



# OpenCL Extended Instruction Set Specification

Boaz Ouriel, Intel

Version 1.00, Revision 2

May 15, 2017



Copyright © 2014-2017 The Khronos Group Inc. All Rights Reserved.

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

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

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

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Binary Form</b>	<b>4</b>
2.1	Math extended instructions . . . . .	6
2.2	Integer instructions . . . . .	34
2.3	Common instructions . . . . .	45
2.4	Geometric instructions . . . . .	48
2.5	Relational instructions . . . . .	51
2.6	Vector Data Load and Store instructions . . . . .	52
2.7	Miscellaneous Vector instructions . . . . .	57
2.8	Misc instructions . . . . .	59
2.9	Image encoding . . . . .	60
2.10	Sampler encoding . . . . .	63
<b>A</b>	<b>Changes and TBD</b>	<b>63</b>
A.1	Changes from Version 0.99, Revision 1 . . . . .	63
A.2	Changes from Version 0.99, Revision 2 . . . . .	63
A.3	Changes from Version 0.99, Revision 3 . . . . .	63
A.4	Changes from Version 1.0, Revision 1 . . . . .	63
A.5	Changes from Version 1.0, Revision 2 . . . . .	64

## Contributors and Acknowledgments

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

## 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 when **Memory Model** is set to **OpenCL**

## 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* when the **Addressing Model** is **Physical32** and *i64* when the **Addressing Model** is **Physical64**.

- *vector*( $n$ ) denotes an **OpTypeVector** where  $n$  indicates the component count.
  - *vector*( $n_1, n_2, \dots, n_i$ ) 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*( $s_1, s_2, \dots, s_i$ ) abbreviates *pointer*( $s_1$ ), *pointer*( $s_2$ ), ... 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.

<b>acos</b>						
Compute the arc cosine of $x$ .						
Result is an angle in radians						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<i>&lt;id&gt;</i> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <i>&lt;id&gt;</i>	0	<i>&lt;id&gt;</i> $x$

<b>acosh</b>						
Compute the inverse hyperbolic cosine of $x$ .						
Result is an angle in radians						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<i>&lt;id&gt;</i> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <i>&lt;id&gt;</i>	1	<i>&lt;id&gt;</i> $x$

<b>acospi</b>						
Compute $\text{acos}(x) / \pi$ .						
Result is an angle in radians						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	2	<id> $x$

<b>asin</b>						
Compute the arc sine of $x$ .						
Result is an angle in radians						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	3	<id> $x$

<b>asinh</b>						
Compute the inverse hyperbolic sine of $x$ .						
Result is an angle in radians						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	4	<id> $x$

<p><b>asinpi</b></p> <p>Compute <math>\text{asin}(x) / \pi</math>.</p> <p>Result is an angle in radians</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	5	<id> $x$

<p><b>atan</b></p> <p>Compute the arc tangent of <math>x</math>.</p> <p>Result is an angle in radians</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	6	<id> $x$

<p><b>atan2</b></p> <p>Compute the arc tangent of <math>y / x</math>.</p> <p>Result is an angle in radians</p> <p><i>Result Type</i>, <math>y</math> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	7	<id> $y$	<id> $x$



**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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	8	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	---	-------------

**atanpi**

Compute  $\text{atan}(x) / \pi$ .

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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	9	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	---	-------------

**atan2pi**

Compute  $\text{atan2}(y, x) / \pi$ .

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.

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

7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	10	<id> $y$	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------	-------------

<b>cbrt</b>						
Compute the cube-root of $x$ .						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	11	<id> $x$

<b>ceil</b>						
Round $x$ to integral value using the round to positive infinity rounding mode.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	12	<id> $x$

<b>copysign</b>							
Returns $x$ with its sign changed to match the sign of $y$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	13	<id> $x$	<id> $y$

<b>cos</b>						
Compute the cosine of $x$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	14	<id> $x$

<b>cosh</b>						
Compute the hyperbolic cosine of $x$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	15	<id> $x$

<b>cospi</b>						
Compute $\cos(x) / \pi$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	16	<id> $x$

<b>erfc</b>						
Complementary error function of $x$ .						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	17	<id> $x$

<b>erf</b>						
Error function of $x$ encountered in integrating the normal distribution.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	18	<id> $x$

<b>exp</b>						
Compute the base-e exponential of $x$ . (i.e. $e^x$ )						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	19	<id> $x$

<b>exp2</b>						
Computes 2 raised to the power of $x$ . (i.e. $2^x$ )						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	20	<id> $x$

<b>exp10</b>						
Computes 10 raised to the power of $x$ . (i.e. $10^x$ )						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	21	<id> $x$

<b>expm1</b>						
Computes $e^x - 1.0$ .						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	22	<id> $x$

<b>fabs</b>							
Compute the absolute value of $x$ .							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	23	<id> $x$	

<b>fdim</b>							
Compute $x - y$ if $x > y$ , $+0$ if $x$ is less than or equal to $y$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	24	<id> $x$	<id> $y$

<b>floor</b>							
Round $x$ to the integral value using the round to negative infinity rounding mode.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	25	<id> $x$	

<b>fma</b>								
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 behavior is per the IEEE 754-2008 standard.								
<i>Result Type</i> , $a$ , $b$ and $c$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	26	<id> $a$	<id> $b$	<id> $c$

**fmax**

Returns  $y$  if  $x < y$ , otherwise it returns  $x$ . If one argument 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* behave 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	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	27	<id> $x$	<id> $y$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	-------------

**fmin**

Returns  $y$  if  $y < x$ , otherwise it returns  $x$ . If one argument 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* behave 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	28	<id> $x$	<id> $y$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	-------------

**fmod**

Modulus. Returns  $x - y * \text{trunc}(x/y)$ .

