のコードである。このコードを拡張して、さらに機能を追加できる。次のような拡張が可能である。

- BIOS に複数のアップデートを組み込んで、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサの複数のステッピングをサポートできる。この機能は、マルチプロセッサ・システム上の混合ステッピング環境での動作に対応している。これによって、ユーザは新しいバージョンのプロセッサにアップグレードできる。この場合は、ローダ・プログラムを修正して、特定のアップデートをロードする前に、ローダが実行されるプロセッサの CPUID とプラットフォーム ID ビットが、アップデート・ヘッダと適合するかどうかをチェックさせる。アップデートの数の制限は、BIOS の利用可能な容量だけである。
- アップデートのロード後、ローダにプロセッサをテストさせ、アップデートが正常にロードされたかどうかを確認できる。9.11.7. 項「アップデートのシグネチャと検証」を参照のこと。
- ローダにチェックサムを実行させて、アップデートのダブルワードの合計がゼロになるかどうかを確認させると、アップデート・データの状態を確認できる。9.11.5. 項「マイクロコード・アップデートのチェックサム」を参照のこと。
- 電源投入時に、アップデートが正常にロードされたことを示すメッセージを表示できる。

9.11.7. アップデートのシグネチャと検証

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサは、特定のアップデートを認証し、現在のアップデート・リビジョンを識別する機能を持っている。この節では、この機能をサポートするプロセッサのモデル固有の拡張機能について説明する。以下に説明するアップデートの検証方法では、BIOS が、プロセッサに現在ロードされているリビジョンより新しいアップデートだけを検証するものとする。

CPUID 命令は、通常のレジスタ戻り値以外に、モデル固有レジスタに 1 つの値を返す。CPUID 命令のセマンティクスにより、この命令は、アドレス 08BH の 64 ビット・モデル固有レジスタ内にアップデート ID 値（IA32_BIOS_SIGN_ID）を返す。プロセッサ上にアップデートが存在しない場合は、この MSR の値は変更されずに残る。BIOS は、CPUID 命令を実行する前に MSR にゼロを前もってロードしておかなければならない。CPUID 命令の実行後も 8BH での MSR の読まれた値がゼロの場合は、アップデートが存在しないことを示す。

RDMSR 命令の実行後に EDX レジスタに返されるアップデート ID 値は、プロセッサにロードされたアップデートのリビジョンを示す。この値は、EAX レジスタに返される CPUID 値と合わせて、特定のアップデートを個々に識別する。このシグネチャ ID と、マイクロコード・アップデート・ヘッダ内のアップデート・リビジョン・フィールドを直接照合して、正しくロードされたかどうかを確認できる。プロセッサの特定のステッピング向けにリリースされた連続するアップデートは、必ずそれぞれ異なるシグネチャを持つ。CPUID 命令によって返されるプロセッサ・シグネチャは、異なるステッピング向けのアップデートごとに異なる。

9.11.7.1. シグネチャの確認

プロセッサに正常にロードされたアップデートは、現在機能しているリビジョンのアップデート・リビジョンと一致するシグネチャを持つ。実際のアップデートがロードされた後は、このシグネチャはいつでも利用できる。このシグネチャを要求しても、現在ロードされているアップデートには影響を与えない。このシグネチャを要求しても、ロードされているアップデートには影響を与えない。

このシグネチャを確認する手続きは、例 9-9. に示すとおりである。

例 9-9. アップデート・リビジョンを取り出すためのアセンブリ・コード

```
MOV    ECX, 08BH  ;IA32_BIOS_SIGN_ID
XOR    EAX, EAX   ;clear EAX
XOR    EDX, EDX   ;clear EDX
WRMSR  ;Load 0 to MSR at 8BH
MOV    EAX, 1
cpuid
MOV    ECX, 08BH  ;IA32_BIOS_SIGN_ID
rdmsr  ;Read Model Specific Register
```

プロセッサ上にアクティブになっているアップデートがある場合は、RDMSR 命令の実行後、そのリビジョンが EDX レジスタに返される。

IA32_BIOS_SIGN_ID　マイクロコード・アップデート・シグネチャ・レジスタ

MSR アドレス :　Qword としてアクセスされる 08BH
デフォルト値 :　XXXX XXXX XXXX XXXXh
アクセス :　読み取り / 書き込み

IA32_BIOS_SIGN_ID レジスタは、CPUID 命令の実行時に、マイクロコード・アップデート・シグネチャを報告するのに使用される。シグネチャは、上位の DWORD に返される（表 9-11.）。

表 9-11. マイクロコード・アップデート・シグネチャ

ビット	説明
63:32	マイクロコード・アップデート・シグネチャ。このフィールドは、CPUID 命令（機能 1）の実行に続いて読み取られた場合、現在ロードされているマイクロコード・アップデートのシグネチャを示す。CPUID（機能 1）を実行する前に、このレジスタ・フィールドにはあらかじめゼロがロードされていなければならない。CPUID の実行後もこのフィールドの値がゼロのままになる場合は、マイクロコード・アップデートはロードされていない。ゼロでない値はシグネチャである。
31:0	予約済み

9.11.7.2. アップデートの認証

BIOS は、例 9-10. のアルゴリズムによって、上記のシグネチャ・プリミティブを使ってアップデートを認証できる。

例 9-10. アップデートを認証するための疑似コード

```
Z ← Obtain Update Revision from the Update Header to be
authenticated;
X ← Obtain Current Update Signature from MSR 8BH;

If (Z > X)
{
     Load Update that is to be authenticated;
     Y ← Obtain New Signature from MSR 8BH;

     If (Z == Y)
        Success
     Else
        Fail
}
Else
     Fail
```

例 9-10. は、BIOS が、現在ロードされているリビジョンよりリビジョン番号の大きいアップデートだけを認証することを想定している（すなわち、Current Signature (X) < New Update Revision (Z) の場合）。アップデートがロードされていないプロセッサは、リビジョン 0 のアップデートを持つと考える必要がある。

この認証手続きは、プロセッサが提供するデコーディングに依存して、危険性のあるソースからのアップデートを検証する。例えば、この機構と他の保護手段を組み合わせれば、フィールド・アップデートを動的に BIOS に組み込むためのセキュリティが得られる。

9.11.8. インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのマイクロコード・アップデートの仕様

この節で説明するインターフェイスを使用して、アプリケーションは、プロセッサ固有のアップデートをシステム BIOS に動的に組み込める。ここでは、アプリケーションをコール元プログラムまたはコール元と呼ぶ。

この節で説明するリアルモードの INT 15 コール仕様は、メーカの BIOS に対するインテルの拡張である。この拡張によって、アプリケーションは、NVRAM 内のマイクロコード・アップデート・データの内容の読み取りと修正を実行できる。ただし、アップデート・ローダは、システム BIOS の一部であるため、このインターフェイスでは更新できない。この仕様に適合するシステムは、この仕様で定義されたすべての

関数を実装していなければならない。INT 15 の関数は、リアルモードからアクセスしなければならない。

9.11.8.1. BIOS の役割

BIOS が存在テスト（INT 15H, AX=0D042H, BL=0H）をパスした場合、その BIOS は、INT 15H, AX==0D042H 仕様に定義されたすべてのサブ関数を実装しているはずである。オプションの関数は存在しない。BIOS は、システムの初期化時に、各プロセッサ用の適切なアップデートをロードしなければならない。

アップデート・ブロックのヘッダ・バージョンの値が 0FFFFFFFFH である場合は、そのアップデート・ブロックは未使用であり、新しいアップデートのストアに使用できる。

BIOS は、システム内の各プロセッサ・ステッピング用として、不揮発性記憶領域（NVRAM）を確保する役割を受け持つ。この記憶単位は、1 つ以上のアップデート・ブロックからなる。アップデート・ブロックは、隣接する 2048 バイトのメモリブロックである。シングル・プロセッサ・システムの BIOS は、1 つのマイクロコード・アップデートのストア用に、アップデート・ブロックを用意すればよい。マルチプロセッサ・システムの BIOS が複合プロセッサ・ステッピング環境をサポートする場合、BIOS は、メーカのシステムボード上のプロセッサ・ソケットごとにそれぞれ独自のマイクロコード・アップデートをストアするのに十分な数のアップデート・ブロックを用意する必要がある。

BIOS は、NVRAM アップデート・ブロックを管理する役割を受け持つ。この管理タスクには、ガーベジ・コレクションが含まれる。例えば、NVRAM 内のマイクロコード・アップデートに対応するプロセッサがシステム内に存在しない場合、BIOS はそのマイクロコード・アップデートを削除する。INT 15 仕様は、セキュリティの保証、エントリの重複の防止、古いエントリのロードの防止などの機構を提供するだけである。実際のアップデート・ブロック管理はプロセッサ固有であり、BIOS ごとに異なる。

例えば、BIOS は、使用可能な最初のブロックを始点としてソートされた CPU シグネチャにしたがって、複数のアップデート・ブロックを順番に昇順で使用できる。また、ガーベジ・コレクションをセットアップ・オプションとして実装し、すべての NVRAM スロットをクリアすることも、BIOS コードとして実装し、ブート時に未使用のエントリを探して削除させることもできる。

注記

ファミリ 0FH およびモデル 03H からの IA-32 プロセッサでは、マイクロコード・アップデートのサイズは最大 16K バイトに達する。したがって、BIOS は、各マイクロコード・アップデートにつき 8 つのアップデート・ブロックを割り当てなければならない。MP システムでは、システム内の各ソケット用に、1 つの共通のマイクロコード・アップデートで十分な場合がある。

ファミリ 0FH およびモデル 03H より以前の IA-32 プロセッサでは、マイクロコード・アップデートのサイズは 2K バイトである。複数のステッピングをサポートする MP 対応 BIOS は、システム内の各ソケットにつき 1 つのブロックを割り当てなければならない。

可変サイズのマイクロコード・アップデートと固定サイズのマイクロコード・アップデートをサポートするシングルプロセッサ BIOS は、1 つの 16K バイト領域と、少なくとも 2K バイトの第 2 の領域を割り当てなければならない。

以下のアルゴリズム（例 9-11.）は、BIOS の初期化時に実行される、プロセッサにアップデートをロードするための手順を記述している。このアルゴリズムは、以下の必要条件を想定している。

- NVRAM 内のすべてのアップデートのヘッダ・バージョンとローダ・バージョンが、BIOS が現在サポートしているバージョンと一致することを BIOS が保証する。
- アップデートに正しいチェックサムが含まれている。
- また、各プロセッサ・ステッピングにつき 1 つのアップデートが存在することを BIOS が保証する。
- アップデートの古いリビジョンがより新しいリビジョンを上書きできない。

これらの必要条件は、このインターフェイスのアップデート書き込み関数の実行時に、BIOS によってチェックされる。BIOS は、NVRAM 内のすべてのアップデート・ブロックをインデックス 0 から順番にスキャンしていき、ヘッダのプロセッサ・フィールドが現在のプロセッサのプロセッサ・シグネチャ（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）プラットフォーム・ビットと一致するアップデートを検出する。

例 9-11. アップデートをロードする前に必要なチェックの疑似コード

```
For each processor in the system
{
   Determine the Processor Signature via CPUID function 1;
   Determine the Platform Bits ← 1 << IA32_PLATFORM_ID[52:50];

   For (I ← UpdateBlock 0, I < NumOfBlocks; I++)
   {
```

```
      If (Update.Header_Version == 0x00000001)
      {
         If ((Update.ProcessorSignature == Processor Signature) &&
             (Update.ProcessorFlags & Platform Bits))
         {
            Load Update.UpdateData into the Processor;
            Verify update was correctly loaded into the processor
            Go on to next processor
               Break;
         }
         Else If (Update.TotalSize > (Update.DataSize + 48))
         {
            N ← 0
            While (N < Update.ExtendedSignatureCount)
            {
               If ((Update.ProcessorSignature[N] ==
                   Processor Signature) &&
                   (Update.ProcessorFlags[N] & Platform Bits))
               {
                  Load Update.UpdateData into the Processor;
                  Verify update was correctly loaded into the processor
                  Go on to next processor
                     Break;
               }
               N ← N + 1
            }
            I ← I + (Update.TotalSize / 2048)
            If ((Update.TotalSize MOD 2048) == 0)
               I ← I + 1
         }
      }
   }
}
```

注記

IA32_PLATFORM_ID のプラットフォーム ID ビットは、3 ビットの 2 進化 10 進数フィールドとしてコード化されている。マイクロコード・アップデート・ヘッダのプラットフォーム・ビットは、個々のビットとしてコード化されている。アルゴリズムは、チェックを実行する前に、2 つのフォーマットの間の変換を行わなければならない。

BIOS は、INT 15H, 0D042H 関数を実行するとき、コール元がプラットフォーム固有の必要条件について何も知らないものと見なさなければならない。NVRAM デバイスを管理するための、チップセットおよびプラットフォーム固有のすべての前提条件を管理するのは、BIOS コールの役割である。BIOS は、アップデート書き込みサブ関数を使ってアップデート・データを書き込むとき、プロセッサ固有のデータ要件（NVRAM チェックサムの更新など）を守らなければならない。また、BIOS は、アップデート

の記録用の記憶デバイスに対する書き込み操作が成功したかどうかを確認しなければならない。

9.11.8.2. コール元プログラムの役割

この節では、コール元プログラムが、INT 15 インターフェイス仕様を使用して BIOS NVRAM にマイクロコード・アップデートをロードする際に担当するタスクについて説明する。

- コール元プログラムは、完全なリアルモードのプログラムから、INT 15H, 0D042H 関数を呼び出さなければならない。また、コール元プログラムは、完全なリアルモードで動作しているシステム上で実行されていなければならない。
- コール元は、存在テスト関数（サブ関数 0）を実行し、シグネチャとその関数の戻りコードを確認しなければならない。
- コール元プログラムは、インターフェイスの定義で指定されたとおりに、BIOS に必要なスクラッチ RAM バッファと適切なスタックサイズを用意する必要がある。
- コール元プログラムは、アップデートのロードが適切であるかどうかを判断するために、BIOS 内の既存のアップデート・データを読み取らなければならない。BIOS は、新しいアップデートを古いバージョンで上書きすることを拒否する。アップデート・ヘッダには、バージョン情報とプロセッサ固有の情報が入っている。コール元プログラムは、この情報を使用して、ロードが適切であるかどうかを判断する。
- アップデートとプロセッサの対応関係は常に厳密である。BIOS は、同一の CPU に対して同時に複数のアップデートが存在することを許してはならない。また、BIOS は、システム内に存在しないプロセッサ用のアップデートをロードすることを拒否しなければならない。
- コール元アプリケーションは、アップデート書き込み関数が正常終了した後に実行される、確認関数を実装していなければならない。この関数は、書き込まれたアップデートを読み取り、書き込まれたイメージと同じイメージを BIOS が返したことを確認する。

例 9-12. は、コール元プログラムを示している。

例 9-12. INT 15 D042 コール元プログラムの疑似コード

```
//
// We must be in real mode
//
If the system is not in Real mode exit
//
// Detect the presence of Genuine Intel processor(s) that can be updated
// using(CPUID)
```

```
//
If no Intel processors exist that can be updated exit
//
// Detect the presence of the Intel microcode update extensions
//
If the BIOS fails the PresenceTestexit
//
// If the APIC is enabled, see if any other processors are out there
//
Read IA32_APICBASE
If APIC enabled
{
    Send Broadcast Message to all processors except self via APIC
    Have all processors execute CPUID and record the Processor Signature
    (i.e.,Extended Family, Extended Model, Type, Family, Model, Stepping)
    Have all processors read IA32_PLATFORM_ID[52:50] and record Platform
     Id Bits

    If current processor cannot be updated
       exit
}
//
// Determine the number of unique update blocks needed for this system
//
NumBlocks = 0
For each processor
{
    If ((this is a unique processor stepping) AND
       (we have a unique update in the database for this processor))
    {
       Checksum the update from the database;
       If Checksum fails
          exit
       NumBlocks ← NumBlocks + size of microcode update / 2048
    }
}

//
// Do we have enough update slots for all CPUs?
//
If there are more blocks required to support the unique processor
steppings than update blocks provided by the BIOS
    exit
//
// Do we need any update blocks at all?  If not, we are done
//
If (NumBlocks == 0)
    exit
//
// Record updates for processors in NVRAM.
//
For (I=0; I<NumBlocks; I++)
```

```
{
     //
     // Load each Update
     //
     Issue the WriteUpdate function

     If (STORAGE_FULL) returned
     {
        Display Error -- BIOS is not managing NVRAM appropriately
        exit
     }

     If (INVALID_REVISION) returned
     {
        Display Message: More recent update already loaded in NVRAM for
         this stepping
        continue
     }

     If any other error returned
     {
        Display Diagnostic
        exit
     }

     //
     // Verify the update was loaded correctly
     //
     Issue the ReadUpdate function

     If an error occurred
     {
        Display Diagnostic
        exit
     }
     //
     // Compare the Update read to that written
     //
     If (Update read != Update written)
     {
        Display Diagnostic
        exit
     }

     I ← I + (size of microcode update / 2048)
}
//
// Enable Update Loading, and inform user
//
Issue the Update Control function with Task = Enable.
```

9.11.8.3. マイクロコード・アップデート関数

表 9-12. は、現在のインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのマイクロコード・アップデート関数の定義を示している。

表 9-12. マイクロコード・アップデート関数

マイクロコード・アップデート関数	関数番号	説明	必須 / オプション
存在テスト	00H	BIOS がサポートしている関数に関する情報を返す。	必須
アップデート・データ書き込み	01H	アップデート・データ領域（スロット）のうち 1 つにデータを書き込む。	必須
アップデート制御	02H	アップデートのロードをグローバルに制御する。	必須
アップデート・データ読み取り	03H	アップデート・データ領域（スロット）のうち 1 つを読み取る。	必須

9.11.8.4. INT 15H ベースのインターフェイス

インテルでは、システム・フラッシュへのマイクロコード・アップデートの追加を可能にする BIOS インターフェイスを使用するように推奨している。INT 15H インターフェイスは、これを行うためにインテルが定義した手法である。

INT 15 インターフェイスを呼び出すプログラムは、読み取り関数と書き込み関数の呼び出し中に BIOS が使用する、3 つの 64KB RAM 領域を用意する役割を受け持つ。BIOS は、読み取り / 書き込み関数呼び出しの間だけ、これらの RAM スクラッチ・パッドを自由な目的で使用できる。コール元ルーチンは、RAM ブロックを指すリアル・モード・セグメントを、CX、DX、SI レジスタに置く。INT 15 インターフェイスの関数を呼び出すときは、BIOS が利用できるスタックが少なくとも 32K バイト必要である。

一般的に、各関数を実行すると、CF はクリアされ、AH にステータスが返される。一般的なリターンコードとその他の定数の定義は、9.11.8.9. 項「リターンコード」に記載されている。

OEM エラー・フィールド（AL）は、メーカがプラットフォーム固有の追加のエラー情報を返すために提供される。BIOS がエラーに関する追加情報を提供しない場合は、OEM エラーの値は SUCCESS に設定されなければならない。AH に SUCCESS（00H）または NOT_IMPLEMENTED（86H）が返された場合、OEM Error フィールドの値は未定義である。その他のすべての場合、OEM エラー・フィールドは、SUCCESS またはメーカに意味のある値に設定される。

INT 15H ベースのインターフェイスの各関数について、以下の項で説明する。

9.11.8.5. 関数 00H - 存在テスト

この関数は、BIOS が必要なマイクロコード・アップデート関数を実装しているかどうかを確認する。表 9-13. は、この関数のパラメータとリターンコードを示している。

表 9-13. 存在テストのパラメータ

入力 :		
AX	関数コード	0D042H
BL	サブ関数	00H - 存在テスト
出力 :		
CF	キャリーフラグ	キャリーがセットされる - 失敗 - AH にステータスが返される。 キャリーがクリアされる - すべての戻り値は有効である。
AH	リターンコード	
AL	OEM エラー	メーカの追加情報
EBX	シグネチャ第 1 部	'INTE' - シグネチャの第 1 部
ECX	シグネチャ第 2 部	'LPEP' - シグネチャの第 2 部
EDX	ローダのバージョン	マイクロコード・アップデート・ローダのバージョン番号
SI	アップデート数	NVRAM 内にマイクロコード・アップデートを記録できる 2048 バイトのアップデート・ブロックの数
リターンコード（コードの定義については表 9-18. を参照）		
SUCCESS		関数は正常に終了した。
NOT_IMPLEMENTED		関数は実装されていない。

説明

コール元は、BIOS が必要な関数を実装していることを確認するために、キャリーフラグ、リターンコード、64 ビットのシグネチャを確認しなければならない。各アップデート・ブロックのサイズはちょうど 2048 バイトである。アップデート数は、1 つの不揮発性 RAM 内に記録できる 2048 バイトのアップデート・ブロックの数を示す。

ローダのバージョン番号は、システム BIOS イメージ内に含まれているアップデート・ローダ・プログラムのリビジョンである。

9.11.8.6. 関数 01H - マイクロコード・アップデート・データ書き込み

この関数は、新しいマイクロコード・アップデートを BIOS 記憶デバイスに組み込む。表 9-14. は、この関数のパラメータとリターンコードを示している。

表 9-14. アップデート・データ書き込み関数のパラメータ

入力:		
AX	関数コード	0D042H
BL	サブ関数	01H ー アップデート書き込み
ES:DI	アップデート・アドレス	インテルのアップデート構造に対するリアル・モード・ポインタ。プロセッサが固定サイズのマイクロコード・アップデートのみをサポートする場合、このバッファのサイズは 2048 バイトである。 インテルのアップデート構造に対するリアル・モード・ポインタ。プロセッサが可変長のマイクロコード・アップデートをサポートする場合、このバッファのサイズは 64K バイトである。
CX	スクラッチ・パッド 1	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス。
DX	スクラッチ・パッド 2	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス。
SI	スクラッチ・パッド 3	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス。
SS:SP	スタック・ポインタ	32K バイトの最小スタック。
出力:		
CF	キャリーフラグ	キャリーがセットされる - 失敗 - AH にステータスが返される。 キャリーがクリアされる - すべての戻り値は有効である。
AH	リターンコード	呼び出しのステータス。
AL	OEM エラー	メーカの追加情報。
リターンコード（コードの定義については表 9-18. を参照）		
SUCCESS		関数は正常に終了した。
WRITE_FAILURE		記憶デバイスへの書き込みに失敗した。
ERASE_FAILURE		記憶デバイスの消去に失敗した。
READ_FAILURE		記憶デバイスの読み取りに失敗した。
STORAGE_FULL		すべての利用可能なアップデート・ブロックが、システム内のプロセッサに必要なアップデートでいっぱいになっているため、BIOS の不揮発性記憶領域にアップデートを格納できない。
CPU_NOT_PRESENT		該当するプロセッサ・ステッピングは、現在はシステム内に存在しない。
INVALID_HEADER		アップデート・ヘッダに、BIOS によって認識されていないヘッダ・バージョンまたはローダ・バージョンが入っている。
INVALID_HEADER_CS		アップデートのチェックサムが不適当である。
SECURITY_FAILURE		プロセッサがアップデートを拒否した。
INVALID_REVISION		記憶デバイス内に、同じリビジョンまたはより新しいリビジョンのアップデートが存在する。

説明

BIOS は、新しいアップデートをストアするための適切なアップデート・ブロックを、不揮発性記憶領域内で選択する役割を受け持つ。また、BIOS は、コール元が提供する情報の状態を確認する役割も受け持つ。例えば、BIOS は、提案されたアップデートを記憶領域に組み込む前に、そのアップデートの認証テストを行わなければならない。

BIOS は、アップデート・ブロックを NVRAM に書き込む前に、アップデートの構造が以下の基準に適合するかどうかを、以下の順番で確認しなければならない。

1. アップデート・ヘッダのバージョンは、BIOS によって認識されたアップデート・ヘッダのバージョンと同じでなければならない。
2. アップデート・ヘッダ内のアップデート・ローダのバージョンは、BIOS イメージに含まれるアップデート・ローダのバージョンと同じでなければならない。
3. アップデート・ブロックはチェックサムを行わなければならない。このチェックサムは、ヘッダ、データ、プロセッサ・シグニチャ・テーブルを含むアップデート構造内のダブルワードすべての 32 ビットの合計として計算される。

BIOS は、候補のアップデートをストアするためのアップデート・ブロックを、不揮発性記憶領域内で選択する。BIOS は、各プロセッサ・ステッピングに対応するアップデートが不揮発性記憶領域に 1 つだけ存在するように保証しさえすれば、利用可能なアップデート・ブロックを自由に選択することができる。選択したアップデート・ブロックにすでにアップデートが入っている場合は、以下の基準が満たされれば、そのブロックが上書きされる。

- 提案されたアップデート内のプロセッサ・シグネチャは、NVRAM 内の既存のアップデートのヘッダ内のプロセッサ・シグネチャと同じでなければならない（プロセッサ・シグニチャ +プラットフォーム ID ビット）
- 提案されたアップデート内のアップデート・リビジョンは、NVRAM 内の既存のアップデートのヘッダ内のアップデート・リビジョンより大きい値でなければならない。

上記の基準が満たされない場合、未使用のアップデート・ブロックが残っていなければ、BIOS は、システムにもはや存在しないプロセッサ・ステッピング用のアップデート・ブロックを上書きできる。この場合、BIOS は、アップデート・ブロックをスキャンし、マルチプロセッサ仕様テーブルに指定されたプロセッサ・ステッピングと、システム内の既存のプロセッサ・ステッピングを照合する。

最後に、BIOS は、提案されたアップデートを NVRAM にストアする前に、9.11.6. 項「マイクロコード・アップデート・ローダ」で説明した機構によって、そのアップデートの認証テストを行わなければならない。これを行うには、現在のプロセッサにアッ

プデートをロードして、CPUID 命令を実行し、MSR 08Bh を読み取り、返された値が、提案されたアップデートのヘッダ内のアップデート・リビジョンと一致するかどうか照合する必要がある。

BIOS は、アップデート書き込み関数を実行する際に、ヘッダ、アップデート・データ、拡張プロセッサ・シグニチャ・テーブル（適用可能な場合）を含むアップデート全体を記録しなければならない。アップデートを書き込むときは、上記の基準が満たされているものと見なして、元の内容を上書きする。この BIOS コールの使用によって新しい方のアップデートが上書きされないように保証することと、各プロセッサ・ステッピングに対応するアップデートが NVRAM 内に 1 つだけ存在するように保証することは、BIOS の役割である。

図 9-8. と図 9-9. は、BIOS が新しいマイクロコード・アップデートをストアする際に、アップデート・ブロックを選択し、アップデート・データの状態を確認する処理の流れを示している。

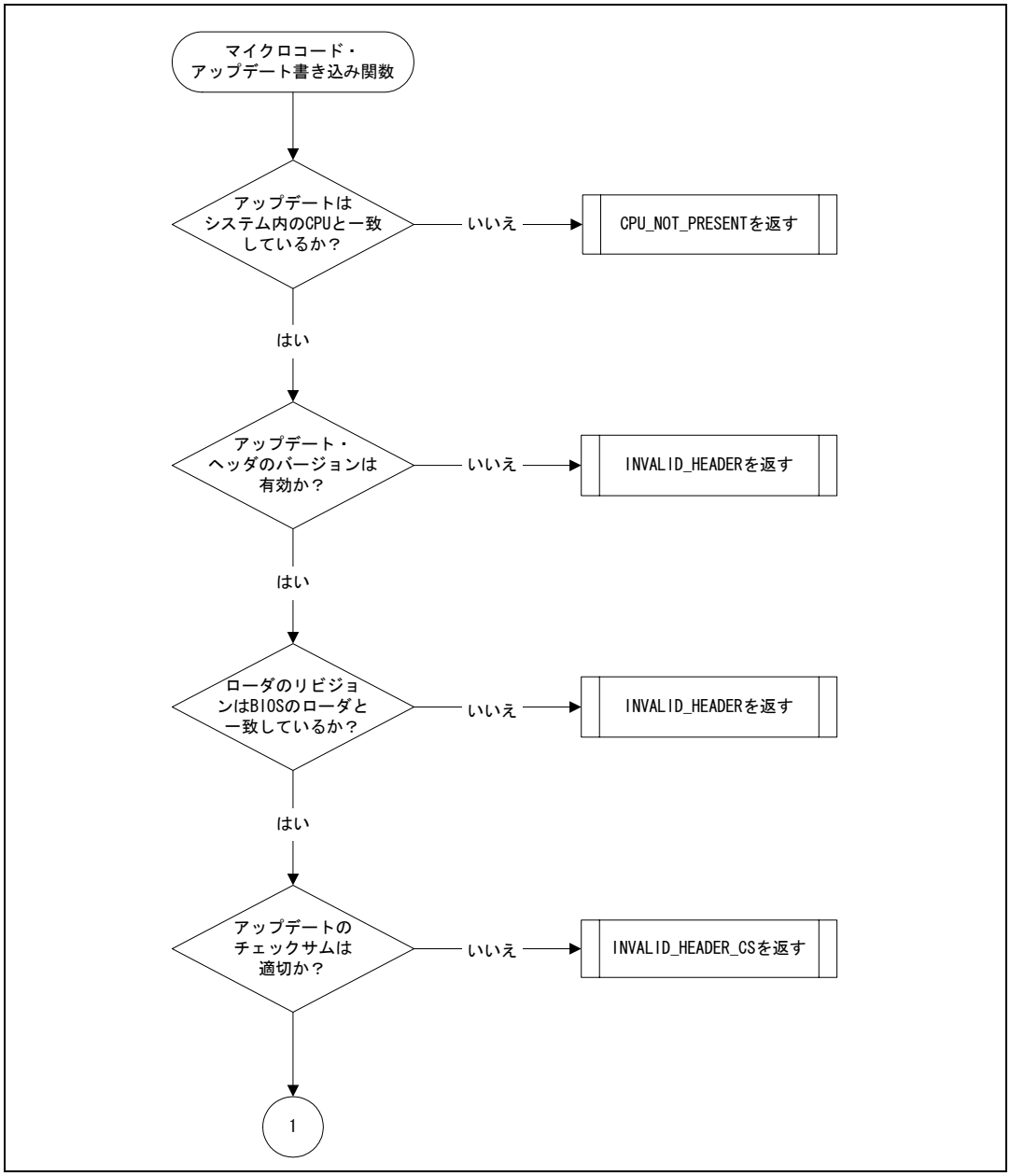


図 9-8. 書き込み操作のフローチャート [1]

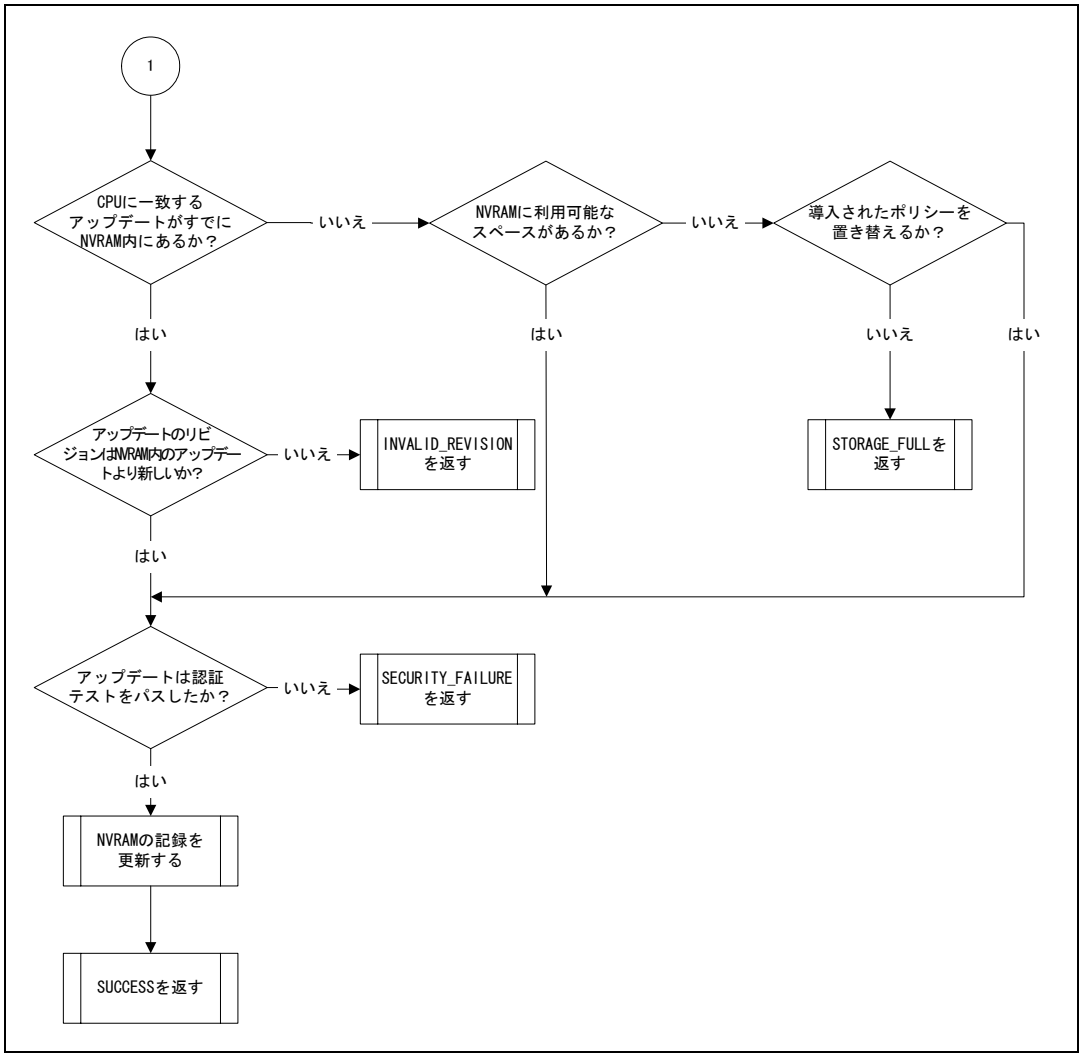


図 9-9. 書き込み操作のフローチャート [2]

9.11.8.7. 関数 02H - マイクロコード・アップデート制御

この関数は、プロセッサへのバイナリ・アップデートのロードをイネーブルにする。表 9-15. は、この関数のパラメータとリターンコードを示している。

表 9-15. アップデート制御サブ関数のパラメータ

入力 :		
AX	関数コード	0D042H
BL	サブ関数	02H ― アップデート制御
BH	タスク	下記の説明を参照。
CX	スクラッチ・パッド 1	64K バイトの RAM ブロックのリアル・モード・セグメント
DX	スクラッチ・パッド 2	64K バイトの RAM ブロックのリアル・モード・セグメント
SI	スクラッチ・パッド 3	64K バイトの RAM ブロックのリアル・モード・セグメント
SS:SP	スタックポインタ	32K バイトの最小スタック
出力 :		
CF	キャリーフラグ	キャリーがセットされる - 失敗 - AH にステータスが返される。 キャリーがクリアされる - すべての戻り値は有効である。
AH	リターンコード	呼び出しのステータス
AL	OEM エラー	メーカの追加情報
BL	アップデート・ステータス	インジケータをイネーブルまたはディスエーブルにする。
リターンコード（コードの定義については表 9-18. を参照）		
SUCCESS		関数は正常に終了した。
READ_FAILURE		記憶デバイスの読み取りに失敗した。

説明

この制御関数は、すべてのアップデートおよびプロセッサに対してグローバルに作用する。コール元は、アップデートのロード機能の現在の状態（イネーブルまたはディスエーブル）を変更することなく、その状態を確認できる。この関数は、コール元がバイナリ・アップデートのロード機能をディスエーブルにすることを許さない。これは、セキュリティ上のリスクを防ぐためである。

コール元は、表 9-16. の値のうち 1 つを BH レジスタに入れることによって、要求する操作を指定する。この関数が正常に終了した後、BL レジスタには、イネーブルまたはディスエーブル指示子が返される。ただし、この関数が失敗した場合は、アップデート・ステータスの戻り値は未定義である。

表 9-16. ニーモニック値

ニーモニック	値	意味
Enable	1	初期化時のアップデートのロードをイネーブルにする。
Query	2	アップデート制御の現在の状態を変更することなく、その状態を確認する。

この関数によって返される READ_FAILURE エラーコードは、制御関数が BIOS の NVRAM 内に実装されている場合にのみ意味を持つ。この機能の状態（イネーブル / ディスエーブル）は、CMOS の RAM ビットを使って実装されることもある。この場合は、読み取り失敗エラーは発生しない。

9.11.8.8. 関数 03H - マイクロコード・アップデート・データ読み取り

この関数は、現在インストールされているマイクロコード・アップデートを、BIOS の記憶領域から、コール元が用意した RAM バッファに読み取る。表 9-17. は、この関数のパラメータとリターンコードを示している。

表 9-17. アップデート制御サブ関数のパラメータ

入力 :		
AX	関数コード	0D042H
BL	サブ関数	03H ー アップデート読み取り
ES:DI	バッファアドレス	バイナリデータで書き込まれるインテルのアップデート構造に対するリアル・モード・ポインタ
ECX	スクラッチ・パッド 1	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス（下位の 16 ビット）
ECX	スクラッチ・パッド 2	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス（上位の 16 ビット）
DX	スクラッチ・パッド 3	64K バイトの RAM ブロックのリアル・モード・セグメント・アドレス
SS:SP	スタックポインタ	32K バイトの最小スタック
SI	アップデート番号	読み取られるアップデート・ブロックのインデックス番号。この値は、ゼロから始まる値で、存在テスト関数によって返されるアップデート数より小さい値でなければならない。
出力 :		
CF	キャリーフラグ	キャリーがセットされる - 失敗 - AH にステータスが返される。
キャリーがクリアされる ー すべての戻り値は有効である。		
AH	リターンコード	呼び出しのステータス
AL	OEM エラー	メーカの追加情報

表 9-17. アップデート制御サブ関数のパラメータ（続き）

リターンコード（コードの定義については表 9-18. を参照）	
SUCCESS	関数は正常に終了した。
READ_FAILURE	記憶デバイスの読み取りに失敗した。
UPDATE_NUM_INVALID	アップデート番号が、BIOS によって実装されるアップデート・ブロックの最大数を超えている。
NOT_EMPTY	指定されたアップデート・ブロックは、複数のブロックにわたる有効なマイクロコード・アップデートのストアに使用されている次のブロックである。指定されたブロックはヘッダブロックではなく、空ではない。

説明

この読み取り関数によって、コール元は、BIOS 内の既存の任意のマイクロコード・アップデート・データを読み取り、新しいアップデートの追加が適切かどうかを判断することができる。呼び出しが正常に終了すると、BIOS は、ES:DI で指定されたロケーションに、指定したマイクロコード・アップデートをストアするのに使用するすべてのアップデート・ブロックの内容をコピーする。

指定されたブロックはヘッダブロックではないが、複数のアップデート・ブロックにわたるマイクロコード・アップデートから得られる有効なデータを格納している場合、BIOS は、NOT_EMPTY エラーコード（失敗）を AH レジスタに返さなければならない。

この関数呼び出しからのリターンの後、アップデートのヘッダ・バージョンの値が 0FFFFFFFFH になった場合は、そのアップデート・ブロックは未使用であり、新しいアップデートのストアに使用できる。NVRAM 記憶領域の実際の管理方法については、BIOS によって異なるため、ここでは説明しない。例えば、空のブロックを示すために BIOS が使用するデータ値は、実際には 0FFFFFFFFH ではなく、ゼロであるかもしれない。この情報を変換して、この関数が用意したヘッダに入れるのは、BIOS の役割である。

9.11.8.9. リターンコード

関数呼び出しの実行後、表9-18.に示したリターンコードがAHレジスタに格納される。

表 9-18. リターンコードの定義

リターンコード	値	説明
SUCCESS	00H	関数は正常に終了した。
NOT_IMPLEMENTED	86H	関数は実装されていない。
ERASE_FAILURE	90H	記憶デバイスの消去に失敗した。
WRITE_FAILURE	91H	記憶デバイスへの書き込みに失敗した。
READ_FAILURE	92H	記憶デバイスの読み取りに失敗した。
STORAGE_FULL	93H	すべての利用可能なアップデート・ブロックが、システム内のプロセッサに必要なアップデートでいっぱいになっているため、BIOS の不揮発性記憶領域にアップデートを格納できない。
CPU_NOT_PRESENT	94H	該当するプロセッサ・ステッピングは、現在はシステム内に存在しない。
INVALID_HEADER	95H	アップデート・ヘッダに、BIOS によって認識されていないヘッダ・バージョンまたはローダ・バージョンが入っている。
INVALID_HEADER_CS	96H	アップデートのチェックサムが不適当である。
SECURITY_FAILURE	97H	プロセッサがアップデートを拒否した。
INVALID_REVISION	98H	記憶デバイス内に、同じリビジョンまたはより新しいリビジョンのアップデートが存在する。
UPDATE_NUM_INVALID	99H	アップデート番号が、BIOS によって実装されるアップデート・ブロックの最大数を超えている。
NOT_EMPTY	9AH	指定されたアップデート・ブロックは、複数のブロックにわたる有効なマイクロコード・アップデートのストアに使用されている次のブロックである。指定されたブロックはヘッダブロックではなく、空ではない。

10

メモリ・キャッシュ制御

第 10 章
メモリ・キャッシュ制御

10

本章では、IA-32 アーキテクチャのメモリ・キャッシュとキャッシュ制御のメカニズム、TLB、ストアバッファについて説明する。また、P6 ファミリ・プロセッサにあるメモリタイプ範囲レジスタ（MTRR）や、それを使って物理メモリ位置のキャッシングを制御する方法について説明する。

10.1. 内部キャッシュ、TLB、バッファ

IA-32 アーキテクチャには、命令やデータの一時的なオンチップ記憶（また外部記憶）用に、キャッシュ、トランスレーション・ルックアサイド・バッファ（TLB）、ストアバッファが用意されている。（図 10-1. に、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサのキャッシュ、TLB、およびストアバッファの配置を示す。）表 10-1. には、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサに用意されている上記のキャッシュとバッファの特性を示している。**これらのユニットのサイズと特性は各プロセッサ特有のものであり、プロセッサの将来のバージョンでは変更されることがある。**CPUID 命令を使用すると、この命令を実行したプロセッサのキャッシュとバッファのサイズと特性が返される（詳細は『IA-32 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、中巻 A』の第 3 章の「CPUID—CPU Identification」を参照）。

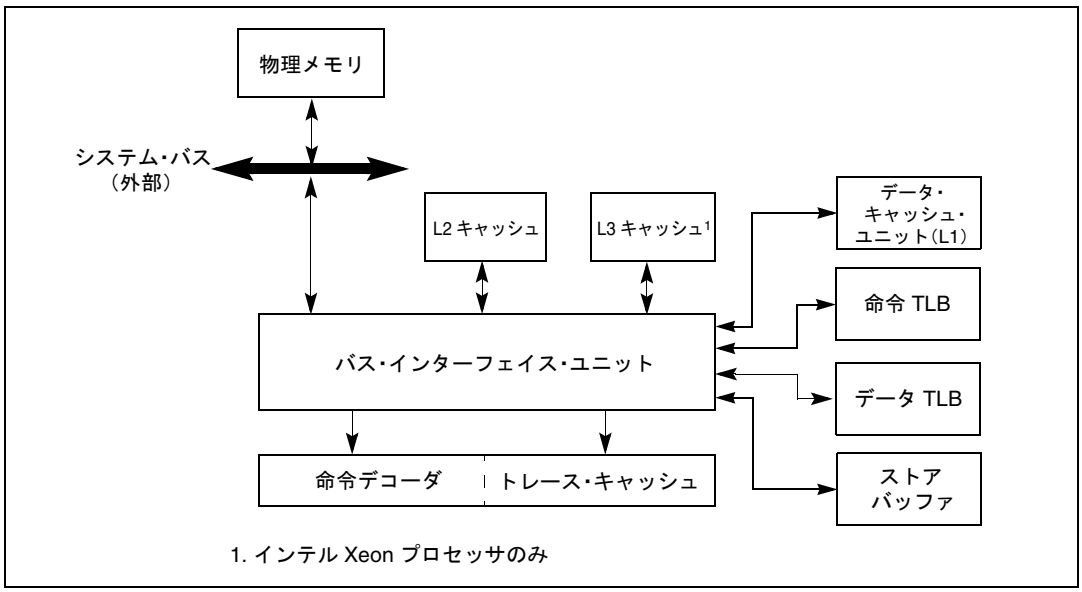


図 10-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサのキャッシュ構造

表 10-1. IA-32 プロセッサに搭載されたキャッシュ、TLB、ストアバッファおよび
ライト・コンバイニング・バッファの特性

キャッシュまたは バッファ	特性
トレース・キャッシュ 1	- インテル ® Pentium® 4 プロセッサおよびインテル ® Xeon™ プロセッサ : 12K μ ops、8 ウェイ・セット・アソシエィティブ。 - インテル ® Pentium® M プロセッサ : 実装されていない。 - P6 ファミリ・プロセッサおよびインテル ® Pentium® プロセッサ : 実装されていない。
L1 命令キャッシュ	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 実装されていない。 - インテル Pentium M プロセッサ : 32K バイト、8 ウェイ・セット・アソシエィティブ。 - P6 ファミリ・プロセッサとインテル Pentium プロセッサ : 8 または 16K バイト、4 ウェイ・セット・アソシエィティブ、32 バイトのキャッシュ・ライン・サイズ。また以前のバージョンのインテル Pentium プロセッサは 2 ウェイ・セット・アソシエィティブ。
L1 データ・キャッシュ	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 8K バイト、4 ウェイ・セット・アソシエィティブ、64 バイト・キャッシュ・ライン・サイズ。 - インテル Pentium M プロセッサ : 32K バイト、8 ウェイ・セット・アソシエィティブ、64 バイト・キャッシュ・ライン・サイズ。 - P6 ファミリ・プロセッサ : 16K バイト、4 ウェイ・セット・アソシアティブ、32 バイトのキャッシュ・ライン・サイズ。以前のバージョンの P6 ファミリ・プロセッサは、8K バイト、2 ウェイ・セット・アソシアティブ。 - インテル Pentium プロセッサ : 16K バイト、4 ウェイ・セット・アソシエィティブ、32 バイトのキャッシュ・ライン・サイズ。また以前のバージョンのインテル Pentium プロセッサは、8K バイト、2 ウェイ・セット・アソシエィティブ。

表 10-1. IA-32 プロセッサに搭載されたキャッシュ、TLB、ストアバッファおよび
ライト・コンバイニング・バッファの特性（続き）

キャッシュまたは バッファ	特性
L2 統合キャッシュ	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 256 または 512K バイト、8 ウェイ・セット・アソシエィティブ、セクタ化、64 バイト・キャッシュ・ライン・サイズ、128 バイト・セクタ・サイズ。 - インテル Pentium M プロセッサ : 1M バイト、8 ウェイ・セット・アソシエィティブ、64 バイト・キャッシュ・ライン・サイズ。 - P6 ファミリ・プロセッサ : 128K バイト、256K バイト、512K バイト、または 1M バイト、4 ウェイ・セット・アソシエィティブ、32 バイトのキャッシュ・ライン・サイズ。 - インテル Pentium プロセッサ（外部オプション）: システム特有、通常は 256 または 512K バイト、4 ウェイ・セット・アソシエィティブ、32 バイトのキャッシュ・ライン・サイズ。
L3 ユニファイド・ キャッシュ	- インテル Xeon プロセッサ : 512K バイトまたは 1M バイト、8 ウェイ・セット・アソシエィティブ、64 バイトのキャッシュ・ライン・サイズ、128 バイト・セクタ・サイズ。
命令 TLB (4K バイト・ページ)	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 128 エントリ、4 ウェイ・セット・アソシエィティブ。 - インテル Pentium M プロセッサ : 128 エントリ、4 ウェイ・セット・アソシエィティブ。 - P6ファミリ・プロセッサ: 32エントリ、4ウェイ・セット・アソシエィティブ。 - インテル Pentium プロセッサ : 32 エントリ、4 ウェイ・セット・アソシエィティブ。また MMX® テクノロジ Pentium プロセッサはフルセット・アソシエィティブ。
データ TLB (4K バイト・ページ)	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 64 エントリ、フルセット・アソシエィティブ ; ラージ・ページ・データ TLB と共用。 - インテル Pentium M プロセッサ : 128 エントリ、4 ウェイ・セット・アソシエィティブ。 - インテル Pentium と P6 ファミリ・プロセッサ : 64 エントリ、4 ウェイ・セット・アソシエィティブ。また MMX テクノロジ Pentium プロセッサはフルセット・アソシエィティブ。
命令 TLB (ラージページ)	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : ラージ・ページはフラグメント化されている。 - インテル Pentium M プロセッサ : 2 エントリ、フル・アソシエィティブ。 - P6 ファミリ・プロセッサ : 2 エントリ、2 ウェイ・セット・アソシエィティブ。 - インテル Pentium プロセッサ : 4K バイトのページに使用する TLB と同じ TLB を使う。
データ TLB (ラージページ)	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 64 エントリ、フルセット・アソシエィティブ ; スモール・ページ・データ TLB と共用。 - インテル Pentium M プロセッサ : 8 エントリ、フル・アソシエィティブ。 - P6 ファミリ・プロセッサ : 8 エントリ、4 ウェイ・セット・アソシエィティブ。 - インテル Pentium プロセッサ : 8 エントリ、4 ウェイ・セット・アソシエィティブ。また MMX テクノロジ Pentium プロセッサで 4K バイト・ページに使用する TLB と同じ TLB を使う。
ストアバッファ	- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ : 24 エントリ。 - インテル Pentium M プロセッサ : 16 エントリ。 - P6 ファミリ・プロセッサ : 12 エントリ。 - インテル Pentium プロセッサ : 2 バッファ、それぞれに 1 エントリ（MMX テクノロジ Pentium プロセッサは 4 エントリで 4 バッファを持つ）。

表 10-1. IA-32 プロセッサに搭載されたキャッシュ、TLB、ストアバッファおよび
ライト・コンバイニング・バッファの特性（続き）

キャッシュまたは バッファ	特性
ライト・コンバイニング（WC）バッファ	- インテルPentium 4プロセッサおよびインテルXeonプロセッサ: 6エントリ。 - インテル Pentium M プロセッサ : 6 エントリ。 - P6 ファミリ・プロセッサ : 4 エントリ。

注 :

1. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサで IA-32 アーキテクチャに導入された。

IA-32 プロセッサは、トレース・キャッシュ、L1（レベル 1）キャッシュ、L2（レベル 2）キャッシュ、L3（レベル 3）キャッシュという、4 つのタイプのキャッシュを実装している（図 10-1. を参照）。これらのキャッシュの用途は、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサとでは異なる。

- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ ― トレース・キャッシュは、命令デコーダからデコードされた命令（μops）をキャッシュする。L1 キャッシュはデータのみを含む。L2 キャッシュおよび L3 キャッシュはデータと命令を扱う統合キャッシュであり、プロセッサ・チップ上に実装されている。
- P6 ファミリ・プロセッサ ― L1 キャッシュは、IA-32 アーキテクチャ命令（事前にデコードされた命令）をキャッシュする部分と、データをキャッシュする部分に分けられる。L2 キャッシュはデータと命令を扱う統合キャッシュであり、プロセッサ・チップ上に実装されている。P6 ファミリ・プロセッサは、トレース・キャッシュを実装していない。
- インテル Pentium プロセッサ ― L1 キャッシュは、P6 ファミリ・プロセッサ上の L1 キャッシュと同じ構造である（トレース・キャッシュは実装されていない）。L2 キャッシュはデータと命令を扱う統合キャッシュであり、古いインテル Pentium プロセッサではプロセッサ・チップの外部に配置されていたが、最近のインテル Pentium プロセッサではプロセッサ・チップ上に実装されている。L2 キャッシュが外部に配置されているインテル Pentium プロセッサの場合、キャッシュへのアクセスはシステムバスを介して行われる。

インテル Pentium 4 プロセッサの L1 キャッシュと L2 キャッシュ、およびインテル Xeon プロセッサの L1 キャッシュ、L2 キャッシュ、L3 キャッシュに使用されるキャッシュ・ラインは、それぞれ 64 バイト幅である。プロセッサは常に、64 バイト境界から始まるシステムメモリからキャッシュ・ラインを読み取る（64 バイト・アライメントのキャッシュ・ラインは、6 つの最下位ビットがクリアされているアドレスを起点としている）。8 つの転送バースト・トランザクションによって、キャッシュ・ラインをメモリから取り入れることができる。キャッシュは、部分的にキャッシュ・ラインを取り込むことをサポートしていないため、単一のダブルワードをキャッシングするときでも、全部のラインをキャッシングする必要がある。

P6 ファミリ・プロセッサとインテル Pentium プロセッサの L1 キャッシュおよび L2 キャッシュのキャッシュ・ラインは、それぞれ 32 バイト幅である。プロセッサは、システムメモリの32バイト境界(メモリアドレスの最下位5ビットがクリアされている)から始まるキャッシュ・ラインを読み取る。キャッシュ・ラインは、4 転送バーストの 1 トランザクションによってメモリから取り入れることができる。キャッシュ・ラインを部分的に取り込む動作はサポートされていない。

インテルPentium 4プロセッサおよびインテルXeonプロセッサのトレース・キャッシュはIntel NetBurst® マイクロアーキテクチャの構成要素であり、保護モード、システム管理モード (SMM)、実アドレスモードのすべての実行モードで使用することができる。L1 キャッシュ、L2 キャッシュ、L3 キャッシュもすべての実行モードで使用できるが、SMM では、これらのキャッシュの扱いに注意が必要である (13.4.2. 項「SMRAM のキャッシング」を参照)。

TLB は、直前に使用されたページ・ディレクトリやページテーブルの各エントリをストアする。ページングがイネーブルである場合、TLB は、システムメモリにストアされているページテーブルの読み取りに必要なメモリアクセスの回数を減らして、メモリアクセスの速度を上げる。TLB は次の 4 つのグループに分割される。1) 4K バイト・ページ用の命令 TLB、2) 4K バイト・ページ用のデータ TLB、3) ラージページ (2M バイト・ページまたは4M バイト・ページ) 用の命令 TLB、4) ラージページ用のデータ TLB。TLB は通常、ページングがイネーブルになっている保護モードでだけアクティブになる。ページングがディスエーブル、またはプロセッサが実アドレスモードである場合、TLB は、明示的または暗黙的にフラッシュされるまで、その内容を保持する (詳細は 10.9. 節「トランスレーション・ルックアサイド・バッファ (TLB) の無効化」を参照)。

ストアバッファは、プロセッサ命令実行ユニットに関連付けられている。このバッファを使用すると、システムメモリへの書き込みまたは内部キャッシュへの書き込み (あるいはその両方) を保存することができる。また場合によっては両方を組み合わせて、プロセッサのバスアクセスを最適化できる。ストアバッファは、すべての実行モードで常にイネーブルである。

プロセッサのキャッシュは、その大部分がソフトウェアに透過である。キャッシュがイネーブルであるとき、命令とデータは、明示的なソフトウェア制御を必要としないでキャッシュを通過する。ただし、このキャッシュの動作を知っておくと、ソフトウェアの性能を最適化するときに役に立つ。例えば、キャッシュの大きさと置換アルゴリズムの知識があれば、キャッシュ・スラッシングを引き起こさないで一度に操作可能なデータ構造の大きさを予測できる。

マルチプロセッサ・システムでは、まれに、キャッシュのコンシステンシを維持するためにシステム・ソフトウェアを介入させなければならないことがある。このまれな

ケースに備えて、プロセッサには、キャッシュのフラッシングとメモリ・オーダリングの強制に使用するための特権的キャッシュ制御命令が用意されている。

インテル® Pentium® IIIプロセッサ、インテル Pentium 4プロセッサ、インテル Xeon プロセッサでは、ソフトウェアにより L1 キャッシュ、L2 キャッシュ、L3 キャッシュのパフォーマンスを向上させるための命令がいくつか導入された。例えば、PREFETCH*h* 命令やCLFLUSH 命令、非テンポラルな移動命令（MOVNTI、MOVNTQ、MOVNTDQ、MOVNTPS、MOVNTPD）などである。これらの命令の使用方法については、10.5.5. 項「キャッシュ管理命令」で説明している。

10.2. キャッシングの用語

（インテル® Pentium® プロセッサを始めとする）IA-32 アーキテクチャは、MESI（Modified ＝修正済み、Exclusive ＝排他的、Shared ＝共有、Invalid ＝無効）キャッシュ・プロトコルを使用して、内部キャッシュ間のコンシステンシと、他のプロセッサのキャッシュ間とのコンシステンシを維持する（10.4. 節「キャッシュ制御プロトコル」を参照）。

メモリから読み取られるオペランドがキャッシュ可能であることをプロセッサが認識した場合、そのプロセッサは、キャッシュ・ライン全体を適切なキャッシュ（L1、L2、L3、あるいはすべて）に読み込む。この動作を**「キャッシュ・ライン・フィル」**という。プロセッサが次にそのオペランドの実行を試みたとき、そのオペランドをストアしているメモリ位置がキャッシュされたままであれば、プロセッサはメモリを参照する代わりにキャッシュからオペランドを読み取ることができる。この動作を**「キャッシュ・ヒット」**という。

プロセッサは、メモリのキャッシュ可能領域にオペランドを書き込もうとするときに、まずそのメモリ位置のキャッシュ・ラインがキャッシュ内に存在するかどうかをチェックする。有効なキャッシュ・ラインが存在している場合、プロセッサは、システムメモリへ書き出す代わりに、(現時点で実施されている書き込み方式に応じて) そのオペランドをキャッシュに書き込むことができる。この動作を「**書き込みヒット**」という。書き込みがキャッシュをミスした（つまり、書き込みの対象となっているメモリ領域が有効なキャッシュ・ラインに存在していない) 場合は、プロセッサがキャッシュ・ライン・フィル、書き込み割り当てを実行する。次にオペランドをキャッシュ・ラインに書き込み、また（現時点で実施されている書き込み方式に応じて）そのオペランドをメモリに書き出すこともできる。オペランドがメモリに書き出される場合は、まず書き込みバッファに書き込まれ、次に、システムバスが利用できるようになった時点でストアバッファからメモリに書き込まれる。(インテル Pentium プロセッサでは、書き込みのミスがあってもキャッシュ・ライン・フィルが実行されないことに注意しなければならない。つまり、書き込みのミスは、常にメモリへの書き込みと

いう結果に終わる。このプロセッサでは、読み取りのミスによってだけキャッシュ・ライン・フィルが実行される。）

MP システムで動作する場合、（Intel486™プロセッサを始めとする）IA-32 プロセッサには、他のプロセッサによるシステムメモリや内部キャッシュへのアクセスを**「スヌープ（snoop＝チェック）」**する機能がある。インテル・アーキテクチャ・プロセッサは、このスヌープ機能を使用して、内部キャッシュとシステムメモリとのコンシステンシや、内部キャッシュとバス上の他のプロセッサ内にあるキャッシュとのコンシステンシを維持する。例えば、インテル Pentium プロセッサと P6 ファミリ・プロセッサの場合、あるプロセッサがすでに**「共有状態」**でキャッシュしているメモリ位置に、別のプロセッサが書き込みをしようとしていることがスヌープによって検出されたとき、スヌープを実行している側のプロセッサは、スヌープを実施しているキャッシュ・ラインを無効にする。そして次に同じメモリ位置にアクセスするときに、そのプロセッサはキャッシュ・ライン・フィルを実行する。

P6 ファミリ・プロセッサを始めとして、あるプロセッサのキャッシュ内ですでに修正されているが、まだシステムメモリにはライトバックされていないメモリ位置があり、そのメモリ位置に他のプロセッサがアクセスしようとしていることを、そのプロセッサが（スヌープによって）検出した場合、スヌープを実行している側のプロセッサは、キャッシュ・ラインが修正された状態に維持されていることを（HITM# 信号を使用して）他方のプロセッサに合図し、修正データの暗黙的なライトバックを実行する。この暗黙的なライトバックは、最初に要求を行ったプロセッサに直接転送される。そしてシステムメモリが更新されているのを確かめるためにメモリ・コントローラによってスヌープされる。この時点で、有効なデータを持つプロセッサは、データをシステムメモリに実際に書き込まないで、そのデータを他のプロセッサに渡すことができる。ただし、この動作をスヌープしてメモリを更新するのは、メモリ・コントローラの役割である。

10.3. 有効なキャッシングの方法

プロセッサは、システムメモリの任意の領域を L1 キャッシュ、L2 キャッシュ、L3 キャッシュにキャッシュすることができる。またシステムメモリの個々のページまたは領域内では、プロセッサが、（**メモリタイプ**ともいう）キャッシングのタイプも指定できる。この場合、各種のシステムフラグやレジスタを使用する（10.5. 節「キャッシュの制御」を参照）。現在のところ、IA-32 アーキテクチャ用に規定されているメモリタイプは以下のとおりである。（表 10-2. では、メモリタイプについてまとめて、それらの基本的な特性について述べている。）

- ストロング・キャッシュ不可（UC: Uncacheable）－ システムメモリ位置はキャッシュされない。すべての読み取りと書き込みはシステムバス上に現れ、順位を変更しないでプログラム順に実行される。推論的なメモリアクセス、推論的なペー

ジテーブル間の移動、または推論に基づく分岐ターゲットのプリフェッチは行われない。このタイプのキャッシュ制御は、メモリにマッピングされたI/Oデバイスに役に立つ。通常のRAMと一緒に使用した場合、プロセッサの処理能力は著しく低下する。

表 10-2. メモリタイプとそれぞれのプロパティ

メモリタイプとニーモニック	キャッシュ可能	ライトバック・キャッシュ可能	推論的な読み取りの許可	メモリ・オーダリング・モデル
ストロング・キャッシュ不可（UC）	いいえ	いいえ	いいえ	ストロング・オーダリング
キャッシュ不可（UC-）	いいえ	いいえ	いいえ	ストロング・オーダリング。PATを通じて選択することのみ可能。MTRRへのWCによりオーバーライドが可能。
ライト・コンバイニング（WC）	いいえ	いいえ	はい	ウィーク・オーダリング。MTRRプログラミングまたはPATを通じて選択することによって利用可能。
ライトスルー（WT）	はい	いいえ	はい	推論的なプロセッサ・オーダリング
ライトバック（WB）	はい	はい	はい	推論的なプロセッサ・オーダリング
ライト・プロテクト（WP）	読み取りは「はい」、書き込みは「いいえ」	いいえ	はい	推論的なプロセッサ・オーダリング。MTRR のプログラミングによって利用可能。

- キャッシュ不可（UC-）－ ストロング・キャッシュ不可（UC）メモリタイプと同じ特性を持つ。ただし、このメモリタイプは、WCメモリタイプのMTRRをプログラミングすることによりオーバーライドが可能である。このメモリタイプはインテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、インテル® Pentium® IIIプロセッサで使用可能であり、PATを介してのみ選択することができる。
- ライト・コンバイニング（WC: Write Combining）－（キャッシュ不可のメモリの場合と同じように）システムメモリ位置はキャッシュされない。またプロセッサのバス・コヒーレンシ・プロトコルによるコヒーレンシは実施されない。推論的な読み取りができる。複数の書き込みが遅延して、ライト・コンバイニング・バッファ（WCバッファ）内で組み合わされメモリのアクセス数を減らすことがある。WCバッファが部分的に一杯になっている場合は、シリアル化イベント（例えばSFENCEまたはMFENCE命令、CPUID命令の実行、キャッシュ不可メモリの読み取りまたは書き込み、割り込みの発生、LOCK命令の実行など）が次に発生するまで、書き込みが遅延されることがある。このタイプのキャッシュ制御は、ビデオ・フレーム・バッファに適している。この場合、書き込みによってメモリが更新されている間は書き込みの順序は重要でないため、書き込み内容をグラフィック画

面上に表示できる。WC メモリタイプのキャッシングの詳細については、10.3.1. 項「ライト・コンバイニング・メモリ位置のバッファリング」を参照。このメモリタイプをインテル® Pentium® Pro プロセッサおよびインテル® Pentium® II プロセッサで使用するには、MTRR をプログラミングする必要がある。インテル Pentium III プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサで使用するには、MTRR をプログラミングするか、または PAT を介して選択する必要がある。

- ライトスルー（WT: Write-through）－ システムメモリへの書き込みとシステムメモリからの読み取りがキャッシュされる。キャッシュ・ヒットではキャッシュ・ラインから読み取りが行われ、読み取りがミスするとキャッシュ・フィルが動作する。推論的な読み取りができる。書き込みはすべて、（可能であれば）キャッシュ・ラインに書き込まれ、システムメモリへのライトスルーを行う。メモリへのライトスルーを行う場合、無効なキャッシュ・ラインは決してフィルされない。そして有効なキャッシュ・ラインはフィルされるか、または無効にされるかのどちらかである。ライト・コンバイニングができる。このタイプのキャッシュ制御は、フレームバッファに適している。またはシステムメモリにアクセスしてもメモリアクセスのスヌープは実行しないデバイスがシステムバス上に存在している場合に適している。プロセッサ内のキャッシュとシステムメモリ間のコヒーレンシを実施する。
- ライトバック（WB: Write-back）－ システムメモリへの書き込みとシステムメモリからの読み取りがキャッシュされる。キャッシュ・ヒットではキャッシュ・ラインから読み取りが行われ、読み取りがミスするとキャッシュ・フィルが動作する。推論的な読み取りができる。書き込みがミスすると（インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは）キャッシュ・ライン・フィルが動作する。そして（可能であれば）キャッシュ内の全体で書き込みが実行される。ライト・コンバイニングが可能である。ライトバックのメモリタイプは、システムメモリへの不要な多数の書き込みを排除することで、バスのトラフィックを減らす。キャッシュ・ラインへの書き込みは、システムメモリにすぐに転送されないで、代わりにキャッシュ内に蓄積される。修正されたキャッシュ・ラインは、後でライトバック動作が実行されたときにシステムメモリに書き込まれる。ライトバック動作が起動されるのは、キャッシュ・ラインの割り当てを解除しなければならない場合である。例えば、すでにいっぱいになっているキャッシュ内に、新たにキャッシュ・ラインを割り当てようとしている場合などである。またライトバック動作は、キャッシュのコンシステンシを維持するためのメカニズムによっても起動される。このタイプのキャッシュ制御によって最高の性能が得られる。ただし条件として、システムメモリにアクセスするシステムバス上のデバイスすべてが、システムメモリとキャッシュのコヒーレンシを確保するためにメモリアクセスをスヌープできなければならない。
- ライト・プロテクト（WP: Write protected）－ 可能であれば、読み取りはキャッシュ・ラインから行われる。また読み取りがミスするとキャッシュ・フィルが動作する。書き込みはシステムバスに伝搬され、バス上のプロセッサすべての対応

するキャッシュ・ラインを無効にする。推論的な読み取りができる。このメモリタイプは、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、MTRR をプログラムすると使用できる（表 10-6. を参照）。

表 10-3. は、インテル Pentium プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサで使用可能なキャッシュ方式を示している。

表 10-3. インテル® Pentium® 4 プロセッサ、P6 ファミリ・プロセッサ、インテル® Xeon™ プロセッサ、インテル Pentium® プロセッサで使用できるキャッシング方法

メモリタイプ	インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ	P6 ファミリ・プロセッサ	インテル Pentium プロセッサ
ストロング・キャッシュ不可（UC）	はい	はい	はい
キャッシュ不可（UC-）	はい	はい *	いいえ
ライト・コンバイニング（WC）	はい	はい	いいえ
ライトスルー（WT）	はい	はい	はい
ライトバック（WB）	はい	はい	はい
ライト・プロテクト（WP）	はい	はい	いいえ

注 :

* インテル Pentium III プロセッサで導入された。インテル Pentium Pro プロセッサやインテル Pentium II プロセッサでは使用できない。

10.3.1. ライト・コンバイニング・メモリ位置のバッファリング

WC（ライト・コンバイニング）メモリタイプへの書き込みは、一般的な「キャッシュ」とは異なる意味でキャッシュされる。これらの書き込みは内部の L1 キャッシュ、L2 キャッシュ、L3 キャッシュとストア・バッファから離れた別の内部ライト・コンバイニング・バッファ（WC バッファ）内に保持される。この WC バッファはスヌープされないため、データのコヒーレンシは得られない。WC メモリへの書き込みのバッファリングを実行すると、ソフトウェアによる小さな時間ウィンドウによって、より多くの修正データが WC バッファに提供されると共に、ソフトウェアへの干渉は最小限に抑えられるようになっている。また WC メモリへの書き込みの WC バッファリングによってデータのコラプス（崩壊）が発生する。すなわち、同じメモリ位置に多重の書き込みをすると、その位置に最後に書き込まれたデータを残して、他の書き込みはすべて失われてしまう）。

WC バッファのサイズと構造は、アーキテクチャ上では規定されていない。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサの場合、WC バッファは複数の 64 バイト WC バッファから構成される。P6 ファミリ・プロセッサの場合、WC バッファは複数の 32 バイト WC バッファから構成される。

ソフトウェアにより WC メモリへの書き込みを開始すると、プロセッサは、一度に 1 つずつ WC バッファをフィルし始める。1 つまたは複数の WC バッファがフィルされると、プロセッサはバッファの内容をシステムメモリに追い出すことができる。WC バッファを追い出すプロトコルはプロセッサごとに異なっている。したがって、ソフトウェア上で、これに依存してシステムメモリのコヒーレンシを維持してはならない。WC メモリタイプを使用する場合、ソフトウェア側では、システムメモリへのデータの書き込みには遅延が伴う事実に注意する。また、システムメモリのコヒーレンシが必要とされる場合は、故意に WC バッファを空にしなければならない。

プロセッサは、いったん起動してデータを WC バッファからシステムメモリに移すと、どれくらいのバッファが有効なデータをストアしているかを基準にして、バス・トランザクションの方式を決定する。バッファがいっぱいの場合（例えば、すべてのバイトが有効な場合）、プロセッサは、バースト書き込みトランザクションをバス上で実行する。その結果、32 バイト（P6 ファミリ・プロセッサ）または 64 バイト（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）がすべて、単一のトランザクションでデータバス上を伝送される。1 つまたは複数の WC バッファのバイトが無効な場合（例えば、ソフトウェアによって書き込まれていない場合）、プロセッサは、「パーシャル・ライト（部分的な書き込み）」トランザクション（一度に 1 つの群、ここで 1 つの群は 8 バイト）を使用して、データをメモリに転送し始める。

メモリに伝送されたデータの WC バッファ 1 つにつき、4 つのパーシャル・ライト・トランザクション（P6 ファミリ・プロセッサ）または 8 つのパーシャル・ライト・トランザクション（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）が最大限度となる。

WC メモリタイプは、定義上、順序設定が緩やかになっている。いったん WC バッファ内のデータが送出され始めると、データは、データ定義のゆるやかなオーダリングに従う。オーダリング（順序付け）は、WC バッファの連続割り当てと割り当て解除のあいだでは守られない（例えば、システムバス上で WC バッファ 1 への書き込みの後に WC バッファ 2 への書き込みが続くことも、バッファ 2 の後にバッファ 1 が続くこともある）。WC バッファがメモリに、パーシャル・ライトとして送出される場合、連続したパーシャル・ライトのオーダリングは保証されない（例えば、チャンク 2 のパーシャル・ライトが、バス上で、チャンク 1 のパーシャル・ライトの前に現れることもあるし、またその逆の場合もある）。システムバスへの WC 伝搬を、トランザクションの最小単位の要素によってのみ保証されている。例えば、P6 ファミリ・プロセッサでは、完全にいっぱいになっている WC バッファは、任意のチャンク順序を使用して、常に単一の 32 ビット・バースト・トランザクションとして伝搬される。部分単位でデータが送出される WC バッファ送出では、同じチャンク（0 係数、8 アライメント）にストアされているすべてのデータが同時に伝搬される。同様に、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、WC バッファが一杯になると、トランザクション内の何らかのチャンク順序を使用して、常に単一バーストの 1 トランザクショ

ンとして送出される。部分単位で WC バッファを送出する場合には、同じチャンクにストアされているすべてのデータが同時に送出される。

10.3.2. メモリタイプの選択

最も単純なシステム・メモリ・モデルでは、読み取りや書き込みによる影響があるメモリ・マッピングされた I/O を使用しない。またフレームバッファも組み込まれない。そして、メモリすべてにライトバックのメモリタイプを使用する。I/O エージェントは、ライト・バック・メモリへの直接メモリアクセス（DMA）を実行できる。またキャッシュ・プロトコルは、キャッシュのコヒーレンシを維持する。

システムは、ストロング・キャッシュ不可メモリを他のメモリマップド I/O に使用できる。この場合、読み取りの影響があるメモリマップされた I/O に対し、常にストロング・キャッシュ不可メモリを使用しなければならない。

デュアルポート・メモリは書き込みの影響があると考えられ、比較的迅速な書き込みが望ましい。なぜならこの場合の書き込みは、その書き込みがメモリ・エージェントに到達するまでもう一方のポートで観察できないからである。システムは、ストロング・キャッシュ不可、キャッシュ不可、ライトスルー、またはライト・コンバイニング・メモリを、画面上に表示されるピクセル値をストアしているフレームバッファやデュアルポート・メモリに使用できる。フレーム・バッファ・メモリは一般的に大規模（数メガバイト）であり、通常はプロセッサによって読み取られる回数より書き込まれる回数のほうが多い。ストロング・キャッシュ不可メモリをフレームバッファに対して使用すると、多量のバス・トラフィックが発生する。なぜならバッファ全体での動作が、ライン書き込みではなくてパーシャル・ライトによって実現されるからである。ライト・スルー・メモリをフレームバッファに使用すると、プロセッサの L2 キャッシュおよび L3 キャッシュと L1 データ・キャッシュ内にある、他の有効なキャッシュ済みラインのほとんどを使い切ってしまう可能性がある。したがって、可能なときは常にシステムは、ライト・コンバイニング・メモリをフレームバッファに使用しなければならない。

ソフトウェアがライトバックのキャッシングの恩恵を受けるような方法でデータ構造にアクセスしない場合、ソフトウェアは、ページレベルのキャッシュ制御を使用して、適切で効果的なメモリタイプを割り当てられる。例えばソフトウェアは、大規模なデータ構造をいったん読み取ってから、その構造が別のエージェントによって書き直されるまで、その構造に再アクセスしないことがある。このような大規模のデータ構造は、キャッシュ不可としてマーク付けする必要がある。そうしないと、プロセッサが再度参照するであろうキャッシュ済みラインが読み取りによって追い出されてしまう。

同じような例に、（別のエージェントにデータをエクスポートするために）書き込まれたとしても、ソフトウェアによって決して読み取られない書き込み専用のデータ構

造があげられる。そのような構造をキャッシュ不可としてマーク付けできる。なぜならソフトウェアは、それ自体が書き込んだ値を決して読み取らないからである。（ただしキャッシュ不可メモリとして、このような構造はパーシャル・ライトによって書き込まれる。一方ライト・バック・メモリとしては、ライン書き込みによって書き込まれる。この書き込みは、もう一方のエージェントが構造を読み取って暗黙的なライトバックを起動するまで発生しない）。

インテル® Pentium® IIIプロセッサ、インテル® Pentium® 4プロセッサ、インテル® Xeon™ プロセッサでは、データのキャッシング、プリフェッチ、ライトバックの特性をソフトウェアでより緻密に制御するための新しい命令が提供されている。これらの命令を利用すると、ソフトウェア上で、ウィーク・オーダリングまたはプロセッサ・オーダリングのメモリタイプを使用してプロセッサのパフォーマンスを向上できる。ただし、必要に応じて、メモリ読み取り / 書き込みに対して強制的にストロング・オーダリングを実施する。また、これらの命令を利用して、データのキャッシングをソフトウェア上でより詳細に制御できる（これらの命令と意図された用途の説明については、10.5.5. 項「キャッシュ管理命令」を参照）。

10.4. キャッシュ制御プロトコル

この節では、IA-32 アーキテクチャ用に現在定義されているキャッシュ制御プロトコルについて説明する。このプロトコルは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサによって使用されている。

L1 データ・キャッシュ、L2 ユニファイド・キャッシュ、L3 ユニファイド・キャッシュでは、MESI（Modified ＝修正済み、Exclusive ＝排他的、Shared ＝共有、Invalid ＝無効）キャッシュ・プロトコルが他のプロセッサのキャッシュとのコンシステンシを維持する。L1 データ・キャッシュ、L2 ユニファイド・キャッシュ、L3 ユニファイド・キャッシュは、キャッシュ・ラインごとに 2 つの MESI ステータス・フラグを持つ。したがって、それぞれのラインは、表 10-4. で定義されている各状態のどれかにあるものとしてマーク付けができる。一般的には、MESI プロトコルの動作はプログラムからは透過である。

表 10-4. MESI キャッシュ・ラインの状態

キャッシュ・ラインの状態	M（修正済み）	E（排他的）	S（共有）	I（無効）
このキャッシュ・ラインは有効？	はい	はい	はい	いいえ
メモリのコピーは？	古い	有効	有効	—
他のプロセッサのキャッシュ内にコピーがある？	いいえ	いいえ	おそらくある	おそらくある
このラインへの書き込みは？	システムバスには送出されない	システムバスには送出されない	プロセッサにラインの排他的所有権を獲得させる	システムバスに直接送出される

P6 ファミリ・プロセッサの L1 命令キャッシュは、書き込みできないため、MESI プロトコルの「SI」部分だけを使用する。命令キャッシュは、データ・キャッシュ内の変更をモニタして、命令に変更があった場合にキャッシュ間のコンシステンシを維持する。キャッシング命令に含まれている意味の詳細については、10.6. 節「自己修正コード」を参照。

10.5. キャッシュの制御

IA-32 アーキテクチャでは、データや命令のキャッシングを制御したり、プロセッサ、キャッシュ、メモリ間の読み取り / 書き込みのオーダリングを制御するためのメカニズムが提供されている。これらのメカニズムは、次の 2 つのグループに分けることができる。

- キャッシュ制御レジスタおよびビット。IA-32 アーキテクチャでは、いくつかの専用レジスタ、制御レジスタ内の各種ビット、および L1 キャッシュ、L2 キャッシュ、L3 キャッシュ内のキャッシング・システム・メモリ位置を制御するページ・テーブル・エントリおよびディレクトリ・テーブル・エントリを規定している。これらのメカニズムにより、仮想メモリページおよび物理メモリ領域のキャッシングを制御する。
- キャッシュ制御命令およびメモリ・オーダリング命令。IA-32 アーキテクチャでは、データのキャッシング、メモリ読み取り / 書き込みのオーダリング、データのプリフェッチを制御する命令をいくつか提供している。これらの命令により、ソフトウェア上で、特定のデータ構造のキャッシングやメモリ内の特定のロケーションのメモリ・コヒーレンシを制御し、プログラム内の特定のロケーションでのストロング・メモリ・オーダリングを強制できる。

以降の各項では、これらの 2 つのグループのキャッシュ制御メカニズムについて説明する。

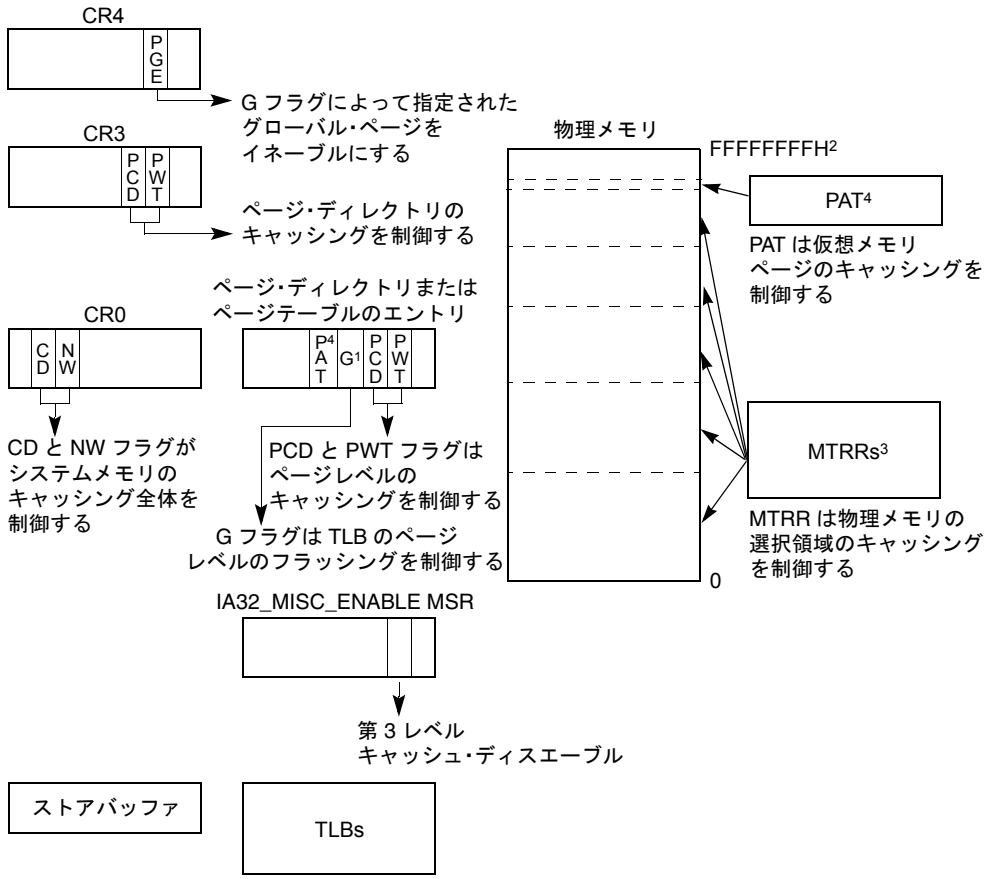
10.5.1. キャッシュ制御レジスタおよびビット

現行の IA-32 アーキテクチャでは、キャッシングをイネーブルにするとき、またはキャッシングをメモリ内のさまざまなページや領域に限定するときに使用するためのキャッシュ制御レジスタおよびビットを以下のように用意している（図 10-2. を参照）。

- CD フラグ、制御レジスタ CR0 のビット 30 ー システムメモリ位置のキャッシングを制御する。詳細は 2.5. 節「制御レジスタ」を参照。CD フラグがクリアされている場合、キャッシングはシステムメモリ全体に対してイネーブルになる。ただし他のキャッシュ制御メカニズムによって、メモリの個々のページや領域に対するキャッシングが制限されることがある。CD フラグが設定された場合、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサではキャッシングがプロセッサのキャッシュ内（キャッシュ階層）に制限され、インテル® Pentium® プロセッサではキャッシングが行われない（以下の注記を参照）。ただし CD フラグが設定されているために、キャッシュはそのままスヌープのトラフィックに応答し続ける。メモリのコヒーレンシを確保するために、キャッシュは明示的にフラッシュしなければならない。プロセッサの処理能力を最高にするためには、制御レジスタ CR0 の CD と NW フラグの両方をクリアしなければならない。表 10-5. に、CD と NW フラグの相互作用を示す。

注記

CD フラグ設定の影響は、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサとインテル Pentium プロセッサでは少し異なる（表 10-5. を参照）。CD フラグの設定後にメモリのコヒーレンシを確保するには、キャッシュを明示的にフラッシュしなければならない（10.5.3. 項「キャッシングの防止」を参照）。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサの場合に CD フラグを設定すると、キャッシュ・ライン・フィルが修正されて動作が更新される。またインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、CD フラグを設定しても、MTTR がディスエーブルであるか、またはすべてのメモリがアンキャッシュド（キャッシュ不可）として参照されるかのどちらか（あるいはその両方）でない限り、メモリアクセスの厳密な順序付けは実施されない（7.2.4. 項「メモリ・オーダリング・モデルのストロング・オーダリング / ウィーク・オーダリング」を参照）。



1. G フラグはインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ内でだけ使用できる。
2. 36ビットの物理アドレス指定を使用するときの最大物理アドレス・サイズは、FFFFFFFFFHである。
3. MTRR はインテル Pentium 4 プロセッサおよび P6 ファミリ・プロセッサ内でだけ使用できる。同じような制御は、KEN# と WB/WT# ピンを持つ Pentium プロセッサ内で利用できる。
4. PAT は Pentium III プロセッサおよび Pentium 4 プロセッサでのみ利用できる。

図 10-2. IA-32 プロセッサで有効なキャッシュ制御レジスタおよびビット

表 10-5. キャッシュ動作モード

CD	NW	キャッシングと読み取り / 書き込みの方式	L1	L2[1]
0	0	キャッシュの通常モード。キャッシュの最高性能動作。		
		- 読み取りヒットでキャッシュにアクセスする。また読み取りミスで置換が発生することがある。	はい	はい
		- 書き込みヒットでキャッシュが更新される。	はい	はい
		- 共有ラインへの書き込みのみ。書き込みミスでシステムメモリが更新される。	はい	はい
		- 書き込みミスでキャッシュ・ライン・フィルが発生する。	はい	はい
		- 書き込みヒットで、関連する読み取り無効サイクルで共有ラインを MTRR の制御下にある変更済みラインに変更できる。	はい	はい
		-（インテル Pentium プロセッサのみ）書き込みミスでキャッシュ・ライン・フィルは発生しない。	はい	
		-（インテル Pentium プロセッサのみ）書き込みヒットで、共有ラインを WB/WT# の制御下にある排他的ラインに変更できる。	はい	
		- 無効化が可能。	はい	はい
		- 外部のスヌープ・トラフィックが用意されている。	はい	はい
0	1	無効な設定。 エラーコード 0 の一般保護例外（#GP）を生成する。	NA	NA
1	0	ノーフィルキャッシュ・モード。メモリのコヒーレンシが維持される。		
		-（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）電源投入またはリセット後のプロセッサの状態である。	はい	はい
		- 読み取りヒットでキャッシュにアクセスする。また読み取りミスで置き換えは発生しない（下記のインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサを参照）。	はい	はい
		- 書き込みヒットでキャッシュが更新される	はい	はい
		- 共有ラインへの書き込みのみ。書き込みミスでシステムメモリが更新される。	はい	はい
		- 書き込みミスでメモリがアクセスされる。	はい	はい
		- 書き込みヒットで、共有ラインを、MTRR の制御下にあって対応する読み取り無効化サイクルを持つ排他的ラインに変更できる。	はい	はい
		-（インテル Pentium プロセッサのみ）書き込みヒットで、共有ラインを WB/WT# の制御下にある排他的ラインに変更できる。	はい	
		-（インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、および P6 ファミリ・プロセッサのみ）MTRR がディスエーブルであるか、またはすべてのメモリがアンキャッシュドとして参照されるかのどちらか（あるいはその両方）でない限り、メモリアクセスの厳密な順序付けは実施されない（7.2.4. 項を参照）。	はい	はい
		- 無効化が可能。	はい	はい
		- 外部のスヌープ・トラフィックが用意されている。	はい	はい
		-（インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの場合）アクセスされたメモリがアンキャッシュド（キャッシュなし）としてマップされていなければ、ラインのフィルおよびキャッシュ階層への置換が可能である。	はい	はい
1	1	メモリのコヒーレンシは維持されない。[2]		
		-（P6 ファミリ・プロセッサおよびインテル Pentium プロセッサ）電源投入またはリセット後のプロセッサの状態である。	はい	はい
		- 読み取りヒットでキャッシュにアクセスする。また読み取りミスで置き換えは発生しない。	はい	はい
		- 書き込みヒットでキャッシュが更新され、排他的ラインが変更済みラインに変更される。	はい	はい
		- 共有ラインは、書込みヒット後も共有されたままになる。	はい	はい
		- 書き込みミスはメモリにアクセスする。	はい	はい
		- スヌープを実行する場合、無効化は禁止される。ただし INVD 命令と WBINVD 命令を使った無効化は可能である。	はい	はい
		- 外部のスヌープ・トラフィックが用意されている。	いいえ	はい

注 :

1. この表の L2/L3 カラムは、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサに限定されている。このカラムの目的は、外部のプラットフォーム独自のライトバック L2 キャッシュを持っているインテル Pentium プロセッサ・ベースのシステムで、何ができるかを表すことである。
2. インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、このモードはサポートされていない。CD ビットおよび NW ビットを 1 にセットすると、非フィル・キャッシュ・モードが選択される。

- NW フラグ、制御レジスタ CR0 のビット 29 ー システムメモリ位置への書き込み方式を制御する（2.5. 節「制御レジスタ」を参照）。NW フラグと CD フラグがクリアされている場合、ライトバックはシステムメモリ全体に対してイネーブルになるが、他のキャッシュ制御メカニズムによってメモリの個々のページやメモリ領域に対しては制限されることがある。表 10-5. では、CD フラグと NW フラグの組み合わせがキャッシングに与える影響を示している。

注記

インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの場合、NW フラグは関係のないフラグである。すなわち、CD フラグをセットすると、NW フラグの設定に関係なく、プロセッサは非フィル・キャッシュ・モードになる。

インテル Pentium プロセッサでは、L1 キャッシュがディスエーブルの場合（制御レジスタ CR0 内の CD フラグと NW フラグが設定されている場合）、外部スヌープは DP（デュアル・プロセッサ）システムで受諾されるが、ユニプロセッサ・システムでは禁止される。スヌープが禁止されているときは、アドレスのパリティはチェックされない。また APCHK# は破損アドレスに対してアサートされない。ただし、スヌープが受諾されるときはアドレスパリティがチェックされ、APCHK# が破損アドレスに対してアサートされる。

- ページ・ディレクトリとページテーブルの各エントリ内にある PCD フラグ ー 個々のページテーブルとページに対するキャッシングをそれぞれ制御する（3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」を参照）。このフラグは、ページングがイネーブルで、制御レジスタ CR0 の CD フラグがクリアされている場合にのみ有効になる。PCD フラグは、クリアされているときにページテーブルまたはページのキャッシングをイネーブルにし、設定されているときはキャッシングしないようにする。
- ページ・ディレクトリとページテーブルの各エントリ内にある PWT フラグ ー 個々のページテーブルとページに対する書き込み方式をそれぞれ制御する（3.7.6. 項「ページ・ディレクトリ・エントリとページ・テーブル・エントリ」を参照）。このフラグは、ページングがイネーブルで、制御レジスタ CR0 の NW フラグがクリアされている場合にのみ有効になる。PWT フラグは、クリアされているときにページテーブルまたはページのライト・バック・キャッシングをイネーブルにし、設定されているときはライト・スルー・キャッシングをイネーブルにする。

- 制御レジスタ CR3 の PCD フラグと PWT フラグ。ページ・ディレクトリに対するグローバル・キャッシングと書き込み方式を制御する（2.5. 節「制御レジスタ」を参照）。PCD フラグは、クリアされているときにページ・ディレクトリのキャッシングをイネーブルにし、設定されているときはキャッシングしないようにする。PWT フラグは、クリアされているときにページ・ディレクトリのライト・バック・キャッシングをイネーブルにし、設定されているときはライト・スルー キャッシングをイネーブルにする。これらのフラグは、個々のページテーブルに対するキャッシングと書き込み方式には影響を及ぼさない。これらのフラグが有効になるのは、ページングがイネーブルにされ、しかも制御レジスタ CR0 の CD フラグがクリアされている場合だけである。
- ページ・ディレクトリとページテーブルの各エントリの G（グローバル）フラグ（P6 ファミリ・プロセッサの IA-32 アーキテクチャに導入）― 個々のページに対する TLB エントリのフラッシングを制御する。このフラグの詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照。
- 制御レジスタ CR4 の PGE（ページ・グローバル・イネーブル）フラグ ― G フラグによるグローバル・ページの設定をイネーブルにする（このフラグの詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照）。
- メモリタイプ範囲レジスタ（MTRR）（P6 ファミリ・プロセッサに導入）― 物理メモリの特定な領域で使用されるキャッシングのタイプを制御する。10.3. 節「有効なキャッシングの方法」で説明しているキャッシングのどのタイプも選択できる。MTRR の詳細については、10.11. 節「メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）」を参照。
- ページ属性テーブル（PAT）MSR ―（インテル® Pentium® III プロセッサで導入された。）プロセッサのメモリタイプ指定機能を拡張し、ページごとにメモリタイプを割り当てることが可能である（10.12. 節「ページ属性テーブル（PAT）」を参照）。
- 第 3 レベル・キャッシュ無効フラグ、IA32_MISC_ENABLE MSR のビット 6（インテル Xeon プロセッサで導入）― L1 および L2 キャッシュとは無関係に、L3 キャッシュを無効または有効にできる。
- KEN# ピンと WB/WT# ピン（インテル Pentium プロセッサ）― 外部ハードウェアはメモリの固有領域に使用されるキャッシング方法を制御できる。各ピンは、（全く同じではないが）MTRR に類似した機能を、P6 ファミリ・プロセッサ内で実行する。
- PCD ピンと PWT ピン（インテル Pentium プロセッサ）― これらの各ピンは、制御レジスタ CR3 内と、ページ・ディレクトリのエントリおよびページテーブルのエントリ内にある PCD フラグと PWT フラグに対応している。これらのピンを使用すると、外部の L2 キャッシュ内でのキャッシングを 1 ページごとに制御できる。この制御は、インテル Pentium プロセッサと Intel486™ プロセッサの L1 キャッシュで使用される制御と一致している。インテル Pentium 4 プロセッサ、インテル Xeon プ

ロセッサ、P6 ファミリ・プロセッサには、チップ・パッケージの内部に L2 キャッシュが封印されているから、これらのピンは用意されていない。

10.5.2. キャッシュ制御の優先

キャッシュ制御フラグとMTRRは階層的に動作してキャッシングを制限する。つまり、CD フラグが設定されている場合には、キャッシングが全体的に実行できなくなる（表 10-5. を参照）。CD フラグがクリアされている場合は、ページ・レベル・キャッシュ制御フラグまたはMTRR のどちらか（あるいはその両方）を使用してキャッシングを制限できる。ページレベルのキャッシング制御と MTRR キャッシング制御とがオーバーラップしている場合は、キャッシングを不可能にするメカニズムが優先する。例えば、MTRR がシステムメモリの領域をキャッシュできないようにした場合、ページ・レベル・キャッシュ制御フラグを使用してもその領域内のページのキャッシングをイネーブルにできない。この逆も同様である。つまり、ページ・レベル・キャッシュ制御がページをキャッシュ不可能と指定している場合、MTRR を使用しても、ページをキャッシュ可能にできない。

ページやメモリ領域に対するライト・バック・キャッシング方式とライト・スルー・キャッシング方式のアサイメントがオーバーラップした場合には、ライトスルー方式が優先する。（MTRR または PAT によってのみ割り当てることができる）ライト・コンバイニング方式は、ライトスルーまたはライトバックのどちらかに優先する。

ページレベルでのメモリタイプの選択は、ページのメモリタイプを選択するのに PAT を使用しているかどうかによって変わる。詳しくは、以降の各項で説明する。

第 3 レベル・キャッシュ無効フラグ（IA32_MISC_ENABLE MSR のビット 6）は、L3 キャッシュの CD フラグ、MTRR、PAT に優先する。つまり、第 3 レベル・キャッシュ無効フラグがセットされている（キャッシュが無効にされている）場合、その他のキャッシュ制御は L3 キャッシュに影響を与えない。このフラグがクリアされている（キャッシュが有効になっている）場合、その他のキャッシュ制御は、L3 キャッシュに対しても L1 および L2 キャッシュと同じ機能を持つ。

10.5.2.1. インテル® Pentium® Pro プロセッサおよびインテル® Pentium® II プロセッサにおけるメモリタイプの選択

インテル® Pentium® Pro プロセッサおよびインテル® Pentium® II プロセッサでは、PAT をサポートしていない。したがって、MTRR、およびページのページ・テーブル・エントリやページ・ディレクトリ・エントリの PCD ビットと PWT ビットを使用して、そのページの有効なメモリタイプを選択する。表 10-6. では、ノーマルなキャッシングが実施される場合の（制御レジスタ CR0 内の CD フラグと NW フラグがクリアされている場合の）、有効なメモリタイプに対する MTRR メモリタイプのマッピングとページレベルのキャッシング属性を示している。灰色で表示されている組み合わせ部分

は、インテル Pentium Pro プロセッサおよびインテル Pentium II プロセッサで定義された値である。したがってシステム設計者は、この定義の組み合わせを避けることが望ましい。

表 10-6. インテル® Pentium® Pro プロセッサおよびインテル® Pentium® II プロセッサ * の有効なページ・レベル・メモリ・タイプ *

MTRR メモリタイプ	PCD の値	PWT の値	有効なメモリタイプ
UC	X	X	UC
WC	0	0	WC
	0	1	WC
	1	0	WC
	1	1	UC
WT	0	X	WT
	1	X	UC
WP	0	0	WP
	0	1	WP
	1	0	WC
	1	1	UC
WB	0	0	WB
	0	1	WT
	1	X	UC

注：

* インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Pentium® III プロセッサで、ページ・テーブル・エントリおよびページ・ディレクトリ・エントリで PAT ビットを使用していない（0 にセットされている）場合は、これらの有効なメモリタイプが適用される。

ノーマルなキャッシングが実施されるときは、以下の各ルールを使用して表 10-6. に示されるような有効なメモリタイプが決定される。

1. ページの PCD 属性と PWT 属性が両方とも 0 の場合、有効なメモリタイプは、MTRR 定義のメモリタイプと同一のものである。
2. PCD フラグが設定されている場合、有効なメモリタイプは UC である。
3. PCD フラグがクリアされていて PWT フラグが設定されている場合、有効なメモリタイプは、WB メモリタイプに対しては WT であり、他のすべてのメモリタイプに対しては MTRR 定義のメモリタイプである。
4. PCD フラグの値と PWT フラグの値が正反対に設定されている場合、WP と WC の各メモリタイプに対してはモデル特有のものとみなされ、WB、WT、UC の各メモリタイプ対しては、アーキテクチャによって定義されたものとみなされる。

10.5.2.2. インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Pentium® III プロセッサにおけるメモリタイプの選択

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Pentium® III プロセッサでは、PAT を使用してページレベルで有効なメモリタイプを選択する。この場合、MTRR、およびページ・テーブル・エントリやページ・ディレクトリ・エントリの PAT、PCD、PWT ビットで選択される PAT エントリの値によって、そのページのメモリタイプが選ばれる（10.12.3. 項「PAT からのメモリタイプの選択」を参照）。表 10-7. には、ノーマルなキャッシングが実施される場合（制御レジスタ CR0 の CD フラグと NW フラグがクリアされている場合）の、有効なメモリタイプに対する MTRR メモリタイプと PAT エントリタイプの対応を示している。灰色で表示されている組み合わせ部分は、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、インテル Pentium III プロセッサにおいて、プロセッサ固有に定義されている値である。したがって、システム設計者は、この定義の組み合わせを避けることが望ましい。

表 10-7. インテル® Pentium® III プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサにおけるページレベルでの有効なメモリタイプ

MTRR メモリタイプ	PAT エントリの値	有効なメモリタイプ
UC	UC	UC[1]
	UC-	UC[1]
	WC	WC
	WT	UC[1]
	WB	UC[1]
	WP	UC[1]
WC	UC	UC[2]
	UC-	WC
	WC	WC
	WT	未定義
	WB	WC
	WP	未定義
WT	UC	UC[2]
	UC-	UC[2]
	WC	WC
	WT	WT
	WB	WT
	WP	未定義

表 10-7. インテル® Pentium® III プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサにおけるページレベルでの有効なメモリタイプ（続き）

MTRR メモリタイプ	PAT エントリの値	有効なメモリタイプ
WB	UC	UC[2]
	UC-	UC[2]
	WC	WC
	WT	WT
	WB	WB
	WP	WP
WP	UC	UC[2]
	UC-	未定義
	WC	WC
	WT	未定義
	WB	WP
	WP	WP

注：

1. この UC 属性は MTRR から得られる。データがキャッシュされている可能性はないため、プロセッサがキャッシュをスヌープする必要はない。パフォーマンス上の理由から、この属性を使用した方が望ましい。
2. この UC 属性はページ・テーブル・エントリまたはページ・ディレクトリ・エントリから得られる。ページの別名定義のためにデータがキャッシュされている可能性があるため、プロセッサはキャッシュをチェックする必要がある。この設定はあまり望ましくない。

10.5.2.3. 異なるメモリタイプのページへの値の書き込み

メモリ内の隣接する 2 つのページのメモリタイプが異なり、これらの 2 つのページのページ境界をまたがるメモリ位置に 1 ワード以上のオペランドが書き込まれる場合、このオペランドが二度メモリに書き込まれることがある。この動作によって、実メモリへの書き込みの際に問題が起きることはない。ただし、これらのページに割り当てられたメモリ空間がデバイスにマップされると、そのデバイスが誤動作する場合がある。

10.5.3. キャッシングの防止

L1 キャッシュ、L2 キャッシュ、L3 キャッシュがイネーブルにされてキャッシュ・フィルを受け取った後に、これらのキャッシュをディスエーブルにするには、以下の操作手順を実行する。

1. ノーフィル・キャッシュ・モードに入る（制御レジスタ CR0 の CD フラグを 1 にセットし、NW フラグを 0 にセットする）。
2. WBINVD 命令を使用して、すべてのキャッシュを空にする。

3. MTRR をディスエーブルにし、デフォルトのメモリタイプをアンキャッシュドに設定するか、またはアンキャッシュド・メモリ・タイプに対するすべての MTRR を設定する（詳細は 10.11.2.1. 項「IA32_MTRR_DEF_TYPE MSR レジスタ」で TYPE フィールドと E フラグについて説明している箇所を参照）。

CD フラグがセットされた後は、キャッシュを空にして（手順 2）、システムメモリのコヒーレンシを保証しなければならない。キャッシュが空にされていないと、読み取り時にキャッシュ・ヒットが発生し、有効なキャッシュ・ラインからデータが読み込まれてしまう。

注記

制御レジスタ CR0 の CD フラグをセットすると、プロセッサのキャッシュ動作が表 10-5. に示すように変わる。ただし、これにより、すべての物理メモリの有効なメモリタイプが UC に強制されるわけではなく、厳密なメモリ・オーダリングが強制されるわけでもない。すべての物理メモリ上で UC メモリタイプおよび厳密なメモリ・オーダリングを強制するには、MTRR のすべてを UC メモリタイプ向けにプログラムするか、あるいは MTRR をディスエーブルにする必要がある。

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサの場合、上記の一連のステップが実行されたあと、WBINVD 命令の終了後に実際に MTRR がディスエーブルにされるまでのコードを含むキャッシュ・ラインが、キャッシュ階層に保持されることがある。この場合、キャッシュからコードを完全に除去するには、MTRR がディスエーブルにされたあとで、WBINVD 命令をもう一度実行する必要がある。

10.5.4. L3 キャッシュの無効化と有効化

第 3 レベル・キャッシュ無効フラグ（IA32_MISC_ENABLE MSR のビット 6）を使用して、L1 および L2 キャッシュとは無関係に、L3 キャッシュを無効または有効にできる。この制御を使用して L3 キャッシュを無効または有効にする前に、ソフトウェアは、10.5.3. 項「キャッシングの防止」で説明したように、すべてのプロセッサ・キャッシュを無垢にしてフラッシュし、L3 キャッシュにストアされた情報が失われないようにする必要がある。L3 キャッシュを無効または有効にした後、プロセッサ全体のキャッシュを復元できる。

10.5.5. キャッシュ管理命令

IA-32アーキテクチャでは、L1キャッシュ、L2キャッシュ、L3キャッシュを管理する命令をいくつか提供している。INVD、WBINVD、WBINVDの各命令は、L1キャッシュ、L2キャッシュ、L3キャッシュをまとめて操作するシステム命令である。SSEおよびSSE2の拡張命令で導入された非テンポラルな移動命令（MOVNTI、MOVNTQ、MOVNTDQ、MOVNTPS、MOVNTPD）、CLFLUSH命令、PREFETCH*h*命令を使用すると、キャッシングをよりきめ細かに制御できる。

INVD命令とWBINVD命令を使用して、L1キャッシュ、L2キャッシュ、L3キャッシュの内容を無効にする。INVD命令は、内部キャッシュのエントリすべてを無効にし、特殊機能のバスサイクルを生成する。このバスサイクルは、外部キャッシュも無効にするように指示する。INVD命令は注意して使用する必要がある。INVD命令は、修正されたキャッシュ・ラインのライトバックを強制しないため、キャッシュ内にストアされたデータと、システムメモリにライトバックされていないデータが失われる。修正されたラインをライトバックしないでキャッシュを無効にしなければならない明確な要求事項や利点がない限り（例えばテストの実行時や障害処理時のように、キャッシュとメインメモリとのコヒーレンシが重要な問題ではない場合は）、ソフトウェアはWBINVD命令を使用しなければならない。

まずWBINVD命令は、あらゆる内部キャッシュ内にある修正済みラインをライトバックし、次にL1キャッシュ、L2キャッシュ、L3キャッシュの内容を無効にする。これによって、実施されている書き込み方式に関係なく（つまり、ライトスルーかライトバックかに関係なく）、キャッシュとメインメモリとのコヒーレンシが確実に維持される。この動作の後、WBINVD命令は特殊機能バスサイクルを1つ（P6ファミリ・プロセッサの場合）または2つ（インテル® Pentium® プロセッサとIntel486™プロセッサの場合）生成して、外部キャッシュ・コントローラに、修正されたデータのライトバックに続いて外部キャッシュを無効にするように指示する。

PREFETCH*h*命令を使用すると、プログラムからプロセッサに対し、システムメモリ内の指定されたロケーションからのキャッシュ・ラインをキャッシュ階層にプリフェッチするよう指示することができる（10.8.節「明示的キャッシング」を参照）。

CLFLUSH命令を使用すると、選択したキャッシュ・ラインをメモリからフラッシュできる。この命令によりプログラムは、システムメモリのキャッシュされた部分がしばらくアクセスされないことが明らかになった時点で、このキャッシュ空間を明示的に開放できる。

非テンポラルな移動命令（MOVNTI、MOVNTQ、MOVNTDQ、MOVNTPS、MOVNTPD）を使用すると、L1キャッシュ、L2キャッシュ、L3キャッシュへの書き込みを行わずに、データを直接プロセッサのレジスタからシステムメモリに移動できる。これらの命令を利用すれば、システムメモリにストアされる前に一度だけ変更されるデータを

操作するときに、キャッシュへの書き込みを防止できる。これらの命令は、汎用レジスタ、MMX® テクノロジ・レジスタ、XMM レジスタ内のデータを操作する。

10.5.6. L1 データ・キャッシュ・コンテキスト・モード

1 次データ・キャッシュ・コンテキスト・モードは、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 プロセッサの機能である。EAX=1 で CPUID 命令を実行した後にコンテキスト ID 機能フラグ（ECX[10]）がセットされた場合、そのプロセッサは、IA32_MISC_ENABLE MSR を使用した L1 データ・キャッシュ・コンテキスト・モードの設定をサポートしている。選択可能なモードは、アダプティブ・モード（デフォルト）と共有モードである。

BIOS は、L1 データ・キャッシュ・コンテキスト・モードを設定する役割を受け持つ。

10.5.6.1. アダプティブ・モード

アダプティブ・モードでは、ページ・ディレクトリを使用してアクセスされるメモリは、同じ L1 データ・キャッシュを共有する複数の論理プロセッサ上で同じようにマッピングされる。マッピングが同じであるため、各論理プロセッサは、ターゲット・キャッシュを（競争的に共有するのではなく）フルサイズとして認識する。

同じ L1 データ・キャッシュを共有する複数の論理プロセッサ上で、CR3 レジスタが同じ設定になっている場合は、L1 データ・キャッシュはアダプティブ・モード機能を利用する。L1 データ・キャッシュがアダプティブ・モードに設定されているにもかかわらず、同じ L1 データ・キャッシュを共有する複数の論理プロセッサ間で CR3 レジスタの設定が異なる場合は、各論理プロセッサは L1 データ・キャッシュのリソースを求めて競合する。この場合は、各論理プロセッサはキャッシュをフルサイズとして認識しない。

10.5.6.2. 共有モード

共有モードでは、L1 データ・キャッシュは競争的に共有される。同じ L1 データ・キャッシュを共有する複数の論理プロセッサ上で CR3 レジスタが同じ設定になっている場合でも、共有モードでは L1 データ・キャッシュは競争的に共有される。

10.6. 自己修正コード

プロセッサに現在キャッシュされているコード・セグメント内にあるメモリ位置への書き込みが行われると、対応するキャッシュ・ライン（1つまたは複数）が無効になる。このチェックは命令の物理アドレスに基づいて行われる。さらに、P6ファミリ・プロセッサとインテル® Pentium® プロセッサは、コード・セグメントへの書き込みが、実行用にプリフェッチされていた命令を修正するかどうかをチェックする。プリフェッチされていた命令が書き込みの影響を受ける場合は、プリフェッチ・キューが無効になる。このチェックは、命令のリニアアドレスに基づいて行われる。インテル® Pentium® 4プロセッサおよびインテル® Xeon™ プロセッサでは、ターゲット命令がすでにデコードされ、トレース・キャッシュに常駐している場合に、コード・セグメント内の命令を書き込みまたはスヌープすると、トレース・キャッシュ全体が無効化される。後者の動作は、インテルPentium 4プロセッサおよびインテルXeonプロセッサ上で動作してコードを自己修正するプログラムが原因であり、パフォーマンスが大幅に低下する可能性があることを意味する。

実際問題として、リニアアドレスに対するチェックによって、IA-32プロセッサ間での互換性にいろいろな問題が生じることがあってはならない。自己修正コードが入っているアプリケーションは、同じリニアアドレスを使用して命令の修正やフェッチを行う。デバッガなどのシステム・ソフトウェアは、命令をフェッチするためのリニアアドレスとは異なるリニアアドレスを使用して、できる限り命令を修正することになる。そして修正された命令が実行される前に、CPUID命令のようなシリアル化動作を実行する。これによって、命令キャッシュとプリフェッチ・キューが自動的に再同期化される。（自己修正コードの詳細については、7.1.3.項「自己修正コードおよびクロス修正コードの処理」を参照。）

Intel486™ プロセッサでは、キャッシュ内の命令への書き込みによって、キャッシュとメモリの両方にある命令が修正される。ただし書き込みの前に命令がプリフェッチされていた場合は、命令の古いバージョンが実行されてしまうことがある。これを防ぐために、命令を修正する書き込みのすぐ後にジャンプ命令をコーディングして、命令プリフェッチ・ユニットをフラッシュする。

10.7. 暗黙的キャッシング（インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ）

たとえメモリ・エレメントが通常のノイマン型シーケンスでアクセスされたことがなくても、そのメモリ・エレメントが潜在的にキャッシュ可能となっている場合は、暗黙的キャッシングが発生する。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ上では、アグレッシブなプリフェッチ、分岐予測および TLB のミス・ハンドリングが原因で発生する。暗黙的キャッシングは、既存の Intel386™ プロセッサ、Intel486™ プロセッサ、インテル® Pentium® プロセッサ・システムの動作を拡張した機能である。つまり、これらのプロセッサ・ファミリで実行されるソフトウェアが命令プリフェッチの動作を確実に予測できなかったため、この暗黙的キャッシングが拡張機能として付加された。

暗黙的キャッシングに付随する問題を避けるため、キャッシュのコヒーレンシ・メカニズムが自動的に処理できない変更がキャッシュ可能データに加えられた場合は、オペレーティング・システムがそのキャッシュを明示的に無効にする必要がある。これには、プロセッサのスヌープ機構が検出できないデュアルポートを持つメモリボードや、物理的に別名が付けられたメモリボードへの書き込み、メモリ内のページ・テーブル・エントリへの変更が含まれる。

例 10-1. のコードでは、ページ・テーブル・エントリに対する暗黙的キャッシングの影響を示している。リニアアドレス F000H は物理位置 B000H を指す（F000H のページ・テーブル・エントリには B000H がストアされる）。またリニアアドレス F000 のページ・テーブル・エントリは PTE_F000 である。

例 10-1. 暗黙的キャッシングがページ・テーブル・エントリに及ぼす影響

```
mov EAX, CR3          ; Invalidate the TLB
mov CR3, EAX          ; by copying CR3 to itself
mov PTE_F000, A000H  ; Change F000H to point to A000H
mov EBX, [F000H];
```

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは実行が推論的になされるため、最後に実行された MOV 命令は、新しい物理アドレス A000H の値ではなく、物理位置 B000H の値を EBX に配置することになる。この状況は、ロードとストア間で TLB を無効化することによって矯正される。

10.8. 明示的キャッシング

インテル® Pentium® IIIプロセッサでは、データのキャッシングを明示的に制御するソフトウェアを提供する4つの命令、PREFETCH*h*命令が新たに追加された。これらの命令は、PREFETCH*h*命令によって要求されたデータを、その使用を見越して、直ちにまたはできるだけ早くキャッシュ階層内に読み込むように指示するヒントをプロセッサに提示するものである。これらの命令は、各種バリエーションのヒントを提供する。このヒントにより、データを読み取るキャッシュ・レベルを選択することができる。

一般に、PREFETCH*h*命令は、メモリからのデータ読み取りに伴う長い待ち時間を減らすのに役立つ。したがって、プロセッサの「ストール」の防止に役立つことになる。ただし、PREFETCH*h*命令は慎重に使用する必要がある。これを使い過ぎるとリソースの競合が引き起こされるため、アプリケーションのパフォーマンスが低下する。また、PREFETCH*h*命令は、データをメモリからプリフェッチするためだけに使用する必要がある。命令をプリフェッチするのには使用してはならない。プリフェッチ命令の正しい使い方については、『インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ最適化リファレンス・マニュアル』（資料番号については、1.4. 節「参考文献」を参照）の第6章「インテル® Pentium® 4プロセッサのキャッシュ利用率の最適化」を参照のこと。

10.9. トランスレーション・ルックアサイド・バッファ（TLB）の無効化

プロセッサは、それ自身のアドレスのトランスレーション・キャッシュ（TLB）をソフトウェアに透過的に更新する。ただしいくつかのメカニズムを使用すると、ソフトウェアやハードウェアがTLBを明示的に無効にするか、または別の動作の副作用として無効にできる。

INVLPG命令は、特定のページに対するTLBを無効にする。この命令は、ソフトウェアが特定のページの無効化だけを必要とする場合に、最も効果を発揮する。なぜなら、この命令によって、TLB全体を無効にする動作より性能が向上するからである。この命令は、ページ・ディレクトリまたはページ・テーブル・エントリ内のGフラグの状態による影響を受けない。

以下の各動作は、グローバル・エントリを除くすべてのTLBエントリを無効にする。（グローバル・エントリとは、対応するページ・ディレクトリとページ・テーブル・エントリ内でG（グローバル）フラグが設定されているエントリのことである。グローバル・フラグは、P6ファミリ・プロセッサのIA-32アーキテクチャに導入された。10.5. 節「キャッシュの制御」を参照。）

- 制御レジスタ CR3 への書き込み。
- 制御レジスタ CR3 を変更するタスクスイッチ。

以下の各動作は、G フラグの設定に関係なく、すべての TLB エントリを無効にする。

- FLUSH# ピンのアサートまたはデアサート。
- （インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのみ）MTRR への書き込み（WRMSR 命令を使用）。
- PG フラグまたは PE フラグを修正するための制御レジスタ CR0 への書き込み。
- （インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのみ）PSE、PGE、または PAE の各フラグを修正するための制御レジスタ CR4 への書き込み。

TLB の追加情報については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照。

10.10.ストアバッファ

IA-32 プロセッサは、メモリへの書き込みが発生するたびに、それをストアバッファに一時的にストアする。ストアバッファによって、プロセッサはメモリまたはキャッシュ（あるいはその両方）への書き込みが完了するのを待たずに命令の実行を継続できるため、プロセッサの性能が向上する。また書き込みバッファによって、メモリアクセスのバスサイクルをさらに効果的に使用するために、書き込みを遅延できる。

一般的には、ストアバッファの存在はソフトウェアに透過である。これは、マルチプロセッサを使用するシステムでも同様である。プロセッサによって、書き込み動作は常にプログラム順に実行される。また以下のような状態では、ストアバッファの内容が常にメモリに排出される。

- 例外または割り込みが生成された場合。
- インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのみ。シリアル化命令が実行された場合。
- I/O 命令が実行された場合。
- LOCK 動作が実行された場合。
- インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサのみ。BINIT 動作が実行された場合。
- インテル® Pentium® III プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサのみ。SFENCE 命令を使用して、ストアの順序を指定した場合。

- インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサのみ。MFENCE 命令を使用して、ストアの順序を指定した場合。

ストアバッファの動作の詳細については、7.2. 節「メモリ・オーダリング」でオーダリングについて説明している箇所を参照。

10.11. メモリタイプ範囲レジスタ（MTRR: Memory Type Range Registers）

この節では、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサに関連する情報だけを扱っている。

メモリタイプ範囲レジスタ（MTRR）では、メモリタイプ（10.3. 節「有効なキャッシングの方法」を参照）をシステムメモリの物理アドレス範囲と関連付けるためのメカニズムが用意されている。MTRR を使用すると、プロセッサは、異なるメモリタイプ（例えば、RAM、ROM、フレームバッファのメモリ、メモリ・マッピングされた I/O デバイス）に対する動作を最適化できる。また MTRR によって、システム・ハードウェアの設計も簡素化されている。つまり、初期の IA-32 プロセッサで同じ機能に使用されていたメモリ制御ピンや、これらのピンの起動に必要だった外部ロジックが排除されている。

MTRR メカニズムにより、物理メモリ内に最大 96 のメモリ範囲を定義できる。また、それぞれの範囲に入っているメモリタイプを指定するためのモデル固有レジスタ（MSR）のセットが定義されている。表 10-8. に、指定可能な各メモリタイプとそれぞれのプロパティを示している。図 10-3. では、MTRR による物理メモリのマッピングを示している。それぞれのメモリタイプの詳細については、10.3. 節「有効なキャッシングの方法」を参照。

ハードウェア・リセットの後、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサは、すべての固定 MTRR と可変 MTRR をディスエーブルにする。これによって事実上すべての物理メモリがキャッシュ不可になる。その後、初期化ソフトウェアは、MTRR をシステム定義された特定のメモリマップに設定する必要がある。一般的には、BIOS（基本入出力システム）ソフトウェアが MTRR を構成する。構成後、オペレーティング・システムまたはエグゼクティブは、通常のページレベルのキャッシュ可能属性を使用してメモリマップを自由に変更できる。

マルチプロセッサ・システムでは、種々のインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサが、必ず同一の MTRR のメモリマップを使用しなければならない。これは、ソフトウェアが、プログラムを実行しているプロセッサに左右されずに、メモリに対して一貫性を保つためである。

表 10-8. MTRR 内でエンコード可能なメモリタイプ

メモリタイプとニーモニック	MTRR 内のエンコーディング
キャッシュ不可（UC）	00H
ライト・コンバイニング（WC）	01H
予約済み *	02H
予約済み *	03H
ライトスルー（WT）	04H
ライト・プロテクト（WP）	05H
ライトバック（WB）	06H
予約済み *	7H ～ FFH

注 :

* これらのエンコーディングを使用すると、結果として一般保護例外（#GP）が生成される。

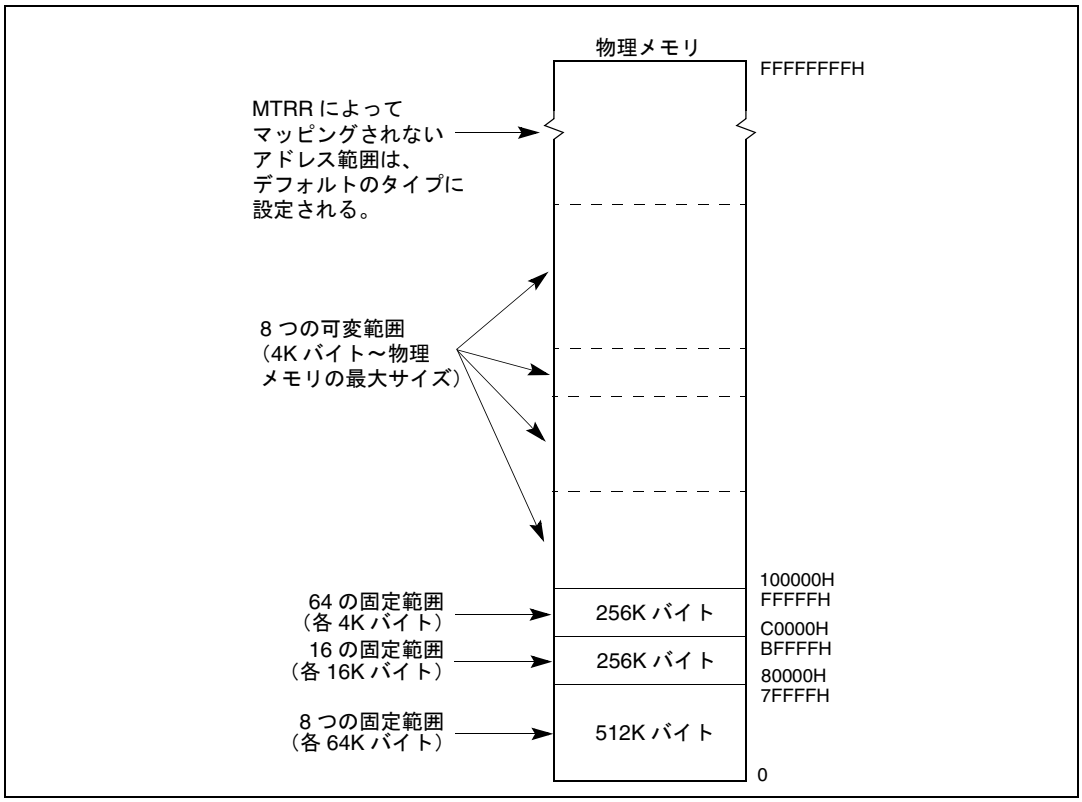


図 10-3. MTRR による物理メモリのマッピング

10.11.1. MTRR 機能の識別

MTRR機能の利用可能度はモデルごとに特定される。ソフトウェアは、CPUID命令を実行し、機能情報レジスタ（EDX）にあるMTRRフラグ（ビット12）の状態を読み取って、MTRRがプロセッサ上でサポートされるかどうかを判定する。

MTRRフラグが設定されている（つまりプロセッサがMTRRを保持していることを示している）場合、MTRRの追加情報を64ビットのIA32_MTRRCAP MSR（P6ファミリ・プロセッサではMTRRcap MSRという）から入手できる。IA32_MTRRCAP MSRは読み取り専用のMSRで、RDMSR命令によって読み取れる。図 10-4. ではIA32_MTRRCAP MSRの内容を示している。このレジスタの各フラグとフィールドの機能は以下のとおりである。

VCNT（可変範囲レジスタカウント）フィールド、ビット0～7

プロセッサに用意されている可変範囲の数を示す。インテル® Pentium® 4プロセッサ、インテル® Xeon™ プロセッサ、P6ファミリ・プロセッサには、8つの可変範囲をセットアップするためのMTRRが8組用意されている。

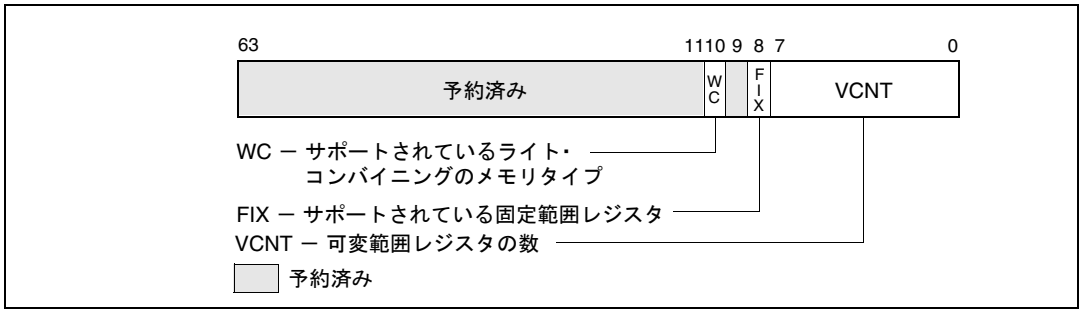


図 10-4. IA32_MTRRCAP レジスタ

FIX（サポートされている固定範囲レジスタ）フラグ、ビット8

このフラグが設定されている場合、固定範囲MTRR（IA32_MTRR_FIX64K_00000 ～ IA32_MTRR_FIX4K_0F8000）がサポートされる。クリアされている場合は、固定範囲レジスタはサポートされない。

WC（ライト・コンバイニング）フラグ、ビット10

このフラグが設定されている場合、ライト・コンバイニング（WC）のメモリタイプがサポートされる。クリアされている場合は、WCタイプはサポートされない。

IA32_MTRRCAP MSRのビット9と、ビット11～63は予約済みである。ソフトウェアがこれらのIA32_MTRRCAP MSRに書き込もうとすると、一般保護例外（#GP）が生成される。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、および P6 ファミリ・プロセッサでは、IA32_MTRRCAP MSR には常に値 508H がストアされる。

10.11.2. MTRR によるメモリ範囲の設定

それぞれの範囲で指定したメモリ範囲とメモリタイプは、3 つのグループのレジスタ（IA32_MTRR_DEF_TYPE MSR、固定範囲 MTRR、可変範囲 MTRR）によって設定される。RDMSR 命令と WRMSR 命令をそれぞれ使用すると、これらのレジスタの読み取りと書き込みができる。IA32_MTRRCAP MSR は、プロセッサ上でのこれらレジスタの利用可能度を示す（詳細は 10.11.1. 項「MTRR 機能の識別」を参照）。

10.11.2.1.IA32_MTRR_DEF_TYPE MSR レジスタ

IA32_MTRR_DEF_TYPE MSR（P6 ファミリ・プロセッサでは MTRRdefType MSR という）は、MTRR が包含していない物理メモリ領域のデフォルトのプロパティを設定する（図 10-5. を参照）。このレジスタの各フラグとフィールドの機能は以下のとおりである。

タイプ・フィールド、ビット 0 ～ 7

MTRR がメモリタイプを指定していない物理メモリアドレス範囲に使用されるデフォルトのメモリタイプを示す（このフィールドのエンコーディングについては、表 10-8. を参照）。MTRR がディスエーブルになっている場合は、このフィールドが物理メモリのすべてに対するメモリタイプを定義する。このフィールドの有効値は、0、1、4、5、6 である。これら以外の値を使用すると、結果として一般保護例外（#GP）が生成される。

