



# CUDA MATH API

v7.0 | March 2015

## API Reference Manual



# TABLE OF CONTENTS

<b>Chapter 1. Modules.....</b>	<b>1</b>
1.1. Mathematical Functions.....	1
1.2. Single Precision Mathematical Functions.....	2
acosf.....	2
acoshf.....	2
asinf.....	3
asinhf.....	3
atan2f.....	3
atanf.....	4
atanhf.....	4
cbrtf.....	5
ceilf.....	5
copysignf.....	5
cosf.....	6
coshf.....	6
cospif.....	6
cyl_bessel_i0f.....	7
cyl_bessel_i1f.....	7
erfcf.....	8
erfcinvf.....	8
erfcxf.....	8
erff.....	9
erfinvf.....	9
exp10f.....	10
exp2f.....	10
expf.....	10
expm1f.....	11
fabsf.....	11
fdimf.....	12
fdividef.....	12
floorf.....	13
fmaf.....	13
fmaxf.....	14
fminf.....	14
fmodf.....	15
frexpf.....	15
hypotf.....	16
ilogbf.....	16
isfinite.....	17
isinf.....	17

isnan.....	17
j0f.....	18
j1f.....	18
jnf.....	19
ldexpf.....	19
lgammaf.....	19
llrintf.....	20
llroundf.....	20
log10f.....	21
log1pf.....	21
log2f.....	21
logbf.....	22
logf.....	22
lrintf.....	23
lroundf.....	23
modff.....	23
nanf.....	24
nearbyintf.....	24
nextafterf.....	24
norm3df.....	25
norm4df.....	25
normcdf.....	26
normcdfinvf.....	26
powf.....	27
rcbrtf.....	27
remainderf.....	28
remquof.....	28
rhypotf.....	29
rintf.....	29
rnorm3df.....	30
roundf.....	30
rsqrtf.....	30
scalblnf.....	31
scalbnf.....	31
signbit.....	32
sincosf.....	32
sincospif.....	33
sinf.....	33
sinhf.....	34
sinpif.....	34
sqrtf.....	34
tanf.....	35
tanhf.....	35

tgammaf.....	36
truncf.....	36
y0f.....	36
y1f.....	37
ynf.....	37
1.3. Double Precision Mathematical Functions.....	38
acos.....	38
acosh.....	38
asin.....	39
asinh.....	39
atan.....	40
atan2.....	40
atanh.....	40
cbrt.....	41
ceil.....	41
copysign.....	42
cos.....	42
cosh.....	42
cospi.....	43
cyl_bessel_i0.....	43
cyl_bessel_i1.....	43
erf.....	44
erfc.....	44
erfcinv.....	45
erfcx.....	45
erfinv.....	45
exp.....	46
exp10.....	46
exp2.....	47
expm1.....	47
fabs.....	47
fdim.....	48
floor.....	48
fma.....	49
fmax.....	49
fmin.....	50
fmod.....	50
frexp.....	51
hypot.....	51
ilogb.....	52
isfinite.....	52
isinf.....	52
isnan.....	53

j0.....	53
j1.....	53
jn.....	54
ldexp.....	54
lgamma.....	55
llrint.....	55
llround.....	56
log.....	56
log10.....	56
log1p.....	57
log2.....	57
logb.....	58
lrint.....	58
lround.....	58
modf.....	59
nan.....	59
nearbyint.....	59
nextafter.....	60
norm3d.....	60
norm4d.....	61
normcdf.....	61
normcdfinv.....	61
pow.....	62
rcbrt.....	63
remainder.....	63
remquo.....	64
rhypot.....	64
rint.....	65
rnorm3d.....	65
round.....	65
rsqrt.....	66
scalbln.....	66
scalbn.....	67
signbit.....	67
sin.....	67
sincos.....	68
sincospi.....	68
sinh.....	69
sinpi.....	69
sqrt.....	69
tan.....	70
tanh.....	70
tgamma.....	71

trunc.....	71
y0.....	71
y1.....	72
yn.....	72
1.4. Single Precision Intrinsics.....	73
__cosf.....	73
__exp10f.....	73
__expf.....	74
__fadd_rd.....	74
__fadd_rn.....	75
__fadd_ru.....	75
__fadd_rz.....	75
__fdiv_rd.....	76
__fdiv_rn.....	76
__fdiv_ru.....	77
__fdiv_rz.....	77
__fdividef.....	77
__fmaf_rd.....	78
__fmaf_rn.....	78
__fmaf_ru.....	79
__fmaf_rz.....	79
__fmul_rd.....	80
__fmul_rn.....	80
__fmul_ru.....	81
__fmul_rz.....	81
__frcp_rd.....	81
__frcp_rn.....	82
__frcp_ru.....	82
__frcp_rz.....	83
__frsqrt_rn.....	83
__fsqrt_rd.....	83
__fsqrt_rn.....	84
__fsqrt_ru.....	84
__fsqrt_rz.....	84
__fsub_rd.....	85
__fsub_rn.....	85
__fsub_ru.....	86
__fsub_rz.....	86
__log10f.....	86
__log2f.....	87
__logf.....	87
__powf.....	88
__saturatef.....	88

__sincosf.....	88
__sinf.....	89
__tanf.....	89
1.5. Double Precision Intrinsics.....	90
__ddiv_rd.....	90
__ddiv_rn.....	90
__ddiv_ru.....	90
__ddiv_rz.....	91
__drcp_rd.....	91
__drcp_rn.....	92
__drcp_ru.....	92
__drcp_rz.....	92
__dsqrt_rd.....	93
__dsqrt_rn.....	93
__dsqrt_ru.....	94
__dsqrt_rz.....	94
1.6. Integer Intrinsics.....	94
__brev.....	95
__brevll.....	95
__byte_perm.....	95
__clz.....	96
__clzll.....	96
__ffs.....	96
__ffsll.....	97
__hadd.....	97
__mul24.....	97
__mul64hi.....	98
__mulhi.....	98
__popc.....	98
__popcll.....	98
__rhadd.....	99
__sad.....	99
__uhadd.....	99
__umul24.....	100
__umul64hi.....	100
__umulhi.....	100
__urhadd.....	101
__usad.....	101
1.7. Type Casting Intrinsics.....	101
__double2int_rz.....	102
__double2ll_rz.....	102
__double2uint_rz.....	102
__double2ull_rz.....	103

__float2half_rn.....	103
__float2int_rd.....	103
__float2int_rn.....	104
__float2int_ru.....	104
__float2int_rz.....	104
__float2ll_rd.....	105
__float2ll_rn.....	105
__float2ll_ru.....	105
__float2ll_rz.....	106
__float2uint_rd.....	106
__float2uint_rn.....	106
__float2uint_ru.....	107
__float2uint_rz.....	107
__float2ull_rd.....	107
__float2ull_rn.....	108
__float2ull_ru.....	108
__float2ull_rz.....	108
__float_as_int.....	109
__half2float.....	109
__int2float_rd.....	109
__int2float_rn.....	110
__int2float_ru.....	110
__int2float_rz.....	110
__int_as_float.....	111
__ll2float_rd.....	111
__ll2float_rn.....	111
__ll2float_ru.....	112
__ll2float_rz.....	112
__uint2float_rd.....	112
__uint2float_rn.....	113
__uint2float_ru.....	113
__uint2float_rz.....	113
__ull2float_rd.....	114
__ull2float_rn.....	114
__ull2float_ru.....	114
__ull2float_rz.....	115
1.8. SIMD Intrinsics.....	115
__vabs2.....	115
__vabs4.....	115
__vabsdiffs2.....	116
__vabsdiffs4.....	116
__vabsdiffu2.....	116
__vabsdiffu4.....	117



__vabsss2.....	117
__vabsss4.....	117
__vadd2.....	118
__vadd4.....	118
__vaddss2.....	118
__vaddss4.....	119
__vaddus2.....	119
__vaddus4.....	119
__vavg2.....	120
__vavg4.....	120
__vavg2.....	120
__vavg4.....	121
__vcmp2.....	121
__vcmp4.....	121
__vcmp2.....	122
__vcmp4.....	122
__vcmp2.....	122
__vcmp4.....	123
__vcmp2.....	123
__vcmp4.....	123
__vcmp2.....	124
__vcmp4.....	124
__vcmp2.....	124
__vcmp4.....	125
__vcmp2.....	125
__vcmp4.....	125
__vcmp2.....	126
__vcmp4.....	126
__vcmp2.....	126
__vcmp4.....	127
__vcmp2.....	127
__vcmp4.....	127
__vhadd2.....	128
__vhadd4.....	128
__vmax2.....	128
__vmax4.....	129
__vmax2.....	129
__vmax4.....	129
__vmin2.....	130
__vmin4.....	130
__vmin2.....	130
__vmin4.....	131
__vneg2.....	131

__vneg4.....	131
__vnegss2.....	132
__vnegss4.....	132
__vsads2.....	132
__vsads4.....	133
__vsadu2.....	133
__vsadu4.....	133
__vseteq2.....	134
__vseteq4.....	134
__vsetges2.....	134
__vsetges4.....	135
__vsetgeu2.....	135
__vsetgeu4.....	135
__vsetgts2.....	136
__vsetgts4.....	136
__vsetgtu2.....	136
__vsetgtu4.....	137
__vsetles2.....	137
__vsetles4.....	137
__vsetleu2.....	138
__vsetleu4.....	138
__vsetlts2.....	138
__vsetlts4.....	139
__vsetltu2.....	139
__vsetltu4.....	139
__vsetne2.....	140
__vsetne4.....	140
__vsub2.....	140
__vsub4.....	141
__vsubss2.....	141
__vsubss4.....	141
__vsubus2.....	142
__vsubus4.....	142
1.9. Half Precision Intrinsics.....	142
Half Arithmetic Functions.....	142
Half2 Arithmetic Functions.....	142
Half Comparison Functions.....	142
Half2 Comparison Functions.....	142
Half Precision Conversion And Data Movement.....	143
1.9.1. Half Arithmetic Functions.....	143
__hadd.....	143
__hadd_sat.....	143
__hfma.....	143

__hfma_sat.....	144
__hmul.....	144
__hmul_sat.....	144
__hneg.....	145
__hsub.....	145
__hsub_sat.....	145
1.9.2. Half2 Arithmetic Functions.....	145
__hadd2.....	146
__hadd2_sat.....	146
__hfma2.....	146
__hfma2_sat.....	147
__hmul2.....	147
__hmul2_sat.....	147
__hneg2.....	148
__hsub2.....	148
__hsub2_sat.....	148
1.9.3. Half Comparison Functions.....	148
__heq.....	149
__hequ.....	149
__hge.....	149
__hgeu.....	149
__hgt.....	150
__hgtu.....	150
__hisinf.....	150
__hisnan.....	150
__hle.....	151
__hleu.....	151
__hlt.....	151
__hltu.....	151
__hne.....	152
__hneu.....	152
1.9.4. Half2 Comparison Functions.....	152
__hbeq2.....	152
__hbequ2.....	153
__hbge2.....	153
__hbgeu2.....	154
__hbg2.....	154
__hbg2u.....	154
__hble2.....	155
__hbleu2.....	155
__hblt2.....	156
__hbltu2.....	156
__hbne2.....	156

__hbneu2.....	157
__heq2.....	157
__hequ2.....	157
__hge2.....	158
__hgeu2.....	158
__hgt2.....	158
__hgtu2.....	159
__hisnan2.....	159
__hle2.....	159
__hleu2.....	160
__hlt2.....	160
__hltu2.....	160
__hne2.....	161
__hneu2.....	161
1.9.5. Half Precision Conversion And Data Movement.....	161
__float22half2_rn.....	161
__float2half2_rn.....	162
__floats2half2_rn.....	162
__half22float2.....	162
__half2half2.....	162
__halves2half2.....	163
__high2float.....	163
__high2half.....	163
__high2half2.....	163
__highs2half2.....	164
__low2float.....	164
__low2half.....	164
__low2half2.....	165
__lowhigh2highlow.....	165
__lows2half2.....	165

# Chapter 1.

## MODULES

Here is a list of all modules:

- ▶ Mathematical Functions
- ▶ Single Precision Mathematical Functions
- ▶ Double Precision Mathematical Functions
- ▶ Single Precision Intrinsic
- ▶ Double Precision Intrinsic
- ▶ Integer Intrinsic
- ▶ Type Casting Intrinsic
- ▶ SIMD Intrinsic
- ▶ Half Precision Intrinsic
  - ▶ Half Arithmetic Functions
  - ▶ Half2 Arithmetic Functions
  - ▶ Half Comparison Functions
  - ▶ Half2 Comparison Functions
  - ▶ Half Precision Conversion And Data Movement

## 1.1. Mathematical Functions

CUDA mathematical functions are always available in device code. Some functions are also available in host code as indicated.

Note that floating-point functions are overloaded for different argument types. For example, the `log()` function has the following prototypes:

```
↑ double log(double x);  
   float log(float x);  
   float logf(float x);
```

## 1.2. Single Precision Mathematical Functions

This section describes single precision mathematical functions.

**`__host__ __device__ float acosf (float x)`**

Calculate the arc cosine of the input argument.

### Returns

Result will be in radians, in the interval  $[0, \pi]$  for  $x$  inside  $[-1, +1]$ .

- ▶ `acosf(1)` returns  $+0$ .
- ▶ `acosf(x)` returns NaN for  $x$  outside  $[-1, +1]$ .

### Description

Calculate the principal value of the arc cosine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**`__host__ __device__ float acoshf (float x)`**

Calculate the nonnegative arc hyperbolic cosine of the input argument.

### Returns

Result will be in the interval  $[0, +\infty]$ .

- ▶ `acoshf(1)` returns  $0$ .
- ▶ `acoshf(x)` returns NaN for  $x$  in the interval  $[-\infty, 1)$ .

### Description

Calculate the nonnegative arc hyperbolic cosine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float asinf (float x)`

Calculate the arc sine of the input argument.

### Returns

Result will be in radians, in the interval  $[-\pi/2, +\pi/2]$  for  $x$  inside  $[-1, +1]$ .

- ▶ `asinf(0)` returns +0.
- ▶ `asinf(x)` returns NaN for  $x$  outside  $[-1, +1]$ .

### Description

Calculate the principal value of the arc sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float asinhf (float x)`

Calculate the arc hyperbolic sine of the input argument.

### Returns

- ▶ `asinhf(0)` returns 1.

### Description

Calculate the arc hyperbolic sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float atan2f (float y, float x)`

Calculate the arc tangent of the ratio of first and second input arguments.

### Returns

Result will be in radians, in the interval  $[-\pi, +\pi]$ .

- ▶ `atan2f(0, 1)` returns +0.

**Description**

Calculate the principal value of the arc tangent of the ratio of first and second input arguments  $y / x$ . The quadrant of the result is determined by the signs of inputs  $y$  and  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float atanf (float x)`

Calculate the arc tangent of the input argument.

**Returns**

Result will be in radians, in the interval  $[-\pi/2, +\pi/2]$ .

- ▶ `atanf(0)` returns `+0`.

**Description**

Calculate the principal value of the arc tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float atanhf (float x)`

Calculate the arc hyperbolic tangent of the input argument.

**Returns**

- ▶ `atanhf(  $\pm 0$  )` returns  $\pm 0$ .
- ▶ `atanhf(  $\pm 1$  )` returns  $\pm \infty$ .
- ▶ `atanhf(x)` returns NaN for  $x$  outside interval  $[-1, 1]$ .

**Description**

Calculate the arc hyperbolic tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.



## `__host__ __device__ float cbrtf (float x)`

Calculate the cube root of the input argument.

### Returns

Returns  $x^{1/3}$ .

- ▶ `cbrtf( ±0 )` returns  $\pm 0$ .
- ▶ `cbrtf( ±∞ )` returns  $\pm \infty$ .

### Description

Calculate the cube root of  $x$ ,  $x^{1/3}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float ceilf (float x)`

Calculate ceiling of the input argument.

### Returns

Returns  $\lceil x \rceil$  expressed as a floating-point number.

- ▶ `ceilf( ±0 )` returns  $\pm 0$ .
- ▶ `ceilf( ±∞ )` returns  $\pm \infty$ .

### Description

Compute the smallest integer value not less than  $x$ .

## `__host__ __device__ float copysignf (float x, float y)`

Create value with given magnitude, copying sign of second value.

### Returns

Returns a value with the magnitude of  $x$  and the sign of  $y$ .

### Description

Create a floating-point value with the magnitude  $x$  and the sign of  $y$ .

## `__host__ __device__ float cosf (float x)`

Calculate the cosine of the input argument.

### Returns

- ▶ `cosf(0)` returns 1.
- ▶ `cosf(  $\pm \infty$  )` returns NaN.

### Description

Calculate the cosine of the input argument  $x$  (measured in radians).



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

## `__host__ __device__ float coshf (float x)`

Calculate the hyperbolic cosine of the input argument.

### Returns

- ▶ `coshf(0)` returns 1.
- ▶ `coshf(  $\pm \infty$  )` returns NaN.

### Description

Calculate the hyperbolic cosine of the input argument  $x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float cospif (float x)`

Calculate the cosine of the input argument  $\times \pi$ .

### Returns

- ▶ `cospif(  $\pm 0$  )` returns 1.
- ▶ `cospif(  $\pm \infty$  )` returns NaN.

**Description**

Calculate the cosine of  $x \times \pi$  (measured in radians), where  $x$  is the input argument.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float cyl_bessel_i0f (float x)`

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

**Returns**

Returns the value of the regular modified cylindrical Bessel function of order 0.

**Description**

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument  $x$ ,  $I_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float cyl_bessel_i1f (float x)`

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

**Returns**

Returns the value of the regular modified cylindrical Bessel function of order 1.

**Description**

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument  $x$ ,  $I_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float erfcf (float x)`

Calculate the complementary error function of the input argument.

### Returns

- ▶ `erfcf( -  $\infty$  )` returns 2.
- ▶ `erfcf( +  $\infty$  )` returns +0.

### Description

Calculate the complementary error function of the input argument  $x$ ,  $1 - \text{erf}(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float erfcinvf (float y)`

Calculate the inverse complementary error function of the input argument.

### Returns

- ▶ `erfcinvf(0)` returns +  $\infty$ .
- ▶ `erfcinvf(2)` returns -  $\infty$ .

