



---

# SIMD Math Library Specification for Cell Broadband Engine Architecture

---

Version 1.1

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

September 14, 2007



© Copyright International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation 2006 - 2007

All Rights Reserved

Printed in the United States of America November 2007

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.

IBM	PowerPC
IBM Logo	PowerPC Architecture
ibm.com	

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc.

Other company, product and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Systems and Technology Group  
2070 Route 52, Bldg. 330  
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**

The IBM semiconductor solutions home page can be found at **ibm.com/chips**

September 14, 2007



# Table of Contents

List of Figures	vi
List of Tables	vi
About This Document	vii
Audience	vii
Version History	vii
Related Documentation	vii
Document Structure	vii
Conventions Used in This Document	viii
1. Overview of the SIMD Math Library	1
1.1. Library and Header Files	1
1.2. Functions Overview	1
1.3. Special Cases	6
1.3.1. Rounding	6
1.3.2. Special Operands	6
1.3.3. Error Conditions	6
1.3.4. Exceptions	6
2. SIMD Function Specifications	8
2.1. Type Definitions	8
divi4_t: Remainder/Quotient Struct for Vector Signed Int	8
divu4_t: Remainder/Quotient Struct for Vector Unsigned Int	8
lldivi2_t: Remainder/Quotient Struct for Vector Signed Long Long (SPU Only)	8
lldivu2_t: Remainder/Quotient Struct for Vector Unsigned Long Long (SPU Only)	8
llroundf4_t: Vector of Four Long Long (SPU Only)	8
2.2. Function Descriptions	9
absi4: Absolute Value of Integer	9
acosd2: Arccosine of Double (SPU Only)	9
acosf4: Arccosine of Float	9
acoshd2: Hyperbolic Arccosine of Double (SPU Only)	9
acoshf4: Hyperbolic Arccosine of Float	9
asind2: Arcsine of Double (SPU Only)	9
asinf4: Arcsine of Float	9
asinhd2: Hyperbolic Arcsine of Double (SPU Only)	10
asinhf4: Hyperbolic Arcsine of Float	10
atand2: Tangent of Double (SPU Only)	10
atanf4: Tangent of Float	10
atanhd2: Hyperbolic Arctangent of Double (SPU Only)	10
atanhf4: Hyperbolic Arctangent of Float	10
atan2d2: Arctangent of Double Quotient (SPU Only)	10
atan2f4: Arctangent of Float Quotient	10
cbrtd2: Cube Root of Double (SPU Only)	11
cbrtf4: Cube Root of Float	11
ceild2: Ceiling of Double (SPU Only)	11
ceilf4: Ceiling of Float	11
copysignd2: Copy Sign of Double (SPU Only)	11
copysignf4: Copy Sign of Float	11
cosd2: Cosine of Double (SPU Only)	11
cosf4: Cosine of Float	12
coshd2: Hyperbolic Cosine of Double (SPU Only)	12
coshf4: Hyperbolic Cosine of Float	12
divd2: Divide Doubles (SPU Only)	12
divf4: Divide Floats	12
divi4: Divide Integer	12
divu4: Divide Unsigned Integer	13
erfcd2: Complementary Error Function Double (SPU Only)	13

erfcf4: Complementary Error Function Float	13
erfd2: Error Function Double (SPU Only)	13
erff4: Error Function Float	13
expd2: $e$ Raised to the Power of Double (SPU Only)	13
expf4: $e$ Raised to the Power of Float	13
exp2d2: 2 Raised to the Power of Double (SPU Only)	13
exp2f4: 2 Raised to the Power of Float	13
expm1d2: $e$ Raised to the Power of Double Minus 1 (SPU Only)	14
expm1f4: $e$ Raised to the Power of Float Minus 1	14
fabsd2: Absolute Value Double (SPU Only)	14
fabsf4: Absolute Value Float	14
fdimd2: Subtract Staying Non-Negative Double (SPU Only)	14
fdimf4: Subtract Staying Non-Negative Float	14
floorf4: Floor Float	14
fmad2: Fused Multiply and Add Double (SPU Only)	15
fmaf4: Fused Multiply and Add Float	15
fmaxd2: Maximum Double (SPU Only)	15
fmaxf4: Maximum Float	15
fmind2: Minimum Double (SPU Only)	15
fminf4: Minimum Float	15
fmodd2: Modulus Double (SPU Only)	15
fmodf4: Modulus Float	16
fpclassifyd2: Classify Double (SPU Only)	16
fpclassifyf4: Classify Float	16
frexp2d2: Represent Double as Fraction and Exponent (SPU Only)	16
frexp4: Represent Float as Fraction and Exponent	16
hypotd2: Hypotenuse Double (SPU Only)	17
hypotf4: Hypotenuse Float	17
ilogbd2: Integer Exponent of Double (SPU Only)	17
ilogbf4: Integer Exponent of Float	17
irintf4: Nearest Integer Float	17
iroundf4: Round Float to Nearest Integer	17
is0denormd2: 0 or Denormalized Double (SPU Only)	18
is0denormf4: 0 or Denormalized Float	18
isequald2: Compare Equal Double (SPU Only)	18
isequalf4: Compare Equal Float	18
isfinitd2: Double Is Finite (SPU Only)	18
isfinitf4: Float Is Finite	18
isgreaterd2: Greater or Equal Double (SPU Only)	19
isgreaterd4: Greater or Equal Float	19
isgreaterd2: Greater Than Double (SPU Only)	19
isgreaterf4: Greater Than Float	19
isinf2: Double Is Infinity (SPU Only)	20
isinf4: Float Is Infinity	20
islessd2: Double Is Less Than (SPU Only)	20
islessequald2: Double Is Less Than or Equal To (SPU Only)	20
islessequalf4: Float Is Less Than or Equal To	20
islessf4: Float Is Less Than	21
islessgreaterd2: Double Is Less Than or Greater Than (SPU Only)	21
islessgreaterf4: Float Is Less Than or Greater Than	21
isnand2: Double Is NaN (SPU Only)	21
isnanf4: Float Is NaN	21
isnormald2: Double Is Normal (SPU Only)	22
isnormalf4: Float Is Normal	22
isunorderedd2: Double Is Unordered (SPU Only)	22
isunorderedf4: Float Is Unordered	22
ldexpd2: Multiply Double by 2 Raised to its Power (SPU Only)	22
ldexpf4: Multiply Float by 2 Raised to its Power	22
lgammd2: Natural Log of Gamma Function of Double (SPU Only)	23
lgammaf4: Natural Log of Gamma Function of Float	23
llabsi2: Absolute Value Long Long (SPU Only)	23

