

The **OpenCL Environment** Specification

Version: 2.1

Document Revision: 3

Khronos OpenCL Working Group

Editors: Raun Krisch

INTRODUCTION.....	6
1. GLOSSARY.....	6
2. SPIR-V CONSUMPTION.....	6
2.1 Numerical Type Formats.....	6
2.2 Image Channel Order Mapping.....	6
2.3 Image Channel Data Type Mapping.....	7
2.4 Extensions	7
2.5 SPIR-V Capabilities	8
3. OPENCL 2.1 CONSUMPTION SPECIALIZATION.....	8
3.1 Full Profile	8
3.2 Embedded Profile	9
3.3 Image Channel Order	9
3.4 SPIR-V Module Validation Rules	10
3.5 Extensions	11
3.5.1 cl_khr_fp16	11
3.5.2 cl_khr_mipmap_image_write	11
3.5.3 cl_khr_mipmap_image_write	11
3.5.4 cl_khr_device_enqueue_local_arg_types	11
3.5.5 cles_khr_int64	11
3.5.6 cl_khr_gl_depth_images.....	12
3.5.7 cl_khr_int64_base_atomics / cl_khr_int64_extended_atomics	12
4. OPENCL 2.0 CONSUMPTION SPECIALIZATION.....	12
4.1 Full Profile	12
4.2 Embedded Profile	13
4.3 Image Channel Order	14
4.4 SPIR-V Module Validation Rules	14
4.5 Extensions	15
4.5.1 cl_khr_fp16	15
4.5.2 cl_khr_mipmap_image_write	15

4.5.3	cl_khr_mipmap_image_write	15
4.5.4	cl_khr_device_enqueue_local_arg_types	16
4.5.5	cles_khr_int64	16
4.5.6	cl_khr_gl_depth_images.....	16
4.5.7	cl_khr_int64_base_atomics / cl_khr_int64_extended_atomics	16
4.5.8	cl_khr_subgroups	17
5.	OPENCL 1.2 CONSUMPTION SPECIALIZATION.....	17
5.1	Full Profile	17
5.2	Embedded Profile	17
5.3	Image Channel Order	18
5.4	SPIR-V Module Validation Rules	18
5.5	Extensions	19
5.5.1	cl_khr_fp16	19
5.5.2	cl_khr_mipmap_image_write.....	19
5.5.3	cl_khr_mipmap_image_write.....	19
5.5.4	cles_khr_int64	20
5.5.5	cl_khr_gl_depth_images.....	20
5.5.6	cl_khr_int64_base_atomics	20
5.5.7	cl_khr_int64_extended_atomics.....	20
5.5.8	cl_khr_subgroups	21
6.	EXECUTION & MEMORY MODEL SPECIALIZATION FOR OPENCL 2.X	21
7.	EXECUTION & MEMORY MODEL SPECIALIZATIONS FOR OPENCL 1.X.....	22
8.	NUMERICAL COMPLIANCE.....	22
8.1	Rounding Modes.....	22
8.2	INF, NaN and Denormalized Numbers	23
8.3	Floating-Point Exceptions.....	23
8.4	Relative Error as ULPs.....	24
8.4.1	Relative Error – Full Profile	24
8.4.2	Relative Error – Embedded Profile	31
8.5	Edge Case Behavior.....	34
8.5.1	Additional Requirements Beyond C99 TC2.....	34
8.5.2	Changes to C99 TC2 Behavior.....	37
8.5.3	Edge Case Behavior in Flush To Zero Mode	37

Copyright (c) 2008-2015 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be re-formatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group web-site should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

Khronos, StreamInput, WebGL, COLLADA, OpenKODE, OpenVG, OpenWF, OpenSL ES, OpenMAX, OpenMAX AL, OpenMAX IL and OpenMAX DL are trademarks and WebCL is a certification mark of the Khronos Group Inc. OpenCL is a trademark of Apple Inc. and OpenGL and OpenML are registered trademarks and the OpenGL ES and OpenGL SC logos are trademarks of Silicon Graphics International used under license by Khronos. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

Acknowledgements

The OpenCL Environment specification is the result of the contributions of many people, representing a cross section of the desktop, hand-held, and embedded computer industry. Following is a partial list of the contributors, including the company that they represented at the time of their contribution:

Lee Howes, QUALCOMM

Introduction

OpenCL (Open Computing Language) is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms.

Parallel programs are expressed through kernels using the OpenCL-C language or SPIR-V intermediate binary modules. This document describes the execution environment an OpenCL 2.1 platform provides when executing kernels and modules.

1. Glossary

The terminology used throughout this document is fully described in section 2, Glossary, of the OpenCL 2.1 API specification and various sections of the SPIR-V 1.0 intermediate binary specification.

2. SPIR-V Consumption

SPIR-V modules are consumed by environments (e.g., an API, or implementation of an API), which need to support the features used by a SPIR-V module. The following section describes how an OpenCL 2.1 platform consumes a SPIR-V module.

2.1 Numerical Type Formats

In an OpenCL platform, floating point types are represented and stored using IEEE-754 semantics. All integer formats are represented and stored using 2's complement format.

2.2 Image Channel Order Mapping

The following table describes how the results of the SPIR-V **OpImageQueryOrder** instruction correspond to the OpenCL host API image channel order.

SPIR-V Image Channel Order	OpenCL Image Channel Order
R	CL_R
A	CL_A
RG	CL_RG
RA	CL_RA
RGB	CL_RGB
RGBA	CL_RGBA
BGRA	CL_BGRA
ARGB	CL_ARGB
Intensity	CL_INTENSITY

Luminance	CL_LUMINANCE
Rx	CL_Rx
RGx	CL_RGx
RGBx	CL_RGBx
Depth	CL_DEPTH
DepthStencil	CL_DEPTH_STENCIL
sRGBx	CL_sRGBx
sRGBA	CL_sRGBA
sBGRA	CL_sBGRA

Table 2.0 – Image Channel Order mapping

2.3 Image Channel Data Type Mapping

The following table describes how the results of the SPIR-V **OpImageQueryFormat** instruction correspond to the OpenCL host API image channel data type.

SPIR-V Image Channel Data Type	OpenCL Image Channel Data Type
SnormInt8	CL_SNORM_INT8
SnormInt16	CL_SNORM_INT16
Unorm8	CL_UNORM_INT8
UnormInt16	CL_UNORM_INT16
UnormShort565	CL_UNORM_SHORT_565
UnormShort555	CL_UNORM_SHORT_555
UnormInt101010	CL_UNORM_INT_101010
SignedInt8	CL_SIGNED_INT8
SignedInt16	CL_SIGNED_INT16
SingedInt32	CL_SIGNED_INT32
UnsignedInt8	CL_UNSIGNED_INT8
UnsignedInt16	CL_UNSIGNED_INT16
UnsingedInt32	CL_UNSIGNED_INT32
HalfFloat	CL_HALF_FLOAT
FLOAT	CL_FLOAT

Table 2.1 – Image Channel Data Type mapping

2.4 Extensions

Using the **OpExtension** SPIR-V instruction a module can require new semantics. Using an extension in a SPIR-V module is dependent on the extension name appearing in the host API CL_PLATFORM_EXTENSIONS or CL_DEVICE_EXTENSIONS string.

While the details of an extension are described in the extension’s specification document, the following describes how validation rules are changed for the cl_khr prefixed extensions.

A valid SPIR-V module that uses an extension must declare the extension using `OpExtension("cl_khr_")`.

2.5 SPIR-V Capabilities

Capabilities used by a particular SPIR-V module are declared early in the module using the `OpCapability` instruction.













