
Vulkanised
OSAKA 2023
MAY 11th
2023
KHRONOS
GROUP
Khronos Developer Day

Introduction to using SPIR-V

Spencer Fricke
LunarG, Inc.

Presented at the May Khronos DevDay in Osaka Japan

LUNARG

日本語のスライド

https://www.lunarg.com/wp-content/uploads/2023/05/J-SPIRV-Osaka.pdf

Introduction to using SPIR-V

Spencer Fricke
LunarG, Inc.

Presented at the May Khronos DevDay in Osaka Japan

LUNARG

Who is Spencer

- Have been working with SPIR-V for 3 years
  - Currently runtime SPIR-V validation for LunarG
- Only have worked with the Vulkan side of SPIR-V
  - Not an OpenCL expert
- Not a compiler engineer by trade
  - Have learned through SPIR-V
- Been part of SPIR-V Working Groups calls before
  - Was not part of its creation

Who is this talk for

- Learning how to read (and understand) SPIR-V
- If you need to make a shader tool
  - Even something simple (ex. How many OpLoad calls are there?)
- If you were curious how the “SPIR-V magic” works

What is SPIR-V

- “SPIR-V is a binary intermediate representation interchange format used to interface with a heterogeneous machine”

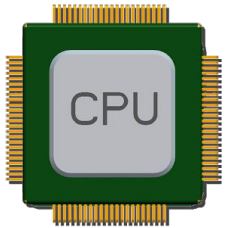
CPU Program

```
for (int i = 0; i < vertices_count; i++) {
    transform_matrix(i);
}
```

C/C++ code

```
subs    w8, w8, w9
b.ge    .LBB0_4
ldr     w0, [sp, #8]
bl      transform_matrix(int)
```

ARM/x86/MIPS assembly (ISA)



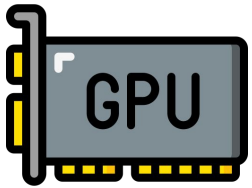
GPU Program

```
int i = get_thread_id();
transform_matrix(i);
```

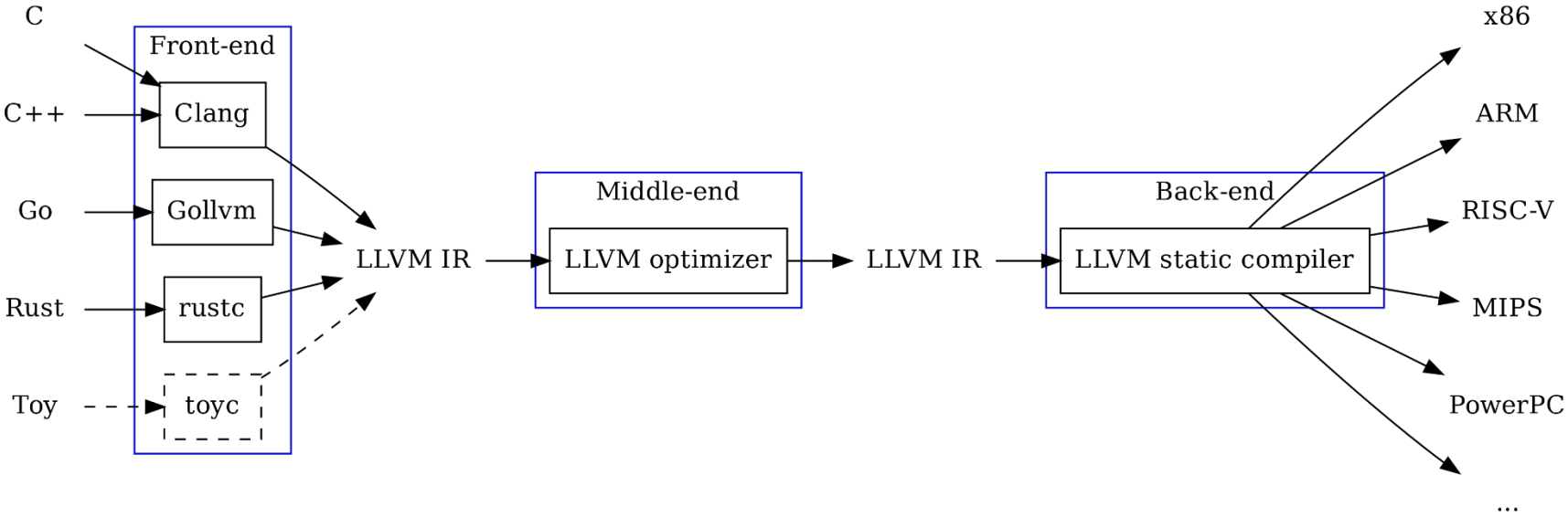
GLSL/HLSL code

```
s_buffer_load_dwordx8  s[52:59], s[0:3], 0x00
v_mul_f32      v1, s23, v1
v_mov_b32      v5, s37
v_fma_f32      v2, s24, v5, v2
```

