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

中巻 A：
命令セット・リファレンス A-M

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、中巻 A：命令セット・リファレンス A-M』（資料番号 253666-013J）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：命令セット・リファレンス N-Z』（資料番号 253667-013J）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 4 巻のうちの 2 巻である。他の 2 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、下巻：システム・プログラミング・ガイド』（資料番号 253668-013J）

『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」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・サポート環境と IA-32 プロセッサの互換性に関する情報について説明している。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、インテル® Pentium® M 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 プロセッサとインテル 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、中巻 A、B：命令セット・リファレンス』の概要

本書は、次の内容で構成されている。

第 1 章 ― 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テーション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 ― 命令フォーマット。IA-32 の全命令が使用するマシンレベル命令のフォーマットについて説明し、許可されるプレフィックスのエンコーディング、オペランド識別子バイト（ModR/M バイト）、アドレッシング・モード指示子バイト（SIB バイト）、ディスプレースメント・バイトと即値バイトについても説明する。

第 3 章 ― 命令セット・リファレンス A-M。IA-32 命令を詳細に説明している。これには、演算アルゴリズムの説明、フラグの影響、オペランド・サイズとアドレスサイズ属性の効果、および発生しうる例外などを詳細に説明する。命令は、アルファベット順に記載している。汎用命令、x87 FPU 命令、MMX® 命令、SSE、SSE2、SSE3、システム命令の説明も含まれる。

第 4 章 ― 命令セット・リファレンス N-Z。第 3 章に続き、IA-32 命令について説明する。この章には、アルファベット順の命令の後半部分が記載されている。この章から、『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』となる。

付録 A ― オペコード・マップ。IA-32 命令セットのオペコード・マップを示す。

付録 B ― 命令フォーマットおよびエンコーディング。IA-32 命令の各フォームのバイナリ・エンコーディングを示す。

付録 C ― 機能的に同等のインテル® C/C++ コンパイラ組み込み関数。IA-32 MMX 命令、SSE、SSE2、SSE3 のそれぞれと機能的に同等のインテル® C/C++ コンパイラ組み込み関数およびアセンブリ・コードを示す。

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進数と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。IA-32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。

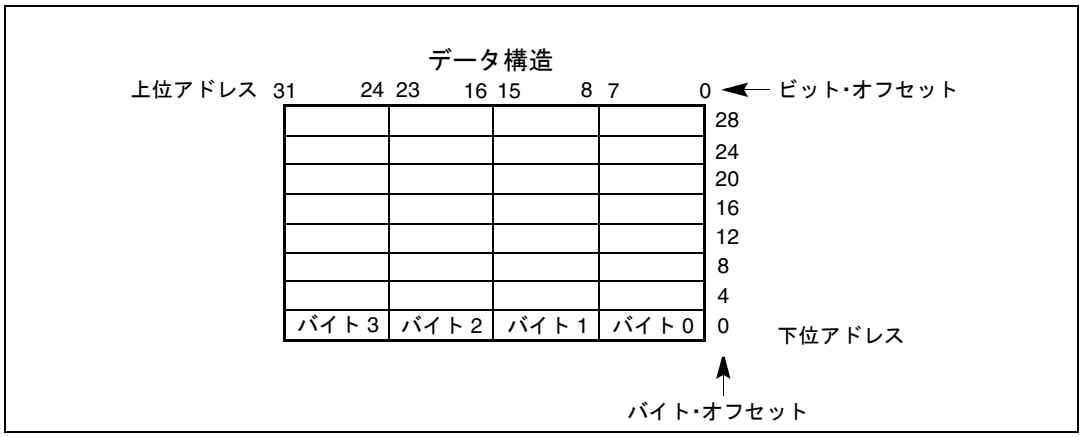

図 1-1. ビット・オーダとバイト・オーダ

1.3.2. 予約ビットとソフトウェア互換性

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、IA-32 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を作成すると、将来のプロセッサとの互換性を損なう危険がある。

1.3.3. 命令オペランド

命令をシンボルで表現する場合は、IA-32 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名である。
- 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、レジスタの予約名であるか、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 進数と 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/design/processor/index.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。

- 特定のインテル IA-32 プロセッサのデータシート
- 特定のインテル IA-32 プロセッサの仕様のアップデート
- 『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）
- 『IA-32 Intel® Architecture Optimization Reference Manual』（資料番号 248966）
- 『IA-32 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 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

命令フォーマット

第 2 章
命令フォーマット

2

本章では、すべてのIA-32プロセッサの命令フォーマットについて説明する。

2.1. 一般的命令フォーマット

IA-32の命令のコード化は、すべて図2-1.に示すフォーマットのサブセットである。命令は、任意指定の命令プリフィックス（順序は任意）、最大3つの基本オペコード・バイト、（必要な場合）ModR/M バイトおよび場合により SIB（スケール・インデックス・ベース）バイトからなるアドレス指定形式指定子、（必要な場合）ディスプレースメント、（必要な場合）即値データ・フィールドからなる。

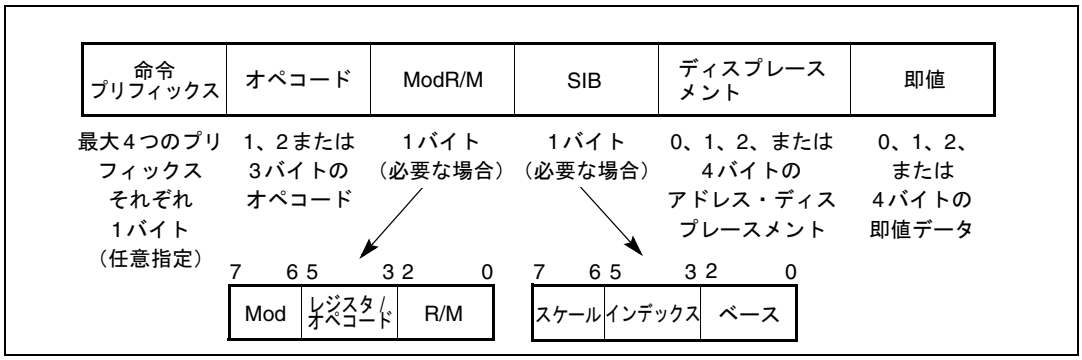

図 2-1. IA-32 プロセッサの命令フォーマット

2.2. 命令プリフィックスの概要

命令プリフィックスは、4 つのグループに分かれている。各グループは、一連の使用可能なプリフィックス・コードで構成される。各命令に対して、4 つのグループ（グループ 1、2、3、4）から 1 つずつプリフィックスを使用できる。プリフィックスの順序に制限はない。

注記

「予約済み」または「未定義」とされている機能や命令について、その特性に依存したり、それが存在しないものと見なしてはならない。予約済みの機能を誤って使用すると、インテル・プロセッサ上でそのコードを実行したとき、予測不可能な動作が起きたり、実行に失敗することがある。

- グループ 1
  - ロックおよびリピート・プリフィックス
    - F0H − LOCK
    - F2H − REPNE/REPNZ（ストリング命令に対してのみ使用。エスケープ・オペコード 0FH と共に使用する場合、このプリフィックスはいくつかの SIMD 命令では必須のプリフィックスとして扱われる）
    - F3H − REP または REPE/REPZ（ストリング命令に対してのみ使用。エスケープ・オペコード 0FH と共に使用する場合、このプリフィックスはいくつかの SIMD 命令では必須のプリフィックスとして扱われる）
- グループ 2
  - セグメント・オーバーライド・プリフィックス
    - 2EH − CSセグメント･オーバーライド(分岐命令に対する使用は予約されている)
    - 36H-SS セグメント・オーバーライド・プリフィックス（分岐命令に対する使用は予約されている）
    - 3EH − DS セグメント・オーバーライド・プリフィックス（分岐命令に対する使用は予約されている）
    - 26H − ES セグメント・オーバーライド・プリフィックス（分岐命令に対する使用は予約されている）
    - 64H − FS セグメント・オーバーライド・プリフィックス（分岐命令に対する使用は予約されている）
    - 65H − GS セグメント・オーバーライド・プリフィックス（分岐命令に対する使用は予約されている）

  - 分岐のヒント :
    - 2EH ― 分岐が成立しない（*Jcc* 命令に対してのみ使用）
    - 分岐が成立する（*Jcc* 命令に対してのみ使用）
- グループ 3
  - 66H ― オペランド・サイズ・オーバーライド・プリフィックス（エスケープ・オペコード 0FH と共に使用する場合、このプリフィックスはいくつかの SIMD 命令では必須のプリフィックスとして扱われる）
- グループ 4
  - 67H ― アドレス・サイズ・オーバーライド・プリフィックス

LOCK プリフィックス（F0H）を使用すると、強制的に操作を実行し、マルチプロセッサ環境において共用メモリが排他的に使用されるように指定できる。LOCK プリフィックスについては、第 3 章「命令セット・リファレンス A-M」の「LOCK—Assert LOCK# Signal Prefix」を参照のこと。

リピート・プリフィックス（F2H または F3H）を使用すると、ストリングの要素ごとに命令が繰り返される。リピート・プリフィックスは、ストリング命令（MOVS、CMPS、SCAS、LODS、STOS、INS、OUTS）に対してのみ使用できる。多くの SSE/SSE2/SSE3 では、これらのプリフィックスに続けて 0FH を使用すると、必須のプリフィックスとして扱われる。ストリング命令以外の IA-32 命令に対するリピート・プリフィックスや未定義のオペコードの使用は予約されている。予約済みの機能を使用すると、予測不可能な動作が起きることがある。

分岐ヒント・プリフィックス（2EH、3EH）を使用すると、プログラムは、分岐で最も使われる可能性の高いコード・パスについてのヒントをプロセッサに提供することができる。分岐ヒント・プリフィックスは、条件付き分岐命令（Jcc）に対してのみ使用できる。Jcc 以外の IA-32 命令に対する分岐ヒント・プリフィックスやその他の未定義のオペコードの使用は予約されている。予約済みの機能を使用すると、予測不可能な動作が起きる場合がある。

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と、プログラムにおいて、オペランド・サイズを 16 ビットと 32 ビットのいずれかに切り替えられる。どちらのサイズも、デフォルトに設定することが可能である。このプリフィックスを使用すると、デフォルトでないサイズが選択される。一部の SSE/SSE2/SSE3 では、66H に続けて 0FH を使用すると、必須のプリフィックスとして扱われる。
MMX/SSE/SSE2/SSE3 に対する 66H プリフィックスのこれ以外の使用は予約されている。予約済みの機能を使用すると、予測不可能な動作が起きる場合がある。

アドレス・サイズ・オーバーライド・プリフィックス（67H）を使用すると、プログラムにおいて、アドレス指定を 16 ビットと 32 ビットのいずれかに切り替えられる。どちらのアドレスサイズも、デフォルトに設定することが可能である。このプリフィッ

クスを使用すると、デフォルトでないサイズが選択される。命令オペランドがメモリに常駐していない場合のアドレス・サイズ・オーバーライド・プリフィックスやその他の未定義のオペコードの使用は予約されている。この場合にこのプリフィックスを使用すると、予期せぬ動作が起きることがある。

2.3. オペコード

基本オペコードは1、2、または3バイト長である。場合によって、ModR/Mバイト内の3ビットの追加オペコード・フィールドがコード化される。小さなコード化フィールドは基本オペコードの中で定義される。これらのフィールドは、操作の方向、ディスプレースメントのサイズ、レジスタのコード化、条件コード、または符号拡張を定義する。オペコードによって使用されるコード化のフィールドは、操作のクラスによって異なる。

汎用命令およびSIMD命令用の2バイト・オペコード形式は、以下の要素で構成される。

- エスケープ・オペコード・バイト 0FH（プライマリ・オペコード）およびオペコードの第2バイト
- 必須のプリフィックス（66F、F2、F3H）、エスケープ・オペコード・バイト、オペコードの第2バイト

例えば、CVTDQ2PD は、バイト・シーケンス F3 OF E6 で構成される。SSE/SSE2/SSE3 では、この表現の第1バイトは必須のプリフィックスになり、リピート・プリフィックスとは見なされない。すべての3バイト・オペコードは予約されている。

ModR/Mバイトは、3つのビット・フィールドで構成される（2.4.節を参照）。一部の命令では reg フィールドが拡張オペコード・フィールドとして扱われるが、それ以外に、ModR/Mバイト内の reg 以外の2つのビット・フィールドの特定のパターンもオペコード情報の表現に使用される。プライマリ・オペコード・バイトの未定義の表現の使用、ModR/Mバイトのオペコード拡張フィールド内での未定義の表現の使用、ModR/Mバイトのその他のビット・フィールド内での未定義の表現の使用は予約されている。有効なオペコード表現については、付録 A と付録 B で定義している。予約済みのオペコード表現を使用すると、予測不可能な動作が起きたり、実行に失敗することがある。

2.4. ModR/M および SIB バイト

メモリ内のオペランドを参照する多くの命令には、基本オペコードの次にアドレス指定形式指定子バイト（ModR/Mバイトという）がある。ModR/Mバイトには、以下の3つの情報フィールドがある。

- mod フィールドは、r/m フィールドと合わせて、存在可能な 32 通りの値、すなわち 8 個のレジスタと 24 個のアドレス指定モードを構成する。
- reg/opcode フィールドは、レジスタ番号 3 ビットの追加オペコード情報を指定する。reg/opecode フィールドの目的は、基本オペコードの中で指定される。
- r/m フィールドは、オペランドとしてレジスタを指定するか、または mod フィールドと組み合わせてアドレス指定モードをコード化できる。mod フィールドと組み合わせて r/m フィールドを使用して、いくつかの命令のオペコード情報を表すこともある。

ModR/M バイトの特定のコード化には、第 2 のアドレス指定バイト、すなわち SIB バイトが必要である。32 ビットの "ベース + インデックス" および "スケール + インデックス" の両形式のアドレス指定には、SIB バイトが必要である。SIB バイトには、以下のフィールドがある。

- scale フィールドはスケーリング・ファクタを指定する。
- index フィールドはインデックス・レジスタのレジスタ番号を指定する。
- base フィールドはベースレジスタのレジスタ番号を指定する。

ModR/M および SIB バイトのコード化については、2.6. 節「ModR/M および SIB バイトのアドレス指定モードのコード化」を参照のこと。

2.5. ディスプレースメント・バイトと即値バイト

一部のアドレス指定形式は、ModR/M バイト（存在する場合は SIB バイト）の直後にディスプレースメントを含む。ディスプレースメントは、必要な場合、1、2、または 4 バイトをもつことができる。

命令が即値オペランドを指定する場合、そのオペランドは常にディスプレースメント・バイトの次の位置を占める。即値オペランドは、1、2、または 4 バイトをもつことができる。

2.6. ModR/M および SIB バイトのアドレス指定モードのコード化

ModR/M および SIB バイトの値と対応するアドレス指定形式を表 2-1. から表 2-3. に示す。ModR/M バイトによって指定される 16 ビットのアドレス指定形式を表 2-1. に示し、32 ビットのアドレス指定形式を表 2-2. に示す。表 2-3. には、SIB バイトによって指定される 32 ビットのアドレス指定形式を示す。ModR/M バイトの中のレジスタ・フィールドとオペコード・フィールドが拡張オペコードを表している場合の有効な命令コード化は、付録 B に記載されている。

表 2-1. および表 2-2. では、最初の（「実効アドレス」という見出しの）欄には、ModR/M バイトのModおよびR/Mの両フィールドを使用して命令のオペランドに割り当てることができる、32 個の異なる実効アドレスを一覧している。最初の 24 個の実効アドレスは、メモリ・ロケーションの指定方法であり、残りの 8 個（Mod = 11B）は、汎用、MMX テクノロジ、XMM レジスタの指定方法である。例えば、最初のレジスタのコード化（Mod = 11B、R/M = 000B）は、汎用レジスタ EAX、AX、AL、MMX テクノロジ・レジスタ MM0、または XMM テクノロジ・レジスタ XMM0 を示す。これらのレジスタのどれが使用されるかは、オペコード・バイトとオペランド・サイズ属性によって決まる。これらは EAX レジスタ（32 ビット）か AX レジスタ（16 ビット）かを選択する。

表 2-1. および表 2-2. の 2 番目と 3 番目の欄は、最初の欄に示されている対応する実効アドレスを得るために必要なそれぞれ ModM/R バイトの Mod および R/M フィールドの 2 進コード化を示す。すなわち、Mod および R/M フィールドの存在可能な 32 通りすべての組み合わせが示されている。

表 2-1. および表 2-2. の両方に、3 ビットの Reg/Opcode フィールドの存在可能な 8 つの値が 10 進（上から 6 行目）および 2 進（上から 7 行目）で示してある。7 行目には "REG=" 見出しが付いており、これはそれら 3 ビットを使用して第 2 のオペランドのロケーションを指定することを表している。ただし、そのロケーションは汎用レジスタ、MMX テクノロジ・レジスタ、または XMM レジスタでなければならない。命令が第 2 のオペランドの指定を必要としない場合は、Reg/Opcode フィールドの 3 ビットをオペコードの拡張として使用でき、それは "/digit (Opcode)" 見出しの 6 行目によって表される。その上の 5 行目は、レジスタ番号に対応するバイト、ワード、ダブルワードの汎用レジスタ、MMX テクノロジ・レジスタ、XMM レジスタを示す。レジスタ番号の割り当ては、Mod フィールドのコードが 11B であるときの R/M フィールドの場合と同じである。R/M フィールド・レジスタ・オプションの場合と同様に、5 つの指定可能なレジスタのどれが使用されるかは、オペコード・バイトとオペランド・サイズ属性の組み合わせによって決まる。

表 2-1. および表 2-2. の本体（「ModR/M バイトの値（16 進表現）」という見出しの下）は、32 x 8 の配列になっており、ModR/M バイトの 256 個のすべての値を 16 進で示している。ビット 3、4、5 はバイトが存在する表の欄によって指定され、行はビット 0、1、2、6、7 を指定する。

表 2-1. ModR/M バイトによる 16 ビット・アドレス指定形式

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) /digit (Opcode) REG =			AL AX EAX MM0 XMM 0 0 000	CL CX ECX MM1 XMM 1 1 001	DL DX EDX MM2 XMM 2 2 010	BL BX EBX MM3 XMM 3 3 011	AH SP ESP MM4 XMM 4 4 100	CH BP EBP MM5 XMM 5 5 101	DH SI ESI MM6 XMM 6 6 110	BH DI EDI MM7 XMM 7 7 111
実効アドレス	**Mod**	**R/M**	**ModR/M バイトの値（16 進表現）**							
[BX+SI]	00	000	00	08	10	18	20	28	30	38
[BX+DI]		001	01	09	11	19	21	29	31	39
[BP+SI]		010	02	0A	12	1A	22	2A	32	3A
[BP+DI]		011	03	0B	13	1B	23	2B	33	3B
[SI]		100	04	0C	14	1C	24	2C	34	3C
[DI]		101	05	0D	15	1D	25	2D	35	3D
disp162		110	06	0E	16	1E	26	2E	36	3E
[BX]		111	07	0F	17	1F	27	2F	37	3F
[BX+SI]+disp83	01	000	40	48	50	58	60	68	70	78
[BX+DI]+disp8		001	41	49	51	59	61	69	71	79
[BP+SI]+disp8		010	42	4A	52	5A	62	6A	72	7A
[BP+DI]+disp8		011	43	4B	53	5B	63	6B	73	7B
[SI]+disp8		100	44	4C	54	5C	64	6C	74	7C
[DI]+disp8		101	45	4D	55	5D	65	6D	75	7D
[BP]+disp8		110	46	4E	56	5E	66	6E	76	7E
[BX]+disp8		111	47	4F	57	5F	67	6F	77	7F
[BX+SI]+disp16	10	000	80	88	90	98	A0	A8	B0	B8
[BX+DI]+disp16		001	81	89	91	99	A1	A9	B1	B9
[BP+SI]+disp16		010	82	8A	92	9A	A2	AA	B2	BA
[BP+DI]+disp16		011	83	8B	93	9B	A3	AB	B3	BB
[SI]+disp16		100	84	8C	94	9C	A4	AC	B4	BC
[DI]+disp16		101	85	8D	95	9D	A5	AD	B5	BD
[BP]+disp16		110	86	8E	96	9E	A6	AE	B6	BE
[BX]+disp16		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM1/XMM1		001	C1	C9	D1	D9	EQ	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AHMM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

注：

1. デフォルト・セグメント・レジスタは、BP インデックスを含む実効アドレスに対しては SS であり、それ以外の実効アドレスに対しては DS である。
2. disp16 は、ModR/M バイトの次の、インデックスに加算される 16 ビットのディスプレースメントを示す。
3. disp8 は、ModR/M バイトの次の、符号拡張され、インデックスに加算される 8 ビットのディスプレースメントを示す。

表 2-2. ModR/M バイトによる 32 ビット・アドレス指定形式

r8(/r)			AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)			AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
mm(/r)			MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
xmm(/r)			XMM	XMM	XMM	XMM	XMM	XMM	XMM	XMM
/digit (Opcode)			0	1	2	3	4	5	6	7
REG =			0	1	2	3	4	5	6	7
			000	001	010	011	100	101	110	111
実効アドレス	**Mod**	**R/M**	**ModR/M バイトの値（16 進表現）**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8
[ECX]+disp32		001	81	89	91	99	A1	A9	B1	B9
[EDX]+disp32		010	82	8A	92	9A	A2	AA	B2	BA
[EBX]+disp32		011	83	8B	93	9B	A3	AB	B3	BB
[--][--]+disp32		100	84	8C	94	9C	A4	AC	B4	BC
[EBP]+disp32		101	85	8D	95	9D	A5	AD	B5	BD
[ESI]+disp32		110	86	8E	96	9E	A6	AE	B6	BE
[EDI]+disp32		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH/MM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

注：

1. [--][--] は、ModR/M バイトの次に SIB があることを意味する。
2. disp32 は、ModR/M バイト（存在する場合は SIB バイト）の次の、インデックスに加算される 32 ビットのディスプレースメントを示す。
3. disp8 は、ModR/M バイト（存在する場合は SIB バイト）の次の、符号拡張され、インデックスに加算される 8 ビットのディスプレースメントを示す。

表 2-3. の構成は、本体が SIB バイトの存在可能な 256 個の値を 16 進で示す点以外は表 2-1.、表 2-2. と同じである。8 つの汎用レジスタのどれがベースとして使用されるかは、表の一番上に、ベース・フィールド（ビット 0、1、2）の 10 進と 2 進の対応値と合わせて示してある。各行は、スケーリング・ファクタ（ビット 6、7 によって決まる）と合わせて、どのレジスタがインデックスとして使用されるか（ビット 3、4、5 によって決まる）を示している。

表 2-3. SIB バイトによる 32 ビット・アドレス指定形式

r32 Base = Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
スケーリング結果 インデックス	**SS**	**インデックス**	**SIB バイトの値（16 進表現）**							
[EAX]	00	000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	89	9A	9B	9C	9D	9E	9F
none		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

注：

1. [*] は、MOD が 00B である場合はベースなしの disp32 を意味する。MOD が 00B でない場合は、[*] は disp8 または disp32+[EBP] を意味する。したがって、以下のアドレス指定モードが提供される。

MOD ビット	有効アドレス
00	[scaled index] + disp32
01	[scaled index] + disp8 + [EBP]
10	[scaled index] + disp32 + [EBP]

3

命令セット・リファレンス A-M

第 3 章
命令セット・リファレンス
A-M

3

本章では、汎用、x87 FPU、MMX® テクノロジ、SSE、SSE 2、SSE3 システム用の命令をカバーする IA-32 の命令セットを構成する各命令について説明する。本章の説明はアルファベット順（A-M）に整理してある。この説明は、IA-32 命令セットのバランスをとるため、第 4 章に続いている。『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 B』の第 4 章も参照のこと。

各命令について、オペコード、必要なオペランド、説明の各項からなる特定の記述形式により各オペランドの組み合わせを示している。その他に、各命令について、命令およびそのオペランド、操作に関する説明、EFLAGS レジスタのフラグに対する命令の影響の説明、および発生する可能性がある例外の要約も併記する。

3.1. 命令リファレンス・ページの読み方

本節では、本章の大部分を構成する命令リファレンス・ページのさまざまな説明項の記載内容について説明する。さらに、それらの項で使用される表記法と省略形式についても説明する。

3.1.1. 命令フォーマット

以下に、本章の各 IA-32 命令の説明に使用するフォーマットの例を示す。各命令の先頭には、使用説明を表すデータを記載している。

CMC—Complement Carry Flag

オペコード	命令	説明
F5	CMC	キャリー・フラグの補数をとる。

3.1.1.1. オペコード欄

「オペコード」欄には、各命令フォーマットに対して生成される完全なオブジェクト・コードを示す。可能な場合、コードは、メモリ内に現れるのと同じ順序で16進バイトとして示される。16進バイト以外のエントリの定義は、以下のとおりである。

- /digit － 0から7までの数字で、命令のModR/Mバイトがr/m（レジスタまたはメモリ）オペランドだけを使用することを示す。regフィールドには、命令のオペコードを拡張する数字が入っている。
- /r － 命令のModR/Mバイトに、レジスタ・オペランドとR/Mオペランドの両方があることを示す。
- cb、cw、cd、cp － オペコードの後に続く1バイト（cb）、2バイト（cw）、4バイト（cd）、または6バイト（cp）の値であり、コード・オフセット、コード・セグメント・レジスタの新しい値を指定することができる。
- ib、iw、id － オペコード、ModM/Rバイト、またはスケール・インデクッス・バイトの後に続く命令への1バイト（ib）、2バイト（iw）、または4バイト（id）の即値オペランドである。オペランドが符号付きかどうかは、オペコードによって決まる。すべてのワードおよびダブルワードが下位バイトから先に示される。
- +rb、+rw、+rd － 0から7までのレジスタコードであり、+符号の左側に現れる16進バイトに加算されて、単一のオペコード・バイトを構成する。すべてのレジスタコードを表3-1.に示す。
- +i － オペランドの1つがFPUレジスタスタックからのST(i)であるときに浮動小数点命令で使用される数値。数値i（0から7までの数値）は+符号の左側に現れる16進バイトに加算されて、単一のオペコード・バイトを構成する。

表3-1. +rb、+rw、および+rdに対応するレジスタのコード化

rb			rw			rd		
AL	=	0	AX	=	0	EAX	=	0
CL	=	1	CX	=	1	ECX	=	1
DL	=	2	DX	=	2	EDX	=	2
BL	=	3	BX	=	3	EBX	=	3
rb			**rw**			**rd**		
AH	=	4	SP	=	4	ESP	=	4
CH	=	5	BP	=	5	EBP	=	5
DH	=	6	SI	=	6	ESI	=	6
BH	=	7	DI	=	7	EDI	=	7

3.1.1.2. 命令欄

「命令」欄は、ASM386プログラムに現れる場合の命令文のシンタックスを示す。以下に、命令文内のオペランドを表現するために使用される記号の一覧を示す。

- rel8 ー 命令の終りの前の 128 バイトから命令の終りの後の 127 バイトまでの範囲の相対アドレス。
- rel16 および rel32 ー アセンブル結果の命令と同じコード・セグメント内の相対アドレス。記号 rel16 は、オペランド・サイズ属性が 16 ビットである命令に適用され、rel32 はオペランド・サイズ属性が 32 ビットである命令に適用される。
- ptr16:16 および ptr16:32 ー 一般的に命令のコード・セグメントとは異なるコード・セグメント内の far ポインタ。16:16 という表記法は、ポインタの値が 2 つの部分からなっていることを示す。コロンの左側の値は、16 ビットのセレクタ、すなわちコード・セグメント・レジスタに送られる値である。右側の値はデスティネーション・セグメント内のオフセットに対応する。記号 ptr16:16 は命令のオペランド・サイズ属性が 16 ビットである場合に使用され、記号 ptr16:32 はオペランド・サイズ属性が 32 ビットである場合に使用される。
- r8 ー 汎用バイトレジスタ AL、CL、DL、BL、AH、CH、DH、および BH の中の 1 つ。
- r16 ー 汎用ワードレジスタ AX、CX、DX、BX、SP、BP、SI、および DI の中の 1 つ。
- r32 ー 汎用ダブルワード・レジスタ EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI の中の 1 つ。
- imm8 ー 即値バイト値。記号 imm8 は -128 から +127 までの符号付き数値である。imm8 がワードまたはダブルワードのオペランドと結合される命令については、即値は符号拡張されてワードまたはダブルワードを構成する。その場合は、ワードの上位バイトは即値の最上位ビットで埋められる。
- imm16 ー オペランド・サイズ属性が 16 ビットである命令に使用される即値ワード値。これは -32,768 から +32,767 までの数値である。
- imm32 ー オペランド・サイズ属性が 32 ビットである命令に使用される即値ダブルワード値。これには -2,147,483,648 から +2,147,483,647 までの数値を使用できる。
- r/m8 ー 汎用バイトレジスタ（AL、BL、CL、DL、AH、BH、CH、DH）か、メモリからのバイトかの内容であるバイト・オペランド。
- r/m16 ー オペランド・サイズ属性が 16 ビットである命令に使用される汎用ワード・レジスタ・オペランドまたはメモリ・オペランド。汎用ワードレジスタは、AX、BX、CX、DX、SP、BP、SI、DI である。メモリの内容は、実効アドレスの計算によって与えられるアドレスにある。

- r/m32 ― オペランド・サイズ属性が 32 ビットである命令に使用される汎用ダブルワード・レジスタ・オペランドまたはメモリ・オペランド。汎用ダブルワード・レジスタは、EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI である。メモリの内容は、実効アドレスの計算によって与えられるアドレスにある。
- m ― 16 または 32 ビットのメモリ・オペランド。
- m8 ― 通常、変数名または配列名として表されるが、DS:(E)SI または ES:(E)DI レジスタによってアドレス指定されるバイト・メモリ・オペランド。これはストリング命令および XLAT 命令に対してのみ使用される。
- m16 ― 通常、変数名または配列名として表されるが、DS:(E)SI または ES:(E)DI レジスタによってアドレス指定されるワード・メモリ・オペランド。これはストリング命令に対してのみ使用される。
- m32 ― 通常、変数名または配列名として表されるが、DS:(E)SI または ES:(E)DI レジスタによってアドレス指定されるダブルワード・メモリ・オペランド。これはストリング命令に対してのみ使用される。
- m64 ― クワッドワード・メモリ・オペランド。これは CMPXCHG8B 命令に対してのみ使用される。
- m128 ― ダブル・クワッドワード・メモリ ・オペランド。これは SSE、SSE2、SSE3 に対してのみ使用される。
- m16:16、m16:32 ― 2 つの数値からなる far ポインタを内容とするメモリ・オペランド。コロンの左側の数値はポインタのセグメント・セレクタに対応し、右側の数値はポインタのオフセットに対応する。
- m16&32、m16&16、m32&32 ― サイズがアンパーサンド記号の左側と右側に示されるデータ項目の対からなるメモリ・オペランド。すべてのメモリアドレス指定モードが可能である。m16&16 および m32&32 オペランドは BOUND 命令で使用され、配列インデックスの上限および下限を内容とするオペランドを提供する。m16&32 オペランドは LIDT および LGDT で使用され、対応する GDTR および IDTR レジスタの limit フィールドにロードするワード、およびベース・フィールドにロードするダブルワードを提供する。
- moffs8、moffs16、moffs32 ― MOV 命令の一部のバリエーションによって使用されるバイト、ワード、またはダブルワード型のシンプルメモリ変数（メモリ・オフセット）。実際のアドレスは、セグメント・ベースからのシンプル・オフセットによって与えられる。命令には ModR/M バイトは使用されない。moffs の次の数値はセグメントのサイズであり、これは命令のアドレスサイズ属性によって決まる。
- Sreg ― セグメント・レジスタ。セグメント・レジスタのビット割り当ては、ES=0、CS=1、SS=2、DS=3、FS=4、GS=5。
- m32fp、m64fp、m80fp ― それぞれ 単精度、倍精度、拡張倍精度浮動小数点数のメモリ・オペランドである。これらのシンボルにより、x87 FPU 浮動小数点命令のオペランドとして使用される浮動小数点の値が指定される。

- m16int、m32int、m64int ― それぞれ ワード、ダブルワード、クワッドワード整数のメモリ・オペランドである。これらのシンボルにより、x87 FPU 整数命令のオペランドとして使用される整数が指定される。
- ST または ST(0) ― FPU レジスタスタックの一番上の要素。
- ST(i) ― FPU レジスタスタックの一番上から i 番目の要素（$i \leftarrow 0 \sim 7$）。
- mm ― MMX® テクノロジ・レジスタ。64 ビット MMX テクノロジ・レジスタ MM0 から MM7 まで。
- mm/m32 ― MMX テクノロジ・レジスタの下位 32 ビットまたは 32 ビットのメモリ・オペランド。64 ビットの MMX テクノロジ・レジスタは MM0 から MM7 まで。メモリの内容は、実効アドレス計算によって与えられるアドレスにある。
- mm/m64 ― MMX テクノロジ・レジスタまたは 64 ビットのメモリ・オペランド。64 ビット MMX テクノロジ・レジスタは MM0 から MM7 まで。メモリの内容は実効アドレス計算によって与えられるアドレスにある。
- xmm ― XMM レジスタ。128 ビットの XMM レジスタは、XMM0 から XMM7 まで。
- xmm/m32 ― XMM レジスタまたは 32 ビットのメモリ・オペランド。128 ビットの XMM レジスタは、XMM0 から XMM7 まで。メモリの内容は、実効アドレス計算によって与えられるアドレスにある。
- xmm/m64 ― XMM レジスタまたは 64 ビットのメモリ・オペランド。128 ビットの SIMD 浮動小数点レジスタは、XMM0 から XMM7 まで。メモリの内容は、実効アドレス計算によって与えられるアドレスにある。
- xmm/m128 ― XMM レジスタまたは 128 ビットのメモリ・オペランド。128 ビットの XMM レジスタは、XMM0 から XMM7 まで。メモリの内容は、実効アドレス計算によって与えられるアドレスにある。

3.1.1.3. 説明欄

「命令」欄の次の「説明」欄では、各種の命令フォーマットについて簡単に説明する。命令のフォーマットの下の「説明」および「操作」の項には、命令の動作の詳細が記載してある。

3.1.1.4. 説明

「説明」の項では、命令の目的と必要なオペランドについて説明する。さらに、命令のフラグに対する影響についても説明する。

3.1.2. 操作

「操作」の項には、命令のアルゴリズムに関する説明を記載する（疑似コードで記述）。この疑似コードは、Algol または Pascal 言語に似た表記法を使用している。アルゴリズムは、以下の要素からなっている。

- コメントは記号の対 "(*" および "*)" で囲まれる。
- 複合文は、それぞれ、if 文については IF、THEN、ELSE、FI、do 文については DO と OD、または case 文については CASE ... OF と ESAC などのキーワードで囲まれる。
- レジスタ名は暗黙にレジスタの内容を意味する。ブラケットで囲まれたレジスタ名は、暗黙に、アドレスがそのレジスタにストアされているロケーションの内容を意味する。例えば、ES:[DI] は、ES セグメント相対アドレスがレジスタ DI にストアされているロケーションの内容を示す。[SI] は、レジスタ SI にストアされている、SI レジスタのデフォルト・セグメント（DS）またはオーバーライドされたセグメントの相対アドレスの内容を示す。
- 例えば (E)SI のように、汎用レジスタ名の中で "E" がかっこで囲まれているのは、現在のアドレスサイズ属性が 16 である場合はオフセットが SI レジスタから読み込まれ、アドレスサイズ属性が 32 である場合は ESI レジスタから読み込まれることを示す。
- ブラケットはメモリ・オペランドに対しても使用され、その場合は、メモリ・ロケーションの内容がセグメント相対オフセットであることを意味する。例えば、[SRC] はソース・オペランドの内容がセグメント相対オフセットであることを示す。
- A ← B; は、B の値が A に代入されることを示す。
- 記号 =、≠、≧、≦は 2 つの値の比較に使用される関係演算子であり、それぞれ、「等しい」、「等しくない」、「より大きいか等しい」、「より小さいか等しい」を意味する。例えば、関係式 A = B は、A の値が B に等しい場合は TRUE（真）であり、そうでない場合は FALSE（偽）である。
- 式 "<< COUNT" と ">> COUNT" は、デスティネーション・オペランドがカウント・オペランドが示すビット数だけそれぞれ左または右にシフトされることを示す。

以下の識別子が、アルゴリズムの記述に使用される。

- OperandSize と AddressSize ― OperandSize 識別子は、16 ビットか 32 ビットの命令のオペランド・サイズ属性を表す。AddressSize 識別子は、16 ビットか 32 ビットのアドレスサイズ属性を表す。例えば、以下の疑似コードは、オペランド・サイズ属性が使用される CMPS 命令のフォーマットに依存することを示している。

```
IF instruction = CMPSW
    THEN OperandSize ← 16;
    ELSE
        IF instruction = CMPSD
            THEN OperandSize ← 32;
        FI;
FI;
```

  これらの属性の決定方法の一般的ガイドラインについては、『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 3 章の「オペランド・サイズ属性とアドレスサイズ属性」を参照のこと。

- StackAddrSize ― 命令に関連するスタックのアドレスサイズ属性を表す。この属性の値は 16 または 32 ビットである（『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、上巻』の第 6 章の「スタックアクセスにおけるアドレスサイズ属性」を参照のこと）。
- SRC ― ソース・オペランドを表す。
- DEST ― デスティネーション・オペランドを表す。

以下の関数が、アルゴリズムの記述に使用される。

- ZeroExtend（値）― 命令のオペランド・サイズ属性に合わせたゼロ拡張値を返す。例えば、オペランド・サイズ属性が 32 である場合は、-10 のバイト値をゼロ拡張すると、そのバイトは F6H からダブルワード値 000000F6H に変換される。ZeroEztend 関数に渡された値とオペランド・サイズ属性とが同じサイズである場合は、ZeroExtend はその値を変えずにそのまま返す。
- SignExtend（値）― 命令のオペランド・サイズ属性に合わせた符号拡張値を返す。例えば、オペランド・サイズ属性が 32 である場合は、内容が値 -10 であるバイトを符号拡張すると、そのバイトは F6H からダブルワード値 FFFFFFF6H に変換される。SignEztend 関数に渡された値とオペランド・サイズ属性とが同じサイズである場合は、SignExtend はその値を変えずにそのまま返す。
- SaturateSignedWordToSignedByte ― 符号付き 16 ビット値を符号付き 8 ビット値に変換する。符号付き 16 ビット値が -128 より小さい場合は、その値は飽和値 -128（80H）で表される。127 より大きい場合は、飽和値 127（7FH）で表される。

- SaturateSignedDWordToSignedWord ー 符号付き 32 ビット値を符号付き 16 ビット値に変換する。符号付き 32 ビット値が -32768 より小さい場合は、その値は飽和値 -32768（8000H）で表される。32767 より大きい場合は、飽和値 32767（7FFFH）で表される。
- SaturateSignedWordToUnsignedByte ー 符号付き 16 ビット値を符号なし 8 ビット値に変換する。符号付き 16 ビット値が 0 より小さい場合は、その値は飽和値 0（00H）で表される。255 より大きい場合は、飽和値 255（FFH）で表される。
- SaturateToSignedByte ー 演算の結果を符号付き 8 ビット値として表す。結果が -128 より小さい場合は、その値は飽和値 -128（80H）で表される。127 より大きい場合は、飽和値 127（7FH）で表される。
- SaturateToSignedWord ー 演算の結果を符号付き 16 ビット値として表す。結果が -32768 より小さい場合は、その値は飽和値 -32768（8000H）で表される。32767 より大きい場合は、飽和値 32767（7FFFH）で表される。
- SaturateToUnsignedByte ー 演算の結果を符号なし 8 ビット値として表す。結果が 0 より小さい場合は、その値は飽和値 0（00H）で表される。255 より大きい場合は、飽和値 255（FFH）で表される。
- SaturateToUnsignedWord ー 演算の結果を符号なし 16 ビット値として表す。結果が 0 より小さい場合は、その値は飽和値 0（00H）で表される。65535 より大きい場合は、飽和値 65535（FFFFH）で表される。
- RoundTowardsZero () ー 直近の整数値にゼロへの丸めを実行されたオペランドを返す。
- LowOrderWord(DEST * SRC) ー ワード・オペランドにワード・オペランドを掛け、ダブルワードの結果の下位ワードをデスティネーション・オペランドにストアする。
- HighOrderWord(DEST * SRC) ー ワード・オペランドにワード・オペランドを掛け、ダブルワードの結果の上位ワードをデスティネーション・オペランドにストアする。
- Push（値）ー 値をスタックにプッシュする。プッシュされるバイト数は命令のオペランド・サイズ属性によって決まる。プッシュ操作の詳細については、本章の「PUSH—Push Word or Doubleword onto the Stack」の「操作」の項を参照のこと。
- Pop() ー 一番上のスタックから値を削除し、その値を返す。文 EAX ← Pop(); は、一番上のスタックから 32 ビット値を EAX に代入する。Pop は、オペランド・サイズ属性に応じて、ワードかダブルワードを返す。ポップ操作の詳細については、本章の「POP—Pop a Value from the Stack」の「操作」の項を参照のこと。
- PopRegisterStack ー FPU の ST(0) レジスタを空としてマークし、FPU レジスタ・スタック・ポインタ（TOP）を 1 だけインクリメントする。
- Switch-Tasks ー タスクスイッチを行う。

- Bit（BitBase, BitOffset）ー メモリまたはレジスタ内の一連のビットであるビット・ストリング内の特定ビットの値を返す。ビットには、レジスタ内またはメモリバイト内で下位から上位に番号が割り振られている。ベース・オペランドがレジスタである場合は、存在可能なオフセットの範囲は 0 ～ 31 である。このオフセットは、指定されたレジスタ内のビットのアドレスを指定する。例として、図 3-1. に関数 Bit[EAX,21] の図解を示す。

  BitBase がメモリアドレスである場合は、BitOffset の存在可能な範囲は -2G ビットから 2G ビットまでである。アドレス指定されるビットには、アドレス（BitBase + (BitOffset DIV 8)）のバイト内の番号（Offset MOD 8）が与えられる。ただし、DIV は負の無限大に向かう丸めを伴う符号付き除算であり、MOD は正の数値を返す。図 3-2. に、この操作の図解を示す。

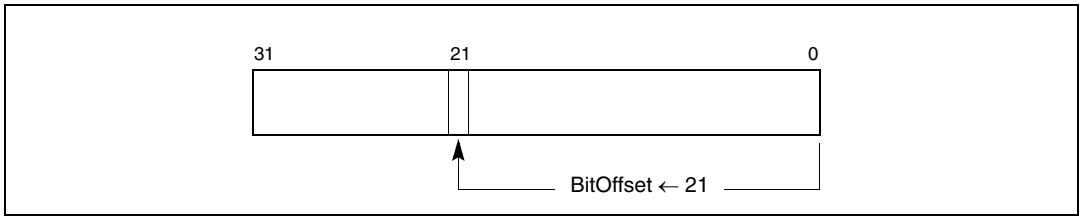

図 3-1. BIT[EAX,21] のビット・オフセット

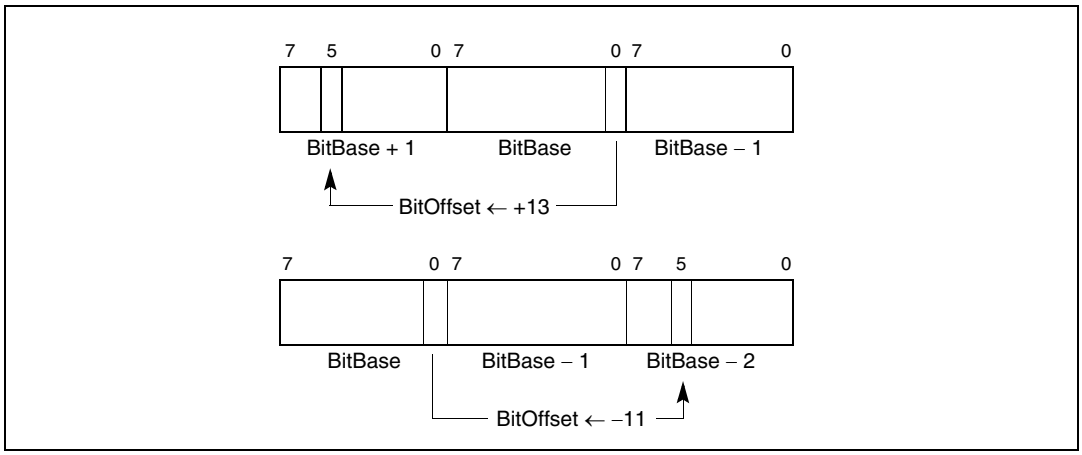

図 3-2. メモリ・ビットのインデックス操作

3.1.3. インテル® C/C++ コンパイラ組み込み関数

インテル® C/C++コンパイラ組み込み関数は、特殊なC/C++コーディングの拡張によって、ハードウェア・レジスタの代わりに、C 関数呼び出しおよびC 変数の構文を使用できるようにする。組み込み関数の使用によって、プログラマは、レジスタの管理やアセンブリ言語のプログラミング作業から解放される。さらに、コンパイラが命令のスケジューリングを最適化するため、実行可能プログラムの速度がアップする。

ここでは、組み込み関数のAPI、MMX® テクノロジ組み込み関数、およびSIMD 浮動小数点組み込み関数について説明する。各組み込み関数およびそれに対応する命令の説明が示される。対応する命令を持たない組み込み関数が存在する可能性があるため、コンパイラのマニュアルを参照して、サポートしているすべての組み込み関数のリストを確認することを強くお勧めする。

『Intel C/C++ Compiler User's Guide With Support for the Streaming SIMD Extensions 2』（資料番号 718195-2001）を参照のこと。組み込み関数の使用についての詳細は、本書の付録C「機能的に同等のインテル® C/C++ コンパイラ組み込み関数」を参照のこと。

3.1.3.1. 組み込み関数の API

MMX® テクノロジ組み込み関数およびSSE、SSE2、SSE3 の組み込み関数を使用したコーディングのメリットは、ハードウェア・レジスタの代わりに、C 関数呼び出しおよびC 変数の構文を使用できることである。これにより、プログラマは、レジスタの管理やアセンブリ言語のプログラミング作業から解放される。さらに、コンパイラが命令のスケジューリングを最適化するため、実行可能プログラムの速度がアップする。新しい命令セットのそれぞれの計算命令およびデータ操作命令に対して、その命令を直接実装したCの組み込み関数が存在する。組み込み関数を使用して、アルゴリズムの基礎となる手法（命令の選択）をプログラマが指定し、命令のスケジューリングとレジスタの割り当てはコンパイラに任せられる。

3.1.3.2. MMX® テクノロジ組み込み関数

MMX® テクノロジ組み込み関数は、MMXテクノロジ・レジスタの内容を表す、新しい __m64データ型に基づいている。値は、バイト、16 ビット整数、32 ビット値、または64 ビット・オブジェクトで指定できる。ただし、__m64 データ型は基本的なANSI Cデータ型ではないため、次の制限を守って使用しなければならない。

- __m64 データは、必ず代入の左辺側、戻り値、またはパラメータとして使用すること。その他の算術式（"+"、">>" など）に __m64データを使用できない。
- __m64 オブジェクトは、バイト要素にアクセスする共用体や構造体など、複数のオブジェクトの集合体として使用すること。1つの __m64オブジェクトのアドレスを使用できる。

- __m64 データは、本書および『Intel C/C++ Compiler User's Guide With Support for the Streaming SIMD Extensions 2』（資料番号 718195-2001）で説明する MMX テクノロジ組み込み関数でのみ使用すること。組み込み関数の使用についての詳細は、本書の付録 C「機能的に同等のインテル® C/C++ コンパイラ組み込み関数」を参照のこと。

3.1.3.3. SSE、SSE2、SSE3 の組み込み関数

SSE、SSE2、SSE3 のすべての組み込み関数は、インテル® Pentium® III プロセッサ、インテル® Pentium® プロセッサ、インテル® Xeon™ プロセッサの XMM レジスタを使用する。これらの組み込み関数は、__m128、__m128d、__m128i の 3 つのデータ型をサポートしている。

- __m128 データ型は、SSE 組み込み関数が使用する XMM レジスタの内容を表現する。これは、4 つのパックド単精度浮動小数点値または 1 つのスカラ単精度浮動小数点値である。
- __m128d データ型は、2 つのパックド倍精度浮動小数点値、または 1 つのスカラ倍精度浮動小数点値を格納する。
- __m128i データ型は、16 個のバイト整数値、8 個のワード整数値、4 個のダブルワード整数値、または 2 個のクワッドワード整数値を格納する。

コンパイラは、__m128、__m128d、__m128i のローカル・データとグローバル・データのアライメントを、スタック上の 16 バイト境界に合わせる。integer 型、float 型、または double 型配列のアライメントを合わせるには、『Intel C/C++ Compiler User's Guide With Support for the Streaming SIMD Extensions 2』（資料番号 718195-2001）の説明にしたがって、declspec 文を使用する。

__m128、__m128d、__m128i データ型は基本的な ANSI C データ型ではないため、次の制限を守って使用しなければならない。

- __m128、__m128d、__m128i データは、必ず代入の左辺側、戻り値、またはパラメータとして使用すること。その他の算術式（"+"、">>" など）に __m64 データを使用することはできない。
- リテラルを持つ __m128、__m128d、__m128i データを初期化してはならない。128 ビットの定数を表現する方法はない。
- __m128、__m128d、__m128i オブジェクトは、（例えば、浮動小数点要素にアクセスする）共用体や構造体など、複数のオブジェクトの集合体として使用すること。1 つの __m128 オブジェクトのアドレスを使用できる。
- __m128、__m128d、__m128i データは、本書で説明する組み込み関数でのみ使用すること。組み込み関数の使用についての詳細は、本書の付録 C「機能的に同等のインテル® C/C++ コンパイラ組み込み関数」を参照。

コンパイラは、ローカルな __m128、__m128d、__m128i データのアライメントをスタックの 16 バイト境界に合わせる。グローバルな __m128 データのアライメントも、16 バイト境界に合わせられる（浮動小数点データの配列のアライメントを合わせるには、後で説明する declspec 指定子を使用できる）。新しい命令セットは、スカラ・データを処理する場合も、パックド・データの場合と同じ方法で SIMD 浮動小数点レジスタを処理する。したがって、スカラ・データを表す __m32 データ型は存在しない。スカラ演算には、__m128 オブジェクトと「スカラ」形式の組み込み関数を使用する必要がある。コンパイラとプロセッサは、32 ビットのメモリ参照を使用してスカラ演算を実行する。

サフィックス ps および ss は、「パックド単精度」および「スカラ単精度」演算を示す。パックド浮動小数点値は、[z, y, x, w] のように、右から左の順番で表される。最下位ワード（一番右）がスカラ演算に使用される。この方式がメモリへのストアにどのように反映されるかを理解するには、次の例を考えればよい。

次の演算は、

```
float a[4] ← { 1.0, 2.0, 3.0, 4.0 };
__m128 t ← _mm_load_ps(a);
```

次の演算と同じ結果を生じる。

```
__m128 t ← _mm_set_ps(4.0, 3.0, 2.0, 1.0);
```

つまり、次のようになる。

```
t ← [ 4.0, 3.0, 2.0, 1.0 ]
```

この場合、「スカラ」要素は 1.0 である。

いくつかの組み込み関数は、2 つ以上の命令を使って実行されるため、「複合組み込み関数」と呼ばれる。組み込み関数を使用してプログラムを作成する際は、SSE、SSE2、SSE3 と MMX® テクノロジが提供するハードウェア機能をよく理解する必要がある。

次の重要事項を忘れないこと。

- _mm_loadr_ps と _mm_cmpgt_ss のように、命令セットで直接サポートしていない組み込み関数もある。これらの組み込み関数はプログラミングに便利であるが、展開されるコード・サイズや命令のコストを考慮に入れる必要がある。
- __m128 オブジェクトとしてロードまたはストアされるデータは、通常は 16 バイト・アライメントでなければならない。
- いくつかの組み込み関数は、SIMD 浮動小数点命令の性質上、引き数が即値、すなわち定数の整数（リテラル）であることを要求する。

- 2つのNaN（Not a Number）引き数を処理する算術演算の結果は未定義である。したがって、複数のNaN引き数を使用する浮動小数点演算は、それに対応するアセンブリ言語の命令の予想される動作と一致しないことがある。

各組み込み関数とその使用方法についての詳細は、『Intel C/C++ Compiler User's Guide With Support for the Streaming SIMD Extensions 2』（資料番号718195-2001）を参照されたい。組み込み関数の使用についての詳細は、本書の付録C「機能的に同等のインテル® C/C++コンパイラ組み込み関数」を参照のこと。

3.1.4. 影響を受けるフラグ

「影響を受けるフラグ」の項には、命令の影響を受けるEFLAGSレジスタのフラグを示す。フラグはクリアされたときは0に等しく、セットされたときは1に等しい。算術演算および論理演算命令は、通常、一定の方法で値をステータス・フラグに代入する（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、上巻』の付録A「EFLAGSクロス・リファレンス」を参照のこと）。「操作」の項では、非従来型の代入について説明している。**未定義**として示してあるフラグの値は、不確定な方法で変更される場合がある。記載されていないフラグは命令によって変更されない。

3.1.5. 影響を受けるFPUフラグ

浮動小数点命令については、「影響を受けるFPUフラグ」という項があり、そこで、各命令がFPUステータス・ワードの4つの条件コードフラグにどのように影響を与えるかを説明している。

3.1.6. 保護モード例外

「保護モード例外」の項には、命令を保護モードで実行したときに発生する可能性がある例外およびそれらの例外の理由を示す。各例外は、番号記号（#）とその後に続く2文字からなるニーモニック、および括弧で囲まれたオプションのエラーコードで表される。例えば、#GP(0)はエラーコード0を伴う一般保護例外を示す。表3-2.に、各2文字のニーモニックと割り込みベクタ番号および例外名との対応を示す。これらの例外の詳細については、『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、下巻』の第5章「割り込みと例外の処理」を参照のこと。

アプリケーション・プログラマは、例外が発生した場合は、該当するオペレーティング・システムに付属のマニュアルを調べて対処すること。

表 3-2. IA-32 の一般例外

ベクタ番号	名称	発生源	保護モード	実アドレスモード	仮想 8086 モード
0	#DE – 除算エラー	DIV および IDIV 命令	○	○	○
1	#DB – デバッグ	任意のコードまたはデータ参照	○	○	○
3	#BP – ブレークポイント	INT 3 命令	○	○	○
4	#OF – オーバーフロー	INTO 命令	○	○	○
5	#BR – BOUND 範囲外	BOUND 命令	○	○	○
6	#UD – 無効オペコード（未定義オペコード）	UD2 命令または予約オペコード	○	○	○
7	#NM – デバイスなし（算術演算コプロセッサなし）	浮動小数点または WAIT/FWAIT 命令	○	○	○
8	#DF – ダブルフォルト	例外、NMI、または INTR を発生する可能性がある任意の命令	○	○	○
10	#TS – 無効 TSS	タスクスイッチまたは TSS アクセス	○	予約	○
11	#NP – セグメントなし	セグメント・レジスタのロードまたはシステム・セグメントのアクセス	○	予約	○
12	#SS – スタック・セグメント・フォルト	スタック操作および SS レジスタロード	○	○	○
13	#GP – 一般保護 *	任意のメモリ参照およびその他の保護チェック	○	○	○
14	#PF – ページフォルト	任意のメモリ参照	○	予約	○
16	#MF – 浮動小数点エラー（算術演算フォルト）	浮動小数点または WAIT/FWAIT 命令	○	○	○
17	#AC – アライメント・チェック	任意のメモリデータ参照	○	予約	○
18	#MC – マシンチェック	マシン・チェック・エラーはモデルに依存	○	○	○
19	#XF – SIMD 浮動小数点数値エラー	SSE、SSE2、SSE3 浮動小数点命令	○	○	○

注：

* 実アドレスモードにおいて、ベクタ 13 はセグメント・オーバーラン例外である。

3.1.7. 実アドレスモード例外

「実アドレスモード例外」の項には、命令を実アドレスモードで実行したときに発生する可能性がある例外を示す（表 3-2. を参照）。

3.1.8. 仮想 8086 モード例外

「仮想 8086 モード例外」の項には、命令を仮想 8086 モードで実行したときに発生する可能性がある例外を示す（表 3-2. を参照）。

3.1.9. 浮動小数点例外

「浮動小数点例外」の項には、x87 FPU の浮動小数点命令を実行したときに発生する可能性がある例外を示す。これらの例外条件はすべて浮動小数点エラー例外（#MF、ベクタ番号 16）を発生させる。表 3-3. に、1 または 2 文字のニーモニックと例外名との対応を示す。これらの例外の詳細については、『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「浮動小数点例外条件」の説明を参照のこと。

表 3-3. x87 FPU の浮動小数点例外

ニーモニック	名称	発生源
#IS #IA	無効浮動小数点演算 : - 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 - 無効算術演算	- x87 FPU スタック・オーバーフローまたは x87 FPU スタック・アンダーフロー - 無効 FPU 算術演算
#Z	0 での浮動小数点除算	ゼロによる除算
#D	浮動小数点演算デノーマル・オペランド	デノーマル・オペランド
#O	浮動小数点数値オーバーフロー	結果がオーバーフロー
#U	浮動小数点数値アンダーフロー	結果がアンダーフロー
#P	浮動小数点不正確結果（精度）	不正確結果（精度）

3.1.10. SIMD 浮動小数点例外

「SIMD 浮動小数点例外」の項では、SSE、SSE2、SSE3 浮動小数点命令を実行したときに発生する可能性がある例外を示す。これらの例外条件は、すべて SIMD 浮動小数点エラー例外（#XF、ベクタ番号 19）を発生させる。表 3-4. に、ニーモニックと例外名の対応を示す。これらの例外についての詳細は、『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、上巻』の第 11 章の「SSE、SSE2、SSE3 の例外」を参照のこと。

表 3-4. SIMD 浮動小数点例外

ニーモニック	名称	発生源
#I	浮動小数点無効演算	無効算術演算またはソース・オペランド
#Z	浮動小数点ゼロによる除算	ゼロによる除算
#D	浮動小数点デノーマル・オペランド	デノーマル・オペランド
#O	浮動小数点数値オーバーフロー	結果がオーバーフロー
#U	浮動小数点数値アンダーフロー	結果がアンダーフロー
#P	浮動小数点不正確結果	不正確結果（精度）

3.2. 命令リファレンス

本章の以降では、IA-32 の各命令の詳細について説明する。

AAA—ASCII Adjust After Addition

オペコード	命令	説明
37	AAA	加算後に AL を ASCII 調整する。

説明

2 つのアンパック BCD 値の和を調整して、アンパック BCD 結果を作成する。AL レジスタは、この命令の暗黙のソース兼デスティネーション・オペランドである。AAA 命令は、ADD 命令の次に実行したときに限り有効である。ADD 命令は、2 つのアンパック BCD 値を加算（2 進加算）し、バイトの結果を AL レジスタにストアする。次に、AAA 命令が AL レジスタの内容を調整して、正しい 1 ケタのアンパック BCD 結果にまとめる。

加算によって 10 進キャリーが生じた場合は、AH レジスタが 1 インクリメントされ、CF および AF フラグがセットされる。10 進キャリーが生じなかった場合は、CF および AF フラグはクリアされ、AH レジスタは変わらない。どちらの場合も、AL レジスタのビット 4 から 7 までは 0 にクリアされる。

操作

```
IF ((AL AND 0FH) > 9) OR (AF ← 1)
    THEN
        AL ← AL + 6;
        AH ← AH + 1;
        AF ← 1;
        CF ← 1;
    ELSE
        AF ← 0;
        CF ← 0;
FI;
AL ← AL AND 0FH;
```

影響を受けるフラグ

調整によって 10 進キャリーが生じた場合は、AF および CF フラグが 1 にセットされる。10 進キャリーが生じなかった場合は、両フラグが 0 にセットされる。OF、SF、ZF、PF フラグは未定義。

例外（すべての操作モード）

なし。

AAD—ASCII Adjust AX Before Division

オペコード	命令	説明
D5 0A	AAD	除算前に AX を ASCII 調整する。
D5 *ib*	（ニーモニックなし）	除算前に AX を基数 *imm8* に調整する。

説明

調整結果に対して除算を行った場合に正しいアンパック BCD 値が生じるように、2 ケタのアンパック BCD 数（AL レジスタの下位桁と AH レジスタの上位桁）を調整する。AAD 命令は、AX レジスタの調整結果の値をアンパック BCD 値で除算（2 進除算）する DIV 命令の直前に実行したときに限り有効である。

AAD 命令は、AL レジスタの値を（AL + (10 * AH)）に設定し、次に AH レジスタを 00H にクリアする。したがって、AX レジスタの値は、AL および AH レジスタの元のパックされていない 2 ケタの（基数 10 の）数に等価な 2 進値に等しい。

この命令の一般的なバージョンでは、imm8 バイトを選択した基数（例えば、8 進数では 08H、10 進数では 0CH、基数 12 の数では 0CH）に設定することにより、パックされていない任意の基数の 2 ケタの数を調整できる（以下の「操作」の項を参照）。AAD ニーモニックは、すべてのアセンブラが ASCII（基数 10 の）値を調整するものとして解釈する。他の基数の値を調整するには、この命令はマシンコード（D5 imm8）でハンド・コーディングしなければならない。

操作

```
tempAL ← AL;
tempAH ← AH;
AL ← (tempAL + (tempAH * imm8)) AND FFH; (* imm8 is set to 0AH for the AAD
mnemonic *)
AH ← 0
```

即値（*imm8*）は命令の第 2 バイトから与えられる。

影響を受けるフラグ

SF、ZF、および PF フラグが AL レジスタの 2 進値の結果にしたがってセットされる。OF、AF、CF フラグは未定義。

例外（すべての操作モード）

なし。

AAM—ASCII Adjust AX After Multiply

オペコード	命令	説明
D4 0A	AAM	乗算後に AX を ASCII 調整する。
D4 *ib*	（ニーモニックなし）	乗算後に AX を基数 *imm8* に調整する。

説明

2 つのアンパック BCD 値間の乗算結果を調整して、1 対のアンパック（基数 10 の）BCD 値を作成する。AX レジスタは、この命令の暗黙のソース兼デスティネーション・オペランドである。AAM 命令は、2 つのアンパック BCD 値を乗算（2 進乗算）し、ワードの結果を AX レジスタにストアする MUL 命令の次に実行したときに限り有効である。次に、AAM 命令が AX レジスタの内容を調整して、2 ケタのアンパック（基数 10 の）BCD 結果にまとめる。

この命令の一般的なバージョンでは、AX の内容を調整して任意の基数の 2 ケタのアンパック数を作成することができる（下記の「操作」の項を参照）。ここで、imm8 バイトは選択した基数（例えば、8 進数では 08H、10 進数では 0AH、または基数 12 の数では 0CH）に設定される。AAM ニーモニックは、すべてのアセンブラが ASCII（基数 10 の）値に調整するものとして解釈する。他の基数の値の調整するには、この命令をマシンコード（D4 imm8）でハンド・コーディングしなければならない。

操作

```
tempAL ← AL;
AH ← tempAL / imm8; (* imm8 is set to 0AH for the AAD mnemonic *)
AL ← tempAL MOD imm8;
```

即値（*imm8*）は命令の第 2 バイトから与えられる。

影響を受けるフラグ

SF、ZF、および PF フラグが AL レジスタの 2 進値の結果にしたがってセットされる。OF、AF、CF フラグは未定義。

例外（すべての操作モード）

0AH のデフォルトの即値に対しては例外はない。ただし、即値として 0 を使用した場合は、#DE（除算エラー）例外が発生する。

AAS—ASCII Adjust AL After Subtraction

オペコード	命令	説明
3F	AAS	減算後に AL を ASCII 調整する。

説明

2 つのアンパック BCD 値間の減算結果を調整して、アンパック BCD の結果を作成する。AL レジスタは、この命令の暗黙のソース兼デスティネーション・オペランドである。AAS 命令は、1 つのアンパック BCD 値からもう 1 つのアンパック BCD 値を減算（2 進減算）し、バイトの結果を AL レジスタにストアする SUB 命令の次に実行したときに限り有効である。次に、AAS 命令が AL レジスタの内容を調整して、正しい 1 ケタのアンパック BCD の結果にまとめる。

減算によって 10 進キャリーが生じた場合は、AH レジスタが 1 デクリメントされ、CF および AF フラグがセットされる。10 進キャリーが生じなかった場合は、CF および AF フラグはクリアされ、AH レジスタは変わらない。どちらの場合も、AL レジスタの上位ニブルは 0 に設定されたままである。

操作

```
IF ((AL AND 0FH) > 9) OR (AF = 1)
THEN
    AL ← AL – 6;
    AH ← AH – 1;
    AF ← 1;
    CF ← 1;
ELSE
    CF ← 0;
    AF ← 0;
FI;
AL ← AL AND 0FH;
```

影響を受けるフラグ

10 進ボローが生じた場合は、AF および CF フラグが 1 にセットされ、10 進ボローが生じなかった場合は両フラグが 0 にセットされる。OF、SF、ZF、PF フラグは未定義。

例外（すべての操作モード）

なし。

ADC—Add with Carry

オペコード	命令	説明
14 *ib*	ADC AL, *imm8*	キャリーを加えて *imm8* を AL に加算する。
15 *iw*	ADC AX, *imm16*	キャリーを加えて *imm16* を AX に加算する。
15 *id*	ADC EAX, *imm32*	キャリーを加えて *imm32* を EAX に加算する。
80 /2 *ib*	ADC *r/m8*, *imm8*	キャリーを加えて *imm8* を *r/m8* に加算する。
81 /2 *iw*	ADC *r/m16*, *imm16*	キャリーを加えて *imm16* を *r/m16* に加算する。
81 /2 *id*	ADC *r/m32*, *imm32*	CF を加えて *imm32* を *r/m32* に加算する。
83 /2 *ib*	ADC *r/m16*, *imm8*	CF を加えて符号拡張 *imm8* を *r/m16* に加算する。
83 /2 *ib*	ADC *r/m32*, *imm8*	CF を加えて符号拡張 *imm8* を *r/m32* に加算する。
10 /*r*	ADC *r/m8*, *r8*	キャリーを加えてバイトレジスタを *r/m8* に加算する。
11 /*r*	ADC *r/m16*, *r16*	キャリーを加えて *r16* を *r/m16* に加算する。
11 /*r*	ADC *r/m32*, *r32*	CF を加えて *r32* を *r/m32* に加算する。
12 /*r*	ADC *r8*, *r/m8*	キャリーを加えて *r/m8* をバイトレジスタに加算する。
13 /*r*	ADC *r16*, *r/m16*	キャリーを加えて *r/m16* を *r16* に加算する。
13 /*r*	ADC *r32*, *r/m32*	CF を加えて *r/m32* を *r32* に加算する。

説明

デスティネーション・オペランド（第1オペランド）、ソース・オペランド（第2オペランド）、およびキャリー（CF）を加算し、結果をデスティネーション・オペランドにストアする。デスティネーション・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ソース・オペランドには、即値、レジスタ、またはメモリ・ロケーションを使用できる。（ただし、1つの命令に2つのメモリ・オペランドを使用することはできない。）CF フラグの状態は、前の加算からのキャリーを表す。オペランドとして即値を使用した場合は、その即値はデスティネーション・オペランド・フォーマットの長さに符号拡張される。

ADC 命令は符号付きと符号なしのオペランドを区別しない。その代わりに、プロセッサが両データ型について結果を評価し、OF および CF フラグをセットして、それぞれ符号付きまたは符号なしの結果のキャリーを示す。SF フラグが符号付き結果の符号を示す。

ADC 命令は、通常、複数バイトまたは複数ワード間の加算の一部として実行され、そのときに ADD 命令の後に ADC 命令が実行される。

この命令を LOCK プリフィックスと共に使用すると、アトミックに命令を実行させることができる。

ADC—Add with Carry（続き）

操作

DEST ← DEST + SRC + CF;

影響を受けるフラグ

OF、SF、ZF、AF、CF、PF フラグが結果にしたがってセットされる。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリをアクセスしたが、その内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

ADD—Add

オペコード	命令	説明
04 *ib*	ADD AL, *imm8*	*imm8* を AL に加算する。
05 *iw*	ADD AX, *imm16*	*imm16* を AX に加算する。
05 *id*	ADD EAX, *imm32*	*imm32* を EAX に加算する。
80 /0 *ib*	ADD *r/m8*, *imm8*	*imm8* を *r/m8* に加算する。
81 /0 *iw*	ADD *r/m16*, *imm16*	*imm16* を *r/m16* に加算する。
81 /0 *id*	ADD *r/m32*, *imm32*	*imm32* を *r/m32* に加算する。
83 /0 *ib*	ADD *r/m16*, *imm8*	符号拡張 *imm8* を *rm16* に加算する。
83 /0 *ib*	ADD *r/m32*, *imm8*	符号拡張 *imm8* を *rm32* に加算する。
00 /*r*	ADD *r/m8*, *r8*	*r8* を *r/m*8 に加算する。
01 /*r*	ADD *r/m16*, *r16*	*r16* を *r/m16* に加算する。
01 /*r*	ADD *r/m32*, *r32*	*r32* を *r/m32* に加算する。
02 /*r*	ADD *r8*, *r/m8*	*r/m8* を *r8* に加算する。
03 /*r*	ADD *r16*, *r/m16*	*r/m16* を *r16* に加算する。
03 /*r*	ADD *r32*, *r/m32*	*r/m32* を *r32* に加算する。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）を加算し、結果をデスティネーション・オペランドにストアする。デスティネーション・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ソース・オペランドには、即値、レジスタ、またはメモリ・ロケーションを使用できる。（ただし、1 つの命令に 2 つのメモリ・オペランドを使用することはできない。）CF フラグの状態は、前の加算からのキャリーを表す。オペランドとして即値を使用した場合は、その即値はデスティネーション・オペランド・フォーマットの長さに符号拡張される。

ADD 命令は、整数加算を実行する。ADD は、符号付き整数オペランドおよび符号なし整数オペランドの両方の結果を評価する。また、符号付き結果または符号なし結果のキャリーを示す CF フラグおよびオーバーフローを示す OF フラグを設定する。SF フラグは、符号付き結果の符号を表す。

この命令を LOCK プリフィックスと共に使用すると、アトミックに命令を実行できる。

操作

```
DEST ← DEST + SRC;
```

影響を受けるフラグ

OF、SF、ZF、AF、CF、PF フラグが結果にしたがってセットされる。

ADD—Add（続き）

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリをアクセスしたが、その内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

ADDPD—Add Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 58 /r	ADDPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド倍精度浮動小数点値を加算して、結果を *xmm1* に格納する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 2 つのパック ド倍精度浮動小数点値の SIMD 加算を実行し、結果のパック ド倍精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。倍精度浮動小数点値の SIMD 演算の図は、『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 11-3. を参照のこと。

操作

```
DEST[63-0] ← DEST[63-0] + SRC[63-0];
DEST[127-64] ← DEST[127-64] + SRC[127-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDPD __m128d _mm_add_pd (m128d a, m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

ADDPD—Add Packed Double-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ADDPS—Add Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 58 /r	ADDPS *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* のパックド単精度浮動小数点値を加算して、結果を *xmm1* レジスタにストアする。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 4 つのパックド単精度浮動小数点値の SIMD 加算を実行し、結果のパックド単精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。単精度浮動小数点値の SIMD 演算の図は、『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 10-5. を参照のこと。

操作

```
DEST[31-0] ← DEST[31-0] + SRC[31-0];
DEST[63-32] ← DEST[63-32] + SRC[63-32];
DEST[95-64] ← DEST[95-64] + SRC[95-64];
DEST[127-96] ← DEST[127-96] + SRC[127-96];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDPS __m128 _mm_add_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。 セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

ADDPS—Add Packed Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ADDSD—Add Scalar Double-Precision Floating-Point Values

オペコード	命令	説明
F2 0F 58 /r	ADDSD *xmm1*, *xmm2/m64*	*xmm2/m64* と *xmm1* の下位の倍精度浮動小数点値を加算して、結果を *xmm1* に格納する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の下位の倍精度浮動小数点値同士を加算し、結果の倍精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位クワッドワードは変更されない。倍精度浮動小数点値のスカラ演算の図は、『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 11-4. を参照のこと。

操作

```
DEST[63-0]  ← DEST[63-0] + SRC[63-0];
* DEST[127-64] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDSD　　__m128d _mm_add_sd (m128d a, m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

ADDSD—Add Scalar Double-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

ADDSS—Add Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F 58 /r	ADDSS *xmm1*, *xmm2/m32*	*xmm1*および*xmm2/m32*の下位の単精度浮動小数点値を加算して、結果を *xmm1* レジスタにストアする。

説明

ソース・オペランド（第2オペランド）とデスティネーション・オペランド（第1オペランド）の下位の単精度浮動小数点値同士を加算し、結果の単精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位 3 つのダブルワードは変更されない。単精度浮動小数点値のスカラ演算の図は、『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 10-6. を参照のこと。

操作

```
DEST[31-0] ← DEST[31-0] + SRC[31-0];
* DEST[127-32] remain unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDSS __m128 _mm_add_ss(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

ADDSS—Add Scalar Single-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

ADDSUBPD—Packed Double-FP Add/Subtract

オペコード	命令	説明
66, 0F, D0, /r	ADDSUBPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド倍精度浮動小数点値を加算 / 減算する。

説明

ソース・オペランドとデスティネーション・オペランドの上位クワッドワードの倍精度浮動小数点値を加算し、結果をデスティネーション・オペランドの上位クワッドワードに格納する。

デスティネーション・オペランドの下位クワッドワードの倍精度浮動小数点値からソース・オペランドの下位クワッドワードの倍精度浮動小数点値を減算し、結果をデスティネーション・オペランドの下位クワッドワードに格納する。

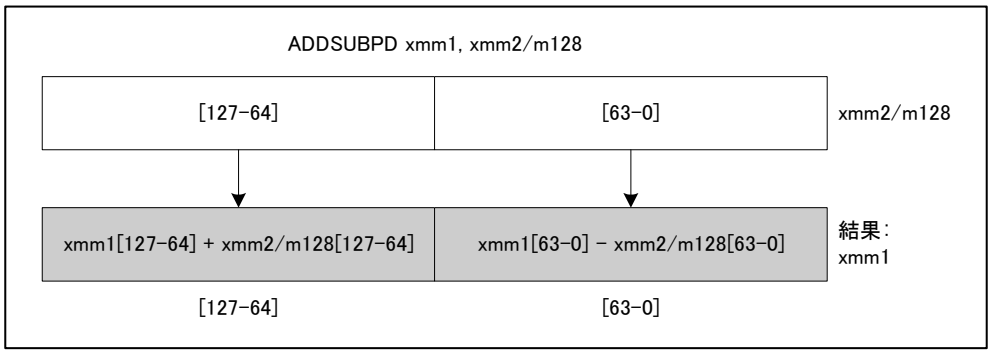

図 3-3. ADDSUBPD: Packed Double-FP Add/Subtract

操作

```
xmm1[63-0]   = xmm1[63-0]   – xmm2/m128[63-0];
xmm1[127-64] = xmm1[127-64] + xmm2/m128[127-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDSUBPD　　__m128d _mm_addsub_pd(__m128d a, __m128d b)

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

ADDSUBPD—Packed Double-FP Add/Subtract（続き）

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。
#UD	CR0.EM = 1 の場合。
	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。
	CR4.OSFXSR（ビット 9）= 0 の場合。
	CPUID.SSE3（ECX ビット 0）= 0 の場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。
#UD	CR0.EM = 1 の場合。
	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。
	CR4.OSFXSR（ビット 9）= 0 の場合。
	CPUID.SSE3（ECX ビット 0）= 0 の場合。

ADDSUBPD—Packed Double-FP Add/Subtract（続き）

仮想 8086 モード例外

GP(0)	オペランドの一部が0～FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。
#UD	CR0.EM = 1 の場合。
	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。
	CR4.OSFXSR（ビット 9）= 0 の場合。
	CPUID.SSE3（ECX ビット 0）= 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

ADDSUBPS—Packed Single-FP Add/Subtract

オペコード	命令	説明
F2, 0F, D0, /r	ADDSUBPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド単精度浮動小数点値を加算 / 減算する。

説明

ソース・オペランドの奇数番号の単精度浮動小数点値と、それらに対応するデスティネーション・オペランドの単精度浮動小数点値を加算し、結果をデスティネーション・オペランドの奇数番号の値に格納する。

デスティネーション・オペランドの偶数番号の単精度浮動小数点値から、それらに対応するソース・オペランドの単精度浮動小数点値を減算し、結果をデスティネーション・オペランドの偶数番号の値に格納する。

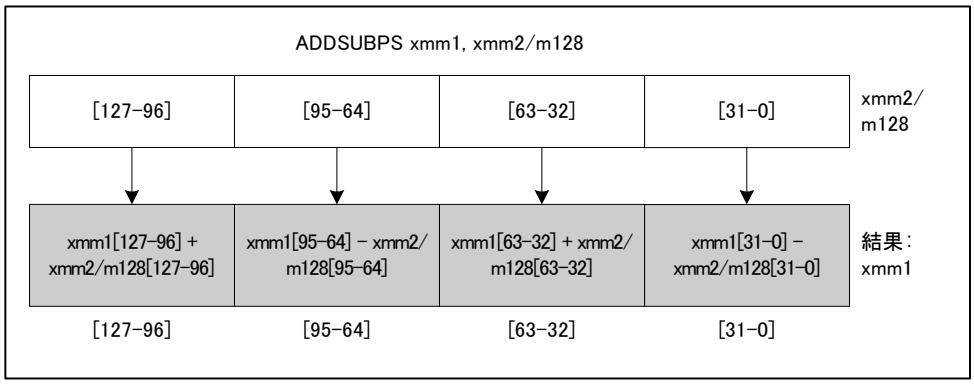

図 3-4. ADDSUBPS: Packed Single-FP Add/Subtract

操作

```
xmm1[31-0]   = xmm1[31-0]   – xmm2/m128[31-0];
xmm1[63-32]  = xmm1[63-32]  + xmm2/m128[63-32];
xmm1[95-64]  = xmm1[95-64]  – xmm2/m128[95-64];
xmm1[127-96] = xmm1[127-96] + xmm2/m128[127-96];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ADDSUBPS　　__m128 _mm_addsub_ps(__m128 a, __m128 b)

ADDSUBPS—Packed Single-FP Add/Subtract（続き）

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、またはGSセグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが16バイトに合っていない場合。
#SS(0)	SSセグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0のTSがセットされた場合。
#XM	マスクされていないストリーミングSIMD浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。
#UD	CR0.EM = 1の場合。
	マスクされていないストリーミングSIMD浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。
	CR4.OSFXSR（ビット9）= 0の場合。
	CPUID.SSE3（ECXビット0）= 0の場合。

実アドレスモード例外

GP(0)	オペランドの一部が0～FFFFHの実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが16バイトに合っていない場合。
#NM	CR0のTSがセットされた場合。
#XM	マスクされていないストリーミングSIMD浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。

ADDSUBPS—Packed Single-FP Add/Subtract（続き）

#UD	CR0.EM = 1 の場合。 マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。 CR4.OSFXSR（ビット 9）= 0 の場合。 CPUID.SSE3（ECX ビット 0）= 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が 0～FFFFH の実効アドレス空間の範囲外の場合。 セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 1）の場合。
#UD	CR0.EM = 1 の場合。 マスクされていないストリーミング SIMD 浮動小数点数値例外（CR4.OSXMMEXCPT = 0）の場合。 CR4.OSFXSR（ビット 9）= 0 の場合。 CPUID.SSE3（ECX ビット 0）= 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

AND—Logical AND

オペコード	命令	説明
24 *ib*	AND AL, *imm8*	AL と *imm8* との AND をとる。
25 *iw*	AND AX, *imm16*	AX と *imm16* との AND をとる。
25 *id*	AND EAX, *imm32*	EAX と *imm32* との AND をとる。
80 /4 *ib*	AND *r/m8*, *imm8*	*r/m8* と *imm8* との AND をとる。
81 /4 *iw*	AND *r/m16*, *imm16*	*r/m16* と *imm16* との AND をとる。
81 /4 *id*	AND *r/m32*, *imm32*	*r/m32* と *imm32* との AND をとる。
83 /4 *ib*	AND *r/m16*, *imm8*	*r/m16* と *imm8*（符号拡張）との AND をとる。
83 /4 *ib*	AND *r/m32*, *imm8*	*r/m32* と *imm8*（符号拡張）との AND をとる。
20 /*r*	AND *r/m8*, *r8*	*r/m8* と *r8* との AND をとる。
21 /*r*	AND *r/m16*, *r16*	*r/m16* と *r16* との AND をとる。
21 /*r*	AND *r/m32*, *r32*	*r/m32* と *r32* との AND をとる。
22 /*r*	AND *r8*, *r/m8*	*r8* と *r/m8* との AND をとる。
23 /*r*	AND *r16*, *r/m16*	*r16* と *r/m16* との AND をとる。
23 /*r*	AND *r32*, *r/m32*	*r32* と *r/m32* との AND をとる。

説明

デスティネーション（第1）オペランドとソース（第2）オペランドとの間のビット単位のAND（論理積）演算を実行し、結果をデスティネーション・オペランドにストアする。ソース・オペランドには、即値、レジスタ、またはメモリ・ロケーションを使用できる。デスティネーション・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。（ただし、1つの命令に2つのメモリ・オペランドを使用することはできない。）その結果の各ビットは、第1オペランドと第2オペランドの対応ビットが共に1の場合は1にセットされ、そうでない場合は0にセットされる。

この命令をLOCKプリフィックスと共に使用すると、アトミックに命令を実行させることができる。

操作

DEST ← DEST AND SRC;

影響を受けるフラグ

OF および CF フラグがクリアされ、SF、ZF、PF フラグが結果にしたがってセットされる。AF フラグの状態は未定義。

AND—Logical AND（続き）

保護モード例外

#GP(0)	デスティネーション・オペランドの指示先が書き込み不可能なセグメントの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

ANDPD—Bitwise Logical AND of Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 54 /r	ANDPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の AND（論理積）演算を実行する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値の間でビット単位の AND（論理積）演算を実行し、結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

操作

```
DEST[127-0] ← DEST[127-0] BitwiseAND SRC[127-0];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ANDPD __m128d _mm_and_pd(__m128d a, __m128d b)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

ANDPD—Bitwise Logical AND of Packed Double-Precision Floating-Point Values（続き）

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ANDPS—Bitwise Logical AND of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 54 /r	ANDPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の AND（論理積）演算を実行する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）内の 4 つのパックド単精度浮動小数点値の間でビット単位の AND（論理積）演算を実行し、結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

操作

