Bringing ARB_gpu_shader_fp64 to Intel GPUs

Iago Toral Quiroga
<itoral@igalia.com>

XDC 2016
Helsinki, Finland

Contents

ARB_gpu_shader_fp64

- **New GLSL types**: double, dvecX and dmatX
  - Variables, Inputs, outputs, uniforms, constants (LF suffix)
  - No 64-bit vertex attributes (handled by another extension)
  - No 64-bit fragment shader outputs (no 64-bit FBs)
- **Arithmetic and relational operators** supported
- **Most built-in functions can take the new types**
  - Some exceptions: angle, trigonometry, exponential, noise
- **New packing functions:** (un)packDouble2x32
- **Conversions from/to 32-bit types**
- **No interpolation**

Scope (i965)

- Combined work by Intel and Igalia for **over a year**
- **~260 new patches** to add **NIR** (~60) and **i965** support (~100 Broadwell+, ~100 Haswell)
  - More IvyBridge patches on the way!
- Usable across **all shader stages**
- **3 IRs** to support: NIR, i965/align1, i965/align16
- **Lots of GL functionality involved:** ALU, varyings, uniforms, UBO, SSBO, shared variables, etc.
- **Lots of internal driver modules involved:** Optimization passes, liveness analysis, register spilling, etc.

Scope (Piglit)

- Piglit coverage was limited
  - Mostly focused on ALU operations.
  - **~2,000 new tests** added

NIR

NIR

- Added support for **bit-sized ALU types**:
  - nir_type_float32 = 32 | nir_type_float, etc
  - nir_alu_get_type_size(), nir_alu_get_base_type()
- We need to be a bit more careful now:

```
- if (alu_info.output_type == nir_type_bool) {
+if (nir_alu_type_get_base_type(alu_info.output_type) == nir_type_bool) {
```

igalia

NIR

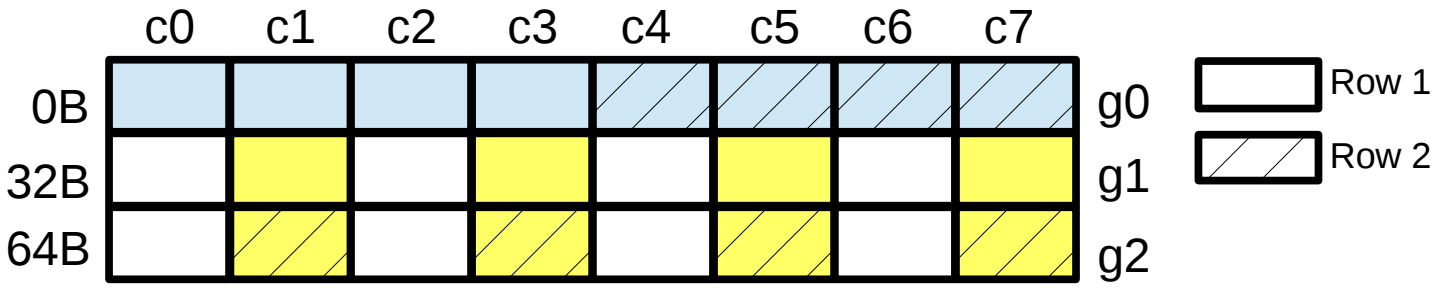
- Lots of plumbing to deal correctly with bit-sizes everywhere.
- Algebraic rules and bit-size validation
- New double-precision opcodes:
  - d2f, d2i, d2u, d2b, f2d, i2d, u2d
  - (un)pack_double_2x32
- etc

NIR

- Added **lowering for unsupported 64-bit operations** on Intel GPU hardware**:**
  - trunc, floor, ceil, fract, round, mod, rcp, sqrt, rsq
- nir_lower_doubles(). Implemented in terms of:
  - Supported 64-bit operations
  - 32-bit float/integer math
- Drivers can choose which lowerings to use:
  - nir_lower_doubles_options

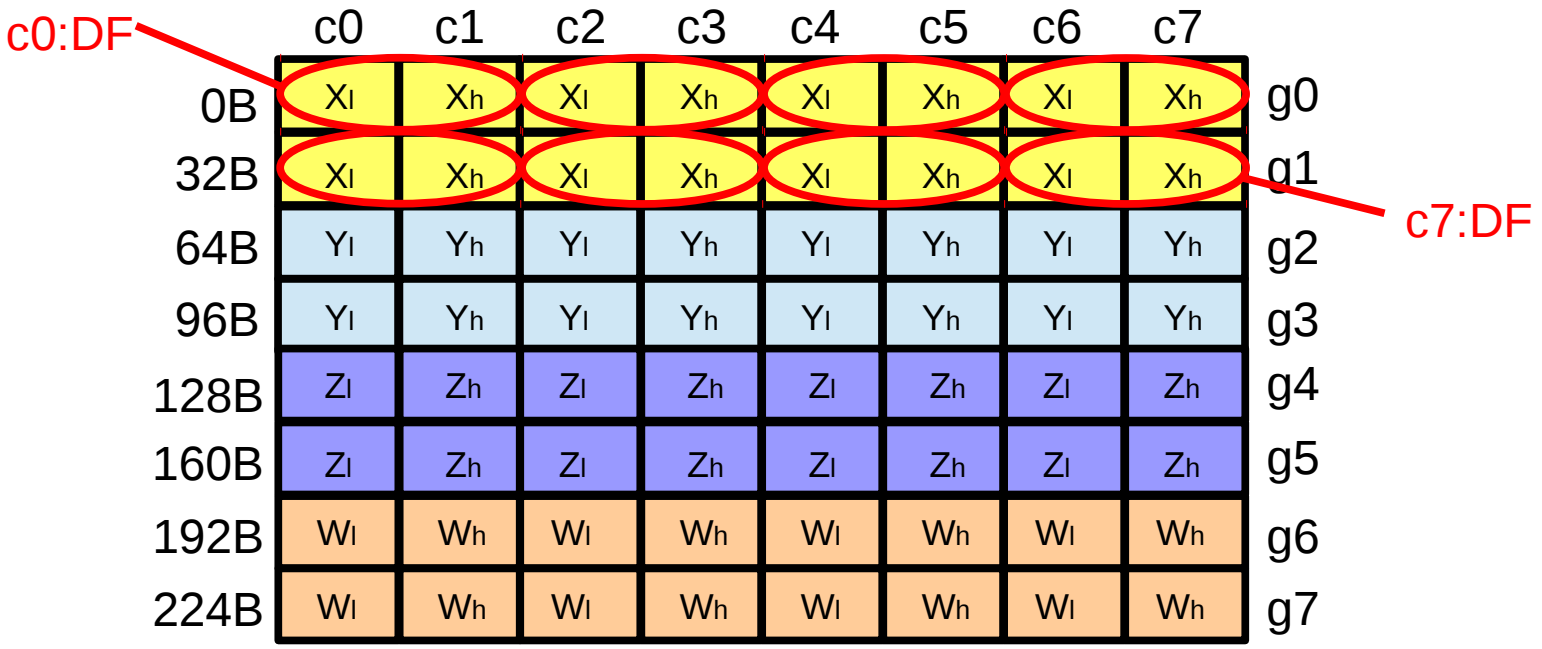
i965 – Fp64 (Align1)