### Description

Calculate the inverse complementary error function of the input argument  $y$ , for  $y$  in the interval  $[0, 2]$ . The inverse complementary error function find the value  $x$  that satisfies the equation  $y = \text{erfc}(x)$ , for  $0 \leq y \leq 2$ , and  $-\infty \leq x \leq \infty$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float erfcxf (float x)`

Calculate the scaled complementary error function of the input argument.

### Returns

- ▶ `erfcxf( -  $\infty$  )` returns +  $\infty$
- ▶ `erfcxf( +  $\infty$  )` returns +0
- ▶ `erfcxf( $x$ )` returns +  $\infty$  if the correctly calculated value is outside the single floating point range.

**Description**

Calculate the scaled complementary error function of the input argument  $x$ ,  $e^{x^2} \cdot \text{erfc}(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float erff (float x)`

Calculate the error function of the input argument.

**Returns**

- ▶ `erff( ± 0 )` returns  $\pm 0$ .
- ▶ `erff( ± ∞ )` returns  $\pm 1$ .

**Description**

Calculate the value of the error function for the input argument  $x$ ,  $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float erfinvf (float y)`

Calculate the inverse error function of the input argument.

**Returns**

- ▶ `erfinvf(1)` returns  $+\infty$ .
- ▶ `erfinvf(-1)` returns  $-\infty$ .

**Description**

Calculate the inverse error function of the input argument  $y$ , for  $y$  in the interval  $[-1, 1]$ . The inverse error function finds the value  $x$  that satisfies the equation  $y = \text{erf}(x)$ , for  $-1 \leq y \leq 1$ , and  $-\infty \leq x \leq \infty$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float exp10f (float x)`

Calculate the base 10 exponential of the input argument.

### Returns

Returns  $10^x$ .

### Description

Calculate the base 10 exponential of the input argument  $x$ .



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

## `__host__ __device__ float exp2f (float x)`

Calculate the base 2 exponential of the input argument.

### Returns

Returns  $2^x$ .

### Description

Calculate the base 2 exponential of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float expf (float x)`

Calculate the base  $e$  exponential of the input argument.

### Returns

Returns  $e^x$ .

### Description

Calculate the base  $e$  exponential of the input argument  $x$ ,  $e^x$ .



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

## `__host__ __device__ float expm1f (float x)`

Calculate the base  $e$  exponential of the input argument, minus 1.

### Returns

Returns  $e^x - 1$ .

### Description

Calculate the base  $e$  exponential of the input argument  $x$ , minus 1.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fabsf (float x)`

Calculate the absolute value of its argument.

### Returns

Returns the absolute value of its argument.

- `fabs(  $\pm \infty$  )` returns  $+\infty$ .
- `fabs(  $\pm 0$  )` returns 0.

### Description

Calculate the absolute value of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fdimf (float x, float y)`

Compute the positive difference between  $x$  and  $y$ .

### Returns

Returns the positive difference between  $x$  and  $y$ .

- ▶ `fdimf(x, y)` returns  $x - y$  if  $x > y$ .
- ▶ `fdimf(x, y)` returns  $+0$  if  $x \leq y$ .

### Description

Compute the positive difference between  $x$  and  $y$ . The positive difference is  $x - y$  when  $x > y$  and  $+0$  otherwise.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__DEVICE_FUNCTIONS_DECL__ float fdividef (float x, float y)`

Divide two floating point values.

### Returns

Returns  $x / y$ .

### Description

Compute  $x$  divided by  $y$ . If `--use_fast_math` is specified, use `__fdividef()` for higher performance, otherwise use normal division.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.



## `__host__ __device__ float floorf (float x)`

Calculate the largest integer less than or equal to  $x$ .

### Returns

Returns  $\log_e(1+x)$  expressed as a floating-point number.

- ▶ `floorf(  $\pm \infty$  )` returns  $\pm \infty$ .
- ▶ `floorf(  $\pm 0$  )` returns  $\pm 0$ .

### Description

Calculate the largest integer value which is less than or equal to  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fmaf (float x, float y, float z)`

Compute  $x \times y + z$  as a single operation.

### Returns

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fmaf(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fmaf(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fmaf( $x$ ,  $y$ ,  $-\infty$ )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fmaf( $x$ ,  $y$ ,  $+\infty$ )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

### Description

Compute the value of  $x \times y + z$  as a single ternary operation. After computing the value to infinite precision, the value is rounded once.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fmaxf (float x, float y)`

Determine the maximum numeric value of the arguments.

### Returns

Returns the maximum numeric values of the arguments  $x$  and  $y$ .

- ▶ If both arguments are NaN, returns NaN.
- ▶ If one argument is NaN, returns the numeric argument.

### Description

Determines the maximum numeric value of the arguments  $x$  and  $y$ . Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fminf (float x, float y)`

Determine the minimum numeric value of the arguments.

### Returns

Returns the minimum numeric values of the arguments  $x$  and  $y$ .

- ▶ If both arguments are NaN, returns NaN.
- ▶ If one argument is NaN, returns the numeric argument.

### Description

Determines the minimum numeric value of the arguments  $x$  and  $y$ . Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float fmodf (float x, float y)`

Calculate the floating-point remainder of  $x / y$ .

### Returns

- ▶ Returns the floating point remainder of  $x / y$ .
- ▶ `fmodf(  $\pm 0$ , y )` returns  $\pm 0$  if  $y$  is not zero.
- ▶ `fmodf(x, y)` returns NaN and raised an invalid floating point exception if  $x$  is  $\pm \infty$  or  $y$  is zero.
- ▶ `fmodf(x, y)` returns zero if  $y$  is zero or the result would overflow.
- ▶ `fmodf(x,  $\pm \infty$ )` returns  $x$  if  $x$  is finite.
- ▶ `fmodf(x, 0)` returns NaN.

### Description

Calculate the floating-point remainder of  $x / y$ . The absolute value of the computed value is always less than  $y$ 's absolute value and will have the same sign as  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float frexpf (float x, int *nptr)`

Extract mantissa and exponent of a floating-point value.

### Returns

Returns the fractional component  $m$ .

- ▶ `frexpf(0, nptr)` returns 0 for the fractional component and zero for the integer component.
- ▶ `frexpf(  $\pm 0$ , nptr )` returns  $\pm 0$  and stores zero in the location pointed to by `nptr`.
- ▶ `frexpf(  $\pm \infty$ , nptr )` returns  $\pm \infty$  and stores an unspecified value in the location to which `nptr` points.
- ▶ `frexpf(NaN, y)` returns a NaN and stores an unspecified value in the location to which `nptr` points.

### Description

Decomposes the floating-point value  $x$  into a component  $m$  for the normalized fraction element and another term  $n$  for the exponent. The absolute value of  $m$  will be greater than or equal to 0.5 and less than 1.0 or it will be equal to 0;  $x = m \cdot 2^n$ . The integer exponent  $n$  will be stored in the location to which `nptr` points.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float __CRTDECL hypotf (float x, float y)`

Calculate the square root of the sum of squares of two arguments.

### Returns

Returns the length of the hypotenuse  $\sqrt{x^2 + y^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

### Description

Calculates the length of the hypotenuse of a right triangle whose two sides have lengths  $x$  and  $y$  without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ int ilogbf (float x)`

Compute the unbiased integer exponent of the argument.

### Returns

- ▶ If successful, returns the unbiased exponent of the argument.
- ▶ `ilogbf(0)` returns `INT_MIN`.
- ▶ `ilogbf(NaN)` returns NaN.
- ▶ `ilogbf(x)` returns `INT_MAX` if  $x$  is  $\infty$  or the correct value is greater than `INT_MAX`.
- ▶ `ilogbf(x)` return `INT_MIN` if the correct value is less than `INT_MIN`.

### Description

Calculates the unbiased integer exponent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ __RETURN_TYPE isfinite (float a)`

Determine whether argument is finite.

### Returns

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is a finite value.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is a finite value.

### Description

Determine whether the floating-point value `a` is a finite value (zero, subnormal, or normal and not infinity or NaN).

## `__host__ __device__ __RETURN_TYPE isinf (float a)`

Determine whether argument is infinite.

### Returns

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is a infinite value.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is a infinite value.

### Description

Determine whether the floating-point value `a` is an infinite value (positive or negative).

## `__host__ __device__ __RETURN_TYPE isnan (float a)`

Determine whether argument is a NaN.

### Returns

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is a NaN value.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is a NaN value.

### Description

Determine whether the floating-point value `a` is a NaN.

## `__host__ __device__ float j0f (float x)`

Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

### Returns

Returns the value of the Bessel function of the first kind of order 0.

- ▶ `j0f( ±∞ )` returns +0.
- ▶ `j0f(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the first kind of order 0 for the input argument  $x$ ,  $J_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float j1f (float x)`

Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

### Returns

Returns the value of the Bessel function of the first kind of order 1.

- ▶ `j1f( ±0 )` returns  $\pm 0$ .
- ▶ `j1f( ±∞ )` returns +0.
- ▶ `j1f(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the first kind of order 1 for the input argument  $x$ ,  $J_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float jnf (int n, float x)`

Calculate the value of the Bessel function of the first kind of order  $n$  for the input argument.

### Returns

Returns the value of the Bessel function of the first kind of order  $n$ .

- ▶ `jnf(n, NaN)` returns NaN.
- ▶ `jnf(n, x)` returns NaN for  $n < 0$ .
- ▶ `jnf(n,  $+\infty$ )` returns +0.

### Description

Calculate the value of the Bessel function of the first kind of order  $n$  for the input argument  $x$ ,  $J_n(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float ldexpf (float x, int exp)`

Calculate the value of  $x \cdot 2^{exp}$ .

### Returns

- ▶ `ldexpf(x)` returns  $\pm \infty$  if the correctly calculated value is outside the single floating point range.

### Description

Calculate the value of  $x \cdot 2^{exp}$  of the input arguments  $x$  and  $exp$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float lgammaf (float x)`

Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

### Returns

- ▶ `lgammaf(1)` returns +0.

- ▶ `lgammaf(2)` returns `+0`.
- ▶ `lgammaf(x)` returns  $\pm \infty$  if the correctly calculated value is outside the single floating point range.
- ▶ `lgammaf(x)` returns  $+\infty$  if  $x \leq 0$ .
- ▶ `lgammaf(-\infty)` returns  $-\infty$ .
- ▶ `lgammaf(+\infty)` returns  $+\infty$ .

### Description

Calculate the natural logarithm of the absolute value of the gamma function of the input argument  $x$ , namely the value of  $\log_e \left| \int_0^\infty e^{-t} t^{x-1} dt \right|$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ long long int llrintf (float x)`

Round input to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded towards zero. If the result is outside the range of the return type, the result is undefined.

## `__host__ __device__ long long int llroundf (float x)`

Round to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the result is undefined.



This function may be slower than alternate rounding methods. See `llrintf()`.



## \_\_host\_\_ \_\_device\_\_ float log10f (float x)

Calculate the base 10 logarithm of the input argument.

### Returns

- ▶  $\log_{10}f(\pm 0)$  returns  $-\infty$ .
- ▶  $\log_{10}f(1)$  returns +0.
- ▶  $\log_{10}f(x)$  returns NaN for  $x < 0$ .
- ▶  $\log_{10}f(+\infty)$  returns  $+\infty$ .

### Description

Calculate the base 10 logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float log1pf (float x)

Calculate the value of  $\log_e(1+x)$ .

### Returns

- ▶  $\log_{1p}f(\pm 0)$  returns  $-\infty$ .
- ▶  $\log_{1p}f(-1)$  returns +0.
- ▶  $\log_{1p}f(x)$  returns NaN for  $x < -1$ .
- ▶  $\log_{1p}f(+\infty)$  returns  $+\infty$ .

### Description

Calculate the value of  $\log_e(1+x)$  of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float log2f (float x)

Calculate the base 2 logarithm of the input argument.

### Returns

- ▶  $\log_2f(\pm 0)$  returns  $-\infty$ .
- ▶  $\log_2f(1)$  returns +0.

- ▶  $\log_2 f(x)$  returns NaN for  $x < 0$ .
- ▶  $\log_2 f(+\infty)$  returns  $+\infty$ .

### Description

Calculate the base 2 logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float logbf (float x)

Calculate the floating point representation of the exponent of the input argument.

### Returns

- ▶  $\logbf(\pm 0)$  returns  $-\infty$
- ▶  $\logbf(+\infty)$  returns  $+\infty$

### Description

Calculate the floating point representation of the exponent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float logf (float x)

Calculate the natural logarithm of the input argument.

### Returns

- ▶  $\logf(\pm 0)$  returns  $-\infty$ .
- ▶  $\logf(1)$  returns +0.
- ▶  $\logf(x)$  returns NaN for  $x < 0$ .
- ▶  $\logf(+\infty)$  returns  $+\infty$ .

### Description

Calculate the natural logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ long int lrintf (float x)`

Round input to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded towards zero. If the result is outside the range of the return type, the result is undefined.

## `__host__ __device__ long int lroundf (float x)`

Round to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the result is undefined.



This function may be slower than alternate rounding methods. See [lrintf\(\)](#).

## `__host__ __device__ float modff (float x, float *iptr)`

Break down the input argument into fractional and integral parts.

### Returns

- ▶ `modff(  $\pm x$ , iptr)` returns a result with the same sign as  $x$ .
- ▶ `modff(  $\pm \infty$ , iptr)` returns  $\pm 0$  and stores  $\pm \infty$  in the object pointed to by `iptr`.
- ▶ `modff(NaN, iptr)` stores a NaN in the object pointed to by `iptr` and returns a NaN.

### Description

Break down the argument  $x$  into fractional and integral parts. The integral part is stored in the argument `iptr`. Fractional and integral parts are given the same sign as the argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float nanf (const char *tagp)`

Returns "Not a Number" value.

### Returns

- `nanf(tagp)` returns NaN.

### Description

Return a representation of a quiet NaN. Argument `tagp` selects one of the possible representations.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float nearbyintf (float x)`

Round the input argument to the nearest integer.

### Returns

- `nearbyintf( ± 0 )` returns  $\pm 0$ .
- `nearbyintf( ± ∞ )` returns  $\pm \infty$ .

### Description

Round argument  $x$  to an integer value in single precision floating-point format.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float nextafterf (float x, float y)`

Return next representable single-precision floating-point value after argument.

### Returns

- `nextafterf( ± ∞ , y )` returns  $\pm \infty$ .

### Description

Calculate the next representable single-precision floating-point value following  $x$  in the direction of  $y$ . For example, if  $y$  is greater than  $x$ , `nextafterf()` returns the smallest representable number greater than  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float norm3df (float a, float b, float c)`

Calculate the square root of the sum of squares of three coordinates of the argument.

### Returns

Returns the length of the 3D  $\sqrt{p.x^2 + p.y^2 + p.z^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

### Description

Calculates the length of three dimensional vector  $p$  in euclidean space without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float norm4df (float a, float b, float c, float d)`

Calculate the square root of the sum of squares of four coordinates of the argument.

### Returns

Returns the length of the 4D vector  $\sqrt{p.x^2 + p.y^2 + p.z^2 + p.t^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

### Description

Calculates the length of four dimensional vector  $p$  in euclidean space without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float normcdf (float y)`

Calculate the standard normal cumulative distribution function.

### Returns

- ▶ `normcdf(+∞)` returns 1
- ▶ `normcdf(-∞)` returns +0

### Description

Calculate the cumulative distribution function of the standard normal distribution for input argument  $y$ ,  $\Phi(y)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float normcdfinvf (float y)`

Calculate the inverse of the standard normal cumulative distribution function.

### Returns

- ▶ `normcdfinvf(0)` returns  $-\infty$ .
- ▶ `normcdfinvf(1)` returns  $+\infty$ .
- ▶ `normcdfinvf(x)` returns NaN if  $x$  is not in the interval  $[0,1]$ .

### Description

Calculate the inverse of the standard normal cumulative distribution function for input argument  $y$ ,  $\Phi^{-1}(y)$ . The function is defined for input values in the interval  $(0, 1)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float powf (float x, float y)`

Calculate the value of first argument to the power of second argument.

### Returns

- ▶ `powf(  $\pm 0$ ,  $y$  )` returns  $\pm \infty$  for  $y$  an integer less than 0.
- ▶ `powf(  $\pm 0$ ,  $y$  )` returns  $\pm 0$  for  $y$  an odd integer greater than 0.
- ▶ `powf(  $\pm 0$ ,  $y$  )` returns +0 for  $y > 0$  and not an odd integer.
- ▶ `powf(-1,  $\pm \infty$  )` returns 1.
- ▶ `powf(+1,  $y$  )` returns 1 for any  $y$ , even a NaN.
- ▶ `powf( $x$ ,  $\pm 0$  )` returns 1 for any  $x$ , even a NaN.
- ▶ `powf( $x$ ,  $y$  )` returns a NaN for finite  $x < 0$  and finite non-integer  $y$ .
- ▶ `powf( $x$ ,  $-\infty$  )` returns  $+\infty$  for  $|x| < 1$ .
- ▶ `powf( $x$ ,  $-\infty$  )` returns +0 for  $|x| > 1$ .
- ▶ `powf( $x$ ,  $+\infty$  )` returns +0 for  $|x| < 1$ .
- ▶ `powf( $x$ ,  $+\infty$  )` returns  $+\infty$  for  $|x| > 1$ .
- ▶ `powf(  $-\infty$ ,  $y$  )` returns -0 for  $y$  an odd integer less than 0.
- ▶ `powf(  $-\infty$ ,  $y$  )` returns +0 for  $y < 0$  and not an odd integer.
- ▶ `powf(  $-\infty$ ,  $y$  )` returns  $-\infty$  for  $y$  an odd integer greater than 0.
- ▶ `powf(  $-\infty$ ,  $y$  )` returns  $+\infty$  for  $y > 0$  and not an odd integer.
- ▶ `powf(  $+\infty$ ,  $y$  )` returns +0 for  $y < 0$ .
- ▶ `powf(  $+\infty$ ,  $y$  )` returns  $+\infty$  for  $y > 0$ .

### Description

Calculate the value of  $x$  to the power of  $y$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float rcbrtf (float x)`

Calculate reciprocal cube root function.

### Returns

- ▶ `rcbrt(  $\pm 0$  )` returns  $\pm \infty$ .
- ▶ `rcbrt(  $\pm \infty$  )` returns  $\pm 0$ .

### Description

Calculate reciprocal cube root function of  $x$



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float remainderf (float x, float y)`

Compute single-precision floating-point remainder.