*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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	29	<id> $x$	<id> $y$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	-------------

**fract**

Returns  $fmin(x - floor(x), 0x1.ffffep-1f \cdot floor(x))$  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 *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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	30	<id> <i>x</i>	<id> <i>ptr</i>
---	----	----------------------------	--------------------------	--------------------------------------	----	------------------	--------------------

**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 *i32* with the same component count as *Result Type* and *x* operands.

7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	31	<id> <i>x</i>	<id> <i>exp</i>
---	----	----------------------------	--------------------------	--------------------------------------	----	------------------	--------------------

**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.

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

7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	32	<id> <i>x</i>	<id> <i>y</i>
---	----	----------------------------	--------------------------	--------------------------------------	----	------------------	------------------

<b>ilogb</b>							
Return the exponent of $x$ as an $i32$ value.							
<i>Result Type</i> must be $i32$ or $vector(2,3,4,8,16)$ of $i32$ values.							
$x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
<i>Result Type</i> and $x$ operands must have the same component count.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	33	<id> $x$	

<b>ldexp</b>							
Multiply $x$ by 2 to the power $k$ .							
$k$ must be $i32$ or $vector(2,3,4,8,16)$ of $i32$ values.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
<i>Result Type</i> and $x$ operands must be of the same type. $exp$ operand must have the same component count as <i>Result Type</i> and $x$ operands.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	34	<id> $x$	<id> $k$

<b>lgamma</b>							
Log gamma function of $x$ . Returns the natural logarithm of the absolute value of the gamma function.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	35	<id> $x$	



**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 *i32* with the same component count as *Result Type* and  $x$  operands.

7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	36	<id> $x$	<id> <i>signp</i>
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	----------------------

**log**

Compute 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	37	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------

**log2**

Compute a base 2 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	38	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------

**log10**

Compute a 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	39	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------

<p><b>log1p</b></p> <p>Compute <math>\log_e(1.0 + x)</math>.</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	40	<id> $x$

<p><b>logb</b></p> <p>Compute the exponent of <math>x</math>, which is the integral part of <math>\log_r  x </math>.</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	41	<id> $x$

<p><b>mad</b></p> <p>mad approximates <math>a * b + c</math>. Whether or how the product of <math>a * b</math> is rounded and how supernormal or subnormal intermediate products are handled is not defined. mad is intended to be used where speed is preferred over accuracy</p> <p><i>Result Type</i>, <math>a</math>, <math>b</math> and <math>c</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p> <p><b>Note:</b> For some usages, e.g. <math>\text{mad}(a, b, -a*b)</math>, the definition of mad() is loose enough that almost any result is allowed from mad() for some values of <math>a</math> and <math>b</math>.</p>								
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	42	<id> $a$	<id> $b$	<id> $c$

<p><b>maxmag</b></p> <p>Returns <math>x</math> if <math> x  &gt;  y </math>, <math>y</math> if <math> y  &gt;  x </math>, otherwise <math>\text{fmax}(x, y)</math>.</p> <p><i>Result Type</i>, <math>x</math> and <math>y</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	43	<id> $x$	<id> $y$

<b>minmag</b>							
Returns $x$ if $ x  <  y $ , $y$ if $ y  <  x $ , otherwise $fmin(x, y)$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	44	<id> $x$	<id> $y$

<b>modf</b>							
Decompose a <i>floating-point</i> number. The <code>modf</code> function breaks the argument $x$ into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by <i>iptr</i>							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
<i>iptr</i> must be a <i>pointer(global, local, private, generic)</i> to <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type, or must be a pointer to the same type.							
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	45	<id> $x$	<id> <i>iptr</i>

<b>nan</b>							
Returns a quiet NaN. The <i>nancode</i> may be placed in the significand of the resulting NaN.							
<i>Result Type</i> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
<i>nancode</i> must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
<i>Result Type</i> and <i>nancode</i> operands must have the same component count. The primitive data type size of <i>nancode</i> and <i>Result Type</i> must be equal.							
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	46	<id> <i>nancode</i>	

<b>nextafter</b>							
Computes the next representable <i>floating-point</i> value following $x$ in the direction of $y$ . Thus, if $y$ is less than $x$ , <i>nextafter()</i> returns the largest representable floating-point number less than $x$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	47	<id> $x$	<id> $y$

<b>pow</b>							
Compute $x$ to the power $y$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	48	<id> $x$	<id> $y$

<b>pown</b>							
Compute $x$ to the power $y$ , where $y$ is an <i>i32</i> integer.							
$y$ must be <i>i32</i> or <i>vector(2,3,4,8,16)</i> of <i>i32</i> values.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.							
<i>Result Type</i> and $x$ operands must be of the same type. $y$ operand must have the same component count as <i>Result Type</i> and $x$ operands.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	49	<id> $x$	<id> $y$

<b>powr</b>							
Compute $x$ to the power $y$ , where $x$ is $\geq 0$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	50	<id> $x$	<id> $y$

**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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	51	<id> $x$	<id> $y$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	-------------

**remquo**

The `remquo` function 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` function. `remquo` also calculates 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 *i32* with the same component count as *Result Type*,  $x$  and  $y$  operands.

