Reference Guide

Compute Abstraction Layer (CAL) Technology

Intermediate Language (IL)

February 2009

v. 2.0

doc_rev1.4

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA 94088-3453
www.amd.com

Preface

About This Document

This document describes the instruction set for the CAL Technology IL compiler.

The document serves two purposes:

1. It specifies the language constructs and behavior, including the organization of each type of instruction in both text syntax and binary format.
2. It provides a reference of instruction operation that compiler writers can use to maximize performance of the processor.

Audience

This document is intended for programmers writing application and system software, including operating systems, compilers, loaders, linkers, device drivers, and system utilities. It assumes an understanding of the R600- and R700-family processor microarchitecture and of programming practices for either graphics or general-purpose computing.

Contact Information

To submit questions or comments about this document, contact our technical documentation staff at: streamcomputing@amd.com.

For questions concerning ATI Stream products, please email: streamcomputing@amd.com.

For questions about developing with ATI Stream, please email: streamdeveloper@amd.com.

You can learn more about ATI Stream at: http://www.amd.com/stream.

We also have a growing community of ATI Stream users! Come visit us at the ATI Stream Developer Forum (http://www.amd.com/streamdevforum) to find out what applications other users are trying on their ATI Stream products!

Organization

This document begins with an overview summarizing the similarities and differences between ATI Intermediate Language (IL) and general-purpose

computer languages. It describes text and binary formats of the IL program instructions. Then, it describes the types of instructions in detail, presenting a high-level description of the instruction fields, and restrictions that must be observed. It also describes the instruction syntax for text representation. Further, it presents the specification of each type of instruction. A glossary of terms and acronyms ends the document.

Endian Order

The R600 and R700 architectures address memory and registers using little-endian byte-ordering and bit-ordering. Multi-byte values are stored with their least-significant (low-order) byte (LSB) at the lowest byte address; they are illustrated with their LSB at the right side. Byte values are stored with their least-significant (low-order) bit (lsb) at the lowest bit address; they are illustrated with their lsb at the right side.

Conventions

The following conventions are used in this document.

`mono-spaced font`	A filename, file path, or code.
*	Any number of alphanumeric characters in the name of a microcode format, microcode parameter, or instruction.
< >	Angle brackets denote streams.
[1,2)	A range that includes the left-most value (in this case, 1) but excludes the right-most value (in this case, 2).
[1,2]	A range that includes both the left-most and right-most values (in this case, 1 and 2).
{x | y}	One of the multiple options listed. In this case, x or y.
0.0	A single-precision (32-bit) floating-point value.
1011b	A binary value, in this example a 4-bit value.
7:4	A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first.

Related Documents

- AMD, *R600-Family Instruction Set Architecture*, Sunnyvale, CA, 2008. This document includes the RV670 GPU instruction details.
- ISO/IEC 9899:TC2 - *International Standard - Programming Languages - C*
- Kernighan Brian W., and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1978.
- IEEE, *754-1985 IEEE Standard for Binary Floating-Point Arithmetic*, 2003.
- I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, *"Brook for GPUs: stream computing on graphics hardware,"* ACM Trans. Graph., vol. 23, no. 3, pp. 777–786, 2004.

- *ATI Compute Abstraction Layer (CAL) Intermediate Language (IL) Reference Manual*. Published by AMD.
- Buck, Ian; Foley, Tim; Horn, Daniel; Sugerman, Jeremy; Hanrahan, Pat; Houston, Mike; Fatahalian, Kayvon. “BrookGPU” http://graphics.stanford.edu/projects/brookgpu/
- Buck, Ian. “Brook Spec v0.2”. October 31, 2003. http://merrimac.stanford.edu/brook/brookspec-05-20-03.pdf
- *OpenGL Programming Guide*, at http://www.glprogramming.com/red/
- *Microsoft DirectX Reference Website*, at http://msdn.microsoft.com/en-us/library/bb219740(VS.85).aspx
- *Microsoft Programming Guide for HLSL*, http://msdn2.microsoft.com/en-us/library/bb509635.aspx
- *GPGPU*: http://www.gpgpu.org, and Stanford BrookGPU discussion forum http://www.gpgpu.org/forums/

Contents

Chapter 4 Register Types

Chapter 5 Enumerated Types

Tables

Chapter 1
Overview

This document defines the format and behavior of the IL. The Intermediate Language (IL) is an abstract representation for hardware vertex, pixel, and geometry shaders that can be taken as input by other modules implementing the IL. An IL compiler uses an IL shader in conjunction with driver state information to translate these shaders into hardware instructions or a software emulation layer.