The features a capability enables in a SPIR-V module are described in the SPIR-V 1.0 specification.

The following section describes the minimum set of SPIR-V capabilities that an OpenCL platform is required to support.

3. OpenCL 2.1 Consumption Specialization

3.1 Full Profile

An OpenCL 2.1 Full Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

-  Address
-  DeviceEnqueue
-  Float16Buffer
-  GenericPointer
-  Group
-  Int64
-  Int16
-  Int8
-  Kernel
-  Linkage
-  Pipes
-  Vector16

Furthermore, the following capabilities may be supported:

-  ImageBasic
-  Float64

If ImageBasic is supported then the following capabilities must also be supported:

-  ImageMipmap

- ✦ ImageReadWrite
- ✦ LiteralSampler

3.2 Embedded Profile

An OpenCL 2.1 Embedded Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

- ✦ Address
- ✦ DeviceEnqueue
- ✦ Float16Buffer
- ✦ GenericPointer
- ✦ Group
- ✦ Int16
- ✦ Int8
- ✦ Kernel
- ✦ Linkage
- ✦ LiteralSampler
- ✦ Pipes
- ✦ Vector16

Furthermore, the following capabilities may be supported:

- ✦ ImageBasic
- ✦ Int64

If ImageBasic is supported then the following capabilities must also be supported:

- ✦ ImageMipmap
- ✦ ImageReadWrite
- ✦ LiteralSampler

3.3 Image Channel Order

If the **ImageBasic** capability is supported by the platform, the following Image Channel Orders must be supported:

- ✦ R
- ✦ A
- ✦ RG
- ✦ RA
- ✦ RGB
- ✦ RGBA

- + BGRA
- + ARGB
- + Intensity
- + Luminance
- + Rx
- + RGx
- + RGBx
- + Depth
- + sRGBx
- + sRGBA
- + sBGRA

3.4 SPIR-V Module Validation Rules

While SPIR-V describes many requirements and limitations on how instructions may be used within a module, an environment may impose additional rules a module must adhere in order to be considered valid.

The following are a list of OpenCL specific validation rules for SPIR-V modules

- + The source language declared in **OpSource** must be “OpenCL.”
- + The execution mode declared in **OpEntryPoint** must be Kernel.
- + The **Addressing Mode** declared in **OpMemoryModel** can be **Physical32** or **Physical64**, and must match the OpenCL API host query `CL_DEVICE_ADDRESS_BITS` query.
- + The memory model declared in **OpMemoryModel** must be **OpenCL**.
- + **OpImageWrite** cannot include any optional Image Operands.
- + Only Lod 0 is allowed on any image access instruction.
- + Data sources to **OpAtomic*** instructions are limited to 32bit integers.
- + Functions passed to **OpEnqueueKernel** may only have **WorkgroupLocal** memory pointer parameters of **OpTypeVoid**.

Scope for execution is limited to:

- + **Workgroup**
- + **Subgroup**

Scope for memory is limited to:

- + **CrossDevice**
- + **Device**
- + **Workgroup**
- + **Invocation**

Scope for **OpAsyncGroupCopy** and **OpWaitGroupEvents** is limited to:

 **Workgroup**

3.5 Extensions

The following describe OpenCL 2.1 specializations for the valid `cl_khr` extensions.





3.5.1 `cl_khr_fp16`

On supporting devices, **OpExtension**("cl_khr_fp16") further enables the following SPIR-V capabilities:

 **Float16**

3.5.2 `cl_khr_mipmap_image_write`

On supporting devices, **OpExtension**("cl_khr_mipmap_image") allows the optional **Lod Image** operand on the following instructions to be non-0:

-  **OpImageSampleExplicitLod**
-  **OpImageFetch**
-  **OpImageRead**
-  **OpImageQuerySizeLod**

3.5.3 `cl_khr_mipmap_image_write`

On supporting devices, **OpExtension**("cl_khr_mipmap_image_write") allows **OpImageWrite** to have a non-0 **Lod Image** Operand.

3.5.4 `cl_khr_device_enqueue_local_arg_types`

On supporting devices "cl_khr_device_enqueue_local_arg_type" allows arguments to functions passed to **OpEnqueueKernel** to declare a pointer to any type in the **WorkgroupLocal** Storage class.

3.5.5 `cles_khr_int64`

On supporting devices, **OpExtension**("cles_khr_int64") further enables the following SPIR-V capabilities:

 **Int64**















3.5.6 cl_khr_gl_depth_images

On supporting devices, **OpExtension**("cl_khr_gl_depth_images") further enables the following SPIR-V Image Channel Order:

 DepthStencil

3.5.7 cl_khr_int64_base_atomics / cl_khr_int64_extended_atomics

On supporting devices, **OpExtension**("cl_khr_int64_base_atomics") and **OpExtension**("cl_khr_int64_extended_atomics") allows 64bit integer sources using the **WorkgroupLocalMemory** or **WorkgroupGlobalMemory** Storage class:

-  OpAtomicUMax
-  OpAtomicSMax
-  OpAtomicUMin
-  OpAtomicSMin
-  OpAtomicAnd
-  OpAtomicOr
-  OpAtomicXOr
-  OpAtomicIAdd
-  OpAtomicISub
-  OpAtomicIIncrement
-  OpAtomicIDecrement
-  OpAtomicExchange
-  OpAtomicCompareExchange
-  OpAtomicCompareExchangeWeak

When the **WorkgroupLocalMemory** Memory Semantic is used the Scope must be **Workgroup**.

All other encodings are undefined.

An OpenCL 2.1 platform must either support both `cl_khr_int64_base_atomics` and `cl_khr_int64_extended_atomics` or neither of these extensions. Only 1 of the extension strings need to be declared via **OpExtension**.

4. OpenCL 2.0 Consumption Specialization

The following sections contain OpenCL 2.0 specializations and additional validation rules for the consumption of SPIR-V.

4.1 Full Profile

An OpenCL 2.0 Full Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

- + Address
- + DeviceEnqueue
- + Float16Buffer
- + GenericPointer
- + Group
- + Int64
- + Int16
- + Int8
- + Kernel
- + Linkage
- + Pipes
- + Vector16

Furthermore, the following capabilities may be supported:

- + ImageBasic
- + Float64

If ImageBasic is supported then the following capabilities must also be supported:

- + ImageMipmap
- + ImageReadWrite
- + LiteralSampler

4.2 Embedded Profile

An OpenCL 2.0 Embedded Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

- + Address
- + DeviceEnqueue
- + Float16Buffer
- + GenericPointer
- + Group
- + Int16
- + Int8
- + Kernel
- + Linkage
- + LiteralSampler
- + Pipes
- + Vector16

Furthermore, the following capabilities may be supported:

- + ImageBasic
- + Int64

If ImageBasic is supported then the following capabilities must also be supported:

- ✚ ImageMipmap
- ✚ ImageReadWrite
- ✚ LiteralSampler

4.3 Image Channel Order

If the **ImageBasic** capability is supported by the platform, the following Image Channel Orders must be supported:

- ✚ R
- ✚ A
- ✚ RG
- ✚ RA
- ✚ RGB
- ✚ RGBA
- ✚ BGRA
- ✚ ARGB
- ✚ Intensity
- ✚ Luminance
- ✚ Rx
- ✚ RGx
- ✚ RGBx
- ✚ Depth
- ✚ sRGBx
- ✚ sRGBA
- ✚ sBGRA

4.4 SPIR-V Module Validation Rules

While SPIR-V describes many requirements and limitations on how instructions may be used within a module, an environment may impose additional rules a module must adhere in order to be considered valid.