8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	52	<id> $x$	<id> $y$	<id> <code>quo</code>
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------	-------------	--------------------------

**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.

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

6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	53	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	----	-------------

<b>rootn</b>							
Compute $x$ to the power $1/y$ .							
$y$ must be $i32$ or $vector(2,3,4,8,16)$ of $i32$ values.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
<i>Result Type</i> and $x$ operands must be of the same type. $y$ operand must have the same component count as <i>Result Type</i> and $x$ operands.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	54	<id> $x$	<id> $y$

<b>round</b>							
Return the integral value nearest to $x$ rounding halfway cases away from zero, regardless of the current rounding direction.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	55	<id> $x$	

<b>rsqrt</b>							
Compute inverse square root of $x$ .							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	56	<id> $x$	

<b>sin</b>							
Compute sine of $x$ radians.							
<i>Result Type</i> and $x$ must be <i>floating-point</i> or $vector(2,3,4,8,16)$ of <i>floating-point</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	57	<id> $x$	

<b>sincos</b>							
<p>Compute sine and cosine of <math>x</math> radians. The computed sine is the return value and computed cosine is returned in <i>cosval</i>.</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p><i>cosval</i> must be a <i>pointer(global, local, private, generic)</i> to <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type, or must be a pointer to the same type.</p>							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	58	<id> $x$	<id> <i>cosval</i>

<b>sinh</b>							
<p>Compute hyperbolic sine of <math>x</math> radians.</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	59	<id> $x$	

<b>sinpi</b>							
<p>Compute <math>\sin(\pi x)</math> radians.</p> <p><i>Result Type</i> and <math>x</math> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p>							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	60	<id> $x$	

<b>sqrt</b>						
Compute square root of $x$ .						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	61	<id> $x$

<b>tan</b>						
Compute tangent of $x$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	62	<id> $x$

<b>tanh</b>						
Compute hyperbolic tangent of $x$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	63	<id> $x$

<b>tanpi</b>						
Compute $\tan(\pi x)$ radians.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	64	<id> $x$



<b>tgamma</b>						
Compute the gamma function of $x$ .						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	65	<id> $x$

<b>trunc</b>						
Round $x$ to integral value using the round to zero rounding mode.						
<i>Result Type</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	66	<id> $x$

<b>half_cos</b>						
Compute cosine of $x$ radians, where $x$ must be in the range $-2^{16} \dots +2^{16}$ .ha						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	67	<id> $x$

<b>half_divide</b>							
Compute $x / y$ .							
<i>Result Type</i> , $x$ and $y$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	68	<id> $x$	<id> $y$

<b>half_exp</b>						
Compute the base-e exponential of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	69	<id> $x$

<b>half_exp2</b>						
Compute the base- 2 exponential of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	70	<id> $x$

<b>half_exp10</b>						
Compute the base- 10 exponential of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	71	<id> $x$

<b>half_log</b>						
Compute natural logarithm of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	72	<id> $x$

**half\_log2**

Compute a 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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	73	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**half\_log10**

Compute a 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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	74	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**half\_powr**

Compute  $x$  to the power  $y$ , where  $x$  is  $\geq 0$ .

*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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	75	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------	-------------

**half\_recip**

Compute reciprocal 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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	76	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

<b>half_rsqrt</b>						
Compute inverse square root of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	77	<id> $x$

<b>half_sin</b>						
Compute sine of $x$ radians, where $x$ must be in the range $-2^{16} \dots +2^{16}$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	78	<id> $x$

<b>half_sqrt</b>						
Compute the square root of $x$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	79	<id> $x$

<b>half_tan</b>						
Compute tangent value of $x$ radians, where $x$ must be in the range $-2^{16} \dots +2^{16}$ .						
<i>Result Type</i> and $x$ must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	80	<id> $x$

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	81	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	82	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------	-------------

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	83	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

<b>native_exp2</b>						
<p>Compute the base- 2 exponential of <math>x</math> over an implementation-defined range. The maximum error is implementation-defined..</p> <p><i>Result Type</i> and <math>x</math> must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p> <p>The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	84	<id> $x$

<b>native_exp10</b>						
<p>Compute the base- 10 exponential of <math>x</math> over an implementation-defined range. The maximum error is implementation-defined..</p> <p><i>Result Type</i> and <math>x</math> must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p> <p>The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	85	<id> $x$

<b>native_log</b>						
<p>Compute natural logarithm of <math>x</math> over an implementation-defined range. The maximum error is implementation-defined.</p> <p><i>Result Type</i> and <math>x</math> must be <i>float</i> or <i>vector(2,3,4,8,16)</i> of <i>float</i> values.</p> <p>All of the operands, including the <i>Result Type</i> operand, must be of the same type.</p> <p>The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function</p>						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	86	<id> $x$

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	87	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	88	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**native\_powr**

Compute  $x$  to the power  $y$ , where  $x$  is  $\geq 0$ .

