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 B：
命令セット・リファレンス N-Z

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。

インテル、Intel ロゴ、Intel386、Intel486、Intel NetBurst、Celeron、MMX、Pentium、Xeon は、アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標、登録商標です。

* その他の社名、製品名などは、一般に各社の商標または登録商標です。

© 1997-2004, Intel Corporation.

4

命令セット・リファレンス N-Z

第 4 章
命令セット・リファレンス
N-Z

4

第 4 章では、第 3 章に続き、IA-32 命令（N-Z）についてアルファベット順に説明する。IA-32 命令の前半部分（A-M）については『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻 A』を参照のこと。

NEG—Two's Complement Negation

オペコード	命令	説明
F6 /3	NEG *r/m8*	2 の補数が *r/m8* をネゲートする。
F7 /3	NEG *r/m16*	2 の補数が *r/m16* をネゲートする。
F7 /3	NEG *r/m32*	2 の補数が *r/m32* をネゲートする。

説明

オペランド（デスティネーション・オペランド）の値をその 2 の補数で置き換える。（この操作は、オペランドの 0 からの減算と同等である。）デスティネーション・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

```
IF DEST = 0
    THEN CF ← 0
    ELSE CF ← 1;
FI;
DEST ← – (DEST)
```

影響を受けるフラグ

ソース・オペランドが 0 である場合は、CF フラグが 0 にセットされる。そうでない場合は、CF フラグが 1 にセットされる。OF、SF、ZF、AF、PF フラグが結果にしたがってセットされる。

NEG—Two's Complement Negation（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。

NOP—No Operation

オペコード	命令	説明
90	NOP	操作なし

説明

何の操作も実行されない。この命令は、命令ストリーム内で空間を占めるが、EIP レジスタを除いて、マシン・コンテキストに影響を与えない1バイト命令である。

NOP 命令は、XCHG (E)AX、(E)AX 命令の別名ニーモニックである。

影響を受けるフラグ

なし。

例外（すべての操作モード）

なし。

NOT—One's Complement Negation

オペコード	命令	説明
F6 /2	NOT *r/m8*	*r/m8* の各ビットを反転する。
F7 /2	NOT *r/m16*	*r/m16* の各ビットを反転する。
F7 /2	NOT *r/m32*	*r/m32* の各ビットを反転する。

説明

デスティネーション・オペランドにビット単位の NOT（否定）演算（各 1 が 0 にセットされ、各 0 が 1 にセットされる）を実行し、結果をデスティネーション・オペランド・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、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 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 ← NOT DEST;

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を指している場合。 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。 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。

NOT—One's Complement Negation（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。

OR—Logical Inclusive OR

オペコード	命令	説明
0C *ib*	OR AL, *imm8*	AL と *imm8* との OR をとる。
0D *iw*	OR AX, *imm16*	AX と *imm16* との OR をとる。
0D *id*	OR EAX, *imm32*	EAX と *imm32* との OR をとる。
80 /1 *ib*	OR *r/m8*, *imm8*	*r/m8* と *imm8* との OR をとる。
81 /1 *iw*	OR *r/m16*, *imm16*	*r/m16* と *imm16* との OR をとる。
81 /1 *id*	OR *r/m32*, *imm32*	*r/m32* と *imm32* との OR をとる。
83 /1 *ib*	OR *r/m16*, *imm8*	*r/m16* と *imm8*（符号拡張）との OR をとる。
83 /1 *ib*	OR *r/m32*, *imm8*	*r/m32* と *imm8*（符号拡張）との OR をとる。
08 /*r*	OR *r/m8*, *r8*	*r/m8* と *r8* との OR をとる。
09 /*r*	OR *r/m16*, *r16*	*r/m16* と *r16* との OR をとる。
09 /*r*	OR *r/m32*, *r32*	*r/m32* と *r32* との OR をとる。
0A /*r*	OR *r8*, *r/m8*	*r8* と *r/m8* との OR をとる。
0B /*r*	OR *r16*, *r/m16*	*r16* と *r/m16* との OR をとる。
0B /*r*	OR *r32*, *r/m32*	*r32* と *r/m32* との OR をとる。

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）との間のビット単位の OR（論理和）演算を実行し、結果をデスティネーション・オペランド・ロケーションにストアする。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。）OR 命令の各ビットの結果は、第 1 オペランドと第 2 オペランドの対応するビットが両方とも 0 である場合は 0 にセットされ、そうでない場合は 1 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 OR 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。

OR—Logical Inclusive OR（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を指している場合。
	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。
	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。

ORPD—Bitwise Logical OR of Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 56 /r	ORPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の OR（論理和）演算を実行する。

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 OR（論理和）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] BitwiseOR SRC[127-0];

同等のインテル® C/C++ コンパイラ組み込み関数

ORPD　　__m128d _mm_or_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。

ORPD—Bitwise Logical OR 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。

ORPS—Bitwise Logical OR of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 56 /r	ORPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の OR（論理和）演算を実行する。

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間でビット単位の OR（論理和）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] BitwiseOR SRC[127-0];

同等のインテル® C/C++ コンパイラ組み込み関数

ORPS　　__m128 _mm_or_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。

ORPS—Bitwise Logical OR 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。

OUT—Output to Port

オペコード	命令	説明
E6 *ib*	OUT *imm8*, AL	AL にあるバイトを I/O ポートアドレス *imm8* に出力する。
E7 *ib*	OUT *imm8*, AX	AX にあるワードを I/O ポートアドレス *imm8* に出力する。
E7 *ib*	OUT *imm8*, EAX	EAX にあるダブルワードを I/O ポートアドレス *imm8* に出力する。
EE	OUT DX, AL	AL にあるバイトを DX にある I/O ポートアドレスに出力する。
EF	OUT DX, AX	AX にあるワードを DX にある I/O ポートアドレスに出力する。
EF	OUT DX, EAX	EAXにあるダブルワードをDXにあるI/Oポートアドレスに出力する。

説明

値を第2オペランド（ソース・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）で指定されたI/Oポートにコピーする。ソース・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。デスティネーション・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。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。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0～65,535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。

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。この場合は、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。

IA-32 アーキテクチャにおける互換性

OUT命令を実行した後、インテル® Pentium® プロセッサは、次の命令の実行を開始する前に、EWBE# ピンがアクティブにサンプリングされていることを保証する。（EWBE# がアクティブでない場合でも命令をプリフェッチすることはできるが、EWBE# ピンがアクティブにサンプリングされるまで命令は実行されないことに注意すること。）インテル Pentium プロセッサ・ファミリだけに EWBE# ピンがあり、他のIA-32プロセッサにはない。

OUT—Output to Port（続き）

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; (* Writes to selected I/O port *)
        FI;
    ELSE (Real Mode or Protected Mode with CPL ≤ IOPL *)
        DEST ← SRC; (* Writes to selected I/O port *)
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。

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。

OUTS/OUTSB/OUTSW/OUTSD—Output String to Port

オペコード	命令	説明
6E	OUTS DX, m8	バイトを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。
6F	OUTS DX, m16	ワードを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。
6F	OUTS DX, m32	ダブルワードを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。
6E	OUTSB	バイトを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。
6F	OUTSW	ワードを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。
6F	OUTSD	ダブルワードを DS:(E)SI に指定されたメモリ・ロケーションから DX に指定された I/O ポートに出力する。

説明

データをソース・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）で指定された I/O ポートにコピーする。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 レジスタまたは DS:DI レジスタから読み取られる。DS セグメントをセグメント・オーバライド・プリフィックスでオーバライドすることができる。デスティネーション・オペランドは、DX レジスタから読み取られる I/O ポートアドレス（0 ～ 65,535）である。アクセスされる I/O ポートのサイズ（すなわち、ソース・オペランドと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。（OUTS 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・オペランドは、I/O ポートのサイズとソースアドレス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、DX 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、OUTS 命令を実行する前に、このレジスタに正しくロードされていなければならない。

OUTS/OUTSB/OUTSW/OUTSD—Output String to Port（続き）

オペランドなし形式は、OUTS 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、DX 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、OUTSB（バイト）、OUTSW（ワード）、または OUTSD（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メモリ・ロケーションから I/O ポートに転送された後、(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、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。

OUTS、OUTSB、OUTSW、OUTSD 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。

IA-32 アーキテクチャにおける互換性

OUTS、OUTSB、OUTSW、または OUTSD 命令を実行した後、インテル® Pentium® プロセッサは、次の命令の実行を開始する前に、EWBE# ピンがアクティブにサンプリングされていることを保証する。（EWBE# がアクティブでない場合でも命令をプリフェッチすることはできるが、EWBE# ピンがアクティブにサンプリングされるまで命令は実行されないことに注意すること。）インテル Pentium プロセッサ・ファミリだけに EWBE# ピンがあり、他の IA-32 プロセッサにはない。インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、および P6 ファミリのプロセッサは、OUTS、OUTSB、OUTSW、または OUTSD 命令の実行時に、そのトランザクションのデータフェーズが完了してから次の命令を実行する。

OUTS/OUTSB/OUTSW/OUTSD—Output String to Port（続き）

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; (* Writes to I/O port *)
        FI;
    ELSE (Real Mode or Protected Mode with CPL ≤ IOPL *)
        DEST ← SRC; (* Writes to I/O port *)
FI;
IF (byte transfer)
    THEN IF DF = 0
        THEN (E)SI ← (E)SI + 1;
        ELSE (E)SI ← (E)SI – 1;
    FI;
    ELSE IF (word transfer)
        THEN IF DF = 0
            THEN (E)SI ← (E)SI + 2;
            ELSE (E)SI ← (E)SI – 2;
        FI;
        ELSE (* doubleword transfer *)
            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。

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。
	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・レジスタの内容がヌル・セグメント・セレクタの場合。
#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。

OUTS/OUTSB/OUTSW/OUTSD—Output String to Port（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)	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。

PACKSSWB/PACKSSDW—Pack with Signed Saturation

オペコード	命令	説明
0F 63 /r	PACKSSWB *mm1*, *mm2/m64*	符号付き飽和処理を使用して、*mm1* と *mm2/m64* の 4 個のパックド符号付きワード整数を、*mm1* の 8 個のパックド符号付きバイト整数に変換する。
66 0F 63 /r	PACKSSWB *xmm1*, *xmm2/m128*	符号付き飽和処理を使用して、*xmm1* と *xmm2/m128* の 8 個のパックド符号付きワード整数を、*xmm1* の 16 個のパックド符号付きバイト整数に変換する。
0F 6B /r	PACKSSDW *mm1*, *mm2/m64*	符号付き飽和処理を使用して、*mm1* と *mm2/m64* の 2 個のパックド符号付きダブルワード整数を、*mm1* の 4 個のパックド符号付きワード整数に変換する。
66 0F 6B /r	PACKSSDW *xmm1*, *xmm2/m128*	符号付き飽和処理を使用して、*xmm1* と *xmm2/m128* の 4 個のパックド符号付きダブルワード整数を、*xmm1* の 8 個のパックド符号付きワード整数に変換する。

説明

オーバーフロー条件を処理する飽和処理を使用して、パックド符号付きワード整数をパックド符号付きバイト整数に変換する（PACKSSWB の場合）か、あるいはパックド符号付きダブルワード整数をパックド符号付きワード整数に変換する（PACKSSDW の場合）。パック操作の例については、図 4-1. を参照のこと。

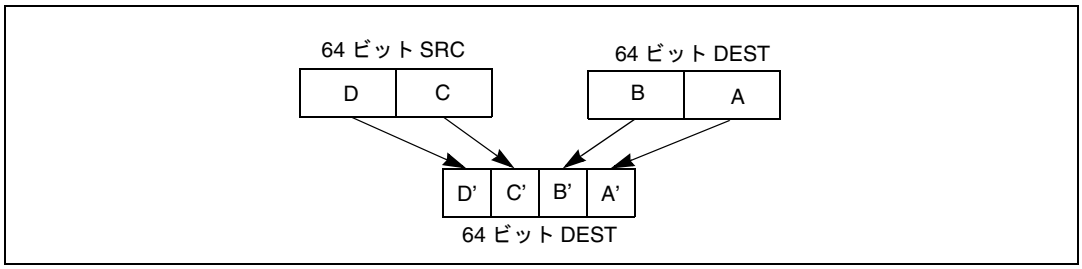


図 4-1. 64 ビット・オペランドを使用しての PACKSSDW 命令の操作

PACKSSWB 命令は、デスティネーション・オペランド（第 1 オペランド）の 4 個または 8 個の符号付きワード整数とソース・オペランド（第 2 オペランド）の 4 個または 8 個の符号付きワード整数を、8 個または 16 個の符号付きバイト整数に変換し、デスティネーション・オペランドにその結果をストアする。符号付きワード整数値が符号付きバイト整数の範囲を超える場合（すなわち、正の整数では 7FH より大きく、負の整数では 80H より大きい場合）は、飽和された符号付きバイト整数値である 7FH または 80H がデスティネーションにストアされる。

PACKSSWB/PACKSSDW—Pack with Signed Saturation（続き）

PACKSSDW 命令は、デスティネーション・オペランド（第 1 オペランド）の 2 個または 4 個の符号付きダブルワードとソース・オペランド（第 2 オペランド）の 2 個または 4 個の符号付きダブルワードを、デスティネーション・オペランドの 4 個または 8 個の符号付きワードにパックする（図 4-1. を参照）。符号付きダブルワード整数値が符号付きワードの範囲を超える場合（すなわち、正の整数では 7FFFH より大きく、負の整数では 8000H より大きい場合）は、飽和された符号付きワード整数値である 7FFFH または 8000H がデスティネーションにストアされる。

PACKSSWB 命令および PACKSSDW 命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

操作

```
PACKSSWB instruction with 64-bit operands
    DEST[7..0] ← SaturateSignedWordToSignedByte DEST[15..0];
    DEST[15..8] ← SaturateSignedWordToSignedByte DEST[31..16];
    DEST[23..16] ← SaturateSignedWordToSignedByte DEST[47..32];
    DEST[31..24] ← SaturateSignedWordToSignedByte DEST[63..48];
    DEST[39..32] ← SaturateSignedWordToSignedByte SRC[15..0];
    DEST[47..40] ← SaturateSignedWordToSignedByte SRC[31..16];
    DEST[55..48] ← SaturateSignedWordToSignedByte SRC[47..32];
    DEST[63..56] ← SaturateSignedWordToSignedByte SRC[63..48];

PACKSSDW instruction with 64-bit operands
    DEST[15..0] ← SaturateSignedDoublewordToSignedWord DEST[31..0];
    DEST[31..16] ← SaturateSignedDoublewordToSignedWord DEST[63..32];
    DEST[47..32] ← SaturateSignedDoublewordToSignedWord SRC[31..0];
    DEST[63..48] ← SaturateSignedDoublewordToSignedWord SRC[63..32];

PACKSSWB instruction with 128-bit operands
    DEST[7-0]   ← SaturateSignedWordToSignedByte (DEST[15-0]);
    DEST[15-8]  ← SaturateSignedWordToSignedByte (DEST[31-16]);
    DEST[23-16] ← SaturateSignedWordToSignedByte (DEST[47-32]);
    DEST[31-24] ← SaturateSignedWordToSignedByte (DEST[63-48]);
    DEST[39-32] ← SaturateSignedWordToSignedByte (DEST[79-64]);
    DEST[47-40] ← SaturateSignedWordToSignedByte (DEST[95-80]);
    DEST[55-48] ← SaturateSignedWordToSignedByte (DEST[111-96]);
    DEST[63-56] ← SaturateSignedWordToSignedByte (DEST[127-112]);
    DEST[71-64] ← SaturateSignedWordToSignedByte (SRC[15-0]);
    DEST[79-72] ← SaturateSignedWordToSignedByte (SRC[31-16]);
```

PACKSSWB/PACKSSDW—Pack with Signed Saturation（続き）

```
    DEST[87-80] ← SaturateSignedWordToSignedByte (SRC[47-32]);
    DEST[95-88] ← SaturateSignedWordToSignedByte (SRC[63-48]);
    DEST[103-96] ← SaturateSignedWordToSignedByte (SRC[79-64]);
    DEST[111-104] ← SaturateSignedWordToSignedByte (SRC[95-80]);
    DEST[119-112] ← SaturateSignedWordToSignedByte (SRC[111-96]);
    DEST[127-120] ← SaturateSignedWordToSignedByte (SRC[127-112]);

PACKSSDW instruction with 128-bit operands
    DEST[15-0] ← SaturateSignedDwordToSignedWord (DEST[31-0]);
    DEST[31-16] ← SaturateSignedDwordToSignedWord (DEST[63-32]);
    DEST[47-32] ← SaturateSignedDwordToSignedWord (DEST[95-64]);
    DEST[63-48] ← SaturateSignedDwordToSignedWord (DEST[127-96]);
    DEST[79-64] ← SaturateSignedDwordToSignedWord (SRC[31-0]);
    DEST[95-80] ← SaturateSignedDwordToSignedWord (SRC[63-32]);
    DEST[111-96] ← SaturateSignedDwordToSignedWord (SRC[95-64]);
    DEST[127-112] ← SaturateSignedDwordToSignedWord (SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m64 _mm_packs_pi16(__m64 m1, __m64 m2)
__m64 _mm_packs_pi32 (__m64 m1, __m64 m2)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PACKSSWB/PACKSSDW—Pack with Signed Saturation（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

仮想 8086 モード例外

実アドレスモード例外と同じ。

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	（64 ビット操作のみ）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。

PACKUSWB—Pack with Unsigned Saturation

オペコード	命令	説明
0F 67 /r	PACKUSWB *mm*, *mm/m64*	符号なし飽和処理を使用して、*mm* の 4 個の符号付きワード整数と *mm/m64* の 4 個の符号付きワード整数を、*mm* の 8 個の符号なしバイト整数に変換する。
66 0F 67 /r	PACKUSWB *xmm1*, *xmm2/m128*	符号なし飽和処理を使用して、*xmm1* の 8 個の符号付きワード整数と *xmm2/m128* の 8 個の符号付きワード整数を、*xmm1* の 16 個の符号なしバイト整数に変換する。

説明

デスティネーション・オペランド（第 1 オペランド）の 4 個または 8 個の符号付きワード整数とソース・オペランド（第 2 オペランド）の 4 個または 8 個の符号付きワード整数を、8 個または 16 個の符号なしバイト整数に変換し、デスティネーション・オペランドにその結果をストアする（パック操作の例については、図 4-1. を参照のこと）。符号付きワード整数値が符号なしバイト整数の範囲を超える場合（すなわち、FFH より大きいか、00H より小さい場合）は、飽和された符号なしバイト整数値である FFH または 00H がデスティネーションにストアされる。

PACKUSWB 命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

操作

```
PACKUSWB instruction with 64-bit operands:
    DEST[7..0] ← SaturateSignedWordToUnsignedByte DEST[15..0];
    DEST[15..8] ← SaturateSignedWordToUnsignedByte DEST[31..16];
    DEST[23..16] ← SaturateSignedWordToUnsignedByte DEST[47..32];
    DEST[31..24] ← SaturateSignedWordToUnsignedByte DEST[63..48];
    DEST[39..32] ← SaturateSignedWordToUnsignedByte SRC[15..0];
    DEST[47..40] ← SaturateSignedWordToUnsignedByte SRC[31..16];
    DEST[55..48] ← SaturateSignedWordToUnsignedByte SRC[47..32];
    DEST[63..56] ← SaturateSignedWordToUnsignedByte SRC[63..48];

PACKUSWB instruction with 128-bit operands:
    DEST[7-0]   ← SaturateSignedWordToUnsignedByte (DEST[15-0]);
    DEST[15-8]  ← SaturateSignedWordToUnsignedByte (DEST[31-16]);
    DEST[23-16] ← SaturateSignedWordToUnsignedByte (DEST[47-32]);
    DEST[31-24] ← SaturateSignedWordToUnsignedByte (DEST[63-48]);
    DEST[39-32] ← SaturateSignedWordToUnsignedByte (DEST[79-64]);
```

PACKUSWB—Pack with Unsigned Saturation（続き）

```
DEST[47-40] ← SaturateSignedWordToUnsignedByte (DEST[95-80]);
DEST[55-48] ← SaturateSignedWordToUnsignedByte (DEST[111-96]);
DEST[63-56] ← SaturateSignedWordToUnsignedByte (DEST[127-112]);
DEST[71-64] ← SaturateSignedWordToUnsignedByte (SRC[15-0]);
DEST[79-72] ← SaturateSignedWordToUnsignedByte (SRC[31-16]);
DEST[87-80] ← SaturateSignedWordToUnsignedByte (SRC[47-32]);
DEST[95-88] ← SaturateSignedWordToUnsignedByte (SRC[63-48]);
DEST[103-96]  ← SaturateSignedWordToUnsignedByte (SRC[79-64]);
DEST[111-104] ← SaturateSignedWordToUnsignedByte (SRC[95-80]);
DEST[119-112] ← SaturateSignedWordToUnsignedByte (SRC[111-96]);
DEST[127-120] ← SaturateSignedWordToUnsignedByte (SRC[127-112]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

__m64 _mm_packs_pu16(__m64 m1, __m64 m2)

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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PACKUSWB—Pack with Unsigned Saturation（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PADDB/PADDW/PADDD—Add Packed Integers

オペコード	命令	説明
0F FC /r	PADDB *mm*, *mm/m64*	*mm/m64* と *mm* のパックドバイト整数を加算する。
66 0F FC /r	PADDB *xmm1*, *mm2/m128*	*xmm2/m128* と *xmm1* のパックドバイト整数を加算する。
0F FD /r	PADDW *mm*, *mm/m64*	*mm/m64* と *mm* のパックドワード整数を加算する。
66 0F FD /r	PADDW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックドワード整数を加算する。
0F FE /r	PADDD *mm*, *mm/m64*	*mm/m64* と *mm* のパックド・ダブルワード整数を加算する。
66 0F FE /r	PADDD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド・ダブルワード整数を加算する。

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) のパックド整数の SIMD 加算を実行し、結果のパックド整数をデスティネーション・オペランドに格納する。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 9-4. を参照のこと。以降の段落で説明するように、オーバーフローはラップアラウンドを使用して処理される。

上記の命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

PADDB 命令は、パックドバイト整数に加算する。個別の結果が 8 ビットで表現するには大きすぎるとき（オーバーフロー)、結果はラップアラウンドされ、下位 8 ビットがデスティネーション・オペランドに書き込まれる。

PADDW 命令は、パックドワード整数に加算する。個別の結果が 16 ビットで表現するには大きすぎるとき（オーバーフロー)、結果はラップアラウンドされ、下位 16 ビットがデスティネーション・オペランドに書き込まれる。

PADDD 命令は、パックド・ダブルワード整数を加算する。個別の結果が 32 ビットで表現するには大きすぎるとき（オーバーフロー）、結果はラップアラウンドされ、下位 32 ビットがデスティネーション・オペランドに書き込まれる。

PADDB/PADDW/PADDD—Add Packed Integers（続き）

PADDB、PADDW、PADDD 命令は、符号なしまたは符号付き（2 の補数表記）のパックド整数を操作できることに注意すること。ただし、これらの命令は、オーバーフローやキャリーを示す EFLAGS レジスタ内のビットをセットしない。このため、検出されないオーバーフロー状態が発生しないように、操作される値の範囲をソフトウェアによって制御しなければならない。

操作

```
PADDB instruction with 64-bit operands:
    DEST[7..0] ← DEST[7..0] + SRC[7..0];
    * repeat add operation for 2nd through 7th byte *;
    DEST[63..56] ← DEST[63..56] + SRC[63..56];

PADDB instruction with 128-bit operands:
    DEST[7-0] ← DEST[7-0] + SRC[7-0];
    * repeat add operation for 2nd through 14th byte *;
    DEST[127-120] ← DEST[111-120] + SRC[127-120];

PADDW instruction with 64-bit operands:
    DEST[15..0] ← DEST[15..0] + SRC[15..0];
    * repeat add operation for 2nd and 3th word *;
    DEST[63..48] ← DEST[63..48] + SRC[63..48];

PADDW instruction with 128-bit operands:
    DEST[15-0]  ← DEST[15-0] + SRC[15-0];
    * repeat add operation for 2nd through 7th word *;
    DEST[127-112] ← DEST[127-112] + SRC[127-112];

PADDD instruction with 64-bit operands:
    DEST[31..0] ← DEST[31..0] + SRC[31..0];
    DEST[63..32] ← DEST[63..32] + SRC[63..32];

PADDD instruction with 128-bit operands:
    DEST[31-0]  ← DEST[31-0]  + SRC[31-0];
    * repeat add operation for 2nd and 3th doubleword *;
    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

PADDB	__m64 _mm_add_pi8(__m64 m1, __m64 m2)
PADDB	__m128i_mm_add_epi8 (__m128ia,__m128ib)
PADDW	__m64 _mm_addw_pi16(__m64 m1, __m64 m2)
PADDW	__m128i _mm_add_epi16 (__m128i a, __m128i b)
PADDD	__m64 _mm_add_pi32(__m64 m1, __m64 m2)
PADDD	__m128i _mm_add_epi32 (__m128i a, __m128i b)

影響を受けるフラグ

なし。

PADDB/PADDW/PADDD—Add Packed Integers（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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PADDQ—Add Packed Quadword Integers

オペコード	命令	説明
0F D4 /r	PADDQ *mm1*, *mm2/m64*	*mm2/m64* と *mm1* のクワッドワード整数を加算し、結果を *mm1* に格納する。
66 0F D4 /r	PADDQ *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド・クワッドワード整数を加算し、結果を *xmm1* に格納する。

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）を加算して、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、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションに格納される 1 つのクワッドワード整数か、XMM レジスタまたは 128 ビットのメモリ・ロケーションに格納される 2 つのパックド・クワッドワード整数である。デスティネーション・オペランドは、MMX テクノロジ・レジスタに格納される 1 つのクワッドワード整数か、XMM レジスタに格納される 2 つのパックド・クワッドワード整数である。パックド・クワッドワードのオペランドを使用する場合は、SIMD 加算が実行される。結果のクワッドワードが大きすぎて 64 ビットで表現できない場合は（オーバーフロー）、結果はラップアラウンドされ、下位 64 ビットがデスティネーション要素に書き込まれる（すなわち、キャリーは無視される）。

PADDQ 命令は、符号なし整数と符号付き整数（2 の補数記法）のどちらを操作することもできる。ただし、この命令は、オーバーフローやキャリーを示す EFLAGS レジスタ内のビットをセットしない。このため、検出されないオーバーフロー状態が発生しないように、操作される値の範囲をソフトウェアによって制御しなければならない。

操作

```
PADDQ instruction with 64-Bit operands:
    DEST[63-0] ¨ DEST[63-0] + SRC[63-0];

PADDQ instruction with 128-Bit operands:
    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

```
PADDQ       __m64 _mm_add_si64 (__m64 a, __m64 b)
PADDQ       __m128i _mm_add_epi64 ( __m128i a, __m128i b)
```

影響を受けるフラグ

なし。

PADDQ—Add Packed Quadword Integers（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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、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。

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、アライメントの合っていないメモリ参照を行った場合。

数値例外

なし。

PADDSB/PADDSW—Add Packed Signed Integers with Signed Saturation

オペコード	命令	説明
0F EC /r	PADDSB *mm*, *mm/m64*	*mm/m64* と *mm* のパックド符号付きバイト整数を加算し、結果を飽和処理する。
66 0F EC /r	PADDSB *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド符号付きバイト整数を加算し、結果を飽和処理する。
0F ED /r	PADDSW *mm*, *mm/m64*	*mm/m64* と *mm* のパックド符号付きワード整数を加算し、結果を飽和処理する。
66 0F ED /r	PADDSW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド符号付きワード整数を加算し、結果を飽和処理する。

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）のパックド符号付き整数の SIMD 加算を実行し、結果のパックド整数をデスティネーション・オペランドに格納する。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 9-4. を参照のこと。以降の段落で説明するように、オーバーフローは符号付き飽和処理を使用して処理される。

上記の命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

PADDSB 命令は、パックド符号付きバイト整数を加算する。個別のバイトの結果が符号付きバイト整数の範囲を超える場合（すなわち、7FH より大きいかまたは 80H より小さい場合）は、それぞれ 7FH または 80H の飽和された値がデスティネーション・オペランドに書き込まれる。

PADDSW 命令は、パックド符号付きワード整数を加算する。個別のワードの結果が符号付きワード整数の範囲を超える場合（すなわち、7FFFH より大きいかまたは 8000H より小さい場合）は、それぞれ 7FFFH または 8000H の飽和された値がデスティネーション・オペランドに書き込まれる。

PADDSB/PADDSW—Add Packed Signed Integers with Signed Saturation（続き）

操作

```
PADDSB instruction with 64-bit operands:
    DEST[7..0] ← SaturateToSignedByte(DEST[7..0] + SRC (7..0]) ;
    * repeat add operation for 2nd through 7th bytes *;
    DEST[63..56] ← SaturateToSignedByte(DEST[63..56] + SRC[63..56] );

PADDSB instruction with 128-bit operands:
    DEST[7-0] ← SaturateToSignedByte (DEST[7-0] + SRC[7-0]);
    * repeat add operation for 2nd through 14th bytes *;
    DEST[127-120] ← SaturateToSignedByte (DEST[111-120] + SRC[127-120]);

PADDSW instruction with 64-bit operands
    DEST[15..0] ← SaturateToSignedWord(DEST[15..0] + SRC[15..0] );
    * repeat add operation for 2nd and 7th words *;
    DEST[63..48] ← SaturateToSignedWord(DEST[63..48] + SRC[63..48] );

PADDSW instruction with 128-bit operands
    DEST[15-0]  ← SaturateToSignedWord (DEST[15-0] + SRC[15-0]);
    * repeat add operation for 2nd through 7th words *;
    DEST[127-112] ← SaturateToSignedWord (DEST[127-112] + SRC[127-112]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PADDSB	__m64 _mm_adds_pi8(__m64 m1, __m64 m2)
PADDSB	__m128i _mm_adds_epi8 (__m128i a, __m128i b)
PADDSW	__m64 _mm_adds_pi16(__m64 m1, __m64 m2)
PADDSW	__m128i _mm_adds_epi16 (__m128i a, __m128i b)

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。 （128 ビット操作のみ）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。
#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。

PADDSB/PADDSW—Add Packed Signed Integers with Signed Saturation（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PADDUSB/PADDUSW—Add Packed Unsigned Integers with Unsigned Saturation

オペコード	命令	説明
0F DC /r	PADDUSB *mm*, *mm/m64*	*mm/m64* と *mm* のパック ド符号なしバイト整数を加算し、結果を飽和処理する。
66 0F DC /r	PADDUSB *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド符号なしバイト整数を加算し、結果を飽和処理する。
0F DD /r	PADDUSW *mm*, *mm/m64*	*mm/m64* と *mm* のパック ド符号なしワード整数を加算し、結果を飽和処理する。
66 0F DD /r	PADDUSW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパック ド符号なしワード整数を加算し、結果を飽和処理する。

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）のパック ド符号なし整数の SIMD 加算を実行し、結果のパック ド整数をデスティネーション・オペランドに格納する。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 9-4. を参照のこと。以降の段落で説明するように、オーバーフローは符号なし飽和処理を使用して処理される。

上記の命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

PADDUSB 命令は、パック ド符号なしバイト整数を加算する。個別のバイトの結果が符号なしバイト整数の範囲を超える場合（すなわち、FFH より大きい場合）は、FFH の飽和された符号なし値がデスティネーション・オペランドに書き込まれる。

PADDUSW 命令は、パック ド符号なしワード整数を加算する。個別のワードの結果が符号なしワード整数の範囲を超える場合（すなわち、FFFFH より大きい場合）は、FFFFH の飽和された符号なし値がデスティネーション・オペランドに書き込まれる。

PADDUSB/PADDUSW—Add Packed Unsigned Integers with Unsigned Saturation（続き）

操作

```
PADDUSB instruction with 64-bit operands:
    DEST[7..0] ← SaturateToUnsignedByte(DEST[7..0] + SRC (7..0] );
    * repeat add operation for 2nd through 7th bytes *:
    DEST[63..56] ← SaturateToUnsignedByte(DEST[63..56] + SRC[63..56]

PADDUSB instruction with 128-bit operands:
    DEST[7-0] ← SaturateToUnsignedByte (DEST[7-0] + SRC[7-0]);
    * repeat add operation for 2nd through 14th bytes *:
    DEST[127-120] ← SaturateToUnSignedByte (DEST[127-120] + SRC[127-120]);

PADDUSW instruction with 64-bit operands:
    DEST[15..0] ← SaturateToUnsignedWord(DEST[15..0] + SRC[15..0] );
    * repeat add operation for 2nd and 3rd words *:
    DEST[63..48] ← SaturateToUnsignedWord(DEST[63..48] + SRC[63..48] );

PADDUSW instruction with 128-bit operands:
    DEST[15-0]  ← SaturateToUnsignedWord (DEST[15-0] + SRC[15-0]);
    * repeat add operation for 2nd through 7th words *:
    DEST[127-112] ← SaturateToUnSignedWord (DEST[127-112] + SRC[127-112]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PADDUSB	__m64 _mm_adds_pu8(__m64 m1, __m64 m2)
PADDUSW	__m64 _mm_adds_pu16(__m64 m1, __m64 m2)
PADDUSB	__m128i _mm_adds_epu8 (__m128i a, __m128i b)
PADDUSW	__m128i _mm_adds_epu16 (__m128i a, __m128i b)

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。 （128 ビット操作のみ）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。
#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。

PADDUSB/PADDUSW—Add Packed Unsigned Integers with Unsigned Saturation（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PAND—Logical AND

オペコード	命令	説明
0F DB /r	PAND *mm*, *mm/m64*	*mm/m64* と *mm* のビット単位の AND（論理積）演算を実行する。
66 0F DB /r	PAND *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）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、MMX® テクノロジ・レジスタか 64 ビット・メモリ・ロケーションを使用でき、または XMM レジスタか 128 ビット・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、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。各ビットの結果は、第 1 と第 2 オペランドの対応するビットが両方とも 1 である場合は 1 にセットされ、そうでない場合はゼロにセットされる。

操作

```
DEST ← DEST AND SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PAND        __m64 _mm_and_si64 (__m64 m1, __m64 m2)
PAND        __m128i _mm_and_si128 ( __m128i a, __m128i b)
```

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。 （128 ビット操作のみ）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。
#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。

PAND—Logical AND（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PANDN—Logical AND NOT

オペコード	命令	説明
0F DF /r	PANDN *mm*, *mm/m64*	*mm/m64* と *mm* のビット単位の AND NOT（否定論理積）演算を実行する。
66 0F DF /r	PANDN *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）のビット単位の NOT（否定）演算を実行した後、反転されたデスティネーション・オペランドとソース・オペランド（第 2 オペランド）の間でビット単位の AND（論理積）演算を実行する。結果はデスティネーション・オペランドに格納される。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・ロケーションか、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・レジスタまたは XMM レジスタである。結果の各ビットは、第 1 オペランド内の対応するビットが 0 で第 2 オペランド内の対応するビットが 1 の場合は 1 に設定され、それ以外の場合は 0 に設定される。

操作

```
DEST ← (NOT DEST) AND SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PANDN     __m64 _mm_andnot_si64 (__m64 m1, __m64 m2)
PANDN     __m128i _mm_andnot_si128 ( __m128i a, __m128i b)
```

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。
	（128 ビット操作のみ）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。
#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。

PANDN—Logical AND NOT（続き）

#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PAUSE—Spin Loop Hint

オペコード	命令	説明
F3 90	PAUSE	spin-wait ループのパフォーマンスを向上させるためのヒントをプロセッサに提供する。

説明

spin-wait ループのパフォーマンスを向上させる。「spin-wait ループ」を実行すると、インテル® Pentium® 4 プロセッサまたはインテル® Xeon™ プロセッサは起こり得るメモリ順序違反を検出したとき、パフォーマンスの面で厳しい制限を受ける。PAUSE 命令は、コード・シーケンスが spin-wait ループになっているプロセッサにヒントを提供する。プロセッサはこのヒントを使用して、大抵の状況におけるメモリ順序違反を回避する。これにより、プロセッサのパフォーマンスは大幅に向上する。こうした理由により、PAUSE 命令をすべての spin-wait ループに配置することが推奨される。

PAUSE 命令は、spin ループの実行時にインテル Pentium 4 プロセッサの消費電力を低減させる追加機能を備えている。インテル Pentium 4 プロセッサは spin-wait ループを極めて高速に実行できるため、リソースの待機中に電力を大量に消費する。spin-wait ループに PAUSE 命令を挿入することで、プロセッサの消費電力が大幅に低減される。

この命令はインテル Pentium 4 プロセッサで導入されたが、すべての IA-32 プロセッサに対して互換性がある。初期の IA-32 プロセッサにおいては、PAUSE 命令は NOP 命令と同じような動作をする。インテル Pentium 4 プロセッサとインテル Xeon プロセッサは、事前定義された待ち時間として PAUSE 命令を実装している。この待ち時間は有限であり、プロセッサによってはゼロである場合もある。この命令は、プロセッサのアーキテクチャ上の状態を変更することはない（つまり、この命令は、基本的には、ノー・オペレーションを実行することで次の命令の実行を遅らせる）。

操作

```
Execute_Next_Instruction(DELAY);
```

保護モード例外

なし。

実アドレスモード例外

なし。

仮想 8086 モード例外

なし。

PAUSE—Spin Loop Hint（続き）

数値例外

なし。

PAVGB/PAVGW—Average Packed Integers

オペコード	命令	説明
0F E0 /r	PAVGB *mm1*, *mm2/m64*	丸めを使用して、*mm2/m64* と *mm1* のパックド符号なしバイト整数の平均を求める。
66 0F E0, /r	PAVGB *xmm1*, *xmm2/m128*	丸めを使用して、*xmm2/m128* と *xmm1* のパックド符号なしバイト整数の平均を求める。
0F E3 /r	PAVGW *mm1*, *mm2/m64*	丸めを使用して、*mm2/m64* と *mm1* のパックド符号なしワード整数の平均を求める。
66 0F E3 /r	PAVGW *xmm1*, *xmm2/m128*	丸めを使用して、*xmm2/m128* と *xmm1* のパックド符号なしワード整数の平均を求める。

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）のパックド符号なし整数の SIMD 平均値計算を実行し、結果をデスティネーション・オペランドに格納する。第 1 オペランドと第 2 オペランドの対応するデータ要素の各ペアについて、データ要素同士を加算して、その和に 1 を加算し、結果を 1 ビット右にシフトする。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・ロケーションか、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・レジスタまたは XMM レジスタである。

PAVGB 命令は、符号なしパックドバイトを処理する。PAVGW 命令は、符号なしパックドワードを処理する。

操作

```
PAVGB instruction with 64-bit operands:
    SRC[7-0) ← (SRC[7-0) + DEST[7-0) + 1) >> 1; * temp sum before shifting is 9 bits *
    * repeat operation performed for bytes 2 through 6;
    SRC[63-56) ← (SRC[63-56) + DEST[63-56) + 1) >> 1;

PAVGW instruction with 64-bit operands:
    SRC[15-0) ← (SRC[15-0) + DEST[15-0) + 1) >> 1; * temp sum before shifting is 17 bits *
    * repeat operation performed for words 2 and 3;
    SRC[63-48) ← (SRC[63-48) + DEST[63-48) + 1) >> 1;

PAVGB instruction with 128-bit operands:
    SRC[7-0) ← (SRC[7-0) + DEST[7-0) + 1) >> 1; * temp sum before shifting is 9 bits *
    * repeat operation performed for bytes 2 through 14;
    SRC[63-56) ← (SRC[63-56) + DEST[63-56) + 1) >> 1;

PAVGW instruction with 128-bit operands:
    SRC[15-0) ← (SRC[15-0) + DEST[15-0) + 1) >> 1; * temp sum before shifting is 17 bits *
    * repeat operation performed for words 2 through 6;
    SRC[127-48) ← (SRC[127-112) + DEST[127-112) + 1) >> 1;
```

PAVGB/PAVGW—Average Packed Integers（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PAVGB	__m64_mm_avg_pu8 (__m64 a, __m64 b)
PAVGW	__m64_mm_avg_pu16 (__m64 a, __m64 b)
PAVGB	__m128i _mm_avg_epu8 (__m128i a, __m128i b)
PAVGW	__m128i _mm_avg_epu16 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

PAVGB/PAVGW—Average Packed Integers（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)	（64 ビット操作のみ）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。

数値例外

なし。

PCMPEQB/PCMPEQW/PCMPEQD—Compare Packed Data for Equal

オペコード	命令	説明
0F 74 /r	PCMPEQB *mm*, *mm/m64*	*mm/m64* と *mm* のパックドバイトを等しいか比較する。
66 0F 74 /r	PCMPEQB *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックドバイトを等しいか比較する。
0F 75 /r	PCMPEQW *mm*, *mm/m64*	*mm/m64* と *mm* のパックドワードを等しいか比較する。
66 0F 75 /r	PCMPEQW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックドワードを等しいか比較する。
0F 76 /r	PCMPEQD *mm*, *mm/m64*	*mm/m64* と *mm* のパックド・ダブルワードを等しいか比較する。
66 0F 76 /r	PCMPEQD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド・ダブルワードを等しいか比較する。

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）のパックドバイト / ワード / ダブルワードが等しいかどうかの SIMD 比較を実行する。データ要素のペアが等しい場合は、デスティネーション・オペランドの対応するデータ要素はすべて 1 に設定される。そうでない場合は、すべてゼロに設定される。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、XMM レジスタまたは 128 ビットのメモリ・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、MMX テクノロジ・レジスタまたは XMM レジスタを使用できる。

PCMPEQB 命令は、デスティネーション・オペランドおよびソース・オペランドの対応するバイトを比較する。PCMPEQW 命令は、デスティネーション・オペランドおよびソース・オペランドの対応するワードを比較する。PCMPEQD 命令は、デスティネーション・オペランドおよびソース・オペランドの対応するダブルワードを比較する。

操作

```
PCMPEQB instruction with 64-bit operands:
    IF DEST[7..0] = SRC[7..0]
        THEN DEST[7  0) ← FFH;
        ELSE DEST[7..0] ← 0;
    * Continue comparison of 2nd through 7th bytes in DEST and SRC *
    IF DEST[63..56] = SRC[63..56]
        THEN DEST[63..56] ← FFH;
        ELSE DEST[63..56] ← 0;

PCMPEQB instruction with 128-bit operands:
    IF DEST[7..0] = SRC[7..0]
        THEN DEST[7  0) ← FFH;
        ELSE DEST[7..0] ← 0;
```

PCMPEQB/PCMPEQW/PCMPEQD—Compare Packed Data for Equal（続き）

```
    * Continue comparison of 2nd through 15th bytes in DEST and SRC *
    IF DEST[63..56] = SRC[63..56]
        THEN DEST[63..56] ← FFH;
        ELSE DEST[63..56] ← 0;

PCMPEQW instruction with 64-bit operands:
    IF DEST[15..0] = SRC[15..0]
        THEN DEST[15..0] ← FFFFH;
        ELSE DEST[15..0] ← 0;
    * Continue comparison of 2nd and 3rd words in DEST and SRC *
    IF DEST[63..48] = SRC[63..48]
        THEN DEST[63..48] ← FFFFH;
        ELSE DEST[63..48] ← 0;

PCMPEQW instruction with 128-bit operands:
    IF DEST[15..0] = SRC[15..0]
        THEN DEST[15..0] ← FFFFH;
        ELSE DEST[15..0] ← 0;
    * Continue comparison of 2nd through 7th words in DEST and SRC *
    IF DEST[63..48] = SRC[63..48]
        THEN DEST[63..48] ← FFFFH;
        ELSE DEST[63..48] ← 0;

PCMPEQD instruction with 64-bit operands:
    IF DEST[31..0] = SRC[31..0]
        THEN DEST[31..0] ← FFFFFFFFH;
        ELSE DEST[31..0] ← 0;
    IF DEST[63..32] = SRC[63..32]
        THEN DEST[63..32] ← FFFFFFFFH;
        ELSE DEST[63..32] ← 0;

PCMPEQD instruction with 128-bit operands:
    IF DEST[31..0] = SRC[31..0]
        THEN DEST[31..0] ← FFFFFFFFH;
        ELSE DEST[31..0] ← 0;
    * Continue comparison of 2nd and 3rd doublewords in DEST and SRC *
    IF DEST[63..32] = SRC[63..32]
        THEN DEST[63..32] ← FFFFFFFFH;
        ELSE DEST[63..32] ← 0;
```

PCMPEQB/PCMPEQW/PCMPEQD—Compare Packed Data for Equal（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PCMPEQB	__m64 _mm_cmpeq_pi8 (__m64 m1, __m64 m2)
PCMPEQW	__m64 _mm_cmpeq_pi16 (__m64 m1, __m64 m2)
PCMPEQD	__m64 _mm_cmpeq_pi32 (__m64 m1, __m64 m2)
PCMPEQB	__m128i _mm_cmpeq_epi8 (__m128i a, __m128i b)
PCMPEQW	__m128i _mm_cmpeq_epi16 (__m128i a, __m128i b)
PCMPEQD	__m128i _mm_cmpeq_epi32 (__m128i a, __m128i b)

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。 （128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の X87 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)	（64 ビット操作のみ）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。

PCMPEQB/PCMPEQW/PCMPEQD—Compare Packed Data for Equal（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の X87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PCMPGTB/PCMPGTW/PCMPGTD—Compare Packed Signed Integers for Greater Than

オペコード	命令	説明
0F 64 /r	PCMPGTB *mm*, *mm/m64*	*mm* と *mm/m64* のパックド符号付きバイト整数を、「より大きい」の条件で比較する。
66 0F 64 /r	PCMPGTB *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパックド符号付きバイト整数を、「より大きい」の条件で比較する。
0F 65 /r	PCMPGTW *mm*, *mm/m64*	*mm* と *mm/m64* のパックド符号付きワード整数を、「より大きい」の条件で比較する。
66 0F 65 /r	PCMPGTW *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパックド符号付きワード整数を、「より大きい」の条件で比較する。
0F 66 /r	PCMPGTD *mm*, *mm/m64*	*mm* と *mm/m64* のパックド符号付きダブルワード整数を、「より大きい」の条件で比較する。
66 0F 66 /r	PCMPGTD *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパックド符号付きダブルワード整数を、「より大きい」の条件で比較する。

説明

デスティネーション・オペランド（第1オペランド）のパックドバイト整数/ワード整数/ダブルワード整数の値が、ソース・オペランド（第2オペランド）のパックドバイト整数/ワード整数/ダブルワード整数の値より大きいかどうかのSIMD符号付き比較を実行する。デスティネーション・オペランドのデータ要素がソース・オペランドの対応するデータ要素より大きい場合は、デスティネーション・オペランドの対応するデータ要素はすべて1に設定される。そうでない場合は、すべて0に設定される。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、XMMレジスタまたは128ビットのメモリ・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、MMXテクノロジ・レジスタまたはXMMレジスタを使用できる。

PCMPGTB命令は、デスティネーション・オペランドおよびソース・オペランドの対応する符号付きバイト整数を比較する。PCMPGTW命令は、デスティネーション・オペランドおよびソース・オペランドの対応する符号付きワード整数を比較する。PCMPGTD命令は、デスティネーション・オペランドおよびソース・オペランドの対応する符号付きダブルワード整数を比較する。

操作

```
PCMPGTB instruction with 64-bit operands:
    IF DEST[7..0] > SRC[7..0]
        THEN DEST[7  0) ← FFH;
        ELSE DEST[7..0] ← 0;
    * Continue comparison of 2nd through 7th bytes in DEST and SRC *
    IF DEST[63..56] > SRC[63..56]
        THEN DEST[63..56] ← FFH;
        ELSE DEST[63..56] ← 0;
```

PCMPGTB/PCMPGTW/PCMPGTD—Compare Packed Signed Integers for Greater Than（続き）

```
PCMPGTB instruction with 128-bit operands:
    IF DEST[7..0] > SRC[7..0]
        THEN DEST[7  0) ← FFH;
        ELSE DEST[7..0] ← 0;
    * Continue comparison of 2nd through 15th bytes in DEST and SRC *
    IF DEST[63..56] > SRC[63..56]
        THEN DEST[63..56] ← FFH;
        ELSE DEST[63..56] ← 0;

PCMPGTW instruction with 64-bit operands:
    IF DEST[15..0] > SRC[15..0]
        THEN DEST[15..0] ← FFFFH;
        ELSE DEST[15..0] ← 0;
    * Continue comparison of 2nd and 3rd words in DEST and SRC *
    IF DEST[63..48] > SRC[63..48]
        THEN DEST[63..48] ← FFFFH;
        ELSE DEST[63..48] ← 0;

PCMPGTW instruction with 128-bit operands:
    IF DEST[15..0] > SRC[15..0]
        THEN DEST[15..0] ← FFFFH;
        ELSE DEST[15..0] ← 0;
    * Continue comparison of 2nd through 7th words in DEST and SRC *
    IF DEST[63..48] > SRC[63..48]
        THEN DEST[63..48] ← FFFFH;
        ELSE DEST[63..48] ← 0;

PCMPGTD instruction with 64-bit operands:
    IF DEST[31..0] > SRC[31..0]
        THEN DEST[31..0] ← FFFFFFFFH;
        ELSE DEST[31..0] ← 0;
    IF DEST[63..32] > SRC[63..32]
        THEN DEST[63..32] ← FFFFFFFFH;
        ELSE DEST[63..32] ← 0;

PCMPGTD instruction with 128-bit operands:
    IF DEST[31..0] > SRC[31..0]
        THEN DEST[31..0] ← FFFFFFFFH;
        ELSE DEST[31..0] ← 0;
    * Continue comparison of 2nd and 3rd doublewords in DEST and SRC *
    IF DEST[63..32] > SRC[63..32]
        THEN DEST[63..32] ← FFFFFFFFH;
        ELSE DEST[63..32] ← 0;
```

PCMPGTB/PCMPGTW/PCMPGTD—Compare Packed Signed Integers for Greater Than（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PCMPGTB	__m64 _mm_cmpgt_pi8 (__m64 m1, __m64 m2)
PCMPGTW	__m64 _mm_pcmpgt_pi16 (__m64 m1, __m64 m2)
DCMPGTD	__m64 _mm_pcmpgt_pi32 (__m64 m1, __m64 m2)
PCMPGTB	__m128i _mm_cmpgt_epi8 (__m128i a, __m128i b)
PCMPGTW	__m128i _mm_cmpgt_epi16 (__m128i a, __m128i b)
DCMPGTD	__m128i _mm_cmpgt_epi32 (__m128i a, __m128i b)

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
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PCMPGTB/PCMPGTW/PCMPGTD—Compare Packed Signed Integers for Greater Than（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PEXTRW—Extract Word

オペコード	命令	説明
0F C5 /r ib	PEXTRW *r32*, *mm*, *imm8*	*imm8* によって指定されたワードを *mm* から抽出し、*r32* に移動する。
66 0F C5 /r ib	PEXTRW *r32*, *xmm*, *imm8*	*imm8* によって指定されたワードを *xmm* から抽出し、*r32* に移動する。

説明

カウント・オペランド（第 3 オペランド）で指定されたソース・オペランド（第 2 オペランド）内のワードを、デスティネーション・オペランド（第 1 オペランド）にコピーする。ソース・オペランドは、MMX® テクノロジ・レジスタまたは 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の下位ワードである。カウント・オペランドは 8 ビットの即値である。MMX テクノロジ・レジスタにワード・ロケーションを指定する場合、カウント・オペランドの下位 2 ビットによってそのロケーションを指定する。XMM レジスタの場合は、下位 3 ビットによってそのロケーションを指定する。デスティネーション・オペランドの上位ワードはクリアされる（すべて 0 に設定される）。

操作

```
PEXTRW instruction with 64-bit source operand:
    SEL ← COUNT AND 3H;
    TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
    r32[15-0] ← TEMP[15-0];
    r32[31-16] ← 0000H;

PEXTRW instruction with 128-bit source operand:
    SEL ← COUNT AND 7H;
    TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
    r32[15-0] ← TEMP[15-0];
    r32[31-16] ← 0000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

PEXTRW　　int_mm_extract_pi16 (__m64 a, int n)
PEXTRW　　int _mm_extract_epi16 (__m128i a, int imm)

影響を受けるフラグ

なし。

PEXTRW—Extract 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。
（#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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PINSRW—Insert Word

オペコード	命令	説明
0F C4 /r ib	PINSRW *mm*, *r32/m16*, *imm8*	*r32* または *m16* の下位ワードを、*imm8* で指定された *mm* 内のワード位置に挿入する。
66 0F C4 /r ib	PINSRW *xmm*, *r32/m16*, *imm8*	*r32* または *m16* の下位ワードを、*imm8* で指定された *xmm* 内のワード位置に移動する。

説明

ソース・オペランド（第2オペランド）から1ワードをコピーして、カウント・オペランド（第3オペランド）で指定されたデスティネーション・オペランド（第1オペランド）内の位置に挿入する（デスティネーション・レジスタのその他のワードは変更されない）。ソース・オペランドは、汎用レジスタまたは16ビットのメモリ・ロケーションである（ソース・オペランドが汎用レジスタの場合は、レジスタの下位ワードがコピーされる）。デスティネーション・オペランドは、MMX® テクノロジ・レジスタまたはXMMレジスタである。カウント・オペランドは8ビットの即値である。MMXテクノロジ・レジスタにワード・ロケーションを指定する場合、カウント・オペランドの下位2ビットによってそのロケーションを指定する。XMMレジスタの場合は、下位3ビットによってそのロケーションを指定する。

操作

```
PINSRW instruction with 64-bit source operand:
    SEL ← COUNT AND 3H;
        CASE (determine word position) OF
            SEL ← 0:    MASK ← 000000000000FFFFH;
            SEL ← 1:    MASK ← 00000000FFFF0000H;
            SEL ← 2:    MASK ← 0000FFFF00000000H;
            SEL ← 3:    MASK ← FFFF000000000000H;
    DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL * 16)) AND MASK);

PINSRW instruction with 128-bit source operand:
    SEL ← COUNT AND 7H;
        CASE (determine word position) OF
            SEL ← 0:    MASK ← 00000000000000000000000000FFFFH;
            SEL ← 1:    MASK ← 00000000000000000000000FFFF0000H;
            SEL ← 2:    MASK ← 000000000000000000000FFFF00000000H;
            SEL ← 3:    MASK ← 0000000000000000FFFF000000000000H;
            SEL ← 4:    MASK ← 00000000000FFFF0000000000000000H;
            SEL ← 5:    MASK ← 000000000FFFF00000000000000000000H;
            SEL ← 6:    MASK ← 0000FFFF0000000000000000000000000H;
            SEL ← 7:    MASK ← FFFF00000000000000000000000000000H;
    DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL * 16)) AND MASK);
```

PINSRW—Insert Word（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PINSRW　__m64 _mm_insert_pi16 (__m64 a, int d, int n)
PINSRW　__m128i _mm_insert_epi16 (__m128i a, int b, int imm)

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。
#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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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	（64 ビット操作のみ）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(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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PINSRW—Insert Word（続き）

数値例外

なし。

PMADDWD—Multiply and Add Packed Integers

オペコード	命令	説明
0F F5 /r	PMADDWD *mm*, *mm/m64*	*mm* のパックドワードに *mm/m64* のパックドワードを掛ける。結果の隣接するダブルワードを加算して *mm* にストアする。
66 0F F5 /r	PMADDWD *xmm1*, *xmm2/m128*	*xmm1* のパックドワード整数に *xmm2/m128* のパックドワード整数を掛けて、得られた隣接するダブルワードを加算して *xmm1* にストアする。

説明

デスティネーション・オペランド（第 1 オペランド）の個別のパックド符号付きワードにソース・オペランド（第 2 オペランド）の対応する符号付きワードを掛け、仮の符号付きダブルワードの結果を生成する。次に、得られた隣接するダブルワードを合計して、デスティネーション・オペランドに格納する。例えば、ソース・オペランドとデスティネーション・オペランド内の対応する下位ワード（15～0）および（31～16）同士がそれぞれ乗算されて、得られたダブルワードが加算され、デスティネーション・レジスタの下位ダブルワード（31～0）に格納される。その他の隣接するワードについても、同じ操作が実行される（図 4-2. は、64 ビット・オペランドを使用する場合の操作を示している）。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・ロケーションか、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・レジスタまたは XMM レジスタである。

PMADD 命令は、1 つのグループ内の操作される 2 対のワードがすべて 8000H である場合にのみ、ラップアラウンドを使用する。この場合、結果は 80000000H にラップアラウンドされる。

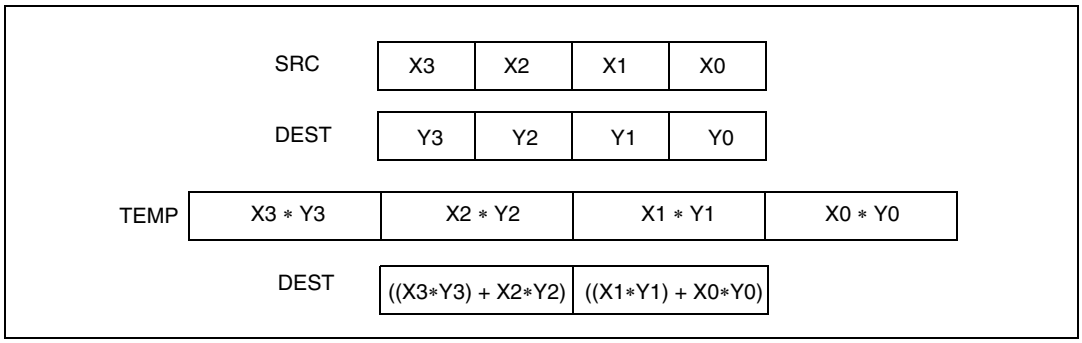


図 4-2. 64 ビット・オペランドを使用した PMADDWD 実行モデル

PMADDWD—Multiply and Add Packed Integers（続き）

操作

```
PMADDWD instruction with 64-bit operands:
    DEST[31..0] ← (DEST[15..0] * SRC[15..0]) + (DEST[31..16] * SRC[31..16]);
    DEST[63..32] ← (DEST[47..32] * SRC[47..32]) + (DEST[63..48] * SRC[63..48]);

PMADDWD instruction with 128-bit operands:
    DEST[31..0] ← (DEST[15..0] * SRC[15..0]) + (DEST[31..16] * SRC[31..16]);
    DEST[63..32] ← (DEST[47..32] * SRC[47..32]) + (DEST[63..48] * SRC[63..48]);
    DEST[95..64) ← (DEST[79..64) * SRC[79..64)) + (DEST[95..80) * SRC[95..80));
    DEST[127..96) ← (DEST[111..96) * SRC[111..96)) + (DEST[127..112) *
        SRC[127..112));
```

同等のインテル® C/C++ コンパイラ組み込み関数

PMADDWD __m64 _mm_madd_pi16(__m64 m1, __m64 m2)
PMADDWD __m128i _mm_madd_epi16 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMADDWD—Multiply and Add Packed Integers（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PMAXSW—Maximum of Packed Signed Word Integers

オペコード	命令	説明
0F EE /r	PMAXSW *mm1*, *mm2/m64*	*mm2/m64* と *mm1* の符号付きワード整数を比較して最大値を返す。
66 0F EE /r	PMAXSW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* の符号付きワード整数を比較して最大値を返す。

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 ド符号付きワード整数の SIMD 比較を実行し、それぞれのワード整数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、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションか、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・レジスタまたは XMM レジスタである。

操作

```
PMAXSW instruction for 64-bit operands:
    IF DEST[15-0] > SRC[15-0]) THEN
        (DEST[15-0] ← DEST[15-0];
    ELSE
        (DEST[15-0] ← SRC[15-0];
    FI
    * repeat operation for 2nd and 3rd words in source and destination operands *
    IF DEST[63-48] > SRC[63-48]) THEN
        (DEST[63-48] ← DEST[63-48];
    ELSE
        (DEST[63-48] ← SRC[63-48];
    FI
PMAXSW instruction for 128-bit operands:
    IF DEST[15-0] > SRC[15-0]) THEN
        (DEST[15-0] ← DEST[15-0];
    ELSE
        (DEST[15-0] ← SRC[15-0];
    FI
    * repeat operation for 2nd through 7th words in source and destination operands *
    IF DEST[127-112] > SRC[127-112]) THEN
        (DEST[127-112] ← DEST[127-112];
    ELSE
        (DEST[127-112] ← SRC[127-112];
    FI
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PMAXSW      __m64 _mm_max_pi16(__m64 a, __m64 b)
PMAXSW      __m128i _mm_max_epi16 ( __m128i a, __m128i b)
```

PMAXSW—Maximum of Packed Signed Word Integers（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMAXSW—Maximum of Packed Signed Word Integers（続き）

数値例外

なし。

PMAXUB—Maximum of Packed Unsigned Byte Integers

オペコード	命令	説明
0F DE /r	PMAXUB *mm1*, *mm2/m64*	*mm2/m64* と *mm1* の符号なしバイト整数を比較して最大値を返す。
66 0F DE /r	PMAXUB *xmm1*, *xmm2/m128*	*xmm2/m128*と*xmm1*の符号なしバイト整数を比較して最大値を返す。

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）のパックド符号なしバイト整数の SIMD 比較を実行し、それぞれのバイト整数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、MMX®テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションか、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・レジスタまたは XMM レジスタである。

操作

```
PMAXUB instruction for 64-bit operands:
    IF DEST[7-0] > SRC[17-0]) THEN
        (DEST[7-0] ← DEST[7-0];
    ELSE
        (DEST[7-0] ← SRC[7-0];
    FI
    * repeat operation for 2nd through 7th bytes in source and destination operands *
    IF DEST[63-56] > SRC[63-56]) THEN
        (DEST[63-56] ← DEST[63-56];
    ELSE
        (DEST[63-56] ← SRC[63-56];
    FI

PMAXUB instruction for 128-bit operands:
    IF DEST[7-0] > SRC[17-0]) THEN
        (DEST[7-0] ← DEST[7-0];
    ELSE
        (DEST[7-0] ← SRC[7-0];
    FI
    * repeat operation for 2nd through 15th bytes in source and destination operands *
    IF DEST[127-120] > SRC[127-120]) THEN
        (DEST[127-120] ← DEST[127-120];
    ELSE
        (DEST[127-120] ← SRC[127-120];
    FI
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PMAXUB      __m64 _mm_max_pu8(__m64 a, __m64 b)
PMAXUB      __m128i _mm_max_epu8 ( __m128i a, __m128i b)
```

PMAXUB—Maximum of Packed Unsigned Byte Integers（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMAXUB—Maximum of Packed Unsigned Byte Integers（続き）

数値例外

なし。

PMINSW—Minimum of Packed Signed Word Integers

オペコード	命令	説明
0F EA /r	PMINSW *mm1*, *mm2/m64*	*mm2/m64* と *mm1* の符号付きワード整数を比較して最小値を返す。
66 0F EA /r	PMINSW *xmm1*, *xmm2/m128*	*xmm2/m128*と*xmm1*の符号付きワード整数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）のパック ド符号付きワード整数の SIMD 比較を実行し、それぞれのワード整数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、MMX®テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションか、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・レジスタまたは XMM レジスタである。

操作

```
PMINSW instruction for 64-bit operands:
    IF DEST[15-0] < SRC[15-0]) THEN
        (DEST[15-0] ← DEST[15-0];
    ELSE
        (DEST[15-0] ← SRC[15-0];
    FI
    * repeat operation for 2nd and 3rd words in source and destination operands *
    IF DEST[63-48] < SRC[63-48]) THEN
        (DEST[63-48] ← DEST[63-48];
    ELSE
        (DEST[63-48] ← SRC[63-48];
    FI

MINSW instruction for 128-bit operands:
    IF DEST[15-0] < SRC[15-0]) THEN
        (DEST[15-0] ← DEST[15-0];
    ELSE
        (DEST[15-0] ← SRC[15-0];
    FI
    * repeat operation for 2nd through 7th words in source and destination operands *
    IF DEST[127-112] < SRC/m64[127-112]) THEN
        (DEST[127-112] ← DEST[127-112];
    ELSE
        (DEST[127-112] ← SRC[127-112];
    FI
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PMINSW      __m64 _mm_min_pi16 (__m64 a, __m64 b)
PMINSW      __m128i _mm_min_epi16 ( __m128i a, __m128i b)
```

PMINSW—Minimum of Packed Signed Word Integers（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMINSW—Minimum of Packed Signed Word Integers（続き）

数値例外

なし。

PMINUB—Minimum of Packed Unsigned Byte Integers

オペコード	命令	説明
0F DA /r	PMINUB *mm1*, *mm2/m64*	*mm2/m64* と *mm1* の符号なしバイト整数を比較して最小値を返す。
66 0F DA /r	PMINUB *xmm1*, *xmm2/m128*	*xmm2/m128*と*xmm1*の符号なしバイト整数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）のパックド符号なしバイト整数の SIMD 比較を実行し、それぞれのバイト整数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、MMX®テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションか、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・レジスタまたは XMM レジスタである。

操作