```
DEST[127-0] ← DEST[127-0] BitwiseAND SRC[127-0];
```

同等のインテル® C/C++ コンパイラ組み込み関数

ANDPS __m128 _mm_and_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

ANDPS—Bitwise Logical AND of Packed Single-Precision Floating-Point Values（続き）

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ANDNPD—Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 55 /r	ADDPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の AND NOT（否定論理積）演算を実行する。

説明

デスティネーション・オペランド（第 1 オペランド）内の 2 つのパックド倍精度浮動小数点値の各ビットを反転して、反転された中間結果とソース・オペランド（第 2 オペランド）内の 2 つのパックド倍精度浮動小数点値の間でビット単位の AND（論理積）演算を実行し、結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

操作

DEST[127-0] ← (NOT(DEST[127-0])) BitwiseAND (SRC[127-0]);

同等のインテル® C/C++ コンパイラ組み込み関数

ANDNPD __m128d _mm_andnot_pd(__m128d a, __m128d b)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

ANDNPD—Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values（続き）

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ANDNPS—Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 55 /r	ANDNPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の AND（論理積）演算を実行する。

説明

デスティネーション・オペランド（第 1 オペランド）内の 4 つのパックド単精度浮動小数点値の各ビットを反転して、反転された中間結果とソース・オペランド（第 2 オペランド）内の 4 つのパックド単精度浮動小数点値の間でビット単位の AND（論理積）演算を実行し、結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

操作

```
DEST[127-0] ← (NOT(DEST[127-0])) BitwiseAND (SRC[127-0]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

ANDNPS __m128 _mm_andnot_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

ANDNPS—Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values（続き）

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

ARPL—Adjust RPL Field of Segment Selector

オペコード	命令	説明
63 /r	ARPL *r/m16*, *r16*	*r/m16* の RPL を *r16* の RPL 未満に調整する。

説明

2つのセグメント・セレクタのRPLフィールドを比較する。第1オペランド（デスティネーション・オペランド）に一方のセグメント・セレクタがあり、第2オペランド（ソース・オペランド）に他方のセグメント・セレクタがある。（RPLフィールドは各オペランドのビット0と1を占める。）デスティネーション・オペランドのRPLフィールドがソース・オペランドのRPLフィールドより小さい場合は、ZFフラグがセットされ、デスティネーション・オペランドのRPLフィールドが増加されてソース・オペランドのRPLフィールドに合わせられる。そうでない場合は、ZFフラグがクリアされ、デスティネーション・オペランドの変更は行われない。（デスティネーション・オペランドには、ワードレジスタまたはメモリ・ロケーションを使用できる。ソース・オペランドは、ワードレジスタでなければならない。）

APRL命令は、オペレーティング・システム・プロシージャに使用させるために設けられている（ただし、アプリケーションも使用できる）。この命令は、一般に、アプリケーションからオペレーティング・システムに渡されていたセグメント・セレクタのRPLを調整して、アプリケーション・プログラムの特権レベルに合わせるために使用する。ここで、オペレーティング・システムに渡されていたセグメント・セレクタがデスティネーション・オペランドに入れられ、アプリケーション・プログラムのコード・セグメントのセグメント・セレクタがソース・オペランドに入れられる。（ソース・オペランドのRPLフィールドはアプリケーション・プログラムの特権レベルを表す。）次に、ARPL命令を実行すれば、オペレーティング・システムが受け取っていたセグメント・セレクタのRPLをアプリケーション・プログラムの特権レベルより低く（特権レベルを高く）させないように保証できる。（アプリケーション・プログラムのコード・セグメントのセグメント・セレクタはプロシージャ・コールの後でスタックから読み出すことができる。）

この命令の使用方法の詳細については、『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、下巻』の第4章の「呼び出し側のアクセス特権のチェック」を参照のこと。

ARPL—Adjust RPL Field of Segment Selector（続き）

操作

```
IF DEST[RPL) < SRC[RPL)
THEN
    ZF ← 1;
    DEST[RPL) ← SRC[RPL);
ELSE
    ZF ← 0;
FI;
```

影響を受けるフラグ

デスティネーション・オペランドのRPLフィールドがソース・オペランドのRPLフィールドより小さい場合は、ZF フラグが 1 にセットされる。そうでない場合は、ZF フラグが 0 にセットされる。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリをアクセスしたが、その内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#UD	実アドレスモードでは、ARPL 命令は認識されない。

仮想 8086 モード例外

#UD	仮想 8086 モードでは、ARPL 命令は認識されない。

BOUND—Check Array Index Against Bounds

オペコード	命令	説明
62 /r	BOUND *r16*, *m16&16*	*r16* (配列インデックス) が *m16&16* の指定範囲内かどうかを確認する。
62 /r	BOUND *r32*, *m32&32*	*r32* (配列インデックス) が *m16&16* の指定範囲内かどうかを確認する。

説明

第 1 オペランド（配列インデックス）が第 2 オペランド（範囲オペランド）によって指定される配列の下限から上限までの範囲内であるかどうか判定する。配列インデックスは、レジスタ内の符号付き整数である。範囲オペランドは、オペランド・サイズ属性が 32 の場合は 1 対の符号付きダブルワード整数を、オペランド・サイズ属性が 16 の場合は 1 対の符号付きワード整数を内容とするメモリ・ロケーションである。第 1 のダブルワード（またはワード）が配列の下限であり、第 2 のダブルワード（またはワード）が配列の上限である。配列インデックスは下限より大きいか等しくなければならず、かつ上限にバイト単位のオペランド・サイズを加算した値より小さいか等しくなければならない。インデックスが上下限の範囲内でない場合は、BOUND 範囲外例外（#BR）が報告される。（この例外が発生したときは、セーブされたリターン命令ポインタが BOUND 命令を指している。）

上下限範囲データ構造 (配列の上下限を内容とする 2 つのワードまたはダブルワード) は、通常、配列自体の直前に置いて、配列の先頭からの一定のオフセットで上下限をアドレス指定できるようにする。配列のアドレスはすでにレジスタにストアされているので、上記のような慣行によって、配列の上下限の実効アドレスを得るために余分なバスサイクルを避けることができる。

操作

```
IF (ArrayIndex < LowerBound OR ArrayIndex > UppderBound)
    (* Below lower bound or above upper bound *)
    THEN
        #BR;
FI;
```

影響を受けるフラグ

なし。

BOUND—Check Array Index Against Bounds（続き）

保護モード例外

#BR	範囲テストが失敗した場合。
#UD	第 2 オペランドがメモリ・ロケーションでない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#BR	範囲テストが失敗した場合。
#UD	第 2 オペランドがメモリ・ロケーションでない場合。
#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#BR	範囲テストが失敗した場合。
#UD	第 2 オペランドがメモリ・ロケーションでない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BSF—Bit Scan Forward

オペコード	命令	説明
0F BC	BSF *r16*, *r/m16*	*r/m16* を順方向にビットスキャンする。
0F BC	BSF *r32*, *r/m32*	*r/m32* を順方向にビットスキャンする。

説明

ソース・オペランド（第 2 オペランド）を検索して、最下位のセットビット（値 1 のビット）を探す。最下位のセットビットが見つかると、そのビット・インデックスがデスティネーション・オペランド（第 1 オペランド）にストアされる。ソース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。デスティネーション・オペランドはレジスタである。ビット・インデックスは、ソース・オペランドのビット 0 からの符号なしオフセットである。ソース・オペランドの内容が 0 の場合は、デスティネーション・オペランドの内容は未定義になる。

操作

```
IF SRC = 0
    THEN
        ZF ← 1;
        DEST is undefined;
    ELSE
        ZF ← 0;
        temp ← 0;
    WHILE Bit(SRC, temp) = 0
    DO
        temp ← temp + 1;
        DEST ← temp;
    OD;
FI;
```

影響を受けるフラグ

ソース・オペランドのすべてのビットが 0 の場合は、ZF フラグが 1 にセットされる。そうでない場合は、ZF フラグがクリアされる。CF、OF、SF、AF、PF フラグは未定義。

BSF—Bit Scan Forward（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BSR—Bit Scan Reverse

オペコード	命令	説明
0F BD	BSR *r16*, *r/m16*	*r/m16* を逆方向にビットスキャンする。
0F BD	BSR *r32*, *r/m32*	*r/m32* を逆方向にビットスキャンする。

説明

ソース・オペランド（第 2 オペランド）を検索して、最上位のセットビット（値 1 のビット）を探す。最上位のセットビットが見つかると、そのビット・インデックスがデスティネーション・オペランド（第 1 オペランド）にストアされる。ソース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。デスティネーション・オペランドはレジスタである。ビット・インデックスは、ソース・オペランドのビット 0 からの符号なしオフセットである。ソース・オペランドの内容が 0 の場合は、デスティネーション・オペランドの内容は未定義になる。

操作

```
IF SRC = 0
    THEN
        ZF ← 1;
        DEST is undefined;
    ELSE
        ZF ← 0;
        temp ← OperandSize – 1;
    WHILE Bit(SRC, temp) = 0
    DO
        temp ← temp – 1;
        DEST ← temp;
    OD;
FI;
```

影響を受けるフラグ

ソース・オペランドのすべてのビットが 0 の場合は、ZF フラグが 1 にセットされる。そうでない場合は、ZF フラグはクリアされる。CF、OF、SF、AF、PF フラグは未定義。

BSR—Bit Scan Reverse（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BSWAP—Byte Swap

オペコード	命令	説明
0F C8+rd	BSWAP *r32*	32 ビット・レジスタのバイト順序を逆にする。

説明

32 ビットの（デスティネーション）レジスタのバイトの順序を逆にする。すなわち、ビット 0 から 7 がビット 24 から 31 と入れ換えられ、ビット 8 から 15 がビット 16 から 23 と入れ換えられる。この命令は、リトル・エンディアン値をビッグ・エンディアン・フォーマットに、およびその逆に変換するために設けられている。

ワード値（16 ビット・レジスタ）内のバイトを入れ換えるには、XCHG 命令を使用する。BSWAP 命令が 16 ビット・レジスタを参照すると、結果は未定義になる。

IA-32 の互換性

BSWAP 命令は、Intel486™ プロセッサ・ファミリより以前の IA-32 プロセッサに対してはサポートされていない。この命令との互換性を保証するためには、Intel486 プロセッサ・ファミリより以前のインテル・プロセッサで実行できる、機能的に等価なコードを組み込むこと。

操作

```
TEMP ← DEST
DEST[7..0] ← TEMP(31..24]
DEST[15..8] ← TEMP(23..16]
DEST[23..16] ← TEMP(15..8]
DEST[31..24] ← TEMP(7..0]
```

影響を受けるフラグ

なし。

例外（すべての操作モード）

なし。

BT—Bit Test

オペコード	命令	説明
0F A3	BT *r/m16*, *r16*	選択したビットを CF フラグにストアする。
0F A3	BT *r/m32*, *r32*	選択したビットを CF フラグにストアする。
0F BA /4 *ib*	BT *r/m16*, *imm8*	選択したビットを CF フラグにストアする。
0F BA /4 *ib*	BT *r/m32*, *imm8*	選択したビットを CF フラグにストアする。

説明

ビット・ベース・オペランド（第 1 オペランド）によって指定されるビット・ストリング内で、ビット・オフセット・オペランド（第 2 オペランド）によって指定されるビット位置のビットを選択し、そのビットの値を CF フラグにストアする。ビット・ベース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ビット・オフセット・オペランドには、レジスタまたは即値を使用できる。ビット・ベース・オペランドがレジスタを指定している場合は、この命令はビット・オフセット・オペランドの（レジスタサイズにしたがって）16 または 32 の剰余をとり、それぞれ 16 または 32 ビットのレジスタで任意のビット位置を選択できるようにしている（図 3-1. を参照）。ビット・ベース・オペランドがメモリ・ロケーションを指定している場合は、このオペランドは対象のビット・ストリングのビットベース（指定されたバイトのビット 0）を含むメモリ内のバイトのアドレスを表している（図 3-2. を参照）。したがって、オフセット・オペランドは、レジスタ・オフセットの場合は -2^{31} から $2^{31} - 1$ までの範囲のビット位置を、また即値オフセットの場合は 0 から 31 までの範囲のビット位置を選択する。

一部のアセンブラは、即値ビット・オフセット・フィールドをメモリ・オペランドのディスプレースメント・フィールドと組み合わせて使用することにより、31 より大きい即値ビット・オフセットをサポートしている。その場合は、アセンブラによって、即値ビット・オフセットの下位 3 または 5 ビット（16 ビット・オペランドの場合は 3、32 ビット・オペランドの場合は 5）が即値ビット・オフセット・フィールドにストアされ、上位ビットはシフトされ、アドレス指定モードのバイト・ディスプレースメントと組み合わされる。プロセッサは、0 でなくても、上位ビットを無視する。

メモリ内のビットをアクセスするときは、プロセッサは、以下の関係を使用して、32 ビットのオペランド・サイズのメモリアドレスから始まる 4 バイトをアクセスできる。

実効アドレス + （4 * (BitOffset DIV 32)）

または、以下の関係を使用して、16 ビット・オペランドのメモリアドレスから始まる 2 バイトをアクセスできる。

BT—Bit Test（続き）

実効アドレス +（2 * (BitOffset DIV 16)）

該当するビットに到達するのに 1 バイトだけのアクセスで足りるときでも、上記の操作に差し支えはない。このビットアドレス指定方式を使用するときは、ソフトウェアはアドレス・スペース・ホールに近いメモリ領域の参照を避けるようにすること。特に、メモリマップド I/O レジスタへの参照は避けなければならない。その代わりに、ソフトウェアに MOV 系の命令を使用してそれらのアドレスからロードまたはそれらのアドレスにストアさせ、レジスタ形式のこれらの命令を使用してデータを操作するようにする。

操作

CF ← Bit(BitBase, BitOffset)

影響を受けるフラグ

CF フラグに選択されたビットの値がストアされる。OF、SF、ZF、AF、PF フラグは未定義。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

BT—Bit Test（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BTC—Bit Test and Complement

オペコード	命令	説明
0F BB	BTC *r/m16*, *r16*	選択したビットを CF フラグにストアし、補数をとる。
0F BB	BTC *r/m32*, *r32*	選択したビットを CF フラグにストアし、補数をとる。
0F BA /7 *ib*	BTC *r/m16*, *imm8*	選択したビットを CF フラグにストアし、補数をとる。
0F BA /7 *ib*	BTC *r/m32*, *imm8*	選択したビットを CF フラグにストアし、補数をとる。

説明

ビット・ベース・オペランド（第1オペランド）によって指定されるビット・ストリング内で、ビット・オフセット・オペランド（第2オペランド）によって指定されるビット位置のビットを選択し、そのビットの値を CF フラグにストアし、ビット・ストリング内の選択したビットを補数に変換する。ビット・ベース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ビット・オフセット・オペランドには、レジスタまたは即値を使用できる。ビット・ベース・オペランドがレジスタを指定している場合は、この命令はビット・オフセット・オペランドの（レジスタサイズにしたがって）16または32の剰余をとり、それぞれ16または32ビットのレジスタで任意のビット位置を選択できるようにしている（図 3-1. を参照）。ビット・ベース・オペランドがメモリ・ロケーションを指定している場合は、このオペランドは対象のビット・ストリングのビットベース（指定されたバイトのビット0）を含むメモリ内のバイトのアドレスを表している（図 3-2. を参照）。したがって、オフセット・オペランドは、レジスタ・オフセットの場合は -2^{31} から $2^{31} - 1$ までの範囲のビット位置を、また即値オフセットの場合は0から31までの範囲のビット位置を選択する。

一部のアセンブラは、即値ビット・オフセット・フィールドをメモリ・オペランドのディスプレースメント・フィールドと組み合わせて使用することにより、31より大きい即値ビット・オフセットをサポートしている。このアドレス指定方式の詳細については、本章の「BT—Bit Test」を参照のこと。

この命令をLOCKプリフィックスと共に使用すると、アトミックに命令を実行できる。

操作

```
CF ← Bit(BitBase, BitOffset)
Bit(BitBase, BitOffset) ← NOT Bit(BitBase, BitOffset);
```

影響を受けるフラグ

CF フラグに選択されたビットの補数に変換される前の値がストアされる。OF、SF、ZF、AF、およびPFフラグは未定義。

BTC—Bit Test and Complement（続き）

保護モード例外

#GP(0)	デスティネーション・オペランドの指示先が書き込み不可能なセグメントの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BTR—Bit Test and Reset

オペコード	命令	説明
0F B3	BTR *r/m16*, *r16*	選択したビットを CF フラグにストアし、クリアする。
0F B3	BTR *r/m32*, *r32*	選択したビットを CF フラグにストアし、クリアする。
0F BA /6 *ib*	BTR *r/m16*, *imm8*	選択したビットを CF フラグにストアし、クリアする。
0F BA /6 *ib*	BTR *r/m32*, *imm8*	選択したビットを CF フラグにストアし、クリアする。

説明

ビット・ベース・オペランド（第 1 オペランド）によって指定されるビット・ストリング内で、ビット・オフセット・オペランド（第 2 オペランド）によって指定されるビット位置のビットを選択し、そのビットの値を CF フラグにストアし、ビット・ストリング内の選択したビットを 0 にクリアする。ビット・ベース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ビット・オフセット・オペランドには、レジスタまたは即値を使用できる。ビット・ベース・オペランドがレジスタを指定している場合は、この命令はビット・オフセット・オペランドの（レジスタサイズにしたがって）16 または 32 の剰余をとり、それぞれ 16 または 32 ビットのレジスタで任意のビット位置を選択できるようにしている（図 3-1. を参照）。ビット・ベース・オペランドがメモリ・ロケーションを指定している場合は、このオペランドは対象のビット・ストリングのビットベース（指定されたバイトのビット 0）を含むメモリ内のバイトのアドレスを表している（図 3-2. を参照）。したがって、オフセット・オペランドは、レジスタ・オフセットの場合は -2^{31} から $2^{31} - 1$ までの範囲のビット位置を、また即値オフセットの場合は 0 から 31 までの範囲のビット位置を選択する。

一部のアセンブラは、即値ビット・オフセット・フィールドをメモリ・オペランドのディスプレースメント・フィールドと組み合わせて使用することにより、31 より大きい即値ビット・オフセットをサポートしている。このアドレス指定方式の詳細については、本章の「BT—Bit Test」を参照のこと。

この命令を LOCK プリフィックスと共に使用すると、アトミックに命令を実行させることができる。

操作

```
CF ← Bit(BitBase, BitOffset)
Bit(BitBase, BitOffset) ← 0;
```

影響を受けるフラグ

CF フラグに選択されたビットのクリアされる前の値がストアされる。OF、SF、ZF、AF、PF フラグは未定義。

BTR—Bit Test and Reset（続き）

保護モード例外

#GP(0)	デスティネーション・オペランドの指示先が書き込み不可能なセグメントの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

BTS—Bit Test and Set

オペコード	命令	説明
0F AB	BTS *r/m16*, *r16*	選択したビットを CF フラグにストアし、セットする。
0F AB	BTS *r/m32*, *r32*	選択したビットを CF フラグにストアし、セットする。
0F BA /5 *ib*	BTS *r/m16*, *imm8*	選択したビットを CF フラグにストアし、セットする。
0F BA /5 *ib*	BTS *r/m32*, *imm8*	選択したビットを CF フラグにストアし、セットする。

説明

ビット・ベース・オペランド（第 1 オペランド）によって指定されるビット・ストリング内で、ビット・オフセット・オペランド（第 2 オペランド）によって指定されるビット位置のビットを選択し、そのビットの値を CF フラグにストアし、ビット・ストリング内の選択したビットを 1 にセットする。ビット・ベース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。ビット・オフセット・オペランドには、レジスタまたは即値を使用できる。ビット・ベース・オペランドがレジスタを指定している場合は、この命令はビット・オフセット・オペランドの（レジスタサイズにしたがって）16 または 32 の剰余をとり、それぞれ 16 または 32 ビットのレジスタで任意のビット位置を選択できるようにしている（図 3-1. を参照）。ビット・ベース・オペランドがメモリ・ロケーションを指定している場合は、このオペランドは対象のビット・ストリングのビットベース（指定されたバイトのビット 0）を含むメモリ内のバイトのアドレスを表している（図 3-2. を参照）。したがって、オフセット・オペランドは、レジスタ・オフセットの場合は -2^{31} から $2^{31} - 1$ までの範囲のビット位置を、また即値オフセットの場合は 0 から 31 までの範囲のビット位置を選択する。

一部のアセンブラは、即値ビット・オフセット・フィールドをメモリ・オペランドのディスプレースメント・フィールドと組み合わせて使用することにより、31 より大きい即値ビット・オフセットをサポートしている。このアドレス指定機構の詳細については、本章の「BT—Bit Test」を参照のこと。

この命令をLOCKプリフィックスと共に使用すると、アトミックに命令を実行できる。

操作

```
CF ← Bit(BitBase, BitOffset)
Bit(BitBase, BitOffset) ← 1;
```

影響を受けるフラグ

CF フラグに選択されたビットのセットされる前の値がストアされる。OF、SF、ZF、AF、PF フラグは未定義。

BTS—Bit Test and Set（続き）

保護モード例外

#GP(0)	デスティネーション・オペランドの指示先が書き込み不可能なセグメントの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CALL—Call Procedure

オペコード	命令	説明
E8 *cw*	CALL *rel16*	相対 near コール、次の命令とディスプレースメント相対。
E8 *cd*	CALL *rel32*	相対 near コール、次の命令とディスプレースメント相対。
FF /2	CALL *r/m16*	絶対間接 near コール、*r/m16* でアドレスを指定。
FF /2	CALL *r/m32*	絶対間接 near コール、*r/m32* でアドレスを指定。
9A *cd*	CALL *ptr16:16*	絶対 far コール、オペランドでアドレスを指定。
9A *cp*	CALL *ptr16:32*	絶対 far コール、オペランドでアドレスを指定。
FF /3	CALL *m16:16*	絶対間接 far コール、*m16:16* でアドレスを指定。
FF /3	CALL *m16:32*	絶対間接 far コール、*m16:32* でアドレスを指定。

説明

プロシージャ・リンク情報をスタックにセーブし、デスティネーション（ターゲット）オペランドで指定されるプロシージャ（コール先プロシージャ）に分岐する。ターゲット・オペランドは、コール先プロシージャの最初の命令のアドレスを指定する。このオペランドには、即値、汎用レジスタ、またはメモリ・ロケーションを使用できる。

この命令を使用して、以下の異なる 4 つのタイプのコールを実行できる。

- near コール ― 現在のコード・セグメント（現在の CS レジスタの指示先のセグメント）内にあるプロシージャへのコール。セグメント内コールともいう。
- far コール ― 現在のコード・セグメントとは異なるセグメント内にあるプロシージャへのコール。セグメント間コールともいう。
- 特権レベル間 far コール ― 現在実行中のプログラムまたはプロシージャの特権レベルとは異なる特権レベルのセグメント内にあるプロシージャへのコール。
- タスクスイッチ ― 異なるタスク内にあるプロシージャへのコール。

上記の最後の 2 つのコールタイプ（特権レベル間コールとタスクスイッチ）は、保護モードでのみ実行できる。near、far、特権レベル間の各コールタイ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 6 章の「CALL と RET によるプロシージャのコール」の説明を参照のこと。CALL 命令によるタスクスイッチ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、下巻』の第 6 章「タスク管理」を参照のこと。

CALL—Call Procedure（続き）

near コール：near コールを実行するときは、プロセッサは、EIP レジスタの値（CALL 命令の次の命令のオフセットを内容とする）を（後でリターン命令のポインタとして使用するために）スタックにプッシュする。プロセッサは、次にターゲット・オペランドで指定される現在のコード・セグメント内のアドレスに分岐する。ターゲット・オペランドは、コード・セグメント内の絶対オフセット（すなわち、コード・セグメントのベースからのオフセット）か、相対オフセット（CALL 命令の次の命令を指示先とする、EIP レジスタ内の命令ポインタの現在値に相対的な符号付きディスプレースメント）かを指定する。CS レジスタは near コールでは変わらない。

near コールについては、絶対オフセットは汎用レジスタまたはメモリ・ロケーション（r/m16 または r/m32）で間接的に指定する。オペランド・サイズ属性によってターゲット・オペランドのサイズ（16 または 32 ビット）が決まる。ターゲット・オフセットは EIP レジスタに直接ロードされる。オペランド・サイズ属性が 16 の場合は、EIP レジスタの上位 2 バイトがクリアされ、命令ポインタの最大サイズが 16 ビットになる。（スタックポインタ[ESP]をベースレジスタとして使用して絶対オフセットを間接的にアクセスするときは、使用されるベース値は命令が実行される前の ESP の値である。）

相対オフセット（rel16 または rel32）は、一般的にアセンブリ・コードではラベルとして指定されるが、マシン・コード・レベルでは符号付きの 16 または 32 ビット即値にコード化される。この値が EIP レジスタの値に加算される。絶対オフセットの場合と同様に、オペランド・サイズ属性によってターゲット・オペランドのサイズ（16 または 32 ビット）が決まる。

実アドレスモードまたは仮想 8086 モードでの far コール：実アドレスモードまたは仮想 8086 モードで far コールを実行するときは、プロセッサは、リターン命令ポインタとして使用するために、CS および EIP 両レジスタの現在の値をスタックにプッシュする。プロセッサは、次にコール先プロシージャのターゲット・オペランドで指定されるコード・セグメントとオペランドへの「far 分岐」を実行する。この場合、ターゲット・オペランドは、絶対 far アドレスをポインタ（ptr16:16 または ptr16:32）で直接的に、またはメモリ・ロケーション（m16:16 または m16:32）で間接的に指定する。ポインタ方式では、コール先プロシージャのセグメントとオフセットは、命令内で 4 バイト（16 ビット・オペランド・サイズ）または 6 バイト（32 ビット・オペランド・サイズ）の far 即値アドレスを使用してコード化される。間接方式では、ターゲット・オペランドは、4 バイト（16 ビット・オペランド・サイズ）または 6 バイト（32 ビット・オペランド・サイズ）の far アドレスがストアされているメモリ・ロケーションを指定する。オペランド・サイズ属性によって far アドレス内のオフセットのサイズ（16 または 32 ビット）が決まる。far アドレスは CS および EIP レジスタに直接ロードされる。オペランド・サイズ属性が 16 の場合は、EIP レジスタの上位 2 バイトが 0 にクリアされる。

CALL—Call Procedure（続き）

保護モードでの far コール: プロセッサが保護モードで動作しているときは、CALL 命令を使用して以下の 3 つのタイプの far コールを実行できる。

- 同じ特権レベルへの far コール
- 異なる特権レベルへの far コール（特権レベル間コール）
- タスクスイッチ（別のタスクへの far コール）

保護モードでは、プロセッサは常に far アドレスのセグメント・セレクタ部分を使用して GDT または LDT 内の対応する記述子をアクセスする。記述子タイプ（コード・セグメント、コールゲート、タスクゲート、または TSS）とアクセス権によって実行されるコール操作のタイプが決まる。

選択された記述子がコード・セグメントのものである場合は、同じ特権レベルのコード・セグメントへの far コールが実行される。（選択されたコード・セグメントが異なる特権レベルにあり、非コンフォーミング・コード・セグメントの場合は、一般保護例外が発生する。）保護モードでの同じ特権レベルへの far コールは、実アドレスモードまたは仮想 8086 モードで実行する far コールに非常によく似ている。ターゲット・オペランドは、絶対 far アドレスをポインタ（ptr16:16 または ptr16:32）で直接的に、またはメモリ・ロケーション（m16:16 または m16:32）で間接的に指定する。オペランド・サイズ属性によって far アドレス内のオフセットのサイズ（16 または 32 ビット）が決まる。新しいコード・セグメント・セレクタとその記述子が CS レジスタにロードされ、命令からのオフセットが EIP レジスタにロードされる。

コールゲート（次の段落で説明）を使用して、同じ特権レベルのコード・セグメントへの far コールも実行できるので注意する。この仕組みを使用すると、特別レベルのインダイレクションが可能になり、これは 16 ビットと 32 ビットとのコード・セグメント間の好ましいコール実行方法である。

CALL—Call Procedure（続き）

特権レベル間の far コールを実行するときは、コール先プロシージャのコード・セグメントにはコールゲートを介してアクセスしなければならない。コールゲートはターゲット・オペランドによって指定されるセグメント・セレクタによって指定される。ターゲット・オペランドは、ここでもやはりコール・ゲート・セグメント・セレクタをポインタ（ptr16:16 または ptr16:32）で直接的に、またはメモリ・ロケーション（m16:16 または m16:32）で間接的に指定できる。プロセッサは、コールゲート記述子から新しいコード・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（オフセット）を得る。（コールゲートを使用するときは、ターゲット・オペランドからのオフセットは無視される。）特権レベル間コールでは、プロセッサはコール先プロシージャの特権レベルのスタックに切り替える。新しいスタック・セグメントのセグメント・セレクタは、現在実行中のタスクの TSS で指定される。新しいコード・セグメントへの分岐はスタックスイッチの後で行われる。（コールゲートを使用して同じ特権レベルのセグメントへの far コールを実行するときは、スタックスイッチは行われないので注意する。）プロセッサは、新しいスタックに、コール元プロシージャのスタックのセグメント・セレクタとスタックポインタ、およびコール元プロシージャのコード・セグメントのセグメント・セレクタと命令ポインタをプッシュする。（コールゲート記述子内の値によって、新しいスタックにコピーするパラメータの数が決まる。）プロセッサは、最後に新しいコード・セグメント内のコール先プロシージャのアドレスに分岐する。

CALL 命令でタスクスイッチを実行するのは、コールゲートを介したコールを実行するのに多少似ている。タスクスイッチの場合、ターゲット・オペランドは切り替え先タスクへのタスクゲートのセグメント・セレクタを指定する（ターゲット・オペランド内のオフセットは無視される）。次に、タスクゲートがタスクの TSS を指す。TSS の内容はタスクのコードおよびスタックの両セグメントのセグメント・セレクタである。TSS には、さらに、タスクが中断される前に、次に実行される予定であった命令の EIP 値も入っている。この命令ポインタ値は EIP レジスタにロードされ、したがってタスクは再び、この中断された点の次の命令から実行を開始する。

CALL 命令は、TSS のセグメント・セレクタを直接指定することもでき、その結果タスクゲートのインダイレクションの必要がなくなる。タスクスイッチの仕組みの詳細については、『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、下巻』の第 6 章「タスク管理」を参照のこと。

CALL—Call Procedure（続き）

CALL 命令でタスクスイッチを実行すると、EFLAGS レジスタにネストされたタスクフラグ（NT）がセットされ、新しいTSSの以前のタスク・リンク・フィールドに前のタスクのTSSセレクタがロードされることに注意する。IRET 命令を実行することにより、コードはこのネストされたタスクを中断するものと予測される。すなわち、NTフラグがセットされているために、IRET 命令は自動的に以前のタスクリンクを使用してコール元タスクに戻る（制御を戻す）。（ネストされたタス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、下巻』の第6章の「タスクのリンク」の説明を参照のこと。）CALL 命令でのタクスの切り替えは、この点において JMP 命令とは異なる。JMP 命令は、NT フラグをセットしないため、IRET 命令がタスクを中断することを予測しない。

16ビットおよび32ビット両コールの混用: 16ビットと32ビットのコード・セグメント間のコールを実行するときは、コールゲートを介して実行する。32 ビットのコード・セグメントから 16 ビットのコード・セグメントへの far コールの場合、コールは 32 ビット・コード・セグメントの最初の 64 K バイトから実行する必要がある。それは、命令のオペランド・サイズ属性が 16 に設定されているため、16 ビットのリターン・アドレス・オフセットしかセーブされないからである。さらに、スタックに 16 ビット値がプッシュされるように、16 ビットのコールゲートを使用する。16 ビットと32 ビットのコード・セグメント間でのコール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、下巻』の第17章「16ビット・コードと32ビット・コードの混在」を参照のこと。

操作

```
IF near call
    THEN IF near relative call
        IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
            THEN IF OperandSize = 32
                THEN
                    IF stack not large enough for a 4-byte return address THEN #SS(0); FI;
                    Push(EIP);
                    EIP ← EIP + DEST; (* DEST is rel32 *)
                ELSE (* OperandSize = 16 *)
                    IF stack not large enough for a 2-byte return address THEN #SS(0); FI;
                    Push(IP);
                    EIP ← (EIP + DEST) AND 0000FFFFH; (* DEST is rel16 *)
            FI;
        FI;
    ELSE (* near absolute call *)
        IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
        IF OperandSize = 32
            THEN
                IF stack not large enough for a 4-byte return address THEN #SS(0); FI;
                Push(EIP);
```

CALL—Call Procedure（続き）

```
                EIP ← DEST; (* DEST is r/m32 *)
            ELSE (* OperandSize = 16 *)
                IF stack not large enough for a 2-byte return address THEN #SS(0); FI;
                Push(IP);
                EIP ← DEST AND 0000FFFFH; (* DEST is r/m16 *)
        FI;
    FI:
FI;

IF far call AND (PE = 0 OR (PE = 1 AND VM = 1)) (* real-address or virtual-8086 mode *)
    THEN
        IF OperandSize = 32
            THEN
                IF stack not large enough for a 6-byte return address THEN #SS(0); FI;
                IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
                Push(CS); (* padded with 16 high-order bits *)
                Push(EIP);
                CS ← DEST[47:32]; (* DEST is ptr16:32 or [m16:32] *)
                EIP ← DEST[31:0]; (* DEST is ptr16:32 or [m16:32] *)
            ELSE (* OperandSize = 16 *)
                IF stack not large enough for a 4-byte return address THEN #SS(0); FI;
                IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
                Push(CS);
                Push(IP);
                CS ← DEST[31:16]; (* DEST is ptr16:16 or [m16:16] *)
                EIP ← DEST[15:0]; (* DEST is ptr16:16 or [m16:16] *)
                EIP ← EIP AND 0000FFFFH; (* clear upper 16 bits *)
        FI;
FI;

IF far call AND (PE = 1 AND VM = 0) (* Protected mode, not virtual-8086 mode *)
    THEN
        IF segment selector in target operand null THEN #GP(0); FI;
        IF segment selector index not within descriptor table limits
            THEN #GP(new code segment selector);
        FI;
        Read type and access rights of selected segment descriptor;
        IF segment type is not a conforming or nonconforming code segment, call gate,
            task gate, or TSS THEN #GP(segment selector); FI;
        Depending on type and access rights
            GO TO CONFORMING-CODE-SEGMENT;
            GO TO NONCONFORMING-CODE-SEGMENT;
            GO TO CALL-GATE;
            GO TO TASK-GATE;
            GO TO TASK-STATE-SEGMENT;
FI;
```

CALL—Call Procedure（続き）

```
CONFORMING-CODE-SEGMENT:
    IF DPL > CPL THEN #GP(new code segment selector); FI;
    IF segment not present THEN #NP(new code segment selector); FI;
    IF OperandSize = 32
        THEN
            IF stack not large enough for a 6-byte return address THEN #SS(0); FI;
            IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
            Push(CS); (* padded with 16 high-order bits *)
            Push(EIP);
            CS ← DEST[NewCodeSegmentSelector);
            (* segment descriptor information also loaded *)
            CS(RPL) ← CPL
            EIP ← DEST[offset);
        ELSE (* OperandSize = 16 *)
            IF stack not large enough for a 4-byte return address THEN #SS(0); FI;
            IF the instruction pointer is not within code segment limit THEN #GP(0); FI;
            Push(CS);
            Push(IP);
            CS ← DEST[NewCodeSegmentSelector);
            (* segment descriptor information also loaded *)
            CS(RPL) ← CPL
            EIP ← DEST[offset) AND 0000FFFFH; (* clear upper 16 bits *)
    FI;
END;

NONCONFORMING-CODE-SEGMENT:
    IF (RPL > CPL) OR (DPL ≠ CPL) THEN #GP(new code segment selector); FI;
    IF segment not present THEN #NP(new code segment selector); FI;
    IF stack not large enough for return address THEN #SS(0); FI;
    tempEIP ← DEST[offset)
    IF OperandSize=16
        THEN
            tempEIP ← tempEIP AND 0000FFFFH; (* clear upper 16 bits *)
    FI;
    IF tempEIP outside code segment limit THEN #GP(0); FI;
    IF OperandSize = 32
        THEN
            Push(CS); (* padded with 16 high-order bits *)
            Push(EIP);
            CS ← DEST[NewCodeSegmentSelector);
            (* segment descriptor information also loaded *)
            CS(RPL) ← CPL;
            EIP ← tempEIP;
        ELSE (* OperandSize = 16 *)
            Push(CS);
            Push(IP);
            CS ← DEST[NewCodeSegmentSelector);
            (* segment descriptor information also loaded *)
            CS(RPL) ← CPL;
```

CALL—Call Procedure（続き）

```
                EIP ← tempEIP;
        FI;
    END;

    CALL-GATE:
        IF call gate DPL < CPL or RPL THEN #GP(call gate selector); FI;
        IF call gate not present THEN #NP(call gate selector); FI;
        IF call gate code-segment selector is null THEN #GP(0); FI;
        IF call gate code-segment selector index is outside descriptor table limits
            THEN #GP(code segment selector); FI;
        Read code segment descriptor;
        IF code-segment segment descriptor does not indicate a code segment
        OR code-segment segment descriptor DPL > CPL
            THEN #GP(code segment selector); FI;
        IF code segment not present THEN #NP(new code segment selector); FI;
        IF code segment is non-conforming AND DPL < CPL
            THEN go to MORE-PRIVILEGE;
            ELSE go to SAME-PRIVILEGE;
        FI;
    END;

    MORE-PRIVILEGE:
        IF current TSS is 32-bit TSS
            THEN
                TSSstackAddress ← new code segment (DPL * 8) + 4
                IF (TSSstackAddress + 7) > TSS limit
                    THEN #TS(current TSS selector); FI;
                newSS ← TSSstackAddress + 4;
                newESP ← stack address;
            ELSE (* TSS is 16-bit *)
                TSSstackAddress ← new code segment (DPL * 4) + 2
                IF (TSSstackAddress + 4) > TSS limit
                    THEN #TS(current TSS selector); FI;
                newESP ← TSSstackAddress;
                newSS ← TSSstackAddress + 2;
        FI;
        IF stack segment selector is null THEN #TS(stack segment selector); FI;
        IF stack segment selector index is not within its descriptor table limits
            THEN #TS(SS selector); FI
        Read code segment descriptor;
        IF stack segment selector's RPL ≠ DPL of code segment
            OR stack segment DPL ≠ DPL of code segment
            OR stack segment is not a writable data segment
                THEN #TS(SS selector); FI
        IF stack segment not present THEN #SS(SS selector); FI;
        IF CallGateSize = 32
            THEN
                IF stack does not have room for parameters plus 16 bytes
                    THEN #SS(SS selector); FI;
```

CALL—Call Procedure（続き）

```
            IF CallGate(InstructionPointer) not within code segment limit THEN #GP(0); FI;
            SS ← newSS;
            (* segment descriptor information also loaded *)
            ESP ← newESP;
            CS:EIP ← CallGate(CS:InstructionPointer);
            (* segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* from calling procedure *)
            temp ← parameter count from call gate, masked to 5 bits;
            Push(parameters from calling procedure's stack, temp)
            Push(oldCS:oldEIP); (* return address to calling procedure *)
        ELSE (* CallGateSize = 16 *)
            IF stack does not have room for parameters plus 8 bytes
                THEN #SS(SS selector); FI;
            IF (CallGate(InstructionPointer) AND FFFFH) not within code segment limit
                THEN #GP(0); FI;
            SS ← newSS;
            (* segment descriptor information also loaded *)
            ESP ← newESP;
            CS:IP ← CallGate(CS:InstructionPointer);
            (* segment descriptor information also loaded *)
            Push(oldSS:oldESP); (* from calling procedure *)
            temp ← parameter count from call gate, masked to 5 bits;
            Push(parameters from calling procedure's stack, temp)
            Push(oldCS:oldEIP); (* return address to calling procedure *)
    FI;
    CPL ← CodeSegment(DPL)
    CS(RPL) ← CPL
END;

SAME-PRIVILEGE:
    IF CallGateSize = 32
        THEN
            IF stack does not have room for 8 bytes
                THEN #SS(0); FI;
            IF EIP not within code segment limit then #GP(0); FI;
            CS:EIP ← CallGate(CS:EIP) (* segment descriptor information also loaded *)
            Push(oldCS:oldEIP); (* return address to calling procedure *)
        ELSE (* CallGateSize = 16 *)
            IF stack does not have room for 4 bytes
                THEN #SS(0); FI;
            IF IP not within code segment limit THEN #GP(0); FI;
            CS:IP ← CallGate(CS:instruction pointer)
            (* segment descriptor information also loaded *)
            Push(oldCS:oldIP); (* return address to calling procedure *)
    FI;
    CS(RPL) ← CPL
END;
TASK-GATE:
    IF task gate DPL < CPL or RPL
```

CALL—Call Procedure（続き）

```
            THEN #GP(task gate selector);
        FI;
        IF task gate not present
            THEN #NP(task gate selector);
        FI;
        Read the TSS segment selector in the task-gate descriptor;
        IF TSS segment selector local/global bit is set to local
            OR index not within GDT limits
                THEN #GP(TSS selector);
        FI;
        Access TSS descriptor in GDT;

        IF TSS descriptor specifies that the TSS is busy (low-order 5 bits set to 00001)
                THEN #GP(TSS selector);
        FI;
        IF TSS not present
            THEN #NP(TSS selector);
        FI;
        SWITCH-TASKS (with nesting) to TSS;
        IF EIP not within code segment limit
            THEN #GP(0);
        FI;
END;

TASK-STATE-SEGMENT:
        IF TSS DPL < CPL or RPL
        OR TSS descriptor indicates TSS not available
            THEN #GP(TSS selector);
        FI;
        IF TSS is not present
            THEN #NP(TSS selector);
        FI;
        SWITCH-TASKS (with nesting) to TSS
        IF EIP not within code segment limit
            THEN #GP(0);
        FI;
END;
```

影響を受けるフラグ

タスクスイッチが行われた場合はすべてのフラグが影響を受け、タスクスイッチが行われなかった場合はどのフラグも影響を受けない。

CALL—Call Procedure（続き）

保護モード例外

#GP(0)	デスティネーション・オペランド内のターゲット・オフセットが新しいコード・セグメントの範囲外の場合。
	デスティネーション・オペランド内のセグメント・セレクタがヌルの場合。
	ゲート内のコード・セグメント・セレクタがヌルの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#GP（セレクタ）	コード・セグメント、ゲート、または TSS のセレクタ・インデックスが記述子テーブルの範囲外の場合。
	デスティネーション・オペランドにあるセグメント・セレクタの指示先のセグメント記述子がコンフォーミング・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、またはタスク・ステート・セグメントのいずれのものでもなかった場合。
	非コンフォーミング・コード・セグメントの DPL が CPL に等しくないか、またはセグメントのセグメント・セレクタの RPL が CPL より大きい場合。
	コンフォーミング・コード・セグメントの DPL が CPL より大きい場合。
	コールゲート、タスクゲート、または TSS のセグメント記述子からの DPL が、CPL、またはコールゲート、タスクゲート、または TSS のセグメント・セレクタの RPL より小さい場合。
	コールゲートからのセグメント・セレクタのセグメント記述子が、そのコールゲートがコード・セグメントであることを示していない場合。
	コールゲートからのセグメント･セレクタが記述子テーブルの範囲外の場合。
	コールゲートから得られたコード・セグメントの DPL が CPL より大きい場合。
	TSS のセグメント・セレクタのローカル / グローバル・ビットがローカルとしてセットされている場合。
	TSS のセグメント記述子が、TSS がビジーであるか、または使用不可能であることを示している場合。

CALL—Call Procedure（続き）

#SS(0)	スタックスイッチが行われなかったときに、リターンアドレス、パラメータ、またはスタック・セグメント・ポインタをスタックにプッシュした結果、スタック・セグメントの範囲を超えた場合。 メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#SS（セレクタ）	スタックスイッチが行われたときに、リターンアドレス、パラメータ、またはスタック・セグメント・ポインタをスタックにプッシュした結果、スタック・セグメントの範囲を超えた場合。 スタックスイッチの一環として SS レジスタへのロードが行われようとするとき、指示先のセグメントが存在しないとマークされていた場合。 スタックスイッチが行われたとき、リターンアドレス、パラメータ、またはスタック・セグメント・ポインタをストアするための余裕がスタック・セグメントにない場合。
#NP（セレクタ）	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、コールゲート、タスクゲート、または TSS が存在しない場合。
#TS（セレクタ）	新しいスタック・セグメント・セレクタと ESP が TSS の終りを超えている場合。 新しいスタック・セグメント・セレクタがヌルの場合。 TSS 内の新しいスタック・セグメント・セレクタの RPL がアクセス先のコード・セグメントの DPL と等しくない場合。 新しいスタック・セグメント用のスタック・セグメント記述子の DPL がコード・セグメント記述子の DPL と等しくない場合。 新しいスタック・セグメントが書き込み可能なデータ・セグメントでない場合。 スタック・セグメントのセグメント・セレクタ・インデックスが記述子テーブルの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 ターゲット・オフセットがコード・セグメントの範囲外の場合。

CALL—Call Procedure（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	ターゲット・オフセットがコード・セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CBW/CWDE—Convert Byte to Word/Convert Word to Doubleword

オペコード	命令	説明
98	CBW	AX ← AL の符号拡張
98	CWDE	EAX ← AX の符号拡張

説明

符号拡張により、ソース・オペランドのサイズを 2 倍に拡張する（『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、上巻』の図 6-5. を参照）。CBW（バイトからワードへの変換）命令は、ソース・オペランドの符号（ビット 7）を AH レジスタのすべてのビットにコピーする。CWDE（ワードからダブルワードへの変換）命令は、AX レジスタ内のワードの符号（ビット 15）を EAX レジスタの上位 16 ビットにコピーする。

CBW および CWDE ニーモニックは同じオペコードを参照する。CBW 命令はオペランド・サイズ属性が 16 のとき、また CWDE 命令はオペランド・サイズ属性が 32 のときに使用することを目的としている。一部のアセンブラには、CBW が使用されたときはオペランド・サイズを 16 ビットに、また CWDE が使用されたときは 32 ビットに強制できるものがある。その他のアセンブラは、これらのニーモニックを同義語（CBW/CWDE）として取り扱い、どちらのニーモニックが使用されても、そのときのオペランド・サイズ属性の設定を使用して変換対象の値のサイズを判定できる。

CWDE 命令は、CWD（ワードからダブルへの変換）命令とは異なる。すなわち、CWD 命令はデスティネーション・オペランドして DX:AX レジスタペアを使用するのに対し、CWDE 命令はデスティネーションとして EAX レジスタを使用する。

操作

```
IF OperandSize = 16 (* instruction = CBW *)
    THEN AX ← SignExtend(AL);
    ELSE (* OperandSize = 32, instruction = CWDE *)
        EAX ← SignExtend(AX);
FI;
```

影響を受けるフラグ

なし。

例外（すべての操作モード）

なし。

CDQ—Convert Double to Quad

「CWD/CDQ—Convert Word to Doubleword/Convert Doubleword to Quadword」を参照のこと。

CLC—Clear Carry Flag

オペコード	命令	説明
F8	CLC	CF フラグをクリアする。

説明

EFLAGS レジスタ内の CF フラグをクリアする。

操作

CF ← 0;

影響を受けるフラグ

CF フラグが 0 にセットされる。OF、ZF、SF、AF、および PF フラグは影響を受けない。

例外（すべての操作モード）

なし。

CLD—Clear Direction Flag

オペコード	命令	説明
FC	CLD	DF フラグをクリアする。

説明

EFLAGS レジスタ内の DF フラグをクリアする。DF フラグが 0 にセットされているときは、ストリング操作を行うと、インデックス・レジスタ（ESI または EDI、あるいは両方）がインクリメントされる。

操作

DF ← 0;

影響を受けるフラグ

DFフラグが0にセットされる。CF、OF、ZF、SF、AF、およびPFフラグは影響を受けない。

例外（すべての操作モード）

なし。

CLFLUSH—Cache Line Flush

オペコード	命令	説明
0F AE /7	CLFLUSH *m8*	*m8* が入っているキャッシュ・ラインをフラッシュする。

説明

プロセッサのキャッシュ階層（データおよび命令）のすべてのレベルで、ソース・オペランドで指定されたリニアアドレスが入っているキャッシュ・ラインを無効化する。この無効化は、キャッシュのコヒーレンシ・ドメイン全体にブロードキャストされる。キャッシュ階層のいずれかのレベルで、指定されたキャッシュ・ラインとメモリの内容が一致しない（キャッシュ・ラインがダーティになっている）場合は、無効化される前に、キャッシュ・ラインの内容がメモリに書き込まれる。ソース・オペランドは、1バイトのメモリ・ロケーションである。

CLFLUSH が使用できるかどうかは、CPUID 機能フラグ CLFSH の状態によって示される（EDX レジスタのビット 19 は「CPUID—CPU Identification」の項を参照）。CLFLUSH 命令の影響を受ける、アライメントの合ったキャッシュ・ラインのサイズも CPUID 命令で示される（EAX レジスタの初期値が 1 の場合は、EBX レジスタのビット 8 ～ビット 15 まで）。

この命令の影響を受けるキャッシュ・ラインを含むページのメモリ属性は、この命令の動作に影響を与えない。ただし、プロセッサは、見込み的な読み込みが許されるメモリタイプ（すなわち、WB、WC、WT メモリタイプ）が割り当てられたシステムメモリ領域から、自由にデータを見込み的にフェッチしてキャッシュに入れることができる。この見込み的な動作に対するヒントをプロセッサに提示するには、PREFETCH*h* 命令を使用する。この見込み的なフェッチ動作は、命令の実行には拘束されず、任意の時点で発生する。したがって、CLFLUSH 命令は、PREFETCH*h* 命令などの見込み的なフェッチ機構に対して順序付けされない（つまり、キャッシュ・ラインを参照する CLFLUSH 命令の実行の直前、実行中、または実行後に、データがキャッシュに見込み的にロードされる可能性がある）。

CLFLUSH は、MFENCE 命令によってのみ順序付けされる。他のフェンス操作命令、シリアル化命令、または他の CLFLUSH 命令による順序付けは保証されない。例えば、ソフトウェアは、MFENCE 命令を使用して、ライトバックに以前のストアが含まれるように保証できる。

CLFLUSH 命令は、すべての特権レベルで使用できる。CLFLUSH 命令には、バイトロードに関連するすべてのパーミッション・チェックとフォルトが適用される（さらに、CLFLUSH 命令は、実行専用セグメント内のリニアアドレスをフラッシュすることが許される）。ロードと同じように、CLFLUSH 命令は、ページテーブル内の A ビットをセットし、D ビットはセットしない。

CLFLUSH—Cache Line Flush（続き）

CLFLUSH 命令は、SSE2 拡張命令に対して導入されたものである。ただし、独自の CPUID 機能フラグを持つため、SSE2 拡張命令が組み込まれていない IA-32 プロセッサに実装することができる。また、CPUID 命令を使って SSE2 拡張命令が存在することが検出されても、プロセッサに CLFLUSH 命令が実装されているかどうかは保証されない。

操作

```
Flush_Cache_Line(SRC)
```

同等のインテル® C/C++ コンパイラ組み込み関数

CLFLUSH　　void_mm_clflush(void const *p)

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#UD	CPUID 機能フラグ CLFSH が 0 の場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#UD	CPUID 機能フラグ CLFSH が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CLI—Clear Interrupt Flag

オペコード	命令	説明
FA	CLI	割り込みフラグをクリアする。割り込みフラグがクリアされているときは割り込みはディスエーブルにされる。

説明

保護モード仮想割り込みがイネーブルでない場合は、EFLAGS レジスタ内の IF フラグをクリアする。その他のフラグは影響を受けない。IF フラグをクリアすると、プロセッサはマスク可能な外部割り込みを無視する。IF フラグと CLI および STI 命令は、例外および NMI 割り込みの発生には関係しない。

保護モード仮想割り込みがイネーブルの場合は、CPL は 3 であり、IOPL は 3 より小さい。CLI 命令は、EFLAGS レジスタの VIF フラグをクリアし、IF フラグには影響を与えない。

表 3-5 は、プロセッサの動作モードによる CLI 命令の処理と、そのとき実行中のプログラムまたはプロシージャの CPL および IOPL を示している。

表 3-5. CLI の結果のデシジョン・テーブル

PE	VM	IOPL	CPL	PVI	VIP	VME	CLI の結果
0	X	X	X	X	X	X	**IF = 0**
1	0	≥ CPL	X	X	X	X	**IF = 0**
1	0	< CPL	3	1	X	X	**VIF = 0**
1	0	< CPL	< 3	X	X	X	**GP フォルト**
1	0	< CPL	X	0	X	X	**GP フォルト**
1	1	3	X	X	X	X	**IF = 0**
1	1	< 3	X	X	X	1	**VIF = 0**
1	1	< 3	X	X	X	0	**GP フォルト**
X = この設定は影響を与えない。							

CLI—Clear Interrupt Flag（続き）

操作

```
IF PE = 0
    THEN
        IF ← 0; (* Reset Interrupt Flag *)
    ELSE
        IF VM = 0;
            THEN
                IF IOPL ≥ CPL
                    THEN
                        IF ← 0; (* Reset Interrupt Flag *)
                ELSE
                    IF ((IOPL < CPL) AND (CPL < 3) AND (PVI = 1))
                        THEN
                            VIF ← 0; (* Reset Virtual Interrupt Flag *)
                    ELSE
                        #GP(0);
                    FI;
                FI;
            ELSE
                IF IOPL = 3
                    THEN
                        IF ← 0; (* Reset Interrupt Flag *)
                    ELSE
                        IF (IOPL < 3) AND (VME = 1)
                            THEN
                                VIF ← 0; (* Reset Virtual Interrupt Flag *)
                            ELSE
                                #GP(0);
                        FI;
                FI;
        FI;
FI;
```

CLI—Clear Interrupt Flag（続き）

影響を受けるフラグ

保護モード仮想割り込みがイネーブルでない場合、CPL が IOPL に等しいか小さい場合は、IF フラグが 0 にセットされる。そうでない場合、IF フラグは変わらない。EFLAGS レジスタ内のその他のフラグは影響を受けない。

保護モード仮想割り込みがイネーブルの場合は、CPL は 3 であり、IOPL は 3 より小さい。CLI 命令は、EFLAGS レジスタの VIF フラグをクリアし、IF フラグには影響を与えない。

保護モード例外

#GP(0)	CPL が現在のプログラムまたはプロシージャの IOPL より大きい（特権が小さい）場合。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	CPL が現在のプログラムまたはプロシージャの IOPL より大きい（特権が小さい）場合。

CLTS—Clear Task-Switched Flag in CR0

オペコード	命令	説明
0F 06	CLTS	CR0 の TS フラグをクリアする。

説明

CR0 レジスタ内のタスクスイッチ（TS）フラグをクリアする。この命令は、オペレーティング・システム内で使用することを目的としている。これは CPL = 0 でのみ実行できる特権命令である。この命令は、保護モード向けの初期化を可能にするため、実アドレスモードで実行できるようになっている。

プロセッサはタスクスイッチが行われるたびに TS フラグをセットする。このフラグは、マルチタスキング・アプリケーションでの FPU コンテキストのセーブを同期させるために使用される。このフラグの詳細については、『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、下巻』の第 2 章の「制御レジスタ」の TS フラグに関する説明を参照のこと。

操作

CR0(TS) ← 0;

影響を受けるフラグ

CR0 レジスタ内の TS フラグがクリアされる。

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	CLTS は、仮想 8086 モードでは認識されない。

CMC—Complement Carry Flag

オペコード	命令	説明
F5	CMC	CF フラグの状態を反転する。

説明

EFLAGS レジスタ内の CF フラグの状態を反転する。

操作

CF ← NOT CF;

影響を受けるフラグ

CF フラグの内容がその元の値の補数になる。OF、ZF、SF、AF、および PF フラグは影響を受けない。

例外（すべての操作モード）

なし。

CMOV*cc*—Conditional Move

オペコード	命令	説明
0F 47 /*r*	CMOVA *r16*, *r/m16*	より上（CF=0 および ZF=0）の場合転送する。
0F 47 /*r*	CMOVA *r32*, *r/m32*	より上（CF=0 および ZF=0）の場合転送する。
0F 43 /*r*	CMOVAE *r16*, *r/m16*	より上か等しい（CF=0）場合転送する。
0F 43 /*r*	CMOVAE *r32*, *r/m32*	より上か等しい（CF=0）場合転送する。
0F 42 /*r*	CMOVB *r16*, *r/m16*	より下（CF=1）の場合転送する。
0F 42 /*r*	CMOVB *r32*, *r/m32*	より下（CF=1）の場合転送する。
0F 46 /*r*	CMOVBE *r16*, *r/m16*	より下か等しい（CF=1 または ZF=1）場合転送する。
0F 46 /*r*	CMOVBE *r32*, *r/m32*	より下か等しい（CF=1 または ZF=1）場合転送する。
0F 42 /*r*	CMOVC *r16*, *r/m16*	キャリーがある（CF=1）場合転送する。
0F 42 /*r*	CMOVC *r32*, *r/m32*	キャリーがある（CF=1）場合転送する。
0F 44 /*r*	CMOVE *r16*, *r/m16*	等しい（ZF=1）場合転送する。
0F 44 /*r*	CMOVE *r32*, *r/m32*	等しい（ZF=1）場合転送する。
0F 4F /*r*	CMOVG *r16*, *r/m16*	より大きい（ZF=0 および SF=OF）場合転送する。
0F 4F /*r*	CMOVG *r32*, *r/m32*	より大きい（ZF=0 および SF=OF）場合転送する。
0F 4D /*r*	CMOVGE *r16*, *r/m16*	より大きいか等しい（SF=OF）場合転送する。
0F 4D /*r*	CMOVGE *r32*, *r/m32*	より大きいか等しい（SF=OF）場合転送する。
0F 4C /*r*	CMOVL *r16*, *r/m16*	より小さい（SF<>OF）場合転送する。
0F 4C /*r*	CMOVL *r32*, *r/m32*	より小さい（SF<>OF）場合転送する。
0F 4E /*r*	CMOVLE *r16*, *r/m16*	より小さいか等しい（ZF=1 または SF<>OF）場合転送する。
0F 4E /*r*	CMOVLE *r32*, *r/m32*	より小さいか等しい（ZF=1 または SF<>OF）場合転送する。
0F 46 /*r*	CMOVNA *r16*, *r/m16*	より上でない（CF=1 または ZF=1）場合転送する。
0F 46 /*r*	CMOVNA *r32*, *r/m32*	より上でない（CF=1 または ZF=1）場合転送する。
0F 42 /*r*	CMOVNAE *r16*, *r/m16*	より上でなく等しくない（CF=1）場合転送する。
0F 42 /*r*	CMOVNAE *r32*, *r/m32*	より上でなく等しくない（CF=1）場合転送する。
0F 43 /*r*	CMOVNB *r16*, *r/m16*	より下でない（CF=0）場合転送する。
0F 43 /*r*	CMOVNB *r32*, *r/m32*	より下でない（CF=0）場合転送する。
0F 47 /*r*	CMOVNBE *r16*, *r/m16*	より下でなく等しくない（CF=0 および ZF=0）場合転送する。
0F 47 /*r*	CMOVNBE *r32*, *r/m32*	より下でなく等しくない（CF=0 および ZF=0）場合転送する。
0F 43 /*r*	CMOVNC *r16*, *r/m16*	キャリーがない（CF=0）場合転送する。
0F 43 /*r*	CMOVNC *r32*, *r/m32*	キャリーがない（CF=0）場合転送する。
0F 45 /*r*	CMOVNE *r16*, *r/m16*	等しくない（ZF=0）場合転送する。
0F 45 /*r*	CMOVNE *r32*, *r/m32*	等しくない（ZF=0）場合転送する。
0F 4E /*r*	CMOVNG *r16*, *r/m16*	より大きくない（ZF=1 または SF<>OF）場合転送する。
0F 4E /*r*	CMOVNG *r32*, *r/m32*	より大きくない（ZF=1 または SF<>OF）場合転送する。
0F 4C /*r*	CMOVNGE *r16*, *r/m16*	より大きくなく等しくない（SF<>OF）場合転送する。
0F 4C /*r*	CMOVNGE *r32*, *r/m32*	より大きくなく等しくない（SF<>OF）場合転送する。

CMOVcc—Conditional Move（続き）

オペコード	命令	説明
0F 4D /*r*	CMOVNL *r16*, *r/m16*	より小さくない（SF=OF）場合転送する。
0F 4D /*r*	CMOVNL *r32*, *r/m32*	より小さくない（SF=OF）場合転送する。
0F 4F /*r*	CMOVNLE *r16*, *r/m16*	より小さくなく等しくない（ZF=0 および SF=OF）場合転送する。
0F 4F /*r*	CMOVNLE *r32*, *r/m32*	より小さくなく等しくない（ZF=0 および SF=OF）場合転送する。
0F 41 /*r*	CMOVNO *r16*, *r/m16*	オーバーフローがない（OF=0）場合転送する。
0F 41 /*r*	CMOVNO *r32*, *r/m32*	オーバーフローがない（OF=0）場合転送する。
0F 4B /*r*	CMOVNP *r16*, *r/m16*	パリティがない（PF=0）場合転送する。
0F 4B /*r*	CMOVNP *r32*, *r/m32*	パリティがない（PF=0）場合転送する。
0F 49 /*r*	CMOVNS *r16*, *r/m16*	符号がない（SF=0）場合転送する。
0F 49 /*r*	CMOVNS *r32*, *r/m32*	符号がない（SF=0）場合転送する。
0F 45 /*r*	CMOVNZ *r16*, *r/m16*	ゼロでない（ZF=0）場合転送する。
0F 45 /*r*	CMOVNZ *r32*, *r/m32*	ゼロでない（ZF=0）場合転送する。
0F 40 /*r*	CMOVO *r16*, *r/m16*	オーバーフローがある（OF=0）場合転送する。
0F 40 /*r*	CMOVO *r32*, *r/m32*	オーバーフローがある（OF=0 場合転送する。
0F 4A /*r*	CMOVP *r16*, *r/m16*	パリティがある（PF=1）場合転送する。
0F 4A /*r*	CMOVP *r32*, *r/m32*	パリティがある（PF=1）場合転送する。
0F 4A /*r*	CMOVPE *r16*, *r/m16*	パリティが偶数（PF=1）の場合転送する。
0F 4A /*r*	CMOVPE *r32*, *r/m32*	パリティが偶数（PF=1）の場合転送する。
0F 4B /*r*	CMOVPO *r16*, *r/m16*	パリティが奇数（PF=0）の場合転送する。
0F 4B /*r*	CMOVPO *r32*, *r/m32*	パリティが奇数（PF=0）の場合転送する。
0F 48 /*r*	CMOVS *r16*, *r/m16*	符号がある（SF=1）場合転送する。
0F 48 /*r*	CMOVS *r32*, *r/m32*	符号がある（SF=1）場合転送する。
0F 44 /*r*	CMOVZ *r16*, *r/m16*	ゼロ（ZF=1）の場合転送する。
0F 44 /*r*	CMOVZ *r32*, *r/m32*	ゼロ（ZF=1）の場合転送する。

CMOV*cc*—Conditional Move（続き）

説明

CMOV*cc* 命令は、EFLAGS レジスタ内のステータス・フラグ（CF、OF、PF、SF、ZF）の 1 つ以上の状態を調べ、それらのフラグが指定された状態（または条件）の場合、転送操作を実行する。各命令ごとに特定の条件コード（*cc*）が対応しており、テスト対象の条件を示している。条件が満たされなかった場合は、転送は行われず、CMOV*cc* 命令の次の命令から実行が継続される。

これらの命令では、メモリからいずれかの汎用レジスタに、または 1 つの汎用レジスタから他の汎用レジスタに 16 または 32 ビットの値を転送する。8 ビット・レジスタ・オペランドの条件付き転送はサポートされていない。

各 CMOV*cc* ニーモニックの条件は、上記の表の説明欄に示してある。「より小さい」および「より大きい」という表現は、符号付き整数の比較に使用され、「より上」および「より下」という表現は符号なし整数の比較に使用されている。

ステータス・フラグの特定の状態はときとして 2 種類に解釈されることがあるので、一部のオペコードに対しては 2 つのニーモニックが定義されている。例えば、CMOVA（より上条件付き転送）命令と CMOVNBE（より下でなく等しくない条件付き転送）命令とは、同一のオペコード 0F 47H に対する 2 つのニーモニックである。

CMOVcc 命令は P6 ファミリ・プロセッサに導入されたが、これらの命令はすべての IA-32 プロセッサでサポートされているわけではない。CMOV*cc* 命令がサポートされているかどうかは、CPUID 命令でプロセッサの機能情報を調べることにより判定できる（本章の「COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS」を参照）。

操作

```
temp ← SRC
IF condition TRUE
    THEN
        DEST ← temp
FI;
```

影響を受けるフラグ

なし。

CMOV*cc*—Conditional Move（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

CMP—Compare Two Operands

オペコード	命令	説明
3C *ib*	CMP AL, *imm8*	*imm8* を AL と比較する。
3D *iw*	CMP AX, *imm16*	*imm16* を AX と比較する。
3D *id*	CMP EAX, *imm32*	*imm32* を EAX と比較する。
80 /7 *ib*	CMP *r/m8*, *imm8*	*imm8* を *r/m8* と比較する。
81 /7 *iw*	CMP *r/m16*, *imm16*	*imm16* を *r/m16* と比較する。
81 /7 *id*	CMP *r/m32*, *imm32*	*imm32* を *r/m32* と比較する。
83 /7 *ib*	CMP *r/m16*, *imm8*	*imm8* を *r/m16* と比較する。
83 /7 *ib*	CMP *r/m32*, *imm8*	*imm8* を *r/m32* と比較する。
38 /*r*	CMP *r/m8*, *r8*	*r8* を *r/m8* と比較する。
39 /*r*	CMP *r/m16*, *r16*	*r16* を *r/m16* と比較する。
39 /*r*	CMP *r/m32*, *r32*	*r32* を *r/m32* と比較する。
3A /*r*	CMP *r8*, *r/m8*	*r/m8* を *r8* と比較する。
3B /*r*	CMP *r16*, *r/m16*	*r/m16* を *r16* と比較する。
3B /*r*	CMP *r32*, *r/m32*	*r/m32* を *r32* と比較する。

説明

第1ソース・オペランドを第2ソース・オペランドと比較し、結果にしたがってEFLAGSレジスタ内のステータス・フラグをセットする。比較は、第1オペランドから第2オペランドを引き、次にSUB命令の場合と同様にステータス・フラグをセットして行われる。オペランドとして即値を使用した場合は、そのオペランドは第1オペランドの長さに符号拡張される。

CMP命令は、一般的に条件付きジャンプ（J*cc*）、条件付き転送（CMOV*cc*）、またはSET*cc*命令と併用される。J*cc*、CMOV*cc*、SET*cc*命令が使用する条件コードはCMP命令の結果に基づく。『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B「EFLAGS条件コード」に、ステータス・フラグと条件コードとの関係が示してある。

操作

```
temp ← SRC1 – SignExtend(SRC2);
ModifyStatusFlags; (* Modify status flags in the same manner as the SUB instruction*)
```

影響を受けるフラグ

CF、OF、SF、ZF、AF、PFフラグが結果にしたがってセットされる。

CMP—Compare Two Operands（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

CMPPD—Compare Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F C2 /r ib	CMPPD *xmm1*, *xmm2/m128*, *imm8*	*imm8* を比較プレディケートとして使用して、*xmm2/m128* のパック ド倍精度浮動小数点値と *xmm1*のパック ド倍精度浮動小数点値を比較する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値の SIMD 比較を実行し、比較の結果をデスティネーション・オペランドに返す。比較プレディケート・オペランド（第 3 オペランド）は、パックド値の各ペアに対して実行される比較のタイプを指定する。各比較の結果は、すべて 1（比較は真）またはすべて 0（比較は偽）のクワッドワード・マスクになる。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。比較プレディケート・オペランドは 8 ビット即値であり、実行される比較のタイプを最初の 3 ビットで定義する（表 3-6. を参照）。即値のビット 4 ～ 7 は予約済みである。

表 3-6. CMPPD 命令と CMPPS 命令の比較プレディケート

プレディケート	imm8 のエンコーディング	説明	関係 : A は第 1 オペランド B は第 2 オペランド	エミュレーション	NaN オペランドの場合の結果	QNaN オペランドの場合に無効が報告されるか
EQ	000B	等しい	A = B		偽	いいえ
LT	001B	より小さい	A < B		偽	はい
LE	010B	より小さいか等しい	A ≤ B		偽	はい
		より大きい	A > B	オペランドを入れ替えて、LT を使用する。	偽	はい
		より大きいか等しい	A ≥ B	オペランドを入れ替えて、LE を使用する。	偽	はい
UNORD	011B	アンオーダー	A、B= アンオーダー		真	いいえ
NEQ	100B	等しくない	A ≠ B		真	いいえ
NLT	101B	より小さくない	NOT（A < B）		真	はい
NLE	110B	より小さくなく等しくない	NOT（A ≤ B）		真	はい
		より大きくない	NOT（A > B）	オペランドを入れ替えて、NLT を使用する。	真	はい
		より大きくなく等しくない	NOT（A ≥ B）	オペランドを入れ替えて、NLE を使用する。	真	はい
ORD	111B	オーダー	A、B= オーダー		偽	いいえ

CMPPD—Compare Packed Double-Precision Floating-Point Values（続き）

2 つの比較対象ソース・オペランドのうち、少なくとも 1 つが NaN である場合、アンオーダー関係は真となる。ソース・オペランドが NaN でない場合は、オーダー関係は真となる。

この命令の後に、結果として得られたデスティネーション・オペランド内のマスクを入力オペランドとして使用して計算命令を実行しても、例外は発生しない。これは、すべて 0 のマスクは浮動小数点値 +0.0 に対応し、すべて 1 のマスクは QNaN に対応するためである。

プロセッサには、「より大きい」、「より大きいか等しい」、「より大きくない」、「より大きくなく等しくない」の関係は実装されていない。これらの比較を行うには、逆の関係を利用するか（すなわち、「より大きい」比較を行うには「より小さくなく等しくない」を使用する）、またはソフトウェア・エミュレーションを利用する。ソフトウェア・エミュレーションを使用する場合、プログラム上では、ソース・オペランドとデスティネーション・オペランドを入れ替え、必要に応じてレジスタをコピーして、新たにデスティネーションに入るデータを保護し、さらに、異なるプレディケートを使って比較を実行しなければならない。これらのエミュレーションに使用されるプレディケートは、表 3-6. の「エミュレーション」の項目に記載されている。

コンパイラとアセンブラは、3 オペランドの CMPPD 命令以外に、以下の 2 オペランドの疑似演算をサポートできる。

表 3-7. 疑似演算と対応する CMPPD 命令

疑似演算	対応する CMPPD 命令
CMPEQPD xmm1, xmm2	CMPPD xmm1, xmm2, 0
CMPLTPD xmm1, xmm2	CMPPD xmm1, xmm2, 1
CMPLEPD xmm1, xmm2	CMPPD xmm1, xmm2, 2
CMPUNORDPD xmm1, xmm2	CMPPD xmm1, xmm2, 3
CMPNEQPD xmm1, xmm2	CMPPD xmm1, xmm2, 4
CMPNLTPD xmm1, xmm2	CMPPD xmm1, xmm2, 5
CMPNLEPD xmm1, xmm2	CMPPD xmm1, xmm2, 6
CMPORDPD xmm1, xmm2	CMPPD xmm1, xmm2, 7

CMPPD—Compare Packed Double-Precision Floating-Point Values（続き）

「より大きい」の関係は、プロセッサが用意していないため、2つ以上の命令を使用してソフトウェア的にエミュレートする必要がある。したがって、これらの関係は疑似演算としてはサポートされない。「より大きい」の条件で比較する場合は、プログラマは、それに対応する「より小さい」の関係のソース・オペランドとデスティネーション・オペランドを入れ替え、移動命令を使用してマスクを適切なデスティネーション・レジスタに移動し、ソース・オペランドが影響を受けないようにする必要がある。

操作

```
CASE (COMPARISON PREDICATE) OF
    0:  OP ← EQ;
    1:  OP ← LT;
    2:  OP ← LE;
    3:  OP ← UNORD;
    4:  OP ← NEQ;
    5:  OP ← NLT;
    6:  OP ← NLE;
    7:  OP ← ORD;
    DEFAULT:    Reserved;
CMP0 ← DEST[63-0]  OP  SRC[63-0];
CMP1 ← DEST[127-64]  OP  SRC[127-64];
IF CMP0 = TRUE
    THEN DEST[63-0] ← FFFFFFFFFFFFFFFFH
    ELSE DEST[63-0] ← 0000000000000000H; FI;
IF CMP1 = TRUE
    THEN DEST[127-64] ← FFFFFFFFFFFFFFFFH
    ELSE DEST[127-64] ← 0000000000000000H; FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CMPPD「等しい」の条件で比較。__m128d _mm_cmpeq_pd(__m128d a, __m128d b)

CMPPD「より小さい」の条件で比較。
__m128d _mm_cmplt_pd(__m128d a, __m128d b)

CMPPD「より小さいか等しい」の条件で比較。
__m128d _mm_cmple_pd(__m128d a, __m128d b)

CMPPD「より大きい」の条件で比較。
__m128d _mm_cmpgt_pd(__m128d a, __m128d b)

CMPPD「より大きいか等しい」の条件で比較。
__m128d _mm_cmpge_pd(__m128d a, __m128d b)

CMPPD「等しくない」の条件で比較。
__m128d _mm_cmpneq_pd(__m128d a, __m128d b)

CMPPD—Compare Packed Double-Precision Floating-Point Values（続き）

CMPPD「より小さくない」の条件で比較。
__m128d _mm_cmpnlt_pd(__m128d a, __m128d b)

CMPPD「より大きくない」の条件で比較。
__m128d _mm_cmpngt_pd(__m128d a, __m128d b)

CMPPD「より大きくなく等しくない」の条件で比較。
__m128d _mm_cmpnge_pd(__m128d a, __m128d b)

CMPPD「オーダー」の条件で比較。
__m128d _mm_cmpord_pd(__m128d a, __m128d b)

CMPPD「アンオーダー」の条件で比較。
__m128d _mm_cmpunord_pd(__m128d a, __m128d b)

CMPPD「より小さくなく等しくない」の条件で比較。
__m128d _mm_cmpnle_pd(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（SNaN オペランドの場合）、無効（QNaN と上記の表のプレディケートの場合）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

CMPPD—Compare Packed Double-Precision Floating-Point Values（続き）

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CMPPS—Compare Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F C2 /r ib	CMPPS *xmm1*, *xmm2/m128*, *imm8*	*imm8* を比較プレディケートとして使用して、*xmm2/mem* のパックド単精度浮動小数点値と *xmm1* のパックド単精度浮動小数点値を比較する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 4 つのパックド単精度浮動小数点値の SIMD 比較を実行し、比較の結果をデスティネーション・オペランドに返す。比較プレディケート・オペランド（第 3 オペランド）は、パックド値の各ペアに対して実行される比較のタイプを指定する。比較の結果は、すべて 1（比較は真）またはすべて 0（比較は偽）の 4 つのダブルワード・マスクになる。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。比較プレディケート・オペランドは 8 ビット即値であり、実行される比較のタイプを最初の 3 ビットで定義する（表 3-6. を参照）。即値のビット 4 ～ 7 は予約済みである。

2 つの比較対象ソース・オペランドのうち、少なくとも 1 つが NaN である場合、アンオーダー関係は真となる。ソース・オペランドが NaN でない場合は、オーダー関係は真となる。

この命令の後に、デスティネーション・オペランド内の結果のマスクを入力オペランドとして使用する計算命令を実行しても、フォルトは発生しない。これは、すべて 0 のマスクは +0.0 の浮動小数点値に対応し、すべて 1 のマスクは QNaN に対応するためである。

表 3-6. に示した一部の比較（「より大きい」、「より大きいか等しい」、「より大きくない」、「より大きくなく等しくない」の関係）は、ソフトウェア・エミュレーションでしか実行できない。これらの比較を行う場合、プログラムは、ソース・オペランドとデスティネーション・オペランドを入れ替え（必要に応じてレジスタをコピーして、新たにデスティネーションに入るデータを保護し）、異なるプレディケートを使って比較を実行しなければならない。これらのエミュレーションに使用されるプレディケートは、表 3-6. の「エミュレーション」の項目に記載されている。

CMPPS—Compare Packed Single-Precision Floating-Point Values（続き）

コンパイラとアセンブラは、3オペランドのCMPPS命令以外に、次の2オペランドの疑似演算をサポートする必要がある。

疑似演算	対応する CMPPS 命令
CMPEQPS xmm1, xmm2	CMPPS xmm1, xmm2, 0
CMPLTPS xmm1, xmm2	CMPPS xmm1, xmm2, 1
CMPLEPS xmm1, xmm2	CMPPS xmm1, xmm2, 2
CMPUNORDPS xmm1, xmm2	CMPPS xmm1, xmm2, 3
CMPNEQPS xmm1, xmm2	CMPPS xmm1, xmm2, 4
CMPNLTPS xmm1, xmm2	CMPPS xmm1, xmm2, 5
CMPNLEPS xmm1, xmm2	CMPPS xmm1, xmm2, 6
CMPORDPS xmm1, xmm2	CMPPS xmm1, xmm2, 7

「より大きい」の関係は、ハードウェア上で用意されていないため、2つ以上の命令を使用してソフトウェア的にエミュレートする必要がある。したがって、これらの関係は疑似演算としてはサポートされない。「より大きい」の条件で比較する場合は、プログラマが、対応する「より小さい」の関係のソース・オペランドとデスティネーション・オペランドを入れ替え、ソース・オペランドが影響を受けないように、移動命令を使用してマスクを適切なデスティネーション・レジスタに移動しなければならない。

操作

```
CASE (COMPARISON PREDICATE) OF
    0:  OP ← EQ;
    1:  OP ← LT;
    2:  OP ← LE;
    3:  OP ← UNORD;
    4:  OP ← NE;
    5:  OP ← NLT;
    6:  OP ← NLE;
    7:  OP ← ORD;
EASC
CMP0 ← DEST[31-0]  OP  SRC[31-0];
CMP1 ← DEST[63-32]  OP  SRC[63-32];
CMP2 ← DEST [95-64]  OP  SRC[95-64];
CMP3 ← DEST[127-96]  OP  SRC[127-96];
IF CMP0 = TRUE
    THEN DEST[31-0] ← FFFFFFFFH
    ELSE DEST[31-0] ← 00000000H; FI;
IF CMP1 = TRUE
    THEN DEST[63-32] ← FFFFFFFFH
    ELSE DEST[63-32] ← 00000000H; FI;
```

CMPPS—Compare Packed Single-Precision Floating-Point Values（続き）

```
IF CMP2 = TRUE
    THEN DEST95-64] ← FFFFFFFFH
    ELSE DEST[95-64] ← 00000000H; FI;
IF CMP3 = TRUE
    THEN DEST[127-96] ← FFFFFFFFH
    ELSE DEST[127-96] ← 00000000H; FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CMPPS「等しい」の条件で比較。 __m128 _mm_cmpeq_ps(__m128 a, __m128 b)

CMPPS「より小さい」の条件で比較。
__m128 _mm_cmplt_ps(__m128 a, __m128 b)

CMPPS「より小さいか等しい」の条件で比較。
__m128 _mm_cmple_ps(__m128 a, __m128 b)

CMPPS「より大きい」の条件で比較。
__m128 _mm_cmpgt_ps(__m128 a, __m128 b)

CMPPS「より大きいか等しい」の条件で比較。
__m128 _mm_cmpge_ps(__m128 a, __m128 b)

CMPPS「等しくない」の条件で比較。
__m128 _mm_cmpneq_ps(__m128 a, __m128 b)

CMPPS「より小さくない」の条件で比較。
__m128 _mm_cmpnlt_ps(__m128 a, __m128 b)

CMPPS「より大きくない」の条件で比較。
__m128 _mm_cmpngt_ps(__m128 a, __m128 b)

CMPPS「より大きくなく等しくない」の条件で比較。
__m128 _mm_cmpnge_ps(__m128 a, __m128 b)

CMPPS「オーダー」の条件で比較。
__m128 _mm_cmpord_ps(__m128 a, __m128 b)

CMPPS「アンオーダー」の条件で比較。
__m128 _mm_cmpunord_ps(__m128 a, __m128 b)

CMPPS「より小さくなく等しくない」の条件で比較。
__m128 _mm_cmpnle_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

無効（SNaN オペランドの場合）、無効（QNaN と上記の表のプレディケートの場合）、デノーマル。

CMPPS—Compare Packed Single-Precision Floating-Point Values（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CMPS/CMPSB/CMPSW/CMPSD—Compare String Operands

オペコード	命令	説明
A6	CMPS m8, m8	アドレス DS:(E)SI のバイトをアドレス ES:(E)DI のバイトと比較し、結果にしたがってステータス・フラグをセットする。
A7	CMPS m16, m16	アドレス DS:(E)SI のワードをアドレス ES:(E)DI のワードと比較し、結果にしたがってステータス・フラグをセットする。
A7	CMPS m32, m32	アドレス DS:(E)SI のダブルワードをアドレス ES:(E)DI のダブルワードと比較し、結果にしたがってステータス・フラグをセットする。
A6	CMPSB	アドレス DS:(E)SI のバイトをアドレス ES:(E)DI のバイトと比較し、結果にしたがってステータス・フラグをセットする。
A7	CMPSW	アドレス DS:(E)SI のワードをアドレス ES:(E)DI のワードと比較し、結果にしたがってステータス・フラグをセットする。
A7	CMPSD	アドレス DS:(E)SI のダブルワードをアドレス ES:(E)DI のダブルワードと比較し、結果にしたがってステータス・フラグをセットする。

説明

第 1 ソース・オペランドで指定されるバイト、ワード、またはダブルワードを第 2 ソース・オペランドで指定されるバイト、ワード、またはダブルワードと比較し、結果にしたがって EFLAGS レジスタ内のステータス・フラグをセットする。両ソース・オペランドはメモリにある。第 1 ソース・オペランドのアドレスは、（命令のそれぞれ 32 または 16 ビットのアドレスサイズ属性によって）DS:ESI、DS:SI のいずれかのレジスタから読み取られる。第 2 ソース・オペランドのアドレスは、（やはり命令のアドレスサイズ属性によって）ES:EDI、ES:DI のいずれかのレジスタから読み取られる。DS セグメントはセグメント・オーバーライド・プリフィックスでオーバーライドすることもできるが、ES セグメントはオーバーライドできない。