lldivi2: Divide Long Long (SPU Only)	23
lldivu2: Divide Unsigned Long Long (SPU Only)	23
llrintd2: Find Nearest Long Long of Double (SPU Only)	23
llrintf4: Find Nearest Long Long of Float (SPU Only)	23
llroundd2: Round Double to Nearest Long Long (SPU Only)	24
llroundf4: Round Float to Nearest Long Long (SPU Only)	24
logd2: Natural Log of Double (SPU Only)	24
logf4: Natural Log of Float	24
log10d2: Log Base 10 of Double (SPU Only)	24
log10f4: Log Base 10 of Float	24
log1pd2: Natural Log of Double Plus 1 (SPU Only)	24
log1pf4: Natural Log of Float Plus 1	25
log2d2: Log Base 2 of Double (SPU Only)	25
log2f4: Log Base 2 of Float	25
logbd2: Represent Double as Fraction Greater Than 1 and Exponent (SPU Only)	25
logbf4: Represent Float as Fraction Greater Than 1 and Exponent	25
modfd2: Represent Double as Proper Fraction and Exponent (SPU Only)	25
modff4: Represent Float as Proper Fraction and Exponent	26
nearbyintd2: Find Nearest Integer for Double (SPU Only)	26
nearbyintf4: Find Nearest Integer for Float	26
negated2: Negate Double (SPU Only)	26
negatef4: Negate Float	26
negatei4: Negate Signed Integer	26
negatell2: Negate Signed Long Long Integer (SPU Only)	26
nextafterd2: Find Next Integer After for Double (SPU Only)	27
nextafterf4: Find Next Integer After for Float	27
powd2: Raise Double to Double Power (SPU Only)	27
powf4: Raise Float to Float Power	27
recipd2: Reciprocal of Double (SPU Only)	27
recipf4: Reciprocal of Float	27
remainderd2: Remainder of Doubles (SPU Only)	28
remainderf4: Remainder of Floats	28
remquod2: Remainder Function of Double (SPU Only)	28
remquof4: Remainder Function of Float	28
rintd2: Round Double to the Nearest Integer (SPU Only)	28
rintf4: Round Float to the Nearest Integer	28
roundd2: Round Double (SPU Only)	28
roundf4: Round Float	29
rsqrt2: Reciprocal Square Root of Double (SPU Only)	29
rsqrtf4: Reciprocal Square Root of Float	29
scalblnd2: Scale Double by Long Long Integer (SPU Only)	29
scalbnf4: Scale Float by Integer	29
signbitd2: Sign Bit of Double (SPU Only)	29
signbitf4: Sign Bit of Float	30
sincosd2: Sine and Cosine of Double (SPU Only)	30
sincosf4: Sine and Cosine of Float	30
sind2: Sine of Double (SPU Only)	30
sinf4: Sine of Float	30
sinhd2: Hyperbolic Sine of Double (SPU Only)	30
sinhf4: Hyperbolic Sine of Float	30
sqrtd2: Square Root of Double (SPU Only)	31
sqrtf4: Square Root of Float	31
tand2: Tangent of Double (SPU Only)	31
tanf4: Tangent of Float	31
tanh2: Hyperbolic Tangent of Double (SPU Only)	31
tanhf4: Hyperbolic Tangent of Float	31
tgammd2: Gamma of Double (SPU Only)	32
tgammaf4: Gamma of Float	32
truncd2: Truncate Double (SPU Only)	32
truncf4: Truncate Float	32



## List of Figures

Figure 1: Big-Endian Byte/Element Ordering for Vector Types

viii

## List of Tables

Table 1: SIMD Math Functions

1



## About This Document

This document contains specifications for a math library that takes advantage of the single instruction, multiple data (SIMD) instructions provided by the PowerPC<sup>®</sup> Processor Unit (PPU) and the Synergistic Processor Unit (SPU) hardware of the Cell Broadband Engine™. By computing multiple results at one time, SIMD math functions allow programmers to obtain much higher performance from their PPU and SPU programs than would be possible from a corresponding traditional scalar math library.

## Audience

This document is intended for system and application programmers who are interested in writing high-performance programs for the Cell Broadband Engine.

## Version History

This section describes significant changes made to each version of this document.

Version Number & Date	Changes
v. 1.1 September 14, 2007	Corrected chapter numbering and other minor errors (TWG_RFC00093-0: CORRECTION NOTICE), TWG_RFC00116-0: CORRECTION NOTICE). Made miscellaneous editorial changes. Specified the behavior of <code>atanhf4()</code> when it is given an argument that has a value of -1 (TWG_RFC00120-1). Changed the way that several functions are described and classified (TWG_RFC00122-0: CORRECTION NOTICE). Made miscellaneous editorial changes.
v. 1.0 November 6, 2006	Created the initial document.

## Related Documentation

The following table provides a list of references and supporting materials for this document:

Document Title	Version	Date
<i>C/C++ Language Extensions for the Cell Broadband Engine Architecture</i>	2.5	September 2007
<i>ISO/IEC Standard 9899:1999 (C Standard)</i>		
<i>IEC Standard 60559:1989 (Standard for Binary Floating-Point Arithmetic)</i>		
<i>Synergistic Processor Unit Instruction Set Architecture</i>	1.2	January 2007
<i>PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual,</i>	2.07c	October 2006
<i>PPE User's Manual</i>	1.0	
<i>PowerPC Architecture Book</i>	2.02	November 2005

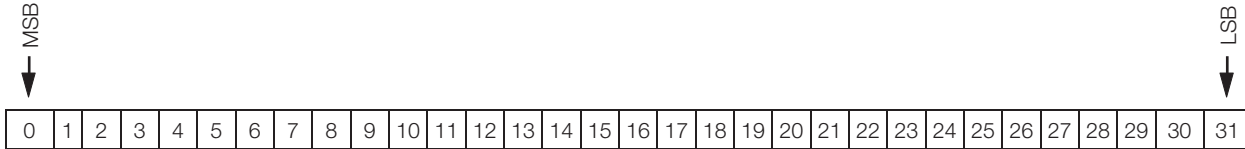
## Document Structure

This document contains two chapters. The first is a SIMD math library overview, and the second is a specification describing the particular math functions in this library.

## Conventions Used in This Document

### Bit Notation

Standard bit notation is used throughout this document. Bits and bytes are numbered in ascending order from left to right. Thus, for a 4-byte word, bit 0 is the most significant bit and bit 31 is the least significant bit, as shown in the following figure:



MSB = Most significant bit

LSB = Least significant bit

Notation for bit encoding is as follows:

- Hexadecimal values are preceded by 0x. For example: 0x0A00.
- Binary values in sentences appear in single quotation marks. For example: '1010'.

### Byte Ordering and Element Numbering

Byte ordering and element/slot numbering are always displayed in big-endian order, as shown in Figure 1.

Figure 1: Big-Endian Byte/Element Ordering for Vector Types

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

### Other Conventions

The following typographic conventions are used throughout this document:

Convention	Meaning
<i>courier</i>	Indicates programming code and literals, such as processing instructions, register names, data types, events, and file names. Also indicates function and macro names. This convention is used only where it facilitates comprehension, especially in narrative descriptions.
<i>courier + italics</i>	Indicates arguments, parameters, and variables. This convention is used only where it facilitates comprehension, especially in narrative descriptions.
<i>italics (without courier)</i>	Indicates emphasis. Except when hyperlinked, book references are in italics. When a term is first defined, it is often in italics.





Convention	Meaning
<a href="#">blue</a>	Indicates a hyperlink (color printers or online only).





## 1. Overview of the SIMD Math Library

The PPU and SPU instruction sets include single instruction, multiple data (SIMD) instructions that are similar to normal instructions but operate on more than one input simultaneously. Traditional math functions operate on a single input and are unable to take advantage of the speed and power of SIMD instructions. The SIMD Math Library contains SIMD versions of the scalar math functions described in the C99 standard, or *ISO/IEC Standard 9899:1999* (*C Standard*). This chapter provides specifications for these special PPU and SPU SIMD libraries.

### 1.1. Library and Header Files

The name of the SIMD library will contain the string `simdmath`. For example, on GNU/Linux the library will be called `libsimdmath.a`, or `libsimdmath.so` (for the shared library version). The `simdmath.h` system header file will contain type declarations and prototypes for the SIMD math functions.

### 1.2. Functions Overview

The functions that comprise the PPU and SPU SIMD math libraries are listed in Table 1. The functions that are listed as “non-standard” have no C99 counterpart.

Names of the SIMD math functions are differentiated from their scalar counterparts by a vector type suffix appended to the standard scalar function name. For example, the SIMD version of `fabsf()`, which acts on a vector float, is called `fabsf4()`. Similarly, a SIMD version of a standard scalar function that acts on a vector double will have `d2` appended to the name.