Align1 – SIMD8 - 32bit




```
mov(8)      g0.0<1>F      g1.1<8,4,2>F      { align1 1Q }
```

Align1 – SIMD8 - 64bit

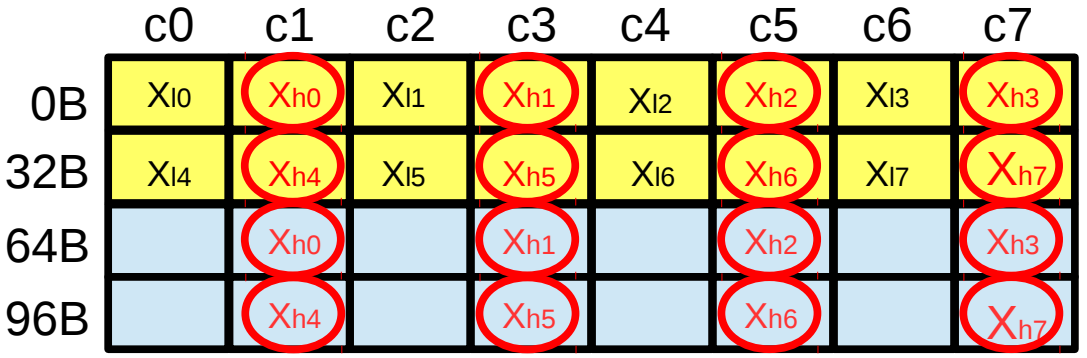


igalia

Align1 – SIMD8 - 64bit (Haswell+)


```
mov(8)        g2<1>DF        g0<4,4,1>DF        { align1 1Q }
```

Align1 – SIMD8 - 64bit (Haswell+)




```
mov(8)    g2.1<2>UD    g0.1<8,4,2>UD    { align1 1Q }
```

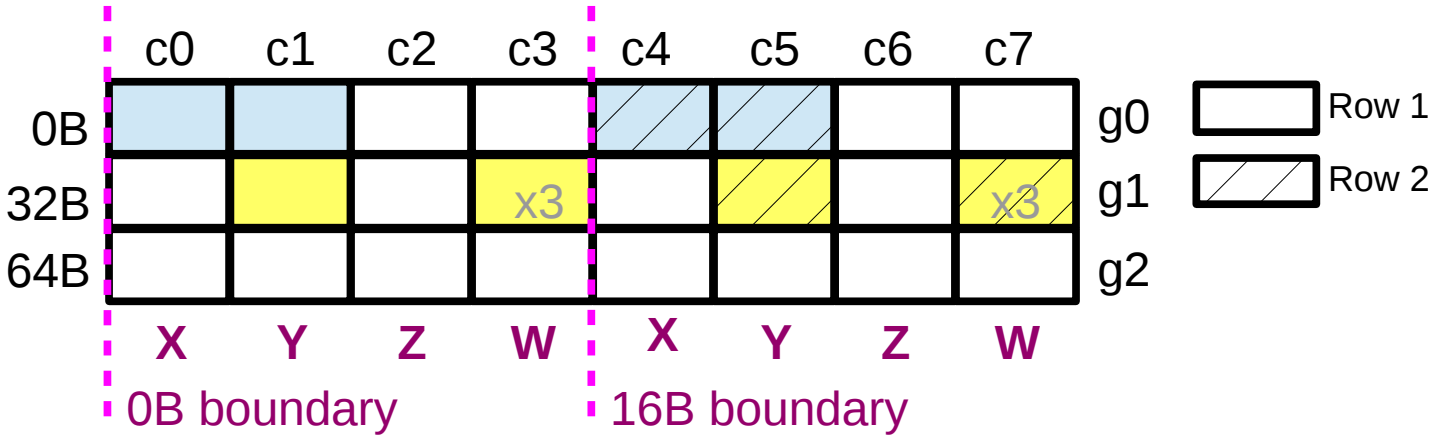
- Use the subscript() helper:

  subscript(reg, BRW_REGISTER_TYPE_UD, 1)

igalia

i965 – Fp64 (Align16)

Align16 – SIMD4x2 - 32bit




```
mov(8)    g0.0<1>.xyF    g1.0<4,4,1>.ywwwF    { align16 1Q }
```