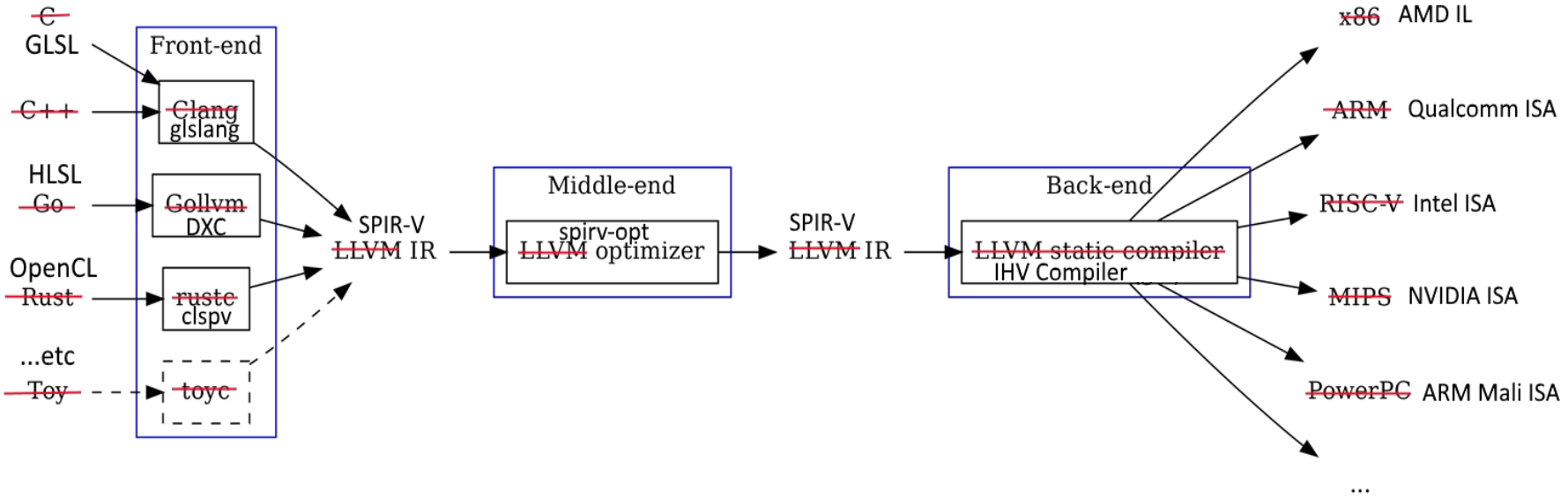
Assembly for each GPU vendor



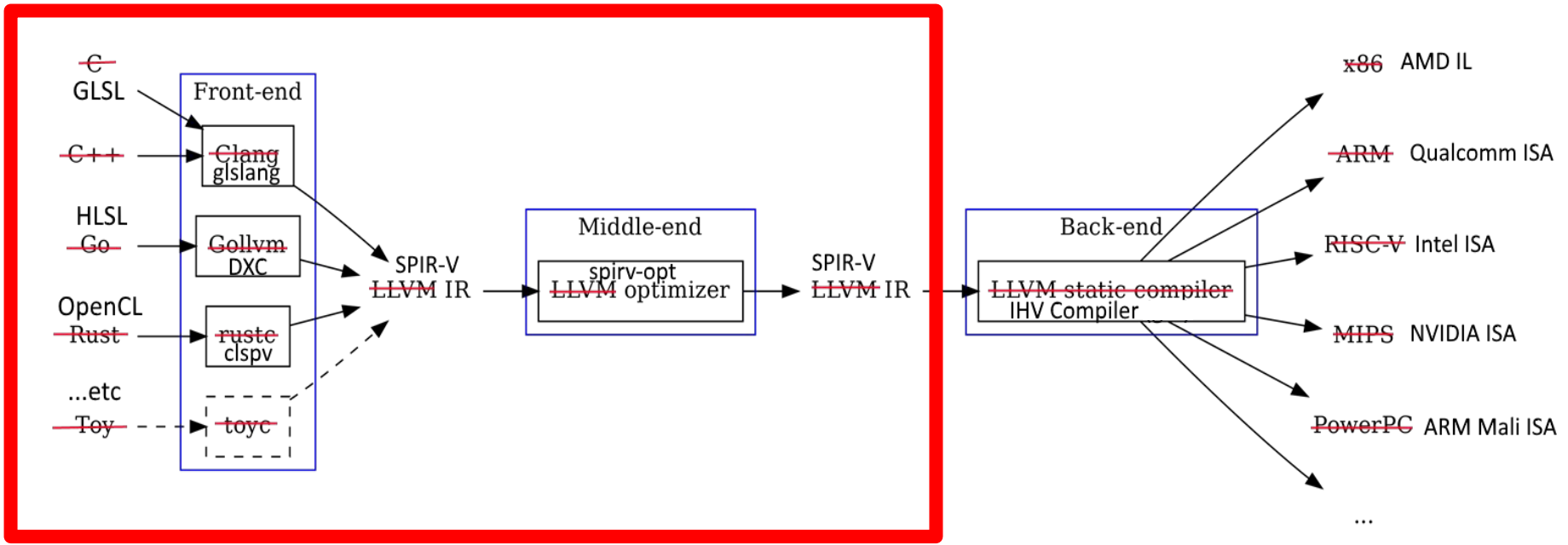
Using LLVM with CPU



Using SPIR-V with GPU



Using SPIR-V with GPU



Work eliminated OpenGL has to do

SPIR-V is inspired by LLVM IR

- LLVM IR is an internal detail of LLVM
- SPIR-V is a interchange format
  - everything in the Vulkan ecosystem uses to deal with shaders
- Structure of SPIR-V similar to LLVM
- bi-directional translation tool
  - https://github.com/KhronosGroup/SPIRV-LLVM-Translator

Shader vs Kernel

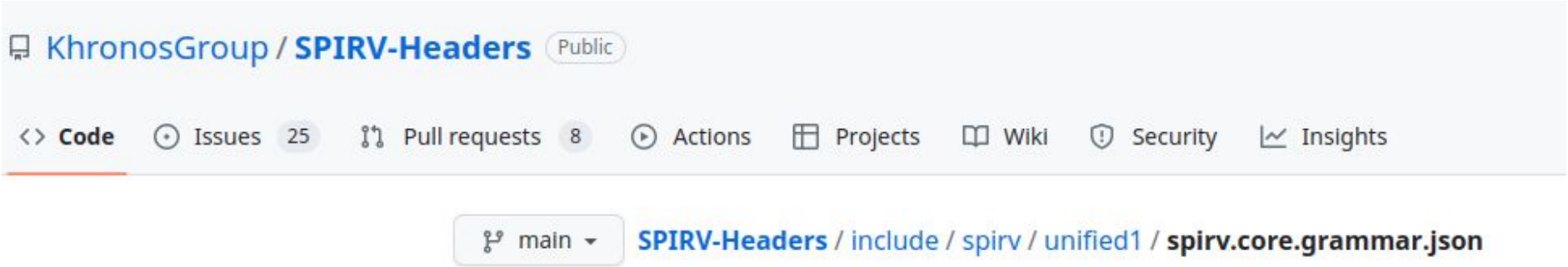
- OpenCL == Kernel
- Vulkan == Shader

Logical vs Physical

- OpenCL == Physical
- Vulkan == Logical (and some Physical via extensions)

SPIR-V Grammar JSON

- JSON file found in the SPIR-V Headers
- Equivalent to Vulkan's vk.xml
- Spec is generated from this file as well



Using SPIR-V Grammar - use case

```cpp
// Return number of optional parameter from ImageOperands
uint32_t ImageOperandsParamCount(uint32_t image_operand) {
    uint32_t count = 0;
    switch (image_operand) {
        case spv::ImageOperandsMaskNone:
        case spv::ImageOperandsNonPrivateTexelMask:
        case spv::ImageOperandsVolatileTexelMask:
        case spv::ImageOperandsSignExtendMask:
        case spv::ImageOperandsZeroExtendMask:
        case spv::ImageOperandsNontemporalMask:
            return 0;
        case spv::ImageOperandsBiasMask:
        case spv::ImageOperandsLodMask:
        case spv::ImageOperandsConstOffsetMask:
        case spv::ImageOperandsOffsetMask:
        case spv::ImageOperandsConstOffsetsMask:
        case spv::ImageOperandsSampleMask:
        case spv::ImageOperandsMinLodMask:
        case spv::ImageOperandsMakeTexelAvailableMask:
        case spv::ImageOperandsMakeTexelVisibleMask:
        case spv::ImageOperandsOffsetsMask:
            return 1;
        case spv::ImageOperandsGradMask:
            return 2;
        default:
            break;
    }
    return count;
```

```
"operand_kinds" : [
  {
    "category" : "BitEnum",
    "kind" : "ImageOperands",
    "enumerants" : [
      {
        "enumerant" : "None",
        "value" : "0x0000"
      },
      {
        "enumerant" : "Bias",
        "value" : "0x0001",
        "capabilities" : [ "Shader" ],
        "parameters" : [
          { "kind" : "IdRef" }
        ]
      },
```

Vulkan-ValidationLayers / layers / generated / **spirv_grammar_helper.cpp**

SPIR-V Extensions and Capabilities system

- Capabilities are how we communicate to the client API (ex Vulkan)
- Capabilities == Vulkan Features bit

```
#version 450
#extension GL_EXT_shader_8bit_storage : enable

layout (set = 0) buffer StorageBuffer {
    uint8_t dataA; // 0xAA
    uint8_t dataB; // 0xBB
} ssbo;

void main() {
    uint a = uint(ssbo.dataA);
    uint b = uint(ssbo.dataB);
}
```

```
OpCapability Shader
OpCapability UniformAndStorageBuffer8BitAccess
OpExtension "SPV_KHR_8bit_storage"
```

If the Vulkan feature is **not** supported, Validation Layers will detect it

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDevice8BitStorageFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer8BitAccess;
    VkBool32           uniformAndStorageBuffer8BitAccess;
    VkBool32           storagePushConstant8;
} VkPhysicalDevice8BitStorageFeatures;
```

SPIR-V Instructions

- SPIR-V is a stream of Instructions
- OpCode - name of instruction
  - Always starts with Op
- Operands
  - The words following the OpCode

```
%16 = OpTypeInt 32 1
%48 = OpVariable %47 Function
%54 = OpLoad %16 %48
%56 = OpSLessThan %25 %54 %55
```

Binary vs Disassembly

- SPIR-V is always a binary
- Presentation is always showing disassembly

```
%16 = OpTypeInt 32 1
%48 = OpVariable %47 Function
%54 = OpLoad %16 %48
%56 = OpSLessThan %25 %54 %55
```

SPIR-V Instruction

