SIMD Math Library Specification for Cell Broadband Engine Architecture

Version 1.0

November 27, 2006
# SIMD Math Library Specification for Cell Broadband Engine Architecture, Version 1.0

## Table of Contents

**About This Document** vii  
**Audience** vii  
**Version History** vii  
**Related Documentation** vii  
**Document Structure** vii  
**Bit Notation** viii  
**Byte Ordering and Element Numbering** viii  
**Typographic Conventions** 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 5  
      1.3.1. Rounding 5  
      1.3.2. Special Operands 5  
      1.3.3. Error Conditions 5  
      1.3.4. Exceptions 5  

2. SIMD Function Specifications 6  
   2.1. Type Definitions 6  
      divi4_t: Remainder/Quotient Struct for Vector Signed Int 6  
      divu4_t: Remainder/Quotient Struct for Vector Unsigned Int 6  
      lldivi2_t: Remainder/Quotient Struct for Vector Signed Long Long (SPU Only) 6  
      lldivu2_t: Remainder/Quotient Struct for Vector Unsigned Long Long (SPU Only) 6  
      llroundf4_t: Vector of Four Long Long (SPU Only) 6  
   2.2. Function Descriptions 7  
      absi4: Absolute Value of Integer 7  
      acosd2: Arcosine of Double (SPU Only) 7  
      acosf4: Arcosine of Float 7  
      acoshd2: Hyperbolic Arcosine of Double (SPU Only) 7  
      acoshf4: Hyperbolic Arcosine of Float 7  
      asind2: Arcsine of Double (SPU Only) 7  
      asinf4: Arcsine of Float 7  
      asinh2: Hyperbolic Arcsine of Double (SPU Only) 7  
      asinhf4: Hyperbolic Arcsine of Float 7  
      atand2: Tangent of Double (SPU Only) 7  
      atand4: Tangent of Float 7  
      atanhd2: Hyperbolic Arctangent of Double (SPU Only) 7  
      atanhf4: Hyperbolic Arctangent of Float 7  
      atan2d2: Arctangent of Double Quotient (SPU Only) 7  
      atan2f4: Arctangent of Float Quotient 7  
      cbrtd2: Cube Root of Double (SPU Only) 7  
      cbrtf4: Cube Root of Float 7  
      ceilf2: Ceiling of Double (SPU Only) 7  
      ceilf4: Ceiling of Float 7  
      copysignd2: Copy Sign of Double (SPU Only) 7  
      copysignf4: Copy Sign of Float 7  
      cosd2: Cosine of Double (SPU Only) 7  
      cosf4: Cosine of Float 7  
      coshd2: Hyperbolic Cosine of Double (SPU Only) 7  
      coshf4: Hyperbolic Cosine of Float 7  
      divd2: Divide Doubles (SPU Only) 7  
      divf4: Divide Floats 7  
      divu4: Divide Unsigned Integer 7  
      erfcd2: Complementary Error Function Double (SPU Only) 7  
      erfcsd2: Complementary Error Function Double (SPU Only) 7  
      erfcf4: Complementary Error Function Float 7  
      erfd2: Error Function Double (SPU Only) 7  
      erfd2: Error Function Double (SPU Only) 7  

