Vulkanised 2026

The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

How to not be Scared of SPIR-V

Spencer Fricke, LunarG

Who is Spencer

- Technical Lead of Validation Layers
  - Wrote most of the runtime SPIR-V Validation
- Have added a lot code to spirv-val (SPIRV-Tools)
- Wrote/Maintain SPIR-V Visualizer
- Wrote/Maintain SPIR-V Guide
- Help (sorry if slow) maintain SPIRV-Reflect
- Got SPIR-V added to Compiler Explorer (godbolt)

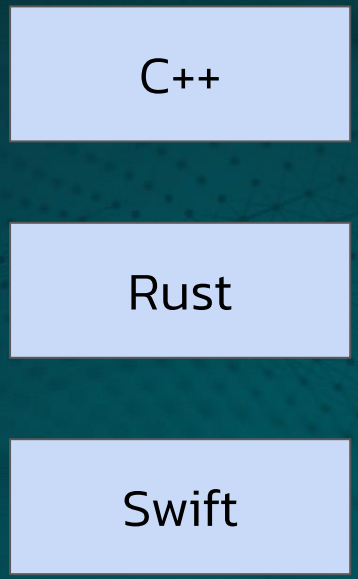
aka

I will probably fix your SPIR-V tooling if you nerd snipe me

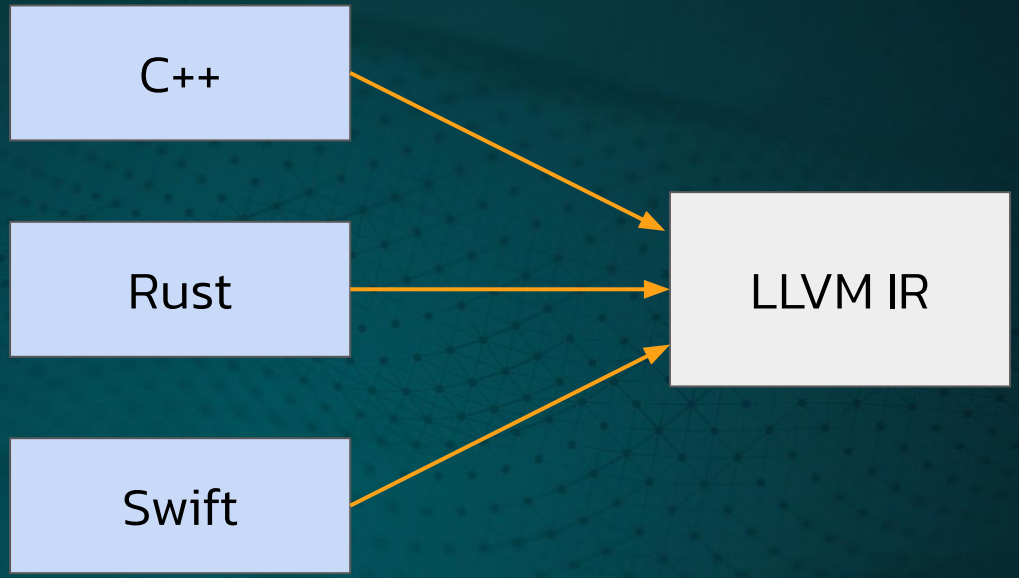
LUNARG

Raise your hand if you have heard of SPIR-V

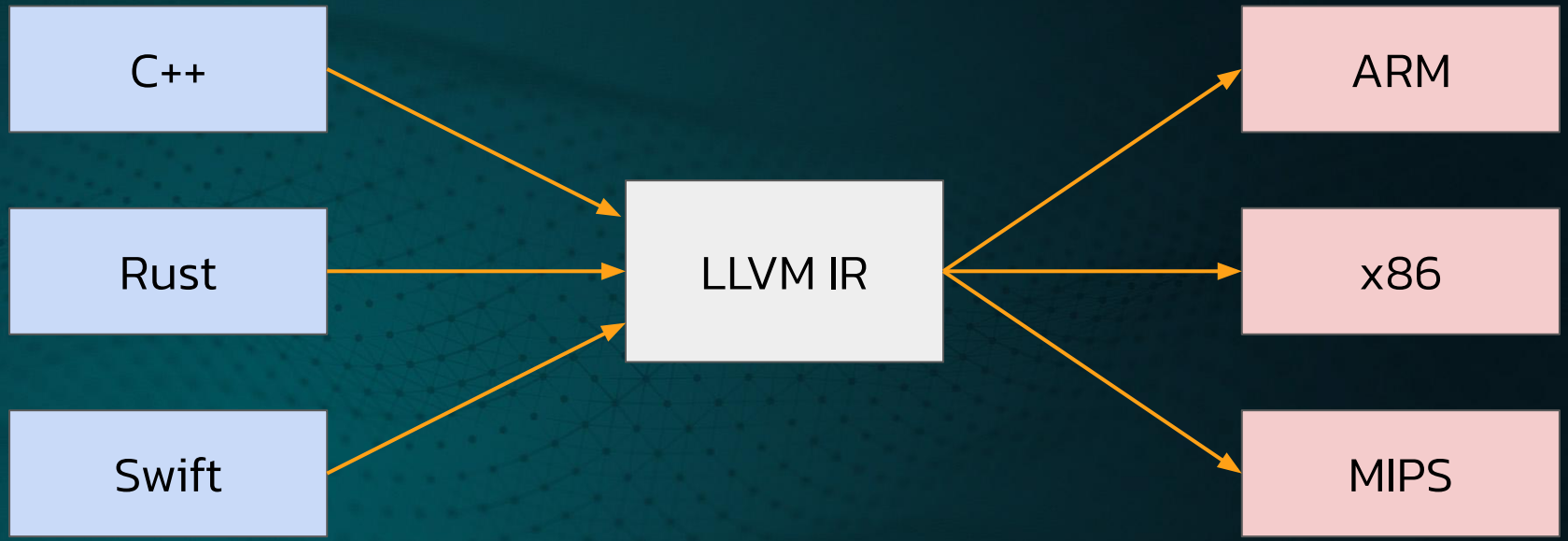
The CPU world with LLVM



The CPU world with LLVM



The CPU world with LLVM



LUNARG

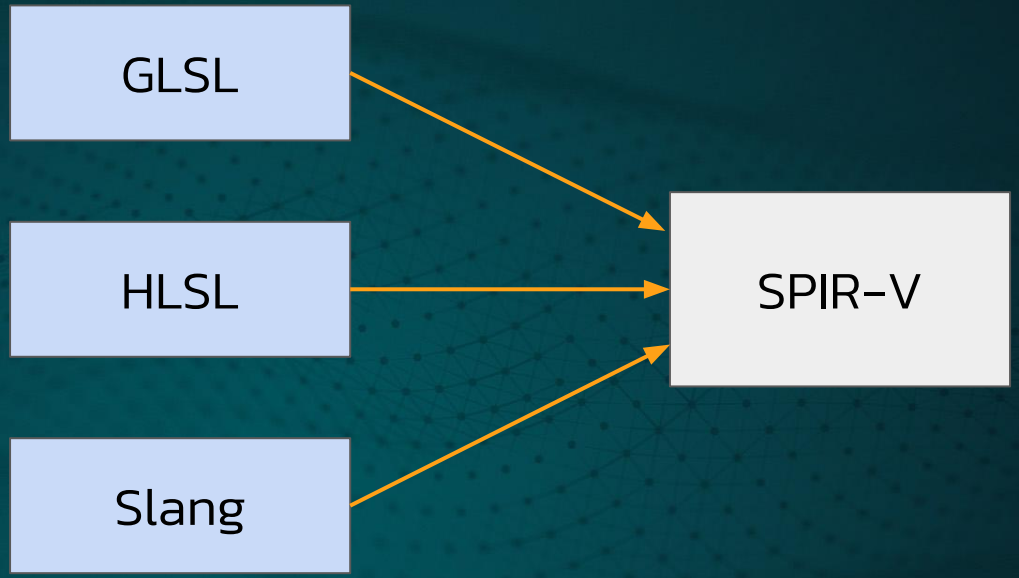
The GPU world with SPIR-V

GLSL

HLSL

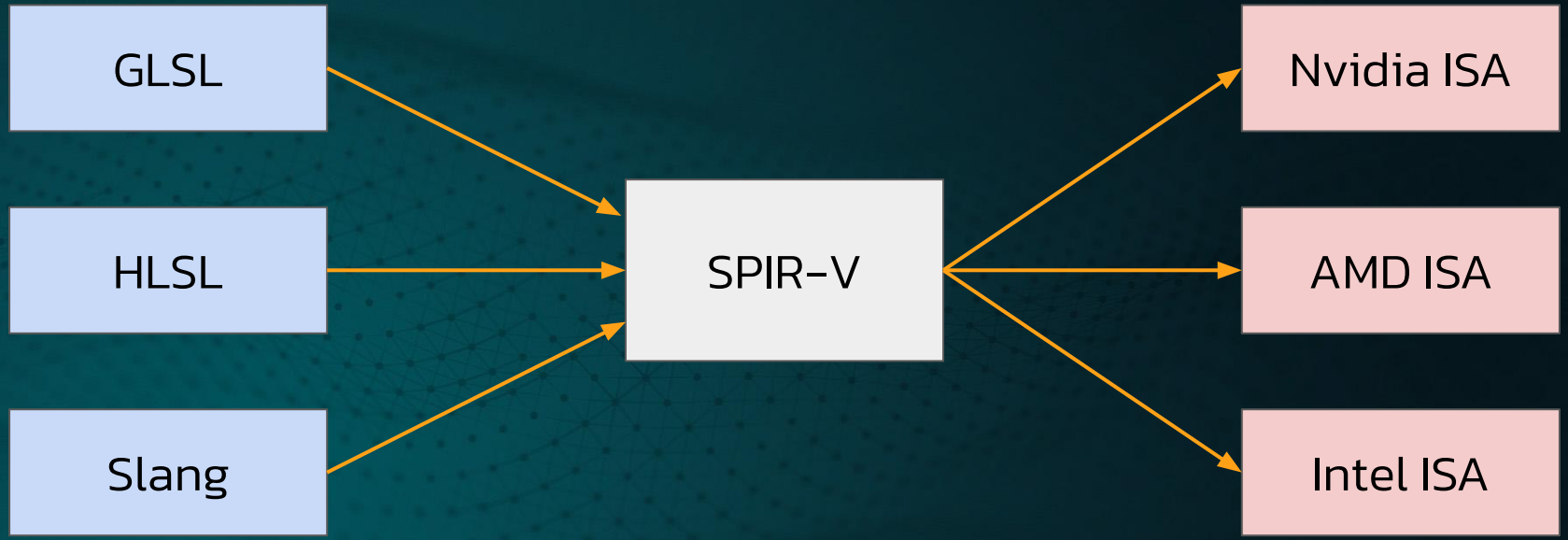
Slang

The GPU world with SPIR-V



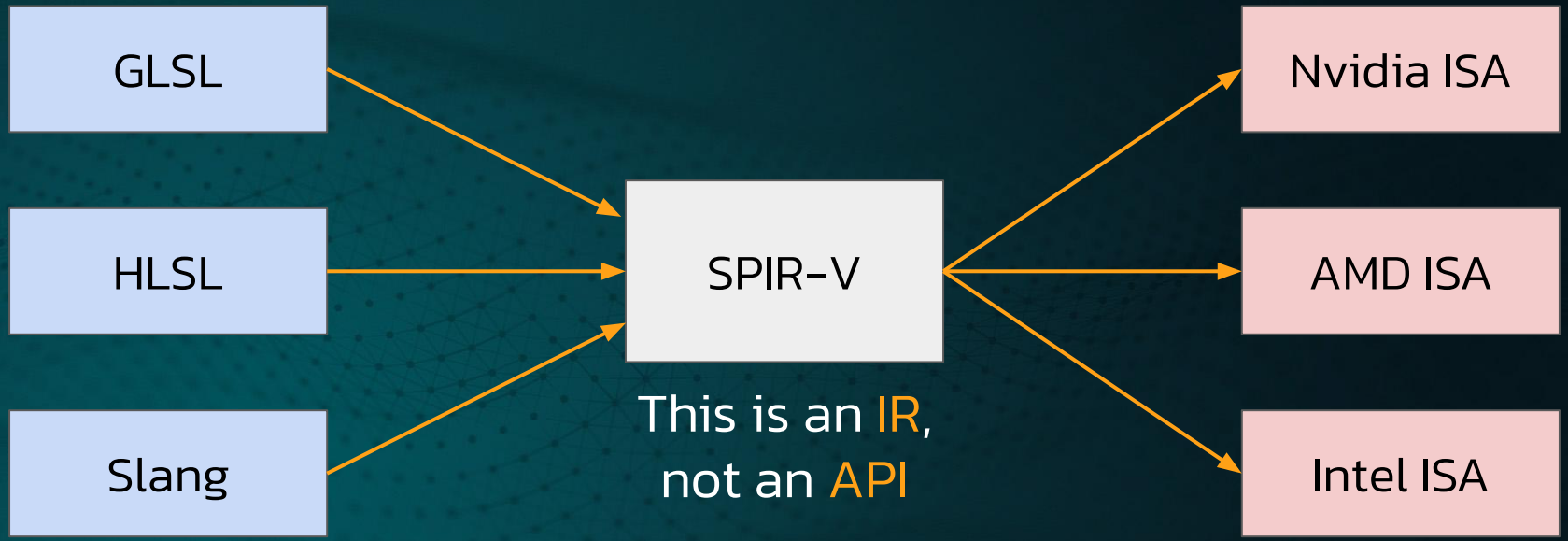
LUNARG

The GPU world with SPIR-V



LUNARG

The GPU world with SPIR-V



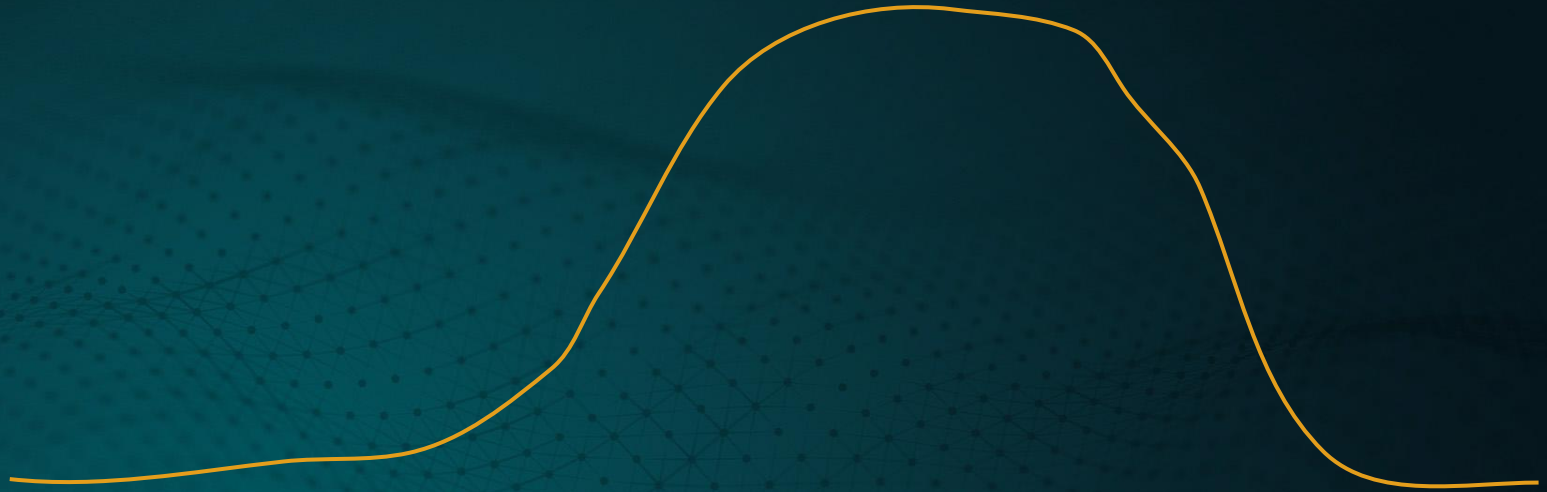
LUNARG

Why did I want to give this talk?