### Returns

- ▶ `remainderf(x, 0)` returns NaN.
- ▶ `remainderf(±∞, y)` returns NaN.
- ▶ `remainderf(x, ±∞)` returns `x` for finite `x`.

### Description

Compute single-precision floating-point remainder  $r$  of dividing  $x$  by  $y$  for nonzero  $y$ . Thus  $r = x - ny$ . The value  $n$  is the integer value nearest  $\frac{x}{y}$ . In the case when  $|n - \frac{x}{y}| = \frac{1}{2}$ , the even  $n$  value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float remquof (float x, float y, int *quo)`

Compute single-precision floating-point remainder and part of quotient.

### Returns

Returns the remainder.

- ▶ `remquof(x, 0, quo)` returns NaN.
- ▶ `remquof(±∞, y, quo)` returns NaN.
- ▶ `remquof(x, ±∞, quo)` returns `x`.

### Description

Compute a double-precision floating-point remainder in the same way as the `remainderf()` function. Argument `quo` returns part of quotient upon division of `x` by `y`. Value `quo` has the same sign as  $\frac{x}{y}$  and may not be the exact quotient but agrees with the exact quotient in the low order 3 bits.





For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float rhypotf (float x, float y)`

Calculate one over the square root of the sum of squares of two arguments.

### Returns

Returns one over the length of the hypotenuse  $\frac{1}{\sqrt{x^2 + y^2}}$ . If the square root would overflow, returns 0. If the square root would underflow, returns  $+\infty$ .

### Description

Calculates one over the length of the hypotenuse of a right triangle whose two sides have lengths  $x$  and  $y$  without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float rintf (float x)`

Round input to nearest integer value in floating-point.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value in floating-point format, with halfway cases rounded towards zero.

## `__host__ __device__ float rnorm3df (float a, float b, float c)`

Calculate one over the square root of the sum of squares of three coordinates of the argument.

### Returns

Returns one over the length of the 3D vector  $\frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2}}$ . If the square root would overflow, returns 0. If the square root would underflow, returns  $+\infty$ .

### Description

Calculates one over the length of three dimension vector  $p$  in euclidean space without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float roundf (float x)`

Round to nearest integer value in floating-point.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value in floating-point format, with halfway cases rounded away from zero.



This function may be slower than alternate rounding methods. See `rintf()`.

## `__host__ __device__ float rsqrtf (float x)`

Calculate the reciprocal of the square root of the input argument.

### Returns

Returns  $1/\sqrt{x}$ .

- `rsqrtf(+∞)` returns `+0`.

- ▶ `rsqrtf( ±0 )` returns  $\pm \infty$ .
- ▶ `rsqrtf(x)` returns NaN if  $x$  is less than 0.

### Description

Calculate the reciprocal of the nonnegative square root of  $x$ ,  $1/\sqrt{x}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float scalblnf (float x, long int n)

Scale floating-point input by integer power of two.

### Returns

Returns  $x * 2^n$ .

- ▶ `scalblnf( ±0 , n )` returns  $\pm 0$ .
- ▶ `scalblnf(x, 0)` returns  $x$ .
- ▶ `scalblnf( ±∞ , n )` returns  $\pm \infty$ .

### Description

Scale  $x$  by  $2^n$  by efficient manipulation of the floating-point exponent.

## \_\_host\_\_ \_\_device\_\_ float scalbnf (float x, int n)

Scale floating-point input by integer power of two.

### Returns

Returns  $x * 2^n$ .

- ▶ `scalbnf( ±0 , n )` returns  $\pm 0$ .
- ▶ `scalbnf(x, 0)` returns  $x$ .
- ▶ `scalbnf( ±∞ , n )` returns  $\pm \infty$ .

### Description

Scale  $x$  by  $2^n$  by efficient manipulation of the floating-point exponent.

## `__host__ __device__ __RETURN_TYPE signbit (float a)`

Return the sign bit of the input.

### Returns

Reports the sign bit of all values including infinities, zeros, and NaNs.

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if `a` is negative.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if `a` is negative.

### Description

Determine whether the floating-point value `a` is negative.

## `__host__ __device__ void sincosf (float x, float *sptr, float *cptr)`

Calculate the sine and cosine of the first input argument.

### Returns

- ▶ none

### Description

Calculate the sine and cosine of the first input argument `x` (measured in radians). The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`.

### See also:

[`sinf\(\)`](#) and [`cosf\(\)`](#).



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

**\_\_host\_\_ \_\_device\_\_ void sincospif (float x, float \*sptr, float \*cptr)**

Calculate the sine and cosine of the first input argument  $\times \pi$ .

### Returns

- ▶ none

### Description

Calculate the sine and cosine of the first input argument,  $x$  (measured in radians),  $\times \pi$ . The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`.

### See also:

`sinpif()` and `cospif()`.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_host\_\_ \_\_device\_\_ float sinf (float x)**

Calculate the sine of the input argument.

### Returns

- ▶ `sinf(  $\pm 0$  )` returns  $\pm 0$ .
- ▶ `sinf(  $\pm \infty$  )` returns NaN.

### Description

Calculate the sine of the input argument  $x$  (measured in radians).



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

## `__host__ __device__ float sinh( float x )`

Calculate the hyperbolic sine of the input argument.

### Returns

- ▶ `sinh( ± 0 )` returns  $\pm 0$ .
- ▶ `sinh( ± ∞ )` returns NaN.

### Description

Calculate the hyperbolic sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float sinpi( float x )`

Calculate the sine of the input argument  $\times \pi$ .

### Returns

- ▶ `sinpi( ± 0 )` returns  $\pm 0$ .
- ▶ `sinpi( ± ∞ )` returns NaN.

### Description

Calculate the sine of  $x \times \pi$  (measured in radians), where  $x$  is the input argument.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float sqrt( float x )`

Calculate the square root of the input argument.

### Returns

Returns  $\sqrt{x}$ .

- ▶ `sqrt( ± 0 )` returns  $\pm 0$ .
- ▶ `sqrt( + ∞ )` returns  $+\infty$ .
- ▶ `sqrt(x)` returns NaN if  $x$  is less than 0.

**Description**

Calculate the nonnegative square root of  $x$ ,  $\sqrt{x}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float tanf (float x)`

Calculate the tangent of the input argument.

**Returns**

- ▶ `tanf( ±0 )` returns  $\pm 0$ .
- ▶ `tanf( ±∞ )` returns NaN.

**Description**

Calculate the tangent of the input argument  $x$  (measured in radians).



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This function is affected by the `--use_fast_math` compiler flag. See the CUDA C Programming Guide, Appendix D.2, Table 8 for a complete list of functions affected.

## `__host__ __device__ float tanhf (float x)`

Calculate the hyperbolic tangent of the input argument.

**Returns**

- ▶ `tanhf( ±0 )` returns  $\pm 0$ .

**Description**

Calculate the hyperbolic tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float tgammaf (float x)`

Calculate the gamma function of the input argument.

### Returns

- ▶ `tgammaf( ± 0 )` returns  $\pm \infty$ .
- ▶ `tgammaf(2)` returns +0.
- ▶ `tgammaf(x)` returns  $\pm \infty$  if the correctly calculated value is outside the single floating point range.
- ▶ `tgammaf(x)` returns NaN if  $x < 0$ .
- ▶ `tgammaf( - ∞ )` returns NaN.
- ▶ `tgammaf( + ∞ )` returns  $+\infty$ .

### Description

Calculate the gamma function of the input argument  $x$ , namely the value of  $\int_0^\infty e^{-t} t^{x-1} dt$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ float truncf (float x)`

Truncate input argument to the integral part.

### Returns

Returns truncated integer value.

### Description

Round  $x$  to the nearest integer value that does not exceed  $x$  in magnitude.

## `__host__ __device__ float y0f (float x)`

Calculate the value of the Bessel function of the second kind of order 0 for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order 0.

- ▶ `y0f(0)` returns  $-\infty$ .
- ▶ `y0f(x)` returns NaN for  $x < 0$ .
- ▶ `y0f( + ∞ )` returns +0.



- ▶ `y0f(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order 0 for the input argument  $x$ ,  $Y_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float y1f (float x)

Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order 1.

- ▶ `y1f(0)` returns  $-\infty$ .
- ▶ `y1f(x)` returns NaN for  $x < 0$ .
- ▶ `y1f(+∞)` returns +0.
- ▶ `y1f(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order 1 for the input argument  $x$ ,  $Y_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## \_\_host\_\_ \_\_device\_\_ float ynf (int n, float x)

Calculate the value of the Bessel function of the second kind of order  $n$  for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order  $n$ .

- ▶ `ynf(n, x)` returns NaN for  $n < 0$ .
- ▶ `ynf(n, 0)` returns  $-\infty$ .
- ▶ `ynf(n, x)` returns NaN for  $x < 0$ .
- ▶ `ynf(n, +∞)` returns +0.

- ▶ `ynf(n, NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order  $n$  for the input argument  $x$ ,  $Y_n(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## 1.3. Double Precision Mathematical Functions

This section describes double precision mathematical functions.

### `__host__ __device__ double acos (double x)`

Calculate the arc cosine of the input argument.

#### Returns

Result will be in radians, in the interval  $[0, \pi]$  for  $x$  inside  $[-1, +1]$ .

- ▶ `acos(1)` returns +0.
- ▶ `acos(x)` returns NaN for  $x$  outside  $[-1, +1]$ .

### Description

Calculate the principal value of the arc cosine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

### `__host__ __device__ double acosh (double x)`

Calculate the nonnegative arc hyperbolic cosine of the input argument.

#### Returns

Result will be in the interval  $[0, +\infty]$ .

- ▶ `acosh(1)` returns 0.
- ▶ `acosh(x)` returns NaN for  $x$  in the interval  $[-\infty, 1)$ .

**Description**

Calculate the nonnegative arc hyperbolic cosine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double asin (double x)`

Calculate the arc sine of the input argument.

**Returns**

Result will be in radians, in the interval  $[-\pi/2, +\pi/2]$  for  $x$  inside  $[-1, +1]$ .

- ▶ `asin(0)` returns +0.
- ▶ `asin(x)` returns NaN for  $x$  outside  $[-1, +1]$ .

**Description**

Calculate the principal value of the arc sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double asinh (double x)`

Calculate the arc hyperbolic sine of the input argument.

**Returns**

- ▶ `asinh(0)` returns 1.

**Description**

Calculate the arc hyperbolic sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double atan (double x)`

Calculate the arc tangent of the input argument.

### Returns

Result will be in radians, in the interval  $[-\pi/2, +\pi/2]$ .

- ▶ `atan(0)` returns +0.

### Description

Calculate the principal value of the arc tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double atan2 (double y, double x)`

Calculate the arc tangent of the ratio of first and second input arguments.

### Returns

Result will be in radians, in the interval  $[-\pi, +\pi]$ .

- ▶ `atan2(0, 1)` returns +0.

### Description

Calculate the principal value of the arc tangent of the ratio of first and second input arguments  $y/x$ . The quadrant of the result is determined by the signs of inputs  $y$  and  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double atanh (double x)`

Calculate the arc hyperbolic tangent of the input argument.

### Returns

- ▶ `atanh( $\pm 0$ )` returns  $\pm 0$ .
- ▶ `atanh( $\pm 1$ )` returns  $\pm \infty$ .
- ▶ `atanh( $x$ )` returns NaN for  $x$  outside interval  $[-1, 1]$ .

**Description**

Calculate the arc hyperbolic tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double cbrt (double x)`

Calculate the cube root of the input argument.

**Returns**

Returns  $x^{1/3}$ .

- ▶ `cbrt(  $\pm 0$  )` returns  $\pm 0$ .
- ▶ `cbrt(  $\pm \infty$  )` returns  $\pm \infty$ .

**Description**

Calculate the cube root of  $x$ ,  $x^{1/3}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double ceil (double x)`

Calculate ceiling of the input argument.

**Returns**

Returns  $\lceil x \rceil$  expressed as a floating-point number.

- ▶ `ceil(  $\pm 0$  )` returns  $\pm 0$ .
- ▶ `ceil(  $\pm \infty$  )` returns  $\pm \infty$ .

**Description**

Compute the smallest integer value not less than  $x$ .

## `__host__ __device__ double copysign (double x, double y)`

Create value with given magnitude, copying sign of second value.

### Returns

Returns a value with the magnitude of  $x$  and the sign of  $y$ .

### Description

Create a floating-point value with the magnitude  $x$  and the sign of  $y$ .

## `__host__ __device__ double cos (double x)`

Calculate the cosine of the input argument.

### Returns

- ▶  $\cos(\pm 0)$  returns 1.
- ▶  $\cos(\pm \infty)$  returns NaN.

### Description

Calculate the cosine of the input argument  $x$  (measured in radians).



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double cosh (double x)`

Calculate the hyperbolic cosine of the input argument.

### Returns

- ▶  $\cosh(0)$  returns 1.
- ▶  $\cosh(\pm \infty)$  returns  $+\infty$ .

### Description

Calculate the hyperbolic cosine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double cospi (double x)`

Calculate the cosine of the input argument  $\times \pi$ .

### Returns

- ▶ `cospi(  $\pm 0$  )` returns 1.
- ▶ `cospi(  $\pm \infty$  )` returns NaN.

### Description

Calculate the cosine of  $x \times \pi$  (measured in radians), where  $x$  is the input argument.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double cyl_bessel_i0 (double x)`

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument.

### Returns

Returns the value of the regular modified cylindrical Bessel function of order 0.

### Description

Calculate the value of the regular modified cylindrical Bessel function of order 0 for the input argument  $x$ ,  $I_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double cyl_bessel_i1 (double x)`

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument.

### Returns

Returns the value of the regular modified cylindrical Bessel function of order 1.

**Description**

Calculate the value of the regular modified cylindrical Bessel function of order 1 for the input argument  $x$ ,  $I_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double erf (double x)**

Calculate the error function of the input argument.

**Returns**

- ▶  $\text{erf}(\pm 0)$  returns  $\pm 0$ .
- ▶  $\text{erf}(\pm \infty)$  returns  $\pm 1$ .

**Description**

Calculate the value of the error function for the input argument  $x$ ,  $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double erfc (double x)**

Calculate the complementary error function of the input argument.

**Returns**

- ▶  $\text{erfc}(-\infty)$  returns 2.
- ▶  $\text{erfc}(+\infty)$  returns +0.

**Description**

Calculate the complementary error function of the input argument  $x$ ,  $1 - \text{erf}(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.



## `__host__ __device__ double erfcinv (double y)`

Calculate the inverse complementary error function of the input argument.

### Returns

- ▶ `erfcinv(0)` returns  $+\infty$ .
- ▶ `erfcinv(2)` returns  $-\infty$ .

### Description

Calculate the inverse complementary error function of the input argument  $y$ , for  $y$  in the interval  $[0, 2]$ . The inverse complementary error function find the value  $x$  that satisfies the equation  $y = \text{erfc}(x)$ , for  $0 \leq y \leq 2$ , and  $-\infty \leq x \leq \infty$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double erfcx (double x)`

Calculate the scaled complementary error function of the input argument.

### Returns

- ▶ `erfcx(-∞)` returns  $+\infty$
- ▶ `erfcx(+∞)` returns  $+0$
- ▶ `erfcx(x)` returns  $+\infty$  if the correctly calculated value is outside the double floating point range.

### Description

Calculate the scaled complementary error function of the input argument  $x$ ,  $e^{x^2} \cdot \text{erfc}(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double erfinv (double y)`

Calculate the inverse error function of the input argument.

### Returns

- ▶ `erfinv(1)` returns  $+\infty$ .
- ▶ `erfinv(-1)` returns  $-\infty$ .

**Description**

Calculate the inverse error function of the input argument  $y$ , for  $y$  in the interval  $[-1, 1]$ . The inverse error function finds the value  $x$  that satisfies the equation  $y = \text{erf}(x)$ , for  $-1 \leq y \leq 1$ , and  $-\infty \leq x \leq \infty$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double exp (double x)**

Calculate the base  $e$  exponential of the input argument.

**Returns**

Returns  $e^x$ .

**Description**

Calculate the base  $e$  exponential of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double exp10 (double x)**

Calculate the base 10 exponential of the input argument.

**Returns**

Returns  $10^x$ .

**Description**

Calculate the base 10 exponential of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double exp2 (double x)**

Calculate the base 2 exponential of the input argument.

**Returns**

Returns  $2^x$ .

**Description**

Calculate the base 2 exponential of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double expm1 (double x)**

Calculate the base  $e$  exponential of the input argument, minus 1.

**Returns**

Returns  $e^x - 1$ .

**Description**

Calculate the base  $e$  exponential of the input argument  $x$ , minus 1.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double fabs (double x)**

Calculate the absolute value of the input argument.

**Returns**

Returns the absolute value of the input argument.

- ▶ `fabs(  $\pm \infty$  )` returns  $+\infty$ .
- ▶ `fabs(  $\pm 0$  )` returns 0.

**Description**

Calculate the absolute value of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double fdim (double x, double y)`

Compute the positive difference between  $x$  and  $y$ .

### Returns

Returns the positive difference between  $x$  and  $y$ .

- ▶ `fdim(x, y)` returns  $x - y$  if  $x > y$ .
- ▶ `fdim(x, y)` returns  $+0$  if  $x \leq y$ .

### Description

Compute the positive difference between  $x$  and  $y$ . The positive difference is  $x - y$  when  $x > y$  and  $+0$  otherwise.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__host__ __device__ double floor (double x)`

Calculate the largest integer less than or equal to  $x$ .

### Returns

Returns  $\log_e(1+x)$  expressed as a floating-point number.

- ▶ `floor(  $\pm \infty$  )` returns  $\pm \infty$ .
- ▶ `floor(  $\pm 0$  )` returns  $\pm 0$ .

### Description

Calculates the largest integer value which is less than or equal to  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double fma (double x, double y, double z)`

Compute  $x \times y + z$  as a single operation.

### Returns

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fma(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fma(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fma( $x$ ,  $y$ ,  $-\infty$  )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fma( $x$ ,  $y$ ,  $+\infty$  )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

### Description

Compute the value of  $x \times y + z$  as a single ternary operation. After computing the value to infinite precision, the value is rounded once.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double fmax (double, double)`

Determine the maximum numeric value of the arguments.

### Returns

Returns the maximum numeric values of the arguments  $x$  and  $y$ .

- ▶ If both arguments are NaN, returns NaN.
- ▶ If one argument is NaN, returns the numeric argument.

