# The OpenCL SPIR-V Environment Specification

*Version:* 2.2

Khronos OpenCL Working Group

*Revision:* v2.2-3

*Editor:* Ben Ashbaugh

May 12, 2017

# Contents

# List of Tables

# Chapter 1

# 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 in OpenCL may be written in the OpenCL C source language, or may compiled from OpenCL C, OpenCL C++, or other source languages into SPIR-V modules.

All SPIR-V intermediate binary modules are consumed by environments, such as an API, a specific version of an API, or an implementation of an API. The environment describes required support for some SPIR-V capabilities, additional semantics for some SPIR-V instructions, and additional validation rules a module must adhere to in order to be considered valid.

This document describes the environment for implementations of the OpenCL API. It is written for compiler developers who are generating SPIR-V modules to be consumed by the OpenCL API, for implementors of the OpenCL API who are consuming SPIR-V modules, and by software developers who are using SPIR-V modules with the OpenCL API.

# Chapter 2

# SPIR-V Consumption

This section describes common properties of all OpenCL environments. Subsequent sections describe environments for specific versions of OpenCL, and how an environment may additionally be modified via OpenCL or SPIR-V extensions.

A SPIR-V module passed to an OpenCL environment is interpreted as a series of 32-bit words in host endianness, with literal strings packed as described in the SPIR-V specification. The first few words of the SPIR-V module must be a magic number and a SPIR-V version number, as described in the SPIR-V specification.

## 2.1 Validation Rules

The following are a list of validation rules that apply to SPIR-V modules executing in all OpenCL environments:

- The *Execution Model* declared in **OpEntryPoint** must be **Kernel**.
- The *Addressing Model* declared in **OpMemoryModel** must be either **Physical32** or **Physical64**:
  - Modules indicating a **Physical32** *Addressing Model* are valid for OpenCL devices reporting `32` for `CL_DEVICE_ADDRESS_BITS`.
  - Modules indicating a **Physical64** *Addressing Model* are valid for OpenCL devices reporting `64` for `CL_DEVICE_ADDRESS_BITS`.
- The *Memory Model* declared in **OpMemoryModel** must be **OpenCL**.
- For all **OpTypeImage** type-declaration instructions:
  - *MS* must be 0, indicating single-sampled content.
  - *Arrayed* may only be set to 1, indicating arrayed content, when *Dim* is set to **1D** or **2D**.
- The image write instruction **OpImageWrite** must not include any optional *Image Operands*.
- The image read instructions **OpImageRead**, **OpImageFetch**, and **OpImageSampleExplicitLod** must not include the optional *Image Operand* **ConstOffset**.
- Only 32-bit integer types are supported for the *Result Type* and/or type of *Value* for all **Atomic Instructions**.
- Recursion is not supported. The static function call graph for an entry point must not contain cycles.

## 2.2 Source Language Encoding

If a SPIR-V module represents a program written in OpenCL C, then the *Source Language* operand for the **OpSource** instruction should be **OpenCL_C**, and the 32-bit literal language *Version* should describe the version of OpenCL C, encoded MSB to LSB as:

```
0 | Major Number | Minor Number | Revision Number (optional)
```

Hence, OpenCL C 1.2 would be encoded as `0x00010200`, and OpenCL C 2.0 as `0x00020000`.

If a SPIR-V module represents a program written in OpenCL C++, then the *Source Language* operand for the **OpSource** instruction should be **OpenCL_CPP**, and the 32-bit literal language *Version* should describe the version of OpenCL C++, encoded similarly. Hence, OpenCL C++ 2.2 would be encoded as `0x00020200`.

The source language version is purely informational and has no semantic meaning.

## 2.3 Numerical Type Formats

For all OpenCL environments, floating-point types are represented and stored using IEEE-754 semantics. All integer formats are represented and stored using 2's compliment format.

## 2.4 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 orders.

Table 2.1: Image Channel Order mapping

| **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` |
| `sRGB` | `CL_sRGB` |
| `sRGBA` | `CL_sRGBA` |
| `sBGRA` | `CL_sBGRA` |
| `sRGBx` | `CL_sRGBx` |

## 2.5 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 types.

Table 2.2: Image Channel Data Type mapping

| SPIR-V Image Channel Data Type | OpenCL Image Channel Data Type |
|---|---|
| `SnormInt8` | `CL_SNORM_INT8` |
| `SnormInt16` | `CL_SNORM_INT16` |
| `UnormInt8` | `CL_UNORM_INT8` |
| `UnormInt16` | `CL_UNORM_INT16` |
| `UnormInt24` | `CL_UNORM_INT24` |
| `UnormShort565` | `CL_UNORM_SHORT_565` |
| `UnormShort555` | `CL_UNORM_SHORT_555` |
| `UnormInt101010` | `CL_UNORM_INT_101010` |
| `SignedInt8` | `CL_SIGNED_INT8` |
| `SignedInt16` | `CL_SIGNED_INT16` |
| `SignedInt32` | `CL_SIGNED_INT32` |
| `UnsignedInt8` | `CL_UNSIGNED_INT8` |
| `UnsignedInt16` | `CL_UNSIGNED_INT16` |
| `UnsignedInt32` | `CL_UNSIGNED_INT32` |
| `HalfFloat` | `CL_HALF_FLOAT` |
| `Float` | `CL_FLOAT` |

## 2.6 Kernels

An **OpFunction** in a SPIR-V module that is identified with **OpEntryPoint** defines an OpenCL kernel that may be invoked using the OpenCL host API enqueue kernel interfaces.

## 2.7 Kernel Return Types

The *Result Type* for an **OpFunction** identified with **OpEntryPoint** must be **OpTypeVoid**.