```
PMINUB instruction for 64-bit operands:
    IF DEST[7-0] < SRC[17-0]) THEN
        (DEST[7-0] ← DEST[7-0];
    ELSE
        (DEST[7-0] ← SRC[7-0];
    FI
    * repeat operation for 2nd through 7th bytes in source and destination operands *
    IF DEST[63-56] < SRC[63-56]) THEN
        (DEST[63-56] ← DEST[63-56];
    ELSE
        (DEST[63-56] ← SRC[63-56];
    FI

PMINUB instruction for 128-bit operands:
    IF DEST[7-0] < SRC[17-0]) THEN
        (DEST[7-0] ← DEST[7-0];
    ELSE
        (DEST[7-0] ← SRC[7-0];
    FI
    * repeat operation for 2nd through 15th bytes in source and destination operands *
    IF DEST[127-120] < SRC[127-120]) THEN
        (DEST[127-120] ← DEST[127-120];
    ELSE
        (DEST[127-120] ← SRC[127-120];
    FI
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PMINUB          __m64 _m_min_pu8 (__m64 a, __m64 b)
PMINUB          ___m128i _mm_min_epu8 ( __m128i a, __m128i b)
```

PMINUB—Minimum of Packed Unsigned Byte Integers（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMINUB—Minimum of Packed Unsigned Byte Integers（続き）

数値例外

なし。

PMOVMSKB—Move Byte Mask

オペコード	命令	説明
0F D7 /r	PMOVMSKB *r32*, *mm*	*mm* のバイトマスクを *r32* に移動する。
66 0F D7 /r	PMOVMSKB *r32*, *xmm*	*xmm* のバイトマスクを *r32* に移動する。

説明

ソース・オペランド（第 2 オペランド）の各バイトの最上位ビットからマスクを作成し、結果をデスティネーション・オペランド（第 1 オペランド）の最下位バイトまたは下位ワードに格納する。ソース・オペランドは、MMX® テクノロジ・レジスタまたは 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。64 ビットのオペランドを操作する場合、バイトマスクは 8 ビットである。128 ビットのオペランドを操作する場合、バイトマスクは 16 ビットである。

操作

```
PMOVMSKB instruction with 64-bit source operand:
r32[0] ← SRC[7];
r32[1] ← SRC[15];
* repeat operation for bytes 2 through 6;
r32[7] ← SRC[63];
r32[31-8] ← 000000H;

PMOVMSKB instruction with 128-bit source operand:
r32[0] ← SRC[7];
r32[1] ← SRC[15];
* repeat operation for bytes 2 through 14;
r32[15] ← SRC[127];
r32[31-16] ← 0000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

PMOVMSKB　int_mm_movemask_pi8(__m64 a)
PMOVMSKB　int _mm_movemask_epi8 (__m128i a)

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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

PMOVMSKB—Move Byte Mask（続き）

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。

数値例外

なし。

PMULHUW—Multiply Packed Unsigned Integers and Store High Result

オペコード	命令	説明
0F E4 /r	PMULHUW *mm1*, *mm2/m64*	*mm1* レジスタと *mm2/m64* のパックド符号なしワード整数を乗算し、結果の上位 16 ビットを *mm1* に格納する。
66 0F E4 /r	PMULHUW *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパックド符号なしワード整数を乗算し、結果の上位 16 ビットを *xmm1* に格納する。

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）のパックド符号なしワード整数の SIMD 符号なし乗算を実行し、それぞれの 32 ビットの中間結果の上位 16 ビットをデスティネーション・オペランドに格納する（図 4-3. は、64 ビット・オペランドを使用する場合の操作を示している）。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・ロケーションか、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・レジスタまたは XMM レジスタである。

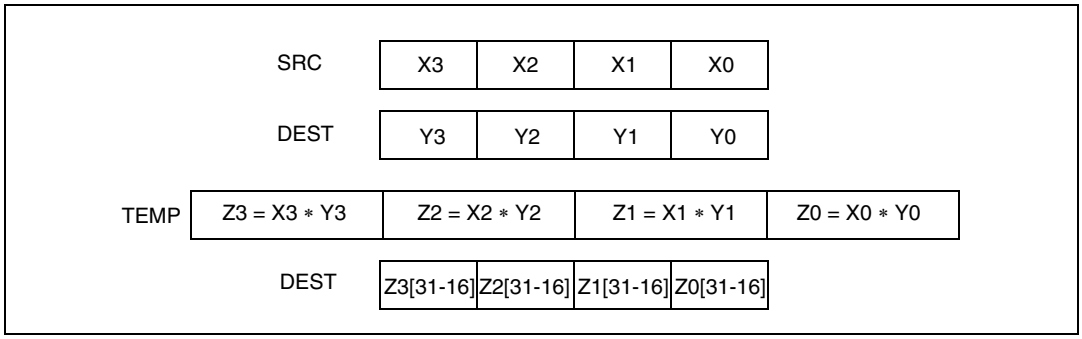


図 4-3. 64 ビット・オペランドを使用した PMULHUW 命令および PMULHW 命令の動作

操作

```
PMULHUW instruction with 64-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Unsigned multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    DEST[15-0] ←  TEMP0[31-16];
    DEST[31-16] ← TEMP1[31-16];
    DEST[47-32] ← TEMP2[31-16];
    DEST[63-48] ← TEMP3[31-16];
```

PMULHUW—Multiply Packed Unsigned Integers and Store High Result（続き）

```
PMULHUW instruction with 128-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Unsigned multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    TEMP4[31-0] ← DEST[79-64] * SRC[79-64];
    TEMP5[31-0] ← DEST[95-80] * SRC[95-80];
    TEMP6[31-0] ← DEST[111-96] * SRC[111-96];
    TEMP7[31-0] ← DEST[127-112] * SRC[127-112];
    DEST[15-0] ←   TEMP0[31-16];
    DEST[31-16] ←  TEMP1[31-16];
    DEST[47-32] ←  TEMP2[31-16];
    DEST[63-48] ←  TEMP3[31-16];
    DEST[79-64] ←  TEMP4[31-16];
    DEST[95-80] ←  TEMP5[31-16];
    DEST[111-96] ← TEMP6[31-16];
    DEST[127-112] ← TEMP7[31-16];
```

同等のインテル® C/C++ コンパイラ組み込み関数

PMULHUW　　__m64 _mm_mulhi_pu16(__m64 a, __m64 b)
PMULHUW　　__m128i _mm_mulhi_epu16 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PMULHUW—Multiply Packed Unsigned Integers and Store High Result（続き）

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PMULHW—Multiply Packed Signed Integers and Store High Result

オペコード	命令	説明
0F E5 /r	PMULHW *mm*, *mm/m64*	*mm1* レジスタと *mm2/m64* のパックド符号付きワード整数を乗算し、結果の上位 16 ビットを *mm1* に格納する。
66 0F E5 /r	PMULHW *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパックド符号付きワード整数を乗算し、結果の上位 16 ビットを *xmm1* に格納する。

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）のパックド符号付きワード整数の SIMD 符号付き乗算を実行し、それぞれの 32 ビットの中間結果の上位 16 ビットをデスティネーション・オペランドに格納する（図 4-3. は、64 ビット・オペランドを使用する場合の操作を示している）。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・ロケーションか、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・レジスタまたは XMM レジスタである。

操作

```
PMULHW instruction with 64-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Signed multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    DEST[15-0] ←  TEMP0[31-16];
    DEST[31-16] ← TEMP1[31-16];
    DEST[47-32] ← TEMP2[31-16];
    DEST[63-48] ← TEMP3[31-16];

PMULHW instruction with 128-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Signed multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    TEMP4[31-0] ← DEST[79-64] * SRC[79-64];
    TEMP5[31-0] ← DEST[95-80] * SRC[95-80];
    TEMP6[31-0] ← DEST[111-96] * SRC[111-96];
    TEMP7[31-0] ← DEST[127-112] * SRC[127-112];
    DEST[15-0] ←  TEMP0[31-16];
    DEST[31-16] ← TEMP1[31-16];
    DEST[47-32] ← TEMP2[31-16];
    DEST[63-48] ← TEMP3[31-16];
    DEST[79-64] ← TEMP4[31-16];
    DEST[95-80] ← TEMP5[31-16];
    DEST[111-96] ← TEMP6[31-16];
    DEST[127-112] ← TEMP7[31-16];
```

PMULHW—Multiply Packed Signed Integers and Store High Result（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PMULHW __m64 _mm_mulhi_pi16 (__m64 m1, __m64 m2)
PMULHW __m128i _mm_mulhi_epi16 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。

PMULHW—Multiply Packed Signed Integers and Store High Result（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PMULLW—Multiply Packed Signed Integers and Store Low Result

オペコード	命令	説明
0F D5 /r	PMULLW *mm*, *mm/m64*	*mm1* レジスタと *mm2/m64* のパック ド符号付きワード整数を乗算し、結果の下位 16 ビットを *mm1* に格納する。
66 0F D5 /r	PMULLW *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* のパック ド符号付きワード整数を乗算し、結果の下位 16 ビットを *xmm1* に格納する。

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）のパック ド符号付きワード整数の SIMD 符号付き乗算を実行し、それぞれの 32 ビットの中間結果の下位 16 ビットをデスティネーション・オペランドに格納する（図 4-4. は、64 ビット・オペランドを使用する場合の操作を示している）。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・ロケーションか、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・レジスタまたは XMM レジスタである。

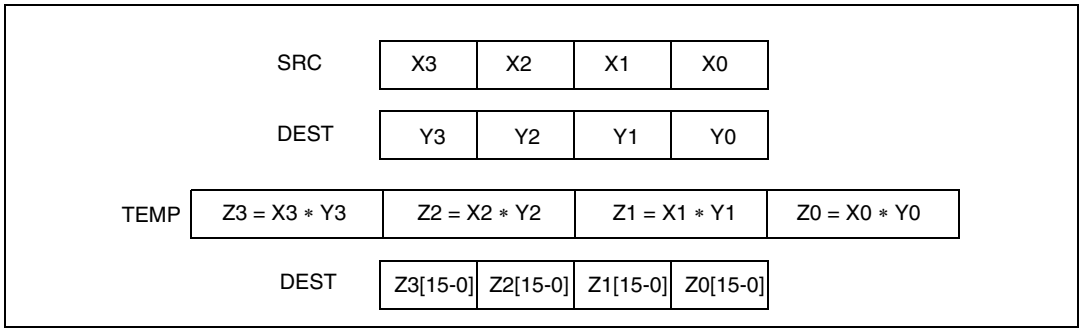


図 4-4. 64 ビット・オペランドを使用した PMULLW 命令の動作

操作

```
PMULLW instruction with 64-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Signed multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    DEST[15-0] ←  TEMP0[15-0];
    DEST[31-16] ← TEMP1[15-0];
    DEST[47-32] ← TEMP2[15-0];
    DEST[63-48] ← TEMP3[15-0];
```

PMULLW—Multiply Packed Signed Integers and Store Low Result（続き）

```
PMULLW instruction with 64-bit operands:
    TEMP0[31-0] ← DEST[15-0] * SRC[15-0]; * Signed multiplication *
    TEMP1[31-0] ← DEST[31-16] * SRC[31-16];
    TEMP2[31-0] ← DEST[47-32] * SRC[47-32];
    TEMP3[31-0] ← DEST[63-48] * SRC[63-48];
    TEMP4[31-0] ← DEST[79-64] * SRC[79-64];
    TEMP5[31-0] ← DEST[95-80] * SRC[95-80];
    TEMP6[31-0] ← DEST[111-96] * SRC[111-96];
    TEMP7[31-0] ← DEST[127-112] * SRC[127-112];
    DEST[15-0] ←  TEMP0[15-0];
    DEST[31-16] ← TEMP1[15-0];
    DEST[47-32] ← TEMP2[15-0];
    DEST[63-48] ← TEMP3[15-0];
    DEST[79-64] ← TEMP4[15-0];
    DEST[95-80] ← TEMP5[15-0];
    DEST[111-96] ← TEMP6[15-0];
    DEST[127-112] ← TEMP7[15-0];
```

同等のインテル® C/C++ コンパイラ組み込み関数

PMULLW　__m64 _mm_mullo_pi16(__m64 m1, __m64 m2)
PMULLW　__m128i _mm_mullo_epi16 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 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。

PMULLW—Multiply Packed Signed Integers and Store Low Result（続き）

#AC(0)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PMULUDQ—Multiply Packed Unsigned Doubleword Integers

オペコード	命令	説明
0F F4 /r	PMULUDQ *mm1*, *mm2/m64*	*mm1* の符号なしダブルワード整数に *mm2/m64* の符号なしダブルワード整数を掛けて、結果のクワッドワードを *mm1* に格納する。
66 OF F4 /r	PMULUDQ *xmm1*, *xmm2/m128*	*xmm1* のパック ド符号なしダブルワード整数に *xmm2/m128* のパック ド符号なしダブルワード整数を掛けて、結果のクワッドワードを *xmm1* に格納する。

説明

第 1 オペランド（デスティネーション・オペランド）に第 2 オペランド（ソース・オペランド）を掛けて、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、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションの下位ダブルワードに格納される 1 つの符号なしダブルワード整数か、XMM レジスタまたは 128 ビットのメモリ・ロケーションの第 1（最下位）ダブルワードと第 3 ダブルワードに格納される 2 つのパック ド符号なしダブルワード整数である。デスティネーション・オペランドは、MMX テクノロジ・レジスタの下位ダブルワードに格納される 1 つの符号なしダブルワード整数か、XMM レジスタの第 1 ダブルワードと第 3 ダブルワードに格納される 2 つのパック ド・ダブルワード整数である。得られる結果は、デスティネーションである MMX テクノロジ・レジスタにストアされた符号なしクワッドワード整数か、XMM レジスタにストアされた 2 つのパック ド符号なしクワッドワード整数である。結果のクワッドワードが大きすぎて 64 ビットで表現できない場合は（オーバーフロー）、結果はラップアラウンドされ、下位 64 ビットがデスティネーション要素に書き込まれる（すなわち、キャリーは無視される）。

64 ビット・メモリ・オペランドの場合は 64 ビットがメモリからフェッチされるが、下位のダブルワードしか計算に使用されない。128 ビット・メモリ・オペランドの場合は 128 ビットがメモリからフェッチされるが、第 1 および第 3 のダブルワードしか計算に使用されない。

操作

```
PMULUDQ instruction with 64-Bit operands:
    DEST[63-0] ← DEST[31-0] * SRC[31-0];

PMULUDQ instruction with 128-Bit operands:
    DEST[63-0] ← DEST[31-0] * SRC[31-0];
    DEST[127-64] ← DEST[95-64] * SRC[95-64];
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PMULUDQ      __m64 _mm_mul_su32 (__m64 a, __m64 b)
PMULUDQ      __m128i _mm_mul_epu32 ( __m128i a, __m128i b)
```

PMULUDQ—Multiply Packed Unsigned Doubleword Integers（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、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。

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、アライメントの合っていないメモリ参照を行った場合。

POP—Pop a Value from the Stack

オペコード	命令	説明
8F /0	POP *m16*	スタックのトップを *m16* にポップし、スタックポインタをインクリメントする。
8F /0	POP *m32*	スタックのトップを *m32* にポップし、スタックポインタをインクリメントする。
58+ *rw*	POP *r16*	スタックのトップを *r16* にポップし、スタックポインタをインクリメントする。
58+ *rd*	POP *r32*	スタックのトップを *r32* にポップし、スタックポインタをインクリメントする。
1F	POP DS	スタックのトップを DS にポップし、スタックポインタをインクリメントする。
07	POP ES	スタックのトップを ES にポップし、スタックポインタをインクリメントする。
17	POP SS	スタックのトップを SS にポップし、スタックポインタをインクリメントする。
0F A1	POP FS	スタックのトップを FS にポップし、スタックポインタをインクリメントする。
0F A9	POP GS	スタックのトップを GS にポップし、スタックポインタをインクリメントする。

説明

値をスタックのトップから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ロードし、スタックポインタをインクリメントする。デスティネーション・オペランドには、汎用レジスタ、メモリ・ロケーション、またはセグメント・レジスタを使用できる。

スタック・セグメントのアドレス・サイズ属性によって、スタックポインタのサイズ（16 ビットまたは 32 ビットーソース・アドレスのサイズ）が決まり、現在のコード・セグメントのオペランド・サイズ属性によって、スタックポインタをインクリメントする量（2 バイトまたは 4 バイト）が決まる。例えば、これらのアドレス・サイズ属性およびオペランド・サイズ属性が 32 である場合は、32 ビットの ESP レジスタ（スタックポインタ）が 4 インクリメントされ、それらの属性が 16 である場合は、16 ビットの SP レジスタが 2 インクリメントされる。（スタック・セグメントのセグメント・ディスクリプタの B フラグによって、スタックのアドレスサイズ属性が決まり、現在のコード・セグメントのセグメント・ディスクリプタの D フラグ（プリフィックスをともなう）によって、オペランド・サイズ属性およびデスティネーション・オペランドのアドレスサイズ属性が決まる。）

POP—Pop a Value from the Stack（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 の 1 つである場合は、レジスタにロードされる値は、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・レジスタにポップすると、そのセグメント・セレクタに関連するディスクリプタ情報がセグメント・レジスタの隠蔽（シャドー）部分に自動的にロードされ、セレクタおよびディスクリプタの情報が有効にされる（下記の「操作」の項を参照）。

一般保護フォルトを発生させずにヌル値（0000 ～ 0003）を DS、ES、FS、または GS レジスタにポップすることができる。ただし、対応するセグメント・レジスタにヌル値がロードされているセグメントをその後で参照しようとすると、一般保護例外（#GP）が発生する。この状況では、メモリ参照は行われず、セグメント・レジスタのセーブされた値はヌルである。

POP 命令では、値を CS レジスタにポップすることはできない。スタックから CS レジスタにロードするには、RET 命令を使用する。

メモリ内のデスティネーション・オペランドのアドレスを指定するベースレジスタとして ESP レジスタを使用すると、POP 命令は、ESP レジスタをインクリメントした後にオペランドの実効アドレスを計算する。16 ビット・スタックで POP 命令を実行した結果、ESP レジスタが 0h にラップされた場合は、結果のメモリ書き込みの位置はプロセッサのファミリによって異なる。

POP ESP 命令は、スタックの古いトップにあるデータをデスティネーションに書き込む前に、スタックポインタ（ESP）をインクリメントする。

POP SS 命令は、次の命令の実行後まで、NMI 割り込みを含めたすべての割り込みを禁止する。この処置によって、POP SS 命令および MOV ESP,EBP 命令を逐次に実行しても、割り込みによってスタックが無効になる危険はない[2]。ただし、SS レジスタおよび ESP レジスタをロードする方法としては、LSS 命令の使用が望ましい。

2. 以降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、次の命令シーケンス

```
STI
POP SS
POP ESP
```

では、STI も 1 命令の間割り込みをディレイさせるので、POP ESP が実行される前に、割り込みが認識されることがある。

POP—Pop a Value from the Stack（続き）

操作

```
IF StackAddrSize = 32
        THEN
        IF OperandSize = 32
            THEN
                DEST ← SS:ESP; (* copy a doubleword *)
                ESP ← ESP + 4;
            ELSE (* OperandSize = 16*)
                DEST ← SS:ESP; (* copy a word *)
            ESP ← ESP + 2;
        FI;
    ELSE (* StackAddrSize = 16* )
        IF OperandSize = 16
            THEN
                DEST ← SS:SP; (* copy a word *)
                SP ← SP + 2;
            ELSE (* OperandSize = 32 *)
                DEST ← SS:SP; (* copy a doubleword *)
                SP ← SP + 4;
        FI;
FI;
```

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・セレクタと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
```

POP—Pop a Value from the Stack（続き）

```
        OR ((segment is a data or nonconforming code segment)
            AND (both RPL and CPL > DPL))
                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。
	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・セグメントまたは読み取り可能なコード・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 および CPL の両方とも DPL より大きい場合。

POP—Pop a Value from the Stack（続き）

#SS(0)	スタックの現在のトップがスタック・セグメント内にない場合。 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）	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。
#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。

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。
#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。

POPA/POPAD—Pop All General-Purpose Registers

オペコード	命令	説明
61	POPA	DI、SI、BP、BX、DX、CX、AX をポップする。
61	POPAD	EDI、ESI、EBP、EBX、EDX、ECX、EAX をポップする。

説明

ダブルワード（POPAD）またはワード（POPA）をスタックから汎用レジスタにポップする。レジスタがロードされる順番は、（オペランド・サイズ属性が 32 である場合は）EDI、ESI、EBP、EBX、EDX、ECX、EAX であり、（オペランド・サイズ属性が 16である場合は）DI、SI、BP、BX、DX、CX、AXである。これらの命令は、PUSHA/PUSHAD 命令の逆の操作を実行する。ESP レジスタまたはSP レジスタのスタック上の値は無視される。その代わりに、ESP レジスタまたはSP レジスタは、各レジスタがロードされた後にインクリメントされる。

POPA（すべてをポップ）ニーモニックおよびPOPAD（すべてのダブルをポップ）ニーモニックは、同じオペコードを参照する。POPA 命令は、オペランド・サイズ属性が 16 であるときに使用するためのものであり、POPAD 命令は、オペランド・サイズ属性が 32 であるときに使用するためのものである。一部のアセンブラは、（必要な場合にはオペランド・サイズ・オーバライド・プリフィックス [66H] を使用して）POPA が使用されるときはオペランド・サイズを 16 に、POPAD が使用されるときは 32 に強制する。他のアセンブラは、これらのニーモニックをシノニム（POPA/POPAD）として取り扱い、オペランド・サイズ属性の現在の設定を使用して、使用されているニーモニックに関係なく、スタックからポップする値のサイズを決定することができる。（現在のコード・セグメントのセグメント・ディスクリプタのD フラグによって、オペランド・サイズ属性が決まる。）

操作

```
IF OperandSize = 32 (* instruction = POPAD *)
THEN
    EDI ← Pop();
    ESI ← Pop();
    EBP ← Pop();
    increment ESP by 4 (* skip next 4 bytes of stack *)
    EBX ← Pop();
    EDX ← Pop();
    ECX ← Pop();
    EAX ← Pop();
```

POPA/POPAD—Pop All General-Purpose Registers（続き）

```
ELSE (* OperandSize = 16, instruction = POPA *)
    DI ← Pop();
    SI ← Pop();
    BP ← Pop();
    increment ESP by 2 (* skip next 2 bytes of stack *)
    BX ← Pop();
    DX ← Pop();
    CX ← Pop();
    AX ← Pop();
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#SS(0)	開始スタックアドレスまたは終了スタックアドレスがスタック・セグメント内にない場合
#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

#SS	開始スタックアドレスまたは終了スタックアドレスがスタック・セグメント内にない場合。

仮想 8086 モード例外

#SS(0)	開始スタックアドレスまたは終了スタックアドレスがスタック・セグメント内にない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	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。

POPF/POPFD—Pop Stack into EFLAGS Register

オペコード	命令	説明
9D	POPF	スタックのトップをEFLAGSの下位16ビットにポップする。
9D	POPFD	スタックのトップを EFLAGS にポップする。

説明

（現在のオペランド・サイズ属性が 32 である場合は）ダブルワード（POPFD）をスタックのトップからポップして、値を EFLAGS レジスタにストアする。（オペランド・サイズ属性が 16 である場合は）ワードをスタックのトップからポップして、値を EFLAGS レジスタの下位 16 ビット（すなわち、FLAGS レジスタ）にストアする。これらの命令は、PUSHF/PUSHFD 命令の逆の操作を実行する。

POPF（フラグをポップ）ニーモニックおよび POPFD（フラグダブルをポップ）ニーモニックは、同じオペコードを参照する。POPF 命令は、オペランド・サイズ属性が 16 であるときに使用するためのものであり、POPFD 命令は、オペランド・サイズ属性が 32 であるときに使用するためのものである。一部のアセンブラは、POPF が使用されるときはオペランド・サイズを 16 に、POPFD が使用されるときは 32 に強制する。他のアセンブラは、これらのニーモニックをシノニム（POPF/POPFD）として取り扱い、オペランド・サイズ属性の現在の設定を使用して、使用されているニーモニックに関係なく、スタックからポップする値のサイズを決定することができる。

EFLAGS レジスタへの POPF/POPFD 命令の影響は、プロセッサの動作モードに応じて少し変わる。プロセッサが特権レベル 0 の保護モード（または特権レベル 0 に同等である実アドレスモード）で動作しているときは、VIP、VIF、VM フラグを除く EFLAGS レジスタのすべての非予約フラグは修正される可能性がある。VIP および VIF フラグはクリアされ、VM フラグは影響を受けない。

特権レベルが 0 より大きいけれども IOPL 以下である保護モードで動作しているときは、IOPL フィールドと VIP、VIF、VM フラグを除くすべてのフラグは修正される可能性がある。この場合も、IOPL のフラグは影響を受けず、VIP および VIF フラグはクリアされ、VM フラグは影響を受けない。割り込みフラグ（IF）は、IOPL と少なくとも同じ特権レベルで動作しているときだけに変更される。POPF/POPFD 命令が不十分な特権で実行されていても、例外は発生しないが、特権ビットは変わらない。

仮想 8086 モードで動作しているときは、I/O 特権レベル（IOPL）は、POPF/POPFD 命令を使用するには 3 に等しくなければならず、VM、RF、IOPL、VIP、VIF フラグは影響を受けない。IOPL が 3 より小さい場合、POPF/POPFD 命令は一般保護例外（#GP）を発生させる。

EFLAGS レジスタの詳細については、『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 3 章の「EFLAGS レジスタ」を参照のこと。

POPF/POPFD—Pop Stack into EFLAGS Register（続き）

操作

```
IF VM=0 (* Not in Virtual-8086 Mode *)
    THEN IF CPL=0
        THEN
            IF OperandSize = 32;
                THEN
                    EFLAGS ← Pop();
                    (* All non-reserved flags except VIP, VIF, and VM can be modified; *)
                    (* VIP and VIF are cleared; VM is unaffected*)
                ELSE (* OperandSize = 16 *)
                    EFLAGS[15:0] ← Pop(); (* All non-reserved flags can be modified; *)
            FI;
        ELSE (* CPL > 0 *)
            IF OperandSize = 32;
                THEN
                    EFLAGS ← Pop()
                    (* All non-reserved bits except IOPL, VIP, and VIF can be modified; *)
                    (* IOPL is unaffected; VIP and VIF are cleared; VM is unaffected *)
                ELSE (* OperandSize = 16 *)
                    EFLAGS[15:0] ← Pop();
                    (* All non-reserved bits except IOPL can be modified *)
                    (* IOPL is unaffected *)
            FI;
    FI;
    ELSE  (* In Virtual-8086 Mode *)
        IF IOPL=3
            THEN IF OperandSize=32
                THEN
                    EFLAGS ← Pop()
                    (* All non-reserved bits except VM, RF, IOPL, VIP, and VIF *)
                    (* can be modified; VM, RF, IOPL, VIP, and VIF are unaffected *)
                ELSE
                    EFLAGS[15:0] ← Pop()
                    (* All non-reserved bits except IOPL can be modified *)
                    (* IOPL is unaffected *)
            FI;
            ELSE (* IOPL < 3 *)
                #GP(0);  (* trap to virtual-8086 monitor *)
        FI;
    FI;
FI;
```

POPF/POPFD—Pop Stack into EFLAGS Register（続き）

影響を受けるフラグ

予約ビットと VM ビットを除くすべてのフラグ。

保護モード例外

#SS(0)	スタックのトップがスタック・セグメント内にない場合。
#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

#SS	スタックのトップがスタック・セグメント内にない場合。

仮想 8086 モード例外

#GP(0)	I/O 特権レベルが 3 より小さい場合。 オペランド・サイズ・オーバライド・プリフィックスの付いた POPF/POPFD 命令を実行しようとした場合。
#SS(0)	スタックのトップがスタック・セグメント内にない場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	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。

POR—Bitwise Logical OR

オペコード	命令	説明
0F EB /r	POR *mm*, *mm/m64*	*mm/m64* と *mm* のビット単位の OR（論理和）演算を実行する。
66 0F EB /r	POR *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の OR（論理和）演算を実行する。

説明

クワッドワードのソース・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）との間のビット単位の OR（論理和）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、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、XMM レジスタまたは 128 ビットのメモリ・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、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。結果の各ビットは、第 1 オペランドと第 2 オペランドの対応するビットのうちいずれか一方または両方が 1 の場合は 1 に設定され、それ以外の場合は 0 に設定される。

操作

```
DEST ← DEST OR SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
POR     __m64 _mm_or_si64(__m64 m1, __m64 m2)
POR     __m128i _mm_or_si128(__m128i m1, __m128i m2)
```

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。
	（128 ビット操作のみ）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。
#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	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

POR—Bitwise Logical OR（続き）

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PREFETCH*h*—Prefetch Data Into Caches

オペコード	命令	説明
0F 18 /1	PREFETCHT0 *m8*	T0 ヒントを使用して、*m8* のデータをプロセッサの近くに移動する。
0F 18 /2	PREFETCHT1 *m8*	T1 ヒントを使用して、*m8* のデータをプロセッサの近くに移動する。
0F 18 /3	PREFETCHT2 *m8*	T2 ヒントを使用して、*m8* のデータをプロセッサの近くに移動する。
0F 18 /0	PREFETCHNTA *m8*	NTA ヒントを使用して、*m8* のデータをプロセッサの近くに移動する。

説明

ソース・オペランドで指定されたバイトを含むメモリから、ローカリティのヒントで指定されたキャッシュ階層内の位置にデータのラインをフェッチする。

- T0 テンポラル・データ ― キャッシュ階層のすべてのレベルにデータをプリフェッチする。
  - ― インテル® Pentium® III プロセッサ ― 第 1 レベル・キャッシュまたは第 2 レベル・キャッシュ
  - ― インテル® Pentium® 4 プロセッサおよびインテル® Xeon™ プロセッサ ― 第 2 レベル・キャッシュ
- T1 第 1 キャッシュ・レベル以上についてテンポラル・データ ― L2 キャッシュ以上にデータをプリフェッチする。
  - ― インテル Pentium III プロセッサ ― 第 2 レベル・キャッシュ
  - ― インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ ― 第 2 レベル・キャッシュ
- T2 第 2 キャッシュ・レベル以上についてテンポラル・データ ― L2 キャッシュ以上にデータをプリフェッチする。
  - ― インテル Pentium III プロセッサ ― 第 2 レベル・キャッシュ
  - ― インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ ― 第 2 レベル・キャッシュ
- NTA すべてのキャッシュ・レベルについて非テンポラル・データ ― キャッシュの汚染を最小限に抑えて、非テンポラルなキャッシュ構造とプロセッサへのロケーション・クローズにデータをプリフェッチする。
  - ― インテル Pentium III プロセッサ ― 第 1 レベル・キャッシュ
  - ― インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ ― 第 2 レベル・キャッシュ

PREFETCH*h*—Prefetch Data Into Caches（続き）

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（ローカリティのヒントは、ModR/M バイトのビット 3 ～ 5 を使用して、マシンレベルの命令にエンコーディングされる。上記の値以外の ModR/M 値を指定すると、予測不可能な動作が発生する）。

選択されたラインが、指定されたレベルよりプロセッサに近いキャッシュ階層レベルにすでに存在する場合は、データの転送は行われない。キャッシュ不可メモリまたは WC メモリに対するプリフェッチ命令は無視される。

PREFETCH*h* 命令は単なるヒントであり、プログラムの動作には影響を与えない。この命令は、実行された場合、これから使用されることが予想されるデータを、プロセッサの近くに移動する。

プリフェッチのローカリティ・ヒントの影響はプロセッサによって異なり、オーバーロードされたり、無視されたりする可能性がある。プリフェッチされるデータのサイズもプロセッサによって異なるが、最小でも 32 バイトのデータがプリフェッチされる。

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、PREFETCH*h* 命令は、フェンス命令（MFENCE、SFENCE、LFENCE）やロックされたメモリ参照に対して順序付けされない。また、PREFETCH*h* 命令は、CLFLUSH 命令、他の PREFETCH*h* 命令、あるいは他の汎用命令に対しても順序付けされない。ただし、PREFETCH*h* 命令は、CPUID、WRMSR、OUT、MOV CR などのシリアル化命令に対しては順序付けされる。

操作

```
FETCH (m8);
```

同等のインテル® C/C++ コンパイラ組み込み関数

void _mm_prefetch(char *p, int i)

引き数 "*p" は、プリフェッチされるバイト（および対応するキャッシュ・ライン）のアドレスを示す。値 "i" は、実行されるプリフェッチ操作のタイプを指定する定数（_MM_HINT_T0、_MM_HINT_T1、_MM_HINT_T2、または _MM_HINT_NTA）を示す。

PREFETCH*h*—Prefetch Data Into Caches（続き）

数値例外

なし。

保護モード例外

なし。

実アドレスモード例外

なし。

仮想 8086 モード例外

なし。

PSADBW—Compute Sum of Absolute Differences

オペコード	命令	説明
0F F6 /r	PSADBW *mm1*, *mm2/m64*	*mm2/m64* と *mm1* の符号なしパックドバイト整数の差の絶対値を計算し、得られた差を合計して 1 符号なしワードの結果を返す。
66 0F F6 /r	PSADBW *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のパックド符号なしバイト整数の差の絶対値を計算し、下位の 8 つの差と上位の 8 つの差を別々に合計して 2 つのワード整数の結果を返す。

説明

ソース・オペランド（第 1 オペランド）の 8 つの符号なしバイト整数とデスティネーション・オペランド（第 2 オペランド）の 8 つの符号なしバイト整数の差の絶対値を計算する。次に、得られた 8 つの差を合計して 1 つの符号なしワード整数を求め、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、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションか、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・レジスタまたは XMM レジスタである。図 4-5. は、64 ビット・オペランドを使用する場合の PSADBW 命令の操作を示している。

64 ビット・オペランドを操作する場合は、結果のワード整数はデスティネーション・オペランドの下位ワードに格納され、デスティネーション・オペランドのその他のバイトはすべて 0 にクリアされる。

128 ビット・オペランドを操作する場合は、2 つの結果が計算され、パックされる。この場合は、ソース・オペランドとデスティネーション・オペランドの下位の 8 バイトを操作して 1 ワードの結果を求め、デスティネーション・オペランドの最下位ワードに格納する。また、上位の 8 バイトを操作して 1 ワードの結果を求め、デスティネーション・オペランドのビット 64 ～ 79 に格納する。デスティネーション・オペランドのその他のバイトはすべてクリアされる。

PSADBW—Compute Sum of Absolute Differences（続き）

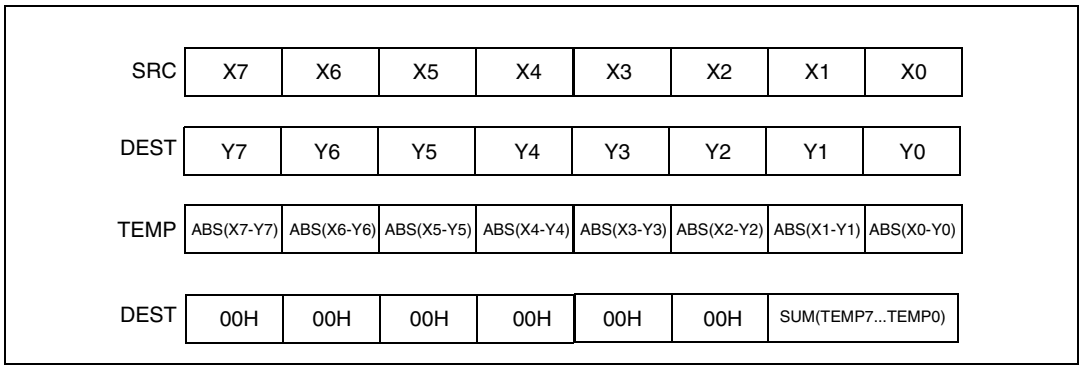


図 4-5. 64 ビット・オペランドを使用した PSADBW 命令の操作

操作

```
PSADBW instructions when using 64-bit operands:
    TEMP0 ← ABS(DEST[7-0] – SRC[7-0]);
    * repeat operation for bytes 2 through 6 *;
    TEMP7 ← ABS(DEST[63-56] – SRC[63-56]);
    DEST[15:0] ← SUM(TEMP0...TEMP7);
    DEST[63:16] ← 000000000000H;

PSADBW instructions when using 128-bit operands:
    TEMP0 ← ABS(DEST[7-0] – SRC[7-0]);
    * repeat operation for bytes 2 through 14 *;
    TEMP15 ← ABS(DEST[127-120] – SRC[127-120]);
    DEST[15-0] ← SUM(TEMP0...TEMP7);
    DEST[63-6] ← 000000000000H;
    DEST[79-64] ← SUM(TEMP8...TEMP15);
    DEST[127-80] ← 000000000000H;
```

同等のインテル® C/C++ コンパイラ組み込み関数

```
PSADBW          __m64_mm_sad_pu8(__m64 a,__m64 b)
PSADBW          __m128i _mm_sad_epu8(__m128i a, __m128i b)
```

影響を受けるフラグ

なし。

PSADBW—Compute Sum of Absolute Differences（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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PSHUFD—Shuffle Packed Doublewords

オペコード	命令	説明
66 0F 70 /r ib	PSHUFD *xmm1*, *xmm2/m128*, *imm8*	*imm8* のエンコーディングに基づいて、*xmm2/m128* のダブルワードをシャッフルし、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）からダブルワードをコピーして、オーダー・オペランド（第 3 オペランド）で選択されたデスティネーション・オペランド（第 1 オペランド）内の位置に挿入する。図 4-6. は、PSHUFD 命令の操作と、オーダー・オペランドのエンコーディングを示している。オーダー・オペランドのそれぞれの 2 ビット・フィールドは、デスティネーション・オペランドの 1 つのダブルワード位置の内容を指定する。例えば、オーダー・オペランドのビット 0 とビット 1 は、デスティネーション・オペランドのダブルワード 0 の内容を選択する。オーダー・オペランドのビット 0 とビット 1 のエンコーディング（図 4-6. の各フィールドのエンコーディングを参照）によって、ソース・オペランドのどのダブルワードがデスティネーション・オペランドのダブルワード 0 にコピーされるかが決まる。

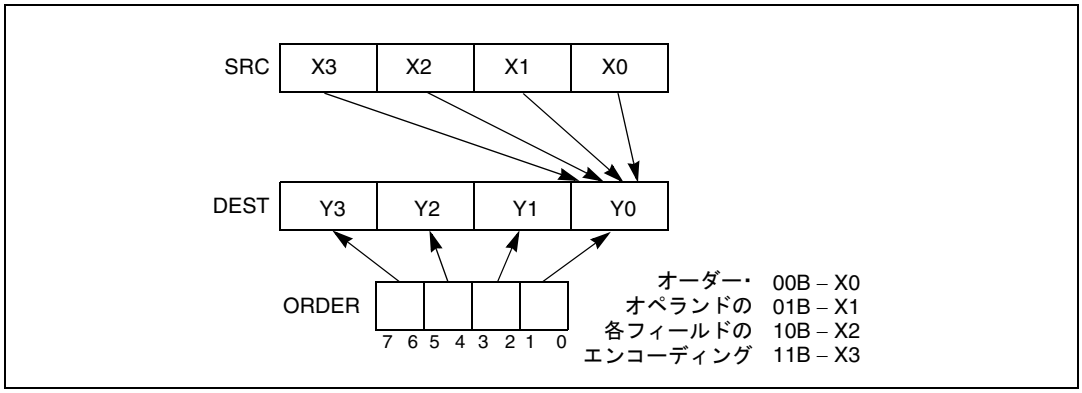


図 4-6. PSHUFD 命令の操作

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。オーダー・オペランドは 8 ビットの即値である。

この命令は、ソース・オペランドのダブルワードを、デスティネーション・オペランドの 2 つ以上のダブルワード位置にコピーすることができる。

PSHUFD—Shuffle Packed Doublewords（続き）

操作

```
DEST[31-0] ← (SRC >> (ORDER[1-0] * 32) )[31-0]
DEST[63-32]  ← (SRC >> (ORDER[3-2] * 32) )[31-0]
DEST[95-64]  ← (SRC >> (ORDER[5-4] * 32) )[31-0]
DEST[127-96]  ← (SRC >> (ORDER[7-6] * 32) )[31-0]
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSHUFD　　__m128i _mm_shuffle_epi32(__m128i a, int n)

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。
	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。
#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。
#NM	CR0 の 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。

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	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。
#NM	CR0 の TS がセットされた場合。

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。

PSHUFD—Shuffle Packed Doublewords（続き）

数値例外

なし。

PSHUFHW—Shuffle Packed High Words

オペコード	命令	説明
F3 0F 70 /r ib	PSHUFHW *xmm1*, *xmm2/m128*, *imm8*	*imm8* のエンコーディングに基づいて、*xmm2/m128* の上位ワードをシャッフルし、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）の上位クワッドワードからワードをコピーして、オーダー・オペランド（第 3 オペランド）で選択された、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内のワード位置に挿入する。この操作は、図 4-6. に示した PSHUFD 命令の操作とよく似ている。PSHUFHW 命令では、オーダー・オペランドのそれぞれの 2 ビット・フィールドは、デスティネーション・オペランドの上位クワッドワード内の 1 つのワード位置の内容を指定する。オーダー・オペランドの各フィールドの 2 進エンコーディングによって、ソース・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・オペランドにコピーされるワード（0、1、2、または 3、4）が選択される。ソース・オペランドの下位のクワッドワードがデスティネーション・オペランドの下位のクワッドワードにコピーされる。

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。オーダー・オペランドは 8 ビットの即値である。

この命令は、ソース・オペランドの上位のクワッドワードのワードを、デスティネーション・オペランドの上位のクワッドワード内の 2 つ以上のワード位置にコピーすることができる。

操作

```
DEST[63-0] ← (SRC[63-0]
DEST[79-64] ← (SRC >> (ORDER[1-0] * 16) )[79-64]
DEST[95-80] ← (SRC >> (ORDER[3-2] * 16) )[79-64]
DEST[111-96]  ← (SRC >> (ORDER[5-4] * 16) )[79-64]
DEST[127-112] ← (SRC >> (ORDER[7-6] * 16) )[79-64]
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSHUFHW　　__m128i _mm_shufflehi_epi16(__m128i a, int n)

影響を受けるフラグ

なし。

PSHUFHW—Shuffle Packed High Words（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。
	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。
#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。
#NM	CR0 の 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。

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	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。
#NM	CR0 の TS がセットされた場合。

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。

数値例外

なし。

PSHUFLW—Shuffle Packed Low Words

オペコード	命令	説明
F2 0F 70 /r ib	PSHUFLW *xmm1*, *xmm2/m128*, *imm8*	*imm8* のエンコーディングに基づいて、*xmm2/m128* の下位ワードをシャッフルし、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）の下位クワッドワードからワードをコピーして、オーダー・オペランド（第 3 オペランド）で選択された、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内のワード位置に挿入する。この操作は、図 4-6. に示した PSHUFD 命令の操作とよく似ている。PSHUFLW 命令では、オーダー・オペランドのそれぞれの 2 ビット・フィールドは、デスティネーション・オペランドの下位クワッドワード内の 1 つのワード位置の内容を指定する。オーダー・オペランドの各フィールドの 2 進エンコーディングによって、ソース・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・オペランドにコピーされるワード（0、1、2、または 3）が選択される。ソース・オペランドの上位のクワッドワードがデスティネーション・オペランドの上位のクワッドワードにコピーされる。

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。オーダー・オペランドは 8 ビットの即値である。

この命令は、ソース・オペランドの下位のクワッドワードのワードを、デスティネーション・オペランドの下位のクワッドワード内の 2 つ以上のワード位置にコピーすることができる。

操作

```
DEST[15-0] ← (SRC >> (ORDER[1-0] * 16) )[15-0]
DEST[31-16]  ← (SRC >> (ORDER[3-2] * 16) )[15-0]
DEST[47-32]  ← (SRC >> (ORDER[5-4] * 16) )[15-0]
DEST[63-48]  ← (SRC >> (ORDER[7-6] * 16) )[15-0]
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

PSHUFLW　　__m128i _mm_shufflelo_epi16(__m128i a, int n)

影響を受けるフラグ

なし。

PSHUFLW—Shuffle Packed Low Words（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。
	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。
#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。
#NM	CR0 の 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。

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	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。
#NM	CR0 の TS がセットされた場合。

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。

数値例外

なし。

PSHUFW—Shuffle Packed Words

オペコード	命令	説明
0F 70 /r ib	PSHUFW *mm1*, *mm2/m64*, *imm8*	*imm8* のエンコーディングに基づいて *mm2/m64* のワードをシャッフルし、*mm1* の結果にストアする。

説明

ソース・オペランド（第 2 オペランド）からワードをコピーして、オーダー・オペランド（第 3 オペランド）で選択されたデスティネーション・オペランド（第 1 オペランド）内のワード位置に挿入する。この操作は、図 4-6. に示した PSHUFD 命令の操作とよく似ている。PSHUFW 命令では、オーダー・オペランドのそれぞれの 2 ビット・フィールドは、デスティネーション・オペランドの 1 つのワード位置の内容を指定する。オーダー・オペランドの各フィールドのエンコーディングによって、ソース・オペランドからデスティネーション・オペランドにコピーされるワードが選択される。

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 MMX テクノロジ・レジスタである。オーダー・オペランドは 8 ビットの即値である。

この命令は、ソース・オペランドのワードを、デスティネーション・オペランドの 2 つ以上のワード位置にコピーすることができる。

操作

```
DEST[15-0] ← (SRC >> (ORDER[1-0] * 16) )[15-0]
DEST[31-16] ← (SRC >> (ORDER[3-2] * 16) )[15-0]
DEST[47-32] ← (SRC >> (ORDER[5-4] * 16) )[15-0]
DEST[63-48] ← (SRC >> (ORDER[7-6] * 16) )[15-0]
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSHUFW __m64 _mm_shuffle_pi16(__m64 a, int n)

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。
#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。
#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。

PSHUFW—Shuffle Packed Words（続き）

#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・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(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	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 x87 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。

数値例外

なし。

PSLLDQ—Shift Double Quadword Left Logical

オペコード	命令	説明
66 0F 73 /7 ib	PSLLDQ *xmm1*, *imm8*	*imm8* で指定されたバイト数だけ *xmm1* を左にシフトし、下位はゼロで埋める。

説明

カウント・オペランド（第 2 オペランド）で指定されたバイト数だけ、デスティネーション・オペランド（第 1 オペランド）を左にシフトする。空いた下位バイトはクリアされる（すべて 0 にセットされる）。カウント・オペランドで指定された値が 15 より大きい場合は、デスティネーション・オペランドはすべて 0 に設定される。デスティネーション・オペランドは XMM レジスタである。カウント・オペランドは 8 ビットの即値である。

操作

```
TEMP ← COUNT;
if (TEMP > 15) TEMP ← 16;
DEST ← DEST << (TEMP * 8);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSLLDQ __m128i _mm_slli_si128 (__m128i a, int imm)

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。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。

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。

数値例外

なし。

PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical

オペコード	命令	説明
0F F1 /r	PSLLW *mm*, *mm/m64*	*mm* のワードを *mm/m64* だけ左にシフトし、下位はゼロで埋める。
66 0F F1 /r	PSLLW *xmm1*, *xmm2/m128*	*xmm1* のワードを *xmm2/m128* だけ左にシフトし、下位はゼロで埋める。
0F 71 /6 ib	PSLLW *mm*, *imm8*	*mm* のワードを *imm8* だけ左にシフトし、下位はゼロで埋める。
66 0F 71 /6 ib	PSLLW *xmm1*, *imm8*	*xmm1* のワードを *imm8* だけ左にシフトし、下位はゼロで埋める。
0F F2 /r	PSLLD *mm*, *mm/m64*	*mm* のダブルワードを *mm/m64* だけ左にシフトし、下位はゼロで埋める。
66 0F F2 /r	PSLLD *xmm1*, *xmm2/m128*	*xmm1* のダブルワードを *xmm2/m128* だけ左にシフトし、下位はゼロで埋める。
0F 72 /6 ib	PSLLD *mm*, *imm8*	*mm* のダブルワードを *imm8* だけ左にシフトし、下位はゼロで埋める。
66 0F 72 /6 ib	PSLLD *xmm1*, *imm8*	*xmm1* のダブルワードを *imm8* だけ左にシフトし、下位はゼロで埋める。
0F F3 /r	PSLLQ *mm*, *mm/m64*	*mm* のクワッドワードを *mm/m64* だけ左にシフトし、下位はゼロで埋める。
66 0F F3 /r	PSLLQ *xmm1*, *xmm2/m128*	*xmm1* のクワッドワードを *xmm2/m128* だけ左にシフトし、下位はゼロで埋める。
0F 73 /6 ib	PSLLQ *mm*, *imm8*	*mm* のクワッドワードを *imm8* だけ左にシフトし、下位はゼロで埋める。
66 0F 73 /6 ib	PSLLQ *xmm1*, *imm8*	*xmm1* のクワッドワードを *imm8* だけ左にシフトし、下位はゼロで埋める。

説明

デスティネーション・オペランド（第 1 オペランド）にある個別のデータ要素（ワード、ダブルワード、クワッドワード）のビットを、カウント・オペランド（第 2 オペランド）に指定されたビット数だけ左にシフトする。データ要素のビットが左にシフトされると、空の下位ビットはクリアされる（ゼロに設定される）。カウント・オペランドによって指定される値が（ワードでは）15、（ダブルワードでは）31、または（クワッドワードでは）63 より大きいと、デスティネーション・オペランドはすべてゼロに設定される（図 4-7. は、64 ビット・オペランド内でワードをシフトする操作の例を示している）。デスティネーション・オペランドは、MMX® テクノロジ・レジスタまたは XMM レジスタでなければならない。カウント・オペランドには、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、XMM レジスタまたは 128 ビットのメモリ・ロケーション、または 8 ビットの即値を使用できる。

PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical（続き）

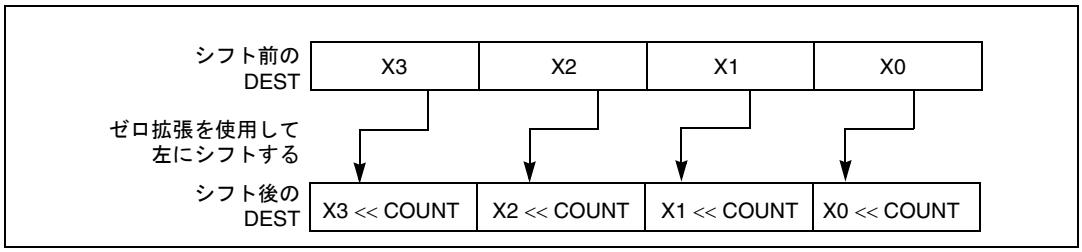


図 4-7. 64 ビット・オペランドを使用した PSLLW 命令、PSLLD 命令、PSLLQ 命令の動作

PSLLW 命令は、デスティネーション・オペランドのワードのそれぞれを、カウント・オペランドに指定されたビット数だけ左にシフトする。PSLLD 命令は、デスティネーション・オペランドのダブルワードのそれぞれをシフトし、PSLLQ 命令は、デスティネーション・オペランドのクワッドワードをシフトする。個別データ要素が左にシフトされると、空の下位ビット位置はゼロで埋められる。

操作

```
PSLLW instruction with 64-bit operand:
    IF (COUNT > 15)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST[15..0] ← ZeroExtend(DEST[15..0] << COUNT);
        * repeat shift operation for 2nd and 3rd words *;
        DEST[63..48] ← ZeroExtend(DEST[63..48] << COUNT);
    FI;

PSLLD instruction with 64-bit operand:
    IF (COUNT > 31)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST[31..0] ← ZeroExtend(DEST[31..0] << COUNT);
        DEST[63..32] ← ZeroExtend(DEST[63..32] << COUNT);
    FI;

PSLLQ instruction with 64-bit operand:
    IF (COUNT > 63)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST ← ZeroExtend(DEST << COUNT);
    FI;
```

PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical（続き）

```
PSLLW instruction with 128-bit operand:
    IF (COUNT > 15)
    THEN
    DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[15-0]  ← ZeroExtend(DEST[15-0] << COUNT);
        * repeat shift operation for 2nd through 7th words *;
        DEST[127-112] ← ZeroExtend(DEST[127-112] << COUNT);
    FI;

PSLLD instruction with 128-bit operand:
    IF (COUNT > 31)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[31-0]  ← ZeroExtend(DEST[31-0] << COUNT);
        * repeat shift operation for 2nd and 3rd doublewords *;
        DEST[127-96] ← ZeroExtend(DEST[127-96] << COUNT);
    FI;

PSLLQ instruction with 128-bit operand:
    IF (COUNT > 15)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[63-0]  ← ZeroExtend(DEST[63-0] << COUNT);
        DEST[127-64] ← ZeroExtend(DEST[127-64] << COUNT);
    FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSLLW	__m64 _mm_slli_pi16 (__m64 m, int count)
PSLLW	__m64 _mm_sll_pi16(__m64 m, __m64 count)
PSLLW	__m128i _mm_slli_pi16(__m64 m, int count)
PSLLW	__m128i _mm_slli_pi16(__m128i m, __m128i count)
PSLLD	__m64 _mm_slli_pi32(__m64 m, int count)
PSLLD	__m64 _mm_sll_pi32(__m64 m, __m64 count)
PSLLD	__m128i _mm_slli_epi32(__m128i m, int count)
PSLLD	__m128i _mm_sll_epi32(__m128i m, __m128i count)
PSLLQ	__m64 _mm_slli_si64(__m64 m, int count)
PSLLQ	__m64 _mm_sll_si64(__m64 m, __m64 count)
PSLLQ	__m128i _mm_slli_si64(__m128i m, int count)
PSLLQ	__m128i _mm_sll_si64(__m128i m, __m128i count)

影響を受けるフラグ

なし。

PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical（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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PSRAW/PSRAD—Shift Packed Data Right Arithmetic

オペコード	命令	説明
0F E1 /r	PSRAW *mm*, *mm/m64*	*mm* のワードを *mm/m64* だけ右にシフトし、上位は符号ビットで埋める。
66 0F E1 /r	PSRAW *xmm1*, *xmm2/m128*	*xmm1* のワードを *xmm2/m128* だけ右にシフトし、上位は符号ビットで埋める。
0F 71 /4 ib	PSRAW *mm*, *imm8*	*mm* のワードを *imm8* だけ右にシフトし、上位は符号ビットで埋める。
66 0F 71 /4 ib	PSRAW *xmm1*, *imm8*	*xmm1* のワードを *imm8* だけ右にシフトし、上位は符号ビットで埋める。
0F E2 /r	PSRAD *mm*, *mm/m64*	*mm* のダブルワードを *mm/m64* だけ右にシフトし、上位は符号ビットで埋める。
66 0F E2 /r	PSRAD *xmm1*, *xmm2/m128*	*xmm1* のダブルワードを *xmm2/m128* だけ右にシフトし、上位は符号ビットで埋める。
0F 72 /4 ib	PSRAD *mm*, *imm8*	*mm* のダブルワードを *imm8* だけ右にシフトし、上位は符号ビットで埋める。
66 0F 72 /4 ib	PSRAD *xmm1*, *imm8*	*xmm1* のダブルワードを *imm8* だけ右にシフトし、上位は符号ビットで埋める。

説明

デスティネーション・オペランド（第 1 オペランド）にある個別のデータ要素（ワードまたはダブルワード）のビットを、カウント・オペランド（第 2 オペランド）に指定されたビット量だけ右にシフトする。データ要素のビットが右にシフトされると、各要素の空の上位ビットは、データ要素の符号ビットの初期値で埋められる。カウント・オペランドによって指定される値が、（ワードでは）15 または（ダブルワードでは）31 より大きいと、デスティネーションの各データ要素は、その要素の符号ビットの初期値で埋められる（図 4-8. は、64 ビット・オペランド内でワードをシフトする操作の例を示している）。

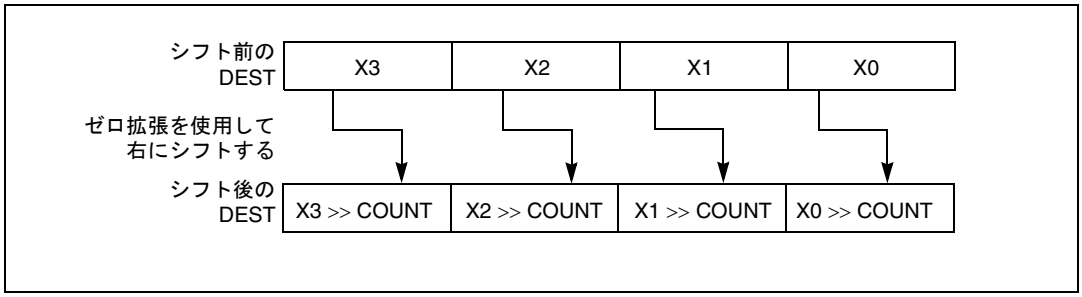


図 4-8. 64 ビット・オペランドを使用した PSRAW 命令と PSRAD 命令の動作

PSRAW/PSRAD—Shift Packed Data Right Arithmetic（続き）

デスティネーション・オペランドは、MMX® テクノロジ・レジスタまたはXMM レジスタでなければならない。カウント・オペランドには、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、XMM レジスタまたは128 ビットのメモリ・ロケーション、または8 ビットの即値を使用できる。

PSRAW 命令は、デスティネーション・オペランドのワードのそれぞれを、カウント・オペランドに指定されたビット数だけ右にシフトする。PSRAD 命令は、デスティネーション・オペランドのダブルワードのそれぞれをシフトする。

操作

```
PSRAW instruction with 64-bit operand:
    IF (COUNT > 15)
        THEN COUNT ← 16;
    FI;
        DEST[15..0] ← SignExtend(DEST[15..0] >> COUNT);
        * repeat shift operation for 2nd and 3rd words *;
        DEST[63..48] ← SignExtend(DEST[63..48] >> COUNT);

PSRAD instruction with 64-bit operand:
    IF (COUNT > 31)
        THEN COUNT ← 32;
    FI;
    ELSE
        DEST[31..0] ← SignExtend(DEST[31..0] >> COUNT);
        DEST[63..32] ← SignExtend(DEST[63..32] >> COUNT);

PSRAW instruction with 128-bit operand:
    IF (COUNT > 15)
        THEN COUNT ← 16;
    FI;
    ELSE
        DEST[15-0]  ← SignExtend(DEST[15-0] >> COUNT);
        * repeat shift operation for 2nd through 7th words *;
        DEST[127-112] ← SignExtend(DEST[127-112] >> COUNT);

PSRAD instruction with 128-bit operand:
    IF (COUNT > 31)
        THEN COUNT ← 32;
    FI;
    ELSE
        DEST[31-0]  ← SignExtend(DEST[31-0] >> COUNT);
        * repeat shift operation for 2nd and 3rd doublewords *;
        DEST[127-96] ← SignExtend(DEST[127-96] >>COUNT);
```

PSRAW/PSRAD—Shift Packed Data Right Arithmetic（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PSRAW	__m64 _mm_srai_pi16 (__m64 m, int count)
PSRAW	__m64 _mm_sraw_pi16 (__m64 m, __m64 count)
PSRAD	__m64 _mm_srai_pi32 (__m64 m, int count)
PSRAD	__m64 _mm_sra_pi32 (__m64 m, __m64 count)
PSRAW	__m128i _mm_srai_epi16(__m128i m, int count)
PSRAW	__m128i _mm_sra_epi16(__m128i m, __m128i count))
PSRAD	__m128i _mm_srai_epi32 (__m128i m, int count)
PSRAD	__m128i _mm_sra_epi32 (__m128i m, __m128i count)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。

PSRAW/PSRAD—Shift Packed Data Right Arithmetic（続き）

#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PSRLDQ—Shift Double Quadword Right Logical

オペコード	命令	説明
66 0F 73 /3 ib	PSRLDQ *xmm1*, *imm8*	*xmm1* を *imm8* だけ右にシフトし、上位はゼロで埋める。

説明

カウント・オペランド（第 2 オペランド）で指定されたバイト数だけ、デスティネーション・オペランド（第 1 オペランド）を右にシフトする。空いた上位バイトはクリアされる（すべて 0 にセットされる）。カウント・オペランドで指定された値が 15 より大きい場合は、デスティネーション・オペランドはすべて 0 に設定される。デスティネーション・オペランドは XMM レジスタである。カウント・オペランドは 8 ビットの即値である。

操作

```
TEMP ← COUNT;
if (TEMP > 15) TEMP ← 16;
DEST ← DEST >> (temp * 8);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSRLDQ　　__m128i _mm_srli_si128 (__m128i a, int imm)

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。
	CR4 の OSFXSR が 0 の場合。
	CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。

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。

数値例外

なし。

PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical

オペコード	命令	説明
0F D1 /r	PSRLW *mm*, *mm/m64*	*mm* のワードを *mm/m64* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
66 0F D1 /r	PSRLW *xmm1*, *xmm2/m128*	*xmm1* のワードを *xmm2/m128* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
0F 71 /2 ib	PSRLW *mm*, *imm8*	*mm* のワードを *imm8* だけ右にシフトし、上位はゼロで埋める。
66 0F 71 /2 ib	PSRLW *xmm1*, *imm8*	*xmm1* のワードを *imm8* だけ右にシフトし、上位はゼロで埋める。
0F D2 /r	PSRLD *mm*, *mm/m64*	*mm* のダブルワードを *mm/m64* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
66 0F D2 /r	PSRLD *xmm1*, *xmm2/m128*	*xmm1* のダブルワードを *xmm2/m128* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
0F 72 /2 ib	PSRLD *mm*, *imm8*	*mm* のダブルワードを *imm8* だけ右にシフトし、上位はゼロで埋める。
66 0F 72 /2 ib	PSRLD *xmm1*, *imm8*	*xmm1* のダブルワードを *imm8* だけ右にシフトし、上位はゼロで埋める。
0F D3 /r	PSRLQ *mm*, *mm/m64*	*mm* を *mm/m64* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
66 0F D3 /r	PSRLQ *xmm1*, *xmm2/m128*	*xmm1* のクワッドワードを *xmm/m128* に指定されたビットだけ右にシフトし、上位はゼロで埋める。
0F 73 /2 ib	PSRLQ *mm*, *imm8*	*mm* を *imm8* だけ右にシフトし、上位はゼロで埋める。
66 0F 73 /2 ib	PSRLQ *xmm1*, *imm8*	*xmm1* を *imm8* だけ右にシフトし、上位はゼロで埋める。

説明

デスティネーション・オペランド（第1オペランド）にある個別のデータ要素（ワード、ダブルワード、またはクワッドワード）のビットを、カウント・オペランド（第2オペランド）に指定されたビット量だけ右にシフトする。データ要素のビットが右にシフトされると、空の上位ビットはクリアされる（ゼロに設定される）。カウント・オペランドによって指定された値が（ワードでは）15、（ダブルワードでは）31、または（クワッドワードでは）63より大きいと、デスティネーション・オペランドはすべてゼロに設定される（図4-9.は、64ビット・オペランド内でワードをシフトする操作の例を示している）。デスティネーション・オペランドは、MMX® テクノロジ・レジスタまたはXMMレジスタでなければならない。カウント・オペランドには、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、XMMレジスタまたは128ビット・メモリ・ロケーション、または8ビットの即値を使用できる。

PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical（続き）

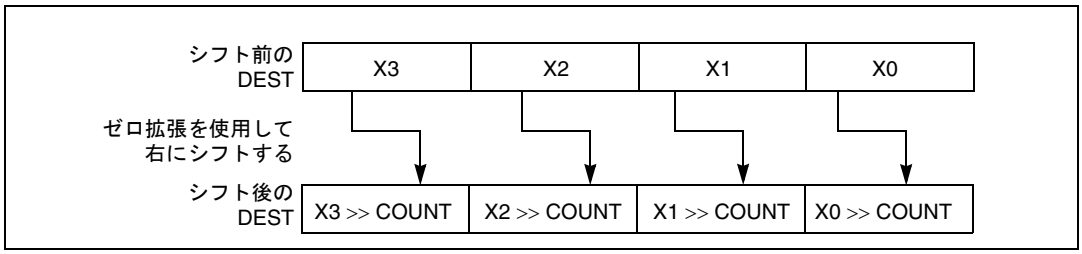


図 4-9. 64 ビット・オペランドを使用した PSRLW 命令、PSRLD 命令、PSRLQ 命令の動作

PSRLW 命令は、デスティネーション・オペランドのワードのそれぞれを、カウント・オペランドに指定されたビット数だけ右にシフトする。PSRLD 命令は、デスティネーション・オペランドのダブルワードのそれぞれをシフトし、PSRLQ 命令は、デスティネーション・オペランドの 64 ビットのクワッドワードをシフトする。