```
%6 = OpTypeInt 32 0
```

SPIR-V Instruction

%6 = OpTypeInt 32 0

0x00040015 0x00000006 0x00000020 0x00000000

SPIR-V Instruction

OpTypeInt

Declare a new *integer type*.

Width specifies how many bits wide the type is. *Width* is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement.

Signedness 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	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0

SPIR-V Instruction

OpTypeInt

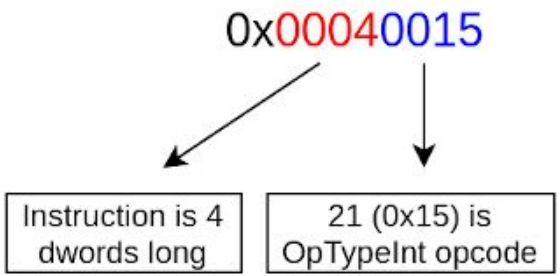
Declare a new *integer type*.

Width specifies how many bits wide the type is. *Width* is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement.

Signedness 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	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0



SPIR-V Instruction

OpTypeInt

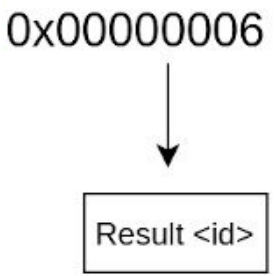
Declare a new *integer type*.

Width specifies how many bits wide the type is. *Width* is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement.

Signedness 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	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0



SPIR-V Instruction

OpTypeInt

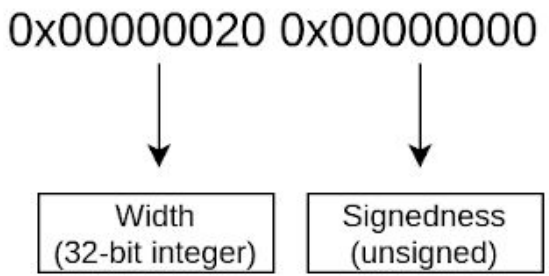
Declare a new *integer type*.

Width specifies how many bits wide the type is. *Width* is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement.

Signedness 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	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0



SPIR-V Instruction

OpTypeInt

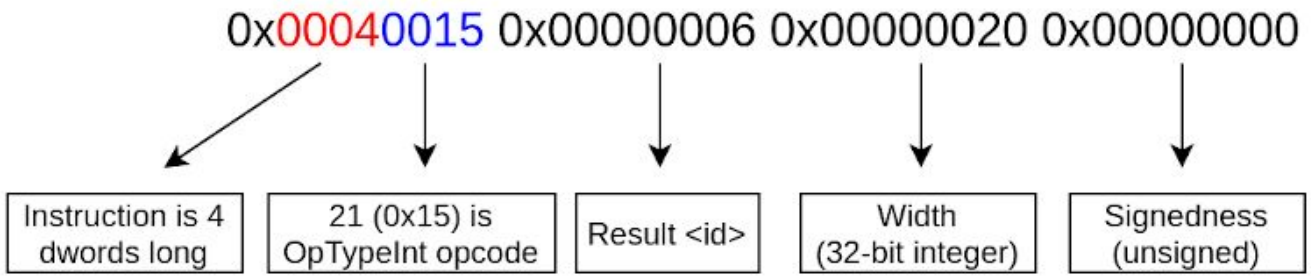
Declare a new *integer type*.

Width specifies how many bits wide the type is. *Width* is an unsigned 32-bit integer. The bit pattern of a signed integer value is two's complement.

Signedness 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	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0



SPIR-V Instruction

