



SPIR-V Specification

The Khronos® SPIR™ Working Group

Version 1.6, Revision 6: Unified

Table of Contents

1. Introduction	6
1.1. Goals	6
1.2. Execution Environment and Client API	7
1.3. About This Document	7
1.3.1. Versioning	7
1.4. Extendability	7
1.5. Debuggability	8
1.6. Design Principles	8
1.7. Static Single Assignment (SSA)	8
1.8. Built-In Variables	9
1.9. Specialization	9
1.10. Example	10
2. Specification	14
2.1. Language Capabilities	14
2.2. Terms	14
2.2.1. Instructions	14
2.2.2. Types	15
2.2.3. Computation	17
2.2.4. Module	17
2.2.5. Control Flow	18
2.2.6. Validity and Defined Behavior	21
2.3. Physical Layout of a SPIR-V Module and Instruction	22
2.4. Logical Layout of a Module	23
2.5. Instructions	24
2.5.1. SSA Form	25
2.6. Entry Point and Execution Model	25
2.7. Execution Modes	25
2.8. Types and Variables	26
2.8.1. Unsigned Versus Signed Integers	26
2.9. Function Calling	27
2.10. Extended Instruction Sets	27
2.11. Structured Control Flow	28
2.11.1. Rules for Structured Control-flow Declarations	28
2.11.2. Structured Control-flow Constructs	29
2.11.3. Rules for Structured Control-flow Constructs	29
2.12. Specialization	30
2.13. Linkage	32
2.14. Relaxed Precision	32
2.15. Debug Information	33
2.15.1. Function-Name Mangling	34
2.16. Validation Rules	34
2.16.1. Universal Validation Rules	34
2.16.2. Validation Rules for Shader Capabilities	39
2.16.3. Validation Rules for Kernel Capabilities	40

2.17. Universal Limits	41
2.18. Memory Model	41
2.18.1. Memory Layout	42
2.18.2. Aliasing	42
2.18.3. Null pointers	44
2.19. Derivatives	44
2.20. Code Motion	44
2.21. Deprecation	44
2.22. Unified Specification	44
2.23. Uniformity	45
3. Binary Form	46
3.1. Magic Number	46
3.2. Enumerants	46
3.2.1. Source Language	46
3.2.2. Execution Model	47
3.2.3. Addressing Model	48
3.2.4. Memory Model	48
3.2.5. Execution Mode	49
3.2.6. Storage Class	64
3.2.7. Dim	67
3.2.8. Sampler Addressing Mode	68
3.2.9. Sampler Filter Mode	68
3.2.10. Image Format	68
3.2.11. Image Channel Order	70
3.2.12. Image Channel Data Type	71
3.2.13. Image Operands	71
3.2.14. FP Fast Math Mode	75
3.2.15. FP Rounding Mode	76
3.2.16. Linkage Type	77
3.2.17. Access Qualifier	77
3.2.18. Function Parameter Attribute	77
3.2.19. Decoration	78
3.2.20. BuiltIn	99
3.2.21. Selection Control	113
3.2.22. Loop Control	114
3.2.23. Function Control	116
3.2.24. Memory Semantics <id>	116
3.2.25. Memory Operands	119
3.2.26. Scope <id>	121
3.2.27. Group Operation	124
3.2.28. Kernel Enqueue Flags	126
3.2.29. Kernel Profiling Info	127
3.2.30. Capability	127
3.2.31. Ray Flags	153
3.2.32. Ray Query Intersection	154
3.2.33. Ray Query Committed Type	154

3.2.34. Ray Query Candidate Type	155
3.2.35. Fragment Shading Rate	155
3.2.36. FP Denorm Mode	155
3.2.37. FP Operation Mode	156
3.2.38. Quantization Mode	156
3.2.39. Overflow Mode	157
3.2.40. Packed Vector Format	157
3.2.41. Cooperative Matrix Operands	157
3.2.42. Cooperative Matrix Layout	158
3.2.43. Cooperative Matrix Use	158
3.2.44. Cooperative Matrix Reduce Mode	158
3.2.45. Tensor Clamp Mode	158
3.2.46. Tensor Addressing Operands	159
3.2.47. Tensor Operands	159
3.2.48. Initialization Mode Qualifier	160
3.2.49. Host Access Qualifier	160
3.2.50. Load Cache Control	160
3.2.51. Store Cache Control	161
3.2.52. Named Maximum Number of Registers	161
3.2.53. Matrix Multiply Accumulate Operands	161
3.2.54. Raw Access Chain Operands	162
3.2.55. FP Encoding	162
3.2.56. Cooperative Vector Matrix Layout	162
3.2.57. Cooperative Vector Matrix Component Type	163
3.3. Instructions	164
3.3.1. Miscellaneous Instructions	164
3.3.2. Debug Instructions	166
3.3.3. Annotation Instructions	170
3.3.4. Extension Instructions	173
3.3.5. Mode-Setting Instructions	175
3.3.6. Type-Declaration Instructions	178
3.3.7. Constant-Creation Instructions	187
3.3.8. Memory Instructions	194
3.3.9. Function Instructions	205
3.3.10. Image Instructions	207
3.3.11. Conversion Instructions	231
3.3.12. Composite Instructions	239
3.3.13. Arithmetic Instructions	244
3.3.14. Bit Instructions	261
3.3.15. Relational and Logical Instructions	267
3.3.16. Derivative Instructions	280
3.3.17. Control-Flow Instructions	285
3.3.18. Atomic Instructions	291
3.3.19. Primitive Instructions	301
3.3.20. Barrier Instructions	302
3.3.21. Group and Subgroup Instructions	305

3.3.22. Device-Side Enqueue Instructions	321
3.3.23. Pipe Instructions	332
3.3.24. Non-Uniform Instructions	344
3.3.25. Tensor Instructions	374
3.3.26. Graph Instructions	375
3.3.27. Reserved Instructions	376
4. Appendix A: Changes	402
4.1. Changes from Version 0.99, Revision 31	402
4.2. Changes from Version 0.99, Revision 32	403
4.3. Changes from Version 1.00, Revision 1	403
4.4. Changes from Version 1.00, Revision 2	405
4.5. Changes from Version 1.00, Revision 3	406
4.6. Changes from Version 1.00, Revision 4	406
4.7. Changes from Version 1.00, Revision 5	406
4.8. Changes from Version 1.00, Revision 6	406
4.9. Changes from Version 1.00, Revision 7	407
4.10. Changes from Version 1.00, Revision 8	407
4.11. Changes from Version 1.00, Revision 9	407
4.12. Changes from Version 1.00, Revision 10	407
4.13. Changes from Version 1.00, Revision 11	408
4.14. Changes from Version 1.00	409
4.15. Changes from Version 1.1, Revision 1	409
4.16. Changes from Version 1.1, Revision 2	409
4.17. Changes from Version 1.1, Revision 3	409
4.18. Changes from Version 1.1, Revision 4	410
4.19. Changes from Version 1.1, Revision 5	410
4.20. Changes from Version 1.1, Revision 6	410
4.21. Changes from Version 1.1, Revision 7	410
4.22. Changes from Version 1.1	410
4.23. Changes from Version 1.2, Revision 1	410
4.24. Changes from Version 1.2, Revision 2	410
4.25. Changes from Version 1.2, Revision 3	410
4.26. Changes from Version 1.2	411
4.27. Changes from Version 1.3, Revision 1	411
4.28. Changes from Version 1.3, Revision 2	412
4.29. Changes from Version 1.3, Revision 3	413
4.30. Changes from Version 1.3, Revision 4	413
4.31. Changes from Version 1.3, Revision 5	413
4.32. Changes from Version 1.3, Revision 6	414
4.33. Changes from Version 1.3, Revision 7	415
4.34. Changes from Version 1.3	416
4.35. Changes from Version 1.4, Revision 1	416
4.36. Changes from Version 1.4	417
4.37. Changes from Version 1.5, Revision 1	417
4.38. Changes from Version 1.5, Revision 2	418
4.39. Changes from Version 1.5, Revision 3	419

4.40. Changes from Version 1.5, Revision 4	420
4.41. Changes from Version 1.5, Revision 5	420
4.42. Changes from Version 1.5.	422
4.43. Changes from Version 1.6, Revision 1	422
4.44. Changes from Version 1.6, Revision 2	423
4.45. Changes from Version 1.6, Revision 3	425
4.46. Changes from Version 1.6, Revision 4	426
4.47. Changes from Version 1.6, Revision 5	427



Copyright 2014-2025 The Khronos Group Inc.

This Specification is protected by copyright laws and contains material proprietary to Khronos. Except as described by these terms, it or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos.

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

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

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

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

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

Some parts of this Specification are purely informative and so are EXCLUDED from the Scope of this Specification. Section 1.3 "About This Document" defines how these parts of the Specification are identified.

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

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

Khronos® and Vulkan® are registered trademarks, and ANARI™, WebGL™, glTF™, NNEF™, OpenVX™, SPIR™, SPIR-V™, SYCL™, OpenVG™, Vulkan SC™, 3D Commerce™ and Kamaros™ are trademarks of The Khronos Group Inc. OpenXR™ is a trademark owned by The Khronos Group Inc. and is registered as a trademark in China, the European Union, Japan and the United Kingdom. OpenCL™ is a trademark of Apple Inc. used under license by Khronos. OpenGL® is a registered trademark and the OpenGL ES™ and OpenGL SC™ logos are trademarks of Hewlett Packard Enterprise used under license by Khronos. ASTC is a trademark of ARM Holdings PLC. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

Contributors and Acknowledgments

Editors

- John Kessenich, Google
- Boaz Ouriel, Intel
- Raun Krisch, Intel
- Victor Lomüller, Codeplay (current)

Contributors

Connor Abbott, Intel
Ben Ashbaugh, Intel
Alexey Bader, Intel
Alan Baker, Google
Dan Baker, Oxide Games
Kenneth Benzie, Codeplay
Jeff Bolz, NVIDIA
Stuart Brady, Arm
Gordon Brown, Codeplay
Pat Brown, NVIDIA
Nate Cesario, LunarG
Diana Po-Yu Chen, MediaTek
Stephen Clarke, Imagination
Joshua Davis, Unity
Hugo Devillers, University of Saarland
Patrick Doane, Blizzard Entertainment
Alastair Donaldson, Google
Yuehai Du, Qualcomm
Stefanus Du Toit, Google
Faith Ekstrand, Collabora
Gregory Fischer, LunarG
Theresa Foley, Intel
Spencer Fricke, Samsung
Ben Gaster, Qualcomm
Alexander Galazin, ARM
Christopher Gautier, ARM
Arcady Goldmints, LunarG
Jeremy Hayes, LunarG
Tobias Hector, AMD
Nicolai Hahnle, AMD

Neil Henning, AMD
Kerch Holt, NVIDIA
Lee Howes, Qualcomm
Samuel Huang, Mediatek
Marty Johnson, Khronos
Roy Ju, MediaTek
Baldur Karlsson, Valve
Ronan Keryell, Xilinx
John Kessenich, Google
Wooyoung Kim, Qualcomm
Vasileios Klimis, Imperial College London
Daniel Koch, NVIDIA
Ashwin Kolhe, NVIDIA
Tim Kong, Samsung
Raun Krisch, Intel
Graeme Leese, Broadcom
Yuan Lin, NVIDIA
Yaxun Liu, AMD
Victor Lomuller, Codeplay
Timothy Lottes, Epic Games
John McDonald, Valve
Mariusz Merecki, Intel
David Neto, Google
Boaz Ouriel, Intel
Kevin Petit, Arm
Robert Quill, Imagination Technologies
Christophe Riccio, Unity
Andrew Richards, Codeplay
Ian Romanick, Intel
Graham Sellers, AMD
Simon Waters, Samsung
Robert Simpson, Qualcomm
Pradyuman Singh, NVIDIA
Bartosz Sochacki, Intel
Nikos Stavropoulos, Think Silicon
Brian Sumner, AMD
John Wickerson, Imperial College London
Andrew Woloszyn, Google

Robin Voetter, StreamHPC

Ruihao Zhang, Qualcomm

Weifeng Zhang, Qualcomm

Chapter 1. Introduction

NOTE

Up-to-date HTML and PDF versions of this specification may be found at the [Khronos SPIR-V Registry](https://www.khronos.org/registry/spir-v/). (<https://www.khronos.org/registry/spir-v/>)

Abstract

SPIR-V is a simple binary intermediate language for graphical shaders and compute kernels. A SPIR-V module contains multiple entry points with potentially shared functions in the entry point's call trees. Each function contains a control-flow graph (CFG) of basic blocks, with optional instructions to express structured control flow. Load/store instructions are used to access declared variables, which includes all input/output (IO). Intermediate results bypassing load/store use static single-assignment (SSA) representation. Data objects are represented logically, with hierarchical type information: There is no flattening of aggregates or assignment to physical register banks, etc. Selectable addressing models establish whether general pointer operations may be used, or if memory access is purely logical.

This document fully defines **SPIR-V**, a Khronos-standard binary intermediate language for representing graphical-shader stages and compute kernels for multiple client APIs.

This is a [unified specification](#), specifying all versions since and including version 1.0.

1.1. Goals

SPIR-V has the following goals:

- Provide a simple binary intermediate language for all functionality appearing in Khronos shaders/kernels.
- Have a concise, transparent, self-contained specification (sections [Specification](#) and [Binary Form](#)).
- Map easily to other intermediate languages.
- Be the form passed by a client API into a driver to set shaders/kernels.
- Support multiple execution environments, specified by client APIs.
- Can be targeted by new front ends for novel high-level languages.
- Allow the first steps of compilation and reflection to be done offline.
- Be low-level enough to require a reverse-engineering step to reconstruct source code.
- Improve portability by enabling shared tools to generate or operate on it.
- Reduce compile time during application run time. (Eliminating most of the compile time during application run time is not a goal of this intermediate language. Target-specific register allocation and scheduling are still expected to take significant time.)
- Allow some optimizations to be done offline.

1.2. Execution Environment and Client API

SPIR-V is adaptable to multiple execution environments: A SPIR-V module is consumed by an execution environment, as specified by a client API. The full set of rules needed to consume SPIR-V in a particular environment comes from the combination of SPIR-V and that environment's client API specification. The client API specifies its SPIR-V execution environment as well as extra rules, limitations, capabilities, etc. required by the form of SPIR-V it can validly consume.

1.3. About This Document

This document aims to:

- Specify everything needed to create and consume non-extended SPIR-V, minus:
 - Extended instruction sets, which are imported and come with their own specifications.
 - Client API-specific rules, which are documented in client API specifications.
- Separate expository and specification language. The specification-proper is in [Specification](#) and [Binary Form](#).

1.3.1. Versioning

The specification covers multiple versions of SPIR-V, as described in the [unified section](#). It has followed a *Major.Minor.Revision* versioning scheme, with the specification's stated version being the most recent version of SPIR-V.

Major and *Minor* (but not *Revision*) are declared within a SPIR-V module.

Major is reserved for future use and has been fixed at 1. *Minor* changes have signified additions, deprecation, and removal of features. *Revision* changes have included clarifications, bug fixes, and [deprecation](#) (but not removal) of existing features.

1.4. Extendability

SPIR-V can be extended by multiple vendors or parties simultaneously:

- Using the [OpExtension](#) instruction to add semantics, which are described in an extension specification.
- Reserving (registering) ranges of the token values, as described further below.
- Aided by instruction skipping, also further described below.

Enumeration Token Values. It is easy to extend all the types, storage classes, opcodes, decorations, etc. by adding to the token values.

Registration. Ranges of token values in the [Binary Form](#) section can be pre-allocated to numerous vendors/parties. This allows combining multiple independent extensions without conflict. To register ranges, use the <https://github.com/KhronosGroup/SPIRV-Headers> repository, and submit pull requests against the `include/spirv/spir-v.xml` file.

Extended Instructions. Sets of extended instructions can be provided and specified in separate specifications. Multiple sets of extended instructions can be imported without conflict, as the extended instructions are selected by {set id, instruction number} pairs.

Instruction Skipping. Tools are encouraged to skip opcodes for features they are not required to process.

This is trivially enabled by the [word count](#) in an instruction, which makes it easier to add new instructions without breaking existing tools.

1.5. Debuggability

SPIR-V can decorate, with a text string, virtually anything created in the shader: types, variables, functions, etc. This is required for externally visible symbols, and also allowed for naming the result of any instruction. This can be used to aid in understandability when disassembling or debugging lowered versions of SPIR-V.

Location information (file names, lines, and columns) can be interleaved with the instruction stream to track the origin of each instruction.

1.6. Design Principles

Regularity. All instructions start with a word count. This allows walking a SPIR-V module without decoding each opcode. All instructions have an opcode that dictates for all operands what kind of operand they are. For instructions with a variable number of operands, the number of variable operands is known by subtracting the number of non-variable words from the instruction's word count.

Non Combinatorial. There is no combinatorial type explosion or need for large encode/decode tables for types. Rather, types are parameterized. Image types declare their dimensionality, arrayness, etc. all orthogonally, which greatly simplify code. This is done similarly for other types. It also applies to opcodes. Operations are orthogonal to scalar/vector size, but not to integer vs. floating-point differences.

Modeless. After a given execution model (e.g., pipeline stage) is specified, internal operation is essentially modeless: Generally, it follows the rule: "same spelling, same semantics", and does not have mode bits that modify semantics. If a change to SPIR-V modifies semantics, it should use a different spelling. This makes consumers of SPIR-V much more robust. There are execution modes declared, but these generally affect the way the module interacts with its execution environment, not its internal semantics. Capabilities are also declared, but this is to declare the subset of functionality that is used, not to change any semantics of what is used.

Declarative. SPIR-V declares externally-visible modes like "writes depth", rather than having rules that require deduction from full shader inspection. It also explicitly declares what addressing modes, execution model, extended instruction sets, etc. will be used. See [Language Capabilities](#) for more information.

SSA. All results of intermediate operations are strictly SSA. However, declared variables reside in memory and use load/store for access, and such variables can be stored to multiple times.

IO. Some storage classes are for input/output (IO) and, fundamentally, IO is done through load/store of variables declared in these storage classes.

1.7. Static Single Assignment (SSA)

SPIR-V includes a phi instruction to allow the merging together of intermediate results from split control flow. This allows split control flow without load/store to memory. SPIR-V is flexible in the degree to which load/store is used; it is possible to use control flow with no phi-instructions, while still staying in SSA form, by using memory load/store.

Some storage classes are for IO and, fundamentally, IO is done through load/store, and initial load and final store won't be eliminated. Other storage classes are shader local and can have their load/store eliminated. It can be considered an optimization to largely eliminate such loads/stores by moving them into intermediate results in SSA form.

1.8. Built-In Variables

SPIR-V identifies built-in variables from a high-level language with an enumerant decoration. This assigns any unusual semantics to the variable. Built-in variables are otherwise declared with their correct SPIR-V type and treated the same as any other variable.

1.9. Specialization

Specialization enables offline creation of a portable SPIR-V module based on constant values that won't be known until a later point in time. For example, to size a fixed array with a constant not known during creation of a module, but known when the module will be lowered to the target architecture.

See [Specialization](#) in the next section for more details.

1.10. Example

The SPIR-V form is binary, not human readable, and fully described in [Binary Form](#). This is an example disassembly to give a basic idea of what SPIR-V looks like:

GLSL fragment shader:

```
#version 450

in vec4 color1;
in vec4 multiplier;
noperspective in vec4 color2;
out vec4 color;

struct S {
    bool b;
    vec4 v[5];
    int i;
};

uniform blockName {
    S s;
    bool cond;
};

void main()
{
    vec4 scale = vec4(1.0, 1.0, 2.0, 1.0);

    if (cond)
        color = color1 + s.v[2];
    else
        color = sqrt(color2) * scale;

    for (int i = 0; i < 4; ++i)
        color *= multiplier;
}
```

Corresponding SPIR-V:

```
; Magic:      0x07230203 (SPIR-V)
; Version:    0x00010000 (Version: 1.0.0)
; Generator:  0x00080001 (Khronos Glslang Reference Front End; 1)
; Bound:      63
; Schema:     0

                OpCapability Shader
%1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Fragment %4 "main" %31 %33 %42 %57
```


OpExecutionMode %4 OriginLowerLeft

; Debug information

OpSource GLSL 450
OpName %4 "main"
OpName %9 "scale"
OpName %17 "S"
OpMemberName %17 0 "b"
OpMemberName %17 1 "v"
OpMemberName %17 2 "i"
OpName %18 "blockName"
OpMemberName %18 0 "s"
OpMemberName %18 1 "cond"
OpName %20 ""
OpName %31 "color"
OpName %33 "color1"
OpName %42 "color2"
OpName %48 "i"
OpName %57 "multiplier"

; Annotations (non-debug)

OpDecorate %15 ArrayStride 16
OpMemberDecorate %17 0 Offset 0
OpMemberDecorate %17 1 Offset 16
OpMemberDecorate %17 2 Offset 96
OpMemberDecorate %18 0 Offset 0
OpMemberDecorate %18 1 Offset 112
OpDecorate %18 Block
OpDecorate %20 DescriptorSet 0
OpDecorate %42 NoPerspective

; All types, variables, and constants

%2 = OpTypeVoid
%3 = OpTypeFunction %2 ; void ()
%6 = OpTypeFloat 32 ; 32-bit float
%7 = OpTypeVector %6 4 ; vec4
%8 = OpTypePointer Function %7 ; function-local vec4*
%10 = OpConstant %6 1
%11 = OpConstant %6 2
%12 = OpConstantComposite %7 %10 %10 %11 %10 ; vec4(1.0, 1.0, 2.0, 1.0)
%13 = OpTypeInt 32 0 ; 32-bit int, sign-less
%14 = OpConstant %13 5
%15 = OpTypeArray %7 %14
%16 = OpTypeInt 32 1
%17 = OpTypeStruct %13 %15 %16
%18 = OpTypeStruct %17 %13
%19 = OpTypePointer Uniform %18
%20 = OpVariable %19 Uniform
%21 = OpConstant %16 1
%22 = OpTypePointer Uniform %13
%25 = OpTypeBool

```

%26 = OpConstant %13 0
%30 = OpTypePointer Output %7
%31 = OpVariable %30 Output
%32 = OpTypePointer Input %7
%33 = OpVariable %32 Input
%35 = OpConstant %16 0
%36 = OpConstant %16 2
%37 = OpTypePointer Uniform %7
%42 = OpVariable %32 Input
%47 = OpTypePointer Function %16
%55 = OpConstant %16 4
%57 = OpVariable %32 Input

```

```

; All functions

```

```

    %4 = OpFunction %2 None %3                ; main()
    %5 = OpLabel
    %9 = OpVariable %8 Function
    %48 = OpVariable %47 Function
           OpStore %9 %12
    %23 = OpAccessChain %22 %20 %21            ; location of cond
    %24 = OpLoad %13 %23                      ; load 32-bit int from cond
    %27 = OpINotEqual %25 %24 %26             ; convert to bool
           OpSelectionMerge %29 None           ; structured if
           OpBranchConditional %27 %28 %41      ; if cond
    %28 = OpLabel                             ; then
    %34 = OpLoad %7 %33
    %38 = OpAccessChain %37 %20 %35 %21 %36    ; s.v[2]
    %39 = OpLoad %7 %38
    %40 = OpFAdd %7 %34 %39
           OpStore %31 %40
           OpBranch %29
    %41 = OpLabel                             ; else
    %43 = OpLoad %7 %42
    %44 = OpExtInst %7 %1 Sqrt %43             ; extended instruction sqrt
    %45 = OpLoad %7 %9
    %46 = OpFMul %7 %44 %45
           OpStore %31 %46
           OpBranch %29
    %29 = OpLabel                             ; endif
           OpStore %48 %35
           OpBranch %49
    %49 = OpLabel
           OpLoopMerge %51 %52 None           ; structured loop
           OpBranch %53
    %53 = OpLabel
    %54 = OpLoad %16 %48
    %56 = OpSlessThan %25 %54 %55             ; i < 4 ?
           OpBranchConditional %56 %50 %51     ; body or break
    %50 = OpLabel                             ; body
    %58 = OpLoad %7 %57
    %59 = OpLoad %7 %31

```

```
%60 = OpFMul %7 %59 %58
      OpStore %31 %60
      OpBranch %52
%52 = OpLabel                                ; continue target
%61 = OpLoad %16 %48
%62 = OpIAdd %16 %61 %21                    ; ++i
      OpStore %48 %62
      OpBranch %49                          ; loop back
%51 = OpLabel                                ; loop merge point
      OpReturn
      OpFunctionEnd
```

Chapter 2. Specification

2.1. Language Capabilities

A SPIR-V module is consumed by a client API that needs to support the features used by that SPIR-V module. Features are classified through [capabilities](#). Capabilities used by a particular SPIR-V module are declared early in that module with the [OpCapability](#) instruction. Then:

- A validator can validate that the module uses only its declared capabilities.
- A client API is allowed to reject modules declaring capabilities it does not support.

All available capabilities and their dependencies form a capability hierarchy, fully listed in the [capability](#) section. Only top-level capabilities need to be explicitly declared; their dependencies are implicitly declared.

If an instruction, enumerant, or other feature specifies multiple enabling capabilities, only one such capability needs to be declared to use the feature. This declaration does not itself imply anything about the presence of the other enabling capabilities: The execution environment needs to support only the declared capability.

The SPIR-V specification provides universal capability-specific validation rules, in the [validation](#) section. Additionally, each client API includes the following:

- Which capabilities in the [capability](#) section it supports or requires, and hence allows in a SPIR-V module.
- Any additional validation rules it has beyond those specified by the SPIR-V specification.
- Required limits, if they are beyond the [Universal Limits](#).

2.2. Terms

2.2.1. Instructions

Word: 32 bits.

<id>: A numerical name; the name used to refer to an object, a type, a function, a label, etc. An *<id>* always consumes one [word](#). The *<id>*s defined by a module obey [SSA](#).

Result <id>: Most instructions define a result, named by an *<id>* explicitly provided in the instruction. The *Result <id>* is used as an operand in other instructions to refer to the instruction that defined it.

Literal: An immediate value, not an *<id>*. Literals larger than one [word](#) consume multiple operands, one per word. An instruction states what type the literal will be interpreted as. A string is interpreted as a nul-terminated stream of characters. All string comparisons are case sensitive. The character set is Unicode in the UTF-8 encoding scheme. The UTF-8 octets (8-bit bytes) are packed four per [word](#), following the little-endian convention (i.e., the first octet is in the lowest-order 8 bits of the word). The final word contains the string's nul-termination character (0), and all contents past the end of the string in the final word are padded with 0. For a numeric literal, the lower-order words appear first. If a numeric type's bit width is less than 32-bits, the value appears in the low-order bits of the word, and the high-order bits must be 0 for a [floating-point type](#) or [integer type](#) with *Signedness* of 0, or sign extended for an integer type with a *Signedness* of 1 (similarly for the remaining bits of widths larger than 32 bits but not a multiple of 32 bits).

Operand: A one-[word](#) argument to an instruction. E.g., it could be an *<id>*, or (or part of) a [literal](#). Which form it holds is always explicitly known from the opcode.

WordCount: The complete number of [words](#) taken by an instruction, including the word holding the word count and opcode, and any optional operands. An instruction's word count is the total space taken by the instruction.

Instruction: After a header, a module is simply a linear list of instructions. An instruction contains a [word count](#), an opcode, an optional [Result <id>](#), an optional [<id>](#) of the instruction's type, and a variable list of operands. All instruction opcodes and semantics are listed in [Instructions](#).

Decoration: Auxiliary information such as built-in variable, stream numbers, invariance, interpolation type, relaxed precision, etc., added to [<id>s](#) or structure-type members through [Decorations](#). Decorations are enumerated in [Decoration](#) in the [Binary Form](#) section.

Object: An instantiation of a non-void type, either as the [Result <id>](#) of an operation, or created through [OpVariable](#).

Memory Object: An object created through [OpVariable](#). Such an object exists only for the duration of a function if it is a function variable, and otherwise exists for the duration of an invocation.

Memory Object Declaration: An [OpVariable](#), or an [OpFunctionParameter](#) of pointer type, or the contents of an [OpVariable](#) that holds either a pointer to the [PhysicalStorageBuffer storage class](#) or an array of such pointers.

Intermediate Object or *Intermediate Value* or *Intermediate Result*: An object created by an operation (not memory allocated by [OpVariable](#)) and dying on its last consumption.

Constant Instruction: Either a specialization-constant instruction or a non-specialization constant instruction: Instructions that start "OpConstant" or "OpSpec".

[a, b]: This square-bracket notation means the range from *a* to *b*, inclusive of *a* and *b*. Parentheses exclude their end point, so, for example, *(a, b]* means *a* to *b* excluding *a* but including *b*.

Non-Semantic Instruction: An instruction that has no semantic impact, and that can be safely removed from the module.

Hint: Either an indication to the compiler a property is likely to be observed or a request to the compiler to perform a specific transformation. They do not affect the semantics of the program. Unless stated otherwise, the compiler must not assume the property will be observed or the transformation is always safe to be performed.

2.2.2. Types

Boolean type: The type declared by [OpTypeBool](#).

Integer type: Any width signed or unsigned type from [OpTypeInt](#). By convention, the lowest-order bit is referred to as bit-number 0, and the highest-order bit as bit-number *Width* - 1.

Floating-point type: Any width and encoding type from [OpTypeFloat](#).

Numerical type: An [integer](#) type or a [floating-point](#) type.

Scalar: A single instance of a [numerical type](#) or [Boolean type](#). Scalars are also called *components* when being discussed either by themselves or in the context of the contents of a [vector](#).

Vector: An ordered homogeneous collection of two or more [scalars](#). Vector sizes are quite restrictive and dependent on the execution model.

Matrix: An ordered homogeneous collection of vectors. The vectors forming a matrix are also called its

columns. Matrix sizes are quite restrictive and dependent on the execution model.

Array: An ordered homogeneous aggregate of any non-void-type objects. The objects forming an array are also called its *elements*. Array sizes are generally not restricted.

Structure: An ordered heterogeneous aggregate of any non-void types. The objects forming a structure are also called its *members*.

Aggregate: A [structure](#) or an [array](#).

Composite: An [aggregate](#), a [matrix](#), or a [vector](#).

Texel: A single scalar or vector element of the data collection described by an [image](#). Each texel is stored in a particular [format](#). If the *Sampled Type* operand of the [image type](#) is not [OpTypeVoid](#), the value is converted according to the *Sampled Type* operand when the texel is read or written.

Image: An opaque descriptor of an ordered, homogeneous, multi-dimensional collection of formatted data elements called texels. Image objects themselves are opaque and cannot be accessed or modified; an image's texels are accessed through dedicated [Image instructions](#). An image type is declared with [OpTypeImage](#). An image does not include any information about how to access, filter, or sample it.

Sampler: Settings that describe how to access, filter, or sample an [image](#). Comes either from literal declarations of settings or from an opaque reference to externally bound settings. A sampler does not include an [image](#).

Sampled Image: An [image](#) combined with a [sampler](#), enabling filtered accesses of the image's contents.

Physical Pointer Type: An [OpTypePointer](#) whose *Storage Class* uses physical addressing according to the [addressing model](#).

Logical Pointer Type: A pointer type that is not a [physical pointer type](#).

Concrete Type: A [numerical](#) scalar, vector, or matrix type, or [physical pointer type](#), or any aggregate containing only these types.

Abstract Type: An [OpTypeVoid](#) or [OpTypeBool](#), or [logical pointer type](#), or any aggregate type containing any of these.

Opaque Type: A type that is, or contains, or points to, or contains pointers to, any of the following types:

- [OpTypeImage](#)
- [OpTypeSampler](#)
- [OpTypeSampledImage](#)
- [OpTypeOpaque](#)
- [OpTypeEvent](#)
- [OpTypeDeviceEvent](#)
- [OpTypeReserveld](#)
- [OpTypeQueue](#)
- [OpTypePipe](#)
- [OpTypeForwardPointer](#)
- [OpTypePipeStorage](#)
- [OpTypeNamedBarrier](#)

Variable pointer: A pointer of logical pointer type that results from one of the following instructions:

- [OpSelect](#)
- [OpPhi](#)
- [OpFunctionCall](#)
- [OpPtrAccessChain](#)
- [OpLoad](#)
- [OpConstantNull](#)

Additionally, any [OpAccessChain](#), [OpInBoundsAccessChain](#), or [OpCopyObject](#) that takes a variable pointer as an operand also produces a variable pointer. An [OpFunctionParameter](#) of pointer type is a variable pointer if any [OpFunctionCall](#) to the function statically passes a variable pointer as the value of the parameter.

Explicit Layout: Types with an explicit layout have decorations defining the relative locations of all of their constituents. A type has an explicit layout if the following statements are true, recursively applied to any nested types:

- Each structure-type member must have an **Offset** decoration.
- Each array type must have an **ArrayStride** decoration, unless it is an array that contains a structure decorated with **Block** or **BufferBlock**, in which case it must not have an **ArrayStride** decoration.
- Each structure-type member that is a matrix or array-of-matrices must be decorated with a **MatrixStride** decoration, and one of the **RowMajor** or **ColMajor** decorations.
- **ArrayStride**, **MatrixStride**, and **Offset** decorations must not cause overlap between elements or with other members.
- Each **ArrayStride** and **MatrixStride** must be greater than zero.
- A pointer to a structure decorated with **Block** or **BufferBlock** must not have an **ArrayStride** decoration
- All members of a given structure must have distinct **Offset** decorations.

2.2.3. Computation

Remainder: When dividing a by b , a *remainder* r is defined to be a value that satisfies $r + q \times b = a$ where q is an integer and $|r| < |b|$.

2.2.4. Module

Module: A single unit of SPIR-V. It can contain multiple [entry points](#), but only one set of [capabilities](#).

Entry Point: A function in a [module](#) where execution begins. A single *entry point* is limited to a single [execution model](#). An entry point is declared using [OpEntryPoint](#).

Execution Model: A graphical-pipeline stage or OpenCL kernel. These are enumerated in [Execution Model](#).

Execution Mode: Modes of operation relating to the interface or execution environment of the module. These are enumerated in [Execution Mode](#). Generally, modes do not change the semantics of instructions within a SPIR-V module.

Vertex Processor: Any stage or execution model that processes vertices: Vertex, tessellation control, tessellation evaluation, and geometry. Explicitly excludes fragment and compute execution models.

2.2.5. Control Flow

Block: A contiguous sequence of instructions starting with an **OpLabel**, ending with a **block termination instruction**. A *block* has no additional label or block termination instructions.

Function Termination Instruction: One of the following, used to terminate execution of a function:

- **OpReturn**
- **OpReturnValue**
- **OpKill**
- **OpUnreachable**
- **OpTerminateInvocation**

Conditional Branch Instruction: One of the following, used as a **block termination instruction**:

- **OpBranchConditional**
- **OpSwitch**

Branch Instruction: an **OpBranch** or a **conditional branch instruction**, used as a **block termination instruction**

Block Termination Instruction: One of the following, used to terminate blocks:

- any **branch instruction**
- any **function termination instruction**

Control-Flow Graph: The graph formed by a function's blocks and branches. The blocks are the graph's nodes, and the branches the graph's edges.

CFG: Control-flow graph.

Merge Instruction: One of the following, used before a branch instruction to declare structured control flow:

- **OpSelectionMerge**
- **OpLoopMerge**

Header Block: A block containing a **merge instruction**.

Loop Header: A **header block** whose merge instruction is an **OpLoopMerge**.

Selection Header: A **header block** whose merge instruction is an **OpSelectionMerge** and whose termination instruction is an **OpBranchConditional**.

Switch Header: A **header block** whose merge instruction is an **OpSelectionMerge** and whose termination instruction is an **OpSwitch**.

Merge Block: A block declared by the *Merge Block* operand of a **merge instruction**.

Branch Edge: There is a *branch edge* from block *A* to block *B* if the terminator of *A* is a **branch instruction** and *B* is one of the target blocks for the branch instruction.

Merge Edge: There is a *merge edge* from block *A* to block *B* if *A* contains a **merge instruction** and *B* is the **merge block** of this merge instruction.

Continue Edge: There is a *continue edge* from block *A* to block *B* if *A* is a [loop header](#) and *B* is the *Continue Target* of the loop header's [OpLoopMerge](#) instruction.

Structured Control-Flow Edge: There is a *structured control-flow edge* from block *A* to block *B* if there is a [branch edge](#), [merge edge](#), or [continue edge](#) from *A* to *B*.

Back Edge: A [branch edge](#) that branches to one of its ancestors in a depth-first search over [structured control-flow edges](#) starting at the function's entry block.

Note: When all loops are structured, each *back edge* corresponds to exactly one loop header, and vice versa, making this set of back edges invariant with respect to which depth-first search found them. This implies that the CFG defined by the function's structured control-flow edges is reducible.

Back-Edge Block: If there is a [back edge](#) from block *A* to block *B* then *A* is a *back-edge block*.

Path: A sequence of blocks B_0, B_1, \dots, B_n where for each $0 \leq i < n$ there is a [branch edge](#) from B_i to B_{i+1} . This forms a *path* from B_0 to B_n .

Structured Control-Flow Path: A sequence of blocks B_0, B_1, \dots, B_n where for each $0 \leq i < n$ there is a [structured control-flow edge](#) from B_i to B_{i+1} . This forms a *structured control-flow path* from B_0 to B_n .

Structurally Reachable: A block *B* is *structurally reachable* if there exists a [structured control-flow path](#) from the entry block of the function containing *B* to *B*.

Dominate: A block *A* *dominates* a block *B*, where *A* and *B* are in the same function, if every [path](#) from the function's entry block to block *B* includes block *A*. *A strictly dominates B* if *A dominates B* and *A* and *B* are different blocks.

Structurally Dominate: A block *A* *structurally dominates* a block *B*, where *A* and *B* are in the same function, if every [structured control-flow path](#) from the function's entry block to block *B* includes block *A*. *A strictly structurally dominates B* if *A structurally dominates B* and *A* and *B* are different blocks.

Structurally Post Dominate: A block *B* *structurally post dominates* a block *A*, where *A* and *B* are in the same function, if every [structured control-flow path](#) from *A* to a [function termination instruction](#) includes block *B*.

Invocation: A single execution of an entry point in a SPIR-V module, operating only on the amount of data explicitly exposed by the semantics of the instructions. (Any implicit operation on additional instances of data would comprise additional invocations.) For example, in compute execution models, a single invocation operates only on a single work item, or, in a vertex execution model, a single invocation operates only on a single vertex.

Quad: The execution environment can partition invocations into *quads*, where invocations within a quad can synchronize and share data with each other efficiently. See the client API specification for more details. It has a size of exactly 4 invocations.

Quad index: The index of an invocation in a [quad](#).

Subgroup: Invocations are partitioned into subgroups, where invocations within a subgroup can synchronize and share data with each other efficiently. In compute models, the current workgroup is a superset of the subgroup. A subgroup's size is defined by the maximum of the current values of the **SubgroupSize** and **SubgroupMaxSize** [built-in variables](#).

Cluster: A partition of invocations in a subgroup. Invocations are partitioned into clusters based on their subgroup local invocation ID and the per-instruction cluster size *ClusterSize*, with *ClusterSize* invocations per cluster. The first *ClusterSize* invocations with the smallest subgroup local invocation IDs are assigned to the first cluster, then the next *ClusterSize* remaining invocations with the smallest local invocation IDs are assigned to the next cluster, and so on. If the current value of the **SubgroupSize** [built-in variable](#) is not evenly divisible by the cluster size then the additional invocations in the last cluster are considered not part

of the [tangle](#).

Workgroup: The set of invocations partitioned in some execution models (e.g. GLCompute, Kernel) as a workgroup. Its size is defined statically by either the **WorkgroupSize** [built-in](#) or the **LocalSize** or **LocalSizeId** [Execution Modes](#), or can be queried via the **WorkgroupSize** [built-in](#). These values can be defined in multiple dimensions, and its total size is the product of the size in each specified dimension.

Invocation Group: The complete set of invocations collectively processing a particular compute workgroup or graphical operation, where the scope of a "graphical operation" is implementation dependent, but at least as large as a single point, line, triangle, or patch, and at most as large as a single rendering command, as defined by the client API.

Derivative Group: Defined only for the **Fragment Execution Model**: The set of invocations collectively processing derivatives, which is at most as large as a single point, line, or triangle, including any helper invocations, as defined by the client API.

Scope: A specific set of invocations that are related to each other as defined by [Scope <id>](#). Each invocation belongs to one or more *scopes*, but belongs to no more than one *scope* for each [Scope <id>](#).

Tangle: The set of invocations that execute the same [dynamic instance](#) of an instruction.

Tangled invocations: Invocations in the same tangle.

Scope Restricted Tangle: A set of invocations in the same [tangle](#) and within the same [scope](#).

Tangled Instruction: One of:

- [Group and subgroup instructions](#)
- [Non-uniform instructions](#)
- **OpControlBarrier**
- **OpGroupReserveReadPipePackets**, **OpGroupReserveWritePipePackets**,
OpGroupCommitReadPipe and **OpGroupCommitWritePipe**
- [Derivative instructions](#)
- [Image instructions](#) that consume an implicit derivative

Tangled instructions communicate between invocations.

Dynamic Instance: Within a single invocation, a single static instruction can be executed multiple times, giving multiple dynamic instances of that instruction. This can happen if the instruction is executed in a loop, or in a function called from multiple call sites, or combinations of multiple of these. Different loop iterations and different dynamic function-call-site chains yield different dynamic instances of such an instruction.

Additionally, a single dynamic instance may be executed by multiple invocations. At the [entry point](#), all invocations (in the [invocation group](#), unless otherwise stated) execute the same dynamic instance of the first instruction in the entry point function. Invocations will continue to execute the same dynamic instances as long as they follow the same control-flow path. When invocations execute a [conditional branch](#) and begin following different control flow paths, they execute different dynamic instances according to the path taken. Invocations that have taken different control flow paths may resume executing the same dynamic instances if their execution reaches the same static instruction. Invocations may only resume executing the same dynamic instances when all invocations reach the same static instruction. Unless otherwise indicated, the only reconvergence conditions are those described in the definition of [uniform control flow](#).

Program Order. Program order is an ordering on [dynamic instances](#) of instructions executed by a single

shader invocation. A [dynamic instance](#) A' of an instruction A is program-ordered before a [dynamic instance](#) B' of an instruction B (and B' is program-ordered after A') if and only if:

- A and B are in the same basic block, A is listed in the module before B , and A' is the n 'th dynamic instance of A and B' is the n 'th dynamic instance of B .
- A is a [branch instruction](#), B is [OpLabel](#), and A' branches to B' .
- A is [OpFunctionCall](#), B is [OpFunction](#), and A' calls B' .
- A is [OpReturn](#) or [OpReturnValue](#), and B' is program-ordered after the [OpFunctionCall](#) which called the function which executed A' .
- A' is program-ordered before a [dynamic instance](#) X' , and X' is program-ordered before B' .

Dynamically Uniform: An [<id>](#) is dynamically uniform for a [dynamic instance](#) consuming it if its value is the same for all invocations (in the [invocation group](#), unless otherwise stated) that execute that dynamic instance.

Uniform Control Flow: Uniform control flow (or converged control flow) is the state when all invocations (in the [invocation group](#), unless otherwise stated) execute the same dynamic instance of an instruction. Uniform control flow is the initial state at the entry point, and lasts until a [conditional branch](#) takes different control paths for different invocations (non-uniform or divergent control flow). Such divergence can reconverge, with all the invocations once again executing the same control-flow path, and this re-establishes the existence of uniform control flow. If control flow is uniform upon entry into a structured loop or selection, and all invocations leave that loop or selection via the header block's declared merge block, then control flow reconverges to be uniform at that merge block.

2.2.6. Validity and Defined Behavior

Most SPIR-V rules are expressed statically. These *statically expressed rules* are based on what can be seen with a direct static examination of the module in the specific places the rule says to look. These are expressed using terms like *must*, *must not*, *valid*, *not valid*, and *invalid*. Such rules establish whether the module is classified as valid or not valid, which in turn provides terms that tools may use in labeling and describing modules they process. A module is valid only if it does not violate any of these statically expressed rules. Such rules might not be considered violated if a specialization constant is involved, as described in the [specialization constant section](#).

Some SPIR-V rules say that *behavior is not defined*, that something results in *undefined behavior*, or that *behavior is defined* only under some circumstances. These all refer only to something that happens dynamically while an invocation of a shader or kernel executes.

An invocation having undefined behavior is independent of a module being valid. Tools containing smart transforms may be able to deduce from a static module that behavior will be undefined if some part were to be executed. However, this does not allow the tool to classify the module as invalid.

Sometimes, SPIR-V refers to the client API to specify what is statically valid or dynamically defined for a specific situation, in which case those rules come from the client API's execution environment. Otherwise, a SPIR-V client API can define an execution environment that adds additional statically expressed rules, further constraining what SPIR-V itself said was valid. However, a client cannot remove any such statically expressed rules. A client will not remove any undefined behavior specified by SPIR-V.

2.3. Physical Layout of a SPIR-V Module and Instruction

A SPIR-V module is a single linear stream of *words*. The first words are shown in the following table:

Table 1. First Words of Physical Layout

Word Number	Contents
0	Magic Number.
1	Version number. The bytes are, high-order to low-order: <i>0 Major Number Minor Number 0</i> Hence, version 1.3 is the value 0x00010300.
2	Generator's magic number. It is associated with the tool that generated the module. Its value does not affect any semantics, and is allowed to be 0. Using a non-0 value is encouraged, and can be registered with Khronos at https://github.com/KhronosGroup/SPIRV-Headers .
3	<i>Bound</i> ; where all <i><id>s</i> in this module are guaranteed to satisfy $0 < id < Bound$ <i>Bound</i> should be small, smaller is better, with all <i><id></i> in a module being densely packed and near 0.
4	0 (Reserved for instruction schema, if needed.)
5	First word of instruction stream, see below.

All remaining words are a linear sequence of instructions.

Each instruction is a stream of *words*:

Table 2. Instruction Physical Layout

Instruction Word Number	Contents
0	Opcode: The 16 high-order bits are the <i>WordCount</i> of the instruction. The 16 low-order bits are the opcode enumerant.
1	Optional instruction type <i><id></i> (presence determined by opcode).
.	Optional instruction <i>Result <id></i> (presence determined by opcode).
.	Operand 1 (if needed)
.	Operand 2 (if needed)

Instruction Word Number	Contents
...	...
<i>WordCount</i> - 1	Operand <i>N</i> (<i>N</i> is determined by <i>WordCount</i> minus the 1 to 3 words used for the opcode, instruction type <i><id></i> , and instruction <i>Result <id></i>).

Instructions are variable length due both to having optional instruction type *<id>* and *Result <id>* words as well as a variable number of operands. The details for each specific instruction are given in the [Binary Form](#) section.

2.4. Logical Layout of a Module

The instructions of a SPIR-V module must be in the following order. For sections earlier than function definitions, it is invalid to use instructions other than those indicated.

1. All [OpCapability](#) instructions.
2. Optional [OpExtension](#) instructions (extensions to SPIR-V).
3. Optional [OpExtInstImport](#) instructions.
4. The single required [OpMemoryModel](#) instruction.
5. All entry point declarations, using [OpEntryPoint](#).
6. All [execution-mode](#) declarations, using [OpExecutionMode](#) or [OpExecutionModelId](#).
7. These [debug](#) instructions, which must be grouped in the following order:
 - a. All [OpString](#), [OpSourceExtension](#), [OpSource](#), and [OpSourceContinued](#), without forward references.
 - b. All [OpName](#) and all [OpMemberName](#).
 - c. All [OpModuleProcessed](#) instructions.
8. All [annotation](#) instructions:
 - a. All decoration instructions.
9. All type declarations ([OpTypeXXX](#) instructions), all [constant instructions](#), and all global variable declarations (all [OpVariable](#) instructions whose [Storage Class](#) is not **Function**). This is the preferred location for [OpUndef](#) instructions, though they can also appear in function bodies. All operands in all these instructions must be declared before being used. Otherwise, they can be in any order. This section is the first section to allow use of:
 - a. [OpLine](#) and [OpNoLine](#) debug information.
 - b. [Non-semantic instructions](#) with [OpExtInst](#).
10. All function declarations ("declarations" are functions without a body; there is no forward declaration to a function with a body). A function declaration is as follows.
 - a. Function declaration, using [OpFunction](#).
 - b. Function parameter declarations, using [OpFunctionParameter](#).
 - c. Function end, using [OpFunctionEnd](#).
11. All function definitions (functions with a body). A function definition is as follows.
 - a. Function definition, using [OpFunction](#).

- b. Function parameter declarations, using **OpFunctionParameter**.
- c. Block.
- d. Block.
- e. ...
- f. Function end, using **OpFunctionEnd**.

Within a function definition:

- A block always starts with an **OpLabel** instruction. This may be immediately preceded by an **OpLine** instruction, but the **OpLabel** is considered as the beginning of the block.
- A block always ends with a **block termination instruction** (see [validation rules](#) for more detail).
- All **OpVariable** instructions in a function must have a **Storage Class** of **Function**.
- All **OpVariable** instructions in a function must be in the first block in the function. These instructions, together with any intermixed **OpLine** and **OpNoLine** instructions, must be the first instructions in that block. (Note the validation rules prevent **OpPhi** instructions in the first block of a function.)
- A function definition (starts with **OpFunction**) can be immediately preceded by an **OpLine** instruction.

Forward references (an operand *<id>* that appears before the *Result <id>* defining it) are allowed for:

- Operands that are an **OpFunction**. This allows for recursion and early declaration of entry points.
- **Annotation**-instruction operands. This is required to fully know everything about a type or variable once it is declared.
- Labels.
- **OpPhi** can contain forward references.
- **OpTypeForwardPointer**:
 - An **OpTypeForwardPointer** *Pointer Type* is a forward reference to an **OpTypePointer**.
 - Subsequent consumption of an **OpTypeForwardPointer** *Pointer Type* can be a forward reference.
- The list of *<id>* provided in the **OpEntryPoint** instruction.
- **OpExecutionModel**.

In all cases, there is enough type information to enable a single simple pass through a module to transform it. For example, function calls have all the type information in the call, phi-functions don't change type, and labels don't have type. The pointer forward reference allows structures to contain pointers to themselves or to be mutually recursive (through pointers), without needing additional type information.

The [Validation Rules](#) section lists additional rules.

2.5. Instructions

Most instructions create a *Result <id>*, as provided in the *Result <id>* field of the instruction. These *Result <id>*s are then referred to by other instructions through their *<id>* operands. All instruction operands are specified in the [Binary Form](#) section.

Instructions are explicit about whether an operand is (or is part of) a self-contained **literal** or an *<id>* referring to another instruction's result. While an *<id>* always takes one operand, one literal takes one or more operands. Some common examples of **literals**:

- A literal 32-bit (or smaller) integer is always one operand directly holding a 32-bit two's-complement

value.

- A literal 32-bit float is always one operand, directly holding a 32-bit IEEE 754 floating-point representation.
- A literal 64-bit float is always two operands, directly holding a 64-bit IEEE 754 representation. The low-order 32 bits appear in the first operand.

2.5.1. SSA Form

A module is always in static single assignment (SSA) form. That is, there is always exactly one instruction resulting in any particular *Result <id>*. Storing into variables declared in memory is not subject to this; such stores do not create *Result <id>*s. Accessing declared variables is done through:

- **OpVariable** to allocate an object in memory and create a *Result <id>* that is the name of a pointer to it.
- **OpAccessChain** or **OpInBoundsAccessChain** to create a pointer to a subpart of a *composite* object in memory.
- **OpLoad** through a pointer, giving the loaded object a *Result <id>* that can then be used as an operand in other instructions.
- **OpStore** through a pointer, to write a value. There is no *Result <id>* for an **OpStore**.

OpLoad and **OpStore** instructions can often be eliminated, using *intermediate* results instead. If this happens in multiple control-flow paths, these values need to be merged again at the path's merge point. Use **OpPhi** to merge such values together.

2.6. Entry Point and Execution Model

The **OpEntryPoint** instruction identifies an *entry point* with two key things: an execution model and a function definition. Execution models include **Vertex**, **GLCompute**, etc. (one for each graphical stage), as well as **Kernel** for OpenCL kernels. For the complete list, see [Execution Model](#). An **OpEntryPoint** also supplies a name that can be used externally to identify the entry point, and a declaration of all the **Input** and **Output** variables that form its input/output interface.

The static function call graphs rooted at two entry points are allowed to overlap, so that function definitions and global variable definitions can be shared. The execution model and any execution modes associated with an entry point apply to the entire static function call graph rooted at that entry point. This rule implies that a function appearing in both call graphs of two distinct entry points may behave differently in each case. Similarly, variables whose semantics depend on properties of an entry point, e.g. those using the **Input Storage Class**, may behave differently if used in call graphs rooted in two different entry points.

2.7. Execution Modes

Information like the following is declared with **OpExecutionMode** instructions. For example,

- number of invocations (**Invocations**)
- vertex-order CCW (**VertexOrderCcw**)
- triangle strip generation (**OutputTriangleStrip**)
- number of output vertices (**OutputVertices**)
- etc.

For a complete list, see [Execution Mode](#).

2.8. Types and Variables

Types are built up hierarchically, using **OpTypeXXX** instructions. The *Result <id>* of an **OpTypeXXX** instruction becomes a type *<id>* for future use where type *<id>*s are needed (therefore, **OpTypeXXX** instructions do not have a type *<id>*, like most other instructions do).

The "leaves" to start building with are types like **OpTypeFloat**, **OpTypeInt**, **OpTypeImage**, **OpTypeEvent**, etc. Other types are built up from the *Result <id>* of these. The numerical types are parameterized to specify bit width and signed vs. unsigned.

Higher-level types are then constructed using opcodes like **OpTypeVector**, **OpTypeMatrix**, **OpTypeImage**, **OpTypeArray**, **OpTypeRuntimeArray**, **OpTypeStruct**, and **OpTypePointer**. These are parameterized by number of components, array size, member lists, etc. The image types are parameterized by their sampling result type, dimensionality, arrayness, etc. To do sampling or filtering operations, a type from **OpTypeSampledImage** is used that contains both an *image* and a *sampler*. Such a *sampled image* can be set directly by the client API or combined in a SPIR-V module from an independent image and an independent sampler.

Types are built bottom up: A parameterizing operand in a type must be defined before being used.

Some additional information about the type of an *<id>* can be provided using the decoration instructions (**OpDecorate**, **OpMemberDecorate**, **OpGroupDecorate**, **OpGroupMemberDecorate**, and **OpDecorationGroup**). These can add, for example, **Invariant** to an *<id>* created by another instruction. See the full list of [Decorations](#) in the [Binary Form](#) section.

Two different type *<id>*s form, by definition, two different types. It is invalid to declare multiple non-aggregate, non-pointer type *<id>*s having the same opcode and operands. It is valid to declare multiple *aggregate* type *<id>*s having the same opcode and operands. This is to allow multiple instances of aggregate types with the same structure to be *decorated* differently. (Different decorations are not required; two different aggregate type *<id>*s are allowed to have identical declarations and decorations, and will still be two different types.) Pointer types are also allowed to have multiple *<id>*s for the same opcode and operands, to allow for differing decorations (e.g., **Volatile**) or different decoration values (e.g., different *Array Stride* values for the **ArrayStride**). If new pointers are formed, their types must be decorated as needed, so the consumer knows how to generate an access through the pointer.

Variables are declared to be of an already built type, and placed in a Storage Class. Storage classes include **UniformConstant**, **Input**, **Workgroup**, etc. and are fully specified in [Storage Class](#). Variables declared with the **Function** Storage Class can have their lifetime's specified within their function using the **OpLifetimeStart** and **OpLifetimeStop** instructions.

Intermediate results are typed by the instruction's type *<id>*, which is constrained by each instruction's description.

Built-in variables have special semantics and are declared using **OpDecorate** or **OpMemberDecorate** with the **BuiltIn Decoration**, followed by a **BuiltIn** enumerator. See the [BuiltIn](#) section for details on what can be decorated as a built-in variable.

2.8.1. Unsigned Versus Signed Integers

The integer type, **OpTypeInt**, is parameterized not only with a size, but also with signedness. There are two different ways to think about signedness in SPIR-V, both are internally consistent and acceptable:

1. As if all integers are "signless", meaning they are neither signed nor unsigned: All **OpTypeInt** instructions select a signedness of 0 to conceptually mean "no sign" (rather than "unsigned"). This is useful if translating from a language that does not distinguish between signed and unsigned types. The

type of operation (signed or unsigned) to perform is always selected by the choice of opcode.

2. As if some integers are signed, and some are unsigned: Some **OpTypeInt** instructions select signedness of 0 to mean "unsigned" and some select signedness of 1 to mean "signed". This is useful if signedness matters to external interface, or if targeting a higher-level language that cares about types being signed and unsigned. The type of operation (signed or unsigned) to perform is still always selected by the choice of opcode, but a small amount of validation can be done where it is non-sensible to use a signed type.

Note in both cases all signed and unsigned operations always work on unsigned types, and the semantics of operation come from the opcode. SPIR-V does not know which way is being used; it is set up to support both ways of thinking.

Note that while SPIR-V aims to not assign semantic meaning to the signedness bit in choosing how to operate on values, there are a few cases known to do this, all confined to modules declaring the **Shader** capability:

- validation for consistency checking for front ends for directly contradictory usage, where explicitly indicated in this specification
- interfaces that might require widening of an input value, and otherwise don't know whether to sign extend or zero extend, including the following bullet
- an image read that might require widening of an operand, in versions where the **SignExtend** and **ZeroExtend** [image operands](#) are not available (if available, these operands are the supported way to communicate this).

2.9. Function Calling

To call a function defined in the current module or a function declared to be imported from another module, use **OpFunctionCall** with an operand that is the *<id>* of the **OpFunction** to call, and the *<id>*s of the arguments to pass. All arguments are passed by value into the called function. This includes pointers, through which a callee object could be modified.

2.10. Extended Instruction Sets

Many operations and/or built-in function calls from high-level languages are represented through *extended instruction sets*. Extended instruction sets include things like

- trigonometric functions: `sin()`, `cos()`, ...
- exponentiation functions: `exp()`, `pow()`, ...
- geometry functions: `reflect()`, `smoothstep()`, ...
- functions having rich performance/accuracy trade-offs
- etc.

Non-extended instructions, those that are core SPIR-V instructions, are listed in the [Binary Form](#) section. Native operations include:

- Basic arithmetic: `+`, `-`, `*`, `min()`, scalar * vector, etc.
- Texturing, to help with back-end decoding and support special code-motion rules.
- Derivatives, due to special code-motion rules.

Extended instruction sets are specified in independent specifications, not in this specification. The separate extended instruction set specification specifies instruction opcodes, semantics, and instruction names.

To use an extended instruction set, first import it by name string using [OpExtInstImport](#) and giving it a [Result <id>](#):

```
<extinst-id> OpExtInstImport "name-of-extended-instruction-set"
```

Where "name-of-extended-instruction-set" is a [literal](#) string. The standard convention for this string is

```
"<source language name>.<package name>.<version>"
```

For example "GLSL.std.450" could be the name of the core built-in functions for GLSL versions 450 and earlier.

NOTE

There is nothing precluding having two "mirror" sets of instructions with different names but the same opcode values, which could, for example, let modifying just the import statement to change a performance/accuracy trade off.

Then, to call a specific extended instruction, use [OpExtInst](#):

```
OpExtInst <extinst-id> instruction-number operand0, operand1, ...
```

Extended instruction-set specifications provide semantics for each "instruction-number". It is up to the specific specification what the overloading rules are on operand type. The specification will be clear on its semantics, and producers/consumers of it must follow those semantics.

By convention, it is recommended that all external specifications include an **enum** {...} listing all the "instruction-numbers", and a mapping between these numbers and a string representing the instruction name. However, there are no requirements that instruction name strings are provided or mangled.

NOTE

Producing and consuming extended instructions can be done entirely through numbers (no string parsing). An extended instruction set specification provides opcode enumerant values for the instructions, and these are produced by the front end and consumed by the back end.

2.11. Structured Control Flow

SPIR-V can explicitly declare structured control-flow *constructs* using [merge instructions](#). These explicitly declare a [header block](#) before the control flow diverges and a [merge block](#) where control flow subsequently converges. (Control flow may partially or fully reconverge before reaching the merge block so long as it converges by the time the merge block is reached.) These blocks delimit constructs that must nest, and must be entered and exited in structured ways, as per the following.

2.11.1. Rules for Structured Control-flow Declarations

Structured control flow declarations must satisfy the following rules:

- the [merge block](#) declared by a [header block](#) must not be a merge block declared by any other header block
- each header block must [strictly structurally dominate](#) its merge block

- all **back edges** must branch to a **loop header**, with each loop header having exactly one back edge branching to it
- for a given loop header, its **merge block**, **OpLoopMerge Continue Target**, and corresponding **back-edge block**:
 - the *Continue Target* and *merge block* must be different blocks
 - the *loop header* must structurally dominate the *Continue Target*
 - the *Continue Target* must structurally dominate the *back-edge block*
 - the *back-edge block* must **structurally post dominate** the *Continue Target*

2.11.2. Structured Control-flow Constructs

A structured control-flow *construct* is defined as one of:

- a *selection construct*: the blocks structurally dominated by a **selection header**, excluding blocks structurally dominated by the selection header's merge block
- a *continue construct*: the blocks that are both structurally dominated by an **OpLoopMerge Continue Target** and structurally post dominated by the corresponding loop's back-edge block
- a *loop construct*: the blocks structurally dominated by a **loop header**, excluding both the loop header's *continue construct* and the blocks structurally dominated by the loop header's merge block
- a *switch construct*: the blocks structurally dominated by a **switch header**, excluding blocks structurally dominated by the switch header's merge block
- a *case construct*: the blocks structurally dominated by an **OpSwitch Target** or *Default* block, excluding the blocks structurally dominated by the **OpSwitch** construct's corresponding merge block (note that as a consequence of this definition, an **OpSwitch Target** or *Default* block that is equal to the **OpSwitch's** corresponding merge block does not give rise to a case construct)

2.11.3. Rules for Structured Control-flow Constructs

Below, we will use the following terminology:

- A branch edge from block *A* to block *B* *exits* a structured control-flow construct *S* if and only if *A* is contained in *S* and *B* is not contained in *S*
- A *single-block loop* is a loop construct where the loop's header block, continue target and back-edge block are all the same.
- The *header block* of a continue construct is the continue target of the associated loop.
- The *header block* of a case construct is the **OpSwitch Target** or *Default* block that defines the case construct.

If the header block of a structured control-flow construct is **structurally reachable** then that structured control-flow construct must satisfy the following rules:

- if a branch edge from block *A* to block *B* exits the structured control-flow construct *S*, then the exit must correspond to one of the following:
 - Breaking from a selection construct: *S* is a selection construct, *S* is the innermost structured control-flow construct containing *A*, and *B* is the merge block for *S*
 - Breaking from the innermost loop: *S* is the innermost loop construct containing *A*, and *B* is the merge block for *S*
 - Entering the innermost loop's continue construct: *S* is the innermost loop construct containing *A*, and *B* is the continue target for *S*

- Next loop iteration: the branch edge from *A* to *B* is a **back edge** (so that *S* is the continue construct of the associated loop)
- Branching from back-edge block to loop merge: *A* is the back-edge block for a loop construct (so that *S* is the continue construct of the associated loop), and *B* is the merge block for the loop construct
- Branching from one case construct to another: *S* is a case construct associated with an **OpSwitch** instruction, and *B* is a target block or default block associated with the **OpSwitch** instruction
- Breaking from the innermost switch construct without breaking from a loop: *S* is the innermost switch construct containing *A*, *B* is the merge block for *S*, and the branch from *A* to *B* does not exit a loop construct
- a branch edge that exits a continue construct must branch to the header block or merge block of the associated loop
- for a loop construct that is not a single block loop, if there is a branch edge from a block *B* to the loop's continue target that is not a **back edge**, then *B* must belong to the loop construct
- if a structured control-flow construct *S* contains the header block for a selection, loop or switch construct different from *S*, then *S* must also contain that construct's merge block
- all branches into a selection, loop or switch construct from structurally-reachable blocks outside the construct must be to the construct's header block
- for a switch construct *S* with associated **OpSwitch** instruction:
 - the header block for *S* must **structurally dominate** every *case construct* associated with *S*
 - each *case construct* associated with *S* must not branch to more than one other *case construct* associated with *S*
 - each *case construct* associated with *S* must not be branched to by more than one other *case construct* associated with *S*
 - if *T1* and *T2* appear as labels of targets in the **OpSwitch** instruction and the case construct defined by *T1* branches to the case construct defined by *T2* then the last target with label *T1* must immediately precede the first target with label *T2* in the list of **OpSwitch Target** operands
 - if *T1* and *T2* appear as labels of targets in the **OpSwitch** instruction and the case construct defined by *T1* branches to the *Default* case construct of the **OpSwitch** which in turn branches to the case construct defined by *T2*, then either:
 - the block that defines the *Default* case construct must appear as a target label in the **OpSwitch** instruction, or
 - the last target with label *T1* must immediately precede the first target with label *T2* in the list of **OpSwitch Target** operands
 - for any label *T*, all targets with label *T* must appear consecutively in the list of **OpSwitch Target** operands

2.12. Specialization

Specialization is intended for constant objects that will not have known constant values until after initial generation of a SPIR-V module. Such objects are called *specialization constants*.

A SPIR-V module containing specialization constants can consume one or more externally provided *specializations*: A set of final constant values for some subset of the module's *specialization constants*. Applying these final constant values yields a new module having fewer remaining specialization constants. A module also contains default values for any specialization constants that never get externally specialized.

NOTE

No optimizing transforms are required to make a *specialized* module functionally correct. The specializing transform is straightforward and explicitly defined below.

NOTE

Ad hoc specializing should not be done through constants (**OpConstant** or **OpConstantComposite**) that get overwritten: A SPIR-V -> SPIR-V transform might want to do something irreversible with the value of such a constant, unconstrained from the possibility that its value could be later changed.

Within a module, a *Specialization Constant* is declared with one of these instructions:

- **OpSpecConstantTrue**
- **OpSpecConstantFalse**
- **OpSpecConstant**
- **OpSpecConstantComposite**
- **OpSpecConstantOp**

The *literal* operands to **OpSpecConstant** are the default numerical specialization constants. Similarly, the "True" and "False" parts of **OpSpecConstantTrue** and **OpSpecConstantFalse** provide the default Boolean specialization constants. These default values make an external specialization optional. However, such a default constant is applied only after all external specializations are complete, and none contained a specialization for it.

An external specialization is provided as a logical list of pairs. Each pair is a **SpecId Decoration** of a scalar specialization instruction along with its specialization constant. The numeric values are exactly what the operands would be to a corresponding **OpConstant** instruction. Boolean values are true if non-zero and false if zero.

Specializing a module is straightforward. The following specialization-constant instructions can be updated with specialization constants. These can be replaced in place, leaving everything else in the module exactly the same:

```
OpSpecConstantTrue -> OpConstantTrue or OpConstantFalse
OpSpecConstantFalse -> OpConstantTrue or OpConstantFalse
OpSpecConstant -> OpConstant
OpSpecConstantComposite -> OpConstantComposite
```

Note that the **OpSpecConstantOp** instruction is not one that can be updated with a specialization constant.

The **OpSpecConstantOp** instruction is specialized by executing the operation and replacing the instruction with the result. The result can be expressed in terms of a *constant instruction* that is not a specialization-constant instruction. (Note, however, this resulting instruction might not have the same size as the original instruction, so is not a "replaced in place" operation.)

When applying an external specialization, the following (and only the following) will be modified to be non-specialization-constant instructions:

- specialization-constant instructions with values provided by the specialization
- specialization-constant instructions that consume nothing but non-specialization constant instructions (including those that the partial specialization transformed from specialization-constant instructions; these are in order, so it is a single pass to do so)

A full specialization can also be done, when requested or required, in which all specialization-constant instructions will be modified to non-specialization-constant instructions, using the default values where required.

If a [statically expressed rule](#) would be broken due to the value of a constant, and that constant is a specialization constant, then that rule is not violated. (Consequently, specialization-constant default values are not relevant to the validity of the module.)

2.13. Linkage

The ability to have partially linked modules and libraries is provided as part of the [Linkage](#) capability.

By default, functions and global variables are private to a module and cannot be accessed by other modules. However, a module may be written to *export* or *import* functions and global (module scope) variables. Imported functions and global variable definitions are resolved at linkage time. A module is considered to be partially linked if it depends on imported values.

Within a module, imported or exported values are decorated using the **Linkage Attributes Decoration**. This decoration assigns the following linkage attributes to decorated values:

- A [Linkage Type](#).
- A *name*, interpreted as a [literal](#) string, is used to uniquely identify exported values.

NOTE

When resolving imported functions, the [Function Control](#) and all [Function Parameter Attributes](#) are taken from the function definition, and not from the function declaration.

2.14. Relaxed Precision

The **RelaxedPrecision Decoration** allows 32-bit integer and 32-bit floating-point operations to execute with a relaxed precision of somewhere between 16 and 32 bits.

For a floating-point operation, operating at relaxed precision means that the minimum requirements for range and precision are as follows:

- the floating point range may be as small as $(-2^{14}, 2^{14})$
- the floating point magnitude range includes 0.0 and $[2^{-14}, 2^{14})$
- the relative floating point precision may be as small as 2^{-10}

The [range notation](#) here means the largest required magnitude is half of the relative precision less than the value given.

Relative floating-point precision is defined as the worst case (i.e. largest) ratio of the smallest step in relation to the value for all non-zero values in the required range:

$$\text{Precision}_{\text{relative}} = (\text{abs}(v_1 - v_2))_{\min} / \text{abs}(v_1)_{\max} \text{ for } v_1 \neq 0, v_2 \neq 0, v_1 \neq v_2$$

It is therefore twice the maximum rounding error when converting from a real number. Subnormal numbers may be supported and may have lower relative precision.

For integer operations, operating at relaxed precision means that the operation is evaluated by an operation in which, for some N , $16 \leq N \leq 32$:

- the operation is executed as though its type were N bits in size, and

- the result is zero or sign extended to 32 bits as determined by the signedness of the result type of the operation.

The **RelaxedPrecision** [Decoration](#) must only be applied to:

- The [<id>](#) of an **OpVariable**, where it refers to the value of the variable.
- The [<id>](#) of an **OpFunctionParameter**, where it refers to the value of the parameter.
- The [Result <id>](#) of an instruction that reads or filters from an image. E.g. **OpImageSampleExplicitLod**, meaning the instruction is to operate at relaxed precision.
- The [Result <id>](#) of an **OpFunction**, where it refers to the value returned by the function.
- A structure-type member (through **OpMemberDecorate**).
- The [Result <id>](#) of an **OpFunctionCall**, where it refers to the result of the function call.
- The [Result <id>](#) of other instructions that operate on numerical types, meaning the instruction is to operate at relaxed precision. The instruction's operands may also be truncated to the relaxed precision.

In all cases, the types of the values that the **RelaxedPrecision** [Decoration](#) refers to must be:

- a scalar, vector, or matrix, or array of scalars, vectors, or matrices, and all the components in the types must be a 32-bit [numerical](#) type,
- a pointer to such a type, where it refers to the value pointed to.

The values that the **RelaxedPrecision** [Decoration](#) refers to can be truncated to relaxed precision.