操作

```
PSRLW instruction with 64-bit operand:
    IF (COUNT > 15)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST[15..0] ← ZeroExtend(DEST[15..0] >> COUNT);
        * repeat shift operation for 2nd and 3rd words *;
        DEST[63..48] ← ZeroExtend(DEST[63..48] >> COUNT);
    FI;

PSRLD instruction with 64-bit operand:
    IF (COUNT > 31)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST[31..0] ← ZeroExtend(DEST[31..0] >> COUNT);
        DEST[63..32] ← ZeroExtend(DEST[63..32] >> COUNT);
    FI;

PSRLQ instruction with 64-bit operand:
    IF (COUNT > 63)
    THEN
        DEST[64..0] ← 0000000000000000H
    ELSE
        DEST ← ZeroExtend(DEST >> COUNT);
    FI;
```

PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical（続き）

```
PSRLW instruction with 128-bit operand:
    IF (COUNT > 15)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[15-0]  ← ZeroExtend(DEST[15-0] >> COUNT);
        * repeat shift operation for 2nd through 7th words *;
        DEST[127-112] ← ZeroExtend(DEST[127-112] >> COUNT);
    FI;

PSRLD instruction with 128-bit operand:
    IF (COUNT > 31)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[31-0]  ← ZeroExtend(DEST[31-0] >> COUNT);
        * repeat shift operation for 2nd and 3rd doublewords *;
        DEST[127-96] ← ZeroExtend(DEST[127-96] >> COUNT);
    FI;

PSRLQ instruction with 128-bit operand:
    IF (COUNT > 15)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[63-0]  ← ZeroExtend(DEST[63-0] >> COUNT);
        DEST[127-64] ← ZeroExtend(DEST[127-64] >> COUNT);
    FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSRLW	__m64 _mm_srli_pi16(__m64 m, int count)
PSRLW	__m64 _mm_srl_pi16 (__m64 m, __m64 count)
PSRLW	__m128i _mm_srli_epi16 (__m128i m, int count)
PSRLW	__m128i _mm_srl_epi16 (__m128i m, __m128i count)
PSRLD	__m64 _mm_srli_pi32 (__m64 m, int count)
PSRLD	__m64 _mm_srl_pi32 (__m64 m, __m64 count)
PSRLD	__m128i _mm_srli_epi32 (__m128i m, int count)
PSRLD	__m128i _mm_srl_epi32 (__m128i m, __m128i count)
PSRLQ	__m64 _mm_srli_si64 (__m64 m, int count)
PSRLQ	__m64 _mm_srl_si64 (__m64 m, __m64 count)
PSRLQ	__m128i _mm_srli_epi64 (__m128i m, int count)
PSRLQ	__m128i _mm_srl_epi64 (__m128i m, __m128i count)

PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical（続き）

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical（続き）

数値例外

なし。

PSUBB/PSUBW/PSUBD—Subtract Packed Integers

オペコード	命令	説明
0F F8 /r	PSUBB *mm*, *mm/m64*	*mm* のパックドバイト整数から *mm/m64* のパックドバイト整数を引く。
66 0F F8 /r	PSUBB *xmm1*, *xmm2/m128*	*xmm1* のパックドバイト整数から *xmm2/m128* のパックドバイト整数を引く。
0F F9 /r	PSUBW *mm*, *mm/m64*	*mm* のパックドワード整数から *mm/m64* のパックドワード整数を引く。
66 0F F9 /r	PSUBW *xmm1*, *xmm2/m128*	*xmm1* のパックドワード整数から *xmm2/m128* のパックドワード整数を引く。
0F FA /r	PSUBD *mm*, *mm/m64*	*mm* のパックド・ダブルワード整数から *mm/m64* のパックド・ダブルワード整数を引く。
66 0F FA /r	PSUBD *xmm1*, *xmm2/m128*	*xmm1* のパックド・ダブルワード整数から *xmm2/m128* のパックド・ダブルワード整数を引く。

説明

デスティネーション・オペランド（第 1 オペランド）のパックド整数からソース・オペランド（第 2 オペランド）のパックド整数を SIMD 減算し、結果のパックド整数をデスティネーション・オペランドに格納する。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 9-4. を参照のこと。以降の段落で説明するように、オーバーフローはラップアラウンドを使用して処理される。

上記の命令は、64 ビット・オペランドまたは 128 ビット・オペランドのいずれかを操作する。64 ビット・オペランドを操作する場合、デスティネーション・オペランドには MMX® テクノロジ・レジスタを使用しなければならないが、ソース・オペランドには MMX テクノロジ・レジスタまたは 64 ビット・メモリ・ロケーションのどちらを使用しても構わない。128 ビット・オペランドを操作する場合は、デスティネーション・オペランドには XMM レジスタを使用しなければならないが、ソース・オペランドには XMM レジスタまたは 128 ビット・メモリ・ロケーションのどちらを使用しても構わない。

PSUBB 命令は、パックドバイト整数を引く。個別の結果がバイトで表現するには大きすぎるまたは小さすぎるときは、結果はラップアラウンドされ、下位 8 ビットがデスティネーション要素に書き込まれる。

PSUBW 命令は、パックドワード整数を引く。個別の結果がワードで表現するには大きすぎるまたは小さすぎるときは、結果はラップアラウンドされ、下位 16 ビットがデスティネーション要素に書き込まれる。

PSUBB/PSUBW/PSUBD—Subtract Packed Integers（続き）

PSUBD 命令は、パックド・ダブルワード整数を引く。個別の結果がダブルワードで表現するには大きすぎるまたは小さすぎるときは、結果はラップアラウンドされ、下位 32 ビットがデスティネーション要素に書き込まれる。

PSUBB、PSUBW、PSUBD 命令は、符号なしまたは符号付き（2 の補数表記）のパックド整数を操作できることに注意すること。ただし、これらの命令は、オーバーフローやキャリーを示す EFLAGS レジスタ内のビットをセットしない。このため、検出されないオーバーフロー状態が発生しないように、操作される値の範囲をソフトウェアによって制御しなければならない。

操作

```
PSUBB instruction with 64-bit operands:
    DEST[7..0] ← DEST[7..0] – SRC[7..0];
    * repeat subtract operation for 2nd through 7th byte *;
    DEST[63..56] ← DEST[63..56] – SRC[63..56];

PSUBB instruction with 128-bit operands:
    DEST[7-0] ← DEST[7-0] – SRC[7-0];
    * repeat subtract operation for 2nd through 14th byte *;
    DEST[127-120] ← DEST[111-120] – SRC[127-120];

PSUBW instruction with 64-bit operands:
    DEST[15..0] ← DEST[15..0] – SRC[15..0];
    * repeat subtract operation for 2nd and 3rd word *;
    DEST[63..48] ← DEST[63..48] – SRC[63..48];

PSUBW instruction with 128-bit operands:
    DEST[15-0]  ← DEST[15-0] – SRC[15-0];
    * repeat subtract operation for 2nd through 7th word *;
    DEST[127-112] ← DEST[127-112] – SRC[127-112];

PSUBD instruction with 64-bit operands:
    DEST[31..0] ← DEST[31..0] – SRC[31..0];
    DEST[63..32] ← DEST[63..32] – SRC[63..32];

PSUBD instruction with 128-bit operands:
    DEST[31-0]  ← DEST[31-0] – SRC[31-0];
    * repeat subtract operation for 2nd and 3rd doubleword *;
    DEST[127-96] ← DEST[127-96] – SRC[127-96];
```

PSUBB/PSUBW/PSUBD—Subtract Packed Integers（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PSUBB	__m64 _mm_sub_pi8(__m64 m1, __m64 m2)
PSUBW	__m64 _mm_sub_pi16(__m64 m1, __m64 m2)
PSUBD	__m64 _mm_sub_pi32(__m64 m1, __m64 m2)
PSUBB	__m128i _mm_sub_epi8 (__m128i a, __m128i b)
PSUBW	__m128i _mm_sub_epi16 (__m128i a, __m128i b)
PSUBD	__m128i _mm_sub_epi32 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

PSUBB/PSUBW/PSUBD—Subtract Packed Integers（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)	（64 ビット操作のみ）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。

数値例外

なし。

PSUBQ—Subtract Packed Quadword Integers

オペコード	命令	説明
0F FB /r	PSUBQ *mm1*, *mm2/m64*	*mm2/m64* から *mm1* のクワッドワード整数を引く。
66 0F FB /r	PSUBQ *xmm1*, *xmm2/m128*	*xmm2/m128* から *xmm1* のパックド・クワッドワード整数を引く。

説明

第 1 オペランド（デスティネーション・オペランド）から第 2 オペランド（ソース・オペランド）を引き、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、MMX® テクノロジ・レジスタまたは 64 ビットのメモリ・ロケーションに格納される 1 つのクワッドワード整数か、XMM レジスタまたは 128 ビットのメモリ・ロケーションに格納される 2 つのパックド・クワッドワード整数である。デスティネーション・オペランドは、MMX テクノロジ・レジスタに格納される 1 つのクワッドワード整数か、XMM レジスタに格納される 2 つのパックド・クワッドワード整数である。パックド・クワッドワードのオペランドを使用する場合は、SIMD 減算が実行される。結果のクワッドワードが大きすぎて 64 ビットで表現できない場合は（オーバーフロー）、結果はラップアラウンドされ、下位 64 ビットがデスティネーション要素に書き込まれる（すなわち、キャリーは無視される）。

PSUBQ 命令は、符号なし整数と符号付き整数（2 の補数記法）のどちらを操作することもできる。ただし、この命令は、オーバーフローやキャリーを示す EFLAGS レジスタ内のビットをセットしない。このため、検出されないオーバーフロー状態が発生しないように、操作される値の範囲をソフトウェアによって制御しなければならない。

操作

```
PSUBQ instruction with 64-Bit operands:
    DEST[63-0] ← DEST[63-0] – SRC[63-0];

PSUBQ instruction with 128-Bit operands:
    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

```
PSUBQ       __m64 _mm_sub_si64(__m64 m1, __m64 m2)
PSUBQ       __m128i _mm_sub_epi64(__m128i m1, __m128i m2)
```

影響を受けるフラグ

なし。

PSUBQ—Subtract Packed Quadword Integers（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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、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。

実アドレスモード例外

#GP(0)	（128 ビット操作のみ）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	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）アライメント・チェックが有効になっており、アライメントの合っていないメモリ参照を行った場合。

数値例外

なし。

PSUBSB/PSUBSW—Subtract Packed Signed Integers with Signed Saturation

オペコード	命令	説明
0F E8 /r	PSUBSB *mm*, *mm/m64*	*mm* の符号付きパックドバイトから *mm/m64* の符号付きパックドバイトを引き、結果を飽和させる。
66 0F E8 /r	PSUBSB *xmm1*, *xmm2/m128*	*xmm1* の符号つきパックドバイトから *xmm2/m128* の符号付きパックドバイトを引き、結果を飽和させる。
0F E9 /r	PSUBSW *mm*, *mm/m64*	*mm* の符号付きパックドワードから *mm/m64* の符号付きパックドワードを引き、結果を飽和させる。
66 0F E9 /r	PSUBSW *xmm1*, *xmm2/m128*	*xmm1* の符号つきパックドワードから *xmm2/m128* の符号付きパックドワードを引き、結果を飽和させる。

説明

デスティネーション・オペランド（第1オペランド）のパックド符号付き整数からソース・オペランド（第2オペランド）のパックド符号付き整数をSIMD減算し、結果のパックド整数をデスティネーション・オペランドに格納する。オーバーフローは、以下に説明する方法で飽和処理される。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9-4.を参照のこと。以降の段落で説明するように、オーバーフローは符号付き飽和処理を使用して処理される。

上記の命令は、64ビット・オペランドまたは128ビット・オペランドのいずれかを操作する。64ビット・オペランドを操作する場合、デスティネーション・オペランドにはMMX®テクノロジ・レジスタを使用しなければならないが、ソース・オペランドにはMMXテクノロジ・レジスタまたは64ビット・メモリ・ロケーションのどちらを使用しても構わない。128ビット・オペランドを操作する場合は、デスティネーション・オペランドにはXMMレジスタを使用しなければならないが、ソース・オペランドにはXMMレジスタまたは128ビット・メモリ・ロケーションのどちらを使用しても構わない。

PSUBSB命令は、符号付きパックドバイト整数を引く。個別のバイト結果が符号付きバイト整数の範囲を超える場合（すなわち、7FHより大きいかまたは80Hより小さい場合）は、それぞれ7FHまたは80Hの飽和された値がデスティネーション・オペランドに書き込まれる。

PSUBSW命令は、符号付きパックドワード整数を引く。個別のワード結果が符号付きワード整数の範囲を超える場合（すなわち、7FFFHより大きいかまたは8000Hより小さい場合）は、それぞれ7FFFHまたは8000Hの飽和された値がデスティネーション・オペランドに書き込まれる。

PSUBSB/PSUBSW—Subtract Packed Signed Integers with Signed Saturation（続き）

操作

```
PSUBSB instruction with 64-bit operands:
    DEST[7..0] ← SaturateToSignedByte(DEST[7..0] – SRC (7..0]) ;
    * repeat subtract operation for 2nd through 7th bytes *;
    DEST[63..56] ← SaturateToSignedByte(DEST[63..56] – SRC[63..56] );

PSUBSB instruction with 128-bit operands:
    DEST[7-0] ← SaturateToSignedByte (DEST[7-0] – SRC[7-0]);
    * repeat subtract operation for 2nd through 14th bytes *;
    DEST[127-120] ← SaturateToSignedByte (DEST[111-120] – SRC[127-120]);

PSUBSW instruction with 64-bit operands
    DEST[15..0] ← SaturateToSignedWord(DEST[15..0] – SRC[15..0] );
    * repeat subtract operation for 2nd and 7th words *;
    DEST[63..48] ← SaturateToSignedWord(DEST[63..48] – SRC[63..48] );

PSUBSW instruction with 128-bit operands
    DEST[15-0]  ← SaturateToSignedWord (DEST[15-0] – SRC[15-0]);
    * repeat subtract operation for 2nd through 7th words *;
    DEST[127-112] ← SaturateToSignedWord (DEST[127-112] – SRC[127-112]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSUBSB	__m64 _mm_subs_pi8(__m64 m1, __m64 m2)
PSUBSB	__m128i _mm_subs_epi8(__m128i m1, __m128i m2)
PSUBSW	__m64 _mm_subs_pi16(__m64 m1, __m64 m2)
PSUBSW	__m128i _mm_subs_epi16(__m128i m1, __m128i m2)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

PSUBSB/PSUBSW—Subtract Packed Signed Integers with Signed Saturation（続き）

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。 （128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。 （128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PSUBUSB/PSUBUSW—Subtract Packed Unsigned Integers with Unsigned Saturation

オペコード	命令	説明
0F D8 /r	PSUBUSB *mm*, *mm/m64*	*mm* の符号なしパックドバイトから *mm/m64* の符号なしパックドバイトを引き、結果を飽和させる。
66 0F D8 /r	PSUBUSB *xmm1*, *xmm2/m128*	*xmm1* の符号なしパックドバイトから *xmm2/m128* の符号なしパックドバイトを引き、結果を飽和させる。
0F D9 /r	PSUBUSW *mm*, *mm/m64*	*mm* の符号なしパックドワードから *mm/m64* の符号なしパックドワードを引き、結果を飽和させる。
66 0F D9 /r	PSUBUSW *xmm1*, *xmm2/m128*	*xmm1* の符号なしパックドワードから *xmm2/m128* の符号なしパックドワードを引き、結果を飽和させる。

説明

デスティネーション・オペランド（第1オペランド）のパックド符号なし整数からソース・オペランド（第2オペランド）のパックド符号なし整数をSIMD減算し、結果のパックド整数をデスティネーション・オペランドに格納する。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9-4.を参照のこと。以降の段落で説明するように、オーバーフローは符号なし飽和処理を使用して処理される。

上記の命令は、64ビット・オペランドまたは128ビット・オペランドのいずれかを操作する。64ビット・オペランドを操作する場合、デスティネーション・オペランドにはMMX®テクノロジ・レジスタを使用しなければならないが、ソース・オペランドにはMMXテクノロジ・レジスタまたは64ビット・メモリ・ロケーションのどちらを使用しても構わない。128ビット・オペランドを操作する場合は、デスティネーション・オペランドにはXMMレジスタを使用しなければならないが、ソース・オペランドにはXMMレジスタまたは128ビット・メモリ・ロケーションのどちらを使用しても構わない。

PSUBUSB命令は、符号なしパックドバイト整数を引く。個別のバイト結果がゼロより小さい場合は、00Hの飽和された符号なし値がデスティネーション・オペランドに書き込まれる。

PSUBUSW命令は、符号なしパックドワード整数を引く。個別のワード結果がゼロより小さい場合は、0000Hの飽和された符号なし値がデスティネーション・オペランドに書き込まれる。

PSUBUSB/PSUBUSW—Subtract Packed Unsigned Integers with Unsigned Saturation（続き）

操作

```
PSUBUSB instruction with 64-bit operands:
    DEST[7..0] ← SaturateToUnsignedByte(DEST[7..0] – SRC (7..0] );
    * repeat add operation for 2nd through 7th bytes *:
    DEST[63..56] ← SaturateToUnsignedByte(DEST[63..56] – SRC[63..56]

PSUBUSB instruction with 128-bit operands:
    DEST[7-0] ← SaturateToUnsignedByte (DEST[7-0] – SRC[7-0]);
    * repeat add operation for 2nd through 14th bytes *:
    DEST[127-120] ← SaturateToUnSignedByte (DEST[127-120] – SRC[127-120]);

PSUBUSW instruction with 64-bit operands:
    DEST[15..0] ← SaturateToUnsignedWord(DEST[15..0] – SRC[15..0] );
    * repeat add operation for 2nd and 3rd words *:
    DEST[63..48] ← SaturateToUnsignedWord(DEST[63..48] – SRC[63..48] );

PSUBUSW instruction with 128-bit operands:
    DEST[15-0]  ← SaturateToUnsignedWord (DEST[15-0] – SRC[15-0]);
    * repeat add operation for 2nd through 7th words *:
    DEST[127-112] ← SaturateToUnSignedWord (DEST[127-112] – SRC[127-112]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

PSUBUSB	__m64 _mm_sub_pu8(__m64 m1, __m64 m2)
PSUBUSB	__m128i _mm_sub_epu8(__m128i m1, __m128i m2)
PSUBUSW	__m64 _mm_sub_pu16(__m64 m1, __m64 m2)
PSUBUSW	__m128i _mm_sub_epu16(__m128i m1, __m128i m2)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 FPU 例外がある場合。

PSUBUSB/PSUBUSW—Subtract Packed Unsigned Integers with Unsigned Saturation（続き）

#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。 （128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。 （128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ— Unpack High Data

オペコード	命令	説明
0F 68 /r	PUNPCKHBW *mm*, *mm/m64*	*mm* および *mm/m64* から *mm* に上位バイトをアンパックしてインタリーブする。
66 0F 68 /r	PUNPCKHBW *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に上位バイトをアンパックしてインタリーブする。
0F 69 /r	PUNPCKHWD *mm*, *mm/m64*	*mm* および *mm/m64* から *mm* に上位ワードをアンパックしてインタリーブする。
66 0F 69 /r	PUNPCKHWD *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に上位ワードをアンパックしてインタリーブする。
0F 6A /r	PUNPCKHDQ *mm*, *mm/m64*	*mm* および *mm/m64* から *mm* に上位ダブルワードをアンパックしてインタリーブする。
66 0F 6A /r	PUNPCKHDQ *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に上位ダブルワードをアンパックしてインタリーブする。
66 0F 6D /r	PUNPCKHQDQ *xmm1*, *xmm2/m128*	*xmm1*および*xmm2/m128*から*xmm1*に上位クワッドワードをアンパックしてインタリーブする。

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）の上位データ要素（バイト、ワード、ダブルワードまたはクワッドワード）をデスティネーション・オペランドにアンパックしてインタリーブする（図 4-10. はビット・オペランドのバイトでアンパックされた動作を示す）。下位データ要素は無視される。

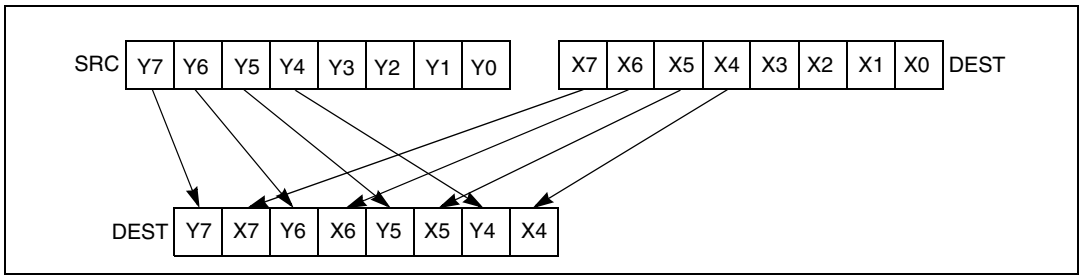


図 4-10. 64 ビット・オペランドを使用した PUNPCKHBW 命令の動作

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ— Unpack High Data（続き）

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、XMM レジスタまたは 128 ビットのメモリ・ロケーションが使用できる。デスティネーション・オペランドには、MMX テクノロジ・レジスタまたは XMM レジスタを使用できる。ソースデータが 64 ビットのメモリ・オペランドからであるときは、64 ビット・オペランドすべてがメモリからアクセスされるが、命令は上位 32 ビットだけを使用する。ソースデータが 128 ビット・メモリ・オペランドからのものである場合、プロセッサによっては、適切な 64 ビットだけがフェッチされる。ただし、この場合も、16 バイト境界へのアライメントと通常のセグメント・チェックが適用される。

PUNPCKHBW 命令は、ソース・オペランドとデスティネーション・オペランドの上位バイトをインタリーブする。PUNPCKHWD 命令は、ソース・オペランドとデスティネーション・オペランドの上位ワードをインタリーブする。PUNPCKHDQ 命令は、ソース・オペランドとデスティネーション・オペランドの上位ダブルワードをインタリーブし、PUNPCKHQDQ 命令は、ソース・オペランドとデスティネーション・オペランドの上位クワッドワードをインタリーブする。

上記の命令では、ソース・オペランドにすべてゼロを入れることにより、バイトからワード、ワードからダブルワード、ダブルワードからクワッドワード、クワッドワードからダブル・クワッドワードへと、それぞれ変換することができる。ソース・オペランドがすべてゼロである場合、（デスティネーション・オペランドにストアされる）結果は、デスティネーション・オペランドの元の値の上位データ要素がゼロ拡張されたものになる。例えば、PUNPCKHBW 命令では、上位バイトがゼロ拡張され（すなわち、符号なしワード整数にアンパックされる）、PUNPCKHWD 命令では、上位ワードがゼロ拡張される（符号なしダブルワード整数にアンパックされる）。

操作

```
PUNPCKHBW instruction with 64-bit operands:
        DEST[7..0] ← DEST[39..32];
        DEST[15..8] ← SRC[39..32];
        DEST[23..16] ← DEST[47..40];
        DEST[31..24] ← SRC[47..40];
        DEST[39..32] ← DEST[55..48];
        DEST[47..40] ← SRC[55..48];
        DEST[55..48] ← DEST[63..56];
        DEST[63..56] ← SRC[63..56];
```

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ— Unpack High Data（続き）

PUNPCKHW instruction with 64-bit operands:

```
DEST[15..0] ← DEST[47..32];
DEST[31..16] ← SRC[47..32];
DEST[47..32] ← DEST[63..48];
DEST[63..48] ← SRC[63..48];
```

PUNPCKHDQ instruction with 64-bit operands:

```
DEST[31..0] ← DEST[63..32]
DEST[63..32] ← SRC[63..32];
```

PUNPCKHBW instruction with 128-bit operands:

```
DEST[7-0]   ← DEST[71-64];
DEST[15-8]  ← SRC[71-64];
DEST[23-16] ← DEST[79-72];
DEST[31-24] ← SRC[79-72];
DEST[39-32] ← DEST[87-80];
DEST[47-40] ← SRC[87-80];
DEST[55-48] ← DEST[95-88];
DEST[63-56] ← SRC[95-88];
DEST[71-64] ← DEST[103-96];
DEST[79-72] ← SRC[103-96];
DEST[87-80] ← DEST[111-104];
DEST[95-88] ← SRC[111-104];
DEST[103-96]  ← DEST[119-112];
DEST[111-104] ← SRC[119-112];
DEST[119-112] ← DEST[127-120];
DEST[127-120] ← SRC[127-120];
```

PUNPCKHWD instruction with 128-bit operands:

```
DEST[15-0]  ← DEST[79-64];
DEST[31-16] ← SRC[79-64];
DEST[47-32] ← DEST[95-80];
DEST[63-48] ← SRC[95-80];
DEST[79-64] ← DEST[111-96];
DEST[95-80] ← SRC[111-96];
DEST[111-96]  ← DEST[127-112];
DEST[127-112] ← SRC[127-112];
```

PUNPCKHDQ instruction with 128-bit operands:

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

PUNPCKHQDQ instruction:

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

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ— Unpack High Data（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PUNPCKHBW __m64 _mm_unpackhi_pi8(__m64 m1, __m64 m2)
PUNPCKHBW __m128i _mm_unpackhi_epi8(__m128i m1, __m128i m2)
PUNPCKHWD __m64 _mm_unpackhi_pi16(__m64 m1,__m64 m2)
PUNPCKHWD __m128i _mm_unpackhi_epi16(__m128i m1,__m128i m2)
PUNPCKHDQ __m64 _mm_unpackhi_pi32(__m64 m1, __m64 m2)
PUNPCKHDQ __m128i _mm_unpackhi_epi32(__m128i m1, __m128i m2)
PUNPCKHQDQ __m128i _mm_unpackhi_epi64 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。

PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ—Unpack High Data（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ— Unpack Low Data

オペコード	命令	説明
0F 60 /r	PUNPCKLBW *mm*, *mm/m32*	*mm* および *mm/m64* から *mm* に下位バイトをアンパックしてインタリーブする。
66 0F 60 /r	PUNPCKLBW *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に下位バイトをアンパックしてインタリーブする。
0F 61 /r	PUNPCKLWD *mm*, *mm/m32*	*mm* および *mm/m64* から *mm* に下位ワードをアンパックしてインタリーブする。
66 0F 61 /r	PUNPCKLWD *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に下位ワードをアンパックしてインタリーブする。
0F 62 /r	PUNPCKLDQ *mm*, *mm/m32*	*mm* および *mm/m64* から *mm* に下位ダブルワードをアンパックしてインタリーブする。
66 0F 62 /r	PUNPCKLDQ *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* に下位ダブルワードをアンパックしてインタリーブする。
66 0F 6C /r	PUNPCKLQDQ *xmm1*, *xmm2/m128*	*xmm1* および *xmm2/m128* から *xmm1* レジスタに下位クワッドワードをアンパックしてインタリーブする。

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）の下位データ要素（バイト、ワード、ダブルワード、およびクワッドワード）をデスティネーション・オペランドにアンパックしてインタリーブする（図 4-11. は 64 ビット・オペランドのバイトでアンパックされた動作を示す）。上位データ要素は無視される。

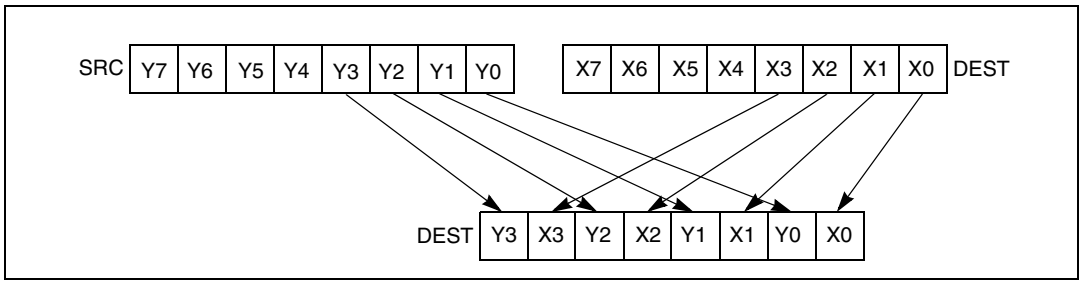


図 4-11. 64 ビット・オペランドを使用した PUNPCKLBW 命令の動作

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ— Unpack Low Data（続き）

ソース・オペランドには、MMX®テクノロジ・レジスタまたは32ビットのメモリ・ロケーション、XMMレジスタまたは128ビットのメモリ・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、MMXテクノロジ・レジスタまたはXMMレジスタを使用できる。ソースデータが128ビット・メモリ・オペランドからのものである場合、プロセッサによっては、適切な64ビットだけがフェッチされる。ただし、この場合も、16バイト境界へのアライメントと通常のセグメント・チェックが適用される。

PUNPCKLBW命令は、ソース・オペランドとデスティネーション・オペランドの下位バイトをインタリーブする。PUNPCKLWD命令は、ソース・オペランドとデスティネーション・オペランドの下位ワードをインタリーブする。PUNPCKLDQ命令は、ソース・オペランドとデスティネーション・オペランドの下位ダブルワードをインタリーブする。

上記の命令では、ソース・オペランドにすべてゼロを入れることにより、バイトからワード、ワードからダブルワード、ダブルワードからクワッドワード、クワッドワードからダブル・クワッドワードへと、それぞれ変換することができる。ソース・オペランドがすべてゼロである場合、（デスティネーション・オペランドにストアされる）結果は、デスティネーション・オペランドの元の値の上位データ要素がゼロ拡張されたものになる。例えば、PUNPCKLBW命令では、上位バイトがゼロ拡張され（すなわち、符号なしワード整数にアンパックされる）、PUNPCKLWD命令では、上位ワードがゼロ拡張される（符号なしダブルワード整数にアンパックされる）。

操作

```
PUNPCKLBW instruction with 64-bit operands:
        DEST[63..56] ← SRC[31..24];
        DEST[55..48] ← DEST[31..24];
        DEST[47..40] ← SRC[23..16];
        DEST[39..32] ← DEST[23..16];
        DEST[31..24] ← SRC[15..8];
        DEST[23..16] ← DEST[15..8];
        DEST[15..8] ← SRC[7..0];
        DEST[7..0] ← DEST[7..0];

PUNPCKLWD instruction with 64-bit operands:
        DEST[63..48] ← SRC[31..16];
        DEST[47..32] ← DEST[31..16];
        DEST[31..16] ← SRC[15..0];
        DEST[15..0] ← DEST[15..0];
```

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ— Unpack Low Data（続き）

PUNPCKLDQ instruction with 64-bit operands:
DEST[63..32] ← SRC[31..0];
DEST[31..0] ← DEST[31..0];

PUNPCKLBW instruction with 128-bit operands:
DEST[7-0] ← DEST[7-0];
DEST[15-8] ← SRC[7-0];
DEST[23-16] ← DEST[15-8];
DEST[31-24] ← SRC[15-8];
DEST[39-32] ← DEST[23-16];
DEST[47-40] ← SRC[23-16];
DEST[55-48] ← DEST[31-24];
DEST[63-56] ← SRC[31-24];
DEST[71-64] ← DEST[39-32];
DEST[79-72] ← SRC[39-32];
DEST[87-80] ← DEST[47-40];
DEST[95-88] ← SRC[47-40];
DEST[103-96] ← DEST[55-48];
DEST[111-104] ← SRC[55-48];
DEST[119-112] ← DEST[63-56];
DEST[127-120] ← SRC[63-56];

PUNPCKLWD instruction with 128-bit operands:
DEST[15-0] ← DEST[15-0];
DEST[31-16] ← SRC[15-0];
DEST[47-32] ← DEST[31-16];
DEST[63-48] ← SRC[31-16];
DEST[79-64] ← DEST[47-32];
DEST[95-80] ← SRC[47-32];
DEST[111-96] ← DEST[63-48];
DEST[127-112] ← SRC[63-48];

PUNPCKLDQ instruction with 128-bit operands:
DEST[31-0] ← DEST[31-0];
DEST[63-32] ← SRC[31-0];
DEST[95-64] ← DEST[63-32];
DEST[127-96] ← SRC[63-32];
PUNPCKLQDQ
DEST[63-0] ← DEST[63-0];
DEST[127-64] ← SRC[63-0];

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ— Unpack Low Data（続き）

同等のインテル® C/C++ コンパイラ組み込み関数

PUNPCKLBW　__m64 _mm_unpacklo_pi8 (__m64 m1, __m64 m2)
PUNPCKLBW　__m128i _mm_unpacklo_epi8 (__m128i m1, __m128i m2)
PUNPCKLWD　__m64 _mm_unpacklo_pi16 (__m64 m1, __m64 m2)
PUNPCKLWD　__m128i _mm_unpacklo_epi16 (__m128i m1, __m128i m2)
PUNPCKLDQ　__m64 _mm_unpacklo_pi32 (__m64 m1, __m64 m2)
PUNPCKLDQ　__m128i _mm_unpacklo_epi32 (__m128i m1, __m128i m2)
PUNPCKLQDQ　__m128i _mm_unpacklo_epi64 (__m128i m1, __m128i m2)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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(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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。

PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ— Unpack Low Data（続き）

#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

PUSH—Push Word or Doubleword onto the Stack

オペコード	命令	説明
FF /6	PUSH *r/m16*	*r/m16* をプッシュする。
FF /6	PUSH *r/m32*	*r/m32* をプッシュする。
50+*rw*	PUSH *r16*	*r16* をプッシュする。
50+*rd*	PUSH *r32*	*r32* をプッシュする。
6A	PUSH *imm8*	*imm8* をプッシュする。
68	PUSH *imm16*	*imm16* をプッシュする。
68	PUSH *imm32*	*imm32* をプッシュする。
0E	PUSH CS	CS をプッシュする。
16	PUSH SS	SS をプッシュする。
1E	PUSH DS	DS をプッシュする。
06	PUSH ES	ES をプッシュする。
0F A0	PUSH FS	FS をプッシュする。
0F A8	PUSH GS	GS をプッシュする。

説明

スタックポインタをデクリメントし、次にソース・オペランドをスタックのトップにストアする。スタック・セグメントのアドレスサイズ属性によって、スタックポインタのサイズ（16 ビットまたは 32 ビット）が決まり、現在のコード・セグメントのオペランド・サイズ属性によって、スタックポインタをデクリメントする量（2 バイトまたは 4 バイト）が決まる。例えば、これらのアドレスサイズ属性およびオペランド・サイズ属性が 32 である場合は、32 ビットの ESP レジスタ（スタックポインタ）が 4 デクリメントされ、それらの属性が 16 である場合は、16 ビットの SP レジスタが 2 デクリメントされる。（スタック・セグメントのセグメント・ディスクリプタの B フラグによって、スタックのアドレスサイズ属性が決まり、現在のコード・セグメントのセグメント・ディスクリプタの D フラグ（プリフィックスを伴う）によって、オペランド・サイズ属性およびソース・オペランドのアドレスサイズ属性が決まる。）スタックのアドレスサイズ属性が 32 であるときに 16 ビット・オペランドをプッシュすると、スタックポインタでアライメント不正が発生することがある（すなわち、スタックポインタはダブルワード境界にアライメントが合っていない）。

PUSH ESP 命令は、命令が実行される前に ESP レジスタの値が存在していたようにその値をプッシュする。そのため、ESP レジスタがオペランド・アドレスの計算にベースレジスタとして使用されるメモリ・オペランドを、PUSH 命令が使用すると、ESP レジスタがデクリメントされる前に、オペランドの実効アドレスが計算される。

PUSH—Push Word or Doubleword onto the Stack（続き）

実アドレスモードでは、PUSH 命令が実行されるときに、ESP レジスタまたは SP レジスタが 1 である場合、プロセッサはスタックスペースの不足によってシャットダウンする。この状態を示す例外は生成されない。

IA-32 アーキテクチャにおける互換性

インテル® 286 プロセッサ以降の IA-32 プロセッサでは、PUSH ESP 命令は、命令が実行される前に ESP レジスタの値が存在していたようにその値をプッシュする。（このことは、実アドレスモードまたは仮想 8086 モードでも同じである。）インテル® 8086 プロセッサでは、PUSH SP 命令は、（2 デクリメントされた後の値である）SP レジスタの新しい値をプッシュする。

操作

```
IF StackAddrSize = 32
THEN
    IF OperandSize = 32
        THEN
            ESP ← ESP – 4;
            SS:ESP ← SRC; (* push doubleword *)
        ELSE (* OperandSize = 16*)
            ESP ← ESP – 2;
            SS:ESP ← SRC; (* push word *)
    FI;
ELSE (* StackAddrSize = 16*)
    IF OperandSize = 16
        THEN
            SP ← SP – 2;
             SS:SP ← SRC; (* push word *)
        ELSE (* OperandSize = 32*)
            SP ← SP – 4;
            SS:SP ← SRC; (* push doubleword *)
    FI;
FI;
```

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。

PUSH—Push Word or Doubleword onto the Stack（続き）

#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。 SP レジスタまたは 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

#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。

PUSHA/PUSHAD—Push All General-Purpose Registers

オペコード	命令	説明
60	PUSHA	AX、CX、DX、BX、元の SP、BP、SI、および DI をプッシュする。
60	PUSHAD	EAX、ECX、EDX、EBX、元の ESP、EBP、ESI、および EDI をプッシュする。

説明

汎用レジスタの内容をスタックにプッシュする。レジスタがスタックにストアされる順番は、（現在のオペランド・サイズ属性が 32 である場合は）EAX、ECX、EDX、EBX、ESP（元の値）、EBP、ESI、EDI であり、（オペランド・サイズ属性が 16 である場合は）AX、CX、DX、BX、SP（元の値）、BP、SI、DI である。これらの命令は、POPA/POPAD 命令の逆の操作を実行する。ESP レジスタまたは SP レジスタとしてプッシュされる値は、最初のレジスタをプッシュする前のその値である（下記の「操作」の項を参照）。

PUSHA（すべてをプッシュ）ニーモニックおよび PUSHAD（すべてのダブルをプッシュ）ニーモニックは、同じオペコードを参照する。PUSHA 命令は、オペランド・サイズ属性が 16 であるときに使用するためのものであり、PUSHAD 命令は、オペランド・サイズ属性が 32 であるときに使用するためのものである。一部のアセンブラは、PUSHA が使用されるときはオペランド・サイズを 16 に、PUSHAD が使用されるときは 32 に強制する。他のアセンブラは、これらのニーモニックをシノニム（PUSHA/PUSHAD）として取り扱い、オペランド・サイズ属性の現在の設定を使用して、使用されるニーモニックに関係なく、スタックにプッシュする値のサイズを決定することができる。

実アドレスモードでは、PUSHA/PUSHAD 命令が実行されるときに、ESP レジスタまたは SP レジスタが 1、3、または 5 である場合、プロセッサはスタックスペースの不足によってシャットダウンする。この状態を示す例外は生成されない。

操作

```
IF OperandSize = 32 (* PUSHAD instruction *)
    THEN
        Temp ← (ESP);
        Push(EAX);
        Push(ECX);
        Push(EDX);
        Push(EBX);
        Push(Temp);
        Push(EBP);
        Push(ESI);
        Push(EDI);
```

PUSHA/PUSHAD—Push All General-Purpose Register（続き）

```
    ELSE (* OperandSize = 16, PUSHA instruction *)
        Temp ← (SP);
        Push(AX);
        Push(CX);
        Push(DX);
        Push(BX);
        Push(Temp);
        Push(BP);
        Push(SI);
        Push(DI);
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#SS(0)	開始スタックアドレスまたは終了スタックアドレスがスタック・セグメント内にない場合。
#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	ESP レジスタまたは SP レジスタの内容が 7、9、11、13、または 15 である場合。

仮想 8086 モード例外

#GP(0)	ESP レジスタまたは SP レジスタの内容が 7、9、11、13、または 15 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。

PUSHF/PUSHFD—Push EFLAGS Register onto the Stack

オペコード	命令	説明
9C	PUSHF	EFLAGS の下位 16 ビットをプッシュする。
9C	PUSHFD	EFLAGS をプッシュする。

説明

（現在のオペランド・サイズ属性が 32 である場合は）スタックポインタを 4 デクリメントし、EFLAGS レジスタの内容全体をスタックにプッシュする。（オペランド・サイズ属性が 16 である場合は）スタックポインタを 2 デクリメントし、EFLAGS レジスタの下位 16 ビット（すなわち、FLAGS レジスタ）をスタックにプッシュする。（これらの命令は、POPF/POPFD 命令の逆の操作を実行する。）EFLAGS レジスタ全体をスタックにコピーするときは、VM および RF フラグ（ビット 16 および 17）はコピーされない。その代わりに、これらのフラグの値は、スタックにストアされる EFLAGS イメージでクリアされる。EFLAGS レジスタの詳細については、『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 3 章の「EFLAGS レジスタ」の項を参照のこと。

PUSHF（フラグをプッシュ）ニーモニックおよび PUSHFD（ダブルのフラグをプッシュ）ニーモニックは、同じオペコードを参照する。PUSHF 命令は、オペランド・サイズ属性が 16 であるときに使用するためのものであり、PUSHFD 命令は、オペランド・サイズ属性が 32 であるときに使用するためのものである。一部のアセンブラは、PUSHF が使用されるときはオペランド・サイズを 16 に、PUSHFD が使用されるときは 32 に強制する。他のアセンブラは、これらのニーモニックをシノニム（PUSHF/PUSHFD）として取り扱い、オペランド・サイズ属性の現在の設定を使用して、使用されているニーモニックに関係なく、スタックにプッシュする値のサイズを決定することができる。

仮想 8086 モードで I/O 特権レベル（IOPL）が 3 より小さい場合、PUSHF/PUSHFD 命令は一般保護例外（#GP）を発生させる。

実アドレスモードでは、PUSHF/PUSHFD 命令が実行されるときに、ESP レジスタまたは SP レジスタが 1、3、または 5 である場合、プロセッサはスタックスペースの不足によってシャットダウンする。この状態を示す例外は生成されない。

PUSHF/PUSHFD—Push EFLAGS Register onto the Stack（続き）

操作

```
IF (PE=0) OR (PE=1 AND ((VM=0) OR (VM=1 AND IOPL=3)))
(* Real-Address Mode, Protected mode, or Virtual-8086 mode with IOPL equal to 3 *)
    THEN
        IF OperandSize = 32
            THEN
                push(EFLAGS AND 00FCFFFFH);
                (* VM and RF EFLAG bits are cleared in image stored on the stack*)
            ELSE
                push(EFLAGS); (* Lower 16 bits only *)
        FI;
    ELSE (* In Virtual-8086 Mode with IOPL less than 3 *)
        #GP(0); (* Trap to virtual-8086 monitor *)
FI;
```

影響を受けるフラグ

なし。

保護モード例外

#SS(0)	ESP　レジスタの新しい値がスタック・セグメントの境界外にある場合。
#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

なし。

仮想 8086 モード例外

#GP(0)	I/O 特権レベルが 3 より小さい場合。
#PF（フォルトコード）	ページフォルトが発生した場合。
#AC(0)	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。

PXOR—Logical Exclusive OR

オペコード	命令	説明
0F EF /r	PXOR *mm*, *mm/m64*	*mm/m64* と *mm* のビット単位の XOR（排他的論理和）演算を実行する。
66 0F EF /r	PXOR *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の XOR（排他的論理和）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）との間のビット単位の XOR（排他的論理和）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、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、XMM レジスタまたは 128 ビットのメモリ・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、MMX テクノロジ・レジスタまたは XMM レジスタを使用できる。各ビットの結果は、2 つのオペランドの対応するビットが異なる場合は 1 になり、同じ場合は 0 になる。

操作

```
DEST ← DEST XOR SRC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

PXOR __m64 _mm_xor_si64 (__m64 m1, __m64 m2)
PXOR __m128i _mm_xor_si128 (__m128i a, __m128i b)

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。
	（128 ビット操作のみ）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。
#UD	CR0 の EM がセットされた場合。
	（128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。
	（128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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。

PXOR—Logical Exclusive OR（続き）

#AC(0)	（64 ビット操作のみ）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(0)	（128 ビット操作のみ）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。 （128 ビット操作のみ）CR4 の OSFXSR が 0 の場合。 （128 ビット操作のみ）CPUID 機能フラグ SSE2 が 0 の場合。
#NM	CR0 の TS がセットされた場合。
#MF	（64 ビット操作のみ）未処理の x87 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)	（64 ビット操作のみ）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。

数値例外

なし。

RCL/RCR/ROL/ROR—Rotate

オペコード	命令	説明
D0 /2	RCL *r/m8*, 1	9 ビット（CF、*r/m8*）を左に 1 回回転させる。
D2 /2	RCL *r/m8*, CL	9 ビット（CF、*r/m8*）を左に CL 回回転させる。
C0 /2 *ib*	RCL *r/m8*, *imm8*	9 ビット（CF、*r/m8*）を左に *imm8* 回回転させる。
D1 /2	RCL *r/m16*, 1	17 ビット（CF、*r/m16*）を左に 1 回回転させる。
D3 /2	RCL *r/m16*, CL	17 ビット（CF、*r/m16*）を左に CL 回回転させる。
C1 /2 *ib*	RCL *r/m16*, *imm8*	17 ビット（CF、*r/m16*）を左に *imm8* 回回転させる。
D1 /2	RCL *r/m32*, 1	33 ビット（CF、*r/m32*）を左に 1 回回転させる。
D3 /2	RCL *r/m32*, CL	33 ビット（CF、*r/m32*）を左に CL 回回転させる。
C1 /2 *ib*	RCL *r/m32*, *imm8*	33 ビット（CF、*r/m32*）を左に *imm8* 回回転させる。
D0 /3	RCR *r/m8*, 1	9 ビット（CF、*r/m8*）を右に 1 回回転させる。
D2 /3	RCR *r/m8*, CL	9 ビット（CF、*r/m8*）を右に CL 回回転させる。
C0 /3 *ib*	RCR *r/m8*, *imm8*	9 ビット（CF、*r/m8*）を右に *imm8* 回回転させる。
D1 /3	RCR *r/m16*, 1	17 ビット（CF、*r/m16*）を右に 1 回回転させる。
D3 /3	RCR *r/m16*, CL	17 ビット（CF、*r/m16*）を右に CL 回回転させる。
C1 /3 *ib*	RCR *r/m16*, *imm8*	17 ビット（CF、*r/m16*）を右に *imm8* 回回転させる。
D1 /3	RCR *r/m32*, 1	33 ビット（CF、*r/m32*）を右に 1 回回転させる。
D3 /3	RCR *r/m32*, CL	33 ビット（CF、*r/m32*）を右に CL 回回転させる。
C1 /3 *ib*	RCR *r/m32*, *imm8*	33 ビット（CF、*r/m32*）を右に *imm8* 回回転させる。
D0 /0	ROL *r/m8*, 1	8 ビット *r/m8* を左に 1 回回転させる。
D2 /0	ROL *r/m8*, CL	8 ビット *r/m8* を左に CL 回回転させる。
C0 /0 *ib*	ROL *r/m8*, *imm8*	8 ビット *r/m8* を左に *imm8* 回回転させる。
D1 /0	ROL *r/m16*, 1	16 ビット *r/m16* を左に 1 回回転させる。
D3 /0	ROL *r/m16*, CL	16 ビット *r/m16* を左に CL 回回転させる。
C1 /0 *ib*	ROL *r/m16*, *imm8*	16 ビット *r/m16* を左に *imm8* 回回転させる。
D1 /0	ROL *r/m32*, 1	32 ビット *r/m32* を左に 1 回回転させる。
D3 /0	ROL *r/m32*, CL	32 ビット *r/m32* を左に CL 回回転させる。
C1 /0 *ib*	ROL *r/m32*, *imm8*	32 ビット *r/m32* を左に *imm8* 回回転させる。
D0 /1	ROR *r/m8*, 1	8 ビット *r/m8* を右に 1 回回転させる。
D2 /1	ROR *r/m8*, CL	8 ビット *r/m8* を右に CL 回回転させる。
C0 /1 *ib*	ROR *r/m8*, *imm8*	8 ビット *r/m8* を右に *imm8* 回回転させる。
D1 /1	ROR *r/m16*, 1	16 ビット *r/m16* を右に 1 回回転させる。
D3 /1	ROR *r/m16*, CL	16 ビット *r/m16* を右に CL 回回転させる。
C1 /1 *ib*	ROR *r/m16*, *imm8*	16 ビット *r/m16* を右に *imm8* 回回転させる。
D1 /1	ROR *r/m32*, 1	32 ビット *r/m32* を右に 1 回回転させる。
D3 /1	ROR *r/m32*, CL	32 ビット *r/m32* を右に CL 回回転させる。
C1 /1 *ib*	ROR *r/m32*, *imm8*	32 ビット *r/m32* を右に *imm8* 回回転させる。

RCL/RCR/ROL/ROR-—Rotate（続き）

説明

第 1 オペランド（デスティネーション・オペランド）のビットを第 2 オペランド（カウント・オペランド）に指定されたビット位置の数だけシフトし（回転させ）、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。カウント・オペランドは、即値または CL レジスタの値が可能で、符号なし整数である。プロセッサは、最下位 5 ビットを除くカウント・オペランドのすべてのビットをマスクして、カウントを 0 から 31 までの数に制限する。

左に回転（ROL）命令およびキャリーを通じて左に回転（RCL）命令は、すべてのビットを最上位ビット位置の方向にシフトするが、最上位ビットだけは最下位ビット・ロケーションに回転される（『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、上巻』の図 7-11. を参照）。右に回転（ROR）命令およびキャリーを通じて右に回転（RCR）命令は、すべてのビットを最下位ビット位置の方向にシフトするが、最下位ビットだけは最上位ビット・ロケーションに回転される（『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、上巻』の図 7-11. を参照）。

RCL 命令および RCR 命令は、回転の中に CF フラグを含める。RCL 命令は、CF フラグを最下位ビットにシフトさせ、最上位ビットを CF フラグにシフトさせる（『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 7-11. を参照）。RCR 命令は、CF フラグを最上位ビットにシフトさせ、最下位ビットを CF フラグにシフトさせる（『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 7-11. を参照）。ROL 命令および ROR 命令では、CF フラグの元の値は結果には含められないが、CF フラグには 1 端からもう 1 端にシフトされたビットのコピーが入る。

OF フラグは、1 ビットの回転だけに定義され、（どのフラグにも影響を与えないゼロビットの回転が何も行わないことを除いて）その他のすべての場合は未定義である。左回転では、OF フラグは、（回転後の）CF ビットと結果の最上位ビットとの排他的論理和に設定される。右回転では、OF フラグは、結果の最上位 2 ビットの排他的論理和に設定される。

IA-32 アーキテクチャにおける互換性

8086 は、回転カウントをマスクしない。ただし、（ インテル® 286 プロセッサから始まる）他のすべての IA-32 プロセッサは、回転カウントを 5 ビットにマスクするので、最大カウントは 31 になる。このマスク設定は（仮想 8086 モードを含めた）すべての動作モードで行われて、命令の最大実行時間を減少させる。

RCL/RCR/ROL/ROR-—Rotate（続き）

操作

```
(* RCL and RCR instructions *)
SIZE ← OperandSize
CASE (determine count) OF
    SIZE ← 8:    tempCOUNT ← (COUNT AND 1FH) MOD 9;
    SIZE ← 16:   tempCOUNT ← (COUNT AND 1FH) MOD 17;
    SIZE ← 32:   tempCOUNT ← COUNT AND 1FH;
ESAC;
(* RCL instruction operation *)
WHILE (tempCOUNT ≠ 0)
    DO
        tempCF ← MSB(DEST);
        DEST ← (DEST ∗ 2) + CF;
        CF ← tempCF;
        tempCOUNT ← tempCOUNT – 1;
    OD;
ELIHW;
IF COUNT = 1
    THEN OF ← MSB(DEST) XOR CF;
    ELSE OF is undefined;
FI;
(* RCR instruction operation *)
IF COUNT = 1
    THEN OF ← MSB(DEST) XOR CF;
    ELSE OF is undefined;
FI;
WHILE (tempCOUNT ≠ 0)
    DO
        tempCF ← LSB(SRC);
        DEST ← (DEST / 2) + (CF ∗ 2^SIZE);
        CF ← tempCF;
        tempCOUNT ← tempCOUNT – 1;
    OD;
(* ROL and ROR instructions *)
SIZE ← OperandSize
CASE (determine count) OF
    SIZE ← 8:    tempCOUNT ← COUNT MOD 8;
    SIZE ← 16:   tempCOUNT ← COUNT MOD 16;
    SIZE ← 32:   tempCOUNT ← COUNT MOD 32;
ESAC;
(* ROL instruction operation *)
WHILE (tempCOUNT ≠ 0)
    DO
        tempCF ← MSB(DEST);
        DEST ← (DEST ∗ 2) + tempCF;
        tempCOUNT ← tempCOUNT – 1;
    OD;
ELIHW;
CF ← LSB(DEST);
```

RCL/RCR/ROL/ROR-—Rotate（続き）

```
IF COUNT = 1
    THEN OF ← MSB(DEST) XOR CF;
    ELSE OF is undefined;
FI;
(* ROR instruction operation *)
WHILE (tempCOUNT ≠ 0)
    DO
        tempCF ← LSB(SRC);
        DEST ← (DEST / 2) + (tempCF ∗ 2^SIZE);
        tempCOUNT ← tempCOUNT – 1;
    OD;
ELIHW;
CF ← MSB(DEST);
IF COUNT = 1
    THEN OF ← MSB(DEST) XOR MSB – 1(DEST);
    ELSE OF is undefined;
FI;
```

影響を受けるフラグ

CF フラグは、そこにシフトされたビットの値を持つ。OF フラグは、単一ビット回転（上記の「説明」の項を参照）の場合だけに影響を受け、複数ビット回転の場合には未定義である。SF、ZF、AF、PF フラグは影響を受けない。

保護モード例外

#GP(0)	ソース・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。

RCL/RCR/ROL/ROR-—Rotate（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。

RCPPS—Compute Reciprocals of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 53 /r	RCPPS *xmm1*, *xmm2/m128*	*xmm2/m128* のパック ド単精度浮動小数点値の逆数の近似値を計算し、その結果を *xmm1* にストアする。

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の逆数の近似値を SIMD 計算し、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。

この近似値の相対誤差は以下のとおりである。

| 相対誤差 | ≤ $1.5 * 2^{-12}$

RCPSS 命令は、MXCSR レジスタの丸め制御ビットの影響を受けない。ソースの値が 0.0 の場合は、ソースの値と同じ符号の∞が返される。ソースの値がデノーマルの場合は、（同じ符号の）0.0 として処理される。極小の結果は、オペランドと同じ符号の 0.0 に常にフラッシュされる（入力値が $|1.11111111110100000000000B*2^{125}|$ と等しいかまたはそれより大きい場合は、極小の結果は生成されないことが保証されている。入力値が $|1.00000000000110000000001B*2^{126}|$ と等しいかまたはそれより小さい場合は、極小の結果が生成され、0.0 にフラッシュされることが保証されている。入力値がこの範囲の中間にある場合は、プロセッサのモデルによって、極小の結果が生成されることも生成されないこともある）。ソースの値が SNaN または QNaN の場合は、QNaN に変換された SNaN か、ソースの QNaN が返される。

操作

```
DEST[31-0] ← APPROXIMATE(1.0/(SRC[31-0]));
DEST[63-32] ← APPROXIMATE(1.0/(SRC[63-32]));
DEST[95-64] ← APPROXIMATE(1.0/(SRC[95-64]));
DEST[127-96] ← APPROXIMATE(1.0/(SRC[127-96]));
```

同等のインテル® C/C++ コンパイラ組み込み関数

RCCPS　　__m128 _mm_rcp_ps(__m128 a)

SIMD 浮動小数点例外

なし。

RCPPS—Compute Reciprocals of 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。
#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。

RCPSS—Compute Reciprocal of Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F 53 /r	RCPSS *xmm1*, *xmm2/m32*	*xmm2/m128* のスカラ単精度浮動小数点値の逆数の近似値を計算し、その結果を *xmm1* 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の逆数の近似値を計算し、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。

この近似値の相対誤差は以下のとおりである。

$|\text{相対誤差}| \leq 1.5 * 2^{-12}$

RCPSS 命令は、MXCSR レジスタの丸め制御ビットの影響を受けない。ソースの値が 0.0 の場合は、ソースの値と同じ符号の∞が返される。ソースの値がデノーマルの場合は、（同じ符号の）0.0 として処理される。極小の結果は、オペランドと同じ符号の 0.0 に常にフラッシュされる（入力値が $|1.11111111110100000000000B*2^{125}|$ と等しいかまたはそれより大きい場合は、極小の結果は生成されないことが保証されている。入力値が $|1.00000000000110000000001B*2^{126}|$ と等しいかまたはそれより小さい場合は、極小の結果が生成され、0.0 にフラッシュされることが保証されている。入力値がこの範囲の中間にある場合は、プロセッサのモデルによって、極小の結果が生成されることも生成されないこともある）。ソースの値が SNaN または QNaN の場合は、QNaN に変換された SNaN か、ソースの QNaN が返される。

操作

```
DEST[31-0] ← APPROX (1.0/(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

RCPSS　　__m128 _mm_rcp_ss(__m128 a)

SIMD 浮動小数点例外

なし。

RCPSS—Compute Reciprocal of 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)	アライメントの合っていないメモリ参照を行った場合。

RDMSR—Read from Model Specific Register

オペコード	命令	説明
0F 32	RDMSR	ECXによって指定されるMSRをEDX:EAXにロードする。

説明

ECX レジスタに指定された 64 ビットのモデル固有レジスタ（MSR）の内容をレジスタ EDX:EAX にロードする。ECX レジスタにロードされる入力値は、読み取られる MSR のアドレスである。EDX レジスタには MSR の上位 32 ビットがロードされ、EAX レジスタには下位 32 ビットがロードされる。読み取られる MSR に 64 より少ないビットしかインプリメントされていない場合は、インプリメントされていないビット・ロケーションで EDX:EAX に返される値は未定義である。

この命令は、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。そうしないと、一般保護例外 #GP(0) が生成される。予約されているかまたはインプリメントされていない MSR アドレスを ECX に指定しても、一般保護例外が生成される。

MSR は、テスト機能、実行トレース、性能モニタリング、マシン・チェック・エラーの機能を制御する。『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、下巻』の付録 B「モデル固有レジスタ（MSR）」では、この命令で読み取ることができるすべての MSR とそれらのアドレスを一覧している。各プロセッサ・ファミリは、独自に一連の MSR を持っていることに注意すること。

この命令を使用する前に、MSR がサポートされている（EDX[5]=1）かどうかを判断するため、CPUID 命令を使用する必要がある。

IA-32 アーキテクチャにおける互換性

MSR および RDMSR 命令でそれらを読み取る機能は、インテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。インテル Pentium プロセッサより以前の IA-32 プロセッサでこの命令を実行すると、無効オペコード例外 #UD が生成される。

操作

```
EDX:EAX ← MSR[ECX];
```

影響を受けるフラグ

なし。

RDMSR—Read from Model Specific 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。
	ECX の値が予約されているかまたはインプリメントされていない MSR アドレスを指定している場合。

実アドレスモード例外

#GP	CX の値が予約されているまたはインプリメントされていない MSR アドレスを指定している場合。

仮想 8086 モード例外

#GP(0)	RDMSR 命令は仮想 8086 モードでは認識されない。

RDPMC—Read Performance-Monitoring Counters

オペコード	命令	説明
0F 33	RDPMC	ECX によって指定される性能モニタリング・カウンタを EDX:EAX に読み込む。

説明

この命令は、ECX レジスタに指定された 40 ビットの性能モニタリング・カウンタの内容をレジスタ EDX:EAX にロードする。EDX レジスタには、カウンタの上位 8 ビットがロードされ、EAX レジスタには、下位 32 ビットがロードされる。読み込まれるカウンタは、ECX レジスタに格納される符号なし整数で指定される。P6 ファミリ・プロセッサと MMX® テクノロジ Pentium® プロセッサには、2 個の性能モニタリング・カウンタ（0 および 1）があり、ECX レジスタにそれぞれ 0000H または 0001H を入れて指定する。インテル® Pentium® 4 プロセッサとインテル® Xeon™ プロセッサには 18 個のカウンタ（0 ～ 17）があり、それぞれ 0000H から 0011H で指定する。

インテル Pentium 4 プロセッサとインテル Xeon プロセッサではパフォーマンス・カウンタの「高速」（32 ビット）読み込みと「低速」（40 ビット）読み込みをサポートしているが、これは、ECX レジスタのビット 31 で選択される。ビット 31 がセットされている場合、RDPMC 命令は、選択されたパフォーマンス・カウンタの下位 32 ビットのみを読み込む。ビット 31 がクリアされている場合は、カウンタの 40 ビットがすべて読み込まれる。読み込まれた 32 ビット・カウンタは EAX レジスタに返され、EDX レジスタは 0 にセットされる。インテル Pentium 4 プロセッサまたはインテル Xeon プロセッサでは、全 40 ビットの読み込みよりも 32 ビットの読み込みの方が高速に実行される。

保護モードまたは仮想 8086 モードでは、レジスタ CR4 の性能モニタリング・カウンタ・イネーブル（PCE）フラグにより、RDPMC 命令の使用が次のように制限される。PCE フラグがセットされている場合、RDPMC 命令をすべての特権レベルで実行できる。フラグがクリアされている場合、この命令は特権レベル 0 でのみ実行できる（実アドレスモードでは、RDPMC 命令は常にイネーブルになる）。

特権レベル 0 で実行した場合、RDMSR 命令を使用して性能モニタリング・カウンタを読み取ることもできる。

性能モニタリング・カウンタは、デコードされた命令数、受け取った割り込み数、キャッシュ・ロードの回数などのイベントをカウントするようにプログラムできるイベントカウンタである。『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』の付録 A「性能モニタリング・イベント」では、インテル Pentium 4 プロセッサ、インテル Xeon プロセッサ、初期の IA-32 プロセッサでカウントできるイベントを一覧している。

RDPMC—Read Performance-Monitoring Counters（続き）

RDPMC 命令は、命令をシリアル化しない。すなわち、先行する命令によって発生したすべてのイベントが完了していること、または後続の命令によって発生するイベントが開始していないことを保証していない。正確なイベントカウントが必要な場合は、ソフトウェアは、RDPMC 命令の前、後、またはその両方で（CPUID 命令などの）シリアル化命令を挿入しなければならない。

インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサでは、バックツーバック高速読み込みの実行が単調になるという保証はない。バックツーバック読み込みの単調性を保証するためには、シリアル化命令を 2 つの RDPMC 命令間に配置する必要がある。

RDPMC 命令は、16 ビット・アドレス指定モードまたは仮想 8086 モードで実行することができるが、ECX レジスタの内容すべてを使用してカウンタを選択し、イベントカウントはすべての EAX レジスタおよび EDX レジスタにストアされる。

RDPMC 命令は、インテル® Pentium® Pro プロセッサおよび MMX® テクノロジを実装したインテル Pentium プロセッサで IA-32 アーキテクチャに導入された。初期のインテル Pentium プロセッサにも性能モニタリング・カウンタがあるが、RDMSR 命令で読み取らなければならない。

操作