### Description

Determines the maximum numeric value of the arguments  $x$  and  $y$ . Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double fmin (double x, double y)`

Determine the minimum numeric value of the arguments.

### Returns

Returns the minimum numeric values of the arguments  $x$  and  $y$ .

- ▶ If both arguments are NaN, returns NaN.
- ▶ If one argument is NaN, returns the numeric argument.

### Description

Determines the minimum numeric value of the arguments  $x$  and  $y$ . Treats NaN arguments as missing data. If one argument is a NaN and the other is legitimate numeric value, the numeric value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double fmod (double x, double y)`

Calculate the floating-point remainder of  $x / y$ .

### Returns

- ▶ Returns the floating point remainder of  $x / y$ .
- ▶  $fmod(\pm 0, y)$  returns  $\pm 0$  if  $y$  is not zero.
- ▶  $fmod(x, y)$  returns NaN and raised an invalid floating point exception if  $x$  is  $\pm \infty$  or  $y$  is zero.
- ▶  $fmod(x, y)$  returns zero if  $y$  is zero or the result would overflow.
- ▶  $fmod(x, \pm \infty)$  returns  $x$  if  $x$  is finite.
- ▶  $fmod(x, 0)$  returns NaN.

### Description

Calculate the floating-point remainder of  $x / y$ . The absolute value of the computed value is always less than  $y$ 's absolute value and will have the same sign as  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double frexp (double x, int *nptr)`

Extract mantissa and exponent of a floating-point value.

### Returns

Returns the fractional component  $m$ .

- ▶ `frexp(0, nptr)` returns 0 for the fractional component and zero for the integer component.
- ▶ `frexp( $\pm 0$ , nptr)` returns  $\pm 0$  and stores zero in the location pointed to by `nptr`.
- ▶ `frexp( $\pm \infty$ , nptr)` returns  $\pm \infty$  and stores an unspecified value in the location to which `nptr` points.
- ▶ `frexp(NaN, y)` returns a NaN and stores an unspecified value in the location to which `nptr` points.

### Description

Decompose the floating-point value  $x$  into a component  $m$  for the normalized fraction element and another term  $n$  for the exponent. The absolute value of  $m$  will be greater than or equal to 0.5 and less than 1.0 or it will be equal to 0;  $x = m \cdot 2^n$ . The integer exponent  $n$  will be stored in the location to which `nptr` points.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double __CRTDECL hypot (double x, double y)`

Calculate the square root of the sum of squares of two arguments.

### Returns

Returns the length of the hypotenuse  $\sqrt{x^2 + y^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

### Description

Calculate the length of the hypotenuse of a right triangle whose two sides have lengths  $x$  and  $y$  without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ int ilogb (double x)`

Compute the unbiased integer exponent of the argument.

### Returns

- ▶ If successful, returns the unbiased exponent of the argument.
- ▶ `ilogb(0)` returns `INT_MIN`.
- ▶ `ilogb(NaN)` returns `NaN`.
- ▶ `ilogb(x)` returns `INT_MAX` if  $x$  is  $\infty$  or the correct value is greater than `INT_MAX`.
- ▶ `ilogb(x)` return `INT_MIN` if the correct value is less than `INT_MIN`.

### Description

Calculates the unbiased integer exponent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ __RETURN_TYPE isfinite (double a)`

Determine whether argument is finite.

### Returns

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if  $a$  is a finite value.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if  $a$  is a finite value.

### Description

Determine whether the floating-point value  $a$  is a finite value (zero, subnormal, or normal and not infinity or NaN).

## `__host__ __device__ __RETURN_TYPE isinf (double a)`

Determine whether argument is infinite.

### Returns

- ▶ With Visual Studio 2013 host compiler: Returns true if and only if  $a$  is a infinite value.
- ▶ With other host compilers: Returns a nonzero value if and only if  $a$  is a infinite value.



**Description**

Determine whether the floating-point value *a* is an infinite value (positive or negative).

**\_\_host\_\_ \_\_device\_\_ \_\_RETURN\_TYPE isnan (double a)**

Determine whether argument is a NaN.

**Returns**

- ▶ With Visual Studio 2013 host compiler: \_\_RETURN\_TYPE is 'bool'. Returns true if and only if *a* is a NaN value.
- ▶ With other host compilers: \_\_RETURN\_TYPE is 'int'. Returns a nonzero value if and only if *a* is a NaN value.

**Description**

Determine whether the floating-point value *a* is a NaN.

**\_\_host\_\_ \_\_device\_\_ double j0 (double x)**

Calculate the value of the Bessel function of the first kind of order 0 for the input argument.

**Returns**

Returns the value of the Bessel function of the first kind of order 0.

- ▶  $j_0(\pm\infty)$  returns +0.
- ▶  $j_0(\text{NaN})$  returns NaN.

**Description**

Calculate the value of the Bessel function of the first kind of order 0 for the input argument *x*,  $J_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double j1 (double x)**

Calculate the value of the Bessel function of the first kind of order 1 for the input argument.

**Returns**

Returns the value of the Bessel function of the first kind of order 1.

- ▶  $j_1(\pm 0)$  returns  $\pm 0$ .
- ▶  $j_1(\pm \infty)$  returns  $+0$ .
- ▶  $j_1(\text{NaN})$  returns NaN.

### Description

Calculate the value of the Bessel function of the first kind of order 1 for the input argument  $x$ ,  $J_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double jn (int n, double x)

Calculate the value of the Bessel function of the first kind of order  $n$  for the input argument.

### Returns

Returns the value of the Bessel function of the first kind of order  $n$ .

- ▶  $jn(n, \text{NaN})$  returns NaN.
- ▶  $jn(n, x)$  returns NaN for  $n < 0$ .
- ▶  $jn(n, +\infty)$  returns  $+0$ .

### Description

Calculate the value of the Bessel function of the first kind of order  $n$  for the input argument  $x$ ,  $J_n(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double ldexp (double x, int exp)

Calculate the value of  $x \cdot 2^{\text{exp}}$ .

### Returns

- ▶  $\text{ldexp}(x)$  returns  $\pm \infty$  if the correctly calculated value is outside the double floating point range.

### Description

Calculate the value of  $x \cdot 2^{\text{exp}}$  of the input arguments  $x$  and  $\text{exp}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double lgamma (double x)`

Calculate the natural logarithm of the absolute value of the gamma function of the input argument.

### Returns

- ▶ `lgamma(1)` returns +0.
- ▶ `lgamma(2)` returns +0.
- ▶ `lgamma(x)` returns  $\pm \infty$  if the correctly calculated value is outside the double floating point range.
- ▶ `lgamma(x)` returns  $+\infty$  if  $x \leq 0$ .
- ▶ `lgamma(-∞)` returns  $-\infty$ .
- ▶ `lgamma(+∞)` returns  $+\infty$ .

### Description

Calculate the natural logarithm of the absolute value of the gamma function of the input argument  $x$ , namely the value of  $\log_e \left| \int_0^\infty e^{-t} t^{x-1} dt \right|$



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ long long int llrint (double x)`

Round input to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded towards zero. If the result is outside the range of the return type, the result is undefined.

## `__host__ __device__ long long int llround (double x)`

Round to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the result is undefined.



This function may be slower than alternate rounding methods. See [llrint\(\)](#).

## `__host__ __device__ double log (double x)`

Calculate the base  $e$  logarithm of the input argument.

### Returns

- ▶  $\log(\pm 0)$  returns  $-\infty$ .
- ▶  $\log(1)$  returns  $+0$ .
- ▶  $\log(x)$  returns NaN for  $x < 0$ .
- ▶  $\log(+\infty)$  returns  $+\infty$ .

### Description

Calculate the base  $e$  logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double log10 (double x)`

Calculate the base 10 logarithm of the input argument.

### Returns

- ▶  $\log_{10}(\pm 0)$  returns  $-\infty$ .
- ▶  $\log_{10}(1)$  returns  $+0$ .
- ▶  $\log_{10}(x)$  returns NaN for  $x < 0$ .
- ▶  $\log_{10}(+\infty)$  returns  $+\infty$ .

**Description**

Calculate the base 10 logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double log1p (double x)`

Calculate the value of  $\log_e(1+x)$ .

**Returns**

- ▶  $\log1p(\pm 0)$  returns  $-\infty$ .
- ▶  $\log1p(-1)$  returns  $+0$ .
- ▶  $\log1p(x)$  returns NaN for  $x < -1$ .
- ▶  $\log1p(+\infty)$  returns  $+\infty$ .

**Description**

Calculate the value of  $\log_e(1+x)$  of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double log2 (double x)`

Calculate the base 2 logarithm of the input argument.

**Returns**

- ▶  $\log2(\pm 0)$  returns  $-\infty$ .
- ▶  $\log2(1)$  returns  $+0$ .
- ▶  $\log2(x)$  returns NaN for  $x < 0$ .
- ▶  $\log2(+\infty)$  returns  $+\infty$ .

**Description**

Calculate the base 2 logarithm of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double logb (double x)`

Calculate the floating point representation of the exponent of the input argument.

### Returns

- ▶ `logb ± 0` returns  $-\infty$
- ▶ `logb ± ∞` returns  $+\infty$

### Description

Calculate the floating point representation of the exponent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ long int lrint (double x)`

Round input to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded towards zero. If the result is outside the range of the return type, the result is undefined.

## `__host__ __device__ long int lround (double x)`

Round to nearest integer value.

### Returns

Returns rounded integer value.

### Description

Round  $x$  to the nearest integer value, with halfway cases rounded away from zero. If the result is outside the range of the return type, the result is undefined.



This function may be slower than alternate rounding methods. See [lrint\(\)](#).

## `__host__ __device__ double modf (double x, double *iptr)`

Break down the input argument into fractional and integral parts.

### Returns

- ▶ `modf(  $\pm x$ , iptr )` returns a result with the same sign as  $x$ .
- ▶ `modf(  $\pm \infty$ , iptr )` returns  $\pm 0$  and stores  $\pm \infty$  in the object pointed to by `iptr`.
- ▶ `modf(NaN, iptr )` stores a NaN in the object pointed to by `iptr` and returns a NaN.

### Description

Break down the argument  $x$  into fractional and integral parts. The integral part is stored in the argument `iptr`. Fractional and integral parts are given the same sign as the argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double nan (const char *tagp)`

Returns "Not a Number" value.

### Returns

- ▶ `nan(tagp )` returns NaN.

### Description

Return a representation of a quiet NaN. Argument `tagp` selects one of the possible representations.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double nearbyint (double x)`

Round the input argument to the nearest integer.

### Returns

- ▶ `nearbyint(  $\pm 0$  )` returns  $\pm 0$ .
- ▶ `nearbyint(  $\pm \infty$  )` returns  $\pm \infty$ .

**Description**

Round argument  $x$  to an integer value in double precision floating-point format.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double nextafter (double x, double y)`

Return next representable double-precision floating-point value after argument.

**Returns**

- ▶ `nextafter(  $\pm \infty$ ,  $y$  )` returns  $\pm \infty$ .

**Description**

Calculate the next representable double-precision floating-point value following  $x$  in the direction of  $y$ . For example, if  $y$  is greater than  $x$ , `nextafter()` returns the smallest representable number greater than  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double norm3d (double a, double b, double c)`

Calculate the square root of the sum of squares of three coordinates of the argument.

**Returns**

Returns the length of 3D vector  $\sqrt{p.x^2 + p.y^2 + p.z^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

**Description**

Calculate the length of three dimensional vector  $p$  in euclidean space without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.



## `__host__ __device__ double norm4d (double a, double b, double c, double d)`

Calculate the square root of the sum of squares of four coordinates of the argument.

### Returns

Returns the length of 4D vector  $\sqrt{p.x^2 + p.y^2 + p.z^2 + p.t^2}$ . If the correct value would overflow, returns  $+\infty$ . If the correct value would underflow, returns 0.

### Description

Calculate the length of four dimensional vector  $p$  in euclidean space without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double normcdf (double y)`

Calculate the standard normal cumulative distribution function.

### Returns

- ▶ `normcdf(  $+\infty$  )` returns 1
- ▶ `normcdf(  $-\infty$  )` returns +0

### Description

Calculate the cumulative distribution function of the standard normal distribution for input argument  $y$ ,  $\Phi(y)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double normcdfinv (double y)`

Calculate the inverse of the standard normal cumulative distribution function.

### Returns

- ▶ `normcdfinv(0)` returns  $-\infty$ .
- ▶ `normcdfinv(1)` returns  $+\infty$ .
- ▶ `normcdfinv(x)` returns NaN if  $x$  is not in the interval  $[0,1]$ .

## Description

Calculate the inverse of the standard normal cumulative distribution function for input argument  $y$ ,  $\Phi^{-1}(y)$ . The function is defined for input values in the interval (0, 1).



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double pow (double x, double y)`

Calculate the value of first argument to the power of second argument.

## Returns

- ▶ `pow( ±0, y)` returns  $\pm\infty$  for  $y$  an integer less than 0.
- ▶ `pow( ±0, y)` returns  $\pm 0$  for  $y$  an odd integer greater than 0.
- ▶ `pow( ±0, y)` returns +0 for  $y > 0$  and not an odd integer.
- ▶ `pow(-1, ±∞)` returns 1.
- ▶ `pow(+1, y)` returns 1 for any  $y$ , even a NaN.
- ▶ `pow(x, ±0)` returns 1 for any  $x$ , even a NaN.
- ▶ `pow(x, y)` returns a NaN for finite  $x < 0$  and finite non-integer  $y$ .
- ▶ `pow(x, -∞)` returns  $+\infty$  for  $|x| < 1$ .
- ▶ `pow(x, -∞)` returns +0 for  $|x| > 1$ .
- ▶ `pow(x, +∞)` returns +0 for  $|x| < 1$ .
- ▶ `pow(x, +∞)` returns  $+\infty$  for  $|x| > 1$ .
- ▶ `pow(-∞, y)` returns -0 for  $y$  an odd integer less than 0.
- ▶ `pow(-∞, y)` returns +0 for  $y < 0$  and not an odd integer.
- ▶ `pow(-∞, y)` returns  $-\infty$  for  $y$  an odd integer greater than 0.
- ▶ `pow(-∞, y)` returns  $+\infty$  for  $y > 0$  and not an odd integer.
- ▶ `pow(+∞, y)` returns +0 for  $y < 0$ .
- ▶ `pow(+∞, y)` returns  $+\infty$  for  $y > 0$ .

## Description

Calculate the value of  $x$  to the power of  $y$



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double rcbrt (double x)`

Calculate reciprocal cube root function.

### Returns

- ▶ `rcbrt(  $\pm 0$  )` returns  $\pm \infty$ .
- ▶ `rcbrt(  $\pm \infty$  )` returns  $\pm 0$ .

### Description

Calculate reciprocal cube root function of  $x$



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double remainder (double x, double y)`

Compute double-precision floating-point remainder.

### Returns

- ▶ `remainder( $x$ , 0)` returns NaN.
- ▶ `remainder(  $\pm \infty$ ,  $y$  )` returns NaN.
- ▶ `remainder( $x$ ,  $\pm \infty$  )` returns  $x$  for finite  $x$ .

### Description

Compute double-precision floating-point remainder  $r$  of dividing  $x$  by  $y$  for nonzero  $y$ .

Thus  $r = x - ny$ . The value  $n$  is the integer value nearest  $\frac{x}{y}$ . In the case when  $|n - \frac{x}{y}| = \frac{1}{2}$ , the even  $n$  value is chosen.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double remquo (double x, double y, int *quo)`

Compute double-precision floating-point remainder and part of quotient.

### Returns

Returns the remainder.

- ▶ `remquo(x, 0, quo)` returns NaN.
- ▶ `remquo(±∞, y, quo)` returns NaN.
- ▶ `remquo(x, ±∞, quo)` returns x.

### Description

Compute a double-precision floating-point remainder in the same way as the `remainder()` function. Argument `quo` returns part of quotient upon division of `x` by `y`. Value `quo` has the same sign as  $\frac{x}{y}$  and may not be the exact quotient but agrees with the exact quotient in the low order 3 bits.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double rhypot (double x, double y)`

Calculate one over the square root of the sum of squares of two arguments.

### Returns

Returns one over the length of the hypotenuse  $\frac{1}{\sqrt{x^2 + y^2}}$ . If the square root would overflow, returns 0. If the square root would underflow, returns  $+\infty$ .

### Description

Calculate one over the length of the hypotenuse of a right triangle whose two sides have lengths `x` and `y` without undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double rint (double x)**

Round to nearest integer value in floating-point.

**Returns**

Returns rounded integer value.

**Description**

Round  $x$  to the nearest integer value in floating-point format, with halfway cases rounded to the nearest even integer value.

**\_\_host\_\_ \_\_device\_\_ double rnorm3d (double a, double b, double c)**

Calculate one over the square root of the sum of squares of three coordinates of the argument.

**Returns**

Returns one over the length of the 3D vector  $\frac{1}{\sqrt{p.x^2 + p.y^2 + p.z^2}}$ . If the square root would overflow, returns 0. If the square root would underflow, returns  $+\infty$ .

**Description**

Calculate one over the length of three dimensional vector  $p$  in euclidean space undue overflow or underflow.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

**\_\_host\_\_ \_\_device\_\_ double round (double x)**

Round to nearest integer value in floating-point.

**Returns**

Returns rounded integer value.

**Description**

Round  $x$  to the nearest integer value in floating-point format, with halfway cases rounded away from zero.



This function may be slower than alternate rounding methods. See [rint\(\)](#).

## `__host__ __device__ double rsqrt (double x)`

Calculate the reciprocal of the square root of the input argument.

### Returns

Returns  $1/\sqrt{x}$ .

- ▶ `rsqrt( +∞ )` returns `+0`.
- ▶ `rsqrt( ±0 )` returns  $\pm\infty$ .
- ▶ `rsqrt(x)` returns NaN if  $x$  is less than 0.

### Description

Calculate the reciprocal of the nonnegative square root of  $x$ ,  $1/\sqrt{x}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double scalbln (double x, long int n)`

Scale floating-point input by integer power of two.

### Returns

Returns  $x * 2^n$ .

- ▶ `scalbln( ±0 , n )` returns  $\pm 0$ .
- ▶ `scalbln(x, 0)` returns  $x$ .
- ▶ `scalbln( ±∞ , n )` returns  $\pm\infty$ .

### Description

Scale  $x$  by  $2^n$  by efficient manipulation of the floating-point exponent.