The following are a list of OpenCL specific validation rules for SPIR-V modules

- ✚ The source language declared in **OpSource** must be “OpenCL.”
- ✚ The execution mode declared in **OpEntryPoint** must be Kernel.
- ✚ The **Addressing Mode** declared in **OpMemoryModel** can be **Physical32** or **Physical64**, and must match the OpenCL API host query CL_DEVICE_ADDRESS_BITS query.
- ✚ The memory model declared in **OpMemoryModel** must be **OpenCL**.
- ✚ **OpImageWrite** cannot include any optional Image Operands.

- ✚ Only Lod 0 is allowed on any image access instruction.
- ✚ Data sources to **OpAtomic*** instructions are limited to 32bit integers.
- ✚ Functions passed to **OpEnqueueKernel** may only have **WorkgroupLocal** memory pointer parameters of **OpTypeVoid**.

Scope for execution is limited to:

- ✚ **Workgroup**

Scope for memory is limited to:

- ✚ **CrossDevice**
- ✚ **Device**
- ✚ **Workgroup**
- ✚ **Invocation**

4.5 Extensions

The following describe OpenCL 2.0 specializations for the valid cl_khr extensions.

4.5.1 cl_khr_fp16

On supporting devices, **OpExtension**("cl_khr_fp16") further enables the following SPIR-V capabilities:

- ✚ Float16

4.5.2 cl_khr_mipmap_image_write

On supporting devices, **OpExtension**("cl_khr_mipmap_image") allows the optional **Lod** Image operand on the following instructions to be non-0:

- ✚ **OpImageSampleExplicitLod**
- ✚ **OpImageFetch**
- ✚ **OpImageRead**
- ✚ **OpImageQuerySizeLod**

4.5.3 cl_khr_mipmap_image_write

On supporting devices, **OpExtension**("cl_khr_mipmap_image_write") allows **OpImageWrite** to have a non-0 **Lod** Image Operand.

4.5.4 cl_khr_device_enqueue_local_arg_types

On supporting devices “cl_khr_device_enqueue_local_arg_type” allows arguments to functions passed to **OpEnqueueKernel** to declare a pointer to any type in the **WorkgroupLocal** Storage class.

4.5.5 cles_khr_int64

On supporting devices, **OpExtension**(“cles_khr_int64”) further enables the following SPIR-V capabilities:

 Int64















4.5.6 cl_khr_gl_depth_images

On supporting devices, **OpExtension**(“cl_khr_gl_depth_images”) further enables the following SPIR-V Image Channel Order:

 DepthStencil

4.5.7 cl_khr_int64_base_atomics / cl_khr_int64_extended_atomics

On supporting devices, **OpExtension**(“cl_khr_int64_base_atomics”) and **OpExtension**(“cl_khr_int64_extended_atomics”) allows 64bit integer sources using the **WorkgroupLocalMemory** or **WorkgroupGlobalMemory** Storage class:

-  OpAtomicUMax
-  OpAtomicSMax
-  OpAtomicUMin
-  OpAtomicSMin
-  OpAtomicAnd
-  OpAtomicOr
-  OpAtomicXOr
-  OpAtomicIAdd
-  OpAtomicISub
-  OpAtomicIIncrement
-  OpAtomicIDecrement
-  OpAtomicExchange
-  OpAtomicCompareExchange
-  OpAtomicCompareExchangeWeak

When the **WorkgroupLocalMemory** Memory Semantic is used the Scope must be **Workgroup**.

An OpenCL 2.0 platform must either support both cl_khr_int64_base_atomics and cl_khr_int64_extended_atomics or neither of these extensions. Only 1 of the extension strings need to be declared via OpExtension.

4.5.8 cl_khr_subgroups










On supporting devices, **OpExtension**("cl_khr_subgroups") allows the **Subgroup** execution **Scope** to be used on any instruction other than **OpAsyncGroupCopy** and **OpWaitGroupEvents**.

5. OpenCL 1.2 Consumption Specialization

The following sections contain OpenCL 1.2 specializations and additional validation rules for the consumption of SPIR-V.

5.1 Full Profile


An OpenCL 1.2 Full Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

-  Address
-  Float16Buffer
-  Group
-  Int64
-  Int16
-  Int8
-  Kernel
-  Linkage
-  Vector16

Furthermore, the following capabilities may be supported:







-  ImageBasic
-  Float64

If **ImageBasic** is supported then the following capabilities must also be supported:

-  LiteralSampler

5.2 Embedded Profile

An OpenCL 1.2 Embedded Profile platform is guaranteed to support, at least, the following SPIR-V capabilities:

-  Address
-  Float16Buffer
-  Group
-  Int16
-  Int8
-  Kernel

- ✚ Linkage
- ✚ LiteralSampler
- ✚ Vector16

Furthermore, the following capabilities may be supported:

- ✚ ImageBasic
- ✚ Int64

If **ImageBasic** is supported then the following capabilities must also be supported:

- ✚ LiteralSampler

5.3 Image Channel Order

If the **ImageBasic** capability is supported by the platform, the following Image Channel Orders must be supported:

- ✚ R
- ✚ A
- ✚ RG
- ✚ RA
- ✚ RGB
- ✚ RGBA
- ✚ BGRA
- ✚ ARGB
- ✚ Intensity
- ✚ Luminance
- ✚ Rx
- ✚ RGx
- ✚ RGBx

5.4 SPIR-V Module Validation Rules

While SPIR-V describes many requirements and limitations on how instructions may be used within a module, an environment may impose additional rules a module must adhere in order to be considered valid.

The following are a list of OpenCL specific validation rules for SPIR-V modules

- ✚ The source language declared in **OpSource** must be “OpenCL.”
- ✚ The execution mode declared in **OpEntryPoint** must be Kernel.

- ✚ The **Addressing Mode** declared in **OpMemoryModel** can be **Physical32** or **Physical64**, and must match the OpenCL API host query `CL_DEVICE_ADDRESS_BITS` query.
- ✚ The memory model declared in **OpMemoryModel** must be **OpenCL**.
- ✚ **OpImageWrite** cannot include any optional Image Operands.
- ✚ Only Lod 0 is allowed on any image access instruction.
- ✚ Data sources to **OpAtomic*** instructions are limited to 32bit integers.

Scope for execution is limited to:

- ✚ **Workgroup**

Scope for memory is limited to:

- ✚ **Device**

5.5 Extensions

The following describe OpenCL 1.2 specializations for the valid `cl_khr` extensions.

5.5.1 `cl_khr_fp16`

On supporting devices, **OpExtension**("cl_khr_fp16") further enables the following SPIR-V capabilities:

- ✚ Float16

5.5.2 `cl_khr_mipmap_image_write`

On supporting devices, **OpExtension**("cl_khr_mipmap_image") allows the optional **Lod** Image operand on the following instructions to be non-0:

- ✚ `OpImageSampleExplicitLod`
- ✚ `OpImageFetch`
- ✚ `OpImageRead`
- ✚ `OpImageQuerySizeLod`

5.5.3 `cl_khr_mipmap_image_write`

On supporting devices, **OpExtension**("cl_khr_mipmap_image_write") allows **OpImageWrite** to have a non-0 **Lod** Image Operand.