```
(* P6 family processors and Pentium processor with MMX technology *)
IF (ECX=0 OR 1) AND ((CR4.PCE=1) OR (CPL=0) OR (CR0.PE=0))
    THEN
        EAX ← PMC(ECX)[31:0];
        EDX ← PMC(ECX)[39:32];
    ELSE (* ECX is not 0 or 1 or CR4.PCE is 0 and CPL is 1, 2, or 3 and CR0.PE is 1*)
        #GP(0); FI;
(* Pentium 4 and Intel Xeon processor *)
IF (ECX[30:0]=0 ... 17) AND ((CR4.PCE=1) OR (CPL=0) OR (CR0.PE=0))
    THEN IF ECX[31] = 0
        THEN
            EAX ← PMC(ECX[30:0])[31:0]; (* 40-bit read *);
            EDX ← PMC(ECX[30:0])[39:32];
        ELSE IF ECX[31] = 1
            THEN
                EAX ← PMC(ECX[30:0])[31:0]; (* 32-bit read *);
                EDX ← 0;
            FI;
        FI;
    ELSE (* ECX[30:0] is not 0...17 or CR4.PCE is 0 and CPL is 1, 2, or 3 and CR0.PE is 1 *)
        #GP(0); FI;
```

RDPMC—Read Performance-Monitoring Counters（続き）

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 でなく、CR4 レジスタの PCE フラグがクリアされている場合。 （P6 ファミリ・プロセッサおよび MMX テクノロジ Pentium プロセッサ）ECX レジスタの値が 0 または 1 でない場合。 （インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）ECX[30:0] の値が 0 ～ 17 の範囲外の場合。ECX レジスタの値が 0 または 1 でない場合。

実アドレスモード例外

#GP	（P6 ファミリ・プロセッサおよび MMX テクノロジ Pentium プロセッサ） ECX レジスタの値が 0 または 1 でない場合。 （インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）ECX[30:0] の値が 0 ～ 17 の範囲外の場合。

仮想 8086 モード例外

#GP(0)	CR4 レジスタの PCE フラグがクリアされている場合。 （P6 ファミリ・プロセッサおよび MMX テクノロジ Pentium プロセッサ）ECX レジスタの値が 0 または 1 でない場合。 （インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ）ECX[30:0] の値が 0 ～ 17 の範囲外の場合。

RDTSC—Read Time-Stamp Counter

オペコード	命令	説明
0F 31	RDTSC	タイムスタンプ・カウンタを EDX:EAX に読み込む。

説明

プロセッサのタイムスタンプ・カウンタの現在の値を EDX:EAX レジスタにロードする。タイムスタンプ・カウンタは、64 ビットの MSR に包含されている。MSR の上位 32 ビットが EDX レジスタにロードされ、下位 32 ビットが EAX レジスタにロードされる。プロセッサは、クロックサイクルごとにタイムスタンプ・カウンタ MSR を単調にインクリメントし、プロセッサがリセットされるとカウンタを 0 にリセットする。タイムスタンプ・カウンタの動作の詳細は、『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、下巻』の第 15 章の「タイムスタンプ・カウンタ」を参照のこと。

保護モードまたは仮想 8086 モードでは、CR4 レジスタのタイムスタンプ・ディスエーブル（TSD）フラグは、以下のように RDTSC 命令の使用を制限する。TSD フラグがクリアされていると、RDTSC 命令はどの特権レベルでも実行することができる。このフラグがセットされていると、命令は特権レベル 0 だけで実行することができる（実アドレスモードでは、RDTSC 命令は常にイネーブルである）。

特権レベル 0 で実行しているときは、RDMSR 命令を使用してタイムスタンプ・カウンタを読み取ることもできる。

RDTSC 命令は、シリアル化命令ではない。そのため、カウンタを読み取る前に、前のすべての命令が実行されるまで待つことをしない。同様に、読み取り操作が行われる前に、後続の命令が実行を開始している場合もある。

この命令は、インテル® Pentium® プロセッサで IA-32 アーキテクチャに導入された。

操作

```
IF (CR4.TSD=0) OR (CPL=0) OR (CR0.PE=0)
    THEN
        EDX:EAX ← TimeStampCounter;
    ELSE (* CR4.TSD is 1 and CPL is 1, 2, or 3 and CR0.PE is 1 *)
        #GP(0)
FI;
```

影響を受けるフラグ

なし。

RDTSC—Read Time-Stamp Counter（続き）

保護モード例外

#GP(0)	CR4 レジスタの TSD フラグがセットされていて、CPL が 0 より大きい場合。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	CR4 レジスタの TSD フラグがセットされた場合。

REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix

オペコード	命令	説明
F3 6C	REP INS *m8*, DX	(E)CX のバイトをポート DX から ES:[(E)DI] に入力する。
F3 6D	REP INS *m16*, DX	(E)CX のワードをポート DX から ES:[(E)DI] に入力する。
F3 6D	REP INS *m32*, DX	(E)CX のダブルワードをポート DX から ES:[(E)DI] に入力する。
F3 A4	REP MOVS *m8*, *m8*	(E)CX のバイトを DS:[(E)SI] から ES:[(E)DI] に転送する。
F3 A5	REP MOVS *m16*, *m16*	(E)CX のワードを DS:[(E)SI] から ES:[(E)DI] に転送する。
F3 A5	REP MOVS *m32*, *m32*	(E)CX のダブルワードを DS:[(E)SI] から ES:[(E)DI] に転送する。
F3 6E	REP OUTS DX, *r/m8*	(E)CX のバイトを DS:[(E)SI] からポート DX に出力する。
F3 6F	REP OUTS DX, *r/m16*	(E)CX のワードを DS:[(E)SI] からポート DX に出力する。
F3 6F	REP OUTS DX, *r/m32*	(E)CX のダブルワードを DS:[(E)SI] からポート DX に出力する。
F3 AC	REP LODS AL	(E)CX のバイトを DS:[(E)SI] から AL にロードする。
F3 AD	REP LODS AX	(E)CX のワードを DS:[(E)SI] から AX にロードする。
F3 AD	REP LODS EAX	(E)CXのダブルワードをDS:[(E)SI]からEAXにロードする。
F3 AA	REP STOS *m8*	ES:[(E)DI] にある (E)CX のバイトを AL で埋める。
F3 AB	REP STOS *m16*	ES:[(E)DI] にある (E)CX のワードを AX で埋める。
F3 AB	REP STOS *m32*	ES:[(E)DI] にある (E)CX のダブルワードを EAX で埋める。
F3 A6	REPE CMPS *m8*, *m8*	ES:[(E)DI] と DS:[(E)SI] にある一致していないバイトを探す。
F3 A7	REPE CMPS *m16*, *m16*	ES:[(E)DI] と DS:[(E)SI] にある一致していないワードを探す。
F3 A7	REPE CMPS *m32*, *m32*	ES:[(E)DI] と DS:[(E)SI] にある一致していないダブルワードを探す。
F3 AE	REPE SCAS *m8*	ES:[(E)DI] で始まる AL でないバイトを探す。
F3 AF	REPE SCAS *m16*	ES:[(E)DI] で始まる AX でないワードを探す。
F3 AF	REPE SCAS *m32*	ES:[(E)DI] で始まる EAX でないダブルワードを探す。
F2 A6	REPNE CMPS *m8*, *m8*	ES:[(E)DI] と DS:[(E)SI] にある一致しているバイトを探す。
F2 A7	REPNE CMPS *m16*, *m16*	ES:[(E)DI] と DS:[(E)SI] にある一致しているワードを探す。
F2 A7	REPNE CMPS *m32*, *m32*	ES:[(E)DI] と DS:[(E)SI] にある一致しているダブルワードを探す。
F2 AE	REPNE SCAS *m8*	ES:[(E)DI] で始まる AL を探す。
F2 AF	REPNE SCAS *m16*	ES:[(E)DI] で始まる AX を探す。
F2 AF	REPNE SCAS *m32*	ES:[(E)DI] で始まる EAX を探す。

REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix（続き）

説明

カウンタレジスタ（(E)CX）に指定された回数か、またはZFフラグの指定された条件が満たされなくなるまで、ストリング命令を繰り返す。REP（リピート）、REPE（等しい間はリピート）、REPNE（等しくない間はリピート）、REPZ（ゼロの間はリピート）、REPNZ（ゼロでない間はリピート）の各ニーモニックは、ストリング命令の1つに付加できるプリフィックスである。REPプリフィックスは、INS、OUTS、MOVS、LODS、およびSTOS命令に付加することができ、REPE、REPNE、REPZ、およびREPNZプリフィックスは、CMPSおよびSCAS命令に付加することができる。（REPZプリフィックスおよびREPNZプリフィックスは、それぞれREPEプリフィックスおよびREPNEプリフィックスのシノニム形式である。）非ストリング命令と共に使用すると、REPプリフィックスの行動は未定義である。

REPプリフィックスは、一度には1つのストリング命令だけに適用される。命令ブロックを繰り返すには、LOOP命令またはその他のループ構造体を使用する。

これらすべてのリピート・プリフィックスは、(E)CXレジスタのカウントが0にデクリメントされるまで、関連する命令を繰り返させる（下記の表を参照）（現在のアドレスサイズ属性が32である場合）。ECXレジスタがカウンタとして使用され、アドレスサイズ属性が16である場合は、CXレジスタが使用される。REPE、REPNE、REPZ、REPNZプリフィックスも、各リピート後にZFフラグの状態をチェックし、ZFフラグが指定された状態になければリピートループを終了する。両方の終了条件がテストされるときは、リピート終了の原因は、JECXZ命令での(E)CXレジスタのテスト、またはJZ、JNZ、JNE命令でのZFフラグのテストによって判断することができる。

リピート・プリフィックス	終了条件 1	終了条件 2
REP	ECX=0	なし
REPE/REPZ	ECX=0	ZF=0
REPNE/REPNZ	ECX=0	ZF=1

REPE/REPZおよびREPNE/REPNZプリフィックスを使用すると、CMPS命令およびSCAS命令の両方が、それらが行う比較の結果にしたがってZFフラグに影響を与えるので、ZFフラグを初期化する必要はない。

REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix（続き）

リピート・ストリング操作は、例外または割り込みによって中断されることがある。こうなっても、レジスタの状態は、例外ハンドラまたは割り込みハンドラからのリターン時にストリング操作を再開できるように保たれる。ソースレジスタおよびデスティネーション・レジスタは操作する次のストリング要素を指し、EIP レジスタはストリング命令を指し、ECX レジスタは命令の最後の正常なリピートの後に保持していた値をもつ。これによって、システムの割り込み応答時間に影響を与えずに、長いストリング操作を進めることができる。

PEPE または REPNE のプリフィックスの付いた CMPS 命令または SCAS 命令の実行中にフォルトが発生すると、EFLAGS の値は命令の実行前の状態にリストアされる。SCAS 命令および CMPS 命令は EFLAGS を入力として使用しないので、プロセッサはページフォルト・ハンドラの後に命令を再開することができる。

REP INS 命令および REP OUTS 命令は、注意して使用しなければならない。これらの命令が実行するレートを処理できない I/O ポートもある。

REP STOS 命令は、大きいメモリブロックを初期化する最も速い方法である。

操作

```
IF AddressSize = 16
    THEN
        use CX for CountReg;
    ELSE (* AddressSize = 32 *)
        use ECX for CountReg;
FI;
WHILE CountReg ≠ 0
    DO
        service pending interrupts (if any);
        execute associated string instruction;
        CountReg ← CountReg – 1;
        IF CountReg = 0
            THEN exit WHILE loop
        FI;
        IF (repeat prefix is REPZ or REPE) AND (ZF=0)
        OR (repeat prefix is REPNZ or REPNE) AND (ZF=1)
            THEN exit WHILE loop
        FI;
    OD;
```

影響を受けるフラグ

なし。ただし、CMPS 命令および SCAS 命令は、EFLAGS レジスタのステータス・フラグを設定する。

REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix（続き）

例外（すべての操作モード）

なし。ただし、リピート・プリフィックスが関係している命令によって例外が生成されることがある。

RET—Return from Procedure

オペコード	命令	説明
C3	RET	コール元プロシージャに near リターンする。
CB	RET	コール元プロシージャに far リターンする。
C2 *iw*	RET *imm16*	コール元プロシージャに near リターンし、*imm16* バイトをスタックからポップする。
CA *iw*	RET *imm16*	コール元プロシージャに far リターンし、*imm16* バイトをスタックからポップする。

説明

プログラム制御をスタックのトップにあるリターンアドレスに移す。通常、アドレスはCALL命令によってスタックに置かれ、リターンはCALL命令の後に続く命令に対して行われる。

オプションのソース・オペランドは、リターンアドレスがポップされた後にリリースされるスタックバイト数を指定し、デフォルトではなしである。このオペランドは、コール先プロシージャに渡されてもう必要ないパラメータをスタックからリリースするために使用することができる。これは、新しいプロシージャへの切り替えに使用されるCALL命令が新しいプロシージャのアクセスにゼロでないワードカウントのコールゲートを使用するときに使用しなければならない。ここで、RET命令のソース・オペランドは、コールゲートのワード・カウント・フィールドに指定されているのと同じバイト数を指定しなければならない。

RET命令を使用して、以下の異なる3つのタイプのリターンを実行することができる。

- near リターン ― 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）内にあるコール元プロシージャへのリターン。セグメント内リターンともいう。
- far リターン ― 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コール元プロシージャへのリターン。セグメント間リターンともいう。
- 特権レベル間 far リターン ― 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への far リターン。

特権レベル間リターンタイプは、保護モードでしか実行することができない。near、far、および特権レベル間の各リターンの詳細については、『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」の節を参照のこと。

RET—Return from Procedure（続き）

near リターンを実行すると、プロセッサは、リターン命令ポインタ（オフセット）をスタックのトップから EIP レジスタにポップし、新しい命令ポインタでのプログラム実行を開始する。CS レジスタは変更されない。

far リターンを実行すると、プロセッサは、リターン命令ポインタをスタックのトップから EIP レジスタにポップし、次にセグメント・セレクタをスタックのトップから CS レジスタにポップする。プロセッサは、その後、新しい命令ポインタにある新しいコード・セグメントでのプログラム実行を開始する。

特権レベル間 far リターンのメカニズムは、セグメント間リターンと同様であるが、プロセッサが戻されるコード・セグメントとスタック・セグメントの特権レベルとアクセス権を調べて、制御の転送を行うことができるかどうかを判断する点が異なる。DS、ES、FS、GS セグメント・レジスタは、新しい特権レベルではアクセスすることができないセグメントをそれらが参照している場合は、特権レベル間リターン中に RET 命令によってクリアされる。特権レベル間リターンではスタックスイッチも行われるので、ESP レジスタおよび SS レジスタがスタックからロードされる。

特権レベル間コール中にパラメータがコール先プロシージャに渡される場合は、RET 命令でオプションのソース・オペランドを使用して、リターン時にパラメータをリリースしなければならない。この場合は、パラメータは、コール先プロシージャのスタックとコール元プロシージャのスタック（すなわち、戻されるスタック）から両方ともリリースされる。

操作

```
(* Near return *)
IF instruction = near return
    THEN;
        IF OperandSize = 32
            THEN
                IF top 12 bytes of stack not within stack limits THEN #SS(0); FI;
                EIP ← Pop();
            ELSE (* OperandSize = 16 *)
                IF top 6 bytes of stack not within stack limits
                    THEN #SS(0)
                FI;
                tempEIP ← Pop();
                tempEIP ← tempEIP AND 0000FFFFH;
                IF tempEIP not within code segment limits THEN #GP(0); FI;
                EIP ← tempEIP;
        FI;
```

RET—Return from Procedure（続き）

```
        IF instruction has immediate operand
            THEN IF StackAddressSize=32
                THEN
                    ESP ← ESP + SRC; (* release parameters from stack *)
                ELSE (* StackAddressSize=16 *)
                    SP ← SP + SRC; (* release parameters from stack *)
            FI;
        FI;

    (* Real-address mode or virtual-8086 mode *)
    IF ((PE = 0) OR (PE = 1 AND VM = 1)) AND instruction = far return
        THEN;
            IF OperandSize = 32
                THEN
                    IF top 12 bytes of stack not within stack limits THEN #SS(0); FI;
                    EIP ← Pop();
                    CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
                ELSE (* OperandSize = 16 *)
                    IF top 6 bytes of stack not within stack limits THEN #SS(0); FI;
                    tempEIP ← Pop();
                    tempEIP ← tempEIP AND 0000FFFFH;
                    IF tempEIP not within code segment limits THEN #GP(0); FI;
                    EIP ← tempEIP;
                    CS ← Pop(); (* 16-bit pop *)
            FI;
        IF instruction has immediate operand
            THEN
                SP ← SP + (SRC AND FFFFH); (* release parameters from stack *)
        FI;
    FI;
    (* Protected mode, not virtual-8086 mode *)
    IF (PE = 1 AND VM = 0) AND instruction = far RET
        THEN
            IF OperandSize = 32
                THEN
                    IF second doubleword on stack is not within stack limits THEN #SS(0); FI;
                ELSE (* OperandSize = 16 *)
                    IF second word on stack is not within stack limits THEN #SS(0); FI;
            FI;
        IF return code segment selector is null THEN GP(0); FI;
        IF return code segment selector addrsses descriptor beyond diescriptor table limit
            THEN GP(selector; FI;
        Obtain descriptor to which return code segment selector points from descriptor table
        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:
```

RET—Return from Procedure（続き）

```
    IF return code segment selector RPL > CPL
        THEN GOTO RETURN-OUTER-PRIVILEGE-LEVEL;
        ELSE GOTO RETURN-TO-SAME-PRIVILEGE-LEVEL
    FI;
END;FI;

RETURN-SAME-PRIVILEGE-LEVEL:
    IF the return instruction pointer is not within ther return code segment limit
        THEN #GP(0);
    FI;
    IF OperandSize=32
        THEN
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            ESP ← ESP + SRC; (* release parameters from stack *)
        ELSE (* OperandSize=16 *)
            EIP ← Pop();
            EIP ← EIP AND 0000FFFFH;
            CS ← Pop(); (* 16-bit pop *)
            ESP ← ESP + SRC; (* release parameters from stack *)
    FI;

RETURN-OUTER-PRIVILEGE-LEVEL:
    IF top (16 + SRC) bytes of stack are not within stack limits (OperandSize=32)
        OR top (8 + SRC) bytes of stack are not within stack limits (OperandSize=16)
            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(selector); FI;
    Read segment descriptor pointed to by return segment selector;
    IF stack segment selector RPL ≠ RPL of the return code segment selector
        OR stack segment is not a writable data segment
        OR stack segment descriptor DPL ≠ RPL of the return code segment selector
            THEN #GP(selector); FI;
        IF stack segment not present THEN #SS(StackSegmentSelector); FI;
    IF the return instruction pointer is not within the return code segment limit
        THEN #GP(0); FI:
     CPL ← ReturnCodeSegmentSelector(RPL);
    IF OperandSize=32
        THEN
            EIP ← Pop();
            CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
             (* segment descriptor information also loaded *)
            CS(RPL) ← CPL;
            ESP ← ESP + SRC; (* release parameters from called procedure's stack *)
            tempESP ← Pop();
            tempSS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
```

RET—Return from Procedure（続き）

```
            (* segment descriptor information also loaded *)
            ESP ← tempESP;
            SS ← tempSS;
        ELSE (* OperandSize=16 *)
            EIP ← Pop();
            EIP ← EIP AND 0000FFFFH;
            CS ← Pop(); (* 16-bit pop; segment descriptor information also loaded *)
            CS(RPL) ← CPL;
            ESP ← ESP + SRC; (* release parameters from called procedure's stack *)
            tempESP ← Pop();
            tempSS ← Pop(); (* 16-bit pop; segment descriptor information also loaded *)
            (* segment descriptor information also loaded *)
            ESP ← tempESP;
            SS ← tempSS;
    FI;
    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; (* DPL in hidden part of segment register *)
                THEN (* segment register invalid *)
                    SegmentSelector ← 0; (* null segment selector *)
            FI;
        OD;
    For each of ES, FS, GS, and DS
    DO
        IF segment selector index is not within descriptor table limits
            OR segment descriptor indicates the segment is not a data or
                readable code segment
            OR if the segment is a data or non-conforming code segment and the segment
                descriptor's DPL < CPL or RPL of code segment's segment selector
                THEN
                    segment selector register ← null selector;
    OD;
    ESP ← ESP + SRC; (* release parameters from calling procedure's stack *)
```

影響を受けるフラグ

なし。

RET—Return from Procedure（続き）

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）	リターン・コード・セグメント・セレクタの RPL が CPL より小さい場合。
	リターンコードまたはスタック・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・テーブルの範囲内にない場合。
	リターン・コード・セグメント・ディスクリプタがコード・セグメントを指定していない場合。
	リターン・コード・セグメントが非コンフォーミングであり、セグメント・セレクタの DPL がコード・セグメントのセグメント・セレクタの RPL に等しくない場合。
	リターン・コード・セグメントがコンフォーミングであり、セグメント・セレクタの DPL がコード・セグメントのセグメント・セレクタの RPL より大きい場合。
	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 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。
	スタック・セグメント・ディスクリプタ 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。
#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。
	リターン・スタック・セグメントが存在しない場合。
#NP（セレクタ）	リターン・コード・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。
#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アクセスが行われた場合。

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。

RET—Return from Procedure（続き）

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。
#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。
#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アクセスが行われた場合。

ROL/ROR—Rotate

「RCL/RCR/ROL/ROR—Rotate」を参照のこと。

RSM—Resume from System Management Mode

オペコード	命令	説明
0F AA	RSM	割り込まれたプログラムの動作を再開する。

説明

プログラム制御をシステム管理モード（SMM）からプロセッサがSMM割り込みを受け取ったときに割り込まれたアプリケーション・プログラムまたはオペレーティング・システム・プロシージャに戻す。プロセッサの状態は、SMMに入ったときに作成されたダンプからリストアされる。プロセッサは、状態リストア中に無効な状態情報を検出すると、シャットダウン状態に入る。以下の無効な情報がシャットダウンを発生させる可能性がある。

- CR4の予約ビットのいずれかが1にセットされている。
- （PG=1とPE=0）や（NW=1とCD=0）などCR0のビットの不当な組み合わせ。
- （インテル® Pentium® プロセッサおよびIntel486™ プロセッサのみ）状態ダンプ・ベース・フィールドにストアされている値が32Kバイト・アライメント・アドレスでない。

モデル固有レジスタの内容は、SMMからのリターンによって影響を受けない。

SMMおよびRSM命令の動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13章「システム管理」を参照のこと。

操作

```
ReturnFromSSM;
ProcessorState ← Restore(SSMDump);
```

影響を受けるフラグ

すべて。

保護モード例外

#UD	プロセッサがSMMにないときにこの命令を実行しようとした場合。

実アドレスモード例外

#UD	プロセッサがSMMにないときにこの命令を実行しようとした場合。

仮想 8086 モード例外

#UD	プロセッサがSMMにないときにこの命令を実行しようとした場合。

RSQRTPS—Compute Reciprocals of Square Roots of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 52 /r	RSQRTPS *xmm1*, *xmm2/m128*	*xmm2/m128* のパックド単精度浮動小数点値の平方根の逆数の近似値を計算し、その結果を *xmm1* にストアする。

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の平方根の逆数の近似値を SIMD 計算し、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。

この近似値の相対誤差は以下のとおりである。

| 相対誤差 | ≤ $1.5 * 2^{-12}$

RSQRTPS 命令は、MXCSR レジスタの丸め制御ビットの影響を受けない。ソース値が 0.0 の場合、ソース値の符号の∞が返される。デノーマル・ソース値は、同じ符号の 0.0 として処理される。ソース値が –0.0 以外の負の値の場合、浮動小数点の未定義値が返される。ソース値が SNaN または QNaN の場合、QNaN またはソース QNaN に変換される SNaN が返される。

操作

```
DEST[31-0] ← APPROXIMATE(1.0/SQRT(SRC[31-0]));
DEST[63-32] ← APPROXIMATE(1.0/SQRT(SRC[63-32]));
DEST[95-64] ← APPROXIMATE(1.0/SQRT(SRC[95-64]));
DEST[127-96] ← APPROXIMATE(1.0/SQRT(SRC[127-96]));
```

同等のインテル® C/C++ コンパイラ組み込み関数

RSQRTPS　　__m128 _mm_rsqrt_ps(__m128 a)

SIMD 浮動小数点例外

なし。

RSQRTPS—Compute Reciprocals of Square Roots of 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。
#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。

RSQRTSS—Compute Reciprocal of Square Root of Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 52 /r	RSQRTSS *xmm1*, *xmm2/m32*	*xmm2/m32* の下位の単精度浮動小数点値の平方根の逆数の近似値を計算し、その結果を *xmm1* 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の平方根の逆数の近似値を計算し、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。

この近似値の相対誤差は以下のとおりである。

$|\text{相対誤差}| \leq 1.5 * 2^{-12}$

RSQRTPS 命令は、MXCSR レジスタの丸め制御ビットの影響を受けない。ソース値が 0.0 の場合、ソース値の符号の∞が返される。デノーマル・ソース値は、同じ符号の 0.0 として処理される。ソース値が –0.0 以外の負の値の場合、浮動小数点の未定義値が返される。ソース値が SNaN または QNaN の場合、QNaN またはソース QNaN に変換される SNaN が返される。

操作

```
DEST[31-0] ← APPROXIMATE(1.0/SQRT(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

RSQRTSS　　__m128 _mm_rsqrt_ss(__m128 a)

SIMD 浮動小数点例外

なし。

RSQRTSS—Compute Reciprocal of Square Root of 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)	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。

SAHF—Store AH into Flags

オペコード	命令	説明
9E	SAHF	9E　SF、ZF、AF、PF、および CF を AH から EFLAGS レジスタにロードする。

説明

EFLAGS レジスタの SF、ZF、AF、PF、CF フラグに AH レジスタの対応するビット（それぞれビット 7、6、4、2、0）からの値をロードする。レジスタ AH のビット 1、3、および 5 は無視される。EFLAGS レジスタの対応する予約ビット（1、3、5）は、以下の「操作」の項に示すように残る。

操作

```
EFLAGS(SF:ZF:0:AF:0:PF:1:CF) ← AH;
```

影響を受けるフラグ

SF、ZF、AF、PF、CF フラグに AH レジスタからの値がロードされる。EFLAGS レジスタのビット 1、3、5 は影響を受けず、それぞれ 1、0、0 の値が残る。

例外（すべての操作モード）

なし。

SAL/SAR/SHL/SHR—Shift

オペコード	命令	説明
D0 /4	SAL *r/m8*	*r/m8* に 2 を 1 回掛ける。
D2 /4	SAL *r/m8*, CL	*r/m8* に 2 を CL 回掛ける。
C0 /4 *ib*	SAL *r/m8*, *imm8*	*r/m8* に 2 を *imm8* 回掛ける。
D1 /4	SAL *r/m16*	*r/m16* に 2 を 1 回掛ける。
D3 /4	SAL *r/m16*, CL	*r/m16* に 2 を CL 回掛ける。
C1 /4 *ib*	SAL *r/m16*, *imm8*	*r/m16* に 2 を *imm8* 回掛ける。
D1 /4	SAL *r/m32*	*r/m32* に 2 を 1 回掛ける。
D3 /4	SAL *r/m32*, CL	*r/m32* に 2 を CL 回掛ける。
C1 /4 *ib*	SAL *r/m32*, *imm8*	*r/m32* に 2 を *imm8* 回掛ける。
D0 /7	SAR *r/m8*	*r/m8* を 2 で 1 回符号付き除算する。*
D2 /7	SAR *r/m8*, CL	*r/m8* を 2 で CL 回符号付き除算する。*
C0 /7 *ib*	SAR *r/m8*, *imm8*	*r/m8* を 2 で *imm8* 回符号付き除算する。*
D1 /7	SAR *r/m16*	*r/m16* を 2 で 1 回符号付き除算する。*
D3 /7	SAR *r/m16*, CL	*r/m16* を 2 で CL 回符号付き除算する。*
C1 /7 *ib*	SAR *r/m16*, *imm8*	*r/m16* を 2 で *imm8* 回符号付き除算する。*
D1 /7	SAR *r/m32*	*r/m32* を 2 で 1 回符号付き除算する。*
D3 /7	SAR *r/m32*, CL	*r/m32* を 2 で CL 回符号付き除算する。*
C1 /7 *ib*	SAR *r/m32*, *imm8*	*r/m32* を 2 で *imm8* 回符号付き除算する。*
D0 /4	SHL *r/m8*	*r/m8* に 2 を 1 回掛ける。
D2 /4	SHL *r/m8*, CL	*r/m8* に 2 を CL 回掛ける。
C0 /4 *ib*	SHL *r/m8*, *imm8*	*r/m8* に 2 を *imm8* 回掛ける。
D1 /4	SHL *r/m16*	*r/m16* に 2 を 1 回掛ける。
D3 /4	SHL *r/m16*, CL	*r/m16* に 2 を CL 回掛ける。
C1 /4 *ib*	SHL *r/m16*, *imm8*	*r/m16* に 2 を *imm8* 回掛ける。
D1 /4	SHL *r/m32*	*r/m32* に 2 を 1 回掛ける。
D3 /4	SHL *r/m32*, CL	*r/m32* に 2 を CL 回掛ける。
C1 /4 *ib*	SHL *r/m32*, *imm8*	*r/m32* に 2 を *imm8* 回掛ける。
D0 /5	SHR *r/m8*	*r/m8* を 2 で 1 回符号なし除算する。
D2 /5	SHR *r/m8*, CL	*r/m8* を 2 で CL 回符号なし除算する。
C0 /5 *ib*	SHR *r/m8*, *imm8*	*r/m8* を 2 で *imm8* 回符号なし除算する。
D1 /5	SHR *r/m16*	*r/m16* を 2 で 1 回符号なし除算する。
D3 /5	SHR *r/m16*, CL	*r/m16* を 2 で CL 回符号なし除算する。
C1 /5 *ib*	SHR *r/m16*, *imm8*	*r/m16* を 2 で *imm8* 回符号なし除算する。
D1 /5	SHR *r/m32*	*r/m32* を 2 で 1 回符号なし除算する。
D3 /5	SHR *r/m32*, CL	*r/m32* を 2 で CL 回符号なし除算する。
C1 /5 *ib*	SHR *r/m32*, *imm8*	*r/m32* を 2 で *imm8* 回符号なし除算する。

注：

* IDIV と同じ除算形式ではないことに注意する。負の無限大方向に丸められる。

SAL/SAR/SHL/SHR—Shift（続き）

説明

第 1 オペランド（デスティネーション・オペランド）のビットを第 2 オペランド（カウント・オペランド）に指定されたビット数だけ左または右にシフトする。デスティネーション・オペランドの境界を超えてシフトされるビットは、まず CF フラグにシフトされ、次に捨てられる。シフト操作の最後では、CF フラグには、デスティネーション・オペランドから最後にシフトされたビットが含まれる。

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。カウント・オペランドには、即値またはレジスタ CL を使用できる。カウントは、5 ビットにマスクされ、0 から 31 までのカウント範囲に制限される。1 のカウントに対しては、特別なオペコード・エンコーディングが備えられている。

左への算術シフト（SAL）命令および左への論理シフト（SHL）命令は、同じ操作を実行しする。すなわち、デスティネーション・オペランドのビットを左（上位ビット・ロケーションの方向）にシフトする。シフトカウントごとに、デスティネーション・オペランドの最上位ビットが CF フラグにシフトされ、最下位ビットはクリアされる（『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、上巻』の図 7-7. を参照）。

右への算術シフト（SAR）命令および右への論理シフト（SHR）命令は、デスティネーション・オペランドのビットを右（下位ビット・ロケーションの方向）にシフトする。シフトカウントごとに、デスティネーション・オペランドの最下位ビットが CF フラグにシフトされ、最上位ビットは、命令タイプに応じてセットまたはクリアされる。SHR 命令は、最上位ビットをクリアする（『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 7-8. を参照のこと）。SAR 命令は、デスティネーション・オペランドの元の値の符号（最上位ビット）に相当するように最上位ビットをセットまたはクリアする。実際上、SAR 命令は、空のビット位置のシフトされた値をシフトされていない値の符号で埋める（『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 7-9. を参照のこと）。

SAR 命令および SHR 命令を使用して、それぞれデスティネーション・オペランドに 2 の累乗による符号付きまたは符号なしの除算を行うことができる。例えば、SAR 命令を使用して符号付き整数を 1 ビット右にシフトすることは、値を 2 で割ることになる。

SAL/SAR/SHL/SHR—Shift（続き）

SAR 命令を使用して除算演算を実行しても、IDIV 命令と同じ結果は生成されない。IDIV 命令の商はゼロ方向に丸められるのに対し、SAR 命令の「商」は負の無限大方向に丸められる。この違いは、負の数に対してだけ明白である。例えば、IDIV 命令を使用して –9 を 4 で除算すると、結果は –2 であり、剰余は –1 である。SAR 命令を使用して –9 を右に 2 ビットだけシフトすると、結果は –3 であり、「剰余」は +3 である。ただし、SAR 命令は、剰余の最上位ビットだけを（CF フラグに）ストアする。

OF フラグは、1 ビット・シフトの場合だけに影響を受ける。左シフトでは、結果の最上位ビットが CF フラグと同じである（すなわち、元のオペランドの最上位 2 ビットが同じである）場合は、OF フラグは 0 にセットされ、そうでない場合は 1 にセットされる。SAR 命令では、OF フラグはすべての 1 ビット・シフトでクリアされる。SHR 命令では、OF フラグは元のオペランドの最上位ビットに設定される。

IA-32 アーキテクチャにおける互換性

8086 は、シフトカウントをマスクしない。しかし、（インテル® 286 プロセッサで始まる）その他のすべての IA-32 プロセッサは、シフトカウントを 5 ビットにマスクするので、最大カウントは 31 になる。このマスク設定は、（仮想 8086 モードを含めた）すべての動作モードで行われ、命令の最大実行時間を減少させる。

操作

```
tempCOUNT ← (COUNT AND 1FH);
tempDEST ← DEST;
WHILE (tempCOUNT ≠ 0)
DO
    IF instruction is SAL or SHL
        THEN
            CF ← MSB(DEST);
        ELSE (* instruction is SAR or SHR *)
            CF ← LSB(DEST);
    FI;
    IF instruction is SAL or SHL
        THEN
            DEST ← DEST ∗ 2;
        ELSE
            IF instruction is SAR
                THEN
                    DEST ← DEST / 2 (*Signed divide, rounding toward negative infinity*);
                ELSE (* instruction is SHR *)
                    DEST ← DEST / 2 ; (* Unsigned divide *);
            FI;
    FI;
    tempCOUNT ← tempCOUNT – 1;
OD;
```

SAL/SAR/SHL/SHR—Shift（続き）

```
(* Determine overflow for the various instructions *)
IF (COUNT and 1FH) = 1
    THEN
        IF instruction is SAL or SHL
            THEN
                OF ← MSB(DEST) XOR CF;
            ELSE
                IF instruction is SAR
                    THEN
                        OF ← 0;
                    ELSE (* instruction is SHR *)
                        OF ← MSB(tempDEST);
                FI;
        FI;
    ELSE IF (COUNT AND 1FH) = 0
        THEN
            All flags remain unchanged;
        ELSE (* COUNT neither 1 or 0 *)
            OF ← undefined;
    FI;
FI;
```

影響を受けるフラグ

CF フラグは、デスティネーション・オペランドから最後にシフトされたビットの値をもつ。このフラグは、カウントがデスティネーション・オペランドの（ビットでの）サイズ以上である SHL 命令と SHR 命令では未定義である。OF フラグは、1 ビット・シフトの場合だけに影響を受け（上記の「説明」の項を参照）、その他の場合は未定義である。SF、ZF、PF フラグは、結果にしたがって設定される。カウントが 0 である場合は、フラグは影響を受けない。カウントがゼロでない場合は、AF フラグは未定義である。

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。

SAL/SAR/SHL/SHR—Shift（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。

SBB—Integer Subtraction with Borrow

オペコード	命令	説明
1C *ib*	SBB AL, *imm8*	AL から *imm8* をボローありで引く。
1D *iw*	SBB AX, *imm16*	AX から *imm16* をボローありで引く。
1D *id*	SBB EAX, *imm32*	EAX から *imm32* をボローありで引く。
80 /3 *ib*	SBB *r/m8*, *imm8*	*r/m8* から *imm8* をボローありで引く。
81 /3 *iw*	SBB *r/m16*, *imm16*	*r/m16* から *imm16* をボローありで引く。
81 /3 *id*	SBB *r/m32*, *imm32*	*r/m32* から *imm32* をボローありで引く。
83 /3 *ib*	SBB *r/m16*, *imm8*	*r/m16* から符号拡張された *imm8* をボローありで引く。
83 /3 *ib*	SBB *r/m32*, *imm8*	*r/m32* から符号拡張された *imm8* をボローありで引く。
18 /*r*	SBB *r/m8*, *r8*	*r/m8* から *r8* をボローありで引く。
19 /*r*	SBB *r/m16*, *r16*	*r/m16* から *r16* をボローありで引く。
19 /*r*	SBB *r/m32*, *r32*	*r/m32* から *r32* をボローありで引く。
1A /*r*	SBB *r8, r/m8*	*r8* から *r/m8* をボローありで引く。
1B /*r*	SBB *r16*, *r/m16*	*r16* から *r/m16* をボローありで引く。
1B /*r*	SBB *r32*, *r/m32*	*r32* から *r/m32* をボローありで引く。

説明

ソース・オペランド（第2オペランド）とキャリー（CF）フラグを加算し、結果をデスティネーション・オペランド（第1オペランド）から引く。減算の結果は、デスティネーション・オペランドにストアされる。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つの命令で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、前の減算からのボローを表す。

即値をオペランドとして使用すると、デスティネーション・オペランドのフォーマットの長さまで符号拡張される。

SBB 命令は、符号付きオペランドと符号なしオペランドとを区別しない。その代わりに、プロセッサは、両方のデータ型の結果を評価し、OF フラグと CF フラグを設定して、それぞれ符号付きの結果または符号なしの結果のボローを示す。SF フラグは、符号付き結果の符号を示す。

通常、SBB 命令は、SUB 命令の後に SBB 命令が続く複数バイトまたは複数ワードの除算の一部として実行される。

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。

SBB—Integer Subtraction with Borrow（続き）

操作

DEST ← DEST – (SRC + CF);

影響を受けるフラグ

OF、SF、ZF、AF、PF、CF フラグが結果にしたがって設定される。

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。

SCAS/SCASB/SCASW/SCASD—Scan String

オペコード	命令	説明
AE	SCAS m8	AL を ES:(E)DI のバイトと比較し、ステータス・フラグを設定する。
AF	SCAS m16	AX を ES:(E)DI のワードと比較し、ステータス・フラグを設定する。
AF	SCAS m32	EAX を ES:(E)DI のダブルワードと比較し、ステータス・フラグを設定する。
AE	SCASB	AL を ES:(E)DI のバイトと比較し、ステータス・フラグを設定する。
AF	SCASW	AX を ES:(E)DI のワードと比較し、ステータス・フラグを設定する。
AF	SCASD	EAX を ES:(E)DI のダブルワードと比較し、ステータス・フラグを設定する。

説明

メモリ・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、AL、AX、または EAX レジスタの値と比較し、結果にしたがって EFLAGS レジスタのステータス・フラグを設定する。メモリ・オペランド・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 ES:EDI レジスタまたは 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・オーバライド・プリフィックスでオーバ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。（SCAS ニーモニックで指定される）明示オペランド形式では、メモリ・オペランドを明示的に指定することができる。この場合、メモリ・オペランドは、オペランド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。レジスタ・オペランドは、この場合にはメモリ・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（バイト比較では AL レジスタ、ワード比較では AX レジスタ、ダブルワード比較では EAX レジスタ）。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、メモリ・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、ストリング比較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。

オペランドなし形式は、SCAS 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、ES:(E)DI がメモリ・オペランドであると想定され、AL、AX、または EAX レジスタがレジスタ・オペランドであると想定される。2 つのオペランドのサイズは、SCASB（バイト比較）、SCASW（ワード比較）、または SCASD（ダブルワード比較）の各ニーモニックで選択される。

SCAS/SCASB/SCASW/SCASD—Scan String（続き）

比較後、(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)DI レジスタはインクリメントされる。DF フラグが 1 である場合は、(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。

SCAS、SCASB、SCASW、SCASD 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。しかし通常は、これらの命令は、次の比較が行われる前にステータス・フラグの設定に基づいてある処置を行うループ構造体で使用されることの方が多い。REP 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 cmparison)
    THEN
        temp ← AL – SRC;
        SetStatusFlags(temp);
            THEN IF DF = 0
                THEN (E)DI ← (E)DI + 1;
                ELSE (E)DI ← (E)DI – 1;
            FI;
    ELSE IF (word comparison)
        THEN
            temp ← AX – SRC;
            SetStatusFlags(temp)
                THEN IF DF = 0
                    THEN (E)DI ← (E)DI + 2;
                    ELSE (E)DI ← (E)DI – 2;
                FI;
        ELSE (* doubleword comparison *)
            temp ← EAX – SRC;
            SetStatusFlags(temp)
                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

OF、SF、ZF、AF、PF、CF フラグが比較の一時的な結果にしたがって設定される。

SCAS/SCASB/SCASW/SCASD—Scan 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 ES セグメントの範囲外の場合。 ES レジスタの内容がヌル・セグメント・セレクタの場合。 ES レジスタに不当なメモリ・オペランド実効アドレスが与えられている場合。
#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。

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。

SET*cc*—Set Byte on Condition

オペコード	命令	説明
0F 97	SETA *r/m8*	より上（CF=0 および ZF=0）の場合バイトを設定する。
0F 93	SETAE *r/m8*	より上か等しい（CF=0）場合バイトを設定する。
0F 92	SETB *r/m8*	より下（CF=1）の場合バイトを設定する。
0F 96	SETBE *r/m8*	より下か等しい（CF=1 または ZF=1）場合バイトを設定する。
0F 92	SETC *r/m8*	キャリーがある（CF=1）場合設定する。
0F 94	SETE *r/m8*	等しい（ZF=1）場合バイトを設定する。
0F 9F	SETG *r/m8*	より大きい（ZF=0 および SF=OF）場合バイトを設定する。
0F 9D	SETGE *r/m8*	より大きいか等しい（SF=OF）場合バイトを設定する。
0F 9C	SETL *r/m8*	より小さい（SF<>OF）場合バイトを設定する。
0F 9E	SETLE *r/m8*	より小さいか等しい（ZF=1 または SF<>OF）場合バイトを設定する。
0F 96	SETNA *r/m8*	より上でない（CF=1 または ZF=1）場合バイトを設定する。
0F 92	SETNAE *r/m8*	より上でなく等しくない（CF=1）場合バイトを設定する。
0F 93	SETNB *r/m8*	より下でない（CF=0）場合バイトを設定する。
0F 97	SETNBE *r/m8*	より下でなく等しくない（CF=0 および ZF=0）場合バイトを設定する。
0F 93	SETNC *r/m8*	キャリーがない（CF=0）場合バイトを設定する。
0F 95	SETNE *r/m8*	等しくない（ZF=0）場合バイトを設定する。
0F 9E	SETNG *r/m8*	より大きくない(ZF=1またはSF<>OF)場合バイトを設定する。
0F 9C	SETNGE *r/m8*	より大きくなく等しくない（SF<>OF）場合設定する。
0F 9D	SETNL *r/m8*	より小さくない（SF=OF）場合バイトを設定する。
0F 9F	SETNLE *r/m8*	より小さくなく等しくない（ZF=0 および SF=OF）場合バイトを設定する。
0F 91	SETNO *r/m8*	オーバーフローがない（OF=0）場合バイトを設定する。
0F 9B	SETNP *r/m8*	パリティがない（PF=0）場合バイトを設定する。
0F 99	SETNS *r/m8*	符号がない（SF=0）場合バイトを設定する。
0F 95	SETNZ *r/m8*	ゼロでない（ZF=0）場合バイトを設定する。
0F 90	SETO *r/m8*	オーバーフローがある（OF=1）場合バイトを設定する。
0F 9A	SETP *r/m8*	パリティがある（PF=1）場合バイトを設定する。
0F 9A	SETPE *r/m8*	パリティが偶数（PF=1）の場合バイトを設定する。
0F 9B	SETPO *r/m8*	パリティが奇数（PF=0）の場合バイトを設定する。
0F 98	SETS *r/m8*	符号がある（SF=1）場合バイトを設定する。
0F 94	SETZ *r/m8*	ゼロ（ZF=1）の場合バイトを設定する。

SET*cc*—Set Byte on Condition（続き）

説明

デスティネーション・オペランドをEFLAGSレジスタのステータス・フラグ（CF、SF、OF、ZF、PF）の設定にしたがって0または1に設定する。デスティネーション・オペランドは、バイトレジスタまたはメモリ内のバイトを指す。条件コード・サフィックス（*cc*）は、テストされる条件を示している。

「より上」および「より下」という表現は、CFフラグに関連付けられ、2つの符号なし整数値間の関係をいっている。「より大きい」および「より小さい」という表現は、SFおよびOFフラグに関連付けられ、2つの符号付き整数値間の関係をいっている。

SET*cc*命令のオペコードの多くには、代替ニーモニックがある。例えば、SETG（より大きい場合バイトを設定）とSETNLE（より小さくなく等しくない場合設定）は、同じオペコードをもち、同じ条件、すなわち、ZFが0に等しく、SFがOFに等しいことをテストする。これらの代替ニーモニックは、コードをよりわかりやすくするために提供されている。『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』の付録B「EFLAGS条件コード」では、さまざまなテスト条件の代替ニーモニックを示している。

一部の言語では、すべてのビットをセットした整数として論理1を表す。この表現は、SET*cc*命令に論理的に反対の条件を選択し、結果をデクリメントすると得ることができる。例えば、オーバーフローがあるかテストするには、SETNO命令を使用し、次に結果をデクリメントする。

操作

```
IF condition
    THEN DEST ← 1
    ELSE DEST ← 0;
FI;
```

影響を受けるフラグ

なし。

SET*cc*—Set Byte on Condition（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。

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。

SFENCE—Store Fence

オペコード	命令	説明
0F AE /7	SFENCE	ストア操作をシリアル化する。

説明

SFENCE 命令より前に発行されたすべてのストアメモリ命令に対して、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 SFENCE 命令に先行するすべてのストア命令が、SFENCE 命令に後続する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。SFENCE 命令は、ストア命令、他の SFENCE 命令、MFENCE 命令、任意のシリアル化命令（CPUID 命令など）に対して順序付けされる。SFENCE 命令は、ロード命令や LFENCE 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、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するかは、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。SFENCE 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。

操作

```
Wait_On_Following_Stores_Until(preceding_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_sfence(void)

保護モード例外

なし。

実アドレスモード例外

なし。

仮想 8086 モード例外

なし。

SGDT—Store Global Descriptor Table Register

オペコード	命令	説明
0F 01 /0	SGDT *m*	GDTR を *m* にストアする。

説明

グローバル・ディスクリプタ・テーブル・レジスタ（GDTR）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、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、レジスタの 16 ビットのリミット・フィールドがメモリ・ロケーションの下位 2 バイトにストアされ、32 ビットのベースアドレスが上位 4 バイトにストアされる。オペランド・サイズ属性が 16 ビットである場合は、範囲が下位 2 バイトにストアされ、24 ビットのベースアドレスが 3 ～ 5 バイト目にストアされ、6 バイト目は 0 で埋められる。

SGDT 命令は、オペレーティング・システム・ソフトウェアだけに有用であるが、例外を生成せずにアプリケーション・プログラムで使用することができる。

GDTRおよびIDTRのローディングに関する詳細については、本章の「LGDT/LIDT—Load Global/Interrupt Descriptor Table Register」 を参照のこと。

IA-32 アーキテクチャにおける互換性

SGDT 命令の 16 ビット形式は、上位 8 ビットが参照されない場合に Intel 286 プロセッサと互換性がある。Intel 286 プロセッサはこれらのビットを 1 で埋め、インテル® Pentium® 4 プロセッサ、インテル® Xeon™ プロセッサ、P6 ファミリ・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、および Intel386™ プロセッサはこれらのビットを 0 で埋める。

操作

```
IF instruction is SGDT
        IF OperandSize = 16
            THEN
                DEST[0:15] ← GDTR(Limit);
                DEST[16:39] ← GDTR(Base); (* 24 bits of base address loaded; *)
                DEST[40:47] ← 0;
            ELSE (* 32-bit Operand Size *)
                DEST[0:15] ← GDTR(Limit);
                DEST[16:47] ← GDTR(Base); (* full 32-bit base address loaded *)
        FI;
FI;
```

SGDT/SIDT—Store Global/Interrupt Descriptor Table Register（続き）

影響を受けるフラグ

なし。

保護モード例外

#UD	デスティネーション・オペランドがレジスタである場合。
#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、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)	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	デスティネーション・オペランドがレジスタである場合。
#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

#UD	デスティネーション・オペランドがレジスタである場合。
#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を行った場合。

SHL/SHR—Shift Instructions

「SAL/SAR/SHL/SHR—Shift」を参照のこと。

SHLD—Double Precision Shift Left

オペコード	命令	説明
0F A4	SHLD *r/m16*, *r16*, *imm8*	*r/m16* を左に *imm8* 位置シフトし、*r16* からのビットを右からシフトインする。
0F A5	SHLD *r/m16*, *r16*, CL	*r/m16* を左に CL 位置シフトし、*r16* からのビットを右からシフトインする。
0F A4	SHLD *r/m32*, *r32*, *imm8*	*r/m32* を左に *imm8* 位置シフトし、*r32* からのビットを右からシフトインする。
0F A5	SHLD *r/m32*, *r32*, CL	*r/m32* を左に CL 位置シフトし、*r32* からのビットを右からシフトインする。

説明

第 1 オペランド（デスティネーション・オペランド）を第 3 オペランド（カウント・オペランド）で指定されたビット数だけ左にシフトする。第 2 オペランド（ソース・オペランド）は、（デスティネーション・オペランドのビット 0 で始まって）右からシフトインするビットを指定する。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・オペランドは、レジスタである。カウント・オペランドは符号なし整数で、即値バイトまたは CL レジスタの内容である。カウントのビット 0 ～ 4 だけが使用され、カウントが 0 ～ 31 の値にマスクされる。カウントがオペランド・サイズより大きい場合は、デスティネーション・オペランドの結果は未定義である。

カウントが 1 以上である場合は、CF フラグにはデスティネーション・オペランドから最後にシフトされたビットが入る。1 ビット・シフトでは、OF フラグは、符号変更が発生するとセットされ、そうでなければクリアされる。カウント・オペランドが 0 である場合は、フラグは影響を受けない。

SHLD 命令は、64 ビット以上の多重精度シフトに有用である。

操作

```
COUNT ← COUNT MOD 32;
SIZE ← OperandSize
IF COUNT = 0
    THEN
        no operation
    ELSE
        IF COUNT > SIZE
            THEN (* Bad parameters *)
                DEST is undefined;
                CF, OF, SF, ZF, AF, PF are undefined;
            ELSE (* Perform the shift *)
                CF ← BIT[DEST, SIZE – COUNT];
                (* Last bit shifted out on exit *)
                FOR i ← SIZE – 1 DOWNTO COUNT
```

SHLD—Double Precision Shift Left（続き）

```
                DO
                    Bit(DEST, i) ← Bit(DEST, i – COUNT);
                OD;
                FOR i ← COUNT – 1 DOWNTO 0
                DO
                    BIT[DEST, i] ← BIT[SRC, i – COUNT + SIZE];
                OD;
        FI;
FI;
```

影響を受けるフラグ

カウントが1以上である場合は、CFフラグにはデスティネーション・オペランドから最後にシフトされたビットが入り、SF、ZF、PFフラグは結果の値にしたがって設定される。1ビット・シフトでは、OFフラグは、符号変更が発生するとセットされ、そうでなければクリアされる。1ビットより大きいシフトでは、OFフラグは未定義である。シフトが行われると、AFフラグは未定義である。カウント・オペランドが0である場合は、フラグは影響を受けない。カウントがオペランド・サイズより大きい場合は、フラグは未定義である。

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。

SHLD—Double Precision Shift Left（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。

SHRD—Double Precision Shift Right

オペコード	命令	説明
0F AC	SHRD *r/m16*, *r16*, *imm8*	*r/m16* を右に *imm8* 位置シフトし、*r16* からのビットを左からシフトインする。
0F AD	SHRD *r/m16*, *r16*, CL	*r/m16* を右に CL 位置シフトし、*r16* からのビットを左からシフトインする。
0F AC	SHRD *r/m32*, *r32*, *mm8*	*r/m32* を右に *imm8* 位置シフトし、*r32* からのビットを左からシフトインする。
0F AD	SHRD *r/m32*, *r32*, CL	*r/m32* を右に CL 位置シフトし、*r32* からのビットを左からシフトインする。

説明

第 1 オペランド（デスティネーション・オペランド）を第 3 オペランド（カウント・オペランド）で指定されたビット数だけ右にシフトする。第 2 オペランド（ソース・オペランド）は、（デスティネーション・オペランドの最上位ビットで始まって）左からシフトインするビットを指定する。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・オペランドは、レジスタである。カウント・オペランドは符号なし整数で、即値バイトまたは CL レジスタの内容である。カウントのビット 0 ～ 4 だけが使用され、カウントが 0 ～ 31 の値にマスクされる。カウントがオペランド・サイズより大きい場合は、デスティネーション・オペランドの結果は未定義である。

カウントが 1 以上である場合は、CF フラグにはデスティネーション・オペランドから最後にシフトされたビットが入る。1 ビット・シフトでは、OF フラグは、符号変更が発生するとセットされ、そうでなければクリアされる。カウント・オペランドが 0 である場合は、フラグは影響を受けない。

SHRD 命令は、64 ビット以上の多重精度シフトに有用である。

操作

```
COUNT ← COUNT MOD 32;
SIZE ← OperandSize
IF COUNT = 0
    THEN
        no operation
    ELSE
        IF COUNT > SIZE
            THEN (* Bad parameters *)
                DEST is undefined;
                CF, OF, SF, ZF, AF, PF are undefined;
            ELSE (* Perform the shift *)
                CF ← BIT[DEST, COUNT – 1]; (* last bit shifted out on exit *)
                FOR i ← 0 TO SIZE – 1 – COUNT
                    DO
```

SHRD—Double Precision Shift Right（続き）

```
                BIT[DEST, i] ← BIT[DEST, i + COUNT];
            OD;
        FOR i ← SIZE – COUNT TO SIZE – 1
            DO
                BIT[DEST,i] ← BIT[SRC, i + COUNT – SIZE];
            OD;
    FI;
FI;
```

影響を受けるフラグ

カウントが 1 以上である場合は、CF フラグにはデスティネーション・オペランドから最後にシフトされたビットが入り、SF、ZF、PF フラグは結果の値にしたがって設定される。1 ビット・シフトでは、OF フラグは、符号変更が発生するとセットされ、そうでなければクリアされる。1 ビットより大きいシフトでは、OF フラグは未定義である。シフトが行われると、AF フラグは未定義である。カウント・オペランドが 0 である場合は、フラグは影響を受けない。カウントがオペランド・サイズより大きい場合は、フラグは未定義である。

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。

SHRD—Double Precision Shift Right（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。

SHUFPD—Shuffle Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F C6 /r ib	SHUFPD *xmm1*, *xmm2/m128*, *imm8*	*imm8* によって選択された、*xmm1* および *xmm1/m128* 内のパックド倍精度浮動小数点値をシャッフルして、*xmm1* に格納する。

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うち 1 つを、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移動し、ソース・オペランド内の 2 つのパックド倍精度浮動小数点値のうち 1 つを、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移動する（図 4-12. を参照）。セレクト・オペランド（第 3 オペランド）によって、どちらの値がデスティネーション・オペランドに転送されるかが決まる。

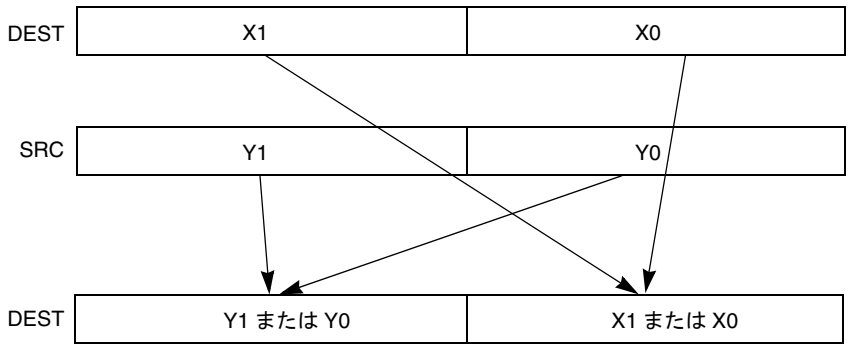


図 4-12. SHUFPD のシャッフル操作

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。セレクト・オペランドは 8 ビットの即値である。セレクト・オペランドのビット 0 は、デスティネーション・オペランドから結果に転送される値を選択する（値が 0 の場合は下位クワッドワードが転送され、1 の場合は上位クワッドワードが転送される）。ビット 1 は、ソース・オペランドから結果に転送される値を選択する。選択オペランドのビット 2 ～ 7 は予約されていて、0 にセットしなければならない。

SHUFPD—Shuffle Packed Double-Precision Floating-Point Values（続き）

操作

```
IF SELECT[0] = 0
    THEN DEST[63-0] ← DEST[63-0];
    ELSE DEST[63-0] ← DEST[127-64]; FI;
IF SELECT[1] = 0
    THEN DEST[127-64] ← SRC[63-0];
    ELSE DEST[127-64] ← SRC[127-64]; FI;
```

同等のインテル® C/C++ コンパイラ組み込み関数

SHUFPD　　__m128d _mm_shuffle_pd(__m128d a, __m128d b, unsigned int imm8)

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。

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。

SHUFPD—Shuffle 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。

SHUFPS—Shuffle Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F C6 /r ib	SHUFPS *xmm1*, *xmm2/m128*, *imm8*	*imm8* によって選択された、*xmm1* および *xmm1/m128* 内のパックド単精度浮動小数点値をシャッフルして、*xmm1* に格納する。

説明

デスティネーション・オペランド（第 1 オペランド）の 4 つのパックド単精度浮動小数点値のうち 2 つを、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移動し、ソース・オペランド（第 2 オペランド）の 4 つの単精度浮動小数点値のうち 2 つを、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移動する（図 4-13. を参照）。セレクト・オペランド（第 3 オペランド）によって、どの値がデスティネーション・オペランドに転送されるかが決まる。

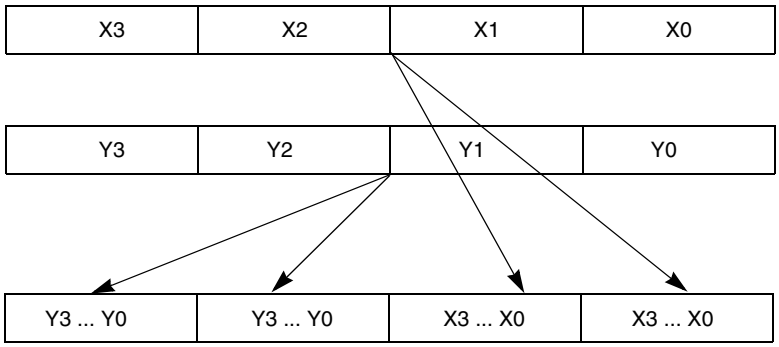


図 4-13. SHUFPS 命令の動作

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。セレクト・オペランドは 8 ビットの即値である。即値のビット 0 とビット 1 は、デスティネーション・オペランドから結果の最下位のダブルワードに転送される値を選択する。ビット 2 とビット 3 は、デスティネーション・オペランドから結果の第 2 ダブルワードに転送される値を選択する。ビット 4 とビット 5 は、ソース・オペランドから結果の第 3 ダブルワードに転送される値を選択する。ビット 6 とビット 7 は、ソース・オペランドから結果の最上位のダブルワードに転送される値を選択する。

SHUFPS—Shuffle Packed Single-Precision Floating-Point Values （続き）

操作

```
CASE (SELECT[1-0]) OF
    0:  DEST[31-0] ← DEST[31-0];
    1:  DEST[31-0] ← DEST[63-32];
    2:  DEST[31-0] ← DEST[95-64];
    3:  DEST[31-0] ← DEST[127-96];
ESAC;
CASE (SELECT[3-2]) OF
    0:  DEST[63-32] ← DEST[31-0];
    1:  DEST[63-32] ← DEST[63-32];
    2:  DEST[63-32] ← DEST[95-64];
    3:  DEST[63-32] ← DEST[127-96];
ESAC;
CASE (SELECT[5-4]) OF
    0:  DEST[95-64] ← SRC[31-0];
    1:  DEST[95-64] ← SRC[63-32];
    2:  DEST[95-64] ← SRC[95-64];
    3:  DEST[95-64] ← SRC[127-96];
ESAC;
CASE (SELECT[7-6]) OF
    0:  DEST[127-96] ← SRC[31-0];
    1:  DEST[127-96] ← SRC[63-32];
    2:  DEST[127-96] ← SRC[95-64];
    3:  DEST[127-96] ← SRC[127-96];
ESAC;
```

同等のインテル® C/C++ コンパイラ組み込み関数

SHUFPS　__m128 _mm_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)

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 ビットがセットされた場合。

SHUFPS—Shuffle 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。

SIDT—Store Interrupt Descriptor Table Register

オペコード	命令	説明
0F 01 /1	SIDT *m*	IDTR を *m* にストアする。

説明

割り込みディスクリプタ・テーブル・レジスタ（IDTR）の内容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、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、レジスタの 16 ビットのリミット・フィールドがメモリ・ロケーションの下位 2 バイトにストアされ、32 ビットのベースアドレスが上位 4 バイトにストアされる。オペランド・サイズ属性が 16 ビットである場合は、範囲が下位 2 バイトにストアされ、24 ビットのベースアドレスが 3 ～ 5 バイト目にストアされ、6 バイト目は 0 で埋められる。

SIDT は、オペレーティング・システム・ソフトウェアだけに有用であるが、例外を生成せずにアプリケーション・プログラムで使用することができる。

GDTR および IDTR のローディングに関する詳細については、本章の「LGDT/LIDT—Load Global/Interrupt Descriptor Table Register」を参照のこと。

IA-32 アーキテクチャにおける互換性

SIDT の 16 ビット形式は、上位 8 ビットが参照されない場合にインテル® 286 プロセッサと互換性がある。インテル 286 プロセッサはこれらのビットを 1 で埋め、インテル® Pentium® 4 プロセッサ、インテル® Xeon ™プロセッサ、P6 ファミリ・プロセッサ、Intel486 ™プロセッサ、Intel386™ プロセッサはこれらのビットを 0 で埋める。

操作

```
IF instruction is SIDT
    THEN
        IF OperandSize = 16
            THEN
                DEST[0:15] ← IDTR(Limit);
                DEST[16:39] ← IDTR(Base); (* 24 bits of base address loaded; *)
                DEST[40:47] ← 0;
            ELSE (* 32-bit Operand Size *)
                DEST[0:15] ← IDTR(Limit);
                DEST[16:47] ← IDTR(Base); (* full 32-bit base address loaded *)
        FI;
FI;
```

影響を受けるフラグ

なし。

SIDT—Store Interrupt Descriptor Table Register（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。

SLDT—Store Local Descriptor Table Register

オペコード	命令	説明
0F 00 /0	SLDT *r/m16*	セグメント・セレクタを LDTR から *r/m16* にストアする。
0F 00 /0	SLDT *r32*	セグメント・セレクタを LDTR から *r32* の下位 16 ビットにストアする。

説明

セグメント・セレクタをローカル・ディスクリプタ・テーブル・レジスタ（LDTR）から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、汎用レジスタまたはメモリ・ロケーションを使用できる。この命令でストアされるセグメント・セレクタは、現在の LDT の（GDT にある）セグメント・ディスクリプタを指す。この命令は、保護モードでしか実行することができない。

デスティネーション・オペランドが 32 ビット・レジスタであるときは、16 ビットのセグメント・セレクタがレジスタの下位 16 ビットにコピーされる。レジスタの上位 16 ビットは、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・プロセッサではクリアされ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサでは未定義である。デスティネーション・オペランドがメモリ・ロケーションであるときは、セグメント・セレクタは、オペランド・サイズに関係なく、16 ビット幅でメモリに書き込まれる。

SLDT 命令は、オペレーティング・システム・ソフトウェアだけに有用であるが、アプリケーション・プログラムで使用することもできる。

操作

```
DEST ← LDTR(SegmentSelector);
```

影響を受けるフラグ

なし。

SLDT—Store Local Descriptor Table Register（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、および 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	SLDT 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	SLDT 命令は仮想 8086 モードでは認識されない。

SMSW—Store Machine Status Word

オペコード	命令	説明
0F 01 /4	SMSW *r/m16*	マシン・ステータス・ワードを *r/m16* にストアする。
0F 01 /4	SMSW *r32/m16*	マシン・ステータス・ワードを *r32* の下位 16 ビットまたは *m16* にストアし、*r32* の上位 16 ビットは未定義である。

説明

マシン・ステータス・ワード（制御レジスタ CR0 のビット 0 ～ 15）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、16 ビットの汎用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・オペランドが 32 ビット・レジスタであるときは、CR0 レジスタの下位 16 ビットがレジスタの下位 16 ビットにコピーされ、レジスタの上位 16 ビットは未定義である。デスティネーション・オペランドがメモリ・ロケーションであるときは、オペランド・サイズに関係なく、CR0 レジスタの下位 16 ビットが 16 ビット幅でメモリに書き込まれる。

SMSW 命令は、オペレーティング・システム・ソフトウェアだけに有用であるが、特権命令ではなく、アプリケーション・プログラムで使用することもできる。

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・プロセッサ、インテル® Pentium® プロセッサ、Intel486™ プロセッサ、Intel386™ プロセッサで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）命令を使用してマシン・ステータス・ワードをロードする必要がある。

操作

