C++ SIMD 命令 クラス・ライブラリ

リファレンス・マニュアル

資料番号：693500J-402

C++ SIMD 命令
クラス・ライブラリ
リファレンス・マニュアル

資料番号 : 693500J-402

改訂番号	改訂履歴	日付
-402	VTune™ **パフォーマンス拡張環境** 4.0 **のリリース**	99 **年** 2 **月**

【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。

本資料の内容についてのお問い合わせは、下記までご連絡下さい。

インテル株式会社　資料販売センタ

〒 305-8603　筑波学園郵便局　私書箱 115 号

Fax：0120-478832

* 一般にブランド名または商品名は各社の商標または登録商標です。

目次

第 3 章　Ivec クラス

第 4 章　FVec クラス

図

表

はじめに 1

C++ SIMD 命令 クラス・ライブラリには、インテル® アーキテクチャに対する SIMD(Single-Instruction, Multiple-Data) 拡張命令（MMX® テクノロジおよびストリーミング SIMD 拡張命令など) をより簡単かつ効率的に使用可能な C++ ベクタ・クラスが含まれています。これらのクラスを使用すれば、アセンブリ言語を手作業でコーディングしたり、Intel C/C++ コンパイラに付属の組み込み関数命令セットを使用する必要はありません。

C++ クラス・ライブラリはインクルード・ファイルであり、Intel C/C++ コンパイラと一緒にインストールされます。C++ クラス・ライブラリの Ivec クラスは MMX テクノロジ命令を、Fvec クラスはストリーミング SIMD 拡張命令をそれぞれ C++ でカプセル化したものです。

ハードウェアとソフトウェアの必要条件

C++ SIMD 命令 クラス・ライブラリは、Intel アーキテクチャ・ベースのコンピュータ上で動作するようにコンパイルされるコードで使用することができます。ライブラリは Intel C/C++ コンパイラに付属するヘッダ・ファイルの中に用意されているので、適切なヘッダ・ファイルを使用するためには、このコンパイラがインストールされている必要があります。

C++ クラス・ライブラリは、Microsoft Windows* および Windows NT* を実行しているシステム上でテストされています。Ivec クラスを使用するためには、ターゲット・システムに MMX テクノロジを搭載した Intel アーキテクチャのプロセッサが搭載されている必要があります。Fvec クラスを使用したコードを動作させるには、Pentium® III プロセッサ搭載のシステムが必要となります。

本書について

本書は、C++ SIMD 命令 クラス・ライブラリについて、各クラスの構文を含めて説明しています。

本書の構成

本書は 4 つの章と 2 つの付録から構成されています。

第 1 章	「はじめに」Intel C++ SIMD 命令 クラス・ライブラリの概要を紹介し、マニュアルの構成を示し、表記の規則について説明します。
第 2 章	「技術概要」SIMD クラスの利点、使い方、および機能について説明します。
第 3 章	「IVec クラス」整数クラスのセット、すなわち "Ivec" クラスについて説明します。
第 4 章	「Fvec クラス」浮動小数点クラスのセット、すなわち "Fvec" クラスについて説明します。
付録 A	「クイック・リファレンス」使用可能なクラスを一覧表にしてあります。クラス内に含まれていない C/C++ コンパイラの組み込み関数の表もあります。
付録 B	「プログラミングの例」Fvec クラスを使用したプログラムの例を紹介します。

対象読者

本書は、Intel アーキテクチャ向けのコード、特に SIMD 命令を意識したコードを書くことが求められているプログラマを対象としています。C++ および C++ クラスの使い方に関する広範な知識を必要とします。

参考文献

Intel C/C++ コンパイラとその他の Intel パフォーマンス・チューニング・ツールに関する最新情報 (プロダクト・アップデート、テクニカル・サポートを含む) については、当社のウェブ・サイト、http://www.intel.co.jp/jp/developer/design/perftool を参照してください。

以下に参考となるドキュメントを示します。

- 『Intel C/C++ コンパイラ ユーザーズ・ガイド』、資料番号 718195J
- 『The Annotated C++ Reference Manual』初版、Margaret Ellis、Bjarne Stroustrup 共著、Addison Wesley 刊 (1991 年)。C++ プログラミング言語について解説しています。

以下に、Intel アーキテクチャに関する情報を含んでいるドキュメントを示します。これらのドキュメントは、インテル・アーキテクチャ・パフォーマンス・トレーニング・センタまたはインテルのウェブ・サイト (http://www.intel.co.jp) で入手できます。

- 『インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル , 上巻 : 基本アーキテクャ』、資料番号 243190J
- 『インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル , 中巻 : 命令セット・リファレンス』、資料番号 243191J
- 『インテル・アーキテクチャ 最適化マニュアル』、資料番号 730795J

表記規則

本書の表記は、以下の規則に従います。

`This type style`	構文の要素、予約語、キーワード、ファイル名、コンピュータ出力、プログラム例の一部分のいずれかを表します。テキストは、大文字に意味がない限り小文字で表記します。 `l` は小文字の L、`1` は数字の 1 です。また、`O` は大文字の O、`0` は数字の 0 です。
`This type style`	入力としてユーザがタイプする文字そのものを表します。
`This type style`	識別子、式、文字列、記号、値のいずれかのプレースホルダを表します。これらの項目のいずれかをプレースホルダと置き換えてください。
`[`*`items`*`]`	角括弧で囲まれた項目はオプションです。
`{`*`item`*` | `*`item`*`}`	中括弧内に列挙された項目から 1 つだけを選択します。縦線 (|) は項目の区切りです。
`...` (ellipses)	直前の項目を複数指定できることを表します。

技術概要 2

この章では、使用可能な C++ クラスを詳しく説明します。Ivec クラスは MMX® テクノロジ命令とストリーミング SIMD 拡張命令の整数命令を C++ でカプセル化したものです。Fvec クラスはストリーミング SIMD 拡張命令の浮動小数点命令を C++ でカプセル化したものです。

SIMD クラスを使用する理由

C++ SIMD 命令クラス・ライブラリは使いやすいインターフェイスを提供しており、プログラマはアセンブリ言語を手作業で書かなくても、MMX 命令とストリーミング SIMD 拡張命令の機能を使用することができます。

これらのクラスを使うとプログラミングが簡単に行えます。では、4 個の 16 ビット符号付き整数値の和を求めるものとし、これをインライン・アセンブリ、コンパイラの組み込み関数、そして C++ クラスを使用したコードを比較してみましょう。

アセンブリ言語：

```
__m64 a,b,c;
  __asm {
    movq mm0,b
    movq mm1,c
    paddw mm0,mm1 /* Adds the 4 16-bit signed integer values */
                  /* of mm0 and mm1 */
    movq a, mm0
  }
```

組み込み関数：

```
__m64 a,b,c;
       a = _m_paddw(b,c) /* Adds 4 signed integer values of b and c */
```

C++:

```
Is16vec4 a,b,c;
     a = b + c    /* Adds 4 signed integer values of b and c */
```

インライン・アセンブリを使用する場合、コンパイラはアセンブリ・ルーチンの前にレジスタをメモリに保存し、アセンブリ・ルーチンの後でレジスタをリロードする必要があります。これでは命令数とメモリのアクセス回数が増える結果になります。C++ クラスまたはコンパイラの組み込み関数を使用すると、このような非効率的なコードを生成する必要はありません。

SIMD クラス

SIMD クラスは、並列での演算が可能な配列、すなわちデータのベクトルというコンセプトに基づいて作られています。たとえば、それぞれ 4 つの要素を含んでいる `A` と `B` という 2 つのベクトルの加算を考えてみましょう。クラスの 1 つ、"`Fvec`" を使用した場合、各配列の対応する要素である `A[i]` と `B[i]` の加算は次のように行われます。

```
short a[4], b[4], c[4];
  for (i=0; i<4; i++)
    c[i] = a[i] + b[i];
```

繰り返しごとの配列要素のアクセスの間に依存関係が存在しないため、個別の加算を同時に実行することができます。Ivec クラスを使用して、各配列の対応する要素 `A[i]` と `B[i]` を次のように加算します。

```
Is16vec4 ivecA, ivecB, ivecC;
  ivecC = ivecA + ivecB;
```

1 つの命令で 4 つの加算を実行できるため、必要な演算の回数は 1/4 に減ります。

このような並列処理は、C++ の一般的な機構では簡単には実現できません。これを可能にするのが SIMD クラスです。C++ SIMD 命令クラス・ライブラリとは、MMX テクノロジ命令、ストリーミング SIMD 拡張命令、およびそれらに対応するデータ・タイプの総称です。

クラス名の規則

クラス名の一般的な形式を次に示します。

<type><signedness><bits> <vec><elements>=

[F|I] [s|u] [64|32|16|8] vec [8|4|2|1]

各項目の意味は次のとおりです。

type	浮動小数点 (F) または整数 (I) を示します。
signedness	符号あり (S) または符号なし (U) を示します。Ivec クラスではこのフィールドが空ならば、中間クラスであることを示します。
bits	要素 1 つ当たりのビット数を指定します。
elements	要素の数を指定します。

例 :

Iu8vec8	符号なしの 8 ビットのパックド整数を 8 つ含むクラス
F32vec4	4 つのパックド単精度浮動小数点値を含むクラス

使用可能なクラス

表 2-1 に、使用可能なクラス、各クラスの一般的な特性、および各クラスが定義されるヘッダ・ファイルを示しています。

表 2-1 Intel SIMD 命令ベクトル・クラス

クラス	下位の命令セット	要素のデータ型	各要素のサイズ	要素の数	ヘッダ・ファイル
I64vec1	MMX テクノロジ	__m64	64	1	ivec.h
I32vec2	MMX テクノロジ	signed または unsigned int	32	2	ivec.h
Is32vec2	MMX テクノロジ	signed int	32	2	ivec.h
Iu32vec2	MMX テクノロジ	unsigned int	32	2	ivec.h
I16vec4	MMX テクノロジ	signed または unsigned short	16	4	ivec.h
Is16vec4	MMX テクノロジ	signed short	16	4	ivec.h
Iu16vec4	MMX テクノロジ	unsigned short	16	4	ivec.h

表 2-1 Intel SIMD 命令ベクトル・クラス (続き)

クラス	下位の命令セット	要素のデータ型	各要素のサイズ	要素の数	ヘッダ・ファイル
I8vec8	MMX テクノロジ	signed または unsigned char	8	8	ivec.h
Is8vec8	MMX テクノロジ	signed char	8	8	ivec.h
Iu8vec8	MMX テクノロジ	unsigned char	8	8	ivec.h
F32vec4	ストリーミング SIMD 拡張命令	float	32	4	fvec.h
F32vec1	ストリーミング SIMD 拡張命令	float	32	1	fvec.h

ほとんどのクラスは、すべてのデータ・タイプに対して同様の機能を持ち、すべての使用可能な組み込み関数を表現できます。ただし、機能の種類によっては、あるデータ型から別のデータ型に移すとパフォーマンスが低下することがあります。このため、個々のクラスでは一部の機能が省略されていることがあります。

注. *即値をとるために、クラスでは簡単に表現できない組み込み関数は、クラスではサポートされていません (*`mm_shuffle`、`_m_pshufw`、`_m_pextrw`、`_m_pinswr` *など)。クラスに含まれていない組み込み関数については、付録 A の表を参照してください。*

使用方法

この節では、SIMD C++ クラスにアクセスする方法と、使用上の注意事項について説明します。

クラスへのアクセス (ヘッダ・ファイル)

必要なクラス・ヘッダ・ファイルは、Intel® C/C++ コンパイラとともにインストールされる "include" ディレクトリに置かれています。

SIMD 整数命令用の Ivec クラスを使用するには、次のようにアプリケーションにヘッダ・ファイル ivec.h をインクルードします。

```
#include <ivec.h>
```

SIMD 浮動小数点用の Fvec クラスを使用するには、次のようにアプリケーションにヘッダ・ファイル fvec.h をインクルードします。

```
#include <fvec.h>
```

fvec.h には ivec.h の内容が含まれています。このため、Ivec クラスと Fvec クラスの両方を使用したい場合には、上記のように fvec.h を呼び出すだけで済みます。

注. *インテルの C/C++ コンパイラ組み込み関数と SIMD C++ クラスの両方を呼び出すプログラムの場合、組み込み関数のヘッダ・ファイルと C++ のヘッダ・ファイルの両方をインクルードする必要はありません。SIMD C++ クラス・ヘッダ・ファイルには、組み込み関数のヘッダ・ファイルが含まれています。ivec.h には、MMX テクノロジのコンパイラ組み込み関数のヘッダ・ファイル mmintrin.h が含まれています。fvec.h には、ストリーミング SIMD 拡張命令のコンパイラ組み込み関数のヘッダ・ファイル xmmintrin.h が含まれています。*

使用上の注意事項

C++ クラスを使用するには一般的なガイドラインがあり、これらに従ってコーディングしなければなりません各クラスの使用上の規則については、第 3 章「Ivec クラス」と第 4 章「Fvec クラス」を参照してください。

MMX® レジスタをクリアする

Ivec クラスと Fvec クラスの両方を同時に使用する場合、プログラムでは Ivec クラスから呼び出される MMX 命令と Fvec クラスから呼び出される Intel x87 アーキテクチャ浮動小数点命令が混在する可能性があります。浮動小数点命令は、次の Fvec 関数に含まれています。

- `fvec` のコンストラクタ
- デバッグ関数 (`cout` と要素アクセス)
- `rsqrt_nr`

MMX レジスタは浮動小数点レジスタにエイリアス化されているので、下記に示すように、x87 浮動小数点命令を発行する前に EMMS 命令組み込み関数で MMX の状態をクリアすることが重要です。

```
ivecA = ivecA & ivecB;    /* Ivec logical operation that uses */
                          /* MMX instructions */
_empty;                   /* clear state */
cout << f32vec4a;         /* F32vec4 operation that uses x87 */
                          /* floating-point instructions */
```

注意. *MMX レジスタのクリアを行わないと、レジスタの不正状態が原因で、不正な実行が行われたりパフォーマンスが低下したりすることがあります。*

EMMS 命令のガイドラインに従う

インテルは、『*Intel® C/C++ コンパイラ Win32* システム対応版ユーザーズ・ガイド*』に記されている EMMS 命令の使用ガイドラインに従うことを強くお勧めします。Fvec クラスと Ivec クラスでコーディングを開始する前に、同資料を参照してください。

機能

各 SIMD C++ クラスには、次の基本機能があります。

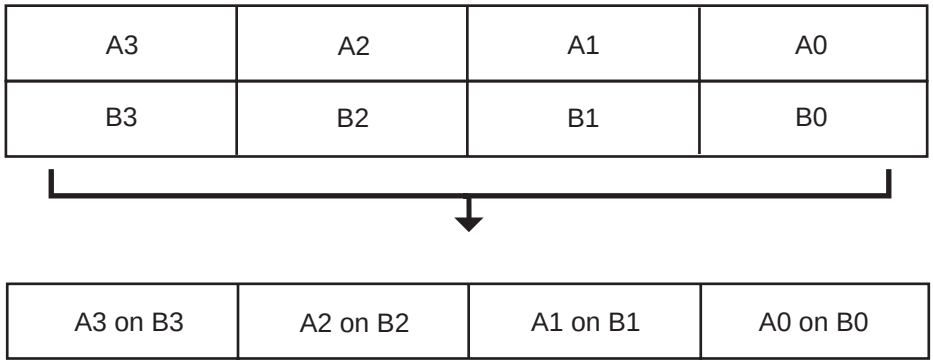
- 計算
- 分岐圧縮／削除
- キャッシング・ヒント
- 水平データ移動

目的の結果を得るためには、個々の能力とその相互作用を理解することが重要です。

計算

SIMD C++ クラスは、ほとんどの算術演算を「垂直」方向にサポートしています。「垂直」という言葉は、図 2-1 のように機能を表現したときのデータの流れの向きを示しています。Ivec クラスはさらにシフトと飽和をサポートしています。

図 2-1 垂直のデータの流れ



計算演算には、+、-、*、/、逆数 (`rcp` および `rcp_nr`)、平方根 (`sqrt`)、逆平方根 (`rsqrt` および `rsqrt_nr`) があります。演算 `rcp` および `rsqrt` は、12 ビット以上の精度ときわめて短い遅延を実現した新しい近似命令です。演算 `rcp_nr` と `rsqrt_nr` では、ソフトウェア改良テクニックを利用して、パフォーマンスに対する影響を最小限に抑えながら、精度を高めることができます。("nr" は、近似結果の精度を向上させる数学的手法、Newton-Raphson 法の略です)。