Distribution Curve of Attendance

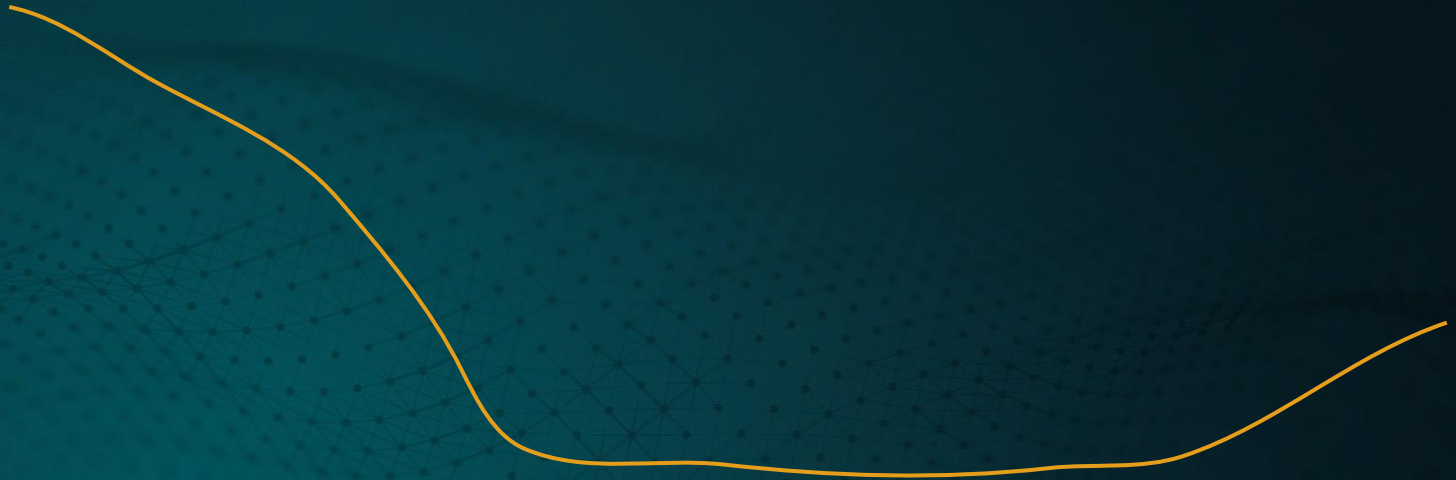
LUNARG

Distribution Curve of Attendance



LUNARG

Distribution Curve of Attendance



Distribution Curve of Attendance

Main 3 types of people using SPIR-V

WRITES

Code generators

- glslang
- dxc
- slang

LUNARG

Main 3 types of people using SPIR-V

WRITES

Code generators

- glslang
- dxc
- slang

READS

Vendor Compiler

- NIR/LLVM devs

Main 3 types of people using SPIR-V

WRITES

Code generators

- glslang
- dxc
- slang

READ/WRITE

Tooling

- Validation
- RenderDoc

READS

Vendor Compiler

- NIR/LLVM devs

Distribution Curve of Attendance

New to SPIR-V

SPIR-V Expert

LUNARG

“Why should I care about SPIR-V?”
(I just use SPIRV-Reflect anyway)

(Vulkan dev)

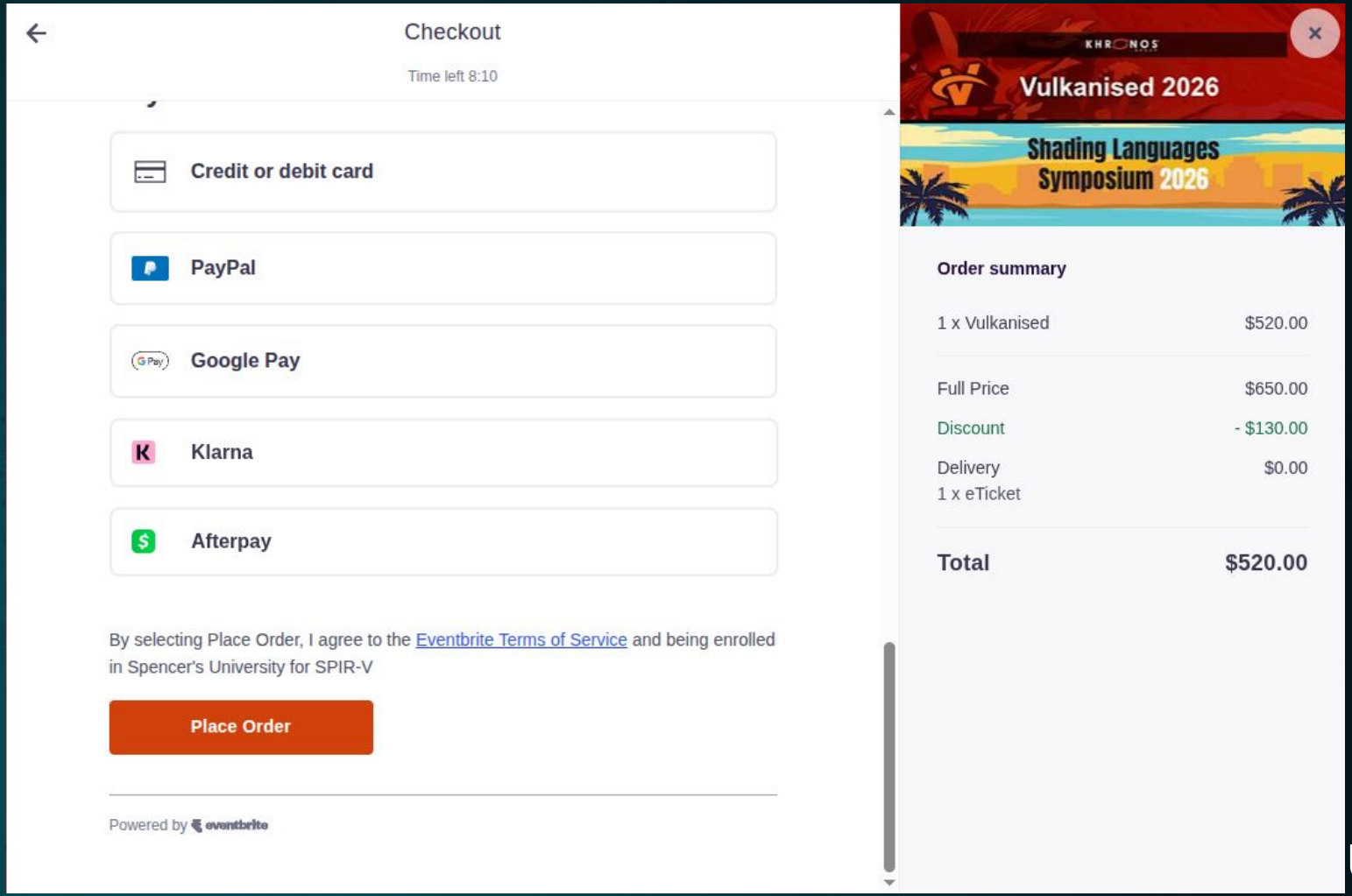
LUNARG

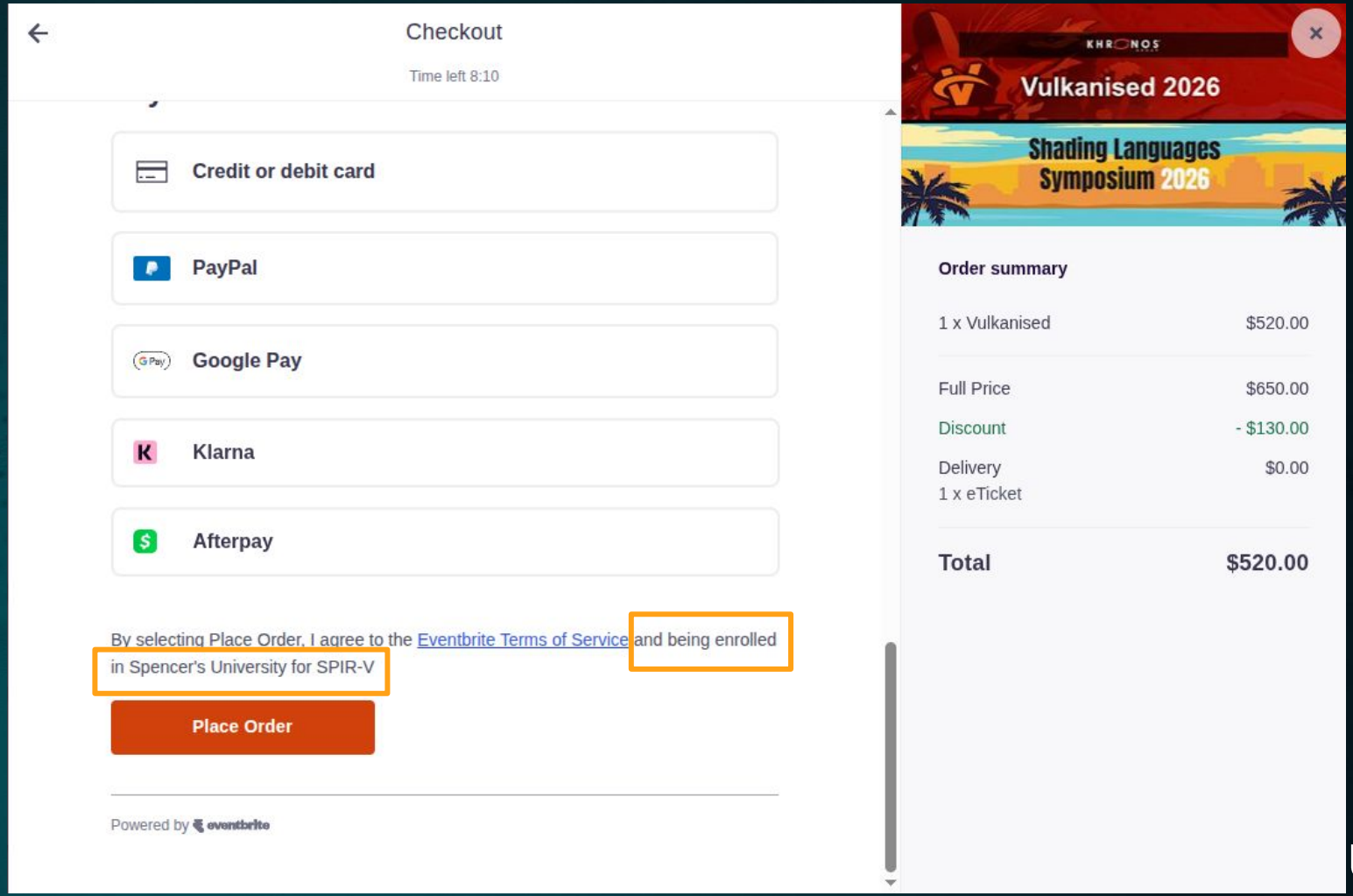
Why care about SPIR-V?

- Shading languages can have bugs!
- You might want something that SPIRV-Reflect doesn't offer
  - Or don't want another dependency
- You don't want to be at the mercy of a LLM when something is going wrong

“but Spencer… I’m a graphics developer,
not a compiler developer”

(Vulkan dev)


Checkout
Time left 8:10
Credit or debit card
PayPal
Google Pay
Klarna
Afterpay
By selecting Place Order, I agree to the Eventbrite Terms of Service and being enrolled in Spencer's University for SPIR-V
Place Order
Powered by eventbrite
KHRONOS
Vulkanised 2026
Shading Languages Symposium 2026
Order summary
1 x Vulkanised $520.00
Full Price $650.00
Discount - $130.00
Delivery $0.00
1 x eTicket
Total $520.00


Checkout
Time left 8:10
Credit or debit card
PayPal
Google Pay
Klarna
Afterpay
By selecting Place Order, I agree to the Eventbrite Terms of Service and being enrolled in Spencer's University for SPIR-V
Place Order
Powered by eventbrite
KHRONOS
Vulkanised 2026
Shading Languages Symposium 2026
Order summary
1 x Vulkanised $520.00
Full Price $650.00
Discount - $130.00
Delivery $0.00
1 x eTicket
Total $520.00

Welcome to University of Spencer's compiler course

Will teach you all you need to know compilers for this talk

Chapter 1 – SSA (Static Single Assignment)

- Just a way to represent instructions
- Always create a new and unique result ID
- Many optimizations are easier in this form

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

```
        OpStore  %a %int_0
        OpStore  %b %int_0
%13 = OpLoad  %int %a
%14 = OpIAdd   %int %13 %int_3
        OpStore  %a %14
%15 = OpLoad   %int %b
%16 = OpLoad   %int %a
%17 = OpIAdd   %int %16 %15
        OpStore  %a %17
```

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

```
        OpStore  %a %int_0
        OpStore  %b %int_0
%13 = OpLoad  %int %a
%14 = OpIAdd   %int %13 %int_3
        OpStore  %a %14
%15 = OpLoad   %int %b
%16 = OpLoad   %int %a
%17 = OpIAdd   %int %16 %15
        OpStore  %a %17
```

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

```
        OpStore  %a %int_0
        OpStore  %b %int_0
%13 = OpLoad  %int %a
%14 = OpIAdd   %int %13 %int_3
        OpStore  %a %14
%15 = OpLoad   %int %b
%16 = OpLoad   %int %a
%17 = OpIAdd   %int %16 %15
        OpStore  %a %17
```

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

```
        OpStore  %a %int_0
        OpStore  %b %int_0
%13 = OpLoad  %int %a
%14 = OpIAdd  %int %13 %int_3
        OpStore  %a %14
%15 = OpLoad  %int %b
%16 = OpLoad  %int %a
%17 = OpIAdd  %int %16 %15
        OpStore  %a %17
```

Classic example

```
void main() {
    int a = 0;
    int b = 0;
    a += 3;
    a += b;
}
```

```
        OpStore  %a %int_0
        OpStore  %b %int_0
%13 = OpLoad  %int %a
%14 = OpIAdd  %int %13 %int_3
        OpStore  %a %14
%15 = OpLoad  %int %b
%16 = OpLoad  %int %a
%17 = OpIAdd  %int %16 %15
        OpStore  %a %17
```

2 loads of the same variable, but 2 unique IDs

Classic example

%13 = OpLoad %int %a

Why are some numbers and some letters?

LUNARG

Why are some numbers and some names?

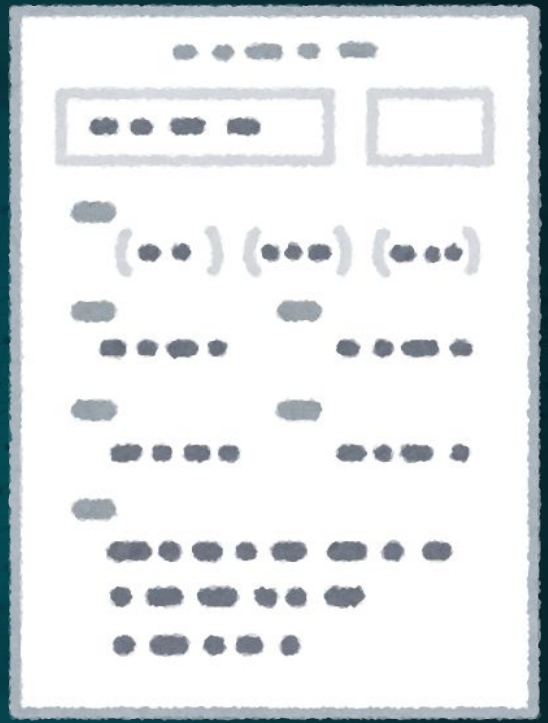
- The name can be just about anything
  - %result
  - %1
  - %2
  - %3
  - %foo_bar

YOU ALL GRADUATED!!

You can all now understand SPIR-V

LUNARG

I lied, there is a final exam!



LUNARG

Programmatically detect if your SPIR-V does 8-bit, 16-bit, 32-bit, or 64-bit adds

Programmatically detect if your SPIR-V does 8-bit, 16-bit, 32-bit, or 64-bit adds

Why?

Programmatically detect if your SPIR-V does 8-bit, 16-bit, 32-bit, or 64-bit adds

1. This is a **real** problem I've seen someone want to solve
2. It will be simple to show the code on a slide

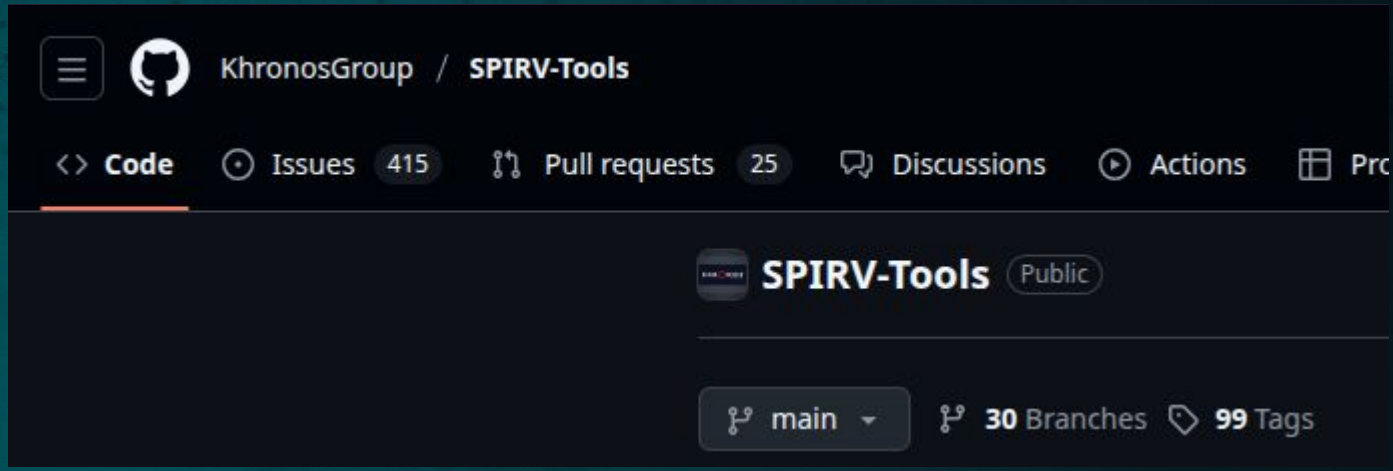
“Fine… how do I get started?