```
DEST ← CR0[15:0]; (* Machine status word *);
```

影響を受けるフラグ

なし。

SMSW—Store Machine Status 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、および 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(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

#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。

SQRTPD—Compute Square Roots of Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 51 /r	SQRTPD *xmm1*, *xmm2/m128*	*xmm2/m128* のパックド倍精度浮動小数点値の平方根を計算し、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）の 2 つのパックド倍精度浮動小数点値の平方根を SIMD 計算し、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] ← SQRT(SRC[63-0]);
DEST[127-64] ← SQRT(SRC[127-64]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

SQRTPD　　__m128d _mm_sqrt_pd (m128d a)

SIMD 浮動小数点例外

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。

SQRTPD—Compute Square Roots of 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。

SQRTPS—Compute Square Roots of Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 51 /r	SQRTPS *xmm1*, *xmm2/m128*	*xmm2/m128* のパックド単精度浮動小数点値の平方根を計算し、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）の 4 つのパックド単精度浮動小数点値の平方根を SIMD 計算し、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] ← SQRT(SRC[31-0]);
DEST[63-32] ← SQRT(SRC[63-32]);
DEST[95-64] ← SQRT(SRC[95-64]);
DEST[127-96] ← SQRT(SRC[127-96]);
```

同等のインテル® C/C++ コンパイラ組み込み関数

SQRTPS　　__m128 _mm_sqrt_ps(__m128 a)

SIMD 浮動小数点例外

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。

SQRTPS—Compute Square Roots of 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。

SQRTSD—Compute Square Root of Scalar Double-Precision Floating-Point Value

オペコード	命令	説明
F2 0F 51 /r	SQRTSD *xmm1*, *xmm2/m64*	*xmm2/m64* の下位の倍精度浮動小数点値の平方根を計算し、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）の下位の倍精度浮動小数点値の平方根を計算し、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] ← SQRT(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

SQRTSD　　__m128d _mm_sqrt_sd (m128d a)

SIMD 浮動小数点例外

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。

SQRTSD—Compute Square Root of 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。

SQRTSS—Compute Square Root of Scalar Single-Precision Floating-Point Value

オペコード	命令	説明
F3 0F 51 /r	SQRTSS *xmm1*, *xmm2/m32*	*xmm2/m32* の最下位の単精度浮動小数点値の平方根を計算し、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）の最下位の単精度浮動小数点値の平方根を計算し、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] ← SQRT (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

SQRTSS　　__m128 _mm_sqrt_ss(__m128 a)

SIMD 浮動小数点例外

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。

SQRTSS—Compute Square Root of 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。

STC—Set Carry Flag

オペコード	命令	説明
F9	STC	CF フラグをセットする。

説明

EFLAGS レジスタの CF フラグをセットする。

操作

CF ← 1;

影響を受けるフラグ

CF フラグがセットされる。OF、ZF、SF、AF、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。

STD—Set Direction Flag

オペコード	命令	説明
FD	STD	DF フラグをセットする。

説明

EFLAGS レジスタの DF フラグをセットする。DF フラグが 1 にセットされると、ストリング操作はインデックス・レジスタ（ESI または EDI、またはその両方）をデクリメントする。

操作

DF ← 1;

影響を受けるフラグ

DF フラグがセットされる。CF、OF、ZF、SF、AF、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。

STI—Set Interrupt Flag

オペコード	命令	説明
FB	STI	割り込みフラグをセットする。次の命令の最後で、外部マスク可能割り込みがイネーブルになる。

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、STI 命令は EFLAGS レジスタの割り込みフラグ（IF）をセットする。IF フラグがセットされた後、プロセッサは、次の命令を実行した後に外部マスク可能割り込みへの応答を開始する。この命令のディレイした効果は、プロシージャ（またはサブルーチン）から戻る直前に割り込みをイネーブルにできるように提供されている。例えば、STI 命令の後に RET 命令が続いている場合に、RET 命令は、外部割り込みが認識される前に実行を認められる[3]。STI 命令の後に（IF フラグをクリアする）CLI 命令が続いていると、STI 命令の効果はネゲートされる。

IF フラグと STI 命令および CLI 命令は、例外および NMI 割り込みの生成は妨げない。STI 命令の後に続く 1 つのマクロ命令については、NMI 割り込みが妨げられることがある。

保護モード仮想割り込みがイネーブル、CPL が 3、かつ IOPL が 3 より小さい場合は、STI は EFLAGS レジスタの VIF フラグをセットし、IF フラグには影響を与えない。

表 4-1. は、プロセッサの動作モードおよび実行中のプログラムまたはプロシージャの CPL と IOPL の設定によって、STI 命令の処置が決まることを示している。

3. 以降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、以降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, AX
MOV ESP, EBP
```

MOV DD, AX が通常は割り込みを 1 命令の間ディレイさせる場合でも、MOV ESP, EBP が実行される前に、割り込みが認識されることがある。

STI—Set Interrupt Flag（続き）

表 4-1. STI 結果のデシジョン・テーブル

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

STI—Set Interrupt Flag（続き）

操作

```
IF PE = 0  (* Executing in real-address mode *)
    THEN
        IF ← 1;  (* Set Interrupt Flag *)
    ELSE  (* Executing in protected mode or virtual-8086 mode *)
        IF VM = 0  (* Executing in protected mode*)
            THEN
                IF IOPL ≥ CPL
                    THEN
                        IF ← 1;  (* Set Interrupt Flag *)
                    ELSE
                        IF (IOPL < CPL) AND (CPL = 3) AND (VIP = 0)
                            THEN
                                VIF ← 1;  (* Set Virtual Interrupt Flag *)
                            ELSE
                                #GP(0);
                        FI;
                FI;
            ELSE  (* Executing in Virtual-8086 mode *)
                IF IOPL = 3
                    THEN
                        IF ← 1;  (* Set Interrupt Flag *)
                ELSE
                    IF ((IOPL < 3) AND (VIP = 0) AND (VME = 1))
                        THEN
                            VIF ← 1;  (* Set Virtual Interrupt Flag *)
                        ELSE
                            #GP(0); (* Trap to virtual-8086 monitor *)
                        FI;)
                FI;
        FI;
FI;
```

影響を受けるフラグ

IF フラグが 1 にセットされる。または、VIF フラグが 1 にセットされる。

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（低い特権をもつ）場合。

実アドレスモード例外

なし。

STI—Set Interrupt Flag（続き）

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（低い特権をもつ）場合。

STMXCSR—Store MXCSR Register State

オペコード	命令	説明
0F AE /3	STMXCSR *m32*	MXCSR レジスタの内容を *m32* にストアする。

説明

MXCSR 制御 / ステータス・レジスタの内容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、32 ビット・メモリ・ロケーションである。MXCSR レジスタの予約ビットは、ゼロとしてストアされる。

操作

m32 ← MXCSR;

同等のインテル® C/C++ コンパイラ組み込み関数

_mm_getcsr(void)

例外

なし。

数値例外

なし。

保護モード例外

#GP(0)	CS、DS、ES、FS、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	CR0.EM が 1 の場合。
#NM	CR0 の TS ビットがセットされている場合。
#AC	アライメントの合っていないメモリ参照が行われた場合。#AC 例外をイネーブルにするには、3 つの条件を真にする必要がある（CR0.AM をセットし、EFLAGS.AC をセットし、現行 CPL を 3 にする）。
#UD	CR4.OSFXSR（ビット 9）が 0 の場合。
	CPUID.SSE（EDX ビット 25）が 0 の場合。

STMXCSR—Store MXCSR Register State（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。
#UD	CR0.EM が 1 の場合。
#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。
	CPUID.SSE（EDX ビット 25）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。

STOS/STOSB/STOSW/STOSD—Store String

オペコード	命令	説明
AA	STOS m8	AL をアドレス ES:(E)DI にストアする。
AB	STOS m16	AX をアドレス ES:(E)DI にストアする。
AB	STOS m32	EAX をアドレス ES:(E)DI にストアする。
AA	STOSB	AL をアドレス ES:(E)DI にストアする。
AB	STOSW	AX をアドレス ES:(E)DI にストアする。
AB	STOSD	EAX をアドレス ES:(E)DI にストアする。

説明

バイト、ワード、またはダブルワードをそれぞれAL、AX、またはEAXレジスタから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ES:EDIレジスタまたは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・オーバライド・プリフィックスでオーバ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。（STOSニーモニックで指定される）明示オペランド形式では、デスティネーション・オペランドを明示的に指定することができる。この場合、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サイズとロケーションを示す記号でなければならない。ソース・オペランドは、この場合にはデスティネーション・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レジスタ、ワード・オペランドではAXレジスタ、ダブルワード・オペランドではEAXレジスタ）。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、ストリング・ストア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。

オペランドなし形式は、STOS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、ES:(E)DIがデスティネーション・オペランドであると想定され、AL、AX、またはEAXレジスタがソース・オペランドであると想定される。デスティネーション・オペランドとソース・オペランドのサイズは、STOSB（レジスタALからのバイト読み取り）、STOSW（AXからのワード読み取り）、またはSTOSD（EAXからのダブルワード読み取り）の各ニーモニックで指定される。

STOS/STOSB/STOSW/STOSD—Store String（続き）

バイト、ワード、またはダブルワードがAL、AX、またはEAXレジスタからメモリ・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)DIレジスタはインクリメントされる。DFフラグが1である場合は、(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。

STOS、STOSB、STOSW、STOSD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、データをストアするにはその前にAL、AX、またはEAXレジスタに転送する必要があるので、これらの命令はループ構造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 store)
    THEN
        DEST ← AL;
            THEN IF DF = 0
                THEN (E)DI ← (E)DI + 1;
                ELSE (E)DI ← (E)DI – 1;
            FI;
    ELSE IF (word store)
        THEN
            DEST ← AX;
                THEN IF DF = 0
                    THEN (E)DI ← (E)DI + 2;
                    ELSE (E)DI ← (E)DI – 2;
                FI;
        ELSE (* doubleword store *)
            DEST ← EAX;
                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。

STOS/STOSB/STOSW/STOSD—Store String（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 ES セグメントの範囲外の場合。 ES レジスタの内容がヌル・セグメント・セレクタの場合。
#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 ES 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 ES 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。

STR—Store Task Register

オペコード	命令	説明
0F 00 /1	STR *r/m16*	セグメント・セレクタを TR から *r/m16* にストアする。

説明

セグメント・セレクタをタスクレジスタ（TR）から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、汎用レジスタまたはメモリ・ロケーションを使用できる。この命令でストアされるセグメント・セレクタは、現在実行中のタスクのタスク・ステート・セグメント（TSS）を指す。

デスティネーション・オペランドが 32 ビット・レジスタであるときは、16 ビットのセグメント・セレクタがレジスタの下位 16 ビットにコピーされ、レジスタの上位 16 ビットはクリアされる。デスティネーション・オペランドがメモリ・ロケーションであるときは、セグメント・セレクタは、オペランド・サイズに関係なく、16 ビット幅でメモリに書き込まれる。

STR 命令は、オペレーティング・システム・ソフトウェアだけに有用である。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

```
DEST ← TR(SegmentSelector);
```

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が書き込み不可能なセグメントにあるメモリ・オペランドである場合、または実効アドレスが CS、DS、ES、FS、または GS セグメントの範囲外の場合。 DS、ES、FS、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	STR 命令は実アドレスモードでは認識されない。

STR—Store Task Register（続き）

仮想 8086 モード例外

#UD	STR 命令は仮想 8086 モードでは認識されない。

SUB—Subtract

オペコード	命令	説明
2C *ib*	SUB AL, *imm8*	AL から *imm8* を引く。
2D *iw*	SUB AX, *imm16*	AX から *imm16* を引く。
2D *id*	SUB EAX, *imm32*	EAX から *imm32* を引く。
80 /5 *ib*	SUB *r/m8*, *imm8*	*r/m8* から *imm8* を引く。
81 /5 *iw*	SUB *r/m16*, *imm16*	*r/m16* から *imm16* を引く。
81 /5 *id*	SUB *r/m32*, *imm32*	*r/m32* から *imm32* を引く。
83 /5 *ib*	SUB *r/m16*, *imm8*	*r/m16* から符号拡張された *imm8* を引く。
83 /5 *ib*	SUB *r/m32*, *imm8*	*r/m32* から符号拡張された *imm8* を引く。
28 /*r*	SUB *r/m8*, *r8*	*r/m8* から *r8* を引く。
29 /*r*	SUB *r/m16*, *r16*	*r/m16* から *r16* を引く。
29 /*r*	SUB *r/m32*, *r32*	*r/m32* から *r32* を引く。
2A /*r*	SUB *r8*, *r/m8*	*r8* から *r/m8* を引く。
2B /*r*	SUB *r16*, *r/m16*	*r16* から *r/m16* を引く。
2B /*r*	SUB *r32*, *r/m32*	*r32* から *r/m32* を引く。

説明

第 1 オペランド（デスティネーション・オペランド）から第 2 オペランド（ソース・オペランド）を引き、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 つの命令で 2 つのメモリ・オペランドを使用することはできない。）即値をオペランドとして使用すると、デスティネーション・オペランドのフォーマットの長さまで符号拡張される。

SUB 命令は、整数の減算を実行する。SUB 命令は、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し、OF フラグと CF フラグを設定して、それぞれ符号付きの結果または符号なしの結果のオーバーフローを示す。SF フラグは、符号付き結果の符号を示す。

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させることができる。

操作

DEST ← DEST – SRC;

影響を受けるフラグ

OF、SF、ZF、AF、PF、CF フラグが結果にしたがって設定される。

SUB—Subtract（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。

SUBPD—Subtract Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 5C /r	SUBPD *xmm1*, *xmm2/m128*	*xmm1* から *xmm2/m128* のパックド倍精度浮動小数点値を引く。

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からソース・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 SIMD 減算し、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

SUBPD __m128d _mm_sub_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。

SUBPD—Subtract 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。

SUBPS—Subtract Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 5C /r	SUBPS *xmm1*, *xmm2/m128*	*xmm1* のパック ド単精度浮動小数点値から *xmm2/mem* のパック ド単精度浮動小数点値を引く。

説明

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からソース・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 SIMD 減算し、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

SUBPS __m128 _mm_sub_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。

SUBPS—Subtract 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。

SUBSD—Subtract Scalar Double-Precision Floating-Point Values

オペコード	命令	説明
F2 0F 5C /r	SUBSD *xmm1*, *xmm2/m64*	*xmm1* から *xmm2/mem64* の下位の倍精度浮動小数点値を引く。

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からソース・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引き、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

SUBSD __m128d _mm_sub_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。

SUBSD—Subtract 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。

SUBSS—Subtract Scalar Single-Precision Floating-Point Values

オペコード	命令	説明
F3 0F 5C /r	SUBSS *xmm1*, *xmm2/m32*	*xmm1*の最下位の単精度浮動小数点値から *xmm2/m32* の最下位の単精度浮動小数点値を引く。

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からソース・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引き、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[127-96] remains unchanged *;
```

同等のインテル® C/C++ コンパイラ組み込み関数

SUBSS __m128 _mm_sub_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。

SUBSS—Subtract 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。

SYSENTER—Fast System Call

オペコード	命令	説明
0F 34	SYSENTER	特権レベル 0 のシステム・プロシージャへの高速コール。

説明

特権レベル 0 のシステム・プロシージャまたはルーチンへの高速コールを実行する。この命令は、SYSEXIT 命令に対するコンパニオン命令である。SYSENTER 命令は、特権レベル 3 で実行中のユーザコードからオペレーティング・システムまたは特権レベル 0 で実行中の実行プロシージャへのシステムコールについて、そのパフォーマンスが最高になるように最適化される。

SYSENTER 命令を実行する前に、ソフトウェア上で以下の MSR に値を書き込むことにより、特権レベル 0 のコード・セグメントとコード・エントリ・ポイント、特権レベル 0 のスタック・セグメントとスタックポインタを指定する必要がある。

- SYSENTER_CS_MSR ― 特権レベル 0 のコード・セグメントの 32 ビット・セグメント・セレクタが入る。（この値は、特権レベル 0 のスタック・セグメントのセグメント・セレクタを計算するのにも使用される）
- SYSENTER_EIP_MSR ― 選択された操作プロシージャまたはルーチンの先頭の命令に対する、特権レベル 0 のコード・セグメントの 32 ビット・オフセットが入る。
- SYSENTER_ESP_MSR ― 特権レベル 0 のスタックの 32 ビット・スタック・ポインタが入る。

これらの MSR は、RDMSR 命令および WRMSR 命令を使用して読み取りおよび書き込みを行うことができる。レジスタアドレスを表 4-2. に示す。これらのアドレスは、今後の IA-32 プロセッサにおいても固定されたままとなるよう定義されている。

表 4-2. SYSENTER 命令および SYSEXIT 命令によって使用される MSR

MSR	アドレス
SYSENTER_CS_MSR	174H
SYSENTER_ESP_MSR	175H
SYSENTER_EIP_MSR	176H

SYSENTER 命令を実行するとき、プロセッサは以下のことを行う。

1. セグメント・セレクタを SYSENTER_CS_MSR から CS レジスタにロードする。
2. 命令ポインタを SYSENTER_EIP_MSR から EIP レジスタにロードする。
3. SYSENTER_CS_MSR の値に 8 を加算し、その合計を SS レジスタにロードする。

SYSENTER—Fast System Call（続き）

4. スタックポインタを SYSENTER_ESP_MSR から ESP レジスタにロードする。
5. 特権レベル 0 に切り替える。
6. （フラグが設定されている場合には）EFLAGS レジスタの VM フラグをクリアする。
7. 選択したシステム・プロシージャの実行を開始する。

プロセッサは、コール元のプロシージャのリターンIPや他の状態情報をセーブしない。

SYSENTER 命令は常に、DPL が 0 の保護モード・コード・セグメントにプログラムの制御を移す。この命令を使用するには、オペレーティング・システムが以下の条件を満たしていなければならない。

- 選択されたシステム・コード・セグメントのセグメント記述子により、実行許可、読み取り許可、アクセス許可、非コンフォーミング許可を持つ最大 4G バイトのフラットな 32 ビット・コード・セグメントが選択されていること。
- 選択されたシステム・スタック・セグメントのセグメント記述子により、読み取り許可、書き込み許可、アクセス許可、拡張許可を持つ最大 4G バイトのフラットな 32 ビット・スタック・セグメントが選択されていること。

SYSENTER 命令は、実アドレスモードを除くすべての操作モードから起動できる。

SYSENTER 命令と SYSEXIT 命令はコンパニオン命令であるが、コール / リターンのペアを形成するものではない。SYSENTER 命令を実行するときに、プロセッサはユーザコードの状態情報をセーブしない。また、SYSENTER 命令も SYSEXIT 命令も、スタックへのパラメータ渡しをサポートしない。

SYSENTER 命令と SYSEXIT 命令をコンパニオン命令として使用し、特権レベル 3 のコードと特権レベル 0 のオペレーティング・システム・プロシージャ間のトランジション（移行）を行うには、以下の規則に従う必要がある。

- 特権レベル 0 のコードとスタック・セグメントのセグメント記述子、特権レベル 3 のコードとスタック・セグメントのセグメント記述子は、グローバル記述子テーブル内で連続していなければならない。この規則により、プロセッサは、SYSENTER_CS_MSR MSR に入力された値からセグメント・セレクタを計算することができる。

SYSENTER—Fast System Call（続き）

- ユーザコードが実行する高速システムコールの「スタブ」ルーチン（通常は、共用ライブラリまたはDLL内にある）では、コール元プロシージャへのリターンが要求される場合、必要なリターンIPとプロセッサ状態情報をセーブしておく必要がある。同様に、SYSENTER命令でコールされるオペレーティング・システムまたは実行プロシージャでは、ユーザコードにリターンするときにアクセス権が必要であると共に、このセーブされているリターンIPと状態に関する情報を使用する必要がある。

SYSENTER命令とSYSEXIT命令は、インテル® Pentium® IIプロセッサでIA-32アーキテクチャに導入された。これらの命令をプロセッサで使用できるかどうかは、CPUID命令によってEDXレジスタに返されるSYSENTER/SYSEXIT Present（SEP）機能フラグの状態によって示される。オペレーティング・システムでSEPフラグを調べる場合は、プロセッサのファミリとモデルも調べて、SYSENTER/SYSEXIT命令を実際にサポートしているかどうかを確認しなければならない。例えば、次のコードを使用できる。

```
IF (CPUID SEP bit is set)
    THEN IF (Family = 6) AND (Model < 3) AND (Stepping < 3)
        THEN
            SYSENTER/SYSEXIT_Not_Supported
        FI;
    ELSE SYSENTER/SYSEXIT_Supported
FI;
```

インテル® Pentium® Proプロセッサ（モデル1）上でCPUID命令を実行すると、SEPフラグがセットされて返されるが、SYSENTER/SYSEXIT命令はサポートされない。

操作

```
IF CR0.PE = 0 THEN #GP(0); FI;
IF SYSENTER_CS_MSR = 0 THEN #GP(0); FI;

EFLAGS.VM ← 0                   (* Insures protected mode execution *)
EFLAGS.IF ← 0                   (* Mask interrupts *)
EFLAGS.RF ← 0

CS.SEL ← SYSENTER_CS_MSR(* Operating system provides CS *)
(* Set rest of CS to a fixed value *)
CS.SEL.CPL ← 0
CS.BASE ← 0                     (* Flat segment *)

CS.LIMIT ← FFFFH                (* 4 GByte limit *)
CS.ARbyte.G ← 1                 (* 4 KByte granularity *)
CS.ARbyte.S ← 1
CS.ARbyte.TYPE ← 1011B          (* Execute + Read, Accessed *)
CS.ARbyte.D ← 1                 (* 32-bit code segment*)
```

SYSENTER—Fast System Call（続き）

```
CS.ARbyte.DPL ← 0
CS.ARbyte.RPL ← 0
CS.ARbyte.P ← 1

SS.SEL ← CS.SEL + 8
(* Set rest of SS to a fixed value *)
SS.BASE ← 0                 (* Flat segment *)
SS.LIMIT ← FFFFH            (* 4 GByte limit *)
SS.ARbyte.G ← 1             (* 4 KByte granularity *)
SS.ARbyte.S ←
SS.ARbyte.TYPE ← 0011B      (* Read/Write, Accessed *)
SS.ARbyte.D ← 1             (* 32-bit stack segment*)
SS.ARbyte.DPL ← 0
SS.ARbyte.RPL ← 0
SS.ARbyte.P ← 1

ESP ← SYSENTER_ESP_MSR
EIP ← SYSENTER_EIP_MSR
```

影響を受けるフラグ

VM、IF、RF（上記の操作を参照）

保護モード例外

#GP(0)	SYSENTER_CS_MSR の値がゼロの場合。

実アドレスモード例外

#GP(0)	保護モードがイネーブルになっていない場合。

仮想 8086 モード例外

#GP(0)	SYSENTER_CS_MSR の値がゼロの場合。

SYSEXIT—Fast Return from Fast System Call

オペコード	命令	説明
0F 35	SYSEXIT	特権レベル 3 のユーザコードに高速リターンする。

説明

特権レベル3のユーザコードへの高速リターンを実行する。この命令は、SYSENTER命令に対するコンパニオン命令である。SYSEXIT命令は、保護レベル0で実行中のシステム・プロシージャから保護レベル3で実行中のユーザ・プロシージャへのリターンについて、そのパフォーマンスが最高になるように最適化される。この命令は、特権レベル0で実行中のコードから実行しなければならない。

SYSEXIT命令を実行する前に、ソフトウェア上で以下のMSRおよび汎用レジスタに値を書き込むことにより、特権レベル3のコード・セグメントとコード・エントリ・ポイント、特権レベル3のスタック・セグメントとスタックポインタを指定する必要がある。

- SYSENTER_CS_MSR － プロセッサが現在実行中の特権レベル0のコード・セグメントの32ビット・セグメント・セレクタが入る。（この値は、特権レベル3のコードおよびスタック･セグメントのセグメント･セレクタを計算するのに使用される）
- EDX － ユーザコードで最初に実行される命令に対する、特権レベル3のコード・セグメントの32ビット・オフセットが入る。
- ECX － 特権レベル3のスタックの32ビット・スタック・ポインタが入る。

SYSENTER_CS_MSR MSRは、RDMSR命令およびWRMSR命令を使用して読み取りおよび書き込みを行うことができる。レジスタアドレスを表4-2.に示す。このアドレスは、今後のIA-32プロセッサにおいても固定されたままとなるよう定義されている。

SYSEXIT命令を実行するとき、プロセッサは以下のことを行う。

1. SYSENTER_CS_MSRの値に16を加算し、その合計をCSセレクタレジスタにロードする。
2. 命令ポインタをEDXレジスタからEIPレジスタにロードする。
3. SYSENTER_CS_MSRの値に24を加算し、その合計をSSセレクタレジスタにロードする。
4. スタックポインタをECXレジスタからESPレジスタにロードする。
5. 特権レベル3に切り替える。
6. EIPアドレスでユーザコードの実行を開始する。

SYSEXIT—Fast Return from Fast System Call（続き）

コールおよびリターンのコンパニオン命令としてSYSENTER 命令とSYSEXIT 命令を使用する方法については、「SYSENTER—Fast System Call」を参照のこと。

SYSEXIT 命令は常に、DPL が3 の保護モード・コード・セグメントにプログラムの制御を移す。この命令を使用するには、オペレーティング・システムが以下の条件を満たしていなければならない。

- 選択されたユーザ・コード・セグメントのセグメント記述子により、実行許可、読み取り許可、アクセス許可、非コンフォーミング許可を持つ最大 4G バイトのフラットな32 ビット・コード・セグメントが選択されていること。
- 選択されたユーザ・スタック・セグメントのセグメント記述子により、拡張許可、読み取り許可、書き込み許可、アクセス許可を持つ最大4G バイトのフラットな32 ビット・スタック・セグメントが選択されていること。

SYSENTER は、実アドレスモードを除くすべての操作モードから起動できる。

SYSENTER 命令とSYSEXIT 命令は、インテル® Pentium® II プロセッサでIA-32 アーキテクチャに導入された。これらの命令をプロセッサで使用できるかどうかは、CPUID 命令によってEDX レジスタに返されるSYSENTER/SYSEXIT Present（SEP）機能フラグの状態によって示される。オペレーティング・システムでSEP フラグを調べる場合は、プロセッサのファミリとモデルも調べて、SYSENTER/SYSEXIT 命令を実際にサポートしているかどうかを確認しなければならない。例えば、次のコードを使用できる。

```
IF (CPUID SEP bit is set)
    THEN IF (Family = 6) AND (Model < 3) AND (Stepping < 3)
        THEN
            SYSENTER/SYSEXIT_Not_Supported
        FI;
    ELSE SYSENTER/SYSEXIT_Supported
FI;
```

インテル® Pentium® Pro プロセッサ（モデル1）上でCPUID 命令を実行すると、SEP フラグがセットされて返されるが、SYSENTER/SYSEXIT 命令はサポートされない。

操作

```
IF SYSENTER_CS_MSR = 0 THEN #GP(0); FI;
IF CR0.PE = 0 THEN #GP(0); FI;
IF CPL ≠ 0 THEN #GP(0)

CS.SEL ← (SYSENTER_CS_MSR + 16)(* Segment selector for return CS *)
(* Set rest of CS to a fixed value *)
CS.BASE ← 0                     (* Flat segment *)
CS.LIMIT ← FFFFH                (* 4 GByte limit *)
```

SYSEXIT—Fast Return from Fast System Call（続き）

```
CS.ARbyte.G ← 1                 (* 4 KByte granularity *)
CS.ARbyte.S ← 1
CS.ARbyte.TYPE ← 1011B          (* Execute, Read, Non-Conforming Code *)
CS.ARbyte.D ← 1                 (* 32-bit code segment*)
CS.ARbyte.DPL ← 3
CS.ARbyte.RPL ← 3
CS.ARbyte.P ← 1

SS.SEL ← (SYSENTER_CS_MSR + 24)(* Segment selector for return SS *)
(* Set rest of SS to a fixed value *)
SS.BASE ← 0                     (* Flat segment *)
SS.LIMIT ← FFFFH                (* 4 GByte limit *)
SS.ARbyte.G ← 1                 (* 4 KByte granularity *)
SS.ARbyte.S ←
SS.ARbyte.TYPE ← 0011B          (* Expand Up, Read/Write, Data *)
SS.ARbyte.D ← 1                 (* 32-bit stack segment*)
SS.ARbyte.DPL ← 3
SS.ARbyte.RPL ← 3
SS.ARbyte.P ← 1

ESP     ← ECX
EIP     ← EDX
```

影響を受けるフラグ

なし。

保護モード例外

#GP(0)	SYSENTER_CS_MSR の値がゼロの場合。

実アドレスモード例外

#GP(0)	保護モードがイネーブルになっていない場合。

仮想 8086 モード例外

#GP(0)	SYSENTER_CS_MSR の値がゼロの場合。

TEST—Logical Compare

オペコード	命令	説明
A8 *ib*	TEST AL, *imm8*	*imm8* と AL との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
A9 *iw*	TEST AX, *imm16*	*imm16* と AX との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
A9 *id*	TEST EAX, *imm32*	*imm32* と EAX との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
F6 /0 *ib*	TEST *r/m8*, *imm8*	*imm8* と *r/m8* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
F7 /0 *iw*	TEST *r/m16*, *imm16*	*imm16* と *r/m16* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
F7 /0 *id*	TEST *r/m32*, *imm32*	*imm32* と *r/m32* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
84 /*r*	TEST *r/m8*, *r8*	*r8* と *r/m8* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
85 /*r*	TEST *r/m16*, *r16*	*r16* と *r/m16* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。
85 /*r*	TEST *r/m32*, *r32*	*r32* と *r/m32* との ADN をとり、結果にしたがって SF、ZF、PF を設定する。

説明

第1オペランド（ソース1オペランド）と第2オペランド（ソース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、結果にしたがって SF、ZF、PF ステータス・フラグを設定する。結果は、その後捨てられる。

操作

```
TEMP ← SRC1 AND SRC2;
SF ← MSB(TEMP);
IF TEMP = 0
    THEN ZF ← 1;
    ELSE ZF ← 0;
FI:
PF ← BitwiseXNOR(TEMP[0:7]);
CF ← 0;
OF ← 0;
(*AF is Undefined*)
```

影響を受けるフラグ

OF および CF フラグが 0 にセットされる。SF、ZF、および PF フラグが結果にしたがって設定される（上記の「操作」の項を参照）。AF フラグの状態は未定義。

TEST—Logical Compare（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。

UCOMISD—Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS

オペコード	命令	説明
66 0F 2E /r	UCOMISD *xmm1*, *xmm2/m64*	*xmm1* と *xmm2/m64* の下位の倍精度浮動小数点値を（アンオーダー）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）の下位クワッドワード内の倍精度浮動小数点値のアンオーダー比較を実行し、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、アンオーダーの結果が返される。

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。

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

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 ← UnorderedCompare(SRC1[63-0] <> SRC2[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_ucomieq_sd(__m128d a, __m128d b)
int_mm_ucomilt_sd(__m128d a, __m128d b)
int_mm_ucomile_sd(__m128d a, __m128d b)
int_mm_ucomigt_sd(__m128d a, __m128d b)
int_mm_ucomige_sd(__m128d a, __m128d b)
int_mm_ucomineq_sd(__m128d a, __m128d b)

UCOMISD—Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS（続き）

SIMD 浮動小数点例外

無効（SNaN 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。

UCOMISD—Unordered Compare Scalar 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。

UCOMISS—Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS

オペコード	命令	説明
0F 2E /r	UCOMISS *xmm1*, *xmm2/m32*	*xmm1* レジスタの最下位の単精度浮動小数点値と *xmm2/mem* の最下位の単精度浮動小数点値を比較し、結果にしたがってステータス・フラグを設定する。

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）の最下位ダブルワード内の単精度浮動小数点値のアンオーダー比較を実行し、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、アンオーダーの結果が返される。

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32ビットのメモリ・ロケーションである。

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

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 ← UnorderedCompare(SRC1[63-0] <> SRC2[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_ucomieq_ss(__m128 a, __m128 b)
int_mm_ucomilt_ss(__m128 a, __m128 b)
int_mm_ucomile_ss(__m128 a, __m128 b)
int_mm_ucomigt_ss(__m128 a, __m128 b)
int_mm_ucomige_ss(__m128 a, __m128 b)
int_mm_ucomineq_ss(__m128 a, __m128 b)

UCOMISS—Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS（続き）

SIMD 浮動小数点例外

無効（SNaN 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。

UCOMISS—Unordered Compare Scalar 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。

UD2—Undefined Instruction

オペコード	命令	説明
0F 0B	UD2	無効オペコード例外を発生させる。

説明

無効オペコードを生成する。この命令は、無効オペコードを明示的に生成してソフトウェアをテストするために提供されている。この命令のオペコードは、この目的のために予約されている。

無効オペコード例外を発生させる以外は、この命令はNOP命令と同じである。

操作

#UD (* Generates invalid opcode exception *);

影響を受けるフラグ

なし。

例外（すべての操作モード）

#UD	命令は、すべての動作モードで無効オペコード例外を発生させることが保証されている。

UNPCKHPD—Unpack and Interleave High Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 15 /r	UNPCKHPD *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* の上位クワッドワードの倍精度浮動小数点値をアンパックしてインタリーブする。

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）の上位の倍精度浮動小数点値をアンパックしてインタリーブする。図 4-14. を参照のこと。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。

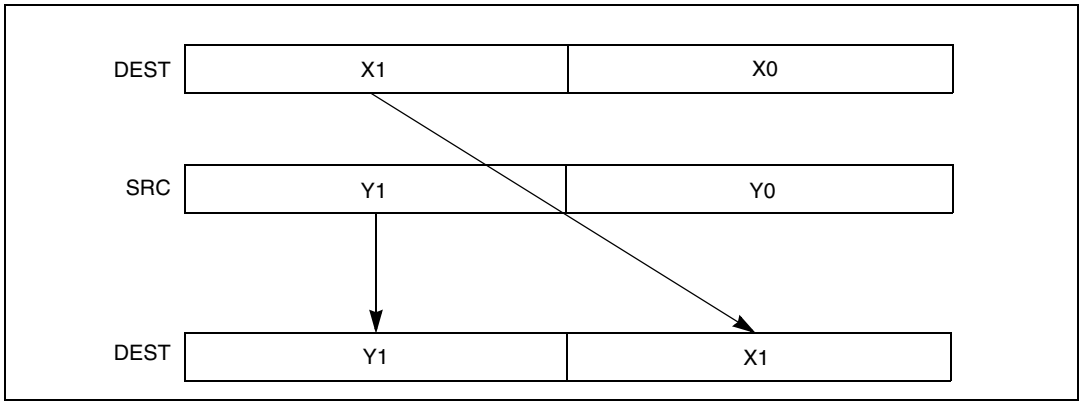


図 4-14. UNPCKHPD 命令の上位アンパックとインタリーブ操作

メモリ・オペランドからアンパックする場合、プロセッサによっては、適切な 64 ビットだけをフェッチすることがある。この場合も、16 バイト・アライメントの条件と通常のセグメント・チェックが適用される。

操作

```
DEST[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

UNPCKHPD　　__m128d _mm_unpackhi_pd(__m128d a, __m128d b)

SIMD 浮動小数点例外

なし。

UNPCKHPD—Unpack and Interleave High 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。
	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。

UNPCKHPS—Unpack and Interleave High Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 15 /r	UNPCKHPS *xmm1*, *xmm2/m128*	*xmm1* の上位クワッドワードの単精度浮動小数点値と *xmm2/mem* の上位半分の単精度浮動小数点値を xmm1 レジスタにアンパックしてインタリーブする。

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）の上位の単精度浮動小数点値をアンパックしてインタリーブする。図 4-15. を参照のこと。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。

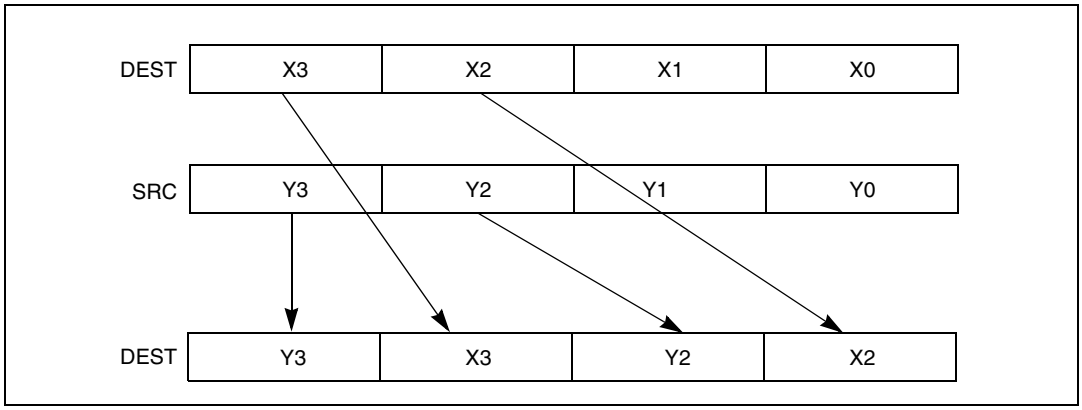


図 4-15. UNPCKHPS 命令の上位アンパックとインタリーブ操作

プロセッサによっては、メモリ・オペランドからのアンパックの際に、適切な 64 ビットだけをフェッチすることがある。しかしこの場合も、16 バイト・アライメントのチェックと通常のセグメント・チェックが行われる。

操作

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

UNPCKHPS __m128 _mm_unpackhi_ps(__m128 a, __m128 b)

UNPCKHPS—Unpack and Interleave High Packed Single-Precision Floating-Point Values（続き）

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。

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。

UNPCKLPD—Unpack and Interleave Low Packed Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 14 /r	UNPCKLPD *xmm1*, *xmm2/m128*	*xmm1* と *xmm2/m128* の下位クワッドワードの倍精度浮動小数点値をアンパックしてインタリーブする。

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アンパックしてインタリーブする。図 4-16. を参照のこと。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。

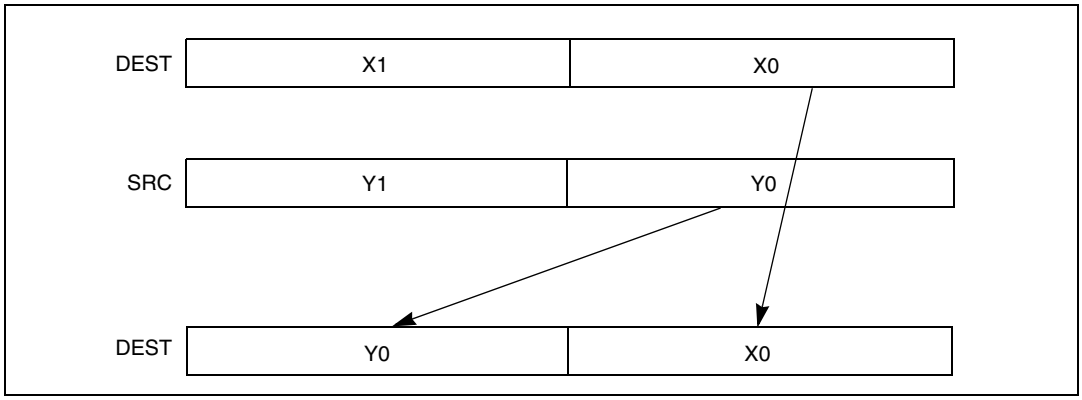


図 4-16. UNPCKLPD 命令の下位アンパックとインタリーブ操作

メモリ・オペランドからアンパックする場合、プロセッサによっては、適切な 64 ビットだけをフェッチすることがある。この場合も、16 バイト・アライメントの条件と通常のセグメント・チェックが適用される。

操作

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

同等のインテル® C/C++ コンパイラ組み込み関数

UNPCKHPD　　__m128d _mm_unpacklo_pd(__m128d a, __m128d b)

SIMD 浮動小数点例外

なし。

UNPCKLPD—Unpack and Interleave Low 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。
	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。

UNPCKLPS—Unpack and Interleave Low Packed Single-Precision Floating-Point Values

オペコード	命令	説明
0F 14 /r	UNPCKLPS *xmm1*, *xmm2/m128*	*xmm1* の下位クワッドワードの単精度浮動小数点値と *xmm2/mem* の下位半分の単精度浮動小数点値を xmm1 レジスタにアンパックしてインタリーブする。

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をアンパックしてインタリーブする。図 4-17. を参照のこと。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。

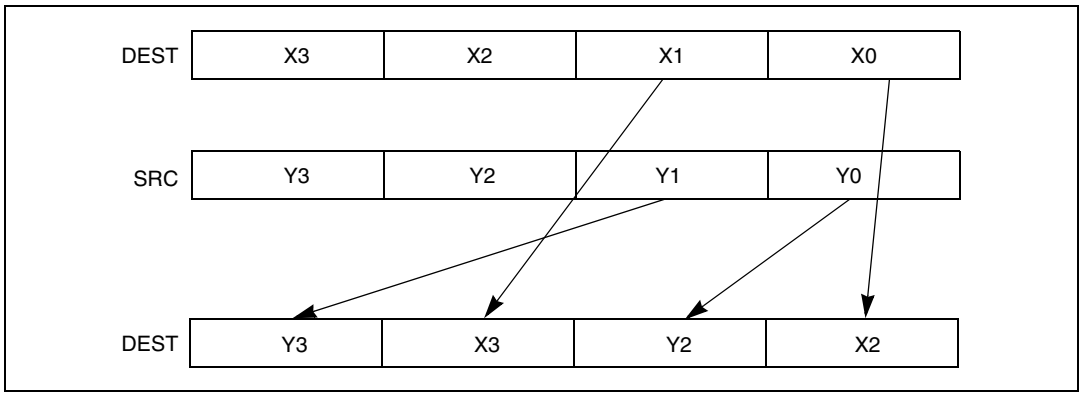


図 4-17. UNPCKLPS 命令の下位アンパックとインタリーブ操作

プロセッサによっては、メモリ・オペランドからのアンパックの際に、適切な 64 ビットだけをフェッチすることがある。しかしこの場合も、16 バイト・アライメントのチェックと通常のセグメント・チェックが行われる。

操作

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

同等のインテル® C/C++ コンパイラ組み込み関数

UNPCKLPS __m128 _mm_unpacklo_ps(__m128 a, __m128 b)

UNPCKLPS—Unpack and Interleave Low Packed Single-Precision Floating-Point Values（続き）

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。

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。

VERR, VERW—Verify a Segment for Reading or Writing

オペコード	命令	説明
0F 00 /4	VERR *r/m16*	*r/m16* で指定されたセグメントを読み取りできる場合 ZF=1 を設定する。
0F 00 /5	VERW *r/m16*	*r/m16* で指定されたセグメントに書き込みできる場合 ZF=1 を設定する。

説明

ソース・オペランドで指定されたコード・セグメントまたはデータ・セグメントが現行特権レベル（CPL）から読み取り可能（VERR）または書き込み可能（VERW）であるかを検証する。ソース・オペランドは、検証するセグメントのセグメント・セレクタをもつ16ビットのレジスタまたはメモリ・ロケーションである。セグメントがアクセス可能で読み取り可能（VERR）または書き込み可能（VERW）である場合は、ZF フラグがセットされる。そうでない場合は、ZF フラグがクリアされる。コード・セグメントが書き込み可能であると検証されることはない。このチェックは、システム・セグメントに行うことはできない。

ZF フラグをセットするには、以下の条件が満たされていなければならない。

- セグメント・セレクタがヌルでない。
- セレクタは、ディスクリプタ・テーブル（GDT または LDT）の範囲内のディスクリプタを指定していなければならない。
- セレクタは、(システム・セグメントまたはゲートのディスクリプタではなく) コード・セグメントまたはデータ・セグメントのディスクリプタを指定していなければならない。
- VERR 命令では、セグメントは読み取り可能でなければならない。
- VERW 命令では、セグメントは書き込み可能なデータ・セグメントでなければならない。
- セグメントがコンフォーミング・コード・セグメントでない場合は、セグメントの DPL は、CPL およびセグメント・セレクタの RPL の両方以上（以下の特権をもつ）でなければならない。

この検証動作は、セグメント・セレクタが DS、ES、FS、または GS レジスタにロードされ、指定されたアクセス（読み取りまたは書き込み）が行われる場合と同じ動作である。セグメント・セレクタの値は保護例外を発生させることはなく、ソフトウェアは起こり得るセグメント・アクセス問題を前もって処理することができる。

VERR, VERW—Verify a Segment for Reading or Writing（続き）

操作

```
IF SRC[Offset) > (GDTR(Limit) OR (LDTR(Limit))
        THEN
            ZF ← 0
Read segment descriptor;
IF SegmentDescriptor(DescriptorType) = 0 (* system segment *)
    OR (SegmentDescriptor(Type) ≠ conforming code segment)
    AND (CPL > DPL) OR (RPL > DPL)
        THEN
            ZF ← 0
        ELSE
            IF ((Instruction = VERR) AND (segment = readable))
                OR ((Instruction = VERW) AND (segment = writable))
                THEN
                    ZF ← 1;
            FI;
FI;
```

影響を受けるフラグ

ZF フラグは、セグメントがアクセス可能で読み取り可能（VERR）または書き込み可能（VERW）である場合は 1 にセットされ、そうでない場合は 0 にクリアされる。

保護モード例外

これらの命令に対して生成される唯一の例外は、ソース・オペランドの不当なアドレス指定に関係する例外である。

#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、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	VERR命令およびVERW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	VERR 命令および VERW 命令は、仮想 8086 モードでは認識されない。

WAIT/FWAIT—Wait

オペコード	命令	説明
9B	WAIT	未処理のマスクされていない浮動小数点例外をチェックする。
9B	FWAIT	未処理のマスクされていない浮動小数点例外をチェックする。

説明

プロセッサは、未処理のマスクされていない浮動小数点例外があるかチェックし、あれば処理してから先に進む。（FWAIT は、WAIT の代替ニーモニックである。）

この命令は、コードの重要な部分で例外を同期させるために有用である。浮動小数点命令の後に WAIT 命令を入れると、この命令が発生させる可能性があるマスクされていない浮動小数点例外があれば、プロセッサはそれらを処理してから命令の結果を修正できることが保証される。WAIT/FWAIT 命令の使用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 8 章の「x87 FPU 例外の同期 」の節を参照のこと。

操作

```
CheckForPendingUnmaskedFloatingPointExceptions;
```

FPU 影響を受けるフラグ

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 MP および TS がセットされた場合。

実アドレスモード例外

#NM	CR0 の MP および TS がセットされた場合。

仮想 8086 モード例外

#NM	CR0 の MP および TS がセットされた場合。

WBINVD—Write Back and Invalidate Cache

オペコード	命令	説明
0F 09	WBINVD	ライトバックして内部キャッシュをフラッシュする。外部キャッシュのライトバックと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で修正されているすべてのキャッシュ・ラインをメインメモリにライトバックし、内部キャッシュを無効化（フラッシュ）する。命令は、その後、外部キャッシュにも、修正されているデータをライトバックするよう指示する特殊機能バスサイクルと、外部キャッシュを無効化するよう指示するもう 1 つのバスサイクルを発行する。

この命令を実行した後、プロセッサは、外部キャッシュのライトバック操作と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・ライトバック信号およびキャッシュ・フラッシュ信号に応答することは、ハードウェアによって行う。

WBINVD 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。この命令はシリアル化命令でもある。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、下巻』の第 7 章の「シリアル化命令」の節を参照。

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場合は、ソフトウェアは INVD 命令を使用することができる。

IA-32 アーキテクチャにおける互換性

WBINVD 命令はプロセッサに依存し、その機能は将来の 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

```
WriteBack(InternalCaches);
Flush(InternalCaches);
SignalWriteBack(ExternalCaches);
SignalFlush(ExternalCaches);
Continue (* Continue execution);
```

影響を受けるフラグ

なし。

WBINVD—Write Back and Invalidate Cache（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。

実アドレスモード例外

なし。

仮想 8086 モード例外

#GP(0)	WBINVD 命令は仮想 8086 モードで実行することはできない。

WRMSR—Write to Model Specific Register

オペコード	命令	説明
0F 30	WRMSR	EDX:EAX の値を ECX で指定される MSR に書き込む。

説明

レジスタEDX:EAXの内容をECXレジスタで指定された64ビットのモデル固有レジスタ（MSR）に書き込む。ECX レジスタにロードされる入力値は、書き込み先の MSR のアドレスである。選択されたMSRの上位32ビットにEDX レジスタの内容がコピーされ、MSR の下位 32 ビットに EAX レジスタの内容がコピーされる。MSR の未定義ビットまたは予約ビットは、以前に読み取られている値に設定される。

この命令は、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。そうしないと、一般保護例外#GP(0)が生成される。予約されているかまたはインプリメントされていないMSRアドレスをECXに指定しても、一般保護例外が生成される。また、予約されている MSR のビットに書き込もうとすると、一般保護例外が生成される。

WRMSR命令を使用してMTRRに書き込むと、グローバル・エントリを含めてTLBが無効化される。詳細は『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、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）」の節を参照のこと。

MSRは、テスト機能、実行トレース、性能モニタリング、マシン・チェック・エラーの機能を制御する。『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、下巻』の付録 B「モデル固有レジスタ（MSR）」では、この命令で読み込むことができるすべてのMSRとそれらのアドレスを一覧している。各プロセッサ・ファミリは、独自に一連のMSRを持っていることに注意すること。

WRMSR命令はシリアル化命令である。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、下巻』の第7章の「シリアル化命令」の節を参照のこと。

この命令を使用する前に、CPUID 命令を使用して MSR がサポートされている（EDX[5]=1）かどうかを確認する必要がある。

IA-32 アーキテクチャにおける互換性

MSR 命令および WRMSR 命令でそれらを読み取る機能は、インテル® Pentium® プロセッサでIA-32アーキテクチャに導入された。インテルPentiumプロセッサより以前のIA-32プロセッサでこの命令を実行すると、無効オペコード例外#UDが生成される。

WRMSR—Write to Model Specific Register（続き）

操作

MSR[ECX] ← EDX:EAX;

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。 ECX の値が予約されているかまたはインプリメントされていない MSR アドレスを指定している場合。

実アドレスモード例外

#GP	ECX の値が予約されているかまたはインプリメントされていない MSR アドレスを指定している場合。

仮想 8086 モード例外

#GP(0)	WRMSR 命令は仮想 8086 モードでは認識されない。

XADD—Exchange and Add

オペコード	命令	説明
0F C0/r	XADD *r/m8*, *r8*	*r8* を *r/m8* と交換し、合計を *r/m8* にロードする。
0F C1/r	XADD *r/m16*, *r16*	*r16* を *r/m16* と交換し、合計を *r/m16* にロードする。
0F C1/r	XADD *r/m32*, *r32*	*r32* を *r/m32* と交換し、合計を *r/m32* にロードする。

説明

第 1 オペランド（デスティネーション・オペランド）を第 2 オペランド（ソース・オペランド）と交換し、次に 2 つの値の合計をデスティネーション・オペランドにロードする。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・オペランドは、レジスタである。

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。

IA-32 アーキテクチャにおける互換性

Intel486™ プロセッサより以前の IA-32 プロセッサは、この命令を認識しない。この命令を使用する場合は、以前のプロセッサ上で実行する同等のコード・シーケンスを備えなければならない。

操作

```
TEMP ← SRC + DEST
SRC ← DEST
DEST ← TEMP
```

影響を受けるフラグ

CF、PF、AF、SF、ZF、OF フラグが、デスティネーション・オペランドにストアされる加算の結果にしたがって設定される。

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。

XADD—Exchange and Add（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	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。

XCHG—Exchange Register/Memory with Register

オペコード	命令	説明
90+*rw*	XCHG AX, *16*	*r16* を AX と交換する。
90+*rw*	XCHG *r16*, X	AX を *r16* と交換する。
90+*rd*	XCHG EAX, *r32*	*r32* を EAX と交換する。
90+*rd*	XCHG *r32*, EAX	EAX を *r32* と交換する。
86 /*r*	XCHG *r/m8*, *r8*	*r8*（バイトレジスタ）を *r/m8* からのバイトと交換する。
86 /*r*	XCHG *r8*, *r/m8*	*r/m8* からのバイトを r8（バイトレジスタ）と交換する。
87 /*r*	XCHG *r/m16*, *r16*	*r16* を *r/m16* からのワードと交換する。
87 /*r*	XCHG *r16*, *r/m16*	*r/m16* からのワードを *r16* と交換する。
87 /*r*	XCHG *r/m32*, *r32*	*r32* を *r/m32* からのダブルワードと交換する。
87 /*r*	XCHG *r32*, *r/m32*	*r/m32* からのダブルワードを *r32* と交換する。

説明

デスティネーション・オペランド（第 1 オペランド）の内容をソース・オペランド（第 2 オペランド）の内容と交換する。オペランドには、2 個の汎用レジスタまたは 1 つのレジスタと 1 つのメモリ・ロケーションを使用できる。メモリ・オペランドが参照されていると、LOCK プリフィックスの有無、あるいは IOPL の値に関係なく、プロセッサのロッキング・プロトコルが交換操作が持続している間自動的にインプリメントされる。（ロッキング・プロトコルに関する詳細については、本章の「LOCK プリフィックスの説明」を参照のこと。）

この命令は、プロセスを同期するためのセマフォまたは同様のデータ構造体のインプリメンテーションに有用である。（バス・ロッキン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 7 章の「バスのロック」の項を参照のこと。）

XCHG 命令は、16 ビット・オペランドでは BSWAP 命令の代わりに使用することもできる。

操作

```
TEMP ← DEST
DEST ← SRC
SRC ← TEMP
```

影響を受けるフラグ

なし。

XCHG—Exchange Register/Memory with Register（続き）

保護モード例外

#GP(0)	いずれかの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。

XLAT/XLATB—Table Look-up Translation

オペコード	命令	説明
D7	XLAT *m8*	AL をメモリバイト DS:[(E)BX+ 符号なし AL] に設定する。
D7	XLATB	AL をメモリバイト DS:[(E)BX+ 符号なし AL] に設定する。

説明

AL レジスタの内容をテーブル・インデックスとして使用して、メモリ内のテーブルのバイトエントリの位置を探し、テーブルエントリの内容を AL レジスタにコピーし直す。AL レジスタのインデックスは、符号なし整数として取り扱われる。XLAT 命令および XLATB 命令は、メモリ内のテーブルのベースアドレスを（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:EBX レジスタまたは DS:BX レジスタから得る。（DS セグメントは、セグメント・オーバライド・プリフィックスを使用してオーバ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」形式と「オペランドなし」形式という 2 つの形式が使用できる。（XLAT ニーモニックで指定される）明示オペランド形式では、テーブルのベースアドレスを記号で明示的に指定することができる。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、記号は、正しいベースアドレスを指定する必要はない。ベースアドレスは、常に DS:(E)BX レジスタによって指定されるので、XLAT 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。

オペランドなし形式（XLATB）は、XLAT 命令の「ショート形式」を提供する。この場合も、プロセッサは DS:(E)BX レジスタがテーブルのベースアドレスをもっていると想定する。

操作