## 2.8 Kernel Arguments

An **OpFunctionParameter** for an **OpFunction** that is identified with **OpEntryPoint** defines an OpenCL kernel argument. Allowed types for OpenCL kernel arguments are:

- **OpTypeInt**
- **OpTypeFloat**
- **OpTypeStruct**
- **OpTypeVector**
- **OpTypePointer**
- **OpTypeSampler**

- **OpTypeImage**
- **OpTypePipe**
- **OpTypeQueue**

For **OpTypeInt** parameters, supported *Widths* are 8, 16, 32, and 64, and may be signed or unsigned.

For **OpTypeFloat** parameters, *Width* must be 32.

For **OpTypeStruct** parameters, supported structure *Member Types* are:

- **OpTypeInt**
- **OpTypeFloat**
- **OpTypeStruct**
- **OpTypeVector**
- **OpTypePointer**

For **OpTypePointer** parameters, supported *Storage Classes* are:

- **CrossWorkgroup**
- **Workgroup**
- **UniformConstant**

OpenCL kernel argument types must have a representation in the OpenCL host API.

Environments that support extensions or optional features may allow additional types in an entry point's parameter list.

# Chapter 3

# OpenCL 2.2

An OpenCL 2.2 environment must accept SPIR-V 1.0, 1.1, and 1.2 modules.

## 3.1 Full Profile

An OpenCL 2.2 Full Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **DeviceEnqueue**
- **Float16Buffer**
- **GenericPointer**
- **Groups**
- **Int64**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Pipes**
- **Vector16**
- **SubgroupDispatch**
- **PipeStorage**

The following capabilities may be optionally supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 3.2 Embedded Profile

An OpenCL 2.2 Embedded Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **DeviceEnqueue**
- **Float16Buffer**
- **GenericPointer**
- **Groups**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Pipes**
- **Vector16**
- **SubgroupDispatch**
- **PipeStorage**

Furthermore, the following capabilities may optionally be supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 3.3 Validation Rules

The following are a list of validation rules for SPIR-V modules executing in an OpenCL 2.2 environment:

*Scope* for *Execution* is generally limited to:

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

*Scope* for *Memory* is generally limited to:

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

*Scope* for *Execution* for the **OpGroupAsyncCopy** and **OpGroupWaitEvents** instructions is specifically limited to:

- **Workgroup**

# Chapter 4

# OpenCL 2.1

An OpenCL 2.1 environment must accept SPIR-V 1.0 modules.

## 4.1 Full Profile

An OpenCL 2.1 Full Profile environment is guaranteed to support the following SPIR-V capabilities:

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

The following capabilities may be optionally supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 4.2 Embedded Profile

An OpenCL 2.1 Embedded Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **DeviceEnqueue**
- **Float16Buffer**
- **GenericPointer**
- **Groups**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Pipes**
- **Vector16**

Furthermore, the following capabilities may optionally be supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 4.3 Validation Rules

The following are a list of validation rules for SPIR-V modules executing in an OpenCL 2.1 environment:

*Scope* for *Execution* is generally limited to:

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

*Scope* for *Memory* is generally limited to:

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

*Scope* for *Execution* for the **OpGroupAsyncCopy** and **OpGroupWaitEvents** instructions is specifically limited to:

- **Workgroup**

# Chapter 5

# OpenCL 2.0

An OpenCL 2.0 environment must accept SPIR-V 1.0 modules if it includes the optional extension `cl_khr_il_program` in the host API `CL_PLATFORM_EXTENSIONS` or `CL_DEVICE_EXTENSIONS` string.

## 5.1 Full Profile

An OpenCL 2.0 Full Profile environment is guaranteed to support the following SPIR-V capabilities:

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

The following capabilities may be optionally supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 5.2 Embedded Profile

An OpenCL 2.0 Embedded Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **DeviceEnqueue**
- **Float16Buffer**
- **GenericPointer**
- **Groups**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Pipes**
- **Vector16**

Furthermore, the following capabilities may optionally be supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **ImageReadWrite**
- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 5.3 Validation Rules

The following are a list of validation rules for SPIR-V modules executing in an OpenCL 2.0 environment:

*Scope* for *Execution* is generally limited to:

- **Workgroup**

*Scope* for *Memory* is generally limited to:

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

# Chapter 6

# OpenCL 1.2

An OpenCL 1.2 environment must accept SPIR-V 1.0 modules if it includes the optional extension `cl_khr_il_program` in the host API `CL_PLATFORM_EXTENSIONS` or `CL_DEVICE_EXTENSIONS` string.

## 6.1 Full Profile

An OpenCL 1.2 Full Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **Float16Buffer**
- **Groups**
- **Int64**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Vector16**

The following capabilities may be optionally supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 6.2 Embedded Profile

An OpenCL 1.2 Embedded Profile environment is guaranteed to support the following SPIR-V capabilities:

- **Address**
- **Float16Buffer**
- **Groups**
- **Int16**
- **Int8**
- **Kernel**
- **Linkage**
- **Pipes**
- **Vector16**

The following capabilities may be optionally supported:

- **ImageBasic**, if `CL_DEVICE_IMAGE_SUPPORT` is `CL_TRUE`
- **Float64**, if the device supports double precision floating-point

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

- **LiteralSampler**
- **Sampled1D**
- **Image1D**
- **SampledBuffer**
- **ImageBuffer**

## 6.3 Validation Rules

The following are a list of validation rules for SPIR-V modules executing in an OpenCL 1.2 environment:

*Scope* for *Execution* is generally limited to:

- **Workgroup**

*Scope* for *Memory* is generally limited to:

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

The following **Group Instructions** are not supported:

- **OpGroupAll**
- **OpGroupAny**
- **OpGroupBroadcast**

- **OpGroupIAdd**
- **OpGroupFAdd**
- **OpGroupFMin**
- **OpGroupUMin**
- **OpGroupSMin**
- **OpGroupFMax**
- **OpGroupUMax**
- **OpGroupSMax**

For the **Barrier Instructions OpControlBarrier** and **OpMemoryBarrier**, the *Scope* for execution must be **Workgroup**, the *Scope* for memory must be **Workgroup**, and the *Memory Semantics* must be **SequentiallyConsistent**.