## `__host__ __device__ double scalbn (double x, int n)`

Scale floating-point input by integer power of two.

### Returns

Returns  $x * 2^n$ .

- ▶ `scalbn( ±0, n)` returns  $±0$ .
- ▶ `scalbn(x, 0)` returns  $x$ .
- ▶ `scalbn( ±∞, n)` returns  $±∞$ .

### Description

Scale  $x$  by  $2^n$  by efficient manipulation of the floating-point exponent.

## `__host__ __device__ __RETURN_TYPE signbit (double a)`

Return the sign bit of the input.

### Returns

Reports the sign bit of all values including infinities, zeros, and NaNs.

- ▶ With Visual Studio 2013 host compiler: `__RETURN_TYPE` is 'bool'. Returns true if and only if  $a$  is negative.
- ▶ With other host compilers: `__RETURN_TYPE` is 'int'. Returns a nonzero value if and only if  $a$  is negative.

### Description

Determine whether the floating-point value  $a$  is negative.

## `__host__ __device__ double sin (double x)`

Calculate the sine of the input argument.

### Returns

- ▶ `sin( ±0 )` returns  $±0$ .
- ▶ `sin( ±∞ )` returns NaN.

### Description

Calculate the sine of the input argument  $x$  (measured in radians).



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

`__host__ __device__ void sincos (double x, double *sptr, double *cptr)`

Calculate the sine and cosine of the first input argument.

#### Returns

- ▶ none

#### Description

Calculate the sine and cosine of the first input argument  $x$  (measured in radians). The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`.

#### See also:

`sin()` and `cos()`.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

`__host__ __device__ void sincospi (double x, double *sptr, double *cptr)`

Calculate the sine and cosine of the first input argument  $x \times \pi$ .

#### Returns

- ▶ none

#### Description

Calculate the sine and cosine of the first input argument,  $x$  (measured in radians),  $x \times \pi$ . The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`.

#### See also:

`sinpi()` and `cospi()`.





For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double sinh (double x)`

Calculate the hyperbolic sine of the input argument.

### Returns

- `sinh( ±0 )` returns  $\pm 0$ .

### Description

Calculate the hyperbolic sine of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double sinpi (double x)`

Calculate the sine of the input argument  $\times \pi$ .

### Returns

- `sinpi( ±0 )` returns  $\pm 0$ .
- `sinpi( ±∞ )` returns NaN.

### Description

Calculate the sine of  $x \times \pi$  (measured in radians), where  $x$  is the input argument.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double sqrt (double x)`

Calculate the square root of the input argument.

### Returns

Returns  $\sqrt{x}$ .

- `sqrt( ±0 )` returns  $\pm 0$ .

- ▶  $\text{sqrt}(+\infty)$  returns  $+\infty$ .
- ▶  $\text{sqrt}(x)$  returns NaN if  $x$  is less than 0.

### Description

Calculate the nonnegative square root of  $x$ ,  $\sqrt{x}$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double tan (double x)

Calculate the tangent of the input argument.

### Returns

- ▶  $\tan(\pm 0)$  returns  $\pm 0$ .
- ▶  $\tan(\pm \infty)$  returns NaN.

### Description

Calculate the tangent of the input argument  $x$  (measured in radians).



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double tanh (double x)

Calculate the hyperbolic tangent of the input argument.

### Returns

- ▶  $\tanh(\pm 0)$  returns  $\pm 0$ .

### Description

Calculate the hyperbolic tangent of the input argument  $x$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double tgamma (double x)`

Calculate the gamma function of the input argument.

### Returns

- ▶ `tgamma( ± 0 )` returns  $\pm \infty$ .
- ▶ `tgamma(2)` returns +0.
- ▶ `tgamma(x)` returns  $\pm \infty$  if the correctly calculated value is outside the double floating point range.
- ▶ `tgamma(x)` returns NaN if  $x < 0$ .
- ▶ `tgamma( -  $\infty$  )` returns NaN.
- ▶ `tgamma( +  $\infty$  )` returns  $+\infty$ .

### Description

Calculate the gamma function of the input argument  $x$ , namely the value of  $\int_0^\infty e^{-t} t^{x-1} dt$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## `__host__ __device__ double trunc (double x)`

Truncate input argument to the integral part.

### Returns

Returns truncated integer value.

### Description

Round  $x$  to the nearest integer value that does not exceed  $x$  in magnitude.

## `__host__ __device__ double y0 (double x)`

Calculate the value of the Bessel function of the second kind of order 0 for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order 0.

- ▶ `y0(0)` returns  $-\infty$ .
- ▶ `y0(x)` returns NaN for  $x < 0$ .
- ▶ `y0( +  $\infty$  )` returns +0.

- ▶ `y0(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order 0 for the input argument  $x$ ,  $Y_0(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double y1 (double x)

Calculate the value of the Bessel function of the second kind of order 1 for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order 1.

- ▶ `y1(0)` returns  $-\infty$ .
- ▶ `y1(x)` returns NaN for  $x < 0$ .
- ▶ `y1(+∞)` returns +0.
- ▶ `y1(NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order 1 for the input argument  $x$ ,  $Y_1(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## \_\_host\_\_ \_\_device\_\_ double yn (int n, double x)

Calculate the value of the Bessel function of the second kind of order  $n$  for the input argument.

### Returns

Returns the value of the Bessel function of the second kind of order  $n$ .

- ▶ `yn(n, x)` returns NaN for  $n < 0$ .
- ▶ `yn(n, 0)` returns  $-\infty$ .
- ▶ `yn(n, x)` returns NaN for  $x < 0$ .
- ▶ `yn(n, +∞)` returns +0.

- ▶ `yn(n, NaN)` returns NaN.

### Description

Calculate the value of the Bessel function of the second kind of order  $n$  for the input argument  $x$ ,  $Y_n(x)$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.

## 1.4. Single Precision Ininsics

This section describes single precision intrinsic functions that are only supported in device code.

### `__DEVICE_FUNCTIONS_DECL__ float __cosf (float x)`

Calculate the fast approximate cosine of the input argument.

#### Returns

Returns the approximate cosine of  $x$ .

#### Description

Calculate the fast approximate cosine of the input argument  $x$ , measured in radians.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Input and output in the denormal range is flushed to sign preserving 0.0.

### `__DEVICE_FUNCTIONS_DECL__ float __exp10f (float x)`

Calculate the fast approximate base 10 exponential of the input argument.

#### Returns

Returns an approximation to  $10^x$ .

#### Description

Calculate the fast approximate base 10 exponential of the input argument  $x$ ,  $10^x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Most input and output values around denormal range are flushed to sign preserving 0.0.

## `__DEVICE_FUNCTIONS_DECL__ float __expf (float x)`

Calculate the fast approximate base  $e$  exponential of the input argument.

### Returns

Returns an approximation to  $e^x$ .

### Description

Calculate the fast approximate base  $e$  exponential of the input argument  $x$ ,  $e^x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Most input and output values around denormal range are flushed to sign preserving 0.0.

## `__DEVICE_FUNCTIONS_DECL__ float __fadd_rd (float x, float y)`

Add two floating point values in round-down mode.

### Returns

Returns  $x + y$ .

### Description

Compute the sum of  $x$  and  $y$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fadd_rn (float x, float y)`

Add two floating point values in round-to-nearest-even mode.

### Returns

Returns  $x + y$ .

### Description

Compute the sum of  $x$  and  $y$  in round-to-nearest-even rounding mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fadd_ru (float x, float y)`

Add two floating point values in round-up mode.

### Returns

Returns  $x + y$ .

### Description

Compute the sum of  $x$  and  $y$  in round-up (to positive infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fadd_rz (float x, float y)`

Add two floating point values in round-towards-zero mode.

### Returns

Returns  $x + y$ .

**Description**

Compute the sum of  $x$  and  $y$  in round-towards-zero mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fdiv\_rd (float x, float y)**

Divide two floating point values in round-down mode.

**Returns**

Returns  $x / y$ .

**Description**

Divide two floating point values  $x$  by  $y$  in round-down (to negative infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fdiv\_rn (float x, float y)**

Divide two floating point values in round-to-nearest-even mode.

**Returns**

Returns  $x / y$ .

**Description**

Divide two floating point values  $x$  by  $y$  in round-to-nearest-even mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.



## `__DEVICE_FUNCTIONS_DECL__ float __fdiv_ru (float x, float y)`

Divide two floating point values in round-up mode.

### Returns

Returns  $x / y$ .

### Description

Divide two floating point values  $x$  by  $y$  in round-up (to positive infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__DEVICE_FUNCTIONS_DECL__ float __fdiv_rz (float x, float y)`

Divide two floating point values in round-towards-zero mode.

### Returns

Returns  $x / y$ .

### Description

Divide two floating point values  $x$  by  $y$  in round-towards-zero mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__DEVICE_FUNCTIONS_DECL__ float __fdividef (float x, float y)`

Calculate the fast approximate division of the input arguments.

### Returns

Returns  $x / y$ .

- ▶ `__fdividef(  $\infty$ , y )` returns NaN for  $2^{126} < y < 2^{128}$ .
- ▶ `__fdividef(x, y)` returns 0 for  $2^{126} < y < 2^{128}$  and  $x \neq \infty$ .

**Description**

Calculate the fast approximate division of  $x$  by  $y$ .



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fmaf\_rd (float x, float y, float z)**

Compute  $x \times y + z$  as a single operation, in round-down mode.

**Returns**

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fmaf(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fmaf(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fmaf( $x$ ,  $y$ ,  $-\infty$ )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fmaf( $x$ ,  $y$ ,  $+\infty$ )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

**Description**

Computes the value of  $x \times y + z$  as a single ternary operation, rounding the result once in round-down (to negative infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fmaf\_rn (float x, float y, float z)**

Compute  $x \times y + z$  as a single operation, in round-to-nearest-even mode.

**Returns**

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fmaf(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fmaf(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fmaf( $x$ ,  $y$ ,  $-\infty$ )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fmaf( $x$ ,  $y$ ,  $+\infty$ )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

## Description

Computes the value of  $x \times y + z$  as a single ternary operation, rounding the result once in round-to-nearest-even mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__DEVICE_FUNCTIONS_DECL__ float __fmaf_ru (float x, float y, float z)`

Compute  $x \times y + z$  as a single operation, in round-up mode.

## Returns

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fmaf(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fmaf(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fmaf( $x$ ,  $y$ ,  $-\infty$  )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fmaf( $x$ ,  $y$ ,  $+\infty$  )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

## Description

Computes the value of  $x \times y + z$  as a single ternary operation, rounding the result once in round-up (to positive infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## `__DEVICE_FUNCTIONS_DECL__ float __fmaf_rz (float x, float y, float z)`

Compute  $x \times y + z$  as a single operation, in round-towards-zero mode.

## Returns

Returns the rounded value of  $x \times y + z$  as a single operation.

- ▶ `fmaf(  $\pm \infty$ ,  $\pm 0$ ,  $z$  )` returns NaN.
- ▶ `fmaf(  $\pm 0$ ,  $\pm \infty$ ,  $z$  )` returns NaN.
- ▶ `fmaf( $x$ ,  $y$ ,  $-\infty$  )` returns NaN if  $x \times y$  is an exact  $+\infty$ .
- ▶ `fmaf( $x$ ,  $y$ ,  $+\infty$  )` returns NaN if  $x \times y$  is an exact  $-\infty$ .

**Description**

Computes the value of  $x \times y + z$  as a single ternary operation, rounding the result once in round-towards-zero mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fmul\_rd (float x, float y)**

Multiply two floating point values in round-down mode.

**Returns**

Returns  $x * y$ .

**Description**

Compute the product of  $x$  and  $y$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fmul\_rn (float x, float y)**

Multiply two floating point values in round-to-nearest-even mode.

**Returns**

Returns  $x * y$ .

**Description**

Compute the product of  $x$  and  $y$  in round-to-nearest-even mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fmul_ru (float x, float y)`

Multiply two floating point values in round-up mode.

### Returns

Returns  $x * y$ .

### Description

Compute the product of  $x$  and  $y$  in round-up (to positive infinity) mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fmul_rz (float x, float y)`

Multiply two floating point values in round-towards-zero mode.

### Returns

Returns  $x * y$ .

### Description

Compute the product of  $x$  and  $y$  in round-towards-zero mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __frcp_rd (float x)`

Compute  $\frac{1}{x}$  in round-down mode.

### Returns

Returns  $\frac{1}{x}$ .

**Description**

Compute the reciprocal of  $x$  in round-down (to negative infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_frcp\_rn (float x)**

Compute  $\frac{1}{x}$  in round-to-nearest-even mode.

**Returns**

Returns  $\frac{1}{x}$ .

**Description**

Compute the reciprocal of  $x$  in round-to-nearest-even mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_frcp\_ru (float x)**

Compute  $\frac{1}{x}$  in round-up mode.

**Returns**

Returns  $\frac{1}{x}$ .

**Description**

Compute the reciprocal of  $x$  in round-up (to positive infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_frcp\_rz (float x)**

Compute  $\frac{1}{x}$  in round-towards-zero mode.

**Returns**

Returns  $\frac{1}{x}$ .

**Description**

Compute the reciprocal of  $x$  in round-towards-zero mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_frsqrt\_rn (float x)**

Compute  $1/\sqrt{x}$  in round-to-nearest-even mode.

**Returns**

Returns  $1/\sqrt{x}$ .

**Description**

Compute the reciprocal square root of  $x$  in round-to-nearest-even mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsqrt\_rd (float x)**

Compute  $\sqrt{x}$  in round-down mode.

**Returns**

Returns  $\sqrt{x}$ .

**Description**

Compute the square root of  $x$  in round-down (to negative infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsqrt\_rn (float x)**

Compute  $\sqrt{x}$  in round-to-nearest-even mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-to-nearest-even mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsqrt\_ru (float x)**

Compute  $\sqrt{x}$  in round-up mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-up (to positive infinity) mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsqrt\_rz (float x)**

Compute  $\sqrt{x}$  in round-towards-zero mode.

### Returns

Returns  $\sqrt{x}$ .



**Description**

Compute the square root of  $x$  in round-towards-zero mode.



For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsub\_rd (float x, float y)**

Subtract two floating point values in round-down mode.

**Returns**

Returns  $x - y$ .

**Description**

Compute the difference of  $x$  and  $y$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_fsub\_rn (float x, float y)**

Subtract two floating point values in round-to-nearest-even mode.

**Returns**

Returns  $x - y$ .

**Description**

Compute the difference of  $x$  and  $y$  in round-to-nearest-even rounding mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- ▶ This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fsub_ru (float x, float y)`

Subtract two floating point values in round-up mode.

### Returns

Returns  $x - y$ .

### Description

Compute the difference of  $x$  and  $y$  in round-up (to positive infinity) mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __fsub_rz (float x, float y)`

Subtract two floating point values in round-towards-zero mode.

### Returns

Returns  $x - y$ .

### Description

Compute the difference of  $x$  and  $y$  in round-towards-zero mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 6.
- This operation will never be merged into a single multiply-add instruction.

## `__DEVICE_FUNCTIONS_DECL__ float __log10f (float x)`

Calculate the fast approximate base 10 logarithm of the input argument.

### Returns

Returns an approximation to  $\log_{10}(x)$ .

**Description**

Calculate the fast approximate base 10 logarithm of the input argument  $x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Most input and output values around denormal range are flushed to sign preserving 0.0.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_log2f (float x)**

Calculate the fast approximate base 2 logarithm of the input argument.

**Returns**

Returns an approximation to  $\log_2(x)$ .

**Description**

Calculate the fast approximate base 2 logarithm of the input argument  $x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Input and output in the denormal range is flushed to sign preserving 0.0.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_logf (float x)**

Calculate the fast approximate base  $e$  logarithm of the input argument.

**Returns**

Returns an approximation to  $\log_e(x)$ .

**Description**

Calculate the fast approximate base  $e$  logarithm of the input argument  $x$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Most input and output values around denormal range are flushed to sign preserving 0.0.

## `__DEVICE_FUNCTIONS_DECL__ float __powf (float x, float y)`

Calculate the fast approximate of  $x^y$ .

### Returns

Returns an approximation to  $x^y$ .

### Description

Calculate the fast approximate of  $x$ , the first input argument, raised to the power of  $y$ , the second input argument,  $x^y$ .



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Most input and output values around denormal range are flushed to sign preserving 0.0.

## `__DEVICE_FUNCTIONS_DECL__ float __saturatef (float x)`

Clamp the input argument to  $[+0.0, 1.0]$ .

### Returns

- ▶ `__saturatef(x)` returns 0 if  $x < 0$ .
- ▶ `__saturatef(x)` returns 1 if  $x > 1$ .
- ▶ `__saturatef(x)` returns  $x$  if  $0 \leq x \leq 1$ .
- ▶ `__saturatef(NaN)` returns 0.

### Description

Clamp the input argument  $x$  to be within the interval  $[+0.0, 1.0]$ .

## `__DEVICE_FUNCTIONS_DECL__ void __sincosf (float x, float *sptr, float *cptr)`

Calculate the fast approximate of sine and cosine of the first input argument.

### Returns

- ▶ none

## Description

Calculate the fast approximate of sine and cosine of the first input argument  $x$  (measured in radians). The results for sine and cosine are written into the second argument, `sptr`, and, respectively, third argument, `cptr`.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Denorm input/output is flushed to sign preserving 0.0.

## \_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_sinf (float x)

Calculate the fast approximate sine of the input argument.

## Returns

Returns the approximate sine of  $x$ .

## Description

Calculate the fast approximate sine of the input argument  $x$ , measured in radians.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ Input and output in the denormal range is flushed to sign preserving 0.0.

## \_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_tanf (float x)

Calculate the fast approximate tangent of the input argument.

## Returns

Returns the approximate tangent of  $x$ .

## Description

Calculate the fast approximate tangent of the input argument  $x$ , measured in radians.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.2, Table 9.
- ▶ The result is computed as the fast divide of `__sinf()` by `__cosf()`. Denormal input and output are flushed to sign-preserving 0.0 at each step of the computation.

## 1.5. Double Precision Ininsics

This section describes double precision intrinsic functions that are only supported in device code.

### `__device__ double __ddiv_rd (double x, double y)`

Divide two floating point values in round-down mode.

#### Returns

Returns  $x / y$ .

#### Description

Divides two floating point values  $x$  by  $y$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

### `__device__ double __ddiv_rn (double x, double y)`

Divide two floating point values in round-to-nearest-even mode.