```
IF AddressSize = 16
    THEN
        AL ← (DS:BX + ZeroExtend(AL))
    ELSE (* AddressSize = 32 *)
        AL ← (DS:EBX + ZeroExtend(AL));
FI;
```

影響を受けるフラグ

なし。

XLAT/XLATB—Table Look-up Translation（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。

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。

XOR—Logical Exclusive OR

オペコード	命令	説明
34 *ib*	XOR AL, *imm8*	AL と *imm8* との XOR をとる。
35 *iw*	XOR AX, *imm16*	AX と *imm16* との XOR をとる。
35 *id*	XOR EAX, *imm32*	EAX と *imm32* との XOR をとる。
80 /6 *ib*	XOR *r/m8*, *imm8*	*r/m8* と *imm8* との XOR をとる。
81 /6 *iw*	XOR *r/m16*, *imm16*	*r/m16* と *imm16* との XOR をとる。
81 /6 *id*	XOR *r/m32*, *imm32*	*r/m32* と *imm32* との XOR をとる。
83 /6 *ib*	XOR *r/m16*, *imm8*	*r/m16* と *imm8*（符号拡張）との XOR をとる。
83 /6 *ib*	XOR *r/m32*, *imm8*	*r/m32* と *imm8*（符号拡張）との XOR をとる。
30 /*r*	XOR *r/m8*, *r8*	*r/m8* と *r8* との XOR をとる。
31 /*r*	XOR *r/m16*, *r16*	*r/m16* と *r16* との XOR をとる。
31 /*r*	XOR *r/m32*, *r32*	*r/m32* と *r32* との XOR をとる。
32 /*r*	XOR *r8*, *r/m8*	*r8* と *r/m8* との XOR をとる。
33 /*r*	XOR *r16*, *r/m16*	*r16* と *r/m16* との XOR をとる。
33 /*r*	XOR *r32*, *r/m32*	*r32* と *r/m32* との XOR をとる。

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）との間のビット単位の XOR（排他的論理和）演算を実行し、結果をデスティネーション・オペランド・ロケーションにストアする。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 つの命令で 2 つのメモリ・オペランドを使用することはできない。）各ビットの結果は、オペランドの対応するビットが異なる場合は 1 になり、対応するビットが同じ場合は 0 になる。

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 XOR SRC;

影響を受けるフラグ

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

XOR—Logical Exclusive OR（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)	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。

XORPD—Bitwise Logical XOR for Double-Precision Floating-Point Values

オペコード	命令	説明
66 0F 57 /r	XORPD *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の XOR（排他的論理和）演算を実行する。

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 XOR（排他的論理和）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] BitwiseXOR SRC[127-0];

同等のインテル® C/C++ コンパイラ組み込み関数

XORPD　　__m128d _mm_xor_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。

XORPD—Bitwise Logical XOR for 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。

XORPS—Bitwise Logical XOR for Single-Precision Floating-Point Values

オペコード	命令	説明
0F 57 /r	XORPS *xmm1*, *xmm2/m128*	*xmm2/m128* と *xmm1* のビット単位の XOR（排他的論理和）演算を実行する。

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間でビット単位の XOR（排他的論理和）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] BitwiseXOR SRC[127-0];

同等のインテル® C/C++ コンパイラ組み込み関数

XORPS　　__m128 _mm_xor_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。

XORPS—Bitwise Logical XOR for 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。

A

オペコード・マップ

付録 A
オペコード・マップ

本付録にあるオペコード・テーブルは、IA-32 オブジェクト・コードを解釈する際に便利である。命令は 3 つのエンコーディング・グループに分かれている。そのグループとは、1 バイト・オペコードのエンコーディング、2 バイト・オペコードのエンコーディング、およびエスケープ（浮動小数点）エンコーディングである。

1 バイトと 2 バイトのオペコードのエンコーディングは、整数、システム、MMX® テクノロジ、SSE、SSE2、SSE3 のエンコーディングに使用される。これらの命令のオペコード・マップを表 A-2. ～ A-3. に示す。A.3.1. 項「1 バイト・オペコード命令」～ A.3.4. 項「1 バイトと 2 バイトのオペコードのオペコード拡張」項では、1 バイトと 2 バイトのオペコード・マップを解釈するための命令を示している。

エスケープ・エンコーディングは、浮動小数点命令をコード化するために使用される。表 A-5. ～ A-20. に、これらの命令のオペコード・マップを示す。A.3.5. 項「エスケープ・オペコード命令」では、エスケープ・オペコード・マップを解釈するための命令を示している。

A.1. オペコード・テーブルの使用に関する注意

本付録の表では、基本オペコード（命令プリフィックスを含む）と ModR/M バイトについて定義する。表中の空白の項目は、予約済みまたは未定義のオペコードを示す。

基本オペコードの上位 4 ビットを、オペコードの表の行に対するインデックスとして使用する。オペコードの下位 4 ビットを、表の列に対するインデックスとして使用する。基本オペコードの最初のバイトが 0FH であるか、または 66H、F2H、F3H のいずれかの後に 0 FH がある場合は、2 バイトのオペコードの表を参照し、オペコードの第 2 バイトをその表の行と列に対するインデックスとして使用する。

ModR/M バイトにオペコード拡張が含まれる場合は、その命令は表 A-2、表 A-3 の命令グループである。ModR/M バイト内のオペコード拡張については、表 A-4 で説明する。

浮動小数点命令用のエスケープ（ESC）オペコードの表では、各ページの一番上にオペコードの上位 8 ビットを示す。オペコードに付随する ModR/M バイトが 00H ～ BFH の範囲内である場合は、各ページの 3 番目の表の一番上の行に示す ModR/M バイトのビット 3 ～ 5 と REG ビットによってオペコードが決まる。ModR/M バイトが 00H ～ BFH の範囲外である場合のオペコード・マップは、各ページの最後の 2 つの表に示す。

ModR/M バイト、レジスタ値、アドレス指定形式の詳細については、『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻 A』の第 2 章を参照のこと。

A.2. 略語の説明

オペランドは、形式 Zz の 2 文字のコードによって識別される。最初の文字（Z）は、アドレス指定方式を指定する。2 番目の文字（z）は、オペランドのタイプを指定する。

A.2.1. アドレス指定方式のコード

以下の略号は、アドレス指定方式に使用される。

A　直接アドレス。命令は、ModR/M バイトをもたない。オペランドのアドレスは、命令内にコード化されている。ベースレジスタ、インデックス・レジスタ、またはスケーリング・ファクタを適用することはできない。例えば、far JMP (EA)。

C　ModR/M バイトの reg フィールドで制御レジスタを選択する（例、MOV (0F20, 0F22)）。

D　ModR/M バイトの reg フィールドでデバッグレジスタを選択する。例えば、MOV (0F21,0F23)。

E　ModR/M バイトがオペコードの後に続き、オペランドを指定する。オペランドは、汎用レジスタまたはメモリアドレスである。オペランドがメモリアドレスである場合は、アドレスは、セグメント・レジスタと、ベースレジスタ、インデックス・レジスタ、スケーリング・ファクタ、ディスプレースメントの任意の組み合わせから計算される。

F　EFLAGS レジスタ。

G　ModR/M バイトの reg フィールドで汎用レジスタを選択する（例、AX(000)）。

I　即値データ。オペランド値は命令の後続バイト内にコード化される。

J　命令は、命令ポインタレジスタに付加する相対オフセットをもつ（例、JMP(0E9)、LOOP）。

M　ModR/M バイトは、メモリだけを参照することができる : mod!=11B（BOUND、LEA、LES、LDS、LSS、LFS、LGS、CMPXCHG8B、LDDQU）。

O　命令は、ModR/M バイトをもたない。オペランドのオフセットが命令内に（アドレスサイズ属性に応じて）ワードまたはダブルワードとして入れられる。ベースレジスタ、インデックス・レジスタ、またはスケーリング・ファクタを適用することはできない（例、MOV (A0-A3)）。

P　ModR/M バイトの reg フィールドでパック・クワッドワード MMX テクノロジ・レジスタを選択する。

Q　ModR/M バイトがオペコードの後に続き、オペランドを指定する。オペランドは、MMX テクノロジ・レジスタまたはメモリアドレスである。オペランドがメモリアドレスである場合は、アドレスは、セグメント・レジスタと、ベースレジスタ、インデックス・レジスタ、スケーリング・ファクタ、ディスプレースメントの任意の組み合わせから計算される。

R　ModR/M バイトの mod フィールドは、汎用レジスタだけを参照することがある（例、MOV (0F20-0F24, 0F26)）。

S　ModR/M バイトの reg フィールドでセグメント・レジスタを選択する（例、MOV (8C, 8E)）。

T　ModR/M バイトの reg フィールドでテストレジスタを選択する（例、MOV (0F24, 0F26)）。

V　ModR/M バイトの reg フィールドで 128 ビット XMM レジスタを選択する。

W　ModR/M バイトがオペコードの後に続き、オペランドを指定する。オペランドは 128 ビット XMM レジスタまたはメモリアドレスである。オペランドがメモリアドレスの場合、アドレスは、セグメント・レジスタと、ベースレジスタ、インデックス・レジスタ、スケーリング・レジスタ、ディスプレースメントの任意の組み合わせから計算される。

X　DS:SI　レジスタペアによってアドレス指定されるメモリ（例、MOVS、CMPS、OUTS、または LODS）。

Y　ES:DI　レジスタペアによってアドレス指定されるメモリ（例、MOVS、CMPS、INS、STOS、または SCAS）。

A.2.2. オペランド・タイプのコード

以下の略号は、オペランド・タイプに使用される。

a　メモリ内の 2 つの 1 ワード・オペランドまたはメモリ内の 2 つのダブルワード・オペランド。オペランド・サイズ属性に依存する（BOUND 命令だけに使用される）。

b　バイト。オペランド・サイズ属性に関係ない。

c　バイトまたはワード。オペランド・サイズ属性に依存する。

d　ダブルワード。オペランド・サイズ属性に関係ない。

dq　ダブル・クワッドワード。オペランド・サイズ属性に関係ない。

p　32 ビットまたは 48 ビットのポインタ。オペランド・サイズ属性に依存する。

pi　クワッドワードの MMX テクノロジ・レジスタ（例えば、mm0）。

pd　128 ビット・パックド倍精度浮動小数点データ。

ps　128 ビットのパックド単精度浮動小数点データ。

q　クワッドワード。オペランド・サイズ属性に関係ない。

s　6 バイトの疑似ディスクリプタ。

sd　128 ビット・パックド倍精度浮動小数点データのスカラ要素。

ss　128 ビットのパックド単精度浮動小数点データのスカラ要素。

si　ダブルワードの整数レジスタ（例えば、eax）。

v　ワードまたはダブルワード。オペランド・サイズ属性に依存する。

w　ワード。オペランド・サイズ属性に関係ない。

A.2.3. レジスタコード

オペランドがオペコード内にコード化された特定のレジスタであるときは、レジスタはその名前（例えば、AX、CL、ESI）によって識別される。レジスタの名前は、そのレジスタの幅が 32 ビット、16 ビット、または 8 ビットのいずれであるかを示す。レジスタの幅がオペランド・サイズ属性に依存する場合は、形式 eXX のレジスタ識別子が使用される。例えば、eAX は、オペランド・サイズ属性が 16 であるときは AX レジスタが使用され、オペランド・サイズ属性が 32 であるときは EAX レジスタが使用されることを示す。

A.3. オペコードの見つけ方の例

ここでは、オペコードマップの使い方の例をいくつか示す。

A.3.1. 1 バイト・オペコード命令

表 A-2. に、1 バイト・オペコードのオペコード・マップを示す。1 バイト・オペコード・マップでは、命令ニーモニックおよびそのオペランドは、16 進の 1 バイト・オペコードの値から決定できる。1 バイト・オペコードのオペコード・マップは、行（16 進値の最下位 4 ビット）と列（16 進値の最上位 4 ビット）で構成される。表中の各項目は、次のいずれかのタイプのオペコードを示す。

- 命令ニーモニックと、付録 A.2.2. に示した表記を使用するオペランド・タイプ
- 命令プリフィックスとして使用されるオペコード

1 つの命令に対応するオペコード・マップの各項目について、基本オペコードの後に続く次のバイトの解釈規則は、次のいずれかの条件に該当する。

- ModR/M バイトが必要である。ModR/M バイトは、付録 A.2. に示した省略形にしたがって解釈される。『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、中巻 A』の第 2 章を参照のこと。オペランド・タイプは、付録 A.2.2. に示した表記にしたがって示される。
- ModR/M バイトが必要である。ModR/M バイト内の reg フィールドにオペコード拡張が含まれる。ModR/M バイトの解釈には、表 A-4 を使用する。

- ModR/M バイトの使用は予約済みまたは未定義である。この条件は、命令プリフィックスを表す項目や、ModR/M に関連するオペランドを持たない命令を表す項目（例えば、60H, PUSHA や 06H, PUSH ES）に適用される。

例えば、以下のオペコード・シーケンスを見付ける場合を考える。

オペコード: 030500000000H

LSB アドレス					MSB アドレス
03	05	00	00	00	00

ADD 命令のオペコード 030500000000H は、1 バイト・オペコード・マップから次のように解釈することができる。すなわち、オペコードの 1 ケタ目（0）はオペコード・マップ・テーブルの行を示し、2 ケタ目 (3) は列を示す。第 1 オペランド（タイプ Gv）は、オペランド・サイズ属性に応じてワードまたはダブルワードである汎用レジスタを示している。第 2 オペランド（タイプ Ev）は、オペランドがワードまたはダブルワードの汎用レジスタあるいはメモリアドレスのいずれであるかを指定する ModR/M バイトが後に続くことを示している。この命令の ModR/M バイトは 05H であり、これは、32 ビットのディスプレースメント（00000000H）が後に続くことを示している。ModR/M バイトの reg/ オペコード部分（ビット 3 ～ 5）は、EAX レジスタを示す 000 である。したがって、このオペコードの命令は、ADD EAX,mem_op であり、mem_op のオフセットは 00000000H であると決定することができる。

1 バイトと 2 バイトのオペコードは、「グループ」番号を指す場合もある。これらのグループ番号は、命令が ModR/M バイトの reg/ オペコード・ビットをオペコード拡張として使用することを示している（A.3.4. 項「1 バイトと 2 バイトのオペコードのオペコード拡張」を参照）。

A.3.2. 2 バイト・オペコード命令

表 A-3 の 2 バイト・オペコード・マップには、長さ 2 バイトまたは 3 バイトの基本オペコードが含まれている。長さ 2 バイトの基本オペコードは、エスケープ・オペコード 0FH から始まる。第 2 バイトの上位 4 ビットと下位 4 ビットは、表 A-3 の特定の行と列へのインデックスとして使用される。長さ 3 バイトの 2 バイト・オペコードは、必須のプリフィックス（66H、F2H、または F3H）から始まり、エスケープ・オペコードが後に続く。第 3 バイトの上位 4 ビットと下位 4 ビットは、表 A-3 の特定の行と列へのインデックスとして使用される。2 バイトのエスケープ・シーケンスは、必須のプリフィックス（66H、F2H、または F3H）と、その後に続くエスケープ・プリフィックス・バイト 0FH で構成される。

オペコード・マップの各項目について、基本オペコードの後に続く次のバイトの解釈規則は、次のいずれかの条件に該当する。

- ModR/M バイトが必要である。ModR/M バイトは、付録 A.2. に示した省略形にしたがって解釈される。ModR/M バイト、レジスタ値、および各種のアドレス指定形式については、『IA-32 インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻 A』の第 2 章を参照のこと。オペランド・タイプは、付録 A.2.2. に示した表記にしたがって示される。
- ModR/M バイトが必要である。ModR/M バイト内の reg フィールドにオペコード拡張が含まれる。ModR/M バイトの解釈には、表 A-4 を使用する。
- ModR/M バイトの使用は予約済みまたは未定義である。この条件は、ModR/M を使用してエンコードされるオペランドを持たない命令を表す項目（例えば、0F77H, EMMS）に適用される。

例えば、オペコード 0FA4050000000003H は、2 バイトのオペコード・マップの行 A、列 4 にある。このオペコードは、オペランド Ev、Gv、および Ib の SHLD 命令を指している。これらのオペランドは、次のように解釈される。

Ev　ワードまたはダブルワードのオペランドを指定する ModR/M バイトがオペコードの後に続く。

Cv　ModR/M バイトの reg フィールドが汎用レジスタを選択する。

Ib　即値データが命令の後続バイト内にコード化されている。

第 3 バイトは、ModR/M バイト（05H）である。mod およびオペコード/reg フィールドは、32 ビットのディスプレースメントが後に続くことおよび EAX レジスタがソースであることを示している。

オペコードの次の部分は、デスティネーション・メモリ・オペランドの 32 ビット・ディスプレースメント（00000000H）である。最後の部分は、シフトのカウントを表す即値バイト（03H）である。

このような分析により、このオペコードが次の命令を表すことが示される。

SHLD DS:00000000H, EAX, 3

SHLD オペコードの次の部分は、デスティネーション・メモリ・オペランドの 32 ビット・ディスプレースメント（00000000H）であり、この後に、シフトのカウントを表す即値バイト（03H）が続く。このブレークダウンによって、オペコード 0FA4050000000003H が命令 SHLD DS: 00000000H, EAX, 3 を表すことが示されている。

SHLD DS:00000000H, EAX, 3.

以下の表では、MMX®テクノロジ、SSE、およびSSE2によって追加されたニーモニックを小文字で強調する。

A.3.3. オペコード・マップの注意事項

表A-1.は、オペコード・マップ表の特定のエンコーディングについての注意事項を示している。以下のオペコード・マップ（表A-2.とA-3.）では、これらの注意を上付き文字で示す。

1バイト・オペコード・マップ（表A-2.）については、グレー表示で命令グループを示す。

表 A-1. オペコード・マップ表の命令エンコーディングについての注意事項

記号	注意
1A	ModR/M バイトのビット 5、4、3 をオペコード拡張として使用（A.3.4. 項「1 バイトと 2 バイトのオペコードのオペコード拡張」を参照のこと）。
1B	故意に無効オペコード例外（#UD）を発生させる場合は、0F0B オペコード（UD2 命令）または 0FB9H オペコードを使用する。
1C	インテル® Pentium® III プロセッサに追加された命令の中には、同じ 2 バイト・オペコードを使用するものがある。命令にバリエーションがある場合、またはオペコードが別々の命令を表す場合は、その命令を区別するために ModR/M バイトが使用される。命令を完全にデコードするために必要な ModR/M バイトの値については、表 A-4. を参照のこと（このような命令としては、PREFETCH およびそのバリエーションだけではなく、SFENCE、STMXCSR、LDMXCSR、FXRSTOR、FXSAVE がある）。
1D	このオペコード表現で表される命令は、基本オペコードの後に続く ModR/M バイトを持たない。
1E	ModR/M バイトの r/m フィールドの有効なエンコーディングは、カッコ内に示される。
1F	このオペコード表現で表される命令は、ソース・オペランドとデスティネーション・オペランドの両方がレジスタである場合をサポートしない。
1G	ソース・オペランドがレジスタの場合、そのオペランドは XMM レジスタでなければならない。
1H	このオペコード表現で表される命令は、オペランドがメモリ上の位置である場合をサポートしない。
1J	このオペコード表現で表される命令は、レジスタ・オペランドをサポートしない。
1K	ModR/M バイトの reg/ オペコード・フィールドの有効なエンコーディングは、カッコ内に示される。

表 A-2. 1 バイトのオペコード・マップ [1,2]

	0	1	2	3	4	5	6	7
0	ADD						PUSH ES[1D]	POP ES[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
1	ADC						PUSH SS[1D]	POP SS[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
2	AND						SEG=ES Prefix	DAA[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
3	XOR						SEG=SS Prefix	AAA[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
4	INC general register							
	eAX[1D]	eCX[1D]	eDX[1D]	eBX[1D]	eSP[1D]	eBP[1D]	eSI[1D]	eDI[1D]
5	PUSH general register[1D]							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSHA/ PUSHAD[1D]	POPA/ POPAD[1D]	BOUND Gv, Ma	ARPL Ew, Gw	SEG=FS Prefix	SEG=GS Prefix	Opd Size Prefix	Addr Size Prefix
7	Jcc, Jb - Short-displacement jump on condition							
	O[1D]	NO[1D]	B/NAE/C[1D]	NB/AE/NC[1D]	Z/E[1D]	NZ/NE[1D]	BE/NA[1D]	NBE/A[1D]
8	Immediate Grp 1[1A]				TEST		XCHG	
	Eb, Ib	Ev, Iv	Eb, Ib	Ev, Ib	Eb, Gb	Ev, Gv	Eb, Gb	Ev, Gv
9	NOP[1D]	XCHG word or double-word register with eAX[1D]						
		eCX	eDX	eBX	eSP	eBP	eSI	eDI
A	MOV[1D]				MOVS/ MOVSB Yb, Xb[1D]	MOVS/ MOVSW/ MOVSD Yv, Xv[1D]	CMPS/ CMPSB Yb, Xb[1D]	CMPS/ CMPSW/ CMPSD Xv, Yv[1D]
	AL, Ob	eAX, Ov	Ob, AL	Ov, eAX				
B	MOV immediate byte into byte register[1D]							
	AL	CL	DL	BL	AH	CH	DH	BH
C	Shift Grp 2[1A]		RET Iw[1D]	RET[1D]	LES Gv, Mp	LDS Gv, Mp	Grp 11[1A] - MOV	
	Eb, Ib	Ev, Ib					Eb, Ib	Ev, Iv
D	Shift Grp 2[1A]				AAM Ib[1D]	AAD Ib[1D]		XLAT/ XLATB[1D]
	Eb, 1	Ev, 1	Eb, CL	Ev, CL				
E	LOOPNE/ LOOPNZ Jb[1D]	LOOPE/ LOOPZ Jb[1D]	LOOP Jb[1D]	JCXZ/ JECXZ Jb[1D]	IN		OUT	
					AL, Ib[1D]	eAX, Ib[1D]	Ib, AL[1D]	Ib, eAX[1D]
F	LOCK Prefix		REPNE Prefix	REP/ REPE Prefix	HLT[1D]	CMC[1D]	Unary Grp 3[1A]	
							Eb	Ev

注 :

1. オペコード・マップ表 A-2. にあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードまたは予約されているオペコードの動作に依存してはならない。
2. この表では、行番号がオペコードの最初の 16 進数文字に対応し、列番号が 2 番目の 16 進数文字に対応する。例えば、[POP ES] のオペコードは 07H になる。

表 A-2. 1 バイトのオペコード・マップ（続き）

	8	9	A	B	C	D	E	F
0	OR						PUSH CS[1D]	Escape opcode to 2-byte
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
1	SBB						PUSH DS[1D]	POP DS[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
2	SUB						SEG=CS Prefix	DAS[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
3	CMP						SEG=DS Prefix	AAS[1D]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib[1D]	eAX, Iv[1D]		
4	DEC general register							
	eAX[1D]	eCX[1D]	eDX[1D]	eBX[1D]	eSP[1D]	eBP[1D]	eSI[1D]	eDI[1D]
5	POP into general register[1D]							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSH Iv[1D]	IMUL Gv, Ev, Iv	PUSH Ib[1D]	IMUL Gv, Ev, Ib	INS/ INSB Yb, DX[1D]	INS/ INSW/ INSD Yv, DX[1D]	OUTS/ OUTSB DX, Xb[1D]	OUTS/ OUTSW/ OUTSD DX, Xv[1D]
7	Jcc, Jb- Short displacement jump on condition							
	S[1D]	NS[1D]	P/PE[1D]	NP/PO[1D]	L/NGE[1D]	NL/GE[1D]	LE/NG[1D]	NLE/G[1D]
8	MOV				MOV Ew, Sw	LEA Gv, M	MOV Sw, Ew	POP Ev
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev				
9	CBW/ CWDE[1D]	CWD/ CDQ[1D]	CALLF Ap[1D]	FWAIT/ WAIT[1D]	PUSHF/ PUSHFD Fv[1D]	POPF/ POPFD Fv[1D]	SAHF[1D]	LAHF[1D]
A	TEST[1D]		STOS/ STOSB Yb, AL [1D]	STOS/ STOSW/ STOSD Yv, eAX[1D]	LODS/ LODSB AL, Xb[1D]	LODS/ LODSW/ LODSD eAX, Xv[1D]	SCAS/ SCASB AL, Yb[1D]	SCAS/ SCASW/ SCASD eAX, Yv[1D]
	AL, Ib	eAX, Iv						
B	MOV immediate word or double into word or double register[1D]							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
C	ENTER Iw, Ib[1D]	LEAVE[1D]	RETF Iw[1D]	RETF[1D]	INT 3[1D]	INT Ib[1D]	INTO[1D]	IRET[1D]
D	ESC (Escape to coprocessor instruction set)							
E	CALL	JMP			IN		OUT	
	Jv[1D]	near Jv[1D]	far Ap[1D]	short Jb[1D]	AL, DX[1D]	eAX, DX[1D]	DX, AL[1D]	DX, eAX[1D]
F	CLC[1D]	STC[1D]	CLI[1D]	STI[1D]	CLD[1D]	STD[1D]	INC/DEC Grp 4[1A]	INC/DEC Grp 5[1A]

表 A-3. 2 バイトのオペコード・マップ（第 1 バイトは 0FH）[1,2]

	0	1	2	3	4	5	6	7
0	Grp 6[1A]	Grp 7[1A]	LAR Gv, Ew	LSL Gv, Ew			CLTS[1D]	
1	MOVUPS Vps, Wps MOVSS (F3) Vss, Wss MOVUPD (66) Vpd, Wpd MOVSD (F2) Vsd, Wsd	MOVUPS Wps, Vps MOVSS (F3) Wss, Vss MOVUPD (66) Wpd, Vpd MOVSD (F2) Wsd, Vsd	MOVLPS Vq, Mq[1F] MOVLPD (66) Vq, Mq[1F] MOVHLPS Vps, Vps MOVDDUP (F2) Vq, Wq[1G] MOVSLDUP (F3) Vps, Wps	MOVLPS Mq, Vq[1F] MOVLPD (66) Mq, Vq[1F]	UNPCKLPS Vps, Wps UNPCKLPD (66) Vpd, Wpd	UNPCKHPS Vps, Wps UNPCKHPD (66) Vpd, Wpd	MOVHPS Vq, Mq[1F] MOVHPD (66) Vq, Mq[1F] MOVLHPS Vps, Vps MOVSHDUP (F3) Vps, Wps	MOVHPS Mq, Vps[1F] MOVHPD (66) Mq, Vpd[1F]
2	MOV Rd, Cd[1H]	MOV Rd, Dd[1H]	MOV Cd, Rd[1H]	MOV Dd, Rd[1H]	MOV Rd, Td[3]		MOV Td, Rd[3]	
3	WRMSR[1D]	RDTSC[1D]	RDMSR[1D]	RDPMC[1D]	SYSENTER[1D]	SYSEXIT[1D]		
4	CMOVcc, (Gv, Ev) - Conditional Move							
	O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
5	MOVMSKPS Gd, Vps[1H] MOVMSKPD (66) Gd, Vpd[1H]	SQRTPS Vps, Wps SQRTSS (F3) Vss, Wss SQRTPD (66) Vpd, Wpd SQRTSD (F2) Vsd, Wsd	RSQRTPS Vps, Wps RSQRTSS (F3) Vss, Wss	RCPPS Vps, Wps RCPSS (F3) Vss, Wss	ANDPS Vps, Wps ANDPD (66) Vpd, Wpd	ANDNPS Vps, Wps ANDNPD (66) Vpd, Wpd	ORPS Vps, Wps ORPD (66) Vpd, Wpd	XORPS Vps, Wps XORPD (66) Vpd, Wpd
6	PUNPCKLBW Pq, Qd PUNPCKLBW (66) Vdq, Wdq	PUNPCKLWD Pq, Qd PUNPCKLWD (66) Vdq, Wdq	PUNPCKLDQ Pq, Qd PUNPCKLDQ (66) Vdq, Wdq	PACKSSWB Pq, Qq PACKSSWB (66) Vdq, Wdq	PCMPGTB Pq, Qq PCMPGTB (66) Vdq, Wdq	PCMPGTW Pq, Qq PCMPGTW (66) Vdq, Wdq	PCMPGTD Pq, Qq PCMPGTD (66) Vdq, Wdq	PACKUSWB Pq, Qq PACKUSWB (66) Vdq, Wdq
7	PSHUFW Pq, Qq, Ib PSHUFD (66) Vdq, Wdq, Ib PSHUFHW (F3) Vdq, Wdq, Ib PSHUFLW (F2) Vdq, Wdq, Ib	(Grp 12[1A])	(Grp 13[1A])	(Grp 14[1A])	PCMPEQB Pq, Qq PCMPEQB (66) Vdq, Wdq	PCMPEQW Pq, Qq PCMPEQW (66) Vdq, Wdq	PCMPEQD Pq, Qq PCMPEQD (66) Vdq, Wdq	EMMS[1D]

注 :

1. オペコード・マップ表 A-3. にあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードまたは予約されているオペコードの動作に依存してはならない。
2. この表では、オペコードの第 1 バイトには 0FH を使用する。第 2 バイトについては、行番号が最初の 16 進数文字に対応し、列番号が 2 番目の 16 進数文字に対応する。例えば、[LSL GV, EW] のオペコードは 0F03H になる。
3. 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・ファミリ以降、現在ではサポートしていない。現在の世代のプロセッサ上でこのオペコードを使用すると、#UD が発生する。将来のプロセッサでは、この値は予約されている。

表 A-3. 2 バイトのオペコード・マップ（先頭バイトは 0FH）（続き）

	8	9	A	B	C	D	E	F
0	INVD[1D]	WBINVD[1D]		UD2				
1	PREFETCH[1C] (Grp 16[1A])							
2	MOVAPS Vps, Wps MOVAPD (66) Vpd, Wpd	MOVAPS Wps, Vps MOVAPD (66) Wpd, Vpd	CVTPI2PS Vps, Qq CVTSI2SS (F3) Vss, Ed CVTPI2PD (66) Vpd, Qq CVTSI2SD (F2) Vsd, Ed	MOVNTPS Mps, Vps[1F] MOVNTPD (66) Mpd, Vpd[1F]	CVTTPS2PI Pq, Wq CVTTSS2SI (F3) Gd, Wss CVTTPD2PI (66) Pq, Wpd CVTTSD2SI (F2) Gd, Wsd	CVTPS2PI Pq, Wq CVTSS2SI (F3) Gd, Wss CVTPD2PI (66) Pq, Wpd CVTSD2SI (F2) Gd, Wsd	UCOMISS Vss, Wss UCOMISD (66) Vsd, Wsd	COMISS Vps, Wps COMISD (66) Vsd, Wsd
3								
4	CMOVcc(Gv, Ev) - Conditional Move							
	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5	ADDPS Vps, Wps ADDSS (F3) Vss, Wss ADDPD (66) Vpd, Wpd ADDSD (F2) Vsd, Wsd	MULPS Vps, Wps MULSS (F3) Vss, Wss MULPD (66) Vpd, Wpd MULSD (F2) Vsd, Wsd	CVTPS2PD Vpd, Wq CVTSS2SD (F3) Vsd, Wss CVTPD2PS (66) Vps, Wpd CVTSD2SS (F2) Vss, Wsd	CVTDQ2PS Vps, Wdq CVTPS2DQ (66) Vdq, Wps CVTTPS2DQ (F3) Vdq, Wps	SUBPS Vps, Wps SUBSS (F3) Vss, Wss SUBPD (66) Vpd, Wpd SUBSD (F2) Vsd, Wsd	MINPS Vps, Wps MINSS (F3) Vss, Wss MINPD (66) Vpd, Wpd MINSD (F2) Vsd, Wsd	DIVPS Vps, Wps DIVSS (F3) Vss, Wss DIVPD (66) Vpd, Wpd DIVSD (F2) Vsd, Wsd	MAXPS Vps, Wps MAXSS (F3) Vss, Wss MAXPD (66) Vpd, Wpd MAXSD (F2) Vsd, Wsd
6	PUNPCKHBW Pq, Qq PUNPCKHBW (66) Vdq, Qdq	PUNPCKHWD Pq, Qq PUNPCKHWD (66) Vdq, Qdq	PUNPCKHDQ Pq, Qq PUNPCKHDQ (66) Vdq, Qdq	PACKSSDW Pq, Qq PACKSSDW (66) Vdq, Qdq	PUNPCKLQDQ (66) Vdq, Wdq	PUNPCKHQDQ (66) Vdq, Wdq	MOVD Pd, Ed MOVD (66) Vd, Ed	MOVQ Pq, Qq MOVDQA (66) Vdq, Wdq MOVDQU (F3) Vdq, Wdq
7	MMX UD (Reserved for future use)				HADDPD (66) Vpd, Wpd HADDPS (F2) Vps, Wps	HSUBPD (66) Vpd, Wpd HSUBPS (F2) Vps, Wps	MOVD Ed, Pd MOVD (66) Ed, Vd MOVQ (F3) Vq, Wq	MOVQ Qq, Pq MOVDQA (66) Wdq, Vdq MOVDQU (F3) Wdq, Vdq

表 A-3. 2 バイトのオペコード・マップ（先頭バイトは 0FH）（続き）

	0	1	2	3	4	5	6	7
8	Jcc, Jv - Long-displacement jump on condition							
	O[1D]	NO[1D]	B/C/NAE[1D]	AE/NB/NC[1D]	E/Z[1D]	NE/NZ[1D]	BE/NA[1D]	A/NBE[1D]
9	SETcc, Eb - Byte Set on condition (000)[1K]							
	O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
A	PUSH FS[1D]	POP FS[1D]	CPUID[1D]	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL		
B	CMPXCHG Eb, Gb	CMPXCHG Ev, Gv	LSS Mp	BTR Ev, Gv	LFS Mp	LGS Mp	MOVZX Gv, Eb	MOVZX Gv, Ew
C	XADD Eb, Gb	XADD Ev, Gv	CMPPS Vps, Wps, Ib CMPSS (F3) Vss, Wss, Ib CMPPD (66) Vpd, Wpd, Ib CMPSD (F2) Vsd, Wsd, Ib	MOVNTI Md, Gd[1F]	PINSRW Pw, Ew, Ib PINSRW (66) Vw, Ew, Ib	PEXTRW Gw, Pw, Ib[1H] PEXTRW (66) Gw, Vw, Ib[1H]	SHUFPS Vps, Wps, Ib SHUFPD (66) Vpd, Wpd, Ib	Grp 9[1A]
D	ADDSUBPD (66) Vpd, Wpd ADDSUBPS (F2) Vps, Wps	PSRLW Pq, Qq PSRLW (66) Vdq, Wdq	PSRLD Pq, Qq PSRLD (66) Vdq, Wdq	PSRLQ Pq, Qq PSRLQ (66) Vdq, Wdq	PADDQ Pq, Qq PADDQ (66) Vdq, Wdq	PMULLW Pq, Qq PMULLW (66) Vdq, Wdq	MOVQ (66) Wq, Vq MOVQ2DQ (F3) Vdq, Qq[1H] MOVDQ2Q (F2) Pq, Vq[1H]	PMOVMSKB Gd, Pq[1H] PMOVMSKB (66) Gd, Vdq[1H]
E	PAVGB Pq, Qq PAVGB (66) Vdq, Wdq	PSRAW Pq, Qq PSRAW (66) Vdq, Wdq	PSRAD Pq, Qq PSRAD (66) Vdq, Wdq	PAVGW Pq, Qq PAVGW (66) Vdq, Wdq	PMULHUW Pq, Qq PMULHUW (66) Vdq, Wdq	PMULHW Pq, Qq PMULHW (66) Vdq, Wdq	CVTPD2DQ (F2) Vdq, Wpd CVTTPD2DQ (66) Vdq, Wpd CVTDQ2PD (F3) Vpd, Wq	MOVNTQ Mq, Vq[1F] MOVNTDQ (66) Mdq, Vdq[1F]
F	LDDQU (F2) Vdq, Mdq	PSLLW Pq, Qq PSLLW (66) Vdq, Wdq	PSLLD Pq, Qq PSLLD (66) Vdq, Wdq	PSLLQ Pq, Qq PSLLQ (66) Vdq, Wdq	PMULUDQ Pq, Qq PMULUDQ (66) Vdq, Wdq	PMADDWD Pq, Qq PMADDWD (66) Vdq, Wdq	PSADBW Pq, Qq PSADBW (66) Vdq, Wdq	MASKMOVQ Pq, Pq[1H] MASKMOV-DQU (66) Vdq, Vdq[1H]

表 A-3. 2 バイトのオペコード・マップ（先頭バイトは 0FH）（続き）

	8	9	A	B	C	D	E	F
8	Jcc, Jv - Long-displacement jump on condition							
	S[1D]	NS[1D]	P/PE[1D]	NP/PO[1D]	L/NGE[1D]	NL/GE[1D]	LE/NG[1D]	NLE/G[1D]
9	SETcc, Eb - Byte Set on condition (000)[1K]							
	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
A	PUSH GS[1D]	POP GS[1D]	RSM[1D]	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	(Grp 15[1A])[1C]	IMUL Gv, Ev
B		Grp 10[1A] Invalid Opcode[1B]	Grp 8[1A] Ev, Ib	BTC Ev, Gv	BSF Gv, Ev	BSR Gv, Ev	MOVSX Gv, Eb	MOVSX Gv, Ew
C	BSWAP[1D]							
	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
D	PSUBUSB Pq, Qq PSUBUSB (66) Vdq, Wdq	PSUBUSW Pq, Qq PSUBUSW (66) Vdq, Wdq	PMINUB Pq, Qq PMINUB (66) Vdq, Wdq	PAND Pq, Qq PAND (66) Vdq, Wdq	PADDUSB Pq, Qq PADDUSB (66) Vdq, Wdq	PADDUSW Pq, Qq PADDUSW (66) Vdq, Wdq	PMAXUB Pq, Qq PMAXUB (66) Vdq, Wdq	PANDN Pq, Qq PANDN (66) Vdq, Wdq
E	PSUBSB Pq, Qq PSUBSB (66) Vdq, Wdq	PSUBSW Pq, Qq PSUBSW (66) Vdq, Wdq	PMINSW Pq, Qq PMINSW (66) Vdq, Wdq	POR Pq, Qq POR (66) Vdq, Wdq	PADDSB Pq, Qq PADDSB (66) Vdq, Wdq	PADDSW Pq, Qq PADDSW (66) Vdq, Wdq	PMAXSW Pq, Qq PMAXSW (66) Vdq, Wdq	PXOR Pq, Qq PXOR (66) Vdq, Wdq
F	PSUBB Pq, Qq PSUBB (66) Vdq, Wdq	PSUBW Pq, Qq PSUBW (66) Vdq, Wdq	PSUBD Pq, Qq PSUBD (66) Vdq, Wdq	PSUBQ Pq, Qq PSUBQ (66) Vdq, Wdq	PADDB Pq, Qq PADDB (66) Vdq, Wdq	PADDW Pq, Qq PADDW (66) Vdq, Wdq	PADDD Pq, Qq PADDD (66) Vdq, Wdq	

A.3.4. 1 バイトと 2 バイトのオペコードのオペコード拡張

1 バイトと 2 バイトのオペコードの一部は、ModR/M バイトのビット 5、4、3（図 A-1. の nnn フィールド）をオペコードの拡張として使用する。ModR/M バイトのビット 5、4、3 の値は、第 3 章で説明したオペコード表記の “/digit” の部分にも対応する。オペコード拡張をもつオペコードは、表 A-4. でグループ番号（グループ 1、グループ 2 など）で示している。2 列目の（1 から 16 までの範囲の）グループ番号は、オペコード拡張フィールドのエンコーディングがある表 A-4. へのエントリポイントを提供する。各命令の ModR/M バイトの r/m フィールドの有効なエンコーディングは、3 番目の列から推測できる。

例えば、80H の 1 バイト・オペコードをもつ ADD 命令は、グループ 1 の命令である。表 A-4 は、この命令の ModR/M バイト内にエンコードされていなければならないオペコード拡張フィールドが 000B であることを示している。この命令の r/m フィールドは、レジスタにアクセスするようにエンコードすることも（11B）、アドレス指定モードを使用してメモリアドレスにアクセスするようにエンコードすることもできる（例えば、mem = 00B、01B、10B）。

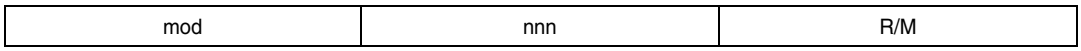


図 A-1. ModR/M バイトの nnn フィールド（ビット 5、4、3）

表 A-4. グループ番号による 1 バイトと 2 バイトのオペコードのオペコード拡張 [1]

			ModR/M バイトのビット 5、4、3 のエンコーディング							
オペコード	グループ	Mod 7,6	000	001	010	011	100	101	110	111
80-83	1	mem, 11B	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
C0, C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B	ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B	TEST Ib/Iv		NOT	NEG	MUL AL/eAX	IMUL AL/eAX	DIV AL/eAX	IDIV AL/eAX
FE	4	mem, 11B	INC Eb	DEC Eb						
FF	5	mem, 11B	INC Ev	DEC Ev	CALLN Ev	CALLF Ep [1J]	JMPN Ev	JMPF Ep[1J]	PUSH Ev	
OF OO	6	mem, 11B	SLDT Ew	STR Ev	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
OF 01	7	mem	SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Ew		LMSW Ew	INVLPG Mb
		11B		MONITOR eAX, eCX, eDX (000)1E MWAIT eAX, eCX (001)1E						
OF BA	8	mem, 11B					BT	BTS	BTR	BTC
OF C7	9	mem		CMPXCH8B Mq						
		11B								

表 A-4. グループ番号による 1 バイトと 2 バイトのオペコードのオペコード拡張 [1]（続き）

オペコード	グループ	Mod 7,6	ModR/M バイトのビット 5、4、3 のエンコーディング							
			000	001	010	011	100	101	110	111
OF B9	10	mem								
		11B								
C6	11	mem, 11B	MOV Eb, Ib							
C7		mem, 11B	MOV Ev, Iv							
OF 71	12	mem								
		11B			PSRLW Pq, Ib PSRLW (66) Pdq, Ib		PSRAW Pq, Ib PSRAW (66) Pdq, Ib		PSLLW Pq, Ib PSLLW (66) Pdq, Ib	
OF 72	13	mem								
		11B			PSRLD Pq, Ib PSRLD (66) Wdq, Ib		PSRAD Pq, Ib PSRAD (66) Wdq, Ib		PSLLD Pq, Ib PSLLD (66) Wdq, Ib	
OF 73	14	mem								
		11B			PSRLQ Pq, Ib PSRLQ (66) Wdq, Ib	PSRLDQ (66) Wdq, Ib			PSLLQ Pq, Ib PSLLQ (66) Wdq, Ib	PSLLDQ (66) Wdq, Ib
OF AE	15	mem	FXSAVE	FXRSTOR	LDMXCSR	STMXCSR				CLFLUSH
		11B						LFENCE (000) [1E]	MFENCE (000) [1E]	SFENCE (000) [1E]
OF 18	16	mem	PREFETCH-NTA	PREFETCH-T0	PREFETCH-T1	PREFETCH-T2				
		11B								

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードまたは予約されているオペコードの動作に依存してはならない。

A.3.5. エスケープ・オペコード命令

表 A-5. ～ A-20. に、算術演算コプロセッサのエスケープ命令オペコード（x87 浮動小数点命令オペコード）のオペコード・マップを示す。これらのオペコード・マップは、D8 ～ DF のオペコードの第 1 バイトによってグループ化される。これらのオペコードはそれぞれ、ModR/M バイトをもつ。ModR/M バイトが 00H ～ BFH の範囲内にある場合は、ModR/M バイトのビット 5、4、3 がオペコード拡張として使用され、1 バイトと 2 バイトのオペコードに使用された技法と同様である（A.3.4. 項「1 バイトと 2 バイトのオペコードのオペコード拡張」を参照）。ModR/M バイトが 00H ～ BFH の範囲外にある場合は、ModR/M バイト全体がオペコード拡張として使用される。

A.3.5.1. ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード

オペコード DD0504000000H は、次のように解釈することができる。すなわち、このオペコードでコード化される命令は、A.3.5.8. 項「第 1 バイトとして DD をもつエスケープ・オペコード」にある。ModR/M バイト（05H）は、00H ～ BFH の範囲内にあるので、このバイトのビット 3 ～ 5 は、オペコードが FLD 倍精度実数命令用であることを示している（表 A-7. を参照）。ロードされる倍精度実数値は、00000004H にであり、これは、このオペコードの後に続き、このオペコードに属している 32 ビットのディスプレースメントである。

A.3.5.2. ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード

オペコード D8C1H は、00H ～ BFH の範囲外にある ModR/M バイトをもつオペコードの例を示している。ここでコード化される命令は、A.3.4. 項「1 バイトと 2 バイトのオペコードのオペコード拡張」にある。表 A-6. で、ModR/M バイト C1H は行 C 列 1 を示し、これは ST(0)、ST(1) をオペランドとして使用する FADD 命令である。

A.3.5.3. 第 1 バイトとして D8 をもつエスケープ・オペコード

表 A-5. と A-6. は、D8H で始まるエスケープ命令オペコードのオペコード・マップである。表 A-5. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-5. ModR/M バイトが 00H ～ BFH 内にあるときの D8 オペコード・マップ [1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-6. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-6. ModR/M バイトが 00H ～ BFH 外にあるときの D8 オペコード・マップ [1]

	0	1	2	3	4	5	6	7
C	FADD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOM							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIV							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOMP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUBR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIVR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.4. 第 1 バイトとして D9 をもつエスケープ・オペコード

表 A-7. と A-8. は、D9H で始まるエスケープ命令オペコードのオペコード・マップである。表 A-7. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-7. ModR/M バイトが 00H ～ BFH 内にあるときの D9 オペコード・マップ [1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B
FLD 単精度実数		FST 単精度実数	FSTP 単精度実数	FLDENV 14/28 バイト	FLDCW 2 バイト	FSTENV 14/28 バイト	FSTCW 2 バイト

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-8. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-8. ModR/M バイトが 00H ～ BFH 外にあるときの D9 オペコード・マップ [1]

	0	1	2	3	4	5	6	7
C	FLD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FNOP							
E	FCHS	FABS			FTST	FXAM		
F	F2XM1	FYL2X	FPTAN	FPATAN	FXTRACT	FPREM1	FDECSTP	FINCSTP

	8	9	A	B	C	D	E	F
C	FXCH							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D								
E	FLD1	FLDL2T	FLDL2E	FLDPI	FLDLG2	FLDLN2	FLDZ	
F	FPREM	FYL2XP1	FSQRT	FSINCOS	FRNDINT	FSCALE	FSIN	FCOS

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.5. 第 1 バイトとして DA をもつエスケープ・オペコード

表 A-9. と A-10. は、DAH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-9. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-9. ModR/M バイトが 00H ～ BFH 内にあるときの DA オペコード・マップ [1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B
FIADD dword 整数	FIMUL dword 整数	FICOM dword 整数	FICOMP dword 整数	FISUB dword 整数	FISUBR dword 整数	FIDIV dword 整数	FIDIVR dword 整数

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-10. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-10. ModR/M バイトが 00H ～ BFH 外にあるときの DA オペコード・マップ [1]

	0	1	2	3	4	5		7
C	FCMOVB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E								
F								

	8	9	A	B	C	D	E	F
C	FCMOVE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E		FUCOMPP						
F								

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.6. 第 1 バイトとして DB をもつエスケープ・オペコード

表 A-11. と A-12. は、DBH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-11. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-11. ModR/M バイトが 00H ～ BFH 内にあるときの DB オペコード・マップ[1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B
FILD dword 整数	FISTTP dword 整数	FIST dword 整数	FISTP dword 整数		FLD 拡張実数		FSTP 拡張実数

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-12. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-12. ModR/M バイトが 00H ～ BFH 外にあるときの DB オペコード・マップ[1]

	0	1	2	3	4	5		7
C	FCMOVNB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E			FNCLEX	FNINIT				
F	FCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FCMOVNE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FUCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.7. 第 1 バイトとして DC をもつエスケープ・オペコード

表 A-13. と A-14. は、DCH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-13. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-13. ModR/M バイトが 00H ～ BFH 内にあるときの DC オペコード・マップ[1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B
FADD 倍精度実数	FMUL 倍精度実数	FCOM 倍精度実数	FCOMP 倍精度実数	FSUB 倍精度実数	FSUBR 倍精度実数	FDIV 倍精度実数	FDIVR 倍精度実数

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-14. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-14. ModR/M バイトが 00H ～ BFH 外にあるときの DC オペコード・マップ[1]

	0	1	2	3	4	5		7
C	FADD							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUB							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIV							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.8. 第 1 バイトとして DD をもつエスケープ・オペコード

表 A-15. と A-16. は、DDH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-15. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-15. ModR/M バイトが 00H ～ BFH 内にあるときの DD オペコード・マップ[1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-16. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-16. ModR/M バイトが 00H ～ BFH 外にあるときの DD オペコード・マップ[1]

	0	1	2	3	4	5		7
C	FFREE							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
D	FST							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOM							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F								

	8	9	A	B	C	D	E	F
C								
D	FSTP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOMP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
F								

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.9. 第 1 バイトとして DE をもつエスケープ・オペコード

表 A-17. と A-18. は、DEH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-17. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-17. ModR/M バイトが 00H ～ BFH 内にあるときの DE オペコード・マップ[1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-18. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-18. ModR/M バイトが 00H ～ BFH 外にあるときの DE オペコード・マップ[1]

	0	1	2	3	4	5		7
C	FADDP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMULP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D		FCOMPP						
E	FSUBP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

注：

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

A.3.5.10. 第 1 バイトとして DF をもつエスケープ・オペコード

表 A-19. と A-20. は、DFH で始まるエスケープ命令オペコードのオペコード・マップである。表 A-19. に、付随する ModR/M バイトが 00H ～ BFH の範囲内にある場合のオペコード・マップを示す。ここでは、ビット 5、4、3（図 A-1. の nnn フィールド）の値が命令を選択する。

表 A-19. ModR/M バイトが 00H ～ BFH 内にあるときの DF オペコード・マップ [1]

ModR/M バイトの nnn フィールド（図 A-1. を参照）							
000B	001B	010B	011B	100B	101B	110B	111B
FILD ワード整数	FISTTP ワード整数	FIST ワード整数	FISTP ワード整数	FBLD パックド BCD	FILD qword 整数	FBSTP パックド BCD	FISTP qword 整数

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

表 A-20. に、付随する ModR/M バイトが 00H ～ BFH の範囲外にある場合のオペコード・マップを示す。この場合は、ModR/M バイトの 1 ケタ目が表の行を選択し、2 ケタ目が列を選択する。

表 A-20. ModR/M バイトが 00H ～ BFH 外にあるときの DF オペコード・マップ [1]

	0	1	2	3	4	5		7
C								
D								
E	FSTSW AX							
F	FCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C								
D								
E	FUCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

注 :

1. オペコード・マップにあるすべての空白は、予約されており、使用してはならない。これらの未定義オペコードの動作に依存してはならない。

B

命令フォーマットおよびエンコーディング

付録 B
命令フォーマットおよびエンコーディング

B

本付録では、IA-32 アーキテクチャ命令の機械語命令のフォーマットとエンコーディングを説明する。最初のセクションでは、IA-32 アーキテクチャの機械語命令のフォーマットを詳しく説明する。次のセクションでは、汎用、MMX、P6 ファミリ、SSE、SSE2、SSE3、x87 FPU の各命令のフォーマットとエンコーディングを説明する。

B.1. マシン命令フォーマット

すべてのインテル・アーキテクチャ命令は、図 B-1. に示す汎用のマシン命令フォーマットのサブセットを使用してコード化される。各命令は、オペコード、ModR/M バイトを構成するレジスタ指定子とアドレスモード指定子（必要な場合）、必要な場合には、スケール・インデックス・ベース（SIB）バイト、ディスプレースメント、即値データ・フィールドからなる。

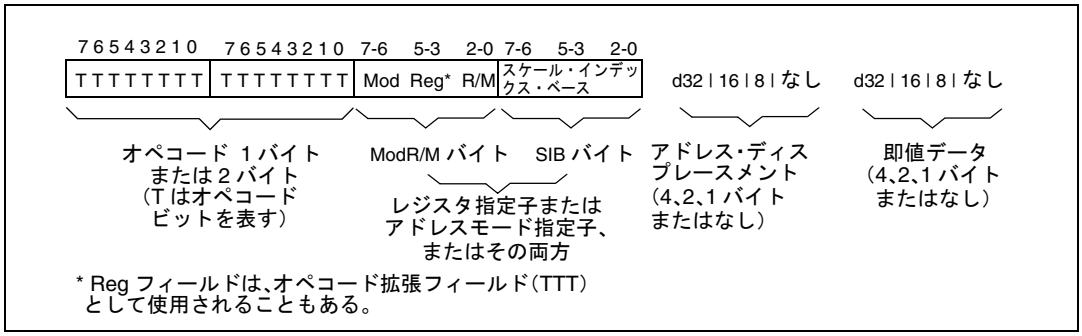


図 B-1. 汎用マシン命令フォーマット

命令のプライマリ・オペコードは、命令の 1 バイトまたは 2 バイトにコード化される。一部の命令では、ModR/M バイトのビット 5、4、3 にコード化されるオペコード拡張フィールドも使用する。プライマリ・オペコード内では、それより小さいエンコーディング・フィールドを定義することができる。これらのフィールドは、実行される操作のクラスによって変わる。フィールドでは、レジスタ・エンコーディング、実行する条件付きテスト、または即値バイトの符号拡張などの情報を定義する。

レジスタやメモリのオペランドを参照するほとんどすべての命令は、オペコードの後にレジスタバイトやアドレス・モード・バイトをもつ。この ModR/M バイトは、mod フィールド、reg フィールド、R/M フィールドからなる。ModR/M バイトの一定のエンコーディングは、第 2 アドレス・モード・バイトである SIB バイトを使用しなければならないことを指定する。

選択されているアドレス指定モードがディスプレースメントを指定している場合は、ディスプレースメント値が ModR/M バイトまたは SIB バイトの直後に置かれる。ディスプレースメントが存在している場合は、可能なサイズは、8、16、または 32 ビットである。

命令が即値オペランドを指定している場合は、即値がディスプレースメントの後に続く。即値オペランドは、指定する場合は、常に命令の最終フィールドである。

表 B-1. に、一定の命令に、オペコード・バイト自体の中にも現れることもあるいくつかのより小さいフィールドまたはビットの一覧を示す。以降の表では、これらのフィールドとビットについて説明し、許容可能な値の一覧を示す。（d ビットを除く）これらのフィールドのすべては、表 B-11. に示す汎用命令フォーマットで示している。

表 B-1. 命令エンコーディング内の特殊フィールド

フィールド名	説明	ビット数
reg	汎用レジスタ指定子（表 B-2. または B-3. を参照）。	3
w	データがバイトであるかまたはフルサイズであるかを指定する。ここで、フルサイズとは 16 ビットまたは 32 ビットである（表 B-4. を参照）。	1
s	即値データ・フィールドの符号拡張を指定する（表 B-5. を参照）。	1
sreg2	CS、SS、DS、ES のセグメント・レジスタ指定子（表 B-6. を参照）。	2
sreg3	CS、SS、DS、ES、FS、GS のセグメント・レジスタ指定子（表 B-6. を参照）。	3
eee	特殊目的（制御またはデバッグ）レジスタを指定する（表 B-7. を参照）。	3
tttn	条件付き命令の場合に、アサートされる条件またはネゲートされる条件を指定する（表 B-8. を参照）。	4
d	データ操作の方向を指定する（表 B-9. を参照）。	1

B.1.1. reg フィールド（reg）

ModR/M バイトの reg フィールドでは、汎用レジスタ・オペランドを指定する。指定されるレジスタのグループは、エンコーディングにおける w ビットの存在と状態によって決定される（表 B-4. を参照）。表 B-2. に、w ビットがエンコーディングで存在していないときの reg フィールドのエンコーディングを示し、表 B-3. に、w ビットが存在しているときの reg フィールドのエンコーディングを示す。

表 B-2. w フィールドが命令に存在していないときの reg フィールドのエンコーディング

reg フィールド	16 ビット・データ操作中に選択されるレジスタ	32 ビット・データ操作中に選択されるレジスタ
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

表 B-3. w フィールドが命令に存在しているときの reg フィールドのエンコーディング

16 ビット・データ操作中に reg フィールドによって指定されるレジスタ		
	w フィールドの機能	
reg	w = 0 のとき	w = 1 のとき
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

32 ビット・データ操作中に reg フィールドによって指定されるレジスタ		
	w フィールドの機能	
reg	w = 0 のとき	w = 1 のとき
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

B.1.2. オペランド・サイズ・ビット（w）のエンコーディング

現在のオペランド・サイズ属性によって、プロセッサが 16 ビットまたは 32 ビットどちらの操作を実行するかが決まる。現在のオペランド・サイズ属性の制約内で、オペランド・サイズ・ビット（w）を使用して、8 ビット・オペランドまたはオペランド・サイズ属性で指定されたオペランド・サイズ全体（16 ビットまたは 32 ビット）への操作を指定することができる。表 B-4. に、現在のオペランド・サイズ属性に依存した w ビットのエンコーディングを示す。

表 B-4. オペランド・サイズ（w）ビットのエンコーディング

w ビット	オペランド・サイズ属性が 16 ビットであるときのオペランド・サイズ	オペランド・サイズ属性が 32 ビットであるときのオペランド・サイズ
0	8 ビット	8 ビット
1	16 ビット	32 ビット

B.1.3. 符号拡張（s）ビット

符号拡張（s）ビットは、8 ビットから 16 ビットまたは 32 ビットに拡張される即値データ・フィールドをもつ命令に主に現れる。表 B-5. に、s ビットのエンコーディングを示す。

表 B-5. 符号拡張（s）ビットのエンコーディング

s	8 ビット即値データへの影響	16 ビットまたは 32 ビット即値データへの影響
0	なし	なし
1	16 ビットまたは 32 ビットのデスティネーションに合うように符号拡張	なし

B.1.4. セグメント・レジスタ・フィールド（sreg）

命令がセグメント・レジスタを操作するときは、ModR/M バイトの reg フィールドを sreg フィールドといい、セグメント・レジスタの指定に使用する。表 B-6. に、sreg フィールドのエンコーディングを示す。このフィールドは、2 ビット・フィールド（sreg2）のときと、3 ビット・フィールド（sreg3）のときがある。

表 B-6. セグメント・レジスタ（sreg）フィールドのエンコーディング

2 ビット sreg フィールド	選択されるセグメント・レジスタ
00	ES
01	CS
10	SS
11	DS

3 ビット sreg フィールド	選択されるセグメント・レジスタ
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	予約済み *
111	予約済み *

* 予約エンコーディングを使用してはならない。

B.1.5. 特殊目的レジスタ（eee）フィールド

制御レジスタまたはデバッグレジスタが命令で参照されているときは、それらのレジスタは、ModR/M バイトのビット 5、4、3 にある eee フィールド内にコード化される。表 B-7. に、eee フィールドのエンコーディングを示す。

表 B-7. 特殊目的レジスタ（eee）フィールドのエンコーディング

eee	制御レジスタ	デバッグレジスタ
000	CR0	DR0
001	予約済み *	DR1
010	CR2	DR2
011	CR3	DR3
100	CR4	予約済み *
101	予約済み *	予約済み *
110	予約済み *	DR6
111	予約済み *	DR7

* 予約エンコーディングを使用してはならない。

B.1.6. 条件テスト・フィールド（tttn）

（条件付きジャンプや条件での設定などの）条件付き命令では、条件テスト・フィールド（tttn）は、テストされる条件に対してコード化される。フィールドの ttt 部分では、テストする条件を指定し、n 部分では、条件（n=0）またはその否定（n=1）のどちらを使用するかを指定する。1 バイト・プライマリ・オペコードでは、tttn フィールドは、オペコード・バイトのビット 3、2、1、0 になる。2 バイト・プライマリ・オペコードでは、tttn フィールドは、第 2 オペコード・バイトのビット 3、2、1、0 になる。表 B-8. に、tttn フィールドのエンコーディングを示す。

表 B-8. 条件付きテスト（tttn）フィールドのエンコーディング

tttn	ニーモニック	条件
0000	O	オーバーフローがある
0001	NO	オーバーフローがない
0010	B, NAE	より下、より上でなく等しくない
0011	NB, AE	より下でない、より上か等しい
0100	E, Z	等しい、ゼロ
0101	NE, NZ	等しくない、ゼロでない
0110	BE, NA	より下か等しい、より上でない
0111	NBE, A	より下でなく等しくない、より上
1000	S	符号がある
1001	NS	符号がない
1010	P, PE	パリティがある、パリティ偶数
1011	NP, PO	パリティがない、パリティ奇数
1100	L, NGE	より小さい、より大きくなく等しくない
1101	NL, GE	より小さくない、より大きいか等しい
1110	LE, NG	より小さいか等しい、より大きくない
1111	NLE, G	より小さくなく等しくない、より大きい

B.1.7. 方向（d）ビット

多くの2オペランド命令では、方向ビット（d）は、どちらのオペランドをソースと考え、どちらのオペランドをデスティネーションと考えるかを指定する。表 B-9. に、dビットのエンコーディングを示す。整数命令に使用するときは、dビットは1バイト・プライマリ・オペコードのビット1になる。このビットは、表B-11. には記号"d"として表されていない。その代わりに、1 または 0 としてビットの実際のエンコーディングが示されている。（表 B-16. にある）浮動小数点命令に使用するときは、dビットはプライマリ・オペコードの第1バイトのビット2として示される。

表 B-9. 操作方向（d）ビットのエンコーディング

d	ソース	デスティネーション
0	reg フィールド	ModR/M バイトまたは SIB バイト
1	ModR/M バイトまたは SIB バイト	reg フィールド

B.1.8. その他の注意事項

表B-10に、特定のエンコーディングに関する注意事項を示す。以下の各項では、表中の注を上付き文字で示す。

表 B-10. 命令のエンコーディングに関する注意事項

記号	注意事項
A	ModR/M バイトのビット 7 とビット 6 の値 11B は予約されている。

B.2.　汎用命令のフォーマットおよびエンコーディング

表 B-11. に、汎用命令のマシン命令フォーマットおよびエンコーディングを示す。

表 B-11. 汎用命令のフォーマットおよびエンコーディング

命令およびフォーマット	エンコーディング
AAA – ASCII Adjust after Addition	0011 0111
AAD – ASCII Adjust AX before Division	1101 0101 : 0000 1010
AAM – ASCII Adjust AX after Multiply	1101 0100 : 0000 1010
AAS – ASCII Adjust AL after Subtraction	0011 1111
ADC – ADD with Carry	
register1 to register2	0001 000w : 11 reg1 reg2
register2 to register1	0001 001w : 11 reg1 reg2
memory to register	0001 001w : mod reg r/m
register to memory	0001 000w : mod reg r/m
immediate to register	1000 00sw : 11 010 reg : immediate data
immediate to AL, AX, or EAX	0001 010w : immediate data
immediate to memory	1000 00sw : mod 010 r/m : immediate data
ADD – Add	
register1 to register2	0000 000w : 11 reg1 reg2
register2 to register1	0000 001w : 11 reg1 reg2
memory to register	0000 001w : mod reg r/m
register to memory	0000 000w : mod reg r/m
immediate to register	1000 00sw : 11 000 reg : immediate data
immediate to AL, AX, or EAX	0000 010w : immediate data
immediate to memory	1000 00sw : mod 000 r/m : immediate data
AND – Logical AND	
register1 to register2	0010 000w : 11 reg1 reg2
register2 to register1	0010 001w : 11 reg1 reg2
memory to register	0010 001w : mod reg r/m
register to memory	0010 000w : mod reg r/m
immediate to register	1000 00sw: 11 100 reg: immediate data
immediate to AL, AX, or EAX	0010 010w : immediate data
immediate to memory	1000 00sw : mod 100 r/m : immediate data
ARPL – Adjust RPL Field of Selector	
from register	0110 0011 : 11 reg1 reg2
from memory	0110 0011 : mod reg r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
BOUND – Check Array Against Bounds	0110 0010 : mod[A] reg r/m
BSF – Bit Scan Forward	
register1, register2	0000 1111 : 1011 1100 : 11 reg1 reg2
memory, register	0000 1111 : 1011 1100 : mod reg r/m
BSR – Bit Scan Reverse	
register1, register2	0000 1111 : 1011 1101 : 11 reg1 reg2
memory, register	0000 1111 : 1011 1101 : mod reg r/m
BSWAP – Byte Swap	0000 1111 : 1100 1 reg
BT – Bit Test	
register, immediate	0000 1111 : 1011 1010 : 11 100 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 100 r/m : imm8 data
register1, register2	0000 1111 : 1010 0011 : 11 reg2 reg1
memory, reg	0000 1111 : 1010 0011 : mod reg r/m
BTC – Bit Test and Complement	
register, immediate	0000 1111 : 1011 1010 : 11 111 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 111 r/m : imm8 data
register1, register2	0000 1111 : 1011 1011 : 11 reg2 reg1
memory, reg	0000 1111 : 1011 1011 : mod reg r/m
BTR – Bit Test and Reset	
register, immediate	0000 1111 : 1011 1010 : 11 110 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 110 r/m : imm8 data
register1, register2	0000 1111 : 1011 0011 : 11 reg2 reg1
memory, reg	0000 1111 : 1011 0011 : mod reg r/m
BTS – Bit Test and Set	
register, immediate	0000 1111 : 1011 1010 : 11 101 reg: imm8 data
memory, immediate	0000 1111 : 1011 1010 : mod 101 r/m : imm8 data
register1, register2	0000 1111 : 1010 1011 : 11 reg2 reg1
memory, reg	0000 1111 : 1010 1011 : mod reg r/m
CALL – Call Procedure (in same segment)	
direct	1110 1000 : full displacement
register indirect	1111 1111 : 11 010 reg
memory indirect	1111 1111 : mod 010 r/m
CALL – Call Procedure (in other segment)	
direct	1001 1010 : unsigned full offset, selector
indirect	1111 1111 : mod 011 r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
CBW – Convert Byte to Word	1001 1000
CDQ – Convert Doubleword to Qword	1001 1001
CLC – Clear Carry Flag	1111 1000
CLD – Clear Direction Flag	1111 1100
CLI – Clear Interrupt Flag	1111 1010
CLTS – Clear Task-Switched Flag in CR0	0000 1111 : 0000 0110
CMC – Complement Carry Flag	1111 0101
CMP – Compare Two Operands	
register1 with register2	0011 100w : 11 reg1 reg2
register2 with register1	0011 101w : 11 reg1 reg2
memory with register	0011 100w : mod reg r/m
register with memory	0011 101w : mod reg r/m
immediate with register	1000 00sw : 11 111 reg : immediate data
immediate with AL, AX, or EAX	0011 110w : immediate data
immediate with memory	1000 00sw : mod 111 r/m : immediate data
CMPS/CMPSB/CMPSW/CMPSD – Compare String Operands	1010 011w
CMPXCHG – Compare and Exchange	
register1, register2	0000 1111 : 1011 000w : 11 reg2 reg1
memory, register	0000 1111 : 1011 000w : mod reg r/m
CPUID – CPU Identification	0000 1111 : 1010 0010
CWD – Convert Word to Doubleword	1001 1001
CWDE – Convert Word to Doubleword	1001 1000
DAA – Decimal Adjust AL after Addition	0010 0111
DAS – Decimal Adjust AL after Subtraction	0010 1111
DEC – Decrement by 1	
register	1111 111w : 11 001 reg
register (alternate encoding)	0100 1 reg
memory	1111 111w : mod 001 r/m
DIV – Unsigned Divide	
AL, AX, or EAX by register	1111 011w : 11 110 reg
AL, AX, or EAX by memory	1111 011w : mod 110 r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
ENTER – Make Stack Frame for High Level Procedure	1100 1000 : 16-bit displacement : 8-bit level (L)
HLT – Halt	1111 0100
IDIV – Signed Divide	
AL, AX, or EAX by register	1111 011w : 11 111 reg
AL, AX, or EAX by memory	1111 011w : mod 111 r/m
IMUL – Signed Multiply	
AL, AX, or EAX with register	1111 011w : 11 101 reg
AL, AX, or EAX with memory	1111 011w : mod 101 reg
register1 with register2	0000 1111 : 1010 1111 : 11 : reg1 reg2
register with memory	0000 1111 : 1010 1111 : mod reg r/m
register1 with immediate to register2	0110 10s1 : 11 reg1 reg2 : immediate data
memory with immediate to register	0110 10s1 : mod reg r/m : immediate data
IN – Input From Port	
fixed port	1110 010w : port number
variable port	1110 110w
INC – Increment by 1	
reg	1111 111w : 11 000 reg
reg (alternate encoding)	0100 0 reg
memory	1111 111w : mod 000 r/m
INS – Input from DX Port	0110 110w
INT n – Interrupt Type n	1100 1101 : type
INT – Single-Step Interrupt 3	1100 1100
INTO – Interrupt 4 on Overflow	1100 1110
INVD – Invalidate Cache	0000 1111 : 0000 1000
INVLPG – Invalidate TLB Entry	0000 1111 : 0000 0001 : mod 111 r/m
IRET/IRETD – Interrupt Return	1100 1111
***J*cc – Jump if Condition is Met**	
8-bit displacement	0111 tttn : 8-bit displacement
full displacement	0000 1111 : 1000 tttn : full displacement
JCXZ/JECXZ – Jump on CX/ECX Zero Address-size prefix differentiates JCXZ and JECXZ	1110 0011 : 8-bit displacement

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
JMP – Unconditional Jump (to same segment)	
short	1110 1011 : 8-bit displacement
direct	1110 1001 : full displacement
register indirect	1111 1111 : 11 100 reg
memory indirect	1111 1111 : mod 100 r/m
JMP – Unconditional Jump (to other segment)	
direct intersegment	1110 1010 : unsigned full offset, selector
indirect intersegment	1111 1111 : mod 101 r/m
LAHF – Load Flags into AHRegister	1001 1111
LAR – Load Access Rights Byte	
from register	0000 1111 : 0000 0010 : 11 reg1 reg2
from memory	0000 1111 : 0000 0010 : mod reg r/m
LDS – Load Pointer to DS	1100 0101 : mod[A] reg r/m
LEA – Load Effective Address	1000 1101 : mod[A] reg r/m
LEAVE – High Level Procedure Exit	1100 1001
LES – Load Pointer to ES	1100 0100 : mod[A] reg r/m
LFS – Load Pointer to FS	0000 1111 : 1011 0100 : mod[A] reg r/m
LGDT – Load Global Descriptor Table Register	0000 1111 : 0000 0001 : mod[A] 010 r/m
LGS – Load Pointer to GS	0000 1111 : 1011 0101 : mod[A] reg r/m
LIDT – Load Interrupt Descriptor Table Register	0000 1111 : 0000 0001 : mod[A] 011 r/m
LLDT – Load Local Descriptor Table Register	
LDTR from register	0000 1111 : 0000 0000 : 11 010 reg
LDTR from memory	0000 1111 : 0000 0000 : mod 010 r/m
LMSW – Load Machine Status Word	
from register	0000 1111 : 0000 0001 : 11 110 reg
from memory	0000 1111 : 0000 0001 : mod 110 r/m
LOCK – Assert LOCK# Signal Prefix	1111 0000
LODS/LODSB/LODSW/LODSD – Load String Operand	1010 110w
LOOP – Loop Count	1110 0010 : 8-bit displacement
LOOPZ/LOOPE – Loop Count while Zero/Equal	1110 0001 : 8-bit displacement
LOOPNZ/LOOPNE – Loop Count while not Zero/Equal	1110 0000 : 8-bit displacement
LSL – Load Segment Limit	
from register	0000 1111 : 0000 0011 : 11 reg1 reg2
from memory	0000 1111 : 0000 0011 : mod reg r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
LSS – Load Pointer to SS	0000 1111 : 1011 0010 : mod[A] reg r/m
LTR – Load Task Register	
from register	0000 1111 : 0000 0000 : 11 011 reg
from memory	0000 1111 : 0000 0000 : mod 011 r/m
MOV – Move Data	
register1 to register2	1000 100w : 11 reg1 reg2
register2 to register1	1000 101w : 11 reg1 reg2
memory to reg	1000 101w : mod reg r/m
reg to memory	1000 100w : mod reg r/m
immediate to register	1100 011w : 11 000 reg : immediate data
immediate to register (alternate encoding)	1011 w reg : immediate data
immediate to memory	1100 011w : mod 000 r/m : immediate data
memory to AL, AX, or EAX	1010 000w : full displacement
AL, AX, or EAX to memory	1010 001w : full displacement
MOV – Move to/from Control Registers	
CR0 from register	0000 1111 : 0010 0010 : 11 000 reg
CR2 from register	0000 1111 : 0010 0010 : 11 010reg
CR3 from register	0000 1111 : 0010 0010 : 11 011 reg
CR4 from register	0000 1111 : 0010 0010 : 11 100 reg
register from CR0-CR4	0000 1111 : 0010 0000 : 11 eee reg
MOV – Move to/from Debug Registers	
DR0-DR3 from register	0000 1111 : 0010 0011 : 11 eee reg
DR4-DR5 from register	0000 1111 : 0010 0011 : 11 eee reg
DR6-DR7 from register	0000 1111 : 0010 0011 : 11 eee reg
register from DR6-DR7	0000 1111 : 0010 0001 : 11 eee reg
register from DR4-DR5	0000 1111 : 0010 0001 : 11 eee reg
register from DR0-DR3	0000 1111 : 0010 0001 : 11 eee reg
MOV – Move to/from Segment Registers	
register to segment register	1000 1110 : 11 sreg3 reg
register to SS	1000 1110 : 11 sreg3 reg
memory to segment reg	1000 1110 : mod sreg3 r/m
memory to SS	1000 1110 : mod sreg3 r/m
segment register to register	1000 1100 : 11 sreg3 reg
segment register to memory	1000 1100 : mod sreg3 r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
MOVS/MOVSB/MOVSW/MOVSD – Move Data from String to String	1010 010w
MOVSX – Move with Sign-Extend	
register2 to register1	0000 1111 : 1011 111w : 11 reg1 reg2
memory to reg	0000 1111 : 1011 111w : mod reg r/m
MOVZX – Move with Zero-Extend	
register2 to register1	0000 1111 : 1011 011w : 11 reg1 reg2
memory to register	0000 1111 : 1011 011w : mod reg r/m
MUL – Unsigned Multiply	
AL, AX, or EAX with register	1111 011w : 11 100 reg
AL, AX, or EAX with memory	1111 011w : mod 100 reg
NEG – Two's Complement Negation	
register	1111 011w : 11 011 reg
memory	1111 011w : mod 011 r/m
NOP – No Operation	1001 0000
NOT – One's Complement Negation	
register	1111 011w : 11 010 reg
memory	1111 011w : mod 010 r/m
OR – Logical Inclusive OR	
register1 to register2	0000 100w : 11 reg1 reg2
register2 to register1	0000 101w : 11 reg1 reg2
memory to register	0000 101w : mod reg r/m
register to memory	0000 100w : mod reg r/m
immediate to register	1000 00sw : 11 001 reg : immediate data
immediate to AL, AX, or EAX	0000 110w : immediate data
immediate to memory	1000 00sw : mod 001 r/m : immediate data
OUT – Output to Port	
fixed port	1110 011w : port number
variable port	1110 111w
OUTS – Output to DX Port	0110 111w
POP – Pop a Word from the Stack	
register	1000 1111 : 11 000 reg
register (alternate encoding)	0101 1 reg
memory	1000 1111 : mod 000 r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
POP – Pop a Segment Register from the Stack	
（注 : CS はこの使用法では sreg2 になれない）	
segment register DS, ES	000 sreg2 111
segment register SS	000 sreg2 111
segment register FS, GS	0000 1111: 10 sreg3 001
POPA/POPAD – Pop All General Registers	0110 0001
POPF/POPFD – Pop Stack into FLAGS or EFLAGS Register	1001 1101
PUSH – Push Operand onto the Stack	
register	1111 1111 : 11 110 reg
register (alternate encoding)	0101 0 reg
memory	1111 1111 : mod 110 r/m
immediate	0110 10s0 : immediate data
PUSH – Push Segment Register onto the Stack	
segment register CS,DS,ES,SS	000 sreg2 110
segment register FS,GS	0000 1111: 10 sreg3 000
PUSHA/PUSHAD – Push All General Registers	0110 0000
PUSHF/PUSHFD – Push Flags Register onto the Stack	1001 1100
RCL – Rotate thru Carry Left	
register by 1	1101 000w : 11 010 reg
memory by 1	1101 000w : mod 010 r/m
register by CL	1101 001w : 11 010 reg
memory by CL	1101 001w : mod 010 r/m
register by immediate count	1100 000w : 11 010 reg : imm8 data
memory by immediate count	1100 000w : mod 010 r/m : imm8 data
RCR – Rotate thru Carry Right	
register by 1	1101 000w : 11 011 reg
memory by 1	1101 000w : mod 011 r/m
register by CL	1101 001w : 11 011 reg
memory by CL	1101 001w : mod 011 r/m
register by immediate count	1100 000w : 11 011 reg : imm8 data
memory by immediate count	1100 000w : mod 011 r/m : imm8 data

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
RDMSR – Read from Model-Specific Register	0000 1111 : 0011 0010
RDPMC – Read Performance Monitoring Counters	0000 1111 : 0011 0011
RDTSC – Read Time-Stamp Counter	0000 1111 : 0011 0001
REP INS – Input String	1111 0011 : 0110 110w
REP LODS – Load String	1111 0011 : 1010 110w
REP MOVS – Move String	1111 0011 : 1010 010w
REP OUTS – Output String	1111 0011 : 0110 111w
REP STOS – Store String	1111 0011 : 1010 101w
REPE CMPS – Compare String	1111 0011 : 1010 011w
REPE SCAS – Scan String	1111 0011 : 1010 111w
REPNE CMPS – Compare String	1111 0010 : 1010 011w
REPNE SCAS – Scan String	1111 0010 : 1010 111w
RET – Return from Procedure (to same segment)	
no argument	1100 0011
adding immediate to SP	1100 0010 : 16-bit displacement
RET – Return from Procedure (to other segment)	
intersegment	1100 1011
adding immediate to SP	1100 1010 : 16-bit displacement
ROL – Rotate Left	
register by 1	1101 000w : 11 000 reg
memory by 1	1101 000w : mod 000 r/m
register by CL	1101 001w : 11 000 reg
memory by CL	1101 001w : mod 000 r/m
register by immediate count	1100 000w : 11 000 reg : imm8 data
memory by immediate count	1100 000w : mod 000 r/m : imm8 data
ROR – Rotate Right	
register by 1	1101 000w : 11 001 reg
memory by 1	1101 000w : mod 001 r/m
register by CL	1101 001w : 11 001 reg
memory by CL	1101 001w : mod 001 r/m
register by immediate count	1100 000w : 11 001 reg : imm8 data
memory by immediate count	1100 000w : mod 001 r/m : imm8 data
RSM – Resume from System Management Mode	0000 1111 : 1010 1010
SAHF – Store AH into Flags	1001 1110
SAL – Shift Arithmetic Left	same instruction as SHL

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
SAR – Shift Arithmetic Right	
register by 1	1101 000w : 11 111 reg
memory by 1	1101 000w : mod 111 r/m
register by CL	1101 001w : 11 111 reg
memory by CL	1101 001w : mod 111 r/m
register by immediate count	1100 000w : 11 111 reg : imm8 data
memory by immediate count	1100 000w : mod 111 r/m : imm8 data
SBB – Integer Subtraction with Borrow	
register1 to register2	0001 100w : 11 reg1 reg2
register2 to register1	0001 101w : 11 reg1 reg2
memory to register	0001 101w : mod reg r/m
register to memory	0001 100w : mod reg r/m
immediate to register	1000 00sw : 11 011 reg : immediate data
immediate to AL, AX, or EAX	0001 110w : immediate data
immediate to memory	1000 00sw : mod 011 r/m : immediate data
SCAS/SCASB/SCASW/SCASD – Scan String	1010 111w
SETcc – Byte Set on Condition	
register	0000 1111 : 1001 tttn : 11 000 reg
memory	0000 1111 : 1001 tttn : mod 000 r/m
SGDT – Store Global Descriptor Table Register	0000 1111 : 0000 0001 : mod[A] 000 r/m
SHL – Shift Left	
register by 1	1101 000w : 11 100 reg
memory by 1	1101 000w : mod 100 r/m
register by CL	1101 001w : 11 100 reg
memory by CL	1101 001w : mod 100 r/m
register by immediate count	1100 000w : 11 100 reg : imm8 data
memory by immediate count	1100 000w : mod 100 r/m : imm8 data
SHLD – Double Precision Shift Left	
register by immediate count	0000 1111 : 1010 0100 : 11 reg2 reg1 : imm8
memory by immediate count	0000 1111 : 1010 0100 : mod reg r/m : imm8
register by CL	0000 1111 : 1010 0101 : 11 reg2 reg1
memory by CL	0000 1111 : 1010 0101 : mod reg r/m

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