Fixed

Fixed

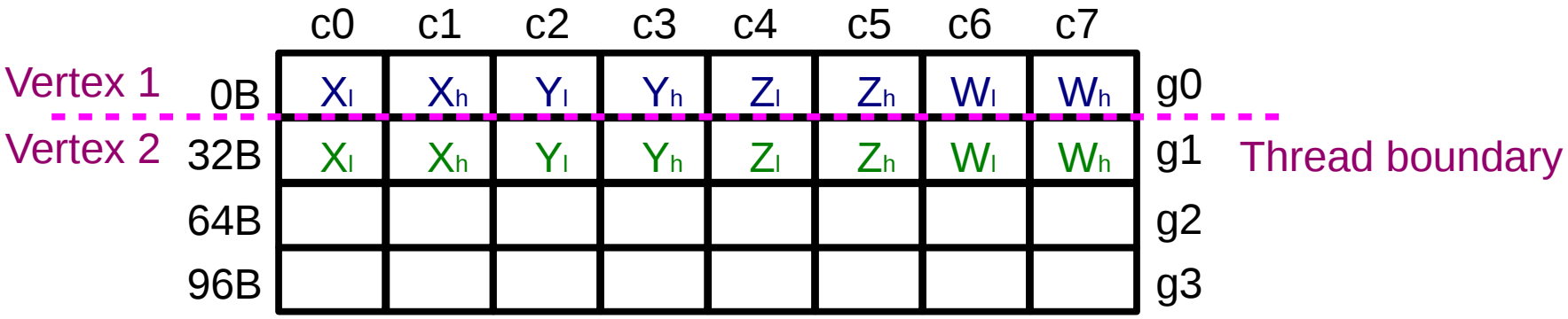
Align16 - 64bit

- Broadwell+ used to need Align16 for **Geometry** and **Tessellation** shaders.
  - Nowadays these platforms are fully scalar and Align1 support is sufficient to expose Fp64
- Older platforms (Haswell, IvyBridge, etc) still need Align16 for **Vertex**, **Geometry** and **Tessellation** shaders.

Align16 - 64bit

- **Swizzle channels are 32-bit**, even on 64-bit operands
  - We can only address DF components XY directly!
- **Writemasks are 64-bit** for DF destinations though
  - WRITEMASK_XY and WRITEMASK_ZW are 32-bit though → no native representation

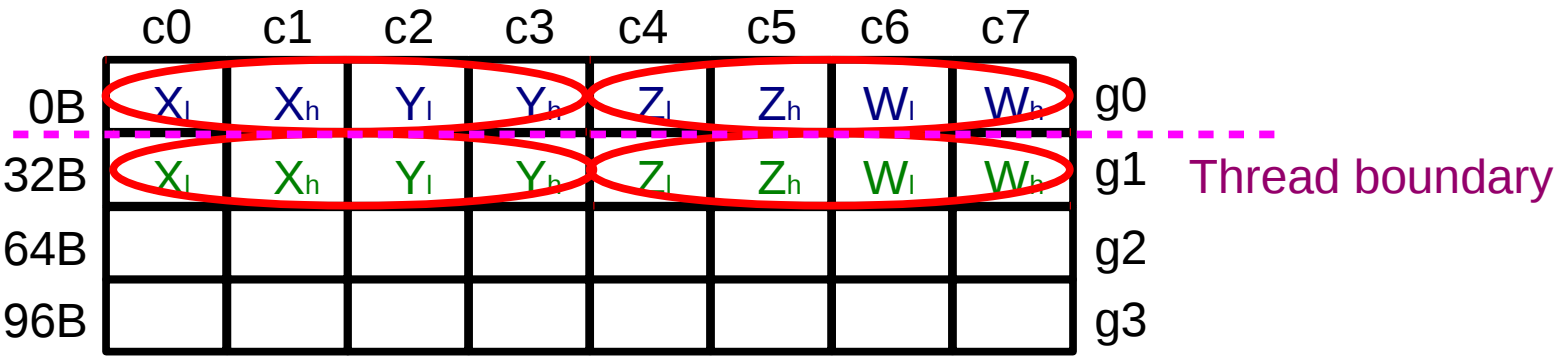
Align16 - SIMD4x2 - 64bit




```
mov(8)    g2<1>.xyDF    g0<2,2,1>.zwzwDF    { align16 1Q }
```

Fixed

Align16 - SIMD4x2 - 64bit

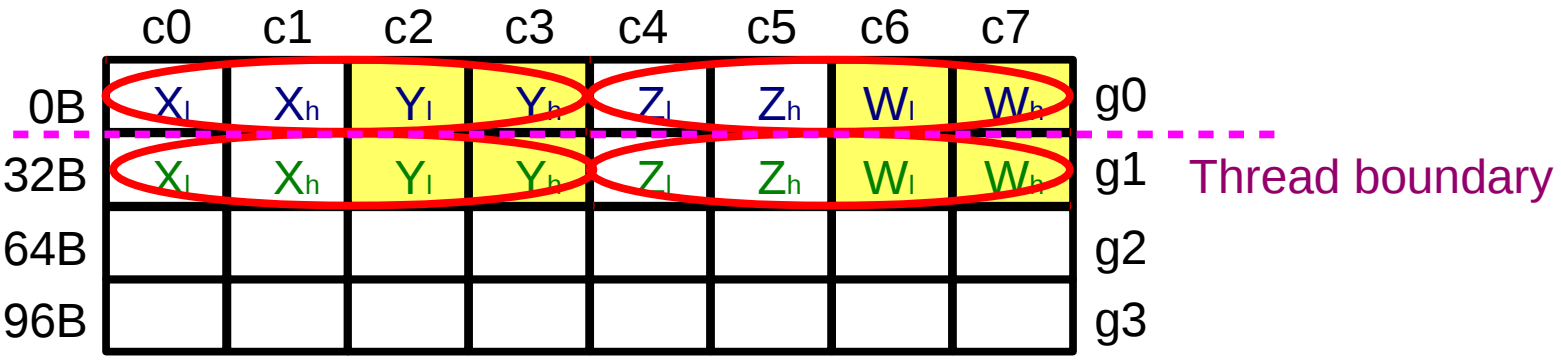



```
mov(8)    g2<1>.xyDF    g0<2,2,1>.zwzwDF    { align16 1Q }
```