For the **Atomic Instructions**, the *Scope* must be **Device**, and the *Memory Semantics* must be **Relaxed**.

# Chapter 7

# OpenCL Extensions

An OpenCL environment may be modified by OpenCL extensions that affect SPIR-V. An implementation indicates support for an OpenCL extension by including the extension name in the host API `CL_PLATFORM_EXTENSIONS` or `CL_DEVICE_EXTENSIONS` string. To enable the behavior described for an extension, a SPIR-V module must declare use of the extension using **OpExtension** and the name of the extension, for example:

```
OpExtension "cl_khr_extension_name"
```

This section describes how the OpenCL environment is modified by Khronos (`khr`) OpenCL extensions. Other OpenCL extensions, such as multi-vendor (`ext`) extensions or vendor-specific extensions, describe how they modify the OpenCL environment in their individual extension documents.

## 7.1 Full and Embedded Profile Extensions

### 7.1.1 `cl_khr_3d_image_writes`

If the OpenCL environment supports the extension `cl_khr_3d_image_writes`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept *Image* operands to **OpImageWrite** that are declared with with dimensionality **3D**.

### 7.1.2 `cl_khr_depth_images`

If the OpenCL environment supports the extension `cl_khr_depth_images`, and use of the extension is declared in the module using **OpExtension**, then the following Image Channel Orders may additionally be returned by **OpImageQueryOrder**:

- **Depth**

### 7.1.3 `cl_khr_device_enqueue_local_arg_types`

If the OpenCL environment supports the extension `cl_khr_device_enqueue_local_arg_types`, and use of the extension is declared in the module using **OpExtension**, then then environment will allow *Invoke* functions to be passed to **OpEnqueueKernel** with **Workgroup** memory pointer parameters of any type.

### 7.1.4 `cl_khr_fp16`

If the OpenCL environment supports the extension `cl_khr_fp16`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept modules that declare the following SPIR-V capabilities:

- **Float16**

### 7.1.5 `cl_khr_fp64`

If the OpenCL environment supports the extension `cl_khr_fp64`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept modules that declare the following SPIR-V capabilities:

- **Float64**

### 7.1.6 `cl_khr_gl_depth_images`

If the OpenCL environment supports the extension `cl_khr_gl_depth_images`, and use of the extension is declared in the module using **OpExtension**, then:

The following Image Channel Orders may additionally be returned by **OpImageQueryOrder**:

- **DepthStencil**

Also, the following Image Channel Data Types may additionally be returned by **OpImageQueryFormat**:

- **UnormInt24**

### 7.1.7 `cl_khr_gl_msaa_sharing`

If the OpenCL environment supports the extension `cl_khr_gl_msaa_sharing`, and use of the extension is declared in the module using **OpExtension**, then 2D multi-sampled image types may be declared using **OpTypeImage** with dimensionality *Dim* equal to **2D** and *MS* equal to 1, indicating multi-sampled content.

The 2D multi-sampled images may be used with the following instructions:

- **OpImageRead**
- **OpImageQuerySizeLod**
- **OpImageQueryFormat**
- **OpImageQueryOrder**
- **OpImageQuerySamples**

### 7.1.8 `cl_khr_int64_base_atomics` and `cl_khr_int64_extended_atomics`

If the OpenCL environment supports the extension `cl_khr_int64_base_atomics` or `cl_khr_int64_extended_atomics`, and use of either extension is declared in the module using **OpExtension**, then the environment must support 64-bit integer operands for all of the SPIR-V **Atomic Instructions**.

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

**Note**
OpenCL environments that consume SPIR-V must support both `cl_khr_int64_base_atomics` and `cl_khr_int64_extended_atomics` or neither of these extensions. Only one of the extension strings need to be declared via **OpExtension**.

### 7.1.9 cl_khr_mipmap_image

If the OpenCL environment supports the extension `cl_khr_mipmap_image`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept non-zero optional **Lod** *Image Operands* for the following instructions:

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

---

**Note**
Implementations that support `cl_khr_mipmap_image` are not guaranteed to support the **ImageMipmap** capability, since this extension does not require non-zero optional **Lod** *Image Operands* for **OpImageWrite**.

---

### 7.1.10 cl_khr_mipmap_image_writes

If the OpenCL environment supports the extension `cl_khr_mipmap_image_writes`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept non-zero optional **Lod** *Image Operands* for the following instructions:

- **OpImageWrite**

---

**Note**
An implementation that supports `cl_khr_mipmap_image_writes` must also support `cl_khr_mipmap_image`, and support for both extensions does guarantee support for the **ImageMipmap** capability.