SIMD Math Library Specification for Cell Broadband Engine Architecture, Version 1.0
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>erff4: Error Function Float</td>
<td>11</td>
</tr>
<tr>
<td>expd2: $e$ Raised to the Power of Double (SPU Only)</td>
<td>11</td>
</tr>
<tr>
<td>expf4: $e$ Raised to the Power of Float</td>
<td>11</td>
</tr>
<tr>
<td>exp2d2: 2 Raised to the Power of Double (SPU Only)</td>
<td>11</td>
</tr>
<tr>
<td>exp2f4: 2 Raised to the Power of Float</td>
<td>11</td>
</tr>
<tr>
<td>expm1d2: $e$ Raised to the Power of Double Minus 1 (SPU Only)</td>
<td>12</td>
</tr>
<tr>
<td>expm1f4: $e$ Raised to the Power of Float Minus 1</td>
<td>12</td>
</tr>
<tr>
<td>fabsd2: Absolute Value Double (SPU Only)</td>
<td>12</td>
</tr>
<tr>
<td>fabsf4: Absolute Value Float</td>
<td>12</td>
</tr>
<tr>
<td>fdmd2: Subtract Staying Non-Negative Double (SPU Only)</td>
<td>12</td>
</tr>
<tr>
<td>fdmf4: Subtract Staying Non-Negative Float</td>
<td>12</td>
</tr>
<tr>
<td>floor2d: Floor Double (SPU Only)</td>
<td>12</td>
</tr>
<tr>
<td>floorf4: Floor Float</td>
<td>12</td>
</tr>
<tr>
<td>fmad2: Fused Multiply and Add Double (SPU Only)</td>
<td>13</td>
</tr>
<tr>
<td>fmaf4: Fused Multiply and Add Float</td>
<td>13</td>
</tr>
<tr>
<td>fmaxd2: Maximum Double (SPU Only)</td>
<td>13</td>
</tr>
<tr>
<td>fmaxf4: Maximum Float</td>
<td>13</td>
</tr>
<tr>
<td>fmin2d: Minimum Double (SPU Only)</td>
<td>13</td>
</tr>
<tr>
<td>fminf4: Minimum Float</td>
<td>13</td>
</tr>
<tr>
<td>fmodd2: Modulus Double (SPU Only)</td>
<td>13</td>
</tr>
<tr>
<td>fmodf4: Modulus Float</td>
<td>13</td>
</tr>
<tr>
<td>fpclassifyd2: Classify Double (SPU Only)</td>
<td>14</td>
</tr>
<tr>
<td>fpclassifyf4: Classify Float</td>
<td>14</td>
</tr>
<tr>
<td>frexp2d: Represent Double as Fraction and Exponent (SPU Only)</td>
<td>14</td>
</tr>
<tr>
<td>frexp4: Represent Float as Fraction and Exponent</td>
<td>14</td>
</tr>
<tr>
<td>hypold2: Hypotenuse Double (SPU Only)</td>
<td>14</td>
</tr>
<tr>
<td>hypotf4: Hypotenuse Float</td>
<td>15</td>
</tr>
<tr>
<td>ilogbd2: Integer Exponent of Double (SPU Only)</td>
<td>15</td>
</tr>
<tr>
<td>ilogbf4: Integer Exponent of Float</td>
<td>15</td>
</tr>
<tr>
<td>intf4: Nearest Integer Float</td>
<td>15</td>
</tr>
<tr>
<td>rintf4: Round Float to Nearest Integer</td>
<td>15</td>
</tr>
<tr>
<td>is0denormd2: 0 or Denormalized Double (SPU Only)</td>
<td>16</td>
</tr>
<tr>
<td>is0denormf4: 0 or Denormalized Float</td>
<td>16</td>
</tr>
<tr>
<td>isequald2: Compare Equal Double (SPU Only)</td>
<td>16</td>
</tr>
<tr>
<td>isequalf4: Compare Equal Float</td>
<td>16</td>
</tr>
<tr>
<td>isnfinite2d: Double is Finite (SPU Only)</td>
<td>16</td>
</tr>
<tr>
<td>isnfinitef4: Float is Finite</td>
<td>16</td>
</tr>
<tr>
<td>isgreaterequald2: Greater or Equal Double (SPU Only)</td>
<td>17</td>
</tr>
<tr>
<td>isgreaterequalf4: Greater or Equal Float</td>
<td>17</td>
</tr>
<tr>
<td>isgreaterd2: Greater than Double (SPU Only)</td>
<td>17</td>
</tr>
<tr>
<td>isgreaterf4: Greater than Float</td>
<td>17</td>
</tr>
<tr>
<td>isinf2d: Double is Infinity (SPU Only)</td>
<td>17</td>
</tr>
<tr>
<td>isinf4: Float is Infinity</td>
<td>18</td>
</tr>
<tr>
<td>islesssd2: Double is Less Than (SPU Only)</td>
<td>18</td>
</tr>
<tr>
<td>islesseguequald2: Double is Less Than or Equal To (SPU Only)</td>
<td>18</td>
</tr>
<tr>
<td>islesseguequalf4: Float is Less Than or Equal To</td>
<td>18</td>
</tr>
<tr>
<td>islessf4: Float is Less Than</td>
<td>18</td>
</tr>
<tr>
<td>islessgreaterd2: Double is Less Than or Greater Than (SPU Only)</td>
<td>19</td>
</tr>
<tr>
<td>islessgreaterf4: Float is Less Than or Greater Than</td>
<td>19</td>
</tr>
<tr>
<td>isnand2: Double is NaN (SPU Only)</td>
<td>19</td>
</tr>
<tr>
<td>isnanf4: Float is NaN</td>
<td>19</td>
</tr>
<tr>
<td>isnormald2: Double is Normal (SPU Only)</td>
<td>19</td>
</tr>
<tr>
<td>isnormalf4: Float is Normal</td>
<td>19</td>
</tr>
<tr>
<td>isnorderedd2: Double is Ordered (SPU Only)</td>
<td>20</td>
</tr>
<tr>
<td>isnorderedf4: Float is Ordered</td>
<td>20</td>
</tr>
<tr>
<td>ldexpd2: Multiply Double by 2 Raised to its Power (SPU Only)</td>
<td>20</td>
</tr>
<tr>
<td>ldexpf4: Multiply Float by 2 Raised to its Power</td>
<td>20</td>
</tr>
<tr>
<td>lgammaad2: Natural Log of Gamma Function of Double (SPU Only)</td>
<td>20</td>
</tr>
<tr>
<td>lgammaaf4: Natural Log of Gamma Function of Float</td>
<td>20</td>
</tr>
<tr>
<td>labs2d: Absolute Value Long Long (SPU Only)</td>
<td>20</td>
</tr>
<tr>
<td>labsf2: Absolute Value Long Long (SPU Only)</td>
<td>20</td>
</tr>
<tr>
<td>lddiv2: Divide Long Long (SPU Only)</td>
<td>21</td>
</tr>
<tr>
<td>lddivu2: Divide Unsigned Long Long (SPU Only)</td>
<td>21</td>
</tr>
</tbody>
</table>
llrintd2: Find Nearest Long Long of Double (SPU Only) 21
llrintf4: Find Nearest Long Long of Float (SPU Only) 21
llroundd2: Round Double to Nearest Long Long (SPU Only) 21
llroundf4: Round Float to Nearest Long Long (SPU Only) 21
logd2: Natural Log of Double (SPU Only) 21
logf4: Natural Log of Float 22
log10d2: Log Base 10 of Double (SPU Only) 22
log10f4: Log Base 10 of Float 22
log1pd2: Natural Log of Double Plus 1 (SPU Only) 22
log1pf4: Natural Log of Float Plus 1 22
log2d2: Log Base 2 of Double (SPU Only) 22
log2f4: Log Base 2 of Float 22
logbd2: Represent Double as Fraction Greater Than 1 and Exponent (SPU Only) 23
logbf4: Represent Float as Fraction Greater Than 1 and Exponent 23
modfd2: Represent Double as Proper Fraction and Exponent (SPU Only) 23
modff4: Represent Float as Proper Fraction and Exponent 23
nearbyintd2: Find Nearest Integer for Double (SPU Only) 24
nearbyintf4: Find Nearest Integer for Float 24
negated2: Negate Double (SPU Only) 24
negatef4: Negate Float 24
negatei4: Negate Signed Integer 24
negateil2: Negate Signed Long Long Integer (SPU Only) 24
nextafterd2: Find Next Integer After for Double (SPU Only) 24
nextafterf4: Find Next Integer After for Float 24
powd2: Raise Double to Double Power (SPU Only) 24
powf4: Raise Float to Float Power 25
recipd2: Reciprocal of Double (SPU Only) 25
recipf4: Reciprocal of Float 25
remainderd2: Remainder of Doubles (SPU Only) 25
remainderf4: Remainder of Floats 25
remquod2: Remainder Function of Double (SPU Only) 26
remquof4: Remainder Function of Float 26
rintd2: Round Double to the Nearest Integer (SPU Only) 26
rintf4: Round Float to the Nearest Integer 26
roundd2: Round Double (SPU Only) 26
roundf4: Round Float 26
rsqrd2: Reciprocal Square Root of Double (SPU Only) 26
rsrqtf4: Reciprocal Square Root of Float 27
scalblld2: Scale Double by Long Long Integer (SPU Only) 27
scalbnf4: Scale Float by Integer 27
signbitd2: Sign Bit of Double (SPU Only) 27
signbitf4: Sign Bit of Float 27
sincosd2: Sine and Cosine of Double (SPU Only) 27
sincosf4: Sine and Cosine of Float 27
sind2: Sine of Double (SPU Only) 28
sinf4: Sine of Float 28
sinhd2: Hyperbolic Sine of Double (SPU Only) 28
sinhf4: Hyperbolic Sine of Float 28
srtld2: Square Root of Double (SPU Only) 28
sqtlf4: Square Root of Float 28
tand2: Tangent of Double (SPU Only) 29
tanf4: Tangent of Float 29
tanhd2: Hyperbolic Tangent of Double (SPU Only) 29
tanhf4: Hyperbolic Tangent of Float 29
tgammad2: Gamma of Double (SPU Only) 29
tgammalf4: Gamma of Float 29
trucd2: Truncate Double (SPU Only) 29
truncf4: Truncate Float 29

Index 31
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™ 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.

