KHR GROUP®

OpenCL Extended Instruction Set Specification

Ben Ashbaugh, Intel

Version 1.00, Revision 8

Table of Contents

| 1. Introduction | 1 |
|---|---|
| 2. Binary Form | 5 |
| 2.1. Math extended instructions | 3 |
| 2.2. Integer instructions |) |
| 2.3. Common instructions | 3 |
| 2.4. Geometric instructions | 7 |
| 2.5. Relational instructions | 1 |
| 2.6. Vector Data Load and Store instructions | 2 |
| 2.7. Miscellaneous Vector instructions | 1 |
| 2.8. Misc instructions | 3 |
| 3. Appendix A: Changes and TBD | 1 |
| 3.1. Changes from Version 0.99, Revision 1 84 | 1 |
| 3.2. Changes from Version 0.99, Revision 2 84 | 1 |
| 3.3. Changes from Version 0.99, Revision 3 84 | 1 |
| 3.4. Changes from Version 1.0, Revision 1 84 | 1 |
| 3.5. Changes from Version 1.0, Revision 2 84 | 1 |
| 3.6. Changes from Version 1.0, Revision 3 | 5 |
| 3.7. Changes from Version 1.0, Revision 4 | 5 |
| 3.8. Changes from Version 1.0, Revision 5 | 5 |
| 3.9. Changes from Version 1.0, Revision 6 | 5 |
| 3.10. Changes from Version 1.0, Revision 7 | 5 |



Copyright 2014-2023 The Khronos Group Inc.

This Specification is protected by copyright laws and contains material proprietary to Khronos. Except as described by these terms, 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.

This Specification has been created under the Khronos Intellectual Property Rights Policy, which is Attachment A of the Khronos Group Membership Agreement available at www.khronos.org/files/member_agreement.pdf.

Khronos grants a conditional copyright license to use and reproduce the unmodified Specification for any purpose, without fee or royalty, EXCEPT no licenses to any patent, trademark or other intellectual property rights are granted under these terms. Parties desiring to implement the Specification and make use of Khronos trademarks in relation to that implementation, and receive reciprocal patent license protection under the Khronos Intellectual Property Rights Policy must become Adopters and confirm the implementation as conformant under the process defined by Khronos for this Specification; see https://www.khronos.org/adopters.

Khronos makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this Specification, including, without limitation: merchantability, fitness for a particular purpose, non-infringement of any intellectual property, correctness, accuracy, completeness, timeliness, and reliability. Under no circumstances will Khronos, 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.

This Specification contains substantially unmodified functionality from, and is a successor to, Khronos specifications including all versions of "The SPIR Specification", "The OpenGL Shading Language", "The OpenGL ES Shading Language", as well as all Khronos OpenCL API and OpenCL programming language specifications.

The Khronos Intellectual Property Rights Policy defines the terms *Scope*, *Compliant Portion*, and *Necessary Patent Claims*.

Where this Specification uses technical terminology, defined in the Glossary or otherwise, that refer to enabling technologies that are not expressly set forth in this Specification, those enabling technologies are EXCLUDED from the Scope of this Specification. For clarity, enabling technologies not disclosed with particularity in this Specification (e.g. semiconductor manufacturing technology, hardware architecture, processor architecture or microarchitecture, memory architecture, compiler technology, object oriented technology, basic operating system technology, compression technology, algorithms, and so on) are NOT to be considered expressly set forth; only those application program interfaces and data structures disclosed with particularity are included in the Scope of this Specification.

For purposes of the Khronos Intellectual Property Rights Policy as it relates to the definition of Necessary Patent Claims, all recommended or optional features, behaviors and functionality set forth in this Specification, if implemented, are considered to be included as Compliant Portions.

Khronos® and Vulkan® are registered trademarks, and ANARI[™], WebGL[™], gITF[™], NNEF[™], OpenVX[™], SPIR[™], SPIR[™], SYCL[™], OpenVG[™], Vulkan SC[™], 3D Commerce[™] and Kamaros[™] are trademarks of

The Khronos Group Inc. OpenXR[™] is a trademark owned by The Khronos Group Inc. and is registered as a trademark in China, the European Union, Japan and the United Kingdom. OpenCL[™] is a trademark of Apple Inc. used under license by Khronos. OpenGL® is a registered trademark and the OpenGL ES[™] and OpenGL SC[™] logos are trademarks of Hewlett Packard Enterprise used under license by Khronos. ASTC is a trademark of ARM Holdings PLC. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

Contributors and Acknowledgments

- Yaxun Liu, AMD
- Brian Sumner, AMD
- Marty Johnson, AMD
- Mandana Baregheh, AMD
- Andrew Richards, Codeplay
- Ben Ashbaugh, Intel
- Alexey Bader, Intel
- Guy Benyei, Intel
- Raun Krisch, Intel
- Boaz Ouriel, Intel
- Yuan Lin, NVIDIA
- Lee Howes, Qualcomm
- Chihong Zang, Qualcomm
- Ben Gaster, Qualcomm
- Jack Liu, Qualcomm
- Ronan Keryell, Xilinx

Chapter 1. Introduction

This is the specification of **OpenCL.std** extended instruction set.

The library is imported into a SPIR-V module in the following manner:

<ext-inst-id> OpExtInstImport "OpenCL.std"

The library can only be imported if Memory Model is set to OpenCL

Chapter 2. Binary Form

This section contains the semantics and exact form of execution of OpenCL extended instructions using the **OpExtInst** instruction.

In this section we use the following naming conventions:

- *void* denote an **OpTypeVoid**.
- half, float and double denote an OpTypeFloat with a width of 16, 32 and 64 bits respectively.
- i8, i16, i32 and i64 denote an OpTypeInt with a width of 8, 16, 32 and 64 bits respectively.
- *bool* denotes an **OpTypeBool**.
- *size_t* denotes an i32 if the Addressing Model is Physical32 and i64 if the Addressing Model is Physical64.
- *vector(n)* denotes an **OpTypeVector** where *n* indicates the component count.
 - vector($n_1, n_2, ..., n_i$) abbreviates vector(n_1), vector(n_2), ... or vector(n_i).
- integer denotes i8, i16, i32 or i64.
- floating-point denotes half, float, double.
- pointer(storage) denotes an OpTypePointer which points to storage Storage Class.
 - pointer(constant) denotes an OpTypePointer with UniformConstant Storage Class.
 - *pointer(generic)* denotes an OpTypePointer with **Generic Storage Class**.
 - *pointer(global)* denotes an OpTypePointer with CrossWorkgroup Storage Class.
 - *pointer(local)* denotes an OpTypePointer with **Workgroup Storage Class**.
 - pointer(private) denotes an OpTypePointer with Function Storage Class.
 - pointer(s₁, s₂, ..., s_i) abbreviates pointer(s₁), pointer(s₂), ... or pointer(s_i).
- image defines all types of image memory objects (See image encoding section).
- sampler a SPIR-V sampler object (See sampler encoding section).

2.1. Math extended instructions

This section describes the list of external math instructions. The external math instructions are categorized into the following:

- A list of instructions that have scalar or vector argument versions, and,
- A list of instructions that only take scalar float arguments.

The vector versions of the math instructions operate component-wise. The description is per-component.

The math instructions are not affected by the prevailing rounding mode in the calling environment, and always return the same value as they would if called with the round to nearest even rounding mode.