When applied to a variable, function parameter, or structure member, all loads and stores from the decorated object may be treated as though they were [decorated](#) with **RelaxedPrecision**. Loads may also be decorated with **RelaxedPrecision**, in which case they are treated as operating at relaxed precision.

All loads and stores involving relaxed precision still read and write 32 bits of data, respectively. Floating-point data read or written in such a manner is written in full 32-bit floating-point format. However, a load or store might reduce the precision (as allowed by **RelaxedPrecision**) of the destination value.

For debugging portability of floating-point operations, **OpQuantizeToF16** may be used to explicitly reduce the precision of a relaxed-precision result to 16-bit precision. (Integer-result precision can be reduced, for example, using left- and right-shift opcodes.)

For image-sampling operations, decorations can appear on both the sampling instruction and the image variable being sampled. If either is decorated, they both should be decorated, and if both are decorated their decorations must match. If only one is decorated, the sampling instruction can behave either as if both were decorated or neither were decorated.

2.15. Debug Information

Debug information is supplied with:

- Source-code text through **OpString**, **OpSource**, and **OpSourceContinued**.
- Object names through **OpName** and **OpMemberName**.
- Line numbers through **OpLine** and **OpNoLine**.

A module does not lose any semantics when all such instructions are removed.

2.15.1. Function-Name Mangling

There is no functional dependency on how functions are named. Signature-typing information is explicitly provided, without any need for name "unmangling".

By convention, for debugging purposes, modules with **OpSource** *Source Language* of OpenCL use the Itanium name-mangling standard.

2.16. Validation Rules

2.16.1. Universal Validation Rules

- When using **OpBitcast** to convert pointers to/from vectors of integers, only vectors of 32-bit integers are allowed.
- If neither the **VariablePointers** nor **VariablePointersStorageBuffer** capabilities are declared, the following rules apply to logical pointer types:
 - **OpVariable** must not allocate an object whose type is or contains a logical pointer type.
 - It is invalid for a pointer to be an operand to any instruction other than:
 - **OpLoad**
 - **OpStore**
 - **OpAccessChain**
 - **OpInBoundsAccessChain**
 - **OpFunctionCall**
 - **OpImageTexelPointer**
 - **OpCopyMemory**
 - **OpCopyObject**
 - **OpArrayLength**
 - all **OpAtomic** instructions
 - extended instruction-set instructions that are explicitly identified as taking pointer operands
 - It is invalid for a pointer to be the *Result <id>* of any instruction other than:
 - **OpVariable**
 - **OpAccessChain**
 - **OpInBoundsAccessChain**
 - **OpFunctionParameter**
 - **OpImageTexelPointer**
 - **OpCopyObject**
 - All indexes in **OpAccessChain** and **OpInBoundsAccessChain** that are **OpConstant** with type of **OpTypeInt** with a *signedness* of 1 must not have their sign bit set.
 - Any pointer operand to an **OpFunctionCall** must point into one of the following storage classes:
 - **UniformConstant**
 - **Function**
 - **Private**

- **Workgroup**
- **AtomicCounter**
- Any pointer operand to an **OpFunctionCall** must be
 - a **memory object declaration**, or
 - a pointer to an element in an array that is a memory object declaration, where the element type is **OpTypeSampler** or **OpTypeImage**.
- The instructions **OpPtrEqual** and **OpPtrNotEqual** must not be used.
- If the **VariablePointers** or **VariablePointersStorageBuffer** **capability** is declared, the following are additionally allowed for **logical pointer types**, while other prohibitions remain:
 - If **OpVariable** allocates an object whose type is or contains a **logical pointer type**, the *Storage Class* operand of the **OpVariable** must be one of the following:
 - **Function**
 - **Private**
 - If a pointer is the *Object* operand of **OpStore** or result of **OpLoad**, the storage class the pointer is stored to or loaded from must be one of the following:
 - **Function**
 - **Private**
 - A pointer type can be the:
 - *Result Type* of **OpFunction**
 - *Result Type* of **OpFunctionCall**
 - *Return Type* of **OpTypeFunction**
 - A pointer can be a **variable pointer**
 - A pointer can be an operand to one of:
 - **OpReturnValue**
 - **OpPtrAccessChain**
 - **OpPtrEqual**
 - **OpPtrNotEqual**
 - **OpPtrDiff**
 - A **variable pointer** must point to one of the following **storage classes**:
 - **StorageBuffer**
 - **Workgroup** (if the **VariablePointers** **capability** is declared)
 - If the **VariablePointers** **capability** is not declared, a variable pointer must be selected from pointers pointing into the same structure or be **OpConstantNull**.
 - A pointer operand to **OpFunctionCall** can point into the **storage class**:
 - **StorageBuffer**
 - For pointer operands to **OpFunctionCall**, the **memory object declaration**-restriction is removed for the following **storage classes**:
 - **StorageBuffer**
 - **Workgroup**
 - The instructions **OpPtrEqual** and **OpPtrNotEqual** can be used only if the *Storage Class* of the

operands' **OpTypePointer** declaration is

- **StorageBuffer** if the **VariablePointersStorageBuffer** [capability](#) is explicitly or implicitly declared, whether or not operands point into the same buffer, or
 - **Workgroup**, which can be used only if the **VariablePointers** [capability](#) was declared.
- A [variable pointer](#) must not:
 - be an operand to an **OpArrayLength** instruction
 - point to an array of structures with a structure type decorated with **Block** or **BufferBlock**.
 - point to an object that is or contains an **OpTypeMatrix**
 - point to a column, or a component in a column, within an **OpTypeMatrix**
 - Memory model
 - Memory accesses that use **NonPrivatePointer** must use pointers in the **Uniform**, **Workgroup**, **CrossWorkgroup**, **Generic**, **Image**, or **StorageBuffer** [storage classes](#).
 - If the **Vulkan** [memory model](#) is declared and any instruction uses **Device** [scope](#), the **VulkanMemoryModelDeviceScope** [capability](#) must be declared.
 - Physical storage buffer
 - If the [addressing model](#) is not **PhysicalStorageBuffer64**, then the **PhysicalStorageBuffer** [storage class](#) must not be used.
 - **OpVariable** must not use the **PhysicalStorageBuffer** [storage class](#).
 - Any pointer value whose [storage class](#) is **PhysicalStorageBuffer** and that points to a matrix, an array of matrices, or a row or element of a matrix must be the result of an **OpAccessChain** or **OpPtrAccessChain** instruction whose *Base* operand is a structure type (or recursively must be the result of a sequence of only access chains from a structure to the final value). Such a pointer must only be used as the *Pointer* operand to **OpLoad** or **OpStore**.
 - The result type of **OpConstantNull** must not be a pointer type with [storage class](#) **PhysicalStorageBuffer**.
 - Operands to **OpPtrEqual**, **OpPtrNotEqual**, and **OpPtrDiff** must not be pointers into the **PhysicalStorageBuffer** [storage class](#).
 - SSA
 - Each [<id>](#) must appear exactly once as the [Result <id>](#) of an instruction.
 - The definition of an SSA [<id>](#) should dominate all uses of it, with the following exceptions:
 - Function calls may call functions not yet defined. However, note that the function's operand and return types are already known at the call site.
 - An **OpPhi** can consume definitions that do not dominate it.
 - Entry Point
 - There is at least one **OpEntryPoint** instruction, unless the [Linkage](#) capability is declared.
 - It is invalid for any function to be targeted by both an **OpEntryPoint** instruction and an **OpFunctionCall** instruction.
 - Each **OpEntryPoint** must not set more than one of the **DenormFlushToZero** or **DenormPreserve** [execution modes](#) for any given *Target Width*.
 - Each **OpEntryPoint** must not set more than one of the **RoundingModeRTE** or **RoundingModeRTZ** [execution modes](#) for any given *Target Width*.
 - Each **OpEntryPoint** must contain at most one of **LocalSize**, **LocalSizeId**, **LocalSizeHint**, or **LocalSizeHintId** [Execution Modes](#).

- Functions
 - A function declaration (an **OpFunction** with no basic blocks), must have a **Linkage Attributes Decoration** with the **Import Linkage Type**.
 - A function definition (an **OpFunction** with basic blocks) must not be **decorated** with the **Import Linkage Type**.
 - A function must not have both a declaration and a definition (no forward declarations).
- Global (Module Scope) Variables
 - A module-scope **OpVariable** with an *Initializer* operand must not be decorated with the **Import Linkage Type**.
- Control-Flow Graph (CFG)
 - Blocks exist only within a function.
 - The first block in a function definition is the entry point of that function and must not be the target of any branch. (Note this means it has no **OpPhi** instructions.)
 - The order of blocks in a function must satisfy the rule that blocks appear before all blocks they dominate.
 - Each block starts with a label.
 - A label is made by **OpLabel**.
 - This includes the first block of a function (**OpFunction** is not a label).
 - Labels are used only to form blocks.
 - The last instruction of each block is a **block termination instruction**.
 - Each **block termination instruction** must be the last instruction in a block.
 - Each **OpLabel** instruction must be within a function.
 - All **branches** within a function must be to labels in that function.
- All **OpFunctionCall** *Function* operands are an *<id>* of an **OpFunction** in the same module.
- Data rules
 - Scalar floating-point types must be parameterized only as 32 bit, plus any additional sizes enabled by **capabilities**.
 - Scalar integer types must be parameterized only as 32 bit, plus any additional sizes enabled by **capabilities**.
 - Vector types must be parameterized only with numerical types or the **OpTypeBool** type.
 - Vector types must be parameterized only with 2, 3, or 4 components, plus any additional sizes enabled by **capabilities**.
 - Matrix types must be parameterized only with floating-point types.
 - Matrix types must be parameterized only with 2, 3, or 4 columns.
 - Specialization constants (see **Specialization**) are limited to integers, Booleans, floating-point numbers, and vectors of these.
 - Image, sampler, and sampled image objects must not appear as operands to **OpPhi** instructions, or **OpSelect** instructions, or any instructions other than the image or sampler instructions specified to operate on them.
 - All **OpSampledImage** instructions, or instructions that load an image or sampler reference, must be in the same block in which their *Result <id>* are consumed.
 - The **capabilities** **StorageBuffer16BitAccess**, **UniformAndStorageBuffer16BitAccess**,

StoragePushConstant16, and **StorageInputOutput16** do not generally add 16-bit operations. Rather, they add only the following specific abilities:

- An **OpTypePointer** pointing to a 16-bit scalar, a 16-bit vector, or a composite containing a 16-bit member can be used as the result type of **OpVariable**, or **OpAccessChain**, or **OpInBoundsAccessChain**.
- **OpLoad** can load 16-bit scalars, 16-bit vectors, and 16-bit matrices.
- **OpStore** can store 16-bit scalars, 16-bit vectors, and 16-bit matrices.
- **OpCopyObject** can be used for 16-bit scalars or composites containing 16-bit members.
- 16-bit scalars or 16-bit vectors can be used as operands to a width-only conversion instruction to another allowed type (**OpFConvert**, **OpSConvert**, or **OpUConvert**), and can be produced as results of a width-only conversion instruction from another allowed type.
- A structure containing a 16-bit member can be an operand to **OpArrayLength**.
- The capabilities **StorageBuffer8BitAccess**, **UniformAndStorageBuffer8BitAccess**, and **StoragePushConstant8**, do not generally add 8-bit operations. Rather, they add only the following specific abilities:
 - An **OpTypePointer** pointing to an 8-bit scalar, an 8-bit vector, or a composite containing an 8-bit member can be used as the result type of **OpVariable**, or **OpAccessChain**, or **OpInBoundsAccessChain**.
 - **OpLoad** can load 8-bit scalars and vectors.
 - **OpStore** can store 8-bit scalars and 8-bit vectors.
 - **OpCopyObject** can be used for 8-bit scalars or composites containing 8-bit members.
 - 8-bit scalars and vectors can be used as operands to a width-only conversion instruction to another allowed type (**OpSConvert**, or **OpUConvert**), and can be produced as results of a width-only conversion instruction from another allowed type.
 - A structure containing an 8-bit member can be an operand to **OpArrayLength**.
- Decoration rules
 - The **Linkage Attributes Decoration** must not be applied to functions targeted by an **OpEntryPoint** instruction.
 - A **BuiltIn Decoration** must be applied only as follows:
 - If applied to a structure-type member, all members of that structure type must also be **decorated** with **BuiltIn**. (No allowed mixing of built-in variables and non-built-in variables within a single structure.)
 - If applied to a structure-type member, that structure type must not be contained as a member of another structure type.
 - There must be no more than one object per Storage Class that contains a structure type containing members **decorated** with **BuiltIn**, consumed per entry-point.
- **OpLoad** and **OpStore** must consume only objects whose type is a pointer.
- A **Result <id>** resulting from an instruction within a function must be used only in that function.
- A function call must have the same number of arguments as the function definition (or declaration) has parameters, and their respective types must match.
- An instruction requiring a specific number of operands must have that many operands. The **word count** must agree.
- Each opcode specifies its own requirements for number and type of operands, and these must be followed.

- Atomic access rules
 - The pointers taken by atomic operation instructions must be a pointer into one of the following **Storage Classes**:
 - **Uniform** when used with the **BufferBlock Decoration**
 - **StorageBuffer**
 - **PhysicalStorageBuffer**
 - **Workgroup**
 - **CrossWorkgroup**
 - **Generic**
 - **AtomicCounter**
 - **Image**
 - **Function**
- It is invalid to have a construct that uses the **StorageBuffer Storage Class** and a construct that uses the **Uniform Storage Class** with the **BufferBlock Decoration** in the same SPIR-V module.
- All **XfbStride Decorations** must be the same for all objects decorated with the same **XfbBuffer XFB Buffer Number**.
- All **Stream Decorations** must be the same for all objects decorated with the same **XfbBuffer XFB Buffer Number**.
- If the workgroup size is statically specified (using the **LocalSize**, **LocalSizeId** execution modes, or the **WorkgroupSize BuiltIn**), the product of all workgroup size dimensions must not be zero.

2.16.2. Validation Rules for Shader Capabilities

- CFG:
 - Loops must be structured. That is, the target basic block of a **back edge** must contain an **OpLoopMerge** instruction.
 - Selections must be structured. That is, an **OpSelectionMerge** instruction is required to precede:
 - an **OpSwitch** instruction
 - an **OpBranchConditional** instruction that has different *True Label* and *False Label* operands where neither are declared **merge blocks** or *Continue Targets*.
- Entry point and execution model
 - Each **entry point** in a module, along with its corresponding static call tree within that module, forms a complete pipeline stage.
 - Each **OpEntryPoint** with the **Fragment Execution Model** must have an **OpExecutionMode** for either the **OriginLowerLeft** or the **OriginUpperLeft Execution Mode**. (Exactly one of these is required.)
 - An **OpEntryPoint** with the **Fragment Execution Model** must not set more than one of the **DepthGreater**, **DepthLess**, or **DepthUnchanged Execution Modes**.
 - An **OpEntryPoint** with one of the **Tessellation Execution Models** must not set more than one of the **SpacingEqual**, **SpacingFractionalEven**, or **SpacingFractionalOdd Execution Modes**.
 - An **OpEntryPoint** with one of the **Tessellation Execution Models** must not set more than one of the **Triangles**, **Quads**, or **Isolines Execution Modes**.
 - An **OpEntryPoint** with one of the **Tessellation Execution Models** must not set more than one of the **VertexOrderCw** or **VertexOrderCcw Execution Modes**.

- An **OpEntryPoint** with the **Geometry Execution Model** must set exactly one of the **InputPoints**, **InputLines**, **InputLinesAdjacency**, **Triangles**, or **TrianglesAdjacency Execution Modes**.
- An **OpEntryPoint** with the **Geometry Execution Model** must set exactly one of the **OutputPoints**, **OutputLineStrip**, or **OutputTriangleStrip Execution Modes**.
- For **structure** objects in the **Input** and **Output Storage Classes**, the following apply:
 - If applied to structure-type members, the **decorations Noperspective**, **Flat**, **Patch**, **Centroid**, and **Sample** must be applied only to the top-level members of the structure type. (Nested objects' types must not be structures whose members are decorated with these decorations.)
- Type Rules
 - All declared types are restricted to those types that are, or are contained within, valid types for an **OpVariable Result Type** or an **OpTypeFunction Return Type**.
 - Aggregate types for *intermediate objects* are restricted to those types that are a valid *Type* of an **OpVariable Result Type** in the global storage classes.
- Decorations
 - It is invalid to apply more than one of **Noperspective** or **Flat decorations** to the same object or member.
 - It is invalid to apply more than one of **Patch**, **Centroid**, or **Sample decorations** to the same object or member.
 - It is invalid to apply more than one of **Block** and **BufferBlock decorations** to a structure type.
 - **Block** and **BufferBlock decorations** must not decorate a structure type that is nested at any level inside another structure type decorated with **Block** or **BufferBlock**.
 - The **FPRoundingMode decoration** must be applied only to a width-only conversion instruction whose only uses are *Object* operands of **OpStore** instructions storing through a pointer to a 16-bit floating-point object in the **StorageBuffer**, **PhysicalStorageBuffer**, **Uniform**, or **Output Storage Classes**.
- All *<id>* used for **Scope <id>** and **Memory Semantics <id>** must be of an **OpConstant**.
- Atomic access rules
 - The pointers taken by atomic operation instructions are further restricted to not point into the **Function storage class**.

2.16.3. Validation Rules for Kernel Capabilities

- The *Signedness* in **OpTypeInt** must always be 0.

2.17. Universal Limits

These quantities are minimum limits for all implementations and validators. Implementations are allowed to support larger quantities. Client APIs may impose larger minimums. See [Language Capabilities](#).

Validators inform when these limits (or explicitly parameterized limits) are crossed.

Table 3. Limits

Limited Entity	Minimum Limit	
	Decimal	Hexadecimal
Characters in a literal string	65,535	FFFF
Result <i><id></i> bound See Physical Layout for the shader-specific bound.	4,194,303	3FFFFFF
Control-flow nesting depth Measured per function, in program order, counting the maximum number of OpBranch , OpBranchConditional , or OpSwitch that are seen without yet seeing their corresponding <i>Merge Block</i> , as declared by OpSelectionMerge or OpLoopMerge .	1023	3FF
Global variables (Storage Class other than Function)	65,535	FFFF
Local variables (Function Storage Class)	524,287	7FFFF
Decorations per target <i><id></i>	Number of entries in the Decoration table.	
Execution modes per entry point	255	FF
Indexes for OpAccessChain , OpInBoundsAccessChain , OpPtrAccessChain , OpInBoundsPtrAccessChain , OpCompositeExtract , and OpCompositeInsert	255	FF
Number of function parameters, per function declaration	255	FF
OpFunctionCall actual arguments	255	FF
OpExtInst actual arguments	255	FF
OpSwitch (literal, label) pairs	16,383	3FFF
OpTypeStruct members	16,383	3FFF
Structure nesting depth	255	FF

2.18. Memory Model

A memory model is chosen using a single [OpMemoryModel](#) instruction near the beginning of the module. This selects both an addressing model and a memory model.

The **Logical addressing model** means pointers are abstract, having no physical size or numeric value. In this mode, pointers must be created only from existing objects, and they must not be stored into an object, unless additional **capabilities**, e.g., **VariablePointers**, are declared to add such functionality.

The non-**Logical addressing models** allow physical pointers to be formed. **OpVariable** can be used to create objects that hold pointers. These are declared for a specific **Storage Class**. Pointers for one Storage Class must not be used to access objects in another Storage Class. However, they can be converted with conversion opcodes. Any particular addressing model describes the bit width of pointers for each of the storage classes.

2.18.1. Memory Layout

Offset, **MatrixStride**, and **ArrayStride Decorations** partially define how a memory buffer is laid out. In addition, the following also define layout of a memory buffer, applied recursively as needed:

- a vector consumes contiguous memory with lower-numbered components appearing in smaller offsets than higher-numbered components, and with component 0 starting at the vector's **Offset Decoration**, if present
- in an array, lower-numbered elements appear at smaller offsets than higher-numbered elements, with element 0 starting at the **Offset Decoration** for the array, if present
- in a matrix, lower-numbered columns appear at smaller offsets than higher-numbered columns, and lower-numbered components within the matrix's vectors appearing at smaller offsets than higher-numbered components, with component 0 of column 0 starting at the **Offset Decoration**, if present (the **RowMajor** and **ColMajor Decorations** dictate what is contiguous)

2.18.2. Aliasing

Two **memory object declarations** are said to *alias* if they can be accessed (in bounds) such that both accesses address the same memory locations during their intersecting dynamic lifetimes. If two memory operations access the same locations, and at least one of them performs a write, the memory consistency model specified by the client API defines the results based on the ordering of the accesses.

How aliasing is managed depends on the **memory model**:

- The **Simple**, **GLSL**, and **Vulkan** memory models can assume that aliasing is generally not present between the **memory object declarations**. Specifically, the consumer is free to assume aliasing is not present between memory object declarations, unless the memory object declarations explicitly indicate they alias. Aliasing is indicated by applying the **Aliased decoration** to a memory object declaration's **<id>**, for **OpVariable** and **OpFunctionParameter**. Applying **Restrict** is allowed, but has no effect. For variables holding **PhysicalStorageBuffer** pointers, applying the **AliasedPointer** decoration on the **OpVariable** indicates that the **PhysicalStorageBuffer** pointers are potentially aliased. Applying **RestrictPointer** is allowed, but has no effect. Only those **memory object declarations** decorated with **Aliased** or **AliasedPointer** may alias each other.
- The **OpenCL** memory model assumes that **memory object declarations** might alias each other. An implementation may assume that memory object declarations decorated with **Restrict** will not alias any other memory object declaration. Applying **Aliased** is allowed, but has no effect.

The **Aliased** decoration can be used to express that certain **memory object declarations** may alias. Referencing the following table, a memory object declaration *P* may alias another declared pointer *Q* if within a single row:

- *P* is an instruction with opcode and storage class from the first pair of columns, and
- *Q* is an instruction with opcode and storage class from the second pair of columns.

First Storage Class	First Instruction(s)	Second Instructions	Second Storage Classes
CrossWorkgroup	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	CrossWorkgroup, Generic
Function	OpFunctionParameter	OpFunctionParameter, OpVariable	Function, Generic
Function	OpVariable	OpFunctionParameter	Function, Generic
Generic	OpFunctionParameter	OpFunctionParameter, OpVariable	CrossWorkgroup, Function, Generic, Workgroup
Image	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	Image, StorageBuffer, PhysicalStorageBuffer, Uniform, UniformConstant
Output	OpFunctionParameter	OpFunctionParameter, OpVariable	Output
Private	OpFunctionParameter	OpFunctionParameter, OpVariable	Private
StorageBuffer	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	Image, StorageBuffer, PhysicalStorageBuffer, Uniform, UniformConstant
PhysicalStorageBuffer	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	Image, StorageBuffer, PhysicalStorageBuffer, Uniform, UniformConstant
Uniform	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	Image, StorageBuffer, PhysicalStorageBuffer, Uniform, UniformConstant
UniformConstant	OpFunctionParameter, OpVariable	OpFunctionParameter, OpVariable	Image, StorageBuffer, PhysicalStorageBuffer, Uniform, UniformConstant
Workgroup	OpFunctionParameter	OpFunctionParameter, OpVariable	Workgroup, Generic
Workgroup	OpVariable	OpFunctionParameter	Workgroup, Generic

In addition to the above table, [memory object declarations](#) in the **CrossWorkgroup**, **Function**, **Input**, **Output**, **Private**, or **Workgroup** storage classes must also have matching pointee types for aliasing to be present. In all other cases the decoration is ignored.

Because aliasing, as described above, only applies to [memory object declarations](#), a consumer does not make any assumptions about whether or not memory regions of non memory object declarations overlap. As such, a consumer needs to perform dependency analysis on non memory object declarations if it wishes to reorder instructions affecting memory.

The memory locations associated with an **OpFunctionParameter** memory object declaration are dependent on the dynamic execution of the associated function. A dynamic instance of an **OpFunctionParameter** memory object declaration can be traced to either an **OpVariable** or an entry point **OpFunctionParameter**. During the execution of an entry point, behavior is undefined if operations on two distinct memory object declarations dynamically access the same memory locations during an intersection of the lifetimes of those two objects, with at least one of them performing a write, and at least one of the memory object declarations does not have the **Aliased** decoration (or is assumed to alias via the memory model).

For the **PhysicalStorageBuffer** storage class, **OpVariable** is understood to mean the **PhysicalStorageBuffer** pointer value(s) stored in the variable. An **Aliased PhysicalStorageBuffer** pointer stored in a **Function** variable can alias with other variables in the same function, global variables, or function parameters.

It is invalid to apply both **Restrict** and **Aliased** to the same *<id>*.

It is invalid to apply both **RestrictPointer** and **AliasedPointer** to the same *<id>*.

2.18.3. Null pointers

A "null pointer" can be formed from an **OpConstantNull** instruction with a pointer result type. The resulting pointer value is abstract, and will not equal the pointer value formed from any declared object or access chain into a declared object. Behavior is undefined if a load or store through **OpConstantNull** is executed.

2.19. Derivatives

Derivatives appear only in the **Fragment Execution Model**. They are either implicit or explicit. Some **image instructions** consume implicit derivatives, while the **derivative instructions** compute explicit derivatives. In all cases, derivatives are well defined when the **derivative group** has **uniform control flow**, otherwise see the client API specification for what behavior is allowed.

2.20. Code Motion

Texturing instructions in the **Fragment Execution Model** that rely on an implicit derivative won't be moved into control flow that is not known to be **uniform control flow** within each **derivative group**.

2.21. Deprecation

A feature may be marked as deprecated by a version of the specification or extension to the specification. Features marked as deprecated in one version of the specification are still present in that version, but future versions may reduce their support or completely remove them. Deprecating before removing allows applications time to transition away from the deprecated feature. Once the feature is removed, all tokens used exclusively by that feature will be reserved and any use of those tokens will become invalid.

2.22. Unified Specification

This document specifies all versions of **SPIR-V**.

There are three kinds of entries in the tables of enumerated tokens:

- **Reservation:** These say **Reserved** in the enabling capabilities. They often contain token names only, lacking a semantic description. They are invalid **SPIR-V** for any version, serving only to reserve the tokens. They may identify enabling capabilities and extensions, in which case any listed extensions

might add the tokens. See the listed extensions for additional information.

- **Conditional:** These say [Missing before](#) or [Missing after](#) in the enabling capabilities. They are invalid **SPIR-V** for the missing versions. They may identify enabling capabilities and extensions, in which case any listed extensions might add the tokens for some of the missing versions. See the listed extensions for additional information. For versions not identified as missing, the tokens are valid **SPIR-V**, subject to any listed enabling capabilities.
- **Universal:** These have no mention of what version they are missing in, or of being reserved. They are valid in all versions of **SPIR-V**.

2.23. Uniformity

SPIR-V has multiple notions of uniformity of values. A *Result* `<id>` [decorated](#) as **Uniform** (for a particular scope) is a contract that all invocations within that scope compute the same value for that result, for a given dynamic instance of an instruction. This is useful to enable implementations to store results in a scalar register file (*scalarization*), for example. Results are assumed not to be uniform unless decorated as such.

An `<id>` is defined to be [dynamically uniform](#) for a dynamic instance of an instruction if all invocations (in an invocation group) that execute the dynamic instance have the same value for that `<id>`. This is not something that is explicitly decorated, it is just a property that arises. This property is assumed to hold for operands of certain instructions, such as the *Image* operand of image instructions, unless that operand is decorated as **NonUniform**. Some implementations require more complex instruction expansions to handle non-dynamically uniform values in certain instructions, and thus it is mandatory for certain operands to be decorated as **NonUniform** if they are not guaranteed to be dynamically uniform.

While the names may suggest otherwise, nothing forbids an `<id>` from being decorated as both **Uniform** and **NonUniform**. Because *dynamically uniform* is at a larger scope (invocation group) than the default **Uniform** scope (subgroup), it is even possible for the `<id>` to be uniform at the subgroup scope but not dynamically uniform.

Chapter 3. Binary Form

This section contains the exact form for all instructions, starting with the numerical values for all fields. See [Physical Layout](#) for the order words appear in.

3.1. Magic Number

Magic number for a SPIR-V module.

TIP

Endianness: A module is defined as a stream of words, not a stream of bytes. However, if stored as a stream of bytes (e.g., in a file), the magic number can be used to deduce what endianness to apply to convert the byte stream back to a word stream.

Magic Number
0x07230203

3.2. Enumerants

3.2.1. Source Language

The source language is for debug purposes only, with no semantics that affect the meaning of other parts of the module.

Used by [OpSource](#).

Source Language		Enabling Capabilities
0	Unknown	
1	ESSL	
2	GLSL	
3	OpenCL_C	
4	OpenCL_CPP	
5	HLSL	
6	CPP_for_OpenCL	
7	SYCL	
8	HERO_C	
9	NZSL	
10	WGSL	
11	Slang	
12	Zig	
13	Rust	

3.2.2. Execution Model

Used by [OpEntryPoint](#) and [OpConditionalEntryPointINTEL](#).

Execution Model		Enabling Capabilities
0	Vertex Vertex shading stage.	Shader
1	TessellationControl Tessellation control (or hull) shading stage.	Tessellation
2	TessellationEvaluation Tessellation evaluation (or domain) shading stage.	Tessellation
3	Geometry Geometry shading stage.	Geometry
4	Fragment Fragment shading stage.	Shader
5	GLCompute Graphical compute shading stage.	Shader
6	Kernel Compute kernel.	Kernel
5267	TaskNV	MeshShadingNV Reserved.
5268	MeshNV	MeshShadingNV Reserved.
5313	RayGenerationKHR (RayGenerationNV)	RayTracingNV, RayTracingKHR Reserved.
5314	IntersectionKHR (IntersectionNV)	RayTracingNV, RayTracingKHR Reserved.
5315	AnyHitKHR (AnyHitNV)	RayTracingNV, RayTracingKHR Reserved.
5316	ClosestHitKHR (ClosestHitNV)	RayTracingNV, RayTracingKHR Reserved.
5317	MissKHR (MissNV)	RayTracingNV, RayTracingKHR Reserved.
5318	CallableKHR (CallableNV)	RayTracingNV, RayTracingKHR Reserved.

Execution Model		Enabling Capabilities
5364	TaskEXT	MeshShadingEXT Reserved.
5365	MeshEXT	MeshShadingEXT Reserved.

3.2.3. Addressing Model

Used by [OpMemoryModel](#).

Addressing Model		Enabling Capabilities
0	Logical	
1	Physical32 Indicates a 32-bit module, where the address width is equal to 32 bits.	Addresses
2	Physical64 Indicates a 64-bit module, where the address width is equal to 64 bits.	Addresses
5348	PhysicalStorageBuffer64 (PhysicalStorageBuffer64EXT) Indicates that pointers with a storage class of PhysicalStorageBuffer are physical pointer types with an address width of 64 bits, while pointers to all other storage classes are logical.	PhysicalStorageBufferAddresses Missing before version 1.5. Also see extensions: SPV_EXT_physical_storage_buffer , SPV_KHR_physical_storage_buffer

3.2.4. Memory Model

Used by [OpMemoryModel](#).

Memory Model		Enabling Capabilities
0	Simple Deprecated (use GLSL450). Memory model is undefined.	Shader
1	GLSL450 Memory model needed by later versions of GLSL and ESSL. Works across multiple versions.	Shader
2	OpenCL OpenCL memory model.	Kernel
3	Vulkan (VulkanKHR) Vulkan memory model , as specified by the client API. This memory model must be declared if and only if the VulkanMemoryModel capability is declared.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model

3.2.5. Execution Mode

Declare the modes an [entry point](#) executes in. All **Extra Operands** that are *<id>s* must be the *<id>s* of [constant instructions](#) unless otherwise stated. It is invalid to apply the same execution mode more than once to any entry point unless explicitly allowed below for a specific execution mode.

Used by [OpExecutionMode](#) and [OpExecutionModelId](#).

Execution Mode		Extra Operands	Enabling Capabilities
0	Invocations <i>Number of invocations</i> is an unsigned 32-bit integer number of times to invoke the geometry stage for each input primitive received. The default is to run once for each input primitive. It is invalid to specify a value greater than the target-dependent maximum. Only valid with the Geometry Execution Model .	<i>Literal</i> <i>Number of invocations</i>	Geometry
1	SpacingEqual Requests the tessellation primitive generator to divide edges into a collection of equal-sized segments. Only valid with one of the tessellation Execution Models .		Tessellation
2	SpacingFractionalEven Requests the tessellation primitive generator to divide edges into an even number of equal-length segments plus two additional shorter fractional segments. Only valid with one of the tessellation Execution Models .		Tessellation
3	SpacingFractionalOdd Requests the tessellation primitive generator to divide edges into an odd number of equal-length segments plus two additional shorter fractional segments. Only valid with one of the tessellation Execution Models .		Tessellation
4	VertexOrderCw Requests the tessellation primitive generator to generate triangles in clockwise order. Only valid with one of the tessellation Execution Models .		Tessellation

	Execution Mode	Extra Operands	Enabling Capabilities
5	VertexOrderCcw Requests the tessellation primitive generator to generate triangles in counter-clockwise order. Only valid with one of the tessellation Execution Models .		Tessellation
6	PixelCenterInteger Pixels appear centered on whole-number pixel offsets. E.g., the coordinate (0.5, 0.5) appears to move to (0.0, 0.0). Only valid with the Fragment Execution Model . If a Fragment entry point does not have this set, pixels appear centered at offsets of (0.5, 0.5) from whole numbers		Shader
7	OriginUpperLeft The coordinates decorated by FragCoord appear to originate in the upper left, and increase toward the right and downward. Only valid with the Fragment Execution Model .		Shader
8	OriginLowerLeft The coordinates decorated by FragCoord appear to originate in the lower left, and increase toward the right and upward. Only valid with the Fragment Execution Model .		Shader
9	EarlyFragmentTests Fragment tests are to be performed before fragment shader execution. Only valid with the Fragment Execution Model .		Shader
10	PointMode Requests the tessellation primitive generator to generate a point for each distinct vertex in the subdivided primitive, rather than to generate lines or triangles. Only valid with one of the tessellation Execution Models .		Tessellation

	Execution Mode	Extra Operands	Enabling Capabilities
11	Xfb This stage runs in transform feedback-capturing mode and this module is responsible for describing the transform-feedback setup. See the XfbBuffer , Offset , and XfbStride Decorations .		TransformFeedback
12	DepthReplacing This mode declares that this entry point dynamically writes the FragDepth -decorated variable. Behavior is undefined if this mode is declared and an invocation does not write to FragDepth , or vice versa. Only valid with the Fragment Execution Model .		Shader
14	DepthGreater Indicates that per-fragment tests may assume that any FragDepth built in -decorated value written by the shader is greater-than-or-equal to the fragment's interpolated depth value (given by the z component of the FragCoord built in -decorated variable). Other stages of the pipeline use the written value as normal. Only valid with the Fragment execution model .		Shader
15	DepthLess Indicates that per-fragment tests may assume that any FragDepth built in -decorated value written by the shader is less-than-or-equal to the fragment's interpolated depth value (given by the z component of the FragCoord built in -decorated variable). Other stages of the pipeline use the written value as normal. Only valid with the Fragment execution model .		Shader

Execution Mode		Extra Operands			Enabling Capabilities
16	DepthUnchanged Indicates that per-fragment tests may assume that any FragDepth <i>built in</i> -decorated value written by the shader is the same as the fragment's interpolated depth value (given by the z component of the FragCoord <i>built in</i> -decorated variable). Other stages of the pipeline use the written value as normal. Only valid with the Fragment <i>execution model</i> .				Shader
17	LocalSize Indicates the workgroup size in the x, y, and z dimensions. <i>x size</i> , <i>y size</i> , and <i>z size</i> are unsigned 32-bit integers. Only valid with the GLCompute or Kernel <i>Execution Models</i> .	<i>Literal</i> <i>x size</i>	<i>Literal</i> <i>y size</i>	<i>Literal</i> <i>z size</i>	
18	LocalSizeHint A hint to the compiler, which indicates the most likely to be used workgroup size in the x, y, and z dimensions. <i>x size</i> , <i>y size</i> , and <i>z size</i> are unsigned 32-bit integers. Only valid with the Kernel <i>Execution Model</i> .	<i>Literal</i> <i>x size</i>	<i>Literal</i> <i>y size</i>	<i>Literal</i> <i>z size</i>	Kernel
19	InputPoints Stage input primitive is <i>points</i> . Only valid with the Geometry <i>Execution Model</i> .				Geometry
20	InputLines Stage input primitive is <i>lines</i> . Only valid with the Geometry <i>Execution Model</i> .				Geometry
21	InputLinesAdjacency Stage input primitive is <i>lines adjacency</i> . Only valid with the Geometry <i>Execution Model</i> .				Geometry
22	Triangles For a geometry stage, input primitive is <i>triangles</i> . For a tessellation stage, requests the tessellation primitive generator to generate triangles. Only valid with the Geometry or one of the tessellation <i>Execution Models</i> .				Geometry, Tessellation

	Execution Mode	Extra Operands	Enabling Capabilities
23	InputTrianglesAdjacency Geometry stage input primitive is <i>triangles adjacency</i> . Only valid with the Geometry Execution Model .		Geometry
24	Quads Requests the tessellation primitive generator to generate <i>quads</i> . Only valid with one of the tessellation Execution Models .		Tessellation
25	Isolines Requests the tessellation primitive generator to generate <i>isolines</i> . Only valid with one of the tessellation Execution Models .		Tessellation
26	OutputVertices <i>Vertex Count</i> is an unsigned 32-bit integer. For a geometry stage, it is the maximum number of vertices the shader will ever emit in a single invocation . For a tessellation-control stage, it is the number of vertices in the output patch produced by the tessellation control shader, which also specifies the number of times the tessellation control shader is invoked. Only valid with the Geometry or one of the tessellation Execution Models .	<i>Literal</i> <i>Vertex count</i>	Geometry, Tessellation, MeshShadingNV, MeshShadingEXT
27	OutputPoints Stage output primitive is <i>points</i> . Only valid with the Geometry Execution Model .		Geometry, MeshShadingNV, MeshShadingEXT
28	OutputLineStrip Stage output primitive is <i>line strip</i> . Only valid with the Geometry Execution Model .		Geometry
29	OutputTriangleStrip Stage output primitive is <i>triangle strip</i> . Only valid with the Geometry Execution Model .		Geometry

	Execution Mode	Extra Operands	Enabling Capabilities
30	<p>VecTypeHint A hint to the compiler, which indicates that most operations used in the entry point are explicitly vectorized using a particular vector type. The 16 high-order bits of the <i>Vector Type</i> operand specify the <i>number of components</i> of the vector. The 16 low-order bits of the <i>Vector Type</i> operand specify the <i>data type</i> of the vector.</p> <p>These are the legal <i>data type</i> values: 0 represents an 8-bit integer value. 1 represents a 16-bit integer value. 2 represents a 32-bit integer value. 3 represents a 64-bit integer value. 4 represents a 16-bit IEEE 754 float value. 5 represents a 32-bit IEEE 754 float value. 6 represents a 64-bit IEEE 754 float value.</p> <p>Only valid with the Kernel Execution Model.</p>	<p><i>Literal</i> <i>Vector type</i></p>	Kernel
31	<p>ContractionOff Indicates that floating-point-expressions contraction is disallowed. Only valid with the Kernel Execution Model.</p>		Kernel
33	<p>Initializer Indicates that this entry point is a module initializer.</p>		Kernel <i>Missing before version 1.1.</i>
34	<p>Finalizer Indicates that this entry point is a module finalizer.</p>		Kernel <i>Missing before version 1.1.</i>
35	<p>SubgroupSize Indicates that this entry point requires the specified <i>Subgroup Size</i>. <i>Subgroup Size</i> is an unsigned 32-bit integer.</p>	<p><i>Literal</i> <i>Subgroup Size</i></p>	SubgroupDispatch <i>Missing before version 1.1.</i>

	Execution Mode	Extra Operands			Enabling Capabilities
36	SubgroupsPerWorkgroup Indicates that this entry point requires the specified number of <i>Subgroups Per Workgroup</i> . <i>Subgroups Per Workgroup</i> is an unsigned 32-bit integer.	<i>Literal</i> <i>Subgroups Per Workgroup</i>			SubgroupDispatch Missing before version 1.1.
37	SubgroupsPerWorkgroupId Same as the SubgroupsPerWorkgroup mode, but using an <i><id></i> operand instead of a literal. The operand is consumed as unsigned and must be an <i>integer type</i> scalar.	<i><id></i> <i>Subgroups Per Workgroup</i>			SubgroupDispatch Missing before version 1.2.
38	LocalSizeId Same as the LocalSize Mode , but using <i><id></i> operands instead of literals. The operands are consumed as unsigned and each must be an <i>integer type</i> scalar.	<i><id></i> x size	<i><id></i> y size	<i><id></i> z size	Missing before version 1.2.
39	LocalSizeHintId Same as the LocalSizeHint Mode , but using <i><id></i> operands instead of literals. The operands are consumed as unsigned and each must be an <i>integer type</i> scalar.	<i><id></i> x size hint	<i><id></i> y size hint	<i><id></i> z size hint	Kernel Missing before version 1.2.
4169	NonCoherentColorAttachmentReadEXT				TileImageColorReadAccessEXT Reserved.
4170	NonCoherentDepthAttachmentReadEXT				TileImageDepthReadAccessEXT Reserved.
4171	NonCoherentStencilAttachmentReadEXT				TileImageStencilReadAccessEXT Reserved.
4421	SubgroupUniformControlFlowKHR				Shader Reserved. Also see extension: SPV_KHR_subgroup_uniform_control_flow
4446	PostDepthCoverage				SampleMaskPostDepthCoverage Reserved. Also see extension: SPV_KHR_post_depth_coverage

	Execution Mode	Extra Operands	Enabling Capabilities
4459	DenormPreserve Any denormalized value input into a shader or potentially generated by any instruction in a shader is preserved. Denormalized values obtained via unpacking an integer into a vector of values with smaller bit width and interpreting those values as floating-point numbers is preserved. Only affects instructions operating on a floating-point type using the IEEE 754 encoding whose component width is <i>Target Width</i> . <i>Target Width</i> is an unsigned 32-bit integer. May be applied at most once per <i>Target Width</i> to any entry point.	<i>Literal</i> <i>Target Width</i>	DenormPreserve Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4460	DenormFlushToZero Any denormalized value input into a shader or potentially generated by any instruction in a shader is flushed to zero. Denormalized values obtained via unpacking an integer into a vector of values with smaller bit width and interpreting those values as floating-point numbers is flushed to zero. Only affects instructions operating on a floating-point type using the IEEE 754 encoding whose component width is <i>Target Width</i> . <i>Target Width</i> is an unsigned 32-bit integer. May be applied at most once per <i>Target Width</i> to any entry point.	<i>Literal</i> <i>Target Width</i>	DenormFlushToZero Missing before version 1.4 . Also see extension: SPV_KHR_float_controls

Execution Mode		Extra Operands	Enabling Capabilities
4461	<p>SignedZeroInfNanPreserve The implementation does not perform optimizations on floating-point instructions that do not preserve sign of a zero, or assume that operands and results are not NaNs or infinities. Bit patterns for NaNs might not be preserved.</p> <p>Only affects instructions operating on a floating-point type using the IEEE 754 encoding whose component width is <i>Target Width</i>. <i>Target Width</i> is an unsigned 32-bit integer. May be applied at most once per <i>Target Width</i> to any entry point.</p>	<p><i>Literal</i> <i>Target Width</i></p>	<p>SignedZeroInfNanPreserve</p> <p>Missing before version 1.4.</p> <p>Also see extension: SPV_KHR_float_controls</p>
4462	<p>RoundingModeRTE The default rounding mode for floating-point arithmetic and conversions instructions is round to nearest even. If an instruction is decorated with FPRoundingMode or defines a rounding mode in its description, that rounding mode is applied and RoundingModeRTE is ignored.</p> <p>Only affects instructions operating on a floating-point type using the IEEE 754 encoding whose component width is <i>Target Width</i>. <i>Target Width</i> is an unsigned 32-bit integer. May be applied at most once per <i>Target Width</i> to any entry point.</p>	<p><i>Literal</i> <i>Target Width</i></p>	<p>RoundingModeRTE</p> <p>Missing before version 1.4.</p> <p>Also see extension: SPV_KHR_float_controls</p>

	Execution Mode	Extra Operands			Enabling Capabilities
4463	RoundingModeRTZ The default rounding mode for floating-point arithmetic and conversions instructions is round toward zero. If an instruction is decorated with FPRoundingMode or defines a rounding mode in its description, that rounding mode is applied and RoundingModeRTZ is ignored. Only affects instructions operating on a floating-point type using the IEEE 754 encoding whose component width is <i>Target Width</i> . <i>Target Width</i> is an unsigned 32-bit integer. May be applied at most once per <i>Target Width</i> to any entry point.	<i>Literal</i> <i>Target Width</i>			RoundingModeRTZ Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4489	NonCoherentTileAttachmentReadQCOM				TileShadingQCOM Reserved.
4490	TileShadingRateQCOM	<i>Literal</i> <i>x rate</i>	<i>Literal</i> <i>y rate</i>	<i>Literal</i> <i>z rate</i>	TileShadingQCOM Reserved.
5017	EarlyAndLateFragmentTestsAMD				Shader Reserved. Also see extension: SPV_AMD_shader_early_and_late_fragment_tests
5027	StencilRefReplacingEXT				StencilExportEXT Reserved. Also see extension: SPV_EXT_shader_stencil_export
5069	CoalescingAMD				ShaderQueueAMD Reserved.
5070	IsApiEntryAMD	<id> <i>Is Entry</i>			ShaderQueueAMD Reserved.
5071	MaxNodeRecursionAMD	<id> <i>Number of recursions</i>			ShaderQueueAMD Reserved.

Execution Mode		Extra Operands			Enabling Capabilities
5072	StaticNumWorkgroupsAMD	<i><id> x size</i>	<i><id> y size</i>	<i><id> z size</i>	ShaderQueueAMD Reserved.
5073	ShaderIndexAMD	<i><id> Shader Index</i>			ShaderQueueAMD Reserved.
5077	MaxNumWorkgroupsAMD	<i><id> x size</i>	<i><id> y size</i>	<i><id> z size</i>	ShaderQueueAMD Reserved.
5079	StencilRefUnchangedFrontAMD				StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export
5080	StencilRefGreaterFrontAMD				StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export
5081	StencilRefLessFrontAMD				StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export
5082	StencilRefUnchangedBackAMD				StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export

Execution Mode		Extra Operands		Enabling Capabilities
5083	StencilRefGreaterBackAMD			StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export
5084	StencilRefLessBackAMD			StencilExportEXT Reserved. Also see extensions: SPV_AMD_shader_early_and_late_fragment_tests , SPV_EXT_shader_stencil_export
5088	QuadDerivativesKHR			QuadControlKHR Reserved.
5089	RequireFullQuadsKHR			QuadControlKHR Reserved.
5102	SharesInputWithAMD	<i><id> Node Name</i>	<i><id> Shader Index</i>	ShaderEnqueueAMD Reserved.
5269	OutputLinesEXT (OutputLinesNV)			MeshShadingNV, MeshShadingEXT Reserved. Also see extensions: SPV_NV_mesh_shader , SPV_EXT_mesh_shader
5270	OutputPrimitivesEXT (OutputPrimitivesNV)	<i>Literal Primitive count</i>		MeshShadingNV, MeshShadingEXT Reserved. Also see extensions: SPV_NV_mesh_shader , SPV_EXT_mesh_shader

Execution Mode		Extra Operands	Enabling Capabilities
5289	DerivativeGroupQuadsKHR (DerivativeGroupQuadsNV)		ComputeDerivativeGroupQuadsNV , ComputeDerivativeGroupQuadsKHR Reserved. Also see extensions: SPV_NV_compute_shader_derivatives , SPV_KHR_compute_shader_derivatives
5290	DerivativeGroupLinearKHR (DerivativeGroupLinearNV)		ComputeDerivativeGroupLinearNV , ComputeDerivativeGroupLinearKHR Reserved. Also see extensions: SPV_NV_compute_shader_derivatives , SPV_KHR_compute_shader_derivatives
5298	OutputTrianglesEXT (OutputTrianglesNV)		MeshShadingNV , MeshShadingEXT Reserved. Also see extensions: SPV_NV_mesh_shader , SPV_EXT_mesh_shader
5366	PixelInterlockOrderedEXT		FragmentShaderPixelInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock
5367	PixelInterlockUnorderedEXT		FragmentShaderPixelInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock

Execution Mode		Extra Operands	Enabling Capabilities
5368	SampleInterlockOrderedEXT		FragmentShaderSampleInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock
5369	SampleInterlockUnorderedEXT		FragmentShaderSampleInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock
5370	ShadingRateInterlockOrderedEXT		FragmentShaderShadingRateInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock
5371	ShadingRateInterlockUnorderedEXT		FragmentShaderShadingRateInterlockEXT Reserved. Also see extension: SPV_EXT_fragment_shader_interlock
5618	SharedLocalMemorySizeINTEL	<i>Literal</i> Size	VectorComputeINTEL Reserved.
5620	RoundingModeRTPINTEL	<i>Literal</i> Target Width	RoundToInfinityINTEL Reserved.
5621	RoundingModeRTNINTEL	<i>Literal</i> Target Width	RoundToInfinityINTEL Reserved.
5622	FloatingPointModeALTINTEL	<i>Literal</i> Target Width	RoundToInfinityINTEL Reserved.
5623	FloatingPointModeIEEEINTEL	<i>Literal</i> Target Width	RoundToInfinityINTEL Reserved.

Execution Mode		Extra Operands			Enabling Capabilities
5893	MaxWorkgroupSizeINTEL	Literal max_x_size	Literal max_y_size	Literal max_z_size	KernelAttributesINTEL Reserved. Also see extension: SPV_INTEL_kernel_attributes
5894	MaxWorkDimINTEL	Literal max_dimensions			KernelAttributesINTEL Reserved. Also see extension: SPV_INTEL_kernel_attributes
5895	NoGlobalOffsetINTEL				KernelAttributesINTEL Reserved. Also see extension: SPV_INTEL_kernel_attributes
5896	NumSIMDWorkitemsINTEL	Literal vector_width			FPGAKernelAttributesINTEL Reserved. Also see extension: SPV_INTEL_kernel_attributes
5903	SchedulerTargetFmaxMhzINTEL	Literal target_fmax			FPGAKernelAttributesINTEL Reserved.
6023	MaximallyReconvergesKHR				Shader Reserved. Also see extension: SPV_KHR_maximal_reconvergence
6028	FPFastMathDefault	<id> Target Type	<id> Fast-Math Mode		FloatControls2 Reserved.
6154	StreamingInterfaceINTEL	Literal StallFreeReturn			FPGAKernelAttributesINTEL Reserved.
6160	RegisterMapInterfaceINTEL	Literal WaitForDoneWrite			FPGAKernelAttributesv2INTEL Reserved.
6417	NamedBarrierCountINTEL	Literal Barrier Count			VectorComputeINTEL Reserved.

Execution Mode		Extra Operands	Enabling Capabilities
6461	MaximumRegistersINTEL	<i>Literal</i> Number of Registers	RegisterLimitsINTEL Reserved.
6462	MaximumRegistersIdINTEL	<id> Number of Registers	RegisterLimitsINTEL Reserved.
6463	NamedMaximumRegistersINTEL	<i>Named Maximum</i> <i>Number of Registers</i> <i>Named Maximum</i> <i>Number of Registers</i>	RegisterLimitsINTEL Reserved.

3.2.6. Storage Class

Class of storage for declared variables. [Intermediate values](#) do not form a storage class, and unless stated otherwise, storage class-based restrictions are not restrictions on intermediate objects and their types.

Used by:

- [OpTypePointer](#)
- [OpTypeForwardPointer](#)
- [OpVariable](#)
- [OpGenericCastToPtrExplicit](#)
- [OpTypeUntypedPointerKHR](#)
- [OpUntypedVariableKHR](#)

Storage Class		Enabling Capabilities
0	UniformConstant Shared externally, visible across all invocations . Graphics uniform memory. OpenCL constant memory. Variables declared with this storage class are read-only. They may have initializers, as allowed by the client API.	
1	Input Input from pipeline. Visible only by the current invocation . Variables declared with this storage class are read-only, and must not have initializers.	
2	Uniform Shared externally, visible across all invocations . Composite objects in this storage class must have a type with an explicit layout .	Shader
3	Output Output to pipeline. Visible only by the current invocation .	Shader
4	Workgroup Visible across all invocations within a workgroup.	

	Storage Class	Enabling Capabilities
5	CrossWorkgroup Visible across all invocations .	
6	Private Visible only by the current invocation .	Shader, VectorComputeINTEL
7	Function Visible only by the current invocation . For memory allocation within a function with specific lifetime. See OpVariable for more information.	
8	Generic For generic pointers, which overload the Function , Workgroup , and CrossWorkgroup Storage Classes .	GenericPointer
9	PushConstant For holding push-constant memory, visible across all invocations . Intended to contain a small bank of values pushed from the client API. Variables declared with this storage class are read-only, and must not have initializers. Composite objects in this storage class must have a type with an explicit layout .	Shader
10	AtomicCounter For holding atomic counters. Visible only by the current invocation .	AtomicStorage
11	Image For holding image memory.	
12	StorageBuffer Shared externally, readable and writable, visible across all invocations . Composite objects in this storage class must have a type with an explicit layout .	Shader Missing before version 1.3. Also see extensions: SPV_KHR_storage_buffer_storage_class , SPV_KHR_variable_pointers
4172	TileImageEXT	TileImageColorReadAccessEXT Reserved.
4491	TileAttachmentQCOM	TileShadingQCOM Reserved.
5068	NodePayloadAMD	ShaderEnqueueAMD Reserved.

	Storage Class	Enabling Capabilities
5328	CallableDataKHR (CallableDataNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5329	IncomingCallableDataKHR (IncomingCallableDataNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5338	RayPayloadKHR (RayPayloadNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5339	HitAttributeKHR (HitAttributeNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5342	IncomingRayPayloadKHR (IncomingRayPayloadNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5343	ShaderRecordBufferKHR (ShaderRecordBufferNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5349	PhysicalStorageBuffer (PhysicalStorageBufferEXT) Shared externally, readable and writable, visible across all invocations . Uses physical addressing. Composite objects in this storage class must have a type with an explicit layout .	PhysicalStorageBufferAddresses Missing before version 1.5. Also see extensions: SPV_EXT_physical_storage_buffer , SPV_KHR_physical_storage_buffer
5385	HitObjectAttributeNV	ShaderInvocationReorderNV Reserved.

Storage Class		Enabling Capabilities
5402	TaskPayloadWorkgroupEXT	MeshShadingEXT Missing before version 1.4. Also see extension: SPV_EXT_mesh_shader
5605	CodeSectionINTEL	FunctionPointersINTEL Reserved. Also see extension: SPV_INTEL_function_pointers
5936	DeviceOnlyINTEL	USMStorageClassesINTEL Reserved. Also see extension: SPV_INTEL_usm_storage_classes
5937	HostOnlyINTEL	USMStorageClassesINTEL Reserved. Also see extension: SPV_INTEL_usm_storage_classes

3.2.7. Dim

Dimensionality of an image. Some uses require capabilities beyond the enabling capabilities, for example where the type's *Sampled* operand is 2, or *Arrayed* operand is 1. See the [capabilities](#) section for more detail.

Used by [OpTypeImage](#).

Dim		Enabling Capabilities
0	1D	Sampled1D
1	2D	
2	3D	
3	Cube	Shader
4	Rect	SampledRect
5	Buffer	SampledBuffer
6	SubpassData	InputAttachment
4173	TileImageDataEXT	TileImageColorReadAccessEXT Reserved.

3.2.8. Sampler Addressing Mode

Addressing mode for creating constant samplers.

Used by [OpConstantSampler](#).

Sampler Addressing Mode		Enabling Capabilities
0	None The image coordinates used to sample elements of the image refer to a location inside the image, otherwise the results are undefined.	
1	ClampToEdge Out-of-range image coordinates are clamped to the extent.	
2	Clamp Out-of-range image coordinates result in a border color.	
3	Repeat Out-of-range image coordinates are wrapped to the valid range. Must only be used with normalized coordinates.	
4	RepeatMirrored Flip the image coordinate at every integer junction. Must only be used with normalized coordinates.	

3.2.9. Sampler Filter Mode

Filter mode for creating constant samplers.

Used by [OpConstantSampler](#).

Sampler Filter Mode		Enabling Capabilities
0	Nearest Use filter nearest mode when performing a read image operation.	
1	Linear Use filter linear mode when performing a read image operation.	

3.2.10. Image Format

Declarative image format.

Used by [OpTypeImage](#).

Image Format		Enabling Capabilities
0	Unknown	
1	Rgba32f	Shader

	Image Format	Enabling Capabilities
2	Rgba16f	Shader
3	R32f	Shader
4	Rgba8	Shader
5	Rgba8Snorm	Shader
6	Rg32f	StorageImageExtendedFormats
7	Rg16f	StorageImageExtendedFormats
8	R11fG11fB10f	StorageImageExtendedFormats
9	R16f	StorageImageExtendedFormats
10	Rgba16	StorageImageExtendedFormats
11	Rgb10A2	StorageImageExtendedFormats
12	Rg16	StorageImageExtendedFormats
13	Rg8	StorageImageExtendedFormats
14	R16	StorageImageExtendedFormats
15	R8	StorageImageExtendedFormats
16	Rgba16Snorm	StorageImageExtendedFormats
17	Rg16Snorm	StorageImageExtendedFormats
18	Rg8Snorm	StorageImageExtendedFormats
19	R16Snorm	StorageImageExtendedFormats
20	R8Snorm	StorageImageExtendedFormats
21	Rgba32i	Shader
22	Rgba16i	Shader
23	Rgba8i	Shader
24	R32i	Shader
25	Rg32i	StorageImageExtendedFormats
26	Rg16i	StorageImageExtendedFormats
27	Rg8i	StorageImageExtendedFormats
28	R16i	StorageImageExtendedFormats
29	R8i	StorageImageExtendedFormats
30	Rgba32ui	Shader
31	Rgba16ui	Shader
32	Rgba8ui	Shader
33	R32ui	Shader

	Image Format	Enabling Capabilities
34	Rgb10a2ui	StorageImageExtendedFormats
35	Rg32ui	StorageImageExtendedFormats
36	Rg16ui	StorageImageExtendedFormats
37	Rg8ui	StorageImageExtendedFormats
38	R16ui	StorageImageExtendedFormats
39	R8ui	StorageImageExtendedFormats
40	R64ui	Int64ImageEXT
41	R64i	Int64ImageEXT

3.2.11. Image Channel Order

The image channel orders that result from [OpImageQueryOrder](#).

	Image Channel Order	Enabling Capabilities
0	R	
1	A	
2	RG	
3	RA	
4	RGB	
5	RGBA	
6	BGRA	
7	ARGB	
8	Intensity	
9	Luminance	
10	Rx	
11	RGx	
12	RGBx	
13	Depth	
14	DepthStencil	
15	sRGB	
16	sRGBx	
17	sRGBA	
18	sBGRA	
19	ABGR	

3.2.12. Image Channel Data Type

Image channel data types that result from [OpImageQueryFormat](#).

Image Channel Data Type		Enabling Capabilities
0	SnormInt8	
1	SnormInt16	
2	UnormInt8	
3	UnormInt16	
4	UnormShort565	
5	UnormShort555	
6	UnormInt101010	
7	SignedInt8	
8	SignedInt16	
9	SignedInt32	
10	UnsignedInt8	
11	UnsignedInt16	
12	UnsignedInt32	
13	HalfFloat	
14	Float	
15	UnormInt24	
16	UnormInt101010_2	
17	UnormInt10X6EXT	
19	UnsignedIntRaw10EXT	
20	UnsignedIntRaw12EXT	
21	UnormInt2_101010EXT	
22	UnsignedInt10X6EXT	
23	UnsignedInt12X4EXT	
24	UnsignedInt14X2EXT	
25	UnormInt12X4EXT	
26	UnormInt14X2EXT	

3.2.13. Image Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Provides additional operands to sampling, or getting texels from, an image. Bits that are set indicate whether an additional operand follows, as described by the table. If there are multiple following operands

indicated, they are ordered: Those indicated by smaller-numbered bits appear first. At least one bit must be set (**None** is invalid).

Used by:

- [OpImageSampleImplicitLod](#)
- [OpImageSampleExplicitLod](#)
- [OpImageSampleDrefImplicitLod](#)
- [OpImageSampleDrefExplicitLod](#)
- [OpImageSampleProjImplicitLod](#)
- [OpImageSampleProjExplicitLod](#)
- [OpImageSampleProjDrefImplicitLod](#)
- [OpImageSampleProjDrefExplicitLod](#)
- [OpImageFetch](#)
- [OpImageGather](#)
- [OpImageDrefGather](#)
- [OpImageRead](#)
- [OpImageWrite](#)
- [OpImageSparseSampleImplicitLod](#)
- [OpImageSparseSampleExplicitLod](#)
- [OpImageSparseSampleDrefImplicitLod](#)
- [OpImageSparseSampleDrefExplicitLod](#)
- [OpImageSparseFetch](#)
- [OpImageSparseGather](#)
- [OpImageSparseDrefGather](#)
- [OpImageSparseRead](#)
- [OpImageSampleFootprintNV](#)

Image Operands		Enabling Capabilities
0x0	None	
0x1	Bias A following operand is the bias added to the implicit level of detail. Only valid with implicit-lod instructions. It must be a 32-bit <i>floating-point type</i> scalar using the IEEE 754 encoding. This must only be used with an OpTypeImage that has a <i>Dim</i> operand of 1D , 2D , 3D , or Cube , and the <i>MS</i> operand must be 0.	Shader

Image Operands		Enabling Capabilities
0x2	<p>Lod</p> <p>A following operand is the explicit level-of-detail to use. Only valid with explicit-lod instructions. For sampling operations, it must be a 32-bit <i>floating-point type</i> scalar using the IEEE 754 encoding. For fetch operations, it must be a 32-bit <i>integer type</i> scalar. This must only be used with an OpTypeImage that has a <i>Dim</i> operand of 1D, 2D, 3D, or Cube, and the <i>MS</i> operand must be 0.</p>	
0x4	<p>Grad</p> <p>Two following operands are <i>dx</i> followed by <i>dy</i>. These are explicit derivatives in the x and y direction to use in computing level of detail. Each is a scalar or vector containing (<i>du/dx</i> [, <i>dv/dx</i> [, <i>dw/dx</i>]) and (<i>du/dy</i> [, <i>dv/dy</i> [, <i>dw/dy</i>]). The number of components of each must equal the number of components in <i>Coordinate</i>, minus the <i>array layer</i> component, if present. Only valid with explicit-lod instructions. They must be a scalar or vector of 32-bits <i>floating-point type</i> using the IEEE 754 encoding. This must only be used with an OpTypeImage that has an <i>MS</i> operand of 0. It is invalid to set both the Lod and Grad bits.</p>	
0x8	<p>ConstOffset</p> <p>A following operand is added to (<i>u</i>, <i>v</i>, <i>w</i>) before texel lookup. It must be an <i><id></i> of a <i>constant instruction</i> with a 32-bit scalar or vector <i>integer type</i>. It is invalid for these to be outside a target-dependent allowed range. The number of components must equal the number of components in <i>Coordinate</i>, minus the <i>array layer</i> component, if present. Not valid with the Cube dimension. An instruction must specify at most one of the ConstOffset, Offset, and ConstOffsets image operands.</p>	
0x10	<p>Offset</p> <p>A following operand is added to (<i>u</i>, <i>v</i>, <i>w</i>) before texel lookup. It must be a 32-bit scalar or vector of <i>integer type</i>. It is invalid for these to be outside a target-dependent allowed range. The number of components must equal the number of components in <i>Coordinate</i>, minus the <i>array layer</i> component, if present. Not valid with the Cube dimension. An instruction must specify at most one of the ConstOffset, Offset, and ConstOffsets image operands.</p>	ImageGatherExtended

Image Operands		Enabling Capabilities
0x20	<p>ConstOffsets</p> <p>A following operand is <i>Offsets</i>. <i>Offsets</i> must be an <i><id></i> of a constant instruction making an array of size four of vectors of two 32-bits integer components. Each gathered texel is identified by adding one of these array elements to the (<i>u</i>, <i>v</i>) sampled location. It is invalid for these to be outside a target-dependent allowed range. Only valid with OpImageGather or OpImageDrefGather. Not valid with the Cube dimension. An instruction must specify at most one of the ConstOffset, Offset, and ConstOffsets image operands.</p>	ImageGatherExtended
0x40	<p>Sample</p> <p>A following operand is the sample number of the sample to use. Only valid with OpImageFetch, OpImageRead, OpImageWrite, OpImageSparseFetch, and OpImageSparseRead. The Sample operand must be used if and only if the underlying OpTypeImage has <i>MS</i> of 1. It must be a 32-bit integer type scalar.</p>	
0x80	<p>MinLod</p> <p>A following operand is the minimum level-of-detail to use when accessing the image. Only valid with Implicit instructions and Grad instructions. It must be a 32-bit floating-point type scalar using the IEEE 754 encoding. This must only be used with an OpTypeImage that has a <i>Dim</i> operand of 1D, 2D, 3D, or Cube, and the <i>MS</i> operand must be 0.</p>	MinLod
0x100	<p>MakeTexelAvailable (MakeTexelAvailableKHR)</p> <p>Perform an availability operation on the texel locations after the store. A following operand is the memory scope that controls the availability operation. Requires NonPrivateTexel to also be set. Only valid with instructions writing images.</p>	<p>VulkanMemoryModel</p> <p>Missing before version 1.5.</p> <p>Also see extension: SPV_KHR_vulkan_memory_model</p>
0x200	<p>MakeTexelVisible (MakeTexelVisibleKHR)</p> <p>Perform a visibility operation on the texel locations before the load. A following operand is the memory scope that controls the visibility operation. Requires NonPrivateTexel to also be set. Only valid with instructions reading images without a sampler.</p>	<p>VulkanMemoryModel</p> <p>Missing before version 1.5.</p> <p>Also see extension: SPV_KHR_vulkan_memory_model</p>

Image Operands		Enabling Capabilities
0x400	NonPrivateTexel (NonPrivateTexelKHR) The image access obeys inter-thread ordering, as specified by the client API.	VulkanMemoryModel Missing before version 1.5 . Also see extension: SPV_KHR_vulkan_memory_model
0x800	VolatileTexel (VolatileTexelKHR) This access cannot be eliminated, duplicated, or combined with other accesses.	VulkanMemoryModel Missing before version 1.5 . Also see extension: SPV_KHR_vulkan_memory_model
0x1000	SignExtend The texel value is converted to the target value via sign extension. Only valid if the texel value type is a scalar or vector of <i>integer type</i> : - for sparse images, the texel value type is the second member of the result type. - for OpImageWrite the texel value type is type of the <i>Texel</i> operand. - otherwise, the texel value type is the result type. It is invalid to set both the ZeroExtend and SignExtend bits.	Missing before version 1.4 .
0x2000	ZeroExtend The texel value is converted to the target value via zero extension. Only valid if the texel value type is a scalar or vector of <i>integer type</i> with signedness of 0: - for sparse images, the texel value type is the second member of the result type. - for OpImageWrite the texel value type is type of the <i>Texel</i> operand. - otherwise, the texel value type is the result type. It is invalid to set both the ZeroExtend and SignExtend bits.	Missing before version 1.4 .
0x4000	Nontemporal Hints that the accessed texels are not likely to be accessed again in the near future.	Missing before version 1.6 .
0x10000	Offsets	

3.2.14. FP Fast Math Mode

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Enables fast math operations which are otherwise unsafe.

Only valid on

- **OpFAdd**, **OpFSub**, **OpFMul**, **OpFDiv**, **OpFRem**, and **OpFMod** instructions
- **Missing before version 1.6:**
 - the **OpFNegate** instruction
 - the **OpOrdered**, **OpUnordered**, **OpFOrdEqual**, **OpFUnordEqual**, **OpFOrdNotEqual**, **OpFUnordNotEqual**, **OpFOrdLessThan**, **OpFUnordLessThan**, **OpFOrdGreaterThan**, **OpFUnordGreaterThan**, **OpFOrdLessThanEqual**, **OpFUnordLessThanEqual**, **OpFOrdGreaterThanEqual**, and **OpFUnordGreaterThanEqual** instructions
 - **OpExtInst** extended instructions, where expressly permitted by the extended instruction set in use.

FP Fast Math Mode		Enabling Capabilities
0x0	None	
0x1	NotNaN Assume parameters and result are not NaN. If this assumption does not hold then the operation returns an undefined value.	
0x2	NotInf Assume parameters and result are not +/- Inf. If this assumption does not hold then the operation returns an undefined value.	
0x4	NSZ Treat the sign of a zero parameter or result as insignificant.	
0x8	AllowRecip Allow the usage of reciprocal rather than perform a division.	
0x10	Fast Allow algebraic transformations according to real-number associative and distributive algebra. This flag implies all the others.	
0x10000	AllowContract (AllowContractFastINTEL)	FloatControls2, FPFastMathModeINTEL Reserved.
0x20000	AllowReassoc (AllowReassocINTEL)	FloatControls2, FPFastMathModeINTEL Reserved.
0x40000	AllowTransform	FloatControls2 Reserved.

3.2.15. FP Rounding Mode

Associate a rounding mode to a floating-point conversion instruction.

FP Rounding Mode		Enabling Capabilities
0	RTE Round to nearest even.	

FP Rounding Mode		Enabling Capabilities
1	RTZ Round towards zero.	
2	RTP Round towards positive infinity.	
3	RTN Round towards negative infinity.	

3.2.16. Linkage Type

Associate a linkage type to functions or global variables. See [linkage](#).

Linkage Type		Enabling Capabilities
0	Export Accessible by other modules as well.	Linkage
1	Import A declaration of a global variable or a function that exists in another module.	Linkage
2	LinkOnceODR	Linkage Reserved . Also see extension: SPV_KHR_linkonce_odr

3.2.17. Access Qualifier

Defines the access permissions.

Used by [OpTypeImage](#), [OpTypePipe](#), and [OpTypeBufferSurfaceINTEL](#).

Access Qualifier		Enabling Capabilities
0	ReadOnly A read-only object.	Kernel
1	WriteOnly A write-only object.	Kernel
2	ReadWrite A readable and writable object.	Kernel

3.2.18. Function Parameter Attribute

Adds additional information to the return type and to each parameter of a function.

Only one of **Zext** and **Sext** can be used to decorate the same *<id>*, and no attribute may be used multiple times on the same *<id>*. Otherwise, multiple function parameter attributes can be applied to the same *<id>*.