---

### 7.1.11 cl_khr_subgroups

If the OpenCL environment supports the extension `cl_khr_subgroups`, and use of the extension is declared in the module using **OpExtension**, then the environment will generally allows the scope for *Execution* to include:

- **Subgroup**

However, the *Scope* for *Execution* for the **OpGroupAsyncCopy** and **OpGroupWaitEvents** instructions still is limited to:

- **Workgroup**

### 7.1.12 cl_khr_subgroup_named_barrier

If the OpenCL environment supports the extension `cl_khr_subgroup_named_barrier`, and use of the extension is declared in the module using **OpExtension**, the the environment must accept modules that declare the following SPIR-V capabilities:

- **NamedBarrier**

## 7.2 Embedded Profile Extensions

### 7.2.1 `cles_khr_int64`

If the OpenCL environment supports the extension `cles_khr_int64`, and use of the extension is declared in the module using **OpExtension**, then the environment must accept modules that declare the following SPIR-V capabilities:

- **Int64**

# Chapter 8

# OpenCL Numerical Compliance

This section describes features of the C++14 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. Half precision floating-point is an optional feature indicated by the **Float16** capability. Double precision floating-point is also an optional feature indicated by the **Float64** capability.

## 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 +infinity*
- *Round toward -infinity*
- *Round toward zero*

The complete set of rounding modes supported by the device are described by the `CL_DEVICE_SINGLE_FP_CONFIG`, `CL_DEVICE_HALF_FP_CONFIG`, and `CL_DEVICE_DOUBLE_FP_CONFIG` device queries.

For double precision operations, *Round to nearest even* is a required rounding mode, and is therefore the default rounding mode for double precision operations.

For single precision operations, devices supporting the full profile must support *Round to nearest even*, therefore for full profile devices this is the default rounding mode for single precision operations. Devices supporting the embedded profile may support either *Round to nearest even* or *Round toward zero* as the default rounding mode for single precision operations.

For half precision operations, devices may support either *Round to nearest even* or *Round toward zero* as the default rounding mode for half precision operations.

Only static selection of rounding mode is supported. Dynamically reconfiguring the rounding mode as specified by the IEEE 754 spec is not supported.

## 8.2 Rounding Modes for Conversions

The rounding mode for conversion operations may be specified explicitly via an **FPRoundingMode** decoration or may use an implicit rounding mode. If no rounding mode is specified explicitly then the default rounding mode for OpenCL conversion operations is:

- *Round to nearest even*, for conversions to floating-point types.
- *Round toward zero*, for conversions from floating-point to integer types.

## 8.3 INF, NaN, and Denormalized Numbers

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

Support for denormalized numbers with single precision and half precision floating-point is optional. Denormalized single precision or half precision floating-point numbers passed as the input or produced as the output of single precision or half precision floating-point operations may be flushed to zero. Support for denormalized numbers is required for double precision floating-point.

Support for INFs, NaNs, and denormalized numbers is described by the `CL_FP_DENORM` and `CL_FP_INF_NAN` bits in the `CL_DEVICE_SINGLE_FP_CONFIG`, `CL_DEVICE_HALF_FP_CONFIG`, and `CL_DEVICE_DOUBLE_FP_CONFIG` device queries.

## 8.4 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 re-clearing 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.

## 8.5 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 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(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*

0 ULP is used for math functions that do not require rounding. The reference value used to compute the ULP value is the infinitely precise result.

### 8.5.1 ULP Values for Math Instructions - Full Profile

The ULP Values for Math Instructions table below describes the minimum accuracy of floating-point math arithmetic instructions for full profile devices given as ULP values.

Table 8.1: ULP Values for Math Instructions - Full Profile

| **SPIR-V Instruction** | **Minimum Accuracy - Float64** | **Minimum Accuracy - Float32** | **Minimum Accuracy - Float16** |
|---|---|---|---|
| **OpFAdd** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFSub** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFMul** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFDiv** | Correctly rounded | <= 2.5 ulp | Correctly rounded |
| **OpExtInst acos** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst acosh** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst acospi** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst asin** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst asinh** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst asinpi** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst atan** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst atanh** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst atanpi** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst atan2** | <= 6 ulp | <= 6 ulp | <= 2 ulp |
| **OpExtInst atan2pi** | <= 6 ulp | <= 6 ulp | <= 2 ulp |
| **OpExtInst cbrt** | <= 2 ulp | <= 2 ulp | <= 2 ulp |
| **OpExtInst ceil** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst copysign** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst cos** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst cosh** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst cospi** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst erfc** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst erf** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst exp** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst exp2** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst exp10** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst expm1** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst fabs** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fdim** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst floor** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst fma** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst fmax** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fmin** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fmod** | 0 ulp | 0 ulp | 0 ulp |

Table 8.1: (continued)

| SPIR-V Instruction | Minimum Accuracy - Float64 | Minimum Accuracy - Float32 | Minimum Accuracy - Float16 |
|---|---|---|---|
| **OpExtInst fract** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst frexp** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst hypot** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst ilogb** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst ldexp** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst lgamma** | Implementation-defined | Implementation-defined | Implementation-defined |
| **OpExtInst lgamma_r** | Implementation-defined | Implementation-defined | Implementation-defined |
| **OpExtInst log** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst log2** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst log10** | <= 3 ulp | <= 3 ulp | <= 2 ulp |
| **OpExtInst log1p** | <= 2 ulp | <= 2 ulp | <= 2 ulp |
| **OpExtInst logb** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst mad** | Implementation-defined | Implemented either as a correctly rounded fma, or as a multiply followed by an add, both of which are correctly rounded | Implementation-defined |
| **OpExtInst maxmag** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst minmag** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst modf** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst nan** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst nextafter** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst pow** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst pown** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst powr** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst remainder** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst remquo** | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient |
| **OpExtInst rint** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst rootn** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst round** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst rsqrt** | <= 2 ulp | <= 2 ulp | <= 1 ulp |
| **OpExtInst sin** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst sincos** | <= 4 ulp for sine and cosine values | <= 4 ulp for sine and cosine values | <= 2 ulp for sine and cosine values |
| **OpExtInst sinh** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst sinpi** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst sqrt** | Correctly rounded | <= 3 ulp | Correctly rounded |
| **OpExtInst tan** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst tanh** | <= 5 ulp | <= 5 ulp | <= 2 ulp |
| **OpExtInst tanpi** | <= 6 ulp | <= 6 ulp | <= 2 ulp |
| **OpExtInst tgamma** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst trunc** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst half_cos** | | <= 8192 ulp | |
| **OpExtInst half_divide** | | <= 8192 ulp | |
| **OpExtInst half_exp** | | <= 8192 ulp | |
| **OpExtInst half_exp2** | | <= 8192 ulp | |
| **OpExtInst half_exp10** | | <= 8192 ulp | |
| **OpExtInst half_log** | | <= 8192 ulp | |