For environments that allow use of **FPFastMathMode** decorations on **OpExtInst** instructions, **FPFastMathMode** decorations may be applied to the math instructions.

| acos | |
|--|--|
| Compute the arc cosine of <i>x</i> . | |
| <i>Result</i> is an angle in radians. | |
| <i>Result Type</i> and <i>x</i> must be <i>floating-point</i> or <i>vector</i> (2,3,4,8,16) of <i>floating-point</i> values. | |

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 0 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|---|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x |

acosh

Compute the inverse hyperbolic cosine of x.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 1 | <id></id> |
|---|----|-------------|------------------|--------------|---|-----------|
| | | Result Type | | instructions | | X |
| | | | | 301 1/42 | | |

acospi

Compute **acos**(x) / pi.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 2 | <id></id> |
|---|----|-------------|------------------|----------------------|---|-----------|
| | | Result Type | | set <i><id></id></i> | | X |

Compute the arc sine of *x*.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 3 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|---|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X |
| | | | | 301 1/42 | | |

asinh

Compute the inverse hyperbolic sine of *x*.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 4 | <id></id> |
|---|----|-------------|------------------|-------------------|---|-----------|
| | | Result Type | | instructions | | x |
| | | | | set < <i>id</i> > | | |

asinpi

Compute **asin**(x) / pi.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 5 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|---|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X |

atan

Compute the arc tangent of *x*.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 6 | <id></id> |
|---|----|-------------|------------------|---------------|---|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

atan2

Compute the arc tangent of y / x.

Result is an angle in radians.

Result Type, y and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 7 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|---|-----------|-----------|
| | | Result Type | | instructions | | У | X |
| | | | | set | | | |

atanh

Compute the hyperbolic arc tangent of x.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 8 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|---|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X |

atanpi

Compute **atan**(*x*) / pi.

Result is an angle in radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 9 | <id></id> |
|---|----|-------------|------------------|---------------|---|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

atan2pi

Compute **atan**(*y*, *x*) / pi.

Result is an angle in radians.

Result Type, y and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 10 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|-----------|
| | | Result Type | | instructions | | У | X |
| | | | | set | | | |

cbrt

Compute the cube root of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 11 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x |

ceil

Round *x* to integral value using the round to positive infinity rounding mode.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 12 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

copysign

Returns *x* with its sign changed to match the sign of *y*.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 13 | <id></id> | <id></id> |
|---|----|------------------|------------------|-------------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set < <i>id</i> > | | | |

cos

Compute the cosine of x radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 14 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

cosh

Compute the hyperbolic cosine of *x* radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 15 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

cospi

Compute cos(x) / pi radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 16 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

erfc

Complementary error function of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 17 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

erf

Error function of *x* encountered in integrating the normal distribution.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 18 | < <i>id</i> > |
|---|----|---------------|------------------|---------------|----|---------------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

ехр

Compute the base-e exponential of x. (i.e. e^x)

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 19 | <id></id> |
|---|----|---------------|------------------|-----------------------------------|----|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x |

exp2

Computes 2 raised to the power of x. (i.e. 2^x)

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 20 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

exp10

Computes 10 raised to the power of x. (i.e. 10^{x})

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 21 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

expm1

Computes $e^x - 1.0$.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 22 | < <i>id</i> > |
|---|----|---------------|------------------|---------------|----|---------------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

fabs

Compute the absolute value of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 23 | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set | | |

fdim

x - y if x > y, +0 if x is less than or equal to y.

Result Type, *x* and *y* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 24 | <id></id> | <id></id> |
|---|----|------------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

floor

Round *x* to the integral value using the round to negative infinity rounding mode.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 25 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

fma

Compute the correctly rounded floating-point representation of the sum of *c* with the infinitely precise product of *a* and *b*. Rounding of intermediate products shall not occur. Edge case results are per the IEEE 754-2008 standard.

Result Type, a, b and c must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 26 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|-----------|
| | | Result Type | | instructions | | а | b | С |
| | | | | set <id></id> | | | | |

fmax

Returns *y* if x < y, otherwise it returns *x*. If one operand is a NaN, **fmax** returns the other argument. If both arguments are NaNs, **fmax** returns a NaN.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: fmax behaves as defined by C99 and may not match the IEEE 754-2008 definition for **maxNum** with regard to signaling NaNs. Specifically, signaling NaNs may behave as quiet NaNs

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 27 | <i><id></id></i> | <id></id> |
|---|----|------------------|------------------|-----------------------------------|----|------------------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | У |
| | | | | 001 102 | | | |

fmin

Returns *y* if y < x, otherwise it returns *x*. If one operand is a NaN, **fmin** returns the other argument. If both arguments are NaNs, **fmin** returns a NaN.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: fmin behaves as defined by C99 and may not match the IEEE 754-2008 definition for **minNum** with regard to signaling NaNs. Specifically, signaling NaNs may behave as quiet NaNs

| 7 | 12 | <id></id> | Result <id></id> | extended | 28 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

fmod

Modulus. Returns x - y * trunc(x/y).

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 29 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | 361 102 | | | |

fract

Returns **fmin**(*x* - **floor**(*x*), *0x1.fffffep-1f*). **floor**(*x*) is returned in *ptr*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

ptr must be a pointer(global, local, private, generic) to floating-point or vector(2,3,4,8,16) of floatingpoint values.

All of the operands, including the *Result Type* operand, must be of the same type, or must be a pointer to the same type.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 30 | <id></id> | <i><id></id></i> |
|---|----|------------------|------------------|---------------|----|-----------|------------------|
| | | Result Type | | instructions | | X | ptr |
| | | | | set <id></id> | | | |

frexp

Extract the mantissa and exponent from *x*. The *Result Type* holds the mantissa, and *exp* points to the exponent. For each component the mantissa returned is a *floating-point* with magnitude in the interval [1/2, 1) or 0. Each component of *x* equals mantissa returned * 2^{exp} .

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

exp must be a pointer(global, local, private, generic) to i32 or vector(2,3,4,8,16) of i32 values.

Result Type and *x* operands must be of the same type. *exp* operand must point to an *i*32 with the same component count as *Result Type* and *x* operands.

| 7 | 12 | <id></id> | Result <id></id> | extended | 31 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | exp |
| | | | | set | | | |

hypot

Compute the value of the square root of $x^2 + y^2$ without undue overflow or underflow.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 32 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

ilogb

Return the exponent of *x* as an *i*32 value.

Result Type must be i32 or vector(2,3,4,8,16) of i32 values.

x must be *floating-point* or *vector(2,3,4,8,16)* of *floating-point* values.

Result Type and *x* operands must have the same component count.

| 6 | 12 | <id></id> | Result <id></id> | extended | 33 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

ldexp

Multiply *x* by 2 to the power *k*.

k must be *i32* or *vector(2,3,4,8,16)* of *i32* values.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

Result Type and *x* operands must be of the same type. *k* operand must have the same component count as *Result Type* and *x* operands.

| 7 | 12 | < <i>i</i> d> | Result <id></id> | extended | 34 | <id></id> | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | k |
| | | | | set <id></id> | | | |

Igamma

Log gamma function of x. Returns the natural logarithm of the absolute value of the gamma function.

Result Type and *x* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 35 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

lgamma_r

Log gamma function of *x*. Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the *signp* operand

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

signp must be a pointer(global, local, private, generic) to i32 or vector(2,3,4,8,16) of i32 values.