Function Parameter Attribute		Enabling Capabilities
0	Zext Zero extend the value, if needed.	Kernel
1	Sext Sign extend the value, if needed.	Kernel
2	ByVal Pass the parameter by value to the function. Only valid for pointer parameters (not for ret value).	Kernel
3	Sret The parameter is the address of a structure that is the return value of the function in the source program. Only applicable to the first parameter, which must be a pointer parameter.	Kernel
4	NoAlias The memory pointed to by a pointer parameter is not accessed via pointer values that are not derived from this pointer parameter. Only valid for pointer parameters. Not valid on return values.	Kernel
5	NoCapture The parameter is not copied into a location that is accessible after returning from the callee. Only valid for pointer parameters. Not valid on return values.	Kernel
6	NoWrite The parameter is not used to write to the memory pointed to. Only valid for pointer parameters. Not valid on return values.	Kernel
7	NoReadWrite The parameter is not dereferenced, either to read or write the memory pointed to. Only valid for pointer parameters. Not valid on return values.	Kernel
5940	RuntimeAlignedINTEL	RuntimeAlignedAttributeINTEL

3.2.19. Decoration

Decorations add additional information to an *<id>* or member of a structure.

It is invalid to decorate any given *<id>* or structure member more than one time with the same [decoration](#), unless explicitly allowed below for a specific decoration.

Used by:

- [OpDecorate](#)
- [OpMemberDecorate](#)
- [OpDecorateId](#)
- [OpDecorateString](#)
- [OpMemberDecorateString](#)

	Decoration	Extra Operands	Enabling Capabilities
0	RelaxedPrecision Allow reduced precision operations. To be used as described in Relaxed Precision .		Shader
1	SpecId Apply only to a scalar specialization constant. <i>Specialization Constant ID</i> is an unsigned 32-bit integer forming the external linkage for setting a specialized value. See specialization .	<i>Literal</i> <i>Specialization Constant ID</i>	Shader, Kernel
2	Block Apply only to a structure type to establish it is a memory interface block.		Shader
3	BufferBlock Deprecated (use Block -decorated StorageBuffer Storage Class objects). Apply only to a structure type to establish it is a memory interface block. When the type is used for a variable in the Uniform Storage Class the memory interface is a StorageBuffer -like interface, distinct from those variables decorated with Block . In all other Storage Classes the decoration is meaningless.		Shader Missing after version 1.3.
4	RowMajor Applies only to a member of a structure type. Only valid on a matrix or array whose most basic element is a matrix. Indicates that components within a row are contiguous in memory. Must not be used with ColMajor on the same matrix or matrix aggregate.		Matrix
5	ColMajor Applies only to a member of a structure type. Only valid on a matrix or array whose most basic element is a matrix. Indicates that components within a column are contiguous in memory. Must not be used with RowMajor on the same matrix or matrix aggregate.		Matrix

	Decoration	Extra Operands	Enabling Capabilities
6	ArrayStride Apply to an array type to specify the stride, in bytes, of the array's elements. Can also apply to a pointer type to an array element. <i>Array Stride</i> is an unsigned 32-bit integer specifying the stride of the array that the element resides in. Must not be applied to any other type.	<i>Literal</i> <i>Array Stride</i>	Shader
7	MatrixStride Applies only to a member of a structure type. Only valid on a matrix or array whose most basic element is a matrix. <i>Matrix Stride</i> is an unsigned 32-bit integer specifying the stride of the rows in a RowMajor -decorated matrix or columns in a ColMajor -decorated matrix.	<i>Literal</i> <i>Matrix Stride</i>	Matrix
8	GLSLShared Apply only to a structure type to get GLSL shared memory layout.		Shader
9	GLSLPacked Apply only to a structure type to get GLSL packed memory layout.		Shader
10	CPacked Apply only to a structure type, to marks it as "packed", indicating that the alignment of the structure is one and that there is no padding between structure members.		Kernel
11	BuiltIn Indicates which built-in variable an object represents. See BuiltIn for more information.	<i>BuiltIn</i>	
13	NoPerspective Must only be used on a memory object declaration or a member of a structure type. Requests linear, non-perspective correct, interpolation. Only valid for the Input and Output Storage Classes .		Shader

	Decoration	Extra Operands	Enabling Capabilities
14	Flat Must only be used on a memory object declaration or a member of a structure type. Indicates no interpolation is done. The non-interpolated value comes from a vertex, as specified by the client API. Only valid for the Input and Output Storage Classes .		Shader
15	Patch Must only be used on a memory object declaration or a member of a structure type. Indicates a tessellation patch. Only valid for the Input and Output Storage Classes . Invalid to use on objects or types referenced by non-tessellation Execution Models .		Tessellation
16	Centroid Must only be used on a memory object declaration or a member of a structure type. If used with multi-sampling rasterization, allows a single interpolation location for an entire pixel. The interpolation location lies in both the pixel and in the primitive being rasterized. Only valid for the Input and Output Storage Classes .		Shader
17	Sample Must only be used on a memory object declaration or a member of a structure type. If used with multi-sampling rasterization, requires per-sample interpolation. The interpolation locations are the locations of the samples lying in both the pixel and in the primitive being rasterized. Only valid for the Input and Output Storage Classes .		SampleRateShading
18	Invariant Apply only to a variable or member of a block-decorated structure type to indicate that expressions computing its value be computed invariantly with respect to other shaders computing the same expressions.		Shader

	Decoration	Extra Operands	Enabling Capabilities
19	Restrict Apply only to a memory object declaration , to indicate the compiler may compile as if there is no aliasing. See the Aliasing section for more detail.		
20	Aliased Apply only to a memory object declaration , to indicate the compiler is to generate accesses to the variable that work correctly in the presence of aliasing. See the Aliasing section for more detail.		
21	Volatile Must be applied only to memory object declarations or members of a structure type. Any such memory object declaration, or any memory object declaration that contains such a structure type, must be one of: <ul style="list-style-type: none"> - An image with <i>Sampled</i> Operand of 2 and <i>Dim</i> other than <i>SubpassData</i> (see OpTypeImage). - A block in the StorageBuffer storage class, or in the Uniform storage class with the BufferBlock decoration. This indicates the memory holding the variable is volatile memory. Accesses to volatile memory cannot be eliminated, duplicated, or combined with other accesses. Volatile applies only to a single invocation and does not guarantee each invocation performs the access. Volatile is not allowed if the declared memory model is Vulkan . The memory operand bit Volatile , the image operand bit VolatileTexel , or the memory semantic bit Volatile can be used instead.		
22	Constant Indicates that a global variable is constant and never modified. Only allowed on global variables.		Kernel

	Decoration	Extra Operands	Enabling Capabilities
23	<p>Coherent</p> <p>Must be applied only to memory object declarations or members of a structure type. Any such memory object declaration, or any memory object declaration that contains such a structure type, must be one of:</p> <ul style="list-style-type: none"> - An image with <i>Sampled</i> Operand of 2 and <i>Dim</i> other than <i>SubpassData</i> (see OpTypeImage). - A block in the StorageBuffer storage class, or in the Uniform storage class with the BufferBlock decoration. <p>This indicates the memory backing the object is coherent. Coherent is not allowed if the declared memory model is Vulkan. The memory operand bits MakePointerAvailable and MakePointerVisible or the image operand bits MakeTexelAvailable and MakeTexelVisible can be used instead.</p>		
24	<p>NonWritable</p> <p>Must be applied only to memory object declarations or members of a structure type. Any such memory object declaration, or any memory object declaration that contains such a structure type, must be one of:</p> <ul style="list-style-type: none"> - An image with <i>Sampled</i> Operand of 2 and <i>Dim</i> other than <i>SubpassData</i> (see OpTypeImage). - A block in the StorageBuffer storage class, or in the Uniform storage class with the BufferBlock decoration. - Missing before version 1.4: An object in the Private or Function storage classes. <p>This indicates that this module does not write to the memory holding the variable. It does not prevent the use of initializers on a declaration.</p>		

	Decoration	Extra Operands	Enabling Capabilities
25	<p>NonReadable</p> <p>Must be applied only to memory object declarations or members of a structure type. Any such memory object declaration, or any memory object declaration that contains such a structure type, must be one of:</p> <ul style="list-style-type: none"> - An image with <i>Sampled</i> Operand of 2 and <i>Dim</i> other than <i>SubpassData</i> (see OpTypeImage). - A block in the StorageBuffer storage class, or in the Uniform storage class with the BufferBlock decoration. <p>This indicates that this module does not read from the memory holding the variable. For image variables, it does not prevent query operations from reading metadata associated with the image.</p>		
26	<p>Uniform</p> <p>Apply only to an object. Asserts that, for each dynamic instance of the instruction that computes the result, all invocations in the same tangle within the invocation's Subgroup scope compute the same result value.</p>		Shader, UniformDecoration
27	<p>UniformId</p> <p>Apply only to an object. Asserts that, for each dynamic instance of the instruction that computes the result, all invocations in the same tangle within the invocation's <i>Execution</i> scope compute the same result value. <i>Execution</i> must not be Invocation.</p>	<p>Scope <id> <i>Execution</i></p>	<p>Shader, UniformDecoration</p> <p>Missing before version 1.4.</p>

	Decoration	Extra Operands	Enabling Capabilities
28	<p>SaturatedConversion Indicates that a conversion to an integer type which is outside the representable range of <i>Result Type</i> is clamped to the nearest representable value of <i>Result Type</i>. <i>NaN</i> is converted to 0.</p> <p>This decoration must be applied only to conversion instructions to integer types, not including the OpSatConvertUToS and OpSatConvertSToU instructions.</p>		Kernel
29	<p>Stream Must only be used on a memory object declaration or a member of a structure type. <i>Stream Number</i> is an unsigned 32-bit integer indicating the stream number to put an output on. Only valid for the Output Storage Class and the Geometry Execution Model.</p>	<i>Literal Stream Number</i>	GeometryStreams
30	<p>Location Apply only to a variable or a structure-type member. <i>Location</i> is an unsigned 32-bit integer that forms the main linkage for Storage Class Input and Output variables:</p> <ul style="list-style-type: none"> - between the client API and vertex-stage inputs, - between consecutive programmable stages, or - between fragment-stage outputs and the client API. <p>It can also tag variables or structure-type members in the UniformConstant Storage Class for linkage with the client API. Only valid for the Input, Output, and UniformConstant Storage Classes.</p>	<i>Literal Location</i>	Shader

	Decoration	Extra Operands	Enabling Capabilities
31	Component Must only be used on a memory object declaration or a member of a structure type. <i>Component</i> is an unsigned 32-bit integer indicating which component within a Location is taken by the decorated entity. Only valid for the Input and Output Storage Classes .	Literal <i>Component</i>	Shader
32	Index Apply only to a variable. <i>Index</i> is an unsigned 32-bit integer identifying a blend equation input index, used as specified by the client API. Only valid for the Output Storage Class and the Fragment Execution Model .	Literal <i>Index</i>	Shader
33	Binding Apply only to a variable. <i>Binding Point</i> is an unsigned 32-bit integer forming part of the linkage between the client API and SPIR-V memory buffers, images, etc. See the client API specification for more detail.	Literal <i>Binding Point</i>	Shader
34	DescriptorSet Apply only to a variable. <i>Descriptor Set</i> is an unsigned 32-bit integer forming part of the linkage between the client API and SPIR-V memory buffers, images, etc. See the client API specification for more detail.	Literal <i>Descriptor Set</i>	Shader
35	Offset Apply only to a structure-type member. <i>Byte Offset</i> is an unsigned 32-bit integer. It dictates the byte offset of the member relative to the beginning of the structure. It can be used, for example, by both uniform and transform-feedback buffers. It must not cause any overlap of the structure's members, or overflow of a transform-feedback buffer's XfbStride .	Literal <i>Byte Offset</i>	Shader

	Decoration	Extra Operands		Enabling Capabilities
36	XfbBuffer Must only be used on a memory object declaration or a member of a structure type. <i>XFB Buffer</i> is an unsigned 32-bit integer indicating which transform-feedback buffer an output is written to. Only valid for the Output Storage Classes of vertex processing Execution Models .	Literal <i>XFB Buffer Number</i>		TransformFeedback
37	XfbStride Apply to anything XfbBuffer is applied to. <i>XFB Stride</i> is an unsigned 32-bit integer specifying the stride, in bytes, of transform-feedback buffer vertices. If the transform-feedback buffer is capturing any double-precision components, the stride must be a multiple of 8, otherwise it must be a multiple of 4.	Literal <i>XFB Stride</i>		TransformFeedback
38	FuncParamAttr Indicates a function return value or parameter attribute. Multiple uses of this decoration are allowed on the same <i><id></i> , as described in the function parameter attributes .	Function Parameter Attribute <i>Function Parameter Attribute</i>		Kernel
39	FPRoundingMode Indicates a floating-point rounding mode.	FP Rounding Mode <i>Floating-Point Rounding Mode</i>		
40	FPFastMathMode Indicates a floating-point fast math flag.	FP Fast Math Mode <i>Fast-Math Mode</i>		Kernel, FloatControls2
41	LinkageAttributes Associate linkage attributes to values. <i>Name</i> is a string specifying what name the <i>Linkage Type</i> applies to. Only valid on OpFunction or global (module scope) OpVariable . See linkage .	Literal <i>Name</i>	Linkage Type <i>Linkage Type</i>	Linkage

	Decoration	Extra Operands	Enabling Capabilities
42	NoContraction Apply only to an arithmetic instruction to indicate the operation cannot be combined with another instruction to form a single operation. For example, if applied to an OpFMul , that multiply can't be combined with an addition to yield a fused multiply-add operation. Furthermore, such operations are not allowed to reassociate; e.g., $\text{add}(a + \text{add}(b+c))$ cannot be transformed to $\text{add}(\text{add}(a+b) + c)$.		Shader
43	InputAttachmentIndex Apply only to a variable. <i>Attachment Index</i> is an unsigned 32-bit integer providing an input-target index (as specified by the client API). Only valid in the Fragment Execution Model and for variables of type OpTypeImage with a <i>Dim</i> operand of SubpassData .	Literal <i>Attachment Index</i>	InputAttachment
44	Alignment Apply only to a pointer. <i>Alignment</i> is an unsigned 32-bit integer declaring a known minimum alignment the pointer has.	Literal <i>Alignment</i>	Kernel
45	MaxByteOffset Apply only to a pointer. <i>Max Byte Offset</i> is an unsigned 32-bit integer declaring a known maximum byte offset this pointer will be incremented by from the point of the decoration. This is a guaranteed upper bound when applied to OpFunctionParameter .	Literal <i>Max Byte Offset</i>	Addresses Missing before version 1.1.
46	AlignmentId Same as the Alignment decoration , but using an <i><id></i> operand instead of a literal. The operand is consumed as unsigned and must be an <i>integer type</i> scalar.	<i><id></i> <i>Alignment</i>	Kernel Missing before version 1.2.

	Decoration	Extra Operands	Enabling Capabilities
47	MaxByteOffsetId Same as the MaxByteOffset decoration , but using an <i><id></i> operand instead of a literal. The operand is consumed as unsigned and must be an <i>integer type</i> scalar.	<i><id></i> <i>Max Byte Offset</i>	Addresses Missing before version 1.2 .
4216	SaturatedToLargestFloat8NormalConversionEXT		Float8EXT Reserved.
4469	NoSignedWrap Apply to an instruction to indicate that it does not cause signed integer wrapping to occur, in the form of overflow or underflow. It must decorate only the following instructions: - OpIAdd - OpISub - OpIMul - OpShiftLeftLogical - OpSNegate - OpExtInst for instruction numbers specified in the extended instruction-set specifications as accepting this decoration. If an instruction decorated with NoSignedWrap does overflow or underflow, behavior is undefined.		Missing before version 1.4 . Also see extension: SPV_KHR_no_integer_wrap_decoration

	Decoration	Extra Operands	Enabling Capabilities
4470	NoUnsignedWrap Apply to an instruction to indicate that it does not cause unsigned integer wrapping to occur, in the form of overflow or underflow. It must decorate only the following instructions: - OpIAdd - OpISub - OpIMul - OpShiftLeftLogical - OpExtInst for instruction numbers specified in the extended instruction-set specifications as accepting this decoration. If an instruction decorated with NoUnsignedWrap does overflow or underflow, behavior is undefined.		Missing before version 1.4. Also see extension: SPV_KHR_no_integer_wrap_decoration
4487	WeightTextureQCOM		Reserved. Also see extension: SPV_QCOM_image_processing
4488	BlockMatchTextureQCOM		Reserved. Also see extension: SPV_QCOM_image_processing
4499	BlockMatchSamplerQCOM		Reserved. Also see extension: SPV_QCOM_image_processing2
4999	ExplicitInterpAMD		Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
5019	NodeSharesPayloadLimitsWithAMD	<i><id></i> <i>Payload Type</i>	ShaderEnqueueAMD Reserved.
5020	NodeMaxPayloadsAMD	<i><id></i> <i>Max number of payloads</i>	ShaderEnqueueAMD Reserved.
5078	TrackFinishWritingAMD		ShaderEnqueueAMD Reserved.

	Decoration	Extra Operands	Enabling Capabilities
5091	PayloadNodeNameAMD X	<i><id></i> <i>Node Name</i>	ShaderEnqueueAMD X Reserved.
5098	PayloadNodeBaseIndexAMD X	<i><id></i> <i>Base Index</i>	ShaderEnqueueAMD X Reserved.
5099	PayloadNodeSparseArrayAMD X		ShaderEnqueueAMD X Reserved.
5100	PayloadNodeArraySizeAMD X	<i><id></i> <i>Array Size</i>	ShaderEnqueueAMD X Reserved.
5105	PayloadDispatchIndirectAMD X		ShaderEnqueueAMD X Reserved.
5248	OverrideCoverageNV		SampleMaskOverrideCoverageNV Reserved. Also see extension: SPV_NV_sample_mask_override_coverage
5250	PassthroughNV		GeometryShaderPassthroughNV Reserved. Also see extension: SPV_NV_geometry_shader_passthrough
5252	ViewportRelativeNV		ShaderViewportMaskNV Reserved.
5256	SecondaryViewportRelativeNV	<i>Literal</i> <i>Offset</i>	ShaderStereoViewNV Reserved. Also see extension: SPV_NV_stereo_view_rendering
5271	PerPrimitiveEXT (PerPrimitiveNV)		MeshShadingNV, MeshShadingEXT Reserved. Also see extensions: SPV_NV_mesh_shader , SPV_EXT_mesh_shader

	Decoration	Extra Operands	Enabling Capabilities
5272	PerViewNV		MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5273	PerTaskNV		MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5285	PerVertexKHR (PerVertexNV)		FragmentBarycentricKHR Reserved. Also see extensions: SPV_NV_fragment_shader_barycentric , SPV_KHR_fragment_shader_barycentric
5300	NonUniform (NonUniformEXT) Apply only to an object. Asserts that the value backing the decorated <i><id></i> is not dynamically uniform . See the client API specification for more detail.		ShaderNonUniform Missing before version 1.5 . Also see extension: SPV_EXT_descriptor_indexing
5355	RestrictPointer (RestrictPointerEXT) Apply only to a memory object declaration , to indicate the compiler may compile as if there is no aliasing of the pointer stored in the variable. See the aliasing section for more detail.		PhysicalStorageBufferAddresses Missing before version 1.5 . Also see extensions: SPV_EXT_physical_storage_buffer , SPV_KHR_physical_storage_buffer
5356	AliasedPointer (AliasedPointerEXT) Apply only to a memory object declaration , to indicate the compiler is to generate accesses to the pointer stored in the variable that work correctly in the presence of aliasing. See the aliasing section for more detail.		PhysicalStorageBufferAddresses Missing before version 1.5 . Also see extensions: SPV_EXT_physical_storage_buffer , SPV_KHR_physical_storage_buffer
5386	HitObjectShaderRecordBufferNV		ShaderInvocationReorderNV Reserved.

	Decoration	Extra Operands	Enabling Capabilities
5398	BindlessSamplerNV		BindlessTextureNV Reserved.
5399	BindlessImageNV		BindlessTextureNV Reserved.
5400	BoundSamplerNV		BindlessTextureNV Reserved.
5401	BoundImageNV		BindlessTextureNV Reserved.
5599	SIMTCallINTEL	<i>Literal N</i>	VectorComputeINTEL Reserved.
5602	ReferencedIndirectlyINTEL		IndirectReferencesINTEL Reserved. Also see extension: SPV_INTEL_function_pointers
5607	ClobberINTEL	<i>Literal Register</i>	AsmINTEL Reserved.
5608	SideEffectsINTEL		AsmINTEL Reserved.
5624	VectorComputeVariableINTEL		VectorComputeINTEL Reserved.
5625	FuncParamIOKindINTEL	<i>Literal Kind</i>	VectorComputeINTEL Reserved.
5626	VectorComputeFunctionINTEL		VectorComputeINTEL Reserved.
5627	StackCallINTEL		VectorComputeINTEL Reserved.
5628	GlobalVariableOffsetINTEL	<i>Literal Offset</i>	VectorComputeINTEL Reserved.

	Decoration	Extra Operands		Enabling Capabilities
5634	CounterBuffer (HlslCounterBufferGOOGLE) The <i><id></i> of a counter buffer associated with the decorated buffer. It must decorate only a variable in the Uniform storage class. <i>Counter Buffer</i> must be a variable in the Uniform storage class.	<i><id></i> <i>Counter Buffer</i>		Missing before version 1.4. Also see extension: SPV_GOOGLE_hlsl_functionality1
5635	UserSemantic (HlslSemanticGOOGLE) <i>Semantic</i> is a string describing a user-defined semantic intent of what it decorates. User-defined semantics are case insensitive. It must decorate only a variable or a member of a structure type. If decorating a variable, the variable must be in the Input or Output storage classes. If decorating a structure member, memory object declarations that contain such structure type can be in any storage class. A variable or a structure member can be decorated more than one time with this decoration, but at most once for any particular string operand.	<i>Literal</i> <i>Semantic</i>		Missing before version 1.4. Also see extension: SPV_GOOGLE_hlsl_functionality1
5636	UserTypeGOOGLE	<i>Literal</i> <i>User Type</i>		Reserved. Also see extension: SPV_GOOGLE_user_type
5822	FunctionRoundingModeINTEL	<i>Literal</i> <i>Target</i> <i>Width</i>	<i>FP Rounding</i> <i>Mode</i> <i>FP Rounding</i> <i>Mode</i>	FunctionFloatControlINTEL Reserved.
5823	FunctionDenormModeINTEL	<i>Literal</i> <i>Target</i> <i>Width</i>	<i>FP Denorm</i> <i>Mode</i> <i>FP Denorm</i> <i>Mode</i>	FunctionFloatControlINTEL Reserved.

	Decoration	Extra Operands	Enabling Capabilities
5825	RegisterINTEL		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5826	MemoryINTEL	<i>Literal Memory Type</i>	FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5827	NumbanksINTEL	<i>Literal Banks</i>	FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5828	BankwidthINTEL	<i>Literal Bank Width</i>	FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5829	MaxPrivateCopiesINTEL	<i>Literal Maximum Copies</i>	FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5830	SinglepumpINTEL		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5831	DoublepumpINTEL		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes

Decoration		Extra Operands		Enabling Capabilities
5832	MaxReplicatesINTEL	<i>Literal</i> Maximum Replicates		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5833	SimpleDualPortINTEL			FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5834	MergeINTEL	<i>Literal</i> Merge Key	<i>Literal</i> Merge Type	FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5835	BankBitsINTEL	<i>Literal</i> Bank Bits		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5836	ForcePow2DepthINTEL	<i>Literal</i> Force Key		FPGAMemoryAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes
5883	StridesizeINTEL	<i>Literal</i> Stride Size		FPGAMemoryAttributesINTEL Reserved.
5884	WordsizeINTEL	<i>Literal</i> Word Size		FPGAMemoryAttributesINTEL Reserved.
5885	TrueDualPortINTEL			FPGAMemoryAttributesINTEL Reserved.
5899	BurstCoalesceINTEL			FPGAMemoryAccessesINTEL Reserved.

Decoration		Extra Operands		Enabling Capabilities
5900	CacheSizeINTEL	<i>Literal</i> Cache Size in bytes		FPGAMemoryAccessesINTEL Reserved.
5901	DontStaticallyCoalesceINTEL			FPGAMemoryAccessesINTEL Reserved.
5902	PrefetchINTEL	<i>Literal</i> Prefetcher Size in bytes		FPGAMemoryAccessesINTEL Reserved.
5905	StallEnableINTEL			FPGAClusterAttributesINTEL Reserved.
5907	FuseLoopsInFunctionINTEL			LoopFuseINTEL Reserved.
5909	MathOpDSPModeINTEL	<i>Literal</i> Mode	<i>Literal</i> Propagate	FPGADSPControlINTEL Reserved.
5914	AliasScopeINTEL	<id> Aliasing Scopes List		MemoryAccessAliasingINTEL Reserved.
5915	NoAliasINTEL	<id> Aliasing Scopes List		MemoryAccessAliasingINTEL Reserved.
5917	InitiationIntervalINTEL	<i>Literal</i> Cycles		FPGAInvocationPipeliningAttributesINTEL Reserved.
5918	MaxConcurrencyINTEL	<i>Literal</i> Invocations		FPGAInvocationPipeliningAttributesINTEL Reserved.
5919	PipelineEnableINTEL	<i>Literal</i> Enable		FPGAInvocationPipeliningAttributesINTEL Reserved.
5921	BufferLocationINTEL	<i>Literal</i> Buffer Location ID		FPGABufferLocationINTEL Reserved.
5944	IOPipeStorageINTEL	<i>Literal</i> IO Pipe ID		IOPipesINTEL Reserved.

Decoration		Extra Operands			Enabling Capabilities
6080	FunctionFloatingPointModelINTEL	Literal Target Width	FP Operation Mode FP Operation Mode		FunctionFloatControlINTEL Reserved.
6085	SingleElementVectorINTEL				VectorComputeINTEL Reserved.
6087	VectorComputeCallableFunctionINTEL				VectorComputeINTEL Reserved.
6140	MediaBlockIOINTEL				VectorComputeINTEL Reserved.
6151	StallFreeINTEL				FPGAClusterAttributesV2INTEL Reserved.
6170	FPMaxErrorDecorationINTEL	Literal Max Error			FPMaxErrorINTEL Reserved.
6172	LatencyControlLabelINTEL	Literal Latency Label			FPGALatencyControlINTEL Reserved.
6173	LatencyControlConstraintINTEL	Literal Relative To	Literal Control Type	Literal Relative Cycle	FPGALatencyControlINTEL Reserved.
6175	ConduitKernelArgumentINTEL				FPGAArgumentInterfacesINTEL Reserved.
6176	RegisterMapKernelArgumentINTEL				FPGAArgumentInterfacesINTEL Reserved.
6177	MMHostInterfaceAddressWidthINTEL	Literal AddressWidth			FPGAArgumentInterfacesINTEL Reserved.
6178	MMHostInterfaceDataWidthINTEL	Literal DataWidth			FPGAArgumentInterfacesINTEL Reserved.
6179	MMHostInterfaceLatencyINTEL	Literal Latency			FPGAArgumentInterfacesINTEL Reserved.
6180	MMHostInterfaceReadWriteModeINTEL	Access Qualifier ReadWriteMode			FPGAArgumentInterfacesINTEL Reserved.

Decoration		Extra Operands		Enabling Capabilities
6181	MMHostInterfaceMaxBurstINTEL	<i>Literal</i> <i>MaxBurstCount</i>		FPGAArgumentInterfacesINTEL <i>Reserved.</i>
6182	MMHostInterfaceWaitRequestINTEL	<i>Literal</i> <i>Waitrequest</i>		FPGAArgumentInterfacesINTEL <i>Reserved.</i>
6183	StableKernelArgumentINTEL			FPGAArgumentInterfacesINTEL <i>Reserved.</i>
6188	HostAccessINTEL	<i>Host Access Qualifier Access</i>	<i>Literal Name</i>	GlobalVariableHostAccessINTEL <i>Reserved.</i>
6190	InitModeINTEL	<i>Initialization Mode Qualifier Trigger</i>		GlobalVariableFPGADecorationsINTEL <i>Reserved.</i>
6191	ImplementInRegisterMapINTEL	<i>Literal Value</i>		GlobalVariableFPGADecorationsINTEL <i>Reserved.</i>
6247	ConditionalINTEL	<id> <i>Condition</i>		SpecConditionalINTEL <i>Reserved.</i>
6442	CacheControlLoadINTEL	<i>Literal Cache Level</i>	<i>Load Cache Control Cache Control</i>	CacheControlsINTEL <i>Reserved.</i>
6443	CacheControlStoreINTEL	<i>Literal Cache Level</i>	<i>Store Cache Control Cache Control</i>	CacheControlsINTEL <i>Reserved.</i>

3.2.20. BuiltIn

Used when **Decoration** is **BuiltIn**. Apply to:

- The result <id> of the **OpVariable** declaration of the built-in variable,
- A structure-type member, if the built-in is a member of a structure, or
- **Deprecated**: a [constant instruction](#), when the built-in is a constant.

As stated per entry below, these have additional semantics and constraints specified by the client API.

For all the declarations of all the global variables and constants statically referenced by the entry-point's call tree, within any specific storage class it is invalid to decorate with a specific **BuiltIn** more than once.

Application to a [constant instruction](#) has previously been used to define the workgroup size with specialization constants in some client APIs. As of version 1.6, all client APIs should instead use the

LocalSizeId [execution mode](#).

BuiltIn		Enabling Capabilities
0	Position Output vertex position from a vertex processing Execution Model . See the client API specification for more detail.	Shader
1	PointSize Output point size from a vertex processing Execution Model . See the client API specification for more detail.	Shader
3	ClipDistance Array of clip distances. See the client API specification for more detail.	ClipDistance
4	CullDistance Array of clip distances. See the client API specification for more detail.	CullDistance
5	VertexId Input vertex ID to a Vertex Execution Model . See the client API specification for more detail.	Shader
6	InstanceId Input instance ID to a Vertex Execution Model . See the client API specification for more detail.	Shader
7	PrimitiveId Primitive ID in a Geometry Execution Model . See the client API specification for more detail.	Geometry, Tessellation, RayTracingNV, RayTracingKHR, MeshShadingNV, MeshShadingEXT
8	InvocationId Invocation ID, input to Geometry and TessellationControl Execution Model . See the client API specification for more detail.	Geometry, Tessellation
9	Layer Layer selection for multi-layer framebuffer. See the client API specification for more detail. The Geometry capability allows for a Layer output by a Geometry Execution Model , input to a Fragment Execution Model . The ShaderLayer capability allows for Layer output by a Vertex or Tessellation Execution Model .	Geometry, ShaderLayer, ShaderViewportIndexLayerEXT, MeshShadingNV, MeshShadingEXT

	BuiltIn	Enabling Capabilities
10	<p>ViewportIndex Viewport selection for viewport transformation when using multiple viewports. See the client API specification for more detail.</p> <p>The MultiViewport capability allows for a ViewportIndex output by a Geometry Execution Model, input to a Fragment Execution Model.</p> <p>The ShaderViewportIndex capability allows for a ViewportIndex output by a Vertex or Tessellation Execution Model.</p>	MultiViewport, ShaderViewportIndex, ShaderViewportIndexLayerEXT, MeshShadingNV, MeshShadingEXT
11	<p>TessLevelOuter Output patch outer levels in a TessellationControl Execution Model. See the client API specification for more detail.</p>	Tessellation
12	<p>TessLevelInner Output patch inner levels in a TessellationControl Execution Model. See the client API specification for more detail.</p>	Tessellation
13	<p>TessCoord Input vertex position in TessellationEvaluation Execution Model. See the client API specification for more detail.</p>	Tessellation
14	<p>PatchVertices Input patch vertex count in a tessellation Execution Model. See the client API specification for more detail.</p>	Tessellation
15	<p>FragCoord Coordinates ($x, y, z, 1/w$) of the current fragment, input to the Fragment Execution Model. See the client API specification for more detail.</p>	Shader
16	<p>PointCoord Coordinates within a <i>point</i>, input to the Fragment Execution Model. See the client API specification for more detail.</p>	Shader
17	<p>FrontFacing Face direction, input to the Fragment Execution Model. See the client API specification for more detail.</p>	Shader
18	<p>SampleId Input sample number to the Fragment Execution Model. See the client API specification for more detail.</p>	SampleRateShading

	BuiltIn	Enabling Capabilities
19	SamplePosition Input sample position to the Fragment Execution Model . See the client API specification for more detail.	SampleRateShading
20	SampleMask Input or output sample mask to the Fragment Execution Model . See the client API specification for more detail.	Shader
22	FragDepth Output fragment depth from the Fragment Execution Model . See the client API specification for more detail.	Shader
23	HelperInvocation Input whether a helper invocation, to the Fragment Execution Model . See the client API specification for more detail.	Shader
24	NumWorkgroups Number of workgroups in GLCompute or Kernel Execution Models . See the client API specification for more detail.	
25	WorkgroupSize Workgroup size in GLCompute or Kernel Execution Models . See the client API specification for more detail.	
26	WorkgroupId Workgroup ID in GLCompute or Kernel Execution Models . See the client API specification for more detail.	
27	LocalInvocationId Local invocation ID in GLCompute or Kernel Execution Models . See the client API specification for more detail.	
28	GlobalInvocationId Global invocation ID in GLCompute or Kernel Execution Models . See the client API specification for more detail.	
29	LocalInvocationIndex Local invocation index in GLCompute Execution Models . See the client API specification for more detail. Workgroup Linear ID in Kernel Execution Models . See the client API specification for more detail.	
30	WorkDim Work dimensions in Kernel Execution Models . See the client API specification for more detail.	Kernel

	BuiltIn	Enabling Capabilities
31	GlobalSize Global size in Kernel Execution Models . See the client API specification for more detail.	Kernel
32	EnqueuedWorkgroupSize Enqueued workgroup size in Kernel Execution Models . See the client API specification for more detail.	Kernel
33	GlobalOffset Global offset in Kernel Execution Models . See the client API specification for more detail.	Kernel
34	GlobalLinearId Global linear ID in Kernel Execution Models . See the client API specification for more detail.	Kernel
36	SubgroupSize Subgroup size. See the client API specification for more detail.	Kernel, GroupNonUniform, SubgroupBallotKHR
37	SubgroupMaxSize Subgroup maximum size in Kernel Execution Models . See the client API specification for more detail.	Kernel
38	NumSubgroups Number of subgroups in GLCompute or Kernel Execution Models . See the client API specification for more detail.	Kernel, GroupNonUniform
39	NumEnqueuedSubgroups Number of enqueued subgroups in Kernel Execution Models . See the client API specification for more detail.	Kernel
40	SubgroupId Subgroup ID in GLCompute or Kernel Execution Models . See the client API specification for more detail.	Kernel, GroupNonUniform
41	SubgroupLocalInvocationId Subgroup local invocation ID. See the client API specification for more detail.	Kernel, GroupNonUniform, SubgroupBallotKHR
42	VertexIndex Vertex index. See the client API specification for more detail.	Shader
43	InstanceIndex Instance index. See the client API specification for more detail.	Shader
4160	CoreIDARM	CoreBuiltinsARM
4161	CoreCountARM	CoreBuiltinsARM
4162	CoreMaxIDARM	CoreBuiltinsARM

	BuiltIn	Enabling Capabilities
4163	WarpIDARM	CoreBuiltinsARM
4164	WarpMaxIDARM	CoreBuiltinsARM
4416	SubgroupEqMask (SubgroupEqMaskKHR) Subgroup invocations bitmask where bit index = SubgroupLocalInvocationId . See the client API specification for more detail.	SubgroupBallotKHR, GroupNonUniformBallot Missing before version 1.3. Also see extension: SPV_KHR_shader_ballot
4417	SubgroupGeMask (SubgroupGeMaskKHR) Subgroup invocations bitmask where bit index >= SubgroupLocalInvocationId . See the client API specification for more detail.	SubgroupBallotKHR, GroupNonUniformBallot Missing before version 1.3. Also see extension: SPV_KHR_shader_ballot
4418	SubgroupGtMask (SubgroupGtMaskKHR) Subgroup invocations bitmask where bit index > SubgroupLocalInvocationId . See the client API specification for more detail.	SubgroupBallotKHR, GroupNonUniformBallot Missing before version 1.3. Also see extension: SPV_KHR_shader_ballot
4419	SubgroupLeMask (SubgroupLeMaskKHR) Subgroup invocations bitmask where bit index <= SubgroupLocalInvocationId . See the client API specification for more detail.	SubgroupBallotKHR, GroupNonUniformBallot Missing before version 1.3. Also see extension: SPV_KHR_shader_ballot
4420	SubgroupLtMask (SubgroupLtMaskKHR) Subgroup invocations bitmask where bit index < SubgroupLocalInvocationId . See the client API specification for more detail.	SubgroupBallotKHR, GroupNonUniformBallot Missing before version 1.3. Also see extension: SPV_KHR_shader_ballot
4424	BaseVertex Base vertex component of vertex ID. See the client API specification for more detail.	DrawParameters Missing before version 1.3. Also see extension: SPV_KHR_shader_draw_parameters
4425	BaseInstance Base instance component of instance ID. See the client API specification for more detail.	DrawParameters Missing before version 1.3. Also see extension: SPV_KHR_shader_draw_parameters

	BuiltIn	Enabling Capabilities
4426	DrawIndex Contains the index of the draw currently being processed. See the client API specification for more detail.	DrawParameters, MeshShadingNV, MeshShadingEXT Missing before version 1.3. Also see extensions: SPV_KHR_shader_draw_parameters , SPV_NV_mesh_shader , SPV_EXT_mesh_shader
4432	PrimitiveShadingRateKHR	FragmentShadingRateKHR Reserved. Also see extension: SPV_KHR_fragment_shading_rate
4438	DeviceIndex Input device index of the logical device. See the client API specification for more detail.	DeviceGroup Missing before version 1.3. Also see extension: SPV_KHR_device_group
4440	ViewIndex Input view index of the view currently being rendered to. See the client API specification for more detail.	MultiView Missing before version 1.3. Also see extension: SPV_KHR_multiview
4444	ShadingRateKHR	FragmentShadingRateKHR Reserved. Also see extension: SPV_KHR_fragment_shading_rate
4492	TileOffsetQCOM	TileShadingQCOM Reserved.
4493	TileDimensionQCOM	TileShadingQCOM Reserved.
4494	TileApronSizeQCOM	TileShadingQCOM Reserved.
4992	BaryCoordNoPerspAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter

	BuiltIn	Enabling Capabilities
4993	BaryCoordNoPerspCentroidAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
4994	BaryCoordNoPerspSampleAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
4995	BaryCoordSmoothAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
4996	BaryCoordSmoothCentroidAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
4997	BaryCoordSmoothSampleAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
4998	BaryCoordPullModelAMD	Reserved. Also see extension: SPV_AMD_shader_explicit_vertex_parameter
5014	FragStencilRefEXT	StencilExportEXT Reserved. Also see extension: SPV_EXT_shader_stencil_export
5021	RemainingRecursionLevelsAMD	ShaderEnqueueAMD Reserved.
5073	ShaderIndexAMD	ShaderEnqueueAMD Reserved.

	BuiltIn	Enabling Capabilities
5253	ViewportMaskNV	ShaderViewportMaskNV, MeshShadingNV Reserved. Also see extensions: SPV_NV_viewport_array2 , SPV_NV_mesh_shader
5257	SecondaryPositionNV	ShaderStereoViewNV Reserved. Also see extension: SPV_NV_stereo_view_rendering
5258	SecondaryViewportMaskNV	ShaderStereoViewNV Reserved. Also see extension: SPV_NV_stereo_view_rendering
5261	PositionPerViewNV	PerViewAttributesNV, MeshShadingNV Reserved. Also see extensions: SPV_NVX_multiview_per_view_attributes , SPV_NV_mesh_shader
5262	ViewportMaskPerViewNV	PerViewAttributesNV, MeshShadingNV Reserved. Also see extensions: SPV_NVX_multiview_per_view_attributes , SPV_NV_mesh_shader
5264	FullyCoveredEXT	FragmentFullyCoveredEXT Reserved. Also see extension: SPV_EXT_fragment_fully_covered
5274	TaskCountNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader

	BuiltIn	Enabling Capabilities
5275	PrimitiveCountNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5276	PrimitiveIndicesNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5277	ClipDistancePerViewNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5278	CullDistancePerViewNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5279	LayerPerViewNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5280	MeshViewCountNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5281	MeshViewIndicesNV	MeshShadingNV Reserved. Also see extension: SPV_NV_mesh_shader
5286	BaryCoordKHR (BaryCoordNV)	FragmentBarycentricKHR Reserved. Also see extensions: SPV_NV_fragment_shader_barycentric , SPV_KHR_fragment_shader_barycentric

	BuiltIn	Enabling Capabilities
5287	BaryCoordNoPerspKHR (BaryCoordNoPerspNV)	FragmentBarycentricKHR Reserved. Also see extensions: SPV_NV_fragment_shader_barycentric , SPV_KHR_fragment_shader_barycentric
5292	FragSizeEXT (FragmentSizeNV)	FragmentDensityEXT Reserved. Also see extensions: SPV_EXT_fragment_invocation_density , SPV_NV_shading_rate
5293	FragInvocationCountEXT (InvocationsPerPixelNV)	FragmentDensityEXT Reserved. Also see extensions: SPV_EXT_fragment_invocation_density , SPV_NV_shading_rate
5294	PrimitivePointIndicesEXT	MeshShadingEXT Reserved. Also see extension: SPV_EXT_mesh_shader
5295	PrimitiveLineIndicesEXT	MeshShadingEXT Reserved. Also see extension: SPV_EXT_mesh_shader
5296	PrimitiveTriangleIndicesEXT	MeshShadingEXT Reserved. Also see extension: SPV_EXT_mesh_shader
5299	CullPrimitiveEXT	MeshShadingEXT Reserved. Also see extension: SPV_EXT_mesh_shader
5319	LaunchIdKHR (LaunchIdNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing

	BuiltIn	Enabling Capabilities
5320	LaunchSizeKHR (LaunchSizeNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5321	WorldRayOriginKHR (WorldRayOriginNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5322	WorldRayDirectionKHR (WorldRayDirectionNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5323	ObjectRayOriginKHR (ObjectRayOriginNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5324	ObjectRayDirectionKHR (ObjectRayDirectionNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5325	RayTminKHR (RayTminNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5326	RayTmaxKHR (RayTmaxNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5327	InstanceCustomIndexKHR (InstanceCustomIndexNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing

BuiltIn		Enabling Capabilities
5330	ObjectToWorldKHR (ObjectToWorldNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5331	WorldToObjectKHR (WorldToObjectNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5332	HitTNV	RayTracingNV Reserved. Also see extension: SPV_NV_ray_tracing
5333	HitKindKHR (HitKindNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing
5334	CurrentRayTimeNV	RayTracingMotionBlurNV Reserved. Also see extension: SPV_NV_ray_tracing_motion_blur
5335	HitTriangleVertexPositionsKHR	RayTracingPositionFetchKHR Reserved.
5337	HitMicroTriangleVertexPositionsNV	RayTracingDisplacementMicromapNV Reserved.
5344	HitMicroTriangleVertexBarycentricsNV	RayTracingDisplacementMicromapNV Reserved.
5351	IncomingRayFlagsKHR (IncomingRayFlagsNV)	RayTracingNV, RayTracingKHR Reserved. Also see extensions: SPV_NV_ray_tracing , SPV_KHR_ray_tracing

	BuiltIn	Enabling Capabilities
5352	RayGeometryIndexKHR	RayTracingKHR Reserved. Also see extension: SPV_KHR_ray_tracing
5359	HitIsSphereNV	RayTracingSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5360	HitIsLSSNV	RayTracingLinearSweptSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5361	HitSpherePositionNV	RayTracingSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5374	WarpsPerSMNV	ShaderSMBuiltinsNV Reserved. Also see extension: SPV_NV_shader_sm_builtins
5375	SMCountNV	ShaderSMBuiltinsNV Reserved. Also see extension: SPV_NV_shader_sm_builtins
5376	WarpIDNV	ShaderSMBuiltinsNV Reserved. Also see extension: SPV_NV_shader_sm_builtins
5377	SMIDNV	ShaderSMBuiltinsNV Reserved. Also see extension: SPV_NV_shader_sm_builtins

BuiltIn		Enabling Capabilities
5396	HitLSSPositionsNV	RayTracingLinearSweptSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5405	HitKindFrontFacingMicroTriangleNV	RayTracingDisplacementMicromapNV Reserved.
5406	HitKindBackFacingMicroTriangleNV	RayTracingDisplacementMicromapNV Reserved.
5420	HitSphereRadiusNV	RayTracingSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5421	HitLSSRadiiNV	RayTracingLinearSweptSpheresGeometryNV Reserved. Also see extension: SPV_NV_linear_swept_spheres
5436	ClusterIDNV	RayTracingClusterAccelerationStructureNV Reserved. Also see extension: SPV_NV_cluster_acceleration_structure
6021	CullMaskKHR	RayCullMaskKHR Reserved. Also see extension: SPV_KHR_ray_cull_mask

3.2.21. Selection Control

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpSelectionMerge](#).

Selection Control		Enabling Capabilities
0x0	None	

Selection Control		Enabling Capabilities
0x1	Flatten Performance hint. Strong request to optimize away the control flow for this selection.	
0x2	DontFlatten Performance hint. Strong request to keep this selection as control flow.	

3.2.22. Loop Control

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Bits that are set indicate whether an additional operand follows, as described by the table. If there are multiple following operands indicated, they are ordered: Those indicated by smaller-numbered bits appear first.

Used by **OpLoopMerge**.

Loop Control		Enabling Capabilities
0x0	None	
0x1	Unroll Performance hint. Strong request to unroll or unwind this loop. This must not be used with the DontUnroll bit.	
0x2	DontUnroll Performance hint. Strong request to keep this loop as a loop, without unrolling.	
0x4	DependencyInfinite Guarantees that there are no dependencies between loop iterations.	Missing before version 1.1.
0x8	DependencyLength Guarantees that there are no dependencies between a number of loop iterations. The dependency length is specified in a subsequent unsigned 32-bit integer literal operand.	Missing before version 1.1.
0x10	MinIterations Unchecked assertion that the loop executes at least a given number of iterations. The iteration count is specified in a subsequent unsigned 32-bit integer literal operand.	Missing before version 1.4.
0x20	MaxIterations Unchecked assertion that the loop executes at most a given number of iterations. The iteration count is specified in a subsequent unsigned 32-bit integer literal operand.	Missing before version 1.4.

Loop Control		Enabling Capabilities
0x40	IterationMultiple Unchecked assertion that the loop executes a multiple of a given number of iterations. The number is specified in a subsequent unsigned 32-bit integer literal operand. It must be greater than 0.	Missing before version 1.4.
0x80	PeelCount Performance hint. Request that the loop be peeled by a given number of loop iterations. The peel count is specified in a subsequent unsigned 32-bit integer literal operand. This must not be used with the DontUnroll bit.	Missing before version 1.4.
0x100	PartialCount Performance hint. Request that the loop be partially unrolled by a given number of loop iterations. The unroll count is specified in a subsequent unsigned 32-bit integer literal operand. This must not be used with the DontUnroll bit.	Missing before version 1.4.
0x10000	InitiationIntervalINTEL	FPGALoopControlsINTEL Reserved.
0x20000	MaxConcurrencyINTEL	FPGALoopControlsINTEL Reserved.
0x40000	DependencyArrayINTEL	FPGALoopControlsINTEL Reserved.
0x80000	PipelineEnableINTEL	FPGALoopControlsINTEL Reserved.
0x100000	LoopCoalesceINTEL	FPGALoopControlsINTEL Reserved.
0x200000	MaxInterleavingINTEL	FPGALoopControlsINTEL Reserved.
0x400000	SpeculatedIterationsINTEL	FPGALoopControlsINTEL Reserved.
0x800000	NoFusionINTEL	FPGALoopControlsINTEL Reserved.
0x1000000	LoopCountINTEL	FPGALoopControlsINTEL Reserved.

Loop Control		Enabling Capabilities
0x2000000	MaxReinvocationDelayINTEL	FPGALoopControlsINTEL Reserved.

3.2.23. Function Control

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by **OpFunction**.

Function Control		Enabling Capabilities
0x0	None	
0x1	Inline Performance hint. Strong request to inline the function.	
0x2	DontInline Performance hint. Strong request to not inline the function.	
0x4	Pure Compiler can assume this function has no side effect, but might read global memory or read through dereferenced function parameters. Always computes the same result when called with the same argument values and the same global state.	
0x8	Const Compiler assumes this function has no side effects, and does not access global memory or dereference function parameters. Always computes the same result for the same argument values.	
0x10000	OptNoneEXT (OptNoneINTEL)	OptNoneEXT Reserved.

3.2.24. Memory Semantics <id>

The <id>'s value is a mask; it can be formed by combining the bits from multiple rows in the table below.

The value's type must be a 32-bit integer scalar. This value is expected to be formed only from the bits in the table below, where at most one of these four bits can be set: **Acquire**, **Release**, **AcquireRelease**, or **SequentiallyConsistent**. If validation rules or the client API require a constant <id>, it is invalid for the value to not be formed this expected way. If non-constant <id> are allowed, behavior is undefined when the value is not formed this expected way.

Requesting both **Acquire** and **Release** semantics is done by setting the **AcquireRelease** bit, not by setting two bits.

Memory semantics define memory-order constraints, and on what storage classes those constraints apply to. The memory order constrains the allowed orders in which memory operations in this [invocation](#) are made visible to another invocation. The storage classes specify to which subsets of memory these constraints are to be applied. Storage classes not selected are not being constrained.

Used by:

- [OpControlBarrier](#)
- [OpMemoryBarrier](#)
- [OpAtomicLoad](#)
- [OpAtomicStore](#)
- [OpAtomicExchange](#)
- [OpAtomicCompareExchange](#)
- [OpAtomicCompareExchangeWeak](#)
- [OpAtomicIIncrement](#)
- [OpAtomicIDecrement](#)
- [OpAtomicIAdd](#)
- [OpAtomicISub](#)
- [OpAtomicSMin](#)
- [OpAtomicUMin](#)
- [OpAtomicSMax](#)
- [OpAtomicUMax](#)
- [OpAtomicAnd](#)
- [OpAtomicOr](#)
- [OpAtomicXor](#)
- [OpAtomicFlagTestAndSet](#)
- [OpAtomicFlagClear](#)
- [OpMemoryNamedBarrier](#)
- [OpAtomicFMinEXT](#)
- [OpAtomicFMaxEXT](#)
- [OpAtomicFAddEXT](#)
- [OpControlBarrierArriveINTEL](#)
- [OpControlBarrierWaitINTEL](#)

Memory Semantics		Enabling Capabilities
0x0	None (Relaxed)	

Memory Semantics		Enabling Capabilities
0x2	Acquire On an atomic instruction, orders memory operations provided in program order after this atomic instruction against this atomic instruction. On a barrier, orders memory operations provided in program order after this barrier against atomic instructions before this barrier. See the client API specification for more detail.	
0x4	Release On an atomic instruction, orders memory operations provided in program order before this atomic instruction against this atomic instruction. On a barrier, orders memory operations provided in program order before this barrier against atomic instructions after this barrier. See the client API specification for more detail.	
0x8	AcquireRelease Has the properties of both Acquire and Release semantics. It is used for read-modify-write operations.	
0x10	SequentiallyConsistent All observers see this memory access in the same order with respect to other sequentially-consistent memory accesses from this invocation. If the declared memory model is Vulkan , SequentiallyConsistent must not be used.	
0x40	UniformMemory Apply the memory-ordering constraints to StorageBuffer , PhysicalStorageBuffer , or Uniform Storage Class memory.	Shader
0x80	SubgroupMemory Apply the memory-ordering constraints to subgroup memory.	
0x100	WorkgroupMemory Apply the memory-ordering constraints to Workgroup Storage Class memory.	
0x200	CrossWorkgroupMemory Apply the memory-ordering constraints to CrossWorkgroup Storage Class memory.	
0x400	AtomicCounterMemory Apply the memory-ordering constraints to AtomicCounter Storage Class memory.	AtomicStorage

Memory Semantics		Enabling Capabilities
0x800	ImageMemory Apply the memory-ordering constraints to image contents (types declared by OpTypeImage), or to accesses done through pointers to the Image Storage Class .	
0x1000	OutputMemory (OutputMemoryKHR) Apply the memory-ordering constraints to Output storage class memory.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
0x2000	MakeAvailable (MakeAvailableKHR) Perform an availability operation on all references in the selected storage classes .	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
0x4000	MakeVisible (MakeVisibleKHR) Perform a visibility operation on all references in the selected storage classes .	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
0x8000	Volatile This access cannot be eliminated, duplicated, or combined with other accesses.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model

3.2.25. Memory Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Provides additional operands to the listed memory instructions. Bits that are set indicate whether an additional operand follows, as described by the table. If there are multiple following operands indicated, they are ordered: Those indicated by smaller-numbered bits appear first. An instruction needing two masks must first provide the first mask followed by the first mask's additional operands, and then provide the second mask followed by the second mask's additional operands.

Used by:

- [OpLoad](#)
- [OpStore](#)
- [OpCopyMemory](#)
- [OpCopyMemorySized](#)
- [OpCooperativeMatrixLoadKHR](#)
- [OpCooperativeMatrixStoreKHR](#)

- [OpCooperativeVectorLoadNV](#)
- [OpCooperativeVectorStoreNV](#)
- [OpCooperativeMatrixLoadNV](#)
- [OpCooperativeMatrixStoreNV](#)
- [OpCooperativeMatrixLoadTensorNV](#)
- [OpCooperativeMatrixStoreTensorNV](#)
- [OpSubgroupBlockPrefetchINTEL](#)

Memory Operands		Enabling Capabilities
0x0	None	
0x1	Volatile This access cannot be eliminated, duplicated, or combined with other accesses.	
0x2	Aligned This access has a known alignment. The alignment is specified in a subsequent unsigned 32-bit integer literal operand. The value must be a power of two. Valid values are defined by the execution environment.	
0x4	Nontemporal Hints that the accessed address is not likely to be accessed again in the near future.	
0x8	MakePointerAvailable (MakePointerAvailableKHR) Perform an availability operation on the locations pointed to by the pointer operand, after a store. A following operand is the memory scope for the availability operation. Requires NonPrivatePointer to also be set. Only valid with instructions writing memory.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
0x10	MakePointerVisible (MakePointerVisibleKHR) Perform a visibility operation on the locations pointed to by the pointer operand, before a load. A following operand is the memory scope for the visibility operation. Requires NonPrivatePointer to also be set. Only valid with instructions reading memory.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
0x20	NonPrivatePointer (NonPrivatePointerKHR) The memory access obeys inter-thread ordering, as specified by the client API.	VulkanMemoryModel Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model

Memory Operands		Enabling Capabilities
0x10000	AliasScopeINTELMask	MemoryAccessAliasingINTEL Reserved. Also see extension: SPV_INTEL_memory_access_aliasing
0x20000	NoAliasINTELMask	MemoryAccessAliasingINTEL Reserved. Also see extension: SPV_INTEL_memory_access_aliasing

3.2.26. Scope <id>

Must be an <id> of a 32-bit integer scalar. Its value is expected to be one of the values in the table below. If validation rules or the client API require a constant <id>, it is invalid for it to not be one of these values. If non-constant <id> are allowed, behavior is undefined if <id> is not one of these values.

If labeled as a memory scope, it specifies the distance of synchronization from the current [invocation](#). If labeled as an execution scope, it specifies the set of executing invocations taking part in the operation. Other usages (neither memory nor execution) of scope are possible, and each such usage defines what scope means in its context.

Used by:

- [OpControlBarrier](#)
- [OpMemoryBarrier](#)
- [OpAtomicLoad](#)
- [OpAtomicStore](#)
- [OpAtomicExchange](#)
- [OpAtomicCompareExchange](#)
- [OpAtomicCompareExchangeWeak](#)
- [OpAtomicIIncrement](#)
- [OpAtomicIDecrement](#)
- [OpAtomicIAdd](#)
- [OpAtomicISub](#)
- [OpAtomicSMin](#)
- [OpAtomicUMin](#)
- [OpAtomicSMax](#)
- [OpAtomicUMax](#)
- [OpAtomicAnd](#)
- [OpAtomicOr](#)
- [OpAtomicXor](#)

- [OpGroupAsyncCopy](#)
- [OpGroupWaitEvents](#)
- [OpGroupAll](#)
- [OpGroupAny](#)
- [OpGroupBroadcast](#)
- [OpGroupIAdd](#)
- [OpGroupFAdd](#)
- [OpGroupFMin](#)
- [OpGroupUMin](#)
- [OpGroupSMin](#)
- [OpGroupFMax](#)
- [OpGroupUMax](#)
- [OpGroupSMax](#)
- [OpGroupReserveReadPipePackets](#)
- [OpGroupReserveWritePipePackets](#)
- [OpGroupCommitReadPipe](#)
- [OpGroupCommitWritePipe](#)
- [OpAtomicFlagTestAndSet](#)
- [OpAtomicFlagClear](#)
- [OpMemoryNamedBarrier](#)
- [OpGroupNonUniformElect](#)
- [OpGroupNonUniformAll](#)
- [OpGroupNonUniformAny](#)
- [OpGroupNonUniformAllEqual](#)
- [OpGroupNonUniformBroadcast](#)
- [OpGroupNonUniformBroadcastFirst](#)
- [OpGroupNonUniformBallot](#)
- [OpGroupNonUniformInverseBallot](#)
- [OpGroupNonUniformBallotBitExtract](#)
- [OpGroupNonUniformBallotBitCount](#)
- [OpGroupNonUniformBallotFindLSB](#)
- [OpGroupNonUniformBallotFindMSB](#)
- [OpGroupNonUniformShuffle](#)
- [OpGroupNonUniformShuffleXor](#)
- [OpGroupNonUniformShuffleUp](#)
- [OpGroupNonUniformShuffleDown](#)
- [OpGroupNonUniformIAdd](#)
- [OpGroupNonUniformFAdd](#)

- `OpGroupNonUniformIMul`
- `OpGroupNonUniformFMul`
- `OpGroupNonUniformSMin`
- `OpGroupNonUniformUMin`
- `OpGroupNonUniformFMin`
- `OpGroupNonUniformSMax`
- `OpGroupNonUniformUMax`
- `OpGroupNonUniformFMax`
- `OpGroupNonUniformBitwiseAnd`
- `OpGroupNonUniformBitwiseOr`
- `OpGroupNonUniformBitwiseXor`
- `OpGroupNonUniformLogicalAnd`
- `OpGroupNonUniformLogicalOr`
- `OpGroupNonUniformLogicalXor`
- `OpGroupNonUniformQuadBroadcast`
- `OpGroupNonUniformQuadSwap`
- `OpGroupNonUniformRotateKHR`
- `OpTypeCooperativeMatrixKHR`
- `OpGroupIAddNonUniformAMD`
- `OpGroupFAddNonUniformAMD`
- `OpGroupFMinNonUniformAMD`
- `OpGroupUMinNonUniformAMD`
- `OpGroupSMinNonUniformAMD`
- `OpGroupFMaxNonUniformAMD`
- `OpGroupUMaxNonUniformAMD`
- `OpGroupSMaxNonUniformAMD`
- `OpReadClockKHR`
- `OpAllocateNodePayloadsAMDX`
- `OpTypeCooperativeMatrixNV`
- `OpAtomicFMinEXT`
- `OpAtomicFMaxEXT`
- `OpAtomicFAddEXT`
- `OpControlBarrierArriveINTEL`
- `OpControlBarrierWaitINTEL`
- `OpGroupIMulKHR`
- `OpGroupFMulKHR`
- `OpGroupBitwiseAndKHR`
- `OpGroupBitwiseOrKHR`

- [OpGroupBitwiseXorKHR](#)
- [OpGroupLogicalAndKHR](#)
- [OpGroupLogicalOrKHR](#)
- [OpGroupLogicalXorKHR](#)

Scope		Enabling Capabilities
0	CrossDevice Scope crosses multiple devices.	
1	Device Scope is the current device.	
2	Workgroup Scope is the current workgroup .	
3	Subgroup Scope is the current subgroup .	
4	Invocation Scope is the current Invocation .	
5	QueueFamily (QueueFamilyKHR) Scope is the current queue family.	VulkanMemoryModel Missing before version 1.5.
6	ShaderCallKHR	RayTracingKHR Reserved.

3.2.27. Group Operation

Defines the class of operation for [group](#) and [non-uniform group](#) instructions.

Used by:

- [OpGroupIAdd](#)
- [OpGroupFAdd](#)
- [OpGroupFMin](#)
- [OpGroupUMin](#)
- [OpGroupSMin](#)
- [OpGroupFMax](#)
- [OpGroupUMax](#)
- [OpGroupSMax](#)
- [OpGroupNonUniformBallotBitCount](#)
- [OpGroupNonUniformIAdd](#)
- [OpGroupNonUniformFAdd](#)
- [OpGroupNonUniformIMul](#)
- [OpGroupNonUniformFMul](#)
- [OpGroupNonUniformSMin](#)

- [OpGroupNonUniformUMin](#)
- [OpGroupNonUniformFMin](#)
- [OpGroupNonUniformSMax](#)
- [OpGroupNonUniformUMax](#)
- [OpGroupNonUniformFMax](#)
- [OpGroupNonUniformBitwiseAnd](#)
- [OpGroupNonUniformBitwiseOr](#)
- [OpGroupNonUniformBitwiseXor](#)
- [OpGroupNonUniformLogicalAnd](#)
- [OpGroupNonUniformLogicalOr](#)
- [OpGroupNonUniformLogicalXor](#)
- [OpGroupIAddNonUniformAMD](#)
- [OpGroupFAddNonUniformAMD](#)
- [OpGroupFMinNonUniformAMD](#)
- [OpGroupUMinNonUniformAMD](#)
- [OpGroupSMinNonUniformAMD](#)
- [OpGroupFMaxNonUniformAMD](#)
- [OpGroupUMaxNonUniformAMD](#)
- [OpGroupSMaxNonUniformAMD](#)
- [OpGroupIMulKHR](#)
- [OpGroupFMulKHR](#)
- [OpGroupBitwiseAndKHR](#)
- [OpGroupBitwiseOrKHR](#)
- [OpGroupBitwiseXorKHR](#)
- [OpGroupLogicalAndKHR](#)
- [OpGroupLogicalOrKHR](#)
- [OpGroupLogicalXorKHR](#)

Group Operation		Enabling Capabilities
0	Reduce A reduction operation for all values of a specific value X specified by invocations within a workgroup.	Kernel, GroupNonUniformArithmetic, GroupNonUniformBallot
1	InclusiveScan A binary operation with an identity I and n (where n is the size of the workgroup) elements $[a_0, a_1, \dots a_{n-1}]$ resulting in $[a_0, (a_0 \text{ op } a_1), \dots (a_0 \text{ op } a_1 \text{ op } \dots \text{ op } a_{n-1})]$	Kernel, GroupNonUniformArithmetic, GroupNonUniformBallot

Group Operation		Enabling Capabilities
2	ExclusiveScan A binary operation with an identity I and n (where n is the size of the workgroup) elements $[a_0, a_1, \dots, a_{n-1}]$ resulting in $[I, a_0, (a_0 \text{ op } a_1), \dots, (a_0 \text{ op } a_1 \text{ op } \dots \text{ op } a_{n-2})]$.	Kernel, GroupNonUniformArithmetic, GroupNonUniformBallot
3	ClusteredReduce	GroupNonUniformClustered Missing before version 1.3.
6	PartitionedReduceNV	GroupNonUniformPartitionedNV Reserved. Also see extension: SPV_NV_shader_subgroup_partitioned
7	PartitionedInclusiveScanNV	GroupNonUniformPartitionedNV Reserved. Also see extension: SPV_NV_shader_subgroup_partitioned
8	PartitionedExclusiveScanNV	GroupNonUniformPartitionedNV Reserved. Also see extension: SPV_NV_shader_subgroup_partitioned

3.2.28. Kernel Enqueue Flags

Specify when the child kernel begins execution.

Note: Implementations are not required to honor this flag. Implementations may not schedule kernel launch earlier than the point specified by this flag, however. Used by [OpEnqueueKernel](#).

Kernel Enqueue Flags		Enabling Capabilities
0	NoWait Indicates that the enqueued kernels do not need to wait for the parent kernel to finish execution before they begin execution.	Kernel

Kernel Enqueue Flags		Enabling Capabilities
1	WaitKernel Indicates that all invocations of the parent kernel finish executing and all immediate side effects are committed before the enqueued child kernel begins execution. Note: Immediate meaning not side effects resulting from child kernels. The side effects would include stores to global memory and pipe reads and writes.	Kernel
2	WaitWorkGroup Indicates that the enqueued kernels wait only for the workgroup that enqueued the kernels to finish before they begin execution. Note: This acts as a memory synchronization point between invocations in a workgroup and child kernels enqueued by invocations in the workgroup.	Kernel

3.2.29. Kernel Profiling Info

The *<id>*'s value is a mask; it can be formed by combining the bits from multiple rows in the table below.

Specifies the profiling information to be queried. Used by [OpCaptureEventProfilingInfo](#).

Kernel Profiling Info		Enabling Capabilities
0x0	None	
0x1	CmdExecTime Indicates that the profiling info queried is the execution time.	Kernel

3.2.30. Capability

Capabilities a module can declare it uses.

All used capabilities need to be declared, either explicitly with [OpCapability](#) or implicitly through the **Implicitly Declares** column: If a capability defined with [statically expressed rules](#) is used, it is invalid to not declare it. If a capability defined in terms of dynamic behavior is used, behavior is undefined unless the capability is declared. The **Implicitly Declares** column lists additional capabilities that are all implicitly declared when the **Capability** entry is explicitly or implicitly declared. It is not necessary, but allowed, to explicitly declare an implicitly declared capability.

See the [capabilities](#) section for more detail.

Used by [OpCapability](#), [OpConditionalCapabilityINTEL](#), and [OpSpecConstantCapabilitiesINTEL](#).

	Capability	Implicitly Declares
0	Matrix Uses OpTypeMatrix .	
1	Shader Uses Vertex , Fragment , or GLCompute Execution Models .	Matrix
2	Geometry Uses the Geometry Execution Model .	Shader
3	Tessellation Uses the TessellationControl or TessellationEvaluation Execution Models .	Shader
4	Addresses Uses physical addressing, non-logical addressing modes.	
5	Linkage Uses partially linked modules and libraries.	
6	Kernel Uses the Kernel Execution Model .	
7	Vector16 Uses OpTypeVector to declare 8 component or 16 component vectors.	Kernel
8	Float16Buffer Allows a 16-bit OpTypeFloat instruction using the IEEE 754 encoding for creating an OpTypePointer to a 16-bit float. Pointers to a 16-bit float must not be dereferenced, unless specifically allowed by a specific instruction. All other uses of 16-bit OpTypeFloat are disallowed.	Kernel
9	Float16 Uses OpTypeFloat to declare the 16-bit floating-point type using the IEEE 754 encoding.	
10	Float64 Uses OpTypeFloat to declare the 64-bit floating-point type using the IEEE 754 encoding.	
11	Int64 Uses OpTypeInt to declare 64-bit integer types.	
12	Int64Atomics Uses atomic instructions on 64-bit integer types.	Int64
13	ImageBasic Uses OpTypeImage or OpTypeSampler in a Kernel .	Kernel
14	ImageReadWrite Uses OpTypeImage with the ReadWrite access qualifier in a kernel.	ImageBasic

	Capability	Implicitly Declares
15	ImageMipmap Uses non-zero Lod Image Operands in a kernel.	ImageBasic
17	Pipes Uses OpTypePipe , OpTypeReserveld or <i>pipe</i> instructions.	Kernel
18	Groups Uses common group instructions.	Also see extension: SPV_AMD_shader_ballot
19	DeviceEnqueue Uses OpTypeQueue , OpTypeDeviceEvent , and <i>device side enqueue</i> instructions.	Kernel
20	LiteralSampler Samplers are made from literals within the module. See OpConstantSampler .	Kernel
21	AtomicStorage Uses the AtomicCounter Storage Class , allowing use of only the OpAtomicLoad , OpAtomicIncrement , and OpAtomicDecrement instructions.	Shader
22	Int16 Uses OpTypeInt to declare 16-bit integer types.	
23	TessellationPointSize Tessellation stage exports point size.	Tessellation
24	GeometryPointSize Geometry stage exports point size	Geometry
25	ImageGatherExtended Uses texture gather with non-constant or independent offsets	Shader
27	StorageImageMultisample An <i>MS</i> operand in OpTypeImage indicates multisampled, used with an OpTypeImage having <i>Sampled == 2</i> .	Shader
28	UniformBufferArrayDynamicIndexing Block -decorated arrays in uniform storage classes use dynamically uniform indexing.	Shader
29	SampledImageArrayDynamicIndexing Arrays of sampled images, samplers, or images with <i>Sampled = 0</i> or <i>1</i> use dynamically uniform indexing.	Shader
30	StorageBufferArrayDynamicIndexing Arrays in the StorageBuffer Storage Class , or BufferBlock -decorated arrays, use dynamically uniform indexing.	Shader

	Capability	Implicitly Declares
31	StorageImageArrayDynamicIndexing Arrays of images with <i>Sampled</i> = 2 are accessed with dynamically uniform indexing.	Shader
32	ClipDistance Uses the ClipDistance BuiltIn .	Shader
33	CullDistance Uses the CullDistance BuiltIn .	Shader
34	ImageCubeArray Uses the Cube Dim with the <i>Arrayed</i> operand in OpTypeImage , with an OpTypeImage having <i>Sampled</i> == 2.	SampledCubeArray
35	SampleRateShading Uses per-sample rate shading.	Shader
36	ImageRect Uses the Rect Dim with an OpTypeImage having <i>Sampled</i> == 2.	SampledRect
37	SampledRect Uses the Rect Dim with an OpTypeImage having <i>Sampled</i> == 0 or 1.	Shader
38	GenericPointer Uses the Generic Storage Class .	Addresses
39	Int8 Uses OpTypeInt to declare 8-bit integer types.	
40	InputAttachment Uses the SubpassData Dim .	Shader
41	SparseResidency Uses OpImageSparse... instructions.	Shader
42	MinLod Uses the MinLod Image Operand .	Shader
43	Sampled1D Uses the 1D Dim with an OpTypeImage having <i>Sampled</i> == 0 or 1.	
44	Image1D Uses the 1D Dim with an OpTypeImage having <i>Sampled</i> == 2.	Sampled1D
45	SampledCubeArray Uses the Cube Dim with the <i>Arrayed</i> operand in OpTypeImage , with an OpTypeImage having <i>Sampled</i> == 0 or 1.	Shader
46	SampledBuffer Uses the Buffer Dim with an OpTypeImage having <i>Sampled</i> == 0 or 1.	

	Capability	Implicitly Declares
47	ImageBuffer Uses the Buffer Dim with an OpTypeImage having <i>Sampled == 2</i> .	SampledBuffer
48	ImageMSArray An <i>MS</i> operand in OpTypeImage indicates multisampled, used with an OpTypeImage having <i>Sampled == 2</i> and <i>Arrayed == 1</i> .	Shader
49	StorageImageExtendedFormats One of a large set of more advanced image formats are used, namely one of those in the Image Format table listed as requiring this capability.	Shader
50	ImageQuery The sizes, number of samples, or lod, etc. are queried.	Shader
51	DerivativeControl Uses fine or coarse-grained derivatives, e.g., OpDPdxFine .	Shader
52	InterpolationFunction Uses one of the InterpolateAtCentroid , InterpolateAtSample , or InterpolateAtOffset GLSL.std.450 extended instructions.	Shader
53	TransformFeedback Uses the Xfb Execution Mode .	Shader
54	GeometryStreams Uses multiple numbered streams for geometry-stage output.	Geometry
55	StorageImageReadWithoutFormat OpImageRead can use the Unknown Image Format .	Shader
56	StorageImageWriteWithoutFormat OpImageWrite can use the Unknown Image Format .	Shader
57	MultiViewport Multiple viewports are used.	Geometry
58	SubgroupDispatch Uses subgroup dispatch instructions.	DeviceEnqueue Missing before version 1.1.
59	NamedBarrier Uses OpTypeNamedBarrier .	Kernel Missing before version 1.1.
60	PipeStorage Uses OpTypePipeStorage .	Pipes Missing before version 1.1.