- Align16 requires 16B alignment
  - → 2 DF components in each row.
- Vstride=2 to cover the entire region

igalia

Align16 - SIMD4x2 - 64bit

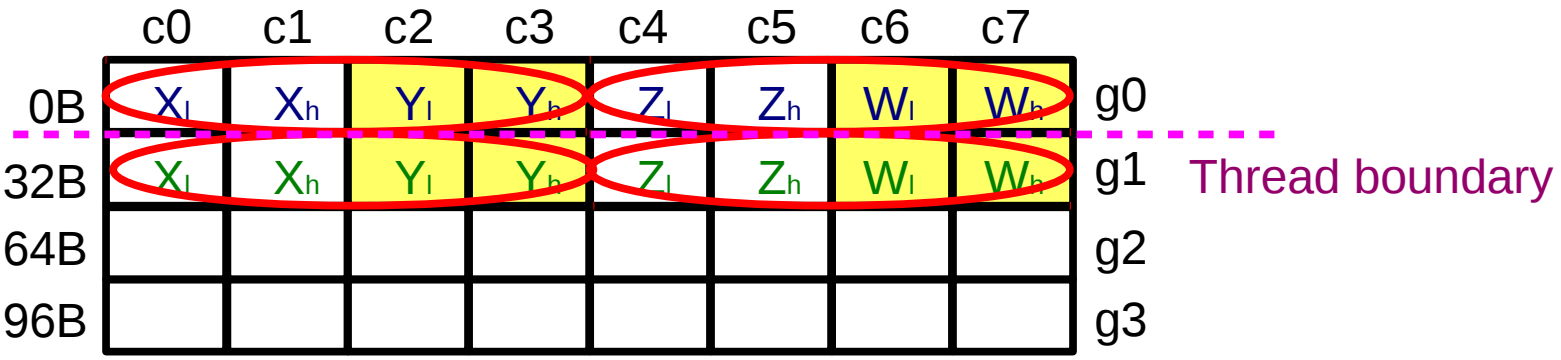



```
mov(8)      g2<1>.xyDF    g0<2,2,1>.zwzwDF      { align16 1Q }
```

- Each 16B region applies the 4-component 32-bit swizzle

igalia

Align16 - SIMD4x2 - 64bit

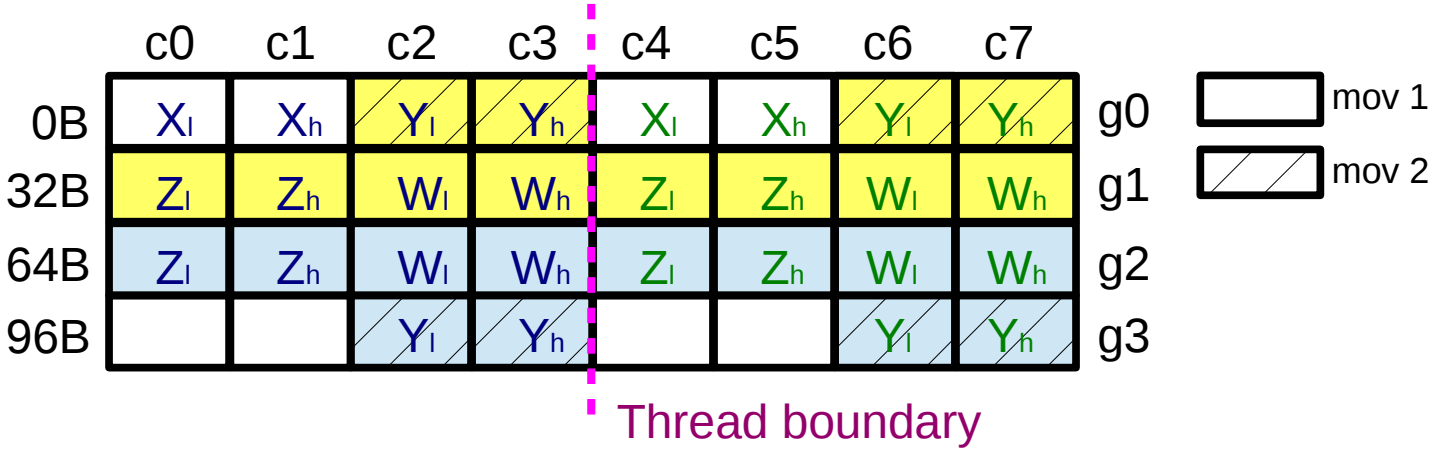



```
mov(8)    g2<1>.xyDF    g0<2,2,1>.zwzwDF    { align16 1Q }
```

- **We can't do all swizzle combinations**:
  - XXXX, YZYZ, XYYZ, etc. are not supported..
- We need to translate our 64-bit swizzles to 32-bit.
  - **X → XY, Y → ZW, Z → ?, W → ?**