Result Type and *x* operands must be of the same type. *signp* operand must point to an *i*32 with the same component count as *Result Type* and *x* operands.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 36 | <id></id> | <id></id> |
|---|----|------------------|------------------|-----------------------------------|----|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | signp |
| | | | | | | | |

log

Compute the natural logarithm of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 37 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

log2

Compute the base 2 logarithm of x.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions | 38 | <id> x</id> |
|---|----|---------------------------|------------------|--------------------------|----|-----------------|
| | | | | set <id></id> | | |

log10

Compute the base 10 logarithm of x.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 39 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

log1p

Compute $\log_{e}(1.0 + x)$.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 40 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

logb

Compute the exponent of x, which is the integral part of $\log_r |x|$.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 41 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

mad

Computes a * b + c. **mad** may compute a * b + c with reduced accuracy in the embedded profile - see the OpenCL SPIR-V Environment specification for details. On some hardware the **mad** instruction may provide better performance than the expanded computation of a * b + c.

Result Type, a, b and c must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: For some usages, e.g. mad(a, b, -a * b), the definition of mad is loose enough that almost any result is allowed from mad for some values of *a* and *b*.

| 8 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set <i><id></id></i> | 42 | <id> a</id> | <id> b</id> | <id> c</id> |
|---|----|---------------------------|------------------|--|----|-----------------|-----------------|-----------------|
|---|----|---------------------------|------------------|--|----|-----------------|-----------------|-----------------|

maxmag

Returns x if |x| > |y|, y if |y| > |x|, otherwise **fmax**(x, y).

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 43 | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x | У |

minmag

Returns x if |x| < |y|, y if |y| < |x|, otherwise **fmin**(x, y).

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 44 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

modf

Decompose a *floating-point* number. The **modf** instruction breaks the operand *x* into integral and fractional parts, each of which has the same sign as the operand. It stores the integral part in the object pointed to by *iptr*

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

iptr must be a *pointer*(*global*, *local*, *private*, *generic*) to *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

All of the operands, including the *Result Type* operand, must be of the same type, or must be a pointer to the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 45 | <id></id> | <id></id> |
|---|----|-------------|------------------|-------------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | iptr |
| | | | | set < <i>id</i> > | | | |

nan

Returns a quiet NaN. The nancode may be placed in the significand of the resulting NaN.

Result Type must be floating-point or vector(2,3,4,8,16) of floating-point values.

nancode must be integer or vector(2,3,4,8,16) of integer values.

Result Type and *nancode* operands must have the same component count. The primitive data type size of *nancode* and *Result Type* must be equal.

| 6 | 12 | <id></id> | Result <id></id> | extended | 46 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | nancode |
| | | | | set <id></id> | | |

nextafter

Computes the next representable *floating-point* value following x in the direction of y. Thus, if y is less than x, **nextafter** returns the largest representable floating-point number less than x.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 47 | <id></id> | <id></id> |
|---|----|------------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

pow

Compute *x* to the power *y*.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | < <i>id</i> > | Result <id></id> | extended | 48 | <id></id> | <id></id> |
|---|----|---------------|------------------|--------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set | | | |

pown

Compute *x* to the power *y*, where *y* is an *i*32 integer.

y must be i32 or vector(2,3,4,8,16) of i32 values.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

Result Type and *x* operands must be of the same type. *y* operand must have the same component count as *Result Type* and *x* operands.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 49 | < <i>id</i> > | < <i>id</i> > |
|---|----|------------------|------------------|-----------------------------------|----|---------------|---------------|
| | | Result Type | | instructions set < <i>id</i> > | | X | у |

powr

Compute x to the power y, where x is ≥ 0 .

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 50 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

remainder

Compute the value r such that $r = x - n^*y$, where n is the integer nearest the exact value of x/y. If there are two integers closest to x/y, n shall be the even one. If r is zero, it is given the same sign as x.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 51 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

remquo

The **remquo** instruction computes the value r such that $r = x - k^*y$, where k is the integer nearest the exact value of x/y. If there are two integers closest to x/y, k shall be the even one. If r is zero, it is given the same sign as x. This is the same value that is returned by the **remainder** instruction. **remquo** also calculates at least the lower seven bits of the integral quotient x/y, and gives that value the same sign as x/y. It stores this signed value in the object pointed to by *quo*.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

quo must be a pointer(global, local, private, generic) to i32 or vector(2,3,4,8,16) of i32 values.

Result Type, *x* and *y* operands must be of the same type. *quo* operand must point to an *i*32 with the same component count as *Result Type*, *x* and *y* operands.

| 8 1 | 12 | <id></id> | Result <id></id> | extended | 52 | <id></id> | <id></id> | <id></id> |
|-----|----|-------------|------------------|-----------------------------------|----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x | У | quo |

rint

Round *x* to integral value (using round to nearest even rounding mode) in floating-point format.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 53 | < <i>i</i> d> |
|---|----|-------------|------------------|-----------------------------------|----|---------------|
| | | Result Type | | instructions set < <i>id</i> > | | X |
| | | | | set <id></id> | | |

rootn

Compute x to the power 1/y.

y must be *i*32 or *vector*(2,3,4,8,16) of *i*32 values.

Result Type and *x* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

Result Type and *x* operands must be of the same type. *y* operand must have the same component count as *Result Type* and *x* operands.

| 7 | 12 | < <i>id</i> > | Result <id></id> | extended | 54 | <id></id> | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

round

Return the integral value nearest to *x* rounding halfway cases away from zero, regardless of the current rounding direction.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 55 | < <i>i</i> d> |
|---|----|-------------|------------------|---------------|----|---------------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

rsqrt

Compute inverse square root of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set < <i>id</i> > | 56 | <id> x</id> |
|---|----|---------------------------|------------------|---|----|-----------------|
| | | | | set | | |

sin

Compute sine of *x* radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 57 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X |
| | | | | 001 102 | | |

sincos

Compute sine and cosine of *x* radians. The computed sine is the return value and computed cosine is returned in *cosval*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

cosval must be a pointer(global, local, private, generic) to floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type, or must be a pointer to the same type.

| 7 | 12 | < <i>id</i> > | Result <id></id> | extended | 58 | <i><id></id></i> | <id></id> |
|---|----|---------------|------------------|-----------------------------------|----|------------------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x | cosval |

sinh

Compute hyperbolic sine of x radians.

Result Type and *x* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 59 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X |

sinpi

Compute sin (pi x) radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 60 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

sqrt

Compute square root of x.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 61 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

tan

Compute tangent of *x* radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 62 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

tanh

Compute hyperbolic tangent of x radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 63 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

tanpi

Compute tan (pi x) radians.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 64 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

tgamma

Compute the gamma function of *x*.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 65 | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x |

trunc

Round *x* to integral value using the round to zero rounding mode.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 66 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_cos

Compute cosine of x radians. The resulting value is undefined if x is not in the range -2^{16} ... $+2^{16}$.ha

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 67 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

half_divide

Compute x / y.

Result Type, x and y must be float or vector(2,3,4,8,16) of float values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 68 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |
half_exp

Compute the base-e exponential of *x*.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 69 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

half_exp2

Compute the base 2 exponential of *x*.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 70 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_exp10

Compute the base 10 exponential of *x*.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 71 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_log

Compute the natural logarithm of *x*.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 72 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

half_log2

Compute the base 2 logarithm of x.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 73 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_log10

Compute the base 10 logarithm of *x*.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 74 | <id></id> |
|---|----|-------------|------------------|-------------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set < <i>id</i> > | | |

half_powr

Compute *x* to the power *y*, where x is >= 0.