アセンブリ・コード・レベルでは、この命令の「明示オペランド」形式と「オペランドなし」形式という 2 つの形式が使用できる。明示オペランド形式（CMPS ニーモニックで指定する）では、2 つのソース・オペランドを明示的に指定できる。その場合、両ソース・オペランドは両ソース値のサイズとロケーションを示す記号とする。この明示オペランド形式はドキュメンテーションを可能にするために設けられたものであるが、この形式によって提供されるドキュメンテーションは誤解を招く場合があるので注意する。すなわち、両ソース・オペランド記号は正しい**タイプ**（サイズ）のオペランド（バイト、ワード、またはダブルワード）を指定しなければならないが、正しい**ロケーション**は指定しなくてもよい。両ソース・オペランドのロケーションは常に DS:(E)SI と ES:(E)DI によって指定されるので、ストリング比較命令を実行する前に、それらのレジスタに正しくロードしなければならない。

CMPS/CMPSB/CMPSW/CMPSD—Compare String Operands（続き）

オペランドなし形式は、「ショート形式」のバイト、ワード、ダブルワード各バージョンのCMPS命令を提供する。この場合も、プロセッサはDS:(E)SIおよびES:(E)DIレジスタをソース・オペランドのロケーションを指定するものとみなす。ソース・オペランドのサイズは、CMPSB（バイト比較）、CMPSW（ワード比較）、またはCMPSD（ダブルワード比較）の各ニーモニックで選択される。

比較の後、EFLAGSレジスタ内のDFフラグの設定にしたがって、(E)SIおよび(E)DIレジスタが自動的にインクリメントまたはデクリメントされる。（DFフラグが0の場合は(E)SIおよび(E)DIレジスタはインクリメントされ、DFフラグが1の場合はデクリメントされる。）これらのレジスタは、バイト操作の場合は1、ワード操作の場合は2、ダブルワード操作の場合は4それぞれインクリメントまたはデクリメントされる。

CMPS、CMPSB、CMPSW、CMPSD命令は、前にREPプリフィックスを付けることにより、ECXバイト、ワード、またはダブルワードのブロック比較を行うことができる。ただし、これらの命令は、次の比較が行われる前にステータスの設定に基づいてなんらかの処置を行うLOOP構成体で使用されることの方が多い。REPプリフィックスについては、本章の「REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix」を参照のこと。

操作

```
temp ←SRC1 − SRC2;
SetStatusFlags(temp);
IF (byte comparison)
    THEN IF DF = 0
        THEN
            (E)SI ← (E)SI + 1;
            (E)DI ← (E)DI + 1;
        ELSE
            (E)SI ← (E)SI – 1;
            (E)DI ← (E)DI – 1;
        FI;
    ELSE IF (word comparison)
        THEN IF DF = 0
            (E)SI ← (E)SI + 2;
            (E)DI ← (E)DI + 2;
        ELSE
            (E)SI ← (E)SI – 2;
            (E)DI ← (E)DI – 2;
        FI;
```

CMPS/CMPSB/CMPSW/CMPSD—Compare String Operands（続き）

```
    ELSE (* doubleword comparison*)
        THEN IF DF = 0
            (E)SI ← (E)SI + 4;
            (E)DI ← (E)DI + 4;
        ELSE
            (E)SI ← (E)SI – 4;
            (E)DI ← (E)DI – 4;
        FI;
FI;
```

影響を受けるフラグ

CF、OF、SF、ZF、AF、PF フラグが比較の一時的結果にしたがってセットされる。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

CMPSD—Compare Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F C2 /r ib	CMPSD *xmm1*, *xmm2/m64*, *imm8*	*imm8* を比較プレディケートとして使用して、*xmm2/m64* の下位の倍精度浮動小数点値と *xmm1* の下位の倍精度浮動小数点値を比較する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の下位の倍精度浮動小数点値を比較し、比較の結果をデスティネーション・オペランドに返す。比較プレディケート・オペランド（第 3 オペランド）は、実行される比較のタイプを指定する。比較の結果は、すべて 1（比較は真）またはすべて 0（比較は偽）のクワッドワード・マスクになる。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードは変更されない。比較プレディケート・オペランドは 8 ビット即値であり、実行される比較のタイプを最初の 3 ビットで定義する（表 3-6. を参照）。即値のビット 4 ～ 7 は予約済みである。

2 つの比較対象ソース・オペランドのうち、少なくとも 1 つが NaN である場合、アンオーダー関係は真となる。ソース・オペランドが NaN でない場合は、オーダー関係は真となる。

この命令の後に、結果として得られたデスティネーション・オペランド内のマスクを入力オペランドとして使用して計算命令を実行しても、フォルトは発生しない。これは、すべて 0 のマスクは浮動小数点値 +0.0 に対応し、すべて 1 のマスクは QNaN に対応するためである。

表 3-6. に示した一部の比較は、ソフトウェア・エミュレーションでしか実行できない。これらの比較を行う場合、プログラムは、ソース・オペランドとデスティネーション・オペランドを入れ替え（必要に応じてレジスタをコピーして、新たにデスティネーション・オペランドに入るデータを保護し）、異なるプレディケートを使って比較を実行しなければならない。これらのエミュレーションに使用されるプレディケートは、表 3-6. の「エミュレーション」の項目に記載されている。

CMPSD—Compare Scalar Double-Precision Floating-Point Value（続き）

コンパイラとアセンブラは、3 オペランドの CMPSD 命令以外に、以下の 2 オペランドの疑似演算をサポートできる。

疑似演算	対応する CMPSD 命令
CMPEQSD xmm1, xmm2	CMPSD xmm1, xmm2, 0
CMPLTSD xmm1, xmm2	CMPSD xmm1, xmm2, 1
CMPLESD xmm1, xmm2	CMPSD xmm1, xmm2, 2
CMPUNORDSD xmm1, xmm2	CMPSD xmm1, xmm2, 3
CMPNEQSD xmm1, xmm2	CMPSD xmm1, xmm2, 4
CMPNLTSD xmm1, xmm2	CMPSD xmm1, xmm2, 5
CMPNLESD xmm1, xmm2	CMPSD xmm1, xmm2, 6
CMPORDSD xmm1, xmm2	CMPSD xmm1, xmm2, 7

「より大きい」の関係は、ハードウェア上で用意されていないため、2 つ以上の命令を使用してソフトウェア的にエミュレートする必要がある。したがって、これらの関係は疑似演算としてはサポートされない。「より大きい」の条件で比較する場合は、プログラマは、それに対応する「より小さい」の関係のソース・オペランドとデスティネーション・オペランドを入れ替え、移動命令を使用してマスクを適切なデスティネーション・レジスタに移動し、ソース・オペランドが影響を受けないようにする必要がある。

操作

```
CASE (COMPARISON PREDICATE) OF
    0:  OP ← EQ;
    1:  OP ← LT;
    2:  OP ← LE;
    3:  OP ← UNORD;
    4:  OP ← NEQ;
    5:  OP ← NLT;
    6:  OP ← NLE;
    7:  OP ← ORD;
    DEFAULT: Reserved;
CMP0 ← DEST[63-0]  OP  SRC[63-0];
IF CMP0 = TRUE
    THEN DEST[63-0] ← FFFFFFFFFFFFFFFFH
    ELSE DEST[63-0] ← 0000000000000000H; FI;
* DEST[127-64] remains unchanged *;
```

CMPSD—Compare Scalar Double-Precision Floating-Point Value（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

CMPSD「等しい」の条件で比較。__m128d _mm_cmpeq_sd(__m128d a, __m128d b)

CMPSD「より小さい」の条件で比較。
__m128d _mm_cmplt_sd(__m128d a, __m128d b)

CMPSD「より小さいか等しい」の条件で比較。
__m128d _mm_cmple_sd(__m128d a, __m128d b)

CMPSD「より大きい」の条件で比較。
__m128d _mm_cmpgt_sd(__m128d a, __m128d b)

CMPSD「より大きいか等しい」の条件で比較。
__m128d _mm_cmpge_sd(__m128d a, __m128d b)

CMPSD「等しくない」の条件で比較。
__m128d _mm_cmpneq_sd(__m128d a, __m128d b)

CMPSD「より小さくない」の条件で比較。
__m128d _mm_cmpnlt_sd(__m128d a, __m128d b)

CMPSD「より大きくない」の条件で比較。
__m128d _mm_cmpngt_sd(__m128d a, __m128d b)

CMPSD「より大きくなく等しくない」の条件で比較。
__m128d _mm_cmpnge_sd(__m128d a, __m128d b)

CMPSD「オーダー」の条件で比較。__m128d _mm_cmpord_sd(__m128d a, __m128d b)

CMPSD「アンオーダー」の条件で比較。
__m128d _mm_cmpunord_sd(__m128d a, __m128d b)

CMPSD「より小さくなく等しくない」の条件で比較。
__m128d _mm_cmpnle_sd(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（SNaN オペランドの場合）、無効（QNaN と上記の表のプレディケートの場合）、デノーマル。

CMPSD—Compare Scalar Double-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CMPSS—Compare Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F C2 /r ib	CMPSS *xmm1*, *xmm2/m32*, *imm8*	*imm8* を比較プレディケートとして使用して、*xmm2/m32* の最下位の単精度浮動小数点値と *xmm1* の最下位の単精度浮動小数点値を比較する。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の最下位の単精度浮動小数点値を比較し、比較の結果をデスティネーション・オペランドに返す。比較プレディケート・オペランド（第 3 オペランド）は、実行される比較のタイプを指定する。比較の結果は、すべて 1（比較は真）またはすべて 0（比較は偽）のダブルワード・マスクになる。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。結果はデスティネーション・オペランドの最下位のダブルワードに格納され、上位 3 つのダブルワードは変更されない。比較プレディケート・オペランドは 8 ビット即値であり、実行される比較のタイプを最初の 3 ビットで定義する（表 3-6. を参照）。即値のビット 4 ～ 7 は予約済みである。

2 つの比較対象ソース・オペランドのうち、少なくとも 1 つが NaN である場合、アンオーダー関係は真となる。ソース・オペランドが NaN でない場合は、オーダー関係は真となる。

この命令の後に、結果として得られたデスティネーション・オペランド内のマスクを入力オペランドとして使用して計算命令を実行しても、フォルトは発生しない。これは、すべて 0 のマスクは浮動小数点値 +0.0 に対応し、すべて 1 のマスクは QNaN に対応するためである。

表 3-6. に示した比較の中には、ソフトウェア・エミュレーション以外では実行できないものがある。これらの比較を行う際は、プログラムが（必要に応じてレジスタをコピーしてデスティネーション・オペランドになるデータを保護し）、ソース・オペランドとデスティネーション・オペランドを入れ替えた後、異なる述語を使って比較を実行しなければならない。これらのエミュレーションに使用される述語は、表 3-6. の「エミュレーション」の欄に記載されている。

CMPSS—Compare Scalar Single-Precision Floating-Point Values（続き）

コンパイラとアセンブラは、3 オペランドの CMPSS 命令以外に、次の 2 オペランドの疑似演算をサポートする必要がある。

疑似演算	対応する CMPSS 命令
CMPEQSS xmm1, xmm2	CMPSS xmm1, xmm2, 0
CMPLTSS xmm1, xmm2	CMPSS xmm1, xmm2, 1
CMPLESS xmm1, xmm2	CMPSS xmm1, xmm2, 2
CMPUNORDSS xmm1, xmm2	CMPSS xmm1, xmm2, 3
CMPNEQSS xmm1, xmm2	CMPSS xmm1, xmm2, 4
CMPNLTSS xmm1, xmm2	CMPSS xmm1, xmm2, 5
CMPNLESS xmm1, xmm2	CMPSS xmm1, xmm2, 6
CMPORDSS xmm1, xmm2	CMPSS xmm1, xmm2, 7

「より大きい」の関係は、ハードウェア上で用意されていないため、2 つ以上の命令を使用してソフトウェア的にエミュレートする必要がある。したがって、これらの関係は疑似演算としてはサポートされない。「より大きい」の条件で比較する場合は、プログラマが、対応する「より小さい」の関係のソース・オペランドとデスティネーション・オペランドを入れ替え、ソース・オペランドが影響を受けないように、移動命令を使用してマスクを適切なデスティネーション・レジスタに確実に移動しなければならない。

操作

```
CASE (COMPARISON PREDICATE) OF
    0:  OP ← EQ;
    1:  OP ← LT;
    2:  OP ← LE;
    3:  OP ← UNORD;
    4:  OP ← NEQ;
    5:  OP ← NLT;
    6:  OP ← NLE;
    7:  OP ← ORD;
    DEFAULT: Reserved;
CMP0 ← DEST[31-0]  OP  SRC[31-0];
IF CMP0 = TRUE
    THEN DEST[31-0] ← FFFFFFFFH
    ELSE DEST[31-0] ← 00000000H; FI;
* DEST[127-32] remains unchanged *;
```

CMPSS—Compare Scalar Single-Precision Floating-Point Values（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

CMPSS「等しい」の条件で比較。__m128 _mm_cmpeq_ss(__m128 a, __m128 b)

CMPSS「より小さい」の条件で比較。
__m128 _mm_cmplt_ss(__m128 a, __m128 b)

CMPSS「より小さいか等しい」の条件で比較。
__m128 _mm_cmple_ss(__m128 a, __m128 b)

CMPSS「より大きい」の条件で比較。
__m128 _mm_cmpgt_ss(__m128 a, __m128 b)

CMPSS「より大きいか等しい」の条件で比較。
__m128 _mm_cmpge_ss(__m128 a, __m128 b)

CMPSS「等しくない」の条件で比較。
__m128 _mm_cmpneq_ss(__m128 a, __m128 b)

CMPSS「より小さくない」の条件で比較。
__m128 _mm_cmpnlt_ss(__m128 a, __m128 b)

CMPSS「より大きくない」の条件で比較。
__m128 _mm_cmpngt_ss(__m128 a, __m128 b)

CMPSS「より大きくなく等しくない」の条件で比較。
__m128 _mm_cmpnge_ss(__m128 a, __m128 b)

CMPSS「オーダー」の条件で比較。
__m128 _mm_cmpord_ss(__m128 a, __m128 b)

CMPSS「アンオーダー」の条件で比較。
__m128 _mm_cmpunord_ss(__m128 a, __m128 b)

CMPSS「より小さくなく等しくない」の条件で比較。
__m128 _mm_cmpnle_ss(__m128 a, __m128 b)

SIMD 浮動小数点例外

無効（SNaN オペランドの場合）、無効（上記の表に記載された QNaN および述語の場合）、デノーマル。

CMPSS—Compare Scalar Single-Precision Floating-Point Values（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CMPXCHG—Compare and Exchange

オペコード	命令	説明
0F B0/*r*	CMPXCHG *r/m8*, *r8*	AL を *r/m8* と比較し、等しい場合は ZF をセットし、*r8* を *r/m8* にロードする。等しくない場合は ZF をクリアし、*r/m8* を AL にロードする。
0F B1/*r*	CMPXCHG *r/m16*, *r16*	AX を *r/m16* と比較し、等しい場合は ZF をセットし、*r16* を *r/m16* にロードする。等しくない場合は ZF をクリアし、*r/m16* を AL にロードする。
0F B1/*r*	CMPXCHG *r/m32*, *r32*	EAX を *r/m32* と比較し、等しい場合は ZF をセットし、*r32* を *r/m32* にロードする。等しくない場合は ZF をクリアし、*r/m32* を AL にロードする。

説明

（オペランドのサイズにしたがって）AL、AX、または EAX レジスタの値を第 1 オペランド（デスティネーション・オペランド）と比較する。2 つの値が等しい場合は、第 2 オペランド（ソース・オペランド）がデスティネーション・オペランドにロードされる。等しくない場合は、デスティネーション・オペランドが AL、AX、または EAX レジスタにロードされる。

この命令は、前に LOCK プリフィックスを付けることにより、自動的に実行させることができる。プロセッサのバスとのインターフェイスを単純にするため、デスティネーション・オペランドは比較の結果にかかわりなく書き込みサイクルを受ける。比較が失敗した場合はデスティネーション・オペランドがデスティネーションに書き戻され、比較が成功した場合はソース・オペランドがデスティネーションに書き込まれる。（プロセッサは、同時にロック書き込みも伴わずにロック読み取りを生じることは決してない。）

IA-32 アーキテクチャにおける互換性

この命令は、Intel486™ プロセッサより以前のインテル・プロセッサではサポートされていない。

操作

```
(* accumulator = AL, AX, or EAX, depending on whether *)
(* a byte, word, or doubleword comparison is being performed*)
IF accumulator = DEST
    THEN
        ZF ← 1
        DEST ← SRC
    ELSE
        ZF ← 0
        accumulator ← DEST
FI;
```

CMPXCHG—Compare and Exchange（続き）

影響を受けるフラグ

デスティネーション・オペランドと AL、AX、または EAX レジスタの値が等しい場合は ZF フラグがセットされ、等しくない場合はクリアされる。CF、PF、AF、SF、OF フラグは比較演算の結果にしたがってセットされる。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

CMPXCHG8B—Compare and Exchange 8 Bytes

オペコード	命令	説明
0F C7 /1 m64	CMPXCHG8B *m64*	EDX:EAX を *m64* と比較し、等しい場合は ZF をセットし、ECX:EBX を *m64* にロードする。等しくない場合は ZF をクリアし、*m64* を EDX:EAX にロードする。

説明

EDX:EAX の 64 ビット値をオペランド（デスティネーション・オペランド）と比較する。値が等しい場合は、ECX:EBX の 64 ビット値がデスティネーション・オペランドにストアされる。等しくない場合は、デスティネーション・オペランドの値がEDX:EAXにロードされる。デスティネーション・オペランドは 8 バイトのメモリ・ロケーションである。EDX:EAX および ECX:EBX のレジスタペアでは、EDX と ECX の内容が 64 ビット値の上位 32 ビットであり、EAX と EBX の内容が下位 32 ビットである。

この命令は、前に LOCK プリフィックスを付けることにより、自動的に実行させることができる。プロセッサのバスとのインターフェイスを単純にするため、デスティネーション・オペランドは比較の結果にかかわりなく書き込みサイクルを受ける。比較が失敗した場合はデスティネーション・オペランドがデスティネーションに書き戻され、比較が成功した場合はソース・オペランドがデスティネーションに書き込まれる。（プロセッサは、同時にロック書き込みも伴わずにロック読み取りを生じることは決してない。）

IA-32 アーキテクチャにおける互換性

この命令は、インテル® Pentium® プロセッサより以前のインテル・プロセッサではサポートされていない。

操作

```
IF (EDX:EAX = DEST)
    ZF ← 1
    DEST ← ECX:EBX
ELSE
    ZF ← 0
    EDX:EAX ← DEST
```

影響を受けるフラグ

デスティネーション・オペランドと EDX:EAX が等しい場合は ZF フラグがセットされ、等しくない場合はクリアされる。CF、PF、AF、SF、OF フラグは影響を受けない。

CMPXCHG8B—Compare and Exchange 8 Bytes（続き）

保護モード例外

#UD	デスティネーション・オペランドがメモリ・ロケーションでない場合。
#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#UD	デスティネーション・オペランドがメモリ・ロケーションでない場合。
#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#UD	デスティネーション・オペランドがメモリ・ロケーションでない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS

オペコード	命令	説明
66 0F 2F /r	COMISD *xmm1*, *xmm2/m64*	*xmm1* と *xmm2/mem64* の下位の倍精度浮動小数点値を比較し、その結果にしたがって EFLAGS フラグをセットする。

説明

ソース・オペランド 1（第 1 オペランド）とソース・オペランド 2（第 2 オペランド）の下位のクワッドワードの倍精度浮動小数点値を比較し、その結果（アンオーダー、より大きい、より小さい、または等しい）にしたがって、EFLAGS レジスタの ZF、PF、CF フラグをセットする。EFLAGS レジスタの OF、SF、AF フラグは 0 にクリアされる。いずれかのソース・オペランドが NaN（QNaN または SNaN）の場合は、アンオーダーのプレディケートが返される。

ソース・オペランド 1 は XMM レジスタである。ソース・オペランド 2 は、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。

COMISD 命令と UCOMISD 命令の相違点は、COMISD 命令は、ソース・オペランドが QNaN または SNaN の場合に SIMD 浮動小数点無効操作例外（#I）を報告することである。UCOMISD 命令は、ソース・オペランドが SNaN の場合にのみ、無効数値例外を報告する。

マスクされていない SIMD 浮動小数点例外が発生した場合は、EFLAGS レジスタは更新されない。

操作

```
RESULT ← OrderedCompare(DEST[63-0] <> SRC[63-0]) {
* Set EFLAGS *CASE (RESULT) OF
    UNORDERED:      ZF,PF,CF ← 111;
    GREATER_THAN:   ZF,PF,CF ← 000;
    LESS_THAN:      ZF,PF,CF ← 001;
    EQUAL:          ZF,PF,CF ← 100;
ESAC;
OF,AF,SF ← 0;
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_comieq_sd(__m128d a, __m128d b)
int_mm_comilt_sd(__m128d a, __m128d b)
int_mm_comile_sd(__m128d a, __m128d b)
int_mm_comigt_sd(__m128d a, __m128d b)
int_mm_comige_sd(__m128d a, __m128d b)
int_mm_comineq_sd(__m128d a, __m128d b)

COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS（続き）

SIMD 浮動小数点例外

無効（SNaN または QNaN オペランドの場合）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS

オペコード	命令	説明
0F 2F /r	COMISS *xmm1*, *xmm2/m32*	*xmm1* と *xmm2/mem32* の最下位の単精度浮動小数点値を比較し、その結果にしたがって EFLAGS フラグをセットする。

説明

ソース・オペランド 1（第 1 オペランド）とソース・オペランド 2（第 2 オペランド）の最下位のダブルワードの単精度浮動小数点値を比較し、その結果（アンオーダー、より大きい、より小さい、または等しい）にしたがって、EFLAGS レジスタの ZF、PF、CF フラグをセットする。EFLAGS レジスタの OF、SF、AF フラグは 0 にクリアされる。いずれかのソース・オペランドが NaN（QNaN または SNaN）の場合は、順序付けされない結果を返す。

ソース・オペランド 1 は XMM レジスタで、ソース・オペランド 2 は XMM レジスタまたは 32 ビットのメモリ・ロケーションを使用できる。

COMISS 命令と UCOMISS 命令の相違点は、COMISS は、ソース・オペランドが QNaN または SNaN オペランドである場合に SIMD 浮動小数点無効操作例外（#I）を報告することである。UCOMISS は、ソース・オペランドが SNaN である場合にのみ無効数値例外を報告する。

マスクされていない SIMD 浮動小数点例外が発生した場合は、EFLAGS レジスタは更新されない。

操作

```
RESULT ← OrderedCompare(SRC1[31-0] <> SRC2[31-0]) {
* Set EFLAGS *CASE (RESULT) OF
    UNORDERED:      ZF,PF,CF ← 111;
    GREATER_THAN:   ZF,PF,CF ← 000;
    LESS_THAN:      ZF,PF,CF ← 001;
    EQUAL:          ZF,PF,CF ← 100;
ESAC;
OF,AF,SF ← 0;
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_comieq_ss(__m128 a, __m128 b)
int_mm_comilt_ss(__m128 a, __m128 b)
int_mm_comile_ss(__m128 a, __m128 b)
int_mm_comigt_ss(__m128 a, __m128 b)
int_mm_comige_ss(__m128 a, __m128 b)
int_mm_comineq_ss(__m128 a, __m128 b)

COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS（続き）

SIMD 浮動小数点例外

無効（SNaN またはQNaN オペランドの場合）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CPUID—CPU Identification

オペコード	命令	説明
0F A2	CPUID	EAX レジスタに最初に入力された値に応じて、プロセッサの識別情報と機能情報を EAX、EBX、ECX、EDX の各レジスタに返す。

説明

EFLAGS レジスタ内の ID フラグ（ビット 21）は、CPUID 命令のサポートの有無を示している。ソフトウェア・プロシージャでこのフラグを設定およびクリアできる場合、そのプロシージャを実行するプロセッサは CPUID 命令をサポートしていることになる。

CPUID 命令は、プロセッサの識別情報と機能情報を EAX、EBX、ECX、EDX の各レジスタに返す。この命令の出力は、実行時の EAX レジスタの内容によって異なる。例えば、以下の疑似コードは、EAX に 00H をロードし、CPUID が最大戻り値とベンダ識別ストリングを適切なレジスタに返すようにする。

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

表 3-8. は、EAX レジスタにロードされる初期値に基づいて、返される情報を示している。表 3-9. は、CPUID 命令が実装されている IA-32 プロセッサの各ファミリについて、CPUID 命令の認識される最大入力値を示している。

基本機能情報と拡張機能情報という 2 つの型の情報が返される。入力された値より大きな値が特定のプロセッサに有効であると、基本情報についての最大値に対する情報が返される。例えば、インテル® Pentium® 4 プロセッサの場合、EAX に 5 が入力されると、入力値が 2 のときの情報が返される。ただし、拡張機能情報を返す場合の入力値については、この規則は適用されない。インテル Pentium 4 プロセッサの場合には、80000005H 以上の値を入力すると、入力値が 2 のときの情報が返される。

CPUID 命令はどの特権レベルでも実行でき、命令の実行をシリアル化することができる。命令の実行をシリアル化することにより、前の命令においてフラグ、レジスタ、メモリに対して修正が行われた場合、それらの修正がすべて完了してから、次の命令がフェッチされて実行されることが保証される。

以下の資料も参照のこと。

『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、下巻』の第 7 章の「シリアル化命令」

『AP-485、インテル® プロセッサの識別と CPUID 命令』（資料番号 241618J）

CPUID—CPU Identification（続き）

表 3-8. CPUID 命令から返される情報

EAX の初期値	提供されるプロセッサに関する情報	
	CPUID 基本情報	
0H	EAX EBX ECX EDX	CPUID 基本情報の最大入力値（表 3-9. を参照） “Genu” “ntel” “inel”
01H	EAX EBX ECX EDX	バージョン情報: タイプ、ファミリ、モデル、ステッピングID（図3-5. を参照） ビット 7 ～ 0: ブランド・インデックス ビット 15 ～ 8: CLFLUSH のライン・サイズ（値 *8 = キャッシュ・ライン・サイズ（バイト単位）） ビット 23 ～ 16: 物理プロセッサ当たりの論理プロセッサの数。ハイパー・スレッド・テクノロジをサポートしている インテル® Pentium® 4 プロセッサでは 2 ビット 31 ～ 24: ローカル APIC の ID 拡張機能情報（図 3-6. および表 3-11. を参照） 機能情報（図 3-7. および表 3-12. を参照）
02H	EAX EBX ECX EDX	キャッシュおよび TLB 情報（表 3-13. を参照） キャッシュおよび TLB 情報 キャッシュおよび TLB 情報 キャッシュおよび TLB 情報
03H	EAX EBX ECX EDX	予約済み 予約済み 96 ビットのプロセッサ・シリアル・ナンバのビット 00 ～ 31（インテル® Pentium® III プロセッサのみ。それ以外、このレジスタの値は予約されている）。 96 ビットのプロセッサ・シリアル・ナンバのビット 32 ～ 63（インテル Pentium III プロセッサのみ。それ以外、このレジスタの値は予約されている）。 注：プロセッサ・シリアル・ナンバ（PSN）は、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サポートしていない。すべてのモデル上で、PSN 機能を利用する前に、PSN フラグ（CPUID 命令によって返される）を使用して PSN のサポートの有無をチェックすること。PSN に関する詳細は、『AP-485、インテル® プロセッサの識別と CPUID 命令』（資料番号 241618J）を参照のこと。
04H	 EAX EBX ECX	決定性のキャッシュ・パラメータ・リーフ ビット 4-0: キャッシュ・タイプ ** ビット 7-5: キャッシュ・レベル（1 から始まる） ビット 8: キャッシュ・レベルの自己初期化（ソフトウェアによる初期化が不要） ビット 9: フル・アソシアティブ・キャッシュ ビット 13-10: 予約済み ビット 25-14: このキャッシュを共有しているスレッドの数 * ビット 31-26: このダイ上のプロセッサ・コアの数（マルチコア）* ビット 11-00: L = システム・コヒーレンシ・キャッシュ・サイズ * ビット 21-12: P = 物理ライン・パーティション * ビット 31-22: W = アソシアティブ・ウェイ数 * ビット 31-00: S = セット数 *

CPUID—CPU Identification（続き）

表 3-8. CPUID 命令から返される情報（続き）

EAX の初期値	提供されるプロセッサに関する情報
	EDX 予約済み = 0 * この数を求めるには、レジスタ・ファイル内の値に 1 を加算する。例えば、プロセッサ・コアの数は、EAX[31:26]+1 になる。 ** キャッシュ・タイプ・フィールド 0 = Null - キャッシュなし 1 = データ・キャッシュ 2 = 命令キャッシュ 3 = ユニファイド・キャッシュ 4-31 = 予約済み 注 : 3 より大きく 80000000 より小さい CPUID リーフは、IA32_CR_MISC_ENABLES.BOOT_NT4（ビット 22）がクリアされている場合（デフォルト）にのみ参照可能である。
05H	MONITOR/MWAIT リーフ EAX ビット 15-00: 最小モニタライン・サイズ（バイト単位）（デフォルトはプロセッサのモニタ・グラニュラリティ） ビット 31-16: 予約済み = 0 EBX ビット 15-00: 最大モニタライン・サイズ（バイト単位）（デフォルトはプロセッサのモニタ・グラニュラリティ） ビット 31-16: 予約済み = 0 ECX 予約済み = 0 EDX 予約済み = 0
	拡張機能 CPUID 情報
80000000H	EAX 拡張機能 CPUID 情報の最大入力値（表 3-9. を参照）。 EBX 予約済み ECX 予約済み EDX 予約済み
80000001H	EAX 拡張されたプロセッサ・シグネチャと拡張された機能ビット（現在は予約されている） EBX 予約済み ECX 予約済み EDX 予約済み
80000002H	EAX プロセッサ・ブランド・ストリング EBX プロセッサ・ブランド・ストリング（続き） ECX プロセッサ・ブランド・ストリング（続き） EDX プロセッサ・ブランド・ストリング（続き）
80000003H	EAX プロセッサ・ブランド・ストリング（続き） EBX プロセッサ・ブランド・ストリング（続き） ECX プロセッサ・ブランド・ストリング（続き） EDX プロセッサ・ブランド・ストリング（続き）

CPUID—CPU Identification（続き）

表 3-8. CPUID 命令から返される情報（続き）

EAX の初期値	提供されるプロセッサに関する情報
80000004H	EAX　プロセッサ・ブランド・ストリング（続き） EBX　プロセッサ・ブランド・ストリング（続き） ECX　プロセッサ・ブランド・ストリング（続き） EDX　プロセッサ・ブランド・ストリング（続き）
80000005H	EAX　予約済み = 0 EBX　予約済み = 0 ECX　予約済み = 0 EDX　予約済み = 0
80000006H	EAX　予約済み = 0 EBX　予約済み = 0 ECX　ビット 0-7: キャッシュ・ライン・サイズ ビット 15-12: L2 アソシアティビティ ビット 31-16: キャッシュ・サイズ（1K 単位） EDX　予約済み = 0
80000007H	EAX　予約済み = 0 EBX　予約済み = 0 ECX　予約済み = 0 EDX　予約済み = 0
80000008H	EAX　予約済み = 0 EBX　予約済み = 0 ECX　予約済み = 0 EDX　予約済み = 0

入力 EAX = 0: プロセッサの基本情報とベンダ識別ストリングについて、CPUID 命令が認識可能な最大値を返す。

EAX レジスタに 0 を設定して CPUID 命令を実行した場合、プロセッサは、プロセッサの基本情報を返すために、CPUID 命令が認識可能な最大値を返す。値が EAX レジスタに返される（表 3-9. を参照）。この値はプロセッサ固有である。

EBX、EDX、ECX の各レジスタには、ストリングが返される。インテル・プロセッサの場合、ベンダ識別ストリングは次のように "GenuineIntel" になり、次のように表される。

```
EBX ← 756e6547h (* "Genu", with G in the low nibble of BL *)
EDX ← 49656e69h (* "ineI", with i in the low nibble of DL *)
ECX ← 6c65746eh (* "ntel", with n in the low nibble of CL *)
```

CPUID—CPU Identification（続き）

入力 EAX = 80000000H: プロセッサの拡張情報について、CPUID 命令が認識可能な最大値を返す。

EAX レジスタを 0 に設定して CPUID 命令を実行した場合、プロセッサは、プロセッサの拡張情報を返すために CPUID 命令が認識可能な最大値を返す。この値は EAX レジスタに返される（表 3-9 を参照）。この値はプロセッサ固有である。

表 3-9. IA-32 プロセッサに対する CPUID 命令のソース・オペランドの最大値

IA-32 プロセッサ	EAX レジスタの最大値	
	基本情報	拡張機能情報
初期 Intel486™ プロセッサ	CPUID は実装されていない	CPUID は実装されていない
後期 Intel486 プロセッサおよび インテル® Pentium® プロセッサ	01H	実装されていない
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、 インテル® Celeron™ プロセッサ	02H	実装されていない
インテル® Pentium® III プロセッサ	03H	実装されていない
インテル® Pentium® 4 プロセッサ	02H	80000004H
インテル® Xeon™ プロセッサ	02H	80000004H
インテル® Pentium® M プロセッサ	02H	80000004H
ハイパー・スレッディング・テクノロジ対応 インテル Pentium 4 プロセッサ	05H	80000008H

CPUID—CPU Identification（続き）

入力 EAX = 1: モデル、ファミリ、ステッピング情報を返す。

EAX レジスタの入力値に 1 を設定して CPUID 命令を実行した場合、プロセッサは EAX レジスタにバージョン情報を返す（図 3-5. を参照）。例えば、インテル ® Pentium® 4 ファミリの最初のプロセッサのモデル、ファミリ、プロセッサ・タイプは、次のようになる。

- モデル－ 0000B
- ファミリ－ 1111B
- プロセッサ・タイプ－ 00B

使用可能なプロセッサ・タイプの値は表 3-10 を参照のこと。ステッピング ID については、必要に応じて情報が提供されている。

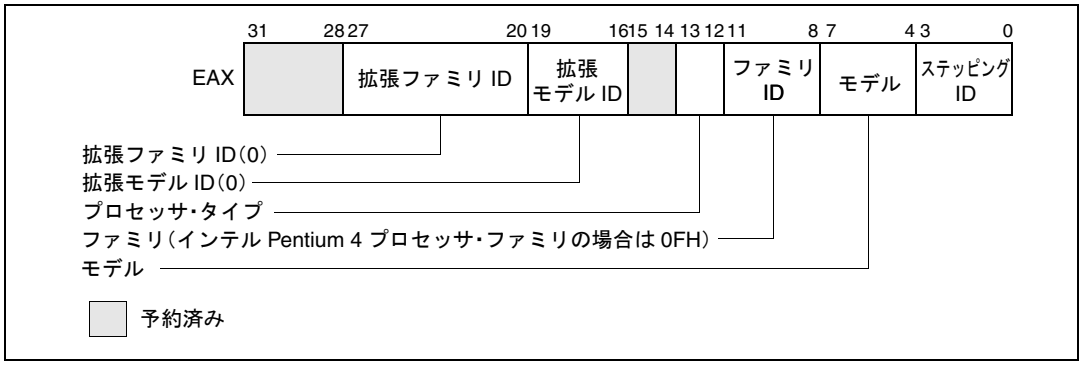

図 3-5. CPUID 命令によって返される EAX レジスタのバージョン情報

CPUID—CPU Identification（続き）

表 3-10. プロセッサ・タイプ・フィールド

タイプ	コード化
オリジナル OEM プロセッサ	00B
インテル® OverDrive® プロセッサ	01B
デュアル・プロセッサ（Intel486™ プロセッサには適用不可）	10B
インテル用に予約済み	11B

注記

初期の IA-32 プロセッサの識別については、『AP-485、インテル® プロセッサの識別と CPUID 命令』（資料番号 241618J）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、上巻』の第 14 章を参照のこと。

ファミリ ID が 0FH またはそれ以上の場合にのみ、拡張ファミリ ID と拡張モデル ID を確認する必要がある。プロセッサ情報は、常にファミリ、モデル、ステッピングを組み合わせて表示すること。

各 ID フィールドを次のように計算して、ファミリを表示する。

```
Displayed family = ((Extended Family ID(4-bits) << 4)) (8-bits)
+ Family ID (4-bits zero extended to 8-bits)
```

表示されるモデルは、モデル ID と拡張モデル ID から次のように計算する。

```
Displayed Model = ((Extended Model ID (4-bits) << 4))(8-bits)
+ Model (4-bits zero extended to 8-bits)
```

CPUID—CPU Identification（続き）

入力 EAX = 1: EBX に追加情報を返す

EAX レジスタに 1 を設定して CPUID 命令を実行した場合、EBX レジスタには、追加情報が返される。

- ブランド・インデックス（EBX の下位バイト）— IA-32 プロセッサのブランド・ストリングで構成されるブランド・ストリング・テーブルのエントリには、この数字が入っている。このフィールドの詳細については、後の項で説明する。
- CLFLUSH 命令キャッシュ・ライン・サイズ（EBX の第 2 バイト）— この数字は、CLFLUSH 命令によって 8 バイト・インクリメントでフラッシュされるキャッシュ・ラインのサイズを示す。このフィールドは、インテル Pentium 4 プロセッサで導入されたものである。
- ローカル APIC ID（EBX の上位バイト）— この数字は、電源入力時にプロセッサのローカル APIC に割り当てられる 8 ビットの ID を示す。このフィールドは、インテル Pentium 4 プロセッサで導入されたものである。

入力 EAX = 1: ECX と EDX に機能情報を返す

EAX レジスタを 1 に設定して CPUID 命令を実行した場合、機能情報は ECX と EDX に返される。

- 図 3-6 と表 3-11 に、ECX のコード化を示す。
- 図 3-7 と表 3-12 に、EDX のコード化を示す。

すべての機能フラグについて、1 はその機能をサポートしていることを示す。機能フラグの正しい解釈については、インテルまでお問い合わせのこと。

注記

ソフトウェアは、プロセッサの特定の機能を使用する前に、CPUID 命令によって返される機能フラグを利用して、その機能が存在することを確認しなければならない。ソフトウェアは、すべての機能を搭載した将来のプロセッサに依存してはならない。

CPUID—CPU Identification（続き）

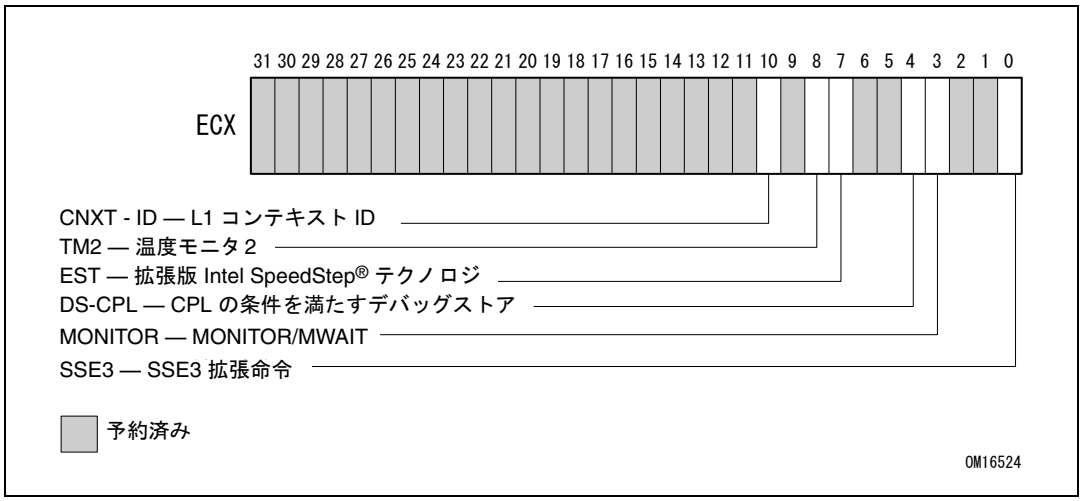

図 3-6. ECX レジスタに返される拡張機能情報

表 3-11. ECX レジスタに返される拡張機能情報の詳細

ビット #	ニーモニック	説明
0	SSE3	ストリーミング SIMD 拡張命令 3（SSE3）。1 の値は、プロセッサがこのテクノロジをサポートしていることを示す。
3	MONITOR	MONITOR/MWAIT。1 の値は、プロセッサがこの機能をサポートしていることを示す。
4	DS-CPL	CPL Qualified Debug Store。CPL の条件を満たすデバッグストア。1 の値は、プロセッサが、CPL の条件を満たす分岐メッセージをストアできるデバッグストア拡張機能をサポートしていることを示す。
7	EST	拡張版 Intel SpeedStep® テクノロジ。1 の値は、プロセッサがこのテクノロジをサポートしていることを示す。
8	TM2	温度モニタ 2。1 の値は、プロセッサがこのテクノロジをサポートしていることを示す。
10	CNXT-ID	L1 コンテキスト ID。1 の値は、L1 データ・キャッシュ・モードをアダプティブ・モードまたは共有モードに設定できることを示す。0 の値は、この機能をサポートしていないことを示す。詳細は、IA32_MISC_ENABLE MSR ビット 24（L1 データ・キャッシュ・コンテキスト・モード）の定義を参照のこと。

CPUID—CPU Identification（続き）

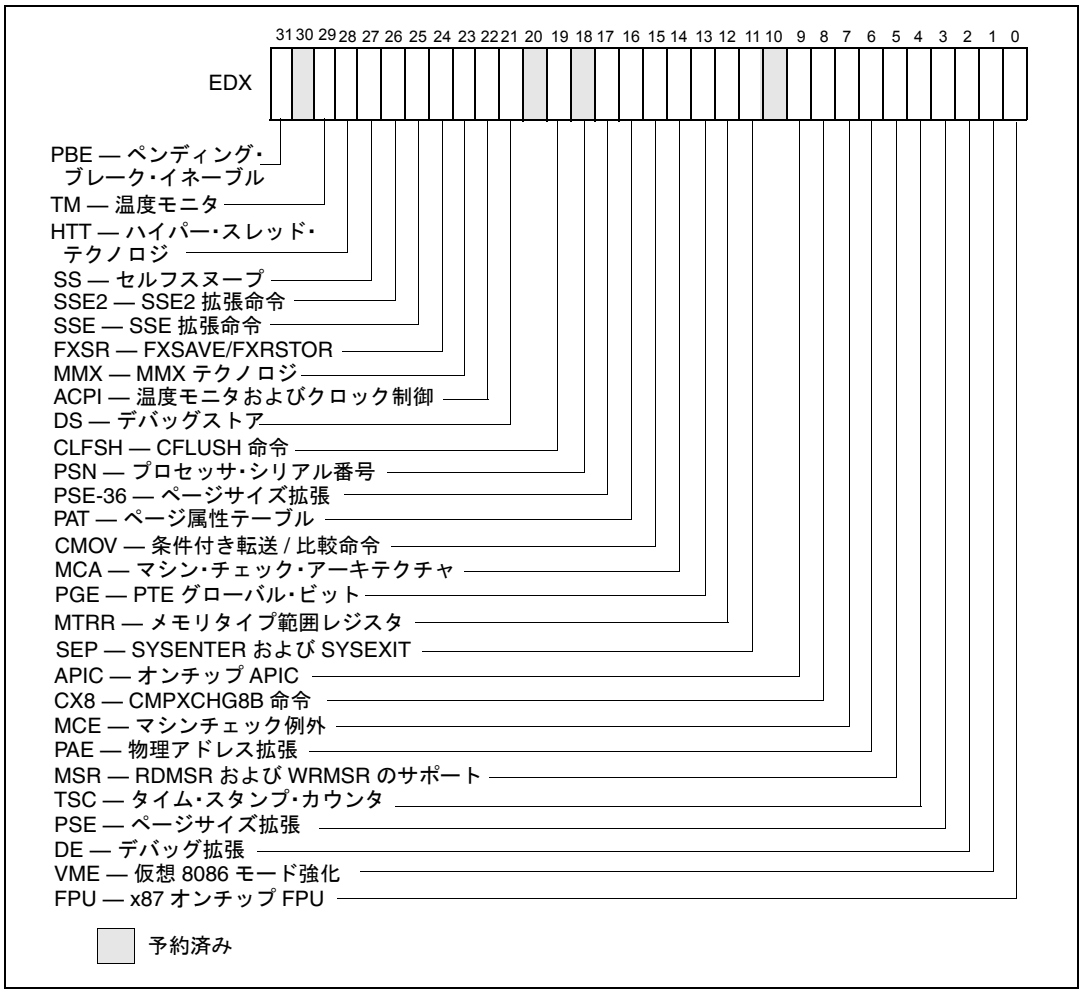

図 3-7. EDX レジスタに返される機能情報

CPUID—CPU Identification（続き）

表 3-12. EDX レジスタに返される機能情報の詳細

ビット #	ニーモニック	説明
0	FPU	オンチップ浮動小数点ユニット。プロセッサは、x87 FPU を搭載している。
1	VME	仮想 8086 モード強化。仮想 8086 モード強化機能には次のものが含まれる。機能制御用の CR4.VME、保護モード仮想割り込み用の CR4.PVI、ソフトウェア割り込みインダイレクション、ソフトウェア・インダイレクション・ビットマップによる TSS の拡張、EFLAGS.VIF フラグ、および EFLAGS.VIP フラグ。
2	DE	デバッグ拡張。機能制御用の CR4.DE、およびオプションの DR4 および DR5 へのアクセストラップを含めて、I/O ブレークポイントをサポート。
3	PSE	ページサイズ拡張。機能制御用の CR4.PSE、PDE（ページ・ディレクトリ・エントリ）内の定義済みのダーティビット、CR3、PDE、および PTE 内のオプションの予約ビット・トラッピングを含めて、4M バイトのラージ・ページ・サイズをサポート。
4	TSC	タイム・スタンプ・カウンタ。特権制御用の CR4.TSD を含めて、RDTSC 命令をサポート。
5	MSR	モデル固有レジスタの RDMSR 命令および WRMSR 命令。RDMSR 命令および WRMSR 命令をサポート。MSR によっては、プロセッサに依存しないものもある。
6	PAE	物理アドレス拡張。32 ビットを超える物理アドレスをサポート。拡張ページ・テーブル・エントリ・フォーマット、ページ変換テーブル内の特別レベルが定義され、PAE ビットが 1 の場合は、4M バイト・ページの代わりに 2M バイト・ページをサポート。32 ビットを超える場合の実際のアドレスビット数は定義されておらず、プロセッサ固有である。
7	MCE	マシンチェック例外。機能制御用の CR4.MCE を含めて、マシンチェック用に例外 18 が定義される。この機能は、モデル固有インプリメンテーションにおけるマシン・チェック・エラーのロギング、レポート、およびプロセッサ・シャットダウンについては定義しない。マシンチェック例外ハンドラは、プロセッサのバージョンにしたがってモデル固有の例外処理を実行するか、またはマシンチェック機能の有無を確認する必要が生じる場合がある。
8	CX8	CMPXCHG8B 命令。8 バイト（64 ビット）比較交換命令をサポート（暗黙的にロックされ、アトミックに行われる）。
9	APIC	オンチップ APIC。プロセッサは、アドバンスト・プログラム可能割り込みコントローラ（APIC）を内蔵し、物理アドレス範囲 FFFE0000H ～ FFFE0FFFH でメモリ・マップ・コマンドに応答する（デフォルトでは、プロセッサによっては APIC の再配置を許可するものもある）。
10	Reserved	予約済み
11	SEP	SYSENTER 命令および SYSEXIT 命令。SYSENTER、SYSEXIT、および関連する MSR をサポート。
12	MTRR	メモリタイプ範囲レジスタ。MTRR をサポート。MTRRcap MSR には機能ビットが含まれていて、サポートされているメモリタイプ、サポートされている可変 MTRR の個数、固定 MTRR のサポートの有無について示される。

CPUID—CPU Identification（続き）

表 3-12. EDX レジスタに返される機能情報の詳細（続き）

ビット #	ニーモニック	説明
13	PGE	PTE グローバル・ビット。ページ・ディレクトリ・エントリ（PDE）およびページ・テーブル・エントリ（PTE）内のグローバル・ビットをサポート。これらは、各種の処理に共通で、フラッシュする必要のない TLB エントリであることを示す。この機能は、CR4.PGE ビットによって制御される。
14	MCA	マシン・チェック・アーキテクチャ。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、将来のプロセッサのエラー・レポート用の互換メカニズムを提供するマシン・チェック・アーキテクチャをサポート。MCG_CAP MSR には、サポートされているエラー・レポート用 MSR バンクの個数を示す機能ビットが含まれる。
15	CMOV	条件付き転送命令。条件付き転送命令（CMOV）をサポート。また、x87 FPU が搭載されている場合は（CPUID.FPU 機能ビットによって示される）、FCOMI 命令および FCMOV 命令をサポート。
16	PAT	ページ属性テーブル。ページ属性テーブルをサポート。この機能は、メモリタイプ範囲レジスタ（MTRR）の数を増やす。これによって、オペレーティング・システムは、リニアアドレスを使用して 4K バイト単位でメモリの属性を指定できる。
17	PSE-36	32 ビット・ページ・サイズ拡張。拡張 4M バイト・ページをサポート。これにより、4G バイトを超える物理メモリのアドレス指定が可能になる。この機能は、4M バイト・ページの物理アドレスの上位 4 ビットが、ページ・ディレクトリ・エントリのビット 13 ～ 16 によってエンコーディングされることを示す。
18	PSN	プロセッサ・シリアル番号。96 ビットのプロセッサ識別番号機能をサポートしており、この機能がイネーブルになっている。
19	CLFSH	CLFLUSH 命令。CLFLUSH 命令をサポート。
20	Reserved	予約済み
21	DS	デバッグストア。メモリ常駐バッファにデバッグ情報を書き込む機能をサポート。この機能は、分岐トレースストア（BTS）およびプリサイス・イベント・ベース・サンプリング（PEBS）機能によって使用される（詳しくは、『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、下巻』の第 15 章「デバッグと性能モニタリング」を参照のこと）。
22	ACPI	温度モニタおよびソフトウェア制御クロック機能。プロセッサは MSR を内部に実装している。MSR を使用すると、ソフトウェアによる制御の下でプロセッサ温度を監視できると共に、事前に定義したデューティ・サイクルでプロセッサのパフォーマンスを調整することができる。
23	MMX	インテル® MMX® テクノロジ。インテル MMX テクノロジをサポート。
24	FXSR	FXSAVE 命令および FXRSTOR 命令。浮動小数点コンテキストの高速な保存 / 復元用の FXSAVE 命令および FXRSTOR 命令をサポート。また、このビットがセットされている場合、オペレーティング・システムは、CR4.OSFXSR を利用して、FXSAVE 命令および FXRSTOR 命令をサポートすることを示すことができる。
25	SSE	SSE。SSE 拡張命令をサポート。
26	SSE2	SSE2。SSE2 拡張命令をサポート。

CPUID—CPU Identification（続き）

表 3-12. EDX レジスタに返される機能情報の詳細（続き）

ビット #	ニーモニック	説明
27	SS	セルフスヌープ。競合するメモリタイプの管理機能をサポートしており、プロセッサのキャッシュ構造のスヌープを実行して、バスに対して発行されたトランザクションを検出できる。
28	HTT	ハイパー・スレッディング・テクノロジ。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する。
29	TM	温度モニタ。プロセッサは、温度モニタ自動温度制御回路（TCC）を実装している。
30	Reserved	予約済み
31	PBE	ペンディング・ブレーク・イネーブル。プロセッサは、ストップクロック状態（STPCLK# がアサートされている状態）での FERR#/PBE# ピンの使用をサポートしている。このピンによって、割り込みがペンディングになっており、通常の動作に戻ってその割り込みを処理する必要があることを、プロセッサに指示できる。この機能は、IA32_MISC_ENABLE MSR レジスタのビット 10（PBE イネーブル）によってイネーブルにされる。

入力 EAX = 2: EAX、EBX、ECX、EDX にキャッシュおよび TLB 情報を返す

EAX レジスタに 2 を設定して CPUID の命令を実行した場合、プロセッサはプロセッサの内部キャッシュおよび TLB に関する情報を EAX、EBX、ECX、および EDX レジスタに返す。

コード化は、以下のとおりである。

- EAX レジスタ（AL レジスタ）の最下位バイトは、入力値を 2 として、プロセッサのキャッシュと TLB の完全な記述を得るために CPUID 命令が実行されなければならない回数を示す。インテル Pentium 4 プロセッサ・ファミリの最初のメンバは 1 を返す。
- 各レジスタの最上位ビット（ビット 31）は、レジスタに有効な情報がある（0 にセット）か、予約されている（1 にセット）かを示す。
- レジスタに有効な情報がある場合は、その情報は 1 バイトの記述子に収められる。表 3-13. に、それらの記述子のコード化を示す。EAX、EBX、ECX、EDX の各レジスタ内の記述子の順序は定義されていない。すなわち、特定のバイトが特定のキャッシュ・タイプまたは TLB タイプの記述子が入るようには指定されていない。記述子は、あらゆる順序で示されることがある。

CPUID—CPU Identification（続き）

表 3-13. キャッシュおよび TLB 記述子のコード化

記述子の値	キャッシュまたは TLB の記述
00H	NULL 記述子
01H	命令 TLB: 4 K バイト・ページ、4 ウェイ・セット・アソシアティブ、32 エントリ
02H	命令 TLB: 4 M バイト・ページ、フル・アソシアティブ、2 エントリ
03H	データ TLB: 4 K バイト・ページ、4 ウェイ・セット・アソシアティブ、64 エントリ
04H	データ TLB: 4 M バイト・ページ、4 ウェイ・セット・アソシアティブ、8 エントリ
06H	第 1 レベルの命令キャッシュ : 8 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
08H	第 1 レベルの命令キャッシュ : 16 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
0AH	第 1 レベルのデータ・キャッシュ : 8 K バイト、2 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
0CH	第 1 レベルのデータ・キャッシュ : 16 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
22H	第 3 レベルのキャッシュ : 512K バイト、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
23H	第 3 レベルのキャッシュ : 1M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
25H	第 3 レベルのキャッシュ : 2M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
29H	第 3 レベルのキャッシュ : 4M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
2CH	第 1 レベルのデータ・キャッシュ 32K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
30H	第 1 レベルのデータ・キャッシュ : 32K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
40H	第 2 レベルのキャッシュなし。プロセッサに有効な第 2 レベルのキャッシュが実装されている場合は、第 3 レベルのキャッシュなし
41H	第 2 レベルのキャッシュ : 128 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
42H	第 2 レベルのキャッシュ : 256 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
43H	第 2 レベルのキャッシュ : 512 K バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
44H	第 2 レベルのキャッシュ : 1 M バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
45H	第 2 レベルのキャッシュ : 2M バイト、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ

CPUID—CPU Identification（続き）

表 3-13. キャッシュおよび TLB 記述子のコード化（続き）

記述子の値	キャッシュまたは TLB の記述
50H	命令 TLB: 4K バイト、2M バイト、4M バイト・ページ、64 エントリ
51H	命令 TLB: 4K バイト、2M バイト、4M バイト・ページ、128 エントリ
52H	命令 TLB: 4K バイト、2M バイト、4M バイト・ページ、256 エントリ
5BH	データ TLB: 4K バイト、4M バイト・ページ、64 エントリ
5CH	データ TLB: 4K バイト、4M バイト・ページ、128 エントリ
5DH	データ TLB: 4K バイト、4M バイト・ページ、256 エントリ
60H	第 1 レベルのデータ・キャッシュ : 16K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
66H	第 1 レベルのデータ・キャッシュ : 8K バイト、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
67H	第 1 レベルのデータ・キャッシュ : 16K バイト、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
68H	第 1 レベルのデータ・キャッシュ : 32K バイト、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
70H	トレース・キャッシュ : 12K-μop、8 ウェイ・セット・アソシアティブ
71H	トレース・キャッシュ : 16K-μop、8 ウェイ・セット・アソシアティブ
72H	トレース・キャッシュ : 32K-μop、8 ウェイ・セット・アソシアティブ
78H	第 2 レベルのキャッシュ : 1M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
79H	第 2 レベルのキャッシュ : 128K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
7AH	第 2 レベルのキャッシュ : 256K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
7BH	第 2 レベルのキャッシュ : 512K バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
7CH	第 2 レベルのキャッシュ : 1M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ、128 バイト・セクタ・サイズ
7DH	第 2 レベルのキャッシュ : 2M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
82H	第 2 レベルのキャッシュ : 256K バイト、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
83H	第 2 レベルのキャッシュ : 512K バイト、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
84H	第 2 レベルのキャッシュ : 1M バイト、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ

CPUID—CPU Identification（続き）

表 3-13. キャッシュおよび TLB 記述子のコード化（続き）

記述子の値	キャッシュまたは TLB の記述
85H	第 2 レベルのキャッシュ : 2M バイト、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
86H	第 2 レベルのキャッシュ : 512K バイト、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
87H	第 2 レベルのキャッシュ : 1M バイト、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
B0H	命令 TLB: 4K-Byte Pages, 4 ウェイ・セット・アソシアティブ、128 エントリ
B3H	データ TLB: 4K-Byte Pages, 4 ウェイ・セット・アソシアティブ、128 エントリ

例 3-1. キャッシュおよび TLB 解釈の例

インテル Pentium 4 プロセッサ・ファミリの最初のメンバは、CPUID 命令が入力値 2 で実行されると、キャッシュと TLB に関する以下の情報を返す。

EAX　66 5B 50 01H
EBX　0H
ECX　0H
EDX　00 7A 70 00H

これらの値は、以下のことを意味する。

- EAX レジスタの最下位バイト（バイト 0）が 01H に設定されており、キャッシュと TLB に関する完全な情報を取り出すのに、CPUID 命令は入力値を 2 として 1 回実行すればよいことを示している。
- 4 つのレジスタすべて（EAX、EBX、ECX、EDX）の最上位ビットが 0 にセットされており、各レジスタが有効な 1 バイトの記述子をもつことを示している。
- EAX レジスタのバイト 1、2、3 は、プロセッサが以下のものを備えていることを示す。
  - 50H - 4 K バイト、2 M バイト、4 M バイト・ページ・マップ用の 64 エントリ命令 TLB
  - 5BH - 4 K バイトおよび 4 M バイト・ページ・マップ用の 64 エントリ・データ TLB
  - 66H - 8 K バイト第 1 レベルのデータ・キャッシュ、4 ウェイ・セット・アソシアティブ、64 バイト・キャッシュ・ライン・サイズ
- EBX および ECX レジスタ内の記述子は有効であるが、内容は Null 記述子である。

CPUID—CPU Identification（続き）

- EDX レジスタのバイト 0、1、2、3 は、プロセッサが以下のものを備えていることを示す。
  - 00H - Null 記述子
  - 70H - 12 K バイト第 1 レベルのコード・キャッシュ、4 ウェイ・セット・アソシアティブ、64 バイト・キャッシュ・ライン・サイズ
  - 7AH - 256 K バイト第 2 レベルのキャッシュ、8 ウェイ・セット・アソシアティブ、セクタ化、64 バイト・キャッシュ・ライン・サイズ
  - 00H - Null 記述子

ブランド情報を返す方法

以下の方法を使用して、ブランド情報にアクセスできる。

1. プロセッサ・ブランド・ストリングの方法。この方法では、プロセッサの最大動作周波数も返される。
2. プロセッサ・ブランド・インデックス。この方法は、ソフトウェアが提供するブランド・ストリング・テーブルを使用する。

これらの 2 つの方法について、以下の各節で説明する。初期のプロセッサで利用できる方法については、『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 章の「従来のインテル® アーキテクチャ・プロセッサの識別」の節を参照のこと。

プロセッサ・ブランド・ストリングの方法

図 3-8 は、ブランド・ストリングの検出に使用されるアルゴリズムを示している。プロセッサ･ブランド識別ソフトウェアは､すべての IA-32 アーキテクチャ互換プロセッサ上で、このアルゴリズムを実行できなければならない。

この方法（インテル Pentium 4 プロセッサで導入）は、ASCII ブランド識別ストリングとプロセッサの最大動作周波数を、EAX、EBX、ECX、EDX の各レジスタに返す。

CPUID—CPU Identification（続き）

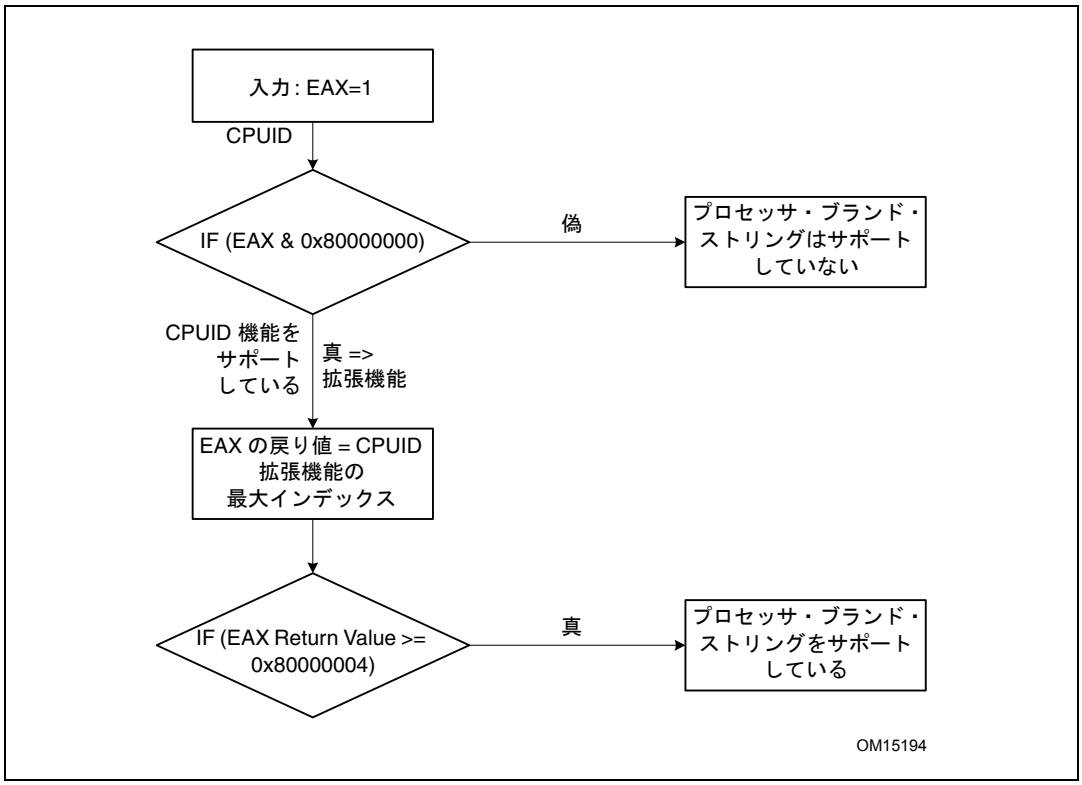

図 3-8. プロセッサ・ブランド・ストリングのサポートの確認

CPUID—CPU Identification（続き）

ブランド・ストリングの機能

ブランド・ストリングの方法を使用するには、EAX の入力値を 8000002H ～ 80000004H に設定して CPUID 命令を実行する。各入力値に対して、CPUID 命令は、EAX、EBX、ECX、EDX を使用して 16 個の ASCII 文字を返す。返されたストリングは NULL で終わる。

表 3-14 は、インテル Pentium 4 プロセッサ・ファミリの最初のプロセッサが返すブランド・ストリングを示している。

表 3-14. インテル® Pentium® 4 プロセッサに関して返されるプロセッサ・ブランド・ストリング

EAX 入力値	戻り値	対応する ASCII ストリング
80000002H	EAX = 20202020H EBX = 20202020H ECX = 20202020H EDX = 6E492020H;	“ ” “ ” “ ” “nI ”
80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	“(let” “P)R” “itne” “R(mu”
80000004H	EAX = 20342029H; EBX = 20555043H; ECX = 30303531H EDX = 007A484DH	“ 4)” “ UPC” “0051” “\0zHM”

ブランド・ストリングからのプロセッサの最大周波数の抽出

図 3-9 は、ソフトウェアがプロセッサ・ブランド・ストリングからプロセッサの最大動作周波数を抽出するアルゴリズムを示している。

注記

ブランド・ストリング内に周波数が示されている場合、この周波数は、そのプロセッサに対して定義されている最大周波数を表すもので、現行の動作周波数を表しているのではない。

CPUID—CPU Identification（続き）

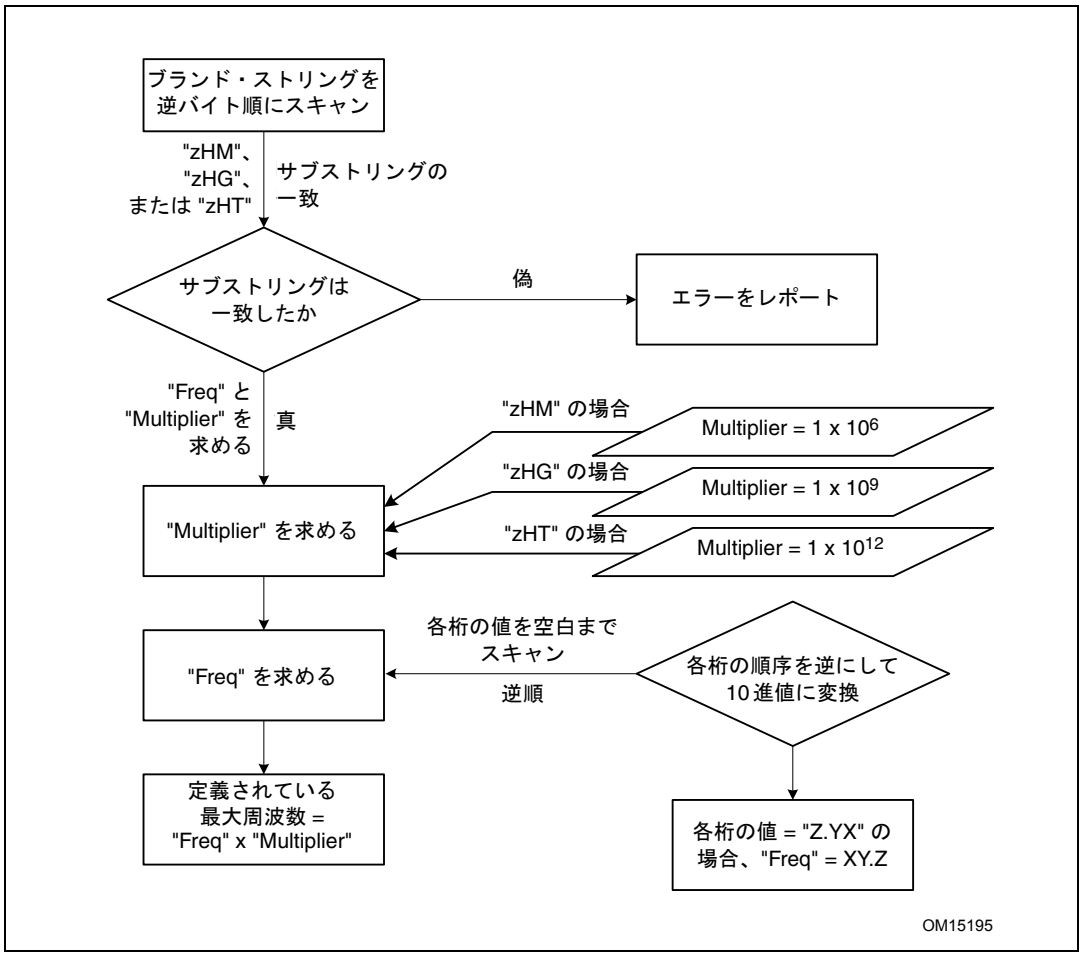

図 3-9. プロセッサの最大周波数を抽出するアルゴリズム

CPUID—CPU Identification（続き）

プロセッサのブランド・インデックスの方法

ブランド・インデックスの方法（インテル® Pentium® III Xeon™ プロセッサで導入）は、ブランド識別テーブルへのエントリポイントとなる。ブランド識別テーブルはシステム・ソフトウェアによってメモリ内に保持され、システムレベルおよびユーザレベルのコードからアクセスが可能である。このブランド識別テーブルでは、各ブランド・インデックスが、公認のインテル・プロセッサ・ファミリであることを示すASCIIブランド識別ストリングとプロセッサのモデル番号とに関連付けられている。

EAX レジスタに値 1 を設定して CPUID 命令を実行すると、EBX レジスタの下位バイトにブランド・インデックスが返される。ソフトウェアでこのインデックスを使用することで、ブランド識別テーブル内にあるプロセッサのブランド識別ストリングを特定できる。ブランド識別テーブルの最初のエントリ（ブランド・インデックス 0）は予約されており、ブランド識別機能をサポートしていないプロセッサとの下方互換性が保たれている。

表 3-15. は、識別ストリングが関連付けられているブランド・インデックスを示している。

CPUID—CPU Identification（続き）

表 3-15. ブランド・インデックスと IA-32 プロセッサ・ブランド・ストリングの対応関係

ブランド・インデックス	ブランド・ストリング
0H	このプロセッサはブランド ID 機能をサポートしていない。
01H	インテル® Celeron® プロセッサ[1]
02H	インテル® Pentium® III プロセッサ [1]
03H	インテル® Pentium® III Xeon™ プロセッサ : プロセッサ・シグニチャ = 000006B1h の場合、インテル Celeron プロセッサ
04H	インテル Pentium III プロセッサ
06H	モバイル インテル® Pentium® III プロセッサ - M
07H	モバイル インテル® Celeron® プロセッサ [1]
08H	インテル® Pentium® 4 プロセッサ
09H	インテル Pentium 4 プロセッサ
0AH	インテル Celeron プロセッサ [1]
0BH	インテル® Xeon™ プロセッサ : プロセッサ・シグニチャ = 00000F13h の場合、インテル® Xeon™ プロセッサ MP
0CH	インテル Xeon プロセッサ MP
0EH	モバイル インテル Pentium 4 プロセッサ -M: プロセッサ・シグニチャ = 00000F13h の場合、インテル Xeon プロセッサ
0FH	モバイル インテル Celeron プロセッサ [1]
13H	モバイル インテル Celeron プロセッサ [1]
16H	インテル® Pentium® M プロセッサ
17H - 0FFH	予約済み。

1. インテル Pentium III プロセッサ以降に発表されたプロセッサのバージョンを指す。

IA-32 アーキテクチャにおける互換性

CPUID 命令は、初期モデルの Intel486™ プロセッサまたは Intel486 プロセッサよりも以前のすべての IA-32 プロセッサでサポートされていない。

CPUID—CPU Identification（続き）

操作

```
CASE (EAX) OF
    EAX = 0:
        EAX ← highest basic function input value understood by CPUID;
        EBX ← Vendor identification string;
        EDX ← Vendor identification string;
        ECX ← Vendor identification string;
    BREAK;
    EAX = 1H:
        EAX[3:0] ← Stepping ID;
        EAX[7:4] ← Model;
        EAX[11:8] ← Family;
        EAX[13:12] ← Processor type;
        EAX[15:14] ← Reserved;
        EAX[19:16] ← Extended Model;
        EAX[23:20] ← Extended Family;
        EAX[31:24] ← Reserved;
        EBX[7:0] ← Brand Index;
        EBX[15:8] ← CLFLUSH Line Size;
        EBX[16:23] ← Reserved;
        EBX[24:31] ← Initial APIC ID;
        ECX ← Feature flags; (* 図 3-6. を参照 *)
        EDX ← Feature flags; (* 図 3-7. を参照 *)
    BREAK;
    EAX = 2H:
        EAX ← Cache and TLB information;
        EBX ← Cache and TLB information;
        ECX ← Cache and TLB information;
        EDX ← Cache and TLB information;
    BREAK;
    EAX = 3H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← ProcessorSerialNumber[31:0];
        (* Pentium III processors only, otherwise reserved *)
        EDX ← ProcessorSerialNumber[63:32];
        (* Pentium III processors only, otherwise reserved *
    BREAK;
    EAX = 4H:
        EAX ← Deterministic Cache Parameters Leaf; /* see page 3-124 */
        EBX ← Deterministic Cache Parameters Leaf;
        ECX ← Deterministic Cache Parameters Leaf;
        EDX ← Deterministic Cache Parameters Leaf;
    BREAK;
```

CPUID—CPU Identification（続き）

```
EAX = 5H:
    EAX ← MONITOR/MWAIT Leaf; /* see page 3-125 */
    EBX ← MONITOR/MWAIT Leaf;
    ECX ← MONITOR/MWAIT Leaf;
    EDX ← MONITOR/MWAIT Leaf;
BREAK;
EAX = 80000000H:
    EAX ← highest extended function input value understood by CPUID;
    EBX ← Reserved;
    ECX ← Reserved;
    EDX ← Reserved;
BREAK;
EAX = 80000001H:
    EAX ← Extended Processor Signature and Feature Bits (*Currently Reserved*);
    EBX ← Reserved;
    ECX ← Reserved;
    EDX ← Reserved;
BREAK;
EAX = 80000002H:
    EAX ← Processor Brand String;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000003H:
    EAX ← Processor Brand String, continued;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000004H:
    EAX ← Processor Brand String, continued;
    EBX ← Processor Brand String, continued;
    ECX ← Processor Brand String, continued;
    EDX ← Processor Brand String, continued;
BREAK;
EAX = 80000005H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Reserved = 0;
    EDX ← Reserved = 0;
BREAK;
EAX = 80000006H:
    EAX ← Reserved = 0;
    EBX ← Reserved = 0;
    ECX ← Cache information;
    EDX ← Reserved = 0;
BREAK;
```

CPUID—CPU Identification（続き）

```
    EAX = 80000007H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = 80000008H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    DEFAULT: (* EAX > highest value recognized by CPUID *)
        EAX ← Reserved; (* undefined*)
        EBX ← Reserved; (* undefined*)
        ECX ← Reserved; (* undefined*)
        EDX ← Reserved; (* undefined*)
    BREAK;
ESAC;
```

影響を受けるフラグ

なし。

例外（すべての操作モード）

なし。

注記

CPUID 命令をサポートしていない初期の IA-32 プロセッサでは、CPUID 命令を実行すると、無効オペコード（#UD）例外が発生する。

CVTDQ2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values

オペコード	命令	説明
F3 0F E6	CVTDQ2PD *xmm1*, *xmm2/m64*	*xmm2/m128* の 2 つのパックド符号付きダブルワード整数を *xmm1* の 2 つのパックド倍精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドが XMM レジスタの場合は、パックド整数はレジスタの下位クワッドワードに置かれる。

操作

```
DEST[63-0] ← Convert_Integer_To_Double_Precision_Floating_Point(SRC[31-0]);
DEST[127-64] ← Convert_Integer_To_Double_Precision_Floating_Point(SRC[63-32]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTDQ2PD　　__m128d _mm_cvtepi32_pd(__m128di a)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

CVTDQ2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values（続き）

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTDQ2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 5B /r	CVTDQ2PS xmm1, xmm2/m128	xmm2/m128 の 4 つのパックド符号付きダブルワード整数を xmm1 の 4 つのパックド単精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 4 つのパックド符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 4 つのパックド単精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸めが実行される。

操作

```
DEST[31-0] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[31-0]);
DEST[63-32] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[63-32]);
DEST[95-64] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[95-64]);
DEST[127-96] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTDQ2PS　　__m128d _mm_cvtepi32_ps(__m128di a)

SIMD 浮動小数点例外

精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTDQ2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPD2DQ—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
F2 0F E6	CVTPD2DQ *xmm1*, *xmm2/m128*	*xmm2/m128* の 2 つのパックド倍精度浮動小数点値を *xmm1* の 2 つのパックド符号付きダブルワード整数に変換する。

ソース・オペランド（第 2 オペランド）の 2 つのパックド倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードはすべて 0 にクリアされる。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer(SRC[63-0]);
DEST[63-32] ← Convert_Double_Precision_Floating_Point_To_Integer(SRC[127-64]);
DEST[127-64] ← 0000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPD2DQ　　__m128d _mm_cvtpd_epi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTPD2DQ—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPD2PI—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
66 0F 2D /r	CVTPD2PI *mm*, *xmm/m128*	*xmm/m128* の 2 つのパックド倍精度浮動小数点値を *mm* の 2 つのパックド符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは MMX® テクノロジ・レジスタである。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTPD2PI 命令を実行する前にその例外が処理される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer(SRC[63-0]);
DEST[63-32]  ← Convert_Double_Precision_Floating_Point_To_Integer(SRC[127-64]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPD2PI　　__m64 _mm_cvtpd_pi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

CVTPD2PI—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#MF	未処理の x87 FPU 例外がある場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPD2PS—Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values

オペコード	命令	説明
66 0F 5A /r	CVTPD2PS *xmm1*, *xmm2/m128*	*xmm2/m128* の 2 つのパックド倍精度浮動小数点値を *xmm1* の 2 つのパックド単精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド単精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードはすべて 0 にクリアされる。変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_To_Single_Precision_
             Floating_Point(SRC[63-0]);
DEST[63-32]  ← Convert_Double_Precision_To_Single_Precision_
               Floating_Point(SRC[127-64]);
DEST[127-64]  ← 0000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPD2PS　　__m128d _mm_cvtpd_ps(__m128d a)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。 セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTPD2PS—Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPI2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 2A /r	CVTPI2PD *xmm*, *mm/m64*	*mm/mem64* の 2 つのパックド符号付きダブルワード整数を *xmm* の 2 つのパックド倍精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値に変換する。ソース・オペランドは、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTPI2PD 命令を実行する前にその例外が処理される。

操作

```
DEST[63-0] ← Convert_Integer_To_Double_Precision_Floating_Point(SRC[31-0]);
DEST[127-64] ← Convert_Integer_To_Double_Precision_Floating_Point(SRC[63-32]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPI2PD　　__m128d _mm_cvtpi32_pd(__m64 a)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。

CVTPI2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values（続き）

#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTPI2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 2A /r	CVTPI2PS *xmm*, *mm/m64*	*mm/m64* の 2 つのパックド符号付きダブルワード整数を *xmm* の 2 つのパックド単精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド単精度浮動小数点値に変換する。ソース・オペランドは、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードは変更されない。変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって、返される値が丸められる。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTPI2PS 命令を実行する前にその例外が処理される。

操作

```
DEST[31-0] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[31-0]);
DEST[63-32] ← Convert_Integer_To_Single_Precision_Floating_Point(SRC[63-32]);
* high quadword of destination remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPI2PS　　__m128 _mm_cvtpi32_ps(__m128 a, __m64 b)

SIMD 浮動小数点例外

精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。

CVTPI2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values（続き）

#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTPS2DQ—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
66 0F 5B /r	CVTPS2DQ *xmm1*, *xmm2/m128*	*xmm2/m128* の 4 つのパックド単精度浮動小数点値を *xmm1* の 4 つのパックド符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 4 つのパックド単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 4 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[31-0]);
DEST[63-32] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[63-32]);
DEST[95-64] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[95-64]);
DEST[127-96] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_cvtps_epi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

CVTPS2DQ—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers（続き）

#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPS2PD—Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values

オペコード	命令	説明
0F 5A /r	CVTPS2PD *xmm1*, *xmm2/m64*	*xmm2/m64*の2つのパックド単精度浮動小数点値を*xmm1*の 2 つのパックド倍精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドが XMM レジスタの場合は、パックド単精度浮動小数点値はレジスタの下位クワッドワードに置かれる。

操作

```
DEST[63-0] ← Convert_Single_Precision_To_Double_Precision_
                Floating_Point(SRC[31-0]);
DEST[127-64] ← Convert_Single_Precision_To_Double_Precision_
                  Floating_Point(SRC[63-32]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTPS2PD　　__m128d _mm_cvtps_pd(__m128 a)

SIMD 浮動小数点例外

無効。デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

CVTPS2PD—Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTPS2PI—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
0F 2D /r	CVTPS2PI *mm*, xmm/m64	*xmm/m64* の 2 つのパックド単精度浮動小数点値を、*mm* の 2 つのパックド符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは MMX® テクノロジ・レジスタである。ソース・オペランドが XMM レジスタの場合は、2 つの単精度浮動小数点値はレジスタの下位クワッドワードに置かれる。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換された結果が符号付きダブルワード整数の最大値より大きい場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTPS2PI 命令を実行する前にその例外が処理される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[31-0]);
DEST[63-32]  ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[63-32]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m64 _mm_cvtps_pi32(__m128 a)

SIMD 浮動小数点例外

無効、精度。

CVTPS2PI—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#MF	未処理の x87 FPU 例外がある場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer

オペコード	命令	説明
F2 0F 2D /r	CVTSD2SI *r32*, *xmm/m64*	*xmm/m64* の 1 つの倍精度浮動小数点値を *r32* の 1 つの符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは汎用レジスタである。ソース・オペランドが XMM レジスタの場合は、倍精度浮動小数点値はレジスタの下位クワッドワードに置かれる。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer(SRC[63-0]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_cvtsd_si32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSD2SS—Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 5A /r	CVTSD2SS *xmm1*, *xmm2/m64*	*xmm2/m64* の 1 つの倍精度浮動小数点値を *xmm1* の 1 つの単精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの単精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドが XMM レジスタの場合は、倍精度浮動小数点値はレジスタの下位クワッドワードに置かれる。変換の結果はデスティネーション・オペランドの最下位のダブルワードに格納され、上位 3 つのダブルワードは変更されない。変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_To_Single_Precision_
                    Floating_Point(SRC[63-0]);
* DEST[127-32] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTSD2SS　　__m128_mm_cvtsd_ss(__m128d a, __m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTSD2SS—Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 2A /r	CVTSI2SD *xmm, r/m32*	*r/m32* の 1 つの符号付きダブルワード整数を *xmm* の 1 つの倍精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 1 つの倍精度浮動小数点値に変換する。ソース・オペランドは、汎用レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードは変更されない。

操作

```
DEST[63-0] ← Convert_Integer_To_Double_Precision_Floating_Point(SRC[31-0]);
* DEST[127-64] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_cvtsd_si32(__m128d a)

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 2A /r	CVTSI2SS *xmm*, *r/m32*	*r/m32* の 1 つの符号付きダブルワード整数を *xmm* の 1 つの単精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの符号付きダブルワード整数を、デスティネーション・オペランド（第 1 オペランド）の 1 つの単精度浮動小数点値に変換する。ソース・オペランドは、汎用レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの最下位のダブルワードに格納され、上位 3 つのダブルワードは変更されない。変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。