<table>
<thead>
<tr>
<th>Version Number &amp; Date</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>v. 1.0 November 6, 2006</td>
<td>Created the initial document.</td>
</tr>
</tbody>
</table>

Related Documentation

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

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ Language Extensions for the Cell Broadband Engine Architecture</td>
<td>2.2.1</td>
<td>November 2006</td>
</tr>
<tr>
<td>ISO/IEC Standard 9899:1999 (C Standard)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEC Standard 60559:1989 (Standard for Binary Floating-Point Arithmetic)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synergistic Processor Unit Instruction Set Architecture</td>
<td>1.11</td>
<td>October 2006</td>
</tr>
<tr>
<td>PPE User's Manual</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>PowerPC User Instruction Set Architecture, Book I</td>
<td>2.02</td>
<td>January 2005</td>
</tr>
<tr>
<td>PowerPC Virtual Environment Architecture, Book II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerPC Operating Environment Architecture, Book III</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 that compose this library.
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:

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>...</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LSB</td>
</tr>
</tbody>
</table>
```

MSB = Most significant bit
LSB = Least significant bit

Notation for bit encoding is as follows:
- Hexadecimal values are preceded by 0x. For example: 0xA00.
- Binary values in sentences appear in single quotation marks. For example: ‘1010’.

Byte Ordering and Element Numbering

As shown in Figure 1, byte ordering and element/slot numbering is always displayed in big endian order.

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

```
<table>
<thead>
<tr>
<th>Byte 0 (MSB)</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
<th>Byte 12</th>
<th>Byte 13</th>
<th>Byte 14</th>
<th>Byte 15 (LSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doubleword 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>word 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doubleword 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>word 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halfword 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Typographic Conventions

In addition to bit notation, the following typographic conventions are used throughout this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>courier</td>
<td>Indicates programming code, processing instructions, register names, data types, events, file names, and other literals. Also indicates function and macro names. This convention is used only where it facilitates comprehension, especially in narrative descriptions.</td>
</tr>
<tr>
<td>courier + italics</td>
<td>Indicates arguments, parameters and variables, including variables of type const. This convention is used only where it facilitates comprehension, especially in narrative descriptions.</td>
</tr>
<tr>
<td>italics (without courier)</td>
<td>Indicates emphasis. Except when hyperlinked, book references are in italics. When a term is first defined, it is likely to be in italics.</td>
</tr>
<tr>
<td>Blue</td>
<td>Indicates a hyperlink (color printers or online only).</td>
</tr>
</tbody>
</table>
1. Overview of the SIMD Math Library