Result Type, x and y must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 75 | <id></id> | <id></id> |
|---|----|-------------|------------------|-------------------|----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set < <i>id</i> > | | | |

half_recip

Compute the reciprocal of *x*.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 76 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

half_rsqrt

Compute the inverse square root of *x*.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the Result Type operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 77 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

half_sin

Compute the sine of x radians. The resulting value is undefined if x is not in the range $2^{16} \dots + 2^{16}$.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 78 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_sqrt

Compute the square root of *x*.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 79 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

half_tan

Compute tangent value of x radians. The resulting values are undefined if x is not in the range $-2^{16} \dots +2^{16}$.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 80 | <id></id> |
|---|----|---------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_cos

Compute cosine of x radians over an implementation-defined range. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 6 | 12 | <id></id> | Result <id></id> | extended | 81 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_divide

Compute x / y over an implementation-defined range. The maximum error is implementation-defined.

Result Type, *x* and *y* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 7 | 12 | <id></id> | Result <id></id> | extended | 82 | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | x | У |

native_exp

Compute the base-e exponential of x over an implementation-defined range. The maximum error is implementation-defined.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 83 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_exp2

Compute the base- 2 exponential of *x* over an implementation-defined range. The maximum error is implementation-defined..

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 6 | 12 | <id></id> | Result <id></id> | extended | 84 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_exp10

Compute the base- 10 exponential of x over an implementation-defined range. The maximum error is implementation-defined..

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 85 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_log

Compute natural logarithm of x over an implementation-defined range. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 6 | 12 | <id></id> | Result <id></id> | extended | 86 | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|
| | | Result Type | | instructions | | Х |
| | | | | set | | |

native_log2

Compute a base 2 logarithm of x over an implementation-defined range. The maximum error is implementation-defined.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 87 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_log10

Compute a base 10 logarithm of x over an implementation-defined range. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 3 <i><id></id></i> |
|--------------------|
| X |
| 3 |

native_powr

Compute x to the power y, where x is >= 0.

Result Type, x and y must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 89 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

native_recip

Compute reciprocal of x over an implementation-defined range. The range of x and y are implementation-defined. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 6 | 12 | <id></id> | Result <id></id> | extended | 90 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

native_rsqrt

Compute inverse square root of x over an implementation-defined range. The maximum error is implementation-defined.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 91 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

native_sin

Compute sine of *x* radians over an implementation-defined range. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction may map to one or more native device instructions and typically has better performance compared to the corresponding non-native instruction. Support for denormal values is implementation-defined for native instructions.

| 6 | 12 | <id></id> | Result <id></id> | extended | 92 | <id></id> |
|---|----|-------------|------------------|--------------|----|-----------|
| | | Result Type | | instructions | | X |
| | | | | 301 1/1/2 | | |

native_sqrt

Compute the square root of *x* over an implementation-defined range. The maximum error is implementation-defined.

Result Type and *x* must be *float* or *vector*(2,3,4,8,16) of *float* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 93 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

native_tan

Compute tangent value of *x* radians over an implementation-defined range. The maximum error is implementation-defined.

Result Type and x must be float or vector(2,3,4,8,16) of float values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 94 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

2.2. Integer instructions

This section describes the list of integer instructions that take scalar or vector arguments. The vector versions of the integer instructions operate component-wise. The description is per-component.

s_abs

Returns |x|, where x is treated as signed integer.

Result Type and x must be integer or vector(2,3,4,8,16) of integer values.

All of the operands, including the *Result Type* operand, must be of the same type.

This instruction can be decorated with **NoSignedWrap**.

| 6 | 12 | <id></id> | Result <id></id> | extended | 141 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

s_abs_diff

Returns |x - y| without modulo overflow, where x and y are treated as signed integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the Result Type operand, must be of the same type.

| 7 | 12 | < <i>i</i> d> | Result <id></id> | extended | 142 | <id></id> | < <i>i</i> d> |
|---|----|---------------|------------------|--------------|-----|-----------|---------------|
| | | Result Type | | instructions | | X | У |
| | | | | set | | | |

s_add_sat

Returns the saturated value of x + y, where x and y are treated as signed integers.

Result Type, x and y must be integer or vector(2,3,4,8,16) of integer values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 143 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

u_add_sat

Returns the saturated value of x + y, where x and y are treated as unsigned integers.

Result Type, x and y must be integer or vector(2,3,4,8,16) of integer values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 144 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|-----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set | | | |

s_hadd

Returns the value of (x + y) >> 1, where x and y are treated as signed integers. The intermediate sum does not modulo overflow.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 145 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|-----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set | | | |

u_hadd

Returns the value of (x + y) >> 1, where x and y are treated as unsigned integers. The intermediate sum does not modulo overflow.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 146 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|-----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set | | | |

s_rhadd

Returns the value of (x + y + 1) >> 1, where x and y are treated as signed integers. The intermediate sum does not modulo overflow.

Result Type, x and y must be integer or vector(2,3,4,8,16) of integer values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 147 | < <i>i</i> d> | <i><id></id></i> |
|---|----|-------------|------------------|---------------|-----|---------------|------------------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

u_rhadd

Returns the value of (x + y + 1) >> 1, where x and y are treated as unsigned integers. The intermediate sum does not modulo overflow.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 148 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

s_clamp

Returns *s_min*(*s_max*(*x,minval*),*maxval*), where *x*, *minval*, and *maxval* are treated as signed integers. The resulting values are undefined if *minval* > *maxval*.

Result Type, x, minval and maxval must be integer or vector(2,3,4,8,16) of integer values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 8 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set <i><id></id></i> | 149 | <id> x</id> | <id> minval</id> | <id> maxval</id> |
|---|----|---------------------------|------------------|--|-----|-----------------|----------------------|----------------------|
|---|----|---------------------------|------------------|--|-----|-----------------|----------------------|----------------------|

u_clamp

Returns $u_{min}(u_{max}(x, minval), maxval)$, where x, minval, and maxval are treated as unsigned integers. The resulting values are undefined if minval > maxval.

Result Type, x, minval and maxval must be integer or vector(2,3,4,8,16) of integer values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 150 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | minval | maxval |

clz

Returns the number of leading 0 bits in x, starting at the most significant bit position. If x is 0, returns the size in bits of the type of x or component type of x, if x is a vector.

Result Type and *x* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 151 | <id></id> |
|---|----|---------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

ctz

Returns the count of trailing 0 bits in x. If x is 0, returns the size in bits of the type of x or component type of x, if x is a vector.

Result Type and *x* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 152 | <id></id> |
|---|----|-------------|------------------|-------------------|-----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set < <i>id</i> > | | |

s_mad_hi

Returns $mul_h(a, b) + c$, where a, b and c are treated as signed integers.

Result Type, a, b and c must be integer or vector(2,3,4,8,16) of integer values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 8 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set < <i>id</i> > | 153 | <id> a</id> | <id> b</id> | <id> c</id> |
|---|----|---------------------------|------------------|---|-----|-----------------|-----------------|-----------------|
| | | | | 361 <10> | | | | |

u_mad_sat

Returns x * y + z and saturates the result where x, y and z are treated as unsigned integers.

Result Type, *x*, *y* and *z* must be *integer* or *vector*(*2*,*3*,*4*,*8*,*16*) of *integer* values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 154 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | У | Ζ |

s_mad_sat

Returns x * y + z and saturates the result where x, y and z are treated as signed integers.

Result Type, *x*, *y* and *z* must be *integer* or *vector*(*2*,*3*,*4*,*8*,*16*) of *integer* values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 155 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set <i><id></id></i> | | X | У | Ζ |
| | | | | | | | | |

s_max

Returns y if x < y, otherwise it returns x, where x and y are treated as signed integers.