SHR – Shift Right	
register by 1	1101 000w : 11 101 reg
memory by 1	1101 000w : mod 101 r/m
register by CL	1101 001w : 11 101 reg
memory by CL	1101 001w : mod 101 r/m
register by immediate count	1100 000w : 11 101 reg : imm8 data
memory by immediate count	1100 000w : mod 101 r/m : imm8 data
SHRD – Double Precision Shift Right	
register by immediate count	0000 1111 : 1010 1100 : 11 reg2 reg1 : imm8
memory by immediate count	0000 1111 : 1010 1100 : mod reg r/m : imm8
register by CL	0000 1111 : 1010 1101 : 11 reg2 reg1
memory by CL	0000 1111 : 1010 1101 : mod reg r/m
SIDT – Store Interrupt Descriptor Table Register	0000 1111 : 0000 0001 : mod[A] 001 r/m
SLDT – Store Local Descriptor Table Register	
to register	0000 1111 : 0000 0000 : 11 000 reg
to memory	0000 1111 : 0000 0000 : mod 000 r/m
SMSW – Store Machine Status Word	
to register	0000 1111 : 0000 0001 : 11 100 reg
to memory	0000 1111 : 0000 0001 : mod 100 r/m
STC – Set Carry Flag	1111 1001
STD – Set Direction Flag	1111 1101
STI – Set Interrupt Flag	1111 1011
STOS/STOSB/STOSW/STOSD – Store String Data	1010 101w
STR – Store Task Register	
to register	0000 1111 : 0000 0000 : 11 001 reg
to memory	0000 1111 : 0000 0000 : mod 001 r/m
SUB – Integer Subtraction	
register1 to register2	0010 100w : 11 reg1 reg2
register2 to register1	0010 101w : 11 reg1 reg2
memory to register	0010 101w : mod reg r/m
register to memory	0010 100w : mod reg r/m
immediate to register	1000 00sw : 11 101 reg : immediate data
immediate to AL, AX, or EAX	0010 110w : immediate data
immediate to memory	1000 00sw : mod 101 r/m : immediate data

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
TEST – Logical Compare	
register1 and register2	1000 010w : 11 reg1 reg2
memory and register	1000 010w : mod reg r/m
immediate and register	1111 011w : 11 000 reg : immediate data
immediate and AL, AX, or EAX	1010 100w : immediate data
immediate and memory	1111 011w : mod 000 r/m : immediate data
UD2 – Undefined instruction	0000 FFFF : 0000 1011
VERR – Verify a Segment for Reading	
register	0000 1111 : 0000 0000 : 11 100 reg
memory	0000 1111 : 0000 0000 : mod 100 r/m
VERW – Verify a Segment for Writing	
register	0000 1111 : 0000 0000 : 11 101 reg
memory	0000 1111 : 0000 0000 : mod 101 r/m
WAIT – Wait	1001 1011
WBINVD – Writeback and Invalidate Data Cache	0000 1111 : 0000 1001
WRMSR – Write to Model-Specific Register	0000 1111 : 0011 0000
XADD – Exchange and Add	
register1, register2	0000 1111 : 1100 000w : 11 reg2 reg1
memory, reg	0000 1111 : 1100 000w : mod reg r/m
XCHG – Exchange Register/Memory with Register	
register1 with register2	1000 011w : 11 reg1 reg2
AX or EAX with reg	1001 0 reg
memory with reg	1000 011w : mod reg r/m
XLAT/XLATB – Table Look-up Translation	1101 0111
XOR – Logical Exclusive OR	
register1 to register2	0011 000w : 11 reg1 reg2
register2 to register1	0011 001w : 11 reg1 reg2
memory to register	0011 001w : mod reg r/m
register to memory	0011 000w : mod reg r/m
immediate to register	1000 00sw : 11 110 reg : immediate data
immediate to AL, AX, or EAX	0011 010w : immediate data
immediate to memory	1000 00sw : mod 110 r/m : immediate data

表 B-11. 汎用命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
Prefix Bytes	
address size	0110 0111
LOCK	1111 0000
operand size	0110 0110
CS segment override	0010 1110
DS segment override	0011 1110
ES segment override	0010 0110
FS segment override	0110 0100
GS segment override	0110 0101
SS segment override	0011 0110

B.3. インテル® Pentium® プロセッサ・ファミリ命令のフォーマットとエンコーディング

以下の表に、インテル® Pentium® プロセッサ・ファミリで導入されたフォーマットとエンコーディングを示す。

表 B-12. インテル® Pentium® プロセッサ・ファミリ命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
CMPXCHG8B – Compare and Exchange 8 Bytes	
memory, register	0000 1111 : 1100 0111 : mod 001 r/m

B.4. MMX® 命令のフォーマットおよびエンコーディング

EMMS 命令を除くすべての MMX® 命令は、2 バイトのインテル・アーキテクチャ整数フォーマットと同様のフォーマットを使用する。これらのフォーマット内のサブフィールド・エンコーディングの詳細について以下に説明する。

B.4.1. グラニュラリティ・フィールド（gg）

グラニュラリティ・フィールド（gg）では、命令が操作するパックド・オペランドのサイズを指定する。このフィールドは、使用するときは、第 2 オペコード・バイトのビット 1 と 0 になる。表 B-13. に、この gg フィールドのエンコーディングを示す。

表 B-13. データ・フィールドのグラニュラリティ（gg）のエンコーディング

gg	データのグラニュラリティ
00	パックドバイト
01	パックドワード
10	パックド・ダブルワード
11	クワッドワード

B.4.2. MMX® テクノロジおよび汎用レジスタ・フィールド（mmxreg および reg）

MMX® テクノロジ・レジスタ（mmxreg）は、オペランドとして使用すると、ModR/M バイトの reg フィールド（ビット 5、4、3）または R/M フィールド（ビット 2、1、0）、またはその両方にコード化される。

MMX 命令が汎用レジスタ（reg）を操作する場合は、レジスタは ModR/M バイトの R/M フィールドにコード化される。

B.4.3. MMX® 命令のフォーマットおよびエンコーディングの表

表 B-14. に、整数命令のフォーマットとエンコーディングを示す。

表 B-14. MMX® 命令のフォーマットおよびエンコーディング

命令およびフォーマット	エンコーディング
EMMS – Empty MMX technology state	0000 1111:01110111
MOVD – Move doubleword	
reg to mmreg	0000 1111:01101110: 11 mmxreg reg
reg from mmxreg	0000 1111:01111110: 11 mmxreg reg
mem to mmxreg	0000 1111:01101110: mod mmxreg r/m
mem from mmxreg	0000 1111:01111110: mod mmxreg r/m
MOVQ – Move quadword	
mmxreg2 to mmxreg1	0000 1111:01101111: 11 mmxreg1 mmxreg2
mmxreg2 from mmxreg1	0000 1111:01111111: 11 mmxreg1 mmxreg2
mem to mmxreg	0000 1111:01101111: mod mmxreg r/m
mem from mmxreg	0000 1111:01111111: mod mmxreg r/m
PACKSSDW[1] – Pack dword to word data (signed with saturation)	
mmxreg2 to mmxreg1	0000 1111:01101011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:01101011: mod mmxreg r/m
PACKSSWB[1] – Pack word to byte data (signed with saturation)	
mmxreg2 to mmxreg1	0000 1111:01100011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:01100011: mod mmxreg r/m
PACKUSWB[1] – Pack word to byte data (unsigned with saturation)	
mmxreg2 to mmxreg1	0000 1111:01100111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:01100111: mod mmxreg r/m
PADD – Add with wrap-around	
mmxreg2 to mmxreg1	0000 1111: 111111gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 111111gg: mod mmxreg r/m
PADDS – Add signed with saturation	
mmxreg2 to mmxreg1	0000 1111: 111011gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 111011gg: mod mmxreg r/m
PADDUS – Add unsigned with saturation	
mmxreg2 to mmxreg1	0000 1111: 110111gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 110111gg: mod mmxreg r/m
PAND – Bitwise And	
mmxreg2 to mmxreg1	0000 1111:11011011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11011011: mod mmxreg r/m

表 B-14. MMX® 命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
PANDN – Bitwise AndNot	
mmxreg2 to mmxreg1	0000 1111:11011111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11011111: mod mmxreg r/m
PCMPEQ – Packed compare for equality	
mmxreg1 with mmxreg2	0000 1111:011101gg: 11 mmxreg1 mmxreg2
mmxreg with memory	0000 1111:011101gg: mod mmxreg r/m
PCMPGT – Packed compare greater (signed)	
mmxreg1 with mmxreg2	0000 1111:011001gg: 11 mmxreg1 mmxreg2
mmxreg with memory	0000 1111:011001gg: mod mmxreg r/m
PMADDWD – Packed multiply add	
mmxreg2 to mmxreg1	0000 1111:11110101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11110101: mod mmxreg r/m
PMUL \HUW – Packed multiplication, store high word (unsigned)	
mmxreg2 to mmxreg1	0000 1111: 1110 0100: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1110 0100: mod mmxreg r/m
PMULHW – Packed multiplication, store high word	
mmxreg2 to mmxreg1	0000 1111:11100101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11100101: mod mmxreg r/m
PMULLW – Packed multiplication, store high word	
mmxreg2 to mmxreg1	0000 1111:11010101: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11010101: mod mmxreg r/m
POR – Bitwise Or	
mmxreg2 to mmxreg1	0000 1111:11101011: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11101011: mod mmxreg r/m
PSLL[2] – Packed shift left logical	
mmxreg1 by mmxreg2	0000 1111:111100gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:111100gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:011100gg: 11 110 mmxreg: imm8 data
PSRA[2] – Packed shift right arithmetic	
mmxreg1 by mmxreg2	0000 1111:111000gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:111000gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:011100gg: 11 100 mmxreg: imm8 data

表 B-14. MMX® 命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
PSRL[2] – Packed shift right logical	
mmxreg1 by mmxreg2	0000 1111:110100gg: 11 mmxreg1 mmxreg2
mmxreg by memory	0000 1111:110100gg: mod mmxreg r/m
mmxreg by immediate	0000 1111:011100gg: 11 010 mmxreg: imm8 data
PSUB – Subtract with wrap-around	
mmxreg2 from mmxreg1	0000 1111:111110gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:111110gg: mod mmxreg r/m
PSUBS – Subtract signed with saturation	
mmxreg2 from mmxreg1	0000 1111:111010gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:111010gg: mod mmxreg r/m
PSUBUS – Subtract unsigned with saturation	
mmxreg2 from mmxreg1	0000 1111:110110gg: 11 mmxreg1 mmxreg2
memory from mmxreg	0000 1111:110110gg: mod mmxreg r/m
PUNPCKH – Unpack high data to next larger type	
mmxreg2 to mmxreg1	0000 1111:011010gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:011010gg: mod mmxreg r/m
PUNPCKL – Unpack low data to next larger type	
mmxreg2 to mmxreg1	0000 1111:011000gg: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:011000gg: mod mmxreg r/m
PXOR – Bitwise Xor	
mmxreg2 to mmxreg1	0000 1111:11101111: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111:11101111: mod mmxreg r/m

注 :

1. パック命令は、1 つの型の符号付きパックドデータを次に小さい型の符号付きまたは符号なしデータに飽和させる。
2. シフト命令のフォーマットには、即値シフトカウントによるシフトをサポートするためのもう 1 つのフォーマットがある。シフト操作は、すべてのデータ型に対して等しくサポートされているわけではない。

B.5. P6 ファミリ命令のフォーマットとエンコーディング

表 B-15. に、P6 ファミリ・プロセッサにおいて IA-32 アーキテクチャに導入された命令のいくつかについて、そのフォーマットとエンコーディングを示す。

表 B-15. P6 ファミリ命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
CMOVcc – Conditional Move	
register2 to register1	0000 1111: 0100 tttn : 11 reg1 reg2
memory to register	0000 1111 : 0100 tttn : mod reg r/m
FCMOVcc – Conditional Move on EFLAG Register Condition Codes	
move if below (B)	11011 010 : 11 000 ST(i)
move if equal (E)	11011 010 : 11 001 ST(i)
move if below or equal (BE)	11011 010 : 11 010 ST(i)
move if unordered (U)	11011 010 : 11 011 ST(i)
move if not below (NB)	11011 011 : 11 000 ST(i)
move if not equal (NE)	11011 011 : 11 001 ST(i)
move if not below or equal (NBE)	11011 011 : 11 010 ST(i)
move if not unordered (NU)	11011 011 : 11 011 ST(i)
FCOMI – Compare Real and Set EFLAGS	11011 011 : 11 110 ST(i)
FXRSTOR – Restore x87 FPU, MMX, SSE, and SSE2 State	00001111:10101110: mod[A] 001 r/m
FXSAVE – Save x87 FPU, MMX, SSE, and SSE2 State	00001111:10101110: mod[A] 000 r/m
SYSENTER – Fast System Call	00001111:00110100
SYSEXIT – Fast Return from Fast System Call	00001111:00110101

注 :

1. FXSAVE 命令および FXRSTOR 命令では、"mod = 11" は予約されている。

B.6. SSE 命令のフォーマットとエンコーディング

SSE 命令は ModR/M フォーマットを使用し、先頭に 0FH プリフィックス・バイトが付く。一般に、一度の処理で二方向の動作（すなわち、ロード処理とストア処理）を実行することができる。

以下の 3 つの表（表 B-16.、B-17.、B-18.）は、それぞれ、SSE SIMD 浮動小数点命令、SSE SIMD 整数命令、SSE キャッシュ可能 / メモリ順序付け命令のフォーマットとエンコーディングを示している。SSE の中には、2 バイト・オペコードの一部として必須のプリフィックスである 66H、F2H、F3H を必要とするものがある。これらの必須プリフィックスは表の中に含まれている。

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
ADDPS – Add Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011000:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011000: mod xmmreg r/m
ADDSS – Add Scalar Single-Precision Floating-Point Values	
xmmreg to xmmreg	11110011:00001111:01011000:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011000: mod xmmreg r/m
ANDNPS – Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010101:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010101: mod xmmreg r/m
ANDPS – Bitwise Logical AND of Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010100:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010100: mod xmmreg r/m
CMPPS – Compare Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	00001111:11000010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	00001111:11000010: mod xmmreg r/m: imm8
CMPSS – Compare Scalar Single-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	11110011:00001111:11000010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	11110011:00001111:11000010: mod xmmreg r/m: imm8

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
COMISS – Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg to xmmreg	00001111:00101111:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:00101111: mod xmmreg r/m
CVTPI2PS – Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values	
mmreg to xmmreg	00001111:00101010:11 xmmreg1 mmreg1
mem to xmmreg	00001111:00101010: mod xmmreg r/m
CVTPS2PI – Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	00001111:00101101:11 mmreg1 xmmreg1
mem to mmreg	00001111:00101101: mod mmreg r/m
CVTSI2SS – Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value	
r32 to xmmreg1	11110011:00001111:00101010:11 xmmreg r32
mem to xmmreg	11110011:00001111:00101010: mod xmmreg r/m
CVTSS2SI – Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	11110011:00001111:00101101:11 r32 xmmreg
mem to r32	11110011:00001111:00101101: mod r32 r/m
CVTTPS2PI – Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	00001111:00101100:11 mmreg1 xmmreg1
mem to mmreg	00001111:00101100: mod mmreg r/m
CVTTSS2SI – Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	11110011:00001111:00101100:11 r32 xmmreg1
mem to r32	11110011:00001111:00101100: mod r32 r/m
DIVPS – Divide Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011110:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011110: mod xmmreg r/m
DIVSS – Divide Scalar Single-Precision Floating-Point Values	
xmmreg to xmmreg	11110011:00001111:01011110:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011110: mod xmmreg r/m

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
LDMXCSR – Load MXCSR Register State	
m32 to MXCSR	00001111:10101110:mod[A] 010 mem
MAXPS – Return Maximum Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011111:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011111: mod xmmreg r/m
MAXSS – Return Maximum Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110011:00001111:01011111:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011111: mod xmmreg r/m
MINPS – Return Minimum Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011101:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011101: mod xmmreg r/m
MINSS – Return Minimum Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110011:00001111:01011101:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011101: mod xmmreg r/m
MOVAPS – Move Aligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	00001111:00101000:11 xmmreg2 xmmreg1
mem to xmmreg1	00001111:00101000: mod xmmreg r/m
xmmreg1 to xmmreg2	00001111:00101001:11 xmmreg1 xmmreg2
xmmreg1 to mem	00001111:00101001: mod xmmreg r/m
MOVHLPS – Move Packed Single-Precision Floating-Point Values High to Low	
xmmreg to xmmreg	00001111:00010010:11 xmmreg1 xmmreg2
MOVHPS – Move High Packed Single-Precision Floating-Point Values	
mem to xmmreg	00001111:00010110: mod xmmreg r/m
xmmreg to mem	00001111:00010111: mod xmmreg r/m
MOVLHPS – Move Packed Single-Precision Floating-Point Values Low to High	
xmmreg to xmmreg	00001111:00010110:11 xmmreg1 xmmreg2
MOVLPS – Move Low Packed Single-Precision Floating-Point Values	
mem to xmmreg	00001111:00010010: mod xmmreg r/m
xmmreg to mem	00001111:00010011: mod xmmreg r/m

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
MOVMSKPS – Extract Packed Single-Precision Floating-Point Sign Mask	
xmmreg to r32	00001111:01010000:11 r32 xmmreg
MOVSS – Move Scalar Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	11110011:00001111:00010000:11 xmmreg2 xmmreg1
mem to xmmreg1	11110011:00001111:00010000: mod xmmreg r/m
xmmreg1 to xmmreg2	11110011:00001111:00010001:11 xmmreg1 xmmreg2
xmmreg1 to mem	11110011:00001111:00010001: mod xmmreg r/m
MOVUPS – Move Unaligned Packed Single-Precision Floating-Point Values	
xmmreg2 to xmmreg1	00001111:00010000:11 xmmreg2 xmmreg1
mem to xmmreg1	00001111:00010000: mod xmmreg r/m
xmmreg1 to xmmreg2	00001111:00010001:11 xmmreg1 xmmreg2
xmmreg1 to mem	00001111:00010001: mod xmmreg r/m
MULPS – Multiply Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011001:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011001: mod xmmreg rm
MULSS – Multiply Scalar Single-Precision Floating-Point Values	
xmmreg to xmmreg	11110011:00001111:01011001:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011001: mod xmmreg r/m
ORPS – Bitwise Logical OR of Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010110:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010110 mod xmmreg r/m
RCPPS – Compute Reciprocals of Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010011:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010011: mod xmmreg r/m
RCPSS – Compute Reciprocals of Scalar Single-Precision Floating-Point Value	
xmmreg to xmmreg	11110011:00001111:01010011:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01010011: mod xmmreg r/m

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
RSQRTPS – Compute Reciprocals of Square Roots of Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010010:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010010 mode xmmreg r/m
RSQRTSS – Compute Reciprocals of Square Roots of Scalar Single-Precision Floating-Point Value	
xmmreg to xmmreg	11110011:00001111:01010010:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01010010 mod xmmreg r/m
SHUFPS – Shuffle Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	00001111:11000110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	00001111:11000110: mod xmmreg r/m: imm8
SQRTPS – Compute Square Roots of Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010001:11 xmmreg1 xmmreg 2
mem to xmmreg	00001111:01010001 mod xmmreg r/m
SQRTSS – Compute Square Root of Scalar Single-Precision Floating-Point Value	
xmmreg to xmmreg	01010011:00001111:01010001:11 xmmreg1 xmmreg 2
mem to xmmreg	01010011:00001111:01010001 mod xmmreg r/m
STMXCSR – Store MXCSR Register State	
MXCSR to mem	00001111:10101110:mod[A] 011 mem
SUBPS – Subtract Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011100:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011100 mod xmmreg r/m
SUBSS – Subtract Scalar Single-Precision Floating-Point Values	
xmmreg to xmmreg	11110011:00001111:01011100:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011100 mod xmmreg r/m
UCOMISS – Unordered Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
xmmreg to xmmreg	00001111:00101110:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:00101110 mod xmmreg r/m
UNPCKHPS – Unpack and Interleave High Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:00010101:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:00010101 mod xmmreg r/m

表 B-16. SSE 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
UNPCKLPS – Unpack and Interleave Low Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:00010100:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:00010100 mod xmmreg r/m
XORPS – Bitwise Logical XOR of Single-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01010111:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01010111 mod xmmreg r/m

表 B-17. SSE SIMD 整数命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
PAVGB/PAVGW – Average Packed Integers	
mmreg to mmreg	00001111:11100000:11 mmreg1 mmreg2
	00001111:11100011:11 mmreg1 mmreg2
mem to mmreg	00001111:11100000 mod mmreg r/m
	00001111:11100011 mod mmreg r/m
PEXTRW – Extract Word	
mmreg to reg32, imm8	00001111:11000101:11 r32 mmreg: imm8
PINSRW – Insert Word	
reg32 to mmreg, imm8	00001111:11000100:11 mmreg r32: imm8
m16 to mmreg, imm8	00001111:11000100 mod mmreg r/m: imm8
PMAXSW – Maximum of Packed Signed Word Integers	
mmreg to mmreg	00001111:11101110:11 mmreg1 mmreg2
mem to mmreg	00001111:11101110 mod mmreg r/m
PMAXUB – Maximum of Packed Unsigned Byte Integers	
mmreg to mmreg	00001111:11011110:11 mmreg1 mmreg2
mem to mmreg	00001111:11011110 mod mmreg r/m
PMINSW – Minimum of Packed Signed Word Integers	
mmreg to mmreg	00001111:11101010:11 mmreg1 mmreg2
mem to mmreg	00001111:11101010 mod mmreg r/m
PMINUB – Minimum of Packed Unsigned Byte Integers	
mmreg to mmreg	00001111:11011010:11 mmreg1 mmreg2
mem to mmreg	00001111:11011010 mod mmreg r/m

表 B-17. SSE SIMD 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
PMOVMSKB – Move Byte Mask To Integer	
mmreg to reg32	00001111:11010111:11 r32 mmreg
PMULHUW – Multiply Packed Unsigned Integers and Store High Result	
mmreg to mmreg	00001111:11100100:11 mmreg1 mmreg2
mem to mmreg	00001111:11100100 mod mmreg r/m
PSADBW – Compute Sum of Absolute Differences	
mmreg to mmreg	00001111:11110110:11 mmreg1 mmreg2
mem to mmreg	00001111:11110110 mod mmreg r/m
PSHUFW – Shuffle Packed Words	
mmreg to mmreg, imm8	00001111:01110000:11 mmreg1 mmreg2: imm8
mem to mmreg, imm8	00001111:01110000:11 mod mmreg r/m: imm8

表 B-18. SSE キャッシュ可能 / メモリ順序付け命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
MASKMOVQ – Store Selected Bytes of Quadword	
mmreg to mmreg	00001111:11110111:11 mmreg1 mmreg2
MOVNTPS – Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg to mem	00001111:00101011 mod xmmreg r/m
MOVNTQ – Store Quadword Using Non-Temporal Hint	
mmreg to mem	00001111:11100111: mod mmreg r/m
PREFETCHT0 – Prefetch Temporal to All Cache Levels	00001111:00011000:mod[A] 001 mem
PREFETCHT1 – Prefetch Temporal to First Level Cache	00001111:00011000:mod[A] 010 mem
PREFETCHT2 – Prefetch Temporal to Second Level Cache	00001111:00011000:mod[A] 011 mem
PREFETCHNTA – Prefetch Non-Temporal to All Cache Levels	00001111:00011000:mod[A] 000 mem
SFENCE – Store Fence	00001111:10101110:11 111 000

B.7. SSE2 命令のフォーマットとエンコーディング

SSE2 命令は ModR/M フォーマットを使用し、先頭に 0FH プリフィックス・バイトが付く。一般に、一度の処理で二方向の動作（すなわち、ロード処理とストア処理）を実行することができる。

以下の 3 つの表は、それぞれ、SSE2 SIMD 浮動小数点命令、SSE2 SIMD 整数命令、SSE2 キャッシュ可能命令のフォーマットとエンコーディングを示している。SSE2 の中には、2 バイト・オペコードの一部として必須のプリフィックスである 66H、F2H、F3H を必要とするものがある。これらの必須プリフィックスは表の中に含まれている。

B.7.1. グラニュラリティ・フィールド（gg）

グラニュラリティ・フィールド（gg）では、命令の操作対象となるパックド・オペランドのサイズを指定する。このフィールドは、第 2 オペコード・バイトのビット 1 と 0 である。表 B-21. に、この gg フィールドのエンコーディングを示す。

表 B-19. データ・フィールドのグラニュラリティ（gg）のエンコーディング

gg	データのグラニュラリティ
00	パックドバイト
01	パックドワード
10	パックド・ダブルワード
11	クワッドワード

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
ADDPD – Add Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011000:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011000: mod xmmreg r/m
ADDSD – Add Scalar Double-Precision Floating-Point Values	
xmmreg to xmmreg	11110010:00001111:01011000:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011000: mod xmmreg r/m
ANDNPD – Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01010101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01010101: mod xmmreg r/m

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
ANDPD – Bitwise Logical AND of Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01010100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01010100: mod xmmreg r/m
CMPPD – Compare Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	01100110:00001111:11000010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	01100110:00001111:11000010: mod xmmreg r/m: imm8
CMPSD – Compare Scalar Double-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	11110010:00001111:11000010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	11110010:00001111:11000010: mod xmmreg r/m: imm8
COMISD – Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg to xmmreg	01100110:00001111:00101111:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:00101111: mod xmmreg r/m
CVTPI2PD – Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values	
mmreg to xmmreg	01100110:00001111:00101010:11 xmmreg1 mmreg1
mem to xmmreg	01100110:00001111:00101010: mod xmmreg r/m
CVTPD2PI – Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	01100110:00001111:00101101:11 mmreg1 xmmreg1
mem to mmreg	01100110:00001111:00101101: mod mmreg r/m
CVTSI2SD – Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value	
r32 to xmmreg1	11110010:00001111:00101010:11 xmmreg r32
mem to xmmreg	11110010:00001111:00101010: mod xmmreg r/m
CVTSD2SI – Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	11110010:00001111:00101101:11 r32 xmmreg
mem to r32	11110010:00001111:00101101: mod r32 r/m

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
CVTTPD2PI – Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to mmreg	01100110:00001111:00101100:11 mmreg xmmreg
mem to mmreg	01100110:00001111:00101100: mod mmreg r/m
CVTTSD2SI – Convert with Truncation Scalar Double-Precision Floating-Point Value to Doubleword Integer	
xmmreg to r32	11110010:00001111:00101100:11 r32 xmmreg
mem to r32	11110010:00001111:00101100: mod r32 r/m
CVTPD2PS – Covert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011010:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011010: mod xmmreg r/m
CVTPS2PD – Covert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011010:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011010: mod xmmreg r/m
CVTSD2SS – Covert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value	
xmmreg to xmmreg	11110010:00001111:01011010:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011010: mod xmmreg r/m
CVTSS2SD – Covert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110011:00001111:01011010:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011010: mod xmmreg r/m
CVTPD2DQ – Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to xmmreg	11110010:00001111:11100110:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:11100110: mod xmmreg r/m
CVTTPD2DQ – Convert With Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to xmmreg	01100110:00001111:11100110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11100110: mod xmmreg r/m

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
CVTDQ2PD – Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values	
xmmreg to xmmreg	11110011:00001111:11100110:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:11100110: mod xmmreg r/m
CVTPS2DQ – Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to xmmreg	01100110:00001111:01011011:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011011: mod xmmreg r/m
CVTTPS2DQ – Convert With Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers	
xmmreg to xmmreg	11110011:00001111:01011011:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01011011: mod xmmreg r/m
CVTDQ2PS – Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	00001111:01011011:11 xmmreg1 xmmreg2
mem to xmmreg	00001111:01011011: mod xmmreg r/m
DIVPD – Divide Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011110: mod xmmreg r/m
DIVSD – Divide Scalar Double-Precision Floating-Point Values	
xmmreg to xmmreg	11110010:00001111:01011110:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011110: mod xmmreg r/m
MAXPD – Return Maximum Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011111:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011111: mod xmmreg r/m
MAXSD – Return Maximum Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110010:00001111:01011111:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011111: mod xmmreg r/m
MINPD – Return Minimum Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011101: mod xmmreg r/m

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
MINSD – Return Minimum Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110010:00001111:01011101:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011101: mod xmmreg r/m
MOVAPD – Move Aligned Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	01100110:00001111:00101001:11 xmmreg2 xmmreg1
mem to xmmreg1	01100110:00001111:00101001: mod xmmreg r/m
xmmreg1 to xmmreg2	01100110:00001111:00101000:11 xmmreg1 xmmreg2
xmmreg1 to mem	01100110:00001111:00101000: mod xmmreg r/m
MOVHPD – Move High Packed Double-Precision Floating-Point Values	
mem to xmmreg	01100110:00001111:00010111: mod xmmreg r/m
xmmreg to mem	01100110:00001111:00010110: mod xmmreg r/m
MOVLPD – Move Low Packed Double-Precision Floating-Point Values	
mem to xmmreg	01100110:00001111:00010011: mod xmmreg r/m
xmmreg to mem	01100110:00001111:00010010: mod xmmreg r/m
MOVMSKPD – Extract Packed Double-Precision Floating-Point Sign Mask	
xmmreg to r32	01100110:00001111:01010000:11 r32 xmmreg
MOVSD – Move Scalar Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	11110010:00001111:00010001:11 xmmreg2 xmmreg1
mem to xmmreg1	11110010:00001111:00010001: mod xmmreg r/m
xmmreg1 to xmmreg2	11110010:00001111:00010000:11 xmmreg1 xmmreg2
xmmreg1 to mem	11110010:00001111:00010000: mod xmmreg r/m
MOVUPD – Move Unaligned Packed Double-Precision Floating-Point Values	
xmmreg2 to xmmreg1	01100110:00001111:00010001:11 xmmreg2 xmmreg1
mem to xmmreg1	01100110:00001111:00010001: mod xmmreg r/m
xmmreg1 to xmmreg2	01100110:00001111:00010000:11 xmmreg1 xmmreg2
xmmreg1 to mem	01100110:00001111:00010000: mod xmmreg r/m
MULPD – Multiply Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011001:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011001: mod xmmreg rm

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
MULSD – Multiply Scalar Double-Precision Floating-Point Values	
xmmreg to xmmreg	11110010:00001111:01011001:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011001: mod xmmreg r/m
ORPD – Bitwise Logical OR of Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01010110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01010110: mod xmmreg r/m
SHUFPD – Shuffle Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg, imm8	01100110:00001111:11000110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	01100110:00001111:11000110: mod xmmreg r/m: imm8
SQRTPD – Compute Square Roots of Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01010001:11 xmmreg1 xmmreg 2
mem to xmmreg	01100110:00001111:01010001: mod xmmreg r/m
SQRTSD – Compute Square Root of Scalar Double-Precision Floating-Point Value	
xmmreg to xmmreg	11110010:00001111:01010001:11 xmmreg1 xmmreg 2
mem to xmmreg	11110010:00001111:01010001: mod xmmreg r/m
SUBPD – Subtract Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01011100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01011100: mod xmmreg r/m
SUBSD – Subtract Scalar Double-Precision Floating-Point Values	
xmmreg to xmmreg	11110010:00001111:01011100:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01011100: mod xmmreg r/m
UCOMISD – Unordered Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	
xmmreg to xmmreg	01100110:00001111:00101110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:00101110: mod xmmreg r/m
UNPCKHPD – Unpack and Interleave High Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:00010101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:00010101: mod xmmreg r/m

表 B-20. SSE2 浮動小数点命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
UNPCKLPD – Unpack and Interleave Low Packed Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:00010100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:00010100: mod xmmreg r/m
XORPD – Bitwise Logical OR of Double-Precision Floating-Point Values	
xmmreg to xmmreg	01100110:00001111:01010111:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01010111: mod xmmreg r/m

表 B-21. SSE2 整数命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
MOVD – Move Doubleword	
reg to xmmeg	01100110:0000 1111:01101110: 11 xmmreg reg
reg from xmmreg	01100110:0000 1111:01111110: 11 xmmreg reg
mem to xmmreg	01100110:0000 1111:01101110: mod xmmreg r/m
mem from xmmreg	01100110:0000 1111:01111110: mod xmmreg r/m
MOVDQA – Move Aligned Double Quadword	
xmmreg to xmmreg	01100110:00001111:01101111:11 xmmreg1 xmmreg2
	01100110:00001111:01111111:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01101111: mod xmmreg r/m
mem from xmmreg	01100110:00001111:01111111: mod xmmreg r/m
MOVDQU – Move Unaligned Double Quadword	
xmmreg to xmmreg	11110011:00001111:01101111:11 xmmreg1 xmmreg2
	11110011:00001111:01111111:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01101111: mod xmmreg r/m
mem from xmmreg	11110011:00001111:01111111: mod xmmreg r/m
MOVQ2DQ – Move Quadword from MMX to XMM Register	
mmreg to xmmreg	11110011:00001111:11010110:11 mmreg1 mmreg2
MOVDQ2Q – Move Quadword from XMM to MMX Register	
xmmreg to mmreg	11110010:00001111:11010110:11 mmreg1 mmreg2

表 B-21. SSE2 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
MOVQ – Move Quadword	
mmxreg2 to mmxreg1	11110011:00001111:01111110: 11 xmmreg1 xmmreg2
mmxreg2 from mmxreg1	01100110:00001111:11010110: 11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:01111110: mod xmmreg r/m
mem from xmmreg	01100110:00001111:11010110: mod xmmreg r/m
PACKSSDW[1] – Pack Dword To Word Data (signed with saturation)	
xmmreg2 to xmmreg1	01100110:0000 1111:01101011: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:01101011: mod xmmreg r/m
PACKSSWB – Pack Word To Byte Data (signed with saturation)	
xmmreg2 to xmmreg1	01100110:0000 1111:01100011: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:01100011: mod xmmreg r/m
PACKUSWB – Pack Word To Byte Data (unsigned with saturation)	
xmmreg2 to xmmreg1	01100110:0000 1111:01100111: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:01100111: mod xmmreg r/m
PADDQ – Add Packed Quadword Integers	
mmreg to mmreg	00001111:11010100:11 mmreg1 mmreg2
mem to mmreg	00001111:11010100: mod mmreg r/m
xmmreg to xmmreg	01100110:00001111:11010100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11010100: mod xmmreg r/m
PADD – Add With Wrap-around	
xmmreg2 to xmmreg1	01100110:0000 1111: 111111gg: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111: 111111gg: mod xmmreg r/m
PADDS – Add Signed With Saturation	
xmmreg2 to xmmreg1	01100110:0000 1111: 111011gg: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111: 111011gg: mod xmmreg r/m
PADDUS – Add Unsigned With Saturation	
xmmreg2 to xmmreg1	01100110:0000 1111: 110111gg: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111: 110111gg: mod xmmreg r/m
PAND – Bitwise And	
xmmreg2 to xmmreg1	01100110:0000 1111:11011011: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11011011: mod xmmreg r/m

表 B-21. SSE2 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
PANDN – Bitwise AndNot	
xmmreg2 to xmmreg1	01100110:0000 1111:11011111: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11011111: mod xmmreg r/m
PAVGB – Average Packed Integers	
xmmreg to xmmreg	01100110:00001111:11100000:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11100000 mod xmmreg r/m
PAVGW – Average Packed Integers	
xmmreg to xmmreg	01100110:00001111:11100011:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11100011 mod xmmreg r/m
PCMPEQ – Packed Compare For Equality	
xmmreg1 with xmmreg2	01100110:0000 1111:011101gg: 11 xmmreg1 xmmreg2
xmmreg with memory	01100110:0000 1111:011101gg: mod xmmreg r/m
PCMPGT – Packed Compare Greater (signed)	
xmmreg1 with xmmreg2	01100110:0000 1111:011001gg: 11 xmmreg1 xmmreg2
xmmreg with memory	01100110:0000 1111:011001gg: mod xmmreg r/m
PEXTRW – Extract Word	
xmmreg to reg32, imm8	01100110:00001111:11000101:11 r32 xmmreg: imm8
PINSRW – Insert Word	
reg32 to xmmreg, imm8	01100110:00001111:11000100:11 xmmreg r32: imm8
m16 to xmmreg, imm8	01100110:00001111:11000100 mod xmmreg r/m: imm8
PMADD – Packed Multiply Add	
xmmreg2 to xmmreg1	01100110:0000 1111:11110101: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11110101: mod xmmreg r/m
PMAXSW – Maximum of Packed Signed Word Integers	
xmmreg to xmmreg	01100110:00001111:11101110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11101110 mod xmmreg r/m
PMAXUB – Maximum of Packed Unsigned Byte Integers	
xmmreg to xmmreg	01100110:00001111:11011110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11011110 mod xmmreg r/m
PMINSW – Minimum of Packed Signed Word Integers	
xmmreg to xmmreg	01100110:00001111:11101010:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11101010 mod xmmreg r/m

表 B-21. SSE2 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
PMINUB – Minimum of Packed Unsigned Byte Integers	
xmmreg to xmmreg	01100110:00001111:11011010:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11011010 mod xmmreg r/m
PMOVMSKB – Move Byte Mask To Integer	
xmmreg to reg32	01100110:00001111:11010111:11 r32 xmmreg
PMULHUW – Packed multiplication, store high word (unsigned)	
xmmreg2 to xmmreg1	0110 0110:0000 1111:1110 0100: 11 xmmreg1 xmmreg2
memory to xmmreg	0110 0110:0000 1111:1110 0100: mod xmmreg r/m
PMULHW – Packed Multiplication, store high word	
xmmreg2 to xmmreg1	01100110:0000 1111:11100101: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11100101: mod xmmreg r/m
PMULLW – Packed Multiplication, store high word	
xmmreg2 to xmmreg1	01100110:0000 1111:11010101: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11010101: mod xmmreg r/m
PMULUDQ – Multiply Packed Unsigned Doubleword Integers	
mmreg to mmreg	00001111:11110100:11 mmreg1 mmreg2
mem to mmreg	00001111:11110100: mod mmreg r/m
xmmreg to xmmreg	01100110:00001111:11110100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11110100: mod xmmreg r/m
POR – Bitwise Or	
xmmreg2 to xmmreg1	01100110:0000 1111:11101011: 11 xmmreg1 xmmreg2
xmemory to xmmreg	01100110:0000 1111:11101011: mod xmmreg r/m
PSADBW – Compute Sum of Absolute Differences	
xmmreg to xmmreg	01100110:00001111:11110110:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11110110: mod xmmreg r/m
PSHUFLW – Shuffle Packed Low Words	
xmmreg to xmmreg, imm8	11110010:00001111:01110000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	11110010:00001111:01110000:11 mod xmmreg r/m: imm8
PSHUFHW – Shuffle Packed High Words	
xmmreg to xmmreg, imm8	11110011:00001111:01110000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	11110011:00001111:01110000:11 mod xmmreg r/m: imm8

表 B-21. SSE2 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
PSHUFD – Shuffle Packed Doublewords	
xmmreg to xmmreg, imm8	01100110:00001111:01110000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	01100110:00001111:01110000:11 mod xmmreg r/m: imm8
PSLLDQ – Shift Double Quadword Left Logical	
xmmreg, imm8	01100110:00001111:01110011:11 111 xmmreg: imm8
PSLL – Packed Shift Left Logical	
xmmreg1 by xmmreg2	01100110:0000 1111:111100gg: 11 xmmreg1 xmmreg2
xmmreg by memory	01100110:0000 1111:111100gg: mod xmmreg r/m
xmmreg by immediate	01100110:0000 1111:011100gg: 11 110 xmmreg: imm8 data
PSRA – Packed Shift Right Arithmetic	
xmmreg1 by xmmreg2	01100110:0000 1111:111000gg: 11 xmmreg1 xmmreg2
xmmreg by memory	01100110:0000 1111:111000gg: mod xmmreg r/m
xmmreg by immediate	01100110:0000 1111:011100gg: 11 100 xmmreg: imm8 data
PSRLDQ – Shift Double Quadword Right Logical	
xmmreg, imm8	01100110:00001111:01110011:11 011 xmmreg: imm8
PSRL – Packed Shift Right Logical	
xmmxreg1 by xmmxreg2	01100110:0000 1111:110100gg: 11 xmmreg1 xmmreg2
xmmxreg by memory	01100110:0000 1111:110100gg: mod xmmreg r/m
xmmxreg by immediate	01100110:0000 1111:011100gg: 11 010 xmmreg: imm8 data
PSUBQ – Subtract Packed Quadword Integers	
mmreg to mmreg	00001111:11111011:11 mmreg1 mmreg2
mem to mmreg	00001111:11111011: mod mmreg r/m
xmmreg to xmmreg	01100110:00001111:11111011:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11111011: mod xmmreg r/m
PSUB – Subtract With Wrap-around	
xmmreg2 from xmmreg1	01100110:0000 1111:111110gg: 11 xmmreg1 xmmreg2
memory from xmmreg	01100110:0000 1111:111110gg: mod xmmreg r/m
PSUBS – Subtract Signed With Saturation	
xmmreg2 from xmmreg1	01100110:0000 1111:111010gg: 11 xmmreg1 xmmreg2
memory from xmmreg	01100110:0000 1111:111010gg: mod xmmreg r/m
PSUBUS – Subtract Unsigned With Saturation	
xmmreg2 from xmmreg1	0000 1111:110110gg: 11 xmmreg1 xmmreg2
memory from xmmreg	0000 1111:110110gg: mod xmmreg r/m

表 B-21. SSE2 整数命令のフォーマットとエンコーディング（続き）

命令およびフォーマット	エンコーディング
PUNPCKH – Unpack High Data To Next Larger Type	
xmmreg to xmmreg	01100110:00001111:011010gg:11 xmmreg1 Xmmreg2
mem to xmmreg	01100110:00001111:011010gg: mod xmmreg r/m
PUNPCKHQDQ – Unpack High Data	
xmmreg to xmmreg	01100110:00001111:01101101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01101101: mod xmmreg r/m
PUNPCKL – Unpack Low Data To Next Larger Type	
xmmreg to xmmreg	01100110:00001111:011000gg:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:011000gg: mod xmmreg r/m
PUNPCKLQDQ – Unpack Low Data	
xmmreg to xmmreg	01100110:00001111:01101100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01101100: mod xmmreg r/m
PXOR – Bitwise Xor	
xmmreg2 to xmmreg1	01100110:0000 1111:11101111: 11 xmmreg1 xmmreg2
memory to xmmreg	01100110:0000 1111:11101111: mod xmmreg r/m

表 B-22. SSE2 キャッシュ可能命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
MASKMOVDQU – Store Selected Bytes of Double Quadword	
xmmreg to xmmreg	01100110:00001111:11110111:11 xmmreg1 xmmreg2
CLFLUSH – Flush Cache Line	
mem	00001111:10101110:mod r/m
MOVNTPD – Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint	
xmmreg to mem	01100110:00001111:00101011: mod xmmreg r/m
MOVNTDQ – Store Double Quadword Using Non-Temporal Hint	
xmmreg to mem	01100110:00001111:11100111: mod xmmreg r/m
MOVNTI – Store Doubleword Using Non-Temporal Hint	
reg to mem	00001111:11000011: mod reg r/m
PAUSE – Spin Loop Hint	11110011:10010000
LFENCE – Load Fence	00001111:10101110: 11 101 000
MFENCE – Memory Fence	00001111:10101110: 11 110 000

B.7.2. SSE3 のフォーマットとエンコーディングの表

本項の表は、開発コード名 Prescott 用の命令のフォーマットとエンコーディングを示している。SSE3 の一部の命令は、2 バイト・オペコードの一部として必須のプリフィックス 66H、F2H、F3H を必要とする。これらの必須のプリフィックスは表に記載されている。

表 B-23. SSE3 浮動小数点命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
ADDSUBPD – Add /Sub packed DP FP numbers from XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:11010000:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:11010000: mod xmmreg r/m
ADDSUBPS – Add /Sub packed SP FP numbers from XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:11010000:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:11010000: mod xmmreg r/m
HADDPD – Add horizontally packed DP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:01111100:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01111100: mod xmmreg r/m
HADDPS – Add horizontally packed SP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:01111100:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01111100: mod xmmreg r/m
HSUBPD – Sub horizontally packed DP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	01100110:00001111:01111101:11 xmmreg1 xmmreg2
mem to xmmreg	01100110:00001111:01111101: mod xmmreg r/m
HSUBPS – Sub horizontally packed SP FP numbers XMM2/Mem to XMM1	
xmmreg2 to xmmreg1	11110010:00001111:01111101:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:01111101: mod xmmreg r/m

表 B-24. SSE3 イベント管理命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
MONITOR – Set up a linear address range to be monitored by hardware	
eax, ecx, edx	0000 1111 : 0000 0001:11 001 000
MWAIT – Wait until write-back store performed within the range specified by the instruction MONITOR	
eax, ecx	0000 1111 : 0000 0001:11 001 001

表 B-25. SSE3 整数命令および移動命令のフォーマットとエンコーディング

命令およびフォーマット	エンコーディング
FISTTP – Store ST in int16 (chop) and pop	
m16int	11011 111 : mod[A] 001 r/m
FISTTP – Store ST in int32 (chop) and pop	
m32int	11011 011 : mod[A] 001 r/m
FISTTP – Store ST in int64 (chop) and pop	
m64int	11011 101 : mod[A] 001 r/m
LDDQU – Load unaligned integer 128-bit	
xmm, m128	11110010:00001111:11110000: mod[A] xmmreg r/m
MOVDDUP – Move 64 bits representing one DP data from XMM2/Mem to XMM1 and duplicate	
xmmreg2 to xmmreg1	11110010:00001111:00010010:11 xmmreg1 xmmreg2
mem to xmmreg	11110010:00001111:00010010: mod xmmreg r/m
MOVSHDUP – Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate high	
xmmreg2 to xmmreg1	11110011:00001111:00010110:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:00010110: mod xmmreg r/m
MOVSLDUP – Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate low	
xmmreg2 to xmmreg1	11110011:00001111:00010010:11 xmmreg1 xmmreg2
mem to xmmreg	11110011:00001111:00010010: mod xmmreg r/m

B.8. 浮動小数点命令のフォーマットおよびエンコーディング

表 B-26. に、浮動小数点命令に使用される 5 個の異なるフォーマットを示す。すべての場合に、命令は少なくとも 2 バイトの長さであり、ビットパターン 11011 で始まる。

表 B-26. 汎用浮動小数点命令フォーマット

	命令												オプションのフィールド	
	第 1 バイト				第 2 バイト									
1	11011	OPA		1	mod		1	OPB		r/m			s-i-b	disp
2	11011	MF		OPA	mod		OPB			r/m			s-i-b	disp
3	11011	d	P	OPA	1	1	OPB	R		ST(i)				
4	11011	0	0	1	1	1	1	OP						
5	11011	0	1	1	1	1	1	OP						
	15–11	10	9	8	7	6	5	4	3	2	1	0		

MF = メモリ・フォーマット
- 00 – 32 ビット実数
- 01 – 32 ビット整数
- 10 – 64 ビット実数
- 11 – 16 ビット整数

P = ポップ
- 0 – スタックをポップしない
- 1 – 操作後にスタックをポップ

d = デスティネーション
- 0 – デスティネーションは ST(0)
- 1 – デスティネーションは ST(i)

R XOR d = 0 – デスティネーション OP ソース
R XOR d = 1 – ソース OP デスティネーション

ST(i) = レジスタスタック要素 *i*
- 000 = スタックのトップ
- 001 = 2 番目のスタック要素
- .
- .
- .
- 111 = 8 番目のスタック要素

ModR/M バイトの Mod フィールドと R/M フィールドは、整数命令の対応するフィールドと同じように解釈される。SIB バイトと disp（ディスプレースメント）は、Mod フィールドと R/M フィールドをもつ命令にオプションで存在する。それらの存在は、整数命令に対するのと同様に、Mod と R/M の値に依存する。

表 B-27. に、浮動小数点命令のフォーマットおよびエンコーディングを示す。

表 B-27. 浮動小数点命令のフォーマットおよびエンコーディング

命令およびフォーマット	エンコーディング
F2XM1 – Compute $2^{ST(0)} - 1$	11011 001 : 1111 0000
FABS – Absolute Value	11011 001 : 1110 0001
FADD – Add	
ST(0) ← ST(0) + 32-bit memory	11011 000 : mod 000 r/m
ST(0) ← ST(0) + 64-bit memory	11011 100 : mod 000 r/m
ST(d) ← ST(0) + ST(i)	11011 d00 : 11 000 ST(i)
FADDP – Add and Pop	
ST(0) ← ST(0) + ST(i)	11011 110 : 11 000 ST(i)
FBLD – Load Binary Coded Decimal	11011 111 : mod 100 r/m
FBSTP – Store Binary Coded Decimal and Pop	11011 111 : mod 110 r/m
FCHS – Change Sign	11011 001 : 1110 0000
FCLEX – Clear Exceptions	11011 011 : 1110 0010
FCOM – Compare Real	
32-bit memory	11011 000 : mod 010 r/m
64-bit memory	11011 100 : mod 010 r/m
ST(i)	11011 000 : 11 010 ST(i)
FCOMP – Compare Real and Pop	
32-bit memory	11011 000 : mod 011 r/m
64-bit memory	11011 100 : mod 011 r/m
ST(i)	11011 000 : 11 011 ST(i)
FCOMPP – Compare Real and Pop Twice	11011 110 : 11 011 001
FCOMIP – Compare Real, Set EFLAGS, and Pop	11011 111 : 11 110 ST(i)
FCOS – Cosine of ST(0)	11011 001 : 1111 1111
FDECSTP – Decrement Stack-Top Pointer	11011 001 : 1111 0110
FDIV – Divide	
ST(0) ← ST(0) ÷ 32-bit memory	11011 000 : mod 110 r/m
ST(0) ← ST(0) ÷ 64-bit memory	11011 100 : mod 110 r/m
ST(d) ← ST(0) ÷ ST(i)	11011 d00 : 1111 R ST(i)

表 B-27. 浮動小数点命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
FDIVP – Divide and Pop	
ST(0) ← ST(0) ÷ ST(i)	11011 110 : 1111 1 ST(i)
FDIVR – Reverse Divide	
ST(0) ← 32-bit memory ÷ ST(0)	11011 000 : mod 111 r/m
ST(0) ← 64-bit memory ÷ ST(0)	11011 100 : mod 111 r/m
ST(d) ← ST(i) ÷ ST(0)	11011 d00 : 1111 R ST(i)
FDIVRP – Reverse Divide and Pop	
ST(0) ← ST(i) ÷ ST(0)	11011 110 : 1111 0 ST(i)
FFREE – Free ST(i) Register	11011 101 : 1100 0 ST(i)
FIADD – Add Integer	
ST(0) ← ST(0) + 16-bit memory	11011 110 : mod 000 r/m
ST(0) ← ST(0) + 32-bit memory	11011 010 : mod 000 r/m
FICOM – Compare Integer	
16-bit memory	11011 110 : mod 010 r/m
32-bit memory	11011 010 : mod 010 r/m
FICOMP – Compare Integer and Pop	
16-bit memory	11011 110 : mod 011 r/m
32-bit memory	11011 010 : mod 011 r/m
FIDIV	
ST(0) ← ST(0) ÷ 16-bit memory	11011 110 : mod 110 r/m
ST(0) ← ST(0) ÷ 32-bit memory	11011 010 : mod 110 r/m
FIDIVR	
ST(0) ← 16-bit memory ÷ ST(0)	11011 110 : mod 111 r/m
ST(0) ← 32-bit memory ÷ ST(0)	11011 010 : mod 111 r/m
FILD – Load Integer	
16-bit memory	11011 111 : mod 000 r/m
32-bit memory	11011 011 : mod 000 r/m
64-bit memory	11011 111 : mod 101 r/m
FIMUL	
ST(0) ← ST(0) × 16-bit memory	11011 110 : mod 001 r/m
ST(0) ← ST(0) × 32-bit memory	11011 010 : mod 001 r/m
FINCSTP – Increment Stack Pointer	11011 001 : 1111 0111
FINIT – Initialize Floating-Point Unit	

表 B-27. 浮動小数点命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
FIST – Store Integer	
16-bit memory	11011 111 : mod 010 r/m
32-bit memory	11011 011 : mod 010 r/m
FISTP – Store Integer and Pop	
16-bit memory	11011 111 : mod 011 r/m
32-bit memory	11011 011 : mod 011 r/m
64-bit memory	11011 111 : mod 111 r/m
FISUB	
ST(0) ← ST(0) – 16-bit memory	11011 110 : mod 100 r/m
ST(0) ← ST(0) – 32-bit memory	11011 010 : mod 100 r/m
FISUBR	
ST(0) ← 16-bit memory – ST(0)	11011 110 : mod 101 r/m
ST(0) ← 32-bit memory – ST(0)	11011 010 : mod 101 r/m
FLD – Load Real	
32-bit memory	11011 001 : mod 000 r/m
64-bit memory	11011 101 : mod 000 r/m
80-bit memory	11011 011 : mod 101 r/m
ST(i)	11011 001 : 11 000 ST(i)
FLD1 – Load +1.0 into ST(0)	11011 001 : 1110 1000
FLDCW – Load Control Word	11011 001 : mod 101 r/m
FLDENV – Load FPU Environment	11011 001 : mod 100 r/m
FLDL2E – Load $\log_2(\varepsilon)$ into ST(0)	11011 001 : 1110 1010
FLDL2T – Load $\log_2(10)$ into ST(0)	11011 001 : 1110 1001
FLDLG2 – Load $\log_{10}(2)$ into ST(0)	11011 001 : 1110 1100
FLDLN2 – Load $\log_{\varepsilon}(2)$ into ST(0)	11011 001 : 1110 1101
FLDPI – Load π into ST(0)	11011 001 : 1110 1011
FLDZ – Load +0.0 into ST(0)	11011 001 : 1110 1110
FMUL – Multiply	
ST(0) ← ST(0) × 32-bit memory	11011 000 : mod 001 r/m
ST(0) ← ST(0) × 64-bit memory	11011 100 : mod 001 r/m
ST(d) ← ST(0) × ST(i)	11011 d00 : 1100 1 ST(i)
FMULP – Multiply	
ST(i) ← ST(0) × ST(i)	11011 110 : 1100 1 ST(i)
FNOP – No Operation	11011 001 : 1101 0000
FPATAN – Partial Arctangent	11011 001 : 1111 0011

表 B-27. 浮動小数点命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
FPREM – Partial Remainder	11011 001 : 1111 1000
FPREM1 – Partial Remainder (IEEE)	11011 001 : 1111 0101
FPTAN – Partial Tangent	11011 001 : 1111 0010
FRNDINT – Round to Integer	11011 001 : 1111 1100
FRSTOR – Restore FPU State	11011 101 : mod 100 r/m
FSAVE – Store FPU State	11011 101 : mod 110 r/m
FSCALE – Scale	11011 001 : 1111 1101
FSIN – Sine	11011 001 : 1111 1110
FSINCOS – Sine and Cosine	11011 001 : 1111 1011
FSQRT – Square Root	11011 001 : 1111 1010
FST – Store Real	
32-bit memory	11011 001 : mod 010 r/m
64-bit memory	11011 101 : mod 010 r/m
ST(i)	11011 101 : 11 010 ST(i)
FSTCW – Store Control Word	11011 001 : mod 111 r/m
FSTENV – Store FPU Environment	11011 001 : mod 110 r/m
FSTP – Store Real and Pop	
32-bit memory	11011 001 : mod 011 r/m
64-bit memory	11011 101 : mod 011 r/m
80-bit memory	11011 011 : mod 111 r/m
ST(i)	11011 101 : 11 011 ST(i)
FSTSW – Store Status Word into AX	11011 111 : 1110 0000
FSTSW – Store Status Word into Memory	11011 101 : mod 111 r/m
FSUB – Subtract	
ST(0) ← ST(0) – 32-bit memory	11011 000 : mod 100 r/m
ST(0) ← ST(0) – 64-bit memory	11011 100 : mod 100 r/m
ST(d) ← ST(0) – ST(i)	11011 d00 : 1110 R ST(i)
FSUBP – Subtract and Pop	
ST(0) ← ST(0) – ST(i)	11011 110 : 1110 1 ST(i)
FSUBR – Reverse Subtract	
ST(0) ← 32-bit memory – ST(0)	11011 000 : mod 101 r/m
ST(0) ← 64-bit memory – ST(0)	11011 100 : mod 101 r/m
ST(d) ← ST(i) – ST(0)	11011 d00 : 1110 R ST(i)
FSUBRP – Reverse Subtract and Pop	
ST(i) ← ST(i) – ST(0)	11011 110 : 1110 0 ST(i)

表 B-27. 浮動小数点命令のフォーマットおよびエンコーディング（続き）

命令およびフォーマット	エンコーディング
FTST – Test	11011 001 : 1110 0100
FUCOM – Unordered Compare Real	11011 101 : 1110 0 ST(i)
FUCOMP – Unordered Compare Real and Pop	11011 101 : 1110 1 ST(i)
FUCOMPP – Unordered Compare Real and Pop Twice	11011 010 : 1110 1001
FUCOMI – Unorderd Compare Real and Set EFLAGS	11011 011 : 11 101 ST(i)
FUCOMIP – Unorderd Compare Real, Set EFLAGS, and Pop	11011 111 : 11 101 ST(i)
FXAM – Examine	11011 001 : 1110 0101
FXCH – Exchange ST(0) and ST(i)	11011 001 : 1100 1 ST(i)
FXTRACT – Extract Exponent and Significand	11011 001 : 1111 0100
FYL2X – ST(1) × $\log_2$(ST(0))	11011 001 : 1111 0001
FYL2XP1 – ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 : 1111 1001
FWAIT – Wait until FPU Ready	1001 1011

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

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

本章の 2 つの表は、インテル® MMX® テクノロジ命令、SSE、SSE2、SSE3 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をまとめたものである。

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。本章は、これらの組み込み関数の使用法を示している。

表 C-1. は簡単な組み込み関数、表 C-2. は複合組み込み関数を示す。「複合組み込み関数」とは、2 つ以上の命令で構成される組み込み関数のことである。

インテル C/C++ コンパイラの組み込み関数名には、次の命名規則が適用される。

_mm_<intrin_op>_<suffix>

各要素の意味は以下のとおりである。

<intrin_op>　組み込み関数の基本動作を示す（例えば、加算の場合は add、減算の場合は sub）。

<suffix>　命令の操作対象となるデータ型を示す。各サフィックスの最初の 1 文字または 2 文字は、パックドデータ（p）、パックド拡張データ（ep）またはスカラデータ（s）を表す。その他の文字は、以下のデータ型を示す。

s	単精度浮動小数点
d	倍精度浮動小数点
i128	符号付き 128 ビット整数
i64	符号付き 64 ビット整数
u64	符号なし 64 ビット整数
i32	符号付き 32 ビット整数
u32	符号なし 32 ビット整数

i16	符号付き 16 ビット整数
u16	符号なし 16 ビット整数
i8	符号付き 8 ビット整数
u8	符号なし 8 ビット整数

変数 r は、通常は組み込み関数の戻り値を示す。変数名に付加された数字は、パックド・オブジェクトの要素を示す。例えば、r0 は r の最下位ワードである。一部の組み込み関数は、2 つ以上の命令を必要とするため、「複合」組み込み関数と呼ばれる。

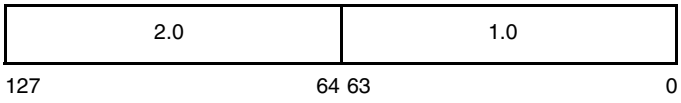
パックド値は右から左の順に表現され、スカラ操作には最下位の値が使用される。例えば、次の操作の例を考える。