(Vulkan dev)

LUNARG

SPIRV-Tools

- Core set of tools
- Binaries found in the Vulkan SDK



LUNARG

Assembler and Disassembler

(spirv-as) (spirv-dis)

Two most basic command line tools

“Viewing” SPIR-V

- SPIR-V is a binary format
  - So you can use a hex editor if you really want…
- View in a text based format
  - Also known as “Disassembled SPIR-V”

spirv-dis

```
fricke@pop-os:~$ spirv-dis triangle.vert.spv
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 7
; Bound: 44
; Schema: 0
               OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Vertex %main "main" %outColor
               OpSource GLSL 450
               OpName %main "main"
               OpName %outColor "outColor"
               OpName %inColor "inColor"
               OpName %gl_PerVertex "gl_PerVertex"
               OpMemberName %gl_PerVertex 0 "gl_Position"
               OpName %_ ""
               OpName %UBO "UBO"
               OpMemberName %UBO 0 "projectionMatrix"
               OpMemberName %UBO 1 "modelMatrix"
               OpMemberName %UBO 2 "viewMatrix"
               OpName %ubo "ubo"
               OpName %inPos "inPos"
```

LUNARG

spirv-dis

```
fricke@pop-os:~$ spirv-dis triangle.vert.spv
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 7
; Bound: 44
; Schema: 0
               OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Vertex %main "main" %outColor
               OpSource GLSL 450
               OpName %main "main"
               OpName %outColor "outColor"
               OpName %inColor "inColor"
               OpName %gl_PerVertex "gl_PerVertex"
               OpMemberName %gl_PerVertex 0 "gl_Position"
               OpName %_ ""
               OpName %UBO "UBO"
               OpMemberName %UBO 0 "projectionMatrix"
               OpMemberName %UBO 1 "modelMatrix"
               OpMemberName %UBO 2 "viewMatrix"
               OpName %ubo "ubo"
               OpName %inPos "inPos"
```

glslang -H

```
fricke@pop-os:~$ glslang -V triangle.vert -H
triangle.vert
// Module Version 10000
// Generated by (magic number): 8000b
// Id's are bound by 44

                              Capability Shader
               1:             ExtInstImport  "GLSL.std.450"
                              MemoryModel Logical GLSL450
                              EntryPoint Vertex 4  "main" 9 11 16 34
                              Source GLSL 450
                              Name 4  "main"
                              Name 9  "outColor"
                              Name 11  "inColor"
                              Name 14  "gl_PerVertex"
                              MemberName 14(gl_PerVertex) 0  "gl_Position
                              Name 16  ""
                              Name 20  "UBO"
                              MemberName 20(UBO) 0  "projectionMatrix"
                              MemberName 20(UBO) 1  "modelMatrix"
                              MemberName 20(UBO) 2  "viewMatrix"
                              Name 22  "ubo"
                              Name 34  "inPos"
```

spirv-dis

```
fricke@pop-os:~$ spirv-dis triangle.vert.spv
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 7
; Bound: 44
; Schema: 0
                OpCapability Shader
         %1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Vertex %main "main" %outColor
                OpSource GLSL 450
                OpName %main "main"
                OpName %outColor "outColor"
                OpName %inColor "inColor"
                OpName %gl_PerVertex "gl_PerVertex"
                OpMemberName %gl_PerVertex 0 "gl_Position"
                OpName %_ ""
                OpName %UBO "UBO"
                OpMemberName %UBO 0 "projectionMatrix"
                OpMemberName %UBO 1 "modelMatrix"
                OpMemberName %UBO 2 "viewMatrix"
                OpName %ubo "ubo"
                OpName %inPos "inPos"
```

glslang -H

```
fricke@pop-os:~$ glslang -V triangle.vert -H
triangle.vert
// Module Version 10000
// Generated by (magic number): 8000b
// Id's are bound by 44

                Capability Shader
1:              ExtInstImport  "GLSL.std.450"
                MemoryModel Logical GLSL450
                EntryPoint Vertex 4  "main" 9 11 16 34
                Source GLSL 450
                Name 4  "main"
                Name 9  "outColor"
                Name 11  "inColor"
                Name 14  "gl_PerVertex"
                MemberName 14(gl_PerVertex) 0  "gl_Position
                Name 16  ""
                Name 20  "UBO"
                MemberName 20(UBO) 0  "projectionMatrix"
                MemberName 20(UBO) 1  "modelMatrix"
                MemberName 20(UBO) 2  "viewMatrix"
                Name 22  "ubo"
                Name 34  "inPos"
```

No “official” disassembly

- There is a spec how the binary format is laid out
- Every tool can represent SPIR-V how they want
- Reality – most people just follow what spirv-as consumes

SPIRV-Visualizer

- Web based way to view SPIR-V
- Front end only, can easily run offline
- Repo: https://github.com/KhronosGroup/SPIRV-Visualizer
- Live link: https://www.khronos.org/spirv/visualizer/

Collapse All
Expand All
Larger Text
Use OpNames
Insert Constants
Copy To Clipboard
Clear All
Load new SPIR-V file

SPIR-V Visualizer

Based on SPIR-V grammar 1.6.4

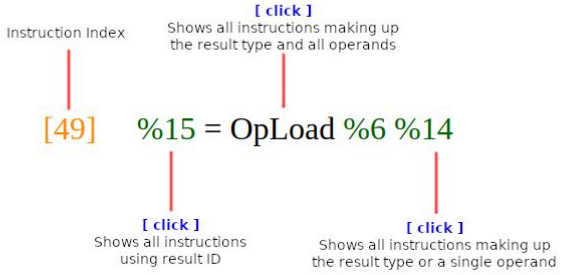
Select SPIR-V binary file to load

Choose File No file chosen

OR

Paste SPIR-V disassembly on the left (and press enter)

How to use



Collapse All
Expand All
Larger Text
Use OpNames
Insert Constants
Copy To Clipboard
Clear All
Load new SPIR-V file

SPIR-V Visualizer

Based on SPIR-V grammar 1.6.4

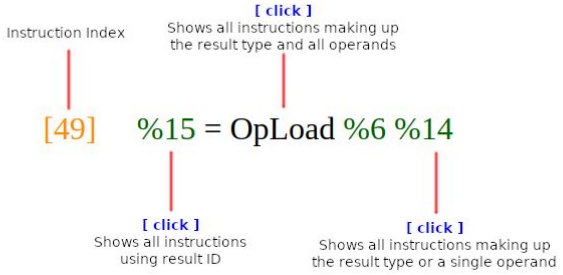
Select SPIR-V binary file to load

Choose File No file chosen

OR

Paste SPIR-V disassembly on the left (and press enter)

How to use



Collapse All
Expand All
Larger Text
Use OpNames
Insert Constants
Copy To Clipboard
Clear All
Load new SPIR-V file

```
                OpCapability Shader
                OpMemoryModel Logical GLSL450
                OpEntryPoint Vertex %vertex "main" %entryPointParams %gl_Position
%entryPointParam_vertex_uv %entryPointParam_vertex_color %entryPointParam_vertex_textureIndex
%v_position %v_uv %v_color
                OpSource Slang 1
                OpName %DisplayRect_std430 "DisplayRect_std430"
                OpMemberName %DisplayRect_std430 0 "translation"
                OpMemberName %DisplayRect_std430 1 "size"
                OpMemberName %DisplayRect_std430 2 "textureIndex"
                OpName %EntryPointParams_std430 "EntryPointParams_std430"
                OpMemberName %EntryPointParams_std430 0 "display"
                OpName %entryPointParams "entryPointParams"
                OpName %entryPointParam_vertex_uv "entryPointParam_vertex.uv"
                OpName %entryPointParam_vertex_color "entryPointParam_vertex.color"
                OpName %entryPointParam_vertex_textureIndex "entryPointParam_vertex.textureIndex"
                OpName %v_position "v.position"
                OpName %v_uv "v.uv"
                OpName %v_color "v.color"
                OpName %textures "textures"
                OpName %VSOutput "VSOutput"
                OpMemberName %VSOutput 0 "position"
                OpMemberName %VSOutput 1 "uv"
                OpMemberName %VSOutput 2 "color"
                OpMemberName %VSOutput 3 "textureIndex"
                OpName %o "o"
                OpName %o "o"
                OpName %DisplayRect_std430_logical "DisplayRect_std430_logical"
                OpMemberName %DisplayRect_std430_logical 0 "translation"
                OpMemberName %DisplayRect_std430_logical 1 "size"
                OpMemberName %DisplayRect_std430_logical 2 "textureIndex"
                OpName %alpha "alpha"
                OpName %blue "blue"
                OpName %green "green"
                OpName %red "red"
                OpName %vertex "vertex"
                OpMemberDecorate %DisplayRect_std430 0 Offset 0
                OpMemberDecorate %DisplayRect_std430 1 Offset 8
                OpMemberDecorate %DisplayRect_std430 2 Offset 16
                OpDecorate %EntryPointParams_std430 Block
                OpMemberDecorate %EntryPointParams_std430 0 Offset 0
                OpDecorate %gl_Position BuiltIn Position
                OpDecorate %entryPointParam_vertex_uv Location 0
                OpDecorate %entryPointParam_vertex_color Location 1
                OpDecorate %entryPointParam_vertex_textureIndex Location 2
                OpDecorate %entryPointParam_vertex_textureIndex Flat
                OpDecorate %v_position Location 0
                OpDecorate %v_uv Location 1
                OpDecorate %v_color Location 2
                OpDecorate %textures Binding 0
                OpDecorate %textures DescriptorSet 0
        %float = OpTypeFloat 32
      %v2float = OpTypeVector %float 2
         %uint = OpTypeInt 32 0
%DisplayRect_std430 = OpTypeStruct %v2float %v2float %uint
%EntryPointParams_std430 = OpTypeStruct %DisplayRect_std430
% ptr_PushConstant_EntryPointParams_std430 = OpTypePointer PushConstant %EntryPointParams_std430
      %v4float = OpTypeVector %float 4
% ptr_Output_v4float = OpTypePointer Output %v4float
% ptr_Output_v2float = OpTypePointer Output %v2float
% ptr_Output_uint = OpTypePointer Output %uint
% ptr_Input_v2float = OpTypePointer Input %v2float
% ptr_Input_uint = OpTypePointer Input %uint
          %22 = OpTypeImage %float 2D 2 0 0 1 Unknown
          %23 = OpTypeSampledImage %22
```

SPIR-V Visualizer

Based on SPIR-V grammar 1.6.4

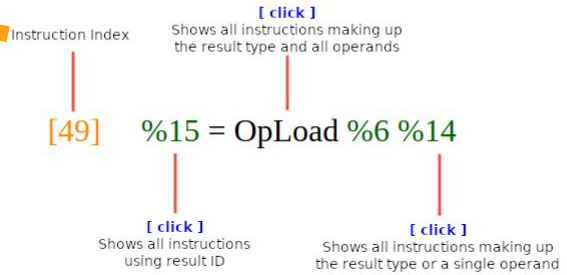
Select SPIR-V binary file to load

Choose File No file chosen

OR

Paste SPIR-V disassembly on the left (and press enter)

How to use



Collapse All | Expand All | Larger Text | Use OpNames | Insert Constants | Copy To Clipboard | Clear All | Load new SPIR-V file

```
    0x07230203, 0x00010300, 0x0008000b, 0x00000097, 0x00000000, 0x00020011, 0x00000001,
0x00020011, 0x00000005, 0x00020011,
    0x0000000b, 0x00020011, 0x000014e3, 0x0009000a, 0x5f565053, 0x5f52484b, 0x73796870,
0x6c616369, 0x6f74735f, 0x65676172,
    0x66756425f, 0x00726566, 0x0006000b, 0x00000001, 0x4c534c47, 0x6474732e, 0x3035342e,
0x00000000, 0x0003000e, 0x000014e4,
    0x00000001, 0x00030003, 0x00000002, 0x000001c2, 0x00070004, 0x415f4c47, 0x675f4252,
0x735f7570, 0x65646168, 0x6e695f72,
    0x00343674, 0x00070004, 0x455f4c47, 0x625f5458, 0x65666675, 0x65725f72, 0x65726566,
0x0065636e, 0x00080004, 0x455f4c47,
    0x625f5458, 0x65666675, 0x65725f72, 0x65726566, 0x3265636e, 0x00000000, 0x00090004,
0x455f4c47, 0x625f5458, 0x65666675,
    0x65725f72, 0x65726566, 0x5f65636e, 0x63657675, 0x00000032, 0x00080004, 0x455f4c47,
0x735f5458, 0x616c6163, 0x6c625f72,
    0x5f6b636f, 0x6f79616c, 0x00007475, 0x000a0004, 0x475f4c47, 0x4c474f4f, 0x70635f45,
0x74735f70, 0x5f656c79, 0x656e696c,
    0x7269645f, 0x69746365, 0x00006576, 0x00080004, 0x475f4c47, 0x4c474f4f, 0x6e695f45,
0x64756c63, 0x69645f65, 0x74636572,
    0x00657669, 0x000e0005, 0x0000000a, 0x74736e69, 0x6675625f, 0x5f726566, 0x69766564,
0x615f6563, 0x65726464, 0x725f7373,
    0x65676e61, 0x3b317528, 0x31343675, 0x3b31753b, 0x003b3175, 0x00050005, 0x00000006,
0x74736e69, 0x66666f5f, 0x00746573,
    0x00040005, 0x00000007, 0x72646461, 0x00000000, 0x00050005, 0x00000008, 0x65636361,
0x745f7373, 0x00657079, 0x00070005,
    0x00000009, 0x65636361, 0x625f7373, 0x5f657479, 0x657a6973, 0x00000000, 0x000e0005,
0x0000000a, 0x74736e69, 0x66756425f,
    0x5f726566, 0x69766564, 0x615f6563, 0x65726464, 0x615f7373, 0x6e67696c, 0x3b317528,
0x31343675, 0x3b31753b, 0x003b3175,
    0x00050005, 0x0000000c, 0x74736e69, 0x66666f5f, 0x00746573, 0x00040005, 0x0000000d,
0x72646461, 0x00000000, 0x00050005,
    0x0000000e, 0x65636361, 0x745f7373, 0x00657079, 0x00050005, 0x0000000f, 0x67696c61,
0x6e656d6e, 0x00000074, 0x00040005,
    0x00000012, 0x676e6152, 0x00000065, 0x00050006, 0x00000012, 0x00000000, 0x69676562,
0x0000006e, 0x00040006, 0x00000012,
    0x00000001, 0x00646e65, 0x00050005, 0x00000014, 0x68636163, 0x61725f65, 0x0065676e,
0x00070005, 0x00000016, 0x66667542,
    0x72646441, 0x75706e49, 0x66754274, 0x00726566, 0x00070006, 0x00000016, 0x00000000,
0x5f616462, 0x676e6172, 0x705f7365,
    0x00007274, 0x00040005, 0x00000017, 0x676e6152, 0x00000065, 0x00050006, 0x00000017,
0x00000000, 0x69676562, 0x0000006e,
    0x00040006, 0x00000017, 0x00000001, 0x00646e65, 0x00090005, 0x00000019, 0x66667542,
0x65447265, 0x65636976, 0x72646441,
    0x52737365, 0x65676e61, 0x00000073, 0x00070006, 0x00000019, 0x00000000, 0x5f616462,
0x676e6172, 0x6f635f65, 0x00746e75,
    0x00070006, 0x00000019, 0x00000001, 0x64646170, 0x5f676e69, 0x73756e75, 0x00006465,
0x00060006, 0x00000019, 0x00000002,
    0x5f616462, 0x676e6172, 0x00007365, 0x00030005, 0x0000001b, 0x00000000, 0x00050005,
0x00000023, 0x65646e69, 0x61635f78,
    0x00656863, 0x00040005, 0x0000003e, 0x676e6172, 0x00695f65, 0x00040005, 0x0000004c,
0x676e6172, 0x00000065, 0x00060005,
    0x00000075, 0x6f727245, 0x79615072, 0x64616f6c, 0x00000000, 0x00060006, 0x00000075,
0x00000000, 0x74736e69, 0x66666f5f,
    0x00746573, 0x00090006, 0x00000075, 0x00000001, 0x64616873, 0x655f7265, 0x726f7272,
0x636e655f, 0x6e69646f, 0x00000067,
    0x00060006, 0x00000075, 0x00000002, 0x61726170, 0x6574656d, 0x00305f72, 0x00060006,
0x00000075, 0x00000003, 0x61726170,
    0x65746564, 0x00315f72, 0x00060006, 0x00000075, 0x00000004, 0x61726170, 0x6574656d,
0x00325f72, 0x00060005, 0x00000077,
    0x6f727265, 0x61705f72, 0x616f6c79, 0x00000064, 0x00090005, 0x00000078, 0x63657053,
0x736e6f43, 0x746e6174, 0x6b6e694c,
    0x64616853, 0x64497265, 0x00000000, 0x000d0047, 0x0000000a, 0x00000029, 0x74736e69,
0x6675625f, 0x5f726566, 0x69766564,
    0x615f6563, 0x65726464, 0x725f7373, 0x65676e61, 0x00000000, 0x00000000, 0x000d0047,
0x00000010, 0x00000029, 0x74736e69,
    0x66756425f, 0x5f726566, 0x69766564, 0x615f6563, 0x65726464, 0x615f7373, 0x6e67696c,
```