The PPU and SPU instruction sets include Single Instruction, Multiple Data (SIMD) instructions, which 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 libsimmath.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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>C99 Name</th>
<th>Function Category</th>
<th>Precision</th>
<th>SPU/PPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>absi4</td>
<td>abs</td>
<td>integer</td>
<td>word</td>
<td>Both</td>
</tr>
<tr>
<td>acosd2</td>
<td>acos</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>acosf4</td>
<td>acosf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>acoshd2</td>
<td>acosh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>acoshf4</td>
<td>acoshf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>asind2</td>
<td>asin</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>asinf4</td>
<td>asinf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>asinhd2</td>
<td>asinh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>asinhf4</td>
<td>asinhf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>atan2d2</td>
<td>atan2</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>atan2f4</td>
<td>atan2f</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>atand2</td>
<td>atan</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>atanf4</td>
<td>atanf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>atanhd2</td>
<td>atanh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>atanhf4</td>
<td>atanhf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>cbrd2</td>
<td>cbt</td>
<td>power</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>cbrtf4</td>
<td>cbtf</td>
<td>power</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>ceild2</td>
<td>ceil</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>ceilf4</td>
<td>ceiff</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>copysignd2</td>
<td>copysign</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>copysignf4</td>
<td>copysignf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>Function Name</td>
<td>C99 Name</td>
<td>Function Category</td>
<td>Precision</td>
<td>SPU/PPU</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>-------------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>cosd2</td>
<td>cos</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>cosf4</td>
<td>cosf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>coshd2</td>
<td>cosh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>coshf4</td>
<td>coshf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>divd2</td>
<td>non-standard</td>
<td>divide</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>divf4</td>
<td>non-standard</td>
<td>divide</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>divi4</td>
<td>div</td>
<td>integer</td>
<td>word</td>
<td>Both</td>
</tr>
<tr>
<td>divu4</td>
<td>non-standard</td>
<td>integer</td>
<td>word</td>
<td>Both</td>
</tr>
<tr>
<td>erfcd2</td>
<td>erfc</td>
<td>error function</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>erfcf4</td>
<td>erfcf</td>
<td>error function</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>erfdo</td>
<td>erf</td>
<td>error function</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>erff4</td>
<td>erff</td>
<td>error function</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>exp2d2</td>
<td>exp2</td>
<td>exp</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>exp2f4</td>
<td>exp2f</td>
<td>exp</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>expd2</td>
<td>exp</td>
<td>exp</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>expf4</td>
<td>expf</td>
<td>exp</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>expm1d2</td>
<td>expm1</td>
<td>exp</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>expm1f4</td>
<td>expm1f</td>
<td>exp</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fabsd2</td>
<td>fabs</td>
<td>abs</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fabsf4</td>
<td>fabsf</td>
<td>abs</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fdimd2</td>
<td>fdim</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fdimf4</td>
<td>fdimf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>floor2</td>
<td>floor</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>floorf4</td>
<td>floorf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fmad2</td>
<td>fma</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fmaf4</td>
<td>fmaf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fmaxd2</td>
<td>fmax</td>
<td>minmax</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fmaxf4</td>
<td>fmaxf</td>
<td>minmax</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fmind2</td>
<td>fmin</td>
<td>minmax</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fminf4</td>
<td>fminf</td>
<td>minmax</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fmodd2</td>
<td>fmod</td>
<td>modrem</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fmodf4</td>
<td>fmodf</td>
<td>modrem</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>fpclassifyd2</td>
<td>fpclassify</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>fpclassifyf4</td>
<td>fpclassify</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>frexp2d</td>
<td>frexp</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>frexp4f</td>
<td>frexpf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>hypot2</td>
<td>hypot</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>hypotf4</td>
<td>hypotf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>iogbd2</td>
<td>iogb</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>iogbf4</td>
<td>iogbf</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>irintf4</td>
<td>non-standard</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>iroundf4</td>
<td>non-standard</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>is0denormf4</td>
<td>non-standard</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>Function Name</td>
<td>C99 Name</td>
<td>Function Category</td>
<td>Precision</td>
<td>SPU/PPU</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>is0denormd2</td>
<td>non-standard</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isequald2</td>
<td>non-standard</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isequalf4</td>
<td>non-standard</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isfinited2</td>
<td>isfinite</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isfinitef4</td>
<td>isfinite</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isgreaterd2</td>
<td>isgreater</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isgreaterf4</td>
<td>isgreater</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isgreatequald2</td>
<td>isgreatequal</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isgreatequalf4</td>
<td>isgreatequal</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>islessd2</td>
<td>isless</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>islessf4</td>
<td>isless</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>islessequald2</td>
<td>islessequal</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>islessequalf4</td>
<td>islessequal</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>islessgreaterd2</td>
<td>islessgreater</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>islessgreaterf4</td>
<td>islessgreater</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isnorderedd2</td>
<td>isunordered</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isnorderedf4</td>
<td>isunordered</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isnfd2</td>
<td>isinf</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isnff4</td>
<td>isinf</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isnand2</td>
<td>isnan</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isnanf4</td>
<td>isnan</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>isnormald2</td>
<td>isnormal</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>isnormalf4</td>
<td>isnormal</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>ldexpd2</td>
<td>ldexp</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>ldexpf4</td>
<td>ldexpf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>lgammad2</td>
<td>lgamma</td>
<td>gamma</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>lgammaf4</td>
<td>lgammaf</td>
<td>gamma</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>llabsi2</td>
<td>llabs</td>
<td>integer</td>
<td>double word</td>
<td>SPU</td>
</tr>
<tr>
<td>lldivi2</td>
<td>lldiv</td>
<td>integer</td>
<td>double word</td>
<td>SPU</td>
</tr>
<tr>
<td>lldivu2</td>
<td>non-standard</td>
<td>integer</td>
<td>double word</td>
<td>SPU</td>
</tr>
<tr>
<td>llrintd2</td>
<td>llrint</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>llrintf4</td>
<td>llrintf</td>
<td>rounding</td>
<td>single</td>
<td>SPU</td>
</tr>
<tr>
<td>llroundd2</td>
<td>llround</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>llroundf4</td>
<td>llroundf</td>
<td>rounding</td>
<td>single</td>
<td>SPU</td>
</tr>
<tr>
<td>log10d2</td>
<td>log10</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>log10f4</td>
<td>log10f</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>log1pd2</td>
<td>log1p</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>log1pf4</td>
<td>log1pf</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>log2d2</td>
<td>log2</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>log2f4</td>
<td>log2f</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>logbd2</td>
<td>logb</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>logbf4</td>
<td>logbf</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>logd2</td>
<td>log</td>
<td>log</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>Function Name</td>
<td>C99 Name</td>
<td>Function Category</td>
<td>Precision</td>
<td>SPU/PPU</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>------------------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>logf4</td>
<td>logf</td>
<td>log</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>modfd2</td>
<td>modf</td>
<td>modrem</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>modff4</td>
<td>modff</td>
<td>modrem</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>nearbyintd2</td>
<td>nearbyint</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>nearbyintf4</td>
<td>nearbyintf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>negated2</td>
<td>non-standard</td>
<td>rounding</td>
<td>integer</td>
<td>Both</td>
</tr>
<tr>
<td>negatef4</td>
<td>non-standard</td>
<td>other</td>
<td>integer</td>
<td>Both</td>
</tr>
<tr>
<td>negatei4</td>
<td>non-standard</td>
<td>integer</td>
<td>word</td>
<td>Both</td>
</tr>
<tr>
<td>negatet2</td>
<td>non-standard</td>
<td>integer</td>
<td>double word</td>
<td>SPU</td>
</tr>
<tr>
<td>nextafterd2</td>
<td>nextafter</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>nextafterf4</td>
<td>nextafterf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>powd2</td>
<td>pow</td>
<td>power</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>powf4</td>
<td>powf</td>
<td>power</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>recipd2</td>
<td>non-standard</td>
<td>recip</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>recipf4</td>
<td>non-standard</td>
<td>recip</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>remainderd2</td>
<td>remainder</td>
<td>modrem</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>remainderf4</td>
<td>remainderf</td>
<td>modrem</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>remquod2</td>
<td>remquo</td>
<td>modrem</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>remquof4</td>
<td>remquof</td>
<td>modrem</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>rintd2</td>
<td>rint</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>rintf4</td>
<td>rintf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>roundd2</td>
<td>round</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>roundf4</td>
<td>roundf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>rsqrtd2</td>
<td>non-standard</td>
<td>sqrt</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>rsqrtf4</td>
<td>non-standard</td>
<td>sqrt</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>scalblnd2</td>
<td>non-standard</td>
<td>other</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>scalbnf4</td>
<td>scalbnf</td>
<td>other</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>signbitf4</td>
<td>signbit</td>
<td>comparison</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>signbitd2</td>
<td>signbit</td>
<td>comparison</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>sincosd2</td>
<td>non-standard</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>sincosf4</td>
<td>non-standard</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>sind2</td>
<td>sin</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>sinf4</td>
<td>sinf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>sinh2d</td>
<td>sinh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>sinhf4</td>
<td>sinhf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>sqrtdd2</td>
<td>sqrt</td>
<td>sqrt</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>sqrtf4</td>
<td>sqrtf</td>
<td>sqrt</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>tand2</td>
<td>tan</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>tanf4</td>
<td>tanf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>tanhd2</td>
<td>tanh</td>
<td>trig</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>tanhf4</td>
<td>tanhf</td>
<td>trig</td>
<td>single</td>
<td>Both</td>
</tr>
<tr>
<td>tgammad2</td>
<td>tgamma</td>
<td>gamma</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>tgammaf4</td>
<td>tgammaf</td>
<td>gamma</td>
<td>single</td>
<td>Both</td>
</tr>
</tbody>
</table>
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* and *Inf* 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 which 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 on 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 fp exceptions; PPU SIMD operations do have hardware support for a subset of the C99 fp exceptions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>C99 Name</th>
<th>Function Category</th>
<th>Precision</th>
<th>SPU/PPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>truncd2</td>
<td>trunc</td>
<td>rounding</td>
<td>double</td>
<td>SPU</td>
</tr>
<tr>
<td>truncf4</td>
<td>truncf</td>
<td>rounding</td>
<td>single</td>
<td>Both</td>
</tr>
</tbody>
</table>
2. SIMD Function Specifications

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

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**

\[(vector \text{ signed int}) \; \text{absi4} \; (vector \text{ signed int } x)\]

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)**

\[(vector \text{ double}) \; \text{acosd2} \; (vector \text{ double } x)\]

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**

\[(vector \text{ float}) \; \text{acosf4} \; (vector \text{ float } x)\]

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)**

\[(vector \text{ double}) \; \text{acoshd2} \; (vector \text{ double } x)\]

A vector double is returned that contains the nonnegative 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**

\[(vector \text{ float}) \; \text{acoshf4} \; (vector \text{ float } x)\]

A vector float is returned that contains the nonnegative 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)**

\[(vector \text{ double}) \; \text{asind2} \; (vector \text{ double } x)\]

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**

\[(vector \text{ float}) \; \text{asinf4} \; (vector \text{ float } x)\]

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 nonnegative hyperbolic arcsines of the corresponding elements of vector double $x$.

