SONY

SONY COMPUTER ENTERTAINMENT

# Cell Broadband Engine™アーキテクチャ用 SIMD 数学ライブラリ仕様書

## Version 1.0

CBEA JSRE Series
Cell Broadband Engine Architecture
Joint Software Reference
Environment Series

2006 年 11 月 6 日



ソニー株式会社
〒141-0001 東京都品川区北品川 6-7-35
（2007年2月より 〒108-0075 東京都港区港南 1-7-1）

株式会社ソニー・コンピュータエンタテインメント
〒107-0062 東京都港区南青山 2-6-21

ソニーのホームページ　**http://www.sony.net**
SCEIのホームページ　**http://www.scei.co.jp**

Cell Broadband Engine のホームページ　**http://cell.scei.co.jp**

2006年11月6日

# 目次

# 図目次



# 表目次

# 本ドキュメントについて

本ドキュメントは、Cell Broadband Engine™ の PowerPC Processor Unit (PPU) と Synergistic Processor Unit (SPU) のハードウェアが提供する SIMD（単一命令複数データ）命令を活用した数学ライブラリの仕様を述べたものです。SIMD 数学関数を使用すれば、一度に複数の結果を計算することにより、従来のスカラ数学ライブラリで可能であったものよりもはるかに高い性能を、プログラマは PPU と SPU のプログラムから得ることができます。

## 対象者

本ドキュメントは、Cell Broadband Engine™ 向けの高性能プログラムを書くことに関心を持つシステムプログラマおよびアプリケーションプログラマを対象としています。.

## 変更履歴

本セクションでは、本ドキュメントの各バージョンに施された重要な変更点を記載しています。

| バージョン番号および日付 | 変更点 |
|---|---|
| v. 1.0<br>2006 年 11 月 6 日 | 初版作成 |

## 関連ドキュメント

以下の表は、本ドキュメントの参考文献および補足資料の一覧です。

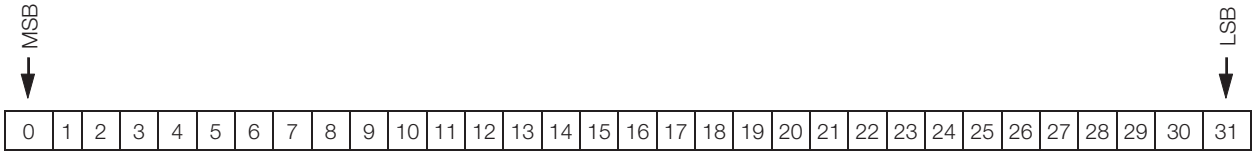
| ドキュメント名 | バージョン | 日付 |
|---|---|---|
| *Cell Broadband Engine™ アーキテクチャ用 C/C++言語拡張* | 2.3 | 2006 年 12 月 |
| *ISO/IEC Standard 9899:1999 (C Standard)* | | |
| *IEC Standard 60559:1989 (Standard for Binary Floating-Point Arithmetic)* | | |
| *Synergistic Processor Unit 命令セット・アーキテクチャ* | 1.11 | 2006 年 10 月 |
| *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*, http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/C40E4C6133B31EE8872570B500791108 | 2.06c | |
| *PowerPC Architecture Book*, http://www-128.ibm.com/developerworks/eserver/library/es-archguide-v2.html | 2.02 | |

## 本ドキュメントの構成

本ドキュメントは 2 つの章から成ります。1 章は SIMD 数学ライブラリの概要であり、2 章は本ライブラリを構成する個別の数学関数を説明した仕様です。

## ビット表記について

本ドキュメントでは、標準ビット表記を採用しています。ビット番号およびバイト番号は、左から右へ昇順に並んでいます。従って、4 バイトワードの場合、下図のようにビット 0 が最上位ビットでビット 31 が最下位ビットになります。




MSB = Most significant ビット（最上位ビット）

LSB = Least significant ビット（最下位ビット）

ビットエンコーディングの表記法は以下の通りです。

- 16 進数の値の前には、`0x` が付いています。例：`0x0A00`
- 文中に出てくるバイナリ（2 進数）値は、引用符（‘ ’）で囲んでいます。例：‘1010’

## バイト順序と要素の番号付け

図 1に示すように、バイト順序と要素（スロット）の番号付けは常にビッグエンディアン順序で示されます。

図 1: ベクタ型のバイトと要素のビッグエンディアン順序

| Byte 0 (MSB) | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 | Byte 9 | Byte 10 | Byte 11 | Byte 12 | Byte 13 | Byte 14 | Byte 15 (LSB) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *doubleword 0* | | | | | | | | *doubleword 1* | | | | | | | |
| *word 0* | | | | *word 1* | | | | *word 2* | | | | *word 3* | | | |
| *halfword 0* | | *halfword 1* | | *halfword 2* | | *halfword 3* | | *halfword 4* | | *halfword 5* | | *halfword 6* | | *halfword 7* | |
| *char 0* | *char 1* | *char 2* | *char 3* | *char 4* | *char 5* | *char 6* | *char 7* | *char 8* | *char 9* | *char 10* | *char 11* | *char 12* | *char 13* | *char 14* | *char 15* |

## 書体表記の規約

本ドキュメントでは、ビット表記に加え、以下の書体表記の規約を採用しています。

| 書体 | 意味 |
|---|---|
| `courier` | プログラミングコード、処理命令、レジスタ名、データ型、イベント、ファイル名、および他のリテラルを表します。関数名およびマクロ名を表す場合にも使用されます。この書体は、理解に役立つ場合にのみ使用され、特に説明文中で使用されます。 |
| *`courier + イタリック体`* | 引数、パラメータ、および変数（`const` 型の変数を含む）を表します。この書体は、理解に役立つ場合にのみ使用され、特に説明文中で使用されます。 |
| *イタリック体 (courier なし)* | 強調を表します。ハイパーリンクの場合を除いて、文献の参照にはイタリック体が使用されます。言葉を初めて定義する場合には、イタリック体が使用されることがよくあります。 |
| 青 | ハイパーリンクを表します（カラープリンタまたはオンライン専用）。 |

# 1. SIMD数学ライブラリの概要

PPU と SPU の命令セットには SIMD（単一命令複数データ）命令が含まれています。それらは通常の命令と類似していますが、同時に 2 個以上の入力に対して演算を行ないます。従来の数学ライブラリは、単一の入力に対する演算であり、SIMD 命令のスピードとパワーを活かすことはできません。SIMD 数学ライブラリには、C99 規格、すなわち *ISO/IEC Standard 9899:1999 (C Standard)*で規定されているスカラ数学関数の SIMD 版が含まれています。本章では、これらの特別な PPU および SPU SIMD ライブラリの仕様を規定します。

## A. ライブラリとヘッダファイル

SIMD ライブラリの名称には `simdmath` という文字列を含むようにします。例えば、GNU/Linux ではライブラリは `libsimdmath.a` あるいは `libsimdmath.so`（共有ライブラリ版の場合）と呼ばれています。システムヘッダファイル `simdmath.h` には、SIMD 数学関数の型宣言やプロトタイプを含むようにします。

## B. 関数概要

PPUおよびSPU SIMD数学ライブラリを構成する関数を 表 1に列挙します。非標準と記されている関数については、C99 には対応するものが存在しません。

SIMD 数学関数の名称としては、標準のスカラ関数名にベクタ型の接尾辞を付けることで、スカラ版の対応物と区別しています。例えば、`fabsf()` の SIMD 版で vector float に対して演算するものは `fabsf4()` と称します。同様に、標準のスカラ関数の SIMD 版で vector double に対して演算するものには、名称に `d2` を付加することになります。

表 1: SIMD 数学関数

| 関数名 | C99 での名称 | 関数カテゴリ | 精度 | SPU/PPU |
|---|---|---|---|---|
| absi4 | abs | 整数 | word | 両方 |
| acosd2 | acos | 三角関数 | double | SPU |
| acosf4 | acosf | 三角関数 | single | 両方 |
| acoshd2 | acosh | 三角関数 | double | SPU |
| acoshf4 | acoshf | 三角関数 | single | 両方 |
| asind2 | asin | 三角関数 | double | SPU |
| asinf4 | asinf | 三角関数 | single | 両方 |
| asinhd2 | asinh | 三角関数 | double | SPU |
| asinhf4 | asinhf | 三角関数 | single | 両方 |
| atan2d2 | atan2 | 三角関数 | double | SPU |
| atan2f4 | atan2f | 三角関数 | single | 両方 |
| atand2 | atan | 三角関数 | double | SPU |
| atanf4 | atanf | 三角関数 | single | 両方 |
| atanhd2 | atanh | 三角関数 | double | SPU |
| atanhf4 | atanhf | 三角関数 | single | 両方 |
| cbrtd2 | cbrt | べき乗 | double | SPU |
| cbrtf4 | cbrtf | べき乗 | single | 両方 |
| ceild2 | ceil | 丸め | double | SPU |
| ceilf4 | ceilf | 丸め | single | 両方 |
| copysignd2 | copysign | その他 | double | SPU |
| copysignf4 | copysignf | その他 | single | 両方 |