igalia

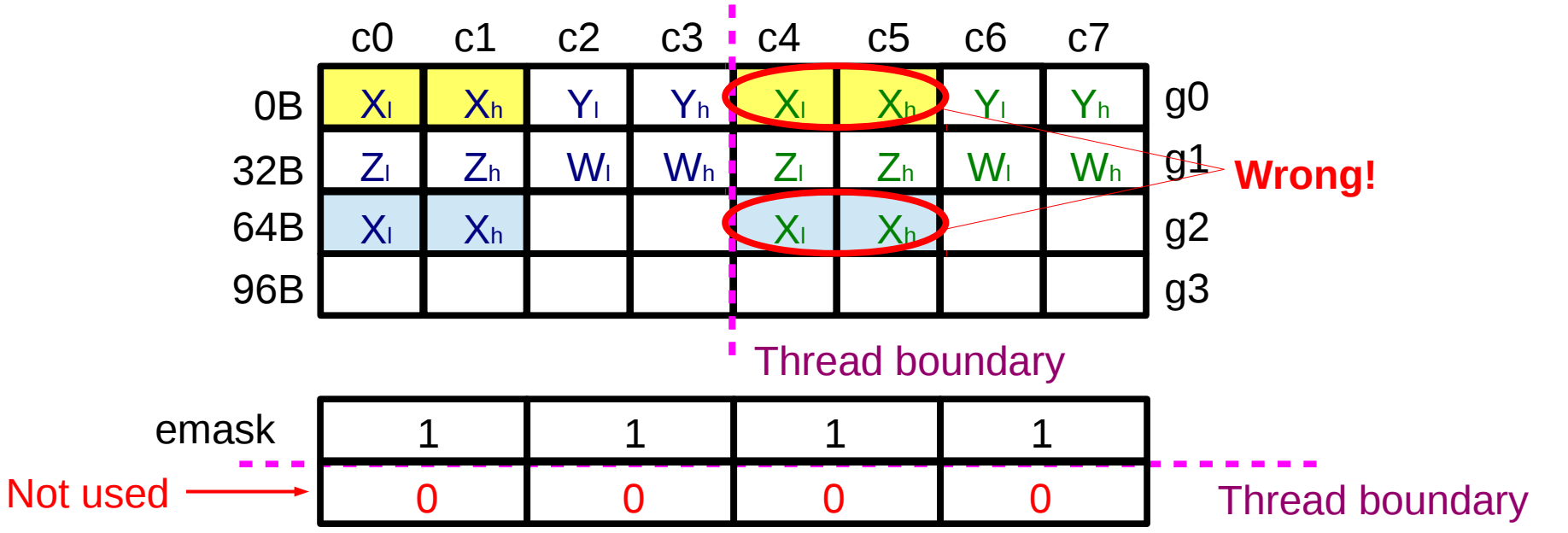
Align16 - SIMD4x2 - 64bit
XY / ZW component splitting




```
mov(8)    g2<1>.xywDF  g0<2,2,1>.zwyDF
                          ↓
mov(4)    g2<1>.xyDF   g0+1<2,2,1>.xyzwDF
mov(4)    g2+1<1>.yDF  g0<2,2,1>.zwzwDF
```

igalia

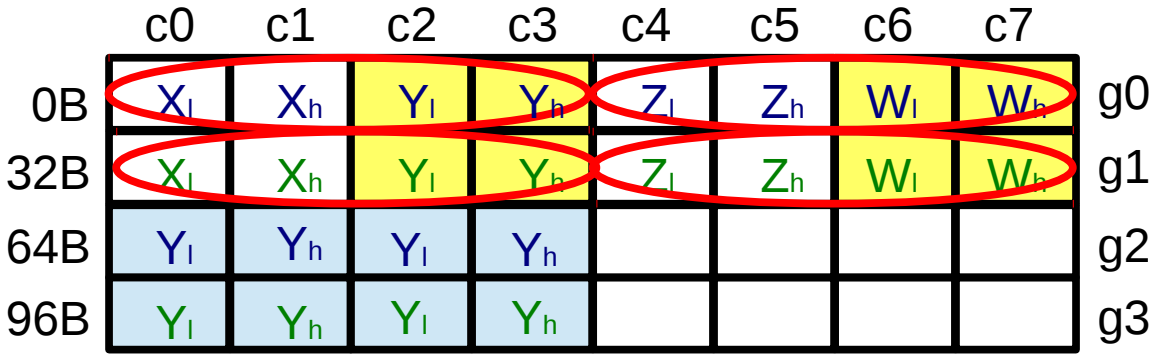
Align16 - SIMD4x2 - 64bit
XY / ZW component splitting




```
mov(4)    g2<1>.xDF    g0<2,2,1>.xyxyDF    { align16 1Q }
```

igalia

Align16 - SIMD4x2 - 64bit

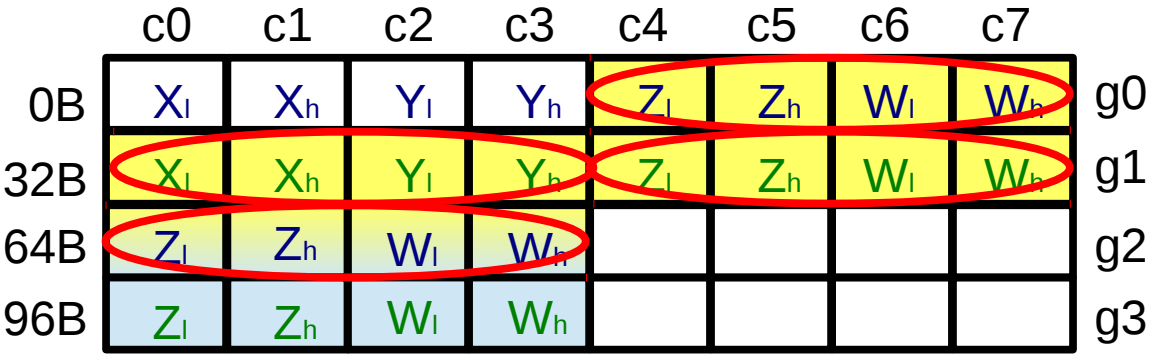



```
mov(8)    g2<1>.xyDF    g0<2,2,1>.zwzwDF    { align16 1Q }
```