#### Returns

Returns  $x / y$ .

#### Description

Divides two floating point values  $x$  by  $y$  in round-to-nearest-even mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

### `__device__ double __ddiv_ru (double x, double y)`

Divide two floating point values in round-up mode.

#### Returns

Returns  $x / y$ .

**Description**

Divides two floating point values  $x$  by  $y$  in round-up (to positive infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

**\_\_device\_\_ double \_\_ddiv\_rz (double x, double y)**

Divide two floating point values in round-towards-zero mode.

**Returns**

Returns  $x / y$ .

**Description**

Divides two floating point values  $x$  by  $y$  in round-towards-zero mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

**\_\_device\_\_ double \_\_drcp\_rd (double x)**

Compute  $\frac{1}{x}$  in round-down mode.

**Returns**

Returns  $\frac{1}{x}$ .

**Description**

Compute the reciprocal of  $x$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## `__device__ double __drcp_rn (double x)`

Compute  $\frac{1}{x}$  in round-to-nearest-even mode.

### Returns

Returns  $\frac{1}{x}$ .

### Description

Compute the reciprocal of  $x$  in round-to-nearest-even mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- Requires compute capability  $\geq 2.0$ .

## `__device__ double __drcp_ru (double x)`

Compute  $\frac{1}{x}$  in round-up mode.

### Returns

Returns  $\frac{1}{x}$ .

### Description

Compute the reciprocal of  $x$  in round-up (to positive infinity) mode.



- For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- Requires compute capability  $\geq 2.0$ .

## `__device__ double __drcp_rz (double x)`

Compute  $\frac{1}{x}$  in round-towards-zero mode.

### Returns

Returns  $\frac{1}{x}$ .

### Description

Compute the reciprocal of  $x$  in round-towards-zero mode.





- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## `__device__ double __dsqrt_rd (double x)`

Compute  $\sqrt{x}$  in round-down mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-down (to negative infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## `__device__ double __dsqrt_rn (double x)`

Compute  $\sqrt{x}$  in round-to-nearest-even mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-to-nearest-even mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## `__device__ double __dsqrt_ru (double x)`

Compute  $\sqrt{x}$  in round-up mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-up (to positive infinity) mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## `__device__ double __dsqrt_rz (double x)`

Compute  $\sqrt{x}$  in round-towards-zero mode.

### Returns

Returns  $\sqrt{x}$ .

### Description

Compute the square root of  $x$  in round-towards-zero mode.



- ▶ For accuracy information for this function see the CUDA C Programming Guide, Appendix D.1, Table 7.
- ▶ Requires compute capability  $\geq 2.0$ .

## 1.6. Integer Intrinsics

This section describes integer intrinsic functions that are only supported in device code.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __brev (unsigned int x)`

Reverse the bit order of a 32 bit unsigned integer.

### Returns

Returns the bit-reversed value of  $x$ . i.e. bit  $N$  of the return value corresponds to bit  $31-N$  of  $x$ .

### Description

Reverses the bit order of the 32 bit unsigned integer  $x$ .

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __brevll (unsigned long long int x)`

Reverse the bit order of a 64 bit unsigned integer.

### Returns

Returns the bit-reversed value of  $x$ . i.e. bit  $N$  of the return value corresponds to bit  $63-N$  of  $x$ .

### Description

Reverses the bit order of the 64 bit unsigned integer  $x$ .

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __byte_perm (unsigned int x, unsigned int y, unsigned int s)`

Return selected bytes from two 32 bit unsigned integers.

### Returns

The returned value  $r$  is computed to be: `result[n] := input[selector[n]]` where `result[n]` is the  $n$ th byte of  $r$ .

### Description

`byte_perm(x,y,s)` returns a 32-bit integer consisting of four bytes from eight input bytes provided in the two input integers  $x$  and  $y$ , as specified by a selector,  $s$ .

The input bytes are indexed as follows: `input[0] = x<7:0>` `input[1] = x<15:8>` `input[2] = x<23:16>` `input[3] = x<31:24>` `input[4] = y<7:0>` `input[5] = y<15:8>` `input[6] = y<23:16>` `input[7] = y<31:24>` The selector indices are as follows (the upper 16-bits of the selector

are not used): selector[0] = s<2:0> selector[1] = s<6:4> selector[2] = s<10:8> selector[3] = s<14:12>

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_clz (int x)**

Return the number of consecutive high-order zero bits in a 32 bit integer.

### **Returns**

Returns a value between 0 and 32 inclusive representing the number of zero bits.

### **Description**

Count the number of consecutive leading zero bits, starting at the most significant bit (bit 31) of x.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_clzll (long long int x)**

Count the number of consecutive high-order zero bits in a 64 bit integer.

### **Returns**

Returns a value between 0 and 64 inclusive representing the number of zero bits.

### **Description**

Count the number of consecutive leading zero bits, starting at the most significant bit (bit 63) of x.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_ffs (int x)**

Find the position of the least significant bit set to 1 in a 32 bit integer.

### **Returns**

Returns a value between 0 and 32 inclusive representing the position of the first bit set.

- `__ffs(0)` returns 0.

### **Description**

Find the position of the first (least significant) bit set to 1 in x, where the least significant bit position is 1.

## `__DEVICE_FUNCTIONS_DECL__ int __ffsll (long long int x)`

Find the position of the least significant bit set to 1 in a 64 bit integer.

### Returns

Returns a value between 0 and 64 inclusive representing the position of the first bit set.

- ▶ `__ffsll(0)` returns 0.

### Description

Find the position of the first (least significant) bit set to 1 in  $x$ , where the least significant bit position is 1.

## `__DEVICE_FUNCTIONS_DECL__ int __hadd (int, int)`

Compute average of signed input arguments, avoiding overflow in the intermediate sum.

### Returns

Returns a signed integer value representing the signed average value of the two inputs.

### Description

Compute average of signed input arguments  $x$  and  $y$  as  $(x + y) \gg 1$ , avoiding overflow in the intermediate sum.

## `__DEVICE_FUNCTIONS_DECL__ int __mul24 (int x, int y)`

Calculate the least significant 32 bits of the product of the least significant 24 bits of two integers.

### Returns

Returns the least significant 32 bits of the product  $x * y$ .

### Description

Calculate the least significant 32 bits of the product of the least significant 24 bits of  $x$  and  $y$ . The high order 8 bits of  $x$  and  $y$  are ignored.

## `__DEVICE_FUNCTIONS_DECL__ long long int __mul64hi (long long int x, long long int y)`

Calculate the most significant 64 bits of the product of the two 64 bit integers.

### Returns

Returns the most significant 64 bits of the product  $x * y$ .

### Description

Calculate the most significant 64 bits of the 128-bit product  $x * y$ , where  $x$  and  $y$  are 64-bit integers.

## `__DEVICE_FUNCTIONS_DECL__ int __mulhi (int x, int y)`

Calculate the most significant 32 bits of the product of the two 32 bit integers.

### Returns

Returns the most significant 32 bits of the product  $x * y$ .

### Description

Calculate the most significant 32 bits of the 64-bit product  $x * y$ , where  $x$  and  $y$  are 32-bit integers.

## `__DEVICE_FUNCTIONS_DECL__ int __popc (unsigned int x)`

Count the number of bits that are set to 1 in a 32 bit integer.

### Returns

Returns a value between 0 and 32 inclusive representing the number of set bits.

### Description

Count the number of bits that are set to 1 in  $x$ .

## `__DEVICE_FUNCTIONS_DECL__ int __popc1l (unsigned long long int x)`

Count the number of bits that are set to 1 in a 64 bit integer.

### Returns

Returns a value between 0 and 64 inclusive representing the number of set bits.

**Description**

Count the number of bits that are set to 1 in  $x$ .

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_rhadd (int, int)**

Compute rounded average of signed input arguments, avoiding overflow in the intermediate sum.

**Returns**

Returns a signed integer value representing the signed rounded average value of the two inputs.

**Description**

Compute average of signed input arguments  $x$  and  $y$  as  $(x + y + 1) \gg 1$ , avoiding overflow in the intermediate sum.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_sad (int x, int y, unsigned int z)**

Calculate  $|x - y| + z$ , the sum of absolute difference.

**Returns**

Returns  $|x - y| + z$ .

**Description**

Calculate  $|x - y| + z$ , the 32-bit sum of the third argument  $z$  plus and the absolute value of the difference between the first argument,  $x$ , and second argument,  $y$ .

Inputs  $x$  and  $y$  are signed 32-bit integers, input  $z$  is a 32-bit unsigned integer.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_uhadd (unsigned int, unsigned int)**

Compute average of unsigned input arguments, avoiding overflow in the intermediate sum.

**Returns**

Returns an unsigned integer value representing the unsigned average value of the two inputs.

**Description**

Compute average of unsigned input arguments  $x$  and  $y$  as  $(x + y) \gg 1$ , avoiding overflow in the intermediate sum.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_umul24 (unsigned int x, unsigned int y)**

Calculate the least significant 32 bits of the product of the least significant 24 bits of two unsigned integers.

**Returns**

Returns the least significant 32 bits of the product  $x * y$ .

**Description**

Calculate the least significant 32 bits of the product of the least significant 24 bits of  $x$  and  $y$ . The high order 8 bits of  $x$  and  $y$  are ignored.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned long long int \_\_umul64hi (unsigned long long int x, unsigned long long int y)**

Calculate the most significant 64 bits of the product of the two 64 unsigned bit integers.

**Returns**

Returns the most significant 64 bits of the product  $x * y$ .

**Description**

Calculate the most significant 64 bits of the 128-bit product  $x * y$ , where  $x$  and  $y$  are 64-bit unsigned integers.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_umulhi (unsigned int x, unsigned int y)**

Calculate the most significant 32 bits of the product of the two 32 bit unsigned integers.

**Returns**

Returns the most significant 32 bits of the product  $x * y$ .



**Description**

Calculate the most significant 32 bits of the 64-bit product  $x * y$ , where  $x$  and  $y$  are 32-bit unsigned integers.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_urhadd  
(unsigned int, unsigned int)**

Compute rounded average of unsigned input arguments, avoiding overflow in the intermediate sum.

**Returns**

Returns an unsigned integer value representing the unsigned rounded average value of the two inputs.

**Description**

Compute average of unsigned input arguments  $x$  and  $y$  as  $(x + y + 1) >> 1$ , avoiding overflow in the intermediate sum.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_usad  
(unsigned int x, unsigned int y, unsigned int z)**

Calculate  $|x - y| + z$ , the sum of absolute difference.

**Returns**

Returns  $|x - y| + z$ .

**Description**

Calculate  $|x - y| + z$ , the 32-bit sum of the third argument  $z$  plus and the absolute value of the difference between the first argument,  $x$ , and second argument,  $y$ .

Inputs  $x$ ,  $y$ , and  $z$  are unsigned 32-bit integers.

## 1.7. Type Casting Ininsics

This section describes type casting intrinsic functions that are only supported in device code.

## `__DEVICE_FUNCTIONS_DECL__ int __double2int_rz(double)`

Convert a double to a signed int in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the double-precision floating point value  $x$  to a signed integer value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ long long int __double2ll_rz(double)`

Convert a double to a signed 64-bit int in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the double-precision floating point value  $x$  to a signed 64-bit integer value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __double2uint_rz(double)`

Convert a double to an unsigned int in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the double-precision floating point value  $x$  to an unsigned integer value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __double2ull_rz (double)`

Convert a double to an unsigned 64-bit int in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the double-precision floating point value  $x$  to an unsigned 64-bit integer value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned short __float2half_rn (float x)`

Convert a single-precision float to a half-precision float in round-to-nearest-even mode.

### Returns

Returns converted value.

### Description

Convert the single-precision float value  $x$  to a half-precision floating point value represented in `unsigned short` format, in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ int __float2int_rd (float x)`

Convert a float to a signed integer in round-down mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to a signed integer in round-down (to negative infinity) mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_float2int\_rn (float x)**

Convert a float to a signed integer in round-to-nearest-even mode.

#### Returns

Returns converted value.

#### Description

Convert the single-precision floating point value  $x$  to a signed integer in round-to-nearest-even mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_float2int\_ru (float)**

Convert a float to a signed integer in round-up mode.

#### Returns

Returns converted value.

#### Description

Convert the single-precision floating point value  $x$  to a signed integer in round-up (to positive infinity) mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_float2int\_rz (float x)**

Convert a float to a signed integer in round-towards-zero mode.

#### Returns

Returns converted value.

#### Description

Convert the single-precision floating point value  $x$  to a signed integer in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ long long int __float2ll_rd (float x)`

Convert a float to a signed 64-bit integer in round-down mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to a signed 64-bit integer in round-down (to negative infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ long long int __float2ll_rn (float x)`

Convert a float to a signed 64-bit integer in round-to-nearest-even mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to a signed 64-bit integer in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ long long int __float2ll_ru (float x)`

Convert a float to a signed 64-bit integer in round-up mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to a signed 64-bit integer in round-up (to positive infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ long long int __float2ll_rz (float x)`

Convert a float to a signed 64-bit integer in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to a signed 64-bit integer in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __float2uint_rd (float x)`

Convert a float to an unsigned integer in round-down mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to an unsigned integer in round-down (to negative infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __float2uint_rn (float x)`

Convert a float to an unsigned integer in round-to-nearest-even mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to an unsigned integer in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __float2uint_ru (float x)`

Convert a float to an unsigned integer in round-up mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to an unsigned integer in round-up (to positive infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __float2uint_rz (float x)`

Convert a float to an unsigned integer in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to an unsigned integer in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __float2ull_rd (float x)`

Convert a float to an unsigned 64-bit integer in round-down mode.

### Returns

Returns converted value.

### Description

Convert the single-precision floating point value  $x$  to an unsigned 64-bit integer in round-down (to negative infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __float2ull_rn (float x)`

Convert a float to an unsigned 64-bit integer in round-to-nearest-even mode.

### **Returns**

Returns converted value.

### **Description**

Convert the single-precision floating point value  $x$  to an unsigned 64-bit integer in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __float2ull_ru (float x)`

Convert a float to an unsigned 64-bit integer in round-up mode.

### **Returns**

Returns converted value.

### **Description**

Convert the single-precision floating point value  $x$  to an unsigned 64-bit integer in round-up (to positive infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ unsigned long long int __float2ull_rz (float x)`

Convert a float to an unsigned 64-bit integer in round-towards-zero mode.

### **Returns**

Returns converted value.

### **Description**

Convert the single-precision floating point value  $x$  to an unsigned 64-bit integer in round-towards-zero mode.



**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ int \_\_float\_as\_int (float x)**

Reinterpret bits in a float as a signed integer.

#### Returns

Returns reinterpreted value.

#### Description

Reinterpret the bits in the single-precision floating point value `x` as a signed integer.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_half2float (unsigned short x)**

Convert a half-precision float to a single-precision float in round-to-nearest-even mode.

#### Returns

Returns converted value.

#### Description

Convert the half-precision floating point value `x` represented in `unsigned short` format to a single-precision floating point value.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_int2float\_rd (int x)**

Convert a signed integer to a float in round-down mode.

#### Returns

Returns converted value.

#### Description

Convert the signed integer value `x` to a single-precision floating point value in round-down (to negative infinity) mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_int2float\_rn (int x)**

Convert a signed integer to a float in round-to-nearest-even mode.

#### Returns

Returns converted value.

#### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-to-nearest-even mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_int2float\_ru (int x)**

Convert a signed integer to a float in round-up mode.

#### Returns

Returns converted value.

#### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-up (to positive infinity) mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_int2float\_rz (int x)**

Convert a signed integer to a float in round-towards-zero mode.

#### Returns

Returns converted value.

#### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-towards-zero mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_int\_as\_float (int x)**

Reinterpret bits in an integer as a float.

#### Returns

Returns reinterpreted value.

#### Description

Reinterpret the bits in the signed integer value  $x$  as a single-precision floating point value.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_ll2float\_rd (long long int x)**

Convert a signed integer to a float in round-down mode.

#### Returns

Returns converted value.

#### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-down (to negative infinity) mode.

**\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_ll2float\_rn (long long int x)**

Convert a signed 64-bit integer to a float in round-to-nearest-even mode.

#### Returns

Returns converted value.

#### Description

Convert the signed 64-bit integer value  $x$  to a single-precision floating point value in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ float __ll2float_ru (long long int x)`

Convert a signed integer to a float in round-up mode.

### Returns

Returns converted value.

### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-up (to positive infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ float __ll2float_rz (long long int x)`

Convert a signed integer to a float in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the signed integer value  $x$  to a single-precision floating point value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ float __uint2float_rd (unsigned int x)`

Convert an unsigned integer to a float in round-down mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-down (to negative infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ float __uint2float_rn` (unsigned int x)

Convert an unsigned integer to a float in round-to-nearest-even mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ float __uint2float_ru` (unsigned int x)

Convert an unsigned integer to a float in round-up mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-up (to positive infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ float __uint2float_rz` (unsigned int x)

Convert an unsigned integer to a float in round-towards-zero mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-towards-zero mode.

## `__DEVICE_FUNCTIONS_DECL__ float __ull2float_rd (unsigned long long int x)`

Convert an unsigned integer to a float in round-down mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-down (to negative infinity) mode.

## `__DEVICE_FUNCTIONS_DECL__ float __ull2float_rn (unsigned long long int x)`

Convert an unsigned integer to a float in round-to-nearest-even mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-to-nearest-even mode.

## `__DEVICE_FUNCTIONS_DECL__ float __ull2float_ru (unsigned long long int x)`

Convert an unsigned integer to a float in round-up mode.

### Returns

Returns converted value.

### Description

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-up (to positive infinity) mode.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ float \_\_ull2float\_rz (unsigned long long int x)**

Convert an unsigned integer to a float in round-towards-zero mode.

### **Returns**

Returns converted value.

### **Description**

Convert the unsigned integer value  $x$  to a single-precision floating point value in round-towards-zero mode.

## 1.8. SIMD Intrinsics

This section describes SIMD intrinsic functions that are only supported in device code.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabs2 (unsigned int a)**

Computes per-halfword absolute value.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes, then computes absolute value for each of parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabs4 (unsigned int a)**

Computes per-byte absolute value.

### **Returns**

Returns computed value.