| 関数名 | C99 での名称 | 関数カテゴリ | 精度 | SPU/PPU |
|---|---|---|---|---|
| cosd2 | cos | 三角関数 | double | SPU |
| cosf4 | cosf | 三角関数 | single | 両方 |
| coshd2 | cosh | 三角関数 | double | SPU |
| coshf4 | coshf | 三角関数 | single | 両方 |
| divd2 | 非標準 | 除算 | double | SPU |
| divf4 | 非標準 | 除算 | single | 両方 |
| divi4 | div | 整数 | word | 両方 |
| divu4 | 非標準 | 整数 | word | 両方 |
| erfcd2 | erfc | 誤差関数 | double | SPU |
| erfcf4 | erfcf | 誤差関数 | single | 両方 |
| erfd2 | erf | 誤差関数 | double | SPU |
| erff4 | erff | 誤差関数 | single | 両方 |
| exp2d2 | exp2 | 指数関数 | double | SPU |
| exp2f4 | exp2f | 指数関数 | single | 両方 |
| expd2 | exp | 指数関数 | double | SPU |
| expf4 | expf | 指数関数 | single | 両方 |
| expm1d2 | expm1 | 指数関数 | double | SPU |
| expm1f4 | expm1f | 指数関数 | single | 両方 |
| fabsd2 | fabs | 絶対値 | double | SPU |
| fabsf4 | fabsf | 絶対値 | single | 両方 |
| fdimd2 | fdim | その他 | double | SPU |
| fdimf4 | fdimf | その他 | single | 両方 |
| floord2 | floor | 丸め | double | SPU |
| floorf4 | floorf | 丸め | single | 両方 |
| fmad2 | fma | その他 | double | SPU |
| fmaf4 | fmaf | その他 | single | 両方 |
| fmaxd2 | fmax | 最小最大 | double | SPU |
| fmaxf4 | fmaxf | 最小最大 | single | 両方 |
| fmind2 | fmin | 最小最大 | double | SPU |
| fminf4 | fminf | 最小最大 | single | 両方 |
| fmodd2 | fmod | 剰余 | double | SPU |
| fmodf4 | fmodf | 剰余 | single | 両方 |
| fpclassifyd2 | fpclassify | 比較 | double | SPU |
| fpclassifyf4 | fpclassify | 比較 | single | 両方 |
| frexpd2 | frexp | その他 | double | SPU |
| frexpf4 | frexpf | その他 | single | 両方 |
| hypotd2 | hypot | その他 | double | SPU |
| hypotf4 | hypotf | その他 | single | 両方 |
| ilogbd2 | ilogb | 対数関数 | double | SPU |
| ilogbf4 | ilogbf | 対数関数 | single | 両方 |
| irintf4 | 非標準 | その他 | single | 両方 |
| iroundf4 | 非標準 | 丸め | single | 両方 |

| 関数名 | C99 での名称 | 関数カテゴリ | 精度 | SPU/PPU |
|---|---|---|---|---|
| is0denormf4 | 非標準 | 比較 | single | 両方 |
| is0denormd2 | 非標準 | 比較 | double | SPU |
| isequald2 | 非標準 | 比較 | double | SPU |
| isequalf4 | 非標準 | 比較 | single | 両方 |
| isfinited2 | isfinite | 比較 | double | SPU |
| isfinitef4 | isfinite | 比較 | single | 両方 |
| isgreaterd2 | isgreater | 比較 | double | SPU |
| isgreaterf4 | isgreater | 比較 | single | 両方 |
| isgreaterequald2 | isgreaterequal | 比較 | double | SPU |
| isgreaterequalf4 | isgreaterequal | 比較 | single | 両方 |
| islessd2 | isless | 比較 | double | SPU |
| islessf4 | isless | 比較 | single | 両方 |
| islessequald2 | islessequal | 比較 | double | SPU |
| islessequalf4 | islessequal | 比較 | single | 両方 |
| islessgreaterd2 | islessgreater | 比較 | double | SPU |
| islessgreaterf4 | islessgreater | 比較 | single | 両方 |
| isunorderedd2 | isunordered | 比較 | double | SPU |
| isunorderedf4 | isunordered | 比較 | single | 両方 |
| isinfd2 | isinf | 比較 | double | SPU |
| isinff4 | isinf | 比較 | single | 両方 |
| isnand2 | isnan | 比較 | double | SPU |
| isnanf4 | isnan | 比較 | single | 両方 |
| isnormald2 | isnormal | 比較 | double | SPU |
| isnormalf4 | isnormal | 比較 | single | 両方 |
| ldexpd2 | ldexp | その他 | double | SPU |
| ldexpf4 | ldexpf | その他 | single | 両方 |
| lgammad2 | lgamma | ガンマ関数 | double | SPU |
| lgammaf4 | lgammaf | ガンマ関数 | single | 両方 |
| llabsi2 | llabs | 整数 | double word | SPU |
| lldivi2 | lldiv | 整数 | double word | SPU |
| lldivu2 | 非標準 | 整数 | double word | SPU |
| llrintd2 | llrint | 丸め | double | SPU |
| llrintf4 | llrintf | 丸め | single | SPU |
| llroundd2 | llround | 丸め | double | SPU |
| llroundf4 | llroundf | 丸め | single | SPU |
| log10d2 | log10 | 対数関数 | double | SPU |
| log10f4 | log10f | 対数関数 | single | 両方 |
| log1pd2 | log1p | 対数関数 | double | SPU |
| log1pf4 | log1pf | 対数関数 | single | 両方 |
| log2d2 | log2 | 対数関数 | double | SPU |
| log2f4 | log2f | 対数関数 | single | 両方 |
| logbd2 | logb | 対数関数 | double | SPU |

| 関数名 | C99 での名称 | 関数カテゴリ | 精度 | SPU/PPU |
|---|---|---|---|---|
| logbf4 | logbf | 対数関数 | single | 両方 |
| logd2 | log | 対数関数 | double | SPU |
| logf4 | logf | 対数関数 | single | 両方 |
| modfd2 | modf | 剰余 | double | SPU |
| modff4 | modff | 剰余 | single | 両方 |
| nearbyintd2 | nearbyint | 丸め | double | SPU |
| nearbyintf4 | nearbyintf | 丸め | single | 両方 |
| negated2 | 非標準 | その他 | double | SPU |
| negatef4 | 非標準 | その他 | single | 両方 |
| negatei4 | 非標準 | 整数 | word | 両方 |
| negatell2 | 非標準 | 整数 | double word | SPU |
| nextafterd2 | nextafter | 丸め | double | SPU |
| nextafterf4 | nextafterf | 丸め | single | 両方 |
| powd2 | pow | べき乗 | double | SPU |
| powf4 | powf | べき乗 | single | 両方 |
| recipd2 | 非標準 | 逆数 | double | SPU |
| recipf4 | 非標準 | 逆数 | single | 両方 |
| remainderd2 | remainder | 剰余 | double | SPU |
| remainderf4 | remainderf | 剰余 | single | 両方 |
| remquod2 | remquo | 剰余 | double | SPU |
| remquof4 | remquof | 剰余 | single | 両方 |
| rintd2 | rint | 丸め | double | SPU |
| rintf4 | rintf | 丸め | single | 両方 |
| roundd2 | round | 丸め | double | SPU |
| roundf4 | roundf | 丸め | single | 両方 |
| rsqrtd2 | 非標準 | 平方根 | double | SPU |
| rsqrtf4 | 非標準 | 平方根 | single | 両方 |
| scalbllnd2 | 非標準 | その他 | double | SPU |
| scalbnf4 | scalbnf | その他 | single | 両方 |
| signbitf4 | signbit | 比較 | single | 両方 |
| signbitd2 | signbit | 比較 | double | SPU |
| sincosd2 | 非標準 | 三角関数 | double | SPU |
| sincosf4 | 非標準 | 三角関数 | single | 両方 |
| sind2 | sin | 三角関数 | double | SPU |
| sinf4 | sinf | 三角関数 | single | 両方 |
| sinhd2 | sinh | 三角関数 | double | SPU |
| sinhf4 | sinhf | 三角関数 | single | 両方 |
| sqrtd2 | sqrt | 平方根 | double | SPU |
| sqrtf4 | sqrtf | 平方根 | single | 両方 |
| tand2 | tan | 三角関数 | double | SPU |
| tanf4 | tanf | 三角関数 | single | 両方 |
| tanhd2 | tanh | 三角関数 | double | SPU |

| 関数名 | C99 での名称 | 関数カテゴリ | 精度 | SPU/PPU |
|---|---|---|---|---|
| tanhf4 | tanhf | 三角関数 | single | 両方 |
| tgammad2 | tgamma | ガンマ関数 | double | SPU |
| tgammaf4 | tgammaf | ガンマ関数 | single | 両方 |
| truncd2 | trunc | 丸め | double | SPU |
| truncf4 | truncf | 丸め | single | 両方 |

## C. 特殊なケース

特記がない場合には、SIMD 結果の各要素は、C99 規格か *IEC 60559:1989* 規格のいずれかに従うことになります。

### 1. 丸め

SPU 上では、倍精度に対しては IEEE の丸めモードの全種類がサポートされますが、単精度に対してはゼロ方向への丸めのみのサポートとなります。PPU 上では SIMD 演算は常に IEEE の最近接値への丸めモードを使用します。