- Back to square one…
  - Z → XY, W → ZW (at 16B offset)

igalia

Align16 – SIMD4x2 - 64bit



mov(8) g2<1>.xyDF g0.2<2,2,1>.xyzwDF { align16 1Q }

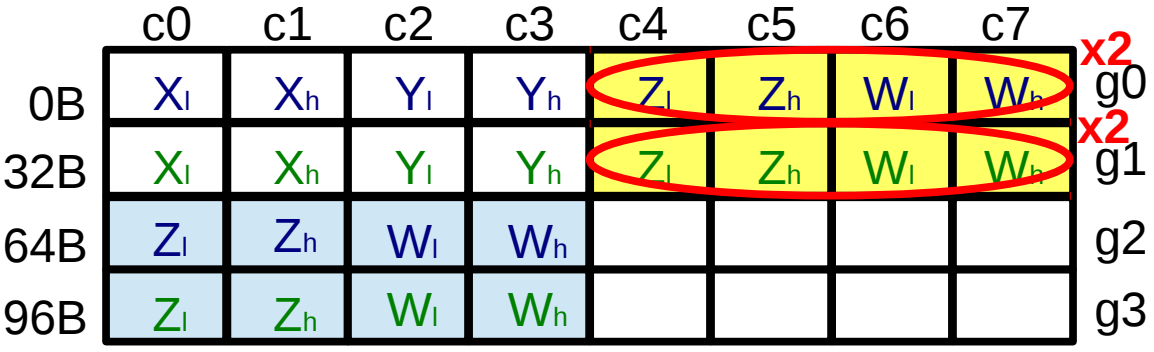
- Not good enough (and violates register region restrictions)
  - We could use a combination of **vstride=0** and **SIMD splitting**.

igalia

Align16 - SIMD4x2 - 64bit

- Gen7 hardware seems to have an interesting ~~bug~~ feature:
  - The second half of a compressed instruction with vstride=0 will ignore the vstride and offset exactly 1 register
  - We can use this to avoid the SIMD splitting

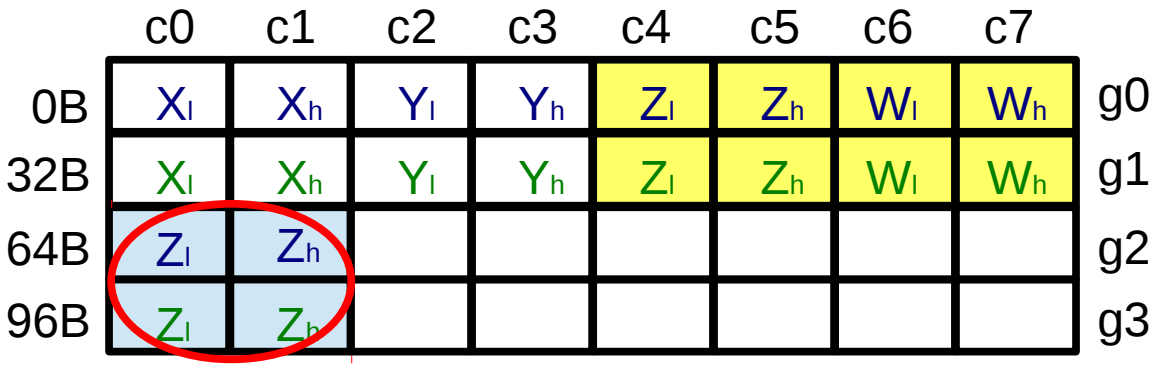
Align16 - SIMD4x2 - 64bit




```
mov(8)    g2<1>.xyDF    g0.2<0,2,1>.xyzwDF    { align16 1Q }
```

igalia

Align16 - SIMD4x2 - 64bit

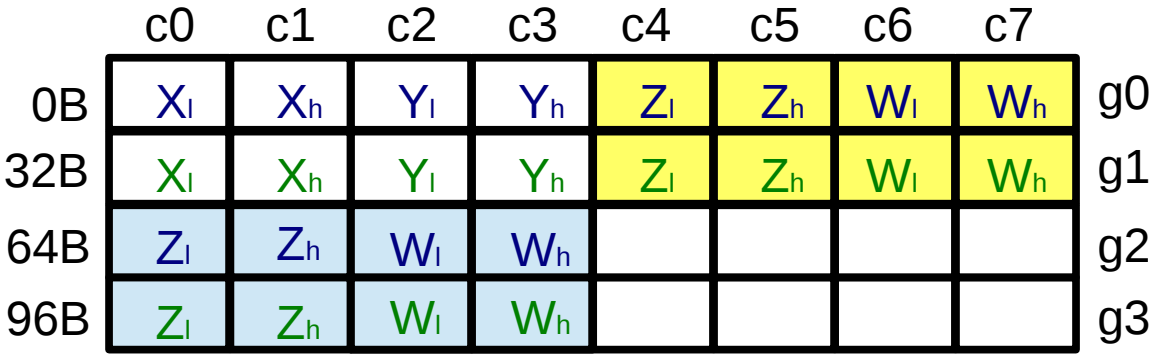



```
mov(8)    g2<1>.xyDF    g0.2<0,2,1>.xyzwDF    { align16 1Q }
```

- Remember that issue with 32-bit writemasks?
  - WRITEMASK_XY == WRITEMASK_X

igalia

Align16 - SIMD4x2 - 64bit




```
mov(8)    g2<1>.xDF    g0.2<0,2,1>.xyzwDF    { align16 1Q }
mov(8)    g2<1>.yDF    g0.2<0,2,1>.xyzwDF    { align16 1Q }
```

igalia

Align16 - SIMD4x2 - 64bit

- This is just an example:
  - Different swizzle combinations may require different implementations
  - 2-src instructions and 3-src instructions

Align16 - SIMD4x2 - 64bit

- Implementation:
  - **Step 1:** scalarize everything, swizzle translation at codegen - Done
  - **Step 2:** let through swizzle classes that we can support natively (e.g. XYZW) - Done
  - **Step 3:** let through swizzle classes that we can support by exploiting the vstride=0 behavior (e.g. XXXX) - Done
  - **Step 4:** use component splitting (partial scalarization) to support more swizzle classes – Not Done (yet)

i965 – Fp64
Common Issues

Multiple hardware generations

- Significant differences between IvyBridge, Haswell and Broadwell+ hardware
- Skylake did not require specific adaptations
  - Broxton, CherryView and Braswell only required minor tweaks:
    - 32b to 64b conversions need 64b aligned source data
    - 64b indirect addressing not supported

32-bit driver

- Before fp64 all GLSL types were implemented as 32-bit types.
  - Driver code assumed 32-bit types (and even hstride=1) in lots of places.
- Manyfixes like:

```
-   int dst_width = inst->exec_size / 8;
+   int dst_width =
        DIV_ROUND_UP(inst->dst.component_size(inst->exec_size), REG_SIZE);
```