Table 1: SIMD Math Functions

Function Name	C99 Name	Function Category	Precision	SPU/PPU
<code>absi4</code>	<code>abs</code>	integer	word	Both
<code>acosd2</code>	<code>acos</code>	trig	double	SPU
<code>acosf4</code>	<code>acosf</code>	trig	single	Both
<code>acoshd2</code>	<code>acosh</code>	hyperbolic	double	SPU
<code>acoshf4</code>	<code>acoshf</code>	hyperbolic	single	Both
<code>asind2</code>	<code>asin</code>	trig	double	SPU
<code>asinf4</code>	<code>asinf</code>	trig	single	Both
<code>asinhd2</code>	<code>asinh</code>	hyperbolic	double	SPU
<code>asinhf4</code>	<code>asinhf</code>	hyperbolic	single	Both
<code>atan2d2</code>	<code>atan2</code>	trig	double	SPU
<code>atan2f4</code>	<code>atan2f</code>	trig	single	Both
<code>atand2</code>	<code>atan</code>	trig	double	SPU
<code>atanf4</code>	<code>atanf</code>	trig	single	Both
<code>atanhd2</code>	<code>atanh</code>	hyperbolic	double	SPU
<code>atanhf4</code>	<code>atanhf</code>	hyperbolic	single	Both
<code>cbrtd2</code>	<code>cbrt</code>	power	double	SPU
<code>cbrtf4</code>	<code>cbrtf</code>	power	single	Both
<code>ceild2</code>	<code>ceil</code>	rounding	double	SPU
<code>ceilf4</code>	<code>ceilf</code>	rounding	single	Both
<code>copysignd2</code>	<code>copysign</code>	other	double	SPU
<code>copysignf4</code>	<code>copysignf</code>	other	single	Both

Function Name	C99 Name	Function Category	Precision	SPU/PPU
cosd2	cos	trig	double	SPU
cosf4	cosf	trig	single	Both
coshd2	cosh	hyperbolic	double	SPU
coshf4	coshf	hyperbolic	single	Both
divd2	non-standard	divide	double	SPU
divf4	non-standard	divide	single	Both
divi4	div	integer	word	Both
divu4	non-standard	integer	word	Both
erfcd2	erfc	error function	double	SPU
erfcf4	erfcf	error function	single	Both
erfd2	erf	error function	double	SPU
erff4	erff	error function	single	Both
exp2d2	exp2	exp	double	SPU
exp2f4	exp2f	exp	single	Both
expd2	exp	exp	double	SPU
expf4	expf	exp	single	Both
expm1d2	expm1	exp	double	SPU
expm1f4	expm1f	exp	single	Both
fabsd2	fabs	abs	double	SPU
fabsf4	fabsf	abs	single	Both
fdimd2	fdim	other	double	SPU
fdimf4	fdimf	other	single	Both
floord2	floor	rounding	double	SPU
floorf4	floorf	rounding	single	Both
fmad2	fma	other	double	SPU
fmaf4	fmaf	other	single	Both
fmaxd2	fmax	minmax	double	SPU
fmaxf4	fmaxf	minmax	single	Both
fmind2	fmin	minmax	double	SPU
fminf4	fminf	minmax	single	Both
fmodd2	fmod	modrem	double	SPU
fmodf4	fmodf	modrem	single	Both
fpclassifyd2	fpclassify	comparison	double	SPU
fpclassifyf4	fpclassify	comparison	single	Both
frexp2d2	frexp	other	double	SPU
frexp4	frexp	other	single	Both
hypotd2	hypot	other	double	SPU
hypotf4	hypotf	other	single	Both
ilogbd2	ilogb	log	double	SPU
ilogbf4	ilogbf	log	single	Both
rintf4	non-standard	other	single	Both
iroundf4	non-standard	rounding	single	Both
is0denormf4	non-standard	comparison	single	Both

Function Name	C99 Name	Function Category	Precision	SPU/PPU
is0denormd2	non-standard	comparison	double	SPU
isequald2	non-standard	comparison	double	SPU
isequalf4	non-standard	comparison	single	Both
isfinitd2	isfinite	comparison	double	SPU
isfinitf4	isfinite	comparison	single	Both
isgreaterd2	isgreater	comparison	double	SPU
isgreaterf4	isgreater	comparison	single	Both
isgreaterequald2	isgreaterequal	comparison	double	SPU
isgreaterequalf4	isgreaterequal	comparison	single	Both
islessd2	isless	comparison	double	SPU
islessf4	isless	comparison	single	Both
islessequald2	islessequal	comparison	double	SPU
islessequalf4	islessequal	comparison	single	Both
islessgreaterd2	islessgreater	comparison	double	SPU
islessgreaterf4	islessgreater	comparison	single	Both
isunorderedd2	isunordered	comparison	double	SPU
isunorderedf4	isunordered	comparison	single	Both
isinf2	isinf	comparison	double	SPU
isinf4	isinf	comparison	single	Both
isnand2	isnan	comparison	double	SPU
isnand4	isnan	comparison	single	Both
isnormald2	isnormal	comparison	double	SPU
isnormalf4	isnormal	comparison	single	Both
ldexpd2	ldexp	other	double	SPU
ldexpf4	ldexpf	other	single	Both
lgammad2	lgamma	gamma	double	SPU
lgammaf4	lgammaf	gamma	single	Both
llabsi2	llabs	integer	doubleword	SPU
lldivi2	lldiv	integer	doubleword	SPU
lldivu2	non-standard	integer	doubleword	SPU
llrintd2	llrint	rounding	double	SPU
llrintf4	llrintf	rounding	single	SPU
llroundd2	llround	rounding	double	SPU
llroundf4	llroundf	rounding	single	SPU
log10d2	log10	log	double	SPU
log10f4	log10f	log	single	Both
log1pd2	log1p	log	double	SPU
log1pf4	log1pf	log	single	Both
log2d2	log2	log	double	SPU
log2f4	log2f	log	single	Both
logbd2	logb	log	double	SPU
logbf4	logbf	log	single	Both
logd2	log	log	double	SPU

Function Name	C99 Name	Function Category	Precision	SPU/PPU
logf4	logf	log	single	Both
modfd2	modf	modrem	double	SPU
modff4	modff	modrem	single	Both
nearbyintd2	nearbyint	rounding	double	SPU
nearbyintf4	nearbyintf	rounding	single	Both
negated2	non-standard	other	double	SPU
negatef4	non-standard	other	single	Both
negatei4	non-standard	integer	word	Both
negatell2	non-standard	integer	doubleword	SPU
nextafterd2	nextafter	rounding	double	SPU
nextafterf4	nextafterf	rounding	single	Both
powd2	pow	power	double	SPU
powf4	powf	power	single	Both
recipd2	non-standard	recip	double	SPU
recipf4	non-standard	recip	single	Both
remainderd2	remainder	modrem	double	SPU
remainderf4	remainderf	modrem	single	Both
remquod2	remquo	modrem	double	SPU
remquof4	remquof	modrem	single	Both
rintd2	rint	rounding	double	SPU
rintf4	rintf	rounding	single	Both
roundd2	round	rounding	double	SPU
roundf4	roundf	rounding	single	Both
rsqrtd2	non-standard	sqrt	double	SPU
rsqrtf4	non-standard	sqrt	single	Both
scalblnd2	non-standard	other	double	SPU
scalbnf4	scalbnf	other	single	Both
signbitf4	signbit	comparison	single	Both
signbitd2	signbit	comparison	double	SPU
sincosd2	non-standard	trig	double	SPU
sincosf4	non-standard	trig	single	Both
sind2	sin	trig	double	SPU
sinf4	sinf	trig	single	Both
sinhd2	sinh	hyperbolic	double	SPU
sinhf4	sinhf	hyperbolic	single	Both
sqrtd2	sqrt	sqrt	double	SPU
sqrtf4	sqrtf	sqrt	single	Both
tand2	tan	trig	double	SPU
tanf4	tanf	trig	single	Both
tanhd2	tanh	hyperbolic	double	SPU
tanhf4	tanhf	hyperbolic	single	Both
tgammad2	tgamma	gamma	double	SPU
tgammaf4	tgammaf	gamma	single	Both



Function Name	C99 Name	Function Category	Precision	SPU/PPU
truncd2	trunc	rounding	double	SPU
truncf4	truncf	rounding	single	Both

## 1.3. Special Cases

Unless otherwise specified, each element of a SIMD result will adhere to either the C99 standard or the *IEC 60559:1989* standard.

### 1.3.1. Rounding

On the SPU, the full range of IEEE rounding modes is supported for double precision, but only round-toward-zero is supported for single precision. On the PPU, SIMD operations always use IEEE round-to-nearest mode.

The mathematical accuracy of the SIMD functions assumes the default rounding mode. Accuracy may be compromised if the functions are called in another rounding mode.

### 1.3.2. Special Operands

On the PPU, NaN (not-a-number) and Inf (infinity) are recognized as special operands.

On the SPU, all values passed to single-precision functions are treated as ordinary operands. NaN and Inf are not recognized as special single-precision operands; however, they are recognized as special double-precision operands, and SIMD functions check for them, as described in C99, *IEC 60559:1989* and the SIMD function specifications. See *Synergistic Processor Unit Instruction Set Architecture* for details.

On both the PPU and the SPU, single-precision floating-point denormal inputs are coerced to zero unless otherwise noted.