操作

```
DEST[31-0] ← Convert_Inteter_To_Single_Precision_Floating_Point(SRC[31-0]);
* DEST[127-32] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128_mm_cvtsi32_ss(__m128d a, int b)

SIMD 浮動小数点例外

精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSS2SD—Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 5A /r	CVTSS2SD *xmm1*, *xmm2/m32*	*xmm2/m32* の 1 つの単精度浮動小数点値を *xmm1* の 1 つの倍精度浮動小数点値に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの倍精度浮動小数点値に変換する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドが XMM レジスタの場合は、単精度浮動小数点値はレジスタの最下位のダブルワードに置かれる。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードは変更されない。

操作

```
DEST[63-0] ← Convert_Single_Precision_To_Double_Precision_
                Floating_Point(SRC[31-0]);
* DEST[127-64] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTSS2SD　__m128d_mm_cvtss_sd(__m128d a, __m128 b)

SIMD 浮動小数点例外

無効。デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTSS2SD—Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer

オペコード	命令	説明
F3 0F 2D /r	CVTSS2SI *r32*, *xmm/m32*	*xmm/m32* の 1 つの単精度浮動小数点値を *r32* の 1 つの符号付きダブルワード整数に変換し、結果を整数レジスタに移動する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは汎用レジスタである。ソース・オペランドが XMM レジスタの場合は、単精度浮動小数点値はレジスタの最下位のダブルワードに置かれる。

変換が不正確な場合は、MXCSR レジスタの丸め制御ビットにしたがって丸められた値が返される。変換された結果が符号付きダブルワード整数の最大値より大きい場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer(SRC[31-0]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_cvtss_si32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTTPD2PI—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
66 0F 2C /r	CVTTPD2PI *mm*, *xmm/m128*	切り捨てを使用して、*xmm/m128* の 2 つのパックド倍精度浮動小数点値を *mm* の 2 つのパックド符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM テクノロジ・レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは MMX® テクノロジ・レジスタである。

変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTPD2PI 命令を実行する前にその例外が処理される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer_
                Truncate(SRC[63-0]);
DEST[63-32]  ← Convert_Double_Precision_Floating_Point_To_Integer_
                  Truncate(SRC[127-64]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTTPD2PI　　__m64 _mm_cvttpd_pi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

CVTTPD2PI—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#MF	未処理の x87 FPU 例外がある場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTTPD2DQ—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
66 0F E6	CVTTPD2DQ *xmm1*, *xmm2/m128*	切り捨てを使用して、*xmm2/m128* の 2 つのパック ド倍精度浮動小数点値を *xmm1* の 2 つのパック ド符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパック ド倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパック ド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換の結果はデスティネーション・オペランドの下位クワッドワードに格納され、上位クワッドワードはすべて 0 にクリアされる。

変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer_
                Truncate(SRC[63-0]);
DEST[63-32] ← Convert_Double_Precision_Floating_Point_To_Integer_
                Truncate(SRC[127-64]);
DEST[127-64] ← 0000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

CVTTPD2DQ　__m128i _mm_cvttpd_epi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

CVTPD2DQ—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers（続き）

#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTTPS2DQ—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
F3 0F 5B /r	CVTTPS2DQ *xmm1*, *xmm2/m128*	切り捨てを使用して、*xmm2/m128* の 4 つの単精度浮動小数点値を *xmm1* の 4 つの符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 4 つのパックド単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 4 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[31-0]);
DEST[63-32] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[63-32]);
DEST[95-64] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[95-64]);
DEST[127-96] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_cvttps_epi32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

CVTTPS2DQ—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers（続き）

#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

CVTTPS2PI—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

オペコード	命令	説明
0F 2C /r	CVTTPS2PI *mm*, *xmm/m64*	切り捨てを使用して、*xmm/m64* の下位の 2 つの単精度浮動小数点値を *mm* の 2 つの符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 2 つのパックド単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 2 つのパックド符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは MMX® テクノロジ・レジスタである。ソース・オペランドが XMM レジスタの場合は、2 つの単精度浮動小数点値はレジスタの下位クワッドワードに置かれる。

変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換された結果が符号付きダブルワード整数の最大値より大きい場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、CVTTPS2PI 命令を実行する前にその例外が処理される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[31-0]);
DEST[63-32]  ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[63-32]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m64 _mm_cvttps_pi32(__m128 a)

SIMD 浮動小数点例外

無効、精度。

CVTTPS2PI—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#MF	未処理の x87 FPU 例外がある場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTTSD2SI—Convert with Truncation Scalar Double-Precision Floating-Point Value to Signed Doubleword Integer

オペコード	命令	説明
F2 0F 2C /r	CVTTSD2SI *r32*, *xmm/m64*	切り捨てを使用して、*xmm/m64* の 1 つの倍精度浮動小数点値を *r32* の 1 つの符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの倍精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは汎用レジスタである。ソース・オペランドが XMM レジスタの場合は、倍精度浮動小数点値はレジスタの下位クワッドワードに置かれる。

変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換の結果が符号付きダブルワード整数の最大値より大きくなる場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Double_Precision_Floating_Point_To_Integer_
                Truncate(SRC[63-0]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_cvttsd_si32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTTSD2SI—Convert with Truncation Scalar Double-Precision Floating-Point Value to Doubleword Integer（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer

オペコード	命令	説明
F3 0F 2C /r	CVTTSS2SI *r32*, *xmm/m32*	切り捨てを使用して、*xmm/32* の 1 つの単精度浮動小数点値を *r32* の 1 つの符号付きダブルワード整数に変換する。

説明

ソース・オペランド（第 2 オペランド）の 1 つの単精度浮動小数点値を、デスティネーション・オペランド（第 1 オペランド）の 1 つの符号付きダブルワード整数に変換する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは汎用レジスタである。ソース・オペランドが XMM レジスタの場合は、単精度浮動小数点値はレジスタの最下位のダブルワードに置かれる。

変換が不正確な場合は、切り捨てられた（ゼロに丸められる）結果が返される。変換された結果が符号付きダブルワード整数の最大値より大きい場合は、浮動小数点無効例外が発生する。この例外がマスクされている場合は、整数不定値（80000000H）が返される。

操作

```
DEST[31-0] ← Convert_Single_Precision_Floating_Point_To_Integer_
                Truncate(SRC[31-0]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_cvttss_si32(__m128d a)

SIMD 浮動小数点例外

無効、精度。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

CWD/CDQ—Convert Word to Doubleword/Convert Doubleword to Quadword

オペコード	命令	説明
99	CWD	DX:AX ← AX の符号拡張
99	CDQ	EDX:EAX ← EAX の符号拡張

説明

（オペランド・サイズにより）AX または EAX レジスタ内のオペランドのサイズを符号拡張によって 2 倍に拡張し、結果をそれぞれ DX:AX または EDX:EAX レジスタにストアする。CWD 命令は、AX レジスタの値の符号（ビット 15）を DX レジスタのすべてのビット位置にコピーする（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、上巻』の図 7-6. を参照）。CDQ 命令は、EAX レジスタの値の符号（ビット 31）を EDX レジスタのすべてのビット位置にコピーする。

CWD 命令を使用すると、ワード除算の前にワードからダブルワードの被除数を作れ、CDQ 命令を使用すると、ダブルワード除算の前にダブルワードからクワッドワードの被除数を作れる。

CWD および CDQ ニーモニックは同じオペコードを参照する。CWD 命令はオペランド・サイズ属性が 16 のとき、また CDQ 命令はオペランド・サイズ属性が 32 のときに使用することを目的としている。一部のアセンブラには、CWD が使用されたときはオペランド・サイズを 16 ビットに、また CDQ が使用されたときは 32 ビットに強制できるものがある。その他のアセンブラは、これらのニーモニックを同義語（CWD/CDQ）として取り扱い、どちらのニーモニックが使用されても、そのときのオペランド・サイズ属性の設定を使用して変換対象の値のサイズを判定できる。

操作

```
IF OperandSize = 16 (* CWD instruction *)
    THEN DX ← SignExtend(AX);
    ELSE (* OperandSize = 32, CDQ instruction *)
        EDX ← SignExtend(EAX);
FI;
```

影響を受けるフラグ

なし。

例外（すべての操作モード）

なし。

CWDE—Convert Word to Doubleword

「CBW/CWDE—Convert Byte to Word/Convert Word to Doubleword」を参照のこと。

DAA—Decimal Adjust AL after Addition

オペコード	命令	説明
27	DAA	加算後に AL を 10 進調整する。

説明

2 つのパック BCD 値の和を調整して、パック BCD 結果を作成する。AL レジスタは、この命令の暗黙のソース兼デスティネーション・オペランドである。DAA 命令は、2 ケタのパック BCD 値を加算（2 進加算）し、バイト結果を AL レジスタにストアする ADD 命令の次に実行したときに限り有効である。そこで、DAA 命令が AL レジスタの内容を調整して、正しい 2 ケタのパック BCD 結果にまとめる。10 進キャリーが検出された場合は、キャリーにしたがって CF および AF フラグがセットされる。

操作

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) OR AF = 1)
    THEN
        AL ← AL + 6;
        CF ← old_CF OR (Carry from AL ← AL + 6);
        AF ← 1;
    ELSE
        AF ← 0;
FI;
IF ((old_AL > 99H) OR (old_CF = 1))
    THEN
        AL ← AL + 60H;
        CF ← 1;
    ELSE
        CF ← 0;
FI;
```

例

```
ADD AL, BL Before: AL=79H  BL=35H EFLAGS(OSZAPC)=XXXXXX
           After:  AL=AEH  BL=35H EFLAGS(0SZAPC)=110000
DAA        Before: AL=AEH  BL=35H EFLAGS(OSZAPC)=110000
           After:  AL=14H  BL=35H EFLAGS(0SZAPC)=X00111
DAA        Before: AL=2EH  BL=35H EFLAGS(OSZAPC)=110000
           After:  AL=04H  BL=35H EFLAGS(0SZAPC)=X00101
```

影響を受けるフラグ

値の調整によって結果のどちらかの桁に 10 進キャリーが生じた場合は、CF および AF フラグがセットされる（上記の「操作」の項を参照）。SF、ZF、PF フラグが結果にしたがってセットされる。OF フラグは未定義。

DAA—Decimal Adjust AL after Addition（続き）

例外（すべての操作モード）

なし。

DAS—Decimal Adjust AL after Subtraction

オペコード	命令	説明
2F	DAS	減算後に AL を 10 進調整する。

説明

2 つのパック BCD 値間の減算結果を調整して、パック BCD 結果を作成する。AL レジスタは、この命令の暗黙のソース兼デスティネーション・オペランドである。DAS 命令は、一方の 2 ケタのパック BCD 値からもう一方の 2 ケタのパック BDC 値を引き（2 進減算）、バイトの結果を AL レジスタにストアする SUB 命令の次に実行したときに限り有効である。そこで、DAS 命令が AL レジスタの内容を調整して、正しい 2 ケタのパック BCD 結果にまとめる。10 進ボローが検出された場合は、ボローにしたがって CF および AF フラグがセットされる。

操作

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) OR AF = 1)
    THEN
        AL ← AL − 6;
        CF ← old_CF OR (Borrow from AL ← AL − 6);
        AF ← 1;
    ELSE
        AF ← 0;
FI;
IF ((old_AL > 99H) OR (old_CF = 1))
    THEN
        AL ← AL − 60H;
        CF ← 1;
    ELSE
        CF ← 0;
FI;
```

例

```
SUB AL, BL Before: AL=35H  BL=47H EFLAGS(OSZAPC)=XXXXXX
           After:  AL=EEH  BL=47H EFLAGS(OSZAPC)=010111
DAA        Before: AL=EEH  BL=47H EFLAGS(OSZAPC)=010111
           After:  AL=88H  BL=47H EFLAGS(OSZAPC)=X10111
```

影響を受けるフラグ

値の調整によって結果のどちらかの桁に 10 進ボローが生じた場合は、CF および AF フラグがセットされる（上記の「操作」の項を参照）。SF、ZF、PF フラグが結果にしたがってセットされる。OF フラグは未定義。

DAS—Decimal Adjust AL after Subtraction（続き）

例外（すべての操作モード）

なし。

DEC—Decrement by 1

オペコード	命令	説明
FE /1	DEC *r/m8*	*r/m8* を 1 デクリメントする。
FF /1	DEC *r/m16*	*r/m16* を 1 デクリメントする。
FF /1	DEC *r/m32*	*r/m32* を 1 デクリメントする。
48+rw	DEC *r16*	*r16* を 1 デクリメントする。
48+rd	DEC *r32*	*r32* を 1 デクリメントする。

説明

CF フラグの状態を変えないで、デスティネーション・オペランドから 1 を引く（デクリメントする）。デスティネーション・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。この命令では、CF フラグの状態を変えることなくループカウンタを更新できる。（CF フラグを更新するデクリメント操作を行うには、値 1 の即値オペランドを使用して SUB 命令を実行する。）

この命令を LOCK プリフィックスと共に使用すると、アトミックに命令を実行させることができる。

操作

DEST ← DEST – 1;

影響を受けるフラグ

CF フラグは影響を受けない。OF、SF、ZF、AF、PF フラグが結果にしたがってセットされる。

保護モード例外

#GP(0)	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

DEC—Decrement by 1（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

DIV—Unsigned Divide

オペコード	命令	説明
F6 /6	DIV *r/m8*	AX を *r/m8* で符号なし除算する。結果は次のようにストアされる。 AL ←商、AH ←剰余
F7 /6	DIV *r/m16*	DX:AX を *r/m16* で符号なし除算する。結果は次のようにストアされる。 AX ←商、DX ←剰余
F7 /6	DIV *r/m32*	EDX:EAX を *r/m32* で符号なし除算する。結果は次のようにストアされる。 EAX ←商、EDX ←剰余

説明

AX レジスタ、DX:AX レジスタ、または EDX:EAX レジスタ内の値（被除数）をソース・オペランド（除数）で（符号なしで）割り、結果をそれぞれ AX（AH:AL）、DX:AX、または EDX:EAX レジスタにストアする。ソース・オペランドには、汎用レジスタまたはメモリ・ロケーションを使用できる。この命令の処理は、以下の表に示すように、オペランド・サイズ（被除数 / 除数）に依存する。

オペランド・サイズ	被除数	除数	商	剰余	商の最大値
ワード / バイト	AX	r/m8	AL	AH	255
ダブルワード / ワード	DX:AX	r/m16	AX	DX	65,535
クワッドワード / ダブルワード	EDX:EAX	r/m32	EAX	EDX	$2^{32} - 1$

非整数の結果は 0 に向かって切り捨てられる。剰余の絶対値は常に除数のそれより小さい。オーバーフローは、CF フラグではなく、#DE（除算エラー）例外で示される。

DIV—Unsigned Divide（続き）

操作

```
IF SRC = 0
    THEN #DE; (* divide error *)
FI;
IF OpernadSize = 8 (* word/byte operation *)
    THEN
        temp ← AX / SRC;
        IF temp > FFH
            THEN #DE; (* divide error *) ;
            ELSE
                AL ← temp;
                AH ← AX MOD SRC;
        FI;
    ELSE
        IF OperandSize = 16 (* doubleword/word operation *)
            THEN
                temp ← DX:AX / SRC;

                IF temp > FFFFH
                    THEN #DE; (* divide error *) ;
                    ELSE
                        AX ← temp;
                        DX ← DX:AX MOD SRC;
                FI;
            ELSE (* quadword/doubleword operation *)
                temp ← EDX:EAX / SRC;
                IF temp > FFFFFFFFH
                    THEN #DE; (* divide error *) ;
                    ELSE
                        EAX ← temp;
                        EDX ← EDX:EAX MOD SRC;
                FI;
        FI;
FI;
```

DIV—Unsigned Divide（続き）

影響を受けるフラグ

CF、OF、SF、ZF、AF、および PF フラグは未定義。

保護モード例外

#DE	ソース・オペランド（除数）が 0 の場合。
	商が大きすぎて、指定されたレジスタにストアできない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#DE	ソース・オペランド（除数）が 0 の場合。
	商が大きすぎて、指定されたレジスタにストアできない場合。
#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#DE	ソース・オペランド（除数）が 0 の場合。
	商が大きすぎて、指定されたレジスタにストアできない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

DIVPD—Divide Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 5E /r	DIVPD *xmm1*, *xmm2/m128*	*xmm1* のパック ド倍精度浮動小数点値を *xmm2/m128* のパック ド倍精度浮動小数点値で割る。

説明

デスティネーション・オペランド（第 1 オペランド）の 2 つのパック ド倍精度浮動小数点値を、ソース・オペランド（第 2 オペランド）の 2 つのパック ド倍精度浮動小数点値で SIMD 除算し、パック ド倍精度浮動小数点の結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。倍精度浮動小数点値の SIMD 演算の図は、『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 11-3. を参照のこと。

操作

```
DEST[63-0] ← DEST[63-0] / (SRC[63-0]);
DEST[127-64] ← DEST[127-64] / (SRC[127-64]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

DIVPD __m128 _mm_div_pd(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、ゼロ除算、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

DIVPD—Divide Packed Double-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

DIVPS—Divide Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 5E /r	DIVPS *xmm1*, *xmm2/m128*	*xmm1* のパック ド単精度浮動小数点値を *xmm2/m128* のパック ド単精度浮動小数点値で割る。

説明

デスティネーション・オペランド（第 1 オペランド）の 2 つのパック ド単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）の 2 つのパック ド単精度浮動小数点値で SIMD 除算し、パック ド倍精度浮動小数点の結果をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。単精度浮動小数点値の SIMD 演算の図は、『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 10-5. を参照のこと。

操作

```
DEST[31-0] ← DEST[31-0] / (SRC[31-0]);
DEST[63-32] ← DEST[63-32] / (SRC[63-32]);
DEST[95-64] ← DEST[95-64] / (SRC[95-64]);
DEST[127-96] ← DEST[127-96] / (SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

DIVPS __m128 _mm_div_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、ゼロによる除算、精度、デノーマル。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが、CS、DS、ES、FS、または GS セグメントの範囲外の場合。 セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

DIVPS—Divide Packed Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

DIVSD—Divide Scalar Double-Precision Floating-Point Values

オペコード	命令	説明
F2 0F 5E /r	DIVSD *xmm1*, *xmm2/m64*	*xmm1* の下位の倍精度浮動小数点値を *xmm2/mem64* の下位の倍精度浮動小数点値で割る。

説明

デスティネーション・オペランド（第1オペランド）の下位の倍精度浮動小数点値を、ソース・オペランド（第2オペランド）の下位の倍精度浮動小数点値で割り、結果の倍精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位クワッドワードは変更されない。倍精度浮動小数点値のスカラ演算の図は、『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 11-4. を参照のこと。

操作

```
DEST[63-0]  ← DEST[63-0] / SRC[63-0];
* DEST[127-64] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

DIVSD __m128d _mm_div_sd (m128d a, m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、ゼロ除算、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

DIVSD—Divide Scalar Double-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

DIVSS—Divide Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F 5E /r	DIVSS *xmm1*, *xmm2/m32*	*xmm1* の下位の単精度浮動小数点値を *xmm2/m32* の下位の単精度浮動小数点値で割る。

説明

デスティネーション・オペランド（第 1 オペランド）の最下位の単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）の最下位の単精度浮動小数点値で割り、結果の単精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位 3 つのダブルワードは変更されない。単精度浮動小数点値のスカラ演算の図は、『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 10-6. を参照のこと。

操作

```
DEST[31-0]  ← DEST[31-0] / SRC[31-0];
* DEST[127-32] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

DIVSS __m128 _mm_div_ss(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、ゼロによる除算、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

DIVSS—Divide Scalar Single-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

EMMS—Empty MMX Technology State

オペコード	命令	説明
0F 77	EMMS	x87 FP タグワードを空に設定する。

説明

x87 FPU タグワードのすべてのタグの値を空（すべて 1）に設定する。この操作によって、（MMX® テクノロジ・レジスタとして別名定義されている）x87 FPU レジスタは、x87 FPU 浮動小数点命令で使用可能としてマークされる（x87 FPU タグワードのフォーマットについては、『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 7-11. を参照のこと）。その他の（EMMS 命令以外の）すべての MMX テクノロジ命令は、x87 FPU タグワード内のすべてのタグを有効（すべて 0）に設定する。

x87 浮動小数点命令を実行する可能性がある他のプロシージャまたはサブルーチンをコールする前には、すべての MMX テクノロジ・プロシージャまたはサブルーチンの終わりで EMMS 命令を使用して MMX テクノロジの状態をクリアしなければならない。x87 FPU データ・タグ・ワードが EMMS 命令によってリセットされる前に x87 浮動小数点命令が x87 FPU レジスタスタックのレジスタの 1 つに値をロードした場合は、x87 浮動小数点レジスタ・スタック・オーバーフローが発生する可能性があり、その結果、x87 浮動小数点例外が発生したり、誤った結果が生じることになる。

操作

```
x87FPUTagWord ← FFFFH;
```

同等のインテル® C/C++ コンパイラ組み込み関数

void_mm_empty()

影響を受けるフラグ

なし。

保護モード例外

#UD	CR0 の EM がセットされた場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の FPU 例外がある場合。

実アドレスモード例外

保護モード例外と同じ。

EMMS—Empty MMX Technology State（続き）

仮想 8086 モード例外

保護モード例外と同じ。

ENTER—Make Stack Frame for Procedure Parameters

オペコード	命令	説明
C8 iw 00	ENTER *imm16*, 0	プロシージャのスタックフレームを作成する。
C8 iw 01	ENTER *imm16*, 1	プロシージャのネストされたスタックフレームを作成する。
C8 iw ib	ENTER *imm16*, *imm8*	プロシージャのネストされたスタックフレームを作成する。

説明

プロシージャのためのスタックフレームを作成する。第1オペランド（サイズ・オペランド）は、スタックフレームのサイズ（すなわち、プロシージャのスタックに割り当てられる動的記憶域のバイト数）を指定する。第2オペランド（ネスティング・レベル・オペランド）は、プロシージャのレキシカル・ネスティング・レベル（0から31まで）を指定する。ネスティング・レベルによって、以前のスタックフレームから新しいスタックフレームの「表示領域」にコピーされるスタック・フレーム・ポインタの数が決まる。これらのオペランドは共に即値である。

スタックサイズ属性によって、BP（16ビット）、EBP（32ビット）のどちらのレジスタで現在のフレームポインタを指定するか、およびSP（16ビット）、ESP（32ビット）のどちらのレジスタでスタックポインタを指定するかが決まる。

対をなすENTER命令とLEAVE命令は、ブロック構造言語をサポートするために設けられたものである。（使用されるときは）ENTER命令は一般的にプロシージャ内の最初の命令であり、プロシージャの新しいスタックフレームをセットアップするために使用される。次に、LEAVE命令をプロシージャの終わり（RET命令の直前）で使用して、スタックフレームを開放する。

ネスティング・レベルが0の場合は、プロセッサはフレームポインタをEBPレジスタからスタックにプッシュし、現在のスタックポインタをESPレジスタからEBPレジスタにコピーし、ESPレジスタに現在のスタックポインタ値からサイズ・オペランドの値を引いた値をロードする。ネスティング・レベルが1以上の場合は、プロセッサはスタックポインタを調整する前に追加フレームポインタをスタックにプッシュする。それらの追加フレームポインタによって、コール先プロシージャにスタック上の他のネストされたフレームへのアクセスポイントが与えられる。ENTER命令の処理の詳細については、『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6章の「ブロック構造言語でのプロシージャ・コール」を参照のこと。

ENTER—Make Stack Frame for Procedure Parameters（続き）

操作

```
NestingLevel ← NestingLevel MOD 32
IF StackSize = 32
    THEN
        Push(EBP) ;
        FrameTemp ← ESP;
    ELSE (* StackSize = 16*)
        Push(BP);
        FrameTemp ← SP;
FI;
IF NestingLevel ← 0
    THEN GOTO CONTINUE;
FI;
IF (NestingLevel > 0)
    FOR i ← 1 TO (NestingLevel – 1)
        DO
            IF OperandSize = 32
                THEN
                    IF StackSize = 32
                        EBP ← EBP – 4;
                        Push([EBP]); (* doubleword push *)
                    ELSE (* StackSize = 16*)
                        BP ← BP – 4;
                        Push([BP]); (* doubleword push *)
                    FI;
                ELSE (* OperandSize = 16 *)
                    IF StackSize = 32
                        THEN
                            EBP ← EBP – 2;
                            Push([EBP]); (* word push *)
                        ELSE (* StackSize = 16*)
                            BP ← BP – 2;
                            Push([BP]); (* word push *)
                    FI;
            FI;
    OD;
    IF OperandSize = 32
        THEN
            Push(FrameTemp); (* doubleword push *)
        ELSE (* OperandSize = 16 *)
            Push(FrameTemp); (* word push *)
    FI;
    GOTO CONTINUE;
```

ENTER—Make Stack Frame for Procedure Parameters（続き）

```
FI;
CONTINUE:
IF StackSize = 32
    THEN
        EBP ← FrameTemp
        ESP ← EBP – Size;
    ELSE (* StackSize = 16*)
        BP ← FrameTemp
        SP ← BP – Size;
FI;
END;
;
```

影響を受けるフラグ

なし。

保護モード例外

#SS(0)	SP または ESP レジスタの新しい値がスタック・セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#SS(0)	SP または ESP レジスタの新しい値がスタック・セグメントの範囲外の場合。

仮想 8086 モード例外

#SS(0)	SP または ESP レジスタの新しい値がスタック・セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

F2XM1—Compute 2^x-1

オペコード	命令	説明
D9 F0	F2XM1	ST(0) を $(2^{ST(0)} - 1)$ で置き換える。

説明

2 のソース・オペランド乗という指数値から 1 を引いた値を計算する。ソース・オペランドは ST(0) レジスタにあり、結果も ST(0) にストアされる。ソース・オペランドの値は –1.0 から +1.0 までの範囲でなければならない。ソース値がこの範囲外の場合は、結果は未定義になる。

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の指数値を計算したときに得られる結果を示す。

ST(0) SRC	ST(0) DEST
–1.0 ～ –0	–0.5 ～ –0
–0	–0
+0	+0
+0 ～ +1.0	+0 ～ 1.0

2 以外の値の累乗は、以下の式を使用して計算する。

$x^y \leftarrow 2^{(y * \log_2 x)}$

操作

$ST(0) \leftarrow (2^{ST(0)} - 1)$;

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされる。

結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。

C0、C2、C3　未定義。

F2XM1—Compute 2^x-1（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
#D	ソースがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FABS—Absolute Value

オペコード	命令	説明
D9 E1	FABS	ST をその絶対値で置き換える。

説明

ST(0) の符号ビットをクリアして、オペランドの絶対値を作成する。以下の表に、さまざまなクラスの数の絶対値を作成したときに得られる結果を示す。

ST(0) SRC	ST(0) DEST
−∞	+∞
−F	+F
−0	+0
+0	+0
+F	+F
+∞	+∞
NaN	NaN

注：

F 有限浮動小数点値を示す。

操作

ST(0) ← |ST(0)|

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にセットされる。

C0、C2、C3　未定義。

浮動小数点例外

#IS　スタック・アンダーフローが発生した場合。

保護モード例外

#NM　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　CR0 の EM または TS がセットされた場合。

FABS—Absolute Value（続き）

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FADD/FADDP/FIADD—Add

オペコード	命令	説明
D8 /0	FADD *m32fp*	*m32fp* を ST(0) に加え、結果を ST(0) にストアする。
DC /0	FADD *m64fp*	*m64fp* を ST(0) に加え、結果を ST(0) にストアする。
D8 C0+i	FADD ST(0), ST(i)	ST(0) を ST(i) に加え、結果を ST(0) にストアする。
DC C0+i	FADD ST(i), ST(0)	ST(i) を ST(0) に加え、結果を ST(i) にストアする。
DE C0+i	FADDP ST(i), ST(0)	ST(0) を ST(i) に加え、結果を ST(i) にストアし、レジスタスタックをポップする。
DE C1	FADDP	ST(0) を ST(1) に加え、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /0	FIADD *m32int*	*m32int* を ST(0) に加え、結果を ST(0) にストアする。
DE /0	FIADD *m16int*	*m16int* を ST(0) に加え、結果を ST(0) にストアする。

説明

ソース・オペランドをデスティネーション・オペランドに加え、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランドは常に FPU レジスタである。ソース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

この命令のオペランドなしバージョンでは、ST(0) レジスタの内容を ST(1) レジスタに加える。1 オペランド・バージョンでは、メモリ・ロケーションの内容（浮動小数点値または整数値）を ST(0) レジスタに加える。2 オペランド・バージョンでは、ST(0) レジスタの内容を ST(i) レジスタに、またはその逆に加算する。ST(0) の値は、以下のコーディングによって 2 倍にすることができる。

```
FADD ST(0), ST(0);
```

FADDP 命令は、結果をストアした後に、追加操作として FPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。（浮動小数点加算命令のオペランドなしバージョンでは、常にレジスタスタックのポップを伴う。一部のアセンブラでは、この命令のニーモニックは FADDP ではなく FADD になっている。）

FIADD 命令は、整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから加算を行う。

次ページの表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数値を加算したときに得られる結果を示す。

FADD/FADDP/FIADD—Add（続き）

符号が反対の2つのオペランドの和が0のときは、–∞への丸めモードの場合を除いて、結果は+0である。–∞方向への丸めモードの場合は、結果は–0である。ソース・オペランドは、整数0のときは+0として取り扱われる。

両方のオペランドが同じ符号で無限大のときは、結果は予期される符号の∞である。両方のオペランドが反対符号で無限大の場合は、無効操作例外が発生する。

		DEST						
		–∞	–F	–0	+0	+F	+∞	NaN
SRC	–∞	–∞	–∞	–∞	–∞	–∞	*	NaN
	–F or –I	–∞	–F	SRC	SRC	±F or ±0	+∞	NaN
	–0	–∞	DEST	–0	±0	DEST	+∞	NaN
	+0	–∞	DEST	±0	+0	DEST	+∞	NaN
	+F or +I	–∞	±F or ±0	SRC	SRC	+F	+∞	NaN
	+∞	*	+∞	+∞	+∞	+∞	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限実数を示す。

I 整数を示す。

* 浮動小数点無効算術オペランド例外（#IA）を示す。

操作

```
IF instruction is FIADD
    THEN
        DEST ← DEST + ConvertToDoubleExtendedPrecisionFP(SRC);
    ELSE (* source operand is floating-point value *)
        DEST ← DEST + SRC;
FI;
IF instruction = FADDP
    THEN
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は0にセットされる。

結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。

C0、C2、C3　未定義。

FADD/FADDP/FIADD—Add（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
	両方のオペランドの符号が反対で、絶対値が無限大の場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FADD/FADDP/FIADD—Add（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FBLD—Load Binary Coded Decimal

オペコード	命令	説明
DF /4	FBLD *m80 dec*	BCD 値を浮動小数点に変換し、FPU スタックにプッシュする。

説明

BCD のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換し、変換結果の値を FPU スタックにプッシュする。ソース・オペランドは丸めエラーなしにロードされる。ソース・オペランドの符号は、–0 の場合の符号を含めて保持される。

パック BCD 数字は 0 から 9 までの範囲とみなされる。すなわち、この命令は無効な数字（AH から FH）の有無をチェックしない。無効なコードをロードしようとすると、結果は未定義になる。

操作

```
TOP ← TOP – 1;
ST(0) ← ConvertToDoubleExtendedPrecisionFP(SRC);
```

FPU 影響を受けるフラグ

C1	スタック・オーバーフローが発生した場合は 1 にセットされ、発生しなかった場合は 0 にクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・オーバーフローが発生した場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FBLD—Load Binary Coded Decimal（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FBSTP—Store BCD Integer and Pop

オペコード	命令	説明
DF /6	FBSTP m80bcd	ST(0) を m80bcd にストアし、ST(0) をポップする。

説明

ST(0) レジスタの値を 18 ケタのパック BCD 整数に変換し、結果をデスティネーション・オペランドにストアし、レジスタスタックをポップする。ソース値は、非整数値の場合、FPU 制御ワードの RC フィールドによって指定される丸めモードにしたがって丸められる。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。

デスティネーション・オペランドは、デスティネーション値の最初のバイトがストアされるアドレスを指定する。結果の BCD 値は、（その符号値を含めて）10 バイトのメモリ空間を必要とする。

以下の表に、さまざまなクラスのパック BCD フォーマットの数をストアしたときに得られる結果を示す。

ST(0)	DEST
–∞ または DEST 形式には値が大きすぎる	*
F < –1	–D
–1 < F < –0	**
–0	–0
+0	+0
+0 < F < +1	**
F > +1	+D
+∞ または DEST 形式には値が大きすぎる	*
NaN	*

注：

F 有限浮動小数点値を示す。

D パック BCD 数を示す。

* 浮動小数点無効操作（#IA）例外を示す。

** 丸めモードによって、±0 または ±1。

FBSTP—Store BCD Integer and Pop（続き）

変換された値が大きすぎてデスティネーション・フォーマットで表現できない場合や、ソース・オペランドが ∞、SNaN、QNAN、またはサポートしていないフォーマットである場合は、無効算術オペランド条件が発生する。無効操作例外がマスクされていない場合、無効算術オペランド例外が発生し、デスティネーション・オペランドに値はストアされない。無効操作例外がマスクされている場合は、パック BCD の未定義値がメモリにストアされる。

操作

```
DEST ← BCD(ST(0));
PopRegisterStack;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	変換された値の長さが 18 ケタの BCD 数を超えている場合。 ソース・オペランドが SNaN、QNaN、±∞、またはサポートしていないフォーマットの場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	セグメント・レジスタに書き込み不可能なセグメントを指示先とするセグメント・セレクタがロードされようとした場合。 メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FBSTP—Store BCD Integer and Pop（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FCHS—Change Sign

オペコード	命令	説明
D9 E0	FCHS	ST(0) の符号を反転する。

説明

ST(0) の符号を反転する。この操作は、正の値を同じ絶対値の負の値に、またはその逆に変換する。以下の表に、さまざまなクラスの数の符号を反転したときに得られる結果を示す。

ST(0) SRC	ST(0) DEST
−∞	+∞
−F	+F
−0	+0
+0	−0
+F	−F
+∞	−∞
NaN	NaN

注：

F 有限浮動小数点値を示す。

操作

SignBit(ST(0)) ← NOT (SignBit(ST(0)))

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にセットされる。

C0、C2、C3　未定義。

浮動小数点例外

#IS　スタック・アンダーフローが発生した場合。

保護モード例外

#NM　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　CR0 の EM または TS がセットされた場合。

FCHS—Change Sign（続き）

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FCLEX/FNCLEX—Clear Exceptions

オペコード	命令	説明
9B DB E2	FCLEX	未処理のマスクされていない浮動小数点例外の有無をチェックした後、浮動小数点例外フラグをクリアする。
DB E2	FNCLEX*	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、浮動小数点例外フラグをクリアする。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

これらの命令は、FPU ステータス・ワード内の浮動小数点例外フラグ（PE、UE、OE、ZE、DE、および IE）、例外サマリ・ステータス・フラグ（ES）、スタック・フォルト・フラグ（SF）、ビジーフラグ（B）をクリアする。FCLEX 命令は、未処理のマスクされていない浮動小数点例外がないかどうかをチェックしてから各例外フラグをクリアする。FNCLEX 命令はこのチェックを行わない。

FCLEX 命令の場合、アセンブラは 2 つの命令を発行する（つまり、FWAIT 命令に続けて FNCLEX 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサまたは Intel486™ プロセッサを MS-DOS* 互換モードで動作させたときは、（通常の状況下で）実行される前に FNCLEX 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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 の「非同期型命令のウインドウ内の x87 FPU 割り込み」の説明を参照のこと。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・プロセッサでは、FNCLEX 命令にこの方法では割り込みをかけられない。

この命令は、x87 FPU 浮動小数点例外フラグにのみ影響を与える。この命令は、MXCSR レジスタの SIMD 浮動小数点例外フラグには影響を与えない。

操作

```
FPUStatusWord[0..7] ← 0;
FPUStatusWord[15] ← 0;
```

FCLEX/FNCLEX—Clear Exceptions（続き）

FPU 影響を受けるフラグ

FPU ステータス・ワード内の PE、UE、OE、ZE、DE、IE、ES、SF、B フラグがクリアされる。C0、C1、C2、C3 フラグは未定義。

浮動小数点例外

なし。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FCMOV*cc*—Floating-Point Conditional Move

オペコード	命令	説明
DA C0+i	FCMOVB ST(0), ST(i)	より下（CF=1）の場合転送する。
DA C8+i	FCMOVE ST(0), ST(i)	等しい（ZF=1）場合転送する。
DA D0+i	FCMOVBE ST(0), ST(i)	より下か等しい（CF=1 または ZF=1）場合転送する。
DA D8+i	FCMOVU ST(0), ST(i)	順序付けなし（PF=1）の場合転送する。
DB C0+i	FCMOVNB ST(0), ST(i)	より下でない（CF=0）場合転送する。
DB C8+i	FCMOVNE ST(0), ST(i)	等しくない（ZF=0）場合転送する。
DB D0+i	FCMOVNBE ST(0), ST(i)	より下でなく等しくない(CF=0 および ZF=0)場合転送する。
DB D8+i	FCMOVNU ST(0), ST(i)	順序付け（PF=0）の場合転送する。

説明

EFLAGS レジスタ内のステータス・フラグをテストし、与えられたテスト条件が真の場合、ソース・オペランド（第 2 オペランド）をデスティネーション・オペランド（第 1 オペランド）に転送する。各ニーモニックの条件は、上記の「説明」の欄、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、上巻』の表 7-4. に示してある。ソース・オペランドは常に ST(i) レジスタにあり、デスティネーション・オペランドは常に ST(0) である。

FCMOV*cc* 命令は小さい IF 構造を最適化する場合に有用である。これらの命令は、さらにIF操作の分岐にかかわるオーバヘッドおよびプロセッサによる分岐の予測ミスを排除する上でも有効である。

プロセッサによっては、FCMOV*cc* 命令をサポートしていないものがある。ソフトウェアで、CPUID 命令（本章の「COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS」を参照）を使用してプロセッサの機能情報を調べることにより、FCMOV*cc* 命令がサポートされているかどうかを確認できる。CMOV および FPU の両機能ビットがセットされていれば、FCMOV*cc* 命令がサポートされている。

IA-32 アーキテクチャにおける互換性

FCMOV*cc* 命令は、P6 ファミリ・プロセッサで IA-32 アーキテクチャに導入され、それより以前の IA-32 プロセッサには備えられていない。

操作

```
IF condition TRUE
    ST(0) ← ST(i)
FI;
```

FCMOV*cc*—Floating-Point Conditional Move（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。

Integer 影響を受けるフラグ

なし。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FCOM/FCOMP/FCOMPP—Compare Floating Point Values

オペコード	命令	説明
D8 /2	FCOM *m32fp*	ST(0) を *m32fp* と比較する。
DC /2	FCOM *m64fp*	ST(0) を *m64fp* と比較する。
D8 D0+i	FCOM ST(i)	ST(0) を ST(i) と比較する。
D8 D1	FCOM	ST(0) を ST(1) と比較する。
D8 /3	FCOMP *m32fp*	ST(0) を *m32fp* と比較し、レジスタスタックをポップする。
DC /3	FCOMP *m64fp*	ST(0) を *m64fp* と比較し、レジスタスタックをポップする。
D8 D8+i	FCOMP ST(i)	ST(0) を ST(i) と比較し、レジスタスタックをポップする。
D8 D9	FCOMP	ST(0) を ST(1) と比較し、レジスタスタックをポップする。
DE D9	FCOMPP	ST(0) を ST(1) と比較し、レジスタスタックを 2 回ポップする。

説明

ST(0) レジスタの内容とソース値を比較し、結果にしたがって FPU ステータス・ワード内の条件コードフラグ C0、C2、C3 をセットする（下記の表を参照）。ソース・オペランドには、データレジスタまたはメモリ・ロケーションを使用できる。ソース・オペランドを指定しない場合は、ST(0) の値は ST(1) の値と比較される。ゼロの符号は無視される。すなわち、-0.0 は +0.0 と等しい。

条件	C3	C2	C0
ST(0) > SRC	0	0	0
ST(0) < SRC	0	0	1
ST(0) = SRC	1	0	0
順序付けなし *	1	1	1

注：

* マスクされていない無効算術オペランド（#IA）例外が発生しても、フラグはセットされない。

この命令は比較対象の両数値のクラスを調べる（本章の「FXAM—Examine」を参照）。いずれかのオペランドが NaN であるか、またはそのフォーマットがサポートされていない場合は、無効算術オペランド例外（#IA）が発生し、さらに、その例外がマスクされている場合は、条件フラグが「順序付けなし」に設定される。無効算術オペランド例外がマスクされていない場合は、条件コードフラグはセットされない。

FCOMP 命令は比較操作の後にレジスタスタックをポップし、FCOMPP 命令は比較操作の後にレジスタスタックを 2 回ポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。

FCOM/FCOMP/FCOMPP—Compare Floating Point Values（続き）

FCOM 命令が行う操作は、QNaN オペランドの取り扱い方を除いて FUCOM 命令のそれと同じである。FCOM 命令は、一方または両方のオペランドが NaN 値であるか、またはそれらのフォーマットがサポートされていないときは、無効算術オペランド例外（#IA）を発生する。FUCOM 命令が行う操作は、オペランドが QNaN 値であっても無効算術オペランド例外を発生しない点を除いて FCOM の操作と同じである。

操作

```
CASE (relation of operands) OF
    ST > SRC:    C3, C2, C0 ← 000;
    ST < SRC:    C3, C2, C0 ← 001;
    ST = SRC:    C3, C2, C0 ← 100;
ESAC;
IF ST(0) or SRC = NaN or unsupported format
    THEN
        #IA
        IF FPUControlWord.IM = 1
            THEN
                C3, C2, C0 ← 111;
        FI;
FI;
IF instruction = FCOMP
    THEN
        PopRegisterStack;
FI;
IF instruction = FCOMPP
    THEN
        PopRegisterStack;
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にクリアされる。
C0、C2、C3	前ページの表を参照。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	一方または両方のオペランドが NaN 値であるか、またはそれらのフォーマットがサポートされていない場合。
	レジスタが空にマークされる場合。
#D	一方または両方のオペランドがデノーマル値である場合。

FCOM/FCOMP/FCOMPP—Compare Floating Point Values（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FCOMI/FCOMIP/ FUCOMI/FUCOMIP—Compare Floating Point Values and Set EFLAGS

オペコード	命令	説明
DB F0+i	FCOMI ST, ST(i)	ST(0) を ST(i) と比較し、結果にしたがってステータス・フラグをセットする。
DF F0+i	FCOMIP ST, ST(i)	ST(0) を ST(i) と比較し、結果にしたがってステータス・フラグをセットし、レジスタスタックをポップする。
DB E8+i	FUCOMI ST, ST(i)	ST(0) を ST(i) と比較し、順序付け値の有無をチェックし、結果にしたがってステータス・フラグをセットする。
DF E8+i	FUCOMIP ST, ST(i)	ST(0) を ST(i) と比較し、順序付け値の有無をチェックし、結果にしたがってステータス・フラグをセットし、レジスタスタックをポップする。

説明

レジスタ ST(0) と ST(i) の内容を順序付けなしで比較し、結果にしたがって EFLAGS レジスタ内のステータス・フラグ ZF、PF、および CF をセットする（以下の表を参照）。これらの命令の比較では、ゼロの符号は無視される。すなわち、–0.0 は +0.0 と等しい。

比較結果	ZF	PF	CF
ST0 > ST(i)	0	0	0
ST0 < ST(i)	0	0	1
ST0 = ST(i)	1	0	0
順序付けなし *	1	1	1

注：

* マスクされていない無効算術オペランド（#IA）例外が生成された場合、フラグはセットされない。

順序づけなし比較は、比較される数のクラスをチェックする（本章の「FXAM—Examine」を参照）。FUCOMI/FUCOMIP 命令は FCOMI/FCOMIP 命令と同じ操作を実行する。唯一の相違点は、FUCOMI/FUCOMIP 命令では、一方または両方のオペランドが SNaN であるか、サポートしていないフォーマットである場合にのみ、無効算術オペランド例外（#IA）が発生することである。オペランドが QNaN の場合は、コードフラグが順序づけなしに設定されるが、例外は発生しない。FCOMI/FCOMIP 命令では、一方または両方のオペランドが NaN 値であるか、サポートしていないフォーマットである場合は、無効操作例外が発生する。

操作の結果、無効算術オペランド例外が発生した場合は、その例外がマスクされている場合にのみ、EFLAGS レジスタ内のステータス・フラグがセットされる。

FCOMI/FCOMIP 命令と FUCOMI/FUCOMIP 命令は、（無効操作例外が検出されたかどうかに関係なく）EFLAGS レジスタ内の OF フラグをクリアする。

FCOMI/FCOMIP/ FUCOMI/FUCOMIP—Compare Floating Point Values and Set EFLAGS（続き）

FCOMIP および FUCOMIP 命令は、比較操作の後にさらにレジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。

IA-32 アーキテクチャにおける互換性

FCOMI/FCOMIP/FUCOMI/FUCOMIP 命令は、P6 ファミリ・プロセッサで IA-32 アーキテクチャに導入され、それより以前の IA-32 プロセッサには備えられていない。

操作

```
CASE (relation of operands) OF
    ST(0) > ST(i): ZF, PF, CF ← 000;
    ST(0) < ST(i): ZF, PF, CF ← 001;
    ST(0) = ST(i): ZF, PF, CF ← 100;
ESAC;
IF instruction is FCOMI or FCOMIP
    THEN
        IF ST(0) or ST(i) = NaN or unsupported format
            THEN
                #IA
                IF FPUControlWord.IM = 1
                    THEN
                        ZF, PF, CF ← 111;
                FI;
        FI;
FI;
IF instruction is FUCOMI or FUCOMIP
    THEN
        IF ST(0) or ST(i) = QNaN, but not SNaN or unsupported format
            THEN
                ZF, PF, CF ← 111;
            ELSE (* ST(0) or ST(i) is SNaN or unsupported format *)
                 #IA;
                IF FPUControlWord.IM = 1
                    THEN
                        ZF, PF, CF ← 111;
                FI;
        FI;
FI;
IF instruction is FCOMIP or FUCOMIP
    THEN
        PopRegisterStack;
FI;
```

FCOMI/FCOMIP/ FUCOMI/FUCOMIP—Compare Floating Point Values and Set EFLAGS（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にセットされる。
C0、C2、C3	影響を受けない。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	（FCOMI または FCOMIP 命令）一方または両方のオペランドが NaN 値であるか、またはそれらのフォーマットがサポートされていない場合。 （FUCOMI または FUCOMIP 命令）一方または両方のオペランドが SNaN 値である（ただし、QNaN ではない）か、またはそれらのフォーマットが定義されていない場合。QNaN 値が検出されても、無効オペランド例外は発生しない。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FCOS—Cosine

オペコード	命令	説明
D9 FF	FCOS	ST(0) をその余弦で置き換える。

説明

ST(0) レジスタ内のソース・オペランドの余弦を計算し、結果を ST(0) にストアする。ソース・オペランドはラジアン単位の± 2^{63} の範囲内の値でなければならない。以下の表に、さまざまなクラスの数の余弦を計算したときに得られる結果を示す。

ST(0) SRC	ST(0) DEST
–∞	*
–F	–1 ～ +1
–0	+1
+0	+1
+F	–1 ～ +1
+∞	*
NaN	NaN

注 :

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

ソース・オペランドが許容可能な範囲を超えた場合は、FPU ステータス・ワードの C2 フラグがセットされ、ST(0) レジスタの値は以前のまま変わらない。この命令は、ソース・オペランドが範囲外でも例外を発生しない。プログラムの責任で、C2 フラグを調べて範囲外条件の有無を確認しなければならない。± 2^{63} の範囲を超えるソース値は、2π の適切な整数倍を引くか、または除数を 2π として FPREM 命令を使用して、命令の許容範囲内に縮小することができる。そのような縮小のための減算に使用する適切な π の値については、『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「π」の説明を参照のこと。

操作

```
IF |ST(0)| < 2^63
THEN
    C2 ← 0;
    ST(0) ← cosine(ST(0));
ELSE (*source operand is out-of-range *)
    C2 ← 1;
FI;
```

FCOS—Cosine（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。 C2 が 1 の場合は未定義。
C2	範囲外（-2^{63} < ソース・オペランド < $+2^{63}$）の場合は 1 にセットされる。範囲内の場合は 0 にセットされる。
C0, C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値または∞であるか、そのフォーマットがサポートされていない場合。
#D	ソースがデノーマル値である場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FDECSTP—Decrement Stack-Top Pointer

オペコード	命令	説明
D9 F6	FDECSTP	FPU ステータス・ワードの TOP フィールドをデクリメントする。

説明

FPU ステータス・ワードの TOP フィールドから 1 を引く（すなわち、スタック・トップ・ポインタをデクリメントする）。TOP フィールドは、内容が 0 の場合は 7 に設定される。この命令の働きは、スタックを 1 レジスタ位置回転させることである。FPU データレジスタおよびタグレジスタの内容は影響を受けない。

操作

```
IF TOP = 0
    THEN TOP ← 7;
    ELSE TOP ← TOP – 1;
FI;
```

FPU 影響を受けるフラグ

C1　　0 にセットされる。

C0、C2、C3　　未定義。

浮動小数点例外

なし。

保護モード例外

#NM　　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　　CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM　　CR0 の EM または TS がセットされた場合。

FDIV/FDIVP/FIDIV—Divide

オペコード	命令	説明
D8 /6	FDIV *m32fp*	ST(0) を *m32fp* で割り、結果を ST(0) にストアする。
DC /6	FDIV *m64fp*	ST(0) を *m64fp* で割り、結果を ST(0) にストアする。
D8 F0+i	FDIV ST(0), ST(i)	ST(0) を ST(i) で割り、結果を ST(0) にストアする。
DC F8+i	FDIV ST(i), ST(0)	ST(i) を ST(0) で割り、結果を ST(i) にストアする。
DE F8+i	FDIVP ST(i), ST(0)	ST(i) を ST(0) で割り、結果を ST(i) にストアし、レジスタスタックをポップする。
DE F9	FDIVP	ST(1) を ST(0) で割り、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /6	FIDIV *m32int*	ST(0) を *m32int* で割り、結果を ST(0) にストアする。
DE /6	FIDIV *m16int*	ST(0) を *m64int* で割り、結果を ST(0) にストアする。

説明

デスティネーション・オペランドをソース・オペランドで割り、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランド（被除数）は常に FPU レジスタである。ソース・オペランド（除数）には、レジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

この命令のオペランドなしバージョンでは、ST(1) レジスタの内容を ST(0) レジスタの内容で割る。1 オペランド・バージョンでは、ST(0) レジスタの内容をメモリ・ロケーションの内容（浮動小数点値または整数値）で割る。2 オペランド・バージョンでは、ST(0) レジスタの内容を ST(i) レジスタの内容で割るか、またはその逆の除算を行う。

FDIVP 命令は、除算結果をストアした後に、追加操作として FPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。これらの浮動小数点除算命令のオペランドなしバージョンでは、常にレジスタスタックのポップを伴う。一部のアセンブラでは、この命令のニーモニックは FDIVP ではなく FDIV になっている。

FIDIV 命令は、整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから除算を行う。ソース・オペランドは、整数 0 のときは +0 として取り扱われる。

ゼロによる除算例外（#Z）が発生しても、それがマスクされていない場合は結果はストアされない。この例外がマスクされていた場合は、正しい符号の∞がデスティネーション・オペランドにストアされる。

FDIV/FDIVP/FIDIV—Divide（続き）

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の除算を行ったときに得られる結果を示す。

		DEST						
		−∞	−F	−0	+0	+F	+∞	NaN
SRC	−∞	*	+0	+0	−0	−0	*	NaN
	−F	+∞	+F	+0	−0	−F	−∞	NaN
	−I	+∞	+F	+0	−0	−F	−∞	NaN
	−0	+∞	**	*	*	**	−∞	NaN
	+0	−∞	**	*	*	**	+∞	NaN
	+I	−∞	−F	−0	+0	+F	+∞	NaN
	+F	−∞	−F	−0	+0	+F	+∞	NaN
	+∞	*	−0	−0	+0	+0	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

I 整数を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

** 浮動小数点 0 による除算（#Z）例外を示す。

操作

```
IF SRC = 0
    THEN
        #Z
    ELSE
        IF instruction is FIDIV
            THEN
                DEST ← DEST / ConvertToDoubleExtendedPrecisionFP(SRC);
            ELSE (* source operand is floating-point value *)
                DEST ← DEST / SRC;
        FI;
FI;
IF instruction = FDIVP
    THEN
        PopRegisterStack
FI;
```

FDIV/FDIVP/FIDIV—Divide（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。 ±∞ / ±∞、± 0/ ± 0 の場合。
#D	結果がデノーマル値である場合。
#Z	DEST/ ± 0 の場合。ただし、DEST は± 0 に等しくない。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FDIV/FDIVP/FIDIV—Divide（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FDIVR/FDIVRP/FIDIVR—Reverse Divide

オペコード	命令	説明
D8 /7	FDIVR *m32fp*	*m32fp* を ST(0) で割り、結果を ST(0) にストアする。
DC /7	FDIVR *m64fp*	*m64fp* を ST(0) で割り、結果を ST(0) にストアする。
D8 F8+i	FDIVR ST(0), ST(i)	ST(i) を ST(0) で割り、結果を ST(0) にストアする。
DC F0+i	FDIVR ST(i), ST(0)	ST(0) を ST(i) で割り、結果を ST(i) にストアする。
DE F0+i	FDIVRP ST(i), ST(0)	ST(0) を ST(i) で割り、結果を ST(i) にストアし、レジスタスタックをポップする。
DE F1	FDIVRP	ST(0) を ST(1) で割り、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /7	FIDIVR *m32int*	*m32int* を ST(0) で割り、結果を ST(0) にストアする。
DE /7	FIDIVR *m16int*	*m16int* を ST(0) で割り、結果を ST(0) にストアする。

説明

ソース・オペランドをデスティネーション・オペランドで割り、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランド（除数）は常に FPU レジスタである。ソース・オペランド（被除数）には、レジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

これらの命令は、それぞれ FDIV、FDIVP、FIDIV 命令の逆の演算を行う。これらは、コーディング効率の向上をサポートする目的で設けられたものである。

これらの命令のオペランドなしバージョンでは、ST(0) レジスタの内容を ST(1) レジスタの内容で割る。1 オペランド・バージョンでは、メモリ・ロケーションの内容（浮動小数点値または整数値）を ST(0) レジスタの内容で割る。2 オペランド・バージョンでは、ST(i) レジスタの内容を ST(0) レジスタの内容で割るか、またはその逆の除算を行う。

FDIVRP 命令は、除算結果をストアした後に、追加操作として FPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。これらの浮動小数点除算命令のオペランドなしバージョンでは、常にレジスタスタックのポップを伴う。一部のアセンブラでは、この命令のニーモニックは FDIVRP ではなく FDIVR になっている。

FIDIVR 命令は、整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから除算を行う。

FDIVR/FDIVRP/FIDIVR—Reverse Divide（続き）

ゼロによる除算例外（#Z）が発生しても、それがマスクされていない場合は結果はストアされない。この例外がマスクされていた場合は、正しい符号の∞がデスティネーション・オペランドにストアされる。

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の除算を行ったときに得られる結果を示す。

		DEST						
		−∞	−F	−0	+0	+F	+∞	NaN
SRC	−∞	*	+∞	+∞	−∞	−∞	*	NaN
	−F	+0	+F	**	**	−F	−0	NaN
	−I	+0	+F	**	**	−F	−0	NaN
	−0	+0	+0	*	*	−0	−0	NaN
	+0	−0	−0	*	*	+0	+0	NaN
	+I	−0	−F	**	**	+F	+0	NaN
	+F	−0	−F	**	**	+F	+0	NaN
	+∞	*	−∞	−∞	+∞	+∞	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

I 整数を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

** 浮動小数点 0 による除算（#Z）例外を示す。

ソース・オペランドは、整数 0 の場合は +0 として取り扱われる。

FDIVR/FDIVRP/FIDIVR—Reverse Divide（続き）

操作

```
IF DEST = 0
    THEN
        #Z
    ELSE
        IF instruction is FIDIVR
            THEN
                DEST ← ConvertToDoubleExtendedPrecisionFP(SRC) / DEST;
            ELSE (* source operand is floating-point value *)
                DEST ← SRC / DEST;
        FI;
FI;
IF instruction = FDIVRP
    THEN
        PopRegisterStack
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。 ±∞ / ±∞、± 0/ ± 0 の場合。
#D	結果がデノーマル値である場合。
#Z	SCR/ ± 0 の場合。ただし、SCR は± 0 に等しくない。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

FDIVR/FDIVRP/FIDIVR—Reverse Divide（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FFREE—Free Floating-Point Register

オペコード	命令	説明
DD C0+i	FFREE ST(i)	ST(i) のタグを空に設定する。

説明

ST(i) レジスタに関連する FPU タグレジスタ内のタグを空（11B）に設定する。ST(i) および FPU スタック・トップ・ポインタ（TOP）の内容は影響を受けない。

操作

TAG(i) ← 11B;

FPU 影響を受けるフラグ

C0, C1, C2, C3　　未定義。

浮動小数点例外

なし。

保護モード例外

#NM　　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　　CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM　　CR0 の EM または TS がセットされた場合。

FICOM/FICOMP—Compare Integer

オペコード	命令	説明
DE /2	FICOM *m16int*	ST(0) を *m16int* と比較する。
DA /2	FICOM *m32int*	ST(0) を *m32int* と比較する。
DE /3	FICOMP *m16int*	ST(0) を *m16int* と比較し、スタックレジスタをポップする。
DA /3	FICOMP *m32int*	ST(0) を *m32int* と比較し、スタックレジスタをポップする。

説明

ST(0) レジスタの値を整数ソース・オペランドと比較し、結果にしたがって FPU ステータス・ワードの条件コードフラグ C0、C2、および C3 をセットする（以下の表を参照）。整数値は、比較が行われる前に拡張倍精度浮動小数点フォーマットに変換される。

条件	C3	C2	C0
ST(0) > SRC	0	0	0
ST(0) < SRC	0	0	1
ST(0) = SRC	1	0	0
順序付けなし	1	1	1

これらの命令では、「順序付けなし比較」を実行する。順序付けなし比較は、さらに比較対象の 2 つの数値のクラスのチェックも行う（本章の「FXAM—Examine」を参照）。どちらかのオペランドが NaN であるか、またはそのフォーマットが定義されていない場合は、条件フラグが「順序付けなし」に設定される。

ゼロの符号は無視される。すなわち、–0.0 ← +0.0 である。

FICOMP 命令は、比較後にレジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。

FICOM/FICOMP—Compare Integer（続き）

操作

```
CASE (relation of operands) OF
    ST(0) > SRC: C3, C2, C0 ← 000;
    ST(0) < SRC: C3, C2, C0 ← 001;
    ST(0) = SRC: C3, C2, C0 ← 100;
    Unordered:   C3, C2, C0 ← 111;
ESAC;
IF instruction = FICOMP
    THEN
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にクリアされる。
C0、C2、C3	前ページの表を参照。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	一方または両方のオペランドが　NaN　値であるか、またはそれらのフォーマットがサポートされていない場合。
#D	一方または両方のオペランドがデノーマル値である場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FICOM/FICOMP—Compare Integer（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FILD—Load Integer

オペコード	命令	説明
DF /0	FILD *m16int*	*m16int* を FPU レジスタスタックにプッシュする。
DB /0	FILD *m32int*	*m32int* を FPU レジスタスタックにプッシュする。
DF /5	FILD *m64int*	*m64int* を FPU レジスタスタックにプッシュする。

説明

符号付き整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換し、変換結果の値を FPU レジスタスタックにプッシュする。ソース・オペランドには、ワード、ダブルワード、またはクワッドワード整数値を使用できる。ソース・オペランドは丸めエラーなしにロードされる。ソース・オペランドの符号が保持される。

操作

```
TOP ← TOP – 1;
ST(0) ← ConvertToDoubleExtendedPrecisionFP(SRC);
```

FPU 影響を受けるフラグ

C1	スタック・オーバーフローが発生した場合は 1 にセットされ、発生しなかった場合は 0 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・オーバーフローが発生した場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FILD—Load Integer（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FINCSTP—Increment Stack-Top Pointer

オペコード	命令	説明
D9 F7	FINCSTP	FPU ステータス・レジスタの TOP フィールドをインクリメントする。

説明

FPU ステータス・ワードの TOP フィールドに 1 を加える（すなわち、スタック・トップ・ポインタをインクリメントする）。TOP フィールドは、内容が 7 の場合は 0 に設定される。この命令の働きは、スタックを 1 レジスタ位置回転させることである。FPU データレジスタおよびタグレジスタの内容は影響を受けない。この操作はスタックをポップすることと等価ではない。それは、前のスタック・トップ・レジスタのタグが空にマークされないためである。

操作

```
IF TOP = 7
    THEN TOP ← 0;
    ELSE TOP ← TOP + 1;
FI;
```

FPU 影響を受けるフラグ

C1	0 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

なし。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FINIT/FNINIT—Initialize Floating-Point Unit

オペコード	命令	説明
9B DB E3	FINIT	未処理のマスクされていない浮動小数点例外の有無をチェックした後、FPU を初期化する。
DB E3	FNINIT*	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、FPU を初期化する。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

FPU 制御、ステータス、タグ、命令ポインタ、およびデータポインタの各レジスタをそれぞれのデフォルト状態に設定する。FPU 制御ワードは 037FH に設定される（最も近い整数に丸められる、全例外がマスクされる、64 ビット精度）。ステータス・ワードはクリアされる（全例外フラグはセットされず、TOP は 0 に設定される）。レジスタスタック内のデータレジスタは以前のまま変わらないが、すべて空（11B）としてタグ付けされる。命令およびデータのポインタはクリアされる。

FINIT 命令は未処理のマスクされていない浮動小数点例外の有無を調べ、処理してから、初期化を行う。FNINIT 命令はこのチェックを行わない。

FINIT 命令の場合、アセンブラは、2 つの命令を発行する（つまり、FWAIT 命令に続けて FNINIT 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサまたは Intel486™ プロセッサを MS-DOS* 互換性モードで動作させたときは、（通常の状況下で）実行される前に FNINIT 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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 の「非同期型命令のウインドウ内の x87 FPU 割り込み」の説明を参照のこと。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・プロセッサでは、FNINIT 命令にこの方法では割り込みをかけられない。

Intel387 数値演算プロセッサでは、FINIT/FNINIT 命令は命令およびデータの両ポインタをクリアしない。

この命令は、x87 FPU のみに影響する。XMM および MXCSR レジスタには影響を与えない。

FINIT/FNINIT—Initialize Floating-Point Unit（続き）

操作

```
FPUControlWord ← 037FH;
FPUStatusWord ← 0;
FPUTagWord ← FFFFH;
FPUDataPointer ← 0;
FPUInstructionPointer ← 0;
FPULastInstructionOpcode ← 0;
```

FPU 影響を受けるフラグ

C0, C1, C2, C3　　0 にセットされる。

浮動小数点例外

なし。

保護モード例外

#NM　　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　　CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM　　CR0 の EM または TS がセットされた場合。

FIST/FISTP—Store Integer

オペコード	命令	説明
DF /2	FIST *m16int*	ST(0) を *m16int* にストアする。
DB /2	FIST *m32int*	ST(0) を *m32int* にストアする。
DF /3	FISTP *m16int*	ST(0) を *m16int* にストアし、レジスタスタックをポップする。
DB /3	FISTP *m32int*	ST(0) を *m32int* にストアし、レジスタスタックをポップする。
DF /7	FISTP *m64int*	ST(0) を *m64int* にストアし、レジスタスタックをポップする。

説明

ST(0) レジスタの値を符号付き整数に変換し、変換結果の値をデスティネーション・オペランドにストアする。値はワード整数またはダブルワード整数のフォーマットでストアできる。デスティネーション・オペランドは、デスティネーション値の最初のバイトがストアされるアドレスを指定する。

FISTP 命令は、FIST 命令と同じ操作を行ってから、レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。FISTP 命令は、値をロング整数フォーマットでもストアできる。

以下の表に、さまざまなクラスの数を整数フォーマットでストアしたときに得られる結果を示す。

ST(0)	DEST
–∞ または DEST 形式には値が大きすぎる	*
F < –1	–I
–1 < F < –0	**
–0	0
+0	0
+0 < F < +1	**
F > +1	+I
+∞ または DEST 形式には値が大きすぎる	*
NaN	*

注：

F 有限浮動小数点値を示す。

I 整数を示す。

* 浮動小数点無効操作（#IA）例外を示す。

** 丸めモードにより、0 または± 1。

FIST/FISTP—Store Integer（続き）

ソース値は、非整数値の場合は、FPU 制御ワードの RC フィールドによって指定される丸めモードにしたがって整数値に丸められる。

変換された値が大きすぎてデスティネーション・フォーマットで表現できない場合や、ソース・オペランドが ∞、SNaN、QNAN、またはサポートしていないフォーマットである場合は、無効算術オペランド条件が発生する。無効操作例外がマスクされていない場合は、無効算術オペランド例外（#IA）が発生し、値はデスティネーション・オペランドにストアされない。無効操作例外がマスクされている場合は、整数の未定義値がメモリにストアされる。

操作

```
DEST ← Integer(ST(0));
IF instruction = FISTP
    THEN
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にクリアされる。 不正確結果例外（#P）が発生した場合は、丸めの方向を示す。0 ← 切り上げなし、1 ← 切り上げ。 発生しなかった場合は 0 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	変換された値が大きすぎて、デスティネーション・フォーマットで表現できない場合。 ソース・オペランドが SNaN、QNaN、±∞、またはそのフォーマットがサポートされていない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

FIST/FISTP—Store Integer（続き）

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FISTTP—Store Integer with Truncation

オペコード	命令	説明
DF /1	FISTTP *m16int*	ST を符号付き整数（切り捨て）として *m16int* にストアし、ST をポップする。
DB /1	FISTTP *m32int*	ST を符号付き整数（切り捨て）として *m32int* にストアし、ST をポップする。
DD /1	FISTTP *m64int*	ST を符号付き整数（切り捨て）として *m64int* にストアし、ST をポップする。

説明

FISTTP 命令は、丸めモードとして切り捨てを使用して ST レジスタ内の値を符号付き整数に変換し、変換された値をデスティネーションに転送し、ST をポップする。FISTTP 命令のデスティネーションは、ワード、short 型整数、または long 型整数である。

以下の表は、さまざまなクラスの数を整数フォーマットでストアしたときに得られる結果を示している。

ST(0)	DEST
–∞ または DEST 形式には値が大きすぎる	*
F ≥ –1	–I
–1 < F < +1	0
F ≥ +1	+I
+∞ または DEST 形式には値が大きすぎる	*
NaN	*

注：

F　有限浮動小数点値を示す。

I　整数を示す。

*　浮動小数点無効操作（#IA）例外を示す。

操作

```
DEST ← ST;
pop ST;
```

FPU 影響を受けるフラグ

C1　　クリアされる。

C0、C2、C3　　未定義。

FISTTP—Store Integer with Truncation（続き）

数値例外

無効、スタック無効（スタック・アンダーフロー）、精度。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントの場合。
	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。
#NM	CR0.EM が 1 の場合。
	CR0 の TS ビットがセットされている場合。
#UD	CPUID.SSE3（ECX ビット 0）が 0 の場合。

実アドレスモード例外

#GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0.EM が 1 の場合。
	CR0 の TS ビットがセットされている場合。
#UD	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

#GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0.EM が 1 の場合。
	CR0 の TS ビットがセットされている場合。
#UD	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメントの合っていないメモリ参照を行った場合。

FLD—Load Floating Point Value

オペコード	命令	説明
D9 /0	FLD *m32fp*	*m32fp* を FPU レジスタスタックにプッシュする。
DD /0	FLD *m64fp*	*m64fp* を FPU レジスタスタックにプッシュする。
DB /5	FLD *m80fp*	*m80fp* を FPU レジスタスタックにプッシュする。
D9 C0+i	FLD ST(i)	ST(i) を FPU レジスタスタックにプッシュする。

説明

ソース・オペランドを FPU レジスタスタックにプッシュする。ソース・オペランドは、単精度、倍精度、または拡張倍精度浮動小数点フォーマットが可能である。ソース・オペランドが、単精度浮動小数点または倍精度浮動小数点数のフォーマットの場合は、スタックにプッシュされる前に自動的に拡張倍精度浮動小数点フォーマットに変換される。

FLD 命令は選択された FPU レジスタ [ST(i)] の値もスタックにプッシュできる。この場合、ST(0) レジスタをプッシュすると、スタックトップとその下のスタックレジスタが共に以前のスタックトップ値になる。

操作

```
IF SRC is ST(i)
    THEN
        temp ← ST(i)
FI;
TOP ← TOP – 1;
IF SRC is memory-operand
    THEN
        ST(0) ← ConvertToDoubleExtendedPrecisionFP(SRC);
    ELSE (* SRC is ST(i) *)
        ST(0) ← temp;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・オーバーフローが発生した場合は 1 にセットされ、発生しなかった場合は 0 にセットされる。
C0、C2、C3	未定義。

FLD—Load Floating Point Value（続き）

浮動小数点例外

#IS	スタック・オーバーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値である場合。ソース・オペランドが拡張倍精度浮動小数点形式（FLD m80fp または FLD ST(i)）の場合は発生しない。
#D	ソース・オペランドがデノーマル値である場合。ソース・オペランドが拡張倍精度浮動小数点フォーマットの場合は発生しない。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ—Load Constant

オペコード	命令	説明
D9 E8	FLD1	+1.0 を FPU レジスタスタックにプッシュする。
D9 E9	FLDL2T	$\log_2 10$ を FPU レジスタスタックにプッシュする。
D9 EA	FLDL2E	$\log_2 e$ を FPU レジスタスタックにプッシュする。
D9 EB	FLDPI	π を FPU レジスタスタックにプッシュする。
D9 EC	FLDLG2	$\log_{10} 2$ を FPU レジスタスタックにプッシュする。
D9 ED	FLDLN2	$\log_e 2$ を FPU レジスタスタックにプッシュする。
D9 EE	FLDZ	+0.0 を FPU レジスタスタックにプッシュする。

説明

拡張倍精度浮動小数点フォーマットで 広く使用されている 7 つの定数の中の 1 つを FPU レジスタスタックにプッシュする。これらの命令でロードできる定数は、+1.0、+0.0、$\log_2 10$、$\log_2 e$、π、$\log_{10} 2$、$\log_e 2$ である。各定数について、66 ビットの内部定数が（FPU 制御ワードの RC フィールドの指定にしたがって）拡張倍精度浮動小数点フォーマットに丸められる。丸めの結果としては、不正確結果例外（#P）は発生しない。結果が切り上げられた場合は、x87 FPU ステータス・ワードで C1 フラグはセットされない。

π 定数については、『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「π」の説明を参照のこと。

操作

```
TOP ← TOP – 1;
ST(0) ← CONSTANT;
```

FPU 影響を受けるフラグ

C1	スタック・オーバーフローが発生した場合は 1 にセットされ、発生しなかった場合は 0 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・オーバーフローが発生した場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ—Load Constant（続き）

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

IA-32 アーキテクチャにおける互換性

RC フィールドが「最も近い整数に丸める」（round-to-nearest）に設定されているときは、FPU は、Intel 8087 および Intel287 数値演算プロセッサが生じるものと同じ定数を生じる。

FLDCW—Load x87 FPU Control Word

オペコード	命令	説明
D9 /5	FLDCW m2byte	m2byte から FPU 制御ワードをロードする。

説明

16 ビットのソース・オペランドを FPU 制御ワードにロードする。ソース・オペランドはメモリ・ロケーションである。この命令は、一般的に FPU の動作モードを設定または変更する場合に使用される。

新しい FPU 制御ワードをロードする前に、FPU ステータス・ワード内の 1 つ以上の例外フラグがセットされていて、新しい制御ワードがそれらの例外の 1 つ以上のマスクを解除した場合は、次の浮動小数点命令が実行されたときに浮動小数点例外が発生する（ただし、「非同期型」（no-wait）浮動小数点命令の場合を除く。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、上巻』の第 8 章の「ソフトウェア内での x87 FPU 例外の処理」の説明を参照）。FPU の動作モードを変更する際に例外を発生させないようにするには、新しい制御ワードをロードする前に、（FCLEX または FNCLEX 命令を使用して）すべての未処理例外をクリアする。

操作

```
FPUControlWord ← SRC;
```

FPU 影響を受けるフラグ

C0, C1, C2, C3　　未定義。

浮動小数点例外

なし。ただし、この操作で FPU ステータス・ワード内の未処理例外のマスクを解除されることがある。その場合は、次の「同期型」（waiting）浮動小数点命令が実行されたときにその例外が発生する。

FLDCW—Load x87 FPU Control Word（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FLDENV—Load x87 FPU Environment

オペコード	命令	説明
D9 /4	FLDENV *m14/28byte*	*m14byte* または *m28byte* から FPU 環境をロードする。

説明

メモリから x87 FPU レジスタに完全な FPU 動作環境をロードする。ソース・オペランドは、メモリ内の動作環境データの最初のバイトを指定する。このデータは、一般的に前もって FSTENV または FNSTENV 命令によって指定されたメモリ・ロケーションに書かれている。

FPU 動作環境は、FPU 制御ワード、ステータス・ワード、タグワード、命令ポインタ、データポインタ、および最後のオペコードからなっている。『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. から 8-12. に、プロセッサの動作モード（保護または実アドレス）および現在のオペランド・サイズ属性（16 ビットまたは 32 ビット）にしたがって、メモリにロードされる動作環境情報のレイアウトを示してある。仮想 8086 モードでは、実アドレスモードのレイアウトが使用される。

FLDENV 命令は、対応する FSTENV/FNSTENV 命令と同じ動作モードで実行する必要がある。

新しい FPU ステータス・ワードに 1 つ以上のマスクされていない例外フラグがセットされている場合は、次の浮動小数点命令が実行されたときに浮動小数点例外が発生する（ただし、「非同期型」（no-wait）浮動小数点命令の場合を除く。『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「ソフトウェア内での x87 FPU の処理」の説明を参照）。新しい環境をロードしたときに例外を発生させないようにするには、ロードされる FPU ステータス・ワード内のすべての例外フラグをクリアする。

この命令の実行中にページフォルトまたはリミットフォルトが発生した場合、フォルトハンドラが認識する x87 FPU レジスタの状態が、メモリからロードされている状態と一致しないことがある。このような場合、フォルトハンドラは、x87 FPU レジスタのステータスを無視して、フォルトを処理し、元の処理に戻る必要がある。その後、FLDENV 命令は x87 FPU レジスタのロードを完了し、コンテキストの不整合は発生しない。

FLDENV—Load x87 FPU Environment（続き）

操作

```
FPUControlWord ← SRC[FPUControlWord);
FPUStatusWord ← SRC[FPUStatusWord);
FPUTagWord ← SRC[FPUTagWord);
FPUDataPointer ← SRC[FPUDataPointer);
FPUInstructionPointer ← SRC[FPUInstructionPointer);
FPULastInstructionOpcode ← SRC[FPULastInstructionOpcode);
```

FPU 影響を受けるフラグ

C0、C1、C2、C3 フラグがロードされる。

浮動小数点例外

なし。ただし、この操作でマスクされていない例外がステータス・ワードにロードされた場合は、次の「同期型」（waiting）浮動小数点命令が実行されたときにその例外が発生する。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FLDENV—Load x87 FPU Environment（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FMUL/FMULP/FIMUL—Multiply

オペコード	命令	説明
D8 /1	FMUL *m32fp*	ST(0) に *m32fp* を掛け、結果を ST(0) にストアする。
DC /1	FMUL *m64fp*	ST(0) に *m64fp* を掛け、結果を ST(0) にストアする。
D8 C8+i	FMUL ST(0), ST(i)	ST(0) に ST(i) を掛け、結果を ST(0) にストアする。
DC C8+i	FMUL ST(i), ST(0)	ST(i) に ST(0) を掛け、結果を ST(i) にストアする。
DE C8+i	FMULP ST(i), ST(0)	ST(i) に ST(0) を掛け、結果を ST(i) にストアし、レジスタスタックをポップする。
DE C9	FMULP	ST(1) に ST(0) を掛け、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /1	FIMUL *m32int*	ST(0) に *m32int* を掛け、結果を ST(0) にストアする。
DE /1	FIMUL *m16int*	ST(0) に *m16int* を掛け、結果を ST(0) にストアする。

説明

デスティネーション・オペランドとソース・オペランドとを掛け合わせ、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランドは常にFPUデータレジスタである。ソース・オペランドには、FPUデータレジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

この命令のオペランドなしバージョンでは、ST(1) レジスタの内容にST(0) レジスタの内容を掛け、結果を ST(1) レジスタにストアする。1 オペランド・バージョンでは、ST(0) レジスタの内容にメモリ・ロケーション（浮動小数点値でも整数値でも可）を掛け、結果をST(0) にストアする。2 オペランド・バージョンでは、ST(0) レジスタの内容にST(i) レジスタの内容を掛けるか、またはその逆に掛け、結果を第1 オペランド（デスティネーション・オペランド）によって指定されるレジスタにストアする。

FMULP 命令は、結果をストアした後に、追加操作としてRPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。浮動小数点乗算命令のオペランドなしバージョンでは、常にレジスタスタックのポップ操作を伴う。一部のアセンブラでは、この命令のニーモニックはFMULP ではなく FMUL になっている。

FIMUL 命令は、整数ソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから乗算を行う。

結果の符号は、掛け合わされる一方または両方の値が 0 または∞であっても、常にソースとデスティネーションの符号の排他的論理和である。ソース・オペランドは、整数 0 の場合、+0 として取り扱われる。

FMUL/FMULP/FIMUL—Multiply（続き）

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の乗算を行ったときに得られる結果を示す。

		DEST						
		−∞	−F	−0	+0	+F	+∞	NaN
SRC	−∞	+∞	+∞	*	*	−∞	−∞	NaN
	−F	+∞	+F	+0	−0	−F	−∞	NaN
	−I	+∞	+F	+0	−0	−F	−∞	NaN
	−0	*	+0	+0	−0	−0	*	NaN
	+0	*	−0	−0	+0	+0	*	NaN
	+I	−∞	−F	−0	+0	+F	+∞	NaN
	+F	−∞	−F	−0	+0	+F	+∞	NaN
	+∞	−∞	−∞	*	*	+∞	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F　有限浮動小数点値を示す。

I　整数を示す。

*　無効算術オペランド（#IA）例外を示す。

操作

```
IF instruction is FIMUL
    THEN
        DEST ← DEST * ConvertToDoubleExtendedPrecisionFP(SRC);
    ELSE (* source operand is floating-point value *)
        DEST ← DEST * SRC;
FI;
IF instruction = FMULP
    THEN
        PopRegisterStack
FI;
```

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされる。

結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。

C0、C2、C3　未定義。

FMUL/FMULP/FIMUL—Multiply（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
	一方のオペランドが± 0 であり、他方のオペランドが±∞の場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FMUL/FMULP/FIMUL—Multiply（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FNOP—No Operation

オペコード	命令	説明
D9 D0	FNOP	何の操作も実行されない。

説明

何の FPU 操作も実行されない。この命令は命令ストリーム上の空間を占めるが、EIP レジスタを除いて、FPUにもマシン・コンテキストにも影響を与えない。

FPU 影響を受けるフラグ

C0, C1, C2, C3　　未定義。

浮動小数点例外

なし。

保護モード例外

#NM　　CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM　　CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM　　CR0 の EM または TS がセットされた場合。

FPATAN—Partial Arctangent

オペコード	命令	説明
D9 F3	FPATAN	ST(1) を arctan(ST(1)/ST(0)) で置き換え、レジスタスタックをポップする。

説明

ST(1) レジスタ内のソース・オペランドを ST(0) レジスタ内のソース・オペランドで割った値の逆正接を計算し、結果を ST(1) にストアし、FPU レジスタスタックをポップする。ST(0) レジスタ内の結果の符号は、ソース・オペランド ST(1) の符号と同じであり、結果の絶対値は+π より小さい。

FPATAN 命令は、X 軸と、原点と点（X,Y）を結ぶ直線とのなす角度を返す。ただし、Y（縦座標）は ST(1) であり、X（横座標）は ST(0) である。この角度は、単に Y/X の比の符号だけでなく、X と Y の符号に別々に依存する。それは、点（–X,Y）が第 2 象限にあり、したがって角度の範囲は π/2 から π までになるのに対し、点（X,–Y）は第 4 象限にあり、角度の範囲は 0 から –π/2 までになるためである。点（–X,–Y）は第 3 象限にあって、角度の範囲は –π/2 から –π までとなる。

以下の表に、アンダーフローが生じないものとして、さまざまなクラスの数の逆正接を計算したときに得られる結果を示す。

ST(1) \ ST(0)	–∞	–F	–0	+0	+F	+∞	NaN
–∞	–3π/4*	–π/2	–π/2	–π/2	–π/2	–π/4*	NaN
–F	–π	–π to –π/2	–π/2	–π/2	–π/2 to –0	–0	NaN
–0	–π	–π	–π*	–0*	–0	–0	NaN
+0	+π	+π	+π*	+0*	+0	+0	NaN
+F	+π	+π to +π/2	+π/2	+π/2	+π/2 to +0	+0	NaN
+∞	+3π/4*	+π/2	+π/2	+π/2	+π/2	+π/4*	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限実数を示す。

* 『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、上巻』の表 8-10. は、2 つの比 0/0 および∞ / ∞が浮動小数点無効算術演算例外を発生し、かつ、この例外がマスクされている場合は、浮動小数点 QNaN の未定義値が返されるものと規定している。FPATAN 命令では、0/0 も∞ / ∞も実際には除算を使って計算されない。その代わりに、引き数として複素数を使用できるように一般化された標準的な数学的定式化から、2 つの変数の間の逆正接が得られる。この複素変数の定式化では、arctangent（0,0）などの値は非常に高精度で確定する。引き数として浮動小数点値しか使用できない FPU 関数に基づいて、複素数の引き数を使用する超越関数を計算するには、それらの値のライブラリを作成する必要がある。