SIIMD 関数の数学的精度（の確保のために）は、デフォルトの丸めモードを想定しています。関数がその他の丸めモードで呼ばれた場合には、精度が低下する場合があります。

### 2. 特殊オペランド

PPU 上では `NaN` と `Inf` は特別なオペランドとして認識されます。

SPU 上では、単精度関数に渡されるすべての値が通常のオペランドとして扱われ、`NaN` と `Inf` も特別な単精度オペランドとしては認識されません。しかし、倍精度オペランドとしては特別な値として認識され、SIMD 関数ではそのチェックが、C99 や *IEC 60559:1989* や各 SIMD 関数の仕様で規定しているとおりに行なわれます。詳細については *Synergistic Processor Unit 命令セット・アーキテクチャ* を参照してください。

PPU 上と SPU 上の両方で、単精度浮動小数点の非正規化数入力は、特記がない場合にはゼロに強制変更されます。

### 3. エラー条件

*定義域エラー*は、入力引数が数学関数の定義されている領域外である場合に生じます。各関数の記述では、必須の定義域エラーがあればそれを列挙してあります。結果のベクタは、入力引数の定義域エラーを含む要素に対応する要素については不定となり、例外やエラーは報告されません。

*値域エラー*は、数学的な結果が指定の型のオブジェクトで表現できない場合に生じます。値域エラーが生じた際には、結果のベクタ要素は `HUGE_VAL`（倍精度結果の場合）か `HUGE_VALF`（単精度結果の場合）のいずれかになります。整数算術関数の結果は、表現できない場合には不定となります。

### 4. 例外

SIMD ライブラリ関数が SPU の浮動小数点ステータス/制御レジスタ（FPSCR）中の例外フラグに対して及ぼす影響は不定です。倍精度引数に対する SPU 関数は FPSCR 中の例外ビットを設定しますので、それを `fegetexcept` ルーチン（`fenv.h` に記載されています）を呼ぶことでテストできます。

SPU は単精度浮動小数点例外に対するハードウェアトラップを引き起こすことはありません。PPU の SIMD 演算では C99 の浮動小数点例外のサブセットに対するハードウェアサポートが実際に行なわれます。

# 2. SIMD関数仕様

本章では、SIMD 数学関数とその引数や戻り値の説明を行ないます。必要に応じて、精度の情報を与え、期待される特定の挙動を明確にしています。特記がない関数についてはすべて、PPU と SPU の両方で使用可能です。

## A. 型定義

関数の戻り値用に以下の型定義が用いられています。

### divi4_t: 剰余/商構造体（Vector Signed Int用）

```
typedef struct divi4_s {
   vector signed int quot;
   vector signed int rem;
} divi4_t;
```

この型の構造体は `divi4()` の戻り値を保持するために用います。メンバ *`quot`* には除算の商が入り、メンバ *`rem`* には除算の剰余が入ります。

### divu4_t: 剰余/商構造体（Vector Unsigned Int用）

```
typedef struct divu4_s {
   vector unsigned int quot;
   vector unsigned int rem;
} divu4_t;
```

この型の構造体は `divu4()` の戻り値を保持するために用います。メンバ *`quot`* には除算の商が入り、メンバ *`rem`* には除算の剰余が入ります。

### lldivi2_t: 剰余/商構造体（Vector Signed Long Long用、SPUのみ）

```
typedef struct lldivi2_s {
   vector signed long long quot;
   vector signed long long rem;
} lldivi2_t;
```

この型の構造体は `lldivi2()` の戻り値を保持するために用います。メンバ *`quot`* には除算の商が入り、メンバ *`rem`* には除算の剰余が入ります。

### lldivu2_t: 剰余/商構造体（Vector Unsigned Long Long用、SPUのみ）

```
typedef struct lldivu2_s {
   vector unsigned long long quot;
   vector unsigned long long rem;
} lldivu2_t;
```

この型の構造体は `lldivu2()` の戻り値を保持するために用います。メンバ *`quot`* には除算の商が入り、メンバ *`rem`* には除算の剰余が入ります。

### llroundf4_t: Long Long 4 個のベクタ（SPUのみ）

```
typedef struct llroundf4_s {
   vector signed long long vll[2];
} llroundf4_t;
```

この型の構造体は、4 要素のベクタに対応する signed long long データを保持するために用います。

## B. 関数説明

以下の関数の説明中では添字を用いてベクタ要素を示します。例えば、ベクタ $x$ の要素 *i* は $x_i$ と表します。

### absi4: Absolute Value of Integer

```
(vector signed int) absi4 (vector signed int x);
```

vector signed int $x$ の各要素の値の絶対値を収めた vector signed int を返します。

$x_i$ の絶対値が表現できない場合には、それに対応する結果は不定となり、エラーは報告されません。

### acosd2: Arccosine of Double (SPU Only)

```
(vector double) acosd2 (vector double x);
```

余弦（コサイン）が vector double $x$ の各要素に一致するような角度を収めた vector double を返します。結果の各要素は、[0, pi] ラジアンの範囲内となります。

$x_i$ の絶対値が 1 より大きい場合には、それに対応する結果は不定となり、エラーは報告されません。

### acosf4: Arccosine of Float

```
(vector float) acosf4 (vector float x);
```

余弦（コサイン）が vector float $x$ の各要素に一致するような角度を収めた vector float を返します。結果の各要素は、[0, pi] ラジアンの範囲内となります。

$x_i$ の絶対値が 1 より大きい場合には、それに対応する結果は不定となり、エラーは報告されません。

### acoshd2: Hyperbolic Arccosine of Double (SPU Only)

```
(vector double) acoshd2 (vector double x);
```

vector double $x$の各要素に対応する双曲線逆余弦(ハイパボリックアークコサイン)の非負のものを収めたvector double を返します。

$x_i$ の値が 1 より小さい場合には、それに対応する結果は不定となり、エラーは報告されません。

### acoshf4: Hyperbolic Arccosine of Float

```
(vector float) acoshf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線逆余弦（ハイパボリックアークコサイン）の非負のものを収めた vector float を返します。

$x_i$ の値が 1 より小さい場合には、それに対応する結果は不定となり、エラーは報告されません。

### asind2: Arcsine of Double (SPU Only)

```
(vector double) asind2 (vector double x);
```

正弦（サイン）が vector double $x$ の各要素に一致するような角度を収めた vector double を返します。結果の各要素は、[-pi/2, +pi/2] ラジアンの範囲内となります。

$x_i$ の絶対値が 1 より大きい場合には、それに対応する結果は不定となり、エラーは報告されません。

### asinf4: Arcsine of Float

```
(vector float) asinf4 (vector float x);
```

正弦（サイン）が vector float $x$ の各要素に一致するような角度を収めた vector float を返します。結果の各要素は、[-pi/2, +pi/2] ラジアンの範囲内となります。

$x_i$ の絶対値が 1 より大きい場合には、それに対応する結果は不定となり、エラーは報告されません。

**asinhd2: Hyperbolic Arcsine of Double (SPU Only)**

```
(vector double) asinhd2 (vector double x);
```

vector double $x$ の各要素に対応する双曲線逆正弦（ハイパボリックアークサイン）を収めた vector double を返します。

**asinhf4: Hyperbolic Arcsine of Float**

```
(vector float) asinhf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線逆正弦（ハイパボリックアークサイン）を収めた vector float を返します。

**atand2: Tangent of Double (SPU Only)**

```
(vector double) atand2 (vector double x);
```

正接（タンジェント）が vector double $x$ の各要素に一致するような角度を収めた vector double を返します。結果の各要素は、[-pi/2, +pi/2] ラジアンの範囲内となります。

**atanf4: Tangent of Float**

```
(vector float) atanf4 (vector float x);
```

正接（タンジェント）が vector float $x$ の各要素に一致するような角度を収めた vector float を返します。結果の各要素は、[-pi/2, +pi/2] ラジアンの範囲内となります。

**atanhd2: Hyperbolic Arctangent of Double (SPU Only)**

```
(vector double) atanhd2 (vector double x)
```

vector double $x$ の各要素に対応する双曲線逆正接（ハイパボリックアークタンジェント）を収めた vector double を返します。

$x_i$ の絶対値が 1 より大きい場合には、それに対応する結果は不定となり、エラーは報告されません。

**atanhf4: Hyperbolic Arctangent of Float**

```
(vector float) atanhf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線逆正接（ハイパボリックアークタンジェント）を収めた vector float を返します。

$x_i$ の絶対値が 1 より大きい場合には、対応する結果は不定となり、エラーは報告されません。

SPU 上で $x_i$ が 1 または -1 に等しい場合には、それに対応する結果の要素には `HUGE_VALF` あるいは `-HUGE_VALF` が返され、エラーは報告されません。

**atan2d2: Arctangent of Double Quotient (SPU Only)**

```
(vector double) atan2d2 (vector double y, vector double x);
```

正接（タンジェント）が vector double $y$ と vector double $x$ の各要素に対して $y_i/x_i$ となるような角度を収めた vector double を返します。結果の各要素は、[-pi, +pi] ラジアンの範囲内となります。

$x_i$ と $y_i$ が共にゼロの場合は、それに対応する結果の要素は不定となり、エラーは報告されません。

**atan2f4: Arctangent of Float Quotient**