Table 8.1: (continued)

| SPIR-V Instruction | Minimum Accuracy - Float64 | Minimum Accuracy - Float32 | Minimum Accuracy - Float16 |
|---|---|---|---|
| **OpExtInst half_log2** | | <= 8192 ulp | |
| **OpExtInst half_log10** | | <= 8192 ulp | |
| **OpExtInst half_powr** | | <= 8192 ulp | |
| **OpExtInst half_recip** | | <= 8192 ulp | |
| **OpExtInst half_rsqrt** | | <= 8192 ulp | |
| **OpExtInst half_sin** | | <= 8192 ulp | |
| **OpExtInst half_sqrt** | | <= 8192 ulp | |
| **OpExtInst half_tan** | | <= 8192 ulp | |
| **OpExtInst native_cos** | | Implementation-defined | |
| **OpExtInst native_divide** | | Implementation-defined | |
| **OpExtInst native_exp** | | Implementation-defined | |
| **OpExtInst native_exp2** | | Implementation-defined | |
| **OpExtInst native_exp10** | | Implementation-defined | |
| **OpExtInst native_log** | | Implementation-defined | |
| **OpExtInst native_log2** | | Implementation-defined | |
| **OpExtInst native_log10** | | Implementation-defined | |
| **OpExtInst native_powr** | | Implementation-defined | |
| **OpExtInst native_recip** | | Implementation-defined | |
| **OpExtInst native_rsqrt** | | Implementation-defined | |
| **OpExtInst native_sin** | | Implementation-defined | |
| **OpExtInst native_sqrt** | | Implementation-defined | |
| **OpExtInst native_tan** | | Implementation-defined | |

### 8.5.2 ULP Values for Math Instructions - Embedded Profile

The ULP Values for Math instructions for Embedded Profile table below describes the minimum accuracy of floating-point math arithmetic operations given as ULP values for the embedded profile.

Table 8.2: ULP Values for Math Instructions - Embedded Profile

| SPIR-V Instruction | Minimum Accuracy - Float64 | Minimum Accuracy - Float32 | Minimum Accuracy - Float16 |
|---|---|---|---|
| **OpFAdd** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFSub** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFMul** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpFDiv** | <= 3 ulp | <= 3 ulp | <= 1 ulp |
| **OpExtInst acos** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst acosh** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst acospi** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst asin** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst asinh** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst asinpi** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst atan** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst atanh** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst atanpi** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst atan2** | <= 6 ulp | <= 6 ulp | <= 3 ulp |
| **OpExtInst atan2pi** | <= 6 ulp | <= 6 ulp | <= 3 ulp |
| **OpExtInst cbrt** | <= 4 ulp | <= 4 ulp | <= 2 ulp |

Table 8.2: (continued)

| SPIR-V Instruction | Minimum Accuracy - Float64 | Minimum Accuracy - Float32 | Minimum Accuracy - Float16 |
|---|---|---|---|
| **OpExtInst ceil** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst copysign** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst cos** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst cosh** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst cospi** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst erfc** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst erf** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst exp** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst exp2** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst exp10** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst expm1** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst fabs** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fdim** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst floor** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst fma** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst fmax** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fmin** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fmod** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst fract** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst frexp** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst hypot** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst ilogb** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst ldexp** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst lgamma** | Implementation-defined | Implementation-defined | Implementation-defined |
| **OpExtInst lgamma_r** | Implementation-defined | Implementation-defined | Implementation-defined |
| **OpExtInst log** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst log2** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst log10** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst log1p** | <= 4 ulp | <= 4 ulp | <= 3 ulp |
| **OpExtInst logb** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst mad** | Implemention-defined | Implemention-defined | Implemention-defined |
| **OpExtInst maxmag** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst minmag** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst modf** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst nan** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst nextafter** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst pow** | <= 16 ulp | <= 16 ulp | <= 5 ulp |
| **OpExtInst pown** | <= 16 ulp | <= 16 ulp | <= 5 ulp |
| **OpExtInst powr** | <= 16 ulp | <= 16 ulp | <= 5 ulp |
| **OpExtInst remainder** | 0 ulp | 0 ulp | 0 ulp |
| **OpExtInst remquo** | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient | 0 ulp for the remainder, at least the lower 7 bits of the integral quotient |
| **OpExtInst rint** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst rootn** | <= 16 ulp | <= 16 ulp | <= 5 ulp |
| **OpExtInst round** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst rsqrt** | <= 4 ulp | <= 4 ulp | <= 1 ulp |
| **OpExtInst sin** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst sincos** | <= 4 ulp for sine and cosine values | <= 4 ulp for sine and cosine values | <= 2 ulp for sine and cosine values |
| **OpExtInst sinh** | <= 4 ulp | <= 4 ulp | <= 3 ulp |