FPATAN—Partial Arctangent（続き）

FPATAN 命令が許容できるソース・オペランドの範囲に制約はない。

IA-32 アーキテクチャにおける互換性

80287 数値演算プロセッサでは、この命令の両方のソース・オペランドは下記の範囲に制限される。

$0 \leq |ST(1)| < |ST(0)| < +\infty$

操作

```
ST(1) ← arctan(ST(1) / ST(0));
PopRegisterStack;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FPREM—Partial Remainder

オペコード	命令	説明
D9 F8	FPREM	ST(0) を ST(1) で割って得られる剰余で ST(0) を置き換える。

説明

ST(0) レジスタの値（被除数）を ST(1) レジスタの値（除数または**法**）で割って得られる剰余を計算し、結果を ST(0) にストアする。剰余の値は以下の式で表される。

剰余 ← ST(0) – (Q * ST(1))

ここで、Q は [ST(0)/ST(1)] の浮動小数点数の商をゼロに向けて切り捨てて得られる整数値である。剰余の符号は被除数の符号と同じである。部分剰余が 1 回も計算されていなくても、剰余の絶対値は法の絶対値より小さい（下で説明）。

この命令は正確な結果を生じる。不正確な結果の例外は発生せず、丸め制御は効果をもたない。以下の表に、アンダーフローが発生しないものとして、さまざまなクラスの数の剰余を計算したときに得られる結果を示す。

		ST(1)						
		–∞	–F	–0	+0	+F	+∞	NaN
ST(0)	–∞	*	*	*	*	*	*	NaN
	–F	ST(0)	–F or –0	**	**	–F or –0	ST(0)	NaN
	–0	–0	–0	*	*	–0	–0	NaN
	+0	+0	+0	*	*	+0	+0	NaN
	+F	ST(0)	+F or +0	**	**	+F or +0	ST(0)	NaN
	+∞	*	*	*	*	*	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

** 浮動小数点 0 による除算（#Z）例外を示す。

結果が 0 のときは、その符号は被除数の符号と同じである。法が∞のときは、結果は ST(0) の値に等しくなる。

FPREM 命令は、IEEE 規格 754 で定義される剰余を計算しない。IEEE で定義される剰余を計算するには、FPREM1 命令を使用する必要がある。FPREM 命令は、インテル® 8087およびインテル® 287数値演算コプロセッサとの互換性のために用意されている。

FPREM—Partial Remainder（続き）

FPREM 命令の「部分剰余」（partial remainder）という名前は、その剰余の計算方法に由来している。この命令は減算を反復して最終的に剰余を得る。すなわち、この命令の 1 回の実行では ST(0) の指数を 63 までしか縮小できない。法より小さい剰余を生じることができた場合に、この演算は完了し、FPU ステータス・ワード内の C2 フラグがクリアされる。法より小さい剰余に達するまでは、C2 はセットされており、ST(0) 内の結果は**部分剰余**と呼ばれる。部分剰余の指数は元の被除数の指数よりも最低 32 は小さくなっている。ソフトウェアは、C2 がクリアされるまで、（ST(0) 内の部分剰余を被除数として使用して）この命令を繰り返し実行できる。（そのような剰余計算ループを実行している間に、FPU を必要とする、優先順位が高い割り込みルーチンが剰余計算ループ内の命令間でコンテキスト・スイッチを強制することが考えられるので注意する。）

FPREM 命令の重要な用途は、周期関数の引き数を縮小することである。縮小が完了すると、この命令は商の最下位 3 ビットを FPU ステータス・ワードの C3、C1、および C0 フラグにストアする。この情報は、単位円の正しい 8 等分（π/4）扇形内の元の角度を示してくれるので、（法として π/4 を使用した）正接関数の引き数の縮小に重要である。

操作

```
D ← exponent(ST(0)) – exponent(ST(1));
IF D < 64
    THEN
        Q ← Integer(TruncateTowardZero(ST(0) / ST(1)));
        ST(0) ← ST(0) – (ST(1) * Q);
        C2 ← 0;
        C0, C3, C1 ← LeastSignificantBits(Q); (* Q2, Q1, Q0 *)
    ELSE
        C2 ← 1;
        N ← an implementation-dependent number between 32 and 63;
        QQ ← Integer(TruncateTowardZero((ST(0)  / ST(1)) / 2^(D – N)));
        ST(0) ← ST(0) – (ST(1) * QQ * 2^(D – N));
FI;
```

FPU 影響を受けるフラグ

C0	商のビット 2（Q2）にセットされる。
C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は商の最下位ビット（Q0）にセットされる。
C2	縮小が完了した場合は 0 にセットされ、完了していない場合は 1 にセットされる。
C3	商のビット 1（Q1）にセットされる。

FPREM—Partial Remainder（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、法が 0 であるか、被除数が∞であるか、またはそれらのいずれかのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FPREM1—Partial Remainder

オペコード	命令	説明
D9 F5	FPREM1	ST(0) を ST(1) で割って得られる IEEE 剰余で ST(0) を置き換える。

説明

ST(0) レジスタの値（被除数）を ST(1) レジスタの値（除数または**法**）で割って得られる IEEE 剰余を計算し、結果を ST(0) にストアする。剰余の値は以下の式で表される。

剰余 ← ST(0) – (Q * ST(1))

ここで、Q は [ST(0)/ST(1)] の浮動小数点数の商を最も近い整数値に向けて丸めて得られる整数値である。部分剰余が 1 回も計算されていなくても、剰余の絶対値は法の絶対値の 1/2 より小さいか、または等しい（下で説明）。

この命令は正確な結果を生じる。精度（不正確）例外は発生せず、丸め制御は効果をもたない。以下の表に、アンダーフローが発生しないものとして、さまざまなクラスの数の剰余を計算したときに得られる結果を示す。

		ST(1)						
		–∞	–F	–0	+0	+F	+∞	NaN
ST(0)	–∞	*	*	*	*	*	*	NaN
	–F	ST(0)	±F or –0	**	**	±F or –0	ST(0)	NaN
	–0	–0	–0	*	*	–0	–0	NaN
	+0	+0	+0	*	*	+0	+0	NaN
	+F	ST(0)	±F or +0	**	**	±F or +0	ST(0)	NaN
	+∞	*	*	*	*	*	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注 :

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

** 浮動小数点 0 による除算（#Z）例外を示す。

結果が 0 のときは、その符号は被除数の符号と同じである。法が∞のときは、結果は ST(0) の値に等しくなる。

FPREM 命令は、IEEE 規格 754 に規定されている剰余を計算する。この命令の操作は、ST(0) を ST(1) で割った商を整数に丸める点で FPREM 命令とは異なる（以下の「操作」の項を参照）。

FPREM1—Partial Remainder（続き）

FPREM1 命令は、FPREM 命令と同様に、減算を反復して最終的に剰余を計算する。すなわち、この命令の 1 回の実行では ST(0) の指数を 63 までしか縮小できない。法より小さい剰余を生じることができた場合に、この演算は完了し、FPU ステータス・ワード内の C2 フラグがクリアされる。法より小さい剰余に達するまでは、C2 はセットされており、ST(0) 内の結果は**部分剰余**と呼ばれる。部分剰余の指数は元の被除数の指数よりも最低 32 は小さくなっている。ソフトウェアは、C2 がクリアされるまで、（ST(0) 内の部分剰余を被除数として使用して）この命令を繰り返し実行できる。（そのような剰余計算ループを実行している間に、FPU を必要とする、優先順位が高い割り込みルーチンが剰余計算ループ内の命令間でコンテキスト・スイッチを強制する可能性があるので注意する。）

FPREM 命令の重要な用途は、周期関数の引き数を縮小することである。縮小が完了すると、この命令は商の最下位 3 ビットを FPU ステータス・ワードの C3、C1、および C0 フラグにストアする。この情報は、単位円の正しい 8 等分（π/4）扇形内の元の角度を示してくれるので、（法として π/4 を使用した）正接関数の引き数の縮小に重要である。

操作

```
D ← exponent(ST(0)) – exponent(ST(1));
IF D < 64
    THEN
        Q ← Integer(RoundTowardNearestInteger(ST(0) / ST(1)));
        ST(0) ← ST(0) – (ST(1) * Q);
        C2 ← 0;
        C0, C3, C1 ← LeastSignificantBits(Q); (* Q2, Q1, Q0 *)
    ELSE
        C2 ← 1;
        N ← an implementation-dependent number between 32 and 63;
        QQ ← Integer(TruncateTowardZero((ST(0)  / ST(1)) / 2^(D – N)));
        ST(0) ← ST(0) – (ST(1) * QQ * 2^(D – N));
FI;
```

FPU 影響を受けるフラグ

C0	商のビット 2（Q2）にセットされる。
C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は商の最下位ビット（Q0）にセットされる。
C2	縮小が完了した場合は 0 にセットされ、完了していない場合は 1 にセットされる。
C3	商のビット 1（Q1）にセットされる。

FPREM1—Partial Remainder（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、法が 0 であるか、被除数が∞であるか、またはそれらのいずれかのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FPTAN—Partial Tangent

オペコード	命令	説明
D9 F2	FPTAN	ST(0) をその正接で置き換え、FPU スタックに 1 をプッシュする。

説明

ST(0) レジスタ内のソース・オペランドの正接を計算し、結果を ST(0) にストアし、FPU レジスタスタックに 1.0 をプッシュする。ソース・オペランドはラジアン単位であり、± 2^{63} の範囲内でなければならない。以下の表に、アンダーフローが発生しないものとして、さまざまなクラスの数の部分正接を計算したときに得られるマスクされていない結果を示す。

ST(0) SRC	ST(0) DEST
–∞	*
–F	–F ～ +F
–0	–0
+0	+0
+F	–F ～ +F
+∞	*
NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

ソース・オペランドが許容可能な範囲外の場合は、FPU ステータス・ワード内の C2 フラグがセットされ、ST(0) レジスタの値は変わらない。この命令は、ソース・オペランドが範囲外のときに例外を発生しない。C2 フラグを調べて範囲外条件の有無を確認するのはプログラムの責任である。ソース値が ± 2^{63} の範囲外の場合は、除数を 2π として FPREM 命令を使用して、2π の該当する整数倍を減算することにより、ソース値を命令の範囲内に縮小することができる。上記のような縮小演算で π に使用する適切な値については、『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「π」の説明を参照のこと。

インテル® 8087 およびインテル® 287 数値演算プロセッサとの互換性を保つため、正接の計算終了後に、レジスタスタックに 1.0 がプッシュされる。この操作によって、さらに他の三角関数の計算も単純化される。例えば、FPTAN 命令の後で FDIVR 命令を実行して余接（正接の逆数）を計算できる。

FPTAN—Partial Tangent（続き）

操作

```
IF ST(0) < 2^63
THEN
    C2 ← 0;
    ST(0) ← tan(ST(0));
    TOP ← TOP – 1;
    ST(0) ← 1.0;
ELSE (*source operand is out-of-range *)
    C2 ← 1;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、スタック・オーバーフローが発生した場合は 1 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C2	範囲外（-2^{63} < ソース・オペランド < $+2^{63}$）の場合は 1 にセットされ、範囲内の場合は 0 にセットされる。
C0, C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、∞であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FRNDINT—Round to Integer

オペコード	命令	説明
D9 FC	FRNDINT	ST(0) を整数に丸める。

説明

ST(0) レジスタのソース値を、現在の丸めモード（FPU 制御ワードの RC フィールドの設定）にしたがって最も近い整数値に丸め、結果を ST(0) にストアする。

ソース値は、∞の場合は変更されない。ソース値が整数値でない場合は、浮動小数点不正確結果例外（#P）が発生する。

操作

```
ST(0) ← RoundToIntegralValue(ST(0));
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#P	ソース・オペランドが整数値でない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FRSTOR—Restore x87 FPU State

オペコード	命令	説明
DD /4	FRSTOR *m94/108byte*	*m94byte* または *m108byte* から FPU 状態をロードする。

説明

ソース・オペランドで指定されるメモリ領域から、FPU状態（動作環境とレジスタスタック）をロードする。この状態データは、一般的に前もってFSAVE/FNSAVE命令によって指定されたメモリ・ロケーションに書かれている。

FPU動作環境は、FPU制御ワード、ステータス・ワード、タグワード、命令ポインタ、データポインタ、最後のオペコードからなっている。『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.から8-12.に、プロセッサの動作モード（保護または実アドレス）および現在のオペランド・サイズ属性（16 ビットまたは32 ビット）にしたがって、メモリにストアされる動作環境情報のレイアウトを示してある。仮想8086モードでは、実アドレスモードのレイアウトが使用される。FPUレジスタスタックの内容は、動作環境イメージの直後の80バイト領域にストアされる。

FRSTOR命令は、対応するFSAVE/FNSAVE命令と同じ動作モードで実行する必要がある。

新しいFPUステータス・ワード内でマスクされていない例外ビットが1つ以上セットされている場合は、浮動小数点例外が生成される。新しいオペレーティング環境をロードするときに例外が発生しないようにするには、ロードされるFPUステータス・ワード内の例外フラグをすべてクリアする。

操作

```
FPUControlWord ← SRC[FPUControlWord);
FPUStatusWord ← SRC[FPUStatusWord);
FPUTagWord ← SRC[FPUTagWord);
FPUDataPointer ← SRC[FPUDataPointer);
FPUInstructionPointer ← SRC[FPUInstructionPointer);
FPULastInstructionOpcode ← SRC[FPULastInstructionOpcode);
ST(0) ← SRC[ST(0));
ST(1) ← SRC[ST(1));
ST(2) ← SRC[ST(2));
ST(3) ← SRC[ST(3));
ST(4) ← SRC[ST(4));
ST(5) ← SRC[ST(5));
ST(6) ← SRC[ST(6));
ST(7) ← SRC[ST(7));
```

FRSTOR—Restore x87 FPU State（続き）

FPU 影響を受けるフラグ

C0、C1、C2、C3 フラグがロードされる。

浮動小数点例外

なし。ただし、検出されたが、マスクされていたために発生しなかった既存の例外が、この操作によってマスクを解除されることがある。そのような例外は、ここで、FRSTOR 命令の終了後に発生する。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSAVE/FNSAVE—Store x87 FPU State

オペコード	命令	説明
9B DD /6	FSAVE *m94/108byte*	未処理のマスクされていない浮動小数点例外の有無をチェックした後、FPU 状態を *m94byte* または *m108byte* にストアし、次に FPU を初期化する。
DD /6	FNSAVE* *m94/108byte*	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、FPU状態を *m94byte* または *m108byte* にストアし、次に FPU を初期化する。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

現在の FPU 状態（動作環境とレジスタスタック）を指定されたメモリ内のデスティネーションにストアし、次に FPU を再初期化する。FSAVE 命令は、未処理のマスクされていない浮動小数点例外の有無をチェックし、処理してから、FPU 状態をストアする。FNSAVE 命令はこのチェックと処理を行わない。

FPU 動作環境は、FPU 制御ワード、ステータス・ワード、タグワード、命令ポインタ、データポインタ、最後のオペコードからなっている。『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. から 8-12. に、プロセッサ の動作モード（保護または実アドレス）および現在のオペランド・サイズ属性（16 ビットまたは 32 ビット）にしたがって、メモリにストアされる動作環境情報のレイアウトを示してある。仮想 8086 モードでは、実アドレスモードのレイアウトが使用される。FPU レジスタスタックの内容は、動作環境イメージの直後の 80 バイト領域にストアされる。

セーブされたイメージは、命令ストリーム内の FSAVE/FNSAVE 命令より前のすべての浮動小数点命令が実行された後の FPU の状態を反映している。

FPU 状態がセーブされた後、FPU は FINIT/FNINIT 命令で設定されるのと同じデフォルト値にリセットされる（本章の「FINIT/FNINIT—Initialize Floating-Point Unit」を参照）。

FSAVE/FNSAVE 命令は、一般的にオペレーティング・システムがコンテキスト・スイッチを実行する必要があるとき、例外ハンドラが FPU を使用する必要があるとき、またはアプリケーション・プログラムが「クリーン」な（初期状態の）FPU をプロシージャに渡す必要があるときに使用される。

FSAVE 命令の場合、アセンブラは、2 つの命令を発行する（つまり、FWAIT 命令に続けて FNSAVE 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

FSAVE/FNSAVE—Store x87 FPU State（続き）

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサより以前のインテル数値演算プロセッサおよびFPUの場合は、前の FSAVE/ FNSAVE 命令でストアされたメモリイメージから読み取る前に、FWAIT 命令を実行されたい。この FWAIT 命令によって、より確実にストア操作を完了することができる。

インテルPentiumプロセッサまたはIntel486™ プロセッサをMS-DOS* 互換性モードで動作させたときは、（通常の状況下で）実行される前に FNSAVE 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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 の「非同期型命令のウィンドウ内の x87 FPU 割り込み」の説明を参照のこと。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・プロセッサでは、FNSAVE 命令にこの方法では割り込みをかけられない。

操作

```
(* Save FPU State and Registers *)
DEST[FPUControlWord) ← FPUControlWord;
DEST[FPUStatusWord) ← FPUStatusWord;
DEST[FPUTagWord) ← FPUTagWord;
DEST[FPUDataPointer) ← FPUDataPointer;
DEST[FPUInstructionPointer) ← FPUInstructionPointer;
DEST[FPULastInstructionOpcode) ← FPULastInstructionOpcode;
DEST[ST(0)) ← ST(0);
DEST[ST(1)) ← ST(1);
DEST[ST(2)) ← ST(2);
DEST[ST(3)) ← ST(3);
DEST[ST(4)) ← ST(4);
DEST[ST(5)) ← ST(5);
DEST[ST(6)) ← ST(6);
DEST[ST(7)) ← ST(7);
(* Initialize FPU *)
FPUControlWord ← 037FH;
FPUStatusWord ← 0;
FPUTagWord ← FFFFH;
FPUDataPointer ← 0;
FPUInstructionPointer ← 0;
FPULastInstructionOpcode ← 0;
```

FPU 影響を受けるフラグ

C0、C1、C2、C3 フラグがセーブされ、次にクリアされる。

FSAVE/FNSAVE—Store x87 FPU State（続き）

浮動小数点例外

なし。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSCALE—Scale

オペコード	命令	説明
D9 FD	FSCALE	ST(0) を ST(1) でスケーリングする。

説明

ソース・オペランドの値を（0 に向かって）切り捨てた、元の値に最も近い整数値に変換し、変換された値をデスティネーション・オペランドの指数に加算する。デスティネーション・オペランドは、レジスタ ST(0) 内の浮動小数点値である。ソース・オペランドは、レジスタ ST(1) 内の浮動小数点値である。この命令によって、2 の整数乗による高速の乗算または除算が可能になる。以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数をスケーリングしたときに得られる結果を示す。

		ST(1)						
		−∞	−F	−0	+0	+F	+∞	NaN
ST(0)	−∞	NaN	−∞	−∞	−∞	−∞	−∞	NaN
	−F	−0	−F	−F	−F	−F	−∞	NaN
	−0	−0	−0	−0	−0	−0	NaN	NaN
	+0	+0	+0	+0	+0	+0	NaN	NaN
	+F	+0	+F	+F	+F	+F	+∞	NaN
	+∞	NaN	+∞	+∞	+∞	+∞	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

ほとんどの場合、指数だけが変更されて仮数は変更されない。ただし、ST(0) 内のスケーリングされる値がデノーマル値のときは、仮数も変更され、結果は正規化数になることがある。同様に、スケーリング操作の結果オーバーフローまたはアンダーフローが発生した場合、結果の仮数はソースの仮数とは異なることになる。

SCALE 命令を使用して、以下の例に示すように FXTRACT 命令の処理を逆にすることもできる。

```
FXTRACT;
FSCALE;
FSTP ST(1);
```

FSCALE—Scale（続き）

この例では、FXTRACT 命令が ST(0) の値から仮数と指数を抽出し、それぞれ ST(0) と ST(1) にストアしている。次に、FSCALE 命令が ST(0) 内の仮数を ST(1) 内の指数でスケーリングし、FXTRACT 操作が行われる前の元の値を作成し直している。FSTP ST(1) 命令は作成し直された値で（FXTRACT 命令によって抽出された）指数を上書きする。その結果、スタックはレジスタ [ST(0)] が 1 つだけ占有された元の状態に戻る。

操作

$ST(0) \leftarrow ST(0) * 2^{RoundTowardZero(ST(1))};$

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FSIN—Sine

オペコード	命令	説明
D9 FE	FSIN	ST(0) をその正弦で置き換える。

説明

ST(0) レジスタ内のソース・オペランドの正弦を計算し、結果を ST(0) レジスタにストアする。ソース・オペランドはラジアン単位であり、± 2^{63} の範囲内でなければならない。以下の表に、アンダーフローが発生しないものとして、さまざまなクラスの数の正弦を計算したときに得られる結果を示す。

SRC (ST(0))	DEST (ST(0))
−∞	*
−F	−1 ～ +1
−0	−0
+0	+0
+F	−1 ～ +1
+∞	*
NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

ソース・オペランドが許容可能な範囲外の場合は、FPU ステータス・ワード内の C2 フラグがセットされ、ST(0) レジスタの値は変わらない。この命令は、ソース・オペランドが範囲外のときに例外を発生させない。C2 フラグを調べて範囲外条件の有無を確認するのはプログラムの責任である。ソース値が± 2^{63} の範囲外の場合は、2π の該当する整数倍を引くか、または除数を 2π として FPREM 命令を使用することにより、ソース値を命令の許容範囲内に縮小することができる。上記のような縮小を行う際の π に使用する適切な値については、『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「π」の説明を参照のこと。

操作

```
IF ST(0) < 2^63
THEN
    C2 ← 0;
    ST(0) ← sin(ST(0));
ELSE (* source operand out of range *)
    C2 ← 1;
FI:
```

FSIN—Sine（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C2	範囲外（-2^{63} < ソース・オペランド < $+2^{63}$）の場合は 1 にセットされ、範囲内の場合は 0 にセットされる。
C0, C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、∞であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FSINCOS—Sine and Cosine

オペコード	命令	説明
D9 FB	FSINCOS	ST(0) の正弦と余弦を計算し、ST(0) を正弦で置き換え、余弦をレジスタスタックにプッシュする。

説明

ST(0) レジスタ内のソース・オペランドの正弦と余弦を計算し、正弦を ST(0) に、余弦を FPU レジスタスタックのトップにプッシュする。（この命令は、FSIN および FCOS 命令を続けて実行するより高速である。）

ソース・オペランドはラジアン単位であり、± 2^{63} の範囲内でなければならない。以下の表に、アンダーフローが発生しないものとして、さまざまなクラスの数の正弦と余弦を計算したときに得られる結果を示す。

SRC	DEST	
ST(0)	ST(1) Cosine	ST(0) Sine
–∞	*	*
–F	–1 ～ +1	–1 ～ +1
–0	+1	–0
+0	+1	+0
+F	–1 ～ +1	–1 ～ +1
+∞	*	*
NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

ソース・オペランドが許容可能な範囲外の場合は、FPU ステータス・ワード内の C2 フラグがセットされ、ST(0) レジスタの値は変わらない。この命令は、ソース・オペランドが範囲外のときに例外を発生させない。C2 フラグを調べて範囲外条件の有無を確認するのはプログラムの責任である。ソース値が± 2^{63} の範囲外の場合は、2π の該当する整数倍を引くか、または除数を 2π として FPREM 命令を使用することにより、ソース値を命令の許容範囲内に縮小することができる。上記のような縮小を行う際の π に使用する適切な値については、『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「π」の説明を参照のこと。

FSINCOS—Sine and Cosine（続き）

操作

```
IF ST(0) < 2^63
THEN
    C2 ← 0;
    TEMP ← cosine(ST(0));
    ST(0) ← sine(ST(0));
    TOP ← TOP – 1;
    ST(0) ← TEMP;
ELSE (* source operand out of range *)
    C2 ← 1;
FI:
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、スタック・オーバーフローが発生した場合は 1 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C2	範囲外（-2^{63} < ソース・オペランド < $+2^{63}$）の場合は 1 にセットされ、範囲内の場合は 0 にセットされる。
C0, C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、∞であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FSQRT—Square Root

オペコード	命令	説明
D9 FA	FSQRT	ST(0) の平方根を計算し、結果を STO(0) にストアする。

説明

ST(0) レジスタのソース値の平方根を計算し、結果を ST(0) にストアする。

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の平方根を計算したときに得られる結果を示す。

SRC (ST(0))	DEST (ST(0))
−∞	*
−F	*
−0	−0
+0	+0
+F	+F
+∞	+∞
NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効算術オペランド（#IA）例外を示す。

操作

ST(0) ← SquareRoot(ST(0));

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされる。

結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。

C0、C2、C3　未定義。

FSQRT—Square Root（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
	ソース・オペランドが負の値（ただし、–0 は除く）である場合。
#D	ソース・オペランドがデノーマル値である場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FST/FSTP—Store Floating Point Value

オペコード	命令	説明
D9 /2	FST *m32fp*	ST(0) を *m32fp* にコピーする。
DD /2	FST *m64fp*	ST(0) を *m64fp* にコピーする。
DD D0+i	FST ST(i)	ST(0) を ST(i) にコピーする。
D9 /3	FSTP *m32fp*	ST(0) を *m32fp* にコピーし、レジスタスタックをポップする。
DD /3	FSTP *m64fp*	ST(0) を *m64fp* にコピーし、レジスタスタックをポップする。
DB /7	FSTP *m80fp*	ST(0) を *m80fp* にコピーし、レジスタスタックをポップする。
DD D8+i	FSTP ST(i)	ST(0) を ST(i) にコピーし、レジスタスタックをポップする。

説明

この命令は、ST(0) レジスタの値をデスティネーション・オペランドにコピーする。デスティネーション・オペランドには、メモリ・ロケーションまたは FPU レジスタスタック内の別のレジスタを使用できる。値をメモリにストアするときは、値は単精度浮動小数点または倍精度浮動小数点のフォーマットに変換される。

FSTP 命令は、FST 命令と同じ操作を実行した後に、レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。FSTP 命令は、値をメモリに拡張倍精度浮動小数点フォーマットでもストアできる。

デスティネーション・オペランドがメモリ・ロケーションの場合、このオペランドはデスティネーション値の最初のバイトがストアされるアドレスを指定する。デスティネーション・オペランドがレジスタの場合、このオペランドはレジスタスタック内のスタックのトップから相対的にレジスタを指定する。

デスティネーションのサイズが単精度浮動小数点または倍精度浮動小数点の場合、ストアされる値の仮数は（FPU 制御ワードの RC フィールドによって指定された丸めモードにしたがって）デスティネーションの幅に丸められ、指数はデスティネーション・フォーマットの幅とバイアスに変換される。ストアされる値が大きすぎてデスティネーション・フォーマットで表現できない場合は、数値オーバーフロー例外（#0）が発生し、その例外がマスクされていない場合はデスティネーション・オペランドには何の値もストアされない。ストアされる値がデノーマル値であっても、デノーマル例外（#D）は発生しない。この条件が単に数値アンダーフロー例外（#U）条件として報告されるだけである。

ストアされる値が ± 0、±∞、または NaN の場合は、仮数および指数の最下位ビットがデスティネーション・フォーマットに合わせて切り捨てられる。この操作では、値のアイデンティティが 0、∞、または NaN として保存される。

FST/FSTP—Store Floating Point Value（続き）

デスティネーション・オペランドが空でないレジスタの場合は、無効操作例外は発生しない。

操作

```
DEST ← ST(0);
IF instruction ← FSTP
    THEN
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。
	不正確結果例外（#P）が発生した場合は、丸めの方向を示す。0 ← 切り上げなし、1 ← 切り上げ。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。ソース・オペランドが拡張倍精度浮動小数点フォーマットの場合は発生しない。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FST/FSTP—Store Floating Point Value（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSTCW/FNSTCW—Store x87 FPU Control Word

オペコード	命令	説明
9B D9 /7	FSTCW *m2byte*	未処理のマスクされていない浮動小数点定例外の有無をチェックした後、FPU 制御ワードを *m2byte* にストアする。
D9 /7	FNSTCW* *m2byte*	未処理のマスクされていない浮動小数点定例外の有無をチェックしないで、FPU 制御ワードを *m2byte* にストアする。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

FPU 制御ワードの現在値をメモリ内の指定されたデスティネーションにストアする。FSTCW 命令は、未処理のマスクされていない浮動小数点例外の有無をチェックし、例外を処理してから、制御ワードをストアする。FNSTCW 命令はこのチェックを行わない。

FSTCW 命令の場合、アセンブラは、2 つの命令を発行する（つまり、FWAIT 命令に続けて FNSTCW 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサまたは Intel486™ プロセッサを MS-DOS* 互換モードで動作させたときは、（通常の状況下で）実行される前に FNSTCW 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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 の「非同期型命令のウィンドウ内の x87 FPU 割り込み」の説明を参照のこと。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・プロセッサでは、FNSTCW 命令にこの方法では割り込みをかけられない。

操作

```
DEST ← FPUControlWord;
```

FPU 影響を受けるフラグ

C0、C1、C2、および C3 フラグは未定義。

浮動小数点例外

なし。

FSTCW/FNSTCW—Store x87 FPU Control Word（続き）

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSTENV/FNSTENV—Store x87 FPU Environment

オペコード	命令	説明
9B D9 /6	FSTENV *m14/28byte*	未処理のマスクされていない浮動小数点例外の有無をチェックした後、FPU 環境を *m14byte* または *m28byte* にストアする。次に、すべての浮動小数点例外をマスクする。
D9 /6	FNSTENV* *m14/28byte*	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、FPU 環境を *m14byte* または *m28byte* にストアする。次に、すべての浮動小数点例外をマスクする。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

現在の FPU 動作環境をデスティネーション・オペランドで指定されたメモリ・ロケーションにセーブし、次にすべての浮動小数点例外をマスクする。FPU 動作環境は、FPU 制御ワード、ステータス・ワード、タグワード、命令ポインタ、データポインタ、最後のオペコードからなっている。『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. から 8-12. に、プロセッサの動作モード（保護または実アドレス）および現在のオペランド・サイズ属性（16 ビットまたは 32 ビット）に応じて、メモリにストアされる動作環境のレイアウトが示してある。仮想 8086 モードでは、実アドレスモードのレイアウトが使用される。

FSTENV 命令は、未処理のマスクされていない浮動小数点例外の有無をチェックし、例外を処理してから、FPU 環境をストアする。FNSTENV 命令はこのチェックを行わない。セーブされたイメージは、命令ストリーム内の FSTENV/FNSTENV 命令より前のすべての浮動小数点命令が実行された後の FPU の状態を反映している。

これらの命令は、FPU 命令およびデータ両ポインタへのアクセスを可能にするので、例外ハンドラに使用されることが多い。環境は一般的にスタックにセーブされる。環境をセーブした後にすべての例外をマスクすると、浮動小数点例外による例外ハンドラへの割り込みがかけられなくなる。

FSTENV 命令の場合、アセンブラは、2 つの命令を発行する（つまり、FWAIT 命令に続けて FNSTENV 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

FSTENV/FNSTENV—Store x87 FPU Environment（続き）

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサまたは Intel486™ プロセッサを MS-DOS* 互換モードで動作させたときは、（通常の状況下で）実行される前に FNSTENV 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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 の「非同期型命令のウィンドウ内の x87 FPU 割り込み」の説明を参照のこと。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・プロセッサでは、FNSTENV 命令にこの方法では割り込みをかけられない。

操作

```
DEST[FPUControlWord) ← FPUControlWord;
DEST[FPUStatusWord) ← FPUStatusWord;
DEST[FPUTagWord) ← FPUTagWord;
DEST[FPUDataPointer) ← FPUDataPointer;
DEST[FPUInstructionPointer) ← FPUInstructionPointer;
DEST[FPULastInstructionOpcode) ← FPULastInstructionOpcode;
```

FPU 影響を受けるフラグ

C0、C1、C2、および C3 フラグは未定義。

浮動小数点例外

なし。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSTENV/FNSTENV—Store x87 FPU Environment（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSTSW/FNSTSW—Store x87 FPU Status Word

オペコード	命令	説明
9B DD /7	FSTSW *m2byte*	未処理のマスクされていない浮動小数点例外の有無をチェックした後、FPU ステータス・ワードを *m2byte* にストアする。
9B DF E0	FSTSW AX	未処理のマスクされていない浮動小数点例外の有無をチェックした後、FPU ステータス・ワードを AX レジスタにストアする。
DD /7	FNSTSW* *m2byte*	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、FPU ステータス・ワードを *m2byte* にストアする。
DF E0	FNSTSW* AX	未処理のマスクされていない浮動小数点例外の有無をチェックしないで、FPU ステータス・ワードを AX レジスタにストアする。

注：

* 下記の「IA-32 アーキテクチャにおける互換性」の項を参照。

説明

x87 FPU ステータス・ワードの現在値をデスティネーション・ロケーションにストアする。デスティネーション・オペランドには、2 バイトのメモリ・ロケーションまたは AX レジスタを使用できる。FSTSW 命令は、未処理のマスクされていない浮動小数点例外の有無をチェックし、例外を処理してから、ステータス・ワードをストアする。FNSTSW 命令はこのチェックを行わない。

この命令の FNSTSW AX 形式は、主として分岐の方向が FPU 条件コードフラグの状態に依存する条件付き分岐（例えば、FPU の比較命令、あるいは FPREM、FPREM1、または FXAM 命令の後）で使用される。（『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 8 章の「条件コードに基づく分岐と条件付き移動」の説明を参照のこと。）この命令は、さらに、割り込みを使用しない環境で、（例外フラグを調べることによって）例外ハンドラを呼び出す場合にも使用される。FNSTSW AX 命令を実行すると、プロセッサは、AX レジスタを更新してからその後の命令を実行する。したがって、AX レジスタにストアされているステータスは、その前の FPU 命令の実行結果から与えられたものであることが保証される。

FSTSW 命令の場合、アセンブラは、2 つの命令を発行する（つまり、FWAIT 命令に続けて FNSTSW 命令）。プロセッサは、これらの命令をそれぞれ個別に実行する。これらの命令のいずれかで例外が発生すると、セーブ EIP は、例外を発生させた命令をポイントする。

FSTSW/FNSTSW—Store x87 FPU Status Word（続き）

IA-32 アーキテクチャにおける互換性

インテル® Pentium® プロセッサまたはIntel486™ プロセッサをMS-DOS* 互換モードで動作させたときは、（通常の状況下で）実行される前にFNSTSW 命令に割り込みをかけて、未処理の FPU 例外を処理させることができる。それらの状況については、『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の「非同期型命令のウィンドウ内のx87 FPU割り込み」の説明を参照のこと。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・プロセッサでは、FNSTSW命令にこの方法では割り込みをかけられない。

操作

```
DEST ← FPUStatusWord;
```

FPU 影響を受けるフラグ

C0、C1、C2、およびC3フラグは未定義。

浮動小数点例外

なし。

保護モード例外

#GP(0)	デスティネーションが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスがCS、DS、ES、FS、またはGSセグメントの範囲外の場合。
	DS、ES、FS、またはGSレジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスがSSセグメントの範囲外の場合。
#NM	CR0のEMまたはTSがセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが3のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスがCS、DS、ES、FS、またはGSセグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスがSSセグメントの範囲外の場合。
#NM	CR0のEMまたはTSがセットされた場合。

FSTSW/FNSTSW—Store x87 FPU Status Word（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSUB/FSUBP/FISUB—Subtract

オペコード	命令	説明
D8 /4	FSUB *m32fp*	ST(0) から *m32fp* を引き、結果を ST(0) にストアする。
DC /4	FSUB *m64fp*	ST(0) から *m64fp* を引き、結果を ST(0) にストアする。
D8 E0+i	FSUB ST(0), ST(i)	ST(0) から ST(i) を引き、結果を ST(0) にストアする。
DC E8+i	FSUB ST(i), ST(0)	ST(i) から ST(0) を引き、結果を ST(i) にストアする。
DE E8+i	FSUBP ST(i), ST(0)	ST(i) から ST(0) を引き、結果を ST(i) にストアし、レジスタスタックをポップする。
DE E9	FSUBP	ST(1) から ST(0) を引き、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /4	FISUB *m32int*	ST(0) から *m32int* を引き、結果を ST(0) にストアする。
DE /4	FISUB *m16int*	ST(0) から *m16int* を引き、結果を ST(0) にストアする。

説明

デスティネーション・オペランドからソース・オペランドを引き、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランドは、常にFPU データレジスタである。ソース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

この命令のオペランドなしバージョンでは、ST(1) レジスタの内容から ST(0) レジスタの内容を引き、結果を ST(1) レジスタにストアする。1 オペランド・バージョンでは、ST(0) レジスタの内容からメモリ・ロケーションの内容（浮動小数点値または整数値）を引き、結果を ST(0) にストアする。2 オペランド・バージョンでは、ST(i) レジスタの内容から ST(0) レジスタの内容を引くか、またはその逆の減算を行う。

FSUBP 命令は、減算の後に、追加操作として FPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。浮動小数点減算命令のオペランドなしバージョンでは、常にレジスタスタックのポップ操作を伴う。一部のアセンブラでは、この命令のニーモニックは FSUBP ではなく FSUB になっている。

FISUB 命令は、整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから減算を行う。

以下の表に、アンダーフローもオーバーフローも発生しないものとして、さまざまなクラスの数の減算を行ったときに得られる結果を示す。この表では、DEST 値から SRC 値が引かれる（DEST – SRC ← 結果）。

FSUB/FSUBP/FISUB—Subtract（続き）

同符号の 2 つのオペランドの差が 0 のときは、–∞ 方向の丸めモードの場合を除いて、結果は +0 になる。–∞ 方向の丸めモードの場合、結果は –0 になる。この命令は、+0 – (–0) = +0、–0 – (+0) = –0 の結果も保証する。ソース・オペランドは、整数 0 のときは +0 として取り扱われる。

一方のオペランドが∞のときは、結果は予期される符号の∞になる。両方のオペランドが同じ符号で無限大のときは、無効操作例外が発生する。

		SRC						
		–∞	–F or –I	–0	+0	+F or +I	+∞	NaN
DEST	–∞	*	–∞	–∞	–∞	–∞	–∞	NaN
	–F	+∞	±F or ±0	DEST	DEST	–F	–∞	NaN
	–0	+∞	–SRC	±0	–0	–SRC	–∞	NaN
	+0	+∞	–SRC	+0	±0	–SRC	–∞	NaN
	+F	+∞	+F	DEST	DEST	±F or ±0	–∞	NaN
	+∞	+∞	+∞	+∞	+∞	+∞	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

I 整数を示す。

* 浮動小数点無効算術オペランド（#IA）例外 を示す。

操作

```
IF instruction is FISUB
    THEN
        DEST ← DEST – ConvertToDoubleExtendedPrecisionFP(SRC);
    ELSE (* source operand is floating-point value *)
        DEST ← DEST – SRC;
FI;
IF instruction is FSUBP
    THEN
        PopRegisterStack
FI;
```

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされる。

結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。

C0、C2、C3　未定義。

FSUB/FSUBP/FISUB—Subtract（続き）

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
	両方のオペランドの符号が同じで、絶対値が無限大の場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FSUB/FSUBP/FISUB—Subtract（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FSUBR/FSUBRP/FISUBR—Reverse Subtract

オペコード	命令	説明
D8 /5	FSUBR *m32fp*	*m32fp* から ST(0) を引き、結果を ST(0) にストアする。
DC /5	FSUBR *m64fp*	*m64fp* から ST(0) を引き、結果を ST(0) にストアする。
D8 E8+i	FSUBR ST(0), ST(i)	ST(i) から ST(0) を引き、結果を ST(0) にストアする。
DC E0+i	FSUBR ST(i), ST(0)	ST(0) から ST(i) を引き、結果を ST(i) にストアする。
DE E0+i	FSUBRP ST(i), ST(0)	ST(0) から ST(i) を引き、結果を ST(i) にストアし、レジスタスタックをポップする。
DE E1	FSUBRP	ST(0) から ST(1) を引き、結果を ST(1) にストアし、レジスタスタックをポップする。
DA /5	FISUBR *m32int*	*m32int* から ST(0) を引き、結果を ST(0) にストアする。
DE /5	FISUBR *m16int*	*m16int* から ST(0) を引き、結果を ST(0) にストアする。

説明

ソース・オペランドからデスティネーション・オペランドを引き、結果をデスティネーション・ロケーションにストアする。デスティネーション・オペランドは、常にFPU データレジスタである。ソース・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。メモリ内のソース・オペランドには、単精度浮動小数点、倍精度浮動小数点、ワード整数、またはダブルワード整数のフォーマットを使用できる。

これらの命令は、それぞれFSUB、FSUBP、FISUB命令の逆の操作を行う。これらは、コーディングの効率向上をサポートする目的で設けられたものである。

この命令のオペランドなしバージョンでは、ST(0) レジスタの内容からST(1) レジスタの内容を引き、結果をST(1) レジスタにストアする。1 オペランド・バージョンでは、メモリ・ロケーションの内容（浮動小数点値または整数値）からST(0) レジスタの内容を引き、結果をST(0) にストアする。2 オペランド・バージョンでは、ST(0) レジスタの内容からST(i) レジスタの内容を引くか、またはその逆の減算を行う。

FSUBRP 命令は、減算の後に、追加操作としてFPU レジスタスタックをポップする。レジスタスタックをポップするため、プロセッサはST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。浮動小数点逆減算命令のオペランドなしバージョンでは、常にレジスタスタックのポップ操作を伴う。一部のアセンブラでは、この命令のニーモニックはFSUBRPではなくFSUBRになっている。

FISUBR 命令は、整数のソース・オペランドを拡張倍精度浮動小数点フォーマットに変換してから減算を行う。

FSUBR/FSUBRP/FISUBR—Reverse Subtract（続き）

以下の表に、アンダーフローもオーバーフローも発生しないものとして、さまざまなクラスの数の減算を行ったときに得られる結果を示す。この表では、SRC 値から DEST 値が引かれる（SRC – DEST = 結果）。

同符号の 2 つのオペランドの差が 0 のときは、–∞ 方向の丸めモードの場合を除いて、結果は +0 になる。–∞ 方向の丸めモードの場合、結果は –0 になる。この命令は、+0 – (–0) = +0、–0 – (+0) = –0 の結果も保証する。ソース・オペランドは、整数 0 のときは +0 として取り扱われる。

一方のオペランドが∞のときは、結果は予期される符号の∞になる。両方のオペランドが同じ符号で無限大のときは、無効操作例外が発生する。

		SRC						
		–∞	–F or –I	–0	+0	+F or +I	+∞	NaN
DEST	–∞	*	+∞	+∞	+∞	+∞	+∞	NaN
	–F	–∞	±F or ±0	–DEST	–DEST	+F	+∞	NaN
	–0	–∞	SRC	±0	+0	SRC	+∞	NaN
	+0	–∞	SRC	–0	±0	SRC	+∞	NaN
	+F	–∞	–F	–DEST	–DEST	±F or ±0	+∞	NaN
	+∞	–∞	–∞	–∞	–∞	–∞	*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F　有限浮動小数点値を示す。

I　整数を示す。

*　浮動小数点無効算術オペランド（#IA）例外を示す。

操作

```
IF instruction is FISUBR
    THEN
        DEST ← ConvertToDoubleExtendedPrecisionFP(SRC) – DEST;
    ELSE (* source operand is floating-point value *)
        DEST ← SRC – DEST;
FI;
IF instruction = FSUBRP
    THEN
        PopRegisterStack
FI;
```

FSUBR/FSUBRP/FISUBR—Reverse Subtract（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。 両方のオペランドの符号が同じで、絶対値が無限大の場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。

FSUBR/FSUBRP/FISUBR—Reverse Subtract（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の EM または TS がセットされた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

FTST—TEST

オペコード	命令	説明
D9 E4	FTST	ST(0) を 0.0 と比較する。

説明

ST(0) レジスタの値を 0.0 と比較し、結果にしたがって FPU ステータス・ワード内の条件コードフラグ C0、C2、および C3 をセットする (以下の表を参照)。

比較結果	C3	C2	C0
ST(0) > 0.0	0	0	0
ST(0) < 0.0	0	0	1
ST(0) = 0.0	1	0	0
順序化不可能	1	1	1

この命令は「順序化不可能比較」を行う。順序化不可能比較は、比較される両数値のクラスのチェックも行う（本章の「FXAM—Examine」を参照）。ST(0) レジスタの値が NaN であるか、またはそのフォーマットが未定義の場合は、条件コードフラグが「順序化不可能」に設定され、無効操作例外が発生する。

ゼロの符号は無視される。すなわち、–0.0 ← +0.0 である。

操作

```
CASE (relation of operands) OF
    Not comparable:C3, C2, C0 ← 111;
    ST(0) > 0.0:   C3, C2, C0 ← 000;
    ST(0) < 0.0:   C3, C2, C0 ← 001;
    ST(0) ← 0.0:  C3, C2, C0 ← 100;
ESAC;
```

FPU 影響を受けるフラグ

C1　スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にクリアされる。

C0、C2、C3　上記の表を参照。

浮動小数点例外

#IS　スタック・アンダーフローが発生した場合。

#IA　ソース・オペランドが NaN 値であるか、またはそのフォーマットがサポートされていない場合。

#D　ソース・オペランドがデノーマル値である場合。

FTST—TEST（続き）

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FUCOM/FUCOMP/FUCOMPP—Unordered Compare Floating Point Values

オペコード	命令	説明
DD E0+i	FUCOM ST(i)	ST(0) を ST(i) と比較する。
DD E1	FUCOM	ST(0) を ST(1) と比較する。
DD E8+i	FUCOMP ST(i)	ST(0) を ST(i) と比較し、レジスタスタックをポップする。
DD E9	FUCOMP	ST(0) を ST(1) と比較し、レジスタスタックをポップする。
DA E9	FUCOMPP	ST(0) を ST(1) と比較し、レジスタスタックを 2 回ポップする。

説明

ST(0) および ST(i) レジスタの内容の順序化不可能比較を行い、結果にしたがって FPU ステータス・ワード内の条件コードフラグ C0、C2、C3 をセットする（下記の表を参照）。オペランドを指定しなかった場合は、ST(0) および ST(1) レジスタの内容が比較される。ゼロの符号は無視される。すなわち、–0.0 は +0.0 と等しい。

比較結果	C3	C2	C0
ST0 > ST(i)	0	0	0
ST0 < ST(i)	0	0	1
ST0 ← ST(i)	1	0	0
順序化不可能	1	1	1

注：

* マスクされていない無効算術オペランド（#IA）例外が発生した場合、フラグはセットされない。

順序化不可能比較は、比較される両数値のクラスのチェックを行う（本章の「FXAM—Examine」を参照）。FUCOM/FUCOMP/FUCOMPP 命令では、一方または両方のオペランドが SNaN であるか、またはそれらのフォーマットがサポートされていないときだけに無効算術オペランド例外（#IA）を発生させるという点を除いて、FUCOM/FUCOMP/FUCOMPP 命令の操作は FCOM/FCOMP/FCOMPP 命令の操作と同じである。QNaN の場合は、条件コードフラグが順序化不可能に設定されるが、例外は発生しない。それに対して、一方または両方のオペランドがいずれかの種類の NaN 値であるか、またはそれらのフォーマットがサポートされていないときは、FCOM/FCOMP/FCOMPP 命令は無効操作例外を発生させる。

FCOM/FCOMP/FCOMPP 命令の場合と同様に、操作の結果によって無効算術オペランド例外が発生する場合は、例外がマスクされている場合だけ条件コードフラグがセットされる。

FUCOM/FUCOMP/FUCOMPP—Unordered Compare Floating Point Values（続き）

FUCOMP 命令は比較操作の後にレジスタスタックをポップし、FUCOMPP 命令は比較操作の後にレジスタスタックを 2 回ポップする。レジスタスタックをポップするため、プロセッサは ST(0) レジスタを空としてマークし、スタックポインタ（TOP）を 1 インクリメントする。

操作

```
CASE (relation of operands) OF
    ST > SRC:    C3, C2, C0 ← 000;
    ST < SRC:    C3, C2, C0 ← 001;
    ST = SRC:    C3, C2, C0 ← 100;
ESAC;
IF ST(0) or SRC = QNaN, but not SNaN or unsupported format
    THEN
        C3, C2, C0 ← 111;
    ELSE (* ST(0) or SRC is SNaN or unsupported format *)
         #IA;
        IF FPUControlWord.IM = 1
            THEN
                C3, C2, C0 ← 111;
        FI;
FI;
IF instruction = FUCOMP
    THEN
        PopRegisterStack;
FI;
IF instruction = FUCOMPP
    THEN
        PopRegisterStack;
        PopRegisterStack;
FI;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。
C0、C2、C3	前ページの表を参照。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	一方または両方のオペランドが SNaN 値であるか、またはそれらのフォーマットがサポートされていない場合。QNaN 値が検出されても、無効オペランド例外は発生しない。
#D	一方または両方のオペランドがデノーマル値である場合。

FUCOM/FUCOMP/FUCOMPP—Unordered Compare Floating Point Values（続き）

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FWAIT—Wait

本章の「WAIT/FWAIT—Wait」を参照のこと。

FXAM—Examine

オペコード	命令	説明
D9 E5	FXAM	ST(0) の値または数値をクラスに分類する。

説明

ST(0) レジスタの内容を調べ、FPU ステータス・ワード内の条件コードフラグ C0、C2、および C3 を設定して ST(0) レジスタの値または数値のクラスを示す（下表を参照）。

クラス	C3	C2	C0
サポートされていない	0	0	0
NaN	0	0	1
通常の有限値	0	1	0
無限大	0	1	1
ゼロ	1	0	0
空	1	0	1
デノーマル数	1	1	0

C1 フラグは、レジスタが空であるか値がロードされているかに関係なく、ST(0) の値の符号に設定される。

操作

```
C1 ← sign bit of ST; (* 0 for positive, 1 for negative *)
CASE (class of value or number in ST(0)) OF
    Unsupported:C3, C2, C0 ← 000;
    NaN:        C3, C2, C0 ← 001;
    Normal:     C3, C2, C0 ← 010;
    Infinity:   C3, C2, C0 ← 011;
    Zero:       C3, C2, C0 ← 100;
    Empty:      C3, C2, C0 ← 101;
    Denormal:   C3, C2, C0 ← 110;
ESAC;
```

FPU 影響を受けるフラグ

C1　　ST(0) の値の符号に設定される。

C0、C2、C3　　上記の表を参照。

浮動小数点例外

なし。

FXAM—Examine（続き）

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FXCH—Exchange Register Contents

オペコード	命令	説明
D9 C8+i	FXCH ST(i)	ST(0) と ST(i) の内容を入れ換える。
D9 C9	FXCH	ST(0) と ST(1) の内容を入れ換える。

説明

ST(0) レジスタの内容と ST(i) レジスタの内容を入れ換える。ソース・オペランドが指定されていない場合は、ST(0) と ST(1) の内容が入れ換えられる。

この命令によって、FPU レジスタスタックの値をスタックのトップである [ST(0)] に転送する単純な手段が提供される。したがって、転送された値は、ST(0) の値だけを操作できる浮動小数点命令だけによって操作できるようになる。例えば、以下の命令シーケンスでは、レジスタスタックのトップから 3 番目のレジスタの平方根を計算する。

```
FXCH ST(3);
FSQRT;
FXCH ST(3);
```

操作

```
IF number-of-operands is 1
    THEN
        temp ← ST(0);
        ST(0) ← SRC;
        SRC ← temp;
    ELSE
        temp ← ST(0);
        ST(0) ← ST(1);
        ST(1) ← temp;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、発生しなかった場合は 0 にクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

FXCH—Exchange Register Contents（続き）

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FXRSTOR—Restore x87 FPU, MMX Technology, SSE, and SSE2 State

オペコード	命令	説明
0F AE /1	FXRSTOR *m512byte*	x87 FPU レジスタ状態、MMX® テクノロジ・レジスタ状態、XMM レジスタ状態、および MXCSR レジスタ状態を *m512byte* からリストアする。

説明

ソース・オペランドに指定された 512 バイトのメモリイメージから、x87 FPU、MMX® テクノロジ、XMM、MXCSR の各レジスタを再ロードする。再ロードされるデータは、FXSAVE 命令を使ってあらかじめメモリに書き込まれていなければならない。また、再ロードされるデータの先頭のバイトは、16 バイト境界上に配置されていなければならない。表 3-16. に、メモリ内の状態情報のレイアウトを示し、FXRSTOR 命令および FXSAVE 命令におけるメモリイメージの各フィールドを説明する。

FXRSTOR 命令で参照される状態情報のイメージは、FXSAVE 命令を使ってセーブしなければならない。あるいは、表 3-16. で示されるのと同じフォーマットでなければならない。FSAVE 命令や FNSAVE 命令を使ってセーブされた状態情報のイメージを参照すると、正しくない状態情報が復元される。

FXRSTOR 命令は、未処理の x87 FPU 例外をフラッシュしない。FXRSTOR 命令で x87 FPU 状態情報をロードするときに未処理の例外をチェックして処理するには、FXRSTOR 命令の後で FWAIT 命令を使用する。

制御レジスタ CR4 の OSFXSR ビットを設定しないと、FXRSTOR 命令を実行しても、XMM レジスタおよび MXCSR レジスタの状態が復元されないことがある。この動作は、プロセッサによって異なる。

マスクされていない例外が MXCSR の状態に含まれていて、それに対応するステータス・フラグがセットされている場合、FXRSTOR 命令でレジスタをロードしても、SIMD 浮動小数点エラー条件は発生しない。このマスクされていない例外が次に発生したときに、例外が発生する。

MXCSR レジスタのビット 6 およびビット 16 ～ 32 は予約済みとして定義されており、0 に設定する必要がある。セーブされている状態情報のイメージのこれらのビットのいずれかに 1 を書き込もうとすると、一般保護例外（#GP）が発生する。

操作

(x87 FPU, MMX, XMM7-XMM0, MXCSR) ← Load(SRC);

x87 FPU および SIMD 浮動小数点例外

なし。

FXRSTOR—Restore x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。（下記のアライメント・チェック例外 [#AC] を参照のこと）
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされている場合。
#UD	CR0 の EM がセットされている場合。
	CPUID 機能フラグの FXSR が 0 の場合。
	命令の前に LOCK プリフィックスが置かれている場合。
#AC	この例外がディスエーブルにされている場合、前述したようにメモリ・オペランドが 16 バイト境界上にアライメントされていなければ、一般保護例外（#GP）が報告される。アライメント・チェック例外（#AC）がイネーブルになっている場合（および CPL が 3 の場合）、#AC は必ずしも報告されるとは限らない。これは、以下に示すように、プロセッサによって異なる。#AC が報告されないすべてのプロセッサでは、#GP が所定の場所で報告される。また、アライメント・チェックの幅も、プロセッサによって異なる。例えば、あるプロセッサでは、2 バイトのミスアライメントに対して #AC が報告され、2 バイト以外のすべてのミスアライメント（4 、18 、16 バイトのミスアライメント）に対しては #GP が報告される。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされている場合。
	CPUID 機能フラのグ SSE2 が 0 の場合。
	命令の前に LOCK オーバーライド・プリフィックスが置かれている場合。

FXRSTOR—Restore x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC	現行特権レベルが 3 のときに、アライメントの合っていないメモリ参照を行った場合。

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State

オペコード	命令	説明
0F AE /0	FXSAVE *m512byte*	x87 FPU レジスタ状態、MMX® テクノロジ・レジスタ状態、XMM レジスタ状態、MXCSR レジスタ状態を *m512byte* に保存する。

説明

x87 FPU、MMX® テクノロジ、XMM、MXCSR の各レジスタの現在の状態を、デスティネーション・オペランドで指定された 512 バイトのメモリ・ロケーションにセーブする。表 3-16. は、メモリ内の状態情報のレイアウトを示している。

表 3-16. FXSAVE と FXRSTOR メモリ領域のレイアウト

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
予約	CS	FPU IP		FOP		FTW	FSW	FCW	0
MXCSR_MASK		MXCSR		予約	DS		FPU DP		16
予約			ST0/MM0						32
予約			ST1/MM1						48
予約			ST2/MM2						64
予約			ST3/MM3						80
予約			ST4/MM4						96
予約			ST5/MM5						112
予約			ST6/MM6						128
予約			ST7/MM7						144
XMM0									160
XMM1									176
XMM2									192
XMM3									208
XMM4									224
XMM5									240
XMM6									256
XMM7									272
予約									288
予約									304
予約									320
予約									336
予約									352
予約									368
予約									384
予約									400
予約									416
予約									432
予約									448
予約									464
予約									480
予約									496

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

デスティネーション・オペランドには、メモリイメージの最初のバイトが格納される。デスティネーション・オペランドは、16バイト境界上にアライメントされていなければならない。デスティネーション・オペランドのアライメントが合っていないと、一般保護 (#GP) 例外 (場合によっては、アライメント・チェック例外 [#AC]) が発生する。

FXSAVE 命令が使用されるのは、オペレーティング・システムがコンテキスト・スイッチを実行する必要がある場合、または、例外ハンドラが、x87 FPU、MMX テクノロジ、XMM、MXCSR の各レジスタの現在の状態をセーブおよび検査する必要がある場合である。

表 3-16. の各フィールドを以下に説明する。

FCW　x87 FPU制御ワード (16 ビット)。x87 FPU制御ワードのレイアウトについては、『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 8-6. を参照のこと。

FSW　x87 FPU ステータス・ワード（16 ビット）。x87 FPU ステータス・ワードのレイアウトについては、『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 8-4. を参照のこと。

FTW　x87 FPU タグワード（8 ビット)。ここにセーブされるタグ情報は短縮形式である。詳しくは、以降の段落で説明する。x87 FPU タグワードのレイアウトについては、『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 8-7. を参照のこと。

FOP　x87 FPU オペコード（16 ビット)。このフィールドの下位 11 ビットにはオペコードが入っており、上位 5 ビットは予約されている。x87 FPU オペコード・フィールドのレイアウトについては、『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 8-8. を参照のこと。

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

FPU IP

x87 FPU 命令ポインタ・オフセット（32 ビット）。このフィールドの内容は、FXSAVE 命令の実行時にプロセッサのアドレス指定モードが32ビットまたは16ビットのどちらであったかによって異なる。

- 32 ビット・モード - 32 ビットの IP オフセット。
- 16 ビット・モード - 下位 16 ビットは IP オフセット。上位 16 ビットは予約済み。

x87 FPU 命令ポインタの説明については、『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」を参照のこと。

CS

x87 FPU 命令ポインタセレクタ（16 ビット）。

FPU DP

x87 FPU 命令オペランド（データ）ポインタ・オフセット（32 ビット）。このフィールドの内容は、FXSAVE 命令の実行時にプロセッサのアドレス指定モードが 32 ビットまたは 16 ビットのどちらであったかによって異なる。

- 32 ビット・モード - 32 ビットの IP オフセット。
- 16 ビット・モード - 下位 16 ビットは IP オフセット。上位 16 ビットは予約済み。

x87 FPU オペランド・ポインタの説明については、『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 命令とデータ（オペランド）ポインタ」を参照のこと。

DS

x87 FPU 命令オペランド（データ）ポインタセレクタ（16 ビット）。

MXCSR

MXCSR レジスタ状態（32 ビット）。MXCSR レジスタのレイアウトについては、『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 10-3. を参照のこと。制御レジスタ CR4 の OSFXSR ビットが設定されていない場合、FXSAVE 命令を実行しても、このレジスタの値がセーブされない場合がある。この動作は、プロセッサによって異なる。

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

MXCSR_ MASK	MXCSR_MASK（32 ビット）。このマスクを使用して、すべての予約ビットが 0 に設定されるように、MXCSR レジスタへ書き込む値を調整することができる。MXCSR レジスタ内のマスクビットとフラグを、SSE および SSE2 SIMD 浮動小数点命令に適した動作モードに設定すること。MXCSR_MASK 値の決定や使用の方法については、『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「MXCSR レジスタへの書き込みに関するガイドライン」を参照のこと。
ST0/MM0 ～ ST7/MM7	x87 FPU/MMX テクノロジ・レジスタ。これらの 80 ビットのフィールドには、FXSAVE 命令を実行する前のプロセッサの状態に応じて、x87 FPU データレジスタまたは MMX テクノロジ・レジスタの値が入る。FXSAVE 命令の前に x87 FPU 命令が実行されていた場合は、x87 FPU データレジスタがセーブされる。MMX 命令（または MMX テクノロジ・レジスタ上で動作していた SSE、SSE2 命令）が実行されていた場合は、MMX テクノロジ・レジスタの値がセーブされる。MMX テクノロジ・レジスタの値がセーブされると、このフィールドの上位 16 ビットは予約される。
XMM0 ～ XMM7	XMM レジスタ（1 フィールド当たり 128 ビット）。制御レジスタ CR4 の OSFXSR ビットを設定しないと、FXSAVE 命令を実行しても、これらのレジスタの値がセーブされない場合がある。この動作は、プロセッサによって異なる。

FXSAVE 命令は、x87 FPU タグワードの短縮形を FTW フィールドにセーブする（タグワードを完全な形でセーブする FSAVE 命令とは異なる）。タグ情報は、トップオブスタック（TOS）の順序ではなく、物理的なレジスタの順序（R0 から R7）でセーブされる。ただし、FXSAVE 命令では、タグごとに単一のビット（1 は有効、0 は空）しかセーブされない。例えば、現在、タグワードが次のように設定されていると仮定する。

R7	R6	R5	R4	R3	R2	R1	R0
11	xx	xx	xx	11	11	11	11

ここで、11B は空のスタック要素を表し、"xx" については、（00B）が有効、（01B）がゼロ、（10B）が特殊を表す。

この例の場合、FXSAVE 命令は、次に示すような 8 ビットの情報しかセーブしない。

R7	R6	R5	R4	R3	R2	R1	R0
0	1	1	1	0	0	0	0

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

ここで、1 は有効、ゼロ、特殊のタグについてセーブされ、0 は空のタグについてセーブされる。

FXSAVE 命令の動作は、以下の点で、FSAVE 命令とは異なっている。

- FXSAVE 命令は、未処理のマスクされていない浮動小数点例外のチェックを行わない。（この点については、FXSAVE 命令と FNSAVE 命令の動作は同じである。）
- x87 FPU、MMX テクノロジ、XMM、MXCSR の各レジスタの状態が FXSAVE 命令によってセーブされると、プロセッサは、これらのレジスタの内容を保持する。この動作のために、アプリケーション・プログラムにおいて FXSAVE 命令を使って「きれいにした」x87 FPU 状態をプロシージャに渡すことはできない。FXSAVE 命令では、現在の状態が保持されるためである。x87 FPU 状態をきれいにするには、アプリケーションにおいて FXSAVE 命令の後に FINIT 命令を明示的に実行し、x87 FPU 状態を再度初期化する必要がある。
- FXSAVE 命令でセーブされるメモリイメージのフォーマットは、現在のアドレス指定モード（32 ビットまたは 16 ビット）や、動作モード（保護、実アドレス、またはシステム管理）にかかわらず、一定である。この点は、アドレス指定モードや動作モードによってメモリイメージのフォーマットが変わる FSAVE 命令とは異なっている。FXSAVE 命令でセーブされたメモリイメージは、イメージ・フォーマットが異なるため、FRSTOR 命令で正しく復元することはできない。また、これと同様に、FSAVE 命令でセーブされた状態は FXRSTOR 命令で正しく復元することはできない。

FSAVE 命令フォーマットの FTW については、以下の表を使用することにより、FTW の有効ビットと、ストアされている 80 ビットの FP データ（ストアされているデータが MMX テクノロジ・レジスタの内容ではない場合）とから再作成することができる。

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

指数 すべて 1	指数 すべて 0	小数 すべて 0	J ビットおよび M ビット	FTW 有効ビット	x87 FTW	
0	0	0	0x	1	特殊	10
0	0	0	1x	1	有効	00
0	0	1	00	1	特殊	10
0	0	1	10	1	有効	00
0	1	0	0x	1	特殊	10
0	1	0	1x	1	特殊	10
0	1	1	00	1	ゼロ	01
0	1	1	10	1	特殊	10
1	0	0	1x	1	特殊	10
1	0	0	1x	1	特殊	10
1	0	1	00	1	特殊	10
1	0	1	10	1	特殊	10
上記のすべての有効な組み合わせ				0	空	11

J ビットは、仮数の小数点の左側の 1 ビットの 2 進整数として定義される。M ビットは、仮数の小数点以下の部分の最上位ビット（すなわち、小数点のすぐ右側のビット）として定義される。

M ビットが仮数の小数点以下の部分の最上位ビットのとき、その小数がすべて 0 の場合は、M ビットは 0 でなければならない。

操作

```
DEST ← Save(x87 FPU, MMX, XXM7-XMM0, MXCSR);
```

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。（下記のアライメント・チェック例外 [#AC] の説明を参照のこと）
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

FXSAVE—Save x87 FPU, MMX Technology, SSE, and SSE2 State（続き）

#UD	CR0 の EM がセットされている場合。 CPUID 機能フラグの FXSR が 0 の場合。 命令の前に LOCK プリフィックスが置かれている場合。
#AC	この例外がディスエーブルにされている場合、前述したようにメモリ・オペランドが 16 バイト境界上にアライメントされていなければ、一般保護例外（#GP）が報告される。アライメント・チェック例外（#AC）がイネーブルになっている場合（および CPL が 3 の場合）、#AC は必ずしも報告されるとは限らない。これは、以下に示すように、プロセッサによって異なる。#AC が報告されないすべてのプロセッサでは、#GP が所定の場所で報告される。また、アライメント・チェックの幅も、プロセッサによって異なる。例えば、あるプロセッサでは、2 バイトのミスアライメントに対して #AC が報告され、2 バイト以外のすべてのミスアライメント（4 、18 、16 バイトのミスアライメント）に対しては #GP が報告される。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされている場合。 CPUID 機能フラグの FXSR が 0 の場合。 命令の前に LOCK プリフィックスが置かれている場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC	現行特権レベルが 3 のときに、アライメントの合っていないメモリ参照を行った場合。

プロセッサに関する注意

命令境界上で一般保護（#GP）例外およびページフォルト（#PF）例外の両方が発生した場合、プロセッサがこれらの例外を報告する順序は、『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、下巻』の表 5-2. に示すとおりである。別の IA-32 プロセッサの FXSAVE 命令では、この順序は異なる。

FXTRACT—Extract Exponent and Significand

オペコード	命令	説明
D9 F4	FXTRACT	ST(0) の値を指数と仮数に分け、指数を ST(0) にストアし、仮数をレジスタスタックにプッシュする。

説明

T(0) レジスタ内のソース値をその指数と仮数に分け、指数を ST(0) にストアし、仮数をレジスタスタックにプッシュする。この操作後は、新しいスタックのトップの ST(0) レジスタの内容は元の仮数の浮動小数点表現値になる。この値の符号と仮数は、ソース・オペランドにあった符号と仮数と同じであり、指数は 3FFFH（真のゼロの指数に対してバイアスがかけられた値）である。ST(1) レジスタの内容は、元のオペランドの真の（バイアスなし）指数の浮動小数点表現値である。（この命令が実行する操作は、IEEE 推奨の log*b*x 関数のスーパセットである。）

この命令と F2XM1 命令は、べき乗および範囲のスケーリング操作を行う場合に有用である。FXTRACT 命令は、拡張倍精度浮動小数点フォーマットの数値を（例えば、印刷または表示するために）10 進表現に変換する場合にも有用である。

浮動小数点ゼロでの除算例外（#Z）がマスクされていて、ソース・オペランドがゼロである場合は、指数値として –∞ が ST(1) レジスタにストアされ、ソース・オペランドの符号と同じ符号の 0 が ST(0) レジスタにストアされる。

操作

```
TEMP ← Significand(ST(0));
ST(0) ← Exponent(ST(0));
TOP← TOP – 1;
ST(0) ← TEMP;
```

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされ、スタック・オーバーフローが発生した場合は 1 にセットされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローまたはスタック・オーバーフローが発生した場合。
#IA	ソース・オペランドが SNaN 値であるか、またはそのフォーマットがサポートされていない場合。
#Z	ST(0) レジスタのオペランドが± 0。
#D	ソース・オペランドがデノーマル値である場合。

FXTRACT—Extract Exponent and Significand（続き）

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FYL2X—Compute y ∗ log_2x

オペコード	命令	説明
D9 F1	FYL2X	ST(1) を (ST(1) ∗ log_2ST(0)) で置き換え、レジスタスタックをポップする。

説明

(ST(1) ∗ log_2(ST(0))) を計算し、結果を ST(1) レジスタにストアし、FPU レジスタスタックをポップする。ST(0) のソース・オペランドはゼロでない正の数値でなければならない。

以下の表に、オーバーフローもアンダーフローも発生しないものとして、さまざまなクラスの数の対数を計算したときに得られる結果を示す。

		ST(0)							
		−∞	−F	±0	+0 < +F < +1	+1	+F > +1	+∞	NaN
ST(1)	−∞	*	*	+∞	+∞	*	−∞	−∞	NaN
	−F	*	*	**	+F	−0	−F	−∞	NaN
	−0	*	*	*	+0	−0	−0	*	NaN
	+0	*	*	*	−0	+0	+0	*	NaN
	+F	*	*	**	−F	+0	+F	+∞	NaN
	+∞	*	*	−∞	−∞	*	+∞	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

注：

F 有限浮動小数点値を示す。

* 浮動小数点無効操作（#IA）例外を示す。

** 浮動小数点ゼロでの除算（#Z）例外を示す。

ゼロでの除算例外がマスクされていて、ST(0) レジスタの内容が± 0 である場合、この命令は、ソース・オペランドの符号と反対の符号の∞を ST(1) レジスタに返す。

FYL2X 命令は、以下に示す組み込み関数乗算を使用して、任意の正の底をもつ対数の計算を最適化するように設計されている。

$log_b x \leftarrow (log_2 b)^{-1} * log_2 x$

操作

```
ST(1) ← ST(1) * log2ST(0);
PopRegisterStack;
```

FYL2X—Compute y ∗ $\log_2$x（続き）

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	いずれかのオペランドが SNaN であるか、またはそのフォーマットがサポートされていない場合。 ST(0) レジスタのソース・オペランドが負の有限値（-0 は除く）である場合。
#Z	ST(0) レジスタのソース・オペランドが± 0 である場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

FYL2XP1—Compute y ∗ $\log_2$(x +1)

オペコード	命令	説明
D9 F9	FYL2XP1	ST(1) を (ST(1) ∗ $\log_2$(ST(0) + 1.0)) で置き換え、レジスタスタックをポップする。

説明

(ST(1) ∗ $\log_2$ (ST(0) + 1.0)) を計算し、結果を ST(1) レジスタにストアし、FPU レジスタスタックをポップする。ST(0) のソース・オペランドは、下記の範囲でなければならない。

$\pm (1 - \sqrt{2}/2)$

ST(1) のソース・オペランドの範囲は、±∞である。ST(0) オペランドがその許容可能な範囲外の場合は、結果は未定義になるので、ソフトウェアは例外が発生するすることに依存してはならない。特定の状況下では、ST(0) が範囲外のとき例外が発生することがあるが、これはプロセッサ固有であり、保証されているものではない。

以下の表に、アンダーフローが発生しないものして、さまざまなクラスの数の ε の対数を計算したときに得られる結果を示す。

		ST(0)				
		$-(1-(\sqrt{2}/2)) \sim -0$	−0	+0	$+0 \sim +(1-(\sqrt{2}/2))$	NaN
ST(1)	−∞	+∞	*	*	−∞	NaN
	−F	+F	+0	−0	−F	NaN
	−0	+0	+0	−0	−0	NaN
	+0	−0	−0	+0	+0	NaN
	+F	−F	−0	+0	+F	NaN
	+∞	−∞	*	*	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN

注 :

F 有限浮動小数点値を示す。

* 浮動小数点無効操作（#IA）例外を示す。

FYL2XP1—Compute y ∗ $\log_2$(x +1)（続き）

この命令は、0 の近似値 ε [ST(0) レジスタの値] に対して最適精度を発揮する。 ε の値が小さい場合は、(ε +1) を引き数として FYL2X 命令を使用するよりも、FYL2XP1 命令を使用する方が、多くの有効数字を残すことができる。(ε +1) という式は、複利や年金の計算によく見受けられる。結果は、ST(1) ソース・オペランドのスケールファクタを含めることにより、簡単に別の対数の底の値に変換することができる。以下の等式は、特定の対数の底のスケールファクタの計算に使用される。ここで、n は FYL2XP1 命令の結果に求められる対数の底である。

scale factor ← $\log_n 2$

操作

ST(1) ← ST(1) ∗ $\log_2$(ST(0) + 1.0);
PopRegisterStack;

FPU 影響を受けるフラグ

C1	スタック・アンダーフローが発生した場合は 0 にセットされる。 結果が切り上げられた場合にセットされる。それ以外の場合はクリアされる。
C0、C2、C3	未定義。

浮動小数点例外

#IS	スタック・アンダーフローが発生した場合。
#IA	いずれかのオペランドが SNaN であるか、またはそのフォーマットがサポートされていない場合。
#D	ソース・オペランドがデノーマル値である場合。
#U	結果が小さすぎて、デスティネーション・フォーマットで表現できない場合。
#O	結果が大きすぎて、デスティネーション・フォーマットで表現できない場合。
#P	値がデスティネーション・フォーマットでは正確に表現できない場合。

保護モード例外

#NM	CR0 の EM または TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の EM または TS がセットされた場合。

FYL2XP1—Compute y ∗ $\log_2(x+1)$（続き）

仮想 8086 モード例外

#NM	CR0 の EM または TS がセットされた場合。

HADDPD—Packed Double-FP Horizontal Add

オペコード	命令	説明
66,0F,7C,/r	HADDPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド倍精度浮動小数点値を水平方向に加算する。

説明

デスティネーション・オペランドの上位クワッドワードと下位クワッドワードの倍精度浮動小数点値を加算し、結果をデスティネーション・オペランドの下位クワッドワードに格納する。

ソース・オペランドの上位クワッドワードと下位クワッドワードの倍精度浮動小数点値を加算し、結果をデスティネーション・オペランドの上位クワッドワードに格納する。

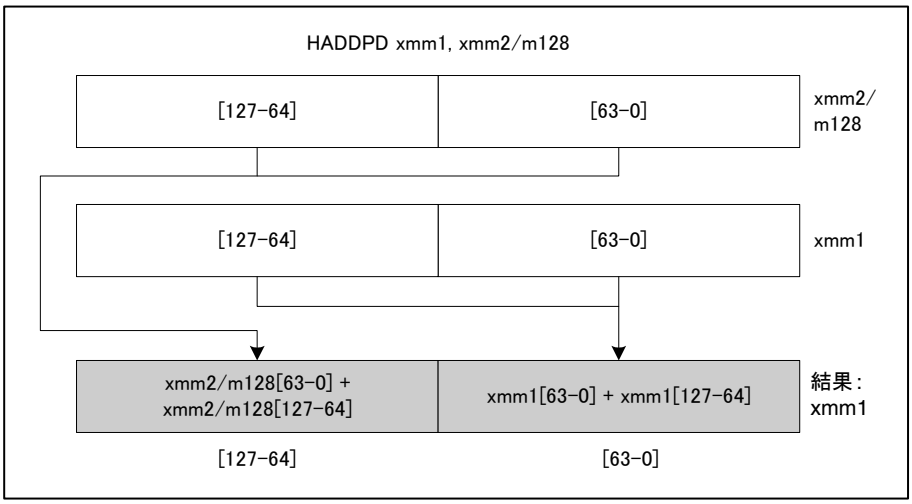

図 3-10. HADDPD: Packed Double-FP Horizontal Add

操作

```
xmm1[63-0]   = xmm1[63-0]   + xmm1[127-64];
xmm1[127-64] = xmm2/m128[63-0] + xmm2/m128[127-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

HADDPD　　__m128d _mm_hadd_pd(__m128d a, __m128d b)

HADDPD—Packed Double-FP Horizontal Add（続き）

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル

保護モード例外

#GP(0)	CS、DS、ES、FS、またはGSセグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが16バイトに合っていない場合。
#SS(0)	SSセグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0のTSビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EMが1の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット9）が0の場合。
	CPUID.SSE3（ECXビット0）が0の場合。

実アドレスモード例外

GP(0)	オペランドの一部が0～FFFFHの実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが16バイトに合っていない場合。
#NM	CR0のTSビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。

HADDPD—Packed Double-FP Horizontal Add（続き）

#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

HADDPS—Packed Single-FP Horizontal Add

オペコード	命令	説明
F2, 0F, 7C, /r	HADDPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド単精度浮動小数点値を水平方向に加算する。

説明

デスティネーション・オペランドの第 1 ダブルワードと第 2 ダブルワードの単精度浮動小数点値を加算し、結果をデスティネーション・オペランドの第 1 ダブルワードに格納する。

デスティネーション・オペランドの第 3 ダブルワードと第 4 ダブルワードの単精度浮動小数点値を加算し、結果をデスティネーション・オペランドの第 2 ダブルワードに格納する。

ソース・オペランドの第 1 ダブルワードと第 2 ダブルワードの単精度浮動小数点値を加算し、結果をデスティネーション・オペランドの第3ダブルワードに格納する。

ソース・オペランドの第 3 ダブルワードと第 4 ダブルワードの単精度浮動小数点値を加算し、結果をデスティネーション・オペランドの第4ダブルワードに格納する。

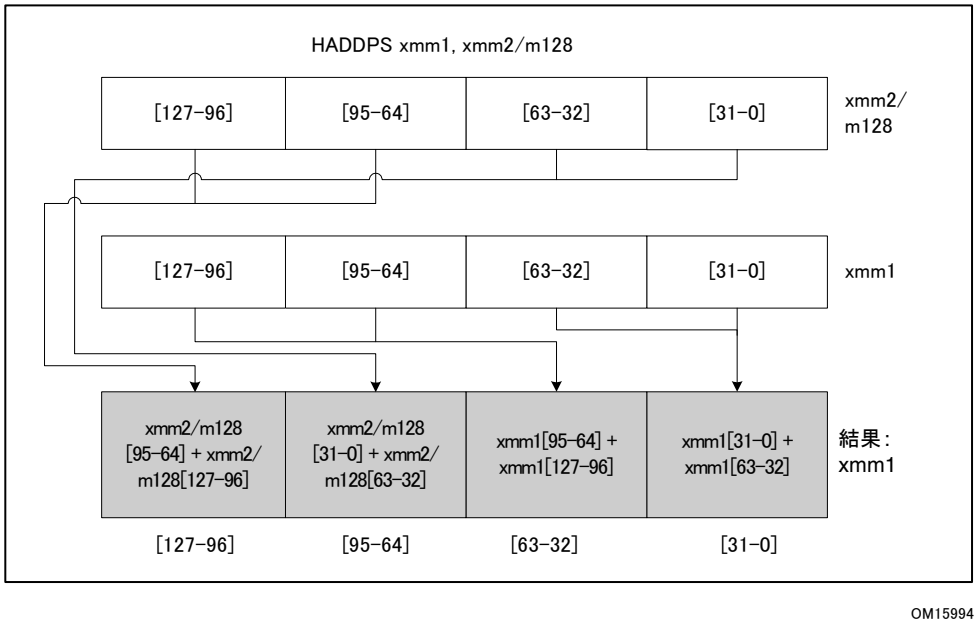

図 3-11. HADDPS: Packed Single-FP Horizontal Add

HADDPS—Packed Single-FP Horizontal Add（続き）

操作

```
xmm1[31-0] = xmm1[31-0] + xmm1[63-32];
xmm1[63-32] = xmm1[95-64] + xmm1[127-96];
xmm1[95-64] = xmm2/m128[31-0] + xmm2/m128[63-32];
xmm1[127-96] = xmm2/m128[95-64] + xmm2/m128[127-96];
```

同等のインテル® C/C++ コンパイラ組み込み関数

HADDPS　　__m128 _mm_hadd_ps(__m128 a, __m128 b)

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル

保護モード例外

#GP(0)	CS、DS、ES、FS、またはGSセグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが16バイトに合っていない場合。
#SS(0)	SSセグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0のTSビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EMが1の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット9）が0の場合。
	CPUID.SSE3（ECXビット0）が0の場合。

HADDPS—Packed Single-FP Horizontal Add（続き）

実アドレスモード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

HLT—Halt

オペコード	命令	説明
F4	HLT	Halt

説明

命令の実行を停止し、プロセッサを HALT 状態にする。イネーブルにされている割り込み（NMI および SMI を含む）、デバッグ例外、BINIT# 信号、INIT# 信号、RESET# 信号によって実行が再開される。HLT 命令の後で割り込み（NMI を含む）を使用して実行を再開する場合、セーブされている命令ポインタ（CS:EIP）は HLT 命令の次の命令を指している。

ハイパー・スレッディング・テクノロジに対応した IA-32 プロセッサ上で HLT 命令を実行した場合は、HLT 命令を実行した論理プロセッサだけが HALT 状態になる。物理プロセッサ内のそれ以外の論理プロセッサは、HLT 命令を実行することで個別に HALT 状態にされない限り、アクティブのままになる。

HLT 命令は特権命令である。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場合は、HLT 命令を実行するには、プログラムまたはプロシージャの特権レベルが 0 でなければならない。

操作

```
Enter Halt state;
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	現行特権レベルが 0 でない場合。

HSUBPD—Packed Double-FP Horizontal Subtract

オペコード	命令	説明
66, 0F, 7D, /r	HSUBPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド倍精度浮動小数点値を水平方向に減算する。

説明

HSUBPD 命令は、ソース・オペランドとデスティネーション・オペランドのパックド倍精度浮動小数点値を水平方向に減算する。

デスティネーション・オペランドの下位クワッドワードの倍精度浮動小数点値からデスティネーション・オペランドの上位クワッドワードの倍精度浮動小数点値を引き、結果をデスティネーション・オペランドの下位クワッドワードに格納する。

ソース・オペランドの下位クワッドワードの倍精度浮動小数点値からソース・オペランドの上位クワッドワードの倍精度浮動小数点値を引き、結果をデスティネーション・オペランドの順位クワッドワードに格納する。

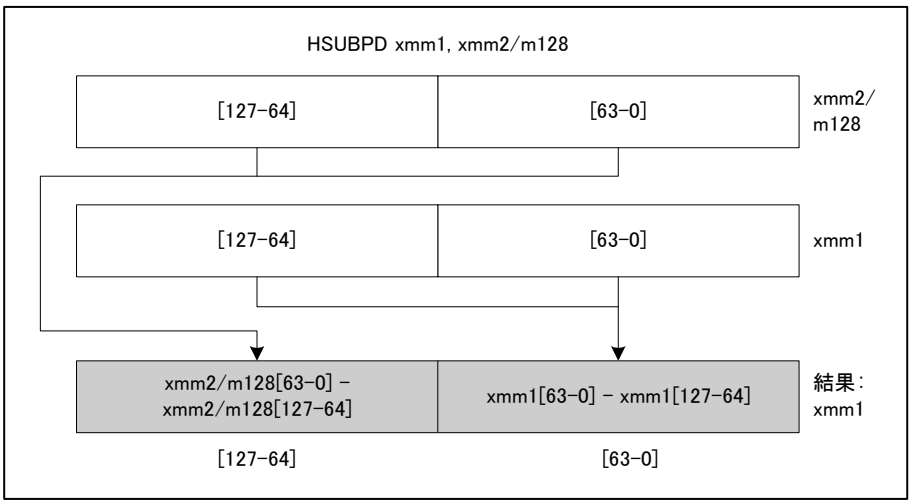

図 3-12. HSUBPD: Packed Double-FP Horizontal Subtract

HSUBPD—Packed Double-FP Horizontal Subtract（続き）

操作