```
(vector float) atan2f4 (vector float y, vector float x);
```

正接（タンジェント）が vector float $y$ と vector float $x$ の各要素に対して $y_i/x_i$ となるような角度を収めた vector float を返します。結果の各要素は、[-pi, +pi] ラジアンの範囲内となります。

$x_i$ と $y_i$ が共にゼロの場合は、それに対応する結果の要素は不定となり、エラーは報告されません。

### cbrtd2: Cube Root of Double (SPU Only)

```
(vector double) cbrtd2 (vector double x);
```

vector double $x$ の各要素に対応する実数立方根 $x_i^{1/3}$ を収めた vector double を返します。

### cbrtf4: Cube Root of Float

```
(vector float) cbrtf4 (vector float x);
```

vector float $x$ の各要素に対応する実数立方根 $x_i^{1/3}$ を収めた vector float を返します。

### ceild2: Ceiling of Double (SPU Only)

```
(vector double) ceild2 (vector double x);
```

vector double $x$ の各要素よりも小さくない最小の整数値の浮動小数点表現を収めた vector double を返します。

### ceilf4: Ceiling of Float

```
(vector float) ceilf4 (vector float x);
```

vector float $x$ の各要素よりも小さくない最小の整数値の浮動小数点表現を収めた vector float を返します。

### copysignd2: Copy Sign of Double (SPU Only)

```
(vector double) copysignd2 (vector double x, vector double y);
```

vector double $x$ の各要素の絶対値と vector double $y$ の各対応要素の符号を収めた vector double を返します。

### copysignf4: Copy Sign of Float

```
(vector float) copysignf4 (vector float x, vector float y);
```

vector float $x$ の各要素の絶対値と vector float $y$ の各対応要素の符号を収めた vector float を返します。

### cosd2: Cosine of Double (SPU Only)

```
(vector double) cosd2 (vector double x);
```

vector double $x$ の各要素に対応する余弦（コサイン）を収めた vector double を返します。

`cosd2()` の結果は、$x$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### cosf4: Cosine of Float

```
(vector float) cosf4 (vector float x);
```

vector float $x$ の各要素に対応する余弦（コサイン）を収めた vector float を返します。

`cosf4()` の結果は、$x$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### coshd2: Hyperbolic Cosine of Double (SPU Only)

```
(vector double) coshd2 (vector double x);
```

vector double $x$ の各要素に対応する双曲線余弦（ハイパボリックコサイン）を収めた vector double を返します。

### coshf4: Hyperbolic Cosine of Float

```
(vector float) coshf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線余弦（ハイパボリックコサイン）を収めた vector float を返します。

SPU においては、結果の要素の値で `HUGE_VALF` よりも大きなものは `HUGE_VALF` として返され、エラーは報告されません。

### divd2: Divide Doubles (SPU Only)

```
(vector double) divd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素についての商 $x_i / y_i$ を収めた vector double を返します。この関数は特殊ケースを以下のように扱います。

- いずれかの入力が NaN であれば、結果は NaN となります。
- Inf/Inf あるいは 0/0 については、結果は NaN となります。
- 有限/0 については、結果は Inf に符号 *sign = sign(x)/sign(y)* を付けたものとなります。
- 有限/±Inf については、結果は 0 に符号 *sign = sign(x)/sign(y)* を付けたものとなります。

### divf4: Divide Floats

```
(vector float) divf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素についての商 $x_i / y_i$ を収めた vector float を返します。この関数は特殊ケースを以下のように扱います。

- いずれかの入力が NaN であれば、結果は NaN となります。
- Inf/Inf あるいは 0/0 については、結果は NaN となります。
- 有限/0 については、結果は Inf に符号 *sign = sign(x)/sign(y)* を付けたものとなります。
- 有限/±Inf については、結果は 0 に符号 *sign = sign(x)/sign(y)* を付けたものとなります。

SPU においては、$y_i$ がゼロの場合は、結果は HUGE_VALF に符号 *sign = sign(x)/sign(y)* を付けたものになります。

### divi4: Divide Integer

```
(divi4_t) divi4 (vector signed int x, vector signed int y);
```

vector signed int $x$ の各要素を vector signed int $y$ の各対応要素で除算し、結果を divi4_t 型の構造体で返します。その中には、対応する商のベクタと対応する剰余のベクタとが入っています。

構造体メンバ *quot* の各要素は、ゼロ方向に切り捨てた算術商です。構造体メンバ *rem* の各要素は、対応する剰余であり、$x_i == quot * y_i + rem$ が成り立ちます。

$y_i$ がゼロの場合は、結果の商の対応する要素はゼロになります。

### divu4: Divide Unsigned Integer

```
(divu4_t) divu4 (vector unsigned int x, vector unsigned int y);
```

vector unsigned int $x$ の各要素を vector unsigned int $y$ の各対応要素で除算し、結果を divu4_t 型の構造体で返します。その中には、対応する商のベクタと対応する剰余のベクタとが入っています。

構造体メンバ *quot* の各要素は、ゼロ方向に切り捨てた算術商です。構造体メンバ *rem* の各要素は、対応する剰余であり、$x_i == quot * y_i + rem$ が成り立ちます。

$y_i$ がゼロの場合は、結果の商の対応する要素はゼロになります。

### erfcd2: Complementary Error Function Double (SPU Only)

```
(vector double) erfcd2 (vector double x);
```

vector double $x$ の各要素に対応する相補誤差関数を収めた vector double を返します。

### erfcf4: Complementary Error Function Float

```
(vector float) erfcf4 (vector float x);
```

vector float $x$ の各要素に対応する相補誤差関数を収めた vector float を返します。

### erfd2: Error Function Double (SPU Only)

```
(vector double) erfd2 (vector double x);
```

vector double $x$ の各要素に対応する誤差関数を収めた vector double を返します。

### erff4: Error Function Float

```
(vector float) erff4 (vector float x);
```

vector float $x$ の各要素に対応する誤差関数を収めた vector float を返します。

### expd2: *e* Raised to the Power of Double (SPU Only)

```
(vector double) expd2 (vector double x);
```

vector double $x$ の各要素に対応する指数関数 $e^{x_i}$ を収めた vector double を返します。

### expf4: *e* Raised to the Power of Float

```
(vector float) expf4 (vector float x);
```

vector float $x$ の各要素に対応する指数関数 $e^{x_i}$ を収めた vector float を返します。

SPU においては、結果の要素の値で `HUGE_VALF` よりも大きなものは `HUGE_VALF` として返され、エラーは報告されません。

### exp2d2: 2 Raised to the Power of Double (SPU Only)

```
(vector double) exp2d2 (vector double x);
```

vector double $x$ の各要素に対応する指数関数 $2^{x_i}$ を収めた vector double を返します。

### exp2f4: 2 Raised to the Power of Float

```
(vector float) exp2f4 (vector float x);
```

vector float $x$ の各要素に対応する指数関数 $2^{x_i}$ を収めた vector float を返します。

SPU においては、結果の要素の値で `HUGE_VALF` よりも大きなものは `HUGE_VALF` として返され、エラーは報告されません。

### expm1d2: *e* Raised to the Power of Double Minus 1 (SPU Only)

```
(vector double) expm1d2 (vector double x);
```

vector double $x$ の各要素に対応する指数関数から 1 を引いた値 $e^{x_i} - 1$ を収めた vector double を返します。

この関数は、$x_i$ が 0 に近い場合、つまり、`exp (`$x_i$`)-1.0` では浮動小数点の桁落ち誤差のために正しくない値が返るような場合においても、数学的に正確な値を返します。

### expm1f4: *e* Raised to the Power of Float Minus 1

```
(vector float) expm1f4 (vector float x);
```

vector float $x$ の各要素に対応する指数関数から 1 を引いた値 $e^{x_i} - 1$ を収めた vector float を返します。

この関数は、$x_i$ が 0 に近い場合、つまり、`exp (`$x_i$`)-1.0` では浮動小数点の桁落ち誤差のために正しくない値が返るような場合においても、数学的に正確な値を返します。

**fabsd2: Absolute Value Double (SPU Only)**

```
(vector double) fabsd2 (vector double x);
```

vector double $x$ の各要素に対応する絶対値 $|x_i|$ を収めた vector double を返します。

**fabsf4: Absolute Value Float**

```
(vector float) fabsf4 (vector float x);
```

vector float $x$ の各要素に対応する絶対値 $|x_i|$ を収めた vector float を返します。

**fdimd2: Subtract Staying Non-Negative Double (SPU Only)**

```
(vector double) fdimd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、$(x_i-y_i)$ とゼロの大きい方を収めた vector double を返します。

**fdimf4: Subtract Staying Non-Negative Float**

```
(vector float) fdimf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、$(x_i-y_i)$ とゼロの大きい方を収めた vector float を返します。