### 1.3.3. Error Conditions

A *domain error* occurs if an input argument is outside the domain over which the mathematical function is defined. The description of each function lists any required domain errors. The resulting vector element is undefined for all corresponding element input arguments that contain a domain error, and no exception or error is reported.

A *range error* occurs when the mathematical result cannot be represented in an object of the specified type. When a range error occurs, the resulting vector element is either `HUGE_VAL` (for double-precision results) or `HUGE_VALF` (for single-precision results). Integer arithmetic function results are undefined when they cannot be represented.

### 1.3.4. Exceptions

The SIMD library functions have an undefined effect on the exception flags in the SPU floating-point status and control register (`FPSCR`). SPU functions with double-precision arguments set exception bits in the `FPSCR` that can be tested by calling the routine `fegetexcept()`, as documented in `fenv.h`.

The SPU does not raise hardware traps for single-precision floating-point exceptions; PPU SIMD operations do have hardware support for a subset of the C99 floating-point exceptions.





## 2. SIMD Function Specifications

This chapter contains descriptions of the SIMD math functions, their arguments, and their return values. Where necessary, information about accuracy is provided to clarify expected specific behavior. Unless otherwise noted, all functions are available on both the PPU and SPU.

### 2.1. Type Definitions

The following type definitions are used for function return values.

#### **divi4\_t: Remainder/Quotient Struct for Vector Signed Int**

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

Structures of this type are used to hold the return value of `divi4()`. The member `quot` contains the quotient, and the member `rem` contains the remainder of the division.

#### **divu4\_t: Remainder/Quotient Struct for Vector Unsigned Int**

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

Structures of this type are used to hold the return value of `divu4()`. The member `quot` contains the quotient, and the member `rem` contains the remainder of the division.

#### **lldivi2\_t: Remainder/Quotient Struct for Vector Signed Long Long (SPU Only)**

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

Structures of this type are used to hold the return value of `lldivi2()`. The member `quot` contains the quotient, and the member `rem` contains the remainder of the division.

#### **lldivu2\_t: Remainder/Quotient Struct for Vector Unsigned Long Long (SPU Only)**

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

Structures of this type are used to hold the return value of `lldivu2()`. The member `quot` contains the quotient, and the member `rem` contains the remainder of the division.

#### **llroundf4\_t: Vector of Four Long Long (SPU Only)**

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

Structures of this type are used to hold signed long long data corresponding to a vector of four elements.

## 2.2. Function Descriptions

In the function descriptions that follow, a subscript is used to indicate a vector element. For example, element  $i$  of vector  $x$  is shown as  $x_i$ .

### absi4: Absolute Value of Integer

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

A vector signed int is returned that contains the absolute value of each corresponding element of vector signed int  $x$ .

If the absolute value of  $x_i$  cannot be represented, the corresponding result is undefined and no error is reported.

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

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

A vector double is returned that contains the angles whose cosines correspond to the respective elements in vector double  $x$ . Each element in the result is in the range  $[0, \pi]$  radians.

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

### acosf4: Arccosine of Float

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

A vector float is returned that contains the angles whose cosines correspond to the respective elements in vector float  $x$ . Each element in the result is in the range  $[0, \pi]$  radians.

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

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

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

A vector double is returned that contains the non-negative hyperbolic arccosines of the corresponding elements of vector double  $x$ .

If the value of  $x_i$  is less than 1, the corresponding result is undefined and no error is reported.

### acoshf4: Hyperbolic Arccosine of Float

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

A vector float is returned that contains the non-negative hyperbolic arccosines of the corresponding elements of vector float  $x$ .

If the value of  $x_i$  is less than 1, the corresponding result is undefined and no error is reported.

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

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

A vector double is returned that contains the angles whose sines correspond to the respective elements in vector double  $x$ . Each element in the result is in the range  $[-\pi/2, +\pi/2]$  radians.

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

### asinf4: Arcsine of Float

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

A vector float is returned that contains the angles whose sines correspond to the respective elements in vector float  $x$ . Each element in the result is in the range  $[-\pi/2, +\pi/2]$  radians.

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

#### **asinh2: Hyperbolic Arcsine of Double (SPU Only)**

```
(vector double) asinh2 (vector double x)
```

A vector double is returned that contains the hyperbolic arcsines of the corresponding elements of vector double  $x$ .

#### **asinh4: Hyperbolic Arcsine of Float**

```
(vector float) asinh4 (vector float x)
```

A vector float is returned that contains the hyperbolic arcsines of the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the angles whose tangents correspond to the respective elements of vector double  $x$ . Each element in the result is in the range  $[-\pi/2, +\pi/2]$  radians.

#### **atanf4: Tangent of Float**

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

A vector float is returned that contains the angles whose tangents correspond to the respective elements of vector float  $x$ . Each element in the result is in the range  $[-\pi/2, +\pi/2]$  radians.

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

```
(vector double) atanh2 (vector double x)
```

A vector double is returned that contains the hyperbolic arctangents of the corresponding elements of vector double  $x$ .

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

#### **atanhf4: Hyperbolic Arctangent of Float**

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

A vector float is returned that contains the hyperbolic arctangents of the corresponding elements of vector float  $x$ .

If the absolute value of  $x_i$  is greater than 1, the corresponding result is undefined and no error is reported.

On the SPU, if  $x_i$  is equal to 1, the corresponding element of the result will be returned as `HUGE_VALF`, and if  $x_i$  is equal to -1, the corresponding element of the result will be returned as `-HUGE_VALF`. In either case, no error is reported.

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

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

A vector double is returned that contains the angles whose tangents are  $y_i/x_i$  for the corresponding elements of vector double  $y$  and vector double  $x$ . Each element in the result is within the range  $[-\pi, +\pi]$  radians.

If  $x_i$  and  $y_i$  are both 0, the corresponding element of the result is undefined and no error is reported.

#### **atan2f4: Arctangent of Float Quotient**

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

A vector float is returned that contains the angles whose tangents are  $y_i/x_i$  for the corresponding elements of vector float  $y$  and vector float  $x$ . Each element in the result is within the range  $[-\pi, +\pi]$  radians.

If  $x_i$  and  $y_i$  are both 0, the corresponding element of the result is undefined, and no error is reported.

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

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

A vector double is returned that contains the real cube roots,  $x_i^{1/3}$ , of the corresponding elements of vector double  $x$ .

#### **cbrtf4: Cube Root of Float**

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

A vector float is returned that contains the real cube roots,  $x_i^{1/3}$ , of the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the smallest integer values, expressed as floating-point numbers, that are not less than the corresponding elements of vector double  $x$ .

#### **ceilf4: Ceiling of Float**

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

A vector float is returned that contains the smallest integer values, expressed as floating-point numbers, that are not less than the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the magnitude of the corresponding element of vector double  $x$  and the sign of the corresponding element of vector double  $y$ .

#### **copysignf4: Copy Sign of Float**

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

A vector float is returned that contains the magnitude of the corresponding element of vector float  $x$  and the sign of the corresponding element of vector float  $y$ .

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

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

A vector double is returned that contains the cosines of the corresponding elements of vector double  $x$ .

The results of `cosd2()` may not be accurate for very large values of  $x$ , but no error is reported. Implementations should document the point at which accuracy is lost.

**cosf4: Cosine of Float**

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

A vector float is returned that contains the cosines of the corresponding elements of vector float  $x$ .

The results of `cosf4()` may not be accurate for very large values of  $x$ , but no error is reported. Implementations should document the point at which accuracy is lost.

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

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

A vector double is returned that contains the hyperbolic cosines of the corresponding elements of vector double  $x$ .

**coshf4: Hyperbolic Cosine of Float**

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

A vector float is returned that contains the hyperbolic cosines of the corresponding elements of vector float  $x$ .

On the SPU, element values of the result that are greater than `HUGE_VALF` are returned as `HUGE_VALF`, and no error is reported.

**divd2: Divide Doubles (SPU Only)**

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

A vector double is returned that contains the quotients  $x_i/y_i$  for the corresponding elements of vector double  $x$  and vector double  $y$ . This function handles special cases as follows:

- If either input is NaN, the result is NaN.
- For `Inf/Inf` or `0/0`, the result is NaN.
- For `finite/0`, the result is `Inf` with `sign = sign(x)/sign(y)`.
- For `finite/±Inf`, the result is 0 with `sign = sign(x)/sign(y)`.