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 cannot 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**.

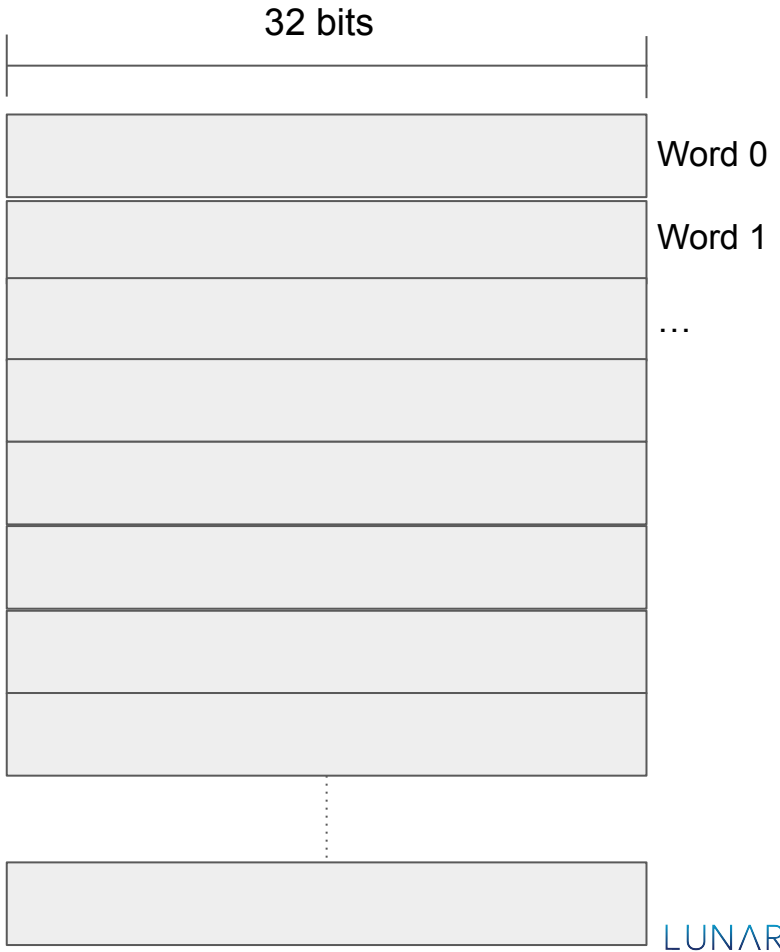
4 + variable	61	<id> *Result Type*	*Result <id>*	<id> *Pointer*	Optional *Memory Operands*

%54 = OpLoad %16 %48

Operand[1] = Result Type ID
Operand[2] = Result ID

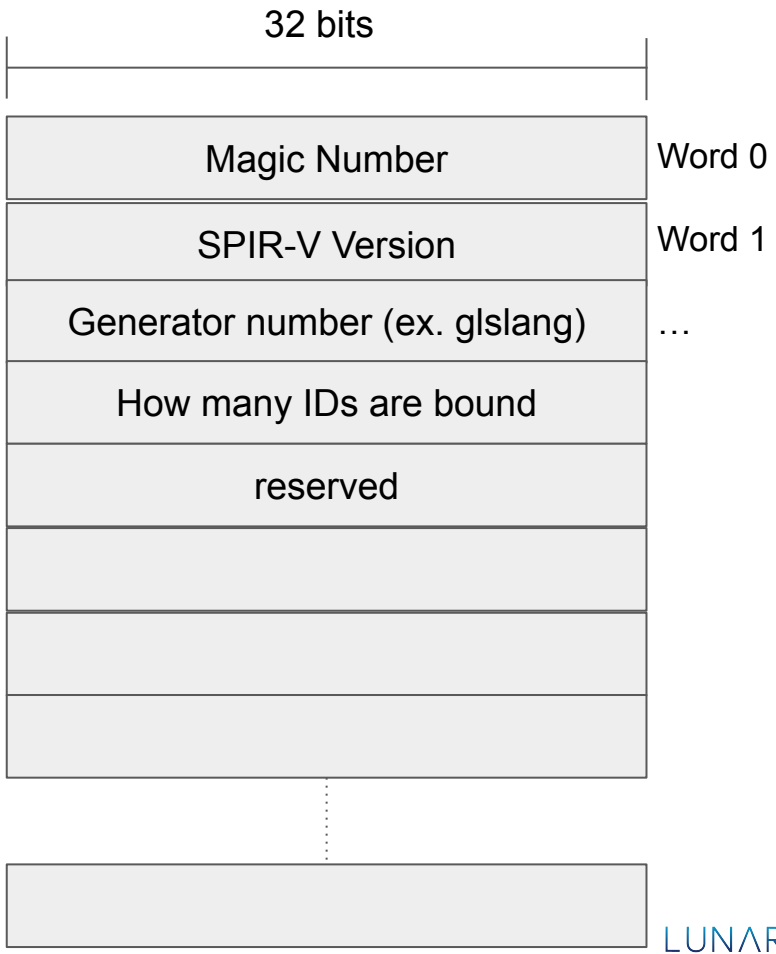
How is SPIR-V laid out

- Stream of 32-bit Words



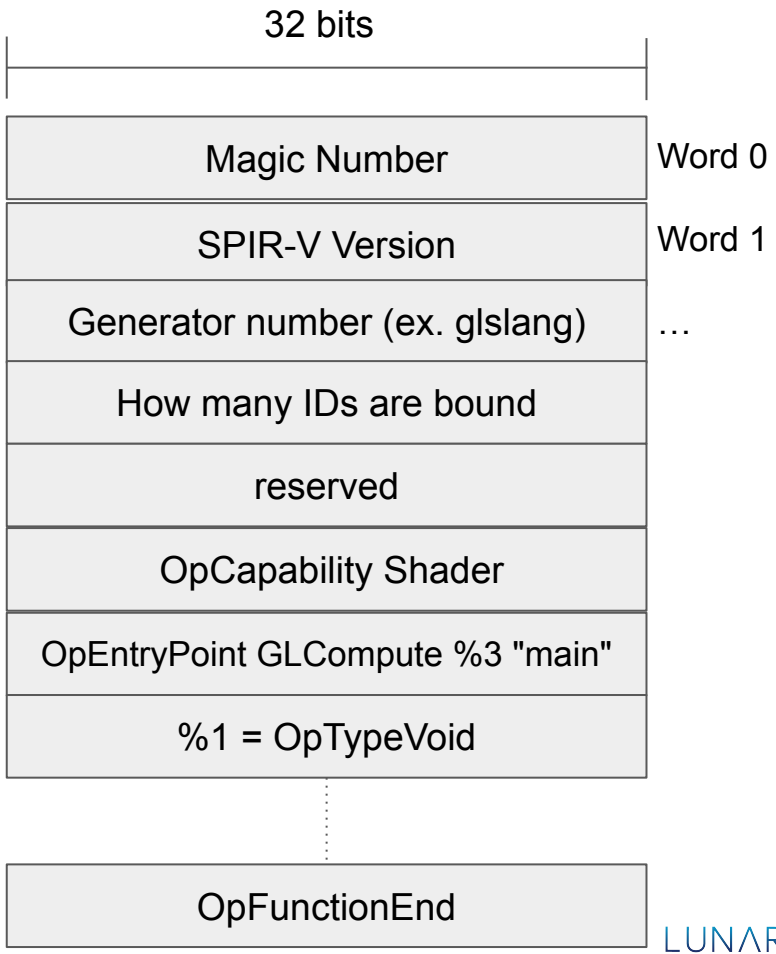
How is SPIR-V laid out

- First 5 words is the header



How is SPIR-V laid out

- Rest of words are the Instructions



How is SPIR-V laid out

- 5 sections
- Each has only certain instructions allowed
  - spirv-val will let you know if this is wrong

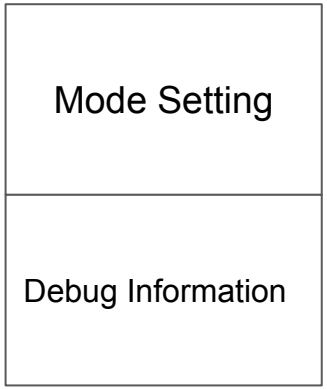
Mode Setting

Mode Setting

- High level details about the module
- Entrypoint
- Capabilities and Extensions
- Memory model
  - Logical vs Physical
- Execution mode
  - Ex. OriginUpperLeft vs OriginLowerLeft

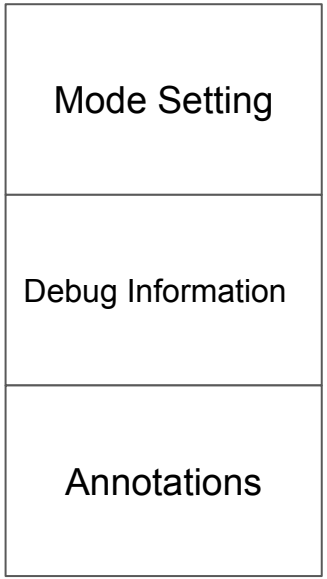
Debug Information

- Name of variables
- Source of SPIR-V
- Debugging tools make use of this section



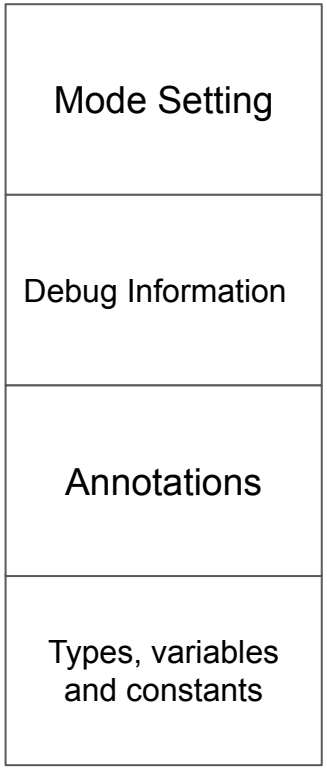
Annotations

- Apply Decorations
- Information for future instructions
  - RelaxedPrecision
  - ArrayStride
  - NonWritable
  - NonReadable



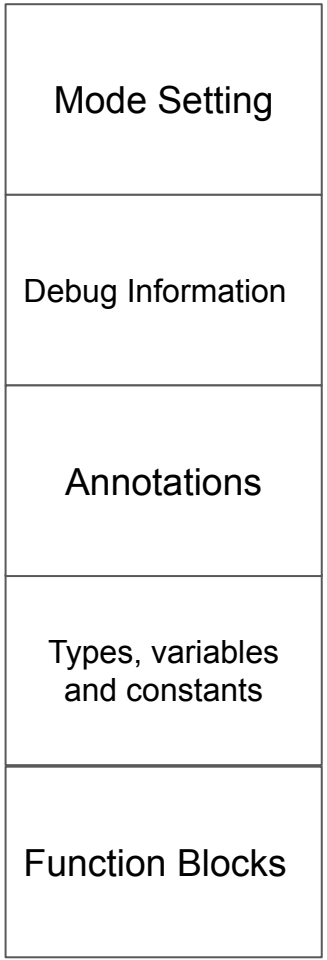
Types, variables and constants

- Declare types
  - OpTypeInt, OpTypeStruct, OpTypeSampler, etc
- Declare interface variables
  - Input/Output between shader stages
  - Descriptors (uniforms, storage image, etc)
- Constants
  - Also specialization constants



Function Blocks

- Blocks of Functions
- Where the logic occurs
  - Loads, stores, calculations, sampling

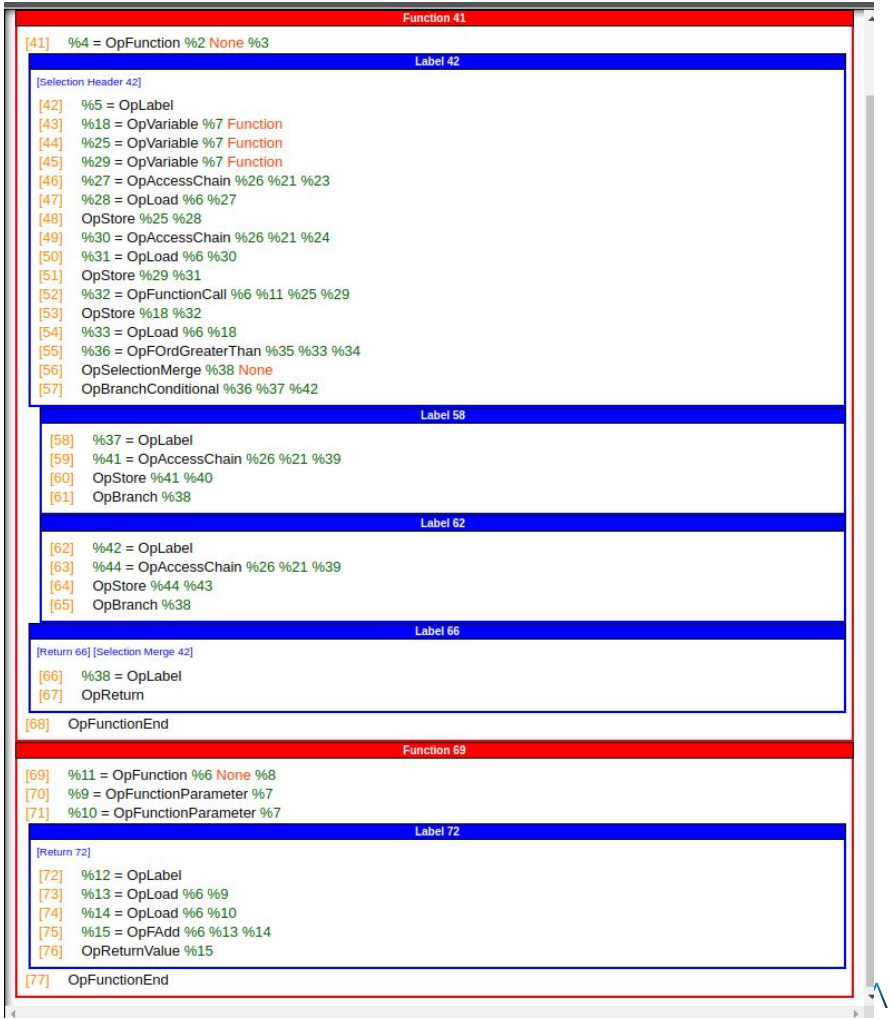


Understanding Blocks

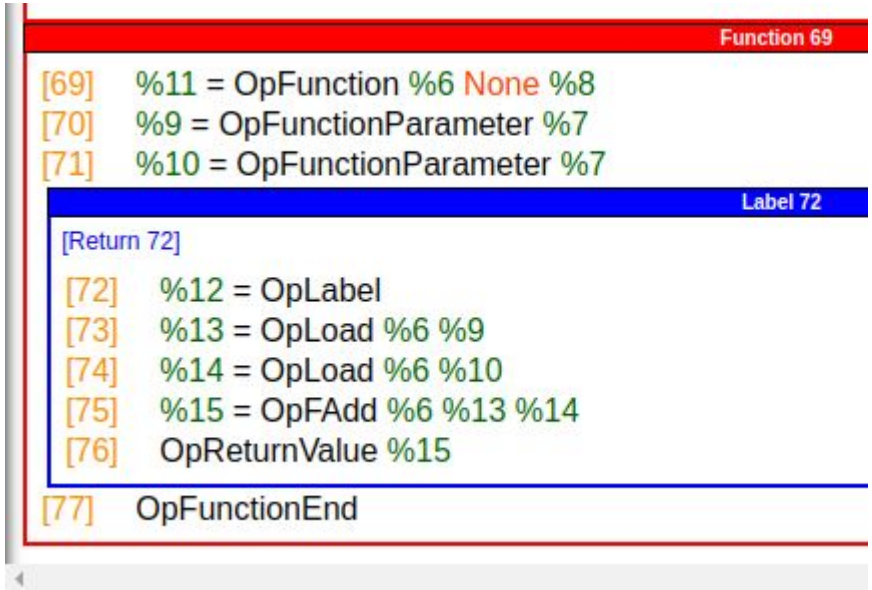
- Same idea as LLVM
- Multiple Function Blocks
- Each Function Block can have its own Blocks