**floord2: Floor Double (SPU Only)**

```
(vector double) floord2 (vector double x);
```

vector double $x$ の各要素よりも大きくない最大の整数値の浮動小数点表現を収めた vector double を返します。

**floorf4: Floor Float**

```
(vector float) floorf4 (vector float x);
```

vector float $x$ の各要素よりも大きくない最大の整数値の浮動小数点表現を収めた vector float を返します。

**fmad2: Fused Multiply and Add Double (SPU Only)**

```
(vector double) fmad2 (vector double x, vector double y, vector double z);
```

vector double $x$、vector double $y$、vector double $z$ の各要素に対して、$(x_i * y_i + z_i)$ の計算結果を収めた vector double を返します。中間結果は任意の精度を有しています。

**fmaf4: Fused Multiply and Add Float**

```
(vector float) fmaf4 (vector float x, vector float y, vector float z);
```

vector float $x$、vector float $y$、vector float $z$ の各要素に対して、$(x_i * y_i + z_i)$ の計算結果を収めた vector float を返します。中間結果は任意の精度を有しています。

**fmaxd2: Maximum Double (SPU Only)**

```
(vector double) fmaxd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、$x_i$ と $y_i$ の（正の方向に）大きい方を収めた vector double を返します。

**fmaxf4: Maximum Float**

```
(vector float) fmaxf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、$x_i$ と $y_i$ の（正の方向に）大きい方を収めた vector float を返します。

SPU においては、この関数は非正規化数をゼロに強制変更しません。そうしないで、通常の値として比較しますが、これは SPU の浮動小数点命令の挙動とは異なっています。

### fmind2: Minimum Double (SPU Only)

```
(vector double) fmind2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、$x_i$ と $y_i$ の（負の方向に）小さい方を収めた vector double を返します。

### fminf4: Minimum Float

```
(vector float) fminf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、$x_i$ と $y_i$ の（負の方向に）小さい方を収めた vector float を返します。

SPU においては、この関数は非正規化数をゼロに強制変更しません。そうしないで、通常の値として比較しますが、これは SPU の浮動小数点命令の挙動とは異なっています。

### fmodd2: Modulus Double (SPU Only)

```
(vector double) fmodd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、以下のように定まる $x_i/y_i$ の剰余を収めた vector double を返します。

- $y_i$ が 0 の場合は、結果は 0 になります。
- そうでなければ、結果の要素が $x_i - int * y_i$ で、$x_i$ と同符号を持ち、絶対値が $|y_i|$ より小さくなるように、一意の符号付整数値 $int$ を求めます。

### fmodf4: Modulus Float

```
(vector float) fmodf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、以下のように定まる $x_i/y_i$ の剰余を収めた vector float を返します。

- $y_i$ が 0 の場合は、結果は 0 になります。
- そうでなければ、結果の要素が $x_i - int * y_i$ で、$x_i$ と同符号を持ち、絶対値が $|y_i|$ より小さくなるように、一意の符号付整数値 $int$ を求めます。

### fpclassifyd2: Classify Double (SPU Only)

```
(vector signed long long) fpclassifyd2 (vector double x);
```

vector double $x$ の各要素に対応する浮動小数点分類を収めた vector signed long long を返します。分類は `math.h` 中に定義されており、`FP_NAN`、`FP_INFINITE`、`FP_NORMAL`、`FP_SUBNORMAL`、および `FP_ZERO` です。

### fpclassifyf4: Classify Float

```
(vector signed int) fpclassifyf4 (vector float x);
```

vector float $x$ の各要素に対応する浮動小数点分類を収めた vector signed int を返します。分類は `math.h` 中に定義されており、`FP_NAN`、`FP_INFINITE`、`FP_NORMAL`、`FP_SUBNORMAL`、および `FP_ZERO` です。

SPU においては、結果のベクタに `FP_NAN` あるいは `FP_INFINITE` が含まれることはありません。

### frexpd2: Represent Double as Fraction and Exponent (SPU Only)

```
(vector double) frexpd2 (vector double x, vector signed long long *pexp);
```

正規化された小数部を収めた vector double が返され、指数整数を収めた vector signed long long が *pexp に格納されます。小数部の各要素 *frac* と指数整数の各要素 *exp* は、*x* の対応する要素の値を以下のように表します。

- |*frac*| は、区間 [1/2, 1) にあるか、あるいはゼロかのいずれかです。
- $x_i$ == frac * $2^{exp}$
- $x_i$ が 0 の場合は、*pexp の対応する要素もゼロになります。
- $x_i$ が NaN の場合は、対応する返り値は NaN になり、*pexp の対応する要素は不定となります。
- $x_i$ が無限大の場合は、対応する返り値は無限大になり、*pexp の対応する要素は不定となります。

### frexpf4: Represent Float as Fraction and Exponent

```
(vector float) frexpf4 (vector float x, vector signed int *pexp);
```

正規化された小数部を収めた vector float が返され、指数整数を収めた vector signed int が *pexp に格納されます。小数部の各要素 *frac* と指数整数の各要素 *exp* は、*x* の対応する要素の値を以下のように表します。

- |*frac*| は、区間 [1/2, 1) にあるか、あるいはゼロかのいずれかです。
- $x_i$ == frac * $2^{exp}$
- $x_i$ が 0 の場合は、*pexp の対応する要素もゼロになります。
- $x_i$ が NaN の場合は、対応する返り値は NaN になり、*pexp の対応する要素は不定となります。
- $x_i$ が無限大の場合は、対応する返り値は無限大になり、*pexp の対応する要素は不定となります。

### hypotd2: Hypotenuse Double (SPU Only)

```
(vector double) hypotd2 (vector double x, vector double y);
```

vector double *x* と vector double *y* の各要素に対する $x_i^2 + y_i^2$ の平方根を収めた vector double を、不必要なオーバフローやアンダフローを生じることなく返します。

### hypotf4: Hypotenuse Float

```
(vector float) hypotf4 (vector float x, vector float y);
```

vector float *x* と vector float *y* の各要素に対する $x_i^2 + y_i^2$ の平方根を収めた vector float を、不必要なオーバフローやアンダフローを生じることなく返します。

### ilogbd2: Integer Exponent of Double (SPU Only)

```
(vector signed long long) ilogbd2 (vector double x);
```

vector double *x* の各要素に対応して、以下で定義される要素を収めた vector signed long long を返します。

- $x_i$ が非数（NaN）の場合は、値はマクロ FP_ILOGBNAN となります。
- $x_i$ がゼロの場合は、値はマクロ FP_ILOGB0 となります。
- $x_i$ が正または負の Inf の場合は、値はマクロ INT_MAX となります。
- それ以外の場合は、(long long)logb($x_i$) を返します。

### ilogbf4: Integer Exponent of Float

```
(vector signed int) ilogbf4 (vector float x);
```

vector float *x* の各要素に対応して、以下で定義される要素を収めた vector signed int を返します。

- $x_i$ が非数（NaN）の場合は、値はマクロ FP_ILOGBNAN となります。
- $x_i$ がゼロの場合は、値はマクロ FP_ILOGB0 となります。
- $x_i$ が正または負の Inf の場合は、値はマクロ INT_MAX となります。
- それ以外の場合は、(int)logb($x_i$) を返します。

SPU では、単精度の Inf と NaN のビットパターンを通常の浮動小数点数値として扱うため、ilogbf4 はこれらの数に対する結果として 128 を返します。倍精度版の ilogb 関数との互換性のために、FP_ILOGBNAN は INT_MAX に設定されています。

### irintf4: Nearest Integer Float

```
(vector signed int) irintf4 (vector float x);
```

vector float $x$ の各要素に対して、現在の丸めモードに則って最も近い整数を収めた vector signed int を返します。丸め結果が戻り型の範囲外の場合には、数値結果は不定となります。

SPU においては、float に対する丸めモードは常にゼロ方向です。

### iroundf4: Round Float to Nearest Integer

```
(vector signed int) iroundf4 (vector float x);
```

vector float $x$ の各要素を丸めた整数値を収めた vector signed int を返します。

要素は最も近い値に丸められます。ちょうど中間の場合は、現在の丸め方向にかかわらず、ゼロから遠い方へ丸められます。

丸め結果が戻り型の範囲外の場合には、数値結果は不定となります。

### is0denormd2: 0 or Denormalized Double (SPU Only)

```
(vector unsigned long long) is0denormd2 (vector double x);
```

vector double $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が非正規化数値あるいはゼロの場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

### is0denormf4: 0 or Denormalized Float

```
(vector unsigned int) is0denormf4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が非正規化数値あるいはゼロの場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

### isequald2: Compare Equal Double (SPU Only)

```
(vector unsigned long long) isequald2 (vector double x, vector double y);
```

vector double x と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ と $y_i$ が等しい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。入力のいずれかが NaN の場合は、比較結果は偽（ゼロ）となります。入力の両方が同符号の Inf の場合は、これら入力は等しいと見なされます。0 と -0 は等しいと見なされます。

### isequalf4: Compare Equal Float

```
(vector unsigned int) isequalf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ と $y_i$ が等しい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。入力のいずれかが NaN の場合は、比較結果は偽（ゼロ）となります。入力の両方が同符号の Inf の場合は、これら入力は等しいと見なされます。0 と -0 は等しいと見なされます。

### isfinited2: Double is Finite (SPU Only)

```
(vector unsigned long long) isfinited2 (vector double x);
```

vector double $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が有限の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