	Capability	Implicitly Declares
61	GroupNonUniform	Missing before version 1.3.
62	GroupNonUniformVote	GroupNonUniform Missing before version 1.3.
63	GroupNonUniformArithmetic	GroupNonUniform Missing before version 1.3.
64	GroupNonUniformBallot	GroupNonUniform Missing before version 1.3.
65	GroupNonUniformShuffle	GroupNonUniform Missing before version 1.3.
66	GroupNonUniformShuffleRelative	GroupNonUniform Missing before version 1.3.
67	GroupNonUniformClustered	GroupNonUniform Missing before version 1.3.
68	GroupNonUniformQuad	GroupNonUniform Missing before version 1.3.
69	ShaderLayer	Missing before version 1.5.
70	ShaderViewportIndex	Missing before version 1.5.
71	UniformDecoration Uses the Uniform or UniformId decoration	Missing before version 1.6.
4165	CoreBuiltinsARM	Reserved. Also see extension: SPV_ARM_core_builtins
4166	TileImageColorReadAccessEXT	Reserved. Also see extension: SPV_EXT_shader_tile_image
4167	TileImageDepthReadAccessEXT	Reserved. Also see extension: SPV_EXT_shader_tile_image
4168	TileImageStencilReadAccessEXT	Reserved. Also see extension: SPV_EXT_shader_tile_image

	Capability	Implicitly Declares
4174	TensorsARM	Reserved. Also see extension: SPV_ARM_tensors
4175	StorageTensorArrayDynamicIndexingARM	Reserved. Also see extension: SPV_ARM_tensors
4176	StorageTensorArrayNonUniformIndexingARM	Reserved. Also see extension: SPV_ARM_tensors
4191	GraphARM	Reserved. Also see extension: SPV_ARM_graph
4201	CooperativeMatrixLayoutsARM	Reserved. Also see extension: SPV_ARM_cooperative_matrix_layouts
4212	Float8EXT	Reserved. Also see extension: SPV_EXT_float8
4213	Float8CooperativeMatrixEXT	Float8EXT, CooperativeMatrixKHR Reserved. Also see extension: SPV_EXT_float8
4422	FragmentShadingRateKHR	Shader Reserved. Also see extension: SPV_KHR_fragment_shading_rate
4423	SubgroupBallotKHR	Reserved. Also see extension: SPV_KHR_shader_ballot
4427	DrawParameters	Shader Missing before version 1.3. Also see extension: SPV_KHR_shader_draw_parameters
4428	WorkgroupMemoryExplicitLayoutKHR	Shader Reserved. Also see extension: SPV_KHR_workgroup_memory_explicit_layout

	Capability	Implicitly Declares
4429	WorkgroupMemoryExplicitLayout8BitAccessKHR	WorkgroupMemoryExplicitLayoutKHR Reserved . Also see extension: SPV_KHR_workgroup_memory_explicit_layout
4430	WorkgroupMemoryExplicitLayout16BitAccessKHR	WorkgroupMemoryExplicitLayoutKHR Reserved . Also see extension: SPV_KHR_workgroup_memory_explicit_layout
4431	SubgroupVoteKHR	Reserved . Also see extension: SPV_KHR_subgroup_vote
4433	StorageBuffer16BitAccess (StorageUniformBufferBlock16) Uses 16-bit OpTypeFloat and OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the StorageBuffer storage class , the PhysicalStorageBuffer storage class, or the Uniform storage class with the BufferBlock decoration .	Missing before version 1.3 . Also see extension: SPV_KHR_16bit_storage
4434	UniformAndStorageBuffer16BitAccess (StorageUniform16) Uses 16-bit OpTypeFloat and OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the StorageBuffer storage class , the PhysicalStorageBuffer storage class, or the Uniform storage class.	StorageBuffer16BitAccess Missing before version 1.3 . Also see extension: SPV_KHR_16bit_storage
4435	StoragePushConstant16 Uses 16-bit OpTypeFloat and OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the PushConstant storage class .	Missing before version 1.3 . Also see extension: SPV_KHR_16bit_storage
4436	StorageInputOutput16 Uses 16-bit OpTypeFloat and OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the Output storage class .	Missing before version 1.3 . Also see extension: SPV_KHR_16bit_storage
4437	DeviceGroup	Missing before version 1.3 . Also see extension: SPV_KHR_device_group

	Capability	Implicitly Declares
4439	MultiView	Shader Missing before version 1.3 . Also see extension: SPV_KHR_multiview
4441	VariablePointersStorageBuffer Allow variable pointers , each confined to a single Block -decorated struct in the StorageBuffer storage class.	Shader Missing before version 1.3 . Also see extension: SPV_KHR_variable_pointers
4442	VariablePointers Allow variable pointers .	VariablePointersStorageBuffer Missing before version 1.3 . Also see extension: SPV_KHR_variable_pointers
4445	AtomicStorageOps	AtomicStorage Reserved. Also see extension: SPV_KHR_shader_atomic_counter_ops
4447	SampleMaskPostDepthCoverage	Reserved. Also see extension: SPV_KHR_post_depth_coverage
4448	StorageBuffer8BitAccess Uses 8-bit OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the StorageBuffer storage class or the PhysicalStorageBuffer storage class .	Missing before version 1.5 . Also see extension: SPV_KHR_8bit_storage
4449	UniformAndStorageBuffer8BitAccess Uses 8-bit OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the StorageBuffer storage class , the PhysicalStorageBuffer storage class , or the Uniform storage class .	StorageBuffer8BitAccess Missing before version 1.5 . Also see extension: SPV_KHR_8bit_storage
4450	StoragePushConstant8 Uses 8-bit OpTypeInt instructions for creating scalar, vector, and composite types that become members of a block residing in the PushConstant storage class .	Missing before version 1.5 . Also see extension: SPV_KHR_8bit_storage
4464	DenormPreserve Uses the DenormPreserve execution mode .	Missing before version 1.4 . Also see extension: SPV_KHR_float_controls

	Capability	Implicitly Declares
4465	DenormFlushToZero Uses the DenormFlushToZero execution mode .	Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4466	SignedZeroInfNanPreserve Uses the SignedZeroInfNanPreserve execution mode .	Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4467	RoundingModeRTE Uses the RoundingModeRTE execution mode .	Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4468	RoundingModeRTZ Uses the RoundingModeRTZ execution mode .	Missing before version 1.4 . Also see extension: SPV_KHR_float_controls
4471	RayQueryProvisionalKHR	Shader Reserved . Also see extension: SPV_KHR_ray_query
4472	RayQueryKHR	Shader Reserved . Also see extension: SPV_KHR_ray_query
4473	UntypedPointersKHR	Reserved . Also see extension: SPV_KHR_untyped_pointers
4478	RayTraversalPrimitiveCullingKHR	RayQueryKHR, RayTracingKHR Reserved . Also see extensions: SPV_KHR_ray_query , SPV_KHR_ray_tracing
4479	RayTracingKHR	Shader Reserved . Also see extension: SPV_KHR_ray_tracing
4484	TextureSampleWeightedQCOM	Reserved . Also see extension: SPV_QCOM_image_processing
4485	TextureBoxFilterQCOM	Reserved . Also see extension: SPV_QCOM_image_processing

	Capability	Implicitly Declares
4486	TextureBlockMatchQCOM	Reserved. Also see extension: SPV_QCOM_image_processing
4495	TileShadingQCOM	Shader Reserved. Also see extension: SPV_QCOM_tile_shading
4496	CooperativeMatrixConversionQCOM	CooperativeMatrixKHR Reserved. Also see extension: SPV_QCOM_cooperative_matrix_conversion
4498	TextureBlockMatch2QCOM	Reserved. Also see extension: SPV_QCOM_image_processing2
5008	Float16ImageAMD	Shader Reserved. Also see extension: SPV_AMD_gpu_shader_half_float_fetch
5009	ImageGatherBiasLodAMD	Shader Reserved. Also see extension: SPV_AMD_texture_gather_bias_lod
5010	FragmentMaskAMD	Shader Reserved. Also see extension: SPV_AMD_shader_fragment_mask
5013	StencilExportEXT	Shader Reserved. Also see extension: SPV_EXT_shader_stencil_export

	Capability	Implicitly Declares
5015	ImageReadWriteLodAMD	Shader Reserved. Also see extension: SPV_AMD_shader_image_load_store_lod
5016	Int64ImageEXT	Shader Reserved. Also see extension: SPV_EXT_shader_image_int64
5055	ShaderClockKHR	Reserved. Also see extension: SPV_KHR_shader_clock
5067	ShaderEnqueueAMD	Shader Reserved. Also see extension: SPV_AMD_shader_enqueue
5087	QuadControlKHR	Reserved. Also see extension: SPV_KHR_quad_control
5112	Int4TypeINTEL	Reserved. Also see extension: SPV_INTEL_int4
5114	Int4CooperativeMatrixINTEL	Int4TypeINTEL, CooperativeMatrixKHR Reserved. Also see extension: SPV_INTEL_int4
5116	BFloat16TypeKHR	Reserved. Also see extension: SPV_KHR_bfloat16
5117	BFloat16DotProductKHR	BFloat16TypeKHR Reserved. Also see extension: SPV_KHR_bfloat16
5118	BFloat16CooperativeMatrixKHR	BFloat16TypeKHR, CooperativeMatrixKHR Reserved. Also see extension: SPV_KHR_bfloat16

	Capability	Implicitly Declares
5249	SampleMaskOverrideCoverageNV	SampleRateShading Reserved . Also see extension: SPV_NV_sample_mask_override_coverage
5251	GeometryShaderPassthroughNV	Geometry Reserved . Also see extension: SPV_NV_geometry_shader_passthrough
5254	ShaderViewportIndexLayerEXT (ShaderViewportIndexLayerNV)	MultiViewport Reserved . Also see extensions: SPV_EXT_shader_viewport_index_layer , SPV_NV_viewport_array2
5255	ShaderViewportMaskNV	ShaderViewportIndexLayerEXT Reserved . Also see extension: SPV_NV_viewport_array2
5259	ShaderStereoViewNV	ShaderViewportMaskNV Reserved . Also see extension: SPV_NV_stereo_view_rendering
5260	PerViewAttributesNV	MultiView Reserved . Also see extension: SPV_NVX_multiview_per_view_attributes
5265	FragmentFullyCoveredEXT	Shader Reserved . Also see extension: SPV_EXT_fragment_fully_covered
5266	MeshShadingNV	Shader Reserved . Also see extension: SPV_NV_mesh_shader

	Capability	Implicitly Declares
5282	ImageFootprintNV	Reserved. Also see extension: SPV_NV_shader_image_footprint
5283	MeshShadingEXT	Shader Reserved. Also see extension: SPV_EXT_mesh_shader
5284	FragmentBarycentricKHR (FragmentBarycentricNV)	Reserved. Also see extensions: SPV_NV_fragment_shader_barycentric , SPV_KHR_fragment_shader_barycentric
5288	ComputeDerivativeGroupQuadsKHR (ComputeDerivativeGroupQuadsNV)	Shader Reserved. Also see extensions: SPV_NV_compute_shader_derivatives , SPV_KHR_compute_shader_derivatives
5291	FragmentDensityEXT (ShadingRateNV)	Shader Reserved. Also see extensions: SPV_EXT_fragment_invocation_density , SPV_NV_shading_rate
5297	GroupNonUniformPartitionedNV	Reserved. Also see extension: SPV_NV_shader_subgroup_partitioned
5301	ShaderNonUniform (ShaderNonUniformEXT) Uses the NonUniform decoration on a variable or instruction.	Shader Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5302	RuntimeDescriptorArray (RuntimeDescriptorArrayEXT) Uses arrays of resources which are sized at run-time.	Shader Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing

	Capability	Implicitly Declares
5303	InputAttachmentArrayDynamicIndexing (InputAttachmentArrayDynamicIndexingEXT) Arrays of InputAttachments use dynamically uniform indexing.	InputAttachment Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5304	UniformTexelBufferArrayDynamicIndexing (UniformTexelBufferArrayDynamicIndexingEXT) Arrays of SampledBuffers use dynamically uniform indexing.	SampledBuffer Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5305	StorageTexelBufferArrayDynamicIndexing (StorageTexelBufferArrayDynamicIndexingEXT) Arrays of ImageBuffers use dynamically uniform indexing.	ImageBuffer Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5306	UniformBufferArrayNonUniformIndexing (UniformBufferArrayNonUniformIndexingEXT) Block-decorated arrays in uniform storage classes use non-uniform indexing.	ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5307	SampledImageArrayNonUniformIndexing (SampledImageArrayNonUniformIndexingEXT) Arrays of sampled images use non-uniform indexing.	ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5308	StorageBufferArrayNonUniformIndexing (StorageBufferArrayNonUniformIndexingEXT) Arrays in the StorageBuffer storage class or BufferBlock-decorated arrays use non-uniform indexing.	ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5309	StorageImageArrayNonUniformIndexing (StorageImageArrayNonUniformIndexingEXT) Arrays of non-sampled images use non-uniform indexing.	ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5310	InputAttachmentArrayNonUniformIndexing (InputAttachmentArrayNonUniformIndexingEXT) Arrays of InputAttachments use non-uniform indexing.	InputAttachment, ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing

	Capability	Implicitly Declares
5311	UniformTexelBufferArrayNonUniformIndexing (UniformTexelBufferArrayNonUniformIndexingEXT) Arrays of SampledBuffers use non-uniform indexing.	SampledBuffer, ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5312	StorageTexelBufferArrayNonUniformIndexing (StorageTexelBufferArrayNonUniformIndexingEXT) Arrays of ImageBuffers use non-uniform indexing.	ImageBuffer, ShaderNonUniform Missing before version 1.5. Also see extension: SPV_EXT_descriptor_indexing
5336	RayTracingPositionFetchKHR	Shader Reserved. Also see extension: SPV_KHR_ray_tracing_position_fetch
5340	RayTracingNV	Shader Reserved. Also see extension: SPV_NV_ray_tracing
5341	RayTracingMotionBlurNV	Shader Reserved. Also see extension: SPV_NV_ray_tracing_motion_blur
5345	VulkanMemoryModel (VulkanMemoryModelKHR) Uses the Vulkan memory model . This capability must be declared if and only if the Vulkan memory model is declared.	Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
5346	VulkanMemoryModelDeviceScope (VulkanMemoryModelDeviceScopeKHR) Uses Device scope with any instruction when the Vulkan memory model is declared.	Missing before version 1.5. Also see extension: SPV_KHR_vulkan_memory_model
5347	PhysicalStorageBufferAddresses (PhysicalStorageBufferAddressesEXT) Uses physical addressing on storage buffers.	Shader Missing before version 1.5. Also see extensions: SPV_EXT_physical_storage_buffer , SPV_KHR_physical_storage_buffer

	Capability	Implicitly Declares
5350	ComputeDerivativeGroupLinearKHR (ComputeDerivativeGroupLinearNV)	Shader Reserved . Also see extensions: SPV_NV_compute_shader_derivatives , SPV_KHR_compute_shader_derivatives
5353	RayTracingProvisionalKHR	Shader Reserved . Also see extension: SPV_KHR_ray_tracing
5357	CooperativeMatrixNV	Shader Reserved . Also see extension: SPV_NV_cooperative_matrix
5363	FragmentShaderSampleInterlockEXT	Shader Reserved . Also see extension: SPV_EXT_fragment_shader_interlock
5372	FragmentShaderShadingRateInterlockEXT	Shader Reserved . Also see extension: SPV_EXT_fragment_shader_interlock
5373	ShaderSMBuiltinsNV	Shader Reserved . Also see extension: SPV_NV_shader_sm_builtins
5378	FragmentShaderPixelInterlockEXT	Shader Reserved . Also see extension: SPV_EXT_fragment_shader_interlock
5379	DemoteToHelperInvocation (DemoteToHelperInvocationEXT)	Shader Missing before version 1.6 . Also see extension: SPV_EXT_demote_to_helper_invocation

	Capability	Implicitly Declares
5380	DisplacementMicromapNV	Shader Reserved. Also see extension: SPV_NV_displacement_micromap
5381	RayTracingOpacityMicromapEXT	Shader Reserved. Also see extension: SPV_EXT_opacity_micromap
5383	ShaderInvocationReorderNV	RayTracingKHR Reserved. Also see extension: SPV_NV_shader_invocation_reorder
5390	BindlessTextureNV	Reserved. Also see extension: SPV_NV_bindless_texture
5391	RayQueryPositionFetchKHR	Shader Reserved. Also see extension: SPV_KHR_ray_tracing_position_fetch
5394	CooperativeVectorNV	Reserved. Also see extension: SPV_NV_cooperative_vector
5404	AtomicFloat16VectorNV	Reserved. Also see extension: SPV_NV_shader_atomic_fp16_vector
5409	RayTracingDisplacementMicromapNV	RayTracingKHR Reserved. Also see extension: SPV_NV_displacement_micromap
5414	RawAccessChainsNV	Reserved. Also see extension: SPV_NV_raw_access_chains

	Capability	Implicitly Declares
5418	RayTracingSpheresGeometryNV	Reserved. Also see extension: SPV_NV_linear_swept_spheres
5419	RayTracingLinearSweptSpheresGeometryNV	Reserved. Also see extension: SPV_NV_linear_swept_spheres
5430	CooperativeMatrixReductionsNV	Reserved. Also see extension: SPV_NV_cooperative_matrix2
5431	CooperativeMatrixConversionsNV	Reserved. Also see extension: SPV_NV_cooperative_matrix2
5432	CooperativeMatrixPerElementOperationsNV	Reserved. Also see extension: SPV_NV_cooperative_matrix2
5433	CooperativeMatrixTensorAddressingNV	Reserved. Also see extension: SPV_NV_cooperative_matrix2
5434	CooperativeMatrixBlockLoadsNV	Reserved. Also see extension: SPV_NV_cooperative_matrix2
5435	CooperativeVectorTrainingNV	Reserved. Also see extension: SPV_NV_cooperative_vector
5437	RayTracingClusterAccelerationStructureNV	RayTracingKHR Reserved. Also see extension: SPV_NV_cluster_acceleration_structure
5439	TensorAddressingNV	Reserved. Also see extension: SPV_NV_tensor_addressing
5568	SubgroupShuffleINTEL	Reserved. Also see extension: SPV_INTEL_subgroups

	Capability	Implicitly Declares
5569	SubgroupBufferBlockIOINTEL	Reserved. Also see extension: SPV_INTEL_subgroups
5570	SubgroupImageBlockIOINTEL	Reserved. Also see extension: SPV_INTEL_subgroups
5579	SubgroupImageMediaBlockIOINTEL	Reserved. Also see extension: SPV_INTEL_media_block_io
5582	RoundToInfinityINTEL	Reserved. Also see extension: SPV_INTEL_float_controls2
5583	FloatingPointModeINTEL	Reserved. Also see extension: SPV_INTEL_float_controls2
5584	IntegerFunctions2INTEL	Reserved. Also see extension: SPV_INTEL_shader_integer_functions2
5603	FunctionPointersINTEL	Reserved. Also see extension: SPV_INTEL_function_pointers
5604	IndirectReferencesINTEL	Reserved. Also see extension: SPV_INTEL_function_pointers
5606	AsmINTEL	Reserved. Also see extension: SPV_INTEL_inline_assembly
5612	AtomicFloat32MinMaxEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float_min_max
5613	AtomicFloat64MinMaxEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float_min_max

	Capability	Implicitly Declares
5616	AtomicFloat16MinMaxEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float_min_max
5617	VectorComputeIntel	VectorAnyIntel Reserved. Also see extension: SPV_INTEL_vector_compute
5619	VectorAnyIntel	Reserved. Also see extension: SPV_INTEL_vector_compute
5629	ExpectAssumeKHR	Reserved. Also see extension: SPV_KHR_expect_assume
5696	SubgroupAvcMotionEstimationIntel	Reserved. Also see extension: SPV_INTEL_device_side_avc_motion_estimation
5697	SubgroupAvcMotionEstimationIntraIntel	Reserved. Also see extension: SPV_INTEL_device_side_avc_motion_estimation
5698	SubgroupAvcMotionEstimationChromaIntel	Reserved. Also see extension: SPV_INTEL_device_side_avc_motion_estimation
5817	VariableLengthArrayIntel	Reserved. Also see extension: SPV_INTEL_variable_length_array
5821	FunctionFloatControlIntel	Reserved. Also see extension: SPV_INTEL_float_controls2
5824	FPGAMemoryAttributesIntel	Reserved. Also see extension: SPV_INTEL_fpga_memory_attributes

	Capability	Implicitly Declares
5837	FPFastMathModelINTEL	Kernel Reserved. Also see extension: SPV_INTEL_fp_fast_math_mode
5844	ArbitraryPrecisionIntegersINTEL	Reserved. Also see extension: SPV_INTEL_arbitrary_precision_integers
5845	ArbitraryPrecisionFloatingPointINTEL	Reserved. Also see extension: SPV_INTEL_arbitrary_precision_floating_point
5886	UnstructuredLoopControlsINTEL	Reserved. Also see extension: SPV_INTEL_unstructured_loop_controls
5888	FPGALoopControlsINTEL	Reserved. Also see extension: SPV_INTEL_fpga_loop_controls
5892	KernelAttributesINTEL	Reserved. Also see extension: SPV_INTEL_kernel_attributes
5897	FPGAKernelAttributesINTEL	Reserved. Also see extension: SPV_INTEL_kernel_attributes
5898	FPGAMemoryAccessesINTEL	Reserved. Also see extension: SPV_INTEL_fpga_memory_accesses
5904	FPGAClusterAttributesINTEL	Reserved. Also see extension: SPV_INTEL_fpga_cluster_attributes
5906	LoopFuseINTEL	Reserved. Also see extension: SPV_INTEL_loop_fuse
5908	FPGADSPControlINTEL	Reserved. Also see extension: SPV_INTEL_fpga_dsp_control

	Capability	Implicitly Declares
5910	MemoryAccessAliasingINTEL	Reserved. Also see extension: SPV_INTEL_memory_access_aliasing
5916	FPGAInvocationPipeliningAttributesINTEL	Reserved. Also see extension: SPV_INTEL_fpga_invocation_pipelining_attributes
5920	FPGABufferLocationINTEL	Reserved. Also see extension: SPV_INTEL_fpga_buffer_location
5922	ArbitraryPrecisionFixedPointINTEL	Reserved. Also see extension: SPV_INTEL_arbitrary_precision_fixed_point
5935	USMStorageClassesINTEL	Reserved. Also see extension: SPV_INTEL_usm_storage_classes
5939	RuntimeAlignedAttributeINTEL	Reserved. Also see extension: SPV_INTEL_runtime_aligned
5943	IOPipesINTEL	Reserved. Also see extension: SPV_INTEL_io_pipes
5945	BlockingPipesINTEL	Reserved. Also see extension: SPV_INTEL_blocking_pipes
5948	FPGARegINTEL	Reserved. Also see extension: SPV_INTEL_fpga_reg
6016	DotProductInputAll (DotProductInputAllKHR) Uses vector of any integer type as input to the dot product instructions	Missing before version 1.6. Also see extension: SPV_KHR_integer_dot_product
6017	DotProductInput4x8Bit (DotProductInput4x8BitKHR) Uses vectors of four components of 8-bit integer type as inputs to the dot product instructions	Int8 Missing before version 1.6. Also see extension: SPV_KHR_integer_dot_product

	Capability	Implicitly Declares
6018	DotProductInput4x8BitPacked (DotProductInput4x8BitPackedKHR) Uses 32-bit integer scalars packing 4-component vectors of 8-bit integers as inputs to the dot product instructions	Missing before version 1.6. Also see extension: SPV_KHR_integer_dot_product
6019	DotProduct (DotProductKHR) Uses dot product instructions	Missing before version 1.6. Also see extension: SPV_KHR_integer_dot_product
6020	RayCullMaskKHR	Reserved. Also see extension: SPV_KHR_ray_cull_mask
6022	CooperativeMatrixKHR	Reserved. Also see extension: SPV_KHR_cooperative_matrix
6024	ReplicatedCompositesEXT	Reserved. Also see extension: SPV_EXT_replicated_composites
6025	BitInstructions	Reserved. Also see extension: SPV_KHR_bit_instructions
6026	GroupNonUniformRotateKHR	GroupNonUniform Reserved. Also see extension: SPV_KHR_subgroup_rotate
6029	FloatControls2	Reserved. Also see extension: SPV_KHR_float_controls2
6033	AtomicFloat32AddEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float_add
6034	AtomicFloat64AddEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float_add
6089	LongCompositesINTEL	Reserved. Also see extension: SPV_INTEL_long_composites

	Capability	Implicitly Declares
6094	OptNoneEXT (OptNoneINTEL)	Reserved. Also see extensions: SPV_EXT_optnone , SPV_INTEL_optnone
6095	AtomicFloat16AddEXT	Reserved. Also see extension: SPV_EXT_shader_atomic_float16_add
6114	DebugInfoModuleINTEL	Reserved. Also see extension: SPV_INTEL_debug_module
6115	BFloat16ConversionINTEL	Reserved. Also see extension: SPV_INTEL_bfloat16_conversion
6141	SplitBarrierINTEL	Reserved. Also see extension: SPV_INTEL_split_barrier
6144	ArithmeticFenceEXT	Reserved. Also see extension: SPV_EXT_arithmetic_fence
6150	FPGAClusterAttributesV2INTEL	FPGAClusterAttributesINTEL Reserved. Also see extension: SPV_INTEL_fpga_cluster_attributes
6161	FPGAKernelAttributesv2INTEL	FPGAKernelAttributesINTEL Reserved. Also see extension: SPV_INTEL_kernel_attributes
6162	TaskSequenceINTEL	Reserved. Also see extension: SPV_INTEL_task_sequence
6169	FPMaxErrorINTEL	Reserved. Also see extension: SPV_INTEL_fp_max_error

	Capability	Implicitly Declares
6171	FPGALatencyControlINTEL	Reserved. Also see extension: SPV_INTEL_fpga_latency_control
6174	FPGAArgumentInterfacesINTEL	Reserved. Also see extension: SPV_INTEL_fpga_argument_interfaces
6187	GlobalVariableHostAccessINTEL	Reserved. Also see extension: SPV_INTEL_global_variable_host_access
6189	GlobalVariableFPGADecorationsINTEL	Reserved. Also see extension: SPV_INTEL_global_variable_fpga_decorations
6220	SubgroupBufferPrefetchINTEL	Reserved. Also see extension: SPV_INTEL_subgroup_buffer_prefetch
6228	Subgroup2DBlockIOINTEL	Reserved. Also see extension: SPV_INTEL_2d_block_io
6229	Subgroup2DBlockTransformINTEL	Subgroup2DBlockIOINTEL Reserved. Also see extension: SPV_INTEL_2d_block_io
6230	Subgroup2DBlockTransposeINTEL	Subgroup2DBlockIOINTEL Reserved. Also see extension: SPV_INTEL_2d_block_io
6236	SubgroupMatrixMultiplyAccumulateINTEL	Reserved. Also see extension: SPV_INTEL_subgroup_matrix_multiply_accumulate
6241	TernaryBitwiseFunctionINTEL	Reserved. Also see extension: SPV_INTEL_ternary_bitwise_function

Capability		Implicitly Declares
6245	SpecConditionalINTEL	Reserved. Also see extension: SPV_INTEL_function_variants
6246	FunctionVariantsINTEL	SpecConditionalINTEL Reserved. Also see extension: SPV_INTEL_function_variants
6400	GroupUniformArithmeticKHR	Reserved. Also see extension: SPV_KHR_uniform_group_instructions
6425	TensorFloat32RoundingINTEL	Reserved. Also see extension: SPV_INTEL_tensor_float32_conversion
6427	MaskedGatherScatterINTEL	Reserved. Also see extension: SPV_INTEL_masked_gather_scatter
6441	CacheControlsINTEL	Reserved. Also see extension: SPV_INTEL_cache_controls
6460	RegisterLimitsINTEL	Reserved. Also see extension: SPV_INTEL_maximum_registers
6528	BindlessImagesINTEL	Reserved. Also see extension: SPV_INTEL_bindless_images

3.2.31. Ray Flags

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Ray Flags		Enabling Capabilities
0x0	None	
0x1	OpaqueKHR	RayQueryKHR, RayTracingKHR Reserved.

Ray Flags		Enabling Capabilities
0x2	NoOpaqueKHR	RayQueryKHR, RayTracingKHR Reserved.
0x4	TerminateOnFirstHitKHR	RayQueryKHR, RayTracingKHR Reserved.
0x8	SkipClosestHitShaderKHR	RayQueryKHR, RayTracingKHR Reserved.
0x10	CullBackFacingTrianglesKHR	RayQueryKHR, RayTracingKHR Reserved.
0x20	CullFrontFacingTrianglesKHR	RayQueryKHR, RayTracingKHR Reserved.
0x40	CullOpaqueKHR	RayQueryKHR, RayTracingKHR Reserved.
0x80	CullNoOpaqueKHR	RayQueryKHR, RayTracingKHR Reserved.
0x100	SkipTrianglesKHR (SkipBuiltinPrimitivesNV)	RayTraversalPrimitiveCullingKHR Reserved.
0x200	SkipAABBsKHR	RayTraversalPrimitiveCullingKHR Reserved.
0x400	ForceOpacityMicromap2StateEXT	RayTracingOpacityMicromapEXT Reserved.

3.2.32. Ray Query Intersection

Ray Query Intersection		Enabling Capabilities
0	RayQueryCandidateIntersectionKHR	RayQueryKHR Reserved.
1	RayQueryCommittedIntersectionKHR	RayQueryKHR Reserved.

3.2.33. Ray Query Committed Type

Ray Query Committed Type		Enabling Capabilities
0	RayQueryCommittedIntersectionNoneKHR	RayQueryKHR Reserved.
1	RayQueryCommittedIntersectionTriangleKHR	RayQueryKHR Reserved.
2	RayQueryCommittedIntersectionGeneratedKHR	RayQueryKHR Reserved.

3.2.34. Ray Query Candidate Type

Ray Query Candidate Type		Enabling Capabilities
0	RayQueryCandidateIntersectionTriangleKHR	RayQueryKHR Reserved.
1	RayQueryCandidateIntersectionAABBKHR	RayQueryKHR Reserved.

3.2.35. Fragment Shading Rate

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Fragment Shading Rate		Enabling Capabilities
0x0	None	
0x1	Vertical2Pixels	FragmentShadingRateKHR Reserved.
0x2	Vertical4Pixels	FragmentShadingRateKHR Reserved.
0x4	Horizontal2Pixels	FragmentShadingRateKHR Reserved.
0x8	Horizontal4Pixels	FragmentShadingRateKHR Reserved.

3.2.36. FP Denorm Mode

Floating point denormalized handling mode.

FP Denorm Mode		Enabling Capabilities
0	Preserve	FunctionFloatControlINTEL Reserved.
1	FlushToZero	FunctionFloatControlINTEL Reserved.

3.2.37. FP Operation Mode

Floating point operation mode.

FP Operation Mode		Enabling Capabilities
0	IEEE	FunctionFloatControlINTEL Reserved.
1	ALT	FunctionFloatControlINTEL Reserved.

3.2.38. Quantization Mode

Quantization Mode		Enabling Capabilities
0	TRN	ArbitraryPrecisionFixedPointINTEL Reserved.
1	TRN_ZERO	ArbitraryPrecisionFixedPointINTEL Reserved.
2	RND	ArbitraryPrecisionFixedPointINTEL Reserved.
3	RND_ZERO	ArbitraryPrecisionFixedPointINTEL Reserved.
4	RND_INF	ArbitraryPrecisionFixedPointINTEL Reserved.
5	RND_MIN_INF	ArbitraryPrecisionFixedPointINTEL Reserved.
6	RND_CONV	ArbitraryPrecisionFixedPointINTEL Reserved.

Quantization Mode		Enabling Capabilities
7	RND_CONV_ODD	ArbitraryPrecisionFixedPointINTEL Reserved.

3.2.39. Overflow Mode

Overflow Mode		Enabling Capabilities
0	WRAP	ArbitraryPrecisionFixedPointINTEL Reserved.
1	SAT	ArbitraryPrecisionFixedPointINTEL Reserved.
2	SAT_ZERO	ArbitraryPrecisionFixedPointINTEL Reserved.
3	SAT_SYM	ArbitraryPrecisionFixedPointINTEL Reserved.

3.2.40. Packed Vector Format

Used by:

- [OpSDot](#)
- [OpUDot](#)
- [OpSUDot](#)
- [OpSDotAccSat](#)
- [OpUDotAccSat](#)
- [OpSUDotAccSat](#)

Packed Vector Format		Enabling Capabilities
0	PackedVectorFormat4x8Bit (PackedVectorFormat4x8BitKHR) Interpret 32-bit scalar integer operands as vectors of four 8-bit components. Vector components follow byte significance order with the lowest-numbered component stored in the least significant byte.	Missing before version 1.6. Also see extension: SPV_KHR_integer_dot_product

3.2.41. Cooperative Matrix Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpCooperativeMatrixMulAddKHR](#), [OpCooperativeVectorMatrixMulNV](#), and [OpCooperativeVectorMatrixMulAddNV](#).

Cooperative Matrix Operands		Enabling Capabilities
0x0	None	
0x1	MatrixASignedComponentsKHR	Reserved.
0x2	MatrixBSignedComponentsKHR	Reserved.
0x4	MatrixCSignedComponentsKHR	Reserved.
0x8	MatrixResultSignedComponentsKHR	Reserved.
0x10	SaturatingAccumulationKHR	Reserved.

3.2.42. Cooperative Matrix Layout

Cooperative Matrix Layout		Enabling Capabilities
0	RowMajorKHR	Reserved.
1	ColumnMajorKHR	Reserved.
4202	RowBlockedInterleavedARM	Reserved.
4203	ColumnBlockedInterleavedARM	Reserved.

3.2.43. Cooperative Matrix Use

Cooperative Matrix Use		Enabling Capabilities
0	MatrixAKHR	Reserved.
1	MatrixBKHR	Reserved.
2	MatrixAccumulatorKHR	Reserved.

3.2.44. Cooperative Matrix Reduce Mode

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpCooperativeMatrixReduceNV](#).

Cooperative Matrix Reduce Mode		Enabling Capabilities
0x0	None	
0x1	Row	Reserved.
0x2	Column	Reserved.
0x4	2x2	Reserved.

3.2.45. Tensor Clamp Mode

Tensor Clamp Mode		Enabling Capabilities
0	Undefined	Reserved.

Tensor Clamp Mode		Enabling Capabilities
1	Constant	Reserved.
2	ClampToEdge	Reserved.
3	Repeat	Reserved.
4	RepeatMirrored	Reserved.

3.2.46. Tensor Addressing Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpCooperativeMatrixLoadTensorNV](#) and [OpCooperativeMatrixStoreTensorNV](#).

Tensor Addressing Operands		Enabling Capabilities
0x0	None	
0x1	TensorView	CooperativeMatrixTensorAddressingNV Reserved.
0x2	DecodeFunc	CooperativeMatrixBlockLoadsNV Reserved.

3.2.47. Tensor Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpTensorReadARM](#) and [OpTensorWriteARM](#).

Tensor Operands		Enabling Capabilities
0x0	None	
0x1	NontemporalARM	TensorsARM Reserved.
0x2	OutOfBoundsValueARM	TensorsARM Reserved.
0x4	MakeElementAvailableARM	TensorsARM Reserved.
0x8	MakeElementVisibleARM	TensorsARM Reserved.
0x10	NonPrivateElementARM	TensorsARM Reserved.

3.2.48. Initialization Mode Qualifier

Initialization Mode Qualifier		Enabling Capabilities
0	InitOnDeviceReprogramINTEL	GlobalVariableFPGADecorationsINTEL Reserved.
1	InitOnDeviceResetINTEL	GlobalVariableFPGADecorationsINTEL Reserved.

3.2.49. Host Access Qualifier

Host Access Qualifier		Enabling Capabilities
0	NoneINTEL	GlobalVariableHostAccessINTEL Reserved.
1	ReadINTEL	GlobalVariableHostAccessINTEL Reserved.
2	WriteINTEL	GlobalVariableHostAccessINTEL Reserved.
3	ReadWriteINTEL	GlobalVariableHostAccessINTEL Reserved.

3.2.50. Load Cache Control

Load Cache Control		Enabling Capabilities
0	UncachedINTEL	CacheControlsINTEL Reserved.
1	CachedINTEL	CacheControlsINTEL Reserved.
2	StreamingINTEL	CacheControlsINTEL Reserved.
3	InvalidateAfterReadINTEL	CacheControlsINTEL Reserved.
4	ConstCachedINTEL	CacheControlsINTEL Reserved.

3.2.51. Store Cache Control

Store Cache Control		Enabling Capabilities
0	UncachedINTEL	CacheControlsINTEL Reserved.
1	WriteThroughINTEL	CacheControlsINTEL Reserved.
2	WriteBackINTEL	CacheControlsINTEL Reserved.
3	StreamingINTEL	CacheControlsINTEL Reserved.

3.2.52. Named Maximum Number of Registers

Named Maximum Number of Registers		Enabling Capabilities
0	AutoINTEL	RegisterLimitsINTEL Reserved.

3.2.53. Matrix Multiply Accumulate Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by [OpSubgroupMatrixMultiplyAccumulateINTEL](#).

Matrix Multiply Accumulate Operands		Enabling Capabilities
0x0	None	
0x1	MatrixASignedComponentsINTEL	Reserved.
0x2	MatrixBSignedComponentsINTEL	Reserved.
0x4	MatrixCBFloat16INTEL	Reserved.
0x8	MatrixResultBFloat16INTEL	Reserved.
0x10	MatrixAPackedInt8INTEL	Reserved.
0x20	MatrixBPackedInt8INTEL	Reserved.
0x40	MatrixAPackedInt4INTEL	Reserved.
0x80	MatrixBPackedInt4INTEL	Reserved.
0x100	MatrixATF32INTEL	Reserved.
0x200	MatrixBTF32INTEL	Reserved.
0x400	MatrixAPackedFloat16INTEL	Reserved.

Matrix Multiply Accumulate Operands		Enabling Capabilities
0x800	MatrixBPackedFloat16INTEL	Reserved.
0x1000	MatrixAPackedBFloat16INTEL	Reserved.
0x2000	MatrixBPackedBFloat16INTEL	Reserved.

3.2.54. Raw Access Chain Operands

This is a literal mask; it can be formed by combining the bits from multiple rows in the table below.

Used by **OpRawAccessChainNV**.

Raw Access Chain Operands		Enabling Capabilities
0x0	None	
0x1	RobustnessPerComponentNV	RawAccessChainsNV Reserved.
0x2	RobustnessPerElementNV	RawAccessChainsNV Reserved.

3.2.55. FP Encoding

Specifies an alternative floating point encoding.

The *Width(s)* column specifies the set of valid width the encoding operand can be used with. If no value is provided, the valid widths for the operand are defined by the client API. Otherwise, the *Width* operand of **OpTypeFloat** must match one the specified values.

Used by **OpTypeFloat**.

FP Encoding		Width(s)	Enabling Capabilities
0	BFloat16KHR	16	BFloat16TypeKHR Reserved.
4214	Float8E4M3EXT	8	Float8EXT Reserved.
4215	Float8E5M2EXT	8	Float8EXT Reserved.

3.2.56. Cooperative Vector Matrix Layout

Cooperative Vector Matrix Layout		Enabling Capabilities
0	RowMajorNV	Reserved.

Cooperative Vector Matrix Layout		Enabling Capabilities
1	ColumnMajorNV	Reserved.
2	InferencingOptimalNV	Reserved.
3	TrainingOptimalNV	Reserved.

3.2.57. Cooperative Vector Matrix Component Type

Cooperative Vector Matrix Component Type		Enabling Capabilities
0	Float16NV	Reserved.
1	Float32NV	Reserved.
2	Float64NV	Reserved.
3	SignedInt8NV	Reserved.
4	SignedInt16NV	Reserved.
5	SignedInt32NV	Reserved.
6	SignedInt64NV	Reserved.
7	UnsignedInt8NV	Reserved.
8	UnsignedInt16NV	Reserved.
9	UnsignedInt32NV	Reserved.
10	UnsignedInt64NV	Reserved.
1000 4910 00	SignedInt8PackedNV	Reserved.
1000 4910 01	UnsignedInt8PackedNV	Reserved.
1000 4910 02	FloatE4M3NV	Reserved.
1000 4910 03	FloatE5M2NV	Reserved.

3.3. Instructions

Form for each instruction:

Opcode Name (name-alias, name-alias, ...) Instruction description. <i>Word Count</i> is the high-order 16 bits of word 0 of the instruction, holding its total <i>WordCount</i> . If the instruction takes a variable number of operands, <i>Word Count</i> also says "+ variable", after stating the minimum size of the instruction. <i>Opcode</i> is the low-order 16 bits of word 0 of the instruction, holding its opcode enumerant. <i>Results</i> , when present, are any <i>Result <id></i> or <i>Result Type</i> created by the instruction. Each <i>Result <id></i> is always 32 bits. <i>Operands</i> , when present, are any literals, other instruction's <i>Result <id></i> , etc., consumed by the instruction. Each operand is always 32 bits.			Capability Enabling Capabilities (when needed)
<i>Word Count</i>	<i>Opcode</i>	<i>Results</i>	<i>Operands</i>

3.3.1. Miscellaneous Instructions

OpNop This has no semantic impact and can safely be removed from a module.			
1			0
OpUndef Make an <i>intermediate</i> object whose value is undefined. <i>Result Type</i> is the type of object to make. <i>Result Type</i> can be any type except <i>OpTypeVoid</i> . Each consumption of <i>Result <id></i> yields an arbitrary, possibly different bit pattern or abstract value resulting in possibly different concrete, abstract, or opaque values.			
3	1	<i><id></i> <i>Result Type</i>	<i>Result <id></i>

OpSizeOf				Capability: Addresses	
Computes the run-time size of the type pointed to by <i>Pointer</i>					
<i>Result Type</i> must be a 32-bit <i>integer type</i> scalar.					
<i>Pointer</i> must point to a concrete type.					
4	321	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pointer</i>	
OpCooperativeMatrixLengthKHR				Capability: CooperativeMatrixKHR	
Reserved.				Reserved.	
4	4460	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Type</i>	
OpAssumeTrueKHR			Capability: ExpectAssumeKHR		
Reserved.			Reserved.		
2	5630	<id> <i>Condition</i>			
OpExpectKHR			Capability: ExpectAssumeKHR		
Reserved.			Reserved.		
5	5631	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Value</i>	<id> <i>ExpectedValue</i>
OpArithmeticFenceEXT				Capability: ArithmeticFenceEXT	
Reserved.				Reserved.	
4	6145	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Target</i>	

3.3.2. Debug Instructions

OpSourceContinued Continue specifying the <i>Source</i> text from the previous instruction. This has no semantic impact and can safely be removed from a module. <i>Continued Source</i> is a continuation of the source text in the previous <i>Source</i> . The previous instruction must be an OpSource or an OpSourceContinued instruction. As is true for all literal strings, the previous instruction's string was nul terminated. That terminating nul from the previous instruction is not part of the source text; the first character of <i>Continued Source</i> logically immediately follows the last character of <i>Source</i> before its nul.						
2 + variable		2			<i>Literal</i> <i>Continued Source</i>	

OpSource Document what <i>source language</i> and text this module was translated from. This has no semantic impact and can safely be removed from a module. <i>Version</i> is the version of the source language. It is an unsigned 32-bit integer. <i>File</i> is an OpString instruction and is the source-level file name. <i>Source</i> is the text of the source-level file. Each client API specifies what form the <i>Version</i> operand takes, per source language.						
3 + variable	3	<i>Source Language</i>	<i>Literal</i> <i>Version</i>		Optional <id> <i>File</i>	Optional <i>Literal</i> <i>Source</i>

OpSourceExtension Document an extension to the source language. This has no semantic impact and can safely be removed from a module. <i>Extension</i> is a string describing a source-language extension. Its form is dependent on the how the source language describes extensions.						
2 + variable		4			<i>Literal</i> <i>Extension</i>	

OpName Assign a name string to another instruction's <i>Result <id></i> . This has no semantic impact and can safely be removed from a module. <i>Target</i> is the <i>Result <id></i> to assign a name to. It can be the <i>Result <id></i> of any other instruction; a variable, function, type, intermediate result, etc. <i>Name</i> is the string to assign.				
3 + variable	5	<id> <i>Target</i>	<i>Literal</i> <i>Name</i>	

OpMemberName Assign a name string to a member of a structure type. This has no semantic impact and can safely be removed from a module. <i>Type</i> is the <id> from an OpTypeStruct instruction. <i>Member</i> is the number of the member to assign in the structure. The first member is member 0, the next is member 1, ... <i>Member</i> is an unsigned 32-bit integer. <i>Name</i> is the string to assign to the member.					
4 + variable	6	<id> <i>Type</i>	<i>Literal</i> <i>Member</i>	<i>Literal</i> <i>Name</i>	

OpString Assign a <i>Result <id></i> to a string for use by other debug instructions (see OpLine and OpSource). This has no semantic impact and can safely be removed from a module. (Removal also requires removal of all instructions referencing <i>Result <id></i> .) <i>String</i> is the string being assigned a <i>Result <id></i> .				
3 + variable	7	<i>Result <id></i>	<i>Literal</i> <i>String</i>	

OpLine

Add source-level location information. This has no semantic impact and can safely be removed from a module.

This location information applies to the instructions physically following this instruction, up to the first occurrence of any of the following: the next end of block, the next **OpLine** instruction, or the next **OpNoLine** instruction.

File must be an **OpString** instruction and is the source-level file name.

Line is the source-level line number. *Line* is an unsigned 32-bit integer.

Column is the source-level column number. *Column* is an unsigned 32-bit integer.

OpLine can generally immediately precede other instructions, with the following exceptions:

- it may not be used until after the **annotation** instructions, (see the **Logical Layout** section)
- must not be the last instruction in a block, which is defined to end with a **termination instruction**
- if a branch **merge instruction** is used, the last **OpLine** in the block must be before its merge instruction

4	8	<id> <i>File</i>	<i>Literal</i> <i>Line</i>	<i>Literal</i> <i>Column</i>
---	---	---------------------	-------------------------------	---------------------------------

OpNoLine

Discontinue any source-level location information that might be active from a previous **OpLine** instruction. This has no semantic impact and can safely be removed from a module.

This instruction must only appear after the **annotation** instructions (see the **Logical Layout** section). It must not be the last instruction in a block, or the second-to-last instruction if the block has a **merge instruction**. There is not a requirement that there is a preceding **OpLine** instruction.

1	317
---	-----

OpModuleProcessed Document a process that was applied to a module. This has no semantic impact and can safely be removed from a module. <i>Process</i> is a string describing a process and/or tool (processor) that did the processing. Its form is dependent on the processor.		Missing before version 1.1.
2 + variable	330	<i>Literal</i> <i>Process</i>

3.3.3. Annotation Instructions

OpDecorate

Add a [Decoration](#) to another `<id>`.

Target is the `<id>` to decorate. It can potentially be any `<id>` that is a forward reference. A set of decorations can be grouped together by having multiple decoration instructions targeting the same [OpDecorationGroup](#) instruction.

This instruction is only valid if the *Decoration* operand is a [decoration](#) that takes no **Extra Operands**, or takes **Extra Operands** that are not `<id>` operands.

3 + variable	71	<code><id></code> <i>Target</i>	Decoration	<i>Literal, Literal, ...</i> See Decoration .
--------------	----	--	----------------------------	--

OpMemberDecorate

Add a [Decoration](#) to a member of a structure type.

Structure type is the `<id>` of a type from [OpTypeStruct](#).

Member is the number of the member to decorate in the type. The first member is member 0, the next is member 1, ...

Note: See **OpDecorate** for creating groups of decorations for consumption by **OpGroupMemberDecorate**

4 + variable	72	<code><id></code> <i>Structure Type</i>	Literal <i>Member</i>	Decoration	<i>Literal, Literal, ...</i> See Decoration .
--------------	----	--	--	----------------------------	--

OpDecorationGroup

[Deprecated](#) (directly use non-group decoration instructions instead).

A collector for [Decorations](#) from [OpDecorate](#) and [OpDecorateId](#) instructions. All such decoration instructions targeting this **OpDecorationGroup** instruction must precede it. Subsequent [OpGroupDecorate](#) and [OpGroupMemberDecorate](#) instructions that consume this instruction's *Result <id>* will apply these decorations to their targets.

2	73	Result <id>
---	----	-----------------------------------

OpGroupDecorate Deprecated (directly use non-group decoration instructions instead). Add a group of Decorations to another <i><id></i> . <i>Decoration Group</i> is the <i><id></i> of an OpDecorationGroup instruction. <i>Targets</i> is a list of <i><id></i> s to decorate with the groups of decorations. The <i>Targets</i> list must not include the <i><id></i> of any OpDecorationGroup instruction.			
2 + variable	74	<i><id></i> <i>Decoration Group</i>	<i><id></i> , <i><id></i> , ... <i>Targets</i>

OpGroupMemberDecorate Deprecated (directly use non-group decoration instructions instead). Add a group of Decorations to members of structure types. <i>Decoration Group</i> is the <i><id></i> of an OpDecorationGroup instruction. <i>Targets</i> is a list of (<i><id></i> , <i>Member</i>) pairs to decorate with the groups of decorations. Each <i><id></i> in the pair must be a target structure type, and the associated <i>Member</i> is the number of the member to decorate in the type. The first member is member 0, the next is member 1, ...			
2 + variable	75	<i><id></i> <i>Decoration Group</i>	<i><id></i> 1, <i>literal</i> 1, <i><id></i> 2, <i>literal</i> 2, ... <i>Targets</i>

OpDecorateId Add a Decoration to another <i><id></i> , using <i><id></i> s as Extra Operands . <i>Target</i> is the <i><id></i> to decorate. It can potentially be any <i><id></i> that is a forward reference. A set of decorations can be grouped together by having multiple decoration instructions targeting the same OpDecorationGroup instruction. This instruction is only valid if the <i>Decoration</i> operand is a decoration that takes Extra Operands that are <i><id></i> operands. All such <i><id></i> Extra Operands must be constant instructions or OpVariable instructions.			
Missing before version 1.2. Also see extension: SPV_GOOGLE_hlsl_functionality1			
3 + variable	332	<i><id></i> <i>Target</i>	<i>Decoration</i> <i><id></i> , <i><id></i> , ... See Decoration .

OpDecorateString (OpDecorateStringGOOGLE) Add a string Decoration to another <i><id></i> . <i>Target</i> is the <i><id></i> to decorate. It can potentially be any <i><id></i> that is a forward reference, except it must not be the <i><id></i> of an OpDecorationGroup . <i>Decoration</i> is a decoration that takes at least one <i>Literal</i> operand, and has only <i>Literal</i> string operands.				Missing before version 1.4 . Also see extensions: SPV_GOOGLE_decorate_string , SPV_GOOGLE_hlsl_functionality1	
4 + variable	5632	<i><id></i> <i>Target</i>	<i>Decoration</i>	<i>Literal</i> See Decoration .	Optional <i>Literals</i> See Decoration .

OpMemberDecorateString (OpMemberDecorateStringGOOGLE) Add a string Decoration to a member of a structure type. <i>Structure Type</i> is the <i><id></i> of an OpTypeStruct . <i>Member</i> is the number of the member to decorate in the type. <i>Member</i> is an unsigned 32-bit integer. The first member is member 0, the next is member 1, ... <i>Decoration</i> is a decoration that takes at least one <i>Literal</i> operand, and has only <i>Literal</i> string operands.				Missing before version 1.4 . Also see extensions: SPV_GOOGLE_decorate_string , SPV_GOOGLE_hlsl_functionality1	
5 + variable	5633	<i><id></i> <i>Struct Type</i>	<i>Literal</i> <i>Member</i>	<i>Decoration</i>	<i>Literal</i> See Decoration . Optional <i>Literals</i> See Decoration .

3.3.4. Extension Instructions

OpExtension Declare use of an extension to SPIR-V. This allows validation of additional instructions, tokens, semantics, etc. <i>Name</i> is the extension's name string.		
2 + variable	10	<i>Literal Name</i>

OpExtInstImport Import an extended set of instructions. It can be later referenced by the <i>Result <id></i> . <i>Name</i> is the extended instruction-set's name string. Before version 1.6, there must be an external specification defining the semantics for this extended instruction set. Starting with version 1.6, if <i>Name</i> starts with "NonSemantic.", including the period that separates the namespace "NonSemantic" from the rest of the name, it is encouraged for a specification to exist on the SPIR-V Registry, but it is not required. Starting with version 1.6, an extended instruction-set name which is prefixed with "NonSemantic." is guaranteed to contain only non-semantic instructions , and all OpExtInst instructions referencing this set can be ignored. All instructions within such a set must have only <i><id></i> operands; no literals. When literals are needed, then the <i>Result <id></i> from an OpConstant or OpString instruction is referenced as appropriate. <i>Result <id></i> s from these non-semantic instruction-set instructions must be used only in other non-semantic instructions. See Extended Instruction Sets for more information.		
3 + variable	11	<i>Result <id></i> <i>Literal Name</i>

OpExtInst Execute an instruction in an imported set of extended instructions. <i>Result Type</i> is defined, per <i>Instruction</i> , in the external specification for <i>Set</i> . <i>Set</i> is the result of an OpExtInstImport instruction. <i>Instruction</i> is the enumerant of the instruction to execute within <i>Set</i> . It is an unsigned 32-bit integer. The semantics of the instruction are defined in the external specification for <i>Set</i> . <i>Operand 1, ...</i> are the operands to the extended instruction.						
5 + variable	12	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Set</i>	<i>Literal</i> <i>Instruction</i>	<id>, <id>, ... <i>Operand 1</i> , <i>Operand 2</i> , ...

OpExtInstWithForwardRefsKHR Reserved.					Reserved. Also see extension: SPV_KHR_relaxed_extended_instruction	
5 + variable	4433	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Set</i>	<i>Literal</i> <i>Instruction</i>	<id>, <id>, ... <i>Operand 1</i> , <i>Operand 2</i> , ...

OpConditionalExtensionINTEL Reserved.				Capability: SpecConditionalINTEL Reserved.	
3 + variable		6248	<id> <i>Condition</i>	<i>Literal</i> <i>Name</i>	

3.3.5. Mode-Setting Instructions

OpMemoryModel Set addressing model and memory model for the entire module. <i>Addressing Model</i> selects the module's Addressing Model . <i>Memory Model</i> selects the module's memory model, see Memory Model .					
3	14	<i>Addressing Model</i>		<i>Memory Model</i>	

OpEntryPoint Declare an entry point , its execution model, and its interface. <i>Execution Model</i> is the execution model for the entry point and its static call tree. See Execution Model . <i>Entry Point</i> must be the <i>Result</i> <i><id></i> of an OpFunction instruction. <i>Name</i> is a name string for the entry point. A module must not have two OpEntryPoint instructions with the same Execution Model and the same <i>Name</i> string. <i>Interface</i> is a list of <i><id></i> of global OpVariable instructions. These declare the set of global variables from a module that form the interface of this entry point. The set of <i>Interface</i> <i><id></i> must be equal to or a superset of the global OpVariable <i>Result</i> <i><id></i> referenced by the entry point's static call tree, within the interface's storage classes. Before version 1.4 , the interface's storage classes are limited to the Input and Output storage classes . Starting with version 1.4 , the interface's storage classes are all storage classes used in declaring all global variables referenced by the entry point's call tree. <i>Interface</i> <i><id></i> are forward references. Before version 1.4 , duplication of these <i><id></i> is tolerated. Starting with version 1.4 , an <i><id></i> must not appear more than once.					
4 + variable	15	<i>Execution Model</i>	<i><id></i> <i>Entry Point</i>	<i>Literal</i> <i>Name</i>	<i><id></i> , <i><id></i> , ... <i>Interface</i>

OpExecutionMode Declare an execution mode for an entry point. <i>Entry Point</i> must be the <i>Entry Point</i> <id> operand of an OpEntryPoint instruction. <i>Mode</i> is the execution mode. See Execution Mode . This instruction is only valid if the <i>Mode</i> operand is an execution mode that takes no Extra Operands , or takes Extra Operands that are not <id> operands.				
3 + variable	16	<id> <i>Entry Point</i>	Execution Mode <i>Mode</i>	<i>Literal, Literal, ...</i> See Execution Mode

OpCapability Declare a capability used by this module. <i>Capability</i> is the capability declared by this instruction. There are no restrictions on the order in which capabilities are declared. See the capabilities section for more detail.				
2	17		Capability <i>Capability</i>	

OpExecutionModelId Declare an execution mode for an entry point, using <id>s as Extra Operands . <i>Entry Point</i> must be the <i>Entry Point</i> <id> operand of an OpEntryPoint instruction. <i>Mode</i> is the execution mode. See Execution Mode . This instruction is only valid if the <i>Mode</i> operand is an execution mode that takes Extra Operands that are <id> operands.				Missing before version 1.2.
3 + variable	331	<id> <i>Entry Point</i>	Execution Mode <i>Mode</i>	<id>, <id>, ... See Execution Mode

OpConditionalEntryPointINTEL Reserved.					Capability: SpecConditionalINTEL Reserved.	
5 + variable	6249	<id> <i>Condition</i>	Execution Model	<id> <i>Entry Point</i>	<i>Literal Name</i>	<id>, <id>, ... <i>Interface</i>

OpConditionalCapabilityINTEL Reserved.			Capability: SpecConditionalINTEL Reserved.
3	6250	<i><id></i> <i>Condition</i>	<i>Capability</i> <i>Capability</i>

3.3.6. Type-Declaration Instructions

OpTypeVoid				
Declare the void type.				
2	19	<i>Result <id></i>		

OpTypeBool				
Declare the <i>Boolean type</i> . Values of this type can only be either true or false . There is no physical size or bit pattern defined for these values. If they are stored (in conjunction with OpVariable), they must only be used with logical addressing operations, not physical, and only with non-externally visible shader <i>storage classes</i> : UniformConstant , Workgroup , CrossWorkgroup , Private , Function , Input , and Output .				
2	20	<i>Result <id></i>		

OpTypeInt				
Declare a new <i>integer type</i> . <i>Width</i> specifies how many bits wide the type is. <i>Width</i> is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement. <i>Signedness</i> specifies whether there are signed semantics to preserve or validate. 0 indicates unsigned, or no signedness semantics 1 indicates signed semantics. In all cases, the type of operation of an instruction comes from the instruction's opcode, not the signedness of the operands.				
4	21	<i>Result <id></i>	<i>Literal Width</i>	<i>Literal Signedness</i>

OpTypeFloat Declare a new <i>floating-point type</i> . <i>Width</i> specifies how many bits wide the type is. <i>Width</i> is an unsigned 32-bit integer. <i>Floating Point Encoding</i> specifies the bit pattern of values. Unless <i>Floating Point Encoding</i> is present, the bit pattern of a floating-point value is the binary format described by the IEEE 754 encoding for the specified <i>Width</i> .				
3 + variable	22	<i>Result <id></i>	<i>Literal Width</i>	Optional <i>FP Encoding</i> <i>Floating Point Encoding</i>

OpTypeVector Declare a new <i>vector type</i> . <i>Component Type</i> is the type of each component in the resulting type. It must be a <i>scalar type</i> . <i>Component Count</i> is the number of components in the resulting type. <i>Component Count</i> is an unsigned 32-bit integer. It must be at least 2. Components are numbered consecutively, starting with 0.				
4	23	<i>Result <id></i>	<i><id></i> <i>Component Type</i>	<i>Literal</i> <i>Component Count</i>

OpTypeMatrix Declare a new matrix type. <i>Column Type</i> is the type of each column in the matrix. It must be vector type. <i>Column Count</i> is the number of columns in the new matrix type. <i>Column Count</i> is an unsigned 32-bit integer. It must be at least 2. Matrix columns are numbered consecutively, starting with 0. This is true independently of any <i>Decorations</i> describing the memory layout of a matrix (e.g., RowMajor or MatrixStride).				<i>Capability:</i> Matrix
4	24	<i>Result <id></i>	<i><id></i> <i>Column Type</i>	<i>Literal</i> <i>Column Count</i>

OpTypeImage

Declare a new [image](#) type. Consumed, for example, by [OpTypeSampledImage](#). This type is opaque: values of this type have no defined physical size or bit pattern.

Sampled Type is the type of the components that result from sampling or reading from this image type. Must be a scalar [numerical type](#) or [OpTypeVoid](#).

Dim is the image [dimensionality](#) (Dim).

All the following literals are integers taking one operand each.

Depth is whether or not this image is a depth image. (Note that whether or not depth comparisons are actually done is a property of the sampling opcode, not of this type declaration.)

0 indicates not a depth image

1 indicates a depth image

2 means no indication as to whether this is a depth or non-depth image

Arrayed must be one of the following indicated values:

0 indicates non-arrayed content

1 indicates arrayed content

MS must be one of the following indicated values:

0 indicates single-sampled content

1 indicates multisampled content

Sampled indicates whether or not this image is accessed in combination with a [sampler](#), and must be one of the following values:

0 indicates this is only known at run time, not at compile time

1 indicates an image compatible with sampling operations

2 indicates an image compatible with read/write operations (a storage or subpass data image).

Image Format is the [Image Format](#), which can be **Unknown**, as specified by the client API.

If *Dim* is **SubpassData**, *Sampled* must be 2, *Image Format* must be **Unknown**, and the [Execution Model](#) must be **Fragment**.

Access Qualifier is an image [Access Qualifier](#).

9 + variable	25	Result <id>	<id> <i>Sampled</i> <i>Type</i>	Dim	Literal <i>Depth</i>	Literal <i>Arrayed</i>	Literal <i>MS</i>	Literal <i>Sampled</i>	Image <i>Format</i>	Optional Access Qualifier
-----------------	----	--	---	---------------------	---	---	--------------------------------------	---	--	---

OpTypeSampler Declare the sampler type. Consumed by OpSampledImage . This type is opaque: values of this type have no defined physical size or bit pattern.			
2	26	<i>Result</i> <id>	

OpTypeSampledImage Declare a sampled image type, the <i>Result Type</i> of OpSampledImage , or an externally combined sampler and image. This type is opaque: values of this type have no defined physical size or bit pattern. <i>Image Type</i> must be an OpTypeImage . It is the type of the image in the combined sampler and image type. It must not have a <i>Dim</i> of SubpassData . Additionally, starting with version 1.6 , it must not have a <i>Dim</i> of Buffer .			
3	27	<i>Result</i> <id>	<id> <i>Image Type</i>

OpTypeArray Declare a new array type. <i>Element Type</i> is the type of each element in the array. <i>Length</i> is the number of elements in the array. It must be at least 1. <i>Length</i> must come from a constant instruction of an integer-type scalar whose value is at least 1. Array elements are numbered consecutively, starting with 0.			
4	28	<i>Result</i> <id>	<id> <i>Element Type</i> <id> <i>Length</i>

OpTypeRuntimeArray Declare a new run-time array type. Its length is not known at compile time. If in a OpTypeStruct , it must have the largest Offset decoration of all members in the structure. <i>Element Type</i> is the type of each element in the array. See OpArrayLength for getting the <i>Length</i> of an array of this type.			Capability: Shader
3	29	<i>Result</i> <id>	<id> <i>Element Type</i>

OpTypeStruct Declare a new structure type. <i>Member N type</i> is the type of member <i>N</i> of the structure. The first member is member 0, the next is member 1, ... It is valid for the structure to have no members. If an operand is not yet defined, it must be defined by an OpTypePointer , where the type pointed to is an OpTypeStruct .				
2 + variable	30	Result <id>	<id> , <id> , ... <i>Member 0 type, member 1 type, ...</i>	

OpTypeOpaque Declare a structure type with no body specified.				Capability: Kernel
3 + variable	31	Result <id>	Literal <i>The name of the opaque type.</i>	

OpTypePointer Declare a new pointer type. <i>Storage Class</i> is the Storage Class of the memory holding the object pointed to. If there was a forward reference to this type from an OpTypeForwardPointer , the <i>Storage Class</i> of that instruction must equal the <i>Storage Class</i> of this instruction. <i>Type</i> is the type of the object pointed to.				
4	32	Result <id>	Storage Class	<id> <i>Type</i>

OpTypeFunction Declare a new function type. OpFunction uses this to declare the return type and parameter types of a function. <i>Return Type</i> is the type of the return value of functions of this type. It must be a concrete or abstract type, or a pointer to such a type. If the function has no return value, <i>Return Type</i> must be OpTypeVoid . <i>Parameter N Type</i> is the type <id> of the type of parameter <i>N</i> . It must not be OpTypeVoid				
3 + variable	33	Result <id>	<id> <i>Return Type</i>	<id> , <id> , ... <i>Parameter 0 Type, Parameter 1 Type, ...</i>

OpTypeEvent		Capability: Kernel	
Declare an OpenCL event type.			
2	34	Result <id>	

OpTypeDeviceEvent		Capability: DeviceEnqueue	
Declare an OpenCL device-side event type.			
2	35	Result <id>	

OpTypeReserveld		Capability: Pipes	
Declare an OpenCL reservation id type.			
2	36	Result <id>	

OpTypeQueue		Capability: DeviceEnqueue	
Declare an OpenCL queue type.			
2	37	Result <id>	

OpTypePipe			Capability: Pipes
Declare an OpenCL pipe type.			
Qualifier is the pipe access qualifier.			
3	38	Result <id>	Access Qualifier Qualifier

OpTypeForwardPointer			Capability: Addresses, PhysicalStorageBufferAddresse s
Declare the storage class for a forward reference to a pointer.			
<p><i>Pointer Type</i> is a forward reference to the result of an OpTypePointer. That OpTypePointer instruction must declare <i>Pointer Type</i> to be a pointer to an OpTypeStruct. Any consumption of <i>Pointer Type</i> before its OpTypePointer declaration must be a type-declaration instruction.</p> <p><i>Storage Class</i> is the Storage Class of the memory holding the object pointed to.</p>			
3	39	<id> <i>Pointer Type</i>	<i>Storage Class</i>

OpTypePipeStorage		Capability: PipeStorage
Declare the OpenCL pipe-storage type.		Missing before version 1.1 .
2	322	<i>Result <id></i>

OpTypeNamedBarrier		Capability: NamedBarrier
Declare the named-barrier type.		Missing before version 1.1 .
2	327	<i>Result <id></i>

OpTypeTensorARM				Capability: TensorsARM	
Reserved.				Reserved.	
3 + variable	4163	<i>Result <id></i>	<i><id></i> <i>Element Type</i>	Optional <i><id></i> <i>Rank</i>	Optional <i><id></i> <i>Shape</i>

OpTypeGraphARM				Capability: GraphARM	
Reserved.				Reserved.	
3 + variable	4190	<i>Result <id></i>	<i>Literal</i> <i>NumInputs</i>	<i><id>, <id>, ...</i> <i>InOutTypes</i>	

OpTypeUntypedPointerKHR			Capability: UntypedPointersKHR
Reserved.			Reserved.
3	4417	<i>Result <id></i>	<i>Storage Class</i>

OpTypeCooperativeMatrixKHR						Capability: CooperativeMatrixKHR	
Reserved.						Reserved.	
7	4456	Result <id>	<id> Component Type	Scope <id> Scope	<id> Rows	<id> Columns	<id> Use

OpTypeRayQueryKHR		Capability: RayQueryKHR
Reserved.		Reserved.
2	4472	<i>Result <id></i>

OpTypeHitObjectNV		Capability: ShaderInvocationReorderNV	
Reserved.		Reserved.	
2	5281	<i>Result <id></i>	

OpTypeCooperativeVectorNV				Capability: CooperativeVectorNV	
Reserved.				Reserved.	
4	5288	<i>Result <id></i>	<id> <i>Component Type</i>	<id> <i>Component Count</i>	

OpTypeAccelerationStructureKHR (OpTypeAccelerationStructureNV)		Capability: RayTracingNV, RayTracingKHR, RayQueryKHR, DisplacementMicromapNV			
Reserved.		Reserved.			
2	5341	<i>Result <id></i>			

OpTypeCooperativeMatrixNV				Capability: CooperativeMatrixNV		
Reserved.				Reserved.		
6	5358	<i>Result <id></i>	<id> <i>Component Type</i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Rows</i>	<id> <i>Columns</i>

OpTypeTensorLayoutNV				Capability: TensorAddressingNV	
Reserved.				Reserved.	
4	5370	<i>Result <id></i>	<id> <i>Dim</i>	<id> <i>ClampMode</i>	

OpTypeTensorViewNV				Capability: TensorAddressingNV	
Reserved.				Reserved.	
4 + variable	5371	<i>Result <id></i>	<id> <i>Dim</i>	<id> <i>HasDimensions</i>	<id>, <id>, ... <i>p</i>

OpTypeBufferSurfaceINTEL Reserved.			Capability: VectorComputeINTEL Reserved.
3	6086	<i>Result <id></i>	<i>Access Qualifier</i> <i>AccessQualifier</i>

OpTypeStructContinuedINTEL Reserved.			Capability: LongCompositesINTEL Reserved.
1 + variable		6090	<i><id>, <id>, ...</i> <i>Member 0 type, member 1 type, ...</i>

OpTypeTaskSequenceINTEL Reserved.			Capability: TaskSequenceINTEL Reserved.
2	6199	<i>Result <id></i>	

3.3.7. Constant-Creation Instructions

OpConstantTrue Declare a true <i>Boolean-type</i> scalar constant. <i>Result Type</i> must be the scalar <i>Boolean type</i> .				
3	41	<id> <i>Result Type</i>	<i>Result <id></i>	

OpConstantFalse Declare a false <i>Boolean-type</i> scalar constant. <i>Result Type</i> must be the scalar <i>Boolean type</i> .				
3	42	<id> <i>Result Type</i>	<i>Result <id></i>	

OpConstant Declare a new <i>integer-type</i> or <i>floating-point-type</i> scalar constant. <i>Result Type</i> must be a scalar <i>integer type</i> or <i>floating-point type</i> . <i>Value</i> is the bit pattern for the constant. Types 32 bits wide or smaller take one word. Larger types take multiple words, with low-order words appearing first.				
4 + variable	43	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Literal Value</i>

OpConstantComposite Declare a new <i>composite</i> constant. <i>Result Type</i> must be a <i>composite</i> type, whose top-level members/elements/components/columns have the same type as the types of the <i>Constituents</i> . The ordering must be the same between the top-level types in <i>Result Type</i> and the <i>Constituents</i> . <i>Constituents</i> become members of a structure, or elements of an array, or components of a vector, or columns of a matrix. There must be exactly one <i>Constituent</i> for each top-level member/element/component/column of the result. The <i>Constituents</i> must appear in the order needed by the definition of the <i>Result Type</i> . The <i>Constituents</i> must all be <id>s of non-specialization constant-instruction declarations or an OpUndef .				
3 + variable	44	<id> <i>Result Type</i>	<i>Result <id></i>	<id>, <id>, ... <i>Constituents</i>

OpConstantSampler Declare a new sampler constant. <i>Result Type</i> must be OpTypeSampler . <i>Sampler Addressing Mode</i> is the addressing mode; a literal from Sampler Addressing Mode . <i>Param</i> is a 32-bit integer and is one of: 0: Non Normalized 1: Normalized <i>Sampler Filter Mode</i> is the filter mode; a literal from Sampler Filter Mode .					Capability: LiteralSampler	
6	45	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Sampler Addressing Mode</i>	<i>Literal Param</i>	<i>Sampler Filter Mode</i>

OpConstantNull Declare a new <i>null</i> constant value. The <i>null</i> value is type dependent, defined as follows: - Scalar Boolean: false - Scalar integer: 0 - Scalar floating point: +0.0 (all bits 0) - All other scalars: Abstract - Composites: Members are set recursively to the null constant according to the null value of their constituent types. <i>Result Type</i> must be one of the following types: - Scalar or vector <i>Boolean type</i> - Scalar or vector <i>integer type</i> - Scalar or vector <i>floating-point type</i> - Pointer type - <i>Event type</i> - <i>Device side event type</i> - <i>Reservation id type</i> - <i>Queue type</i> - <i>Composite type</i>						
3	46	<id> <i>Result Type</i>	<i>Result <id></i>			

OpSpecConstantTrue Declare a <i>Boolean-type</i> scalar specialization constant with a default value of true . This instruction can be specialized to become either an OpConstantTrue or OpConstantFalse instruction. <i>Result Type</i> must be the scalar <i>Boolean type</i> . See Specialization .			
3	48	<i><id></i> <i>Result Type</i>	<i>Result <id></i>

OpSpecConstantFalse Declare a <i>Boolean-type</i> scalar specialization constant with a default value of false . This instruction can be specialized to become either an OpConstantTrue or OpConstantFalse instruction. <i>Result Type</i> must be the scalar <i>Boolean type</i> . See Specialization .			
3	49	<i><id></i> <i>Result Type</i>	<i>Result <id></i>

OpSpecConstant Declare a new <i>integer-type</i> or <i>floating-point-type</i> scalar specialization constant. <i>Result Type</i> must be a scalar <i>integer type</i> or <i>floating-point type</i> . <i>Value</i> is the bit pattern for the default value of the constant. Types 32 bits wide or smaller take one word. Larger types take multiple words, with low-order words appearing first. This instruction can be specialized to become an OpConstant instruction. See Specialization .				
4 + variable	50	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i>Literal Value</i>

<p>OpSpecConstantComposite</p> <p>Declare a new <i>composite</i> specialization constant.</p> <p><i>Result Type</i> must be a <i>composite</i> type, whose top-level members/elements/components/columns have the same type as the types of the <i>Constituents</i>. The ordering must be the same between the top-level types in <i>Result Type</i> and the <i>Constituents</i>.</p> <p><i>Constituents</i> become members of a structure, or elements of an array, or components of a vector, or columns of a matrix. There must be exactly one <i>Constituent</i> for each top-level member/element/component/column of the result. The <i>Constituents</i> must appear in the order needed by the definition of the type of the result. The <i>Constituents</i> must be the <i><id></i> of other specialization constants, constant declarations, or an OpUndef.</p> <p>This instruction will be specialized to an OpConstantComposite instruction.</p> <p>See Specialization.</p>				
3 + variable	51	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> , <i><id></i> , ... <i>Constituents</i>

OpSpecConstantOp

Declare a new specialization constant that results from doing an operation.