```
float add(float a, float b) {
    return a + b;
}

void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

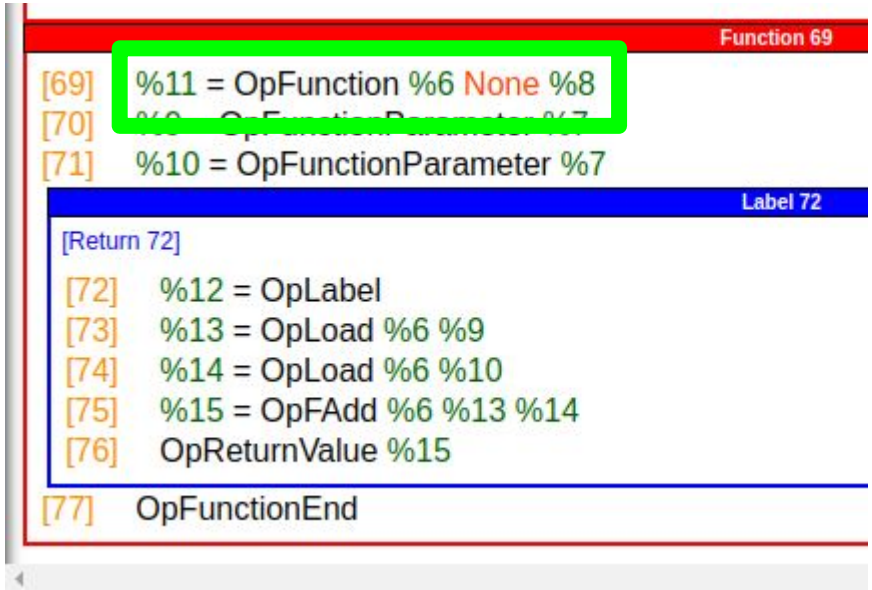


```
float add(float a, float b) {
    return a + b;
}
```

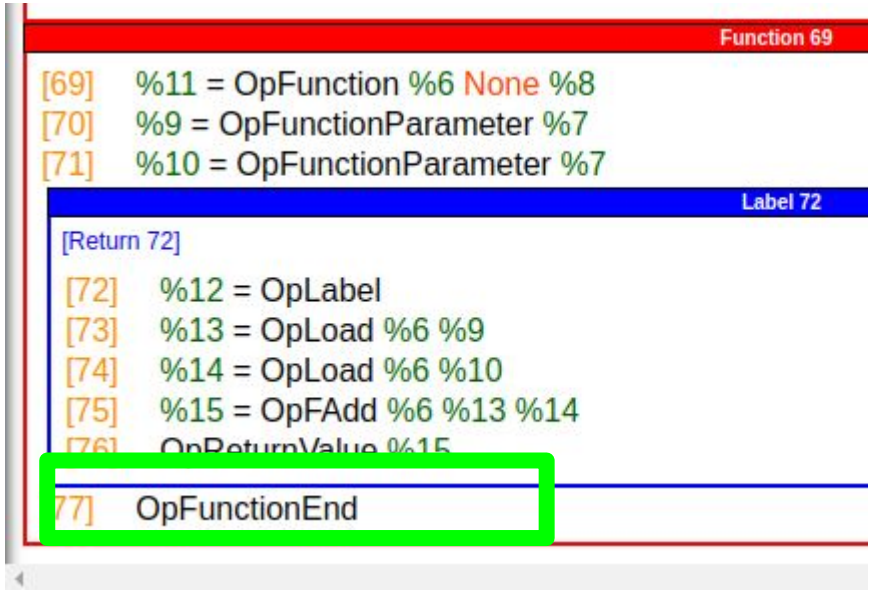


```
float add(float a, float b) {
    return a + b;
}
```

defines new Function Block



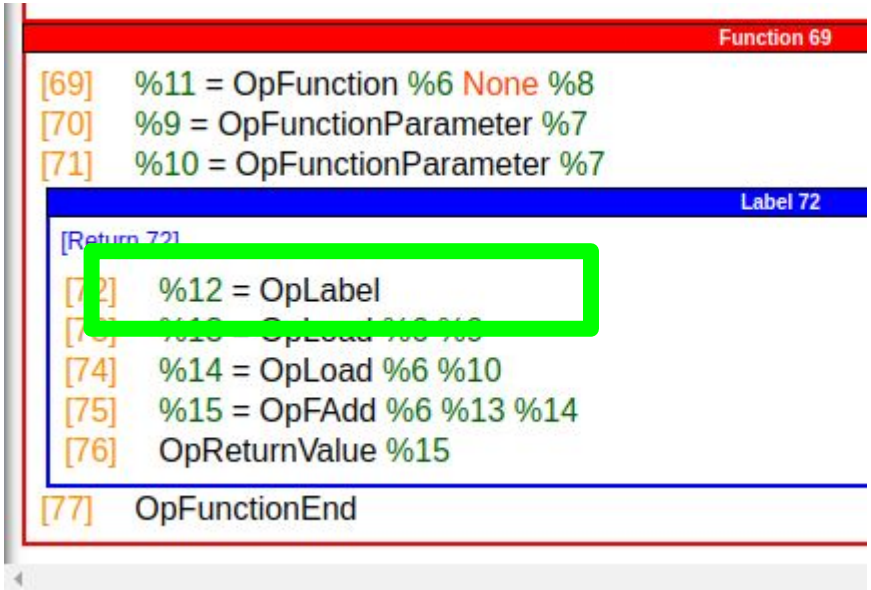
```
float add(float a, float b) {
    return a + b;
}
```