### isfinitef4: Float is Finite

```
(vector unsigned int) isfinitef4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が有限の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

SPU においては、無限大の値は単精度では表現できません。したがって、$x_i$ の値にかかわらず結果の要素の全ビットが 1 にセットされます。

### isgreaterequald2: Greater or Equal Double (SPU Only)

```
(vector unsigned long long) isgreaterequald2 (vector double x, vector double y);
```

vector double `x` と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ と等しいか大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。入力のいずれかが `NaN` の場合は、比較結果は偽（ゼロ）となります。入力の両方が同符号の `Inf` の場合は、これら入力は等しいと見なされます。`0` と `-0` は等しいと見なされます。

### isgreaterequalf4: Greater or Equal Float

```
(vector unsigned int) isgreaterequalf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ と等しいか大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。入力のいずれかが `NaN` の場合は、比較結果は偽（ゼロ）となります。入力の両方が同符号の `Inf` の場合は、これら入力は等しいと見なされます。`0` と `-0` は等しいと見なされます。

### isgreaterd2: Greater than Double (SPU Only)

```
(vector unsigned long long) isgreaterd2 (vector double x, vector double y);
```

vector double `x` と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ より大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

### isgreaterf4: Greater than Float

```
(vector unsigned int) isgreaterf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ より大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

### isinfd2: Double is Infinity (SPU Only)

```
(vector unsigned long long) isinfd2 (vector double x);
```

vector double $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が無限大の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

### isinff4: Float is Infinity

```
(vector unsigned int) isinff4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が無限大の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

SPU においては、無限大の値は単精度では表現できません。したがって、$x_i$ の値にかかわらず結果の要素の全ビットがゼロにセットされます。

### islessd2: Double is Less Than (SPU Only)

```
(vector unsigned long long) islessd2 (vector double x, vector double y);
```

vector double x と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ より小さい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

### islessequald2: Double is Less Than or Equal To (SPU Only)

```
(vector unsigned long long) islessequald2 (vector double x, vector double y);
```

vector double x と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ より小さいか等しい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

### islessequalf4: Float is Less Than or Equal To

```
(vector unsigned int) islessequalf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ より小さいか等しい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

### islessf4: Float is Less Than

```
(vector unsigned int) islessf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ より小さい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

**islessgreaterd2: Double is Less Than or Greater Than (SPU Only)**

```
(vector unsigned long long) islessgreaterd2 (vector double x, vector double y);
```

vector double x と vector double y の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ より小さいあるいは大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

**islessgreaterf4: Float is Less Than or Greater Than**

```
(vector unsigned int) islessgreaterf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ より小さいあるいは大きい場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

この関数は、非正規化数も正しく比較します。

**isnand2: Double is NaN (SPU Only)**

```
(vector unsigned long long) isnand2 (vector double x);
```

vector double $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が NaN の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

**isnanf4: Float is NaN**

```
(vector unsigned int) isnanf4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が NaN の場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

SPU においては、NaN は単精度では表現できません。したがって、$x_i$ の値にかかわらず結果の要素の全ビットがゼロにセットされます。

**isnormald2: Double is Normal (SPU Only)**

```
(vector unsigned long long) isnormald2 (vector double x);
```

vector double $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が正規化数であって NaN でも無限大でもない場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

**isnormalf4: Float is Normal**

```
(vector unsigned int) isnormalf4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が正規化数であって NaN でも無限大でもない場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

### isunorderedd2: Double is Unordered (SPU Only)

```
(vector unsigned long long) isunorderedd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ が $y_i$ に対して順序付けられていない場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

NaN はいかなるオペランド（NaN 自身を含む）に対しても順序付けられません。

### isunorderedf4: Float is Unordered

```
(vector unsigned int) isunorderedf4 (vector float x, vector float y);
```

vector float $x$ と vectory float $y$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ が $y_i$ に対して順序付けられていない場合は、結果の要素の全ビットを 1 にセットします。
- それ以外ではゼロです。

NaN はいかなるオペランド（NaN 自身を含む）に対しても順序付けられません。SPU においては、単精度には NaN は存在しません。したがって、この関数は常にゼロを返すことになります。

### ldexpd2: Multiply Double by 2 Raised to its Power (SPU Only)

```
(vector double) ldexpd2 (vector double x, vector signed long long ex);
```

vector double $x$ と vector signed long long $ex$ の各要素に対する $x_i * 2^{ex_i}$ を収めた vector double を返します。$ex$ の要素の大きな値（オーバフロー）については、結果中の要素は適切な符号付の HUGE_VAL に飽和します。$ex$ の要素の小さな値（アンダフロー）については、結果の対応する要素は 0 になります。

### ldexpf4: Multiply Float by 2 Raised to its Power

```
(vector float) ldexpf4 (vector float x, vector signed int ex);
```

vector float $x$ と vector signed int $ex$ の各要素に対する $x_i * 2^{ex_i}$ を収めた vector float を返します。$ex$ の要素の大きな値（オーバフロー）については、結果中の要素は適切な符号付の HUGE_VALF に飽和します。$ex_i$ の小さな値（アンダフロー）については、結果の対応する要素は 0 になります。

### lgammad2: Natural Log of Gamma Function of Double (SPU Only)

```
(vector double) lgammad2 (vector double x);
```

vector double $x$ の各要素に対するガンマ関数の値の絶対値の自然対数を収めた vector double を返します。

### lgammaf4: Natural Log of Gamma Function of Float

```
(vector float) lgammaf4 (vector float x);
```

vector float $x$ の各要素に対するガンマ関数の値の絶対値の自然対数を収めた vector float を返します。

### llabsi2: Absolute Value Long Long (SPU Only)

```
(vector long long) llabsi2 (vector signed long long x);
```

vector signed long long $x$ の各要素の絶対値 $|x_i|$ を収めた vector long long を返します。

$x_i$ の絶対値が表現できない場合には、対応する結果は不定となり、エラーは報告されません。

### lldivi2: Divide Long Long (SPU Only)

```
(lldivi2_t) lldivi2 (vector signed long long x, vector signed long long y);
```

vector signed long long $x$ の各要素を vector signed long long $y$ の各要素で除算し、結果を `lldivi2_t` 型の構造体で返します。その中には、商のベクタと剰余のベクタとが入っています。

構造体メンバ *quot* 中のベクタの各要素は、ゼロ方向に切り捨てた算術商です。構造体メンバ *rem* 中のベクタの各要素は、対応する剰余であり、$x_i == quot * y_i + rem$ が成り立ちます。

$y_i$ がゼロの場合は、結果の商の対応する要素はゼロになります。

### lldivu2: Divide Unsigned Long Long (SPU Only)

```
(lldivu2_t) lldivu2 (vector unsigned long long x, vector unsigned long long y);
```

vector unsigned long long $x$ の各要素を vector unsigned long long $y$ の各要素で除算し、結果を `lldivu2_t` 型の構造体で返します。その中には、商のベクタと剰余のベクタとが入っています。

構造体メンバ *quot* 中のベクタの各要素は、ゼロ方向に切り捨てた算術商です。構造体メンバ *rem* 中のベクタの各要素は、対応する剰余であり、$x_i == quot * y_i + rem$ が成り立ちます。

$y_i$ がゼロの場合は、結果の商の対応する要素はゼロになります。

### llrintd2: Find Nearest Long Long of Double (SPU Only)

```
(vector signed long long) llrintd2 (vector double x);
```

vector double $x$ の各要素に対して、現在の丸めモードに則って最も近い long long 整数を収めた vector signed long long を返します。丸めた値が戻り型の範囲外の場合には、数値結果は不定となります。

### llrintf4: Find Nearest Long Long of Float (SPU Only)

```
(llroundf4_t) llrintf4 (vector float x);
```

vector float $x$ の各要素に対して、現在の丸めモードに則って最も近い long long 整数を収めた `llroundf4_t` 型を返します。丸めた値が戻り型の範囲外の場合には、数値結果は不定となります。

SPU においては、float に対する丸めモードは常にゼロ方向です。

### llroundd2: Round Double to Nearest Long Long (SPU Only)

```
(vector signed long long) llroundd2 (vector double x);
```

vector double $x$ の各要素を最も近い値に丸めたものを収めた vector signed long long を返します。ちょうど中間の場合は、現在の丸め方向にかかわらず、ゼロから遠い方へ丸められます。丸めた値が戻り型の範囲外の場合には、数値結果は不定となります。

### llroundf4: Round Float to Nearest Long Long (SPU Only)

```
(llroundf4_t) llroundf4 (vector float x);
```

vector float $x$ の各要素を最も近い値に丸めたものを収めた `llroundf4_t` 型構造体を返します。ちょうど中間の場合は、現在の丸め方向にかかわらず、ゼロから遠い方へ丸められます。丸めた値が戻り型の範囲外の場合には、数値結果は不定となります。

### logd2: Natural Log of Double (SPU Only)

```
(vector double) logd2 (vector double x);
```

vector double $x$ の各要素の自然対数を収めた vector double を返します。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

### logf4: Natural Log of Float

```
(vector float) logf4 (vector float x);
```

vector float $x$ の各要素の自然対数を収めた vector float を返します。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