5.5.4 cles_khr_int64

On supporting devices, **OpExtension**("cles_khr_int64") further enables the following SPIR-V capabilities:

 Int64







5.5.5 cl_khr_gl_depth_images

On supporting devices, **OpExtension**("cl_khr_gl_depth_images") further enables the following SPIR-V Image Channel Order:

 DepthStencil

5.5.6 cl_khr_int64_base_atomics

On supporting devices, **OpExtension**("cl_khr_int64_base_atomics") allows the following atomic operations to accept 64bit integer type data sources using the **WorkgroupLocalMemory** or **WorkgroupGlobalMemory**:








 OpAtomicIAdd
 OpAtomicISub
 OpAtomicIIncrement
 OpAtomicIDecrement
 OpAtomicExchange
 OpAtomicCompareExchange

When the **WorkgroupLocalMemory** Memory Semantic is used the Scope must be **Workgroup**.

When the **WorkgroupGlobalMemory** Memory Semantic is used, the scope must be **Device**.

5.5.7 cl_khr_int64_extended_atomics

On supporting devices, **OpExtension**("cl_khr_int64_extended_atomics") allows the following atomic operations to accept 64bit integer type data sources using the **WorkgroupLocalMemory** or **WorkgroupGlobalMemory** Memory Semantics:

 OpAtomicUMax
 OpAtomicSMax
 OpAtomicUMin
 OpAtomicSMin
 OpAtomicAnd
 OpAtomicOr
 OpAtomicXOr

When the **WorkgroupLocalMemory** Memory Semantic is used the Scope must be **Workgroup**.

When the **WorkgroupGlobalMemory** Memory Semantic is used, the scope must be **Device**.

All other encodings are undefined.

5.5.8 cl_khr_subgroups

On supporting devices, **OpExtension**(“cl_khr_subgroups”) allows the **Subgroup** execution **Scope** to be used on any instruction other than **OpAsyncGroupCopy** and **OpWaitGroupEvents**.

6. Execution & Memory Model Specialization for OpenCL 2.x

In the OpenCL 2.x execution environments, the local-happens-before and global-happens-before orders are separate. Image memory takes part in neither local-happens-before nor global-happens-before.

All memory scopes have meaning for OpenCL 2.x.

Memory scopes applied to OpenCL 2.x barrier, fence and atomic operations apply with the following scopes:

OpenCL 2.x scope	SPIR-V scope
memory_scope_work_item	Invocation
memory_scope_sub_group	Subgroup
memory_scope_work_group	Workgroup
memory_scope_device	Device
memory_scope_all_svm_devices	CrossDevice

Table 6.0 – OpenCL Scope to SPIR-V scope mapping

Address spaces passed to OpenCL 2.x barrier and fence operations apply as follows:

OpenCL 2.x address space	SPIR-V Memory semantic
CLK_IMAGE_MEM_FENCE	ImageMemory
CLK_LOCAL_MEM_FENCE	WorkgroupLocalMemory
CLK_GLOBAL_MEM_FENCE	WorkgroupGlobalMemory

Table 6.1 – OpenCL Address Space to SPIR-V Memory Semantic

OpenCL work-group barrier operations map to SPIR-V **OpControlBarrier** with *Workgroup* execution scope. OpenCL work -group cross-lane operations map to SPIR-V **OpGroup*** with *Workgroup* execution scope.

OpenCL sub-group barrier operations map to SPIR-V **OpControlBarrier** with *Subgroup* execution scope. OpenCL sub-group cross-lane operations map to SPIR-V **OpGroup*** with *Subgroup* execution scope.

7. Execution & Memory Model Specializations for OpenCL 1.x

In the OpenCL 1.x execution environments, the local-happens-before and global-happens-before orders are separate. Image memory takes part in neither local-happens-before nor global-happens-before.

CrossDevice scope has no meaning for OpenCL 1.x.

Address spaces passed to OpenCL 1.x barrier and fence operations apply as follows:

OpenCL 2.x address space	SPIR-V Memory semantic
CLK_IMAGE_MEM_FENCE	ImageMemory
CLK_LOCAL_MEM_FENCE	WorkgroupLocalMemory
CLK_GLOBAL_MEM_FENCE	WorkgroupGlobalMemory

Table 7.0 – OpenCL Address Space to SPIR-V Memory Semantic mapping

OpenCL 1.x fence and barrier operations execute at *Workgroup* execution scope and *Workgroup* memory scope with sequentially consistent memory order.

OpenCL 1.x fence operations applied to image memory generate SPIR-V *Invocation* scope fences with *ImageMemory* semantic.

OpenCL 1.x atomic operations generate relaxed Device-scoped atomic operations.

8. Numerical Compliance

This section describes features of the C99 and IEEE 754 standards that must be supported by all OpenCL compliant devices.

This section describes the functionality that must be supported by all OpenCL devices for single precision floating-point numbers. Currently, only single precision floating-point is a requirement. Double precision floating-point is an optional feature.

8.1 Rounding Modes

Floating-point calculations may be carried out internally with extra precision and then rounded to fit into the destination type. IEEE 754 defines four possible rounding modes:

- ✚ Round to nearest even
- ✚ Round toward $+\infty$
- ✚ Round toward $-\infty$
- ✚ Round toward zero

Round to nearest even is currently the only rounding mode required¹ by the OpenCL specification for single precision and double precision operations and is therefore the default rounding mode. In addition, only static selection of rounding mode is supported. Dynamically reconfiguring the rounding modes as specified by the IEEE 754 spec is unsupported.

8.2 INF, NaN and Denormalized Numbers

INF and NaNs must be supported. Support for signaling NaNs is not required.

Support for denormalized numbers with single precision floating-point is optional. Denormalized single precision floating-point numbers passed as input or produced as the output of single precision floating-point operations such as add, sub, mul, divide, and the functions defined in *sections 6.11.2* (math functions), *6.11.4* (common functions) and *6.11.5* (geometric functions) may be flushed to zero.

8.3 Floating-Point Exceptions

Floating-point exceptions are disabled in OpenCL. The result of a floating-point exception must match the IEEE 754 spec for the exceptions not enabled case. Whether and when the implementation sets floating-point flags or raises floating-point exceptions is implementation-defined. This standard provides no method for querying, clearing or setting floating-point flags or trapping raised exceptions. Due to non-performance, non-portability of trap mechanisms and the impracticality of servicing precise exceptions in a vector context (especially on heterogeneous hardware), such features are discouraged.

Implementations that nevertheless support such operations through an extension to the standard shall initialize with all exception flags cleared and the exception masks set so that exceptions raised by arithmetic operations do not trigger a trap to be taken. If the underlying work is reused by the implementation, the implementation is however not responsible for reclearing the flags or resetting exception masks to default values before entering the kernel. That is to say that kernels that do not inspect flags or enable traps are licensed to expect that their arithmetic will not trigger a trap. Those kernels that do examine flags or enable traps are responsible for clearing flag state and disabling all traps before returning control to the implementation. Whether or when the underlying work-item (and accompanying global floating-point state if any) is reused is implementation-defined.

¹ Except for the embedded profile whether either round to zero or round to nearest rounding mode may be supported for single precision floating-point.

The expressions **math_errorhandling** and **MATH_ERREXCEPT** are reserved for use by this standard, but not defined. Implementations that extend this specification with support for floating-point exceptions shall define **math_errorhandling** and **MATH_ERREXCEPT** per ISO / IEC 9899 : TC2.

8.4 Relative Error as ULPs

In this section we discuss the maximum relative error defined as ulp (units in the last place). Addition, subtraction, multiplication, fused multiply-add and conversion between integer and a single precision floating-point format are IEEE 754 compliant and are therefore correctly rounded. Conversion between floating-point formats and explicit conversions specified in *section 6.2.3* must be correctly rounded.