**asinhf4: Hyperbolic Arcsine of Float**

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

A vector float is returned that contains the nonnegative 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) atanhd2 (vector double x)
```

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

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

On the SPU, if the absolute value of $x_i$ is equal to 1, the corresponding element of the result will be returned as $\text{HUGE\_VALF}$ and 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 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 zero, 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 \( \frac{y_i}{x_i} \) for 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 zero, the corresponding element of the result is undefined, and no error is reported.

cbrtd2: Cube Root of Double (SPU Only)

\[
\text{(vector double) cbrtd2 (vector double } x)\; \\
\text{A vector double is returned that contains the real cube roots, } x_i^{1/3} \text{ of the corresponding elements of vector double } x.
\]

cbrtf4: Cube Root of Float

\[
\text{(vector float) cbrtf4 (vector float } x)\; \\
\text{A vector float is returned that contains the real cube roots, } x_i^{1/3} \text{ of the corresponding elements of vector float } x.
\]

celd2: Ceiling of Double (SPU Only)

\[
\text{(vector double) ceild2 (vector double } x)\; \\
\text{A vector double is returned that contains the smallest integer values, expressed as floating-point numbers, that are not less than the corresponding elements of vector double } x.
\]

celf4: Ceiling of Float

\[
\text{(vector float) ceilf4 (vector float } x)\; \\
\text{A vector float is returned that contains the smallest integer values, expressed as floating-point numbers, that are not less than the corresponding elements of vector float } x.
\]

copysignd2: Copy Sign of Double (SPU Only)

\[
\text{(vector double) copysignd2 (vector double } x, \text{ vector double } y)\; \\
\text{A vector double is returned that contains the magnitude of the corresponding element of vector double } x \text{ and the sign of the corresponding element of vector double } y.
\]

copysignf4: Copy Sign of Float

\[
\text{(vector float) copysignf4 (vector float } x, \text{ vector float } y)\; \\
\text{A vector float is returned that contains the magnitude of the corresponding element of vector float } x \text{ and the sign of the corresponding element of vector float } y.
\]

cosd2: Cosine of Double (SPU Only)

\[
\text{(vector double) cosd2 (vector double } x)\; \\
\text{A vector double is returned that contains the cosines of the corresponding elements of vector double } x.
\]

The results of \text{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() are may not be accurate for very large values, 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 quotient $x_i/y_i$, 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 zero, 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 $x_i = quot * y_i + rem$.

If $y_i$ is zero, the corresponding element of the resulting quotient is zero.
divu4: Divide Unsigned Integer

\( \text{divu4} (\text{vector unsigned int } x, \text{vector unsigned int } y) \)

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 \( \text{divu4}_t() \) that contains a vector of corresponding quotients and a vector of corresponding remainders.

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

If \( y_i \) is zero, the corresponding element of the resulting quotient is zero.

erfcd2: Complementary Error Function Double (SPU Only)

\( \text{erfcd2} (\text{vector double } x) \)

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

ercf4: Complementary Error Function Float

\( \text{erfcf4} (\text{vector float } x) \)

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)

\( \text{erfd2} (\text{vector double } x) \)

A vector double is returned that contains the error functions of the corresponding elements of vector double \( x \).

erff4: Error Function Float

\( \text{erff4} (\text{vector float } x) \)

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)

\( \text{expd2} (\text{vector double } x) \)

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

\( \text{expf4} (\text{vector float } x) \)

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 \( \text{HUGE\_VALF} \) are returned as \( \text{HUGE\_VALF} \), and no error is reported.

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

\( \text{exp2d2} (\text{vector double } x) \)

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

\( \text{exp2f4} (\text{vector float } x) \)

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)**

`expm1d2 (vector double x);`

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

This function returns mathematically accurate values, even when $x_i$ is near 0, 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**

`expm1f4 (vector float x);`

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

This function returns mathematically accurate values, even when an element of $x_i$ is near 0, or when $\expf (x_i) - 1.0f$ would return bad values due to floating-point cancellation errors.

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

`fabsd2 (vector double x);`

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

**fabsf4: Absolute Value Float**

`fabsf4 (vector float x);`

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

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

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

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

**fdimf4: Subtract Staying Non-Negative 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 corresponding elements of vector float $x$ and vector float $y$.

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

`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**

`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)**

\[(\text{vector double}) \text{fmad2}(\text{vector double } x, \text{vector double } y, \text{vector double } z);\]

A vector double is returned that contains the results of the calculation of \((x_i \times y_i + z_i)\), 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**

\[(\text{vector float}) \text{fmaf4}(\text{vector float } x, \text{vector float } y, \text{vector float } z);\]

A vector float is returned that contains the results of the calculation of \((x_i \times y_i + z_i)\), 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)**

\[(\text{vector double}) \text{fmaxd2}(\text{vector double } x, \text{vector double } y);\]

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

**fmaxf4: Maximum Float**

\[(\text{vector float}) \text{fmaxf4}(\text{vector float } x, \text{vector float } y);\]

A vector float is returned that contains the larger (more positive) of \(x_i\) and \(y_i\), for 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)**

\[(\text{vector double}) \text{fmind2}(\text{vector double } x, \text{vector double } y);\]

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

**fminf4: Minimum Float**

\[(\text{vector float}) \text{fminf4}(\text{vector float } x, \text{vector float } y);\]

A vector float is returned that contains the smaller (more negative) of \(x_i\) and \(y_i\), for 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)**

\[(\text{vector double}) \text{fmodd2}(\text{vector double } x, \text{vector double } y);\]

A vector double is returned that contains the remainder of \(x_i/y_i\), for corresponding elements of vector double \(x\) and vector double \(y\), as described below:

- If \(y_i\) is 0, the result is 0
- Otherwise, the function determines the unique signed integer value \(i\) such that the returned element is \(x_i - i \times y_i\) with the same sign as \(x_i\) and magnitude less than \(|y_i|\)

**fmodf4: Modulus Float**

\[(\text{vector float}) \text{fmodf4}(\text{vector float } x, \text{vector float } y);\]
A vector float is returned that contains the remainder of \( x_i / y_i \), for corresponding elements of vector float \( x \) and vector float \( y \), as defined below:

- If \( y_i = 0 \), the result is 0
- Otherwise, \( \text{fmodf4()} \) determines the unique signed integer value \( i \) such that the returned element is \( x_i - i * y_i \) with the same sign as \( x_i \) and magnitude less than \( |y_i| \)

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

\[(\text{vector signed long long}) \text{fpclassifyd2} (\text{vector double } x)\]

A vector signed long long is returned that contains the floating-point classifications for 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**

\[(\text{vector signed int}) \text{fpclassifyf4} (\text{vector float } x)\]

A vector signed int is returned that contains the floating-point classifications for 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)**

\[(\text{vector double}) \text{frexpd2} (\text{vector double } x, \text{vector signed long long } *\text{pexp});\]

A vector double is returned that contains normalized fractions, and a vector signed long long is stored in \( *\text{pexp} \) that contains exponent integers. Each fraction element \( \text{frac} \) and each exponent integer element \( \text{exp} \) represents the value of the corresponding element of \( x \), such that:

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

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

\[(\text{vector float}) \text{frexpf4} (\text{vector float } x, \text{vector signed int } *\text{pexp});\]

A vector float is returned that contains normalized fractions, and a vector signed int is stored in \( *\text{pexp} \) that contains exponent integers. Each fraction element \( \text{frac} \) and each exponent integer element \( \text{exp} \) represents the value of the corresponding element of \( x \), such that:

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

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

\[(\text{vector double}) \text{hypotd2} (\text{vector double } x, \text{vector double } y);\]
A vector double is returned that contains the square root of $x^2_i + y^2_i$ without undue overflow or underflow, for 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 $x^2_i + y^2_i$ without undue overflow or underflow, for 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 that contains the elements defined below, for corresponding elements of vector double $x$.

- If $x_i$ is not-a-number (NaN), the value is the macro FP_ILOGBNAN.
- If $x_i$ is equal to zero, the value is the macro FP_ILOGB0.
- If $x_i$ is equal to positive or negative $\text{Inf}$, the value is the macro FP_ILOGB0.
- Otherwise, the result is $(\text{int})\logb(x_i)$.

**ilogbf4: Integer Exponent of Float**

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

A vector signed int is returned that contains the elements defined below, for corresponding elements of vector float $x$.

- If $x_i$ is not-a-number (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 $\text{Inf}$, the value is the macro FP_ILOGB0.
- Otherwise, the result is $(\text{int})\logb(x_i)$.

Because the SPU treats single-precision $\text{Inf}$ 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 that contains the elements defined below for corresponding elements of vector double x.

- All bits of the resulting element are set to 1 if \( x_i \) is a denormalized value or zero.
- Zero otherwise.

is0denormf4: 0 or Denormalized Float

(vector unsigned int) is0denormf4 (vector float x);
A vector unsigned int is returned that contains the elements defined below for corresponding elements of vector float x.

- All bits of the resulting element are set to 1 if \( x_i \) is a denormalized value or zero.
- Zero otherwise.

isequald2: Compare Equal Double (SPU Only)

(vector unsigned long long) isequald2 (vector double x, vector double y);
A vector unsigned long long is returned that contains the elements defined below for 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.
- Zero otherwise.

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

isequalf4: Compare Equal Float

(vector unsigned int) isequalf4 (vector float x, vector float y);
A vector unsigned int is returned that contains the elements defined below for 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.
- Zero otherwise.

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

isfinited2: Double is Finite (SPU Only)

(vector unsigned long long) isfinited2 (vector double x);
A vector unsigned long long is returned that contains the elements defined below for corresponding elements of vector double x.

- All bits of the resulting element are set to 1 if \( x_i \) is finite.
- Zero otherwise.

isfinitef4: Float is Finite

(vector unsigned int) isfinitef4 (vector float x);
A vector unsigned int is returned that contains elements defined below for corresponding elements of vector float x.

- All bits of the resulting element are set to 1 if \( x_i \) is finite.
- Zero otherwise.
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)**

\[(\text{vector unsigned long long}) \hspace{1em} \text{isgreaterequald2 (vector double } x, \text{ vector double } y)\];

A vector unsigned long long is returned that contains elements defined below for corresponding elements of vector double \( x \) and vector double \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) equals to or greater than \( y_i \).
- Zero otherwise.

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**

\[(\text{vector unsigned int}) \hspace{1em} \text{isgreaterequalf4 (vector float } x, \text{ vector float } y)\];

A vector unsigned int is returned that contains elements defined below for corresponding elements of vector float \( x \) and vector float \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) equals to or greater than \( y_i \).
- Zero otherwise.

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)**

\[(\text{vector unsigned long long}) \hspace{1em} \text{isgreaterd2 (vector double } x, \text{ vector double } y)\];

A vector unsigned long long is returned that contains elements defined below for corresponding elements of vector double \( x \) and vector double \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) greater than \( y_i \).
- Zero otherwise.

The function correctly compares denormalized numbers.

**isgreaterf4: Greater than Float**

\[(\text{vector unsigned int}) \hspace{1em} \text{isgreaterf4 (vector float } x, \text{ vector float } y)\];

A vector unsigned int is returned that contains elements defined below for corresponding elements of vector float \( x \) and vector float \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) greater than \( y_i \).
- Zero otherwise.

The function correctly compares denormalized numbers.

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

\[(\text{vector unsigned long long}) \hspace{1em} \text{isinfd2 (vector double } x)\];

A vector unsigned long long is returned that contains elements defined below, for corresponding elements of vector double \( x \).

- All bits of the resulting element are set to 1 if \( x_i \) is infinite.
- Zero otherwise.
**isinff4: Float is Infinity**