Result Type must be the type required by the *Result Type* of *Opcode*.

Opcode is an unsigned 32-bit integer. It must equal one of the following opcodes.

OpSConvert, **OpUConvert** (missing before version 1.4),
OpFConvert
OpSNegate, **OpNot**, **OpIAdd**, **OpISub**
OpIMul, **OpUDiv**, **OpSDiv**, **OpUMod**, **OpSRem**, **OpSMod**
OpShiftRightLogical, **OpShiftRightArithmetic**,
OpShiftLeftLogical
OpBitwiseOr, **OpBitwiseXor**, **OpBitwiseAnd**
OpVectorShuffle, **OpCompositeExtract**, **OpCompositeInsert**
OpLogicalOr, **OpLogicalAnd**, **OpLogicalNot**,
OpLogicalEqual, **OpLogicalNotEqual**
OpSelect
OpIEqual, **OpINotEqual**
OpULessThan, **OpSLessThan**
OpUGreaterThan, **OpSGreaterThan**
OpULessThanEqual, **OpSLessThanEqual**
OpUGreaterThanEqual, **OpSGreaterThanEqual**

If the **Shader** capability was declared, **OpQuantizeToF16** is also valid.

If the **Kernel** capability was declared, the following opcodes are also valid:

OpConvertFToS, **OpConvertSToF**
OpConvertFToU, **OpConvertUToF**
OpUConvert, **OpConvertPtrToU**, **OpConvertUToPtr**
OpGenericCastToPtr, **OpPtrCastToGeneric**, **OpBitcast**
OpFNegate, **OpFAdd**, **OpFSub**, **OpFMul**, **OpFDiv**, **OpFRem**,
OpFMod
OpAccessChain, **OpInBoundsAccessChain**
OpPtrAccessChain, **OpInBoundsPtrAccessChain**

Operands are the operands required by *opcode*, and satisfy the semantics of *opcode*. In addition, all *Operands* that are *<id>s* must be either:

- the *<id>s* of other [constant instructions](#), or
- **OpUndef**, when allowed by *opcode*, or
- for the **AccessChain** named opcodes, their *Base* is allowed to be a global (module scope) [OpVariable](#) instruction.

See [Specialization](#).

4 + variable	52	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i>Literal</i> <i>Opcode</i>	<i><id></i> , <i><id></i> , ... <i>Operands</i>
--------------	----	---	--------------------------	---------------------------------	--

OpConstantCompositeReplicateEXT Reserved.				Capability: ReplicatedCompositesEXT Reserved.
4	4461	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> Value

OpSpecConstantCompositeReplicateEXT Reserved.				Capability: ReplicatedCompositesEXT Reserved.
4	4462	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> Value

OpConstantCompositeContinuedINTEL Reserved.		Capability: LongCompositesINTEL Reserved.		
1 + variable		6091	<id>, <id>, ... Constituents	

OpSpecConstantCompositeContinuedINTEL Reserved.		Capability: LongCompositesINTEL Reserved.		
1 + variable		6092	<id>, <id>, ... Constituents	

OpSpecConstantTargetINTEL Reserved.				Capability: FunctionVariantsINTEL Reserved.	
4 + variable	6251	<id> <i>Result Type</i>	<i>Result</i> <id>	<i>Literal</i> Target	<i>Literal</i> , <i>Literal</i> , ... Features

OpSpecConstantArchitectureINTEL Reserved.						Capability: FunctionVariantsINTEL Reserved.	
7	6252	<id> <i>Result Type</i>	<i>Result</i> <id>	<i>Literal</i> Category	<i>Literal</i> Family	<i>Literal</i> Opcode	<i>Literal</i> Architecture

OpSpecConstantCapabilitiesINTEL Reserved.				Capability: FunctionVariantsINTEL Reserved.
3 + variable	6253	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	Optional <i>Capability</i> <i>Capabilities</i>

3.3.8. Memory Instructions

<div><div>OpVariable</div><div>Allocate an object in memory, resulting in a pointer to it, which can be used with OpLoad and OpStore.</div><div><i>Result Type</i> must be an OpTypePointer. Its <i>Type</i> operand is the type of object in memory.</div><div><i>Storage Class</i> is the Storage Class of the memory holding the object. It must not be Generic. It must be the same as the <i>Storage Class</i> operand of the <i>Result Type</i>. If <i>Storage Class</i> is Function, the memory is allocated on execution of the instruction for the current invocation for each dynamic instance of the function. The current invocation's memory is deallocated when it executes any function termination instruction of the dynamic instance of the function it was allocated by.</div><div><i>Initializer</i> is optional. If <i>Initializer</i> is present, it will be the initial value of the variable's memory content. <i>Initializer</i> must be an <i><id></i> from a constant instruction or a global (module scope) OpVariable instruction. <i>Initializer</i> must have the same type as the type pointed to by <i>Result Type</i>.</div></div>					
4 + variable	59	<id> Result Type	Result <id>	Storage Class	Optional <id> Initializer

OpImageTexelPointer

Form a pointer to a texel of an image. Use of such a pointer is limited to atomic operations.

Result Type must be an **OpTypePointer** whose *Storage Class* operand is **Image**. Its *Type* operand must be a scalar *numerical type* or **OpTypeVoid**.

Image must have a type of **OpTypePointer** with *Type* **OpTypeImage**. The *Sampled Type* of the type of *Image* must be the same as the *Type* pointed to by *Result Type*. The *Dim* operand of *Type* must not be **SubpassData**.

Coordinate and *Sample* specify which texel and sample within the image to form a pointer to.

Coordinate must be a scalar or vector of *integer type*. It must have the number of components specified below, given the following *Arrayed* and *Dim* operands of the type of the **OpTypeImage**.

If *Arrayed* is 0:

- 1D**: scalar
- 2D**: 2 components
- 3D**: 3 components
- Cube**: 3 components
- Rect**: 2 components
- Buffer**: scalar

If *Arrayed* is 1:

- 1D**: 2 components
- 2D**: 3 components
- Cube**: 3 components; the face and layer combine into the 3rd component, *layer_face*, such that face is *layer_face* % 6 and layer is floor(*layer_face* / 6)

Sample must be an *integer type* scalar. It specifies which sample to select at the given coordinate. Behavior is undefined unless it is a valid *<id>* for the value 0 when the **OpTypeImage** has *MS* of 0.

6	60	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Image</i>	<i><id></i> <i>Coordinate</i>	<i><id></i> <i>Sample</i>
---	----	---	--------------------------	-----------------------------------	--	------------------------------------

OpLoad

Load through a pointer.

Result Type is the type of the loaded object. It must be a type with fixed size; i.e., it must not be, nor include, any **OpTypeRuntimeArray** types.

Pointer is the pointer to load through. Its type must be an **OpTypePointer** whose *Type* operand is the same as *Result Type*.

If present, any *Memory Operands* must begin with a **memory operand** literal. If not present, it is the same as specifying the **memory operand None**.

4 + variable	61	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pointer</i>	Optional <i>Memory Operands</i>
--------------	----	----------------------------	--------------------	------------------------	------------------------------------

OpStore

Store through a pointer.

Pointer is the pointer to store through. Its type must be an **OpTypePointer** whose *Type* operand is the same as the type of *Object*.

Object is the object to store.

If present, any *Memory Operands* must begin with a **memory operand** literal. If not present, it is the same as specifying the **memory operand None**.

3 + variable	62	<id> <i>Pointer</i>	<id> <i>Object</i>	Optional <i>Memory Operands</i>
--------------	----	------------------------	-----------------------	------------------------------------

OpCopyMemory

Copy from the memory pointed to by *Source* to the memory pointed to by *Target*. Both operands must be non-void pointers and having the same *<id> Type* operand in their **OpTypePointer** type declaration. Matching Storage Class is not required. The amount of memory copied is the size of the type pointed to. The copied type must have a fixed size; i.e., it must not be, nor include, any **OpTypeRuntimeArray** types.

If present, any *Memory Operands* must begin with a [memory operand](#) literal. If not present, it is the same as specifying the [memory operand](#) **None**. Before **version 1.4**, at most one [memory operands](#) mask can be provided. Starting with **version 1.4** two masks can be provided, as described in [Memory Operands](#). If no masks or only one mask is present, it applies to both *Source* and *Target*. If two masks are present, the first applies to *Target* and must not include **MakePointerVisible**, and the second applies to *Source* and must not include **MakePointerAvailable**.

3 + variable	63	<i><id></i> <i>Target</i>	<i><id></i> <i>Source</i>	Optional Memory Operands	Optional Memory Operands
--------------	----	------------------------------------	------------------------------------	---	---

OpCopyMemorySized

Copy from the memory pointed to by *Source* to the memory pointed to by *Target*.

Size is the number of bytes to copy. It must have a scalar [integer type](#). If it is a [constant instruction](#), the constant value must not be 0. It is invalid for both the constant's type to have *Signedness* of 1 and to have the sign bit set. Otherwise, as a run-time value, *Size* is treated as unsigned, and if its value is 0, no memory access is made.

If present, any *Memory Operands* must begin with a [memory operand](#) literal. If not present, it is the same as specifying the [memory operand](#) **None**. Before **version 1.4**, at most one [memory operands](#) mask can be provided. Starting with **version 1.4** two masks can be provided, as described in [Memory Operands](#). If no masks or only one mask is present, it applies to both *Source* and *Target*. If two masks are present, the first applies to *Target* and must not include **MakePointerVisible**, and the second applies to *Source* and must not include **MakePointerAvailable**.

[Capability:](#)
Addresses,
UntypedPointersKHR

4 + variable	64	<i><id></i> <i>Target</i>	<i><id></i> <i>Source</i>	<i><id></i> <i>Size</i>	Optional Memory Operands	Optional Memory Operands
--------------	----	------------------------------------	------------------------------------	----------------------------------	---	---

OpAccessChain

Create a pointer into a *composite* object.

Result Type must be an **OpTypePointer**. Its *Type* operand must be the type reached by walking the *Base*'s type hierarchy down to the last provided index in *Indexes*, and its *Storage Class* operand must be the same as the Storage Class of *Base*.

If *Result Type* is an array-element pointer that is *decorated* with **ArrayStride**, its *Array Stride* must match the *Array Stride* of the array's type. If the array's type is not decorated with **ArrayStride**, *Result Type* also must not be decorated with **ArrayStride**.

Base must be a pointer, pointing to the base of a composite object.

Indexes walk the type hierarchy to the desired depth, potentially down to scalar granularity. The first index in *Indexes* selects the top-level member/element/component/column of the base composite. All composite constituents use zero-based numbering, as described by their **OpType...** instruction. The second index applies similarly to that result, and so on. Once any non-composite type is reached, there must be no remaining (unused) indexes.

Each index in *Indexes*

- must have a scalar *integer type*
- is treated as signed
- if indexing into a structure, must be an **OpConstant** whose value is in bounds for selecting a member
- if indexing into a vector, array, or matrix, with the result type being a *logical pointer type*, causes undefined behavior if not in bounds.

4 + variable	65	<id> Result Type	Result <id>	<id> Base	<id>, <id>, ... Indexes
--------------	----	---------------------	-------------	--------------	----------------------------

OpInBoundsAccessChain

Has the same semantics as **OpAccessChain**, with the addition that the resulting pointer is known to point within the base object.

4 + variable	66	<id> Result Type	Result <id>	<id> Base	<id>, <id>, ... Indexes
--------------	----	---------------------	-------------	--------------	----------------------------

OpPtrAccessChain

Has the same semantics as [OpAccessChain](#), with the addition of the *Element* operand.

Base is treated as the address of an element in an array, and a new element address is computed from *Base* and *Element* to become the **OpAccessChain** *Base* to walk the type hierarchy as per **OpAccessChain**. This computed *Base* has the same type as the originating *Base*.

To compute the new element address, *Element* is treated as a signed count of elements *E*, relative to the original *Base* element *B*, and the address of element $B + E$ is computed using enough precision to avoid overflow and underflow. For objects in [storage classes](#) requiring [explicit layout](#), the element's address or location is calculated using a stride, which will be the *Base*-type's *Array Stride* if the *Base* type is decorated with **ArrayStride**. For all other objects, the implementation calculates the element's address or location.

With one exception, undefined behavior results when $B + E$ is not an element in the same array (same innermost array, if array types are nested) as *B*. The exception being when $B + E = L$, where *L* is the length of the array: the address computation for element *L* is done with the same stride as any other $B + E$ computation that stays within the array.

If the [storage class](#) of *Base* requires an [explicit layout](#) then its type must be [decorated](#) with **ArrayStride**.

If *Base* points to a structure decorated with **Block** or **BufferBlock** and the value of *Element* is not zero then behavior is undefined.

Note: If *Base* is typed to be a pointer to an array and the desired operation is to select an element of that array, [OpAccessChain](#) should be directly used, as its first *Index* selects the array element.

Capability:

Addresses, VariablePointers, VariablePointersStorageBuffer, PhysicalStorageBufferAddresses

5 + variable	67	<id> Result Type	Result <id>	<id> Base	<id> Element	<id>, <id>, ... Indexes
--------------	----	---------------------	-------------	--------------	-----------------	----------------------------

OpArrayLength Length of a run-time array. The contents of the array are not accessed. <i>Result Type</i> must be an OpTypeInt with 32-bit <i>Width</i> and 0 <i>Signedness</i> . <i>Structure</i> must be a logical pointer to an OpTypeStruct whose last member is a run-time array. <i>Array member</i> is an unsigned 32-bit integer index of the last member of the structure that <i>Structure</i> points to. That member's type must be from OpTypeRuntimeArray .				Capability: Shader	
5	68	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Structure</i>	<i>Literal</i> <i>Array member</i>

OpGenericPtrMemSemantics Result is a valid Memory Semantics which includes mask bits set for the Storage Class for the specific (non-Generic) Storage Class of <i>Pointer</i> . <i>Pointer</i> must point to Generic Storage Class . <i>Result Type</i> must be an OpTypeInt with 32-bit <i>Width</i> and 0 <i>Signedness</i> .				Capability: Kernel	
4	69	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pointer</i>	

OpInBoundsPtrAccessChain					Capability: Addresses	
Has the same semantics as OpPtrAccessChain , with the addition that the resulting pointer is known to point within the base object.						
5 + variable	70	<id> Result Type	Result <id>	<id> Base	<id> Element	<id>, <id>, ... Indexes

OpPtrEqual Result is true if <i>Operand 1</i> and <i>Operand 2</i> have the same value. Result is false if <i>Operand 1</i> and <i>Operand 2</i> have different values. <i>Result Type</i> must be a Boolean type scalar. The types of <i>Operand 1</i> and <i>Operand 2</i> must be OpTypePointer of the same type.				Missing before version 1.4.	
5	401	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpPtrNotEqual Result is true if <i>Operand 1</i> and <i>Operand 2</i> have different values. Result is false if <i>Operand 1</i> and <i>Operand 2</i> have the same value. <i>Result Type</i> must be a <i>Boolean type</i> scalar. The types of <i>Operand 1</i> and <i>Operand 2</i> must be OpTypePointer of the same type.				Missing before version 1.4.	
5	402	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpPtrDiff Element-number subtraction: The number of elements to add to <i>Operand 2</i> to get to <i>Operand 1</i> . <i>Result Type</i> must be an <i>integer type</i> scalar. It is computed as a signed value, as negative differences are allowed, independently of the signed bit in the type. The result equals the low-order <i>N</i> bits of the correct result <i>R</i> , where <i>R</i> is computed with enough precision to avoid overflow and underflow and <i>Result Type</i> has a bitwidth of <i>N</i> bits. The units of <i>Result Type</i> are a count of elements. I.e., the same value you would use as the <i>Element</i> operand to OpPtrAccessChain . The types of <i>Operand 1</i> and <i>Operand 2</i> must be OpTypePointer of exactly the same type, and point to a type that can be aggregated into an array. For an array of length <i>L</i> , <i>Operand 1</i> and <i>Operand 2</i> can point to any element in the range <i>[0, L]</i> , where element <i>L</i> is outside the array but has a representative address computed with the same stride as elements in the array. Additionally, <i>Operand 1</i> must be a valid <i>Base</i> operand of OpPtrAccessChain . Behavior is undefined if <i>Operand 1</i> and <i>Operand 2</i> are not pointers to element numbers in <i>[0, L]</i> in the same array.				Capability: Addresses, VariablePointers, VariablePointersStorageBuffer Missing before version 1.4.	
5	403	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpUntypedVariableKHR					Capability: UntypedPointersKHR	
Reserved.					Reserved.	
4 + variable	4418	<id> <i>Result Type</i>	<i>Result</i> <id>	<i>Storage Class</i>	Optional <id> <i>Data Type</i>	Optional <id> <i>Initializer</i>

OpUntypedAccessChainKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
5 + variable	4419	<id> Result Type	Result <id>	<id> Base Type	<id> Base	<id>, <id>, ... Indexes

OpUntypedInBoundsAccessChainKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
5 + variable	4420	<id> Result Type	Result <id>	<id> Base Type	<id> Base	<id>, <id>, ... Indexes

OpUntypedPtrAccessChainKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
6 + variable	4423	<id> Result Type	Result <id>	<id> Base Type	<id> Base	<id> Element ... Indexes

OpUntypedInBoundsPtrAccessChainKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
6 + variable	4424	<id> Result Type	Result <id>	<id> Base Type	<id> Base	<id> Element ... Indexes

OpUntypedArrayLengthKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
6	4425	<id> Result Type	Result <id>	<id> Structure	<id> Pointer	Literal Array member

OpUntypedPrefetchKHR Reserved.					Capability: UntypedPointersKHR Reserved.	
3 + variable	4426	<id> Pointer Type	<id> Num Bytes	Optional <id> RW	Optional <id> Locality	Optional <id> Cache Type

OpCooperativeMatrixLoadKHR Reserved.						Capability: CooperativeMatrixKHR Reserved.	
5 + variable	4457	<id> Result Type	Result <id>	<id> Pointer	<id> MemoryLayout	Optional <id> Stride	Optional Memory Operands Memory Operand

OpCooperativeMatrixStoreKHR					Capability: CooperativeMatrixKHR	
Reserved.					Reserved.	
4 + variable	4458	<id> Pointer	<id> Object	<id> MemoryLayout	Optional <id> Stride	Optional Memory Operands Memory Operand

OpCooperativeVectorLoadNV					Capability: CooperativeVectorNV	
Reserved.					Reserved.	
5 + variable	5302	<id> Result Type	Result <id>	<id> Pointer	<id> Offset	Optional Memory Operands

OpCooperativeVectorStoreNV Reserved.				Capability: CooperativeVectorNV Reserved.	
4 + variable	5303	<id> Pointer	<id> Offset	<id> Object	Optional Memory Operands

OpCooperativeMatrixLoadTensorNV Reserved.						Capability: CooperativeMatrixTensorAddressingNV Reserved.	
8	5367	<id> Result Type	Result <id>	<id> Pointer	<id> Object	<id> TensorLayout	Memory Operands Memory Operand Tensor Addressing Operands

OpCooperativeMatrixStoreTensorNV Reserved.					Capability: CooperativeMatrixTensorAddressingNV Reserved.	
6	5368	<id> Pointer	<id> Object	<id> TensorLayout	Memory Operands Memory Operand	Tensor Addressing Operands Tensor Addressing Operands

OpRawAccessChainNV Reserved.							Capability: RawAccessChainsNV Reserved.	
7 + variable	5398	<id> Result Type	Result <id>	<id> Base	<id> Byte stride	<id> Element index	<id> Byte offset	Optional Raw Access Chain Operands

OpMaskedGatherINTEL Reserved.						Capability: MaskedGatherScatterINTEL Reserved.	
7	6428	<id> Result Type	Result <id>	<id> PtrVector	Literal Alignment	<id> Mask	<id> FillEmpty

OpMaskedScatterINTEL Reserved.				Capability: MaskedGatherScatterINTEL Reserved.		
5	6429	<id> InputVector	<id> PtrVector	Literal Alignment	<id> Mask	

3.3.9. Function Instructions

<div><div>OpFunction</div><div>Add a function. This instruction must be immediately followed by one OpFunctionParameter instruction per each formal parameter of this function. This function's body or declaration terminates with the next OpFunctionEnd instruction.</div><div>Result Type must be the same as the Return Type declared in Function Type.</div><div>Function Type is the result of an OpTypeFunction, which declares the types of the return value and parameters of the function.</div></div>					
5	54	<id> Result Type	Result <id>	Function Control	<id> Function Type

<div><div>OpFunctionParameter</div><div>Declare a formal parameter of the current function.</div><div>Result Type is the type of the parameter.</div><div>This instruction must immediately follow an OpFunction or OpFunctionParameter instruction. The order of contiguous OpFunctionParameter instructions is the same order arguments are listed in an OpFunctionCall instruction to this function. It is also the same order in which Parameter Type operands are listed in the OpTypeFunction of the Function Type operand for this function's OpFunction instruction.</div></div>					
3	55	<id> Result Type	Result <id>		

<div><div>OpFunctionEnd</div><div>Last instruction of a function.</div></div>					
1		56			

OpFunctionCall

Call a function.

Result Type is the type of the return value of the function. It must be the same as the *Return Type* operand of the *Function Type* operand of the *Function* operand.

Function is an **OpFunction** instruction. This could be a forward reference.

Argument N is the object to copy to parameter *N* of *Function*.

Note: A forward call is possible because there is no missing type information: *Result Type* must match the *Return Type* of the function, and the calling argument types must match the formal parameter types.

4 + variable	57	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Function</i>	<id>, <id>, ... <i>Argument 0</i> , <i>Argument 1</i> , ...
--------------	----	----------------------------	--------------------	-------------------------	---

OpCooperativeMatrixPerElementOpNV

Reserved.

Capability:

CooperativeMatrixPerElementOperationsNV

Reserved.

5 + variable	5369	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Matrix</i>	<id> <i>Func</i>	<id>, <id>, ... <i>Operands</i>
--------------	------	----------------------------	--------------------	-----------------------	---------------------	------------------------------------

3.3.10. Image Instructions

<div><div>OpSampledImage</div><div>Create a <a>sampld image, containing both a <a>sampler and an <a>image.</div><div><i>Result Type</i> must be <a>OpTypeSampledImage.</div><div><i>Image</i> is an object whose type is an <a>OpTypeImage, whose <i>Sampled</i> operand is 0 or 1, and whose <i>Dim</i> operand is not SubpassData. Additionally, <a>starting with version 1.6, the <i>Dim</i> operand must not be Buffer.</div><div><i>Sampler</i> must be an object whose type is <a>OpTypeSampler.</div><div>If the client API does not ignore <i>Depth</i>, the <i>Image Type</i> operand of the <i>Result Type</i> must be the same as the type of <i>Image</i>. Otherwise, the type of <i>Image</i> and the <i>Image Type</i> operand of the <i>Result Type</i> must be two <a>OpTypeImage with all operands matching each other except for <i>Depth</i> which can be different.</div></div>					
5	86	<id> Result Type	Result <id>	<id> Image	<id> Sampler

<div>OpImageSampleImplicitLod</div> <p>Sample an image with an implicit level of detail.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X'.</p> <p><i>Result Type</i> must be a vector of four components of floating-point type or integer type. Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid).</p> <p><i>Sampled Image</i> must be an object whose type is OpTypeSampledImage. Its OpTypeImage must not have a <i>Dim</i> of Buffer. The <i>MS</i> operand of the underlying OpTypeImage must be 0.</p> <p><i>Coordinate</i> must be a scalar or vector of floating-point type. It contains (<i>u</i>[, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i>. It may be a vector larger than needed, but all unused components appear after all used components.</p> <p><i>Image Operands</i> encodes what operands follow, as per Image Operands.</p> <p>This instruction is only valid in the Fragment Execution Model. In addition, it consumes an implicit derivative that can be affected by code motion.</p>						Capability: Shader	
5 + variable	87	<id> Result Type	Result <id>	<id> Sampled Image	<id> Coordinate	Optional Image Operands	Optional <id>, <id>, ...

OpImageSampleExplicitLod

Sample an image using an explicit level of detail.

Result Type must be a vector of four components of *floating-point type* or *integer type*. Its components must be the same as *Sampled Type* of the underlying **OpTypeImage** (unless that underlying *Sampled Type* is **OpTypeVoid**).

Sampled Image must be an object whose type is **OpTypeSampledImage**. Its **OpTypeImage** must not have a *Dim* of **Buffer**. The *MS* operand of the underlying **OpTypeImage** must be 0.

Coordinate must be a scalar or vector of *floating-point type* or *integer type*. It contains (*u*[, *v*] ... [, *array layer*]) as needed by the definition of *Sampled Image*. Unless the **Kernel capability** is declared, it must be floating point. It may be a vector larger than needed, but all unused components appear after all used components.

Image Operands encodes what operands follow, as per **Image Operands**. Either **Lod** or **Grad** image operands must be present.

7 + variable	88	<id> Result Type	Result <id>	<id> Sampled Image	<id> Coordinate	Image Operands	<id>	Optional <id>, <id>, ...
-----------------	----	------------------------	----------------	--------------------------	--------------------	-------------------	------	--------------------------------

OpImageSampleDrefImplicitLod Sample an image doing depth-comparison with an implicit level of detail. An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar of integer type or floating-point type . It must be the same as <i>Sampled Type</i> of the underlying OpTypeImage . <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must not have a Dim of Buffer . The <i>MS</i> operand of the underlying OpTypeImage must be 0. <i>Coordinate</i> must be a scalar or vector of floating-point type . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . It may be a vector larger than needed, but all unused components appear after all used components. <i>D_{ref}</i> is the depth-comparison reference value. It must be a 32-bit floating-point type scalar. <i>Image Operands</i> encodes what operands follow, as per Image Operands . This instruction is only valid in the Fragment Execution Model . In addition, it consumes an implicit derivative that can be affected by code motion.							Capability: Shader	
6 + variable	89	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	Optional Image Operands	Optional <id>, <id>, ...

OpImageSampleDrefExplicitLod

Sample an image doing depth-comparison using an explicit level of detail.

Result Type must be a scalar of *integer type* or *floating-point type*. It must be the same as *Sampled Type* of the underlying **OpTypeImage**.

Sampled Image must be an object whose type is **OpTypeSampledImage**. Its **OpTypeImage** must not have a *Dim* of **Buffer**. The *MS* operand of the underlying **OpTypeImage** must be 0.

Coordinate must be a scalar or vector of *floating-point type*. It contains (*u*[, *v*] ... [, *array layer*]) as needed by the definition of *Sampled Image*. It may be a vector larger than needed, but all unused components appear after all used components.

D_{ref} is the depth-comparison reference value. It must be a 32-bit *floating-point type* scalar.

Image Operands encodes what operands follow, as per **Image Operands**. Either **Lod** or **Grad** image operands must be present.

Capability:
Shader

8 + variable	90	<id> Result Type	Result <id>	<id> Sampled Image	<id> Coordinat e	<id> D _{ref}	Image Operands	<id>	Optional <id>, <id>, ...
-----------------	----	------------------------	----------------	--------------------------	------------------------	--------------------------	-------------------	------	--------------------------------

OpImageSampleProjImplicitLod

Sample an image with with a project coordinate and an implicit level of detail.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [derivative group](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Result Type must be a vector of four components of [floating-point type](#) or [integer type](#). Its components must be the same as *Sampled Type* of the underlying [OpTypeImage](#) (unless that underlying *Sampled Type* is [OpTypeVoid](#)).

Sampled Image must be an object whose type is [OpTypeSampledImage](#). The *Dim* operand of the underlying [OpTypeImage](#) must be **1D**, **2D**, **3D**, or **Rect**, and the *Arrayed* and *MS* operands must be 0.

Coordinate must be a vector of [floating-point type](#). It contains (*u* [, *v*] [, *w*], *q*), as needed by the definition of *Sampled Image*, with the *q* component consumed for the projective division. That is, the actual sample coordinate is (*u*/*q* [, *v*/*q*] [, *w*/*q*]), as needed by the definition of *Sampled Image*. It may be a vector larger than needed, but all unused components appear after all used components.

Image Operands encodes what operands follow, as per [Image Operands](#).

This instruction is only valid in the **Fragment Execution Model**. In addition, it consumes an implicit derivative that can be affected by code motion.

Capability:
Shader

5 + variable	91	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	Optional Image Operands	Optional <id>, <id>, ...
--------------	----	----------------------------	-----------------------------------	------------------------------	---------------------------	--	-----------------------------

OpImageSampleProjExplicitLod Sample an image with a project coordinate using an explicit level of detail. <i>Result Type</i> must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . The <i>Dim</i> operand of the underlying OpTypeImage must be 1D , 2D , 3D , or Rect , and the <i>Arrayed</i> and <i>MS</i> operands must be 0. <i>Coordinate</i> must be a vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] [, <i>w</i>], <i>q</i>), as needed by the definition of <i>Sampled Image</i> , with the <i>q</i> component consumed for the projective division. That is, the actual sample coordinate is (<i>u</i> / <i>q</i> [, <i>v</i> / <i>q</i>] [, <i>w</i> / <i>q</i>]), as needed by the definition of <i>Sampled Image</i> . It may be a vector larger than needed, but all unused components appear after all used components. <i>Image Operands</i> encodes what operands follow, as per <i>Image Operands</i> . Either Lod or Grad image operands must be present.							Capability: Shader
7 + variable	92	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<i>Image Operands</i>	<id> Optional <id>, <id>, ...

OpImageSampleProjDrefImplicitLod

Capability:
Shader

Sample an image with a project coordinate, doing depth-comparison, with an implicit level of detail.

An invocation will not execute a [dynamic instance](#) of this instruction (X') until all invocations in its [derivative group](#) have executed all [dynamic instances](#) that are [program-ordered before \$X'\$](#) .

Result Type must be a scalar of [integer type](#) or [floating-point type](#). It must be the same as *Sampled Type* of the underlying [OpTypeImage](#).

Sampled Image must be an object whose type is [OpTypeSampledImage](#). The *Dim* operand of the underlying [OpTypeImage](#) must be **1D**, **2D**, **3D**, or **Rect**, and the *Arrayed* and *MS* operands must be 0.

Coordinate must be a vector of [floating-point type](#). It contains $(u[, v][, w], q)$, as needed by the definition of *Sampled Image*, with the q component consumed for the projective division. That is, the actual sample coordinate is $(u/q[, v/q][, w/q])$, as needed by the definition of *Sampled Image*. It may be a vector larger than needed, but all unused components appear after all used components.

D_{ref}/q is the depth-comparison reference value. D_{ref} must be a 32-bit [floating-point type](#) scalar.

Image Operands encodes what operands follow, as per [Image Operands](#).

This instruction is only valid in the **Fragment Execution Model**. In addition, it consumes an implicit derivative that can be affected by code motion.

6 + variable	93	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> D_{ref}	Optional Image Operands	Optional <id>, <id>, ...
-----------------	----	--------------------------------	--------------------------------	----------------------------------	---------------------------	-------------------	--	--------------------------------

OpImageSampleProjDrefExplicitLod Sample an image with a project coordinate, doing depth-comparison, using an explicit level of detail. <i>Result Type</i> must be a scalar of <i>integer type</i> or <i>floating-point type</i> . It must be the same as <i>Sampled Type</i> of the underlying OpTypeImage . <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . The <i>Dim</i> operand of the underlying OpTypeImage must be 1D , 2D , 3D , or Rect , and the <i>Arrayed</i> and <i>MS</i> operands must be 0. <i>Coordinate</i> must be a vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] [, <i>w</i>], <i>q</i>), as needed by the definition of <i>Sampled Image</i> , with the <i>q</i> component consumed for the projective division. That is, the actual sample coordinate is (<i>u</i> / <i>q</i> [, <i>v</i> / <i>q</i>] [, <i>w</i> / <i>q</i>]), as needed by the definition of <i>Sampled Image</i> . It may be a vector larger than needed, but all unused components appear after all used components. <i>D_{ref} / q</i> is the depth-comparison reference value. <i>D_{ref}</i> must be a 32-bit <i>floating-point type</i> scalar. <i>Image Operands</i> encodes what operands follow, as per Image Operands . Either Lod or Grad image operands must be present.							Capability: Shader		
8 + variable	94	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	<i>Image Operands</i>	<id>	Optional <id>, <id>, ...

OpImageFetch Fetch a single texel from an image whose <i>Sampled</i> operand is 1. <i>Result Type</i> must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must not be Cube , and its <i>Sampled</i> operand must be 1. <i>Coordinate</i> must be a scalar or vector of <i>integer type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>Image Operands</i> encodes what operands follow, as per Image Operands .									
5 + variable	95	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>	<id> <i>Coordinate</i>		Optional <i>Image Operands</i>		Optional <id>, <id>, ...

OpImageGather Gathers the requested component from four texels. <i>Result Type</i> must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). It has one component per gathered texel. <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must have a <i>Dim</i> of 2D , Cube , or Rect . The <i>MS</i> operand of the underlying OpTypeImage must be 0. <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>Component</i> is the component number gathered from all four texels. It must be a 32-bit <i>integer type</i> scalar. Behavior is undefined if its value is not 0, 1, 2 or 3. <i>Image Operands</i> encodes what operands follow, as per <i>Image Operands</i> .							Capability: Shader	
6 + variable	96	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>Component</i> <i>t</i>	Optional <i>Image Operands</i>	Optional <id>, <id>, ...

OpImageDrefGather Gathers the requested depth-comparison from four texels. <i>Result Type</i> must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). It has one component per gathered texel. <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must have a <i>Dim</i> of 2D , Cube , or Rect . The <i>MS</i> operand of the underlying OpTypeImage must be 0. <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>D_{ref}</i> is the depth-comparison reference value. It must be a 32-bit <i>floating-point type</i> scalar. <i>Image Operands</i> encodes what operands follow, as per <i>Image Operands</i> .							Capability: Shader	
6 + variable	97	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	Optional <i>Image Operands</i>	Optional <id>, <id>, ...

OpImageRead

Read a texel from an [image](#) without a [sampler](#).

Result Type must be a scalar or vector of [floating-point type](#) or [integer type](#). It must be a scalar or vector with component type the same as *Sampled Type* of the [OpTypeImage](#) (unless that *Sampled Type* is **OpTypeVoid**).

Image must be an object whose type is [OpTypeImage](#) with a *Sampled* operand of 0 or 2. If the *Arrayed* operand is 1, then additional capabilities may be required; e.g., **ImageCubeArray**, or **ImageMSArray**.

Coordinate must be a scalar or vector of [floating-point type](#) or [integer type](#). It contains non-normalized texel coordinates (u , v) ... [i , *array layer*] as needed by the definition of *Image*. See the client API specification for handling of coordinates outside the image.

If the *Image Dim* operand is **SubpassData**, *Coordinate* is relative to the current fragment location. See the client API specification for more detail on how these coordinates are applied.

If the *Image Dim* operand is not **SubpassData**, the *Image Format* must not be **Unknown**, unless the **StorageImageReadWithoutFormat** or **Kernel Capabilities** were declared.

Image Operands encodes what operands follow, as per [Image Operands](#).

5 + variable	98	<id> <i>Result Type</i>	Result <id>	<id> <i>Image</i>	<id> <i>Coordinate</i>	Optional Image Operands	Optional <id>, <id>, ...
--------------	----	----------------------------	-----------------------------------	----------------------	---------------------------	--	-----------------------------

OpImageWrite

Write a texel to an [image](#) without a [sampler](#).

Image must be an object whose type is [OpTypeImage](#) with a *Sampled* operand of 0 or 2. If the *Arrayed* operand is 1, then additional capabilities may be required; e.g., **ImageCubeArray**, or **ImageMSArray**. Its *Dim* operand must not be **SubpassData**.

Coordinate must be a scalar or vector of [floating-point type](#) or [integer type](#). It contains non-normalized texel coordinates (u [, v] ... [, *array layer*]) as needed by the definition of *Image*. See the client API specification for handling of coordinates outside the image.

Texel is the data to write. It must be a scalar or vector with component type the same as *Sampled Type* of the [OpTypeImage](#) (unless that *Sampled Type* is **OpTypeVoid**).

The [Image Format](#) must not be **Unknown**, unless the **StorageImageWriteWithoutFormat** or **Kernel Capabilities** were declared.

Image Operands encodes what operands follow, as per [Image Operands](#).

4 + variable	99	<id> <i>Image</i>	<id> <i>Coordinate</i>	<id> <i>Texel</i>	Optional Image Operands	Optional <id>, <id>, ...
--------------	----	----------------------	---------------------------	----------------------	--	-----------------------------

OpImage

Extract the image from a sampled image.

Result Type must be [OpTypeImage](#).

Sampled Image must have type [OpTypeSampledImage](#) whose *Image Type* is the same as *Result Type*.

4	100	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>
---	-----	----------------------------	-----------------------------------	------------------------------

OpImageQueryFormat

Query the image format of an image created with an **Unknown** [Image Format](#).

Result Type must be a scalar [integer type](#). The resulting value is an enumerant from [Image Channel Data Type](#).

Image must be an object whose type is [OpTypeImage](#).

[Capability:](#)
Kernel

4	101	<id> <i>Result Type</i>	Result <id>	<id> <i>Image</i>
---	-----	----------------------------	-----------------------------------	----------------------

OpImageQueryOrder Query the channel order of an image created with an Unknown Image Format . <i>Result Type</i> must be a scalar integer type . The resulting value is an enumerant from Image Channel Order . <i>Image</i> must be an object whose type is OpTypeImage .				Capability: Kernel
4	102	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>

OpImageQuerySizeLod Query the dimensions of <i>Image</i> for mipmap level for <i>Level of Detail</i> . <i>Result Type</i> must be an integer type scalar or vector. The number of components must be 1 for the 1D dimensionality , 2 for the 2D and Cube dimensionalities , 3 for the 3D dimensionality , plus 1 more if the image type is arrayed. This vector is filled in with (<i>width</i> [, <i>height</i>] [, <i>depth</i>] [, <i>elements</i>]) where <i>elements</i> is the number of layers in an image array, or the number of cubes in a cube-map array. <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must be one of 1D , 2D , 3D , or Cube , and its <i>MS</i> must be 0. See OpImageQuerySize for querying image types without level of detail. See the client API specification for additional image type restrictions. <i>Level of Detail</i> is used to compute which mipmap level to query and must be a 32-bit integer type scalar.				Capability: Kernel, ImageQuery	
5	103	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>	<id> <i>Level of Detail</i>

OpImageQuerySize Query the dimensions of <i>Image</i> , with no level of detail. <i>Result Type</i> must be an integer type scalar or vector. The number of components must be: 1 for the 1D and Buffer dimensionalities , 2 for the 2D , Cube , and Rect dimensionalities , 3 for the 3D dimensionality , plus 1 more if the image type is arrayed. This vector is filled in with (<i>width</i> [, <i>height</i>] [, <i>elements</i>]) where <i>elements</i> is the number of layers in an image array or the number of cubes in a cube-map array. <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must be one of those listed under <i>Result Type</i> , above. Additionally, if its <i>Dim</i> is 1D , 2D , 3D , or Cube , it must also have either an <i>MS</i> of 1 or a <i>Sampled</i> of 0 or 2. There is no implicit level-of-detail consumed by this instruction. See OpImageQuerySizeLod for querying images having level of detail. See the client API specification for additional image type restrictions.				
Capability: Kernel, ImageQuery				
4	104	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>

OpImageQueryLod Query the mipmap level and the level of detail for a hypothetical sampling of <i>Image</i> at <i>Coordinate</i> using an implicit level of detail. An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a two-component floating-point type vector. The first component of the result contains the mipmap array layer. The second component of the result contains the implicit level of detail relative to the base level. <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage Dim operand must be one of 1D , 2D , 3D , or Cube , and its <i>MS</i> must be 0. <i>Coordinate</i> must be a scalar or vector of floating-point type . It contains (<i>u</i> [, <i>v</i>] ...) as needed by the definition of <i>Sampled Image</i> , not including any array layer index. This instruction is only valid in the Fragment Execution Model . In addition, it consumes an implicit derivative that can be affected by code motion.				Capability: ImageQuery	
5	105	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>

OpImageQueryLevels Query the number of mipmap levels accessible through <i>Image</i> . <i>Result Type</i> must be a scalar integer type . The result is the number of mipmap levels, as specified by the client API. <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must be one of 1D , 2D , 3D , or Cube , and its <i>MS</i> must be 0. See the client API specification for additional image type restrictions.				Capability: Kernel, ImageQuery	
4	106	<id> <i>Result Type</i>	Result <id>	<id> <i>Image</i>	

OpImageQuerySamples Query the number of samples available per texel fetch in a multisample image. <i>Result Type</i> must be a scalar integer type . The result is the number of samples. <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must be one of 2D and <i>MS</i> of 1.				Capability: Kernel, ImageQuery
4	107	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>

<div>OpImageSparseSampleImplicitLod</div> <div>Sample a sparse image with an implicit level of detail.</div> <div>An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X.</div> <div><i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an integer type scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident. The second member must be a vector of four components of floating-point type or integer type. Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid).</div> <div><i>Sampled Image</i> must be an object whose type is OpTypeSampledImage. Its OpTypeImage must not have a <i>Dim</i> of Buffer. The <i>MS</i> operand of the underlying OpTypeImage must be 0.</div> <div><i>Coordinate</i> must be a scalar or vector of floating-point type. It contains (<i>u</i>[, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i>. It may be a vector larger than needed, but all unused components appear after all used components.</div> <div><i>Image Operands</i> encodes what operands follow, as per Image Operands.</div> <div>This instruction is only valid in the Fragment Execution Model. In addition, it consumes an implicit derivative that can be affected by code motion.</div>						<div>Capability:</div> <div>SparseResidency</div>	
5 + variable	305	<id> Result Type	Result <id>	<id> Sampled Image	<id> Coordinate	Optional Image Operands	Optional <id>, <id>, ...

OpImageSparseSampleExplicitLod Sample a sparse image using an explicit level of detail. <i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident . The second member must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must not have a <i>Dim</i> of Buffer . The <i>MS</i> operand of the underlying OpTypeImage must be 0. <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> or <i>integer type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . Unless the Kernel capability is declared, it must be floating point. It may be a vector larger than needed, but all unused components appear after all used components. <i>Image Operands</i> encodes what operands follow, as per Image Operands . Either Lod or Grad image operands must be present.							Capability: SparseResidency
7 + variable	306	<id> Result Type	<i>Result</i> <id>	<id> Sampled Image	<id> Coordinate	<i>Image</i> Operands	<id> Optional <id>, <id>, ...

OpImageSparseSampleDrefImplicitLod

Capability:
SparseResidency

Sample a sparse image doing depth-comparison with an implicit level of detail.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [derivative group](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Result Type must be an [OpTypeStruct](#) with two members. The first member's type must be an [integer type](#) scalar. It holds a *Residency Code* that can be passed to [OpImageSparseTexelsResident](#). The second member must be a scalar of [integer type](#) or [floating-point type](#). It must be the same as *Sampled Type* of the underlying [OpTypeImage](#).

Sampled Image must be an object whose type is [OpTypeSampledImage](#). Its [OpTypeImage](#) must not have a *Dim* of **Buffer**. The *MS* operand of the underlying [OpTypeImage](#) must be 0.

Coordinate must be a scalar or vector of [floating-point type](#). It contains (*u*[, *v*] ... [, *array layer*]) as needed by the definition of *Sampled Image*. It may be a vector larger than needed, but all unused components appear after all used components.

D_{ref} is the depth-comparison reference value. It must be a 32-bit [floating-point type](#) scalar.

Image Operands encodes what operands follow, as per [Image Operands](#).

This instruction is only valid in the **Fragment Execution Model**. In addition, it consumes an implicit derivative that can be affected by code motion.

6 + variable	307	<id> <i>Result Type</i>	Result <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	Optional Image Operands	Optional <id>, <id>, ...
-----------------	-----	--------------------------------	--------------------------------	----------------------------------	---------------------------	--------------------------------	--	--------------------------------

OpImageSparseSampleDrefExplicitLod Sample a sparse image doing depth-comparison using an explicit level of detail. <i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident . The second member must be a scalar of <i>integer type</i> or <i>floating-point type</i> . It must be the same as <i>Sampled Type</i> of the underlying OpTypeImage . <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must not have a <i>Dim</i> of Buffer . The <i>MS</i> operand of the underlying OpTypeImage must be 0. <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . It may be a vector larger than needed, but all unused components appear after all used components. <i>D_{ref}</i> is the depth-comparison reference value. It must be a 32-bit <i>floating-point type</i> scalar. <i>Image Operands</i> encodes what operands follow, as per Image Operands . Either Lod or Grad image operands must be present.							Capability: SparseResidency		
8 + variable	308	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	<i>Image Operands</i>	<id>	Optional <id>, <id>, ...

OpImageSparseFetch Fetch a single texel from a sampled sparse image whose <i>Sampled</i> operand is 1. <i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident . The second member must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). <i>Image</i> must be an object whose type is OpTypeImage . Its <i>Dim</i> operand must not be Cube . <i>Coordinate</i> must be a scalar or vector of <i>integer type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>Image Operands</i> encodes what operands follow, as per Image Operands .							Capability: SparseResidency		
5 + variable	313	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Image</i>	<id> <i>Coordinate</i>		Optional <i>Image Operands</i>		Optional <id>, <id>, ...

OpImageSparseGather Gathers the requested component from four texels of a sparse image. <i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident . The second member must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). It has one component per gathered texel. <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must have a <i>Dim</i> of 2D , Cube , or Rect . <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>Component</i> is the component number gathered from all four texels. It must be a 32-bit <i>integer type</i> scalar. Behavior is undefined if its value is not 0, 1, 2 or 3. <i>Image Operands</i> encodes what operands follow, as per Image Operands .							Capability: SparseResidency	
6 + variable	314	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>Component</i> <i>t</i>	Optional <i>Image Operands</i>	Optional <id>, <id>, ...

OpImageSparseDrefGather Gathers the requested depth-comparison from four texels of a sparse image. <i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident . The second member must be a vector of four components of <i>floating-point type</i> or <i>integer type</i> . Its components must be the same as <i>Sampled Type</i> of the underlying OpTypeImage (unless that underlying <i>Sampled Type</i> is OpTypeVoid). It has one component per gathered texel. <i>Sampled Image</i> must be an object whose type is OpTypeSampledImage . Its OpTypeImage must have a <i>Dim</i> of 2D , Cube , or Rect . <i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> . It contains (<i>u</i> [, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Sampled Image</i> . <i>D_{ref}</i> is the depth-comparison reference value. It must be a 32-bit <i>floating-point type</i> scalar. <i>Image Operands</i> encodes what operands follow, as per Image Operands .							Capability: SparseResidency	
6 + variable	315	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Sampled Image</i>	<id> <i>Coordinate</i>	<id> <i>D_{ref}</i>	Optional <i>Image Operands</i>	Optional <id>, <id>, ...

OpImageSparseTexelsResident Translates a <i>Resident Code</i> into a Boolean. Result is false if any of the texels were in uncommitted texture memory, and true otherwise. <i>Result Type</i> must be a <i>Boolean type</i> scalar. <i>Resident Code</i> is a value from an OpImageSparse... instruction that results in a resident code.				Capability: SparseResidency
4	316	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Resident Code</i>

OpImageSparseRead						Capability: SparseResidency	
Read a texel from a sparse image without a sampler .							
<p><i>Result Type</i> must be an OpTypeStruct with two members. The first member's type must be an <i>integer type</i> scalar. It holds a <i>Residency Code</i> that can be passed to OpImageSparseTexelsResident. The second member must be a scalar or vector of <i>floating-point type</i> or <i>integer type</i>. It must be a scalar or vector with component type the same as <i>Sampled Type</i> of the OpTypeImage (unless that <i>Sampled Type</i> is OpTypeVoid).</p> <p><i>Image</i> must be an object whose type is OpTypeImage with a <i>Sampled</i> operand of 2.</p> <p><i>Coordinate</i> must be a scalar or vector of <i>floating-point type</i> or <i>integer type</i>. It contains non-normalized texel coordinates (<i>u</i>[, <i>v</i>] ... [, <i>array layer</i>]) as needed by the definition of <i>Image</i>. See the client API specification for handling of coordinates outside the image.</p> <p>The <i>Image Dim</i> operand must not be SubpassData. The <i>Image Format</i> must not be Unknown unless the StorageImageReadWithoutFormat or Kernel Capabilities were declared.</p> <p><i>Image Operands</i> encodes what operands follow, as per Image Operands.</p>							
5 + variable	320	<id> Result Type	Result <id>	<id> Image	<id> Coordinate	Optional Image Operands	Optional <id>, <id>, ...

OpColorAttachmentReadEXT Reserved.				Capability: TileImageColorReadAccessEXT Reserved.	
4 + variable	4160	<id> Result Type	Result <id>	<id> Attachment	Optional <id> Sample

OpDepthAttachmentReadEXT Reserved.				Capability: TileImageDepthReadAccessEXT Reserved.
3 + variable	4161	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	Optional <i><id></i> <i>Sample</i>

OpStencilAttachmentReadEXT Reserved.				Capability: TileImageStencilReadAccessEXT Reserved.
3 + variable	4162	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	Optional <i><id></i> <i>Sample</i>

OpImageSampleWeightedQCOM					Capability: TextureSampleWeightedQCOM	
Reserved.					Reserved.	
6	4480	<id> Result Type	Result <id>	<id> Texture	<id> Coordinates	<id> Weights

OpImageBoxFilterQCOM					Capability: TextureBoxFilterQCOM	
Reserved.					Reserved.	
6	4481	<id> Result Type	Result <id>	<id> Texture	<id> Coordinates	<id> Box Size

OpImageBlockMatchSSDQCOM							Capability: TextureBlockMatchQCOM	
Reserved.							Reserved.	
8	4482	<id> Result Type	Result <id>	<id> Target	<id> Target Coordinates	<id> Reference	<id> Reference Coordinates	<id> Block Size

OpImageBlockMatchSADQCOM Reserved.							Capability: TextureBlockMatchQCOM Reserved.	
8	4483	<id> Result Type	Result <id>	<id> Target	<id> Target Coordinates	<id> Reference	<id> Reference Coordinates	<id> Block Size

OpImageBlockMatchWindowSSDQCOM Reserved.							Capability: TextureBlockMatch2QCOM Reserved.	
8	4500	<id> Result Type	Result <id>	<id> Target Sampled Image	<id> Target Coordinates	<id> Reference Sampled Image	<id> Reference Coordinates	<id> Block Size

OpImageBlockMatchWindowSADQCOM Reserved.							Capability: TextureBlockMatch2QCOM Reserved.	
8	4501	<id> Result Type	Result <id>	<id> Target Sampled Image	<id> Target Coordinates	<id> Reference Sampled Image	<id> Reference Coordinates	<id> Block Size

OpImageBlockMatchGatherSSDQCOM Reserved.							Capability: TextureBlockMatch2QCOM Reserved.	
8	4502	<id> Result Type	Result <id>	<id> Target Sampled Image	<id> Target Coordinates	<id> Reference Sampled Image	<id> Reference Coordinates	<id> Block Size

OpImageBlockMatchGatherSADQCOM Reserved.							Capability: TextureBlockMatch2QCOM Reserved.	
8	4503	<id> Result Type	Result <id>	<id> Target Sampled Image	<id> Target Coordinates	<id> Reference Sampled Image	<id> Reference Coordinates	<id> Block Size

OpImageSampleFootprintNV Reserved.							Capability: ImageFootprintNV Reserved.		
7 + variable	5283	<id> Result Type	Result <id>	<id> Sampled Image	<id> Coordinate	<id> Granularity	<id> Coarse	Optional Image Operands	Optional <id>, <id>, ...

OpConvertHandleToImageINTEL Reserved.					Capability: BindlessImagesINTEL Reserved.				
4	6529	<id> Result Type	Result <id>		<id> Operand				

OpConvertHandleToSamplerINTEL Reserved.					Capability: BindlessImagesINTEL Reserved.				
4	6530	<id> Result Type	Result <id>		<id> Operand				

OpConvertHandleToSampledImageINTEL Reserved.					Capability: BindlessImagesINTEL Reserved.				
4	6531	<id> Result Type	Result <id>		<id> Operand				

3.3.11. Conversion Instructions

OpConvertFToU

Convert value numerically from floating point to unsigned integer, with round toward 0.0.

Result Type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0. Behavior is undefined if *Result Type* is not wide enough to hold the converted value.

Float Value must be a scalar or vector of *floating-point type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	109	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Float Value</i>
---	-----	----------------------------	--------------------	----------------------------

OpConvertFToS

Convert value numerically from floating point to signed integer, with round toward 0.0.

Result Type must be a scalar or vector of *integer type*. Behavior is undefined if *Result Type* is not wide enough to hold the converted value.

Float Value must be a scalar or vector of *floating-point type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	110	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Float Value</i>
---	-----	----------------------------	--------------------	----------------------------

OpConvertSToF

Convert value numerically from signed integer to floating point.

Result Type must be a scalar or vector of *floating-point type*.

Signed Value must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	111	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Signed Value</i>
---	-----	----------------------------	--------------------	-----------------------------

OpConvertUToF

Convert value numerically from unsigned integer to floating point.

Result Type must be a scalar or vector of *floating-point type*.

Unsigned Value must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	112	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Unsigned Value</i>
---	-----	----------------------------	--------------------	-------------------------------

OpUConvert

Convert unsigned width. This is either a truncate or a zero extend.

Result Type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

Unsigned Value must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*. The component width must not equal the component width in *Result Type*.

Results are computed per component.

4	113	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Unsigned Value</i>
---	-----	----------------------------	--------------------	-------------------------------

OpSConvert

Convert signed width. This is either a truncate or a sign extend.

Result Type must be a scalar or vector of *integer type*.

Signed Value must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*. The component width must not equal the component width in *Result Type*.

Results are computed per component.

4	114	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Signed Value</i>
---	-----	----------------------------	--------------------	-----------------------------

OpFConvert Convert value numerically from one floating-point width to another width. <i>Result Type</i> must be a scalar or vector of <i>floating-point type</i> . <i>Float Value</i> must be a scalar or vector of <i>floating-point type</i> . It must have the same number of components as <i>Result Type</i> . The component type must not equal the component type in <i>Result Type</i> . Results are computed per component.				
4	115	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Float Value</i>

OpQuantizeToF16 Quantize a floating-point value to what is expressible by a 16-bit floating-point value. <i>Result Type</i> must be a scalar or vector of <i>floating-point type</i> . The component width must be 32 bits and must not have a <i>Floating Point Encoding</i> operand. <i>Value</i> is the value to quantize. The type of <i>Value</i> must be the same as <i>Result Type</i> . If <i>Value</i> is an infinity, the result is the same infinity. If <i>Value</i> is a NaN, the result is a NaN, but not necessarily the same NaN. If <i>Value</i> is positive with a magnitude too large to represent as a 16-bit floating-point value, the result is positive infinity. If <i>Value</i> is negative with a magnitude too large to represent as a 16-bit floating-point value, the result is negative infinity. If the magnitude of <i>Value</i> is too small to represent as a normalized 16-bit floating-point value, the result must be either +0 or -0. The RelaxedPrecision <i>Decoration</i> has no effect on this instruction. Results are computed per component.				Capability: Shader
4	116	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Value</i>

OpConvertPtrToU Bit pattern-preserving conversion of a pointer to an unsigned scalar integer of possibly different bit width. <i>Result Type</i> must be a scalar of <i>integer type</i> , whose <i>Signedness</i> operand is 0. <i>Pointer</i> must be a <i>physical pointer type</i> . If the bit width of <i>Pointer</i> is smaller than that of <i>Result Type</i> , the conversion zero extends <i>Pointer</i> . If the bit width of <i>Pointer</i> is larger than that of <i>Result Type</i> , the conversion truncates <i>Pointer</i> . For same bit width <i>Pointer</i> and <i>Result Type</i> , this is the same as OpBitcast .				Capability: Addresses, PhysicalStorageBuffer Addresses
4	117	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pointer</i>

OpSatConvertSToU Convert a signed integer to unsigned integer. Converted values outside the representable range of <i>Result Type</i> are clamped to the nearest representable value of <i>Result Type</i> . <i>Result Type</i> must be a scalar or vector of <i>integer type</i> . <i>Signed Value</i> must be a scalar or vector of <i>integer type</i> . It must have the same number of components as <i>Result Type</i> . Results are computed per component.				Capability: Kernel
4	118	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Signed Value</i>

OpSatConvertUToS Convert an unsigned integer to signed integer. Converted values outside the representable range of <i>Result Type</i> are clamped to the nearest representable value of <i>Result Type</i> . <i>Result Type</i> must be a scalar or vector of <i>integer type</i> . <i>Unsigned Value</i> must be a scalar or vector of <i>integer type</i> . It must have the same number of components as <i>Result Type</i> . Results are computed per component.				Capability: Kernel
4	119	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Unsigned Value</i>

OpConvertUToPtr Bit pattern-preserving conversion of an unsigned scalar integer to a pointer. <i>Result Type</i> must be a physical pointer type . <i>Integer Value</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0. If the bit width of <i>Integer Value</i> is smaller than that of <i>Result Type</i> , the conversion zero extends <i>Integer Value</i> . If the bit width of <i>Integer Value</i> is larger than that of <i>Result Type</i> , the conversion truncates <i>Integer Value</i> . For same-width <i>Integer Value</i> and <i>Result Type</i> , this is the same as OpBitcast . Behavior is undefined if the storage class of <i>Result Type</i> does not match the one used by the operation that produced the value of <i>Integer Value</i> .				Capability: Addresses, PhysicalStorageBuffer Addresses
4	120	<id> <i>Result Type</i>	Result <id>	<id> <i>Integer Value</i>

OpPtrCastToGeneric Convert a pointer's Storage Class to Generic . <i>Result Type</i> must be an OpTypePointer . Its Storage Class must be Generic . <i>Pointer</i> must point to the Workgroup , CrossWorkgroup , or Function Storage Class . <i>Result Type</i> and <i>Pointer</i> must point to the same type.				Capability: Kernel
4	121	<id> <i>Result Type</i>	Result <id>	<id> <i>Pointer</i>

OpGenericCastToPtr Convert a pointer's Storage Class to a non- Generic class. <i>Result Type</i> must be an OpTypePointer . Its Storage Class must be Workgroup , CrossWorkgroup , or Function . <i>Pointer</i> must point to the Generic Storage Class . <i>Result Type</i> and <i>Pointer</i> must point to the same type.				Capability: Kernel
4	122	<id> <i>Result Type</i>	Result <id>	<id> <i>Pointer</i>

OpGenericCastToPtrExplicit Attempts to explicitly convert <i>Pointer</i> to <i>Storage</i> storage-class pointer value. <i>Result Type</i> must be an OpTypePointer . Its <i>Storage Class</i> must be <i>Storage</i> . <i>Pointer</i> must have a type of OpTypePointer whose <i>Type</i> is the same as the <i>Type</i> of <i>Result Type</i> . <i>Pointer</i> must point to the Generic Storage Class . If the cast fails, the instruction result is an OpConstantNull pointer in the <i>Storage Storage Class</i> . <i>Storage</i> must be one of the following literal values from <i>Storage Class</i> : Workgroup , CrossWorkgroup , or Function .				Capability: Kernel	
5	123	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Pointer</i>	<i>Storage Class</i> <i>Storage</i>

OpBitcast

Bit pattern-preserving type conversion.

Result Type must be an **OpTypePointer**, or a scalar or vector of *numerical-type*.

Operand must have a type of **OpTypePointer**, or a scalar or vector of *numerical-type*. It must be a different type than *Result Type*.

Before **version 1.5**: If either *Result Type* or *Operand* is a pointer, the other must be a pointer or an integer scalar.

Starting with **version 1.5**: If either *Result Type* or *Operand* is a pointer, the other must be a pointer, an integer scalar, or an integer vector.

If both *Result Type* and the type of *Operand* are pointers, they both must point into same *storage class*.

Behavior is undefined if the *storage class* of *Result Type* does not match the one used by the operation that produced the value of *Operand*.

If *Result Type* has the same number of components as *Operand*, they must also have the same component width, and results are computed per component.

If *Result Type* has a different number of components than *Operand*, the total number of bits in *Result Type* must equal the total number of bits in *Operand*. Let *L* be the type, either *Result Type* or *Operand*'s type, that has the larger number of components. Let *S* be the other type, with the smaller number of components. The number of components in *L* must be an integer multiple of the number of components in *S*. The first component (that is, the only or lowest-numbered component) of *S* maps to the first components of *L*, and so on, up to the last component of *S* mapping to the last components of *L*. Within this mapping, any single component of *S* (mapping to multiple components of *L*) maps its lower-ordered bits to the lower-numbered components of *L*.

4	124	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>
---	-----	----------------------------	--------------------	------------------------

OpBitCastArrayQCOM

Reserved.

Capability:
CooperativeMatrixConversionQCOM

Reserved.

4	4497	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Source Array</i>
---	------	----------------------------	--------------------	-----------------------------

OpCooperativeMatrixConvertNV Reserved.				Capability: CooperativeMatrixConversionsNV Reserved.
4	5293	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Matrix</i>

OpCooperativeMatrixTransposeNV Reserved.				Capability: CooperativeMatrixConversionsNV Reserved.
4	5390	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Matrix</i>

OpConvertFToBF16INTEL Reserved.				Capability: BFloat16ConversionINTEL Reserved.
4	6116	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Float Value</i>

OpConvertBF16ToFIntel Reserved.				Capability: BFloat16ConversionINTEL Reserved.
4	6117	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>BFloat16 Value</i>

OpRoundFToTF32INTEL Reserved.				Capability: TensorFloat32RoundingINTEL Reserved.
4	6426	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Float Value</i>

3.3.12. Composite Instructions

OpVectorExtractDynamic

Extract a single, dynamically selected, component of a vector.

Result Type must be a [scalar](#) type.

Vector must have a type [OpTypeVector](#) whose *Component Type* is *Result Type*.

Index must be a scalar [integer](#). It is interpreted as a 0-based index of which component of *Vector* to extract.

Behavior is undefined if *Index*'s value is less than zero or greater than or equal to the number of components in *Vector*.

5	77	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector</i>	<id> <i>Index</i>
---	----	----------------------------	-----------------------------------	-----------------------	----------------------

OpVectorInsertDynamic

Make a copy of a vector, with a single, variably selected, component modified.

Result Type must be an [OpTypeVector](#).

Vector must have the same type as *Result Type* and is the vector that the non-written components are copied from.

Component is the value supplied for the component selected by *Index*. It must have the same type as the type of components in *Result Type*.

Index must be a scalar [integer](#). It is interpreted as a 0-based index of which component to modify.

Behavior is undefined if *Index*'s value is less than zero or greater than or equal to the number of components in *Vector*.

6	78	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector</i>	<id> <i>Component</i>	<id> <i>Index</i>
---	----	----------------------------	-----------------------------------	-----------------------	--------------------------	----------------------

OpVectorShuffle

Select arbitrary components from two vectors to make a new vector.

Result Type must be an **OpTypeVector**. The number of components in *Result Type* must be the same as the number of *Component* operands.

Vector 1 and *Vector 2* must both have vector types, with the same *Component Type* as *Result Type*. They do not have to have the same number of components as *Result Type* or with each other. They are logically concatenated, forming a single vector with *Vector 1*'s components appearing before *Vector 2*'s. The components of this logical vector are logically numbered with a single consecutive set of numbers from 0 to $N - 1$, where N is the total number of components.

Components are these logical numbers (see above), selecting which of the logically numbered components form the result. Each component is an unsigned 32-bit integer. They can select the components in any order and can repeat components. The first component of the result is selected by the first *Component* operand, the second component of the result is selected by the second *Component* operand, etc. A *Component literal* may also be FFFFFFFF, which means the corresponding result component has no source and is undefined. All *Component literals* must either be FFFFFFFF or in $[0, N - 1]$ (*inclusive*).

Note: A vector “swizzle” can be done by using the vector for both *Vector* operands, or using an **OpUndef** for one of the *Vector* operands.

5 + variable	79	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	<i>Literal, Literal, ... Components</i>
--------------	----	----------------------------	--------------------	-------------------------	-------------------------	---

OpCompositeConstruct

Construct a new *composite* object from a set of constituent objects.

Result Type must be a *composite* type, whose top-level members/elements/components/columns have the same type as the types of the operands, with one exception. The exception is that for constructing a vector, the operands may also be vectors with the same component type as the *Result Type* component type. If constructing a vector, the total number of components in all the operands must equal the number of components in *Result Type*.

Constituents become members of a structure, or elements of an array, or components of a vector, or columns of a matrix. There must be exactly one *Constituent* for each top-level member/element/component/column of the result, with one exception. The exception is that for constructing a vector, a contiguous subset of the scalars consumed can be represented by a vector operand instead. The *Constituents* must appear in the order needed by the definition of the type of the result. If constructing a vector, there must be at least two *Constituent* operands.

3 + variable	80	<id> <i>Result Type</i>	<i>Result</i> <id>	<id>, <id>, ... <i>Constituents</i>
--------------	----	----------------------------	--------------------	--

OpCompositeExtract Extract a part of a <i>composite</i> object. <i>Result Type</i> must be the type of object selected by the last provided index. The instruction result is the extracted object. <i>Composite</i> is the composite to extract from. <i>Indexes</i> walk the type hierarchy, potentially down to component granularity, to select the part to extract. All indexes must be in bounds. All composite constituents use zero-based numbering, as described by their OpType... instruction. Each index is an unsigned 32-bit integer.						
4 + variable	81	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Composite</i>	<i>Literal, Literal, ... Indexes</i>	

OpCompositeInsert Make a copy of a <i>composite</i> object, while modifying one part of it. <i>Result Type</i> must be the same type as <i>Composite</i> . <i>Object</i> is the object to use as the modified part. <i>Composite</i> is the composite to copy all but the modified part from. <i>Indexes</i> walk the type hierarchy of <i>Composite</i> to the desired depth, potentially down to component granularity, to select the part to modify. All indexes must be in bounds. All composite constituents use zero-based numbering, as described by their OpType... instruction. The type of the part selected to modify must match the type of <i>Object</i> . Each index is an unsigned 32-bit integer.						
5 + variable	82	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Object</i>	<id> <i>Composite</i>	<i>Literal, Literal, ... Indexes</i>

OpCopyObject Make a copy of <i>Operand</i> . There are no pointer dereferences involved. <i>Result Type</i> must equal <i>Operand</i> type. <i>Result Type</i> can be any type except OpTypeVoid .						
4	83	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>		

OpTranspose Transpose a matrix. <i>Result Type</i> must be an OpTypeMatrix . <i>Matrix</i> must be an object of type OpTypeMatrix . The number of columns and the column size of <i>Matrix</i> must be the reverse of those in <i>Result Type</i> . The types of the scalar components in <i>Matrix</i> and <i>Result Type</i> must be the same. <i>Matrix</i> must have of type of OpTypeMatrix .				Capability: Matrix
4	84	<id> <i>Result Type</i>	Result <id>	<id> <i>Matrix</i>

OpCopyLogical Make a logical copy of <i>Operand</i> . There are no pointer dereferences involved. <i>Result Type</i> must not equal the type of <i>Operand</i> (see OpCopyObject), but <i>Result Type</i> must <i>logically match</i> the <i>Operand</i> type. <i>Logically match</i> is recursively defined by these three rules: 1. They must be either both be OpTypeArray or both be OpTypeStruct 2. If they are OpTypeArray : - they must have the same <i>Length</i> operand, and - their <i>Element Type</i> operands must be either the same or must <i>logically match</i> . 3. If they are OpTypeStruct : - they must have the same number of <i>Member type</i> , and - <i>Member N type</i> for the same <i>N</i> in the two types must be either the same or must <i>logically match</i> .				Missing before version 1.4.
4	400	<id> <i>Result Type</i>	Result <id>	<id> <i>Operand</i>

OpCompositeConstructReplicateEXT Reserved.				Capability: ReplicatedCompositesEXT Reserved.
4	4463	<id> <i>Result Type</i>	Result <id>	<id> <i>Value</i>

OpCompositeConstructCoopMatQCOM Reserved.				Capability: CooperativeMatrixConversionQCOM Reserved.
4	4540	<id> <i>Result Type</i>	Result <id>	<id> <i>Source Array</i>

OpCompositeExtractCoopMatQCOM Reserved.				Capability: CooperativeMatrixConversionQCOM Reserved.
4	4541	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> Source Cooperative Matrix

OpExtractSubArrayQCOM Reserved.				Capability: CooperativeMatrixConversionQCOM Reserved.	
5	4542	<id> Result Type	Result <id>	<id> Source Array	<id> index

OpCompositeConstructContinuedINTEL				Capability:
Reserved.				LongCompositesINTEL
				Reserved.
3 + variable	6096	<id> Result Type	Result <id>	<id>, <id>, ... Constituents

OpConditionalCopyObjectINTEL				Capability: SpecConditionalINTEL
Reserved.				Reserved.
3 + variable	6254	<id> Result Type	Result <id>	<id>, <id>, ... Condition 0, Operand 0, Condition 1, Operand 1, ...