```
xmm1[63-0] = xmm1[63-0] – xmm1[127-64];
xmm1[127-64] = xmm2/m128[63-0] – xmm2/m128[127-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

HSUBPD　　__m128d _mm_hsub_pd(__m128d a, __m128d b)

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

HSUBPD—Packed Double-FP Horizontal Subtract（続き）

実アドレスモード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

HSUBPS—Packed Single-FP Horizontal Subtract

オペコード	命令	説明
F2, 0F, 7D, /r	HSUBPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド単精度浮動小数点値を水平方向に減算する。

説明

デスティネーション・オペランドの第 1 ダブルワードの単精度浮動小数点値からデスティネーション・オペランドの第 2 ダブルワードの単精度浮動小数点値を引き、結果をデスティネーション・オペランドの第 1 ダブルワードに格納する。

デスティネーション・オペランドの第 3 ダブルワードの単精度浮動小数点値からデスティネーション・オペランドの第 4 ダブルワードの単精度浮動小数点値を引き、結果をデスティネーション・オペランドの第 2 ダブルワードに格納する。

ソース・オペランドの第 1 ダブルワードの単精度浮動小数点値からソース・オペランドの第 2 ダブルワードの単精度浮動小数点値を引き、結果をデスティネーション・オペランドの第 3 ダブルワードに格納する。

ソース・オペランドの第 3 ダブルワードの単精度浮動小数点値からソース・オペランドの第 4 ダブルワードの単精度浮動小数点値を引き、結果をデスティネーション・オペランドの第 4 ダブルワードに格納する。

図 3-13 を参照のこと。

HSUBPS—Packed Single-FP Horizontal Subtract（続き）

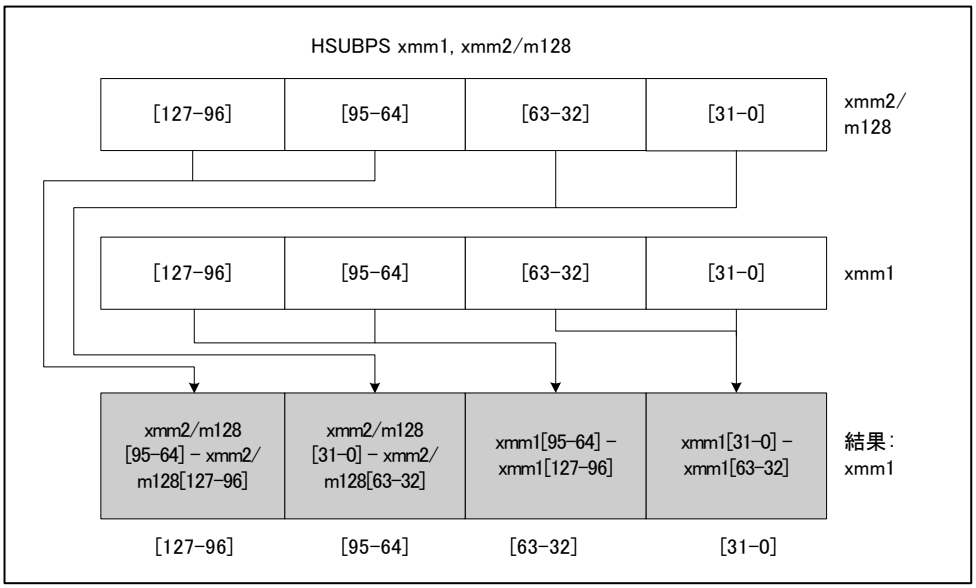

図 3-13. HSUBPS: Packed Single-FP Horizontal Subtract

操作

```
xmm1[31-0] = xmm1[31-0] – xmm1[63-32];
xmm1[63-32] = xmm1[95-64] – xmm1[127-96];
xmm1[95-64] = xmm2/m128[31-0] – xmm2/m128[63-32];
xmm1[127-96] = xmm2/m128[95-64] – xmm2/m128[127-96];
```

同等のインテル® C/C++ コンパイラ組み込み関数

HSUBPS __m128 _mm_hsub_ps(__m128 a, __m128 b)

例外

ソース・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

数値例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル

HSUBPS—Packed Single-FP Horizontal Subtract（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

実アドレスモード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

HSUBPS—Packed Single-FP Horizontal Subtract（続き）

仮想 8086 モード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#NM	CR0 の TS ビットがセットされている場合。
#XM	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 1）。
#UD	CR0.EM が 1 の場合。
	マスクされていないストリーミング SIMD 拡張命令数値例外が発生した場合（CR4.OSXMMEXCPT = 0）。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

IDIV—Signed Divide

オペコード	命令	説明
F6 /7	IDIV *r/m8*	AX を *r/m8* で符号付き除算する。結果は次のようにストアされる。 AL ←商、AH ←剰余
F7 /7	IDIV *r/m16*	DX:AX を *r/m16* で符号付き除算する。結果は次のようにストアされる。 AX ←商、DX ←剰余
F7 /7	IDIV *r/m32*	EDX:EAX を *r/m32* で符号付き除算する。結果は次のようにストアされる。 EAX ←商、EDX ←剰余

説明

AX、DX:AX、EDX:EAX の各レジスタの値（被除数）をソース・オペランド（除数）で（符号付きで）除算し、その結果をそれぞれ AX(AH:AL)、DX:AX、EDX:EAX の各レジスタにストアする。ソース・オペランドには、汎用レジスタまたはメモリ・ロケーションを使用できる。この命令の処理は、以下の表に示すようにオペランド・サイズ（被除数/除数）に依存する。

オペランド・サイズ	被除数	除数	商	剰余	商の範囲
ワード / バイト	AX	r/m8	AL	AH	–128 ～ +127
ダブルワード / ワード	DX:AX	r/m16	AX	DX	–32,768 ～ +32,767
クワッドワード / ダブルワード	EDX:EAX	r/m32	EAX	EDX	-2^{31} ～ $2^{32} - 1$

整数でない結果は 0 に向かって切り捨てられる。剰余の符号は、常に被除数の符号と同じである。剰余の絶対値は、常に除数の絶対値より小さい。オーバーフローは、OF（オーバーフロー）フラグではなく、#DE（除算エラー）例外で示される。

操作

```
IF SRC = 0
    THEN #DE; (* divide error *)
FI;
IF OpernadSize = 8 (* word/byte operation *)
    THEN
        temp ← AX / SRC; (* signed division *)
        IF (temp > 7FH) OR (temp < 80H)
        (* if a positive result is greater than 7FH or a negative result is less than 80H *)
            THEN #DE; (* divide error *) ;
            ELSE
                AL ← temp;
                AH ← AX SignedModulus SRC;
        FI;
```

IDIV—Signed Divide（続き）

```
        ELSE
            IF OpernadSize = 16 (* doubleword/word operation *)
                THEN
                    temp ← DX:AX / SRC; (* signed division *)
                    IF (temp > 7FFFH) OR (temp < 8000H)
                    (* if a positive result is greater than 7FFFH *)
                    (* or a negative result is less than 8000H *)
                        THEN #DE; (* divide error *) ;
                        ELSE
                            AX ← temp;
                            DX ← DX:AX SignedModulus SRC;
                    FI;
                ELSE (* quadword/doubleword operation *)
                    temp ← EDX:EAX / SRC; (* signed division *)
                    IF (temp > 7FFFFFFFH) OR (temp < 80000000H)
                    (* if a positive result is greater than 7FFFFFFFH *)
                    (* or a negative result is less than 80000000H *)
                        THEN #DE; (* divide error *) ;
                        ELSE
                            EAX ← temp;
                            EDX ← EDXE:AX SignedModulus SRC;
                    FI;
            FI;
    FI;
```

影響を受けるフラグ

CF、OF、SF、ZF、AF、およびPFフラグは未定義。

保護モード例外

#DE	ソース・オペランド（除数）が0である場合。
	デスティネーションに対して符号付きの結果(商)が大きすぎる場合。
#GP(0)	メモリ・オペランドの実効アドレスがCS、DS、ES、FS、またはGSセグメントの範囲外の場合。
	DS、ES、FS、またはGSレジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ･オペランドの実効アドレスがSSセグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが3のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

IDIV—Signed Divide（続き）

実アドレスモード例外

#DE	ソース・オペランド（除数）が 0 である場合。 デスティネーションに対して符号付きの結果（商）が大きすぎる場合。
#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#DE	ソース・オペランド（除数）が 0 である場合。 デスティネーションに対して符号付きの結果（商）が大きすぎる場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

IMUL—Signed Multiply

オペコード	命令	説明
F6 /5	IMUL *r/m8*	AX← AL∗ *r/m* バイト
F7 /5	IMUL *r/m16*	DX:AX ← AX ∗ *r/m* ワード
F7 /5	IMUL *r/m32*	EDX:EAX ← EAX ∗ *r/m* ダブルワード
0F AF /*r*	IMUL *r16*, *r/m16*	ワードレジスタ ← ワードレジスタ ∗*r/m* ワード
0F AF /*r*	IMUL *r32*, *r/m32*	ダブルワード・レジスタ ← ダブルワード・レジスタ ∗ *r/m* ダブルワード
6B /*r ib*	IMUL *r16*, *r/m16, imm8*	ワードレジスタ ← *r/m16* ∗ 符号拡張即値バイト
6B /*r ib*	IMUL *r32*, *r/m32, imm8*	ダブルワード・レジスタ ← *r/m32* ∗ 符号拡張即値バイト
6B /*r ib*	IMUL *r16*, *imm8*	ワードレジスタ ← ワードレジスタ ∗ 符号拡張即値バイト
6B /*r ib*	IMUL *r32*, *imm8*	ダブルワード・レジスタ ← ダブルワード・レジスタ ∗ 符号拡張即値バイト
69 /*r iw*	IMUL *r16*, *r/m16*, *imm16*	ワードレジスタ ← *r/m16* ∗ 即値ワード
69 /*r id*	IMUL *r32*, *r/m32*, *imm32*	ダブルワード・レジスタ ← *r/m32* ∗ 即値ダブルワード
69 /*r iw*	IMUL *r16*, *imm16*	ワードレジスタ ← *r/m16* ∗ 即値ワード
69 /*r id*	IMUL *r32*, *imm32*	ダブルワード・レジスタ ← *r/m32* ∗ 即値ダブルワード

説明

2 つのオペランドの符号付き乗算を行う。この命令には、オペランドの数に応じて下記の 3 つの形式がある。

- **1 オペランド形式**: この形式は、MUL 命令に使用されている形式と同じである。この形式では、（汎用レジスタまたはメモリ・ロケーション内の）ソース・オペランドに（オペランド・サイズに応じて）AL、AX、または EAX レジスタの値が掛けられ、結果がそれぞれ AX、DX:AX、または EDX:EAX レジスタにストアされる。
- **2 オペランド形式**: この形式では、デスティネーション・オペランド（第 1 オペランド）にソース・オペランド（第 2 オペランド）が掛けられる。デスティネーション・オペランドは汎用レジスタであり、ソース・オペランドは即値、汎用レジスタ、またはメモリ・ロケーションである。乗算後、結果はデスティネーション・オペランド・ロケーションにストアされる。
- **3 オペランド形式**: この形式には、デスティネーション・オペランド（第 1 オペランド）と 2 つのソース・オペランド（第 2 および第 3 オペランド）が必要である。この形式では、第 1 のソース・オペランド（汎用レジスタまたはメモリ・ロケーション）に第 2 のソース・オペランド（即値）が掛けられる。結果はデスティネーション・オペランド（汎用レジスタ）にストアされる。

即値は、オペランドとして使用されると、デスティネーション・オペランドのフォーマットの長さに符号拡張される。

IMUL—Signed Multiply（続き）

結果の上位半分への有効ビット（サインビットを含む）のキャリーがあったときは、CF および OF フラグがセットされる。結果（サインビットを含む）が結果の下位半分に収まったときは、CF および OF フラグがクリアされる。

IMUL 命令の 3 つの形式は、積の長さが両オペランドの長さの 2 倍に計算される点ではみな同じである。1 オペランド形式では、結果はデスティネーションの長さでストアされる。しかし、2 および 3 オペランド形式では、結果はデスティネーションの長さに切り捨てられてからデスティネーション・レジスタにストアされる。この切り捨てがあるので、CF または OF フラグをテストして、有効ビットの脱落がないよう保証する必要がある。

2 および 3 オペランド形式には、積の下位半分は両オペランドの符号のありなしに関係なく同じなので、符号なしオペランドに対しても使用できる。ただし、CF および OF フラグを使用して、結果の上位半分が非ゼロであるかどうかを判定することはできない。

操作

```
IF (NumberOfOperands = 1)
    THEN IF (OperandSize = 8)
        THEN
            AX ← AL ∗ SRC  (* signed multiplication *)
            IF AL = AX
                THEN CF ← 0; OF ← 0;
                ELSE CF ← 1; OF ← 1;
            FI;
        ELSE IF OperandSize = 16
            THEN
                DX:AX ← AX ∗ SRC  (* signed multiplication *)
                IF sign_extend_to_32 (AX) = DX:AX
                    THEN CF ← 0; OF ← 0;
                    ELSE CF ← 1; OF ← 1;
                FI;
            ELSE (* OperandSize = 32 *)
                EDX:EAX ← EAX ∗ SRC  (* signed multiplication *)
                IF EAX = EDX:EAX
                    THEN CF ← 0; OF ← 0;
                    ELSE CF ← 1; OF ← 1;
                FI;
        FI;
```

IMUL—Signed Multiply（続き）

```
    ELSE IF (NumberOfOperands = 2)
        THEN
            temp ← DEST ∗ SRC    (* signed multiplication; temp is double DEST size*)
            DEST ← DEST ∗ SRC  (* signed multiplication *)
            IF temp ≠ DEST
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0;
            FI;

        ELSE (* NumberOfOperands = 3 *)
            DEST ← SRC1 ∗ SRC2   (* signed multiplication *)
            temp ← SRC1 ∗ SRC2    (* signed multiplication; temp is double SRC1 size *)
            IF temp ≠ DEST
                THEN CF ← 1; OF ← 1;
                ELSE CF ← 0; OF ← 0;
            FI;
    FI;
FI;
```

影響を受けるフラグ

この命令の 1 オペランド形式では、結果の上位半分への有効ビットのキャリーがあるときにCFおよびOFフラグがセットされ、結果が結果の下位半分に収まるときはクリアされる。命令の2および3オペランド形式では、デスティネーション・オペランド・サイズに合わせるために結果を切り捨てなければならないときはCFおよびOFフラグがセットされ、結果がデスティネーション・オペランド・サイズに収まるときはクリアされる。SF、ZF、AF、PF フラグは未定義。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

IMUL—Signed Multiply（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

IN—Input from Port

オペコード	命令	説明
E4 *ib*	IN AL, *imm8*	I/O ポートアドレス *imm8* から AL にバイトを入力する。
E5 *ib*	IN AX, *imm8*	I/O ポートアドレス *imm8* から AX にバイトを入力する。
E5 *ib*	IN EAX, *imm8*	I/O ポートアドレス *imm8* から EAX にバイトを入力する。
EC	IN AL, DX	DX 内の I/O ポートから AL にバイトを入力する。
ED	IN AX, DX	DX 内の I/O ポートから AX にワードを入力する。
ED	IN EAX, DX	DX 内の I/O ポートから EAX にダブルワードを入力する。

説明

値をソース・オペランド（第2オペランド）で指定されたI/Oポートからデスティネーション・オペランド（第1オペランド）にコピーする。ソース・オペランドには、バイト即値またはDXレジスタを使用できる。デスティネーション・オペランドには、アクセスされるポートのサイズ（8、16、または32ビット）に応じて、それぞれAL、AX、またはEAXレジスタを使用できる。ソース・オペランドとしてDXレジスタを使用すると、I/Oポートアドレス0～65,535をアクセスすることができる。バイト即値を使用すると、I/Oポートアドレス0～255をアクセスすることができる。

アクセスされるI/Oポートのサイズは、8ビットのI/Oポートではオペコードによって決まり、16ビットおよび32ビットのI/Oポートでは命令のオペランド・サイズ属性によって決まる。

マシン・コード・レベルでは、I/O命令は、8ビットのI/Oポートをアクセスするときは短くなる。この場合、ポートアドレスの上位8ビットは0になる。

この命令は、プロセッサのI/Oアドレス空間にあるI/Oポートのアクセスだけに有用である。I/Oアドレス空間にあるI/Oポートのアクセス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』の第13章「入出力」を参照のこと。

操作

```
IF ((PE = 1) AND ((CPL > IOPL) OR (VM = 1)))
    THEN (* Protected mode with CPL > IOPL or virtual-8086 mode *)
        IF (Any I/O Permission Bit for I/O port being accessed = 1)
            THEN (* I/O operation is not allowed *)
                #GP(0);
            ELSE ( * I/O operation is allowed *)
                DEST ← SRC; (* Reads from selected I/O port *)
        FI;
    ELSE (Real Mode or Protected Mode with CPL ≤ IOPL *)
        DEST ← SRC; (* Reads from selected I/O port *)
FI;
```

IN—Input from Port（続き）

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	CPL が I/O 特権レベル（IOPL）より大きく（低い特権をもつ）、アクセスされる I/O ポートの TSS にある対応する I/O パーミッション・ビットのいずれかが 1 である場合。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	アクセスされる I/O ポートの TSS にある対応する I/O パーミッション・ビットのいずれかが 1 である場合。

INC—Increment by 1

オペコード	命令	説明
FE /0	INC *r/m8*	*r/m* バイトを 1 インクリメントする。
FF /0	INC *r/m16*	*r/m* ワードを 1 インクリメントする。
FF /0	INC *r/m32*	*r/m* ダブルワードを 1 インクリメントする。
40+ *rw*	INC *r16*	ワードレジスタを 1 インクリメントする。
40+ *rd*	INC *r32*	ダブルワード・レジスタを 1 インクリメントする。

説明

CF フラグの状態を変えないで、デスティネーション・オペランドに 1 を加える。デスティネーション・オペランドには、レジスタまたはメモリ・ロケーションを使用できる。この命令では、CF フラグの状態を変えることなくループカウンタを更新できる。（CF フラグを更新するインクリメント操作を行うには、値 1 の即値オペランドを使用して ADD 命令を実行する。）

この命令を LOCK プリフィックスと共に使用すると、アトミックに命令を実行させることができる。

操作

DEST ← DEST +1;

影響を受けるフラグ

CF フラグは影響を受けない。OF、SF、ZF、AF、および PF フラグが結果にしたがってセットされる。

保護モード例外

#GP(0)	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

INC—Increment by 1（続き）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

INS/INSB/INSW/INSD—Input from Port to String

オペコード	命令	説明
6C	INS m8, DX	バイトを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。
6D	INS m16, DX	ワードを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。
6D	INS m32, DX	ダブルワードを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。
6C	INSB	バイトを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。
6D	INSW	ワードを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。
6D	INSD	ダブルワードを DX で指定された I/O ポートから ES:(E)DI で指定されたメモリ・ロケーションに入力する。

説明

データをソース・オペランド（第2オペランド）で指定された I/O ポートからデスティネーション・オペランド（第1オペランド）にコピーする。ソース・オペランドは I/O ポートアドレス（0 ～ 65,535）であり、そのアドレスは DX レジスタから読み取られる。デスティネーション・オペランドはメモリ・ロケーションであり、そのアドレスは（32 ビットまたは 16 ビットの命令のアドレスサイズ属性に応じて）ES:(E)DI または ES:DI レジスタから読み取られる。ES セグメントはセグメント・オーバーライド・プリフィックスでオーバーライドすることはできない。アクセスされる I/O ポートのサイズ（すなわち、ソース・オペランドおよびデスティネーション・オペランドのサイズ）は、8 ビットの I/O ポートではオペコードによって決まり、16 ビットまたは 32 ビットの I/O ポートでは命令のオペランド・サイズ属性によって決まる。

アセンブリ・コード・レベルでは、この命令の「明示オペランド」形式と「オペランドなし」形式という 2 つの形式が使用できる。（INS ニーモニックで指定される）明示オペランド形式 では、ソース・オペランドおよびデスティネーション・オペランドを明示的に指定できる。この場合、ソース・オペランドは、"DX" でなければならない。デスティネーション・オペランドは、I/O ポートのサイズとデスティネーション・アドレスを示す記号でなければならない。この明示オペランド形式は、ドキュメンテーションを可能にするために設けられたものであるが、この形式によって提供されるドキュメンテーションは誤解を招く場合があるので注意する。すなわち、デスティネーション・オペランドの記号はオペランドの正しい**タイプ**（サイズ：バイト、ワード、またはダブルワード）を指定しなければならないが、正しい**ロケーション**を指定する必要はない。ロケーションは、常に ES:(E)DI レジスタによって指定されるので、INS 命令を実行する前に、これらのレジスタに正しくロードされていなければならない。

INS/INSB/INSW/INSD—Input from Port to String（続き）

オペランドなし形式は、INS 命令のバイト、ワード、およびダブルワード各バージョンの「ショート形式」を提供する。この場合も、DX レジスタがソース・オペランドであると想定され、ES:(E)DI レジスタがデスティネーション・オペランドであると想定される。I/O ポートのサイズは、INSB（バイト）、INSW（ワード）、または INSD（ダブルワード）の各ニーモニックの選択で指定される。

バイト、ワード、またはダブルワードが I/O ポートからメモリ・ロケーションに転送された後、(E)DI レジスタは EFLAGS レジスタ内の EF フラグの設定にしたがって自動的にインクリメントまたはデクリメントされる。（DF フラグが 0 である場合、(E)DI レジスタはインクリメントされる。DF フラグが 1 である場合、(E)DI レジスタはデクリメントされる。）(E)DI レジスタは、バイト操作の場合は 1、ワード操作の場合は 2、ダブルワード操作の場合は 4、それぞれインクリメントまたはデクリメントされる。

INS、INSB、INSW、INSD 命令は、前に REP プリフィックスを付けることにより、ECX バイト、ワード、またはダブルワードのブロック入力を行うことができる。REP プリフィックスの説明については、本章の「REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix」を参照のこと。

これらの命令は、プロセッサの I/O アドレス空間にある I/O ポートのアクセスだけに有用である。I/O アドレス空間にある I/O ポートへのアクセス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』の第 13 章「入出力」を参照のこと。

操作

```
IF ((PE = 1) AND ((CPL > IOPL) OR (VM = 1)))
    THEN (* Protected mode with CPL > IOPL or virtual-8086 mode *)
        IF (Any I/O Permission Bit for I/O port being accessed ← 1)
            THEN (* I/O operation is not allowed *)
                #GP(0);
            ELSE ( * I/O operation is allowed *)
                DEST ← SRC; (* Reads from I/O port *)
        FI;
    ELSE (Real Mode or Protected Mode with CPL ≤ IOPL *)
        DEST ← SRC; (* Reads from I/O port *)
FI;
IF (byte transfer)
    THEN IF DF ← 0
        THEN (E)DI ← (E)DI + 1;
        ELSE (E)DI ← (E)DI – 1;
    FI;
```

INS/INSB/INSW/INSD—Input from Port to String（続き）

```
    ELSE IF (word transfer)
        THEN IF DF ← 0
            THEN (E)DI ← (E)DI + 2;
            ELSE (E)DI ← (E)DI – 2;
        FI;
        ELSE (* doubleword transfer *)
            THEN IF DF ← 0
                THEN (E)DI ← (E)DI + 4;
                ELSE (E)DI ← (E)DI – 4;
            FI;
    FI;
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	CPL が I/O 特権レベル（IOPL）より大きく（低い特権をもつ）、アクセスされる I/O ポートの TSS にある対応する I/O パーミッション・ビットのいずれかが 1 である場合。
	デスティネーションが書き込み不可能なセグメントにある場合。
	ES セグメントにイリーガルなメモリ・オペランドの実効アドレスが指定された場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	アクセスされる I/O ポートの TSS にある対応する I/O パーミッション・ビットのいずれかが 1 である場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

INT *n*/INTO/INT 3—Call to Interrupt Procedure

オペコード	命令	説明
CC	INT 3	割り込み 3 － デバッガへのトラップ。
CD *ib*	INT *imm8*	割り込みベクタ番号の即値バイトによる指定。
CE	INTO	割り込み 4 － オーバーフロー・フラグが 1 である場合。

説明

INT *n* 命令は、デスティネーション・オペランドで指定された割り込みハンドラまたは例外ハンドラへのコールを生成する（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、上巻』の第 6 章の「割り込みと例外」の説明を参照）。デスティネーション・オペランドは、8 ビットの符号なし中間値としてコード化された 0 から 255 までの割り込みベクタ番号を指定する。各割り込みベクタ番号は、IDT 内の特定のゲート・ディスクリプタへのインデックスになる。最初の 32 個の割り込みベクタ番号はシステム用に予約されている。これらの割り込みの一部は、内部的に生成される例外に使用される。

INT *n* 命令は、ソフトウェアによって生成される割り込みハンドラへのコールを実行するための一般的ニーモニックである。INTO 命令は、オーバーフロー例外（#OF）、すなわち割り込みベクタ番号 4 をコールするための特殊ニーモニックである。オーバーフロー割り込みは EFLAGS レジスタ内の OF フラグをチェックし、それが 1 にセットされている場合にオーバーフロー例外ハンドラをコールする。

INT 3 命令は、デバッグ例外ハンドラをコールすることを目的とする特別な 1 バイト・オペコード（CC）を生成する。（この 1 バイト形式は、それを使用すると他のコードを上書きすることなく、他の 1 バイト命令を含めて任意の命令の最初のバイトをブレークポイントで置き換えることができるので、貴重である。）デバッグ・ブレークポイントとしてのその機能をさらにサポートするため、CC オペコードで生成される割り込みも、下記の点で正規のソフトウェア割り込みとは異なっている。

- VME　モードでは割り込みのリダイレクションが行われない。割り込みは、保護モードハンドラによって処理される。
- 仮想 8086 モードの IOPL チェックが行われない。割り込みは、IOPL レベルでのフォルトを一切伴わないで取り扱われる。

INT 3 の「通常」の 2 バイト・オペコード（CD03）には、これらの特別な機能はないので注意する。インテルおよび Microsoft* アセンブラは、どのニーモニックからも CD03 オペコードを生成しない。その代わりに、このオペコードは直接数値コード定義または自己修正コードによって作成できる。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

（INTO およびINT 3 命令を含む）INT *n* 命令の処理は、CALL 命令によって行われる far コールの処理に似ている。最も大きな相違は、INT *n* 命令では、EFLAGS レジスタがリターンアドレスより先にスタックにプッシュされる点である。（リターンアドレスは、CS およびEIP レジスタの現在値からなる far アドレスである。）割り込みプロシージャからのリターンはIRET 命令で処理される。IRET 命令はスタックからEFLAGS 情報とリターンアドレスをポップする。

割り込みベクタ番号は、割り込みディスクリプタ・テーブル（IDT）内の特定の割り込みディスクリプタを指定する。すなわち、この番号はIDT へのインデックスの役割を果たす。それに対し、選択された割り込みディスクリプタの内容は割り込みまたは例外ハンドラ・プロシージャへのポインタになっている。保護モードでは、IDT は、それぞれが割り込みゲート、トラップゲート、またはタスクゲートである 8 バイトのディスクリプタの配列になっている。実アドレスモードでは、IDT は、それぞれ選択されたセグメント内のプロシージャを直接指示先とする 4 バイトの far ポインタ（2 バイトのコードセレクタと 2 バイトの命令ポインタ）の配列である。（実アドレスモードでは、IDT は**割り込みベクタテーブル**と呼ばれ、そのポインタがコール先割り込みベクタと呼ばれる。）

以下のデシジョン・テーブルは、テーブルの太線より上の部分の条件が与えられた場合に、太線より下の部分のどの処置が行われるかを示している。デシジョン・テーブルの下の部分の各 Y は、この命令の「操作」の項に定義されているプロシージャ（#GP を除く）を表している。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

PE	0	1	1	1	1	1	1	1
VM	–	–	–	–	–	0	1	1
IOPL	–	–	–	–	–	–	<3	=3
DPL/CPL の関係	–	DPL< CPL	–	DPL> CPL	DPL= CPL or C	DPL< CPL & NC	–	–
割り込みタイプ	–	S/W	–	–	–	–	–	–
ゲートタイプ	–	–	タスク	トラップまたは割り込み	トラップまたは割り込み	トラップまたは割り込み	トラップまたは割り込み	トラップまたは割り込み
REAL-ADDRESS-MODE	Y							
PROTECTED-MODE		Y	Y	Y	Y	Y	Y	Y
TRAP-OR-INTERRUPT-GATE				Y	Y	Y	Y	Y
INTER-PRIVILEGE-LEVEL-INTERRUPT						Y		
INTRA-PRIVILEGE-LEVEL-INTERRUPT					Y			
INTERRUPT-FROM-VIRTUAL-8086-MODE								Y
TASK-GATE			Y					
#GP		Y		Y			Y	

注：

– 無指定。

Y Yes）行われる処置。

空白 処置は行われない。

プロセッサが仮想 8086 モードで動作しているときには、IOPL によって INT *n* 命令の処置が決まる。IOPL が 3 より小さい場合は、プロセッサは一般保護例外（#GP）を生成する。IOPL が 3 の場合は、プロセッサは特権レベル 0 への保護モード割り込みを実行する。特権レベル 0 への保護モード割り込みを実行するためには、割り込みゲートの DPL が 3 に設定されていて、かつ割り込みハンドラ・プロシージャのターゲット CPL が 0 でなければならない。

割り込みディスクリプタ・テーブル・レジスタ（IDTR）には、IDT のベース・リニア・アドレスおよび範囲を指定する。プロセッサの電源投入後またはリセット後の IDTR の初期ベースアドレス値は 0 である。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

操作

以下の操作の記述は、INT *n* 命令および INTO 命令だけでなく、外部割り込みおよび例外にも適用される。

```
IF PE = 0
    THEN
        GOTO REAL-ADDRESS-MODE;
    ELSE (* PE = 1 *)
        IF (VM = 1 AND IOPL < 3 AND INT n)
            THEN
                #GP(0);
            ELSE (* protected mode or virtual-8086 mode interrupt *)
                GOTO PROTECTED-MODE;
        FI;
FI;

REAL-ADDRESS-MODE:
    IF ((DEST ∗ 4) + 3) is not within IDT limit THEN #GP; FI;
    IF stack not large enough for a 6-byte return information THEN #SS; FI;
    Push (EFLAGS[15:0]);
    IF ← 0; (* Clear interrupt flag *)
    TF ← 0; (* Clear trap flag *)
    AC ← 0; (*Clear AC flag*)
    Push(CS);
    Push(IP);
    (* No error codes are pushed *)
    CS ← IDT(Descriptor (vector_number ∗ 4), selector));
    EIP ← IDT(Descriptor (vector_number ∗ 4), offset)); (* 16 bit offset AND 0000FFFFH *)
END;

PROTECTED-MODE:
    IF ((DEST ∗ 8) + 7) is not within IDT limits
        OR selected IDT descriptor is not an interrupt-, trap-, or task-gate type
            THEN #GP((DEST ∗ 8) + 2 + EXT);
            (* EXT is bit 0 in error code *)
    FI;
    IF software interrupt (* generated by INT n, INT 3, or INTO *)
        THEN
            IF gate descriptor DPL < CPL
                THEN #GP((vector_number ∗ 8) + 2 );
                (* PE = 1, DPL<CPL, software interrupt *)
            FI;
    FI;
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
    IF gate not present THEN #NP((vector_number * 8) + 2 + EXT); FI;
    IF task gate (* specified in the selected interrupt table descriptor *)
        THEN GOTO TASK-GATE;
        ELSE GOTO TRAP-OR-INTERRUPT-GATE; (* PE = 1, trap/interrupt gate *)
    FI;
END;

TASK-GATE: (* PE = 1, task gate *)
    Read segment selector in task gate (IDT descriptor);
        IF local/global bit is set to local
            OR index not within GDT limits
                THEN #GP(TSS selector);
        FI;
        Access TSS descriptor in GDT;
        IF TSS descriptor specifies that the TSS is busy (low-order 5 bits set to 00001)
            THEN #GP(TSS selector);
        FI;
        IF TSS not present
            THEN #NP(TSS selector);
        FI;
    SWITCH-TASKS (with nesting) to TSS;
    IF interrupt caused by fault with error code
        THEN
            IF stack limit does not allow push of error code
                THEN #SS(0);
            FI;
            Push(error code);
    FI;
    IF EIP not within code segment limit
        THEN #GP(0);
    FI;
END;
TRAP-OR-INTERRUPT-GATE
    Read segment selector for trap or interrupt gate (IDT descriptor);
    IF segment selector for code segment is null
        THEN #GP(0H + EXT); (* null selector with EXT flag set *)
    FI;
    IF segment selector is not within its descriptor table limits
        THEN #GP(selector + EXT);
    FI;
    Read trap or interrupt handler descriptor;
    IF descriptor does not indicate a code segment
        OR code segment descriptor DPL > CPL
            THEN #GP(selector + EXT);
    FI;
    IF trap or interrupt gate segment is not present,
        THEN #NP(selector + EXT);
    FI;
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
        IF code segment is non-conforming AND DPL < CPL
            THEN IF VM=0
                THEN
                    GOTO INTER-PRIVILEGE-LEVEL-INTERRUPT;
                    (* PE = 1, interrupt or trap gate, nonconforming *)
                    (* code segment, DPL<CPL, VM = 0 *)
                ELSE (* VM = 1 *)
                    IF code segment DPL ≠ 0 THEN #GP(new code segment selector); FI;
                    GOTO INTERRUPT-FROM-VIRTUAL-8086-MODE;
                    (* PE = 1, interrupt or trap gate, DPL<CPL, VM = 1 *)
            FI;
            ELSE (* PE = 1, interrupt or trap gate, DPL ≥ CPL *)
                IF VM = 1 THEN #GP(new code segment selector); FI;
                IF code segment is conforming OR code segment DPL = CPL
                    THEN
                        GOTO INTRA-PRIVILEGE-LEVEL-INTERRUPT;
                    ELSE
                        #GP(CodeSegmentSelector + EXT);
                        (* PE = 1, interrupt or trap gate, nonconforming *)
                        (* code segment, DPL>CPL *)
                FI;
        FI;
    END;

    INTER-PRIVILEGE-LEVEL-INTERRUPT
        (* PE=1, interrupt or trap gate, non-conforming code segment, DPL<CPL *)
        (* Check segment selector and descriptor for stack of new privilege level in current
        TSS *)
        IF current TSS is 32-bit TSS
            THEN
                TSSstackAddress ← (new code segment DPL * 8) + 4
                IF (TSSstackAddress + 7) > TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS ← TSSstackAddress + 4;
                NewESP ← stack address;
            ELSE (* TSS is 16-bit *)
                TSSstackAddress ← (new code segment DPL * 4) + 2
                IF (TSSstackAddress + 4) > TSS limit
                    THEN #TS(current TSS selector); FI;
                NewESP ← TSSstackAddress;
                NewSS ← TSSstackAddress + 2;
        FI;
        IF segment selector is null THEN #TS(EXT); FI;
        IF segment selector index is not within its descriptor table limits
            OR segment selector's RPL ≠ DPL of code segment,
                THEN #TS(SS selector + EXT);
        FI;
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
Read segment descriptor for stack segment in GDT or LDT;
    IF stack segment DPL ≠ DPL of code segment,
        OR stack segment does not indicate writable data segment,
            THEN #TS(SS selector + EXT);
    FI;
    IF stack segment not present THEN #SS(SS selector+EXT); FI;
    IF 32-bit gate
        THEN
            IF new stack does not have room for 24 bytes (error code pushed)
                OR 20 bytes (no error code pushed)
                    THEN #SS(segment selector + EXT);
            FI;
        ELSE (* 16-bit gate *)
            IF new stack does not have room for 12 bytes (error code pushed)
                OR 10 bytes (no error code pushed);
                    THEN #SS(segment selector + EXT);
            FI;
    FI;
    IF instruction pointer is not within code segment limits THEN #GP(0); FI;
    SS:ESP ← TSS(NewSS:NewESP) (* segment descriptor information also loaded *)
    IF 32-bit gate
        THEN
            CS:EIP ← Gate(CS:EIP); (* segment descriptor information also loaded *)
        ELSE (* 16-bit gate *)
            CS:IP ← Gate(CS:IP); (* segment descriptor information also loaded *)
    FI;
    IF 32-bit gate
        THEN
            Push(far pointer to old stack); (* old SS and ESP, 3 words padded to 4 *);
            Push(EFLAGS);
            Push(far pointer to return instruction); (* old CS and EIP, 3 words padded to 4*);
            Push(ErrorCode); (* if needed, 4 bytes *)
        ELSE(* 16-bit gate *)
            Push(far pointer to old stack); (* old SS and SP, 2 words *);
            Push(EFLAGS(15..0]);
            Push(far pointer to return instruction); (* old CS and IP, 2 words *);
            Push(ErrorCode); (* if needed, 2 bytes *)
    FI;
    CPL ← CodeSegmentDescriptor(DPL);
    CS(RPL) ← CPL;
    IF interrupt gate
        THEN IF ← 0 (*interrupt flag set to 0: disabled*);
    FI;
    TF ← 0;
    VM ← 0;
    RF ← 0;
    NT ← 0;
END;
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
INTERRUPT-FROM-VIRTUAL-8086-MODE:
    (* Check segment selector and descriptor for privilege level 0 stack in current TSS *)
    IF current TSS is 32-bit TSS
        THEN
            TSSstackAddress ← (new code segment DPL * 8) + 4
            IF (TSSstackAddress + 7) > TSS limit
                THEN #TS(current TSS selector); FI;
            NewSS ← TSSstackAddress + 4;
            NewESP ← stack address;
        ELSE (* TSS is 16-bit *)
            TSSstackAddress ← (new code segment DPL * 4) + 2
            IF (TSSstackAddress + 4) > TSS limit
                THEN #TS(current TSS selector); FI;
            NewESP ← TSSstackAddress;
            NewSS ← TSSstackAddress + 2;
    FI;
        IF segment selector is null THEN #TS(EXT); FI;
        IF segment selector index is not within its descriptor table limits
            OR segment selector's RPL ≠ DPL of code segment,
                THEN #TS(SS selector + EXT);
        FI;
    Access segment descriptor for stack segment in GDT or LDT;
    IF stack segment DPL ≠ DPL of code segment,
        OR stack segment does not indicate writable data segment,
            THEN #TS(SS selector + EXT);
    FI;
    IF stack segment not present
        THEN #SS(SS selector+EXT);
    FI;
    IF 32-bit gate
        THEN
            IF new stack does not have room for 40 bytes (error code pushed)
                OR 36 bytes (no error code pushed);
                    THEN #SS(segment selector + EXT);
            FI;
        ELSE (* 16-bit gate *)
            IF new stack does not have room for 20 bytes (error code pushed)
                OR 18 bytes (no error code pushed);
                    THEN #SS(segment selector + EXT);
            FI;
    FI;
    IF instruction pointer is not within code segment limits
        THEN #GP(0);
    FI;
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
    tempEFLAGS ← EFLAGS;
    VM ← 0;
    TF ← 0;
    RF ← 0;
    IF service through interrupt gate
        THEN IF = 0;
    FI;
    TempSS ← SS;
    TempESP ← ESP;
    SS:ESP ← TSS(SS0:ESP0); (* Change to level 0 stack segment *)
    (* Following pushes are 16 bits for 16-bit gate and 32 bits for 32-bit gates *)
    (* Segment selector pushes in 32-bit mode are padded to two words *)
    Push(GS);
    Push(FS);
    Push(DS);
    Push(ES);
    Push(TempSS);
    Push(TempESP);
    Push(TempEFlags);
    Push(CS);
    Push(EIP);
    GS ← 0; (*segment registers nullified, invalid in protected mode *)
    FS ← 0;
    DS ← 0;
    ES ← 0;
    CS ← Gate(CS);
    IF OperandSize = 32
        THEN
            EIP ← Gate(instruction pointer);
        ELSE (* OperandSize is 16 *)
            EIP ← Gate(instruction pointer) AND 0000FFFFH;
    FI;
    (* Starts execution of new routine in Protected Mode *)
END;

INTRA-PRIVILEGE-LEVEL-INTERRUPT:
    (* PE=1, DPL = CPL or conforming segment *)
    IF 32-bit gate
        THEN
            IF current stack does not have room for 16 bytes (error code pushed)
                OR 12 bytes (no error code pushed); THEN #SS(0);
            FI;
        ELSE (* 16-bit gate *)
            IF current stack does not have room for 8 bytes (error code pushed)
                OR 6 bytes (no error code pushed); THEN #SS(0);
            FI;
    FI;
    IF instruction pointer not within code segment limit
        THEN #GP(0);
```

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

```
    FI;
    IF 32-bit gate
        THEN
            Push (EFLAGS);
            Push (far pointer to return instruction); (* 3 words padded to 4 *)
            CS:EIP ← Gate(CS:EIP); (* segment descriptor information also loaded *)
            Push (ErrorCode); (* if any *)
        ELSE (* 16-bit gate *)
            Push (FLAGS);
            Push (far pointer to return location); (* 2 words *)
            CS:IP ← Gate(CS:IP); (* segment descriptor information also loaded *)
            Push (ErrorCode); (* if any *)
    FI;
    CS(RPL) ← CPL;
    IF interrupt gate
        THEN IF ← 0; (*interrupt flag set to 0: disabled*)
    FI;
    TF ← 0;
    NT ← 0;
    VM ← 0;
    RF ← 0;
END;
```

影響を受けるフラグ

EFLAGS レジスタがスタックにプッシュされる。INT 命令が実行されるときのプロセッサの動作モードに応じて、IF、TF、NT、AC、RF、VM フラグがクリアされることがある（「操作」の項を参照）。割り込みがタスクゲートを使用する場合は、新しいタスクのTSS内のEFLAGSイメージによる制御のもとに、任意のフラグがセットまたはクリアされることがある。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

保護モード例外

#GP(0)	IDT、割り込みゲート、トラップゲート、またはタスクゲート内の命令ポインタがコード・セグメントの範囲を超えている場合。
#GP（セレクタ）	割り込みゲート、トラップゲート、またはタスクゲート内のセグメント・セレクタがヌルの場合。
	割り込みゲート、トラップゲート、タスクゲート、コード・セグメント、または TSS のセグメント・セレクタ・インデックスがそのディスクリプタ・テーブルの範囲外の場合。
	割り込みベクタ番号が IDT の範囲外の場合。
	IDT ディスクリプタが、割り込みディスクリプタ、トラップ・ディスクリプタ、またはタスク・ディスクリプタのいずれでもない場合。
	INT *n*、INT 3、または INTO 命令によって割り込みが発生し、割り込みディスクリプタ、トラップ・ディスクリプタ、またはタスク・ディスクリプタの DPL が CPL より小さい場合。
	割り込みゲートまたはトラップゲート内のセグメント・セレクタの指示先がコード・セグメントのセグメント・ディスクリプタでない場合。
	TSS のセグメント・セレクタのローカル / グローバル・ビットがローカルに設定されている場合。
	TSS のセグメント・ディスクリプタが、その TSS がビジーであるか使用不可能であると指定している場合。
#SS(0)	リターンアドレス、フラグ、またはエラーコードをスタックにプッシュして、スタック・セグメントの範囲を超えたが、スタックスイッチが行われなかった場合。
#SS（セレクタ）	SS レジスタがロードされようとしたとき、指示先のセグメントが存在しないとマークされていた場合。
	スタックスイッチが発生したとき、リターンアドレス、フラグ、エラーコード、またはスタック・セグメント・ポインタをプッシュして、新しいスタック・セグメントの範囲を超えた場合。
#NP（セレクタ）	コード・セグメント、割り込みゲート、トラップゲート、タスクゲート、または TSS が存在しない場合。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

#TS（セレクタ）	TSS 内のスタック・セグメント・セレクタの RPL が、割り込みゲートまたはトラップゲートによってアクセスされるコード･セグメントの DPL に等しくない場合。
	TSS 内のスタック・セグメント・セレクタによって指されているスタック・セグメント・ディスクリプタの DPL が割り込みゲートまたはトラップゲートのコード・セグメント・ディスクリプタの DPL に等しくない場合。
	TSS 内のスタック・セグメント・セレクタがヌルの場合。
	TSS のスタック・セグメントが書き込み可能なデータ・セグメントでない場合。
	スタック・セグメントのセグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	割り込みベクタ番号が IDT の範囲外の場合。
#SS	プッシュ時スタック範囲違反の場合。
	リターンアドレス、フラグ、またはエラーコードをスタックにプッシュして、スタック・セグメントの範囲を超えた場合。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

仮想 8086 モード例外

#GP(0)	（INT *n*、INTO、または BOUND 命令の場合）IOPL が 3 より小さいか、または割り込みゲート、トラップゲート、またはタスクゲートのディスクリプタの DPL が 3 に等しくない場合。
	IDT、割り込みゲート、トラップゲート、またはタスクゲート内の命令ポインタがコード・セグメントの範囲を超えている場合。
#GP（セレクタ）	割り込みゲート、トラップゲート、またはタスクゲート内のセグメント・セレクタがヌルの場合。
	割り込みゲート、トラップゲート、タスクゲート、コード・セグメント、または TSS のセグメント・セレクタ・インデックスがそのディスクリプタ・テーブルの範囲外の場合。
	割り込みベクタ番号が IDT の範囲外の場合。
	IDT ディスクリプタが、割り込みディスクリプタ、トラップ・ディスクリプタ、またはタスク・ディスクリプタのいずれでもない場合。
	INT *n* 命令によって割り込みが発生し、割り込みディスクリプタ、トラップ・ディスクリプタ、またはタスク・ディスクリプタの DPL が CPL より小さい場合。
	割り込みゲートまたはトラップゲート内のセグメント・セレクタの指示先がコード・セグメントのセグメント・ディスクリプタでない場合。
	TSS のセグメント・セレクタのローカル / グローバル・ビットがローカルに設定されている場合。
#SS（セレクタ）	SS レジスタがロードされようとしたとき、指示先のセグメントが存在しないとマークされていた場合。
	リターンアドレス、フラグ、エラーコード、スタック・セグメント・ポインタ、またはデータ・セグメントをプッシュして、スタック・セグメントの範囲を超えた場合。
#NP（セレクタ）	コード・セグメント、割り込みゲート、トラップゲート、タスクゲート、または TSS が存在しない場合。

INT *n*/INTO/INT 3—Call to Interrupt Procedure（続き）

#TS（セレクタ）	TSS 内のスタック・セグメント・セレクタの RPL が、割り込みゲートまたはトラップゲートによってアクセスされるコード･セグメントの DPL に等しくない場合
	TSS のスタック・セグメントに対するスタック・セグメント・ディスクリプタの DPL が､割り込みゲートまたはトラップゲートのコード･セグメント・ディスクリプタの DPL に等しくない場合。
	TSS 内のスタック・セグメント・セレクタがヌルの場合。
	TSS のスタック・セグメントが書き込み可能なデータ・セグメントでない場合。
	スタック・セグメントのセグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#BP	INT 3 命令が実行された場合。
#OF	INTO 命令が実行され、OF フラグがセットされた場合。

INVD—Invalidate Internal Caches

オペコード	命令	説明
0F 08	INVD	内部キャッシュをフラッシュする。外部キャッシュのフラッシュを開始させる。

説明

プロセッサの内部キャッシュを無効化（フラッシュ）し、外部キャッシュにも自身をフラッシュするよう指示する特殊機能バスサイクルを発行する。内部キャッシュに保持されていたデータはメインメモリにライトバックされない。

この命令を実行した後、プロセッサは、外部キャッシュのフラッシュ操作の完了を待たずに命令の実行を継続する。キャッシュ・フラッシュ信号への応答は、ハードウェアによって行う。

INVD 命令は特権命令である。プロセッサが保護モードで動作している場合は、この命令を実行するには、プログラムまたはプロシージャの CPL が 0 でなければならない。

この命令を使用する際は注意が必要である。内部でキャッシュされ、メインメモリにライトバックされないデータは失われる。（例えば、メインメモリとのキャッシュ・コヒーレンシが重要でないテストやフォルトリカバリのように）修正したキャッシュ・ラインをライトバックしないでキャッシュをフラッシュする特別な必要性、またはそうすることに特別な利点がなければ、ソフトウェアには WBINVD 命令を使用すべきである。

IA-32 アーキテクチャにおける互換性

INVD 命令はプロセッサに依存し、その機能は異なるファミリの IA-32 プロセッサでは異なってサポートされる可能性もある。この命令は、Intel486™ プロセッサより以前の IA-32 プロセッサではサポートされていない。

操作

```
Flush(InternalCaches);
SignalFlush(ExternalCaches);
Continue (* Continue execution);
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。

INVD—Invalidate Internal Caches（続き）

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	INVD 命令は仮想 8086 モードで実行することはできない。

INVLPG—Invalidate TLB Entry

オペコード	命令	説明
0F 01/7	INVLPG m	m があるページの TLB エントリを無効にする。

説明

ソース・オペランドで指定されたトランスレーション・ルックアサイド・バッファ（TLB）のエントリを無効化（フラッシュ）する。ソース・オペランドは、メモリアドレスである。プロセッサは、そのアドレスが含まれるページを判定し、そのページのTLBエントリをフラッシュする。

INVLPG 命令は特権命令である。プロセッサが保護モードで動作している場合は、この命令を実行するには、プログラムまたはプロシージャのCPLが0でなければならない。

通常、INVLPG命令は指定されたページのTLBエントリだけをフラッシュするが、場合によってはTLB全体をフラッシュすることもある。TLBをフラッシュする操作の詳細については、本章の「MOV—Move to/from Control Registers」を参照のこと。

IA-32 アーキテクチャにおける互換性

INVLPG 命令はプロセッサに依存し、その機能は異なるファミリの IA-32 プロセッサでは異なってサポートされる可能性もある。この命令は、Intel486™ プロセッサより以前のIA-32プロセッサではサポートされていない。

操作

```
Flush(RelevantTLBEntries);
Continue (* Continue execution);
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
#UD	オペランドがレジスタである場合。

実アドレスモード例外

#UD	オペランドがレジスタである場合。

仮想 8086 モード例外

#GP(0)	INVLPG 命令は仮想 8086 モードで実行することはできない。

IRET/IRETD—Interrupt Return

オペコード	命令	説明
CF	IRET	割り込みリターン（オペランド・サイズ : 16 ビット）。
CF	IRETD	割り込みリターン（オペランド・サイズ : 32 ビット）。

説明

プログラム制御を例外ハンドラまたは割り込みハンドラから、例外、外部割り込み、またはソフトウェア生成割り込みによって中断されていたプログラムまたはプロシージャに戻す。これらの命令は、ネストされたタスクからのリターンを行う場合にも使用される。（ネストされたタスクは、CALL 命令を使用してタスクスイッチを開始したとき、あるいは割り込みまたは例外によって割り込みハンドラまたは例外ハンドラへのタスクスイッチが行われたときに作成される。）『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、下巻』の第 6 章の「タスクのリンク」を参照のこと。

IRET および IRETD は同じオペコードに対する 2 つのニーモニックである。IRETD ニーモニック（Interrupt Return Double）は、32 ビットのオペランド・サイズを使用しているときに割り込みから戻る場合に使用することを目的としているが、ほとんどのアセンブラはどちらのオペランド・サイズに対しても IRET ニーモニックを使用している。

実アドレスモードでは、IRET 命令は割り込みによって中断されていたプログラムまたはプロシージャへの far リターンを実行する。この操作では、プロセッサはスタックから EIP、CS、EFLAGS のレジスタにそれぞれリターン命令ポインタ、リターン・コード・セグメント・セレクタ、EFLAGS イメージをポップし、次に中断されていたプログラムまたはプロシージャの実行を再開する。

保護モードでは、IRET 命令の処理は、EFLAGS レジスタ内の NT（ネストされたタスク）および VM フラグの設定と、現在のスタックにストアされている EFLAGS イメージの VM フラグの設定に依存する。これらのフラグの設定に応じて、プロセッサは以下のタイプの割り込みリターンのいずれかを実行する。

- 仮想 8086 モードからのリターン
- 仮想 8086 モードへのリターン
- 特権レベル内のリターン
- 特権レベル間のリターン
- ネストされたタスクからのリターン（タスクスイッチ）

IRET/IRETD—Interrupt Return（続き）

NT フラグ（EFLAGS レジスタ）がクリアされている場合は、IRET 命令はタスクスイッチなしに割り込みプロシージャからの far リターンを実行する。リターン先のコード・セグメントは、特権レベルが（スタックからポップされたコード・セグメント・セレクタの RPL フィールドによって示される）割り込みハンドラルーチンの特権レベル以下でなければならない。実アドレスモードの割り込みリターンの場合と同様に、IRET 命令はリターン命令ポインタ、リターン・コード・セグメント・セレクタ、EFLAGS イメージをスタックからそれぞれ EIP、CS、EFLAGS のレジスタにポップし、次に中断されていたプログラムまたはプロシージャの実行を再開する。別の特権レベルへのリターンの場合は、IRET 命令は、スタックからさらにスタックポインタと SS レジスタもポップしてからプログラムの実行を再開する。仮想 8086 モードへのリターンの場合は、プロセッサはスタックからさらにデータ・セグメント・レジスタもポップする。

NT フラグがセットされている場合は、IRET 命令はネストされたタスク（CALL 命令、割り込み、または例外でコールされたタスク）からコール元または中断されていたタスクへのタスクスイッチ（リターン）を実行する。IRET 命令を実行しているタスクの更新後の状態がその TSS にセーブされる。このタスクが後で再起動される場合は、IRET 命令の次からコードが実行される。

操作

```
IF PE = 0
    THEN
        GOTO REAL-ADDRESS-MODE:;
    ELSE
        GOTO PROTECTED-MODE;
FI;

REAL-ADDRESS-MODE;
    IF OperandSize = 32
        THEN
            IF top 12 bytes of stack not within stack limits THEN #SS; FI;
            IF instruction pointer not within code segment limits THEN #GP(0); FI;
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            tempEFLAGS ← Pop();
            EFLAGS ← (tempEFLAGS AND 257FD5H) OR (EFLAGS AND 1A0000H);
        ELSE (* OperandSize = 16 *)
            IF top 6 bytes of stack are not within stack limits THEN #SS; FI;
            IF instruction pointer not within code segment limits THEN #GP(0); FI;
            EIP ← Pop();
            EIP ← EIP AND 0000FFFFH;
            CS ← Pop(); (* 16-bit pop *)
            EFLAGS[15:0] ← Pop();
```

IRET/IRETD—Interrupt Return（続き）

```
    FI;
END;
PROTECTED-MODE:
    IF VM = 1 (* Virtual-8086 mode: PE=1, VM=1 *)
        THEN
            GOTO RETURN-FROM-VIRTUAL-8086-MODE; (* PE=1, VM=1 *)
    FI;
    IF NT = 1
        THEN
            GOTO TASK-RETURN;( *PE=1, VM=0, NT=1 *)
    FI;
    IF OperandSize=32
        THEN
            IF top 12 bytes of stack not within stack limits
                THEN #SS(0)
            FI;
            tempEIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
        ELSE (* OperandSize = 16 *)
            IF top 6 bytes of stack are not within stack limits
                THEN #SS(0);
            FI;
            tempEIP ← Pop();
            tempCS ← Pop();
            tempEFLAGS ← Pop();
            tempEIP ← tempEIP AND FFFFH;
            tempEFLAGS ← tempEFLAGS AND FFFFH;
    FI;
    IF tempEFLAGS(VM) = 1 AND CPL=0
        THEN
            GOTO RETURN-TO-VIRTUAL-8086-MODE;
            (* PE=1, VM=1 in EFLAGS image *)
        ELSE
            GOTO PROTECTED-MODE-RETURN;
            (* PE=1, VM=0 in EFLAGS image *)
    FI;

RETURN-FROM-VIRTUAL-8086-MODE:
(* Processor is in virtual-8086 mode when IRET is executed and stays in virtual-8086
mode *)
    IF IOPL=3 (* Virtual mode: PE=1, VM=1, IOPL=3 *)
        THEN IF OperandSize = 32
            THEN
                IF top 12 bytes of stack not within stack limits THEN #SS(0); FI;
                IF instruction pointer not within code segment limits THEN #GP(0); FI;
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                EFLAGS ← Pop();
                (*VM,IOPL,VIP,and VIF EFLAGS bits are not modified by pop *)
```

IRET/IRETD—Interrupt Return（続き）

```
            ELSE (* OperandSize = 16 *)
                IF top 6 bytes of stack are not within stack limits THEN #SS(0); FI;
                IF instruction pointer not within code segment limits THEN #GP(0); FI;
                EIP ← Pop();
                EIP ← EIP AND 0000FFFFH;
                CS ← Pop(); (* 16-bit pop *)
                EFLAGS[15:0] ← Pop(); (* IOPL in EFLAGS is not modified by pop *)
            FI;
        ELSE
            #GP(0); (* trap to virtual-8086 monitor: PE=1, VM=1, IOPL<3 *)
    FI;
END;

RETURN-TO-VIRTUAL-8086-MODE:
(* Interrupted procedure was in virtual-8086 mode: PE=1, VM=1 in flags image *)
    IF top 24 bytes of stack are not within stack segment limits
        THEN #SS(0);
    FI;
    IF instruction pointer not within code segment limits
        THEN #GP(0);
    FI;
    CS ← tempCS;
    EIP ← tempEIP;
    EFLAGS ← tempEFLAGS
    TempESP ← Pop();
    TempSS ← Pop();
    ES ← Pop(); (* pop 2 words; throw away high-order word *)
    DS ← Pop(); (* pop 2 words; throw away high-order word *)
    FS ← Pop(); (* pop 2 words; throw away high-order word *)
    GS ← Pop(); (* pop 2 words; throw away high-order word *)
    SS:ESP ← TempSS:TempESP;
    CPL ← 3;
    (* Resume execution in Virtual-8086 mode *)
END;

TASK-RETURN: (* PE=1, VM=1, NT=1 *)
    Read segment selector in link field of current TSS;
    IF local/global bit is set to local
        OR index not within GDT limits
            THEN #TS (TSS selector);
    FI;
    Access TSS for task specified in link field of current TSS;
    IF TSS descriptor type is not TSS or if the TSS is marked not busy
        THEN #TS (TSS selector);
    FI;
    IF TSS not present
        THEN #NP(TSS selector);
    FI;
    SWITCH-TASKS (without nesting) to TSS specified in link field of current TSS;
```

IRET/IRETD—Interrupt Return（続き）

```
    Mark the task just abandoned as NOT BUSY;
    IF EIP is not within code segment limit
        THEN #GP(0);
    FI;
END;

PROTECTED-MODE-RETURN: (* PE=1, VM=0 in flags image *)
    IF return code segment selector is null THEN GP(0); FI;
    IF return code segment selector addrsses descriptor beyond descriptor table limit
        THEN GP(selector; FI;
    Read segment descriptor pointed to by the return code segment selector
    IF return code segment descriptor is not a code segment THEN #GP(selector); FI;
    IF return code segment selector RPL < CPL THEN #GP(selector); FI;
    IF return code segment descriptor is conforming
        AND return code segment DPL > return code segment selector RPL
            THEN #GP(selector); FI;
    IF return code segment descriptor is not present THEN #NP(selector); FI:
    IF return code segment selector RPL > CPL
        THEN GOTO RETURN-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL
    FI;
END;

RETURN-TO-SAME-PRIVILEGE-LEVEL: (* PE=1, VM=0 in flags image, RPL=CPL *)
    IF EIP is not within code segment limits THEN #GP(0); FI;
    EIP ← tempEIP;
    CS ← tempCS; (* segment descriptor information also loaded *)
    EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
    IF OperandSize=32
        THEN
            EFLAGS(RF, AC, ID) ← tempEFLAGS;
    FI;
    IF CPL ≤ IOPL
        THEN
            EFLAGS(IF) ← tempEFLAGS;
    FI;
    IF CPL = 0
        THEN
            EFLAGS(IOPL) ← tempEFLAGS;
            IF OperandSize=32
                THEN EFLAGS(VM, VIF, VIP) ← tempEFLAGS;
            FI;
    FI;
END;

RETURN-TO-OUTER-PRIVILGE-LEVEL:
    IF OperandSize=32
        THEN
            IF top 8 bytes on stack are not within limits THEN #SS(0); FI;
```

IRET/IRETD—Interrupt Return（続き）

```
            ELSE (* OperandSize=16 *)
                IF top 4 bytes on stack are not within limits THEN #SS(0); FI;
        FI;
        Read return segment selector;
        IF stack segment selector is null THEN #GP(0); FI;
        IF return stack segment selector index is not within its descriptor table limits
                THEN #GP(SSselector); FI;
        Read segment descriptor pointed to by return segment selector;
        IF stack segment selector RPL ≠ RPL of the return code segment selector
            IF stack segment selector RPL ≠ RPL of the return code segment selector
            OR the stack segment descriptor does not indicate a a writable data segment;
            OR stack segment DPL ≠ RPL of the return code segment selector
                    THEN #GP(SS selector);
            FI;
            IF stack segment is not present THEN #SS(SS selector); FI;
        IF tempEIP is not within code segment limit THEN #GP(0); FI;
        EIP ← tempEIP;
        CS ← tempCS;
        EFLAGS (CF, PF, AF, ZF, SF, TF, DF, OF, NT) ← tempEFLAGS;
        IF OperandSize=32
            THEN
                EFLAGS(RF, AC, ID) ← tempEFLAGS;
        FI;
        IF CPL ≤ IOPL
            THEN
                EFLAGS(IF) ← tempEFLAGS;
        FI;
        IF CPL = 0
            THEN
                EFLAGS(IOPL) ← tempEFLAGS;
                IF OperandSize=32
                    THEN EFLAGS(VM, VIF, VIP) ← tempEFLAGS;
                FI;
        FI;
        CPL ← RPL of the return code segment selector;
        FOR each of segment register (ES, FS, GS, and DS)
            DO;
                IF segment register points to data or non-conforming code segment
                AND CPL > segment descriptor DPL (* stored in hidden part of segment
                register *)
                    THEN (* segment register invalid *)
                        SegmentSelector ← 0; (* null segment selector *)
                FI;
            OD;
    END:
```

IRET/IRETD—Interrupt Return（続き）

影響を受けるフラグ

プロセッサの動作モードに応じて、EFLAGS レジスタ内のすべてのフラグおよびフィールドが修正される可能性がある。ネストされたタスクから以前のタスクへのリターンを実行した場合は、EFLAGS レジスタは前のタスクの TSS にストアされていた EFLAGS イメージにしたがって修正される。

保護モード例外

#GP(0)	リターンコードまたはスタック・セグメント・セレクタがヌルの場合。
	リターン命令ポインタがリターン・コード・セグメントの範囲内にない場合。
#GP（セレクタ）	セグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲外の場合。
	リターン・コード・セグメント・セレクタの RPL が CPL より大きい場合。
	コンフォーミング・コード・セグメントの DPL がリターン・コード・セグメント・セレクタの RPL より大きい場合。
	非コンフォーミング・コード・セグメントの DPL がコード・セグメント・セレクタの RPL に等しくない場合。
	スタック・セグメント・ディスクリプタの DPL がリターン・コード・セグメント・セレクタの RPL に等しくない場合。
	スタック・セグメントが書き込み可能なデータ・セグメントでない場合。
	スタック・セグメント・セレクタの RPL がリターン・コード・セグメント・セレクタの RPL に等しくない場合。
	コード・セグメントのセグメント・ディスクリプタが、それがコード・セグメントであることを示していない場合。
	TSS のセグメント・セレクタのローカル / グローバル・ビットがローカルに設定されている場合。
	TSS のセグメント・ディスクリプタが、その TSS がビジーでないと指定している場合。
	TSS のセグメント・ディスクリプタが、その TSS が使用不可能であると指定している場合。
#SS(0)	スタックのトップバイトがスタックの範囲内にない場合。
#NP（セレクタ）	リターンコードまたはスタック・セグメントが存在しない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

IRET/IRETD—Interrupt Return（続き）

#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	リターン命令ポインタがリターン・コード・セグメントの範囲内にない場合。
#SS	スタックのトップバイトがスタックの範囲内にない場合。

仮想 8086 モード例外

#GP(0)	リターン命令ポインタがリターン・コード・セグメントの範囲内にない場合。 IOPL が 3 に等しくない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#SS(0)	スタックのトップバイトがスタックの範囲内にない場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

J*cc*—Jump if Condition Is Met

オペコード	命令	説明
77 *cb*	JA *rel8*	より上（CF=0 および ZF=0）の場合 short ジャンプする。
73 *cb*	JAE *rel8*	より上か等しい（CF=0）場合 short ジャンプする。
72 *cb*	JB *rel8*	より下（CF=1）の場合 short ジャンプする。
76 *cb*	JBE *rel8*	より下か等しい（CF=1 または ZF=1）場合 short ジャンプする。
72 *cb*	JC *rel8*	キャリーがある（CF=1）場合 short ジャンプする。
E3 *cb*	JCXZ *rel8*	CX レジスタが 0 の場合 short ジャンプする。
E3 *cb*	JECXZ *rel8*	ECX レジスタが 0 の場合 short ジャンプする。
74 *cb*	JE *rel8*	等しい（ZF=1）場合 short ジャンプする。
7F *cb*	JG *rel8*	より大きい（ZF=0 および SF=OF）場合 short ジャンプする。
7D *cb*	JGE *rel8*	より大きいか等しい（SF=OF）場合 short ジャンプする。
7C *cb*	JL *rel8*	より小さい（SF<>OF）場合 short ジャンプする。
7E *cb*	JLE *rel8*	より小さいか等しい(ZF=1 または SF<>OF)場合 short ジャンプする。
76 *cb*	JNA *rel8*	より上でない（CF=1 または ZF=1）場合 short ジャンプする。
72 *cb*	JNAE *rel8*	より上でなく等しくない（CF=1）場合 short ジャンプする。
73 *cb*	JNB *rel8*	より下でない（CF=0）場合 short ジャンプする。
77 *cb*	JNBE *rel8*	より下でなく等しくない(CF=0およびZF=0)場合shortジャンプする。
73 *cb*	JNC *rel8*	キャリーがない（CF=0）場合 short ジャンプする。
75 *cb*	JNE *rel8*	等しくない（ZF=0）場合 short ジャンプする。
7E *cb*	JNG *rel8*	より大きくない（ZF=1 または SF<>OF）場合 short ジャンプする。
7C *cb*	JNGE *rel8*	より大きくなく等しくない（SF<>OF）場合 short ジャンプする。
7D *cb*	JNL *rel8*	より小さくない（SF=OF）場合 short ジャンプする。
7F *cb*	JNLE *rel8*	より小さくなく等しくない（ZF=0 および SF=OF）場合 short ジャンプする。
71 *cb*	JNO *rel8*	オーバーフローがない（OF=0）場合 short ジャンプする。
7B *cb*	JNP *rel8*	パリティがない（PF=0）場合 short ジャンプする。
79 *cb*	JNS *rel8*	符号がない（SF=0）場合 short ジャンプする。
75 *cb*	JNZ *rel8*	ゼロでない（ZF=0）場合 short ジャンプする。
70 *cb*	JO *rel8*	オーバーフローがある（OF=1）場合 short ジャンプする。
7A *cb*	JP *rel8*	パリティがある（PF=1）場合 short ジャンプする。
7A *cb*	JPE *rel8*	パリティが偶数（PF=1）の場合 short ジャンプする。
7B *cb*	JPO *rel8*	パリティが奇数（PF=0）の場合 short ジャンプする。
78 *cb*	JS *rel8*	符号がある（SF=1）場合 short ジャンプする。
74 *cb*	JZ *rel8*	ゼロ（ZF=1）の場合 short ジャンプする。
0F 87 *cw/cd*	JA *rel16/32*	より上（CF=0 および ZF=0）の場合 near ジャンプ。
0F 83 *cw/cd*	JAE *rel16/32*	より上か等しい（CF=0）場合 near ジャンプする。
0F 82 *cw/cd*	JB *rel16/32*	より下（CF=1）の場合 near ジャンプする。
0F 86 *cw/cd*	JBE *rel16/32*	より下か等しい（CF=1 または ZF=1）場合 near ジャンプする。

J*cc*—Jump if Condition Is Met（続き）

オペコード	命令	説明
0F 82 *cw/cd*	JC *rel16/32*	キャリーがある（CF=1）場合 near ジャンプする。
0F 84 *cw/cd*	JE *rel16/32*	等しい（ZF=1）場合 near ジャンプする。
0F 84 *cw/cd*	JZ *rel16/32*	ゼロ（ZF=1）の場合 near ジャンプする。
0F 8F *cw/cd*	JG *rel16/32*	より大きい（ZF=0 および SF=OF）場合 near ジャンプする。
0F 8D *cw/cd*	JGE *rel16/32*	より大きいか等しい（SF=OF）場合 near ジャンプする。
0F 8C *cw/cd*	JL *rel16/32*	より小さい（SF<>OF）場合 near ジャンプする。
0F 8E *cw/cd*	JLE *rel16/32*	より小さいか等しい(ZF=1 または SF<>OF)場合 near ジャンプする。
0F 86 *cw/cd*	JNA *rel16/32*	より上でない（CF=1 または ZF=1）場合 near ジャンプする。
0F 82 *cw/cd*	JNAE *rel16/32*	より上でなく等しくない（CF=1）場合 near ジャンプする。
0F 83 *cw/cd*	JNB *rel16/32*	より下でない（CF=0）場合 near ジャンプする。
0F 87 *cw/cd*	JNBE *rel16/32*	より下でなく等しくない(CF=0およびZF=0)場合nearジャンプする。
0F 83 *cw/cd*	JNC *rel16/32*	キャリーがない（CF=0）場合 near ジャンプする。
0F 85 *cw/cd*	JNE *rel16/32*	等しくない（ZF=0）場合 near ジャンプする。
0F 8E *cw/cd*	JNG *rel16/32*	より大きくない（ZF=1 または SF<>OF）場合 near ジャンプする。
0F 8C *cw/cd*	JNGE *rel16/32*	より大きくなく等しくない（SF<>OF）場合 near ジャンプする。
0F 8D *cw/cd*	JNL *rel16/32*	より小さくない（SF=OF）場合 near ジャンプする。
0F 8F *cw/cd*	JNLE *rel16/32*	より小さくなく等しくない（ZF=0 および SF=OF）場合 near ジャンプする。
0F 81 *cw/cd*	JNO *rel16/32*	オーバーフローがない（OF=0）場合 near ジャンプする。
0F 8B *cw/cd*	JNP *rel16/32*	パリティがない（PF=0）場合 near ジャンプする。
0F 89 *cw/cd*	JNS *rel16/32*	符号がない（SF=0）場合 near ジャンプする。
0F 85 *cw/cd*	JNZ *rel16/32*	ゼロでない（ZF=0）場合 near ジャンプする。
0F 80 *cw/cd*	JO *rel16/32*	オーバーフローがある（OF=1）場合 near ジャンプする。
0F 8A *cw/cd*	JP *rel16/32*	パリティがある（PF=1）場合 near ジャンプする。
0F 8A *cw/cd*	JPE *rel16/32*	パリティが偶数（PF=1）の場合 near ジャンプする。
0F 8B *cw/cd*	JPO *rel16/32*	パリティが奇数（PF=0）の場合 near ジャンプする。
0F 88 *cw/cd*	JS *rel16/32*	符号がある（SF=1）場合 near ジャンプする。
0F 84 *cw/cd*	JZ *rel16/32*	0（ZF=1）の場合 near ジャンプする。

説明

EFLAGS レジスタ内のステータス・フラグ（CF、OF、PF、SF、ZF）の 1 つ以上の状態を調べ、それらのフラグが指定された状態（条件）にある場合は、デスティネーション・オペランドによって指定されたターゲット命令へのジャンプを実行する。各命令に特定の条件コード（*cc*）が対応しており、テストされる条件を示している。条件が満たされなかった場合は、ジャンプは実行されず、J*cc* 命令の次の命令から実行が継続される。

J*cc*—Jump if Condition Is Met（続き）

ターゲット命令は相対オフセット（EIP レジスタ内の命令ポインタの現在値に相対的な符号付きオフセット）で指定される。相対オフセット（rel8、rel16、または rel32）は、アセンブリ・コードでは一般的にラベルとして指定されるが、マシン・コード・レベルでは、符号付きの 8 ビットまたは 32 ビットの即値としてコード化され、命令ポインタに加算される。この命令のコーディングは、–128 ～ +127 のオフセットの場合に最も効率がよい。オペランド・サイズ属性が 16 である場合は、EIP レジスタの上位 2 バイトはクリアされ、命令ポインタの最大サイズは 16 ビットになる。

各 J*cc* ニーモニックの条件は、前 2 ページの表の「説明」欄に示してある。「より小さい」および「より大きい」という表現は、符号付き整数の比較に使用され、「より上」および「より下」という表現は、符号なし整数の比較に使用されている。

ステータス・フラグの特定の状態はときとして 2 種類に解釈されることがあるので、一部のオペコードに対しては 2 つのニーモニックが定義されている。例えば、JA（より上の場合ジャンプ）命令と JNBE（より下でなく等しくない場合ジャンプ）命令は、オペコード 77H に対する 2 つのニーモニックである。

J*cc* 命令では、far ジャンプ（他のコード・セグメントへのジャンプ）をサポートしていない。条件付きジャンプのターゲットが別のセグメントにある場合は、J*cc* 命令に対するテスト対象の条件と反対の条件を使用し、次に他のセグメントへの無条件 far ジャンプ（JMP 命令）でターゲットにアクセスする。例えば、以下の条件付き far ジャンプは不当である。

```
JZ FARLABEL;
```

この far ジャンプを行うには、以下の 2 つの命令を使用する。

```
JNZ BEYOND;
JMP FARLABEL;
BEYOND:
```

JECXZ 命令および JCXZ 命令は、ステータス・フラグをチェックしない点で他の J*cc* 命令とは異なっている。その代わりに、これらの命令はそれぞれ ECX および CX レジスタの内容が 0 かどうかをチェックする。CX、ECX のどちらのレジスタを選ぶかは、アドレスサイズ属性による。これらの命令は、（LOOPNE などの）条件付きループ命令で終了する条件付きループの先頭に使用すると役立つ。すなわち、ECX または CX レジスタが 0 に等しいときループに入らないようにして、ループがゼロ回でなくそれぞれ 2^{32} または 64K 回実行してしまうのを回避する。

条件付きジャンプはすべて、ジャンプアドレスにも、キャッシュ可能かどうかにも関係なく、1 または 2 キャッシュ・ラインのコードフェッチに変換される。

J*cc*—Jump if Condition Is Met（続き）

操作

```
IF condition
    THEN
        EIP ← EIP + SignExtend(DEST);
        IF OperandSize = 16
            THEN
                EIP ← EIP AND 0000FFFFH;
        FI;
            ELSE (* OperandSize = 32 *)
                IF EIP < CS.Base OR EIP > CS.Limit
                    #GP
        FI;
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	ジャンプ先のオフセットが CS セグメントの範囲を超えている場合。

実アドレスモード例外

#GP(0)	ジャンプ先のオフセットが CS セグメントの範囲を超えているか、または 0 ～ FFFFH の実効アドレス空間外の場合。この状態は、32 ビット・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た場合に生じることがある。

仮想 8086 モード例外

実アドレスモードと同じ例外。

JMP—Jump

オペコード	命令	説明
EB *cb*	JMP *rel8*	次の命令との相対分量分だけ相対 short ジャンプする。
E9 *cw*	JMP *rel16*	次の命令との相対分量分だけ相対 near ジャンプする。
E9 *cd*	JMP *rel32*	次の命令との相対分量分だけ相対 near ジャンプする。
FF /4	JMP *r/m16*	*r/m16* で指定されるアドレスに絶対間接 near ジャンプする。
FF /4	JMP *r/m32*	*r/m32* で指定されるアドレスに絶対間接 near ジャンプする。
EA *cd*	JMP *ptr16:16*	オペランドで指定されるアドレスに絶対 far ジャンプする。
EA *cp*	JMP *ptr16:32*	オペランドで指定されるアドレスに絶対 far ジャンプする。
FF /5	JMP *m16:16*	*m16:16* で指定されるアドレスに絶対間接 far ジャンプする。
FF /5	JMP *m16:32*	*m16:32* で指定されるアドレスに絶対間接 far ジャンプする。

説明

リターン情報を記録しないで、プログラムの制御を命令ストリーム内の別の点に移す。デスティネーション（ターゲット）オペランドには、ジャンプ先の命令のアドレスを指定する。このオペランドには、即値、汎用レジスタ、またはメモリ・ロケーションを使用できる。

この命令を使用して、以下の異なる 4 つのタイプのジャンプを実行することができる。

- near ジャンプ ― 現在のコード・セグメント（現在の CS レジスタの指示先のセグメント）内にある命令へのジャンプ。セグメント内ジャンプともいう。
- short ジャンプ ― ジャンプ範囲が EIP の現在値の –128 ～ +127 に限られる near ジャンプ。
- far ジャンプ ― 現在のコード・セグメントとは異なるが特権レベルは同じであるセグメント内にある命令へのジャンプ。セグメント間ジャンプともいう。
- タスクスイッチ ― 異なるタスク内にある命令へのジャンプ。

タスクスイッチは保護モードでしか実行することができない。JMP 命令でのタスクスイッチ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、下巻』の第 6 章「タスク管理」を参照のこと。

near ジャンプと short ジャンプ: near ジャンプを実行すると、プロセッサはターゲット・オペランドで指定された（現在のコード・セグメント内の）アドレスにジャンプする。ターゲット・オペランドは、絶対オフセット（コード・セグメントのベースからのオフセット）か、または相対オフセット（EIP レジスタ内の命令ポインタの現在値に相対的な符号付きディスプレースメント）を指定する。8 ビットの相対オフセット（*rel8*）への near ジャンプのことを short ジャンプという。near ジャンプと short ジャンプでは CS レジスタは変更されない。

JMP—Jump（続き）

絶対オフセットは、汎用レジスタまたはメモリ・ロケーション（*r/m16* または *r/m32*）で間接的に指定される。ターゲット・オペランドのサイズ（16 または 32 ビット）はオペランド・サイズ属性によって決まる。絶対オフセットは EIP レジスタに直接ロードされる。オペランド・サイズ属性が 16 である場合は、EIP レジスタの上位 2 バイトはクリアされ、命令ポインタの最大サイズは 16 ビットになる。

相対オフセット（*rel8*、*rel16*、または *rel32*）は、アセンブリ・コードでは一般的にラベルとして指定されるが、マシン・コード・レベルでは、符号付きの 8、16、または 32 ビットの即値としてコード化され、命令ポインタに加算される。（これで、EIP レジスタの内容は JMP 命令の次の命令のアドレスになる。）相対オフセットを使用するときは、ターゲット・オペランドのサイズ（8、16、または 32 ビット）は、（short ジャンプか near ジャンプかの）オペコードと（相対 near ジャンプの）オペランド・サイズ属性によって決まる。

実アドレスモードまたは仮想 8086 モードでの far ジャンプ: 実アドレスモードまたは仮想 8086 モードで far ジャンプを実行すると、プロセッサはターゲット・オペランドで指定されたコード・セグメントとオフセットにジャンプする。この場合は、ターゲット・オペランドは、絶対 far アドレスをポインタ（*ptr16:16* または *ptr16:32*）で直接に、またはメモリ・ロケーション（*m16:16* または *m16:32*）で間接的に指定する。ポインタ方式では、4 バイト（16 ビット・オペランド・サイズ）または 6 バイト（32 ビット・オペランド・サイズ）の far アドレス即値を使用して、コール先プロシージャのセグメントおよびアドレスが命令内にコード化される。間接方式では、ターゲット・オペランドが 4 バイト（16 ビット・オペランド・サイズ）または 6 バイト（32 ビット・オペランド・サイズ）の far アドレスを内容とするメモリ・ロケーションを指定する。far アドレスは、CS および EIP レジスタに直接ロードされる。オペランド・サイズ属性が 16 である場合は、EIP レジスタの上位 2 バイトはクリアされる。

保護モードでの far ジャンプ: プロセッサが保護モードで動作しているときは、JMP 命令を使用して以下の 3 つのタイプの far ジャンプを実行することができる。

- コンフォーミングまたは非コンフォーミングのコード・セグメントへの far ジャンプ。
- コールゲートを介した far ジャンプ。
- タスクスイッチ。

（JMP 命令を使用して特権レベル間の far ジャンプを実行することはできない。）

JMP—Jump（続き）

保護モードでは、プロセッサは、常に far アドレスのセグメント・セレクタ部を使用して、GDT または LDT 内の対応するディスクリプタをアクセスする。ディスクリプタのタイプ（コード・セグメント、コールゲート、タスクゲート、または TSS）およびアクセス権によって、実行されるジャンプのタイプが決まる。

選択されたディスクリプタがコード・セグメントのディスクリプタである場合は、同じ特権レベルのコード・セグメントへの far ジャンプが実行される。（選択されたコード・セグメントが異なる特権レベルにあり、かつそのコード・セグメントが非コンフォーミングである場合は、一般保護例外が発生する。）保護モードでの同じ特権レベルへの far ジャンプは、実アドレスモードまたは仮想 8086 モードで実行される far ジャンプによく似ている。ターゲット・オペランドは、絶対 far アドレスをポインタ（*ptr16:16* または *ptr16:32*）で直接に、またはメモリ・ロケーション（*m16:16* または *m16:32*）で間接的に指定する。オペランド・サイズ属性によって、far アドレス内のオフセットのサイズ（16 ビットまたは 32 ビット）が決まる。新しいコード・セグメント・セレクタとそのディスクリプタが CS レジスタにロードされ、命令からのオフセットが EIP レジスタにロードされる。コールゲート（次の段落で説明）を使用して、同じ特権レベルのコード・セグメントへの far コールも実行することができるので注意する。この仕組みを使用すると、特別レベルのインダイレクションが可能になり、これは 16 ビットと 32 ビットとのコード・セグメント間のジャンプの優先実行方式になっている。

コールゲートを介して far ジャンプを実行するときは、ターゲット・オペランドによって指定されたセグメント・セレクタによってコールゲートが識別される。（ターゲット・オペランドのオフセット部は無視される。）そこで、プロセッサはコール・ゲート・ディスクリプタで指定されたコード・セグメントにジャンプし、コールゲートで指定されたオフセットから命令の実行を開始する。スタックスイッチは行われない。この場合もやはり、ターゲット・オペランドは、コールゲートの far アドレスをポインタ（*ptr16:16* または *ptr16:32*）で直接にも、メモリ・ロケーション（*m16:16* または *m16:32*）で間接的にも指定できる。