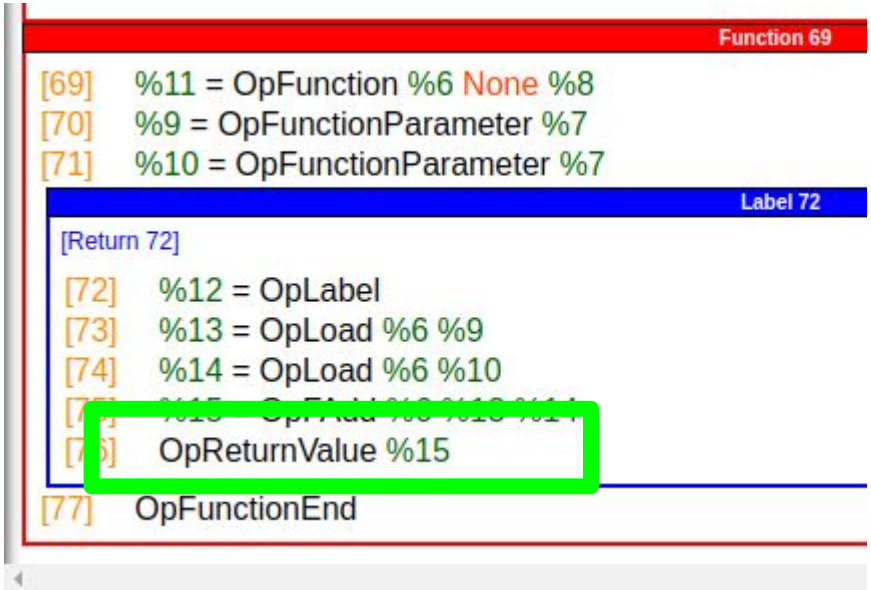
terminates Function Block

```
float add(float a, float b) {
    return a + b;
}
```

defines new Block



```
float add(float a, float b) {
    return a + b;
}
```



terminates Block

```
Function 41
[41]  %4 = OpFunction %2 None %3
  Label 42
  [Selection Header 42]
  [42]  %5 = OpLabel
  [43]  %18 = OpVariable %7 Function
  [44]  %25 = OpVariable %7 Function
  [45]  %29 = OpVariable %7 Function
  [46]  %27 = OpAccessChain %26 %21 %23
  [47]  %28 = OpLoad %6 %27
  [48]  OpStore %25 %28
  [49]  %30 = OpAccessChain %26 %21 %24
  [50]  %31 = OpLoad %6 %30
  [51]  OpStore %29 %31
  [52]  %32 = OpFunctionCall %6 %11 %25 %29
  [53]  OpStore %18 %32
  [54]  %33 = OpLoad %6 %18
  [55]  %36 = OpFOrdGreaterThan %35 %33 %34
  [56]  OpSelectionMerge %38 None
  [57]  OpBranchConditional %36 %37 %42
    Label 58
    [58]  %37 = OpLabel
    [59]  %41 = OpAccessChain %26 %21 %39
    [60]  OpStore %41 %40
    [61]  OpBranch %38
    Label 62
    [62]  %42 = OpLabel
    [63]  %44 = OpAccessChain %26 %21 %39
    [64]  OpStore %44 %43
    [65]  OpBranch %38
  Label 66
  [Return 66] [Selection Merge 42]
  [66]  %38 = OpLabel
  [67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Makes function call

```
Function 41
[41]  %4 = OpFunction %2 None %3
  Label 42
  [Selection Header 42]
  [42]  %5 = OpLabel
  [43]  %18 = OpVariable %7 Function
  [44]  %25 = OpVariable %7 Function
  [45]  %29 = OpVariable %7 Function
  [46]  %27 = OpAccessChain %26 %21 %23
  [47]  %28 = OpLoad %6 %27
  [48]  OpStore %25 %28
  [49]  %30 = OpAccessChain %26 %21 %24
  [50]  %31 = OpLoad %6 %30
  [51]  OpStore %29 %31
  [52]  %32 = OpFunctionCall %6 %11 %25 %29
  [53]  OpStore %18 %32
  [54]  %33 = OpLoad %6 %18
  [55]  %36 = OpFOrdGreaterThan %35 %33 %34
  [56]  OpSelectionMerge %38 None
  [57]  OpBranchConditional %36 %37 %42
    Label 58
    [58]  %37 = OpLabel
    [59]  %41 = OpAccessChain %26 %21 %39
    [60]  OpStore %41 %40
    [61]  OpBranch %38
    Label 62
    [62]  %42 = OpLabel
    [63]  %44 = OpAccessChain %26 %21 %39
    [64]  OpStore %44 %43
    [65]  OpBranch %38
  Label 66
  [Return 66] [Selection Merge 42]
  [66]  %38 = OpLabel
  [67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Declare Control Flow

```
Function 41
[41]  %4 = OpFunction %2 None %3
  Label 42
  [Selection Header 42]
  [42]  %5 = OpLabel
  [43]  %18 = OpVariable %7 Function
  [44]  %25 = OpVariable %7 Function
  [45]  %29 = OpVariable %7 Function
  [46]  %27 = OpAccessChain %26 %21 %23
  [47]  %28 = OpLoad %6 %27
  [48]  OpStore %25 %28
  [49]  %30 = OpAccessChain %26 %21 %24
  [50]  %31 = OpLoad %6 %30
  [51]  OpStore %29 %31
  [52]  %32 = OpFunctionCall %6 %11 %25 %29
  [53]  OpStore %18 %32
  [54]  %33 = OpLoad %6 %18
  [ ]   %36 = OpFOrdGreaterThan %33 %35 %34
  [ ]   OpSelectionMerge %38 None
  [ ]   OpBranchConditional %36 %37 %42
    Label 58
    [58]  %37 = OpLabel
    [59]  %41 = OpAccessChain %26 %21 %39
    [60]  OpStore %41 %40
    [61]  OpBranch %38
    Label 62
    [62]  %42 = OpLabel
    [63]  %44 = OpAccessChain %26 %21 %39
    [64]  OpStore %44 %43
    [65]  OpBranch %38
  Label 66
  [Return 66] [Selection Merge 42]
  [66]  %38 = OpLabel
  [67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Declare Control Flow

Which Block the Control Flow merges

```
Function 41
[41]  %4 = OpFunction %2 None %3
  Label 42
  [Selection Header 42]
  [42]  %5 = OpLabel
  [43]  %18 = OpVariable %7 Function
  [44]  %25 = OpVariable %7 Function
  [45]  %29 = OpVariable %7 Function
  [46]  %27 = OpAccessChain %26 %21 %23
  [47]  %28 = OpLoad %6 %27
  [48]  OpStore %25 %28
  [49]  %30 = OpAccessChain %26 %21 %24
  [50]  %31 = OpLoad %6 %30
  [51]  OpStore %29 %31
  [52]  %32 = OpFunctionCall %6 %11 %25 %29
  [53]  OpStore %18 %32
  [54]  %33 = OpLoad %6 %18
  [ ]  OpSelectionMerge %38 None
    Label 58
    [58]  %37 = OpLabel
    [59]  %41 = OpAccessChain %26 %21 %39
    [60]  OpStore %41 %40
    [61]  OpBranch %38
    Label 62
    [62]  %42 = OpLabel
    [63]  %44 = OpAccessChain %26 %21 %39
    [64]  OpStore %44 %43
    [65]  OpBranch %38
  Label 66
  [66]  %38 = OpLabel
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Conditional decides where to go next

```
Function 41
[41]  %4 = OpFunction %2 None %3
Label 42
[Selection Header 42]
[42]  %5 = OpLabel
[43]  %18 = OpVariable %7 Function
[44]  %25 = OpVariable %7 Function
[45]  %29 = OpVariable %7 Function
[46]  %27 = OpAccessChain %26 %21 %23
[47]  %28 = OpLoad %6 %27
[48]  OpStore %25 %28
[49]  %30 = OpAccessChain %26 %21 %24
[50]  %31 = OpLoad %6 %30
[51]  OpStore %29 %31
[52]  %32 = OpFunctionCall %6 %11 %25 %29
[53]  OpStore %18 %32
[54]  %33 = OpLoad %6 %18
[55]  %36 = OpFOrdGreaterThan %35 %33 %34
[57]  OpBranchConditional %36 %37 %42
[58]  %37 = OpLabel
[59]  %41 = OpAccessChain %26 %21 %39
[60]  OpStore %41 %40
[61]  OpBranch %38
Label 62
[62]  %42 = OpLabel
[63]  %44 = OpAccessChain %26 %21 %39
[64]  OpStore %44 %43
[65]  OpBranch %38
Label 66
[Return 66] [Selection Merge 42]
[66]  %38 = OpLabel
[67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Conditional decides where to go next