Table 8.2: (continued)

| SPIR-V Instruction | Minimum Accuracy - Float64 | Minimum Accuracy - Float32 | Minimum Accuracy - Float16 |
|---|---|---|---|
| **OpExtInst sinpi** | <= 4 ulp | <= 4 ulp | <= 2 ulp |
| **OpExtInst sqrt** | <= 4 ulp | <= 4 ulp | <= 1 ulp |
| **OpExtInst tan** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst tanh** | <= 5 ulp | <= 5 ulp | <= 3 ulp |
| **OpExtInst tanpi** | <= 6 ulp | <= 6 ulp | <= 3 ulp |
| **OpExtInst tgamma** | <= 16 ulp | <= 16 ulp | <= 4 ulp |
| **OpExtInst trunc** | Correctly rounded | Correctly rounded | Correctly rounded |
| **OpExtInst half_cos** | | <= 8192 ulp | |
| **OpExtInst half_divide** | | <= 8192 ulp | |
| **OpExtInst half_exp** | | <= 8192 ulp | |
| **OpExtInst half_exp2** | | <= 8192 ulp | |
| **OpExtInst half_exp10** | | <= 8192 ulp | |
| **OpExtInst half_log** | | <= 8192 ulp | |
| **OpExtInst half_log2** | | <= 8192 ulp | |
| **OpExtInst half_log10** | | <= 8192 ulp | |
| **OpExtInst half_powr** | | <= 8192 ulp | |
| **OpExtInst half_recip** | | <= 8192 ulp | |
| **OpExtInst half_rsqrt** | | <= 8192 ulp | |
| **OpExtInst half_sin** | | <= 8192 ulp | |
| **OpExtInst half_sqrt** | | <= 8192 ulp | |
| **OpExtInst half_tan** | | <= 8192 ulp | |
| **OpExtInst native_cos** | | Implementation-defined | |
| **OpExtInst native_divide** | | Implementation-defined | |
| **OpExtInst native_exp** | | Implementation-defined | |
| **OpExtInst native_exp2** | | Implementation-defined | |
| **OpExtInst native_exp10** | | Implementation-defined | |
| **OpExtInst native_log** | | Implementation-defined | |
| **OpExtInst native_log2** | | Implementation-defined | |
| **OpExtInst native_log10** | | Implementation-defined | |
| **OpExtInst native_powr** | | Implementation-defined | |
| **OpExtInst native_recip** | | Implementation-defined | |
| **OpExtInst native_rsqrt** | | Implementation-defined | |
| **OpExtInst native_sin** | | Implementation-defined | |
| **OpExtInst native_sqrt** | | Implementation-defined | |
| **OpExtInst native_tan** | | Implementation-defined | |

### 8.5.3 ULP Values for Math Instructions - Fast Relaxed Math

The ULP Values for Math Instructions with Fast Relaxed Math table below describes the minimum accuracy of commonly used single precision floating-point math arithmetic instructions given as ULP values if the *-cl-fast-relaxed-math* compiler option is specified when compiling or building the OpenCL program.

For derived implementations, the operations used in the derivation may themselves be relaxed according to the ULP Values for Math Instructions with Fast Relaxed Math table.

The minimum accuracy of math functions not defined in the ULP Values for Math Instructions with Fast Relaxed Math table when the *-cl-fast-relaxed-math* compiler option is specified is as defined in the ULP Values for Math Instructions for Full Profile table when operating in the full profile, and as defined in the ULP Values for Math instructions for Embedded Profile table when operating in the embedded profile.

Table 8.3: ULP Values for Single Precision Math Instructions with *-cl-fast-relaxed-math*

| SPIR-V Instruction | Minimum Accuracy |
|---|---|
| **OpFDiv** for 1.0 / *x* | <= 2.5 ulp for x in the domain of $2^{-126}$ to $2^{126}$ for the full profile, and <= 3 ulp for the embedded profile. |
| **OpFDiv** for *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 $2^{62}$ for the full profile, and <= 3 ulp for the embedded profile. |
| **OpExtInst acos** | <= 4096 ulp |
| **OpExtInst acosh** | Implemented as log( x + sqrt(x*x - 1) ). |
| **OpExtInst acospi** | Implemented as acos(x) * M_PI_F. For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst asin** | <= 4096 ulp |
| **OpExtInst asinh** | Implemented as log( x + sqrt(x*x + 1) ). |
| **OpExtInst asinpi** | Implemented as asin(x) * M_PI_F. For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst atan** | <= 4096 ulp |
| **OpExtInst atanh** | Defined for x in the domain (-1, 1). For x in [$-2^{-10}$, $2^{-10}$], implemented as x. For x outside of [$-2^{-10}$, $2^{-10}$], implemented as 0.5f * log( (1.0f + x) / (1.0f - x) ). For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst atanpi** | Implemented as atan(x) * M_1_PI_F. For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst atan2** | 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. |
| **OpExtInst atan2pi** | Implemented as atan2(y, x) * M_1_PI_F. For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst cbrt** | Implemented as rootn(x, 3). For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst cos** | For x in the domain [$-\pi$, $\pi$], the maximum absolute error is <= $2^{-11}$ and larger otherwise. |
| **OpExtInst cosh** | Defined for x in the domain [-88, 88] and implemented as 0.5f * ( exp(x) + exp(-x) ). For non-derived implementations, the error is <= 8192 ULP. |
| **OpExtInst cospi** | For x in the domain [-1, 1], the maximum absolute error is <= $2^{-11}$ and larger otherwise. |
| **OpExtInst exp** | <= 3 + floor( fabs(2 * x) ) ulp for the full profile, and <= 4 ulp for the embedded profile. |
| **OpExtInst exp2** | <= 3 + floor( fabs(2 * x) ) ulp for the full profile, and <= 4 ulp for the embedded profile. |
| **OpExtInst exp10** | Derived implementations implement this as exp2( x * log2(10) ). For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst expm1** | Derived implementations implement this as exp(x) - 1. For non-derived implementations, the error is <= 8192 ulp. |