Result Type, x and y must be integer or vector(2,3,4,8,16) of integer values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 1 | 12 | <id></id> | Result <id></id> | extended | 156 | <id></id> | <id></id> |
|-----|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | У |

u_max

Returns y if x < y, otherwise it returns x, where x and y are treated as unsigned integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 157 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

s_min

Returns y if y < x, otherwise it returns x, where x and y are treated as signed integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | < <i>i</i> d> | Result <id></id> | extended | 158 | <id></id> | <id></id> |
|---|----|---------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

u_min

Returns y if y < x, otherwise it returns x, where x and y are treated as unsigned integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 159 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

s_mul_hi

Computes x * y and returns the high half of the product of x and y, where x and y are treated as signed integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | < <i>id</i> > | Result <id></id> | extended | 160 | < <i>i</i> d> | < <i>i</i> d> |
|---|----|---------------|------------------|-------------------|-----|---------------|---------------|
| | | Result Type | | instructions | | X | У |
| | | | | set < <i>id</i> > | | | |

rotate

For each element in v, the bits are shifted left by the number of bits given by the corresponding element in *i*. Bits shifted off the left side of the element are shifted back in from the right.

Result Type, *v* and *i* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | < <i>id</i> > | Result <id></id> | extended | 161 | <id></id> | < <i>i</i> d> |
|---|----|---------------|------------------|---------------|-----|-----------|---------------|
| | | Result Type | | instructions | | V | i |
| | | | | set <id></id> | | | |

s_sub_sat

Returns the saturated value of x - y, where x and y are treated as signed integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 162 | <i><id></id></i> | <id></id> |
|---|----|-------------|------------------|---------------|-----|------------------|-----------|
| | | Result Type | | instructions | | X | У |
| | | | | set <id></id> | | | |

u_sub_sat

Returns the saturated value of x - y, where x and y are treated as unsigned integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 163 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

u_upsample

If hi and lo component type is i8:

Result = ((upcast...to i16)*hi* << 8) | *lo*

If hi and lo component type is i16:

Result = ((upcast...to i32)hi << 16) | lo

If *hi* and *lo* component i32:

Result = ((upcast...to i64)*hi* << 32) | *lo*

hi and lo are treated as unsigned integers.

hi and *lo* must be *i8*, *i16* or *i32* or *vector*(*2*,*3*,*4*,*8*,*16*) of *i8*, *i16* or *i32* values.

Result Type must be *i16*, *i32* or *i64* or *vector*(*2*, *3*, *4*, *8*, *16*) of *i16*, *i32* or *i64* values.

hi and *lo* operands must be of the same type. If *hi* and *lo* component type is i8, the *Result Type* component type must be i16. If *hi* and *lo* component type is i16, the *Result Type* component type must be i32. If *hi* and *lo* component type is i32, the *Result Type* component type must be i64. *Result Type* must have the same component count as *hi* and *lo* operands.

| 7 | 12 | <id></id> | Result <id></id> | extended | 164 | <id></id> | < <i>i</i> d> |
|---|----|-------------|------------------|---------------|-----|-----------|---------------|
| | | Result Type | | instructions | | hi | lo |
| | | | | set <id></id> | | | |

s_upsample

If hi and lo component type is i8:

Result = ((upcast...to i16)*hi* << 8) | *lo*

If hi and lo component type is i16:

Result = ((upcast...to i32)hi << 16) | lo

If *hi* and *lo* component i32:

Result = ((upcast...to i64) hi << 32) | lo

hi is treated as a signed integer and lo is treated as an unsigned integer.

hi and *lo* must be *i8*, *i16* or *i32* or *vector*(*2*,*3*,*4*,*8*,*16*) of *i8*, *i16* or *i32* values.

Result Type must be *i16*, *i32* or *i64* or *vector*(*2*,*3*,*4*,*8*,*16*) of *i16*, *i32* or *i64* values.

hi and *lo* operands must be of the same type. If *hi* and *lo* component type is i8, the *Result Type* component type must be i16. If *hi* and *lo* component type is i16, the *Result Type* component type must be i32. If *hi* and *lo* component type is i32, the *Result Type* component type must be i64. *Result Type* must have the same component count as *hi* and *lo* operands.

| 7 | 12 | <id></id> | Result <id></id> | extended | 165 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | Instructions | | ni | 10 |
| | | | | set <id></id> | | | |
| | | | | | | | |

popcount

Returns the number of non-zero bits in x.

Result Type and *x* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 166 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

s_mad24

Multiply two 24-bit integer values x and y and add the 32-bit integer result to the 32-bit integer z. Refer to definition of s_mul24 to see how the 24-bit integer multiplication is performed.

Result Type, *x*, *y* and *z* must be *i*32 or *vector*(*2*,*3*,*4*,*8*,*16*) of *i*32 values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 8 | 12 | <id></id> | Result <id></id> | extended | 167 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-------------------|-----|-----------|-----------|-----------|
| | | Result Type | | set < <i>id</i> > | | * | у | 2 |

u_mad24

Multiply two 24-bit integer values x and y and add the 32-bit integer result to the 32-bit integer z. Refer to definition of u_mul24 to see how the 24-bit integer multiplication is performed.

Result Type, *x*, *y* and *z* must be *i*32 or *vector*(*2*,*3*,*4*,*8*,*16*) of *i*32 values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 8 | 12 | <id></id> | Result <id></id> | extended | 168 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions | | X | У | Ζ |
| | | | | set <id></id> | | | | |

s_mul24

Multiply two 24-bit integer values x and y, where x and y are treated as signed integers. x and y are 32-bit integers but only the low-order 24 bits are used to perform the multiplication. s_mul24 should only be used if values in x and y are in the range [- 2^{23} , 2^{23} -1]. If x and y are not in this range, the multiplication result is implementation-defined.

Result Type, *x* and *y* must be *i*32 or *vector*(2,3,4,8,16) of *i*32 values.

| 7 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set <i><id></id></i> | 169 | <id> x</id> | <id> y</id> |
|---|----|---------------------------|------------------|--|-----|-----------------|-----------------|
|---|----|---------------------------|------------------|--|-----|-----------------|-----------------|

u_mul24

Multiply two 24-bit integer values x and y, where x and y are treated as unsigned integers. x and y are 32bit integers but only the low-order 24 bits are used to perform the multiplication. u_mul24 should only be used if values in x and y are in the range [0, 2^{24} -1]. If x and y are not in this range, the multiplication result is implementation-defined.

Result Type, *x* and *y* must be *i32* or *vector*(*2*,*3*,*4*,*8*,*16*) of *i32* values.

All of the operands, including the Result Type operand, must be of the same type.

| 7 | 12 | <i><id></id></i> | Result <id></id> | extended | 170 | < <i>i</i> d> | <i><id></id></i> |
|---|----|------------------|------------------|-----------------------------------|-----|---------------|------------------|
| | | Result Type | | instructions set < <i>id</i> > | | x | У |

u_abs

Returns |x|, where x is treated as unsigned integer.

Result Type and *x* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 201 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | x |
| | | | | set <id></id> | | |

u_abs_diff

Returns |x - y| without modulo overflow, where x and y are treated as unsigned integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 202 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

u_mul_hi

Computes x * y and returns the high half of the product of x and y, where x and y are treated as unsigned integers.

Result Type, *x* and *y* must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions | 203 | <id> x</id> | <id> y</id> |
|---|----|---------------------------|------------------|-----------------------|-----|-----------------|-----------------|
| | | | | set < <i>id</i> > | | | |

u_mad_hi

Returns $mul_h(a, b) + c$, where a, b and c are treated as unsigned integers.

Result Type, a, b and c must be integer or vector(2,3,4,8,16) of integer values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 204 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | а | b | С |

2.3. Common instructions

This section describes the list of common instructions that take scalar or vector arguments. The vector versions of the integer instructions operate component-wise. The description is per-component. The common instructions are implemented using the round to nearest even rounding mode.