分岐の圧縮／削除

SIMD アーキテクチャでの分岐は複雑かつ高価であり、分岐先の予測が困難でコード・サイズが大きく膨れ上がる可能性があります。SIMD C++ クラスには、論理演算、max 関数と min 関数、条件付き選択、比較により、分岐を一括削除する機能があります。

下記のサンプル・コードを検証してみましょう。

```
float  a[4], b[4], c[4];
for (i=0; i<4; i++)
  c[i] = a[i] > b[i] ? a[i] : b[i];
```

この演算はインデックス値 `i` には依存しておらず、各 `i` について、結果は実際の値に応じて A または B になります。分岐を一括削除する単純な方法としては、MAX 命令を使って、演算を分岐なしで行う `select_gt` 関数を利用することが考えられます。次に例を示します。

```
Is16vec4 a, b, c
C = select_gt(a, b, a , b)
```

算術演算の水平サポート

SIMD C++ クラスでは、一部の算術演算については「水平」方向にもサポートしています。ここでいう「水平」とは、１つのベクトル (配列) の全要素にまたがる演算が可能であることを意味します。これは、前述の、垂直方向のサポート、つまり、異なる 2 つのベクトル間での要素対応の演算と対極をなすものです。

`add_horizontal`、`unpack_low`、および `pack sat` 関数は、「水平」演算の例です。この命令は、SIMD 命令の能力を十分に活用できない一部のアルゴリズムを可能にするために用意されています。

シャッフル組み込み関数も「水平」方向のデータ・フローの例して挙げることができます。シャッフル組み込み関数は即値を引数とするので、C++ クラスには表現されていませんが、C++ クラスではシャッフル組み込み関数などの組み込み関数を他の C++ 関数と混在させることができます。次に例を示します。

```
F32vec4 fveca, fvecb, fvecd;
fveca += fvecb;
fvecd = _mm_shuffle_ps(fveca,fvecb,0);
```

一般に、水平方向のデータ・フローを含むすべての命令は、その展開においてある程度の非効率的な部分を含まざるをえません。アルゴリズム導入の際には、できればこの種の水平演算は使わずに済ませたいものです。

キャッシング・ヒント

ストリーミング SIMD 拡張命令では、プリフェッチとストリーミングをアプリケーションレベルで考慮することができます。データをプリフェッチすると、メモリ遅延を最小限に抑えられます。ストリーミング・ヒントでは、ある種のデータはキャッシュに格納しない方がいいことがわかります。これにより、キャッシュに格納されるべきデータのパフォーマンスが向上します。

Ivec クラス

3

Ivec クラスは、MMX® テクノロジの命令とストリーミング SIMD 拡張命令の整数命令へのインターフェイスを提供します。以下に Ivec クラスを示します。

表 3-1 Ivec のユーザ・クラス

クラス名	説明
I64vec1	_m64 データ型を表します。
Is32vec2	2 つのパックド符号付き 32 ビット値を表します
Iu32vec2	2 つのパックド符号なし 32 ビット値を表します
Is16vec4	4 つのパックド符号付き 16 ビット値を表します
Iu16vec4	4 つのパックド符号なし 16 ビット値を表します
Is8vec8	8 つのパックド符号付き 8 ビット値を表します
Iu8vec8	8 つのパックド符号なし 32 ビット値を表します

表 3-2 Ivec の補助クラス

クラス名	説明
I32vec2	2 つのパックド 32 ビット値を表します。符号の有無は不明 (中間クラス)
I16vec4	4 つのパックド 16 ビット値を表します。符号の有無は不明 (中間クラス)
I8vec8	8 つのパックド 8 ビット値を表します。符号の有無は不明 (中間クラス)
M64	親クラスである __m64 データ型を表します

注：M64 は __m64 データ型ではなく、Ivec の補助クラスのことを指しています

図 3-1 にクラス階層を示します。

図 3-1 Ivec のクラス階層

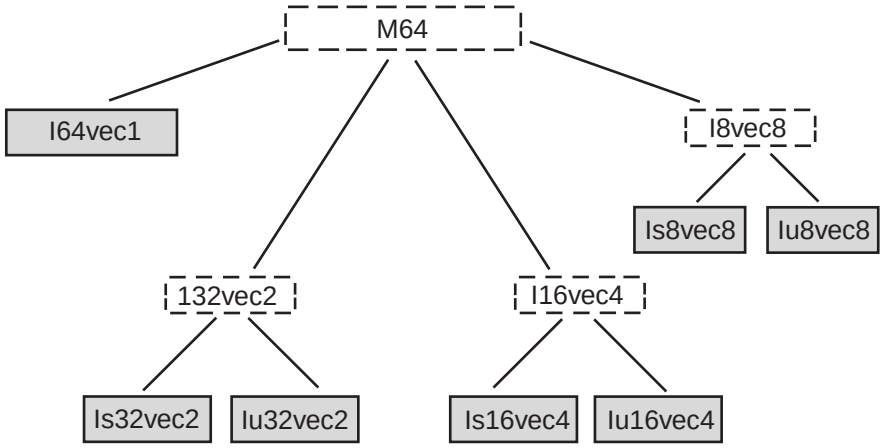


図 3-1 の 3 つの中間クラス I8vec8、I16vec4、および I32vec2 は、その関数と、下位の MMX テクノロジ命令と組み込み関数が、サイズには意味があるが、符号の有無には意味がない値に作用するデータ型を表しています。たとえば、加算や等値比較などの演算は、データの要素の数が等しく、要素 1 つ当たりのビット数が等しければ、符号付きデータと符号なしデータのどちらにも適用できます。

データの符号の有無が問題とならない場合を除き、中間クラスは使用しないことをお勧めします。また、M64 クラスを明示的に使用しないことをお勧めします。これらのクラスはユーザ・クラス I64vec1、Is32vec2、Iu32vec2、Is16vec4、Iu16vec4、Is8vec8、および Iu8vec8 の間での暗黙の変換を支援するために使われるものです。

用語の定義

最も近い共通の原型 (nearest common ancestor)

同じサイズだが、符号の有無が異なるクラスがあった場合、同じサイズの中間クラスを「最も近い共通の原型」として定義します。たとえば、`Iu8vec8` と `Is8vec8` の最も近い共通の原型は `I8vec8` です。

コンストラクタ

```
I64vec1 A;
I32vec2 A;
Is32vec2 A;
Iu32vec2 A;
I16vec4 A;
Is16vec4 A;
Iu16vec4 A;
I8vec8 A;
Is8vec8 A;
Iu8vec8 A;
```

デフォルト・コンストラクタは初期化されていない Ivec オブジェクトを作成します。

対応する組み込み関数：なし

```
I64vec1 A(__m64 m);
I32vec2 A(__m64 m);
Is32vec2 A(__m64 m);
Iu32vec2 A(__m64 m);
I16vec4 A(__m64 m);
Is16vec4 A(__m64 m);
Iu16vec4 A(__m64 m);
I8vec8 A(__m64 m);
Is8vec8 A(__m64 m);
Iu8vec8 A(__m64 m);
```

Ivec オブジェクトを `__m64` データで初期化します。

対応する組み込み関数：なし

```
I64vec1 A = __int64 m;
I32vec2 A =__int64 m;
Is32vec2 A =__int64 m;
Iu32vec2 A =__int64 m;
I16vec4 A =__int64 m;
Is16vec4 A =__int64 m;
Iu16vec4 A =__int64 m;
I8vec8 A =__int64 m;
Is8vec8 A =__int64 m;
Iu8vec8 A =__int64 m;
```

Ivec オブジェクトを `__int64` データで初期化します。

対応する組み込み関数：なし

```
I64vec1 A = int i;
I32vec2 A = int i;
Is32vec2 A = int i;
Iu32vec2 A = int i;
I16vec4 A = int i;
Is16vec4 A = int i;
Iu16vec4 A = int i;
I8vec8 A = int i;
Is8vec8 A = int i;
Iu8vec8 A = int i;
```

Ivec オブジェクトを整数で初期化します。

対応する組み込み関数：`_m_from_int`

```
Is32vec2 A(signed int A1, signed int A0);
```

`Is32vec` オブジェクトを、2 つの符号付き整数を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

```
Iu32vec2 A(unsigned int A1, unsigned int A0);
```

`Iu32vec2` オブジェクトを、2 つの符号なし整数を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

```
Is16vec4 A(signed short A3, signed short A2,
          signed short A1, signed short A0);
```

Is16vec4 オブジェクトを、4 つの符号付き short を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

```
Iu16vec4 A(unsigned short A3, unsigned short A2,
           unsigned short A1, unsigned short A0);
```

Iu16vec4 オブジェクトを、4 つの符号なし short を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

```
Is8vec8 A( signed char A7, signed char A6,
           signed char A5, signed char A4,
           signed char A3, signed char A2,
           signed char A1, signed char A0);
```

Is8vec8 オブジェクトを、8 つの符号付き char を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

```
Iu8vec8 A( unsigned char A7, unsigned char A6,
           unsigned char A5, unsigned char A4,
           unsigned char A3, unsigned char A2,
           unsigned char A1, unsigned char A0);
```

Iu8vec8 オブジェクトを、8 つの符号なし char を使って、右から左の順序で初期化します。

対応する組み込み関数：なし

代入演算子

任意の Ivec オブジェクトを、他の任意の Ivec オブジェクトに代入することができます。1 つの Ivec オブジェクトを他の任意の Ivec オブジェクトに代入したときの変換は自動的に行われます。型キャストは不要です。

```
Is16vec4 A;
Is8vec8 B;
I64vec1 C;
A = B;          /* assign Is8vec8 to Is16vec4 */
B = C;          /* assign I64vec1 to Is8vec8 */
B = A & C;      /* assign M64 result of ‘&’ to Is8vec8 */
```

論理演算子

論理演算子の規則を次に示します。

- 論理演算子のオペランドには、任意の Ivec データ型が使用できます。オペランドには、フィールドの数と符号の有無を混在させることができます。

```
I64vec1 A;
Is16vec4 B;
Iu8vec8 C;
C = A & B;
```

- 論理演算子のサイズが混在しているオペランドは M64 データを返します。このデータは、いずれかのユーザ・クラス型に代入しなければなりません。得られた結果は、Ivec クラス型に暗黙のうちに変換されます (左オペランド)。

```
I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = A & B; /* A and B converted to M64. Result assigned to Iu8vec8.*/
```

- 異なるサイズのオペランドに挟まれている論理演算子を非論理式の中で使用した場合、論理演算の結果 (M64 オブジェクト) は別の型にキャストされなくてはなりません。

```
I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = Iu8vec8(A&B)+ C; /*A&B returns M64, which is cast to Iu8vec8 */
```

- 同じサイズだが、符号の有無が異なる論理演算子のオペランドは、「最も近い共通の祖先」のデータを返します。

```
I32vec2 R = Is32vec2 A ^ Iu32vec2 B;
```

- 同じサイズで、符号の有無も等しい論理演算子のオペランドは、同じサイズと符号の有無を持つデータを返し、暗黙の変換は行われません。

```
Iu8vec8 R = Iu8vec8 A & Iu8vec8 B;
```

- 論理演算子はすべてのユーザ・クラスに存在します。

注. *ここでは、わかりやすいように、論理式の考えうる組み合わせの一部だけを示しています。*

ビット単位の AND 演算子

```
I64vec1 R = I64vec1 A & I64vec1 B;

Is32vec2 R = Is32vec2 A & Is32vec2 B;
I32vec2 R = Is32vec2 A & Iu32vec2 B;
I32vec2 R = Is32vec2 A & I32vec2 B;
Iu32vec2 R = Iu32vec2 A & Iu32vec2 B;

Is16vec4 R = Is16vec4 A & Is16vec4 B;
I16vec4 R = Is16vec4 A & Iu16vec4 B;
I16vec4 R = Is16vec4 A & I16vec4 B;
Iu16vec4 R = Iu16vec4 A & Iu16vec4 B;

Is8vec8 R = Is8vec8 A & Is8vec8 B;
I8vec8 R = Is8vec8 A & Iu8vec8 B;
I8vec8 R = Is8vec8 A & I8vec8 B;
Iu8vec8 R = Iu8vec8 A & Iu8vec8 B;

M64 R = Iu8vec8 A & Is8vec8 B &  Iu32vec2 C & Is32vec2 D & I64vec1 D;
/* Assign to one of main Ivec classes */
```

A と B のビット単位の論理積 (AND) を求めます。

対応する組み込み関数：_m_pand

```
I64vec1 R &= I64vec1 A;

Is32vec2 R &= Is32vec2 A;
Is32vec2 R &= Iu32vec2 A;
Is32vec2 R &= I32vec2 A;
```

```
Iu16vec4 R &= Is16vec4 A;
Iu16vec4 R &= Iu16vec4 A;
Iu16vec4 R &= I16vec4 A;

Is8vec8 R &= Is8vec8 A;
Is8vec8 R &= Iu8vec8 A;
Is8vec8 R &= I8vec8 A;
```

R と A のビット単位の論理積 (AND) を実行します。
対応する組み込み関数：_m_pand

ビット単位の OR 演算子

```
I64vec1 R = I64vec1 A | I64vec1 B;

Is32vec2 R = Is32vec2 A | Is32vec2 B;
I32vec2 R = Is32vec2 A | Iu32vec2 B;
I32vec2 R = Is32vec2 A | I32vec2 B;
Iu32vec2 R = Iu32vec2 A | Iu32vec2 B;

Is16vec4 R = Is16vec4 A | Is16vec4 B;
I16vec4 R = Is16vec4 A | Iu16vec4 B;
I16vec4 R = Is16vec4 A | I16vec4 B;
Iu16vec4 R = Iu16vec4 A | Iu16vec4 B;

Is8vec8 R = Is8vec8 A | Is8vec8 B;
I8vec8 R = Is8vec8 A | Iu8vec8 B;
I8vec8 R = Is8vec8 A | I8vec8 B;
Iu8vec8 R = Iu8vec8 A | Iu8vec8 B;
```

A と B のビット単位の論理和 (OR) を実行します。
対応する組み込み関数：_m_por

```
I64vec1 R |= I64vec1 A;

Is32vec2 R |= Is32vec2 A;
Is32vec2 R |= Iu32vec2 A;
Is32vec2 R |= I32vec2 A;
```

```
Iu16vec4 R |= Is16vec4 A;
Iu16vec4 R |= Iu16vec4 A;
Iu16vec4 R |= I16vec4 A;

Is8vec8 R |= Is8vec8 A;
Is8vec8 R |= Iu8vec8 A;
Is8vec8 R |= I8vec8 A;
```

R と A のビット単位の論理和 (OR) を実行します。
対応する組み込み関数：_m_por

ビット単位の XOR 演算子

```
I64vec1 R = I64vec1 A ^ I64vec1 B;

Is32vec2 R = Is32vec2 A ^ Is32vec2 B;
I32vec2 R = Is32vec2 A ^ Iu32vec2 B;
I32vec2 R = Is32vec2 A ^ I32vec2 B;
Iu32vec2 R = Iu32vec2 A ^ Iu32vec2 B;

Is16vec4 R = Is16vec4 A ^ Is16vec4 B;
I16vec4 R = Is16vec4 A ^ Iu16vec4 B;
I16vec4 R = Is16vec4 A ^ I16vec4 B;
Iu16vec4 R = Iu16vec4 A ^ Iu16vec4 B;

Is8vec8 R = Is8vec8 A ^ Is8vec8 B;
I8vec8 R = Is8vec8 A ^ Iu8vec8 B;
I8vec8 R = Is8vec8 A ^ I8vec8 B;
Iu8vec8 R = Iu8vec8 A ^ Iu8vec8 B;
```

A と B のビット単位の排他的論理和 (XOR) を実行します。
対応する組み込み関数：_m_pxor

```
I64vec1 R ^= I64vec1 A;

Is32vec2 R ^= Is32vec2 A;
Is32vec2 R ^= Iu32vec2 A;
Is32vec2 R ^= I32vec2 A;

Iu16vec4 R ^= Is16vec4 A;
Iu16vec4 R ^= Iu16vec4 A;
Iu16vec4 R ^= I16vec4 A;

Is8vec8 R ^= Is8vec8 A;
Is8vec8 R ^= Iu8vec8 A;
Is8vec8 R ^= I8vec8 A;
```