SPIR-V Visualizer

Based on SPIR-V grammar 1.6.4

Select SPIR-V binary file to load

Choose File | No file chosen

OR

Paste SPIR-V disassembly on the left (and press enter)

How to use

Instruction Index

[click]
Shows all instructions making up the result type and all operands

[49] %15 = OpLoad %6 %14

[click]
Shows all instructions using result ID

[click]
Shows all instructions making up the result type or a single operand

Collapse All
Expand All
Larger Text
Use OpNames
Insert Constants
Copy To Clipboard
Clear All
Load new SPIR-V file
Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

SPIR-V 1.6 (Max ID Bound: 112)

Mode Setting

```
[0]  OpCapability Shader
[1]  OpMemoryModel Logical GLSL450
[2]  OpEntryPoint Vertex %29 "main" %7 %10 %12 %13 %15 %17 %18 %20
```

Debug Information

```
[3]  OpSource Slang 1
[4]  OpName %2 "DisplayRect_std430"
[5]  OpMemberName %2 0 "translation"
[6]  OpMemberName %2 1 "size"
[7]  OpMemberName %2 2 "textureIndex"
[8]  OpName %1 "EntryPointParams_std430"
[9]  OpMemberName %1 0 "display"
[10] OpName %7 "entryPointParams"
[11] OpName %12 "entryPointParam_vertex.uv"
[12] OpName %13 "entryPointParam_vertex.color"
[13] OpName %15 "entryPointParam_vertex.textureIndex"
[14] OpName %17 "v.position"
[15] OpName %18 "v.uv"
[16] OpName %20 "v.color"
[17] OpName %27 "textures"
[18] OpName %32 "VSOutput"
[19] OpMemberName %32 0 "position"
[20] OpMemberName %32 1 "uv"
[21] OpMemberName %32 2 "color"
[22] OpMemberName %32 3 "textureIndex"
[23] OpName %34 "o"
[24] OpName %34 "o"
[25] OpName %35 "DisplayRect_std430_logical"
[26] OpMemberName %35 0 "translation"
[27] OpMemberName %35 1 "size"
[28] OpMemberName %35 2 "textureIndex"
[29] OpName %48 "alpha"
[30] OpName %54 "blue"
[31] OpName %59 "green"
[32] OpName %64 "red"
[33] OpName %29 "vertex"
```

Annotations

```
[34] OpMemberDecorate %2 0 Offset 0
[35] OpMemberDecorate %2 1 Offset 8
[36] OpMemberDecorate %2 2 Offset 16
[37] OpDecorate %1 Block
[38] OpMemberDecorate %1 0 Offset 0
[39] OpDecorate %10 BuiltIn Position
[40] OpDecorate %12 Location 0
[41] OpDecorate %13 Location 1
[42] OpDecorate %15 Location 2
[43] OpDecorate %15 Flat
[44] OpDecorate %17 Location 0
[45] OpDecorate %18 Location 1
[46] OpDecorate %20 Location 2
[47] OpDecorate %27 Binding 0
[48] OpDecorate %27 DescriptorSet 0
```

Types, variables and constants

```
[49] %3 = OpTypeFloat 32
```

Collapse All | Expand All | Larger Text | Use OpNames | Insert Constants | Copy To Clipboard | Clear All | Load new SPIR-V file

Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

SPIR-V 1.6 (Max ID Bound: 112)

Mode Setting

```
[0]  OpCapability Shader
[1]  OpMemoryModel Logical GLSL450
[2]  OpEntryPoint Vertex %29 "main" %7 %10 %12 %13 %15 %17 %18 %20
```

Debug Information

```
[3]  OpSource Slang 1
[4]  OpName %2 "DisplayRect_std430"
[5]  OpMemberName %2 0 "translation"
[6]  OpMemberName %2 1 "size"
[7]  OpMemberName %2 2 "textureIndex"
[8]  OpName %1 "EntryPointParams_std430"
[9]  OpMemberName %1 0 "display"
[10] OpName %7 "entryPointParams"
[11] OpName %12 "entryPointParam_vertex.uv"
[12] OpName %13 "entryPointParam_vertex.color"
[13] OpName %15 "entryPointParam_vertex.textureIndex"
[14] OpName %17 "v.position"
[15] OpName %18 "v.uv"
[16] OpName %20 "v.color"
[17] OpName %27 "textures"
[18] OpName %32 "VSOutput"
[19] OpMemberName %32 0 "position"
[20] OpMemberName %32 1 "uv"
[21] OpMemberName %32 2 "color"
[22] OpMemberName %32 3 "textureIndex"
[23] OpName %34 "o"
[24] OpName %34 "o"
[25] OpName %35 "DisplayRect_std430_0"
[26] OpMemberName %35 0 "translation"
[27] OpMemberName %35 1 "size"
[28] OpMemberName %35 2 "textureIndex"
[29] OpName %48 "alpha"
[30] OpName %54 "blue"
[31] OpName %59 "green"
[32] OpName %64 "red"
[33] OpName %29 "vertex"
```

Annotations

```
[34] OpMemberDecorate %2 0 Offset 0
[35] OpMemberDecorate %2 1 Offset 8
[36] OpMemberDecorate %2 2 Offset 16
[37] OpDecorate %1 Block
[38] OpMemberDecorate %1 0 Offset 0
[39] OpDecorate %10 BuiltIn Position
[40] OpDecorate %12 Location 0
[41] OpDecorate %13 Location 1
[42] OpDecorate %15 Location 2
[43] OpDecorate %15 Flat
[44] OpDecorate %17 Location 0
[45] OpDecorate %18 Location 1
[46] OpDecorate %20 Location 2
[47] OpDecorate %27 Binding 0
[48] OpDecorate %27 DescriptorSet 0
```

Types, variables and constants

```
[49] %3 = OpTypeFloat 32
```

Collapse All | Expand All | Larger Text | Use OpNames | Insert Constants | Copy To Clipboard | Clear All | Load new SPIR-V file

Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

SPIR-V 1.6 (Max ID Bound: 112)

Mode Setting

```
[0]  OpCapability Shader
[1]  OpMemoryModel Logical GLSL450
[2]  OpEntryPoint Vertex %29 "main" %7 %10 %12 %13 %15 %17 %18 %20
```

Debug Information

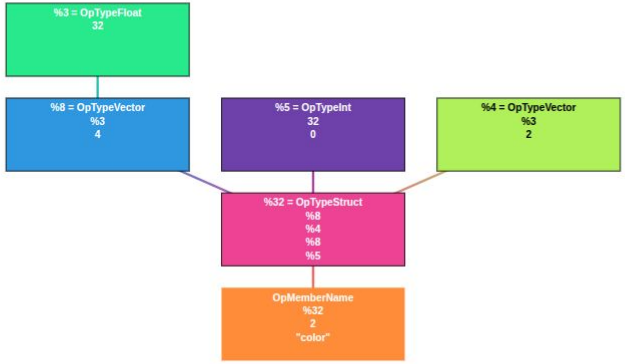
```
[3]  OpSource Slang 1
[4]  OpName %2 "DisplayRect_std430"
[5]  OpMemberName %2 0 "translation"
[6]  OpMemberName %2 1 "size"
[7]  OpMemberName %2 2 "textureIndex"
[8]  OpName %1 "EntryPointParams_std430"
[9]  OpMemberName %1 0 "display"
[10] OpName %7 "entryPointParams"
[11] OpName %12 "entryPointParam_vertex.uv"
[12] OpName %13 "entryPointParam_vertex.color"
[13] OpName %15 "entryPointParam_vertex.textureIndex"
[14] OpName %17 "v.position"
[15] OpName %18 "v.uv"
[16] OpName %20 "v.color"
[17] OpName %27 "textures"
[18] OpName %32 "VSOutput"
[19] OpMemberName %32 0 "position"
[20] OpMemberName %32 1 "uv"
[21] OpMemberName %32 2 "color"
[22] OpMemberName %32 3 "textureIndex"
[23] OpName %34 "o"
[24] OpName %34 "o"
[25] OpName %35 "DisplayRect_std430_logical"
[26] OpMemberName %35 0 "translation"
[27] OpMemberName %35 1 "size"
[28] OpMemberName %35 2 "textureIndex"
[29] OpName %48 "alpha"
[30] OpName %54 "blue"
[31] OpName %59 "green"
[32] OpName %64 "red"
[33] OpName %29 "vertex"
```

Annotations

```
[34] OpMemberDecorate %2 0 Offset 0
[35] OpMemberDecorate %2 1 Offset 8
[36] OpMemberDecorate %2 2 Offset 16
[37] OpDecorate %1 Block
[38] OpMemberDecorate %1 0 Offset 0
[39] OpDecorate %10 BuiltIn Position
[40] OpDecorate %12 Location 0
[41] OpDecorate %13 Location 1
[42] OpDecorate %15 Location 2
[43] OpDecorate %15 Flat
[44] OpDecorate %17 Location 0
[45] OpDecorate %18 Location 1
[46] OpDecorate %20 Location 2
[47] OpDecorate %27 Binding 0
[48] OpDecorate %27 DescriptorSet 0
```

Types, variables and constants

```
[49] %3 = OpTypeFloat 32
```



Collapse All | Expand All | Larger Text | Use OpNames | Insert Constants | Copy To Clipboard | Clear All | Load new SPIR-V file

Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

SPIR-V 1.6 (Max ID Bound: 112)

Mode Setting

```
[0]  OpCapability Shader
[1]  OpMemoryModel Logical GLSL450
[2]  OpEntryPoint Vertex %29 "main" %7 %10 %12 %13 %15 %17 %18 %20
```

Debug Information

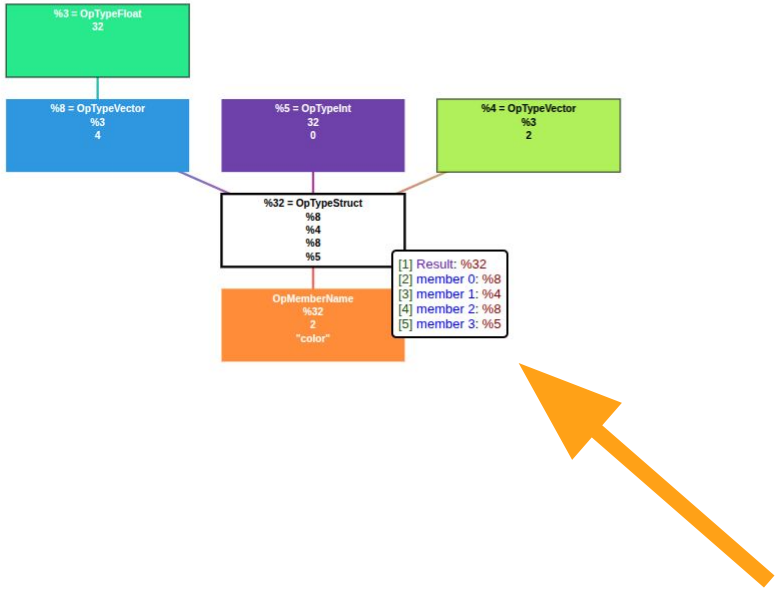
```
[3]  OpSource Slang 1
[4]  OpName %2 "DisplayRect_std430"
[5]  OpMemberName %2 0 "translation"
[6]  OpMemberName %2 1 "size"
[7]  OpMemberName %2 2 "textureIndex"
[8]  OpName %1 "EntryPointParams_std430"
[9]  OpMemberName %1 0 "display"
[10] OpName %7 "entryPointParams"
[11] OpName %12 "entryPointParam_vertex.uv"
[12] OpName %13 "entryPointParam_vertex.color"
[13] OpName %15 "entryPointParam_vertex.textureIndex"
[14] OpName %17 "v.position"
[15] OpName %18 "v.uv"
[16] OpName %20 "v.color"
[17] OpName %27 "textures"
[18] OpName %32 "VSOutput"
[19] OpMemberName %32 0 "position"
[20] OpMemberName %32 1 "uv"
[21] OpMemberName %32 2 "color"
[22] OpMemberName %32 3 "textureIndex"
[23] OpName %34 "o"
[24] OpName %34 "o"
[25] OpName %35 "DisplayRect_std430_logical"
[26] OpMemberName %35 0 "translation"
[27] OpMemberName %35 1 "size"
[28] OpMemberName %35 2 "textureIndex"
[29] OpName %48 "alpha"
[30] OpName %54 "blue"
[31] OpName %59 "green"
[32] OpName %64 "red"
[33] OpName %29 "vertex"
```

Annotations

```
[34] OpMemberDecorate %2 0 Offset 0
[35] OpMemberDecorate %2 1 Offset 8
[36] OpMemberDecorate %2 2 Offset 16
[37] OpDecorate %1 Block
[38] OpMemberDecorate %1 0 Offset 0
[39] OpDecorate %10 BuiltIn Position
[40] OpDecorate %12 Location 0
[41] OpDecorate %13 Location 1
[42] OpDecorate %15 Location 2
[43] OpDecorate %15 Flat
[44] OpDecorate %17 Location 0
[45] OpDecorate %18 Location 1
[46] OpDecorate %20 Location 2
[47] OpDecorate %27 Binding 0
[48] OpDecorate %27 DescriptorSet 0
```

Types, variables and constants

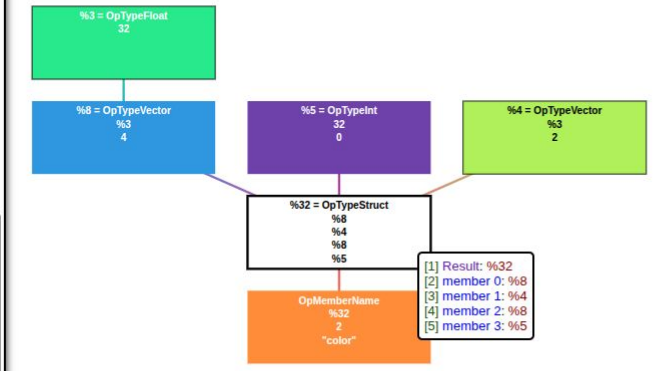
```
[49] %3 = OpTypeFloat 32
```



Collapse All | Expand All | Larger Text | Use OpNames | Copy To Clipboard | Clear All | Load new SPIR-V file

Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

```
[42]  OpDecorate %15 Location 2
[43]  OpDecorate %15 Flat
[44]  OpDecorate %17 Location 0
[45]  OpDecorate %18 Location 1
[46]  OpDecorate %20 Location 2
[47]  OpDecorate %27 Binding 0
[48]  OpDecorate %27 DescriptorSet 0

Types, variables and constants

[49]  %3 = OpTypeFloat 32
[50]  %4 = OpTypeVector %3 2
[51]  %5 = OpTypeInt 32 0
[52]  %2 = OpTypeStruct %4 %4 %5
[53]  %1 = OpTypeStruct %2
[54]  %6 = OpTypePointer PushConstant %1
[55]  %8 = OpTypeVector %3 4
[56]  %9 = OpTypePointer Output %8
[57]  %11 = OpTypePointer Output %4
[58]  %14 = OpTypePointer Output %5
[59]  %16 = OpTypePointer Input %4
[60]  %19 = OpTypePointer Input %5
[61]  %22 = OpTypeImage %3 2D 2 0 0 1 Unknown
[62]  %23 = OpTypeSampledImage %22
[63]  %24 = OpTypeInt 32 1
[64]  %25 = OpConstant %24 1024
[65]  %21 = OpTypeArray %23 %25
[66]  %26 = OpTypePointer UniformConstant %21
[67]  %28 = OpTypeVoid
[68]  %30 = OpTypeFunction %28
[69]  %32 = OpTypeStruct %8 %4 %8 %5
[70]  %33 = OpTypePointer Function %32
[71]  %35 = OpTypeStruct %4 %4 %5
[72]  %36 = OpTypePointer Function %35
[73]  %44 = OpConstant %24 24
[74]  %46 = OpConstant %5 255
[75]  %49 = OpConstant %3 255.0
[76]  %51 = OpConstant %24 16
[77]  %56 = OpConstant %24 8
[78]  %61 = OpConstant %24 0
[79]  %65 = OpTypePointer Function %8
[80]  %67 = OpTypePointer PushConstant %2
[81]  %72 = OpConstant %24 1
[82]  %73 = OpTypePointer Function %4
[83]  %84 = OpConstant %3 0.0
[84]  %85 = OpConstant %3 1.0
[85]  %89 = OpConstant %24 2
[86]  %93 = OpConstant %24 3
[87]  %94 = OpTypePointer Function %5
[88]  %7 = OpVariable %6 PushConstant
[89]  %10 = OpVariable %9 Output
[90]  %12 = OpVariable %11 Output
[91]  %13 = OpVariable %9 Output
[92]  %15 = OpVariable %14 Output
[93]  %17 = OpVariable %16 Input
[94]  %18 = OpVariable %16 Input
[95]  %20 = OpVariable %19 Input
[96]  %27 = OpVariable %26 UniformConstant

Function 97
```