メモリが存在しないすべての物理メモリアドレスには、UC（アンキャッシュド）メモリタイプの使用を推奨する。存在しないメモリ位置に UC タイプを割り当てるには、このメモリタイプをデフォルト・タイプとしてタイプ・フィールドに指定するか、または固定 MTRR と可変 MTRR を使用して明示的に割り当てるかのどちらかを実行する。

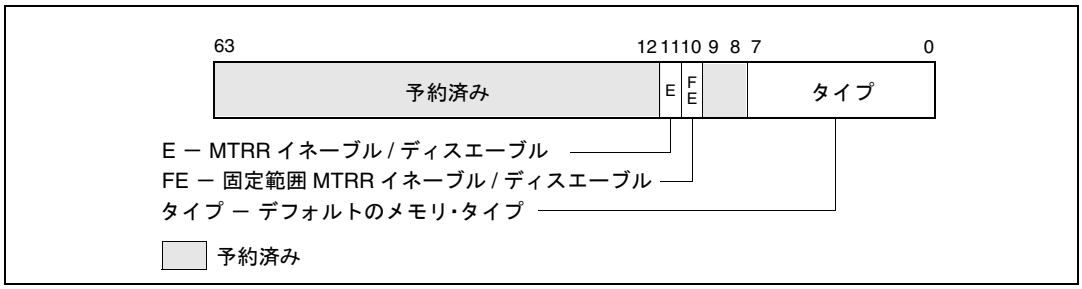


図 10-5. IA32_MTRR_DEF_TYPE MSR

FE（固定 MTRR イネーブル）フラグ、ビット 10

このフラグが設定されている場合、固定範囲 MTRR がイネーブルになり、クリアされている場合は、固定範囲 MTRR がディスエーブルになる。固定範囲 MTRR がイネーブルになっている場合に複数の範囲がオーバーラップすると、固定範囲 MTRR が可変範囲 MTRR に優先する。固定範囲 MTRR がディスエーブルになっている場合は、可変範囲 MTRR をそのまま使用して、固定範囲 MTRR が通常カバーしている範囲をマップできる。

E（MTRR イネーブル）フラグ、ビット 11

このフラグが設定されている場合、MTRR がイネーブルになる。クリアされた場合は、すべての MTRR がディスエーブルになり、UC メモリタイプが物理メモリのすべてに適用される。このフラグが設定されると、FE フラグは固定範囲 MTRR をディスエーブルにできる。クリアされると FE フラグは無効になる。E フラグが設定されると、デフォルトのメモリ・タイプ・フィールドで指定されているタイプが、固定 MTRR または可変 MTRR のどちらによってもマップされていないメモリ領域に使用される。

IA32_MTRR_DEF_TYPE MSR のビット 8 と 9、ビット 12 ～ 63 は予約済みである。ソフトウェアがこれらのビットにゼロ以外の値を書き込もうとすると、プロセッサが一般保護例外（#GP）を生成する。

10.11.2.2.固定範囲 MTRR

固定メモリ範囲は、それぞれが 64 ビットの長さを持つ 11 個の固定範囲レジスタを使用してマップされる。各固定範囲レジスタは、8 ビットのフィールドに分割され、レジスタが制御する各サブ範囲のメモリタイプを指定するために使用される。

- レジスタ IA32_MTRR_FIX64K_00000。0H ～ 7FFFFH に 512K バイトのアドレス範囲をマップする。この範囲は、64K バイトの容量を持つ 8 つのサブ範囲に分割される。
- レジスタ IA32_MTRR_FIX16K_80000 と IA32_MTRR_FIX16K_A0000。80000H ～ BFFFFH に 2 つの 128K バイトのアドレス範囲をマップする。この範囲は、16K バイトの容量を持つ 16 のサブ範囲に分割される（1 つのレジスタごとに 8 つの範囲）。
- レジスタ IA32_MTRR_FIX4K_C0000 から IA32_MTRR_FIX4K_F8000。C0000H ～ FFFFFH に 8 つの 32K バイトのアドレス範囲をマップする。この範囲は、4K バイトの容量を持つ 64 のサブ範囲に分割される（1 つのレジスタごとに 8 つの範囲）。

表 10-9. では、固定物理アドレス範囲と、それに対応する固定範囲 MTRR のフィールドとの関係を示している。また表 10-8. では、これらのフィールドの可能なエンコーディングを示している。

P6 ファミリ・プロセッサでは、固定範囲 MTRR に対するプリフィックスは MTRRfix である。

10.11.2.3.可変範囲 MTRR

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサを使用すると、ソフトウェアが 8 つの可変サイズアドレス範囲にメモリタイプを指定できる。この場合、それぞれの範囲にひと組の MTRR を使う。各組の最初の MTRR（IA32_MTRR_PHYSBASE*n*）は、範囲のベースアドレスとメモリタイプを定義し、2 番目の MTRR（IA32_MTRR_PHYSMASK*n*）は、アドレス範囲の決定に使用されるマスクをストアする。接尾辞の「*n*」は、0 から 7 までのレジスタの組を示す。

P6 ファミリ・プロセッサの場合、可変範囲 MTRR のプリフィックスは、MTRR-physBase と MTRRphysMask である。

表 10-9. 固定範囲 MTRR に対するアドレス・マッピング

アドレス範囲（16 進）								MTRR
63　56	55　48	47　40	39　32	31　24	23　16	15　8	7　0	
70000-7FFFF	60000-6FFFF	50000-5FFFF	40000-4FFFF	30000-3FFFF	20000-2FFFF	10000-1FFFF	00000-0FFFF	IA32_MTRR_FIX64K_00000
9C000 9FFFF	98000-98FFF	94000-97FFF	90000-93FFF	8C000-8FFFF	88000-8BFFF	84000-87FFF	80000-83FFF	IA32_MTRR_FIX16K_80000
BC000 BFFFF	B8000-BBFFF	B4000-B7FFF	B0000-B3FFF	AC000-AFFFF	A8000-ABFFF	A4000-A7FFF	A0000-A3FFF	IA32_MTRR_FIX16K_A0000
C7000 C7FFF	C6000-C6FFF	C5000-C5FFF	C4000-C4FFF	C3000-C3FFF	C2000-C2FFF	C1000-C1FFF	C0000-C0FFF	IA32_MTRR_FIX4K_C0000
CF000 CFFFF	CE000-CEFFF	CD000-CDFFF	CC000-CCFFF	CB000-CBFFF	CA000-CAFFF	C9000-C9FFF	C8000-C8FFF	IA32_MTRR_FIX4K_C8000
D7000 D7FFF	D6000-D6FFF	D5000-D5FFF	D4000-D4FFF	D3000-D3FFF	D2000-D2FFF	D1000-D1FFF	D0000-D0FFF	IA32_MTRR_FIX4K_D0000
DF000 DFFFF	DE000-DEFFF	DD000-DDFFF	DC000-DCFFF	DB000-DBFFF	DA000-DAFFF	D9000-D9FFF	D8000-D8FFF	IA32_MTRR_FIX4K_D8000
E7000 E7FFF	E6000-E6FFF	E5000-E5FFF	E4000-E4FFF	E3000-E3FFF	E2000-E2FFF	E1000-E1FFF	E0000-E0FFF	IA32_MTRR_FIX4K_E0000
EF000 EFFFF	EE000-EEFFF	ED000-EDFFF	EC000-ECFFF	EB000-EBFFF	EA000-EAFFF	E9000-E9FFF	E8000-E8FFF	IA32_MTRR_FIX4K_E8000
F7000 F7FFF	F6000-F6FFF	F5000-F5FFF	F4000-F4FFF	F3000-F3FFF	F2000-F2FFF	F1000-F1FFF	F0000-F0FFF	IA32_MTRR_FIX4K_F0000
FF000 FFFFF	FE000-FEFFF	FD000-FDFFF	FC000-FCFFF	FB000-FBFFF	FA000-FAFFF	F9000-F9FFF	F8000-F8FFF	IA32_MTRR_FIX4K_F8000

図 10-6. では、これらのレジスタのフラグとフィールドを示している。これらのレジスタのフラグとフィールドの機能は以下のとおりである。

タイプ・フィールド、ビット 0 ～ 7

範囲のメモリタイプを指定する（このフィールドのエンコーディングについては、表 10-8. を参照）。

PhysBase フィールド、ビット 12 ～ 35

アドレス範囲のベースアドレスを指定する。この 24 ビット値は、ベースアドレスを形成するために下位で 12 ビットだけ拡張され、4K バイト境界にアドレスのアライメントが自動的に合わされる。

PhysMask フィールド、ビット 12 ～ 35

マッピングが行われる範囲を決定する 24 ビットのマスクを、次の関係式にしたがって指定する。

Address_Within_Range AND PhysMask = PhysBase AND PhysMask

この 24 ビット値は、マスク値を形成するために下位で 12 ビットだけ拡張される。ベースアドレスとマスクの計算例と詳細については、10.11.3. 項「ベースとマスクの計算例」を参照。

V（有効）フラグ、ビット 11

このフラグが設定されている場合、レジスタの組がイネーブルになる。クリアされている場合は、レジスタの組はディスエーブルになる。

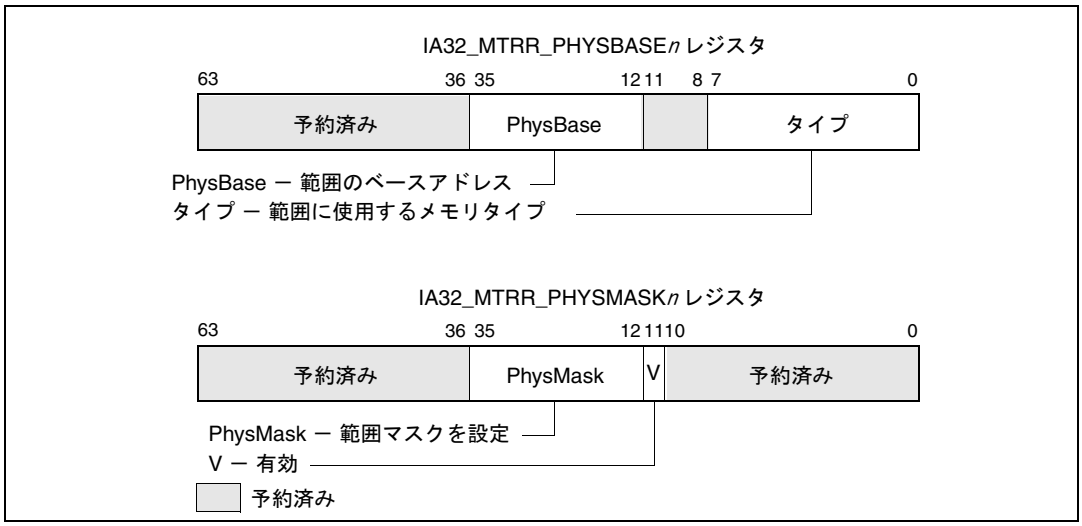


図 10-6. IA32_MTRR_PHYSBASE*n* と IA32_MTRR_PHYSMASK*n* の可変範囲レジスタの組

IA32_MTRR_PHYSBASE*n* レジスタと IA32_MTRR_PHYSMASK*n* レジスタの他のビットは、すべて予約済みである。ソフトウェアがこれらのビットに書き込もうとすると、プロセッサが一般保護例外（#GP）を生成する。

可変 MTRR 範囲のオーバーラップは、一般的にはサポートされない。ただし、以下の条件が満たされる場合は、2 つの可変範囲のオーバーラップが許可される。

- 範囲が両方とも UC（アンキャッシュド）であること。
- 1 つの範囲のタイプが UC で、もう一方の範囲のタイプが WB（ライトバック）であること。

上記の両方の場合とも、オーバーラップをする側の領域の有効なタイプは UC である。可変範囲をオーバーラップするその他の場合については、プロセッサの動作は定義されていない。

可変範囲は、（固定範囲 MTRR がイネーブルになっている場合）固定範囲をオーバーラップできる。この場合は、固定範囲レジスタ内で指定されているメモリタイプが、可変範囲レジスタの組で指定されているメモリタイプに優先する。

注記

マスク値によっては、不連続な範囲が発生することがある。不連続な範囲では、マスク値によってマップされていない領域は、デフォルトのメモリタイプに設定される。この範囲では、4G バイトの物理メモリマップ全体にわたって物理メモリが必要になるため、不連続な範囲を使用しないようにすることが望ましい。メモリマップ全体にメモリが用意されていない場合のプロセッサの動作は未定義である。

10.11.3. ベースとマスクの計算例

可変範囲 MTRR の組に入力されるベース値とマスク値は 24 ビットの値で、プロセッサによって 36 ビットに拡張される。例えば、IA32_MTRR_PHYSBASE3 レジスタに 2M バイト（200000H）のベースアドレスを入力する場合、下位 12 ビットが切り捨てられ、000200H の値が PhysBase フィールドに入力される。マスク値でも同じ動作が実行されなければならない。例えば、200000H ～ 3FFFFFH（2M バイト～ 4M バイト）のアドレス範囲をマップする場合、FFFE00000H のマスク値が必要になる。この場合でも、このマスク値の下位 12 ビットが切り捨てられ、IA32_MTRR_PHYSMASK3 レジスタの PhysMask フィールドに FFFFE00H の値が入力される。このマスクが選択されると、200000H ～ 3FFFFFH の範囲内にあるアドレスがマスク値と AND されたときに、ベースアドレスがマスク値（200000H）と AND された場合と同じ値が返されるようになる。

400000H ～ 7FFFFFH（4M バイト～ 8M バイト）のアドレス範囲をマップするには、000400H のベース値を PhysBase フィールドに入力し、FFFC00H のマスク値を PhysMask フィールドに入力する。

ここでは、システム全体にMTRRセットアップするときの実例を示す。システムには以下の特性があるものと想定する。

- 最高のシステム性能を得るため、96Mバイトのシステムメモリがライト・バック・メモリ（WB）としてマップされている。
- 4Mバイトのカスタム I/O カードが、64M バイトのベースアドレスで、アンキャッシュド・メモリ（UC）にマップされている。この制限によって、96Mバイトのシステムメモリの0～64Mバイトと68Mバイト～100Mバイトへのアドレス指定が強制される。また I/O カード用に4Mバイトの「穴」が残される。
- 8-M バイトのグラフィックス・カードが、A0000000H のアドレスから始まるライト・コンバイニング・メモリ（WC）メモリにマップされている。
- 15Mバイト～16Mバイトの BIOS 領域が、UC メモリにマップされている。

以下の MTRR の設定によって、このシステム・コンフィギュレーションに対する物理アドレス空間を適切にマッピングできる。

IA32_MTRR_PHYSBASE0 = 0000 0000 0000 0006H
IA32_MTRR_PHYSMASK0 = 0000 000F FC00 0800H
0 ～ 64M バイトを WB キャッシュ・タイプとしてキャッシュする。
IA32_MTRR_PHYSBASE1 = 0000 0000 0400 0006H
IA32_MTRR_PHYSMASK1 = 0000 000F FE00 0800H
64 ～ 96M バイトを WB キャッシュ・タイプとしてキャッシュする。
IA32_MTRR_PHYSBASE2 = 0000 0000 0600 0006H
IA32_MTRR_PHYSMASK2 = 0000 000F FFC0 0800H
96 ～ 100M バイトを WB キャッシュ・タイプとしてキャッシュする。
IA32_MTRR_PHYSBASE3 = 0000 0000 0400 0000H
IA32_MTRR_PHYSMASK3 = 0000 000F FFC0 0800H
64 ～ 68M バイトを UC キャッシュ・タイプとしてキャッシュする。
IA32_MTRR_PHYSBASE4 = 0000 0000 00F0 0000H
IA32_MTRR_PHYSMASK4 = 0000 000F FFF0 0800H
15 ～ 16M バイトを UC キャッシュ・タイプとしてキャッシュする。
IA32_MTRR_PHYSBASE5 = 0000 0000 A000 0001H
IA32_MTRR_PHYSMASK5 = 0000 000F FF80 0800H
A0000000 ～ A0800000 を WC キャッシュ・タイプとしてキャッシュする。

このMTRRセットアップでは、（範囲がWBとUCの両メモリ・タイプにマップされる限り）任意の2つのメモリ範囲をオーバーラップする機能を使用して、メモリ環境の設定に必要なMTRRレジスタの数を最小限に抑える。またこのセットアップでは、オペレーティング・システム用に2組のレジスタを残しておく必要条件も満たされる。

10.11.4. 範囲のサイズとアライメントの要件

可変範囲 MTRR にマッピングされる範囲は、以下の「2 の累乗」サイズとアライメント規則を満たすものでなければならない。

1. 最小範囲サイズは 4K バイトであり、またこの範囲のベースアドレスは少なくとも 4K バイト境界上になければならない。
2. 4K バイトを超える範囲については、それぞれの範囲は 2^n の長さを持たなければならない。またそのベースアドレスは 2^n の境界にアライメントを合わせなければならない。この場合の「*n*」は、12 以上の値である。ベースアドレスのアライメント値は、その長さより小さいものであってはならない。例えば、8K バイトの範囲は 4K バイトの境界にアライメントを合わせられない。この範囲は、少なくとも 8K バイトの境界にアライメントを合わせなければならない。

10.11.4.1.MTRR の優先

MTRR が（IA32_MTRR_DEF_TYPE MSR レジスタ内の E フラグを設定して）イネーブルになっていない場合、すべてのメモリアクセスは UC メモリタイプとなる。MTRR がイネーブルになっている場合は、メモリアクセスに使用されるメモリタイプが、以下のようにして決定される。

1. 物理アドレスが物理メモリの最初の 1M バイト以内にあり、そして固定 MTRR がイネーブルになっている場合は、プロセッサは、該当する固定範囲 MTRR 用にストアされているメモリタイプを使用する。
2. そうでない場合、プロセッサは以下のようにして、可変範囲 MTRR の組によって設定されたメモリタイプ範囲と物理アドレスとの突き合わせを試みる。
   a. 1 つの可変メモリ範囲が一致する場合、プロセッサは、その範囲用の IA32_MTRR_PHYSBASE*n* レジスタにストアされたメモリタイプを使用する。
   b. 2 つ以上の可変メモリ範囲が一致し、メモリタイプも同じである場合は、そのメモリタイプが使用される。
   c. 2 つ以上の可変メモリ範囲が一致して、メモリタイプの 1 つが UC である場合、UC メモリタイプが使用される。
   d. 2 つ以上の可変メモリ範囲が一致して、メモリタイプが WT と WB の場合、WT メモリタイプが使用される。
   e. 2 つ以上の可変メモリ範囲が一致して、メモリタイプが UC と WB 以外の場合、プロセッサの動作は未定義になる。
3. 一致する固定メモリ範囲や可変メモリ範囲が存在しない場合、プロセッサはデフォルトのメモリタイプを使用する。

10.11.5. MTRR の初期化

ハードウェア・リセットの時点で、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、または P6 ファミリ・プロセッサは可変範囲 MTRR 内の有効フラグをクリアし、またすべての MTRR をディスエーブルにするために IA32_MTRR_DEF_TYPE MSR レジスタ内の E フラグをクリアする。MTRR の他のビットはすべて未定義である。MTRR の初期化に先立って、ソフトウェア（通常はシステム BIOS）は、固定範囲 MTRR レジスタと可変範囲 MTRR レジスタのすべてのフィールドを 0 に初期化しなければならない。その後にソフトウェアは、既知のメモリのタイプにしたがって MTRR を初期化できる。このメモリには、自動コンフィギュレーションをするデバイスのメモリも含まれる。この初期化は、オペレーティング・システムをブートする前に実行されなければならない。

マルチプロセッサ（MP）システムでの MTRR の初期化に関する情報については、10.11.8. 項「MP システムでの MTRR の注意点」を参照。

10.11.6. メモリタイプの再マッピング

システム設計者は、性能を調整するためにメモリタイプのマップ変更を実行できる。または将来のプロセッサが、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサによってサポートされるすべてのメモリタイプをサポートしない可能性があるため、メモリタイプのマップ変更を実行できる。以下の規則を守ると、メモリタイプのマップ変更のコヒーレンシが保たれる。

1. メモリタイプは、それより弱いメモリ・オーダリング・モデルを持つ他のメモリタイプにマップしてはならない。例えば、キャッシュ不可のタイプは、他のどのタイプにもマップできない。またライトバック、ライトスルー、およびライト・プロテクトの各タイプは、ウィーク・オーダリングのライト・コンバイニング・タイプにマップできない。
2. 書き込みを遅延させないメモリタイプを、書き込みを遅延させるメモリタイプにマップしてはならない。なぜなら、このようなメモリタイプのアプリケーションが、そのライトスルーの動作に依存する可能性があるためである。したがって、ライト・バック・タイプをライト・スルー・タイプにマップできない。
3. 例えばライト・プロテクト・タイプのように、後続の読み取りによって必ずしもストアされたり読み戻される必要のない書き込みデータを表示するメモリタイプは、同じ動作を持つ別のタイプ（ただしインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサには別のタイプは存在しない）やキャッシュ不可のタイプにだけマップできる。

具体的にはほとんどの場合、システム設計者はメモリタイプが使用されている状態の追加情報を得られる。そしてこの情報を使用して、追加マッピングを実行できる。例

えば、関連する書き込みの副作用のないライト・スルー・メモリを、ライト・バック・メモリにマップできる。

10.11.7. MTRR メンテナンス用プログラミング・インターフェイス

オペレーティング・システムは、ブート後にMTRRをメンテナンスし、メモリマップされたデバイスのメモリタイプをセットアップするか、または変更する。オペレーティング・システムは、MTRRへのアクセスと設定をするドライバとアプリケーション・プログラミング・インターフェイス（API）を規定しなければならない。MemTypeGet() と MemTypeSet() の 2 つの関数を呼び出すことによって、このインターフェイスが定義される。

10.11.7.1.MemTypeGet() 関数

MemTypeGet() 関数は、パラメータのベースとサイズで指定された物理メモリ範囲のメモリタイプを返す。ベースアドレスは開始物理アドレスになり、サイズはメモリ範囲のバイト数になる。この関数は自動的に 4K バイト境界にベースアドレスとサイズのアライメントを合わせる。MemTypeGet() 関数の疑似コードを例 10-2. でとりあげている。

例 10-2. MemTypeGet() 関数の疑似コード

```
#define MIXED_TYPES -1     /* 0 < MIXED_TYPES || MIXED_TYPES > 256 */

IF CPU_FEATURES.MTRR /* processor supports MTRRs */
    THEN
        Align BASE and SIZE to 4-KByte boundary;
        IF (BASE + SIZE) wrap 4-GByte address space
            THEN return INVALID;
        FI;
        IF MTRRdefType.E = 0
            THEN return UC;
        FI;
        FirstType ← Get4KMemType (BASE);
        /* Obtains memory type for first 4-KByte range */
        /* See Get4KMemType (4KByteRange) in 例 10-3. */
        FOR each additional 4-KByte range specified in SIZE
            NextType ← Get4KMemType (4KByteRange);
            IF NextType ≠ FirstType
                THEN return MixedTypes;
            FI;
        ROF;
        return FirstType;
    ELSE return UNSUPPORTED;
FI;
```

プロセッサがMTRRをサポートしていない場合、この関数はUNSUPPORTEDを返す。MTRRがイネーブルになっていない場合は、UCメモリタイプが返される。指定された範囲に複数のメモリタイプが対応している場合は、MIXED_TYPESの状態が返される。それ以外の場合は、範囲に対して定義されたメモリタイプ（UC、WC、WT、WB、またはWP）が返される。

例 10-3.に示されているGet4KmemType()関数の疑似コードは、所定の物理アドレスにある単一の4Kバイト範囲に対するメモリタイプを入手する。このサンプルコードは、PHY_ADDRESSを既知の固定範囲［0～7FFFFH（64Kバイト領域）、80000H～BFFFFH（16Kバイト領域）、C0000H～FFFFFH（4Kバイト領域）］と比較して、このアドレスが固定範囲内にあるかどうかを判定する。あるアドレスが上記のどれかの範囲内にある場合は、そのMTRRのどれかの中にある適切なビットがメモリタイプを決定する。

例 10-3. Get4KMemType() 関数の疑似コード

```
IF IA32_MTRRCAP.FIX AND MTRRdefType.FE /* fixed registers enabled */
    THEN IF PHY_ADDRESS is within a fixed range
        return IA32_MTRR_FIX.Type;
FI;
FOR each variable-range MTRR in IA32_MTRRCAP.VCNT
    IF IA32_MTRR_PHYSMASK.V = 0
        THEN continue;
    FI;
    IF (PHY_ADDRESS AND IA32_MTRR_PHYSMASK.Mask) =
(IA32_MTRR_PHYSBASE.Base
            AND IA32_MTRR_PHYSMASK.Mask)
        THEN
            return IA32_MTRR_PHYSBASE.Type;
    FI;
ROF;
return MTRRdefType.Type;
```

10.11.7.2.MemTypeSet() 関数

例 10-4.のMemTypeSet()関数は、パラメータのベースとサイズで指定された物理メモリ範囲のMTRRを、Typeで指定されたタイプに設定する。ベースアドレスとサイズは4Kバイトの倍数であり、サイズは0以外の値となる。

例 10-4. MemTypeSet 関数の疑似コード

```
IF CPU_FEATURES.MTRR (* processor supports MTRRs *)
    THEN
        IF BASE and SIZE are not 4-KByte aligned or size is 0
            THEN return INVALID;
        FI;
        IF (BASE + SIZE) wrap 4-GByte address space
```

```
            THEN return INVALID;
        FI;
        IF TYPE is invalid for Pentium 4 and P6 family processors
            THEN return UNSUPPORTED;
        FI;
        IF TYPE is WC and not supported
            THEN return UNSUPPORTED;
        FI;
        IF IA32_MTRRCAP.FIX is set AND range can be mapped using a fixed-range
        MTRR
            THEN
                pre_mtrr_change();
                update affected MTRR;
                post_mtrr_change();
        FI;

    ELSE (* try to map using a variable MTRR pair *)
        IF IA32_MTRRCAP.VCNT = 0
            THEN return UNSUPPORTED;
        FI;
        IF conflicts with current variable ranges
            THEN return RANGE_OVERLAP;
        FI;
        IF no MTRRs available
            THEN return VAR_NOT_AVAILABLE;
        FI;
        IF BASE and SIZE do not meet the power of 2 requirements for variable MTRRs
            THEN return INVALID_VAR_REQUEST;
        FI;
        pre_mtrr_change();
        Update affected MTRRs;
        post_mtrr_change();
FI;

pre_mtrr_change()
    BEGIN
        disable interrupts;
        Save current value of CR4;
        disable and flush caches;
```

```
        flush TLBs;
        disable MTRRs;
        IF multiprocessing
            THEN maintain consistency through IPIs;
        FI;
    END
post_mtrr_change()
    BEGIN
        flush caches and TLBs;
        enable MTRRs;
        enable caches;
        restore value of CR4;
        enable interrupts;
    END
```

MemTypeSet 関数内の、可変範囲に対する物理アドレス・マッピングのアルゴリズムは、現行の可変範囲レジスタとの衝突を検出する。この検出は、現行の可変範囲レジスタを循環し、該当する物理アドレスが現行の範囲のどれかに一致するかどうかを判定することで行われる。このスキャン中に、アルゴリズムは、現行の可変範囲のどれかがオーバーラップしていないかどうか、または単一の範囲に連結できるかどうかを検出できる。

pre_mtrr_change() 関数は、MTRR の変更に先立って割り込みをディスエーブルにして、部分的に有効な MTRR のセットアップによってコードが実行されないようにする。アルゴリズムは、制御レジスタ CR0 内の CD フラグを設定し、NW フラグをクリアしてキャッシングをディスエーブルにする。キャッシュは、WBINVD 命令によって無効にされる。必要であれば、アルゴリズムは制御レジスタ CR4 内のページ・グローバル・フラグ（PGE）をディスエーブルにし、その後制御レジスタ CR3 を更新してすべての TLB エントリをフラッシュする。最後に、アルゴリズムは IA32_MTRR_DEF_TYPE MSR 内の E フラグをクリアして MTRR をディスエーブルにする。

メモリタイプが更新された後は、post_mtrr_change() 関数が MTRR をイネーブルしなおして、再度キャッシュと TLB を無効にする。この 2 度目の無効化が必要なのは、プロセッサが命令とデータの両方をアグレッシブにプリフェッチするからである。アルゴリズムは CD フラグを設定すれば、割り込みを復元して、キャッシングをイネーブルしなおす。

オペレーティング・システムは複数の MTRR 更新をバッチできる。その結果、キャッシュ無効化が一組だけ発生する。

10.11.8. MP システムでの MTRR の注意点

MP（マルチプロセッサ）システムでは、オペレーティング・システムは、マルチプロセッサ・システム内にあるすべてのプロセッサ間でMTRR のコンシステンシを維持する必要がある。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサには、このコンシステンシを維持するハードウェアのサポートは用意されていない。一般的には、すべてのプロセッサが同じMTRR 値を持っている必要がある。

この必要条件が暗黙的に表しているのは、オペレーティング・システムが MP システムを初期化するとき、MTRRdefType レジスタのE フラグが 0 になっている間にブート・プロセッサのMTRR をロードしなければならないという条件である。その後オペレーティング・システムは、他のプロセッサに対して、同じメモリマップをそのプロセッサ固有のMTRR にロードするように指示をする。すべてのプロセッサが固有のMTRR をロードし終わった後、オペレーティング・システムは、各プロセッサに対してそのMTRR をイネーブルにするように信号で指示する。MTRR がイネーブルになっていることをすべてのプロセッサが示すまで、他のメモリアクセスが行われないようにバリア同期が使用される。この同期は、共有変数とプロセッサ間割込みを持つシュートダウン・スタイルのアルゴリズムとなる可能性が高い。

MP システム内で MTRR の値に変更を加えると、オペレーティング・システムはロードとイネーブルのプロセスを繰り返して一貫性を維持する必要がある。これには以下の操作手順を使用する。

1. すべてのプロセッサに同時通信して、次に続くコード・シーケンスを実行させる。
2. 割り込みをディスエーブルにする。
3. すべてのプロセッサが上記の状態になるまで待つ。
4. 非フィル・キャッシュ・モードに入る。（制御レジスタ CR0 の CD フラグを 1 に、NW フラグを 0 に設定する。）
5. WBINVD 命令を使用して、すべてのキャッシュをフラッシュする。セルフ・スヌーピング（CPUID 機能、フラグビット 27）をサポートしているプロセッサでは、この手順は不要であることに注意する。
6. 制御レジスタ CR4 の PGE フラグが設定されている場合は、それをクリアする。
7. すべての TLB をフラッシュする。（制御レジスタ CR3 から別のレジスタへ移動するためのMOV を実行し、その後で、そのレジスタからCR3 へ戻るためのMOV を実行する。）
8. （MTRRdefType レジスタのE フラグをクリアして）すべての範囲レジスタをディスエーブルにする。可変範囲だけが修正される場合、ソフトウェアは、影響を受けたレジスタの組に対する有効ビットを代わりにクリアできる。
9. MTRR を更新する。

10.（MTRRdefType レジスタの E フラグを設定して）すべての範囲レジスタをイネーブルにする。可変範囲レジスタだけが修正されていて、レジスタの個々の有効ビットがクリアされている場合は、影響を受けた範囲に対する有効ビットを代わりに設定する。

11. 再度、すべてのキャッシュとすべての TLB をフラッシュする。（TLB のフラッシュは、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサに必須である。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサを使用している場合は WBINVD 命令を実行する必要はないが、将来のシステムでは必要になる可能性がある。）

12. 通常のキャッシュ・モードに移行して、キャッシングをイネーブルにしなおす。（制御レジスタ CR0 の CD フラグと NW フラグを、0 に設定する。）

13. 制御レジスタ CR4 の PGE フラグがクリアされていた場合は、PGE フラグを設定する。

14. すべてのプロセッサが上記の状態になるまで待つ。

15. 割り込みをイネーブルにする。

10.11.9. ラージ・ページ・サイズの注意点

MTRR は、4K バイトのグラニュラリティ（4K バイトのページと同じグラニュラリティ）を持つ制限された数の領域に対して、メモリタイプの指定を行う。特定のページのメモリタイプは、プロセッサの TLB 内でキャッシュされる。ラージページ（2M バイトまたは 4M バイト）を使用する場合は、単一のページ・テーブル・エントリには、複数の 4K バイトのグラニュールが記載される。またそれぞれに単一のメモリタイプが付随する。ラージページのメモリタイプは TLB 内でキャッシュされるため、MTRR が複数のメモリタイプによってマッピングしたメモリの領域にラージページがマップされた場合、プロセッサは、未定義のままで動作する可能性がある。

未定義の動作を避けるには、ラージページ内にあるすべての MTRR メモリタイプ範囲が確実に同じタイプになるように処理する。MTRR によって定義された別のメモリタイプをストアしているメモリ領域にラージページをマップする場合は、その範囲に対して最も保存性のあるメモリタイプに、ページ・テーブル・エントリ内の PCD フラグと PWT フラグを設定しなければならない。例えば、メモリ・マッピングされた I/O と通常のメモリに使用されるラージページは、UC メモリとしてマップされる。もう 1 つの方法として、オペレーティング・システムは、ページごとに固有のメモリタイプを持つ複数の 4K バイトのページを使用して、領域をマップできる。

ラージページ内の 4K バイトの範囲すべてが同じメモリタイプでなければならない必要条件が暗黙的に示しているのは、メモリタイプが異なっているラージページを使用すると、最少公分母のメモリタイプを使用してラージページをマーク付けしなければならないため、性能を犠牲にしてしまう可能性があることである。

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサは、固定 MTRR と可変 MTRR の両方によってマップされる可能性のある 0 ～ 4M バイトの物理メモリ範囲を特別にサポートできる。このサポート機能が起動するのは、固定 MTRR と矛盾しているメモリタイプにこの物理メモリ範囲の最初の 1M バイトをオーバーラップさせているラージページを、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサが検出したときである。このときプロセッサは、複数の 4K バイト・ページとしてメモリ範囲を TLB 内にマップする。この動作によって正常な動作が保証されるが、性能は犠牲になる。この性能上の犠牲を避けるには、4M バイト以上のアドレスにあるメモリ領域のラージ・ページ・オプションを、オペレーティング・システムのソフトウェアで予約しなければならない。

10.12. ページ属性テーブル（PAT）

ページ属性テーブル（PAT）は、IA-32 アーキテクチャのページ・テーブル・フォーマットを拡張したものである。リニア・アドレス・マッピングに基づいて、各メモリタイプを物理メモリ領域に割り当てることが可能である。PAT は、MTRR に対するコンパニオン機能である。すなわち、MTRR では物理アドレス空間の領域にメモリタイプをマップでき、PAT ではリニアアドレス空間内のページにメモリタイプをマップできる。MTRR は物理範囲のメモリタイプを静的に記述する際に役立ち、通常は、システム BIOS によってセットアップされる。PAT は、ページテーブル内の PCD ビットおよび PWT ビットの機能を拡張したものである。PAT では、MTRR で割り当てが可能な 5 つのメモリタイプのすべて（および、それに加えてもう 1 つの追加メモリタイプ）を、リニアアドレス空間内のページに動的に割り当てることも可能である。

PAT は、インテル® Pentium® III プロセッサで IA-32 アーキテクチャに導入された。これは、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでも使用できる。

注記

マルチプロセッサ・システムでは、オペレーティング・システムは、システム内のすべてのプロセッサの MTRR の整合性を保たなければならない（つまり、すべてのプロセッサは同じ MTRR 値を使用しなければならない）。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサは、この整合性の確保についてはハードウェア的にサポートしていない。

10.12.1. PAT 機能のサポートの検出

オペレーティング・システムやエグゼクティブでは、EAX レジスタに 1 をセットして CPUID 命令を実行することにより、PAT が使用可能かどうかを検出する。PAT のサポー

トの有無は、PAT フラグ（EDX レジスタに返される値のビット 16）で示される。PAT がサポートされている場合、オペレーティング・システムやエグゼクティブでは、IA32_CR_PAT MSR を使用して PAT をプログラムできる。メモリタイプが PAT 内のエントリに割り当てられている場合、ソフトウェア上で、ページ・テーブル・エントリおよびページ・ディレクトリ・エントリ内の PAT インデックス・ビット（PAT）と、PCD ビットおよびPWT ビットを一緒に使用して、メモリタイプを PAT から個別のページに割り当てられる。

ただし、PAT 機能をイネーブルにする特別なフラグや制御ビットは、どの制御レジスタにも存在しない。プロセッサが PAT をサポートしている場合は、PAT は常にイネーブルになる。また、ページングがイネーブルになっている場合は、すべてのページング・モードで、必ずページ属性テーブルの検索が行われる。

10.12.2. IA32_CR_PAT MSR

IA32_CR_PAT MSR は、MSR アドレス 277H に配置されている（付録 B「モデル固有レジスタ（MSR）」を参照）。このアドレスは、PAT 機能をサポートする将来の IA-32 プロセッサでも変更されることはない。図 10-7. は、64 ビット IA32_CR_PAT MSR のフォーマットを示している。

IA32_CR_PAT MSR には、PA0 ～ PA7 の 8 つのページ属性フィールドが含まれている。各フィールドの下位 3 ビットは、メモリタイプの指定に使用される。各フィールドの上位 5 ビットは予約されており、すべて 0 にセットする必要がある。これら 8 つのページ属性フィールドには、表 10-10. で示されるメモリ・タイプ・エンコーディングのいずれかが入れられる。

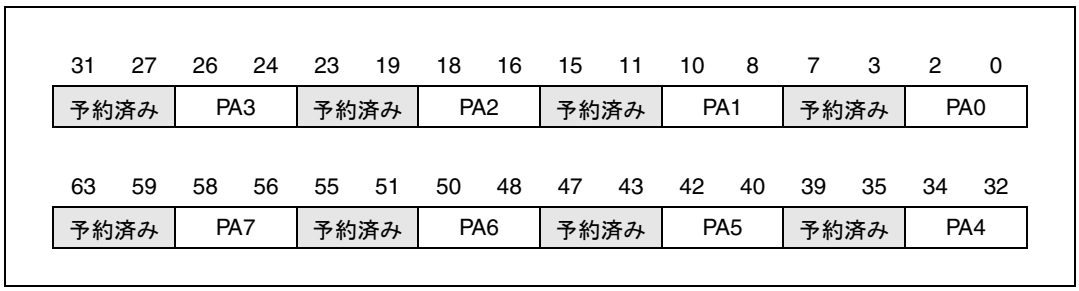


図 10-7. IA32_CR_PAT MSR

P6 ファミリ・プロセッサでは、IA32_CR_PAT MSR は PAT MSR であるのに注意する。

表 10-10. PAT でエンコード可能なメモリタイプ

エンコーディング	ニーモニック
00H	キャッシュ不可（UC）
01H	ライト・コンバイニング（WC）
02H	予約済み *
03H	予約済み *
04H	ライトスルー（WT）
05H	ライト・プロテクト（WP）
06H	ライトバック（WB）
07H	キャッシュなし（UC-）
08H - FFH	予約済み *

注 :

* これらのエンコーディングを使用すると、結果として一般保護例外（#GP）が生成される。

10.12.3. PAT からのメモリタイプの選択

ページのメモリタイプを PAT から選択するには、PAT、PCD、PWT の各ビットで構成される 3 ビットのインデックスを、ページのページ・テーブル・エントリまたはページ・ディレクトリ・エントリ内でエンコードする必要がある。表 10-11. は、PAT、PCD、PWT の各ビットのエンコーディングと、各エンコーディングで選択された PAT エントリを示している。PAT ビットは、4K バイト・ページを指すページ・テーブル・エントリのビット 7（図 3-14. と図 3-20. を参照）と、2M バイトまたは 4M バイトのページを指すページ・ディレクトリ・エントリのビット 12（図 3-15.、図 3-21.、図 3-23. を参照）である。PCD ビットはページ・テーブル・エントリのビット 4、PWT ビットはページ・ディレクトリ・エントリのビット 3 である。

表 10-7. に示すように、あるページに対して選択された PAT エントリは、そのページがマップされる物理メモリ領域の MTRR 設定と一緒に使用されて、そのページの有効なメモリタイプが決定される。

表 10-11. PAT、PCD、PWT フラグによる PAT エントリの選択

PAT	PCD	PWT	PAT エントリ
0	0	0	PAT0
0	0	1	PAT1
0	1	0	PAT2
0	1	1	PAT3
1	0	0	PAT4
1	0	1	PAT5
1	1	0	PAT6
1	1	1	PAT7

10.12.4. PAT のプログラミング

表 10-12. は、プロセッサの電源投入後またはリセット後における、各 PAT エントリのデフォルト設定を示している。ソフトリセット（INIT リセット）では、設定は変わらない。

表 10-12. 電源投入後またはリセット後における PAT エントリのメモリタイプ設定

PAT エントリ	電源投入後またはリセット後のメモリタイプ
PAT0	WB
PAT1	WT
PAT2	UC-
PAT3	UC
PAT4	WB
PAT5	WT
PAT6	UC-
PAT7	UC

PAT エントリの値を変更するには、WRMSR 命令を使用して IA32_CR_PAT MSR に書き込みを行う。RDMSR 命令および WRMSR 命令を使用して IA32_CR_PAT MSR の読み取り / 書き込みを行うには、動作中のソフトウェアの CPL が 0 でなければならない。表 10-10. は、PAT エントリに対して許されるエンコーディングを示している。未定義のメモリタイプのエンコーディングを PAT に書き込もうとすると、一般保護（#GP）例外が生成される。

注記

マルチプロセッサ・システムの場合は、すべてのプロセッサの PAT に同じ値を入れる必要がある。

オペレーティング・システムは、PAT エントリの変更によってプロセッサのキャッシュとトランスレーション・ルックアサイド・バッファ（TLB）の整合性が失われないようにする役割を受け持つ。そのためには、10.11.8. 項「MP システムでの MTRR の注意点」に指定された手順にしたがって、マルチプロセッサ・システムで MTRR の値を変更しなければならない。この手順では、プロセッサ・キャッシュと TLB を空にするなどの特定の操作シーケンスが必要である。

PAT を使用すれば、ページテーブル内でメモリタイプを自由に指定できる。したがって、メモリタイプが異なる 2 つ以上のリニアアドレスを、1 つの物理ページにマッピングも可能である。しかし、これを行うと、システム障害を引き起こす未定義の動作が発生する可能性があるので、これらはサポートしていない。特に、WC 書き込みはプロセッサのキャッシュをチェックしないことがあるため、WC ページをキャッシュ可能ページとして別名定義してはならない。以前にキャッシュ可能メモリタイプとしてマッピングされていたページを、WC ページに再マッピングする場合は、オペレーティング・システムは、以下の方法でこのような別名定義を防止する。

1. ページテーブル内のキャッシュ可能メモリタイプへの以前のマッピングを削除する。すなわち、それらが存在しないようにする。
2. （見込み的なマッピングも含めて）マッピングを使用した可能性のあるプロセッサの TLB を空にする。
3. 新しいメモリタイプ（例えば、WC）を使用して、同じ物理アドレスに対する新しいマッピングを作成する。
4. 以前にマッピングを使用した可能性のあるすべてのプロセッサのキャッシュを空にする。セルフ・スヌーピング（CPUID 機能、フラグビット 27）をサポートしているプロセッサでは、この手順は不要である。

オペレーティング・システムがページ・ディレクトリを（マップ・ラージ・ページへの）ページテーブルとして使用し、ページサイズ拡張（PSE）をイネーブルにしている場合、オペレーティング・システムは、4K バイトのページ・テーブル・エントリの PATi インデックス・ビットを注意深く使用しなければならない。ページ・テーブル・エントリの PAT インデックス・ビット（ビット 7）は、ページ・ディレクトリ・エントリのページ・サイズ・ビットに対応する。したがって、オペレーティング・システムは、ページ・ディレクトリとしても使用されるページテーブルのキャッシング・タイプを設定するとき、PAT エントリ PA0 ～ PA3 だけを使用できる。オペレーティング・システムがこのメモリをページテーブルとして使用しているときに、PAT エントリ PA4

～PA7を使用しようとすると、実際には、このメモリにページ・ディレクトリとしてアクセスするためのPSビットを設定する。

注記

PATをサポートしない従来のIA-32プロセッサとの互換性を保持させる場合は、PATエントリのエンコーディングを慎重に選択する必要がある（10.12.5.項「従来のIA-32プロセッサとのPATの互換性」を参照）。

10.12.5. 従来のIA-32プロセッサとのPATの互換性

PATをサポートするIA-32プロセッサの場合、IA32_CR_PAT MSRは常にアクティブである。すなわち、ページ・テーブル・エントリおよびページ・ディレクトリ・エントリ（これらはページを指す）のPCDビットとPWTビットは、PAT内のエントリを選択すれば、必ず間接的にページのメモリタイプを選択する。これらは、PATを実装していない従来のIA-32プロセッサの場合とは異なり、ページのメモリタイプを直接的に選択するものではない（表10-6.を参照）。

PATをサポートしない従来のIA-32プロセッサ用に作成されたコードとの互換性を得るため、PATメカニズムは従来のプロセッサとの下位互換性が実現されるように設計されている。この互換性は、3ビットのPATエントリ・インデックス内のPAT、PCD、PWTの各ビットのオーダリングによって提供される。PATを実装していないプロセッサの場合、PATインデックス・ビット（ページ・テーブル・エントリのビット7、ページ・ディレクトリ・エントリのビット12）は予約されており、0にセットされている。PATビットが予約されている場合は、PATの先頭の4つのエントリだけが、PCDビットおよびPWTビットで選択できる。電源を投入またはリセットすると（表10-12.を参照）、これらの先頭の4つのエントリがエンコードされ、PATを実装していないIA-32プロセッサにおいて通常はPCDビットおよびPWTビットで直接的に選択するのと同じメモリタイプが選択される。そのため、電源投入後またはリセット後にPAT内の先頭の4つのエントリのエンコーディングが変更されなければ、PATを実装していない従来のIA-32プロセッサ用に作成されたコードが、PATを実装しているIA-32プロセッサ上で正常に動作する。

11

インテル® MMX® テクノロジ・システム・プログラミング

第 11 章
インテル® MMX® テクノロジ・システム・プログラミング

11

本章では、インテル® MMX® テクノロジのさまざまな特長について説明している。MMX テクノロジをサポートするためにオペレーティング・システムを設計または強化するときには、以下のさまざまな特長を考慮しなければならない。本章では、MMX 命令セットのエミュレーション、MMX テクノロジ・ステート、MMX テクノロジ・レジスタの別名定義、MMX テクノロジ・ステートの保存、タスクとコンテキスト・スイッチングの注意点、例外処理、デバッギングについて取り上げている。

11.1. MMX® 命令セットのエミュレーション

IA-32 アーキテクチャは、MMX® テクノロジのエミュレーションをサポートしないで、x87 FPU 命令をサポートする。制御レジスタ CR0 内にある（x87 FPU 命令のエミュレーションを呼び出すための）EM フラグを、MMX テクノロジのエミュレーションに使用することはできない。EM フラグが設定されているときに MMX 命令が実行されると、無効なオペコード例外（UD#）が生成される。表 11-1. は、MMX 命令の実行時における制御レジスタ CR0 の EM、MP、TS の各フラグの関係を示している。

表 11-1. EM、MP、TS のさまざまな組み合わせに対して MMX® 命令が取る処置

CR0 フラグ			処置
EM	MP*	TS	
0	1	0	実行
0	1	1	#NM 例外
1	1	0	#UD 例外
1	1	1	#UD 例外

注：

* MMX 命令をサポートするプロセッサの場合は、MP フラグをセットする必要がある。

11.2. MMX® テクノロジ・ステートと MMX® テクノロジ・レジスタの別名定義

MMX® テクノロジ・ステートは、8 つの 64 ビット・レジスタ（MM0 ～ MM7）で構成される。これらのレジスタは、浮動小数点レジスタ R0 ～ R7 の下位 64 ビット（ビット 0 ～ 63）に別名定義される（図 11-1. を参照）。注意すべき点は、MMX テクノロジ・レジスタが浮動小数点レジスタ（R0 ～ R7）の物理位置にマップされるが、浮動小数点レジスタスタック（ST0 ～ ST7）内にあるレジスタの相対的な位置にはマップされないことである。結果として、MMX テクノロジ・レジスタのマッピングは固定されていて、浮動小数点ステータス・ワード（ビット 11 ～ 13）内のトップ・オブ・スタック（TOS）フィールドにある値の影響は受けない。

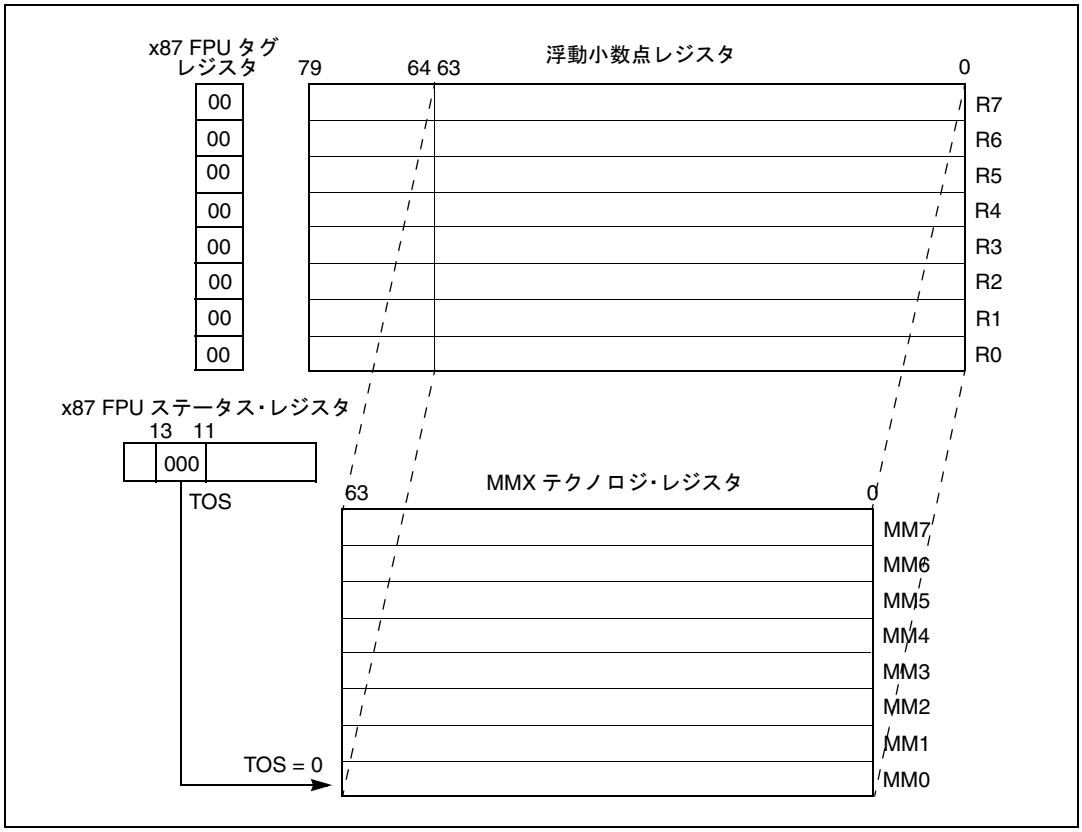


図 11-1. MMX® テクノロジ・レジスタの浮動小数点レジスタへのマッピング

MMX 命令によって値が MMX テクノロジ・レジスタに書き込まれると、その値は、対応する浮動小数点レジスタのビット 0 ～ 63 にも表示される。同様に、x87 FPU によっ

て浮動小数点の値が浮動小数点レジスタに書き込まれたときも、対応する MMX テクノロジ・レジスタ内にその値の下位 64 ビットが表示される。

MMX 命令を実行すると、浮動小数点レジスタ、x87 FPU タグワード、x87 FPU ステータス・ワードにストアされている浮動小数点ユニット（x87 FPU）の状態にさまざまな影響が現れる。これらの影響の内容は以下のとおりである。

- MMX 命令が値を MMX テクノロジ・レジスタに書き込むと同時に、対応する浮動小数点レジスタのビット 64 ～ 79 が、すべて 1 に設定される。
- （EMMS 命令以外の）MMX 命令が実行されると、x87 FPU タグワード内のタグ・フィールドがそれぞれ 00B（無効）に設定される。（11.2.1. 項「MMX® 命令、x87 FPU、FXSAVE、FXRSTOR 命令が x87 FPU タグワードに及ぼす影響」を参照。）
- EMMS 命令が実行されると、x87 FPU タグワード内の各タグ・フィールドが 11B（空き）に設定される。
- MMX 命令が実行されるごとに、TOS 値が 000B に設定される。

MMX 命令を実行しても、x87 FPU ステータス・ワードの他のビット（ビット 0 ～ 10、14、および 15）には影響を与えない。また、x87 FPU ステート（x87 FPU 制御ワード、命令ポインタ、データポインタ、またはオペコード・レジスタ）を構成している他の x87 FPU レジスタの内容にも影響を与えない。

表 11-2. では、MMX 命令が x87 FPU ステートに与える影響を要約している。

表 11-2. MMX® 命令が x87 FPU ステートに与える影響

MMX 命令タイプ	x87 FPU タグワード	x87 FPU ステータス・ワードの TOS フィールド	他の x87 FPU レジスタ	x87 FPU データレジスタのビット 64 ～ 79	x87 FPU データレジスタのビット 0 ～ 63
MMX テクノロジ・レジスタからの読み取り	すべてのタグは 00B に設定（有効）	000B	未変更	未変更	未変更
MMX テクノロジ・レジスタへの書き込み	すべてのタグは 00B に設定（有効）	000B	未変更	すべて 1 に設定	MMX データによる上書き
EMMS	すべてのフィールドは 11B（空き）に設定	000B	未変更	未変更	未変更

11.2.1. MMX® 命令、x87 FPU、FXSAVE、FXRSTOR 命令が x87 FPU タグワードに及ぼす影響

表 11-3. では、MMX® 命令、x87 FPU、FXSAVE、FXRSTOR 命令が x87 FPU タグワードのタグと、メモリにストアされているタグワードのイメージの中でそれに対応しているタグに及ぼす影響について要約している。

x87 FPU タグワードのフィールド内の値は、MMX テクノロジ・レジスタの内容または MMX 命令の実行に影響を与えない。ただし、MMX 命令は x87 FPU タグワードの内容を修正する。これについては、11.2. 節「MMX® テクノロジ・ステートと MMX® テクノロジ・レジスタの別名定義」で説明している。x87 FPU 命令を実行するときに、命令の実行に先立って x87 FPU ステートが初期化されていないか、または復元されていない場合は、x87 FPU の動作はこの修正内容の影響を受ける。

また留意すべき点は、(x87 FPU の状態情報を保存する）FSAVE、FXSAVE、FSTENV 命令が x87 FPU タグレジスタと、浮動小数点レジスタのそれぞれの内容を読み取り、各レジスタの実効タグ値（空白、非ゼロ、ゼロ、または特殊値）を決定し、そして更新されたタグワードをメモリにストアすることである。これらの命令を実行した後、x87 FPU タグワード内のすべてのタグは空白値（11B）に設定される。同様に EMMS 命令は、x87 FPU タグワード内のすべてのタグを 11B に設定して、MMX テクノロジ・ステートを MMX テクノロジ/浮動小数点レジスタからクリアする。

表 11-3. MMX® 命令、x87 FPU、FXSAVE、FXRSTOR 命令が x87 FPU タグワードに及ぼす影響

命令タイプ	命令	x87 FPU タグワード	メモリにストアされている x87 FPU タグワードのイメージ
MMX 命令	すべて（EMMS を除く）	すべてのタグは 00B（有効）に設定	影響なし
MMX 命令	EMMS	すべてのタグは 11B（空き）に設定	影響なし
x87 FPU	すべて（FSAVE、FSTENV、FRSTOR、FLDENV を除く）	修正された浮動小数点レジスタ用のタグが 00B または 11B に設定される。	影響なし
x87 FPU および FXSAVE	FSAVE、FSTENV、FXSAVE	タグ値とレジスタ値の読み取りと解釈が実行され、すべてのタグが 11B に設定される。	浮動小数点レジス内の実効値にしたがってタグが設定される。つまり、空白のレジスタは 11B とマーク付けられ、有効なレジスタは 00B（非ゼロの値）、01B（ゼロ）、または 10B（特殊値）とマーク付けられる。

表 11-3. MMX® 命令、x87 FPU、FXSAVE、FXRSTOR 命令が x87 FPU タグワードに及ぼす影響（続き）

命令タイプ	命令	x87 FPU タグワード	メモリにストアされている x87 FPU タグワードのイメージ
x87 FPU および FXRSTOR	FRSTOR、FLDENV、FXRSTOR	メモリ内の 11B とマーク付けられているタグがすべて 11B に設定される。他のすべてのタグは、対応する浮動小数点レジスタ内の値、00B（非ゼロの値）、01B（ゼロ）、または 10B（特殊値）にしたがって設定される。	タグが読み取られて解釈される。ただし修正されない。

11.3. MMX® テクノロジ・ステートとレジスタの保存と復元

MMX® テクノロジ・レジスタは x87 FPU データレジスタに別名定義されるため、以下の方法により、MMX テクノロジのステートをメモリに保存したり、メモリから復元したりできる。

- FSAVE、FNSAVE、または FXSAVE 命令を実行し、MMX テクノロジのステートをメモリに保存する。（FXSAVE 命令は、XMM レジスタおよび MXCSR レジスタのステートも保存する）
- FRSTOR または FXRSTOR 命令を実行し、MMX テクノロジのステートをメモリから復元する。（FXRSTOR 命令は、XMM レジスタおよび MXCSR レジスタのステートも復元する）

オペレーティング・システムには上述の保存方法と復元方法が必須である（11.4. 節「タスクスイッチまたはコンテキスト・スイッチ時の MMX® テクノロジ・ステートの保存」を参照）。場合によっては、以下の方法でアプリケーションが MMX テクノロジ・レジスタだけを保存し復元できる。

- 8 つの MOVQ 命令を実行して、MMX0 ～ MMX7 レジスタの内容をメモリに保存する。その後で、（任意で）EMMS 命令を実行して x87 FPU 内の MMX テクノロジ・ステートをクリアできる。
- 8 つの MOVQ 命令を実行して、MMX テクノロジ・レジスタの保存ずみ内容をメモリから MMX0 ～ MMX7 レジスタに読み込む。

注記

IA-32 アーキテクチャでは、x87 FPU タグワードのスキャンをサポートしないで、有効なエントリの保存だけをサポートしている。

11.4. タスクスイッチまたはコンテキスト・スイッチ時の MMX® テクノロジ・ステートの保存

あるタスクまたはコンテキストを別のタスクまたはコンテキストに切り替える場合、MMX® テクノロジ・ステートを保存しなければならないときがよくある。一般的な規則として、オペレーティング・システムに使用される既存のタスク・スイッチング・コードにx87 FPUステートを保存する機能が含まれている場合は、この機能を信頼してタスク・スイッチング・コードを書き直さずにMMXテクノロジ・ステートを保存できる。なぜこの機能を信頼できるかというと、MMXテクノロジ・ステートがFPUステートに別名定義されるからである（11.2. 節「MMX® テクノロジ・ステートとMMX®テクノロジ・レジスタの別名定義」を参照）。

IA-32アーキテクチャには、FXSAVE命令とFXRSTOR命令、SSE、SSE2、SSE3の拡張命令が導入されている。そのため、ステートを保存する機能をオペレーティング・システムまたはエグゼクティブに構築し、x87 FPU、MMXテクノロジ、SSE、SSE2、SSE3のステートを1回の操作で（しかも効率良く）保存することが可能になった。12.5. 節「オペレーティング・システムにおけるタスク・スイッチおよびコンテキスト・スイッチ時のx87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3ステートの自動セーブ機能の設計」では、こうした機能の設計方法について説明している。この節で説明する手法を応用すれば、必要に応じてMMXテクノロジおよびx87 FPUのステートだけを保存することもできる。

11.5. MMX® 命令実行時に発生する例外

MMX® 命令はx87 FPU浮動小数点例外を生成しないし、またEFLAGSレジスタやx87 FPUステータス・ワード内にあるプロセッサのステータス・フラグに影響を与えない。MMX命令の実行時に、以下の例外が生成される可能性がある。

- メモリアクセス時の例外
  - スタック・セグメントの故障（#SS）。
  - 一般保護（#GP）。
  - ページの故障（#PF）。
  - アライメント・チェック（#AC）（アライメント・チェックがイネーブルである場合）。
- システムの例外
  - 無効なオペコード（#UD）。MMX命令が実行されたときに制御レジスタCR0内のEMフラグが設定されている場合（11.1. 節「MMX® 命令セットのエミュレーション」を参照）。

— デバイス使用不可例外（#NM）。制御レジスタ CR0 内の TS フラグが設定されているときに、MMX 命令が実行された場合（12.5.1. 項「TS フラグによる x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの保存動作の制御」を参照）。

- 浮動小数点エラー（#MF）。（11.5.1. 項「保留中の x87 浮動小数点例外に対する MMX® 命令の影響」を参照。）
- その他の例外は、上記の例外に対する例外ハンドラの不正な実行を間接的な原因として発生する可能性がある。

11.5.1. 保留中の x87 浮動小数点例外に対する MMX® 命令の影響

x87 FPU浮動小数点例外が保留されている状態でプロセッサがMMX® 命令に遭遇した場合、プロセッサは x87 FPU 浮動小数点エラー（#MF）を MMX 命令の実行に先立って生成する。これによって、x87 FPU 浮動小数点エラー例外ハンドラが保留中の例外を処理できるようになる。この例外ハンドラが実行されている間、x87 FPU ステートは保守され、ハンドラに対して可視的である。11.2. 節「MMX® テクノロジ・ステートと MMX® テクノロジ・レジスタの別名定義」で説明されているように、MMX 命令は、例外ハンドラから返された時点で実行されて x87 FPU ステートを変更する。

11.6. MMX® テクノロジ・コードのデバッギング

MMX® 命令を実行するとき、IA-32 アーキテクチャのデバッグ機能は、他の IA-32 アーキテクチャ命令を実行する場合と同じ方法で動作する。

メモリ内の FSAVE/FNSAVE イメージや FXSAVE イメージから MMX テクノロジ・レジスタまたは x87 FPU レジスタの内容を正しく解釈するためには、デバッガは、x87 FPU レジスタの TOS に相対的な論理位置と、MMX テクノロジ・レジスタの物理位置との関係を考慮する必要がある。

x87 FPU のコンテキストでは、「ST*n*」は TOS に相対的な「*n*」位置の x87 FPU レジスタを表している。ただし x87 FPU タグワード内のタグは、x87 FPU レジスタ（R0 ～ R7）の物理位置に関連付けられている。また MMX テクノロジ・レジスタは、常にレジスタの物理位置を（R0 ～ R7 にマッピングされた MM0 ～ MM7 によって）表している。図 11-2. に、この関係を示す。ここでは、内周の円は x87 FPU の物理位置と MMX テクノロジ・レジスタを表している。外周の円は、x87 FPU レジスタの現行の TOS との相対位置を表している。

TOS が 0 に等しい場合（図 11-2. のケース A の場合）、ST0 は浮動小数点スタック上の物理位置 R0 を指す。MM0 は ST0 に、MMI は ST1 にマップされる。以下同様にマップされる。

TOS が 2 に等しい場合（図 11-2. のケース B の場合）、ST0 は物理位置 R2 を指す。MM0 は ST6 に、MMI は ST7 に、MM2 は ST0 にマップされる。以下同様にマップされる。

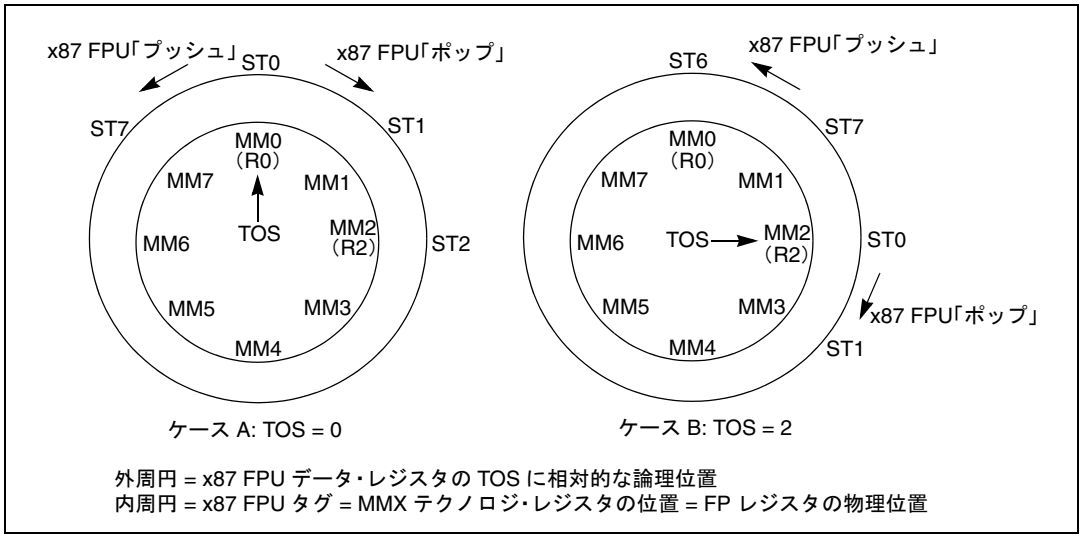


図 11-2. MMX® テクノロジ・レジスタの x87 FPU データ・レジスタ・スタックへのマッピング

12

SSE/SSE2/SSE3
システム・プログラミング

第 12 章
SSE/SSE2/SSE3
システム・プログラミング

12

本章では、ストリーミング SIMD 拡張命令（SSE）、ストリーミング SIMD 拡張命令 2（SSE2）、ストリーミング SIMD 拡張命令 3（SSE3）のさまざまな特徴について説明する。インテル® Pentium® III プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサをサポートするようにオペレーティング・システムを設計または強化するときは、これらの特徴を考慮に入れなければならない。本章では、SSE、SSE2、SSE3 拡張命令のイネーブル設定、オペレーティング・システムやエグゼクティブに対する SSE、SSE2、SSE3 拡張命令用のサポート、SIMD 浮動小数点例外、例外処理、タスク（コンテキスト）スイッチングの注意点を説明する。

12.1. オペレーティング・システムに対する SSE、SSE2、SSE3 拡張命令用のサポート

オペレーティング・システムやエグゼクティブにおいては、SSE、SSE2、SSE3 の拡張命令に対していくつかのサポートを提供しなければならない。例えば、これらの拡張命令を使用するためのプロセッサの初期化、ステート保存命令である FXSAVE および FXRSTOR の処理、SIMD 浮動小数点例外の処理などに対するサポートである。以降の各項では、オペレーティング・システムやエグゼクティブにおいてこのサポートを提供するためのガイドラインをいくつか説明する。SSE、SSE2、SSE3 の拡張命令では同じステートを共用し、同種の操作を実行するため、これらのガイドラインは 3 つすべての拡張命令セットに適用される。

『IA-32 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』の第 11 章「ストリーミング SIMD 拡張命令 2（SSE2）によるプログラミング」と第 12 章「ストリーミング SIMD 拡張命令 3（SSE3）によるプログラミング」では、アプリケーション・プログラムの観点から、SSE、SSE2、SSE3 の拡張命令のサポートを説明している。

12.1.1. オペレーティング・システムへの SSE、SSE2、SSE3 拡張命令用のサポートの追加

以下のガイドラインは、SSE、SSE2、SSE3 の拡張命令をサポートするために、オペレーティング・システムやエグゼクティブが実行しなければならない操作を説明したものである。

- プロセッサがSSE、SSE2、SSE3の拡張命令をサポートしているかどうかをチェックする。
- プロセッサがFXSAVE命令およびFXRESTOR命令をサポートしているかどうかをチェックする。
- SSE、SSE2、SSE3ステートに対する初期化を提供する。
- FXSAVE命令およびFXRSTOR命令に対するサポートを提供する。
- SSE命令およびSSE2命令によって生成された例外に対して、非数値例外ハンドラにサポートを提供する（必要な場合）。
- SIMD浮動小数点例外（#XF）に対して例外ハンドラを提供する。

以降の各項では、これらのガイドラインを実行する方法を説明する。

12.1.2. SSE、SSE2、SSE3 のサポートのチェック

サポート対象外のSSE命令、SSE2命令、SSE3命令を実行しようとすると、プロセッサは無効オペコード例外（#UD）を生成する。

オペレーティング・システムやエグゼクティブでは、SSE/SSE2/SSE3拡張命令を使用する前に、これらの命令がプロセッサでサポートされているかどうかをチェックする必要がある。このチェックを行うには、EAXレジスタに引き数1をセットしてCPUID命令を実行し、以下のビットがセットされていることを確認する。

- EDX、ビット25（SSE）
- EDX、ビット26（SSE2）
- ECX、ビット0（SSE3）

12.1.3. FXSAVE 命令および FXRSTOR 命令に対するサポートのチェック

プロセッサがFXSAVE命令およびFXRSTOR命令をサポートとしているかどうかを確認するには、別にチェックが必要となる。このチェックを行うには、EAXレジスタに引き数1をセットしてCPUID命令を実行し、以下のビットがセットされていることを確認する。

- EDX、ビット24（FXSR）

12.1.4. SSE、SSE2、SSE3 拡張命令の初期化

オペレーティング・システムやエグゼクティブにおいては、以下のステップによりSSE、SSE2、SSE3の拡張命令をセットアップし、アプリケーション・プログラムでこれらの命令を使用できるようにする必要がある。

1. CR4 のビット 9（OSFXSR ビット）を 1 にセットする。このフラグをセットすると、FXSAVE 命令と FXRSTOR 命令を使って SSE/SSE2/SSE3 のステートを保存および復元する機能がオペレーティング・システムにより提供されているものと見なされる。これらの命令は、タスク・スイッチ時および SIMD 浮動小数点例外（#XF）ハンドラの起動時に、SSE/SSE2/SSE3 のステートを保存するのによく使用される（12.4. 節「タスク・スイッチ時またはコンテキスト・スイッチ時における SSE、SSE2、SSE3 ステートのセーブ」および 12.1.6. 項「SIMD 浮動小数点例外（#XF）に対するハンドラの提供」を参照）。プロセッサが FXSAVE 命令および FXRSTOR 命令をサポートしていない場合、OSFXSR フラグをセットしようとすると、例外（#GP）が生成される。
2. CR4 のビット 10（OSXMMEXCPT ビット）を 1 にセットする。このフラグをセットすると、SIMD 浮動小数点例外（#XF）ハンドラがオペレーティング・システムにより提供されているものと見なされる（12.1.6. 項「SIMD 浮動小数点例外（#XF）に対するハンドラの提供」を参照）。

注記

制御レジスタ CR4 の OSFXSR ビットおよび OSXMMEXCPT ビットは、オペレーティング・システムがセットする必要がある。FXSAVE 命令および FXRSTOR 命令、または SIMD 浮動小数点例外の処理がオペレーティング・システムによってサポートされているかどうかについては、プロセッサは他に検出する手段がない。

---

3. CR0 制御レジスタの EM フラグ（ビット 2）をクリアする。この操作により、x87 FPU のエミュレーションがディスエーブルになる。これは、SSE 命令、SSE2 命令、SSE3 命令を実行するときに必要とされる操作である（2.5. 節「制御レジスタ」を参照）。
4. 制御レジスタ CR0 の MP フラグ（ビット 1）をクリアする。この設定は、SSE、SSE2、SSE3 の拡張命令をサポートするすべての IA-32 プロセッサで必要である（9.2.1. 項「x87 FPU 環境の構成」を参照）。

表 12-1. は、SSE 命令、SSE2 命令、SSE3 命令を実行したときのプロセッサの処置を示している。プロセッサの処置は、以下の設定で決定される。

- 制御レジスタ CR4 の OSFXSR フラグと OSXMMEXCPT フラグ
- CPUID 命令で返される SSE、SSE2、SSE3 の機能フラグ
- 制御レジスタ CR0 の EM、MP、TS フラグ

表 12-1. OSFXSR、OSXMMEXCPT、SSE、SSE2、SSE3、EM、MP、TS[1] の各種の組み合わせに対して取られる処置 [1]

CR4		CPUID	CR0 フラグ			処置
OSFXSR	OSXMMEXCPT	SSE、SSE2、SSE3	EM	MP[2]	TS	
0	X[3]	X	X	1	X	#UD 例外。
1	X	0	X	1	X	#UD 例外。
1	X	1	1	1	X	#UD 例外。
1	0	1	0	1	0	命令を実行。マスクされていない SIMD 浮動小数点例外が検出された場合は、#UD 例外。
1	1	1	0	1	0	命令を実行。マスクされていない SIMD 浮動小数点例外が検出された場合は、#XF 例外。
1	X	1	0	1	1	#NM 例外。

注 :

1. PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH を除く SSE 命令、SSE2 命令、SSE3 命令を実行する場合。
2. MMX® 命令をサポートするプロセッサの場合は、MP フラグをセットする必要がある。
3. X ー関係ない。

MXCSR レジスタの SIMD 浮動小数点例外マスク・ビット（ビット 7 ～ 12）、フラッシュ・ツー・ゼロ・フラグ（ビット 15）、デノーマル・ゼロ・フラグ（ビット 6）、丸め制御フィールド（ビット 13 および 14）は、デフォルト値 0 のままにしておく必要がある。これにより、アプリケーションがこれらの機能の使用方法を決めることができる。

12.1.5. SSE 命令、SSE2 命令、SSE3 命令によって生成される例外に対する非数値例外ハンドラの提供

SSE 命令、SSE2 命令、SSE3 命令は、他の IA-32 アーキテクチャ命令が生成するのと同じタイプのメモリ・アクセス例外（ページ・フォルト、セグメント不在、リミット違反など）と、これ以外の非数値例外を生成できる。

通常、既存の例外ハンドラは、これらの非数値例外や他の非数値例外をコードを修正せずに処理することができる。ただし、既存の例外ハンドラで使用されているメカニズムによっては、何らかの修正が必要になる場合もある。

SSE、SSE2、SSE3 の拡張命令は、以下に示すような非数値例外を生成できる。

- メモリ・アクセス例外
  - ー 無効オペコード（#UD）。

— スタック・セグメント・フォルト（#SS）。

— 一般保護（#GP）。通常、アライメントの合っていない 128 ビット・メモリ参照を使用して SSE 命令および SSE2 命令を実行すると、一般保護例外が生成される（MOVUPS 命令および MOVUPD 命令では、一般保護例外を生成しないで、アライメントの合っていない 128 ビット・メモリ位置をロードまたはストアできる）。16 バイト境界にアライメントが合っていないスタック・セグメントに対して 128 ビットの参照が行われた場合にも、スタック・セグメント・フォルト例外（#SS）の代わりに一般保護例外が生成される。

— ページ・フォルト（#PF）。

— アライメント・チェック（#AC）。このタイプのアライメント・チェックがイネーブルにされると、128 ビットより小さなサイズである 16 ビット、32 ビット、64 ビットのオペランドを操作する。アライメント・チェック例外の生成をイネーブルにするには、以下の作業を行う必要がある。

- AM フラグ（制御レジスタ CR0 のビット 18）をセットする。
- AC フラグ（EFLAGS レジスタのビット 18）をセットする。
- CPL を 3 にする。

アライメント・チェック例外がイネーブルになっている場合、MOVUPD 命令および MOVUPS 命令では 16 ビット、32 ビット、64 ビットのミスアライメントが検出されるが、128 ビットのミスアライメントの検出は保証されない。128 ビットのミスアライメントが検出されるかどうかは、プロセッサの世代によって異なる。

- システム例外

— 無効オペコード例外（#UD）。この例外は、以下の条件の下で SSE 命令、SSE2 命令、SSE3 命令を実行した場合に生成される。

- CPUID 命令で返された SSE/SSE2/SSE3 機能フラグが 0 にセットされている場合。この条件は、CLFLUSH 命令には影響を与えない。
- CPUID 命令で返された CLFSH 機能フラグが 0 にセットされている場合。この例外条件は、CLFLUSH 命令を実行するときにだけ関係する。
- 制御レジスタ CR0 の TS フラグ（ビット 3）の値とは関係なく、CR0 の EM フラグ（ビット 2）が 1 にセットされている場合。この条件は、PAUSE、PREFETCHh、MOVNTI、SFENCE、LFENCE、MFENSE、CLFLUSH には影響を与えない。
- 制御レジスタ CR4 の OSFXSR フラグ（ビット 9）が 0 にセットされている場合。この条件は、PAVGB、PAVGW、PEXTRW、PINSRW、PMAXSW、PMAXUB、PMINSW、PMINUB、PMOVMSKB、PMULHUW、PSADBW、PSHUFW、MASKMOVQ、MOVNTQ、MOVNTI、PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、CLFLUSH 命令には影響を与えない。

- 制御レジスタ CR4 の OSXMMEXCPT フラグ（ビット 10）が 0 にセットされているときに、SIMD 浮動小数点例外を生成する命令を実行した場合。12.5.1. 項「TS フラグによる x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの保存動作の制御」を参照。

– デバイス使用不可（#NM）。この例外は、CR0 の TS フラグ（ビット 3）が 1 にセットされているときに、SSE 命令、SSE2 命令、SSE3 命令を実行すると生成される。

上記の例外に対する不正な実行を間接的な原因として、上記以外の例外が発生する可能性がある。

12.1.6. SIMD 浮動小数点例外（#XF）に対するハンドラの提供

SSE、SSE2、SSE3 命令は、パックド整数操作に対しては数値例外を生成しない。パックドおよびスカラの単精度および倍精度浮動小数点操作に対しては以下の数値（SIMD 浮動小数点）例外を生成できる。

- 無効操作（#I）
- ゼロ除算（#Z）
- デノーマル・オペランド（#D）
- 数値オーバーフロー（#O）
- 数値アンダーフロー（#U）
- 不正確結果（精度）（#P）

これらの SIMD 浮動小数点例外（デノーマル・オペランド例外は除く）は、2 進浮動小数点算術演算についての IEEE 規格 754 で規定されており、x87 FPU 命令に対して x87 FPU 浮動小数点エラー例外（#MF）が生成されるのと同じ条件を表す。

これらの例外は、それぞれマスクできる。その場合、プロセッサは、例外ハンドラを起動せずにデスティネーション・オペランドに妥当な結果を返す。ただし、これらの例外がマスクされないままでいると、例外条件が検出されたときに SIMD 浮動小数点例外（#XF）が生成される。第 5 章の「割り込み 19 － SIMD 浮動小数点例外（#XF）」を参照。

マスクされていない SIMD 浮動小数点例外を処理するには、オペレーティング・システムやエグゼクティブにおいて例外ハンドラを提供する必要がある。『IA-32 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』の第 11 章の「SSE および SSE2 の SIMD 浮動小数点例外」の項では、SIMD 浮動小数点例外クラスについて説明し、それらを処理する例外ハンドラの作成に関する情報を提供している。

SIMD 浮動小数点例外（#XF）に対するハンドラがオペレーティング・システムにより提供されることを示すには、制御レジスタ CR0 の OSXMMEXCPT フラグ（ビット 10）をセットする必要がある。

12.1.6.1. 数値エラー・フラグと IGNNE#

SSE、SSE2、SSE3の拡張命令は、制御レジスタCR0のNEフラグを無視し（すなわち、NEフラグは常に設定されているものとして扱う）、IGNNE# ピンも無視する。マスクされていないSIMD浮動小数点例外が検出されると、常にSIMD浮動小数点例外（#XF）を生成することによって、その例外を報告する。

12.2. SSE、SSE2、SSE3 拡張命令のエミュレーション

IA-32アーキテクチャでは、x87 FPU命令のエミュレーションはサポートしているが、SSE命令、SSE2命令、SSE3命令のエミュレーションはサポートしていない。制御レジスタCR0のEMフラグ（x87 FPU命令のエミュレーションを起動するために用意されている）を、SSE命令、SSE2命令、SSE3命令のエミュレーションを起動するために使用できない。EMフラグがセットされているときにSSE命令、SSE2命令、SSE3命令を実行すると、無効オペコード例外（#UD）が生成される（表12-1.を参照）。

12.3. SSE、SSE2、SSE3 ステートのセーブとリストア

SSE、SSE2、SSE3ステートには、XMMとMXCSRレジスタのステートが含まれる。このステートをセーブ/リストアする際は、以下の方法を推奨する。

- FXSAVE命令を実行して、XMMレジスタとMXCSRレジスタのステートをメモリにセーブする。
- FXRSTOR 命令を実行して、XMM レジスタと MXCSR レジスタのステートを、FXSAVE命令によってメモリにセーブされたイメージからリストアする。

オペレーティング・システムは、必ずこの方法でセーブとリストアを行わなければならない（12.5.節「オペレーティング・システムにおけるタスク・スイッチおよびコンテキスト・スイッチ時のx87 FPU、MMX®テクノロジ、SSE、SSE2、SSE3ステートの自動セーブ機能の設計」を参照）。

アプリケーションは、必要に応じて、以下の方法でXMMおよびMXCSRレジスタだけをセーブすることができる。

- 8つのMOVDQ命令を実行して、XMM0～XMM7レジスタの内容をメモリにセーブする。
- STMXCSR命令を実行して、MXCSRレジスタのステートをメモリにセーブする。

アプリケーションは、必要に応じて、以下の方法でXMMおよびMXCSRレジスタだけをリストアできる。

- 8 つの MOVDQ 命令を実行して、XMM レジスタのセーブされた内容を、メモリからレジスタ XMM0 ～ XMM7 に読み込む。
- LDMXCSR 命令を実行して、MXCSR レジスタのステートを、メモリからリストアする。

12.4. タスク・スイッチ時またはコンテキスト・スイッチ時における SSE、SSE2、SSE3 ステートのセーブ

通常、あるタスクまたはコンテキストを別のタスクまたはコンテキストに切り替えるときは、SSE、SSE2、SSE3 のステートをセーブする必要がある。FXSAVE 命令と FXRSTOR 命令では、このステートを容易にセーブおよびリストアできる（12.3. 節「SSE、SSE2、SSE3 ステートのセーブとリストア」を参照）。これらの命令には、x87 FPU と MMX テクノロジのステートもセーブするさらなる利点がある。こうしたプロシージャを作成するガイドラインは、次の 12.5. 節「オペレーティング・システムにおけるタスク・スイッチおよびコンテキスト・スイッチ時の x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの自動セーブ機能の設計」で説明する。

12.5. オペレーティング・システムにおけるタスク・スイッチおよびコンテキスト・スイッチ時の x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの自動セーブ機能の設計

x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 のステートは、x87 FPU、MMX テクノロジ、XMM、MXCSR の各レジスタのステートで構成される。FXSAVE 命令と FXRSTOR 命令では、このステートを高速にセーブおよびリストアできる。
FSAVE/FNSAVE 命令と FRSTOR 命令により x87 FPU および MMX テクノロジのステートをセーブするオペレーティング・システムやエグゼクティブに、タスク・スイッチ機能またはコンテキスト・スイッチ機能がすでに実装されている場合は、これらの機能は、FSAVE/FNSAVE 命令と FRSTOR 命令の代わりに FXSAVE 命令と FXRSTOR 命令を使用すると、SSE、SSE2、SSE3 のステートもセーブおよびリストアされるように拡張できることが多い。

タスク・スイッチ機能またはコンテキスト・スイッチ機能を初めから作成しなければならない場合、FXSAVE 命令と FXRSTOR 命令で x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 のステートをセーブおよびリストアできるようにするには、いくつかの方法が選択できる。

- オペレーティング・システムは、タスクとして実行されるアプリケーションが、タスク・スイッチの際に、タスクが一時停止される前に x87 FPU、MMX テクノロジ、

XMM、MXCSR レジスタのステートをセーブし、タスクが再開されるときに、そのレジスタをリストアする役割を受け持つように要求できる。協調マルチタスク・オペレーティング・システムでは、この方法が望ましい。協調マルチタスク・オペレーティング・システムでは、アプリケーションは、タスク・スイッチが起こる時点を制御または決定でき、タスク・スイッチの前にステートをセーブできる。

- オペレーティング・システムは、タスク・スイッチ・プロセスの一部として、FXSAVE 命令を使用して x87 FPU、MMX テクノロジ、XMM、MXCSR レジスタを自動的にセーブし、一時停止されたタスクが再開されるときに、FXRSTOR 命令を使用してレジスタのステートを自動的にリストアできる。この場合は、x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートをタスク・ステートの一部としてセーブしなければならない。プリエンティティブなマルチタスク・オペレーティング・システムでは、この方法が適切である。プリエンティティブなマルチタスク・オペレーティング・システムでは、アプリケーションはいつ割り込みが発生するかわからないため、事前にタスク・スイッチの準備ができない。このため、オペレーティング・システムが、必要に応じて、タスク・ステートと x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートをセーブ / リストアする役割を受け持つ。
- オペレーティング・システムは、タスク・スイッチ・プロセスの一部として x87 FPU、MMX テクノロジ、XMM、MXCSR レジスタをセーブする役割を受け持ち、新しいタスクによって x87 FPU、MMX 命令、SSE、SSE2、SSE3 命令が実際に実行されるまで、MMX テクノロジおよび x87 FPU ステートのセーブ動作を遅延させることができる。この方法を使用すれば、新しいタスクの中で x87 FPU、MMX、SSE、SSE2、SSE3 命令を実行する必要がある場合にのみ、x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートがセーブされる（このセーブ方法の詳細については、12.5.1. 項「TS フラグによる x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの保存動作の制御」を参照）。

12.5.1. TS フラグによる x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートの保存動作の制御

FXSAVE 命令を使用して x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートをセーブすると、プロセッサ・オーバーヘッドが必要になる。新しいタスクが x87 FPU、MMX テクノロジ、XMM、MXCSR レジスタにアクセスしない場合は、タスク・スイッチ時にステートを自動的にセーブしないことによって、このオーバーヘッドを避けられる。

制御レジスタ CR0 の TS フラグの設定によって、オペレーティング・システムは、新しいタスク内で命令が実際にアクセスされ、この状態が検出されるまで、x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートのセーブ動作を遅延できる。TS フラグがセットされている場合、プロセッサは、x87 FPU 命令、MMX 命令、SSE 命令、SSE2 命令、SSE3 命令の命令ストリームを監視する。プロセッサは、これらの命令の 1 つを

検出すると、その命令を実行する前に、デバイス使用不可例外（#NM）を発生させる。その後、デバイス使用不可例外ハンドラを使用して、（FXSAVE 命令を使って）直前のタスクの x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートをセーブし、（FXRSTOR 命令を使って）現在のタスクの x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートをロードできる。切り替え先のタスクで x87 FPU 命令、MMX 命令、SSE 命令、SSE2 命令、SSE3 命令が検出されない場合は、デバイス使用不可例外は生成されず、タスク・ステートはセーブされない。

TS フラグは、（制御レジスタ CR0 に対して MOV 命令を実行することによって）明示的に設定することも、（IA-32 アーキテクチャのネイティブなタスク・スイッチング機構を使用して）暗黙的に設定することもできる。ネイティブなタスク・スイッチング機構を使用する場合は、プロセッサは、タスク・スイッチ時に TS フラグを自動的にセットする。デバイス使用不可ハンドラは、x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートをセーブした後、CLTS 命令を実行して、TS フラグをクリアしなければならない。

図 12-1. は、TS フラグを使用して x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートのセーブ動作を実行する、オペレーティング・システムの例を示している。この例では、タスク A が現在実行中のタスクであり、タスク B が新しいタスクである。オペレーティング・システムは、各タスクの x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステート保存領域を維持し、ステートを所有するタスクを示す変数（x87_MMX_SSE_SSE2_SSE3_StateOwner）を定義する。この例では、タスク A が現在のオーナである。

タスク・スイッチ時には、オペレーティング・システムのタスク・スイッチング・コードは、以下の疑似コードを実行して、現在の x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートのオーナに応じて TS フラグを設定しなければならない。新しいタスク（この例ではタスク B）が現在のこのステートのオーナと一致しない場合は、TS フラグは 1 に設定される。それ以外の場合は、TS フラグは 0 に設定される。

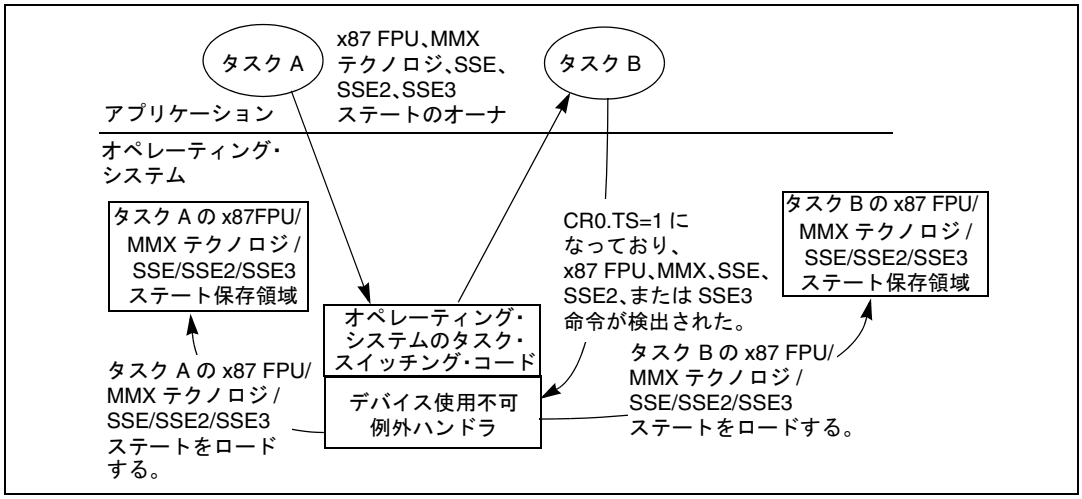


図 12-1. オペレーティング・システムによって制御されたタスク・スイッチ時の、x87 FPU、MMX® テクノロジ、SSE、SSE2、SSE3 ステートのセーブ動作の例

```
IF Task_Being_Switched_To ≠ x87FPU_MMX_SSE_SSE2_SSE3_StateOwner
   THEN
      CR0.TS ← 1;
   ELSE
      CR0.TS ← 0;
FI;
```

TS フラグが 1 にセットされているときに、新しいタスクが x87 FPU、MMX テクノロジ、XMM、または MXCSR レジスタにアクセスしようとすると、デバイス使用不可例外（#NM）が生成される。デバイス使用不可例外ハンドラが以下の疑似コードを実行する。

```
FSAVE “To x87FPU/MMX/SSE/SSE2/SSE3 State Save Area for Current
    x87FPU_MMX_SSE_SSE2_SSE3_StateOwner”;
FRSTOR “x87FPU/MMX/SSE/SSE2/SSE3 State From Current Task’s
    x87FPU/MMX/SSE/SSE2/SSE3 State Save Area”;
x87FPU_MMX_SSE_SSE2_SSE3_StateOwner ← Current_Task;
CR0.TS ← 0;
```

この例外ハンドラ・コードは、以下のタスクを実行する。

- 現在の x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートのオーナのステート保存領域に、x87 FPU、MMX テクノロジ、XMM、または MXCSR レジスタをセーブする。
- 新しいタスクの x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステート保存領域から、x87 FPU、MMX テクノロジ、XMM、または MXCSR レジスタをリストアする。

- 現在の x87 FPU、MMX テクノロジ、SSE、SSE2、SSE3 ステートのオーナを、現在のタスクにアップデートする。
- TS フラグをクリアする。

13

システム管理

第 13 章
システム管理

13

本章では、システムリソースを管理するために使用する IA-32 アーキテクチャの 2 つの側面、すなわちシステム管理モード（SMM: System Management Mode）と温度モニタ機能について説明する。

SMM は、代替の動作環境を提供する。この環境を使用すると、より効率的なエネルギー利用のために各種のシステムリソースを監視および管理し、システム・ハードウェアを制御し、専用コードを実行できる。SMM は、Intel386™ SL プロセッサ（Intel386 プロセッサのモバイル専用バージョン）で IA-32 アーキテクチャに導入された。また SMM は、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、および（Intel486™ SL プロセッサと Intel486 プロセッサの機能強化バージョンを始めとする）Intel486 プロセッサで使用できる。SMM をサポートしているハードウェアの詳細については、IA-32 プロセッサの各デベロッパーズ・マニュアルを参照。

温度モニタ機能では、IA-32 プロセッサのコア温度を監視および制御できる。この機能は、P6 ファミリ・プロセッサで導入されたが、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、インテル® Pentium® M プロセッサでその機能が拡張された。

13.1. システム管理モード（SMM）の概要

SMM は、電力管理、システム・ハードウェア制御、専用 OEM 設計コードなどのシステム全体の機能を処理するための専用の動作モードである。SMM は、システムのファームウェアだけによって使用されるように設計されているため、アプリケーション・ソフトウェアや汎用システム・ソフトウェアは SMM を使用できない。SMM の主な利点をあげると、明確で簡単な独立したプロセッサ環境を提供できることである。このプロセッサ環境は、オペレーティング・システム、エグゼクティブ・アプリケーション、ソフトウェア・アプリケーションに対して透過的に動作する。

システム管理割り込み（SMI）を通じて SMM を起動すると、プロセッサは、プロセッサの現行の状態（プロセッサのコンテキスト）をセーブしてから、次にシステム管理 RAM（SMRAM）に入れられている独立した動作環境に切り替わる。SMM にある間、プロセッサは SMI ハンドラコードを実行して、使っていないディスクドライブやモニタなどの電源を切断したり、専用コードを実行したり、システム全体を中断状態にするといった操作を実行する。SMI ハンドラが操作を完了すると、ハンドラは再開（RSM）

命令を実行する。この命令によって、プロセッサはセーブされていたプロセッサのコンテキストを再ロードし、保護モードまたは実モードに切り替え、割り込まれていたアプリケーションやオペレーティング・システムのプログラムまたはタスクの実行を再開する。

次の SMM の各メカニズムによって、SMM はアプリケーション・プログラムやオペレーティング・システムに対して透過的になっている。

- SMM には、SMI（システム管理割り込み）によってのみ移行できる。
- プロセッサは、他の動作モードからのアクセスが不可能な独立したアドレス空間（SMRAM）内で SMM コードを実行する。
- SMM に移行する際に、プロセッサは割り込まれたプログラムまたはタスクのコンテキストをセーブする。
- 通常オペレーティング・システムによって処理されるすべての割り込みは、SMM に移行するとディスエーブルになる。
- RSM 命令は SMM 内だけで実行できる。

SMM は、特権レベルやアドレス・マッピングが存在しない点で実アドレスモードに似ている。SMM プログラムは、最大 4G バイトのメモリをアドレス指定でき、またすべての I/O と適用可能システム命令を実行できる。SMM の実行環境の詳細については、13.5. 節「SMI ハンドラの実行環境」を参照。

注記

P6 ファミリ・プロセッサで使用可能な物理アドレス拡張（PAE）メカニズムは、プロセッサが SMM にあるときはサポートされない。

13.2. システム管理割り込み（SMI: System Management Interrupt）

SMM に移行するには、プロセッサ上の SMI# ピンを通じて SMI に信号を送信するか、または APIC バスを通じて受け取った SMI メッセージによって SMI に信号を送信する以外に方法はない。SMI は、マスク不可能な外部割り込みであり、プロセッサの割り込み / 例外処理のメカニズムやローカル APIC とは独立して動作する。SMI は、NMI とマスク可能割り込みに優先する。SMM は再入可能のモードではない。つまり、プロセッサが SMM にある間、SMI はディスエーブルになる。

注記

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサでは、アプリケーション・プロセッサとして指定されているプロセッサが MP 初期化シーケンス中に起動 IPI（SIPI）を待っているときには、プロセッサはSMIがマスクされているモードに設定されている。ただし、アプリケーション・プロセッサがSIPI待機モードのときにSMIが受信されると、そのSMIは保留される。このときプロセッサは、SIPI を受信すると、その応答として直ちに保留中のSMIを処理し、SMMに移行後にそのSIPIを処理する。

13.3. SMM と他のプロセッサ動作モード間のスイッチング

図2-2.に、プロセッサがSMMとその他のプロセッサ動作モード（保護、実アドレス、仮想 8086 モード）間を移動する方法を示す。プロセッサが実アドレス、保護、仮想8086のどれかのモードにある間にSMIに信号を送信すると、常にプロセッサがSMMに切り替わる。また、RSM 命令を実行すると、プロセッサは SMI が発生した時点のモードに必ず戻る。

13.3.1. SMM への移行

プロセッサがSMIを扱うのは、常に、プログラム実行においてアーキテクチャ的に定義されている「割り込み可能」ポイント上である（このポイントは、一般にIA-32アーキテクチャの命令境界上にある）。プロセッサが SMI を受け取ると、プロセッサはすべての命令がリタイアし、またすべてのストアが完了するまで待機する。次にプロセッサは、その現行コンテキストをSMRAM内にセーブし（13.4.節「SMRAM」を参照）、SMMに移行し、SMIハンドラの実行を開始する。

SMMに移行すると、プロセッサはSMM処理が開始されたことを外部ハードウェアに信号で知らせる。この信号送信メカニズムは、プロセッサに依存している。P6ファミリ・プロセッサでは、SMIアクノレッジ・トランザクションがシステムバス上で生成される。プロセッサが SMM にある間、バス・トランザクションが生成されるたびに多重化ステータス信号の EXF4 がアサートされる。インテル® Pentium® プロセッサとIntel486™プロセッサでは、SMIACT# ピンがアサートされる。

SMIは、デバッグ例外や外部割り込みより高い優先順位を持つ。したがって、命令境界でSMIと共にNMI、マスク可能ハードウェア割り込み、またはデバッグ例外が発生した場合は、SMIだけが処理される。プロセッサがSMMにある間は、後続のSMI要求は認知されない。RSM命令によってプロセッサがSMMを終了すると、プロセッサがSMMにある間（つまりSMMが外部ハードウェアに対して認知された後）に発生した最初の SMI 割り込み要求がラッチされて処理される。プロセッサは、SMM にある間は1つのSMIしかラッチしない。

SMM にある間の実行環境の詳細については、13.5. 節「SMI ハンドラの実行環境」を参照。

13.3.2. SMM の終了

SMM を終了する唯一の方法は、RSM 命令を実行することである。RSM 命令は、SMI ハンドラにしか使用できない。プロセッサが SMM にない場合に RSM 命令を実行しようとすると、無効オペコード例外（#UD）が発生する。

RSM 命令は、状態保存イメージを SMRAM からプロセッサのレジスタに再ロードすることで、プロセッサのコンテキストをリストアする。その後、プロセッサは、SMIACK トランザクションをシステムバス上に戻し、割り込みをかけられたプログラムにプログラム制御を戻す。

RSM 命令が問題なく完了すると、プロセッサは、SMM が終了したことを外部ハードウェアに通知する。P6 ファミリ・プロセッサでは、SMI アクノレッジ・トランザクションがシステムバス上で生成されるが、多重化ステータス信号 EXF4 はシステムバス上で生成されなくなる。インテル® Pentium® プロセッサと Intel486™ プロセッサでは、SMIACT# ピンが放棄される。

プロセッサは、SMRAM にセーブされている状態情報が無効であることを検出すると、シャットダウン状態に移行し、この状態に入ったことを示す特殊なバスサイクルを生成する。シャットダウンが発生するのは、次の状況に限られる。

- CR4 への書き込み時に、制御レジスタ CR4 内の予約ビットが 1 にセットされている場合。このエラーは、SMI ハンドラコードが SMRAM の保存状態マップの予約領域を修正しない限り、発生しない（13.4.1. 項「SMRAM 状態保存マップ」を参照）。CR4 は、予約済みの位置にあるステートマップに保存され、その保存ステートでは読み込みまたは修正を行えないことに注意する。
- 制御レジスタ CR0 に、無効な組み合わせのビットが書き込まれている場合。特に、PG が 1 に、PE が 0 に設定されているか、NW が 1 に、または CD が 0 に設定されている場合。
- （インテル Pentium プロセッサと Intel486 プロセッサのみ）SMBASE レジスタにストアされているアドレスが、RSM 命令の実行時に 32K バイトの境界上にアライメントが合っていない場合。この制限は P6 ファミリ・プロセッサには適用されない。

シャットダウン状態では、RESET#、INIT# または NMI# がアサートされるまでインテル・プロセッサは命令を実行しない。インテル Pentium ファミリ・プロセッサはシャットダウン状態で SMI# 信号を認識するが、P6 ファミリ・プロセッサと Intel486 プロセッサは認識しない。インテルでは、どのプロセッサ・ファミリでも、SMI# を使用したシャットダウン状態からの回復をサポートしていない。この状況でのプロセッサの応答は定義されていない。インテル® Pentium® 4 プロセッサおよびそれ以降のプロセッサでは、シャッ

トダウンによってINTRとA20Mは禁止されるが、他の禁止事項は変更されない。これらのプロセッサ上では、SMMハンドラ内でNMIの禁止を解除する処置がとられない場合、NMIは禁止される（13.8.項を参照）。

SMIを受け取ったときにプロセッサがHALT状態にある場合は、プロセッサは少し違う方法でSMMからの戻りを処理する（13.11.節「自動HALT再スタート」を参照）。また、SMMから戻るときにSMBASEアドレスを変更できる（13.12.節「SMBASEの再配置」を参照）。

13.4. SMRAM

SMMにある間、プロセッサはSMRAM空間でコードを実行し、データをストアする。SMRAM空間は、プロセッサの物理アドレス空間にマッピングされる。また、サイズは最大4Gバイトまで可能である。プロセッサは、この空間を使用してプロセッサのコンテキストをセーブし、SMIハンドラのコード、データ、スタックをストアする。またこの空間を使用して、システム管理情報（例えば、システム・コンフィギュレーション情報や電源を切断されたデバイスの特殊情報）やOEM固有の情報もストアできる。

SMRAMのデフォルト・サイズは64Kバイトで、物理メモリ内のSMBASEと呼ばれるベース物理アドレスから始まる（図13-1.を参照）。ハードウェア・リセット後のSMBASEデフォルト値は30000Hである。プロセッサは、SMIハンドラの最初の命令をアドレス[SMBASE + 8000H]で探す。プロセッサは、プロセッサの状態を[SMBASE + FE00H]～[SMBASE + FFFFH]の領域内にストアする。状態保存領域のマッピングの説明については、13.4.1.項「SMRAM状態保存マップ」を参照。

システムロジックは、SMRAMに対する物理アドレス範囲を[SMBASE + 8000H]から[SMBASE + FFFFH]までデコードすることが最低限求められる。必要であれば、さらに大きな領域をデコードできる。このSMRAMの設定可能サイズは32Kバイト～4Gバイトである。

SMRAMの位置は、SMBASE値を変更することで変更できる（13.12.節「SMBASEの再配置」を参照）。マルチプロセッサ・システム内のプロセッサすべてが、同じSMBASE値（30000H）で初期化されることに注意する必要がある。初期化ソフトウェアは、連続して各プロセッサをSMMに置き、他のプロセッサのSMBASEとオーバーラップしないようにそのSMBASEを変更しなければならない。

SMRAMの実際の物理位置を、システムメモリまたは個別のRAMメモリ内に置くことができる。プロセッサが（P6ファミリ・プロセッサの場合）SMIアクノレッジ・トランザクションを生成したり、または（インテル® Pentium® プロセッサとIntel486™プロセッサの場合）SMIACT# ピンをアサートしたりするのは、プロセッサがSMIを受け取ったときである（13.3.1.項「SMMへの移行」を参照）。

システムロジックは、SMI アクノレッジ・トランザクションを使用して、または SMIACT# ピンをアサートして、SMRAM へのアクセスをデコードし、（希望する場合）各アクセスを特定の SMRAM メモリにリダイレクトできる。個別の RAM メモリを SMRAM に使用しているときにプロセッサが SMM にない場合、SMRAM をシステムメモリ空間にマッピングするプログラム可能な方法を、システムロジックは提供しなければならない。このメカニズムによって、起動プロシージャをイネーブルにし、SMRAM 空間を初期化（つまり SMI ハンドラをロード）して、SMM 時に SMI ハンドラを実行できる。

13.4.1. SMRAM 状態保存マップ

プロセッサは、初めに SMM に移行すると、プロセッサ自体の状態を SMRAM の状態保存領域に書き込む。状態保存領域は [SMBASE + 8000H + 7FFFH] から始まり、[SMBASE + 8000H + 7E00H] まで拡張される。表 13-1. に状態保存マップを示す。1 列目のオフセットは、SMBASE + 8000H に対する相対値である。ソフトウェアは予約済みの領域を使用してはならない。

SMRAM の（3 列目に「はい」が記入されている）状態保存領域のレジスタには、SMI ハンドラが読み取って変更できるものがある。変更された値は、RSM 命令によってプロセッサのレジスタにリストアされる。またいくつかのレジスタイメージは読み取り専用であるため、修正してはならない（修正すると、レジスタが予測できない動作をすることになる）。SMI ハンドラは、予約済みと記されている領域内にストアされているどんな値にも依存してはならない。

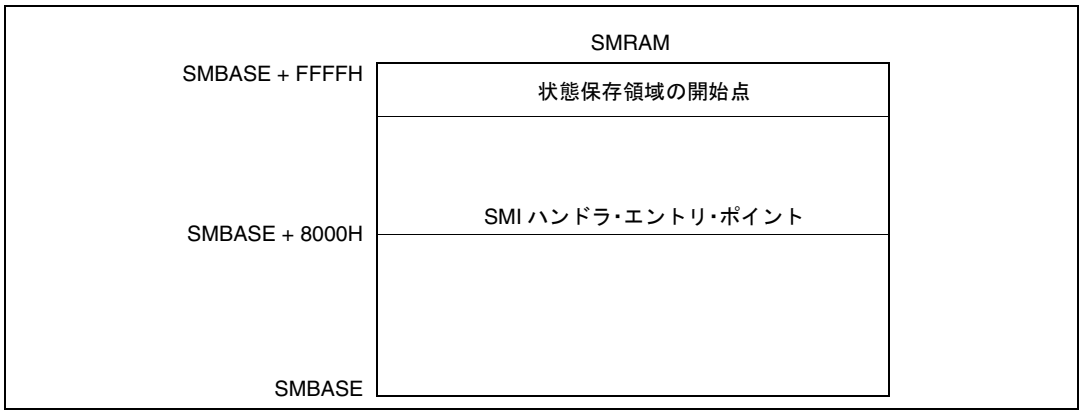


図 13-1. SMRAM の使用

表 13-1. SMRAM 状態保存マップ

オフセット（SMBASE + 8000H に追加）	レジスタ	書き込み可能か
7FFCH	CR0	いいえ
7FF8H	CR3	いいえ
7FF4H	EFLAGS	はい
7FF0H	EIP	はい
7FECH	EDI	はい
7FE8H	ESI	はい
7FE4H	EBP	はい
7FE0H	ESP	はい
7FDCH	EBX	はい
7FD8H	EDX	はい
7FD4H	ECX	はい
7FD0H	EAX	はい
7FCCH	DR6	いいえ
7FC8H	DR7	いいえ
7FC4H	TR*	いいえ
7FC0H	予約済み	いいえ
7FBCH	GS*	いいえ
7FB8H	FS*	いいえ
7FB4H	DS*	いいえ
7FB0H	SS*	いいえ
7FACH	CS*	いいえ
7FA8H	ES*	いいえ
7FA4H	I/O ステート・フィールド。13.7. 節を参照。	いいえ
7FA0H	I/O メモリ・アドレス・フィールド。13.7. 節を参照。	いいえ
7F9FH-7F03H	予約済み	いいえ
7F02H	自動 HALT 再スタート・フィールド（ワード）	はい
7F00H	I/O 命令再スタート・フィールド（ワード）	はい
7EFCH	SMM リビジョン識別子フィールド（ダブルワード）	いいえ
7EF8H	SMBASE フィールド（ダブルワード）	はい
7EF7H - 7E00H	予約済み	いいえ

注記：

* 上位 2 バイトは予約済み。

次のレジスタはセーブされて（ただし、読み取り不可）、SMM の終了と同時にリストアされる。

- 制御レジスタ CR4（SMM の実行中は、このレジスタはすべて 0 に設定される）。
- CS、DS、ES、FS、GS、SS の各セグメント・レジスタにストアされている隠されたセグメント・ディスクリプタ情報

プロセッサの電源を切断するために SMI 要求を発行した場合は、SMM 状態保存で予約された位置すべての値を不揮発性メモリにセーブしなければならない。

SMI 命令と RSM 命令の後、次の状態は自動的にはセーブもリストアもされない。

- デバッグレジスタ DR0 ～ DR3
- x87 FPU レジスタ
- MTRR
- 制御レジスタ CR2
- モデル固有レジスタ（P6 ファミリ・プロセッサとインテル® Pentium® プロセッサの場合）、またはテストレジスタ TR3 ～ TR7（インテル Pentium プロセッサおよび Intel486™ プロセッサ内で使用する場合）
- トラップ・コントローラの状態
- マシン・チェック・アーキテクチャ・レジスタ
- APIC 内部割り込み状態（ISR、IRR など）
- マイクロコードの更新状態

SMI を使用してプロセッサの電源を切断する場合は、SMM に戻る前に電源投入リセットが必要となる。このリセットによってプロセッサの状態のほとんどがプロセッサのデフォルト値にリセットされる。したがって、電源切断をトリガしようとする SMI ハンドラは、最初に上記の各レジスタを直接読み取って、（RAM のその他の部分と一緒に）不揮発性メモリにセーブする必要がある。電源投入リセットの後、実行が再開された SMI ハンドラは、これらの値をシステムの他の状態と一緒にリストアする必要がある。SMI ハンドラは、プロセッサ内のレジスタを変更したときにいつでも、これらのレジスタをセーブしリストアしなければならない。

注記

MSRの小さなサブセット（例えば、タイムスタンプ・カウンタや性能モニタリング・カウンタ）に、勝手に書き込むことはできない。したがって、勝手にセーブもリストアもできない。SMMベースの電源切断とリストアは、これらのレジスタ値を使用していない、また依存していないオペレーティング・システムによってのみ実行されなければならない。オペレーティング・システムの開発者は、この事実を認識して、オペレーティング・システム支援の電源切断とリストア用のソフトウェアが、これらのレジスタ値の予期しない変更に影響されないようにしておく必要がある。

13.4.2. SMRAM のキャッシング

IA-32プロセッサは、SMMに移行するまたはSMMから出る前にプロセッサのキャッシュを自動的にライトバックしないし無効にもしない。こうした動作のため、システムメモリにSMRAMを配置する場合は注意が必要である。また、SMMと保護モードの間を何度も切り替えるときに、キャッシュのコヒーレンシを維持するためにSMRAMをキャッシュする場合にも注意しなければならない。以下の3つの方法のいずれかでSMRAMをシステムメモリに配置すれば、キャッシュのコヒーレンシが保証される。

- オペレーティング・システムやアプリケーションからアクセスできないシステムメモリ専用のセクションにSRAMを配置する。ここで、SRAMをキャッシュ可能（WB、WT、WC）として指定すると、プロセッサのパフォーマンスを最適化できる。このとき、SMMへの移行またはSMMの終了によって、キャッシュのコヒーレンシが損なわれる危険はない。
- オペレーティング・システムが使用している領域とオーバーラップするメモリ・セクション（ビデオメモリなど）にSRAMを配置する。ただし、SMRAMはキャッシュ不可（UC）として指定する。この方法により、SMMでのキャッシュ・アクセスが防止され、キャッシュのコヒーレンシが維持されるが、キャッシュ不可のメモリを使用するとSMMコードのパフォーマンスは低下する。
- オペレーティング・システムやアプリケーション・コードが使用している領域とオーバーラップするシステム・メモリ・セクションにSRAMを配置する。ただし、SMMモードに移行したりSMMモードを終了するときは、キャッシュを明示的にフラッシュ（ライトバックおよび無効化）する。この方法は、キャッシュのコヒーレンシを維持できるが、キャッシュを完全に二回フラッシュするというオーバーヘッドを伴う。

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサの場合は、最初の2つの方法でSMRAMを配置するのが望ましい。SMRAMは、

メモリのオーバーラップ領域と専用領域の間で分割される。SMM に移行すると、アクセスされる SMRAM 空間はビデオメモリ（通常は下位メモリに配置される（とオーバーラップする。この SMRAM セクションは、UC メモリとして指定される。次に、初期 SMM コードは、システムメモリ専用領域（通常は上位メモリ）に配置されている第 2 の SMRAM セクションにジャンプする。この SMRAM セクションをキャッシュすると、プロセッサのパフォーマンスを最適化できる。

SMM への移行時にキャッシュを明示的にフラッシュするシステム（上記の第 3 の方式）の場合、キャッシュのフラッシュは、SMM へ移行する要求と同時に FLUSH# ピンをアサートして達成される（通常は、SMI# ピンをアサートすると起動される）。FLUSH# ピンと SMI# ピンでは、FLUSH# が最初に処理される。この動作を保証するために、プロセッサは、FLUSH# ピンと SMI# ピンの相互作用に関する次の制限事項を満たしていなければならない。FLUSH# ピンと SMI# ピンの動作時が同じで、セットアップ・タイムとホールドタイムが一緒になるシステムでは、FLUSH# ピンと SMI# ピンを同じクロックでアサートすることができる。非同期のシステムでは、確実に FLUSH# ピンが最初に処理されるように、SMI# ピンの少なくとも 1 クロック前に FLUSH# ピンがアサートされなければならない。

SMM を終了するときは（キャッシュを明示的にフラッシュするシステムの場合）、WBINVD 命令を実行してから SMM を終了し、キャッシュをフラッシュする必要がある。

注記

インテル Pentium プロセッサをベースとするシステムでは、SMM に移行する前に FLUSH# ピンを使用してキャッシュの内容のライトバックと無効化が実行されるが、プロセッサは少なくとも 1 つのキャッシュ・ラインをプリフェッチする。このプリフェッチは、フラッシュ・アクノレッジ・サイクルが実行され、引き続いて SMI# が認識され SMIACT# がアサートされたときに行われる。このシステムでは、インテル Pentium プロセッサに対してアクティブでない KEN# を返しても、これらのラインがキャッシュされないようにしなければならない。

13.5. SMI ハンドラの実行環境

プロセッサは、プロセッサの現行コンテキストをセーブした後、それ自体のコアレジスタを、表 13-2. に示されている値に初期化する。SMM に移行すると、制御レジスタ CR0 内の PE フラグと PG フラグがクリアされ、これによってプロセッサが実アドレスモードに似た環境に置かれる。SMM の実行環境と実アドレスモードの実行環境の相違点は、次のとおりである。

- アドレス指定可能な SMRAM アドレス空間の範囲は 0 ～ FFFFFFFFH（4G バイト）である。（制御レジスタ CR4 の PAE フラグによってイネーブルになる物理アドレス拡張は、SMM ではサポートされない。）
- 実アドレスモードに対するノーマルな 64K バイト・セグメント・リミットは、4G バイトに拡大されている。
- デフォルトのオペランド・サイズとアドレスサイズは 16 ビットに設定される。アドレス指定可能な SMRAM アドレス空間は、この設定によって、実アドレスモードのコードのリミットである 1M バイトに制限される。ただし、オペランド・サイズとアドレスサイズのオーバーライド・プリフィックスを使用すれば、1M バイトを超えるアドレス空間にアクセス可能になる。

表 13-2. SMM でのプロセッサ・レジスタの初期化

レジスタ	内容
汎用レジスタ	未定義
EFLAGS	00000002H
EIP	00008000H
CS セレクタ	4 ビット右シフトされた SMM ベース（デフォルト 3000H）
CS ベース	SMM ベース（デフォルト 30000H）
DS、ES、FS、GS、SS セレクタ	0000H
DS、ES、FS、GS、SS ベース	000000000H
DS、ES、FS、GS、SS リミット	0FFFFFFFFH
CR0	0 に設定された PE、EM、TS、PG の各フラグ。その他は修正なし。
CR4	0 にクリアされる。
DR6	未定義
DR7	00000400H

- 32 ビットのオペランド・サイズ・オーバーライド・プリフィックスを使用する場合は、4G バイトのアドレス空間のどこにでも near ジャンプや near コールを実行できる。ベースアドレスを形成する実アドレスモードのスタイルが原因で、far コールや far ジャンプでは、20 ビット（1M バイト）を超えるベースアドレスを持つセグメントに制御を転送できない。ただし、SMM 内のセグメント・リミットが 4G

バイトであるため、32 ビットのオペランド・サイズ・オーバーライド・プリフィックスを使用する場合は、1M バイトのリミットを超えるセグメントへのオフセットが可能になる。32 ビットのオペランド・サイズ・オーバーライド・プリフィックスを持たないプログラム制御転送では、EIP 値は下位 16 ビットに切り捨てられる。

- データとスタックは、4G バイトのアドレス空間のどこにでも配置できるが、1M バイトを超える位置に配置されている場合は、32 ビットのアドレス・サイズ・オーバーライドを使用したときだけにアクセスが可能になる。コード・セグメントの場合と同じように、データ・セグメントまたはスタック・セグメントのベースアドレスは、20 ビットを超えられない。

セグメント・レジスタ CS の値は、4 ビット右シフトされた SMBASE（3000H）に対するデフォルトの 30000H に自動的に設定される。EIP レジスタは 8000H に設定される。シフトされた CS 値（SMBASE）に EIP 値が追加されると、結果として得られたリニアアドレスが SMI ハンドラの最初の命令を指す。

他のセグメント・レジスタ（DS、SS、ES、FS、GS）は 0 にクリアされ、セグメント・リミットは 4G バイトに設定される。この状態では、SMRAM のアドレス空間を単一のフラットな 4G バイトのリニアアドレス空間として扱える。セグメント・レジスタに 16 ビット値がロードされた場合、その値は 4 ビット左シフトされ、セグメント・ベース（セグメント・レジスタの見えない部分）にロードされる。リミットと属性は修正されない。

プロセッサが SMM に移行すると、マスク可能ハードウェア割り込み、例外、NMI 割り込み、SMI 割り込み、A20M 割り込み、シングル・ステップ・トラップ、ブレークポイント・トラップ、INIT 操作は抑止される。SMM 実行環境が、割り込みテーブル、必要な割り込み / 例外ハンドラを提供し初期化する場合は、マスク可能ハードウェア割り込み、例外、シングル・ステップ・トラップ、ブレークポイント・トラップを SMM 内でイネーブルにできる（13.6. 節「SMM 内での例外と割り込み」を参照）。

13.6. SMM 内での例外と割り込み

プロセッサが SMM に移行すると、次の方法でハードウェア割り込みすべてがディスエーブルになる。

- EFLAGS レジスタの IF フラグがクリアされる。この結果、マスク可能ハードウェア割り込みの生成が抑止される。
- EFLAGS レジスタの TF フラグがクリアされる。この結果、シングル・ステップ・トラップがディスエーブルになる。
- デバッグレジスタ DR7 がクリアされる。この結果、ブレークポイント・トラップがディスエーブルになる。（この処置によって、SMRAM 内のコードまたはデータ

をオーバーレイするノーマルなアドレス空間内にデバッグ・ブレークポイントが設定された場合に、デバッガが誤ってSMMハンドラに割り込まなくなる。）

- NMI、SMI、A20Mの各割り込みが、内部SMMロジックによってブロックされる。（SMM内でNMIが処理される方法の詳細については、13.8. 節「SMM内でのNMI処理」を参照。）

SMM内にある場合でも、ソフトウェア起動の割り込みと例外が発生する可能性が依然として残っている。またIFフラグをセットして、マスク可能ハードウェア割り込みをイネーブルにすることができる。（INT *n*、INT 0、INT 3, またはBOUNDの各命令によって）ソフトウェア割り込みが起動されたり、例外が生成されることがないようなSMMコードを記述しておくことを推奨する。

SMMハンドラが割り込みと例外の処理を必要とする場合は、SMM割り込みテーブルと、必要な例外ハンドラと割り込みハンドラを、SMM内で作成して初期化する必要がある。割り込みテーブルが（LIDT命令を使用して）正しく初期化されるまで、例外やソフトウェア割り込みは、予測できないプロセッサの動作の原因となる。

SMM割り込みと例外の処理機能を設計するときに適用される制限項目は、次のとおりである。

- 割り込みテーブルは、リニアアドレス 0 に配置され、実アドレス・モード・スタイルの割り込みベクタ（CSとIPを含む4バイト）が入れられていなければならない。
- ベースアドレスを形成する実アドレスモードのスタイルが原因で、割り込みまたは例外が、20ビットを超えるベースアドレスを持つセグメントに制御を転送できない。
- 割り込みまたは例外は、16 ビット（64K バイト）を超えるセグメント・オフセットに制御を転送できない。
- 例外または割り込みが発生した場合は、戻りアドレス（EIP）の最下位 16 ビットだけがスタックにプッシュされる。割り込まれたプロシージャのオフセットが64Kバイトより大きい場合は、割り込み/例外ハンドラはそのプロシージャに制御を返すことができない。（この問題を解決する1つの方法は、ハンドラがスタック上での戻りアドレスを調整することである。）
- SMBASE の再配置機能は、SMI ハンドラが実行されている間に生成された割り込みと例外からプロセッサが戻る方法に影響を与える。例えば、SMBASEが1Mバイトを超えて再配置されていて、例外ハンドラが1Mバイト未満である場合は、SMIハンドラに正常に戻れなくなる。この問題を解決する1つの方法は、スタック上の16ビットの戻りアドレスから1Mバイトを超える戻りアドレスを計算するメカニズムを例外ハンドラに与えて、その後で32ビットのfarコールを使用して、割り込まれたプロシージャに戻ることである。

- SMI ハンドラがデバッグトラップ機能にアクセスする必要がある場合は、SMM でアクセスできるデバッグハンドラを利用できるようにして、デバッグレジスタ DR0 ～ DR3 の現在の内容を（あとのリストア用に）セーブしなければならない。その後で、デバッグレジスタ DR0 ～ DR3 と DR7 を適切な値で初期化しなければならない。
- SMI ハンドラがシングルステップのメカニズムにアクセスする必要がある場合は、SMM でアクセスできるシングル・ステップ・ハンドラを利用できるようにして、その後に EFLAGS レジスタ内の TF フラグをセットしなければならない。
- SMI 設計でプロセッサに対して、マスク可能ハードウェア割り込みやソフトウェアが生成した割り込みに（SMM にある間に）応答することが求められる場合は、SMM でアクセスできる割り込みハンドラを利用できるようにして、その後に（STI 命令を使用して）EFLAGS レジスタ内の IF フラグをセットしなければならない。SMM に移行してもソフトウェア割り込みはブロックされないため、割り込みをイネーブルにする必要はない。

13.7. 同期および非同期のシステム管理割り込みの管理

マルチプロセッサ・システムまたは HT テクノロジ対応システム向けのコーディングでは、SMI ハンドラが同期 SMI（I/O 命令の実行中にトリガされる）と非同期 SMI を区別することが常に可能とは限らない。これらの 2 つのイベントを区別しやすいように、SMM 状態保存マップに追加状態情報が追加された。

30004H またはそれ以上の SMM リビジョン ID を持つプロセッサは、以下に説明する追加状態情報を持つ。

13.7.1. I/O 状態の実装

拡張 SMM 状態保存マップ内の 1 ビット（IO_SMI）は、I/O 命令の正常終了の直後または REP I/O 命令の正常な反復の後に SMI が発生した場合にのみセットされる（ただし、正常とはプロセッサの観点から見た概念であり、対応するプラットフォームの機能には必ずしも該当しない）。IO_SMI ビットがセットされることは、それに対応する SMI が同期 SMI であることの強い指標になる。この場合、SMM 状態保存マップは、I/O 操作のポートアドレスも示す。IO_SMI ビットおよび I/O ポートアドレスと、プラットフォームが記録した情報を組み合わせて使用すれば、その SMI が実際に同期 SMI であることを確認できる。

なお、IO_SMI ビット単独では、SMI が同期 SMI であることの強い指標にはなるが、それを保証するものではない。これは、I/O 命令の後に、非同期 SMI が偶然の一致で発生する可能性があるためである。このような場合にも、SMM 状態保存マップ内で IO_SMI ビットがセットされる。

I/O 命令の特性を示す情報は、SMM 状態保存マップ内の 2 つの場所に保存される（表 13-3.）。ただし、IO_SMI ビットは、それ以外の I/O 情報フィールドの有効ビットとしても機能する。IO_SMI ビットがセットされていない場合、これらの I/O 情報フィールドの内容は未定義である。

表 13-3. SMM 状態保存マップ内の I/O 命令情報

状態（SMM リビジョン ID: 30004H またはそれ以上）	フォーマット				
	31 16	15 8	7 4	3 1	0
I/O 状態フィールド SMRAM オフセット 7FA4	I/O ポート	予約済み	I/O タイプ	I/O 長	IO_SMI
	31				0
I/O メモリ・アドレス・フィールド SMRAM オフセット 7FA0	I/O メモリアドレス				

IO_SMI がセットされている場合、その他のフィールドは次のように解釈される。

- I/O 長
  - 001 – バイト
  - 010 – ワード
  - 100 – ダブルワード
- I/O 命令タイプ

表 13-4. I/O 命令タイプのエンコーディング

命令	エンコーディング
IN 即時	1001
IN DX	0001
OUT 即時	1000
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

13.8. SMM 内での NMI 処理

NMI 割り込みは、SMI ハンドラに移行した時点でブロックされる。SMI ハンドラの実行中に NMI 要求が発生した場合は、その要求はラッチされ、プロセッサが SMM を終了した後に処理される。SMI ハンドラの実行中には、ただ 1 つの NMI 要求がラッチされる。プロセッサが RSM 命令を実行したときに NMI 要求がペンディングであれば、割り込まれたコード・シーケンスの次の命令が実行される前に NMI が処理される。これは、SMI が発生する前に NMI はブロックされていなかったと想定している。SMI が発生する前に NMI がブロックされていた場合は、RSM の実行後にそれらはブロックされる。