R と A のビット単位の排他的論理和 (XOR) を実行します。
対応する組み込み関数：_m_pxor

ビット単位の NAND 演算子

```
I64vec1 R = andnot(I64vec1 A, I64vec1 B);

Is32vec2 R = andnot(Is32vec2 A, Is32vec2 B);
I32vec2 R = andnot(Is32vec2 A, Iu32vec2 B);
I32vec2 R = andnot(Is32vec2 A, I32vec2 B);
Iu32vec2 R = andnot(Iu32vec2 A, Iu32vec2 B);

Is16vec4 R = andnot(Is16vec4 A, Is16vec4 B);
I16vec4 R = andnot(Is16vec4 A, Iu16vec4 B);
I16vec4 R = andnot(Is16vec4 A, I16vec4 B);
Iu16vec4 R = andnot(Iu16vec4 A, Iu16vec4 B);

Is8vec8 R = andnot(Is8vec8 A, Is8vec8 B);
I8vec8 R = andnot(Is8vec8 A, Iu8vec8 B);
I8vec8 R = andnot(Is8vec8 A, I8vec8 B);
Iu8vec8 R = andnot(Iu8vec8 A,Iu8vec8 B);
```

```
M64 R = andnot(Is8vec8 A, Is16vec4 B);
M64 R = andnot(Iu8vec8 A, I64vec1 B);
```

A のビット単位の論理否定 (NOT) を実行し、その結果を使用して B とのビット単位の論理積 (AND) を実行します。

対応する組み込み関数：_m_pandn

加算および減算演算子

加算および減算演算子の規則を次に示します。

- オペランドの要素の数は同じでなければなりません。サイズの異なる Ivec オブジェクトに対する演算を行う場合、明示的なキャストが必要です。

```
Is16vec4 A,C;
Iu32vec2 B;
C = A + C;
C = A + (Is16vec4)B; /* Explicitly convert B to a Is16vec4 */
```

- オペランドは、符号の有無が同じである必要はありません。符号の有無が混在しているが、フィールドの数が等しい Ivec オペランドを含んでいる加算式と減算式は、「最も近い共通の祖先」の型を返します。

```
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B; /* return nearest common ancestor type, I16vec4 */
```

- 同じ型 (符号の有無とサイズが等しい) の Ivec オペランドは、その型を返します。

```
Iu32vec2 A,B,C;
C = A - B;
```

- 減算および加算代入演算子は同じサイズのオペランドを持っていなくてはなりません。右オペランドの符号の有無は関係ありません。返される結果型は、左オペランドの型です。

```
Is16vec4 A;
Iu16vec4 B;
A += B;
B -= A;
```

- 加算および減算演算子は、I64vec1 クラスを除くすべてのユーザ・クラスに存在します。

加算演算子

```
Is32vec2 R = Is32vec2 A + Is32vec2 B;
I32vec2 R = Is32vec2 A + Iu32vec2 B;
I32vec2 R = Is32vec2 A + I32vec2 B;
I32vec2 R = Iu32vec2 A + Is32vec2 B;
Iu32vec2 R = Iu32vec2 A + Iu32vec2 B;
I32vec2 R = Iu32vec2 A + I32vec2 B;
```

A の 2 つの 32 ビット値を、B の 2 つの 32 ビット値に加えます。

対応する組み込み関数：_m_paddd

```
Is32vec2 R += Is32vec2 A;
Iu32vec2 R += Iu32vec2 A;
Is32vec2 R += Iu32vec2 A;
Iu32vec2 R += Is32vec2 A;
Is32vec2 R += I32vec2 A;
Iu32vec2 R += I32vec2 A;
```

R の 2 つの 32 ビット値を、A の 2 つの 32 ビット値に加えます。

対応する組み込み関数：_m_paddd

```
Is16vec4 R = Is16vec4 A + Is16vec4 B;
I16vec4 R = Is16vec4 A + Iu16vec4 B;
I16vec4 R = Is16vec4 A + I16vec4 B;
```

```
I16vec4 R = Iu16vec4 A + Is16vec4 B;
Iu16vec4 R = Iu16vec4 A + Iu16vec4 B;
I16vec4 R = Iu16vec4 A + I16vec4 B;
```

A の 4 つの 16 ビット値を、B の 4 つの 16 ビット値に加えます。

対応する組み込み関数：_m_paddw

```
Is16vec4 R += Is16vec4 A;
Is16vec4 R += Iu16vec4 A;
Is16vec4 R += I16ec4 A;
Iu16vec4 R += Is16vec4 A;
Iu16vec4 R += Iu16vec4 A;
Iu16vec4 R += I16vec4 A;
```

A の 4 つの 16 ビット値を、R の 4 つの 16 ビット値に加えます。

対応する組み込み関数：_m_paddw

```
Is16vec4 R = sat_add(Is16vec4 A, Is16vec4 B);
```

A の 4 つの符号付き 16 ビット値を、B の 4 つの符号付き 16 ビット値に加え、飽和させます。

対応する組み込み関数：_m_paddsw

```
Iu16vec4 R = sat_add(Iu16vec4, Iu16vec4);
```

A の 4 つの符号なし 16 ビット値を、B の 4 つの符号なし 16 ビット値に加え、飽和させます。

対応する組み込み関数：_m_paddusw

```
Is8vec8 R = Is8vec4 A + Is8vec8 B;
```

```
I8vec8 R = Is8vec4 A + Iu8vec8 B;
I8vec8 R = Is8vec4 A + I8vec8 B;
I8vec8 R = Iu8vec4 A + Is8vec8 B;
Iu8vec8 R = Iu8vec4 A + Iu8vec8 B;
I8vec8 R = Iu8vec4 A + I8vec8 B;
```

A の 8 つの 8 ビット値を、B の 8 つの 8 ビット値に加えます。

対応する組み込み関数：_m_paddb

```
Is8vec8 R += Is8vec8 A;
Is8vec8 R += Iu8vec8 A;
Is8vec8 R += I8vec8 A;
Iu8vec8 R += Is8vec8 A;
Iu8vec8 R += Iu8vec8 A;
Iu8vec8 R += I8vec8 A;
```

A の 8 つの 8 ビット値を、R の 8 つの 8 ビット値に加えます。

対応する組み込み関数：_m_paddb

```
Is8vec8 R = sat_add(Is8vec8 A, Is8vec8 B);
```

A の 8 つの符号付き 8 ビット値を、B の 8 つの符号付き 8 ビット値に加え、飽和させます。

対応する組み込み関数：_m_paddsb

```
Iu8vec8 R = sat_add(Iu8vec8 A, Iu8vec8 B);
```

A の 8 つの符号なし 8 ビット値を、B の 8 つの符号なし 8 ビット値に加え、飽和させます。

対応する組み込み関数：_m_paddusb

減算演算子

```
Is32vec2 R = Is32vec2 A - Is32vec2 B;
I32vec2 R = Is32vec2 A - Iu32vec2 B;
```

```
I32vec2 R = Is32vec2 A - I32vec2 B;
I32vec2 R = Iu32vec2 A - Is32vec2 B;
Iu32vec2 R = Iu32vec2 A - Iu32vec2 B;
I32vec2 R = Iu32vec2 A - I32vec2 B;
```

A の 2 つの 32 ビット値から、B の 2 つの 32 ビット値を引きます。
対応する組み込み関数：_m_psubd

```
Is32vec2 R -= Is32vec2 A;
Is32vec2 R -= Iu32vec2 A;
Is32vec2 R -= I32vec2 A;
Iu32vec2 R -= Is32vec2 A;
Iu32vec2 R -= Iu32vec2 A;
Iu32vec2 R -= I32vec2 A;
```

R の 2 つの 32 ビット値から、A の 2 つの 32 ビット値を引きます。
対応する組み込み関数：_m_psubd

```
Is16vec4 R = Is16vec4 A - Is16vec4 B;
I16vec4 R = Is16vec4 A - Iu16vec4 B;
I16vec4 R = Is16vec4 A - I16vec4 B;
I16vec4 R = Iu16vec4 A - Is16vec4 B;
Iu16vec4 R =Iu16vec4 A - Iu16vec4 B;
I16vec4 R = Iu16vec4 A - I16vec4 B;
```

A の 4 つの 16 ビット値から、B の 4 つの 16 ビット値を引きます。
対応する組み込み関数：_m_psubw

```
Is16vec4 R -= Is16vec4 A;
Is16vec4 R -= Iu16vec4 A;
Is16vec4 R -= I16vec4 A;
Iu16vec4 R -= Is16vec4 A;
Iu16vec4 R -= Iu16vec4 A;
Iu16vec4 R -= I16vec4 A;
```

R の 4 つの 16 ビット値から、A の 4 つの 16 ビット値を引きます。

対応する組み込み関数：_m_psubw

```
Is16vec4 R = sat_sub(Is16vec4 A, Is16vec4 B)
```

A の 4 つの符号付き 16 ビット値から、B の 4 つの符号月 16 ビット値を引き、飽和させます。

対応する組み込み関数：_m_psubsw

```
Iu16vec4 R = sat_sub(Iu16vec4 A, Iu16vec4 B)
```

A の 4 つの符号なし 16 ビット値から、B の 4 つの符号なし 16 ビット値を引き、飽和させます。

対応する組み込み関数：_m_psubusw

```
Is8vec8 R = Is8vec4 A - Is8vec8 B;
I8vec8 R = Is8vec4 A - Iu8vec8 B;
I8vec8 R = Is8vec4 A - I8vec8 B;
I8vec8 R = Iu8vec4 A - Is8vec8 B;
Iu8vec8 R = Iu8vec4 A - Iu8vec8 B;
I8vec8 R = Iu8vec4 A - I8vec8 B;
```

A の 8 つの 8 ビット値から、B の 8 つの 8 ビット値を引きます。

対応する組み込み関数：_m_psubb

```
Is8vec8 R -= Is8vec8 A;
Is8vec8 R -= Iu8vec8 A;
Is8vec8 R -= I8vec8 A;
Iu8vec8 R -= Is8vec8 A;
Iu8vec8 R -= Iu8vec8 A;
Iu8vec8 R -= I8vec8 A;
```

R の 8 つの 8 ビット値から、A の 8 つの 8 ビット値を引きます。

対応する組み込み関数：_m_psubb

```
Is8vec8 R = sat_sub(Is8vec8 A, Is8vec8 B);
```

Aの8つの符号付き8ビット値から、Bの8つの符号付き8ビット値を引き、飽和させます。

対応する組み込み関数：_m_psubsb

```
Iu8vec8 R = sat_sub(Iu8vec8 A, Iu8vec8 B);
```

Aの8つの符号なし8ビット値から、Bの8つの符号なし8ビット値を引き、飽和させます。

対応する組み込み関数：_m_psubusb

乗算演算子

乗算演算子の規則を次に示します。

- * 演算子のオペランドは、I16vec4、Is16vec4、または Iu16vec4 オブジェクトでなくてはなりません。サイズの異なる Ivec オブジェクトに対して演算を行う場合、明示的なキャストが必要です。

```
Is16vec4 A,C;
Iu32vec2 B;
C = A * C;
C = A * (Is16vec4)B; /* Explicitly convert B to Is16vec4 */
```

- * 演算子のオペランドは、4 つの 16 ビット要素を持っているかぎり、符号の有無が同じである必要はありません。符号の有無が混在している * 演算子のオペランドは、「最も近い共通の原型」型を返します。

```
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B; /* return nearest common ancestor type, I16vec4 */
```

- 同じ型 (符号の有無とサイズが同じ) の * 演算子の Ivec オペランドは、その型を返します。

```
Is32vec2 A,B,C;
C = A * B;
```

- `mul_high` および `mul_add` 関数は Is16vec4 データだけを受け付けます。

```
Is16vec4 A,B,C,D;
C = mul_high(A,B);
D = mul_add(A,B);
```

```
Is16vec4 R = Is16vec4 A * Is16vec4 B;
I16vec4 R = Is16vec4 A * Iu16vec4 B;
I16vec4 R = Is16vec4 A * I16vec4 B;
I16vec4 R = Iu16vec4 A * Is16vec4 B;
Iu16vec4 R = Iu16vec4 A * Iu16vec4 B;
I16vec4 R = Iu16vec4 A * I16vec4 B;
```

`A` の 4 つの 16 ビット値に、`B` の 4 つの 16 ビット値を掛け、4 つの結果の下位 16 ビットを取ります。
対応する組み込み関数：`_m_pmullw`

```
Is16vec4 R *= Is16vec4 A;
Is16vec4 R *= Iu16vec4 A;
Is16vec4 R *= I16vec4 A;
Iu16vec4 R *= Is16vec4 A;
Iu16vec4 R *= Iu16vec4 A;
Iu16vec4 R *= I16vec4 A;
```

R の 4 つの 16 ビット値に、A の 4 つの 16 ビット値を掛け、4 つの結果の下位 16 ビットを取ります。
対応する組み込み関数：_m_pmullw

```
Is16vec4 R = mul_high(Is16vec4 A, Is16vec4 B);
```

A の 4 つの符号付き 16 ビット値に、B の 4 つの符号付き 16 ビット値を掛け、4 つの結果の上位 16 ビットを取ります。
対応する組み込み関数：_m_pmulhw

```
Is32vec2 R = mul_add(const Is16vec4 A, const Is16vec4 B);
```

A の 4 つの符号付き 16 ビット値に、B の 4 つの符号付き 16 ビット値を掛け、4 つの 32 ビット中間値を得ます。これらの中間値は 2 組に分けて加算され、2 つの 32 ビット値 (符号付き) が得られます。
対応する組み込み関数：_mul_pmaddwd

シフト演算子

シフト演算子の規則を次に示します。

- 右シフト引数は任意の引数または Ivec 値です。右オペランドは暗黙のうちに M64 データ型に変換されます。
- '<<' の第 1 または左オペランドは、I64vec1、I32vec2、Is32vec2、Iu32vec2、I16vec4、Is16vec4、または Iu16vec4 オブジェクトでなくてはなりません。'>>' の第 1 オペランドは、I64vec1、Is32vec2、Iu32vec2、Is16vec4、または Iu16vec4 オブジェクトでなくてはなりません。'<<' と '>>' の第 1 オペランドは、I8vec8、Is8vec8、または Iu8vec8 値であってはなりません。

- シフト演算のオペランドは、サイズと符号の有無が混在していてもかまいません。シフト演算子の第 1 引数は、第 2 引数と同じ Ivec 型である必要はありません。

```
Is16vec4 A,C;
Iu32vec2 B;
C = A << B;
```

- すべてのシフト演算の戻り型は、第 1 引数の型と一致します。

```
Is16vec4 A, count;
Is16vec4 R = A << count;
```

- 符号付きクラスの ‘>>’(第 1 引数が `Is32vec2` または `Is16vec4` 値) は算術シフトに対応します。
- 符号なしクラスの ‘>>’ と ‘<<’(第 1 引数が `Iu32vec2` または `Iu16vec4` 値) は論理シフトに対応します。
- `I64vec1` クラスの ‘>>’ と ‘<<’(第 1 引数が `I64vec1` 値) は論理シフトに対応します。
- ‘>>’ の第 1 引数は、中間型であってはなりません。‘>>’ 演算子は中間クラスには存在しません。右シフトの左引数が `I32vec2` または `I16vec4` オブジェクトである場合、適切な論理シフト算術シフトが行われるようにするため、明示的なキャストが必要です。

```
Is16vec4 A,C;
Iu16vec4 B, R;
R = (Iu16vec4)(A & B) >> C;
    /* A&B returns I16vec4, which must be cast to */
    /* Iu16vec4 to ensure logical shift, not */
    /* arithmetic shift */
R = (Is16vec4)(A & B) >> C;
    /* A&B returns I16vec4, which must be cast to */
    /* Is16vec4 to ensure arithmetic shift,  */
    /* not logical shift */
```

- I8vec8 以外の中間クラスの ‘<<’ は、論理シフトに対応します。

```
Is32vec2 A,C;
Iu32vec2 B;
C = (A & B) << C;
I64vec1 R = I64vec1 A << M64 count;
    /* 2nd arg can be any ivec value */
```