- This could happen anywhere in the driver
  - Piglit was the driving force to find these

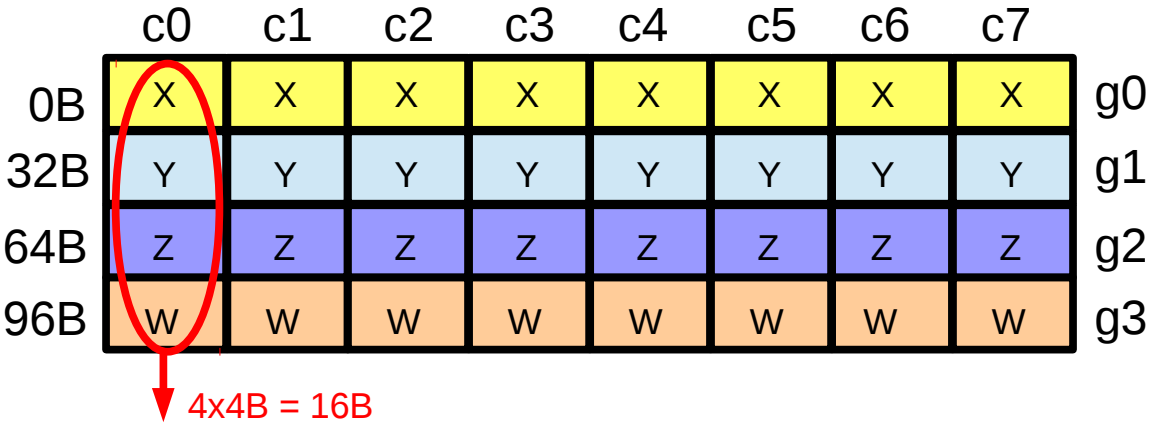
Unfamiliar code patterns

- Fp64 operation produces new code patterns:
  - 32-bit access patterns on low/high 32-bit chunks of 64-bit data
  - Horizontal strides != 1
- Some parts of the driver did not handle these scenarios properly.
    - Copy-propagation received at least 7 patches!

32-bit read/write messages

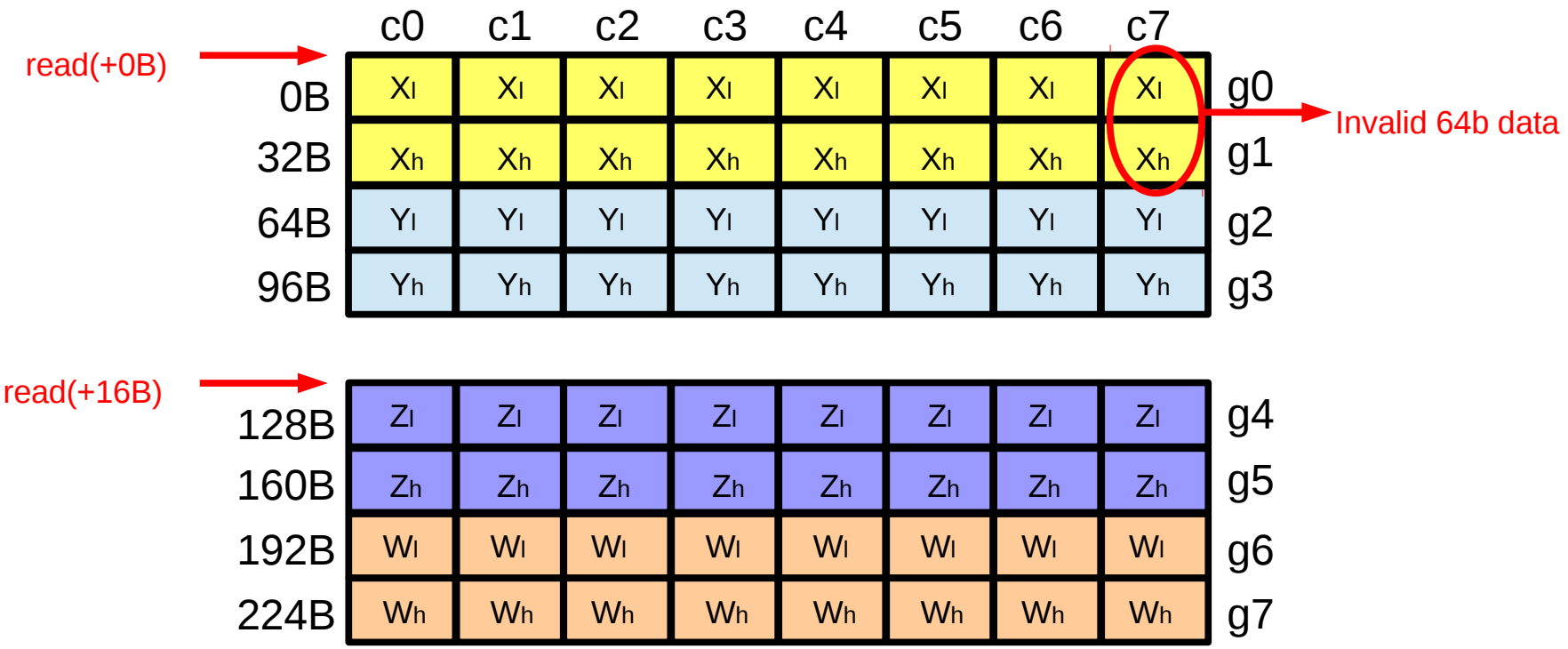
- **All read/write messages are 32-bit**
  - Pull loads, UBOs, SSBOs, URB, scratch...
- 64bit data needs to be shuffled into 32-bit channels before writing
- 32bit data reads need to be shuffled into valid 64bit data channels
  - shuffle_64bit_data_for_32bit_write()
  - shuffle_32bit_load_result_to_64bit_data()

32-bit read/write messages (SIMD8 read)