```
double a[2] = {1.0, 2.0};
__m128d t = _mm_load_pd(a);
```

この操作の結果は、次の各操作の結果と同じになる。

```
__m128d t = _mm_set_pd(2.0, 1.0);
__m128d t = _mm_setr_pd(1.0, 2.0);
```

つまり、値 t を保持する XMM レジスタは、次の図のようになる。



「スカラ」要素は 1.0 である。命令の性質上、一部の組み込み関数は、引き数として即値（定数整数リテラル）を必要とする。

コード内で組み込み関数を使用するには、次の構文の行を挿入する。

```
data_type intrinsic_name (parameters)
```

各要素の意味は次のとおりである。

data_type	戻りデータ型。void、int、__m64、__m128、__m128d、または __m128i である。void を返すのは、_mm_empty 組み込み関数だけである。
intrinsic_name	組み込み関数の名前。C/C++ コード内で、実際の命令の代わりに使用できる関数として機能する。
parameters	各組み込み関数が必要とするパラメータ。

C.1. 簡単な組み込み関数

表 C-1. 簡単な組み込み関数

ニーモニック	組み込み関数	説明
ADDPD	__m128d _mm_add_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点（DP FP）値を加算する。
ADDPS	__m128 _mm_add_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点（SP FP）値を加算する。
ADDSD	__m128d _mm_add_sd(__m128d a, __m128d b)	a と b の最下位の倍精度浮動小数点値（DP FP）を加算する。上位の 3 つの倍精度浮動小数点値は、a からそのまま渡される。
ADDSS	__m128 _mm_add_ss(__m128 a, __m128 b)	a と b の最下位の単精度浮動小数点（SP FP）値を加算する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
ADDSUBPD	__m128d _mm_addsub_pd(__m128d a, __m128d b)	XMM2/Mem と XMM1 のパック ド倍精度浮動小数点値を加算 / 減算する。
ADDSUBPS	__m128 _mm_addsub_ps(__m128 a, __m128 b)	XMM2/Mem と XMM1 のパック ド単精度浮動小数点値を加算 / 減算する。
ANDNPD	__m128d _mm_andnot_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の間でビット単位の AND-NOT 演算を実行する。
ANDNPS	__m128 _mm_andnot_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の間でビット単位の AND-NOT 演算を実行する。
ANDPD	__m128d _mm_and_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の間でビット単位の AND 演算を実行する。
ANDPS	__m128 _mm_and_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の間でビット単位の AND 演算を実行する。
CLFLUSH	void _mm_clflush(void const *p)	コヒーレンシ・ドメイン内のすべてのキャッシュから、p が入っているキャッシュ・ラインをフラッシュし、無効化する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
CMPPD	__m128d _mm_cmpeq_pd(__m128d a, __m128d b)	「等しい」の条件で比較する。
	__m128d _mm_cmplt_pd(__m128d a, __m128d b)	「より小さい」の条件で比較する。
	__m128d _mm_cmple_pd(__m128d a, __m128d b)	「より小さいか等しい」の条件で比較する。
	__m128d _mm_cmpgt_pd(__m128d a, __m128d b)	「より大きい」の条件で比較する。
	__m128d _mm_cmpge_pd(__m128d a, __m128d b)	「より大きいか等しい」の条件で比較する。
	__m128d _mm_cmpneq_pd(__m128d a, __m128d b)	「等しくない」の条件で比較する。
	__m128d _mm_cmpnlt_pd(__m128d a, __m128d b)	「より小さくない」の条件で比較する。
	__m128d _mm_cmpngt_pd(__m128d a, __m128d b)	「より大きくない」の条件で比較する。
	__m128d _mm_cmpnge_pd(__m128d a, __m128d b)	「より大きくなく等しくない」の条件で比較する。
	__m128d _mm_cmpord_pd(__m128d a, __m128d b)	「順序付けあり」の条件で比較する。
	__m128d _mm_cmpunord_pd(__m128d a, __m128d b)	「順序付けなし」の条件で比較する。
	__m128d _mm_cmpnle_pd(__m128d a, __m128d b)	「より小さくなく等しくない」の条件で比較する。
CMPPS	__m128 _mm_cmpeq_ps(__m128 a, __m128 b)	「等しい」の条件で比較する。
	__m128 _mm_cmplt_ps(__m128 a, __m128 b)	「より小さい」の条件で比較する。
	__m128 _mm_cmple_ps(__m128 a, __m128 b)	「より小さいか等しい」の条件で比較する。
	__m128 _mm_cmpgt_ps(__m128 a, __m128 b)	「より大きい」の条件で比較する。
	__m128 _mm_cmpge_ps(__m128 a, __m128 b)	「より大きいか等しい」の条件で比較する。
CMPPS	__m128 _mm_cmpneq_ps(__m128 a, __m128 b)	「等しくない」の条件で比較する。
	__m128 _mm_cmpnlt_ps(__m128 a, __m128 b)	「より小さくない」の条件で比較する。
	__m128 _mm_cmpngt_ps(__m128 a, __m128 b)	「より大きくない」の条件で比較する。
	__m128 _mm_cmpnge_ps(__m128 a, __m128 b)	「より大きくなく等しくない」の条件で比較する。
	__m128 _mm_cmpord_ps(__m128 a, __m128 b)	「順序付けあり」の条件で比較する。
	__m128 _mm_cmpunord_ps(__m128 a, __m128 b)	「順序付けなし」の条件で比較する。
	__m128 _mm_cmpnle_ps(__m128 a, __m128 b)	「より小さくなく等しくない」の条件で比較する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
CMPSD	__m128d _mm_cmpeq_sd(__m128d a, __m128d b)	「等しい」の条件で比較する。
	__m128d _mm_cmplt_sd(__m128d a, __m128d b)	「より小さい」の条件で比較する。
	__m128d _mm_cmple_sd(__m128d a, __m128d b)	「より小さいか等しい」の条件で比較する。
	__m128d _mm_cmpgt_sd(__m128d a, __m128d b)	「より大きい」の条件で比較する。
	__m128d _mm_cmpge_sd(__m128d a, __m128d b)	「より大きいか等しい」の条件で比較する。
	__m128 _mm_cmpneq_sd(__m128d a, __m128d b)	「等しくない」の条件で比較する。
	__m128 _mm_cmpnlt_sd(__m128d a, __m128d b)	「より小さくない」の条件で比較する。
	__m128d _mm_cmpnle_sd(__m128d a, __m128d b)	「より大きくない」の条件で比較する。
	__m128d _mm_cmpngt_sd(__m128d a, __m128d b)	「より大きくなく等しくない」の条件で比較する。
	__m128d _mm_cmpnge_sd(__m128d a, __m128d b)	「順序付けあり」の条件で比較する。
	__m128d _mm_cmpord_sd(__m128d a, __m128d b)	「順序付けなし」の条件で比較する。
	__m128d _mm_cmpunord_sd(__m128d a, __m128d b)	「より小さくなく等しくない」の条件で比較する。
CMPSS	__m128 _mm_cmpeq_ss(__m128 a, __m128 b)	「等しい」の条件で比較する。
	__m128 _mm_cmplt_ss(__m128 a, __m128 b)	「より小さい」の条件で比較する。
	__m128 _mm_cmple_ss(__m128 a, __m128 b)	「より小さいか等しい」の条件で比較する。
	__m128 _mm_cmpgt_ss(__m128 a, __m128 b)	「より大きい」の条件で比較する。
	__m128 _mm_cmpge_ss(__m128 a, __m128 b)	「より大きいか等しい」の条件で比較する。
	__m128 _mm_cmpneq_ss(__m128 a, __m128 b)	「等しくない」の条件で比較する。
	__m128 _mm_cmpnlt_ss(__m128 a, __m128 b)	「より小さくない」の条件で比較する。
	__m128 _mm_cmpnle_ss(__m128 a, __m128 b)	「より大きくない」の条件で比較する。
	__m128 _mm_cmpngt_ss(__m128 a, __m128 b)	「より大きくなく等しくない」の条件で比較する。
	__m128 _mm_cmpnge_ss(__m128 a, __m128 b)	「順序付けあり」の条件で比較する。
	__m128 _mm_cmpord_ss(__m128 a, __m128 b)	「順序付けなし」の条件で比較する。
	__m128 _mm_cmpunord_ss(__m128 a, __m128 b)	「より小さくなく等しくない」の条件で比較する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
COMISD	int _mm_comieq_sd(__m128d a, __m128d b)	「a と b が等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a と b が等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comilt_sd(__m128d a, __m128d b)	「a が b より小さい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より小さい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comile_sd(__m128d a, __m128d b)	「a が b より小さいか等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より小さいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comigt_sd(__m128d a, __m128d b)	「a が b より大きい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より大きい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comige_sd(__m128d a, __m128d b)	「a が b より大きいか等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より大きいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comineq_sd(__m128d a, __m128d b)	「a と b が等しくない」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a と b が等しくない場合は、1 が返される。それ以外の場合は、0 が返される。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
COMISS	int _mm_comieq_ss(__m128 a, __m128 b)	「a と b が等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a と b が等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comilt_ss(__m128 a, __m128 b)	「a が b より小さい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より小さい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comile_ss(__m128 a, __m128 b)	「a が b より小さいか等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より小さいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comigt_ss(__m128 a, __m128 b)	「a が b より大きい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より大きい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comige_ss(__m128 a, __m128 b)	「a が b より大きいか等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より大きいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_comineq_ss(__m128 a, __m128 b)	「a と b が等しくない」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a と b が等しくない場合は、1 が返される。それ以外の場合は、0 が返される。
CVTDQ2PD	__m128d _mm_cvtepi32_pd(__m128i a)	a のパックド形式の最下位の 2 つの 32 ビット符号付き整数値を、2 つの倍精度浮動小数点値に変換する。
CVTDQ2PS	__m128 _mm_cvtepi32_ps(__m128i a)	a のパックド形式の 4 つの 32 ビット符号付き整数値を、4 つの単精度浮動小数点値に変換する。
CVTPD2DQ	__m128i _mm_cvtpd_epi32(__m128d a)	a の 2 つの倍精度浮動小数点値を、2 つの 32 ビット符号付き整数値に変換する。
CVTPD2PI	__m64 _mm_cvtpd_pi32(__m128d a)	a の 2 つの倍精度浮動小数点値を、2 つの 32 ビット符号付き整数値に変換する。
CVTPD2PS	__m128 _mm_cvtpd_ps(__m128d a)	a の 2 つの倍精度浮動小数点値を、2 つの倍精度浮動小数点値に変換する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
CVTPI2PD	__m128d _mm_cvtpi32_pd(__m64 a)	a の 2 つの 32 ビット整数値を、2 つの単精度浮動小数点値に変換する。
CVTPI2PS	__m128 _mm_cvt_pi2ps(__m128 a, __m64 b) __m128 _mm_cvtpi32_ps(__m128 a, __m64 b)	b のパック ド形式の 2 つの 32 ビット整数値を、2 つの単精度浮動小数点値に変換する。上位の 2 つの単精度浮動小数点値は、a からそのまま渡される。
CVTPS2DQ	__m128i _mm_cvtps_epi32(__m128 a)	現在の丸めモードにしたがって、a の 4 つの単精度浮動小数点値を 4 つの 32 ビット符号付き整数に変換する。
CVTPS2PD	__m128d _mm_cvtps_pd(__m128 a)	a の最下位の 2 つの単精度浮動小数点値を、倍精度浮動小数点値に変換する。
CVTPS2PI	__m64 _mm_cvt_ps2pi(__m128 a) __m64 _mm_cvtps_pi32(__m128 a)	現在の丸めモードにしたがって、a の下位の 2 つの単精度浮動小数点値を 32 ビット整数に変換し、パック ド形式の整数を返す。
CVTSD2SI	int _mm_cvtsd_si32(__m128d a)	a の最下位の倍精度浮動小数点値を、32 ビット整数値に変換する。
CVTSD2SS	__m128 _mm_cvtsd_ss(__m128 a, __m128d b)	b の最下位の倍精度浮動小数点値を、単精度浮動小数点値に変換する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
CVTSI2SD	__m128d _mm_cvtsi32_sd(__m128d a, int b)	32 ビット整数値 b を倍精度浮動小数点値に変換する。上位の倍精度浮動小数点値は、a からそのまま渡される。
CVTSI2SS	__m128 _mm_cvt_si2ss(__m128 a, int b) __m128 _mm_cvtsi32_ss(__m128a, int b)	32 ビット整数値 b を単精度浮動小数点値に変換する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
CVTSS2SD	__m128d _mm_cvtss_sd(__m128d a, __m128 b)	b の最下位の単精度浮動小数点値を、倍精度浮動小数点値に変換する。上位の倍精度浮動小数点値は、a からそのまま渡される。
CVTSS2SI	int _mm_cvt_ss2si(__m128 a) int _mm_cvtss_si32(__m128 a)	a の最下位の単精度浮動小数点値を 32 ビット整数に変換する。
CVTTPD2DQ	__m128i _mm_cvttpd_epi32(__m128d a)	切り捨てを使用して、a の 2 つの倍精度浮動小数点値を、2 つの 32 ビット符号付き整数値に変換する。上位の 2 つの整数値は、0 である。
CVTTPD2PI	__m64 _mm_cvttpd_pi32(__m128d a)	切り捨てを使用して、a の 2 つの倍精度浮動小数点値を、32 ビット符号付き整数値に変換する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
CVTTPS2DQ	__m128i _mm_cvttps_epi32(__m128 a)	切り捨てを使用して、a の 4 つの単精度浮動小数点値を、4 つの 32 ビット整数に変換する。
CVTTPS2PI	__m64 _mm_cvtt_ps2pi(__m128 a) __m64 _mm_cvttps_pi32(__m128 a)	切り捨てを使用して、a の下位の 2 つの単精度浮動小数点値を 2 つの 32 ビット整数に変換し、パックド形式の整数を返す。
CVTTSD2SI	int _mm_cvttsd_si32(__m128d a)	切り捨てを使用して、a の最下位の倍精度浮動小数点値を、32 ビット符号付き整数に変換する。
CVTTSS2SI	int _mm_cvtt_ss2si(__m128 a) int _mm_cvttss_si32(__m128 a)	現在の丸めモードにしたがって、a の最下位の単精度浮動小数点値を 32 ビット整数に変換する。
	__m64 _mm_cvtsi32_si64(int i)	整数オブジェクト i を 64 ビットの __m64 オブジェクトに変換する。整数値は 64 ビットにゼロ拡張される。
	int _mm_cvtsi64_si32(__m64 m)	__m64 オブジェクト m の下位の 32 ビットを整数に変換する。
DIVPD	__m128d _mm_div_pd(__m128d a, __m128d b)	a の 2 つの倍精度浮動小数点値を b の 2 つの倍精度浮動小数点値で割る。
DIVPS	__m128 _mm_div_ps(__m128 a, __m128 b)	a の 4 つの単精度浮動小数点値を b の 4 つの単精度浮動小数点値で割る。
DIVSD	__m128d _mm_div_sd(__m128d a, __m128d b)	a の最下位の倍精度浮動小数点値を b の最下位の倍精度浮動小数点値で割る。上位の 3 つの倍精度浮動小数点値は、a からそのまま渡される。
DIVSS	__m128 _mm_div_ss(__m128 a, __m128 b)	a の最下位の単精度浮動小数点値を b の最下位の単精度浮動小数点値で割る。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
EMMS	void _mm_empty()	MMX テクノロジの状態をクリアする。
HADDPD	__m128d _mm_hadd_pd(__m128d a, __m128d b)	XMM2/Mem 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。
HADDPS	__m128 _mm_hadd_ps(__m128 a, __m128 b)	XMM2/Mem 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。
HSUBPD	__m128d _mm_hsub_pd(__m128d a, __m128d b)	XMM2/Mem 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。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
HSUBPS	__m128 _mm_hsub_ps(__m128 a, __m128 b)	XMM2/Mem 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。
LDDQU	__m128i _mm_lddqu_si128(__m128i const *p)	Mem から XMM レジスタに 128 ビットをロードする。
LDMXCSR	_mm_setcsr(unsigned int i)	制御レジスタを指定された値に設定する。
LFENCE	void _mm_lfence(void)	プログラム順序で LFENCE 命令に先行するすべてのロード命令が、その LFENCE の後（後続）に実行されるどのロード命令よりも前にグローバルにアクセス可能になることを保証する。
MASKMOVDQU	void _mm_maskmoveu_si128(__m128i d, __m128i n, char *p)	d のバイト要素をアドレス p に条件付きでストアする。セレクタ n の各バイトの最上位ビットによって、それに対応する d のバイトがストアされるかどうかが決まる。
MASKMOVQ	void _mm_maskmove_si64(__m64 d, __m64 n, char *p)	d のバイト要素をアドレス p に条件付きでストアする。セレクタ n の各バイトの最上位ビットによって、それに対応する d のバイトがストアされるかどうかが決まる。
MAXPD	__m128d _mm_max_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の最大値を計算する。
MAXPS	__m128 _mm_max_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の最大値を計算する。
MAXSD	__m128d _mm_max_sd(__m128d a, __m128d b)	a と b の最下位の倍精度浮動小数点値の最大値を計算する。上位の倍精度浮動小数点値は、a からそのまま渡される。
MAXSS	__m128 _mm_max_ss(__m128 a, __m128 b)	a と b の最下位の単精度浮動小数点値の最大値を計算する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
MFENCE	void _mm_mfence(void)	プログラムの順序でメモリフェンス命令に先行するすべてのメモリアクセスが、フェンスに後続するメモリアクセス命令より前に、グローバルにアクセス可能になることを保証する。
MINPD	__m128d _mm_min_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の最小値を計算する。
MINPS	__m128 _mm_min_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の最小値を計算する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
MINSD	__m128d _mm_min_sd(__m128d a, __m128d b)	a と b の最下位の倍精度浮動小数点値の最小値を計算する。上位の倍精度浮動小数点値は、a からそのまま渡される。
MINSS	__m128 _mm_min_ss(__m128 a, __m128 b)	a と b の最下位の単精度浮動小数点値の最小値を計算する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
MONITOR	void _mm_monitor(void const *p, unsigned extensions, unsigned hints)	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。
MOVAPD	__m128d _mm_load_pd(double * p)	2 つの倍精度浮動小数点値をロードする。アドレス p は 16 バイト・アライメントでなければならない。
	void_mm_store_pd(double *p, __m128d a)	2 つの倍精度浮動小数点値をアドレス p にストアする。アドレス p は 16 バイト・アライメントでなければならない。
MOVAPS	__m128 _mm_load_ps(float * p)	4 つの単精度浮動小数点値をロードする。アドレス p は 16 バイト・アライメントでなければならない。
	void_mm_store_ps(float *p, __m128 a)	4 つの単精度浮動小数点値をストアする。アドレス p は 16 バイト・アライメントでなければならない。
MOVD	__m128i _mm_cvtsi32_si128(int a)	32 ビット整数 a を 128 ビット・デスティネーションの最下位 32 ビットに移動し、上位ビットをゼロ拡張する。
	int _mm_cvtsi128_si32(__m128i a)	a の最下位の 32 ビット整数を 32 ビット符号付き整数に移動する。
	__m64 _mm_cvtsi32_si64(int a)	32 ビット整数 a を 64 ビット・デスティネーションの下位 32 ビットに移動し、上位ビットをゼロ拡張する。
	int _mm_cvtsi64_si32(__m64 a)	a の下位の 32 ビット整数を 32 ビット符号付き整数に移動する。
MOVDDUP	__m128d _mm_movedup_pd(__m128d a) __m128d _mm_loaddup_pd(double const * dp)	下位の倍精度データ要素を表す 64 ビットを XMM2/Mem から XMM1 レジスタに移動し、複製する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
MOVDQA	__m128i _mm_load_si128(__m128i * p)	p から 128 ビット値をロードする。アドレス p は 16 バイト・アライメントでなければならない。
	void_mm_store_si128(__m128i *p, __m128i a)	a の 128 ビット値をアドレス p にストアする。アドレス p は 16 バイト・アライメントでなければならない。
MOVDQU	__m128i _mm_loadu_si128(__m128i * p)	p から 128 ビット値をロードする。アドレス p は 16 バイト・アライメントである必要はない。
	void_mm_storeu_si128(__m128i *p, __m128i a)	a の 128 ビット値をアドレス p にストアする。アドレス p は 16 バイト・アライメントである必要はない。
MOVDQ2Q	__m64 _mm_movepi64_pi64(__m128i a)	a の最下位の 64 ビットを __m64 タイプに返す。
MOVHLPS	__m128 _mm_movehl_ps(__m128 a, __m128 b)	b の上位の 2 つの単精度浮動小数点値を、結果の下位の 2 つの単精度浮動小数点値に移動する。a の上位の 2 つの単精度浮動小数点値は、そのまま結果に渡される。
MOVHPD	__m128d _mm_loadh_pd(__m128d a, double * p)	倍精度浮動小数点値を、アドレス p からデスティネーションの上位の 64 ビットにロードする。下位の 64 ビットは、a からそのまま渡される。
	void _mm_storeh_pd(double * p, __m128d a)	a の上位の倍精度浮動小数点値をアドレス p にストアする。
MOVHPS	__m128 _mm_loadh_pi(__m128 a, __m64 * p)	上位の 2 つの単精度浮動小数点値として、アドレス p から 64 ビットのデータをロードする。下位の 2 つの値は、a からそのまま渡される。
	void _mm_storeh_pi(__m64 * p, __m128 a)	a の上位の 2 つの単精度浮動小数点値をアドレス p にストアする。
MOVLPD	__m128d _mm_loadl_pd(__m128d a, double * p)	倍精度浮動小数点値を、アドレス p からデスティネーションの下位の 64 ビットにロードする。上位の 64 ビットは、a からそのまま渡される。
	void _mm_storel_pd(double * p, __m128d a)	a の下位の倍精度浮動小数点値をアドレス p にストアする。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
MOVLPS	__m128 _mm_loadl_pi(__m128 a, __m64 *p)	下位の 2 つの単精度浮動小数点値として、アドレス p から 64 ビットのデータをロードする。上位の 2 つの値は、a からそのまま渡される。
	void_mm_storel_pi(__m64 * p, __m128 a)	a の下位の 2 つの単精度浮動小数点値をアドレス p にストアする。
MOVLHPS	__m128 _mm_movelh_ps(__m128 a, __m128 b)	b の下位の 2 つの単精度浮動小数点値を、結果の上位の 2 つの単精度浮動小数点値に移動する。a の下位の 2 つの単精度浮動小数点値は、そのまま結果に渡される。
MOVMSKPD	int _mm_movemask_pd(__m128d a)	a の 2 つの倍精度浮動小数点値の符号付きビットから 2 ビット・マスクを作成する。
MOVMSKPS	int _mm_movemask_ps(__m128 a)	4 つの単精度浮動小数点値の最上位ビットから 4 ビット・マスクを作成する。
MOVNTDQ	void_mm_stream_si128(__m128i * p, __m128i a)	キャッシュを汚染せずに、a のデータをアドレス p にストアする。p が入っているキャッシュ・ラインがすでにキャッシュ内にある場合は、キャッシュは更新される。アドレスは 16 バイト・アライメントでなければならない。
MOVNTPD	void_mm_stream_pd(double * p, __m128d a)	キャッシュを汚染せずに、a のデータをアドレス p にストアする。アドレスは 16 バイト・アライメントでなければならない。
MOVNTPS	void_mm_stream_ps(float * p, __m128 a)	キャッシュを汚染せずに、a のデータをアドレス p にストアする。アドレスは 16 バイト・アライメントでなければならない。
MOVNTI	void_mm_stream_si32(int * p, int a)	キャッシュを汚染せずに、a のデータをアドレス p にストアする。
MOVNTQ	void_mm_stream_pi(__m64 * p, __m64 a)	キャッシュを汚染せずに、a のデータをアドレス p にストアする。
MOVQ	__m128i _mm_loadl_epi64(__m128i * p)	p から下位 64 ビットをデスティネーションの下位 64 ビットにロードし、上位 64 ビットをゼロ拡張する。
	void_mm_storel_epi64(_m128i * p, __m128i a)	a の下位 64 ビットを p の下位 64 ビットにストアする。
	__m128i _mm_move_epi64(__m128i a)	a の下位 64 ビットをデスティネーションの下位 64 ビットに移動する。上位 64 ビットはクリアされる。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
MOVQ2DQ	__m128i _mm_movpi64_epi64(__m64 a)	a の 64 ビットをデスティネーションの下位 64 ビットに移動し、上位ビットをゼロ拡張する。
MOVSD	__m128d _mm_load_sd(double * p)	倍精度浮動小数点値を p から最下位の倍精度浮動小数点値にロードし、上位の倍精度浮動小数点値をクリアする。アドレス p は 16 バイト・アライメントである必要はない。
	void_mm_store_sd(double * p, __m128d a)	最下位の倍精度浮動小数点値をアドレス p にストアする。アドレス p は 16 バイト・アライメントである必要はない。
	__m128d _mm_move_sd(__m128d a, __m128d b)	b の最下位の倍精度浮動小数点値をデスティネーションに設定する。上位の倍精度浮動小数点値は、a からそのまま渡される。
MOVSHDUP	__m128 _mm_movehdup_ps(__m128 a)	パックド単精度データ要素を表す 128 ビットを XMM2/Mem から XMM1 レジスタに移動し、上位の要素を複製する。
MOVSLDUP	__m128 _mm_moveldup_ps(__m128 a)	パックド単精度データ要素を表す 128 ビットを XMM2/Mem から XMM1 レジスタに移動し、下位の要素を複製する。
MOVSS	__m128 _mm_load_ss(float * p)	単精度浮動小数点値を最下位ワードにロードし、上位の 3 ワードをクリアする。
	void_mm_store_ss(float * p, __m128 a)	最下位の単精度浮動小数点値をストアする。
	__m128 _mm_move_ss(__m128 a, __m128 b)	最下位ワードを b の単精度浮動小数点値に設定する。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
MOVUPD	__m128d _mm_loadu_pd(double * p)	p から 2 つの倍精度浮動小数点値をロードする。アドレス p が 16 バイト・アライメントである必要はない。
	void_mm_storeu_pd(double *p, __m128d a)	a の 2 つの倍精度浮動小数点値を p にストアする。アドレス p が 16 バイト・アライメントである必要はない。
MOVUPS	__m128 _mm_loadu_ps(float * p)	4 つの単精度浮動小数点値をロードする。アドレスが 16 バイト・アライメントである必要はない。
	void_mm_storeu_ps(float *p, __m128 a)	4 つの単精度浮動小数点値をストアする。アドレスが 16 バイト・アライメントである必要はない。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
MULPD	__m128d _mm_mul_pd(__m128d a, __m128d b)	a の 2 つの倍精度浮動小数点値に b の 2 つの倍精度浮動小数点値を掛ける。
MULPS	__m128 _mm_mul_ss(__m128 a, __m128 b)	a の 4 つの単精度浮動小数点値に b の 4 つの単精度浮動小数点値を掛ける。
MULSD	__m128d _mm_mul_sd(__m128d a, __m128d b)	a の最下位の倍精度浮動小数点値に b の最下位の倍精度浮動小数点値を掛ける。上位の倍精度浮動小数点値は、a からそのまま渡される。
MULSS	__m128 _mm_mul_ss(__m128 a, __m128 b)	a の最下位の単精度浮動小数点値に b の最下位の単精度浮動小数点値を掛ける。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。
MWAIT	void _mm_mwait(unsigned extensions, unsigned hints)	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、インプリメンテーション・デペンド・オプティマイズド・ステータスに移行するように指示するヒント。
ORPD	__m128d _mm_or_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の間でビット単位の OR 演算を実行する。
ORPS	__m128 _mm_or_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の間でビット単位の OR 演算を実行する。
PACKSSWB	__m128i _mm_packs_epi16(__m128i m1, __m128i m2)	符号付きの飽和を使用して、m1 の 8 つの 16 ビット値を結果の下位の 8 つの 8 ビット値にパックし、m2 の 8 つの 16 ビット値を結果の上位の 8 つの 8 ビット値にパックする。
PACKSSWB	__m64 _mm_packs_pi16(__m64 m1, __m64 m2)	符号付きの飽和を使用して、m1 の 4 つの 16 ビット値を結果の下位の 4 つの 8 ビット値にパックし、m2 の 4 つの 16 ビット値を結果の上位の 4 つの 8 ビット値にパックする。
PACKSSDW	__m128i _mm_packs_epi32 (__m128i m1, __m128i m2)	符号付きの飽和を使用して、m1 の 4 つの 32 ビット値を結果の下位の 4 つの 16 ビット値にパックし、m2 の 4 つの 32 ビット値を結果の上位の 4 つの 16 ビット値にパックする。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PACKSSDW	__m64 _mm_packs_pi32 (__m64 m1, __m64 m2)	符号付きの飽和を使用して、m1 の 2 つの 32 ビット値を結果の下位の 2 つの 16 ビット値にパックし、m2 の 2 つの 32 ビット値を結果の上位の 2 つの 16 ビット値にパックする。
PACKUSWB	__m128i _mm_packus_epi16(__m128i m1, __m128i m2)	符号なしの飽和を使用して、m1 の 8 つの 16 ビット値を結果の下位の 8 つの 8 ビット値にパックし、m2 の 8 つの 16 ビット値を結果の上位の 8 つの 8 ビット値にパックする。
PACKUSWB	__m64 _mm_packs_pu16(__m64 m1, __m64 m2)	符号なしの飽和を使用して、m1 の 4 つの 16 ビット値を結果の下位の 4 つの 8 ビット値にパックし、m2 の 4 つの 16 ビット値を結果の上位の 4 つの 8 ビット値にパックする。
PADDB	__m128i _mm_add_epi8(__m128i m1, __m128i m2)	m1 の 16 個の 8 ビット値を、m2 の 16 個の 8 ビット値に加算する。
PADDB	__m64 _mm_add_pi8(__m64 m1, __m64 m2)	m1 の 8 つの 8 ビット値を、m2 の 8 つの 8 ビット値に加算する。
PADDW	__m128i _mm_addw_epi16(__m128i m1, __m128i m2)	m1 の 8 つの 16 ビット値を、m2 の 8 つの 16 ビット値に加算する。
PADDW	__m64 _mm_addw_pi16(__m64 m1, __m64 m2)	m1 の 4 つの 16 ビット値を、m2 の 4 つの 16 ビット値に加算する。
PADDD	__m128i _mm_add_epi32(__m128i m1, __m128i m2)	m1 の 4 つの 32 ビット値を、m2 の 4 つの 32 ビット値に加算する。
PADDD	__m64 _mm_add_pi32(__m64 m1, __m64 m2)	m1 の 2 つの 32 ビット値を、m2 の 2 つの 32 ビット値に加算する。
PADDQ	__m128i _mm_add_epi64(__m128i m1, __m128i m2)	m1 の 2 つの 64 ビット値を、m2 の 2 つの 64 ビット値に加算する。
PADDQ	__m64 _mm_add_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値を、m2 の 64 ビット値に加算する。
PADDSB	__m128i _mm_adds_epi8(__m128i m1, __m128i m2)	m1 の 16 個の符号付き 8 ビット値を、m2 の 16 個の符号付き 8 ビット値に加算し、飽和させる。
PADDSB	__m64 _mm_adds_pi8(__m64 m1, __m64 m2)	m1 の 8 つの符号付き 8 ビット値を、m2 の 8 つの符号付き 8 ビット値に加算し、飽和させる。
PADDSW	__m128i _mm_adds_epi16(__m128i m1, __m128i m2)	m1 の 8 つの符号付き 16 ビット値を、m2 の 8 つの符号付き 16 ビット値に加算し、飽和させる。
PADDSW	__m64 _mm_adds_pi16(__m64 m1, __m64 m2)	m1 の 4 つの符号付き 16 ビット値を、m2 の 4 つの符号付き 16 ビット値に加算し、飽和させる。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PADDUSB	__m128i _mm_adds_epu8(__m128i m1, __m128i m2)	m1 の 16 個の符号なし 8 ビット値を、m2 の 16 個の符号なし 8 ビット値に加算し、飽和させる。
PADDUSB	__m64 _mm_adds_pu8(__m64 m1, __m64 m2)	m1 の 8 つの符号なし 8 ビット値を、m2 の 8 つの符号なし 8 ビット値に加算し、飽和させる。
PADDUSW	__m128i _mm_adds_epu16(__m128i m1, __m128i m2)	m1 の 8 つの符号なし 16 ビット値を、m2 の 8 つの符号なし 16 ビット値に加算し、飽和させる。
PADDUSW	__m64 _mm_adds_pu16(__m64 m1, __m64 m2)	m1 の 4 つの符号なし 16 ビット値を、m2 の 4 つの符号なし 16 ビット値に加算し、飽和させる。
PAND	__m128i _mm_and_si128(__m128i m1, __m128i m2)	m1 の 128 ビット値と m2 の 128 ビット値の間でビット単位の AND 演算を実行する。
PAND	__m64 _mm_and_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値と m2 の 64 ビット値の間でビット単位の AND 演算を実行する。
PANDN	__m128i _mm_andnot_si128(__m128i m1, __m128i m2)	m1 の 128 ビット値に対して NOT 演算を実行し、その結果と m2 の 128 ビット値の間でビット単位の AND 演算を実行する。
PANDN	__m64 _mm_andnot_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値に対して NOT 演算を実行し、その結果と m2 の 64 ビット値の間でビット単位の AND 演算を実行する。
PAUSE	void _mm_pause(void)	次の命令の実行をプロセッサ固有の時間だけ遅らせる。アーキテクチャ上の状態は変更されない。
PAVGB	__m128i _mm_avg_epu8(__m128i a, __m128i b)	2 つのオペランドの 16 個の 8 ビット値のパックド平均を計算する。
PAVGB	__m64 _mm_avg_pu8(__m64 a, __m64 b)	2 つのオペランドの 8 つの 8 ビット値のパックド平均を計算する。
PAVGW	__m128i _mm_avg_epu16(__m128i a, __m128i b)	2 つのオペランドの 8 つの 16 ビット値のパックド平均を計算する。
PAVGW	__m64 _mm_avg_pu16(__m64 a, __m64 b)	2 つのオペランドの 4 つの 16 ビット値のパックド平均を計算する。
PCMPEQB	__m128i _mm_cmpeq_epi8(__m128i m1, __m128i m2)	m1 の 8 ビット値と m2 の 8 ビット値がすべて等しい場合は、結果の 8 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PCMPEQB	__m64 _mm_cmpeq_pi8(__m64 m1, __m64 m2)	m1 の 8 ビット値と m2 の 8 ビット値がすべて等しい場合は、結果の 8 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPEQW	__m128i _mm_cmpeq_epi16 (__m128i m1, __m128i m2)	m1 の 16 ビット値と m2 の 16 ビット値がすべて等しい場合は、結果の 62 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPEQW	__m64 _mm_cmpeq_pi16 (__m64 m1, __m64 m2)	m1 の 16 ビット値と m2 の 16 ビット値がすべて等しい場合は、結果の 16 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPEQD	__m128i _mm_cmpeq_epi32(__m128i m1, __m128i m2)	m1 の 32 ビット値と m2 の 32 ビット値がすべて等しい場合は、結果の 32 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPEQD	__m64 _mm_cmpeq_pi32(__m64 m1, __m64 m2)	m1 の 32 ビット値と m2 の 32 ビット値がすべて等しい場合は、結果の 32 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPGTB	__m128i _mm_cmpgt_epi8 (__m128i m1, __m128i m2)	m1 の 8 ビット値がすべて m2 の 8 ビット値より大きい場合は、結果の 8 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPGTB	__m64 _mm_cmpgt_pi8 (__m64 m1, __m64 m2)	m1 の 8 ビット値がすべて m2 の 8 ビット値より大きい場合は、結果の 8 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPGTW	__m128i _mm_cmpgt_epi16(__m128i m1, __m128i m2)	m1 の 16 ビット値がすべて m2 の 16 ビット値より大きい場合は、結果の 16 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPGTW	__m64 _mm_cmpgt_pi16 (__m64 m1, __m64 m2)	m1 の 16 ビット値がすべて m2 の 16 ビット値より大きい場合は、結果の 16 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PCMPGTD	__m128i _mm_cmpgt_epi32(__m128i m1, __m128i m2)	m1 の 32 ビット値がすべて m2 の 32 ビット値より大きい場合は、結果の 32 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PCMPGTD	__m64 _mm_cmpgt_pi32(__m64 m1, __m64 m2)	m1 の 32 ビット値がすべて m2 の 32 ビット値より大きい場合は、結果の 32 ビット値をすべて 1 にセットする。それ以外の場合は、すべて 0 にセットする。
PEXTRW	int _mm_extract_epi16(__m128i a, int n)	a の 8 ワードのうち 1 つを抽出する。セレクタ n は即値でなければならない。
PEXTRW	int _mm_extract_pi16(__m64 a, int n)	a の 4 ワードのうち 1 つを抽出する。セレクタ n は即値でなければならない。
PINSRW	__m128i _mm_insert_epi16(__m128i a, int d, int n)	a の 8 ワードのうち 1 つにワード d を挿入する。セレクタ n は即値でなければならない。
PINSRW	__m64 _mm_insert_pi16(__m64 a, int d, int n)	a の 4 ワードのうち 1 つにワード d を挿入する。セレクタ n は即値でなければならない。
PMADDWD	__m128i _mm_madd_epi16(__m128i m1, __m128i m2)	m1 の 8 つの 16 ビット値に m2 の 8 つの 16 ビット値を掛けて、8 つの 32 ビットの中間結果を求める。次に、これらの値を 2 つずつ合計して、4 つの 32 ビットの結果を求める。
PMADDWD	__m64 _mm_madd_pi16(__m64 m1, __m64 m2)	m1 の 4 つの 16 ビット値に m2 の 4 つの 16 ビット値を掛けて、4 つの 32 ビットの中間結果を求める。次に、これらの値を 2 つずつ合計して、2 つの 32 ビットの結果を求める。
PMAXSW	__m128i _mm_max_epi16(__m128i a, __m128i b)	a と b の 16 ビット整数を比較して、要素ごとに最大値を求める。
PMAXSW	__m64 _mm_max_pi16(__m64 a, __m64 b)	a と b のワードを比較して、要素ごとに最大値を求める。
PMAXUB	__m128i _mm_max_epu8(__m128i a, __m128i b)	a と b の符号なしバイトを比較して、要素ごとに最大値を求める。
PMAXUB	__m64 _mm_max_pu8(__m64 a, __m64 b)	a と b の符号なしバイトを比較して、要素ごとに最大値を求める。
PMINSW	__m128i _mm_min_epi16(__m128i a, __m128i b)	a と b の 16 ビット整数を比較して、要素ごとに最小値を求める。
PMINSW	__m64 _mm_min_pi16(__m64 a, __m64 b)	a と b のワードを比較して、要素ごとに最小値を求める。
PMINUB	__m128i _mm_min_epu8(__m128i a, __m128i b)	a と b の符号なしバイトを比較して、要素ごとに最小値を求める。
PMINUB	__m64 _mm_min_pu8(__m64 a, __m64 b)	a と b の符号なしバイトを比較して、要素ごとに最小値を求める。
PMOVMSKB	int _mm_movemask_epi8(__m128i a)	a の各バイトの最上位ビットから 16 ビット・マスクを作成する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PMOVMSKB	int _mm_movemask_pi8(__m64 a)	a の各バイトの最上位ビットから 8 ビット・マスクを作成する。
PMULHUW	__m128i _mm_mulhi_epu16(__m128i a, __m128i b)	a の 8 つの符号なしワードに b の 8 つの符号なしワードを掛けて、パック ド形式の 8 つの 32 ビットの中間結果の上位 16 ビットを返す。
PMULHUW	__m64 _mm_mulhi_pu16(__m64 a, __m64 b)	a の 4 つの符号なしワードに b の 4 つの符号なしワードを掛けて、パック ド形式の 4 つの 32 ビットの中間結果の上位 16 ビットを返す。
PMULHW	__m128i _mm_mulhi_epi16(__m128i m1, __m128i m2)	m1 の 8 つの符号付き 16 ビット値に m2 の 8 つの符号付き 16 ビット値を掛けて、8 つの結果の上位 16 ビットを返す。
PMULHW	__m64 _mm_mulhi_pi16(__m64 m1, __m64 m2)	m1 の 4 つの符号付き 16 ビット値に m2 の 4 つの符号付き 16 ビット値を掛けて、4 つの結果の上位 16 ビットを返す。
PMULLW	__m128i _mm_mullo_epi16(__m128i m1, __m128i m2)	m1 の 8 つの符号付き 16 ビット値に m2 の 8 つの符号付き 16 ビット値を掛けて、8 つの結果の下位 16 ビットを返す。
PMULLW	__m64 _mm_mullo_pi16(__m64 m1, __m64 m2)	m1 の 4 つの符号付き 16 ビット値に m2 の 4 つの符号付き 16 ビット値を掛けて、4 つの結果の下位 16 ビットを返す。
PMULUDQ	__m64 _mm_mul_su32(__m64 m1, __m64 m2) __m128i _mm_mul_epu32(__m128i m1, __m128i m2)	m1 の下位の符号なし 32 ビット値に m2 の下位の符号なし 32 ビット値を掛けて、64 ビットの結果をストアする。 m1 の下位の 2 つの符号なし 32 ビット値に m2 の下位の 2 つの符号なし 32 ビット値を掛けて、2 つの 64 ビットの結果をストアする。
POR	__m64 _mm_or_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値と m2 の 64 ビット値の間でビット単位の OR 演算を実行する。
POR	__m128i _mm_or_si128(__m128i m1, __m128i m2)	m1 の 128 ビット値と m2 の 128 ビット値の間でビット単位の OR 演算を実行する。
PREFETCHh	void _mm_prefetch(char *a, int sel)	1 キャッシュ・ラインのデータを、アドレス p からプロセッサに「より近い」位置にロードする。値 sel は、プリフェッチ操作のタイプを指定する。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PSADBW	__m128i _mm_sad_epu8(__m128i a, __m128i b)	a と b の 16 個の符号なし 8 ビット値の差の絶対値を計算して、上位の 8 つの差と下位の 8 つの差をそれぞれ合計し、2 つの 16 ビットの結果を上位 64 ビットと下位 64 ビットに格納する。
PSADBW	__m64 _mm_sad_pu8(__m64 a, __m64 b)	a と b の 8 つの符号なし 8 ビット値の差の絶対値を計算して、8 つの差を合計し、16 ビットの結果を格納する。上位 3 ワードはクリアされる。
PSHUFD	__m128i _mm_shuffle_epi32(__m128i a, int n)	a の 4 つの dword の組み合わせを返す。セレクタ n は即値でなければならない。
PSHUFHW	__m128i _mm_shufflehi_epi16(__m128i a, int n)	n で指定された、a の上位 4 つの 16 ビット・ワードをシャッフルする。セレクタ n は即値でなければならない。
PSHUFLW	__m128i _mm_shufflelo_epi16(__m128i a, int n)	n で指定された、a の下位 4 つの 16 ビット・ワードをシャッフルする。セレクタ n は即値でなければならない。
PSHUFW	__m64 _mm_shuffle_pi16(__m64 a, int n)	a の 4 ワードの組み合わせを返す。セレクタ n は即値でなければならない。
PSLLW	__m128i _mm_sll_epi16(__m128i m, __m128i count)	m の 8 つの各 16 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。
PSLLW	__m128i _mm_slli_epi16(__m128i m, int count)	m の 8 つの各 16 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。
PSLLW	__m64 _mm_sll_pi16(__m64 m, __m64 count) __m64 _mm_slli_pi16(__m64 m, int count)	m の 4 つの 16 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。 m の 4 つの 16 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSLLD	__m128i _mm_slli_epi32(__m128i m, int count) __m128i _mm_sll_epi32(__m128i m, __m128i count)	m の 4 つの各 32 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。 m の 4 つの各 32 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PSLLD	__m64 _mm_slli_pi32(__m64 m, int count)	m の 2 つの 32 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。
	__m64 _mm_sll_pi32(__m64 m, __m64 count)	m の 2 つの 32 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSLLQ	__m64 _mm_sll_si64(__m64 m, __m64 count)	m の 64 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。
	__m64 _mm_slli_si64(__m64 m, int count)	m の 64 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSLLQ	__m128i _mm_sll_epi64(__m128i m, __m128i count)	m の 2 つの各 64 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。
	__m128i _mm_slli_epi64(__m128i m, int count)	m の 2 つの各 64 ビット値を、count で指定された量だけ左にシフトし、下位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSLLDQ	__m128i _mm_slli_si128(__m128i m, int imm)	m の 128 ビット値を、imm バイトで左にシフトし、下位はゼロで埋める。
PSRAW	__m128i _mm_sra_epi16(__m128i m, __m128i count)	m の 8 つの各 16 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。
	__m128i _mm_srai_epi16(__m128i m, int count)	m の 8 つの各 16 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRAW	__m64 _mm_sra_pi16(__m64 m, __m64 count)	m の 4 つの 16 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。
	__m64 _mm_srai_pi16(__m64 m, int count)	m の 4 つの 16 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PSRAD	__m128i _mm_sra_epi32 (__m128i m, __m128i count)	m の 4 つの各 32 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。
	__m128i _mm_srai_epi32 (__m128i m, int count)	m の 4 つの各 32 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRAD	__m64 _mm_sra_pi32 (__m64 m, __m64 count)	m の 2 つの 32 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。
	__m64 _mm_srai_pi32 (__m64 m, int count)	m の 2 つの 32 ビット値を、count で指定された量だけ右にシフトし、上位は符号ビットで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRLW	_m128i _mm_srl_epi16 (__m128i m, __m128i count)	m の 8 つの各 16 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m128i _mm_srli_epi16 (__m128i m, int count)	m の 8 つの各 16 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
PSRLW	__m64 _mm_srl_pi16 (__m64 m, __m64 count)	m の 4 つの 16 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m64 _mm_srli_pi16(__m64 m, int count)	m の 4 つの 16 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRLD	__m128i _mm_srl_epi32 (__m128i m, __m128i count)	m の 4 つの各 32 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m128i _mm_srli_epi32 (__m128i m, int count)	m の 4 つの各 32 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRLD	__m64 _mm_srl_pi32 (__m64 m, __m64 count)	m の 2 つの 32 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m64 _mm_srli_pi32 (__m64 m, int count)	m の 2 つの 32 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PSRLQ	__m128i _mm_srl_epi64 (__m128i m, __m128i count)	m の 2 つの各 64 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m128i _mm_srli_epi64 (__m128i m, int count)	m の 2 つの各 64 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRLQ	__m64 _mm_srl_si64 (__m64 m, __m64 count)	m の 64 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。
	__m64 _mm_srli_si64 (__m64 m, int count)	m の 64 ビット値を、count で指定された量だけ右にシフトし、上位はゼロで埋める。最高のパフォーマンスを得るためには、count は定数でなければならない。
PSRLDQ	__m128i _mm_srli_si128(__m128i m, int imm)	m の 128 ビット値を、imm バイトで右にシフトし、下位はゼロで埋める。
PSUBB	__m128i _mm_sub_epi8(__m128i m1, __m128i m2)	m1 の 16 個の 8 ビット値から m2 の 16 個の 8 ビット値を引く。
PSUBB	__m64 _mm_sub_pi8(__m64 m1, __m64 m2)	m1 の 8 つの 8 ビット値から m2 の 8 つの 8 ビット値を引く。
PSUBW	__m128i _mm_sub_epi16(__m128i m1, __m128i m2)	m1 の 8 つの 16 ビット値から m2 の 8 つの 16 ビット値を引く。
PSUBW	__m64 _mm_sub_pi16(__m64 m1, __m64 m2)	m1 の 4 つの 16 ビット値から m2 の 4 つの 16 ビット値を引く。
PSUBD	__m128i _mm_sub_epi32(__m128i m1, __m128i m2)	m1 の 4 つの 32 ビット値から m2 の 4 つの 32 ビット値を引く。
PSUBD	__m64 _mm_sub_pi32(__m64 m1, __m64 m2)	m1 の 2 つの 32 ビット値から m2 の 2 つの 32 ビット値を引く。
PSUBQ	__m128i _mm_sub_epi64(__m128i m1, __m128i m2)	m1 の 2 つの 64 ビット値から m2 の 2 つの 64 ビット値を引く。
PSUBQ	__m64 _mm_sub_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値から m2 の 64 ビット値を引く。
PSUBSB	__m128i _mm_subs_epi8(__m128i m1, __m128i m2)	m1 の 16 個の符号付き 8 ビット値から m2 の 16 個の符号付き 8 ビット値を引き、飽和させる。
PSUBSB	__m64 _mm_subs_pi8(__m64 m1, __m64 m2)	m1 の 8 つの符号付き 8 ビット値から m2 の 8 つの符号付き 8 ビット値を引き、飽和させる。
PSUBSW	__m128i _mm_subs_epi16(__m128i m1, __m128i m2)	m1 の 8 つの符号付き 16 ビット値から m2 の 8 つの符号付き 16 ビット値を引き、飽和させる。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PSUBSW	__m64 _mm_subs_pi16(__m64 m1, __m64 m2)	m1 の 4 つの符号付き 16 ビット値から m2 の 4 つの符号付き 16 ビット値を引き、飽和させる。
PSUBUSB	__m128i _mm_sub_epu8(__m128i m1, __m128i m2)	m1 の 16 個の符号なし 8 ビット値から m2 の 16 個の符号なし 8 ビット値を引き、飽和させる。
PSUBUSB	__m64 _mm_sub_pu8(__m64 m1, __m64 m2)	m1 の 8 つの符号なし 8 ビット値から m2 の 8 つの符号なし 8 ビット値を引き、飽和させる。
PSUBUSW	__m128i _mm_sub_epu16(__m128i m1, __m128i m2)	m1 の 8 つの符号なし 16 ビット値から m2 の 8 つの符号なし 16 ビット値を引き、飽和させる。
PSUBUSW	__m64 _mm_sub_pu16(__m64 m1, __m64 m2)	m1 の 4 つの符号なし 16 ビット値から m2 の 4 つの符号なし 16 ビット値を引き、飽和させる。
PUNPCKHBW	__m64 _mm_unpackhi_pi8(__m64 m1, __m64 m2)	m1 の上位半分の 4 つの 8 ビット値と m2 の上位半分の 4 つの 8 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKHBW	__m128i _mm_unpackhi_epi8(__m128i m1, __m128i m2)	m1 の上位半分の 8 つの 8 ビット値と m2 の上位半分の 8 つの 8 ビット値をインタリーブする。
PUNPCKHWD	__m64 _mm_unpackhi_pi16(__m64 m1,__m64 m2)	m1 の上位半分の 2 つの 16 ビット値と m2 の上位半分の 2 つの 16 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKHWD	__m128i _mm_unpackhi_epi16(__m128i m1, __m128i m2)	m1 の上位半分の 4 つの 16 ビット値と m2 の上位半分の 4 つの 16 ビット値をインタリーブする。
PUNPCKHDQ	___m64 _mm_unpackhi_pi32(__m64 m1, __m64 m2)	m1 の上位半分の 32 ビット値と m2 の上位半分の 32 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKHDQ	__m128i _mm_unpackhi_epi32(__m128i m1, __m128i m2)	m1 の上位半分の 2 つの 32 ビット値と m2 の上位半分の 2 つの 32 ビット値をインタリーブする。
PUNPCKHQDQ	__m128i _mm_unpackhi_epi64(__m128i m1, __m128i m2)	m1 の上位半分の 64 ビット値と m2 の上位半分の 64 ビット値をインタリーブする。
PUNPCKLBW	__m64 _mm_unpacklo_pi8 (__m64 m1, __m64 m2)	m1 の下位半分の 4 つの 8 ビット値と m2 の下位半分の 4 つの 8 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKLBW	__m128i _mm_unpacklo_epi8 (__m128i m1, __m128i m2)	m1 の下位半分の 8 つの 8 ビット値と m2 の下位半分の 8 つの 8 ビット値をインタリーブする。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
PUNPCKLWD	__m64 _mm_unpacklo_pi16(__m64 m1, __m64 m2)	m1 の下位半分の 2 つの 16 ビット値と m2 の下位半分の 2 つの 16 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKLWD	__m128i _mm_unpacklo_epi16(__m128i m1, __m128i m2)	m1 の下位半分の 4 つの 16 ビット値と m2 の下位半分の 4 つの 16 ビット値をインタリーブする。
PUNPCKLDQ	__m64 _mm_unpacklo_pi32(__m64 m1, __m64 m2)	m1 の下位半分の 32 ビット値と m2 の下位半分の 32 ビット値をインタリーブし、m1 の最下位の要素をとる。
PUNPCKLDQ	__m128i _mm_unpacklo_epi32(__m128i m1, __m128i m2)	m1 の下位半分の 2 つの 32 ビット値と m2 の下位半分の 2 つの 32 ビット値をインタリーブする。
PUNPCKLQDQ	__m128i _mm_unpacklo_epi64(__m128i m1, __m128i m2)	m1 の下位半分の 64 ビット値と m2 の下位半分の 64 ビット値をインタリーブする。
PXOR	__m64 _mm_xor_si64(__m64 m1, __m64 m2)	m1 の 64 ビット値と m2 の 64 ビット値の間でビット単位の XOR 演算を実行する。
PXOR	__m128i _mm_xor_si128(__m128i m1, __m128i m2)	m1 の 128 ビット値と m2 の 128 ビット値の間でビット単位の XOR 演算を実行する。
RCPPS	__m128 _mm_rcp_ps(__m128 a)	a の 4 つの単精度浮動小数点値の逆数の近似値を計算する。
RCPSS	__m128 _mm_rcp_ss(__m128 a)	a の最下位の単精度浮動小数点値の逆数の近似値を計算する。上位の 3 つの単精度浮動小数点値はそのまま渡される。
RSQRTPS	__m128 _mm_rsqrt_ps(__m128 a)	a の 4 つの単精度浮動小数点値の平方根の逆数の近似値を計算する。
RSQRTSS	__m128 _mm_rsqrt_ss(__m128 a)	a の最下位の単精度浮動小数点値の平方根の逆数の近似値を計算する。上位の 3 つの単精度浮動小数点値はそのまま渡される。
SFENCE	void_mm_sfence(void)	フェンスより前にあるすべてのストア命令が、フェンスより後にある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。
SHUFPD	__m128d _mm_shuffle_pd(__m128d a, __m128d b, unsigned int imm8)	マスク imm8 に基づいて、a と b から 2 つの倍精度浮動小数点値を選択する。マスクは即値でなければならない。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
SHUFPS	__m128 _mm_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)	マスク imm8 に基づいて、a と b から 4 つの単精度浮動小数点値を選択する。マスクは即値でなければならない。
SQRTPD	__m128d _mm_sqrt_pd(__m128d a)	a の 2 つの倍精度浮動小数点値の平方根を計算する。
SQRTPS	__m128 _mm_sqrt_ps(__m128 a)	a の 4 つの単精度浮動小数点値の平方根を計算する。
SQRTSD	__m128d _mm_sqrt_sd(__m128d a)	a の最下位の倍精度浮動小数点値の平方根を計算する。上位の倍精度浮動小数点値はそのまま渡される。
SQRTSS	__m128 _mm_sqrt_ss(__m128 a)	a の最下位の単精度浮動小数点値の平方根を計算する。上位の 3 つの単精度浮動小数点値はそのまま渡される。
STMXCSR	_mm_getcsr(void)	制御レジスタの内容を返す。
SUBPD	__m128d _mm_sub_pd(__m128d a, __m128d b)	a の 2 つの倍精度浮動小数点値から b の 2 つの倍精度浮動小数点値を引く。
SUBPS	__m128 _mm_sub_ps(__m128 a, __m128 b)	a の 4 つの単精度浮動小数点値から b の 4 つの単精度浮動小数点値を引く。
SUBSD	__m128d _mm_sub_sd(__m128d a, __m128d b)	a の最下位の倍精度浮動小数点値から b の最下位の倍精度浮動小数点値を引く。上位の倍精度浮動小数点値は、a からそのまま渡される。
SUBSS	__m128 _mm_sub_ss(__m128 a, __m128 b)	a の最下位の単精度浮動小数点値から b の最下位の単精度浮動小数点値を引く。上位の 3 つの単精度浮動小数点値は、a からそのまま渡される。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
UCOMISD	int _mm_ucomieq_sd(__m128d a, __m128d b)	「a と b が等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a と b が等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomilt_sd(__m128d a, __m128d b)	「a が b より小さい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より小さい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomile_sd(__m128d a, __m128d b)	「a が b より小さいか等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より小さいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomigt_sd(__m128d a, __m128d b)	「a が b より大きい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より大きい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomige_sd(__m128d a, __m128d b)	「a が b より大きいか等しい」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a が b より大きいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomineq_sd(__m128d a, __m128d b)	「a と b が等しくない」という条件で、a と b の最下位の倍精度浮動小数点値を比較する。a と b が等しくない場合は、1 が返される。それ以外の場合は、0 が返される。

表 C-1. 簡単な組み込み関数（続き）

ニーモニック	組み込み関数	説明
UCOMISS	int _mm_ucomieq_ss(__m128 a, __m128 b)	「a と b が等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a と b が等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomilt_ss(__m128 a, __m128 b)	「a が b より小さい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より小さい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomile_ss(__m128 a, __m128 b)	「a が b より小さいか等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より小さいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomigt_ss(__m128 a, __m128 b)	「a が b より大きい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より大きい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomige_ss(__m128 a, __m128 b)	「a が b より大きいか等しい」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a が b より大きいか等しい場合は、1 が返される。それ以外の場合は、0 が返される。
	int _mm_ucomineq_ss(__m128 a, __m128 b)	「a と b が等しくない」という条件で、a と b の最下位の単精度浮動小数点値を比較する。a と b が等しくない場合は、1 が返される。それ以外の場合は、0 が返される。
UNPCKHPD	__m128d _mm_unpackhi_pd(__m128d a, __m128d b)	a と b の上位の倍精度浮動小数点値を選択し、インタリーブする。
UNPCKHPS	__m128 _mm_unpackhi_ps(__m128 a, __m128 b)	a と b の上位の 2 つの単精度浮動小数点値を選択し、インタリーブする。
UNPCKLPD	__m128d _mm_unpacklo_pd(__m128d a, __m128d b)	a と b の下位の倍精度浮動小数点値を選択し、インタリーブする。
UNPCKLPS	__m128 _mm_unpacklo_ps(__m128 a, __m128 b)	a と b の下位の 2 つの単精度浮動小数点値を選択し、インタリーブする。
XORPD	__m128d _mm_xor_pd(__m128d a, __m128d b)	a と b の 2 つの倍精度浮動小数点値の間でビット単位の EXOR（排他的論理和）演算を実行する。
XORPS	__m128 _mm_xor_ps(__m128 a, __m128 b)	a と b の 4 つの単精度浮動小数点値の間でビット単位の EXOR（排他的論理和）演算を実行する。

C.2. 複合組み込み関数

表 C-2. 複合組み込み関数

ニーモニック	組み込み関数	説明
(composite)	__m128i _mm_set_epi64(__m64 q1, __m64 q0)	2 つの 64 ビット値を 2 つの入力値に設定する。
(composite)	__m128i _mm_set_epi32(int i3, int i2, int i1, int i0)	4 つの 32 ビット値を 4 つの入力値に設定する。
(composite)	__m128i _mm_set_epi16(short w7,short w6,short w5, short w4, short w3, short w2,short w1,short w0)	8 つの 16 ビット値を 8 つの入力値に設定する。
(composite)	__m128i _mm_set_epi8(char w15,char w14,char w13, char w12, char w11, char w10,char w9,char w8,char w7,char w6,char w5, char w4, char w3, char w2,char w1,char w0)	16 個の 8 ビット値を 16 個の入力値に設定する。
(composite)	__m128i _mm_set1_epi64(__m64 q)	2 つの 64 ビット値を入力値に設定する。
(composite)	__m128i _mm_set1_epi32(int a)	4 つの 32 ビット値を入力値に設定する。
(composite)	__m128i _mm_set1_epi16(short a)	8 つの 16 ビット値を入力値に設定する。
(composite)	__m128i _mm_set1_epi8(char a)	16 個の 8 ビット値を入力値に設定する。
(composite)	__m128i _mm_setr_epi64(__m64 q1, __m64 q0)	2 つの 64 ビット値を、逆の順番で 2 つの入力値に設定する。
(composite)	__m128i _mm_setr_epi32(int i3, int i2, int i1, int i0)	4 つの 32 ビット値を、逆の順番で 4 つの入力値に設定する。
(composite)	__m128i _mm_setr_epi16(short w7,short w6,short w5, short w4, short w3, short w2,short w1,short w0)	8 つの 16 ビット値を、逆の順番で 8 つの入力値に設定する。
(composite)	__m128i _mm_setr_epi8(char w15,char w14,char w13, char w12, char w11, char w10,char w9,char w8,char w7,char w6,char w5, char w4, char w3, char w2,char w1,char w0)	16 個の 8 ビット値を、逆の順番で 16 個の入力値に設定する。
(composite)	__m128i _mm_setzero_si128()	すべてのビットを 0 に設定する。
(composite)	__m128 _mm_set_ps1(float w) __m128 _mm_set1_ps(float w)	4 つの単精度浮動小数点値を w に設定する。
(composite)	__m128d _mm_set1_pd(double w)	2 つの倍精度浮動小数点値を w に設定する。
(composite)	__m128d _mm_set_sd(double w)	下位の倍精度浮動小数点値を w に設定する。
(composite)	__m128d _mm_set_pd(double z, double y)	2 つの倍精度浮動小数点値を 2 つの入力値に設定する。
(composite)	__m128 _mm_set_ps(float z, float y, float x, float w)	4 つの単精度浮動小数点値を 4 つの入力値に設定する。
(composite)	__m128d _mm_setr_pd(double z, double y)	2 つの倍精度浮動小数点値を、逆の順番で 2 つの入力値に設定する。

表 C-2. 複合組み込み関数（続き）

ニーモニック	組み込み関数	説明
(composite)	__m128 _mm_setr_ps(float z, float y, float x, float w)	4 つの単精度浮動小数点値を、逆の順番で 4 つの入力値に設定する。
(composite)	__m128d _mm_setzero_pd(void)	2 つの倍精度浮動小数点値をクリアする。
(composite)	__m128 _mm_setzero_ps(void)	4 つの単精度浮動小数点値をクリアする。
MOVSD + shuffle	__m128d _mm_load_pd(double * p) __m128d _mm_load1_pd(double *p)	1 つの倍精度浮動小数点値をロードし、両方の倍精度浮動小数点値にコピーする。
MOVSS + shuffle	__m128 _mm_load_ps1(float * p) __m128 _mm_load1_ps(float *p)	1 つの単精度浮動小数点値をロードし、4 ワードすべてにコピーする。
MOVAPD + shuffle	__m128d _mm_loadr_pd(double * p)	2 つの倍精度浮動小数点値を逆の順番でロードする。アドレスは 16 バイト・アライメントでなければならない。
MOVAPS + shuffle	__m128 _mm_loadr_ps(float * p)	4 つの単精度浮動小数点値を逆の順番でロードする。アドレスは 16 バイト・アライメントでなければならない。
MOVSD + shuffle	void _mm_store1_pd(double *p, __m128d a)	最下位の倍精度浮動小数点値を両方の倍精度浮動小数点値にストアする。
MOVSS + shuffle	void _mm_store_ps1(float * p, __m128 a) void _mm_store1_ps(float *p, __m128 a)	最下位の単精度浮動小数点値を 4 ワードにストアする。
MOVAPD + shuffle	_mm_storer_pd(double * p, __m128d a)	2 つの倍精度浮動小数点値を逆の順番でストアする。アドレスは 16 バイト・アライメントでなければならない。
MOVAPS + shuffle	_mm_storer_ps(float * p, __m128 a)	4 つの単精度浮動小数点値を逆の順番でストアする。アドレスは 16 バイト・アライメントでなければならない。

索 引

索 引

記号・数字

A

B

C

D

E

F

G

H

I

J

L

M

S

T

U

V

W

X