プロセッサが SMM モードに入ると、NMI 要求はブロックされるが、IRET/IRETD 命令を実行することで、ソフトウェアによって NMI 要求をイネーブルにできる。SMM ハンドラは、NMI 割り込みの使用を要求する場合は、IRET/IRETD 命令を実行するためのダミー割り込みサービスルーチンを起動しなければならない。IRET/IRETD 命令が実行されると、NMI 割り込み要求は、SMM モード外で処理される場合と同じ「リアルモード」方式でサービスされる。

SMI ハンドラが NMI ハンドラ内でネストしている場合は、特殊な状況となり、別の NMI が発生する。NMI 割り込みの処理中に NMI 割り込みがディスエーブルになるため、通常の場合は、IRET 命令によって NMI 割り込みが 1 つずつ処理されて動作を完了する。NMI ハンドラを実行している間にプロセッサが SMM に移行すると、プロセッサは SMRAM 状態保存マップをセーブするが、NMI 割り込みをディスエーブル状態に保つための属性はセーブしない。可能性として、NMI ハンドラの前の実行が完了していないときでも、（SMM にある間または SMM の終了時に）NMI がラッチされ、SMM の終了時に NMI が処理されることがある。この場合、1 つまたは複数の NMI を、最初の NMI ハンドラにネストできる。NMI 割り込みハンドラでは、この可能性を考慮に入れておく必要がある。

また、インテル® Pentium® プロセッサでは、トラップまたはフォルトハンドラを起動する例外によって、SMM の内部からの NMI 割り込みをイネーブルにできる。この動作はインテル Pentium プロセッサに固有のインプリメンテーションで、IA-32 アーキテクチャに必須の要素ではない。

13.9. SMM 内での x87 FPU ステートのセーブ

例えば、0 ボルトの中断状態に移行するときにシステムメモリの電源を切断する前のような場合には、SMM にある間に x87 FPU のステートをセーブしなければならない。この操作を実行するときに、関連する x87 FPU ステートの情報が失われないように注意しなければならない。このタスクを実行する上で最も安全な方法は、x87 FPU ステートをセーブする前に、プロセッサを 32 ビット保護モードに置くことである。この理由を次に述べる。

FSAVE 命令は、x87 FPU コンテキストを 4 種のフォーマットのいずれかでセーブする。セーブ・フォーマットは、FSAVE が実行されているときにプロセッサがどのモードに置かれているかによって決定される（『IA-32 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』の図 8-9. ～図 8-12. を参照）。SMM にあるとき、デフォルトでは、16 ビットの実アドレスモードのフォーマットが使用される（図 8-12. を参照）。プロセッサが 16 ビットの実アドレスモード以外のモードにある間に SMI 割り込みが発生すると、FSAVE 命令と FRSTOR 命令は、x87 FPU 関連の全情報のセーブとリストアができなくなる。この状態で、割り込まれたプログラムが再開されると、誤動作が発生する場合がある。この問題が発生しないようにするには、FSAVE 命令と FRSTOR 命令を実行するときに、プロセッサを 32 ビットの保護モードに設定しなければならない。

SMI ハンドラから保護モードに移行して x87 FPU 状態をセーブしたりリストアするときには、次のガイドラインに従わなければならない。

- CPUID 命令を使用して、プロセッサが x87 FPU を内蔵しているようにする。
- 32 ビット・コード・セグメントを SMRAM 空間内に作成する。この空間には、FSAVE 命令と FRSTOR 命令を使用してそれぞれ x87 FPU のセーブとリストアを行うプロシージャやルーチンが入れられる。また 32 ビット・コード・セグメントに対して適切なコード・セグメント・ディスクリプタ（D ビットは 1 にセット）を持つ GDT も、SMRAM に置く必要がある。
- x87 FPU ステートをセーブしたりリストアするために、SMI ハンドラによって呼び出すことができるプロシージャまたはルーチンを書き込む。このプロシージャは、次の操作を実行しなければならない。
  - プロセッサを 32 ビット保護モードに設定する（9.9.1. 項「保護モードへの切り替え」を参照）。
  - x87 FPU セーブとリストアの各プロシージャが入れられている 32 ビット・コード・セグメントへの far ジャンプを実行する。
  - SMI ハンドラに戻る前に、プロセッサを 16 ビットの実アドレスモードに戻す（9.9.2. 項「実アドレスモードへの再切り替え」を参照）。

SMI ハンドラは、x87 FPU ステートがセーブされた後に保護モードで実行を継続できる。また、保護モードから、割り込まれたプログラムに安全に戻れる。ただし、主に 16 ビットまたは 32 ビットの実アドレスモードでハンドラを実行することを推奨する。

13.10.SMM リビジョン識別子

SMM リビジョン識別子フィールドを使用して、SMM のバージョンと、プロセッサがサポートする SMM の拡張機能を示すことができる（図 13-2. 参照）。SMM リビジョン識別子は、SMM に移行しているときに書き込まれ、オフセット 7EFCH の SMRAM 空間内で検査できる。SMM リビジョン識別子の下位ワードは、SMM のベース・アーキテクチャのバージョンを表している。

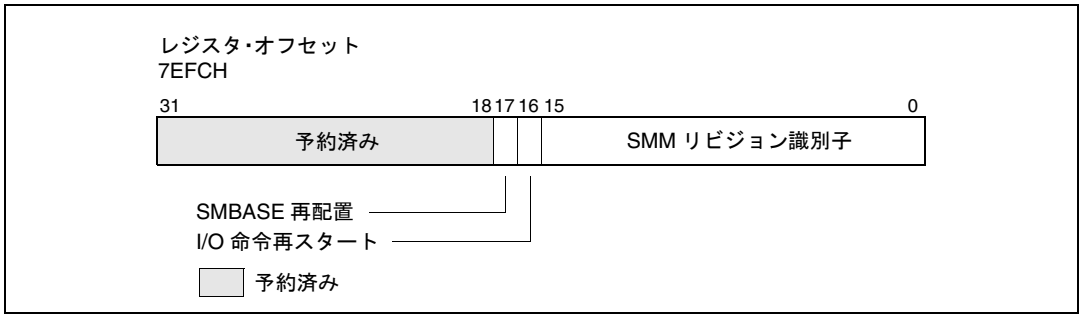


図 13-2. SMM リビジョン識別子

SMM リビジョン識別子の上位ワードは、利用可能な拡張機能を示している。I/O 命令の再スタートフラグ（ビット 16）がセットされている場合は、プロセッサは I/O 命令の再スタートをサポートする（13.13. 節「I/O 命令の再スタート」を参照）。SMBASE 再配置フラグ（ビット 17）がセットされている場合は、SMRAM ベースアドレスの再配置がサポートされる（13.12. 節「SMBASE の再配置」を参照）。

13.11.自動 HALT 再スタート

プロセッサが SMI を受け取ったときに、(HLT 命令が事前に実行されているために) プロセッサが HALT 状態にある場合、プロセッサは、この事実をセーブ済みプロセッサ状態にある自動 HALT 再スタートフラグに記録する（図 13-3. を参照）。(このフラグは、SMRAM の状態保存領域にあるオフセット 7F02H とビット 0 に配置されている。)

SMM に移行した時点でプロセッサが自動 HALT 再スタートフラグをセットする場合 (つまりプロセッサが HALT 状態にあったときに SMI が発生したことを示している場合)、SMI ハンドラには次の 2 つのオプションがある。

- 自動HALT再スタートフラグをセットされたままにしておく。これによって、RSM命令に対して、プログラム制御をHLT命令に返すように指示が出される。この結果、SMIを処理した後でプロセッサはHALT状態に再移行する。（これがデフォルト動作である。）
- 自動 HALT 再スタートフラグをクリアする。この場合、RSM 命令に対して、HLT命令の直後の命令にプログラム制御を返すように指示が出される。

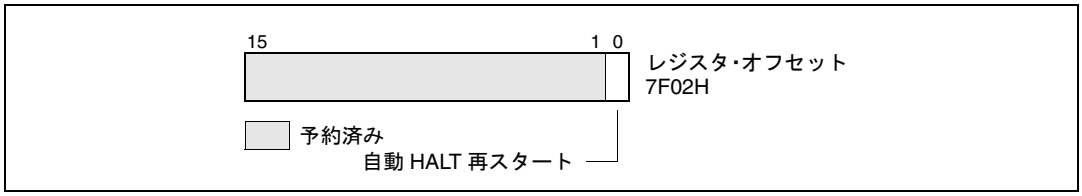


図 13-3. 自動 HALT 再スタート・フィールド

表13-5.に、上記の各オプションを要約したものを示す。SMIを受け取ったときにプロセッサがHALT状態にない場合（自動HALT再スタートフラグがクリアされている場合）にフラグを1にセットすると、RSM命令が実行されたときに、予測できない動作が発生することに注意しなければならない。

表 13-5. 自動 HALT 再スタートフラグ値

SMM に移行後のフラグ値	SMM 終了時のフラグ値	SMM 終了時のプロセッサの動作
0	0	割り込まれたプログラムやタスク内の次の命令に戻る。
0	1	予測不可能
1	0	HLT 命令後の次の命令に戻る。
1	1	HALT 状態に戻る。

HLT 命令が再スタートすると、（HLT 命令が内部キャッシュにない場合は）プロセッサが、HLT 命令をフェッチするメモリアクセスを生成して、HLT バス・トランザクションを実行する。この動作の結果、同じHLT命令に対して複数のHLTバス・トランザクションが発生する。

13.11.1. SMM 内での HLT 命令の実行

EFLAGS レジスタの IF フラグをセットして割り込みをイネーブルにしていない限り、SMMにある間にHLT命令を実行してはならない。プロセッサがSMMにある間にHALT状態になった場合、プロセッサをHALT状態から解除できる唯一のイベントは、マスク可能ハードウェア割り込みか、またはハードウェア・リセットだけである。

13.12.SMBASE の再配置

SMRAM のデフォルトのベースアドレスは、30000H である。この値は、SMBASE レジスタと呼ばれる内部プロセッサ・レジスタ内に入れられる。オペレーティング・システムまたはエグゼクティブは、保存状態マップ内の（オフセット 7EF8H にある）SMBASE フィールドを新しい値に設定すると、SMRAM を再配置できる（図 13-4. を参照）。RSM 命令は、SMM を終了するたびに、SMBASE フィールド内のこの値を内部 SMBASE レジスタに再ロードする。それ以降のすべての SMI 要求は、新しい SMBASE 値を使用して、（SMBASE + 8000H にある）SMI ハンドラの開始アドレスと、（SMBASE + FE00H ～ SMBASE + FFFFH の）SMRAM 状態保存領域を見付ける。（プロセッサは、RESET 時に内部 SMBASE レジスタの値を 30000H にリセットするが、INIT 時には値を変更しない。）

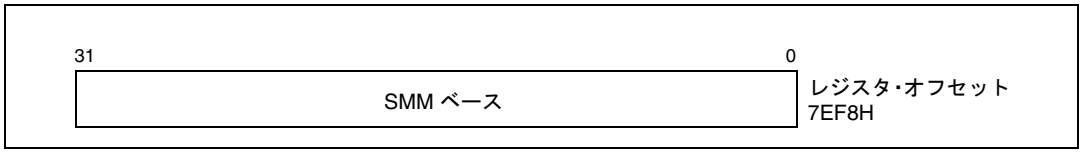


図 13-4. SMBASE 再配置フィールド

マルチプロセッサ・システムでは、初期化ソフトウェアはそれぞれのプロセッサに対する SMBASE 値を調整し、各プロセッサの SMRAM 状態保存領域がオーバーラップしないようにしなければならない。（インテル® Pentium® プロセッサと Intel486™ プロセッサでは、SMBASE 値を 32K バイトの境界上にアライメントを合わせなければならない。そうしないとプロセッサは、RSM 命令の実行中にシャットダウン状態になる。）

SMM リビジョン識別子フィールド内で SMBASE 再配置フラグがセットされている場合は、SMBASE が再配置できることを示している（13.10. 節「SMM リビジョン識別子」を参照）。

13.12.1. 1M バイトを超えるアドレスへの SMRAM の再配置

SMM では、セグメント・ベース・レジスタの更新は、セグメント・レジスタの値を変更して行う。セグメント・レジスタには 16 ビットだけしか入れられないため、セグメント・ベース・アドレスには 20 ビットしか使用できない（セグメント・レジスタは 4 ビット左シフトされて、セグメント・ベース・アドレスを決定する）。SMRAM が 1M バイトを超えるアドレスに再配置された場合は、実アドレスモードで動作するソフトウェアは、セグメント・レジスタを初期化して SMRAM ベースアドレス（SMBASE）を指せなくなる。

32 ビットのアドレス・サイズ・オーバーライド・プリフィックスを使用して正しいアドレスへのオフセットを生成すれば、SMRAM にアクセスできる。例えば、SMBASE

が（16Mバイト境界のすぐ下にある）FFFFFFHに再配置された状態にあって、しかもDS、ES、FS、GSの各レジスタが0Hに初期化されたままである場合は、次の例に示すように、32ビットのディスプレースメント・レジスタを使用することで、SMRAMのデータにアクセスできる。

```
mov    esi,00FFxxxxH; 64K segment immediately below 16M
mov    ax,ds:[esi]
```

1Mバイトを超える境界に配置されたスタックにも、同じ方法でアクセスできる。

13.13.I/O 命令の再スタート

SMMリビジョン識別子フィールド内のI/O命令再スタートフラグがセットされている場合（13.10.節「SMMリビジョン識別子」を参照）、I/O命令再スタート・メカニズムがプロセッサで使用できる。このメカニズムを使用すると、SMMモードから戻った時点で、割り込まれたI/O命令を再実行できる。例えば、電源を切断されたI/Oデバイスにアクセスするために I/O 命令を使用する場合、このデバイスをサポートしているチップセットによってアクセスを遮断できる。またSMI#をアサートして応答できる。この処置によって、SMIハンドラが起動されてデバイスに電源が投入される。SMIハンドラから戻った時点で、I/O 命令再スタート・メカニズムを使用して、SMI 発生の原因となったI/O命令を再実行できる。

（SMM 状態保存領域内のオフセット 7F00H にある）I/O 命令再スタート・フィールド（図 13-5. を参照）によって、I/O 命令再スタートが制御される。RSM 命令が実行されるときにこのフィールドに値 FFH が入っている場合、SMI 要求を受け取った I/O 命令を指すようにEIPレジスタが修正される。その後、プロセッサはSMIによってトラップされていたI/O命令を自動的に再実行する。（プロセッサは、命令の再実行がコヒーレンシを保って処理されるように必要なマシン状態をセーブする。）

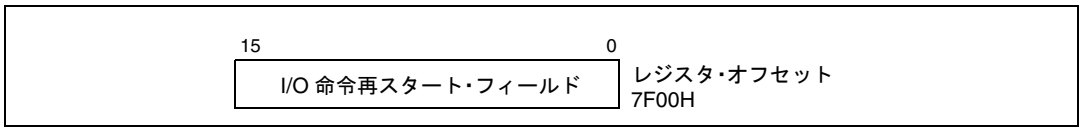


図 13-5. I/O 命令再スタート・フィールド

RSM命令が実行されたときに、I/O命令再スタート・フィールドに値00Hが入っている場合は、プロセッサはI/O命令の後に続く命令を使用してプログラムの実行を開始する。（リピート・プリフィックスを使用している場合は、次に続く命令がリピートループ内にある次のI/O命令になる可能性がある。）デフォルトでは、割り込まれたI/O命令は再実行されない。またプロセッサは、SMMへの移行時に、I/O命令再スタート・フィールドを00Hに自動的に初期化する。表13-6.に、I/O命令再スタート・フィールドの状態を要約したものを示す。

表 13-6. I/O 命令再スタート・フィールドの値

SMM に移行後のフラグ値	SMM 終了時のフラグ値	SMM 終了時のプロセッサの動作
00H	00H	トラップされていた I/O 命令を再実行しない。
00H	FFH	トラップされていた I/O 命令を再実行する。

I/O 命令再スタート・メカニズムでは、SMI の原因を指示しないことに注意する。SMI の原因を判定するためにプロセッサの状態を検査し、I/O 命令が割り込まれたのかどうか、また SMM を終了したときに割り込まれた I/O 命令を再スタートさせるかどうかを決定するのは、SMI ハンドラが責任を持って行わなければならない。I/O 命令以外の境界上で SMI 割り込みが通知された場合、RSM 命令を実行する前に I/O 命令再スタート・フィールドを FFH に設定すると、結果としてプログラム・エラーが発生する可能性がある。

13.13.1. I/O 命令再スタートを使用しているときの連続 SMI 割り込み

プロセッサが I/O 命令境界上で発生した SMI 割り込みを処理している間に、SMI 割り込みが通知された場合、プロセッサは、最初に割り込まれた I/O 命令を再スタートする前に、新しい SMI 要求を処理する。別の SMI ハンドラから戻る前に I/O 命令再スタート・フィールドが FFH に設定された場合、EIP は最初に割り込まれた I/O 命令とは異なるアドレスを指す。この結果、プログラム・エラーが発生する可能性がある。こういう状況を避けるために、SMI ハンドラは、I/O 命令再スタートが使用されているときに、連続 SMI 割り込みの発生が認識できなければならない。また SMI ハンドラの 2 回目の起動から戻る前に、ハンドラによって I/O 命令再スタート・フィールドが 00H に設定されなければならない。

13.14. SMM でマルチプロセッサを使用する際の注意点

マルチプロセッサ・システムを設計する場合は、次の点に注意しなければならない。

- マルチプロセッサ・システム内では、どのプロセッサも SMM に応答できる。
- 各プロセッサには、固有の SMRAM 空間が必要である。この空間は、システムメモリ内または個別の RAM 内に配置できる。
- 異なるプロセッサに対する複数の SMRAM を、同じメモリ空間内でオーバーラップできる。唯一の規定項目は、各プロセッサには固有の状態保存領域と動的データ記憶領域が必要なことである。（また、インテル® Pentium® プロセッサと Intel486™ プロセッサでは、SMBASE アドレスは 32K バイトの境界上に配置されなければならない。）コードと静的データは、各プロセッサ間で共有できる。SMRAM 空間のオーバーラップは、P6 ファミリ・プロセッサを使用すると、さらに効率的に実行できる。なぜなら、32K バイト境界上に SMBASE アドレスを置く必要がないためである。

- SMIハンドラは、プロセッサごとにSMBASEを初期化する必要がある。
- プロセッサは、プロセッサの SMI# ピンを通じてローカル SMI に応答できる。または、APICインターフェイスを通じて受け取ったSMIに応答できる。APICインターフェイスは、SMIを別々のプロセッサに配布できる。
- 複数のプロセッサをSMM内で同時に実行できる。
- インテル Pentium プロセッサがデュアルプロセッサ（DP）モードで動作している場合、SMIACT# ピンはMRMプロセッサによってのみドライブされる。またADS#によってサンプリングされなければならない。詳細については、『Pentium® Processor Family User's Manual, Volume 1』の第14章を参照。

SMRAM状態保存マップがSMBASEに関連して固定されているため、SMMは再入不可能である。SMMモードで同時に複数のプロセッサをサポートする必要がある場合は、それぞれのプロセッサは、専用のSMRAM空間を持っていなければならない。これは、SMBASE再配置機能を使用して実現できる（13.12.節「SMBASEの再配置」を参照）。

13.15.拡張版 Intel SpeedStep® テクノロジ

インテル® Pentium® Mプロセッサ上の拡張版Intel SpeedStep®テクノロジは、パフォーマンス状態の移行によって、プロセッサの消費電力を効率的に管理する。プロセッサのパフォーマンス状態は、異なるクロック周波数に対応する個別の動作点として定義される。

インテルPentium Mプロセッサ上の拡張版Intel SpeedStepテクノロジは、以前の世代のIntel SpeedStepテクノロジとは以下の2つの基本的な点で異なる。

- モデル固有レジスタの使用による、プロセッサ内の制御機構とソフトウェア・インターフェイスの集中化
- ハードウェア・オーバーヘッドの低減。これにより、パフォーマンス状態の移行がより頻繁に行える。

以前の世代の Intel SpeedStep テクノロジでは、プロセッサをディープスリープ状態にする必要があり、パフォーマンス状態の移行中はバスマスタ転送が先延ばしにされる。拡張版 Intel SpeedStep テクノロジでは、パフォーマンス状態の移行は、目標となる新しい周波数に対する個別の移行になる。

拡張版Intel SpeedStepテクノロジのサポートは、CPUID命令によってECX機能ビット07 で確認できる。拡張版 Intel SpeedStep テクノロジをイネーブルにするには、IA32_MISC_ENABLE MSR のビット16をセットする必要がある。リセット時には、IA32_MISC_ENABLE MSRのビット16はクリアされる。

13.15.1. パフォーマンス状態の移行を開始するためのソフトウェア・インターフェイス

パフォーマンス状態の移行を開始するには、MSR_PERF_CTL レジスタに 16 ビット値を書き込む。前回の移行の途中であれば、新しい値への移行はその後に有効になる。

MSR_PERF_CTL を読み取ると、最新の目標となった動作点を確認できる。現在の動作点は、MSR_PERF_STATUS から読み取れる。MSR_PERF_STATUS は動的に更新される。

有効な動作点を定義する 16 ビット・エンコーディングは、モデル固有である。アプリケーションやパフォーマンス・ツールが MSR_PERF_CTL または MSR_PERF_STATUS を使用することは予想されていない。アプリケーションやパフォーマンス・ツールは、これらのレジスタを予約済みとして扱う必要がある。性能モニタリング・ツールは、モデル固有のイベントにアクセスし、状態の移行の発生回数を報告できる。

13.16. 温度監視と保護

IA-32 アーキテクチャは、IA-32 プロセッサの温度を監視し消費電力を制御するための機構を 3 つ備えている。

1. プロセッサのコア温度が事前に設定された制限を超えた場合にプロセッサの実行を強制的に停止する、**突発的シャットダウン・ディテクタ**。
2. 事前に決められた温度制限を保持するために消費電力を強制的に減らす、**自動温度監視機構**。
3. **ソフトウェアによって制御されるクロック調整機構**。この機構によって、オペレーティング・システムは、省電力ポリシーを適用して IA-32 プロセッサの消費電力を軽減できる。この機能は、自動温度監視機構による消費電力の低減に加えて作用する。

第 1 の機構は、ソフトウェアには認識できない。第 2 および第 3 の機構は、EAX=1 で CPUID を実行したときに返されるプロセッサ機能情報を使用して、ソフトウェアが認識できる。

第 2 の機構（自動温度監視機構）は、2 つの動作モードを備えている。第 1 のモードは、クロック・デューティ・サイクルを調整する。第 2 のモードは、プロセッサのクロック周波数を変更する。プロセッサ・コアの温度は、両方のモードを使用して制御される。

第 3 の機構は、プロセッサのクロック・デューティ・サイクルを調整する。図 13-6. に示すように、「デューティ・サイクル」という用語は、クロック信号の実際のデューティ・サイクルを指すのではなく、クロック信号がプロセッサ・チップを駆動できる

時間を指している。クロック停止機構を使用してプロセッサのクロックの間隔を制御することで、プロセッサの消費電力を調整できる。

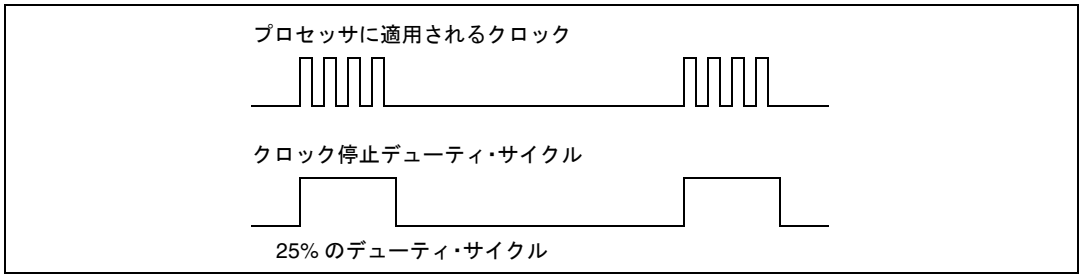


図 13-6. クロック停止機構によるプロセッサ調整

13.16.1. 突発的シャットダウン・ディテクタ

P6 ファミリ・プロセッサには、突発的シャットダウン・ディテクタとして機能する温度センサが組み込まれている。この突発的シャットダウン・ディテクタは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、およびインテル® Pentium® M プロセッサにも搭載されている。この機能は常にイネーブルになる。プロセッサ・コアの温度が工場出荷時の設定レベルに達すると、センサがトリップし、次のリセットサイクルが経過するまでプロセッサの実行がホルト状態になる。

13.16.2. 温度モニタ

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Pentium® M プロセッサには第 2 の温度センサが組み込まれている。これは、プロセッサのコア温度が、推奨される温度設計に対応するレベルを超えたときにトリップするよう出荷時に調整されている。第 2 のセンサのトリップ温度も調整されており、突発的シャットダウン・ディテクタに割り当てられた温度よりも低い温度になるように設定されている。

13.16.2.1.温度モニタ 1

インテル® Pentium® 4 プロセッサは、TM1（温度モニタ 1）と呼ばれる機構と第 2 の温度センサを組み合わせて、プロセッサ・コアの温度を制御する。TM1 は、プロセッサ・クロックのデューティ・サイクルを調整することで、プロセッサの温度を制御する。デューティ・サイクルの調整は、プロセッサ・モデルに固有である。なお、この機能にはプロセッサの STPCLK# ピンは使用されない。クロック停止回路は内部で制御される。

TM1 のサポートは、CPUID の EDX 機能ビット 29 によって示される。

TM1 をイネーブルにするには、IA32_MISC_ENABLE 内の温度モニタ・イネーブル・フラグ（ビット 3）をセットする必要がある [付録 B「モデル固有レジスタ（MSR）」」を参照]。電源投入またはリセットの後、このフラグはクリアされ、TM1 はディスエーブルになる。BIOS は、いずれか一方の自動温度監視モードをイネーブルにする必要がある。オペレーティング・システムやアプリケーションは、これらの機構の動作をディスエーブルにしてはならない。

13.16.2.2.温度モニタ 2

温度モニタ 2（TM2）と呼ばれる追加の自動温度保護機構は、インテル® Pentium® M プロセッサで導入され、インテル® Pentium® 4 プロセッサ・ファミリの最近のモデルにも組み込まれている。TM2 は、プロセッサの動作周波数および電圧を下げることで、プロセッサ・コアの温度を制御する。同じ消費電力レベルでは、TM2 は TM1 より高いパフォーマンス・レベルを提供する。

TM2 は、TM1 と同じ温度センサによってトリガされる。IA-32 プロセッサ・ファミリの TM2 のサポートは、CPUID の ECX 機能ビット 8 によって示される。TM2 をイネーブルにする機構の実装方法は、（CPUID シグネチャのファミリ・エンコーディング値が異なる）IA-32 プロセッサ・ファミリ間では異なる場合があるが、同じ IA-32 プロセッサ・ファミリ内では共通である。

インテル Pentium M プロセッサの場合 : MSR_THERM2_CTL レジスタの TM_SELECT フラグ（ビット 16）が 1 にセットされ、IA32_MISC_ENABLE レジスタのビット 3 が 1 にセットされている場合は、TM2 がイネーブルになっている。

電源投入またはリセットの後、TM_SELECT フラグはクリアされる。BIOS は、TM1 または TM2 のいずれかをイネーブルにする必要がある。オペレーティング・システムやアプリケーションは、TM1 または TM2 をイネーブルにする機構をディスエーブルにしてはならない。IA32_MISC_ENABLE レジスタのビット 3 がセットされ、MSR_THERM2_CTL レジスタの TM_SELECT フラグがクリアされている場合は、TM1 がイネーブルになっている。

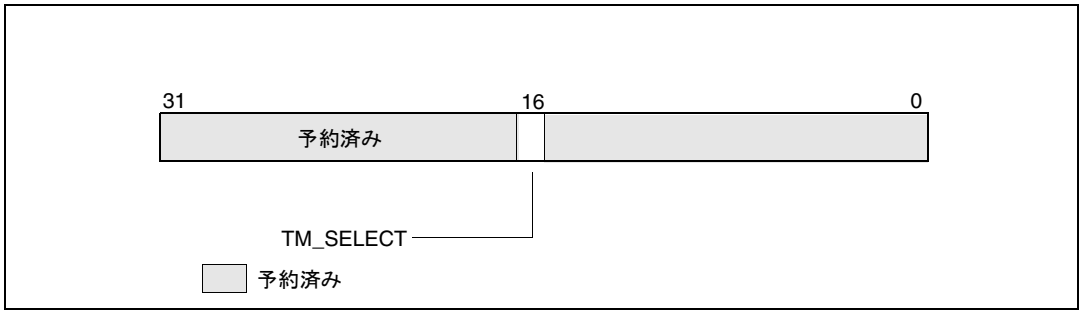


図 13-7. インテル® Pentium® M プロセッサの MSR_THERM2_CTL レジスタ

インテル Pentium 4 プロセッサの場合 : TM2 のサポートは、インテル Pentium M プロセッサの場合と同様に、CPUID 命令の ECX ビット 8 を使用して報告される。しかし、TM2 をイネーブルにするためのインターフェイスは多少異なる。TM2 をサポートするインテル Pentium 4 プロセッサの場合、TM2 をイネーブルにするには、IA32_MISC_ENABLE レジスタのビット 13 を 1 にセットする必要がある。

TM2 がトリガされた後の移行の目標となる動作周波数および電圧は、MSR_THERM2_CTL のビット 15:0 に書き込まれる値によって指定される。電源投入またはリセットの後、BIOS は、2 つの温度監視機構のうち少なくとも 1 つをイネーブルにする必要がある。プロセッサが TM1 と TM2 の両方をサポートしている場合、BIOS は、TM1 の代わりに TM2 をイネーブルにすることができる。オペレーティング・システムやアプリケーションは、TM1 または TM2 をイネーブルにする機構をディスエーブルにしてはならない。また、オペレーティング・システムやアプリケーションは、MSR_THERM2_CTL レジスタのビット 15:0 の値を変更してはならない。

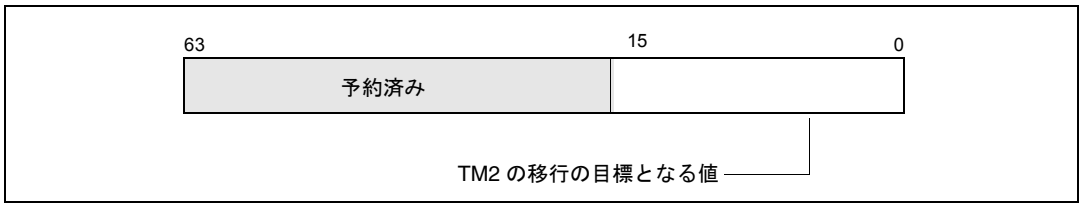


図 13-8. TM2 をサポートするインテル® Pentium® 4 プロセッサの MSR_THERM2_CTL レジスタ

13.16.2.3.パフォーマンス状態の移行と温度監視機能

温度モニタ（TM1/TM2）の温度制御回路（TCC）がアクティブになっている場合、MSR_PERF_CTL への書き込みは、目標となる新しい動作点に次のような影響を与える。

- TM1 がイネーブルになっており、TCC が関与している場合は、TCC が解除される前にパフォーマンス状態の移行を開始できる。
- TM2 がイネーブルになっており、TCC が関与している場合は、TCC が解除された後で、MSR_PERF_CTL への書き込みによって指定されたパフォーマンス状態の移行が開始される。

13.16.2.4.温度ステータス情報

温度モニタ（TM1/TM2）を起動する温度センサのステータスは、IA32_THERM_STATUS MSR（図 13-9. を参照）内の温度ステータス・フラグ（ビット 0）および温度ステータス・ログ・フラグ（ビット 1）により示される。

これらのフラグの機能は、以下のとおりである。

- **温度ステータス・フラグ、ビット 0。**このフラグがセットされているときは、プロセッサの現在のコア温度が温度モニタのトリップ温度になっており、プロセッサの消費電力がTM1またはTM2のいずれかイネーブルになっている方を通じて減らされていることを示す。このフラグがクリアされているときは、コア温度が温度モニタトリップ温度より低いことを示す。このフラグは、読み取り専用である。
- **温度ステータス・ログ・フラグ、ビット　1。**このフラグがセットされているときは、最後に電源を投入またはリセットしたあとで、またはソフトウェアがこのフラグを最後にクリアしたあとで、温度センサがトリップしていることを示している。このフラグは、スティッキー・ビットである。一度セットすると、ソフトウェアによってクリアされるか、または、プロセッサの電源が投入またはリセットされるまで、セットされたままになる。デフォルトではクリアされている。

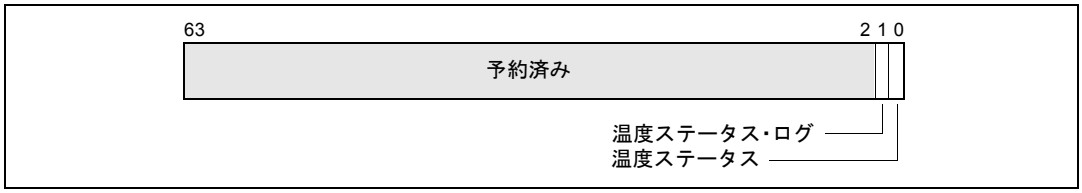


図 13-9. IA32_THERM_STATUS MSR

第2の温度センサがトリップされると、温度モニタ（TM1/TM2）は、最少周期（1ms）を保証される。温度モニタは、プロセッサのコア温度が温度センサのプリセットされたトリップ温度より下がるまで、ヒステリシスを考慮しながら保証される。

プロセッサがクロック停止ステートにある間は、割り込みがあってもプロセッサは中断されない。このように、割り込みが受け付けられないと、割り込みの待ち時間が増加する。しかし、これによって、割り込みが失われない。未処理の割り込みは、クロックの調整が完了するまで保留状態に置かれる。

温度モニタは、温度センサがトリップした時点でプロセッサに割り込みをかけるようにプログラムすることができる。この割り込みの伝達モードであるマスクとベクタは、ローカルAPICのLVT（8.5.1.項「ローカル・ベクタ・テーブル」を参照）の温度エントリを介してプログラムすることができる。IA32_THERM_INTERRUPT MSR（図13-10. を参照）内の低温割り込みイネーブル・フラグ（ビット 0）と高温割り込みイネーブル・フラグ（ビット 1）は、割り込みが生成されるとき、すなわち、トリップ点をまたいで温度が上昇または下降するときに制御される。

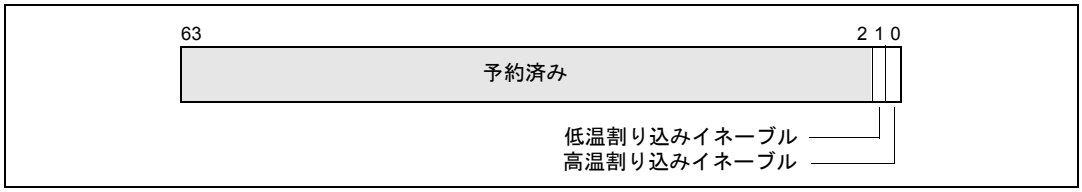


図 13-10. IA32_THERM_INTERRUPT MSR

- **低温割り込みイネーブル・フラグ、ビット 1。**このフラグがセットされると、高温から低温へ移行するときに割り込みが生成される。フラグがクリアされていると、割り込みは生成されない。
- **高温割り込みイネーブル・フラグ、ビット 0。**このフラグがセットされると、低温から高温へ移行するときに割り込みが生成される。フラグがクリアされていると、割り込みは生成されない（R/W）。

温度モニタ割り込みは、温度 LVT エントリでマスクできる。電源を投入またはリセットすると、IA32_THERM_INTERRUPT MSR の低温割り込みイネーブル・フラグと高温割り込みイネーブル・フラグはクリアされ（割り込みがディスエーブルになる）、温度 LVT エントリは割り込みをマスクするようにセットされる。この割り込みは、オペレーティング・システム・コードまたはシステム管理モード（SMM）コードで処理する必要がある。

温度監視機構の動作により、プロセッサ内部の高分解能タイマ（タイムスタンプ・カウンタ）のクロックレートに影響することはない。

13.16.3. ソフトウェア制御クロック調整

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Pentium® M プロセッサでは、ソフトウェア制御クロック調整もサポートしている。これによって、オペレーティング・システムは消費電力管理ポリシーを導入して、プロセッサの消費電力を減らせる。この場合、クロック停止デューティ・サイクルは、IA32_CLOCK_MODULATION_MSR（図 13-11. を参照）を通じてソフトウェアにより制御される。

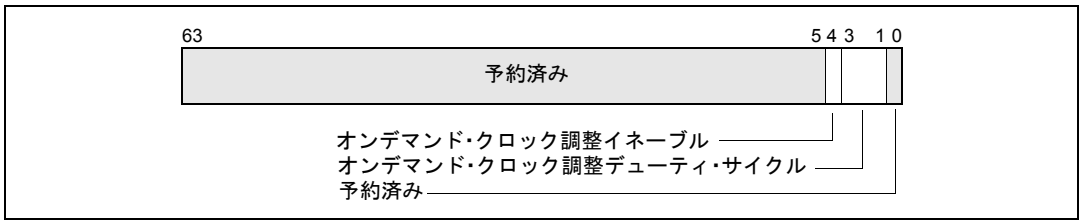


図 13-11. IA32_CLOCK_MODULATION_MSR

IA32_CLOCK_MODULATION_MSR には、以下のフラグとフィールドが含まれている。これを使用すると、ソフトウェア制御クロック調整機能をイネーブルにしたり、クロック調整デューティ・サイクルを選択できる。

- **オンデマンド・クロック調整イネーブル、ビット 4。**これをセットすると、オンデマンドのソフトウェア制御クロック調整機能がイネーブルになる。これをクリアすると、ソフトウェア制御クロック調整機能がディスエーブルになる。
- **オンデマンド・クロック調整デューティ・サイクル、ビット 1 ～ 3。**オンデマンド・クロック調整デューティ・サイクル（表 13-7. を参照）を選択するのに使用する。このフィールドは、オンデマンド・クロック調整イネーブル・フラグがセットされていないと、有効にならない。

オンデマンド・クロック調整機構は、（温度モニタと同様に）プロセッサのクロック停止回路を内部的に制御し、クロック信号を調整する。STPCLK# ピンはこの機能では使用されない。

表 13-7. オンデマンド・クロック調整デューティ・サイクル・フィールドのエンコーディング

デューティ・サイクル・フィールドのエンコーディング	デューティ・サイクル
000B	予約済み
001B	12.5%（デフォルト）
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

オンデマンド・クロック調整機構を利用して、プロセッサの消費電力を管理することができる。消費電力管理ソフトウェア上で IA32_CLOCK_MODULATION_MSR に書き込みを行うことで、クロック調整機能をイネーブルにしたり、調整デューティ・サイクルを選択できる。オンデマンド・クロック調整機能および TM1 が両方ともイネーブルで、プロセッサの温度ステータスがホット（IA32_THERM_STATUS MSR のビット 0 がセット）の場合は、オンデマンド・クロック調整デューティ・サイクルの設定とは無関係に、TM1 によって指定されたデューティ・サイクルでクロック調整が優先される。

ハイパー・スレッディング・テクノロジに対応したプロセッサでは、論理プロセッサごとに IA32_CLOCK_MODULATION レジスタが複製される。オンデマンド・クロック調整機能が正常に機能するためには、同じ物理プロセッサ内のすべての論理プロセッサ上で IA32_CLOCK_MODULATION レジスタが同じ設定になっていなければならない。

P6 ファミリ・プロセッサでは、オンデマンド・クロック調整機能はチップセットによって実装されており、クロックの調整はプロセッサの STPCLK# ピンを介して制御される。

13.16.4. 温度モニタ機能およびソフトウェア制御クロック調整機能の検出

CPUID 機能フラグの ACPI フラグ（ビット 22）は、IA32_THERM_STATUS、IA32_THERM_INTERRUPT、IA32_CLOCK_MODULATION_MSR、xAPIC 温度 LVT エントリの有無を示す。

CPUID 機能フラグの TM1 フラグ（ビット 29）は、クロック・デューティ・サイクルを調整する自動温度監視機能の有無を示す。

14

マシン・チェック・アーキテクチャ

第 14 章
マシン・チェック・アーキテクチャ

14

本章では、インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサのマシン・チェック・アーキテクチャとマシンチェック例外メカニズムについて説明する。マシンチェック例外の詳細については、第5章の「割り込み18 ーマシンチェック例外（#MC)」を参照。インテル® Pentium® プロセッサのマシンチェック機能についても簡単に説明する。

14.1. マシンチェック例外とマシン・チェック・アーキテクチャ

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサに用意されているマシン・チェック・アーキテクチャは、システム・バス・エラー、ECCエラー、パリティエラー、キャッシュ・エラー、TLBエラーなどのハードウェア（マシン）エラーを検出し、レポートするためのメカニズムを提供する。マシン・チェック・アーキテクチャは、マシンチェック機能をセットアップする際に使用するモデル固有レジスタ（MSR）のセットと、検出されたエラーをレポートするために使用する追加の数バンクのMSRで構成される。

プロセッサは、マシンチェック例外（#MC）を生成することで、マシン・チェック・エラーの検出を通知する。マシンチェック例外は、アボートクラスの例外になる。P6ファミリ・プロセッサのマシン・チェック・アーキテクチャでは通常、マシンチェック例外が生成された後でプロセッサを確実に再スタートはできない。ただし、マシンチェック例外ハンドラがマシン・チェック・エラーに関する情報をマシンチェックMSRからの収集は可能になる。

14.2. インテル® Pentium® プロセッサとの互換性

インテル® Pentium® 4プロセッサ、インテル® Xeon™プロセッサ、P6ファミリ・プロセッサでは、インテル® Pentium® プロセッサで導入されていたマシンチェック例外メカニズムがサポートされると共に、機能がさらに拡張されている。インテル Pentiumプロセッサがレポートするのは、次のマシン・チェック・エラーである。

- 読み取りサイクル実行時のデータ・パリティ・エラー
- 不完全に終了したバスサイクル

上記のエラーは P5_MC_TYPE と P5_MC_ADDR MSR を使用してレポートされる（インテル Pentium プロセッサに固有）。RDMSR 命令を使用して、これらの MSR を読み取る。アドレスについては、表 B-4. を参照。

インテル Pentium プロセッサで使用されているマシン・チェック・エラーのレポート・メカニズムは、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサで使用されているメカニズムと似ている。エラーが検出されると、P5_MC_TYPE と P5_MC_ADDR に記録され、プロセッサはマシンチェック例外（#MC）を生成する。

インテル Pentium プロセッサで実行するために記述されたマシン・チェック・コードと、P6 ファミリ・プロセッサで実行するために記述されたコードとの互換性については、14.3.3. 項「インテル® Pentium® プロセッサのマシン・チェック・エラーをマシン・チェック・アーキテクチャにマッピングする」と 14.7.3. 項「インテル® Pentium® プロセッサのマシンチェック例外の処理」を参照。

14.3. マシンチェック MSR

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのマシンチェック MSR は、グローバル制御レジスタならびにステータス・レジスタと、数バンクのエラー・レポート・レジスタで構成される（図 14-1. を参照）。それぞれのエラー・レポート・バンクは、プロセッサにある特定のハードウェア・ユニット（またはハードウェア・ユニットのグループ）に関連付けられる。RDMSR 命令と WRMSR 命令を使用して、これらのレジスタに対する読み取りと書き込みを行う。

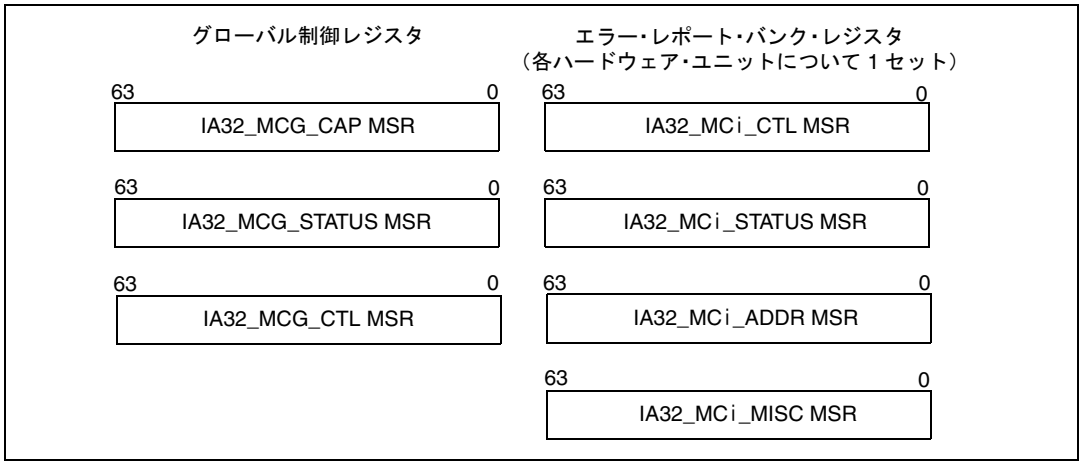


図 14-1. マシンチェック MSR

14.3.1. マシン・チェック・グローバル制御 MSR

マシン・チェック・グローバル制御 MSR には、IA32_MCG_CAP、IA32_MCG_STATUS、IA32_MCG_CTL が含まれる。これらのレジスタのアドレスについては、付録 B「モデル固有レジスタ（MSR）」を参照。

IA32_MCG_CAP の構造は、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサと P6 ファミリ・プロセッサとでは異なる方法で実装されている。また、P6 ファミリ・プロセッサに使用されるレジスタ名は、「IA-32」プリフィックスを持たない。

14.3.1.1. IA32_MCG_CAP MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

IA32_MCG_CAP MSR は、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサに用意されているマシン・チェック・アーキテクチャに関する情報を読み取るための専用レジスタである（図 14-2. を参照）。

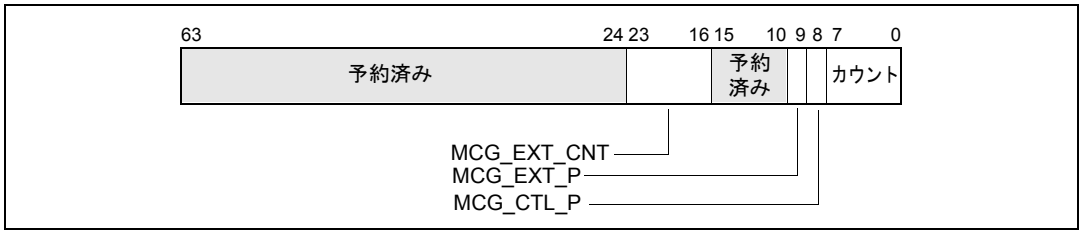


図 14-2. IA32_MCG_CAP レジスタ

ここで、

カウント・フィールド、ビット 0 ～ 7
個々のプロセッサで使用可能なハードウェア・ユニット・エラー・レポート・バンクの個数を示す。

MCG_CTL_P（制御 MSR 存在）フラグ、ビット 8
セットされている場合は、プロセッサが IA32_MCG_CTL MSR を実装していることを示す。クリアされている場合は、このレジスタが提供されていないことを示す。

MCG_EXT_P（拡張 MSR 存在）フラグ、ビット 9
セットされている場合は、MSR アドレス 180H で始まる拡張マシン・チェック・ステート・レジスタをプロセッサが実装していることを示す。クリアされている場合は、これらのレジスタが提供されていないことを示す。これは、インテル Pentium 4 プロセッサで導入された機能である。

MCG_EXT_CNT、ビット 16 ～ 23
: 存在する拡張マシン・チェック・ステート・レジスタの個数を示す。このフィールドは、MCG_EXT_P フラグがセットされているときだけ有効である。

ビット 10 ～ 15 およびビット 24 ～ 63 は予約されている。IA32_MCG_CAP レジスタに書き込んだ場合の影響は、定義されていない。

14.3.1.2. MCG_CAP MSR（P6 ファミリ・プロセッサ）

MCG_CAP MSR は、P6 ファミリ・プロセッサに用意されたマシン・チェック・アーキテクチャに関する情報を読み取る専用レジスタである（図 14-3. を参照）。

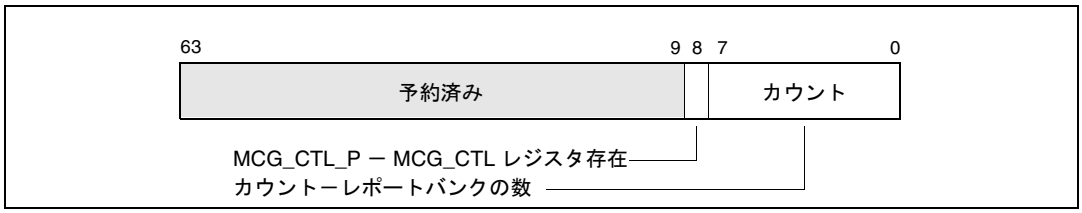


図 14-3. MCG_CAP レジスタ

ここで、

カウント・フィールド、ビット 0 ～ 7
: 個々のプロセッサで使用可能なハードウェア・ユニット・エラー・レポート・バンクの数を示す。

MCG_CTL_P（レジスタ存在）フラグ、ビット 8
: セットされた場合は、MCG_CTL レジスタが提供されることを示す。クリアされた場合は、提供されないことを示す。

ビット 9 ～ 63 は、予約されている。MCG_CAP レジスタに書き込んだ場合の効果は、定義されていない。

14.3.1.3. IA32_MCG_STATUS MSR

IA32_MCG_STATUS MSR（P6ファミリ・プロセッサではMCG_STATUS MSRという）は、マシンチェック例外が発生した後のプロセッサの現行の状態を記述する（図14-4.を参照）。

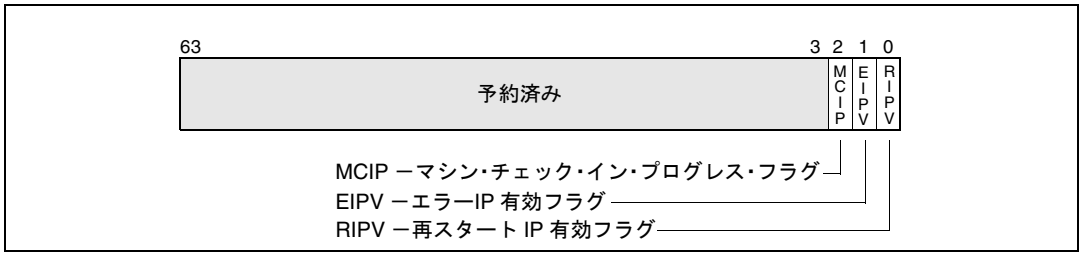


図 14-4. IA32_MCG_STATUS レジスタ

ここで、

RIPV（再スタート IP 有効）フラグ、ビット 0

セットされた場合は、マシンチェック例外が生成されたときに、スタックにプッシュされていた命令ポインタが指す命令からプログラムの実行を確実に再スタートできることを示す。クリアされた場合は、プッシュされていた命令ポインタでプログラムの実行を確実に再スタートすることは不可能になる。

EIPV（エラー IP 有効）フラグ、ビット 1

セットされた場合は、マシンチェック例外が生成されたときに、スタックにプッシュされていた命令ポインタが指す命令がエラーに直接関係していることを示す。クリアされた場合は、命令ポインタが指す命令は、エラーとは関係していない可能性がある。

MCIP（マシン・チェック・イン・プログレス）フラグ、ビット 2

セットされた場合は、マシンチェック例外が生成されたことを示す。このフラグは、ソフトウェアがセットするかクリアすることができる。MCIP がセットされている間にもう一度マシン・チェック・イベントが発生した場合は、プロセッサがシャットダウン状態に移行する可能性がある。シャットダウン状態のときのプロセッサの動作については、第 5 章「割り込みと例外の処理」の「割り込み 8 － ダブルフォルト例外（#DF）」を参照。

IA32_MCG_STATUS レジスタのビット 3 ～ 63 は、予約されている。

14.3.1.4. IA32_MCG_CTL MSR

IA32_MCG_CTL MSR（P6 ファミリ・プロセッサでは MCG_CTL MSR という）は、IA32_MCG_CAP MSR（または MCG_CAP MSR）のキャパビリティ・フラグ MCG_CTL_P がセットされている場合に使用できる。

IA32_MCG_CTL（または MCG_CTL）は、マシンチェック例外のレポート機能を制御する。このレジスタが使用可能な場合は、このレジスタにすべて 1 だけを書き込むと、すべてのマシンチェック機能がイネーブルになる。すべて 0 だけを書き込むと、すべてのマシンチェック機能がディスエーブルになる。これ以外の値は、定義されていないか、プロセッサ固有になる。

14.3.2. エラー・レポート・レジスタ・バンク

それぞれのエラー・レポート・レジスタ・バンクには、IA32_MCi_CTL、IA32_MCi_STATUS、IA32_MCi_ADDR、IA32_MCi_MISC の各 MSR（P6 ファミリ・プロセッサでは MCi_CTL、MCi_STATUS、MCi_ADDR、MCi_MISC という）を入れることができる。インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、エラー・レポート・レジスタのバンクが 4 つ用意されている。P6 ファミリ・プロセッサでは、エラー・レポート・レジスタのバンクが 5 つ用意されている。最初のエラー・レポート・レジスタ（IA32_MC0_CTL）は、常にアドレス 400H から開始される。

インテル Pentium 4 ファミリ・プロセッサおよびインテル Xeon プロセッサのエラー・レポート・レジスタのアドレスについては、表 B-1. を参照。P6 ファミリのエラー・レポート・レジスタのアドレスについては、表 B-3. を参照。

14.3.2.1. IA32_MCi_CTL MSR

IA32_MCi_CTL MSR（P6 ファミリ・プロセッサでは MCi_CTL という）は、特定のハードウェア・ユニット（または、ハードウェア・ユニットのグループ）が生成したエラーのエラーレポートを制御する。64 のフラグ（EEj）それぞれが、エラーの可能性を表す。EEj フラグをセットすると、関連するエラーのレポート機能がイネーブルになる。クリアすると、エラーのレポート機能がディスエーブルになる。MCi_CTL レジスタに 64 ビット値 FFFF FFFF FFFF FFFFH を書き込むと、すべてのエラーのログ機能がイネーブルになる。プロセッサは、定義されていないビットへの変更は書き込まない。図 14-5. に、IA32_MCi_CTL のビット・フィールドを示す。

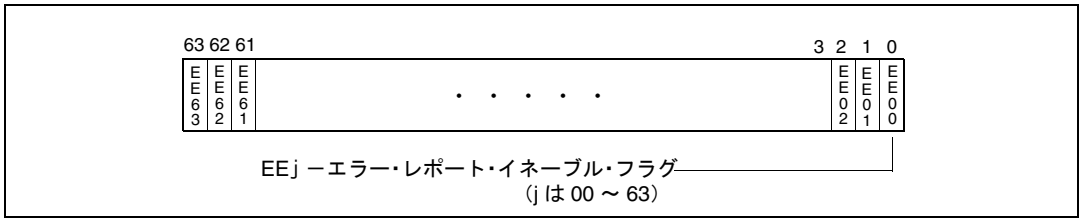


図 14-5. IA32_MCi_CTL レジスタ

注記

（P6ファミリ・プロセッサのみ）MC0_CTL MSRの内容は、オペレーティング・システムやエグゼクティブ・ソフトウェアで変更してはならない。このMSRは、内部でEBL_CR_POWERON MSRにエイリアスされ、プラットフォーム固有のエラー処理機能を制御する。これらの機能は、プラットフォーム固有になる。MC0_CTL MSRを適切に初期化するのは、システム固有のファームウェア（すなわちBIOS）の責任になる。現行のP6ファミリ・プロセッサでは、MC*i*_CTL MSRにはすべて1を書き込むかすべて0を書き込むことしか許可していない。

14.3.2.2. IA32_MCi_STATUS MSR

IA32_MCi_STATUS MSR（P6ファミリ・プロセッサではMCi_STATUSという）には、それ自体のVAL（有効）フラグがセットされている場合に、マシン・チェック・エラーに関連する情報が入れられる（図14-6.を参照）。IA32_MCi_STATUS MSRをクリアするには、ソフトウェアが責任を持ってそれらに0を明示的に書き込まなければならない。これらのMSRに1を書き込むと、一般保護例外が発生する。

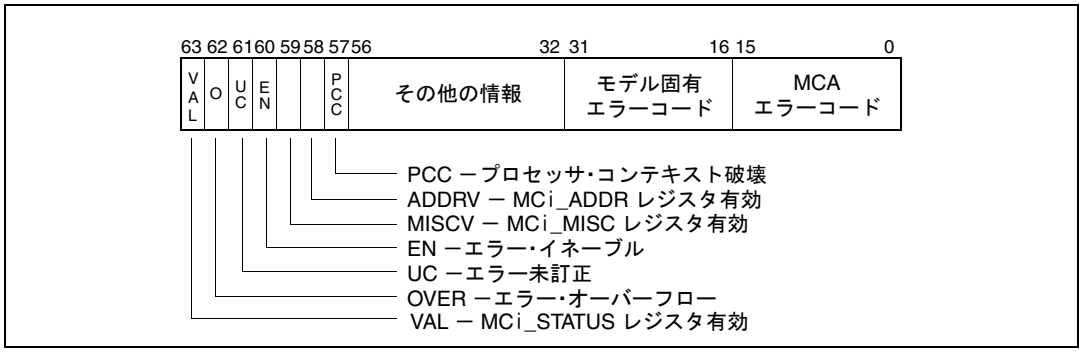


図 14-6. IA32_MCi_STATUS レジスタ

ここで、

MCA（マシン・チェック・アーキテクチャ）エラー・コード・フィールド、ビット 0 ～ 15
検出されたマシン・チェック・エラー条件に対して、マシン・チェック・アーキテクチャ定義のエラーコードを指定する。マシン・チェック・アーキテクチャ定義のエラーコードは、マシン・チェック・アーキテクチャが使用可能な IA-32 プロセッサであれば、すべて同じであることが保証されている。マシン・チェック・エラー・コードの詳細については、14.6. 節「MCA エラーコードの解釈」と付録 E「マシン・チェック・エラー・コードの解釈」を参照。

モデル固有エラー・コード・フィールド、ビット 16 ～ 31
検出されたマシン・チェック・エラー条件を独自に識別するモデル固有のエラーコードを指定する。モデル固有のエラーコードは、マシン・チェック・エラー条件が同じであっても、IA-32 プロセッサ間で異なることがある。モデル固有のエラーコードについては、付録 E「マシン・チェック・エラー・コードの解釈」を参照。

その他の情報フィールド、ビット 32 ～ 56
これらのビットの機能はプロセッサ固有であり、またマシン・チェック・アーキテクチャの一部ではない。IA-32 プロセッサ間での移植が意図されているソフトウェアは、これらの値に依存してはならない。

PCC（プロセッサ・コンテキスト破壊）フラグ、ビット 57
セットされた場合は、検出されたエラー条件によってプロセッサの状態が破壊されている可能性があり、プロセッサを確実に再スタートさせることが不可能であることを示す。クリアされた場合は、エラーがプロセッサの状態には影響を与えないことを示す。

ADDRV（IA32_MCi_ADDR レジスタ有効）フラグ、ビット 58
セットされた場合は、エラーが発生したアドレスが IA32_MCi_ADDR レジスタに入れられていることを示す（14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照）。クリアされた場合は、IA32_MCi_ADDR レジスタが実装されていないか、またはエラーが発生したアドレスが IA32_MCi_ADDR レジスタに入れられていないことを示す。このレジスタがプロセッサにない場合は、レジスタを読み取ってはならない。

MISCV（IA32_MCi_MISC レジスタ有効）フラグ、ビット 59
セットされた場合は、エラーに関する追加情報が IA32_MCi_MISC レジスタに入れられていることを示す。クリアされた場合は、IA32_MCi_MISC レジスタが実装されていないか、またはエラーに関する追加情報が IA32_MCi_MISC レジスタに入れられていないことを示す。このレジスタがプロセッサにない場合は、レジスタを読み取ってはならない。

EN（エラー・イネーブル）フラグ、ビット 60
セットされた場合は、IA32_MCi_CTL レジスタの関連する EEj ビットによってエラーがイネーブルになったことを示す。

UC（エラー未訂正）フラグ、ビット 61

セットされた場合は、プロセッサがエラー条件を訂正しなかったか訂正できなかったことを示す。クリアされた場合は、プロセッサがエラー条件を訂正できたことを示す。

OVER（マシン・チェック・オーバーフロー）フラグ、ビット 62

セットされた場合は、前のエラーによる結果がエラー・レポート・レジスタ・バンクにまだ存在する（すなわち、VAL ビットが IA32_MCi_STATUS レジスタ内ですでにセットされている）間に、マシン・チェック・エラーが発生したことを示す。この場合、プロセッサが OVER フラグをセットする。このフラグは、ソフトウェアの責任でクリアしなければならない。イネーブルになったエラーはディスエーブルになったエラーに上書きされ、また未訂正エラーは訂正済みエラーに上書きされる。未訂正エラーは、前の有効な未訂正エラーには上書きされない。

VAL（IA32_MCi_STATUS レジスタ有効）フラグ、ビット 63

セットされた場合は、IA32_MCi_STATUS レジスタ内の情報が有効であることを示す。このフラグがセットされると、前の有効なエントリを上書きする際に、プロセッサは IA32_MCi_STATUS レジスタの OVER フラグに対して与えられた規則に従う。VAL フラグは、プロセッサがセットし、またソフトウェアが責任を持ってクリアしなければならない。

14.3.2.3. IA32_MCi_ADDR MSR

IA32_MCi_ADDR MSR（P6 ファミリ・プロセッサでは MCi_ADDR という）には、IA32_MCi_STATUS レジスタの ADDRV フラグ（14.7. 節「マシン・チェック・ソフトウェアを記述する際の ガイドライン」を参照）がセットされている場合に、マシン・チェック・エラーが生成されたコードかデータのメモリ位置のアドレスが入れられる。IA32_MCi_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MCi_ADDR レジスタはサポートされていないか、またはアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。

返されるアドレスは、検出されたエラーのタイプによって、32 ビット仮想アドレス、32 ビット・リニア・アドレス、36 ビット物理アドレスのいずれかになる。

このレジスタのビット 36 ～ 63 は、将来のアドレス拡張に備えて予約されており、常に 0 として読み取られる。これらのレジスタは、明示的にすべて 0 を書き込むことによってクリアできる。1 を書き込むと、一般保護例外が生成される（図 14-7. を参照）。

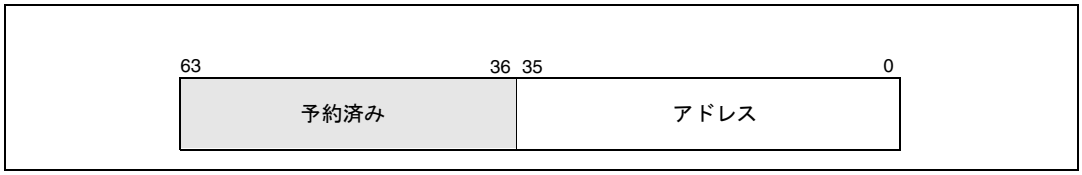


図 14-7. IA32_MCi_ADDR MSR

14.3.2.4. IA32_MCi_MISC MSR

IA32_MCi_MISC MSR（P6 ファミリ・プロセッサでは MCi_MISC MSR という）には、IA32_MCi_STATUS レジスタの MISCV フラグがセットされている場合に、マシン・チェック・エラーを記述する情報が入れられる。IA32_MCi_STATUS レジスタの MISCV フラグがクリアされている場合は、IA32_MCi_MISC_MASR レジスタはサポートされていないか、または追加情報を格納していない。

この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。プロセッサに実装されている場合、これらのレジスタは、明示的にすべて 0 を書き込むことによってクリアできる。1 を書き込むと、一般保護例外が生成される。このレジスタは、P6 ファミリ・プロセッサのいずれのエラー・レポート・レジスタ・バンクにも定義されていない。

14.3.2.5. IA32_MCG 拡張マシン・チェック・ステート MSR

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサは、可変数の拡張マシン・チェック・ステート MSR を実装している（表 14-1. を参照）。IA32_MCG_CAP MSR の MCG_EXT_P フラグはこれらのレジスタの有無を示し、MCG_EXT_CNT フィールドはこれらのレジスタの実際に実装されている個数を示す（14.3.1.1. 項「IA32_MCG_CAP MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照）。

IA32_MCG_MISC レジスタ以外にも、利用可能なレジスタが存在する場合がある。これらのレジスタは、実際の数に基づいて、IA32_MCG_RESERVED1 ～ IA32_MCG_RESERVEDn と呼ばれる。

表 14-1. 拡張マシン・チェック・ステート MSR

MSR	アドレス	説明
IA32_MCG_EAX	180H	マシン・チェック・エラーが発生したときのEAX レジスタのステート。
IA32_MCG_EBX	181H	マシン・チェック・エラーが発生したときのEBX レジスタのステート。
IA32_MCG_ECX	182H	マシン・チェック・エラーが発生したときのECX レジスタのステート。
IA32_MCG_EDX	183H	マシン・チェック・エラーが発生したときのEDX レジスタのステート。
IA32_MCG_ESI	184H	マシン・チェック・エラーが発生したときの ESI レジスタのステート。
IA32_MCG_EDI	185H	マシン・チェック・エラーが発生したときの EDI レジスタのステート。
IA32_MCG_EBP	186H	マシン・チェック・エラーが発生したときの EBP レジスタのステート。
IA32_MCG_ESP	187H	マシン・チェック・エラーが発生したときの ESP レジスタのステート。
IA32_MCG_EFLAGS	188H	マシン・チェック・エラーが発生したときの EFLAGS レジスタのステート。
IA32_MCG_EIP	189H	マシン・チェック・エラーが発生したときの EIP レジスタのステート。
IA32_MCG_MISC	18AH	セットされている場合､DS通常操作の間にページアシストまたはページフォルトが発生したことを示す。

インテル Pentium 4 プロセッサまたはインテル Xeon プロセッサでマシン・チェック・エラーが検出されると、プロセッサは、これらの拡張マシン・チェック・ステート MSR 内の EIP、EFLAGS レジスタ、および汎用レジスタのステートを保存する。この情報を利用して、デバッガでエラーを分析できる。

これらのレジスタは、読み出し / ゼロ書き込みレジスタである。つまり、ソフトウェアはこれらのレジスタを読み出せるが、書き込む場合は、すべて 0 の書き込みだけが許される。ソフトウェアがこれらのレジスタに 0 でない値を書き込もうとすると、一般保護（#GP）例外が発生する。ハードウェア・リセット（電源投入または RESET）を実行すると、これらのレジスタはクリアされる。ソフトリセット（INIT リセット）を実行すると、レジスタの内容は保持される。

14.3.3. インテル® Pentium® プロセッサのマシン・チェック・エラーをマシン・チェック・アーキテクチャにマッピングする

インテル® Pentium® プロセッサは、P5_MC_TYPE と P5_MC_ADDR の 2 つのレジスタを使用してマシン・チェック・エラーをレポートする。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサは、これらのレジスタを、エラー・レポート・レジスタ・バンクの IA32_MCi_STATUS と IA32_MCi_ADDR にマッピングする。P5_MC_TYPE と P5_MC_ADDR 内でレポートされた外部バスエラーと同じタイプをレポートする。

これらのレジスタ内に入れられた情報には、2 つの方法でアクセスできる。

- インテル Pentium 4 プロセッサおよび P6 ファミリ・プロセッサ用に記述された汎用マシンチェック例外ハンドラの一部として、IA32_MC*i*_STATUS レジスタと MC*i*_ADDR レジスタを読み取ることでアクセスする。
- RDMSR 命令を使用して P5_MC_TYPE レジスタと P5_MC_ADDR レジスタを読み取ることでアクセスする。

2 番目の方法を使用する場合は、インテル Pentium プロセッサ用に記述されたマシンチェック例外ハンドラを、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサ上で実行することが可能になる。ただし、制限によって、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサが返す情報は、インテル Pentium プロセッサが返す情報とは異なってエンコードされる。インテル Pentium プロセッサのマシンチェック例外ハンドラをインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサ上で実行するためには、P5_MC_TYPE レジスタのエンコーディングを正しく解釈できるようにハンドラを記述し直す必要がある。

14.4. マシンチェックの使用可能性

マシン・チェック・アーキテクチャとマシンチェック例外（#MC）は、モデル固有の機能である。ソフトウェアは、CPUID 命令を実行すると、これらの機能がプロセッサに用意されているかどうかを判断できる。CPUID 命令を実行した後は、EDX の MCA フラグ（ビット 14）と MCE フラグ（ビット 7）の設定に、プロセッサにマシン・チェック・アーキテクチャとマシンチェック例外が用意されているかどうかが示される。

14.5. マシンチェックの初期化

プロセッサのマシン・チェック・アーキテクチャを使用するためには、ソフトウェアがプロセッサを初期化し、マシンチェック例外メカニズムとエラー・レポート・メカニズムを起動しなければならない。

例 14-1. に、この初期化を実行するための疑似コードを示す。この疑似コードは、マシン・チェック・アーキテクチャとマシンチェック例外が存在するかどうかをまずチェックし、その後でマシンチェック例外とエラー・レポート・レジスタ・バンクをイネーブルにする。示された疑似コードは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ間で互換性が維持される。

電源投入後または電源の再投入後、例 14-1. の初期化用疑似コードに示すように、ソフトウェアが最初に IA32_MCi_STATUS レジスタを 0 にクリアするまでは、IA32_MCi_STATUS レジスタが有効なデータを保持することは保証されない。さらに、

P6ファミリ・プロセッサを使用しているとき、ソフトウェアはソフトリセットを行う際にMCi_STATUSレジスタを0にしなければならない。

例 14-1. マシンチェック初期化用の疑似コード

```
Check CPUID Feature Flags for MCE and MCA support
IF CPU supports MCE
THEN
    IF CPU supports MCA
    THEN
        IF (IA32_MCG_CAP.MCG_CTL_P = 1)
        (* IA32_MCG_CTL register is present *)
        THEN
            IA32_MCG_CTL ← FFFFFFFFFFFFFFFFH;
            (* enables all MCA features *)
        FI

        (* Determine number of error-reporting banks supported *)
        COUNT¨ IA32_MCG_CAP.Count;
        MAX_BANK_NUMBER ← COUNT - 1;

        IF (Processor Family is 6H)
        THEN
            (* Enable logging of all errors except for MC0_CTL register *)
            FOR error-reporting banks (1 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFFH;
            OD

            (* Clear all errors *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_STATUS ← 0;
            OD

        ELSE IF (Processor Family is 0FH) (*any Processor Extended Family *)
        THEN
            (* Enable logging of all errors including MC0_CTL register *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFFH;
            OD

            (* BIOS clears all errors only on power-on reset *)
            IF (BIOS detects Power-on reset)
            THEN
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    IA32_MCi_STATUS ← 0;
                OD
            ELSE
```

```
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    (Optional for BIOS and OS) Log valid errors
                    (OS only) IA32_MCi_STATUS ← 0;
                OD

            FI
        FI
    FI

    Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

    Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions
FI
```

14.6. MCA エラーコードの解釈

マシン・チェック・エラー条件を検出すると、プロセッサはいずれかの IA32_MCi_STATUS レジスタの MCA エラー・コード・フィールドに 16 ビットのエラーコードを書き込み、そのレジスタの VAL（有効）フラグをセットする。プロセッサはまた、マシン・チェック・アーキテクチャに応じて、16 ビットのモデル固有エラーコードを IA32_MCi_STATUS レジスタにも書き込む。

MCA エラーコードは、IA-32 プロセッサに対してアーキテクチャ的に定義される。ただし、コードが書き込まれる個々の IA32_MCi_STATUS レジスタはモデル固有になる。マシンチェック例外の原因を判断するとき、マシンチェック例外ハンドラはまず、それぞれの IA32_MCi_STATUS レジスタに対する VAL フラグを読み取らなければならない。フラグがセットされていれば、マシンチェック例外ハンドラはレジスタの MCA エラー・コード・フィールドを読み取らなければならない。レポートされているエラーのタイプを決定するのは、MCA エラー・コード・フィールド [15:0] 値のエンコーディングであり、エラーをレポートしているレジスタバンクではない。

MCA エラーコードには、単純エラーコードと複合エラーコードの 2 種類が存在する。

14.6.1. 単純エラーコード

表 14-2. に、単純エラーコードを示す。これら独自のコードは、グローバル・エラー情報を表す。

表 14-2. IA32_MCi_STATUS [15:0] 単純エラーコードのエンコーディング

エラーコード	バイナリ・エンコーディング	意味
エラーなし	0000 0000 0000 0000	エラー・レポート・レジスタのこのバンクには、エラーはレポートされていない。
未分類	0000 0000 0000 0001	このエラーは、MCA エラークラスには分類されていない。
マイクロコード ROM パリティエラー	0000 0000 0000 0010	内部マイクロコード ROM 内のパリティエラー
外部エラー	0000 0000 0000 0011	他のプロセッサからの BINIT# により、このプロセッサがマシンチェックに移行。[1]
FRC エラー	0000 0000 0000 0100	FRC（ファンクション重複チェック）マスタ / スレーブエラー
内部、未分類	0000 01xx xxxx xxxx	内部の未分類のエラー[2]

注 :

1. 当該プロセッサ（または同じ外部バス上の任意のプロセッサ）の BINIT# 観察が電源投入時の環境設定で有効になっており（ハードウェア・ストラップ）、マシンチェック例外が有効になっている（CR4.MCE = 1 にセットされている）場合は、BINIT# がアサートされると、マシンチェック例外が発生する。
2. 内部の未分類のエラーは、マシン・チェック・レジスタに追加情報が含まれていないため、まだ分類されていないエラーである。

14.6.2. 複合エラーコード

複合エラーコードは、TLB、メモリ、キャッシュ、バス、相互接続論理および内部タイマなどに関連するエラーを記述する。すべての複合エラーでは、1 セットのサブ・フィールドが共通になる。これらのサブ・フィールドは、アクセスのタイプ、メモリ階層内でのレベル、要求のタイプを記述する。表 14-4. に、複合エラーコードの一般的な形式を示す。表の解釈の列は、複合エラーの名前を示す。この名前は、表 14-4. ～ 14-7. に示されたニーモニックを、中括弧内に示されたサブ・フィールド名と置き換えれば作成できる。

例えば、エラーコード ICACHEL1_RD_ERR は、次の形式から作成される。

{TT}CACHE{LL}_{RRRR}_ERR
{TT} は I で、{LL} は L1 で、{RRRR} は RD でそれぞれ置き換えられる。

表 14-3. IA32_MCi_STATUS [15:0] の複合エラーコードのエンコーディング

タイプ	形式	解釈
TLB エラー	0000 0000 0001 TTLL	{TT}TLB{LL}_ERR
メモリ階層エラー	0000 0001 RRRR TTLL	{TT}CACHE{LL}_{RRRR}_ERR
バスエラーと相互接続エラー	0000 1PPT RRRR IILL	BUS{LL}_{PP}_{RRRR}_{II}_{T}_ERR
内部タイマ	0000 0100 0000 0000	

2 ビットの TT サブ・フィールド（表 14-4. を参照）は、トランザクションのタイプ（データ、命令、または一般）を示す。TT サブ・フィールドは、TLB、キャッシュ、相互接続エラー条件に適用される。相互接続エラー条件は、主に P6 ファミリ・プロセッサとインテル® Pentium® プロセッサに関連していて、システムバスから分離した外部 APIC バスを利用している。プロセッサがトランザクションのタイプを決定できない場合は、一般タイプがレポートされる。

表 14-4. TT（トランザクション・タイプ）サブ・フィールドに対するエンコーディング

トランザクション・タイプ	ニーモニック	バイナリ・エンコーディング
命令	I	00
データ	D	01
一般	G	10

2 ビットの LL サブ・フィールド（表 14-5. を参照）は、エラーが発生したメモリ階層内でのレベル（レベル 0、レベル 1、レベル 2、または一般）を示す。LL サブ・フィールドはまた、TLB、キャッシュ、および相互接続エラー条件に対しても適用される。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサおよび P6 ファミリ・プロセッサは、キャッシュ階層内で 2 つのレベル、TLB 内で 1 つのレベルをサポートする。ここでも、プロセッサが階層レベルを決定できない場合は一般タイプがレポートされる。

表 14-5. LL（メモリ階層レベル）サブ・フィールドに対するレベル・エンコーディング

階層レベル	ニーモニック	バイナリ・エンコーディング
レベル 0	L0	00
レベル 1	L1	01
レベル 2	L2	10
一般	LG	11

4 ビットの RRRR サブ・フィールド（表 14-6. を参照）は、エラーに関連付けられた処置のタイプを示す。処置には、読み取り操作と書き込み操作、プリフェッチ、キャッシュの立ち退き、スヌープが含まれる。エラーのタイプを決定できない場合は、一般エラーが返される。また、エラーの原因となった命令かデータ要求のタイプをプロセッサが決定できない場合は、一般読み取りと一般書き込みが返される。立ち退き要求とスヌープ要求は、キャッシュに対してのみ適用される。これ以外のすべての要求は、TLB、キャッシュ、相互接続に対して適用される。

表 14-6. 要求（RRRR）サブ・フィールドのエンコーディング

要求タイプ	ニーモニック	バイナリ・エンコーディング
一般エラー	ERR	0000
一般読み取り	RD	0001
一般書き込み	WR	0010
データ読み取り	DRD	0011
データ書き込み	DWR	0100
命令フェッチ	IRD	0101
プリフェッチ	PREFETCH	0110
立ち退き	EVICT	0111
スヌープ	SNOOP	1000

バスエラーと相互接続エラーは、LL サブ・フィールドと RRRR サブ・フィールドに加え、2 ビットの PP（パーティシペーション）、1 ビットの T（タイムアウト）、2 ビットの II（メモリか I/O）の各サブ・フィールドで定義される（表 14-7. を参照）。バスエラー条件は、プロセッサ固有のバスタイプに関連する。同様に、相互接続エラー条件は、記憶領域階層の異なるレベル間の接続を記述する相互接続モデル（相互接続モデルは、個々のプロセッサに従属する）を基に判断される。バスのタイプは、プロセッサに依存するため、本書では指定しない。バス・トランザクションか相互接続トランザクションは、アドレスと応答を含む要求で構成される。

表 14-7. PP、T、および II の各サブ・フィールドのエンコーディング

サブ・フィールド	トランザクション	ニーモニック	バイナリ・エンコーディング
PP（パーティシペーション）	ローカル・プロセッサ 1 が要求を生成した	SRC	00
	ローカル・プロセッサ 1 が要求に応答した	RES	01
	ローカル・プロセッサ 1 が、サードパーティとしてエラーを観察した	OBS	10
	一般		11
T（タイムアウト）	要求がタイムアウトした	TIMEOUT	1
	要求はタイムアウトしなかった	NOTIMEOUT	0
II（メモリか I/O）	メモリアクセス	M	00
	予約済み		01
	I/O	IO	10
	他のトランザクション		11

注 :

1. ローカル・プロセッサは、他のシステム・コンポーネント（APIC、他のプロセッサなど）からのエラーを報告しているプロセッサを区別する。

14.6.3. マシン・チェック・エラー・コードの解釈の例

付録 E「マシン・チェック・エラー・コードの解釈」では、MCA エラーコード、モデル固有のエラーコード、その他の情報エラーコードのフィールドの解釈について説明する。P6 ファミリ・プロセッサについては、外部バスエラーのデコードに関する情報が追加されている。インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサについては、外部バスエラー、内部タイマエラー、メモリ階層エラーに関する情報が記載されている。

14.7. マシン・チェック・ソフトウェアを記述する際のガイドライン

マシン・チェック・アーキテクチャとエラーログ機能は、次の 2 つの用途で使用することができる。

- マシンチェック例外（#MC）を使用し、通常の命令実行時のマシンエラーを検出するために使用する。
- マシンエラーを定期的にチェックし、ログするために使用する。

マシンチェック例外を使用するためには、オペレーティング・システムかエグゼクティブ・ソフトウェアがマシンチェック例外ハンドラを提供しなければならない。このハンドラは、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ、または P6 ファミリ・プロセッサ専用として設計できる。また、IA-32 プロセッサのいくつかの世代のマシン・チェック・エラーも扱えるような移植可能なハンドラとして設計できる。

マシンエラーをログするためには、特殊なプログラムかユーティリティが必要になる。

この後の各項では、マシンチェック例外ハンドラかマシン・エラー・ログ・ユーティリティを記述する際のガイドラインについて説明する。

14.7.1. マシンチェック例外ハンドラ

マシンチェック例外（#MC）は、ベクタ 18 に対応する。マシンチェック例外を処理するためには、トラップゲートを IDT に追加し、トラップゲート内のポインタがマシンチェック例外ハンドラを指さなければならない。例外ハンドラを設計する際は、2 つのアプローチがある。

1. ハンドラは、すべてのマシン・ステータスとエラー情報を単にログするだけで、その後はデバッガを呼び出すかシステムをシャットダウンする。
2. ハンドラは、レポートされたエラー情報を解析し、場合によってはエラーの訂正とプロセッサの再スタートを試みる。

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサでは、ほとんどすべてのマシンチェック条件を訂正できない（アボートタイプの例外が発生する）。したがって、ステータスとエラー情報のログ機能が基本的に必要な機能になる。エラーのログの詳細については、14.7. 節を参照。

マシン・チェック・エラーからの回復が可能になったら、マシンチェック例外ハンドラを記述する際は、次の点に配慮しなければならない。

- エラーの性質を判断するため、ハンドラはそれぞれのエラー・レポート・レジスタ・バンクを読み取る必要がある。レジスタバンクの数は、IA32_MCG_CAP レジスタのカウント・フィールドで与えられる。また、レジスタバンク 0 の最初のレジスタは、アドレス 400H に置かれる。
- レジスタ内のエラー情報が有効であるかどうかは、それぞれの IA32_MC*i*_STATUS レジスタの VAL（有効）フラグが示す。このフラグがクリアされている場合は、バンク内のレジスタには有効なエラー情報は入れられず、したがってチェックする必要はない。

- 移植可能な例外ハンドラを記述するためには、IA32_MC*i*_STATUS レジスタの MCA エラー・コード・フィールドだけをチェックするようにしなければならない。このフィールドを解釈するためのアルゴリズムを記述する際に利用できる情報については、14.6. 節を参照。
- IA32_MC*i*_STATUS レジスタの RIPV、PCC、OVER の各フラグは、エラーからの回復が可能であるかどうかを示す。これらのフィールドの 1 つがセットされている場合は、回復は不可能になる。OVER フィールドは、複数のマシン・チェック・エラーが発生したことを示す。回復が不可能である場合、ハンドラは通常、エラー情報を記録し、オペレーティング・システムにアボートを通知する。
- 訂正されたエラーは、プロセッサによって自動的に訂正される。エラーがプロセッサによって自動的に訂正されたものであるかどうかは、それぞれの IA32_MC*i*_STATUS レジスタの UC フラグが示す。
- 命令ポインタ（例外が発生したとき、スタックにプッシュされた命令のアドレス）が指す命令からプログラムを再スタートできるかどうかは、IA32_MCG_STATUS レジスタの RIPV フラグが示す。このフラグがクリアされている場合でも、（デバッグを実行するために）プロセッサを再スタートすることは可能になるが、プログラムの継続性は失われる。
- 回復不能エラーに対しては、例外が発生したとき、スタックにプッシュされた命令ポインタが指す命令がエラーに関係しているかどうかを IA32_MCG_STATUS レジスタの EIPV フラグが示す。このフラグがクリアされている場合は、プッシュされた命令はエラーに関係していないと考えられる。
- マシンチェック例外が発生したかどうかは、IA32_MCG_STATUS レジスタの MCIP フラグが示す。マシンチェック例外ハンドラから戻る前に、ソフトウェアはこのフラグをクリアし、エラー・ログ・ユーティリティがこのフラグを確実に使用できるようにしておかなければならない。MCIP フラグは再帰も検出できるが、マシン・チェック・アーキテクチャは再帰をサポートしていない。プロセッサがマシンチェック再帰を検出した場合、プロセッサはシャットダウン状態に移行する。

14.7.2. BINIT# ドライブおよび BINIT# 観察の有効化

プロセッサのマシンチェック機能が正常に動作するためには、システム BIOS が BINIT# ドライブおよび BINIT# 観察を有効にする必要がある。これにより、プロセッサは BINIT# を使用して、内部や外部のブロック状態をクリアできる。また、プロセッサは広範囲にわたるマシンチェック例外を正しく報告できる。

例えば、次のような状況のインテル® Pentium® III プロセッサを考える。

- ロックされた CMPXCHG8B 命令を実行している。
- 最初のデータ読み取りでマシンチェック例外が報告される。

- 比較操作が失敗する。

このプロセッサは、ロックされたシーケンスの完了後に BINIT# 信号をアサートしてバスをアンロックする。BINIT# ドライブが有効になっていない場合（UP 環境）またはBINIT# ドライブと BINIT# 観察が有効になっていない場合は（MP 環境）、マシン・チェック・エラーは記録されるが、マシンチェック例外は発生しない（MCE が有効になっている場合）。

例 14-2. に、マシンチェック例外ハンドラが実行する典型的なステップを示す。

例 14-2. マシンチェック例外ハンドラの疑似コード

```
IF CPU supports MCE
    THEN
        IF CPU supports MCA
            THEN
                call errorlogging routine; (* returns restartability *)
        FI;
    ELSE (* Pentium(R) processor compatible *)
        READ P5_MC_ADDR
        READ P5_MC_TYPE;
        report RESTARTABILITY to console;
FI;
IF error is not restartable
    THEN
        report RESTARTABILITY to console;
        abort system;
FI;
CLEAR MCIP flag in IA32_MCG_STATUS;
```

14.7.3. インテル® Pentium® プロセッサのマシンチェック例外の処理

マシンチェック例外ハンドラをインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサに移植するため、CPUID 命令を使用してプロセッサ・タイプを判別できる。この後は、このプロセッサ・タイプを基に、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、またはインテル Pentium プロセッサ専用にマシンチェック例外の処理が可能になる。

インテルPentiumプロセッサに対してマシンチェック例外をイネーブルにした場合（制御レジスタ CR0 の MCE フラグをセット）、マシンチェック例外ハンドラは RDMSR 命令を使用して P5_MC_TYPE レジスタからエラータイプを、P5_MC_ADDR レジスタからマシン・チェック・アドレスをそれぞれ読み取る。この後、ハンドラは実行をアボートする前に、これらのレジスタ値をシステム・コンソールにレポートする（例 14-2. を参照）。

14.7.4. 訂正可能マシン・チェック・エラーのログ

マシン・チェック・エラーが訂正可能なものである場合は、そのエラーに対してプロセッサはマシンチェック例外を生成しない。訂正可能マシン・チェック・エラーを検出するためには、ユーティリティ・プログラムを記述し、それぞれのマシン・チェック・エラー・レポート・レジスタ・バンクを読み取ると共に、アカウント・ファイルかデータ構造内に結果をログしなければならない。このユーティリティは、次のいずれかの方法で実現できる。

- 1 時間か 1 日に 1 回程度の間隔でレジスタバンクをポーリングするシステムデーモンを使用する。
- レジスタバンクをポーリングし、例外を記録するユーザ起動のアプリケーションを使用する。この場合、実際のポーリング・サービスは、オペレーティング・システムのドライバが提供するか、システム・コール・インターフェイスを通じて提供される。

例 14-3. に、エラー・ログ・ユーティリティ用の疑似コードを示す。

例 14-3. マシン・チェック・エラー・ログ用の疑似コード

```
Assume that execution is restartable;
IF the processor supports MCA
    THEN
    FOR each bank of machine-check registers
        DO
            READ IA32_MCi_STATUS;
            IF VAL flag in IA32_MCi_STATUS = 1
                THEN
                    IF ADDRV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_ADDR;
                    FI;
                    IF MISCV flag in IA32_MCi_STATUS = 1
                        THEN READ IA32_MCi_MISC;
                    FI;
                    IF MCIP flag in IA32_MCG_STATUS = 1
                        (* Machine-check exception is in progress *)
                        AND PCC flag in IA32_MCi_STATUS = 1
                        AND RIPV flag in IA32_MCG_STATUS = 0
                        (* execution is not restartable *)
                            THEN
                                RESTARTABILITY = FALSE;
                                return RESTARTABILITY to calling procedure;
                    FI;
                    Save time-stamp counter and processor ID;
                    Set IA32_MCi_STATUS to all 0s;
                    Execute serializing instruction (i.e., CPUID);
            FI;
        OD;
FI;
```

プロセッサがマシン・チェック・アーキテクチャをサポートしている場合は、このユーティリティはエラー・レポート・レジスタのバンクを読み取って有効なレジスタエントリを探す。次に有効なバンクそれぞれに対するIA32_MCi_STATUS、IA32_MCi_ADDR、IA32_MCi_MISC、IA32_MCG_STATUSの各レジスタの値をセーブする。このルーチンは、生データをシステムデータ構造かファイルに記録し、ポーリングに関与するオーバーヘッドを低減することで、処理時間を最小限に抑える。収集したデータは、ユーザ・ユーティリティを使用してオフライン環境で解析する。

IA32_MCG_STATUS レジスタ内でMCIPフラグがセットされている場合は、マシンチェック例外が進行中であり、またマシンチェック例外ハンドラによって例外ログルーチンがすでに呼び出されていることを表している。

ログプロセスが完了したら、例外処理ルーチンは、実行が再スタートできるかどうかを判断しなければならない。損傷が発生しておらず（IA32_MCi_STATUS レジスタのPCC フラグがクリアされている）、しかもプロセッサが実行を再スタート可能であると保証できる（IA32_MCG_STATUS レジスタのRIPV フラグがセットされる）場合は、通常は実行を再スタートできる。実行を再スタートできない場合は、システムは回復不能になり、また例外処理ルーチンは、この後に続くシャットダウン用にエラー・ステータスをオペレーティング・システムのカーネルに返す前に、コンソールに適切な通知を送らなければならない。

マシン・チェック・アーキテクチャでは、特定のエラー・レポート・バンクからの例外をバッファすることが可能になる。ただし、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでは、この機能はサポートされていない。エラー・ログ・ルーチンは、それぞれのハードウェア・エラー・レポート・バンクのIA32_MCi_STATUS レジスタを読み取り、その後でこのレジスタのOVER フラグとVAL フラグに0を書き込んでクリアすることで、将来のプロセッサとの互換性を維持しなければならない。エラー・ログ・ユーティリティは、バンクに対するIA32_MCi_STATUS レジスタを再度読み取り、有効ビットがクリアされているのを確認しなければならない。プロセッサは、次に発生したエラーをこのレジスタバンクに書き込み、VAL フラグをセットする。

例外ログルーチンがストアしなければならない追加情報には、プロセッサのタイムスタンプ・カウンタ値が含まれる。この値は、例外の頻度を示すためのメカニズムを提供する。マルチプロセッサを使用するオペレーティング・システムは、例外が発生したプロセッサ・ノードの識別を、独自の識別子（プロセッサのAPIC ID など）を使用してストアする（8.8. 節「割り込みの処理」を参照）。

例 14-3. に示した基本アルゴリズムを変更して、より強固な回復手法を提供することができる。例えば、ソフトウェアは、ハードウェアには利用できない情報を使用して回復を試みる柔軟性を持っている。具体的には、マシンチェック例外ハンドラは、実行の再スタートが不可能なエラーがエラー・ログ・ルーチンによってレポートされた

とき、ログを行った後でエラーをレポートしたレジスタを注意深く解析することができる。これらの回復手法は、エラーレポートで提供される外部バス関連のモデル固有情報を使用すれば、システム内のエラー発生箇所を特定し、適当な回復方針を決定できる。

15

デバッグと性能モニタリング

第 15 章
デバッグと性能モニタリング 15

IA-32 アーキテクチャは、コードのデバッグや、コードの実行およびプロセッサ性能のモニタリングなどで使用する充実したデバッグ機能を備えている。これらの機能は、アプリケーション・ソフトウェア、システム・ソフトウェア、マルチタスキング・オペレーティング・システムなどのデバッグに非常に価値のあるものである。

デバッグのサポートには、デバッグレジスタ（DB0 ～ DB7）と 2 つのモデル固有レジスタ（MSR）を通じてアクセスできる。IA-32 プロセッサのデバッグレジスタは、ブレークポイントと呼ばれるメモリ位置と I/O ロケーションのアドレスを保持する。ブレークポイントとは、プログラム、データストア領域、または特定の I/O ポートのユーザが選択したロケーションであり、これらのブレークポイントで、プログラマやシステム設計者はプログラムの実行を停止させ、デバッガ・ソフトウェアを呼び出してプロセッサのステートを調べることができる。これらのブレークポイントアドレスのいずれかにメモリアクセスまたは I/O アクセスが行われると、デバッグ例外（#DB）が発生する。ブレークポイントは、メモリ読み取り / 書き込み操作、または I/O 読み取り / 書き込み操作などの特定形式のメモリアクセスまたは I/O アクセスに対して指定する。デバッグレジスタは命令およびデータ両ブレークポイントをサポートする。MSR（P6 ファミリ・プロセッサで IA-32 アーキテクチャに採用）は、分岐、割り込み、例外をモニタし、最後に行われた分岐、割り込み、または例外処理のアドレス、および割り込みまたは例外の前の最後に行われた分岐処理を記録する。

15.1. デバッグサポート機能の概要

デバッグと性能モニタリングは以下のプロセッサ機能によってサポートされている。

- **デバッグ例外**（#DB）－デバッグイベントが発生したとき、プログラムの制御をデバッガ・プロシージャまたはタスクに転送する。
- **ブレークポイント例外**（#BP）－ INT 3 命令が実行されたとき、プログラムの制御をデバッガ・プロシージャまたはタスクに転送する。
- **ブレークポイント・アドレス・レジスタ**（DB0 ～ DB3）－最高 4 つのブレークポイントのアドレスを指定する。
- **デバッグ・ステータス・レジスタ**（DB6）－デバッグ例外またはブレークポイント例外が発生したときに有効であった条件を報告する。
- **デバッグ制御レジスタ**（DB7）－ブレークポイントを発生させるメモリアクセスまたは I/O アクセスの形式を指定する。

- **T（trap）フラグ、TSS** －その TSS に T（トラップ）フラグがセットされているタスクに切り替えようとしたとき、デバッグ例外（#DB）を発生する。
- **RF（resume）フラグ、EFLAGS レジスタ** －同一命令に対する複数の例外の発生を抑止する。
- **TF（trap）フラグ、EFLAGS レジスタ** －特定の命令が実行されるたびにデバッグ例外（#DB）を発生する。
- **ブレークポイント命令（INT 3）** －ブレークポイント例外（#BP）を発生する。その結果、プログラムの制御がデバッガ・プロシージャまたはタスクに転送される。この命令は、デバッグレジスタを使用したコード・ブレークポイントの代替設定手段である。ブレークポイントを 5 つ以上使用したいとき、またはブレークポイントをソースコード内に配置するときに特に効果的である。
- **最新分岐記録機能** － 15.5. 節「最新の分岐、割り込み、例外の記録（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」および 15.6. 節「最新の分岐、割り込み、例外の記録（P6 ファミリ・プロセッサ）」を参照のこと。

これらの機能により、デバッガを個別のタスクとしても、または現行のプログラムまたはタスクのコンテキスト内のプロシージャとしても呼び出すことができる。デバッガは以下の条件を使用して呼び出せる。

- 特定タスクへのタスクスイッチ
- ブレークポイント命令の実行。
- 任意の命令の実行。
- 指定アドレスでの命令の実行。
- 指定メモリアドレスのバイト、ワード、またはダブルワードの読み取り / 書き込み。
- 指定メモリアドレスのバイト、ワード、またはダブルワードの書き込み。
- 指定 I/O アドレスのバイト、ワード、またはダブルワードの入力。
- 指定 I/O アドレスのバイト、ワード、またはダブルワードの出力。
- デバッグレジスタの内容の変更が試みられるとき。

15.2. デバッグレジスタ

プロセッサのデバッグ操作は、8つのデバッグレジスタ（図15-1.を参照）によって制御される。これらのレジスタには、MOV命令のデバッグレジスタへの移動またはデバッグレジスタからの移動形式を使用して読み書きできる。デバッグレジスタは、これらの命令のソース・オペランドにもデスティネーション・オペランドにもなり得る。ただし、デバッグレジスタは特権的リソースであるため、これらのレジスタにアクセスするMOV命令は、実アドレスモード、SMM、またはCPL = 0の保護モードでしか実行できない。これら以外の特権レベルからデバッグレジスタに読み書きしようとすると、一般保護例外（#GP）が発生する。

デバッグレジスタの第一の機能は、1～4のブレークポイント（0～3の番号が割り当てられる）をセットアップし、モニタすることである。各ブレークポイントに対して、デバッグレジスタで以下の情報を指定し、検出できる。

- ブレークポイントを発生させるリニアアドレス。
- ブレークポイント・ロケーションの長さ（1、2、または4バイト）。
- デバッグ例外を発生させるために、該当アドレスで実行されなければならない操作。
- ブレークポイントをイネーブルにするかどうかの区別。
- デバッグ例外が発生したときのブレークポイント条件の有無。

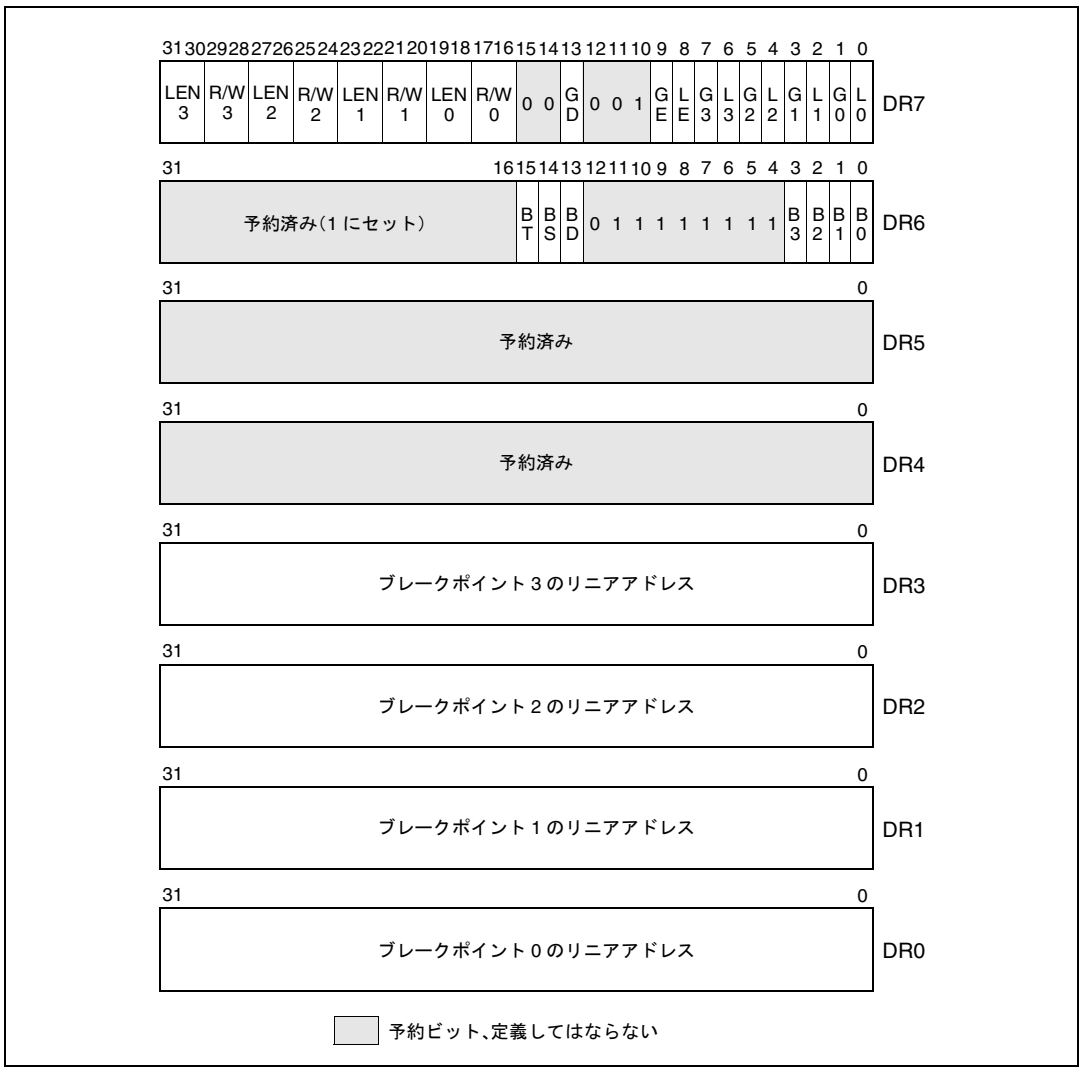


図 15-1. デバッグレジスタ

以降の各項で、デバッグレジスタのフラグとフィールドの機能について説明する。

15.2.1. デバッグ・アドレス・レジスタ（DR0 ～ DR3）

デバッグ・アドレス・レジスタ（DR0 ～ DR3）は、それぞれブレークポイントの 32 ビット・リニア・アドレスを保持する（図 15-1. を参照）。ブレークポイントの比較は、物理アドレスへの変換が行われる前に行われる。さらに、デバッグレジスタ DR7 の内容が各ブレークポイント条件を指定する。

15.2.2. デバッグレジスタ DR4 および DR5

デバッグレジスタ DR4 および DR5 は、デバッグ拡張がイネーブルにされている（制御レジスタ CR4 の DE フラグがセットされている）ときは予約されており、DR4 および DR5 レジスタを参照しようとすると、無効オペコード例外（#UD）が発生する。デバッグ拡張がイネーブルにされていない（DE フラグがクリアされている）ときは、これらのレジスタには別名が付けられレジスタ DR6 および DR7 がデバッグされる。

15.2.3. デバッグ・ステータス・レジスタ（DR6）

デバッグ・ステータス・レジスタ（DR6）は、最後のデバッグ例外が発生したときにサンプルされたデバッグ条件を報告する（図 15-1. を参照）。このレジスタの更新は、例外が発生したときだけ行われる。このレジスタのフラグは以下の情報を示す。

B0 ～ B3（ブレークポイント条件検出）フラグ（ビット 0 ～ 3）

（セットされると）デバッグ例外が発生したとき、関連のブレークポイント条件が満たされたことを示す。これらのフラグは、デバッグ制御レジスタ DR7 の LEN*n* および R/W*n* フラグによって表される各ブレークポイントの条件が真であった場合にセットされる。これらのフラグは、DR7 レジスタの L*n* および G*n* フラグによってブレークポイントがイネーブルにされていなくてもセットされる。

BD（デバッグ・レジスタ・アクセス検出）フラグ（ビット 13）

命令ストリーム内の次の命令が、デバッグレジスタ（DR0 ～ DR7）のいずれかにアクセスすることを示す。このフラグは、デバッグ制御レジスタ DR7 の GD（一般検出）フラグがセットされたときイネーブルになる。このフラグの目的の詳細については、15.2.4.「デバッグ制御レジスタ（DR7）」を参照。

BS（シングルステップ）フラグ（ビット 14）

（セットされると）シングルステップ実行モード（EFLAGS レジスタの TF フラグでイネーブルにする）によってデバッグ例外がトリガされたことを示す。シングル・ステップ・モードは、優先順位の最も高いデバッグ例外である。BS フラグがセットされると、他のデバッグ・ステータス・ビットもすべてセットされる可能性がある。

BT（タスクスイッチ）フラグ（ビット 15）

（セットされると）タスクスイッチでターゲット・タスクの TSS の T フラグ（デバッグ・トラップ・フラグ）がセットされ、その結果デバッグ例外が発生したことを示す（TSS のフォーマットについては、6.2.1.「タスク・ステート・セグメント（TSS）」を参照）。デバッグ制御レジスタ DR7 には、この例外をイネーブルにするフラグも、ディスエーブルにす

るフラグも存在しない。TSS の T（トラップ）フラグが、この例外をイネーブルにする唯一のフラグである。

デバッグ例外の中には、ビット 0 ～ 3 をクリアするものもある。DR6 レジスタの残りの内容は、決してプロセッサによってクリアされない。デバッグ例外を識別する際の混乱を避けるため、デバッグハンドラは、割り込みをかけられたタスクに復帰する前に、このレジスタをクリアする必要がある。

15.2.4. デバッグ制御レジスタ（DR7）

デバッグ制御レジスタ（DR7）は、ブレークポイントをイネーブルまたはディスエーブルにし、ブレークポイント条件を設定する（図 15-1. を参照）。このレジスタの各フラグおよびフィールドは以下の制御を行う。

L0 ～ L3（ローカル・ブレークポイント・イネーブル）フラグ（ビット 0、2、4、6）

（セットすると）現行タスクの関連ブレークポイントのブレークポイント条件をイネーブルにする。ブレークポイント条件が検出され、かつその関連 L*n* フラグがセットしてあると、デバッグ例外が発生する。プロセッサは、切り替え先のタスクでの不要なブレークポイント条件を避けるため、タスクスイッチのたびにこれらのフラグを自動的にクリアする。

G0 ～ G3（グローバル・ブレークポイント・イネーブル）フラグ（ビット 1、3、5、7）

（セットすると）すべてのタスクの関連ブレークポイントのブレークポイント条件をイネーブルにする。ブレークポイント条件が検出され、かつその関連 G*n* フラグがセットしてあると、デバッグ例外が発生する。プロセッサは、すべてのタスクに対してブレークポイントをイネーブルにできるように、タスクスイッチでこれらのフラグをクリアしない。

LE および GE（ローカル / グローバル・イグザクト・ブレークポイント・イネーブル）フラグ（ビット 8 および 9）

（P6 ファミリ・プロセッサおよびそれ以降の IA-32 プロセッサではサポートされていない。）これらのフラグをセットすると、プロセッサはデータ・ブレークポイント条件を生じた正確な命令を検出する。他の IA-32 プロセッサとの上位および下位互換性を保証するため、インテルでは、正確なブレークポイントを必要とする場合、LE および GE フラグを 1 にセットするように推奨する。

GD（一般検出イネーブル）フラグ（ビット 13）

（セットすると）デバッグレジスタ保護がイネーブルになる。その結果、デバッグレジスタにアクセスするどの MOV 命令の前でもデバッグ例外が発生する。このような条件が検出されると、例外を発生する前に、デバッグ・ステータス・レジスタ DR6 の BD フラグがセットされる。この条件は、インサーキット・エミュレータをサポートするために用意され

ているものである。(エミュレータがデバッグレジスタにアクセスする必要があるときは、エミュレータ・ソフトウェアは GD フラグをセットして、そのときプロセッサ上で実行中のプログラムからの干渉を防止できる。) プロセッサは、デバッグ例外ハンドラにプログラムの制御を移行するときに GD フラグをクリアし、ハンドラがデバッグレジスタにアクセスできるようにする。

R/W0 ～ R/W3（読み取り / 書き込み）フィールド（ビット 16、17、20、21、24、25、28、および 29）

対応するブレークポイントのブレークポイント条件を指定する。R/W*n* フィールドの構成ビットがどのように解釈されるかは、制御レジスタ CR4 の DE（デバッグ拡張）フラグが決定する。DE フラグをセットすると、プロセッサは R/W*n* フィールド・ビットを次のように解釈する。

00 －命令実行時にのみブレークする。
01 －データ書き込み時にのみブレークする。
10 － I/O 読み取りまたは I/O 書き込み時にブレークする。
11 －データ読み取りまたは書き込み時にブレークする。ただし、命令フェッチを除く。

DE フラグをクリアすると、プロセッサは R/W*n* ビットを Intel486™ プロセッサおよび Intel386™ プロセッサの場合と同様に解釈する。つまり、次のように解釈する。

00 －命令実行時にのみブレークする。
01 －データ書き込み時にのみブレークする。
10 －未定義。
11 －データ読み取りまたは書き込み時にブレークする。ただし、命令フェッチを除く。

LEN0 ～ LEN3（長さ）フィールド（ビット 18、19、22、23、26、27、30、31）

対応のブレークポイント・アドレス・レジスタ（DR0 ～ DR3）で指定されるアドレスのメモリ位置のサイズを指定する。これらのフィールドは、次のように解釈される。

00 － 1 バイト長
01 － 2 バイト長
10 －未定義
11 － 4 バイト長

レジスタ DR7 の対応 RW*n* フィールドが 00 の場合（命令実行）は、LEN*n* フィールドも 00 にしなければならない。他のバイト長を使用した場合の影響は不確定である。これらのフィールドの使用上の詳細は、15.2.5.「ブレークポイント・フィールドの認識」を参照。

15.2.5. ブレークポイント・フィールドの認識

データ・ブレークポイントまたはI/Oブレークポイントのシーケンシャル・バイト・アドレスの範囲は、ブレークポイント・アドレス・レジスタ（デバッグレジスタDR0～DR3）、および各ブレークポイントに対するLEN*n*フィールドによって定義される。LEN*n*フィールドで、対応のデバッグレジスタ（DR*n*）で指定されるリニアアドレスから始まる1バイト、2バイト、または4バイトの範囲を指定できる。その際、2バイトの範囲はワード境界にアライメントが合わなければならず、また4バイトの範囲はダブルワード境界にアライメントが合わなければならない。I/Oブレークポイント・アドレスは、選択されたデバッグレジスタのブレークポイント・アドレスとの比較のために16ビットから32ビットにゼロ拡張される。これらの必要条件はプロセッサから課されるものであり、プロセッサはLEN*n*フィールドのビットを使用してデバッグレジスタの下位アドレスビットをマスクする。データ・ブレークポイント・アドレスでも、I/Oブレークポイント・アドレスでも、アライメントが合わない場合は予期される結果が生じない。

アクセスに関与するバイトがいずれもブレークポイント・アドレス・レジスタとそのLEN*n*フィールドによって定義される範囲内にある場合、データ読み取りまたは書き込み用のデータ・ブレークポイントがトリガされる。表15-1.に、デバッグレジスタのセットアップ例、および結果としてブレークポイントでトラップを発生するデータアクセスと発生しないデータアクセスを示す。

アライメントが合っていないオペランドに対するデータ・ブレークポイントは、2つのブレークポイントを使用して組み立てられる。その場合、各ブレークポイントはバイト境界にアライメントを合わせ、2つのブレークポイントが一体でそのオペランドをカバーする。これらのブレークポイントは、そのオペランドに対してのみ例外を発生し、隣接バイトに対しては発生しない。

命令ブレークポイント・アドレスの長さ指定は1バイトでなければならない（LEN*n*フィールドは00に設定される）。その他のオペランド・サイズに対するコード・ブレークポイントの振る舞いは不確定である。プロセッサは、命令ブレークポイント・アドレスが命令の最初のバイトを指す場合にのみ、その命令ブレークポイント・アドレスを認識する。命令がプリフィックスを持つ場合は、ブレークポイント・アドレスは最初のプリフィックスを指さなければならない。

15.3. デバッグ例外

IA-32 プロセッサは、2 つの割り込みベクタをデバッグ例外の処理専用に当てている。それらは、ベクタ 1（デバッグ例外、#DB）とベクタ 3（ブレークポイント例外、#BP）である。以降の各項で、これらの例外がどのように発生するかと、これらの例外処理用の例外ハンドラの代表的操作について説明する。

表 15-1. ブレークポイントの例

デバッグレジスタのセットアップ			
デバッグレジスタ	R/W*n*	ブレークポイント・アドレス	LEN*n*
DR0 DR1 DR2 DR3	R/W0 = 11（読み取り / 書き込み） R/W1 = 01（書き込み） R/W2 = 11（読み取り / 書き込み） R/W3 = 01（書き込み）	A0001H A0002H B0002H C0000H	LEN0 = 00（1 バイト） LEN1 = 00（1 バイト） LEN2 = 01（2 バイト） LEN3 = 11（4 バイト）
データアクセス			
操作		アドレス	アクセス長さ（バイト単位）
トラップを発生するデータ操作 - 読み取りまたは書き込み - 読み取りまたは書き込み - 書き込み - 書き込み - 読み取りまたは書き込み - 読み取りまたは書き込み - 読み取りまたは書き込み - 書き込み - 書き込み - 書き込み		 A0001H A0001H A0002H A0002H B0001H B0002H B0002H C0000H C0001H C0003H	 1 2 1 2 4 1 2 4 2 1
トラップを発生しないデータ操作 - 読み取りまたは書き込み - 読み取り - 読み取りまたは書き込み - 読み取りまたは書き込み - 読み取り - 読み取りまたは書き込み		 A0000H A0002H A0003H B0000H C0000H C0004H	 1 1 4 2 2 4

15.3.1. デバッグ例外（#DB）— 割り込みベクタ 1

デバッグ例外ハンドラは、通常、デバッガ・プログラムであるかまたはそれより上位のシステム・ソフトウェアの一部である。プロセッサは、いくつかの条件のどれに対してもデバッグ例外を発生する。デバッガは、DR6 および DR7 レジスタのフラグを調べて、どの条件が例外を発生させたか、またその他のどの条件がその例外の原因に該当する可能性があるかを判定できる。表 15-2. に、各種のブレークポイント条件が生じた後のこれらのフラグのステートを示す。

命令ブレークポイント条件および一般検出条件（15.3.1.3.「一般検出例外条件」を参照）は、結果としてフォルトを発生する。その他のデバッグ例外条件は、トラップを発生する。デバッグ例外は、フォルトまたはトラップ、あるいは同時にそれらの両方を報告することがある。以降の各項で、デバッグ例外の各クラスについて説明する。この例外に関するその他の事項については、第 5 章の「割り込み 1 ーデバッグ例外（#DB）」を参照。

15.3.1.1. 命令ブレークポイント例外条件

プロセッサは、ブレークポイント・アドレス・レジスタ（DB0 ～ DR3）が指定する、命令の実行を検出するように設定（R/W フラグが 0 にセット）されているアドレスで命令を実行しようとしたとき、命令ブレークポイントを報告する。命令ブレークポイントを報告する際、プロセッサは、このブレークポイントのターゲット命令を実行する前にフォルトクラスのデバッグ例外（#DB）を発生する。

命令ブレークポイントは最高優先順位のデバッグ例外である。命令のデコードまたは実行中に他の例外が検出されても、それらのどれよりも前に処理される。ただし、コードの命令ブレークポイントが POP SS/MOV SS 命令の直後の命令に置かれている場合は、命令ブレークポイントがトリガされないときがある。通常、POP SS/MOV SS はこのような割り込みを禁止する（『IA-32 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、中巻 A』の「MOV—Move」と『IA-32 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、中巻 B』の「POP—Pop a Value from the Stack」を参照）。

表 15-2. デバッグ例外条件

デバッグまたはブレークポイント条件	DR6 フラグの テスト結果	DR7 フラグの テスト結果	例外クラス
シングル・ステップ・トラップ	BS = 1		トラップ
DR*n* および LEN*n* によって定義されるアドレスの命令ブレークポイント	B*n*=1 で（G*n*=1 または L*n*=1）	R/W*n* = 0	フォルト
DR*n* および LEN*n* によって定義されるアドレスのデータ書き込みブレークポイント	B*n*=1 で（G*n*=1 または L*n*=1）	R/W*n* = 1	トラップ
DR*n* および LEN*n* によって定義されたアドレスのI/O読み取りまたは書き込みブレークポイント	B*n*=1 で（G*n*=1 または L*n*=1）	R/W*n* = 2	トラップ
DR*n* および LEN*n* によって定義されるアドレスのデータ読み取りまたは書き込み（命令フェッチは除く）	B*n*=1 で（G*n*=1 または L*n*=1）	R/W*n* = 3	トラップ
デバッグレジスタの内容を変更しようとした結果（通常インサーキット・エミュレーションと連携して）発生する一般検出フォルト	BD = 1		フォルト
タスクスイッチ	BT = 1		トラップ

命令ブレークポイントのデバッグ例外は、その命令が実行される前に発生するので、例外ハンドラによって命令ブレークポイントが削除されなければ、その命令が再起動され、もう一度デバッグ例外を発生したとき、プロセッサは再びその命令ブレークポイントを検出することになる。このような命令ブレークポイントでのループ現象を防止するため、IA-32 アーキテクチャは EFLAGS レジスタに RF フラグ（resume フラグ）を設けている（2.3. 節「EFLAGS レジスタのシステムフラグとフィールド」を参照）。RF フラグがセットされているときは、プロセッサは命令ブレークポイントを無視する。

IA-32プロセッサはすべてRFフラグを以下のように管理する。これらのプロセッサは、命令ブレークポイントに応答して発生したデバッグ例外以外のフォルトクラス例外に対して例外ハンドラを呼び出す前に、必ず自動的に RF フラグをセットする。命令ブレークポイントの結果によるデバッグ例外に対しては、プロセッサはデバッグ例外ハンドラを呼び出す前に RF フラグをセットしない。呼び出されたデバッグ例外ハンドラは、そこで、命令ブレークポイントをディスエーブルにするか、またはスタック上の EFLAGS イメージ内の RF フラグをセットするかを選択できる。プロセッサが例外ハンドラから復帰したときに EFLAGS イメージの RF フラグがセットされていた場合は、このフラグイメージが、プロセッサをハンドラから復帰させた IRETD 命令またはタスクスイッチ命令によって EFLAGS レジスタの RF フラグにコピーされる。その後、プロセッサは次の命令の実行中命令ブレークポイントを無視する。（POPF、POPFD、IRET 命令は RF イメージを EFLAGS レジスタに転送しないので注意する。）RF フラグをセットすると、（I/O ブレークポイントやデータ・ブレークポイントなど）他のタイプのデバッグ例外条件の検出も、非デバッグ例外の発生も抑止されなくなる。IRETD 命令の後、またはタスクスイッチを生じた JMP、CALL、または INT *n* 命令の後を除いて、命令の実行が正常に終了すると、プロセッサは EFLAGS レジスタの RF フラグをクリアする。

（プロセッサは、トラップクラス例外用、ハードウェア割り込み用、またはソフトウェア発生割り込み用の例外または割り込みハンドラを呼び出すときも、RF フラグをセットしないので注意する。）

インテル® Pentium® プロセッサの場合は、命令ブレークポイントが（ページフォルトなどの）もう 1 つのフォルトタイプの例外と同時に発生したときは、デバッグ例外ハンドラが EFLAGS イメージの RF フラグをセットしても、プロセッサは第 2 の例外の処理が終了した後に見せかけのデバッグ例外を 1 つ発生することがある。インテル Pentium プロセッサでのこの見せかけ例外の発生を防止するため、フォルトクラスの例外ハンドラはすべて EFLAGS イメージの RF フラグをセットする必要がある。

15.3.1.2. データ・メモリブレークポイント例外条件と I/O ブレークポイント例外条件

ブレークポイント・アドレス・レジスタ（DB0 ～ DR3）が指定する、データアクセスまたはI/Oアクセスを検出するように設定（R/Wフラグを1、2、または3に設定）されているメモリアドレスまたはI/Oアドレスにプロセッサがアクセスしようとすると、データメモリおよびI/O両ブレークポイントが報告される。プロセッサは、このアクセスを行った命令を実行した後に例外を発生する。したがって、これらのブレークポイント条件によってトラップクラスの例外が発生する。

データ・ブレークポイントはトラップであるため、元のデータはトラップ例外が発生する前に上書きされる。デバッガは、書き込みブレークポイント・ロケーションの内容をセーブする必要がある場合は、ブレークポイントをセットする前に元の内容をセーブしなければならない。ハンドラは、ブレークポイントがトリガされた後に、セーブした値を報告できる。ブレークポイントをトリガした命令によってストアされた新しい値のロケーションは、デバッグレジスタのアドレスを使用して知ることができる。

Intel486™プロセッサ以降の各IA-32プロセッサはDR7のGEおよびLEフラグを無視する。Intel386™プロセッサでは、GEおよびLEのいずれかまたは両フラグをセットしてイネーブルにしなければ、データ・ブレークポイントの正確な一致が生じることはない。

それに対して、P6ファミリ・プロセッサは、ブレークポイントが発生した反復サイクルの次の反復サイクルが終了するまでは、目的とするREP MOVSおよびREP STOS命令のデータ・ブレークポイントを正しく報告できない。

I/Oブレークポイント・デバッグ例外を発生するINSおよびOUTS命令がREP命令がによって繰り返される場合は、プロセッサは最初の反復サイクルが終了した後に例外を発生する。繰り返されるINSおよびOUTS命令は、メモリ・アドレス・ブレークポイント・ロケーションがアクセスされた反復サイクルの後に、I/Oブレークポイント・デバッグ例外を発生する。

15.3.1.3. 一般検出例外条件

DR7のGDフラグがセットされている場合は、デバッグレジスタ（DR0 ～ DR7）のどれでも、エミュレータやデバッガなど他のアプリケーションが使用している最中に、プログラムが同時にそのレジスタにアクセスしようとすると一般検出デバッグ例外が発生する。この追加保護機能により、必要に応じて、デバッグレジスタの完全なコントロールが保証される。デバッグ例外ハンドラは、DR6レジスタのBDフラグのステートを調べると、この条件を検出できる。プロセッサは、デバッグレジスタにアクセスし、結果としてフォルトクラスの例外を発生させるMOV命令を実行する前に一般検出デバッグ例外を発生する。

15.3.1.4. シングルステップ例外条件

プロセッサは、（命令の実行中に）EFLAGS レジスタの TF フラグがセットされていることを検出した場合、シングル・ステップ・デバッグ例外を発生する。この例外は、命令が実行された後に発生するのでトラップクラスの例外である。（プロセッサは、TF フラグをセットした命令の直後にはこの例外を発生しないので注意する。例えば、POPF 命令を使用して TF フラグをセットした場合、POPF 命令の次の命令の後までシングル・ステップ・トラップは発生しない。）

プロセッサは例外ハンドラを呼び出す前に TF フラグをクリアする。タスクスイッチ時に TSS に TF フラグがセットされていた場合、例外は切り替え先タスクの最初に実行された命令の後に発生する。

TF フラグは、通常、タスク内の特権変さらによってはクリアされないで、INT *n* および INTO 命令によってクリアされる。したがって、コードをシングルステップ実行するソフトウェア・デバッガは、INT *n* または INTO 命令を直接実行する代りに、これらの命令を識別し、エミュレートしなければならない。また、保護を維持するため、オペレーティング・システムはシングル・ステップ・トラップの後では必ず CPL を調べ、現行の特権レベルでシングルステップ実行を継続すべきかどうかを確認する必要がある。

外部割り込みが発生した場合には、割り込みの優先順位によって、シングルステップ実行が停止するよう保証される。外部割り込みとシングルステップ割り込みが同時に発生したときは、シングルステップ割り込みが先に処理される。この操作は TF フラグをクリアする。戻りアドレスをセーブするか、またはタスクを切り替えた後には、シングル・ステップ・ハンドラの最初の命令が実行される前に、外部割り込み入力のチェックが行われる。外部割り込みが未処理の場合は、これが処理される。外部割り込みハンドラは、シングル・ステップ・モードでは動作しない。割り込みハンドラをシングルステップ実行するには、割り込みハンドラを呼び出す INT *n* 命令をシングルステップ実行にする。

15.3.1.5. タスクスイッチ例外条件

切り替え先のタスクの TSS に T フラグがセットされていた場合は、プロセッサはタスクスイッチ後にデバッグ例外を発生する。この例外は、プログラムの制御が新しいタスクに移行された後で、そのタスクの最初の命令が実行される前に発生する。例外ハンドラは、DR6 レジスタの BT フラグを調べることによりこの条件を検出できる。

デバッグ例外ハンドラがタスクである場合は、その TSS の T ビットをセットしてはならないことに注意する。この規則に従わないと、プロセッサがループに入る。

15.3.2. ブレークポイント例外 ー（#BP）割り込みベクタ 3

ブレークポイント例外（割り込み 3）は、INT 3 命令の実行の結果発生する（第 5 章の「割り込み 3 ーブレークポイント例外（#BP）」を参照）。デバッガは、ブレークポイント・レジスタを使用するのと同様にブレークポイント例外を使用する。つまり、レジスタとメモリ位置を調べるためにプログラムの実行を中断するメカニズムとしてブレークポイント例外を使用する。Intel386™ より以前の IA-32 プロセッサでは、ブレークポイント例外は広く命令ブレークポイントを設定するために使用されてきた。

Intel386 プロセッサ以降の IA-32 プロセッサでは、ブレークポイント・アドレス・レジスタ（DR0 ～ DR3）でブレークポイントを設定する方が便利になっている。しかし、ブレークポイント例外は個別の例外ハンドラを呼び出せるので、デバッガをブレークポイントでコントロールするのには依然効果的である。ブレークポイント例外はまた、存在するデバッグレジスタより多数のブレークポイントを設定する必要があるときや、開発中のプログラムのソースコードにブレークポイントを配置するときにも効果的である。

インテル ® Pentium® M プロセッサでは、高速ストリング操作に対する #BP はキャッシュ・ライン境界上のときのみレポートされる。

15.4. 最新の分岐記録の概要

P6 ファミリ・プロセッサは、実施される分岐、割り込み、例外上にブレークポイントをセットして、ある分岐から次の分岐へシングルステップ実行を行う機能を備えている。インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、この機能を修正および拡張し、分岐トレースストア（BTS）バッファ内にある分岐トレース・メッセージのログをメモリに記録できるようにしている。最新分岐記録機構については、以下の節を参照のこと。

- 15.5. 節「最新の分岐、割り込み、例外の記録（インテル ® Pentium® 4 プロセッサおよびインテル ® Xeon™ プロセッサ）」
- 15.6. 節「最新の分岐、割り込み、例外の記録（P6 ファミリ・プロセッサ）」

最新分岐記録機構で追跡される IA-32 分岐命令は、JMP、J*cc*、LOOP、CALL である。

15.5. 最新の分岐、割り込み、例外の記録（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、実施される分岐、割り込み、例外を記録するために、以下の手段を提供している。

- 最後に実施された分岐、割り込み、または例外について、MSR 内の分岐レコードを最新分岐レコード（LBR）スタック MSR にストアする。分岐レコードは、分岐移行前命令アドレスおよび分岐移行先命令アドレスで構成される。
- 分岐レコードを分岐トレース・メッセージ（BTM）としてシステムバス上に送出する。
- メモリ常駐の分岐トレースストア（BTS）バッファに BTM のログを記録する。