左シフト演算子

```
I64vec1 R = I64vec1 A << int count;
I64vec1 R <<= M64 count;
I64vec1 R <<= int count;
```

A の 64 ビット値を、ゼロでシフトしながら count で指定されたビットだけ左にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_psllq、_m_psllqi

```
I32vec2 R = I32vec2 A << M64 count;
Iu32vec2 R = Iu32vec2 A << M64 count;
Is32vec2 R = Is32vec2 A << M64 count;
I32vec2 R = I32vec2 A << int count;
Iu32vec2 R = Iu32vec2 A << int count;
Is32vec2 R = Is32vec2 A << int count;
I32vec2 A <<= M64 count;
Iu32vec2 A <<= M64 count;
Is32vec2 A <<= M64 count;
I32vec2 A <<= int count;
Iu32vec2 A <<= int count;
Is32vec2 A <<= int count;
```

A の 2 つの 32 ビット値を、ゼロでシフトしながら count で指定されたビットだけ左にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_pslld、_m_pslldi

```
I16vec4 R = I16vec4 A << M64 count;
Is16vec4 R = Is16vec4 A << M64 count;
I16vec4 R = I16vec4 A << int count;
Is16vec4 R = Is16vec4 A << int count;
I16vec4 A <<= M64 count;
Is16vec4 A <<= M64 count;
I16vec4 A <<= int count;
Is16vec4 A <<= int count;
```

A の 4 つの 16 ビット値を、ゼロでシフトしながら count で指定されたビットだけ左にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_psllw、_m_psllwi

右シフト演算子

```
I64vec1 R = I64vec1 A >> M64 count;
I64vec1 R = I64vec1 A >> int count;
I64vec1 A >>= M64 count;
I64vec1 A >>= int count;
```

A の 64 ビット値を、ゼロでシフトしながら count で指定されたビットだけ右にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数 : _m_psrlq、_m_psrlqi

```
Iu32vec2 R = Iu32vec2 A >> M64 count;
Iu32vec2 R = Iu32vec2 A >> int count;
Iu32vec2 A >>= M64 count;
Iu32vec2 A >>= int count;
```

A の 2 つの 32 ビット値を、ゼロでシフトしながら count で指定されたビットだけ右にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_psrld、_m_psrldi

```
Is32vec2 R = Is32vec2 A >> M64 count;
Is32vec2 R = Is32vec2 A >> int count;
Is32vec2 A >>= M64 count;
Is32vec2 A >>= int count;
```

A の 2 つの 32 ビット値を、符号ビットでシフトしながら count で指定されたビットだけ右にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_psrad、_m_psradi

```
Is16vec4 R = Is16vec4 A >> M64 count;
Is16vec4 R = Is16vec4 A >> int count;
Is16vec4 A >>= M64 count;
Is16vec4 A >>= int count;
```

A の 4 つの 16 ビット値を、符号ビットでシフトしながら count で指定されたビットだけ右にシフトします。第 2 引数の M64 は、任意の Ivec 値が使用できることを示しています。

対応する組み込み関数：_m_psraw、_m_psrawi

「等しい」または「等しくない」の比較

「等しい」と「等しくない」の比較の規則を次に示します。

- オペランドは、符号の有無は混在してもかまいませんが、サイズは同じでなくてはなりません。「最も近い共通の原型」型が返されます。

  ```
  Iu8vec8 A;
  Is8vec8 B;
  I8vec8 C;
  C = cmpneq(A,B);
  ```

- 同じ型 (サイズと符号の有無が同じ) のオペランドは、その型を返します。

  ```
  Iu8vec8 R = cmpeq(Iu8vec8,Iu8vec8)
  ```

- サイズの異なる要素の間で「等しい」と「等しくない」を比較する場合、明示的な型キャストを行う必要があります。

```
Iu8vec8 A,C;
Is16vec4 B;
C = cmpeq(A,(Iu8vec8)B);
```

- 「等しい」と「等しくない」の比較関数は、次の演算子型を持ちます : I32vec2、I16vec4、I8vec8、Is32vec2、Is16vec4、Is8vec8、Iu32vec2、Iu16vec4、Iu8vec8。I64vec1 データを受け付け可能な型に変換するためには、明示的なキャストを使用する必要があります。

```
Iu8vec8 A,C;
I64vec1 B;
C = cmpeq(A,(Iu8vec8)B);
```

```
Is32vec2 R = cmpeq(Is32vec2 A, Is32vec2 B);
Iu32vec2 R = cmpeq(Iu32vec2 A, Iu32vec2 B);
I32vec2 R = cmpeq(I32vec2 A, I32vec2 B);
I32vec2 R = cmpeq(Is32vec2 A, Iu32vec2 B);
I32vec2 R = cmpeq(Iu32vec2 A, Is32vec2 B);
```

```
Is32vec2 R = cmpneq(Is32vec2 A, Is32vec2 B);
Iu32vec2 R = cmpneq(Iu32vec2 A, Iu32vec2 B);
I32vec2 R = cmpneq(I32vec2 A, I32vec2 B);
I32vec2 R = cmpneq(Is32vec2 A, Iu32vec2 B);
I32vec2 R = cmpneq(Iu32vec2 A, Is32vec2 B);
```

A の中の個々の 32 ビット値が、B の中の対応する 32 ビット値と等しい／等しくない場合に、結果の中のそれぞれの 32 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpeqd、_m_pandn

```
Is16vec4 R = cmpeq(Is16vec4 A, Is16vec4 B);
Iu16vec4 R = cmpeq (Iu16vec4 A, Iu16vec4 B);
I16vec4 R = cmpeq(I16vec4 A, I16vec4 B);
I16vec4 R = cmpeq (Is16vec4 A, Iu16vec4 B);
I16vec4 R = cmpeq (Iu16vec4 A, Is16vec4 B);
```

```
Is16vec4 R = cmpneq(Is16vec4 A, Is16vec4 B);
Iu16vec4 R = cmpneq (Iu16vec4 A, Iu16vec4 B);
I16vec4 R = cmpneq(I16vec4 A, I16vec4 B);
I16vec4 R = cmpneq (Is16vec4 A, Iu16vec4 B);
I16vec4 R = cmpneq (Iu16vec4 A, Is16vec4 B);
```

A の中の個々の 16 ビット値が、B の中の対応する 16 ビット値と等しい／等しくない場合に、結果のそれぞれの 16 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpeqw、_m_pandn

```
Is8vec8 R = cmpeq (Is8vec8 A, Is8vec8 B);
Iu8vec8 R = cmpeq (Iu8vec8 A, Iu8vec8 B);
I8vec8 R = cmpeq (I8vec8 A, I8vec8 B);
I8vec8 R = cmpeq (Is8vec8 A, Iu8vec8 B);
I8vec8 R = cmpeq (Iu8vec8 A, Is8vec8 B);

Is8vec8 R = cmpneq (Is8vec8 A, Is8vec8 B);
Iu8vec8 R = cmpneq (Iu8vec8 A, Iu8vec8 B);
I8vec8 R = cmpneq (I8vec8 A, I8vec8 B);
I8vec8 R = cmpneq (Is8vec8 A, Iu8vec8 B);
I8vec8 R = cmpneq (Iu8vec8 A, Is8vec8 B);
```

A の中の個々の 8 ビット値が、B の中の対応する 8 ビット値と等しい／等しくない場合に、結果のそれぞれの 8 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpeqb、_m_pandn

その他の比較演算

「より大きい」、「より小さい」、「より大きい、または等しい」、「より小さい、または等しい」演算の比較の規則を次に示します。

- オペランドは同じサイズの符号付きの値でなくてはなりません。
- オペランドは同じ型でなくてはなりません。この型が戻り型になります。

```
Is16vec4 cmpge(Is16vec4,Is16vec4);
```

- 両方のオペランドが Is8vec8、Is16vec4、または Is32vec2 でなくてはなりません。他の型、すなわち I8vec8、Iu8vec8、I16vec4、Iu16vec4、I32vec4、Iu32vec4 は、これらの符号付きクラス型の 1 つに明示的にキャストする必要があります。

```
Iu16vec4 A;
Is16vec4 B,C;
C = cmpge((Is16vec4)A,B);
C = cmpgt(B,C);
```

```
Is32vec2 R = cmpgt(Is32vec2 A, Is32vec2 B);
Is32vec2 R = cmplt(Is32vec2 A, Is32vec2 B);
Is32vec2 R = cmpge(Is32vec2 A, Is32vec2 B);
Is32vec2 R = cmple(Is32vec2 A, Is32vec2 B);
```

A の中の個々の 32 ビット値が、B の中の対応する 32 ビット値と比べて「より大きい」／「より小さい」／「より大きい、または等しい」／「より小さい、または等しい」場合に、結果の中のそれぞれの 32 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpgtd、_m_pandn

```
Is16vec4 R = cmpgt(Is16vec4 A, Is16vec4 B);
Is16vec4 R = cmplt(Is16vec4 A, Is16vec4 B);
Is16vec4 R = cmpge(Is16vec4 A, Is16vec4 B);
Is16vec4 R = cmple(Is16vec4 A, Is16vec4 B);
```

A の中の個々の 16 ビット値が、B の中の対応する 16 ビット値と比べて「より大きい」／「より小さい」／「より大きい、または等しい」／「より小さい、または等しい」場合に、結果の中のそれぞれの 16 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpgtw、_m_pandn

```
Is8vec8 R = cmpgt (Is8vec8 A, Is8vec8 B);
Is8vec8 R = cmplt (Is8vec8 A, Is8vec8 B);
Is8vec8 R = cmpge (Is8vec8 A, Is8vec8 B);
Is8vec8 R = cmple (Is8vec8 A, Is8vec8 B);
```

A の中の個々の 8 ビット値が、B の中の対応する 8 ビット値と比べて「より大きい」／「より小さい」／「より大きい、または等しい」／「より小さい、または等しい」場合に、結果の中のそれぞれの 8 ビット値をすべて 1 に設定します。それ以外の場合は、すべてゼロに設定します。

対応する組み込み関数：_m_pcmpgtb、_m_pandn

「等しい」と「等しくない」の条件付き選択

「等しい」と「等しくない」の条件付き選択演算の規則を次に示します。

- オペランドは、符号の有無は混在してもかまいませんが、サイズは同じでなくてはなりません。
- 第 3 と第 4 のオペランドが、返される型を決定します。第 3 と第 4 のオペランドが同じサイズだが、符号の有無が異なる場合には、「最も近い共通の原型」のデータが返されます。

```
I16vec4 select_neq(Is16vec4,Is16vec4, Is16vec4, Iu16vec4);
```

- 第 3 と第 4 のオペランドが同じ型 (サイズと符号の有無が同じ) である場合には、その型が返されます。

```
Is16vec4 select_eq(Is16vec4,Is16vec4, Is16vec4, Is16vec4);

Is32vec2 R = select_eq(I32vec2 A, I32vec2 B, Is32vec2 C, Is32vec2 D);
Iu32vec2 R = select_eq(I32vec2 A, I32vec2 B, Iu32vec2 C, Iu32vec2 D);
I32vec2 R = select_eq(I32vec2 A, I32vec2 B, I32vec2 C, I32vec2 D);
```

「等しい」の条件付き選択。

```
R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
```

対応する組み込み関数：_m_pand、_m_por、_m_pcmpeqd

```
Is32vec2 R = select_neq(I32vec2 A, I32vec2 B, Is32vec2 C, Is32vec2 D);
Iu32vec2 R = select_neq(I32vec2 A, I32vec2 B, Iu32vec2 C, Iu32vec2 D);
I32vec2 R = select_neq(I32vec2 A, I32vec2 B, I32vec2 C, I32vec2 D);
```

「等しくない」の条件付き選択。

```
R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpeqd

```
Is16vec4 R = select_eq(I16vec4 A, I16vec4 B, Is16vec4 C, Is16vec4 D);
Iu16vec4 R = select_eq(I16vec4 A, I16vec4 B, Iu16vec4 C, Iu16vec4 D);
I16vec4 R = select_eq(I16vec4 A, I16vec4 B, I16vec4 C, I16vec4 D);
```

「等しい」の条件付き選択。

```
R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;
```

対応する組み込み関数：_m_pand、_m_por、_m_pcmpeqd

```
Is16vec4 R = select_neq(I16vec4 A, I16vec4 B, Is16vec4 C, Is16vec4 D);
Iu16vec4 R = select_neq(I16vec4 A, I16vec4 B, Iu16vec4 C, Iu16vec4 D);
I16vec4 R = select_neq(I16vec4 A, I16vec4 B, I16vec4 C, I16vec4 D);
```

「等しくない」の条件付き選択。

```
R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
R2 := (A2 != B2) ? C2 : D2;
R3 := (A3 != B3) ? C3 : D3;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpeqw

```
Is8vec8 R = select_eq(I8vec8 A, I8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_eq(I8vec8 A, I8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_eq(I8vec8 A, I8vec8 B, I8vec8 C, I8vec8 D);
```

「等しい」の条件付き選択。

```
R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;
R4 := (A4 == B4) ? C4 : D4;
R5 := (A5 == B5) ? C5 : D5;
R6 := (A6 == B6) ? C6 : D6;
R7 := (A7 == B7) ? C7 : D7;
```

対応する組み込み関数：_m_pand、_m_por、_m_pcmpeqd

```
Is8vec8 R = select_neq(I8vec8 A, I8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_neq(I8vec8 A, I8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_neq(I8vec8 A, I8vec8 B, I8vec8 C, I8vec8 D);
```

「等しくない」の条件付き選択。

```
R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
R2 := (A2 != B2) ? C2 : D2;
R3 := (A3 != B3) ? C3 : D3;
R4 := (A4 != B4) ? C4 : D4;
R5 := (A5 != B5) ? C5 : D5;
R6 := (A6 != B6) ? C6 : D6;
R7 := (A7 != B7) ? C7 : D7;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpeqb

その他の条件付き選択演算

「より大きい」、「より小さい」、「以上」、「以下」の条件付き選択の規則を次に示します。

- 第 1 と第 2 のオペランドは、同じサイズの符号付きの値でなくてはなりません。第 3 と第 4 のオペランドは、第 1 と第 2 のオペランドと同じサイズでなくてはなりません。

```
Is16vec4 R = select_gt(Is16vec4,Is16vec4,Is16vec4,Is16vec4);
```

- 第 3 と第 4 のオペランドが同じ型であれば、その型が返されます。これらのオペランドの符号の有無が異なる場合には、「最も近い共通の原型」が返されます。

```
Is16vec4 R = select_gt(Is16vec4,Is16vec4,Is16vec4,Is16vec4);
Iu16vec4 R = select_gt(Is16vec4,Is16vec4,Iu16vec4,Iu16vec4);
I16vec4 R = select_gt(Is16vec4,Is16vec4,Is16vec4,Iu16vec4);
```

- 第 1 と第 2 のオペランドは Is8vec8、Is16vec4、または Is32vec2 でなくてはなりません。その他の型、すなわち I8vec8、Iu8vec8、I16vec4、Iu16vec4、I32vec4、Iu32vec4 では、これらの符号付きクラス型の 1 つへの明示的なキャストが必要です。

```
Iu16vec4 A;
Is16vec4 B,C;
C = select_ge((Is16vec4)A,B,C,D);
```

```
Is32vec2 R = select_gt(Is32vec2 A, Is32vec2 B, Is32vec2 C,Is32vec2 D);
Iu32vec2 R = select_gt(Is32vec2 A, Is32vec2 B, Iu32vec2 C,Iu32vec2 D);
I32vec2 R = select_gt(Is32vec2 A, Is32vec2 B, I32vec2 C,I32vec2 D);
```

「より大きい」の条件付き選択。

```
R0 := (A0 > B0) ? C0 : D0;
R1 := (A1 > B1) ? C1 : D1;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtd

```
Is32vec2 R = select_lt(Is32vec2 A, Is32vec2 B, Is32vec2 C,Is32vec2 D);
Iu32vec2 R = select_lt(Is32vec2 A, Is32vec2 B, Iu32vec2 C,Iu32vec2 D);
I32vec2 R = select_lt(Is32vec2 A, Is32vec2 B, I32vec2 C,I32vec2 D);
```