3.3.13. Arithmetic Instructions

OpSNegate

Signed-integer subtract of *Operand* from zero.

Result Type must be a scalar or vector of *integer type*.

Operand's type must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*. The component width must equal the component width in *Result Type*.

Results are computed per component.

4	126	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>
---	-----	----------------------------	--------------------	------------------------

OpFNegate

Inverts the sign bit of *Operand*. (Note, however, that **OpFNegate** is still considered a floating-point instruction, and so is subject to the general floating-point rules regarding, for example, subnormals and NaN propagation).

Result Type must be a scalar or vector of *floating-point type*.

The type of *Operand* must be the same as *Result Type*.

Results are computed per component.

4	127	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>
---	-----	----------------------------	--------------------	------------------------

OpIAdd

Integer addition of *Operand 1* and *Operand 2*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

The resulting value equals the low-order *N* bits of the correct result *R*, where *N* is the component width and *R* is computed with enough precision to avoid overflow and underflow.

Results are computed per component.

5	128	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFAdd

Floating-point addition of *Operand 1* and *Operand 2*.

Result Type must be a scalar or vector of *floating-point type*.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component.

5	129	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpISub

Integer subtraction of *Operand 2* from *Operand 1*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

The resulting value equals the low-order *N* bits of the correct result *R*, where *N* is the component width and *R* is computed with enough precision to avoid overflow and underflow.

Results are computed per component.

5	130	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFSub

Floating-point subtraction of *Operand 2* from *Operand 1*.

Result Type must be a scalar or vector of *floating-point type*.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component.

5	131	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpIMul

Integer multiplication of *Operand 1* and *Operand 2*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

The resulting value equals the low-order *N* bits of the correct result *R*, where *N* is the component width and *R* is computed with enough precision to avoid overflow and underflow.

Results are computed per component.

5	132	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFMul

Floating-point multiplication of *Operand 1* and *Operand 2*.

Result Type must be a scalar or vector of *floating-point type*.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component.

5	133	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpUDiv

Unsigned-integer division of *Operand 1* divided by *Operand 2*.

Result Type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component. Behavior is undefined if *Operand 2* is 0.

5	134	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSDiv

Signed-integer division of *Operand 1* divided by *Operand 2*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

Results are computed per component. Behavior is undefined if *Operand 2* is 0. Behavior is undefined if *Operand 2* is -1 and *Operand 1* is the minimum representable value for the operands' type, causing signed overflow.

5	135	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFDiv

Floating-point division of *Operand 1* divided by *Operand 2*.

Result Type must be a scalar or vector of *floating-point type*.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component.

5	136	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpUMod

Unsigned modulo operation of *Operand 1* modulo *Operand 2*.

Result Type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

The types of *Operand 1* and *Operand 2* both must be the same as *Result Type*.

Results are computed per component. Behavior is undefined if *Operand 2* is 0.

5	137	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSRem

Signed remainder operation for the remainder whose sign matches the sign of *Operand 1*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

Results are computed per component. Behavior is undefined if *Operand 2* is 0. Behavior is undefined if *Operand 2* is -1 and *Operand 1* is the minimum representable value for the operands' type, causing signed overflow. Otherwise, the result is the *remainder* *r* of *Operand 1* divided by *Operand 2* where if $r \neq 0$, the sign of *r* is the same as the sign of *Operand 1*.

5	138	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSMod

Signed remainder operation for the remainder whose sign matches the sign of *Operand 2*.

Result Type must be a scalar or vector of *integer type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

Results are computed per component. Behavior is undefined if *Operand 2* is 0. Behavior is undefined if *Operand 2* is -1 and *Operand 1* is the minimum representable value for the operands' type, causing signed overflow. Otherwise, the result is the *remainder* *r* of *Operand 1* divided by *Operand 2* where if $r \neq 0$, the sign of *r* is the same as the sign of *Operand 2*.

5	139	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFRem The floating-point <i>remainder</i> whose sign matches the sign of <i>Operand 1</i> . <i>Result Type</i> must be a scalar or vector of <i>floating-point type</i> . The types of <i>Operand 1</i> and <i>Operand 2</i> both must be the same as <i>Result Type</i> . Results are computed per component. The resulting value is undefined if <i>Operand 2</i> is 0. Otherwise, the result is the <i>remainder</i> <i>r</i> of <i>Operand 1</i> divided by <i>Operand 2</i> where if $r \neq 0$, the sign of <i>r</i> is the same as the sign of <i>Operand 1</i> .					
5	140	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpFMod The floating-point <i>remainder</i> whose sign matches the sign of <i>Operand 2</i> . <i>Result Type</i> must be a scalar or vector of <i>floating-point type</i> . The types of <i>Operand 1</i> and <i>Operand 2</i> both must be the same as <i>Result Type</i> . Results are computed per component. The resulting value is undefined if <i>Operand 2</i> is 0. Otherwise, the result is the <i>remainder</i> <i>r</i> of <i>Operand 1</i> divided by <i>Operand 2</i> where if $r \neq 0$, the sign of <i>r</i> is the same as the sign of <i>Operand 2</i> .					
5	141	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpVectorTimesScalar Scale a floating-point vector. <i>Result Type</i> must be a vector of <i>floating-point type</i> . The type of <i>Vector</i> must be the same as <i>Result Type</i> . Each component of <i>Vector</i> is multiplied by <i>Scalar</i> . <i>Scalar</i> must have the same type as the <i>Component Type</i> in <i>Result Type</i> .					
5	142	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Vector</i>	<id> <i>Scalar</i>

OpMatrixTimesScalar Scale a floating-point matrix. <i>Result Type</i> must be an OpTypeMatrix whose <i>Column Type</i> is a vector of <i>floating-point type</i> . The type of <i>Matrix</i> must be the same as <i>Result Type</i> . Each component in each column in <i>Matrix</i> is multiplied by <i>Scalar</i> . <i>Scalar</i> must have the same type as the <i>Component Type</i> in <i>Result Type</i> .				Capability: Matrix	
5	143	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Matrix</i>	<id> <i>Scalar</i>

OpVectorTimesMatrix Linear-algebraic <i>Vector X Matrix</i> . <i>Result Type</i> must be a vector of <i>floating-point type</i> . <i>Vector</i> must be a vector with the same <i>Component Type</i> as the <i>Component Type</i> in <i>Result Type</i> . Its number of components must equal the number of components in each column in <i>Matrix</i> . <i>Matrix</i> must be a matrix with the same <i>Component Type</i> as the <i>Component Type</i> in <i>Result Type</i> . Its number of columns must equal the number of components in <i>Result Type</i> .				Capability: Matrix	
5	144	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Vector</i>	<id> <i>Matrix</i>

OpMatrixTimesVector Linear-algebraic <i>Matrix X Vector</i> . <i>Result Type</i> must be a vector of <i>floating-point type</i> . <i>Matrix</i> must be an OpTypeMatrix whose <i>Column Type</i> is <i>Result Type</i> . <i>Vector</i> must be a vector with the same <i>Component Type</i> as the <i>Component Type</i> in <i>Result Type</i> . Its number of components must equal the number of columns in <i>Matrix</i> .				Capability: Matrix	
5	145	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Matrix</i>	<id> <i>Vector</i>

OpMatrixTimesMatrix Linear-algebraic multiply of <i>LeftMatrix</i> X <i>RightMatrix</i> . <i>Result Type</i> must be an OpTypeMatrix whose <i>Column Type</i> is a vector of <i>floating-point type</i> . <i>LeftMatrix</i> must be a matrix whose <i>Column Type</i> is the same as the <i>Column Type</i> in <i>Result Type</i> . <i>RightMatrix</i> must be a matrix with the same <i>Component Type</i> as the <i>Component Type</i> in <i>Result Type</i> . Its number of columns must equal the number of columns in <i>Result Type</i> . Its columns must have the same number of components as the number of columns in <i>LeftMatrix</i> .				Capability: Matrix	
5	146	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>LeftMatrix</i>	<id> <i>RightMatrix</i>

OpOuterProduct Linear-algebraic outer product of <i>Vector 1</i> and <i>Vector 2</i> . <i>Result Type</i> must be an OpTypeMatrix whose <i>Column Type</i> is a vector of <i>floating-point type</i> . <i>Vector 1</i> must have the same type as the <i>Column Type</i> in <i>Result Type</i> . <i>Vector 2</i> must be a vector with the same <i>Component Type</i> as the <i>Component Type</i> in <i>Result Type</i> . Its number of components must equal the number of columns in <i>Result Type</i> .				Capability: Matrix	
5	147	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>

OpDot Dot product of <i>Vector 1</i> and <i>Vector 2</i> . <i>Result Type</i> must be a <i>floating-point type</i> scalar. <i>Vector 1</i> and <i>Vector 2</i> must be vectors of the same type, and their component type must be <i>Result Type</i> .					
5	148	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>

OpIAddCarry

Result is the unsigned integer addition of *Operand 1* and *Operand 2*, including its carry.

Result Type must be from **OpTypeStruct**. The struct must have two members, and the two members must be the same type. The member type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

Operand 1 and *Operand 2* must have the same type as the members of *Result Type*. These are consumed as unsigned integers.

Results are computed per component.

Member 0 of the result gets the low-order bits (full component width) of the addition.

Member 1 of the result gets the high-order (carry) bit of the result of the addition. That is, it gets the value 1 if the addition overflowed the component width, and 0 otherwise.

5	149	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpISubBorrow

Result is the unsigned integer subtraction of *Operand 2* from *Operand 1*, and what it needed to borrow.

Result Type must be from **OpTypeStruct**. The struct must have two members, and the two members must be the same type. The member type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

Operand 1 and *Operand 2* must have the same type as the members of *Result Type*. These are consumed as unsigned integers.

Results are computed per component.

Member 0 of the result gets the low-order bits (full component width) of the subtraction. That is, if *Operand 1* is larger than *Operand 2*, member 0 gets the full value of the subtraction; if *Operand 2* is larger than *Operand 1*, member 0 gets $2^w + \text{Operand 1} - \text{Operand 2}$, where w is the component width.

Member 1 of the result gets 0 if *Operand 1* \geq *Operand 2*, and gets 1 otherwise.

5	150	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpUMulExtended

Result is the full value of the unsigned integer multiplication of *Operand 1* and *Operand 2*.

Result Type must be from **OpTypeStruct**. The struct must have two members, and the two members must be the same type. The member type must be a scalar or vector of *integer type*, whose *Signedness* operand is 0.

Operand 1 and *Operand 2* must have the same type as the members of *Result Type*. These are consumed as unsigned integers.

Results are computed per component.

Member 0 of the result gets the low-order bits of the multiplication.

Member 1 of the result gets the high-order bits of the multiplication.

5	151	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSMulExtended

Result is the full value of the signed integer multiplication of *Operand 1* and *Operand 2*.

Result Type must be from **OpTypeStruct**. The struct must have two members, and the two members must be the same type. The member type must be a scalar or vector of *integer type*.

Operand 1 and *Operand 2* must have the same type as the members of *Result Type*. These are consumed as signed integers.

Results are computed per component.

Member 0 of the result gets the low-order bits of the multiplication.

Member 1 of the result gets the high-order bits of the multiplication.

5	152	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSDot (OpSDotKHR) Signed integer dot product of <i>Vector 1</i> and <i>Vector 2</i> . <i>Result Type</i> must be an integer type whose <i>Width</i> must be greater than or equal to that of the components of <i>Vector 1</i> and <i>Vector 2</i> . <i>Vector 1</i> and <i>Vector 2</i> must have the same type. <i>Vector 1</i> and <i>Vector 2</i> must be either 32-bit integers (enabled by the DotProductInput4x8BitPacked capability) or vectors of integer type (enabled by the DotProductInput4x8Bit or DotProductInputAll capability). When <i>Vector 1</i> and <i>Vector 2</i> are scalar integer types, <i>Packed Vector Format</i> must be specified to select how the integers are to be interpreted as vectors. All components of the input vectors are sign-extended to the bit width of the result's type. The sign-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. The resulting value will equal the low-order N bits of the correct result R, where N is the result width and R is computed with enough precision to avoid overflow and underflow.				Capability: DotProduct Missing before version 1.6.		
5 + variable	4450	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	Optional Packed Vector Format <i>Packed Vector Format</i>

OpUDot (OpUDotKHR)

Unsigned integer dot product of *Vector 1* and *Vector 2*.

Result Type must be an integer type with *Signedness* of 0 whose *Width* must be greater than or equal to that of the components of *Vector 1* and *Vector 2*.

Vector 1 and *Vector 2* must have the same type.

Vector 1 and *Vector 2* must be either 32-bit integers (enabled by the **DotProductInput4x8BitPacked** [capability](#)) or vectors of integer type with *Signedness* of 0 (enabled by the **DotProductInput4x8Bit** or **DotProductInputAll** [capability](#)).

When *Vector 1* and *Vector 2* are scalar integer types, *Packed Vector Format* must be specified to select how the integers are to be interpreted as vectors.

All components of the input vectors are zero-extended to the bit width of the result's type. The zero-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. The resulting value will equal the low-order N bits of the correct result R, where N is the result width and R is computed with enough precision to avoid overflow and underflow.

[Capability:](#)
DotProduct

[Missing before version 1.6.](#)

5 + variable	4451	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	Optional Packed Vector Format <i>Packed Vector Format</i>
--------------	------	----------------------------	-----------------------------------	-------------------------	-------------------------	---

OpSUDot (OpSUDotKHR) Mixed-signedness integer dot product of <i>Vector 1</i> and <i>Vector 2</i> . Components of <i>Vector 1</i> are treated as signed, components of <i>Vector 2</i> are treated as unsigned. <i>Result Type</i> must be an integer type whose <i>Width</i> must be greater than or equal to that of the components of <i>Vector 1</i> and <i>Vector 2</i> . <i>Vector 1</i> and <i>Vector 2</i> must be either 32-bit integers (enabled by the DotProductInput4x8BitPacked capability) or vectors of integer type with the same number of components and same component <i>Width</i> (enabled by the DotProductInput4x8Bit or DotProductInputAll capability). When <i>Vector 1</i> and <i>Vector 2</i> are vectors, the components of <i>Vector 2</i> must have a <i>Signedness</i> of 0. When <i>Vector 1</i> and <i>Vector 2</i> are scalar integer types, <i>Packed Vector Format</i> must be specified to select how the integers are to be interpreted as vectors. All components of <i>Vector 1</i> are sign-extended to the bit width of the result's type. All components of <i>Vector 2</i> are zero-extended to the bit width of the result's type. The sign- or zero-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. The resulting value will equal the low-order N bits of the correct result R, where N is the result width and R is computed with enough precision to avoid overflow and underflow.					Capability: DotProduct Missing before version 1.6.	
5 + variable	4452	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	Optional Packed Vector Format <i>Packed Vector Format</i>

<p>OpSDotAccSat (OpSDotAccSatKHR)</p> <p>Signed integer dot product of <i>Vector 1</i> and <i>Vector 2</i> and signed saturating addition of the result with <i>Accumulator</i>.</p> <p><i>Result Type</i> must be an integer type whose <i>Width</i> must be greater than or equal to that of the components of <i>Vector 1</i> and <i>Vector 2</i>.</p> <p><i>Vector 1</i> and <i>Vector 2</i> must have the same type.</p> <p><i>Vector 1</i> and <i>Vector 2</i> must be either 32-bit integers (enabled by the DotProductInput4x8BitPacked capability) or vectors of integer type (enabled by the DotProductInput4x8Bit or DotProductInputAll capability).</p> <p>The type of <i>Accumulator</i> must be the same as <i>Result Type</i>.</p> <p>When <i>Vector 1</i> and <i>Vector 2</i> are scalar integer types, <i>Packed Vector Format</i> must be specified to select how the integers are to be interpreted as vectors.</p> <p>All components of the input vectors are sign-extended to the bit width of the result's type. The sign-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. Finally, the resulting sum is added to the input accumulator. This final addition is saturating.</p> <p>If any of the multiplications or additions, with the exception of the final accumulation, overflow or underflow, the result of the instruction is undefined.</p>						<p>Capability: DotProduct</p> <p>Missing before version 1.6.</p>	
6 + variable	4453	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	<id> <i>Accumulator</i>	Optional Packed Vector Format <i>Packed Vector Format</i>

<p>OpUDotAccSat (OpUDotAccSatKHR)</p> <p>Unsigned integer dot product of <i>Vector 1</i> and <i>Vector 2</i> and unsigned saturating addition of the result with <i>Accumulator</i>.</p> <p><i>Result Type</i> must be an integer type with <i>Signedness</i> of 0 whose <i>Width</i> must be greater than or equal to that of the components of <i>Vector 1</i> and <i>Vector 2</i>.</p> <p><i>Vector 1</i> and <i>Vector 2</i> must have the same type.</p> <p><i>Vector 1</i> and <i>Vector 2</i> must be either 32-bit integers (enabled by the DotProductInput4x8BitPacked capability) or vectors of integer type with <i>Signedness</i> of 0 (enabled by the DotProductInput4x8Bit or DotProductInputAll capability).</p> <p>The type of <i>Accumulator</i> must be the same as <i>Result Type</i>.</p> <p>When <i>Vector 1</i> and <i>Vector 2</i> are scalar integer types, <i>Packed Vector Format</i> must be specified to select how the integers are to be interpreted as vectors.</p> <p>All components of the input vectors are zero-extended to the bit width of the result's type. The zero-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. Finally, the resulting sum is added to the input accumulator. This final addition is saturating.</p> <p>If any of the multiplications or additions, with the exception of the final accumulation, overflow or underflow, the result of the instruction is undefined.</p>						<p>Capability: DotProduct</p> <p>Missing before version 1.6.</p>	
6 + variable	4454	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	<id> <i>Accumulator</i>	Optional Packed Vector Format <i>Packed Vector Format</i>

OpSUDotAccSat (OpSUDotAccSatKHR) Mixed-signedness integer dot product of <i>Vector 1</i> and <i>Vector 2</i> and signed saturating addition of the result with <i>Accumulator</i> . Components of <i>Vector 1</i> are treated as signed, components of <i>Vector 2</i> are treated as unsigned. <i>Result Type</i> must be an integer type whose <i>Width</i> must be greater than or equal to that of the components of <i>Vector 1</i> and <i>Vector 2</i> . <i>Vector 1</i> and <i>Vector 2</i> must be either 32-bit integers (enabled by the DotProductInput4x8BitPacked capability) or vectors of integer type with the same number of components and same component <i>Width</i> (enabled by the DotProductInput4x8Bit or DotProductInputAll capability). When <i>Vector 1</i> and <i>Vector 2</i> are vectors, the components of <i>Vector 2</i> must have a <i>Signedness</i> of 0. The type of <i>Accumulator</i> must be the same as <i>Result Type</i> . When <i>Vector 1</i> and <i>Vector 2</i> are scalar integer types, <i>Packed Vector Format</i> must be specified to select how the integers are to be interpreted as vectors. All components of <i>Vector 1</i> are sign-extended to the bit width of the result's type. All components of <i>Vector 2</i> are zero-extended to the bit width of the result's type. The sign- or zero-extended input vectors are then multiplied component-wise and all components of the vector resulting from the component-wise multiplication are added together. Finally, the resulting sum is added to the input accumulator. This final addition is saturating. If any of the multiplications or additions, with the exception of the final accumulation, overflow or underflow, the result of the instruction is undefined.						Capability: DotProduct Missing before version 1.6.	
6 + variable	4455	<id> <i>Result Type</i>	Result <id>	<id> <i>Vector 1</i>	<id> <i>Vector 2</i>	<id> <i>Accumulator</i>	Optional Packed Vector Format Packed Vector Format

OpCooperativeMatrixMulAddKHR						Capability: CooperativeMatrixKHR	
Reserved.						Reserved.	
6 + variable	4459	<id> Result Type	Result <id>	<id> A	<id> B	<id> C	Optional Cooperative Matrix Operands Cooperative Matrix Operands

OpCooperativeMatrixReduceNV Reserved.					Capability: CooperativeMatrixReductionsNV Reserved.	
6	5366	<id> Result Type	Result <id>	<id> Matrix	Cooperative Matrix Reduce Mode Reduce	<id> CombineFunc

3.3.14. Bit Instructions

OpShiftRightLogical

Shift the bits in *Base* right by the number of bits specified in *Shift*. The most-significant bits are zero filled.

Result Type must be a scalar or vector of *integer type*.

The type of each *Base* and *Shift* must be a scalar or vector of *integer type*. *Base* and *Shift* must have the same number of components. The number of components and bit width of the type of *Base* must be the same as in *Result Type*.

Shift is consumed as an unsigned integer. The resulting value is undefined if *Shift* is greater than or equal to the bit width of the components of *Base*.

Results are computed per component.

5	194	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Base</i>	<id> <i>Shift</i>
---	-----	----------------------------	--------------------------	---------------------	----------------------

OpShiftRightArithmetic

Shift the bits in *Base* right by the number of bits specified in *Shift*. The most-significant bits are filled with the most-significant bit from *Base*.

Result Type must be a scalar or vector of *integer type*.

The type of each *Base* and *Shift* must be a scalar or vector of *integer type*. *Base* and *Shift* must have the same number of components. The number of components and bit width of the type of *Base* must be the same as in *Result Type*.

Shift is treated as unsigned. The resulting value is undefined if *Shift* is greater than or equal to the bit width of the components of *Base*.

Results are computed per component.

5	195	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Base</i>	<id> <i>Shift</i>
---	-----	----------------------------	--------------------------	---------------------	----------------------

OpShiftLeftLogical

Shift the bits in *Base* left by the number of bits specified in *Shift*. The least-significant bits are zero filled.

Result Type must be a scalar or vector of *integer type*.

The type of each *Base* and *Shift* must be a scalar or vector of *integer type*. *Base* and *Shift* must have the same number of components. The number of components and bit width of the type of *Base* must be the same as in *Result Type*.

Shift is treated as unsigned. The resulting value is undefined if *Shift* is greater than or equal to the bit width of the components of *Base*.

The number of components and bit width of *Result Type* must match those *Base* type. All types must be integer types.

Results are computed per component.

5	196	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>	<id> <i>Shift</i>
---	-----	----------------------------	--------------------	---------------------	----------------------

OpBitwiseOr

Result is 1 if either *Operand 1* or *Operand 2* is 1. Result is 0 if both *Operand 1* and *Operand 2* are 0.

Results are computed per component, and within each component, per bit.

Result Type must be a scalar or vector of *integer type*. The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

5	197	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpBitwiseXor

Result is 1 if exactly one of *Operand 1* or *Operand 2* is 1.
Result is 0 if *Operand 1* and *Operand 2* have the same value.

Results are computed per component, and within each component, per bit.

Result Type must be a scalar or vector of *integer type*. The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

5	198	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpBitwiseAnd

Result is 1 if both *Operand 1* and *Operand 2* are 1. Result is 0 if either *Operand 1* or *Operand 2* are 0.

Results are computed per component, and within each component, per bit.

Result Type must be a scalar or vector of *integer type*. The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same number of components as *Result Type*. They must have the same component width as *Result Type*.

5	199	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpNot

Complement the bits of *Operand*.

Results are computed per component, and within each component, per bit.

Result Type must be a scalar or vector of *integer type*.

Operand's type must be a scalar or vector of *integer type*. It must have the same number of components as *Result Type*. The component width must equal the component width in *Result Type*.

4	200	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>
---	-----	----------------------------	--------------------	------------------------

OpBitFieldInsert

Capability:
Shader, BitInstructions

Make a copy of an object, with a modified bit field that comes from another object.

Results are computed per component.

Result Type must be a scalar or vector of *integer type*.

The type of *Base* and *Insert* must be the same as *Result Type*.

Any result bits numbered outside [*Offset*, *Offset* + *Count* - 1] (*inclusive*) come from the corresponding bits in *Base*.

Any result bits numbered in [*Offset*, *Offset* + *Count* - 1] come, in order, from the bits numbered [0, *Count* - 1] of *Insert*.

Count must be an *integer type* scalar. *Count* is the number of bits taken from *Insert*. It is consumed as an unsigned value. *Count* can be 0, in which case the result is *Base*.

Offset must be an *integer type* scalar. *Offset* is the lowest-order bit of the bit field. It is consumed as an unsigned value.

The resulting value is undefined if *Count* or *Offset* or their sum is greater than the number of bits in the result.

7	201	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>	<id> <i>Insert</i>	<id> <i>Offset</i>	<id> <i>Count</i>
---	-----	----------------------------	--------------------	---------------------	-----------------------	-----------------------	----------------------

OpBitFieldSExtract Extract a bit field from an object, with sign extension. Results are computed per component. <i>Result Type</i> must be a scalar or vector of <i>integer type</i> . The type of <i>Base</i> must be the same as <i>Result Type</i> . If <i>Count</i> is greater than 0: The bits of <i>Base</i> numbered in [<i>Offset</i> , <i>Offset</i> + <i>Count</i> - 1] (<i>inclusive</i>) become the bits numbered [0, <i>Count</i> - 1] of the result. The remaining bits of the result will all be the same as bit <i>Offset</i> + <i>Count</i> - 1 of <i>Base</i> . <i>Count</i> must be an <i>integer type</i> scalar. <i>Count</i> is the number of bits extracted from <i>Base</i> . It is consumed as an unsigned value. <i>Count</i> can be 0, in which case the result is 0. <i>Offset</i> must be an <i>integer type</i> scalar. <i>Offset</i> is the lowest-order bit of the bit field to extract from <i>Base</i> . It is consumed as an unsigned value. The resulting value is undefined if <i>Count</i> or <i>Offset</i> or their sum is greater than the number of bits in the result.				Capability: Shader, BitInstructions		
6	202	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>	<id> <i>Offset</i>	<id> <i>Count</i>

OpBitFieldUExtract Extract a bit field from an object, without sign extension. The semantics are the same as with OpBitFieldSExtract with the exception that there is no sign extension. The remaining bits of the result will all be 0.				Capability: Shader, BitInstructions		
6	203	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>	<id> <i>Offset</i>	<id> <i>Count</i>

OpBitReverse Reverse the bits in an object. Results are computed per component. <i>Result Type</i> must be a scalar or vector of <i>integer type</i> . The type of <i>Base</i> must be the same as <i>Result Type</i> . The bit-number <i>n</i> of the result is taken from bit-number <i>Width</i> - 1 - <i>n</i> of <i>Base</i> , where <i>Width</i> is the OpTypeInt operand of the <i>Result Type</i> .				Capability: Shader, BitInstructions		
4	204	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>		

OpBitCount Count the number of set bits in an object. Results are computed per component. <i>Result Type</i> must be a scalar or vector of <i>integer type</i> . The components must be wide enough to hold the unsigned <i>Width</i> of <i>Base</i> as an unsigned value. That is, no sign bit is needed or counted when checking for a wide enough result width. <i>Base</i> must be a scalar or vector of <i>integer type</i> . It must have the same number of components as <i>Result Type</i> . The result is the unsigned value that is the number of bits in <i>Base</i> that are 1.					
4	205	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Base</i>	

OpBitwiseFunctionINTEL Reserved.						Capability: TernaryBitwiseFunctionINTEL Reserved.	
7	6242	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>A</i>	<id> <i>B</i>	<id> <i>C</i>	<id> <i>LUTIndex</i>

3.3.15. Relational and Logical Instructions

OpAny

Result is **true** if any component of *Vector* is **true**, otherwise result is **false**.

Result Type must be a *Boolean type* scalar.

Vector must be a vector of *Boolean type*.

4	154	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Vector</i>
---	-----	----------------------------	--------------------------	-----------------------

OpAll

Result is **true** if all components of *Vector* are **true**, otherwise result is **false**.

Result Type must be a *Boolean type* scalar.

Vector must be a vector of *Boolean type*.

4	155	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Vector</i>
---	-----	----------------------------	--------------------------	-----------------------

OpIsNan

Result is **true** if *x* is a NaN for the floating-point encoding used by the type of *x*, otherwise result is **false**.

Result Type must be a scalar or vector of *Boolean type*.

x must be a scalar or vector of *floating-point type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	156	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>x</i>
---	-----	----------------------------	--------------------------	------------------

OpIsInf

Result is **true** if *x* is an Inf for the floating-point encoding used by the type of *x*, otherwise result is **false**

Result Type must be a scalar or vector of *Boolean type*.

x must be a scalar or vector of *floating-point type*. It must have the same number of components as *Result Type*.

Results are computed per component.

4	157	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>x</i>
---	-----	----------------------------	--------------------------	------------------

OpIsFinite Result is true if <i>x</i> is a finite number for the floating-point encoding used by the type of <i>x</i> , otherwise result is false . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . <i>x</i> must be a scalar or vector of <i>floating-point type</i> . It must have the same number of components as <i>Result Type</i> . Results are computed per component.				Capability: Kernel
4	158	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>x</i>

OpIsNormal Result is true if <i>x</i> is a normal number for the floating-point encoding used by the type of <i>x</i> , otherwise result is false . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . <i>x</i> must be a scalar or vector of <i>floating-point type</i> . It must have the same number of components as <i>Result Type</i> . Results are computed per component.				Capability: Kernel
4	159	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>x</i>

OpSignBitSet Result is true if <i>x</i> has its sign bit set, otherwise result is false . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . <i>x</i> must be a scalar or vector of <i>floating-point type</i> . It must have the same number of components as <i>Result Type</i> . Results are computed per component.				Capability: Kernel
4	160	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>x</i>

<p>OpLessOrGreater</p> <p>Deprecated (use OpFordNotEqual).</p> <p>Has the same semantics as OpFordNotEqual.</p> <p><i>Result Type</i> must be a scalar or vector of <i>Boolean type</i>.</p> <p>x must be a scalar or vector of <i>floating-point type</i>. It must have the same number of components as <i>Result Type</i>.</p> <p>y must have the same type as x.</p> <p>Results are computed per component.</p>				<p>Capability: Kernel</p> <p>Missing after version 1.5.</p>	
5	161	<id> <i>Result Type</i>	<i>Result <id></i>	<id> x	<id> y

<p>OpOrdered</p> <p>Result is true if both $x == x$ and $y == y$ are true, where OpFordEqual is used as comparison, otherwise result is false.</p> <p><i>Result Type</i> must be a scalar or vector of <i>Boolean type</i>.</p> <p>x must be a scalar or vector of <i>floating-point type</i>. It must have the same number of components as <i>Result Type</i>.</p> <p>y must have the same type as x.</p> <p>Results are computed per component.</p>				<p>Capability: Kernel</p>	
5	162	<id> <i>Result Type</i>	<i>Result <id></i>	<id> x	<id> y

<p>OpUnordered</p> <p>Result is true if either x or y is an NaN for the floating-point encoding used by the type of x and y, otherwise result is false.</p> <p><i>Result Type</i> must be a scalar or vector of <i>Boolean type</i>.</p> <p>x must be a scalar or vector of <i>floating-point type</i>. It must have the same number of components as <i>Result Type</i>.</p> <p>y must have the same type as x.</p> <p>Results are computed per component.</p>				<p>Capability: Kernel</p>	
5	163	<id> <i>Result Type</i>	<i>Result <id></i>	<id> x	<id> y

OpLogicalEqual

Result is **true** if *Operand 1* and *Operand 2* have the same value. Result is **false** if *Operand 1* and *Operand 2* have different values.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* must be the same as *Result Type*.

The type of *Operand 2* must be the same as *Result Type*.

Results are computed per component.

5	164	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2
---	-----	---------------------	-------------	-------------------	-------------------

OpLogicalNotEqual

Result is **true** if *Operand 1* and *Operand 2* have different values. Result is **false** if *Operand 1* and *Operand 2* have the same value.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* must be the same as *Result Type*.

The type of *Operand 2* must be the same as *Result Type*.

Results are computed per component.

5	165	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2
---	-----	---------------------	-------------	-------------------	-------------------

OpLogicalOr

Result is **true** if either *Operand 1* or *Operand 2* is **true**. Result is **false** if both *Operand 1* and *Operand 2* are **false**.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* must be the same as *Result Type*.

The type of *Operand 2* must be the same as *Result Type*.

Results are computed per component.

5	166	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2
---	-----	---------------------	-------------	-------------------	-------------------

OpLogicalAnd Result is true if both <i>Operand 1</i> and <i>Operand 2</i> are true . Result is false if either <i>Operand 1</i> or <i>Operand 2</i> are false . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . The type of <i>Operand 1</i> must be the same as <i>Result Type</i> . The type of <i>Operand 2</i> must be the same as <i>Result Type</i> . Results are computed per component.					
5	167	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

OpLogicalNot Result is true if <i>Operand</i> is false . Result is false if <i>Operand</i> is true . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . The type of <i>Operand</i> must be the same as <i>Result Type</i> . Results are computed per component.					
4	168	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand</i>	

OpSelect Select between two objects. Before version 1.4 , results are only computed per component. Before version 1.4 , <i>Result Type</i> must be a pointer, scalar, or vector. Starting with version 1.4 , <i>Result Type</i> can additionally be a <i>composite</i> type other than a vector. The types of <i>Object 1</i> and <i>Object 2</i> must be the same as <i>Result Type</i> . <i>Condition</i> must be a scalar or vector of <i>Boolean type</i> . If <i>Condition</i> is a scalar and true , the result is <i>Object 1</i> . If <i>Condition</i> is a scalar and false , the result is <i>Object 2</i> . If <i>Condition</i> is a vector, <i>Result Type</i> must be a vector with the same number of components as <i>Condition</i> and the result is a mix of <i>Object 1</i> and <i>Object 2</i> : If a component of <i>Condition</i> is true , the corresponding component in the result is taken from <i>Object 1</i> , otherwise it is taken from <i>Object 2</i> .						
6	169	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Condition</i>	<id> <i>Object 1</i>	<id> <i>Object 2</i>

OpIEqual

Integer comparison for equality.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	170	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------------	--------------------------	--------------------------

OpINotEqual

Integer comparison for inequality.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	171	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------------	--------------------------	--------------------------

OpUGreaterThan

Unsigned-integer comparison if *Operand 1* is greater than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	172	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------------	--------------------------	--------------------------

OpSGreaterThan

Signed-integer comparison if *Operand 1* is greater than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	173	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpUGreaterThanEqual

Unsigned-integer comparison if *Operand 1* is greater than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	174	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSGreaterThanEqual

Signed-integer comparison if *Operand 1* is greater than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	175	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpULessThan

Unsigned-integer comparison if *Operand 1* is less than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	176	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSLessThan

Signed-integer comparison if *Operand 1* is less than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	177	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpULessThanEqual

Unsigned-integer comparison if *Operand 1* is less than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	178	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpSLessThanEqual

Signed-integer comparison if *Operand 1* is less than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *integer type*. They must have the same component width, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	179	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdEqual

Floating-point comparison for being ordered and equal.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	180	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordEqual

Floating-point comparison for being unordered or equal.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	181	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdNotEqual

Floating-point comparison for being ordered and not equal.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	182	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordNotEqual

Floating-point comparison for being unordered or not equal.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	183	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdLessThan

Floating-point comparison if operands are ordered and *Operand 1* is less than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	184	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordLessThan

Floating-point comparison if operands are unordered or *Operand 1* is less than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	185	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdGreaterThan

Floating-point comparison if operands are ordered and *Operand 1* is greater than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	186	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordGreaterThan

Floating-point comparison if operands are unordered or *Operand 1* is greater than *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	187	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdLessThanEqual

Floating-point comparison if operands are ordered and *Operand 1* is less than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	188	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordLessThanEqual

Floating-point comparison if operands are unordered or *Operand 1* is less than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	189	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFOrdGreaterThanEqual

Floating-point comparison if operands are ordered and *Operand 1* is greater than or equal to *Operand 2*.

Result Type must be a scalar or vector of *Boolean type*.

The type of *Operand 1* and *Operand 2* must be a scalar or vector of *floating-point type*. They must have the same type, and they must have the same number of components as *Result Type*.

Results are computed per component.

5	190	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>
---	-----	----------------------------	--------------------	--------------------------	--------------------------

OpFUnordGreaterThanOrEqual Floating-point comparison if operands are unordered or <i>Operand 1</i> is greater than or equal to <i>Operand 2</i> . <i>Result Type</i> must be a scalar or vector of <i>Boolean type</i> . The type of <i>Operand 1</i> and <i>Operand 2</i> must be a scalar or vector of <i>floating-point type</i> . They must have the same type, and they must have the same number of components as <i>Result Type</i> . Results are computed per component.					
5	191	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Operand 1</i>	<id> <i>Operand 2</i>

3.3.16. Derivative Instructions

OpDPdx Same result as either OpDPdxFine or OpDPdxCoarse on <i>P</i> . Selection of which one is based on external factors. An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before <i>X'</i> . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of <i>P</i> must be the same as <i>Result Type</i> . <i>P</i> is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .					Capability: Shader
4	207	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>P</i>	

OpDPdy Same result as either OpDPdyFine or OpDPdyCoarse on <i>P</i> . Selection of which one is based on external factors. An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before <i>X'</i> . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of <i>P</i> must be the same as <i>Result Type</i> . <i>P</i> is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .					Capability: Shader
4	208	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>P</i>	

OpFwidth Result is the same as computing the sum of the absolute values of OpDPdx and OpDPdy on <i>P</i> . An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of <i>P</i> must be the same as <i>Result Type</i> . <i>P</i> is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .					Capability: Shader
4	209	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>P</i>	

OpDPdxFine Result is the partial derivative of <i>P</i> with respect to the window <i>x</i> coordinate. Uses local differencing based on the value of <i>P</i> for the current fragment and its immediate neighbor(s). An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of <i>P</i> must be the same as <i>Result Type</i> . <i>P</i> is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .					Capability: DerivativeControl
4	210	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>P</i>	

OpDPdyFine Result is the partial derivative of P with respect to the window y coordinate. Uses local differencing based on the value of P for the current fragment and its immediate neighbor(s). An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of P must be the same as <i>Result Type</i> . P is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .				Capability: DerivativeControl
4	211	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> P

OpFwidthFine Result is the same as computing the sum of the absolute values of OpDPdxFine and OpDPdyFine on P . An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of P must be the same as <i>Result Type</i> . P is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .				Capability: DerivativeControl
4	212	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> P

OpDPdxCoarse Result is the partial derivative of P with respect to the window x coordinate. Uses local differencing based on the value of P for the current fragment's neighbors, and possibly, but not necessarily, includes the value of P for the current fragment. That is, over a given area, the implementation can compute x derivatives in fewer unique locations than would be allowed for OpDPdxFine . An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of P must be the same as <i>Result Type</i> . P is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .				Capability: DerivativeControl
4	213	$\langle id \rangle$ <i>Result Type</i>	<i>Result $\langle id \rangle$</i>	$\langle id \rangle$ P

OpDPdyCoarse Result is the partial derivative of P with respect to the window y coordinate. Uses local differencing based on the value of P for the current fragment's neighbors, and possibly, but not necessarily, includes the value of P for the current fragment. That is, over a given area, the implementation can compute y derivatives in fewer unique locations than would be allowed for OpDPdyFine . An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its derivative group have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits. The type of P must be the same as <i>Result Type</i> . P is the value to take the derivative of. This instruction is only valid in the Fragment Execution Model .				Capability: DerivativeControl
4	214	$\langle id \rangle$ <i>Result Type</i>	<i>Result $\langle id \rangle$</i>	$\langle id \rangle$ P

OpFwidthCoarse				
<p>Result is the same as computing the sum of the absolute values of OpDPdxCoarse and OpDPdyCoarse on <i>P</i>.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its derivative group have executed all dynamic instances that are program-ordered before <i>X'</i>.</p> <p><i>Result Type</i> must be a scalar or vector of floating-point type using the IEEE 754 encoding. The component width must be 32 bits.</p> <p>The type of <i>P</i> must be the same as <i>Result Type</i>. <i>P</i> is the value to take the derivative of.</p> <p>This instruction is only valid in the Fragment Execution Model.</p>				
4	215	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>P</i>

3.3.17. Control-Flow Instructions

OpPhi

The SSA phi function.

The result is selected based on control flow: If control reached the current block from *Parent i*, *Result Id* gets the value that *Variable i* had at the end of *Parent i*.

Result Type can be any type except **OpTypeVoid**.

Operands are a sequence of pairs: (*Variable 1*, *Parent 1* block), (*Variable 2*, *Parent 2* block), ... Each *Parent i* block is the label of an immediate predecessor in the CFG of the current block. There must be exactly one *Parent i* for each parent block of the current block in the CFG. If *Parent i* is reachable in the CFG and *Variable i* is defined in a block, that defining block must dominate *Parent i*. All *Variables* must have a type matching *Result Type*.

Within a block, this instruction must appear before all non-**OpPhi** instructions (except for **OpLine** and **OpNoLine**, which can be mixed with **OpPhi**).

3 + variable	245	<id> <i>Result Type</i>	<i>Result</i> <id>	<id>, <id>, ... <i>Variable, Parent, ...</i>
--------------	-----	----------------------------	--------------------	---

OpLoopMerge

Declare a structured loop.

This instruction must immediately precede either an **OpBranch** or **OpBranchConditional** instruction. That is, it must be the second-to-last instruction in its block.

Merge Block is the label of the merge block for this structured loop.

Continue Target is the label of a block targeted for processing a loop "continue".

Loop Control Parameters appear in **Loop Control**-table order for any *Loop Control* setting that requires such a parameter.

See **Structured Control Flow** for more detail.

4 + variable	246	<id> <i>Merge Block</i>	<id> <i>Continue Target</i>	<i>Loop Control</i>	<i>Literal, Literal, ... Loop Control Parameters</i>
--------------	-----	----------------------------	--------------------------------	---------------------	--

OpSelectionMerge Declare a structured selection. This instruction must immediately precede either an OpBranchConditional or OpSwitch instruction. That is, it must be the second-to-last instruction in its block. <i>Merge Block</i> is the label of the merge block for this structured selection. See Structured Control Flow for more detail.			
3	247	<id> <i>Merge Block</i>	<i>Selection Control</i>

OpLabel The label instruction of a block . References to a block are through the <i>Result <id></i> of its label.			
2	248		<i>Result <id></i>

OpBranch Unconditional branch to <i>Target Label</i> . <i>Target Label</i> must be the <i>Result <id></i> of an OpLabel instruction in the current function. This instruction must be the last instruction in a block.			
2	249	<id> <i>Target Label</i>	

<p>OpBranchConditional</p> <p>If <i>Condition</i> is true, branch to <i>True Label</i>, otherwise branch to <i>False Label</i>.</p> <p><i>Condition</i> must be a <i>Boolean type</i> scalar.</p> <p><i>True Label</i> must be an OpLabel in the current function.</p> <p><i>False Label</i> must be an OpLabel in the current function.</p> <p>Starting with version 1.6, <i>True Label</i> and <i>False Label</i> must not be the same <i><id></i>.</p> <p><i>Branch weights</i> are unsigned 32-bit integer literals. There must be either no <i>Branch Weights</i> or exactly two branch weights. If present, the first is the weight for branching to <i>True Label</i>, and the second is the weight for branching to <i>False Label</i>. The implied probability that a branch is taken is its weight divided by the sum of the two <i>Branch weights</i>. At least one weight must be non-zero. A weight of zero does not imply a branch is dead or permit its removal; branch weights are only hints. The sum of the two weights must not overflow a 32-bit unsigned integer.</p> <p>If <i>Condition</i> is an OpUndef, behavior is undefined.</p> <p>This instruction must be the last instruction in a block.</p>					
4 + variable	250	<i><id></i> <i>Condition</i>	<i><id></i> <i>True Label</i>	<i><id></i> <i>False Label</i>	<i>Literal, Literal, ...</i> <i>Branch weights</i>

OpSwitch Multi-way branch to one of the operand label <i><id></i> . <i>Selector</i> must have a type of OpTypeInt . <i>Selector</i> is compared for equality to the <i>Target</i> literals. <i>Default</i> must be the <i><id></i> of a label. If <i>Selector</i> does not equal any of the <i>Target</i> literals, control flow branches to the <i>Default</i> label <i><id></i> . <i>Target</i> must be alternating scalar integer <i>literals</i> and the <i><id></i> of a label. If <i>Selector</i> equals a <i>literal</i> , control flow branches to the following <i>label <id></i> . It is invalid for any two <i>literal</i> to be equal to each other. If <i>Selector</i> does not equal any <i>literal</i> , control flow branches to the <i>Default</i> label <i><id></i> . Each <i>literal</i> is interpreted with the type of <i>Selector</i> . The bit width of <i>Selector</i> 's type is the width of each <i>literal</i> 's type. If this width is not a multiple of 32-bits and the OpTypeInt Signedness is set to 1, the <i>literal</i> values are interpreted as being sign extended. If <i>Selector</i> is an OpUndef , behavior is undefined. This instruction must be the last instruction in a block.				
3 + variable	251	<i><id></i> <i>Selector</i>	<i><id></i> <i>Default</i>	<i>literal 1, label <id> 1,</i> <i>literal 2, label <id> 2,</i> ... <i>Target</i>

OpKill Deprecated (use OpTerminateInvocation or OpDemoteToHelperInvocation). Fragment-shader discard. Ceases all further processing in any <i>invocation</i> that executes it: Only instructions these invocations executed before OpKill have observable side effects. If this instruction is executed in non-uniform <i>control flow</i> , all subsequent control flow is non-uniform (for invocations that continue to execute). This instruction must be the last instruction in a block. This instruction is only valid in the Fragment Execution Model .		Capability: Shader
1		252

OpReturn Return with no value from a function with void return type. This instruction must be the last instruction in a block.		
1		253

OpReturnValue Return a value from a function. <i>Value</i> is the value returned, by copy, and must match the <i>Return Type</i> operand of the OpTypeFunction type of the OpFunction body this return instruction is in. <i>Value</i> must not have type OpTypeVoid . This instruction must be the last instruction in a block.			
2	254	<id> <i>Value</i>	

OpUnreachable Behavior is undefined if this instruction is executed. This instruction must be the last instruction in a block.			
1			255

OpLifetimeStart Declare that an object was not defined before this instruction. <i>Pointer</i> is a pointer to the object whose lifetime is starting. Its type must be an OpTypePointer with Storage Class Function . <i>Size</i> is an unsigned 32-bit integer. <i>Size</i> must be 0 if <i>Pointer</i> is a pointer to a non-void type or the Addresses capability is not declared. If <i>Size</i> is non-zero, it is the number of bytes of memory whose lifetime is starting.			Capability: Kernel
3	256	<id> <i>Pointer</i>	<i>Literal</i> <i>Size</i>

OpLifetimeStop Declare that an object is dead after this instruction. <i>Pointer</i> is a pointer to the object whose lifetime is ending. Its type must be an OpTypePointer with Storage Class Function . <i>Size</i> is an unsigned 32-bit integer. <i>Size</i> must be 0 if <i>Pointer</i> is a pointer to a non-void type or the Addresses capability is not declared. If <i>Size</i> is non-zero, it is the number of bytes of memory whose lifetime is ending.			Capability: Kernel
3	257	<id> <i>Pointer</i>	<i>Literal</i> <i>Size</i>

<p>OpTerminateInvocation</p> <p>Fragment-shader terminate.</p> <p>Ceases all further processing in any invocation that executes it: Only instructions these invocations executed before OpTerminateInvocation will have observable side effects. If this instruction is executed in non-uniform control flow, all subsequent control flow is non-uniform (for invocations that continue to execute).</p> <p>This instruction must be the last instruction in a block.</p> <p>This instruction is only valid in the Fragment Execution Model.</p>	<p>Capability: Shader</p> <p>Missing before version 1.6.</p>
1	4416

<p>OpDemoteToHelperInvocation (OpDemoteToHelperInvocationEXT)</p> <p>Demote this fragment shader invocation to a helper invocation. Any stores to memory after this instruction are suppressed and the fragment does not write outputs to the framebuffer.</p> <p>Unlike the OpTerminateInvocation instruction, this does not necessarily terminate the invocation which might be needed for derivative calculations. It is not considered a flow control instruction (flow control does not become non-uniform) and does not terminate the block. The implementation may terminate helper invocations before the end of the shader as an optimization, but doing so must not affect derivative calculations and does not make control flow non-uniform.</p> <p>After an invocation executes this instruction, any subsequent load of HelperInvocation within that invocation will load an undefined value unless the HelperInvocation built-in variable is decorated with Volatile or the load included Volatile in its Memory Operands</p> <p>This instruction is only valid in the Fragment Execution Model.</p>	<p>Capability: DemoteToHelperInvocation</p> <p>Missing before version 1.6.</p>
1	5380

3.3.18. Atomic Instructions

OpAtomicLoad Atomically load through <i>Pointer</i> using the given <i>Semantics</i> . All subparts of the value that is loaded are read atomically with respect to all other atomic accesses to it within <i>Memory</i> . <i>Result Type</i> must be a scalar of <i>integer type</i> or <i>floating-point type</i> . <i>Pointer</i> is the pointer to the memory to read. The type of the value pointed to by <i>Pointer</i> must be the same as <i>Result Type</i> . <i>Memory</i> is a memory <i>Scope</i> .						
6	227	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics

OpAtomicStore Atomically store through <i>Pointer</i> using the given <i>Semantics</i> . All subparts of <i>Value</i> are written atomically with respect to all other atomic accesses to it within <i>Memory</i> . <i>Pointer</i> is the pointer to the memory to write. The type it points to must be a scalar of <i>integer type</i> or <i>floating-point type</i> . <i>Value</i> is the value to write. The type of <i>Value</i> and the type pointed to by <i>Pointer</i> must be the same type. <i>Memory</i> is a memory <i>Scope</i> .						
5	228	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value	

OpAtomicExchange

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* from copying *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be a scalar of *integer type* or *floating-point type*.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	229	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicCompareExchange

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* from *Value* only if *Original Value* equals *Comparator*, and
- 3) store the *New Value* back through *Pointer* only if *Original Value* equaled *Comparator*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

Use *Equal* for the memory semantics of this instruction when *Value* and *Original Value* compare equal.

Use *Unequal* for the memory semantics of this instruction when *Value* and *Original Value* compare unequal. *Unequal* must not be set to **Release** or **Acquire and Release**. In addition, *Unequal* cannot be set to a stronger memory-order than *Equal*.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*. This type must also match the type of *Comparator*.

Memory is a memory *Scope*.

9	230	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Equal	Memory Semantics <id> Unequal	<id> Value	<id> Comparat or
---	-----	------------------------	----------------	-----------------	-------------------------	--------------------------------------	--	---------------	------------------------

OpAtomicCompareExchangeWeak Deprecated (use OpAtomicCompareExchange). Has the same semantics as OpAtomicCompareExchange . <i>Memory</i> is a memory <i>Scope</i> .							Capability: Kernel Missing after version 1.3 .		
9	231	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Equal	Memory Semantics <id> Unequal	<id> Value	<id> Comparat or

OpAtomicIncrement Perform the following steps atomically with respect to any other atomic accesses within <i>Memory</i> to the same location: 1) load through <i>Pointer</i> to get an <i>Original Value</i> , 2) get a <i>New Value</i> through integer addition of 1 to <i>Original Value</i> , and 3) store the <i>New Value</i> back through <i>Pointer</i> . The instruction's result is the <i>Original Value</i> . <i>Result Type</i> must be an <i>integer type</i> scalar. The type of the value pointed to by <i>Pointer</i> must be the same as <i>Result Type</i> . <i>Memory</i> is a memory <i>Scope</i> .									
6	232	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics			

OpAtomicDecrement Perform the following steps atomically with respect to any other atomic accesses within <i>Memory</i> to the same location: 1) load through <i>Pointer</i> to get an <i>Original Value</i> , 2) get a <i>New Value</i> through integer subtraction of 1 from <i>Original Value</i> , and 3) store the <i>New Value</i> back through <i>Pointer</i> . The instruction's result is the <i>Original Value</i> . <i>Result Type</i> must be an <i>integer type</i> scalar. The type of the value pointed to by <i>Pointer</i> must be the same as <i>Result Type</i> . <i>Memory</i> is a memory <i>Scope</i> .									
6	233	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics			

OpAtomicIAdd

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by integer addition of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	234	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicISub

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by integer subtraction of *Value* from *Original Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	235	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicSMin

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by finding the smallest signed integer of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	236	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicUMin

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by finding the smallest unsigned integer of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	237	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicSMax

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by finding the largest signed integer of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	238	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicUMax

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by finding the largest unsigned integer of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	239	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicAnd

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by the bitwise AND of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	240	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicOr

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by the bitwise OR of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	241	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicXor

Perform the following steps atomically with respect to any other atomic accesses within *Memory* to the same location:

- 1) load through *Pointer* to get an *Original Value*,
- 2) get a *New Value* by the bitwise exclusive OR of *Original Value* and *Value*, and
- 3) store the *New Value* back through *Pointer*.

The instruction's result is the *Original Value*.

Result Type must be an *integer type* scalar.

The type of *Value* must be the same as *Result Type*. The type of the value pointed to by *Pointer* must be the same as *Result Type*.

Memory is a memory *Scope*.

7	242	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value
---	-----	---------------------	-------------	-----------------	----------------------	--	---------------

OpAtomicFlagTestAndSet

Atomically sets the flag value pointed to by *Pointer* to the set state.

Pointer must be a pointer to a 32-bit integer type representing an atomic flag.

The instruction's result is true if the flag was in the set state or false if the flag was in the clear state immediately before the operation.

Result Type must be a *Boolean type*.

The resulting values are undefined if an atomic flag is modified by an instruction other than **OpAtomicFlagTestAndSet** or **OpAtomicFlagClear**.

Memory is a memory *Scope*.

Capability:
Kernel

6	318	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	
---	-----	---------------------	-------------	-----------------	----------------------	---------------------------------------	--

OpAtomicFlagClear Atomically sets the flag value pointed to by <i>Pointer</i> to the clear state. <i>Pointer</i> must be a pointer to a 32-bit integer type representing an atomic flag. Memory Semantics must not be Acquire or AcquireRelease The resulting values are undefined if an atomic flag is modified by an instruction other than OpAtomicFlagTestAndSet or OpAtomicFlagClear . <i>Memory</i> is a memory <i>Scope</i> .					Capability: Kernel
4	319	<id> <i>Pointer</i>	<i>Scope</i> <id> <i>Memory</i>	<i>Memory Semantics</i> <id> <i>Semantics</i>	

OpAtomicFMinEXT						Capability: AtomicFloat16MinMaxEXT, AtomicFloat32MinMaxEXT, AtomicFloat64MinMaxEXT, AtomicFloat16VectorNV	
Reserved.						Reserved.	
7	5614	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value

OpAtomicFMaxEXT						Capability: AtomicFloat16MinMaxEXT, AtomicFloat32MinMaxEXT, AtomicFloat64MinMaxEXT, AtomicFloat16VectorNV	
Reserved.						Reserved.	
7	5615	<id> Result Type	Result <id>	<id> Pointer	Scope <id> Memory	Memory Semantics <id> Semantics	<id> Value

OpAtomicFAddEXT Reserved.						Capability: AtomicFloat16AddEXT, AtomicFloat32AddEXT, AtomicFloat64AddEXT, AtomicFloat16VectorNV Reserved.	
7	6035	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Pointer</i>	<i>Scope <id></i> <i>Memory</i>	<i>Memory Semantics</i> <i><id></i> <i>Semantics</i>	<i><id></i> <i>Value</i>

3.3.19. Primitive Instructions

OpEmitVertex Emits the current values of all output variables to the current output primitive. After execution, the values of all output variables are undefined. This instruction must only be used when only one stream is present.		Capability: Geometry
1		218

OpEndPrimitive Finish the current primitive and start a new one. No vertex is emitted. This instruction must only be used when only one stream is present.		Capability: Geometry
1		219

OpEmitStreamVertex Emits the current values of all output variables to the current output primitive. After execution, the values of all output variables are undefined. <i>Stream</i> must be an <i><id></i> of a <i>constant instruction</i> with a scalar integer type. That constant is the output-primitive stream number. This instruction must only be used when multiple streams are present.		Capability: GeometryStreams
2	220	<i><id></i> <i>Stream</i>

OpEndStreamPrimitive Finish the current primitive and start a new one. No vertex is emitted. <i>Stream</i> must be an <i><id></i> of a <i>constant instruction</i> with a scalar integer type. That constant is the output-primitive stream number. This instruction must only be used when multiple streams are present.		Capability: GeometryStreams
2	221	<i><id></i> <i>Stream</i>

3.3.20. Barrier Instructions

OpControlBarrier

Wait for all invocations in the [scope restricted tangle](#) to reach the current point of execution before executing further instructions.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before](#) *X*'.

An invocation will not execute [dynamic instances](#) that are [program-ordered after](#) a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed *X*'.

When *Execution* is **Workgroup** or larger, behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

If *Semantics* is not **None**, this instruction also serves as an **OpMemoryBarrier** instruction, and also performs and adheres to the description and semantics of an **OpMemoryBarrier** instruction with the same *Memory* and *Semantics* operands. This allows atomically specifying both a control barrier and a memory barrier (that is, without needing two instructions). If *Semantics* is **None**, *Memory* is ignored.

Before **version 1.3**, it is only valid to use this instruction with **TessellationControl**, **GLCompute**, or **Kernel** [execution models](#). There is no such restriction starting with **version 1.3**.

If used with the **TessellationControl** [execution model](#), it also implicitly synchronizes the **Output Storage Class**: Writes to **Output** variables performed by any invocation executed prior to a **OpControlBarrier** are visible to any other invocation proceeding beyond that **OpControlBarrier**.

4	224	Scope <id> <i>Execution</i>	Scope <id> <i>Memory</i>	Memory Semantics <id> <i>Semantics</i>
---	-----	--	---	---

OpMemoryBarrier Control the order that memory accesses are observed. Ensures that memory accesses issued before this instruction are observed before memory accesses issued after this instruction. This control is ensured only for memory accesses issued by this invocation and observed by another invocation executing within <i>Memory</i> scope. If the Vulkan memory model is declared, this ordering only applies to memory accesses that use the NonPrivatePointer memory operand or NonPrivateTexel image operand . <i>Semantics</i> declares what kind of memory is being controlled and what kind of control to apply. To execute both a memory barrier and a control barrier, see OpControlBarrier .				
3	225	Scope <id> <i>Memory</i>	Memory Semantics <id> <i>Semantics</i>	

OpNamedBarrierInitialize Declare a new named-barrier object. <i>Result Type</i> must be the type OpTypeNamedBarrier . <i>Subgroup Count</i> must be a 32-bit integer type scalar representing the number of subgroups that must reach the current point of execution.				
		Capability: NamedBarrier Missing before version 1.1.		
4	328	<id> <i>Result Type</i>	Result <id>	<id> <i>Subgroup Count</i>

OpMemoryNamedBarrier Wait for other invocations of this module to reach the current point of execution. <i>Named Barrier</i> must be the type OpTypeNamedBarrier . If <i>Semantics</i> is not None , this instruction also serves as an OpMemoryBarrier instruction, and also performs and adheres to the description and semantics of an OpMemoryBarrier instruction with the same <i>Memory</i> and <i>Semantics</i> operands. This allows atomically specifying both a control barrier and a memory barrier (that is, without needing two instructions). If <i>Semantics</i> None , <i>Memory</i> is ignored.				
		Capability: NamedBarrier Missing before version 1.1.		
4	329	<id> <i>Named Barrier</i>	Scope <id> <i>Memory</i>	Memory Semantics <id> <i>Semantics</i>

OpControlBarrierArriveINTEL Reserved.				Capability: SplitBarrierINTEL Reserved.
4	6142	<i>Scope <id> Execution</i>	<i>Scope <id> Memory</i>	<i>Memory Semantics <id> Semantics</i>

OpControlBarrierWaitINTEL Reserved.				Capability: SplitBarrierINTEL Reserved.
4	6143	<i>Scope <id> Execution</i>	<i>Scope <id> Memory</i>	<i>Memory Semantics <id> Semantics</i>

3.3.21. Group and Subgroup Instructions

OpGroupAsyncCopy

Capability:
Kernel

Perform an asynchronous group copy of *Num Elements* elements from *Source* to *Destination*. The asynchronous copy is performed by all invocations in the [scope restricted tangle](#).

This instruction results in an event object that can be used by [OpGroupWaitEvents](#) to wait for the async copy to finish.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*') until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Result Type must be an [OpTypeEvent](#) object.

Destination must be a pointer to a scalar or vector of [floating-point type](#) or [integer type](#).

Destination pointer [Storage Class](#) must be **Workgroup** or **CrossWorkgroup**.

The type of *Source* must be the same as *Destination*.

If *Destination* pointer [Storage Class](#) is **Workgroup**, the *Source* pointer [Storage Class](#) must be **CrossWorkgroup**. In this case *Stride* defines the stride in elements when reading from *Source* pointer.

If *Destination* pointer [Storage Class](#) is **CrossWorkgroup**, the *Source* pointer [Storage Class](#) must be **Workgroup**. In this case *Stride* defines the stride in elements when writing each element to *Destination* pointer.

Stride and *NumElements* must be a 32-bit [integer type](#) scalar if the [addressing model](#) is *Physical32* and 64 bit [integer type](#) scalar if the *Addressing Model* is *Physical64*.

Event must have a type of [OpTypeEvent](#).

Event can be used to associate the copy with a previous copy allowing an event to be shared by multiple copies. Otherwise *Event* should be an [OpConstantNull](#).

If *Event* is not [OpConstantNull](#), the result is the event object supplied by the *Event* operand.

9	259	<id> Result Type	Result <id>	Scope <id> Execution	<id> Destinatio n	<id> Source	<id> Num Elements	<id> Stride	<id> Event
---	-----	------------------------	--------------------------------	--	-------------------------	----------------	-------------------------	----------------	---------------

OpGroupWaitEvents Wait for events generated by OpGroupAsyncCopy operations to complete. <i>Events List</i> points to <i>Num Events</i> event objects, which is released after the wait is performed. <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' . <i>Num Events</i> must be a 32-bit integer type scalar. <i>Events List</i> must be a pointer to OpTypeEvent .				Capability: Kernel
4	260	Scope <id> <i>Execution</i>	<id> <i>Num Events</i>	<id> <i>Events List</i>

OpGroupAll Evaluates a predicate for all invocations in the scope restricted tangle ,resulting in true if predicate evaluates to true for all invocations in the scope restricted tangle, otherwise the result is false . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a Boolean type . <i>Predicate</i> must be a Boolean type .				Capability: Groups	
5	261	<id> Result Type	Result <id>	Scope <id> Execution	<id> Predicate

OpGroupAny Evaluates a predicate for all invocations in the scope restricted tangle , resulting in true if predicate evaluates to true for any invocation in the scope restricted tangle, otherwise the result is false . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' . <i>Result Type</i> must be a Boolean type . <i>Predicate</i> must be a Boolean type .				Capability: Groups	
5	262	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	<id> <i>Predicate</i>

OpGroupBroadcast					Capability: Groups	
<p>Broadcast the <i>Value</i> of the invocation identified by the local id <i>LocalId</i> to the result of all invocations in the scope restricted tangle.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command.</p> <p>Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p> <p><i>Result Type</i> must be a scalar or vector of floating-point type, integer type, or Boolean type.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>LocalId</i> must be an integer datatype. It must be a scalar, a vector with 2 components, or a vector with 3 components. Behavior is undefined unless <i>LocalId</i> is the same for all invocations in the group, or if it is greater than or equal to the size of the group in any dimension.</p>						
6	263	<id> Result Type	Result <id>	Scope <id> Execution	<id> Value	<id> LocalId

OpGroupIAdd An integer add group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before <i>X</i> '. <i>Result Type</i> must be a scalar or vector of integer type . The identity <i>I</i> for <i>Operation</i> is 0. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	264	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupFAdd A floating-point add group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before <i>X</i> '. <i>Result Type</i> must be a scalar or vector of floating-point type . The identity <i>I</i> for <i>Operation</i> is 0. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	265	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupFMin A floating-point minimum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X . <i>Result Type</i> must be a scalar or vector of floating-point type . The identity <i>I</i> for <i>Operation</i> is +INF. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	266	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupUMin An unsigned integer minimum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X . <i>Result Type</i> must be a scalar or vector of integer type . The identity <i>I</i> for <i>Operation</i> is UINT_MAX when <i>X</i> is 32 bits wide and ULONG_MAX when <i>X</i> is 64 bits wide. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	267	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupSMin A signed integer minimum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i> ') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X '. <i>Result Type</i> must be a scalar or vector of integer type . The identity <i>I</i> for <i>Operation</i> is INT_MAX when <i>X</i> is 32 bits wide and LONG_MAX when <i>X</i> is 64 bits wide. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	268	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupFMax A floating-point maximum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i> ') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X '. <i>Result Type</i> must be a scalar or vector of floating-point type . The identity <i>I</i> for <i>Operation</i> is -INF. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	269	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupUMax An unsigned integer maximum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X . <i>Result Type</i> must be a scalar or vector of integer type . The identity <i>I</i> for <i>Operation</i> is 0. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	270	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpGroupSMax A signed integer maximum group operation specified for all values of <i>X</i> specified by invocations in the scope restricted tangle . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. Behavior is undefined unless all invocations within <i>Execution</i> execute the same dynamic instance of this instruction. An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X . <i>Result Type</i> must be a scalar or vector of integer type . The identity <i>I</i> for <i>Operation</i> is INT_MIN when <i>X</i> is 32 bits wide and LONG_MIN when <i>X</i> is 64 bits wide. The type of <i>X</i> must be the same as <i>Result Type</i> .					Capability: Groups	
6	271	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>X</i>

OpSubgroupBallotKHR Reserved.				Capability: SubgroupBallotKHR Reserved.
4	4421	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Predicate</i>

OpSubgroupFirstInvocationKHR Reserved.				Capability: SubgroupBallotKHR Reserved.
4	4422	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Value</i>

OpSubgroupAllKHR Reserved.				Capability: SubgroupVoteKHR Reserved.
4	4428	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Predicate</i>

OpSubgroupAnyKHR Reserved.				Capability: SubgroupVoteKHR Reserved.
4	4429	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Predicate</i>

OpSubgroupAllEqualKHR Reserved.				Capability: SubgroupVoteKHR Reserved.
4	4430	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Predicate</i>

OpGroupNonUniformRotateKHR						Capability: GroupNonUniformRotateKHR	
Reserved.						Reserved.	
6 + variable	4431	<id> Result Type	Result <id>	Scope <id> Execution	<id> Value	<id> Delta	Optional <id> ClusterSize

OpSubgroupReadInvocationKHR Reserved.				Capability: SubgroupBallotKHR Reserved.	
5	4432	<id> Result Type	Result <id>	<id> Value	<id> Index

OpGroupIAddNonUniformAMD Reserved.				Capability: Groups Reserved. Also see extension: SPV_AMD_shader_ballot	
6	5000	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation <id> Operation X

OpGroupFAddNonUniformAMD Reserved.				Capability: Groups Reserved. Also see extension: SPV_AMD_shader_ballot	
6	5001	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation <id> Operation X

OpGroupFMinNonUniformAMD Reserved.				Capability: Groups Reserved. Also see extension: SPV_AMD_shader_ballot	
6	5002	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation <id> Operation X

OpGroupUMinNonUniformAMD Reserved.				Capability: Groups Reserved. Also see extension: SPV_AMD_shader_ballot	
6	5003	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation <id> Operation X

OpGroupSMinNonUniformAMD Reserved.					Capability: Groups Reserved. Also see extension: SPV_AMD_shader_ballot	
6	5004	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

OpGroupFMaxNonUniformAMD					Capability: Groups
Reserved.					Reserved.
					Also see extension: SPV_AMD_shader_ballot
6	5005	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation <id> X

OpGroupUMaxNonUniformAMD					Capability: Groups	
Reserved.					Reserved.	
					Also see extension: SPV_AMD_shader_ballot	
6	5006	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

OpGroupSMaxNonUniformAMD					Capability: Groups	
Reserved.					Reserved.	
					Also see extension: SPV_AMD_shader_ballot	
6	5007	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

OpSubgroupShuffleINTEL Reserved.					Capability: SubgroupShuffleINTEL Reserved.
5	5571	<i><id> Result Type</i>	<i>Result <id></i>	<i><id> Data</i>	<i><id> InvocationId</i>

OpSubgroupShuffleDownINTEL Reserved.					Capability: SubgroupShuffleINTEL Reserved.	
6	5572	<id> Result Type	Result <id>	<id> Current	<id> Next	<id> Delta

OpSubgroupShuffleUpINTEL Reserved.					Capability: SubgroupShuffleINTEL Reserved.	
6	5573	<id> Result Type	Result <id>	<id> Previous	<id> Current	<id> Delta

OpSubgroupShuffleXorINTEL Reserved.					Capability: SubgroupShuffleINTEL Reserved.	
5	5574	<id> Result Type	Result <id>	<id> Data	<id> Value	

OpSubgroupBlockReadINTEL				Capability:
Reserved.				SubgroupBufferBlockIOINTEL
				Reserved.
4	5575	<id> Result Type	Result <id>	<id> Ptr

OpSubgroupBlockWriteINTEL Reserved.					Capability: SubgroupBufferBlockIOINTEL Reserved.	
3	5576	<id> Ptr	<id> Data			

OpSubgroupImageBlockReadINTEL Reserved.					Capability: SubgroupImageBlockIOINTEL Reserved.	
5	5577	<id> Result Type	Result <id>	<id> Image	<id> Coordinate	

OpSubgroupImageBlockWriteINTEL Reserved.					Capability: SubgroupImageBlockIOINTEL Reserved.
4	5578	<id> Image	<id> Coordinate	<id> Data	

OpSubgroupImageMediaBlockReadINTEL Reserved.							Capability: SubgroupImageMediaBlockIOINTEL Reserved.
7	5580	<id> Result Type	Result <id>	<id> Image	<id> Coordinate	<id> Width	<id> Height

OpSubgroupImageMediaBlockWriteINTEL Reserved.							Capability: SubgroupImageMediaBlockIOINTEL Reserved.
6	5581	<id> Image	<id> Coordinate	<id> Width	<id> Height	<id> Data	

OpSubgroupBlockPrefetchINTEL				Capability:
Reserved.				SubgroupBufferPrefetchINTEL
				Reserved.
3 + variable	6221	<i><id>Ptr</i>	<i><id>NumBytes</i>	Optional <i>Memory Operands</i>

OpSubgroup2DBlockLoadINTEL									Capability: Subgroup2DBlockIOINTEL		
Reserved.									Reserved.		
1 1	6231	<id> Element Size	<id> Block Width	<id> Block Height	<id> Block Count	<id> Src Base Pointer	<id> Memory Width	<id> Memory Height	<id> Memory Pitch	<id> Coordin ate	<id> Dst Pointer

OpSubgroup2DBlockLoadTransformINTEL Reserved.									Capability: Subgroup2DBlockTransformINTEL Reserved.		
1 1	6232	<id> Element Size	<id> Block Width	<id> Block Height	<id> Block Count	<id> Src Base Pointer	<id> Memory Width	<id> Memory Height	<id> Memory Pitch	<id> Coordin ate	<id> Dst Pointer

OpSubgroup2DBlockLoadTransposeINTEL Reserved.									Capability: Subgroup2DBlockTransposeINTEL Reserved.		
1 1	6233	<id> Element Size	<id> Block Width	<id> Block Height	<id> Block Count	<id> Src Base Pointer	<id> Memory Width	<id> Memory Height	<id> Memory Pitch	<id> Coordin ate	<id> Dst Pointer

OpSubgroup2DBlockPrefetchINTEL Reserved.									Capability: Subgroup2DBlockIOINTEL Reserved.		
1 0	6234	<id> Element Size	<id> Block Width	<id> Block Height	<id> Block Count	<id> Src Base Pointer	<id> Memory Width	<id> Memory Height	<id> Memory Pitch	<id> Memory Pitch	<id> Coordin ate

OpSubgroup2DBlockStoreINTEL Reserved.									Capability: Subgroup2DBlockIOINTEL Reserved.		
1 1	6235	<id> Element Size	<id> Block Width	<id> Block Height	<id> Block Count	<id> Src Pointer	<id> Dst Base Pointer	<id> Memory Width	<id> Memory Height	<id> Memory Pitch	<id> Coordin ate

OpSubgroupMatrixMultiplyAccumulateINTEL Reserved.							Capability: SubgroupMatrixMultiplyAccumulateINTEL Reserved.
7 + variable	6237	<id> Result Type	Result <id>	<id> K Dim	<id> Matrix A	<id> Matrix B	<id> Matrix C Optional Matrix Multiply Accumulate Operands

OpGroupIMulKHR Reserved.							Capability: GroupUniformArithmeticKHR Reserved.
6	6401	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X	

OpGroupFMulKHR Reserved.							Capability: GroupUniformArithmeticKHR Reserved.
6	6402	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X	

OpGroupBitwiseAndKHR Reserved.							Capability: GroupUniformArithmeticKHR Reserved.
6	6403	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X	

OpGroupBitwiseOrKHR Reserved.							Capability: GroupUniformArithmeticKHR Reserved.
6	6404	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X	

OpGroupBitwiseXorKHR Reserved.							Capability: GroupUniformArithmeticKHR Reserved.
6	6405	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X	

OpGroupLogicalAndKHR Reserved.					Capability: GroupUniformArithmeticKHR Reserved.	
6	6406	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

OpGroupLogicalOrKHR Reserved.					Capability: GroupUniformArithmeticKHR Reserved.	
6	6407	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

OpGroupLogicalXorKHR Reserved.					Capability: GroupUniformArithmeticKHR Reserved.	
6	6408	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> X

3.3.22. Device-Side Enqueue Instructions

<div><div>OpEnqueueMarker</div><div>Enqueue a marker command to the queue object specified by <i>Queue</i>. The marker command waits for a list of events to complete, or if the list is empty it waits for all previously enqueued commands in <i>Queue</i> to complete before the marker completes.</div><div><i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. A successful enqueue results in the value 0. A failed enqueue results in a non-0 value.</div><div><i>Queue</i> must be of the type OpTypeQueue.</div><div><i>Num Events</i> specifies the number of event objects in the wait list pointed to by <i>Wait Events</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer.</div><div><i>Wait Events</i> specifies the list of wait event objects and must be a pointer to OpTypeDeviceEvent.</div><div><i>Ret Event</i> is a pointer to a device event which gets implicitly retained by this instruction. It must have a type of OpTypePointer to OpTypeDeviceEvent. If <i>Ret Event</i> is set to null this instruction becomes a no-op.</div></div>				<div>Capability:</div> <div>DeviceEnqueue</div>			
7	291	<id> Result Type	Result <id>	<id> Queue	<id> Num Events	<id> Wait Events	<id> Ret Event

OpEnqueueKernel

Enqueue the function specified by *Invoke* and the NDRange specified by *ND Range* for execution to the queue object specified by *Queue*.