```c
(vector unsigned int) isnff4 (vector float x)
```

A vector unsigned long int is returned that contains elements defined below for corresponding elements of vector float `x`.

- All bits of the resulting element are set to 1 if `x_i` is infinite.
- Zero otherwise.

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)**

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

A vector unsigned long long is returned that contains elements defined below, for corresponding elements of vector double `x` and vector double `y`.

- All bits of the resulting element are set to 1 if `x_i` less than `y_i`.
- Zero otherwise.

The function correctly compares denormalized numbers.

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

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

A vector unsigned long long is returned that contains elements defined below for corresponding elements of vector double `x` and vector double `y`.

- All bits of the resulting element are set to 1 if `x_i` less than or equal `y_i`.
- Zero otherwise.

The function correctly compares denormalized numbers.

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

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

A vector unsigned int is returned that contains elements defined below for corresponding elements of vector float `x` and vector float `y`.

- All bits of the resulting element are set to 1 if `x_i` less than or equal `y_i`.
- Zero otherwise.

The function correctly compares denormalized numbers.

**islessf4: Float is Less Than**

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

A vector unsigned int is returned that contains elements defined below for corresponding elements of vector float `x` and vector float `y`.

- All bits of the resulting element are set to 1 if `x_i` less than `y_i`.
- Zero otherwise.

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 that contains elements defined below for corresponding elements of vector double \( x \) and vector double \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) less than or greater than \( y_i \).
- Zero otherwise.

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 that contains elements defined below, for corresponding elements of vector float \( x \) and vector float \( y \).

- All bits of the resulting element are set to 1 if \( x_i \) less than or greater than \( y_i \).
- Zero otherwise.

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 that contains elements defined below for corresponding elements of vector double \( x \).

- All bits of the resulting element are set to 1 if \( x_i \) is a NaN.
- Zero otherwise.

isnanf4: Float is NaN

(vector unsigned int) isnanf4 (vector float x)

A vector unsigned int is returned that contains elements defined below, for corresponding elements of vector float \( x \).

- All bits of the resulting element are set to 1 if \( x_i \) is a NaN.
- Zero otherwise.

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 that contains elements defined below for corresponding elements 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.
- Zero otherwise.

isnormalf4: Float is Normal

(vector unsigned int) isnormalf4 (vector float x)

A vector unsigned int is returned that contains elements defined below for corresponding elements 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.
- Zero otherwise.
isunorderedd2: Double is Unordered (SPU Only)

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

A vector unsigned long long is returned that contains elements defined below for 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 to the element of \( y_i \).
- Zero otherwise.

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 that contains elements defined below for 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 \( y_i \).
- Zero otherwise.

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 zero.

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 \( x_i \times 2^{e_i} \) for the corresponding elements of vector double \( x \) and vector
signed long long \( \text{ex} \). For large elements of \( \text{ex} \) (overflow), the element in the result saturates to HUGE_VAL with an
appropriate sign. For small elements of \( \text{ex} \) (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 \( x_i \times 2^{e_i} \) for the corresponding elements of vector float \( x \) and vector signed
int \( \text{ex} \). For large elements of \( \text{ex} \) (overflow), the element in the result saturates to HUGE_VALF with an appropriate
sign. For small \( \text{ex} \) (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 \( x_i = quot \times y_i + rem \).

If \( y_i \) is zero, the corresponding element of the resulting quotient is zero.

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(), containing 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 \( x_i = quot \times y_i + rem \).

If \( y_i \) is zero, the corresponding element of the resulting quotient is zero.

Ilrintd2: Find Nearest Long Long of Double (SPU Only)

(vector signed long long) ilrintd2 (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.

Ilrintf4: Find Nearest Long Long of Float (SPU Only)

(llroundf4_t) ilrintf4 (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.

Ilroundd2: Round Double to Nearest Long Long (SPU Only)

(vector signed long long) ilroundd2 (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 0 regardless of the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified.

Ilroundf4: Round Float to Nearest Long Long (SPU Only)

(llroundf4_t) ilroundf4 (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 0 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**

\[
\text{(vector float) logf4 (vector float x);}\]

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 zero, the result is \(-\text{HUGEVALF}\).

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

\[
\text{(vector double) log10d2 (vector double x);}\]

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**

\[
\text{(vector float) log10f4 (vector float x);}\]

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 zero, the result is \(-\text{HUGEVALF}\).

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

\[
\text{(vector double) log1pd2 (vector double x);}\]

A vector double is returned that contains the natural logarithm of \(1 + x_i\) for the corresponding elements of vector double \(x\). The function returns mathematically accurate values even when the corresponding element of \(x_i\) 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**

\[
\text{(vector float) log1pf4 (vector float x);}\]

A vector float is returned that contains the natural logarithms of \(1 + x_i\), for corresponding elements of vector float \(x\). The function returns mathematically accurate values even when the corresponding element of \(x_i\) is near zero. If an element of \(x_i\) is \(-1\), the result is \(-\text{HUGEVALF}\). 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)**

\[
\text{(vector double) log2d2 (vector double x);}\]

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 zero, the result is \(-Huge\_valf\).

If \( x_i \) is less than zero, 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 \( frac_i \) that represents the value of a finite element are determined for corresponding elements of vector double \( x \). A vector double is returned that contains the value of \( ex_i \) for \( x_i \), such that:

- \( x_i = frac_i \times FLT\_RADIX^{ex_i} \)
- \( |frac_i| \) is in the interval \([1, \text{FLT\_RADIX})\).

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 a NaN, the corresponding result is also a NaN.

logbf4: Represent Float as Fraction Greater Than 1 and Exponent

(vector float) logbf4 (vector float x);

An integer exponent \( ex_i \) and a fraction \( frac_i \) that represents the value of a finite element are determined for corresponding elements of vector float \( x \). A vector float is returned that contains the values of \( ex_i \) for \( x_i \), such that:

- \( x_i = frac_i \times FLT\_RADIX^{ex_i} \)
- \( |frac_i| \) is in the interval \([1, \text{FLT\_RADIX})\).

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 \( i \) and a fractional part \( frac \). A vector double is returned that contains the corresponding \( frac \) elements, and another vector double is stored in \( *pint \) that contains the corresponding \( i \) elements, such that:

- \( x_i = frac + i \)
- \( |frac| \) is in the interval \([0, 1)\).
- both \( frac \) and \( i \) 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 \( i \) and a fractional part \( frac \). A vector float is returned that contains the corresponding \( frac \) elements, and another vector float is stored in \( *pint \) that contains the corresponding \( i \) elements, such that:

- \( x_i = frac + i \)
- \( |frac| \) is in the interval \([0, 1)\).
- both \( frac \) and \( i \) 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 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 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 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 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 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 \( x_i \) 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 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 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**