これらの機能をサポートするため、プロセッサは、以下の MSR を用意している。

- MSR_DEBUGCTLA MSR － 最新の分岐、割り込み、例外の記録をイネーブルにする。また、実施される分岐に対するシングルステップ実行、分岐トレース・メッセージ（BTM）、分岐トレースストア（BTS）をイネーブルにする。P6 ファミリ・プロセッサでは、このレジスタを DebugCtlMSR と呼ぶ。
- CPUID 命令で返されるデバッグストア（DS）機能フラグ（ビット 21）－プロセッサが、メモリ常駐 BTS バッファに BTS を格納できるデバッグストア（DS）機構を備えていることを示す。
- IA32_MISC_ENABLE MSR － プロセッサが提供する BTS 機能を示す。
- 最新分岐レコード（LBR）スタック － インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ・ファミリ [CPUID ファミリ 0FH、モデル 0H ～ 02H] では、LBR スタックは、4 個の MSR（MSR_LASTBRANCH_0 ～ MSR_LASTBRANCH_3）で構成される循環スタックである。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ・ファミリ [CPUID ファミリ 0FH、モデル 03H] では、LBR スタックは、16 組の MSR（MSR_LASTBRANCH_0_FROM_LIP ～ MSR_LASTBRANCH_15_FROM_LIP と MSR_LASTBRANCH_0_TO_LIP ～ MSR_LASTBRANCH_15_TO_LIP）で構成される。
- 最新分岐レコードのトップ・オブ・スタック（TOS）ポインタ － インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ・ファミリ [CPUID ファミリ 0FH、モデル 0H ～ 02H] では、TOS ポインタ MSR は、記録された最新の分岐、割り込み、または例外を保持する LBR スタック内の MSR に対する 2 ビット・ポインタ（0 ～ 3）を格納する。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ・ファミリ [CPUID ファミリ 0FH、モデル 03H] では、このポインタは 4 ビット・ポインタ（0 ～ 15）になる。表 15-3.、図 15-2.、15.5.2. 項も参照のこと。

- 最新例外レコード ― 15.5.6.「最新例外レコード（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ)」を参照のこと。

表 15-3. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ・ファミリの LBR MSR スタック構造

ファミリ 0FH、モデル 0H ～ 02H の LBR MSR、位置 1DBH ～ 1DEH の MSR	MSR_LASTBRANCH_TOS（ビット 0 ～ 1）内の TOS ポインタの 10 進値
MSR_LASTBRANCH_0	0
MSR_LASTBRANCH_1	1
MSR_LASTBRANCH_2	2
MSR_LASTBRANCH_3	3
ファミリ 0FH、モデル 03H の LBR MSR、位置 680H ～ 68FH の MSR	**MSR_LASTBRANCH_TOS（ビット 0 ～ 3）内の TOS ポインタの 10 進値**
MSR_LASTBRANCH_0_FROM_LIP	0
MSR_LASTBRANCH_1_FROM_LIP	1
MSR_LASTBRANCH_2_FROM_LIP	2
MSR_LASTBRANCH_3_FROM_LIP	3
MSR_LASTBRANCH_4_FROM_LIP	4
MSR_LASTBRANCH_5_FROM_LIP	5
MSR_LASTBRANCH_6_FROM_LIP	6
MSR_LASTBRANCH_7_FROM_LIP	7
MSR_LASTBRANCH_8_FROM_LIP	8
MSR_LASTBRANCH_9_FROM_LIP	9
MSR_LASTBRANCH_10_FROM_LIP	10
MSR_LASTBRANCH_11_FROM_LIP	11
MSR_LASTBRANCH_12_FROM_LIP	12
MSR_LASTBRANCH_13_FROM_LIP	13
MSR_LASTBRANCH_14_FROM_LIP	14
MSR_LASTBRANCH_15_FROM_LIP	15
ファミリ 0FH、モデル 03H の LBR MSR、位置 6C0H ～ 6CFH の MSR	
MSR_LASTBRANCH_0_TO_LIP	0
MSR_LASTBRANCH_1_TO_LIP	1
MSR_LASTBRANCH_2_TO_LIP	2
MSR_LASTBRANCH_3_TO_LIP	3
MSR_LASTBRANCH_4_TO_LIP	4
MSR_LASTBRANCH_5_TO_LIP	5
MSR_LASTBRANCH_6_TO_LIP	6
MSR_LASTBRANCH_7_TO_LIP	7
MSR_LASTBRANCH_8_TO_LIP	8
MSR_LASTBRANCH_9_TO_LIP	9
MSR_LASTBRANCH_10_TO_LIP	10
MSR_LASTBRANCH_11_TO_LIP	11
MSR_LASTBRANCH_12_TO_LIP	12
MSR_LASTBRANCH_13_TO_LIP	13
MSR_LASTBRANCH_14_TO_LIP	14
MSR_LASTBRANCH_15_TO_LIP	15

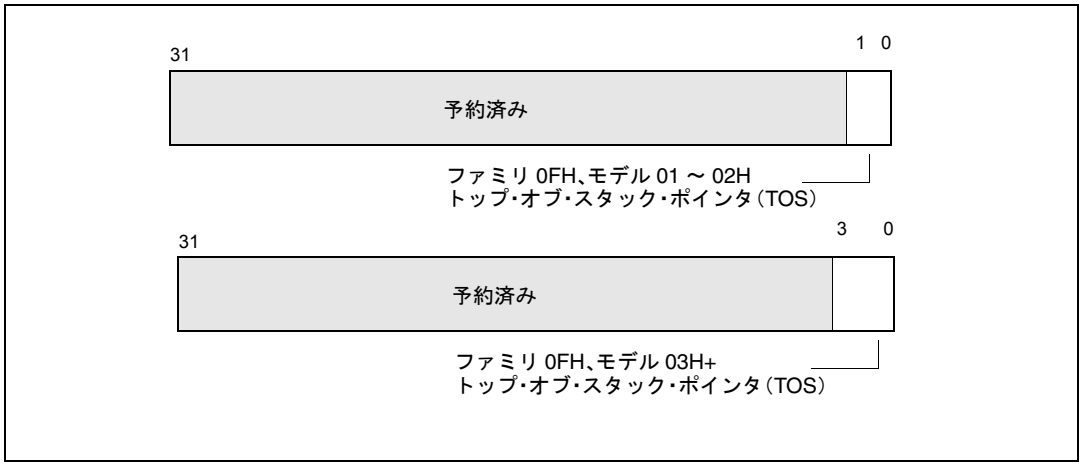


図 15-2. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ・ファミリの MSR_LASTBRANCH_TOS MSR のレイアウト

以降の各項では、MSR_DEBUGCTLA MSR と、さまざまな最新分岐記録機構を説明する。前述した最新分岐記録用の MSR の詳細については、付録 B「モデル固有レジスタ（MSR）」を参照のこと。

15.5.1. MSR_DEBUGCTLA MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

MSR_DEBUGCTLA MSR を使って、前項で説明した各種の最新分岐記録機構をイネーブルおよびディスエーブルにする。このレジスタは、特権レベルが 0 または実アドレスモードで動作中に、WRMSR 命令を使用して書き込むことができる。ユーザがこのレジスタにアクセスするには、保護モードのオペレーティング・システム・プロシージャが必要である。図 15-3. は、MSR_DEBUGCTLA MSR のフラグを示している。これらのフラグの機能は、以下のとおりである。

LBR（最新分岐 / 割り込み / 例外）フラグ（ビット 0）

セットされている場合は、プロセッサは、最後に行った分岐、割り込み、または例外の実行トレースを、（デバッグ例外が生成される前に）最新分岐レコード（LBR）スタックに記録する。それぞれの分岐、割り込み、または例外は、64 ビットの分岐レコードとして記録される（15.5.2.「LBR スタック（インテル ® Pentium® 4 プロセッサおよびインテル ® Xeon™ プロセッサ）」を参照）。プロセッサは、デバッグ例外が生成されるたびに（例えば、命令ブレークポイント、データ・ブレークポイント、またはシングル・ステップ・トラップが発生したときに）、このフラグをクリアする。

BTF（分岐シングルステップ実行）フラグ（ビット 1）

セットされている場合は、プロセッサは、EFLAGS レジスタの TF フラグを「命令シングルステップ実行」フラグとしてではなく、「分岐シングルステップ実行」フラグとして扱う。この機構により、分岐、割り込み、例外の各処理をプロセッサにシングルステップで実行できる。BTF フラグの詳細については、15.5.4.「分岐、例外、割り込みのシングルステップ実行」を参照のこと。

TR（トレース・メッセージ・イネーブル）フラグ（ビット 2）

セットされると、分岐トレース・メッセージがイネーブルになる。これ以降、プロセッサは、実施された分岐、割り込み、または例外を検出すると、分岐レコードを分岐トレース・メッセージ（BTM）としてシステムバス上に送出する。TR フラグの詳細については、15.5.5.「分岐トレース・メッセージ」を参照のこと。

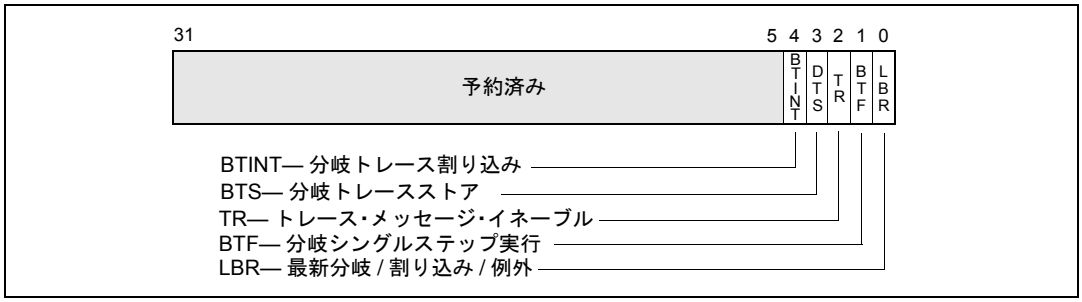


図 15-3. MSR_DEBUGCTLA MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

BTS（分岐トレースストア）フラグ（ビット 3）

セットされると BTS 機能がイネーブルになり、DS セーブ領域の一部であるメモリ常駐 BTS バッファに、BTM のログが記録される（15.9.5.「DS セーブ領域」を参照）。

BTINT（分岐トレース割り込み）フラグ（ビット 4）

セットされている場合は、BTS 機能により、BTS バッファがいっぱいになった時点で割り込みが生成される。クリアされている場合は、BTM のログが BTS バッファに循環方式で記録される（この機構は、15.5.7.「分岐トレースストア（BTS: Branch Trace Store）」を参照）。

15.5.2. LBR スタック（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

LBR スタックは、LBR MSR から構成される。プロセッサは、これらの LBR MSR を循環式スタックとして扱う。TOS ポインタ（MSR_LASTBRANCH_TOS MSR）は、スタック上に配置された最新の（最後の）分岐レコードを含む LBR MSR（または LBR MSR のペア）を指す。TOS は、新しい分岐レコードをスタックに配置する前に、1 だけインクリメントされる。TOS ポインタは、最大値に達すると 0 に循環される。表 15-3. と図 15-2. を参照。

MSR_LASTBRANCH_TOS MSR と LBR MSR スタック内のレジスタは読み取り専用で、RDMSR 命令を使用して読み取ることができる。

図 15-4. は、LBR MSR（または MSR のペア）内の分岐レコードのレイアウトを示している。それぞれの分岐レコードは、2 つのリニアアドレスから構成される。これらのリニアアドレスは、分岐、割り込み、または例外の「移行前」命令ポインタおよび「移行先」命令ポインタを表している。移行前アドレスおよび移行先アドレスの内容は、分岐の発生個所に応じて変わる。

- 実施される分岐 ― 実施される分岐用のレコードの場合は、分岐命令のアドレスが「移行前」アドレスになり、分岐のターゲット命令が「移行先」アドレスになる。
- 割り込み ― 割り込み用のレコードの場合は、その割り込みについて保存されている戻り命令ポインタ（RIP）が「移行前」アドレスとなり、割り込みハンドラルーチンの先頭の命令のアドレスが「移行先」アドレスとなる。RIP は、割り込みハンドラから戻ったときに実行される次の命令のリニアアドレスである。
- 例外 ― 例外用のレコードの場合は、例外を生成した命令のリニアアドレスが「移行前」アドレスとなり、例外ハンドラルーチンの先頭の命令のアドレスが「移行先」アドレスとなる。

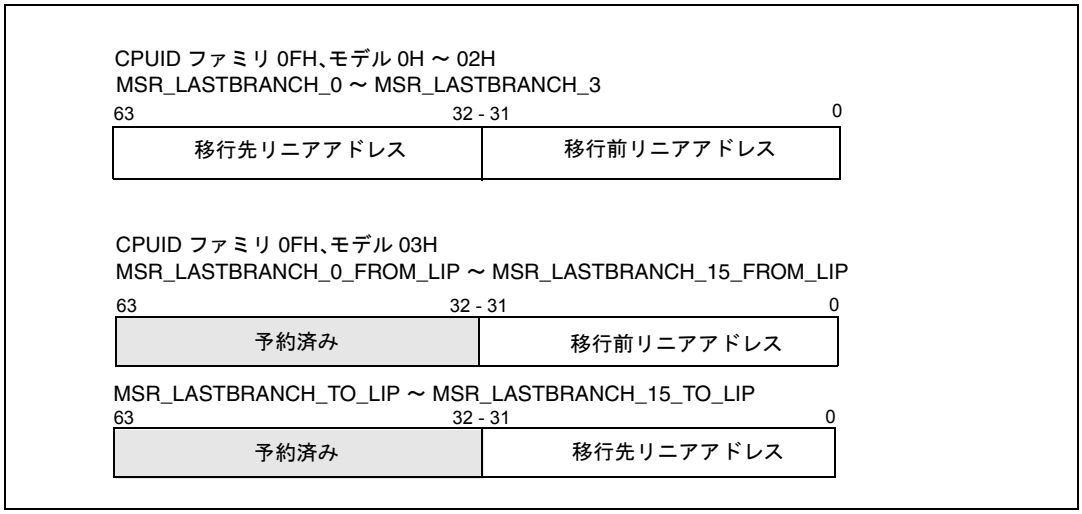


図 15-4. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ・ファミリの LBR MSR 分岐レコードのレイアウト

分岐命令と一緒に例外や割り込みが発生すると、追加情報が保存される。分岐命令がトラップタイプの例外を生成すると、LBR スタックに2つの分岐レコードがストアされる。すなわち、分岐命令用の分岐レコードに続いて、例外用の分岐レコードがストアされる。

分岐命令がフォルトタイプの例外を生成すると、分岐レコードは、分岐命令用の LBR スタックではなく、例外用の LBR スタックにストアされる。この場合、分岐命令のロケーションは、プロセッサがスタックに書き込む例外スタックフレーム内の CS レジスタおよびEIP レジスタから求められる。

分岐命令の直後に割り込みが発生すると、割り込みレコードの前の分岐命令用の LBR スタックに分岐レコードがストアされる。

15.5.3. 分岐、例外、割り込みのモニタリング（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

MSR_DEBUGCTLA MSR の LBR フラグをセットすると、プロセッサは、実施された分岐、割り込み、例外（デバッグ例外は除く）の分岐レコードを自動的に LBR スタックの MSR に記録し始める。

プロセッサは、デバッグ例外（#DB）を生成すると、例外ハンドラを実行する前に LBR フラグを自動的にクリアクリアする。この処理では、以前にストアされた LBR スタックの MSR はクリアしない。そのため、実施された分岐、割り込み、または例外について、最後の4つの分岐レコードはデバッグ・プログラムの分析用に保持される。

デバッガは、LBR スタックのリニアアドレスを使用して、ブレークポイント・アドレス・レジスタ（DR0～DR3）のブレークポイントをリセットできる。したがって、特定のバグからそのソースに遡るバックワード・トレースが可能になる。

LBR フラグがクリアされても、IA32_DEBUGCTLTR MSR 内の TR フラグがセットされたままになっている場合は、プロセッサは LBR スタック MSR の更新を続ける。これは、LBR スタック内のエントリから BTM 情報を生成する必要があるためである（14.5.5 項を参照）。#DB によって自動的に TR フラグがクリアされるわけではない。

15.5.4. 分岐、例外、割り込みのシングルステップ実行

MSR_DEBUGCTLA MSR の BTF フラグと EFLAGS レジスタの TF フラグをソフトウェアが共にセットすると、プロセッサは次に分岐を実行するか、割り込みを処理するか、例外を生成したときにシングル・ステップ・デバッグ例外を発生する。このメカニズムにより、デバッガは、分岐、割り込み、または例外の結果生じる制御の移行をシングルステップ実行できることになる。この「制御フローのシングルステップ実行」は、命令のシングルステップ実行で検索範囲をさらに狭める前に、バグを特定のコードブロックに隔離するのに効果的である。プロセッサがデバッグ例外を発生したときに BTF フラグがセットされていた場合は、プロセッサは BTF フラグを TF フラグと一緒にクリアする。デバッガは、その後、プログラムの実行を再開して制御フローのシングルステップ実行を継続する前に、BTF フラグと TF フラグをセットし直さなければならない。

15.5.5. 分岐トレース・メッセージ

MSR_DEBUGCTLA MSR に TR フラグをセットすると、分岐トレース・メッセージ（BTM）がイネーブルになる。これ以降、プロセッサは、分岐、例外、または割り込みを検出すると、分岐レコードを BTM としてシステムバスに送出する。システムバスを監視しているデバッグデバイスがこれらのメッセージを読み取り、実施された分岐、割り込み、例外イベントと操作を同期できる。

実施された分岐と一緒に割り込みまたは例外が発生すると、さらにそれらの BTM がバス上に送出される（15.5.3.「分岐、例外、割り込みのモニタリング（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照）。

このフラグをセットすると、プロセッサのパフォーマンスは大幅に低下する。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサでは、P6 ファミリ・プロセッサとは異なり、MSR_DEBUGCTLA MSR に TR フラグと LBR フラグの両方がセットされている場合は、BTM をシステムバスに送出する一方で、LBR スタックの MSR 内の分岐レコードを収集できる。

15.5.6. 最新例外レコード（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサは、2つの32ビットMSR（MSR_LER_TO_LIPおよびMSR_LER_FROM_LIP MSR）を用意している。これらは、P6ファミリ・プロセッサのLastExceptionToIPとLastExceptionFromIP MSRの機能を複製したものである。MSR_LER_TO_LIPおよびMSR_LER_FROM_LIP MSRには、例外または割り込みが生成される前にプロセッサが実施した最後の分岐についての分岐レコードが入っている。

15.5.7. 分岐トレーススストア（BTS: Branch Trace Store）

実施された分岐、割り込み、および例外のトレースは、コードのデバッグの際に役に立つ。特定のコード・ロケーションへ到達するために選択するデシジョン・パスを判定できるからである。インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサでは、実施された分岐、割り込み、および例外の記録をキャプチャして、それらを、最新分岐レコード（LBR: Last Branch Record）スタックのMSRに保存したり、BTMとしてシステムバス上に送出する機構を提供している。分岐トレーススストア（BTS）機構においては、DSセーブ領域の一部であるメモリ常駐BTSバッファに分岐レコードを保存する機能も提供している（15.9.5.「DSセーブ領域」を参照）。BTSバッファは、循環するように設定しておけば、最新の分岐レコードを常に参照できる。また、バッファがいっぱいに近づいた時点で割り込みを生成するように設定すれば、分岐レコードをすべて保存できる。

15.5.7.1. BTS 機能の検出

CPUID命令で返されるDS機能フラグ（ビット21）がセットされている場合は、プロセッサ上でDS機構が使用可能である。DS機構は、BTS（およびPEBS）機能をサポートする。このビットがセットされている場合は、以下のBTS機能が使用できる。

- IA32_MISC_ENABLE MSR内のBTS_UNAVAILABLEフラグがクリアされている場合は、BTS機能（MSR_DEBUGCTLA MSR内のBTSビットおよびBTINTビットをセットする機能など）が使用できることを示す。
- DSセーブ領域を指すように、IA32_DS_AREA MSRをプログラムすることが可能。

15.5.7.2. DS セーブ領域のセットアップ

BTSバッファに分岐レコードを保存するには、以下の手順にしたがって、まずDSセーブ領域をメモリ内にセットアップする必要がある。DSセーブ領域にBTSバッファまたはPEBSバッファをセットアップする方法については、15.5.7.3.「BTSバッファのセットアップ」、または15.9.8.3.「PEBSバッファのセットアップ」を参照のこと。

1. メモリ内に DS バッファ管理情報領域を作成する（レイアウトについては、15.9.5.「DS セーブ領域」を参照)。この項の補足情報を参照のこと。
2. DS バッファ管理領域のベース・リニア・アドレスを IA32_DS_AREA MSR に書き込む。
3. xAPIC LVT 内の性能カウンタエントリを、固定伝達モードおよびエッジ・センシティブにしてセットアップする。8.5.1.「ローカル・ベクタ・テーブル」を参照のこと。
4. xAPIC LVT 内の性能カウンタエントリに対応付けられたベクタ用に、割り込みハンドラを IDT 内に設定する。
5. 割り込みを処理する割り込みサービスルーチンを作成する（15.5.7.4.「DS 割り込みサービスルーチンの作成」を参照)。

DS セーブ領域には、以下の制約を適用させる必要がある。

- 3 つの DS セーブ領域セクションは非ページプールから割り振り、アクセス済みおよびダーティとしてマークする必要がある。バッファを含むページを常に提示し、それらをアクセス済みおよびダーティとしてマークするのは、オペレーティング・システムで行わなければならない。すなわち、オペレーティング・システムは、これらのページに対して、「レイジー」なページ・テーブル・エントリ伝搬を実行できない。
- DS セーブ領域は 1 ページよりも大きくできるが、その場合は、連続したリニアアドレスに複数のページをマップする必要がある。バッファはページを共有できるため、4K バイト境界にアライメントを合わせる必要はない。パフォーマンス上の理由から、バッファのベースは、ダブルワード境界にアライメントを合わせる必要がある。また、キャッシュ・ライン境界にアライメントを合わせるのが望ましい。
- BTS バッファと PEBS バッファのバッファサイズは、対応するレコードサイズの整数倍にするのが望ましい。
- プリサイス・イベント・レコード・バッファは、割り込み処理の待機中に発生しうるプリサイス・イベント・レコードの個数を余裕を持って保持できる大きさにすること。
- DS セーブ領域は、カーネル空間内にあること。自己修正コード・アクションが起動されるのを避けるため、DS セーブ領域はコードと同じページに配置しないこと。
- バッファにメモリタイプの制限はない。ただし、パフォーマンス上の理由から、WB メモリタイプとしてバッファを指定するのが望ましい。
- DS セーブ領域が有効であるときに、システムが A20M モードへ移行しないようにすること。あるいは、バッファ境界内のすべてのアドレスのビット 20 を 0 にセットすること。
- バッファを含むページを、すべてのプロセスについて同じ物理アドレスにマップすること。これにより、制御レジスタ CR3 が変更されても、DS アドレスは変更されないようになる。

- DS セーブ領域は、APIC がイネーブルであるシステム上でのみ使用すること。APCI 内の LVT 性能カウンタエントリは、トラップゲートではなく割り込みゲートを使用するように初期化する必要がある。

15.5.7.3. BTS バッファのセットアップ

MSR_DEBUGCTLA MSR の 3 つのフラグ、TR、BTS、BTINT（表 15-4. を参照）は、分岐レコードの生成を制御し、それらを BTS バッファへストアするのを制御する。TR フラグは、BTM の生成をイネーブルにする。BTS フラグをクリアすると BTM はシステムバスに送出され、BTS フラグをセットすると BTM は BTS バッファにストアされる。BTM をシステムバスへ送出するのと同時に BTS バッファへ記録ができない。BTINT フラグは、BTS バッファがいっぱいになったときの割り込みの生成をイネーブルにする。BTINT フラグをクリアすると、BTS バッファは循環式バッファになる。

表 15-4. MSR_DEBUGCTLA MSR フラグのエンコーディング

TR	BTS	BTINT	説明
0	X	X	分岐トレース・メッセージ（BTM）がオフ
1	0	X	BTM を生成する
1	1	0	BTM を BTS バッファにストアする（ここでは循環式バッファとして使用）
1	1	1	BTM を BTS バッファにストアし、バッファがいっぱいに近づいたら割り込みを生成する

以下の手順では、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサをセットアップして、DS セーブ領域にある BTS バッファ内の分岐レコードを収集する方法を説明する。

1. DS バッファ管理領域の BTS バッファベース、BTS インデックス、BTS 絶対最大値、BTS 割り込みしきい値フィールドに値をセットし、メモリ内の BTS バッファをセットアップする。
2. MSR_DEBUGCTLA MSR の TR フラグと BTS フラグをセットする。
3. 循環式 BTS バッファをセットアップする場合は、MSR_DEBUGCTLA MSR の BTINT フラグをクリアする。BTS バッファがいっぱいに近づいた時点で割り込みを生成する場合は、BTINT フラグをセットする。

15.5.7.4. DS 割り込みサービスルーチンの作成

BTS、非プリサイス・イベント・ベースのサンプリング、および PEBS 機能では、同じ割り込みベクタと割り込みサービスルーチン（デバッグストア割り込みサービスルーチン（DS ISR）と呼ばれる）を共有する。BTS、非プリサイス・イベント・ベース・サンプリング、PEBS 割り込みを処理するには、独立したハンドラルーチンが DS ISR に含まれていなければならない。DS ISR を作成して、BTS、非プリサイス・イベ

ント・ベース・サンプリング、または PEBS 割り込みを処理する場合は、以下のガイドラインに従うこと。

- バッファ記憶領域を保護するため、DS 割り込みサービスルーチン（ISR）はカーネルドライバに含め、CPL=0 で動作させること。
- BTS、非プリサイス・イベント・ベース・サンプリング、PEBS 機能では同じ割り込みベクタを共有するため、DS ISR は、これらの機能から起こり得る割り込みの原因をすべてチェックし、制御を適切なハンドラに渡す必要がある。

  バッファ・インデックスが、指定された割り込みしきい値に一致またはそれを超えた場合、BTS および PEBS バッファのオーバーフローは割り込みの原因になる。割り込みの原因としての非プリサイス・イベント・ベース・サンプリングを検出するには、カウンタ・オーバーフローをチェックする。
- マルチプロセッサ・システムでは、プロセッサごとに独立したセーブ領域、バッファ、およびステートが必要である。
- DS セーブ領域へのアクセスで競合状態が起きないようにするため、ISR に入ったら、分岐トレース・メッセージと PEBS をディスエーブルにする必要がある。これを行うには、MSR_DEBUGCTLA MSR の TR フラグと、IA32_PEBS_ENABLE MSR のプリサイス・イベント・イネーブル・フラグをクリアすればよい。これらの設定は、ISR を出るときに元の値に復元する必要がある。
- バッファがいっぱいになった時点で循環モードが選択されていなければ、プロセッサは、DS セーブ領域をディスエーブルにしない。DS の現行設定を保持しておいて、ISR の終了時に復元する必要がある。
- 所定のバッファのデータを現行インデックスの直前まで読み取ったあと、ISR でバッファ・インデックスをバッファの先頭にリセットする必要がある。そうしないと、次回 ISR を呼び出したときに、インデックスまでのすべての内容が新規エントリのように見えることになる。
- ISR では、性能カウンタ LVT エントリのマスクビットをクリアする必要がある。
- ISR が PEBS によって発生したオーバーフロー PMI を処理している場合は、ISR は CCCR の ENABLE ビットを再び有効にする必要がある。

15.6. 最新の分岐、割り込み、例外の記録（P6 ファミリ・プロセッサ）

P6 ファミリ・プロセッサは、プロセッサが最後に行った分岐、割り込みまたは例外処理の記録用として、DebugCtlMSR、LastBranchToIP、LastBranchFromIP、LastExceptionToIP、LastExceptionFromIP という 5 つの MSR を備えている。これらのレジスタを使用して、最新の分岐記録を収集したり、分岐、割り込み、および例外に対

してブレークポイントを設定したり、分岐から次の分岐までシングルステップ実行できる。

前述した最新分岐記録用の MSR の詳細については、付録 B「モデル固有レジスタ（MSR）」を参照のこと。

15.6.1. DebugCtlMSR レジスタ（P6 ファミリ・プロセッサ）

P6 ファミリ・プロセッサの DebugCtlMSR レジスタは、最新の分岐、割込み、例外の記録をイネーブルにする。また、実行された分岐に対するブレークポイント、ブレークポイント報告ピン、トレース・メッセージをイネーブルにする。このレジスタには、特権レベル 0 での動作時または実アドレスモード時に WRMSR 命令を使用して書くことができる。このレジスタへのユーザアクセスを可能にするには、保護モードのオペレーティング・システム・プロシージャが必要である。図 15-5. に P6 ファミリ・プロセッサの DebugCtlMSR レジスタのフラグを示す。これらのフラグの機能を以降で説明する。

LBR（最新分岐 / 割り込み / 例外）フラグ（ビット 0）

セットすると、プロセッサはデバッグ例外が発生する前の最後に行った分岐、および例外または割り込み処理のソースアドレスとターゲット・アドレス（LastBranchToIP、LastBranchFromIP、LastExceptionToIP、LastExceptionFromIP の各 MSR の中）を記録する。プロセッサは、命令ブレークポイント、データ・ブレークポイント、またはシングル・ステップ・トラップなどのデバッグ例外が発生するたびにこのフラグをクリアする。

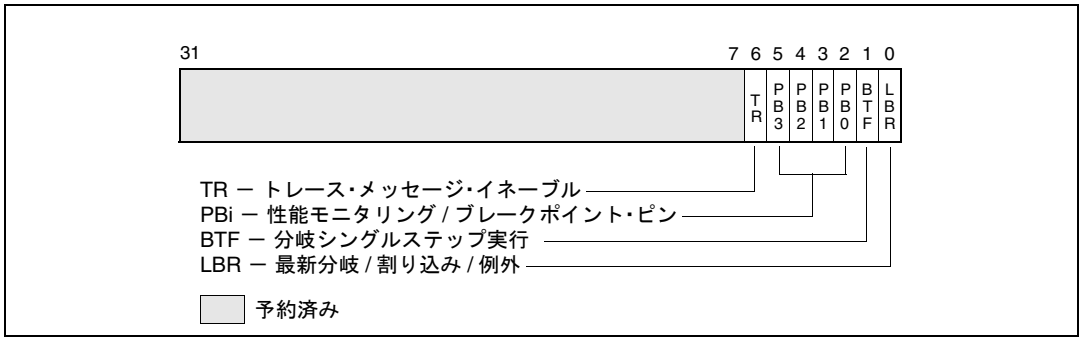


図 15-5. DebugCtlMSR レジスタ（P6 ファミリ・プロセッサ）

BTF（分岐シングルステップ実行）フラグ（ビット 1）

セットすると、プロセッサは EFLAGS レジスタの TF フラグを「分岐シングルステップ実行」フラグとして扱う（15.5.4.「分岐、例外、割り込みのシングルステップ実行」を参照）。

PBi（性能モニタリング / ブレークポイント・ピン）フラグ（ビット 2 ～ 5）
これらのフラグをセットすると、プロセッサの性能モニタリング / ブレークポイント・ピン（BP0#、BP1#、BP2#、BP3#）が対応するブレークポイント・アドレス・レジスタ（DR0 ～ DR3）のブレークポイントの一致を報告する。ブレークポイントの一致が発生すると、プロセッサは対応 BPi# ピンをいったんアサートしてからディアサートする。PB*i* フラグをクリアすると、性能モニタリング / ブレークポイント・ピンは性能イベントを報告する。プロセッサの実行は性能イベントの報告によって左右されない。

TR（トレース・メッセージ・イネーブル）フラグ（ビット 6）
セットすると、15.5.5.「分岐トレース・メッセージ」で説明したようにトレース・メッセージがイネーブルになる。このフラグをセットすると、プロセッサの性能が大幅に低下する。トレース・メッセージをイネーブルにしているときは、LastBranchToIP、LastBranchFromIP、LastExceptionToIP、LastExceptionFromIP の各 MSR の値は不確定である。

15.6.2. 最新分岐 MSR と最新例外 MSR（P6 ファミリ・プロセッサ）

LastBranchToIP および LastBranchFromIP 両 MSR は、デバッグ例外が発生する前の最後にプロセッサが行った分岐、割り込みまたは例外処理の命令ポインタを記録するための 32 ビット・レジスタである。分岐が行われると、プロセッサは分岐命令のアドレスを LastBranchFromIP MSR にロードし、分岐のターゲット・アドレスを LastBranchToIP MSR にロードする。

割り込みまたは（デバッグ例外以外の）例外が発生すると、例外または割り込みによって中断された命令のアドレスが LastBranchFromIP MSR にロードされ、呼び出される例外または割り込みハンドラのアドレスが LastBranchToIP MSR にロードされる。

LastExceptionToIP および LastExceptionFromIP 両 MSR（やはり 32 ビット・レジスタ）は、例外または割り込みが発生する前の最後にプロセッサが行った分岐処理の命令ポインタを記録する。例外または割り込みが発生すると、LastBranchToIP MSR と LastBranchFromIP MSR の内容がそれぞれ LastExceptionToIP MSR と LastExceptionFromIP MSR にコピーされてから、例外または割り込みの「移行先アドレス」と「移行前アドレス」が LastBranchToIP MSR と LastBranchFromIP MSR に記録される。

これらのレジスタは RDMSR 命令を使用して読むことができる。

LastBranchToIP、LastBranchFromIP、LastExceptionToIP、LastExceptionFromIP MSR にストアされる値は、現在のコード・セグメントへのオフセットである。これは、インテル® Pentium® 4 プロセッサの最新分岐レコードに保存されるリニアアドレスとは異なるものである。

15.6.3. 分岐、例外、割り込みのモニタリング（P6 ファミリ・プロセッサ）

DebugCtlMSR レジスタの LBR フラグをセットすると、プロセッサは、実行した分岐、発生した（デバッグ例外以外の）例外、処理した割り込みの記録を自動的に開始する。分岐、割り込み、または例外が発生するたびに、プロセッサは「移行先命令ポインタ」と「移行前命令ポインタ」をそれぞれ LastBranchToIP MSR と LastBranchFromIP MSR に記録する。割り込みと例外については、プロセッサは、さらに、LastBranchToIP MSR と LastBranchFromIP MSR の内容をそれぞれ LastExceptionToIP MSR と LastExceptionFromIP MSR にコピーしてから、割り込みまたは例外の「移行先アドレス」および「移行前アドレス」を記録する。

プロセッサは、デバッグ例外（#DB）を発生すると、例外ハンドラを実行する前に LBR フラグを自動的にクリアするが、最新分岐および最新例外 MSR の内容は変えない。そのため、最後に行われた分岐、割り込み、または例外処理のアドレスはそれぞれ LastBranchToIP MSR と LastBranchFromIP MSR に保持されており、割り込みまたは例外の前の最後の分岐のアドレスは LastExceptionToIP MSR と LastExceptionFromIP MSR に保持されている。

デバッガは、最新の分岐、割り込み、例外のいずれかあるいはそれらの任意の組み合わせのアドレスを、スタックから読み取ったコード・セグメント・セレクタと組み合わせてブレークポイント・アドレス・レジスタ（DR0 ～ DR3）のブレークポイント設定を元に戻すことができ、したがって、特定のバグからそのソースに遡るバックワード・トレースが可能になる。LastBranchToIP、LastBranchFromIP、LastExceptionToIP、LastExceptionFromIP の各 MSR に記録されている命令ポインタはコード･セグメントのオフセットであるため、ソフトウェアは制御の移行に関連するコード・セグメントのセグメント・ベース・アドレスを知り、ブレークポイント・アドレス・レジスタにロードするリニアアドレスを計算しなければならない。セグメント・ベース・アドレスを知るには、スタックからコード･セグメントのセグメント･セレクタを読み取り、それを使用してセグメントのセグメント・ディスクリプタの GDT または LDT 内のロケーションを知る。次に、セグメント・ベース・アドレスをセグメント・ディスクリプタから読み取ることができる。

デバッグ例外ハンドラからプログラムの実行を再開する前に、ハンドラは LBR フラグを再びセットして、最新分岐および最新例外 / 割り込みの記録をイネーブルに戻す必要がある。

15.7. タイムスタンプ・カウンタ

（インテル® Pentium® プロセッサ以降の）IA-32 アーキテクチャは、プロセッサ・イベント発生の相対時間をモニタおよび判別するためのタイムスタンプ・カウンタ・メカニズムを持っている。タイムスタンプ・カウンタのアーキテクチャは、タイムスタンプ・カウンタ IA32_TIME_STAMP_COUNTER MSR（P6 ファミリ・プロセッサおよび

インテル Pentium プロセッサでは TSC MSR と呼ばれる)、タイムスタンプ・カウンタ読み出し命令（RDTSC)、CPUID 命令で読み出せる機能ビット（TCS フラグ)、制御レジスタ CR4 内のタイムスタンプ・カウンタ無効ビット（TSD フラグ）で構成される。

CPUID 命令の実行後、（セットされているとき）EDX レジスタの TSC フラグ（ビット 4）は特定の IA-32 プロセッサにタイムスタンプ・カウンタが存在することを示す。(『IA-32 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、中巻 A』の第 3 章の「CPUID—CPU Identification」を参照。)

(インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサに内蔵されている）タイムスタンプ・カウンタは 64 ビットのカウンタであり、プロセッサのハードウェア・リセット後に 0 に設定される。リセット後は、HLT 命令または外部 STPCLK# ピンによってプロセッサが停止されても、このカウンタはプロセッサ・クロック・サイクルごとに値を 1 ずつ増加する。ただし、外部 DPSLP# ピンのアサートによって、タイムスタンプ・カウンタが停止することがある。また、Intel SpeedStep® テクノロジの移行により、プロセッサの内部クロック周波数にしたがって、タイムスタンプ・カウンタの値が増加する周波数が変化することがある。

RDTSC 命令は、タイムスタンプ・カウンタ読み取り用の命令であり、64 ビットのカウンタ・ラップアラウンドの場合を除き、実行されるたびに、常に異なる増加値を返すよう保証されている。インテルでは、このタイムスタンプ・カウンタは、その周波数と構成によるアーキテクチャ上、0 にリセットしてから 10 年間はラップアラウンドを生じないことを保証している。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサでは、カウンタのラップアラウンド周期は数千年である。

RDTSC 命令は、通常、任意の特権レベルおよび仮想 8086 モードで動作するプログラムおよびプロシージャから実行できる。また、制御レジスタ CR4 の TSD フラグ（ビット 2）により、この命令を特権レベル 0 で動作するプログラムとプロシージャだけに使用させるように制限できる。安全保護を考慮するオペレーティング・システムでは、一般的に、システム初期化時に TSD フラグをセットして、タイムスタンプ・カウンタへのユーザアクセスをディスエーブルにする。タイムスタンプ・カウンタへのユーザアクセスをディスエーブルにしているオペレーティング・システムでは、ユーザアクセス可能なプログラミング・インターフェイスを介して RDTSC 命令をエミュレートする必要がある。

RDTSC 命令は、他の命令とシリアル化もオーダリングもされることはない。したがって、カウンタを読むのに、RDTSC 命令は必ずしもそれより前のすべての命令の実行が終わるまで待っているわけではない。同様に、RDTSC 命令の操作が行われる前にそれより後の命令も実行を開始できる。

RDMSR 命令と WRMSR 命令は、それぞれに、タイムスタンプ・カウンタを MSR（MSR アドレス 10H）として読み出しと書き込みを実行できる。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、RDMSR 命令を使用して（RDTSC 命令を使用する場合と同様に）タイムスタンプ・カウンタの 64 ビットすべてを読み出せる。ただし、WRMSR 命令を使用してタイムスタンプ・カウンタへの書き込みを行う場合、書き込めるのはタイムスタンプ・カウンタの下位 32 ビットだけであり、上位 32 ビットはすべて 0 にクリアされる。

15.8. 性能モニタリングの概要

性能モニタリングの機能は、インテル® Pentium® プロセッサで、モデル固有の性能モニタリング・カウンタ MSR のセットによって IA-32 アーキテクチャに導入された。これらのカウンタによって、プロセッサ性能パラメータの範囲をモニタし、計測できる。これらのカウンタから得られた情報は、システムおよびコンパイラの性能のチューニングに使用できる。

インテル P6 ファミリ・プロセッサでは、モニタ可能なイベントの範囲を拡張し、モニタ対象イベント選択のコントロールを増強させるよう、性能モニタリング・メカニズムが修正、強化された。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサには、新しい性能モニタリング機構と、カウント可能な新規の一連の性能イベントが導入されている。

インテル Pentium プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサで規定されている性能モニタリング機構と性能イベントは、アーキテクチャ的なものではない。性能モニタリング機構と性能イベントは、どちらもモデル固有のもので、この 3 つの IA-32 プロセッサ・ファミリの間で互換性はない。

以降の各項で、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサの性能モニタリング・メカニズムについて説明する。

- 15.9.「性能モニタリング（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」
- 15.11.「性能モニタリング・カウンタ（P6 ファミリ・プロセッサ）」
- 15.12.「性能モニタリング（インテル® Pentium® プロセッサ）」

15.9. 性能モニタリング（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサで提供されている性能モニタリング機構は、P6ファミリ・プロセッサやインテル® Pentium® プロセッサのものとはかなり異なっている。WRMSR命令、RDMSR命令、RDPMC命令による性能イベントの選択、フィルタリング、カウント、読み取りの基本概念は変わっていないが、セットアップ機構とMSRのレイアウトがP6ファミリ・プロセッサやインテルPentiumプロセッサとは異なっており、互換性はない。また、RDPMC命令が拡張され、インテルPentium 4プロセッサおよびインテルXeonプロセッサで追加提供された性能カウンタの読み取りと、性能カウンタの高速読み取りが可能になっている。

インテルPentium 4プロセッサおよびインテルXeonプロセッサで提供されているイベント・モニタリング機構は、以下の機能から構成される。

- IA32_MISC_ENABLE MSR。これは、IA-32プロセッサで性能モニタリング機能およびプリサイス・イベント・ベース・サンプリング（PEBS）機能が使用できるかどうかを示す。
- イベント選択制御（ESCR）MSR。特定の性能カウンタを使って監視するイベントを選択する。使用できるこれらのレジスタの数は、ファミリまたはモデルによって異なる（43～45）。
- 18個の性能カウンタMSR。イベントをカウントする。
- 18個のカウンタ設定制御（CCCR）MSR。1個のCCCRが各性能カウンタに対応付けられている。それぞれのCCCRによって、各自に対応付けられた性能カウンタを特定の方法や方式でカウントできるようにセットアップする。
- メモリ内のデバッグストア（DS）セーブ領域。PEBSレコードをストアする。
- IA32_DS_AREA MSR。DSセーブ領域のロケーションを設定する。
- CPUID命令で返されるデバッグストア（DS）機能フラグ（ビット21）。IA-32プロセッサでDS機構が使用できるかどうかを示す。
- IA32_PEBS_ENABLE MSR。PEBS機能をイネーブルにし、リタイアメント時イベントのカウントで使用されるタグ付けをやり直す。
- 事前定義された一連のイベントおよびイベント・メトリック。特定のイベントをカウントする性能カウンタのセットアップを容易にする。

表15-5.は、性能カウンタとそれに対応付けられたCCCR、および各性能カウンタに対してカウントされるイベントを選択するためのESCRを示している。事前定義されたイベント・メトリックおよびイベントは、付録A「性能モニタリング・イベント」の表に記載している。

表 15-5. 性能カウンタ MSR、対応付けられた CCCR と ESCR MSR
（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

カウンタ			CCCR		ESCR		
名前	No.	アドレス	名前	アドレス	名前	No.	アドレス
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_FLAME_ COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_ COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_ COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H

表 15-5. 性能カウンタ MSR、対応付けられた CCCR と ESCR MSR
（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）（続き）

カウンタ			CCCR		ESCR		
名前	No.	アドレス	名前	アドレス	名前	No.	アドレス
MSR_FLAME_ COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR01 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR01 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR11 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR11 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR01 MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR11 MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

注

1. MSR_IQ_ESCR0 と MSR_IQ_ESCR1 は、初期のプロセッサ・ビルド（ファミリ 0FH、モデル 01H ～ 02H）でのみ利用可能である。これらの MSR は、最近のバージョンでは利用できない。

これらの性能モニタリング機能でカウントできるイベントのタイプは、非リタイアメント・イベントおよびリタイアメント時イベントの2つのクラスに分けられる。

- 非リタイアメント・イベント（表 A-1. を参照）は、命令の実行時にいつでも発生するイベントである（例えば、バス・トランザクション、キャッシュ・トランザクションなど）。
- リタイアメント時イベント（表 A-2. を参照）は、命令実行のリタイアメント・ステージでカウントされるイベントである。これにより、イベントのカウントおよ

びマシンステートのキャプチャで、よりきめの細かいグラニュラリティが提供される。リタイアメント時カウント機構には、命令の実行時に特定の性能イベントを検出したμopをタグ付けする機能が備えられている。タグ付けをすれば、イベントをソートできる。例えば、最終的に結果がキャンセルされ、アーキテクチャ・ステートにコミットされなかったときに実行パス上で発生したイベント（予測ミスした分岐の実行など）もソートできるし、実行パス上で発生し、結果としてリタイアメント時にアーキテクチャ・ステートがコミットされたイベントもソートできる。

インテルPentium 4プロセッサおよびインテルXeonプロセッサの性能モニタリング機能は、以下に示す3つの使用モデルをサポートしている。最初の2つのモデルは非リタイアメント・イベントおよびリタイアメント時イベントのカウントに使用でき、3番目のモデルは一部のリタイアメント時イベントのカウントにのみ使用できる。

- **イベントのカウント。**性能カウンタは、1つまたは複数のタイプのイベントをカウントするように設定されている。カウンタがカウントを行っている間、ソフトウェアにより、選択された間隔でカウンタを読み取り、この間隔の間にカウントされたイベントの数を判定する。
- **非プリサイス・イベント・ベース・サンプリング。**性能カウンタは、1つまたは複数のタイプのイベントをカウントし、オーバーフローが発生した時点で割り込みを生成するように設定されている。オーバーフローをトリガさせるため、カウンタには事前に係数値がセットされている。これは、特定の個数のイベントがカウントされたあとにカウンタをオーバーフローさせる値である。カウンタがオーバーフローすると、プロセッサは、性能モニタリング割り込み（PMI）を生成する。次にPMIの割り込みサービスルーチンは、戻り命令ポインタ（RIP）を記録してから係数をリセットし、カウンタを再始動させる。コードの性能は、VTune™パフォーマンス・アナライザなどのツールでRIPの分配を検査することにより、分析できる。
- **プリサイス・イベント・ベース・サンプリング（PEBS）。**このタイプの性能モニタリングは、非プリサイス・イベント・ベース・サンプリングと基本的に同じであるが、カウンタがオーバーフローするたびに、メモリバッファを使用してプロセッサのアーキテクチャ・ステートのレコードを保存する点が異なっている。アーキテクチャ・ステートのレコードにより、性能のチューニングに使用するための追加情報が提供される。プリサイス・イベント・ベース・サンプリングを使用してカウントできるのは、リタイアメント時イベントの一部だけである。

以降の各項では、インテルPentium 4プロセッサおよびインテルXeonプロセッサの性能モニタリングで使用されるMSRとデータ構造について説明し、前述した3つの使用モデルでこれらの機能を使用する方法を説明する。

15.9.1. ESCR MSR

ソフトウェアにより、45 個の ESCR MSR（表 15-5. を参照）を使って、カウントするイベントを選択することができる。通常、各 ESCR は、性能カウンタのペアに対応付けられている（表 15-5. を参照）。また、各性能カウンタには複数の ESCR が対応付けられているため、各種のイベントの中からカウントするイベントを選択可能である。

図 15-6. は、ESCR MSR のレイアウトを示している。それぞれのフラグとフィールドの機能は、以下のとおりである。

USR フラグ、ビット 2

セットされている場合、プロセッサが現行特権レベル (CPL)=1、2、または 3 で動作しているときにイベントがカウントされる。通常、これらの特権レベルは、アプリケーション・コードや保護されていないオペレーティング・システム・コードが使用する。

OS フラグ、ビット 3

セットされている場合、プロセッサが CPL=0 で動作しているときにイベントがカウントされる。通常、この特権レベルは、保護されているオペレーティング・システム・コード用に予約されている（OS フラグと USR フラグが両方セットされている場合は、すべての特権レベルでイベントがカウントされる）。

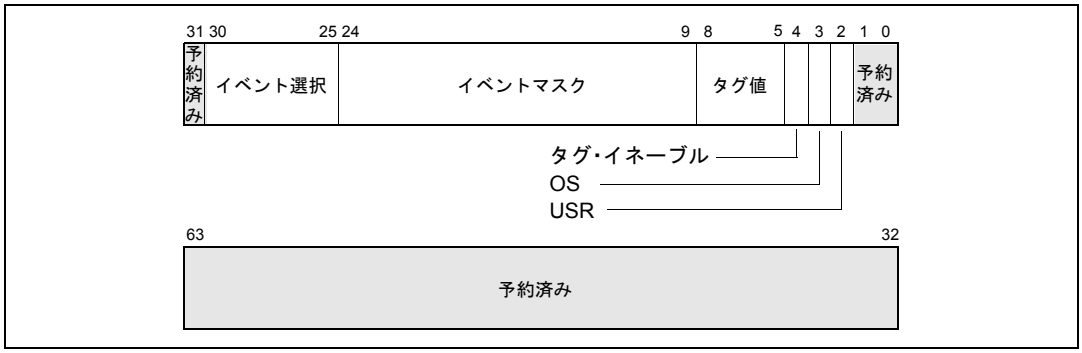


図 15-6. HT テクノロジ非対応のインテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサのイベント選択制御レジスタ（ESCR）

タグ・イネーブル、ビット 4

セットされている場合、リタイアメント時イベントのカウントをアシストする μop のタグ付けがイネーブルになる。クリアされている場合は、タグ付けがディスエーブルになる。詳しくは、15.9.7.「リタイアメント時カウント」を参照のこと。

タグ値フィールド、ビット 5 ～ 8
リタイアメント時イベントのカウントをアシストする μop に対応付けるタグ値を選択する。

イベント・マスク・フィールド、ビット 9 ～ 24
イベント選択フィールドで選択したイベントクラスから、カウントするイベントを選択する。

イベント選択フィールド、ビット 25 ～ 30
カウントするイベントのクラスを選択する。このクラス内のカウント対象のイベントは、イベント・マスク・フィールドで選択される。

ESCR をセットアップするときは、イベント選択フィールドを使用して、リタイアされた分岐などの、カウントするイベントのクラスを選択する。カウントするクラス内のイベントを 1 つまたは複数選択するには、イベント・マスク・フィールドを使用する。例えば、リタイアされた分岐をカウントするときは、予測されて実施されなかった分岐、誤って予測されて実施されなかった分岐、予測されて実施された分岐、誤って予測されて実施された分岐という 4 つの異なるイベントをカウントできる。OS フラグおよび USR フラグを使用すると、オペレーティング・システム・コードやアプリケーション・コードの実行中に発生するイベントに対して、カウントをイネーブルにすることができる。OS フラグと USR フラグのいずれもセットされていない場合は、イベントはカウントされない。

リセットにより、ESCR はすべて 0 に初期化される。ESCR のフラグとフィールドを設定するためには、WRMSR 命令を使って ESCR に書き込む。表 15-5. は、ESCR MSR のアドレスを示している。

注記

ESCR MSR に書き込みを行っても、カウントするイベントが選択されるだけであり、対応付けられた性能カウンタによるカウントがイネーブルになるわけではない。選択された性能カウンタの CCCR も設定する必要がある。CCCR を設定するには、ESCR を選択し、カウンタをイネーブルにしなければならない。

15.9.2. 性能カウンタ

ESCR で選択されたイベントをフィルタリングしてカウントするには、性能カウンタとカウンタ設定制御レジスタ（CCCR）が使用される。インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサは 18 個の性能カウンタを備えており、これらは 9 つのペアにまとめられている。性能カウンタの 1 つのペアは、イベントと ESCR の特定のサブセットに対応付けられている（表 15-5. を参照）。カウンタのペアは、以下の 4 つのグループに分類される。

- BPU グループ。以下の 2 つの性能カウンタのペアが含まれる。
  - MSR_BPU_COUNTER0 および MSR_BPU_COUNTER1
  - MSR_BPU_COUNTER2 および MSR_BPU_COUNTER3
- MS グループ。以下の 2 つの性能カウンタのペアが含まれる。
  - MSR_MS_COUNTER0 および MSR_MS_COUNTER1
  - MSR_MS_COUNTER2 および MSR_MS_COUNTER3
- FLAME グループ。以下の 2 つの性能カウンタのペアが含まれる。
  - MSR_FLAME_COUNTER0 および MSR_FLAME_COUNTER1
  - MSR_FLAME_COUNTER2 および MSR_FLAME_COUNTER3
- IQ グループ。以下の 3 つの性能カウンタのペアが含まれる。
  - MSR_IQ_COUNTER0 および MSR_IQ_COUNTER1
  - MSR_IQ_COUNTER2 および MSR_IQ_COUNTER3
  - MSR_IQ_COUNTER4 および MSR_IQ_COUNTER5

IQ グループの MSR_IQ_COUNTER4 カウンタは、PEBS をサポートする。

各グループの代替カウンタはカスケードできる。すなわち、あるペアの先頭のカウンタによって、2 番目のペアの先頭のカウンタを開始できる（その逆も可能）。これと同様のカスケードは、各ペアの 2 番目のカウンタでも可能である。例えば、BPU グループのカウンタでは、MSR_BPU_COUNTER0 が MSR_BPU_COUNTER2 を開始できるし（その逆も可能）、MSR_BPU_COUNTER1 が MSR_BPU_COUNTER3 をも開始できる（その逆も可能）。詳しくは、15.9.6.6.「カウンタのカスケード」を参照のこと。性能カウンタの CCCR レジスタのカスケード・フラグで、カウンタのカスケードをイネーブルにする。

性能カウンタは、それぞれ 40 ビット幅である（図 15-7. を参照）。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは RDPMC 命令が拡張されており、全カウンタ幅（40 ビット）、またはカウンタの下位 32 ビットのいずれかで読み取ることができる。下位 32 ビットの読み取りは、全カウンタ幅の読み取りよりも高速である。したがって、カウントが 32 ビットに十分収まるサイズの場合は、32 ビット読み取りが適している。

任意の特権レベルで仮想 8086 モードで実行中のプロセッサやプロシージャでは、RDPMC 命令を使用して、これらのカウンタを読み取ることができる。制御レジスタ CR4 の PCE フラグ（ビット 8）を使って、特権レベル 0 で実行中のプログラムやプロシージャだけしかこの命令を使用しないように制限できる。

図 15-7. 性能カウンタ（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）

RDPMC 命令は、他の命令によってシリアル化されたりオーダリングされたりしない。したがって、RDPMC 命令は必ずしもそれより前のすべての命令の実行が終わるまで待ってからカウンタを読むわけではない。同様に、RDPMC 命令の操作が行われる前にそれより後の命令が実行を開始することもできる。

RDMSR 命令および WRMSR 命令を使用して直接性能カウンタを操作できるのは、特権レベル 0 で実行中のオペレーティング・システムだけである。安全保護されたオペレーティング・システムでは、システムの初期化の際に PCE フラグをクリアして、ユーザが性能モニタリング・カウンタを直接操作できないようにする。ただし、RDPMC 命令をエミュレートするための、ユーザが使用可能なプログラミング・インターフェイスを提供しておく。

性能カウンタは、カウントを開始する前に（すなわち、カウンタがイネーブルになる前に）、カウンタをプリセットしてから使用しなければならない場合がある。これを行うには、WRMSR 命令を使用してカウンタに書き込みを行えばよい。オーバーフローに達するまでのカウント数をカウンタにセットするには、2 の補数を用いた負の整数をカウンタに入力する。このようにすることで、カウンタは、プリセットされた値から -1 までカウントし、それからオーバーフローするようになる。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、WRMSR 命令を使って性能カウンタに書き込みを行うと、カウンタの全 40 ビットが書き込まれる。

15.9.3. CCCR MSR

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの 18 個の性能カウンタには、それぞれ 1 つの CCCR MSR が対応付けられている（表 15-5. を参照）。CCCR は、割り込みの生成だけではなく、イベントのフィルタリングやカウントも制御する。図 15-8. は、CCCR MSR のレイアウトを示している。それぞれのフラグとフィールドの機能は、以下のとおりである。

イネーブル・フラグ、ビット 12

セットされている場合、カウントがイネーブルになる。クリアされている場合、カウンタがディスエーブルになる。このフラグは、リセット時にクリアされる。

ESCR 選択フィールド、ビット 13 ～ 15

CCCR に対応付けられたカウンタでカウントするイベントを選択するのに使用する ESCR を識別する。

比較フラグ、ビット 18

セットされている場合、イベントカウントのフィルタリングがイネーブルになる。クリアされている場合、フィルタリングがディスエーブルになる。フィルタリングの方法は、しきい値、補数、エッジの各フラグで選択できる。

補数フラグ、ビット 19

着信イベントカウントとしきい値を比較する方法を選択する。セットされている場合、イベントカウントがしきい値以下のときに、単一のカウントが性能カウンタに渡される。クリアされている場合は、イベントカウントがしきい値を超えたときに、単一のカウントが性能カウンタに渡される（15.9.6.2.「イベントのフィルタリング」を参照）。補数フラグは比較フラグがセットされている場合にだけ有効である。

しきい値フィールド、ビット 20 ～ 23

比較に使用するしきい値を選択する。プロセッサは、比較フラグがセットされている場合にだけ、このフィールドを検査する。またプロセッサは、補数フラグの設定値を使用して、実行するしきい値比較のタイプを決定する。このフィールドに入力できる値の有効範囲は、カウントされるイベントのタイプによって変わる（15.9.6.2.「イベントのフィルタリング」を参照）。

エッジフラグ、ビット 24

セットされている場合、イベントカウントをフィルタリングしたときのしきい値比較出力に対して、立ち上がりエッジ（偽から真）によるエッジ検出がイネーブルになる。クリアされている場合、立ち上がりエッジによる検出はディスエーブルになる。このフラグは、比較フラグがセットされている場合にだけ有効になる。

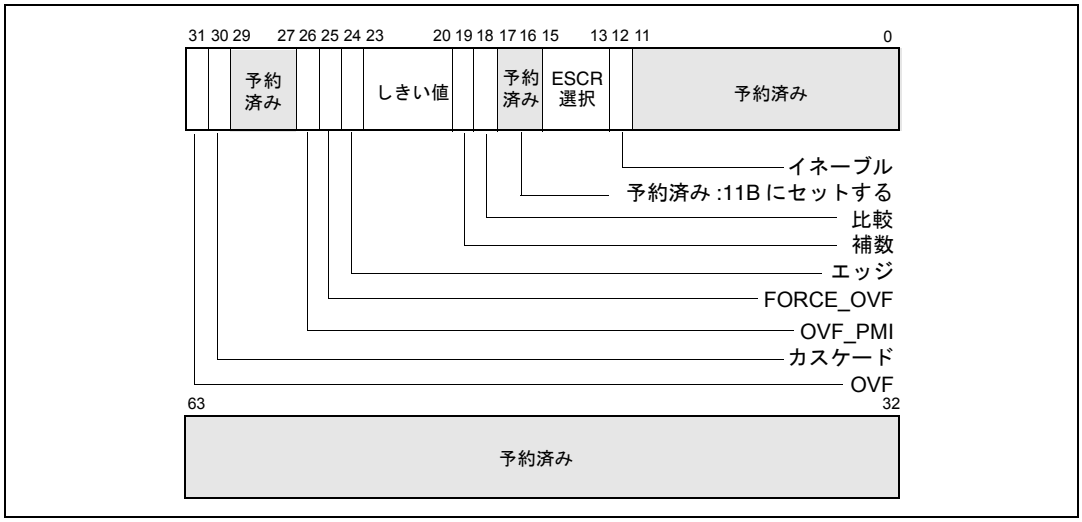


図 15-8. カウンタ設定制御レジスタ（CCCR）

FORCE_OVF フラグ、ビット 25

セットされている場合、カウンタがインクリメントされるたびにカウンタのオーバーフローが発生する。クリアされている場合は、カウンタが実際にオーバーフローしたときにだけオーバーフローが発生する。

OVF_PMI フラグ、ビット 26

セットされている場合、カウンタのオーバーフローが発生したときに性能モニタリング割り込み（PMI）が生成される。クリアされている場合は、PMI の生成はディスエーブルになる。PMI は、カウンタがオーバーフローしてから、その次のイベントカウントで生成される。

カスケード・フラグ、ビット 30

セットされている場合、同じカウンタグループ内にある一方のカウンタペアの代替カウンタがオーバーフローしたときに、もう一方のカウンタペアのカウンタ上でカウントがイネーブルになる（詳しくは、15.9.2.「性能カウンタ」を参照）。クリアすると、カウンタのカスケードがディスエーブルになる。

OVF フラグ、ビット 31

セットされている場合、カウンタがオーバーフローしたことを示す。このフラグは、ソフトウェアが明示的にクリアする必要のあるスティッキー・フラグである。

リセットにより、CCCR はすべて 0 に初期化される。

イネーブルにされている性能カウンタが実際にカウントするイベントは、ESCR レジスタおよびCCCR レジスタ内の以下のフラグとフィールドによって、以下の限定順序で選択され、フィルタリングされる。

1. ESCR 内のイベント選択フィールドとイベント・マスク・フィールドにより、カウントするイベントのクラス、およびクラス内の 1 つまたは複数のイベントタイプがそれぞれ選択される。
2. ESCR 内の OS フラグおよび USR フラグにより、イベントをカウントする特権レベルが選択される。
3. CCCR の ESCR 選択フィールドにより ESCR が選択される。それぞれのカウンタには複数の ESCR が対応付けられているため、カウントされる可能性のあるイベントのクラスを選択するには、ESCR を 1 つ選択する必要がある。
4. CCCR の比較フラグ、補数フラグ、しきい値フィールドにより、イベントカウントの限定に使用できる任意設定のしきい値が選択される。
5. CCCR のエッジフラグを使用すると、立ち上がりエッジでのみイベントをカウントできる。

上記の限定順序では、ある「ステージ」についてフィルタリングされた出力が、次のステージの入力となることが示されている。例えば、特権レベルフラグを使用してフィルタリングされたイベントに対して、比較フラグ、補数フラグ、しきい値フィールドを使ってさらに制限を加え、このしきい値基準に一致したイベントに対してエッジ検出でさらに制限を加えられる。

CCCR のそれぞれのフラグとフィールドの使用方法については、15.9.6.「非リタイアメント・イベント用性能カウンタのプログラミング」でさらに詳しく説明している。

15.9.4. デバッグストア（DS）機構

デバッグストア（DS）機構は、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで導入された。この機構を使用すると、各種の情報をメモリ常駐バッファ内に収集し、それをプログラムのデバッグやチューニングに使用できる。インテルPentium 4プロセッサおよびインテルXeonプロセッサの場合、DS機構は、分岐レコードおよびプリサイス・イベント・ベース・サンプリング（PEBS）レコードの 2 つのタイプの情報を収集するのに使用される（これらの機能の説明については、15.5.7.「分岐トレースストア（BTS: Branch Trace Store）」、15.9.8.「プリサイス・イベント・ベース・サンプリング（PEBS: Precise Event-based Sampling）」を参照）。プロセッサでDS機構が使用できるかどうかは、CPUID命令で返されるDS機能フラグ（ビット21）で示される。

DS機構で収集されたレコードは、DSセーブ領域に保存される（15.9.5.「DSセーブ領域」を参照）。

15.9.5. DS セーブ領域

デバッグストア（DS）セーブ領域はソフトウェアで指定するメモリ領域であり、以下の2つのタイプの情報を収集するのに使用される。

- **分岐レコード。**MSR_DEBUGCTLA MSR の BTS フラグがセットされると、実施された分岐、割り込み、または例外が検出されるたびに、DS セーブ領域の BTS バッファに分岐レコードがストアされる。
- **PEBS レコード。**性能カウンタが PEBS 用に設定されると、カウンタのオーバーフローが発生するたびに、DS セーブ領域の PEBS バッファに PEBS レコードがストアされる。このレコードには、カウンタをオーバーフローさせたイベントが発生した時点でのプロセッサのアーキテクチャ・ステート（8 個の汎用レジスタ、EIP レジスタ、EFLAGS レジスタのステート）が格納されている。ステート情報がログに記録されている場合、カウンタはあらかじめ選択された値に自動的にリセットされ、イベントのカウントが再度開始される。この機能は、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの性能イベントの一部分に対してのみ使用できる。

注記

DS セーブ領域および DS 記録機構は、SMM では使用できない。この機能は、SMM モードへ移行する時点でディスエーブルになる。同様に、DS 記録は、マシンチェック例外の生成時にディスエーブルになり、プロセッサの RESET および INIT によってクリアされる。DS 記録は、実アドレスモードで使用できる。

BTS 機能と PEBS 機能は、どの IA-32 プロセッサでも使用できるとは限らない。これらの機能が使用できるかどうかは、IA32_MISC_ENABLE MSR の BTS_UNAVAILABLE フラグと PEBS_UNAVAILABLE フラグでそれぞれ示される（表 B-1. を参照）。

DS セーブ領域は、バッファ管理領域、分岐トレースストア（BTS）バッファ、PEBS バッファの3つの部分に分けられる（図 15-9. を参照）。バッファ管理領域は、BTS バッファと PEBS バッファのロケーションおよびサイズを規定するのに使用される。プロセッサは、このバッファ管理領域を使用して、それぞれのバッファ内の分岐レコードや PEBS レコードを追跡し、性能カウンタのリセット値を記録する。DS バッファ管理領域の先頭バイトのリニアアドレスは、IA32_DS_AREA MSR で指定される。

バッファ管理領域の各フィールドは、以下のとおりである。

BTS バッファベース

BTS バッファの先頭バイトのリニアアドレス。このアドレスは、ダブルワードの自然境界を指していなければならない。

BTS インデックス

書き込み先となる次の BTS レコードの先頭バイトのリニアアドレス。最初の時点では、このアドレスは BTS バッファ・ベース・フィールドのアドレスと同じでなければならない。

BTS 絶対最大値

BTS バッファの末尾の次のバイトのリニアアドレス。このアドレスは、BTS レコードサイズ（12 バイト）の倍数に 1 を加えた値でなければならない。

BTS 割り込みしきい値

割り込みが生成される BTS レコードのリニアアドレス。このアドレスは、BTS バッファベースからのオフセットを指し、BTS レコードサイズの倍数でなければならない。また、これは、プロセッサが BTS 絶対最大値レコードを書き込む前に保留中の割り込みが処理されるようにするため、BTS 絶対最大値アドレスよりもレコード数個分だけ小さくなければならない。

PEBS バッファベース

PEBS バッファの先頭バイトのリニアアドレス。このアドレスは、ダブルワードの自然境界を指していなければならない。

PEBS インデックス

書き込み先となる次の PEBS レコードの先頭バイトのリニアアドレス。最初の時点では、このアドレスは PEBS バッファ・ベース・フィールドのアドレスと同じでなければならない。

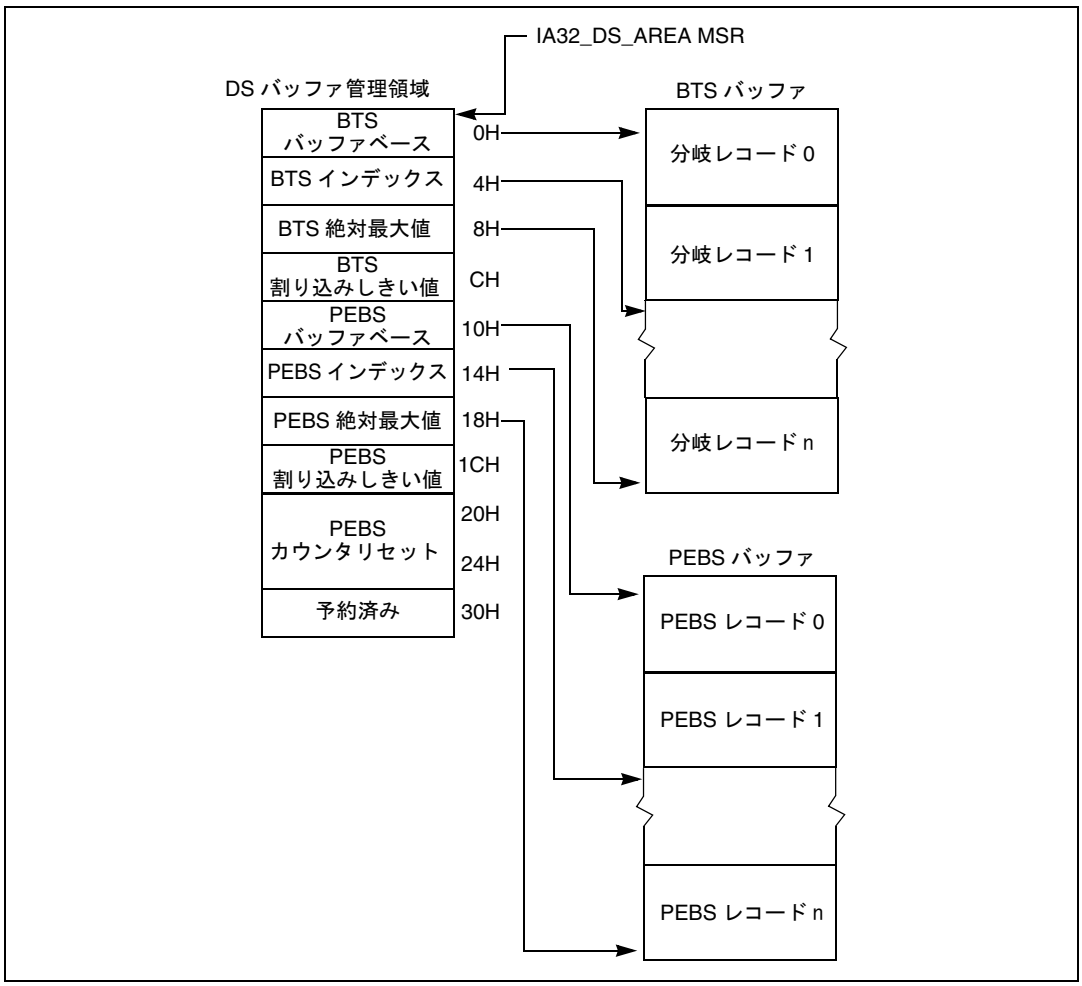


図 15-9. DS セーブ領域

PEBS 絶対最大値

PEBS バッファの末尾の次のバイトのリニアアドレス。このアドレスは、PEBS レコードサイズ（40 バイト）の倍数に 1 を加えた値でなければならない。

PEBS 割り込みしきい値

割り込みが生成される PEBS レコードのリニアアドレス。このアドレスは、PEBS レコードサイズからのオフセットを指し、PEBS バッファベースの倍数でなければならない。また、これは、プロセッサが PEBS 絶対最大値レコードを書き込む前に保留中の割り込みが処理されるようにするため、PEBS 絶対最大値アドレスよりもレコード数個分だけ小さくなければならない。

PEBS カウンタリセット値

ステート情報が収集された結果、カウンタがオーバーフローした後に、カウンタがリセットされる 40 ビット値。この値を使用すると、プリセットされた個数のイベントがカウントされてから、ステート情報が収集されるように設定できる。

図 15-10. は、BTS バッファ内の 12 バイトの分岐レコードの構造を示している。各レコードのフィールドは、以下のとおりである。

最後の分岐元 分岐、割り込み、または例外を処理した命令のリニアアドレス。

最後の分岐先 分岐ターゲット、あるいは割り込みや例外のサービスルーチン内の先頭命令のリニアアドレス。

分岐予測 このフィールドのビット 4 は、実施された分岐が予測されていたか(セット)、または予測されていなかったか（クリア）を示す。

図 15-10. 分岐トレースレコードのフォーマット

図 15-11. は、40 バイトの PEBS レコードの構造を示している。名目上、レジスタの値は、イベントを引き起こした命令の最初のレジスタ値である。ただし、レジスタは、部分的に変更されたステートで記録される場合がある。リニア IP フィールドは、現在のコード・セグメントへのオフセットからリニアアドレスに変換された EIP レジスタの値を示している。

15.9.6. 非リタイアメント・イベント用性能カウンタのプログラミング

性能カウンタをプログラムしてイベントのカウントを開始するには、ソフトウェアで以下の操作を実行する必要がある。

1. カウントするイベントを選択する。
2. 各イベントについて、表 A-1. の「ESCR 限定」の行の値を使用して、そのイベントをサポートする ESCR を選択する。
3. 表 A-1. の「CCCR 選択」の値および ESCR 名と、表 15-5. の「ESCR 名」および「ESCR 番号」の項目の値を一致させて、CCCR と性能カウンタを選択する。

4. カウント対象の特定のイベント用の ESCR、およびイベントをカウントする特権レベルをセットアップする。
5. 選んだ ESCR を選択し、希望のイベントフィルタを選択して、イベントのカウントに使用する性能カウンタの CCCR をセットアップする。
6. イベントカウントのカスケードが実行されるように CCCR をセットアップする（任意設定）。これにより、選択したカウンタがオーバーフローすると、代替カウンタがカウントを開始するようになる。
7. カウンタがオーバーフローしたときに性能モニタリング割り込み（PMI）が生成されるように CCCR をセットアップする（任意設定）。（PMI の生成がイネーブルの場合は、割り込みがプロセッサへ伝達されるようにローカルAPICをセットアップする必要がある。また、割り込みハンドラを使用可能にしておく必要がある。）
8. カウンタをイネーブルにし、カウントを開始する。

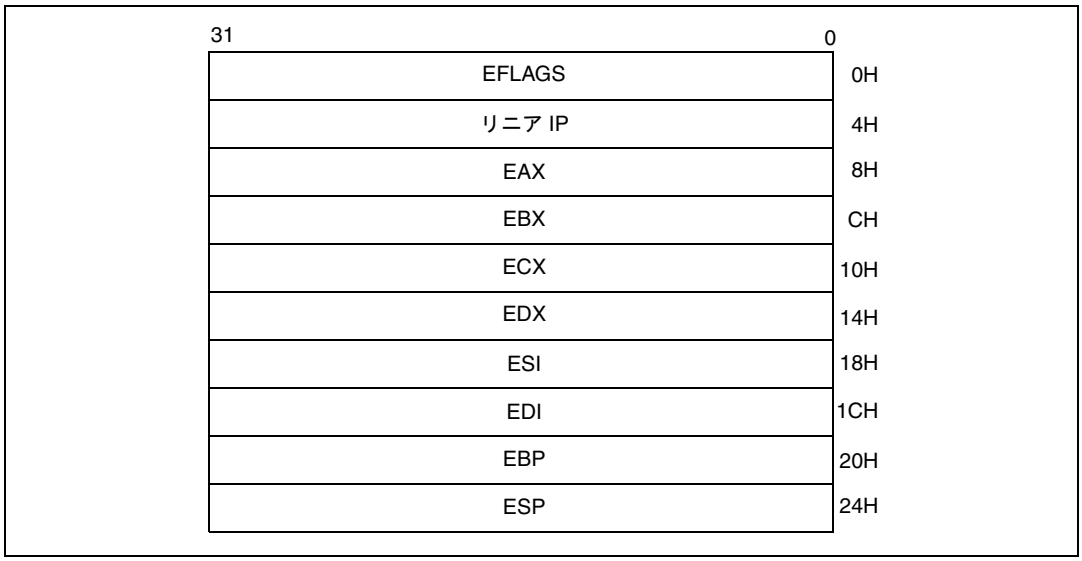


図 15-11. PEBS レコードのフォーマット

15.9.6.1. カウントするイベントの選択

表 A-1. には、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの一連の非リタイアメント・イベントが記載されている。表 A-1. に記載されている各イベントには、特定のセットアップ情報が用意されている。図 15-12. では、表 A-1. の非リタイアメント・イベントの 1 つを例として挙げている。

表 A-1. および表 A-2. では、イベント名の列にはイベントの名前を示し、イベント・パラメータの列にはイベントおよびその他の情報を定義する各種のパラメータを示し

ている。パラメータ値の列にはイベント固有のパラメータを、説明の列には補足説明を示している。イベント・パラメータの列の各項目は、以下のとおりである。

ESCR 限定

イベントのプログラムに使用可能な ESCR の一覧である。一般に、イベントをカウントするには、ESCR が 1 つだけあればよい。

図 15-12. イベントの例

イベント名	イベント・パラメータ	パラメータ値	説明
Branch_retired			分岐のリタイアメントをカウントする。実施された分岐、実施されなかった分岐、予測された分岐、および誤って予測された分岐の任意の組み合わせを選択するには、1 つまたは複数のマスクビットを指定する。
	ESCR 限定	MSR_CRU_ESCR2 MSR_CRU_ESCR3	ESCR MSR のアドレスについては、OSWG の表 15-3. を参照のこと。
	ESCR ごとの カウンタ番号	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	各 ESCR に対応付けられたカウンタ番号が示されている。性能カウンタおよび対応する CCCR は、表 15-3. から知ることができる。
	ESCR イベント選択	06H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] 予測されて実施されなかった分岐 誤って予測されて実施されなかった分岐 予測されて実施された分岐 誤って予測されて実施された分岐
	OCCR 選択	05H	CCCR[15:13]
	イベント固有の 注意事項		P6: EMON_BR_INST_RETIRED
	PEBS サポート可能	なし	
	タグ付け用に別の MSR が必要	なし	

ESCR ごとのカウンタ番号

それぞれの ESCR に対応付けられている性能カウンタの一覧である。表 15-5. に、カウンタ番号ごとに、カウンタと CCCR の名前を示してある。一般に、イベントをカウントするには、カウンタが 1 つだけあればよい。

ESCR イベント選択

イベントを選択するために ESCR イベント選択フィールドに入れる値を示す。

ESCR イベントマスク

カウント対象のサブイベントを選択するために ESCR イベント・マスク・フィールドに入れる値を示す。パラメータ値の列には、0 から始まるビッ

ト位置相対オフセットを持つ表記ビットを定義している（相対オフセット 0 の絶対ビット位置は、ESCR のビット 9）。表記されていないビットはすべて予約されており、0 にセットする必要がある。

CCCR 選択　CCCR（イベント定義用の ESCR を選択するカウンタに対応付けられている）の ESCR 選択フィールドに入れる値を示す（この値は ESCR のアドレスではなく、表 15-5. の No. 列の ESCR の番号である）。

イベント固有の注意事項
イベントについての補足情報を示す。P6 ファミリ・プロセッサで定義されたのと同じまたは同等のイベント名などである。

PEBS サポート可能
イベントに対して PEBS がサポートされるかどうかを示す（この情報は、表 A-2. に記載したリタイアメント時イベントに対してのみ示される）。

タグ付け用に別の MSR が必要
イベントをカウントするのに、さらに MSR をプログラムする必要があるかどうかを示す（この情報は、表 A-2. に記載したリタイアメント時イベントに対してのみ示される）。

注記

付録 A「性能モニタリング・イベント」に記載している性能モニタリング・イベントは、性能をチューニングする際のガイドとして使用されることを想定している。報告されるカウンタ値が完全に正確であることは保証されていないため、チューニングする際の参考程度として使用するべきである。該当する個所には、既知の矛盾点を記載している。

以下の手順では、性能カウンタをセットアップして、基本的なカウントを行う方法を説明する。すなわち、指定されたイベントを無期限にカウントし、最大カウントに達するたびに循環が行われるようにする。この手順は、以降の 4 つの項に渡って説明する。

表 A-1. の情報を使用して、以下のようにして、カウントするイベントを選択する。

1. カウントするイベントを選択する。
2. ESCR 限定フィールドから、カウント対象のイベントを選択するのに使用する ESCR を選ぶ。
3. ESCR ごとのカウンタ番号フィールドから、イベントをカウントするのに使用するカウンタの番号を選ぶ。
4. 表 15-5. から、カウンタおよびカウンタに対応付けられている CCCR の名前と、カウンタ、CCCR、ESCR の MSR アドレスを確認する。

5. WRMSR 命令を使用して、表 A-1. の ESCR イベント選択と ESCR イベントマスクの値を ESCR の該当するフィールドに書き込む。同時に、ESCR の USR フラグと OS フラグを必要に応じてセットまたはクリアする。
6. WRMSR 命令を使用して、表 A-1. の OCCR 選択の値を、CCCR の該当するフィールドに書き込む。

注記

通常、CCCR のフィールドおよびフラグはすべて、1 回の WRMSR 命令で書き込まれる。ただし、この手順では、CCCR の各種のフィールドおよびフラグの使い方をより具体的に説明するため、数回に渡って WRMSR 命令による書き込みを行っている。

このセットアップ手順は、次の 15.9.6.2.「イベントのフィルタリング」に続く。

15.9.6.2. イベントのフィルタリング

各カウンタは、イベントをカウントするプロセッサ・ハードウェアから、最大 4 本の入力ラインを受け入れる。カウンタは、これらの入力を 2 進数の入力値（入力 0 は値 1、入力 1 は値 2、入力 2 は値 4、および入力 3 は値 8）として扱う。カウンタがイネーブルになると、この 2 進入力値を各クロックサイクルのカウンタ値に加算する。各クロックサイクルでカウンタに追加される値の範囲は、0（イベントなし）～15 である。

多くのイベントでは、入力ライン 0 しか有効にならない。そのため、カウンタは、入力 0 がアサートされているクロックサイクルだけをカウントする。ただし、イベントによっては、2 本以上の入力ラインが使用されることもある。この場合、カウンタしきい値設定を使用すると、イベントをフィルタリングできる。比較、補数、しきい値、エッジの各フィールドを指定することで、カウンタのインクリメントに対する入力値によるフィルタリングを制御できる。

比較フラグをセットすると、入力値がしきい値に対して「より大きい」のか、それとも「より小さいか等しい」のかを比較できる。補数フラグでは、フラグをセットすると「より小さいか等しい」が選択され、フラグをクリアすると「より大きい」が選択される。しきい値フィールドでは、0～15 のしきい値を選択できる。例えば、補数フラグをクリアして、しきい値フィールドを 6 にセットした場合、カウンタの 4 つの入力に対して 7 以上の値を入力すると、カウンタが 1 だけインクリメントされる。7 より小さな値を入力すると、カウンタは 0 だけインクリメントされる（インクリメントなし）。反対に、補数フラグをセットすると、0～6 の値が入力されるとカウンタがインクリメントされ、7～15 の値が入力されてもカウンタはインクリメントされない。しきい値の条件が満たされている場合、カウンタへの入力は常に 1 であって、しきい値フィルタに与えられている入力値ではない。

エッジフラグを使って、しきい値比較の実行時にカウンタ入力に対してさらにフィルタリングすることができる。エッジフラグは、比較フラグをセットしたときにだけ有効になる。エッジフラグをセットすると、しきい値フィルタを通した結果得られる出力（値 0 または 1）が、エッジフィルタの入力として使用される。クロックサイクルごとにエッジフィルタは最新の入力値を検査し、「立ち上がりエッジ」イベント（すなわち、偽から真への遷移）を検出したときにだけ、カウントをカウンタへ送信する。図 15-13. は、立ち上がりエッジのフィルタリングを説明したものである。

以下の手順では、しきい値フィルタとエッジフィルタを使ってイベントのフィルタリングを行うように CCCR を設定する方法を説明する。この手順は、15.9.6.1.「カウントするイベントの選択」で説明したセットアップ手順の続きである。

7. （任意選択）しきい値によるフィルタリングを行うようにカウンタをセットアップするには、WRMSR 命令を使用して、CCCR の比較フラグと補数フラグ、およびしきい値フィールドに値を書き込む。
   - 比較フラグをセットする。
   - 「より小さいか等しい」比較を行う場合は補数フラグをセットし、「より大きい」比較を行う場合は補数フラグをクリアする。
   - しきい値フィールドに 0 ～ 15 の値を入力する。
8. （任意選択）CCCR のエッジフラグをセットして、立ち上がりエッジによるフィルタリングを選択する。

このセットアップ手順は、次の 15.9.6.3.「イベントのカウントの開始」に続く。

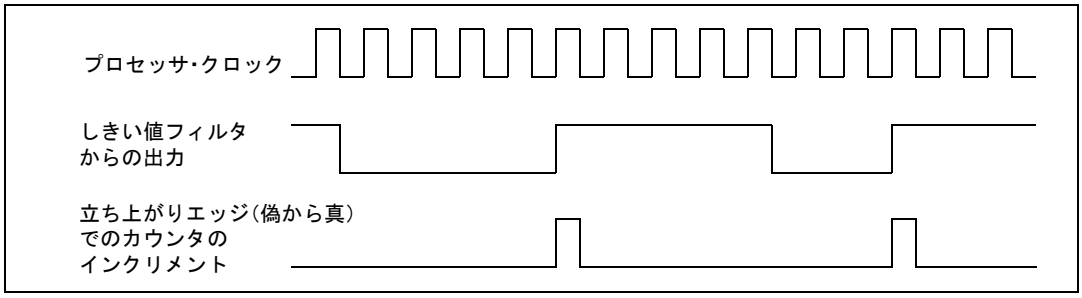


図 15-13. エッジ・フィルタリングの効果

15.9.6.3. イベントのカウントの開始

性能カウンタでイベントのカウントを開始するには2つの方法がある。一般的なのは、カウンタのCCCRのイネーブル・フラグをセットする方法である。イネーブル・フラグをセットする命令を実行すると、イベントのカウントが開始する。カウントは、停止させられるまで続行する（15.9.6.5.「イベントのカウントの停止」を参照）。

以下の手順は、イベントのカウントを開始する方法を示している。この手順は、15.9.6.2.「イベントのフィルタリング」で説明したセットアップ手順の続きである。

9. イベントのカウントを開始するには、WRMSR 命令を使用して、性能カウンタの CCCR のイネーブル・フラグをセットする。

このセットアップ手順は、次の15.9.6.4.「性能カウンタのカウンタの読み取り」に続く。

カウンタを開始する2番目の方法は、カスケード機能を使用するものである。この場合、あるカウンタがオーバーフローすると、自動的にその代替カウンタが開始される（15.9.6.6.「カウンタのカスケード」を参照）。

15.9.6.4. 性能カウンタのカウンタの読み取り

インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサの性能カウンタは、RDPMC命令またはRDMSR命令で読み取ることができる。RDPMC命令の拡張機能（高速読み取りも含む）については、15.9.2.「性能カウンタ」で説明している。これらの命令を使用すると、カウント中であっても、カウントが停止しているときであっても、性能カウンタを読み取ることができる。

以下の手順は、イベントカウンタを読み取る方法を示している。この手順は、15.9.6.3.「イベントのカウントの開始」で説明したセットアップ手順の続きである。

10. 性能カウンタの現在のイベントカウントを読み取るには、表 15-5. に示されているカウンタ番号をオペランドに指定して、RDPMC 命令を実行する。

このセットアップ手順は、次の15.9.6.5.「イベントのカウントの停止」に続く。

15.9.6.5. イベントのカウントの停止

性能カウンタが開始すると（イネーブルになると）、無期限にカウントを続ける。カウンタがオーバーフローすると（カウントが最大カウント数を超えると）、循環してカウントを続行する。カウンタが循環すると、OVFフラグがセットされ、カウンタがオーバーフローしたことが示される。OVFフラグはスティッキー・フラグであり、OVFビットが最後にクリアされてから1回以上カウンタがオーバーフローしたことを示す。

カウントを停止するには、そのカウンタのCCCRのイネーブル・フラグをクリアする必要がある。

以下の手順は、イベントカウントを停止する方法を示している。この手順は、15.9.6.4.「性能カウンタのカウンタの読み取り」で説明したセットアップ手順の続きである。

11. イベントのカウントを停止するには、WRMSR 命令を実行して、性能カウンタの CCCR のイネーブル・フラグをクリアする。

カスケード・カウンタ（代替カウンタがオーバーフローしたときに開始されたカウンタ）を停止するには、カスケード・カウンタのCCCR MSRのカスケード・フラグまたは代替カウンタのCCCR MSRのOVFフラグをクリアする。

15.9.6.6. カウンタのカスケード

15.9.2.「性能カウンタ」で説明したように、18個の性能カウンタがペアで実装されている。9組のカウンタとそれに対応付けられたCCCRは、さらにBPU、MS、FLAME、IQの4ブロックに編成される（表15-5.を参照）。BPU、MS、FLAMEブロックは、それぞれ2組のカウンタで構成される。IQブロックは、3組のカウンタ（12～17）とそれに対応付けられたCCCR（MSR_IQ_CCCR0～MSR_IQ_CCCR5）で構成される。

最初の8組のカウンタ（0～15）は、ESCRを使用して性能モニタリング・イベントを検出するようにプログラミングできる。4つのブロックのそれぞれに含まれるESCRのペアを使用して、さまざまなタイプのイベントをカウントできる。CCCR MSR内のカスケード・フラグを使用して、第1のカウンタを同じブロック内の別の組の第2のカウンタに対してカスケードし、ネストしたイベント・モニタリングを実行できる（フラグの位置は、図15-8.を参照）。

カウンタ0とカウンタ1は、BPUブロック内の最初のペアになる。MSR_MO B_ESCR0を使用してカウンタ0またはカウンタ1をプログラムし、イベントを検出できる。カウンタ0とカウンタ2をカスケードする順番に制限はない。カウンタ1とカウンタ3についても同様である。同じブロック内の4個のカウンタをカスケードして、2組の独立したイベントを検出することも可能である。ここで説明したBPUブロック内のペアの作成方法は、それ以降のブロックにも適用される。IQ PUBには2個の追加カウンタがあるため、カウンタ16とカウンタ17を使用する場合、カスケードの動作は多少異なる。IQブロック内で、カウンタ16は、カウンタ14以外のカウンタから（カウンタ12から）カスケードはできない。カウンタ14は、カウンタ16からCCCRカスケード・ビット機構を使用してカスケードはできない。同様の制限は、カウンタ17にも適用される。

カウント条件の例

イベントAを200回カウントするようにカウンタXを設定し、次にイベントBを400回カウントするようにカウンタYを設定する条件を考える。各カウンタは、特定のイベントをカウントし、オーバーフロー時に次のカウンタを起動するように設定される。上記の例で、カウンタXは–200のカウントに設定され、カウンタYは–400のカウントに設定される。この設定により、カウンタXは200回目のカウントでオーバーフローし、カウンタYは400回目のカウントでオーバーフローする。

この条件を続けて、（15.9.6.1.「カウントするイベントの選択」」からの性能カウンタの基本設定手順にしたがって）イベントAを無制限にカウントし、オーバーフロー時にラップアラウンドするように、カウンタXを設定する。カウンタYの設定は、対応付けられたCCCR MSR内のカスケード・フラグを1にセットし、有効フラグを0にクリアする。

ネストしたカウンティングを開始するには、カウンタXの有効ビットをセットする。カウンタXは、一度イネーブルになると、オーバーフローするまでカウントを続ける。この時点で、カウンタYが自動的にイネーブルになり、カウンティングを開始する。したがって、カウンタXは、イベントAが200回発生した後にオーバーフローする。この時点でカウンタYが起動し、イベントBを400回カウントした後にオーバーフローする。性能カウンタをカスケードする場合、15.9.6.9.「オーバーフロー時の割り込みの生成」で説明するように、通常はカウンタYはオーバーフロー時に割り込みを生成するように設定される。

カウンタのカスケード機構は、1種類のイベントのカウントにも使用できる。第1のカウンタでカウンティングを開始し、そのカウンタがオーバーフローした後、第2のカウンタでカウンティングを続行する。この方法は、2つのカウンタの内容を合計できるため、2倍のイベントカウントを記録できる。

15.9.6.7. 拡張カスケード

拡張カスケードは、Intel NetBurst® マイクロアーキテクチャ内のモデル固有の機能である。この機能は、ファミリ・エンコーディングが15、モデル・エンコーディングが2またはそれ以上のインテル® Pentium® 4プロセッサおよびインテル® Xeon™ プロセッサ・ファミリで利用可能である。この機能は、IQブロックに対応付けられたCCCR内のビット11を使用する。以下の表を参照のこと。

表 15-6. CCR の名称とビット位置

CCCR の名称 : ビット位置	ビット名	説明
MSR_IQ_CCCR1|2:11	予約済み	
MSR_IQ_CCCR0:11	CASCNT4INTO0	カウンタ 4 をカウンタ 0 にカスケードできる
MSR_IQ_CCCR3:11	CASCNT5INTO3	カウンタ 5 をカウンタ 3 にカスケードできる
MSR_IQ_CCCR4:11	CASCNT5INTO4	カウンタ 5 をカウンタ 4 にカスケードできる
MSR_IQ_CCCR5:11	CASCNT4INTO5	カウンタ 4 をカウンタ 5 にカスケードできる

拡張カスケード機能は、性能モニタリングのサンプリング使用モデルに適応できる。ただし、カスケード・モードまたは拡張カスケード・モードでは、エラッタのために性能カウンタが PMI を生成しないことがわかっている。このエラッタは、モデル・エンコーディングが 2 のインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサに適用される。モデル・エンコーディングが 0 または 1 のインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、このエラッタは、ステッピング・エンコーディングが 09H より大きいプロセッサに適用される。

15.9.6.8. 拡張カスケード

IQ ブロック内のカウンタ 16 とカウンタ 17 は、プリサイス・イベント・ベース・サンプリングや、パイプライン内のストール状態を示すイベントのリタイアメント時のカウンティングによく使用される。カウンタ 16 やカウンタ 17 から、CCCR 内のカスケード・ビットを使用してカウンタペアのカスケードを開始できない。

拡張カスケード機能により、性能モニタリング・ツールは、カウンタ 16 とカウンタ 17 を使用して、IQ ブロック内の 2 つのカウンタのカスケードを開始できる。カウンタ 16 とカウンタ 17 からの拡張カスケードは、概念的には他のカウンタのカスケードと同じであるが、CCCR の CASCADE ビットを使用するのではなく、4 つの CASCNTxINTOy ビットのうち 1 つを使用する。

カウント条件の例

拡張カスケードの使用条件として、最初の 4096 個の命令が論理プロセッサ 0 上でリタイアした後に、論理プロセッサ 1 上でリタイアした命令をサンプリングする場合を考える。この条件で拡張カスケードをプログラムする手順は、次のとおりである。

1. カウンタ 12 に値 0 を書き込む。
2. 値 04000603H を MSR_CRU_ESCR0 に書き込む（論理プロセッサ 1 に限定する制限付きで NBOGNTAG および NBOGTAG イベントマスクを選択する操作に対応）。
3. 値 04038800H を MSR_IQ_CCCR0 に書き込み、CASCNT4INTO0 および OVF_PMI をイネーブルにする。この場合、ISR は命令アドレスに基づいてサンプリングを実行できる（ENABLE や CASCADE をセットしないこと）。

4. 値 FFFFF000H をカウンタ 16 に書き込む。
5. 値 0400060CH を MSR_CRU_ESCR2 に書き込む（論理プロセッサ 0 に限定する制限付きで NBOGNTAG および NBOGTAG イベントマスクを選択する操作に対応）。
6. 値 00039000H を MSR_IQ_CCCR4 に書き込む（OVF_PMI ではなく、ENABLE ビットをセットする）。

カスケードのもう 1 つの使い方として、マルチスレッド・アプリケーション内でストールした実行を検出できる。スレッド B 内の MOB 再生が原因で、スレッド A がストールする場合を考える。この条件で、ストールした実行のサンプルを収集するには、次の手順に従う。

1. スレッド B 上の MOB 再生をカウントするように、カウンタ B を設定する。
2. スレッド A 上のリソースストールをカウントするように、カウンタ A を設定する。カウンタ A の強制オーバーフロー・ビットと適切な CASCNTxINTOy ビットをセットする。
3. 性能モニタリング割り込みを使用して、ストールしたスレッドのプログラム実行データを収集する。

15.9.6.9. オーバーフロー時の割り込みの生成

性能カウンタは、オーバーフローしたときに性能モニタリング割り込み（PMI）を生成するように設定することができる。オーバーフローが発生すると、PMI 割り込みサービスルーチンがプロセッサやプログラムのステートに関する情報を収集できる。この情報を利用して、VTune™ パフォーマンス・アナライザなどのツールでプログラムの性能を分析してチューニングすることができる。

カウンタのオーバーフロー時の割り込みの生成をイネーブルにするには、カウンタに対応付けられた CCCR MSR の OVR_PMI フラグをセットする必要がある。オーバーフローが発生すると、ローカル APIC を介して PMI が生成される（このとき、ローカル・ベクタ・テーブル [LVT] の性能カウンタエントリは、PMI が生成した割り込みをプロセッサに伝達するようにセットアップされる）。

複数のカウンタが PMI を生成するように設定されている場合、PMI サービスルーチンにおいては、OVF フラグを使用してオーバーフローしたカウンタを判別できる。

オーバーフロー時に割り込みを生成させる場合は、イベントが所定の数だけカウントされたあとにオーバーフローを発生させる値に 1 を加えた値を、使用する性能カウンタにプリセットする必要がある。プリセットする値を選択するには、15.9.6.6.「カウンタのカスケード」で説明したように、負の数字をカウンタに書き込むのが最も簡単である。ただし、イベントを 100 回カウントしてから割り込みを生成する場合は、カウンタにマイナス 100 プラス 1（-100+1）、すなわち 99 をプリセットする必要がある。このようにすると、カウンタは、99 個のイベントをカウントしてからオーバーフロー

し、次の（100 番目の）イベントをカウントした時点で割り込みを生成する。カウントに 1 を加えることで、選択したイベントカウントに達した直後に割り込みが生成されるようになる。カウンタを介してオーバーフローが伝搬されるのを待つ必要はない。

マイクロアーキテクチャでは、イベントの生成とオーバーフロー時の割り込みの生成との間にレイテンシがある。そのため、割り込みの原因となったイベントの直後に割り込みを生成するのは、難しい場合がある。このような場合、CCCR の FORCE_OVF フラグを使用すると、通知が改善される。このフラグをセットすると、カウンタがインクリメントするたびにカウンタがオーバーフローする。結果として、カウンタがインクリメントするたびに、割り込みがトリガされるようになる。

15.9.6.10.カウンタ使用上のガイドライン

カウントのロジックは、注意を払って適切に設定しなければならない場合がある。そうしないと、コンピュータが停止することがある。タグ付けのためだけに使用する場合であっても、ESCR を使用するときは、特定の ESCR（またはそのペアの ESCR）が接続可能なカウンタの（任意の）1 つをイネーブルにしておく必要がある。そのようにしないと、結果として何もカウントされない場合がある。同様に、カウンタを使用する場合は、対応する ESCR で選択される特定のイベント（一般に 0 の選択値を持つ no_event 以外）が必要である。

15.9.7. リタイアメント時カウント

リタイアメント時カウントでは、アーキテクチャ・ステートにコミットされた作業を表すイベントだけがカウントされ、スペキュレーティブに（見込み的に）実行された後で破棄される作業は無視される。

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで使用される Intel NetBurst® マイクロアーキテクチャでは、処理効率を向上させる試みの中で、多数の見込み的なアクティビティが実行される。この見込み的なアクティビティの一例として分岐予測がある。通常、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサは、分岐の方向を予測し、次に分岐の実際の決定を見越して、予測パスに沿いながら命令をデコードおよび実行する。分岐の予測ミスが発生した場合、誤って予測されたパスに沿ってデコードおよび実行された命令の結果はキャンセルされる。実行された命令をすべてカウントするように性能カウンタがセットアップされている場合は、結果がアーキテクチャ・ステートにコミットされた命令だけではなく、結果がキャンセルされた命令もカウントに含まれる。

こうした状況でのイベントカウントのグラニュラリティを高めるため、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの性能モニタリング機能では、イベントをタグ付けした後、コミットされた結果を表すこれらのタグ付けされたイベ

ントだけをカウントする機構が提供されている。この機構は、「リタイアメント時カウント」と呼ばれる。

表 A-2. から表 A-6. は、リタイアメント時カウントを使用するときにイベントのタグ付けに使用できる、事前定義されたリタイアメント時イベントとイベント・メトリックを示している。以下の用語は、リタイアメント時カウントを説明するときに使用されるものである。

偽、非偽、リタイア

リタイアメント時イベントの説明で、「偽」の用語は、予測ミスされた分岐によって成立するパス上にあるため、キャンセルする必要のある命令または µop を指す。「リタイア」および「非偽」の用語は、実行中のプログラムの要求に応じてアーキテクチャ・ステートの変更をコミットする、実行パス上の命令または µop を指す。したがって、命令および µop は「偽」または「非偽」のいずれかであり、その両方にはならない。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの性能モニタリング・イベント（表 A-2. の Instruction_Retired や Uops_Retired など）の中には、「非偽」に対する「偽」の特徴付けに基づいて、リタイアされた命令や µop をカウントできるものがある。

タグ付け

タグ付けは、特定の性能イベントを検出した µop をマーキングする手段である。タグ付けを行えば、特定の性能イベントを検出した µop をリタイアメント時にカウントできる。実行の進行中、1 つの µop につき同じイベントが 2 回以上発生することがある。そのため、イベントを直接カウントしたのでは、いくつの µop がそのイベントを検出したのかわからない。

タグ付け機構を使用すると、µop はライフタイムの間一度だけタグ付けされるため、リタイアメント時に一度だけカウントされることになる。リタイアされたサフィックスは性能メトリック用に使用され、イベントごとにではなく、µop ごとに一回カウントがインクリメントされる。例えば、ライフタイムの間 µop は 2 回以上キャッシュ・ミスを検出する場合があるが、「ミスリタイア」メトリック（キャッシュ・ミスを検出した、リタイアされた µop の数をカウントする）は、その µop に対して一回だけインクリメントする。「ミスリタイア」メトリックは、特定の命令シーケンスのキャッシュ階層の性能を特徴付ける際に役に立つ。各種の性能メトリック、およびこれらのメトリックをインテル Pentium 4 プロセッサおよびインテルXeonプロセッサの性能イベントを使用して構築する方法については、『インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ最適化リファレンス・マニュアル』(1.4. 節「参考文献」を参照）で説明されている。

再生　一般的な状況で性能を最大にするため、Intel NetBurst マイクロアーキテクチャでは、正しい実行のための条件がすべて確実に満たされる前に、µop を積極的にスケジュールして実行する。これらの条件がすべて満たされない場合は、µop を再発行する必要がある。µop の再発行のためにインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサが使用する機構は、再生と呼ばれる。再生は、キャッシュ・ミス、依存違反、および予測できないリソースの制約などが原因で生じる。通常の操作では、多少の再生はよく生じるものであり、避けることはできない。再生の数が多すぎる場合は、パフォーマンスに問題があることが考えられる。

アシスト　あるイベントを処理するのにハードウェアがマイクロコードの支援を必要とする場合、プロセッサはアシストを使用する。例えば、浮動小数点演算の入力オペランドのアンダーフロー条件がその例である。ハードウェアは、演算を処理するためにはオペランドのフォーマットを内部的に変更する必要がある。アシストは、開始する前にマシン全体の µop をクリアするので、コストが大きくなる。

15.9.7.1. リタイアメント時カウントの使用法

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサは、指定されたイベントを検出したイベントおよびµopの両方をカウントできる。表 A-2. に記載されたリタイアメント時イベントの一部では、イベントを検出したときにµopにタグ付けすることができる。タグ付け機構は、非プリサイス・イベント・ベース・サンプリングで使用でき、これらの機構の一部は、PEBS で使用できる。タグ付け機構には以下の 4 つがあり、各機構は異なるイベントを使用して、その機構でタグ付けされた µop をカウントする。

- **フロント・エンド・タグ付け。**この機構は、フロント・エンド・イベント（例えば、トレース・キャッシュや命令カウントなど）を検出したµopのタグ付けに関係し、Front_end_eventイベントでカウントされる。
- **実行タグ付け。**この機構は、実行イベント（例えば、命令タイプなど）を検出したµopのタグ付けに関係し、Execution_Eventイベントでカウントされる。
- **再生タグ付け。**この機構は、リタイアメントが再生された µop（例えば、キャッシュ・ミス）のタグ付けに関係し、Replay_eventイベントでカウントされる。分岐予測ミスもこの機構でタグ付けされる。
- **タグなし。**この機構は、タグを使用しない。Instr_retiredイベントとUops_retiredイベントを使用する。

それぞれのタグ付け機構は、他のものとは独立している。すなわち、ある機構でタグ付けされたµopは、他の機構のµopタグ付けディテクタでは検出されない。例えば、フロント・エンド・タグ付け機構を使用してµopがタグ付けされている場合、Replay_event

は、再生タグ付け機構でもタグ付けされていないと、タグ付けされたμopとしてμopをカウントすることはない。ただし、実行タグでは、リタイアメント時に実行タグ付けを介して、最大4つまでの異なるタイプのμopをカウントできる。

PEBSを使用しているときは、タグ付け機構の独立性は保持されない。PEBSを使用するときは、一度に1つのタグ付け機構しか使用してはならない。

I/Oアクセス、キャッシュ不可アクセス、ロックされたアクセス、リターン、far転送などのμopは、タグ付けできない。

表A-2.には、リタイアメント時カウントをサポートする性能モニタリング・イベントとして、Front_end_event、Execution_event、Replay_event、Inst_retired、Uops_retiredの各イベントが記載されている。以降の各項では、これらのイベントを使用してμopをタグ付けし、そのタグ付けされたμopをカウントするためのタグ付け機構を説明する。

15.9.7.2. FRONT_END_EVENTのタグ付け機構

Front_end_eventは、以下のいずれかのイベントを検出したものとしてタグ付けされているμopをカウントする。

- **μopデコードイベント**。μopデコードイベントのμopをタグ付けするには、性能モニタリング・イベントUop_typeに対応付けられたESCRのビットを指定する必要がある。
- **トレース・キャッシュ・イベント**。トレース・キャッシュ・イベントのμopをタグ付けするには、MSR_TC_PRECISE_EVENT MSRの特定のビットを指定しなければならない場合がある（表A-4.を参照）。

表A-2.はFront_end_eventを説明し、表A-4.はFront_end_eventカウントのセットアップに使用するメトリックを説明している。

イベントをカウントするには、フロント・エンド・タグ付け機構でサポートされる表A-2.で指定されたMSRをセットし、Front_end_eventイベントマスク内のNBOGUSビットおよびBOGUSビットの両方または一方をセットする必要がある。現在サポートされているイベントで、MSR_TC_PRECISE_EVENT MSRの使用を必要とするものはない。

15.9.7.3. EXECUTION_EVENTのタグ付け機構

表A-2.はExecution_eventを説明し、表A-5.はExecution_eventカウントのセットアップに使用されるメトリックを説明している。

実行タグ付け機構は、タグ付けを引き起こす方法が他のタグ付け機構と異なる。アップストリームESCRは、検出するイベントを指定し、そのイベントを識別するタグ値

（ビット 5 ～ 8）を指定するのに使用される。ダウンストリーム ESCR は、イベント選択用の Execution_event を用いてそのタグ値識別子でタグ付けした µop を検出するのに使用される。

イベントをカウントするアップストリーム ESCR には、タグ・イネーブル・フラグ（ビット 4）をセットし、タグ値フィールドに適切なタグ値マスクを入力する必要がある。4 ビットのタグ値マスクにより、特定の µop 用にセットするタグビットを指定する。タグ値に選択される値は、ダウンストリーム ESCR で選択されるイベントマスクと同じでなければならない。例えば、1 のタグ値をセットした場合は、NBOGUS0 のイベントマスクをダウンストリーム ESCR でも同様にイネーブルにする必要がある。ダウンストリーム ESCR は、タグ付けされた µop を検出およびカウントする。ダウンストリーム ESCR 内の通常の（タグ値ではない）マスクビットにより、カウントするタグビットを指定する。マスクで選択したタグビットのいずれかがセットされている場合は、関連するカウンタが 1 だけインクリメントされる。この機構は、実行タグ付け機構でサポートされる表 A-5. のメトリックにまとめられている。タグ・イネーブル・ビットとタグ値ビットは、Execution_event を選択するのに使用するダウンストリーム ESCR には関連がない。

この 4 つの独立したタグビットを使用すると、ユーザは、リタイアメント時に最大 4 つの実行イベントを同時かつ別個にカウントできる（これは、非プリサイス・イベント・ベース・サンプリングに適用される。PEBS の場合は、15.9.8.3.「PEBS バッファのセットアップ」で説明するように、さらに制約事項がある）。アップストリーム ESCR の複数のタグ値ビット、またはダウンストリーム ESCR の複数のマスクビットをセットすれば、イベントの組み合わせを検出またはカウントすることも可能である。例えば、アップストリーム ESCR には 3H のタグ値を使用し、ダウンストリーム ESCR イベントマスクには NBOGUS0/NBOGUS1 を使用する。

15.9.7.4. REPLAY_EVENT のタグ付け機構

表 A-2. は Replay_event を説明し、表 A-6. は Replay_event カウントのセットアップに使用するメトリックを説明している。

再生機構は、リタイアメントの前に、すべての再生のうちの一部についての µop に対するタグ付けをイネーブルにする。再生機構を使用するには、MSR_PEBS_MATRIX_VERT MSR には再生を受ける可能性のある µop のタイプを選択し、IA32_PEBS_ENABLE MSR にはイベントのタイプを選択する必要がある。また、再生タグ付けは、IA32_PEBS_ENABLE MSR の UOP_Tag フラグ（ビット 24）でイネーブルにする必要がある。

表 A-6. は、再生タグ付け機構およびそれを使用するリタイアメント時イベントをサポートするメトリックを示すと共に、該当する MSR をどのように設定する必要があるかを示している。また、表 A-6. で定義されている再生タグは、プリサイス・イベン

ト・ベース・サンプリング（PEBS）を有効にする（15.9.8. 項を参照）。これらの再生タグは、IA_32_PEBS_ENABLE_MSR のビット 24 とビット 25 がセットされていない場合は、通常のサンプリングにも使用できる。タグ付けされた μop をカウントするには、これらのメトリックのそれぞれで、Replay_Event（表 A-2. を参照）を使用する必要がある。

15.9.8. プリサイス・イベント・ベース・サンプリング（PEBS: Precise Event-based Sampling）

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサのデバッグストア（DS）機構では、PEBS レコードと BTS レコードの 2 つのタイプの情報を収集し、これらをプログラムのデバッグとチューニングに使用できる（BTS 機構については、15.5.7.「分岐トレースストア（BTS: Branch Trace Store）」を参照）。

PEBS では、プリサイス・イベント・レコード・バッファ内の 1 つまたは複数の性能イベントに対応付けられているプリサイス・アーキテクチャ情報を保存可能である（プリサイス・イベント・レコード・バッファは DS セーブ領域の一部である。詳しくは、15.9.5.「DS セーブ領域」を参照）。この機構を使用するため、カウンタは、プリセットされた数だけイベントをカウントしたあとオーバーフローするように設定されている。カウンタがオーバーフローすると、プロセッサは、汎用レジスタ、EFLAGS レジスタ、命令ポインタの現在のステートを、プリサイス・イベント・レコード・バッファ内のレコードにコピーする。次にプロセッサは、性能カウンタのカウントをリセットし、カウンタを再始動させる。プリサイス・イベント・レコード・バッファがいっぱいに近づくと割り込みが生成され、プリサイス・イベント・レコードが保存される。プリサイス・イベント・レコードでは、循環式バッファはサポートされていない。

PEBS は、Execution_event、Front_end_event、Replay_event のリタイアメント時イベントの一部に対してのみサポートされる。また、PEBS は、性能カウンタ MSR_IQ_COUNTER4 MSR を使用しないと実行できない。

15.9.8.1. PEBS 機能の使用可否の検出

CPUID 命令で返される DS 機能フラグ（ビット 21）がセットされている場合は、プロセッサ上で DS 機構が使用可能である。DS 機構は、PEBS（および BTS）機能をサポートする。このビットがセットされている場合は、以下の PEBS 機能を使用できる。

- IA32_MISC_ENABLE MSR 内の PEBS_UNAVAILABLE フラグがクリアされている場合は、PEBS 機能（IA32_PEBS_ENABLE MSR を含む）を使用できることを示す。
- IA32_PEBS_ENABLE MSR 内の PEBS フラグ（ビット 24）によって、PEBS を有効（セット）または無効（クリア）にできる。
- DS セーブ領域を指すように、IA32_DS_AREA MSR をプログラムできる。

15.9.8.2. DS セーブ領域のセットアップ

DS セーブ領域をセットアップしてイネーブルにする方法は、15.5.7.2.「DS セーブ領域のセットアップ」で説明している。この手順は、PEBS と BTS で共通である。

15.9.8.3. PEBS バッファのセットアップ

PEBS においては、性能カウンタ MSR_IQ_COUNTER4 しか使用することができない。プロセッサおよびこの PEBS 用カウンタをセットアップするには、以下の手順に従うこと。

1. プリサイス・イベント・バッファ機能をセットアップする。DS バッファ管理領域のプリサイス・イベント・バッファ・ベース、プリサイス・イベント・インデックス、プリサイス・イベント絶対最大値、プリサイス・イベント割り込みしきい値、プリサイス・イベント・カウンタ・リセットの各フィールドに値をセットして（図 15-9. を参照）、メモリ内にプリサイス・イベント・レコード・バッファをセットアップする。
2. PEBS をイネーブルにする。IA32_PEBS_ENABLE MSR の PEBS イネーブル・フラグ（ビット 24）をセットする。
3. 表A-2.から表A-6.に示すように、PEBS用の1つまたは複数のESCR、MSR_IQ_COUNTER4 性能カウンタ、およびそれに対応付けられている CCCR をセットアップする。

15.9.8.4. PEBS 割り込み・サービスルーチンの作成

PEBS 機能は、同じ割り込みベクタと割り込みサービスルーチン（DS ISR と呼ばれる）を、非プリサイス・イベント・ベース・サンプリングおよび BTS 機能と共用する。PEBS 割り込みを処理するには、PEBS ハンドラコードを DS ISR に含める必要がある。DS ISR 作成のガイドラインについては、15.5.7.4.「DS 割り込みサービスルーチンの作成」を参照のこと。

15.9.8.5. DS 機構のその他の注意事項

DS 機構は、SMM では使用することはできない。この機構は、SMM モードへ移行する時点でディスエーブルになる。同様に、DS 機構はマシンチェック例外の生成時にディスエーブルになり、プロセッサの RESET および INIT でクリアされる。DS 機構は、実アドレスモードで使用できる。

15.9.9. クロックのカウント

サイクルカウント（クロックチックとも呼ばれる）は、プログラムの実行に必要な時間を測定する上で基本的な要素である。クロックチックは、命令当たりのサイクル数（CPI）などの効率比の重要な要素としても使用される。プロセッサ・クロックは、次のような条件下でチックを停止するときがある。

- CPU が行う作業が何もないときは、プロセッサはホルトにされる。例えば、コンピュータがI/O要求を処理している間、消費電力の軽減のためにプロセッサがホルトにされるときがある。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場合、性能モニタリング・カウンタをパワーダウン状態にするには、両方の論理プロセッサをホルトにする必要がある。
- ホルトにされた結果として、あるいはパワー・マネージメント技術のため、プロセッサがスリープ状態になっている場合。スリープにはさまざまなレベルがある。あるディープ・スリープ・レベルでは、タイムスタンプ・カウンタはカウントを停止する。

性能モニタリングのためにプロセッサのクロックサイクルを監視する場合、以下の3つの方法を使用できる。

- 非ホルト・クロックチック – 指定した論理プロセッサがホルト状態にもパワーセーブ状態にもなっていない期間のクロックサイクルを測定する。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場合、これらのクロックチックは論理プロセッサごとに測定できる。
- 非スリープ・クロックチック – 指定した論理プロセッサがスリープモードにもパワーセーブ状態にもなっていない期間のクロックサイクルを測定する。これらのクロックチックは、論理プロセッサごとには測定できない。
- タイムスタンプ・カウンタ – 指定した論理プロセッサがディープスリープ状態になっていない期間のクロックサイクルを測定する。これらのクロックチックは、論理プロセッサごとには測定できない。

最初の2つの方法では性能カウンタを使用する。したがって、これらのクロックチックを設定して、サンプリングでオーバーフローが発生したときに割り込みを生成できる。また、これらのクロックチックは、ツールでタイムスタンプ・カウンタを使用するより性能カウンタを読み出す方が簡単な場合にも効果的である（タイムスタンプ・カウンタは、RDTSC命令によってアクセスされる）。

大量のI/Oを処理するアプリケーションには、以下の2つの重要な比がある。

- 非ホルト CPI: 非ホルト・クロックチック数 / リタイアされた命令数。CPU が使用されているフェーズのCPIを示す。ハイパー・スレッディング・テクノロジが有効になっている場合、この比は論理プロセッサ・ベースで測定できる。

- 公称 CPI: タイムスタンプ・カウンタのチック数 / リタイアされた命令数。マシンが I/O を待機してホルトになっている期間を含む、プログラムの持続時間にわたる CPI を示す。

非ホルト・クロックチック :

以下の手順にしたがって、ESCR と CCCR をプログラムすれば、非ホルト・クロックチックが得られる。

1. global_power_events 用の ESCR を選択して、RUNNING サブイベント・マスクと測定対象のプロセッサの適切な T0_OS/T0_USR/T1_OS/T1_USR ビットを指定する。
2. 適切なカウンタを選択する。
3. CCCR 内の有効ビットをセットして、そのカウンタのカウンティングを有効にする。

非スリープ・クロックチック :

性能モニタリング・ハードウェアがパワーダウン状態になっていないときは常にクロックチックをカウントするように、性能モニタリング・カウンタを設定できる。性能モニタリング・カウンタで非スリープ・クロックチックをカウントするには、以下の手順を実行する。

1. 18 個のカウンタのうち 1 つを選択する。
2. 選択したカウンタでイベントをカウントできる任意の ESCR を選択する。ESCR のイベント選択を no_event 以外の値に設定する。この設定を省略すると、カウンタが無効にされる場合がある。
3. CCCR 内の比較ビットを 1 にセットして、しきい値の比較をオンにする。
4. CCCR 内でしきい値を 15 に設定し、補数を 1 に設定する。このしきい値を超えるイベントは存在しないため、すべてのサイクルでこのしきい値条件が満たされる。カウンタはすべてのサイクルをカウントする。この設定は、ESCR 内で指定されたすべての条件（例えば、CPL による制限）を無効にする。
5. CCCR 内の有効ビットをセットして、そのカウンタのカウンティングを有効にする。

各物理パッケージが 1 つの論理プロセッサをサポートするとすれば、ほとんどの場合、非ホルト測定基準によって得られるカウントと非スリープ測定基準によって得られるカウントは一致し、パワーセーブ状態にはならない。オペレーティング・システムは、HLT 命令を実行して、物理プロセッサをパワーセーブ状態にすることがある。

ハイパー・スレッディング（HT）テクノロジ対応プロセッサでは、各物理パッケージは 2 つ以上の論理プロセッサをサポートする。現在の HT テクノロジ対応プロセッサは、各物理プロセッサに付き 2 つの論理プロセッサをサポートしている。2 つの論理プロセッサが 2 つのスレッドを同時に実行できるが、2 つの論理プロセッサの間で実行リソースを共有せずに 1 つの論理プロセッサが処理を実行できるように、もう 1 つの論理プロセッサをホルトにもできる。

非ホルト・クロックチックは、論理プロセッサがホルトにされていないときにのみ、その論理プロセッサのプロセッサ・クロック・サイクル数をカウントするように設定できる（このカウントには、その論理プロセッサがホルト状態への移行を完了するまでのクロックサイクルの一部も含まれる）。HTテクノロジに対応した物理プロセッサは、すべての論理プロセッサがホルトになった場合、パワーセーブ状態に移行する。非スリープ・クロックチック機構は、CCCR内のフィルタリング機構を利用する。この機構は、1つの論理プロセッサがホルト状態にもパワーセーブ状態にもなっていない間は値が増加し続ける。アプリケーションは、OSのアイドルループに制御を渡すOSサービスを利用して、プロセッサをパワーセーブ状態に移行できる。このアイドルループは、プロセッサによって異なる期間中プロセッサが行う作業がない場合、プロセッサをパワーセーブ状態に移行する。

タイムスタンプ・カウンタ

タイムスタンプ・カウンタは、スリープピンがアサートされていないときや、システムバス上のクロック信号がアクティブになっているときは、常にインクリメントされる。このカウンタは、RDTSC命令で読み出される。2回の読み出しの値の差（モジュロ2^{64}）を計算することにより、2回の読み出しの間のプロセッサ・クロック数が求められる。

タイムスタンプ・カウンタと非スリープ・クロックチックは、事実上すべての場合に一致するはずである。ただし、物理パッケージ内の両方の論理プロセッサがホルトにされたため、（性能モニタリング・ハードウェアを含む）チップの大部分がパワーダウン状態になる可能性がある。この状態でも、システムバス上のクロック信号はアクティブであるため、タイムスタンプ・カウンタはインクリメントされ続ける。性能モニタリング・ハードウェアがパワーセーブ状態になりパワーダウン状態になるため、非スリープ・クロックチックのインクリメントは停止する。

15.9.10. オペレーティング・システムの注意点

DS機構は、オペレーティング・システムが障害分析を容易に行うためのデバッグ拡張機能として使用できる。この機能を使用すると、実施されるすべての分岐で発生するトレースストアの影響のために、25～30回のスローダウンが生じる場合がある。

使用目的にもよるが、分岐レコードまたはPEBSレコードの一部になっている命令ポインタは、対応するプロセスと関連付けられていることが必要である。1つの解決策として、DS固有のオペレーティング・システム・モジュールをコンテキスト・スイッチにチェーンできるようにする方法がある。このようにすれば、コンテキスト・スイッチのたびに適切にセーブされセットアップされた設定領域を指すMSR、各対象プロセスについて、個別のバッファを保持できる。

BTS機能がイネーブルになっている場合は、これをディスエーブルにする必要がある。また、BTS機能は、プロセッサ・コンテキストが失われるスリープステートへシステ

ムが移行したときにストアされたステートでなければならない。このステートは、スリープステートから復帰するときに復元する必要がある。

DS割り込みには､エンドレスな割り込みループの生成を防止するためのトラップゲートではなく、割り込みゲートを使用する必要がある。

バッファを含むページは、すべてのプロセス / 論理プロセッサについて、CR3 を変更しても DS アドレスが変更されないような、同じ物理アドレスへのマッピングが必要である。この要件（すなわち、機能がスレッド/プロセスごとにイネーブルになる）が満たされない場合、オペレーティング・システムは、コンテキスト・スイッチ・コードによって機能をイネーブル/ディスエーブルにしなければならない。

15.10. 性能モニタリングとハイパー・スレッディング・テクノロジ

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサの性能モニタリング機能は、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの性能モニタリング機能とほぼ同じである。ただし、ハイパー・スレッディング・テクノロジ対応プロセッサの性能モニタリング機能は、次のように拡張されている。

- 論理プロセッサ ID によって制限されたイベントを選択するように、性能カウンタをプログラムできる。
- 物理プロセッサ内の特定の論理プロセッサに、性能モニタリング割り込みを転送できる。

この節では、性能カウンタの使用、論理プロセッサ ID によるイベントの制限、ESCR および CCCR 内の追加プログラマブル・ビットに関連するプログラミング・インターフェイスと、特殊目的の IA32_PEBS_ENABLE、MSR_PEBS_MATRIX_VERT、MSR_TC_PRECISE_EVENT の各 MSR について説明する。

ハイパー・スレッディング・テクノロジ対応インテル IA-32 プロセッサでは、これらのレジスタは、物理プロセッサ内の 2 つの論理プロセッサによって共有される。これらの共有レジスタを使用して、一方または両方の論理プロセッサ上で性能イベントを監視できるように､ESCR MSR および CCCR MSR と IA32_PEBS_ENABLE MSR に対して新しいフラグが追加された。以下の各項では、これらの追加フラグの機能と、ハイパー・スレッディング・テクノロジがアクティブになっているときにこれらの追加フラグがイベント監視に与える影響について説明する。

15.10.1. ESCR MSR

図 15-14. は、ハイパー・スレッディング・テクノロジ対応インテル IA-32 プロセッサ内の ESCR MSR のレイアウトを示している。

それぞれのフラグとフィールドの機能は次のとおりである。

T1_USR フラグ、ビット 0

セットされている場合は、スレッド 1（論理プロセッサ 1）が現行特権レベル（CPL）= 1、2、または 3 で実行されているときにイベントがカウントされる。これらの特権レベルは、一般的に、アプリケーション・コードと保護されていないオペレーティング・システム・コードによって使用される。

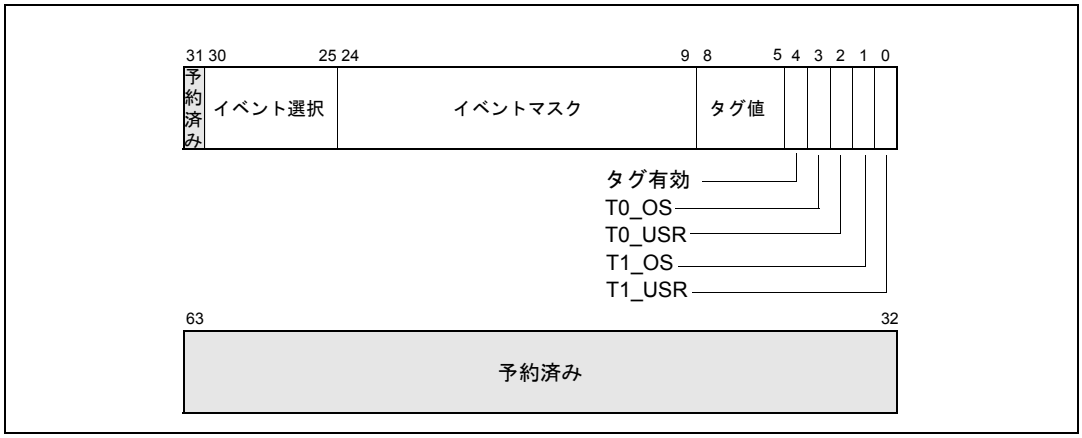


図 15-14. ハイパー・スレッディング・テクノロジ対応のインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、インテル® Xeon™ プロセッサ MP のイベント選択制御レジスタ（ESCR）