Table 8.3: (continued)

| SPIR-V Instruction | Minimum Accuracy |
|---|---|
| **OpExtInst log** | For x in the domain [0.5, 2] the maximum absolute error is <= $2^{-21}$; otherwise the maximum error is <=3 ulp for the full profile and <= 4 ulp for the embedded profile |
| **OpExtInst log2** | For x in the domain [0.5, 2] the maximum absolute error is <= $2^{-21}$; otherwise the maximum error is <=3 ulp for the full profile and <= 4 ulp for the embedded profile |
| **OpExtInst log10** | For x in the domain [0.5, 2] the maximum absolute error is <= $2^{-21}$; otherwise the maximum error is <=3 ulp for the full profile and <= 4 ulp for the embedded profile |
| **OpExtInst log1p** | Derived implementations implement this as log(x + 1). For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst pow** | Undefined for x = 0 and y = 0. Undefined for x < 0 and non-integer y. Undefined for x < 0 and y outside the domain [$-2^{24}$, $2^{24}$]. For x > 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 x == 0 and nonzero y, derived implementations return zero. For non-derived implementations, the error is <= 8192 ULP. [1] |
| **OpExtInst pown** | Defined only for integer values of y. Undefined for x = 0 and y = 0. For x >= 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. |
| **OpExtInst powr** | Defined only for x >= 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. |
| **OpExtInst rootn** | Defined for x > 0 when y is non-zero, derived implementations implement this case as exp2( log2(x) / y ). Defined for x < 0 when y is odd, derived implementations implement this case as -exp2( log2(-x) / y ). Defined for x = +/-0 when y > 0, derived implementations will return +0 in this case. For non-derived implementations, the error is <= 8192 ULP. |
| **OpExtInst sin** | For x in the domain [$-\pi$, $\pi$], the maximum absolute error is <= $2^{-11}$ and larger otherwise. |
| **OpExtInst sincos** | ulp values as defined for sin(x) and cos(x). |

Table 8.3: (continued)

| SPIR-V Instruction | Minimum Accuracy |
| --- | --- |
| **OpExtInst sinh** | Defined for x in the domain [-88, 88]. For x in [$-2^{-10}$, $2^{-10}$], derived implementations implement as x. For x outside of [$-2^{-10}$, $2^{-10}$], derived implement as 0.5f * ( exp(x) - exp(-x) ). For non-derived implementations, the error is <= 8192 ULP. |
| **OpExtInst sinpi** | For x in the domain [-1, 1], the maximum absolute error is <= $2^{-11}$ and larger otherwise. |
| **OpExtInst tan** | Derived implementations implement this as sin(x) * ( 1.0f / cos(x) ). For non-derived implementations, the error is <= 8192 ulp. |
| **OpExtInst tanh** | Defined for x in the domain [-88, 88]. For x in [$-2^{-10}$, $2^{-10}$], derived implementations implement as x. For x outside of [$-2^{-10}$, $2^{-10}$], derived implementations implement as ( exp(x) - exp(-x) ) / ( exp(x) + exp(-x) ). For non-derived implementations, the error is <= 8192 ULP. |
| **OpExtInst tanpi** | 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]. |
| **OpFMul** and **OpFAdd** for $x * y + z$ | Implemented either as a correctly rounded fma or as a multiply and an add both of which are correctly rounded. |

## 8.6 Edge Case Behavior

The edge case behavior of the math functions shall conform to sections F.9 and G.6 of ISO/IEC 9899:TC 2, except where noted below in the *Additional Requirements Beyond ISO/IEC 9899:TC2 section*.

### 8.6.1 Additional Requirements Beyond ISO/IEC 9899: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.

The usual allowances for rounding error (*Relative Error as ULPs section*) or flushing behavior (*Edge Case Behavior in Flush To Zero Mode section*) shall not apply for those values for which *section F.9* of ISO/IEC 9899:,TC2, or *Additional Requirements Beyond ISO/IEC 9899:TC2* and *Edge Case Behavior in Flush To Zero Mode sections* 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 ± symbol is used, the sign shall be preserved. For example, sin(±0) = ±0 shall be interpreted to mean sin(+0) is +0 and sin(-0) is -0.

- **OpExtInst acospi**:

[1] On some implementations, `powr()` or `pown()` may perform faster than `pow()`. If `x` is known to be `>=0`, consider using `powr()` in place of `pow()`, or if `y` is known to be an integer, consider using `pown()` in place of `pow()`.