The ULP is defined as follows:

If x is a real number that lies between two finite consecutive floating-point numbers a and b , without being equal to one of them, then $ulp(x) = |b - a|$, otherwise $ulp(x)$ is the distance between the two non-equal finite floating-point numbers nearest x . Moreover, $ulp(\text{NaN})$ is NaN.

Attribution: This definition was taken with consent from Jean-Michel Muller with slight clarification for behavior at zero. Refer to <ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-5504.pdf>.

8.4.1 Relative Error – Full Profile

Table 8.1² describes the minimum accuracy of floating-point arithmetic operations given as ULP values. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Function	Min Accuracy - ULP values ³ Float32/Float64	Min Accuracy - ULP values ⁴ Float16
$x + y$	Correctly rounded	Correctly rounded
$x - y$	Correctly rounded	Correctly rounded
$x * y$	Correctly rounded	Correctly rounded

² The ULP values for built-in math functions **lgamma** and **lgamma_r** is currently undefined.

³ 0 ulp is used for math functions that do not require rounding.

⁴ 0 ulp is used for math functions that do not require rounding.

1.0 / x	≤ 2.5 ulp	Correctly rounded
x / y	≤ 2.5 ulp	Correctly rounded
acos	≤ 4 ulp	≤ 2 ulp
acospi	≤ 5 ulp	≤ 2 ulp
asin	≤ 4 ulp	≤ 2 ulp
asinpi	≤ 5 ulp	≤ 2 ulp
atan	≤ 5 ulp	≤ 2 ulp
atan2	≤ 6 ulp	≤ 2 ulp
atanpi	≤ 5 ulp	≤ 2 ulp
atan2pi	≤ 6 ulp	≤ 2 ulp
acosh	≤ 4 ulp	≤ 2 ulp
asinh	≤ 4 ulp	≤ 2 ulp
atanh	≤ 5 ulp	≤ 2 ulp
cbrt	≤ 2 ulp	≤ 2 ulp
ceil	Correctly rounded	Correctly rounded
copysign	0 ulp	0 ulp
cos	≤ 4 ulp	≤ 2 ulp
cosh	≤ 4 ulp	≤ 2 ulp
cospi	≤ 4 ulp	≤ 2 ulp
erfc	≤ 16 ulp	≤ 4 ulp
erf	≤ 16 ulp	≤ 4 ulp
exp	≤ 3 ulp	≤ 2 ulp
exp2	≤ 3 ulp	≤ 2 ulp
exp10	≤ 3 ulp	≤ 2 ulp
expm1	≤ 3 ulp	≤ 2 ulp
fabs	0 ulp	0 ulp
fdim	Correctly rounded	Correctly rounded
floor	Correctly rounded	Correctly rounded
fma	Correctly rounded	Correctly rounded
fmax	0 ulp	0 ulp
fmin	0 ulp	0 ulp
fmod	0 ulp	0 ulp
fract	Correctly rounded	Correctly rounded
frexp	0 ulp	0 ulp
hypot	≤ 4 ulp	≤ 2 ulp
ilogb	0 ulp	0 ulp
ldexp	Correctly rounded	Correctly rounded
log	≤ 3 ulp	≤ 2 ulp
log2	≤ 3 ulp	≤ 2 ulp
log10	≤ 3 ulp	≤ 2 ulp
log1p	≤ 2 ulp	≤ 2 ulp
logb	0 ulp	0 ulp
mad	Implemented either as a correctly rounded fma or as a multiply	Any value allowed (infinite ulp)

	followed by an add both of which are correctly rounded	
maxmag	0 ulp	0 ulp
minmag	0 ulp	0 ulp
modf	0 ulp	0 ulp
nan	0 ulp	0 ulp
nextafter	0 ulp	0 ulp
pow(x, y)	≤ 16 ulp	≤ 4 ulp
pown(x, y)	≤ 16 ulp	≤ 4 ulp
powr(x, y)	≤ 16 ulp	≤ 4 ulp
remainder	0 ulp	0 ulp
remquo	0 ulp	0 ulp
rint	Correctly rounded	Correctly rounded
rootn	≤ 16 ulp	≤ 4 ulp
round	Correctly rounded	Correctly rounded
rsqrt	≤ 2 ulp	≤ 1 ulp
sin	≤ 4 ulp	≤ 2 ulp
sincos	≤ 4 ulp for sine and cosine values	≤ 2 ulp for sine and cosine values
sinh	≤ 4 ulp	≤ 2 ulp
sinpi	≤ 4 ulp	≤ 2 ulp
sqrt	≤ 3 ulp	Correctly rounded
tan	≤ 5 ulp	≤ 2 ulp
tanh	≤ 5 ulp	≤ 2 ulp
tanpi	≤ 6 ulp	≤ 2 ulp
tgamma	≤ 16 ulp	≤ 4 ulp
trunc	Correctly rounded	Correctly rounded
half_cos	≤ 8192 ulp	≤ 8192 ulp
half_divide	≤ 8192 ulp	≤ 8192 ulp
half_exp	≤ 8192 ulp	≤ 8192 ulp
half_exp2	≤ 8192 ulp	≤ 8192 ulp
half_exp10	≤ 8192 ulp	≤ 8192 ulp
half_log	≤ 8192 ulp	≤ 8192 ulp
half_log2	≤ 8192 ulp	≤ 8192 ulp
half_log10	≤ 8192 ulp	≤ 8192 ulp
half_powr	≤ 8192 ulp	≤ 8192 ulp
half_recip	≤ 8192 ulp	≤ 8192 ulp
half_rsqrt	≤ 8192 ulp	≤ 8192 ulp
half_sin	≤ 8192 ulp	≤ 8192 ulp
half_sqrt	≤ 8192 ulp	≤ 8192 ulp
half_tan	≤ 8192 ulp	≤ 8192 ulp
native_cos	Implementation-defined	Implementation-defined
native_divide	Implementation-defined	Implementation-defined

native_exp	Implementation-defined	Implementation-defined
native_exp2	Implementation-defined	Implementation-defined
native_exp10	Implementation-defined	Implementation-defined
native_log	Implementation-defined	Implementation-defined
native_log2	Implementation-defined	Implementation-defined
native_log10	Implementation-defined	Implementation-defined
native_powr	Implementation-defined	Implementation-defined
native_recip	Implementation-defined	Implementation-defined
native_rsqrt	Implementation-defined	Implementation-defined
native_sin	Implementation-defined	Implementation-defined
native_sqrt	Implementation-defined	Implementation-defined
native_tan	Implementation-defined	Implementation-defined

Table 8.1 *ULP values for single precision built-in math functions*

Table 8.2 describes the minimum accuracy of commonly used single precision floating-point arithmetic operations given as ULP values if the `-cl-fast-relaxed-math` compiler option is specified when compiling or building an OpenCL program or if the RelaxedPrecision SPIR-V decoration is used. The minimum accuracy of math functions not defined in table 7.2 when the `-cl-fast-relaxed-math` compiler option is specified or RelaxedPrecision decoration is as defined in table 8.1. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Function	Min Accuracy - ULP values ⁵
1.0 / x	≤ 2.5 ulp for x in the domain of 2^{-126} to 2^{126}
x / y	≤ 2.5 ulp for x in the domain of 2^{-62} to 2^{62} and y in the domain of 2^{-62} to 2^{62} .
acos(x)	≤ 4096 ulp
acospi(x)	Implemented as acos(x) * M_PI_F. For non-derived implementations, the error is ≤ 8192 ulp.
asin(x)	≤ 4096 ulp
asinpi(x)	Implemented as asin(x) * M_PI_F. For non-derived implementations, the error is ≤ 8192 ulp.
atan(x)	≤ 4096 ulp
atan2(y, x)	Implemented as atan(y/x) for $x > 0$, atan(y/x) + M_PI_F for $x < 0$ and $y > 0$ and atan(y/x) - M_PI_F for $x < 0$ and $y < 0$.
atanpi(x)	Implemented as atan(x) * M_PI_F. For non-derived implementations, the error is ≤ 8192 ulp.
atan2pi(y, x)	Implemented as atan2(y, x) * M_PI_F. For non-derived implementations, the error is ≤ 8192 ulp.
acosh(x)	Implemented as log(x + sqrt(x*x - 1)) .