### **Description**

Splits argument by bytes. Computes absolute value of each byte. Result is stored as unsigned int.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsdiffs2 (unsigned int a, unsigned int b)**

Computes per-halfword sum of absolute difference of signed integer.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsdiffs4 (unsigned int a, unsigned int b)**

Computes per-byte absolute difference of signed integer.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsdiffu2 (unsigned int a, unsigned int b)**

Performs per-halfword absolute difference of unsigned integer computation:  $|a - b|$ .

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute difference. Result is stored as unsigned int and returned.



## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsdiffu4 (unsigned int a, unsigned int b)**

Computes per-byte absolute difference of unsigned integer.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute difference. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsss2 (unsigned int a)**

Computes per-halfword absolute value with signed saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes, then computes absolute value with signed saturation for each of parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vabsss4 (unsigned int a)**

Computes per-byte absolute value with signed saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of argument into 4 parts, each consisting of 1 byte, then computes absolute value with signed saturation for each of parts. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vadd2 (unsigned int a, unsigned int b)`

Performs per-halfword (un)signed addition, with wrap-around:  $a + b$ .

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs unsigned addition on corresponding parts. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vadd4 (unsigned int a, unsigned int b)`

Performs per-byte (un)signed addition.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs unsigned addition on corresponding parts. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vaddss2 (unsigned int a, unsigned int b)`

Performs per-halfword addition with signed saturation.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs addition with signed saturation on corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vaddss4 (unsigned int a, unsigned int b)**

Performs per-byte addition with signed saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs addition with signed saturation on corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vaddus2 (unsigned int a, unsigned int b)**

Performs per-halfword addition with unsigned saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes, then performs addition with unsigned saturation on corresponding parts.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vaddus4 (unsigned int a, unsigned int b)**

Performs per-byte addition with unsigned saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte, then performs addition with unsigned saturation on corresponding parts.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vavgs2 (unsigned int a, unsigned int b)**

Performs per-halfword signed rounded average computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. then computes signed rounded average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vavgs4 (unsigned int a, unsigned int b)**

Computes per-byte signed rounder average.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes signed rounded average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vavgu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned rounded average computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. then computes unsigned rounded average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vavg4 (unsigned int a, unsigned int b)**

Performs per-byte unsigned rounded average.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes unsigned rounded average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpeq2 (unsigned int a, unsigned int b)**

Performs per-halfword (un)signed comparison.

### **Returns**

Returns 0xffff computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if they are equal, and 0000 otherwise. For example `__vcmpeq2(0x1234aba5, 0x1234aba6)` returns 0xffff0000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpeq4 (unsigned int a, unsigned int b)**

Performs per-byte (un)signed comparison.

### **Returns**

Returns 0xff if a = b, else returns 0.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if they are equal, and 00 otherwise. For example `__vcmpeq4(0x1234aba5, 0x1234aba6)` returns 0xffffff00.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpges2 (unsigned int a, unsigned int b)**

Performs per-halfword signed comparison:  $a \geq b ? 0xffff : 0$ .

### **Returns**

Returns 0xffff if  $a \geq b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part  $\geq$  'b' part, and 0000 otherwise. For example `__vcmpges2(0x1234aba5, 0x1234aba6)` returns 0xffff0000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpges4 (unsigned int a, unsigned int b)**

Performs per-byte signed comparison.

### **Returns**

Returns 0xff if  $a \geq b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part  $\geq$  'b' part, and 00 otherwise. For example `__vcmpges4(0x1234aba5, 0x1234aba6)` returns 0xffffff00.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgeu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned comparison:  $a \geq b ? 0xffff : 0$ .

### **Returns**

Returns 0xffff if  $a \geq b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part  $\geq$  'b' part, and 0000 otherwise. For example `__vcmpgeu2(0x1234aba5, 0x1234aba6)` returns 0xffff0000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgeu4 (unsigned int a, unsigned int b)**

Performs per-byte unsigned comparison.

### **Returns**

Returns 0xff if a = b, else returns 0.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part >= 'b' part, and 00 otherwise. For example `__vcmpgeu4(0x1234aba5, 0x1234aba6)` returns 0xffffff00.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgts2 (unsigned int a, unsigned int b)**

Performs per-halfword signed comparison: a > b ? 0xffff : 0.

### **Returns**

Returns 0xffff if a > b, else returns 0.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part > 'b' part, and 0000 otherwise. For example `__vcmpgts2(0x1234aba5, 0x1234aba6)` returns 0x00000000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgts4 (unsigned int a, unsigned int b)**

Performs per-byte signed comparison.

### **Returns**

Returns 0xff if a > b, else returns 0.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part > 'b' part, and 00 otherwise. For example `__vcmpgts4(0x1234aba5, 0x1234aba6)` returns 0x00000000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgtu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned comparison:  $a > b ? 0xffff : 0$ .

### **Returns**

Returns 0xffff if  $a > b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part > 'b' part, and 0000 otherwise. For example `__vcmpgtu2(0x1234aba5, 0x1234aba6)` returns 0x00000000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmpgtu4 (unsigned int a, unsigned int b)**

Performs per-byte unsigned comparison.

### **Returns**

Returns 0xff if  $a > b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part > 'b' part, and 00 otherwise. For example `__vcmpgtu4(0x1234aba5, 0x1234aba6)` returns 0x00000000.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vcmples2 (unsigned int a, unsigned int b)**

Performs per-halfword signed comparison:  $a \leq b ? 0xffff : 0$ .

### **Returns**

Returns 0xffff if  $a \leq b$ , else returns 0.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part  $\leq$  'b' part, and 0000 otherwise. For example `__vcmples2(0x1234aba5, 0x1234aba6)` returns 0xffffffff.



## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmples4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 0xff if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part  $\leq$  'b' part, and 00 otherwise. For example `__vcmples4(0x1234aba5, 0x1234aba6)` returns 0xffffffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmplesu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned comparison:  $a \leq b ? 0xffff : 0$ .

### Returns

Returns 0xffff if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part  $\leq$  'b' part, and 0000 otherwise. For example `__vcmplesu2(0x1234aba5, 0x1234aba6)` returns 0xffffffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmplesu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 0xff if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part  $\leq$  'b' part, and 00 otherwise. For example `__vcmplesu4(0x1234aba5, 0x1234aba6)` returns 0xffffffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmplts2 (unsigned int a, unsigned int b)`

Performs per-halfword signed comparison:  $a < b ? 0xffff : 0$ .

### Returns

Returns 0xffff if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part < 'b' part, and 0000 otherwise. For example `__vcmplts2(0x1234aba5, 0x1234aba6)` returns 0x0000ffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmplts4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 0xff if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part < 'b' part, and 00 otherwise. For example `__vcmplts4(0x1234aba5, 0x1234aba6)` returns 0x000000ff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmpltu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned comparison:  $a < b ? 0xffff : 0$ .

### Returns

Returns 0xffff if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part < 'b' part, and 0000 otherwise. For example `__vcmpltu2(0x1234aba5, 0x1234aba6)` returns 0x0000ffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmpltu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 0xff if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part < 'b' part, and 00 otherwise. For example `__vcmpltu4(0x1234aba5, 0x1234aba6)` returns 0x000000ff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmpne2 (unsigned int a, unsigned int b)`

Performs per-halfword (un)signed comparison:  $a \neq b ? 0xffff : 0$ .

### Returns

Returns 0xffff if  $a \neq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts result is ffff if 'a' part  $\neq$  'b' part, and 0000 otherwise. For example `__vcmplts2(0x1234aba5, 0x1234aba6)` returns 0x0000ffff.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vcmpne4 (unsigned int a, unsigned int b)`

Performs per-byte (un)signed comparison.

### Returns

Returns 0xff if  $a \neq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts result is ff if 'a' part  $\neq$  'b' part, and 00 otherwise. For example `__vcmplts4(0x1234aba5, 0x1234aba6)` returns 0x000000ff.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vhaddu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned average computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. then computes unsigned average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vhaddu4 (unsigned int a, unsigned int b)**

Computes per-byte unsigned average.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. then computes unsigned average of corresponding parts. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vmaxs2 (unsigned int a, unsigned int b)**

Performs per-halfword signed maximum computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed maximum. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vmaxs4 (unsigned int a, unsigned int b)`

Computes per-byte signed maximum.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed maximum. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vmaxu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned maximum computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned maximum. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vmaxu4 (unsigned int a, unsigned int b)`

Computes per-byte unsigned maximum.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned maximum. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vmins2 (unsigned int a, unsigned int b)**

Performs per-halfword signed minimum computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes signed minimum. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vmins4 (unsigned int a, unsigned int b)**

Computes per-byte signed minimum.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes signed minimum. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vminu2 (unsigned int a, unsigned int b)**

Performs per-halfword unsigned minimum computation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes unsigned minimum. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vminu4 (unsigned int a, unsigned int b)**

Computes per-byte unsigned minimum.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes unsigned minimum. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vneg2 (unsigned int a)**

Computes per-halfword negation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vneg4 (unsigned int a)**

Performs per-byte negation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vnegss2` (unsigned int a)

Computes per-halfword negation with signed saturation.

### Returns

Returns computed value.

### Description

Splits 4 bytes of argument into 2 parts, each consisting of 2 bytes. For each part function computes negation. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vnegss4` (unsigned int a)

Performs per-byte negation with signed saturation.

### Returns

Returns computed value.

### Description

Splits 4 bytes of argument into 4 parts, each consisting of 1 byte. For each part function computes negation. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsads2` (unsigned int a, unsigned int b)

Performs per-halfword sum of absolute difference of signed.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts functions computes absolute difference and sum it up. Result is stored as unsigned int and returned.



## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsads4 (unsigned int a, unsigned int b)`

Computes per-byte sum of abs difference of signed.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts functions computes absolute difference and sum it up. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsadu2 (unsigned int a, unsigned int b)`

Computes per-halfword sum of abs diff of unsigned.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function computes absolute differences, and returns sum of those differences.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsadu4 (unsigned int a, unsigned int b)`

Computes per-byte sum af abs difference of unsigned.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function computes absolute differences, and returns sum of those differences.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vseteq2 (unsigned int a, unsigned int b)`

Performs per-halfword (un)signed comparison.

### Returns

Returns 1 if  $a = b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part == 'b' part. If both equalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vseteq4 (unsigned int a, unsigned int b)`

Performs per-byte (un)signed comparison.

### Returns

Returns 1 if  $a = b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part == 'b' part. If both equalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetges2 (unsigned int a, unsigned int b)`

Performs per-halfword signed comparison.

### Returns

Returns 1 if  $a \geq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\geq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetges4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 1 if  $a \geq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\geq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgeu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned minimum unsigned comparison.

### Returns

Returns 1 if  $a \geq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\geq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgeu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 1 if  $a \geq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\geq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgts2 (unsigned int a, unsigned int b)`

Performs per-halfword signed comparison.

### Returns

Returns 1 if  $a > b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgts4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 1 if  $a > b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgtu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned comparison.

### Returns

Returns 1 if  $a > b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetgtu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 1 if  $a > b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part > 'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetles2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned minimum computation.

### Returns

Returns 1 if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetles4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 1 if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetleu2 (unsigned int a, unsigned int b)`

Performs per-halfword signed comparison.

### Returns

Returns 1 if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetleu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 1 if  $a \leq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 part, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetlts2 (unsigned int a, unsigned int b)`

Performs per-halfword signed comparison.

### Returns

Returns 1 if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetlts4 (unsigned int a, unsigned int b)`

Performs per-byte signed comparison.

### Returns

Returns 1 if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetltu2 (unsigned int a, unsigned int b)`

Performs per-halfword unsigned comparison.

### Returns

Returns 1 if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetltu4 (unsigned int a, unsigned int b)`

Performs per-byte unsigned comparison.

### Returns

Returns 1 if  $a < b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\leq$  'b' part. If both inequalities are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetne2 (unsigned int a, unsigned int b)`

Performs per-halfword (un)signed comparison.

### Returns

Returns 1 if  $a \neq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts function performs comparison 'a' part  $\neq$  'b' part. If both conditions are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsetne4 (unsigned int a, unsigned int b)`

Performs per-byte (un)signed comparison.

### Returns

Returns 1 if  $a \neq b$ , else returns 0.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts function performs comparison 'a' part  $\neq$  'b' part. If both conditions are satisfied, function returns 1.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsub2 (unsigned int a, unsigned int b)`

Performs per-halfword (un)signed subtraction, with wrap-around.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts functions performs subtraction. Result is stored as unsigned int and returned.



## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vsub4 (unsigned int a, unsigned int b)**

Performs per-byte subtraction.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts functions performs subtraction. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vsubss2 (unsigned int a, unsigned int b)**

Performs per-halfword (un)signed subtraction, with signed saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts functions performs subtraction with signed saturation. Result is stored as unsigned int and returned.

## **\_\_DEVICE\_FUNCTIONS\_DECL\_\_ unsigned int \_\_vsubss4 (unsigned int a, unsigned int b)**

Performs per-byte subtraction with signed saturation.

### **Returns**

Returns computed value.

### **Description**

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts functions performs subtraction with signed saturation. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsubus2 (unsigned int a, unsigned int b)`

Performs per-halfword subtraction with unsigned saturation.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 2 parts, each consisting of 2 bytes. For corresponding parts functions performs subtraction with unsigned saturation. Result is stored as unsigned int and returned.

## `__DEVICE_FUNCTIONS_DECL__ unsigned int __vsubus4 (unsigned int a, unsigned int b)`

Performs per-byte subtraction with unsigned saturation.

### Returns

Returns computed value.

### Description

Splits 4 bytes of each argument into 4 parts, each consisting of 1 byte. For corresponding parts functions performs subtraction with unsigned saturation. Result is stored as unsigned int and returned.

## 1.9. Half Precision Intrinsics

This section describes half precision intrinsic functions that are only supported in device code.

### Half Arithmetic Functions

#### Half2 Arithmetic Functions

#### Half Comparison Functions

#### Half2 Comparison Functions

## Half Precision Conversion And Data Movement

### 1.9.1. Half Arithmetic Functions

Half Precision Intrinsics

`__CUDA_FP16_DECL__ __half __hadd (const __half a, const __half b)`

Performs `half` addition in round-to-nearest mode.

#### Returns

Returns the `half` result of adding `a` and `b`.

#### Description

Performs `half` addition of inputs `a` and `b`, in round-to-nearest mode.

`__CUDA_FP16_DECL__ __half __hadd_sat (const __half a, const __half b)`

Performs `half` addition in round-to-nearest mode, with saturation to `[0.0, 1.0]`.

#### Returns

Returns the `half` result of adding `a` and `b` with saturation.

#### Description

Performs `half` add of inputs `a` and `b`, in round-to-nearest mode, and clamps the result to range `[0.0, 1.0]`. NaN results are flushed to `+0.0`.

`__CUDA_FP16_DECL__ __half __hfma (const __half a, const __half b, const __half c)`

Performs `half` fused multiply-add in round-to-nearest mode.

#### Returns

Returns the `half` result of the fused multiply-add operation on `a`, `b`, and `c`.

#### Description

Performs `half` multiply on inputs `a` and `b`, then performs a `half` add of the result with `c`, rounding the result once in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half __hfma_sat (const __half a, const __half b, const __half c)`**

Performs `half` fused multiply-add in round-to-nearest mode, with saturation to [0.0, 1.0].

### Returns

Returns the `half` result of the fused multiply-add operation on `a`, `b`, and `c` with saturation.

### Description

Performs `half` multiply on inputs `a` and `b`, then performs a `half` add of the result with `c`, rounding the result once in round-to-nearest mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

**`__CUDA_FP16_DECL__ __half __hmul (const __half a, const __half b)`**

Performs `half` multiplication in round-to-nearest mode.

### Returns

Returns the `half` result of multiplying `a` and `b`.

### Description

Performs `half` multiplication of inputs `a` and `b`, in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half __hmul_sat (const __half a, const __half b)`**

Performs `half` multiplication in round-to-nearest mode, with saturation to [0.0, 1.0].

### Returns

Returns the `half` result of multiplying `a` and `b` with saturation.

### Description

Performs `half` multiplication of inputs `a` and `b`, in round-to-nearest mode, and clamps the result to range [0.0, 1.0]. NaN results are flushed to +0.0.

**`__CUDA_FP16_DECL__ __half __hneg (const __half a)`**

Negates input `half` number and returns the result.

### Returns

Returns negated `half` input `a`.

### Description

Negates input `half` number and returns the result.

**`__CUDA_FP16_DECL__ __half __hsub (const __half a, const __half b)`**

Performs `half` subtraction in round-to-nearest mode.

### Returns

Returns the `half` result of subtraction `b` from `a`.

### Description

Subtracts `half` input `b` from input `a` in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half __hsub_sat (const __half a, const __half b)`**

Performs `half` subtraction in round-to-nearest mode, with saturation to `[0.0, 1.0]`.

### Returns

Returns the `half` result of subtraction `b` from `a` with saturation.

### Description

Subtracts `half` input `b` from input `a` in round-to-nearest mode, and clamps the result to range `[0.0, 1.0]`. NaN results are flushed to `+0.0`.

## 1.9.2. Half2 Arithmetic Functions

Half Precision Intrinsics

**`__CUDA_FP16_DECL__ __half2 __hadd2 (const __half2 a, const __half2 b)`**

Performs `half2` vector addition in round-to-nearest mode.

### Returns

Returns the `half2` vector result of adding vectors `a` and `b`.

### Description

Performs `half2` vector add of inputs `a` and `b`, in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half2 __hadd2_sat (const __half2 a, const __half2 b)`**

Performs `half2` vector addition in round-to-nearest mode, with saturation to `[0.0, 1.0]`.

### Returns

Returns the `half2` vector result of adding vectors `a` and `b` with saturation.

### Description

Performs `half2` vector add of inputs `a` and `b`, in round-to-nearest mode, and clamps the results to range `[0.0, 1.0]`. NaN results are flushed to `+0.0`.

**`__CUDA_FP16_DECL__ __half2 __hfma2 (const __half2 a, const __half2 b, const __half2 c)`**

Performs `half2` vector fused multiply-add in round-to-nearest mode.

### Returns

Returns the `half2` vector result of the fused multiply-add operation on vectors `a`, `b`, and `c`.

### Description

Performs `half2` vector multiply on inputs `a` and `b`, then performs a `half2` vector add of the result with `c`, rounding the result once in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half2 __hfma2_sat (const __half2 a, const __half2 b, const __half2 c)`**

Performs `half2` vector fused multiply-add in round-to-nearest mode, with saturation to [0.0, 1.0].