T1_OS フラグ、ビット 1

セットされている場合は、スレッド 1（論理プロセッサ 1）が CPL = 0 で実行されているときにイベントがカウントされる。これらの特権レベルは、一般的に、保護されているオペレーティング・システム・コード用に予約されている（T1_OS フラグと T1_USR フラグが両方ともセットされている場合は、スレッド 1 のイベントはすべての特権レベルでカウントされる）。

T0_USR フラグ、ビット 2

セットされている場合は、スレッド 0（論理プロセッサ 0）が CPL = 1、2、または 3 で実行されているときにイベントがカウントされる。

T0_OS フラグ、ビット 3

セットされている場合は、スレッド 0（論理プロセッサ 0）が CPL = 0 で

実行されているときにイベントがカウントされる（T0_OS フラグと T0_USR フラグが両方ともセットされている場合は、スレッド 0 のイベントはすべての特権レベルでカウントされる）。

タグ有効、ビット 4

セットされている場合は、リタイアメント時イベントのカウンティングを支援する μop のタギングが有効になる。クリアされている場合は、タギングは無効になる。15.9.7.「リタイアメント時カウント」を参照。

タグ値フィールド、ビット 5 ～ 8

リタイアメント時イベントのカウンティングを支援する μop に関連付けられるタグ値を選択する。

イベント・マスク・フィールド、ビット 9 ～ 24

イベント選択フィールドで選択されているイベントクラスから、カウントするイベントを選択する。

イベント選択フィールド、ビット 25 ～ 30

カウントされるイベントのクラスを選択する。このクラス内でカウントされるイベントは、イベント・マスク・フィールドで選択される。

T0_OS および T0_USR フラグと T1_OS および T1_USR フラグにより、インテル® Xeon™ プロセッサ MP 内の特定の論理プロセッサ（0 または 1）について、イベントのカウンティングとサンプリングを指定できる（インテル Xeon プロセッサ MP 内の論理プロセッサの指定については、7.7.5.「MP システム内の論理プロセッサの識別」を参照のこと）。

性能モニタリング・イベントの中には、インテル Xeon プロセッサ MP 内の論理プロセッサ単位で検出できないものもある（15.10.4.「性能モニタリング・イベント」を参照）。（イベント・マスク・ビットで指定される）一部のサブイベントは、検出されるイベントがどちらの論理プロセッサに関連付けられているかに関係なく、カウントまたはサンプリングされる。

15.10.2. CCCR MSR

図 15-15. は、ハイパー・スレッディング・テクノロジ対応インテル IA-32 プロセッサ内の CCCR MSR のレイアウトを示している。

それぞれのフラグとフィールドの機能は次のとおりである。

有効フラグ、ビット 12

セットされている場合は、カウンタは有効になる。クリアされている場合は、カウンタは無効になる。このフラグは、リセット時にクリアされる。

ESCR 選択フィールド、ビット 13 ～ 15

このビットで指定された ESCR を使用して、CCCR に関連するカウンタでカウントされるイベントが選択される。

アクティブ・スレッド・フィールド、ビット 16 および 17

どちらの論理プロセッサがアクティブ（スレッドを実行している）かに基づいて、カウンティングを有効にする。このフィールドは、論理プロセッサの状態（アクティブまたは非アクティブ）に基づいてイベントのフィルタリングを有効にする。このフィールドのエンコーディングは次のとおりである。

00 － なし。両方の論理プロセッサが非アクティブのときにのみカウントする。

01 － 単一。一方の論理プロセッサ（0 または 1）だけがアクティブのときにのみカウントする。

10 － 両方。両方の論理プロセッサがアクティブのときにのみカウントする。

11 － 任意。一方または両方の論理プロセッサがアクティブのときにカウントする。

ただし、ホルト状態の論理プロセッサや「SIPI 待機」状態の論理プロセッサは、非アクティブと見なされる。

比較フラグ、ビット 18

セットされている場合は、イベントカウントのフィルタリングが有効になる。クリアされている場合は、フィルタリングは無効になる。フィルタリング方法は、しきい値フラグ、補数フラグ、エッジフラグで選択される。

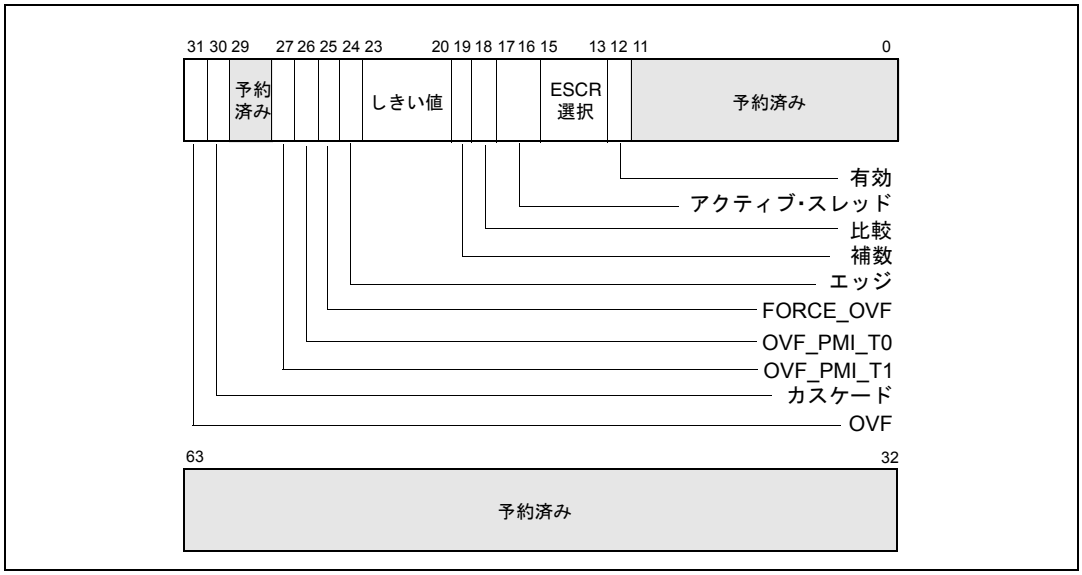


図 15-15. カウンタ設定制御レジスタ（CCCR）

補数フラグ、ビット 19

着信イベントカウントとしきい値を比較する方法を選択する。セットされている場合は、イベントカウントがしきい値より小さいか等しい場合に、単一のカウントが性能カウンタに渡される。クリアされている場合は、イベントカウントがしきい値より大きい場合に、カウントが性能カウンタに渡される（15.9.6.2.「イベントのフィルタリング」を参照）。補数フラグは、比較フラグがセットされている場合にのみ有効である。

しきい値フィールド、ビット 20 ～ 23

比較に使用するしきい値を選択する。プロセッサは、比較フラグがセットされている場合にのみ、このフィールドをチェックする。また、プロセッサは、補数フラグの設定に基づいて、実行するしきい値比較のタイプを決定する。このフィールドに入力できる値の有効範囲は、カウントされるイベントのタイプによって異なる（15.9.6.2.「イベントのフィルタリング」を参照）。

エッジフラグ、ビット 24

セットされている場合は、イベントカウントをフィルタリングするためのしきい値比較の出力に対して、立ち上がりエッジ（偽から真）によるエッジ検出が有効になる。クリアされている場合は、立ち上がりエッジの検出は無効になる。このフラグは、比較フラグがセットされている場合にのみ有効である。

FORCE_OVF フラグ、ビット 25

セットされている場合は、カウンタがインクリメントされるたびにカウンタのオーバーフローが発生する。クリアされている場合は、カウンタが実際にオーバーフローしたときにのみオーバーフローが発生する。

OVF_PMI_T0 フラグ、ビット 26

セットされている場合は、カウンタのオーバーフローが発生したとき、性能モニタ割り込み（PMI）が論理プロセッサ 0 に送信される。クリアされている場合は、論理プロセッサ 0 に対する PMI の生成は無効になる。ただし、この PMI は、カウンタがオーバーフローした後、次のイベントカウントで発生する。

OVF_PMI_T1 フラグ、ビット 27

セットされている場合は、カウンタのオーバーフローが発生したとき、性能モニタ割り込み（PMI）が論理プロセッサ 1 に送信される。クリアされている場合は、論理プロセッサ 1 に対する PMI の生成は無効になる。ただし、この PMI は、カウンタがオーバーフローした後、次のイベントカウントで発生する。

カスケード・フラグ、ビット 30

セットされている場合は、同じカウンタグループ内の一方のカウンタペアのうち 1 つのカウンタがオーバーフローしたとき、もう一方のカウンタペアの代替カウンタ上でカウンティングが有効になる（詳細については、15.9.2.「性能カウンタ」を参照）。クリアされている場合は、カウンタのカスケードは無効になる。

OVF フラグ、ビット 31

セットされている場合は、カウンタがオーバーフローしたことを示す。このフラグはスティッキー・フラグであり、ソフトウェアが明示的にクリアしなければならない。

15.10.3. IA32_PEBS_ENABLE MSR

ハイパー・スレッディング・テクノロジ対応 IA-32 プロセッサでは、IA32_PEBS_ENABLE MSR 内の 2 ビットによって、PEBS を有効にし、条件を制限できる。つまり、ビット 25（ENABLE_PEBS_MY_THR）によって PEBS を有効にし、ビット 26（ENABLE_PEBS_OTH_THR）によって PEBS の対象を制限する。これらのビットは、論理プロセッサ ID（T0 または T1）によって特定の論理プロセッサを明示的に指定するわけではない。これらのビットに基づいて、ソフトウェア・エージェントは、そのエージェントを実行している同じ論理プロセッサ（「マイスレッド」）上か、同じ物理パッケージ内でそのエージェントを実行していないもう 1 つの論理プロセッサ（「他スレッド」）上で、それ以降の実行スレッドに対する PEBS を有効にする。

PEBS は、リタイアメント時イベント Execution_event、Front_end_event、Replay_event に対してのみ適用できる。また、PEBS の実行に使用できる性能カウンタは、論理プロセッサ 0 用の MSR_IQ_CCCR4（MSR アドレス 370H）と論理プロセッサ 1 用の MSR_IQ_CCCR5（MSR アドレス 371H）の 2 つだけである。

注記

性能モニタリング・ツールは、プロセッサ・アフィニティ・マスクを使用して、IA32_PEBS_ENABLE MSR 内の ENABLE_PEBS_MY_THR ビットと ENABLE_PEBS_OTH_THR ビットを変更する必要があるカーネル・モード・コンポーネントを、特定の論理プロセッサに結合する必要がある。これによって、これらのカーネル・モード・コンポーネントが、OS のスケジューリングのために異なる論理プロセッサの間での移行を防止できる。

15.10.4. 性能モニタリング・イベント

インテル® Xeon™ プロセッサ MP では、表 A-1. と表 A-2. に記載されているすべてのイベントを使用できる。ハイパー・スレッディング・テクノロジがアクティブになっている場合、初期 APIC ID のビット 0 に対応する論理プロセッサ ID によって、多くの性能モニタリング・イベントの検出条件を制限できる。これによって、論理プロセッサのうちいくつかまたはすべてで、イベントをカウントできる。ただし、イベントの中には、論理プロセッサやスレッドを指定できないものもある。

各イベントは、次の 2 つのカテゴリのいずれかに該当する。

- スレッド固有（TS）。このイベントは、特定の論理プロセッサ上で発生するものとして制限できる。
- スレッド独立（TI）。このイベントは、特定の論理プロセッサに関連付けて制限できない。

表 A-7. は、表 A-1. と表 A-2. で説明する各イベントについて、論理プロセッサ固有の情報（TS または TI）を示している。

例えば、論理プロセッサ T0 上で TS イベントが発生した場合、そのイベントのカウント方法は（表 15-7. に示すように）、イベントカウンタの設定に使用される ESCR 内の T0_USR フラグと T0_OS フラグの設定によって決まる。T1_USR フラグと T1_OS フラグは、カウントに影響を与えない。

表 15-7. 論理プロセッサと CPL による論理プロセッサ固有（TS）イベントの制限の影響

	T1_OS/T1_USR=00	T1_OS/T1_USR=01	T1_OS/T1_USR=11	T1_OS/T1_USR=10
T0_OS/T0_USR=00	ゼロカウント	T1 が USR のときにカウント	T1 が OS または USR のときにカウント	T1 が OS または USR のときにカウント
T0_OS/T0_USR=01	T0 が USR のときにカウント	T0 が USR または T1 が USR のときにカウント	(a) T0 が USR、または (b) T1 が OS、または (c) T1 が USR のときにカウント	(a) T0 が OS、または (b) T1 が OS のときにカウント
T0_OS/T0_USR=11	T0 が OS または USR のときにカウント	(a) T0 が OS、または (b) T0 が USR、または (c) T1 が USR のときにカウント	CPL、T0、T1 に関係なくカウント	(a) T0 が OS、または (b) T0 が USR、または (c) T1 が OS のときにカウント
T0_OS/T0_USR=10	T0 が OS のときにカウント	T0 が OS または T1 が USR のときにカウント	(a) T0 が OS、または (b) T1 が OS、または (c) T1 が USR のときにカウント	(a) T0 が OS、または (b) T1 が OS のときにカウント

イベント・マスク・フィールド内のビットが TI のとき、関連する ESCR のビット 0 ～ 3 の指定による影響を表 15-8. に示す。表 15-8. は、付録 A の TI イベントについて、T0_USR、T0_OS、T1_USR、T1_OS の各ビットを選択した上で指定した場合の影響を示している。

表 15-8. 論理プロセッサと CPL による非論理プロセッサ固有（TI）イベントの制限の影響

	T1_OS/T1_USR=00	T1_OS/T1_USR=01	T1_OS/T1_USR=11	T1_OS/T1_USR=10
T0_OS/T0_USR=00	ゼロカウント	(a) T0 が USR、または (b) T1 が USR のときにカウント	CPL、T0、T1 に関係なくカウント	(a) T0 が OS、または (b) T1 が OS のときにカウント
T0_OS/T0_USR=01	(a) T0 が USR、または (b) T1 が USR のときにカウント	(a) T0 が USR、または (b) T1 が USR のときにカウント	CPL、T0、T1 に関係なくカウント	CPL、T0、T1 に関係なくカウント
T0_OS/T0_USR=11	CPL、T0、T1 に関係なくカウント	CPL、T0、T1 に関係なくカウント	CPL、T0、T1 に関係なくカウント	CPL、T0、T1 に関係なくカウント
T0_OS/T0_USR=10	(a) T0 が OS、または (b) T1 が OS のときにカウント	CPL、T0、T1 に関係なくカウント	CPL、T0、T1 に関係なくカウント	(a) T0 が OS、または (b) T1 が OS のときにカウント

15.11.性能モニタリング・カウンタ（P6 ファミリ・プロセッサ）

P6 ファミリ・プロセッサは 40 ビットの性能カウンタを 2 つ備えており、同時に 2 種類のイベントのモニタを可能にしている。これらのカウンタでは、イベント回数をカウントすることも、持続時間を計測することもできる。イベント回数をカウントするときは、カウンタは指定されたイベントが発生するたびに、またはイベントが指定された回数発生するたびに値を 1 増加する。持続時間を計測するときは、カウンタは、指定された条件が真である間に発生するプロセッサ・クロックの数をカウントする。両カウンタは、どの特権レベルにおいても、発生するイベントのカウントまたは持続時間の計測が可能である。付録 A「性能モニタリング・イベント」の表 A-10. に、P6 ファミリの性能モニタリング・カウンタでカウントできるイベントの一覧が示してある。

注記

付録 A「性能モニタリング・イベント」に記載している性能モニタリング・イベントは、性能をチューニングする際のガイドとして使用されることを想定している。報告されるカウンタ値が完全に正確であることは保証されていないため、チューニングする際の参考程度として使用するべきである。該当する個所には、既知の矛盾点を記載している。

これらの性能モニタリング・カウンタは、4 つの MSR、すなわち性能イベント選択 MSR（PerfEvtSel0 および PerfEvtSel1）と性能カウンタ MSR（PerfCtr0 および PerfCtr1）によってサポートされる。これらのレジスタは、それぞれ、RDMSR および WRMSR 命令を使用して読み書きができる。これらの命令を使用して性能モニタリング・カウンタにアクセスできるのは、特権レベル 0 での動作時に限られる。PerfCtr0 および PerfCtr1 両 MSR は、RDPMC（性能モニタリング・カウンタ読み取り）命令を使用して、どの特権レベルからも読むことができる。

注記

PerfEvtSel0、PerfEvtSel1、PerfCtr0、PerfCtr1 の各 MSR と、表 A-10. のイベントは、P6 ファミリ・プロセッサにモデル固有であり、今後の IA-32 プロセッサで提供される保証はない。

15.11.1. PerfEvtSel0 および PerfEvtSel1 MSR

PerfEvtSel0 および PerfEvtSel1 両 MSR は性能モニタリング・カウンタの動作の制御用であり、それぞれのカウンタのセットアップにこれらのレジスタが 1 つずつ使用される。両 MSR は、カウント対象のイベント、カウントする方法、カウントが行われる特権レベルを指定する。図 15-16. に、これらの MSR のフラグとフィールドを示す。

以降に、PerfEvtSel0 MSR と PerfEvtSel1 MSR の各フラグとフィールドの機能を示す。

イベント選択フィールド（ビット 0 ～ 7）

モニタの対象とするイベントを選択する（イベントの一覧と各イベントの 8 ビット・コードについては、表 A-10. を参照）。

ユニットマスク（UMASK）フィールド（ビット 8 ～ 15）

イベント選択フィールドで選択されたイベントを詳細化つまり限定する。例えば、一部のキャッシュ・イベントについては、キャッシュ・ステートの MESI プロトコル修飾子としてマスクが使用される（表 A-10. を参照）。

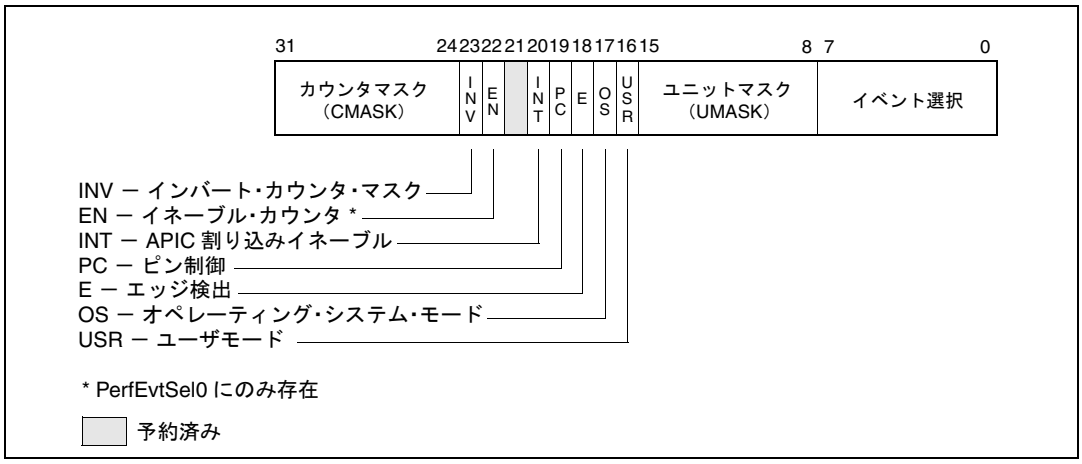


図 15-16. PerfEvtSel0 および PerfEvtSel1 MSR

USR（ユーザモード）フラグ（ビット 16）

プロセッサが特権レベル 1、2、または 3 で動作しているときだけ、イベントがカウントされるよう指定する。このフラグは OS フラグと併用できる。

OS（オペレーティング・システム・モード）フラグ（ビット 17）

プロセッサが特権レベル 0 で動作しているときだけ、イベントがカウントされよう指定する。このフラグは USR フラグと併用できる。

E（エッジ検出）フラグ（ビット 18）

セットすると、イベントのエッジ検出をイネーブルにする。プロセッサは、他のフィールドで表せるどの条件についても、ディアサート状態からアサート状態への遷移の回数をカウントする。このメカニズムは、バック・ツー・バック・アサーションを区別できない点に制約がある。このメカニズムにより、ソフトウェアは特定のステートで経過した非常に短い時間を計測できるだけでなく、そのようなステートでの平均経過時間（例えば、割り込み処理の待ち時間）も計測できる。

PC（ピン制御）フラグ（ビット 19）

セットすると、プロセッサは性能モニタリング・イベントが発生したとき PMi ピンをトグルし、カウンタの値を 1 増加する。このフラグをクリアすると、プロセッサはカウンタがオーバフローしたとき PM*i* ピンをトグルする。ピンのトグルとは、定義によれば、1 バス・クロックの間ピンをアサートし、その後ディアサートすることである。

INT（APIC 割り込みイネーブル）フラグ（ビット 20）

セットすると、プロセッサはカウンタがオーバーフローしたときそのローカル APIC を通じて例外を発生する。

EN（イネーブル・カウンタ）フラグ（ビット 22）

このフラグは、PerfEvtSel0 MSR にしか存在しない。このフラグをセットすると、両方の性能モニタリング・カウンタの性能カウントがイネーブルになる。クリアすると、両カウンタがディスエーブルになる。

INV（インバート）フラグ（ビット 23）

セットすると、カウンタマスク比較の結果を反転する。その結果、「より大きい」と「より小さい」の 2 つの比較が可能になる。

カウンタマスク（CMASK）フィールド（ビット 24 ～ 31）

ゼロ以外のとき、プロセッサはこのマスクをシングルサイクル内のイベントのカウント数と比較する。イベントのカウントが、このマスクに等しいか大きい場合は、カウンタの値が 1 だけ増加する。それ以外の場合は増加しない。このマスクは、1 クロックの間にイベントが複数回発生する場合（例えば、1 クロックの間に複数の命令がリタイアしする場合）だけイベントのカウントに使用できる。カウンタ・マスク・フィールドが 0 の場合は、カウンタの値は各サイクルでそれぞれ発生したイベントの数だけ増加する。

15.11.2. PerfCtr0 および PerfCtr1 MSR

性能モニタリング・カウンタ MSR（PerfCtr0 および PerfCtr1）は、内容として、選択されたカウント対象イベントのイベントカウント値、または持続時間カウント値を持つ。これらのカウンタは RDPMC 命令で読むことができるが、RDPMC 命令は任意の特権レベルおよび仮想 8086 モードで実行されるプログラムまたはプロシージャで使用できる。制御レジスタ CR4 の PCE フラグ（ビット 8）により、この命令を特権レベル 0 で実行されるプログラムまたはプロシージャだけに使用させるように制限できる。

RDPMC 命令は、他の命令とシリアル化もオーダリングもされることはない。したがって、カウンタを読むのに、RDPMC 命令は必ずしもそれより前のすべての命令の実行が終わるまで待っているわけではない。同様に、RDPMC 命令の操作が行われる前にそれより後の命令も実行を開始できる。

RDMSR および WRMSR 命令を使用して性能カウンタを直接操作できるのは、特権レベル 0 で実行されるオペレーティング・システムだけである。安全保護を考慮するオペレーティング・システムでは、一般的に、システム初期化時に PCE フラグをクリアして性能モニタリング・カウンタへのユーザアクセスをディスエーブルにし、その代わりに、ユーザアクセス可能なプログラミング・インターフェイスを提供し、それで RDPMC 命令をエミュレートできるようにする。

WRMSR 命令は、性能モニタリング・カウンタ MSR（PerfCtr0 および PerfCtr1）に任意に書き込むことはできない。代りに、各 MSR の下位 32 ビットに任意の値を書き、上位 8 ビットは、ビット 31 の値にしたがって符号拡張できる。この操作により、性能モニタリング・カウンタに正負両方の値を書き込める。

15.11.3. 性能モニタリング・カウンタの始動と停止

性能モニタリング・カウンタは、PerfEvtSel0 および PerfEvtSel1 のいずれか、あるいは両方の MSR に有効なセットアップ情報を書き込み、PerfEvtSel0 MSR のイネーブル・カウンタ・フラグをセットすることにより始動する。セットアップが有効であれば、カウンタは、イネーブル・カウンタ・フラグをセットした WRMSR 命令の実行後にカウントを開始する。性能モニタリング・カウンタは、イネーブル・カウンタ・フラグをクリアするか、または PerfEvtSel0 および PerfEvtSel1 両 MSR の全ビットをクリアすると停止できる。カウンタ 1 は、PerfEvtSel1 MSR をクリアして、単独に停止できる。

15.11.4. イベント / タイムスタンプ・モニタリング・ソフトウェア

性能モニタリング・カウンタおよびタイムスタンプ・カウンタを使用するには、オペレーティング・システムはイベント・モニタリング・デバイス・ドライバを提供する必要がある。このドライバは、以下の操作を扱うプロシージャを備えていなければならない。

- フィーチャ・チェック。
- カウンタの初期化と始動。
- カウンタの停止。
- イベントカウンタの読み取り。
- タイムスタンプ・カウンタの読み取り。

現在使用されているプロセッサが性能モニタリング・カウンタおよびタイムスタンプ・カウンタをサポートするかどうかは、イベント・モニタ・フィーチャ判定プロシージャによって判定しなければならない。このプロシージャは、CPUID 命令が返したプロセッサのファミリおよびモデルを、性能モニタリングをサポートすることがわかっているプロセッサのそれらと比較する。（性能モニタリング・カウンタをサポートしているのは、インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサである。）このプロシージャは、さらに、CPUID 命令から EDX レジスタに返された MSR フラグと TSC フラグを調べて、MSR と RDTSC 命令がサポートされているかどうかを判定する。

カウンタ初期化 / 始動プロシージャは、カウント対象のイベントと、イベントのカウント方法を指定する PerfEvtSel0 および PerfEvtSel1 MSR のいずれかまたは両方を設定し、カウンタ MSR（PerfCtr0 および PerfCtr1）を開始点のカウント値に初期化する。カウンタ停止プロシージャは性能モニタリング・カウンタを停止する。（カウンタの始動と停止の詳細については、15.11.3.「性能モニタリング・カウンタの始動と停止」を参照。）

PerfCtr0 および PerfCtr1 MSR の値はカウンタ読み取りプロシージャで読み、またタイムスタンプ・カウンタはタイムスタンプ・カウンタ読み取りプロシージャで読む。これらのプロシージャは、アプリケーション・コードからのカウンタの読み取りを可能にする RDTSC および RDPMC 命令をイネーブルにする代替手段として利用できる。

15.11.5. モニタリング・カウンタのオーバーフロー

P6ファミリ・プロセッサは、性能モニタリング・カウンタがオーバーフローしたときにローカルAPIC割り込みを発生させるかどうかの選択オプションを備えている。このメカニズムは、PerfEvtSel0 MSRかPerfEvtSel1 MSRの割り込みイネーブル・フラグをセットするとイネーブルにできる。このオプションの第一の用途は、性能の統計的サンプリングである。

このオプションを使用するには、オペレーティング・システムには以下の機能が必要である。

- カウンタ・オーバーフロー割り込み処理用の割り込みベクタを提供する。
- APIC PERFローカル・ベクタ・エントリを初期化して性能モニタ・カウンタ・オーバフロー・イベントの処理をイネーブルにする。
- 命令を1つも実行しないで復帰するスタブ例外ハンドラを指すIDTのエントリを提供する。
- 実際の割り込みハンドラを提供し、さらにその割り込みルーチンを指す予約済みIDTエントリを修正するイベント・モニタ・ドライバを提供する。

カウンタ・オーバーフローによって割み込みをかけられたとき、割り込みハンドラは次の処置を行う必要がある。

- 割り込み発生時に、命令ポインタ（EIPレジスタ）、コード・セグメント・セレクタ、TSSセグメント・セレクタ、およびカウンタの各値、ならびにその他の関連情報をセーブする。
- カウンタをその初期設定にリセットし、割り込みから復帰する。

プロファイルの対象となったアプリケーションの性能解析用に収集された情報は、イベント・モニタ・アプリケーション・ユーティリティまたはその他のアプリケーション・プログラムで読むことができる。

15.12.性能モニタリング（インテル® Pentium® プロセッサ）

インテル® Pentium® プロセッサは40ビットの性能カウンタを2つ備えているが、これらのカウンタはイベント数のカウントにも、持続時間の計測にも使用できる。性能モニタリング・カウンタは3つのMSR、つまり制御/イベント選択MSR（CESR）および2つの性能カウンタMSR（CTR0およびCTR1）によってサポートされている。

これらのレジスタは、それぞれRDMSRおよびWRMSR命令を使用して読み書きできる。これらのレジスタには、特権レベル0での動作時だけこれらの命令を使用してアクセスできる。各カウンタには、カウンタのステートを外部ハードウェアに示すための関連外部ピン（PM0/BP0およびPM1/BP1）がある。

注記

CESR、CTR0、およびCTR1の各MSRと表A-10.に示されているイベントはインテルPentiumプロセッサにモデル固有である。

付録B「モデル固有レジスタ（MSR）」に記載している性能モニタリング・イベントは、性能をチューニングする際のガイドとして使用されることを想定している。報告されるカウンタ値が完全に正確であることは保証されていないため、チューニングする際の参考程度として使用するべきである。該当する個所には、既知の矛盾点を記載している。

15.12.1. 制御/イベント選択レジスタ（CESR）

32ビットの制御/イベント選択MSR（CESR）の用途は、性能モニタリング・カウンタCTR0およびCTR1の動作、およびそれぞれの関連ピンの制御である（図15-17.を参照）。各カウンタを制御するために、CESRレジスタには6ビットのイベント選択フィールド（ES0およびES1）、ピン制御フラグ（PC0およびPC1）、3ビットのカウンタ制御フィールド（CC0およびCC1）がある。これらのフィールドの機能を以下に示す。

ES0およびES1（イベント選択）フィールド（ビット0～5、ビット16～21）

（フィールドにイベントコードを入力することにより）モニタの対象とするイベントを2つまで選択できる。選択可能なイベントコードについては、表A-10.の一覧を参照。

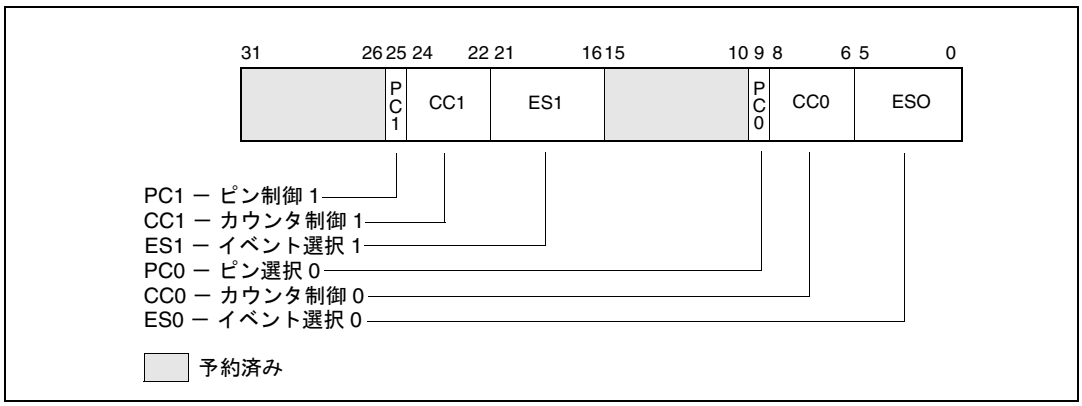


図 15-17. CESR MSR（インテル® Pentium® プロセッサのみ）

CC0 および CC1（カウンタ制御）フィールド（ビット 6 ～ 8、ビット 22 ～ 24）

カウンタの動作を制御する。選択可能な制御コードは以下のとおり。

CCn	意味
000	何もカウントしない（カウンタ・ディスエーブル状態）
001	CPL が 0、1、または 2 の間、選択されたイベントをカウントする
010	CPL が 3 の間、選択されたイベントをカウントする
011	CPL の値にかかわりなく、選択されたイベントをカウントする
100	何もカウントしない（カウンタ・ディスエーブル状態）
101	CPL が 0、1、または 2 の間、クロックをカウントする（持続時間を計測する）
110	CPL が 3 の間、クロックをカウントする（持続時間を計測する）
111	CPL の値にかかわりなく、クロックをカウントする（持続時間を計測する）

最上位ビットはイベントのカウントかクロックのカウント（持続時間の計測）かを選択し、中間のビットは CPL が 3 のときカウントをイネーブルにし、最下位ビットは CPL が 0、1、または 2 のときにカウントをイネーブルにすることに注意する。

PC0 および PC1（ピン制御）フラグ（ビット 9、ビット 25）

外部性能モニタリング・カウンタ・ピン（PM0/BP0 および PM1/BP1）の機能を選択する。これらのフラグのいずれかを 1 にセットすると、プロセッサはカウンタがオーバフローしたときにそのフラグの関連ピンをアサートする。フラグを 0 にセットすると、カウンタ値が 1 増加したとき関連ピンがアサートされる。これらのフラグによって、オーバフローまたはカウンタ値増加条件を示すように各ピンを個別にプログラムできる。信号のラッチおよびバッファ操作が介在するため、これらのピンの信号によるイベントの外部通知には内部イベントの 2、3 クロック分の遅延が伴う。

カウンタは、その内容をサンプリングするために停止させる必要はないが、新しいイベントに切り替える前には停止し、クリアまたはプリセットしなければならない。1 つのカウンタを個別に設定することはできない。1 つのイベントだけを変更する必要がある場合は、CESR レジスタを読み、該当ビットを修正し、次に全ビットを CESR に書き戻さなければならない。リセット時には、CESR レジスタの全ビットがクリアされる。

15.12.2. 性能モニタピンの用途

性能モニタリング・カウンタの値が増加し、「発生イベント」がカウントされたときにその事実を示すように性能モニタピン PM0/BP0 および PM1/BP1 のいずれかまたは両方が構成されている場合、イベントが発生するたびに対応のピンがアサートされる。「持続時間イベント」がカウントされたときは、イベントが持続している間ずっと対応の PM ピンがアサートされている。カウンタがオーバフローしたときにその事実を示すように性能モニタピンが構成されている場合、対応の PM ピンはカウンタがオーバフローするまでアサートされない。

カウンタの値が増加したことを信号で通知するように性能モニタピン PM0/BP0 および PM1/BP1 のいずれかまたは両方が構成されている場合、カウンタが 1 クロックの間に値を 1 または 2 増加することがあっても、これれらのピンではイベントが発生したという事実しか示せないので注意する。さらに、内部クロックの周波数は外部クロックのそれより高い可能性があるので、1 外部クロックが複数の内部クロックに対応することがある。

カウンタのオーバフローを信号で通知するようにイベントピンをプログラムした場合、「カウントアップ上限」機能が利用できる。各カウンタは 40 ビットなので、ビット 39 からのキャリーがオーバフローを示す。カウンタは $2^{40} - 1$ 未満の特定値にプリセットできる。カウンタがイネーブルにされていて、イベントが指定された回数発生すると、その後でカウンタはオーバフローする。

その約 5 クロック後に、オーバフローが外部に示され、割り込み信号送出などの適切な処置が取られる。

PM0/BP0 および PM1/BP1 ピンは、インサーキット・エミュレーション時にはブレークポイントの一致を示す役割も果たす。インサーキット・エミュレーション時には、これらのピンのカウンタ値増加およびカウンタ・オーバフローのいずれの機能も使用できない。ただし、RESET、PM0/BP0、および PM1/BP1 の各ピンを性能モニタリング向けに構成した後に、ブレークポイントの一致を示すようにこれらのピンの構成をハードウェア・デバッガに変更できる。

15.12.3. カウント対象のイベント

性能モニタリング・カウンタでカウントし、CTR0 および CTR1 MSR に記録するように設定できるイベントは、発生型と持続時間型という 2 つのカテゴリに分類される。発生イベントは、そのイベントが発生するたびにカウントされるイベントである。カウンタの値が増加したときにその事実を示すように PM0/BP0 または PM1/BP1 ピンが構成されている場合は、これらのピンはカウンタ値が増加するたびにそのクロックの間アサートされる。1 クロックの間に同一のイベントが 2 回発生する可能性がある場合は、カウンタ値は 2 増加するが、ピンは 1 回しかアサートされないので注意する。

持続時間イベントの場合は、カウンタは当該条件が真である間の合計クロック数をカウントする。カウンタ値が増加するときにその事実を示すように構成すると、PM0/BP0 および PM1/BP1 ピンのいずれかまたは両方が、イベントが持続している間アサートされている。

表 A-10. に、インテル® Pentium® プロセッサの性能モニタリング・カウンタでカウントできるイベントの一覧が示してある。

16

8086 エミュレーション

第 16 章
8086 エミュレーション

16

(Intel386™ プロセッサ以降の) IA-32 プロセッサには、インテル® 8086 プロセッサで動作するようにアセンブル/コンパイルされた新旧のプログラムを実行する次の2つの方法が用意されている。

- 実アドレスモード
- 仮想 8086 モード

図 2-2. に、これら 2 つの動作モードと、保護モードおよびシステム管理モード (SMM) との関係を示す。

プロセッサは、電源投入またはリセット後は実アドレスモードになっている。いくつかの拡張点を除けば、この動作モードは、インテル 8086 プロセッサの実行環境とほぼ同じである。IA-32 プロセッサでは、インテル 8086 プロセッサで動作するようにアセンブル / コンパイルされたプログラムはすべてこのモードで実行される。

インテル・アーキテクチャ・プロセッサは、保護モードでの動作時には、仮想 8086 モードに切り替えれば、8086 プログラムを実行できる。このモードも、いくつかの拡張点を除けば、インテル 8086 プロセッサの実行環境と同じである。仮想 8086 モードでは、8086 プログラムは個別の保護モードタスクとして実行される。したがって、古い 8086 プログラムを、保護モードの利点を活かせる (Microsoft Windows* などの) オペレーティング・システムの下で実行でき、さらに、保護モードの割り込み / 例外処理機能のような保護モード機能を利用できる。保護モードのマルチタスキングでは、同一のプロセッサ上で、複数の仮想 8086 モード・タスク (それぞれ個別の 8086 プログラムを実行する) を仮想 8086 モード以外の他のタスクと一緒に実行できる。

本章では、基本的な実アドレスモード実行環境と、Intel386 プロセッサ以降の IA-32 プロセッサに用意されている仮想 8086 モード実行環境の両環境について説明する。

16.1. 実アドレスモード

IA-32 アーキテクチャの実アドレスモードでは、インテル® 8086 プロセッサ、インテル® 8088 プロセッサ、インテル® 80186 プロセッサ、インテル® 80188 プロセッサ用に記述されたプログラム、またはインテル® 286 プロセッサ、Intel386™ プロセッサ、Intel486™ プロセッサ、インテル® Pentium® プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサの実アドレスモード用に記述されたプログラムを実行する。

実アドレスモードのプロセッサの実行環境は、インテル 8086 プロセッサの実行環境と同じになるように設計されている。8086 プログラムにとっては、実アドレスモードで動作するプロセッサの動きは高速 8086 プロセッサのそれに似ている。このアーキテクチャの主要機能は、『IA-32 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』の第 3 章「基本実行環境」に定義してある。

以降に、8086 用に記述されたプログラムから見た、実アドレスモード実行環境の中心となる機能の要約を示す。

- プロセッサは 1M バイトの物理アドレス空間をサポートする（詳細については、16.1.1. 項「実アドレスモードでのアドレス変換」を参照）。このアドレス空間は、それぞれ最大サイズが 64K バイトであるセグメント単位に分割される。セグメントのベースは 16 ビットのセグメント・セレクタで指定される。このセグメント・セレクタは、ゼロ拡張されてアドレス空間のアドレス 0 からの 20 ビットのオフセットを形成する。セグメント内のオペランドは、セグメントのベースからの 16 ビットのオフセットでアドレス指定される。物理アドレスは、このように、オフセットを 20 ビットのセグメント・ベースに加算して形成される（16.1.1. 項「実アドレスモードでのアドレス変換」を参照）。
- 「ネイティブ 8086 コード」のオペランドはすべて 8 ビットまたは 16 ビットの値である。（オペランド・サイズ・オーバライド・プリフィックスを使用して 32 ビットのオペランドにアクセスできる。）
- 8 つの 16 ビット汎用レジスタ、AX、BX、CX、DX、SP、BP、SI、DI が備えられている。拡張 32 ビット・レジスタ（EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI）は、サイズ・オーバライド操作を明示的に実行してプログラムからアクセスできる。
- 4 つのセグメント・レジスタ、CS、DS、SS、ES が備えられている。（ES レジスタと GS レジスタは、明示的にアクセスすることによってプログラムからアクセスできる。）CS レジスタにはコード・セグメントのセグメント・セレクタを、DS レジスタと ES レジスタにはデータ・セグメントのセグメント・セレクタを、また SS レジスタにはスタック・セグメントのセグメント・セレクタをそれぞれストアする。
- 8086 の 16 ビット命令ポインタ（IP）は EIP レジスタの下位 16 ビットにマッピングされる。このレジスタは 32 ビット・レジスタであり、意図しないアドレス・ラッピングが発生することがあるので注意する。
- 16 ビットの FLAGS レジスタにはステータス・フラグと制御フラグをストアする。（このレジスタは 32 ビットの EFLAGS レジスタの下位 16 ビットにマッピングされる。）
- インテル 8086 プロセッサの命令はすべてサポートされる（16.1.3. 項「実アドレスモードでサポートされる命令」を参照）。

- プロシージャ・コール処理用、および割り込みハンドラと例外ハンドラ呼び出し用として16ビット幅のスタックが1つ備えられている。このスタックは、SSレジスタで識別されるスタック・セグメント内にある。SP（スタックポインタ）レジスタにはスタック・セグメントへのオフセットをストアする。スタックは、スタックポインタの指示先から下方に（下位のセグメント・オフセットに向かって）伸長する。BP（ベースポインタ）レジスタもスタック・セグメントへのオフセットをストアするが、このオフセットはパラメータ・リストのポインタとして使用できる。CALL命令が実行されると、プロセッサは現行の命令ポインタ（EIPレジスタの下位16ビット。farコール時にはさらにCSレジスタの現在値）をスタックにプッシュする。RET命令で開始された戻り時には、プロセッサはセーブしてある命令ポインタをスタックからEIPレジスタ（farリターンの場合はさらにCSレジスタ）にポップする。割り込みハンドラまたは例外ハンドラへの暗黙の呼び出しが実行されたときは、プロセッサはEIP、CS、EFLAGS（下位16ビットのみ）の各レジスタをスタックにプッシュする。IRET命令で開始された、割り込みハンドラまたは例外ハンドラからの戻り時には、プロセッサはセーブしてある命令ポインタとEFLAGSイメージをスタックからEIP、CS、EFLAGSレジスタにポップする。
- 割り込み / 例外処理用として、「割り込みベクタテーブル」または「割り込みテーブル」という割り込み用のテーブルが1つ備えられている（図16-2.を参照）。割り込みテーブル（エントリは4バイト長）は、保護モードの割り込み/例外処理時に、割り込みディスクリプタ・テーブル（IDT、8バイト・エントリ）の代わりに使用される。割り込みベクタ番号と例外ベクタ番号は、割り込みテーブルのエントリへのインデックスになる。各エントリが割り込みまたは例外処理プロシージャへのポインタ（「ベクタ」という）を提供する。

  詳細については、16.1.4.項「割り込みと例外の処理」を参照。Intel386プロセッサ以降のIA-32プロセッサでは、ソフトウェアでLIDT命令を使用して、IDTの配置を変更することができる。
- 実アドレスモードでは、x87 FPUがアクティブでありx87 FPU命令の実行が可能である。実アドレスモードでは、インテル8087とインテル®287マス・コプロセッサで実行されるように記述されたプログラムを修正することなく実行できる。

IA-32アーキテクチャの実アドレスモードでは、次に示すインテル8086プロセッサの実行環境の拡張が使用可能である。286プロセッサおよびインテル8086プロセッサとの下位互換性が必要な場合は、これらの機能は、実アドレスモードで実行されるように記述された新しいプログラムには使用してはならない。

- 2つの追加セグメント・レジスタ（FSおよびGS）が使用可能である。
- IA-32　プロセッサに追加されている整数命令とシステム命令の多くが実アドレスモードで実行できる（16.1.3.項「実アドレスモードでサポートされる命令」を参照）。
- 実アドレスモードのプログラムでは、32 ビットのオペランド・プリフィックスを使用して32ビット形式の命令を実行できる。このプリフィックスによって、実ア

ドレスモードのプログラムはさらにプロセッサの 32 ビット汎用レジスタも使用できる。

- 実アドレスモードのプログラムでは、32 ビットのアドレス・プリフィックスを使用でき、32 ビットのオフセットを扱うことができる。

以降の各項では、実アドレスモードでのアドレスの形成、レジスタ、使用可能な命令、割り込み / 例外処理について説明する。実アドレスモードの I/O については、『IA-32 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』の第 13 章「入出力」を参照。

16.1.1. 実アドレスモードでのアドレス変換

実アドレスモードでは、プロセッサはセグメント・セレクタをディスクリプタ・テーブルへのインデックスとしては解釈せず、8086 プロセッサがそうするように、セグメント・セレクタを使用して直接にリニアアドレスを形成する。その際、セグメント・セレクタを 4 ビット左シフトして 20 ビットのベースアドレスを形成する（図 16-1. を参照）。セグメントへのオフセットがベースアドレスに加算されて、物理アドレス空間に直接にマッピングされるリニアアドレスが作成される。

8086 スタイルのアドレス変換を使用すると、1M バイトを超えるアドレスを指定できる。例えば、セグメント・セレクタ値が FFFFH であって、オフセットが FFFFH である場合、リニア（および物理）アドレスは 10FFEFH（1M バイト +64K バイト）になる。8086 プロセッサでは、20 ビット長までのアドレスしか形成できないので、上位ビットを切り捨てる。したがって、このアドレスは FFEFH に「ラップ」される。しかし、実アドレスモードでの動作時には、プロセッサはそのようなアドレスを切り捨てることなく、物理アドレスとして使用する。（ただし、Intel486™ プロセッサ以降の IA-32 プロセッサについては、実アドレスモードでは A20M# 信号を使用してアドレスライン A20 をマスクでき、それによって疑似的に 8086 プロセッサの 20 ビット・ラップアラウンド動作を実行することに注意する。マルチプロセッサ・ベースのシステムでは、A20M# に基づくアドレスラップ動作が確実に正しく取り扱われるように注意する必要がある。

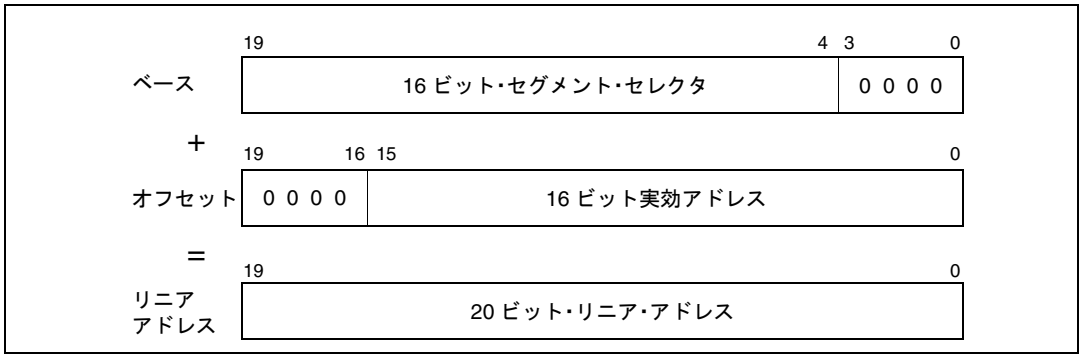


図 16-1. 実アドレスモードでのアドレス変換

Intel386™ プロセッサ以降の IA-32 プロセッサは、アドレス・オーバライド・プリフィックスを使用して 32 ビットのオフセットを生成できる。ただし、実アドレスモードでは、32 ビットのオフセットの値は FFFFH を超えてはならない。超えた場合は例外が発生する。

インテル® 286 プロセッサの実アドレスモードと完全な互換性を保つため、0 ～ FFFFH の範囲を超える 32 ビットのオフセットが生成された場合は、疑似保護フォルト（割り込み 12 または 13）が発生する。

16.1.2. 実アドレスモードでサポートされるレジスタ

実アドレスモードで使用可能なレジスタのセットには、8086 プロセッサ用として定義されているレジスタと、8086 プロセッサより後の IA-32 プロセッサに導入された新しいレジスタ、すなわち、FS および GS セグメント・レジスタ、デバッグレジスタ、制御レジスタ、浮動小数点ユニットのレジスタなどが含まれる。32 ビットのオペランド・プリフィックスを使用すると、実アドレスモードのプログラムで 32 ビットの汎用レジスタ（EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI）を使用できる。

16.1.3. 実アドレスモードでサポートされる命令

次に示す命令は、8086 プロセッサの命令セットを構成する。インテル® 286 プロセッサおよびインテル® 8086 プロセッサとの下位互換性が必要な場合は、実アドレスモードで実行される新しいプログラムを作成する際に、これらの命令だけを使用するようにしなければならない。

- 汎用レジスタ間、セグメント・レジスタ間、メモリと汎用レジスタ間でオペランドを移動する移動（Move）命令
- 交換（XCHG）命令

- セグメント・レジスタ・ロード命令 : LDS と LES
- 演算命令 : ADD、ADC、SUB、SBB、MUL、IMUL、DIV、IDIV、INC、DEC、CMP、NEG
- 論理命令 : AND、OR、XOR、NOT
- 10 進命令 : DAA、DAS、AAA、AAS、AAM、AAD
- スタック命令 : PUSH と（汎用レジスタおよびセグメント・レジスタへの）POP
- タイプ変換命令 : CWD、CDQ、CBW、CWDE
- シフトとローテート命令 : SAL、SHL、SHR、SAR、ROL、ROR、RCL、RCR
- TEST 命令
- 制御命令 : JMP、J*cc*、CALL、RET、LOOP、LOOPE、LOOPNE
- 割り込み命令 : INT *n*、INTO、IRET
- EFLAGS 制御命令 : STC、CLC、CMC、CLD、STD、LAHF、SAHF、PUSHF、POPF
- I/O 命令 : IN、INS、OUT、OUTS
- 実効アドレスロード（LEA）命令とアドレス変換（XLATB）命令
- LOCK プリフィックス
- リピート・プリフィックス : REP、REPE、REPZ、REPNE、REPNZ
- プロセッサ停止命令 : HLT
- ノー・オペレーション命令 : NOP

次に示す命令は、より新しい IA-32 プロセッサに（一部は 286 プロセッサで、その他は Intel386™ プロセッサで）追加されたものであり、インテル® 8086 プロセッサとの下位互換性が必要でない場合に実アドレスモードで実行できる。

- 制御レジスタとデバッグレジスタを操作対象とする移動（MOV）命令
- セグメント・レジスタ・ロード命令 : LSS、LFS、LGS
- 一般化された乗算命令と即値データの乗算
- 即値カウントによるシフトとローテート
- スタック命令 : PUSHA、PUSHAD、POPA、POPAD、即値データの PUSH
- 符号拡張付き移動命令 : MOVSX と MOVZX
- ロング・ディスプレースメント J*cc* 命令
- 交換命令 : CMPXCHG、CMPXCHG8B、XADD
- ストリング命令 : MOVS、CMPS、SCAS、LODS、STOS
- ビットテストとビットスキャン命令 : BT、BTS、BTR、BTC、BSF、BSR、条件付きバイトセット（SET*cc*）命令、バイトスワップ（BSWAP）命令
- ダブルシフト命令 : SHLD と SHRD

- EFLAGS 制御命令 : PUSHF と POPF
- 制御命令 : ENTER と LEAVE
- BOUND 命令
- CPU 識別命令 : CPUID
- システム命令 : CLTS、INVD、WINVD、INVLPG、LGDT、SGDT、LIDT、SIDT、LMSW、SMSW、RDMSR、WRMSR、RDTSC、RDPMC

実アドレスモードで、上記以外の（2つのリストに記載されていない）どの IA-32 アーキテクチャ命令を実行しても、無効オペコード例外（#UD）が発生する。

16.1.4. 割り込みと例外の処理

実アドレスモードで動作するときは、ソフトウェアは保護モードで提供されるものとは別の割り込みと例外の処理機能を用意しなければならない。プロセッサがまだ実アドレスモードにあるプロセッサ初期化の初期段階でも、簡単な実アドレスモードの割り込み / 例外処理機能を提供してプロセッサ動作の信頼性を保証するか、または初期化コードが割り込みも例外も生成しないように保証しなければならない。

IA-32 プロセッサは、実アドレスモードでも、割り込みと例外を保護モードの場合と同様に処理する。プロセッサは、割り込みを受け取るか、または例外を生成すると、その割り込みまたは例外のベクタ番号を割り込みテーブルへのインデックスとして使用する（保護モードでは、割り込みテーブルは**割り込みディスクリプタ・テーブル**（IDT: Interrupt Descriptor Table）というが、実アドレスモードでは、このテーブルは一般に**割り込みベクタテーブル**または単に**割り込みテーブル**という）。割り込みベクタテーブルのエントリは、割り込みまたは例外処理プロシージャを指すポインタを提供する（このポインタには、コード・セグメントのセグメント・セレクタとそのセグメントへの 16 ビットのオフセットが含まれる）。プロセッサは、次に示す処置を行って、選択したハンドラに暗黙の呼び出しをかける。

1. CS レジスタと EIP レジスタの現在の値をスタックにプッシュする。（EIP レジスタについては、下位 16 ビットだけがプッシュされる。）
2. EFLAGS レジスタの下位 16 ビットをスタックにプッシュする。
3. EFLAGS レジスタの IF フラグをクリアして、割り込みをディスエーブルにする。
4. EFLAGS レジスタの TF、RC、AC フラグをクリアする。
5. プログラムの制御を割り込みベクタテーブルで指定された位置に移行する。

ハンドラ・プロシージャの最後の IRET 命令は、これらのステップを逆にたどって、割り込みをかけられたプログラムにプログラムの制御を戻す。実アドレスモードでは、例外はエラーコードを返さない。

割り込みベクタテーブルは、4 バイトのエントリの配列である（図 16-2. を参照）。各エントリには、ハンドラ・プロシージャへの far ポインタが入っており、このポインタはセグメント・セレクタとオフセットで構成される。プロセッサは、割り込みベクタまたは例外ベクタを 4 倍して割り込みテーブルへのオフセットを得る。リセット後は、割り込みベクタテーブルのベースは物理アドレス 0 に置かれ、そのリミットは 3FFH に設定される。インテル® 8086 プロセッサでは、割り込みベクタテーブルのベースアドレスとリミットは変更できない。インテル 8086 プロセッサより後の IA-32 プロセッサでは、割り込みベクタテーブルのベースアドレスとリミットは IDTR レジスタにストアされており、LIDT 命令を使用して変更できる。

（インテル 8086 プロセッサとの下位互換性を保つためには、割り込みベクタテーブルのデフォルトのベースアドレスとリミットは変更してはならない。）

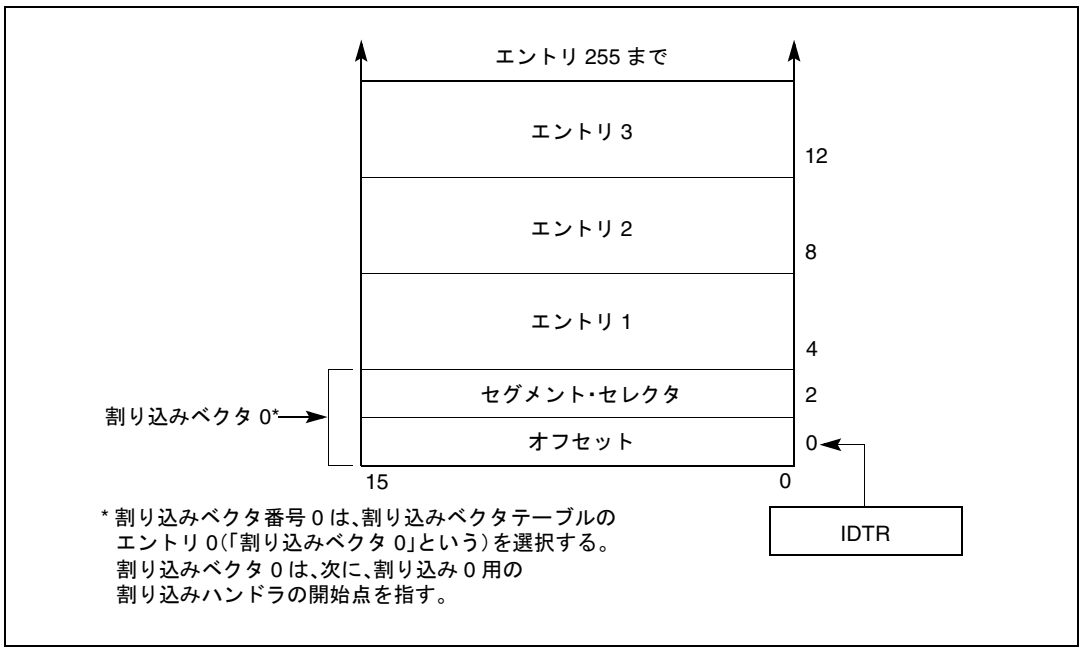


図 16-2. 実アドレスモードでの割り込みベクタテーブル

表 16-1. に、実アドレスモード、仮想 8086 モード、インテル 8086 プロセッサで生成される割り込みベクタと例外ベクタを示す。各例外条件については、第 5 章「割り込みと例外の処理」を参照。

16.2. 仮想 8086 モード

仮想 8086 モードは、実際には保護モードで実行される特殊なタスクである。オペレーティング・システムまたはエグゼクティブが仮想 8086 モードのタスクに切り替わると、プロセッサはインテル® 8086 プロセッサをエミュレートする。8086 エミュレーション状態にあるときのプロセッサの実行環境は、拡張機能を含めて、16.1.節「実アドレスモード」での実アドレスモードに関する説明と同じである。これら 2 つのモード間の主な相違点は、仮想 8086 モードでは、8086 エミュレータが（保護モードの割り込み / 例外処理機能やページング機能などの）いくつかの保護モードサービスを使用することである。

実アドレスモードの場合と同様に、インテル 8086 プロセッサで実行されるようにアセンブル / コンパイルされたプログラムは、新旧を問わず仮想 8086 モードのタスクで実行される。さらに、プロセッサのマルチタスキング機能を使用して、複数の 8086 プログラムを仮想 8086 モードのタスクとして通常の保護モードタスクと並行して実行できる。

表 16-1. 実アドレスモードの例外と割り込み

ベクタ番号	説明	実アドレスモード	仮想 8086 モード	インテル 8086 プロセッサ
0	除算エラー（#DE）	あり	あり	あり
1	デバッグ例外（#DB）	あり	あり	なし
2	NMI 割り込み	あり	あり	あり
3	ブレークポイント（#BP）	あり	あり	あり
4	オーバフロー（#OF）	あり	あり	あり
5	BOUND 範囲外（#BR）	あり	あり	予約済み
6	無効オペコード（#UD）	あり	あり	予約済み
7	デバイス使用不可能（#NM）	あり	あり	予約済み
8	ダブルフォルト（#DF）	あり	あり	予約済み
9	（インテルがすでに予約済み、使用不可）	予約済み	予約済み	予約済み
10	無効 TSS（#TS）	予約済み	あり	予約済み
11	セグメント不在（#NP）	予約済み	あり	予約済み
12	スタックフォルト（#SS）	あり	あり	予約済み
13	一般保護（#GP）*	あり	あり	予約済み
14	ページフォルト（#PF）	予約済み	あり	予約済み
15	（インテル社がすでに予約済み、使用不可）	予約済み	予約済み	予約済み
16	浮動小数点エラー（#MF）	あり	あり	予約済み
17	アライメント・チェック（#AC）	予約済み	あり	予約済み
18	マシンチェック（#MC）	あり	あり	予約済み

表 16-1. 実アドレスモードの例外と割り込み（続き）

ベクタ番号	説明	実アドレスモード	仮想 8086 モード	インテル 8086 プロセッサ
19-31	（インテルがすでに予約済み、使用不可）	予約済み	予約済み	予約済み
32-255	ユーザ定義割り込み	あり	あり	あり

注記 :

* 実アドレスモードでは、ベクタ 13 はセグメント・オーバラン例外になる。保護モードと仮想 8086 モードでは、この例外は、仮想 8086 モードから仮想 8086 モニタへのトラップを含めて、すべての一般保護エラー条件をカバーする。

16.2.1. 仮想 8086 モードの有効化

EFLAGS レジスタの VM（仮想マシン）フラグがセットされているときは、プロセッサは仮想 8086 モードで動作する。このフラグは、プロセッサが新しい保護モードのタスクに切り替わるときか、または IRET 命令を通じて仮想 8086 モードを再開するときだけセットできる。

システム・ソフトウェアは、（例えば、POPFD 命令を使用して）EFLAGS レジスタの VM フラグの状態を直接変更できない。代わりに、TSS にストアされた EFLAGS レジスタのイメージ内のフラグを変更するか、割り込みハンドラ・プロシージャまたは例外ハンドラ・プロシージャの呼び出し後にスタック上でフラグを変更する。例えば、ソフトウェアは、最初に仮想 8086 タスクを作成するときは、TSS 内の EFLAGS イメージの VM フラグをセットする。

プロセッサは、次の 3 つの一般的条件下で VM フラグをテストする。

- 8086 スタイルのアドレス変換を使用するかどうかを決定するため、セグメント・レジスタをロードする時。
- 仮想 8086 モードでサポートされていない命令、および IOPL にセンシティブな命令を判断するために、命令をデコードする時。
- ページアクセスでの特権命令をチェックする時、またはその他の許可チェックを実行する時。（仮想 8086 モードは常に CPL 3 で実行される。）

16.2.2. 仮想 8086 タスクの構造

仮想 8086 モードのタスクは、次の項目で構成される。

- タスクの 32 ビット TSS
- 8086 プログラム
- 仮想 8086 モニタ
- 8086 オペレーティング・システム・サービス

16 ビットの TSS は、VM フラグがある EFLAGS レジスタの最上位ワードをロードしないので、新しいタスクの TSS は、16 ビットの TSS ではなく、32 ビットの TSS でなければならない。仮想 8086 モード時に例外の処理に使用される TSS、スタック、データ、コードもやはりすべて 32 ビット・セグメントでなければならない。

プロセッサは、8086 プログラムを実行する際は仮想 8086 モードになり、仮想 8086 モニタを実行する際は保護モードに戻る。

仮想 8086 モニタは、CPL 0 で動作する 32 ビットの保護モード・コード・モジュールである。このモニタは、初期化プロシージャ、割り込み / 例外処理プロシージャ、パーソナル・コンピュータまたはその他の 8086 ベースのプラットフォームをエミュレートする I/O エミュレーション・プロシージャで構成されている。このモニタは、一般的には、やはり CPL 0 で動作する保護モードの一般保護（#GP）例外ハンドラの一部であるか、またはそれと密接に関連している。すべての保護モード・コード・モジュールの場合と同様に、仮想 8086 モニタのコード・セグメント・ディスクリプタは GDT またはタスクの LDT 内になければならない。アドレス空間の最初の 1M バイト内にある IDT や 8086 プログラムのその他の部分を調べられるように、仮想 8086 モニタにはデータ・セグメント・ディスクリプタも必要な場合がある。10FFEFH より上のリニアアドレスは、モニタ、オペレーティング・システム、その他のシステム・ソフトウェアで使用可能である。

8086 オペレーティング・システムのサービスは、8086 プログラムの呼び出し対象となるカーネル・プロシージャおよびオペレーティング・システム・プロシージャのいずれかまたは両方で構成される。これらのサービスは、次の 2 つの方法のいずれかで実現できる。

- 8086 プログラムに組み込む。この方法は、次のいずれかの場合に望ましい。
  - 8086 プログラム・コードで 8086 オペレーティング・システム・サービスを修正する場合。
  - 8086 オペレーティング・システム・サービスをメイン・オペレーティング・システムまたはエグゼクティブにマージするための十分な開発時間がない場合。

- 仮想 8086 モニタ内にとり込む、または同モニタ内でエミュレートする。この方法は、次のいずれかの場合に望ましい。
  - 複数の仮想 8086 タクス間で、8086 オペレーティング・システム・プロシージャを簡単にコーディネートできる場合。
  - 複数の仮想 8086 タスクに 8086 オペレーティング・システム・プロシージャ・コードをコピーしないことによって、メモリを節約できる場合。
  - メイン・オペレーティング・システムまたはエグゼクティブへの呼び出しによって、8086 オペレーティング・システム・プロシージャを簡単にエミュレートできる場合。

8086 オペレーティング・システム・サービスをサポートする際に選択した方法によっては、仮想 8086 モード・タスクが異なる 8086 オペレーティング・システム・サービスを使用する結果となることがある。

16.2.3. 仮想 8086 タスクのページング

仮想 8086 モードで実行されるプログラムが使用できるのは 20 ビットのリニアアドレスだけであるが、プロセッサはそれらのアドレスを物理アドレス空間にマッピングする前に 32 ビットのリニアアドレスに変換する。ページングを使用する場合は、仮想 8086 モードで実行されるプログラムの 8086 アドレス空間は、物理アドレス空間のページセットにページングしたり置くことができる。ページングを使用した場合は、プロセッサ上で実行される任意のタスクの場合と同じように、ページングは仮想 8086 モードで実行されるプログラムから透過的になる。

ページングは、単一の仮想 8086 モード・タスクには必要ではなく、次の状況で必要または効果的なものである。

- 複数の仮想 8086 モード・タスクの実行時。この場合は、ページングによって、各仮想 8086 モード・タスクのリニアアドレス空間の下位 1M バイトを異なる物理アドレス位置にマッピングできる。
- 1M バイトで発生する 8086 のアドレス・ラップアラウンドのエミュレーション時。8086 スタイルのアドレス変換を使用するときは、1M バイトを超えるアドレスを指定できる。そのようなアドレスは、インテル 8086 プロセッサでは自動的にラップアラウンドする（16.1.1. 項「実アドレスモードでのアドレス変換」を参照）。アドレス・ラップアラウンドに依存している場合は、どの 8086 プログラムでも、100000H ～ 110000H の範囲のリニアアドレスと 0 ～ 10000H の範囲のリニアアドレスを同じ物理アドレスにマッピングすることによって、仮想 8086 モード・タスクで同じ効果を実現できる。
- それぞれ異なる 8086 モード・タスクとして実行される複数の 8086 プログラムに共通の 8086 オペレーティング・システム・サービスまたは ROM コードの共有時。
- メモリマップド I/O デバイスへの参照のリダイレクションまたはトラップ時。

16.2.4. 仮想 8086 タスク内での保護

8086プログラムのセグメント間では、保護は行われない。次のいずれかの手法を使用して、仮想 8086 モード・タスク内で実行されるシステム・ソフトウェアを 8086 プログラムから保護できる。

- 各タスクのリニアアドレス空間の最初の "1M バイト +64K バイト " を 8086 プログラム用に予約する。8086 プロセッサ・タスクは、この範囲外のアドレスを生成できない。
- ページ・テーブル・エントリの U/S フラグを使用して、仮想 8086 モード・タスク空間内の仮想 8086 モニタやその他のシステム・ソフトウェアを保護する。プロセッサが仮想 8086 モードのときは、CPL は 3 である。したがって、8086 プロセッサ・プログラムにはユーザ特権しかない。仮想 8086 モニタのページがスーパーバイザ特権を持っている場合は、8086 プログラムからそれらのページにはアクセスできない。

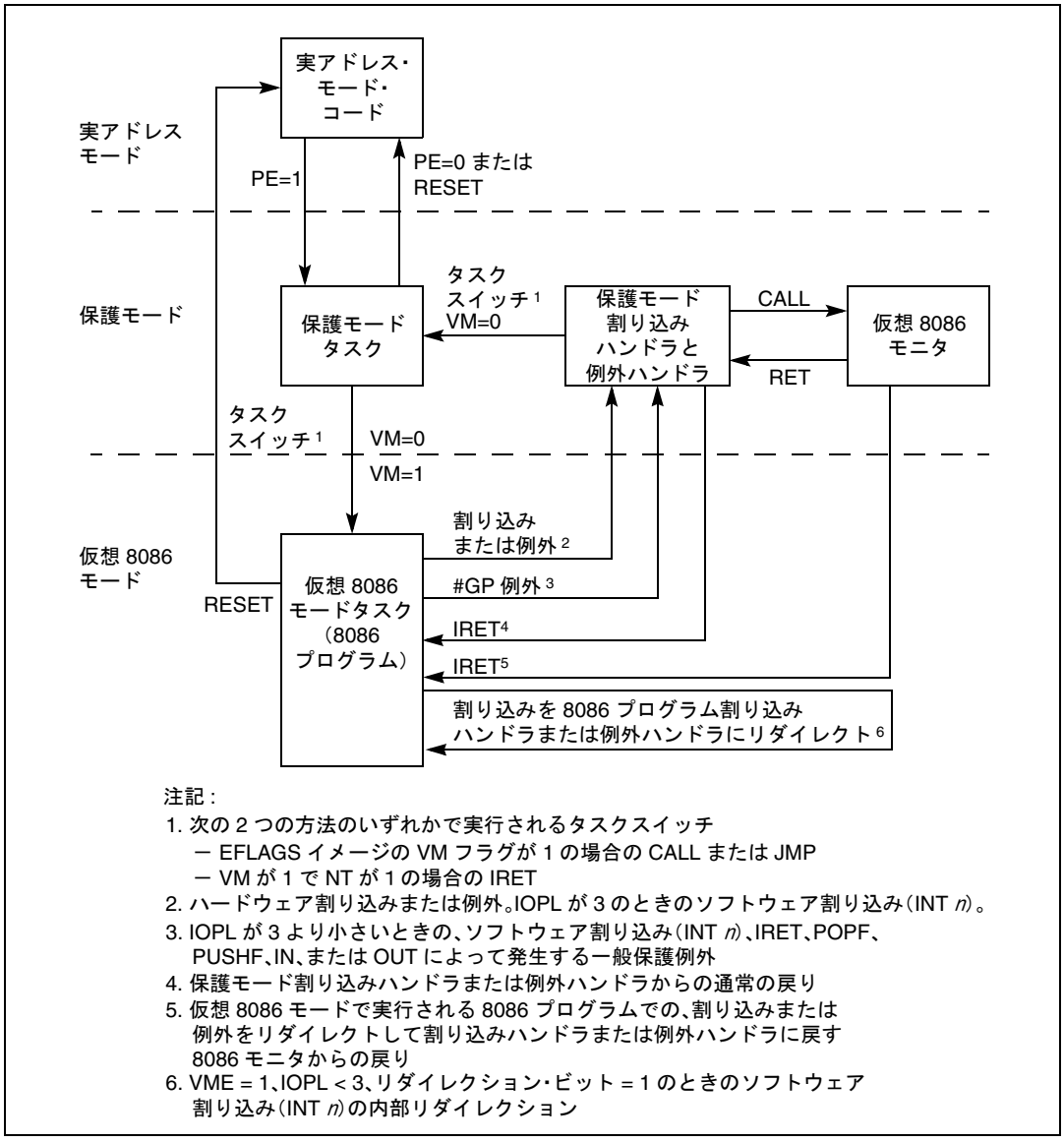
16.2.5. 仮想 8086 モードへの移行

図 16-3. に、仮想 8086 モードに移行する方法と同モードを終了する方法の概要を示す。プロセッサは、次のいずれかの状況が発生した場合に仮想 8086 モードに切り替わる。

- タスクの TSS にストアされている EFLAGS レジスタのイメージの VM フラグが 1 にセットされているときのタスクスイッチ。この状況では、タスクスイッチは次の 2 つの方法のいずれかで開始できる。
  - CALL 命令または JMP 命令
  - EFLAGS イメージの NT フラグが 1 にセットされている場合の IRET 命令
- スタック上の EFLAGS レジスタイメージの VM フラグが 1 にセットされているときの、保護モード割り込みハンドラまたは例外ハンドラからの戻り

タスクスイッチを使用して仮想 8086 モードに移行するときは、仮想 8086 モード・タスクの TSS は 32 ビットの TSS でなければならない（新しい TSS が 16 ビットの TSS である場合は、EFLAGS レジスタの上位ワードが TSS 内になく、プロセッサは EFLAGS レジスタにロードするとき VM フラグをクリアしてしまう）。プロセッサは、新しい TSS 内のセグメント・レジスタのイメージからセグメント・レジスタにロードする前に VM フラグを更新する。新しい VM フラグの設定によって、プロセッサがセグメント・レジスタの内容を 8086 スタイルのセグメント・セレクタ、保護モードのセグメント・セレクタのいずれとして解釈するかが決まる。VM フラグがセットされているときは、セグメント・レジスタは TSS からロードされ、8086 スタイルのアドレス変換を使用してベースアドレスが形成される。

割り込みハンドラまたは例外ハンドラからの戻り時の仮想 8086 モードへの移行については、16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照。



注記 :

1. 次の 2 つの方法のいずれかで実行されるタスクスイッチ
   － EFLAGS イメージの VM フラグが 1 の場合の CALL または JMP
   － VM が 1 で NT が 1 の場合の IRET
2. ハードウェア割り込みまたは例外。IOPL が 3 のときのソフトウェア割り込み（INT *n*）。
3. IOPL が 3 より小さいときの、ソフトウェア割り込み（INT *n*）、IRET、POPF、PUSHF、IN、または OUT によって発生する一般保護例外
4. 保護モード割り込みハンドラまたは例外ハンドラからの通常の戻り
5. 仮想 8086 モードで実行される 8086 プログラムでの、割り込みまたは例外をリダイレクトして割り込みハンドラまたは例外ハンドラに戻す 8086 モニタからの戻り
6. VME = 1、IOPL < 3、リダイレクション・ビット = 1 のときのソフトウェア割り込み（INT *n*）の内部リダイレクション

図 16-3. 仮想 8086 モードへの移行と終了

16.2.6. 仮想 8086 モードの終了

プロセッサは、割り込みまたは例外を介してのみ仮想 8086 モードを終了できる。以降に、割り込みまたは例外の結果としてプロセッサが仮想 8086 モードを終了する状況を示す（図 16-3. を参照）。

- 仮想 8086 アプリケーションの実行が中断されたことを通知するために発生したハードウェア割り込みをプロセッサが処理する場合。このようなハードウェア割り込みは、タイマまたはその他の外部メカニズムによって発生できる。ハードウェア割り込みを受け取ると、プロセッサは保護モードになり、保護モードの IDT 内のタスクゲートを通じて、あるいはタスクスイッチを開始するハンドラを指すトラップゲートまたは割り込みゲートを通じて、保護モード（または別の仮想 8086 モード）のタスクに切り替わる。仮想 8086 タスクから別のタスクへのタスクスイッチによって、新しいタスクの TSS から EFLAGS レジスタにロードされる。新しい EFLAGS の VM フラグの値によって、新しいタスクが仮想 8086 モードで実行されるかどうかが決まる。
- プロセッサが、仮想 8086 タスクを実行するコードによって発生した例外を処理するか、または仮想 8086 タスクに「属する」ハードウェア割り込みを処理する場合。この場合、プロセッサは、保護モードになり、（通常は割り込みゲートまたはトラップゲートを通じて）保護モードの IDT と、それぞれ保護モード例外ハンドラまたは保護モード割り込みハンドラを通じて例外またはハードウェア割り込みを処理する。プロセッサは、仮想 8086 タスクのコンテキスト内で例外または割り込みを処理し、ハンドラ・プロシージャからの戻り時に仮想 8086 モードに戻る。プロセッサは、さらに、タスクスイッチを実行し、別のタスクのコンテキスト内で例外または割り込みを処理することもある。
- プロセッサが仮想 8086 タスク内で実行される（MS-DOS* オペレーティング・システム・ルーチンを呼び出しするソフトウェア割り込みなどの）コードによって発生したソフトウェア割り込みを処理する場合。プロセッサには、そのようなソフトウェア割り込みの処理方法がいくつかある。それらの手段については、16.3.3. 項「クラス 3 ― 仮想 8086 モードでのソフトウェア割り込みの処理」で説明している。それらの手段には、ほとんど、主として一般保護（#GP）例外によるプロセッサの保護モードへの移行が関与する。保護モードでは、プロセッサは割り込みを仮想 8086 モニタに送り、そこで処理するか、または割り込みを仮想 8086 モード・タスク内で実行されているアプリケーション・プログラムにリダイレクトして戻し、そこで処理できる。あるいはそれらの両処置を併用できる。

  仮想モード拡張（制御レジスタ CR4 の VME フラグでイネーブルにする）を組み入れている IA-32 プロセッサは、仮想 8086 モードを終了させないで、ソフトウェアが生成した割り込みをリダイレクションによってプログラムの割り込みハンドラに返すことができる。このメカニズムの詳細については、16.3.3.4. 項「方法 5: ソフトウェア割り込みの処理」を参照。

- RESET ピンまたは INIT ピンをアサートして起動するハードウェア・リセットは特殊な割り込みである。プロセッサが仮想 8086 モードのときに RESET 信号または INIT 信号が発せられると、プロセッサは仮想 8086 モードを終了し、実アドレスモードに移行する。
- 仮想 8086 モードで HLT 命令が実行されると、一般保護（GP#）フォルトが発生し、保護モードハンドラは一般にそれを仮想 8086 モニタに送る。そこで、仮想 8086 モニタは、仮想 8086 モニタへの制御の移行が HLT 命令の実行の結果として生じたことを確認した後に、正しい実行シーケンスを決定する。

仮想 8086 モードを終了して仮想 8086 モードで発生した割り込みまたは例外を処理する場合の詳細については、16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照。

16.2.7. センシティブな命令

IA-32 プロセッサが仮想 8086 モードで動作しているときは、CLI、STI、PUSHF、POPF、INT *n*、IRET の各命令は IOPL に対してセンシティブである。一方、IN、INS、OUT、OUTS 命令は、保護モードでは IOPL に対してセンシティブであるが、仮想 8086 モードではセンシティブではない。

仮想 8086 モードで動作中は、CPL は常に 3 である。IOPL が 3 より小さい場合は、上記の IOPL に依存する命令を使用しようとすると、一般保護（#GP）例外がトリガされる。これらの命令は IOPL に依存するため、それらの命令の影響を受ける機能をエミュレートする機会が仮想 8086 モニタに与えられる。

16.2.8. 仮想 8086 モードの入出力

マルチタスキング・システム用に記述された 8086 プログラムの多くは、I/O ポートに直接にアクセスする。この方法はマルチタスキング環境では問題を生じる可能性がある。複数のプログラムが同一のポートにアクセスした場合は、それらのプログラム間に相互干渉が生じることがある。大部分のマルチタスキング・システムでは、アプリケーション・プログラムはオペレーティング・システムを通じて I/O ポートにアクセスする必要がある。その結果、制御が単純になり、集中化される。

プロセッサでは、環境との互換性があり、8086 プログラムに透過的な I/O を作成するための I/O の保護方法が提供される。設計者は、次に示す可能なアプローチを自由に組み合わせて I/O ポートの保護がはかれる。

- I/O アドレス空間を保護し、I/O を直接に実行しようとする試みすべてに対して例外を発生させる。
- 8086 プログラムに I/O を直接に実行させる。
- 特定の I/O ポートへのアクセスに対して例外を発生させる。

- 特定のメモリマップド I/O ポートへのアクセスに対して例外を発生させる。

I/O ポートへのアクセスを制御する方法は、I/O が I/O ポートマップド型であるか、メモリマップド型であるかによって異なる。

16.2.8.1. I/O ポートマップド I/O

TSS の I/O 許可ビットマップを使用して、特定の I/O ポートアドレスへのアクセスが試みられた時点で例外を発生させられる。各仮想 8086 モード・タスクの I/O 許可ビットマップによって、どの I/O アドレスがそのタスクに対して例外を生成するかが決まる。各タスクがそれぞれ異なる I/O 許可ビットマップを持てるので、タスクごとに、例外を生成するアドレスを変えられる。この点で保護モードとは異なり、保護モードでは、CPL が IOPL より小さいか等しい場合は、I/O 許可ビットマップを『IA-32 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』の第 13 章「入出力」を参照。

16.2.8.2. メモリマップド I/O

メモリマップド I/O を使用するシステムでは、プロセッサのページング機能を使用して I/O ポートへのアクセス試行に対して例外を発生させることができる。仮想 8086 モニタは、ページングを使用して、次に示す方法でメモリマップド I/O を制御できる。

- I/O を実行する必要がある各タスクのリニアアドレス空間の一部を、I/O ポートが置かれている物理アドレス空間にマッピングする。I/O ポートをそれぞれ（異なるページの）異なるアドレスに置くことによって、ページング・メカニズムでタスク間の分離を強化できる。
- リニアアドレス空間の一部を存在しないページにマッピングする。それによって、タスクがそれらのページに I/O を実行しようとするたびに例外が発生する。したがって、システム・ソフトウェアは、I/O 操作が実行されようとしていると解釈できる。

I/O 空間のソフトウェア・エミュレーションには、特定の状態の下ではオペレーティング・システムの介入が多くなりすぎることがある。そのような場合は、最初の I/O アクセスだけに対して例外を発生できる。その後、システム・ソフトウェアは、プログラムに一時的に I/O を排他的に制御させてよいか、I/O 空間の保護を解除してよいか、さらにプログラムを最高速度で実行させてよいかを決定する。

16.2.8.3. 特殊な I/O バッファ

ページ・マッピングを使用すると、インテリジェント・コントローラのバッファ（例えば、ビット・マップド・フレーム・バッファ）もエミュレートできる。そのようなバッファのリニア空間は、各仮想 8086 モード・タスクごとに異なる物理空間にマッピング可能である。したがって、仮想 8086 モニタは、どの仮想バッファを物理アドレス空間内の実際のバッファにコピーするかを制御できる。

16.3. 仮想 8086 モードでの割り込み / 例外処理

プロセッサは、仮想 8086 モードで動作中に割り込みを受け取るか、または例外条件を検出すると、保護モードまたは実アドレスモードの場合と全く同様に、割り込みハンドラまたは例外ハンドラを呼び出す。呼び出される割り込みハンドラまたは例外ハンドラ、およびハンドラの呼び出しに使用されるメカニズムは、検出された例外または発生した割り込みのクラスと、各種のシステムフラグおよびフィールドの状態によって決まる。

仮想 8086 モードでは、割り込みと例外は処理の目的上次の 3 つのクラスに分けられる。

- クラス 1 － NMI 割り込みとプロセッサの外部割り込み伝達ピンに送られるハードウェア割り込みを含むすべてのプロセッサ生成例外とすべてのハードウェア割り込み。クラス 1 の例外と割り込みはすべて、保護モードの例外ハンドラと割り込みハンドラによって処理される。
- クラス 2 － 仮想モード拡張がイネーブルにされているときのマスク可能ハードウェア割り込みの特殊ケース（5.3.2. 項「マスク可能ハードウェア割り込み」を参照）。
- クラス 3 － すべてのソフトウェア生成割り込み、つまり INT *n* 命令で生成される割り込み[1]。

クラス 2 と 3 の割り込みの処理にプロセッサが使用する方法は、次に示すフラグとフィールドの設定によって決まる。

- IOPL フィールド（EFLAGS レジスタのビット 12 と 13）－ プロセッサが仮想 8086 モードのとき、クラス 3 のソフトウェア割り込みをどのように処理するかを制御する（2.3. 節「EFLAGS レジスタのシステムフラグとフィールド」を参照）。このフィールドは、VME フラグがセットされているときは、EFLAGS レジスタの VIF フラグと VIP フラグの有効化も制御する。VIF フラグと VIP フラグは、クラス 2 のマスク可能ハードウェア割り込みの処理の支援用として用意されている。

1. INT 3 命令は特殊ケースである（『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」の INT *n* 命令についての説明を参照）。

- VME フラグ（制御レジスタ CR4 のビット 0）ー このフラグがセットされると、プロセッサの仮想モード拡張がイネーブルになる（2.5. 節「制御レジスタ」を参照）。
- ソフトウェア割り込みリダイレクション・ビット・マップ（TSS の 32 バイト、図 16-5. を参照）ー 仮想 8086 モードで発生したときに、クラス 3 のソフトウェア割り込みをどのように処理するかを示す 256 個のフラグが含まれる。ソフトウェア割り込みは、その時点で実行されている 8086 プログラムの割り込みハンドラと例外ハンドラにも、保護モードの割り込みハンドラと例外ハンドラにもダイレクトできる。
- EFLAGS レジスタの仮想割り込みフラグ（VIF）と仮想割り込みペンディング・フラグ（VIP）ー クラス 2 のマスク可能ハードウェア割り込み処理用の**仮想割り込みサポート**を提供する（16.3.2. 項「クラス 2 ー 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理」を参照）。

注記

仮想モード拡張をサポートする IA-32 プロセッサでは、VME フラグ、ソフトウェア割り込みリダイレクション・ビット・マップ、VIF フラグおよび VIP フラグしか使用できない。これらの拡張は、インテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。

以降の各項で、上で説明した 2 つのクラスの割り込みに対してプロセッサが行う処置と、割り込みハンドラと例外ハンドラが行える処置について説明する。これらの項では、考えられる 3 つのタイプの割り込みハンドラと例外ハンドラについて説明する。

- 保護モード用割り込みハンドラと例外ハンドラ ー これらは、プロセッサが保護モード IDT を通じて呼び出しする標準ハンドラである。
- 仮想 8086 モニタ用割り込みハンドラと例外ハンドラ ー これらのハンドラは、仮想 8086 モニタに常駐し、一般に保護モードの一般保護例外ハンドラにダイレクトされる一般保護例外（#GP、割り込み 13）を通じてアクセスされる。
- 8086 プログラム用割り込みハンドラと例外ハンドラ ー これらのハンドラは、仮想 8086 モードで実行中の 8086 プログラムの一部である。

以降の各項で、選択されたクラスと割り込み / 例外処理方法によって、これらのハンドラがどのように使用されるかを説明する。

16.3.1. クラス 1 ー 仮想 8086 モードでのハードウェア割り込み / 例外の処理

仮想 8086 モードでは、インテル® Pentium® プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサは、Intel486™ プロセッサおよびIntel386™ プロセッサの場合と同様にハードウェア割り込みと例外を処理する。これらのプロセッサは、IDT 内の、割り込みベクタまたは例外ベクタの指示先の保護モード割り込みハンドラまたは例外ハンドラを呼び出す。この場合、IDT のエントリには、32 ビットのトラップゲート、割り込みゲート、またはタスクゲートが含まれていなければならない。以降の各項で、保護モードハンドラが呼び出された後で可能な仮想 8086 モードでの各種の割り込みまたは例外処理方法について説明する。

仮想 8086 モードでのマスク可能ハードウェア割り込み処理用の仮想割り込みメカニズムについては、16.3.2. 項「クラス 2 ー 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理」を参照。このメカニズムが使用できないか、またはイネーブルにされていないときは、以降の各項で説明するように、マスク可能ハードウェア割り込みは例外と同様に処理される。

16.3.1.1. 保護モードのトラップゲートまたは割り込みゲートを通じた割り込み / 例外の処理

割り込みベクタまたは例外ベクタがIDT内の32ビットのトラップゲートまたは割り込みゲートを指しているときは、そのゲートは非コンフォーミングの、特権レベル 0 のコード・セグメントを指さなければならない。このコード・セグメントにアクセスするときは、プロセッサは次の手順を実行する。

1. 32 ビット保護モードおよび特権レベル 0 に切り替える。
2. プロセッサの状態を特権レベル 0 のスタックにセーブする。EIP、CS、EFLAGS、ESP、SS、ES、DS、FS、GS レジスタの各状態がセーブされる（図 16-4. を参照）。
3. セグメント・レジスタをクリアする。DS、ES、FS、GS レジスタをスタックにセーブし、次にそれらのレジスタをクリアすると、レジスタがストアしているタイプ・セグメント・セレクタ（保護モードまたは 8086 スタイル）に関係なく、割り込みハンドラまたは例外ハンドがそれらのレジスタを安全にセーブおよびリストアすることが可能になる。割り込みハンドラと例外ハンドラは、保護モードタスクと仮想 8086 モード・タスクのどちらのコンテキスト内でも呼び出しでき、任意のタスクに対するレジスタのセーブとリストアにも同じコード・シーケンスを使用できる。IRET 命令の実行前にこれらのレジスタをクリアしても、割り込みハンドラ内でトラップは発生しない。割り込みプロシージャは、セグメント・レジスタに値を期待するか、またはセグメント・レジスタに値を返す場合、特権レベル 0 のスタックにセーブされているレジスタ・イメージを使用しなければならない。
4. EFLAGS レジスタの VM フラグをクリアする。

5. 選択された割り込みハンドラまたは例外ハンドラの実行を開始する。

トラップゲートまたは割り込みゲートがコンフォーミング・セグメント内または0以外の特権レベルのセグメント内のプロシージャを参照した場合、プロセッサは一般保護例外（#GP）を生成する。その場合、エラーコードは呼び出しが行われようとした先のコード・セグメントのセグメント・セレクタである。

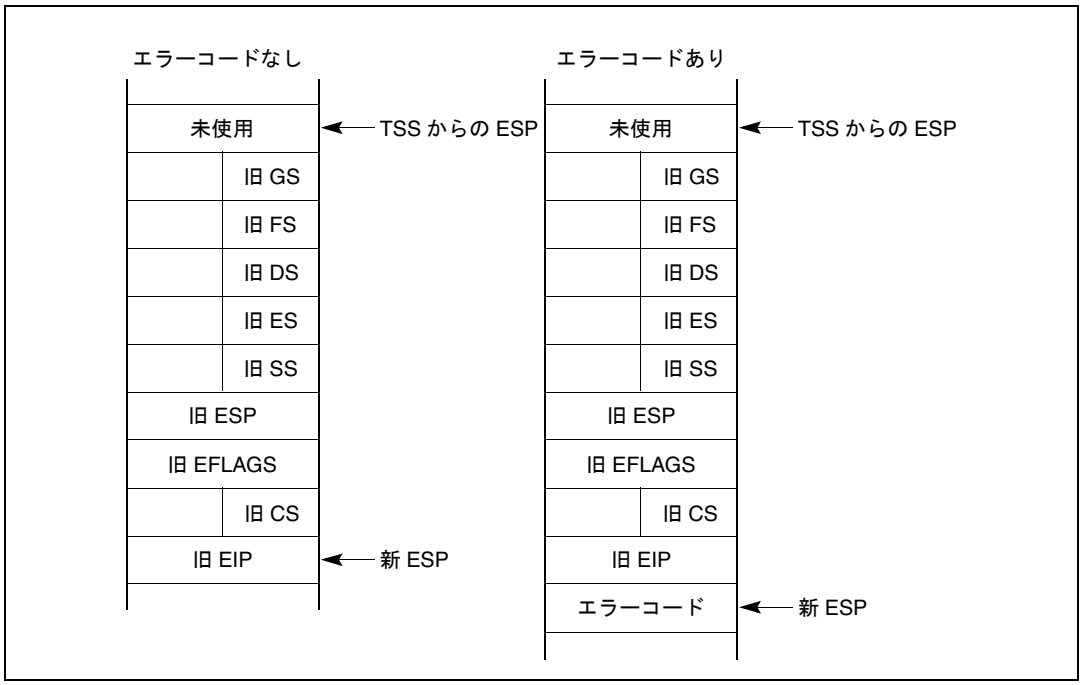


図 16-4. 仮想 8086 モードの割り込みまたは例外後の特権 0 スタック

割り込みハンドラと例外ハンドラは、スタック上のVMフラグを調べて、割り込みをかけられたプロシージャが仮想 8086 モードで実行されていたのかどうかを判定できる。仮想8086モードで実行されていた場合は、割り込みまたは例外は次の3つの方法のいずれかで処理できる。

- 呼び出しされた保護モード割り込みハンドラまたは例外ハンドラが、割り込みまたは例外を処理する。
- 保護モード割り込みハンドラまたは例外ハンドラが、仮想 8086 モニタを呼び出して割り込みまたは例外を処理する。
- （呼び出しされた場合）仮想 8086 モニタが、今度は、8086 プログラムの割り込みハンドラや例外ハンドラに制御を戻す。

割り込みまたは例外が保護モードハンドラで処理される場合は、IRET命令を実行すると、ハンドラは割り込みをかけられた仮想8086モードのプログラムに戻れる。この命

令は、特権レベル0のスタックにセーブされていたイメージからEFLAGSレジスタとセグメント・レジスタにロードする（図16-4.を参照）。EFLAGSイメージのVMフラグがセットされている場合は、プロセッサは切り替えられて仮想8086モードに戻される。IRET命令が実行される時点では、CPLは0でなければならない。そうでなければ、プロセッサはVMフラグの状態を変更しない。

仮想8086モニタは、保護モード割り込みハンドラや例外ハンドラと同様に特権レベル0で動作する。このモニタは、一般的に保護モードの一般保護例外（#GP、ベクタ13）に緊密に結びつけられている。保護モード割り込みハンドラまたは例外ハンドラが仮想8086モニタを呼び出しして割り込みまたは例外を処理した場合は、仮想8086モニタから割り込みをかけられた仮想8086モード・プログラムへの戻りの際には2つの戻り命令が必要である。すなわち、保護モードハンドラに戻る場合のRET命令と、割り込みをかけられたプログラムに戻る場合のIRET命令である。

仮想8086モニタは、16.3.1.2.項「8086プログラムの割り込みハンドラまたは例外ハンドラによる割り込み/例外の処理」の説明にしたがって、割り込みをかけられた8086プログラムの一部である割り込みハンドラまたは例外ハンドラに割り込みや例外をダイレクトして戻すように選択できる。

16.3.1.2. 8086 プログラムの割り込みハンドラまたは例外ハンドラによる割り込み / 例外の処理

仮想8086モード・タスクで実行される8086プログラムは、8086プロセッサ上で実行されるように設計されているので、リニアアドレス0から始まる8086スタイルの割り込みベクタテーブルを持つ。仮想8086モニタが割り込みベクタまたは例外ベクタを正しくダイレクトして、その発生元である仮想8086モードのタスクに戻せれば、8086プログラム内のハンドラが割り込みまたは例外を処理することができる。仮想8086モニタは、割り込みまたは例外を8086プログラムに送り返すには、次に示す手順を実行しなければならない。

1. 8086割り込みベクタを使用して8086プログラムの割り込みテーブル内にある該当ハンドラ・プロシージャが存在する位置を見付ける。
2. 8086プログラムのEFLAGS（下位16ビットのみ）、CS、およびEIPの値を特権レベル3のスタックにストアする。これは、仮想8086モード・タスクが使用しているスタックである。（8086ハンドラはこれらの情報を使用したり、修正ができる。）
3. 特権レベル0のスタック上の戻りリンクを、特権レベル3のハンドラ・プロシージャを指すように変更する。
4. IRET命令を実行して制御を8086プログラムのハンドラに渡す。

5. 特権レベル 3 のハンドラからの IRET 命令が一般保護例外（#GP）をトリガし、したがって、結果的に再び仮想 8086 モニタを呼び出ししたとき、戻りリンクを特権レベル 0 のスタックにリストアして、割り込みをかけられた、元の特権レベル 3 のプロシージャを指すようにする。
6. 特権レベル 3 のスタックから、EFLAGS イメージの下位 16 ビットを特権レベル 0 のスタックにコピーする（これは、一部の 8086 ハンドラが、割り込みを発生させたコードに情報を返すようにこれらのフラグを変更するためである）。
7. IRET 命令を実行して、割り込みをかけられた 8086 プログラムに制御を戻す。

オペレーティング・システムがすべての 8086 MS-DOS* ベースのプログラムをサポートするようにする場合は、プログラムに添付されている実際の 8086 割り込みハンドラと例外ハンドラを使用する必要がある。その理由は、一部のプログラムが、それぞれの独自の割り込みベクタテーブルを修正して、代わりに独自の特殊化された割り込みハンドラや例外ハンドラを使用する（またはシリーズにフック）ためである。

16.3.1.3. タスクゲートを通じた割り込み / 例外の処理

割り込みベクタまたは例外ベクタが IDT 内のタスクゲートを指しているときは、プロセッサは選択された割り込みまたは例外処理タスクにタスクを切り替える。このタスクスイッチの一環として、次に示す処置が実行される。

1. VM フラグがセットされた EFLAGS レジスタが現在の TSS にセーブされる。
2. 呼び出し先タスクの TSS のリンク・フィールドに、割り込みがかけられた仮想 8086 モードタスクに対する TSS のセグメント・セレクタがロードされる。
3. 新しい TSS のイメージから EFLAGS レジスタにロードされる。それによって、VM フラグがクリアされ、プロセッサが保護モードに切り替えられる。
4. EFLAGS レジスタの NT フラグがセットされる。
5. プロセッサが選択された割り込みハンドラタスクまたは例外ハンドラタスクの実行を開始する。

ハンドラタスク内で IRET 命令が実行され、EFLAGS レジスタの NT フラグがセットされると、プロセッサは保護モードの割り込みハンドラまたは例外ハンドラから切り替わって仮想 8086 モード・タスクに戻る。ここで、仮想 8086 モード・タスク用の TSS にセーブされていたイメージから EFLAGS レジスタとセグメント・レジスタにロードされる。EFLAGS イメージの VM フラグがセットされていた場合は、プロセッサはタスクスイッチで切り替わって仮想 8086 モードに戻る。IRET 命令が実行されるときの CPL は 0 でなければならない。0 でない場合は、プロセッサは VM フラグの状態を変更しない。

16.3.2. クラス 2 ー 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理

マスク可能ハードウェア割り込みとは、INTR# ピンを通じて、またはローカル APIC への割り込み要求を通じて伝達される割り込みのことである（5.3.2. 項「マスク可能ハードウェア割り込み」を参照）。これらの割り込みは、EFLAGS レジスタの IF フラグをクリアすると、実行中のプログラムまたはタスクの中断を禁止（マスク）できる。

制御レジスタ CR4 の VME フラグがセットされており、かつ EFLAGS レジスタの IOPL フィールドが 3 より小さいときは、EFLAGS レジスタの次の 2 つのフラグがさらにアクティブにされる。

- VIF（仮想割り込み）フラグ、EFLAGS レジスタのビット 19
- VIP（仮想割り込みペンディング）フラグ、EFLAGS レジスタのビット 20

これらのフラグによって、仮想 8086 モード・タスクの実行中に発生するマスク可能なハードウェア割り込みの処理に対する仮想 8086 モニタの制御の効率が向上する。これらのフラグは、さらに、（PUSHF、POPF、CLI、STI 命令などの）仮想 8086 モニタにトラップするためのすべての IF 関係の操作の必要を取り除けば、割り込み処理のオーバヘッドの低減ももたらす。これらのフラグの目的と用途は次のとおりである。

注記

VIF フラグと VIP フラグは、仮想モード拡張をサポートする IA-32 プロセッサだけで使用可能である。これらの拡張機能はインテル® Pentium® プロセッサで IA-32 プロセッサアーキテクチャに導入された。このメカニズムが使用できないか、またはイネーブルにされていないときは、マスク可能ハードウェア割り込みはクラス 1 の割り込みとして処理される。その場合、VIF フラグと VIP フラグが必要な場合は、仮想 8086 モニタが両フラグをソフトウェアでサポートできる。

在来の 8086 プログラムは、一般的に、マスク可能ハードウェア割り込みをイネーブルにしたり、ディスエーブルにしたりする場合に（例えば、別の割り込みまたは例外を処理中に割り込みをディスエーブルにする場合）、EFLAGS レジスタの IF フラグをそれぞれセットしたり、クリアする。この方法は、シングルタスク環境では十分有効であるが、多くの場合、アプリケーション・プログラムにハードウェア割り込みの処理を直接に制御させないようにするのが望ましいマルチタスキング環境やマルチプロセッサ環境では問題を生じることがある。以前の IA-32 プロセッサの使用に際しては、多くの場合この問題をソフトウェアで仮想的 IF フラグを作成して解決してきた。（インテル Pentium プロセッサ以降の）IA-32 プロセッサでは、VIF フラグと VIP フラグを通じてこの仮想 IF フラグのハードウェア・サポートを提供している。

VIF フラグは IF フラグの仮想化バージョンであり、仮想 8086 タスク内から実行されたアプリケーション・プログラムがマスク可能ハードウェア割り込み処理の制御に使用できる。VIF フラグがイネーブルにされると、CLI 命令と STI 命令は IF フラグに代えて VIF フラグにしたがって動作する。8086 プログラムが CLI 命令を実行すると、プロセッサは VIF フラグをクリアして、仮想 8086 モニタがマスク可能ハードウェア割り込みによるプログラム実行の中断を禁止するように要求する。8086 プログラムが STI 命令を実行すると、プロセッサは VIF フラグをセットして、仮想 8086 モニタが 8086 プログラムに対してマスク可能ハードウェア割り込みをイネーブルにするように要求する。しかし実際には、IF フラグがオペレーティング・システムによって管理されて、常にマスク可能割り込みをイネーブルにするかどうかを制御している。また、これらの状況のもとで 8086 プログラムが PUSHF 命令または POPF 命令を使用して IF フラグを読み取ろうとしたり、変更しようとすると、代わってプロセッサが VIF フラグを変更し、IF フラグは変更されないままになる。

VIP フラグによって、ソフトウェアで据え置き（またはペンディング）のマスク可能ハードウェア割り込みの有無を記録できる。このフラグはプロセッサによって読み取られるが、プロセッサによって明示的に書き込まれることはない。このフラグはソフトウェアからしか書き込めない。

IF フラグがセットされており、かつ VIF フラグと VIP フラグがイネーブルにされていて、プロセッサがマスク可能ハードウェア割り込み（割り込みベクタ 0 ～ 255）を受け取った場合は、プロセッサは次に示す操作を実行し、したがって、割り込みハンドラ・ソフトウェアは併記するような対応操作の実行を求められる。

1. プロセッサが、次の手順の説明にしたがって、受け取った割り込みに対する保護モード割り込みハンドラを呼び出す。これらのステップは、16.3.1.1. 項「保護モードのトラップゲートまたは割り込みゲートを通じた割り込み / 例外の処理」の方法 1 の割り込みと例外の処理について説明されているものとほぼ同じである。
   a. 32 ビット保護モードおよび特権レベル 0 に切り替える。
   b. プロセッサの状態を特権レベル 0 のスタックにセーブする。EIP、CS、EFLAGS、ESP、SS、ES、DS、FS、GS の各レジスタの状態がセーブされる（図 16-4. を参照）。スタック上の EFLAGS イメージ内で、IOPL フィールドが 3 に設定され、VIF フラグが IF フラグにコピーされる。
   c. セグメント・レジスタをクリアする。
   d. EFLAGS レジスタの VM フラグをクリアする。
   e. 選択された保護モード割り込みハンドラの実行を開始する。
2. 保護モード割り込みハンドラの推奨処置は、スタック上の EFLAGS イメージから VM フラグを読み取ることである。このフラグがセットされていた場合、ハンドラは仮想 8086 モニタに呼び出しをかける。

3. 仮想 8086 モニタが EFLAGS レジスタの VIF フラグを読み取らなければならない。
   - VIF フラグがクリアされていた場合は、仮想 8086 モニタはスタック上の EFLAGS イメージの VIP フラグをセットして、ペンディングの据え置き割り込みがあることを示し、保護モードハンドラに戻る。
   - VIF フラグがセットされていた場合は、割り込みが割り込みをかけられた仮想 8086 タスクで実行されていた 8086 プログラムに「属して」いれば（つまり、8086 プログラムの一部であれば）、仮想 8086 モニタが割り込みを処理できる。「属していない」場合は、仮想 8086 モニタは保護モード割り込みハンドラを呼び出しして、それに割り込みを処理させられる。
4. 保護モードハンドラが仮想 8086 モードで実行されていたプログラムへの戻りを実行する。
5. 仮想 8086 モードに戻ったら、プロセッサは 8086 プログラムの実行を継続する。

8086 プログラムは、マスク可能ハードウェア割り込みを受け取り可能であると、STI 命令を実行して VIF フラグをセットする（マスク可能ハードウェア割り込みをイネーブルにする）。VIF フラグをセットする前に、プロセッサは自動的に VIP フラグを調べ、このフラグの状態によって次のいずれかの処置を実行する。

- VIP フラグがクリアされていた（ペンディングの割り込みが存在しないことを示している）場合は、プロセッサは VIF フラグをセットする。
- VIP フラグがセットされていた（ペンディングの割り込みが存在することを示している）場合は、プロセッサは一般保護例外（#GP）を生成する。

ここでの保護モードの一般保護例外ハンドラの推奨処置は、次に、仮想 8086 モニタを呼び出しし、それにペンディングの割り込みの処理である。ペンディングの割り込みを処理した後の仮想 8086 モニタの典型的な処置は、スタック上の EFLAGS イメージの VIP フラグをクリアし、VIF フラグをセットし、次に、仮想 8086 モードへの戻りを実行することである。プロセッサは、次にマスク可能ハードウェア割り込みを受け取ったときは、上記のステップ 1 ～ 5 の説明にしたがってその割り込みを処理する。

与えられた命令の始めで VIP および VIF の両フラグがセットされていることを知った場合、プロセッサは一般保護例外を生成する。この処置によって、仮想 8086 モニタは、VIF フラグをイネーブルにさせた原因である仮想 8086 モード・タスクのペンディングの割り込みを処理できる。この状況は、POPF 命令または IRET 命令が実行された直後か、またはタスクスイッチを介して制御が仮想 8086 モード・タスクに移行した後以外には発生するはあり得ないことに注意する。

VIF フラグと VIP フラグの状態は、実アドレスモードでも、実アドレスモードから保護モードへの、またその逆の遷移中にも変更はされないので注意する。

注記

本項で説明している仮想割り込みメカニズムは、保護モードでも使用できる。16.4. 節「保護モード仮想割り込み」を参照。

16.3.3. クラス 3 ー 仮想 8086 モードでのソフトウェア割り込みの処理

プロセッサは、仮想 8086 モードで動作中にソフトウェア割り込み（INT *n* 命令で発生した割り込み）を受け取ると、6 つの異なる割り込み処理方法を使用できる。選択される方法は、制御レジスタ CR4 の VME フラグ、EFLAGS レジスタの IOPL フィールド、TSS のソフトウェア割り込みリダイレクション・ビット・マップの設定によって決まる。表 16-2. に、仮想 8086 モードの 6 つのソフトウェア割り込み処理方法と、各方法それぞれの VME フラグ、IOPL フィールド、割り込みリダイレクション・ビット・マップの構成ビットの設定の一覧を示す。この表には、各方法に対してプロセッサが行う各種の処置の要約も示してある。

VME フラグは、インテル® Pentium® プロセッサとそれ以降の IA-32 プロセッサの仮想 8086 モード拡張をイネーブルにする。このフラグがクリアされていると、プロセッサは仮想 8086 モードで、Intel386™ プロセッサまたは Intel486™ プロセッサの場合と同様に、割り込みと例外に応答する。このフラグがセットされると、仮想モード拡張によって、仮想 8086 モードに対して次の強化策がもたらされる。

- プロセッサに仮想 8086 モニタをバイパスさせ、ソフトウェア割り込みをリダイレクトして現在実行中の 8086 プログラムの一部である割り込みハンドラに戻せば、仮想 8086 モードでのソフトウェア生成割り込みの処理速度を上げる。
- 8086 プロセッサで実行されるように記述されたソフトウェアに対して仮想割り込みをサポートする。

IOPL の値は、VME フラグと割り込みリダイレクション・ビット・マップのビットとの相互関係によって、個々のソフトウェア割り込みをどのように処理するのかを決定する。

ソフトウェア割り込みリダイレクション・ビット・マップ（図 16-5. を参照）は、TSS 内の 32 バイト・フィールドである。このマップは、TSS 内の I/O 許可ビットマップのすぐ下に配置される。割り込みリダイレクション・ビット・マップの各ビットはそれぞれ特定の割り込みベクタにマッピングされる。割り込みリダイレクション・ビット・マップのビット 0（これは割り込みテーブル内のベクタ 0 にマッピングされる）は、TSS 内の I/O ベース・マップ・アドレスから 32 バイトを引いた位置にある。このビットマップを構成するビットは、セットされていると、それぞれ関連するソフトウェア割り込み（INT *n* 命令で発生する割り込み）が保護モード IDT および割り込みハンドラや例外ハンドラを通じて処理されることを示す。このビットマップのビットがクリ

アされていると、プロセッサは関連するソフトウェア割り込みをリダイレクトして、8086プログラム内の割り込みテーブル（プログラムのアドレス空間のリニアアドレス0にある）に戻す。

注記

ソフトウェア割り込みリダイレクション・ビット・マップは、ハードウェア生成の割り込みと例外には影響を与えない。ハードウェア生成の割り込みと例外は常に保護モードの割り込みハンドラと例外ハンドラによって処理される。

表 16-2. 仮想 8086 モードでのソフトウェア割り込み処理方法

方法	VME	IOPL	リダイレクション・ビット・マップのビット *	プロセッサの処置
1	0	3	X	割り込みが保護モード割り込みハンドラにダイレクトされる。 －VM フラグと TF フラグをクリアする。 －割り込みゲートを通じて処理される場合は、IF フラグをクリアする。 －特権レベル 0 スタックに切り替える。 －GS、FS、DS、ES を特権レベル 0 スタックにプッシュする。 －GS、FS、DS、ES を 0 にクリアする。 －割り込みをかけられたタスクの SS、ESP、EFLAGS、CS、および EIP を特権レベル 0 スタックにプッシュする。 －割り込みゲートから CS と EIP を設定する。
2	0	< 3	X	割り込みが保護モード一般保護例外（#GP）ハンドラにダイレクトされる。
3	1	< 3	1	割り込みが保護モード一般保護例外（#GP）ハンドラにダイレクトされる。クラス 2 のマスク可能ハードウェア割り込み処理のための VIF フラグと VIP フラグのサポート。
4	1	3	1	割り込みが保護モード割り込みハンドラにダイレクトされる（方法 1 のプロセッサ処置を参照）。
5	1	3	0	割り込みが 8086 プログラムの割り込みハンドラにリダイレクトされる。 －NT をクリアし、IOPL を 0 に設定して、EFLAGS をプッシュする。 －CS と EIP（下位 16 ビットのみ）をプッシュする。 －IF フラグをクリアする。 －TF フラグをクリアする。 －現在の仮想 8086 タスクの割り込みベクタテーブル内にある選択されたエントリから CS と EIP（下位 16 ビットのみ）にロードする。

表 16-2. 仮想 8086 モードでのソフトウェア割り込み処理方法（続き）

方法	VME	IOPL	リダイレクション・ビット・マップのビット *	プロセッサの処置
6	1	< 3	0	割り込みが 8086 プログラムの割り込みハンドラにリダイレクトされる。クラス 2 のマスク可能ハードウェア割り込み処理のための VIF フラグと VIP フラグのサポート。 －IOPL を 3 に設定し、VIF を IF にコピーして、EFLAGS をプッシュする。 －CS と EIP（下位 16 ビットのみ）をプッシュする。 －VIF フラグをクリアする。 －TF フラグをクリアする。 －現在の仮想 8086 タスクの割り込みベクタテーブル内にある選択されたエントリから CS と EIP（下位 16 ビットのみ）にロードする。

注：

* 0 に設定されているときは、ソフトウェア割り込みはリダイレクトされて 8086 プログラムの割り込みハンドラに戻され、1 にセットされているときは、割り込みは保護モードのハンドラにダイレクトされる。

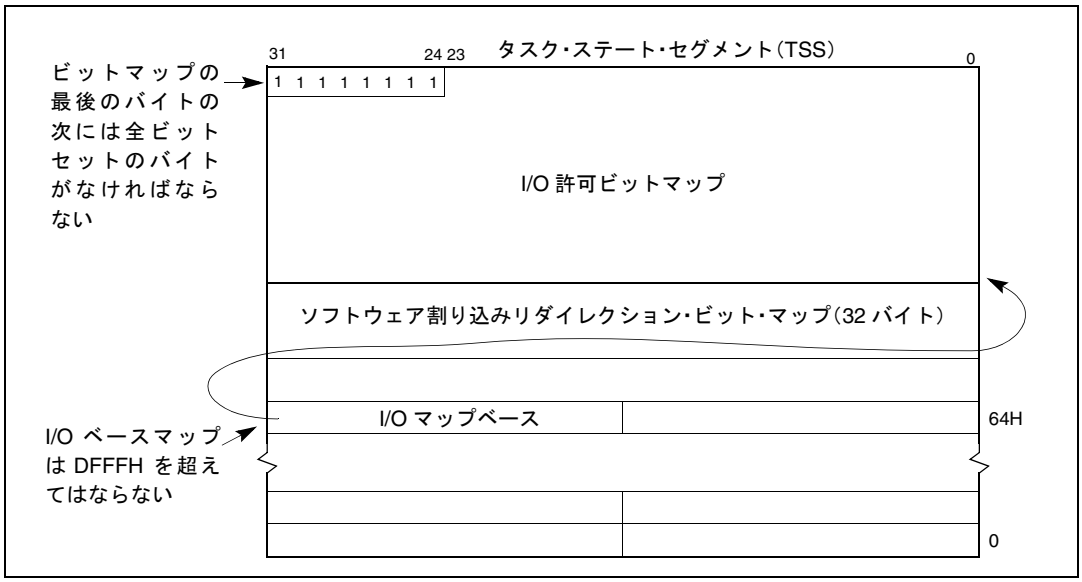


図 16-5. TSS 内のソフトウェア割り込みリダイレクション・ビット・マップ

ソフトウェア割り込みをリダイレクトして 8086 プログラムに戻すと、仮想 8086 モードから保護モードへの、およびその逆の切り替えが必要でなくなるため、潜在的な速度の向上が見込める。この後者の割り込み処理手法は、INT *n* 命令を使用してオペレーティング・システム・プロシージャを呼び出しする（MS-DOS* などの）8086 のオペレーティング・システムにとって特に効果的である。

CPUID 命令を使用して、仮想モード拡張がプロセッサでサポートされているかどうかを確認できる。機能フラグレジスタのビット 1（EDX）は、仮想モード拡張を備えていることを示す（『IA-32 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、中巻 A』の第 3 章の「CPUID—CPU Identification」を参照）。

以降の各項で、仮想 8086 モードでの 6 つのソフトウェア割り込み処理方法（またはメカニズム）について説明する。EFLAGS レジスタの VIF フラグと VIP フラグのマスク可能ハードウェア割り込みに対する用途については、16.3.2. 項「クラス 2 ― 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理」を参照。

16.3.3.1. 方法 1: ソフトウェア割り込みの処理

制御レジスタ CR4 の VME フラグがクリアされており、かつ IOPL フィールドが 3 であるときは、インテル® Pentium® プロセッサまたはそれ以降の IA-32 プロセッサは、ソフトウェア割り込みを Intel386™ プロセッサまたは Intel486™ プロセッサの場合と同様に処理する。つまり、割り込みベクタの指示先の保護モード IDT 内の割り込みハンドラに明示的な呼び出しを実行する。このメカニズムとその可能な用途については、16.3.1. 項「クラス 1 ― 仮想 8086 モードでのハードウェア割り込み/例外の処理」を参照。

16.3.3.2. 方法 2 と 3: ソフトウェア割り込みの処理

仮想 8086 モードでソフトウェア割り込みが発生し、方法 2 または 3 の用意があるときは、プロセッサは一般保護例外（#GP）を生成する。VME フラグが 0 に設定されており、IOPL の値が 3 より小さいときは、方法 2 がイネーブルになる。この場合は、IOPL の値を使用して保護モード割り込みハンドラがバイパスされ、仮想 8086 モードで発生するソフトウェア割り込みはすべて保護モードの一般保護例外（#GP）として取り扱われる。一般保護例外ハンドラは仮想 8086 モニタを呼び出しし、このモニタは次に 8086 プログラム割り込みハンドラをエミュレートするか、または、16.3.1.2. 項「8086 プログラムの割り込みハンドラまたは例外ハンドラによる割り込み / 例外の処理」で説明したように、制御を 8086 プログラムのハンドラに戻すことができる。

方法 3 の処理は、VME フラグが 1 にセットされており、IOPL の値が 3 より小さく、かつソフトウェア割り込みリダイレクション・ビット・マップ内のソフトウェア割り込みに対応するビットが 1 にセットされているときにイネーブルになる。この場合は、プロセッサの操作は方法 2 のソフトウェア割り込み処理に対するそれと同じである。ソフトウェア割り込みリダイレクション・ビット・マップ内のソフトウェア割り込みに対応するビットが 0 に設定されている場合は、割り込みは方法 6 を使用して処理される（16.3.3.5. 項「方法 6: ソフトウェア割り込みの処理」を参照）。

16.3.3.3. 方法 4: ソフトウェア割り込みの処理

方法4の処理は、VMEフラグが1にセットされており、IOPLの値が3であり、かつリダイレクション・ビット・マップ内の割り込みベクタに対するビットが1にセットされているときにイネーブルになる。方法4のソフトウェア割り込み処理では、仮想モード拡張がイネーブルにされているときに方法1スタイルの処理が可能である。つまり、割り込みは保護モードハンドラにダイレクトされる（16.3.3.1. 項「方法1: ソフトウェア割り込みの処理」を参照）。

16.3.3.4. 方法 5: ソフトウェア割り込みの処理

方法5のソフトウェア割り込み処理では、仮想8086モードで発生したソフトウェア割り込み（INT *n* 命令で起動）を8086プログラムの割り込みベクタテーブルとその割り込みハンドラへ返す、ストリームライン化されたリダイレクション方法が利用できる。方法5の処理は、VMEフラグが1にセットされており、IOPLの値が3であり、かつリダイレクション・ビット・マップ内の割り込みベクタに対するビットが0に設定されているときにイネーブルになる。プロセッサは、次に示す処置を実行して選択された8086プログラム割り込みハンドラに暗黙の呼び出しを実行する。

1. NT ビットと IOPL ビットをクリアして、EFLAGS レジスタの下位 16 ビットをスタックにプッシュする。
2. CS レジスタと EIP レジスタの現在値を現行スタックにプッシュする。（EIP レジスタの下位 16 ビットだけがプッシュされ、スタックの切り替えは行われない。）
3. EFLAGS レジスタの IF フラグをクリアして割り込みをディスエーブルにする。
4. EFLAGS レジスタの TF フラグをクリアする。
5. 8086 プログラムの割り込みベクタテーブルを 8086 モード・タスクのリニアアドレス 0 に配置する。
6. 割り込みベクタ番号の指示先の割り込みベクタ・テーブル・エントリから値を CS レジスタと EIP レジスタにロードする。EIP の下位 16 ビットだけがロードされ、上位 16 ビットは 0 に設定される。割り込みベクタテーブルは現在の仮想 8086 タスクのリニアアドレス 0 にあるものとみなされる。
7. 選択された割り込みハンドラの実行を開始する。

ハンドラ・プロシージャの最後にIRET命令を配置すると、上記の手順が逆になり、割り込みをかけられた8086プログラムにプログラムの制御が返される。

方法5の処理では、仮想8086モードから保護モードへの切り替えは行われない。プロセッサは、割り込み処理操作全体を通じて仮想8086モードのままである。

方法5の処理における処置は、実質的には、実アドレスモードでソフトウェア割り込みを処理するときにプロセッサが行う処置と同じである。方法5の処理を使用して

8086プログラムのハンドラにアクセスすることのメリットは、方法2と3の処理のオーバヘッドが回避される点にある。つまり後者の2つの方法では、オーバヘッドとして、まず仮想 8086 モニタに、次に 8086 プログラムのハンドラに制御を移し、その後、割り込みをかけられた 8086 プログラムに戻る前に再び仮想 8086 モニタに戻る必要がある（16.3.1.2. 項「8086 プログラムの割り込みハンドラまたは例外ハンドラによる割り込み / 例外の処理」を参照）。

注記

方法 1 と 4 の処理では、仮想 8086 タスク内で、正規の保護モードハンドラを使用してソフトウェア割り込みを処理できるが、この方法では、すべての仮想 8086 タスクが同じソフトウェア割り込みハンドラを使用する必要があり、それでは仮想 8086 タスクで実行されるプログラム、特に MS-DOS* プログラムに十分な自由が与えられない。

16.3.3.5. 方法 6: ソフトウェア割り込みの処理

方法 6 の処理は、VME フラグが 1 にセットされ、IOPL の値が 3 より小さく、かつリダイレクション・ビット・マップ内の割り込みベクタまたは例外ベクタに対するビットが 0 に設定されているときにイネーブルになる。方法 6 の割り込み処理では、ソフトウェア割り込みは方法 5 の処理の説明（16.3.3.4. 項「方法 5: ソフトウェア割り込みの処理」を参照）と同様に処理される。

方法 6 は、IOPL の値が 3 より小さく設定され、EFLAGS レジスタの VIF フラグと VIP フラグがイネーブルにされており、クラス 2 のマスク可能ハードウェア割り込み処理に対する仮想割り込み（16.3.2. 項「クラス 2 ― 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理」を参照）をサポートする点が方法 5 と異なる。これらのフラグは、仮想 8086 モード・タスクの実行中に発生するマスク可能ハードウェア割り込みの効率的な処理手段を仮想 8086 モニタに提供する。さらに、IOPL の値が 3 より小さく、VIF フラグがイネーブルにされているので、割り込みハンドラを呼び出すときにプロセッサがスタックにプッシュする情報が、方法 5 と 6 とではわずかに異なる（表 16-2. を参照）。

16.4. 保護モード仮想割り込み

（インテル® Pentium® プロセッサ以降の）IA-32 プロセッサも、CR4 レジスタの PVI（保護モード仮想割り込み）フラグをセットすることによって、保護モードで EFLAGS レジスタの VIF フラグと VIP フラグをサポートする。PVI フラグをセットすると、特権レベル 3 で動作しているアプリケーションは、一般保護例外（#GP）を発生させることも、ハードウェア割り込みに影響を与えることもなく、CLI 命令と STI 命令を実行できる。

PVI フラグが 1 にセットされており、CPL が 3 であり、かつ IOPL が 3 より小さいと、STI 命令と CLI 命令は EFLAGS レジスタの VIF フラグをセット、クリアし、IF フラグの設定は変えない。この動作モードでは、保護モードでかつ CPL 3 で動作しているアプリケーションは、仮想 8086 モードのタスクに対しては、16.3.2. 項「クラス 2 － 仮想 8086 モードでの仮想割り込みメカニズムによるマスク可能ハードウェア割り込みの処理」の説明と同様に割り込みを禁止できる。アプリケーションが CLI 命令を実行すると、プロセッサは VIF フラグをクリアする。マスク可能ハードウェア割り込みを受け取った場合、プロセッサは保護モード割り込みハンドラを呼び出す。このハンドラは EFLAGS レジスタの VIF フラグの状態を調べる。VIF フラグがクリアされている（アクティブ・タスクが今は割り込みを処理させたくないことを示す）場合は、ハンドラはスタック上の EFLAGS イメージの VIP フラグをセットし、特権レベル 3 のアプリケーションに戻り、そのアプリケーションがプログラムの実行を継続する。アプリケーションが STI 命令を実行して VIF フラグをセットすると、プロセッサは自動的に一般保護例外ハンドラを呼び出し、そこで、このハンドラがペンディングの割り込みを処理できる。ペンディングの割り込みを処理した後、ハンドラは一般的にはスタック上の EFLAGS イメージの VIF フラグをセットし、VIP フラグをクリアし、アプリケーション・プログラムへの戻りを実行する。プロセッサは、次にマスク可能ハードウェア割り込みを受け取ったときは、CLP 3 での動作時に受け取った割り込みに対する通常の方法でその割り込みを処理する。

仮想モード拡張（CR4 レジスタの VME フラグでイネーブルにする）の場合と同様に、保護モードの仮想割り込みの拡張機能は、マスク可能ハードウェア割り込み（割り込みベクタ 32 ～ 255）だけに影響を与える。NMI の割り込みと例外は通常の方法で処理される。

保護モードの仮想割り込みがディスエーブルになっていると（つまり、制御レジスタ CR4 の PVI フラグが 0 に設定されているか、CPL が 3 より小さいか、または IOPL の値が 3 であると）、CLI 命令と STI 命令は Intel486™ プロセッサと互換性のある方法で実行される。つまり、CPL が I/O 特権レベル（IOPL）より大きい（特権が低い）場合は、一般保護例外が発生する。IOPL の値が 3 の場合は、CLI と STI はそれぞれ IF をクリアまたはセットする。

PUSHF、POPF、IRET、INT は、保護モードの仮想割り込みがイネーブルにされているかどうかに関係なく、Intel486™ プロセッサと同様に実行される。

仮想 8086 モードに移行するには、タスクスイッチか、または IRET 命令を実行する以外に方法はなく、仮想 8086 モードを終了するには、フォルトを発生させて保護モード割り込みハンドラに制御を移す（一般的には一般保護例外ハンドラ。そのハンドラが次に仮想 8086 モード・モニタを呼び出しする）以外に方法はない。いずれの場合も、EFLAGS レジスタがセーブされ、リストアされる。ただし、このことは PVI フラグがセットされていて、プロセッサが仮想 8086 モードでないときの保護モードにはあてはまらない。この保護モード場合は、異なる特権レベルでプロシージャを呼び出しできるが、そうした場合は EFLAGS レジスタはセーブも変更もされない。ただし、CPL が 3 以外のときは、プロセッサは VIF フラグと VIP フラグの状態を決して調べない。

17

16 ビット・コードと 32 ビット・コードの混在

第 17 章
16 ビット・コードと 32 ビット・コードの混在

17

IA-32 プロセッサで実行するように記述されるプログラム・モジュールは、16 ビットまたは 32 ビットのいずれのモジュールであってもよい。表 17-1. に、16 ビット・モジュールと 32 ビット・モジュールの特性を示す。

表 17-1. 16 ビット・プログラム・モジュールと 32 ビット・プログラム・モジュールの特性

特性	16 ビット・プログラム・モジュール	32 ビット・プログラム・モジュール
セグメント・サイズ	0 ～ 64K バイト	0 ～ 4G バイト
オペランド・サイズ	8 ビットと 16 ビット	8 ビットと 32 ビット
ポインタ・オフセット・サイズ（アドレスサイズ）	16 ビット	32 ビット
スタック・ポインタ・サイズ	16 ビット	32 ビット
このサイズのコード・セグメントで可能な制御の移行	16 ビット	32 ビット

IA-32 プロセッサは、32 ビットのプログラム・モジュールを実行するときに最も効率的に機能するが、次に示すいずれかの方法で、16 ビットのプログラム・モジュールも実行できる。