```
Function 41
[41]  %4 = OpFunction %2 None %3
Label 42
[Selection Header 42]
[42]  %5 = OpLabel
[43]  %18 = OpVariable %7 Function
[44]  %25 = OpVariable %7 Function
[45]  %29 = OpVariable %7 Function
[46]  %27 = OpAccessChain %26 %21 %23
[47]  %28 = OpLoad %6 %27
[48]  OpStore %25 %28
[49]  %30 = OpAccessChain %26 %21 %24
[50]  %31 = OpLoad %6 %30
[51]  OpStore %29 %31
[52]  %32 = OpFunctionCall %6 %11 %25 %29
[53]  OpStore %18 %32
[54]  %33 = OpLoad %6 %18
[55]  %36 = OpFOrdGreaterThan %35 %33 %34
[57]  OpBranchConditional %36 %37 %42
58
[58]  %37 = OpLabel
[60]  OpStore %41 %40
[61]  OpBranch %38
62
[62]  %42 = OpLabel
[64]  OpStore %44 %43
[65]  OpBranch %38
Label 66
[Return 66] [Selection Merge 42]
[66]  %38 = OpLabel
[67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Blocks never just “fall through”

```
Function 41
[41]  %4 = OpFunction %2 None %3
  Label 42
  [Selection Header 42]
  [42]  %5 = OpLabel
  [43]  %18 = OpVariable %7 Function
  [44]  %25 = OpVariable %7 Function
  [45]  %29 = OpVariable %7 Function
  [46]  %27 = OpAccessChain %26 %21 %23
  [47]  %28 = OpLoad %6 %27
  [48]  OpStore %25 %28
  [49]  %30 = OpAccessChain %26 %21 %24
  [50]  %31 = OpLoad %6 %30
  [51]  OpStore %29 %31
  [52]  %32 = OpFunctionCall %6 %11 %25 %29
  [53]  OpStore %18 %32
  [54]  %33 = OpLoad %6 %18
  [55]  %36 = OpFOrdGreaterThan %35 %33 %34
  [56]  OpSelectionMerge %38 None
  [57]  OpBranchConditional %36 %37 %42
    Label 58
    [58]  %37 = OpLabel
    [59]  %41 = OpAccessChain %26 %21 %39
    [61]  OpBranch %38
    62
    [62]  %42 = OpLabel
    [63]  %44 = OpAccessChain %26 %21 %39
    [65]  OpBranch %38
    66
    [Return 66] [Selection Merge 42]
    [66]  %38 = OpLabel
    [67]  OpReturn
[68]  OpFunctionEnd
```

```
void main() {
    float c = add(in_a, in_b);
    if (c > 0.0) {
        out_data = 1.0;
    } else {
        out_data = -1.0;
    }
}
```

Parsing a SPIR-V binary

```
void parseModule (uint32_t* pCode, uint32_t codeSize) {
    uint32_t offset = 5; // first 5 words in module are the headers
    while(offset < codeSize) {
        uint32_t instruction = pCode[offset];

        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0x0ffffu;

        offset += length;
    }
}
```

Parsing a SPIR-V binary

```
void parseModule(uint32_t* pCode, uint32_t codeSize) {
    uint32_t offset = 5; // first 5 words in module are the headers
    while(offset < codeSize) {
        uint32_t instruction = pCode[offset];

        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0x0ffffu;

        offset += length;
    }
}
```

Parsing a SPIR-V binary

```
void parseModule (uint32_t* pCode, uint32_t codeSize) {
    uint32_t offset = 5; // first 5 words in module are the headers
    while(offset < codeSize) {
        uint32_t instruction = pCode[offset];

        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0x0ffffu;

        offset += length;
    }
}
```

Parsing a SPIR-V binary

```
void parseModule (uint32_t* pCode, uint32_t codeSize) {
    uint32_t offset = 5; // first 5 words in module are the headers
    while(offset < codeSize) {
        uint32_t instruction = pCode[offset];

        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0x0ffffu;

        offset += length;
    }
}
```

Parsing a SPIR-V binary

```
void parseModule (uint32_t* pCode, uint32_t codeSize) {
    uint32_t offset = 5; // first 5 words in module are the headers
    while(offset < codeSize) {
        uint32_t instruction = pCode[offset];

        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0x0ffffu;

        offset += length;
    }
}
```

Extended instruction sets

- allows SPIR-V to be agnostic
- Languages describe different rules around the same instructions
  - Ex. Accuracy of functions like **sin()**

```
void main() {
    float x = sin(data);
}
```

```
void main() {
    float x = sin(data);
}
```

https://registry.khronos.org/SPIR-V/specs/unified1/GLSL.std.450.pdf

Sin	
The standard trigonometric sine of *x* radians. The operand *x* must be a scalar or vector whose component type is 16-bit or 32-bit floating-point. *Result Type* and the type of *x* must be the same type. Results are computed per component.	
13	*<id>* *x*

```
void main() {
    float x = sin(data);
}
```

```
%1 = OpExtInstImport "GLSL.std.450"

%17 = OpExtInst %6 %1 Sin %16
```

Sin

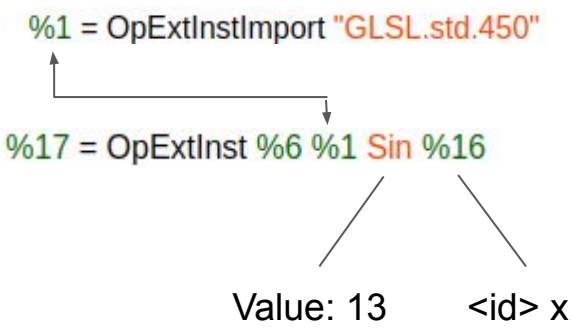
The standard trigonometric sine of *x* radians.

The operand *x* must be a scalar or vector whose component type is 16-bit or 32-bit floating-point.

Result Type and the type of *x* must be the same type. Results are computed per component.

13	*<id>* *x*

```
void main() {
    float x = sin(data);
}
```



Sin

The standard trigonometric sine of *x* radians.

The operand *x* must be a scalar or vector whose component type is 16-bit or 32-bit floating-point.

Result Type and the type of *x* must be the same type. Results are computed per component.