⁵ 0 ulp is used for math functions that do not require rounding.

asinh(x)	Implemented as log(x + sqrt(x*x + 1)) .
cbirt(x)	Implemented as rootn(x, 3) . For non-derived implementations, the error is ≤ 8192 ulp.
cos(x)	For x in the domain $[-\pi, \pi]$, the maximum absolute error is $\leq 2^{-11}$ and larger otherwise.
cosh(x)	Implemented as 0.5 * exp(x) + exp(-x) . For non-derived implementations, the error is ≤ 8192 ulp.
cospi(x)	For x in the domain $[-1, 1]$, the maximum absolute error is $\leq 2^{-11}$ and larger otherwise.
exp(x)	$\leq 3 + \text{floor}(\text{fabs}(2 * x))$ ulp
exp2(x)	$\leq 3 + \text{floor}(\text{fabs}(2 * x))$ ulp
exp10(x)	Derived implementations implement this as exp2(x * log2(10)) . For non-derived implementations, the error is ≤ 8192 ulp.
expm1(x)	Derived implementations implement this as exp(x) - 1 . For non-derived implementations, the error is ≤ 8192 ulp.
log(x)	For x in the domain $[0.5, 2]$ the maximum absolute error is $\leq 2^{-21}$; otherwise the maximum error is ≤ 3 ulp for the full profile and ≤ 4 ulp for the embedded profile
log2(x)	For x in the domain $[0.5, 2]$ the maximum absolute error is $\leq 2^{-21}$; otherwise the maximum error is ≤ 3 ulp for the full profile and ≤ 4 ulp for the embedded profile
log10(x)	For x in the domain $[0.5, 2]$ the maximum absolute error is $\leq 2^{-21}$; otherwise the maximum error is ≤ 3 ulp for the full profile and ≤ 4 ulp for the embedded profile
log1p(x)	Derived implementations implement this as log(x + 1) . For non-derived implementations, the error is ≤ 8192 ulp.
pow(x, y)	Undefined for $x = 0$ and $y = 0$ or for $x < 0$ and non-integer y . For $x \geq 0$ or $x < 0$ and even y , derived implementations implement this as exp2(y * log2(x)) . For $x < 0$ and odd y , derived implementations implement this as -exp2(y * log2(fabs(x))) . For non-derived implementations, the error is ≤ 8192 ulp.
pown(x, y)	Defined only for integer values of y . Undefined for $x = 0$ and $y = 0$. For $x \geq 0$ or $x < 0$ and even y , derived implementations implement this as exp2(y * log2(x)) . For $x < 0$ and odd y , derived implementations implement this as -exp2(y * log2(fabs(x))) . For non-derived implementations, the error is ≤ 8192 ulp.

powr(x, y)	Defined only for $x \geq 0$. Undefined for $x = 0$ and $y = 0$. Derived implementations implement this as exp2(y * log2(x)) . For non-derived implementations, the error is ≤ 8192 ulp.
rootn(x, y)	Defined for $x > 0$ and for $x < 0$ when y is odd. Undefined for $x = 0$ and $y = 0$. Derived implementations implement this as exp2(log2(x) / y) for $x > 0$. Derived implementations implement this as -exp2(log2(-x) / y) for $x < 0$. For non-derived implementations, the error is ≤ 8192 ulp.
sin(x)	For x in the domain $[-\pi, \pi]$, the maximum absolute error is $\leq 2^{-11}$ and larger otherwise.
sincos(x)	ulp values as defined for sin(x) and cos(x)
sinh(x)	Implemented as 0.5 * exp(x) - exp(-x) . For non-derived implementations, the error is ≤ 8192 ulp.
sinpi(x)	For x in the domain $[-1, 1]$, the maximum absolute error is $\leq 2^{-11}$ and larger otherwise.
tan(x)	Derived implementations implement this as sin(x) * (1.0f / cos(x)) . For non-derived implementations, the error is ≤ 8192 ulp.
tanpi(x)	Derived implementations implement this as tan(x * M_PI_F) . For non-derived implementations, the error is ≤ 8192 ulp for x in the domain $[-1, 1]$.
x * y + z	Implemented either as a correctly rounded fma or as a multiply and an add both of which are correctly rounded.

Table 8.2 *ULP values for single precision built-in math functions with fast relaxed math*

Table 8.3 describes the minimum accuracy of double precision floating-point arithmetic operations given as ULP values. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Function	Min Accuracy - ULP values ⁶
x + y	Correctly rounded
x - y	Correctly rounded
x * y	Correctly rounded
1.0 / x	Correctly rounded
x / y	Correctly rounded
acos	≤ 4 ulp
acospi	≤ 5 ulp

⁶ 0 ulp is used for math functions that do not require rounding.

asin	<= 4 ulp
asinpi	<= 5 ulp
atan	<= 5 ulp
atan2	<= 6 ulp
atanpi	<= 5 ulp
atan2pi	<= 6 ulp
acosh	<= 4 ulp
asinh	<= 4 ulp
atanh	<= 5 ulp
cbrt	<= 2 ulp
ceil	Correctly rounded
copysign	0 ulp
cos	<= 4 ulp
cosh	<= 4 ulp
cospi	<= 4 ulp
erfc	<= 16 ulp
erf	<= 16 ulp
exp	<= 3 ulp
exp2	<= 3 ulp
exp10	<= 3 ulp
expm1	<= 3 ulp
fabs	0 ulp
fdim	Correctly rounded
floor	Correctly rounded
fma	Correctly rounded
fmax	0 ulp
fmin	0 ulp
fmod	0 ulp
fract	Correctly rounded
frexp	0 ulp
hypot	<= 4 ulp
ilogb	0 ulp
ldexp	Correctly rounded
log	<= 3 ulp
log2	<= 3 ulp
log10	<= 3 ulp
log1p	<= 2 ulp
logb	0 ulp
mad	Any value allowed (infinite ulp)
maxmag	0 ulp
minmag	0 ulp
modf	0 ulp
nan	0 ulp
nextafter	0 ulp
pow(x, y)	<= 16 ulp

pown(x, y)	<= 16 ulp
powr(x, y)	<= 16 ulp
remainder	0 ulp
remquo	0 ulp
rint	Correctly rounded
rootn	<= 16 ulp
round	Correctly rounded
rsqrt	<= 2 ulp
sin	<= 4 ulp
sincos	<= 4 ulp for sine and cosine values
sinh	<= 4 ulp
sinpi	<= 4 ulp
sqrt	Correctly rounded
tan	<= 5 ulp
tanh	<= 5 ulp
tanpi	<= 6 ulp
tgamma	<= 16 ulp
trunc	Correctly rounded

Table 8.3 *ULP values for double precision built-in math functions*

8.4.2 Relative Error – Embedded Profile

Table 8.4 describes the minimum accuracy of floating-point arithmetic operations given as ULP values for the embedded profile. The reference value used to compute the ULP value of an arithmetic operation is the infinitely precise result.