- 実アドレスモード
- 仮想 8086 モード
- システム管理モード（SMM）
- タスクに対するコード、データ、スタックの各セグメントがすべて 16 ビット・セグメントとして構成されているときは、保護モードタスクとして。
- 16 ビット・セグメントと 32 ビット・セグメントを 1 つの保護モードタスクに統合して。
- 16 ビットの動作を 32 ビット・コード・セグメントに統合して。

実アドレスモード、仮想 8086 モード、SMM は、ネイティブ 16 ビット・モードである。インテル® 8086 プロセッサまたはインテル® 286 プロセッサで実行するようにアセンブル / コンパイルされた旧来のプログラムは、実アドレスモードか仮想 8086 モードでは修正しないで実行できる。16 ビットのプログラム・モジュールはまた、システム初期化処理用として実アドレスモードで実行するようにも、またはシステム管理機能処理用として SMM で実行するようにも記述することができる。実アドレスモードと仮

想8086モードの詳細については、第16章「8086エミュレーション」を、SMMの詳細については第13章「システム管理」をそれぞれ参照。

本章では、保護モードでの動作時に16ビット・プログラム・モジュールと32ビット・プログラム・モジュールを統合する方法、および16ビット・コードと32ビット・コードを32ビット・コード・セグメント内に混在させる方法について説明する。

17.1. 16 ビット・プログラム・モジュールと 32 ビット・プログラム・モジュールの定義

次に示すIA-32メカニズムを使用して、16ビットと32ビットのセグメントと動作を識別したり、サポートができる。

- コード・セグメント・ディスクリプタのD（デフォルト・オペランド/アドレスサイズ）フラグ
- スタック・セグメント・ディスクリプタのB(デフォルト・スタック・サイズ) フラグ
- 16ビットと32ビットのコールゲート、割り込みゲート、およびトラップゲート
- オペランド・サイズ命令プリフィックスとアドレスサイズ命令プリフィックス
- 16ビット汎用レジスタと32ビット汎用レジスタ

コード・セグメント・ディスクリプタのDフラグは、コード・セグメントの命令に対するデフォルトのオペランド・サイズとアドレスサイズを決定する。(実アドレスモードと仮想8086モードでは、セグメント・ディスクリプタは使用されず、デフォルトは16ビットである)。Dフラグがセットされているコード・セグメントは32ビット・セグメントであり、Dフラグがクリアされているコード・セグメントは16ビット・セグメントである。

スタック・セグメント・ディスクリプタのBフラグは、暗黙のスタック参照を行う際にプロセッサが使用するスタックポインタのサイズ (32ビットのESPレジスタまたは16ビットのSPレジスタ）を指定する。すべてのデータ・ディスクリプタのBフラグは、エクスパンドダウン・セグメントの上位アドレス範囲の制御も行う。

コールゲート、割り込みゲート、またはトラップゲートを通じてプログラムの制御を他のコード・セグメントに移行するときは、移行中に使用されるオペランド・サイズは使用されるゲートのタイプ（16ビットまたは32ビット）によって決まる（移行命令のDフラグ、プリフィックスのいずれにもよらない)。ゲートのタイプは、戻り情報がどのように（1つ以上の）スタックにセーブされるかを決定する。

最も効率的にかつトラブルなしにプロセッサを動作させるためには、32ビットのプログラムまたはタスクでは、コード・セグメント・ディスクリプタのDフラグとスタック・セグメント・ディスクリプタのBフラグを共にセットし、16ビットのプログラム

またはタスクでは、それらのフラグを共にクリアしなければならない。16 ビット・セグメントと 32 ビット・セグメント間のプログラム制御の移行は、コールゲート、割り込みゲート、またはトラップゲートを通じて行うと最も効率的に処理される。

命令プリフィックスを使用して、コード・セグメントのデフォルトのオペランド・サイズとアドレスサイズをオーバーライドできる。これらのプリフィックスは、実アドレスモードだけでなく保護モードや仮想 8086 モードでも使用できる。オペランド・サイズ用、アドレスサイズ用のどちらのプリフィックスも、サイズが変更されるのは命令の実行時間の間だけである。

17.2. コード・セグメント内の 16 ビット動作と 32 ビット動作の混在

次の 2 つの命令プリフィックスを使用して、1 セグメント内で 32 ビットと 16 ビットの動作を混在できる。

- オペランド・サイズ・プリフィックス（66H）
- アドレス・サイズ・プリフィックス（67H）

これらのプリフィックスは、コード・セグメント・ディスクリプタの D フラグによって選択されたデフォルト・サイズを逆にする。例えば、プロセッサは（MOV mem, reg）命令を次の 4 通りのいずれかに解釈できる。

- 32 ビット・コード・セグメントの場合:
  - 32 ビットの実効アドレスを使用して、32 ビットを 32 ビット・レジスタからメモリに移動する。
  - オペランド・サイズ・プリフィックスが付いている場合は、32 ビットの実効アドレスを使用して、16 ビットを 16 ビット・レジスタからメモリに移動する。
  - アドレス・サイズ・プリフィックスが付いている場合は、16 ビットの実効アドレスを使用して、32 ビットを 32 ビット・レジスタからメモリに移動する。
  - アドレスサイズとオペランド・サイズの両プリフィックスが付いている場合は、16 ビットの実効アドレスを使用して、16 ビットを 16 ビット・レジスタからメモリに移動する。
- 16 ビット・コード・セグメントの場合:
  - 16 ビットの実効アドレスを使用して、16 ビットを 16 ビット・レジスタからメモリに移動する。
  - オペランド・サイズ・プリフィックスが付いている場合は、16 ビットの実効アドレスを使用して、32 ビットを 32 ビット・レジスタからメモリに移動する。

— アドレス・サイズ・プリフィックスが付いている場合は、32 ビットの実効アドレスを使用して、16 ビットを 16 ビット・レジスタからメモリに移動する。

— アドレスサイズとオペランド・サイズの両プリフィックスが付いている場合は、32 ビットの実効アドレスを使用して、32 ビットを 32 ビット・レジスタからメモリに移動する。

上記の各例は、命令が 16 ビットと 32 ビットのどちらのセグメントにあるかに関係なく、任意の命令がオペランド・サイズとアドレスサイズの任意の組み合わせを生成できることを示している。コード・セグメントに対して 16 ビットと 32 ビットのどちらのデフォルトを選択するかは、通常次に示す基準による。

- **性能** — 可能なときは常に 32 ビット・コード・セグメントを使用する。P6 ファミリ・プロセッサでは、32 ビット・コード・セグメントの方が 16 ビット・コード・セグメントよりも実行がはるかに高速であり、P6 ファミリ・プロセッサより前の IA-32 プロセッサではいく分高速である。
- **コード・セグメントが実行されるオペレーティング・システム** — オペレーティング・システムが 16 ビットのオペレーティング・システムの場合は、32 ビットのプログラム・モジュールはサポートされない可能性がある。
- **動作モード** — コード・セグメントは、実アドレスモード、仮想 8086 モード、または SMM で実行されるように設計する場合は、16 ビットのコード・セグメントでなければならない。
- **初期の IA-32 プロセッサとの下位互換性** — コード・セグメントはインテル® 8086 プロセッサまたはインテル® 286 プロセッサで実行できなければならない。つまり、16 ビット・コード・セグメントでなければならない。

17.3. 異なるサイズのコード・セグメント間でのデータの共有

データ・セグメントには、16 ビットと 32 ビットの両コード・セグメントからアクセスできる。64K バイトを超えるデータ・セグメントを 16 ビット・コード・セグメントと 32 ビット・コード・セグメントで共有するときは、16 ビット・コード・セグメントからアクセスされるデータは、データ・セグメントの最初の 64K バイト内になければならない。その理由は、定義によって、16 ビット・ポインタはセグメントの最初の 64K バイトしか指せないためである。

スタックは、サイズが 64K バイト未満であれば、16 ビットと 32 ビットの両コード・セグメントによって共有できる。このクラスのスタックとしては、次のものがある。

- スタック・セグメント・ディスクリプタの G（グラニュラリティ）フラグと B（ビッグ）フラグがクリアされているエクスパンドアップ・セグメント内のスタック。

- G フラグと B フラグがクリアされているエクスパンドダウン・セグメント内のスタック。
- Gフラグがセットされ、Bフラグがクリアされているエクスパンドアップ・セグメント内にあって、かつ下位64Kバイト内に完全に収まっているスタック。（スタック以外の、共有されないデータには、FFFFHを超えるオフセットが使用できる。）

G フラグ、B フラグ、エクスパンドダウン・スタック・タイプについては、3.4.3. 項「セグメント・ディスクリプタ」を参照。

一般的には、16ビット・コード・セグメントで使用されるスタックのサイズはBフラグを使用して変更はできない。このフラグが制御するのは、割り込み、例外の他に、PUSH、POP、CALL、RETの各命令によって生じる暗黙のスタック参照用のスタックポインタのサイズだけである。B フラグは、パラメータやローカル変数へのアクセスなどの明示的なスタック参照は制御しない。16 ビット・コード・セグメントが 32 ビット・スタックを使用できるのは、そのスタックへのすべての明示的な参照に 32 ビットのアドレス・サイズ・プリフィックスを付け（それによって、それらの参照に 32 ビットのアドレス指定が使用される）、スタックポインタへの明示的な書き込みに 32 ビットのオペランド・サイズ・プリフィックスを付けた場合に限られる。

32 ビットのエクスパンドダウン・セグメントでは、すべてのオフセットが 64K バイトを超える可能性がある。したがって、32 ビットのアドレス指定を使用するようにコード・セグメントを修正しなければ、16 ビット・コードはこの種のスタック・セグメントを使用できない。

17.4. 異なるサイズのコード・セグメント間での制御の移行

16 ビット・コード・セグメントのプロシージャが 32 ビット・コード・セグメントに安全に呼び出しを実行する方法は次の 3 つである。

- 32 ビットのコールゲートを通じて呼び出しを実行する。
- 32 ビットのインターフェイス・プロシージャに 16 ビット呼び出しを実行する。インターフェイス・プロシージャが次に目的のデスティネーションに 32 ビット呼び出しを実行する。
- 16 ビット・プロシージャを修正する。つまり、呼び出しの前にオペランド・サイズ・プリフィックスを挿入して 32 ビット呼び出しに変更する。

同様に、32 ビット・コード・セグメントのプロシージャが 16 ビット・コード・セグメントに安全に呼び出しを実行する方法は次の 3 つである。

- 16 ビットのコールゲートを通じて呼び出しを実行する。この場合、CALL 命令における EIP の値は FFFFH を超えてはならない。
- 16 ビットのインターフェイス・プロシージャに 32 ビット呼び出しを実行する。インターフェイス・プロシージャが次に目的のデスティネーションに 16 ビット呼び出しを実行する。
- 32 ビットのプロシージャを修正する。つまり、呼び出しの前にオペランド・サイズ・プリフィックスを挿入して 16 ビット呼び出しに変更する。必ず、戻りオフセットが FFFFH を超えないようにする。

プログラム制御を移行する際に上記の方法を使用すれば、次に示す、16 ビット・コード・セグメントと 32 ビット・コード・セグメント間の呼び出しにおけるアーキテクチャ上の制約を回避できる。

- 16 ビット・コード・セグメントからのポインタ（デフォルトでは、16 ビットのポインタしか許されない）は、32 ビット・セグメント内の FFFFH を超える位置ではデータもコードもアドレス指定できない。
- スタックのコヒーレンシ（一貫性）を維持するには、CALL 命令とそれと対をなす RETURN 命令のオペランド・サイズ属性を同じにしなければならない。これは、割り込みハンドラと例外ハンドラに対する暗黙の呼び出し、およびそれらと対をなす IRET 命令についてもあてはまる。
- FFFFH を超える 32 ビット・パラメータ（特にポインタ・パラメータ）は、スタック上の 16 ビットのパラメータ位置に圧縮してストアできない。
- スタックポインタ（SP または ESP）のサイズは、16 ビット・コード・セグメントと 32 ビット・コード・セグメント間の切り替え時に変わる。

これらの制約について、以降の各項で十分詳細に説明する。

17.4.1. コード・セグメント・ポインタのサイズ

ポインタを使用して次の命令を識別する制御移行命令（つまり、ゲートを使用しない制御移行命令）に対しては、オペランド・サイズ属性がポインタのオフセット部分のサイズを決定する。この規則は、次のような意味を持つ。

- 32 ビット・オペランド・サイズを使用して 32 ビット・ポインタが FFFFH を超えない限り、32 ビット・セグメントから 16 ビット・セグメントへの JMP、CALL、または RET 命令は常に可能である。
- 16 ビット・セグメントから 32 ビット・セグメントへの JMP、CALL、または RET 命令は、命令にオペランド・サイズ・プリフィックスが付いていなければ、FFFFH を超えるデスティネーションをアドレス指定できない。

16 ビット・セグメントから 32 ビット・セグメント内の FFFFH を超えるデスティネーションにプログラムの制御を移行できるインターフェイス・プロシージャについては、17.4.5. 項「インターフェイス・プロシージャの記述」を参照。

17.4.2. 制御移行のためのスタック管理

スタックは 16 ビットのプロシージャ呼び出しと 32 ビット呼び出しとでは異なる方法で管理されるため、RET 命令のオペランド・サイズ属性は CALL 命令のそれと一致しなければならない（図 17-1. を参照）。16 ビット呼び出しでは、プロセッサは 16 ビットの IP レジスタと（異なる特権レベル間の呼び出しの場合）16 ビットの SP レジスタの内容をプッシュする。CALL 命令と対をなす RET 命令も 16 ビットのオペランド・サイズを使用して、スタックから 16 ビット・レジスタにそれらの 16 ビット値をポップしなければならない。

32 ビットの CALL 命令は、32 ビットの EIP レジスタと（異なる特権レベル間の呼び出しの場合）32 ビットの ESP レジスタの内容をプッシュする。この場合、CALL 命令と対をなす RET 命令は 32 ビットのオペランド・サイズを使用して、スタックから 32 ビット・レジスタにそれらの 32 ビット値をポップしなければならない。対をなす CALL/RET 命令のオペランド・サイズが一致しない場合、スタックは正しく管理されず、命令ポインタとスタックポインタの値は正しい値にリストアされない。

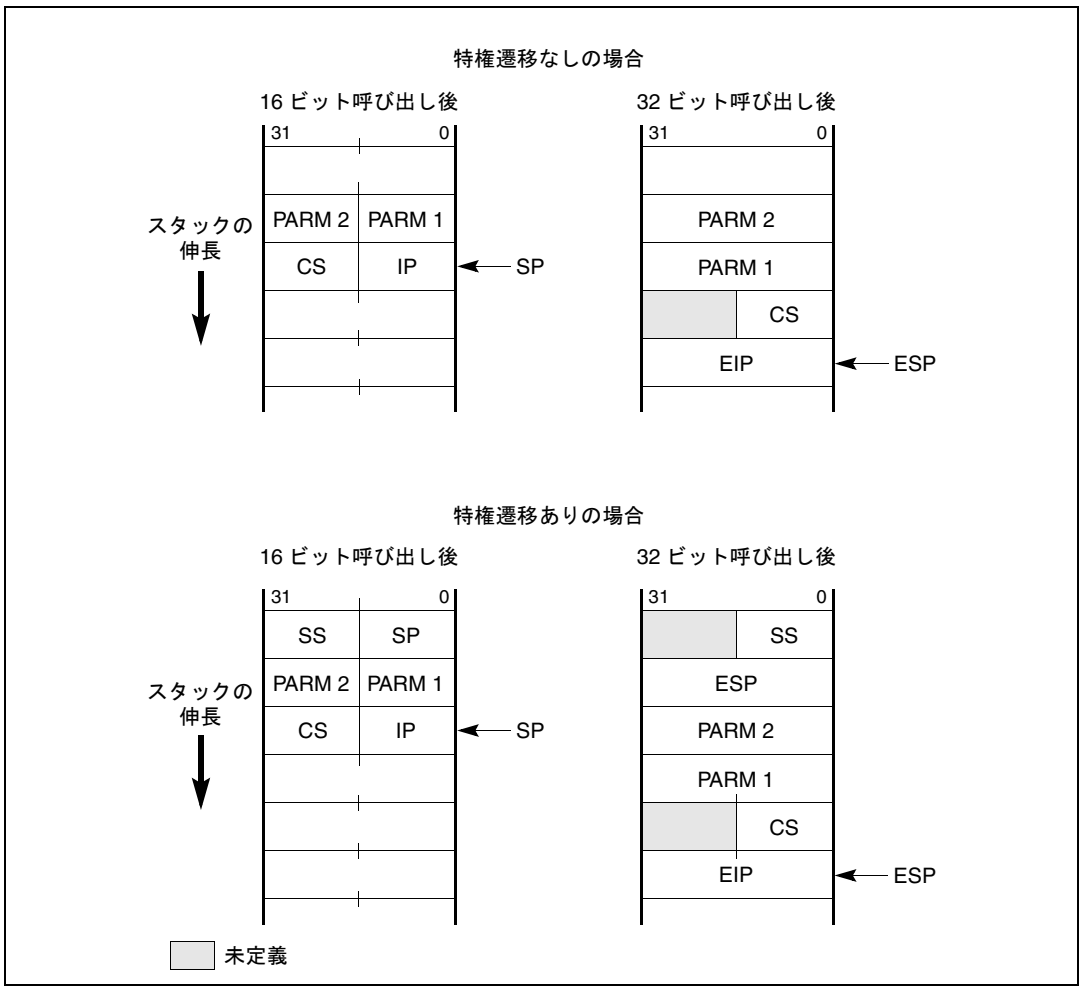


図 17-1. 16 ビットと 32 ビットの far コール後のスタック

32 ビットのコードの実行中に、16 ビットのコールゲートを通じて、同位以上の特権レベル（つまり、呼び出し先コード・セグメントの DPL が呼び出し側コード・セグメントの CPL より小さいか等しい）の 16 ビット・コード・セグメントへの呼び出しが実行された場合は、32 ビットのコード・セグメントへの戻り時（つまり、16 ビット・コード・セグメントでの RET の実行後）には ESP レジスタの上位 16 ビットは信頼できないことがある。

CALL 命令とそれと対をなす RET 命令が、D フラグの値が同じであるコード・セグメント（つまり、共に 32 ビット・コード・セグメントか、共に 16 ビット・コード・セグメント）にあるときは、デフォルト設定を使用できる。CALL 命令とそれと対をなす RET 命令が、D フラグの設定が異なるセグメントにあるときは、オペランド・サイズ・プリフィックスを使用しなければならない。

17.4.2.1. 呼び出しのオペランド・サイズ属性の制御

呼び出しのオペランド・サイズは、次の3つの項目によって決定できる。

- 呼び出し側コード・セグメントのセグメント・ディスクリプタのDフラグ
- 命令のオペランド・サイズ・プリフィックス
- 呼び出しがコールゲートを通じて行われる場合、コールゲートのタイプ（16 ビットまたは32ビット）

呼び出しが（コールゲートではなく）ポインタを使用して行われるときは、呼び出し側コード・セグメントのDフラグによってCALL命令のオペランド・サイズが決まる。このオペランド・サイズ属性は、CALL 命令の前にオペランド・サイズ・プリフィックスを挿入するとオーバーライド（無効に）できる。したがって、例えばコード・セグメントの D フラグが 16 ビット用としてセットされており、CALL 命令にオペランド・サイズ・プリフィックスが使用された場合は、プロセッサは情報を 32 ビットのフォーマットでスタックにストアする。呼び出しが32ビット・コード・セグメント用の場合は、そのコード・セグメント内の命令は、コヒーレンシが損なわずにスタックを読み取れる。32ビット・コード・セグメントからのRET命令の場合も、やはり、オペランド・サイズ・プリフィックスなしでスタックのコヒーレンシを維持し、16ビット・コード・セグメントに戻る。

CALL 命令がコール・ゲート・ディスクリプタを参照するときは、呼び出しのタイプはコールゲートのタイプ（16 ビットまたは 32 ビット）によって決まる。呼び出し先のコード・セグメント内のデスティネーションへのオフセットは、ゲート・ディスクリプタから得られる。したがって、32ビットのコールゲートが使用された場合は、32ビットのコールゲートは32ビットのオフセットを使用するので、16ビット・コード・セグメント内のプロシージャは32ビット・コード・セグメントのベースから64Kバイトを超える位置にあるプロシージャを呼び出せる。

呼び出しのオペランド・サイズおよびそれがどのように決定されるかに関係なく、使用されるスタックポインタのサイズ（SP または ESP）は、常に現在使用中のスタック・セグメント・ディスクリプタのBフラグによって制御される（つまり、Bがクリアされているときは SP が使用され、B がセットされているときは ESP が使用される）。

8086プロセッサ上やそれ以降のIA-32アーキテクチャ・プロセッサ上の実アドレスモードで正常に実行されていた16ビット・コード・セグメントは、無修正のままではそのDフラグがクリアされ、オペランド・サイズ・オーバーライド・プリフィックスを使用しなくなる。結果として、このコード・セグメント内にあるすべてのCALL命令は、16ビットのオペランド・サイズ属性を使用することになる。そのようなコード・セグメント内のプロシージャは、次の2つの方法のいずれかによって、32ビット・コード・セグメントに安全にプロシージャを呼び出すように修正できる。

- 32 ビットのコールゲートを指示先にするように CALL 命令のリンクを変更する（17.4.2.2. 項「ゲートによるパラメータの受け渡し」を参照）。
- 各 CALL 命令に 32 ビット・オペランド・サイズ・プリフィックスを追加する。

17.4.2.2. ゲートによるパラメータの受け渡し

16 ビットのプロシージャで 32 ビットのゲートを参照するときは、各プロシージャ呼び出しで渡されるパラメータの数を考慮することが重要である。ゲート・ディスクリプタのカウント・フィールドは、現行スタックからより上位の特権レベルの（特権レベル番号が小さい）プロシージャのスタックにコピーされるパラメータ・ストリングのサイズを指定する。16 ビット・ゲートのカウント・フィールドは、コピーされる 16 ビット・ワード数を指定する。それに対し、32 ビット・ゲートのカウント・フィールドは、コピーされる 32 ビット・ダブルワード数を指定する。したがって、32 ビット・ゲートのカウント・フィールドは、16 ビットのプロシージャによってスタックに投入されるワード数サイズの半分でなければならない。また、16 ビット・プロシージャは、パラメータとして偶数のワード数を使用しなければならない。

17.4.3. 割り込み制御移行

例外または割り込みの結果生じるプログラム制御の移行は、常に割り込みゲートまたはトラップゲート（IDT 内にある）を通じて行われる。この場合は、ゲートのタイプ（16 ビットまたは 32 ビット）によって、別のコード・セグメント内にある例外ハンドラ・プロシージャまたは割り込みハンドラ・プロシージャへの暗黙の呼び出しに使用されるオペランド・サイズ属性が決まる。

例外または割り込みが 32 ビットまたは 16 ビットのいずれのコード・セグメントで発生しても、32 ビットの割り込みゲートまたはトラップ・ゲートを使用すると、32 ビットの例外ハンドラまたは割り込みハンドラに安全なインターフェイスを提供できる。しかし、スタックには 16 ビットの戻りアドレスしかセーブされないので、ときとして、16 ビット・コード・セグメントに例外ハンドラまたは割り込みハンドラを使用することは実用的でないことがある。EIP が FFFFH より大きいときに 32 ビット・コード・セグメントで例外または割り込みが発生した場合は、16 ビット・ハンドラ・プロシージャでは正しい戻りアドレスを提供できない。

17.4.4. パラメータの変換

セグメント・オフセットまたは（セグメント・オフセットをストアする）ポインタがパラメータとして 16 ビットと 32 ビットのプロシージャ間で受け渡しされるときは、若干の変換が必要になる。32 ビット・プロシージャが 64K バイトを超える位置にあるデータを指すポインタを 16 ビット・プロシージャに渡した場合は、16 ビット・プロ

シージャはそれを使用できない。この制約を除けば、インターフェイス・コードは、32 ビットと 16 ビットのポインタ間で必要とされる任意のフォーマット変換を行える。

32 ビット・コードと 16 ビット・コードの間で値によって受け渡されるパラメータについても、32 ビットと 16 ビットのフォーマット間の変換が必要な場合がある。この変換の形式はアプリケーションに依存する。

17.4.5. インターフェイス・プロシージャの記述

32 ビット・プロシージャと 16 ビット・プロシージャの間にインターフェイス・コードを挿入すれば、次のインターフェイス問題を解決できることがある。

- 16 ビット・コード・セグメント内のプロシージャに、32 ビット・コード・セグメント内で FFFFH より大きいオフセットを持つプロシージャを呼び出すことを許可する。
- 対をなす CALL 命令と RET 命令との間のオペランド・サイズ属性を一致させる。
- カウントが可変であるか、または 16 ビット・ワード数が奇数である管理パラメータ・ストリングを含めて、パラメータ（データ）を変換する。
- 可能な場合、ESP レジスタの上位ビットを無効にする。

インターフェイス・プロシージャは、次の規則に従うことによって単純化される。

1. インターフェイス・プロシージャは 32 ビット・コード・セグメントに常駐させなければならない（コード・セグメント・ディスクリプタの D フラグがセットされる）。
2. 16 ビット・プロシージャが呼び出す可能性のあるプロシージャについてはすべて、オフセットが FFFFH を超えてはならない。
3. 16 ビット・プロシージャがセーブする戻りアドレスについてはすべて、オフセットが FFFFH を超えてはならない。

これらの規則のどれに違反しても、インターフェイス・プロシージャはより複雑になる。例えば、16 ビット・プロシージャがエントリ・ポイントが FFFFH を超える 32 ビット・プロシージャを呼び出す場合は、インターフェイス・プロシージャがエントリポイントへのオフセットを提供する必要がある。コールゲートのゲート・ディスクリプタが持っているアドレスが 32 ビットであるため、16 ビット・アドレスと 32 ビット・アドレスとの間のマッピングは、コールゲートを使用した場合しか自動的には行われない。コールゲートを使用しない場合は、インターフェイス・コードが 32 ビット・アドレスを提供しなければならない。

インターフェイス・プロシージャの構造は、次のように、それがサポートする呼び出しのタイプによって異なる。

- **16 ビット・プロシージャから 32 ビット・プロシージャへの呼び出し。**16 ビット・コード・セグメントからインターフェイス・プロシージャへの呼び出しは、（デフォルトでは、呼び出し側コード・セグメント・ディスクリプタの D フラグがクリアされているため）16 ビットの CALL 命令を使用して実行され、インターフェイス・プロシージャから呼び出し側プロシージャに戻る RET 命令には 16 ビットのオペランド・サイズ・プリフィックスが使用される。インターフェイス・プロシージャから 32 ビット・プロシージャへの呼び出しは、（デフォルトでは、インターフェイス・プロシージャのコード・セグメントの D フラグがセットされているため）32 ビットの CALL 命令を使用して実行され、呼び出し先プロシージャからインターフェイス・プロシージャへの戻りは、（やはりデフォルトでは）32 ビットの RET 命令を使用して実行される。
- **32 ビット・プロシージャから 16 ビット・プロシージャへの呼び出し。**32 ビット・コード・セグメントからインターフェイス・プロシージャへの呼び出しは、（デフォルトでは）32 ビットの CALL 命令を使用して実行され、インターフェイス・プロシージャから呼び出し側プロシージャへの戻りは、（やはりデフォルトでは）32 ビットの RET 命令を使用して実行される。インターフェイス・プロシージャから 16 ビット・プロシージャへの呼び出しでは、CALL 命令にオペランド・サイズ・プリフィックスを使用する必要があり、呼び出し先プロシージャからインターフェイス・プロシージャへの戻りは、（デフォルトでは）16 ビットの RET 命令を使用して実行される。

18

IA-32 の互換性

第 18 章
IA-32 の互換性

18

IA-32 プロセッサは、すべてバイナリレベルで互換性がある。互換性とは、特定の限定された制約条件内で、前世代の IA-32 プロセッサ上で実行するようにデザインされたプログラムを、後世代の IA-32 プロセッサ上で実行したときに同じ結果が得られることである。互換性の制約条件、およびそれらの IA-32 プロセッサ間に存在するすべての相違について、本章で説明する。

新しい IA-32 プロセッサでは、ソフトウェアから見えるアーキテクチャがこれまでの IA-32 プロセッサに見受けられたものから拡張されている。それらの拡張項目は、従来と将来のプロセッサとの互換性に配慮しながら定義されている。本章では、それらの拡張項目に対する互換性にかかわる配慮点についても概説する。

18.1. IA-32 プロセッサ・ファミリとカテゴリ

本章では、次に示すように、関係する互換性情報のタイプによっていくつかの異なる呼び方で IA-32 プロセッサを指す。

- **IA-32 プロセッサ** － インテル® IA-32 アーキテクチャに基づくすべてのインテル・プロセッサ。これらには、8086/88 プロセッサ、インテル® 286 プロセッサ、Intel386™ プロセッサ、Intel486™ プロセッサ、インテル® Pentium® プロセッサ、インテル® Pentium® Pro プロセッサ、インテル® Pentium® II プロセッサ、インテル® Pentium® III プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサが含まれる。
- **32 ビット・プロセッサ** － 32 ビット・アーキテクチャを使用しているすべての IA-32 プロセッサ。これらには、Intel386 プロセッサ、Intel486 プロセッサ、インテル Pentium プロセッサ、インテル Pentium Pro プロセッサ、インテル Pentium II プロセッサ、インテル Pentium III プロセッサ、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサが含まれる。
- **16 ビット・プロセッサ** － 16 ビット・アーキテクチャを使用しているすべての IA-32 プロセッサ。これらには、8086/88 プロセッサと 286 プロセッサが含まれる。
- **P6 ファミリ・プロセッサ** － P6 マイクロアーキテクチャに基づくすべての IA-32 プロセッサ。これらには、インテル Pentium Pro プロセッサ、インテル Pentium II プロセッサ、インテル Pentium III プロセッサが含まれる。
- **インテル Pentium 4 ファミリ・プロセッサ** － Intel NetBurst® マイクロアーキテクチャに基づくすべての IA-32 プロセッサ。

- **インテル Xeon ファミリ・プロセッサ** — Intel NetBurst マイクロアーキテクチャに基づく IA-32 プロセッサ製品ファミリ。これには、インテル Xeon プロセッサとインテル Xeon プロセッサ MP が含まれる。

18.2. 予約ビット

本書全体を通じて、多くのレジスタとメモリ・レイアウトの記述で特定のビットが予約（済み）と記されている。ビットが未定義または予約（済み）と記されているときは、ソフトウェアは、それらのビットを将来何らかの機能を持つものとして取り扱うことが、将来のプロセッサとの互換性にとって不可欠である。予約ビットの取り扱いに際しては、ソフトウェアは次に示すガイドラインに従わなければならない。

- 予約ビットが存在するレジスタまたはメモリ位置の値のテストに際しては、どの予約ビットの状態にも依存してはならない。予約ビットは、テストの前にマスクしてテスト対象から除外する。
- 予約ビットをメモリまたはレジスタにストアする際は、どの予約ビットの状態にも依存してはならない。
- 予約ビットに書き込まれた情報はそのまま維持されるとは限らないため、それらに依存してはならない。
- レジスタにロードする際は、予約ビットには、マニュアルに指示がある場合はその指示された値をロードするか、または同じレジスタから前に読み取った値をロードし直すこと。

既存の IA-32 プロセッサ用に記述されたソフトウェアは、予約ビットを正しく処理すれば、保護例外を生成しないで、将来の IA-32 プロセッサに移植できる。

18.3. 新しい機能とモードの有効化

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサ用に定義されている新しい制御機能は、大部分が制御レジスタ（主としてレジスタ CR4）の新しいモードフラグによってイネーブルにされる。このレジスタは、インテル Pentium プロセッサより前の IA-32 プロセッサに対しては未定義である。Intel486™ 以前の IA-32 プロセッサでこのレジスタにアクセスしようとすると、無効オペコード例外（#UD）が発生する。その結果、Intel486 以前の IA-32 プロセッサで正しく実行されるプログラムがこれらの機能を誤ってイネーブルにすることはあり得ない。レジスタ CR4 の予約ビットをその元の値以外の値に設定しようとすると、一般保護例外（#GP）が発生する。したがって、P6 ファミリ・プロセッサとインテル Pentium プロセッサで実行されるプログラムが、将来の IA-32 プロセッサでサポートされる可能性のある機能を誤ってイネーブルにすることはあり得ない。

P6 ファミリ・プロセッサとインテル Pentium プロセッサは、モデル固有レジスタの予約ビットをセットしようとしたかどうかのチェックは行わない。この方法を取り入れるのは、ソフトウェア作成者の義務である。それらの予約ビットは将来のインテル・プロセッサで使用される可能性がある。

18.4. ソフトウェアによる新機能の有無検出

ソフトウェアでは、アーキテクチャ上の新機能と拡張が存在するかどうかを次の 2 つの方法のいずれかで調べられる。

- 機能または拡張の有無をテストする — ソフトウェアは、EFLAGS レジスタと制御レジスタに新しいフラグが存在するかどうかをテストできる。それらのフラグが予約されている（テストを実行しているプロセッサには存在しないことを意味している）場合は、例外が発生する。同様に、ソフトウェアは、新しい命令を試しに実行して、その命令がサポートされていない場合に、無効オペコード例外 (#UD) を発生させることもできる。
- CPUID 命令を実行する — CPUID 命令（インテル® Pentium® プロセッサで IA-32 プロセッサアーキテクチャに追加された）は新しい機能の有無を直接に示す。

新しいプロセッサ機能と拡張の検出の詳細については、『IA-32 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』の第 14 章「プロセッサの識別と機能の判定」を参照。

18.5. インテル® MMX® テクノロジ

MMX® テクノロジは、一連の MMX 命令と共に、MMX テクノロジ Pentium プロセッサで IA-32 アーキテクチャに導入された。MMX® 命令については、『IA-32 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』の第 5 章「命令セットの要約」に要約が示してあり、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」、『IA-32 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、中巻 B』の第 4 章「命令セット・リファレンス N-Z」に詳述してある。MMX テクノロジはインテル® Pentium® II プロセッサ、インテル® Pentium® III プロセッサ、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサにも取り入れられている。

18.6. ストリーミング SIMD 拡張命令（SSE）

ストリーミング SIMD 拡張命令（SSE）は、インテル® Pentium® III プロセッサで導入された。SSE は、新しい命令セットとレジスタセットで構成される。新しいレジスタには、8 個の 128 ビット XXM レジスタと 1 個の 32 ビット MXCSR 制御 / ステータス・レジスタが含まれている。これらの命令とレジスタは、単精度浮動小数点数に対する SIMD 演算を実行できるように設計されている。これらの新しい命令の一部は、MMX® レジスタも操作する。SSE の命令とレジスタについては、『IA-32 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』の第 10 章「ストリーミング SIMD 拡張命令（SSE）によるプログラミング」『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」、『IA-32 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、中巻 B』の第 4 章「命令セット・リファレンス N-Z」 を参照のこと。

18.7. ストリーミング SIMD 拡張命令 2（SSE2）

ストリーミング SIMD 拡張命令 2 は、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで導入された。SSE2 は、XXM レジスタと MXCSR レジスタを操作し、倍精度浮動小数点値と整数値の SIMD 演算を実行する、新しい命令セットで構成される。これらの新しい命令の一部は、MMX® レジスタも操作する。SSE2 の命令とレジスタについては、『IA-32 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』の第 11 章「ストリーミング SIMD 拡張命令 2（SSE2）によるプログラミング」『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」、『IA-32 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、中巻 B』の第 4 章「命令セット・リファレンス N-Z」 を参照のこと。

18.8. ストリーミング SIMD 拡張命令 3（SSE3）

ストリーミング SIMD 拡張命令 3（SSE3）は、HT テクノロジ インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで導入された。SSE3 は、13 個の命令で構成される。13 個の命令のうち 10 個は、SSE/SSE2 で使用される SIMD（Single Instruction, Multiple Data）実行モデルをサポートする。1 つの SSE3 命令は、整数への変換のための x87 方式プログラミングを容易にする。これ以外の 2 つの命令（MONITOR と MWAIT）は、スレッドの同期を高速化する。SSE3 については、『IA-32 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』の第 12 章「ストリーミング SIMD 拡張命令 3（SSE3）によるプログラミング」、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」、『IA-32 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、中巻 B』の第 4 章「命令セット・リファレンス N-Z」 を参照のこと。

18.9. ハイパー・スレッディング・テクノロジ

ハイパー・スレッディング・テクノロジは、1 つの物理プロセッサが 2 つ以上の別々のコード・ストリーム（スレッドと呼ばれる）を同時に実行できるように、IA-32 アーキテクチャを拡張する技術である。この機能は、インテル® Xeon™ プロセッサ MP およびインテル Xeon プロセッサの最近のステッピングで IA-32 アーキテクチャに導入された。これらのプロセッサは、1 つの物理プロセッサ当たり 2 つの論理プロセッサを持つハイパー・スレッディング・テクノロジをサポートしている。ハイパー・スレッディング・テクノロジのアーキテクチャの詳細については、7.6. 節「ハイパー・スレッディング・テクノロジ」を参照のこと。

18.10. インテル® Pentium® プロセッサ以降の IA-32 プロセッサの新しい命令

表 18-1. に、インテル® Pentium® プロセッサ以降の IA-32 プロセッサで導入された命令を示す。

18.10.1. インテル® Pentium® プロセッサより前に追加された命令

次に示す命令は、Intel486™ プロセッサに追加された。

- BSWAP（バイトスワップ、Byte Swap）命令
- XADD（交換 / 加算、Exchange and Add）命令
- CMPXCHG（比較 / 交換、Compare and Exchange）命令
- INVD（キャッシュ無効化、Invalidate Cache）命令
- WBINVD（ライトバック / キャッシュ無効化、Write-Back and Invalidate Cache）命令
- INVLPG（TLB エントリ無効化、Invalidate TLB Entry）命令

表 18-1. インテル® Pentium® プロセッサ以降の IA-32 プロセッサでの新しい命令

命令	CPUID 表示ビット	導入されたプロセッサ
CMOV*cc*（条件付きデータ移動、Conditional Move）	EDX、ビット 15	インテル® Pentium® Pro プロセッサ
FCMOV*cc*（条件付き浮動小数点データ移動、Floating-Point Conditional Move）	EDX、ビット 0 と 15	
FCOMI（浮動小数点 EFLAGS 比較 / セット、Floating-Point Compare and Set EFLAGS）	EDX、ビット 0 と 15	
RDPMC（性能モニタリング・カウンタ読み取り、Read Performance Monitoring Counter）	EAX、ビット 8 ～ 11: 6H に設定、注記 1 を参照	
UD2（未定義）	EAX、ビット 8 ～ 11: 6H に設定	

表 18-1. インテル® Pentium® プロセッサ以降の IA-32 プロセッサでの新しい命令（続き）

命令	CPUID 表示ビット	導入されたプロセッサ
CMPXCHG8B（8 バイト比較 / 交換、Compare and Exchange 8 Bytes）	EDX、ビット 8	インテル® Pentium® プロセッサ
CPUID（CPU 識別、CPU Identification）	なし、注記 2 を参照	
RDTSC（タイムスタンプ・カウンタ読み取り、Read Time-Stamp Counter）	EDX、ビット 4	
RDMSR（モデル固有レジスタ読み取り、Read Model-Specific Register）	EDX、ビット 5	
WRMSR（モデル固有レジスタ書き込み、Write Model-Specific Register）	EDX、ビット 5	
MMX® 命令	EDX、ビット 23	

注 :

1. RDPMC 命令は P6 ファミリ・プロセッサで採用され、後期モデルのインテル Pentium プロセッサに追加された。この命令は、性格的にはモデル固有であり、アーキテクチャにかかわる命令ではない。
2. CPUID 命令は、すべてのインテル Pentium プロセッサ、P6 ファミリ・プロセッサ、および後期モデルの Intel486 プロセッサに提供されている。EFLAGS レジスタの ID フラグ（ビット 21）がセットおよびクリアできるかどうかによって、CPUID 命令がサポートされているかどうかが分かる。

次に示す命令は、Intel386™ プロセッサに追加された。

- LSS、LFS、LGS（SS、FS、GS レジスタロード）命令
- 条件付きロング・ディスプレースメント・ジャンプ命令
- シングルビット命令
- ビットスキャン命令
- ダブルシフト命令
- 条件付きバイトセット命令
- 符号 / ゼロ拡張付きデータ移動
- 一般化乗算命令
- 制御レジスタとの MOV 命令
- テストレジスタとの MOV 命令（現在は廃止）
- デバッグレジスタとの MOV 命令
- RSM（SMM から再開、Resume from System Management Mode）命令。この命令は、Intel386™ SL プロセッサと Intel486™ SL プロセッサに追加された。

次に示す命令は、インテル® 387 マス・コプロセッサに追加された。

- FPREM1 命令
- FUCOM、FUCOMP、FUCOMPP 命令

18.11.廃止された命令

テストレジスタとのMOV命令は、インテル® Pentium® プロセッサと将来のIA-32プロセッサから削除された。これらの命令を実行すると、無効オペコード例外（#UD）が発生する。

18.12.未定義のオペコード

IA-32 プロセッサ用に新たに定義された命令はすべて、それぞれ前世代プロセッサで予約されていたバイナリ・エンコーディングを使用している。予約されているオペコードを実行しようとすると、必ず無効オペコード（#UD）例外が発生する。結果として、前世代のプロセッサで正しく実行されるプログラムがこれらの命令を誤って実行することはなく、したがって、後世代の IA-32 プロセッサで実行されたときに予期されない結果が生じることもない。

18.13.EFLAGS レジスタの新しいフラグ

『IA-32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』の第3章の「EFLAGS レジスタ」の項に、P6ファミリ・プロセッサ用のEFLAGSレジスタにあるフラグの構成が示してある。P6ファミリ・プロセッサでは、このレジスタに新しいフラグは追加されていない。インテル® Pentium® プロセッサとIntel486™ プロセッサでこのレジスタに追加されたフラグについて、次の項で説明する。

インテルPentiumプロセッサには、EFLAGSレジスタに次のフラグが追加された。

- VIF（仮想割り込みフラグ）、ビット19
- VIP（仮想割り込みペンディング・フラグ）、ビット20
- ID（識別フラグ）、ビット21

Intel486プロセッサには、EFLAGSレジスタにACフラグ（ビット18）が追加された。

18.13.1. EFLAGS フラグによる 32 ビット・インテル® アーキテクチャ・プロセッサ間の識別

次に示すEFLAGSレジスタのビットを使用すると、32ビットIA-32プロセッサ間の識別が可能になる。

- ビット 18（AC フラグ）を使用すると、P6 ファミリ・プロセッサ、インテル ® Pentium® プロセッサ、Intel486™ プロセッサから Intel386™ プロセッサを識別できる。Intel386プロセッサでは、サポートされていないので、このフラグは常にクリアされている。
- ビット 21（ID フラグ）は、アプリケーションが CPUID 命令を実行できるかどうかを示す。このビットをセットおよびクリアできれば、プロセッサは P6 ファミリ・プロセッサかインテル Pentium プロセッサである。その後で CPUID 命令を使用して、どちらのプロセッサであるかが分かる。
- ビット 19（VIF フラグ）とビット 20（VIP フラグ）は、仮想モード拡張をサポートしないプロセッサでは常に0である。インテルPentiumプロセッサより前のすべての32ビット・プロセッサが該当する。

プロセッサの識別の詳細については、『IA-32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』の第 14 章「プロセッサの識別と機能の判定」を参照。

18.14.スタック操作

本節では、各種のIA-32プロセッサ間のスタック操作の相違点について説明する。

18.14.1. PUSH SP 命令

P6 ファミリ・プロセッサ、インテル ® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサ、インテル® 286 プロセッサは、PUSH SP 命令に対して 8086 プロセッサとは異なる値をスタックにプッシュする。32 ビット・プロセッサは、SP レジスタの値を、プッシュ操作の一環としてデクリメントされる前にプッシュする。8086 プロセッサは、SP レジスタの値をデクリメントされた後でプッシュする。プッシュされる値が重要である場合は、PUSH SP 命令に代えて次の3つの命令を使用する。

```
PUSH BP
MOV  BP, SP
XCHG BP, [BP]
```

このコードは、P6ファミリ・プロセッサ、インテルPentiumプロセッサ、Intel486プロセッサ、Intel386プロセッサ、286プロセッサで8086プロセッサのPUSH SP命令として機能する。

18.14.2. スタックにプッシュされた EFLAGS

32 ビット IA-32 プロセッサでの、PUSHF 命令、割り込み、例外によってストアされた EFLAGS レジスタのビット 12 ～ 15（IOPL フィールドと NT フラグ）の値の設定は、8086 プロセッサとインテル® 286 プロセッサでのそれと異なる。それらの相違は次のとおりである。

- 8086 プロセッサ — ビット 12 ～ 15 は常にセット。
- 286 プロセッサ — 実アドレスモードではビット 12 ～ 15 は常にクリア。
- 実アドレスモードでの 32 ビット・プロセッサ — ビット 15（予約ビット）は常にクリア、ビット 12 ～ 14 は最後にロードされた値。

18.15. x87 FPU

本節では、前世代の IA-32 プロセッサとマス・コプロセッサで実行するように設計された浮動小数点ソフトウェアを、統合された x87 FPU を装備したインテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、またはインテル® Pentium® プロセッサに移植する際に直面しなければならない問題について掘り下げて説明する。ソフトウェアから見ると、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサはインテル Pentium プロセッサに非常によく似ている。インテル Pentium プロセッサまたは Intel486™ DX プロセッサ、Intel486™ SX プロセッサ / インテル® 487 SX マス・コプロセッサ・システム、あるいは Intel386™ プロセッサ / インテル® 387 マス・コプロセッサ・システムで実行される浮動小数点ソフトウェアは、少し修正するだけで、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、または P6 ファミリ・プロセッサで実行できる。インテル® 286 プロセッサ / インテル® 287 マス・コプロセッサ・システムまたはインテル® 8086 プロセッサ /8087 マス・コプロセッサ・システムからインテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、またはインテル Pentium プロセッサにコードを直接に移植するには、その他のいくつかの追加問題についても詳細に説明しなければならない。

以降の各項の説明では、用語「32 ビット x87 FPU」は P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、および Intel486 DX プロセッサ、さらに 487 SX プロセッサと 387 プロセッサのマス・コプロセッサのことを意味し、用語「16 ビット IA-32 マス・コプロセッサ」は 287 プロセッサと 8087 プロセッサのマス・コプロセッサのことを意味する。

18.15.1. 制御レジスタ CR0 のフラグ

制御レジスタ CR0 の ET、NE、および MP フラグは、IA-32 プロセッサの整数ユニットと、内部 x87 FPU か外部マス・コプロセッサとの間のインタフェースを制御する。各種の IA-32 プロセッサにおけるこれらのフラグの効果について、次に説明する。

ET（拡張タイプ）フラグ（CR0 レジスタのビット 4）の用途は、Intel386™ プロセッサで、システム内のマス・コプロセッサがインテル® 287 マス・コプロセッサである（フラグがクリアされている場合）かインテル® 387 DX マス・コプロセッサである（フラグがセットされている場合）かを示すことである。このビットは、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは 1 に設定されている。

NE（数値演算エラー例外）フラグ（CR0 レジスタのビット 5）の用途は、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサで、マスクされていない浮動小数点例外が割り込みベクタ 16 を通じて内部的にレポートされる（フラグがセットされている場合）か、外部割り込みを通じて外部的にレポートされる（フラグがクリアされている場合）かを示すことである。ハードウェア・リセットで、NE フラグは 0 に初期化される。したがって、自動内部エラー・レポート・メカニズムを使用するソフトウェアは、このフラグを 1 にセットしなければならない。このフラグは Intel386 プロセッサには存在しない。

インテル® 286 プロセッサと Intel386 プロセッサの場合と同様に、MP（モニタ・コプロセッサ）フラグ（レジスタ CR0 のビット 1）は、x87 FPU のコンテキストが現在実行中のタスクのコンテキストと異なるときに、WAIT/FWAIT 命令または待機中の浮動小数点命令にトラップを発生させるかどうかを決定する。MP フラグと TS フラグがセットされている場合は、WAIT/FWAIT 命令と待機命令はデバイス使用不可能例外（割り込みベクタ 7）を発生させる。286 プロセッサと Intel386 プロセッサでは、MP フラグを使用して、マス・コプロセッサ以外のデバイスで待機させるために WAIT/FWAIT 命令の使用をサポートする。デバイスは、BUSY# ピンを通じてそのステータスをレポートする。P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、および Intel486 プロセッサにはそのようなピンがないので、MP フラグには該当する用途がなく、正常に動作させるためには 1 にセットしておく必要がある。

18.15.2. x87 FPU ステータス・ワード

本項では、各種の IA-32 プロセッサとマス・コプロセッサの x87 FPU ステータス・ワードの相違点、それらの相違の理由、それら相違がソフトウェアに及ぼす影響について説明する。

18.15.2.1.条件コードフラグ（C0 ～ C3）

ここでは、x87 FPU ステータス・ワードのビット 8、9、10、14 を占める条件コードフラグ（C0 ～ C3）の用途上の相違点に関係して説明する。

条件コードフラグは、FINIT 命令の実行後、または 32 ビットの x87 FPU でのハードウェア・リセット後に 0 にクリアされる。16 ビットの IA-32 マス・コプロセッサでは、同じ操作を行っても、これらのフラグは以前の値をそのまま保持している。この操作上

の相違によって、ソフトウェアには影響を与えずに、リセット後の状態のコンシステンシ（一貫性）が保証される。

超越関数命令の結果は、中核的範囲を占める P6 ファミリ・プロセッサとインテル® Pentium® プロセッサでは、最後の桁位置の単位数（ulp）が Intel486™ DX プロセッサおよびインテル® 487 SX マス・コプロセッサと 2 または 3 だけ異なる可能性がある — (『IA-32 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』の第 8 章の「超越関数命令の精度」を参照)。結果として、C1 フラグにセーブされる値も異なる可能性がある。

32 ビット x87 FPU では、未完了の FPREM/FPREM1 命令の実行後には、C0、C1、C3 フラグは 0 にクリアされる。16 ビット IA-32 のマス・コプロセッサでは、同じ演算後にこれらのフラグは以前のまま変わらない。

32 ビット x87 FPU では、C2 フラグは FTAN 命令の未完了フラグの役割を果たす。16 ビット IA-32 のマス・コプロセッサでは、FPTAN 命令に対して C2 フラグは未定義である。インテル® 287 プロセッサと 8087 プロセッサのいずれのプログラムも FPTAN 命令の実行後に C2 を調べないので、この相違はソフトウェアには影響を与えない。それらより後のプロセッサでは、このフラグを使用してオペランド範囲の高速チェックが可能になる。

18.15.2.2.スタック・フォルト・フラグ

32 ビット x87 FPU でマスクされていないスタック・オーバーフローまたはスタック・アンダーフローが発生すると、x87 FPU ステータス・ワードの IE フラグ（ビット 0）と SF フラグ（ビット 6）がセットされて、スタックフォルトが発生したことを示し、条件コードフラグ C1 がセットまたはクリアされて、それぞれオーバーフローであるかまたはアンダーフローであるかを示す。16 ビット IA-32 マス・コプロセッサでマスクされていないスタック・オーバーフローまたはスタック・アンダーフローが発生すると、IE フラグだけがセットされる。これらのプロセッサではビット 6 は予約されている。32 ビット x87 FPU に SF フラグが追加されたが、これはソフトウェアに影響を与えない。既存の例外ハンドラを変更する必要はないが、追加情報を利用できるようにアップグレードするのは差し支えない。

18.15.3. x87 FPU 制御ワード

32 ビット x87 FPU では、無限大制御用として類似クロージャしかサポートされない。これらのプロセッサでは、無限大制御フラグ（x87 FPU 制御ワードのビット 12）はプログラム可能のまま残されているが、働きは失っている。これは、IEEE Standard 754（2 進浮動小数点算術用）に適合させるためになされた変更である。16 ビット IA-32 マス・コプロセッサでは、ビット 12 の設定によって、類似と投影のクロージャがサポートされる。ハードウェア・リセット後のビット 12 のデフォルト値は投影である。投影

型無限大の数値演算を必要とするソフトウェアの場合は、生じる結果が異なる場合がある。

18.15.4. x87 FPU のタグワード

FLDENV 命令、FRSTOR 命令または FXRSTOR（インテル® Pentium® III プロセッサのみ）命令を使用して 32 ビット x87 FPU のタグワードをロードする際に、プロセッサは入ってくるタグを調べ、その位置を空か空でないかだけに分類する。したがって、00、01、10 のタグ値は、プロセッサによって空でない位置を示すものとして解釈される。11 のタグ値は空の位置を示すものとして解釈される。以降の空でないレジスタに対する操作では、常に、レジスタのタグの値ではなく、レジスタの値が調べられる。FSTENV 命令に FSAVE 命令または FXRSTOR（インテル Pentium III プロセッサのみ）命令は、空でないレジスタを調べ、タグに正しい値を設定してからタグワードをストアする。

16 ビット IA-32 マス・コプロセッサでは、各レジスタがアクセスされるたびに、アクセスに先立って対応するタグがチェックされ、レジスタのオペランドのクラスが判定される。レジスタが変更されるたびに、変更の後で、常にレジスタの最新のステータスを反映するようにタグが更新される。ソフトウェアは、タグにレジスタの内容と一致しない値をロードすることがある（例えば、レジスタの内容が有効な値であるのに、タグは特殊を意味している）。その場合、16 ビット IA-32 マス・コプロセッサはタグを優先し、レジスタは調べない。

FLDENV、FRSTOR または FXSAVE 命令を使用してタグを実際のレジスタの内容と異なる（空でない）値に変更する場合は、16 ビット IA-32 マス・コプロセッサで実行するように記述されたソフトウェアが 16 ビット x87 FPU では正しく動作しない可能性がある。

32 ビット x87 FPU のタグワードでの、サポートされていない（疑似 0 やアンノーマルなどの）データ・フォーマットに対するエンコーディングは、IEEE Standard 754 に準拠させるには特殊（10B）である。16 ビット IA-32 マス・コプロセッサでの、疑似 0 またはアンノーマルのフォーマットに対するエンコーディングは有効（00B）であり、その他のサポートされていないデータ・フォーマットに対するエンコーディングは特殊（10B）である。したがって、32 ビット x87 FPU に移植する場合は、疑似 0 フォーマットまたはアンノーマル・フォーマットを有効として認識するコードは変更しなければならない。

18.15.5. データタイプ

本項では、各種のインテル・アーキテクチャの x87 FPU とマス・コプロセッサに対するデータタイプの相違点について説明する。

18.15.5.1.NaNs

32 ビット x87 FPU は、シグナル型 NaNs（SNaNs）とクワイエット型 NaNs（QNaNs）を識別する。これらの x87 FPU は、QNaNs しか生成せず、通常は QNaN が現れても例外を生成しない。FCOM、FIST、FBSTP 命令の場合を除いて、SNaN が現れたときだけ無効操作例外（#I）を生成する。FCOM、FIST、FBSTP 命令は、QNaNs に対しても無効操作例外を生成する。この動作は IEEE Standard 754 に適合する。

16 ビット IA-32 マス・コプロセッサは、1 種類の NaN（QNaN に相当）しか生成しないが、どの種類の NaN が現れても無効操作例外を生成する。

16 ビット IA-32 マス・コプロセッサで実行するように記述されたソフトウェアを 32 ビット x87 FPU に移植する際は、初期化されていないメモリ位置に QNaNs がストアされている場合、初期化されていないメモリ位置が参照されたときに x87 FPU またはマス・コプロセッサをフォルトさせるように、その QNaNs を SNaNs に変更しなければならない。

18.15.5.2.疑似 0、疑似 NaN、疑似無限大、アンノーマルのフォーマット

32 ビット x87 FPU は、疑似 0、疑似 NaN、疑似無限大、アンノーマルの各フォーマットを生成も、サポートもしない。それらの FPU は、これらのフォーマットが算術演算に現れるたびに無効操作例外を生成する。16 ビット IA-32 マス・コプロセッサは、これらのフォーマットの特殊な処理を定義し、かつサポートしている。32 ビット FPU では、これらのフォーマットのサポートは、IEEE Standard 754（2 進浮動小数点算術用）に準拠するために除外されたわけである。

この変更が、16 ビット IA-32 マス・コプロセッサから 32 ビット x87 FPU に移植されたソフトウェアに影響を与えてはならない。32 ビット x87 FPU はこれらのフォーマットを生成せず、したがって、ソフトウェアが明示的にこれらのフォーマットをデータレジスタにロードしない限り、これらのフォーマットが現れたりはしない。ソフトウェアのタグワード内のタグの処理方法には、唯一影響する場合がある。(18.15.4. 項「x87 FPU のタグワード」を参照)。

18.15.6. 浮動小数点例外

本項では、各種の x87 FPU とマス・コプロセッサにおける浮動小数点命令に対する例外処理相違点について説明する。

18.15.6.1.デノーマル・オペランド例外（#D）

デノーマル・オペランド例外がマスクされているときは、32 ビット x87 FPU は、可能な場合はデノーマライズされた数値を自動的にノーマライズする。それに対し、16

ビット IA-32 マス・コプロセッサはデノーマル結果を返す。16 ビット IA-32 マス・コプロセッサでは、デノーマライズされたオペランドをノーマライズするだけのためにデノーマル例外を使用するが、そのプロセッサで実行するように記述されたプログラムは、32 ビット x87 FPU で実行するときは冗長である。そのようなプログラムを 32 ビット x87 FPU で実行する場合は、デノーマル例外をマスクすると性能を向上できる。FPU がデノーマライズされたオペランドをノーマライズすると、浮動小数点プログラムの実行速度が向上する。

16 ビット IA-32 マス・コプロセッサでは、超越関数命令と FXTRACT 命令に対しては、デノーマル・オペランド例外は発生しない。それに対し、32 ビット x87 FPU では、上記の命令に対してこの例外が発生する。これら後者のプロセッサに移植された例外ハンドラは、異なるオペコードに特殊な取り扱いを行う場合に限り変更する必要がある。

18.15.6.2.数値オーバーフロー例外（#O）

32 ビット x87 FPU では、数値オーバーフロー例外がマスクされており、かつ丸めモードが（0 側への）チョップ（丸め）に設定されているときは、結果は正の最大有限数または負の最小有限数になる。16 ビット IA-32 マス・コプロセッサは、マスクされている応答が∞でないときは、オーバーフロー例外を通知しない。つまり、それらの数値演算プロセッサは、丸め制御が 0 への丸めに設定されていないときだけオーバーフローを通知する。丸めが（0 側への）チョップに設定されている場合は、結果は正または負の∞になる。最も一般的な丸めモードでは、この相違は既存のソフトウェアに影響を与えない。

丸めが 0 に向かう（チョップ）場合は、オーバーフロー条件のもとでは、32 ビット x87 FPU 上のプログラムが生成する結果は、16 ビット IA-32 マス・コプロセッサでの結果と比較して、仮数の最下位ビットが異なる。この相違の理由は、IEEE Standard 754 との互換性の保証にある。

オーバーフロー例外がマスクされていないときは、32 ビット x87 FPU では精度例外のフラグが立てられる。結果がスタックにストアされるとき、FPU 制御ワードの精度制御（PC）フィールドにしたがって、またはオペコードにしたがって仮数が丸められる。16 ビット IA-32 マス・コプロセッサでは、精度例外のフラグは立てられず、仮数は丸められない。既存のソフトウェアに対する影響は、結果がスタックにストアされる場合、オーバーフロー条件下では、32 ビット x87 FPU で実行されるプログラムが生成する結果が 16 ビット IA-32 マス・コプロセッサでのそれと異なることである。この相違は例外ハンドラからしかわからない。この相違の理由は、IEEE Standard 754 との互換性の保証にある。

18.15.6.3.数値アンダーフロー例外（#U）

32 ビット x87 FPU でアンダーフロー例外がマスクされているときは、結果が非常に小さく、デノーマライズ処理の結果精度が失われるという 2 つの条件が共に成立すると、アンダーフロー例外が通知される。アンダーフロー例外がマスクされておらず、命令が結果をスタックにストアすることになっているときは、指数の調整後、仮数が該当する精度に丸められる（つまり、PC によって制御される命令に対しては FPU 制御ワードの PC フラグにしたがって丸められ、そうでない命令に対しては拡張精度に丸められる）。

16 ビット IA-32 マス・コプロセッサでアンダーフロー例外がマスクされており、丸めが 0 に向かうときは、非常に小さい結果に対しては、精度が失われるかどうかに関係なくアンダーフロー例外のフラグが立てられる。アンダーフロー例外がマスクされておらず、デスティネーションがスタックであるときは、仮数は丸められず、以前のまま維持される。

アンダーフロー例外がマスクされているときは、この相違は既存のソフトウェアに影響を与えない。丸めが 0 に向かうときの方が、アンダーフロー例外の発生頻度が低い。

アンダーフロー例外がマスクされていないときは、結果がスタックにストアされる場合、32 ビット x87 FPU がアンダーフロー条件発生時に生成する結果は、16 ビット IA-32 マス・コプロセッサでのそれと異なる。相違は、仮数の最下位ビットだけに現れ、例外ハンドラからしかわからない。

18.15.6.4.例外の優先順位

32 ビット x87 FPU では、マスクされていてもいなくても、デノーマル・オペランド例外の優先順位に相違はない。16 ビット IA-32 マス・コプロセッサでは、デノーマル・オペランド例外がマスクされていないときは、この例外は他のすべての例外に優先する。この相違によって既存のソフトウェアに影響を与えないが、Intel486™ プロセッサとインテル® 387 マス・コプロセッサではデノーマライズされたオペランドの一部の不要なノーマライズ処理が防止される。

18.15.6.5.FPU 例外用の CS レジスタと EIP レジスタ

32 ビット x87 FPU では、浮動小数点例外用として CS レジスタと EIP レジスタからセーブされた値は、浮動小数点命令の前に挿入されているプリフィックスを指す。それに対し、8087 マス・コプロセッサでは、セーブされた CS レジスタと IP レジスタは浮動小数点命令を指す。

18.15.6.6.FPU のエラー信号

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサへの浮動小数点エラー信号は、割り込みコントローラを通じては伝達されず、インテル® 387 プロセッサ、インテル® 287 プロセッサ、または 8087 マス・コプロセッサからの INT# 信号によって伝達される。8086 プロセッサがこの 8087 割り込みに対して別の例外を使用する場合は、両例外ベクタが浮動小数点エラー例外ハンドラを呼び出す必要がある。浮動小数点エラー例外ハンドラの命令には、割り込みコントローラを使用する場合は一部削除する必要があるものもある。P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサには、外部論理を加えて、多くのパーソナル・コンピュータに使用されている割り込みメカニズムのエミュレーションのレポートをサポートする信号がある。

P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサでは、未定義の浮動小数点オペコードは無効オペコード例外（#UD、割り込みベクタ 6）を生成する。未定義の浮動小数点オペコードは、正しい浮動小数点オペコードと同様に、制御レジスタ CR0 の TS フラグか EM フラグがセットされていると、デバイス使用不可例外（#NM、割り込みベクタ 7）を生成する。P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサは、未定義の浮動小数点オペコードが現れても、浮動小数点エラー条件の有無を調べない。

18.15.6.7.FERR# ピンのアサーション

浮動小数点例外の処理に MS-DOS* 互換モードを使用するときは、FERR# ピンを外部割り込みコントローラの入力に接続しなければならない。そうすると、FERR# 出力が割り込みコントローラの入力をドライブしたとき外部割り込みが発生し、今度は、割り込みコントローラがプロセッサの INTR ピンをドライブする。

P6 ファミリ・プロセッサと Intel386™ プロセッサの場合は、マスクされていない浮動小数点例外が発生するたびに、例外を発生させた命令の実行の終了時に FERR# ピンをアサートする。インテル® Pentium® プロセッサと Intel486™ プロセッサの場合は、マスクされていない浮動小数点例外が発生したとき、例外を発生させた浮動小数点命令の終了時か、またはその次の浮動小数点命令が実行される直前に FERR# ピンをアサートできる。（次の浮動小数点命令は、ペンディングのマスクされていない例外の処理が終了するまで実行されないことになるので注意しなければならない。）MS-DOS 互換モードを使用した浮動小数点例外の処理に必要なメカニズムの全般については、『IA-32 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』の付録 D を参照。

18.15.6.8.デノーマル数に対する無効操作例外

32 ビット x87 FPU では、FSQRT、FDIV、または FPREM 命令の実行時にデノーマル値が現れたとも、BCD または整数への変換に際しても、無効操作例外は発生しない。操作はまず値のノーマライズから始まる。16 ビット IA-32 マス・コプロセッサでは、この状況が現れると、無効操作例外が発生する。この相違は既存のソフトウェアに影響を与えない。32 ビット x87 FPU で実行されるソフトウェアは、16 ビット IA-32 マス・コプロセッサでトラップが発生した場合に実行を継続する。この変更の理由は、例外の発生を除去することにあった。

18.15.6.9.アライメント・チェック例外（#AC）

アライメント・チェックがイネーブルにされている場合は、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは、プログラムまたはプロシージャが特権レベル 3 で実行されているときにアライメントが合わないデータ・オペランドが現れると、FSAVE/FNSAVE、FXSAVE、FRSTOR、FXRSTOR 命令のスタック部分を除いて、アライメント・チェック例外（#AC）が発生する。

18.15.6.10.FLDENV 命令実行中のセグメント不在例外

Intel486™ プロセッサでは、FLDENV 命令の実行中にセグメント不在例外（#NP）が発生すると、環境の一部がロードされ、他の部分はロードされない状況が発生することがある。そのような場合は、FPU 制御ワードは値 007FH のままになる。F6 ファミリ・プロセッサとインテル® Pentium® プロセッサは、内部状態を更新する前に環境の最初と最後のバイトを読み取ると、内部状態が常時正しく保たれるように保証する。

18.15.6.11.デバイス使用不可能例外（#NM）

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは、2.5. 節「制御レジスタ」の表 2-1. と第 5 章の「割り込み 7 ーデバイス使用不可例外（#NM）」の説明にしたがって、デバイス使用不可例外（#NM、割り込み 7）が発生する。

18.15.6.12.コプロセッサ・セグメント・オーバーラン例外

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは、コプロセッサ・セグメント・オーバーラン例外（割り込み 9）は発生しない。
インテル® 387 マス・コプロセッサが割り込み 9 を発生させるような状況では、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサは単に命令を打ち切る。したがって、セグメント・オーバーランを検出しないで見過ごさないように、浮動小数点のセーブ領域は TSS と同じページに配置するように推奨する。そのような配置では、オペレーティング・システムによるタスクの切り替えの間、FLDENV 命令、

FRSTOR 命令または FXRSTOR 命令の実行時にページフォルトが発生しても、FPU 環境が失われなくなる。

18.15.6.13.一般保護例外（#GP）

浮動小数点オペランドの開始アドレスがセグメントのサイズ外に出た場合に、一般保護例外（#GP、割り込み 13）が発生する。そのようなプログラミング・エラーをレポートするために、例外ハンドラを取り入れなければならない。

18.15.6.14.浮動小数点エラー例外（#MF）

実アドレスモードと保護モード（仮想 8086 モード以外）では、割り込みベクタ 16 は浮動小数点例外ハンドラを指さなければならない。仮想 8086 モードでは、割り込みベクタの別の位置を浮動小数点例外に対応させるように、仮想 8086 モニタをプログラムできる。

18.15.7. 浮動小数点命令に対する変更

本項では、各種インテル FPU とマス・コプロセッサのアーキテクチャ間の浮動小数点命令に関する相違点、それらの相違の理由、ソフトウェアへの影響について説明する。

18.15.7.1.FDIV、FPREM、FSQRT 命令

32 ビット x87 FPU では、デノーマライズされたオペランドの操作をサポートし、検出されたときは、IEEE Standard 754 との互換性を保証するためにアンダーフロー例外を生成できる。16 ビット IA-32 マス・コプロセッサでは、デノーマライズされたオペランドの操作を行わず、アンダーフロー結果も返さない。その代わりに、これらの数値演算プロセッサはアンダーフロー条件を検出すると無効操作例外を生成する。既存のアンダーフロー例外ハンドラは、異なるオペコードに対して異なる取り扱いを行う場合しか変更を必要としない。さらに、発生する無効操作例外の数が少なくなる可能性がある。

18.15.7.2.FSCALE 命令

32 ビット x87 FPU では、スケーリング（ソース）オペランドの範囲に制限がない。（0 < ｜ ST(1) ｜ < 1）の場合は、スケーリング・ファクタは 0 であり、したがって、ST(0) は変わらない。丸め結果が正確でない場合、または精度が失われた（アンダーフローがマスクされている）場合は、精度例外が通知される。16 ビット IA-32 マス・コプロセッサでは、スケーリング・オペランドの範囲が制限される。（0 < ｜ ST(1) ｜ < 1）の場合は、結果は不確定であり、例外は通知されない。この相違による既存のソフト

ウェアへの影響は、（0 < ｜ ST(1) ｜ < 1）のときは、32 ビット FPU と 16 ビット・マス・コプロセッサとで伝達される結果が異なることである。

18.15.7.3.FPREM1 命令

32 ビット x87 FPU は、IEEE Standard 754 にしたがって部分剰余を計算する。この命令は、16 ビット IA-32 マス・コプロセッサには存在しない。FPREM1 命令が使用可能であるかないかは、既存のソフトウェアには影響を与えない。

18.15.7.4.FPREM 命令

32 ビット x87 FPU では、ステータス・ワードの条件コードフラグ C0、C3、C1 は FPREM 命令実行後の商の下位 3 ビットを正しく反映する。16 ビット IA-32 マス・コプロセッサでは、（N ≧ 1）でありかつ M が 1 または 2 の場合に（$64^N + M$）の縮小を実行すると、それらの商のビットは正しくなくなる。この相違は既存のソフトウェアに影響を与えない。バグまでも取り扱うようなソフトウェアは影響を受けてはならない。

18.15.7.5.FUCOM、FUCOMP、FUCOMPP 命令

FUCOM、FUCOMP、FUCOMPP 命令の実行時には、32 ビット x87 FPU は IEEE Standard 754 規格にしたがってオーダリングのない比較を実行する。これらの命令は、16 ビット IA-32 マス・コプロセッサには存在しない。これらの新しい命令が使用可能であるかないかは、既存のソフトウェアに影響を与えない。

18.15.7.6.FPTAN 命令

32 ビット x87 FPU では、FPTAN 命令のオペランドの範囲は、前世代のマス・コプロセッサの場合よりも制限がはるかに小さい（｜ ST(0) ｜ < 2^{63}）。この命令では、より精度が高い π/4 の内部定数を内部的に使用してオペランドを縮小する。16 ビット IA-32 マス・コプロセッサでは、オペランドの範囲は（｜ ST(0) ｜ < π/4）に制限される。FPREM を使用して、オペランドをこの範囲に縮小しなければならない。この変更は既存のソフトウェアに影響を与えない。

18.15.7.7.スタック・オーバーフロー

32 ビット x87 FPU では、無効操作例外がマスクされているときに FPU スタック・オーバーフローが発生した場合、実行されている命令によって、FPU は実数、整数、または BCD 整数の無限大値をデスティネーション・オペランドに返す。16 ビット IA-32 マス・コプロセッサでは、オペランドはスタック・オーバーフロー後も元の値のままであるが、レジスタ ST(1) にロードされる。この相違は既存のソフトウェアに影響を与えない。

18.15.7.8.FSIN、FCOS、FSINCOS 命令

32 ビット x87 FPU では、これらの命令は 3 つの一般的な三角関数を実行する。16 ビット IA-32 マス・コプロセッサには、これらの命令は存在しない。これらの命令が使用可能であるかないかは既存のソフトウェアに影響を与えないが、これらの命令を使用すると性能が向上する。

18.15.7.9.FPATAN 命令

32 ビット x87 FPU では、FPATAN 命令のオペランドの範囲に制限はない。16 ビット IA-32 マス・コプロセッサでは、レジスタ ST(0) のオペランドの絶対値はレジスタ ST(1) のオペランドの絶対値よりも小さくなければならない。この相違は既存のソフトウェアに影響を与えない。

18.15.7.10.F2XM1 命令

32 ビット x87 FPU の方が、サポートされる F2XM1 命令のオペランドの範囲が広い（$-1 < ST(0) < +1$）。16 ビット IA-32 マス・コプロセッサでサポートされるオペランドの範囲は（$0 \leqq ST(0) \leqq 0.5$）である。この相違は既存のソフトウェアに影響を与えない。

18.15.7.11.FLD 命令

32 ビット x87 FPU では、FLD 命令は算術演算命令ではないので、この命令を使用して拡張実数値をロードしたときにデノーマル・オペランド例外は発生しない。16 ビット IA-32 マス・コプロセッサでは、上記の状況でデノーマル・オペランド例外が発生する。この相違は既存のソフトウェアに影響を与えない。

32 ビット x87 FPU では、単精度または倍精度実数フォーマットのデノーマル値をロードすると、その値が拡張実数フォーマットに変換される。16 ビット IA-32 マス・コプロセッサでデノーマル値をロードすると、その値はアンノーマル値に変換される。その次の命令が FXTRACT または FXAM である場合は、32 ビット x87 FPU での結果は 16 ビット IA-32 マス・コプロセッサでの結果と異なる。この変更は、IEEE Standard 754 との互換性を保証するために行われたものである。

32 ビット x87 FPU では、単精度または倍精度実数フォーマットの SNaN をロードすると、無効操作例外が発生する。16 ビット IA-32 マス・コプロセッサでは、シグナル型 NaN をロードしたとき例外は発生しない。16 ビット・マス・コプロセッサ・ソフトウェアの無効操作例外ハンドラは、ソフトウェアを 32 ビット FPU に移植するときにこの条件を処理するように更新する必要がある。この変更は、IEEE Standard 754 との互換性を保証するために行われたものである。

18.15.7.12.FXTRACT 命令

32 ビット x87 FPU では、FXTRACT 命令のオペランドが 0 の場合は、ゼロ除算例外がレポートされ、レジスタ ST(1) に – ∞が転送される。オペランドが + ∞の場合は、例外はレポートされない。16 ビット IA-32 マス・コプロセッサでは、オペランドが 0 の場合は、レジスタ ST(1) に 0 が転送され、例外はレポートされない。オペランドが + ∞の場合は、無効操作例外がレポートされる。これらの相違点は既存のソフトウェアに影響を与えない。ソフトウェアは通常 0 と∞をバイパスする。この変更は、"logb" 関数の完全なサポートを求める IEEE 754 勧告によるものである。

18.15.7.13.定数ロード命令

32 ビット x87 FPU では、定数ロード命令に対して丸め制御が有効である。16 ビット IA-32 マス・コプロセッサでは、丸め制御は有効でない。FLDPI、FLDLN2、FLDLG2、FLDL2E 命令の結果は、丸め制御が最近値への丸めまたは + ∞に向かう丸めに設定されているときは、16 ビット IA-32 マス・コプロセッサの場合と同じである。丸め制御が最近値への丸め、– ∞に向かう丸めまたは 0 側への丸めに設定されているときは、それらの結果は FLDL2T の場合と同じである。FLDPI、FLDLN2、FLDLG2、FLDL2E 命令に対して丸め制御が – ∞に向かう丸めまたは 0 側への丸めに設定されている場合、結果は、16 ビット IA-32 マス・コプロセッサの場合と仮数の最下位ビットが異なる。+ ∞に向かう丸めが指定されている場合は、FLDL2T 命令に対しては結果が異なる。これらの変更は、IEEE Standard 754（2 進浮動小数点算術用）勧告との互換性を保証するために用意されたものである。

18.15.7.14.FSETPM 命令

32 ビット x87 FPU では、FSETPM 命令は NOP（ノー・オペレーション）として取り扱われる。この命令は、インテル® 287 マス・コプロセッサにプロセッサが保護モードであることを知らせる。この変更は既存のソフトウェアに影響を与えない。32 ビット x87 FPU は、保護モードであってもなくても、すべてのアドレス指定と例外ポインタ情報を処理する。

18.15.7.15.FXAM 命令

32 ビット x87 FPU では、FXAM 命令の実行時に空レジスタが現れた場合、x87 FPU は C0 ～ C3 の組み合わせ値として 1101 も 1111 も生成しない。16 ビット IA-32 マス・コプロセッサは、その他の組み合わせはもとより、これらの組み合わせ値も生成できる。この相違は既存のソフトウェアに影響を与えない。この相違によって、再現可能な結果を生じるように性能が向上する。

18.15.7.16.FSAVE 命令と FSTENV 命令

32 ビット x87 FPU では、前の浮動小数点命令がメモリを参照しなかった場合、その次の FSAVE または FSTENV によってストアされるメモリ・オペランド・ポインタのアドレスは不確定である。

18.15.8. 超越関数命令

中核的範囲をなす P6 ファミリ・プロセッサとインテル® Pentium® プロセッサの超越関数命令に対する浮動小数点結果は、Intel486™ プロセッサの場合とは約 2 または 3 ulp 異なることがある（『IA-32 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』の第 8 章の「超越関数命令の精度」を参照）。結果として、ステータス・ワードの条件コードフラグ C1 が異なることがある。アンダーフローとオーバーフローの厳密なしきい値が 2 または 3 ulp 変化する。P6 ファミリ・プロセッサとインテル Pentium プロセッサの結果には、偶数の近似値に丸めるときは 1 ulp 未満の、またその他のモードで丸めるときは 1.5 ulp 未満のワースト・ケース・エラーが生じる。超越関数命令は、命令がサポートする領域全体にわたって、入力オペランドに関して均一であるように保証されている。

超越関数命令は、32 ビット x87 FPU では、切り上げフラグ（C1）に異なる結果を生成することがある。16 ビット IA-32 マス・コプロセッサでは、これらの命令に対して切り上げフラグは定義されていない。この相違は既存のソフトウェアに影響を与えない。

18.15.9. 古くなった命令

8087 マス・コプロセッサの FENI 命令と FDISI 命令、インテル® 287 マス・コプロセッサの FSETPM 命令は、32 ビット x87 FPU では整数 NOP 命令として取り扱われる。これらのオペコードが命令ストリーム内で検出されても、なんら特別な操作は実行されず、どの内部状態も変わらない。

18.15.10.WAIT/FWAIT プリフィックスに関する相違点

Intel486™ プロセッサでは、浮動小数点命令（自身をその前の浮動小数点命令と自動的に同期させる命令）の前に WAIT/FWAIT 命令が挿入されると、それらの命令はノー・オペレーションとして取り扱われる。前の浮動小数点命令からのペンディングの浮動小数点例外は、WAIT/FWAIT 命令で処理されないで、その次の浮動小数点命令で処理される。そのような場合は、Intel486 プロセッサでは、浮動小数点例外からのレポートは、P6 ファミリ・プロセッサの FPU またはインテル® Pentium® プロセッサの FPU の場合やインテル® 387 マス・コプロセッサの場合より 1 命令分遅く現れることがある。

18.15.11.セグメント / ページ境界でのオペランドの分割

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサの FPU では、記述するオペランドの前半がページまたはセグメント内側にあって、後半が外側に出ているときは、メモリフォルトが発生して、前半はストアされるが後半はストアされないことがある。この状況では、インテル® 387 マス・コプロセッサは何もストアしない。

18.15.12.FPU 命令の同期

32 ビット x87 FPU では、すべての浮動小数点命令が自動的に同期される。つまり、プロセッサは、自動的にそれぞれ前の浮動小数点命令の実行が終了するまで待ってから次の浮動小数点命令を実行する。この同期を保証するために、明示的に WAIT/FWAIT 命令を使用する必要はない。8087 マス・コプロセッサについては、同期を保証するために、各浮動小数点命令の前に明示的に待機を挿入する必要がある。明示的な WAIT 命令を使用している 8087 プログラムは、アセンブルし直さないで 32 ビット IA-32 プロセッサで完全に実行され、それらの WAIT 命令は不要になる。

18.16.シリアル化命令

次の命令が実行される前に、確実にフラグ、レジスタ、メモリの修正を完了させる（または、P6 ファミリ・プロセッサの用語によれば「マシン状態に委ねられる」）ように、命令の実行をシリアル化する目的のためにいくつかの命令が定義されている。P6 ファミリ・プロセッサでは、分岐予測手法とオーダリング外実行手法を使用して性能向上をはかっているので、実行された命令の結果がマシン状態に委ねられるまでは命令の実行は一般的にシリアル化されない（『IA-32 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』の第 2 章「IA-32 インテル® アーキテクチャの概説」を参照）。

結果として、次の命令を実行する前にそれより前のすべての命令の実行を完了させることが必要不可欠であるようなプログラムやタスク内の位置（例えば、分岐、プロシージャの最後、またはマルチプロセッサに依存するコード）には、シリアル化命令を追加すると役立つ。シリアル化命令の詳細については、7.4. 節「シリアル化命令」を参照。

18.17.FPU とマス・コプロセッサの初期化

表 9-1. に、電源投入、リセット、INIT 命令実行、または FINIT/FNINIT 命令実行の後の、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサの FPU の状態、インテル® 387 マス・コプロセッサとインテル® 287 コプロセッサの状態が示してある。以降で、x87 FPU とマス・コプロセッサの初期化に関する互換性上の追加情報について説明する。

18.17.1. インテル® 387 プロセッサとインテル® 287 プロセッサのマス・コプロセッサの初期化

Intel386™ プロセッサは、リセットされた後に、RESET# 信号の立ち下がりエッジの少し後で、最初の浮動小数点命令の実行前に ERROR# 入力をサンプリングして、そのプロセッサ・タイプ（インテル® 287 プロセッサまたはインテル® 387 DX マス・コプロセッサ）を識別する。287 コプロセッサは、ハードウェア・リセット後に ERROR# 出力を非アクティブ状態に維持し、387 コプロセッサは、ハードウェア・リセット後に ERROR# 出力をアクティブ状態に維持する。

ハードウェア・リセット後または FINIT/FNINIT 命令実行後に、387 マス・コプロセッサは信号でエラー条件の存在を通知する。それに対し、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、および Intel486™ プロセッサは、287 コプロセッサと同様に、エラー条件を通知しない。

18.17.2. Intel486™ SX プロセッサとインテル® 487 SX マス・コプロセッサの初期化

Intel486™ SX プロセッサとインテル® 487 SX マス・コプロセッサの初期化時には、初期化ルーチンはマス・コプロセッサの有無を調べ、制御レジスタ CR0 の FPU に関係するフラグ（EM、MP、および NE）を適切に設定する必要がある（これらのフラグの詳細については、2.5. 節「制御レジスタ」を参照）。表 18-2. に、マス・コプロセッサが存在するときのこれらのフラグの推奨設定を示す。FSTCW 命令は、Intel486 SX マイクロプロセッサに対しては値 FFFFH を、487 SX マス・コプロセッサに対しては値 037FH を設定する。

表 18-2. Intel486™ SX マイクロプロセッサ / インテル® 487 SX マス・コプロセッサ・システムの EM、MP、NE フラグの推奨値

CR0 フラグ	Intel486™ SX プロセッサのみの場合	インテル® 487 SX マス・コプロセッサが存在する場合
EM	1	0
MP	0	1
NE	1	MS-DOS* システムに対しては 0 ユーザ定義例外ハンドラに対しては 1

レジスタ CR0 の EM フラグと MP フラグは、表 18-3. に示すように解釈される。

表 18-3. EM フラグと MP フラグの解釈

EM	MP	解釈
0	0	浮動小数点命令は FPU に渡される。WAIT/FWAIT や他の待機型命令は TS を無視する。
0	1	浮動小数点命令は FPU に渡される。WAIT/FWAIT や他の待機型命令は TS をテストする。
1	0	浮動小数点命令はエミュレータにトラップされる。WAIT/FWAIT や他の待機型命令は TS を無視する。
1	1	浮動小数点命令はエミュレータにトラップされる。WAIT/FWAIT や他の待機型命令は TS をテストする。

次に、システムを初期化し、Intel486 SX プロセッサ /487 SX マス・コプロセッサの有無をチェックするコード・シーケンス例を示す。

```
fninit
fstcw mem_loc
mov ax, mem_loc
cmp ax, 037fh
jz Intel487_SX_Math_CoProcessor_present;ax=037fh
jmp Intel486_SX_microprocessor_present;ax=ffffh
```

487 SX マス・コプロセッサが存在しない場合は、次に示すコードを実行して Intel486 SX プロセッサのレジスタ CR0 を設定できる。

```
mov eax, cr0
and eax, fffffffdh  ;make MP=0
or eax, 0024h       ;make EM=1, NE=1
mov cr0, eax
```

この初期化により、どの浮動小数点命令でもデバイス使用不可能例外（#NM）、割り込み 7 が発生する。そこで、ソフトウェア・エミュレーションが制御を受け取り、上記の一連の命令を実行する。このコードは、システムに 487 SX マス・コプロセッサが

存在する場合は必要ない。その場合は、Intel486 SX マイクロプロセッサ用の一般的な初期化ルーチンで十分である。

さらに、Intel486 SX プロセッサ・ベースの 487 SX マス・コプロセッサを装備したシステムを設計するに当たっては、タイミング・ループをクロック速度と命令当たりクロック数から独立させる必要がある。これを達成する 1 つの方法は、それらのループをソフトウェア（例えば BIOS）ではなくハードウェアに実現することである。

18.18. 制御レジスタ

ここでは、各種のプロセッサ・ファミリの 32 ビット IA-32 に新たに導入された制御レジスタ、制御レジスタフラグ、制御レジスタ・フィールドについて説明する。これらのフラグやフィールドが制御レジスタのどの位置にあるかについては、図 2-5. を参照。

インテル® Pentium® III プロセッサでは、制御レジスタ CR4 に新しい制御フラグが 1 つ導入された。

- OSXMMEXCPT（ビット 10）－オペレーティング・システムがマスクされていない SIMD 浮動小数点例外をサポートしている場合は、このビットがセットされる。

インテル® Pentium® II プロセッサでは、制御レジスタ CR4 に新しい制御フラグが 1 つ導入された。

- OSFXSR（ビット 9）－オペレーティング・システムは、コンテキスト・スイッチ時のインテル Pentium III プロセッサ・ステートのセーブとリストアをサポートしている。

インテル® Pentium® Pro プロセッサでは、制御レジスタ CR4 に新しい制御フラグが導入された。

- PAE（ビット 5）－ 物理アドレス拡張。セットされると、36 ビットの物理アドレスを参照するためのページング・メカニズムをイネーブルにする。クリアされると、物理アドレスは 32 ビットに制限される（18.19.1.1. 項「物理メモリアドレス指定拡張」を参照）。
- PGE（ビット 7）－ ページ・グローバル・イネーブル。頻繁に使用または共用されるページがタスクの切り替え時にフラッシュされるのを禁止する（18.19.1.2. 項「グローバル・ページ」を参照）。
- PCE（ビット 8）－ 性能モニタリング・カウンタ・イネーブル。任意の保護レベルで RDPMC 命令の実行をイネーブルにする。

ハードウェア・リセット後の CR4 の内容は 0H になる。

制御レジスタ CR4 はインテル® Pentium® プロセッサで導入された。このレジスタには、インテル Pentium プロセッサに新たに設けられた拡張機能をイネーブルにするフラグが入っている。

- VME ― 仮想 8086 モード拡張。仮想 8086 モードで仮想割り込みフラグに対するサポートをイネーブルにする（16.3. 節「仮想 8086 モードでの割り込み / 例外処理」を参照）。
- PVI ― 保護モード仮想割り込み。保護モードで仮想割り込みフラグに対するサポートをイネーブルにする（16.4. 節「保護モード仮想割り込み」を参照）。
- TSD ― タイムスタンプ・ディスエーブル。RDTSC 命令の実行を、特権レベル 0 で実行されるプロシージャに制限する。
- DE ― デバッグ拡張。性能向上のため、デバッグレジスタ DR4 と DR5 が参照されたときに、未定義オペコード（#UD）例外を発生させる（15.2.2. 項「デバッグレジスタ DR4 および DR5」を参照）。
- PSE ― ページサイズ拡張。セットされると、4M バイト・ページをイネーブルにする（3.6.1. 項「ページングのオプション」を参照）。
- MCE ― マシン・チェック・イネーブル。マシンチェック例外をイネーブルにし、特定のハードウェア・エラー条件の例外処理を可能にする（第 14 章「マシン・チェック・アーキテクチャ」を参照）。

Intel486™ プロセッサでは、制御レジスタ CR0 に 5 つの新しいフラグが導入された。

- NE ― 数値エラー。浮動小数点数値エラーレポート用の標準メカニズムをイネーブルにする。
- WP ― 書き込み保護。ユーザレベルのページをスーパーバイザ・モード・アクセスに対して書き込み保護する。
- AM ― アライメント・マスク。アライメント・チェックを実行するかどうかを制御する。AC（アライメント・チェック）フラグと連携して動作する。
- NW ― ノット・ライトスルー。クリアされると、ライトスルーとキャッシュ無効化サイクルをイネーブルにする。セットされると、無効化サイクルと、キャッシュでヒットするライトスルーをディスエーブルにする。
- CD ― キャッシュ・ディスエーブル。クリアされると、内部キャッシュをイネーブルにし、セットされるとキャッシュをディスエーブルにする。

Intel486 プロセッサでは、制御レジスタ CR3 に 2 つの新しいフラグが導入された。

- PCD ― ページ・レベル・キャッシュ・ディスエーブル。ページングがイネーブルになっているとき、割り込みアクノレッジ・サイクルなどの、ページングされないバスサイクル中に、このフラグの状態が PCD# ピンにドライブされる。PCD# ピンは、サイクルごとに外部キャッシュ内のキャッシング制御に使用される。

- PWT ― ページ・レベル・ライトスルー。ページングがイネーブルになっているとき、割り込みアクノレッジ・サイクルなどの、ページングされないバスサイクル中に、このフラグの状態が PWT# ピンにドライブされる。PWT# ピンは、サイクルごとに外部キャッシュ内のライトスルー制御に使用される。

18.19. メモリ管理機能

以降の各項では、各種の IA-32 プロセッサで使用可能な新しいメモリ管理機能について説明し、また一部の互換性上の相違点について説明する。

18.19.1. 新しいメモリ管理制御フラグ

インテル® Pentium® Pro プロセッサでは、3 つの新しいメモリ管理機能、つまり物理メモリアドレス指定拡張、ページ・テーブル・エントリ内のグローバル・ビット、ページサイズ拡張に対する一般サポートが導入された。これらの機能は、保護モードの動作時にのみ使用可能である。

18.19.1.1. 物理メモリアドレス指定拡張

制御レジスタ CR4 の新しい PAE（物理アドレス拡張）フラグ（ビット 5）は、プロセッサに対して 4 つの追加アドレスラインをイネーブルにして、36 ビットの物理アドレスを使用可能にする。このオプションは、ページングがイネーブルになっているときだけ使用することができ、拡張された物理アドレス範囲をサポートするために新たに設けられたページ・テーブル・メカニズム（3.8. 節「PAE ページング・メカニズムを使用した 36 ビット 物理アドレス指定」を参照）を使用する。

18.19.1.2. グローバル・ページ

制御レジスタ CR4 の新しい PGE（ページ・グローバル・イネーブル）フラグ（ビット 7）は、使用頻度の高いページをトランスレーション・ルックアサイド・バッファ（TLB）からフラッシュさせないようにするメカニズムを提供する。このフラグがセットされると、ページ・ディレクトリまたはページ・テーブル・エントリ内のグローバル・フラグをセットすれば、（カーネル・プロシージャや共通データテーブルをストアしているページなどの）使用頻度の高いページをグローバルとマークできる。

タスクの切り替え時、または制御レジスタ CR3 への書き込み時（これらは通常 TLB のフラッシュを生じる）に、グローバルとマークされている TLB 内のエントリはフラッシュされない。このようにしてページをグローバルとマークすると、使用頻度の高いページで TLB が見つからないといった事態が発生せず、TLB の不要な再ロードを防止できる。このメカニズムの詳細については、3.11. 節「トランスレーション・ルックアサイド・バッファ（TLB）」を参照。

18.19.1.3.ページサイズ拡張

P6 ファミリ・プロセッサでは大きなページサイズをサポートしている。この機能は、制御レジスタ CR4 の PSE（ページサイズ拡張）フラグ（ビット 4）でイネーブルにできる。このフラグがセットされると、プロセッサは、通常のページングを使用しているときは 4K バイトか 4M バイトのページサイズをサポートし、物理アドレス拡張を使用しているときは 4K バイトと 2M バイトのページサイズをサポートする。ページサイズ拡張の詳細は 3.6.1. 項「ページングのオプション」を参照。

18.19.2. CD キャッシュ制御フラグと NW キャッシュ制御フラグ

制御レジスタ CR0 の CD フラグと NW フラグは Intel486™ プロセッサで導入された。P6 ファミリ・プロセッサとインテル® Pentium® プロセッサでは、これらのフラグはデータ・キャッシュにライトバック方式を制御するために使用される。Intel486 プロセッサでは、ライトスルー方式を制御するために使用される。P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサにおけるこれらのビットの比較については、10-5. を参照。キャッシングの全般については、第 10 章「メモリ・キャッシュ制御」を参照。

18.19.3. ディスクリプタのタイプと内容

ディスクリプタ・テーブルのスペースを管理するオペレーティング・システム・コードは、未使用のエントリを識別するために、ディスクリプタ・テーブル・エントリのアクセス権フィールドに無効な値を設定することがよくある。アクセス権の値 80H と 00H は、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサ、インテル® 286 プロセッサに対しては依然無効である。これら以外の値で、286 プロセッサでは無効となる値でも、ビットの用途が定義されているものであれば 32 ビット・プロセッサで有効になる。

18.19.4. セグメント・ディスクリプタ・ロードの変更

Intel386™ プロセッサでは、セグメント・ディスクリプタをロードすると、ディスクリプタのアクセス済みビットをセットするために読み取りと書き込みがロックされる。P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサでは、ビットがまだセットされていない場合に限り、読み取りと書き込みがロックされる。

18.20. デバッグ機能

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサでは、ブレークポイント用に Intel486™ プロセッサのデバッグ・サポートの拡張機能を取り入れている。それらの新しいブレークポイント機能を使用するには、制御レジスタ CR4 の DE フラグをセットする必要がある。

18.20.1. デバッグレジスタ DR6 の相違

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサでは、デバッグ・ステータス・レジスタ DR6 の予約ビット 12 に 1 を書き込めないが、Intel486™ プロセッサでは、このビットに 1 を書き込むことができる。電源投入後またはハードウェア・リセット後にこのレジスタを設定する際の相違については、表 9-1. を参照。

18.20.2. デバッグレジスタ DR7 の相違

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサは、次のようにして、デバッグ制御レジスタ DR7 の R/W0 ～ R/W3 フィールドによってブレークポイント・アクセスのタイプを判定する。

00 命令実行でのみブレーク。

01 データ書き込みでのみブレーク。

10 制御レジスタ CR4 の DE フラグがクリアされている場合は未定義。制御レジスタ CR4 の DE フラグがセットされている場合は、I/O 読み取りまたは書き込みでブレークするが、命令フェッチではブレークしない。

11 データ読み取りまたは書きみでブレークするが、命令フェッチではブレークしない。

P6 ファミリ・プロセッサとインテル Pentium プロセッサでは、予約ビット 11、12、14、15 は 0 に配線されている。ただし、Intel486™ プロセッサでは、ビット 12 はセット可能である。電源投入後またはハードウェア・リセット後にこのレジスタを設定する際の相違については、表 9-1. を参照。

18.20.3. デバッグレジスタ DR4 および DR5

DR4 レジスタと DR5 レジスタは、マニュアルには予約と記載されているが、前世代のプロセッサは、これらのレジスタへの参照をそれぞれデバッグレジスタ DR6 と DR7 にエイリアスしていた。デバッグ拡張がイネーブルになっていない（制御レジスタ CR4 の DE フラグがクリアされている）ときは、P6 ファミリ・プロセッサとインテル®

Pentium® プロセッサはエイリアスされたこれらの参照を許可にすると、既存のソフトウェアとの互換性を維持している。デバッグ拡張がイネーブルになっている（DE フラグがセットされている）ときは、レジスタ DR4 または DR5 に参照しようとすると、無効オペコード例外（#UD）が発生する。

18.20.4. ブレークポイントの認識

インテル® Pentium® プロセッサでは、デバッグ・プログラムは、デバッグ操作の対象となるプログラムに戻る前に LGDT 命令を実行して確実にブレークポイントを検出するように推奨する。この操作は、P6 ファミリ・プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサでは実行する必要はない。

18.21. テストレジスタ

Intel486™ プロセッサでは、テストレジスタを使用してキャッシュと TLB のテストが行われるが、P6 ファミリ・プロセッサとインテル® Pentium® プロセッサでは、MSR を使用するように設計変更されている。（この機能に使用されている MSR は、P6 ファミリ・プロセッサとインテル Pentium プロセッサでは異なっているので注意しなければらない。）P6 ファミリ・プロセッサでは、テストレジスタとのデータ転送用 MOV 命令を使用すると、無効オペコード例外（#UD）が発生する。

18.22. 例外と例外条件

本節では、32 ビット IA-32 プロセッサに新たに追加された例外と例外条件、既存の例外処理における相違点について説明する。IA-32 の例外の詳細については、第 5 章「割り込みと例外の処理」を参照。

インテル® Pentium® III プロセッサでは、XMM レジスタと共に新しいステートが導入された。これらのレジスタ内のデータを使用する計算は、例外を発生させることがある。新しい MXCSR 制御 / ステータス・レジスタを使用して、どの例外が発生したかを確認できる。XMM レジスタに関連する例外が発生すると、割り込みが生成される。

- SIMD 浮動小数点例外（#XF、割り込み 19）－ SIMD 浮動小数点レジスタおよびそれらを使用する計算に関連する、新しい例外。

インテル® Pentium® Pro プロセッサとインテル® Pentium® II プロセッサには新しい例外は追加されていない。つまり、提供される一連の例外は、インテル® Pentium® プロセッサの場合と同じである。ただし、インテル Pentium Pro プロセッサで使用している IA-32 には、次の例外条件が追加されている。

- マシンチェック例外（#MC、割り込み 18）ー 新しい例外条件。マシンチェック例外には多数の例外条件が追加されており、ハードウェア・エラーの処理とレポート用として新たにアーキテクチャが追加された。新しい条件の詳細については、第 14 章「マシン・チェック・アーキテクチャ」を参照。

インテル Pentium プロセッサで使用している IA-32 には、次に示す例外や例外条件が追加されている。

- マシンチェック例外（#MC、割り込み 18）ー 新しい例外。この例外は、パリティエラーやその他のハードウェア・エラーをレポートする。これはモデル固有の例外であり、将来のプロセッサではサポートされないかも知れないし、異なる形でサポートされる可能性もある。制御レジスタ CR4 の MCE フラグによって、マシンチェック例外がイネーブルになる。このビットがクリアされているとき（リセット時の状態）は、プロセッサはマシンチェック例外の発生を禁止する。
- 一般保護例外（#GP、割り込み 13）ー 新たに追加された例外条件。特定のレジスタの予約ビット位置に 1 を書き込もうとすると、一般保護例外が発生する。
- ページフォルト例外（#GP、割り込み 14）ー 新たに追加された例外条件。アドレス変換実行時に、ページ・テーブル・エントリ、ページ・ディレクトリ・エントリ、またはページ・ディレクトリ・ポインタの予約ビット位置のいずれかで 1 が検出されると、ページフォルト例外が発生する。

Intel486™ プロセッサには、次に示す例外が追加されている。

- アライメント・チェック例外（#AC、割り込み 17）ー 新しい例外。アライメント・チェックの実行時に、アライメントが合っていないメモリ参照をレポートする。

Intel386™ プロセッサには、次に示す例外や例外条件が追加されている。

- 除算エラー例外（#DE、割り込み 0）
  - ー 例外処理の変更。Intel386 プロセッサでは、除算エラー例外が発生すると、セーブされた CS:IP 値は常にエラーを発生した命令を指したままになる。8086 プロセッサでは、CS:IP 値は次の命令を指す。
  - ー 例外処理の変更。Intel386 プロセッサは、IDIV 命令の商として負の最大数（80H と 8000H）を生成できる。8086 プロセッサは、代わりに除算エラー例外を生成する。
- 無効オペコード例外（#UD、割り込み 6）ー 新たに追加された例外条件。LOCK 命令プリフィックスの使用法に問題があると、無効オペコード例外が発生することがある。
- ページフォルト例外（#PF、割り込み 14）ー 新たに追加された例外条件。16 ビット・プログラムでページングがイネーブルにされている場合は、以下のようにページフォルト例外を発生させることができる。すべてのタスクが同じページ・ディレクトリを使用する場合は、16 ビット・タスクを持つシステムにページングを使

用できる。16 ビットの TSS には PDBR レジスタをストアする場所がないので、16 ビット・タスクに切り替えても、PDBR レジスタの値は変わらない。インテル® 286 プロセッサから移植されたタスクでは、ページングを完全に利用できるようにするため 32 ビットの TSS にしなければならない。

- 一般保護例外（#GP、割り込み 13）ー 新たに追加された例外条件。Intel386 プロセッサでは、命令の長さが 15 バイトに制限されている。この制限を回避できる唯一の方法は、命令の前に余分なプリフィックスを付けることである。命令長の制限に違反すると、一般保護例外が発生する。8086 プロセッサには、命令長の制限はない。

18.22.1. マシン・チェック・アーキテクチャ

インテル® Pentium® Pro プロセッサでは、マシンチェック例外を処理しレポートするための新しいアーキテクチャを IA-32 に導入している。このマシン・チェック・アーキテクチャ（第 14 章「マシン・チェック・アーキテクチャ」に詳述してある）によって、プロセッサが内部ハードウェア・エラーをレポートするための機能が大幅に拡張されている。

18.22.2. 例外の優先順位

例外の優先順位は、大きく分けて次のカテゴリに分類できる。

1. 前の命令でのトラップ
2. 外部割り込み
3. 次の命令をフェッチする際のフォルト
4. 次の命令をデコードする際のフォルト
5. 命令を実行する際のフォルト

これらの主要なカテゴリの優先順位は、プロセッサによって変化しない。ただし、これらのカテゴリ内の例外はプロセッサに依存し、プロセッサごとに異なる可能性がある。

18.23. 割り込み

以降に、各種の IA-32 プロセッサ間に見受けられる割り込み処理上の相違点を示す。

18.23.1. 割り込みの伝搬遅延

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサはスーパースカラ設計を採用しているため、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサでは、それぞれに、外部ハードウェア割り込みが認識される命令境界が異なることがある。したがって、割り込みを処理するときにスタックにプッシュされる EIP は、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサ、Intel386 プロセッサに対して異なる可能性がある。

18.23.2. NMI 割り込み

P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサ、インテル® 286 プロセッサでは、8086 プロセッサの場合と異なり、NMI 割り込みが認識された後、最初の IRET 命令が実行されるまで NMI 割り込みがマスクされる。

18.23.3. IDT の制限

LIDT 命令を使用して、IDT のサイズに制限を設定できる。割り込みまたは例外がこの制限を超えてベクタを読み取ろうとした場合、ダブルフォルト例外（#DF）が発生する。この後、ダブル・フォルト・ハンドラ・ベクタがこの制限を超えた場合は、32 ビット IA-32 プロセッサではシャットダウンが発生する。(8086 プロセッサにはシャットダウン・モードも制限もない。)

18.24. アドバンスト・プログラマブル割り込みコントローラ（APIC）

アドバンスト・プログラマブル割り込みコントローラ（APIC）（本書では**ローカル APIC** と呼ばれる）は、インテル® Pentium® プロセッサで（735/90 および 815/100 モデルから）IA-32 プロセッサに導入され、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサに搭載されている。ローカル APIC の機能は、Intel486™ プロセッサおよび初期のインテル Pentium プロセッサで使用されていた、インテル® 82489DX 外部 APIC を受け継いでいる。インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、ローカル APIC アーキテクチャの新しい機能が追加されている。

18.24.1. ローカル APIC と 82489DX のソフトウェア的に認識可能な相違点

ローカル APIC の機能と 82489DX 外部 APIC の機能は、以下の点で異なっている。

- スプリアス割り込みベクタ MSR 内の APIC ソフトウェア有効 / 無効フラグをクリアしてローカル APIC を無効にした場合、LVT 内のマスクビットがすべてセットされてプロセッサに対するローカル割り込みがブロックされることを除いて、ローカル APIC の内部レジスタの状態は影響を受けない。この場合、ローカル APIC は、INIT、SMI、NMI、およびスタートアップ IPI 以外の IPI の受け入れを停止する。82489DX では、ローカルユニットが無効にされると、IRR、ISR、TMR を含むすべての内部レジスタがクリアされ、LVT 内のマスクビットがセットされる。この状態では、82489DX ローカルユニットは、リセット・デアサート・メッセージだけを受け入れる。
- ローカル APIC では、NMI と INIT（INIT デアサートを除く）は、トリガ設定に関係なく、常にエッジトリガ割り込みとして扱われる。82489DX 内では、これらの割り込みは、常にレベルトリガ割り込みになる。
- ローカル APIC では、ICR を使用して生成された IPI は、常にエッジトリガ割り込みとして扱われる（INIT デアサートを除く）。82489DX では、ICR を使用して、エッジトリガ IPI とレベルトリガ IPI を生成できる。
- ローカル APIC では、論理デスティネーション・レジスタは 8 ビットに対応する。82489DX では、論理デスティネーション・レジスタは 32 ビットに対応する。
- ローカル APIC では、APIC ID レジスタの幅は 4 ビットである。82489DX では、APIC ID レジスタの幅は 8 ビットである。
- 82489DX とインテル® Pentium® プロセッサ用ローカル APIC でサポートしているリモート読み出し伝達モードは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサのローカル APIC ではサポートされない。
- 82489DX の場合、最低優先度伝達モードでは、デスティネーション・フィールドで指定されたすべての送信先ローカル APIC が、最低優先度アービトレーションに参加する。ローカル APIC の場合、空いている割り込みスロットを持つローカル APIC だけが、最低優先度アービトレーションに参加する。

18.24.2. P6 ファミリ・プロセッサおよびインテル® Pentium® プロセッサのローカル APIC に追加された新機能

インテル® Pentium® プロセッサおよび P6 ファミリ・プロセッサのローカル APIC は、82489DX 外部 APIC が持っていない、次のような新機能を備えている。

- 論理デスティネーション・モードでクラスタアドレス指定をサポートする。
- フォーカス・プロセッサのチェックを有効 / 無効にできる。

- LINT0 ピンと LINT1 ピンの割り込み入力信号の極性をプログラムできる。
- ICR と I/O リダイレクション・テーブルによって、SMI IPI をサポートする。
- APIC エラーの記録および報告用のエラー・ステータス・レジスタが、LVT に組み込まれている。

P6 ファミリ・プロセッサでは、ローカル APIC に、性能モニタリング・カウンタ割り込みを処理するための追加の LVT レジスタが組み込まれている。

18.24.3. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサのローカル APIC に追加された新機能

インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサのローカル APIC は、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、82489DX が持っていない、次のような新機能を備えている。

- ローカル APIC ID が 8 ビットに拡張された。
- 温度センサ割り込みを処理するための温度センサレジスタが、LVT に組み込まれている。
- フォーカス・プロセッサに最低優先度割り込みを伝達する機能は廃止された。
- フラットクラスタ論理デスティネーション・モードはサポートしていない。

18.25. タスク・スイッチングと TSS

本節では、プロセッサによるタスク・スイッチングの相違点、TSS への追加事項、TSS と TSS セグメント・セレクタの処理について説明する。

18.25.1. P6 ファミリ・プロセッサとインテル® Pentium® プロセッサの TSS

仮想モードの拡張機能が（制御レジスタ CR4 の VME フラグをセットして）イネーブルにされているときは、P6 ファミリ・プロセッサとインテル® Pentium® プロセッサの TSS には割り込みリダイレクション・ビット・マップがストアされている。このビットマップは、割り込みを 8086 プログラムにリダイレクトして戻す場合に仮想 8086 モードで使用される。

18.25.2. TSS セレクタの書き込み

タスク状態をセーブする際に、Intel486™ プロセッサは 32 ビットの TSS に、上位 16 ビットを未定義のままにして 2 バイトのセグメント・セレクタを書き込む。性能上の理由から、P6 ファミリ・プロセッサとインテル® Pentium® プロセッサは、上位 2 バイトを

0にして4バイトのセグメント・セレクタをTSSに書き込む。互換性を維持するため、コードはTSS内のセレクタの上位16ビットの値に依存しないようにする。

18.25.3. TSS の読み取り / 書き込みの順序

TSSの読み取りと書き込みの順序はプロセッサによって異なる。TSSがページ境界にまたがっている（この状況は推奨できない）場合は、P6ファミリ・プロセッサとインテル® Pentium® プロセッサは、同じTSS領域内にある制御レジスタCR2に、Intel486™プロセッサやIntel386™プロセッサとは異なるページ・フォルト・アドレスを生成するときがある。

18.25.4. 32 ビット構造での 16 ビット TSS の使用

16ビットのTSSを使用するタスクスイッチは、16ビットだけのコードに対してのみ使用しなければならない。32ビット構造（オペランド、アドレス指定、またはEFLAGSレジスタの上位ワード）を使用して新たに記述されるコードではすべて、32ビットのTSSだけを使用するようにしなければならない。これは、32ビット・プロセッサは16ビットのTSSにはEFLAGSの上位16ビットをセーブしないためである。タスクスイッチで仮想モードで実行されていた16ビット・タスクに戻っても、このフラグはTSSのEFLAGS値の上位16ビットにはセーブされていなかったので、仮想モードが再びイネーブルになることは決してない。したがって、マルチタスキング環境での正しい動作を保証するために、32ビット構造を使用しているコードは必ず32ビットのTSSを使用するよう強く推奨する。

18.25.5. I/O マップ・ベース・アドレスの相違

Intel486™プロセッサは、TSSセグメントを16ビット・セグメントと見なし、64K境界でラップアラウンドする。どのI/Oアクセスでも、I/Oベースアドレスに I/Oオフセットを加えた位置で、このI/Oアドレスにアクセスするための許可をチェックする。I/Oマップ・ベース・アドレスが指定されたリミット0DFFFHを超えた場合は、I/Oアクセスはラップアラウンドし、TSS内の誤った位置のI/Oアドレスに対する許可が与えられる。Intel486プロセッサでは、この状況でもTSSリミット違反は発生しない。しかし、P6ファミリ・プロセッサとインテル® Pentium® プロセッサはTSSを32ビット・セグメントと見なすため、I/Oベースアドレスに I/Oオフセットを加えた値がTSSリミットを超えるとリミット違反が発生する。推奨される仕様にしたがってI/Oベースアドレスを0DFFFHより小さくすれば、Intel486プロセッサはラップアラウンドせず、TSS内の誤った位置へのアクセスによるI/Oポート違反は生じない。また、P6ファミリ・プロセッサとインテルPentiumプロセッサで一般保護例外（#GP）が発生することもない。図 18-1. に、Intel486プロセッサとP6ファミリ・プロセッサおよびインテルPentiumプロセッサとのアクセス領域の相違を示す。

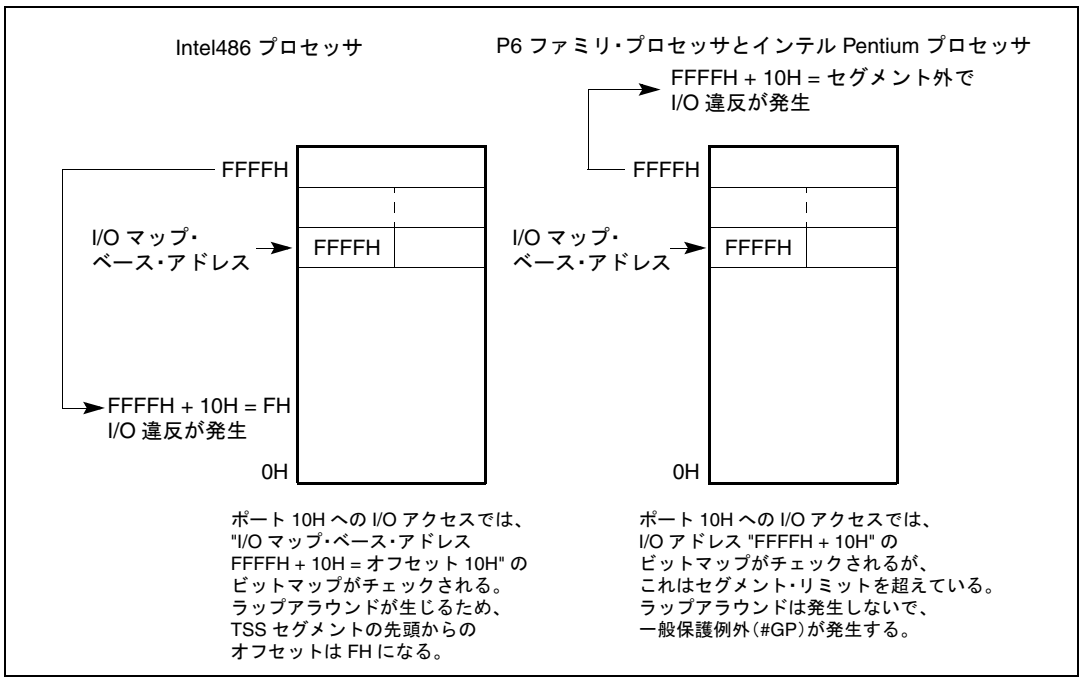


図 18-1. I/O マップ・ベース・アドレスの相違

18.26.キャッシュ管理

P6 ファミリ・プロセッサには、2 つのレベルの内部キャッシュ、L1（レベル 1）と L2（レベル 2）が内蔵されている。L1 キャッシュは命令キャッシュとデータ・キャッシュに分かれている。L2 キャッシュは汎用キャッシュである。これらのキャッシュの詳細については、10.1. 節「内部キャッシュ、TLB、バッファ」を参照。（インテル® Pentium® II プロセッサの L2 キャッシュは物理的にはカセット内の別のチップにあるが、内部キャッシュと見なされていることに注意しなければならない。）

インテル® Pentium® プロセッサには、独立したレベル 1 の命令キャッシュとデータ・キャッシュが内蔵されている。データ・キャッシュは、メモリ更新用としてライトバック方式（またはそれに代えてライン単位によるライトスルー方式）をサポートしている。インテル Pentium プロセッサのキャッシュの編成と動作の詳細については、『Pentium Processor Data Book』を参照。

Intel486™ プロセッサには、命令とデータの両方で使用される単一のレベル 1 キャッシュが内蔵されている。

P6 ファミリ・プロセッサとインテル Pentium プロセッサに対しては、制御レジスタ CR0 の CD フラグと NW フラグの意味が再定義されている。これらのプロセッサについては、両フラグの設定値が 00B（推奨値）の場合、インテル Pentium プロセッサのデー

タ・キャッシュおよびP6ファミリ・プロセッサのL1データ・キャッシュとL2キャッシュに対してライトバックがイネーブルになる。それに対してIntel486プロセッサについては、これらのフラグを00Bに設定すると、キャッシュに対してライトスルーがイネーブルになる。

必要な場合は、インテルPentiumプロセッサに対して、外部システム・ハードウェアからキャッシングをディスエーブルにさせたり、またはライトスルー方式を使用できる。インテルPentiumプロセッサのキャッシュのハードウェア制御に関する詳細については、『Pentium Processor Data Book』を参照。P6ファミリ・プロセッサでは、MTRRを使用してCDフラグとNWフラグをオーバーライドできる（表10-6.を参照）。

P6ファミリ・プロセッサとインテルPentiumプロセッサは、制御レジスタCR3のPCDフラグとPWTフラグ、ページ・ディレクトリ・エントリ、ページ・テーブル・エントリを使用すれば、Intel486プロセッサの場合と同様にページレベルのキャッシュ管理をサポートする。それに対し、Intel486プロセッサの内部キャッシュはライトスルー型キャッシュなので、Intel486プロセッサはPWTフラグの状態には左右されない。

18.26.1. キャッシュ・イネーブル時の自己修正コード

Intel486™ プロセッサでは、キャッシュ内の命令への書き込みが行われると、キャッシュとメモリの両方で命令が修正される。ところが、命令は書き込みより前にプリフェッチされているので、旧バージョンの命令が実行されたことになる。この問題を防止するには、命令を修正するようなすべての書き込みの直後にジャンプ命令をコーディングすることによって、Intel486プロセッサの命令プリフェッチ・ユニットをフラッシュする必要がある。それに対して、P6ファミリ・プロセッサとインテル® Pentium® プロセッサでは、実行に備えてプリフェッチされている命令が書き込みによって修正される可能性があるかどうかをチェックする。このチェックは、命令のリニアアドレスに基づいて行われる。命令のリニアアドレスがプリフェッチ・キューに存在することがわかった場合は、P6ファミリ・プロセッサとインテルPentiumプロセッサはプリフェッチ・キューをフラッシュする。したがって、命令を修正するような書き込みの後のジャンプ命令のコーディングは不要になる。

書き込みのリニアアドレスは、プリフェッチされた命令のリニアアドレスに対してチェックされる。このため、命令と書き込まれるデータの物理アドレスは同じであるが、それらのリニアアドレスが異なるときには、自己修正コードを正しく動作させるためには格別の注意が必要である。そのような場合は、書き込みの後、修正された命令を実行する前に、シリアル化操作を実行してプリフェッチ・キューをフラッシュする必要がある。シリアル化命令の詳細については、7.4.節「シリアル化命令」を参照。

注記

上記のリニアアドレスのチェックは、実際には互換性にかかわる問題ではない。自己修正コードを含むアプリケーションは、命令の修正とフェッチを行う際に同じリニアアドレスを使用する。命令のフェッチに使用されたものとは異なるリニアアドレスを使用して命令を修正する可能性がある、デバッガなどのシステム・ソフトウェアは、修正された命令が実行される前に、IRET などのシリアル化操作を実行しなければならない。

18.26.2. L3 キャッシュの無効化

ユニファイド第 3 レベル（L3）キャッシュは、第 3 レベル・キャッシュ無効フラグ（IA32_MISC_ENABLE MSR のビット 6）と合わせて、インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサで導入された（10.1. 節「内部キャッシュ、TLB、バッファ」を参照）。第 3 レベル・キャッシュ無効フラグによって、L1 および L2 キャッシュとは無関係に、L3 キャッシュを無効または有効にできる（10.5.4. 項「L3 キャッシュの無効化と有効化」を参照）。

18.27. ページング

本節では、ページング・メカニズムに追加された拡充点と各種 IA-32 プロセッサでのページング・メカニズムの相違点について説明する。

18.27.1. 拡張サイズページ

インテル® Pentium® プロセッサでは、大きな（4M バイト）ページサイズを取り扱えるように、IA-32 のメモリ管理 / ページング機能を拡張した（3.6.1. 項「ページングのオプション」）。初期の P6 ファミリ・プロセッサ（インテル® Pentium® Pro プロセッサ）では、IA-32 に物理アドレス拡張（PAE）機能と合わせて 2M バイトのページサイズを追加している（3.8. 節「PAE ページング・メカニズムを使用した 36 ビット 物理アドレス指定」を参照）。

任意の IA-32 プロセッサで拡張サイズページが使用可能かどうかを知るには、引き数 1 を使用して CPUID 命令を実行した後に、レジスタ EDX のフィーチャ・ビット 3（PSE）を調べるようにする。CPUID 命令をサポートしないインテル® プロセッサでは、ページサイズの拡張もサポートしない。（CPUID 命令の詳細については、『IA-32 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、中巻 A』の第 3 章「命令セット・リファレンス A-M」の第 3 章「命令セット・リファレンス A-M」の「CPUID—CPU Identification」、および『AP-485、インテル® プロセッサの識別と CPUID 命令』を参照。

18.27.2. PCD フラグと PWT フラグ

PCD フラグと PWT フラグは、ページ単位のキャッシング制御用として Intel486™ プロセッサで IA-32 に導入された。

- PCD（ページ・レベル・キャッシュ・ディスエーブル）フラグ ― キャッシングをページ単位で制御する。
- PWT（ページ・レベル・ライトスルー）フラグ ― ライトスルー / ライトバック・キャッシング方式をページ単位で制御する。Intel486 プロセッサの内部キャッシュはライトスルー・キャッシュなので、PWT フラグの状態によって左右されない。

18.27.3. ページングのイネーブルとディスエーブル

ページングは、PG フラグを修正する制御レジスタ CR0 に値をロードすることによってイネーブルまたはディスエーブルにする。IA-32 プロセッサとの下位および上位互換性を保証するため、インテルでは、ページングをイネーブルまたはディスエーブルにする際は次に示す操作を実行するように推奨する。

1. MOV CR0, REG 命令を実行して、PG フラグをセットする（ページングをイネーブルにする）か、クリアする（ページングをディスエーブルにする）。
2. near JMP 命令を実行する。

MOV 命令と JMP 命令の間に挟まれたシーケンスは、アイデンティティ・マッピングする必要がある（つまり、それらの命令は、そのリニアアドレスと物理アドレスが同じであるページ上に存在しなければならない）。

P6 ファミリ・プロセッサでは、MOV CR0, REG 命令でシリアル化が行われ、ジャンプ操作は不要になる。ただし、下位互換性のために、JMP 命令は挿入する必要がある。

18.28. スタック操作

本節では、各種 IA-32 プロセッサのスタック・メカニズムの相違点について説明する。

18.28.1. セレクタのプッシュとポップ

スタックにセグメント・セレクタをプッシュするときは、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、Intel486™ プロセッサでは、ESP レジスタをオペランド・サイズだけデクリメントしてから、2 バイトを書き込む。オペランド・サイズが 32 ビットの場合、書き込み時の上位 2 バイトは変更されない。インテル® Pentium® プロセッサでは、ESP レジスタをオペランド・サイズ

だけデクリメントし、オペランド・サイズによって書き込みのサイズを決定する。オペランド・サイズが 32 ビットの場合、上位 2 バイトには 0 が書き込まれる。

スタックからセグメント・セレクタをポップするとき、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、Intel486 プロセッサは 2 バイトを読み取り、命令のオペランド・サイズによって ESP レジスタを増分する。インテル Pentium プロセッサでは、オペランド・サイズから読み取りのサイズを決定し、ESP レジスタをオペランド・サイズだけインクリメントする。

32 ビット・セレクタによるプッシュまたはポップのアライメントを合わせることが可能であり、この場合、インテル Pentium プロセッサ上ではこの操作により例外が生成され、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、Intel486 プロセッサでは例外は生成されない。これが生じるのは、この操作の 3 番目または 4 番目のバイトをセグメント・リミットを超えて配置する場合か、あるいは不在ページまたはアクセス不能なページ上に配置する場合である。

メモリへの POP 命令が以下の条件を満たす場合 :

- スタック・セグメントのサイズが 16 ビットである。
- ベースレジスタとして ESP を指定する SIB バイトを含む 32 ビット・アドレス指定形式を使用する。
- 初期スタックポインタは FFFCh（32 ビット・オペランド）または FFFEh（16 ビット・オペランド）であり、POP 操作の結果、0H にラップアラウンドされる。

メモリ書き込みの結果は、実装によって異なる。例えば、P6 ファミリ・プロセッサでは、メモリ書き込みの結果は、SS:0H + スケーリングされたインデックスとディスプレースメントになる。インテル Pentium プロセッサでは、メモリ書き込みの結果は、スタックフォルトになるか（リアルモードまたは保護モードで、スタック・セグメントのサイズが 64K バイトの場合）、または SS:10000H + スケーリングされたインデックスとディスプレースメントへの書き込みになる（保護モードで、スタック・セグメントのサイズが 64K バイトを超える場合）。

18.28.2. エラーコードのプッシュ

Intel486™ プロセッサは、16 ビット値のエラーコードをスタックにプッシュする。32 ビット・スタックにプッシュするときは、Intel486 プロセッサは 2 バイトだけをプッシュし、ESP を 4 だけ更新する。P6 ファミリ・プロセッサとインテル® Pentium® プロセッサのエラーコードは完全な 32 ビットであり、それらのうち上位 16 ビットはゼロに設定される。したがって、P6 ファミリ・プロセッサとインテル Pentium プロセッサは 4 バイトをプッシュし、ESP を 4 だけ更新する。上位 16 ビットの状態に依存するコードでは常に、一定した結果が生成されない可能性がある。

18.28.3. フォルト処理のスタックへの影響

CALL や PUSHA などの特定の命令の処理中は、各種のプロセッサのさまざまなシーケンスでフォルトが発生する可能性がある。例えば far コールでは、Intel486™ プロセッサは、発生した分岐フォルトが解決される前に、以前の CS と EIP をプッシュする。分岐フォルトとは、分岐命令が原因で発生するフォルトで、セグメント・リミットまたはアクセス権の違反によって発生する。分岐フォルトが発生した場合は、Intel486 プロセッサや P6 ファミリ・プロセッサでは、スタックポインタより下のメモリの内容が破壊されるが、ESP レジスタはバックアップされ、命令は再スタート可能になる。P6 ファミリ・プロセッサとインテル® Pentium® プロセッサは、プッシュの前に分岐を発行する。したがって、分岐フォルトが発生しても、これらのプロセッサではスタックポインタより下のメモリの内容は破壊されない。ただし、スタックポインタ以上の値しか有効と見なされないので、プロセッサの相違によって互換性上の問題が生じることはない。

18.28.4. 16 ビット割り込みゲートまたはコールゲートからのレベル間 RET/IRET

32 ビット・スタック環境から、16 ビット・ゲートを通じて呼び出しまたは割り込みが行われる場合は、スタックにプッシュできるのは以前の ESP の 16 ビットだけである。その後の RET/IRET でこの 16 ビットの ESP がポップされるが、制御が 32 ビットのスタック環境で再開されるので、32 ビット・フル・サイズの ESP が更新される。Intel486™ プロセッサは、SS セレクタを ESP の上位 16 ビットに書き込む。P6 ファミリ・プロセッサとインテル® Pentium® プロセッサは、上位 16 ビットにゼロを書き込む。