*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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	89	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------	-------------

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	90	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

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

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	92	<id> $x$
---	----	----------------------------	--------------------	--------------------------------------	----	-------------



**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

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

**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.

The function may map to one or more native device instructions and will typically have better performance compared to the non native corresponding functions. Support for denormal values is implementation-defined for this function

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

## 2.2 Integer instructions

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

<b>s_abs</b>						
Returns $ x $ , where $x$ is treated as signed integer.						
<i>Result Type</i> and $x$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	141	<id> $x$

<b>s_abs_diff</b>							
Returns $ x - y $ without modulo overflow, where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	142	<id> $x$	<id> $y$

<b>s_add_sat</b>							
Returns the saturated value of $x + y$ , where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	143	<id> $x$	<id> $y$

<b>u_add_sat</b>							
Returns the saturated value of $x + y$ , where $x$ and $y$ are treated as unsigned integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	144	<id> $x$	<id> $y$

<b>s_hadd</b>							
Returns the value of $(x + y) \gg 1$ , where $x$ and $y$ are treated as signed integers. The intermediate sum does not modulo overflow.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	145	<id> $x$	<id> $y$

<b>u_hadd</b>							
Returns the value of $(x + y) \gg 1$ , where $x$ and $y$ are treated as unsigned integers. The intermediate sum does not modulo overflow.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	146	<id> $x$	<id> $y$

<b>s_rhadd</b>							
Returns the value of $(x + y + 1) \gg 1$ , where $x$ and $y$ are treated as signed integers. The intermediate sum does not modulo overflow.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	147	<id> $x$	<id> $y$

**u\_rhadd**

Returns the value of  $(x + y + 1) \gg 1$ , 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	148	<id> $x$	<id> $y$
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------	-------------

**s\_clamp**

Returns  $s\_min(s\_max(x, minval), maxval)$ , where  $x$ ,  $minval$ , and  $maxval$  are treated as signed integers. Results 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	149	<id> $x$	<id> $minval$	<id> $maxval$
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------	------------------	------------------

**u\_clamp**

Returns  $u\_min(u\_max(x, minval), maxval)$ , where  $x$ ,  $minval$ , and  $maxval$  are treated as unsigned integers. Results 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	150	<id> $x$	<id> $minval$	<id> $maxval$
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------	------------------	------------------

**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	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	151	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------

<b>ctz</b>							
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.							
<i>Result Type</i> and $x$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	152	<id> $x$	

<b>s_mad_hi</b>								
Returns $mul\_hi(a, b) + c$ , where $a, b$ and $c$ are treated as signed integers.								
<i>Result Type</i> , $a$ , $b$ and $c$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	153	<id> $a$	<id> $b$	<id> $c$

<b>u_mad_sat</b>								
Returns $x * y + z$ and saturates the result where $x$ , $y$ and $z$ are treated as unsigned integers.								
<i>Result Type</i> , $x$ , $y$ and $z$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	154	<id> $x$	<id> $y$	<id> $z$

<b>s_mad_sat</b>								
Returns $x * y + z$ and saturates the result where $x$ , $y$ and $z$ are treated as signed integers.								
<i>Result Type</i> , $x$ , $y$ and $z$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	155	<id> $x$	<id> $y$	<id> $z$

<b>s_max</b>							
Returns $y$ if $x < y$ , otherwise it returns $x$ , where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	156	<id> $x$	<id> $y$

<b>u_max</b>							
Returns $y$ if $x < y$ , otherwise it returns $x$ , where $x$ and $y$ are treated as unsigned integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	157	<id> $x$	<id> $y$

<b>s_min</b>							
Returns $y$ if $y < x$ , otherwise it returns $x$ , where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	158	<id> $x$	<id> $y$

<b>u_min</b>							
Returns $y$ if $y < x$ , otherwise it returns $x$ , where $x$ and $y$ are treated as unsigned integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	159	<id> $x$	<id> $y$

<b>s_mul_hi</b>							
Computes $x * y$ and returns the high half of the product of $x$ and $y$ , where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	160	<id> $x$	<id> $y$

<b>rotate</b>							
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.							
<i>Result Type</i> , $v$ and $i$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	161	<id> $v$	<id> $i$

<b>s_sub_sat</b>							
Returns the saturated value of $x - y$ , where $x$ and $y$ are treated as signed integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	162	<id> $x$	<id> $y$

<b>u_sub_sat</b>							
Returns the saturated value of $x - y$ , where $x$ and $y$ are treated as unsigned integers.							
<i>Result Type</i> , $x$ and $y$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	163	<id> $x$	<id> $y$

**u\_upsample**

When *hi* and *lo* component type is i8:

$$\text{Result} = ((\text{upcast} \dots \text{ to i16})hi \ll 8) | lo$$

When *hi* and *lo* component type is i16:

$$\text{Result} = ((\text{upcast} \dots \text{ to i32})hi \ll 8) | lo$$

When *hi* and *lo* component i32:

$$\text{Result} = ((\text{upcast} \dots \text{ to i64})hi \ll 8) | lo$$

*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. When *hi* and *lo* component type is i8, the *Result Type* component type must be i16. When *hi* and *lo* component type is i16, the *Result Type* component type must be i32. When *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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	164	<id> <i>hi</i>	<id> <i>lo</i>
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------------	-------------------