**divf4: Divide Floats**

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

A vector float is returned that contains the quotients  $x_i/y_i$  for the corresponding elements of vector float  $x$  and vector float  $y$ . This function handles special cases as follows:

- If either input is NaN, the result is NaN.
- For `Inf/Inf` or `0/0`, the result is NaN.
- For `finite/0`, the result is `Inf` with `sign = sign(x)/sign(y)`.
- For `finite/±Inf`, the result is 0 with `sign = sign(x)/sign(y)`.

On the SPU, if  $y_i$  is 0, the result is `HUGE_VALF` with `sign = sign(x)/sign(y)`.

**divi4: Divide Integer**

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

Each element of vector signed int  $x$  is divided by the corresponding element of vector signed int  $y$ , and the result is returned in a structure of type `divi4_t` that contains a vector of corresponding quotients and a vector of corresponding remainders.

Each element in the structure member `quot` is the algebraic quotient truncated towards zero. Each element in the structure member `rem` is the corresponding remainder, such that  $x_i == \text{quot} * y_i + \text{rem}$ .

If  $y_i$  is 0, the corresponding element of the resulting quotient is 0.

**divu4: Divide Unsigned Integer**

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

Each element of vector unsigned int  $x$  is divided by the corresponding element of vector unsigned int  $y$ , and the result is returned in a structure of type `divu4_t` that contains a vector of corresponding quotients and a vector of corresponding remainders.

Each element in the structure member `quot` is the algebraic quotient truncated towards zero. Each element in the structure member `rem` is the corresponding remainder, such that  $x_i == \text{quot} * y_i + \text{rem}$ .

If  $y_i$  is 0, the corresponding element of the resulting quotient is 0.

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

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

A vector double is returned that contains the complementary error functions of the corresponding elements of vector double  $x$ .

**erfcf4: Complementary Error Function Float**

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

A vector float is returned that contains the complementary error functions of the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the error functions of the corresponding elements of vector double  $x$ .

**erff4: Error Function Float**

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

A vector float is returned that contains the error functions of the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the corresponding exponentials  $e^{x_i}$  for each element of vector double  $x$ .

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

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

A vector float is returned that contains the corresponding exponentials  $e^{x_i}$  for each element of vector float  $x$ .

On the SPU, element values of the result that are greater than `HUGE_VALF` are returned as `HUGE_VALF`, and no error is reported.

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

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

A vector double is returned that contains the corresponding exponentials  $2^{x_i}$  for each element of vector double  $x$ .

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

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

A vector float is returned that contains the corresponding exponentials  $2^{x_i}$  for each element of vector float  $x$ .

On the SPU, element values of the result that are greater than `HUGE_VALF` are returned as `HUGE_VALF`, and no error is reported.

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

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

A vector double is returned that contains the exponential minus 1,  $e^{x_i} - 1$ , for the corresponding elements of vector double  $x$ .

This function returns mathematically accurate values even when  $x_i$  is near zero or when  $\exp(x_i) - 1.0$  would return bad values due to floating-point cancellation errors.

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

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

A vector float is returned that contains the exponential minus 1,  $e^{x_i} - 1$ , for the corresponding elements of vector float  $x$ .

This function returns mathematically accurate values even when  $x_i$  is near zero or when  $\expf(x_i) - 1.0f$  would return bad values due to floating-point cancellation errors.

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

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

A vector double is returned that contains the absolute values,  $|x_i|$ , for the corresponding elements of vector double  $x$ .

#### **fabsf4: Absolute Value Float**

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

A vector float is returned that contains the absolute values,  $|x_i|$ , for the corresponding elements of vector float  $x$ .

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

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

A vector double is returned that contains the larger of  $(x_i - y_i)$  and zero, for the corresponding elements of vector double  $x$  and vector double  $y$ .

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

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

A vector float is returned that contains the larger of  $(x_i - y_i)$  and zero, for the corresponding elements of vector float  $x$  and vector float  $y$ .

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

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

A vector double is returned that contains the largest integer values, expressed as floating-point numbers, that are not greater than the corresponding elements of vector double  $x$ .

#### **floorf4: Floor Float**

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

A vector float is returned that contains the largest integer values, expressed as floating-point numbers, that are not greater than the corresponding elements of vector float  $x$ .



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

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

A vector double is returned that contains the results of the calculation of  $(x_i * y_i + z_i)$  for the corresponding elements of vector double  $x$ , vector double  $y$ , and vector double  $z$ . Intermediate results are of arbitrary precision.

**fmaf4: Fused Multiply and Add Float**

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

A vector float is returned that contains the results of the calculation of  $(x_i * y_i + z_i)$  for the corresponding elements of vector float  $x$ , vector float  $y$ , and vector float  $z$ . Intermediate results are of arbitrary precision.

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

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

A vector double is returned that contains the larger (more positive) of  $x_i$  and  $y_i$ , for the corresponding elements of vector double  $x$  and vector double  $y$ .

**fmaxf4: Maximum Float**

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

A vector float is returned that contains the larger (more positive) of  $x_i$  and  $y_i$ , for the corresponding elements of vector float  $x$  and vector float  $y$ .

On the SPU, this function does not coerce denormals to zero. Instead, it compares them as normal values even though the SPU's floating-point instructions do not.

**fmind2: Minimum Double (SPU Only)**

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

A vector double is returned that contains the smaller (more negative) of  $x_i$  and  $y_i$ , for the corresponding elements of vector double  $x$  and vector double  $y$ .

**fminf4: Minimum Float**

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

A vector float is returned that contains the smaller (more negative) of  $x_i$  and  $y_i$ , for the corresponding elements of vector float  $x$  and vector float  $y$ .

On the SPU, this function does not coerce denormals to zero. Instead, it compares them as normal values even though the SPU's floating-point instructions do not.

**fmodd2: Modulus Double (SPU Only)**

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

A vector double is returned where each element contains the remainder of  $x_i / y_i$ , for the corresponding elements of vector double  $x$  and vector double  $y$ , as defined below:

- If  $y_i$  is 0, the result is 0.
- Otherwise, the function determines the unique signed integer value  $z$  such that the returned element is  $x_i - z * y_i$  with the same sign as  $x_i$  and a magnitude less than  $|y_i|$ .

**fmodf4: Modulus Float**

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

A vector float is returned where each element contains the remainder of  $x_i/y_i$ , for the corresponding elements of vector float  $x$  and vector float  $y$ , as defined below:

- If  $y_i$  is 0, the result is 0.
- Otherwise, the function determines the unique signed integer value  $z$  such that the returned element is  $x_i - z * y_i$  with the same sign as  $x_i$  and a magnitude less than  $|y_i|$ .

**fpclassifyd2: Classify Double (SPU Only)**

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

A vector signed long long is returned that contains the floating-point classifications for the corresponding elements of vector double  $x$ . The classifications, which are defined in `math.h`, are `FP_NAN`, `FP_INFINITE`, `FP_NORMAL`, `FP_SUBNORMAL`, and `FP_ZERO`.

**fpclassifyf4: Classify Float**

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

A vector signed int is returned that contains the floating-point classifications for the corresponding elements of vector float  $x$ . The classifications, which are defined in `math.h`, are `FP_NAN`, `FP_INFINITE`, `FP_NORMAL`, `FP_SUBNORMAL`, and `FP_ZERO`.

On the SPU, the resulting vector will never contain `FP_NAN` or `FP_INFINITE`.

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

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

A vector double is returned that contains normalized fractions, and a vector signed long long is stored in `*pexp`, which contains exponent integers. Each fraction element `frac` and each exponent integer element `exp` represent the value of the corresponding element of  $x$ , such that:

- $|frac|$  is in the interval  $[1/2, 1)$  or is 0.
- $x_i == frac * 2^{exp}$
- If  $x_i$  is 0, the corresponding element of `*pexp` is also 0.
- If  $x_i$  is NaN, the corresponding result is NaN and the corresponding element of `*pexp` is undefined.
- If  $x_i$  is infinite, the corresponding result is infinite and the corresponding element of `*pexp` is undefined.