「より小さい」の条件付き選択。

```
R0 := (A0 < B0) ? C0 : D0;
R1 := (A1 < B1) ? C1 : D1;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtd

```
Is32vec2 R = select_ge(Is32vec2 A, Is32vec2 B, Is32vec2 C,Is32vec2 D);
Iu32vec2 R = select_ge(Is32vec2 A, Is32vec2 B, Iu32vec2 C,Iu32vec2 D);
I32vec2 R = select_ge(Is32vec2 A, Is32vec2 B, I32vec2 C,I32vec2 D);
```

「等しい、またはより大きい」の条件付き選択。

```
R0 := (A0 >= B0) ? C0 : D0;
R1 := (A1 >= B1) ? C1 : D1;
```

対応する組み込み関数：_m_pandn(2)、_m_pand、_m_por、_m_pcmpgtd

```
Is32vec2 R = select_le(Is32vec2 A, Is32vec2 B, Is32vec2 C,Is32vec2 D);
Iu32vec2 R = select_le(Is32vec2 A, Is32vec2 B, Iu32vec2 C,Iu32vec2 D);
I32vec2 R = select_le(Is32vec2 A, Is32vec2 B, I32vec2 C,I32vec2 D);
```

「等しい、またはより小さい」の条件付き選択。

```
R0 := (A0 <= B0) ? C0 : D0;
R1 := (A1 <= B1) ? C1 : D1;
```

対応する組み込み関数：_m_pandn(2)、_m_pand、_m_por、_m_pcmpgtd

```
Is16vec4 R = select_gt(Is16vec4 A, Is16vec4 B, Is16vec4 C,Is16vec4 D);
Iu16vec4 R = select_gt(Is16vec4 A, Is16vec4 B, Iu16vec4 C,Iu16vec4 D);
I16vec4 R = select_gt(Is16vec4 A, Is16vec4 B, I16vec4 C,I16vec4 D);
```

「より大きい」の条件付き選択。

```
R0 := (A0 > B0) ? C0 : D0;
R1 := (A1 > B1) ? C1 : D1;
R2 := (A2 > B2) ? C2 : D2;
R3 := (A3 > B3) ? C3 : D3;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtw

```
Is16vec4 R = select_lt(Is16vec4 A, Is16vec4 B, Is16vec4 C,Is16vec4 D);
Iu16vec4 R = select_lt(Is16vec4 A, Is16vec4 B, Iu16vec4 C,Iu16vec4 D);
I16vec4 R = select_lt(Is16vec4 A, Is16vec4 B, I16vec4 C,I16vec4 D);
```

「より小さい」の条件付き選択。

```
R0 := (A0 < B0) ? C0 : D0;
R1 := (A1 < B1) ? C1 : D1;
R2 := (A2 < B2) ? C2 : D2;
R3 := (A3 < B3) ? C3 : D3;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtw

```
Is16vec4 R = select_ge(Is16vec4 A, Is16vec4 B, Is16vec4 C,Is16vec4 D);
Iu16vec4 R = select_ge(Is16vec4 A, Is16vec4 B, Iu16vec4 C,Iu16vec4 D);
I16vec4 R = select_ge(Is16vec4 A, Is16vec4 B, I16vec4 C,I16vec4 D);
```

「等しい、またはより大きい」の条件付き選択。

```
R0 := (A0 >= B0) ? C0 : D0;
R1 := (A1 >= B1) ? C1 : D1;
R2 := (A2 >= B2) ? C2 : D2;
R3 := (A3 >= B3) ? C3 : D3;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtw

```
Is16vec4 R = select_le(Is16vec4 A, Is16vec4 B, Is16vec4 C,Is16vec4 D);
Iu16vec4 R = select_le(Is16vec4 A, Is16vec4 B, Iu16vec4 C,Iu16vec4 D);
I16vec4 R = select_le(Is16vec4 A, Is16vec4 B, I16vec4 C,I16vec4 D);
```

「等しい、またはより小さい」の条件付き選択。

```
R0 := (A0 <= B0) ? C0 : D0;
R1 := (A1 <= B1) ? C1 : D1;
R2 := (A2 <= B2) ? C2 : D2;
R3 := (A3 <= B3) ? C3 : D3;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtw

```
Is8vec8 R = select_gt(Is8vec8 A, Is8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_gt(Is8vec8 A, Is8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_gt(Is8vec8 A, Is8vec8 B, I8vec8 C, I8vec8 D);
```

「より大きい」の条件付き選択。

```
R0 := (A0 > B0) ? C0 : D0;
R1 := (A1 > B1) ? C1 : D1;
R2 := (A2 > B2) ? C2 : D2;
R3 := (A3 > B3) ? C3 : D3;
R4 := (A4 > B4) ? C4 : D4;
R5 := (A5 > B5) ? C5 : D5;
R6 := (A6 > B6) ? C6 : D6;
R7 := (A7 > B7) ? C7 : D7;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtb

```
Is8vec8 R = select_lt(Is8vec8 A, Is8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_lt(Is8vec8 A, Is8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_lt(Is8vec8 A, Is8vec8 B, I8vec8 C, I8vec8 D);
```

「より小さい」の条件付き選択。.

```
R0 := (A0 < B0) ? C0 : D0;
R1 := (A1 < B1) ? C1 : D1;
R2 := (A2 < B2) ? C2 : D2;
R3 := (A3 < B3) ? C3 : D3;
```

```
R4 := (A4 < B4) ? C4 : D4;
R5 := (A5 < B5) ? C5 : D5;
R6 := (A6 < B6) ? C6 : D6;
R7 := (A7 < B7) ? C7 : D7;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtb

```
Is8vec8 R = select_ge(Is8vec8 A, Is8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_ge(Is8vec8 A, Is8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_ge(Is8vec8 A, Is8vec8 B, I8vec8 C, I8vec8 D);
```

「等しい、またはより大きい」の条件付き選択。

```
R0 := (A0 >= B0) ? C0 : D0;
R1 := (A1 >= B1) ? C1 : D1;
R2 := (A2 >= B2) ? C2 : D2;
R3 := (A3 >= B3) ? C3 : D3;
R4 := (A4 >= B4) ? C4 : D4;
R5 := (A5 >= B5) ? C5 : D5;
R6 := (A6 >= B6) ? C6 : D6;
R7 := (A7 >= B7) ? C7 : D7;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtb

```
Is8vec8 R = select_le(Is8vec8 A, Is8vec8 B, Is8vec8 C, Is8vec8 D);
Iu8vec8 R = select_le(Is8vec8 A, Is8vec8 B, Iu8vec8 C, Iu8vec8 D);
I8vec8 R = select_le(Is8vec8 A, Is8vec8 B, I8vec8 C, I8vec8 D);
```

「等しい、またはより小さい」の条件付き選択。

```
R0 := (A0 <= B0) ? C0 : D0;
R1 := (A1 <= B1) ? C1 : D1;
R2 := (A2 <= B2) ? C2 : D2;
R3 := (A3 <= B3) ? C3 : D3;
R4 := (A4 <= B4) ? C4 : D4;
R5 := (A5 <= B5) ? C5 : D5;
R6 := (A6 <= B6) ? C6 : D6;
R7 := (A7 <= B7) ? C7 : D7;
```

対応する組み込み関数：_m_pandn、_m_pand、_m_por、_m_pcmpgtb

デバッグ

デバッグ演算は、MMX テクノロジ命令の組み込み関数のいずれにも対応していません。これらはプログラムのデバッグのためにのみ提供されているものです。これらを使用するとパフォーマンスが低下することがあるので、デバッグ目的以外での使用はお勧めできません。

出力

```
cout << Is32vec2 A;
cout << Iu32vec2 A;
cout << hex << Iu32vec2 A; /* print in hex format */
```

A の 2 つの 32 ビット値が出力バッファに格納され、次の形式でプリントされます (デフォルトでは 10 進)。

```
“[1]:A1 [0]:A0”
```

対応する組み込み関数：なし

```
cout << Is16vec4 A;
cout << Iu16vec4 A;
cout << hex << Iu16vec4 A; /* print in hex format */
```

A の 4 つの 16 ビット値が出力バッファに格納され、次の形式でプリントされます (デフォルトでは 10 進)。

```
“[3]:A3 [2]:A2 [1]:A1 [0]:A0”
```

対応する組み込み関数：なし

```
cout << Is8vec8 A;
cout << Iu8vec8 A;
cout << hex << Iu8vec8 A;
/* print in hex format instead of decimal*/
```

A の 8 つの 8 ビット値が出力バッファに格納され、次の形式でプリントされます (デフォルトでは 10 進)。

```
“[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0”
```

対応する組み込み関数：なし

要素アクセス演算子

```
int R = Is32vec2 A[i];
unsigned int R = Iu32vec2 A[i];
short R = Is16vec4 A[i];
unsigned short R = Iu16vec4 A[i];
signed char R =  Is8vec8 A[i];
unsigned char R = Iu8vec8 A[i];
```

A の要素 i にアクセスし、読み取ります。DEBUG が有効になっており、ユーザが A の外部の要素へのアクセスを試みると、診断メッセージがプリントされ、プログラムはアボートします。

対応する組み込み関数：なし

要素代入演算子

```
Is32vec2 A[i] = int R;
Iu32vec2 A[i] = unsigned int R;
Is16vec4 A[i] = short R;
Iu16vec4 A[i] = unsigned short R;
Is8vec8 A[i] = signed char R;
Iu8vec8 A[i] = unsigned char R;
```

R を A の要素 i に代入します。DEBUG が有効になっており、ユーザが A の外部の要素へのアクセスを試みると、診断メッセージがプリントされ、プログラムはアボートします。

対応する組み込み関数：なし

アンパック演算子

```
I32vec2 unpack_high(I32vec2 A, I32vec2 B)
Is32vec2 unpack_high(Is32vec2 A, Is32vec2 B)
Iu32vec2 unpack_high(Iu32vec2 A, Iu32vec2 B)
```

A の上半分の 32 ビット値を、B の上半分の 32 ビット値とインタリーブします。

```
R0 = A1;
R1 = B1;
```

対応する組み込み関数：`_m_punpckhdq`

```
I16vec4 unpack_high(I16vec4 A, I16vec4 B)
Is16vec4 unpack_high(Is16vec4 A, Is16vec4 B)
Iu16vec4 unpack_high(Iu16vec4 A, Iu16vec4 B)
```

A の上半分の 2 つの 16 ビット値を、B の上半分の 2 つの 16 ビット値とインタリーブします。

```
R0 = A2;
R1 = B2;
R2 = A3;
R3 = B3;
```

対応する組み込み関数：`_m_punpckhwd`

```
I8vec8 unpack_high(I8vec8 A, I8vec8 B)
Is8vec8 unpack_high(Is8vec8 A, I8vec8 B)
Iu8vec8 unpack_high(Iu8vec8 A, I8vec8 B)
```

Aの上半分の4つの8ビット値を、Bの上半分の4つの8ビット値とインタリーブします。

```
R0 = A4;
R1 = B4;
R2 = A5;
R3 = B5;
R4 = A6;
R5 = B6;
```

```
R6 = A7;
R7 = B7;
```

対応する組み込み関数：_m_punpckhbw

```
I32vec2 unpack_low(I32vec2 A, I32vec2 B);
Is32vec2 unpack_low(Is32vec2 A, Is32vec2 B);
Iu32vec2 unpack_low(Iu32vec2 A, Iu32vec2 B);
```

A の下半分の 32 ビット値を、B の下半分の 32 ビット値とインタリーブします。

```
R0 = A0;
R1 = B0;
```

対応する組み込み関数：_m_punpckldq

```
I16vec4 unpack_low(I16vec4 A, I16vec4 B);
Is16vec4 unpack_low(Is16vec4 A, Is16vec4 B);
Iu16vec4 unpack_low(Iu16vec4 A, Iu16vec4 B);
```

A の下半分の 2 つの 16 ビット値を、B の下半分の 2 つの 16 ビット値とインタリーブします。

```
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
```

対応する組み込み関数：_m_punpcklwd

```
I8vec8 unpack_low(I8vec8 A, I8vec8 B);
Is8vec8 unpack_low(Is8vec8 A, Is8vec8 B);
Iu8vec8 unpack_low(Iu8vec8 A, Iu8vec8 B);
```

Aの下半分の4つの8ビット値を、Bの下半分の4つの8ビット値とインタリーブします。

```
R0 = A0;
R1 = B0;
R2 = A1;
```

```
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;
```

対応する組み込み関数：_m_punpcklbw

パック演算子

```
Is16vec4 pack_sat(Is32vec2 A,Is32vec2 B);
```

AおよびBの4つの32ビット値を、符号付き飽和で4つの16ビット値にパックします。

対応する組み込み関数：_m_packssdw

```
Is8vec8 pack_sat(Is16vec4 A,Is16vec4 B);
```

AおよびBの8つの16ビット値を、符号付き飽和で8つの8ビット値にパックします。

対応する組み込み関数：_m_packsswb

```
Iu8vec8 packu_sat(Is16vec4 A,Is16vec4 B);
```

AおよびBの8つの16ビット値を、符号なし飽和で8つの8ビット値にパックします。

対応する組み込み関数：_m_packuswb

クリア MMX® ステート演算子

```
void empty(void);
```

MMX レジスタを空にして、MMX 状態をクリアします。EMMS 命令組み込み関数の詳細については、『*Intel® C/C++ コンパイラ Win32* システム対応版 ユーザーズ・ガイド*』の第 12 章の「EMMS 命令を使用する際のガイドライン」の節を参照してください。

対応する組み込み関数：_m_empty

ストリーミング SIMD 拡張命令の整数組み込み関数

NOTE. *以下の機能を使用するには、*`fvec.h` *ヘッダ・ファイルをインクルードしなくてはなりません。*

```
Iu16vec4 mul_high(Iu16vec4 A, Iu16vec4 B);
```

A および B の中の 4 つの符号なしワードを掛け、32 ビットの中間結果の上位 16 ビットを返します。

対応する組み込み関数：`_m_pmulhuw`

```
Is16vec4 simd_max(Is16vec4 A, Is16vec4 B);
```

A および B の中のそれぞれの符号付き整数ワードのフィールドごとの最大値を計算します。

対応する組み込み関数：`_m_pmaxsw`

```
Is16vec4 simd_min(Is16vec4 A, Is16vec4 B);
```

A および B の中のそれぞれの符号付き整数ワードのフィールドごとの最小値を計算します。

対応する組み込み関数：`_m_pminsw`

```
Iu8vec8 simd_max(Iu8vec8 A, Iu8vec8 B);
```

A および B の中のそれぞれの符号なしバイトのフィールドごとの最大値を計算します。

対応する組み込み関数：`_m_pmaxub`

```
Iu8vec8 simd_min(Iu8vec8 A, Iu8vec8 B);
```

AおよびBの中のそれぞれの符号なしバイトのフィールドごとの最小値を計算します。

対応する組み込み関数：_m_pminub

```
int move_mask(I8vec8 A);
```

A の中のバイトの最上位ビットから 8 ビットのマスクを作成します。

対応する組み込み関数：_m_pmovmskb

```
void mask_move(I8vec8 A, I8vec8 B, signed char *p);
```

A のバイトフィールドを条件的にアドレス p に格納します。セレクタ B の各バイトの上位ビットが、A の中の対応するバイトを格納するかどうかを決定します。

対応する組み込み関数：_m_maskmovq

```
void store_nta(__m64 *p, M64 A);
```

A のデータを、キャッシュに影響を与えずにアドレス p に格納します。A は任意の Ivec 型です。

対応する組み込み関数：_m_stream_pi

Fvec と Ivec の間の変換

```
int F32vec4ToInt(F32vec4 A)
```

a の下位の浮動小数点値を、切り捨てモードで 32 ビット整数に変換します。

```
r := (int)A0
```

```
Is32vec2 F32vec4ToIs32vec2 (F32vec4 A)
```

A の下位の 2 つの浮動小数点値を、切り捨てモードで 32 ビット整数に変換し、パック形式の整数として返します。

```
r0 := (int)A0
r1 := (int)A1
```

```
F32vec4 IntToF32vec4(F32vec4 A, int B)
```

32 ビット整数値 B を浮動小数点値に変換します。上位 3 つの浮動小数点値は A から渡されます。

```
r0 := (float)B
r1 := A1 ; r2 := A2 ; r3 := A3
```

```
F32vec4 Is32vec2ToF32vec4(F32vec4 A, Is32vec2 B)
```