JMP 命令でタスクスイッチを実行するのは、コールゲートを介したジャンプを実行するのに多少似ている。タスクスイッチの場合は、ターゲット・オペランドは、切り替え先タスクへのタスクゲートのセグメント・セレクタを指定する（ターゲット・オペランドのオフセット部は無視される）。次に、選択されたタスクゲートが、タスクのコード・セグメントおよびスタック・セグメントのセグメント・セレクタがストアされているタスクの TSS を指す。TSS には、さらにタスクが中断されなければ次に実行される予定であった命令の EIP 値も入っている。この命令ポインタ値は EIP レジスタにロードされ、したがって、タスクはその実行されなかった最初の命令から実行を再開する。

JMP—Jump（続き）

JMP 命令は、TSS のセグメント・セレクタを直接指定することもでき、その結果タスクゲートのインダイレクションの必要がなくなる。タスクスイッチの仕組みの詳細については、『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、下巻』の第 6 章「タスク管理」を参照のこと。

JMP 命令でタスクスイッチを実行するときは、EFLAGS レジスタにネストされたタスクフラグ（NT）がセットされず、新しい TSS の以前のタスク・リンク・フィールドに前のタスクの TSS セレクタがロードされないので注意する。したがって、前のタスクへのリターンは IRET 命令の実行では実現できない。JMP 命令でタスクスイッチを実行するのは、その点で CALL 命令と異なる。すなわち、CALL 命令は NT フラグをセットし、以前のタスクリンク情報をセーブするので、IRET 命令でのコール元タスクへのリターンが可能になる。

操作

```
IF near jump
    THEN IF near relative jump
        THEN
            tempEIP ← EIP + DEST; (* EIP is instruction following JMP instruction*)
        ELSE (* near absolute jump *)
            tempEIP ← DEST;
    FI;
    IF tempEIP is beyond code segment limit THEN #GP(0); FI;
    IF OperandSize = 32
        THEN
            EIP ← tempEIP;
        ELSE (* OperandSize=16 *)
            EIP ← tempEIP AND 0000FFFFH;
    FI;
FI:

IF far jump AND (PE = 0 OR (PE = 1 AND VM = 1)) (* real-address or virtual-8086 mode *)
    THEN
        tempEIP ← DEST[offset); (* DEST is ptr16:32 or [m16:32] *)
        IF tempEIP is beyond code segment limit THEN #GP(0); FI;
        CS ← DEST[segment selector); (* DEST is ptr16:32 or [m16:32] *)
        IF OperandSize ← 32
            THEN
                EIP ← tempEIP; (* DEST is ptr16:32 or [m16:32] *)
            ELSE (* OperandSize=16 *)
                EIP ← tempEIP AND 0000FFFFH; (* clear upper 16 bits *)
        FI;
FI;
IF far jump AND (PE = 1 AND VM = 0) (* Protected mode, not virtual-8086 mode *)
    THEN
        IF effective address in the CS, DS, ES, FS, GS, or SS segment is illegal
            OR segment selector in target operand null
```

JMP—Jump（続き）

```
                THEN #GP(0);
            FI;
            IF segment selector index not within descriptor table limits
                THEN #GP(new selector);
            FI;
            Read type and access rights of segment descriptor;
            IF segment type is not a conforming or nonconforming code segment, call gate,
                task gate, or TSS THEN #GP(segment selector); FI;
            Depending on type and access rights
                GO TO CONFORMING-CODE-SEGMENT;
                GO TO NONCONFORMING-CODE-SEGMENT;
                GO TO CALL-GATE;
                GO TO TASK-GATE;
                GO TO TASK-STATE-SEGMENT;
        ELSE
            #GP(segment selector);
FI;

CONFORMING-CODE-SEGMENT:
    IF DPL > CPL THEN #GP(segment selector); FI;
    IF segment not present THEN #NP(segment selector); FI;
    tempEIP ← DEST[offset);
    IF OperandSize=16
        THEN tempEIP ← tempEIP AND 0000FFFFH;
    FI;
    IF tempEIP not in code segment limit THEN #GP(0); FI;
    CS ← DEST[SegmentSelector); (* segment descriptor information also loaded *)
    CS(RPL) ← CPL
    EIP ← tempEIP;
END;

NONCONFORMING-CODE-SEGMENT:
    IF (RPL > CPL) OR (DPL ≠ CPL) THEN #GP(code segment selector); FI;
    IF segment not present THEN #NP(segment selector); FI;
    IF instruction pointer outside code segment limit THEN #GP(0); FI;
    tempEIP ← DEST[offset);
    IF OperandSize=16
        THEN tempEIP ← tempEIP AND 0000FFFFH;
    FI;
    IF tempEIP not in code segment limit THEN #GP(0); FI;
    CS ← DEST[SegmentSelector); (* segment descriptor information also loaded *)
    CS(RPL) ← CPL
    EIP ← tempEIP;
END;

CALL-GATE:
    IF call gate DPL < CPL
        OR call gate DPL < call gate segment-selector RPL
            THEN #GP(call gate selector); FI;
```

JMP—Jump（続き）

```
        IF call gate not present THEN #NP(call gate selector); FI;
        IF call gate code-segment selector is null THEN #GP(0); FI;
        IF call gate code-segment selector index is outside descriptor table limits
            THEN #GP(code segment selector); FI;
        Read code segment descriptor;
        IF code-segment segment descriptor does not indicate a code segment
            OR code-segment segment descriptor is conforming and DPL > CPL
            OR code-segment segment descriptor is non-conforming and DPL ≠ CPL
                THEN #GP(code segment selector); FI;
        IF code segment is not present THEN #NP(code-segment selector); FI;
        IF instruction pointer is not within code-segment limit THEN #GP(0); FI;
        tempEIP ← DEST[offset);
        IF GateSize=16
            THEN tempEIP ← tempEIP AND 0000FFFFH;
        FI;
        IF tempEIP not in code segment limit THEN #GP(0); FI;
        CS ← DEST[SegmentSelector); (* segment descriptor information also loaded *)
        CS(RPL) ← CPL
        EIP ← tempEIP;
    END;

    TASK-GATE:
        IF task gate DPL < CPL
            OR task gate DPL < task gate segment-selector RPL
                THEN #GP(task gate selector); FI;
        IF task gate not present THEN #NP(gate selector); FI;
        Read the TSS segment selector in the task-gate descriptor;
        IF TSS segment selector local/global bit is set to local
            OR index not within GDT limits
            OR TSS descriptor specifies that the TSS is busy
                THEN #GP(TSS selector); FI;
        IF TSS not present THEN #NP(TSS selector); FI;
        SWITCH-TASKS to TSS;
        IF EIP not within code segment limit THEN #GP(0); FI;
    END;

    TASK-STATE-SEGMENT:
        IF TSS DPL < CPL
            OR TSS DPL < TSS segment-selector RPL
            OR TSS descriptor indicates TSS not available
                THEN #GP(TSS selector); FI;
        IF TSS is not present THEN #NP(TSS selector); FI;
        SWITCH-TASKS to TSS
        IF EIP not within code segment limit THEN #GP(0); FI;
    END;
```

JMP—Jump（続き）

影響を受けるフラグ

タスクスイッチが行われた場合はすべてのフラグが影響を受け、タスクスイッチが行われなかった場合はどのフラグも影響を受けない。

保護モード例外

#GP(0)	ターゲット・オペランド、コールゲート、または TSS 内のオフセットがコード・セグメントの範囲を超えている場合。
	デスティネーション・オペランド、コールゲート、タスクゲート、または TSS 内のセグメント・セレクタがヌルの場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#GP（セレクタ）	セグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲外の場合。
	デスティネーション・オペランド内のセグメント・セレクタによって指されているセグメント・ディスクリプタが、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、タスク・ステート・セグメントのいずれのディスクリプタでもない場合。
	非コンフォーミング・コード・セグメントの DPL が CPL に等しくない場合。
	（コールゲートを使用しない場合）セグメントのセグメント・セレクタの RPL が CPL より大きい場合。
	コンフォーミング・コード・セグメントの DPL が CPL より大きい場合。
	コールゲート、タスクゲート、または TSS のセグメント・ディスクリプタからの DPL が CPL より小さいか、あるいはコールゲート、タスクゲート、または TSS のセグメント・セレクタの RPL より小さい場合。
	コールゲート内のセレクタのセグメント・ディスクリプタが、それがコード・セグメントであることを示していない場合。
	タスクゲート内のセグメント・セレクタのセグメント・ディスクリプタが使用可能な TSS を示していない場合。
	TSS のセグメント・セレクタのローカル / グローバル・ビットがローカルに設定されている場合。
	TSS のセグメント・ディスクリプタが、その TSS がビジーであるか使用不可能であると指定している場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

JMP—Jump（続き）

#NP（セレクタ）	アクセスされるコード・セグメントが存在しない場合。 コールゲート、タスクゲート、または TSS が存在しない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。（メモリからターゲットをフェッチするときだけ発生する。）

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	ターゲット・オペランドがコード・セグメントの範囲を超えている場合。 メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。（メモリからターゲットをフェッチするときだけ発生する。）

LAHF—Load Status Flags into AH Register

オペコード	命令	説明
9F	LAHF	AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF）をロードする。

説明

EFLAGS レジスタの下位バイト（ここにはステータス・フラグ SF、ZF、AF、PF、および CF がある）を AH レジスタに転送する。EFLAGS レジスタの予約ビット 1、3、5 は、AH レジスタ内では下記の「操作」の項に示すように設定される。

操作

```
AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF);
```

影響を受けるフラグ

なし（すなわち、EFLAGS レジスタ内のフラグの状態は影響を受けない）。

例外（すべての操作モード）

なし。

LAR—Load Access Rights Byte

オペコード	命令	説明
0F 02 /r	LAR *r16, r/m16*	*r16* ← FF00H でマスクされた *r/m16*。
0F 02 /r	LAR *r32, r/m32*	*r32* ← 00FxFF00H でマスクされた *r/m32*。

説明

第 2 オペランド（ソース・オペランド）によって指定されたセグメント・ディスクリプタから第 1 オペランド（デスティネーション・オペランド）にアクセス権をロードし、EFLAGS レジスタ内の ZF フラグをセットする。ソース・オペランド（レジスタまたはメモリ・ロケーション）の内容は、アクセスされるセグメント・ディスクリプタのセグメント・セレクタである。デスティネーション・オペランドは、汎用レジスタである。

プロセッサは、ロードプロセスの一環としてアクセスチェックを行う。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をさらに追加チェックすることができる。

オペランド・サイズが 32 ビットのときは、セグメント・ディスクリプタに対するアクセス権には、タイプ・フィールドおよび DPL フィールドと、S、P、AVL、D/B、G フラグが含まれ、これらはすべてセグメント・ディスクリプタの 2 番目のダブルワード（バイト 4 ～ 7）内にある。このダブルワードは、00FXFF00H でマスクされてからデスティネーション・オペランドにロードされる。オペランド・サイズが 16 ビットのときは、アクセス権にはタイプ・フィールドおよび DPL フィールドが含まれる。この場合は、ダブルワードの下位 2 バイトが FF00H でマスクされてからデスティネーション・オペランドにロードされる。

この命令では、アクセス権をデスティネーション・レジスタにロードする前に以下のチェックを行う。

- セグメント・セレクタがヌルでないことを確認する。
- セグメント・セレクタの指示先がアクセスされる GDT または LDT の範囲内のディスクリプタであることを確認する。
- ディスクリプタのタイプがこの命令に対して有効であることを確認する。LAR 命令に対しては、すべてのコード・セグメント・ディスクリプタとデータ・セグメント・ディスクリプタが有効である（すなわち、LAR 命令でアクセスできる）。有効なシステム・セグメントおよびゲートのディスクリプタ・タイプが下記の表に示してある。

LAR—Load Access Rights Byte（続き）

- 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、この命令は指定されたセグメント・ディスクリプタがCPLでアクセスできる（すなわち、CPLとセグメント・セレクタのRPLがセグメント・セレクタのDPL以下である）かを確認する。

セグメント・ディスクリプタがアクセスできないか、またはこの命令にとって無効なタイプである場合は、ZFフラグがクリアされ、アクセス権は何もデスティネーション・オペランドにロードされない。

LAR命令は保護モードでしか実行することができない。

タイプ	名前	有効 / 無効
0	予約済み	無効
1	使用可能 16 ビット TSS	有効
2	LDT	有効
3	ビジー 16 ビット TSS	有効
4	16 ビット・コール・ゲート	有効
5	16 ビット /32 ビット・タスク・ゲート	有効
6	16 ビット割り込みゲート	無効
7	16 ビット・トラップ・ゲート	無効
8	予約済み	無効
9	使用可能 32 ビット TSS	有効
A	予約済み	無効
B	ビジー 32 ビット TSS	有効
C	32 ビット・コール・ゲート	有効
D	予約済み	無効
E	32 ビット割り込みゲート	無効
F	32 ビット・トラップ・ゲート	無効

LAR—Load Access Rights Byte（続き）

操作

```
IF SRC[Offset) > descriptor table limit THEN ZF = 0; FI;
Read segment descriptor;
IF SegmentDescriptor(Type) ≠ conforming code segment
    AND (CPL > DPL) OR (RPL > DPL)
    OR Segment type is not valid for instruction
        THEN
            ZF ← 0
        ELSE
            IF OperandSize = 32
                THEN
                    DEST ← [SRC] AND 00FxFF00H;
                ELSE (*OperandSize = 16*)
                    DEST ← [SRC] AND FF00H;
            FI;
FI;
```

影響を受けるフラグ

アクセス権が正常にロードされた場合は ZF フラグが 1 にセットされ、ロードできなかった場合は 0 にクリアされる。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。（メモリからターゲットをフェッチするときだけ発生する。）

実アドレスモード例外

#UD	LAR 命令は実アドレスモードでは認識されない。

仮想 8086 モード例外

#UD	LAR 命令は仮想 8086 モードでは認識されない。

LDDQU—Load Unaligned Integer 128 bits

オペコード	命令	説明
F2, 0F, F0, /r	LDDQU *xmm*, *mem*	*mem* からデータをロードし、*xmm* レジスタに 128 ビットを返す。

説明

この命令は、メモリからロードする MOVDQU xmm, m128 に機能的によく似ている。すなわち、ソース・メモリ・オペランド（第 2 オペランド）によって指定されるアドレスを始点とする 16 バイトのデータをメモリから取り出し、デスティネーション・レジスタ（第 1 オペランド）に置く。ソース・オペランドのアライメントが 16 バイトに合っている必要はない。メモリから最大 32 バイトまでロードできる。この値はプロセッサのモデルによって異なる。

ソース・オペランドがキャッシュ・ライン境界を超えている場合は、この命令を使用すると、MOVDQU 命令よりパフォーマンスが向上することがある。LDDQU 命令によってロードしたデータを修正して同じ位置に格納する必要がある場合は、LDDQU 命令ではなく、MOVDQU 命令または MOVDQA 命令を使用する。16 バイトに合っていることがわかっているメモリ上の位置との間でダブル・クワッドワードを移動する場合は、MOVDQA 命令を使用する。

実装に関する注意

- ソース・オペランドのアライメントが 16 バイトに合っている場合、プロセッサのモデルによっては、16 バイトが 2 回以上ロードされることがある。この理由で、キャッシュ不可またはライト・コンバイン（WC）メモリ領域を使用する場合は、LDDQU 命令の使用を避けるべきである。キャッシュ不可または WC メモリ領域には、MOVDQU 命令を使用する。
- この命令は、キャッシュ・ラインの分割がパフォーマンスに大きな影響を与える状況で、MOVDQU（ロード）命令を置き換えるものである。ストア / ロード・フォワーディングによる高いパフォーマンスが要求される状況では、この命令を使用するべきではない。ストア / ロード・フォワーディングのパフォーマンスが重要なアプリケーションでは、データのアライメントが 128 ビットに合っている場合は MOVDQA のストア / ロードのペアを使用し、データのアライメントが 128 ビットに合っていない場合は MOVDQU のストア / ロードのペアを使用する。

操作

```
xmm[127-0] = m128;
```

同等のインテル® C/C++ コンパイラ組み込み関数

HADDPS __m128i _mm_lddqu_si128(__m128i const *p)

LDDQU—Load Unaligned Integer 128 bits（続き）

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR4.OSFXSR（ビット 9）が 0 の場合。
	CR0.EM が 1 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

LDMXCSR—Load MXCSR Register

オペコード	命令	説明
0F AE /2	LDMXCSR *m32*	*m32* から MXCSR レジスタをロードする。

説明

ソース・オペランドをMXCSR制御/ステータス・レジスタへロードする。ソース・オペランドは、32ビット・メモリ・ロケーションである。MXCSRレジスタおよびその内容については、『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制御およびステータス・レジスタ」を参照のこと。

一般に、LDMXCSR命令はSTMXCSR命令と一緒に使用される。STMXCSR命令は、MXCSRレジスタの内容をメモリにストアする。

リセットしたときのMXCSRのデフォルト値は1F80Hである。

LDMXCSR命令によってSIMD浮動小数点例外マスクビットをクリアし、対応する例外フラグビットを設定した場合、SIMD浮動小数点例外はすぐには発生しない。この例外が発生するのは、この命令の後に引き続いて、特定のSIMD浮動小数点例外を報告させるSSE命令またはSSE2命令を実行したときだけである。

操作

```
MXCSR ← m32;
```

同等のインテル® C/C++ コンパイラ組み込み関数

_mm_setcsr(unsigned int i)

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、またはGSセグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	MXCSRの予約されているビットをセットしようとした場合。
#SS(0)	SSセグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0のTSがセットされた場合。

LDMXCSR—Load MXCSR Register（続き）

#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
	MXCSR の予約されているビットをセットしようとした場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

LDS/LES/LFS/LGS/LSS—Load Far Pointer

オペコード	命令	説明
C5 /*r*	LDS *r16*, *m16:16*	メモリから DS:*r16* に far ポインタをロードする。
C5 /*r*	LDS *r32*, *m16:32*	メモリから DS:*r32* に far ポインタをロードする。
0F B2 /*r*	LSS *r16*, *m16:16*	メモリから SS:*r16* に far ポインタをロードする。
0F B2 /*r*	LSS *r32*, *m16:32*	メモリから SS:*r32* に far ポインタをロードする。
C4 /*r*	LES *r16*, *m16:16*	メモリから ES:*r16* に far ポインタをロードする。
C4 /*r*	LES *r32*, *m16:32*	メモリから ES:*r32* に far ポインタをロードする。
0F B4 /*r*	LFS *r16*, *m16:16*	メモリから FS:*r16* に far ポインタをロードする。
0F B4 /*r*	LFS *r32*, *m16:32*	メモリから FS:*r32* に far ポインタをロードする。
0F B5 /*r*	LGS *r16*, *m16:16*	メモリから GS:*r16* に far ポインタをロードする。
0F B5 /*r*	LGS *r32*, *m16:32*	メモリから GS:*r32* に far ポインタをロードする。

説明

第 2 オペランド（ソース・オペランド）からセグメント・レジスタおよび第 1 オペランド（デスティネーション・オペランド）に far ポインタ（セグメント・セレクタとオフセット）をロードする。ソース・オペランドには、オペランド・サイズ属性の現在の設定（32 ビットまたは 16 ビット）に応じて、48 ビットまたは 32 ビットのメモリ内ポインタを指定する。命令のオペコードとデスティネーション・オペランドには、セグメント・レジスタ / 汎用レジスタペアを指定する。ソース・オペランドからの 16 ビット・セグメント・セレクタは、オペコードで指定されたセグメント・レジスタ（DS、SS、ES、FS、または GS）にロードされる。32 ビットまたは 16 ビットのオフセットは、デスティネーション・オペランドで指定されたレジスタにロードされる。

これらの命令の 1 つを保護モードで実行した場合は、ソース・オペランドのセグメント・セレクタが指しているセグメント・ディスクリプタからの追加情報が、選択されたセグメント・レジスタの隠蔽部分にロードされる。

保護モードでも、保護例外を発生させることなく、DS、ES、FS、または GS レジスタにヌルセレクタ（0000 ～ 0003 の値）をロードすることができる。（ロードされると、以降は、対応するセグメント・レジスタにヌルセレクタがロードされているセグメントを参照しようとすると、必ず一般保護例外（#GP）が発生し、そのセグメントへのメモリ参照は行われない。）

LDS/LES/LFS/LGS/LSS—Load Far Pointer（続き）

操作

```
IF ProtectedMode
    THEN IF SS is loaded
        THEN IF SegementSelector = null
            THEN #GP(0);
        FI;
        ELSE IF Segment selector index is not within descriptor table limits
        OR Segment selector RPL ≠ CPL
        OR Access rights indicate nonwritable data segment
        OR DPL ≠ CPL
            THEN #GP(selector);
        FI;
        ELSE IF Segment marked not present
            THEN #SS(selector);
        FI;
        SS ← SegmentSelector(SRC);
        SS ← SegmentDescriptor([SRC]);
    ELSE IF DS, ES, FS, or GS is loaded with non-null segment selector
        THEN IF Segment selector index is not within descriptor table limits
        OR Access rights indicate segment neither data nor readable code segment
        OR (Segment is data or nonconforming-code segment
            AND both RPL and CPL > DPL)
            THEN #GP(selector);
        FI;
        ELSE IF Segment marked not present
            THEN #NP(selector);
        FI;
        SegmentRegister ← SegmentSelector(SRC) AND RPL;
        SegmentRegister ← SegmentDescriptor([SRC]);
    ELSE IF DS, ES, FS, or GS is loaded with a null selector:
        SegmentRegister ← NullSelector;
        SegmentRegister(DescriptorValidBit) ← 0; (*hidden flag; not accessible by
            software*)
    FI;
FI;
IF (Real-Address or Virtual-8086 Mode)
    THEN
        SegmentRegister ← SegmentSelector(SRC);
FI;
DEST ← Offset(SRC);
```

影響を受けるフラグ

なし。

LDS/LES/LFS/LGS/LSS—Load Far Pointer（続き）

保護モード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。
#GP(0)	SS レジスタにヌルセレクタがロードされた場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#GP（セレクタ）	SS レジスタがロードされようとしたとき、次のいずれかが真であった場合。(1) セグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲内でない。(2) セグメント・セレクタの RPL が CPL に等しくない。(3) セグメントが書き込み不可能なデータ・セグメントである。(4) DPL が CPL に等しくない。
	DS、ES、FS、または GS レジスタにヌルでないセグメント・セレクタがロードされようとしたとき、次のいずれかが真であった場合。(1) セグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲内でない。(2) セグメントがデータ・セグメントでも読み出し可能なコード・セグメントでもない。(3) セグメントがデータ・セグメントまたは非コンフォーミング・コード・セグメントであり、かつ RPL と CPL が共に DPL より大きい。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#SS（セレクタ）	SS レジスタがロードされようとしたとき、セグメントが存在しないとマークされていた場合。
#NP（セレクタ）	DS、ES、FS、または GS レジスタにヌルでないセグメント・セレクタがロードされようとしたとき、セグメントが存在しないとマークされていた場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#UD	ソース・オペランドがメモリ・ロケーションでない場合。

LDS/LES/LFS/LGS/LSS—Load Far Pointer（続き）

仮想 8086 モード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

LEA—Load Effective Address

オペコード	命令	説明
8D /r	LEA *r16*, *m*	*m* の実効アドレスをレジスタ *r16* にストアする。
8D /r	LEA *r32*, *m*	*m* の実効アドレスをレジスタ *r32* にストアする。

説明

第2オペランド（ソース・オペランド）の実効アドレスを計算し、第1オペランド（デスティネーション・オペランド）にストアする。ソース・オペランドは、プロセッサのアドレス指定モードの1つで指定されたメモリアドレス（オフセット部）である。デスティネーション・オペランドは、汎用レジスタである。この命令が実行する処理は、以下の表に示すように、アドレスサイズ属性およびオペランド・サイズ属性によって変わる。この命令のオペランド・サイズ属性は選択されたレジスタによって決まり、アドレスサイズ属性はコード・セグメントの属性によって決まる。

オペランド・サイズ	アドレスサイズ	実行される処理
16	16	16 ビットの実効アドレスが計算され、要求された 16 ビット・レジスタのデスティネーションにストアされる。
16	32	32 ビットの実効アドレスが計算される。そのアドレスの下位 16 ビットが要求された 16 ビット・レジスタのデスティネーションにストアされる。
32	16	16 ビットの実効アドレスが計算される。16 ビット・アドレスはゼロ拡張され、要求された 32 ビット・レジスタのデスティネーションにストアされる。
32	32	32 ビットの実効アドレスが計算され、要求された 32 ビット・レジスタのデスティネーションにストアされる。

アセンブラが異なると、ソース・オペランドのサイズ属性と記号参照に基づいて、使用されるアルゴリズムが変わることもある。

LEA—Load Effective Address（続き）

操作

```
IF OperandSize = 16 AND AddressSize = 16
    THEN
        DEST ← EffectiveAddress(SRC); (* 16-bit address *)
    ELSE IF OperandSize = 16 AND AddressSize = 32
        THEN
            temp ← EffectiveAddress(SRC); (* 32-bit address *)
            DEST ← temp[0..15]; (* 16-bit address *)
    ELSE IF OperandSize = 32 AND AddressSize = 16
        THEN
            temp ← EffectiveAddress(SRC); (* 16-bit address *)
            DEST ← ZeroExtend(temp); (* 32-bit address *)
    ELSE IF OperandSize = 32 AND AddressSize = 32
        THEN
            DEST ← EffectiveAddress(SRC); (* 32-bit address *)
    FI;
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。

実アドレスモード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。

仮想 8086 モード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。

LEAVE—High Level Procedure Exit

オペコード	命令	説明
C9	LEAVE	SP を BP に設定し、次に BP をポップする。
C9	LEAVE	ESP を EBP に設定し、次に EBP をポップする。

説明

先行の ENTER 命令によってセットアップされたスタックフレームをリリースする。LEAVE 命令は、（EBP レジスタ内の）フレームポインタをスタック・ポインタ・レジスタ（ESP）にコピーし、それによってスタックフレームに割り当てられていたスタック空間がリリースされる。次に、古いフレームポインタ（ENTER 命令によってセーブされたコール元プロシージャのフレームポインタ）がスタックから EBP レジスタにポップされ、コール元プロシージャのスタックフレームがリストアされる。

一般的に、LEAVE 命令の次に RET 命令が実行されて、プログラム制御がコール元プロシージャに返される。

ENTER 命令および LEAVE 命令の使用の詳細については、『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 6 章の「ブロック構造言語でのプロシージャ・コール」を参照のこと。

操作

```
IF StackAddressSize = 32
    THEN
        ESP ← EBP;
    ELSE (* StackAddressSize = 16*)
        SP ← BP;
FI;
IF OperandSize = 32
    THEN
        EBP ← Pop();
    ELSE (* OperandSize = 16*)
        BP ← Pop();
FI;
```

影響を受けるフラグ

なし。

LEAVE—High Level Procedure Exit（続き）

保護モード例外

#SS(0)	EBP レジスタの指示先のロケーションが現在のスタック・セグメントの範囲内にない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	EBP レジスタの指示先のロケーションが 0 ～ FFFFH の実効アドレス空間の範囲外の場合。

仮想 8086 モード例外

#GP(0)	EBP レジスタの指示先のロケーションが 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

LES—Load Full Pointer

「LDS/LES/LFS/LGS/LSS—Load Far Pointer」を参照のこと。

LFENCE—Load Fence

オペコード	命令	説明
0F AE /5	LFENCE	ロード操作をシリアル化する。

説明

LFENCE 命令より前に発行されたすべてのメモリからのロード命令に対して、シリアル化操作を実行する。このシリアル化操作は、プログラムの順序で LFENCE 命令に先行するすべてのロード命令が、LFENCE 命令に後続するロード命令より前に、グローバルにアクセス可能になることを保証する。LFENCE 命令は、ロード命令、他の LFENCE 命令、すべての MFENCE 命令、およびすべてのシリアル化命令（CPUID 命令など）に対して順序付けされる。LFENCE 命令は、ストア命令や SFENCE 命令に対しては順序付けされない。

順序設定の緩いメモリタイプを使用して、アウト・オブ・オーダー発行や見込み的な読み込みなどの手法により、プロセッサ・パフォーマンスの向上を達成する。データを参照する側のルーチンが、順序設定の緩いデータであることをどの程度認識しているかは、アプリケーションによって異なり、データを生成する側のルーチンにはわからない。LFENCE 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参照するルーチンの間のロードの順序付けを保証するための効率的な方法である。

プロセッサは、見込み的な読み込みが許されるメモリタイプ（すなわち、WB、WC、WT メモリタイプ）が割り当てられたシステムメモリ領域から、いつでもデータを見込み的にフェッチしてキャッシュに入れることができる。PREFETCH*h* 命令は、この見込み的な動作に対するヒントと見なされる。この見込み的なフェッチ動作は、命令の実行には拘束されず、任意の時点で発生する。したがって、LFENCE 命令は、PREFETCH*h* 命令などの見込み的なフェッチ機構に対して順序付けされない（つまり、LFENCE 命令の実行の直前、実行中、または実行後に、データがキャッシュに見込み的にロードされる可能性がある）。

操作

```
Wait_On_Following_Loads_Until(preceding_loads_globally_visible);
```

同等のインテル® C/C++ コンパイラ組み込み関数

void_mm_lfence(void)

例外（すべての動作モード）

なし。

LFS—Load Full Pointer

「LDS/LES/LFS/LGS/LSS—Load Far Pointer」を参照のこと。

LGDT/LIDT—Load Global/Interrupt Descriptor Table Register

オペコード	命令	説明
0F 01 /2	LGDT *m16&32*	*m* を GDTR にロードする。
0F 01 /3	LIDT *m16&32*	*m* を IDTR にロードする。

説明

ソース・オペランド内の値をグローバル・ディスクリプタ・テーブル・レジスタ（GDTR）または割り込みディスクリプタ・テーブル・レジスタ（IDTR）にロードする。ソース・オペランドでは、グルーバル・ディスクリプタ・テーブル（GDT）または割り込みディスクリプタ・テーブル（IDT）のベースアドレス（リニアアドレス）と範囲（バイト単位のテーブルサイズ）を内容とする 6 バイトのメモリ・ロケーションを指定する。オペランド・サイズ属性が 32 ビットである場合は、16 ビットの範囲（6 バイトのデータ・オペランドの下位 2 バイト）と 32 ビットのベースアドレス（データ・オペランドの上位 4 バイト）がレジスタにロードされる。オペランド・サイズ属性が 16 ビットである場合は、16 ビットの範囲（下位 2 バイト）と 24 ビットのベースアドレス（第 3、第 4、第 5 バイト）がロードされる。この場合は、オペランドの上位バイトは使用されず、GDTR または IDTR のベースアドレスの上位バイトにはゼロが埋められる。

LGDT 命令および LIDT 命令は、オペレーティング・システム・ソフトウェアだけで使用される。これらの命令は、アプリケーション・プログラムでは使用されない。保護モードでリニアアドレス（すなわち、セグメントに相対的でないアドレス）と範囲を直接ロードするのはこれらの命令だけである。LGDT 命令および LIDT 命令は、一般的に、保護モードへの切り替え前にプロセッサの初期化を可能にするために、実アドレスモードで実行される。

GDTR と IDTR の内容のストアについては、本章の「SGDT—Store Global Descriptor Table Register」を参照のこと。

操作

```
IF instruction is LIDT
    THEN
        IF OperandSize = 16
            THEN
                IDTR(Limit) ← SRC[0:15];
                IDTR(Base) ← SRC[16:47] AND 00FFFFFFH;
            ELSE (* 32-bit Operand Size *)
                IDTR(Limit) ← SRC[0:15];
                IDTR(Base) ← SRC[16:47];
        FI;
    ELSE (* instruction is LGDT *)
        IF OperandSize = 16
```

LGDT/LIDT—Load Global/Interrupt Descriptor Table Register（続き）

```
        THEN
            GDTR(Limit) ← SRC[0:15];
            GDTR(Base) ← SRC[16:47] AND 00FFFFFFH;
        ELSE (* 32-bit Operand Size *)
            GDTR(Limit) ← SRC[0:15];
            GDTR(Base) ← SRC[16:47];
    FI; FI;
```

影響を受けるフラグ

なし。

保護モード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。
#GP(0)	現行特権レベルが 0 でない場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#UD	ソース・オペランドがメモリ・ロケーションでない場合。
#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	LGDT 命令と LIDT 命令は、仮想 8086 モードでは認識されない。
#GP	現行特権レベルが 0 でない場合。

LLDT—Load Local Descriptor Table Register

オペコード	命令	説明
0F 00 /2	LLDT *r/m16*	セグメント・セレクタ *r/m16* を LDTR にロードする。

説明

ソース・オペランドをローカル・ディスクリプタ・テーブル・レジスタ（LDTR）のセグメント・セレクタ・フィールドにロードする。ソース・オペランド（汎用レジスタまたはメモリ・ロケーション）の内容 は、ローカル・ディスクリプタ・テーブル（LDT）を指示先とするセグメント・セレクタである。セグメント・セレクタが LDTR にロードされた後に、プロセッサはそのセグメント・セレクタを使用して、グローバル・ディスクリプタ・テーブル（GDT）内にある LDT のセグメント・ディスクリプタの位置を探す。次に、セグメント・ディスクリプタから LDTR に LDT のセグメント範囲とベースアドレスをロードする。DS、ES、SS、FS、GS、CS の各セグメント・レジスタはこの命令の影響を受けない。現在のタスクに対するタスク・ステート・セグメント（TSS）内の LDTR フィールドも影響を受けない。

ソース・オペランドが 0 である場合は、LDTR は無効とマークされ、LDT 内のディスクリプタへのすべての参照（LAR、VERR、VERW、または LSL 命令によるものを除く）によって一般保護例外（#GP）が発生する。

オペランド・サイズ属性は、この命令には効果をもたない。

LLDT 命令は、オペレーティング・システム・ソフトウェアで使用するために設けられたものであり、アプリケーション・プログラムでは使用してはならない。さらに、この命令は保護モードでしか実行することができない。

操作

```
IF SRC[Offset) > descriptor table limit THEN #GP(segment selector); FI;
Read segment descriptor;
IF SegmentDescriptor(Type) ≠ LDT THEN #GP(segment selector); FI;
IF segment descriptor is not present THEN #NP(segment selector);
LDTR(SegmentSelector) ← SRC;
LDTR(SegmentDescriptor) ← GDTSegmentDescriptor;
```

影響を受けるフラグ

なし。

LLDT—Load Local Descriptor Table Register（続き）

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#GP（セレクタ）	セレクタ・オペランドの指示先がグローバル・ディスクリプタ・テーブル（GDT）でない場合、または GDT 内のエントリがローカル・ディスクリプタ・テーブルでない場合。
	セグメント・セレクタが GDT の範囲を超えている場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NP（セレクタ）	LDT ディスクリプタが存在しない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#UD	LLDT 命令は実アドレスモードでは認識されない。

仮想 8086 モード例外

#UD	LLDT 命令は仮想 8086 モードでは認識されない。

LIDT—Load Interrupt Descriptor Table Register

「LGDT/LIDT—Load Global/Interrupt Descriptor Table Register」を参照のこと。

LMSW—Load Machine Status Word

オペコード	命令	説明
0F 01 /6	LMSW *r/m16*	*r/m16*をCR0のマシン・ステータス・ワードにロードする。

説明

ソース・オペランドをマシン・ステータス・ワード、すなわちCR0レジスタのビット0～15にロードする。ソース・オペランドには、16ビット汎用レジスタまたはメモリ・ロケーションを使用できる。ソース・オペランドの下位4ビット（PE、MP、EM、TSフラグ）だけがCR0にロードされる。CR0のPG、CD、NW、AM、WP、NE、ETフラグは影響を受けない。オペランド・サイズ属性は、この命令には効果をもたない。

ソース・オペランドのPEフラグ（ビット0）が1にセットされている場合は、この命令によってプロセッサは保護モードに切り替わる。保護モードでは、LMSW命令を使用してPEフラグをクリアし、実アドレスモードへのスイッチバックを強制することはできない。

LMSW命令は、オペレーティング・システム・ソフトウェアで使用するために設けられたものであり、アプリケーション・プログラムでは使用してはならない。保護モードまたは仮想8086モードでは、この命令はCPL 0でしか実行することができない。

この命令は、インテル® 286プロセッサとの互換性を保つために設けられたものである。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、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサで実行するように意図されたプログラムおよびプロシージャでは、MOV（制御レジスタ）命令を使用してCR0レジスタ全体をロードする必要がある。MOV CR0命令を使用すると、CR0のPEフラグをセットおよびクリアすることができ、プログラムやプロシージャによる保護モードと実アドレスモードとの間の切り替えが可能になる。

この命令はシリアル化命令である。

操作

```
CR0[0:3] ← SRC[0:3];
```

影響を受けるフラグ

なし。

LMSW—Load Machine Status Word（続き）

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

LOCK—Assert LOCK# Signal Prefix

オペコード	命令	説明
F0	LOCK	後の命令の実行中 LOCK# 信号をアサートする。

説明

このプリフィックスの後の命令の実行中、プロセッサの LOCK# 信号をアサートさせる（すなわち、その命令をアトム命令に変える）。マルチプロセッサ環境では、LOCK# 信号は、それがアサートされている間プロセッサが任意の共有メモリを独占的に使用できるよう保証する。

（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・プロセッサを含む）後から現れた IA-32 プロセッサでは、LOCK# 信号をアサートさせないでロックを行うこともできる。下記の「IA-32 アーキテクチャにおける互換性」の項を参照のこと。

LOCK プリフィックスは、ADD、ADC、AND、BTC、BTR、BTS、CMPXCHG、CMPXCH8B、DEC、INC、NEG、NOT、OR、SBB、SUB、XOR、XADD、XCHG の各命令の前、および、デスティネーション・オペランドがメモリ・オペランドである形式の命令の前にのみ使用できる。LOCK プリフィックスがこれらの命令のいずれかに使用され、しかもソース・オペランドがメモリ・オペランドである場合には、未定義のオペコード例外（#UD）が発生することがある。LOCK プリフィックスを上記以外の命令に使用した場合は、未定義のオペコード例外も発生する。XCHG 命令は、LOCK プリフィックスの有無に関係なく、常に LOCK# 信号をアサートする。

一般的に、LOCK プリフィックスは、共有メモリ環境内のメモリ・ロケーションでの読み取り / 修正 / 書き込み操作を実行する場合に BTS 命令に付けて使用される。

LOCK プリフックスの有効性はメモリ・フィールドのアライメントによって損なわれない。アライメントがどのようにずれているメモリ・フィールドに対しても、メモリロックは変わりなく有効である。

IA-32 アーキテクチャにおける互換性

P6 ファミリ・プロセッサ以降では、命令の前に LOCK プリフィックスを付けて、アクセスするメモリ領域をプロセッサ内で内部的にキャッシュさせるときは、一般的に LOCK# 信号はアサートされない。その代わりに、プロセッサのキャッシュだけがロックされる。その場合、プロセッサのキャッシュ・コヒーレンシのメカニズムによって、メモリに関しては操作がアトミックに実行されるよう保証される。キャッシュのロック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、下巻』の第 7 章の「プロセッサ内部キャッシュ上のロック操作」を参照のこと。

LOCK—Assert LOCK# Signal Prefix（続き）

操作

AssertLOCK#(DurationOfAccompaningInstruction)

影響を受けるフラグ

なし。

保護モード例外

#UD	LOCK プリフィックスが、上記の「説明」の項に示していない命令に使用された場合。この、LOCK プリフックスが適用されようとした命令からは、他にも例外が発生する可能性がある。

実アドレスモード例外

#UD	LOCK プリフィックスが、上記の「説明」の項に示していない命令に使用された場合。この、LOCK プリフックスが適用されようとした命令からは、他にも例外が発生する可能性がある。

仮想 8086 モード例外

#UD	LOCK プリフィックスが、上記の「説明」の項に示していない命令に使用された場合。この、LOCK プリフックスが適用されようとした命令からは、他にも例外が発生する可能性がある。

LODS/LODSB/LODSW/LODSD—Load String

オペコード	命令	説明
AC	LODS m8	アドレス DS:(E)SI のバイトを AL にロードする。
AD	LODS m16	アドレス DS:(E)SI のワードを AX にロードする。
AD	LODS m32	アドレス DS:(E)SI のダブルワードを EAX にロードする。
AC	LODSB	アドレス DS:(E)SI のバイトを AL にロードする。
AD	LODSW	アドレス DS:(E)SI のワードを AX にロードする。
AD	LODSD	アドレス DS:(E)SI のダブルワードを EAX にロードする。

説明

ソース・オペランドからAL、AX、またはEAXレジスタに、バイト、ワード、またはダブルワードをロードする。ソース・オペランドはメモリ・ロケーションであり、そのアドレスは、(命令のアドレスサイズ属性、32または16に応じて）それぞれDS:EDIまたはDS:SIレジスタから読み取られる。DSセグメントは、セグメント・オーバーライド・プリフィックスでオーバーライドすることができる。

アセンブリ・コード・レベルでは、この命令の「明示オペランド」形式および「オペランドなし」形式という2つの形式が使用できる。(LODSニーモニックで指定される）明示オペランド形式では、ソース・オペランドを明示的に指定することができる。この場合、ソース・オペランドは、ソース値のサイズとロケーションを示す記号でなければならない。デスティネーション・オペランドは、この場合にはソース・オペランドのサイズに一致するように自動的に選択される（バイト・オペランドではAL、ワード・オペランドではAX、ダブルワード・オペランドではEAX)。この明示オペランド形式は、ドキュメンテーションを可能にするために設けられたものであるが、この形式によって提供されるドキュメンテーションは誤解を招く場合があるので注意する。すなわち、ソース・オペランドの記号は、オペランドの正しい**タイプ**（サイズ : バイト、ワード、またはダブルワード）を指定しなければならないが、正しい**ロケーション**を指定する必要はない。ロケーションは、常にDS:(E)SIレジスタによって指定されるので、ストリング・ロード命令が実行される前に、これらのレジスタに正しくロードされていなければならない。

オペランドなし形式は、LODS命令のバイト、ワード、ダブルワード各バージョンの「ショート形式」を提供する。この場合も、DS:(E)SIがソース・オペランドであると想定され、AL、AX、またはEAXレジスタがデスティネーション・オペランドであると想定される。ソース・オペランドとデスティネーション・オペランドのサイズは、LODSB（ALレジスタへのバイトのロード)、LODSW（AXレジスタへのワードのロード)、またはLODSD（EAXレジスタへのダブルワードのロード）の各ニーモニックで選択される。

LODS/LODSB/LODSW/LODSD—Load String（続き）

バイト、ワード、またはダブルワードがメモリ・ロケーションからAL、AX、またはEAXレジスタに転送された後、(E)SIレジスタはEFLAGSレジスタ内のDFフラグの設定にしたがって自動的にインクリメントまたはデクリメントされる。（DFフラグが0である場合は、(E)SIレジスタはインクリメントされる。DFフラグが1である場合は、(E)SIレジスタはデクリメントされる。）(E)SIレジスタは、バイト操作の場合は1、ワード操作の場合は2、またダブルワードの場合は4、それぞれインクリメントまたはデクリメントされる。

LODS、LODSB、LODSW、LODSD命令は、前にREPプリフィックスを付けることにより、ECXバイト、ワード、またはダブルワードのブロックロードを行うことができる。しかし通常は、レジスタに転送されたデータを次の転送が行われる前にさらに処理する必要があるので、これらの命令はLOOP構成体で使用されることの方が多い。REPプリフィックスの説明については、本章の「REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix」を参照のこと。

操作

```
IF (byte load)
    THEN
        AL ← SRC; (* byte load *)
            THEN IF DF = 0
                THEN (E)SI ← (E)SI + 1;
                ELSE (E)SI ← (E)SI – 1;
            FI;
    ELSE IF (word load)
        THEN
            AX ← SRC; (* word load *)
                THEN IF DF = 0
                    THEN (E)SI ← (E)SI + 2;
                    ELSE (E)SI ← (E)SI – 2;
                FI;
        ELSE (* doubleword transfer *)
            EAX ← SRC; (* doubleword load *)
                THEN IF DF = 0
                    THEN (E)SI ← (E)SI + 4;
                    ELSE (E)SI ← (E)SI – 4;
                FI;
    FI;
FI;
```

影響を受けるフラグ

なし。

LODS/LODSB/LODSW/LODSD—Load String（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

LOOP/LOOP*cc*—Loop According to ECX Counter

オペコード	命令	説明
E2 cb	LOOP *rel8*	カウントをデクリメントし、カウント ≠ 0 の場合 short ジャンプする。
E1 cb	LOOPE *rel8*	カウントをデクリメントし、カウント ≠ 0 でかつ ZF=1 の場合 short ジャンプする。
E1 cb	LOOPZ *rel8*	カウントをデクリメントし、カウント ≠ 0 でかつ ZF=1 の場合 short ジャンプする。
E0 cb	LOOPNE *rel8*	カウントをデクリメントし、カウント ≠ 0 でかつ ZF=0 の場合 short ジャンプする。
E0 cb	LOOPNZ *rel8*	カウントをデクリメントし、カウント ≠ 0 でかつ ZF=0 の場合 short ジャンプする。

説明

ECX または CX レジスタをカウンタとして使用して、ループ操作を実行する。LOOP 命令が実行されるたびに、カウントレジスタが 1 デクリメントされ、次に 0 かどうかチェックされる。カウントが 0 である場合は、ループは終了し、LOOP 命令の次の命令からプログラムの実行が継続される。カウントが 0 でない場合は、デスティネーション（ターゲット）オペランドへの near ジャンプが行われる。ターゲットは恐らくはループの先頭である。アドレスサイズ属性が 32 ビットである場合は、カウントレジスタとして ECX レジスタが使用される。そうでない場合は、CX レジスタが使用される。

ターゲット命令は相対オフセット（EIP レジスタ内の命令ポインタの現在値に相対的な符号付きオフセット）で指定される。相対オフセットは、アセンブリ・コードでは一般的にラベルとして指定されるが、マシン・コード・レベルでは、符号付きの 8 ビット即値としてコード化され、命令ポインタに加算される。この命令には、–128 ～ +127 までのオフセットが使用できる。

ループ命令（LOOP*cc*）の一部の形式では、カウントが 0 に達する前にループを終了させるための条件として ZF フラグも指定できる。それらの形式の命令には、各命令に条件コード（*cc* が対応しており、それぞれにテストされる条件を示している。その場合、LOOP*cc* 命令そのものは ZF フラグの状態を変えない。ZF フラグを変更するのは、ループ内の他の命令である。

LOOP/LOOP*cc*—Loop According to ECX Counter（続き）

操作

```
IF AddressSize = 32
    THEN
        Count is ECX;
    ELSE (* AddressSize = 16 *)
        Count is CX;
FI;
Count ← Count – 1;

IF instruction is not LOOP
    THEN
        IF (instruction ← LOOPE) OR (instruction ← LOOPZ)
            THEN
                IF (ZF =1) AND (Count ≠ 0)
                    THEN BranchCond ← 1;
                    ELSE BranchCond ← 0;
                FI;
        FI;
        IF (instruction ← LOOPNE) OR (instruction ← LOOPNZ)
            THEN
                IF (ZF =0 ) AND (Count ≠ 0)
                    THEN BranchCond ← 1;
                    ELSE BranchCond ← 0;
                FI;
        FI;
    ELSE (* instruction = LOOP *)
        IF (Count ≠ 0)
            THEN BranchCond ← 1;
            ELSE BranchCond ← 0;
        FI;
FI;
IF BranchCond = 1
    THEN
        EIP ← EIP + SignExtend(DEST);
        IF OperandSize = 16
            THEN
                EIP ← EIP AND 0000FFFFH;
            ELSE (* OperandSize = 32 *)
                IF EIP < CS.Base OR EIP > CS.Limit
                    #GP
        FI;
    ELSE
        Terminate loop and continue program execution at EIP;
FI;
```

影響を受けるフラグ

なし。

LOOP/LOOP*cc*—Loop According to ECX Counter（続き）

保護モード例外

#GP(0)	ジャンプ先のオフセットが CS セグメントの範囲を超えている場合。

実アドレスモード例外

#GP	ジャンプ先のオフセットが CS セグメントの範囲を超えている場合、または、0 ～ FFFFH の実効アドレス空間から外れている場合。この状態は、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た場合に起きることがある。

仮想 8086 モード例外

実アドレスモードと同じ例外。

LSL—Load Segment Limit

オペコード	命令	説明
0F 03 /*r*	LSL *r16*, *r/m16*	"*r16* ← セグメント範囲、セレクタ *r/m16* " のロードを行う。
0F 03 /*r*	LSL *r32*, *r/m32*	"*r32* ← セグメント範囲、セレクタ *r/m32* " のロードを行う。

説明

スクランブルされていないセグメント範囲を第 2 オペランド（ソース・オペランド）で指定されたセグメント・ディスクリプタから第 1 オペランド（デスティネーション・オペランド）にロードし、EFLAGS レジスタ内の ZF フラグをセットする。ソース・オペランド（レジスタまたはメモリ・ロケーション）の内容は、アクセスされるセグメント・ディスクリプタのセグメント・セレクタである。デスティネーション・オペランドは、汎用レジスタである。

プロセッサは、ロードプロセスの一環としてアクセスチェックを行う。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範囲をポインタのオフセットと比較することができる。

セグメント範囲は、セグメント・ディスクリプタのバイト 0 と 1、およびバイト 6 の最初の 4 ビットを含む 20 ビットの値である。ディスクリプタのセグメント範囲がバイトを最小単位とする（グラニュラリティ・フラグが 0 にされている）場合は、デスティネーション・オペランドにバイト単位値（バイト範囲）がロードされる。ディスクリプタのセグメント範囲がページを最小単位とする（グラニュラリティ・フラグが 1 にセットされている）場合は、LSL 命令はセグメント範囲をページ単位範囲（ページ範囲）からバイト範囲に変換してから、それをデスティネーション・オペランドにロードする。この変換は、20 ビットの「生」の範囲を左に 12 ビットシフトし、下位 12 ビットを 1 で埋めて行われる。

オペランド・サイズが 32 ビットのときは、32 ビットのバイト範囲がデスティネーション・オペランドにストアされる。オペランド・サイズが 16 ビットのときは、有効な 32 ビットのバイト範囲が計算されるが、上位 16 ビットは切り捨てられ、下位 16 ビットだけがデスティネーション・オペランドにロードされる。

この命令では、セグメント範囲をデスティネーション・レジスタにロードする前に以下のチェックを行う。

- セグメント・セレクタがヌルでないことを確認する。
- セグメント・セレクタの指示先がアクセスされる GDT または LDT の範囲内のディスクリプタであることを確認する。

LSL—Load Segment Limit（続き）

- ディスクリプタのタイプがこの命令に対して有効であることを確認する。LSL 命令に対しては、すべてのコード・セグメント・ディスクリプタとデータ・セグメント・ディスクリプタが有効である（すなわちLSL命令でアクセスできる）。有効な特殊なセグメントおよびゲートのディスクリプタ・タイプが下記の表に示してある。
- 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、この命令は指定されたセグメント・ディスクリプタがCPLでアクセスできる（すなわち、CPLとセグメント・セレクタのRPLがセグメント・セレクタのDPL以下である）かを確認する。

セグメント・ディスクリプタがアクセスできないか、またはこの命令にとって無効なタイプである場合は、ZFフラグがクリアされ、値は何もデスティネーション・オペランドにロードされない。

タイプ	名前	有効 / 無効
0	予約済み	無効
1	使用可能 16 ビット TSS	有効
2	LDT	有効
3	ビジー 16 ビット TSS	有効
4	16 ビット・コール・ゲート	無効
5	16 ビット /32 ビット・タスク・ゲート	無効
6	16 ビット割り込みゲート	無効
7	16 ビット・トラップ・ゲート	無効
8	予約済み	無効
9	使用可能 32 ビット TSS	有効
A	予約済み	無効
B	ビジー 32 ビット TSS	有効
C	32 ビット・コール・ゲート	無効
D	予約済み	無効
E	32 ビット割り込みゲート	無効
F	32 ビット・トラップ・ゲート	無効

LSL—Load Segment Limit（続き）

操作

```
IF SRC[Offset) > descriptor table limit
    THEN ZF ← 0; FI;
Read segment descriptor;
IF SegmentDescriptor(Type) ≠ conforming code segment
    AND (CPL > DPL) OR (RPL > DPL)
    OR Segment type is not valid for instruction
        THEN
            ZF ← 0
        ELSE
            temp ← SegmentLimit([SRC]);
            IF (G ← 1)
                THEN
                    temp ← ShiftLeft(12, temp) OR 00000FFFH;
            FI;
            IF OperandSize = 32
                THEN
                    DEST ← temp;
                ELSE (*OperandSize = 16*)
                    DEST ← temp AND FFFFH;
            FI;
FI;
```

影響を受けるフラグ

セグメント範囲が正常にロードされた場合はZFフラグが1にセットされ、ロードできなかった場合は0にセットされる。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスがCS、DS、ES、FS、またはGSセグメントの範囲外の場合。
	DS、ES、FS、またはGSレジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#SS(0)	メモリ・オペランドの実効アドレスがSSセグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが3のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#UD	LSL命令は実アドレスモードでは認識されない。

仮想8086モード例外

#UD	LSL命令は仮想8086モードでは認識されない。

LSS—Load Full Pointer

「LDS/LES/LFS/LGS/LSS—Load Far Pointer」を参照のこと。

LTR—Load Task Register

オペコード	命令	説明
0F 00 /3	LTR *r/m16*	*r/m16* をタスクレジスタにロードする。

説明

ソース・オペランドをタスクレジスタのセグメント・セレクタ・フィールドにロードする。ソース・オペランド（汎用レジスタまたはメモリ・ロケーション）の内容は、タスク・ステート・セグメント（TSS）を指示先とするセグメント・セレクタである。セグメント・セレクタがタスクレジスタにロードされた後に、プロセッサはそのセグメント・セレクタを使用して、グローバル・ディスクリプタ・テーブル（GDT）内にあるTSSのセグメント・ディスクリプタの位置を探す。次に、セグメント・ディスクリプタからタスクレジスタにTSSのセグメント範囲とベースアドレスをロードする。タスクレジスタの指示先のタスクはビジーとマークされが、そのタスクへの切り替えは行われない。

LTR命令はオペレーティング・システム・ソフトウェアで使用するために設けられたものであり、アプリケーション・プログラムでは使用してはならない。この命令は、CPLが0のときは保護モードでしか実行することができない。LTR命令は、最初に実行されるタスクを設定する初期化コードでよく使用される。

オペランド・サイズ属性は、この命令に影響を与えない。

操作

```
IF SRC[Offset) > descriptor table limit OR IF SRC[type) ≠ global
    THEN #GP(segment selector);
FI;
Read segment descriptor;
IF segment descriptor is not for an available TSS THEN #GP(segment selector); FI;
IF segment descriptor is not present THEN #NP(segment selector);
TSSsegmentDescriptor(busy) ← 1;
(* Locked read-modify-write operation on the entire descriptor when setting busy flag *)
TaskRegister(SegmentSelector) ← SRC;
TaskRegister(SegmentDescriptor) ← TSSSegmentDescriptor;
```

影響を受けるフラグ

なし。

LTR—Load Task Register（続き）

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタを使用してメモリがアクセスされ、レジスタの内容がヌル・セグメント・セレクタであった場合。
#GP（セレクタ）	ソースセレクタの指示先のセグメントが TSS でないか、またはそのセグメントのタスクがすでにビジーの場合。
	セレクタが LDT を指しているか、または GDT の範囲を超えている場合。
#NP（セレクタ）	TSS が存在しないとマークされている場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#UD	LTR 命令は実アドレスモードでは認識されない。

仮想 8086 モード例外

#UD	LTR 命令は仮想 8086 モードでは認識されない。

MASKMOVDQU—Store Selected Bytes of Double Quadword

オペコード	命令	説明
66 0F F7 /r	MASKMOVDQU *xmm1*, *xmm2*	*xmm2* のバイトマスクを使用して、*xmm1* のバイトを選択した上でメモリ・ロケーションに書き込む。

説明

ソース・オペランド（第1オペランド）の選択されたバイトを、128ビットのメモリ・ロケーションにストアする。マスク・オペランド（第2オペランド）は、ソース・オペランド内のどのバイトがメモリに書き込まれるかを選択する。ソース・オペランドとマスク・オペランドはXMMレジスタである。メモリ・ロケーションの最初のバイトの位置は、DI/EDIレジスタとDSレジスタで指定される。メモリ・ロケーションのアライメントは、自然境界に合わせる必要はない（ストアアドレスのサイズは、アドレスサイズ属性によって異なる）。

マスク・オペランドの各バイトの最上位ビットは、ソース・オペランド内の対応するバイトがメモリ内の対応するバイト位置に書き込まれるかどうかを指定する。0の場合は書き込みは行われず、1の場合は書き込まれる。

MASKMOVEDQU命令は、キャッシュの汚染を最小限に抑えるために、プロセッサに対して非テンポラルなヒントを生成する。非テンポラルなヒントは、WC（Write Combining）メモリタイプのプロトコルを使用して実現される（詳しくは、『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「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと）。WCプロトコルでは、順序設定のゆるいメモリ整合性モデルを使用する。そのため、複数のプロセッサが異なるメモリタイプを使用してデスティネーション・メモリ・ロケーションの読み込み / 書き込みを実行する可能性がある場合は、SFENCE命令またはMFENCE命令で実現されるフェンス操作をMASKMOVEDQU命令と一緒に使用する必要がある。

マスクがすべて0の場合、この命令の動作は次のようになる。

- データはメモリに書き込まれない。
- ブレークポイント（コードまたはデータ）の報告は保証されていない。これらのブレークポイントが報告されるかどうかは、プロセッサによって異なる。
- この場合も、メモリのアドレス指定に関連する例外とページフォルトは報告されることがある（プロセッサによって異なる）。

MASKMOVDQU—Store Selected Bytes of Double Quadword（続き）

- デスティネーション・メモリ領域がUCまたはWPとしてマッピングされている場合、これらのメモリタイプに関連するセマンティクスの実行は保証されていない（つまり、予約されている）。これらのセマンティクスが実行されるかどうかは、プロセッサによって異なる。

MASKMOVDQU 命令を使用して、バイトごとに選択した上でデータをマージする必要があるアルゴリズムのパフォーマンスを向上させることができる。ただし、MASKMOVDQU 命令で所有権の読み込みを行ってはならない。これを行うと、ストアの前に元のデータを割り当てることなく、バイトマスクを使用してデータが直接書き込まれるため、不要な帯域幅が発生する。

操作

```
IF (MASK[7] = 1)
    THEN DEST[DI/EDI] ← SRC[7-0] ELSE * memory location unchanged *; FI;
IF (MASK[15] = 1)
    THEN DEST[DI/EDI+1] ← SRC[15-8] ELSE * memory location unchanged *; FI;
    * Repeat operation for 3rd through 14th bytes in source operand *;
IF (MASK[127] = 1)
    THEN DEST[DI/EDI+15] ← SRC[127-120] ELSE * memory location unchanged *; FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

void_mm_maskmoveu_si128(__m128i d, __m128i n, char * p)

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合（マスクがすべて 0 の場合にも発生する）。
	デスティネーション・オペランドが書き込み不可能なセグメントの場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	SS セグメント内のアドレスが無効の場合（マスクがすべて 0 の場合にも発生する）。
#PF（フォルトコード）	ページフォルトが発生した場合（プロセッサ固有）。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MASKMOVDQU—Store Selected Bytes of Double Quadword（続き）

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合（マスクがすべて 0 の場合にも発生する）。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合（プロセッサ固有）。

MASKMOVQ—Store Selected Bytes of Quadword

オペコード	命令	説明
0F F7 /r	MASKMOVQ *mm1*, *mm2*	*mm2* のバイトマスクを使用して、*mm1* のバイトを選択した上でメモリ・ロケーションに書き込む。

説明

ソース・オペランド（第 1 オペランド）の選択されたバイトを、64 ビットのメモリ・ロケーションにストアする。マスク・オペランド（第 2 オペランド）は、ソース・オペランド内のどのバイトがメモリに書き込まれるかを選択する。ソース・オペランドとマスク・オペランドは MMX® テクノロジ・レジスタである。メモリ・ロケーションの最初のバイトの位置は、DI/EDI レジスタと DS レジスタで指定される（ストアアドレスのサイズは、アドレスサイズ属性によって異なる）。

マスク・オペランドの各バイトの最上位ビットは、ソース・オペランド内の対応するバイトがメモリ内の対応するバイト位置に書き込まれるかどうかを指定する。0 の場合は書き込みは行われず、1 の場合は書き込まれる。

MASKMOVQ 命令は、キャッシュの汚染を最小限に抑えるために、プロセッサに対して非テンポラルなヒントを生成する。非テンポラルなヒントは、WC（Write Combining）メモリタイプのプロトコルを使用して実現される（詳しくは、『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「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと）。WC プロトコルでは、順序設定のゆるいメモリ整合性モデルを使用する。そのため、複数のプロセッサが異なるメモリタイプを使用してデスティネーション・メモリ・ロケーションの読み込み / 書き込みを実行する可能性がある場合は、SFENCE または MFENCE 命令で実現されるフェンス操作を MASKMOVEDQU 命令と一緒に使用する必要がある。

この命令を使用すると、x87 FPU ステートから MMX テクノロジ・ステートへの移行が発生する（つまり、x87 FPU のトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。

マスクがすべて 0 の場合、MASKMOVQ 命令の動作は次のようになる。

- データはメモリに書き込まれない。
- x87 FPU ステートから MMX テクノロジ・ステートへの移行が発生する。
- この場合も、メモリのアドレス指定に関連する例外とページフォルトは報告されることがある（プロセッサによって異なる）。
- ブレークポイント（コードまたはデータ）の報告については保証していない（プロセッサによって異なる）。

MASKMOVQ—Store Selected Bytes of Quadword（続き）

- デスティネーション・メモリ領域が UC または WP としてマッピングされる場合、これらのメモリタイプ用の関連するセマンティクスの実装については保証していない（予約済みである）。セマンティクスの実装については、プロセッサによって異なる。

MASKMOVQ 命令を使用して、バイトごとに選択した上で最小単位としてデータをマージする必要があるアルゴリズムのパフォーマンスを強化することができる。MASKMOVQ 命令で所有権の読み込みを行ってはならない。これを行うと、ストアの前に元のデータを割り当てずに、バイトマスクを使用してデータが直接書き込まれるため、不要な帯域幅が発生する。

操作

```
IF (MASK[7] = 1)
    THEN DEST[DI/EDI] ← SRC[7-0] ELSE * memory location unchanged *; FI;
IF (MASK[15] = 1)
    THEN DEST[DI/EDI+1] ← SRC[15-8] ELSE * memory location unchanged *; FI;
    * Repeat operation for 3rd through 6th bytes in source operand *;
IF (MASK[63] = 1)
    THEN DEST[DI/EDI+15] ← SRC[63-56] ELSE * memory location unchanged *; FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

void_mm_maskmove_si64(__m64d, __m64n, char * p)

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合（マスクがすべて 0 の場合）。
	デスティネーション・オペランドが書き込み不可能なセグメントの場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	SS セグメント内のアドレスが無効の場合（マスクがすべて 0 の場合）。
#PF（フォルトコード）	ページフォルトが発生した場合（プロセッサ固有）。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の FPU 例外がある場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
	ModR/M バイトの Mod フィールドが 11B でない場合。

MASKMOVQ—Store Selected Bytes of Quadword（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合（マスクがすべて 0 の場合）。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の FPU 例外がある場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合（プロセッサ固有）。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MAXPD—Return Maximum Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 5F /r	MAXPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* の間で倍精度浮動小数点値の最大値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）のパック ド倍精度浮動小数点値の SIMD 比較を実行し、それぞれの値のペアの最大値をデスティネーション・オペランドに返す。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに転送される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MAXPD のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←      IF ((DEST[63-0] = 0.0) AND (SRC[63-0] = 0.0)) THEN SRC[63-0]
                  ELSE IF (DEST[63-0] = SNaN) THEN SRC[63-0];
                  ELSE IF SRC[63-0] = SNaN) THEN SRC[63-0];
                  ELSE IF (DEST[63-0] > SRC[63-0])
                      THEN DEST[63-0]
                      ELSE SRC[63-0];
                  FI;
DEST[127-64] ←    IF ((DEST[127-64] = 0.0) AND (SRC[127-64] = 0.0))
                      THEN SRC[127-64]
                  ELSE IF (DEST[127-64] = SNaN) THEN SRC[127-64];
                  ELSE IF SRC[127-64] = SNaN) THEN SRC[127-64];
                  ELSE IF (DEST[127-64] > SRC[63-0])
                      THEN DEST[127-64]
                      ELSE SRC[127-64];
                  FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_max_pd(__m128d a, __m128d b)

MAXPD—Return Maximum Packed Double-Precision Floating-Point Values（続き）

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドの場合を含む）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MAXPD—Return Maximum Packed Double-Precision Floating-Point Values（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）　ページフォルトが発生した場合。

MAXPS—Return Maximum Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 5F /r	MAXPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* の単精度浮動小数点値を比較して、各要素の最大値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）のパック ド単精度浮動小数点値の SIMD 比較を実行し、それぞれの値のペアの最大値をデスティネーション・オペランドに返す。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MAXPS のアクションをエミュレートする必要がある。

操作

```
DEST[31-0] ←        IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]
                    ELSE IF (DEST[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF SRC[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF (DEST[31-0] > SRC[31-0])
                        THEN DEST[31-0]
                        ELSE SRC[31-0];
                    FI;
* repeat operation for 2nd and 3rd doublewords *;
DEST[127-64] ←      IF ((DEST[127-96] = 0.0) AND (SRC[127-96] = 0.0))
                        THEN SRC[127-96]
                    ELSE IF (DEST[127-96] = SNaN) THEN SRC[127-96];
                    ELSE IF SRC[127-96] = SNaN) THEN SRC[127-96];
                    ELSE IF (DEST[127-96] > SRC[127-96])
                        THEN DEST[127-96]
                        ELSE SRC[127-96];
                    FI;
```

MAXPS—Return Maximum Packed Single-Precision Floating-Point Values（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_max_ps(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドを含む）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

MAXPS—Return Maximum Packed Single-Precision Floating-Point Values（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）　ページフォルトが発生した場合。

MAXSD—Return Maximum Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 5F /r	MAXSD *xmm1*, *xmm2/m64*	*xmm2/mem64* と *xmm1* の間でスカラ倍精度浮動小数点値の最大値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）の下位の倍精度浮動小数点値を比較し、最大値をデスティネーション・オペランドの下位クワッドワードに返す。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドがメモリ・オペランドの場合は、64 ビットだけがアクセスされる。デスティネーション・オペランドの上位クワッドワードは、変更されない。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MAXSD のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←    IF ((DEST[63-0] = 0.0) AND (SRC[63-0] = 0.0)) THEN SRC[63-0]
                IF (DEST[63-0] = SNaN) THEN SRC[63-0];
                ELSE IF SRC[63-0] = SNaN) THEN SRC[63-0];
                ELSE IF (DEST[63-0] > SRC[63-0])
                    THEN DEST[63-0]
                    ELSE SRC[63-0];
                FI;
* DEST[127-64] is unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_max_sd(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドの場合を含む）、デノーマル。

MAXSD—Return Maximum Scalar Double-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MAXSS—Return Maximum Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 5F /r	MAXSS *xmm1*, *xmm2/m32*	*xmm2/mem32* と *xmm1* のスカラ単精度浮動小数点値を比較して、各要素の最大値を返す。

説明

デスティネーション・オペランド（第1オペランド）とソース・オペランド（第2オペランド）の最下位の単精度浮動小数点値を比較し、最大値をデスティネーション・オペランドの最下位のダブルワードに返す。ソース・オペランドは、XMM レジスタまたは32ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドがメモリ・オペランドの場合は、32ビットだけがアクセスされる。デスティネーション・オペランドの上位3ダブルワードは、変更されない。

比較される値が両方とも0.0（どちらかの符号）の場合は、第2オペランド（ソース・オペランド）の値が返される。第2オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第2オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第1または第2オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MAXSS のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←        IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]
                    ELSE IF (DEST[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF SRC[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF (DEST[31-0] > SRC[31-0])
                        THEN DEST[31-0]
                        ELSE SRC[31-0];
                    FI;
* DEST[127-32] is unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_max_ss(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドを含む）、デノーマル。

MAXSS—Return Maximum Scalar Single-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MFENCE—Memory Fence

オペコード	命令	説明
0F AE /6	MFENCE	ロード操作とストア操作をシリアル化する。

説明

MFENCE 命令より前に発行されたすべてのメモリからのロード命令とメモリからのストア命令に対して、シリアル化操作を実行する。このシリアル化操作は、プログラムの順序で MFENCE 命令に先行するすべてのロード命令とストア命令が、MFENCE 命令に後続するロード命令とストア命令より前に、グローバルにアクセス可能になることを保証する。MFENCE 命令は、すべてのロード命令とストア命令、他の MFENCE 命令、すべての SFENCE 命令と LFENCE 命令、およびすべてのシリアル化命令（CPUID 命令など）に対して順序付けされる。

順序設定の緩いメモリタイプを使用して、アウト・オブ・オーダー発行、見込み的な読み込み、ライト・コンバイニング、ライト・コラプシングなどの手法により、プロセッサ・パフォーマンスの向上を達成する。データを参照する側のルーチンが、順序設定の緩いデータであることをどの程度認識しているかは、アプリケーションによって異なり、データを生成する側のルーチンにはわからない。MFENCE 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参照するルーチンの間のロードとストアの順序付けを保証するための効率的な方法である。

プロセッサは、見込み的な読み込みが許されるメモリタイプ（すなわち、WB、WC、および WT メモリタイプ）が割り当てられたシステムメモリ領域から、いつでもデータを見込み的にフェッチしてキャッシュに入れることができる。PREFETCH*h* 命令は、この見込み的な動作に対するヒントと見なされる。この見込み的なフェッチ動作は、命令の実行には拘束されず、任意の時点で発生する。したがって、MFENCE 命令は、PREFETCH*h* 命令などの見込み的なフェッチ機構に対して順序付けされない（つまり、MFENCE 命令の実行の直前、実行中、または実行後に、データがキャッシュに見込み的にロードされる可能性がある）。

操作

```
Wait_On_Following_Loads_And_Stores_Until(preceding_loads_and_stores
    _globally_visible);
```

同等のインテル® C/C++ コンパイラ組み込み関数

void_mm_mfence(void)

例外（すべての動作モード）

なし。

MINPD—Return Minimum Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 5D /r	MINPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* の間で倍精度浮動小数点値の最小値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）のパック ド倍精度浮動小数点値の SIMD 比較を実行し、それぞれの値のペアの最小値をデスティネーション・オペランドに返す。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MINPD のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←      IF ((DEST[63-0] = 0.0) AND (SRC[63-0] = 0.0)) THEN SRC[63-0]
                  ELSE IF (DEST[63-0] = SNaN) THEN SRC[63-0];
                  ELSE IF SRC[63-0] = SNaN) THEN SRC[63-0];
                  ELSE IF (DEST[63-0] < SRC[63-0])
                      THEN DEST[63-0]
                      ELSE SRC[63-0];
                  FI;
DEST[127-64] ←    IF ((DEST[127-64] = 0.0) AND (SRC[127-64] = 0.0))
                      THEN SRC[127-64]
                  ELSE IF (DEST[127-64] = SNaN) THEN SRC[127-64];
                  ELSE IF SRC[127-64] = SNaN) THEN SRC[127-64];
                  ELSE IF (DEST[127-64] < SRC[63-0])
                      THEN DEST[127-64]
                      ELSE SRC[127-64];
                  FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_min_pd(__m128d a, __m128d b)

MINPD—Return Minimum Packed Double-Precision Floating-Point Values（続き）

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドの場合を含む）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MINPD—Return Minimum Packed Double-Precision Floating-Point Values（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）　ページフォルトが発生した場合。

MINPS—Return Minimum Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 5D /r	MINPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* の単精度浮動小数点値を比較して、各要素の最小値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）のパック ド単精度浮動小数点値の SIMD 比較を実行し、それぞれの値のペアの最小値をデスティネーション・オペランドに返す。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MINPS のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←        IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]
                    ELSE IF (DEST[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF SRC[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF (DEST[31-0] > SRC[31-0])
                        THEN DEST[31-0]
                        ELSE SRC[31-0];
                    FI;
* repeat operation for 2nd and 3rd doublewords *;
DEST[127-64] ←      IF ((DEST127-96] = 0.0) AND (SRC[127-96] = 0.0))
                        THEN SRC[127-96]
                    ELSE IF (DEST[127-96] = SNaN) THEN SRC[127-96];
                    ELSE IF SRC[127-96] = SNaN) THEN SRC[127-96];
                    ELSE IF (DEST[127-96] < SRC[127-96])
                        THEN DEST[127-96]
                        ELSE SRC[127-96];
                    FI;
```

MINPS—Return Minimum Packed Single-Precision Floating-Point Values（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_min_ps(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドを含む）、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

MINPS—Return Minimum Packed Single-Precision Floating-Point Values（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）　ページフォルトが発生した場合。

MINSD—Return Minimum Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 5D /r	MINSD *xmm1*, *xmm2/m64*	*xmm2/mem64* と *xmm1* の間でスカラ倍精度浮動小数点値の最小値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）の下位の倍精度浮動小数点値を比較し、最小値をデスティネーション・オペランドの下位クワッドワードに返す。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドがメモリ・オペランドの場合は、64 ビットだけがアクセスされる。デスティネーション・オペランドの上位クワッドワードは、変更されない。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MINSD のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←        IF ((DEST[63-0] = 0.0) AND (SRC[63-0] = 0.0)) THEN SRC[63-0]
                    ELSE IF (DEST[63-0] = SNaN) THEN SRC[63-0];
                    ELSE IF SRC[63-0] = SNaN) THEN SRC[63-0];
                    ELSE IF (DEST[63-0] < SRC[63-0])
                        THEN DEST[63-0]
                        ELSE SRC[63-0];
                    FI;
* DEST[127-64] is unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_min_sd(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドの場合を含む）、デノーマル。

MINSD—Return Minimum Scalar Double-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MINSS—Return Minimum Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 5D /r	MINSS *xmm1*, *xmm2/m32*	*xmm2/mem32* と *xmm1* のスカラ単精度浮動小数点値を比較して、最小値を返す。

説明

デスティネーション・オペランド（第 1 オペランド）とソース・オペランド（第 2 オペランド）の最下位の単精度浮動小数点値を比較し、最小値をデスティネーション・オペランドの最下位のダブルワードに返す。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。ソース・オペランドがメモリ・オペランドの場合は、32 ビットだけがアクセスされる。デスティネーション・オペランドの上位 3 ダブルワードは、変更されない。

比較される値が両方とも 0.0（どちらかの符号）の場合は、第 2 オペランド（ソース・オペランド）の値が返される。第 2 オペランドの値が SNaN の場合は、その SNaN がそのままデスティネーションに返される（つまり、SNaN の QNaN 版は返されない）。

この命令では、オペランド値のうち一方だけが NaN（SNaN または QNaN）である場合は、第 2 オペランド（ソース・オペランド）の値、すなわち NaN または有効な浮動小数点値が結果に書き込まれる。この動作の代わりに（第 1 または第 2 オペランドの）NaN のソース・オペランドを返す必要がある場合は、一連の命令（例えば、比較の後に AND、ANDN、OR など）を使用して、MINSS のアクションをエミュレートする必要がある。

操作

```
DEST[63-0] ←        IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]
                    ELSE IF (DEST[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF SRC[31-0] = SNaN) THEN SRC[31-0];
                    ELSE IF (DEST[31-0] < SRC[31-0])
                        THEN DEST[31-0]
                        ELSE SRC[31-0];
                    FI;
* DEST[127-32] is unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m128d _mm_min_ss(__m128d a, __m128d b)

SIMD 浮動小数点例外

無効（QNaN ソース・オペランドを含む）、デノーマル。

MINSS—Return Minimum Scalar Single-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MONITOR—Setup Monitor Address

オペコード	命令	説明
0F, 01, C8	MONITOR	ハードウェアによってモニタされるリニアアドレス範囲を設定し、モニタをアクティブにする。このアドレス範囲は、ライトバック・メモリ・キャッシュ・タイプでなければならない。

説明

MONITOR 命令は、EAX レジスタ内で指定されたアドレスを使用して、アドレス・モニタリング・ハードウェアを使用可能な状態にする。モニタリング・ハードウェアがストア操作をチェックするアドレス範囲は、CPUID 命令によって確認できる。モニタリング・ハードウェアは、このアドレス範囲内のアドレスへのストアを検出し、書き込みが検出された時点でモニタ・ハードウェアをトリガする。モニタ・ハードウェアの状態は、MWAIT 命令によって使用される。

EAX レジスタの内容は、実効アドレスである。デフォルトでは、DS セグメントを使用して作成されたリニアアドレスがモニタされる。MONITOR 命令と合わせて、セグメント・オーバーライドを使用できる。

ECX と EDX を使用して、その他の情報を MONITOR 命令に伝達する。ECX は、MONITOR 命令のオプションの拡張機能を指定する。EDX は、MONITOR 命令に対するオプションのヒントを指定する。このヒントは、MONITOR 命令のアーキテクチャ上の動作を変更しない。ファミリ = 15 およびモデル = 3 の CPUID シグネチャを持つインテル® Pentium® 4 プロセッサには、拡張機能やヒントは定義されていない。EDX 内で未定義のヒントを指定した場合、プロセッサはそのヒントを無視する。ECX 内で未定義の拡張機能を指定した場合、MONITOR 命令の実行時に一般保護フォルト例外が発生する。

設定するアドレス範囲は、ライトバック・タイプのメモリの範囲内でなければならない。モニタされるアドレス範囲に対するライトバック・メモリ・タイプのストアが検出された場合にのみ、モニタリング・ハードウェアがトリガされる。アドレス範囲がライトバック・タイプのメモリの範囲を外れている場合は、アドレス・モニタ・ハードウェアは正しく設定されない。MONITOR 命令は、他のメモリ・トランザクションに対して、ロード操作として順序づけされる。ウェークアップが間違って検出されないようにアドレス範囲を決定する方法については、『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、下巻』の第 7 章を参照のこと。

MONITOR—Setup Monitor Address（続き）

MONITOR 命令はすべての特権レベルで使用できる。MONITOR 命令には、バイトロードに関連するすべてのパーミッション・チェックおよびフォルトが適用される。ロードと同様に、MONITOR 命令は、ページテーブル内で D ビットではなく、A ビットをセットする。CPUID 命令の MONITOR 機能フラグ（EAX=1 で CPUID を実行した場合、ECX のビット 3）は、プロセッサ上での MONITOR 命令と MWAIT 命令の利用可能性を示す。このフラグがセットされている場合、特権レベル 0 では MONITOR を無条件で実行でき、特権レベル 1 ～ 3 では条件付きで実行できる（ソフトウェアは、これらの命令を無条件で使用する前に、これらの命令が適切にサポートされているかをテストする必要がある）。オペレーティング・システムまたはシステム BIOS は、IA32_MISC_ENABLES MSR を使用してこの命令をディスエーブルにできる。この命令をディスエーブルにすると、該当する CPUID 機能フラグがクリアされる。この状態で MONITOR 命令を実行すると、無効オペコード例外が発生する。

操作

MONITOR は、EAX レジスタの内容を実効アドレスとして、モニタ・ハードウェアのアドレス範囲を設定し、モニタ・ハードウェアを使用可能な状態にする。このメモリアドレス範囲は、ライトバック・キャッシュ・タイプのメモリの範囲内でなければならない。指定されたアドレス範囲へのストアによって、モニタ・ハードウェアがトリガされる。ECX と EDX の内容を使用して、その他の情報をモニタ・ハードウェアに伝達できる。

同等のインテル® C/C++ コンパイラ組み込み関数

MONITOR　　void _mm_monitor(void const *p, unsigned extensions,unsigned hints)

例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#GP(0)	ECX の値が 0 以外の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルト（TBD）の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合。
	この命令が利用可能でないときに特権レベル 1 ～ 3 で実行した場合。
	LOCK、REP、REPNE/NZ、およびオペランド・サイズのオーバーライド・プリフィックスが使用されている場合。

MONITOR—Setup Monitor Address（続き）

実アドレスモード例外

#GP	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#GP(0)	ECX の値が 0 以外の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合。
	LOCK、REP、REPNE/NZ、およびオペランド・サイズのオーバーライド・プリフィックスが使用されている場合。

仮想 8086 モード例外

#GP	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#GP(0)	ECX の値が 0 以外の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合。
	この命令が利用可能でないときに特権レベル 1 ～ 3 で実行した場合。
	LOCK、REP、REPNE/NZ、およびオペランド・サイズのオーバーライド・プリフィックスが使用されている場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

MOV—Move

オペコード	命令	説明
88 /*r*	MOV *r/m8*, *r8*	*r8* を *r/m8* に転送する。
89 /*r*	MOV *r/m16*, *r16*	*r16* を *r/m16* に転送する。
89 /*r*	MOV *r/m32*, *r32*	*r32* を *r/m32* に転送する。
8A /*r*	MOV *r8*, *r/m8*	*r/m8* を *r8* に転送する。
8B /*r*	MOV *r16*, *r/m16*	*r/m16* を *r16* に転送する。
8B /*r*	MOV *r32*, *r/m32*	*r/m32* を *r32* に転送する。
8C /*r*	MOV *r/m16*, *Sreg***	セグメント・レジスタを *r/m16* に転送する。
8E /*r*	MOV *Sreg*, *r/m16***	*r/m16* をセグメント・レジスタに転送する。
A0	MOV AL, *moffs8**	（セグメント : オフセット）のバイトを AL に転送する。
A1	MOV AX, *moffs16**	（セグメント : オフセット）のワードを AX に転送する。
A1	MOV EAX, *moffs32**	（セグメント:オフセット）のダブルワードをEAXに転送する。
A2	MOV *moffs8**, AL	AL を（セグメント : オフセット）に転送する。
A3	MOV *moffs16**, AX	AX を（セグメント : オフセット）に転送する。
A3	MOV *moffs32**, EAX	EAX を (セグメント : オフセット）に転送する。
B0+ *rb*	MOV *r8*, *imm8*	*imm8* を *r8* に転送する。
B8+ *rw*	MOV *r16*, *imm16*	*imm16* を *r16* に転送する。
B8+ *rd*	MOV *r32*, *imm32*	*imm32* を *r32* に転送する。
C6 /*0*	MOV *r/m8*, *imm8*	*imm8* を *r/m8* に転送する。
C7 /*0*	MOV *r/m16*, *imm16*	*imm16* を *r/m16* に転送する。
C7 /*0*	MOV *r/m32*, *imm32*	*imm32* を *r/m32* に転送する。

注 :

* moffs8、moffs16、moffs32 オペランドは、8、16、32 がデータのサイズを参照するセグメント・ベースに相対的な単純オフセットを指定する。命令のアドレスサイズ属性によって、オフセットのサイズ（16 ビットまたは 32 ビット）が決まる。

** 32 ビット・モードでは、アセンブラは、この命令に 16 ビット・オペランド・サイズ・プリフィックスを挿入することがある（詳細については、以下の「説明」の項を参照のこと）。

説明

第 2 オペランド（ソース・オペランド）を第 1 オペランド（デスティネーション・オペランド）にコピーする。ソース・オペランドには、即値、汎用レジスタ、セグメント・レジスタ、またはメモリ・ロケーションを使用できる。デスティネーション・オペランドには、汎用レジスタ、セグメント・レジスタ、またはメモリ・ロケーションを使用できる。両方のオペランドとも、同じサイズ（バイト、ワード、またはダブルワード）でなければならない。

MOV命令を使用してCS レジスタをロードすることはできない。そうしようとすると、無効オペコード例外（#UD）が発生する。CS レジスタをロードするには、far JMP、CALL、またはRET 命令を使用する。

MOV—Move（続き）

デスティネーション・オペランドがセグメント・レジスタ（DS、ES、FS、GS、またはSS）である場合は、ソース・オペランドは有効なセグメント・セレクタでなければならない。保護モードでは、セグメント・セレクタをセグメント・レジスタに転送すると、そのセグメント・セレクタに関連するセグメント・ディスクリプタ情報が自動的にセグメント・レジスタの隠蔽（シャドー）部分にロードされる。この情報をロードしている間は、セグメント・セレクタおよびセグメント・ディスクリプタ情報が有効になる（下記の「操作」のアルゴリズムを参照）。セグメント・ディスクリプタ・データは、指定したセグメント・セレクタのGDTエントリまたはLDTエントリから得られる。

ヌルのセグメント・セレクタ（値0000 ～ 0003）を保護例外を発生させずにDS、ES、FS、およびGSレジスタにロードすることができる。ただし、その対応するセグメント・レジスタがヌル値をロードされているセグメントをその後に参照しようとすると、一般保護例外（#GP）が発生し、メモリ参照は行われない。

SSレジスタをMOV命令でロードすると、次の命令の実行後まですべての割り込みが禁止される。この操作によって、割り込みが発生する前に、次の命令（MOV ESP、**スタックポインタ値**）でESPレジスタにスタックポインタをロードすることができる[1]。LSS命令によって、SSレジスタとESPレジスタをより効率的にロードできることに注意する。

1. コード命令ブレークポイント（デバッグ用）がMOV SS命令の直後の命令に置かれている場合、そのブレークポイントはトリガされないことがある。

   以下の命令を過ぎて割り込みを個別にディレイさせる命令シーケンスでは、シーケンスの最初の命令は、割り込みをディレイさせることが保証されるが、後続の割り込みディレイ命令は、割り込みをディレイさせない場合があることに注意する。そのため、以下の命令シーケンスでは、

   ```
   STI
   MOV SS, EAX
   MOV ESP, EBP
   ```

   STIも1命令の間割り込みをディレイさせるので、MOV ESP,EBPが実行される前に、割り込みが認識されることもある。

MOV—Move（続き）

32 ビット・モードで動作しているときにセグメント・レジスタと汎用レジスタとの間でデータを転送すると、32 ビットの IA-32 プロセッサでは、この命令で 16 ビット・オペランド・サイズ・プリフィックス（値 66H をもつバイト）を使用する必要はないが、ほとんどのアセンブラでは、命令の標準形式（例えば、MOV DS,AX）が使用されているとそのプリフィックスを挿入する。プロセッサは、この命令を正しく実行するが、通常は余分なクロックを必要とする。ほとんどのアセンブラでは、命令形式 MOV DS,EAX を使用すると、この不必要な 66H プリフィックスを避けられる。プロセッサは、32 ビット汎用レジスタを使用する命令を実行するときは、汎用レジスタの下位 16 ビットがデスティネーション・オペランドまたはソース・オペランドであると想定する。レジスタがデスティネーション・オペランドである場合は、レジスタの上位 2 バイトにある結果の値は、プロセッサに依存する。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、上位 2 バイトにはゼロが入る。それより以前の 32 ビットの IA-32 プロセッサでは、上位 2 バイトは未定義である。

操作