For environments that allow use of **FPFastMathMode** decorations on **OpExtInst** instructions, **FPFastMathMode** decorations may be applied to the common instructions.

fclamp

Returns *fmin(fmax(x, minval), maxval*). The resulting values are undefined if *minval* > *maxval*.

Result Type, x, minval and maxval must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 8 1 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set <i><id></id></i> | 95 | <id> x</id> | <id> minval</id> | <id> maxval</id> |
|-----|----|---------------------------|------------------|--|----|-----------------|----------------------|----------------------|
|-----|----|---------------------------|------------------|--|----|-----------------|----------------------|----------------------|

degrees

Converts radians to degrees, i.e. (180 / pi) * radians.

Result Type and *radians* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 96 | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|
| | | Result Type | | instructions | | radians |
| | | | | set <id></id> | | |

fmax_common

Returns y if x < y, otherwise it returns x. If x or y are infinite or NaN, the resulting values are undefined.

Result Type, *x* and *y* must be *floating-point* or *vector*(*2*,*3*,*4*,*8*,*16*) of *floating-point* values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 97 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|----|-----------|-----------|
| | | Result Type | | instructions | | x | У |
| | | | | set <id></id> | | | |

fmin_common

Returns y if y < x, otherwise it returns x. If x or y are infinite or NaN, the resulting values are undefined.

Result Type, x and y must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 7 | 12 | <id></id> | Result <id></id> | extended | 98 | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | У |

mix

Returns the linear blend of x & y implemented as:

x + (y - x) * a

Result Type, x, y and a must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction can be implemented using contractions such as mad or fma.

| 8 | 12 | <id></id> | Result <id></id> | extended | 99 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | X | У | а |

radians

Converts degrees to radians, i.e. (pi / 180) * degrees.

Result Type and *degrees* must be *floating-point* or *vector*(2,3,4,8,16) of *floating-point* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 100 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | degrees |
| | | | | set <id></id> | | |

step

Returns 0.0 if x < edge, otherwise it returns 1.0.

Result Type, edge and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the Result Type operand, must be of the same type.

| Result Type instructions edge x | 7 | 12 | <id></id> | Result <id></id> | extended | 101 | <id></id> | <id></id> |
|---------------------------------|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|
| set | | | Result Type | | instructions set < <i>id</i> > | | edge | x |

smoothstep

Returns 0.0 if $x \le edge_0$ and 1.0 if $x \ge edge_1$ and performs smooth Hermite interpolation between 0 and 1, if $edge_0 < x < edge_1$.

This is equivalent to :

```
t = fclamp((x - edge_0) / (edge_1 - edge_0), 0, 1);
```

return t * t * (3 - 2 * t);

The resulting values are undefined if $edge_0 >= edge_1$ or if x, $edge_0$ or $edge_1$ is a NaN.

Result Type, $edge_0$, $edge_1$ and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

Note: This instruction can be implemented using contractions such as mad or fma.

| 8 | 12 | <id></id> | Result <id></id> | extended | 102 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-------------------|-----------|-----------|
| | | Result Type | | instructions | | edge ₀ | edge₁ | x |
| | | | | set <id></id> | | | | |

sign

Returns 1.0 if x > 0, -0.0 if x = -0.0, +0.0 if x = +0.0, or -1.0 if x < 0. Returns 0.0 if x is a NaN.

Result Type and x must be floating-point or vector(2,3,4,8,16) of floating-point values.

| 6 | 12 | < <i>i</i> d> | Result <id></id> | extended | 103 | <id></id> |
|---|----|---------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | X |
| | | | | set <id></id> | | |

2.4. Geometric instructions

This section describes the list of geometric instructions. In this section x, y, z and w denote the first, second, third and fourth component respectively, of vectors with 3 and four components. The geometric instructions are implemented using the round to nearest even rounding mode.

Note: The geometric instructions can be implemented using contractions such as mad or fma

For environments that allow use of **FPFastMathMode** decorations on **OpExtInst** instructions, **FPFastMathMode** decorations may be applied to the geometric instructions.

| cros | cross | | | | | | | | | |
|--|--|---------------------------|------------------|---|-----|-----------------------------|------------------|--|--|--|
| Retu | Returns the cross product of p_0 .xyz and p_1 .xyz. | | | | | | | | | |
| lf the | If the vector component count is 4, the w component returned is 0.0. | | | | | | | | | |
| <i>Result Type</i> , p_0 and p_1 must be <i>vector(3,4)</i> of <i>floating-point</i> values. All of the operands, including the <i>Result Type</i> operand, must be of the same type. | | | | | | | | | | |
| 7 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set < <i>id</i> > | 104 | <id> p₀</id> | <id> p1</id> | | | |

distance

Returns the distance between p_0 and p_1 . This is calculated as *length*($p_0 - p_1$).

Result Type must be floating-point.

 p_0 and p_1 must be floating-point or vector(2,3,4) of floating-point values.

 p_0 and p_1 operands must have the same type. *Result Type*, p_0 and p_1 operands must have the same component type

| 7 | 12 | <id></id> | Result <id></id> | extended | 105 | <id></id> | < <i>i</i> d> |
|---|----|-------------|------------------|--------------|-----|-----------|-----------------------|
| | | Result Type | | instructions | | p_o | <i>p</i> ₁ |
| | | | | set | | | |

length

Return the length of vector p, i.e. sqrt($p.x^2 + p.y^2 + ...$)

Result Type must be floating-point.

p must be *floating-point* or *vector*(2,3,4) of *floating-point* values.

Result Type and p operands must have the same component type

| 6 | 12 | <id></id> | Result <id></id> | extended | 106 | <id></id> |
|---|----|-------------|------------------|--------------|-----|-----------|
| | | Result Type | | instructions | | р |
| | | | | set | | |

normalize

Returns a vector in the same direction as *p* but with a length of 1.

Result Type and p must be floating-point or vector(2,3,4) of floating-point values.

All of the operands, including the *Result Type* operand, must be of the same type.

| 6 | 12 | <id></id> | Result <id></id> | extended | 107 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | p |
| | | | | set <id></id> | | |

fast_distance

Returns *fast_length*($p_0 - p_1$).

Result Type must be floating-point.

 p_0 and p_1 must be floating-point or vector(2,3,4) of floating-point values.

 p_0 and p_1 operands must have the same type. *Result Type*, p_0 and p_1 operands must have the same component type

| 7 | 12 | <id></id> | Result <id></id> | extended | 108 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | p_0 | p_1 |
| | | | | set <id></id> | | | |

fast_length

Return the length of vector *p* computed as: *half_sqrt*($p.x^2 + p.y^2 + ...$)

Result Type must be floating-point.

p must be *vector*(2,3,4) of *floating-point* values.

Result Type and *p* operands must have the same component type

| 6 | 12 | <id></id> | Result <id></id> | extended | 109 | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|
| | | Result Type | | instructions | | p |
| | | | | set <id></id> | | |

fast_normalize

Returns a vector in the same direction as *p* but with a length of 1 computed as:

 $p * half_rsqrt(p.x^2 + p.y^2...)$

The result shall be within 8192 ulps error from the infinitely precise result of:

if (all(p == 0.0f)) { result = p; }

else { result = $p / sqrt(p.x^2 + p.y^2 + ...); }$

with the following exceptions :

1) If the sum of squares is greater than FLT_MAX then the value of the floating-point values in the result vector are undefined.

2) If the sum of squares is less than FLT_MIN then the implementation may return back *p*.