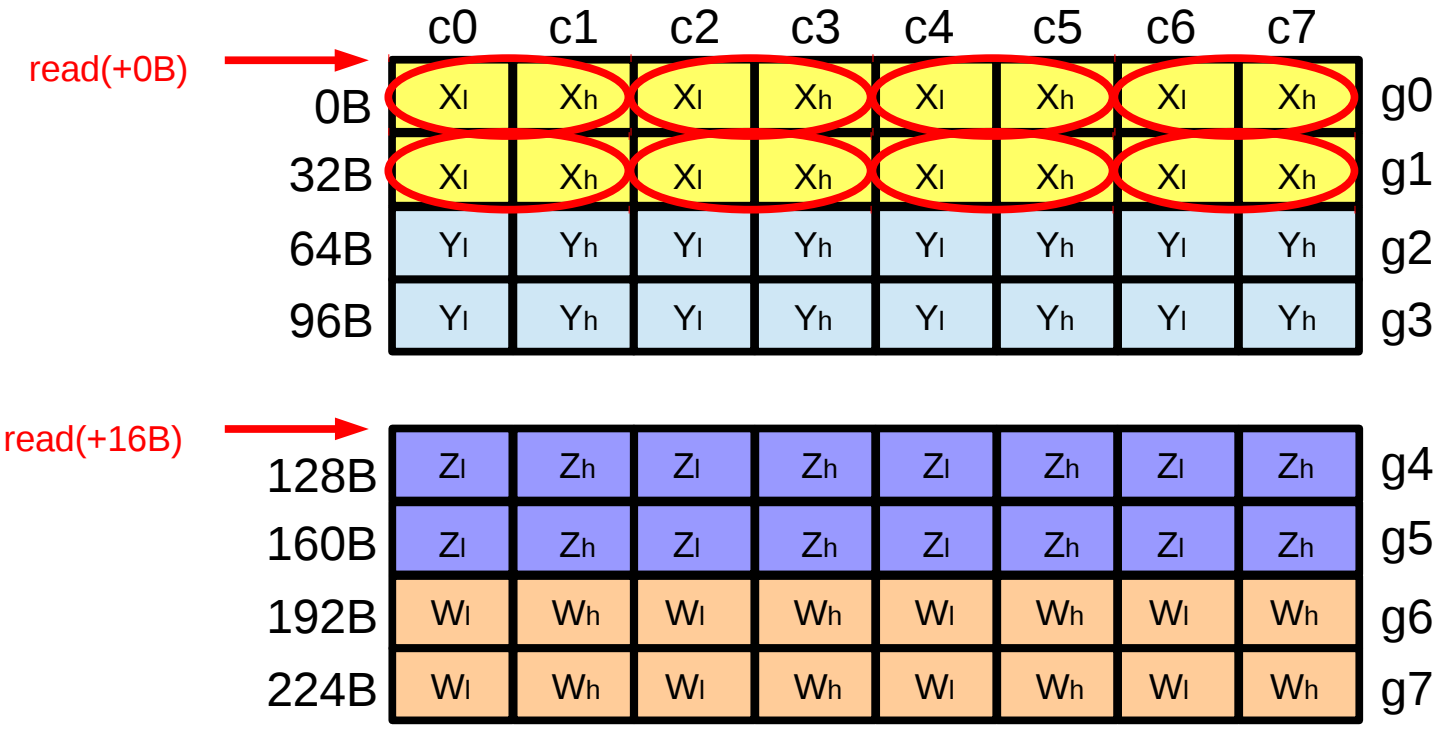


- Just what we want for 32-bit scalar operation
  - 8x16B SIMD8 read messages
  - Separate variables (registers) for each component
  - Consecutive components in consecutive registers

32-bit read/write messages (SIMD8 read)



32-bit read/write messages (SIMD8 read)



64-bit immediates (gen7)

- **No support for 64-bit immediates** (Haswell, IvyBridge)
  - Haswell provides the *DIM* instruction specifically for this purpose, we just had to add support for this in the driver.
  - IvyBridge requires that we emit code to load each 32-bit chunk of the constant into a register and then return either a XXXX swizzle (align16) or a stride 0 (align1).

Bugs / restrictions (Align1)

- **Second half of compressed instructions that don’t write all channels have wrong emask** (Haswell)
  - Requires SIMD splitting
- **Second half of compressed 64-bit instructions has wrong emask** (IvyBridge)
  - Requires unconditional SIMD splitting of **all** 64-bit instructions :-(

Bugs / restrictions (Align16)

- **Vertical stride 0 doesn't work** (gen7)
  - The second half of a compressed instruction will invariably offset a full register
  - Requires SIMD splitting
- **Instructions that write 2 registers must also read 2 registers** (gen7)
  - This was a known issue in gen7, but it was never triggered in Align16 before Fp64
  - Requires SIMD splitting

A simple test that just copies a DF uniform to the output hits both of these bugs! :-(

Bugs / restrictions (Align16)

- **3-src instructions can’t use RepCtrl=1**
  - Not supported for 64-bit instructions
  - Only affects to MAD
  - RepCtrl=0 leads to <4,4,1>:DF regions so it can only be used to work with components XY
    - Requires temporaries and component splitting, but leads to quite bad code in general
    - Avoiding MAD altogether seems a better option for now

Bugs / restrictions (Align16)

- **Compressed bcsel**
  - Does not read the predication mask properly
  - Requires SIMD splitting
- **Dependency control**
  - Can’t be used with 64-bit instructions → GPU hangs

Current State

Status

- **Skylake:** Available in Mesa 12.0
- **Broadwell:** Available in Mesa 12.0
- **Haswell:**
  - ARB_gpu_shader_fp64: Implemented, in review
  - ARB_vertex_attrib_64bit: Implemented
- **IvyBridge:**
  - ARB_gpu_shader_fp64: Align1 implemented, Align16 implementation in progress
  - ARB_vertex_attrib_64bit: Not started

Questions?