```
DEST ← SRC;
```

保護モードの間にセグメント・レジスタをロードすると、以下のリストで説明しているように特殊なチェックと処置が行われる。これらのチェックは、セグメント・セレクタおよびそれが指しているセグメント・ディスクリプタに対して行われる。

```
IF SS is loaded;
    THEN
        IF segment selector is null
            THEN #GP(0);
        FI;
        IF segment selector index is outside descriptor table limits
            OR segment selector's RPL ≠ CPL
            OR segment is not a writable data segment
            OR DPL ≠ CPL
                THEN #GP(selector);
        FI;
        IF segment not marked present
            THEN #SS(selector);
    ELSE
        SS ← segment selector;
        SS ← segment descriptor;
    FI;
FI;
IF DS, ES, FS, or GS is loaded with non-null selector;
THEN
    IF segment selector index is outside descriptor table limits
        OR segment is not a data or readable code segment
        OR ((segment is a data or nonconforming code segment)
            AND (both RPL and CPL > DPL))
```

MOV—Move（続き）

```
                THEN #GP(selector);
            IF segment not marked present
                THEN #NP(selector);
        ELSE
            SegmentRegister ← segment selector;
            SegmentRegister ← segment descriptor;
        FI;
FI;
IF DS, ES, FS, or GS is loaded with a null selector;
    THEN
        SegmentRegister ← segment selector;
        SegmentRegister ← segment descriptor;
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	ヌル・セグメント・セレクタで SS レジスタをロードしようとした場合。
	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#GP（セレクタ）	セグメント・セレクタ・インデックスがディスクリプタ・テーブルの範囲外の場合。
	SS レジスタがロードされ、セグメント・セレクタの RPL およびセグメント・ディスクリプタの DPL が CPL に等しくない場合。
	SS レジスタがロードされ、指示先のセグメントが書き込み不可能なデータ・セグメントである場合。
	DS、ES、FS、または GS レジスタがロードされ、指示先のセグメントがデータ・セグメントまたは書き込み可能なコード・セグメントでない場合。
	DS、ES、FS、または GS レジスタがロードされ、指示先のセグメントがデータ・セグメントまたは非コンフォーミング・コード・セグメントであるが、RPL と CPL の両方とも DPL より大きい場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#SS（セレクタ）	SS レジスタがロードされ、指示先のセグメントが存在しないとマークされている場合。

MOV—Move（続き）

#NP	DS、ES、FS、または GS レジスタがロードされ、指示先のセグメントが存在しないとマークされている場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。
#UD	CS レジスタをロードしようとした場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#UD	CS レジスタをロードしようとした場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。
#UD	CS レジスタをロードしようとした場合。

MOV—Move to/from Control Registers

オペコード	命令	説明
0F 22 /*r*	MOV CR0, *r32*	0F 22 *r32* を CR0 に転送する。
0F 22 /*r*	MOV CR2, *r32*	0F 22 *r32* を CR2 に転送する。
0F 22 /*r*	MOV CR3, *r32*	0F 22 *r32* を CR3 に転送する。
0F 22 /*r*	MOV CR4, *r32*	0F 22 *r32* を CR4 に転送する。
0F 20 /*r*	MOV *r32*, CR0	0F 20 CR0 を *r32* に転送する。
0F 20 /*r*	MOV *r32*, CR2	0F 20 CR2 を *r32* に転送する。
0F 20 /*r*	MOV *r32*, CR3	0F 20 CR3 を *r32* に転送する。
0F 20 /*r*	MOV *r32*, CR4	0F 20 CR4 を *r32* に転送する。

説明

制御レジスタ（CR0、CR2、CR3、または CR4）の内容を汎用レジスタに、またはその逆に転送する。これらの命令のオペランド・サイズは、オペランド・サイズ属性に関係なく常に 32 ビットである。制御レジスタのフラグとフィールドの詳細な説明については、『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、下巻』の第 2 章の「制御レジスタ」を参照のこと。この命令は、現在の特権レベルが 0 である場合にだけ実行できる。

制御レジスタをロードするときは、プログラムは予約ビットの変更を試みてはならない。すなわち、常に前に読み取られた値に予約ビットを設定する。CR4 の予約ビットの変更を試みると、一般保護フォルトが発生する。CR0 および CR3 のロードの実行後、これらのレジスタ内の予約ビットはクリアされたままである。これらのビットをセットしようとしても、何も影響も与えない。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・プロセッサでは、CR0 のロードの実行後、CR0.ET はセットされたままになる。このビットをクリアしようとしても、何も影響も与えない。

オペコード・レベルでは、ModR/M バイト内の reg フィールドは、どの制御レジスタをロードするかまたは読み取るかを指定する。mod フィールドの 2 ビットは、常に 11B である。r/m フィールドは、ロードされるかまたは読み取られる汎用レジスタを指定する。

これらの命令は、以下の二次的な影響も与える。

- 制御レジスタ CR3 に書き込むと、すべての非グローバル TLB エントリがフラッシュされる。制御レジスタのフラグとフィールドの詳細な説明については、『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、下巻』の第 3 章の「トランスレーション・ルックアサイド・バッファ（TLB）」を参照のこと。

MOV—Move to/from Control Registers（続き）

以下の二次的な影響は、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・プロセッサではプロセッサ固有である。すべての IA-32 プロセッサで、ソフトウェアはこの機能に依存してはならない。

- 制御レジスタのページング・フラグ（CR0 レジスタの PE と PG、CR4 レジスタの PGE、PSE、PAE）のいずれかを修正すると、グローバル・エントリを含めたすべての TLB エントリがフラッシュされる。
- PG フラグを 1 にセットし、（物理アドレス拡張モードをイネーブルにするために）PAE フラグを 1 にセットするように制御レジスタ CR4 に書き込むと、ページ・ディレクトリ・ポインタ・テーブル（PDPT）のポインタがプロセッサに（内部非アーキテクチャ・レジスタに）ロードされる。
- PAE フラグを 1 にセットし、PG フラグを 1 にセットしている場合に、制御レジスタ CR3 に書き込むと、PDPTR がプロセッサに再ロードされる。PAE フラグを 1 にセットし、PG フラグをセットするように制御レジスタ CR0 に書き込むと、PDPTR がプロセッサに再ロードされる。

操作

```
DEST ← SRC;
```

影響を受けるフラグ

OF、SF、ZF、AF、PF、および CF フラグは未定義。

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
	（PE フラグが 0 にセットされているときに PG フラグを 1 にセットするか、または NW フラグが 1 にセットされているときに CD フラグを 0 にセットするなど）無効なビット組み合わせの書き込みを CR0 に行おうとした場合。
	CR4 の予約ビットに 1 を書き込もうとした場合。
	ページ・ディレクトリ・ポインタ・テーブル（PDPT）内の予約ビットがセットされていて、制御レジスタのロードによって PDPT がプロセッサ内にロードされた場合。

実アドレスモード例外

#GP	CR4 の予約ビットに 1 を書き込もうとした場合。
	（PE フラグが 0 にセットされているときに PG フラグを 1 にセットするなど）無効なビット組み合わせの書き込みを CR0 に行おうとした場合。

MOV—Move to/from Control Registers（続き）

仮想 8086 モード例外

#GP(0)	これらの命令は、仮想 8086 モードで実行することはできない。

MOV—Move to/from Debug Registers

オペコード	命令	説明
0F 21/*r*	MOV *r32*, DR0-DR7	デバッグレジスタを *r32* に転送する。
0F 23 /*r*	MOV DR0-DR7, *r32*	*r32* をデバッグレジスタに転送する。

説明

デバッグレジスタ（DR0、DR1、DR2、DR3、DR4、DR5、DR6、またはDR7）の内容を汎用レジスタに、またはその逆に転送する。これらの命令のオペランド・サイズは、オペランド・サイズ属性に関係なく常に32ビットである。デバッグレジスタのフラグとフィールドの詳細な説明については、『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、下巻』の第15章「デバッグと性能モニタリング」を参照のこと。

これらの命令は、特権レベル0または実アドレスモードで実行しなければならない。

CR4レジスタのデバッグ拡張（DE）フラグをクリアすると、これらの命令は、Intel386™ プロセッサおよびIntel486™ プロセッサと互換性があるようにデバック・レジスタを操作する。このモードでは、DR4およびDR5への参照は、それぞれDR6およびDR7を参照する。CR4のDEをセットすると、DR4およびDR5への参照の試みは、未定義オペコード（#UD）例外を発生させる。（CR4レジスタは、インテル® Pentium® プロセッサで初めてIA-32アーキテクチャに追加された。）

オペコード・レベルでは、ModR/Mバイトのregフィールドは、どのデバッグレジスタをロードするかまたは読み取るかを指定する。modフィールドの2ビットは、常に11である。r/mフィールドは、ロードされるかまたは読み取られる汎用レジスタを指定する。

操作

```
IF ((DE = 1)  and (SRC or DEST = DR4 or DR5))
THEN
    #UD;
ELSE
    DEST ← SRC;
```

影響を受けるフラグ

OF、SF、ZF、AF、PF、およびCFフラグは未定義。

MOV—Move to/from Debug Registers（続き）

保護モード例外

#GP(0)	現行特権レベルが 0 でない場合。
#UD	CR4 の DE（デバッグ拡張）ビットがセットされていて、MOV 命令が DR4 または DR5 を伴って実行された場合。
#DB	デバッグレジスタ DR7 の GD フラグがセットされている間に、いずれかのデバッグレジスタがアクセスされた場合。

実アドレスモード例外

#UD	CR4 の DE（デバッグ拡張）ビットがセットされていて、MOV 命令が DR4 または DR5 を伴って実行された場合。
#DB	デバッグレジスタ DR7 の GD フラグがセットされている間に、いずれかのデバッグレジスタがアクセスされた場合。

仮想 8086 モード例外

#GP(0)	仮想 8086 モードになっているときに、デバッグレジスタをロードするかまたは読み取ることはできない。

MOVAPD—Move Aligned Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 28 /r	MOVAPD *xmm1*, *xmm2/m128*	パック ド倍精度浮動小数点値を、*xmm2/m128* から *xmm1* に移動する。
66 0F 29 /r	MOVAPD *xmm2/m128*, *xmm1*	パック ド倍精度浮動小数点値を、*xmm1* から *xmm2/m128* に移動する。

説明

2 つのパック ド倍精度浮動小数点値が入っているダブル・クワッドワードを、ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）に移動する。この命令を使用して、XMM レジスタを 128 ビットのメモリ・ロケーションからロードする操作、XMM レジスタの内容を 128 ビットのメモリ・ロケーションにストアする操作、または 2 つの XMM レジスタの間でデータを転送する操作が可能である。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは 16 バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

アライメントの合っていないメモリ・ロケーションとの間で倍精度浮動小数点値を移動する場合は、MOVUPD 命令を使用する。

操作

```
DEST ← SRC;
* #GP if SRC or DEST unaligned memory operand *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
__m128 _mm_load_pd(double * p)
void_mm_store_pd(double *p, __m128 a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

MOVAPD—Move Aligned Packed Double-Precision Floating-Point Values（続き）

#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVAPS—Move Aligned Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 28 /r	MOVAPS *xmm1*, *xmm2/m128*	パック ド単精度浮動小数点値を、*xmm2/m128* から *xmm1* レジスタに移動する。
0F 29 /r	MOVAPS *xmm2/m128*, *xmm1*	パック ド単精度浮動小数点値を、*xmm1* レジスタから *xmm2/m128* に移動する。

説明

4つのパック ド単精度浮動小数点値が入っているダブル・クワッドワードを、ソース・オペランド（第2オペランド）からデスティネーション・オペランド（第1オペランド）に移動する。この命令を使用して、XMM レジスタを128ビットのメモリ・ロケーションからロードする操作、XMM レジスタの内容を128ビットのメモリ・ロケーションにストアする操作、または2つの XMM レジスタの間でデータを転送する操作が可能である。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは16バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

アライメントの合っていないメモリ・ロケーションとの間でパック ド単精度浮動小数点値を移動する場合は、MOVUPS 命令を使用する。

操作

```
DEST ← SRC;
* #GP if SRC or DEST unaligned memory operand *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
__m128 _mm_load_ps (float * p)
void_mm_store_ps (float *p, __m128 a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。 セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

MOVAPS—Move Aligned Packed Single-Precision Floating-Point Values（続き）

#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVD—Move Doubleword

オペコード	命令	説明
0F 6E /r	MOVD *mm*, *r/m32*	ダブルワードを *r/m32* から *mm* に転送する。
0F 7E /r	MOVD *r/m32*, *mm*	ダブルワードを *mm* から *r/m32* に転送する。
66 0F 6E /r	MOVD *xmm*, *r/m32*	ダブルワードを *r/m32* から *xmm* に転送する。
66 0F 7E /r	MOVD *r/m32*, *xmm*	ダブルワードを *xmm* レジスタから *r/m32* に転送する。

説明

ダブルワードをソース・オペランド（第2オペランド）からデスティネーション・オペランド（第1オペランド）にコピーする。ソース・オペランドとデスティネーション・オペランドには、汎用レジスタ、MMX® テクノロジ・レジスタ、XMM レジスタ、または32ビット・メモリ・ロケーションを使用できる。この命令を使用して、MMX テクノロジ・レジスタの下位ダブルワードと汎用レジスタまたは32ビットのメモリ・ロケーションの間、またはXMM レジスタの最下位のダブルワードと汎用レジスタまたは32 ビットのメモリ・ロケーションの間で、ダブルワードを転送することができる。ただし、この命令は、2つのMMXテクノロジ・レジスタの間、2つのXMMレジスタの間、2 つの汎用レジスタの間、または2 つのメモリ・ロケーションの間のデータ転送には使用できない。

デスティネーション・オペランドがMMXテクノロジ・レジスタの場合は、ソース・オペランドはそのレジスタの下位ダブルワードに書き込まれ、レジスタは64ビットにゼロ拡張される。デスティネーション・オペランドがXMMレジスタの場合は、ソース・オペランドはそのレジスタの最下位のダブルワードに書き込まれ、レジスタは128ビットにゼロ拡張される。

操作

```
MOVD instruction when destination operand is MMX technology register:
    DEST[31-0] ← SRC;
    DEST[63-32] ← 00000000H;
MOVD instruction when destination operand is XMM register:
    DEST[31-0] ← SRC;
    DEST[127-32] ← 00000000000000000000000000H;
MOVD instruction when source operand is MMX technology or XMM register:
    DEST ← SRC[31-0];
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVD        __m64 _mm_cvtsi32_si64 (int i )
MOVD        int _mm_cvtsi64_si32 ( __m64m )
MOVD        __m128i _mm_cvtsi32_si128 (int a)
MOVD        int _mm_cvtsi128_si32 ( __m128i a)
```

MOVD—Move Doubleword（続き）

影響を受けるフラグ

なし。

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。 メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合にのみ、128 ビット操作によって #UD が発生する。SSE2 に対応していないプロセッサ（MMX テクノロジ対応プロセッサ）上で 128 ビット命令を実行した場合、その命令は mm レジスタを操作し、#UD は発生しない。
#NM	CR0 の TS がセットされた場合。
#MF	(MMXテクノロジ・レジスタ操作のみ)未処理のFPU例外がある場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	オペランドのいずれかの部分が実効アドレス空間 0 ～ FFFFH の外にある場合。
#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合にのみ、128 ビット操作によって #UD が発生する。SSE2 に対応していないプロセッサ（MMX テクノロジ対応プロセッサ）上で 128 ビット命令を実行した場合、その命令は mm レジスタを操作し、#UD は発生しない。
#NM	CR0 の TS がセットされた場合。
#MF	(MMXテクノロジ・レジスタ操作のみ)未処理のFPU例外がある場合。

MOVD—Move Doubleword（続き）

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

MOVDDUP—Move One Double-FP and Duplicate

オペコード	命令	説明
F2, 0F, 12, /r	MOVDDUP *xmm1*, *xmm2/m64*	下位の倍精度データ要素を表す 64 ビットを *xmm2/m64* から *xmm1* レジスタに移動し、複製する。

説明

リニアアドレスは、参照されるメモリデータの最下位バイトのアドレスに対応する。メモリアドレスが指定されると、メモリ上の位置 m64 から 8 バイトのデータがロードされる。レジスタ - レジスタ形式の操作を使用した場合は、128 ビット・ソース・レジスタの下位半分が複製され、128 ビット・デスティネーション・レジスタにコピーされる。図 3-14 を参照のこと。

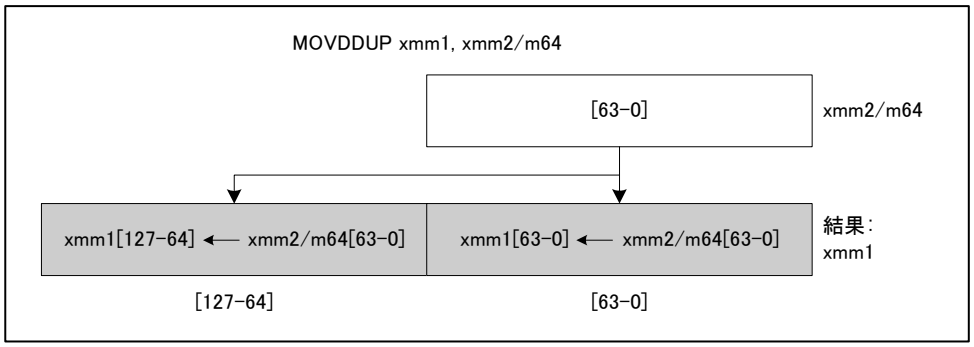

図 3-14. MOVDDUP: Move One Double-FP and Duplicate

操作

```
if (source == m64) {
        // load instruction
        xmm1[63-0] = m64;
        xmm1[127-64] = m64;
}
else {
        // move instruction
        xmm1[63-0]   = xmm2[63-0];
        xmm1[127-64] = xmm2[63-0];
}
```

MOVDDUP—Move One Double-FP and Duplicate（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

MOVDDUP __m128d _mm_movedup_pd(__m128d a)
__m128d _mm_loaddup_pd(double const * dp)

例外

なし。

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NP	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

MOVDDUP—Move One Double-FP and Duplicate（続き）

仮想 8086 モード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVDQA—Move Aligned Double Quadword

オペコード	命令	説明
66 0F 6F /r	MOVDQA *xmm1*, *xmm2/m128*	アライメントの合ったダブル・クワッドワードを、*xmm2/m128* から *xmm1* に移動する。
66 0F 7F /r	MOVDQA *xmm2/m128*, *xmm1*	アライメントの合ったダブル・クワッドワードを、*xmm1* から *xmm2/m128* に移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）にダブル・クワッドワードを移動する。この命令を使用して、128 ビット・メモリ・ロケーションから XMM レジスタへのロード、128 ビット・メモリ・ロケーションへの XMM レジスタの内容のストア、あるいは、2 つの XMM レジスタ間でのデータの移動を行うことができる。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合は、そのオペランドのアライメントは 16 バイトに合っていなければならない。アライメントが合っていない場合は、一般保護例外（#GP）が発生する。

アライメントの合っていないメモリ・ロケーションとの間でダブル・クワッドワードを移動する場合は、MOVDQU 命令を使用する。

操作

```
DEST ← SRC;
* #GP if SRC or DEST unaligned memory operand *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVDQA      __m128i _mm_load_si128 ( __m128i *p)
MOVDQA      void _mm_store_si128 ( __m128i *p, __m128i a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#PF（フォルトコード）	ページフォルトが発生した場合。
#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の TS がセットされた場合。

MOVDQA—Move Aligned Double Quadword（続き）

#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID.SSE2 = 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID.SSE2 = 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVDQU—Move Unaligned Double Quadword

オペコード	命令	説明
F3 0F 6F /r	MOVDQU *xmm1*, *xmm2/m128*	アライメントの合わないダブル・クワッドワードを、*xmm2/m128* から *xmm1* に移動する。
F3 0F 7F /r	MOVDQU *xmm2/m128*, *xmm1*	アライメントの合わないダブル・クワッドワードを、*xmm1* から *xmm2/m128* に移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）にダブル・クワッドワードを移動する。この命令を使用して、128 ビット・メモリ・ロケーションから XMM レジスタへのロード、128 ビット・メモリ・ロケーションへの XMM レジスタの内容のストア、あるいは、2 つの XMM レジスタ間でのデータの移動を行うことができる。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合、そのオペランドのアライメントが 16 バイトに合っていなくても、一般保護例外（#GP）は発生しない。

ダブル・クワッドワードの転送先または転送元のメモリ・ロケーションのアライメントが 16 バイトに合っていることがわかっている場合は、MOVDQA 命令を使用する。

16 ビット・アドレス指定モードでの実行中は、128 ビット・データ・アクセスのリニアアドレスが 16 ビット・セグメントの終点からはみ出すことは許されない。これは、予約済みの動作として定義されている。この場合に一般保護例外（#GP）が発生するかどうかは、プロセッサによって異なる。16 ビット・セグメントの範囲を超えたアドレスは、そのセグメントの始点にラップアラウンドされることもあるし、ラップアラウンドされないこともある。

操作

```
DEST ← SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVDQU      void _mm_storeu_si128 ( __m128i *p, __m128i a)
MOVDQU      __m128i _mm_loadu_si128 ( __m128i *p)
```

SIMD 浮動小数点例外

なし。

MOVDQU—Move Unaligned Double Quadword（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID.SSE = 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

実アドレスモード例外

#GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID.SSE = 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVDQ2Q—Move Quadword from XMM to MMX Technology Register

オペコード	命令	説明
F2 0F D6	MOVDQ2Q *mm*, *xmm*	*xmm* レジスタの下位クワッドワードを MMX® テクノロジ・レジスタに移動する。

説明

ソース・オペランド（第2オペランド）の下位クワッドワードをデスティネーション・オペランド（第1オペランド）に移動する。ソース・オペランドはXMMレジスタである。デスティネーション・オペランドはMMX®テクノロジ・レジスタである。

この命令を使用すると、x87 FPU操作からMMXテクノロジ操作への移行が発生する（つまり、x87 FPUのトップオブスタック・ポインタは0に設定され、x87 FPUタグワードはすべて0[有効]に設定される）。未処理のx87 FPU浮動小数点例外があるときにこの命令を実行しようとすると、MOVDQ2Q命令を実行する前にその例外が処理される。

操作

DEST ← SRC[63-0]

同等のインテル® C/C++ コンパイラ組み込み関数

MOVDQ2Q __m64 _mm_movepi64_pi64 (__m128i a)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#MF	未処理の x87 FPU 例外がある場合。

実アドレスモード例外

保護モードと同じ例外。

仮想 8086 モード例外

保護モードと同じ例外。

MOVHLPS—Move Packed Single-Precision Floating-Point Values High to Low

オペコード	命令	説明
OF 12 /r	MOVHLPS *xmm1*, *xmm2*	2つのパックド単精度浮動小数点値を、*xmm2*の上位クワッドワードから *xmm1* の下位クワッドワードに移動する。

説明

2 つのパックド単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）の上位クワッドワードからデスティネーション・オペランド（第 1 オペランド）の下位クワッドワードに移動する。デスティネーション・オペランドの上位クワッドワードは変更されない。

操作

```
DEST[63-0] ← SRC[127-64];
* DEST[127-64] unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVHLPS　　__m128 _mm_movehl_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

保護モードと同じ例外。

仮想 8086 モード例外

保護モードと同じ例外。

MOVHPD—Move High Packed Double-Precision Floating-Point Value

オペコード	命令	説明
66 0F 16 /r	MOVHPD *xmm*, *m64*	*m64* から *xmm* の上位クワッドワードに倍精度浮動小数点値を移動する。
66 0F 17 /r	MOVHPD *m64*, *xmm*	*xmm* の上位クワッドワードから *m64* に倍精度浮動小数点値を移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）に倍精度浮動小数点値を移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの上位クワッドワードとメモリの間で、倍精度浮動小数点値を転送することができる。ただし、この命令は、レジスタ同士の間またはメモリ同士の間のデータ転送には使用できない。デスティネーション・オペランドが XMM レジスタの場合は、レジスタの下位クワッドワードは変更されない。

操作

```
MOVHPD instruction for memory to XMM move:
    DEST[127-64] ← SRC ;
    * DEST[63-0] unchanged *;

MOVHPD instruction for XMM to memory move:
    DEST ← SRC[127-64] ;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVHPD      __m128d _mm_loadh_pd ( __m128d a, double *p)
MOVHPD      void _mm_storeh_pd (double *p, __m128d a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MOVHPD—Move High Packed Double-Precision Floating-Point Value（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVHPS—Move High Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 16 /r	MOVHPS *xmm*, *m64*	2 つのパックド単精度浮動小数点値を、*m64* から *xmm* の上位クワッドワードに移動する。
0F 17 /r	MOVHPS *m64*, *xmm*	2つのパックド単精度浮動小数点値を、*xmm*の上位クワッドワードから *m64* に移動する。

説明

2 つのパックド単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）に移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの上位クワッドワードとメモリの間で、2 つの単精度浮動小数点値を転送することができる。ただし、この命令は、レジスタ同士の間またはメモリ同士の間のデータ転送には使用できない。デスティネーション・オペランドが XMM レジスタの場合は、レジスタの下位クワッドワードは変更されない。

操作

```
MOVHPD instruction for memory to XMM move:
    DEST[127-64] ← SRC ;
    * DEST[63-0] unchanged *;

MOVHPD instruction for XMM to memory move:
    DEST ← SRC[127-64] ;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVHPS      __m128d _mm_loadh_pi ( __m128d a, __m64 *p)
MOVHPS      void _mm_storeh_pi (__m64 *p, __m128d a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

MOVHPS—Move High Packed Single-Precision Floating-Point Values（続き）

#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVLHPS—Move Packed Single-Precision Floating-Point Values Low to High

オペコード	命令	説明
OF 16 /r	MOVLHPS *xmm1*, *xmm2*	2 つのパック ド単精度浮動小数点値を、*xmm2* の下位クワッドワードから *xmm1* の上位クワッドワードに移動する。

説明

2 つのパック ド単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）の下位クワッドワードからデスティネーション・オペランド（第 1 オペランド）の上位クワッドワードに移動する。デスティネーション・オペランドの下位クワッドワードは変更されない。

操作

```
DEST[127-64] ← SRC[63-0];
* DEST[63-0] unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVHLPS __m128 _mm_movelh_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

保護モードと同じ例外。

仮想 8086 モード例外

保護モードと同じ例外。

MOVLPD—Move Low Packed Double-Precision Floating-Point Value

オペコード	命令	説明
66 0F 12 /r	MOVLPD *xmm*, *m64*	*m64* から *xmm* レジスタの下位クワッドワードに倍精度浮動小数点値を移動する。
66 0F 13 /r	MOVLPD *m64*, *xmm*	*xmm* レジスタの下位クワッドワードから *m64* に倍精度浮動小数点値を移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）に倍精度浮動小数点値を移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの下位クワッドワードとメモリの間で、倍精度浮動小数点値を転送することができる。ただし、この命令は、レジスタ同士の間またはメモリ同士の間のデータ転送には使用できない。デスティネーション・オペランドが XMM レジスタの場合は、レジスタの上位クワッドワードは変更されない。

操作

```
MOVLPD instruction for memory to XMM move:
    DEST[63-0] ← SRC ;
    * DEST[127-64] unchanged *;

MOVLPD instruction for XMM to memory move:
    DEST ← SRC[63-0] ;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVLPD      __m128d _mm_loadl_pd ( __m128d a, double *p)
MOVLPD      void _mm_storel_pd (double *p, __m128d a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MOVLPD—Move Low Packed Double-Precision Floating-Point Value（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVLPS—Move Low Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 12 /r	MOVLPS *xmm*, *m64*	2 つのパックド単精度浮動小数点値を、*m64* から *xmm* の下位クワッドワードに移動する。
0F 13 /r	MOVLPS *m64*, *xmm*	2つのパックド単精度浮動小数点値を、*xmm*の下位クワッドワードから *m64* に移動する。

説明

2 つのパックド単精度浮動小数点値を、ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）に移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの下位クワッドワードとメモリの間で、2 つの単精度浮動小数点値を転送することができる。ただし、この命令は、レジスタ同士の間またはメモリ同士の間のデータ転送には使用できない。デスティネーション・オペランドが XMM レジスタの場合は、レジスタの上位クワッドワードは変更されない。

操作

```
MOVLPD instruction for memory to XMM move:
    DEST[63-0] ← SRC ;
    * DEST[127-64] unchanged *;

MOVLPD instruction for XMM to memory move:
    DEST ← SRC[63-0] ;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVLPS      __m128 _mm_loadl_pi ( __m128 a, __m64 *p)
MOVLPS      void _mm_storel_pi (__m64 *p, __m128 a)
```

SIMD 浮動小数点例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。

MOVLPS—Move Low Packed Single-Precision Floating-Point Values（続き）

#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVMSKPD—Extract Packed Double-Precision Floating-Point Sign Mask

オペコード	命令	説明
66 0F 50 /r	MOVMSKPD *r32*, *xmm*	*xmm* から 2 ビットの符号マスクを抽出し、*r32* に格納する。

説明

ソース・オペランド（第 2 オペランド）のパックド倍精度浮動小数点値から符号ビットを抽出して、2 ビット・マスクとしてフォーマットし、デスティネーション・オペランド（第 1 オペランド）に格納する。ソース・オペランドは XMM レジスタである。デスティネーション・オペランドは汎用レジスタである。マスクはデスティネーション・オペランドの最下位 2 ビットに格納される。

操作

```
DEST[0] ← SRC[63];
DEST[1] ← SRC[127];
DEST[3-2] ← 00B;
DEST[31-4] ← 0000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVMSKPD　　int _mm_movemask_pd (__m128 a)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

保護モードと同じ例外。

仮想 8086 モード例外

保護モードと同じ例外。

MOVMSKPS—Extract Packed Single-Precision Floating-Point Sign Mask

オペコード	命令	説明
0F 50 /r	MOVMSKPS *r32*, *xmm*	*xmm* から 4 ビットの符号マスクを抽出し、*r32* に格納する。

説明

ソース・オペランド（第 2 オペランド）のパックド単精度浮動小数点値から符号ビットを抽出して、4 ビット・マスクとしてフォーマットし、デスティネーション・オペランド（第 1 オペランド）に格納する。ソース・オペランドは XMM レジスタである。デスティネーション・オペランドは汎用レジスタである。マスクはデスティネーション・オペランドの最下位 4 ビットに格納される。

操作

```
DEST[0] ← SRC[31];
DEST[1] ← SRC[63];
DEST[1] ← SRC[95];
DEST[1] ← SRC[127];
DEST[31-4] ← 000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

int_mm_movemask_ps(__m128 a)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

保護モードと同じ例外。

仮想 8086 モード例外

保護モードと同じ例外。

MOVNTDQ—Store Double Quadword Using Non-Temporal Hint

オペコード	命令	説明
66 0F E7 /r	MOVNTDQ *m128*, *xmm*	非テンポラルなヒントを使用して、*xmm* から *m128* にダブル・クワッドワードを移動する。

説明

非テンポラルなヒントを使用して、メモリへの書き込み時にデータがキャッシュしないように防ぎ、ソース・オペランド（第2オペランド）のダブル・クワッドワードをデスティネーション・オペランド（第1オペランド）に移動する。ソース・オペランドはXMMレジスタであり、整数データ（パックドバイト、パックドワード、パックド・ダブルワード、またはパックド・クワッドワード）が入っていると見なされる。デスティネーション・オペランドは128ビットのメモリ・ロケーションである。

非テンポラルなヒントを有効にするために、メモリにデータを書き込むとき、ライト・コンバイニング（WC）メモリ・タイプ・プロトコルが使用される。このプロトコルを使用した場合、プロセッサは、キャッシュ階層へのデータの書き込みを行わず、メモリからキャッシュ階層内にキャッシュ・ラインをフェッチしない。非テンポラルなストア先に指定されたメモリアドレスが、キャッシュ不可（UC）または書き込み禁止（WP）メモリ領域内にある場合は、書き込み先領域のメモリタイプのために非テンポラルなヒントが無効になる可能性がある。非テンポラルなストアについての詳細は『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、上巻』の第10章「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと。

WCプロトコルは、順序設定の緩いメモリ整合性モデルを使用する。したがって、複数のプロセッサが、異なるメモリタイプを使用して、同じデスティネーション・メモリ・ロケーションの読み込み / 書き込みを実行する可能性がある場合は、MOVNTDQ命令と合わせてSFENCE命令またはMFENCE命令で実装されるフェンス操作を使用する必要がある。

操作

```
DEST ← SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVNTDQ　　void_mm_stream_si128 (__m128i *p, __m128i a)

SIMD 浮動小数点例外

なし。

MOVNTDQ—Store Double Quadword Using Non-Temporal Hint（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVNTI—Store Doubleword Using Non-Temporal Hint

オペコード	命令	説明
0F C3 /r	MOVNTI *m32*, *r32*	非テンポラルなヒントを使用して、*r32* から *m32* にダブルワードを移動する。

説明

非テンポラルなヒントを使用して、メモリへの書き込み時のキャッシュ汚染を最小限に抑え、ソース・オペランド（第2オペランド）のダブルワード整数をデスティネーション・オペランド（第1オペランド）に移動する。ソース・オペランドは汎用レジスタである。デスティネーション・オペランドは32ビットのメモリ・ロケーションである。

非テンポラルなヒントを有効にするために、メモリにデータを書き込むとき、ライト・コンバイニング（WC）メモリ・タイプ・プロトコルが使用される。このプロトコルを使用した場合、プロセッサは、キャッシュ階層へのデータの書き込みを行わず、メモリからキャッシュ階層内にキャッシュ・ラインをフェッチしない。非テンポラルなストア先に指定されたメモリアドレスが、キャッシュ不可（UC）または書き込み禁止（WP）メモリ領域内にある場合は、書き込み先領域のメモリタイプのために非テンポラルなヒントが無効になる可能性がある。非テンポラルなストアについての詳細は『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、上巻』の第10章「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと。

WCプロトコルは、順序設定の緩いメモリ整合性モデルを使用する。したがって、複数のプロセッサが、異なるメモリタイプを使用して、同じデスティネーション・メモリ・ロケーションの読み込み/書き込みを実行する可能性がある場合は、MOVNTI命令と合わせてSFENCE命令またはMFENCE命令で実装されるフェンス操作を使用する必要がある。

操作

DEST ← SRC;

同等のインテル® C/C++ コンパイラ組み込み関数

MOVNTDQ　　void_mm_stream_si32 (int *p, int a)

SIMD 浮動小数点例外

なし。

MOVNTI—Store Doubleword Using Non-Temporal Hint（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#UD	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#UD	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVNTPD—Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint

オペコード	命令	説明
66 0F 2B /r	MOVNTPD *m128*, *xmm*	非テンポラルなヒントを使用して、*xmm* から *m128* にパックド倍精度浮動小数点値を移動する。

説明

非テンポラルなヒントを使用して、メモリへの書き込み時のキャッシュ汚染を最小限に抑え、ソース・オペランド（第2オペランド）のダブル・クワッドワードをデスティネーション・オペランド（第1オペランド）に移動する。ソース・オペランドはXMMレジスタであり、2つのパックド倍精度浮動小数点値が入っていると見なされる。デスティネーション・オペランドは128ビットのメモリ・ロケーションである。

非テンポラルなヒントを有効にするために、メモリにデータを書き込むとき、ライト・コンバイニング（WC）メモリ・タイプ・プロトコルが使用される。このプロトコルを使用した場合、プロセッサは、キャッシュ階層へのデータの書き込みを行わず、メモリからキャッシュ階層内にキャッシュ・ラインをフェッチしない。非テンポラルなストア先に指定されたメモリアドレスが、キャッシュ不可（UC）または書き込み禁止（WP）メモリ領域内にある場合は、書き込み先領域のメモリタイプのために非テンポラルなヒントが無効になる可能性がある。非テンポラルなストアについての詳細は『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、上巻』の第10章「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと。

WCプロトコルは、順序設定の緩いメモリ整合性モデルを使用する。したがって、複数のプロセッサが、異なるメモリタイプを使用して、同じデスティネーション・メモリ・ロケーションの読み込み / 書き込みを実行する可能性がある場合は、MOVNTPD命令と合わせてSFENCE命令またはMFENCE命令で実装されるフェンス操作を使用する必要がある。

操作

DEST ← SRC;

同等のインテル® C/C++ コンパイラ組み込み関数

MOVNTDQ　　void_mm_stream_pd(double *p, __m128i a)

SIMD 浮動小数点例外

なし。

MOVNTPD—Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVNTPS—Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint

オペコード	命令	説明
0F 2B /r	MOVNTPS *m128*, *xmm*	非テンポラルなヒントを使用して、4 つのパックド単精度浮動小数点値を、*xmm* から *m128* に移動する。

説明

非テンポラルなヒントを使用して、メモリへの書き込み時のキャッシュ汚染を最小限に抑え、ソース・オペランド（第 2 オペランド）のダブル・クワッドワードをデスティネーション・オペランド（第 1 オペランド）に移動する。ソース・オペランドは XMM レジスタであり、4 つのパックド単精度浮動小数点値が入っていると見なされる。デスティネーション・オペランドは 128 ビットのメモリ・ロケーションである。

非テンポラルなヒントを有効にするために、メモリにデータを書き込むとき、ライト・コンバイニング（WC）メモリ・タイプ・プロトコルが使用される。このプロトコルを使用した場合、プロセッサは、キャッシュ階層へのデータの書き込みを行わず、メモリからキャッシュ階層内にキャッシュ・ラインをフェッチしない。非テンポラルなストア先に指定されたメモリアドレスが、キャッシュ不可（UC）または書き込み禁止（WP）メモリ領域内にある場合は、書き込み先領域のメモリタイプのために非テンポラルなヒントが無効になる可能性がある。非テンポラルなストアについての詳細は『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、上巻』の第 10 章「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと。

WC プロトコルは、順序設定の緩いメモリ整合性モデルを使用する。したがって、複数のプロセッサが、異なるメモリタイプを使用して、デスティネーション・メモリ・ロケーションの読み込み / 書き込みを実行する可能性がある場合は、MOVNTPS 命令と合わせてSFENCE命令またはMFENCE命令で実装されるフェンス操作を使用する必要がある。

操作

DEST ← SRC;

同等のインテル® C/C++ コンパイラ組み込み関数

MOVNTDQ　　void_mm_stream_ps(float * p, __m128 a)

SIMD 浮動小数点例外

なし。

MOVNTPS—Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVNTQ—Store of Quadword Using Non-Temporal Hint

オペコード	命令	説明
0F E7 /r	MOVNTQ *m64*, *mm*	非テンポラルなヒントを使用してクワッドワードを、*mm* から *mm64* に移動する。

説明

非テンポラルなヒントを使用して、メモリへの書き込み時のキャッシュ汚染を最小限に抑え、ソース・オペランド（第2オペランド）のクワッドワードをデスティネーション・オペランド（第1オペランド）に移動する。ソース・オペランドはMMXテクノロジ・レジスタであり、パックド整数データ（パックドバイト、パックドワード、またはパックド・ダブルワード）が入っていると見なされる。デスティネーション・オペランドは64ビットのメモリ・ロケーションである。

非テンポラルなヒントを有効にするために、メモリにデータを書き込むとき、ライト・コンバイニング（WC）メモリ・タイプ・プロトコルが使用される。このプロトコルを使用した場合、プロセッサは、キャッシュ階層へのデータの書き込みを行わず、メモリからキャッシュ階層内にキャッシュ・ラインをフェッチしない。非テンポラルなストア先に指定されたメモリアドレスが、キャッシュ不可（UC）または書き込み禁止（WP）メモリ領域内にある場合は、書き込み先領域のメモリタイプのために非テンポラルなヒントが無効になる可能性がある。非テンポラルなストアについての詳細は『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、上巻』の第10章「テンポラルなデータと非テンポラルなデータのキャッシュ処理」を参照のこと。

WCプロトコルは、順序設定の緩いメモリ整合性モデルを使用する。したがって、複数のプロセッサが、異なるメモリタイプを使用して、同じデスティネーション・メモリ・ロケーションの読み込み/書き込みを実行する可能性がある場合は、MOVNTQ命令と合わせてSFENCE命令またはMFENCE命令で実装されるフェンス操作を使用する必要がある。

操作

DEST ← SRC;

同等のインテル® C/C++ コンパイラ組み込み関数

MOVNTQ　　void_mm_stream_pi(__m64 * p, __m64 a)

SIMD 浮動小数点例外

なし。

MOVNTQ—Store of Quadword Using Non-Temporal Hint（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#MF	未処理の x87 FPU 例外がある場合。
#UD	CR0 の EM がセットされた場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#MF	保留中の x87 FPU 例外がある場合。
#UD	CR0 の EM がセットされた場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVSHDUP—Move Packed Single-FP High and Duplicate

オペコード	命令	説明
F3, 0F, 16, /r	MOVSHDUP *xmm1*, *xmm2/m128*	*xmm2/m128* 内の各クワッドワードの上位 32 ビット・オペランドから 2 つの単精度浮動小数点値を *xmm1* に移動し、各 32 ビット・オペランドを各クワッドワードの下位 32 ビットに複製する。

説明

リニアアドレスは、参照されるメモリデータの最下位バイトのアドレスに対応する。メモリアドレスが指定されると、メモリ上の位置 m128 から 16 バイトのデータがロードされ、位置 1 と位置 3 の単精度要素が複製される。レジスタ ー レジスタ形式の操作を使用した場合は、128 ビット・ソース・レジスタから得られるデータに対して同じ操作が実行される。図 3-15. を参照のこと。

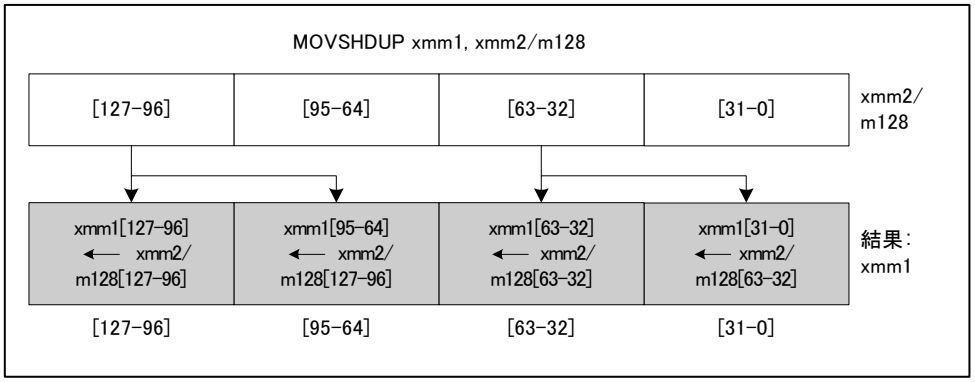

図 3-15. MOVSHDUP: Move Packed Single-FP High and Duplicate

MOVSHDUP—Move Packed Single-FP High and Duplicate（続き）

操作

```
if (source == m128) {
        // load instruction
        xmm1[31-0] = m128[63-32];
        xmm1[63-32] = m128[63-32]
        xmm1[95-64]  = m128[127-96];
        xmm1[127-96] = m128[127-96];
}
else {
        // move instruction
        xmm1[31-0] = xmm2[63-32];
        xmm1[63-32] = xmm2[63-32];
        xmm1[95-64] = xmm2[127-96];
        xmm1[127-96] = xmm2[127-96];
}
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVSHDUP　__m128 _mm_movehdup_ps(__m128 a)

例外

アライメントが16バイトに合っていない場合は、セグメントに関係なく、一般保護例外が発生する。

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、またはGSセグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SSセグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0のTSビットがセットされている場合。
#UD	CR0.EMが1の場合。
	CR4.OSFXSR（ビット9）が0の場合。
	CPUID.SSE3（ECXビット0）が0の場合。

MOVSHDUP—Move Packed Single-FP High and Duplicate（続き）

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

MOVSLDUP—Move Packed Single-FP Low and Duplicate

オペコード	命令	説明
F3, 0F, 12, /r	MOVSLDUP *xmm1*, *xmm2/m128*	パック単精度データ要素を表す 128 ビットを *xmm2/m128* から *xmm1* レジスタに移動し、下位の要素を複製する。

説明

リニアアドレスは、参照されるメモリデータの最下位バイトのアドレスに対応する。メモリアドレスが指定されると、メモリ上の位置 m128 から 16 バイトのデータがロードされ、位置 0 と位置 2 の単精度要素が複製される。レジスタ ー レジスタ形式の操作を使用した場合は、128 ビット・ソース・レジスタから得られるデータに対して同じ操作が実行される。図 3-16. を参照のこと。

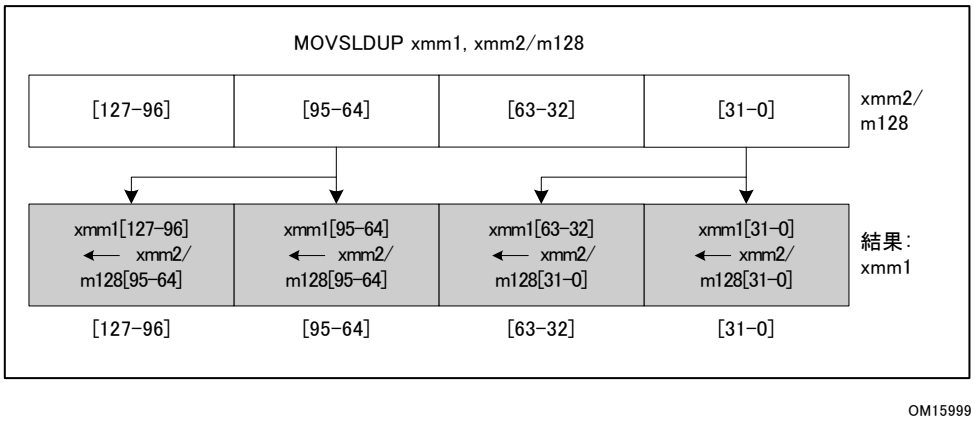

図 3-16. MOVSLDUP: Move Packed Single-FP Low and Duplicate

MOVSLDUP—Move Packed Single-FP Low and Duplicate（続き）

操作

```
if (source == m128) {
        // load instruction
        xmm1[31-0]   = m128[31-0];
        xmm1[63-32] = m128[31-0]
        xmm1[95-64] = m128[95-64];
        xmm1[127-96] = m128[95-64];
}
else {
        // move instruction
        xmm1[31-0]   = xmm2[31-0];
        xmm1[63-32]  = xmm2[31-0];
        xmm1[95-64]  = xmm2[95-64];
        xmm1[127-96] = xmm2[95-64];
}
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVSLDUP　　__m128 _mm_moveldup_ps(__m128 a)

例外

アライメントが 16 バイトに合っていない場合は、セグメントに関係なく、一般保護例外が発生する。

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

MOVSLDUP—Move Packed Single-FP Low and Duplicate（続き）

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。

仮想 8086 モード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS ビットがセットされている場合。
#UD	CR0.EM が 1 の場合。
	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE3（ECX ビット 0）が 0 の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

MOVQ—Move Quadword

オペコード	命令	説明
0F 6F /r	MOVQ *mm*, *mm/m64*	クワッドワードを *mm/m64* から *mm* に転送する。
0F 7F /r	MOVQ *mm/m64*, *mm*	クワッドワードを *mm* から *mm/m64* に転送する。
F3 0F 7E	MOVQ *xmm1*, *xmm2/m64*	クワッドワードを*xmm2/mem64*から*xmm1*に転送する。
66 0F D6	MOVQ *xmm2/m64*, *xmm1*	クワッドワードを*xmm1*から*xmm2/mem64*に転送する。

説明

クワッドワードをソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）にコピーする。ソース・オペランドまたはデスティネーション・オペランドには、MMX® テクノロジ・レジスタ、XMM レジスタ、または 64 ビット・メモリ・ロケーションを使用できる。この命令を使用して、2 つの MMX テクノロジ・レジスタの間、MMX テクノロジ・レジスタと 64 ビットのメモリ・ロケーションの間でクワッドワードを転送するか、2 つの XMM レジスタの間、または XMM レジスタと 64 ビットのメモリ・ロケーションの間で、データを転送することができる。ただし、この命令は、2 つのメモリ・ロケーションの間のデータ転送には使用できない。

ソース・オペランドが XMM レジスタの場合は、下位クワッドワードが転送される。デスティネーション・オペランドが XMM レジスタの場合は、転送されたクワッドワードは XMM レジスタの下位クワッドワードに格納され、上位クワッドワードはすべて 0 にクリアされる。

操作

```
MOVQ instruction when operating on MMX technology registers and memory locations:
    DEST ← SRC;
MOVQ instruction when source and destination operands are XMM registers:
    DEST[63-0] ← SRC[63-0];
MOVQ instruction when source opernad is XMM register and destination
operand is memory location:
    DEST ← SRC[63-0];
MOVQ instruction when source operand is memory location and destination
operand is XMM register:
    DEST[63-0] ← SRC;
    DEST[127-64] ← 0000000000000000H;
```

影響を受けるフラグ

なし。

SIMD 浮動小数点例外

なし。

MOVQ—Move Quadword（続き）

保護モード例外

#GP(0)	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合にのみ、128 ビット操作によって #UD が発生する。SSE2 に対応していないプロセッサ（MMX テクノロジ対応プロセッサ）上で 128 ビット命令を実行した場合、その命令は mm レジスタを操作し、#UD は発生しない。
#NM	CR0 の TS がセットされた場合。
#MF	(MMXテクノロジ・レジスタ操作のみ)未処理のFPU例外がある場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	オペランドのいずれかの部分が実効アドレス空間 0 ～ FFFFH の外にある場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合にのみ、128 ビット操作によって #UD が発生する。SSE2 に対応していないプロセッサ（MMX テクノロジ対応プロセッサ）上で 128 ビット命令を実行した場合、その命令は mm レジスタを操作し、#UD は発生しない。
#NM	CR0 の TS がセットされた場合。
#MF	(MMXテクノロジ・レジスタ操作のみ)未処理のFPU例外がある場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

MOVQ2DQ—Move Quadword from MMX Technology to XMM Register

オペコード	命令	説明
F3 0F D6	MOVQ2DQ *xmm*, *mm*	*mmx* のクワッドワードを *xmm* の下位クワッドワードに移動する。

説明

ソース・オペランド（第 2 オペランド）のクワッドワードを、デスティネーション・オペランド（第 1 オペランド）の下位クワッドワードに移動する。ソース・オペランドはMMX® テクノロジ・レジスタである。デスティネーション・オペランドはXMM レジスタである。

この命令を使用すると、x87 FPU 操作から MMX テクノロジ操作への移行が発生する（つまり、x87 FPUのトップオブスタック・ポインタは 0 に設定され、x87 FPU タグワードはすべて 0[有効] に設定される）。未処理の x87 FPU 浮動小数点例外があるときにこの命令を実行しようとすると、MOVQ2DQ 命令を実行する前にその例外が処理される。

操作

```
DEST[63-0] ← SRC[63-0];
DEST[127-64] ← 00000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MOVQ2DQ　　__128i _mm_movpi64_pi64 (__m64 a)

SIMD 浮動小数点例外

なし。

保護モード例外

#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#MF	未処理の x87 FPU 例外がある場合。

実アドレスモード例外

保護モードと同じ例外。

MOVQ2DQ—Move Quadword from MMX Technology to XMM Register（続き）

仮想 8086 モード例外

保護モードと同じ例外。

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String

オペコード	命令	説明
A4	MOVS m8, m8	アドレスDS:(E)SIのバイトをアドレスES:(E)DIに転送する。
A5	MOVS m16, m16	アドレスDS:(E)SIのワードをアドレスES:(E)DIに転送する。
A5	MOVS m32, m32	アドレス DS:(E)SI のダブルワードをアドレス ES:(E)DI に転送する。
A4	MOVSB	アドレスDS:(E)SIのバイトをアドレスES:(E)DIに転送する。
A5	MOVSW	アドレスDS:(E)SIのワードをアドレスES:(E)DIに転送する。
A5	MOVSD	アドレス DS:(E)SI のダブルワードをアドレス ES:(E)DI に転送する。

説明

第2オペランド（ソース・オペランド）で指定されたバイト、ワード、またはダブルワードを第1オペランド（デスティネーション・オペランド）で指定されたロケーションに転送する。ソース・オペランドとデスティネーション・オペランドの両方とも、メモリに位置している。ソース・オペランドのアドレスは、（命令のアドレスサイズ属性、32または16に応じて）それぞれDS:ESIレジスタまたはDS:SIレジスタから読み取られる。デスティネーション・オペランドのアドレスは、（やはり命令のアドレスサイズ属性に応じて）ES:EDIレジスタまたはES:DIレジスタから読み取られる。DSセグメントは、セグメント・オーバーライド・プリフィックスでオーバーライドすることができるが、ESセグメントをオーバーライドすることはできない。

アセンブリ・コード・レベルでは、この命令の「明示オペランド」形式と「オペランドなし」形式という2つの形式が使用できる。（MOVSニーモニックで指定される）明示オペランド形式では、ソース・オペランドとデスティネーション・オペランドを明示的に指定できる。この場合、ソース・オペランドとデスティネーション・オペランドは、それぞれソース値とデスティネーションのサイズとロケーションを示す記号でなければならない。この明示オペランド形式は、ドキュメンテーションを可能にするために設けられたものであるが、この形式によって提供されるドキュメンテーションは誤解を招く場合があるので注意する。すなわち、ソース・オペランドとデスティネーション・オペランドの記号は、オペランドの正しい**タイプ**（サイズ: バイト、ワード、またはダブルワード）を指定しなければならないが、正しい**ロケーション**を指定する必要はない。ソース・オペランドとデスティネーション・オペランドのロケーションは、常にDS:(E)SIレジスタとES:(E)DIレジスタによって指定されるので、ストリング移動命令を実行する前に、これらのレジスタに正しくロードされていなければならない。

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String（続き）

オペランドなし形式は、MOVS命令のバイト、ワード、ダブルワード各バージョンの「ショート形式」を提供する。この場合も、DS:(E)SIとES:(E)DIがそれぞれソース・オペランドとデスティネーション・オペランドであると想定される。ソース・オペランドとデスティネーション・オペランドのサイズは、MOVSB（バイト移動）、MOVSW（ワード移動）、またはMOVSD（ダブルワード移動）の各ニーモニックで選択される。

移動操作の後、(E)SIレジスタと(E)DIレジスタは、EFLAGSレジスタ内のDFフラグの設定にしたがって自動的にインクリメントまたはデクリメントされる。（DFフラグが0である場合は、(E)SIレジスタと(E)DIレジスタはインクリメントされ、DFフラグが1である場合は、(E)SIレジスタと(E)DIレジスタはデクリメントされる。）これらのレジスタは、バイト操作の場合は1、ワード操作の場合は2、ダブルワード操作の場合は4、それぞれインクリメントまたはデクリメントされる。

MOVS、MOVSB、MOVSW、MOVSD命令は、前にREPプリフィックスを付けることにより、ECXバイト、ワード、またはダブルワードのブロック転送を行うことができる。（本章の「REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix」を参照）。

操作

```
DEST ←SRC;
IF (byte move)
    THEN IF DF = 0
        THEN
            (E)SI ← (E)SI + 1;
            (E)DI ← (E)DI + 1;
        ELSE
            (E)SI ← (E)SI – 1;
            (E)DI ← (E)DI – 1;
        FI;
    ELSE IF (word move)
        THEN IF DF = 0
            (E)SI ← (E)SI + 2;
            (E)DI ← (E)DI + 2;
        ELSE
            (E)SI ← (E)SI – 2;
            (E)DI ← (E)DI – 2;
        FI;
```

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String（続き）

```
    ELSE (* doubleword move*)
        THEN IF DF = 0
            (E)SI ← (E)SI + 4;
            (E)DI ← (E)DI + 4;
        ELSE
            (E)SI ← (E)SI – 4;
            (E)DI ← (E)DI – 4;
        FI;
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	デスティネーション・オペランドが書き込み不可能なセグメントにある場合。
	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

MOVSD—Move Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 10 /r	MOVSD *xmm1*, *xmm2/m64*	スカラ倍精度浮動小数点値を *xmm2/m64* から *xmm1* レジスタに移動する。
F2 0F 11 /r	MOVSD *xmm2/m64*, *xmm*	スカラ倍精度浮動小数点値を *xmm1* レジスタから *xmm2/m64* に移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）にスカラ倍精度浮動小数点値を移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの下位クワッドワードと 64 ビットのメモリ・ロケーションの間、または 2 つの XMM レジスタの下位クワッドワードの間で、倍精度浮動小数点値を転送することができる。ただし、この命令は、2 つのメモリ・ロケーションの間のデータ転送には使用できない。

ソース・オペランドとデスティネーション・オペランドが XMM レジスタの場合は、デスティネーション・オペランドの上位クワッドワードは変更されない。ソース・オペランドがメモリ・ロケーションで、デスティネーション・オペランドが XMM レジスタの場合は、デスティネーション・オペランドの上位クワッドワードはすべて 0 にクリアされる。

操作

```
MOVSD instruction when source and destination operands are XMM registers:
    DEST[63-0] ← SRC[63-0];
    * DEST[127-64] remains unchanged *;
MOVSD instruction when source operand is XMM register and destination
operand is memory location:
    DEST ← SRC[63-0];
MOVSD instruction when source operand is memory location and destination
operand is XMM register:
    DEST[63-0] ← SRC;
    DEST[127-64] ← 0000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVSD       __m128d _mm_load_sd (double *p)
MOVSD       void _mm_store_sd (double *p, __m128d a)
MOVSD       __m128d _mm_store_sd (__m128d a, __m128d b)
```

SIMD 浮動小数点例外

なし。

MOVSD—Move Scalar Double-Precision Floating-Point Value（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVSS—Move Scalar Single--Precision Floating-Point Values

オペコード	命令	説明
F3 0F 10 /r	MOVSS *xmm1*, *xmm2/m32*	スカラ単精度浮動小数点値を、*xmm2/m64* から *xmm1* レジスタに移動する。
F3 0F 11 /r	MOVSS *xmm2/m32*, *xmm1*	スカラ単精度浮動小数点値を、*xmm1* レジスタから *xmm2/m64* に移動する。

説明

ソース・オペランド（第 2 オペランド）からデスティネーション・オペランド（第 1 オペランド）にスカラ単精度浮動小数点値を移動する。ソース・オペランドとデスティネーション・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。この命令を使用して、XMM レジスタの最下位のダブルワードと 32 ビットのメモリ・ロケーションの間、または 2 つの XMM レジスタの最下位のダブルワードの間で、単精度浮動小数点値を転送することができる。ただし、この命令は、2 つのメモリ・ロケーションの間のデータ転送には使用できない。

ソース・オペランドとデスティネーション・オペランドが XMM レジスタの場合は、デスティネーション・オペランドの上位 3 つのダブルワードは変更されない。ソース・オペランドがメモリ・ロケーションで、デスティネーション・オペランドが XMM レジスタの場合は、デスティネーション・オペランドの上位 3 つのダブルワードはすべて 0 にクリアされる。

操作

```
MOVSS instruction when source and destination operands are XMM registers:
    DEST[31-0] ← SRC[31-0];
    * DEST[127-32] remains unchanged *;
MOVSS instruction when source operand is XMM register and destination
operand is memory location:
    DEST ← SRC[31-0];
MOVSS instruction when source operand is memory location and destination
operand is XMM register:
    DEST[31-0] ← SRC;
    DEST[127-32] ← 00000000000000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVSS       __m128 _mm_load_ss(float * p)
MOVSS       void_mm_store_ss(float * p, __m128 a)
MOVSS       __m128 _mm_move_ss(__m128 a, __m128 b)
```

SIMD 浮動小数点例外

なし。

MOVSS—Move Scalar Single--Precision Floating-Point Values（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照を行った場合。

MOVSX—Move with Sign-Extension

オペコード	命令	説明
0F BE /r	MOVSX *r16, r/m8*	バイトをワードに符号拡張して転送する。
0F BE /r	MOVSX *r32, r/m8*	バイトをダブルワードに符号拡張して転送する。
0F BF /r	MOVSX *r32, r/m16*	ワードをダブルワードに符号拡張して転送する。

説明

ソース・オペランド（レジスタまたはメモリ・ロケーション）の内容をデスティネーション・オペランド（レジスタ）にコピーし、値を 16 ビットまたは 32 ビットに符号拡張する。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、上巻』の図 7-6. を参照。変換された値のサイズは、オペランド・サイズ属性に依存する。

操作

```
DEST ← SignExtend(SRC);
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

MOVSX—Move with Sign-Extension（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。

MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 10 /r	MOVUPD *xmm1*, *xmm2/m128*	パック ド倍精度浮動小数点値を、*xmm2/m128* から *xmm1* に移動する。
66 0F 11 /r	MOVUPD *xmm2/m128*, *xmm*	パック ド倍精度浮動小数点値を、*xmm1* から *xmm2/m128* に移動する。

説明

2つのパック ド倍精度浮動小数点値が入っているダブル・クワッドワードを、ソース・オペランド（第2オペランド）からデスティネーション・オペランド（第1オペランド）に移動する。この命令を使用して、XMM レジスタを128ビットのメモリ・ロケーションからロードする操作、XMM レジスタの内容を128ビットのメモリ・ロケーションにストアする操作、または2つの XMM レジスタの間でデータを転送する操作が可能である。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合、そのオペランドのアライメントが16バイトに合っていなくても、一般保護例外（#GP）は発生しない。

倍精度浮動小数点値の転送先または転送元のメモリ・ロケーションのアライメントが16バイトに合っていることがわかっている場合は、MOVAPD 命令を使用する。

16ビット・アドレス指定モードでの実行中に、128ビット・データ・アクセスのリニアアドレスが16ビット・セグメントの終点からはみ出すことは許されない。これは予約済みの動作として定義されている。この場合に一般保護例外（#GP）が発生するかどうかは、プロセッサによって異なる。16ビット・セグメントの範囲を超えたアドレスは、そのセグメントの始点にラップアラウンドされることも、ラップアラウンドされないこともある。

操作

DEST ← SRC;

同等のインテル® C/C++ コンパイラ組み込み関数

MOVUPD __m128 _mm_loadu_pd(double * p)
MOVUPD void_mm_storeu_pd(double *p, __m128 a)

SIMD 浮動小数点例外

なし。

MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 10 /r	MOVUPS *xmm1*, *xmm2/m128*	パック ド単精度浮動小数点値を、*xmm2/m128* から *xmm1* レジスタに移動する。
0F 11 /r	MOVUPS *xmm2/m128*, *xmm1*	パック ド単精度浮動小数点値を、*xmm1* レジスタから *xmm2/m128* に移動する。

説明

4つのパック ド単精度浮動小数点値が入っているダブル・クワッドワードを、ソース・オペランド（第2オペランド）からデスティネーション・オペランド（第1オペランド）に移動する。この命令を使用して、XMM レジスタを128ビットのメモリ・ロケーションからロードする操作、XMM レジスタの内容を128ビットのメモリ・ロケーションにストアする操作、または2つの XMM レジスタの間でデータを転送する操作が可能である。ソース・オペランドまたはデスティネーション・オペランドがメモリ・オペランドの場合、そのオペランドのアライメントが16バイトに合っていなくても、一般保護例外（#GP）は発生しない。

パック ド単精度浮動小数点値の転送先または転送元のメモリ・ロケーションのアライメントが16バイトに合っていることがわかっている場合は、MOVAPS 命令を使用する。

16ビット・アドレス指定モードでの実行中に、128ビット・データ・アクセスのリニアアドレスが16ビット・セグメントの終わりと重複することは許されない。この場合に一般保護例外（#GP）が発生するかどうかは、プロセッサによって異なる。16ビット・セグメントの範囲を超えたアドレスは、そのセグメントの始点にラップアラウンドされることも、ラップアラウンドされないこともある。

操作

```
DEST ← SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
MOVUPS      __m128 _mm_loadu_ps(double * p)
MOVUPS      void_mm_storeu_ps(double *p, __m128 a)
```

SIMD 浮動小数点例外

なし。

MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values（続き）

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#UD	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MOVZX—Move with Zero-Extend

オペコード	命令	説明
0F B6 /r	MOVZX *r16*, *r/m8*	バイトをワードにゼロ拡張して転送する。
0F B6 /r	MOVZX *r32*, *r/m8*	バイトをダブルワードにゼロ拡張して転送する。
0F B7 /r	MOVZX *r32*, *r/m16*	ワードをダブルワードにゼロ拡張して転送する。

説明

ソース・オペランド（レジスタまたはメモリ・ロケーション）の内容をデスティネーション・オペランド（レジスタ）にコピーし、値を 16 ビットまたは 32 ビットにゼロ拡張する。変換された値のサイズは、オペランド・サイズ属性に依存する。

操作

DEST ← ZeroExtend(SRC);

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

MOVZX—Move with Zero-Extend（続き）

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

MUL—Unsigned Multiply

オペコード	命令	説明
F6 /4	MUL *r/m8*	符号なし乗算（AX ← AL∗ *r/m8*）
F7 /4	MUL *r/m16*	符号なし乗算（DX:AX ← AX ∗ *r/m16*）
F7 /4	MUL *r/m32*	符号なし乗算（EDX:EAX ← EAX∗ *r/m32*）

説明

第 1 オペランド（デスティネーション・オペランド）と第 2 オペランド（ソース・オペランド）との符号なし乗算を行い、結果をデスティネーション・オペランドにストアする。デスティネーション・オペランドは、（オペランドのサイズに応じて）AL、AX、または EAX レジスタとなる暗黙のオペランドである。ソース・オペランドは、汎用レジスタまたはメモリ・ロケーションである。この命令の処置と結果のロケーションは、以下の表に示すようにオペコードとオペランド・サイズによって変わる。

オペランド・サイズ	ソース 1	ソース 2	デスティネーション
バイト	AL	r/m8	AX
ワード	AX	r/m16	DX:AX
ダブルワード	EAX	r/m32	EDX:EAX

結果は、（オペランド・サイズに応じて）レジスタ AX、レジスタペア DX:AX、またはレジスタペア EDX:EAX にストアされ、積の上位ビットは、それぞれ AH、DX、または EDX に入る。積の上位ビットが 0 である場合は、CF フラグと OF フラグがクリアされ、そうでない場合は、これらのフラグが 1 にセットされる。

操作

```
IF byte operation
    THEN
        AX ← AL ∗ SRC
    ELSE (* word or doubleword operation *)
        IF OperandSize = 16
            THEN
                DX:AX ← AX ∗ SRC
            ELSE (* OperandSize = 32 *)
                EDX:EAX ← EAX ∗ SRC
        FI;
FI;
```

影響を受けるフラグ

結果の上半分が 0 である場合は、OF フラグと CF フラグが 0 にクリアされる。そうでない場合は、それらのフラグが 1 にセットされる。SF、ZF、AF、PF フラグは未定義。

MUL—Unsigned Multiply（続き）

保護モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
	DS、ES、FS、または GS レジスタの内容がヌル・セグメント・セレクタの場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	現行特権レベルが 3 のときに、アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

実アドレスモード例外

#GP	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。

仮想 8086 モード例外

#GP(0)	メモリ・オペランドの実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。
#SS(0)	メモリ・オペランドの実効アドレスが SS セグメントの範囲外の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、アライメントが合わないメモリ参照が行われた場合。

MULPD—Multiply Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 59 /r	MULPD *xmm1*, *xmm2/m128*	*xmm2/m128* のパック ド倍精度浮動小数点値に *xmm1* のパック ド倍精度浮動小数点値を掛ける。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 2 つのパックド倍精度浮動小数点値の SIMD 乗算を実行し、結果のパックド倍精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。倍精度浮動小数点値の SIMD 演算の図は、『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 11-3. を参照のこと。

操作

```
DEST[63-0] ← DEST[63-0] * SRC[63-0];
DEST[127-64] ← DEST[127-64] * SRC[127-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

MULPD __m128d _mm_mul_pd (m128d a, m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

MULPD—Multiply Packed Double-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE2 が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MULPS—Multiply Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 59 /r	MULPS *xmm1*, *xmm2/m128*	*xmm1* のパック ド単精度浮動小数点値に *xmm2/mem* のパック ド単精度浮動小数点値を掛けて、結果を *xmm1* レジスタにストアする。

説明

ソース・オペランド（第 2 オペランド）とデスティネーション・オペランド（第 1 オペランド）の 4 つのパックド単精度浮動小数点値の SIMD 乗算を実行し、結果のパックド単精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 128 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。単精度浮動小数点値の SIMD 演算の図は、『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 10-5. を参照のこと。

操作

```
DEST[31-0] ← DEST[31-0] * SRC[31-0];
DEST[63-32] ← DEST[63-32] * SRC[63-32];
DEST[95-64] ← DEST[95-64] * SRC[95-64];
DEST[127-96] ← DEST[127-96] * SRC[127-96];
```

同等のインテル® C/C++ コンパイラ組み込み関数

MULPS __m128 _mm_mul_ps(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

MULPS—Multiply Packed Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

実アドレスモード例外

#GP(0)	セグメントに関係なく、メモリ・オペランドのアライメントが 16 バイトに合っていない場合。 オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。 CR0 の EM がセットされた場合。 CR4 の OSFXSR が 0 の場合。 CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。

MULSD—Multiply Scalar Double-Precision Floating-Point Values

オペコード	命令	説明
F2 0F 59 /r	MULSD *xmm1*, *xmm2/m64*	*xmm2/mem64* の下位の倍精度浮動小数点値に *xmm1* の下位の倍精度浮動小数点値を掛ける。

説明

ソース・オペランド（第 2 オペランド）の下位の倍精度浮動小数点値にデスティネーション・オペランド（第 1 オペランド）の下位の倍精度浮動小数点値を掛けて、結果の倍精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 64 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位クワッドワードは変更されない。倍精度浮動小数点値のスカラ演算の図は、『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 11-4. を参照のこと。

操作

```
DEST[63-0] ← DEST[63-0] * xmm2/m64[63-0];
* DEST[127-64] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MULSD __m128d _mm_mul_sd (m128d a, m128d b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

MULSD—Multiply Scalar Double-Precision Floating-Point Values（続き）

#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC	現行特権レベルが 3 のときに、アライメントの合っていないメモリ参照を行った場合。

MULSS—Multiply Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F 59 /r	MULSS *xmm1*, *xmm2/m32*	*xmm1* の下位の単精度浮動小数点値に *xmm2/mem* の下位の単精度浮動小数点値を掛けて、結果を *xmm1* レジスタにストアする。

説明

ソース・オペランド（第2オペランド）の最下位の単精度浮動小数点値にデスティネーション・オペランド（第1オペランド）の最下位の単精度浮動小数点値を掛けて、結果の単精度浮動小数点値をデスティネーション・オペランドに格納する。ソース・オペランドは、XMM レジスタまたは 32 ビットのメモリ・ロケーションである。デスティネーション・オペランドは XMM レジスタである。デスティネーション・オペランドの上位 3 つのダブルワードは変更されない。単精度浮動小数点値のスカラ演算の図は、『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 10-6. を参照のこと。

操作

```
DEST[31-0] ← DEST[31-0] * SRC[31-0];
* DEST[127-32] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

MULSS __m128 _mm_mul_ss(__m128 a, __m128 b)

SIMD 浮動小数点例外

オーバーフロー、アンダーフロー、無効、精度、デノーマル。

保護モード例外

#GP(0)	CS、DS、ES、FS、または GS セグメント内のメモリ・オペランドの実効アドレスが無効の場合。
#SS(0)	SS セグメント内のアドレスが無効の場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。

MULSS—Multiply Scalar Single-Precision Floating-Point Values（続き）

#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。
#AC(0)	アライメント・チェックがイネーブルにされていて、現行特権レベルが 3 のときにアライメントが合わないメモリ参照を行った場合。

実アドレスモード例外

GP(0)	オペランドの一部が 0 ～ FFFFH の実効アドレス空間の範囲外の場合。
#NM	CR0 の TS がセットされた場合。
#XM	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 1 の場合。
#UD	マスクされていない SIMD 浮動小数点例外が発生し、CR4 の OSXMMEXCPT が 0 の場合。
	CR0 の EM がセットされた場合。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE が 0 の場合。

仮想 8086 モード例外

実アドレスモードと同じ例外。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC	アライメントの合っていないメモリ参照を行った場合。

MWAIT—Monitor Wait

Opcode	Instruction	Description
0F, 01, C9	MWAIT	特定のクラスのイベントが発生するまで、プロセッサが命令の実行を中止し、プロセッサに依存する最適化された状態に移行できるようにするヒント。これはアーキテクチャ的には NOP 命令と同じ命令である。

説明

MWAIT 命令は、MONITOR 命令と合わせて使用されるように設計されている。MONITOR 命令は 'wait' アドレスを定義し、MWAIT 命令は事前に定義された「インプリメンテーション・デペンド・オプティマイズド・オペレーション」を 'wait' アドレスから開始させる。MWAIT の実行は、プログラム・フロー内の先行する MONITOR 命令によって設定されたアドレス範囲へのストア操作または特定のイベントを待機する間、インプリメンテーション・デペンド・オプティマイズド・ステータスに移行できることを示す、プロセッサに対するヒントになる。

EAX と ECX を使用して、プロセッサが移行するオプティマイズド・ステータスの種類など、その他の情報を MWAIT 命令に伝達できる。ECX は、MWAIT 命令のオプションの拡張機能を指定する。EAX は、プロセッサが移行する優先された最適な状態などのヒントを指定できる。ファミリ = 15 およびモデル = 3 の CPUID シグネチャを持つインテル® Pentium® 4 プロセッサでは、EAX と ECX のゼロ以外の値はすべて予約されている。ECX に予約済みの値を指定して MWAIT 命令を実行すると、一般保護フォルトが発生する。EAX の予約ビットをセットして MWAIT 命令を実行すると、プロセッサはそれらのビットを無視する。

MONITOR 命令によって設定されたアドレス範囲へのストア、割り込み、NMI または SMI、デバッグ例外、マシンチェック例外、BINIT# 信号、INIT# 信号、または RESET# 信号が検出されると、インプリメンテーション・デペンド・オプティマイズド・ステータスは終了する。ただし、割り込みによってオプティマイズド・ステータスが終了するのは、その状態に移行したときに割り込みがイネーブルにされていた場合に限られる。

MWAIT—Monitor Wait（続き）

指定されたアドレス範囲へのストアによってオプティマイズド・ステータスが終了した場合は、MWAIT 命令に続く命令から実行が再開される。割り込み（NMI を含む）によってインプリメンテーション・デペンド・オプティマイズド・ステータス・プロセッサが終了した場合は、プロセッサはその状態を終了して割り込みを処理する。SMI によってインプリメンテーション・デペンド・オプティマイズド・ステータスが終了した場合は、SMI の処理の実行後、MWAIT に続く命令から実行が再開される。HLT 命令とは異なり、MWAIT 命令は、MWAIT 命令での実行の再開をサポートしていない。これ以外にも、インプリメンテーション・デペンド・オプティマイズド・ステータスを終了させ、MWAIT に続く命令で実行を再開させる、インプリメンテーション・デペンド・イベントまたはタイムアウトが存在する可能性がある。

先行する MONITOR 命令によってアドレス範囲が正しく設定されていない場合や、MWAIT の実行前に MONITOR 命令が実行されていない場合は、プロセッサは、インプリメンテーション・デペンド・オプティマイズド・ステータスに移行しない。実行は、MWAIT に続く命令から再開される。

MWAIT 命令はすべての特権レベルで実行できる。CPUID 機能フラグ MONITOR（EAX=1 で CPUID を実行した場合、ECX[ビット 3]）は、プロセッサ上での MONITOR 命令と MWAIT 命令の利用可能性を示す。このフラグがセットされている場合、特権レベル 0 では MWAIT を無条件で実行でき、特権レベル 1 ～ 3 では条件付きで実行できる（ソフトウェアは、これらの命令を無条件で使用する前に、これらの命令が適切にサポートされているかをテストする必要がある）。

オペレーティング・システムまたはシステム BIOS は、IA32_MISC_ENABLES MSR を使用してこの命令をディスエーブルにできる。この命令をディスエーブルにすると、該当する CPUID 機能フラグがクリアされる。この状態で MWAIT 命令を実行すると、無効オペコード例外が発生する。

操作

```
// MWAIT takes the argument in EAX as a hint extension and is
// architected to take the argument in ECX as an instruction
// extension
// MWAIT EAX, ECX
{
WHILE (! ("Monitor Hardware is in armed state")) {
        implementation_dependent_optimized_state(EAX, ECX);
}
Set the state of Monitor Hardware as Triggered;
}
```

MWAIT—Monitor Wait（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

MWAIT　　void _mm_mwait(unsigned extensions, unsigned hints)

例

MWAIT 命令の実行によってモニタ・ハードウェアがトリガされるため、MONITOR 命令と MWAIT 命令は同じループ内でコーディングしなければならない。MONITOR 命令を一度実行してから MWAIT 命令をループ内で実行するのは、正しい使い方ではない。MONITOR をセットアップして MWAIT を実行しなくても、悪影響はない。

通常、MONITOR/MWAIT のペアは、次のようなシーケンスで使用される。

```
EAX = Logical Address(Trigger)
ECX = 0    // Hints
EDX = 0    // Hints
If ( !trigger_store_happened) {
           MONITOR EAX, ECX, EDX
           If ( !trigger_store_happened ) {
           MWAIT EAX, ECX
           }
}
```

上記のコード・シーケンスは、トリガの 1 回目のチェックとモニタ命令の実行の間にトリガとなるイベントが発生していないことを確認する。2 回目のチェックがなければ、トリガとなるストアは検出されないままになる。MONITOR と MWAIT の一般的な使い方では、上記のコード・シーケンスをループの中で使用する。

例外

なし。

数値例外

なし。

保護モード例外

#GP(0)	ECX の値が 0 以外の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合。
	この命令が利用可能でないときに特権レベル 1 ～ 3 で実行した場合。
	LOCK プリフィックスが使用されている場合。
	REPE、REPNE、またはオペランド・サイズのプリフィックスが使用されている場合。

MWAIT—Monitor Wait（続き）

実アドレスモード例外

#GP(0)	ECX の値が 0 以外の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合。
	LOCK プリフィックスが使用されている場合。
	REPE、REPNE、またはオペランド・サイズのプリフィックスが使用されている場合。

仮想 8086 モード例外

#GP(0)	ECX の値が 0 以外の場合。
#UD	CPUID 機能フラグ MONITOR が 0 の場合、またはこの命令が利用可能でないときに特権レベル 1-2-3 で実行した場合。
	LOCK プリフィックスが使用されている場合。
	REPE、REPNE、またはオペランド・サイズのプリフィックスが使用されている場合。