B にパック形式で含まれている 2 つの 32 ビット整数値を、2 つの浮動小数点値に変換します。上位 2 つの浮動小数点値は A から渡されます。

```
r0 := (float)B0
r1 := (float)B1
r2 := A2
r3 := A3
```

FVec クラス 4

Fvec クラス、F32vec および F32vec1 は、ストリーミング SIMD 拡張命令の SIMD 浮動小数点とのインターフェイスを提供します。クラス仕様は次のとおりです。

```
F32vec4 A(float z, float y, float x, float w);
F32vec1 B(float w);
```

F32vec4 クラスは、4 つのパックド 32 ビット単精度浮動小数点数を表します。F32vec1 は、1 つのスカラー 32 ビット単精度浮動小数点数を表します。

パックド浮動小数点の入力値は右から左の順で表現されます。最下位の浮動小数点値が一番右、最上位の浮動小数点値が一番左です。F32vec1 クラスは最下位の値のみを使用し、上位 3 値を無視します。

データ・アライメント

ストリーミング SIMD 拡張命令を使用するメモリ操作は、できるだけ 16 バイトでアライメントされたデータに対して行うようにすべきです。

F32vec4 オブジェクト変数は、デフォルトで適切にアライメントされています。浮動小数点配列は自動的にはアライメントされません。16 バイト境界でアライメントしたい場合は、アライメント _declspec を使用します。

```
__declspec( align(16) ) float A[4];
```

変換

```
__m128 mm = A & B;    /* where A,B are F32vec4 object variables */
__m128 mm = A & B;    /* where A,B are F32vec1 object variables */
```

F32vec4 および F32vec1 オブジェクト変数は、暗黙的に __m128 データ型に変換できます。たとえば、F32vec4 または F32vec1 オブジェクト変数に対して実行した計算の結果を __m128 データ型に代入できます。

コンストラクタ

Fvec クラスを使用すると、F32vec4 または F32vec1 オブジェクトを何通りかの方法で作成できます。

```
F32vec4 A;
F32vec1 B;
```

デフォルト・コンストラクタは、初期化されていない単精度浮動小数点値を 4 つ含むオブジェクト A および B を作成します。F32vec1 クラスの関数は、F32vec1 オブジェクトの最下位の浮動小数点値だけを使用します。

対応する組み込み関数：なし

```
F32vec4 A(__m128 mm);
F32vec1 B(__m128 mm);
```

オブジェクト A および B の 4 つの単精度浮動小数点値を __m128 データで初期化します。F32vec1 クラスの関数は、F32vec1 オブジェクトの最下位の浮動小数点値だけを使用します。

```
F32vec4 A(float f3, float f2, float f1, float f0);
```

オブジェクト A の 4 つの単精度浮動小数点値を、浮動小数点値で右から左の順に初期化します。

```
A0 := f0;
A1 := f1;
A2 := f2;
A3 := f3;
```

対応する組み込み関数：mm_set_ps

```
F32vec4 A = F32vec4(float f3, float f2 ,float f1 ,float f0);
```

A に含まれている 4 つの単精度浮動小数点値を、浮動小数点値で右から左の順に初期化します。

```
A0 := f0;
A1 := f1;
A2 := f2;
A3 := f3;
```

対応する組み込み関数：mm_set_ps

```
F32vec4 A(float f0);
```

A に含まれている 4 つの単精度浮動小数点値を、同じ浮動小数点値で初期化します。

```
A0 := f0;
A1 := f0;
A2 := f0;
A3 := f0;
```

対応する組み込み関数：_mm_set_ps1

```
F32vec1 B(float f0);
```

オブジェクト B の最下位の単精度浮動小数点値を、浮動小数点値 f0 で初期化します。上位 3 つの浮動小数点値はゼロに設定されます。F32vec1 クラスの関数は、最下位の浮動小数点値だけを使用します。

```
B0 := f0;
```

対応する組み込み関数：_mm_set_ss

```
F32vec1 B(int i);
```

整数 I を浮動小数点値に変換し、この浮動小数点値で、B の最下位の単精度浮動小数点値を初期化します。B の上位 3 つの浮動小数点値は未定義です。

対応する組み込み関数：_mm_ct_si2ss

算術演算

加算演算子

```
F32vec4 R = F32vec4 A + F32vec4 B;
```

A と B の単精度浮動小数点値を加算します (各 4)。

```
R0 := A0 + B0
R1 := A1 + B1
R2 := A2 + B2
R3 := A3 + B3
```

対応する組み込み関数：_mm_add_ps

```
F32vec1 R = F32vec1 A + F32vec1 B;
```

A と B の最下位単精度浮動小数点値を加算します。

```
R0 := A0 + B0
```

対応する組み込み関数：_mm_add_ss

```
F32vec4 R += F32vec4 A;
```

A と R の単精度浮動小数点値を加算します (各 4)。

```
R0 := R0 + A0
R1 := R1 + A1
R2 := R2 + A2
R3 := R3 + A3
```

対応する組み込み関数：_mm_add_ps

```
F32vec1 R += F32vec1 A;
```

A と R の最下位単精度浮動小数点値を加算します。

```
R0 := R0 + A0
```

対応する組み込み関数：_mm_add_ss

減算演算子

```
F32vec4 R = F32vec4 A - F32vec4 B;
```

A の単精度浮動小数点値から B の単精度浮動小数点値を減算します (各 4)。

```
R0 := A0 - B0
R1 := A1 - B1
R2 := A2 - B2
R3 := A3 - B3
```

対応する組み込み関数：_mm_sub_ps

```
F32vec1 R = F32vec1 A - F32vec1 B;
```

A の最下位浮動小数点値から B の最下位単精度浮動小数点値を減算します。

```
R0 := A0 - B0
```

対応する組み込み関数：_mm_sub_ss

```
F32vec4 R -= F32vec4 A;
```

R から A の単精度浮動小数点値を減算します (各 4)。

```
R0 := R0 - A0
R1 := R1 - A1
R2 := R2 - A2
R3 := R3 - A3
```

対応する組み込み関数：_mm_sub_ps

```
F32vec1 R -= F32vec1 A;
```

R の最下位浮動小数点値から A の最下位単精度浮動小数点値を減算します。

```
R0 := R0 - A0
```

対応する組み込み関数：_mm_sub_ss

乗算演算子

```
F32vec4 R = F32vec4 A * F32vec4 B;
```

A と B の単精度浮動小数点値を乗算します (各 4)。

```
R0 := A0 * B0
R1 := A1 * B1
R2 := A2 * B2
R3 := A3 * B3
```

対応する組み込み関数：_mm_mul_ps

```
F32vec1 R = F32vec1 A * F32vec1 B;
```

A と B の最下位単精度浮動小数点値を乗算します。

```
R0 := A0 * B0
```

対応する組み込み関数：_mm_mul_ss

```
F32vec4 R *= F32vec4 A;
```

A と R の単精度浮動小数点値を乗算します (各 4)。

```
R0 := R0 * A0
R1 := R1 * A1
R2 := R2 * A2
R3 := R3 * A3
```

対応する組み込み関数：_mm_mul_ps

```
F32vec4 R *= F32vec4 A;
```

A と R の最下位浮動小数点値を乗算します。

```
R0 := R0 * A0
```

対応する組み込み関数：_mm_mul_ss

除算演算子

```
F32vec4 R = F32vec4 A / F32vec4 B;
```

A を B の単精度浮動小数点値で除算します (各 4)。

```
R0 := A0 / B0
R1 := A1 / B1
R2 := A2 / B2
R3 := A3 / B3
```

対応する組み込み関数：_mm_div_ps

```
F32vec1 R = F32vec1 A / F32vec1 B;
```

A の最下位単精度浮動小数点値を B の最下位浮動小数点値で除算します。

```
R0 := A0 / B0
```

対応する組み込み関数：_mm_div_ss

```
F32vec4 R /= F32vec4 A;
```

R を A の単精度浮動小数点値で除算します (各 4)。

```
R0 := R0 / A0
R1 := R1 / A1
R2 := R2 / A2
R3 := R3 / A3
```

対応する組み込み関数：_mm_div_ps

```
F32vec1 R /= F32vec1 A;
```

R の最下位単精度浮動小数点値を A の最下位浮動小数点値で除算します。

```
R0 := R0 / A0
```

対応する組み込み関数：_mm_div_ss

平方根演算子

```
F32vec4 R = sqrt(F32vec4 A);
```

A の 4 つの単精度浮動小数点値の平方根を計算します。

```
R0 = sqrt(A0);
R1 = sqrt(A1);
R2 = sqrt(A2);
R3 = sqrt(A3);
```

対応する組み込み関数：_mm_sqrt_ps

```
F32vec1 R = sqrt(F32vec1 A);
```

A の最下位単精度浮動小数点値の平方根を計算します。

```
R0 = sqrt(A0);
```

対応する組み込み関数：_mm_sqrt_ss

逆数演算子

```
F32vec4 R = rcp(F32vec4 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します。

```
R0 = recip(A0);
R1 = recip(A1);
R2 = recip(A2);
R3 = recip(A3);
```

対応する組み込み関数：_mm_rcp_ps

```
F32vec1 R = rcp(F32vec1 A);
```

A の最下位単精度浮動小数点値の逆数の近似値を計算します。

```
R0 = recip(A0);
```

対応する組み込み関数：_mm_rcp_ss

```
F32vec4 R = rsqrt(F32vec4 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します。

```
R0 = recip(sqrt(A0));
R1 = recip(sqrt(A1));
R2 = recip(sqrt(A2));
R3 = recip(sqrt(A3));
```

対応する組み込み関数：_mm_rsqrt_ps

```
F32vec1 R = rsqrt(F32vec1 A);
```

A の最下位単精度浮動小数点値の平方根の逆数の近似値を計算します。

```
R0 = recip(sqrt(A0));
```

対応する組み込み関数：_mm_rsqrt_ss

```
F32vec4 R = rcp_nr(F32vec4 A);
```

A の 4 つの単精度浮動小数点値の Newton-Raphson 逆数を計算します。

```
R0 = rcp_nr(A0);
R1 = rcp_nr(A1);
R2 = rcp_nr(A2);
R3 = rcp_nr(A3);
```

対応する組み込み関数：_mm_sub_ps、_mm_add_ps、_mm_mul_ps、_mm_rcp_ps

```
F32vec1 R = rcp_nr(F32vec1 A);
```

A の最下位単精度浮動小数点値の Newton-Raphson 逆数を計算します。

```
R0 = rcp_nr(A0);
```

対応する組み込み関数：_mm_sub_ss、_mm_add_ss、_mm_mul_ss、_mm_rcp_ss

```
F32vec4 R = rsqrt_nr(F32vec4 A);
```

A の 4 つの単精度浮動小数点値の Newton-Raphson 平方根逆数を計算します。

```
R0 = rsqrt_nr(A0);
R1 = rsqrt_nr(A1);
R2 = rsqrt_nr(A2);
R3 = rsqrt_nr(A3);
```

対応する組み込み関数：_mm_rsqrt_ps、_mm_sub_ps、_mm_mul_ps、_mm_rsqrt_ps

```
F32vec1 R = rsqrt_nr(F32vec1 A);
```

A の最下位単精度浮動小数点値の Newton-Raphson 平方根逆数を計算します。

```
R0 = rsqrt_nr(A0);
```

対応する組み込み関数：_mm_rsqrt_ss、_mm_sub_ss、_mm_mul_ss、_mm_rsqrt_ss

水平加算演算子

```
float f = add_horizontal(F32vec4 A);
```

F32vec4 クラス・オブジェクトの 4 つのパックド浮動小数点数の総和を計算します。

```
f := A0 + A1 + A2 + A3;
```

対応する組み込み関数：_mm_add_ss、_mm_shuffle_ps

論理演算子

ビット単位の加算演算子

```
F32vec4 R = F32vec4 A & F32vec4 B;
```

値 A と B のビット単位の論理積 (AND) を求めます。

対応する組み込み関数：_mm_and_ps

```
F32vec1 R = F32vec1 A & F32vec1 B;
```

値 A と B のビット単位の論理積 (AND) を求めます。下位 32 ビットだけが使用されます。

対応する組み込み関数：_mm_and_ps (対応するスカラー組み込み関数なし)

```
F32vec4 R &= F32vec4 A;
```

値 R と A のビット単位の論理積 (AND) を求めます。
対応する組み込み関数：_mm_and_ps

```
F32vec1 R &= F32vec1 A;
```

値 R と A のビット単位の論理積 (AND) を求めます。下位 32 ビットだけが使用されます。
対応する組み込み関数：_mm_and_ps (対応するスカラー組み込み関数なし)

ビット単位の OR 演算子

```
F32vec4 R = F32vec4 A | F32vec4 B;
```

値 A と B のビット単位の論理和 (OR) を求めます。
対応する組み込み関数：_mm_or_ps

```
F32vec1 R = F32vec1 A | F32vec1 B;
```

値 A と B のビット単位の論理和 (OR) を求めます。下位 32 ビットだけが使用されます。
対応する組み込み関数：_mm_or_ps (対応するスカラー組み込み関数なし)

```
F32vec4 R |= F32vec4 A;
```

値 R と A のビット単位の論理和 (OR) を求めます。
対応する組み込み関数：_mm_or_ps

```
F32vec1 R |= F32vec1 A;
```

値 R と A のビット単位の論理和 (OR) を求めます。下位 32 ビットだけが使用されます。
対応する組み込み関数：_mm_or_ps (対応するスカラー組み込み関数なし)

ビット単位の XOR 演算子

```
F32vec4 R = F32vec4 A ^ F32vec4 B;
```

値 A と B のビット単位の排他的論理和 (XOR) を求めます。

対応する組み込み関数：_mm_xor_ps

```
F32vec1 R = F32vec1 A ^ F32vec1 B;
```

値 A と B のビット単位の排他的論理和 (XOR) を求めます。下位 32 ビットだけが使用されます。

対応する組み込み関数：_mm_xor_ps (対応するスカラー組み込み関数なし)

```
F32vec4 R ^= F32vec4 A;
```

値 R と A のビット単位の排他的論理和 (XOR) を求めます。

対応する組み込み関数：_mm_xor_ps

```
F32vec1 R ^= F32vec1 A;
```

値 R と A のビット単位の排他的論理和 (XOR) を求めます。下位 32 ビットだけが使用されます。

対応する組み込み関数：_mm_xor_ps (対応するスカラー組み込み関数なし)

最小および最大演算子

```
F32vec4 R = simd_min(F32vec4 A, F32vec4 B)
```

A と B の 4 つの単精度浮動小数点値の最小値を計算します。

```
R0 := min(A0,B0);
R1 := min(A1,B1);
R2 := min(A2,B2);
R3 := min(A3,B3);
```

対応する組み込み関数：_mm_min_ps

```
F32vec1 R = simd_min(F32vec1 A, F32vec1 B)
```

A と B の最下位単精度浮動小数点値の最小値を計算します。

```
R0 := min(A0,B0);
```

対応する組み込み関数：_mm_min_ss

```
F32vec4 simd_max(F32vec4 A, F32vec4 B)
```

A と B の 4 つの単精度浮動小数点値の最大値を計算します。

```
R0 := max(A0,B0);
R1 := max(A1,B1);
R2 := max(A2,B2);
R3 := max(A3,B3);
```

対応する組み込み関数：_mm_max_ps

```
F32vec1 simd_max(F32vec1 A, F32vec1 B)
```

A と B の最下位単精度浮動小数点値の最大値を計算します。

```
R0 := max(A0,B0);
```

対応する組み込み関数：_mm_max_ss

比較演算

各比較関数は、A と B の単精度浮動小数点値の比較を実行します。2 つの F32vec4 オブジェクトの間の比較では、1 つの F32vec4 オブジェクトが返されます。2 つの F32vec1 オブジェクトの間の比較では、1 つの F32vec1 オブジェクトが返されます。各単精度浮動小数点について、比較が真の場合はマスクが 0xffffffff に、偽の場合は 0x00000000 に設定されます。