3) If the device is in "denorms are flushed to zero" mode, individual operand elements with magnitude less than *sqrt*(FLT_MIN) may be flushed to zero before proceeding with the calculation.

Result Type and *p* must be *floating-point* or *vector(2,3,4)* of *floating-point* values.

| 6 | 12 | <id></id> | Result <id></id> | extended | 110 | <id></id> |
|---|----|-------------|------------------|------------------|-----|-----------|
| | | Result Type | | instructions set | | p |
| | | | | <id></id> | | |
2.5. Relational instructions

This section describes the list of relational instructions that take scalar or vector arguments. The vector versions of the integer instructions operate component-wise. The description is per-component.

bitselect

Each bit of the result is the corresponding bit of *a* if the corresponding bit of *c* is 0. Otherwise it is the corresponding bit of *b*.

Result Type, a, b and c must be floating-point or integer or vector(2,3,4,8,16) of floating-point or integer values.

All of the operands, including the Result Type operand, must be of the same type.

| 8 | 12 | <id></id> | Result <id></id> | extended | 186 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | а | b | С |

select

For each component of a vector type, the result is *a* if the most significant bit of *c* is zero, otherwise it is *b*.

For a scalar type, the result is *a* if *c* is zero, otherwise it is *b*.

c must be *integer* or *vector*(2,3,4,8,16) of *integer* values.

Result Type, a and b must be floating-point or integer or vector(2,3,4,8,16) of floating-point or integer values.

Result Type, *a* and *b* must have the same type. *c* operand must have the same component count and component bit width as the rest of the operands.

| 8 | 12 | <id></id> | Result <id></id> | extended | 187 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | а | b | С |

2.6. Vector Data Load and Store instructions

This section describes the list of instructions that allow reading and writing of vector types from a pointer to memory.

For environments that allow use of **FPFastMathMode** decorations on **OpExtInst** instructions, **FPFastMathMode** decorations may be applied to vector data load and store instructions that convert to or from *half* values.

vloadn

Reads *n* components from the address computed as (p + (offset * n)) and creates a vector result value from the *n* components.

Behavior is undefined if the computed address is not 8-bit aligned when p points to an i8 value; 16-bit aligned when p points to an i16 or half value; 32-bit aligned when p points to an i32 or float value; 64-bit aligned when p points to an i64 or double value.

offset must be size_t.

p must be a pointer(global, local, private, constant, generic) to floating-point, integer.

Result Type must be *vector*(2,3,4,8,16) of *floating-point* or *integer* values.

Result Type component count must be equal to *n* and its component type must be equal to the type pointed by *p*.

n must be 2, 3, 4, 8 or 16.

| 8 | 12 | <id></id> | Result <id></id> | extended | 171 | <id></id> | <id></id> | Literal |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|---------|
| | | Result Type | | instructions | | offset | p | n |
| | | | | set <id></id> | | | | |

vstoren

Writes *n* components from the *data* vector value to the address computed as (p + (offset * n)), where *n* is equal to the component count of the vector *data*.

Behavior is undefined if the computed address is not 8-bit aligned when p points to an i8 value; 16-bit aligned when p points to an i16 or half value; 32-bit aligned when p points to an i32 or float value; 64-bit aligned when p points to an i64 or double value.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to floating-point, integer.

data must be vector(2,3,4,8,16) of floating-point or integer values.

data component type must be equal to the type pointed by p.

| 8 | 12 | <id></id> | Result <id></id> | extended | 172 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | data | offset | p |

vload_half

Reads a half value from the address computed as (p + (offset)) and converts it to a float result value.

Behavior is undefined if the computed address is not 16-bit aligned.

Result Type must be float.

offset must be size_t.

p must be a pointer(global, local, private, constant, generic) to half.

| 7 | 12 | <id></id> | Result <id></id> | extended | 173 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|
| | | Result Type | | instructions | | offset | p |
| | | | | set <id></id> | | | |

vload_halfn

Reads *n* half components from the address (p + (offset * n)), converts to *n* float components, and creates a float vector result value from the *n* float components.

Behavior is undefined if the computed address is not 16-bit aligned.

offset must be size_t.

p must be a pointer(global, local, private, constant, generic) to half.

Result Type must be vector(2,3,4,8,16) of float values.

Result Type component count must be equal to *n*.

n must be 2, 3, 4, 8 or 16.

| 8 | 12 | <id> Result Type</id> | Result <id></id> | extended instructions set <i><id></id></i> | 174 | <id> offset</id> | <id> p</id> | Literal n |
|---|----|---------------------------|------------------|--|-----|----------------------|-----------------|--------------|
|---|----|---------------------------|------------------|--|-----|----------------------|-----------------|--------------|

vstore_half

Converts the *data* float or double value to a half value using the default rounding mode and writes the half value to the address computed as (p + offset).

Behavior is undefined if the computed address is not 16-bit aligned.

data must be float or double.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 8 | 12 | <id></id> | Result <id></id> | extended | 175 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | data | offset | p |

vstore_half_r

Converts the *data* float or double value to a half value using the specified rounding mode *mode* and writes the half value to the address computed as (p + offset).

Behavior is undefined if the computed address is not 16-bit aligned.

data must be float or double.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 9 | 12 | <id></id> | Result | extended | 176 | <id></id> | <id></id> | <id></id> | FP |
|---|----|-----------|-----------|---------------------|-----|-----------|-----------|-----------|----------|
| | | Result | <id></id> | instruction | | data | offset | p | Rounding |
| | | Туре | | s set < <i>id</i> > | | | | | Mode |
| | | | | | | | | | mode |

vstore_halfn

Converts the *data* vector of float or vector of double values to a vector of half values using the default rounding mode and writes the half values to memory.

Let *n* be the component count of the vector *data*.

The *n* components from the converted vector of half values are written to the address computed as (p + (offset * n)).

Behavior is undefined if the computed address is not 16-bit aligned.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 8 | 12 | <id></id> | Result <id></id> | extended | 177 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions | | data | offset | p |
| | | | | set <id></id> | | | | |

vstore_halfn_r

Converts the *data* vector of float or vector of double values to a vector of half values using the specified rounding mode *mode* and writes the half values to memory.

Let *n* be the component count of the vector *data*.

The *n* components from the converted vector of half values are written to the address computed as (p + (offset * n)).

Behavior is undefined if the computed address is not 16-bit aligned.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 9 | 12 | <id></id> | Result | extended | 178 | <id></id> | <id></id> | <id></id> | FP |
|---|----|-----------|-----------|---------------------|-----|-----------|-----------|-----------|----------|
| | | Result | <id></id> | instruction | | data | offset | p | Rounding |
| | | Туре | | s set < <i>id</i> > | | | | | Mode |
| | | | | | | | | | mode |

vloada_halfn

Reads a vector of *n* half values from aligned memory and converts it to a float vector result value.

For *n* equal to 2, 4, 8, and 16, the vector of *n* half values is read from the address computed as (p + (offset * n)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * n) bytes.

For *n* equal to 3, the vector of *n* half values are read from the address computed as (p + (offset * 4)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * 4) bytes.

offset must be size_t.

p must be a pointer(global, local, private, constant, generic) to half.

Result Type must be *vector*(2,3,4,8,16) of *float* values.

Result Type component count must be equal to n.

n must be 2, 3, 4, 8 or 16.

| 8 | 12 | <id></id> | Result <id></id> | extended | 179 | <id></id> | <id></id> | Literal |
|---|----|-------------|------------------|---------------|-----|-----------|-----------|---------|
| | | Result Type | | instructions | | offset | p | n |
| | | | | set <id></id> | | | | |

vstorea_halfn

Converts the *data* vector of float or vector of double values to a vector of half values using the default rounding mode, and then writes the converted vector of half values to aligned memory.