Collapse All
Expand All
Larger Text
Use OpNames
Insert Constants
Copy To Clipboard
Clear All
Load new SPIR-V file
Loaded: dump_1_before.spv
binary parsed in 0.006 seconds

SPIR-V 1.6 (Max ID Bound: 112)

Mode Setting

```
[0]  OpCapability Shader
[1]  OpMemoryModel Logical GLSL450
[2]  OpEntryPoint Vertex %29 "main" %7 %10 %12 %13 %15 %17 %18 %20
```

Debug Information

```
[3]  OpSource Slang 1
[4]  OpName %2 "DisplayRect_std430"
[5]  OpMemberName %2 0 "translation"
[6]  OpMemberName %2 1 "size"
[7]  OpMemberName %2 2 "textureIndex"
[8]  OpName %1 "EntryPointParams_std430"
[9]  OpMemberName %1 0 "display"
[10] OpName %7 "entryPointParams"
[11] OpName %12 "entryPointParam_vertex.uv"
[12] OpName %13 "entryPointParam_vertex.color"
[13] OpName %15 "entryPointParam_vertex.textureIndex"
[14] OpName %17 "v.position"
[15] OpName %18 "v.uv"
[16] OpName %20 "v.color"
[17] OpName %27 "textures"
[18] OpName %32 "VSOutput"
[19] OpMemberName %32 0 "position"
[20] OpMemberName %32 1 "uv"
[21] OpMemberName %32 2 "color"
[22] OpMemberName %32 3 "textureIndex"
[23] OpName %34 "o"
[24] OpName %34 "o"
[25] OpName %35 "DisplayRect_std430_logical"
[26] OpMemberName %35 0 "translation"
[27] OpMemberName %35 1 "size"
[28] OpMemberName %35 2 "textureIndex"
[29] OpName %48 "alpha"
[30] OpName %54 "blue"
[31] OpName %59 "green"
[32] OpName %64 "red"
[33] OpName %29 "vertex"
```

Annotations

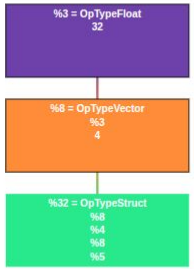
Types, variables and constants

Function 97

```
[97]  %29 = OpFunction %28 None %30
```

Label 98

```
[160] OpFunctionEnd
```



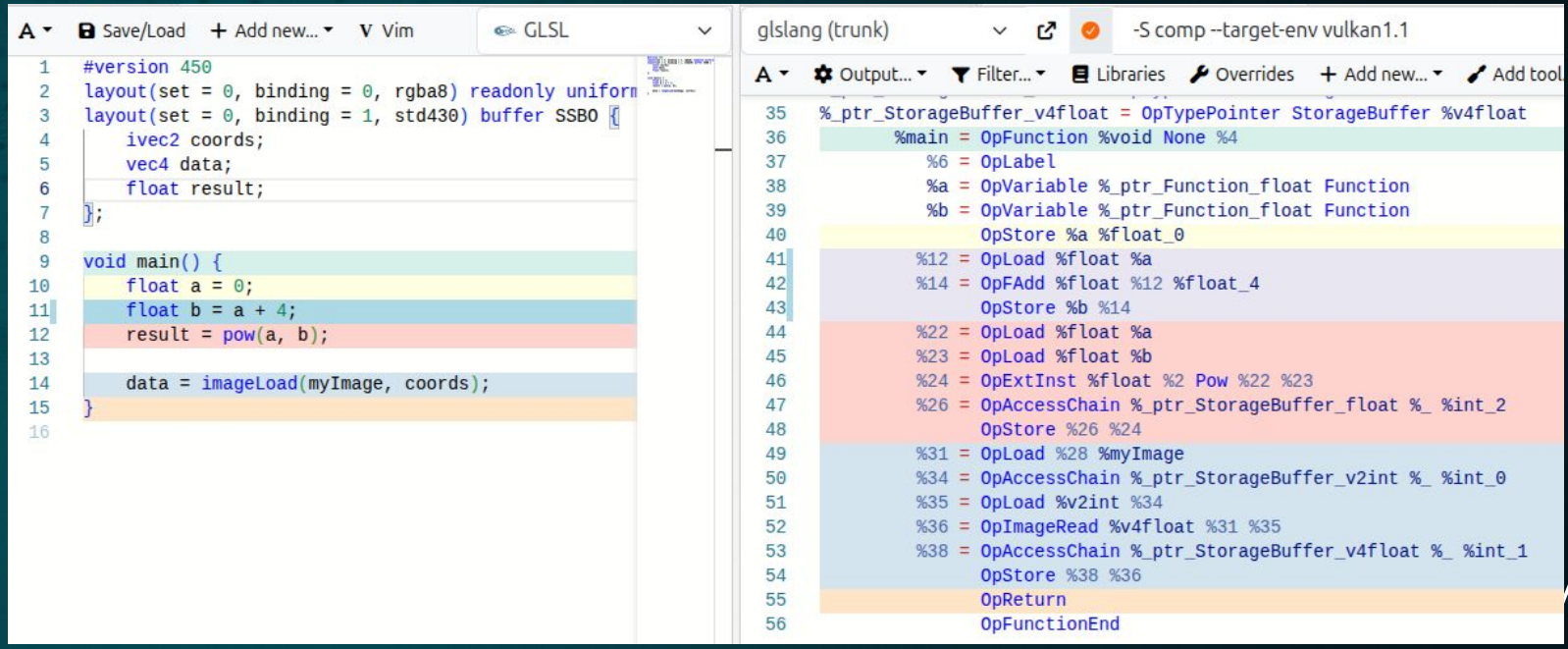
SPIRV-Cross

- Amazing tool to map SPIR-V back to GLSL/HLSL/MSL
  - Thanks Hans Kristian!
- Great way to grasp what the SPIR-V is trying to do

```
# GLSL
./spirv-cross -V example.spv
# HLSL
./spirv-cross --hlsl example.spv
# MSL
./spirv-cross --msl example.spv
```

Compiler Explorer (godbolt)

- Has SPIR-V as an input
- Can see how GLSL/HLSL/Slang generate SPIR-V



Validating your SPIR-V
(3 Levels)

LUNARG

Level 1 – “standalone” validation

- spirv-val
  - Found in SPIRV-Tools
- Detects invalid/ill-formed SPIR-V
  - Using a float instead of an int
  - SSA is incorrect

Level 1 – “standalone” validation

- spirv-val
  - Found in SPIRV-Tools
- Detects invalid/ill-formed SPIR-V
  - Using a float instead of an int
  - SSA is incorrect

Note: Validation Layers already calls spirv-val for you!

Level 2 – “runtime” validaiton

- Vulkan-Validation Layers
- Detects on the CPU still
  - Features are supported
  - Descriptors match the shader

Capabilities

- Think “feature” bit in Vulkan
- Way to declare requirement for the SPIR-V

Capabilities – Example

New instruction: OpTraceRayKHR

Capabilities – Example

New instruction: OpTraceRayKHR

SPIR-V Capability: OpCapability RayTracingKHR

Now the SPIR-V is valid, but can your GPU support it?

Capabilities – Example

New instruction: OpTraceRayKHR

SPIR-V Capability: OpCapability RayTracingKHR

Vulkan feature:

```
RayTracingKHR
        VkPhysicalDeviceRayTracingPipelineFeaturesKHR::rayTracingPipeline
```


LUNARG

Level 3 – GPU Runtime Validation

- Requires GPU Assisted Validation (GPU-AV)
- Validate SPIR-V when executed on the GPU
- See my Shading Languages Symposium later this week for more details!

Number 1 mistake seen from people using validation:

Number 1 mistake seen from people using validation:

VERSIONS

```
--target-env                    {vulkan1.0|vulkan1.1spv1.4|vulkan1.1|vulkan1.2|vulkan1.3|vulkan1.4
                                 |spv1.0|spv1.1|spv1.2|spv1.3|spv1.4|spv1.5|spv1.6|opencl1.2embedded
                                 |opencl1.2|opencl2.0embedded|opencl2.0|opencl2.1embedded|opencl2.1
                                 |opencl2.2embedded|opencl2.2|opengl4.0|opengl4.1|opengl4.2|opengl4.3
                                 |opengl4.5}
```

```
--target-env                    {vulkan1.0|vulkan1.1spv1.4|vulkan1.1|vulkan1.2|vulkan1.3|vulkan1.4
                                 |spv1.0|spv1.1|spv1.2|spv1.3|spv1.4|spv1.5|spv1.6|opencl1.2embedded
                                 |opencl1.2|opencl2.0embedded|opencl2.0|opencl2.1embedded|opencl2.1
                                 |opencl2.2embedded|opencl2.2|opengl4.0|opengl4.1|opengl4.2|opengl4.3
                                 |opengl4.5}
```

LUNARG

Versions

- SPIR-V 1.0 – Vulkan 1.0
- SPIR-V 1.1
- SPIR-V 1.2
- SPIR-V 1.3 – Vulkan 1.1
- SPIR-V 1.4
- SPIR-V 1.5 – Vulkan 1.2
- SPIR-V 1.6 – Vulkan 1.3+

- In Vulkan 1.1, only Input/Output interface variables allowed
- In Vulkan 1.2, all variables are required in the interface

--target-env vulkan 1.1 ❌
--target-env vulkan 1.2 ✅

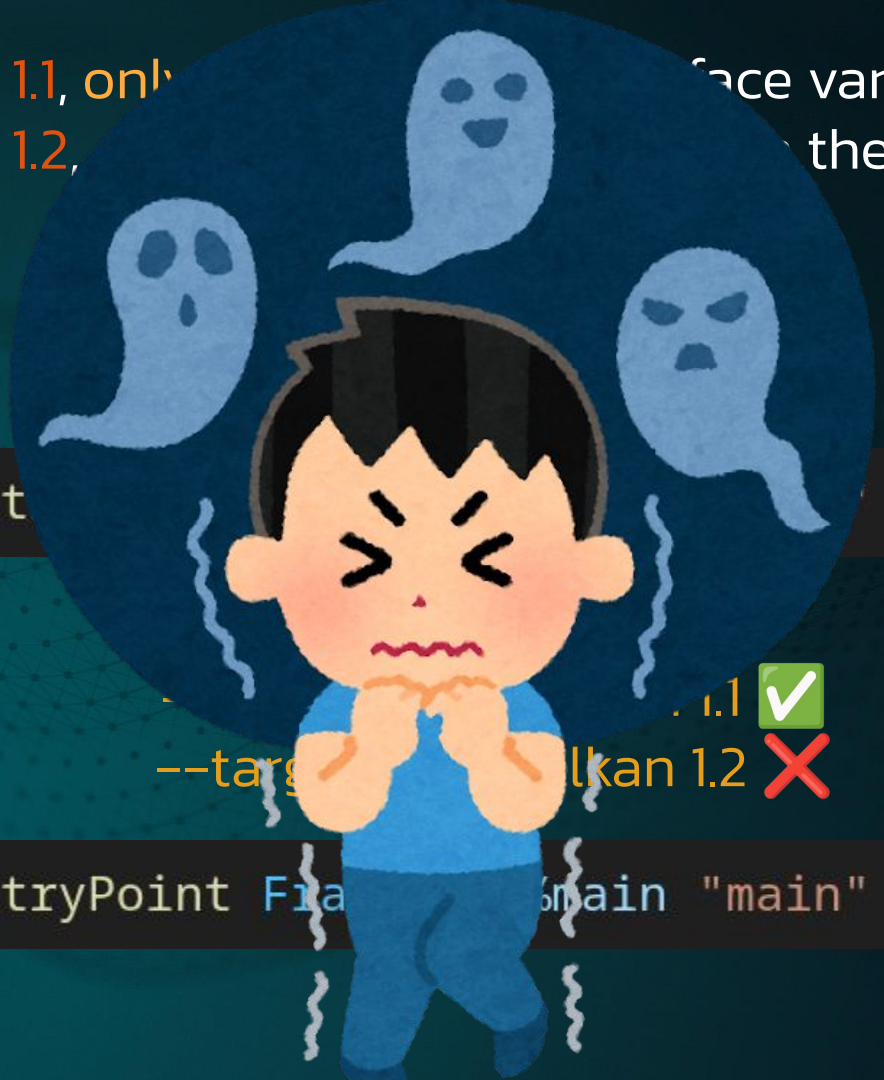
```
OpEntryPoint Fragment %main "main" %a %b
```

--target-env vulkan 1.1 ✅
--target-env vulkan 1.2 ❌

```
OpEntryPoint Fragment %main "main" %a
```


LUNARG

- In Vulkan 1.1, onl… …face variables allowed
- In Vulkan 1.2, … the interface



```
OpEnt …  %a %b
```

…1.1 ✅

--targ… …lkan 1.2 ❌

```
OpEntryPoint Fra… …main "main" %a
```


LUNARG

What you need to remember

Make sure to provide the correct version when generating your SPIR-V from your shading language!

(Default for glslang is version 1.0)

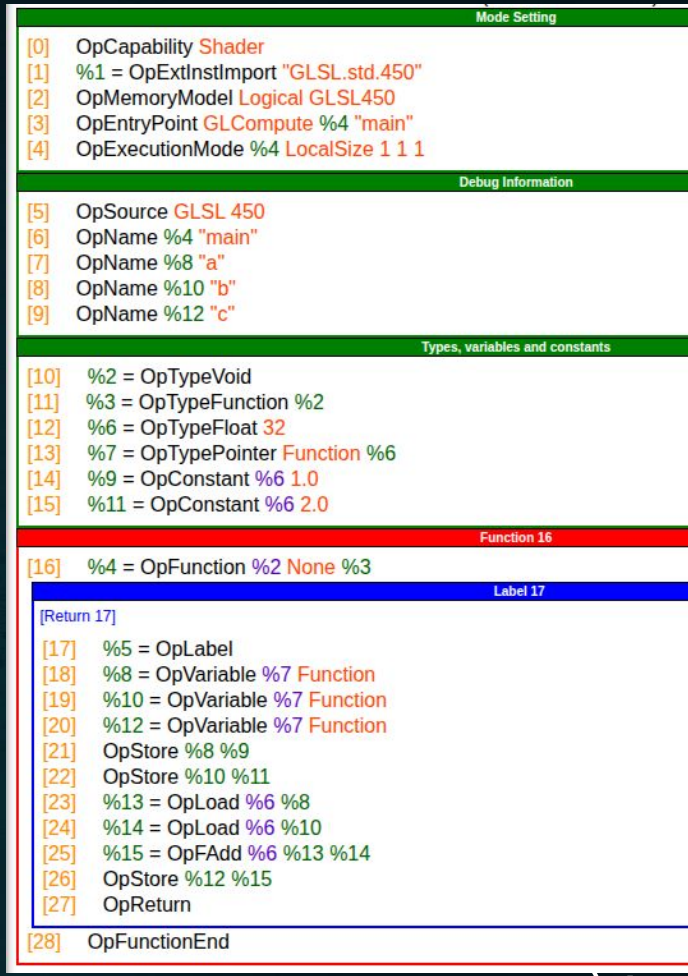
How to detect which version you have

```
fricke@pop-os:~$ spirv-dis triangle.vert.spv
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 7
; Bound: 44
; Schema: 0
               OpCapability Shader
```

“When I look at SPIR-V disassembly, I get lost…”