18.29.16 ビット・セグメントと 32 ビット・セグメントの混在

16 ビットインテル® 286 プロセッサの機能は、32 ビット IA-32 プロセッサの対応する機能のオブジェクト・コード・レベルで互換可能なサブセットである。セグメント・ディスクリプタ内の D（デフォルト動作サイズ）フラグは、プロセッサがコード・セグメントまたはデータ・セグメントを 16 ビットと 32 ビットのどちらのセグメントとして取り扱うかを示し、セグメント・ディスクリプタ内の B（デフォルト・スタック・サイズ）フラグは、プロセッサがスタック・セグメントを 16 ビットと 32 ビットのどちらのセグメントとして取り扱うかを示す。

286 プロセッサが使用するセグメント・ディスクリプタは、ディスクリプタのインテル予約ワード（最上位ワード）がクリアされている場合は、32 ビット IA-32 プロセッサによってサポートされる。32 ビット IA-32 プロセッサでは、このワードにベースアドレスの上位ビットとセグメント・リミットがストアされる。

データ・セグメント、コード・セグメント、ローカル・ディスクリプタ・テーブル、タスクゲートのセグメント・ディスクリプタ（グローバル・ディスクリプタ・テーブルのディスクリプタはない）は、16 ビットと 32 ビットのプロセッサに対して同じである。その他の 16 ビット・ディスクリプタ（TSS セグメント、コールゲート、割り込みゲート、トラップゲートのディスクリプタ）は 32 ビットのプロセッサによってサポートされる。

32 ビットのプロセッサには、32 ビット・アーキテクチャをサポートする TSS セグメント、コールゲート、割り込みゲート、トラップゲートのディスクリプタもある。両種類のディスクリプタを同一システム内で使用できる。

16 ビットと 32 ビットの両プロセッサに共通のセグメント・ディスクリプタについては、予約ワードのビットがクリアされている場合は、32 ビット・プロセッサはそれらのディスクリプタを 286 プロセッサの場合と全く同様に解釈する。つまり、次のとおりである。

- ベースアドレス － 32 ビット・ベース・アドレスの上位 8 ビットがクリアされている場合は、ベースアドレスが 24 ビットに制限される。
- リミット －リミット・フィールドの上位 4 ビットがクリアされている場合は、リミット・フィールドの値が 64K バイトに制限される。
- グラニュラリティ・ビット － G（グラニュラリティ）フラグがクリアされている場合は、16 ビット・リミットの値が 1 バイト単位で解釈されることを示す。
- ビッグビット －データ・セグメント・ディスクリプタでは、32 ビット・プロセッサが使用するセグメント・ディスクリプタの B フラグがクリアされている場合は、セグメントが 64K バイトより大きくないことを示す。
- デフォルト・ビット －コード・セグメント・ディスクリプタでは、D フラグがクリアされている場合は、16 ビットのアドレス指定とオペランドがデフォルトであることを示す。スタック・セグメント・ディスクリプタでは、D フラグがクリアされている場合は、（ESP レジスタの代わりに）SP レジスタと 64K バイトの最大セグメント・リミットが使用されることを示す。

アプリケーション内での 16 ビット・コードと 32 ビット・コードの混在に関する詳細については、第 17 章「16 ビット・コードと 32 ビット・コードの混在」を参照。

18.30.セグメントとアドレスのラップアラウンド

本節では、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサ、インテル® 286 プロセッサ、8086 プロセッサにおけるセグメントとアドレスのラップアラウンドの相違点について説明する。

18.30.1. セグメント・ラップアラウンド

8086 プロセッサでは、オフセット 65,535（0FFFFH）かオフセット 0 をまたぐようなメモリ・オペランドにアクセスしようとする（例えば、ワードをオフセット 65,535 に移動するか、またはスタックポインタが 1 に設定されているときにワードをプッシュする）と、オフセットはモジュロ 65,536（010000H）でラップアラウンドする。インテル® 286 プロセッサでは、16M バイトを超えてアドレス指定するベースとオフセットの組み合わせでは常に 1M バイトのアドレス空間にラップアラウンドする。実アドレスモードにある P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサでは、これらの場合に例外が発生する。

- セグメントがデータ・セグメントの場合（つまり、CS、DS、ES、FS、または GS レジスタがセグメントのアドレス指定に使用される場合）は、一般保護例外（#GP）が発生する。
- セグメントがスタック・セグメントの場合（つまり、SS レジスタが使用される場合）は、スタックフォルト例外（#SS）が発生する。

この動作の例外は、スタックアクセスがデータのアライメントに合っていて、かつスタックポインタが、サイズの上限であるスタックの一番上のアライメントが合った最後の部分を指しているとき（ESP が FFFFFFFCH のとき）に生じる。このデータがポップされても、セグメント・リミット違反は発生せず、スタックポインタは 0 にラップアラウンドする。

P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサのアドレス空間は、実アドレスモードでは 1M バイトにラップアラウンドする可能性がある。外部ピン A20M# がイネーブルになっている場合は、強制的にラップアラウンドさせられる。インテル 8086 プロセッサでは、1M バイトを超えるアドレスを指定できる。例えば、セレクタ値を FFFFH、オフセットを FFFFH とすると、実効アドレスは 10FFEFH（1M バイト + 65519 バイト）になる。8086 プロセッサは、形成できるアドレスの最長ビット数が 20 までなので、最上位ビットを切り捨てる。したがって、このアドレスは FFFFH に「ラップ」されることになる。それに対して、A20M# がイネーブルになっていない場合は、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサはこのビットを切り捨てない。

スタック操作によってアドレスリミットでラップアラウンドが生じた場合は、プロセッサはシャットダウンする。（8086 プロセッサには、シャットダウン・モードもリミットもない。）

4GB セレクタのリミット（リミット =0xFFFFFFFF）の近くで実行する場合の動作は、インテル® Pentium® Pro プロセッサとインテル® Pentium® 4 プロセッサ・ファミリの間で異なる。インテル Pentium Pro プロセッサでは、リミットを超える命令（例えば、ちょうどリミットの位置から始まる、0xFF 0xC0 としてエンコーディングされる INC EAX などの 2 バイト命令）は、セグメント違反のためにフォルトになる（0xFFFFFFFF の位置の 1 バイト命令では、例外は発生しない）。インテル Pentium 4 マイクロプロセッサ・ファミリでは、上記のいずれの状況でもフォルトは発生しない。

18.31. ストアバッファとメモリのオーダリング

インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサには、メモリへの書き込み（ストア）を一時的にストアするためのストアバッファが用意されている（10.10. 節「ストアバッファ」を参照）。このストアバッファにストアされた書き込みは、「高速ストリング」ストア操作（7.2.3. 項「インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサでのストリングからの アウト・オブ・オーダー・ストア操作」を参照）の場合を除いて、常にプログラム順にメモリにストアされる。

インテル® Pentium® プロセッサでは 2 つのストアバッファが用意されており、それぞれ各パイプラインに対応している。これらのバッファにストアされた書き込みは必ず、それらがプロセッサ・コアから生成された順序でメモリに書き込まれる。

バッファの対象となるのはメモリへの書き込みだけであり、I/O 書き込みはバッファの対象にならないことに注意しなければならない。インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサ、インテル Pentium プロセッサ、Intel486 プロセッサは、バス上でのメモリ書き込みの終了と書き込み後の命令の実行とを同期させない。I/O、ロックされた命令、またはシリアル化命令を実行して、書き込みを次の命令と同期させる必要がある（7.4. 節「シリアル化命令」を参照）。

インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、P6 ファミリ・プロセッサでは、プロセッサによるオーダリングを使用して、プログラム上のデータの読み取り（ロード）および書き込み（ストア）の順序と、プロセッサが実際にそれらの読み取りおよび書き込みを実行する順序とのコンシステンシを維持する。このタイプのオーダリングでは、読み取りが推論的に実行でき、バッファされている書き込みを任意の順序で受け渡せる。また、メモリへの書き込みは常にプログラム順に実行できる。（プロセッサによるオーダリングの詳細については、7.2. 節「メモリ・オーダリング」を参照。）インテル® Pentium® III プロセッサでは、書き込みをシリアル化してグローバルにアクセス可能にするための新しい命令が追加された。データを生産するルーチンと

消費するルーチンの間には、メモリ・オーダリングの問題が発生する可能性がある。SFENCE 命令は、順序設定の緩い結果を生産するルーチンとそのデータを消費するルーチンの間のオーダリングを保証するための効率的な方法である。

7.2.1. 項「インテル ® Pentium® プロセッサおよび Intel486™ プロセッサでの メモリ・オーダリング」と以下の Intel486™ プロセッサに関する説明に記されている条件下を除いて、インテル Pentium プロセッサでは読み取りのリオーダリングは行われない。

特に、書き込みバッファは IN 命令が実行される前にフラッシュされる。キャッシュ・ミスの結果発生するいずれの読み取りも、あらかじめ生成されて書き込みバッファにストアされている書き込みの前後にリオーダリングはされない。それは、その後のバスサイクルが外部バスで実行される前に、ストアバッファがフラッシュされる、つまり空にされることを意味している。

Intel486 プロセッサとインテル Pentium プロセッサでは、特定の条件の下では、プログラム実行のより早い時点で書き込みが発生していても、バッファ内でペンディングのメモリ書き込みより前に、メモリ読み取りが外部バスに送出される。メモリ読み取りは、バッファ内でペンディングのすべての書き込みがキャッシュ・ヒットであり、その読み取りがキャッシュ・ミスである場合に限り、バッファ内でペンディングのすべての書き込みの前にリオーダリングされる。このような条件の下では、Intel486 プロセッサとインテル Pentium プロセッサは、ペンディングのいずれかの書き込みによって更新される必要がある外部メモリ位置からは読み取らない。

ロックされたバスサイクル中は、Intel486 プロセッサは常に外部メモリにアクセスし、決してオンチップ・キャッシュ内の位置を探すことはない。Intel486 プロセッサでは、ストアバッファ内のすべての書き込みペンディング・データがメモリに書き込まれてから初めて、ロックされたサイクルの外部バスへの送出が許される。したがって、Intel486 プロセッサでは、ロックされたバスサイクルを使用して、読み取りサイクルのリオーダリングが生じる可能性を排除できる。インテル Pentium プロセッサは、読み取り - 修正 - 書き込みアクセスでそのキャッシュをチェックし、キャッシュ・ラインが修正されていた場合は、その内容をメモリにライトバックしてからバスをロックする。P6 ファミリ・プロセッサは、（そのアクセスが 2 キャッシュ・ライン上にまたがらない場合は）読み取り - 修正 - 書き込み操作でキャッシュに書き込み、システムメモリにはライトバックしない。アクセスが 2 キャッシュ・ラインにまたがる場合は、P6 ファミリ・プロセッサはバスをロックし、システムメモリにアクセスする。

IA-32 プロセッサでは、I/O 読み取りは決してバッファされたメモリ書き込みの前にリオーダリングされることはない。それによって、I/O デバイスからステータスを読み取る前に、すべてのメモリ位置が確実に更新される。

18.32. バスのロック

インテル® 286プロセッサが行うバスロック操作は、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™プロセッサ、Intel386™プロセッサのそれとは異なる。286プロセッサに固有のメモリロック操作形式を使用するプログラムは、それより後のプロセッサで実行したとき正しく実行されないことがある。

ロックされた命令は、デスティネーション・オペランドによって定義されるメモリ領域だけをロックするように保証されているが、それより大きなメモリ領域をロックすることがある。例えば、代表的な8086プロセッサや286プロセッサの構成では、物理メモリ空間全体がロックされる。ただし、プログラムはそのことを当てにしてはならない。

286プロセッサでは、LOCKプリフィックスはIOPLに依存する。CPLがIOPLより大きい場合は、一般保護例外（#GP）が発生する。Intel386 DXプロセッサ、Intel486プロセッサ、インテルPentiumプロセッサ、P6ファミリ・プロセッサでは、IOPLに対するチェックは行われない。

インテルPentiumプロセッサは、外部割り込みを確認したときに自動的にLOCK#信号をアサートする。この信号で割り込み要求が通知されると、外部割り込みコントローラがデータバスを使用して割り込みベクタをプロセッサに送ることができる。割り込み要求信号を受け取ると、プロセッサはLOCK#信号をアサートして、その割り込みベクタが受け取られるまではデータバス上に他のデータが現れないようにする。このバスロックは、P6ファミリ・プロセッサでは行われない。

18.33. バスホールド

8086プロセッサやインテル® 286プロセッサとは異なるが、Intel386™プロセッサやIntel486™プロセッサと同様に、インテル® Pentium® プロセッサとP6ファミリ・プロセッサは、ダブルワードを形成する2ワードのような、アライメントが合っていないオペランドの各部分の転送と転送の間にも、DMAコントローラなど、使用される可能性のある他のバスマスタからのバス制御の要求に応答する。Intel386プロセッサとは異なり、P6ファミリ・プロセッサ、インテルPentiumプロセッサ、Intel486プロセッサは、リセットによる初期化実行時にバスホールドに応答する。

18.34.インテル® アーキテクチャのモデル固有の拡張

IA-32への一部の拡張は、IA-32プロセッサのプロセッサまたはファミリに固有であり、将来のプロセッサに同じ方法でサポートされるかどうかは不明である。以降の各項で、これらのモデル固有の拡張について説明する。CPUID 命令を使用すれば、一部のモデル固有機能の備えがあるかどうかを知ることができる。

18.34.1. モデル固有レジスタ

インテル® Pentium® プロセッサには、ハードウェア機能を制御する際や性能モニタリングを制御する際に使用する一連のモデル固有レジスタ（MSR）が導入されている。これらの MSR にアクセスするため、IA-32 アーキテクチャには新たに 2 つの命令、MSR 読み取り（RDMSR）と MSR 書き込み（WRMSR）が追加されている。インテル Pentium プロセッサの MSR が、次世代の IA-32 プロセッサにそのまま継承されるかどうかは保証の限りではない。

P6 ファミリ・プロセッサでは、ソフトウェアから使用できる MSR の数が大幅に増大されている。付録 B「モデル固有レジスタ（MSR）」に、使用可能な全 MSR の一覧が記載してある。新しいレジスタは、デバッグ拡張機能、性能カウンタ、マシンチェック例外機能、マシン・チェック・アーキテクチャ、および各 MTRR の制御用である。これらのレジスタには、RDMSR 命令と WRMSR 命令を使用してアクセスできる。以降の各項で、これらの新しい MSR のいくつかについて説明する。インテル Pentium プロセッサの MSR の場合と同様に、P6 ファミリ・プロセッサの MSR も、次世代の IA-32 プロセッサにそのまま継承されるかどうかは保証の限りではない。

18.34.2. RDMSR 命令と WRMSR 命令

RDMSR（モデル固有レジスタ読み取り）命令と WRMSR（モデル固有レジスタ書き込み）命令は、P6 ファミリ・プロセッサでは従来よりはるかに多数のモデル固有レジスタを認識する。（これらの命令の詳細については、『IA-32 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、中巻 B』の第 4 章の「RDMSR—Read from Model Specific Register」と「WRMSR—Write to Model Specific Register」を参照。）

18.34.3. メモリ・タイプ・レンジ・レジスタ

メモリ・タイプ・レンジ・レジスタ（MTRR）は、インテル® Pentium® Pro プロセッサで IA-32 に新たに導入された機能である。MTRR によって、プロセッサは、RAM、ROM、フレーム・バッファ・メモリ、メモリマップド I/O などのさまざまなタイプのメモリに対するメモリ操作を最適化できる。

MTRR は、物理アドレス範囲が各種のメモリにどのようにマップされるかを定義する内部マップストア用の MSR である。プロセッサはこの内部メモリマップを使用して、さまざまな物理メモリ位置がキャッシュ可能かどうか、およびメモリ位置へのアクセスに対する最適の方法を判定する。例えば、メモリ位置がライトスルー・メモリとして MTRR に指定されている場合は、プロセッサはこの位置へのアクセスを次のように処理する。プロセッサは、その位置からデータをライン単位で読み取り、読み取ったデータをキャッシュするか、またはその位置へのすべての書き込みをバスにマッピングし、キャッシュを更新してキャッシュのコヒーレンシを維持する。MTRR を使用して物理アドレス空間をマッピングする際は、プロセッサは 5 つのタイプのメモリ、つまりキャッシュ不可能（UC）、キャッシュ不可能 / 推論可能 / 書き込み組み合わせ（USWC）、ライトスルー（WT）、書き込み保護（WP）、ライトバック（WB）を認識する。

（Intel486™ プロセッサやインテル® Pentium® プロセッサなどの）前世代の IA-32 プロセッサでは、外部メモリマップの保守を行う際や、キャッシュ可能なアクセスをプロセッサに通知する際に、KEN#（キャッシュ・イネーブル）ピンと外部論理を使用していた。MTRR メカニズムによって、KEN# ピンとそのドライブに必要な外部論理の必要がなくなり、ハードウェア設計が単純化されている。

MTRR の詳細については、第 9 章「プロセッサの管理と初期化」と付録 B「モデル固有レジスタ（MSR）」を参照。

18.34.4. マシンチェック例外 / アーキテクチャ

インテル® Pentium® プロセッサでは、マシンチェック例外（#MC）と呼ばれる新しい例外が導入されている。この例外は、読み取りサイクル時のパリティエラーなどのハードウェア関係エラーの検出に使用される。

P6 ファミリ・プロセッサでは、検出可能で、かつマシンチェック例外を生成するエラーのタイプが拡張されている。また、マシン・チェック・エラーに関する情報を記録するための新しいマシン・チェック・アーキテクチャが追加され、回復機能も拡張されている。

マシン・チェック・アーキテクチャでは、マシン・チェック・エラーを記録するための数バンクのレポート用レジスタが用意されている。各レジスタバンクは、それぞれプロセッサ内の特定のハードウェア・ユニットに関連付けられている。マシンチェックの対象となるのは、主にバス操作と相互接続操作であるが、チェックはさらにトランスレーション・ルックアサイド・バッファ（TLB）とキャッシュの操作に対しても行われる。

マシン・チェック・アーキテクチャは、一部のエラーを自動的に訂正でき、命令の実行を確実に再起動できる。また、ハードウェアでは修復できないその他のマシンエラーをソフトウェアが修復する際に使用できるように、十分な情報を収集する。

マシンチェック例外とマシン・チェック・アーキテクチャの詳細については、第 14 章「マシン・チェック・アーキテクチャ」を参照。

18.34.5. 性能モニタリング・カウンタ

P6 ファミリ・プロセッサとインテル® Pentium® プロセッサには、内部ハードウェア動作をモニタリングするための 2 つの性能モニタリング・カウンタが用意されている。これらのカウンタは、イベントカウンタであり、デコードされた命令数、受け取った割り込み数、キャッシュ・ロード回数などのさまざまなタイプのイベント数をカウントするようにプログラムできる。付録 A「性能モニタリング・イベント」に、カウント可能なすべてのイベントの一覧が記載してある（P6 ファミリ・プロセッサの一覧は表 A-10.、インテル Pentium プロセッサの一覧は表 A-11.）。これらのカウンタは、2 つの MSR と RDMSR 命令および WRMSR 命令を使用して、セットアップ、起動、および停止を行うことができる。P6 ファミリ・プロセッサでは、新しい RDPMC 命令を使用して特定のカウンタの現在のカウント値を読み取ることができる。

性能モニタリング・カウンタは、プログラムのデバッグ、コードの最適化、システム障害の診断、またはハードウェア設計の改善に役立つ。これらのカウンタの詳細については、第 15 章「デバッグと性能モニタリング」を参照。

18.35. インテル® 286 プロセッサ・タスクの 2 つの実行方法

16 ビット・プログラムを 32 ビット IA-32 プロセッサで実行するように移植する際は、2 つの方法が考えられる。

- 16 ビット・ソフトウェア・システムを、旧オペレーティング・システム、ローダ、およびシステムビルダと一緒に 32 ビット・プロセッサに移植する。この場合は、すべてのタスクの TSS が 16 ビットの TSS になる。つまり、32 ビット・プロセッサはあたかも高速バージョンの 16 ビット・プロセッサのように使用される。
- 選択した 16 ビット・アプリケーションを、32 ビットのオペレーティング・システム、ローダ、およびシステムビルダを使用する 32 ビット・プロセッサ環境で実行できるように移植する。この場合は、286 タスクを表現するための TSS は 32 ビット TSS に変更する必要がある。16 ビットと 32 ビットの TSS を混在させることは可能であるが、そのメリットは少なく、問題は多い。32 ビット・ソフトウェア・システム内では、すべてのタスクの TSS は 32 ビットの TSS にしなければならない。16 ビットのオブジェクト・モジュールそのものを変更する必要はない。TSS は、通常はオペレーティング・システム、ローダ、またはシステムビルダによって構築される。16 ビット・コードと 32 ビット・コードを混在させる場合の詳細については、第 17 章「16 ビット・コードと 32 ビット・コードの混在」を参照。

32 ビット・プロセッサは 16 ビット・セグメント・ディスクリプタの予約ワードの内容を使用するので、このワードに値を書き込むような 16 ビット・プログラムは 32 ビット・プロセッサ上では正しく実行されないことがある。

A

性能モニタリング・イベント

付録 A
性能モニタリング・イベント

A

この付録では、IA-32プロセッサでモニタリングすることができる性能モニタリング・イベントのリストを記載している。IA-32プロセッサでは、性能モニタリング・イベントをモニタリングする機能とモニタリングできるイベントは、モデルに固有である。A.1.節「インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサの性能モニタリング・イベント」では、インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでモニタリングすることができるイベントを一覧して説明している。A.3. 節「P6 ファミリ・プロセッサの性能モニタリング・イベント」では、P6 ファミリ・プロセッサでモニタリングできるイベントを一覧して説明している。A.4. 節「インテル® Pentium® プロセッサの 性能モニタリング・イベント」では、インテル® Pentium® プロセッサでモニタリングできるイベントを一覧して説明している。

注記

以下の性能モニタリング・イベントは、性能をチューニングする際のガイドとして使用するためのものである。性能モニタリング・イベントによってレポートされる値は適切で、ソフトウェアをチューニングする際の相対的なガイドとして役立つ。既知の矛盾点は、適当な場合には本書に記載している。

A.1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの性能モニタリング・イベント

表 A-1.、表 A-2.、表 A-3. は、カウントまたはサンプリングが可能なインテル® Pentium® 4プロセッサとインテル® Xeon™プロセッサの性能モニタリング・イベントを示している。表 A-1. は非リタイアメント・イベントを、表 A-2. はリタイアメント時イベントを示している。表 A-4.、表 A-5.、表 A-6. は、表 A-2. で定義されたリタイアメント時カウントイベントの 3 つで使用できる、3 組のパラメータを説明している。表 A-7. は、非リタイアメント・イベントとリタイアメント時イベントのうち、どれが論理プロセッサ固有（TS）イベント（15.10.4. 項「性能モニタリング・イベント」を参照）であり、どれが非論理プロセッサ固有（TI）イベントであるかを示している。

インテルPentium 4プロセッサとインテルXeonプロセッサの一部の性能モニタリング・イベントはIA-32 プロセッサ・ファミリの特定のモデルでのみ利用できる。表 A-1. および表 A-2. に記載された性能モニタリング・イベントは、CPUID シグニチャがファミリ・エンコーディング 15、モデル・エンコーディング 0、1、2、または 3 に相当する

プロセッサに適用される。表 A-3. は、CPUID シグニチャがファミリ・エンコーディング 15、モデル・エンコーディング 3 に相当する IA-32 プロセッサに適用される。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント

イベント名	イベント・パラメータ	パラメータ値	説明
TC_deliver_mode			このイベントは、プロセッサ・パッケージ内のトレース・キャッシュとデコードエンジンの動作モードの持続時間（クロックサイクル数）をカウントする。このモードは、1 つまたは複数のイベント・マスク・ビットで指定する。
	ESCR 限定	MSR_TC_ESCR0 MSR_TC_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	01H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: DD 1: DB 2: DI 3: BD 4: BB 5: BI 6: ID 7: IB	ESCR[24:9] 両方の論理プロセッサが伝達モードである。 論理プロセッサ 0 は伝達モードであり、論理プロセッサ 1 は構築モードである。 論理プロセッサ 0 は伝達モードであり、論理プロセッサ 1 はホルト状態、マシンクリア状態、または長いマイクロコード・フローに移行中のいずれかである。 論理プロセッサ 0 は構築モードであり、論理プロセッサ 1 は伝達モードである。 両方の論理プロセッサが構築モードである。 論理プロセッサ 0 は構築モードであり、論理プロセッサ 1 はホルト状態、マシンクリア状態、または長いマイクロコード・フローに移行中のいずれかである。 論理プロセッサ 0 はホルト状態、マシンクリア状態、または長いマイクロコード・フローに移行中のいずれかである。論理プロセッサ 1 は伝達モードである。 論理プロセッサ 0 はホルト状態、マシンクリア状態、または長いマイクロコード・フローに移行中のいずれかである。論理プロセッサ 1 は構築モードである。
	CCCR 選択	01H	CCCR[15:13]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	イベント固有の注意事項		物理プロセッサ・パッケージ内で使用可能な論理プロセッサが 1 つである場合、このイベントマスクは、論理プロセッサ 1 がホルト状態であると解釈される。イベントマスクのビット 2 は、以前は "DELIVER" と呼ばれていた。ビット 5 は、以前は "BUILD" と呼ばれていた。
BPU_fetch_ request			このイベントは、分岐予測ユニットで指定された要求タイプの命令フェッチ要求をカウントする。要求タイプを限定するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_BPU_ESCR0 MSR_BPU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	03H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: TCMISS	ESCR[24:9] トレース・キャッシュ・ルックアップ・ミス
	CCCR 選択	00H	CCCR[15:13]
ITLB_ reference			このイベントは、命令トランスレーション・ルックアサイド・バッファ（ITLB）を使用する変換をカウントする。
	ESCR 限定	MSR_ITLB_ESCR0 MSR_ITLB_ESCR1	
	ESCR ごとのカウント番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	18H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: HIT 1: MISS 2: HIT_UC	ESCR[24:9] ITLB ヒット ITLB ミス キャッシュ不可 ITLB ヒット
	CCCR 選択	03H	CCCR[15:13]
	イベント固有の注意事項		すべてのページ参照は、ページサイズにかかわらず、実際の 4K バイト・ページとしてルックアップされる。より保存性のあるカウントを行なうには、ITMISS マスクを指定して、page_walk_type イベントを使用する。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
memory_ cancel			このイベントは、データ・キャッシュ・アドレス制御ユニット（DAC）内の各種タイプの要求のキャンセルをカウントする。キャンセルする要求のタイプを選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	02H	ESCR[31:25]
	ESCR イベントマスク	ビット 2: ST_RB_FULL 3: 64K_CONF	ESCR[24:9] 使用できるストア要求バッファがないために再生された。 64K エリアシングのため競合した。
	CCCR 選択	05H	CCCR[15:13]
	イベント固有の注意事項		All_CACHE_MISS では、キャッシュ不可メモリがカウントに含まれる。
memory_ complete			このイベントは、ロード分割、ストア分割、キャッシュ不可（UC）分割、または UC ロードの完了をカウントする。カウントする操作を選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	08H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: LSC 1: SSC	ESCR[24:9] UC/WC ロードを除いて、完了したロード分割。 完了したすべての分割ストア。
	CCCR 選択	02H	CCCR[15:13]
load_port_ replay			このイベントは、再生されたイベントをロードポートでカウントする。再生の原因を選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベント選択	04H	ESCR[31:25]
	ESCR イベントマスク	ビット 1: SPLIT_LD	ESCR[24:9] 分割ロード
	CCCR 選択	02H	CCCR[15:13]
	イベント固有の注意事項		リタイアメント時カウンティングには、ESCR1 を使用しなければならない。
store_port_replay			このイベントは、再生されたイベントをストアポートでカウントする。再生の原因を選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	05H	ESCR[31:25]
	ESCR イベントマスク	ビット 1: SPLIT_ST	ESCR[24:9] 分割ストア
	CCCR 選択	02H	CCCR[15:13]
	イベント固有の注意事項		リタイアメント時カウンティングには、ESCR1 を使用しなければならない。
MOB_load_replay			このイベントは、メモリ・オーダ・バッファ（MOB）によりロード操作が再生された場合にトリガされる。再生の原因を選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_MOB_ESCR0 MSR_MOB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	03H	ESCR[31:25]
	ESCR イベントマスク	ビット 1: NO_STA 3: NO_STD 4: PARTIAL_DATA 5: UNALGN_ADDR	ESCR[24:9] ストアアドレスが未知であったために再生された。 ストアデータが未知であったために再生された。 ロード操作およびストア操作間のデータアクセスが部分的にオーバーラップしたために再生された。 ロード操作およびストア操作間でリニアアドレスの下位 4 ビットが一致しないために再生された。
	CCCR 選択	02H	CCCR[15:13]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
page_walk_type			このイベントは、ページ・ミス・ハンドラ（PMH）が実行する各種のページウォークをカウントする。
	ESCR 限定	PMH_CR_ESCR0 PMH_CR_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	01H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: DTMISS 1: ITMISS	ESCR[24:9] データ TLB ミス用のページウォーク（ロードまたはストアのいずれか） 命令 TLB ミス用のページウォーク
	CCCR 選択	04H	CCCR[15:13]
BSQ_cache_reference			このイベントは、バスユニットによって観察されるキャッシュ参照（L2 キャッシュまたは L3 キャッシュ）をカウントする。 1 つ以上のマスクビットを指定して、アクセスのタイプ（読み出しタイプには、ロードと RFO が含まれる。書き込みタイプには、ライトバックと排出が含まれる）とアクセスの結果（ヒット、ミス）に基づいて、カウントされるアクセスを選択できる。
	ESCR 限定	BSU_CR_ESCR0 BSU_CR_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	0CH	ESCR[31:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0: RD_2ndL_HITS 1: RD_2ndL_HITE 2: RD_2ndL_HITM 3: RD_3rdL_HITS 4: RD_3rdL_HITE 5: RD_3rdL_HITM 8: RD_2ndL_MISS 9: RD_3rdL_MISS 10: WR_2ndL_MISS	ESCR[24:9] Shared（共有）L2 キャッシュ・ヒットを読み取る（ロードおよび RFO を含む）。 Exclusive（排他的）L2 キャッシュ・ヒットを読み取る（ロードおよび RFO を含む）。 Modified（修正済み）L2 キャッシュ・ヒットを読み取る（ロードおよび RFO を含む）。 共有（Shared）L3 キャッシュの読み取りのヒット（ロードおよび RFO を含む） 排他的（Exclusive）L3 キャッシュの読み取りのヒット（ロードおよび RFO を含む） 修正済み（Modified）L3 キャッシュの読み取りのヒット（ロードおよび RFO を含む） L2 キャッシュの読み取りのミス（ロードおよび RFO を含む） L3 キャッシュの読み取りのミス（ロードおよび RFO を含む） DAC からのライトバック・ルックアップによる L2 キャッシュ・ミス（発生する可能性は小さい）。
	CCCR 選択	07H	CCCR[15:13]
	イベント固有の注意事項		1: 現在のインテル® Pentium® 4 プロセッサおよびインテル® Xeon™プロセッサでは、このイベントは、ロード操作または RFO（Request For Ownership）要求を「読み出し」タイプの操作として扱う。 2: 現時点では、このイベントは、エラッタのために、キャッシュ参照を実際より 2 倍多くカウントしたり、実際より 1/2 少なくカウントする場合がある。 3: プリフェッチとして開始されたトランザクションの場合、トランザクションの内部ステータスが変化したために、トランザクションがプリフェッチでなくなったり、このイベントによって観察されるアクセス結果ステータス（ヒット、ミス）が変わる可能性がある。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
IOQ_allocation			このイベントは、バス上の各種のトランザクションをカウントする。指定されたマスクビットに一致する IOQ にトランザクションが割り当てられるたびに、カウントが生成される。割り当てられるエントリは、セクタ（64 バイト）または 8 バイトのチャンクである。 要求は再試行のたびに 1 回ずつカウントされる。イベント・マスク・ビットは、4 つのビット・フィールドを構成する。各ビット・フィールドの値の解釈によって、トランザクションのタイプが指定される。ビット・フィールド内の 1 つ以上のイベント・マスク・ビットを指定して、ビット・フィールドの値を選択できる。 各フィールド（ビット 0 ～ 4 が 1 つのフィールドである）は互いに独立しており、互いの OR 演算が可能である。要求タイプ・フィールドは、さらにビット 5 およびビット 6 と組み合わされ、2 進表現を構成する。ビット 7 およびビット 8 は、ターゲット・アドレスのメモリタイプを指定するビット・フィールドを構成する。 ビット 13 およびビット 14 は、要求のソース・エージェントを指定するビット・フィールドを構成する。ビット 15 は、読み出し操作にのみ影響を与える。このイベントは、論理式（((要求タイプ) OR ビット 5 OR ビット 6) OR（メモリタイプ)）AND（ソース・エージェント）を評価することによってトリガされる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	03H	ESCR[31:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0 ～ 4 （単一フィールド） 5:ALL_READ 6:ALL_WRITE 7:MEM_UC 8: MEM_WC 9: MEM_WT 10: MEM_WP 11: MEM_WB 13: OWN 14: OTHER 15: PREFETCH	ESCR[24:9] バス要求タイプ（無効またはデフォルトの場合には 00001 を使用）。 読み取りエントリをカウントする。 書き込みエントリをカウントする。 UC メモリ・アクセス・エントリをカウントする。 WC メモリ・アクセス・エントリをカウントする。 ライトスルー（WT）メモリ・アクセス・エントリをカウントする。 ライト・プロテクト（WP）メモリ・アクセス・エントリをカウントする。 WB メモリ・アクセス・エントリをカウントする。 プロセッサによってドライブされたストア要求をすべてカウントする。これは、他のプロセッサや DMA とは対立するものである。 他のプロセッサや DMA によってドライブされた要求をすべてカウントする。 カウントに HW および SW のプリフェッチ要求を含める。
	CCCR 選択	06H	CCCR[15:13]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	イベント固有の注意事項		1: PREFETCH ビットがクリアされている場合は、プリフェッチによってフェッチされたセクタはカウントから除外される。PREFETCH ビットがセットされている場合は、すべてのセクタまたはチャンクの読み出しがカウントされる。 2: 二重カウントを避けるために、CCCR 内でエッジトリガを指定する。 3: 解釈したビット・フィールドの値をトランザクション・タイプにマッピングする方法は、インテル Pentium 4 プロセッサ・ファミリのプロセッサ・モデルによって異なる。性能モニタリング・イベントをプログラムするアプリケーションは、このイベントを使用する場合、CPUID 命令を使用してプロセッサ・モデルを検出する必要がある。以下に説明するように、このイベントをトリガする論理式は、CPUID 機能フラグの MODEL フィールドのエンコーディングが 0、1、2 に等しい場合に適用される。 4a: CPUID の MODEL フィールドのエンコーディングが 2 に等しいかそれより大きいインテル Pentium 4 プロセッサやインテル Xeon プロセッサでは、このイベントは、論理式（(要求タイプ）and （ビット 5 or ビット 6）and （メモリタイプ）and （ソース・エージェント)）の評価によってトリガされる。 4b: CPUID の MODEL フィールドのエンコーディングが 2 より小さいインテル Pentium 4 プロセッサやインテル Xeon プロセッサでは、このイベントは、論理式 [（(要求タイプ）or （ビット 5 or ビット 6）or （メモリタイプ）] and （ソース・エージェント）の評価によってトリガされる。 5: ALL_READ または ALL_WRITE が指定されている場合は、メモリタイプのイベント・マスク・ビットは無視される。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	イベント固有の注意事項（続き）		6: インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサの初期のプロセッサでは、このイベントは CPL を無視する。ユーザ要求と OS 要求の両方がカウントに含まれる。この動作は、CPUID シグニチャが 0xF27（ファミリ 15、モデル 2、ステッピング 7）のインテル Pentium 4 プロセッサとインテル Xeon プロセッサ以降では修正されている。 7: ライトスルー（WT）およびライトプロテクト（WP）メモリタイプの場合、このイベントは、64 バイト・セクタの数として読み出しをカウントする。書き込みは、個々のチャンク単位でカウントされる。 8: キャッシュ不可（UC）メモリタイプの場合、このイベントは、割り当てられた 8 バイト・チャンクの数をカウントする。 9: CPUID シグニチャが 0xf27 よりも小さいインテル Pentium 4 プロセッサとインテル Xeon プロセッサでは、MSR_FSB_ESCR0 のみ利用できる。
IOQ_active_entries			このイベントは、IOQ 内のアクティブなエントリ数（15 で切り捨て）をカウントする。割り当てられるエントリは、セクタ（64 バイト）または 8 バイトのチャンクである。 このイベントは、IOQ_allocation と組み合わせてプログラムしなければならない。1 つ以上のイベント・マスク・ビットを指定して、カウントされるトランザクションを選択できる。
	ESCR 限定	MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR1: 2、3	
	ESCR イベント選択	01AH	ESCR[30:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0 ～ 4 （単一フィールド） 5: ALL_READ 6: ALL_WRITE 7: MEM_UC 8: MEM_WC 9: MEM_WT 10: MEM_WP 11: MEM_WB 13: OWN 14: OTHER 15: PREFETCH	ESCR[24:9] バス要求のタイプ（無効またはデフォルトの場合は 00001 を使用）。 読み出しエントリをカウントする。 書き込みエントリをカウントする。 UC メモリ・アクセス・エントリをカウントする。 WC メモリ・アクセス・エントリをカウントする。 ライトスルー（WT）メモリ・アクセス・エントリをカウントする。 ライトプロテクト（WP）メモリ・アクセス・エントリをカウントする。 WB メモリ・アクセス・エントリをカウントする。 他のプロセッサや DMA ではなく、当該プロセッサがドライブしたすべてのストア要求をカウントする。 他のプロセッサまたは DMA がドライブしたすべてのストア要求をカウントする。 HW および SW プリフェッチ要求をカウントに含める。
	CCCR 選択	06H	CCCR[15:13]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	イベント固有の注意事項		1: ESCR0 および ESCR1 内で希望のマスクビットを指定する。 2: マスクビットについては、ioq_allocation イベントを参照。 3: サイクル数をカウントするときは、エッジ・トリガリングを使用してはならない。 4: 解釈したビット・フィールドの値をトランザクション・タイプにマッピングする方法は、インテル Pentium 4 プロセッサ・ファミリのプロセッサ・モデルによって異なる。性能モニタリング・イベントをプログラムするアプリケーションは、このイベントを使用する場合、CPUID 命令を使用してプロセッサ・モデルを検出する必要がある。以下に説明するように、このイベントをトリガする論理式は、CPUID 機能フラグの MODEL フィールドのエンコーディングが 0、1、2 に等しい場合に適用される。 5: CPUID の MODEL フィールドのエンコーディングが 2 に等しいかそれより大きいインテル Pentium 4 プロセッサや Xeon プロセッサでは、このイベントは、論理式（(要求タイプ）and （ビット 5 or ビット 6）and（メモリタイプ）and（ソース・エージェント)）の評価によってトリガされる。 6a: CPUID の MODEL フィールドのエンコーディングが 2 より小さいインテル Pentium 4 プロセッサや Xeon プロセッサでは、このイベントは、論理式 [（(要求タイプ）or （ビット 5 or ビット 6）or （メモリタイプ）] and（ソース・エージェント）の評価によってトリガされる。 6b: ALL_READ または ALL_WRITE が指定されている場合は、メモリタイプのイベント・マスク・ビットは無視される。 7: 現在のインテル Pentium 4 プロセッサや Xeon プロセッサでは、このイベントは CPL を無視することがわかっている。ユーザ要求と OS 要求の両方がカウントに含まれる。 8: 割り当てられるエントリは、フルライン（64 バイト）または個々の 8 バイト・チャンク単位である。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
FSB_data_activity			このイベントは、フロント・サイド・バスで発生する DRDY イベントまたは DBSY イベントごとに 1 回インクリメントする。このイベントでは、特定の DRDY イベントまたは DBSY イベントを選択できる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	17H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: DRDY_DRV	ESCR[24:9] このプロセッサがバス上にデータをドライブしたときにカウントする。書き込みと暗黙的なライトバックを含む。不完全な書き込みの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、フルラインの書き込みの場合は（通常は連続するバスクロックで）4 プロセッサ・クロックの間アサートされる。
		1: DRDY_OWN	このプロセッサがバスからデータを読み出したときにカウントする。ロードと一部の PIC トランザクションを含む。不完全な読み出しの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、フルラインの読み出しの場合は（通常は連続するバスクロックで）4 プロセッサ・クロックの間アサートされる。
		2: DRDY_OTHER	ドライブされる DRDY イベントをカウントする。 所有されている、サンプリングされた DRDY イベントをカウントする。 データがバス上にあるが、プロセッサによってサンプリングされていないときにカウントする。データはこのプロセッサによってドライブされていても、ドライブされていなくてもかまわない。不完全なトランザクションの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、フルラインのトランザクションの場合は（通常は連続するバスクロックで）4 プロセッサ・クロックの間アサートされる。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
		3: DBSY_DRV	このプロセッサが、データをドライブするために、次のバスサイクルで使用するバスを予約したときにカウントする。フルラインの書き込みの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、不完全なラインの書き込みの場合はアサートされない。バスがキャッシュ・ロックの完了を待機してストールした場合、（連続するバスクロックで）複数回アサートされるときがある。
		4: DBSY_OWN	このプロセッサがサンプリングするデータをドライブするために、何らかのエージェントが次のバスサイクルで使用するバスを予約したときにカウントする。フルラインの書き込みの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、不完全なラインの書き込みの場合はアサートされない。何らかの理由でバスがストールした場合、（すべて別々の 1 バス・クロックで）複数回アサートされるときがある。
		5:DBSY_OTHER	このプロセッサがサンプリングしないデータをドライブするために、何らかのエージェントが次のバスサイクルで使用するバスを予約したときにカウントする。データはこのプロセッサによってドライブされていても、ドライブされていなくてもかまわない。不完全なトランザクションの場合は 2 プロセッサ・クロック・サイクルの間アサートされ、フルラインのトランザクションの場合は（通常は連続するバスクロックで）4 プロセッサ・クロックの間アサートされる。
	CCCR 選択	06H	CCCR[15:13]
	イベント固有の注意事項		二重カウントを避けるために、CCCR MSR 内でエッジトリガを指定する。 DRDY_OWN と DRDY_OTHER は互いに両立しない。同様に、DBSY_OWN と DBSY_OTHER も互いに両立しない。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
BSQ_allocation			このイベントは、指定されたマスク・ビット・エンコーディングに応じて、バス・シーケンス・ユニット（BSQ）内の割り振りをカウントする。イベント・マスク・ビットは、4 つのサブグループで構成される。 • 要求のタイプ • 要求の長さ • メモリのタイプ • 大部分の独立ビット（5、6、7、8、9、10 ビット） エンコーディングは、サブグループごとに指定する。
	ESCR 限定	MSR_BSU_ESCR0	
	ESCR ごとのカウンタ番号	ESCR0: 0、1	
	ESCR イベント選択	05H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: REQ_TYPE0 1: REQ_TYPE1 2: REQ_LEN0 3: REQ_LEN1 5: REQ_IO_TYPE 6: REQ_LOCK_TYPE 7: REQ_CACHE_TYPE 8: REQ_SPLIT_TYPE 9: REQ_DEM_TYPE 10: REQ_ORD_TYPE 11: MEM_TYPE0 12: MEM_TYPE1 13: MEM_TYPE2	ESCR[24:9] 要求のタイプのエンコーディング（ビット 0 とビット 1）： 0 － 読み出し（読み出しの無効化を除く）。 1 - 読み出しの無効化。 2 - 書き込み（ライトバックを除く）。 3 - ライトバック（キャッシュからの排出）。 （パブリック） 要求の長さのエンコーディング（ビット 2、3）： 0 － 0 チャンク 1 － 1 チャンク 3 － 8 チャンク 要求のタイプは入力または出力である。 要求のタイプはバスロックである。 要求のタイプはキャッシュ可能である。 要求のタイプは、8 バイト境界を超えるバスの 8 バイト・チャンクである。 セットされている場合、要求のタイプはデマンドである。 0 の場合、要求のタイプは HW.SW プリフェッチである。 要求はオーダード・タイプである。 メモリタイプのエンコーディング（ビット 11 ～ 13）： 0 － UC 1 － USWC 4 － WT 5 － WP 6 － WB

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	CCCR 選択	07H	CCCR[15:13]
	イベント固有の注意事項		1: 二重カウントを避けるために、CCCR 内でエッジトリガを指定する。 2: L2 キャッシュから L3 キャッシュへのライトバックは、個別のエントリとしてカウントされる。これはバスに対する要求に割り当てられたエントリに追加される。 3: WB メモリタイプに対する読み出し要求があると、（ターゲット・アドレスを含む）64 バイト・セクタに対する要求が発生し、それに続いて隣接セクタに対するプリフェッチ要求が発生する。 4: CPUID のモデル・エンコーディング値が 0 および 1 に等しいインテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、割り当てられる BSQ エントリに、デマンドセクタとプリフェッチされる第 2 のセクタが含まれる。 5: データチャンクに割り当てられる BSQ エントリは、64 バイトより小さい任意の要求である。 6a: このイベントは、データアドレスがモジュロ 64 バイト境界にまたがる分割型トランザクション要求を、実際より少なくカウントする。 6b: このイベントは、WC または UC アドレスからの 16 バイト・オペランドの読み出し要求を、実際より少なくカウントする。 6c: このイベントは、ダブルワードであるストア・オペランドから発生した不完全な WC 要求を、実際より少なくカウントする。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
bsq_active_entries			このイベントは、BSQ 内の割り当ての際にサブイベント・マスク基準を満たす BSQ エントリのうち、現在アクティブ（有効）になっているエントリの数（15 で切り捨て）を表す。アクティブな BSQ エントリは、割り当て解除されるまで、BSQ 上に割り当てられている。 エントリが割り当て解除されるのは、必ずしも要求が実現されたことを意味しない。このイベントは、BSQ_allocation と組み合わせてプログラムしなければならない。1 つ以上のイベント・マスク・ビットを指定して、カウントされるトランザクションを選択できる。
	ESCR 限定	ESCR1	
	ESCR ごとのカウンタ番号	ESCR1: 2、3	
	ESCR イベント選択	06H	ESCR[30:25]
	ESCR イベントマスク		ESCR[24:9]
	CCCR 選択	07H	CCCR[15:13]
	イベント固有の注意事項		1: ESCR0 および ESCR1 内で希望のマスクビットを指定する。 2: マスクビットについては、BSQ_allocation イベントを参照。 3: サイクル数をカウントするときは、エッジ・トリガリングを使用してはならない。 4: このイベントを使用して、BSQ 内の割り当てから割り当て解除までのトランザクションのレイテンシを推定できる。BSQ_allocation によって観察されるレイテンシには、FSB のレイテンシと追加のオーバーヘッドが含まれる。追加のオーバーヘッドには、2 つの要求（デマンドセクタのフェッチと隣接セクタのプリフェッチ）を発行するのに必要な時間が含まれる。隣接セクタのプリフェッチはデマンドフェッチより優先度が低いため、負荷の大きいシステムでは、隣接セクタのプリフェッチは次のバス・アービトレーションまで待たなければならない確率が高い。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
SSE_input_assist			このイベントは、DAZ ビットがセットされないときに、SSE、SSE2、SSE3 操作の入力オペランドに最も重要なデノーマル・ソース・オペランドを指定して問題を処理するようにアシストが要求された回数をカウントする。このイベントを使用するには、イベントマスクのビット 15 をセットする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	34H	ESCR[31:25]
	ESCR イベントマスク	15: ALL	ESCR[24:9] SSE および SSE2 のすべての μop についてアシストをカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		1: アシストに対する要求の中には、実際には実行されないものもある。このイベントは、性能上のペナルティが発生しない、リタイアされないパス上の命令からのアシスト要求をカウントしてしまうため、要求の数を実際より多くカウントすることがわかっている。非偽の μop に対してのみ、アシストが実際に実行される。このイベントで大量のカウントが検出された場合は、DAZ ビットまたは FTZ ビットをセットする必要があるか、この状態を解消するためにソースコードを修正する必要があることを示している。 2: SSE、SSE2、または SSE3 操作がアシストを必要とする一般的な状況には次の 2 つがある。(1) デノーマル定数が入力として使用され、かつ、DAZ（デノーマル・ゼロ）モードがセットされていない場合、(2) 以前の SSE、SSE2、または SSE3 操作のアンダーフローした結果を入力オペランドが使用し、かつ、DAZ モードも FTZ（フラッシュ・ツー・ゼロ）モードもセットされていない場合。DAZ モードをイネーブルにすると、SSE、SSE2、または SSE3 操作は第 1 の状況でのアシストの必要性がなくなる。FTZ モードをイネーブルにすると、SSE、SSE2、または SSE3 操作は第 2 の状況でのアシストの必要性がなくなる。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
packed_SP_uop			このイベントは、検出用のイベントマスクを介して指定されるパックド単精度 µop ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	08H	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] パックド単精度オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		1: 命令に複数のパックド SP µop が含まれる場合は、イベントマスクで指定されるパックド SP µop がそれぞれカウントされる。 2: このメトリックは、繰り返し移動ストリング内のパックドメモリ µop のインスタンスをカウントする。
packed_DP_uop			このイベントは、検出用のイベントマスクを介して指定されるパックド倍精度 µop ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	0CH	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] パックド倍精度オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		命令に複数のパックド DP µop が含まれる場合は、イベントマスクで指定されるパックド DP µop がそれぞれカウントされる。
scalar_SP_uop			このイベントは、検出用のイベントマスクを介して指定されるスカラ単精度 µop ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	0AH	ESCR[31:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] スカラ単精度オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		命令に複数のスカラ SP µop が含まれる場合は、イベントマスクで指定されるスカラ SP µop がそれぞれカウントされる。
scalar_DP_uop			このイベントは、検出用のイベントマスクを介して指定されるスカラ倍精度 µop ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	0EH	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] スカラ倍精度オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		命令に複数のスカラ DP µop が含まれる場合は、イベントマスクで指定されるスカラ DP µop がそれぞれカウントされる。
64bit_MMX_uop			このイベントは、64 ビット SIMD オペランドを操作する MMX 命令ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	02H	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] メモリまたは MMX テクノロジ・レジスタ内の 64 ビット SIMD 整数オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		命令に複数の 64 ビット MMX µop が含まれる場合は、イベントマスクで指定される 64 ビット MMX µop がそれぞれカウントされる。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
128bit_MMX_uop			このイベントは、128 ビット SIMD オペランドを操作する整数 SIMD SSE2 命令ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	1AH	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] メモリまたは XMM レジスタ内の 128 ビット SIMD 整数オペランドを操作している µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		命令に複数の 128 ビット MMX µop が含まれる場合は、イベントマスクで指定される 128 ビット MMX µop がそれぞれカウントされる。
x87_FP_uop			このイベントは、検出用のイベントマスクを介して指定される x87 浮動小数点 µop ごとにインクリメントする。
	ESCR 限定	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	04H	ESCR[31:25]
	ESCR イベントマスク	ビット 15: ALL	ESCR[24:9] x87 FP µop をすべてカウントする。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		1: 1 つの命令に 2 つ以上の x87 FP µop が含まれている場合は、イベントマスクによって指定された各 x87 FP µop がカウントされる。 2: このイベントは、レジスタ間のロード、ストア、移動用の x87 FP µop はカウントしない。
TC_misc			このイベントは、TC によって検出された各種のイベントをカウントする。カウンタは、イベント発生ごとに 2 回カウントする。
	ESCR 限定	MSR_TC_ESCR0 MSR_TC_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	06H	ESCR[31:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	CCCR 選択	01H	CCCR[15:13]
	ESCR イベントマスク	ビット 4: FLUSH	ESCR[24:9] フラッシュ数。
global_power_events			このイベントは、プロセッサが停止していない時間を累積する。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	013H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: 実行中	ESCR[24:9] プロセッサはアクティブである（HLT STPCLK の処理中およびスロットル中を含む）。
	CCCR 選択	06H	CCCR[15:13]
tc_ms_xfer			このイベントは、µop の伝達が TC から MS ROM に変更された回数をカウントする。
	ESCR 限定	MSR_MS_ESCR0 MSR_MS_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	05H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: CISC	ESCR[24:9] TC から MS への移行が発生した。
	CCCR 選択	0H	CCCR[15:13]
uop_queue_writes			このイベントは、µop キューに書き込まれた有効な µop の数をカウントする。1 つ以上のマスクビットを指定して、書き込みのソースタイプを選択する。
	ESCR 限定	MSR_MS_ESCR0 MSR_MS_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	09H	ESCR[31:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0: FROM_TC_BUILD 1: FROM_TC_DELIVER 2: FROM_ROM	ESCR[24:9] TC 構築モードから書き込まれる µop をカウントする。 TC 伝達モードから書き込まれる µop をカウントする。 マイクロコード ROM から書き込まれる µop をカウントする。
	CCCR 選択	0H	CCCR[15:13]
retired_mispred_branch_type			このイベントは、リタイアする予測ミスの分岐をタイプごとにカウントする。
	ESCR 限定	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	05H	ESCR[30:25]
	ESCR イベントマスク	ビット 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	ESCR[24:9] 条件付きジャンプ 間接呼び出し分岐 リターン分岐 リターン、間接呼び出し、または間接ジャンプ
	CCCR 選択	02H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、以下の場合、条件付き分岐の数を実際より多くカウントする。 a: 分岐の予測ミスのために、トレース・キャッシュと伝達エンジンが新しいトレースを作成した。 b: プロセッサのパイプラインがクリア中のとき。
retired_branch_type			このイベントは、リタイアする分岐をタイプごとにカウントする。1 つ以上のマスクビットを指定して、カウントする分岐の条件をタイプごとに制限できる。
	ESCR 限定	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 4、5 ESCR1: 6、7	
	ESCR イベント選択	04H	ESCR[30:25]

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	ESCR[24:9] 条件付きジャンプ 直接または間接呼び出し リターン分岐 リターン、間接呼び出し、または間接ジャンプ
	CCCR 選択	02H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、以下の場合、条件付き分岐の数を実際より多くカウントする。 a: 分岐の予測ミスのために、トレース・キャッシュと伝達エンジンが新しいトレースを作成した。 b: プロセッサのパイプラインがクリア中のとき。
resource_stall			このイベントは、アロケータでのストールの発生やレイテンシを監視する。
	ESCR 限定	MSR_ALF_ESCR0 MSR_ALF_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	
	ESCR イベント選択	01H	ESCR[30:25]
	イベントマスク	ビット 5: SBFULL	ESCR[24:9] ストアバッファの不足によるストール。
	CCCR 選択	01H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、プロセッサ・ファミリの全モデルでサポートされているとは限らない。
WC_Buffer			このイベントは、イベントマスクで選択されたライト・コンバイニング・バッファ処理をカウントする。
	ESCR 限定	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 8、9 ESCR1: 10、11	
	ESCR イベント選択	05H	ESCR[30:25]
	イベントマスク	ビット 0: WCB_EVICTS 1: WCB_FULL_ EVICT	ESCR[24:9] あらゆる原因に基づく WC バッファ送出。 WC バッファ送出 : WC バッファは利用不可。

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	CCCR 選択	05H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、ヒットで修正された条件によって満たされたライト・コンバイニング・バッファが顕著な影響をもたらさない限り、コストが大きい 64K 別名定義ケース（ストアを含む 64K 別名定義）のサブセットを検出するのに有効である。
b2b_cycles			このイベントは、サブイベント・マスク・ビット 1 ～ 6 を使用して、連続バスサイクル数をカウントするように設定できる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	016H	ESCR[30:25]
	イベントマスク	ビット	ESCR[24:9]
	CCCR 選択	03H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、プロセッサ・ファミリの全モデルでサポートされているとは限らない。
bnr			このイベントは、サブイベント・マスク・ビット 0 ～ 2 を使用して、バスが対応していない条件をカウントするように設定できる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	08H	ESCR[30:25]
	イベントマスク	ビット	ESCR[24:9]
	CCCR 選択	03H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、プロセッサ・ファミリの全モデルでサポートされているとは限らない。
snoop			このイベントは、サブイベント・マスク・ビット 2、6、7 を使用して、スヌープヒットで修正されたバス・トラフィックをカウントするように設定できる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	

表 A-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの非リタイアメント・カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	06H	ESCR[30:25]
	イベントマスク	ビット	ESCR[24:9]
	CCCR 選択	03H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、プロセッサ・ファミリの全モデルでサポートされているとは限らない。
Response			このイベントは、サブイベント・マスク・ビット 1、2、8、9 を使用して、各種の応答をカウントするように設定できる。
	ESCR 限定	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 0、1 ESCR1: 2、3	
	ESCR イベント選択	04H	ESCR[30:25]
	イベントマスク	ビット	ESCR[24:9]
	CCCR 選択	03H	CCCR[15:13]
	イベント固有の注意事項		このイベントは、プロセッサ・ファミリの全モデルでサポートされているとは限らない。

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント

イベント名	イベント・パラメータ	パラメータ値	説明
front_end_event			このイベントは、フロント・エンド・タグ付け機構を介して指定される、タグ付けされた μop のリタイアメントをカウントする。イベントマスクにより、偽または非偽の μop を指定する。
	ESCR 限定	MSR_CRU_ESCR2、MSR_CRU_ESCR3	
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	
	ESCR イベント選択	08H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUS 1: BOGUS	ESCR[24:9] マークされた μop は偽ではない。 マークされた μop は偽である。
	CCCR 選択	05H	CCCR[15:13]
	PEBS サポート可能	はい	
	タグ付け用に別の MSR が必要	選択された ESCR/MSR_TC_PRECISE_EVENT	Front_end タグ付けでサポートされるメトリックのリスト（表 A-4.）を参照。
execution_event			このイベントは、実行タグ付け機構を介して指定される、タグ付けされた μop のリタイアメントをカウントする。イベントマスクにより、1 ～ 4 つのタイプの μop を、タグ付けされる偽または非偽の μop として指定できる。
	ESCR 限定	MSR_CRU_ESCR2、MSR_CRU_ESCR3	
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	
	ESCR イベント選択	0CH	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUS0 1: NBOGUS1 2: NBOGUS2 3: NBOGUS3 4: BOGUS0 5: BOGUS1 6: BOGUS2 7: BOGUS3	ESCR[24:9] マークされた μop は偽ではない。 マークされた μop は偽ではない。 マークされた μop は偽ではない。 マークされた μop は偽ではない。 マークされた μop は偽である。 マークされた μop は偽である。 マークされた μop は偽である。 マークされた μop は偽である。
	CCCR 選択	05H	CCCR[15:13]

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	イベント固有の注意事項		偽 / 非偽の µop を指定するための 4 つの各スロットは、ESCR 内の 4 つの TagValue ビットで調整する必要がある（例えば、NBOGUS0 は、ESCR 内の TagValue フィールドの最下位ビットが 1 でなければならないし、NBOGUS1 は、TagValue フィールドの最下位ビットの次のビットが 1 でなければならない）。
	PEBS サポート可能	はい	
	タグ付け用に別の MSR が必要	アップストリーム・イベント用の ESCR	実行タグ付けでサポートされるメトリックのリスト（表 A-5.）を参照。
replay_event			このイベントは、再生タグ付け機構を介して指定される、タグ付けされた µop のリタイアメントをカウントする。イベントマスクにより、偽または非偽の µop を指定する。
	ESCR 限定	MSR_CRU_ESCR2、MSR_CRU_ESCR3	
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	
	ESCR イベント選択	09H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUS 1: BOGUS	ESCR[24:9] マークされた µop は偽ではない。 マークされた µop は偽である。
	CCCR 選択	05H	CCCR[15:13]
	イベント固有の注意事項		別の MSR を使用して、タグ付けされた µop のカウントをサポートする。
	PEBS サポート可能	はい	
	タグ付け用に別の MSR が必要	IA32_PEBS_ENABLE、MSR_PEBS_MATRIX_VERT、選択された ESCR	再生タグ付けでサポートされるメトリックのリスト（表 A-6.）を参照。
instr_retired			このイベントは、1 クロック・サイクルの間にリタイアされる命令をカウントする。マスクビットにより、命令が偽か非偽か（また、フロントエンドのタギング機構によって命令にタグが付けられているかどうか）を指定する。
	ESCR 限定	MSR_CRU_ESCR0、MSR_CRU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベント選択	02H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUSNTAG 1: NBOGUSTAG 2: BOGUSNTAG 3: BOGUSTAG	ESCR[24:9] タグ付けされていない非偽命令。 タグ付けされている非偽命令。 タグ付けされていない偽命令。 タグ付けされている偽命令。
	CCCR 選択	04H	CCCR[15:13]
	イベント固有の注意事項		1: イベントカウントは、イベント検出が有効にされたときのプロセッサのマイクロアーキテクチャ・ステートによって異なる。 2: 複雑な μop フローを持つ一部の IA-32 命令が、リタイアメント前に割り込みをかけられた場合、2 回以上カウントされることがある。
	PEBS サポート可能	いいえ	
uops_retired			このイベントは、1 クロック・サイクルの間にリタイアされる μop をカウントする。マスクビットにより、偽または非偽を指定する。
	ESCR 限定	MSR_CRU_ESCR0、MSR_CRU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	
	ESCR イベント選択	01H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUS 1: BOGUS	ESCR[24:9] マークされた μop は偽ではない。 マークされた μop は偽である。
	CCCR 選択	04H	CCCR[15:13]
	イベント固有の注意事項		P6: EMON_UOPS_RETIRED
	PEBS サポート可能	いいえ	
uop_type			このイベントとフロントエンドのリタイアメント時機構を組み合わせて使用すると、ロード μop とストア μop にタグを付けられる。
	ESCR 限定	MSR_RAT_ESCR0、MSR_RAT_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	
	ESCR イベント選択	02H	ESCR[31:25]

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 1: TAGLOADS 2: TAGSTORES	ESCR[24:9] µop はロード操作である。 µop はストア操作である。
	CCCR 選択	02H	CCCR[15:13]
	イベント固有の注意事項		TAGLOADS および TAGSTORES マスクビットをセットしても、カウンタはインクリメントされない。これらのマスクビットは、µop のタギングにのみ使用される。
	PEBS をサポートするか	いいえ	
branch_retired			このイベントは、分岐のリタイアメントをカウントする。実施された分岐、実施されなかった分岐、予測された分岐、および誤って予測された分岐の任意の組み合わせを選択するには、1 つまたは複数のマスクビットで指定する。
	ESCR 限定	MSR_CRU_ESCR2 MSR_CRU_ESCR3	ESCR MSR のアドレスについては、表 15-5. を参照のこと。
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	各 ESCR に対応付けられたカウンタ番号が示されている。性能カウンタおよび対応する CCCR は、表 15-5. から得ることができる。
	ESCR イベント選択	06H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] 予測されて実施されなかった分岐 誤って予測されて実施されなかった分岐 予測されて実施された分岐 誤って予測されて実施された分岐
	CCCR 選択	05H	CCCR[15:13]
	イベント固有の注意事項		P6: EMON_BR_INST_RETIRED
	PEBS をサポートするか	いいえ	
mispred_branch_retired			このイベントは、誤って予測された IA-32 分岐命令のリタイアメントを表す。
	ESCR 限定	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	
	ESCR イベント選択	03H	ESCR[31:25]

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0: NBOGUS	ESCR[24:9] リタイアされた命令は偽ではない。
	CCCR 選択	04H	CCCR[15:13]
	PEBS をサポートするか	いいえ	
x87_assist			このイベントは、特別な処理を要求した x87 命令のリタイアメントをカウントする。アシストのタイプを選択するには、1 つまたは複数のイベント・マスク・ビットで指定する。
	ESCR 限定	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	
	ESCR イベント選択	03H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: FPSU 1: FPSO 2: POAO 3: POAU 4: PREA	ESCR[24:9] FP スタック・アンダーフローを処理する。 FP スタック・オーバーフローを処理する。 x87 出力オーバーフローを処理する。 x87 出力アンダーフローを処理する。 x87 入力アシストを処理する。
	CCCR 選択	05H	CCCR[15:13]
	PEBS をサポートするか	いいえ	
machine_clear			このイベントは、プロセッサのパイプライン全体がクリアされている間、指定されたマスクビットに応じてインクリメントする。原因を選択するには、マスクビットを 1 つ指定する。
	ESCR 限定	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	ESCR ごとのカウンタ番号	ESCR2: 12、13、16 ESCR3: 14、15、17	
	ESCR イベント選択	02H	ESCR[31:25]

表 A-2. インテル Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの
リタイアメント時カウント用性能モニタリング・イベント（続き）

イベント名	イベント・パラメータ	パラメータ値	説明
	ESCR イベントマスク	ビット 0: CLEAR 2: MOCLEAR 3: SMCLEAR	ESCR[24:9] 何らかの原因でプロセッサがクリアされている間、多数のサイクルの一部についてカウントする。持続時間に対する発生回数をカウントする場合にのみ、このビットにエッジ・トリガリングを使用する。 メモリ・オーダリングの問題のためにマシンがクリアされるたびにインクリメントする。 自己修正コードの問題でプロセッサがクリアされるたびにインクリメントする。
	CCCR 選択	05H	CCCR[15:13]
	PEBS をサポートするか	いいえ	

表 A-3. モデル固有の性能モニタリング・イベント（モデル・エンコーディング 3 のみ）

イベント名	イベント・パラメータ	パラメータ値	説明
instr_ completed			このイベントは、1 クロック・サイクルの間に完了しリタイアされる命令をカウントする。マスクビットにより、命令が偽か非偽かを指定する。
	ESCR 限定	MSR_CRU_ESCR0、MSR_CRU_ESCR1	
	ESCR ごとのカウンタ番号	ESCR0: 12、13、16 ESCR1: 14、15、17	
	ESCR イベント選択	07H	ESCR[31:25]
	ESCR イベントマスク	ビット 0: NBOGUS 1: BOGUS	ESCR[24:9] 非偽命令 偽命令
	CCCR 選択	04H	CCCR[15:13]
	イベント固有の注意事項		このメトリックは、命令が開始した回数ではなく完了した命令をカウントする点で、instr_retired とは異なる。
	PEBS をサポートするか	いいえ	

表 A-4. Front_end タグ付けで使用可能なメトリックのリスト（Front_end イベントのみ）

フロント・エンド・メトリック [1]	MSR_TC_PRECISE_ EVENT MSR ビット・フィールド	追加の MSR	Front_end_event の イベントマスク値
memory_loads	なし	イベント Uop_Type に対応する ESCR の TAGLOADS ビットをセットする。	NBOGUS
memory_stores	なし	イベント Uop_Type に対応する ESCR の TAGSTORES ビットをセットする。	NBOGUS

注 :

1. 浮動小数点スタックのオーバーフローまたはアンダーフローがある場合は、フロント・エンド・イベントが実際より少なくカウントされることがある。

表 A-5. 実行タグ付けで使用可能なメトリックのリスト（実行イベントのみ）

実行メトリック	アップストリーム ESCR	アップストリーム ESCR の TagValue	execution_event のイベントマスク値
packed_SP_retired	イベントマスクのすべてのビット、packed_SP_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
packed_DP_retired	イベントマスクのすべてのビット、packed_DP_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
scalar_SP_retired	イベントマスクのすべてのビット、scalar_SP_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
scalar_DP_retired	イベントマスクのすべてのビット、scalar_DP_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
128_bit_MMX_retired	イベントマスクのすべてのビット、128_bit_MMX_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
64_bit_MMX_retired	イベントマスクのすべてのビット、64_bit_MMX_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
X87_FP_retired	イベントマスクのすべてのビット、x87_FP_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0
X87_SIMD_memory_moves_retired	イベントマスクの ALLP0 ビット、ALLP2 ビット、X87_SIMD_moves_uop の ESCR の TagUop ビットをセットする。	1	NBOGUS0

表 A-6. 再生タグ付けで使用可能なメトリックのリスト（再生イベントのみ）

再生メトリック [1]	セットする IA32_PEBS _ENABLE フィールド	セットする MSR_PEBS_ MATRIX_VERT ビット・フィールド	追加の MSR/ イベント	Replay_event のイベント マスク値
1stL_cache_ load_miss_retired	ビット 0、ビット 24、ビット 25	ビット 0	なし	NBOGUS
2ndL_cache_ load_miss_retired[2]	ビット 1、ビット 24、ビット 25	ビット 0	なし	NBOGUS
DTLB_load_ miss_retired	ビット 2、ビット 24、ビット 25	ビット 0	なし	NBOGUS
DTLB_store_ miss_retired	ビット 2、ビット 24、ビット 25	ビット 1	なし	NBOGUS
DTLB_all_miss_ retired	ビット 2、ビット 24、ビット 25	ビット 0、ビット 1	なし	NBOGUS
Tagged_mispred_b ranch	ビット 15、ビット 16、ビット 24、ビット 25	ビット 4	なし	NBOGUS
MOB_load_ replay_retired[3]	ビット 9、ビット 24、ビット 25	ビット 0	MOB_load_replay イベントを選択し、PARTIAL_DATA ビットおよび UNALGN_ADDR ビットをセットする。	NBOGUS
split_load_retired	ビット 10、ビット 24、ビット 25	ビット 0	MSR_SAAT_ESCR 1 MSR と共に load_port_replay イベントを選択し、SPLIT_LD マスクビットをセットする。	NBOGUS
split_store_retired	ビット 10、ビット 24、ビット 25	ビット 1	MSR_SAAT_ESCR 0 MSR と共に store_port_replay イベントを選択し、SPLIT_ST マスクビットをセットする。	NBOGUS

注 :

1. µop のタイプによってはタグ付けできないものがある。例えば、I/O 操作、UC アクセス、ロックされたアクセス、リターン、far 転送などである。
2. リタイアされた L2 ミスのイベントは、すべての L2 ミスをカウントするわけではない。高速検出ロジックがミスとして検出した参照だけがカウントに含まれ、後でミスと判明した参照はカウントに含まれない。
3. MOB 再生の原因はいくつかあるが、このイベントマスク設定でイベントがカウントされるのは、別の方法で転送されるロードからのデータが、先行するストアからのデータのアライメントの合ったサブセットでない場合である。

表 A-7. イベントマスクによる論理プロセッサの制限

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
Non-Retirement	BPU_fetch_request	ビット 0: TCMISS	TS
	BSQ_allocation	ビット	
		0: REQ_TYPE0	TS
		1: REQ_TYPE1	TS
		2: REQ_LEN0	TS
		3: REQ_LEN1	TS
		5: REQ_IO_TYPE	TS
		6: REQ_LOCK_TYPE	TS
		7: REQ_CACHE_TYPE	TS
		8: REQ_SPLIT_TYPE	TS
		9: REQ_DEM_TYPE	TS
		10: REQ_ORD_TYPE	TS
		11: MEM_TYPE0	TS
		12: MEM_TYPE1	TS
		13: MEM_TYPE2	TS
	BSQ_cache_reference	ビット	
		0: RD_2ndL_HITS	TS
		1: RD_2ndL_HITE	TS
		2: RD_2ndL_HITM	TS
		3: RD_3rdL_HITS	TS
		4: RD_3rdL_HITE	TS
		5: RD_3rdL_HITM	TS
		6: WR_2ndL_HIT	TS
		7: WR_3rdL_HIT	TS
		8: RD_2ndL_MISS	TS
		9: RD_3rdL_MISS	TS
		10: WR_2ndL_MISS	TS
		11: WR_3rdL_MISS	TS
	memory_cancel	ビット	
		2: ST_RB_FULL	TS
		3: 64K_CONF	TS
	SSE_input_assist	ビット 15: ALL	TI
	64bit_MMX_uop	ビット 15: ALL	TI
	packed_DP_uop	ビット 15: ALL	TI

表 A-7. イベントマスクによる論理プロセッサの制限（続き）

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
	packed_SP_uop	ビット 15: ALL	TI
	scalar_DP_uop	ビット 15: ALL	TI
	scalar_SP_uop	ビット 15: ALL	TI
	128bit_MMX_uop	ビット 15: ALL	TI
	x87_FP_uop	ビット 15: ALL	TI
	x87_SIMD_moves_uop	ビット 3: ALLP0	TI
		4: ALLP2	TI
	FSB_data_activity	ビット 0: DRDY_DRV	TI
		1: DRDY_OWN	TI
		2: DRDY_OTHER	TI
		3: DBSY_DRV	TI
		4: DBSY_OWN	TI
		5: DBSY_OTHER	TI
	IOQ_allocation	ビット 0: ReqA0	TS
		1: ReqA1	TS
		2: ReqA2	TS
		3: ReqA3	TS
		4: ReqA4	TS
		5: ALL_READ	TS
		6: ALL_WRITE	TS
		7: MEM_UC	TS
		8: MEM_WC	TS
		9: MEM_WT	TS
		10: MEM_WP	TS
		11: MEM_WB	TS
		13: OWN	TS
		14: OTHER	TS
		15: PREFETCH	TS

表 A-7. イベントマスクによる論理プロセッサの制限（続き）

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
	IOQ_active_entries	ビット 0: ReqA0	TS
		1:ReqA1	TS
		2: ReqA2	TS
		3: ReqA3	TS
		4: ReqA4	TS
		5: ALL_READ	TS
		6: ALL_WRITE	TS
		7: MEM_UC	TS
		8: MEM_WC	TS
		9: MEM_WT	TS
		10: MEM_WP	TS
		11: MEM_WB	TS
		13: OWN	TS
		14: OTHER	TS
		15: PREFETCH	TS
	global_power_events	ビット 0: RUNNING	TS
	ITLB_reference	ビット 0: HIT	TS
		1: MISS	TS
		2: HIT_UC	TS
	MOB_load_replay	ビット 1: NO_STA	TS
		3: NO_STD	TS
		4: PARTIAL_DATA	TS
		5: UNALGN_ADDR	TS
	page_walk_type	ビット 0: DTMISS	TI
		1: ITMISS	TI
	uop_type	ビット 1: TAGLOADS	TS
		2: TAGSTORES	TS
	load_port_replay	ビット 1: SPLIT_LD	TS
	store_port_replay	ビット 1: SPLIT_ST	TS

表 A-7. イベントマスクによる論理プロセッサの制限（続き）

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
	memory_complete	ビット 0: LSC	TS
		1: SSC	TS
		2: USC	TS
		3: ULC	TS
	retired_mispred_branch_type	ビット 0: UNCONDITIONAL	TS
		1: CONDITIONAL	TS
		2: CALL	TS
		3: RETURN	TS
		4: INDIRECT	TS
	retired_branch_type	ビット 0: UNCONDITIONAL	TS
		1: CONDITIONAL	TS
		2: CALL	TS
		3: RETURN	TS
		4: INDIRECT	TS
	tc_ms_xfer	ビット 0: CISC	TS
	tc_misc	ビット 4: FLUSH	TS
	TC_deliver_mode	ビット 0: DD	TI
		1: DB	TI
		2: DI	TI
		3: BD	TI
		4: BB	TI
		5: BI	TI
		6: ID	TI
		7: IB	TI
	uop_queue_writes	ビット 0: FROM_TC_BUILD	TS
		1: FROM_TC_DELIVER	TS
		2: FROM_ROM	TS
	resource_stall	ビット 5:SBFULL	TS

表 A-7. イベントマスクによる論理プロセッサの制限（続き）

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
Non-Retirement	WC_Buffer	ビット 0: WCB_EVICTS	TI
		1: WCB_FULL_EVICT	TI
		2: WCB_HITM_EVICT	TI
At Retirement	instr_retired	ビット 0: NBOGUSNTAG	TS
		1: NBOGUSTAG	TS
		2: BOGUSNTAG	TS
		3: BOGUSTAG	TS
	machine_clear	ビット 0: CLEAR	TS
		2: MOCLEAR	TS
		6: SMCCLEAR	TS
	front_end_event	ビット 0: NBOGUS	TS
		1: BOGUS	TS
	replay_event	ビット 0: NBOGUS	TS
		1: BOGUS	TS
	execution_event	ビット 0: NONBOGUS0	TS
		1: NONBOGUS1	TS
		2: NONBOGUS2	TS
		3: NONBOGUS3	TS
		4: BOGUS0	TS
		5: BOGUS1	TS
		6: BOGUS2	TS
		7: BOGUS3	TS
	x87_assist	ビット 0: FPSU	TS
		1: FPSO	TS
		2: POAO	TS
		3: POAU	TS
		4: PREA	TS
	branch_retired	ビット 0: MMNP	TS
		1: MMNM	TS
		2: MMTP	TS
		3: MMTM	TS

表 A-7. イベントマスクによる論理プロセッサの制限（続き）

イベントのタイプ	イベント名	イベントマスク、ESCR[24:9]	TS または TI
	mispred_branch_retired	ビット 0: NBOGUS	TS
	uops_retired	ビット 0: NBOGUS	TS
		1: BOGUS	TS
	instr_completed	ビット 0: NBOGUS	TS
		1: BOGUS	TS

A.2. インテル® Pentium® M プロセッサの性能モニタリング・イベント

インテル® Pentium® M プロセッサの性能モニタリング・イベントは、P6 ファミリ・プロセッサ向けのモニタリング・イベントがベースになっている。この性能モニタリング・イベントはすべて、インテル Pentium M プロセッサのモデル固有のものであり、その他のプロセッサではこの形式で利用することができない。表 A-8. に、インテル Pentium M プロセッサで追加された性能モニタリング・イベントを示す。

表 A-8. インテル® Pentium® M プロセッサの性能モニタリング・イベント

イベント名	16 進値	説明
パワー管理		
EMON_EST_TRANS	58H	拡張版 Intel SpeedStep® テクノロジの遷移の回数 マスク = 00H – すべての遷移 マスク = 02H – 周波数の遷移のみ
EMON_THERMAL_TRIP	59H	サーマルトリップの持続期間 / 発生回数。サーマルトリップの回数をカウントするには、PerfEvtSel0/1 のビット 22 をセットしてエッジ検出をイネーブルにする必要がある
BPU		
BR_INST_EXEC	88H	実行された分岐命令（リタイアされている必要はない）
BR_MISSP_EXEC	89H	実行され、実行時に予測ミスがあった分岐命令
BR_BAC_MISSP_EXEC	8AH	実行され、フロントエンド（BAC）で予測ミスがあった分岐命令
BR_CND_EXEC	8BH	実行された条件付き分岐命令
BR_CND_MISSP_EXEC	8CH	実行され、予測ミスがあった条件付き分岐命令
BR_IND_EXEC	8DH	実行された間接分岐命令

表 A-8. インテル® Pentium® M プロセッサの性能モニタリング・イベント（続き）

イベント名	16 進値	説明
BR_IND_MISSP_EXEC	8EH	実行され、予測ミスがあった間接分岐命令
BR_RET_EXEC	8FH	実行されたリターン分岐命令
BR_RET_MISSP_EXEC	90H	実行され、実行時に予測ミスがあったリターン分岐命令
BR_RET_BAC_MISSP_EXEC	91H	実行され、フロントエンド（BAC）で予測ミスがあったリターン分岐命令
BR_CALL_EXEC	92H	実行された呼び出し命令
BR_CALL_MISSP_EXEC	93H	実行され、予測ミスがあった呼び出し命令
BR_IND_CALL_EXEC	94H	実行された間接呼び出し命令
デコーダ		
EMON_SIMD_INSTR_RETIRED	CEH	リタイアされた MMX® 命令の数
EMON_SYNCH_UOPS	D3H	同期化 μop
EMON_ESP_UOPS	D7H	μop の総数
EMON_FUSED_UOPS_RET	DAH	リタイアされた混合 μop の数 マスク = 0 – すべての混合 μop マスク = 1 – ロード + Op μop のみ マスク = 2 – std + sta μop のみ
EMON_UNFUSION	DBH	ROB での非フュージョン・イベントの数（混合 μop を除き FP で発生）
Prefetcher		
EMON_PREF_RQSTS_UP	F0H	上位方向に実行されたプリフェッチの数
EMON_PREF_RQSTS_DN	F8H	下位方向に実行されたプリフェッチの数

P6 ファミリ・プロセッサ向けの性能モニタリング・イベントのうち一部は、インテル Pentium M プロセッサ用に変更されている。表 A-9. に、インテル Pentium M プロセッサ用に変更され、P6 ファミリ・プロセッサ向けのものとは異なる性能モニタリング・イベントを示す。

表 A-9. インテル® Pentium® M プロセッサ用に変更された性能モニタリング・イベント

イベント名	16 進値	説明	
CPU_CLK_UNHALTED	79H	プロセッサが停止されておらず、サーマルトリップが発生していなかったサイクル数	
EMON_SSE_SSE2_INST_RETIRED	D8H	リタイアされた SSE 命令 マスク = 0 – SSE パックド単精度およびスカラ単精度 マスク = 1 – SSE スカラ単精度 マスク = 2 – SSE2 パックド倍精度 マスク = 3 – SSE2 スカラ倍精度	
EMON_SSE_SSE2_COMP_INST_RETIRED	D9H	リタイアされた SSE 演算命令 マスク = 0 – SSE パックド単精度 マスク = 1 – SSE スカラ単精度 マスク = 2 – SSE2 パックド倍精度 マスク = 3 – SSE2 スカラ倍精度	
L2_LD	29H	L2 データロード	マスク [0] = 1 – I 状態ラインをカウント マスク [1] = 1 – S 状態ラインをカウント マスク [2] = 1 – E 状態ラインをカウント マスク [3] = 1 – M 状態ラインをカウント マスク [5:4] 00H – ハードウェア・プリフェッチされたラインを除く 00H – ハードウェア・プリフェッチされたラインのみ 02H/03H – すべて（ハードウェア・プリフェッチされたラインと、ハードウェア・プリフェッチではないライン）
L2_LINES_IN	24H	割り当てられた L2 ライン	
L2_LINES_OUT	26H	送出された L2 ライン	
L2_M_LINES_OUT	27H	送出された LwM 状態ライン	

A.3. P6 ファミリ・プロセッサの性能モニタリング・イベント

表 A-10. に、P6 ファミリ・プロセッサの性能モニタリング・カウンタでカウントでき、RDPMC 命令で読み取れるイベントを一覧して示す。「ユニット」列は、イベントを生成するマイクロアーキテクチャまたはバスユニットを示している。「イベント番号」列は、イベントを識別する 16 進数を示している。「ニーモニック・イベント名」列は、イベントの名前を示している。「ユニットマスク」列は、必要なユニットマスク（必要な場合）を示している。「説明」列は、イベントについて説明している。「コメント」列は、イベントについての追加情報を示している。

P6 ファミリ・プロセッサの場合、これらの性能イベントはすべてモデルによって異なり、インテル® Pentium® 4 プロセッサやインテル® Pentium® プロセッサではこの形式は使用できない。イベントによっては（例えば、後世代の P6 ファミリ・プロセッサで追加されたイベントなど）、P6 ファミリ・プロセッサの特定のプロセッサでしか使用

できないものもある。表 A-10. に一覧されていないすべての性能イベント・エンコーディングは予約されており、それらを使用すると未定義のカウンタ動作が引き起こされる結果となる。

表にあるエントリに関連する注記については、表の最後を参照。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで カウントできるイベント

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
データ・キャッシュ・ユニット（DCU）	43H	DATA_MEM_REFS	00H	任意のメモリタイプからのすべてのロード。任意のメモリタイプへのすべてのストア。スプリットの各部分は別々にカウントされる。内部ロジックは、メモリのロードとストアだけでなく、内部の再試行もカウントする。 **注**：80 ビット浮動小数点数へのアクセスは、16 ビット指数のロードと 64 ビット仮数のロードに分解されるため、2 回カウントされる。メモリアクセスは、実際に実行された場合にのみカウントされる（例えば、以前のキャッシュ・ミスが同じアドレスに対して未解決になっているために、ロードが一度却下され、その後に実行された場合は、1 回だけカウントされる）。 I/O アクセスなどの非メモリアクセスは含まない。	
	45H	DCU_LINES_IN	00H	DCU に割り当てられている合計ライン	
	46H	DCU_M_LINES_IN	00H	DCU に割り当てられている M 状態ラインの数	
	47H	DCU_M_LINES_OUT	00H	DCU から立ち退かされた M 状態ラインの数。これにはスヌープ HITM、干渉、または置換による立ち退きが含まれる。	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	48H	DCU_MISS_OUTSTANDING	00H	DCU ミスが未解決である間の重み付けされたサイクル数。特定の時刻に未解決になっているキャッシュ・ミスの数だけインクリメントされる。 キャッシュ可能な読み取り要求だけがカウントされる。 キャッシュ不可能な要求は除外される。 所有権読み取り、ラインフィル、無効化、ストアはカウントされる。	L2 もミスしたアクセスは、2 サイクルでショートチェンジされる（つまり、N サイクルをカウントする場合は、N+2 サイクルである必要がある）。 同じキャッシュ・ラインへの後続のロードが行われた場合は、追加のカウントは発生しない。 カウント値は正確ではないが、それでも役立つ。
命令フェッチユニット（IFU）	80H	IFU_IFETCH	00H	命令フェッチの数、キャッシュ可能とキャッシュ不可能の両方とも	
	81H	IFU_IFETCH_MISS	00H	命令フェッチミスの数 IFU にヒットしない（つまり、メモリ要求を発生させる）すべての命令フェッチ。 UC アクセスを含む。	
	85H	ITLB_MISS	00H	ITLB ミスの数	
	86H	IFU_MEM_STALL	00H	命令フェッチが何らかの理由でストールしたサイクル数。 IFU キャッシュ・ミス、ITLB ミス、ITLB フォルト、その他の小さなフォルトを含む。	
	87H	ILD_STALL	00H	命令長デコーダがストールしたサイクル数	
L2 キャッシュ [1]	28H	L2_IFETCH	MESI 0FH	L2 命令フェッチの数 このイベントは、正常な命令フェッチが L2 に受け入れられたことを示す。 このカウントは、L2 キャッシュ可能な命令フェッチだけを含む。 UC 命令フェッチは含まない。 ITLB ミスアクセスは含まない。	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	29H	L2_LD	MESI 0FH	L2 データロードの数 このイベントは、正常な、ロックされていない、ロード・メモリ・アクセスが、L2 に受け入れられたことを示す。 このカウントは、L2 キャッシュ可能なメモリアクセスだけを含む。I/O アクセスなどの非メモリアクセスや、UC/WT メモリアクセスなどのメモリアクセスは含まない。 L2 キャッシュ可能な TLB ミス・メモリ・アクセスを含む。	
	2AH	L2_ST	MESI 0FH	L2 データストアの数 このイベントは、正常な、ロックされていない、ストア・メモリ・アクセスが、L2 に受け入れられたことを示す。 厳密には、このイベントは、DCU が所有権読み取り要求を L2 に送ったことを示す。 このカウントは、DCU から L2 に送られた、無効な要求から修正された要求までの要求を含む。 このカウントは、L2 キャッシュ可能なメモリアクセスだけを含む。I/O アクセスなどの非メモリアクセスや、UC/WT メモリアクセスなどのメモリアクセスは含まない。 TLB ミス・メモリ・アクセスを含む。	
	24H	L2_LINES_IN	00H	L2 に割り当てられたラインの数	
	26H	L2_LINES_OUT	00H	何らかの理由で L2 から削除されたラインの数	
	25H	L2_M_LINES_INM	00H	L2 に割り当てられた修正されたラインの数	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	27H	L2_M_LINES_OUTM	00H	何らかの理由で L2 から削除された修正されたラインの数	
	2EH	L2_RQSTS	MESI 0FH	L2 要求の総数	
	21H	L2_ADS	00H	L2 アドレス・ストローブの数	
	22H	L2_DBUS_BUSY	00H	データバスがビジーであったサイクル数	
	23H	L2_DBUS_BUSY_RD	00H	データを L2 からプロセッサに転送するためにデータバスがビジーであったサイクル数	
外部バスロジック（EBL）[2]	62H	BUS_DRDY_CLOCKS	00H（セルフ） 20H（任意）	DRDY# がアサートされている間のクロック数。 データ転送時の外部システム・データ・バスの利用率。	ユニットマスク = 00H プロセッサが DRDY# をドライブしているときのバスクロックをカウントする。 ユニットマスク = 20H 任意のエージェントが DRDY# をドライブしているときに、プロセッサ・クロックでカウントする。
	63H	BUS_LOCK_CLOCKS	00H（セルフ） 20H（任意）	外部システムバス上で LOCK# がアサートされている間のクロック数。[3]	常にプロセッサ・クロックでカウントする。
	60H	BUS_REQ_OUTSTANDING	00H（セルフ）	未解決のバス要求の数 このカウンタは、特定のサイクルで未解決になっているキャッシュ可能なバス読み取り要求の数だけインクリメントされる。	DCU フルライン・キャッシュ可能読み取りだけをカウントし、RFO、書き込み、命令フェッチ、またはその他のものはカウントしない。「バスの完了待ち」（受け取った最後のデータチャンク）をカウントする。
	65H	BUS_TRAN_BRD	00H（セルフ） 20H（任意）	バースト読み取りトランザクションの数	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	66H	BUS_TRAN_RFO	00H（セルフ） 20H（任意）	所有権トランザクションに対する読み取り完了数	
	67H	BUS_TRANS_WB	00H（セルフ） 20H（任意）	ライトバック・トランザクション完了数	
	68H	BUS_TRAN_IFETCH	00H（セルフ） 20H（任意）	命令フェッチ・トランザクション完了数	
	69H	BUS_TRAN_INVAL	00H（セルフ） 20H（任意）	無効化トランザクション完了数	
	6AH	BUS_TRAN_PWR	00H（セルフ） 20H（任意）	パーシャル・ライト・トランザクション完了数	
	6BH	BUS_TRANS_P	00H（セルフ） 20H（任意）	パーシャル・トランザクション完了数	
	6CH	BUS_TRANS_IO	00H（セルフ） 20H（任意）	I/O トランザクション完了数	
	6DH	BUS_TRAN_DEF	00H（セルフ） 20H（任意）	延期されたトランザクション完了数	
	6EH	BUS_TRAN_BURST	00H（セルフ） 20H（任意）	バースト・トランザクション完了数	
	70H	BUS_TRAN_ANY	00H（セルフ） 20H（任意）	すべてのトランザクション完了数 アドレスバス利用率は、最小アドレスバス占有率から計算できる。 特殊サイクルなどを含む。	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	6FH	BUS_TRAN_MEM	00H（セルフ） 20H（任意）	メモリ・トランザクション完了数	
	64H	BUS_DATA_RCV	00H（セルフ） 20H（任意）	このプロセッサがデータを受け取っている間に発生したバス・クロック・サイクル数	
	61H	BUS_BNR_DRV	00H（セルフ）	このプロセッサがBNR# ピンをドライブしている間に発生したバス・クロック・サイクル数	
	7AH	BUS_HIT_DRV	00H（セルフ）	このプロセッサがHIT# ピンをドライブしている間に発生したバス・クロック・サイクル数	スヌープストールによるサイクルを含む。 このイベントは正しくカウントされるが、BPMi ピンの機能は、PC ビット（PerfEvtSel0 および PerfEvtSel1 レジスタのビット 19）の設定によって異なる。 • コアクロックとバスクロックの比が 2:1 または 3:1 であり、PC ビットがセットされている場合は、BPMi ピンは、カウンタがオーバーフローしたときに 1 クロックの間アサートされる。 • PC ビットがクリアされている場合は、プロセッサはカウンタがオーバーフローしたときに BPMi ピンをトグルする。 • クロック比が 2:1 または 3:1 でない場合は、BPMi ピンはこれらの性能モニタリング・カウンタ・イベントでは機能しない。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	7BH	BUS_HITM_DRV	00H（セルフ）	このプロセッサがHITM# ピンをドライブしている間に発生したバス・クロック・サイクル数	スヌープストールによるサイクルを含む。 このイベントは正しくカウントされるが、BPMi ピンの機能は、PC ビット（PerfEvtSel0 および PerfEvtSel1 レジスタのビット 19）の設定によって異なる。 • コアクロックとバスクロックの比が 2:1 または 3:1 であり、PC ビットがセットされている場合は、BPMi ピンは、カウンタがオーバーフローしたときに 1 クロックの間アサートされる。 • PC ビットがクリアされている場合は、プロセッサはカウンタがオーバーフローしたときに BPMi ピンをトグルする。 • クロック比が 2:1 または 3:1 でない場合は、BPMi ピンはこれらの性能モニタリング・カウンタ・イベントでは機能しない。
	7EH	BUS_SNOOP_STALL	00H（セルフ）	バスがスヌープによってストールしていたクロックサイクル数	
浮動小数点ユニット	C1H	FLOPS	00H	リタイアされた浮動小数点演算の数 トラップまたはアシストを発生させる浮動小数点演算を除く。 アシストハンドラによって実行される浮動小数点演算を含む。 超越関数などの複素浮動小数点命令の内部サブ演算を含む。 浮動小数点ロードおよびストアを除く。	カウンタ 0 のみ

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	10H	FP_COMP_OPS_EXE	00H	実行された浮動小数点演算の数 FADD、FSUB、FCOM、FMUL、整数 MUL および IMUL、FDIV、FPREM、FSQRTS、整数 DIV、および IDIV の数。 この数は、サイクル数ではなく、演算の回数を含む。 このイベントは、超越関数フローの途中で使用される FADD と個別の FADD 命令を区別しない。	カウンタ 0 のみ
	11H	FP_ASSIST	00H	マイクロコードによって処理された浮動小数点例外ケースの数	カウンタ 1 のみ このイベントは、スペキュレーティブ・エグゼキューションによるカウントを含む。
	12H	MUL	00H	乗算の数 このカウントは、整数の乗算と浮動小数点の乗算を含む。見込み的に実行された演算を含む。	カウンタ 1 のみ
	13H	DIV	00H	除算の数 このカウントは、整数の除算と浮動小数点の除算を含む。見込み的に実行された演算を含む。	カウンタ 1 のみ
	14H	CYCLES_DIV_BUSY	00H	除算器がビジーになっており、新しい除算を受け付けない間に発生するサイクル数。 これは、整数の除算、浮動小数点の除算、FPREM や FPSQRT などの命令を含む。見込み的に実行された演算を含む。	カウンタ 0 のみ

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
メモリ・オーダリング	03H	LD_BLOCKS	00H	ストア・バッファ・ブロックのために遅延されたロード操作の数 これまでのストアのうち、アドレスが不明なもの、アドレスはわかっているがデータが不明なもの、ロードと一部重複するものによって発生したカウントを含む。	
	04H	SB_DRAINS	00H	ストア・バッファ・ドレイン・サイクルの数 ストアバッファが排出されるサイクルごとにインクリメントされる。 ストアバッファの排出は、CPUID などのシリアル化操作、XCHG などの同期化操作、割り込み確認、その他の条件（キャッシュのフラッシュなど）によって発生する。	
	05H	MISALIGN_MEM_REF	00H	アライメントの合っていないデータメモリ参照の数 プロセッサのロードまたはストア・パイプラインがアライメントの合っていない uop をディスパッチする間に発生する各サイクルにつき、1 ずつインクリメントされる。 データメモリ参照が前半または後半だけである場合や、ブロックされたり、却下されたり、ミスした場合も、カウントが実行される。 ここで、「アライメントの合っていない」は、64 ビット境界にまたがる意味である。	MISALIGN_MEM_REF は、アライメントの合っていないメモリ参照の真の数の近似値にすぎない。 返される値は、アライメントの合っていないメモリアクセスの数（問題の大きさ）にほぼ比例する。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	07H	EMON_KNI_PREF_DISPATCHED	00H 01H 02H 03H	ディスパッチされた、ストリーミング SIMD 拡張命令のプリフェッチ命令 / 順序設定の緩い命令の数（見込み的なプリフェッチもカウントに含まれる）。 0: プリフェッチ NTA 1: プリフェッチ T1 2: プリフェッチ T2 3: 順序設定の緩いストア	カウンタ 0 および 1。 インテル® Pentium® III プロセッサのみ。
	4BH	EMON_KNI_PREF_MISS	00H 01H 02H 03H	すべてのキャッシュをミスした、プリフェッチ命令 / 順序設定の緩い命令の数。 0: プリフェッチ NTA 1: プリフェッチ T1 2: プリフェッチ T2 3: 順序設定の緩いストア	カウンタ 0 および 1。 インテル Pentium III プロセッサのみ。
命令デコーディングとリタイア	C0H	INST_RETIRED	00H	リタイアされた命令の数	REP STOS フローの最後の反復の間およびその後にハードウェア割り込みを受け取ると、カウンタが 1 命令だけ実際より少なくカウントされる。HLT 命令を実行中に SMI を受け取ると、性能カウンタが RSM 命令をカウントしないため、実際より 1 命令だけ少なくカウントされる。
	C2H	UOPS_RETIRED	00H	リタイアされた μOP の数	
	D0H	INST_DECODER	00H	デコードされた命令の数	
	D8H	EMON_KNI_INST_RETIRED	00H 01H	リタイアされたストリーミング SIMD 拡張命令の数。 0: パックドおよびスケーラ 1: スケーラ	カウンタ 0 および 1。 インテル Pentium III プロセッサのみ。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	D9H	EMON_KNI_COMP_INST_RET	00H 01H	リタイアされたストリーミング SIMD 拡張命令計算命令の数。 0: パックドおよびスケーラ 1: スケーラ	カウンタ 0 および 1。 インテル Pentium III プロセッサのみ。
割り込み	C8H	HW_INT_RX	00H	受け取ったハードウェア割り込みの数	
	C6H	CYCLES_INT_MASKED	00H	割り込みがディスエーブルになっていたプロセッサ・サイクル数	
	C7H	CYCLES_INT_PENDING_AND_MASKED	00H	割り込みがディスエーブルになっており、割り込みがペンディングであったプロセッサ・サイクル数	
分岐	C4H	BR_INST_RETIRED	00H	リタイアされた分岐命令の数	
	C5H	BR_MISS_PRED_RETIRED	00H	間違って予測された分岐のなかで、リタイアされた分岐の数	
	C9H	BR_TAKEN_RETIRED	00H	実施される分岐のなかで、リタイアされた分岐の数	
	CAH	BR_MISS_PRED_TAKEN_RET	00H	実施される間違って予測された分岐のなかで、リタイアされた分岐の数	
	E0H	BR_INST_DECODED	00H	デコードされた分岐命令の数	
	E2H	BTB_MISSES	00H	BTB が予測を生成しなかった分岐の数。	
	E4H	BR_BOGUS	00H	偽の分岐の数	
	E6H	BACLEARS	00H	BACLEAR がアサートされた回数 これは、BTB が分岐予測を行わなかったために分岐デコーダによって実行された、静的な分岐予測の回数を示す。	

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
ストール	A2H	RESOURCE_ STALLS	00H	リソースに関連するストールが存在した各サイクルにつき、1 ずつインクリメントされる。 レジスタ・リネーミング・バッファ・エントリ、メモリ・バッファ・エントリを含む。 バスキューが一杯になっている、キャッシュ・ミスが多すぎるなどの原因によるストールは含まない。 このイベントは、リソースに関連するストール以外に、若干のイベントをカウントする。 分岐の予測ミスの回復中に発生したストールも含む。例えば、同期化操作によってストアバッファを排出している間、予測ミスした分岐のリタイアが遅延されたために、ストールが発生した場合など。	
	D2H	PARTIAL_RAT_ STALLS	00H	パーシャル・ストールのイベント数またはサイクル数。 これは、フラグ・パーシャル・ストールを含む。	
セグメント・レジスタ・ロード	06H	SEGMENT_ REG_LOADS	00H	セグメント・レジスタのロードの数	
クロック	79H	CPU_CLK_ UNHALTED	00H	プロセッサが停止されていなかったサイクル数	
MMX® 命令ユニット	B0H	MMX_INSTR_ EXEC	00H	実行された MMX 命令の数	インテル® Celeron® プロセッサ、インテル® Pentium® II プロセッサ、インテル Pentium II インテル® Xeon™ プロセッサでのみ使用できる。 レジスタからメモリへの MOVQ および MOVD ストアは含まない。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
	B1H	MMX_SAT_INSTR_EXEC	00H	実行された MMX 飽和命令の数	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
	B2H	MMX_UOPS_EXEC	0FH	実行された MMX μOPS の数	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
	B3H	MMX_INSTR_TYPE_EXEC	01H 02H 04H 08H 10H 20H	実行された MMX テクノロジ・パックド乗算命令 実行された MMX テクノロジ・パックド・シフト命令 実行された MMX テクノロジ・パック操作命令 実行された MMX テクノロジ・アンパック操作命令 実行された MMX テクノロジ・パックド論理命令 実行された MMX テクノロジ・パックド算術演算命令	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
	CCH	FP_MMX_TRANS	00H 01H	MMX 命令から浮動小数点命令への遷移 浮動小数点命令から MMX 命令への遷移	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
	CDH	MMX_ASSIST	00H	MMX 命令アシストの数（つまり、実行された EMMS 命令の数）	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
	CEH	MMX_INSTR_RET	00H	リタイアされた MMX 命令の数	インテル Pentium II プロセッサだけで使用できる。

表 A-10. P6 ファミリ・プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

ユニット	イベント番号	ニーモニック・イベント名	ユニットマスク	説明	コメント
セグメント・レジスタ・リネーミング	D4H	SEG_ RENAME_ STALLS		セグメント・レジスタ・リネーミング・ストールの数	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
			01H	セグメント・レジスタ ES	
			02H	セグメント・レジスタ DS	
			04H	セグメント・レジスタ FS	
			08H	セグメント・レジスタ GS	
			0FH	セグメント・レジスタ ES + DS + FS + GS	
	D5H	SEG_REG_ RENAMES		セグメント・レジスタ・リネームの数	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。
			01H	セグメント・レジスタ ES	
			02H	セグメント・レジスタ DS	
			04H	セグメント・レジスタ FS	
			08H	セグメント・レジスタ GS	
			0FH	セグメント・レジスタ ES + DS + FS + GS	
	D6H	RET_SEG_ RENAMES	00H	リタイアされたセグメント・レジスタ・リネーム・イベントの数	インテル Pentium II プロセッサおよびインテル Pentium III プロセッサでのみ使用できる。

注記 :

1. L2 キャッシュ・イベントは、注記されている場合には、PerfEvtSel0 レジスタと PerfEvtSel1 レジスタのユニットマスク（UMSK）フィールドを使用してさらに詳細に識別できる。ユニット・マスク・フィールドの下位 4 ビットは、L2 イベントと連係して使用され、関連するキャッシュ状態を示す。P6 ファミリ・プロセッサでは、"MESI" プロトコルを使用してキャッシュ状態を識別し、結果として、ユニット・マスク・フィールドの各ビットは、4 つの状態（UMSK[3] = M(8H) 状態、UMSK[2] = E(4H) 状態、UMSK[1] = S(2H) 状態、UMSK[0] = 1(1H) 状態）のうちの 1 つを表す。UMSK[3:0] = MES"(FH) を使用してすべての状態のデータを集める必要がある。適用可能なイベントで UMSK =0H である場合は、何もカウントされない。
2. すべての外部バスロジック（EBL）イベントは、注記されている場合を除いて、PerfEvtSel0 レジスタと PerfEvtSel1 レジスタのユニットマスク（UMSK）フィールドを使用してさらに詳細に識別することができる。UMSK フィールドのビット 5 は、EBL イベントと連係して使用され、プロセッサが、自己生成であるトランザクション（UMSK[5] = 0）またはバス上にある任意のプロセッサから生じるトランザクション（UMSK[5] = 1）のどちらをカウントする必要があるかを示す。
3. L2 キャッシュがロックされるため、カウントがゼロになる可能性がある。

A.4. インテル® Pentium® プロセッサの性能モニタリング・イベント

表 A-11. に、インテル® Pentium® プロセッサの性能モニタリング・カウンタでカウントすることができるイベントを一覧して示す。「イベント番号」列は、イベントを識別し、CESR MSR の ES0 または ES1（イベント選択）フィールドに入れる 16 進コードを示している。「ニーモニック・イベント名」列は、イベントの名前を示し、「説明」列と「コメント」列はイベントの詳細な説明を示している。大部分のイベントは、カウンタ 0 またはカウンタ 1 のいずれでもカウントできるが、（注記されているように）カウンタ 0 またはカウンタ 1 だけでしかカウントできないイベントもある。

注記

表で網掛けにされているイベントは、MMX® テクノロジ Pentium® プロセッサだけにインプリメントされている。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタでカウントできるイベント

イベント番号	ニーモニック・イベント名	説明	コメント
00H	DATA_READ	（内部データ・キャッシュのヒットとミスを組み合わせた）メモリデータ読み取りの数	スプリット・サイクル読み取りは個別にカウントされる。TLB ミス処理の一部であるデータメモリ読み取りは含まれない。これらのイベントは、最大 2 回 / クロック発生することがある。I/O は含まれない。
01H	DATA_WRITE	（内部データ・キャッシュのヒットとミスを組み合わせた）メモリデータ書き込みの数。I/O は含まれない。	スプリット・サイクル書き込みは個別にカウントされる。これらのイベントは、最大 2 回 / クロック発生することがある。I/O は含まれない。
0H2	DATA_TLB_MISS	データ・キャッシュのトランスレーション・ルックアサイド・バッファへのミスの数	
03H	DATA_READ_MISS	アクセスがキャッシュ可能かまたはキャッシュ不可能かに関係なく、内部データ・キャッシュをミスしたメモリ読み取りアクセスの数	バースト・ライン・フィルの最初の BRDY# が戻された後で、最後の（4 番目の）BRDY# が戻される前に、同じキャッシュ・ラインをさらに読み取っても、カウンタはさらにはインクリメントされない。TLB ミス処理の一部であるデータアクセスは含まれない。I/O 空間へのアクセスは含まれない。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
04H	DATA WRITE MISS	アクセスがキャッシュ可能かまたはキャッシュ不可能かに関係なく、内部データ・キャッシュをミスしたメモリ書き込みアクセスの数	TLB ミス処理の一部であるデータアクセスは含まれない。I/O 空間へのアクセスは含まれない。
05H	WRITE_HIT_TO_ M-_OR_E-STATE_ LINES	データ・キャッシュの排他的なラインまたは修正されたラインへの書き込みヒットの数	これらは、EWBE# が非アクティブである場合は停止される可能性のある書き込みである。これらのイベントは、最大 2 回 / クロック発生することがある。
06H	DATA_CACHE_ LINES_ WRITTEN_ BACK	原因に関係なくライトバックされる（すべての）ダーティラインの数	置換、内部スヌープ、および外部スヌープは、すべてライトバックを発生させる可能性があり、カウントされる。
07H	EXTERNAL_ SNOOPS	コード・キャッシュまたはデータ・キャッシュでヒットしたかあるいはどちらでもヒットしなかったかに関係なく、受け付けた外部スヌープの数	サンプリング・インターバル外の EADS# のアサーションはカウントされない。内部スヌープはカウントされない。
08H	EXTERNAL_DATA_ CACHE_SNOOP_ HITS	データ・キャッシュへの外部スヌープの数	データ・キャッシュ、データ・ライン・フィル・バッファ、またはライトバック・バッファの 1 つの有効なラインへのスヌープヒットは、すべてヒットとしてカウントされる。
09H	MEMORY ACCESSES IN BOTH PIPES	パイプラインの両方のパイプでペアにされるデータメモリ読み取りまたはデータメモリ書き込みの数	これらのアクセスは、キャッシュ・ミスやバンク衝突などのために、必ずしも並行して実行されない。
0AH	BANK CONFLICTS	実際のバンク衝突の数	
0BH	MISALIGNED DATA MEMORY OR I/O REFERENCES	アライメントの合っていないメモリまたは I/O の読み取りまたは書き込みの数	2 バイトまたは 4 バイトのアクセスは、4 バイト境界をまたがるとアライメントが合っていない。8 バイトのアクセスは、8 バイト境界をまたがるとアライメントが合っていない。10 バイトのアクセスは、それぞれ 8 バイトと 2 バイトの 2 つの別々のアクセスとして扱われる。
0CH	CODE READ	読み取りがキャッシュ可能かまたはキャッシュ不可能かに関係なく、命令読み取りの数	個別の 8 バイト・キャッシュ不可能命令読み取りはカウントされる。
0DH	CODE TLB MISS	読み取りがキャッシュ可能かまたはキャッシュ不可能かに関係なく、コード TLB をミスする命令読み取りの数	個別の 8 バイト・キャッシュ不可能命令読み取りはカウントされる。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
0EH	CODE CACHE MISS	読み取りがキャッシュ可能かまたはキャッシュ不可能かに関係なく、内部コード・キャッシュをミスする命令読み取りの数	個別の 8 バイト・キャッシュ不可能命令読み取りはカウントされる。
0FH	ANY SEGMENT REGISTER LOADED	LDTR、GDTR、IDTR、TR を含む実アドレスモードまたは保護モードでの任意のセグメント・レジスタへの書き込みの数	セグメント・ロードは、明示的セグメント・レジスタ・ロード命令、far 制御転送、およびタスクスイッチによって引き起こされる。特権レベル変更を引き起こす far 制御転送とタスクスイッチは、このイベントを 2 回通知する。割り込みと例外が far 制御転送を開始する可能性がある。
10H	予約済み		
11H	予約済み		
12H	分岐	条件分岐、ジャンプ、呼び出し、戻り、ソフトウェア割り込み、割り込み戻りを含む、実施される分岐と実施されない分岐の数	シリアル化命令、VERR 命令、VERW 命令、一部のセグメント・ディスクリプタ・ロード、(FLUSH# を含む) ハードウェア割り込み、トラップハンドラまたはフォルトハンドラを起動するプログラム例外も実施される分岐としてカウントされる。パイプは必ずしもフラッシュされない。実際に実行された分岐の数が測定され、予測された分岐の数ではない。
13H	BTB_HITS	発生した BTB ヒットの数	このヒットは、実際に実行された命令に対するものだけがカウントされる。
14H	TAKEN_BRANCH_ OR_BTB_HIT	発生した実施される分岐または BTB ヒットの数	このイベントタイプは、実施される分岐と BTB ヒットの論理和である。これは、BTB でヒットを起こさせる可能性のあるイベントを表している。特に、これは、BTB への空間の候補であるか、またはすでに BTB にある。
15H	PIPELINE FLUSHES	発生したパイプライン・フラッシュの数。パイプライン・フラッシュは、実施される分岐、誤った予測、例外、割り込み、一部のセグメント・ディスクリプタ・ロードでの BTB ミスによって発生する。	カウンタは、シリアル化命令（シリアル化命令はプリフェッチ・キューをフラッシュさせるが、パイプライン・フラッシュ・イベント・カウンタをトリガしない）とソフトウェア割り込み（ソフトウェア割り込みはパイプラインをフラッシュしない）に対してはインクリメントされない。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
16H	INSTRUCTIONS_ EXECUTED	（最大 2 回 / クロック）実行された命令の数	フォルトハンドラの起動は、命令と見なされる。すべてのハードウェア割り込み、ソフトウェア割り込み、例外もカウントされる。ループ基準が満たされるまでリピートループは同じ命令を複数回実行するのにもかかわらず、リピート・プリフィックス付きストリング命令があると、このカウンタは 1 回だけインクリメントされる。 このことは、すべてのリピート・ストリング命令プリフィックス（つまり、REP、REPE、REPZ、REPNE、および REPNZ）に適用される。また、プロセッサの HALT 状態が何サイクル続いたかに関係なく、このカウンタは、実行された HLT 命令ごとに 1 回だけインクリメントされる。
17H	INSTRUCTIONS_ EXECUTED_V PIPE	V パイプで実行された命令の数。これは、ペアにされた命令の数を示す。	このイベントは 16H イベントと同じであるが、V パイプで実際に実行された命令の数だけをカウントすることが異なる。
18H	BUS_CYCLE_ DURATION	バスサイクルが進行中である間のクロック数。このイベントは、バスの使用を測定する。	カウントは、HLDA、AHOLD、BOFF# クロックを含む。
19H	WRITE_BUFFER_ FULL_STALL_ DURATION	パイプラインがフル・ライト・バッファによってストールしている間のクロック数	フル・ライト・バッファは、S 状態ラインへのデータメモリ読み取りミス、データメモリ書き込みミス、データメモリ書き込みヒットをストールさせる。I/O アクセス時のストールは含まれない。
1AH	WAITING_FOR_ DATA_MEMORY_ READ_STALL_ DURATION	データメモリ読み取りを待っている間にパイプラインがストールしている間のクロック数	データ TLB ミス処理もカウントに含まれる。ラインがフィルされている間にバイパスされない読み取りの試みを含めて、データメモリ読み取りが進行中である間は、パイプラインはストールする。
1BH	STALL ON WRITE TO AN E- OR M-STATE LINE	E または M 状態ラインへの書き込み時のストールの数	
1CH	LOCKED BUS CYCLE	LOCK プリフィックス、LOCK 命令、ページテーブル更新、ディスクリプタ・テーブル更新の結果として発生するロックされたバスサイクルの数	ロックされた読み取り - 修正 - 書き込みの読み取り部分だけがカウントされる。スプリット・ロックト・サイクル（SCYC アクセス）は、2 つの別々のアクセスとしてカウントされる。BOFF# によって再スタートされたサイクルは、再度カウントされない。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
1DH	I/O READ OR WRITE CYCLE	I/O 空間へのバスサイクルの数	アライメントの合っていない I/O アクセスは、2 つのバスサイクルを生成する。BOFF# によって再スタートされたバスサイクルは、再度カウントされない。
1EH	NONCACHEABLE_ MEMORY_READS	キャッシュ不可能命令またはデータメモリ読み取りバスサイクルの数。カウントは、TLB ミスによって発生した読み取りサイクルを含むが、I/O 空間への読み取りサイクルを含まない。	BOFF# によって再スタートされたサイクルは、再度カウントされない。
1FH	PIPELINE_AGI_ STALLS	アドレス生成インターロック（AGI）ストールの数。U と V 両方のパイプラインで同じクロックに発生する AGI は、このイベントを 2 回通知する。	U または V パイプラインの実行ステージにある命令が U または V パイプラインの D2（アドレス生成）ステージにある命令のインデックスまたはベース・アドレス・レジスタに書き込んでいるときに、AGI が発生する。
20H	予約済み		
21H	予約済み		
22H	FLOPS	発生する浮動小数点演算の数	浮動小数点の加算、減算、乗算、除算、剰余、平方根の数がカウントされる。超越関数命令は、複数の加算と乗算で構成され、このイベントを複数回通知する。ゼロによる除算、負の平方根、特殊オペランド、またはスタック例外を生成する命令は、カウントされない。 その他のすべての浮動小数点例外を生成する命令は、カウントされる。x87 FPU を使用する整数乗算命令とその他の命令は、カウントされる。
23H	BREAKPOINT MATCH ON DR0 REGISTER	レジスタ DR0 ブレークポイントでの一致の数	ブレークポイントがイネーブルになっているかどうかに関係なく、カウンタはインクリメントされる。ただし、ブレークポイントかイネーブルになっていない場合は、コードブレークポイント一致は、V パイプで実行される命令に対してはチェックされず、このカウンタはインクリメントされない。（これらは、ブレークポイントがイネーブルになっていないときだけに U パイプで実行される命令でチェックされる。） これらのイベントは、BP[3:0] ピンでドライブされる信号に対応する。詳細については、第 15 章「デバッグと性能モニタリング」を参照。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
24H	BREAKPOINT MATCH ON DR1 REGISTER	レジスタ DR1 ブレークポイントでの一致の数	23H イベントのコメントを参照。
25H	BREAKPOINT MATCH ON DR2 REGISTER	レジスタ DR2 ブレークポイントでの一致の数	23H イベントのコメントを参照。
26H	BREAKPOINT MATCH ON DR3 REGISTER	レジスタ DR3 ブレークポイントでの一致の数	23H イベントのコメントを参照。
27H	HARDWARE INTERRUPTS	実施される INTR 割り込みと NMI 割り込みの数	
28H	DATA_READ_OR_WRITE	(内部データ・キャッシュのヒットとミスを組み合わせた）メモリデータ読み取りまたは書き込み、あるいはその両方の数	スプリット・サイクル読み取りとスプリット・サイクル書き込みは個別にカウントされる。TLB ミス処理の一部であるデータメモリ読み取りは含まれない。これらのイベントは、最大 2 回 / クロック発生することがある。I/O は含まれない。
29H	DATA_READ_MISS OR_WRITE MISS	アクセスがキャッシュ可能かまたはキャッシュ不可能かに関係なく、内部データ・キャッシュをミスするメモリ読み取りまたはメモリ書き込み、あるいはその両方の数	バースト・ライン・フィルの最初の BRDY# が戻された後で、最後の（4 番目の）BRDY# が戻される前に、同じキャッシュ・ラインをさらに読み取っても、カウンタはさらにはインクリメントされない。TLB ミス処理の一部であるデータアクセスは含まれない。I/O 空間へのアクセスは含まれない。
2AH	BUS_OWNERSHIP_LATENCY（カウンタ 0）	LRM バス所有権要求からバス所有権承認までの時間（つまり、早いほうの PBREQ(0)、PHITM#、または HITM# アサーションから PBGNT アサーションまでの時間）	カウンタ 0 とカウンタ 1 でカウントされる 2AH イベントの比率は、バス所有権衝突による平均ストール時間である。
2AH	BUS OWNERSHIP TRANSFERS（カウンタ 1）	バス所有権転送の数（つまり、PBREQ(0) アサーションの数）	カウンタ 0 とカウンタ 1 でカウントされる 2AH イベントの比率は、バス所有権衝突による平均ストール時間である。
2BH	MMX_INSTRUCTIONS_EXECUTED_U-PIPE（カウンタ 0）	U パイプで実行された MMX 命令の数	
2BH	MMX_INSTRUCTIONS_EXECUTED_V-PIPE（カウンタ 1）	V パイプで実行された MMX 命令の数	

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
2CH	CACHE_M-STATE_ LINE_SHARING （カウンタ 0）	プロセッサが他のプロセッサのメモリアクセスによって修正されたラインへのヒットを識別した回数（PHITM(0)）	システムの平均メモリ・レイテンシが既知である場合は、このイベントによって、ユーザは、PHITM(0) ペナルティでのライトバックと Hit Modified(1) ペナルティでのレイテンシをカウントできる。
2CH	CACHE_LINE_ SHARING （カウンタ 1）	L1 キャッシュの共有データラインの数（PHIT(0)）	
2DH	EMMS_ INSTRUCTIONS_ EXECUTED （カウンタ 0）	実行された EMMS 命令の数	
2DH	TRANSITIONS_ BETWEEN_MMX_ AND_FP_ INSTRUCTIONS （カウンタ 1）	MMX 命令と浮動小数点命令の間の遷移の数。偶数カウントは、プロセッサが MMX テクノロジ・レジスタの状態にあることを示し、奇数カウントは、FP レジスタの状態にあることを示す。	このイベントは、MMX 命令の後の最初の浮動小数点命令または浮動小数点命令の後の最初の MMX 命令をカウントする。このカウントを使用して、浮動小数点状態と MMX 状態の間の遷移のペナルティを見積もれる。
2EH	BUS_UTILIZATION_ DUE_TO_ PROCESSOR_ ACTIVITY （カウンタ 0）	プロセッサ独自の活動、つまりプロセッサによって生じたバス活動によってバスがビジーであったクロック数	
2EH	WRITES_TO_ NONCACHEABLE_ MEMORY （カウンタ 1）	キャッシュ不可能メモリへの書き込みアクセスの数	カウントには、TLB ミスによって発生した書き込みサイクルと I/O 書き込みサイクルが含まれる。
2FH	SATURATING_MMX_ INSTRUCTIONS_ EXECUTED （カウンタ 0）	実際に飽和したかどうかに関係なく、実行された飽和 MMX 命令の数	
2FH	SATURATIONS_ PERFORMED （カウンタ 1）	飽和算術演算を使用し、結果のうちの少なくとも 1 つが実際に飽和した MMX 命令の数	4 つのダブルワードを操作する MMX 命令が 4 つの結果のうちの 3 つで飽和した場合でも、カウンタは 1 だけインクリメントされる。
30H	NUMBER_OF_ CYCLES_NOT_IN_ HALT_STATE （カウンタ 0）	プロセッサが HLT 命令によるアイドル状態ではないサイクル数	このイベントによって、ユーザは、「ネット CPI」を計算できる。プロセッサが HLT 命令を実行している間もタイムスタンプ・カウンタは無効にならないことに注意する必要がある。このイベントはカウンタ制御 CC0、CC1 によって制御されるので、これを使用して、TSC が提供できない CPL=3 での CPI を計算できる。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
30H	DATA_CACHE_ TLB_MISS_STALL_ DURATION （カウンタ 1）	データ・キャッシュのトランスレーション・ルックアサイド・バッファ（TLB）ミスによってパイプラインがストールするクロック数	
31H	MMX_ INSTRUCTION_ DATA_READS （カウンタ 0）	MMX 命令のデータ読み取りの数	
31H	MMX_ INSTRUCTION_ DATA_READ_ MISSES （カウンタ 1）	MMX 命令のデータ読み取りミスの数	
32H	FLOATING_POINT_ STALLS_DURATION （カウンタ 0）	浮動小数点フリーズによってパイプがストールしたクロック数	
32H	TAKEN_BRANCHES （カウンタ 1）	実施される分岐の数	
33H	D1_STARVATION_ AND_FIFO_IS_ EMPTY （カウンタ 0）	FIFO バッファが空であるために D1 ステージが何の命令も発行することができない回数	命令が命令 FIFO バッファで使用できる場合は、D1 ステージは、0、1、または 2 命令 / クロックを発行できる。
33H	D1_STARVATION_ AND_ONLY_ONE_ INSTRUCTION_IN_ FIFO（カウンタ 1）	FIFO バッファに 1 つの命令しかレディでなかったので、D1 ステージが 1 つの命令だけを発行した回数	命令が命令 FIFO バッファで使用できる場合は、D1 ステージは、0、1、または 2 命令 / クロックを発行できる。前に定義されているイベント、実行された命令（16H）および V パイプで実行された命令（17H）と組み合わせると、このイベントによって、ユーザは、ペアリング規則によって 2 つの命令の発行が阻止された回数を計算できる。
34H	MMX_ INSTRUCTION_ DATA_WRITES （カウンタ 0）	MMX 命令によって発生したデータ書き込みの数	
34H	MMX_ INSTRUCTION_ DATA_WRITE_ MISSES （カウンタ 1）	MMX 命令によって発生したデータ書き込みミスの数	

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
35H	PIPELINE_ FLUSHES_DUE_ TO_WRONG_ BRANCH_ PREDICTIONS （カウンタ 0）	E ステージまたは WB ステージで解決された間違った分岐予測によるパイプライン・フラッシュの数	カウントには、パイプラインが正しく追っていなかった分岐によるパイプライン・フラッシュが含まれる。これは、分岐が BTB になかったケース、分岐が BTB にあったけれども誤って予測されたケース、および分岐が正しく予測されたけれども間違ったアドレスに予測されたケースを含む。分岐は、実行ステージ（E ステージ）またはライトバック・ステージ（WB ステージ）で解決される。後者の場合には、予測ミスのペナルティは、1 クロックだけ大きい。カウンタ 0 とカウンタ 1 での 35H イベント・カウントの差は、E ステージで解決された分岐の数である。
35H	PIPELINE_ FLUSHES_ DUE_TO_WRONG_ BRANCH_ PREDICTIONS_ RESOLVED_IN_ WB-STAGE （カウンタ 1）	WB ステージで解決された間違った分岐予測によるパイプライン・フラッシュの数	イベント 35H（カウンタ 0）の注記を参照。
36H	MISALIGNED_ DATA_MEMORY_ REFERENCE_ON_ MMX_ INSTRUCTIONS （カウンタ 0）	MMX 命令を実行するときのアライメントの合っていないデータメモリ参照の数	
36H	PIPELINE_ISTALL_ FOR_MMX_ INSTRUCTION_ DATA_MEMORY_ READS （カウンタ 1）	パイプライン・ストールが待ち形式の MMX 命令データメモリ読み取りによって発生した間のクロック数	
37H	MISPREDICTED_ OR_ UNPREDICTED_ RETURNS （カウンタ 1）	間違って予測されたかまたは全く予測されなかった戻りの数	カウントは、実行された戻りの合計数と正しく予測された戻りの数との差である。RET 命令だけがカウントされる（例えば、IRET 命令はカウントされない）。
37H	PREDICTED_ RETURNS （カウンタ 1）	（正しくまたは間違って予測されたかどうかに関係なく）予測された戻りの数	RET 命令だけがカウントされる（例えば、IRET 命令はカウントされない）。

表 A-11. インテル® Pentium® プロセッサの性能モニタリング・カウンタで
カウントできるイベント（続き）

イベント番号	ニーモニック・イベント名	説明	コメント
38H	MMX_MULTIPLY_UNIT_INTERLOCK（カウンタ 0）	前の MMX 乗算命令のデスティネーションがまだレディではなかったので、パイプがストールしたクロック数	ストールに別の原因がある場合は、カウンタはインクリメントされない。乗算インターロックの発生ごとに、このイベントは、（ストールした命令が乗算後の次のクロックである場合は）2 回カウントされるか、（ストールした命令が乗算の 2 クロック後にある場合は）1 回だけカウントされる。
38H	MOVD/MOVQ_STORE_STALL_DUE_TO_PREVIOUS_MMX_OPERATION（カウンタ 1）	デスティネーションがストア命令で使用されていた前の MMX 操作によって、MOVD/MOVQ 命令が D2 ステージでストールしたクロック数	
39H	RETURNS（カウンタ 0）	実行された戻りの数	RET 命令だけがカウントされ、IRET 命令はカウントされない。RET 命令で実施される例外または RET 命令の実行前に命令境界でプロセッサによって認識される割り込みによってもカウンタはインクリメントされる。
39H	予約済み		
3AH	BTB_FALSE_ENTRIES（カウンタ 0）	分岐ターゲット・バッファにある偽のエントリの数	偽のエントリは、間違った予測以外の予測ミスの原因になる。
3AH	BTB_MISS_PREDICTION_ON_NOT-TAKEN_BRANCH（カウンタ 1）	BTB が実施されない分岐を実施されるとして予測した回数	
3BH	FULL_WRITE_BUFFER_STALL_DURATION_WHILE_EXECUTING_MMX_INSTRUCTIONS（カウンタ 0）	MMX 命令を実行している間にフル・ライト・バッファによってパイプラインがストールした間のクロック数	
3BH	STALL_ON_MMX_INSTRUCTION_WRITE_TO E-_OR_M-STATE_LINE（カウンタ 1）	E または M 状態ラインに書き込んでいる MMX 命令でのストール中のクロック数	