<b>s_upsample</b>							
When <i>hi</i> and <i>lo</i> component type is i8:							
Result = ((upcast... to i16) <i>hi</i> << 8)   <i>lo</i>							
When <i>hi</i> and <i>lo</i> component type is i16:							
Result = ((upcast... to i32) <i>hi</i> << 8)   <i>lo</i>							
When <i>hi</i> and <i>lo</i> component i32:							
Result = ((upcast... to i64) <i>hi</i> << 8)   <i>lo</i>							
<i>hi</i> and <i>lo</i> are treated as signed integers.							
<i>hi</i> and <i>lo</i> must be <i>i8</i> , <i>i16</i> or <i>i32</i> or <i>vector(2,3,4,8,16)</i> of <i>i8</i> , <i>i16</i> or <i>i32</i> values.							
<i>Result Type</i> must be <i>i16</i> , <i>i32</i> or <i>i64</i> or <i>vector(2,3,4,8,16)</i> of <i>i16</i> , <i>i32</i> or <i>i64</i> values.							
<i>hi</i> and <i>lo</i> operands must be of the same type. When <i>hi</i> and <i>lo</i> component type is i8, the <i>Result Type</i> component type must be i16. When <i>hi</i> and <i>lo</i> component type is i16, the <i>Result Type</i> component type must be i32. When <i>hi</i> and <i>lo</i> component type is i32, the <i>Result Type</i> component type must be i64. <i>Result Type</i> must have the same component count as <i>hi</i> and <i>lo</i> operands.							
7	12	< <i>id</i> > <i>Result Type</i>	<i>Result</i> < <i>id</i> >	extended instructions set < <i>id</i> >	165	< <i>id</i> > <i>hi</i>	< <i>id</i> > <i>lo</i>

<b>popcount</b>							
Returns the number of non-zero bits in <i>x</i> .							
<i>Result Type</i> and <i>x</i> must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
6	12	< <i>id</i> > <i>Result Type</i>	<i>Result</i> < <i>id</i> >	extended instructions set < <i>id</i> >	166	< <i>id</i> > <i>x</i>	

<b>s_mad24</b>								
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 <code>s_mul24</code> to see how the 24-bit integer multiplication is performed.								
<i>Result Type</i> , $x$ , $y$ and $z$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	167	<id> $x$	<id> $y$	<id> $z$

<b>u_mad24</b>								
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 <code>u_mul24</code> to see how the 24-bit integer multiplication is performed.								
<i>Result Type</i> , $x$ , $y$ and $z$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	168	<id> $x$	<id> $y$	<id> $z$

<b>s_mul24</b>								
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. <code>s_mul24</code> should only be used when 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.								
<i>Result Type</i> , $x$ and $y$ must be <i>i32</i> or <i>vector(2,3,4,8,16)</i> of <i>i32</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
7	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	169	<id> $x$	<id> $y$	

**u\_mul24**

Multiply two 24-bit integer values  $x$  and  $y$ , where  $x$  and  $y$  are treated as unsigned integers.  $x$  and  $y$  are 32-bit integers but only the low-order 24 bits are used to perform the multiplication. `u_mul24` should only be used when 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	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	170	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------	-------------

**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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	201	<id> $x$	
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------	--

**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.

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

7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	202	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------	-------------

**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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	203	<id> $x$	<id> $y$
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------	-------------

<b>u_mad_hi</b>								
Returns $mul\_hi(a, b) + c$ , where $a, b$ and $c$ are treated as unsigned integers.								
<i>Result Type</i> , $a$ , $b$ and $c$ must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	204	<id> $a$	<id> $b$	<id> $c$

## 2.3 Common instructions

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

<b>fclamp</b>								
Returns $fmin(fmax(x, minval), maxval)$ . Results are undefined if $minval > maxval$ .								
<i>Result Type</i> , $x$ , $minval$ and $maxval$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	95	<id> $x$	<id> $minval$	<id> $maxval$

<b>degrees</b>								
Converts <i>radians</i> to degrees, i.e. $(180 / \pi) * radians$ .								
<i>Result Type</i> and <i>radians</i> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	96	<id> $radians$		

<b>fmax_common</b>								
Returns $y$ if $x < y$ , otherwise it returns $x$ . If $x$ or $y$ are infinite or NaN, the return values are undefined.								
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	97	<id> $x$	<id> $y$	

<b>fmin_common</b>							
Returns $y$ if $y < x$ , otherwise it returns $x$ . If $x$ or $y$ are infinite or NaN, the return values are undefined.							
<i>Result Type</i> , $x$ and $y$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	98	<id> $x$	<id> $y$

<b>mix</b>								
Returns the linear blend of $x$ & $y$ implemented as:								
$x + (y - x) * a$								
<i>Result Type</i> , $x$ , $y$ and $a$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
<b>Note:</b> This function can be implemented using contractions such as mad or fma								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	99	<id> $x$	<id> $y$	<id> $a$

<b>radians</b>						
Converts <i>degrees</i> to radians, i.e. $(\pi / 180) * \text{degrees}$ .						
<i>Result Type</i> and <i>degrees</i> must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	100	<id> <i>degrees</i>

<b>step</b>							
Returns 0.0 if $x < \text{edge}$ , otherwise it returns 1.0.							
<i>Result Type</i> , <i>edge</i> and $x$ must be <i>floating-point</i> or <i>vector(2,3,4,8,16)</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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	101	<id> <i>edge</i>	<id> $x$

**smoothstep**

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

This is equivalent to :

```
t = fclamp((x - edge0) / (edge1 - edge0), 0, 1);
```

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

Results are undefined if  $edge_0 \geq edge_1$  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 function can be implemented using contractions such as mad or fma

8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	102	<id> $edge_0$	<id> $edge_1$	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	-----	------------------	------------------	-------------

**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.

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

6	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	103	<id> $x$
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------

## 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 functions can be implemented using contractions such as mad or fma

<b>cross</b>							
Returns the cross product of $p_0.xyz$ and $p_1.xyz$ .							
When the vector component count is 4, the $w$ component returned will be 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> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	104	<id> $p_0$	<id> $p_1$