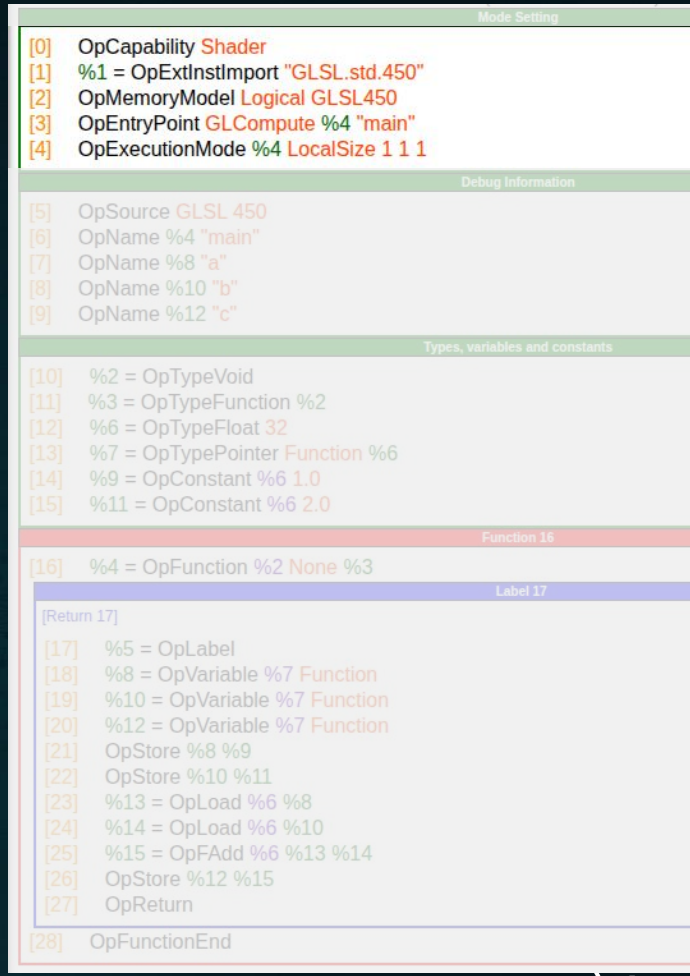
(Vulkan dev)

LUNARG


Mode Setting
[0] OpCapability Shader
[1] %1 = OpExtInstImport "GLSL.std.450"
[2] OpMemoryModel Logical GLSL450
[3] OpEntryPoint GLCompute %4 "main"
[4] OpExecutionMode %4 LocalSize 1 1 1
Debug Information
[5] OpSource GLSL 450
[6] OpName %4 "main"
[7] OpName %8 "a"
[8] OpName %10 "b"
[9] OpName %12 "c"
Types, variables and constants
[10] %2 = OpTypeVoid
[11] %3 = OpTypeFunction %2
[12] %6 = OpTypeFloat 32
[13] %7 = OpTypePointer Function %6
[14] %9 = OpConstant %6 1.0
[15] %11 = OpConstant %6 2.0
Function 16
[16] %4 = OpFunction %2 None %3
Label 17
[Return 17]
[17] %5 = OpLabel
[18] %8 = OpVariable %7 Function
[19] %10 = OpVariable %7 Function
[20] %12 = OpVariable %7 Function
[21] OpStore %8 %9
[22] OpStore %10 %11
[23] %13 = OpLoad %6 %8
[24] %14 = OpLoad %6 %10
[25] %15 = OpFAdd %6 %13 %14
[26] OpStore %12 %15
[27] OpReturn
[28] OpFunctionEnd

Settings / Metadata

High level details about the module



LUNARG

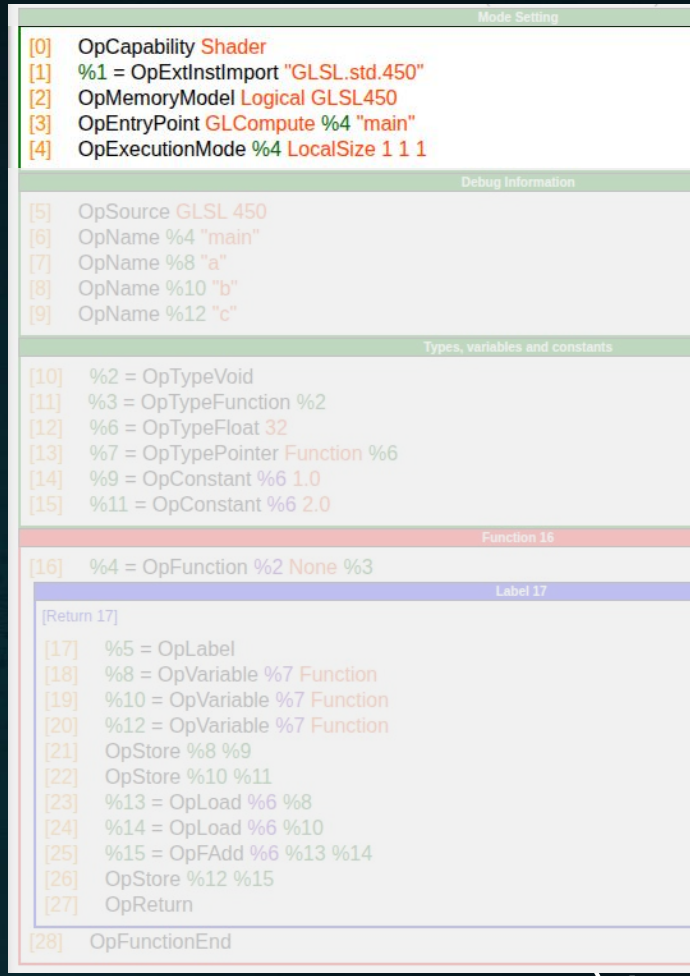
Settings / Metadata

High level details about the module

Name mapped in Vulkan

```
OpEntryPoint GLCompute %4 "main"
```

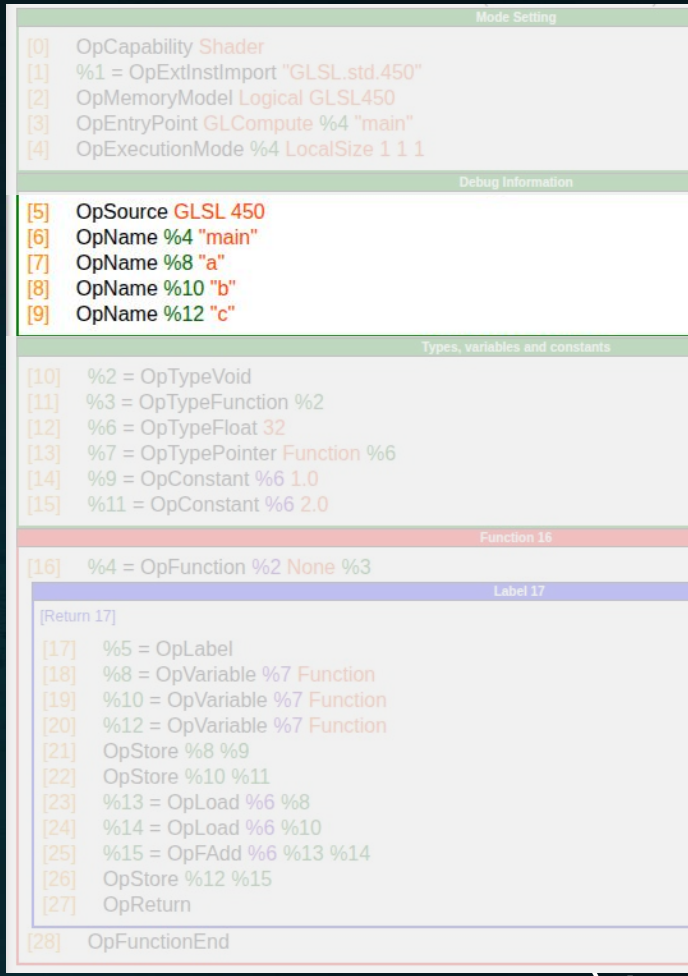
ID of function to start



LUNARG

Debug Information

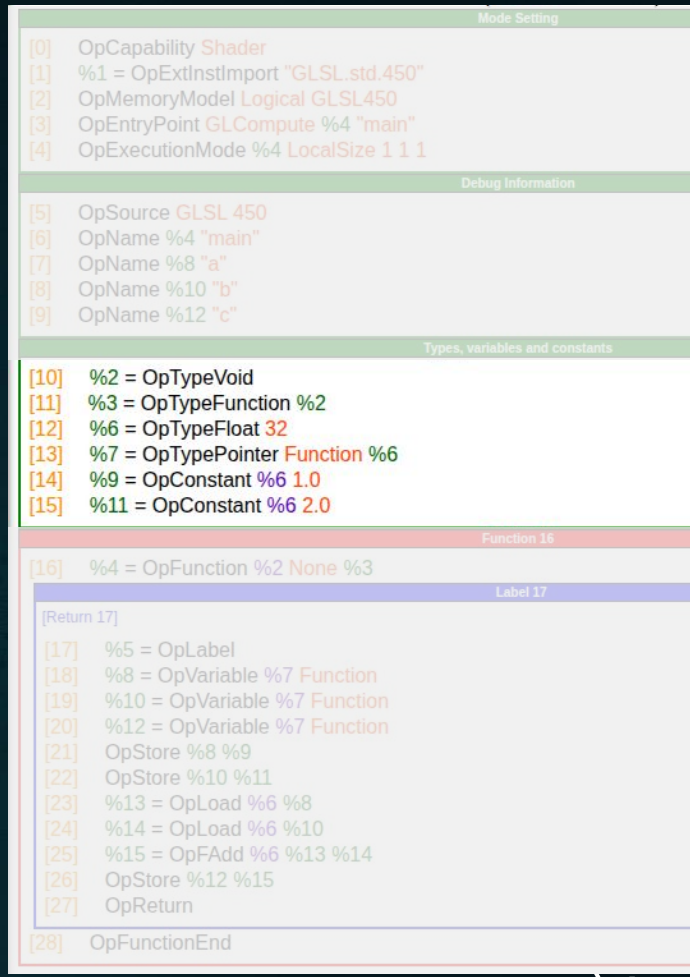
Optional, useful for debugging



LUNARG

Types, Constants, and Global Variables

Things that only need to be defined once



LUNARG

Functions and Blocks

The rest of the module is the code that is “executed”

```
Mode Setting
[0]  OpCapability Shader
[1]  %1 = OpExtInstImport "GLSL.std.450"
[2]  OpMemoryModel Logical GLSL450
[3]  OpEntryPoint GLCompute %4 "main"
[4]  OpExecutionMode %4 LocalSize 1 1 1
Debug Information
[5]  OpSource GLSL 450
[6]  OpName %4 "main"
[7]  OpName %8 "a"
[8]  OpName %10 "b"
[9]  OpName %12 "c"
Types, variables and constants
[10]  %2 = OpTypeVoid
[11]  %3 = OpTypeFunction %2
[12]  %6 = OpTypeFloat 32
[13]  %7 = OpTypePointer Function %6
[14]  %9 = OpConstant %6 1.0
[15]  %11 = OpConstant %6 2.0
Function 16
[16]  %4 = OpFunction %2 None %3
    Label 17
    [Return 17]
    [17]  %5 = OpLabel
    [18]  %8 = OpVariable %7 Function
    [19]  %10 = OpVariable %7 Function
    [20]  %12 = OpVariable %7 Function
    [21]  OpStore %8 %9
    [22]  OpStore %10 %11
    [23]  %13 = OpLoad %6 %8
    [24]  %14 = OpLoad %6 %10
    [25]  %15 = OpFAdd %6 %13 %14
    [26]  OpStore %12 %15
    [27]  OpReturn
[28]  OpFunctionEnd
```

Functions and Blocks

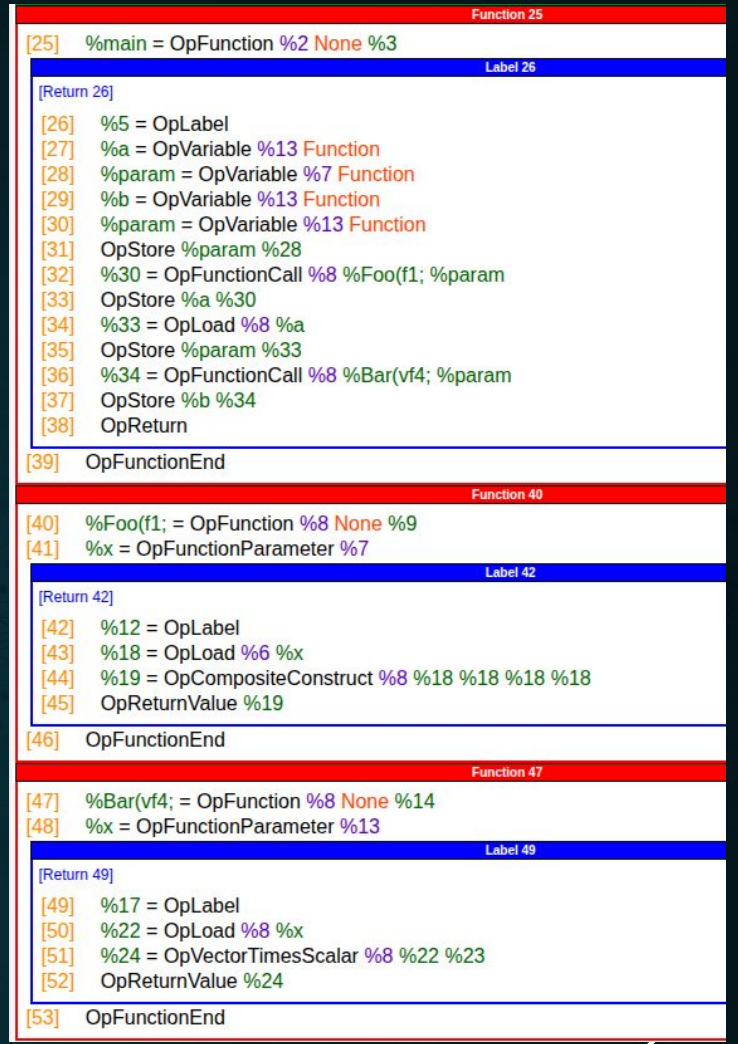
The rest of the module is the code that is “executed”

Wait, what function?

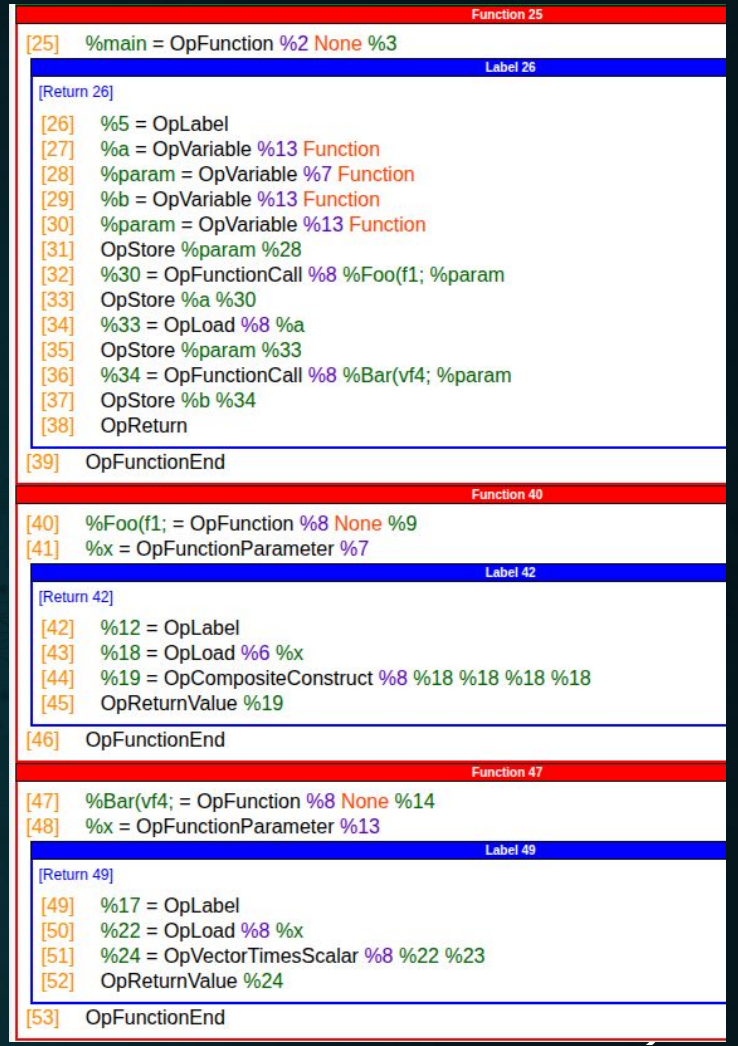
```
Mode Setting
[0]  OpCapability Shader
[1]  %1 = OpExtInstImport "GLSL.std.450"
[2]  OpMemoryModel Logical GLSL450
[3]  OpEntryPoint GLCompute %4 "main"
[4]  OpExecutionMode %4 LocalSize 1 1 1
Debug Information
[5]  OpSource GLSL 450
[6]  OpName %4 "main"
[7]  OpName %8 "a"
[8]  OpName %10 "b"
[9]  OpName %12 "c"
Types, variables and constants
[10]  %2 = OpTypeVoid
[11]  %3 = OpTypeFunction %2
[12]  %6 = OpTypeFloat 32
[13]  %7 = OpTypePointer Function %6
[14]  %9 = OpConstant %6 1.0
[15]  %11 = OpConstant %6 2.0
Function 16
[16]  %4 = OpFunction %2 None %3
  Label 17
  [Return 17]
  [17]  %5 = OpLabel
  [18]  %8 = OpVariable %7 Function
  [19]  %10 = OpVariable %7 Function
  [20]  %12 = OpVariable %7 Function
  [21]  OpStore %8 %9
  [22]  OpStore %10 %11
  [23]  %13 = OpLoad %6 %8
  [24]  %14 = OpLoad %6 %10
  [25]  %15 = OpFAdd %6 %13 %14
  [26]  OpStore %12 %15
  [27]  OpReturn
[28]  OpFunctionEnd
```

```
vec4 Foo(float x) {
    return vec4(x);
}

vec4 Bar(vec4 x) {
    return x * 2;
}

void main() {
    vec4 a = Foo(1.0);
    vec4 b = Bar(a);
}
```