Function	Min Accuracy - ULP values⁷ Float32/Float64	Min Accuracy - ULP values⁸ Float16
$x + y$	Correctly rounded	Correctly rounded
$x - y$	Correctly rounded	Correctly rounded
$x * y$	Correctly rounded	Correctly rounded
$1.0 / x$	<= 3 ulp	Correctly rounded
x / y	<= 3 ulp	Correctly rounded
acos	<= 4 ulp	<= 2 ulp
acospi	<= 5 ulp	<= 2 ulp
asin	<= 4 ulp	<= 2 ulp
asinpi	<= 5 ulp	<= 2 ulp
atan	<= 5 ulp	<= 2 ulp
atan2	<= 6 ulp	<= 2 ulp

⁷ 0 ulp is used for math functions that do not require rounding.

⁸ 0 ulp is used for math functions that do not require rounding.

atanpi	≤ 5 ulp	≤ 2 ulp
atan2pi	≤ 6 ulp	≤ 2 ulp
acosh	≤ 4 ulp	≤ 2 ulp
asinh	≤ 4 ulp	≤ 2 ulp
atanh	≤ 5 ulp	≤ 2 ulp
cbrt	≤ 4 ulp	≤ 2 ulp
ceil	Correctly rounded	Correctly rounded
copysign	0 ulp	0 ulp
cos	≤ 4 ulp	≤ 2 ulp
cosh	≤ 4 ulp	≤ 2 ulp
cospi	≤ 4 ulp	≤ 2 ulp
erfc	≤ 16 ulp	≤ 4 ulp
erf	≤ 16 ulp	≤ 4 ulp
exp	≤ 4 ulp	≤ 2 ulp
exp2	≤ 4 ulp	≤ 2 ulp
exp10	≤ 4 ulp	≤ 2 ulp
expm1	≤ 4 ulp	≤ 2 ulp
fabs	0 ulp	0 ulp
fdim	Correctly rounded	Correctly rounded
floor	Correctly rounded	Correctly rounded
fma	Correctly rounded	Correctly rounded
fmax	0 ulp	0 ulp
fmin	0 ulp	0 ulp
fmod	0 ulp	0 ulp
fract	Correctly rounded	Correctly rounded
frexp	0 ulp	0 ulp
hypot	≤ 4 ulp	≤ 2 ulp
ilogb	0 ulp	0 ulp
ldexp	Correctly rounded	Correctly rounded
log	≤ 4 ulp	≤ 2 ulp
log2	≤ 4 ulp	≤ 2 ulp
log10	≤ 4 ulp	≤ 2 ulp
log1p	≤ 4 ulp	≤ 2 ulp
logb	0 ulp	0 ulp
mad	Any value allowed (infinite ulp)	Any value allowed (infinite ulp)
maxmag	0 ulp	0 ulp
minmag	0 ulp	0 ulp
modf	0 ulp	0 ulp
nan	0 ulp	0 ulp
nextafter	0 ulp	0 ulp
pow(x, y)	≤ 16 ulp	≤ 4 ulp
pown(x, y)	≤ 16 ulp	≤ 4 ulp
powr(x, y)	≤ 16 ulp	≤ 4 ulp
remainder	0 ulp	0 ulp

remquo	0 ulp	0 ulp
rint	Correctly rounded	Correctly rounded
rootn	≤ 16 ulp	≤ 4 ulp
round	Correctly rounded	Correctly rounded
rsqrt	≤ 4 ulp	≤ 1 ulp
sin	≤ 4 ulp	≤ 2 ulp
sincos	≤ 4 ulp for sine and cosine values	≤ 2 ulp for sine and cosine values
sinh	≤ 4 ulp	≤ 2 ulp
sinpi	≤ 4 ulp	≤ 2 ulp
sqrt	≤ 4 ulp	Correctly rounded
tan	≤ 5 ulp	≤ 2 ulp
tanh	≤ 5 ulp	≤ 2 ulp
tanpi	≤ 6 ulp	≤ 2 ulp
tgamma	≤ 16 ulp	≤ 4 ulp
trunc	Correctly rounded	Correctly rounded
half_cos	≤ 8192 ulp	Built-in not defined
half_divide	≤ 8192 ulp	Built-in not defined
half_exp	≤ 8192 ulp	Built-in not defined
half_exp2	≤ 8192 ulp	Built-in not defined
half_exp10	≤ 8192 ulp	Built-in not defined
half_log	≤ 8192 ulp	Built-in not defined
half_log2	≤ 8192 ulp	Built-in not defined
half_log10	≤ 8192 ulp	Built-in not defined
half_powr	≤ 8192 ulp	Built-in not defined
half_recip	≤ 8192 ulp	Built-in not defined
half_rsqrt	≤ 8192 ulp	Built-in not defined
half_sin	≤ 8192 ulp	Built-in not defined
half_sqrt	≤ 8192 ulp	Built-in not defined
half_tan	≤ 8192 ulp	Built-in not defined
native_cos	Implementation-defined	Built-in not defined
native_divide	Implementation-defined	Built-in not defined
native_exp	Implementation-defined	Built-in not defined
native_exp2	Implementation-defined	Built-in not defined
native_exp10	Implementation-defined	Built-in not defined
native_log	Implementation-defined	Built-in not defined
native_log2	Implementation-defined	Built-in not defined
native_log10	Implementation-defined	Built-in not defined
native_powr	Implementation-defined	Built-in not defined
native_recip	Implementation-defined	Built-in not defined
native_rsqrt	Implementation-defined	Built-in not defined
native_sin	Implementation-defined	Built-in not defined
native_sqrt	Implementation-defined	Built-in not defined

native_tan	Implementation-defined	Built-in not defined
-------------------	------------------------	----------------------

Table 8.4 *ULP values for built-in math functions*

8.5 Edge Case Behavior

The edge case behavior of the math functions (*section 6.13.2*) shall conform to sections F.9 and G.6 of ISO/IEC 9899:TC 2 (commonly known as C99, TC2), except where noted below in *section 7.5.1*.

8.5.1 Additional Requirements Beyond C99 TC2

Functions that return a NaN with more than one NaN operand shall return one of the NaN operands. Functions that return a NaN operand may silence the NaN if it is a signaling NaN. A non-signaling NaN shall be converted to a non-signaling NaN. A signaling NaN shall be converted to a NaN, and should be converted to a non-signaling NaN. How the rest of the NaN payload bits or the sign of NaN is converted is undefined.

half_<funcname> functions behave identically to the function of the same name without the **half_** prefix. They must conform to the same edge case requirements (see sections F.9 and G.6 of C99, TC2). For other cases, except where otherwise noted, these single precision functions are permitted to have up to 8192 ulps of error (as measured in the single precision result), although better accuracy is encouraged.