```c
(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 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)**

```c
(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 Inf with the sign of $x_i$.
- When $x_i$ is NaN, the result is NaN.

**recipf4: Reciprocal of Float**

```c
(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 an element of $x_i$ is NaN, the result is NaN.

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

```c
(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 zero, the corresponding element of the result is undefined and no error is reported.

**remainderf4: Remainder of Floats**

```c
(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 zero, 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 that 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 that 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.

rsqrtd2: Reciprocal Square Root of Double (SPU Only)

(vector double) rsqrtd2 (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 zero, 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 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 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 that contains the elements defined below for corresponding elements of vector double \( x \).

- All bits of the resulting element are set to 1 if the sign bit is set in \( x_i \)
- Zero otherwise

signbitf4: Sign Bit of Float

(vector unsigned int) signbitf4 (vector float x)

A vector unsigned int is returned that contains the elements defined below for corresponding elements of vector float \( x \).

- All bits of the resulting element are set to 1 if the sign bit is set in \( x_i \)
- Zero otherwise

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 contains 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)**

\[
\text{(vector double) sind2 (vector double x)};
\]

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**

\[
\text{(vector float) sinf4 (vector float x)};
\]

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)**

\[
\text{(vector double) sinhd2 (vector double x)}
\]

A vector double is returned that contains the corresponding hyperbolic sines of the elements of vector double \( x \).