**frexpf4: Represent Float as Fraction and Exponent**

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

A vector float is returned that contains normalized fractions, and a vector signed int is stored in `*pexp`, which contains exponent integers. Each fraction element `frac` and each exponent integer element `exp` represent the value of the corresponding element of  $x$ , such that:

- $|frac|$  is in the interval  $[1/2, 1)$  or is 0.
- $x_i == frac * 2^{exp}$
- If  $x_i$  is 0, the corresponding element of `*pexp` is also 0.
- If  $x_i$  is NaN, the corresponding result is NaN and the corresponding element of `*pexp` is undefined.
- If  $x_i$  is infinite, the corresponding result is infinite and the corresponding element of `*pexp` is undefined.

**hypotd2: Hypotenuse Double (SPU Only)**

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

A vector double is returned that contains the square root of  $x_i^2 + y_i^2$  without undue overflow or underflow, for the corresponding elements of vector double  $x$  and vector double  $y$ .

**hypotf4: Hypotenuse Float**

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

A vector float is returned that contains the square root of  $x_i^2 + y_i^2$  without undue overflow or underflow, for the corresponding elements of vector float  $x$  and vector float  $y$ .

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

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

A vector signed long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- If  $x_i$  is NaN, the value is the macro `FP_ILOGBNAN`.
- If  $x_i$  is equal to 0, the value is the macro `FP_ILOGB0`.
- If  $x_i$  is equal to positive or negative `Inf`, the value is the macro `INT_MAX`.
- Otherwise, the result is `(int)logb( $x_i$ )`.

**ilogbf4: Integer Exponent of Float**

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

A vector signed int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- If  $x_i$  is NaN, the value is the macro `FP_ILOGBNAN`.
- If  $x_i$  is equal to 0, the value is the macro `FP_ILOGB0`.
- If  $x_i$  is equal to positive or negative `Inf`, the value is the macro `INT_MAX`.
- Otherwise, the result is `(int)logbf( $x_i$ )`.

Because the SPU treats single-precision `Inf` and `NaN` codes as regular floating-point numbers, `ilogbf4()` returns a result of 128 for these numbers. For compatibility with the double function `ilogb()`, `FP_ILOGBNAN` is set to `INT_MAX`.

**irintf4: Nearest Integer Float**

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

A vector signed int is returned that contains the nearest integer to the corresponding element of vector float  $x$ , consistent with the current rounding mode. If the rounded value is outside the range of the return type, the numeric result is unspecified.

On the SPU, the rounding mode for floats is always towards zero.

**iroundf4: Round Float to Nearest Integer**

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

A vector signed int is returned that contains the rounded integer value of the corresponding element of vector float  $x$ .

Elements are rounded to the nearest value; halfway cases are rounded away from zero, regardless of the current rounding direction.

If the rounded value is outside the range of the return type, the numeric result is unspecified.

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is a denormalized value or zero.
- Otherwise, the bits are set to zero.

**is0denormf4: 0 or Denormalized Float**

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

A vector unsigned int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is a denormalized value or zero.
- Otherwise, the bits are set to zero.

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  and  $y_i$  are equal.
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers. If either input is NaN, the comparison result is false (zero). If both inputs are Inf with the same sign, the inputs are considered equal. The values 0 and -0 are considered equal.

**isqualf4: Compare Equal Float**

```
(vector unsigned int) isqualf4 (vector float x, vector float y)
```

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  and  $y_i$  are equal.
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers. If either input is NaN, the comparison is false (zero). If both inputs are Inf with the same sign, the inputs are considered equal. The values 0 and -0 are considered equal.

**isfinitd2: Double Is Finite (SPU Only)**

```
(vector unsigned long long) isfinitd2 (vector double x)
```

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is finite.
- Otherwise, the bits are set to zero.

**isfinitef4: Float Is Finite**

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

A vector unsigned int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is finite.
- Otherwise, the bits are set to zero.

On the SPU, infinite values are not representable in single precision. Therefore, all bits of the resulting element are set to 1 regardless of the value of  $x_i$ .

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is equal to or greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers. If either element of the input is NaN, the comparison is false. If both elements of the inputs are Inf with the same sign, the inputs are considered equal. The values 0 and -0 are considered equal.

#### **isgreaterequalf4: Greater or Equal Float**

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is equal to or greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers. If either element of the input is NaN, the comparison is false. If both elements of the inputs are Inf with the same sign, the inputs are considered equal. The values 0 and -0 are considered equal.

#### **isgreaterd2: Greater Than Double (SPU Only)**

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

#### **isgreaterf4: Greater Than Float**

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

**isinf2: Double Is Infinity (SPU Only)**

```
(vector unsigned long long) isinf2 (vector double x)
```

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is infinite.
- Otherwise, the bits are set to zero.

**isinf4: Float Is Infinity**

```
(vector unsigned int) isinf4 (vector float x)
```

A vector unsigned long int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is infinite.
- Otherwise, the bits are set to zero.

On the SPU, infinite values are not representable in single precision. Therefore, all bits of the resulting element are set to zero, regardless of the value of  $x_i$ .

**islessd2: Double Is Less Than (SPU Only)**

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

**islessequald2: Double Is Less Than or Equal To (SPU Only)**

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than or equal to  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

**islessequalf4: Float Is Less Than or Equal To**

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than or equal to  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

**islessf4: Float Is Less Than**

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than or greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

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

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is less than or greater than  $y_i$ .
- Otherwise, the bits are set to zero.

The function correctly compares denormalized numbers.

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is NaN.
- Otherwise, the bits are set to zero.

**isnanf4: Float Is NaN**

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

A vector unsigned int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is NaN.
- Otherwise, the bits are set to zero.

On the SPU, NaN is not representable in single precision. Therefore, all bits of the resulting element are set to zero, regardless of the value of  $x_i$ .

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is normal (not a NaN or an infinity).
- Otherwise, the bits are set to zero.

**isnormalf4: Float Is Normal**

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

A vector unsigned int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if  $x_i$  is normal (not a NaN or an infinity).
- Otherwise, the bits are set to zero.

**isunorderedd2: Double Is Unordered (SPU Only)**

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

A vector unsigned long long is returned where each element is defined below for the corresponding elements of vector double  $x$  and vector double  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is unordered with respect to  $y_i$ .
- Otherwise, the bits are set to zero.

NaN is unordered to any operand, including NaN itself.

**isunorderedf4: Float Is Unordered**

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

A vector unsigned int is returned where each element is defined below for the corresponding elements of vector float  $x$  and vector float  $y$ :

- All bits of the resulting element are set to 1 if  $x_i$  is unordered to  $y_i$ .
- Otherwise, the bits are set to zero.

NaN is unordered to any operand, including NaN itself. On the SPU, NaN does not exist in single precision. Therefore, this function will always return 0.

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

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

A vector double is returned that contains  $x_i * 2^{ex_i}$  for the corresponding elements of vector double  $x$  and vector signed long long  $ex$ . For large elements of  $ex$  (overflow), the element in the result saturates to HUGE\_VAL with an appropriate sign. For small elements of  $ex$  (underflow), the corresponding element of the result is 0.

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

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

A vector float is returned that contains  $x_i * 2^{ex_i}$  for the corresponding elements of vector float  $x$  and vector signed int  $ex$ . For large elements of  $ex$  (overflow), the element in the result saturates to HUGE\_VALF with an appropriate sign. For small elements of  $ex$  (underflow), the corresponding element of the result is 0.



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

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

A vector double is returned that contains the natural logarithm of the absolute value of the result of the gamma function for the corresponding elements of vector double  $x$ .

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

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

A vector float is returned that contains the natural logarithm of the absolute value of the result of the gamma function for the corresponding element of vector float  $x$ .

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

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

A vector long long is returned that contains the absolute value,  $|x_i|$ , of the corresponding element of vector signed long long  $x$ .

If the absolute value of  $x_i$  cannot be represented, the corresponding result is undefined and no error is reported.

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

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

Each element of vector signed long long  $x$  is divided by each element of vector signed long long  $y$ , and the result is returned in a structure of type `lldivi2_t`, which contains a vector of quotients and a vector of remainders.

Each element of the vector in the structure member *quot* is the algebraic quotient truncated towards zero. Each element of the vector in the structure member *rem* is the corresponding remainder, such that  $x_i == \text{quot} * y_i + \text{rem}$ .

If  $y_i$  is 0, the corresponding element of the resulting quotient is 0.