  - acospi( 1 ) = +0.
  - acospi( x ) returns a NaN for | x | > 1.
- **OpExtInst asinpi**:
  - asinpi( ±0 ) = ±0.
  - asinpi( x ) returns a NaN for | x | > 1.
- **OpExtInst atanpi**:
  - atanpi( ±0 ) = ±0.
  - atanpi ( ±∞ ) = ±0.5.
- **OpExtInst atan2pi**:
  - atan2pi ( ±0, -0 ) = ±1.
  - atan2pi ( ±0, +0 ) = ± 0.
  - atan2pi ( ±0, x ) returns ± 1 for x < 0.
  - atan2pi ( ±0, x) returns ± 0 for x > 0.
  - atan2pi ( y, ±0 ) returns -0.5 for y < 0.
  - atan2pi ( y, ±0 ) returns 0.5 for y > 0.
  - atan2pi ( ±y, -∞ ) returns ± 1 for finite y > 0.
  - atan2pi ( ±y, +∞ ) returns ± 0 for finite y > 0.
  - atan2pi ( ±∞, x ) returns ± 0.5 for finite x.
  - atan2pi (±∞, -∞ ) returns ±0.75.
  - atan2pi (±∞, +∞ ) returns ±0.25.
- **OpExtInst ceil**:
  - ceil( -1 < x < 0 ) returns -0.
- **OpExtInst cospi**:
  - cospi( ±0 ) returns 1
  - cospi( n + 0.5 ) is +0 for any integer n where n + 0.5 is representable.
  - cospi( ±∞ ) returns a NaN.
- **OpExtInst exp10**:
  - exp10( ±0 ) returns 1.
  - exp10( -∞ ) returns +0.
  - exp10( +∞ ) returns +∞.
- **OpExtInst distance**:
  - distance(x, y) calculates the distance from x to y without overflow or extraordinary precision loss due to underflow.
- **OpExtInst fdim**:
  - fdim( any, NaN ) returns NaN.
  - fdim( NaN, any ) returns NaN.
- **OpExtInst fmod**:
  - fmod( ±0, NaN ) returns NaN.

- **OpExtInst fract**:
  - 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, iptr ) returns +0 and +0 in iptr.
  - fract( -0, iptr ) returns -0 and -0 in iptr.
  - fract( +inf, iptr ) returns +0 and +inf in iptr.
  - fract( -inf, iptr ) returns -0 and -inf in iptr.
  - fract( NaN, iptr ) returns the NaN and NaN in iptr.
- **OpExtInst frexp**:
  - frexp( ±∞, exp ) returns ±∞ and stores 0 in exp.
  - frexp( NaN, exp ) returns the NaN and stores 0 in exp.
- **OpExtInst length**:
  - length calculates the length of a vector without overflow or extraordinary precision loss due to underflow.
- **OpExtInst lgamma_r**:
  - lgamma_r( x, signp ) returns 0 in signp if x is zero or a negative integer.
- **OpExtInst nextafter**:
  - nextafter( -0, y > 0 ) returns smallest positive denormal value.
  - nextafter( +0, y < 0 ) returns smallest negative denormal value.
- **OpExtInst normalize**:
  - 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];
```

- **OpExtInst pow**:
  - pow( ±0, -∞ ) returns +∞
- **OpExtInst pown**:
  - pown( x, 0 ) is 1 for any x, even zero, NaN or infinity.
  - pown( ±0, n ) is ±∞ for odd n < 0.
  - pown( ±0, n ) is +∞ for even n < 0.
  - pown( ±0, n ) is +0 for even n > 0.
  - pown( ±0, n ) is ±0 for odd n > 0.
- **OpExtInst powr**:
  - powr( x, ±0 ) is 1 for finite x > 0.
  - powr( ±0, y ) is +∞ for finite y < 0.
  - powr( ±0, -∞) is +∞.
  - powr( ±0, y ) is +0 for y > 0.

- powr( +1, y ) is 1 for finite y.
- powr( x, y ) returns NaN for x < 0.
- powr( ±0, ±0 ) returns NaN.
- powr( +∞, ±0 ) returns NaN.
- powr( +1, ±∞ ) returns NaN.
- powr( x, NaN ) returns the NaN for x >= 0.
- powr( NaN, y ) returns the NaN.

- **OpExtInst rint**:
  - rint( -0.5 <= x < 0 ) returns -0.
- **OpExtInst remquo**:
  - remquo(x, y, &quo) returns a NaN and 0 in quo if x is ±∞, or if y is 0 and the other argument is non-NaN or if either argument is a NaN.
- **OpExtInst rootn**:
  - rootn( ±0, n ) is ±∞ for odd n < 0.
  - rootn( ±0, n ) is +∞ for even n < 0.
  - rootn( ±0, n ) is +0 for even n > 0.
  - rootn( ±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.
- **OpExtInst round**:
  - round( -0.5 < x < 0 ) returns -0.
- **OpExtInst sinpi**:
  - sinpi( ±0 ) returns ±0.
  - sinpi( +n) returns +0 for positive integers n.
  - sinpi( -n ) returns -0 for negative integers n.
  - sinpi( ±∞ ) returns a NaN.
- **OpExtInst tanpi**:
  - tanpi( ±0 ) returns ±0.
  - tanpi( ±∞ ) 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 +∞ where n + 0.5 is representable.
  - tanpi( n + 0.5 ) for odd integer n is -∞ where n + 0.5 is representable.
- **OpExtInst trunc**:
  - trunc( -1 < x < 0 ) returns -0.

### 8.6.2 Changes to ISO/IEC 9899: TC2 Behavior

**OpExtInst modf** behaves as though implemented by:

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

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

### 8.6.3 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 **OpExtInst nextafter** instead of those listed in *Additional Requirements Beyond ISO/IEC 9899:TC2 section*:

- nextafter ( +smallest normal, y < +smallest normal ) = +0.
- nextafter ( -smallest normal, y > -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 < TYPE_MIN and -TYPE_MIN < x < -0. They do not include $\pm 0$. A non-zero number is said to be sub-normal before rounding if, after normalization, its radix-2 exponent is less than (TYPE_MIN_EXP - 1). [2]

[2] 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.