$x_i$ がゼロの場合は、結果は -HUGE_VALF となります。

### log10d2: Log Base 10 of Double (SPU Only)

```
(vector double) log10d2 (vector double x);
```

vector double $x$ の各要素の 10 を底とする対数を収めた vector double を返します。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

### log10f4: Log Base 10 of Float

```
(vector float) log10f4 (vector float x);
```

vector float $x$ の各要素の 10 を底とする対数を収めた vector float を返します。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

$x_i$ がゼロの場合は、結果は -HUGE_VALF となります。

### log1pd2: Natural Log of Double Plus 1 (SPU Only)

```
(vector double) log1pd2 (vector double x);
```

vector double $x$ の各要素に対して、$1 + x_i$ の自然対数を収めた vector double を返します。

この関数は、$x$ の要素がゼロに近い場合でも数学的に正確な値を返します。

$x_i$ が -1 よりも小さい場合は、それに対応する結果は不定となり、エラーは報告されません。

### log1pf4: Natural Log of Float Plus 1

```
(vector float) log1pf4 (vector float x);
```

vector float $x$ の各要素に対して、$1 + x_i$ の自然対数を収めた vector float を返します。

この関数は、$x$ の要素がゼロに近い場合でも数学的に正確な値を返します。$x$ の要素が -1 の場合には -HUGE_VALF が返されます。

$x_i$ が -1 よりも小さい場合は、それに対応する結果は不定となり、エラーは報告されません。

### log2d2: Log Base 2 of Double (SPU Only)

```
(vector double) log2d2 (vector double x);
```

vector double $x$ の各要素の 2 を底とする対数を収めた vector double を返します。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

### log2f4: Log Base 2 of Float

```
(vector float) log2f4 (vector float x);
```

vector float $x$ の各要素の 2 を底とする対数を収めた vector float を返します。

$x_i$ がゼロの場合は -HUGE_VALF が返されます。

$x_i$ が負の場合は、それに対応する結果は不定となり、エラーは報告されません。

### logbd2: Represent Double as Fraction Greater Than 1 and Exponent (SPU Only)

```
(vector double) logbd2 (vector double x);
```

vector double $x$ の各要素について、有限の要素の値を表現する整数指数 $ex_i$ および小数部 $frac_i$ を求めます。$x_i$ について以下を満たす $ex_i$ の値を収めた vector double を返します。

- $x_i == frac * FLT\_RADIX^{ex_i}$
- $|frac|$ は [1, FLT_RADIX) の区間内

$x_i$ が 0 の場合は、それに対応する結果は不定となり、エラーは報告されません。

SPU においては、$x_i$ が 0 ならば対応結果は -HUGE_VALF、$x_i$ が無限大ならば対応結果は正の無限大、$x_i$ が NaN ならば対応結果も NaN となります。

### logbf4: Represent Float as Fraction Greater Than 1 and Exponent

```
(vector float) logbf4 (vector float x);
```

vector float $x$ の各要素について、有限の要素の値を表現する整数指数 $ex_i$ および小数部 $frac_i$ を求めます。$x_i$ について以下を満たす $ex_i$ の値を収めた vector float を返します。

- $x_i == frac * FLT\_RADIX^{ex_i}$
- $|frac|$ は [1, FLT_RADIX) の区間内

$x_i$ が 0 の場合は、それに対応する結果は不定となり、エラーは報告されません。

### modfd2: Represent Double as Proper Fraction and Exponent (SPU Only)

```
(vector double) modfd2 (vector double x, vector double *pint);
```

vector double $x$ の各要素を整数部 $int_i$ と小数部 $frac_i$ に分割します。対応する $frac_i$ 要素を収めた vector double を返し、もう一方の vector double を $*pint$ に格納します。そこには対応する $int_i$ 要素が収められており、以下を満足します。

- $x_i == frac + int$
- |frac| は [0, 1) の区間内
- frac と $int$ はともに $x_i$ と同じ符号を持つ

### modff4: Represent Float as Proper Fraction and Exponent

```
(vector float) modff4 (vector float x, vector float *pint);
```

vector float $x$ の各要素を整数部 $int_i$ と小数部 $frac_i$ に分割します。対応する $frac_i$ 要素を収めた vector float を返し、もう一方の vector float を $*pint$ に格納します。そこには対応する $int_i$ 要素が収められており、以下を満足します。

- $x_i == frac + int$
- |frac| は [0, 1) の区間内
- frac と $int$ はともに $x_i$ と同じ符号を持つ

### nearbyintd2: Find Nearest Integer for Double (SPU Only)

```
(vector double) nearbyintd2 (vector double x);
```

vector double $x$ の各要素を、現在の丸めモードに則って最も近い整数へ丸めたものを収めた vector double を返します。その際、不正確浮動小数点例外を引き起こすことはありません。

### nearbyintf4: Find Nearest Integer for Float

```
(vector float) nearbyintf4 (vector float x);
```

vector float $x$ の各要素を、現在の丸めモードに則って最も近い整数へ丸めたものを収めた vector float を返します。その際、不正確浮動小数点例外を引き起こすことはありません。

SPU においては、float の丸めモードは常にゼロ方向です。

### negated2: Negate Double (SPU Only)

```
(vector double) negated2 (vector double x);
```

vector double $x$ の各要素に対する $-x_i$ を収めた vector double を返します。

### negatef4: Negate Float

```
(vector float) negatef4 (vector float x);
```

vector float $x$ の各要素に対する $-x_i$ を収めた vector float を返します。

### negatei4: Negate Signed Integer

```
(vector signed int) negatei4 (vector signed int x);
```

vector signed int $x$ の各要素に対する $-x_i$ を収めた vector signed int を返します。

$-x_i$ が表現できない場合は、それに対応する結果は不定となり、エラーは報告されません。

### negatell2: Negate Signed Long Long Integer (SPU Only)

```
(vector signed long long) negatell2 (vector signed long long x);
```

vector signed long long $x$ の各要素に対する $-x_i$ を収めた vector signed long long を返します。

$-x_i$ が表現できない場合は、それに対応する結果は不定となり、エラーは報告されません。

### nextafterd2: Find Next Integer After for Double (SPU Only)

```
(vector double) nextafterd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、$y_i$ に近づく方向で $x_i$ の次の表現可能な値を収めた vector double を返します。$x_i$ と $y_i$ が等しい場合は、$y_i$ を返します。

$x_i$ の絶対値が表現可能な最大有限値である場合は、結果は不定となります。

### nextafterf4: Find Next Integer After for Float

```
(vector float) nextafterf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、$y_i$ に近づく方向で $x_i$ の次の表現可能な値を収めた vector float を返します。$x_i$ と $y_i$ が等しい場合は、$y_i$ を返します。

$x_i$ の絶対値が表現可能な最大有限値である場合は、結果は不定となります。

### powd2: Raise Double to Double Power (SPU Only)

```
(vector double) powd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、$x_i$ を $y_i$ 乗した $x^{y}{}_{i}$ を収めた vector double を返します。

$x_i$ が有限の負数であり、かつ $y_i$ が有限の非整数値の場合には、それに対応する結果は不定となり、エラーは報告されません。

**powf4: Raise Float to Float Power**

```
(vector float) powf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、$x_i$ を $y_i$ 乗した $x^{y}{}_i$ を収めた vector float を返します。

SPU においては、結果が HUGE_VALF を上回ってしまう場合には、結果は HUGE_VALF に飽和し、エラーは報告されません。

**recipd2: Reciprocal of Double (SPU Only)**

```
(vector double) recipd2 (vector double x);
```

vector double $x$ の各要素の逆数を収めた vector double を返します。

この関数では特殊ケースを以下のように扱います。

- $x_i$ が ±Inf の場合は、結果は $x_i$ の符号を付けた 0 となります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号を付けた Inf となります。
- $x_i$ が NaN 場合は、結果は NaN となります。

**recipf4: Reciprocal of Float**

```
(vector float) recipf4 (vector float x);
```

vector float $x$ の各要素の逆数を収めた vector float を返します。

この関数では特殊ケースを以下のように扱います。

- $x_i$ が ±Inf の場合は、結果は $x_i$ の符号を付けた 0 となります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号を付けた Inf となります。
- $x_i$ が NaN 場合は、結果は NaN となります。

**remainderd2: Remainder of Doubles (SPU Only)**

```
(vector double) remainderd2 (vector double x, vector double y);
```

vector double $x$ と vector double $y$ の各要素に対して、剰余 $x_i$ REM $y_i$ を収めた vector double を返します。

$y_i$ がゼロの場合には、対応する結果の要素は不定となり、エラーは報告されません。

**remainderf4: Remainder of Floats**

```
(vector float) remainderf4 (vector float x, vector float y);
```

vector float $x$ と vector float $y$ の各要素に対して、剰余 $x_i$ REM $y_i$ を収めた vector float を返します。

$y_i$ がゼロの場合には、対応する結果の要素は不定となり、エラーは報告されません。

**remquod2: Remainder Function of Double (SPU Only)**

```
(vector double) remquod2 (vector double x, vector double y, vector signed long long *pquo);
```