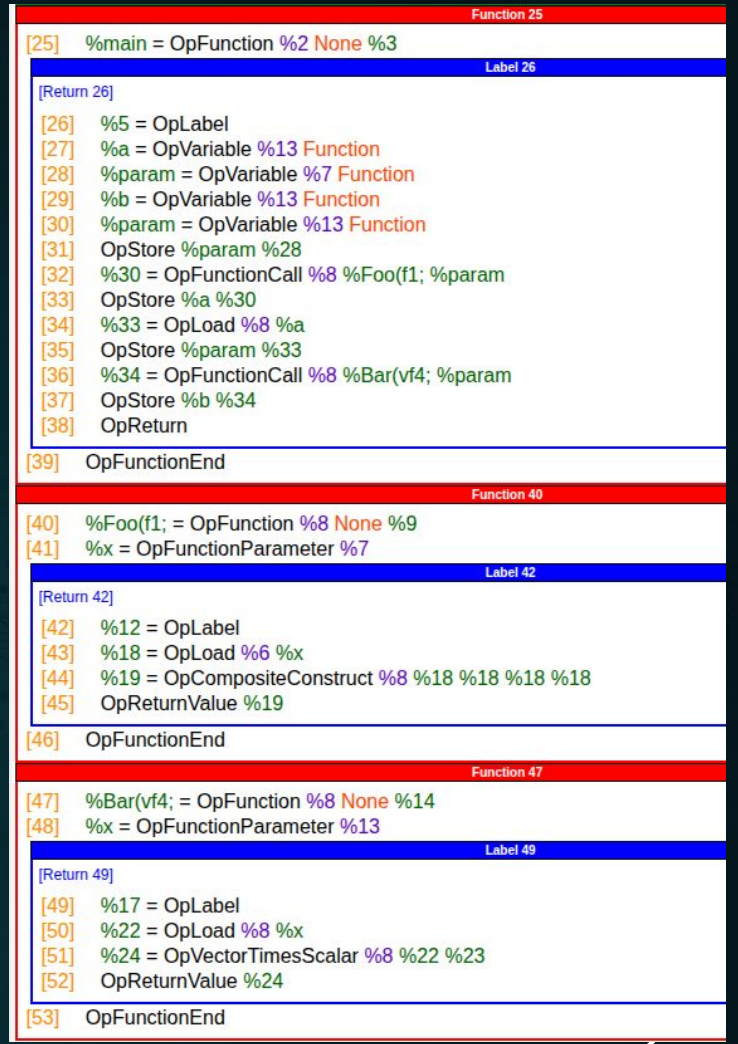
```
vec4 Foo(float x) {
    return vec4(x);
}

vec4 Bar(vec4 x) {
    return x * 2;
}

void main() {
    vec4 a = Foo(1.0);
    vec4 b = Bar(a);
}
```

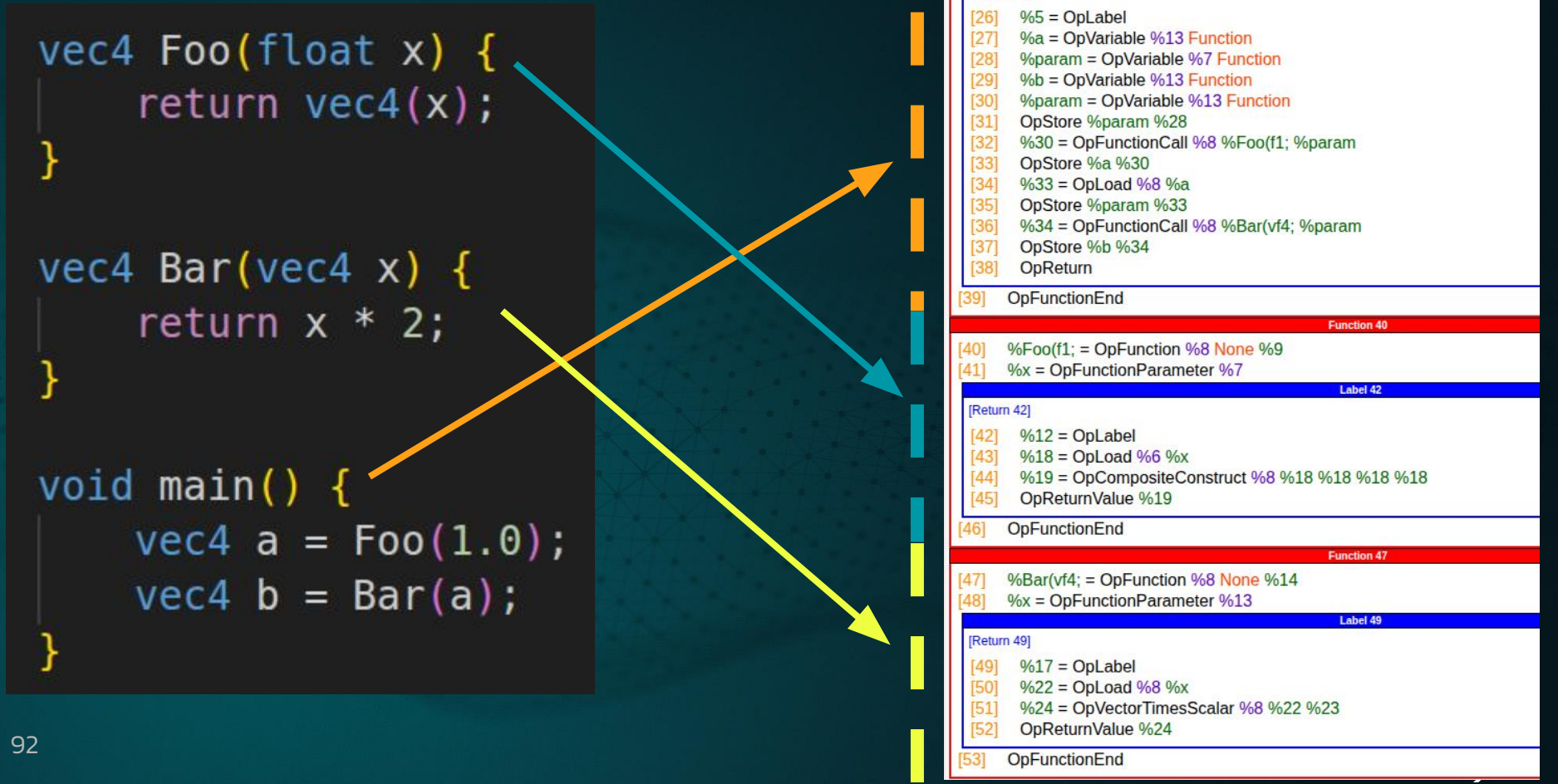


```
vec4 Foo(float x) {
    return vec4(x);
}

vec4 Bar(vec4 x) {
    return x * 2;
}

void main() {
    vec4 a = Foo(1.0);
    vec4 b = Bar(a);
}
```



```
vec4 Foo(float x) {
    return vec4(x);
}

vec4 Bar(vec4 x) {
    return x * 2;
}

void main() {
    vec4 a = Foo(1.0);
    vec4 b = Bar(a);
}
```



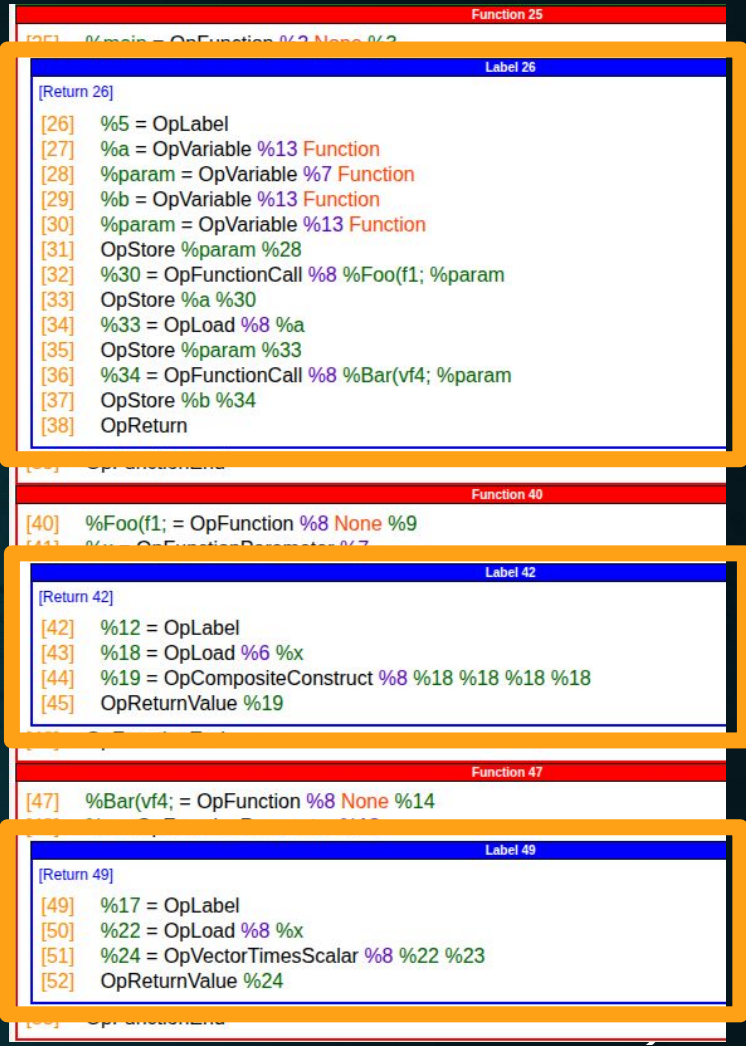

vec4 Foo(float x) {
return vec4(x);
}
vec4 Bar(vec4 x) {
return x * 2;
}
void main() {
vec4 a = Foo(1.0);
vec4 b = Bar(a);
}
Function 25
[25] %main = OpFunction %2 None %3
Label 26
[Return 26]
[26] %5 = OpLabel
[27] %a = OpVariable %13 Function
[28] %param = OpVariable %7 Function
[29] %b = OpVariable %13 Function
[30] %param = OpVariable %13 Function
[31] OpStore %param %28
[32] %30 = OpFunctionCall %8 %Foo(f1; %param
[33] OpStore %a %30
[34] %33 = OpLoad %8 %a
[35] OpStore %param %33
[36] %34 = OpFunctionCall %8 %Bar(vf4; %param
[37] OpStore %b %34
[38] OpReturn
[39] OpFunctionEnd
Function 40
[40] %Foo(f1; = OpFunction %8 None %9
[41] %x = OpFunctionParameter %7
Label 42
[Return 42]
[42] %12 = OpLabel
[43] %18 = OpLoad %6 %x
[44] %19 = OpCompositeConstruct %8 %18 %18 %18 %18
[45] OpReturnValue %19
[46] OpFunctionEnd
Function 47
[47] %Bar(vf4; = OpFunction %8 None %14
[48] %x = OpFunctionParameter %13
Label 49
[Return 49]
[49] %17 = OpLabel
[50] %22 = OpLoad %8 %x
[51] %24 = OpVectorTimesScalar %8 %22 %23
[52] OpReturnValue %24
[53] OpFunctionEnd

Functions always start with **OpFunction**

Functions always start with **OpFunction**

… and end with **OpFunctionEnd**

Every function is made up of “blocks”



Every function is made up of “blocks”

Wait, what is a block?

```
Function 25
Label 26
[Return 26]
[26]  %5 = OpLabel
[27]  %a = OpVariable %13 Function
[28]  %param = OpVariable %7 Function
[29]  %b = OpVariable %13 Function
[30]  %param = OpVariable %13 Function
[31]  OpStore %param %28
[32]  %30 = OpFunctionCall %8 %Foo(f1; %param
[33]  OpStore %a %30
[34]  %33 = OpLoad %8 %a
[35]  OpStore %param %33
[36]  %34 = OpFunctionCall %8 %Bar(vf4; %param
[37]  OpStore %b %34
[38]  OpReturn
Function 40
[40]  %Foo(f1; = OpFunction %8 None %9
Label 42
[Return 42]
[42]  %12 = OpLabel
[43]  %18 = OpLoad %6 %x
[44]  %19 = OpCompositeConstruct %8 %18 %18 %18 %18
[45]  OpReturnValue %19
Function 47
[47]  %Bar(vf4; = OpFunction %8 None %14
Label 49
[Return 49]
[49]  %17 = OpLabel
[50]  %22 = OpLoad %8 %x
[51]  %24 = OpVectorTimesScalar %8 %22 %23
[52]  OpReturnValue %24
```

Control Flow!

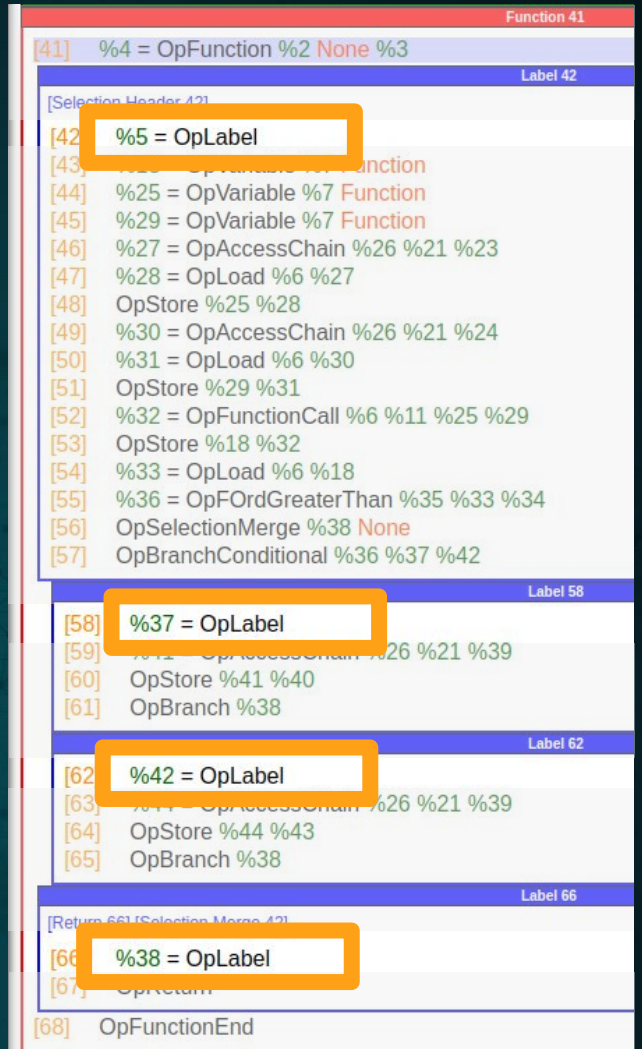
Any code that starts execution in block is guaranteed to reach the end.