13	*<id>* *x*

Entry Point, Execution Model, and Execution Mode

- Module == SPIR-V File
  - May have multiple Entry Points
- Model vs Mode

Entry Point, Execution Model, and Execution Mode

- Can have more than 1 Entry Point in a module

```
float foo(float bar) {
    return bar / 2.0;
}

void vertex_main() {
    // set RoundingModeRTE
    foo(3.0);
}

void fragment_main() {
    // set RoundingModeRTZ
    foo(3.0);
}
```

Entry Point, Execution Model, and Execution Mode

- Can have more than 1 Entry Point in a module

```
float foo(float bar) {
    return bar / 2.0;
}

void vertex_main() {
    // set RoundingModeRTE
    foo(3.0);
}

void fragment_main() {
    // set RoundingModeRTZ
    foo(3.0);
}
```

```
        OpEntryPoint Vertex %v_main "vertex_main" %vert_out
        OpEntryPoint Fragment %f_main "fragment_main"
        OpExecutionMode %v_main RoundingModeRTE 32
        OpExecutionMode %f_main RoundingModeRTZ 32
        // ...
%foo = OpFunction %float None %1
%bar = OpFunctionParameter %ptr_float
  %2 = OpLabel
  %3 = OpLoad %float %bar
  %4 = OpFDiv %float %3 %float_2
        OpReturnValue %4
        OpFunctionEnd
```

Entry Point, Execution Model, and Execution Mode

- Execution Model is defined

```
float foo(float bar) {
    return bar / 2.0;
}

void vertex_main() {
    // set RoundingModeRTE
    foo(3.0);
}

void fragment_main() {
    // set RoundingModeRTZ
    foo(3.0);
}
```

```
        OpEntryPoint Vertex %v_main "vertex_main" %vert_out
        OpEntryPoint Fragment %f_main "fragment_main"
        OpExecution                    ngModeRTE 32
        OpExecutionMode %f_main RoundingModeRTZ 32
        // ...
%foo = OpFunction %float None %1
%bar = OpFunctionParameter %ptr_float
  %2 = OpLabel
  %3 = OpLoad %float %bar
  %4 = OpFDiv %float %3 %float_2
        OpReturnValue %4
        OpFunctionEnd
```

Entry Point, Execution Model, and Execution Mode

- Execution Mode applies to Function, not Entry Point

```
float foo(float bar) {
    return bar / 2.0;
}

void vertex_main() {
    // set RoundingModeRTE
    foo(3.0);
}

void fragment_main() {
    // set RoundingModeRTZ
    foo(3.0);
}
```

```
        OpEntryPoint Vertex %v_main "vertex_main" %vert_out
        OpExecutionMode %v_main RoundingModeRTE 32
        OpExecutionMode %f_main RoundingModeRTZ 32
%foo = OpFunction %float None %1
%bar = OpFunctionParameter %ptr_float
  %2 = OpLabel
  %3 = OpLoad %float %bar
  %4 = OpFDiv %float %3 %float_2
       OpReturnValue %4
       OpFunctionEnd
```

Types

- OpType*
- Can use types to make bigger types
- Define the types using once, shared across the module

Types - mat3x2

- Same as
  - vec2
  - vec2
  - vec2

Types - mat3x2

- %float = OpTypeFloat 32

Types - mat3x2

- %float = OpTypeFloat 32
- %v2float = OpTypeVector %float 2

Types - mat3x2

- %float = OpTypeFloat 32
- %v2float = OpTypeVector %float 2
- %mat3v2float = OpTypeMatrix %v2float 3

Types - mat3x2

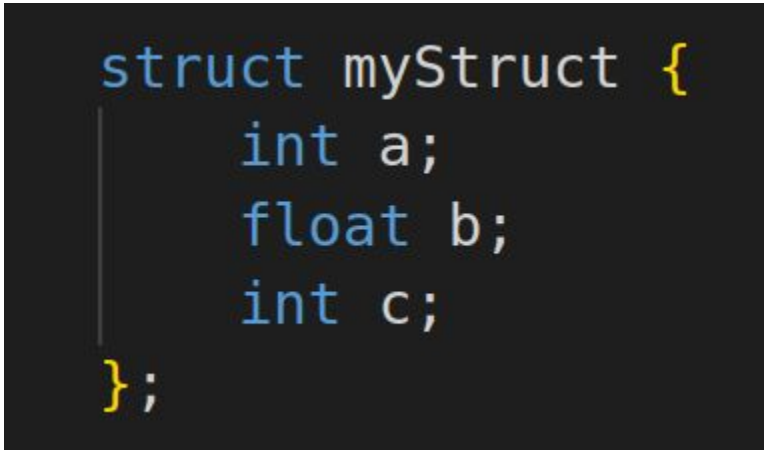
- %float = OpTypeFloat 32
- %v2float = OpTypeVector %float 2
- %mat3v2float = OpTypeMatrix %v2float 3
- %ptr = OpTypePointer Input %mat3v2float

Type - Structs

```
struct myStruct {
    int a;
    float b;
    int c;
};
```

Type - Structs

- %int = OpTypeInt 32 1
- %float = OpTypeFloat 32



Type - Structs

- %int = OpTypeInt 32 1
- %float = OpTypeFloat 32
- %myStruct = OpTypeStruct %int %float %int

```
struct myStruct {
    int a;
    float b;
    int c;
};
```

Access Chains

- Used to access part of a variable
- It is a “chain” of “accesses” through the variable

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

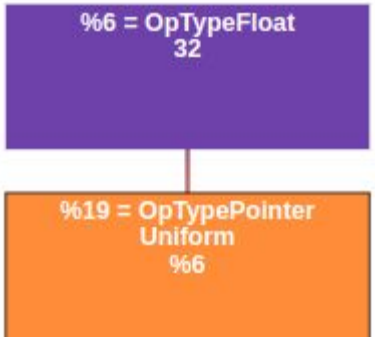
Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

```
%20 = OpAccessChain %19 %13 %15 %16 %18
```

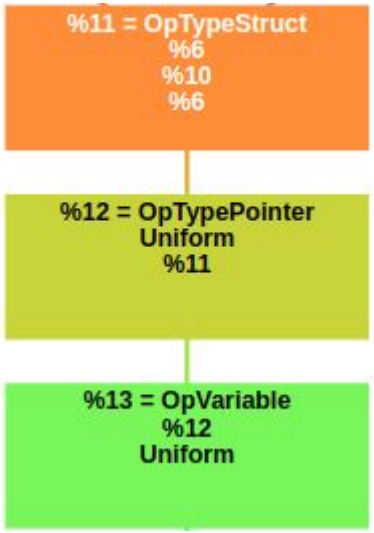
Access Chain



Result type

%20 = OpAccessChain %19 %13 %15 %16 %18

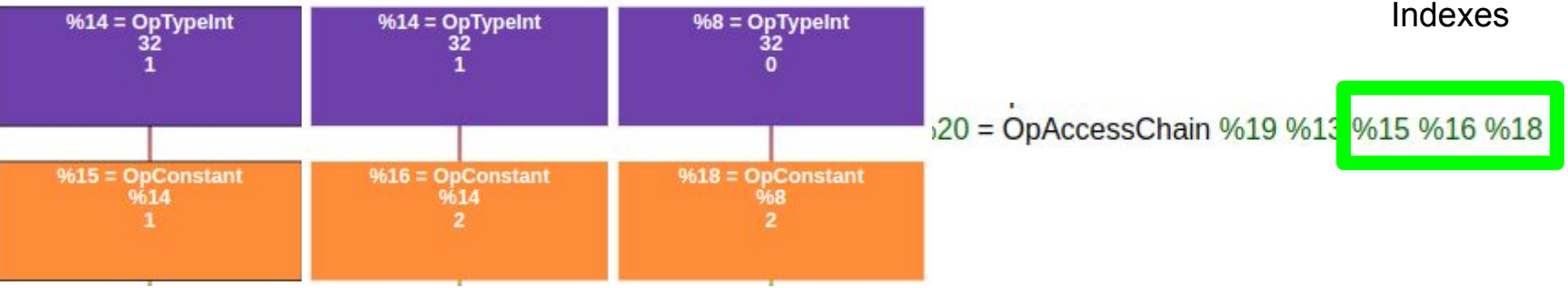
Access Chain



Base object

%20 = OpAccessChain %19 %13 %15 %16 %18

Access Chain



Access Chain

Struct

```
%20 = OpAccessChain %19 %13 1 2 2
```

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 [4];
    float c;
};

void main() {
    [2] z = 0.0f;
}
```

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

Access Chain

```
layout(set = 0) buffer ssbo {
    float a;
    vec3 b[4];
    float c;
};

void main() {
    b[2].z = 0.0f;
}
```

- %20 = OpAccessChain %19 %13 %15 %16 %18
- OpStore %20 %float_0


Questions?