この関数は remainderd2() と同一の vector double の結果を返します。更に、*pquo に vector signed long long を格納し、その対応要素の値は、符号が $x_i$ / $y_i$ の符号で、絶対値が $x_i$ / $y_i$ の整数商の絶対値のモジュロ $2^n$ となります。ここで、$n$ は実装定義の 3 以上の整数です。

**remquof4: Remainder Function of Float**

```
(vector float) remquof4 (vector float x, vector float y, vector signed int *pquo);
```

この関数は remainderf4() と同一の vector float の結果を返します。更に、*pquo に vector signed int を格納し、その対応要素の値は、符号が $x_i$ / $y_i$ の符号で、絶対値が $x_i$ / $y_i$ の整数商の絶対値のモジュロ $2^n$ となります。ここで、$n$ は実装定義の 3 以上の整数です。

### rintd2: Round Double to the Nearest Integer (SPU Only)

```
(vector double) rintd2 (vector double x);
```

vector double $x$ の各要素を、現在の丸めモードに則って最も近い整数に丸めたものを収めた vector double を返します。

### rintf4: Round Float to the Nearest Integer

```
(vector float) rintf4 (vector float x);
```

vector float $x$ の各要素を、現在の丸めモードに則って最も近い整数に丸めたものを収めた vector float を返します。

SPU においては、float の丸めモードは常にゼロ方向です。

### roundd2: Round Double (SPU Only)

```
(vector double) roundd2 (vector double x);
```

vector double $x$ の各要素を丸めたものを収めた vector double を返します。丸めは、最も近い整数値に向けて、浮動小数点形式にて行ないます。ちょうど中間の場合は、現在の丸め方向にかかわらず、ゼロから遠い方へ丸めます。

### roundf4: Round Float

```
(vector float) roundf4 (vector float x);
```

vector float $x$ の各要素を丸めたものを収めた vector float を返します。丸めは、最も近い整数値に向けて、浮動小数点形式にて行ないます。ちょうど中間の場合は、現在の丸め方向にかかわらず、ゼロから遠い方へ丸めます。

### rsqrtd2: Reciprocal Square Root of Double (SPU Only)

```
(vector double) rsqrtd2 (vector double x);
```

vector double $x$ の各要素に対して、$x_i$ の平方根の逆数を収めた vector double を返します。特殊ケースは以下のように扱います。

- $x_i$ が 0 より小さい場合は、結果は NaN になります。
- $x_i$ が+Inf の場合は、結果は+0 になります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号をつけた Inf になります。
- $x_i$ が NaN の場合は、結果は NaN になります。

### rsqrtf4: Reciprocal Square Root of Float

```
(vector float) rsqrtf4 (vector float x);
```

vector float $x$ の各要素に対して、$x_i$ の平方根の逆数を収めた vector float を返します。特殊ケースは以下のように扱います。

- $x_i$ が 0 より小さい場合は、結果は NaN になります。
- $x_i$ が+Inf の場合は、結果は+0 になります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号をつけた Inf になります。
- $x_i$ が NaN の場合は、結果は NaN になります。

SPU においては、$x_i$ が 0 より小さい場合は、結果は不定となります。

### scalblInd2: Scale Double by Long Long Integer (SPU Only)

```
(vector double) scalbllnd2 (vector double x, vector signed long long n);
```

vector double $x$ と vector signed long long $n$ の各要素に対して、$x_i$ に $2^n$ を効率よく乗じたものを収めた vector double を返します。

### scalbnf4: Scale Float by Integer

```
(vector float) scalbnf4 (vector float x, vector signed int n);
```

vector float $x$ と vector signed int $n$ の各要素に対して、$x_i$ に $2^n$ を効率よく乗じたものを収めた vector float を返します。

### signbitd2: Sign Bit of Double (SPU Only)

```
(vector unsigned long long) signbitd2 (vector double x);
```

vector double x の各要素に対して、以下で定義される要素を収めた vector unsigned long long を返します。

- $x_i$ の符号ビットがセットされている場合は、結果の要素の全ビットを 1 にセットします。
- さもなければゼロです。

### signbitf4: Sign Bit of Float

```
(vector unsigned int) signbitf4 (vector float x);
```

vector float $x$ の各要素に対して、以下で定義される要素を収めた vector unsigned int を返します。

- $x_i$ の符号ビットがセットされている場合は、結果の要素の全ビットを 1 にセットします。
- さもなければゼロです。

### sincosd2: Sine and Cosine of Double (SPU Only)

```
(void) sincosd2 (vector double x, vector double *sx, vector double *cx);
```

vector double $x$ の各要素に対する正弦（サイン）と余弦（コサイン）をそれぞれ収めた、vector double を $*sx$ に、vector double を $*cx$ に格納します。

sincosd2() の結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### sincosf4: Sine and Cosine of Float

```
(void) sincosf4 (vector float x, vector float *sx, vector float *cx);
```

vector float $x$ の各要素に対する正弦（サイン）と余弦（コサイン）をそれぞれ収めた、vector float を $*sx$ に、vector float を $*cx$ に格納します。

sincosf4() の結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### sind2: Sine of Double (SPU Only)

```
(vector double) sind2 (vector double x);
```

vector double $x$ の各要素に対応する正弦（サイン）を収めた vector double を返します。

sind2() の結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### sinf4: Sine of Float

```
(vector float) sinf4 (vector float x);
```

vector float $x$ の各要素に対応する正弦（サイン）を収めた vector float を返します。

sinf4() の結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### sinhd2: Hyperbolic Sine of Double (SPU Only)

```
(vector double) sinhd2 (vector double x);
```

vector double $x$ の各要素に対応する双曲線正弦（ハイパボリックサイン）を収めた vector double を返します。

### sinhf4: Hyperbolic Sine of Float

```
(vector float) sinhf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線正弦（ハイパボリックサイン）を収めた vector float を返します。

SPU においては、結果の要素で HUGE_VALF よりも大きなものは HUGE_VALF として返され、エラーは報告されません。

### sqrtd2: Square Root of Double (SPU Only)

```
(vector double) sqrtd2 (vector double x);
```

vector double $x$ の各要素に対する実数平方根 $x^{1/2}{}_i$ を収めた vector double を返します。

この関数では特殊ケースを以下のように扱います。

- $x_i$ が 0 より小さい場合は、結果は NaN になります。
- $x_i$ が+Inf の場合は、結果は+Inf になります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号がついた 0 になります。
- $x_i$ が NaN の場合は、結果は NaN になります。

### sqrtf4: Square Root of Float

```
(vector float) sqrtf4 (vector float x);
```

vector float $x$ の各要素に対する実数平方根 $x^{1/2}{}_i$ を収めた vector float を返します。

この関数では特殊ケースを以下のように扱います。

- $x_i$ が 0 より小さい場合は、結果は NaN になります。
- $x_i$ が+Inf の場合は、結果は+Inf になります。
- $x_i$ が 0 の場合は、結果は $x_i$ の符号がついた 0 になります。
- $x_i$ が NaN の場合は、結果は NaN になります。

SPU においては、$x_i$ が負の場合は結果は不定となります。

### tand2: Tangent of Double (SPU Only)

```
(vector double) tand2 (vector double x);
```

vector double $x$ の各要素に対応する正接（タンジェント）を収めた vector double を返します。

結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### tanf4: Tangent of Float

```
(vector float) tanf4 (vector float x);
```

vector float $x$ の各要素に対応する正接（タンジェント）を収めた vector float を返します。

結果は $x_i$ の非常に大きな値に対しては正確ではないかもしれませんが、エラーは報告されません。実装においては、精度が失われる地点を明記すべきです。

### tanhd2: Hyperbolic Tangent of Double (SPU Only)

```
(vector double) tanhd2 (vector double x);
```

vector double $x$ の各要素に対応する双曲線正接（ハイパボリックタンジェント）を収めた vector double を返します。

### tanhf4: Hyperbolic Tangent of Float

```
(vector float) tanhf4 (vector float x);
```

vector float $x$ の各要素に対応する双曲線正接（ハイパボリックタンジェント）を収めた vector float を返します。

### tgammad2: Gamma of Double (SPU Only)

```
(vector double) tgammad2 (vector double x);
```

vector double $x$ の各要素にガンマ関数を適用した結果を収めた vector double を返します。

$x_i$ が負の整数の場合は、それに対応する結果の要素は不定となり、エラーは報告されません。

### tgammaf4: Gamma of Float

```
(vector float) tgammaf4 (vector float x);
```

vector float $x$ の各要素にガンマ関数を適用した結果を収めた vector float を返します。

$x_i$ が負の整数の場合は、それに対応する結果の要素は不定となり、エラーは報告されません。

### truncd2: Truncate Double (SPU Only)

```
(vector double) truncd2 (vector double x);
```

vector double $x$ の各要素に対して、$x_i$ を絶対値で $x_i$ よりも大きくない最も近い整数に丸めた（ゼロ方向へ丸めた）ものを収めた vector double を返します。

### truncf4: Truncate Float

```
(vector float) truncf4 (vector float x);
```

vector float x の各要素に対して、$x_i$ を絶対値で $x_i$ よりも大きくない最も近い整数に丸めた（ゼロ方向へ丸めた）ものを収めた vector float を返します。

# 索引

**End of Document**