**sinhf4: Hyperbolic Sine of Float**

\[
\text{(vector float) sinhf4 (vector float x)}
\]

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)**

\[
\text{(vector double) sqrtd2 (vector double x)};
\]

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**

\[
\text{(vector float) sqrtf4 (vector float x)};
\]

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 containing 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 containing 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.

tanhd2: 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.

trucnd2: Truncate Double (SPU Only)

(vector double) trucnd2 (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.

trucnf4: Truncate Float

(vector float) trucnf4 (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.
SIMD Function Specifications, Function

Definitions

absi4 ...................................................... 7
acosd2 ..................................................... 7
acosf2 ..................................................... 7
acosfd2 ................................................... 7
acosf4 ..................................................... 7
asin2 ....................................................... 7
asin4 ....................................................... 7
atan2d2 .................................................... 8
atan2f4 .................................................... 8
atand2 ..................................................... 8
atand4 ..................................................... 8
atanh2 ..................................................... 8
atanhf4 .................................................... 8
cbrtd2 ..................................................... 9
cbrtf4 ..................................................... 9
celd2 ..................................................... 9
celf4 ...................................................... 9
copysignfd2 ............................................. 9
copysignf4 .............................................. 9
cosd2 ..................................................... 9
cosf4 ..................................................... 10
coshd2 ................................................... 10
cohsf4 ................................................... 10
divd2 ..................................................... 10
divf4 ..................................................... 10
div4 ....................................................... 10
diu4 ....................................................... 11
erfc2 ...................................................... 11
erfcf4 ..................................................... 11
errf2 ...................................................... 11
errf4 ...................................................... 11
exp2d2 .................................................... 11
exp2f4 .................................................... 11
expd2 ..................................................... 11
expf4 ..................................................... 11
expm1d2 .................................................. 12
expm1f4 .................................................. 12
fabsd2 .................................................... 12
fabsf4 .................................................... 12
fdimd2 .................................................... 12
fdimf4 .................................................... 12
floor2 .................................................... 12
floorf4 ................................................... 12
fmod2 ..................................................... 13
fmodf4 ................................................... 13
fpclassifyd2 ............................................ 14
fpclassifyf4 ............................................ 14
frexp2 .................................................... 14
frexfp4 .................................................. 14
hypotd2 .................................................. 14
hypotf4 .................................................. 15
ilogbd2 .................................................. 15
ilogbf4 .................................................. 15
irintf4 .................................................. 15
iroundf4 ................................................ 15
is0denormd2 ............................................ 16
is0denormf4 ............................................ 16
isequald2 ................................................ 16
isequalf4 ................................................ 16
isfinite2 ................................................ 16
isfinite4 ................................................ 16
isgreaterd2 ............................................. 17
isgreatequald2 ........................................ 17
isgreatequalf4 ........................................ 17
isgreaterf4 ............................................. 17
isinf2 .................................................... 18
isinf4 .................................................... 18
islessd2 ................................................ 18
islesseqf2 .............................................. 18
islesseqf4 .............................................. 18
islessf4 ................................................ 18
islesseqd2 .............................................. 19
islesseqf4 .............................................. 19
isnormald2 ............................................. 19
isnormalf4 ............................................. 19
isunordered2 ........................................... 20
isunorderedf4 .......................................... 20
ldexp2 ................................................... 20
ldexpf4 .................................................. 20
lgamma2 .................................................. 20
lgamma4 ................................................ 20
llabs2 .................................................... 20
lldiv2 .................................................... 21
lldivu2 .................................................. 21
llintd2 ................................................... 21
llintf4 ................................................... 21
llround2 ................................................ 21
llroundf4 ............................................... 21
log10d2 ................................................ 22
log10f4 ................................................ 22
log1pd2 ................................................ 22
log1pf4 ................................................ 22
log2d2 ................................................... 22
log2f4 ................................................... 23
logbd2 .................................................. 23
logbf4 .................................................. 23
logd2 .................................................... 21
logf4 .................................................... 22
modfd2 .................................................. 23
modff4 .................................................. 23
nearbyintd2 .......................................... 24
nearbyintf4 .......................................... 24
negated2 ............................................... 24
negatef4 ............................................... 24
negatei4 ............................................... 24
negatei2 ............................................... 24
nextafter2 ............................................. 24

SIMD Math Library Specification for Cell Broadband Engine Architecture, Version 1.0
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextafterf4</td>
<td>24</td>
</tr>
<tr>
<td>powd2</td>
<td>24</td>
</tr>
<tr>
<td>powf4</td>
<td>25</td>
</tr>
<tr>
<td>recipd2</td>
<td>25</td>
</tr>
<tr>
<td>recipf4</td>
<td>25</td>
</tr>
<tr>
<td>remainderd2</td>
<td>25</td>
</tr>
<tr>
<td>remainderf4</td>
<td>25</td>
</tr>
<tr>
<td>remquod2</td>
<td>26</td>
</tr>
<tr>
<td>remquof4</td>
<td>26</td>
</tr>
<tr>
<td>rintd2</td>
<td>26</td>
</tr>
<tr>
<td>rintf4</td>
<td>26</td>
</tr>
<tr>
<td>roundd2</td>
<td>26</td>
</tr>
<tr>
<td>roundf4</td>
<td>26</td>
</tr>
<tr>
<td>rsqrd2</td>
<td>26</td>
</tr>
<tr>
<td>rsqrf4</td>
<td>27</td>
</tr>
<tr>
<td>scalblnd2</td>
<td>27</td>
</tr>
<tr>
<td>scalbnf4</td>
<td>27</td>
</tr>
<tr>
<td>signbtd2</td>
<td>27</td>
</tr>
<tr>
<td>signbitf4</td>
<td>27</td>
</tr>
<tr>
<td>sincosd2</td>
<td>27</td>
</tr>
<tr>
<td>sincosf4</td>
<td>27</td>
</tr>
<tr>
<td>sind2</td>
<td>28</td>
</tr>
<tr>
<td>sinf4</td>
<td>28</td>
</tr>
<tr>
<td>sinh2</td>
<td>28</td>
</tr>
<tr>
<td>sinhf4</td>
<td>28</td>
</tr>
<tr>
<td>sqrt2</td>
<td>28</td>
</tr>
<tr>
<td>sqrtf4</td>
<td>28</td>
</tr>
<tr>
<td>tand2</td>
<td>29</td>
</tr>
<tr>
<td>tany4</td>
<td>29</td>
</tr>
<tr>
<td>tanhd2</td>
<td>29</td>
</tr>
<tr>
<td>tanhf4</td>
<td>29</td>
</tr>
<tr>
<td>tgammad2</td>
<td>29</td>
</tr>
<tr>
<td>tgammaf4</td>
<td>29</td>
</tr>
<tr>
<td>truncd2</td>
<td>29</td>
</tr>
<tr>
<td>truncf4</td>
<td>29</td>
</tr>
</tbody>
</table>

SIMD Function Specifications, Type Definitions

<table>
<thead>
<tr>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>divi4_t</td>
<td>6</td>
</tr>
<tr>
<td>divu4_t</td>
<td>6</td>
</tr>
<tr>
<td>lldivi2_t</td>
<td>6</td>
</tr>
<tr>
<td>lldivu2_t</td>
<td>6</td>
</tr>
<tr>
<td>llroundf4_t</td>
<td>6</td>
</tr>
</tbody>
</table>