### Returns

Returns the `half2` vector result of the fused multiply-add operation on vectors `a`, `b`, and `c` with saturation.

### Description

Performs `half2` vector multiply on inputs `a` and `b`, then performs a `half2` vector add of the result with `c`, rounding the result once in round-to-nearest mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

**`__CUDA_FP16_DECL__ __half2 __hmul2 (const __half2 a, const __half2 b)`**

Performs `half2` vector multiplication in round-to-nearest mode.

### Returns

Returns the `half2` vector result of multiplying vectors `a` and `b`.

### Description

Performs `half2` vector multiplication of inputs `a` and `b`, in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half2 __hmul2_sat (const __half2 a, const __half2 b)`**

Performs `half2` vector multiplication in round-to-nearest mode, with saturation to [0.0, 1.0].

### Returns

Returns the `half2` vector result of multiplying vectors `a` and `b` with saturation.

### Description

Performs `half2` vector multiplication of inputs `a` and `b`, in round-to-nearest mode, and clamps the results to range [0.0, 1.0]. NaN results are flushed to +0.0.

**`__CUDA_FP16_DECL__ __half2 __hneg2 (const __half2 a)`**

Negates both halves of the input `half2` number and returns the result.

### Returns

Returns `half2` number with both halves negated.

### Description

Negates both halves of the input `half2` number `a` and returns the result.

**`__CUDA_FP16_DECL__ __half2 __hsub2 (const __half2 a, const __half2 b)`**

Performs `half2` vector subtraction in round-to-nearest mode.

### Returns

Returns the `half2` vector result of subtraction vector `b` from `a`.

### Description

Subtracts `half2` input vector `b` from input vector `a` in round-to-nearest mode.

**`__CUDA_FP16_DECL__ __half2 __hsub2_sat (const __half2 a, const __half2 b)`**

Performs `half2` vector subtraction in round-to-nearest mode, with saturation to `[0.0, 1.0]`.

### Returns

Returns the `half2` vector result of subtraction vector `b` from `a` with saturation.

### Description

Subtracts `half2` input vector `b` from input vector `a` in round-to-nearest mode, and clamps the results to range `[0.0, 1.0]`. NaN results are flushed to `+0.0`.

## 1.9.3. Half Comparison Functions

Half Precision Intrinsics



**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_heq (const \_\_half a, const \_\_half b)**

Performs `half` if-equal comparison.

### Returns

Returns boolean result of if-equal comparison of a and b.

### Description

Performs `half` if-equal comparison of inputs a and b. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hequ (const \_\_half a, const \_\_half b)**

Performs `half` unordered if-equal comparison.

### Returns

Returns boolean result of unordered if-equal comparison of a and b.

### Description

Performs `half` if-equal comparison of inputs a and b. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hge (const \_\_half a, const \_\_half b)**

Performs `half` greater-equal comparison.

### Returns

Returns boolean result of greater-equal comparison of a and b.

### Description

Performs `half` greater-equal comparison of inputs a and b. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hgeu (const \_\_half a, const \_\_half b)**

Performs `half` unordered greater-equal comparison.

### Returns

Returns boolean result of unordered greater-equal comparison of a and b.

### Description

Performs `half` greater-equal comparison of inputs a and b. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hgt (const \_\_half a, const \_\_half b)**

Performs `half` greater-than comparison.

### Returns

Returns boolean result of greater-than comparison of `a` and `b`.

### Description

Performs `half` greater-than comparison of inputs `a` and `b`. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hgtu (const \_\_half a, const \_\_half b)**

Performs `half` unordered greater-than comparison.

### Returns

Returns boolean result of unordered greater-than comparison of `a` and `b`.

### Description

Performs `half` greater-than comparison of inputs `a` and `b`. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ int \_\_hisinf (const \_\_half a)**

Checks if the input `half` number is infinite.

### Returns

Returns -1 iff `a` is equal to negative infinity, 1 iff `a` is equal to positive infinity and 0 otherwise.

### Description

Checks if the input `half` number `a` is infinite.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hisnan (const \_\_half a)**

Determine whether `half` argument is a NaN.

### Returns

Returns boolean true iff argument is a NaN, boolean false otherwise.

### Description

Determine whether `half` value `a` is a NaN.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hle (const \_\_half a, const \_\_half b)**

Performs `half` less-equal comparison.

### Returns

Returns boolean result of less-equal comparison of `a` and `b`.

### Description

Performs `half` less-equal comparison of inputs `a` and `b`. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hleu (const \_\_half a, const \_\_half b)**

Performs `half` unordered less-equal comparison.

### Returns

Returns boolean result of unordered less-equal comparison of `a` and `b`.

### Description

Performs `half` less-equal comparison of inputs `a` and `b`. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hlt (const \_\_half a, const \_\_half b)**

Performs `half` less-than comparison.

### Returns

Returns boolean result of less-than comparison of `a` and `b`.

### Description

Performs `half` less-than comparison of inputs `a` and `b`. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hltn (const \_\_half a, const \_\_half b)**

Performs `half` unordered less-than comparison.

### Returns

Returns boolean result of unordered less-than comparison of `a` and `b`.

**Description**

Performs `half` less-than comparison of inputs `a` and `b`. NaN inputs generate true results.

`__CUDA_FP16_DECL__ bool __hne (const __half a, const __half b)`

Performs `half` not-equal comparison.

**Returns**

Returns boolean result of not-equal comparison of `a` and `b`.

**Description**

Performs `half` not-equal comparison of inputs `a` and `b`. NaN inputs generate false results.

`__CUDA_FP16_DECL__ bool __hneu (const __half a, const __half b)`

Performs `half` unordered not-equal comparison.

**Returns**

Returns boolean result of unordered not-equal comparison of `a` and `b`.

**Description**

Performs `half` not-equal comparison of inputs `a` and `b`. NaN inputs generate true results.

## 1.9.4. Half2 Comparison Functions

Half Precision Intrinsics

`__CUDA_FP16_DECL__ bool __hbeq2 (const __half2 a, const __half2 b)`

Performs `half2` vector if-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

**Returns**

Returns boolean true if both `half` results of if-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector if-equal comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

`__CUDA_FP16_DECL__ bool __hbequ2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered if-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

**Returns**

Returns boolean true if both `half` results of unordered if-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector if-equal comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` if-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

`__CUDA_FP16_DECL__ bool __hbge2 (const __half2 a, const __half2 b)`

Performs `half2` vector greater-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

**Returns**

Returns boolean true if both `half` results of greater-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector greater-equal comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hbgeu2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector unordered greater-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of unordered greater-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

### Description

Performs `half2` vector greater-equal comparison of inputs `a` and `b`. The bool result is set to true only if both `half` greater-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hbgt2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector greater-than comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of greater-than comparison of vectors `a` and `b` are true, boolean false otherwise.

### Description

Performs `half2` vector greater-than comparison of inputs `a` and `b`. The bool result is set to true only if both `half` greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hbgtu2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector unordered greater-than comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of unordered greater-than comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector greater-than comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` greater-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

`__CUDA_FP16_DECL__ bool __hble2 (const __half2 a, const __half2 b)`

Performs `half2` vector less-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

**Returns**

Returns boolean true if both `half` results of less-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector less-equal comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

`__CUDA_FP16_DECL__ bool __hbleu2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered less-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

**Returns**

Returns boolean true if both `half` results of unordered less-equal comparison of vectors `a` and `b` are true, boolean false otherwise.

**Description**

Performs `half2` vector less-equal comparison of inputs `a` and `b`. The `bool` result is set to true only if both `half` less-equal comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hblt2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector less-than comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of less-than comparison of vectors `a` and `b` are true, boolean false otherwise.

### Description

Performs `half2` vector less-than comparison of inputs `a` and `b`. The bool result is set to true only if both `half` less-than comparisons evaluate to true, or false otherwise. NaN inputs generate false results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hbltu2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector unordered less-than comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of unordered less-than comparison of vectors `a` and `b` are true, boolean false otherwise.

### Description

Performs `half2` vector less-than comparison of inputs `a` and `b`. The bool result is set to true only if both `half` less-than comparisons evaluate to true, or false otherwise. NaN inputs generate true results.

**\_\_CUDA\_FP16\_DECL\_\_ bool \_\_hbne2 (const \_\_half2 a, const \_\_half2 b)**

Performs `half2` vector not-equal comparison, and returns boolean true iff both `half` results are true, boolean false otherwise.

### Returns

Returns boolean true if both `half` results of not-equal comparison of vectors `a` and `b` are true, boolean false otherwise.



**Description**

Performs `half2` vector not-equal comparison of inputs `a` and `b`. The `bool` result is set to `true` only if both `half` not-equal comparisons evaluate to `true`, or `false` otherwise. NaN inputs generate `false` results.

`__CUDA_FP16_DECL__ bool __hneu2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered not-equal comparison, and returns boolean `true` iff both `half` results are `true`, boolean `false` otherwise.

**Returns**

Returns boolean `true` if both `half` results of unordered not-equal comparison of vectors `a` and `b` are `true`, boolean `false` otherwise.

**Description**

Performs `half2` vector not-equal comparison of inputs `a` and `b`. The `bool` result is set to `true` only if both `half` not-equal comparisons evaluate to `true`, or `false` otherwise. NaN inputs generate `true` results.

`__CUDA_FP16_DECL__ __half2 __heq2 (const __half2 a, const __half2 b)`

Performs `half2` vector if-equal comparison.

**Returns**

Returns the `half2` vector result of if-equal comparison of vectors `a` and `b`.

**Description**

Performs `half2` vector if-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for `true`, or 0.0 for `false`. NaN inputs generate `false` results.

`__CUDA_FP16_DECL__ __half2 __hequ2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered if-equal comparison.

**Returns**

Returns the `half2` vector result of unordered if-equal comparison of vectors `a` and `b`.

**Description**

Performs `half2` vector if-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

`__CUDA_FP16_DECL__ __half2 __hge2 (const __half2 a, const __half2 b)`

Performs `half2` vector greater-equal comparison.

**Returns**

Returns the `half2` vector result of greater-equal comparison of vectors `a` and `b`.

**Description**

Performs `half2` vector greater-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

`__CUDA_FP16_DECL__ __half2 __hgeu2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered greater-equal comparison.

**Returns**

Returns the `half2` vector result of unordered greater-equal comparison of vectors `a` and `b`.

**Description**

Performs `half2` vector greater-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

`__CUDA_FP16_DECL__ __half2 __hgt2 (const __half2 a, const __half2 b)`

Performs `half2` vector greater-than comparison.

**Returns**

Returns the `half2` vector result of greater-than comparison of vectors `a` and `b`.

**Description**

Performs `half2` vector greater-than comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

**`__CUDA_FP16_DECL__ __half2 __hgtu2 (const __half2 a, const __half2 b)`**

Performs `half2` vector unordered greater-than comparison.

### Returns

Returns the `half2` vector result of unordered greater-than comparison of vectors `a` and `b`.

### Description

Performs `half2` vector greater-than comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

**`__CUDA_FP16_DECL__ __half2 __hisnan2 (const __half2 a)`**

Determine whether `half2` argument is a NaN.

### Returns

Returns `half2` which has the corresponding `half` results set to 1.0 for true, or 0.0 for false.

### Description

Determine whether each half of input `half2` number `a` is a NaN.

**`__CUDA_FP16_DECL__ __half2 __hle2 (const __half2 a, const __half2 b)`**

Performs `half2` vector less-equal comparison.

### Returns

Returns the `half2` vector result of less-equal comparison of vectors `a` and `b`.

### Description

Performs `half2` vector less-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

`__CUDA_FP16_DECL__ __half2 __hleu2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered less-equal comparison.

### Returns

Returns the `half2` vector result of unordered less-equal comparison of vectors `a` and `b`.

### Description

Performs `half2` vector less-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

`__CUDA_FP16_DECL__ __half2 __hlt2 (const __half2 a, const __half2 b)`

Performs `half2` vector less-than comparison.

### Returns

Returns the `half2` vector result of less-than comparison of vectors `a` and `b`.

### Description

Performs `half2` vector less-than comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

`__CUDA_FP16_DECL__ __half2 __hltu2 (const __half2 a, const __half2 b)`

Performs `half2` vector unordered less-than comparison.

### Returns

Returns the `half2` vector result of unordered less-than comparison of vectors `a` and `b`.

### Description

Performs `half2` vector less-than comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

**`__CUDA_FP16_DECL__ __half2 __hne2 (const __half2 a, const __half2 b)`**

Performs `half2` vector not-equal comparison.

### Returns

Returns the `half2` vector result of not-equal comparison of vectors `a` and `b`.

### Description

Performs `half2` vector not-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate false results.

**`__CUDA_FP16_DECL__ __half2 __hneu2 (const __half2 a, const __half2 b)`**

Performs `half2` vector unordered not-equal comparison.

### Returns

Returns the `half2` vector result of unordered not-equal comparison of vectors `a` and `b`.

### Description

Performs `half2` vector not-equal comparison of inputs `a` and `b`. The corresponding `half` results are set to 1.0 for true, or 0.0 for false. NaN inputs generate true results.

## 1.9.5. Half Precision Conversion And Data Movement

Half Precision Intrinsics

**`__CUDA_FP16_DECL__ __half2 __float2half2_rn (const float2 a)`**

Converts both components of `float2` number to half precision in round-to-nearest mode and returns `half2` with converted values.

### Returns

Returns `half2` which has corresponding halves equal to the converted `float2` components.

### Description

Converts both components of `float2` to half precision in round-to-nearest mode and combines the results into one `half2` number. Low 16 bits of the return value correspond to `a.x` and high 16 bits of the return value correspond to `a.y`.

**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_float2half2\_rn (const float a)**

Converts input to half precision in round-to-nearest mode and populates both halves of `half2` with converted value.

**Returns**

Returns `half2` with both halves equal to the converted half precision number.

**Description**

Converts input `a` to half precision in round-to-nearest mode and populates both halves of `half2` with converted value.

**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_floats2half2\_rn (const float a, const float b)**

Converts both input floats to half precision in round-to-nearest mode and returns `half2` with converted values.

**Returns**

Returns `half2` which has corresponding halves equal to the converted input floats.

**Description**

Converts both input floats to half precision in round-to-nearest mode and combines the results into one `half2` number. Low 16 bits of the return value correspond to the input `a`, high 16 bits correspond to the input `b`.

**\_\_CUDA\_FP16\_DECL\_\_ float2 \_\_half22float2 (const \_\_half2 a)**

Converts both halves of `half2` to `float2` and returns the result.

**Returns**

Returns converted `float2`.

**Description**

Converts both halves of `half2` input `a` to `float2` and returns the result.

**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_half2half2 (const \_\_half a)**

Returns `half2` with both halves equal to the input value.

**Returns**

Returns `half2` with both halves equal to the input `a`.

**Description**

Returns `half2` number with both halves equal to the input `half` number.

**`__CUDA_FP16_DECL__ __half2 __halves2half2 (const __half a, const __half b)`**

Combines two `half` numbers into one `half2` number.

**Returns**

Returns `half2` number which has one half equal to `a` and the other to `b`.

**Description**

Combines two input `half` number `a` and `b` into one `half2` number. Input `a` is stored in low 16 bits of the return value, input `b` is stored in high 16 bits of the return value.

**`__CUDA_FP16_DECL__ float __high2float (const __half2 a)`**

Converts high 16 bits of `half2` to float and returns the result.

**Returns**

Returns high 16 bits of `a` converted to float.

**Description**

Converts high 16 bits of `half2` input `a` to 32 bit floating point number and returns the result.

**`__CUDA_FP16_DECL__ __half __high2half (const __half2 a)`**

Returns high 16 bits of `half2` input.

**Returns**

Returns `half` which contains high 16 bits of the input.

**Description**

Returns high 16 bits of `half2` input `a`.

**`__CUDA_FP16_DECL__ __half2 __high2half2 (const __half2 a)`**

Extracts high 16 bits from `half2` input.

**Returns**

Returns `half2` with both halves equal to high 16 bits from the input.

**Description**

Extracts high 16 bits from `half2` input `a` and returns a new `half2` number which has both halves equal to the extracted bits.

**`__CUDA_FP16_DECL__ __half2 __high2half2 (const __half2 a, const __half2 b)`**

Extracts high 16 bits from each of the two `half2` inputs and combines into one `half2` number.

**Returns**

Returns `half2` which contains high 16 bits from `a` and `b`.

**Description**

Extracts high 16 bits from each of the two `half2` inputs and combines into one `half2` number. High 16 bits from input `a` is stored in low 16 bits of the return value, high 16 bits from input `b` is stored in high 16 bits of the return value.

**`__CUDA_FP16_DECL__ float __low2float (const __half2 a)`**

Converts low 16 bits of `half2` to float and returns the result.

**Returns**

Returns low 16 bits of `a` converted to float.

**Description**

Converts low 16 bits of `half2` input `a` to 32 bit floating point number and returns the result.

**`__CUDA_FP16_DECL__ __half __low2half (const __half2 a)`**

Returns low 16 bits of `half2` input.

**Returns**

Returns `half` which contains low 16 bits of the input.

**Description**

Returns low 16 bits of `half2` input `a`.



**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_low2half2 (const \_\_half2 a)**

Extracts low 16 bits from `half2` input.

### Returns

Returns `half2` with both halves equal to low 16 bits from the input.

### Description

Extracts low 16 bits from `half2` input `a` and returns a new `half2` number which has both halves equal to the extracted bits.

**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_lowhigh2highlow (const \_\_half2 a)**

Swaps both halves of the `half2` input.

### Returns

Returns `half2` with halves swapped.

### Description

Swaps both halves of the `half2` input and returns a new `half2` number with swapped halves.

**\_\_CUDA\_FP16\_DECL\_\_ \_\_half2 \_\_lows2half2 (const \_\_half2 a, const \_\_half2 b)**

Extracts low 16 bits from each of the two `half2` inputs and combines into one `half2` number.

### Returns

Returns `half2` which contains low 16 bits from `a` and `b`.

### Description

Extracts low 16 bits from each of the two `half2` inputs and combines into one `half2` number. Low 16 bits from input `a` is stored in low 16 bits of the return value, low 16 bits from input `b` is stored in high 16 bits of the return value.

## **Notice**

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

## **Trademarks**

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## **Copyright**

© 2007-2015 NVIDIA Corporation. All rights reserved.