「等しい」かどうかの比較

```
F32vec4 R = cmpeq(F32vec4 A, F32vec4 B);
```

比較して、「等しい」かどうかを調べます。

```
R0 := (A0 == B0) ? 0xffffffff : 0x0;
R1 := (A1 == B1) ? 0xffffffff : 0x0;
R2 := (A2 == B2) ? 0xffffffff : 0x0;
R3 := (A3 == B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpeq_ps`

```
F32vec1 R = cmpeq(F32vec1 A, F32vec1 B);
```

比較して、「等しい」かどうかを調べます。

```
R0 := (A0 == B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpeq_ss`

「等しくない」かどうかの比較

```
F32vec4 R = cmpneq(F32vec4 A, F32vec4 B);
```

比較して、「等しくない」かどうかを調べます。

```
R0 := !(A0 == B0) ? 0xffffffff : 0x0;
R1 := !(A1 == B1) ? 0xffffffff : 0x0;
R2 := !(A2 == B2) ? 0xffffffff : 0x0;
R3 := !(A3 == B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpneq_ps`

```
F32vec1 R = cmpneq(F32vec1 A, F32vec1 B);
```

比較して、「等しくない」かどうかを調べます。

```
R0 := !(A0 == B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpneq_ss`

「より小さい」かどうかの比較

```
F32vec4 R = cmplt(F32vec4 A, F32vec4 B);
```

比較して、「より小さい」かどうかを調べます。

```
R0 := (A0 < B0) ? 0xffffffff : 0x0;
R1 := (A1 < B1) ? 0xffffffff : 0x0;
R2 := (A2 < B2) ? 0xffffffff : 0x0;
R3 := (A3 < B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmplt_ps`

```
F32vec1 R = cmplt(F32vec1 A, F32vec1 B);
```

比較して、「より小さい」かどうかを調べます。

```
R0 := (A0 < B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmplt_ss`

「以下」かどうかの比較

```
F32vec4 R = cmple(F32vec4 A, F32vec4 B);
```

比較して、「等しい、またはより小さい」かどうかを調べます。

```
R0 := (A0 <= B0) ? 0xffffffff : 0x0;
R1 := (A1 <= B1) ? 0xffffffff : 0x0;
R2 := (A2 <= B2) ? 0xffffffff : 0x0;
R3 := (A3 <= B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmple_ps`

```
F32vec1 R = cmple(F32vec1 A, F32vec1 B);
```

比較して、「等しい、またはより小さい」かどうかを調べます。

```
R0 := (A0 <= B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmple_ss`

「より大きい」かどうかの比較

```
F32vec4 R = cmpgt(F32vec4 A, F32vec4 B);
```

比較して、「より大きい」かどうかを調べます。

```
R0 := (A0 > B0) ? 0xffffffff : 0x0;
R1 := (A1 > B1) ? 0xffffffff : 0x0;
R2 := (A2 > B2) ? 0xffffffff : 0x0;
R3 := (A3 > B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpgt_ps`

```
F32vec1 R = cmpgt(F32vec1 A, F32vec1 B);
```

比較して、「より大きい」かどうかを調べます。

```
R0 := (A0 > B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpgt_ss`

「以上」かどうかの比較

```
F32vec4 R = cmpge(F32vec4 A, F32vec4 B);
```

比較して、「等しい、またはより大きい」かどうかを調べます。

```
R0 := (A0 >= B0) ? 0xffffffff : 0x0;
R1 := (A1 >= B1) ? 0xffffffff : 0x0;
R2 := (A2 >= B2) ? 0xffffffff : 0x0;
R3 := (A3 >= B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpge_ps`

```
F32vec1 R = cmpge(F32vec1 A, F32vec1 B);
```

比較して、「等しい、またはより大きい」かどうかを調べます。

```
R0 := (A0 >= B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpge_ss`

「より小さい」の否定が成り立つかどうかの比較

```
F32vec4 R = cmpnlt(F32vec4 A, F32vec4 B);
```

比較して、「より小さい」の否定が成り立つかどうか調べます。

```
R0 := !(A0 < B0) ? 0xffffffff : 0x0;
R1 := !(A1 < B1) ? 0xffffffff : 0x0;
R2 := !(A2 < B2) ? 0xffffffff : 0x0;
R3 := !(A3 < B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnlt_ps`

```
F32vec1 R = cmpnlt(F32vec1 A, F32vec1 B);
```

比較して、「より小さい」の否定が成り立つかどうか調べます。

```
R0 := !(A0 < B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnlt_ss`

「以下」の否定が成り立つかどうかの比較

```
F32vec4 R = cmpnle(F32vec4 A, F32vec4 B);
```

比較して、「等しい、またはより小さい」の否定が成り立つかどうか調べます。

```
R0 := !(A0 <= B0) ? 0xffffffff : 0x0;
R1 := !(A1 <= B1) ? 0xffffffff : 0x0;
R2 := !(A2 <= B2) ? 0xffffffff : 0x0;
R3 := !(A3 <= B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnle_ps`

```
F32vec1 R = cmpnle(F32vec1 A, F32vec1 B);
```

比較して、「等しい、またはより小さい」の否定が成り立つかどうか調べます。

```
R0 := !(A0 <= B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnle_ss`

「より大きい」の否定が成り立つかどうかの比較

```
F32vec4 R = cmpngt(F32vec4 A, F32vec4 B);
```

比較して、「より大きい」の否定が成り立つかどうかを調べます。

```
R0 := !(A0 > B0) ? 0xffffffff : 0x0;
R1 := !(A1 > B1) ? 0xffffffff : 0x0;
R2 := !(A2 > B2) ? 0xffffffff : 0x0;
R3 := !(A3 > B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpngt_ps`

```
F32vec1 R = cmpngt(F32vec1 A, F32vec1 B);
```

比較して、「より大きい」の否定が成り立つかどうかを調べます。

```
R0 := !(A0 > B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpngt_ss`

「以上」の否定が成り立つかどうかの比較

```
F32vec4 R = cmpnge(F32vec4 A, F32vec4 B);
```

比較して、「等しい、またはより大きい」の否定が成り立つかどうかを調べます。

```
R0 := !(A0 >= B0) ? 0xffffffff : 0x0;
R1 := !(A1 >= B1) ? 0xffffffff : 0x0;
R2 := !(A2 >= B2) ? 0xffffffff : 0x0;
R3 := !(A3 >= B3) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnge_ps`

```
F32vec1 R = cmpnge(F32vec1 A, F32vec1 B);
```

比較して、「等しい、またはより大きい」の否定が成り立つかどうかを調べます。

```
R0 := !(A0 >= B0) ? 0xffffffff : 0x0;
```

対応する組み込み関数：`_mm_cmpnge_ss`

条件付き選択演算

「等しい」場合の条件付き選択

```
F32vec4 R = select_eq(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しい」場合の条件付き選択。

```
R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpeq_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_eq(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しい」場合の条件付き選択。

```
R0 := (A0 == B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpeq_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「等しくない」場合の条件付き選択

```
F32vec4 R = select_neq(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しくない」場合の条件付き選択。

```
R0 := !(A0 == B0) ? C0 : D0;
R1 := !(A1 == B1) ? C1 : D1;
R2 := !(A2 == B2) ? C2 : D2;
R3 := !(A3 == B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpneq_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_neq(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しくない」場合の条件付き選択。

```
R0 := !(A0 == B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpneq_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「より小さい」場合の条件付き選択

```
F32vec4 R = select_lt(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「より小さい」場合の条件付き選択。

```
R0 := (A0 < B0) ? C0 : D0;
R1 := (A1 < B1) ? C1 : D1;
R2 := (A2 < B2) ? C2 : D2;
R3 := (A3 < B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmplt_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_lt(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「より小さい」場合の条件付き選択。

```
R0 := (A0 < B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmplt_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「以下」場合の条件付き選択

```
F32vec4 R = select_le(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しい、またはより小さい」場合の条件付き選択。

```
R0 := (A0 <= B0) ? C0 : D0;
R1 := (A1 <= B1) ? C1 : D1;
R2 := (A2 <= B2) ? C2 : D2;
R3 := (A3 <= B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmplte_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_le(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しい、またはより小さい」場合の条件付き選択。

```
R0 := (A0 <= B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmplte_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「より大きい」場合の条件付き選択

```
F32vec4 R = select_gt(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「より大きい」場合の条件付き選択。

```
R0 := (A0 > B0) ? C0 : D0;
R1 := (A1 > B1) ? C1 : D1;
R2 := (A2 > B2) ? C2 : D2;
R3 := (A3 > B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpgt_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_gt(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「より大きい」場合の条件付き選択。

```
R0 := (A0 > B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpgt_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「以上」場合の条件付き選択

```
F32vec4 R = select_ge(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しい、またはより大きい」場合の条件付き選択。

```
R0 := (A0 >= B0) ? C0 : D0;
R1 := (A1 >= B1) ? C1 : D1;
R2 := (A2 >= B2) ? C2 : D2;
R3 := (A3 >= B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpge_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_ge(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しい、またはより大きい」場合の条件付き選択。

```
R0 := (A0 >= B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpge_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「より小さい」の否定の場合の条件付き選択

```
F32vec4 R = select_nlt(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「より小さい」の否定の場合の条件付き選択。

```
R0 := !(A0 < B0) ? C0 : D0;
R1 := !(A1 < B1) ? C1 : D1;
R2 := !(A2 < B2) ? C2 : D2;
R3 := !(A3 < B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpnlt_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_nlt(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「より小さい」の否定の場合の条件付き選択。

```
R0 := !(A0 < B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpnlt_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「以下」の否定の場合の条件付き選択

```
F32vec4 R = select_nle(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しい、またはより小さい」の否定の場合の条件付き選択。

```
R0 := !(A0 <= B0) ? C0 : D0;
R1 := !(A1 <= B1) ? C1 : D1;
R2 := !(A2 <= B2) ? C2 : D2;
R3 := !(A3 <= B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpnle_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_nle(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しい、またはより小さい」の否定の場合の条件付き選択。

```
R0 := !(A0 <= B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpnle_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「より大きい」の否定の場合の条件付き選択

```
F32vec4 R = select_ngt(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「より大きい」の否定の場合の条件付き選択。

```
R0 := !(A0 > B0) ? C0 : D0;
R1 := !(A1 > B1) ? C1 : D1;
R2 := !(A2 > B2) ? C2 : D2;
R3 := !(A3 > B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpngt_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_ngt(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「より大きい」の否定の場合の条件付き選択。

```
R0 := !(A0 > B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpngt_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

「以上」の否定の場合の条件付き選択

```
F32vec4 R = select_nge(F32vec4 A, F32vec4 B, F32vec4 C, F32vec4 D);
```

「等しい、またはより大きい」の否定の場合の条件付き選択。

```
R0 := !(A0 >= B0) ? C0 : D0;
R1 := !(A1 >= B1) ? C1 : D1;
R2 := !(A2 >= B2) ? C2 : D2;
R3 := !(A3 >= B3) ? C3 : D3;
```

対応する組み込み関数：_mm_cmpnge_ps、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

```
F32vec1 R = select_nge(F32vec1 A, F32vec1 B, F32vec1 C, F32vec1 D);
```

「等しい、またはより大きい」の否定の場合の条件付き選択。

```
R0 := !(A0 >= B0) ? C0 : D0;
```

対応する組み込み関数：_mm_cmpnge_ss、_mm_andnot_ps、_mm_or_ps、_mm_and_ps

キャッシング可能性サポート演算

```
void store_nta(float *p, F32vec4 A);
```

A の 4 つの単精度浮動小数点値を (非一時的に) 格納します。16 バイト境界でのアドレスのアライメントが必要です。

対応する組み込み関数：_mm_stream_ps

デバッグ

デバッグ演算は、MMX® テクノロジ命令またはストリーミング SIMD 拡張命令のコンパイラ組み込み関数のいずれにも対応していません。これらはプログラムのデバッグのためにのみ提供されているものです。これらを使用するとパフォーマンスが低下することがあるので、デバッグ目的以外での使用はお勧めできません。

出力演算

```
cout << F32vec4 A;
```

A の 4 つの単精度浮動小数点値を出力バッファに格納し、次のような 10 進数形式でプリントします。

```
“[3]:A3 [2]:A2 [1]:A1 [0]:A0”
```

対応する組み込み関数：なし

```
cout << F32vec1 A;
```

A の最下位単精度浮動小数点値を出力バッファに格納し、プリントします。

対応する組み込み関数：なし

フィールドアクセス演算

```
float f = F32vec4 A[int i]
```

A の 4 つの単精度浮動小数点値のうちの 1 つを、対応する浮動小数点フィールドを変更することなく読み取ります。int i の有効値は、0、1、2、3 です。

```
float f = F32vec4 A[2];
```

DEBUG がイネーブルで int i の値が 0、1、2、3 のいずれでもない場合、診断メッセージが出力され、プログラムはアボートします。

対応する組み込み関数：なし

要素代入演算

```
F32vec4 A[int i] = float f;
```

A の 4 つの単精度浮動小数点値の 1 つを変更します。int i の有効値は、0、1、2、3 です。

```
F32vec4 A[3] = float f;
```

DEBUG がイネーブルで int i の値が 0、1、2、3 のいずれでもない場合、診断メッセージが出力され、プログラムはアボートします。

対応する組み込み関数：なし

ロードおよび格納演算子

```
void loadu(F32vec4 A, float *p)
```

4 つの単精度浮動小数点値をロードし、これらを A の 4 つの浮動小数点値にコピーします。アライメントについての条件はありません。

対応する組み込み関数：mm_loadu_ps

```
void storeu(float *p, F32vec4 A);
```

A の 4 つの FP SP 値を格納します。アライメントについての条件はありません。

対応する組み込み関数：mm_storeu_ps

アンパック演算子

```
F32vec4 R = unpack_low(F32vec4 A, F32vec4 B);
```

A と B の下位 2 つの単精度浮動小数点値を選択し、インタリーブします。
対応する組み込み関数：`mm_unpacklo_ps(a, b)`

```
F32vec4 R = unpack_high(F32vec4 A, F32vec4 B);
```

A と B の上位 2 つの単精度浮動小数点値を選択し、インタリーブします。
対応する組み込み関数：`mm_unpackhi_ps(a, b)`

マスク移動 (move mask) 演算子

```
int i = move_mask(F32vec4 A)
```

A の 4 つの単精度浮動小数点値の最上位ビットから 4 ビットのマスクを作成します。

```
i := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)<<0
```

対応する組み込み関数：`mm_movemask_ps`

クイック・リファレンス A

以下の表 A-1 ～表 A-9 に、クラス、機能、および C++ SIMD 命令クラスの各クラスに対応する組み込み関数を示します。表 A-10 には、C++ クラスには含まれないすべての Intel® C/C++ コンパイラの組み込み関数を示します。

表 A-1 **クラス** I64vec1

クラス	機能	組み込み関数
I64vec1	&, &=	_m_pand
I64vec1	|, |=	_m_por
I64vec1	^, ^=	_m_pxor
I64vec1	Andnot	_m_pandn
I64vec1	<<, <<=	_m_psllq
I64vec1	<<, <<=	_m_psllqi
I64vec1	>>, >>=	_m_psrlq
I64vec1	>>, >>=	_m_psrlqi

表 A-2 クラス Is32vec2

クラス	機能	組み込み関数
Is32vec2	&, &=	_m_pand
Is32vec2	|, |=	_m_por
Is32vec2	^, ^=	_m_pxor
Is32vec2	Andnot	_m_pandn
Is32vec2	+, +=	_m_paddd
Is32vec2	-, -=	_m_psubd
Is32vec2	<<, <<=	_m_pslld
Is32vec2	<<, <<=	_m_pslldi
Is32vec2	>>, >>=	_m_psrad
Is32vec2	>>, >>=	_m_psradi
Is32vec2	cout <<	なし
Is32vec2	[]	なし
Is32vec2	Cmpeq	_m_pcmpeqd
Is32vec2	Cmpneq	_m_pcmpeqd, _m_pandn
Is32vec2	Cmpgt	_m_pcmpgtd
Is32vec2	Cmplt	_m_pcmpgtd, _m_pandn
Is32vec2	Cmpge	_m_pcmpgtd, _m_pandn
Is32vec2	Unpack_low	_m_punpckldq
Is32vec2	Unpack_high	_m_punpckhdq
Is32vec2	Pack_sat	_m_packssdw
Is32vec2	Select_eq	_m_pand, _m_por, _m_pcmpeqd, _m_pandn
Is32vec2	Select_neq	_m_pand, _m_por, _m_pcmpeqd, _m_pandn
Is32vec2	Select_gt	_m_pand, _m_por, _m_pcmpgtd, _m_pandn
Is32vec2	Select_lt	_m_pand, _m_por, _m_pcmpgtd, _m_pandn
Is32vec2	Select_ge	_m_pand, _m_por, _m_pcmpgtd, _m_pandn