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

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

Each element of vector unsigned long long  $x$  is divided by each element of vector unsigned long long  $y$ , and the result is returned in a structure of type `lldivu2_t`, which contains a vector of quotients and a vector of remainders.

Each element of the vector in the structure member *quot* is the algebraic quotient truncated towards zero. Each element of the vector in the structure member *rem* is the corresponding remainder, such that  $x_i == \text{quot} * y_i + \text{rem}$ .

If  $y_i$  is 0, the corresponding element of the resulting quotient is 0.

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

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

A vector signed long long is returned that contains the nearest long long integer to the corresponding element of vector double  $x$  consistent with the current rounding mode. If the rounded value is outside the range of the return type, the numeric result is unspecified.

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

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

A structure of type `llroundf4_t` is returned that contains the nearest long long integer to the corresponding element of vector float  $x$  consistent with the current rounding mode. If the rounded value is outside the range of the return type, the numeric result is unspecified.

On the SPU, the rounding mode for floats is always towards zero.

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

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

A vector signed long long is returned that contains the corresponding elements of vector double  $x$  rounded to the nearest value, rounding halfway values away from zero regardless of the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified.

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

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

A structure of type `llroundf4_t` is returned that contains the corresponding elements of vector float  $x$  rounded to the nearest value, rounding halfway cases away from zero regardless of the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified.

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

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

A vector double is returned that contains the natural logarithms of the corresponding elements of vector double  $x$ .

If  $x_i$  is negative, the corresponding result is undefined and no error is reported.

### **logf4: Natural Log of Float**

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

A vector float is returned that contains the natural logarithms of the corresponding elements of vector float  $x$ .

If  $x_i$  is negative, the corresponding result is undefined and no error is reported.

If  $x_i$  is 0, the result is `-HUGE_VALF`.

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

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

A vector double is returned that contains the base-10 logarithm of the corresponding elements of vector double  $x$ .

If  $x_i$  is negative, the corresponding result is undefined and no error is reported.

### **log10f4: Log Base 10 of Float**

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

A vector float is returned that contains the base-10 logarithm of the corresponding elements of vector float  $x$ .

If  $x_i$  is negative, the corresponding result is undefined and no error is reported.

If  $x_i$  is 0, the result is `-HUGE_VALF`.

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

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

A vector double is returned that contains the natural logarithm of  $1 + x_i$ .

The function returns mathematically accurate values even when  $x_i$  is near zero.

If  $x_i$  is less than -1, the corresponding result is undefined and no error is reported.

### log1pf4: Natural Log of Float Plus 1

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

A vector float is returned that contains the natural logarithms of  $1 + x_i$ .

The function returns mathematically accurate values even when  $x_i$  is near zero. If an element of  $x$  is  $-1$ , the result is

`-HUGE_VALF`.

If  $x_i$  is less than  $-1$ , the corresponding result is undefined and no error is reported.

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

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

A vector double is returned that contains the base-2 logarithm of the corresponding elements of vector double  $x$ .

If  $x_i$  is less than 0, the corresponding result is undefined and no error is reported.

### log2f4: Log Base 2 of Float

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

A vector float is returned that contains the base-2 logarithm of the corresponding elements of vector float  $x$ .

If  $x_i$  is 0, the result is `-HUGE_VALF`.

If  $x_i$  is less than 0, the corresponding result is undefined and no error is reported.

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

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

An integer exponent  $ex_i$  and a fraction  $frac_i$  are determined for the corresponding elements of vector double  $x$ .

A vector double is returned where each element contains the value of  $ex_i$  for  $x_i$ , such that:

- $x_i == frac * FLT\_RADIX^{ex}$
- $|frac|$  is in the interval  $[1, FLT\_RADIX)$ .

If  $x_i$  is 0, the corresponding result is undefined and no error is reported.

On the SPU, if  $x_i$  is 0, the corresponding result is `-HUGE_VALF`. If  $x_i$  is infinite, the corresponding result is positive infinite. If  $x_i$  is NaN, the corresponding result is also NaN.

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

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

An integer exponent  $ex$  and a fraction  $frac_i$  are determined for the corresponding elements of vector float  $x$ . A

vector float is returned where each element contains the value of  $ex_i$  for  $x_i$ , such that:

- $x_i == frac * FLT\_RADIX^{ex}$
- $|frac|$  is in the interval  $[1, FLT\_RADIX)$ .

If  $x_i$  is 0, the corresponding result is undefined and no error is reported.

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

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

Each element of vector double  $x$  is split into an integral part  $v_i$  and a fractional part  $frac_i$ . A vector double is returned where each element contains the corresponding  $frac_i$  element, and another vector double is stored in `*pint`, which contains the corresponding  $v_i$  elements, such that:

- $x_i == frac_i + v_i$

- $|frac_i|$  is in the interval  $[0, 1)$ .
- Both  $frac_i$  and  $v_i$  have the same sign as  $x_i$ .

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

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

Each element of vector float  $x$  is split into an integral part  $v_i$  and a fractional part  $frac_i$ . A vector float is returned where each element contains the corresponding  $frac_i$  element, and another vector float is stored in  $*pint$ , which contains the corresponding  $v_i$  elements, such that:

- $x_i == frac_i + v_i$
- $|frac_i|$  is in the interval  $[0, 1)$ .
- Both  $frac_i$  and  $v_i$  have the same sign as  $x_i$ .

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

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

A vector double is returned that contains the corresponding elements of vector double  $x$  rounded to the nearest integer consistent with the current rounding mode, but without raising an inexact floating-point exception.

#### nearbyintf4: Find Nearest Integer for Float

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

A vector float is returned that contains the corresponding elements of vector float  $x$  rounded to the nearest integer, consistent with the current rounding mode, but without raising an inexact floating-point exception.

On the SPU, the rounding mode for a float is always towards zero.

#### negated2: Negate Double (SPU Only)

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

A vector double is returned that contains  $-x_i$  for the corresponding elements of vector double  $x$ .

#### negatef4: Negate Float

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

A vector float is returned that contains  $-x_i$  for the corresponding elements of vector float  $x$ .

#### negatei4: Negate Signed Integer

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

A vector signed int is returned that contains  $-x_i$  for the corresponding elements of vector signed int  $x$ .

If  $-x_i$  cannot be represented, the corresponding result is undefined and no error is reported.

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

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

A vector signed long long is returned that contains  $-x_i$  for the corresponding elements of vector signed long long  $x$ .

If  $-x_i$  cannot be represented, the corresponding result is undefined and no error is reported.

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

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

A vector double is returned that contains the next representable value after  $x_i$  in the direction of  $y_i$  for the corresponding elements of vector double  $x$  and vector double  $y$ . If  $x_i$  is equal to  $y_i$ , the result is  $y_i$ .

If the magnitude of  $x_i$  is the largest finite value representable, the result is undefined.

### nextafterf4: Find Next Integer After for Float

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

A vector float is returned that contains the next representable value after  $x_i$  in the direction of  $y_i$  for the corresponding elements of vector float  $x$  and vector float  $y$ . If the element of  $x_i$  is equal to  $y_i$ , the result is  $y_i$ .

If the magnitude of  $x_i$  is the largest finite value representable, the result is undefined.

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

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

A vector double is returned that contains  $x_i$  raised to the power of  $y_i$ ,  $x_i^{y_i}$ , for the corresponding elements of vector double  $x$  and vector double  $y$ .

If  $x_i$  is finite and negative and  $y_i$  is finite and not a integer value, the corresponding result is undefined and no error is reported.

### powf4: Raise Float to Float Power

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

A vector float is returned that contains  $x_i$  raised to the power of  $y_i$ ,  $x_i^{y_i}$ , for the corresponding elements of vector float  $x$  and vector float  $y$ .

On the SPU, if the result would be greater than `HUGE_VALF`, the result is saturated to `HUGE_VALF` and no error is reported.

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

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

A vector double is returned that contains the reciprocal of the corresponding elements of vector double  $x$ .

The function handles special cases as follows:

- When  $x_i$  is  $\pm\text{Inf}$ , the result is 0 with the sign of  $x_i$ .
- When  $x_i$  is 0, the result is  $\text{Inf}$  with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

### recipf4: Reciprocal of Float

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

A vector float is returned that contains the reciprocal of the corresponding elements of vector float  $x$ .