Let *n* be the component count of the vector *data*.

For *n* equal to 2, 4, 8, and 16, the converted vector of half values is written to the address computed as (p + (offset * n)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * n) bytes.

For *n* equal to 3, the converted vector of half values is written to the address computed as (p + (offset * 4)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * 4) bytes.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 8 | 12 | <id></id> | Result <id></id> | extended | 180 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|-----------|
| | | Result Type | | instructions set < <i>id</i> > | | data | offset | р |

vstorea_halfn_r

Converts the *data* vector of float or vector of double values to a vector of half values using the specified rounding mode *mode*, and then write the converted vector of half values to aligned memory.

Let *n* be the component count of the vector *data*.

For *n* equal to 2, 4, 8, and 16, the converted vector of half values is written to the address computed as (p + (offset * n)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * n) bytes.

For *n* equal to 3, the converted vector of half values is written to the address computed as (p + (offset * 4)). Behavior is undefined if the computed address is not aligned to (sizeof(half) * 4) bytes.

offset must be size_t.

Result Type must be void.

p must be a pointer(global, local, private, generic) to half.

| 9 | 12 | <id> Result Type</id> | Result <id></id> | extended instruction s set < <i>id</i> > | 181 | <id> data</id> | <id> offset</id> | <id> p</id> | FP Rounding Mode mode |
|---|----|-------------------------------|---------------------|--|-----|--------------------|----------------------|-----------------|--------------------------------|
|---|----|-------------------------------|---------------------|--|-----|--------------------|----------------------|-----------------|--------------------------------|

2.7. Miscellaneous Vector instructions

This section describes additional vector instructions.

shuffle

Construct a permutation of components from *x* vector value, returning a vector value with the same component type as *x* and component count that is the same as *shuffle mask*.

For this instruction, only the *ilogb*(2 m - 1) least significant bits of each mask element are considered, where *m* is equal to the component count of *x*.

shuffle mask operand specifies, for each component in the result vector, which component of x it gets.

The size of each component in *shuffle mask* must match the size of each component in *Result Type*.

Result Type must have the same component type as *x* and component count as *shuffle mask*.

shuffle mask must be *vector(2,4,8,16)* of *integer* values.

Result Type and x must be vector(2,4,8,16) of floating-point or integer values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 182 | <id></id> | <id></id> |
|---|----|-------------|------------------|---------------|-----|-----------|--------------|
| | | Result Type | | instructions | | x | shuffle mask |
| | | | | set <id></id> | | | |

shuffle2

Construct a permutation of components from *x* and *y* vector values, returning a vector value with the same component type as *x* and *y* and component count that is the same as *shuffle mask*.

For this instruction, only the *ilogb*(2 m - 1) + 1 least significant bits of each mask component are considered, where *m* is equal to the component count of *x* and *y*.

shuffle mask operand specifies, for each component in the result vector, which component of x or y it gets. Where component count begins with x and then proceeds to y.

x and y must be of the same type.

The size of each component in shuffle mask must match the size of each component in Result Type.

Result Type must have the same component type as *x* and component count as *shuffle mask*.

shuffle mask must be *vector(2,4,8,16)* of *integer* values.

Result Type, *x* and *y* must be *vector*(2,4,8,16) of *floating-point* or *integer* values.

| 8 | 12 | <id></id> | Result <id></id> | extended | 183 | <id></id> | <id></id> | <id></id> |
|---|----|-------------|------------------|-----------------------------------|-----|-----------|-----------|--------------|
| | | Result Type | | instructions set < <i>id</i> > | | x | У | shuffle mask |

2.8. Misc instructions

This section describes additional miscellaneous instructions.

printf

The *printf* extended instruction writes output to an implementation-defined stream such as stdout under control of the string pointed to by format that specifies how subsequent arguments are converted for output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored. The printf instruction returns when the end of the format string is encountered

printf returns 0 if it was executed successfully and -1 otherwise

Result Type must be i32.

format must be a pointer(constant) to i8.

| 6 + | 12 | <i><id></id></i> | Result <id></id> | extended | 184 | < <i>i</i> d> | <id>, <id>,</id></id> |
|------|----|------------------|------------------|---------------|-----|---------------|-----------------------|
| vari | | Result Type | | instructions | | format | additional |
| able | | | | set <id></id> | | | arguments |

prefetch

Prefetch *num_elements* * size in bytes of the type pointed by *p*, into the global cache. The prefetch instruction is applied to an invocation in a workgroup and does not affect the functionality of the kernel.

num_elements must be size_t.

Result Type must be void.

ptr must be a pointer(global) to floating-point, integer or vector(2,3,4,8,16) of floating-point, integer values.

| 7 | 12 | <id></id> | Result <id></id> | extended | 185 | <id></id> | <id></id> |
|---|----|-------------|------------------|--------------|-----|-----------|-------------|
| | | Result Type | | instructions | | ptr | num_element |
| | | | | set | | | S |

Chapter 3. Appendix A: Changes and TBD

• Fork the revision stream, changes section, TBD, etc. from the core specification, so this specification has its own, starting numbering at revision 1. This document now lives independently.

3.1. Changes from Version 0.99, Revision 1

- Move to use the updated image/texturing/sampling, instead of extended instructions. Also, see changes in core specification related to this.
 - 14241 Implement OpenCL Extended Instructions for images/samplers with core OpImageSample instructions
- · Fixed internal bugs
 - 13455 Merged the OpenCL 1.2, 2.0, and 2.1 extended-instruction set into a single OpenCL extended-instruction set.
- Fixed public bugs

3.2. Changes from Version 0.99, Revision 2

- 14679 moved precision information to the OpenCL environment spec
- 14636 clarified trig functions to accept and return radians

3.3. Changes from Version 0.99, Revision 3

- Fixed internal bugs:
 - 14862 removed remaining image instructions as core versions are sufficient
 - 14636 Fixed type-o's in several trig functions accepting radian inputs and/or producing radian results
 - Flattened opcode numbers

3.4. Changes from Version 1.0, Revision 1

- Fixed internal bugs:
 - Issue 8 order of parameters for prefetch was reversed; pointer operand should be first.
 - Issue 103 typo: singp should be signp
- Fixed public bugs
 - 1469 incorrect specification of pow and pown

3.5. Changes from Version 1.0, Revision 2

- · Fixed internal bugs:
 - Issue 261 clarified that s_mad24 and u_mad24 only support 32-bit integers
 - Issue 262 added scalars to the types supported by length
 - Issue 266 fixed shuffle and shuffle2 description
 - Issue 267 fixed description of Idexp operands

3.6. Changes from Version 1.0, Revision 3

- Moved image and sampler encoding to the OpenCL environment specification
- · Editorial fixes and improvements
- Fixed internal bugs:
 - Issue 271 storage class inconsistency between vloadn/vstoren and vload_half/vstore_half
 - Issue 312 bad wording for vstorea_halfn

3.7. Changes from Version 1.0, Revision 4

Support SPV_KHR_no_integer_wrap_decoration, in the **s_abs** instruction.

3.8. Changes from Version 1.0, Revision 5

- Fixed internal bugs:
 - Issue 497 fixed description for s_upsample

3.9. Changes from Version 1.0, Revision 6

- Fixed internal bugs:
 - Issue 515 permit use of FPFastMathMode decorations with math, common, geometric, and vector data load/store instructions for environments that allow it.

3.10. Changes from Version 1.0, Revision 7

- Fixed internal bugs:
 - Corrected the description of u_upsample and s_upsample.