<b>distance</b>							
Returns the distance between $p_0$ and $p_1$ . This is calculated as $length(p_0 - p_1)$ .							
<i>Result Type</i> must be <i>floating-point</i> .							
$p_0$ and $p_1$ must be <i>floating-point</i> or <i>vector(2,3,4)</i> of <i>floating-point</i> values.							
$p_0$ and $p_1$ operands must have the same type. <i>Result Type</i> , $p_0$ and $p_1$ operands must have the same component type							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	105	<id> $p_0$	<id> $p_1$

<b>length</b>							
Return the length of vector $p$ , i.e. $sqrt(p.x^2 + p.y^2 + \dots)$							
<i>Result Type</i> must be <i>floating-point</i> .							
$p$ must be <i>vector(2,3,4)</i> of <i>floating-point</i> values.							
<i>Result Type</i> and $p$ operands must have the same component type							
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	106	<id> $p$	



<b>normalize</b>						
Returns a vector in the same direction as $p$ but with a length of 1.						
<i>Result Type</i> and $p$ must be <i>floating-point</i> or <i>vector(2,3,4)</i> of <i>floating-point</i> values.						
All of the operands, including the <i>Result Type</i> operand, must be of the same type.						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	107	<id> $p$

<b>fast_distance</b>							
Returns $fast\_length(p_0 - p_1)$ .							
<i>Result Type</i> must be <i>floating-point</i> .							
$p_0$ and $p_1$ must be <i>floating-point</i> or <i>vector(2,3,4)</i> of <i>floating-point</i> values.							
$p_0$ and $p_1$ operands must have the same type. <i>Result Type</i> , $p_0$ and $p_1$ operands must have the same component type							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	108	<id> $p_0$	<id> $p_1$

<b>fast_length</b>						
Return the length of vector $p$ computed as: $half\_sqrt(p.x^2 + p.y^2 + \dots)$						
<i>Result Type</i> must be <i>floating-point</i> .						
$p$ must be <i>vector(2,3,4)</i> of <i>floating-point</i> values.						
<i>Result Type</i> and $p$ operands must have the same component type						
6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	109	<id> $p$

**fast\_normalize**

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

$$p * \text{half\_rsqrt}(p.x^2 + p.y^2 \dots)$$

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  $\text{sqrt}(\text{FLT\_MIN})$  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.

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

6	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	110	<id> $p$
---	----	----------------------------	--------------------	--------------------------------------	-----	-------------

## 2.5 Relational instructions

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

<b>bitselect</b>								
Each bit of the result is the corresponding bit of <i>a</i> if the corresponding bit of <i>c</i> is 0. Otherwise it is the corresponding bit of <i>b</i> .								
<i>Result Type</i> , <i>a</i> , <i>b</i> and <i>c</i> must be <i>floating-point</i> or <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> or <i>integer</i> values.								
All of the operands, including the <i>Result Type</i> operand, must be of the same type.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	186	<id> <i>a</i>	<id> <i>b</i>	<id> <i>c</i>

<b>select</b>								
For each component of a vector type, the result is <i>a</i> if the most significant bit of <i>c</i> is zero, otherwise it is <i>b</i> .								
For a scalar type, the result is <i>a</i> if <i>c</i> is zero, otherwise it is <i>b</i> .								
<i>c</i> must be <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>integer</i> values.								
<i>Result Type</i> , <i>a</i> and <i>b</i> must be <i>floating-point</i> or <i>integer</i> or <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> or <i>integer</i> values.								
<i>Result Type</i> , <i>a</i> and <i>b</i> must have the same type. <i>c</i> operand must have the same component count and component bit width as the rest of the operands.								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	187	<id> <i>a</i>	<id> <i>b</i>	<id> <i>c</i>

## 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.

<b>vloadn</b>								
Return a vector value which is read from address $(p + (offset * n))$ .								
The address computed as $(p + (offset * n))$ must be 8-bit aligned if $p$ points to i8 value; 16-bit aligned if $p$ points to i16 or half value; 32-bit aligned if $p$ points to i32 or float value; 64-bit aligned if $p$ points to i64 or double value.								
$offset$ must be $size\_t$ .								
$p$ must be a <i>pointer(constant, generic)</i> to <i>floating-point, integer</i> .								
<i>Result Type</i> must be <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> or <i>integer</i> values.								
<i>Result Type</i> 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	171	<id> <i>offset</i>	<id> $p$	Literal Number $n$

<b>vstoren</b>								
Write <i>data</i> vector value to the address $(p + (offset * compCountOf(data)))$ , where $compCountOf(data)$ is equal to the component count of the vector <i>data</i> .								
The address computed as $(p + (offset * compCountOf(data)))$ must be 8-bit aligned if $p$ points to i8 value; 16-bit aligned if $p$ points to i16 or half value; 32-bit aligned if $p$ points to i32 or float value; 64-bit aligned if $p$ points to i64 or double value.								
$offset$ must be $size\_t$ .								
<i>Result Type</i> must be <i>void</i> .								
$p$ must be a <i>pointer(generic)</i> to <i>floating-point, integer</i> .								
<i>data</i> must be <i>vector(2,3,4,8,16)</i> of <i>floating-point</i> or <i>integer</i> values.								
<i>data</i> component type must be equal to the type pointed by $p$ .								
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	172	<id> <i>data</i>	<id> <i>offset</i>	<id> $p$