The function handles special cases as follows:

- When  $x_i$  is  $\pm\text{Inf}$ , the result is 0 with the sign of  $x_i$ .
- When  $x_i$  is 0, the result is `HUGE_VALF` with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

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

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

A vector double is returned that contains the remainder  $x_i \text{ REM } y_i$  for the corresponding elements of vector double  $x$  and vector double  $y$ .

If  $y_i$  is 0, the corresponding element of the result is undefined and no error is reported.

**remainderf4: Remainder of Floats**

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

A vector float is returned that contains the remainder  $x_i \text{ REM } y_i$  for the corresponding elements of vector float  $x$  and vector float  $y$ .

If  $y_i$  is 0, the corresponding element of the result is undefined and no error is reported.

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

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

This function returns the same vector double result as `remainderd2()`. In addition a vector signed long long is stored in `*pquo`, which contains the corresponding element values whose sign is the sign of  $x_i / y_i$  and whose magnitude is congruent modulo  $2^n$  to the magnitude of the integral quotient of  $x_i / y_i$ , where  $n$  is an implementation-defined integer greater than or equal to 3.

**remquof4: Remainder Function of Float**

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

This function returns the same vector float result as `remainderf4()`. In addition a vector signed int is stored in `*pquo`, which contains the corresponding element values whose sign is the sign of  $x_i / y_i$  and whose magnitude is congruent modulo  $2^n$  to the magnitude of the integral quotient of  $x_i / y_i$ , where  $n$  is an implementation-defined integer greater than or equal to 3.

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

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

A vector double is returned that contains the corresponding elements of vector double  $x$  rounded to the nearest integer, consistent with the current rounding mode.

**rintf4: Round Float to the Nearest Integer**

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

A vector float is returned that contains the corresponding elements of vector float  $x$  rounded to the nearest integer, consistent with the current rounding model.

On the SPU, the rounding mode for float is always towards zero.

**roundd2: Round Double (SPU Only)**

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

A vector double is returned that contains the rounded elements of vector double  $x$ . Rounding is done to the nearest integer value in floating-point format. Halfway cases are rounded away from zero regardless of the current rounding direction.

**roundf4: Round Float**

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

A vector float is returned that contains the rounded elements of vector float  $x$ . Rounding is done to the nearest integer value in floating-point format. Halfway cases are rounded away from zero regardless of the current rounding direction.

**rsqrt2: Reciprocal Square Root of Double (SPU Only)**

```
(vector double) rsqrt2 (vector double x)
```

A vector double is returned that contains the reciprocal of the square root of  $x_i$  for the corresponding elements of vector double  $x$ . Special cases are handled as follows:

- When  $x_i$  is less than 0, the result is NaN.
- When  $x_i$  is +Inf, the result is +0.
- When  $x_i$  is 0, the result is Inf with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

**rsqrtf4: Reciprocal Square Root of Float**

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

A vector float is returned that contains the reciprocal of the square root of  $x_i$  for the corresponding elements of vector float  $x$ . Special cases are handled as follows:

- When  $x_i$  is less than 0, the result is NaN.
- When  $x_i$  is +Inf, the result is +0.
- When  $x_i$  is 0, the result is Inf with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

On the SPU, if  $x_i$  is less than 0, the corresponding result is undefined.

**scalblnd2: Scale Double by Long Long Integer (SPU Only)**

```
(vector double) scalblnd2 (vector double x, vector signed long long n)
```

A vector double is returned that contains  $x_i$  efficiently multiplied by  $2^n$  for the corresponding elements of vector double  $x$  and vector signed long long  $n$ .

**scalbnf4: Scale Float by Integer**

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

A vector float is returned that contains  $x_i$  efficiently multiplied by  $2^n$  for the corresponding elements of vector float  $x$  and vector signed int  $n$ .

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

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

A vector unsigned long long is returned where each element is defined below for the corresponding element of vector double  $x$ :

- All bits of the resulting element are set to 1 if the sign bit is set in  $x_i$ .
- Otherwise, the bits are set to zero.

**signbitf4: Sign Bit of Float**

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

A vector unsigned int is returned where each element is defined below for the corresponding element of vector float  $x$ :

- All bits of the resulting element are set to 1 if the sign bit is set in  $x_i$ .
- Otherwise, the bits are set to zero.

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

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

A vector double is stored in  $*sx$  and a vector double is stored in  $*cx$  that contain the respective sines and cosines of the corresponding elements of vector double  $x$ .

The results of `sincosd2()` may not be accurate for very large values of  $x_i$ , and no error is reported. Implementations should document the point at which accuracy is lost.

**sincosf4: Sine and Cosine of Float**

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

A vector float is stored in  $*sx$  and a vector float is stored in  $*cx$  that contain the respective sines and cosines of the corresponding elements of vector float  $x$ .

The results of `sincosf4()` may not be accurate for very large values of  $x_i$ , and no error is reported. Implementations should document the point at which accuracy is lost.

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

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

A vector double is returned that contains the corresponding sines of the elements of vector double  $x$ .

The results of `sind2()` may not be accurate for very large values of  $x_i$ , and no error is reported. Implementations should document the point at which accuracy is lost.

**sinf4: Sine of Float**

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

A vector float is returned that contains the corresponding sines of the elements of vector float  $x$ .

The results of `sinf4()` may not be accurate for very large values of  $x_i$ , and no error is reported. Implementations should document the point at which accuracy is lost.

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

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

A vector double is returned that contains the corresponding hyperbolic sines of the elements of vector double  $x$ .

**sinhf4: Hyperbolic Sine of Float**

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

A vector float is returned that contains the corresponding hyperbolic sines of the elements of vector float  $x$ .

On the SPU, element values of the result that are greater than `HUGE_VALF` are returned as `HUGE_VALF`, and no error is reported.



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

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

A vector double is returned that contains the real square roots  $x_i^{1/2}$  for the corresponding elements of vector double  $x$ .

This function handles special cases as follows:

- When  $x_i$  is less than 0, the result is NaN.
- When  $x_i$  is +Inf, the result is +Inf.
- When  $x_i$  is 0, the result is 0 with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

### **sqrtf4: Square Root of Float**

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

A vector float is returned that contains the real square roots  $x_i^{1/2}$  for the corresponding elements of vector float  $x$ .

This function handles special cases as follows:

- When  $x_i$  is less than 0, the result is NaN.
- When  $x_i$  is +Inf, the result is +Inf.
- When  $x_i$  is 0, the result is 0 with the sign of  $x_i$ .
- When  $x_i$  is NaN, the result is NaN.

On the SPU, the result is undefined when  $x_i$  is negative.

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

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

A vector double is returned that contains the corresponding tangents of the elements of vector double  $x$ .

The results may not be accurate for very large values of  $x_i$  and no error is reported. Implementations should document the point at which accuracy is lost.

### **tanf4: Tangent of Float**

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

A vector float is returned that contains the corresponding tangents of the elements of vector float  $x$ .

The results may not be accurate for very large values of  $x_i$  and no error is reported. Implementations should document the point at which accuracy is lost.

### **tanh2: Hyperbolic Tangent of Double (SPU Only)**

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

A vector double is returned that contains the corresponding hyperbolic tangents of the elements of vector double  $x$ .

### **tanhf4: Hyperbolic Tangent of Float**

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

A vector float is returned that contains the corresponding hyperbolic tangents of the elements of vector float  $x$ .

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

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

A vector double is returned that contains the corresponding results of the gamma function applied to the respective elements of vector double  $x$ .

If  $x_i$  is a negative integer, the corresponding element of the result is undefined and no error is reported.

**tgammaf4: Gamma of Float**

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

A vector float is returned that contains the corresponding results of the gamma function applied to the respective elements of vector float  $x$ .

If  $x_i$  is a negative integer, the corresponding element of the result is undefined and no error is reported.

**truncd2: Truncate Double (SPU Only)**

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

A vector double is returned that contains  $x_i$  rounded to the nearest integer  $n$  that is not larger in magnitude than  $x_i$  (rounded towards zero) for each corresponding element of vector double  $x$ .

**truncf4: Truncate Float**

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

A vector float is returned that contains  $x_i$  rounded to the nearest integer  $n$  that is not larger in magnitude than  $x_i$  (rounded towards zero) for each corresponding element of vector float  $x$ .

**End of Document**