表 A-3 **クラス** Iu32vec2

クラス	機能	組み込み関数
Iu32vec2	&, &=	_m_pand
Iu32vec2	|, |=	_m_por
Iu32vec2	^, ^=	_m_pxor
Iu32vec2	andnot	_m_pandn
Iu32vec2	+, +=	_m_paddd
Iu32vec2	-, -=	_m_psubd
Iu32vec2	<<, <<=	_m_pslld
Iu32vec2	<<, <<=	_m_pslldi
Iu32vec2	>>, >>=	_m_psrld
Iu32vec2	>>, >>=	_m_psrldi
Iu32vec2	cout <<	なし
Iu32vec2	[]	なし
Iu32vec2	cmpeq	_m_pcmpeqd
Iu32vec2	cmpneq	_m_pcmpeqd, _m_pandn
Iu32vec2	unpack_low	_m_punpckldq
Iu32vec2	unpack_high	_m_punpckhdq
Iu32vec2	select_eq	_m_pand, _m_por, _m_pcmpeqd, _m_pandn
Iu32vec2	select_neq	_m_pand, _m_por, _m_pcmpeqd, _m_pandn

表 A-4 **クラス** Is16vec4

クラス	機能	組み込み関数
Is16vec4	&, &=	_m_pand
Is16vec4	|, |=	_m_por
Is16vec4	^, ^=	_m_pxor
Is16vec4	andnot	_m_pandn
Is16vec4	+, +=	_m_paddw
Is16vec4	-, -=	_m_psubw
Is16vec4	*,*=	_m_pmullw
Is16vec4	<<, <<=	_m_psllw
Is16vec4	<<, <<=	_m_psllwi
Is16vec4	>>, >>=	_m_psraw
Is16vec4	>>, >>=	_m_psrawi
Is16vec4	cout <<	なし
Is16vec4	[]	なし
Is16vec4	cmpeq	_m_pcmpeqw
Is16vec4	cmpneq	_m_pcmpeqw, _m_pandn
Is16vec4	cmpgt	_m_pcmpgtw
Is16vec4	cmplt	_m_pcmpgtw, _m_pandn
Is16vec4	cmpge	_m_pcmpgtw, _m_pandn
Is16vec4	unpack_low	_m_punpcklwd
Is16vec4	unpack_high	_m_punpckhwd
Is16vec4	pack_sat	_m_packsswd
Is16vec4	packu_sat	_m_packuswd
Is16vec4	sat_add	_m_paddsw
Is16vec4	sat_sub	_m_psubsw
Is16vec4	mul_high	_m_pmulhw
Is16vec4	mul_add	_m_pmaddwd

（**続く**）

表 A-4 クラス Is16vec4 (続き)

クラス	機能	組み込み関数
Is16vec4	select_eq	_m_pand, _m_por, _m_pcmpeqw, _m_pandn
Is16vec4	select_neq	_m_pand, _m_por, _m_pcmpeqw, _m_pandn
Is16vec4	select_gt	_m_pand, _m_por, _m_pcmpgtw, _m_pandn
Is16vec4	select_lt	_m_pand, _m_por, _m_pcmpgtw, _m_pandn
Is16vec4	select_ge	_m_pand, _m_por, _m_pcmpgtw, _m_pandn

表 A-5 クラス Iu16vec4

クラス	機能	組み込み関数
Iu16vec4	&, &=	_m_pand
Iu16vec4	|, |=	_m_por
Iu16vec4	^, ^=	_m_pxor
Iu16vec4	andnot	_m_pandn
Iu16vec4	+, +=	_m_paddw
Iu16vec4	-, -=	_m_psubw
Iu16vec4	*, *=	_m_pmullw
Iu16vec4	<<, <<=	_m_psllw
Iu16vec4	<<, <<=	_m_psllwi
Iu16vec4	>>, >>=	_m_psrlw
Iu16vec4	>>, >>=	_m_psrlwi
Iu16vec4	cout <<	なし
Iu16vec4	[]	なし
Iu16vec4	cmpeq	_m_pcmpeqw
Iu16vec4	cmpneq	_m_pcmpeqw, _m_pandn
Iu16vec4	unpack_low	_m_punpcklwd

(続く)

表 A-5　クラス Iu16vec4 (続き)

クラス	機能	組み込み関数
Iu16vec4	unpack_high	_m_punpckhwd
Iu16vec4	sat_add	_m_paddusw
Iu16vec4	sat_sub	_m_psubusw
Iu16vec4	select_eq	_m_pand, _m_por, _m_pcmpeqw, _m_pandn
Iu16vec4	select_neq	_m_pand, _m_por, _m_pcmpeqw, _m_pandn

表 A-6　クラス Is8vec8

クラス	機能	組み込み関数
Is8vec8	&, &=	_m_pand
Is8vec8	|, |=	_m_por
Is8vec8	^, ^=	_m_pxor
Is8vec8	andnot	_m_pandn
Is8vec8	+, +=	_m_paddb
Is8vec8	-, -=	_m_psubb
Is8vec8	cout <<	なし
Is8vec8	[]	なし
Is8vec8	cmpeq	_m_pcmpeqb
Is8vec8	cmpneq	_m_pcmpeqb, _m_pandn
Is8vec8	cmpgt	_m_pcmpgtb
Is8vec8	cmplt	_m_pcmpgtb, _m_pandn
Is8vec8	cmpge	_m_pcmpgtb, _m_pandn
Is8vec8	unpack_high	_m_punpckhbw
Is8vec8	sat_add	_m_paddsb
Is8vec8	sat_sub	_m_psubsb

(続く)

表 A-6 クラス Is8vec8 (続き)

クラス	機能	組み込み関数
Is8vec8	select_eq	_m_pand, _m_por, _m_pcmpeqb, _m_pandn
Is8vec8	select_neq	_m_pand, _m_por, _m_pcmpeqb, _m_pandn
Is8vec8	select_gt	_m_pand, _m_por, _m_pcmpgtb, _m_pandn
Is8vec8	select_lt	_m_pand, _m_por, _m_pcmpgtb, _m_pandn
Is8vec8	select_ge	_m_pand, _m_por, _m_pcmpgtb, _m_pandn

表 A-7 クラス Iu8vec8

クラス	機能	組み込み関数
Iu8vec8	&, &=	_m_pand
Iu8vec8	|, |=	_m_por
Iu8vec8	^, ^=	_m_pxor
Iu8vec8	andnot	_m_pandn
Iu8vec8	+, +=	_m_paddb
Iu8vec8	-, -=	_m_psubb
Iu8vec8	cout <<	なし
Iu8vec8	[]	なし
Iu8vec8	cmpeq	_m_pcmpeqb
Iu8vec8	cmpneq	_m_pcmpeqb, _m_pandn
Iu8vec8	unpack_low	_m_punpcklbw
Iu8vec8	unpack_high	_m_punpckhbw
Iu8vec8	sat_add	_m_paddusb
Iu8vec8	sat_sub	_m_psubusb
Iu8vec8	select_eq	_m_pand, _m_por, _m_pcmpeqb, _m_pandn
Iu8vec8	select_neq	_m_pand, _m_por, _m_pcmpeqb, _m_pandn

表 A-8 **クラス** F32vec4

クラス	機能	組み込み関数
F32vec4	&, &=	_mm_and_ps
F32vec4	|, |=	_mm_or_ps
F32vec4	^, ^=	_mm_xor_ps
F32vec4	+, +=	_m_add_ps
F32vec4	-, -=	_m_sub_ps
F32vec4	*, *=	_mm_mul_ps
F32vec4	/, /=	_mm_div_ps
F32vec4	add_horizontal	_mm_shuffle_ps(3),_mm_add_ss(3)
F32vec4	sqrt	_mm_sqrt_ps
F32vec4	rcp	_mm_rcp_ps
F32vec4	rsqrt	_mm_rsqrt_ps
F32vec4	rcp_nr	_mm_rcp_ps, _mm_add_ps, _mm_mul_ps(2)
F32vec4	rsqrt_nr	_mm_rsqrt_ps, _mm_sub_ps, _mm_mul_ps(4)
F32vec4	cout <<	なし
F32vec4	[]	なし
F32vec4	cmpeq	_mm_cmpeq
F32vec4	cmpneq	_mm_cmpneq
F32vec4	cmpgt	_mm_cmpgt
F32vec4	cmplt	_mm_cmplt
F32vec4	cmpge	_mm_cmpge
F32vec4	cmpngt	_mm_cmpngt
F32vec4	cmpnlt	_mm_cmpnlt
F32vec4	cmpnge	_mm_cmpnge
F32vec4	simd_max	_mm_max_ps
F32vec4	simd_min	_mm_min_ps

（**続く**）

表 A-8 クラス F32vec4 (続き)

クラス	機能	組み込み関数
F32vec4	unpack_low	_mm_unpacklo_ps
F32vec4	unpack_high	_mm_unpackhi_ps
F32vec4	move_mask	_mm_movemask
F32vec4	loadu	_mm_loadu_ps
F32vec4	store_nta	_mm_stream_ps
F32vec4	select_eq	_mm_and_ps, _mm_or_ps, _mm_cmpeq_ps, _mm_andnot_ps
F32vec4	select_neq	_mm_and_ps, _mm_or_ps, _mm_cmpneq_ps, _mm_andnot_ps
F32vec4	select_gt	_mm_and_ps, _mm_or_ps, _mm_cmpgt_ps, _mm_andnot_ps
F32vec4	select_lt	_mm_and_ps, _mm_or_ps, _mm_cmplt_ps, _mm_andnot_ps
F32vec4	select_ge	_mm_and_ps, _mm_or_ps, _mm_cmpge_ps, _mm_andnot_ps
F32vec4	select_ngt	_mm_and_ps, _mm_or_ps, _mm_cmpngt_ps, _mm_andnot_ps
F32vec4	select_nlt	_mm_and_ps, _mm_or_ps, _mm_cmpnlt_ps, _mm_andnot_ps
F32vec4	select_nge	_mm_and_ps, _mm_or_ps, _mm_cmpnge_ps, _mm_andnot_ps

表 A-9 クラス F32vec1

クラス	機能	組み込み関数
F32vec1	&, &=	_mm_and_ps
F32vec1	|, |=	_mm_or_ps
F32vec1	^, ^=	_mm_xor_ps

(続く)

表 A-9 クラス F32vec1 (続き)

クラス	機能	組み込み関数
F32vec1	+, +=	_mm_add_ss
F32vec1	-,-=	_mm_sub_ss
F32vec1	*,*=	_mm_mul_ss
F32vec1	/,/=	_mm_div_ss
F32vec1	sqrt	_mm_sqrt_ss
F32vec1	rcp	_mm_rcp_ss
F32vec1	rsqrt	_mm_rsqrt_ss
F32vec1	rcp_nr	_mm_rcp_ss, _mm_add_ss, _mm_mul_ss(2)
F32vec1	rsqrt_nr	_mm_rsqrt_ss, _mm_sub_ss, _mm_mul_ss(4)
F32vec1	cmpeq	_mm_cmpeq_ss
F32vec1	cmplt	_mm_cmplt_ss
F32vec1	cmple	_mm_cmple_ss
F32vec1	cmpgt	_mm_cmpgt_ss
F32vec1	cmpge	_mm_cmpge_ss
F32vec1	cmpneq	_mm_cmpneq_ss
F32vec1	cmpnlt	_mm_cmpnlt_ss
F32vec1	cmpnle	_mm_cmpnle_ss
F32vec1	cmpngt	_mm_cmpngt_ss
F32vec1	cmpnge	_mm_cmpnge_ss
F32vec1	simd_min	_mm_min_ss
F32vec1	simd_max	_mm_max_ss
F32vec1	cout <<	なし
F32vec1	select_eq	_mm_and_ps, _mm_or_ps, _mm_cmpeq_ss, _mm_andnot_ps
F32vec1	select_le	_mm_and_ps, _mm_or_ps, _mm_cmple_ss, _mm_andnot_ps
F32vec1	select_gt	_mm_and_ps, _mm_or_ps, _mm_cmpgt_ss, _mm_andnot_ps

(続く)

表 A-9 クラス F32vec1 (続き)

クラス	機能	組み込み関数
F32vec1	select_ge	_mm_and_ps, _mm_or_ps, _mm_cmpge_ss, _mm_andnot_ps
F32vec1	select_neq	_mm_and_ps, _mm_or_ps, _mm_cmpneq_ss, _mm_andnot_ps
F32vec1	select_nlt	_mm_and_ps, _mm_or_ps, _mm_cmpnlt_ss, _mm_andnot_ps
F32vec1	select_nle	_mm_and_ps, _mm_or_ps, _mm_cmpnle_ss, _mm_andnot_ps
F32vec1	select_ngt	_mm_and_ps, _mm_or_ps, _mm_cmpngt_ss, _mm_andnot_ps
F32vec1	select_nge	_mm_and_ps, _mm_or_ps, _mm_cmpnge_ss, _mm_andnot_ps
F32vec1	F32vec1ToInt	_mm_cvtt_ss2si

次の表には、ベクトル・クラス・ライブラリでサポートされていない組み込み関数を示します。

表 A-10 クラスに含まれないコンパイラの組み込み関数

_m_pextrw
_m_pinsrw
_m_pshufw
_mm_avg_pu8
_mm_avg_pu16
_mm_free
_mm_load_ps
_mm_load_ps1
_mm_loadl_pi
_mm_loadr_ps
_mm_malloc
_mm_movehl_ps
_mm_movelh_ps

(続く)

表 A-10 クラスに含まれないコンパイラの組み込み関数 (続き)

_mm_prefetch
_mm_sad_pu8
_mm_setcsr
_mm_sfence
_mm_shuffle_ps
_mm_store_ps
_mm_store_ps1
_mm_storel_pi
_mm_storer_ps

プログラミングの例 B

このサンプル・プログラムは、F32vec4 クラスを使って、20 個の要素を持つ浮動小数点配列の要素の平均値を計算するというものです。このコードはファイルにもサンプルとして付属しています (AvgClass.cpp ファイル)。

```
// Include Streaming SIMD Extension Class Definitions
#include <fvec.h>

// Shuffle any 2 single precision floating point from a
// into low 2 SP FP and shuffle any 2 SP FP from b
//into high 2 SP FP of destination
#define SHUFFLE(a,b,i) (F32vec4)_mm_shuffle_ps(a,b,i)

#include <stdio.h>

#define SIZE 20

// Global variables
float result;
float array [SIZE];
```

```
//*********************************************************************
//
//  Function: Add20ArrayElements
//     Add all the elements of a 20 element array
//
//*********************************************************************
void Add20ArrayElements (F32vec4 *array, float *result)
{

       F32vec4 vec0, vec1;

       vec0 = _mm_load_ps ((float *) array);  // Load array's first 4 floats

//*****************************************************
//  Add all elements of the array, 4 elements at a time
//*****************************************************
       vec0 += array[1];// Add elements 5-8
       vec0 += array[2];// Add elements 9-12
       vec0 += array[3];// Add elements 13-16
       vec0 += array[4];// Add elements 17-20

//*********************************************************************
// There are now 4 partial sums.  Add the 2 lowers to the 2 highers,
//    then add those 2 results together
//*********************************************************************
       vec1 = SHUFFLE(vec1, vec0, 0x40);
       vec0 += vec1;
       vec1 = SHUFFLE(vec1, vec0, 0x30);
       vec0 += vec1;
       vec0 = SHUFFLE(vec0, vec0, 2);
       _mm_store_ss (result, vec0); /* Store the final sum */

}

void main(int argc, char *argv[])
{
```

```
      int       i ;

      // initialize the array
      for (i=0; i < SIZE; i++)
      {
            array[i] = (float) i;
      }

      // Call function to add all array elements
      Add20ArrayElements (array, &result);

      // Print average array element value
   printf ("Average of all array values = %f\n", result/20.);
      printf ("The correct answer is %f\n\n\n", 9.5);
}
```