Result Type must be a 32-bit *integer type* scalar. A successful enqueue results in the value 0. A failed enqueue results in a non-0 value.

Queue must be of the type **OpTypeQueue**.

Flags must be an *integer type* scalar. The content of *Flags* is interpreted as *Kernel Enqueue Flags* mask.

The type of *ND Range* must be an **OpTypeStruct** whose members are as described by the *Result Type* of **OpBuildNDRange**.

Num Events specifies the number of event objects in the wait list pointed to by *Wait Events* and must be 32-bit *integer type* scalar, which is treated as an unsigned integer.

Wait Events specifies the list of wait event objects and must be a pointer to **OpTypeDeviceEvent**.

Ret Event must be a pointer to **OpTypeDeviceEvent** which gets implicitly retained by this instruction.

Invoke must be an **OpFunction** whose **OpTypeFunction** operand has:

- *Result Type* must be **OpTypeVoid**.
- The first parameter must have a type of **OpTypePointer** to an 8-bit **OpTypeInt**.
- An optional list of parameters, each of which must have a type of **OpTypePointer** to the **Workgroup Storage Class**.

Param is the first parameter of the function specified by *Invoke* and must be a pointer to an 8-bit *integer type* scalar.

Param Size is the size in bytes of the memory pointed to by *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

Param Align is the alignment of *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

Each *Local Size* operand corresponds (in order) to one **OpTypePointer** to **Workgroup Storage Class** parameter to the *Invoke* function, and specifies the number of bytes of **Workgroup** storage used to back the pointer during the execution of the *Invoke* function.

Capability:
DeviceEnqueue

13 + variab le	292	<id> Resul t Type	Resul t <id>	<id> Queu e	<id> Flags	<id> ND Rang e	<id> Num Event s	<id> Wait Event s	<id> Ret Event	<id> Invok e	<id> Para m	<id> Para m Size	<id> Para m Align	<id>, <id>, ... Local Size
----------------------	-----	-------------------------	-----------------	-------------------	---------------	-------------------------	---------------------------	----------------------------	----------------------	--------------------	-------------------	---------------------------	----------------------------	--

OpGetKernelNDRangeSubGroupCount Result is the number of subgroups in each workgroup of the dispatch (except for the last in cases where the global size does not divide cleanly into workgroups) given the combination of the passed NDRange descriptor specified by <i>ND Range</i> and the function specified by <i>Invoke</i> . <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. The type of <i>ND Range</i> must be an OpTypeStruct whose members are as described by the <i>Result Type</i> of OpBuildNDRange . <i>Invoke</i> must be an OpFunction whose OpTypeFunction operand has: <ul style="list-style-type: none"> - <i>Result Type</i> must be OpTypeVoid. - The first parameter must have a type of OpTypePointer to an 8-bit OpTypeInt. - An optional list of parameters, each of which must have a type of OpTypePointer to the Workgroup Storage Class. <i>Param</i> is the first parameter of the function specified by <i>Invoke</i> and must be a pointer to an 8-bit <i>integer type</i> scalar. <i>Param Size</i> is the size in bytes of the memory pointed to by <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer. <i>Param Align</i> is the alignment of <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer.							Capability: DeviceEnqueue	
8	293	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>ND Range</i>	<id> <i>Invoke</i>	<id> <i>Param</i>	<id> <i>Param Size</i>	<id> <i>Param Align</i>

OpGetKernelNDRangeMaxSubGroupSize

Capability:
DeviceEnqueue

Result is the maximum subgroup size for the function specified by *Invoke* and the NDRange specified by *ND Range*.

Result Type must be a 32-bit *integer type* scalar.

The type of *ND Range* must be an **OpTypeStruct** whose members are as described by the *Result Type* of **OpBuildNDRange**.

Invoke must be an **OpFunction** whose **OpTypeFunction** operand has:

- *Result Type* must be **OpTypeVoid**.
- The first parameter must have a type of **OpTypePointer** to an 8-bit **OpTypeInt**.
- An optional list of parameters, each of which must have a type of **OpTypePointer** to the **Workgroup Storage Class**.

Param is the first parameter of the function specified by *Invoke* and must be a pointer to an 8-bit *integer type* scalar.

Param Size is the size in bytes of the memory pointed to by *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

Param Align is the alignment of *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

8	294	<id> Result Type	Result <id>	<id> ND Range	<id> Invoke	<id> Param	<id> Param Size	<id> Param Align
---	-----	---------------------	-------------	------------------	----------------	---------------	--------------------	---------------------

OpGetKernelWorkGroupSize Result is the maximum workgroup size that can be used to execute the function specified by <i>Invoke</i> on the device. <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. <i>Invoke</i> must be an OpFunction whose OpTypeFunction operand has: <ul style="list-style-type: none"> - <i>Result Type</i> must be OpTypeVoid. - The first parameter must have a type of OpTypePointer to an 8-bit OpTypeInt. - An optional list of parameters, each of which must have a type of OpTypePointer to the Workgroup Storage Class. <i>Param</i> is the first parameter of the function specified by <i>Invoke</i> and must be a pointer to an 8-bit <i>integer type</i> scalar. <i>Param Size</i> is the size in bytes of the memory pointed to by <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer. <i>Param Align</i> is the alignment of <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer.						Capability: DeviceEnqueue	
7	295	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Invoke</i>	<id> <i>Param</i>	<id> <i>Param Size</i>	<id> <i>Param Align</i>

OpGetKernelPreferredWorkGroupSizeMultiple Result is the preferred multiple of workgroup size for the function specified by <i>Invoke</i> . This is a performance hint. Specifying a workgroup size that is not a multiple of this result as the value of the local work size does not fail to enqueue <i>Invoke</i> for execution unless the workgroup size specified is larger than the device maximum. <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. <i>Invoke</i> must be an OpFunction whose OpTypeFunction operand has: <ul style="list-style-type: none"> - <i>Result Type</i> must be OpTypeVoid. - The first parameter must have a type of OpTypePointer to an 8-bit OpTypeInt. - An optional list of parameters, each of which must have a type of OpTypePointer to the Workgroup Storage Class. <i>Param</i> is the first parameter of the function specified by <i>Invoke</i> and must be a pointer to an 8-bit <i>integer type</i> scalar. <i>Param Size</i> is the size in bytes of the memory pointed to by <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer. <i>Param Align</i> is the alignment of <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer.				Capability: DeviceEnqueue			
7	296	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Invoke</i>	<id> <i>Param</i>	<id> <i>Param Size</i>	<id> <i>Param Align</i>

OpRetainEvent Increments the reference count of the event object specified by <i>Event</i> . Behavior is undefined if <i>Event</i> is not a valid event.		Capability: DeviceEnqueue	
2	297	<id> <i>Event</i>	

OpReleaseEvent Decrements the reference count of the event object specified by <i>Event</i> . The event object is deleted once the event reference count is zero, the specific command identified by this event has completed (or terminated) and there are no commands in any device command queue that require a wait for this event to complete. Behavior is undefined if <i>Event</i> is not a valid event.		Capability: DeviceEnqueue	
2	298	<id> <i>Event</i>	

OpCreateUserEvent Create a user event. The execution status of the created event is set to a value of 2 (CL_SUBMITTED). <i>Result Type</i> must be OpTypeDeviceEvent .			Capability: DeviceEnqueue
3	299	<id> <i>Result Type</i>	<i>Result <id></i>

OpIsValidEvent Result is true if the event specified by <i>Event</i> is a valid event, otherwise false . <i>Result Type</i> must be a <i>Boolean type</i> . <i>Event</i> must have a type of OpTypeDeviceEvent			Capability: DeviceEnqueue
4	300	<id> <i>Result Type</i>	<i>Result <id></i> <id> <i>Event</i>

OpSetUserEventStatus Sets the execution status of a user event specified by <i>Event.Status</i> can be either 0 (CL_COMPLETE) to indicate that this kernel and all its child kernels finished execution successfully, or a negative integer value indicating an error. <i>Event</i> must have a type of OpTypeDeviceEvent that was produced by OpCreateUserEvent . <i>Status</i> must have a type of 32-bit OpTypeInt treated as a signed integer.			Capability: DeviceEnqueue
3	301	<id> <i>Event</i>	<id> <i>Status</i>

OpCaptureEventProfilingInfo Captures the profiling information specified by <i>Profiling Info</i> for the command associated with the event specified by <i>Event</i> in the memory pointed to by <i>Value</i> . The profiling information is available in the memory pointed to by <i>Value</i> after the command identified by <i>Event</i> has completed. <i>Event</i> must have a type of OpTypeDeviceEvent that was produced by OpEnqueueKernel or OpEnqueueMarker . <i>Profiling Info</i> must be an <i>integer type</i> scalar. The content of <i>Profiling Info</i> is interpreted as <i>Kernel Profiling Info</i> mask. <i>Value</i> must be a pointer to a scalar 8-bit <i>integer type</i> in the CrossWorkgroup Storage Class . If <i>Profiling Info</i> is CmdExecTime , <i>Value</i> behavior is defined only if it points to 128-bit memory range. The first 64 bits contain the elapsed time CL_PROFILING_COMMAND_END - CL_PROFILING_COMMAND_START for the command identified by <i>Event</i> in nanoseconds. The second 64 bits contain the elapsed time CL_PROFILING_COMMAND_COMPLETE - CL_PROFILING_COMMAND_START for the command identified by <i>Event</i> in nanoseconds. Note: What is captured is undefined if this instruction is called multiple times for the same event.				Capability: DeviceEnqueue
4	302	<id> <i>Event</i>	<id> <i>Profiling Info</i>	<id> <i>Value</i>

OpGetDefaultQueue The result is the default device queue, or if a default device queue has not been created, a null queue object. <i>Result Type</i> must be an OpTypeQueue .			Capability: DeviceEnqueue
3	303	<id> <i>Result Type</i>	<i>Result <id></i>

OpBuildNDRange

Given the global work size specified by *GlobalWorkSize*, local work size specified by *LocalWorkSize* and global work offset specified by *GlobalWorkOffset*, builds the result as a 1D, 2D, or 3D ND-range descriptor structure.

Result Type must be an **OpTypeStruct** with the following ordered list of members, starting from the first to last:

- 1) A 32-bit *integer type* scalar that specifies the number of dimensions in the global size and the workgroup size.
- 2) An **OpTypeArray** with 3 elements, where each element is a 32-bit *integer type* scalar if the *addressing model* is **Physical32** or a 64-bit *integer type* scalar if the *addressing model* is **Physical64**. This is an array of per-dimension unsigned values that specifies the global offset used to calculate the global ID for an invocation.
- 3) An **OpTypeArray** with 3 elements, where each element is a 32-bit *integer type* scalar if the *addressing model* is **Physical32** or a 64-bit *integer type* scalar if the *addressing model* is **Physical64**. This is an array of per-dimension unsigned values that specifies the number of global invocations that execute the kernel function.
- 4) An **OpTypeArray** with 3 elements, where each element is a 32-bit *integer type* scalar if the *addressing model* is **Physical32** or a 64-bit *integer type* scalar if the *addressing model* is **Physical64**. This is an array of per-dimension unsigned values that specifies the number of invocations in a workgroup.

GlobalWorkSize must be a scalar or an array with 2 or 3 components. Where the type of each element in the array is 32-bit *integer type* scalar if the *addressing model* is **Physical32** or 64-bit *integer type* scalar if the *addressing model* is **Physical64**.

The type of *LocalWorkSize* must be the same as *GlobalWorkSize*.

The type of *GlobalWorkOffset* must be the same as *GlobalWorkSize*.

Capability:
DeviceEnqueue

6	304	<id> Result Type	Result <id>	<id> GlobalWorkSize	<id> LocalWorkSize	<id> GlobalWorkOffset
---	-----	---------------------	-------------	------------------------	-----------------------	--------------------------

OpGetKernelLocalSizeForSubgroupCount Result is the 1D local size to enqueue <i>Invoke</i> with <i>Subgroup Count</i> subgroups per workgroup. <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. <i>Subgroup Count</i> must be a 32-bit <i>integer type</i> scalar. <i>Invoke</i> must be an OpFunction whose OpTypeFunction operand has: <ul style="list-style-type: none"> - <i>Result Type</i> must be OpTypeVoid. - The first parameter must have a type of OpTypePointer to an 8-bit OpTypeInt. - An optional list of parameters, each of which must have a type of OpTypePointer to the Workgroup Storage Class. <i>Param</i> is the first parameter of the function specified by <i>Invoke</i> and must be a pointer to an 8-bit <i>integer type</i> scalar. <i>Param Size</i> is the size in bytes of the memory pointed to by <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer. <i>Param Align</i> is the alignment of <i>Param</i> and must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned integer.						Capability: SubgroupDispatch Missing before version 1.1.		
8	325	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Subgroup Count</i>	<id> <i>Invoke</i>	<id> <i>Param</i>	<id> <i>Param Size</i>	<id> <i>Param Align</i>

OpGetKernelMaxNumSubgroups

Result is the maximum number of subgroups that can be used to execute *Invoke* on the device.

Result Type must be a 32-bit *integer type* scalar.

Invoke must be an **OpFunction** whose **OpTypeFunction** operand has:

- *Result Type* must be **OpTypeVoid**.
- The first parameter must have a type of **OpTypePointer** to an 8-bit **OpTypeInt**.
- An optional list of parameters, each of which must have a type of **OpTypePointer** to the **Workgroup Storage Class**.

Param is the first parameter of the function specified by *Invoke* and must be a pointer to an 8-bit *integer type* scalar.

Param Size is the size in bytes of the memory pointed to by *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

Param Align is the alignment of *Param* and must be a 32-bit *integer type* scalar, which is treated as an unsigned integer.

Capability:
SubgroupDispatch

Missing before version 1.1.

7	326	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Invoke</i>	<id> <i>Param</i>	<id> <i>Param Size</i>	<id> <i>Param Align</i>
---	-----	----------------------------	--------------------	-----------------------	----------------------	---------------------------	----------------------------

3.3.23. Pipe Instructions

OpReadPipe Read a packet from the pipe object specified by <i>Pipe</i> into <i>Pointer</i> . Result is 0 if the operation is successful and a negative value if the pipe is empty. <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. <i>Pipe</i> must have a type of OpTypePipe with ReadOnly <i>access qualifier</i> . <i>Pointer</i> must have a type of OpTypePointer with the same data type as <i>Pipe</i> and a Generic <i>Storage Class</i> . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .						Capability: Pipes	
7	274	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Pointer</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpWritePipe Write a packet from <i>Pointer</i> to the pipe object specified by <i>Pipe</i> . Result is 0 if the operation is successful and a negative value if the pipe is full. <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar. <i>Pipe</i> must have a type of OpTypePipe with WriteOnly <i>access qualifier</i> . <i>Pointer</i> must have a type of OpTypePointer with the same data type as <i>Pipe</i> and a Generic <i>Storage Class</i> . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .						Capability: Pipes	
7	275	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Pointer</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpReservedReadPipe

Read a packet from the reserved area specified by *Reserve Id* and *Index* of the pipe object specified by *Pipe* into *Pointer*. The reserved pipe entries are referred to by indices that go from 0 ... *Num Packets* - 1. Result is 0 if the operation is successful and a negative value otherwise.

Result Type must be a 32-bit *integer type* scalar.

Pipe must have a type of **OpTypePipe** with **ReadOnly** *access qualifier*.

Reserve Id must have a type of **OpTypeReserveld**.

Index must be a 32-bit *integer type* scalar, which is treated as an unsigned value.

Pointer must have a type of **OpTypePointer** with the same data type as *Pipe* and a **Generic Storage Class**.

Packet Size must be a 32-bit *integer type* scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit *integer type* scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

Capability:
Pipes

9	276	<id> Result Type	<i>Result</i> <id>	<id> Pipe	<id> Reserve Id	<id> Index	<id> Pointer	<id> Packet Size	<id> Packet Alignment
---	-----	------------------------	-----------------------	--------------	-----------------------	---------------	-----------------	------------------------	-----------------------------

OpReservedWritePipe

Capability:
Pipes

Write a packet from *Pointer* into the reserved area specified by *Reserve Id* and *Index* of the pipe object specified by *Pipe*. The reserved pipe entries are referred to by indices that go from 0 ... *Num Packets* - 1. Result is 0 if the operation is successful and a negative value otherwise.

Result Type must be a 32-bit *integer type* scalar.

Pipe must have a type of **OpTypePipe** with **WriteOnly** *access qualifier*.

Reserve Id must have a type of **OpTypeReserveld**.

Index must be a 32-bit *integer type* scalar, which is treated as an unsigned value.

Pointer must have a type of **OpTypePointer** with the same data type as *Pipe* and a **Generic** *Storage Class*.

Packet Size must be a 32-bit *integer type* scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit *integer type* scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

9	277	<id> Result Type	<i>Result</i> <id>	<id> Pipe	<id> Reserve Id	<id> Index	<id> Pointer	<id> Packet Size	<id> Packet Alignment
---	-----	------------------------	-----------------------	--------------	-----------------------	---------------	-----------------	------------------------	-----------------------------

OpReserveReadPipePackets Reserve <i>Num Packets</i> entries for reading from the pipe object specified by <i>Pipe</i> . Result is a valid reservation ID if the reservation is successful. <i>Result Type</i> must be an OpTypeReserveld . <i>Pipe</i> must have a type of OpTypePipe with ReadOnly <i>access qualifier</i> . <i>Num Packets</i> must be a 32-bit <i>integer type</i> scalar, which is treated as an unsigned value. <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .						Capability: Pipes	
7	278	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Num Packets</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpReserveWritePipePackets Reserve <i>num_packets</i> entries for writing to the pipe object specified by <i>Pipe</i> . Result is a valid reservation ID if the reservation is successful. <i>Pipe</i> must have a type of OpTypePipe with WriteOnly <i>access qualifier</i> . <i>Num Packets</i> must be a 32-bit OpTypeInt which is treated as an unsigned value. <i>Result Type</i> must be an OpTypeReserveld . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .						Capability: Pipes	
7	279	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Num Packets</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpCommitReadPipe Indicates that all reads to <i>Num Packets</i> associated with the reservation specified by <i>Reserve Id</i> and the pipe object specified by <i>Pipe</i> are completed. <i>Pipe</i> must have a type of OpTypePipe with ReadOnly <i>access qualifier</i> . <i>Reserve Id</i> must have a type of OpTypeReserveld . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .				Capability: Pipes	
5	280	<id> <i>Pipe</i>	<id> <i>Reserve Id</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpCommitWritePipe Indicates that all writes to <i>Num Packets</i> associated with the reservation specified by <i>Reserve Id</i> and the pipe object specified by <i>Pipe</i> are completed. <i>Pipe</i> must have a type of OpTypePipe with WriteOnly <i>access qualifier</i> . <i>Reserve Id</i> must have a type of OpTypeReserveld . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .				Capability: Pipes	
5	281	<id> <i>Pipe</i>	<id> <i>Reserve Id</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpIsValidReserveld Result is true if <i>Reserve Id</i> is a valid reservation id and false otherwise. <i>Result Type</i> must be a <i>Boolean type</i> . <i>Reserve Id</i> must have a type of OpTypeReserveld .				Capability: Pipes
4	282	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Reserve Id</i>

OpGetNumPipePackets					Capability: Pipes	
<p>Result is the number of available entries in the pipe object specified by <i>Pipe</i>. The number of available entries in a pipe is a dynamic value. The result is considered immediately stale.</p> <p><i>Result Type</i> must be a 32-bit <i>integer type</i> scalar, which should be treated as an unsigned value.</p> <p><i>Pipe</i> must have a type of OpTypePipe with ReadOnly or WriteOnly <i>access qualifier</i>.</p> <p><i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe.</p> <p><i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe.</p> <p>Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i>.</p>						
6	283	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpGetMaxPipePackets Result is the maximum number of packets specified by the creation of <i>Pipe</i> . <i>Result Type</i> must be a 32-bit <i>integer type</i> scalar, which should be treated as an unsigned value. <i>Pipe</i> must have a type of OpTypePipe with ReadOnly or WriteOnly <i>access qualifier</i> . <i>Packet Size</i> must be a 32-bit <i>integer type</i> scalar that represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> must be a 32-bit <i>integer type</i> scalar that represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> .					Capability: Pipes	
6	284	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>Pipe</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>

OpGroupReserveReadPipePackets

Reserve *Num Packets* entries for the [scope restricted tangle](#) for reading from the pipe object specified by *Pipe*. Result is a valid reservation id if the reservation is successful.

The reserved pipe entries are referred to by indices that go from 0 ... *Num Packets* - 1.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

An invocation will not execute a [dynamic instance](#) of this instruction (X') until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Result Type must be an [OpTypeReserveld](#).

Pipe must have a type of [OpTypePipe](#) with **ReadOnly** [access qualifier](#).

Num Packets must be a 32-bit [integer type](#) scalar, which is treated as an unsigned value.

Packet Size must be a 32-bit [integer type](#) scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit [integer type](#) scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

Capability:
Pipes

8	285	<id> Result Type	Result <id>	Scope <id> Execution	<id> Pipe	<id> Num Packets	<id> Packet Size	<id> Packet Alignment
---	-----	---------------------	-----------------------------------	---	--------------	------------------------	---------------------	-----------------------------

OpGroupReserveWritePipePackets

Capability:
Pipes

Reserve *Num Packets* entries for the [scope restricted tangle](#) for writing to the pipe object specified by *Pipe*. Result is a valid reservation id if the reservation is successful.

The reserved pipe entries are referred to by indices that go from 0 ... *Num Packets* - 1.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

An invocation will not execute a [dynamic instance](#) of this instruction (X') until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Result Type must be an [OpTypeReserveld](#).

Pipe must have a type of [OpTypePipe](#) with **WriteOnly** [access qualifier](#).

Num Packets must be a 32-bit [integer type](#) scalar, which is treated as an unsigned value.

Packet Size must be a 32-bit [integer type](#) scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit [integer type](#) scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

8	286	<id> Result Type	Result <id>	Scope <id> Execution	<id> Pipe	<id> Num Packets	<id> Packet Size	<id> Packet Alignment
---	-----	---------------------	-----------------------------------	---	--------------	------------------------	---------------------	-----------------------------

OpGroupCommitReadPipe

Capability:
Pipes

Indicates that all reads to *Num Packets* associated with the reservation specified by *Reserve Id* and the pipe object specified by *Pipe* were completed by the [scope restricted tangle](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Pipe must have a type of **OpTypePipe** with **ReadOnly** [access qualifier](#).

Reserve Id must have a type of **OpTypeReserveld**.

Packet Size must be a 32-bit [integer type](#) scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit [integer type](#) scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

6	287	Scope <id> <i>Execution</i>	<id> <i>Pipe</i>	<id> <i>Reserve Id</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>
---	-----	--	---------------------	---------------------------	----------------------------	---------------------------------

OpGroupCommitWritePipe

Indicates that all writes to *Num Packets* associated with the reservation specified by *Reserve Id* and the pipe object specified by *Pipe* were completed by the [scope restricted tangle](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Behavior is undefined unless all invocations within *Execution* execute the same dynamic instance of this instruction.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Pipe must have a type of [OpTypePipe](#) with **WriteOnly** [access qualifier](#).

Reserve Id must have a type of [OpTypeReserveld](#).

Packet Size must be a 32-bit [integer type](#) scalar that represents the size in bytes of each packet in the pipe.

Packet Alignment must be a 32-bit [integer type](#) scalar that represents the alignment in bytes of each packet in the pipe.

Behavior is undefined unless *Packet Alignment* > 0 and evenly divides *Packet Size*.

Capability:
Pipes

6	288	Scope <id> <i>Execution</i>	<id> <i>Pipe</i>	<id> <i>Reserve Id</i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>
---	-----	--	---------------------	---------------------------	----------------------------	---------------------------------

OpConstantPipeStorage Creates a pipe-storage object. <i>Result Type</i> must be OpTypePipeStorage . <i>Packet Size</i> is an unsigned 32-bit integer. It represents the size in bytes of each packet in the pipe. <i>Packet Alignment</i> is an unsigned 32-bit integer. It represents the alignment in bytes of each packet in the pipe. Behavior is undefined unless <i>Packet Alignment</i> > 0 and evenly divides <i>Packet Size</i> . <i>Capacity</i> is an unsigned 32-bit integer. It is the minimum number of <i>Packet Size</i> blocks the resulting OpTypePipeStorage can hold.				Capability: PipeStorage Missing before version 1.1 .		
6	323	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Literal</i> <i>Packet Size</i>	<i>Literal</i> <i>Packet Alignment</i>	<i>Literal</i> <i>Capacity</i>

OpCreatePipeFromPipeStorage Creates a pipe object from a pipe-storage object. <i>Result Type</i> must be OpTypePipe . <i>Pipe Storage</i> must be a pipe-storage object created from OpConstantPipeStorage . <i>Qualifier</i> is the pipe access qualifier.				Capability: PipeStorage Missing before version 1.1 .		
4	324	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Pipe Storage</i>		

OpReadPipeBlockingINTEL Reserved.				Capability: BlockingPipesINTEL Reserved.		
5	5946	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>	

OpWritePipeBlockingINTEL Reserved.				Capability: BlockingPipesINTEL Reserved.		
5	5947	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Packet Size</i>	<id> <i>Packet Alignment</i>	

3.3.24. Non-Uniform Instructions

OpGroupNonUniformElect Result is true only in the tangled invocation with the lowest id within the <i>Execution</i> scope, otherwise result is false. <i>Result Type</i> must be a Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .				Capability: GroupNonUniform Missing before version 1.3.
4	333	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>

OpGroupNonUniformAll Evaluates a predicate for all tangled invocations within the <i>Execution</i> scope, resulting in true if predicate evaluates to true for all tangled invocations within the <i>Execution</i> scope, otherwise the result is false . <i>Result Type</i> must be a Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . <i>Predicate</i> must be a Boolean type . An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .			Capability: GroupNonUniformVote Missing before version 1.3.		
5	334	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<i><id></i> <i>Predicate</i>

OpGroupNonUniformAny Evaluates a predicate for all tangled invocations within the <i>Execution</i> scope, resulting in true if predicate evaluates to true for any tangled invocations within the <i>Execution</i> scope, otherwise the result is false . <i>Result Type</i> must be a Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . <i>Predicate</i> must be a Boolean type . An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .			Capability: GroupNonUniformVote Missing before version 1.3.		
5	335	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Predicate</i>

OpGroupNonUniformAllEqual Evaluates a value for all tangled invocations within the <i>Execution</i> scope. The result is true if <i>Value</i> is equal for all tangled invocations within the <i>Execution</i> scope. Otherwise, the result is false . <i>Result Type</i> must be a Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . <i>Value</i> must be a scalar or vector of floating-point type , integer type , or Boolean type . The compare operation is based on this type, and if it is a floating-point type, an ordered-and-equal compare is used. An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .			Capability: GroupNonUniformVote Missing before version 1.3.		
5	336	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>

OpGroupNonUniformBroadcast					Capability: GroupNonUniformBallot	
Result is the <i>Value</i> of the invocation identified by the id <i>Id</i> to all tangled invocations within the <i>Execution</i> scope.					Missing before version 1.3 .	
<i>Result Type</i> must be a scalar or vector of floating-point type , integer type , or Boolean type .						
<i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup .						
The type of <i>Value</i> must be the same as <i>Result Type</i> .						
<i>Id</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0.						
Before version 1.5 , <i>Id</i> must come from a constant instruction . Starting with version 1.5 , this restriction is lifted. However, behavior is undefined when <i>Id</i> is not dynamically uniform .						
The resulting value is undefined if <i>Id</i> is not part of the scope restricted tangle , or is greater than or equal to the size of the scope.						
An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .						
6	337	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Id</i>

OpGroupNonUniformBroadcastFirst Result is the <i>Value</i> of the invocation from the tangled invocations with the lowest id within the <i>Execution</i> scope to all tangled invocations within the <i>Execution</i> scope. <i>Result Type</i> must be a scalar or vector of floating-point type , integer type , or Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . The type of <i>Value</i> must be the same as <i>Result Type</i> . An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .			Capability: GroupNonUniformBallot Missing before version 1.3 .		
5	338	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>

OpGroupNonUniformBallot

Result is a bitfield value combining the *Predicate* value from all [tangled invocations](#) within the *Execution* scope that execute the same dynamic instance of this instruction. The bit is set to 1 if the corresponding invocation is part of the [tangled invocations](#) within the *Execution* scope and the *Predicate* for that invocation evaluated to true; otherwise, it is set to 0.

Result Type must be a vector of four components of [integer type](#) scalar, whose *Width* operand is 32 and whose *Signedness* operand is 0.

Result is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the [scope](#)) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command.

Predicate must be a [Boolean type](#).

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Capability:
GroupNonUniformBallot

Missing before **version 1.3**.

5	339	<id> Result Type	Result <id>	Scope <id> Execution	<id> Predicate
---	-----	---------------------	-------------	-------------------------	-------------------

OpGroupNonUniformInverseBallot

Evaluates a value for all [tangled invocations](#) within the *Execution* scope, resulting in **true** if the bit in *Value* for the corresponding invocation is set to 1, otherwise the result is **false**.

Result Type must be a [Boolean type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

Value must be a vector of four components of [integer type](#) scalar, whose *Width* operand is 32 and whose *Signedness* operand is 0.

Behavior is undefined unless *Value* is the same for all invocations that execute the same dynamic instance of this instruction.

Value is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the [scope](#)) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

[Capability](#):
GroupNonUniformBallot

[Missing before version 1.3.](#)

5	340	<id> <i>Result Type</i>	<i>Result</i> <id>	<i>Scope</i> <id> <i>Execution</i>	<id> <i>Value</i>
---	-----	----------------------------	--------------------	---------------------------------------	----------------------

OpGroupNonUniformBallotBitExtract

Evaluates a value for all [tangled invocations](#) within the *Execution* scope, resulting in **true** if the bit in *Value* that corresponds to *Index* is set to one, otherwise the result is **false**.

Result Type must be a [Boolean type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

Value must be a vector of four components of [integer type](#) scalar, whose *Width* operand is 32 and whose *Signedness* operand is 0.

Value is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the [scope](#)) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle.

Index must be a scalar of [integer type](#), whose *Signedness* operand is 0.

The resulting value is undefined if *Index* is greater than or equal to the size of the scope.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Capability:
GroupNonUniformBallot

[Missing before version 1.3.](#)

6	341	<id> Result Type	Result <id>	Scope <id> Execution	<id> Value	<id> Index
---	-----	---------------------	-------------	-------------------------	---------------	---------------

OpGroupNonUniformBallotBitCount

Result is the number of bits that are set to 1 in *Value*, considering only the bits in *Value* required to represent all bits of the [scope restricted tangle](#).

Result Type must be a scalar of [integer type](#), whose *Signedness* operand is 0.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is 0.

Value must be a vector of four components of [integer type](#) scalar, whose *Width* operand is 32 and whose *Signedness* operand is 0.

Value is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the [scope](#)) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*') until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Capability:
GroupNonUniformBallot

[Missing before version 1.3.](#)

6	342	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> Value
---	-----	---------------------	-------------	-------------------------	------------------------------	---------------

OpGroupNonUniformBallotFindLSB

Find the least significant bit set to 1 in *Value*, considering only the bits in *Value* required to represent all bits of the [scope restricted tangle](#). If none of the considered bits is set to 1, the resulting value is undefined.

Result Type must be a scalar of [integer type](#), whose *Signedness* operand is 0.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

Value must be a vector of four components of [integer type](#) scalar, whose *Width* operand is 32 and whose *Signedness* operand is 0.

Value is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the [scope](#)) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

[Capability](#):
GroupNonUniformBallot

[Missing before version 1.3.](#)

5	343	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	<id> <i>Value</i>
---	-----	----------------------------	-----------------------------------	--	----------------------

OpGroupNonUniformBallotFindMSB Find the most significant bit set to 1 in <i>Value</i> , considering only the bits in <i>Value</i> required to represent all bits of the scope restricted tangle . If none of the considered bits is set to 1, the resulting value is undefined. <i>Result Type</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0. <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . <i>Value</i> must be a vector of four components of integer type scalar, whose <i>Width</i> operand is 32 and whose <i>Signedness</i> operand is 0. <i>Value</i> is a set of bitfields where the first invocation is represented in the lowest bit of the first vector component and the last (up to the size of the scope) is the higher bit number of the last bitmask needed to represent all bits of the invocations in the scope restricted tangle. An invocation will not execute a dynamic instance of this instruction (X') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .				Capability: GroupNonUniformBallot Missing before version 1.3 .	
5	344	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>

OpGroupNonUniformShuffle				Capability: GroupNonUniformShuffle		
Result is the <i>Value</i> of the invocation identified by the id <i>Id</i> .						
<i>Result Type</i> must be a scalar or vector of floating-point type , integer type , or Boolean type .						
<i>Execution</i> is the scope defining the scope restricted tangle affected by this command.						
The type of <i>Value</i> must be the same as <i>Result Type</i> .						
<i>Id</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0.						
The resulting value is undefined if <i>Id</i> is not part of the scope restricted tangle , or is greater than or equal to the size of the scope.						
An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X' .						
6	345	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Id</i>

OpGroupNonUniformShuffleXor Result is the <i>Value</i> of the invocation identified by the current invocation's id within the scope xor'ed with <i>Mask</i> . <i>Result Type</i> must be a scalar or vector of floating-point type , integer type , or Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . The type of <i>Value</i> must be the same as <i>Result Type</i> . <i>Mask</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0. The resulting value is undefined if current invocation's id within the scope xor'ed with <i>Mask</i> is not part of the scope restricted tangle , or is greater than or equal to the size of the scope. An invocation will not execute a dynamic instance of this instruction (<i>X</i> ') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X '.					Capability: GroupNonUniformShuffle Missing before version 1.3 .	
6	346	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Mask</i>

OpGroupNonUniformShuffleUp Result is the <i>Value</i> of the invocation identified by the current invocation's id within the scope - <i>Delta</i> . <i>Result Type</i> must be a scalar or vector of floating-point type , integer type , or Boolean type . <i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup . The type of <i>Value</i> must be the same as <i>Result Type</i> . <i>Delta</i> must be a scalar of integer type , whose <i>Signedness</i> operand is 0. <i>Delta</i> is treated as unsigned. The resulting value is undefined if <i>Delta</i> is greater than the current invocation's id within the scope or if the identified invocation is not in scope restricted tangle . An invocation will not execute a dynamic instance of this instruction (<i>X</i> ') until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X '.					Capability: GroupNonUniformShuffleRelative Missing before version 1.3 .	
6	347	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Delta</i>

OpGroupNonUniformShuffleDown

Result is the *Value* of the [invocation](#) identified by the current invocation's id within the [scope](#) + *Delta*.

Result Type must be a scalar or vector of [floating-point type](#), [integer type](#), or [Boolean type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The type of *Value* must be the same as *Result Type*.

Delta must be a scalar of [integer type](#), whose *Signedness* operand is 0.

Delta is treated as unsigned. The resulting value is undefined if *Delta* is greater than or equal to the size of the scope, or if the identified invocation is not in [scope restricted tangle](#)

An invocation will not execute a [dynamic instance](#) of this instruction (*X*') until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Capability:
GroupNonUniformShuffleRelative

Missing before **version 1.3**.

6	348	<id> Result Type	Result <id>	Scope <id> Execution	<id> Value	<id> Delta
---	-----	---------------------	-------------	-------------------------	---------------	---------------

OpGroupNonUniformIAdd

An integer add [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is 0. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

[Capability](#):
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3](#).

6 + variable	349	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation Operation	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

OpGroupNonUniformFAdd

A floating point add [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [floating-point type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is 0. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*. The method used to perform the group operation on the contributed *Value*(s) from the [tangled invocations](#) is implementation defined.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Capability:
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3.](#)

6 + variable	350	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> Value	Optional <id> ClusterSize
--------------	-----	---------------------	-------------	-------------------------	---------------------------------	---------------	---------------------------------

OpGroupNonUniformIMul

An integer multiply [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is 1. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

[Capability](#):
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3](#).

6 + variable	351	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

<p>OpGroupNonUniformFMul</p> <p>A floating point multiply group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of floating-point type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is 1. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>. The method used to perform the group operation on the contributed <i>Value</i>(s) from the tangled invocations is implementation defined.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	352	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

OpGroupNonUniformSMin

A signed integer minimum [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is INT_MAX. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

[Capability](#):
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3](#).

6 + variable	353	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

OpGroupNonUniformUMin

An unsigned integer minimum [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#), whose *Signedness* operand is 0.

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is UINT_MAX. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X'*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X'](#).

Capability:
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

Missing before **version 1.3**.

6 + variable	354	<id> Result Type	Result <id>	Scope <id> Execution	Group Operation Operation	<id> Value	Optional <id> ClusterSize
--------------	-----	---------------------	-------------	-------------------------	---------------------------------	---------------	---------------------------------

<p>OpGroupNonUniformFMin</p> <p>A floating point minimum group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of floating-point type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is +INF. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>. The method used to perform the group operation on the contributed <i>Value</i>(s) from the tangled invocations is implementation defined. From the set of <i>Value</i>(s) provided by the tangled invocations within a subgroup, if for any two <i>Values</i> one of them is a NaN, the other is chosen. If all <i>Value</i>(s) that are used by the current invocation are NaN, then the result is an undefined value.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	355	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

OpGroupNonUniformSMax

A signed integer maximum [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is INT_MIN. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Capability:
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

Missing before **version 1.3.**

6 + variable	356	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

<p>OpGroupNonUniformUMax</p> <p>An unsigned integer maximum group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of integer type, whose <i>Signedness</i> operand is 0.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is 0. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	357	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

<p>OpGroupNonUniformFMax</p> <p>A floating point maximum group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of floating-point type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is -INF. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>. The method used to perform the group operation on the contributed <i>Value</i>(s) from the tangled invocations is implementation defined. From the set of <i>Value</i>(s) provided by the tangled invocations within a subgroup, if for any two <i>Values</i> one of them is a NaN, the other is chosen. If all <i>Value</i>(s) that are used by the current invocation are NaN, then the result is an undefined value.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X'</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	358	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

OpGroupNonUniformBitwiseAnd

A bitwise and [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is ~0. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

Capability:

GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3.](#)

6 + variable	359	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

OpGroupNonUniformBitwiseOr

A bitwise or [group operation](#) of all *Value* operands contributed by all [tangled invocations](#) within the *Execution* scope.

Result Type must be a scalar or vector of [integer type](#).

Execution is the [scope](#) defining the [scope restricted tangle](#) affected by this command. It must be **Subgroup**.

The identity *I* for *Operation* is 0. If *Operation* is **ClusteredReduce**, *ClusterSize* must be present.

The type of *Value* must be the same as *Result Type*.

ClusterSize is the size of cluster to use. *ClusterSize* must be a scalar of [integer type](#), whose *Signedness* operand is 0. *ClusterSize* must come from a [constant instruction](#). Behavior is undefined unless *ClusterSize* is at least 1 and a power of 2. If *ClusterSize* is greater than the size of the [scope](#), executing this instruction results in undefined behavior.

An invocation will not execute a [dynamic instance](#) of this instruction (*X*) until all invocations in its [scope restricted tangle](#) have executed all [dynamic instances](#) that are [program-ordered before X](#)'.

[Capability](#):
GroupNonUniformArithmetic,
GroupNonUniformClustered,
GroupNonUniformPartitionedNV

[Missing before version 1.3](#).

6 + variable	360	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>
--------------	-----	----------------------------	-----------------------------------	--	---	----------------------	--

<p>OpGroupNonUniformBitwiseXor</p> <p>A bitwise xor group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of integer type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is 0. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	361	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

<p>OpGroupNonUniformLogicalAnd</p> <p>A logical and group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of Boolean type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is ~0. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	362	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

<p>OpGroupNonUniformLogicalOr</p> <p>A logical or group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of Boolean type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is 0. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	363	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

<p>OpGroupNonUniformLogicalXor</p> <p>A logical xor group operation of all <i>Value</i> operands contributed by all tangled invocations within the <i>Execution</i> scope.</p> <p><i>Result Type</i> must be a scalar or vector of Boolean type.</p> <p><i>Execution</i> is the scope defining the scope restricted tangle affected by this command. It must be Subgroup.</p> <p>The identity <i>I</i> for <i>Operation</i> is 0. If <i>Operation</i> is ClusteredReduce, <i>ClusterSize</i> must be present.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>ClusterSize</i> is the size of cluster to use. <i>ClusterSize</i> must be a scalar of integer type, whose <i>Signedness</i> operand is 0. <i>ClusterSize</i> must come from a constant instruction. Behavior is undefined unless <i>ClusterSize</i> is at least 1 and a power of 2. If <i>ClusterSize</i> is greater than the size of the scope, executing this instruction results in undefined behavior.</p> <p>An invocation will not execute a dynamic instance of this instruction (<i>X</i>) until all invocations in its scope restricted tangle have executed all dynamic instances that are program-ordered before X'.</p>						<p>Capability: GroupNonUniformArithmetic, GroupNonUniformClustered, GroupNonUniformPartitionedNV</p> <p>Missing before version 1.3.</p>	
6 + variable	364	<id> <i>Result Type</i>	Result <id>	Scope <id> <i>Execution</i>	Group Operation <i>Operation</i>	<id> <i>Value</i>	Optional <id> <i>ClusterSize</i>

<p>OpGroupNonUniformQuadBroadcast</p> <p>Result is the <i>Value</i> of the <i>invocation</i> within the <i>quad</i> with a <i>quad index</i> equal to <i>Index</i>.</p> <p><i>Result Type</i> must be a scalar or vector of <i>floating-point type</i>, <i>integer type</i>, or <i>Boolean type</i>.</p> <p><i>Execution</i> is a <i>Scope</i>, but has no effect on the behavior of this instruction. It must be Subgroup.</p> <p>The type of <i>Value</i> must be the same as <i>Result Type</i>.</p> <p><i>Index</i> must be a scalar of <i>integer type</i>, whose <i>Signedness</i> operand is 0.</p> <p>Before version 1.5, <i>Index</i> must come from a <i>constant instruction</i>. Starting with version 1.5, <i>Index</i> must be <i>dynamically uniform</i>.</p> <p>If the value of <i>Index</i> is greater than or equal to 4, or refers to an invocation not part of the <i>tangled invocations</i> within the <i>quad</i>, the resulting value is undefined.</p> <p>An invocation will not execute a <i>dynamic instance</i> of this instruction (<i>X</i>') until all invocations in its quad have executed all <i>dynamic instances</i> that are <i>program-ordered before X</i>'.</p>				<p>Capability: GroupNonUniformQuad</p> <p>Missing before version 1.3.</p>		
6	365	<id> <i>Result Type</i>	<i>Result</i> <id>	<i>Scope</i> <id> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Index</i>

OpGroupNonUniformQuadSwap Swap the <i>Value</i> of the <i>invocation</i> within the <i>quad</i> with another invocation in the quad using <i>Direction</i> . <i>Result Type</i> must be a scalar or vector of <i>floating-point type</i> , <i>integer type</i> , or <i>Boolean type</i> . <i>Execution</i> is a <i>Scope</i> , but has no effect on the behavior of this instruction. It must be Subgroup . The type of <i>Value</i> must be the same as <i>Result Type</i> . <i>Direction</i> is the kind of swap to perform. <i>Direction</i> must be a scalar of <i>integer type</i> , whose <i>Signedness</i> operand is 0. <i>Direction</i> must come from a <i>constant instruction</i> . The value returned in <i>Result</i> is the value provided to <i>Value</i> by another invocation in the same quad scope instance. The invocation providing this value is determined according to <i>Direction</i> . A <i>Direction</i> of 0 indicates a horizontal swap; - Invocations with <i>quad indices</i> of 0 and 1 swap values - Invocations with <i>quad indices</i> of 2 and 3 swap values A <i>Direction</i> of 1 indicates a vertical swap; - Invocations with <i>quad indices</i> of 0 and 2 swap values - Invocations with <i>quad indices</i> of 1 and 3 swap values A <i>Direction</i> of 2 indicates a diagonal swap; - Invocations with <i>quad indices</i> of 0 and 3 swap values - Invocations with <i>quad indices</i> of 1 and 2 swap values <i>Direction</i> must be one of the above values. If a <i>tangled invocation</i> within the <i>quad</i> reads <i>Value</i> from an invocation not part of the <i>tangled invocation</i> within the same <i>quad</i> , the resulting value is undefined. An invocation will not execute a <i>dynamic instance</i> of this instruction (<i>X</i>) until all invocations in its quad have executed all <i>dynamic instances</i> that are <i>program-ordered before X</i> '.					Capability: GroupNonUniformQuad Missing before version 1.3.	
6	366	<id> <i>Result Type</i>	<i>Result <id></i>	<i>Scope <id></i> <i>Execution</i>	<id> <i>Value</i>	<id> <i>Direction</i>

OpGroupNonUniformQuadAllKHR Reserved.					Capability: QuadControlKHR Reserved.	
4	5110	<id> <i>Result Type</i>	<i>Result <id></i>		<id> <i>Predicate</i>	

OpGroupNonUniformQuadAnyKHR Reserved.				Capability: QuadControlKHR Reserved.
4	5111	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Predicate</i>

OpGroupNonUniformPartitionNV Reserved.				Capability: GroupNonUniformPartitionedNV Reserved.
4	5296	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Value</i>

3.3.25. Tensor Instructions

OpTensorReadARM Reserved.					Capability: TensorsARM Reserved.	
5 + variable	4164	<id> Result Type	Result <id>	<id> Tensor	<id> Coordinates	Optional Tensor Operands

OpTensorWriteARM Reserved.					Capability: TensorsARM Reserved.	
4 + variable	4165	<id> Tensor	<id> Coordinates	<id> Object	Optional Tensor Operands	

OpTensorQuerySizeARM Reserved.					Capability: TensorsARM Reserved.	
5	4166	<id> Result Type	Result <id>	<id> Tensor	<id> Dimension	

3.3.26. Graph Instructions

OpGraphConstantARM Reserved.				Capability: GraphARM Reserved.
4	4181	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i>Literal</i> <i>GraphConstantID</i>

OpGraphEntryPointARM Reserved.				Capability: GraphARM Reserved.
3 + variable	4182	<i><id></i> <i>Graph</i>	<i>Literal</i> <i>Name</i>	<i><id>, <id>, ...</i> <i>Interface</i>

OpGraphARM Reserved.				Capability: GraphARM Reserved.
3	4183	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	

OpGraphInputARM Reserved.				Capability: GraphARM Reserved.	
4 + variable	4184	<i><id></i> Result Type	Result <i><id></i>	<i><id></i> InputIndex	<i><id></i> , <i><id></i> , ... ElementIndex

OpGraphSetOutputARM Reserved.				Capability: GraphARM Reserved.
3 + variable	4185	<i><id></i> <i>Value</i>	<i><id></i> <i>OutputIndex</i>	<i><id>, <id>, ...</i> <i>ElementIndex</i>

OpGraphEndARM Reserved.				Capability: GraphARM Reserved.
1				4186

3.3.27. Reserved Instructions

OpTraceRayKHR										Capability: RayTracingKHR		
Reserved.										Reserved.		
1	444	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
2	5	Accel	Ray Flags	Cull Mask	SBT Offset	SBT Stride	Miss Index	Ray Origin	Ray Tmin	Ray Direction	Ray Tmax	Payload

OpExecuteCallableKHR				Capability: RayTracingKHR			
Reserved.				Reserved.			
3	4446			<id> SBT Index			<id> Callable Data

OpConvertUToAccelerationStructureKHR					Capability: RayTracingKHR, RayQueryKHR				
Reserved.					Reserved.				
4	4447		<id> Result Type		Result <id>		<id> Accel		

OpIgnoreIntersectionKHR				Capability: RayTracingKHR			
Reserved.				Reserved.			
1				4448			

OpTerminateRayKHR				Capability: RayTracingKHR			
Reserved.				Reserved.			
1				4449			

OpRayQueryInitializeKHR					Capability: RayQueryKHR				
Reserved.					Reserved.				
9	4473	<id> RayQuery	<id> Accel	<id> RayFlags	<id> CullMask	<id> RayOrigin	<id> RayTMin	<id> RayDirection	<id> RayTMax

OpRayQueryTerminateKHR		Capability: RayQueryKHR
Reserved.		Reserved.
2	4474	<id> RayQuery

OpRayQueryGenerateIntersectionKHR			Capability: RayQueryKHR
Reserved.			Reserved.
3	4475	<id> RayQuery	<id> HitT

OpRayQueryConfirmIntersectionKHR		Capability: RayQueryKHR
Reserved.		Reserved.
2	4476	<id> RayQuery

OpRayQueryProceedKHR				Capability: RayQueryKHR
Reserved.				Reserved.
4	4477	<id> Result Type	Result <id>	<id> RayQuery

OpRayQueryGetIntersectionTypeKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	4479	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpFragmentMaskFetchAMD				Capability: FragmentMaskAMD	
Reserved.				Reserved.	
5	5011	<id> Result Type	Result <id>	<id> Image	<id> Coordinate

OpFragmentFetchAMD Reserved.					Capability: FragmentMaskAMD Reserved.	
6	5012	<id> Result Type	Result <id>	<id> Image	<id> Coordinate	<id> Fragment Index

OpReadClockKHR Reserved.					Capability: ShaderClockKHR Reserved.	
4	5056	<id> Result Type	Result <id>		Scope <id> Scope	

OpAllocateNodePayloadsAMD Reserved.					Capability: ShaderEnqueueAMD Reserved.	
6	5074	<id> Result Type	Result <id>	Scope <id> Visibility	<id> Payload Count	<id> Node Index

OpEnqueueNodePayloadsAMD Reserved.				Capability: ShaderEnqueueAMD Reserved.		
2		5075		<id> Payload Array		

OpTypeNodePayloadArrayAMD Reserved.					Capability: ShaderEnqueueAMD Reserved.	
3	5076		Result <id>		<id> Payload Type	

OpFinishWritingNodePayloadAMD Reserved.					Capability: ShaderEnqueueAMD Reserved.	
4	5078	<id> Result Type	Result <id>		<id> Payload	

OpNodePayloadArrayLengthAMD Reserved.				Capability: ShaderEnqueueAMD Reserved.
4	5090	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Payload Array</i>

OpIsNodePayloadValidAMD				Capability:	
Reserved.				ShaderEnqueueAMD	
Reserved.				Reserved.	
5	5101	<id> Result Type	Result <id>	<id> Payload Type	<id> Node Index

OpConstantStringAMD Reserved.				Capability: ShaderEnqueueAMD Reserved.
3 + variable		5103	<i>Result <id></i>	<i>Literal</i> <i>Literal String</i>

OpSpecConstantStringAMD Reserved.				Capability: ShaderEnqueueAMD Reserved.
3 + variable		5104	<i>Result <id></i>	<i>Literal</i> <i>Literal String</i>

OpHitObjectRecordHitMotionNV												Capability: ShaderInvocationReorderNV, RayTracingMotionBlurNV			
Reserved.												Reserved.			
1	52	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
5	49	Hit Object t	Accel eration Struct ure	Insta nceld	Primit iveld	Geo metry Index	Hit Kind	SBT Recor d Offset	SBT Recor d Stride	Origin	TMin	Direct ion	TMax	Curre nt Time	HitOb ject Attrib utes

OpHitObjectRecordHitWithIndexMotionNV Reserved.											Capability: ShaderInvocationReorderNV, RayTracingMotionBlurNV Reserved.			
1	525	<id> Hit Object	<id> Acceleration Structure	<id> InstanceID	<id> PrimitiveID	<id> GeometryIndex	<id> Hit Kind	<id> SBT Record Index	<id> Origin	<id> TMin	<id> Direction	<id> TMax	<id> Current Time	<id> HitObject Attributes

OpHitObjectRecordMissMotionNV							Capability: ShaderInvocationReorderNV, RayTracingMotionBlurNV	
Reserved.							Reserved.	
8	5251	<id> Hit Object	<id> SBT Index	<id> Origin	<id> TMin	<id> Direction	<id> TMax	<id> Current Time

OpHitObjectGetWorldToObjectNV				Capability:
Reserved.				ShaderInvocationReorderNV
				Reserved.
4	5252	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetObjectToWorldNV				Capability:
Reserved.				ShaderInvocationReorderNV
				Reserved.
4	5253	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetObjectRayDirectionNV				Capability:
Reserved.				ShaderInvocationReorderNV
				Reserved.
4	5254	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetObjectRayOriginNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
4	5255	<id> Result Type		Result <id>	<id> Hit Object

OpHitObjectTraceRayMotionNV											Capability: ShaderInvocationReorderNV, RayTracingMotionBlurNV			
Reserved.											Reserved.			
1 4	525 6	<id> Hit Object	<id> Accel eration Structure	<id> RayFlags	<id> CullMask	<id> SBT Record Offset	<id> SBT Record Stride	<id> Miss Index	<id> Origin	<id> TMin	<id> Direction	<id> TMax	<id> Time	<id> Payload

OpHitObjectGetShaderRecordBufferHandleNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
4	5257	<id> Result Type		Result <id>	<id> Hit Object

OpHitObjectGetShaderBindingTableRecordIndexNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
4	5258	<id> Result Type		Result <id>	<id> Hit Object

OpHitObjectRecordEmptyNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
2	5259		<id> Hit Object	

OpHitObjectTraceRayNV Reserved.										Capability: ShaderInvocationReorderNV Reserved.			
1	526	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
3	0	Hit Object	Accele ration Structu re	RayFla gs	Cullma sk	SBT Recor d Offset	SBT Recor d Stride	Miss Index	Origin	TMin	Directi on	TMax	Payloa d

OpHitObjectRecordHitNV Reserved.										Capability: ShaderInvocationReorderNV Reserved.			
1	526	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
4	1	Hit Object	Accel eration Struct ure	Instan celd	Primiti veld	Geom etryInd ex	Hit Kind	SBT Recor d Offset	SBT Recor d Stride	Origin	TMin	Directi on	TMax HitObj ect Attribu tes

OpHitObjectRecordHitWithIndexNV Reserved.										Capability: ShaderInvocationReorderNV Reserved.			
1	526	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
3	2	Hit Object	Accele ration Structu re	Instan celd	Primiti veld	Geom etryInd ex	Hit Kind	SBT Recor d Index	Origin	TMin	Directi on	TMax	HitObj ect Attribut es

OpHitObjectRecordMissNV Reserved.										Capability: ShaderInvocationReorderNV Reserved.			
7	5263	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
		Hit Object		SBT Index		Origin		TMin		Direction		TMax	

OpHitObjectExecuteShaderNV Reserved.										Capability: ShaderInvocationReorderNV Reserved.			
3		5264	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
			Hit Object									Payload	

OpHitObjectGetCurrentTimeNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5265	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectGetAttributesNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
3	5266	<id> <i>Hit Object</i>		<id> <i>Hit Object Attribute</i>

OpHitObjectGetHitKindNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5267	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectGetPrimitiveIndexNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5268	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectGetGeometryIndexNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5269	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectGetInstanceIdNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5270	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectGetInstanceCustomIndexNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5271	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetWorldRayDirectionNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5272	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetWorldRayOriginNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5273	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetRayTMaxNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5274	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectGetRayTMinNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5275	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectIsEmptyNV Reserved.				Capability: ShaderInvocationReorderNV Reserved.
4	5276	<id> Result Type	Result <id>	<id> Hit Object

OpHitObjectIsHitNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
4	5277	<id> Result Type	Result <id>	<id> Hit Object	

OpHitObjectIsMissNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
4	5278	<id> Result Type	Result <id>	<id> Hit Object	

OpReorderThreadWithHitObjectNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
2 + variable	5279	<id> Hit Object	Optional <id> Hint	Optional <id> Bits	

OpReorderThreadWithHintNV Reserved.					Capability: ShaderInvocationReorderNV Reserved.
3	5280	<id> Hint		<id> Bits	

OpCooperativeVectorMatrixMulNV											Capability: CooperativeVectorNV			
Reserved.											Reserved.			
12 + variab le	5289	<id> Result Type	Result <id>	<id> Input	<id> Input/Interpretation	<id> Matrix	<id> Matrix Offset	<id> Matrix Interpretation	<id> M	<id> K	<id> MemoryLayout	<id> Transpose	Optional <id> Matrix Stride	Optional Cooperative Matrix Operands

OpCooperativeVectorOuterProductAccumulateNV Reserved.							Capability: CooperativeVectorTrainingNV Reserved.	
7 + variable	5290	<id> Pointer	<id> Offset	<id> A	<id> B	<id> MemoryLayout	<id> MatrixInterpretation	Optional <id> MatrixStride

OpCooperativeVectorReduceSumAccumulateNV Reserved.					Capability: CooperativeVectorTrainingNV Reserved.		
4	5291	<id> Pointer		<id> Offset		<id> V	

OpCooperativeVectorMatrixMulAddNV Reserved.													Capability: CooperativeVectorNV Reserved.				
15 + variable	5292	<id> ResultType	Result <id>	<id> Input	<id> InputInterpretation	<id> Matrix	<id> MatrixOffset	<id> MatrixInterpretation	<id> Bias	<id> BiasOffset	<id> BiasInterpretation	<id> M	<id> K	<id> MemoryLayout	<id> Transpose	Optional <id> MatrixStride	Optional CooperativeMatrixOperands

OpEmitMeshTasksEXT				Capability: MeshShadingEXT	
Reserved.				Reserved.	
4 + variable	5294	<id> Group Count X	<id> Group Count Y	<id> Group Count Z	Optional <id> Payload

OpSetMeshOutputsEXT Reserved.				Capability: MeshShadingEXT Reserved.			
3		5295		<id> Vertex Count		<id> Primitive Count	

OpWritePackedPrimitiveIndices4x8NV			Capability: MeshShadingNV Reserved.
3	5299	<id> <i>Index Offset</i>	<id> <i>Packed Indices</i>

OpFetchMicroTriangleVertexPositionNV							Capability: DisplacementMicromapNV	
Reserved.							Reserved.	
8	5300	<id> Result Type	Result <id>	<id> Accel	<id> Instance Id	<id> Geometry Index	<id> Primitive Index	<id> Barycentric

OpFetchMicroTriangleVertexBarycentricNV							Capability: DisplacementMicromapNV Reserved.	
8	5301	<id> Result Type	Result <id>	<id> Accel	<id> Instance Id	<id> Geometry Index	<id> Primitive Index	<id> Barycentric

OpReportIntersectionKHR (OpReportIntersectionNV)				Capability: RayTracingNV, RayTracingKHR	
Reserved.				Reserved.	
5	5334	<id> Result Type	Result <id>	<id> Hit	<id> HitKind

OpIgnoreIntersectionNV		Capability: RayTracingNV Reserved.
1	5335	

OpTerminateRayNV		Capability: RayTracingNV Reserved.
1	5336	

OpTraceNV Reserved.										Capability: RayTracingNV Reserved.		
1	533	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
2	7	Accel	Ray Flags	Cull Mask	SBT Offset	SBT Stride	Miss Index	Ray Origin	Ray Tmin	Ray Direction	Ray Tmax	Payload

OpTraceMotionNV Reserved.										Capability: RayTracingMotionBlurNV Reserved.		
1	533	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
3	8	Accel	Ray Flags	Cull Mask	SBT Offset	SBT Stride	Miss Index	Ray Origin	Ray Tmin	Ray Direction	Ray Tmax	Time
												Payload

OpTraceRayMotionNV Reserved.										Capability: RayTracingMotionBlurNV Reserved.		
1	533	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>	<id>
3	9	Accel	Ray Flags	Cull Mask	SBT Offset	SBT Stride	Miss Index	Ray Origin	Ray Tmin	Ray Direction	Ray Tmax	Time
												Payload

OpRayQueryGetIntersectionTriangleVertexPositionsKHR Reserved.										Capability: RayQueryPositionFetchKHR Reserved.		
5	5340	<id>	<id>			Result <id>			<id>		<id>	
			Result Type						RayQuery		Intersection	

OpExecuteCallableNV Reserved.										Capability: RayTracingNV Reserved.		
3		5344	<id>							<id>		
			SBT Index							Callable DataId		

OpRayQueryGetClusterIdNV Reserved.				Capability: RayTracingClusterAccelerationStructureNV Reserved.	
5	5345	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpHitObjectGetClusterIdNV Reserved.				Capability: RayTracingClusterAccelerationStructureNV Reserved.	
4	5346	<id> Result Type	Result <id>	<id> Hit Object	

OpCooperativeMatrixLoadNV Reserved.						Capability: CooperativeMatrixNV Reserved.	
6 + variable	5359	<id> Result Type	Result <id>	<id> Pointer	<id> Stride	<id> Column Major	Optional Memory Operands

OpCooperativeMatrixStoreNV Reserved.						Capability: CooperativeMatrixNV Reserved.	
5 + variable	5360	<id> Pointer	<id> Object	<id> Stride	<id> Column Major		Optional Memory Operands

OpCooperativeMatrixMulAddNV Reserved.						Capability: CooperativeMatrixNV Reserved.	
6	5361	<id> Result Type	Result <id>	<id> A	<id> B	<id> C	

OpCooperativeMatrixLengthNV Reserved.				Capability: CooperativeMatrixNV Reserved.		
4	5362	<id> Result Type	Result <id>	<id> Type		

OpBeginInvocationInterlockEXT Reserved.			Capability: FragmentShaderSampleInterlockEXT, FragmentShaderPixelInterlockEXT, FragmentShaderShadingRateInterlockEXT Reserved.
1			5364

OpEndInvocationInterlockEXT Reserved.			Capability: FragmentShaderSampleInterlockEXT, FragmentShaderPixelInterlockEXT, FragmentShaderShadingRateInterlockEXT Reserved.
1			5365

OpCreateTensorLayoutNV Reserved.			Capability: TensorAddressingNV Reserved.
3	5372	<i><id></i> <i>Result Type</i>	<i>Result <id></i>

OpTensorLayoutSetDimensionNV				Capability:	
Reserved.				TensorAddressingNV	
				Reserved.	
4 + variable	5373	<i><id></i> Result Type	Result <i><id></i>	<i><id></i> TensorLayout	<i><id></i> , <i><id></i> , ... Dim

OpTensorLayoutSetStrideNV Reserved.				Capability: TensorAddressingNV Reserved.	
4 + variable	5374	<i><id></i> Result Type	Result <i><id></i>	<i><id></i> TensorLayout	<i><id></i> , <i><id></i> , ... Stride

OpTensorLayoutSliceNV Reserved.				Capability: TensorAddressingNV Reserved.	
4 + variable	5375	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>TensorLayout</i>	<i><id>, <id>, ...</i> <i>Operands</i>

OpTensorLayoutSetClampValueNV Reserved.				Capability: TensorAddressingNV Reserved.	
5	5376	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>TensorLayout</i>	<i><id></i> <i>Value</i>

OpCreateTensorViewNV Reserved.				Capability: TensorAddressingNV Reserved.	
3	5377	<i><id></i> <i>Result Type</i>	<i>Result <id></i>		

OpTensorViewSetDimensionNV Reserved.				Capability: TensorAddressingNV Reserved.	
4 + variable	5378	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>TensorView</i>	<i><id>, <id>, ...</i> <i>Dim</i>

OpTensorViewSetStrideNV Reserved.				Capability: TensorAddressingNV Reserved.	
4 + variable	5379	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>TensorView</i>	<i><id>, <id>, ...</i> <i>Stride</i>

OpIsHelperInvocationEXT Reserved.				Capability: DemoteToHelperInvocation Reserved.	
3	5381	<i><id></i> <i>Result Type</i>	<i>Result <id></i>		

OpTensorViewSetClipNV Reserved.							Capability: TensorAddressingNV Reserved.	
8	5382	<id> Result Type	Result <id>	<id> TensorView	<id> ClipRowOffset	<id> ClipRowSpan	<id> ClipColOffset	<id> ClipColSpan

OpTensorLayoutSetBlockSizeNV Reserved.					Capability: TensorAddressingNV Reserved.		
4 + variable		5384	<id> Result Type	Result <id>	<id> TensorLayout	<id>, <id>, ... BlockSize	

OpConvertUToImageNV Reserved.				Capability: BindlessTextureNV Reserved.			
4	5391	<id> Result Type	Result <id>	<id> Operand			

OpConvertUToSamplerNV Reserved.				Capability: BindlessTextureNV Reserved.			
4	5392	<id> Result Type	Result <id>	<id> Operand			

OpConvertImageToUNV Reserved.				Capability: BindlessTextureNV Reserved.			
4	5393	<id> Result Type	Result <id>	<id> Operand			

OpConvertSamplerToUNV Reserved.				Capability: BindlessTextureNV Reserved.			
4	5394	<id> Result Type	Result <id>	<id> Operand			

OpConvertUToSampedImageNV Reserved.				Capability: BindlessTextureNV Reserved.
4	5395	<id> Result Type	Result <id>	<id> Operand

OpConvertSampledImageToUNV Reserved.				Capability: BindlessTextureNV Reserved.
4	5396	<id> Result Type	Result <id>	<id> Operand

OpSamplerImageAddressingModeNV		Capability: BindlessTextureNV
Reserved.		Reserved.
2	5397	<i>Literal</i> <i>Bit Width</i>

OpRayQueryGetIntersectionSpherePositionNV				Capability: RayTracingSpheresGeometryNV	
Reserved.				Reserved.	
5	5427	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionSphereRadiusNV				Capability: RayTracingSpheresGeometryNV	
Reserved.				Reserved.	
5	5428	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionLSSPositionsNV				Capability: RayTracingLinearSweptSpheresGeometryNV	
Reserved.				Reserved.	
5	5429	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionLSSRadiiNV Reserved.				Capability: RayTracingLinearSweptSpheresGeometryNV Reserved.	
5	5430	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionLSSHitValueNV Reserved.				Capability: RayTracingLinearSweptSpheresGeometryNV Reserved.	
5	5431	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpHitObjectGetSpherePositionNV Reserved.				Capability: RayTracingSpheresGeometryNV Reserved.	
4	5432	<id> Result Type	Result <id>	<id> Hit Object	

OpHitObjectGetSphereRadiusNV Reserved.				Capability: RayTracingSpheresGeometryNV Reserved.	
4	5433	<id> Result Type	Result <id>	<id> Hit Object	

OpHitObjectGetLSSPositionsNV Reserved.				Capability: RayTracingLinearSweptSpheresGeometryNV Reserved.	
4	5434	<id> Result Type	Result <id>	<id> Hit Object	

OpHitObjectGetLSSRadiiNV Reserved.				Capability: RayTracingLinearSweptSpheresGeometryNV Reserved.	
4	5435	<id> Result Type	Result <id>	<id> Hit Object	

OpHitObjectIsSphereHitNV Reserved.				Capability: RayTracingSpheresGeometryNV Reserved.
4	5436	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpHitObjectIsLSSHitNV Reserved.				Capability: RayTracingLinearSweptSpheresGeometryNV Reserved.
4	5437	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Hit Object</i>

OpRayQueryIsSphereHitNV				Capability: RayTracingSpheresGeometryNV	
Reserved.				Reserved.	
5	5438	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryIsLSSHitNV				Capability:	
Reserved.				RayTracingLinearSweptSpheresGeometryNV	
				Reserved.	
5	5439	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpUCountLeadingZerosINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.
4	5585	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand</i>

OpUCountTrailingZerosINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.
4	5586	<id> <i>Result Type</i>	<i>Result <id></i>	<id> <i>Operand</i>

OpAbsSubINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5587	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpAbsUSubINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5588	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpIAddSatINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5589	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpUAddSatINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5590	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpIAverageINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5591	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpUAverageINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5592	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpIAverageRoundedINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5593	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpUAverageRoundedINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5594	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpISubSatINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5595	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpUSubSatINTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5596	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpIMul32x16INTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5597	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpUMul32x16INTEL Reserved.				Capability: IntegerFunctions2INTEL Reserved.	
5	5598	<id> Result Type	Result <id>	<id> Operand 1	<id> Operand 2

OpLoopControlIntel		Capability: UnstructuredLoopControlsIntel	
Reserved.		Reserved.	
1 + variable	5887	<i>Literal, Literal, ... Loop Control Parameters</i>	

OpFPGARegIntel				Capability: FPGARegIntel
Reserved.				Reserved.
4	5949	<i><id> Result Type</i>	<i>Result <id></i>	<i><id> Input</i>

OpRayQueryGetRayTMinKHR				Capability: RayQueryKHR
Reserved.				Reserved.
4	6016	<i><id> Result Type</i>	<i>Result <id></i>	<i><id> RayQuery</i>

OpRayQueryGetRayFlagsKHR				Capability: RayQueryKHR
Reserved.				Reserved.
4	6017	<i><id> Result Type</i>	<i>Result <id></i>	<i><id> RayQuery</i>

OpRayQueryGetIntersectionTKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	6018	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionInstanceCustomIndexKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	6019	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionInstanceIdKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6020	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionInstanceShaderBindingTableRecordOffsetKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6021	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionGeometryIndexKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6022	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionPrimitiveIndexKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6023	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionBarycentricsKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6024	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionFrontFaceKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6025	<id> <i>Result Type</i>	<i>Result</i> <id>	<id> <i>RayQuery</i>	<id> <i>Intersection</i>

OpRayQueryGetIntersectionCandidateAABBOpaqueKHR Reserved.				Capability: RayQueryKHR Reserved.
4	6026	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>RayQuery</i>

OpRayQueryGetIntersectionObjectRayDirectionKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	6027	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionObjectRayOriginKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	6028	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetWorldRayDirectionKHR Reserved.				Capability: RayQueryKHR Reserved.
4	6029	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>RayQuery</i>

OpRayQueryGetWorldRayOriginKHR Reserved.				Capability: RayQueryKHR Reserved.
4	6030	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>RayQuery</i>

OpRayQueryGetIntersectionObjectToWorldKHR				Capability: RayQueryKHR	
Reserved.				Reserved.	
5	6031	<id> Result Type	Result <id>	<id> RayQuery	<id> Intersection

OpRayQueryGetIntersectionWorldToObjectKHR Reserved.				Capability: RayQueryKHR Reserved.	
5	6032	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>RayQuery</i>	<i><id></i> <i>Intersection</i>

OpTaskSequenceCreateINTEL Reserved.							Capability: TaskSequenceINTEL Reserved.	
8	6163	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Function</i>	<i>Literal</i> <i>Pipelined</i>	<i>Literal</i> <i>UseStallEn</i> <i>ableCluster</i> <i>s</i>	<i>Literal</i> <i>GetCapacit</i> <i>y</i>	<i>Literal</i> <i>AsyncCapa</i> <i>city</i>

OpTaskSequenceAsyncINTEL Reserved.				Capability: TaskSequenceINTEL Reserved.			
2 + variable		6164	<i><id></i> <i>Sequence</i>	<i><id></i> , <i><id></i> , ... <i>Arguments</i>			

OpTaskSequenceGetINTEL Reserved.				Capability: TaskSequenceINTEL Reserved.			
4	6165	<i><id></i> <i>Result Type</i>	<i>Result <id></i>	<i><id></i> <i>Sequence</i>			

OpTaskSequenceReleaseINTEL Reserved.			Capability: TaskSequenceINTEL Reserved.				
2	6166		<i><id></i> <i>Sequence</i>				

Chapter 4. Appendix A: Changes

4.1. Changes from Version 0.99, Revision 31

- Added the **PushConstant** [Storage Class](#).
- Added [OpIAddCarry](#), [OpISubBorrow](#), [OpUMulExtended](#), and [OpSMulExtended](#).
- Added [OpInBoundsPtrAccessChain](#).
- Added the [Decoration](#) **NoContraction** to prevent combining multiple operations into a single operation (bug 14396).
- Added sparse texturing (14486):
 - Added **OpImageSparse...** for accessing images that might not be resident.
 - Added **MinLod** functionality for accessing images with a minimum level of detail.
- Added back the **Alignment** [Decoration](#), for the **Kernel** capability (14505).
- Added a **Nontemporal** [Memory Operand](#) (14566).
- [Structured control flow](#) changes:
 - Changed structured loops to have a structured continue *Continue Target* in [OpLoopMerge](#) (14422).
 - Added rules for how "fall through" works with **OpSwitch** (13579).
 - Added definitions for what is "inside" a structured control-flow construct (14422).
- Added **SubpassData** [Dim](#) to support input targets written by a previous subpass as an output target (14304). This is also a [Decoration](#) and a [Capability](#), and can be used by some image ops to read the input target.
- Added [OpTypeForwardPointer](#) to establish the Storage Class of a forward reference to a pointer type (13822).
- Improved Debuggability
 - Changed [OpLine](#) to not have a target *<id>*, but instead be placed immediately preceding the instruction(s) it is annotating (13905).
 - Added [OpNoLine](#) to terminate the affect of **OpLine** (13905).
 - Changed [OpSource](#) to include the source code:
 - Allow multiple occurrences.
 - Be mixed in with the [OpString](#) instructions.
 - Optionally consume an [OpString](#) result to say which file it is annotating.
 - Optionally include the source text corresponding to that [OpString](#).
 - Included adding [OpSourceContinued](#) for source text that is too long for a single instruction.
- Added a large number of [Capabilities](#) for subsetting functionality (14520, 14453), including 8-bit integer support for OpenGL kernels.
- Added **VertexIndex** and **InstanceIndex** [BuiltIn Decorations](#) (14255).
- Added **GenericPointer** capability that allows the ability to use the **Generic** [Storage Class](#) (14287).
- Added **IndependentForwardProgress** [Execution Mode](#) (14271).
- Added [OpAtomicFlagClear](#) and [OpAtomicFlagTestAndSet](#) instructions (14315).
- Changed [OpEntryPoint](#) to take a list of **Input** and **Output** *<id>* for declaring the entry point's interface.