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

```
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
```


LUNARG

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



Blocks start with a label

LUNARG

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

```
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
  [56]  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
  [Return 66] [Selection Merge 42]
  [66]  %38 = OpLabel
  [67]  OpReturn
[68]  OpFunctionEnd
```

Declare Control Flow

LUNARG

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

```
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
  [56]  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
```

Which Block the Control Flow merges

LUNARG

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

```
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
```

Conditional decides where to go next

%36 is a OpTypeBool

LUNARG

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

```
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
```

if %36 == true

if %36 == false

LUNARG

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

```
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
```

Blocks always terminate

Never just “fall through”

LUNARG

“Ok, let me look at other code to understand how they use SPIR-V”

(Vulkan dev)

```
uint32_t result_id = insn.Word(1);
uint32_t width     = insn.Word(2);
```

```
uint32_t result_id = insn.Word(1);
uint32_t width     = insn.Word(2);
```

LUNARG

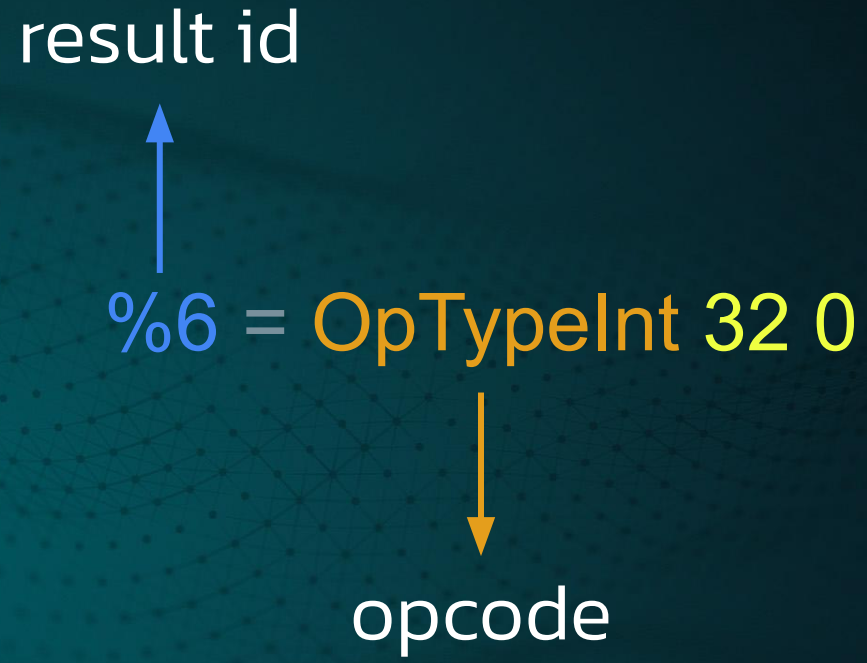
Trust me, it's not actually scary!

```
uint32_t result_id = insn.Word(1);
uint32_t width     = insn.Word(2);
```

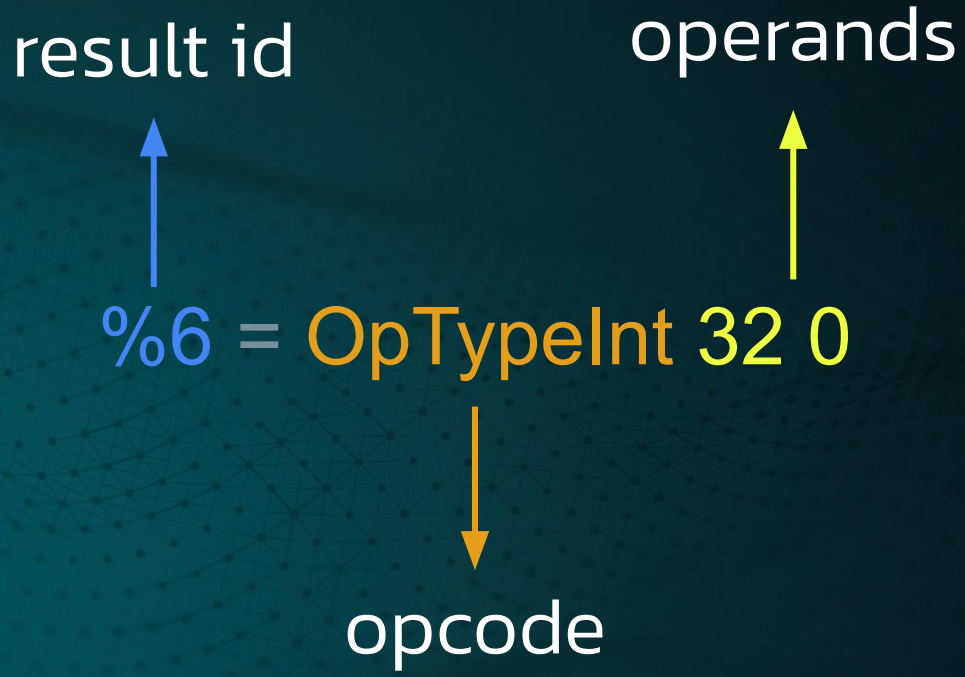
LUNARG

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

LUNARG



LUNARG



LUNARG

https://registry.khronos.org/SPIR-V/specs/unified1/SPIRV.html#OpTypeInt

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*

LUNARG

https://registry.khronos.org/SPIR-V/specs/unified1/SPIRV.html#OpTypeInt

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*

LUNARG

https://registry.khronos.org/SPIR-V/specs/unified1/SPIRV.html#OpTypeInt

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*

Word(0) | Word(1) | Word(2) | Word(3)

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

Instruction is 4 dwords long

(every instruction has a different length)

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal Width*	*Literal Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

21 (0x15) is OpTypeInt opcode

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal Width*	*Literal Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

SSA ID for the Result

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

Width
(32 bit)

Signedness
(unsigned)

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

LUNARG

%6 = OpTypeInt 32 0

4	21	*Result <id>*	*Literal* *Width*	*Literal* *Signedness*

0x00040015 0x00000006 0x00000020 0x00000000

id is just a unique, arbitrary value
(used to reference later in the SSA)

LUNARG

“Ok, I’m ready for the exam”

(Vulkan dev)

LUNARG

Programmatically detect if your SPIR-V does 8-bit, 16-bit, 32-bit, or 64-bit adds

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {


}
```

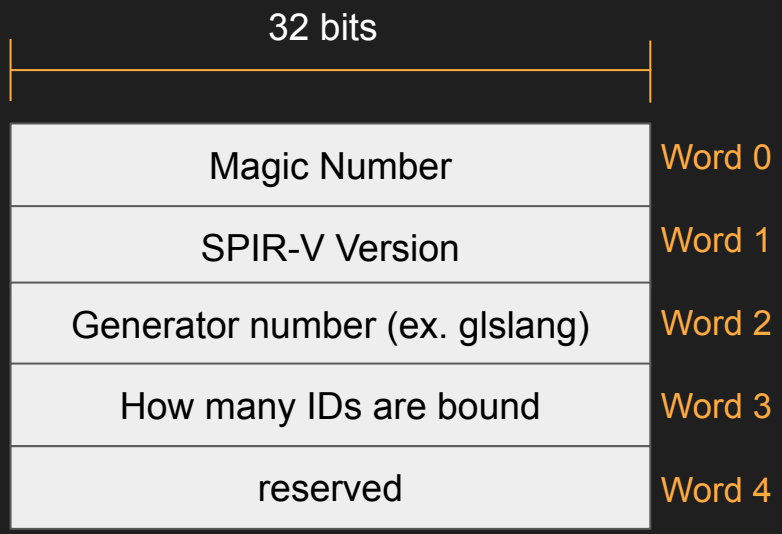
```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

}
```

```
// Provided by VK_VERSION_1_0
typedef struct VkShaderModuleCreateInfo {
    VkStructureType              sType;
    const void*                  pNext;
    VkShaderModuleCreateFlags    flags;
    size_t                       codeSize;
    const uint32_t*              pCode;
} VkShaderModuleCreateInfo;
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header

}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header

}
```



```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header

}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {

    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];

    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;









        offset += length;
    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;



        offset += length;
    }
}
```

These 6 lines of code successfully will loop the entire SPIR-V ... not too scary

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;









        offset += length;
    }
}
```

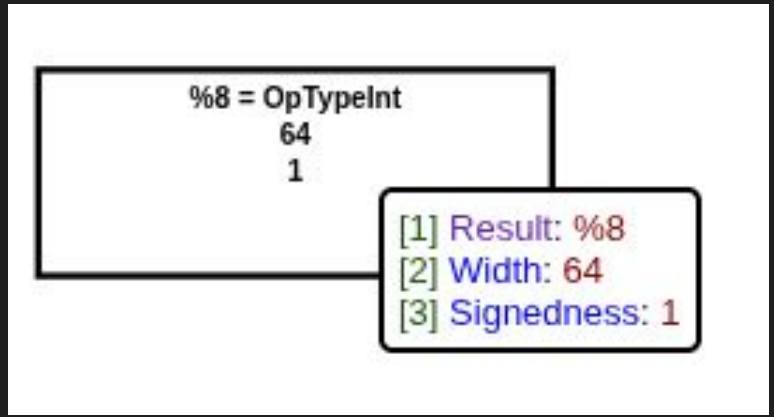
```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {



        }



        offset += length;
    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1];
            uint32_t width     = pCode[offset + 2];


        }




        offset += length;
    }
}
```

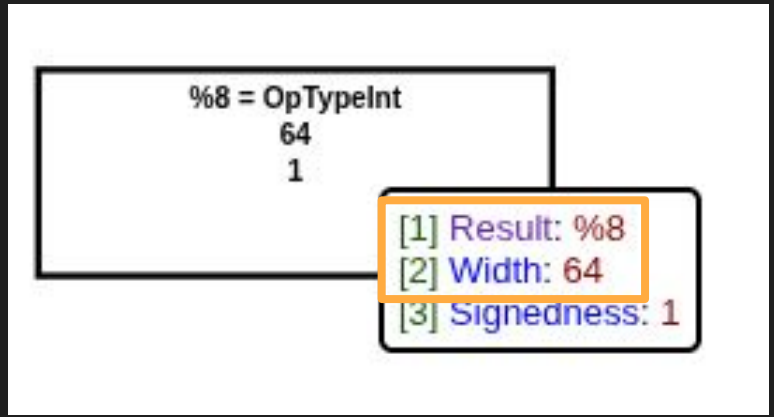
```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1];
            uint32_t width     = pCode[offset + 2];

        }

        offset += length;
    }
}
```

?

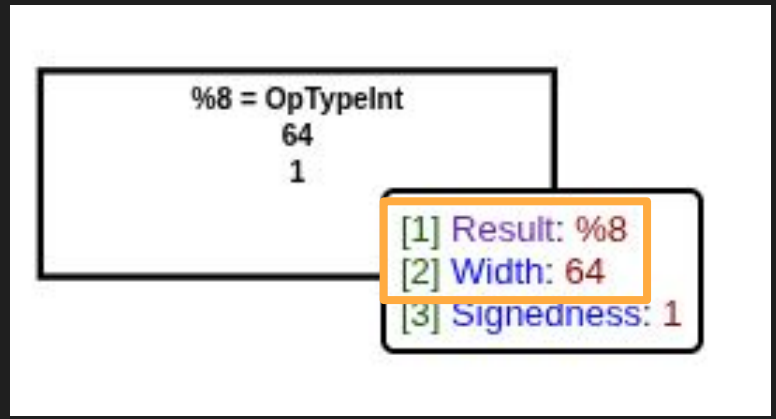
```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1];
            uint32_t width     = pCode[offset + 2];

        }

        offset += length;
    }
}
```



```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1];
            uint32_t width     = pCode[offset + 2];

        }

        offset += length;
    }
}
```



```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64

        }

        offset += length;
    }
}
```



```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {

    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64

        }




        offset += length;
    }
}
```

```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64

        }



        offset += length;
    }
}
```

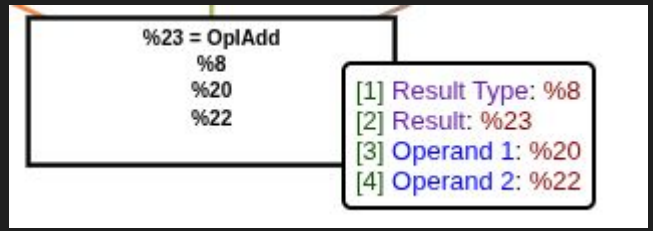
```
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        }



        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {



        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];



        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];       // type_id = 8



        }

        offset += length;
    }
}
```



```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];       // type_id = 8



        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];       // type_id = 8
            uint32_t width   = int_width_map[type_id];

        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];       // type_id = 8
            uint32_t width   = int_width_map[type_id];  // width   = 64

        }

        offset += length;
    }
}
```

```cpp
void ParseModule(uint32_t* pCode, uint32_t codeSize) {
    std::map<uint32_t, uint32_t> int_width_map; // <type_id, bit width>
    uint32_t offset = 5; // skip header
    while (offset < codeSize) {
        uint32_t instruction = pCode[offset];
        uint32_t length = instruction >> 16;
        uint32_t opcode = instruction & 0xffff;

        if (opcode == spv::OpTypeInt) {
            uint32_t result_id = pCode[offset + 1]; // result_id = 8
            uint32_t width     = pCode[offset + 2]; // width     = 64
            int_width_map[result_id] = width;
        } else if (opcode == spv::OpIAdd) {
            uint32_t type_id = pCode[offset + 1];       // type_id = 8
            uint32_t width   = int_width_map[type_id];  // width   = 64
            WidthUsed(width); // exam answer
        }

        offset += length;
    }
}
```

30 minutes ago

now

LUNARG

Final ask

- Find me after in the hallway and tell me you experiences using SPIR-V
  - Was it painful? Why?
  - Tools you wish was out there?
- How could we make SPIR-V less scary?
- Feel free to message me on the Khronos Discord/Slack as well!

Bonus speed round!

~~Bonus speed round!~~

Things I think are important to know, but didn't fit in the slides anywhere!

~~Bonus speed round!~~

Things I think are important to know, but didn't fit in the slides anywhere!

LUNARG

SPIR-V Grammar

- JSON file found in the SPIR-V Headers
  - https://github.com/KhronosGroup/SPIRV-Headers/blob/main/include/spirv/unified1/spirv.core.grammar.json
- Spec and instructions “source of truth”
- Equivalent to Vulkan’s vk.xml
- No reason to look, but good to be aware

Client API	SPIR-V “term”
OpenCL	Kernel
Vulkan	Shader

Execution Model and Execution Mode

- Model = shader stage
  - Vertex, Fragment, GLCompute, etc
- Mode = how an entry point executes
  - OriginUpperLeft vs OriginLowerLeft
  - LocalSize

Types

- Are actually simple and easy!
- Use one type to build another type
- Example: mat3x2

```
%float = OpTypeFloat 32
%v2float = OpTypeVector %float 2
%mat3v2 = OpTypeMatrix %v2float 3
```

Note for future reference

- LLVM uses unstructured control flow
  - Can represent goto
- SPIR-V uses structured control flow
  - Must know where every branch ends
- Not saying one is “correct” but the difference is a common source of pain going between IR

spirv-dis --raw-id

```
       %void = OpTypeVoid
          %3 = OpTypeFunction %void
      %float = OpTypeFloat 32
    %v3float = OpTypeVector %float 3
%_ptr_Output_v3float = OpTypePointer Output %v3float
   %outColor = OpVariable %_ptr_Output_v3float Output
%_ptr_Input_v3float = OpTypePointer Input %v3float
    %inColor = OpVariable %_ptr_Input_v3float Input
    %v4float = OpTypeVector %float 4
%gl_PerVertex = OpTypeStruct %v4float
%_ptr_Output_gl_PerVertex = OpTypePointer Output %gl_PerVertex
          %_ = OpVariable %_ptr_Output_gl_PerVertex Output
        %int = OpTypeInt 32 1
      %int_0 = OpConstant %int 0
%mat4v4float = OpTypeMatrix %v4float 4
        %UBO = OpTypeStruct %mat4v4float %mat4v4float %mat4v4fl
%_ptr_Uniform_UBO = OpTypePointer Uniform %UBO
        %ubo = OpVariable %_ptr_Uniform_UBO Uniform
%_ptr_Uniform_mat4v4float = OpTypePointer Uniform %mat4v4float
      %int_2 = OpConstant %int 2
      %int_1 = OpConstant %int 1
      %inPos = OpVariable %_ptr_Input_v3float Input
```

```
   %2 = OpTypeVoid
   %3 = OpTypeFunction %2
   %6 = OpTypeFloat 32
   %7 = OpTypeVector %6 3
   %8 = OpTypePointer Output %7
   %9 = OpVariable %8 Output
  %10 = OpTypePointer Input %7
  %11 = OpVariable %10 Input
  %13 = OpTypeVector %6 4
  %14 = OpTypeStruct %13
  %15 = OpTypePointer Output %14
  %16 = OpVariable %15 Output
  %17 = OpTypeInt 32 1
  %18 = OpConstant %17 0
  %19 = OpTypeMatrix %13 4
  %20 = OpTypeStruct %19 %19 %19
  %21 = OpTypePointer Uniform %20
  %22 = OpVariable %21 Uniform
  %23 = OpTypePointer Uniform %19
  %26 = OpConstant %17 2
  %30 = OpConstant %17 1
  %34 = OpVariable %10 Input
```

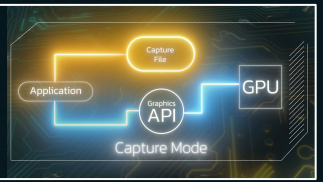
LUNARG
POWER YOUR SUCCESS

Thank you –
KHRONOS GROUP

KosmicKrisp

Come to the LunarG Table!

See KosmicKrisp & GFXReconstruct



Take the 2026 Vulkan Ecosystem Survey!

LunarG Presentations

Vulkanised 2026

LunarG Presentations

Shading Languages Symposium 2026


LUNARG
POWER YOUR SUCCESS