**vload\_half**

Reads a half value from the address ( $p + (offset)$ ) and converts it to a float return value. The address computed as ( $p + (offset)$ ) must be 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	173	<id> <i>offset</i>	<id> <i>p</i>
---	----	----------------------------	--------------------------	--------------------------------------	-----	-----------------------	------------------

**vload\_halfn**

Reads a half vector value from the address ( $p + (offset * n)$ ) and converts it to a float vector return value. The address computed as ( $p + (offset * n)$ ) must be 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	174	<id> <i>offset</i>	<id> <i>p</i>	Literal Number <i>n</i>
---	----	----------------------------	--------------------------	--------------------------------------	-----	-----------------------	------------------	-------------------------------

**vstore\_half**

Converts *data* float or double value to a half value and then write the converted value to the address ( $p + offset$ ). The address computed as ( $p + offset$ ) must be 16-bit aligned.

This function uses the default rounding mode when converting *data* to a half value. The default rounding mode is round to nearest even.

*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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	175	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>
---	----	----------------------------	--------------------------	--------------------------------------	-----	---------------------	-----------------------	------------------

<b>vstore_half_r</b>									
<p>Converts <i>data</i> float or double value to a half value and then write the converted value to the address (<math>p + offset</math>). The address computed as (<math>p + offset</math>) must be 16-bit aligned.</p> <p>This function uses <i>mode</i> rounding mode when converting <i>data</i> to a half value.</p> <p><i>data</i> must be <i>float</i> or <i>double</i>.</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>Result Type</i> must be <i>void</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, generic)</i> to <i>half</i>.</p>									
9	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	176	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>	FP Rounding Mode <i>mode</i>

<b>vstore_halfn</b>									
<p>Converts <i>data</i> vector of float or vector of double values to a vector of half values and then write the converted value to the address (<math>p + (offset * compCountOf(data))</math>), where <math>compCountOf(data)</math> is equal to the component count of the vector <i>data</i>.</p> <p>The address computed as (<math>p + (offset * compCountOf(data))</math>) must be 16-bit aligned.</p> <p>This function uses the default rounding mode when converting <i>data</i> to a vector of half values. The default rounding mode is round to nearest even.</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>Result Type</i> must be <i>void</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, generic)</i> to <i>half</i>.</p> <p><i>data</i> must be <i>vector(2,3,4,8,16)</i> of <i>float</i> or <i>double</i> values.</p>									
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	177	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>	

<b>vstore_halfn_r</b>									
<p>Converts <i>data</i> vector of float or vector of double values to a vector of half values and then write the converted value to the address (<math>p + (offset * compCountOf(data))</math>), where <math>compCountOf(data)</math> is equal to the component count of the vector <i>data</i>.</p> <p>The address computed as (<math>p + (offset * compCountOf(data))</math>) must be 16-bit aligned.</p> <p>This function uses <i>mode</i> rounding mode when converting <i>data</i> to a half value.</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>Result Type</i> must be <i>void</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, generic)</i> to <i>half</i>.</p> <p><i>data</i> must be <i>vector(2,3,4,8,16)</i> of <i>float</i> or <i>double</i> values.</p>									
9	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	178	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>	FP Rounding Mode <i>mode</i>

<b>vloada_halfn</b>									
<p>Reads a half vector value from the address (<math>p + (offset * n)</math>) and converts it to a float vector return value. The address computed as (<math>p + (offset * n)</math>) must be (<math>2 * n</math>) bytes aligned, when <math>n = 2,4,8,16</math>; For <math>n = 3</math>, the function returns a vector of 3 float values from the address (<math>p + (offset * 4)</math>). The address computed as (<math>p + (offset * 4)</math>) must be 8-bytes aligned</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, constant, generic)</i> to <i>half</i>.</p> <p><i>Result Type</i> must be <i>vector(2,3,4,8,16)</i> of <i>float</i> values.</p> <p><i>Result Type</i> component count must be equal to <i>n</i>.</p> <p><i>n</i> must be 2,3,4,8 or 16.</p>									
8	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	179	<id> <i>offset</i>	<id> <i>p</i>	Literal Number <i>n</i>	

<b>vstorea_halfn</b>									
<p>Converts <i>data</i> vector of float or vector of double values to a vector of half values and then write the converted value to the address <math>(p + (offset * compCountOf(data)))</math>, where <math>compCountOf(data)</math> is equal to the component count of the vector <i>data</i>.</p> <p>The address computed as <math>(p + (offset * compCountOf(data)))</math> must be <math>(2 * compCountOf(data))</math> bytes aligned, when <math>n = 2,4,8,16</math>; For <math>n = 3</math>, the function returns a vector of 3 float values from the address <math>(p + (offset * 4))</math>. The address computed as <math>(p + (offset * 4))</math> must be 8-bytes aligned.</p> <p>This function uses the default rounding mode when converting <i>data</i> to a vector of half values. The default rounding mode is round to nearest even.</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>Result Type</i> must be <i>void</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, generic)</i> to <i>half</i>.</p> <p><i>data</i> must be <i>vector(2,3,4,8,16)</i> of <i>float</i> or <i>double</i> values.</p>									
8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	180	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>	