The usual allowances for rounding error (*section 7.4*) or flushing behavior (*section 7.5.3*) shall not apply for those values for which *section F.9* of C99, TC2, or *sections 7.5.1* and *7.5.3* below (and similar sections for other floating-point precisions) prescribe a result (e.g. **ceil** ($-1 < x < 0$) returns -0). Those values shall produce exactly the prescribed answers, and no other. Where the \pm symbol is used, the sign shall be preserved. For example, $\sin(\pm 0) = \pm 0$ shall be interpreted to mean $\sin(+0)$ is $+0$ and $\sin(-0)$ is -0 .

acospi (1) = $+0$.

acospi (x) returns a NaN for $|x| > 1$.

asinpi (± 0) = ± 0 .

asinpi (x) returns a NaN for $|x| > 1$.

atanpi (± 0) = ± 0 .

atanpi ($\pm\infty$) = ± 0.5 .

atan2pi ($\pm 0, -0$) = ± 1 .

atan2pi ($\pm 0, +0$) = ± 0 .

atan2pi ($\pm 0, x$) returns ± 1 for $x < 0$.
atan2pi ($\pm 0, x$) returns ± 0 for $x > 0$.
atan2pi ($y, \pm 0$) returns -0.5 for $y < 0$.
atan2pi ($y, \pm 0$) returns 0.5 for $y > 0$.
atan2pi ($\pm y, -\infty$) returns ± 1 for finite $y > 0$.
atan2pi ($\pm y, +\infty$) returns ± 0 for finite $y > 0$.
atan2pi ($\pm\infty, x$) returns ± 0.5 for finite x .
atan2pi ($\pm\infty, -\infty$) returns ± 0.75 .
atan2pi ($\pm\infty, +\infty$) returns ± 0.25 .

ceil ($-1 < x < 0$) returns -0 .

cospi (± 0) returns 1
cospi ($n + 0.5$) is $+0$ for any integer n where $n + 0.5$ is representable.
cospi ($\pm\infty$) returns a NaN.

exp10 (± 0) returns 1 .
exp10 ($-\infty$) returns $+0$.
exp10 ($+\infty$) returns $+\infty$.

distance (x, y) calculates the distance from x to y without overflow or extraordinary precision loss due to underflow.

fdim (any, NaN) returns NaN.
fdim (NaN, any) returns NaN.

fmod ($\pm 0, \text{NaN}$) returns NaN.

frexp ($\pm\infty, \text{exp}$) returns $\pm\infty$ and stores 0 in exp .
frexp (NaN, exp) returns the NaN and stores 0 in exp .

fract (x, iptr) shall not return a value greater than or equal to 1.0 , and shall not return a value less than 0 .

fract ($+0, \text{iptr}$) returns $+0$ and $+0$ in iptr .
fract ($-0, \text{iptr}$) returns -0 and -0 in iptr .
fract ($+\text{inf}, \text{iptr}$) returns $+0$ and $+\text{inf}$ in iptr .
fract ($-\text{inf}, \text{iptr}$) returns -0 and $-\text{inf}$ in iptr .
fract (NaN, iptr) returns the NaN and NaN in iptr .

length calculates the length of a vector without overflow or extraordinary precision loss due to underflow.

lgamma_r (x, signp) returns 0 in signp if x is zero or a negative integer.

nextafter ($-0, y > 0$) returns smallest positive denormal value.
nextafter ($+0, y < 0$) returns smallest negative denormal value.

normalize shall reduce the vector to unit length, pointing in the same direction without overflow or extraordinary precision loss due to underflow.

normalize (v) returns v if all elements of v are zero.

normalize (v) returns a vector full of NaNs if any element is a NaN.

normalize (v) for which any element in v is infinite shall proceed as if the elements in v were replaced as follows:

```
for( i = 0; i < sizeof(v) / sizeof(v[0] ); i++)  
    v[i] = isinf(v[i] ) ? copysign(1.0, v[i]) : 0.0 * v [i];
```

pow ($\pm 0, -\infty$) returns $+\infty$

pow ($x, 0$) is 1 for any x , even zero, NaN or infinity.

pow ($\pm 0, n$) is $\pm\infty$ for odd $n < 0$.

pow ($\pm 0, n$) is $+\infty$ for even $n < 0$.

pow ($\pm 0, n$) is $+0$ for even $n > 0$.

pow ($\pm 0, n$) is ± 0 for odd $n > 0$.

pow ($x, \pm 0$) is 1 for finite $x > 0$.

pow ($\pm 0, y$) is $+\infty$ for finite $y < 0$.

pow ($\pm 0, -\infty$) is $+\infty$.

pow ($\pm 0, y$) is $+0$ for $y > 0$.

pow ($+1, y$) is 1 for finite y .

pow (x, y) returns NaN for $x < 0$.

pow ($\pm 0, \pm 0$) returns NaN.

pow ($+\infty, \pm 0$) returns NaN.

pow ($+1, \pm\infty$) returns NaN.

pow (x, NaN) returns the NaN for $x \geq 0$.

pow (NaN, y) returns the NaN.

rint ($-0.5 \leq x < 0$) returns -0 .

remquo ($x, y, \&quo$) returns a NaN and 0 in *quo* if x is $\pm\infty$, or if y is 0 and the other argument is non-NaN or if either argument is a NaN.

rootn ($\pm 0, n$) is $\pm\infty$ for odd $n < 0$.

rootn ($\pm 0, n$) is $+\infty$ for even $n < 0$.

rootn ($\pm 0, n$) is $+0$ for even $n > 0$.

rootn ($\pm 0, n$) is ± 0 for odd $n > 0$.

rootn (x, n) returns a NaN for $x < 0$ and n is even.

rootn ($x, 0$) returns a NaN.

round ($-0.5 < x < 0$) returns -0 .

sinpi (± 0) returns ± 0 .
sinpi ($+n$) returns $+0$ for positive integers n .
sinpi ($-n$) returns -0 for negative integers n .
sinpi ($\pm\infty$) returns a NaN.

tanpi (± 0) returns ± 0 .
tanpi ($\pm\infty$) returns a NaN.
tanpi (n) is **copysign**(0.0, n) for even integers n .
tanpi (n) is **copysign**(0.0, $-n$) for odd integers n .
tanpi ($n + 0.5$) for even integer n is $+\infty$ where $n + 0.5$ is representable.
tanpi ($n + 0.5$) for odd integer n is $-\infty$ where $n + 0.5$ is representable.

trunc ($-1 < x < 0$) returns -0 .

8.5.2 Changes to C99 TC2 Behavior

modf behaves as though implemented by:

```
gentype modf ( gentype value, gentype *iptr )
{
    *iptr = trunc( value );
    return copysign( isinf( value ) ? 0.0 : value - *iptr, value );
}
```

rint always rounds according to round to nearest even rounding mode even if the caller is in some other rounding mode.

8.5.3 Edge Case Behavior in Flush To Zero Mode

If denormals are flushed to zero, then a function may return one of four results:

1. Any conforming result for non-flush-to-zero mode
2. If the result given by 1. is a sub-normal before rounding, it may be flushed to zero
3. Any non-flushed conforming result for the function if one or more of its sub-normal operands are flushed to zero.
4. If the result of 3. is a sub-normal before rounding, the result may be flushed to zero.

In each of the above cases, if an operand or result is flushed to zero, the sign of the zero is undefined.

If subnormals are flushed to zero, a device may choose to conform to the following edge cases for **nextafter** instead of those listed in *section 7.5.1*:

nextafter (+smallest normal, $y < +\text{smallest normal}$) = +0.

nextafter (-smallest normal, $y > -\text{smallest normal}$) = -0.

nextafter (-0, $y > 0$) returns smallest positive normal value.

nextafter (+0, $y < 0$) returns smallest negative normal value.

For clarity, subnormals or denormals are defined to be the set of representable numbers in the range $0 < x < \text{TYPE_MIN}$ and $-\text{TYPE_MIN} < x < -0$. They do not include ± 0 . A non-zero number is said to be sub-normal before rounding if after normalization, its radix-2 exponent is less than $(\text{TYPE_MIN_EXP} - 1)$.⁹

⁹ Here `TYPE_MIN` and `TYPE_MIN_EXP` should be substituted by constants appropriate to the floating-point type under consideration, such as `FLT_MIN` and `FLT_MIN_EXP` for float.