B

モデル固有レジスタ（MSR）

付録 B
モデル固有レジスタ（MSR）

B

この付録では、表B-1.、表B-3.、表B-4.に、インテル® Pentium® 4プロセッサとインテル® Xeon™プロセッサ、P6ファミリ・プロセッサ、インテル® Pentium® プロセッサに用意されているMSRを一覧して示す。一覧されているすべてのMSRは、RDMSR命令を使用して読み取りができ、WRMSR命令を使用して書き込める。「レジスタアドレス」は、16進と10進の両方で示す。「レジスタ名」は、ニーモニック・レジスタ名である。「ビット説明」は、レジスタの個別ビットについて説明している。

表B-5.に、アーキテクチャ的なMSRを示す。

B.1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR

以下のMSRは、インテル® Pentium® 4プロセッサおよびインテル® Xeon™プロセッサ用に定義されたものである。

- 「IA32_」プリフィックスの付いた MSR は、「アーキテクチャ的」と指定されている。すなわち、これらのMSRの機能およびアドレスは、IA-32プロセッサの後続ファミリでも変わらない。
- 「MSR_」プリフィックスの付いた MSR は、アドレス機能の点ではモデル固有である。「モデル利用」列は、指定されたレジスタアドレスにおけるインテル Pentium 4プロセッサ・ファミリおよびインテルXeonプロセッサ・ファミリ内のモデル・エンコーディング値を示している。プロセッサのモデル・エンコーディング値は、CPUIDを利用して問い合わせることができる。『IA-32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、中巻A』の「CPUID—CPUID Identification」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3	共有	B.4. 節「インテル® Pentium® プロセッサの MSR」を参照のこと。
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3	共有	B.4. 節「インテル® Pentium® プロセッサの MSR」を参照のこと。
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3	共有	7.7.4. 項「Monitor/Mwait のアドレス範囲の決定」を参照のこと。
		15:0			**モニタ・フィルタ・ラインのサイズ。**（R/W）キャッシュ・ラインまたはチップセット・ライン・バッファのバイト数を指定する。値を 40H（デフォルト）に指定すると、サイズは 64 バイトになる。 このレジスタ・フィールドは、MONITOR 命令および MWAIT 命令におけるセマフォのスペースとアライメントのサイズを指定するのに使用される。 BIOS は、このフィールドとチップセット・ライン・バッファ・レジスタを読み出す。次に、2 つの値のうち大きいほうを使ってこのレジスタ・フィールドのプログラムを行う。
		63:16			**予約済み。**
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3	独自	**タイムスタンプ・カウンタ。** 15.7. 節「タイムスタンプ・カウンタ」を参照のこと。
		63:0			**タイムスタンプ・カウント値。**（R/W）現在のタイムスタンプ・カウント値を返す。全 64 ビットが読み取り可能だが、書き込みできるのは下位 32 ビットだけである。下位 32 ビットに書き込みを行なうときは、上位 32 ビットがクリアされる。
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3	共有	**プラットフォーム ID。**（R）オペレーティング・システムではこの MSR を使用して、プロセッサの「スロット」情報と、ロードするのに適切なマイクロコード・アップデートを判別できる。
		49:0			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名 フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		52:50			**プラットフォーム ID。**（R） 当該プロセッサ向けのプラットフォームに関する情報が入る。 52 51 50 0 0 0 プロセッサ・フラグ 0 0 0 1 プロセッサ・フラグ 1 0 1 0 プロセッサ・フラグ 2 0 1 1 プロセッサ・フラグ 3 1 0 0 プロセッサ・フラグ 4 1 0 1 プロセッサ・フラグ 5 1 1 0 プロセッサ・フラグ 6 1 1 1 プロセッサ・フラグ 7
		63:53			**予約済み。**
1BH	27	IA32_APIC_BASE	0, 1, 2, 3	独自	**APIC のロケーションおよびステータス。**（R/W） APIC に関するロケーションおよびステータスの情報が入る（8.4.4. 項「ローカル APIC のステータスと位置」を参照）。
		7:0			**予約済み。**
		8			**ブートストラップ・プロセッサ（BSP）。**プロセッサが BPS の場合にセットする。
		10:9			**予約済み。**
		11			**APIC グローバル・イネーブル。**イネーブルの場合はセットし、ディスエーブルの場合はクリアする。
		31:12			**APIC ベースアドレス。**xAPIC メモリマップのベースアドレス。
		63:32			**予約済み。**
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3	共有	**プロセッサ・ハード・パワー・オン設定。**（R/W） プロセッサの機能をイネーブルおよびディスエーブルにする。（R）現在のプロセッサの設定を示す。
		0			**出力トライ・ステート・イネーブル。**（R） SMI# のストラップによってセットされるときの、トライステート出力がイネーブル（1）なのか、ディスエーブル（0）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		1			**BIST の実行。**（R） INIT# のストラップによってセットされるときの、BIST の実行がイネーブル（1）なのか、ディスエーブル（0）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。
		2			**イン・オーダー・キューの深さ。**（R） A7# のストラップによってセットされるときの、システムバスのイン・オーダー・キューの深さが 1（1）なのか、最大 12（0）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。
		3			**MCERR# 観察ディスエーブル。**（R） A9# のストラップによって決められるときの、MCERR# 観察がイネーブル（0）なのか、ディスエーブル（1）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。
		4			**BINIT# 観察イネーブル。**（R） A10# のストラップによって決められるときの、BINIT# 観察がイネーブル（0）なのか、ディスエーブル（1）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。
		6:5			**APIC クラスタ ID。**（R） A12# および A11# のストラップによってセットされるときの論理 APIC クラスタ ID 値が入る。論理クラスタ ID 値は、RESET# のデアサート時にフィールドへ書き込まれる。アドレスバス信号がアサートされている場合、フィールドは 1 にセットされる。
		7			**バス・パーク・ディスエーブル。**（R） A15# のストラップによってセットされるときの、バスパークがイネーブル（0）なのか、ディスエーブル（1）なのかを示す。このビットの値は、RESET# のデアサート時に書き込まれる。アドレスバス信号がアサートされている場合、このビットは 1 にセットされる。
		11:8			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
		13:12			**エージェント ID。**（R） BR[3:0] のストラップによってセットされるときの論理エージェント ID 値が入る。論理 ID 値は、RESET# のデアサート時にフィールドへ書き込まれる。アドレスバス信号がアサートされている場合、フィールドは 1 にセットされる。
		63:14			**予約済み。**
2BH	43	MSR_EBC_SOFT_POWERON	0, 1, 2, 3	共有	**プロセッサ・ソフト・パワー・オン設定。**（R/W） プロセッサの機能をイネーブルおよびディスエーブルにする。
		0			**要求エンコード時 RCNT/SCNT イネーブル。**（R/W） 要求エンコード時の RCNT/SCNT のドライブを制御する。セットするとイネーブル（1）、クリアするとディスエーブルになる（0、デフォルト）。
		1			**データ・エラー・チェック・ディスエーブル。**（R/W） セットすると、システム・データ・バスのパリティチェックがディスエーブルになり、クリアするとイネーブルになる。
		2			**応答エラー・チェック・ディスエーブル。**（R/W） セットするとディスエーブルになり（デフォルト）、クリアするとイネーブルになる。
		3			**アドレス / 要求エラー・チェック・ディスエーブル。**（R/W） セットするとディスエーブルになり（デフォルト）、クリアするとイネーブルになる。
		4			**イニシエータ MCERR# ディスエーブル。**（R/W） セットすると、イニシエータ・バス要求に対する MCERR# のドライブがディスエーブルになり（デフォルト）、クリアするとイネーブルになる。
		5			**内部 MCERR# ディスエーブル。**（R/W） セットすると、イニシエータ内部エラー対する MCERR# のドライブがディスエーブルになり（デフォルト）、クリアするとイネーブルになる。
		6			**BINIT# ドライバ・ディスエーブル。**（R/W） セットすると、BINIT# ドライバがディスエーブルになり（デフォルト）、クリアするとイネーブルになる。
		63:7			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
2CH	44	MSR_EBC_FREQUENCY_ID	2, 3	共有	**プロセッサ周波数設定。** この MSR のビット・フィールドのレイアウトは、CPUID バージョン情報の MODEL 値によって異なる。以下のビット・フィールドのレイアウトは、MODEL エンコーディングが 2 に等しいかそれより大きいインテル® Pentium® 4 プロセッサとインテル® Xeon™ プロセッサに適用される。 （R）このフィールドは、現在のプロセッサ周波数の設定を示す。
		15:0			**予約済み。**
		18:16			**スケーラブル・バス速度。**（R/W） 所期のスケーラブル・バス速度を示す。 エンコーディング　スケーラブル・バス速度 000B　100 MHz 001B　133 MHz 010B　200 MHz 011B　166 MHz エンコーディングが 001B のときにシステムバス速度を使って計算を行う場合は、133.33MHz を利用する必要がある。 エンコーディングが 011B のときにシステムバス速度を使って計算を行う場合は、166.67MHz を利用する必要がある。 その他すべては予約済み。
		63:19			**予約済み。**
2CH	44	MSR_EBC_FREQUENCY_ID	0, 1	共有	**プロセッサ周波数設定。** この MSR のビット・フィールドのレイアウトは、CPUID バージョン情報の MODEL 値によって異なる。このビット・フィールドのレイアウトは、MODEL エンコーディングが 2 より小さいインテル Pentium 4 プロセッサとインテル Xeon プロセッサに適用される。 （R）現在のプロセッサ周波数の設定を示す。
		20:0			**予約済み。**
		23:21			**スケーラブル・バス速度。**（R/W） 所期のスケーラブル・バス速度を示す。 エンコーディング　スケーラブル・バス速度 000B　100 MHz その他すべては予約済み。
		63:24			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
79H	121	IA32_BIOS_UPDT_TRIG	0, 1, 2, 3	共有	**BIOS 更新トリガレジスタ。**（R/W） この MSR に対して WRMSR 命令を実行すると、マイクロコード・アップデートがプロセッサにロードされる（9.11.6. 項「マイクロコード・アップデート・ローダ」を参照）。
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3	独自	**BIOS 更新シグニチャ ID。**（R/W） EAX に 1 をセットして CPUID 命令を実行すると、マイクロコード更新シグニチャが返される。
		31:0			**予約済み。**
		63:32			**マイクロコード更新シグニチャ。**（R/W） このフィールドには、CPUID 命令を実行する前に、0 をプリロードしておくのが望ましい。CPUID 命令を実行してもフィールドが 0 のままの場合、ロードされたマイクロコード・アップデートはないことを示す。0 以外の値は、マイクロコード更新シグニチャを表す。
FEH	254	IA32_MTRRCAP	0, 1, 2, 3	独自	**MTRR 情報。** 10.11.1. 項「MTRR 機能の識別」を参照のこと。
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3	独自	**CPL 0 コード用の CS レジスタ・ターゲット。**（R/W） SYSENTER 命令および SYSEXIT 命令によって使用される（4.8.7. 項「SYSENTER 命令と SYSEXIT 命令によるシステム・プロシージャへの 高速呼び出しの実行」を参照）。
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3	独自	**CPL 0 スタック用のスタックポインタ。**（R/W） SYSENTER 命令および SYSEXIT 命令によって使用される（4.8.7. 項「SYSENTER 命令と SYSEXIT 命令によるシステム・プロシージャへの 高速呼び出しの実行」を参照）。
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3	独自	**CPL 0 コード・エントリ・ポイント。**（R/W） SYSENTER 命令および SYSEXIT 命令によって使用される（4.8.7. 項「SYSENTER 命令と SYSEXIT 命令によるシステム・プロシージャへの 高速呼び出しの実行」を参照）。
179H	377	IA32_MCG_CAP	0, 1, 2, 3	独自	**マシンチェック機能。**（R） プロセッサのマシン・チェック・アーキテクチャの機能を返す（14.3.1.1. 項「IA32_MCG_CAP MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照）。
17AH	378	IA32_STATUS	0, 1, 2, 3	独自	**マシン・チェック・ステータス。**（R） マシンチェック例外が生成された後のマシン・チェック・ステートを返す（14.3.1.3. 項「IA32_MCG_STATUS MSR」を参照）。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
17BH	379	IA32_CTL			**マシンチェック機能イネーブル。**(R/W) マシンチェック機能をイネーブルにする（14.3.1.4. 項「IA32_MCG_CTL MSR」を参照）。
180H	384	IA32_MCG_EAX	0, 1, 2, 3	独自	**マシンチェック EAX 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EAX レジスタの内容。**(R/W から 0) 最後にマシン・チェック・エラーが起きた時点での EAX レジスタのステートが入る。
		63:32			**予約済み。**
181H	385	IA32_MCG_EBX	0, 1, 2, 3	独自	**マシンチェック EBX 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EBX レジスタの内容。**(R/W から 0) 最後にマシン・チェック・エラーが起きた時点での EBX レジスタのステートが入る。
		63:32			**予約済み。**
182H	386	IA32_MCG_ECX	0, 1, 2, 3	独自	**マシンチェック ECX 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**ECX レジスタの内容。**(R/W から 0) 最後にマシン・チェック・エラーが起きた時点での ECX レジスタのステートが入る。
		63:32			**予約済み。**
183H	387	IA32_MCG_EDX	0, 1, 2, 3	独自	**マシンチェック EDX 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと
		31:0			**EDX レジスタの内容。**(R/W から 0) 最後にマシン・チェック・エラーが起きた時点での EDX レジスタのステートが入る。
		63:32			**予約済み。**
184H	388	IA32_MCG_ESI	0, 1, 2, 3	独自	**マシンチェック ESI 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**ESI レジスタの内容。**(R/W から 0) 最後にマシン・チェック・エラーが起きた時点での ESI レジスタのステートが入る。
		63:32			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
185H	389	IA32_MCG_EDI	0, 1, 2, 3	独自	**マシンチェック EDI 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EDI レジスタの内容。**（R/W から 0） 最後にマシン・チェック・エラーが起きた時点での EDI レジスタのステートが入る。
		63:32			**予約済み。**
186H	390	IA32_MCG_EBP	0, 1, 2, 3	独自	**マシンチェック EBP 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EBP レジスタの内容。**（R/W から 0） 最後にマシン・チェック・エラーが起きた時点での EBP レジスタのステートが入る。
		63:32			**予約済み。**
187H	391	IA32_MCG_ESP	0, 1, 2, 3	独自	**マシンチェック ESP 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**ESP レジスタの内容。**（R/W から 0） 最後にマシン・チェック・エラーが起きた時点での ESP レジスタのステートが入る。
		63:32			**予約済み。**
188H	392	IA32_MCG_EFLAGS	0, 1, 2, 3	独自	**マシンチェック EFLAGS 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EFLAGS レジスタの内容。**（R/W から 0） 最後にマシン・チェック・エラーが起きた時点での EFLAGS レジスタのステートが入る。
		63:32			**予約済み。**
189H	393	IA32_MCG_EIP	0, 1, 2, 3	独自	**マシンチェック EIP 保存ステート。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		31:0			**EIP レジスタの内容。**（R/W から 0） 最後にマシン・チェック・エラーが起きた時点での EIP レジスタのステートが入る。
		63:32			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
18AH	394	IA32_MCG_MISC	0, 1, 2, 3	独自	**各種マシンチェック。** 14.3.2.5. 項「IA32_MCG 拡張マシン・チェック・ステート MSR」を参照のこと。
		0			DS。 セットされている場合、通常の DS 操作時にページアシストまたはページフォルトが発生したことを示す。プロセッサはシャットダウンする。 このビットは、DS 処理コードのデバッグ用のために使用される。通常の操作では、ユーザ（BIOS またはオペレーティング・システム）がこのビットをクリアする必要がある。
		63:1			**予約済み。**
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3	独自	**温度モニタ制御。**（R/W） オンデマンド・クロック調整をイネーブルまたはディスエーブルに設定し、オンデマンド・クロック調整のデューティ・サイクルの選択を可能にする。 13.16.3. 項「ソフトウェア制御クロック調整」を参照。
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3	独自	**温度割り込み制御。**（R/W） プロセッサの温度センサおよび温度モニタで検出される温度変化に対する割り込みの生成をイネーブルおよびディスエーブルに設定する。 13.16.2. 項「温度モニタ」を参照。
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3	共有	**温度モニタ・ステータス。**（R/W） プロセッサの温度センサおよび自動温度モニタ機能に関するステータス情報が入る。 13.16.2. 項「温度モニタ」を参照。
19DH	413	IMSR_THERM2_CTL	3	共有	**温度モニタ 2 制御。**（R/W） 読み出しを行うと、最後に書き込まれたターゲット TM2 遷移の値を指定する。セットすると、TM2 遷移の次のターゲット値をセットする。
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3	共有	**各種プロセッサ機能のイネーブル。**（R/W） 各種のプロセッサ機能をイネーブルおよびディスエーブルに設定できる。
		0			**高速ストリング・イネーブル。** セットすると、インテル Pentium 4 プロセッサの高速ストリング機能がイネーブルになる（デフォルト）。クリアするとディスエーブルになる。
		1			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		2			**x87 FPU Fopcode 互換性モード・イネーブル。** セットすると、fopcode 互換性モードがイネーブルになる。クリアするとイネーブルになる（デフォルト）。 『IA-32 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』の第 8 章「fopcode 互換性モード」を参照のこと。
		3			**温度モニタ・イネーブル。** セットすると、プロセッサ内部の温度センサによるクロック調整制御がイネーブルになる。クリアすると、自動クロック調整がディスエーブルになる（デフォルト）。 13.16.2. 項「温度モニタ」を参照。
		4			**スプリット・ロック・ディスエーブル。** このデバッグ機能は、インテル Pentium 4 プロセッサに固有のものである。 セットすると、ビットはスプリット・ロック・サイクルの代わりに #AC 例外を生成する。このビットをセットするオペレーティング・システムでは、システム構造のアライメントを合わせることでスプリット・ロックを防止しなければならない。 ビットをクリアすると（デフォルト）、通常のスプリット・ロックがバスに発行される。
		5			**予約済み。**
		6			**L3 キャッシュ・ディスエーブル。**（R/W） セットすると、L3 キャッシュがディスエーブルになる。クリアすると、L3 キャッシュがイネーブルになる（デフォルト）。L3 キャッシュを持たないプロセッサでは、このフラグは予約済みになる。 このビットは、L3 キャッシュだけを制御する。また、制御レジスタ CR0 の CD フラグ、ページレベルのキャッシュ制御、MTRR によってキャッシュ全体がイネーブルになっている場合にのみ、このビットは有効である。 10.5.4. 項「L3 キャッシュの無効化と有効化」を参照。
		7			**性能モニタリング使用可能。**（R） セットされている場合、性能モニタリングはイネーブルである。クリアされている場合、ディスエーブルである。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		8			**ロック・イネーブル抑制。** セットすると、スプリット・ロック・アクセス中はバス上のロックのアサートが抑制される。クリアすると、ロックは抑制されない（デフォルト）。
		9			**プリフェッチ・キュー・ディスエーブル。** セットすると、プリフェッチ・キューがディスエーブルになる。クリアすると、プリフェッチ・キューがイネーブルになる（デフォルト）。
		10			**FERR# 割り込みレポート・イネーブル。**（R/W） セットすると、FERR# ピンによる割り込みレポートがイネーブルになる。クリアすると、この割り込みレポート機能がディスエーブルになる。 このフラグがセットされており、プロセッサがクロック停止状態である（STPCLK# がアサートされている）とき、FERR# ピンをアサートすると、割り込み（INIT#、BINIT#、INTR、NMI、SMI#、または RESET# など）が保留中になっており、プロセッサは通常の動作に戻ってその割り込みを処理しなければならないことを、プロセッサに通知する。 このフラグは、STPCLK# ピンがアサートされていないときの FERR# ピンの通常の動作（マスクされていない浮動小数点エラーを示す）には影響を与えない。
		11			**分岐トレース・ストレージ使用不可（BTS_UNAVILABLE）。**（R） セットされている場合、分岐トレース・ストレージ（BTS）がプロセッサでサポートされない。クリアされている場合、BTS がサポートされる。
		12			**プリサイス・イベント・ベース・サンプリング使用不可（PEBS_UNAVILABLE）。**（R） セットされている場合、プリサイス・イベント・ベース・サンプリング（PEBS）がプロセッサでサポートされない。クリアされている場合、PEBS がサポートされる。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		13	3		**TM2 イネーブル。**（R/W） このビットをセットした場合（1）、ダイ温度があらかじめ決められたしきい値に達したことを温度センサが示すと、温度モニタ 2 のメカニズムが使用される。TM2 は、MSR_THERM2_CTL ビット 15:0 に最後に書き込まれた値に応じて、バスとコアの比と、電圧を削減する。 このビットをクリアすると（0、デフォルト）、プロセッサが温度管理ステートに入っても、プロセッサは VID 信号や、バスとコアの比を変更しない。 **注**：EAX=1 を指定して CPUID を実行した後に TM2 機能フラグ（ECX[8]）が 1 にセットされていない場合、この機能はサポートされていないので、BIOS はこのビット位置の内容を変更してはならない。このビットと TM1 ビットが両方ともディスエーブル状態にセットされている場合、プロセッサは仕様外の動作をしている。
		17:14			**予約済み。**
		18	3		**イネーブル・モニタ FSM。**（R/W） 0 にセットすると、EAX=1 を指定して CPUID 命令を実行した際に返される MONITOR 機能フラグ ECX[3] がクリアされ、MONITOR 命令と MWAIT 命令がサポートされていないことを示す。 このビットが 0 のときにソフトウェアが MONITOR 命令または MWAIT 命令を実行しようとすると、不正命令例外が生成される。 1 にセットすると（デフォルト）、このビットは MONITOR 命令と MWAIT 命令がサポートされていることを示す。 **注**：SSE3 機能フラグ ECX[0] がセットされていない場合、オペレーティング・システムはこのビットを変更してはならない。BIOS は、このビットをデフォルトの状態で維持する必要がある。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
		19			**隣接キャッシュ・ライン・プリフェッチ・ディスエーブル。**（R/W） 1 にセットすると、プロセッサは、現在必要なデータが格納された 128 バイト・セクタのキャッシュ・ラインをフェッチする。0 にセットすると、プロセッサはセクタ内の両方のキャッシュ・ラインをフェッチする。 シングル・プロセッサ・プラットフォームでは、このビットをセットしてはならない。サーバ・プラットフォームは、検証やテストで判明したプラットフォームの性能に基づいて、このビットをセットまたはクリアする必要がある。 BIOS は、このビットの設定を制御するセットアップ・オプションを搭載できる。
		21:20			**予約済み。**
		22	3		**CPUID MAXVAL の制限。**（R/W） 1 にセットすると、EAX=0 の CPUID は、EAX[7:0] で最大値 3 を返す。0 にセットすると（デフォルト）、EAX=0 の CPUID は、サポートされている最大の標準機能に応じた値を返す。 **注** : 一部の古いオペレーティング・システムでは、3 より大きい MAXVAL を扱えない。そのような古いオペレーティング・システムがインストールされていることをユーザが指示できるように、BIOS にはセットアップ用の質問が必要である。このビットをセットする前に、BIOS は EAX=0 を指定して CPUID 命令を実行し、EAX[7:0] で返される最大値を調べなければならない。最大値が 3 よりも大きい場合は、このビットがサポートされている。それ以外の場合は、このビットはサポートされていないので、BIOS はこのビット位置の内容を変更してはならない。
		23			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
		24			**L1 データ・キャッシュ・コンテキスト・モード。**（R/W） 1 にセットすると、L1 データ・キャッシュが共有モードになる。0 にセットすると（デフォルト）、L1 データ・キャッシュが適応モードになる。 L1 が適応モードで動作し、CR3 が同一の場合、L1 内のデータは論理プロセッサ間で共有される。それ以外の場合は、L1 は共有されず、キャッシュの使用は競合的になる。 **注**：EAX=1 を指定して CPUID を実行した後に Context ID 機能フラグ（ECX[10]）が 0 にセットされた場合、モードを切り替える機能はサポートされていないので、BIOS は IA32_MISC_ENABLE[24] の内容を変更してはならない。
		63:25			**予約済み。**
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3	独自	**リニア IP からの最新例外レコード。**（R） 最後に生成された例外または最後に処理された割り込みの前にプロセッサが実行した最新の分岐命令を指すポインタが入る。 15.5.6. 項「最新例外レコード（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照。
		31:0			**リニア IP から**：最新分岐命令のリニアアドレス。
		63:32			**予約済み。**
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3	独自	**リニア IP への最新例外レコード。**（R） 最後に生成された例外または最後に処理された割り込みの前にプロセッサが実行した最新の分岐命令のターゲットを指すポインタが入る。 15.5.6. 項「最新例外レコード（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照。
		31:0			**リニア IP から**：最新分岐命令のターゲットのリニアアドレス。
		63:32			**予約済み。**
1D9H	473	IA32_DEBUGCTL	0, 1, 2, 3	独自	**デバッグ制御。**（R/W） いくつかあるデバッグ機能の使い方を制御する。 15.5.1. 項「MSR_DEBUGCTLA MSR（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
1DAH	474	MSR_LASTBRANCH_TOS	0, 1, 2, 3	独自	**最新分岐レコードスタック TOS。**（R） 最新分岐レコードスタックの先頭を指す（すなわち、最新分岐レコードを含む MSR のインデックスを指す）インデックス（0 ～ 3 または 0 ～ 15）が入る。 15.5.2. 項「LBR スタック（インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ）」とアドレス 1DBH ～ 1DEH、680H ～ 68FH を参照のこと。
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2, 3	独自	**最新分岐レコード 0。**（R/W） 最新分岐レコードスタック上の 4 つある最新分岐レコードレジスタの 1 つ。プロセッサが実施した最後の 4 つの分岐、例外、または割り込みの 1 つに対するソースおよびデスティネーション命令へのポインタが入る。 **注** : MSR_LASTBRANCH_0 から MSR_LASTBRANCH_3 は、1DBH ～ 1DEH では、ファミリ 0FH、モード 0H ～ 02H でのみ利用可能である。680H ～ 68FH および 6C0H ～ 6CFH では、MSR に置き換えられている。詳細については、15.5. 節を参照のこと。
1DCH	476	MSR_LASTBRANCH_1	0, 1, 2, 3	独自	**最新分岐レコード 1。** 1DBH の MSR_LASTBRANCH_0 MSR の説明を参照のこと。
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2, 3	独自	**最新分岐レコード 1。** 1DBH の MSR_LASTBRANCH_0 MSR の説明を参照のこと。
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2, 3	独自	**最新分岐レコード 1。** 1DBH の MSR_LASTBRANCH_0 MSR の説明を参照のこと。
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3	共有	**可変範囲ベース MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3	共有	**可変範囲ベース MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
203H	515	IA32_MTRR_PHYSMASK1	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
204H	516	IA32_MTRR_PHYSBASE2	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
205H	517	IA32_MTRR_PHYSMASK2	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
206H	518	IA32_MTRR_PHYSBASE3	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
207H	519	IA32_MTRR_PHYSMASK3	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
208H	520	IA32_MTRR_PHYSBASE4	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
209H	521	IA32_MTRR_PHYSMASK4	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20AH	522	IA32_MTRR_PHYSBASE5	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20BH	523	IA32_MTRR_PHYSMASK5	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20CH	524	IA32_MTRR_PHYSBASE6	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20DH	525	IA32_MTRR_PHYSMASK6	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20EH	526	IA32_MTRR_PHYSBASE7	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
20FH	527	IA32_MTRR_PHYSMASK7	0, 1, 2, 3	共有	**可変範囲マスク MTRR。** 10.11.2.3. 項「可変範囲 MTRR」を参照のこと。
250H	592	IA32_MTRR_FIX64K_00000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
258H	600	IA32_MTRR_FIX16K_80000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
259H	601	IA32_MTRR_FIX16K_A0000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
268H	616	IA32_MTRR_FIX4K_C0000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
269H	617	IA32_MTRR_FIX4K_C8000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26AH	618	IA32_MTRR_FIX4K_D0000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26BH	619	IA32_MTRR_FIX4K_D8000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26CH	620	IA32_MTRR_FIX4K_E0000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26DH	621	IA32_MTRR_FIX4K_E8000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26EH	622	IA32_MTRR_FIX4K_F0000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
26FH	623	IA32_MTRR_FIX4K_F8000	0, 1, 2, 3	共有	**固定範囲 MTRR。** 10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
277H	631	IA32_CR_PAT	0, 1, 2, 3	独自	**ページ属性テーブル。** この MSR の詳細については、10.11.2.2. 項「固定範囲 MTRR」を参照のこと。
2FFH	767	IA32_MTRR_DEF_TYPE	0, 1, 2, 3	共有	**デフォルト・メモリ・タイプ。**（R/W） MTRR によってマップされていない物理メモリ領域のメモリタイプをセットする。 10.11.2.1. 項「IA32_MTRR_DEF_TYPE MSR レジスタ」を参照のこと。
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
307H	775	MSR_MS_COUNTER 3	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
308H	776	MSR_FLAME_ COUNTER0	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
309H	777	MSR_FLAME_ COUNTER1	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
30AH	778	MSR_FLAME_ COUNTER2	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
30BH	779	MSR_FLAME_ COUNTER3	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
3OCH	780	MSR_IQ_COUNTER0	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
3ODH	781	MSR_IQ_COUNTER1	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
3OEH	782	MSR_IQ_COUNTER2	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
3OFH	783	MSR_IQ_COUNTER3	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3	共有	15.9.2. 項「性能カウンタ」を参照のこと。
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
364H	868	MSR_MS_CCCR0	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
365H	869	MSR_MS_CCCR1	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
366H	870	MSR_MS_CCCR2	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
367H	871	MSR_MS_CCCR3	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
368H	872	MSR_FLAME_ CCCR0	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタ アドレス		レジスタ名 フィールドおよび フラグ	モデル 利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
369H	873	MSR_FLAME_ CCCR1	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36AH	874	MSR_FLAME_ CCCR2	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36BH	875	MSR_FLAME_ CCCR3	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3	共有	15.9.3. 項「CCCR MSR」を参照のこと。
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A6H	934	MSR_FLAME_ ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A7H	935	MSR_FLAME_ ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	共有	15.9.1. 項「ESCR MSR」を参照のこと。 **注**：この MSR は、最近のプロセッサでは利用できない。プロセッサ・ファミリ 0FH、モデル 01H ～ 02H でのみ利用できる。
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	共有	15.9.1. 項「ESCR MSR」を参照のこと。 **注**：この MSR は、最近のプロセッサでは利用できない。プロセッサ・ファミリ 0FH、モデル 01H ～ 02H でのみ利用できる。
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3C9H	969	MSR_IX_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3FOH	1008	MSR_TC_PRECISE_EVENT	0, 1, 2, 3	共有	15.9.1. 項「ESCR MSR」を参照のこと。
3F1H	1009	IA32_PEBS_ENABLE	0, 1, 2, 3	共有	プリサイス・イベント・ベース・サンプリング（PEBS）。(R/W) プリサイス・イベント・サンプリングおよび再生タグ付けのイネーブルを制御する。
		12:0			表 A-6. を参照のこと。
		23:13			予約済み。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自[1]	ビット説明
16 進	10 進				
		24			UOP タグ。 セットすると、再生タグ付けがイネーブルになる。
		25			ENABLE_PEBS_MY_THR。（R/W） セットすると、ターゲット論理プロセッサに対する PEBS がイネーブルになる。クリアすると、PEBS がディスエーブルになる（デフォルト）。 ターゲット論理プロセッサについては、15.10.3. 項「IA32_PEBS_ENABLE MSR」を参照のこと。 ハイパー・スレッディング・テクノロジをサポートしない IA-32 プロセッサでは、このビットは ENABLE_PEBS と呼ばれる。
		26			ENABLE_PEBS_OTH_THR。（R/W） セットすると、ターゲット論理プロセッサに対する PEBS がイネーブルになる。クリアすると、PEBS がディスエーブルになる（デフォルト）。 ターゲット論理プロセッサについては、15.10.3. 項「IA32_PEBS_ENABLE MSR」を参照のこと。 ハイパー・スレッディング・テクノロジをサポートしない IA-32 プロセッサでは、このビットは予約済みになる。
		63:27			予約済み。
3F2H	1010	MSR_PEBS_MATRIX_VERT	0, 1, 2, 3	共有	表 A-6. を参照のこと。
400H	1024	IA32_MC0_CTL	0, 1, 2, 3	共有	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
401H	1025	IA32_MC0_STATUS	0, 1, 2, 3	共有	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
402H	1026	IA32_MC0_ADDR	0, 1, 2, 3	共有	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。 IA32_MC0_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC0_ADDR レジスタはサポートされていないか、アドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
403H	1027	IA32_MC0_MISC	0, 1, 2, 3	共有	14.3.2.4. 項「IA32_MCi_MISC MSR」を参照のこと。 IA32_MC0_STATUS レジスタの MISCV フラグがクリアされている場合は、IA32_MC0_MISC MSR はサポートされていないか、追加情報を格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
404H	1028	IA32_MC1_CTL	0, 1, 2, 3	共有	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3	共有	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3	共有	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。 IA32_MC1_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC1_ADDR レジスタはサポートされていないか、アドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
407H	1031	IA32_MC1_MISC		共有	14.3.2.4. 項「IA32_MCi_MISC MSR」を参照のこと。 IA32_MC1_STATUS レジスタの MISCV フラグがクリアされている場合は、IA32_MC1_MISC MSR はサポートされていないか、追加情報を格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
408H	1032	IA32_MC2_CTL	0, 1, 2, 3	共有	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3	共有	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
40AH	1034	IA32_MC2_ADDR			14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。 IA32_MC2_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC2_ADDR レジスタはサポートされていないか、アドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
40BH	1035	IA32_MC2_MISC			14.3.2.4. 項「IA32_MCi_MISC MSR」を参照のこと。 IA32_MC2_STATUS レジスタの MISCV フラグがクリアされている場合は、IA32_MC2_MISC MSR はサポートされていないか、追加情報を格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3	共有	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3	共有	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3	共有	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。 IA32_MC3_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC3_ADDR レジスタはサポートされていないか、アドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3	共有	14.3.2.4. 項「IA32_MCi_MISC MSR」を参照のこと。 IA32_MC3_STATUS レジスタの MISCV フラグがクリアされている場合は、IA32_MC3_MISC MSR はサポートされていないか、追加情報を格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
600H	1536	IA32_DS_AREA	0, 1, 2, 3	独自	**DS セーブ領域。**（R/W） BTS バッファおよび PEBS バッファの管理に使用される DS バッファ管理領域を指す（15.9.4. 項「デバッグストア（DS）機構」を参照）。
		31:0			**DS バッファ管理領域。** DS バッファ管理領域の先頭バイトのリニアアドレス。
		63:32			**予約済み。**

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
680H	1664	MSR_LASTBRANCH_0_FROM_LIP	3	独自	**最終分岐レコード 0。**（R/W） 最終分岐レコードスタック上の 16 ペアある最終分岐レコードレジスタの 1 つ（680H ～ 68FH）。スタックのこの部分には、プロセッサが実施した最後の 16 の分岐、例外、または割り込みの 1 つに対するソース命令へのポインタが入る。 **注** : 680H ～ 68FH、6C0H ～ 6CfH の MSR は、ファミリ 0FH、モデル 03H より前にリリースされたプロセッサでは利用できない。これらの MSR は、以前のリリースでは 1DBH ～ 1DEH に置かれ同じ機能を実行していた MSR に取って代わるものである。詳細については、15.5. 節を参照のこと。
681H	1665	MSR_LASTBRANCH_1_FROM_LIP	3	独自	**最終分岐レコード 1。** 680H の MSR_LASTBRANCH_0 の説明を参照。
682H	1666	MSR_LASTBRANCH_2_FROM_LIP	3	独自	**最終分岐レコード 2。** 680H の MSR_LASTBRANCH_0 の説明を参照。
683H	1667	MSR_LASTBRANCH_3_FROM_LIP	3	独自	**最終分岐レコード 3。** 680H の MSR_LASTBRANCH_0 の説明を参照。
684H	1668	MSR_LASTBRANCH_4_FROM_LIP	3	独自	**最終分岐レコード 4。** 680H の MSR_LASTBRANCH_0 の説明を参照。
685H	1669	MSR_LASTBRANCH_5_FROM_LIP	3	独自	**最終分岐レコード 5。** 680H の MSR_LASTBRANCH_0 の説明を参照。
686H	1670	MSR_LASTBRANCH_6_FROM_LIP	3	独自	**最終分岐レコード 6。** 680H の MSR_LASTBRANCH_0 の説明を参照。
687H	1671	MSR_LASTBRANCH_7_FROM_LIP	3	独自	**最終分岐レコード 7。** 680H の MSR_LASTBRANCH_0 の説明を参照。
688H	1672	MSR_LASTBRANCH_8_FROM_LIP	3	独自	**最終分岐レコード 8。** 680H の MSR_LASTBRANCH_0 の説明を参照。
689H	1673	MSR_LASTBRANCH_9_FROM_LIP	3	独自	**最終分岐レコード 9。** 680H の MSR_LASTBRANCH_0 の説明を参照。
68AH	1674	MSR_LASTBRANCH_10_FROM_LIP	3	独自	**最終分岐レコード 10。** 680H の MSR_LASTBRANCH_0 の説明を参照。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
68BH	1675	MSR_LASTBRANCH_11_FROM_LIP	3	独自	**最終分岐レコード 11。** 680H の MSR_LASTBRANCH_0 の説明を参照。
68CH	1676	MSR_LASTBRANCH_12_FROM_LIP	3	独自	**最終分岐レコード 12。** 680H の MSR_LASTBRANCH_0 の説明を参照。
68DH	1677	MSR_LASTBRANCH_13_FROM_LIP	3	独自	**最終分岐レコード 13。** 680H の MSR_LASTBRANCH_0 の説明を参照。
68EH	1678	MSR_LASTBRANCH_14_FROM_LIP	3	独自	**最終分岐レコード 14。** 680H の MSR_LASTBRANCH_0 の説明を参照。
68FH	1679	MSR_LASTBRANCH_15_FROM_LIP	3	独自	**最終分岐レコード 15。** 680H の MSR_LASTBRANCH_0 の説明を参照。
6C0H	1728	MSR_LASTBRANCH_0_TO_LIP	3	独自	**最終分岐レコード 0。**（R/W） 最終分岐レコードスタック上の 16 ペアある最終分岐レコードレジスタの 1 つ（6C0H ～ 6CFH）。スタックのこの部分には、プロセッサが実施した最後の 16 の分岐、例外、または割り込みの 1 つに対するデスティネーション命令へのポインタが入る。 詳細については、15.5. 節を参照のこと。
6C1H	1729	MSR_LASTBRANCH_1_TO_LIP	3	独自	**最終分岐レコード 1。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C2H	1730	MSR_LASTBRANCH_2_TO_LIP	3	独自	**最終分岐レコード 2。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C3H	1731	MSR_LASTBRANCH_3_TO_LIP	3	独自	**最終分岐レコード 3。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C4H	1732	MSR_LASTBRANCH_4_TO_LIP	3	独自	**最終分岐レコード 4。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C5H	1733	MSR_LASTBRANCH_5_TO_LIP	3	独自	**最終分岐レコード 5。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C6H	1734	MSR_LASTBRANCH_6_TO_LIP	3	独自	**最終分岐レコード 6。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C7H	1735	MSR_LASTBRANCH_7_TO_LIP	3	独自	**最終分岐レコード 7。** 6C0H の MSR_LASTBRANCH_0 の説明を参照。

表 B-1. インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサの MSR（続き）

レジスタアドレス		レジスタ名フィールドおよびフラグ	モデル利用	共有 / 独自 [1]	ビット説明
16 進	10 進				
6C8H	1736	MSR_LASTBRANCH_8_TO_LIP	3	独自	最終分岐レコード 8。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6C9H	1737	MSR_LASTBRANCH_9_TO_LIP	3	独自	最終分岐レコード 9。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CAH	1738	MSR_LASTBRANCH_10_TO_LIP	3	独自	最終分岐レコード 10。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CBH	1739	MSR_LASTBRANCH_11_TO_LIP	3	独自	最終分岐レコード 11。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CCH	1740	MSR_LASTBRANCH_12_TO_LIP	3	独自	最終分岐レコード 12。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CDH	1741	MSR_LASTBRANCH_13_TO_LIP	3	独自	最終分岐レコード 13。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CEH	1742	MSR_LASTBRANCH_14_TO_LIP	3	独自	最終分岐レコード 14。 6C0H の MSR_LASTBRANCH_0 の説明を参照。
6CFH	1743	MSR_LASTBRANCH_15_TO_LIP	3	独自	最終分岐レコード 15。 6C0H の MSR_LASTBRANCH_0 の説明を参照。

1　ハイパー・スレッディング（HT）テクノロジ対応のプロセッサでは、1 つの物理ユニットごとに 2 つの論理プロセッサが存在する。つまり MSR が「共有」の場合は、1 つの MSR が 2 つの論理プロセッサ間で共有される。MSR が「独自」の場合は、各論理プロセッサがそれぞれ独自の MSR を持つ。

B.2. インテル® Pentium® M プロセッサの MSR

インテル® Pentium® M プロセッサのモデル固有レジスタ（MSR）は、B.3. 節で説明する P6 ファミリ・プロセッサの MSR に類似している。以下の表では、新しい MSR と、インテル Pentium M プロセッサで動作が変更された MSR について説明する。

表 B-2. インテル® Pentium® M プロセッサの MSR

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
0H	0	P5_MC_ADDR	付録 B.4.「インテル® Pentium® プロセッサの MSR」を参照のこと。
1H	1	P5_MC_TYPE	付録 B.4.「インテル® Pentium® プロセッサの MSR」を参照のこと。
10H	16	IA32_TIME_STAMP_ COUNTER	15.6. 節「最新の分岐、割り込み、例外の記録（P6 ファミリ・プロセッサ）」を参照のこと。
17H	23	IA32_PLATFORM_ID	**プラットフォーム ID。**（R）オペレーティング・システムではこの MSR を使用して、プロセッサの「スロット」情報と、ロードするのに適切なマイクロコード・アップデートを判別できる。
		49:0	**予約済み。**
		52:50	**プラットフォーム ID。**（R）当該プロセッサ向けのプラットフォームに関する情報が入る。 52 51 50 0 0 0 プロセッサ・フラグ 0 0 0 1 プロセッサ・フラグ 1 0 1 0 プロセッサ・フラグ 2 0 1 1 プロセッサ・フラグ 3 1 0 0 プロセッサ・フラグ 4 1 0 1 プロセッサ・フラグ 5 1 1 0 プロセッサ・フラグ 6 1 1 1 プロセッサ・フラグ 7
		63:53	**予約済み。**
2AH	42	MSR_EBL_CR_ POWERON	**プロセッサ・ハード・パワー・オン設定。**（R/W）プロセッサの機能をイネーブルおよびディスエーブルにする。（R）現在のプロセッサの設定を示す。
		0	**予約済み。**
		1	**データ・エラー・チェック・イネーブル。**（R/W） 1 = イネーブル 0 = ディスエーブル **注 :** インテル® Pentium® M プロセッサでは常に 0。
		2	**応答エラー・チェック・イネーブル。**（R/W） FRCERR 観察イネーブル 1 = イネーブル 0 = ディスエーブル **注 :** インテル Pentium M プロセッサでは常に 0。

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		3	AERR# ドライブ・イネーブル。(R/W) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		4	イニシエータ・バス要求に対する BERR# イネーブル。(R/W) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		5	予約済み。
		6	イニシエータ内部エラーに対する BERR# イネーブル。(R/W) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		7	BINIT# ドライバ・イネーブル。(R/W) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		8	出力トライ・ステート・イネーブル。(R/O) 1 = イネーブル 0 = ディスエーブル
		9	BIST の実行。(R/O) 1 = イネーブル 0 = ディスエーブル
		10	AERR# 観察イネーブル。(R/O) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		11	予約済み。
		12	BINIT# 観察イネーブル。(R/O) 1 = イネーブル 0 = ディスエーブル 注 : インテル Pentium M プロセッサでは常に 0。
		13	イン・オーダー・キュー深さ。(R/O) 1 = 1 0 = 8
		14	1M バイト・パワー・オン・リセット・ベクタ。(R/O) 1 = 1M バイト 0 = 4G バイト 注 : インテル Pentium M プロセッサでは常に 0。

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		15	予約済み。
		17:16	APIC クラスタ ID。（R/O） 注 : インテル Pentium M プロセッサでは常に 00B。
		18	システムバス周波数。（R/O） 0= 00MHz 1= 予約済み 注 : インテル Pentium M プロセッサでは常に 0。
		19	予約済み。
		21:20	対称型アービトレーション ID。（R/O） 注 : インテル Pentium M プロセッサでは常に 00B。
		26:22	クロック周波数比。（R/O）
		63:27	予約済み。
119h	281	MSR_BBL_CR_CTL	
		63:0	予約済み。
11Eh	281	MSR_BBL_CR_CTL3	
		0	L2 ハードウェア・イネーブル。（RO） 1 = L2 がハードウェア・イネーブルの場合 0 = L2 がハードウェア・ディスエーブルの場合に通知
		4:1	予約済み。
		5	ECC チェック・イネーブル。（RO） このビットは、キャッシュ・データ・バスでの ECC チェックをイネーブルにする。 ECC は常に、書き込みサイクルで生成される。 0 = ディスエーブル（デフォルト） 1 = イネーブル 注 : インテル Pentium M プロセッサの場合、キャッシュ・データ・バスでの ECC チェックは常にイネーブルである。
		7:6	予約済み。
		8	L2 イネーブル。（R/W） 1 = L2 キャッシュが初期化済み 0 = ディスエーブル（デフォルト） 注 : このビットがセットされるまで、プロセッサは WBINVD 命令や FLUSH# 入力のアサーションには応答しない。
		19:9	予約済み。

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		22:20	**L2 物理アドレス範囲サポート。**（RO） このフィールドは、キャッシュ・コントローラのアドレス機能が以下のように定義されていることを示す。 111 = 64G バイト（A35-A3） 110 = 32G バイト（A34-A3） 101 = 16G バイト（A33-A3） 100 = 8G バイト（A32-A3） 011 = 4G バイト（A31-A3） 010 = 2G バイト（A30-A3） 001 = 1G バイト（A29-A3） 000 = 512M バイト（A28-A3） インテル Pentium M プロセッサは、4G バイトの物理アドレス範囲をサポートする。
		23	**L2 不在。**（RO） 0 = L2 が存在 1 = L2 が不在
		63:24	**予約済み。**
179H	377	IA32_MCG_CAP	
		7:0	**カウント。**（RO） プロセッサで利用可能なハードウェア・ユニット・エラー・レポーティング・バンクの数を示す。
		8	**IA32_MCG_CTL 存在。**（RO） 1 = MSR 17BH の MSR_MCG_CTL レジスタをプロセッサが実装していることを示す。 0 = サポートされていない。
		63:9	**予約済み。**
17AH	378	IA32_MCG_STATUS	
		0	**RIPV。**セットすると、このビットは、（マシンチェックの生成時に）スタックにプッシュされた命令ポインタによってアドレス指定された命令がプログラムの再起動に利用できることを示す。このビットをクリアすると、プログラムの確実な再起動は不可能になる。
		1	**EIPV。**セットすると、このビットは、（マシンチェックの生成時に）スタックにプッシュされた命令ポインタによってアドレス指定された命令がエラーに直接関連付けられていることを示す。
		2	**MCIP。**セットすると、このビットは、マシンチェックが生成されたことを示す。このビットがまだセットされている時点で第 2 のマシンチェックが検出された場合、プロセッサはシャットダウン・ステートに入る。ソフトウェアは、マシンチェック例外を処理した後にこのビットを 0 に書き込む必要がある。
		63:3	**予約済み。**

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
198H	408	IA32_PERF_STAUS	
		15:0	現在の性能ステート値。
		63:16	**予約済み。**
199H	409	IA32_PERF_CTL	
		15:0	目標の性能ステート値。
		63:16	**予約済み。**
19AH	410	IA32_CLOCK_ MODULATION	**クロック調整。**（R/W）オンデマンド・クロック調整をイネーブルまたはディスエーブルに設定し、オンデマンド・クロック調整のデューティ・サイクルの選択を可能にする。13.16.3. 項「ソフトウェア制御クロック調整」を参照のこと。 **注 :** IA32_CLOCK_MODULATION MSR の元の名称は IA32_THERM_CONTROL MSR である。
19BH	411	IA32_THERM_ INTERRUPT	**温度割り込み制御。**（R/W）プロセッサの温度センサおよび温度モニタで検出される温度変化に対する割り込みの生成をイネーブルおよびディスエーブルに設定する。 13.16.2. 項「温度モニタ」を参照。
19CH	412	IA32_THERM_ STATUS	**温度モニタ・ステータス。**（R/W）温度モニタ・ステータス。（R/W）プロセッサの温度センサおよび自動温度モニタ機能に関するステータス情報が入る。 13.16.2. 項「温度モニタ」を参照。
19DH	413	MSR_THERM2_CTL	
		15:0	**予約済み。**
		16	TM_SELECT。（R/W）自動温度モニタのモード。 0 = 温度モニタ 1（温度に反応して開始されるクロック停止デューティ・サイクルのオンダイ調整） 1 = 温度モニタ 2（温度に反応して開始される周波数遷移） IA32_MISC_ENABLE レジスタのビット 3 をクリアすると、TM_SELECT は無効になる。TM1 も TM2 もイネーブルにならない。
		63:16	**予約済み。**
1A0	416	IA32_MISC_ENABLE	**各種プロセッサ機能のイネーブル。**（R/W） 各種のプロセッサ機能をイネーブルおよびディスエーブルに設定できる。
		2:0	**予約済み。**

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		3	**自動温度制御回路イネーブル。**（R/W） 1 = このビットをセットすると、インテル温度モニタ機能の温度制御回路（TCC）部分がイネーブルになる。これにより、プロセッサの温度センサの処理に基づいてプロセッサ・クロックが自動調整される。 0 = ディスエーブル（デフォルト）。 自動温度制御回路のイネーブル・ビットは、プロセッサが最大動作温度を超えそうであるとプロセッサ内部の温度センサが判断した際に TCC がアクティブになるかどうかを決定する。TCC がアクティブになり TM1 がイネーブルになると、プロセッサ・クロックは強制的に 50% のデューティ・サイクルに移行される。BIOS は、この機能をイネーブルにする必要がある。 このビットと、オンデマンド温度制御回路イネーブル・ビットを混同してはならない。
		6:4	**予約済み。**
		7	**性能モニタリング使用可能。**（R） 1 = 性能モニタリングがイネーブル 0 = 性能モニタリングがディスエーブル
		9:8	**予約済み。**
		10	**FERR# 多重化イネーブル。**（R/W） 1 = プロセッサによってアサートされた FERR# であり、プロセッサ内で保留中のブレークイベントを示す。 0 = 互換性のある FERR# 信号動作を示す。 **注 :** XAPIC 割り込みモデルの利用をサポートするには、このビットを 1 にセットする必要がある。
		11	**分岐トレース・ストレージ使用可能。**（RO） 1 = プロセッサは分岐トレース・ストレージ（BTS）をサポートしていない。 0 = BTS がサポートされている。
		12	**プリサイズ・イベント・ベース・サンプリング使用不可。**（RO） 1 = プロセッサはプリサイズ・イベント・ベース・サンプリング（PEBS）をサポートしていない。 0 = PEBS がサポートされている。 **注 :** インテル Pentium M プロセッサは PEBS をサポートしていない。
		15:13	**予約済み。**
		16	**拡張版 Intel SpeedStep® テクノロジ・イネーブル。**（R/W） 1 = 拡張版 Intel SpeedStep テクノロジがイネーブル **注 :** インテル Pentium M プロセッサでは、このビットを読み取り専用に設定できる。
		63:17	**予約済み。**

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
2FFH	767	IA32_MTRR_DEF_TYPE	**デフォルト・メモリ・タイプ。**（R/W）TRR によってマップされていない物理メモリ領域のメモリタイプをセットする。10.11.2.1. 項「IA32_MTRR_DEF_TYPE MSR レジスタ」を参照のこと。
400	1024	IA32_MC0_CTL	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
401	1025	IA32_MC0_STATUS	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
402	1026	IA32_MC0_ADDR	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。IA32_MC0_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC0_ADDR レジスタはサポートされていないかアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
404	1028	IA32_MC1_CTL	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
405	1029	IA32_MC1_STATUS	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
406	1030	IA32_MC1_ADDR	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。IA32_MC1_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC1_ADDR レジスタはサポートされていないかアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
408	1032	IA32_MC2_CTL	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
409	1033	IA32_MC2_STATUS	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
40A	1034	IA32_MC2_ADDR	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。IA32_MC2_STATUS レジスタの ADDRV フラグがクリアされている場合は、IA32_MC2_ADDR レジスタはサポートされていないかアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
40C	1036	MSR_MC4_CTL	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
40D	1037	MSR_MC4_STATUS	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。
40E	1038	MSR_MC4_ADDR	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。MSR_MC4_STATUS レジスタの ADDRV フラグがクリアされている場合は、MSR_MC4__ADDR レジスタはサポートされていないかアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。
410	1040	MSR_MC3_CTL	14.3.2.1. 項「IA32_MCi_CTL MSR」を参照のこと。
411	1041	MSR_MC3_STATUS	14.3.2.2. 項「IA32_MCi_STATUS MSR」を参照のこと。

表 B-2. インテル® Pentium® M プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
412	1042	MSR_MC3_ADDR	14.3.2.3. 項「IA32_MCi_ADDR MSR」を参照のこと。MSR_MC3_STATUS レジスタの ADDRV フラグがクリアされている場合は、MSR_MC3_ADDR レジスタはサポートされていないかアドレスを格納していない。この MSR がプロセッサ内でサポートされていない場合、この MSR の読み出しや書き込みを行うと、一般保護例外が発生する。

B.3. P6 ファミリ・プロセッサの MSR

以下の MSR は、P6 ファミリ・プロセッサ用に定義されたものである。表内の共用される MSR は、インテル® Pentium® II プロセッサおよびインテル® Pentium® III プロセッサでのみ使用できる。インテル® Pentium® 4 プロセッサ以降、このリスト内の MSR のいくつかは「アーキテクチャ的」と規定され、名前も変更されている。アーキテクチャ的な MSR のリストについては、表 B-5. を参照のこと。

表 B-3. P6 ファミリ・プロセッサの MSR

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
0H	0	P5_MC_ADDR	付録 B.4.「インテル® Pentium® プロセッサの MSR」を参照のこと。
1H	1	P5_MC_TYPE	付録 B.4.「インテル® Pentium® プロセッサの MSR」を参照のこと。
10H	16	TSC	15.7. 節「タイムスタンプ・カウンタ」を参照のこと。
17H	23	IA32_PLATFORM_ID	プラットフォーム ID。（R）オペレーティング・システムではこの MSR を使用して、プロセッサの「スロット」情報と、ロードするのに適切なマイクロコード・アップデートを判別できる。
		49:0	予約済み。
		52:50	プラットフォーム ID。（R）当該プロセッサ向けのプラットフォームに関する情報が入る。 52 51 50 0 0 0　プロセッサ・フラグ 0 0 0 1　プロセッサ・フラグ 1 0 1 0　プロセッサ・フラグ 2 0 1 1　プロセッサ・フラグ 3 1 0 0　プロセッサ・フラグ 4 1 0 1　プロセッサ・フラグ 5 1 1 0　プロセッサ・フラグ 6 1 1 1　プロセッサ・フラグ 7
		56:53	L2 キャッシュ・レイテンシ読み取り
		59:57	予約済み

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		60	クロック周波数比読み取り
		63:61	予約済み
1BH	27	APIC_BASE	8.4.4. 項「ローカル APIC のステータスと位置」を参照。
		7:0	予約済み
		8	ブート・ストラップ・プロセッサ・インジケータ・ビット、BSP=1
		10:9	予約済み
		11	APIC グローバル・イネーブル・ビット - リセットされるまで恒久的 イネーブル = 1、ディスエーブル = 0
		31:12	APIC ベースアドレス
		63:32	予約済み
2AH	42	EBL_CR_POWERON	プロセッサ・ハード・パワー・オン設定。（R/W）プロセッサの機能をイネーブルおよびディスエーブルにする。（R）現在のプロセッサの設定を示す。
		0	予約済み [1]
		1	データ・エラー・チェック・イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み
		2	応答エラー・チェック・イネーブル FRCERR 観察イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み
		3	AERR# ドライブ・イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み
		4	イニシエータ・バス要求に対する BERR# イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み
		5	予約済み
		6	イニシエータ内部エラーに対する BERR# イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み
		7	BINIT# ドライバ・イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り / 書き込み

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		8	出力トライ・ステート・イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り
		9	BIST の実行 1 = イネーブル 0 = ディスエーブル 読み取り
		10	AERR# 観察イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り
		11	予約済み
		12	BINIT# 観察イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り
		13	イン・オーダー・キュー深さ 1 = 1 0 = 8 読み取り
		14	1M バイト・パワー・オン・リセット・ベクタ 1 = 1M バイト 0 = 4G バイト 読み取り専用
		15	FRC モード・イネーブル 1 = イネーブル 0 = ディスエーブル 読み取り専用
		17:16	APIC クラスタ ID 読み取り
		19:18	システムバス周波数読み取り 00 = 66MHz 10 = 100MHz 01 = 133MHz 11 = 予約済み
		21:20	対称型アービトレーション ID 読み取り
		25:22	クロック周波数比 読み取り
		26	低電力モード・イネーブル 読み取り / 書き込み
		27	クロック周波数比
		63:28	予約済み [1]

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
33H	51	TEST_CTL	テスト制御レジスタ
		29:0	予約済み
		30	ストリーミング・バッファ・ディスエーブル
		31	スプリット・ロックト・アクセスに対する LOCK# アサーションのディスエーブル
79H	121	BIOS_UPDT_TRIG	BIOS 更新トリガレジスタ
88	136	BBL_CR_D0[63:0]	チャンク 0 データ・レジスタ D[63:0]: L2 との間の書き込みと読み取りに使用される。
89	137	BBL_CR_D1[63:0]	チャンク 1 データ・レジスタ D[63:0]: L2 との間の書き込みと読み取りに使用される。
8A	138	BBL_CR_D2[63:0]	チャンク 2 データ・レジスタ D[63:0]: L2 との間の書き込みと読み取りに使用される。
8BH	139	BIOS_SIGN/BBL_CR_D3 [63:0]	BIOS 更新シグニチャ・レジスタまたはチャンク 3 データ・レジスタ D[63:0]: 使用するモデルに応じて L2 との間の書き込みと読み取りに使用される。
C1H	193	PerfCtr0 (PERFCTR0)	
C2H	194	PerfCtr1 (PERFCTR1)	
FEH	254	MTRRcap	
116	278	BBL_CR_ADDR [63:0] BBL_CR_ADDR [63:32] BBL_CR_ADDR [31:3] BBL_CR_ADDR [2:0]	アドレスレジスタ : キャッシュ初期化アクセスの間に指定されたアドレス（A31 ～ A3）を L2 に送るために使用される。 予約済み アドレスビット [35:3] 予約済み 0 に設定する。
118	280	BBL_CR_DECC[63:0]	データ ECC レジスタ D[7:0]: L2 との間の ECC の書き込みと読み取りに使用される。

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
119	281	BBL_CR_CTL	制御レジスタ : キャッシュ構成アクセス・メカニズムを通じて発行される L2 コマンドをプログラムするために使用される。また、L2 ルックアップ応答を受け取る。
		BL_CR_CTL[63:22]	予約済み
		BBL_CR_CTL[21]	プロセッサ・ナンバ 2 ディスエーブル = 1 イネーブル = 0
		BBL_CR_CTL[20:19]	予約済み
		BBL_CR_CTL[18]	ユーザ供給 ECC
		BBL_CR_CTL[17]	予約済み
		BBL_CR_CTL[16]	L2 ヒット
		BBL_CR_CTL[15:14]	予約済み
		BBL_CR_CTL[13:12]	L2 からの状態 修正された − 11、排他的 − 10、共有 − 01、 無効 − 00
		BBL_CR_CTL[11:10]	L2 からのウェイ ウェイ 0 − 00、ウェイ 1 − 01、ウェイ 2 − 10、 ウェイ 3 − 11
		BBL_CR_CTL[9:8]	L2 へのウェイ
		BBL_CR_CTL[7]	予約済み
		BBL_CR_CTL[6:5]	L2 への状態
		BBL_CR_CTL[4:0]	L2 コマンド
		01100	データ読み取り w/ LRU 更新（RLU）
		01110	タグ読み取り w/ データ読み取り（TRR）
		01111	タグ照会（TI）
		00010	L2 制御レジスタ読み取り（CR）
		00011	L2 制御レジスタ書き込み（CW）
		010 + MESI エンコード	タグ書き込み w/ データ読み取り（TWR）
		111 + MESI エンコード	タグ書き込み w/ データ書き込み（TWW）
		100 + MESI エンコード	タグ書き込み（TW）
11A	282	BBL_CR_TRIG	トリガレジスタ : キャッシュ構成アクセスがアクセスを開始するために使用される。データだけでの書き込み = 0
11B	283	BBL_CR_BUSY	ビジーレジスタ : キャッシュ構成アクセス L2 コマンドが進行中であることを示す。D[0] = 1 = ビジー

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
11E	286	BBL_CR_CTL3	制御レジスタ 3: L2 キャッシュを構成するために使用される。
		BBL_CR_CTL3[63:26]	予約済み
		BBL_CR_CTL3[25]	キャッシュ・バス・フラクション（読み取り専用）
		BBL_CR_CTL3[24]	予約済み
		BBL_CR_CTL3[23]	L2 ハードウェア・ディスエーブル（読み取り専用）
		BBL_CR_CTL3[22:20]	L2 物理アドレス範囲サポート
		111	64G バイト
		110	32G バイト
		101	16G バイト
		100	8G バイト
		011	4G バイト
		010	2G バイト
		001	1G バイト
		000	512M バイト
		BBL_CR_CTL3[19]	予約済み
		BBL_CR_CTL3[18]	キャッシュ状態エラー・チェック・イネーブル（読み取り / 書き込み）
		BBL_CR_CTL3[17:13]	キャッシュ・サイズ / バンク（読み取り / 書き込み）
		00001	256K バイト
		00010	512K バイト
		00100	1M バイト
		01000	2M バイト
		10000	4M バイト
		BBL_CR_CTL3[12:11]	L2 バンクの数（読み取り専用）
		BBL_CR_CTL3[10:9]	L2 アソシエイティビティ（読み取り専用）
		00	ダイレクト・マップト
		01	2 ウェイ
		10	4 ウェイ
		11	予約済み
		BBL_CR_CTL3[8]	L2 イネーブル（読み取り / 書き込み）
		BBL_CR_CTL3[7]	CRTN パリティ・チェック・イネーブル（読み取り / 書き込み）
		BBL_CR_CTL3[6]	アドレス・パリティ・チェック・イネーブル（読み取り / 書き込み）
		BBL_CR_CTL3[5]	ECC チェック・イネーブル（読み取り / 書き込み）
		BBL_CR_CTL3[4:1]	L2 キャッシュ・レイテンシ（読み取り / 書き込み）
		BBL_CR_CTL3[0]	L2 構成済み（読み取り / 書き込み）
174H	372	SYSENTER_CS_MSR	CPL 0 コード用の CS レジスタ・ターゲット
175H	373	SYSENTER_ESP_MSR	CPL 0 スタック用のスタックポインタ
176H	374	SYSENTER_EIP_MSR	CPL 0 コード・エントリ・ポイント
179H	377	MCG_CAP	
17AH	378	MCG_STATUS	
17BH	379	MCG_CTL	
186H	390	PerfEvtSel0 (EVNTSEL0)	
		7:0	イベント選択 （イベント・エンコーディングのリストについては、「性能カウンタ」の項を参照）

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		15:8	UMASK: ユニット・マスク・レジスタ すべてのカウント・オプションをイネーブルにするには 0 に設定する。
		16	USER: 1、2、3 の特権レベルでのイベントのカウント機能を制御する。
		17	OS: 0 の特権レベルでのイベントのカウント機能を制御する。
		18	E: 発生 / 持続期間モード選択 1 = 発生 0 = 持続期間
		19	PC: BP0 ピンによる性能カウンタのオーバーフローの通知をイネーブルにする。
		20	INT: APIC への入力によるカウンタのオーバーフローの通知をイネーブルにする。 1 = イネーブル 0 = ディスエーブル
		22	ENABLE: 両方のカウンタでの性能イベントのカウント機能をイネーブルにする。 1 = イネーブル 0 = ディスエーブル
		23	INV: CMASK 条件の結果を反転させる。 1 = 反転させる 0 = 反転させない
		31:24	CMASK（カウンタマスク）
187H	391	PerfEvtSel1 (EVNTSEL1)	
		7:0	イベント選択 （イベント・エンコーディングのリストについては、「性能カウンタ」の項を参照）
		15:8	UMASK（ユニットマスク） すべてのカウント・オプションをイネーブルにするにはユニット・マスク・レジスタをゼロに設定する。
		16	USER: 1、2、3 の特権レベルでのイベントのカウント機能を制御する。
		17	OS: 0 の特権レベルでのイベントのカウント機能を制御する。

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス		レジスタ名	ビット説明
16 進	10 進		
		18	E: 発生 / 持続期間モード選択 1 = 発生 0 = 持続期間
		19	PC: BP0 ピンによる性能カウンタのオーバーフローの通知をイネーブルにする。
		20	INT: APIC への入力によるカウンタのオーバーフローの通知をイネーブルにする。 1 = イネーブル 0 = ディスエーブル
		23	INV: CMASK 条件の結果を反転させる。 1 = 反転させる 0 = 反転させない
		31:24	CMASK（カウンタマスク）
1D9H	473	DEBUGCTLMSR	
		0	最後の分岐の記録をイネーブル / ディスエーブルにする。
		1	分岐トラップフラグ
		2	性能モニタリング / ブレークポイント・ピン
		3	性能モニタリング / ブレークポイント・ピン
		4	性能モニタリング / ブレークポイント・ピン
		5	性能モニタリング / ブレークポイント・ピン
		6	実行トレース・メッセージをイネーブル / ディスエーブルにする。
		13:7	予約済み
		14	実行トレース・メッセージをイネーブル / ディスエーブルにする。
		15	実行トレース・メッセージをイネーブル / ディスエーブルにする。
1DBH	475	LASTBRANCHFROMIP	
1DCH	476	LASTBRANCHTOIP	
1DDH	477	LASTINTFROMIP	
1DEH	478	LASTINTTOIP	
1E0H	480	ROB_CR_BKUPTMPDR6	
		1:0	予約済み
		2	高速ストリング・イネーブル・ビット。デフォルトはイネーブルである。
200H	512	MTRRphysBase0	

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス		レジスタ名	ビット説明
16 進	10 進		
201H	513	MTRRphysMask0	
202H	514	MTRRphysBase1	
203H	515	MTRRphysMask1	
204H	516	MTRRphysBase2	
205H	517	MTRRphysMask2	
206H	518	MTRRphysBase3	
207H	519	MTRRphysMask3	
208H	520	MTRRphysBase4	
209H	521	MTRRphysMask4	
20AH	522	MTRRphysBase5	
20BH	523	MTRRphysMask5	
20CH	524	MTRRphysBase6	
20DH	525	MTRRphysMask6	
20EH	526	MTRRphysBase7	
20FH	527	MTRRphysMask7	
250H	592	MTRRfix64K_00000	
258H	600	MTRRfix16K_80000	
259H	601	MTRRfix16K_A0000	
268H	616	MTRRfix4K_C0000	
269H	617	MTRRfix4K_C8000	
26AH	618	MTRRfix4K_D0000	
26BH	619	MTRRfix4K_D8000	
26CH	620	MTRRfix4K_E0000	
26DH	621	MTRRfix4K_E8000	
26EH	622	MTRRfix4K_F0000	
26FH	623	MTRRfix4K_F8000	
2FFH	767	MTRRdefType	
		2:0	デフォルトのメモリタイプ
		10	固定 MTRR イネーブル
		11	MTRR イネーブル
400H	1024	MC0_CTL	
401H	1025	MC0_STATUS	
		63	MC_STATUS_V
		62	MC_STATUS_O

表 B-3. P6 ファミリ・プロセッサの MSR（続き）

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
		61	MC_STATUS_UC
		60	MC_STATUS_EN（注 : MC0_STATUS 用にのみ、このビットは 1 に固定される。）
		59	MC_STATUS_MISCV
		58	MC_STATUS_ADDRV
		57	MC_STATUS_DAM
		31:16	MC_STATUS_MCACOD
		15:0	MC_STATUS_MSCOD
402H	1026	MC0_ADDR	
403H	1027	MC0_MISC	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	MC0_STATUS と同じビット定義。
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	MC0_STATUS と同じビット定義。
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	MC0_STATUS と同じビット定義。
40EH	1038	MC4_ADDR	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。
40FH	1039	MC4_MISC	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	ビット 0、4、57、61 が 1 に固定されていることを除いて、MC0_STATUS と同じビット定義。
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	MCA アーキテクチャでは定義されているが、P6 ファミリ・プロセッサにはインプリメントされていない。

注 :

1. このレジスタのビット 0 は何回か再定義されているが、P6 ファミリ・プロセッサではもはや使用されていない。
2. プロセッサ・ナンバ機能をディスエーブルにするには、BBL_CR_CTL MSR（アドレス 119h のモデル固有レジスタ）のビット 21 を "1" にセットする。一度セットすると、BBL_CR_CTL のビット 21 はクリアできなくなる。このビットは、1 回だけ書き込みできる。一度ディスエーブルにされたプロセッサ・ナンバ機能は、プロセッサがリセットされるまでディスエーブルのままになる。
3. インテル Pentium III プロセッサは、新しいシャットダウン機構を使用して、FSB 周波数のオーバークロッキングを防止する。選択された FSB 周波数が内部 FSB 周波数より大きい場合は、プロセッサはシャットダウンされる。選択された FSB 周波数が内部 FSB 周波数より小さい場合は、BIOS は、ビット 11 を使用して BIOS 独自のシャットダウン・ポリシーを使用できる。

B.4. インテル® Pentium® プロセッサの MSR

以下の MSR は、インテル® Pentium® プロセッサ用に定義されたものである。P5_MC_ADDR、P5_MC_TYPE、および TSC MSR（インテル® Pentium ® 4 プロセッサでは、IA32_P5_MC_ADDR、IA32_P5_MC_TYPE、IA32_TIME_STAMP_COUNTER と呼ばれる）は、アーキテクチャ的である。すなわち、これらのレジスタをアクセスするコードは、例外を生成することなく、インテル Pentium 4 プロセッサおよび P6 ファミリ・プロセッサ上で動作する（B.5. 節「アーキテクチャ的な MSR」を参照）。CESR、CTR0、CTR1 の MSR は、インテル Pentium プロセッサ固有のものである。これらのレジスタにアクセスするコードは、インテル Pentium 4 プロセッサおよび P6 ファミリ・プロセッサ上では例外を生成する。

表 B-4. インテル® Pentium® プロセッサの MSR

レジスタアドレス			
16 進	10 進	レジスタ名	ビット説明
0H	0	P5_MC_ADDR	14.7.3. 項「インテル® Pentium® プロセッサのマシンチェック例外の処理」を参照のこと。
1H	1	P5_MC_TYPE	14.7.3. 項「インテル® Pentium® プロセッサのマシンチェック例外の処理」を参照のこと。
10H	16	TSC	15.7. 節「タイムスタンプ・カウンタ」を参照のこと。
11H	17	CESR	15.12.1. 項「制御 / イベント選択レジスタ（CESR）」を参照のこと。
12H	18	CTR0	15.12.3. 項「カウント対象のイベント」を参照のこと。
13H	19	CTR1	15.12.3. 項「カウント対象のイベント」を参照のこと。

B.5. アーキテクチャ的な MSR

表 B-1.、表 B-3.、表 B-4. に示す MSR の多くは、後継の IA-32 プロセッサ・ファミリへと引き継がれているため、IA-32 アーキテクチャの一部とみなすことができる。インテル® Pentium® 4 プロセッサ以降、これらの「アーキテクチャ的な MSR」は名前が変更され、「IA32_」プリフィックスが付けられた。表 B-5. は、アーキテクチャ的な MSR、そのアドレス、現在の名前、以前の IA-32 プロセッサでの名前、およびこれらの MSR が導入された IA-32 プロセッサ・ファミリを示している。表 B-1.、表 B-3.、表 B-4. には載っているが表 B-5. には載っていない MSR（および、インテル Pentium 4 プロセッサでプリフィックス「MSR_」が付いている MSR）は、プロセッサ固有のものと見なされる。プロセッサ固有の MSR にアクセスするコードがその MSR をサポートしないプロセッサ上で実行されると、例外が生成される。

表 B-5. IA-32 アーキテクチャ的な MSR

レジスタ アドレス		アーキテクチャ的な名前	以前の名前	導入された IA-32 プロセッサ・ファミリ
16 進	10 進			
0H	0	IA32_P5_MC_ADDR	P5_MC_ADDR	インテル® Pentium® プロセッサ
1H	1	IA32_P5_MC_TYPE	P5_MC_TYPE	インテル Pentium プロセッサ
10H	16	IA32_TIME_STAMP_COUNTER	TSC	インテル Pentium プロセッサ
17H	23	IA32_PLATFORM_ID	MSR_PLATFORM_ID	P6 ファミリ・プロセッサ
1BH	27	IA32_APIC_BASE	APIC_BASE	P6 ファミリ・プロセッサ
79H	121	IA32_BIOS_UPDT_TRIG	BIOS_UPDT_TRIG	P6 ファミリ・プロセッサ
8BH	139	IA32_BIOS_SIGN_ID	BIOS_SIGN/BBL_CR_D3	P6 ファミリ・プロセッサ
FEH	254	IA32_MTRRCAP	MTRRcap	P6 ファミリ・プロセッサ
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR	P6 ファミリ・プロセッサ
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR	P6 ファミリ・プロセッサ
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR	P6 ファミリ・プロセッサ
179H	377	IA32_MCG_CAP	MCG_CAP	P6 ファミリ・プロセッサ
17AH	378	IA32_MCG_STATUS	MCG_STATUS	P6 ファミリ・プロセッサ
17BH	379	IA32_MCG_CTL	MCG_CTL	P6 ファミリ・プロセッサ
180H	384	IA32_MCG_EAX		インテル® Pentium® 4 プロセッサ
181H	385	IA32_MCG_EBX		インテル Pentium 4 プロセッサ
182H	386	IA32_MCG_ECX		インテル Pentium 4 プロセッサ

表 B-5. IA-32 アーキテクチャ的な MSR（続き）

レジスタアドレス				
16 進	10 進	アーキテクチャ的な名前	以前の名前	導入された IA-32 プロセッサ・ファミリ
183H	387	IA32_MCG_EDX		インテル Pentium 4 プロセッサ
184H	388	IA32_MCG_ESI		インテル Pentium 4 プロセッサ
185H	389	IA32_MCG_EDI		インテル Pentium 4 プロセッサ
186H	390	IA32_MCG_EBP		インテル Pentium 4 プロセッサ
187H	391	IA32_MCG_ESP		インテル Pentium 4 プロセッサ
188H	392	IA32_MCG_EFLAGS		インテル Pentium 4 プロセッサ
189H	393	IA32_MCG_EIP		インテル Pentium 4 プロセッサ
18AH	394	IA32_MCG_MISC		インテル Pentium 4 プロセッサ
19AH	410	IA32_CLOCK_MODULATION		インテル Pentium 4 プロセッサ
19BH	411	IA32_THERM_INTERRUPT		インテル Pentium 4 プロセッサ
19CH	412	IA32_THERM_STATUS		インテル Pentium 4 プロセッサ
1A0H	416	IA32_MISC_ENABLE		インテル Pentium 4 プロセッサ
1D9H	473	IA32_DEBUGCTL	DEBUGCTLMSR	P6 ファミリ・プロセッサ
200H	512	IA32_MTRR_PHYSBASE0	MTRRphysBase0	P6 ファミリ・プロセッサ
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	P6 ファミリ・プロセッサ
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	P6 ファミリ・プロセッサ
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	P6 ファミリ・プロセッサ
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	P6 ファミリ・プロセッサ
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	P6 ファミリ・プロセッサ
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	P6 ファミリ・プロセッサ
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	P6 ファミリ・プロセッサ
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	P6 ファミリ・プロセッサ
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	P6 ファミリ・プロセッサ
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	P6 ファミリ・プロセッサ
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	P6 ファミリ・プロセッサ
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	P6 ファミリ・プロセッサ

表 B-5. IA-32 アーキテクチャ的な MSR（続き）

レジスタアドレス		アーキテクチャ的な名前	以前の名前	導入された IA-32 プロセッサ・ファミリ
16 進	10 進			
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	P6 ファミリ・プロセッサ
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	P6 ファミリ・プロセッサ
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	P6 ファミリ・プロセッサ
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	P6 ファミリ・プロセッサ
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	P6 ファミリ・プロセッサ
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	P6 ファミリ・プロセッサ
268H	616	IA32_MTRR_FIX4K_C0000	MTRRfix4K_C0000	P6 ファミリ・プロセッサ
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	P6 ファミリ・プロセッサ
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	P6 ファミリ・プロセッサ
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	P6 ファミリ・プロセッサ
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	P6 ファミリ・プロセッサ
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	P6 ファミリ・プロセッサ
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	P6 ファミリ・プロセッサ
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	P6 ファミリ・プロセッサ
277H	631	IA32_CR_PAT	IA32_CR_PAT	P6 ファミリ・プロセッサ
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType	P6 ファミリ・プロセッサ
3F1H	1009	IA32_PEBS_ENABLE		インテル Pentium 4 プロセッサ
400H	1024	IA32_MC0_CTL	MC0_CTL	P6 ファミリ・プロセッサ
401H	1025	IA32_MC0_STATUS	MC0_STATUS	P6 ファミリ・プロセッサ
402H	1026	IA32_MC0_ADDR	MC0_ADDR	P6 ファミリ・プロセッサ
403H	1027	IA32_MC0_MISC	MC0_MISC	P6 ファミリ・プロセッサ
404H	1028	IA32_MC1_CTL	MC1_CTL	P6 ファミリ・プロセッサ
405H	1029	IA32_MC1_STATUS	MC1_STATUS	P6 ファミリ・プロセッサ
406H	1030	IA32_MC1_ADDR	MC1_ADDR	P6 ファミリ・プロセッサ
407H	1031	IA32_MC1_MISC	MC1_MISC	P6 ファミリ・プロセッサ
408H	1032	IA32_MC2_CTL	MC2_CTL	P6 ファミリ・プロセッサ
409H	1033	IA32_MC2_STATUS	MC2_STATUS	P6 ファミリ・プロセッサ
40AH	1034	IA32_MC2_ADDR	MC2_ADDR	P6 ファミリ・プロセッサ
40BH	1035	IA32_MC2_MISC	MC2_MISC	P6 ファミリ・プロセッサ
40CH	1036	IA32_MC3_CTL	MC3_CTL	P6 ファミリ・プロセッサ
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	P6 ファミリ・プロセッサ
40EH	1038	IA32_MC3_ADDR	MC3_ADDR	P6 ファミリ・プロセッサ
40FH	1039	IA32_MC3_MISC	MC3_MISC	P6 ファミリ・プロセッサ

表 B-5. IA-32 アーキテクチャ的な MSR（続き）

レジスタ アドレス		アーキテクチャ的な名前	以前の名前	導入された IA-32 プロセッサ・ファミリ
16 進	10 進			
600H	1536	IA32_DS_AREA		インテル Pentium 4 プロセッサ

C

P6 ファミリ・プロセッサの MP 初期化

付録 C
P6 ファミリ・プロセッサの MP 初期化

C

本章では、複数の P6 ファミリ・プロセッサを使用するシステムの MP 初期化プロセスについて説明する。このプロセスは、インテル® Pentium® Pro プロセッサで導入された MP 初期化プロトコルを使用する（7.5. 節「マルチプロセッサ（MP）初期化」を参照）。P6 ファミリ・プロセッサの場合、通常はこのプロトコルを使用して、単一のシステムバス上に存在する 2 個または 4 個のプロセッサをブートする。ただし、複数の APIC バスが連結されている場合は、このプロトコルはマルチクラスタ・システム内の 2 ～ 15 個のプロセッサをサポートする。それより大きなシステムはサポートしていない。

C.1. P6 ファミリ・プロセッサの MP 初期化プロセスの概要

MP 初期化プロトコルの実行中に、1 つのプロセッサがブートストラップ・プロセッサ（BSP）として選択され、その他の論理プロセッサはアプリケーション・プロセッサ（AP）として指定される（7.5.1. 項「BSP プロセッサと AP プロセッサ」を参照）。その後は、BSP が自分自身と AP の初期化を管理する。この初期化プロセスには、BIOS の初期化コードとオペレーティング・システムの初期化コードの実行が含まれる。

MP プロトコルを実行するシステムは、以下の必要条件と制限を満たす必要がある。

- APIC クロック（APICLK）をサポートする必要がある。
- MP プロトコルは、電源投入または RESET 後にのみ実行される。MP プロトコルが完了し、BSP が選択されてしまうと、これ以降に（特定のプロセッサに対して、またはシステム全体に対して）INIT が発行されても、MP プロトコルは繰り返し実行されない。この場合、各プロセッサは、（APIC_BASE MSR 内の）自分の BSP フラグを確認し、BIOS のブートストラップ・コードを実行するか（そのプロセッサが BSP の場合）、SIPI 待機状態に移行するか（AP の場合）を判断する。
- MP 初期化プロトコルが完了するまでは、プロセッサに割り込みを伝達できるシステム内のすべてのデバイスに対して、プロセッサへの割り込みの伝達を禁止しなければならない。割り込みが禁止される期間には、BSP が AP に対して INIT-SIPI-SIPI シーケンスを発行してから、AP がそのシーケンス内の最後の SIPI に応答するまでの時間が含まれる。

MP 初期化プロトコルのブート段階では、以下の特殊目的のプロセッサ間割り込み（IPI）が使用される。これらの IPI は、APIC バス上にブロードキャストされる。

- ブート IPI（BIPI）－ システムバス上のプロセッサのグループの中から BSP を選択し、それ以外のプロセッサを AP として指定する、アービトレーション機構を開始する。電源投入または RESET 後、システムバス上の各プロセッサは、すべてのプロセッサに対して BIPI をブロードキャストする。
- 最終ブート IPI（FIPI）－ BSP の BIOS 初期化プロシージャを開始する。この IPI は、システムバス上のすべてのプロセッサにブロードキャストされるが、BSP だけがそれに応答する。BSP は、この IPI に応答して、リセットベクタの位置にある BIOS 初期化コードの実行を開始する。
- スタートアップ IPI（SIPI）－ AP の初期化プロシージャを開始する。SIPI メッセージには、BIOS 内の AP 初期化コードへのベクタが含まれる。

表 C-1. は、ブート段階の IPI の各種のフィールドを示している。

表 C-1. ブート段階の IPI メッセージの形式

タイプ	デスティネーション・フィールド	デスティネーション簡略表記	トリガモード	レベル	デスティネーション・モード	伝達モード	ベクタ（16 進数）
BIPI	使用しない	自分自身を含むすべてのプロセッサ	エッジ	デアサート	無関係	固定（000）	40 ～ 4E*
FIPI	使用しない	自分自身を含むすべてのプロセッサ	エッジ	デアサート	無関係	固定（000）	10
SIPI	使用される	自分自身を除くすべてのプロセッサ	エッジ	アサート	物理	スタートアップ（110）	00 ～ FF

注 :

*　すべての P6 ファミリ・プロセッサの場合。

BIPI メッセージの場合、ベクタ・フィールドの下位 4 ビットには、メッセージを発行しているプロセッサの APIC ID が格納され、上位 4 ビットには、メッセージの「世代 ID」が格納される。すべての P6 ファミリ・プロセッサは、4H の世代 ID を持つ。したがって、BIPI は、40H ～ 4EH の範囲のベクタ値を使用する（FH は有効な APIC ID ではないため、4FH は使用できない）。

C.2. MP 初期化プロトコルのアルゴリズム

システムの電源投入またはRESET後、システム内のP6ファミリ・プロセッサは、MP初期化プロトコルのアルゴリズムを実行し、システムバス上の各プロセッサを初期化する。このアルゴリズムの実行の過程で、以下のブートアップ操作と初期化操作が実行される。

1. システム構成に基づいて、システムバス上の各プロセッサに個別の APIC ID が割り当てられる（7.5.5. 項「MP システム内のプロセッサの識別」を参照）。この ID は、各プロセッサのローカル APIC ID レジスタに書き込まれる。
2. 各プロセッサは、システムバス上の他のプロセッサと同時に、自分の内部 BIST を実行する。BIST が完了すると（T0 の時点）、各プロセッサは、「自分自身を含むすべてのプロセッサ」に対して BIPI をブロードキャストする（図 C-1. を参照）。
3. APIC アービトレーション・ハードウェアにより、すべての APIC が、一度に 1 つずつ BIPI に応答する（T1、T2、T3、T4 の時点）。
4. 最初の BIPI が受信されると（T1 の時点）、各 APIC は、BIPI のベクタ・フィールドの最下位 4 ビットと自分の APIC ID を比較する。ベクタと APIC ID が一致した場合は、そのプロセッサは、自分の IA32_APIC_BASE MSR 内の BSP フラグをセットし、自分を BSP として選択する。ベクタと APIC ID が一致しない場合は、プロセッサは、自分を AP として選択し、「SIPI 待機」状態に移行する。（図 C-1. では、プロセッサ 1 が送信した BIPI が最初に処理されているので、BIPI が BSP になる。）
5. 新たに設定された BSP は、「自分自身を含むすべてのプロセッサ」に対して FIPI メッセージをブロードキャストする。FIPI は、BSP でないプロセッサが発行した BIPI が完了した後で処理される。

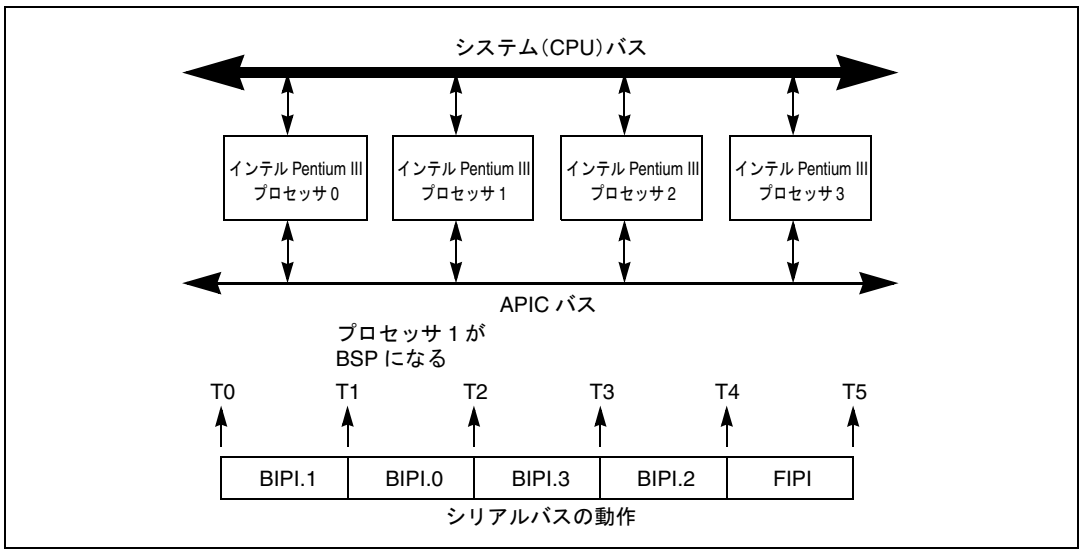


図 C-1. 複数のインテル® Pentium® III プロセッサを使用する MP システム

6. BSP の設定後、すべてのプロセッサは、未処理の BIPI を一度に 1 つずつ（T2、T3、T4 の時点で）受信し、それらの BIPI を無視する。

7. 最後に FIPI が受信されると（T5 の時点）、BSP だけがそれに応答する。BSP は、FIPI に応答して、リセットベクタ（物理アドレス FFFF FFF0H）を始点とする BIOS ブートストラップ・コードをフェッチし、実行する。

8. ブートストラップ・コードの一部として、BSP は、ACPI テーブルと MP テーブルを作成し、これらのテーブルに自分の初期 APIC ID を追加する。

9. ブートストラップ・プロシージャの最後に、BSP は、システム内のすべての AP に対して SIPI メッセージをブロードキャストする。この SIPI メッセージには、(000V V000H の位置にある）BIOS の AP 初期化コードへのベクタが含まれている（VV は、SIPI メッセージに含まれるベクタである)。

10. すべての AP は、SIPI メッセージに応答して、BIOS 初期化セマフォに対する競争に参加する。最初にセマフォにアクセスした AP が、初期化コードの実行を開始する（セマフォの詳細については、「MP 初期化コード」を参照)。AP 初期化プロシージャの一部として、この AP は、ACPI テーブルと MP テーブルに自分の APIC ID 番号を追加する。初期化プロシージャが完了すると、この AP は、CLI 命令を実行して（EFLAGS レジスタの IF フラグをクリアし)、ホルト状態に移行する。

11. すべての AP がセマフォにアクセスして AP 初期化コードを実行し、すべての AP の APIC ID が ACPI テーブルと MP テーブル内の適切な位置に書き込まれると、BSP は、システムバスに接続されているプロセッサの数に合わせてカウントを設定し、BIOS のブートストラップ・コードの実行を完了する。次に、BSP は、オペレーティング・システムのブートストラップ / スタートアップ・コードの実行を開始する。

12. BSP がオペレーティング・システムのブートストラップ / スタートアップ・コードを実行している間、AP はホルト状態のままになる。ホルト状態の AP は、INIT、NMI、SMI にのみ応答する。また、これらのプロセッサは、スヌープと STPCLK# ピンのアサートにも応答する。

MP プロトコルを使用して MP 内の IA-32 プロセッサをブートする例については、7.5.4. 項「MP 初期化の例」を参照のこと。このコードは、MP プロトコルを使用するすべての IA-32 プロセッサ上で正常に実行される。

C.2.1. MP 初期化プロトコルの実行中のエラーの検出と処理

MP 初期化段階で、APIC バス上にエラーが発生することがある。これらのエラーには、一時的なものと永続的なものがあり、各種の障害メカニズム（例えば、トレースの破壊、バスの使用中のソフトエラーなど）によって発生する。シリアルバスに関連するエラーが発生すると、APIC チェックサム・エラーまたは受け入れエラーが報告される。

MP 初期化プロトコルは、初期化の過程でエラーが発生した場合、プロセッサが次のように動作することを前提としている。

- MP初期化プロトコルの実行中にAPICバス上でエラーが検出された場合は、エラーを検出したプロセッサはシャットダウンされる。
- プロセッサが BIST シーケンスの実行に失敗した場合でも、それらのプロセッサは MP 初期化プロトコルを実行する。

D

LINT0 入力と LINT1 入力のプログラミング

付録 D
LINT0 入力と LINT1 入力のプログラミング

D

次の手順では、(付録C「P6ファミリ・プロセッサのMP初期化」と付録D「LINT0入力とLINT1入力の プログラミング」で説明しているように)マルチプロセッサをブートして初期化した後にプロセッサのLINT0とLINT1ローカルAPICピンをプログラムする方法について説明している。

D.1. 定数

次の定数が定義されている。

```
LVT1        EQU 0FEE00350H
LVT2        EQU 0FEE00360H
LVT3        EQU 0FEE00370H
SVR         EQU 0FEE000F0H
```

D.2. LINT[0:1] ピンのプログラミング手順

次の手順を使用してLINT[1:0]ピンをプログラムする。

1. 8259割り込みをマスクする。
2. まだイネーブルになっていない場合は、SVR（スプリアス・ベクタ・レジスタ）によってAPICをイネーブルにする。

```
MOV ESI, SVR           ; address of SVR
MOV EAX, [ESI]
OR  EAX, APIC_ENABLED ; set bit 8 to enable (0 on reset)
MOV [ESI], EAX
```

3. LVT1をExtINTとしてプログラムする。このExtINTは、外部的に接続されている割り込みコントローラで始まった割り込みとしてデスティネーションにリストされているすべてのプロセッサ・コアのINTR信号に信号を送信する。

```
MOV ESI, LVT1
MOV EAX, [ESI]
AND EAX, 0FFFE58FFH   ; mask off bits 8-10, 12, 14 and 16
OR  EAX, 700H         ; Bit 16=0 for not masked, Bit 15=0 for edge
                      ; triggered, Bit 13=0 for high active input
                      ; polarity, Bits 8-10 are 111b for ExtINT
MOV [ESI], EAX        ; Write to LVT1
```

4. LVT2 を NMI としてプログラムする。これは、デスティネーションにリストされているすべてのプロセッサ・コアの NMI 信号で信号を送信する。

```
MOV ESI, LVT2
MOV EAX, [ESI]
AND EAX, 0FFFE58FFH   ; mask off bits 8-10 and 15
OR  EAX, 000000400H   ; Bit 16=0 for not masked, Bit 15=0 edge
                      ; triggered, Bit 13=0 for high active input
                      ; polarity, Bits 8-10 are 100b for NMI
MOV [ESI], EAX        ; Write to LVT2
;Unmask 8259 interrupts and allow NMI.
```

E

マシン・チェック・エラー・コードの解釈

付録 E
マシン・チェック・エラー・コードの解釈

E

モデル固有およびその他の情報フィールドのエンコーディングは、06H プロセッサ・ファミリと 0FH プロセッサ・ファミリでは異なっている。以下の各節では、その違いについて説明する。モデル情報をチェックするには、CPUID 命令を使用する。

E.1. ファミリ 06H 固有のマシン・エラー・コードのデコーディング

プロセッサ・ファミリ 06H によるマシン・エラー・コードのレポーティングは、IA32_MCi_STATUS から読み出された値に基づく（図 E-1. を参照）。

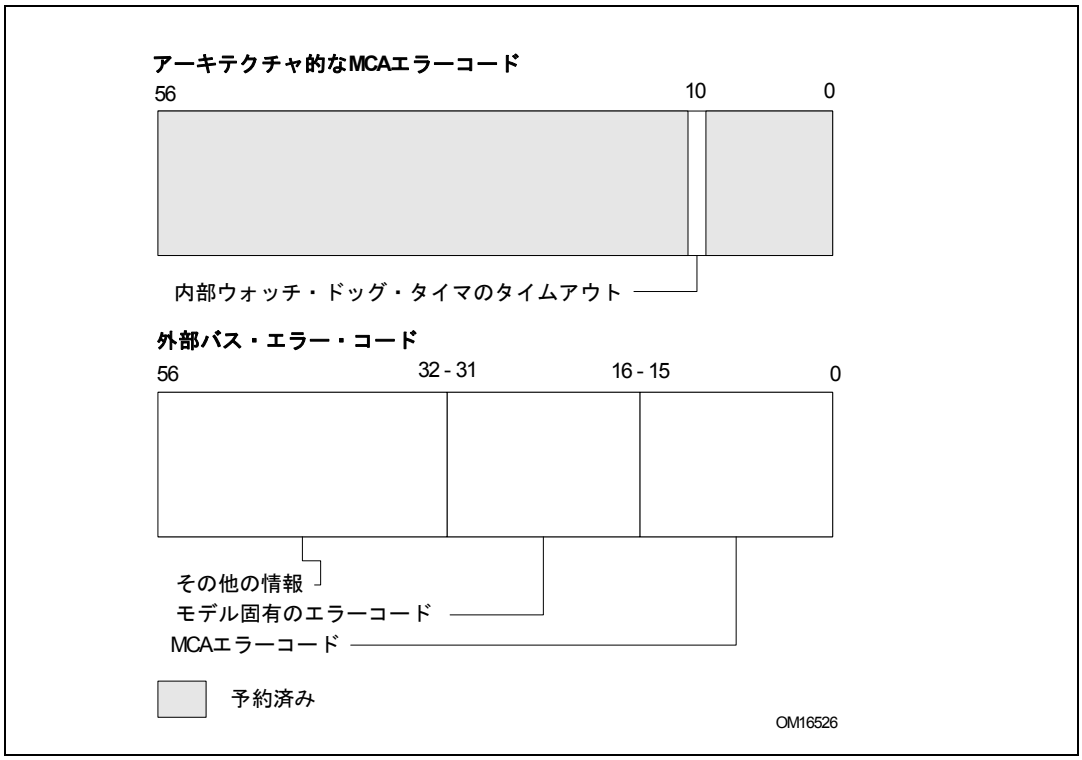


図 E-1. ファミリ 06H の IA32_MCi_STATUS のエンコーディング

表 E-1. に、プロセッサ・ファミリ 06H の IA32_MCi_STATUS で報告された内部ウォッチ・ドッグ・タイマのタイムアウト・マシン・チェック・エラーを解釈する方法を示す。

表 E-1. ファミリ 06H の IA32_MCi_STATUS で報告された
内部ウォッチ・ドッグ・タイマ・エラーに対するエンコーディング

	ビット番号	ビットの機能	ビットの記述
アーキテクチャ的な MCA エラーコード	0-15	0000010000000000	内部ウォッチ・ドッグ・タイマのタイムアウト。ウォッチ・ドッグ・タイマのタイムアウトは、BINIT ドライバ回路がイネーブルの場合にのみ発生することに注意する。
モデル固有のエラーコード	16-31	予約済み	予約済み
その他の情報	32-56	予約済み	予約済み

表 E-2. に、外部バス上で発生したエラーを解釈する方法を示す。

表 E-2. ファミリ 06H の外部バスエラーに対する 32_MCi_STATUS のエンコーディング

タイプ	ビット番号	ビットの機能	ビットの記述
MCA 予約済みエラーコード	0-1	予約済み	予約済み
	2-3	外部バスエラーの場合 : 特殊サイクルまたは I/O 内部タイムアウトの場合 : 予約済み	外部バスエラーの場合 • アクセスが特殊サイクルであった場合は、ビット 2 が 1 にセットされる。 • アクセスが特殊サイクルか I/O サイクルであった場合は、ビット 3 が 1 にセットされる。 内部タイムアウトの場合 : 予約済み
	4-7	外部バスエラーの場合 : 読み取り / 書き込み 内部タイムアウトの場合 : 予約済み	外部バスエラー 00WR の場合 書き込みに対しては W=1 読み取りに対しては R=1 内部タイムアウトの場合 : 予約済み
	8-9	予約済み	予約済み
	10-11	10 01	外部バスエラー 内部ウォッチ・ドッグ・タイマのタイムアウト
	12-15	予約済み	予約済み
モデル固有エラー	16-18	予約済み	予約済み

表 E-2. ファミリ 06H の外部バスエラーに対する 32_MCi_STATUS のエンコーディング（続き）

タイプ	ビット番号	ビットの機能	ビットの記述
	19-24	バスキューの要求タイプ	BQ_DCU_READ_TYPE エラーに対しては 000000。 BQ_IFU_DEMAND_TYPE エラーに対しては 000010。 BQ_IFU_DEMAND_NC_TYPE エラーに対しては 000011。 BQ_DCU_RFO_TYPE エラーに対しては 000100。 BQ_DCU_RFO_LOCK_TYPE エラーに対しては 000101。 BQ_DCU_ITOM_TYPE エラーに対しては 000110。 BQ_DCU_WB_TYPE エラーに対しては 001000。 BQ_DCU_WCEVICT_TYPE エラーに対しては 001010。 BQ_DCU_WCLINE_TYPE エラーに対しては 001011。 BQ_DCU_BTM_TYPE エラーに対しては 001100。
			BQ_DCU_INTACK_TYPE エラーに対しては 001101。 BQ_DCU_INVALL2_TYPE エラーに対しては 001110。 BQ_DCU_FLUSHL2_TYPE エラーに対しては 001111。 BQ_DCU_PART_RD_TYPE エラーに対しては 010000。 BQ_DCU_PART_WR_TYPE エラーに対しては 010010。 BQ_DCU_SPEC_CYC_TYPE エラーに対しては 010100。 BQ_DCU_IO_RD_TYPE エラーに対しては 011000。 BQ_DCU_IO_WR_TYPE エラーに対しては 011001。 BQ_DCU_LOCK_RD_TYPE エラーに対しては 011100。 BQ_DCU_SPLOCK_RD_TYPE エラーに対しては 011110。 BQ_DCU_LOCK_WR_TYPE エラーに対しては 011101。 BQ_IFU_DEMAND_TYPE エラーに対しては 000010。 BQ_IFU_DEMAND_NC_TYPE エラーに対しては 000011。 BQ_DCU_RFO_TYPE エラーに対しては 000100。 BQ_DCU_RFO_LOCK_TYPE エラーに対しては 000101。 BQ_DCU_ITOM_TYPE エラーに対しては 000110。 BQ_DCU_WB_TYPE エラーに対しては 001000。 BQ_DCU_WCEVICT_TYPE エラーに対しては 001010。 BQ_DCU_WCLINE_TYPE エラーに対しては 001011。 BQ_DCU_BTM_TYPE エラーに対しては 001100。

表 E-2. ファミリ 06H の外部バスエラーに対する 32_MCi_STATUS のエンコーディング（続き）

タイプ	ビット番号	ビットの機能	ビットの記述
			BQ_DCU_INTACK_TYPE エラーに対しては 001101。 BQ_DCU_INVALL2_TYPE エラーに対しては 001110。 BQ_DCU_FLUSHL2_TYPE エラーに対しては 001111。 BQ_DCU_PART_RD_TYPE エラーに対しては 010000。 BQ_DCU_PART_WR_TYPE エラーに対しては 010010。 BQ_DCU_SPEC_CYC_TYPE エラーに対しては 010100。 BQ_DCU_IO_RD_TYPE エラーに対しては 011000。 BQ_DCU_IO_WR_TYPE エラーに対しては 011001。 BQ_DCU_LOCK_RD_TYPE エラーに対しては 011100。 BQ_DCU_SPLOCK_RD_TYPE エラーに対しては 011110。 BQ_DCU_LOCK_WR_TYPE エラーに対しては 011101。
	27-25	バスキューのエラータイプ	BQ_ERR_HARD_TYPE エラーに対しては 000。 BQ_ERR_DOUBLE_TYPE エラーに対しては 001。 BQ_ERR_AERR2_TYPE エラーに対しては 010。 BQ_ERR_SINGLE_TYPE エラーに対しては 100。 BQ_ERR_AERR1_TYPE エラーに対しては 101。
	28	FRC エラー	FRC エラーがアクティブである場合は 1。
	29	BERR	BERR がドライブされている場合は 1。
	30	内部 BINIT	このプロセッサに対して BINIT がドライブされている場合は 1。
	31	予約済み	予約済み

表 E-2. ファミリ 06H の外部バスエラーに対する 32_MCi_STATUS のエンコーディング（続き）

タイプ	ビット番号	ビットの機能	ビットの記述
他の情報	32-34	予約済み	予約済み
	35	外部 BINIT	外部バスから BINIT を受け取った場合は 1。
	36	RESPONSE PARITY ERROR	このコンポーネントが応答トランザクションに対して RS[2:0]# ピン上でパリティエラーを受け取った場合は、このビットが IA32_MCi_STATUS 内でアサートされる。RS 信号は、RSP# 外部ピンによってチェックされる。
	37	BUS BINIT	このコンポーネントがスプリット・トランザクション（64 ビット外部バス・インターフェイスを越えて 2 つのアクセスに分割する必要があったアクセス）に対してハードエラー応答を受け取った場合は、このビットが IA32_MCi_STATUS 内でアサートされる。
	38	TIMEOUT BINIT	このコンポーネントが ROB タイムアウト（あらかじめ決められた時間内にリタイアしたマイクロ命令がなかったことを示す）を検出した場合は、このビットが IA32_MCi_STATUS 内でアサートされる。 ROB タイムアウトは、15 ビットの ROB タイムアウト・カウンタの上位ビットに 1 が入れられた場合に発生する。タイマは、マイクロ命令がリタイアするか、コア・プロセッサによって例外が検出されるか、RESET がアサートされるか、ROB BINIT が発生した場合にクリアされる。 ROB タイムアウト・カウンタは、バスクロック（バスクロックは、コアクロックの 1:2、1:3、1:4 になる）の 128 分の 1 である 8 ビットの PIC タイマによってあらかじめスケーリングされている。8 ビットの PIC タイマの繰り上げが発生した場合は、ROB カウンタは 1 だけカウントアップされる。このビットがアサートされている間は、別のエラーで上書きできない。
	39-41	予約済み	予約済み
	42	HARD ERROR	このコンポーネントが、ハードエラー応答を受け取ったバス・トランザクションを起動した場合は、このビットが IA32_MCi_STATUS 内でアサートされる。このビットがアサートされている間は、別のエラーで上書きできない。
	43	IERR	このコンポーネントが、IERR ピンがアサートされる原因となった故障を経験した場合は、このビットが IA32_MCi_STATUS 内でアサートされる。このビットがアサートされている間は、別のエラーで上書きできない。

表 E-2. ファミリ 06H の外部バスエラーに対する 32_MCi_STATUS のエンコーディング（続き）

タイプ	ビット番号	ビットの機能	ビットの記述
	44	AERR	このコンポーネントが、アドレス・パリティ・エラーが原因で失敗した 2 つのバス・トランザクションを起動した場合は、このビットが IA32_MCi_STATUS 内でアサートされる（AERR がアサート）。このビットがアサートされている間は、別のエラーで上書きできない。
	45	UECC	未訂正の ECC エラーに対しては、訂正不可能 ECC エラービットが IA32_MCi_STATUS 内でアサートされる。このビットがアサートされている間は、ECC シンドローム・フィールドは上書きされない。
	46	CECC	訂正ずみの ECC エラーに対しては、訂正可能 ECC エラービットが IA32_MCi_STATUS 内でアサートされる。
	47-54	ECC シンドローム	エラーが訂正可能 / 訂正不可能 ECC エラーであり、しかも前の有効な ECC エラー・シンドロームが IA32_MCi_STATUS 内にログされていなかった場合は、IA32_MCi_STATUS の ECC シンドローム・フィールドに 8 ビット ECC シンドロームだけが入れられる。 IA32_MCi_STATUS 内の前の有効な ECC エラーは、アサートされている IA32_MCi_STATUS.bit45（訂正不可能なエラーが発生した）によって示される。将来の ECC エラー・シンドロームをログすることが可能になるように、ECC エラーを処理した後は、マシンチェック処理ソフトウェアが IA32_MCi_STATUS.bit45 をクリアしなければならない。
	55-56	予約済み	予約済み

E.2. ファミリ 0FH 固有のマシン・エラー・コードのデコーディング

プロセッサ・ファミリ 0FH によるマシン・エラー・コードのレポーティングも、IA32_MCi_STATUS から読み出された値に基づく（図 E-2. を参照）。

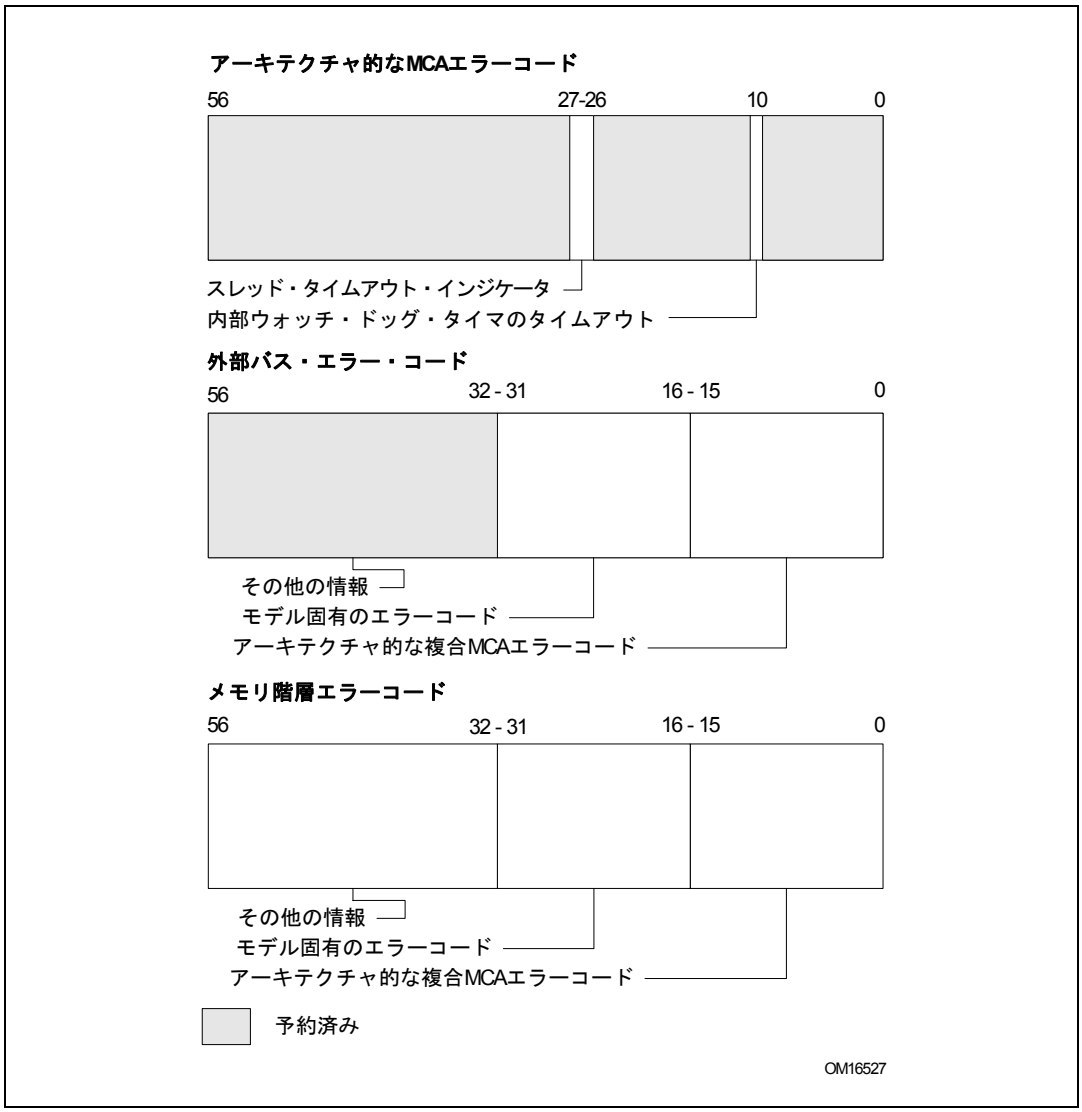


図 E-2. ファミリ 0FH の IA32_MCi_STATUS のエンコーディング

表 E-3. に、プロセッサ・ファミリ 0FH の内部ウォッチ・ドッグ・タイマのタイムアウト・マシン・チェックに対するエラー・コード・フィールドを解釈するための情報を示す。

表 E-3. ファミリ 0FH の内部ウォッチ・ドッグ・タイマ・エラーに対する IA32_MCi_STATUS のエンコーディング

	ビット番号	ビットの機能	ビットの記述
アーキテクチャ的な MCA エラーコード	0-15	0000010000000000	内部ウォッチ・ドッグ・タイマのタイムアウト ウォッチ・ドッグ・タイマのタイムアウトは、BINIT ドライバ回路がイネーブルの場合にのみ発生することに注意する。
モデル固有のエラーコード	16-25	予約済み	予約済み
	26-27	スレッド・タイムアウト・インジケータ（TT）	タイムアウトしたスレッドを示す。 01 ー スレッド 0 がタイムアウト 10 ー スレッド 1 がタイムアウト 11 ー 両方のスレッドがタイムアウト
	28-31	予約済み	予約済み
その他の情報	32-56	予約済み	予約済み

表 E-4. に、外部バス上で発生したエラーを解釈するための情報を示す。プロセッサ・ファミリ 0FH では外部バスエラーに複合 MCA コード形式が使用されていることに注意する。詳細については、第 14 章「マシン・チェック・アーキテクチャ」を参照のこと。

表 E-4. ファミリ 0FH の外部バスエラーに対する IA32_MCi_STATUS のエンコーディング

	ビット番号	ビットの機能	ビットの記述
アーキテクチャ的な複合 MCA エラーコード	0-1	メモリ階層レベル（LL）	メモリ階層レベル（LL）サブ・フィールドのデコーディングの詳細については、表 14-5. を参照のこと。
	2-3	メモリか I/O（II）	メモリまたは IO（II）サブ・フィールドのデコーディングの詳細については、表 14-7. を参照のこと。
	4-7	要求（RRRR）	要求（RRRR）サブ・フィールドのデコーディングの詳細については、表 14-6. を参照のこと。
	8	タイムアウト（T）	タイムアウト（T）サブ・フィールドのデコーディングの詳細については、表 14-7. を参照のこと。
	9-10	パーティシペーション（PP）	パーティシペーション（PP）サブ・フィールドのデコーディングの詳細については、表 14-7. を参照のこと。
	11-15	00001	バスエラーと相互接続エラー

表 E-4. ファミリ 0FH の外部バスエラーに対する IA32_MCi_STATUS のエンコーディング（続き）

モデル固有のエラーコード	16	FSB アドレスパリティ	アドレス・パリティ・エラー検出 1 = アドレス・パリティ・エラーを検出 0 = アドレス・パリティ・エラーがない
	17	応答ハード故障	応答時にハードウェア故障を検出
	18	応答パリティ	応答時にパリティエラーを検出
	19	PIC および FSB データパリティ	PIC または FSB のアクセスの際にデータパリティを検出
	20x	プロセッサ・シグニチャ = 00000F04H: 無効 PIC 要求 その他のすべてのプロセッサ : 予約済み	プロセッサ・シグニチャ = 00000F04H。無効 PIC 要求によるエラーを示す（WB メモリによって PIC スペースへのアクセスが行われた）。 1 = 無効 PIC 要求エラー 0 = 無効 PIC 要求エラーがない 予約済み
	21	パッド・ステート・マシン	P および N のデータストローブの相対タイミングを追跡するステートマシンが非同期になったか、グリッチが検出された。
	22	パッド・ストローブ・グリッチ	データ・ストローブ・グリッチ
	23	パッド・アドレス・グリッチ	アドレス・ストローブ・グリッチ
	24-31	予約済み	予約済み
その他の情報	32-56	予約済み	予約済み

表 E-5. に、メモリ階層内で発生したエラーを解釈するための情報を示す。

表 E-5. ファミリ 0FH のメモリ階層エラーに対する IA32_MCi_STATUS のエンコーディング

	ビット番号	ビットの機能	ビットの記述
アーキテクチャ的な複合 MCA エラーコード	0-1	メモリ階層レベル（LL）	メモリ階層レベル（LL）サブ・フィールドのデコーディングの詳細については、表 14-5. を参照のこと。
	2-3	トランザクション・タイプ（TT）	トランザクション・タイプ（TT）サブ・フィールドのデコーディングの詳細については、表 14-5. を参照のこと。
	4-7	要求（RRRR）	要求（RRRR）サブ・フィールドのデコーディングの詳細については、表 14-6. を参照のこと。
	8-15	00000001	メモリ階層エラー形式
モデル固有のエラーコード	16-17	タグ・エラー・コード	マシン・チェック・エラーのタグ・エラー・コード 00 = エラーが検出されない 01 = クリーンなラインのあるタグミスでのパリティエラー 10 = タグヒットでのパリティエラー / 複数のタグマッチ 11 = タグミスでのパリティエラー / 複数のタグマッチ
	18-19	データ・エラー・コード	マシン・チェック・エラーのデータ・エラー・コード 00 = エラーが検出されない 01 = 1 ビット・エラー 10 = クリーンなラインでの 2 ビット・エラー 11 = 変更されたラインでの 2 ビット・エラー
	20	L3 エラー	マシン・チェック・エラーが L3 で発生した場合にこのビットがセットされる（無効 PIC 要求エラーでは無視可能）。 1 = L3 エラー 0 = L2 エラー
	21	無効 PIC 要求	無効 PIC 要求によるエラーを示す（WB メモリによって PIC スペースへのアクセスが行われた）。 1 = 無効 PIC 要求エラー 0 = 無効 PIC 要求エラーがない
	22-31	予約済み	予約済み
その他の情報	32-39	8 ビット・エラー・カウント	リセット以降のエラー数を保持する。カウンタは、最初のエラー発生時に 0 から始まり、254 で飽和する。
	40-56	予約済み	予約済み

F

APIC バス・メッセージの形式

付録 F
APIC バス・メッセージの形式

F

この付録では、シリアルAPICバス上でメッセージを転送する際に使用されるメッセージ形式を説明する。ここで取り上げる情報は、インテル® Pentium® プロセッサおよびP6ファミリ・プロセッサにだけ関係する。

F.1. バス・メッセージの形式

ローカルAPICとI/O APICがシリアルAPICバス上で転送するメッセージには3つのタイプがある。つまりEOIメッセージ、ショート・メッセージ、非フォーカス最低優先度メッセージである。次に各タイプのメッセージの目的とその形式を示す。

F.2. EOI メッセージ

ローカルAPICはI/O APICに対して14サイクルのEOIメッセージを送り、レベルトリガ割り込みがプロセッサに受け入れられたことを知らせる。ただしこの割り込みは、ローカルAPICのEOIレジスタにソフトウェアが書き込みを行った結果生成されるものである。表F-1.に、1つのEOIメッセージ内の各サイクルを示す。

表 F-1. EOI メッセージ（14 サイクル）

サイクル	ビット 1	ビット 0	
1	1	1	11 = EOI
2	ArbID3	0	アービトレーション ID はビット 3 ～ 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	割り込みベクタ V7 ～ V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	サイクル 6 ～ 9 のチェックサム
11	0	0	
12	A	A	ステータス・サイクル 0
13	A1	A1	ステータス・サイクル 1
14	0	0	アイドル

サイクル6～9でチェックサムが計算される。これは2ビット（ビット1:ビット0）の論理データ値の累積合計である。つまり、最後を除くすべての加算のキャリーアウトが合計に加算される。どれかのAPICが計算したチェックサムが、サイクル10でバスに現れるチェックサムと異なる場合、そのAPICはエラーを表す信号として、サイクル12中のAPICバスにサイクル11をドライブする。この場合、各APICはメッセージを無視する。送信元のAPICは該当するエラー標識を受け取る（8.5.3.項「エラー処理」を参照）と、メッセージを再送する。ステータス・サイクルの定義は表F-4.に示す。

F.2.1. ショート・メッセージ

ショート・メッセージ（21サイクル）は、固定、NMI、SMI、INIT、スタートアップ、ExtINT、最低優先度フォーカス割り込みの伝送に使用される。表F-2.に、ショート・メッセージの各サイクルを示す。

表F-2. ショート・メッセージ（21サイクル）

サイクル	ビット1	ビット0	
1	0	1	01 = ノーマル
2	ArbID3	0	アービトレーション ID ビット3～0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = デスティネーション・モード
7	M1	M0	M2 ～ M0 = 伝達モード
8	L	TM	L = レベル、TM = トリガモード
9	V7	V6	V7 ～ V0 = 割り込みベクタ
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7 ～ D0 = デスティネーション
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	サイクル6～16のチェックサム
18	0	0	
19	A	A	ステータス・サイクル0
20	A1	A1	ステータス・サイクル1
21	0	0	アイドル

物理伝達モードを使用する場合、サイクル 15 と 16 は APIC ID を表現し、サイクル 13 と 14 は受信側には「関係なし」とみなされる。論理伝達モードを使用する場合、サイクル 13 ～ 16 は 8 ビットの論理デスティネーション・フィールドを表す。

「all-incl-self」と「all-excl-self」の簡略表記の場合は、物理伝達モードとアービトレーション優先度 15（D0:D3=1111）が使用される。メッセージを送信するエージェントだけは、これら 2 つの簡略表記を判別することが必要であり、内部情報に基づいてこれを判別する。

既存のフォーカス・プロセッサに最低優先度伝達を使用する場合、そのフォーカス・プロセッサはサイクル 19 中にサイクル 10 をドライブして自己を識別し、割り込みを受け入れる。これは、他の APIC に対してアービトレーションを終了せよという合図である。フォーカス・プロセッサが見つからなかった場合、オンザフライでショート・メッセージが延長され、非フォーカス最低優先度メッセージとなる。EOI メッセージの場合を除き、チェックサムまたは受け入れのエラー（8.5.3. 項「エラー処理」を参照）を生成するメッセージはサイクル 21 の後で終了することに注意する。

F.2.2. 非フォーカス最低優先度メッセージ

この 34 サイクルのメッセージ（表 F-3. を参照）は、フォーカス・プロセッサが存在しない場合に最低優先度伝達モードで使用される。サイクル 1 ～ 20 についてはショート・メッセージの場合と同じである。ステータス・サイクル（サイクル 19）中に（A:A）フラグの状態が 10B である場合は、フォーカス・プロセッサが識別されたことであり、ショート・メッセージ形式が使用される（表 F-2. を参照）。（A:A）フラグが 00B に設定されている場合、最低優先度アービトレーションが開始され、非フォーカス最低優先度メッセージが 34 サイクルで伝達される。ステータス・フラグのそれ以外の組み合わせについては、F.2.3. 項「APIC バス・ステータス・サイクル」を参照。

表 F-3. 非フォーカス最低優先度メッセージ（34 サイクル）

サイクル	ビット 1	ビット 0	
1	0	1	01 = ノーマル
2	ArbID3	0	アービトレーション ID ビット 3 ～ 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = デスティネーション・モード
7	M1	M0	M2 ～ M0 = 伝達モード
8	L	TM	L = レベル、TM = トリガモード
9	V7	V6	V7 ～ V0 = 割り込みベクタ
10	V5	V4	

表 F-3. 非フォーカス最低優先度メッセージ（34 サイクル）（続き）

サイクル	ビット 1	ビット 0	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7 ～ D0 = デスティネーション
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	サイクル 6 ～ 16 のチェックサム
18	0	0	
19	A	A	ステータス・サイクル 0
20	A1	A1	ステータス・サイクル 1
21	P7	0	P7 ～ P0 = 逆プロセッサ優先度
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	アービトレーション ID3 ～ 0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	ステータス・サイクル
34	0	0	アイドル

サイクル 21 ～ 28 は、最低優先度プロセッサを選ぶアービトレーションに使用される。アービトレーションに参加するプロセッサは、自己の逆プロセッサ優先度をバスにドライブする。空の割り込みスロットを持つローカル APIC だけが最低優先度アービトレーションに参加する。そのような APIC が存在しない場合、メッセージは拒否され、後でやり直す必要がある。

2 つ以上のプロセッサが同じ最低優先度を持つ場合は、サイクル 29 ～ 32 もアービトレーションに使用される。最低優先度伝達モードでは、サイクル 33 のすべてのエラーの組み合わせ（A2:A2）により、エラー・ステータス・レジスタの「受け入れエラー」ビットがセットされる（図 8-9. を参照）。アービトレーション優先度の更新はサイクル 20 で行われるので、サイクル 33 で検出されるエラーには影響されない。最低優先度

アービトレーションで勝ち残ったローカル APIC だけがサイクル 33 をドライブする。サイクル 33 でエラーが起こると、送信者はメッセージの再送を強制される。

F.2.3. APIC バス・ステータス・サイクル

APIC バス・メッセージ内部のサイクルの一部はステータス・サイクルである。ステータス・サイクルの間にステータス・フラグ（A:A）と（A1:A1）の検査が行われる。表 F-4. に、これらのステータス・フラグが現行の伝達モードとフォーカス・プロセッサの存否に応じてどのように解釈されるかを示す。

表 F-4. APIC バス・ステータス・サイクルの解釈

伝達モード	A ステータス	A1 ステータス	A2 ステータス	ArbID およびサイクル数の更新	メッセージの長さ	再試行
EOI	00: CS_OK	10: Accept	XX:	はい、13	14 サイクル	いいえ
	00: CS_OK	11: Retry	XX:	はい、13	14 サイクル	はい
	00: CS_OK	0X: Accept Error	XX:	いいえ	14 サイクル	はい
	11: CS_Error	XX:	XX:	いいえ	14 サイクル	はい
	10: Error	XX:	XX:	いいえ	14 サイクル	はい
	01: Error	XX:	XX:	いいえ	14 サイクル	はい
固定	00: CS_OK	10: Accept	XX:	はい、20	21 サイクル	いいえ
	00: CS_OK	11: Retry	XX:	はい、20	21 サイクル	はい
	00: CS_OK	0X: Accept Error	XX:	いいえ	21 サイクル	はい
	11: CS_Error	XX:	XX:	いいえ	21 サイクル	はい
	10: Error	XX:	XX:	いいえ	21 サイクル	はい
	01: Error	XX:	XX:	いいえ	21 サイクル	はい
NMI、SMI、INIT、ExtINT、スタートアップ	00: CS_OK	10: Accept	XX:	はい、20	21 サイクル	いいえ
	00: CS_OK	11: Retry	XX:	はい、20	21 サイクル	はい
	00: CS_OK	0X: Accept Error	XX:	いいえ	21 サイクル	はい
	11: CS_Error	XX:	XX:	いいえ	21 サイクル	はい
	10: Error	XX:	XX:	いいえ	21 サイクル	はい
	01: Error	XX:	XX:	いいえ	21 サイクル	はい

表 F-4. APIC バス・ステータス・サイクルの解釈（続き）

伝達モード	A ステータス	A1 ステータス	A2 ステータス	ArbID およびサイクル数の更新	メッセージの長さ	再試行
最低優先度	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	はい、20	34 サイクル	いいえ
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	はい、20	34 サイクル	はい
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	はい、20	34 サイクル	はい
	00: CS_OK, NoFocus	10: End and Retry	XX:	はい、20	34 サイクル	はい
	00: CS_OK, NoFocus	0X: Error	XX:	いいえ	34 サイクル	はい
	10: CS_OK, Focus	XX:	XX:	はい、20	34 サイクル	いいえ
	11: CS_Error	XX:	XX:	いいえ	21 サイクル	はい
	01: Error	XX:	XX:	いいえ	21 サイクル	はい

索引

索 引

記号・数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

あ

く

け

こ

さ

し

ふ

へ

ほ

ま

み

む

め

も

ゆ

よ

ら

り

れ

ろ

わ