<b>vstorea_halfn_r</b>									
<p>Converts <i>data</i> vector of float or vector of double values to a vector of half values and then write the converted value to the address <math>(p + (offset * compCountOf(data)))</math>, where <math>compCountOf(data)</math> is equal to the component count of the vector <i>data</i>.</p> <p>The address computed as <math>(p + (offset * compCountOf(data)))</math> must be <math>(2 * compCountOf(data))</math> bytes aligned, when <math>n = 2,4,8,16</math>; For <math>n = 3</math>, the function returns a vector of 3 float values from the address <math>(p + (offset * 4))</math>. The address computed as <math>(p + (offset * 4))</math> must be 8-bytes aligned.</p> <p>This function uses <i>mode</i> rounding mode when converting <i>data</i> to a vector of half values.</p> <p><i>offset</i> must be <i>size_t</i>.</p> <p><i>Result Type</i> must be <i>void</i>.</p> <p><i>p</i> must be a <i>pointer(global, local, private, generic)</i> to <i>half</i>.</p> <p><i>data</i> must be <i>vector(2,3,4,8,16)</i> of <i>float</i> or <i>double</i> values.</p>									
9	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	181	<id> <i>data</i>	<id> <i>offset</i>	<id> <i>p</i>	FP Rounding Mode <i>mode</i>



## 2.7 Miscellaneous Vector instructions

This section describes additional vector instructions.

<b>shuffle</b>							
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 <i>shuffle mask</i> .							
In this function, only the $\text{ilogb}(2m - 1)$ least significant bits of each mask element are considered, where $m$ is equal to the component count of $x$ .							
<i>shuffle mask</i> operand specifies, for each component in the result vector, which component of $x$ it gets.							
The size of each component in <i>shuffle mask</i> must match the size of each component in <i>Result Type</i> .							
<i>Result Type</i> must have the same component type as $x$ and component count as <i>shuffle mask</i> .							
<i>shuffle mask</i> must be <i>vector(2,4,8,16)</i> of <i>integer</i> values.							
<i>Result Type</i> and $x$ must be <i>vector(2,4,8,16)</i> of <i>floating-point</i> or <i>integer</i> values.							
All of the operands, including the <i>Result Type</i> operand, must be of the same type.							
7	12	<id> <i>Result Type</i>	<i>Result</i> <id>	extended instructions set <id>	182	<id> $x$	<id> <i>shuffle mask</i>

**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*.

In this function, only the  $\text{ilogb}(2 m - 1) + 1$  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.

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

8	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	183	<id> $x$	<id> $y$	<id> <i>shuffle mask</i>
---	----	----------------------------	--------------------------	--------------------------------------	-----	-------------	-------------	---------------------------------

## 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* function 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 + vari- able	12	<id> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	184	<id> <i>format</i>	<id>, <id>, ... <i>additional arguments</i>
----------------------	----	----------------------------	--------------------------	--------------------------------------	-----	-----------------------	---

### **prefetch**

Prefetch *num\_elements* \* *size* in bytes of the type pointed by *p*, into the global cache. The prefetch instruction is applied to a work-item in a work-group and does not affect the functional behavior 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> <i>Result Type</i>	<i>Result &lt;id&gt;</i>	extended instructions set <id>	185	<id> <i>ptr</i>	<id> <i>num_elements</i>
---	----	----------------------------	--------------------------	--------------------------------------	-----	--------------------	-----------------------------

## 2.9 Image encoding

The following list denotes the different valid **OpTypeImage** encodings of image objects.

<b>image1d</b>									
A 1D image									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 1D	<i>Depth</i> 0	<i>Arrayed</i> 0	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image1dBuffer</b>									
A 1D image created from a buffer object.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> Buffer	<i>Depth</i> 0	<i>Arrayed</i> 0	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image1dArray</b>									
A 1D image array.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 1D	<i>Depth</i> 0	<i>Arrayed</i> 1	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2d</b>									
A 2D image.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 0	<i>Arrayed</i> 0	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dArray</b>									
A 2D image array.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 0	<i>Arrayed</i> 1	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dDepth</b>									
A 2D depth image.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 1	<i>Arrayed</i> 0	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dArrayDepth</b>									
A 2D depth image array.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 1	<i>Arrayed</i> 1	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dMsaa</b>									
A 2D multi-sample color image.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 0	<i>Arrayed</i> 0	<i>MS</i> 1	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dArrayMsaa</b>									
A 2D multi-sample color image array.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 0	<i>Arrayed</i> 1	<i>MS</i> 1	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dMsaaDepth</b>									
A 2D multi-sample depth image.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 1	<i>Arrayed</i> 0	<i>MS</i> 1	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image2dArrayMsaaDepth</b>									
A 2D multi-sample depth image array.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 2D	<i>Depth</i> 1	<i>Arrayed</i> 1	<i>MS</i> 1	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

<b>image3d</b>									
A 3D image object.									
9	25	<i>Result</i> <id>	<i>Sampled</i> Type Op- TypeVoid	<i>Dim</i> 3D	<i>Depth</i> 0	<i>Arrayed</i> 0	<i>MS</i> 0	<i>Sampled</i> 0	<i>Image</i> <i>Format</i> Unknown

## 2.10 Sampler encoding

A SPIR-V *sampler* object is encoded via the **OpTypeSampler** instruction via a kernel function argument:

In addition, it is possible to define a constant (or inline) *sampler* using the **OpConstantSampler** instruction.

## 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.

### A.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

### A.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

### A.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

### A.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"

## A.5 Changes from Version 1.0, Revision 2

- Fixed internal bugs
  - Issue 121 - Incorrect descriptions of OpenCL instructions (powr, select, s\_clamp, u\_clamp)