- Fixed internal bugs
 - 14411 Added missing documentation for `mad_sat` OpenCL extended instructions (enums existed, just the documentation was missing)
 - 14241 Removed shader capability requirement from **OpImageQueryLevels** and **OpImageQuerySamples**.
 - 14241 Removed unneeded `OpImageQueryDim` instruction.
 - 14241 Filled in *TBD* section for `OpAtomicCompareExchangeWeak`
 - 14366 All **OpSampledImage** must appear before uses of sampled images (and still in the first block of the entry point).
 - 14450 `DeviceEnqueue` capability is required for `OpTypeQueue` and `OpTypeDeviceEvent`
 - 14363 `OpTypePipe` is opaque - moved packet size and alignment to opcodes
 - 14367 `Float16Buffer` capability clarified
 - 14241 Clarified how **OpSampledImage** can be used
 - 14402 Clarified **OpTypeImage** encodings for OpenCL extended instructions
 - 14569 Removed mention of non-existent `OpFunctionDecl`
 - 14372 Clarified usage of **OpGenericPtrMemSemantics**
 - 13801 Clarified the **SpecId Decoration** is just for constants
 - 14447 Changed literal values of **Memory Semantic** enums to match OpenCL/C++11 atomics, and made the **Memory Semantic None** and **Relaxed** be aliases
 - 14637 Removed subgroup scope from **OpGroupAsyncCopy** and **OpGroupWaitEvents**

4.2. Changes from Version 0.99, Revision 32

- Added **UnormInt101010_2** to the **Image Channel Data Type** table.
- Added place holder for C++11 atomic *Consume* Memory Semantics along with an explicit *AcquireRelease* memory semantic.
- Fixed internal bugs:
 - 14690 **OpSwitch** *literal* width (and hence number of operands) is determined by the type of *Selector*, and be rigorous about how sub-32-bit literals are stored.
 - 14485 The client API owns the semantics of built-ins that only have "pass through" semantics WRT SPIR-V.
 - 14862 Removed the **IndependentForwardProgress Execution Mode**.
- Fixed public bugs:
 - 1387 Don't describe result type of **OpImageWrite**.

4.3. Changes from Version 1.00, Revision 1

- Adjusted **Capabilities**:
 - Split geometry-stream functionality into its own **GeometryStreams** capability (14873).
 - Have **InputAttachmentIndex** to depend on **InputAttachment** instead of **Shader** (14797).
 - Merge **AdvancedFormats** and **StorageImageExtendedFormats** into just **StorageImageExtendedFormats** (14824).

- Require **StorageImageReadWithoutFormat** and **StorageImageWriteWithoutFormat** to read and write storage images with an **Unknown Image Format**.
- Removed the **ImageSRGBWrite** capability.
- Clarifications
 - **RelaxedPrecision Decoration** can be applied to **OpFunction** (14662).
- Fixed internal bugs:
 - 14797 The literal argument was missing for the **InputAttachmentIndex Decoration**.
 - 14547 Remove the **FragColor BuiltIn**, so that no implicit broadcast is implied.
 - 13292 Make statements about "Volatile" be more consistent with the memory model specification (non-functional change).
 - 14948 Remove image-"Query" overloading on image/sampled-image type and "fetch" on non-sampled images, by adding the **OpImage** instruction to get the image from a sampled image.
 - 14949 Make consistent placement between **OpSource** and **OpSourceExtension** in the **logical layout** of a module.
 - 14865 Merge **WorkgroupLinearId** with **LocalInvocationId BuiltIn Decorations**.
 - 14806 Include 3D images for **OpImageQuerySize**.
 - 14325 Removed the **Smooth Decoration**.
 - 12771 Make the version word formatted as: "0 | Major Number | Minor Number | 0" in the **physical layout**.
 - 15035 Allow **OpTypeImage** to use a *Depth* operand of 2 for not indicating a depth or non-depth image.
 - 15009 Split the **OpenCL Source Language** into two: **OpenCL_C** and **OpenCL_CPP**.
 - 14683 **OpSampledImage** instructions can only be the consuming block, for scalars, and directly consumed by an image lookup or query instruction.
 - 14325 mutual exclusion validation rules of **Execution Modes** and **Decorations**
 - 15112 add definitions for **invocation**, **dynamically uniform**, and **uniform control flow**.
- Renames
 - **InputTargetIndex Decoration** -> **InputAttachmentIndex**
 - **InputTarget Capability** -> **InputAttachment**
 - **InputTarget Dim** -> **SubpassData**
 - **WorkgroupLocal Storage Class** -> **Workgroup**
 - **WorkgroupGlobal Storage Class** -> **CrossWorkgroup**
 - **PrivateGlobal Storage Class** -> **Private**
 - **OpAsyncGroupCopy** -> **OpGroupAsyncCopy**
 - **OpWaitGroupEvents** -> **OpGroupWaitEvents**
 - **InputTriangles Execution Mode** -> **Triangles**
 - **InputQuads Execution Mode** -> **Quads**
 - **InputIsolines Execution Mode** -> **Isolines**

4.4. Changes from Version 1.00, Revision 2

- Updated example at the end of Section 1 to conform to the KHR_vulkan_glsl extension and treat OpTypeBool as an abstract type.
- Adjusted [Capabilities](#):
 - **MatrixStride** depends on **Matrix** (15234).
 - **Sample**, **SampleId**, **SamplePosition**, and **SampleMask** depend on **SampleRateShading** (15234).
 - **ClipDistance** and **CullDistance** [BuiltIns](#) depend on, respectively, **ClipDistance** and **CullDistance** (1407, 15234).
 - **ViewportIndex** depends on **MultiViewport** (15234).
 - **AtomicCounterMemory** should be the **AtomicStorage** (15234).
 - **Float16** has no dependencies (15234).
 - **Offset** [Decoration](#) should only be for **Shader** (15268).
 - **Generic** [Storage Class](#) is supposed to need the **GenericPointer** [Capability](#) (14287).
 - Remove capability restriction on the **BuiltIn** [Decoration](#) (15248).
- Fixed internal bugs:
 - 15203 Updated description of **SampleMask** [BuiltIn](#) to include "Input or output...", not just "Input..."
 - 15225 Include no re-association as a constraint required by the **NoContraction** [Decoration](#).
 - 15210 Clarify **OpPhi** semantics that operand values only come from parent blocks.
 - 15239 Add [OpImageSparseRead](#), which was missing (supposed to be 12 sparse-image instructions, but only 11 got incorporated, this adds the 12th).
 - 15299 Move [OpUndef](#) back to the Miscellaneous section.
 - 15321 [OpTypeImage](#) does not have a *Depth* restriction when used with **SubpassData**.
 - 14948 Fix the **Lod** [Image Operands](#) to allow both integer and floating-point values.
 - 15275 Clarify specific storage classes allowed for atomic operations under universal validation rules "Atomic access rules".
 - 15501 Restrict [Patch Decoration](#) to one of the tessellation execution models.
 - 15472 Reserved use of **OpImageSparseSampleProjImplicitLod**, **OpImageSparseSampleProjExplicitLod**, **OpImageSparseSampleProjDrefImplicitLod**, and **OpImageSparseSampleProjDrefExplicitLod**.
 - 15459 Clarify what makes different aggregate types in "[Types and Variables](#)".
 - 15426 Don't require [OpQuantizeToF16](#) to preserve NaN patterns.
 - 15418 Don't set both **Acquire** and **Release** bits in [Memory Semantics](#).
 - 15404 [OpFunction](#) *Result* <id> can only be used by **OpFunctionCall**, **OpEntryPoint**, and decoration instructions.
 - 15437 Restrict element type for [OpTypeRuntimeArray](#) by adding a definition of [concrete](#) types.
 - 15403 Clarify [OpTypeFunction](#) can only be consumed by [OpFunction](#) and functions can only return concrete and abstract types.
- Improved accuracy of the opcode word count in each instruction regarding which operands are optional. For sampling operations with explicit LOD, this included not marking the required LOD operands as optional.

- Clarified that when **NonWritable**, **NonReadable**, **Volatile**, and **Coherent Decorations** are applied to the **Uniform** storage class, the **BufferBlock** decoration must be present.
- Fixed external bugs:
 - 1413 (see internal 15275)
 - 1417 Added definitions for block, **dominate**, post dominate, CFG, and **back edge**. Removed use of "dominator tree".

4.5. Changes from Version 1.00, Revision 3

- Added definition of **derivative group**, and use it to say when derivatives are well defined.

4.6. Changes from Version 1.00, Revision 4

- Expanded the list of instructions that may use or return a pointer in the **Logical addressing model**.
- Added missing ABGR **Image Channel Order**

4.7. Changes from Version 1.00, Revision 5

- Khronos SPIR-V issue #27: Removed **Shader** dependency from **SampledBuffer** and **Sampled1D Capabilities**.
- Khronos SPIR-V issue #56: Clarify that the meaning of "read-only" in the **Storage Classes** includes not allowing initializers.
- Khronos SPIR-V issue #57: Clarify "modulo" means "remainder" in **OpFMod**'s description.
- Khronos SPIR-V issue #60: **OpControlBarrier** synchronizes **Output** variables when used in tessellation-control shader.
- Public SPIRV-Headers issue #1: Remove the **Shader** capability requirement from the **Input Storage Class**.
- Public SPIRV-Headers issue #10: Don't say the $(u [, v] [, w] , q)$ has four components, as it can be closed up when the optional ones are missing. Seen in the **projective image** instructions.
- Public SPIRV-Headers issues #12 and #13 and Khronos SPIR-V issue #65: Allow **OpVariable** as an initializer for another **OpVariable** instruction or the *Base* of an **OpSpecConstantOp** with an **AccessChain** opcode.
- Public SPIRV-Headers issues #14: add **Max** enumerants of 0x7FFFFFFF to each of the non-mask enums in the C-based header files.

4.8. Changes from Version 1.00, Revision 6

- Khronos SPIR-V issue #63: Be clear that **OpUndef** can be used in sequence 9 (and is preferred to be) of the **Logical Layout** and can be part of partially-defined **OpConstantComposite**.
- Khronos SPIR-V issue #70: Don't explicitly require operand truncation for integer operations when operating at **RelaxedPrecision**.
- Khronos SPIR-V issue #76: Include **OpNotEqual** in the list of allowed instructions for **OpSpecConstantOp**.
- Khronos SPIR-V issue #79: Remove implication that **OpImageQueryLod** should have a component for the array index.
- Public SPIRV-Headers issue #17: **Decorations NoPerspective**, **Flat**, **Patch**, **Centroid**, and **Sample**

can apply to a top-level member that is itself a structure, so don't disallow it through restrictions to numeric types.

4.9. Changes from Version 1.00, Revision 7

- Khronos SPIR-V issue #69: [OpImageSparseFetch](#) editorial change in summary: include that it is sampled image.
- Khronos SPIR-V issue #74: [OpImageQueryLod](#) requires a sampler.
- Khronos SPIR-V issue #82: Clarification to the [Float16Buffer Capability](#).
- Khronos SPIR-V issue #89: Editorial improvements to [OpMemberDecorate](#) and [OpDecorationGroup](#).

4.10. Changes from Version 1.00, Revision 8

- Add `SPV_KHR_subgroup_vote` tokens.
- Typo: Change "without a sampler" to "with a sampler" for the description of the [SampledBuffer Capability](#).
- Khronos SPIR-V issue #61: Clarification of packet size and alignment on all instructions that use the [Pipes Capability](#).
- Khronos SPIR-V issue #99: Use "invalid" language to replace any "compile-time error" language.
- Khronos SPIR-V issue #55: Distinguish between [branch instructions](#) and [termination instructions](#).
- Khronos SPIR-V issue #94: Add missing [OpSubgroupReadInvocationKHR](#) enumerant.
- Khronos SPIR-V issue #114: Header blocks [strictly dominate](#) their merge blocks.
- Khronos SPIR-V issue #119: [OpSpecConstantOp](#) allows [OpUndef](#) where allowed by its *opcode*.

4.11. Changes from Version 1.00, Revision 9

- Khronos Vulkan issue #652: Remove statements about matrix offsets and padding. These are described correctly in the Vulkan API specifications.
- Khronos SPIR-V issue #113: Remove the "By Default" statements in [FP Rounding Mode](#). These should be properly specified by the client API.
- Add extension enumerants for
 - `SPV_KHR_16bit_storage`
 - `SPV_KHR_device_group`
 - `SPV_KHR_multiview`
 - `SPV_NV_sample_mask_override_coverage`
 - `SPV_NV_geometry_shader_passthrough`
 - `SPV_NV_viewport_array2`
 - `SPV_NV_stereo_view_rendering`
 - `SPV_NVX_multiview_per_view_attributes`

4.12. Changes from Version 1.00, Revision 10

- Add [HLSL source language](#).

- Add **StorageBuffer** [storage class](#).
- Add **StorageBuffer16BitAccess**, **UniformAndStorageBuffer16BitAccess**, **VariablePointersStorageBuffer**, and **VariablePointers** [capabilities](#).
- Khronos SPIR-V issue #163: Be more clear that **OpTypeStruct** allows zero members. Also affects **ArrayStride** and **Offset** decoration [validation rules](#).
- Khronos SPIR-V issue #159: List allowed **AtomicCounter** instructions with the **AtomicStorage** [capability](#) rather than the validation rules.
- Khronos SPIR-V issue #36: Describe more clearly the type of *ND Range* in **OpGetKernelNDRangeSubGroupCount**, **OpGetKernelNDRangeMaxSubGroupSize**, and **OpEnqueueKernel**.
- Khronos SPIR-V issue #128: Be clear the **OpDot** operates only on vectors.
- Khronos SPIR-V issue #80: Loop headers must dominate their continue target. See [Structured Control Flow](#).
- Khronos SPIR-V issue #150 allow **UniformConstant** [storage-class](#) variables to have initializers, depending on the client API.

4.13. Changes from Version 1.00, Revision 11

- Public issue #2: Disallow the **Cube** dimension from use with the **Offset**, **ConstOffset**, and **ConstOffset** [image operands](#).
- Public issue #48: **OpConvertPtrToU** only returns a scalar, not a vector.
- Khronos SPIR-V issue #130: Be more clear which masks are literal and which are not.
- Khronos SPIR-V issue #154: Clarify only one of the listed [Capabilities](#) needs to be declared to use a feature that lists multiple capabilities. The non-declared capabilities need not be supported by the underlying implementation.
- Khronos SPIR-V issue #174: **OpImageDrefGather** and **OpImageSparseDrefGather** return vectors, not scalars.
- Khronos SPIR-V issue #182: The **SampleMask** [built in](#) does not depend on **SampleRateShading**, only **Shader**.
- Khronos SPIR-V issue #183: **OpQuantizeToF16** with too-small magnitude can result in either +0 or -0.
- Khronos SPIR-V issue #203: **OpImageTexelPointer** has 3 components for cube arrays, not 4.
- Khronos SPIR-V issue #217: Clearer language for **OpArrayLength**.
- Khronos SPIR-V issue #213: **Image Operand LoD** is not used by query operations.
- Khronos SPIR-V issue #223: **OpPhi** has exactly one parent operand per parent block.
- Khronos SPIR-V issue #212: In the [Validation Rules](#), make clear a pointer can be an operand in an extended instruction set.
- Add extension enumerants for
 - SPV_AMD_shader_ballot
 - SPV_KHR_post_depth_coverage
 - SPV_AMD_shader_explicit_vertex_parameter
 - SPV_EXT_shader_stencil_export
 - SPV_INTEL_subgroups

4.14. Changes from Version 1.00

- Moved version number to SPIR-V 1.1
- New functionality:
 - Bug 14202 named barriers:
 - Added the **NamedBarrier** [Capability](#).
 - Added the instructions: **OpTypeNamedBarrier**, **OpNamedBarrierInitialize**, and **OpMemoryNamedBarrier**.
 - Bug 14201 subgroup dispatch:
 - Added the **SubgroupDispatch** [Capability](#).
 - Added the instructions: **OpGetKernelLocalSizeForSubgroupCount** and **OpGetKernelMaxNumSubgroups**.
 - Added **SubgroupSize** and **SubgroupsPerWorkgroup** [Execution Modes](#).
 - Bug 14441 program-scope pipes:
 - Added the **PipeStorage** [Capability](#).
 - Added Instructions: **OpTypePipeStorage**, **OpConstantPipeStorage**, and **OpCreatePipeFromPipeStorage**.
 - Bug 15434 Added the **OpSizeOf** instruction.
 - Bug 15024 support for OpenCL-C++ ivdep loop attribute:
 - Added **DependencyInfinite** and **DependencyLength** [Loop Controls](#).
 - Updated **OpLoopMerge** to support these.
 - Bug 14022 Added **Initializer** and **Finalizer** and [Execution Modes](#).
 - Bug 15539 Added the **MaxByteOffset** [Decoration](#).
 - Bug 15073 Added the **Kernel** [Capability](#) to the **SpecId** [Decoration](#).
 - Bug 14828 Added the **OpModuleProcessed** instruction.
- Fixed internal bugs:
 - Bug 15481 Clarification on alignment and size operands for pipe operands

4.15. Changes from Version 1.1, Revision 1

- Incorporated bug fixes from Revision 6 of Version 1.00 (see section 4.7. Changes from Version 1.00, Revision 5).

4.16. Changes from Version 1.1, Revision 2

- Incorporated bug fixes from Revision 7 of Version 1.00 (see section 4.8. Changes from Version 1.00, Revision 6).

4.17. Changes from Version 1.1, Revision 3

- Incorporated bug fixes from Revision 8 of Version 1.00 (see section 4.9. Changes from Version 1.00, Revision 7).

4.18. Changes from Version 1.1, Revision 4

- Incorporated bug fixes from Revision 9 of Version 1.00 (see section 4.10. Changes from Version 1.00, Revision 8).

4.19. Changes from Version 1.1, Revision 5

- Incorporated changes from Revision 10 of Version 1.00 (see section 4.11. Changes from Version 1.00, Revision 9).

4.20. Changes from Version 1.1, Revision 6

- Incorporated changes from Revision 11 of Version 1.00 (see section 4.12. Changes from Version 1.00, Revision 10).

4.21. Changes from Version 1.1, Revision 7

- Incorporated changes from Revision 12 of Version 1.00 (see section 4.13. Changes from Version 1.00, Revision 11).
- State where all **OpModuleProcessed** belong, in [the logical layout](#).

4.22. Changes from Version 1.1

- Moved version number to SPIR-V 1.2
- New functionality:
 - Added **OpExecutionModelId** to allow using an `<id>` to set the [execution modes](#) **SubgroupsPerWorkgroupId**, **LocalSizeId**, and **LocalSizeHintId**.
 - Added **OpDecoratId** to allow using an `<id>` to set the [decorations](#) **AlignmentId** and **MaxByteOffsetId**.

4.23. Changes from Version 1.2, Revision 1

- Incorporated changes from Revision 12 of Version 1.00 (see section 4.13. Changes from Version 1.00, Revision 11).
- Incorporated changes from Revision 8 of Version 1.1 (see section 4.21. Changes from Version 1.1, Revision 7).

4.24. Changes from Version 1.2, Revision 2

- Combine the 1.0, 1.1, and 1.2 specifications, making a [unified specification](#). The previous 1.0, 1.1, and 1.2 specifications are replaced with this one unified specification.

4.25. Changes from Version 1.2, Revision 3

Fixed Khronos-internal issues:

- #249: Improve description of **OpTranspose**.
- #251: Undefined values in **OpUndef** include abstract and opaque values.

- #258: Deprecate [OpAtomicCompareExchangeWeak](#) in favor of [OpAtomicCompareExchange](#).
- #241: Use "invalid" instead of "compile-time" error for **ConstOffsets**.
- #248: [OpImageSparseRead](#) is not for **SubpassData**.
- #257: Allow **OpImageSparseFetch** and **OpImageSparseRead** with the **Sample** [image operands](#).
- #229: Some sensible constraints on branch hints for [OpBranchConditional](#).
- #236: [OpVariable](#)'s storage class must match storage class of the pointer type.
- #216: Can [decorate pointer types](#) with **Coherent** and **Volatile**.
- #247: Don't say [Scope <id>](#) is a mask; it is not.
- #254: Remove [validation](#) rules about the types atomic instructions can operate on. These rules belong instead to the client API.
- #265: [OpGroupDecorate](#) cannot target an **OpDecorationGroup**.

4.26. Changes from Version 1.2

- Moved version number to SPIR-V 1.3
- New functionality:
 - Added subgroup operations:
 - the [OpGroupNonUniform](#) instructions and [capabilities](#).
 - **Subgroup-mask** [built-in decorations](#).
 - Khronos SPIR-V issue #125, #138, #196: Removed capabilities from the [rounding modes](#).
 - Khronos SPIR-V issue #110: Removed the execution-model restrictions from [OpControlBarrier](#).
- Incorporated the following extensions:
 - SPV_KHR_shader_draw_parameters
 - SPV_KHR_16bit_storage
 - SPV_KHR_device_group
 - SPV_KHR_multiview
 - SPV_KHR_storage_buffer_storage_class
 - SPV_KHR_variable_pointers
- Reserved symbols for
 - SPV_GOOGLE_decorate_string
 - SPV_GOOGLE_hlsl_functionality1
 - SPV_AMD_gpu_shader_half_float_fetch
- Added [deprecation model](#).

4.27. Changes from Version 1.3, Revision 1

- Fixed Issues:
 - Public SPIRV-Headers PR #73: Add missing fields for some NVIDIA-specific tokens.
 - Khronos SPIR-V Issue #202: [Shader Validation](#): Be clear that arrays of blocks set by the client API cannot have an **ArrayStride**.

- Khronos SPIR-V Issue #210: Clarify the *Result Type* of **OpSampledImage**.
- Khronos SPIR-V Issue #211: State that **Derivative** instructions only work on 32-bit width components.
- Khronos SPIR-V Issue #239: Clarify **OpImageFetch** is for an image whose *Sampled* operand is 1.
- Khronos SPIR-V Issue #256: **OpAtomicCompareExchange** does not store if comparison fails.
- Khronos SPIR-V Issue #269: Be more clear which bits are mutually exclusive for **memory semantics**.
- Khronos SPIR-V Issue #278: Delete **OpTypeRuntimeArray** restriction on storage classes, as this is already covered by the client API.
- Khronos SPIR-V Issue #279:
 - Add section expository section 2.8.1 "Unsigned Versus Signed Integers".
 - As expected, **OpUConvert** can have vector *Result Type*.
- Khronos SPIR-V Issue #280: **OpImageQuerySizeLod** and **OpImageQueryLevels** can be limited by the client API.
- Khronos SPIR-V Issue #285: Remove **Kernel** as a **capability** implicitly declared by **Int8**.
- Khronos SPIR-V Issue #290: Clarify implicit declaration of **capabilities**, in part by changing the column heading to "Implicitly Declares".
- Khronos SPIR-V Issues #295: Explicitly say blocks cannot be nested in blocks, in the **validation** section. (This was already indirectly required.)
- Khronos SPIR-V Issue #299: Add the **ImageGatherExtended** capability to **ConstOffsets** in **the image operands section**.
- Khronos SPIR-V Issues #303 and #304: **OpGroupNonUniformBallotBitExtract** documentation: add **Result Type** and fix **Index** parameter.
- Khronos SPIR-V Issue #310: Remove instruction word count from the **Limits** table, as it is already intrinsically limited.
- Khronos SPIR-V Issue #313: Move the **FPRoundingMode**-decoration validation rule to the **shader validation** section (not a universal rule). Also, include the **StorageBuffer** storage class in this rule.

4.28. Changes from Version 1.3, Revision 2

- New enumarents:
 - For SPV_KHR_8bit_storage
- Fixed Issues:
 - Add definition of **Memory Object Declaration**.
 - Khronos SPIR-V Issue #275: Clarify the meaning of **Aliased** and **Restrict** in the **Aliasing** section.
 - Khronos SPIR-V Issue #315: Be more specific about where many **decorations** are allowed, particularly for **OpFunctionParameter**. Includes being clear that the **BuiltIn** decoration does not apply to **OpFunctionParamater**.
 - Khronos SPIR-V Issue #348: Clarify **remainder** descriptions in **OpFRem**, **OpFMod**, **OpSRem**, and **OpSMod**.
 - Khronos SPIR-V Issue #342: State the **DepthReplacing** **execution-mode** behavior more specifically.
 - Khronos SPIR-V Issue #341: More specific wording for depth-hint **execution modes** **DepthGreater**, **DepthLess**, and **DepthUnchanged**.

- Khronos SPIR-V Issues #276 and #311: Take more care with unreachable blocks in [structured control flow](#) and how to branch into a construct.
- Khronos SPIR-V Issue #320: Include **OpExecutionModelId** in the [logical layout](#).
- Khronos SPIR-V Issue #238: Fix description of **OpImageQuerySize** to correct *Sampled Type* -> *Sampled* and list the correct set of dimensions.
- Khronos SPIR-V Issue #346: Remove ordered rule for structures in the [memory layout](#): Vulkan allows out-of-order **Offset** layouts.
- Khronos SPIR-V Issue #322: Allow **OpImageQuerySize** to query the size of a **NonReadable** image.
- Khronos SPIR-V Issue #244: Be more clear about the connections between [dimensionalities](#) and capabilities, and in referring to them from **OpImageRead** and **OpImageWrite**.
- Khronos SPIR-V Issue #333: Be clear about overflow behavior for **OpIAdd**, **OpISub**, and **OpIMul**.

4.29. Changes from Version 1.3, Revision 3

- Add enumerants for
 - SPV_KHR_vulkan_memory_model
- Fixed Issues:
 - Typo: say **OpMatrixTimesVector** is **Matrix X Vector**.
 - Update on Khronos SPIR-V issue #244: Added **Shader** and **Kernel** capabilities to the **2D dimensionality**.
 - Khronos SPIR-V Issue #317: Clarify that the **Uniform decoration** should apply only to objects, and that the [dynamic instance](#) of the object is the same, rather than at the consumer usage.
 - Khronos SPIR-V Issue #335: Clarify and correct [when it is valid](#) for pointers to be operands to **OpFunctionCall**. Corrections are believed to be consistent with existing front-end and back-end support.
 - Khronos SPIR-V Issue #344: don't include inactive invocations in what makes the result of **OpGroupNonUniformBallotBitExtract** undefined.

4.30. Changes from Version 1.3, Revision 4

- Add enumerants for
 - SPV_NV_fragment_shader_barycentric
 - SPV_NV_compute_shader_derivatives
 - SPV_NV_shader_image_footprint
 - SPV_NV_shading_rate
 - SPV_NV_mesh_shader
 - SPV_NVX_Raytracing
- Formatting: Removed **Enabling Extensions** column and instead list the extensions in the **Enabling Capabilities** column.

4.31. Changes from Version 1.3, Revision 5

- Reserve Tokens for:

- SPV_KHR_no_integer_wrap_decoration
- SPV_KHR_float_controls
- Fixed Issues:
 - Khronos SPIR-V Issue #352: Remove from **OpFunction** the statement limiting the use its result. This does not result in any change in intent; it only avoids any past and potential future contradictions.
 - Khronos SPIR-V Issue #308: Don't allow runtime-sized arrays to be loaded or copied by **OpLoad** or **OpCopyMemory**.
 - Include back-edge blocks in the list of blocks that can branch outside their own construct in the [structured control-flow rules](#).
 - Khronos OpenGL API issue #77: Clarify the **OriginUpperLeft** and **OriginLowerLeft** [execution modes](#) apply only to **FragCoord**.
 - State the **XfbStride** and **Stream** restrictions in the [Universal Validation Rules](#).
 - Khronos SPIR-V Issue #357: The *Memory Operands* of **OpCopyMemory** and **OpCopyMemorySized** applies to both *Source* and *Target*.
 - Khronos SPIR-V Issue #385: Be more clear what type *<id>* must be the same in **OpCopyMemory**.
 - Khronos SPIR-V Issue #359: **OpAccessChain** and **OpPtrAccessChain** do indexing with signed indexes, and **OpPtrAccessChain** is allowed to compute addresses of elements one past the end of an array.
 - Khronos SPIR-V Issue #367: [General validation rules](#) allow the **Function** storage class for atomic access, while the [shader-specific validation rules](#) do not.
 - Khronos SPIR-V Issue #382: In **OpTypeFunction**, disallow parameter types from being **OpTypeVoid**.
 - Khronos SPIR-V Issue #374: [Built-in](#) decorations can also apply to a constant instruction.
- Editorial:
 - Make it more clear in **OpVariable** what *Storage Classes* must be the same.
 - Remove references to specific APIs, and instead generally refer only to "client API"s. Note that the previous lists of APIs was nonnormative.
 - State the **FPRoundingMode** decoration rule more clearly in the section listing [Validation Rules for Shader Capabilities](#).
 - Don't say "value preserving" in the [Conversion](#) instructions. These now convert the "value numerically".
 - State variable-pointer [validation rules](#) more clearly.

4.32. Changes from Version 1.3, Revision 6

- Reserve Tokens for:
 - SPV_INTEL_media_block_io
 - SPV_NV_cooperative_matrix
 - SPV_INTEL_device_side_avc_motion_estimation, partially. See the SPV_INTEL_device_side_avc_motion_estimation extension specification for a full listing of tokens.
- Fixed Issues:
 - Khronos SPIR-V Issue #406: [Scope](#) values must come from the table of scope values.

- Khronos SPIR-V Issue #419: [Validation rules](#) include **AtomicCounter** in the list of storage classes allowed for pointer operands to an **OpFunctionCall**.
- Khronos SPIR-V Issue #325: **OpPhi** clarifications regarding parent dominance, in the instruction and the [validation rules](#), and forward references in the [Logical Layout section](#).
- Khronos SPIR-V Issue #415: Remove the non-writable storage classes **PushConstant** and **Input** from the **FPRoundingMode** decoration [shader validation rule](#).
- Khronos SPIR-V Issue #404: Clarify when **OpGroupNonUniformShuffleXor**, **OpGroupNonUniformShuffleUp**, and **OpGroupNonUniformShuffleDown** are valid or result in undefined values.
- Khronos SPIR-V Issue #393: Be more clear that **OpConvertUToPtr** and **OpConvertPtrToU** operate only on unsigned scalar integers.
- Khronos SPIR-V Issue #416: Result are undefined for all [Shift instructions](#) for shifts amounts equal to the bit width of the operand.
- Khronos SPIR-V Issue #399: Refine the definition of a [variable pointer](#), particularly for function parameters receiving a variable pointer.
- Khronos SPIR-V Issue #441: Clarify that [atomic instruction's Scope <id>](#) must be a valid memory scope. More generally, all *Scope <id>* operands are now either *Memory* or *Execution*.
- Khronos SPIR-V Issue #426: Be more direct about undefined behavior for non-uniform control flow in **OpControlBarrier** and the **OpGroup...** instructions that discuss this.
- Deprecate
 - Khronos SPIR-V Issue #429: Deprecate **OpDecorationGroup**, **OpGroupDecorate**, and **OpGroupMemberDecorate**
- Editorial
 - Add more clarity that the full [client API describes the execution environment](#) (there is not a separate specification from the client API specification).

4.33. Changes from Version 1.3, Revision 7

- Fixed Issues:
 - Khronos SPIR-V Issue #371: Restrict [intermediate object](#) types to variable types allowed at global scope. See [shader validation data rules](#).
 - Khronos SPIR-V Issue #408: (Re)allow the [decorations](#) **Volatile**, **Coherent**, **NonWritable**, and **NonReadable** on members of blocks. (Temporarily dropping this functionality was accidental/clerical; intent is that it has always been present.)
 - Khronos SPIR-V Issue #418: Add statements about undefinedness and how NaNs are mixed to **OpGroupNonUniformFAdd**, **OpGroupNonUniformFMul**, **OpGroupNonUniformFMin**, and **OpGroupNonUniformFMax**.
 - Khronos SPIR-V Issue #435: Expand the [universal validation](#) rule for variable pointers and matrices to also disallow pointing within a matrix.
 - Khronos SPIR-V Issue #447: Remove implication that **OpPtrAccessChain** obeys an **ArrayStride** decoration in storage classes laid out by the implementation.
 - Khronos SPIR-V Issue #450: Allow pointers to **OpFunctionCall** to be pointers to an element of an array of samplers or images. See the [universal validation rules](#) under the **Logical** addressing model without variable pointers.
 - Khronos SPIR-V Issue #452: **OpGroupNonUniformAllEqual** uses ordered compares for floating-point values.

- Khronos SPIR-V Issue #454: Add **OpExecutionModelId** to the list of allowed forward references in the [Logical Layout of a Module](#).

4.34. Changes from Version 1.3

- New Functionality:
 - Public issue #35: **OpEntryPoint** must list all global variables in the interface. Additionally, duplication in the list is not allowed.
 - Khronos SPIR-V Issue #140: Generalize **OpSelect** to select between two objects.
 - Khronos SPIR-V Issue #156: Add **OpUConvert** to the list of required opcodes in **OpSpecConstantOp**.
 - Khronos SPIR-V Issue #345: Generalize the **NonWritable** [decoration](#) to include **Private** and **Function** storage classes. This helps identify lookup tables.
 - Khronos SPIR-V Issue #84: Add **OpCopyLogical** to copy similar but unequal types.
 - Khronos SPIR-V Issue #170: Add **OpPtrEqual** and **OpPtrNotEqual** to compare pointers.
 - Khronos SPIR-V Issue #362: Add **OpPtrDiff** to count the number of elements between two element pointers.
 - Khronos SPIR-V Issue #332: Add **SignExtend** and **ZeroExtend** [image operands](#).
 - Khronos SPIR-V Issue #340: Add the **UniformId** [decoration](#), which takes a *Scope* operand.
 - Khronos SPIR-V Issue #112: Add iteration-control [loop controls](#).
 - Khronos SPIR-V Issue #366: Change *Memory Access* operands and the **Memory Access** section to now be *Memory Operands* and the **Memory Operands** section.
 - Khronos SPIR-V Issue #357: Allow **OpCopyMemory** and **OpCopyMemorySized** to have [Memory Operands](#) for both their *Source* and *Target*.
- New Extensions Incorporated into SPIR-V 1.4:
 - SPV_KHR_no_integer_wrap_decoration. See **NoSignedWrap** and **NoUnsignedWrap** [decorations](#) and [universal validation](#) decoration rules.
 - SPV_GOOGLE_decorate_string. See **OpDecorateString** and **OpMemberDecorateString**.
 - SPV_GOOGLE_hlsl_functionality1. See **CounterBuffer** and **UserSemantic** [decorations](#).
 - SPV_KHR_float_controls. See **DenormPreserve**, **DenormFlushToZero**, **SignedZeroInfNanPreserve**, **RoundingModeRTE**, and **RoundingModeRTZ** [execution modes](#) and [capabilities](#).
- Removed:
 - Khronos SPIR-V Issue #437: Removed **OpAtomicCompareExchangeWeak**, and the **BufferBlock** [decoration](#).

4.35. Changes from Version 1.4, Revision 1

- GitHub SPIRV-Registry Issue #25: Remove validation rule for simultaneous use of **RowMajor** and **ColMajor**, instead stating this in the [decoration](#) cells themselves.
- Khronos Issue #319: Bring in fixes to the SPV_KHR_16bit_storage extension. See the **StorageBuffer16BitAccess** and the related 16-bit [capabilities](#).
- Khronos Issue #363: **OpTypeBool** can be used in the Input and Output storage classes, but the client APIs still only allow built-in Boolean variables (e.g. FrontFacing), not user variables.

- Khronos Issue #432: Remove the untrue expository statement "**OpFunction** is the only valid use of **OpTypeFunction**."
- Khronos Issue #465: Distinguish between the **Groups** [capability](#) and the [Group and Subgroup instructions](#).
- Khronos Issue #484: Have [OpTypeArray](#) and [OpTypeStruct](#) point to their definitions.
- Khronos Issue #477: Include 0.0 in the range of required values for **RelaxedPrecision** and other minor clarifications in [the relaxed-precision section](#) regarding floating-point precision.
- Khronos Issue #226: Be more clear about explicit level-of-detail being either **Lod** or **Grad** throughout the sampling instructions, and that **ConstOffset**, **Offset**, and **ConstOffsets** are mutually exclusive in [the image operand's descriptions](#).
- Khronos Issue #390: The **Volatile** [decoration](#) does not guarantee each invocation performs the access.
- Reserved New Tokens for:
 - SPV_EXT_fragment_shader_interlock
 - SPV_NV_shader_sm_builtins
 - SPV_INTEL_shader_integer_functions2
 - SPV_EXT_demote_to_helper_invocation
 - SPV_KHR_shader_clock
 - SPV_GOOGLE_user_type
 - **Volatile**, for SPV_KHR_vulkan_memory_model

4.36. Changes from Version 1.4

- Extensions Incorporated into SPIR-V 1.5:
 - SPV_KHR_8bit_storage
 - SPV_EXT_descriptor_indexing
 - SPV_EXT_shader_viewport_index_layer, with changes: Replaced the single **ShaderViewportIndexLayerEXT** capability with the two new [capabilities](#) **ShaderViewportIndex** and **ShaderLayer**. Declaring both is equivalent to declaring **ShaderViewportIndexLayerEXT**.
 - SPV_EXT_physical_storage_buffer and SPV_KHR_physical_storage_buffer
 - SPV_KHR_vulkan_memory_model
- Khronos Issue #402: Relax [OpGroupNonUniformBroadcast](#) *Id* from constant to dynamically uniform, starting with version 1.5.
- Khronos Issue #493: Relax [OpGroupNonUniformQuadBroadcast](#) *Id* from constant to dynamically uniform, starting with version 1.5.
- Khronos Issue #494: Update the [Dynamically Uniform](#) definition to say that the invocation group is the set of invocations, *unless otherwise stated*.
- Khronos Issue #485: When [RelaxedPrecision](#) is applied to a numerical instruction, the operands may be truncated.

4.37. Changes from Version 1.5, Revision 1

- Khronos Issue #511: Allow non-execution non-memory scopes in the introduction to the [Scope <id> section](#).

- Khronos MR !147: Fix **OpFNegate** so it handles 0.0f properly
- Khronos Issue #502: **OpAccessChain** array indexes must be in-bounds for logical pointer types.
- Khronos Issue #518: Include both **VariablePointers** and **VariablePointersStorageBuffer** capabilities in the **validation** rules when discussing variable pointer rules.
- Khronos Issue #496: Allow **Invariant** to **decorate** a block member.
- Khronos Issue #469: Disallow **OpConstantNull** result and **OpPtrEqual**, **OpPtrNotEqual**, and **OpPtrDiff** operands from being pointers into the **PhysicalStorageBuffer** storage class. See the **PhysicalStorageBuffer validation rules**.
- Khronos Issue #425: Clarify what variables can allocate pointers, in the **validation rules**, based on the declarations of the **VariablePointers** or **VariablePointersStorageBuffer** capabilities.
- Khronos Issue #442: Add a note pointing out where **signedness** has some semantic meaning.
- Khronos Issue #498: Relaxed the set of allowed types for some **Group** and **Subgroup** instructions.
- Khronos Issue #500: Deprecate **OpLessOrGreater** in favor of **OpFOrdNotEqual**.
- Khronos Issue #354: Rationalize **literals** throughout the specification. Remove "immediate" as a separate definition. Be more rigid about a single literal mapping to one or more operands, and that the instruction description defines the type of the literal.
- Khronos Issue #479: Disallow intermediate aggregate types that could not be used to declare global variables, and disallow all types that can't be used for declaring variables. See the **shader validation "Type Rules"**. Also, more strongly state that intermediate values don't form a storage class, in the introduction to **storage classes**.
- Khronos Issue #78: Use a more correct definition of **back edge**.
- Khronos Issue #492: Overflow with **OpSDiv**, **OpSRem**, and **OpSMod** results in undefined behavior.

4.38. Changes from Version 1.5, Revision 2

- Reserve enumerants for SPV_KHR_ray_query and SPV_KHR_ray_tracing.
- Khronos MR #164: Subtract all exits from what a construct contains, not just the construct's merge block. See **the Structured Control Flow section**.
- Khronos Issues #394 and #473: More clearly state that the **<id>** declared by an **OpTypeForwardPointer** can be consumed by any **type-declaration instruction** that can legally consume the type of **<id>**. Also consolidated the rules for this within the instruction itself.
- Khronos Vulkan Issue #1951: Clarify that the **SampledImageArrayDynamicIndexing** **capability** applies to dynamic indexing of image, sampler and sampled image objects.
- Khronos Issue #523: Label as memory **Scope** the additional operand for each of
 - **MakeTexelAvailable** and **MakeTexelVisible** **image operands**, and
 - **MakePointerAvailable** and **MakePointerVisible** **memory operands**.
- Khronos Issue #529: Allow the scope of **uniform control flow** to be defined by the client API.
- Khronos Issue #530: Allow the definition of **derivative group** to be set by the client API.
- Khronos Issue #293: Editorial simplification and clarification of different types under **Types and Variables**.
- Khronos Issue #506: Add to the definition of **Pure** under **Function Control** that assuming it computes the same results also requires the same global state.
- Khronos Issue #539: Clarify out-of-bounds indexes for **OpAccessChain**.
- Khronos Issue #550: Include **OpUndef** in the allowed constituents for **OpSpecConstantComposite**.

- Khronos Issue #389: Be more clear which instructions can be updated with a specialization constant in [the specialization section](#).
- Khronos Issue #544: Be more concise with **OpLabel** language.
- Khronos Issue #245: State that D_{ref} operands must be 32-bit scalar floats in the [image instructions](#).
- Khronos Issue #457: Change rule for **OpUnreachable** to being that behavior is undefined if it is executed.
- Khronos Issue #231: Explicitly state that the component numbers 0, 1, 2, and 3 are 32-bit scalar integers for **OpImageGather** and **OpImageSparseGather**.
- Khronos Issue #534: State where **OpNoLine** can be in the [logical layout](#) and with **OpPhi**.
- Khronos MR #168: Add definitions of [quad](#) and [quad index](#), used by **OpGroupNonUniformQuadBroadcast** and **OpGroupNonUniformQuadSwap**.

4.39. Changes from Version 1.5, Revision 3

- Reserve enumerants for the extensions
 - SPV_INTEL_fpga_loop_controls
 - SPV_INTEL_blocking_pipes
 - SPV_INTEL_unstructured_loop_controls
 - SPV_INTEL_fpga_reg
 - SPV_INTEL_fpga_memory_attributes
 - SPV_INTEL_kernel_attributes
 - SPV_INTEL_function_pointers
 - SPV_EXT_shader_image_int64
 - SPV_KHR_fragment_shading_rate
 - SPV_EXT_shader_atomic_float_add
- Establish formal meanings for validity (being statically expressed) and behavior (regarding dynamic execution), in [Validity and Defined Behavior](#). This also changed a number of uses of these terms throughout the specifications to be consistent with these definitions.
 - Main issue for this: Khronos issue #540.
 - Addresses Khronos issues #542, #540, #545, #546, #547, and #548.
 - Khronos issue #491: For **OpConvertFToU** and **OpConvertFToS**, behavior is undefined if *Result Type* is not wide enough to hold the converted value.
 - Khronos issue #591: Module validity does not depend on the default values of [specialization constants](#).
- Fix Khronos issues:
 - #214: LoD and gather [Image Instructions](#) need non-multisampled images (*MS* of 0), while others that provide a [Sample Image Operand](#) need a multisampled image (*MS* of 1).
 - #324: For several [Capabilities](#), explicitly list the values **OpTypeImage** has for *Sampled*, instead of saying sampled or unsampled.
 - #361: Stop requiring **OpTypeRuntimeArray** to be concrete, in the description of **OpTypeRuntimeArray**. (This may still be restricted elsewhere though.)
 - #553: Add definition of a [tangled instruction](#) and update the definitions of [dynamic instance](#) and [uniform control flow](#).

- #517: Expand the [About This Document](#) section to also discuss versioning.
- #564: Depth hint for the **DepthLess** [execution mode](#) means less-than-or-equal to.
- #558: Explicitly say (rather than imply) that **ImageMipmap** and **ImageReadWrite** [capabilities](#) apply to kernels.
- #563: Delete unnecessary statement about incomplete images in [OpImageQueryLod](#).
- #570: Update the definitions of the **Acquire** and **Release** [memory semantics](#).
- #560: It is not valid to make duplicate [BuiltIn](#) variables.
- #566: The Client API specifies what happens with image coordinates outside the image for [OpImageRead](#), [OpImageWrite](#), and [OpImageSparseRead](#).
- #573: Clarify the type read/written is scalar or vector in [OpImageRead](#), [OpImageWrite](#), and [OpImageSparseRead](#).
- #595: Remove the parenthetical partial list of annotation instructions in the [logical layout section](#).
- #574: Constituents of [OpConstantComposite](#) must not be specialization [constants](#).
- #444: Use more restrictive "only" language for what [decorations](#) may apply to.
- MR !182: See the client API for how **SubpassData** coordinates are applied in [OpImageRead](#).

4.40. Changes from Version 1.5, Revision 4

- Update to January 7, 2021 public headers.

4.41. Changes from Version 1.5, Revision 5

- Ported the specification itself to use asciidoctor instead of asciidoc.
- Reserve enumerants for the extensions:
 - SPV_INTEL_float_controls2
 - SPV_INTEL_vector_compute
 - SPV_INTEL_arbitrary_precision_floating_point
 - SPV_INTEL_usm_storage_classes
 - SPV_INTEL_unstructured_loop_controls
 - SPV_KHR_subgroup_uniform_control_flow
 - SPV_KHR_linkonce_odr
 - SPV_KHR_expect_assume
 - SPV_EXT_shader_atomic_float_min_max
 - SPV_KHR_integer_dot_product
 - SPV_KHR_bit_instructions
 - SPV_NV_ray_tracing_motion_blur
 - SPV_INTEL_optnone
 - SPV_NV_bindless_texture
- Add **CPP_for_OpenCL** [source language](#).
- Clarify that [OpFDiv](#) has a defined result when the divisor is 0. (MR !195.)
- Fix [execution-mode](#) table to show all 3 operands for **LocalSizeHintId**.

- Fix GitHub SPIRV-Registry issues:
 - #79: Clarify the definitions of **StorageImageMultisample** and **ImageMSArray** capabilities.
- Fix Khronos issues:
 - #351: **OpUDiv** and **OpUMod** have undefined behavior if the divisor is 0.
 - #621: Clarify the definition of the *Sampled* operand for **OpTypeImage**.
 - #611: Clarifying **string literals** are case sensitive for comparisons.
 - #615: Clarify **Block** and **BufferBlock** decorations.
 - #654: Clarify that the **ZeroExtend** image operand is not valid with signed types.
 - #623: Clarify **OpAccessChain** doesn't create any extra restrictions.
 - #647: Clarify **NoWrite** and **NoReadWrite** function parameter attributes apply to the pointer, not to the underlying memory.
 - #585: Clarify that **OpCopyObject** cannot have result type **OpTypeVoid**.
 - #614: Clarify that **OpUndef**, **OpPhi**, and **OpReturnValue** cannot have result type **OpTypeVoid**.
 - #115: Clarify the **Shader validation rules** for when **OpSelectionMerge** and **OpLoopMerge** instructions are necessary.
 - #656: Clarify the *<id>*-based rules for operands apply only to operands that are *<id>*s, in the **OpSpecConstantOp** instruction.
 - #627: Clarify the places that the **RelaxedPrecision** decoration must apply to.
 - #549: Clarify the **VariablePointers** and **VariablePointersStorageBuffer** capabilities enable additional features for logical pointers, but keep other prohibitions. Also that the **VariablePointers** and **VariablePointersStorageBuffer** capabilities allow a pointer to be an operand to **OpReturnValue**.
 - #640: Add parenthetical note in **structured control flow** about reconverging before reaching a merge block.
 - #656: Clarify the *<id>*-based rules for **OpSpecConstantOp** operands apply only to operands that are *<id>*s.
 - #651: Add a **validation rule** that the workgroup size cannot have a dimension with the value zero statically.
 - #580: Clarify that **SubpassInput** is not valid as the *Dim* operand of **OpTypeSampledImage**, and that sampled images with a *Dim* of **Buffer** are not valid in **image sampling instructions**.
 - #619: Add a **validation rule** that **LocalSize**, **LocalSizeId**, **LocalSizeHint**, and **LocalSizeHintId** can't be used at the same time.
 - #663: Restrict **OpSwitch** from being used to directly break or continue in a **structured loop**.
 - #678: Allow the **AliasedPointer** and **RestrictPointer** decorations to apply to **memory object declarations**.
 - #682: Clarify that the **VariablePointersStorageBuffer** capability is sufficient to compare pointers that point into different storage buffers using **OpPtrEqual** and **OpPtrNotEqual**.
- Changes from public headers
 - PR #240: Remove the **Kernel** capability from **fast-math flags**.
 - PR #257: Remove the **Shader** implicit declaration from **SPV_EXT_shader_atomic_float_add capabilities**.

4.42. Changes from Version 1.5

- New Functionality:
 - Khronos SPIR-V issue #515: The **FPFastMathMode** decoration may now be used with **OpFNegate**, with the binary floating-point comparison instructions (including **OpOrdered** and **OpUnordered**), and with **OpExtInst** where expressly permitted by the extended instruction set.
 - #661: Added a **Nontemporal Image Operand**.
- Extensions Incorporated into SPIR-V 1.6:
 - SPV_KHR_non_semantic_info, see **OpExtInstImport**.
 - SPV_KHR_integer_dot_product
 - SPV_KHR_terminate_invocation
 - SPV_EXT_demote_to_helper_invocation, with changes: Only **OpDemoteToHelperInvocationEXT** was incorporated. Instead of using **OpIsHelperInvocationEXT**, modules should use **Volatile** loads of the **HelperInvocation** built-in variable.
- Deprecations and Removals, from Khronos SPIR-V issues:
 - Removed **OpLessOrGreater**. Use **OpFOrdNotEqual** instead.
 - #620: The **WorkgroupSize** built-in is deprecated starting with version 1.6.
 - #645: The *True Label* and *False Label* of an **OpBranchConditional** must not be the same, starting with version 1.6.
 - #584: Disallow *Dim Buffer* in **OpTypeSampledImage** and **OpSampledImage** starting with version 1.6.
 - Deprecated **OpKill**, in favor of **OpTerminateInvocation**, or **OpDemoteToHelperInvocation**.
- Reserve enumerants for the SPV_KHR_fragment_shader_barycentric extension.

4.43. Changes from Version 1.6, Revision 1

- Reserve enumerants for:
 - SPV_KHR_ray_cull_mask
 - SPV_KHR_uniform_group_instructions
 - SPV_AMD_shader_early_and_late_fragment_tests
 - SPV_INTEL_vector_compute
 - SPV_INTEL_memory_access_aliasing
 - SPV_INTEL_split_barrier
 - SYCL [source language](#)
- Fix Khronos issues:
 - #680, #685, #696: Refine, clarify, and fix [structured control-flow](#) definitions and rules:
 - Add the concept of a [structured control-flow path](#) to better express the rules for structured control flow, as defined by the following terms.
 - Terms: Define the terms [branch edge](#), [merge edge](#), [continue edge](#), [structured control-flow edge](#), [path](#), [structured control-flow path](#), [structurally reachable](#), [structurally dominate](#), and [structurally post dominate](#). Remove "post dominate". Revise definition of [back edge](#) to refer to *branch edge* instead of *branch*. Pull out [back-edge block](#) into its own definition. Rename the term "termination instruction" to [block termination instruction](#) and introduce the term [function](#)

termination instruction.

- Rework and simplify structured control-flow rules using the terms above. Clarify that a loop's continue target must be different from its merge block. Remove redundant condition that a loop's continue construct must contain the loop's back-edge block. Precisely define the rules for exiting structured control-flow constructs.
- #672, #673, #674: Clarify branching rules for the **OpSwitch** instruction, for:
 - the order in which target operands appear in an **OpSwitch** instruction,
 - duplicated targets, and
 - branching between case constructs, to make it clear that branch edges do not have to start at a switch target, but can come from anywhere in a switch construct.
- #695: For most cases, disallow multiple uses of the same **decoration** on the same *<id>* or structure member.
- #696: Change **validation rules** for physical storage buffers to clarify they apply to pointers nested in other types (not just arrays).
- #672, #704: Clarify branching rules under **switch construct rules** for the **OpSwitch** instruction, making it clear that the rules about target ordering only apply to targets that define case constructs, and resolving ambiguity about what is allowed when the default case construct appears in the list of targets.
- Clarify the meaning of **fast math flags** when the asserted properties are not true.

4.44. Changes from Version 1.6, Revision 2

- Reserve enumerants for:
 - SPV_KHR_ray_tracing_position_fetch
 - SPV_QCOM_image_processing
 - SPV_ARM_core_builtins
 - SPV_NV_shader_invocation_reorder
 - SPV_NV_displacement_micromap
 - SPV_AMD_shader_enqueue
 - SPV_INTEL_fp_max_error
 - SPV_INTEL_kernel_attributes
 - SPV_INTEL_cache_controls
 - SPV_INTEL_global_variable_fpga_decorations
 - SPV_INTEL_global_variable_host_access
 - SPV_INTEL_bfloat16_conversion
 - SPV_INTEL_runtime_aligned
 - SPV_INTEL_fpga_argument_interfaces
 - SPV_INTEL_fpga_dsp_control
 - SPV_INTEL_fpga_invocation_pipelining_attributes
 - SPV_INTEL_fpga_latency_control
 - SPV_INTEL_fpga_loop_controls
 - SPV_INTEL_fpga_memory_attributes

- SPV_EXT_image_raw10_raw12
- SPV_EXT_shader_tile_image
- SPV_EXT_mesh_shader
- SPV_EXT_opacity_micromap
- Other changes from public headers
 - Added [source languages](#) HERO_C, NZSL, WGSL, and Slang
 - Removed the **Kernel** enabling capability from the [sampler addressing modes](#).
- Fix SPIR-V Registry issues:
 - #72: Be consistent in **OpTypeBool** that SPIR-V can support Booleans in the **UniformConstant** storage class.
 - #197: Clarify that **OpQuantizeToF16** must flush denormalized values to 0.
- Fix Khronos SPIR-V issues:
 - #689: Clarify use of **OpPhi** on **OpTypeImage** in the [universal validation rules](#).
 - #708: Remove unused definitions of Break Block, Continue Block and Return Block.
 - #707: Clarify that using a bad *Direction* in **OpGroupNonUniformQuadSwap** is invalid SPIR-V.
 - #712: Clarify multiple **UserSemantic** [decorations](#) can apply to a variable or structure member.
 - #731: Clarify that [aliasing](#) is based on dynamic execution.
 - #736: Clarify that **OpArrayLength** may have a logical pointer operand in the [universal validation rules](#).
 - #737: Clarify [validation rule](#) restricting **OpConstantNull** from pointing into the **PhysicalStorageBuffer** [storage class](#).
 - #738: Restrict **OpImageQueryLevels** and **OpImageQueryLod** images to have MS of 0.
 - #295: Clarify that the **ZeroExtend** and **SignExtend** [image operands](#) are not valid together.
 - #753: Clarify that **GroupNonUniformQuad** instructions are not affected by their execution scopes, and require the value to be [subgroup](#).
 - #754: Modify *ClusterSize* operands to refer to the size of the group of invocations participating in the instruction instead of always talking about **SubgroupSize**.
 - #755: Clarify set of invocations affected by a group operation:
 - Add definition of group (invocations).
 - Add definition of [workgroup](#).
 - Link to new definitions throughout the specification.
 - Define sizes of [quad](#), [subgroup](#), and [workgroup](#).
 - Modify description of *Execution Scope* to clarify that it identifies the group an instruction affects.
 - Remove restrictions on *Execution Scope* for most instructions, leaving it up to client APIs to restrict them.
 - Clarify that [non-uniform instructions](#) require the value of *Execution Scope* to be [subgroup](#).
 - Clarify that **GroupNonUniformQuad** instructions are not affected by their execution scopes.
 - #757: Restrict the type of ballot bit sets to be 4-component vectors of 32-bit unsigned integers in [Non-Uniform Instructions](#).
 - #758: Add the definition of a [cluster](#).

- #772: Clarify that **OpPtrAccessChain** does not dereference any pointer.
- #750: Update **validation rules** to reflect support for image and sampler array non-uniform indexing.
- Khronos SPIR-V MRs:
 - #261: Clarify that *Sampled* operand for **OpImageSparseFetch** is restricted to 1, bringing it in line with the constraint for **OpImageFetch**.
 - #280: **Control barriers** wait only for active invocations.
- Deprecations:
 - Issue #756: Deprecated the use of **BuiltIn** to decorate a constant to set its value and removed the deprecation of the **WorkgroupSize** **built-in**. That is, **WorkgroupSize** is kept but no longer marked as deprecated (it is still required by OpenCL). The use of **BuiltIn** to decorate a constant to set its value was only for **WorkgroupSize**, which has been superseded by the **LocalSizeId** **execution mode**.
 - MR #277: Deprecated **Simple memory model** in favor of **GLSL450**.

4.45. Changes from Version 1.6, Revision 3

- Reserve enumerants for:
 - SPV_KHR_float_controls2
 - SPV_KHR_maximal_reconvergence
 - SPV_KHR_quad_control
 - SPV_KHR_relaxed_extended_instruction
 - SPV_EXT_replicated_composites
 - SPV_INTEL_fpga_cluster_attributes
 - SPV_INTEL_masked_gather_scatter
 - SPV_INTEL_maximum_registers
 - SPV_QCOM_image_processing2
 - SPV_NV_shader_atomic_fp16_vector
 - SPV_NV_raw_access_chains
- Other changes from public headers
 - Enforce Core, KHR, EXT, Vendor ordering conventions for aliased names
 - Added **source languages** Zig
 - Removed the **Kernel** enabling capability from **Image Channel Order** and **Image Channel Data Type**.
- Fix Khronos SPIR-V Issues:
 - #638: Clarify that most **execution modes** must be applied at most once to a given **entry point**.
 - #766: Clarify the texel value type for the **ZeroExtend** and **SignExtend** **image operands**.
 - #724: Clarify that the **storage class** must match when performing an **OpBitcast** between two **OpTypePointer**. Clarify that the behavior is undefined when using the result of a bit cast between a scalar and a pointer (**OpBitcast** and **OpConvertUToPtr**) if the **storage class** scalar.
 - Add optional operand for **OpTypeFloat** to specify bit pattern of values. Clarify that **OpFConvert** operates on different types not just width. Clarify the following uses IEEE 754 floating-points: **OpQuantizeToF16**, **Image Operands** taking **floating-point type** operands, **VecTypeHint**, **DenormPreserve**, **DenormFlushToZero**, **SignedZeroInfNanPreserve**, **RoundingModeRTE** and

RoundingModeRTZ [execution mode](#), [Derivative instructions](#), **Float16Buffer**, **Float16** and **Int64 capabilities**. Clarify that **OpIsNan**, **OpIsInf**, **OpIsFinite**, **OpOrdered** and **OpUnordered** results depends on the floating-point encoding.

- #767: Rework the **Function Storage Class** definition. Clarify the memory is visible across all functions and not just the declaring function. Clarify that an **OpVariable** with a **Function Storage Class** is only allocated from its declaration until reaching a [function termination instruction](#).

4.46. Changes from Version 1.6, Revision 4

- Reserve enumerants for:
 - SPV_ARM_cooperative_matrix_layouts
 - SPV_EXT_arithmetic_fence
 - SPV_EXT_optnone
 - SPV_KHR_untyped_pointers
 - SPV_INTEL_subgroup_buffer_prefetch
 - SPV_INTEL_2d_block_io
 - SPV_INTEL_subgroup_matrix_multiply_accumulate
 - SPV_NV_cooperative_matrix2
 - SPV_NV_tensor_addressing
 - Rust [source language](#)
- Updated SPV_AMD_shader_enqueue enumerants
- Fix Khronos SPIR-V Issues:
 - #798: Clarify that **ArrayStride** applies objects in **PhysicalStorageBuffer** when computing the new address with **OpPtrAccessChain**. State the explicit layout requirement in each relevant [storage classes](#) entry.
 - #808: Add definition for *hint* and clarify that the following bits are hints:
 - [Selection Control](#): **Flatten** and **DontFlatten**
 - [Loop Control](#): **Unroll**, **DontUnroll**, **PeelCount** and **PartialCount**
 - [Function Control](#): **Inline** and **DontInline**
 - #813: Allow mismatching *Depth* for **OpSampledImage**
 - #809: Clarify structure with members decorated with **UserSemantic** can be used with any storage class.
 - #811: Refactor validation rules for **MakeTexelVisible**, **MakeTexelAvailable**, **MakePointerVisible**, **MakePointerAvailable**:
 - Remove mentions in universal validation rules
 - Make **MakeTexelVisible**, **MakeTexelAvailable**, **MakePointerVisible**, **MakePointerAvailable** description more generic to also capture instruction described in extensions
 - #797: No longer print duplicated tokens in enum and mask values. Instead aliases are printed between parentheses.
 - !321: Remove point of execution reachability paragraph from group operation as it is already implied by dynamic instance. Non semantic change.
 - !323: Turn the validation rules for explicit layout into a new [term](#) definition.

- #831: Fix use of *element* instead of *column* in **OpAccessChain**
- #837: Fix use of *Memory* scope operand in **atomic instruction** descriptions.
- #815: Clarify that an *image* is a handle and does not represent directly the memory holding the texels.
- #827: Lift the requirements to add **AliasedPointer** and **RestrictPointer** **decorations** on memory object declarations with holding **PhysicalStorageBuffer** pointers.
- #833: Remove entries for **OpImageSparseSampleProjImplicitLod**, **OpImageSparseSampleProjExplicitLod**, **OpImageSparseSampleProjDrefImplicitLod**, **OpImageSparseSampleProjDrefExplicitLod**. The instructions had no definition since 1.0.3, enums are still reserved and kept in the grammar.
- #691 / #832:
 - Introduce **scope**, **tangle**, **tangle invocations** and **scope restricted tangle** terms.
 - Remove use of *group* for invocations.
 - Fix missing **OpGroupReserveReadPipePackets**, **OpGroupReserveWritePipePackets**, **OpGroupCommitReadPipe** and **OpGroupCommitWritePipe** instructions from **tangled instructions** list.
 - Reworded **tangled instructions** to better define which invocations are involved in the operation by replacing use of active and inactive invocations as well as group.
 - Specify that for tangled instructions all invocations in the **scope restricted tangle** must reach the instruction before executing it.
 - Remove *as if all invocations execute simultaneously* wording in favor of a wording based on **program order**. State the program ordering requirement on all affected instruction.
 - Clarify that no dynamic instances program order after an **OpControlBarrier** can be executed until all invocations in the **scope restricted tangle** executed the dynamic instance.

4.47. Changes from Version 1.6, Revision 5

- Reserve enumerants for:
 - SPV_KHR_bfloat16
 - SPV_EXT_float8
 - SPV_ARM_graph
 - SPV_ARM_tensors
 - SPV_INTEL_bindless_images
 - SPV_INTEL_function_variants
 - SPV_INTEL_int4
 - SPV_INTEL_task_sequence
 - SPV_INTEL_ternary_bitwise_function
 - SPV_INTEL_tensor_float32_conversion
 - SPV_NV_linear_swept_spheres
 - SPV_NV_cluster_acceleration_structure
 - SPV_NV_cooperative_vector
 - SPV_QCOM_cooperative_matrix_conversion

- SPV_QCOM_tile_shading
- [Image Channel Data Type](#) for cl_ext_image_unsigned_10x6_12x4_14x2
- Fix Khronos SPIR-V Issues:
 - #843: Clarify that **OpShiftRightArithmetic** fills the bits according to the most-significant bit of *Base*.
 - !336: Clarify that Aligned **Memory Operands** must be a power of two.
 - Khronos Vulkan Issue #4193: Clarify runtime array must be last struct member by memory layout.
 - !334: Clarify **OpArrayLength** does not access the array contents.
 - #859: Clarify that the *Level of Detail* operand of **OpImageQuerySizeLod** is a 32-bit [integer type](#) scalar.
 - #860: Clarify that **Bias**, **Lod**, **Grad**, **ConstOffset**, **Offset**, **Sample**, **MinLod** and **Bias Image Operands** uses 32-bit [integer](#) or [floating point](#) types.
 - #865: Clarify **OpUndef** causes undefined behavior if used as an operand to **OpBranchConditional** and **OpSwitch**.
 - !350: Clarify that variable pointers on arrays of blocks are disallowed.
- Fix Github SPIRV-Headers Issues:
 - #487: State that **Unknown Image Format** can also be used if **Kernel** capability is declared.
 - #510: Header fix to stop enabling **PerTaskNV** with SPV_EXT_mesh_shader
- Fix Github SPIRV-Registry Issues:
 - #336: Fix [remainder](#) definition, 'q' is an integer.
 - #313: Clarify the definition of [dynamic instance](#) and which threads form the same tangle.