1.1 Open Design

The IL adopts an open design, where most instructions can appear in any kind of shader.

Note that the IL does not enforce usage restrictions of the input language, but only verifies that the IL, as specified, is properly programmed. For example, in DX, instruction modifiers such as _x2 general-purpose not be applied to texture address instructions. The IL does not enforce this restriction, since both the IL and current architectures support such operations.

Also, this manual does not describe how to optimally code the IL. By design, the IL has an extensive set of operations, but some of these are not supported by existing hardware. The programmer is advised to adhere in most cases to the syntax restrictions and performance guidelines of the input language.

1.2 DirectX as a Design Basis

Because of changes between DX9 and DX10, many instructions occur in multiple forms:

- unconditional
- conditional (comparing two float values)
- logical (comparing a single integer value with zero)
- Boolean (comparing a Boolean register with true)

To support DX10, many operations required several similar IL opcodes. For example, there are three forms of break: unconditional, break on a Boolean, and break on a logical value.

Other operations were required both in vector and scalar forms. For example, there are two `rsq` instructions: `rsq` corresponds to the IL 1 scalar opcode,

rsq_vec corresponds to the DX10 vector form that computes the reciprocal square root on each component.

In DX9, 0* any value was defined to be 0. DX10 changed this to more closely match IEEE arithmetic which defines 0*Nan = Nan.

All float operations containing a multiply now take a flag to specify Nan behavior.

1.3 Threading Model

A hierarchical threading model is used. A *kernel* (a shader program running on a GPU) can be launched with a number of thread groups. Threads within this group can communicate through local shared memory. There is no supported communication between thread groups. Threads in a thread group run in units called wavefronts. All threads in a wavefront run in SIMD fashion (in lock steps). All wavefronts within a thread group can be synchronized using a synchronization barrier operation.

1.4 Access Model for Local Shared Memory

Each processor has an amount of local memory that can be shared across the threads in a thread group. IL provides two models of memory access to local data store (LDS).

The first memory access model, called owner-computes, is supported by the HD4000-family of devices. In owner-computes, each thread in a thread group owns a area of LDS memory. The size of the area is declared in the shader. Each thread in a group can write only to the area of memory it owns; however, a thread can read any chunk of memory that is owned by either itself or other threads. An LDS shared memory read is specified by (owner_thread_ID, offset): read the memory area owned by that thread_ID with an offset within the area.

Different from the access model for threads within a wavefront, the access mode for different wavefronts (within a thread group) is specified by the sharing mode, which is either relative or absolute. If it is relative, new and consecutive space is allocated for each wavefront; if it is absolute, all wavefronts are mapped to the same set of memory starting at address 0. In this mode, wavefronts can overwrite each other's data.

The second memory access model is a general read write: each thread can read or write any address in the LDS.

Both models allow threads to read or write memory (video or system), but do not provide synchronization to memory.

Supported inter-thread communication includes:

- SR – Globally shared registers.
- Sharing between all wavefronts in a SIMD.
- Column sharing on the SIMD.

- Persistent registers.
- LDS – local data store - read/write.
- Data sharing between all threads in a group.
- Required synchronization.
- Memory - read/write.
- Constant buffers
- Texture cache

The following indexing values are available in the compute shader:

- vTid – ID of thread within a group
- aTid – ID of thread within a kernel

Chapter 2
Binary Stream Format

The following chapter defines the format in which kernels written using the IL are passed to the compiler.

2.1 IL Stream

Clients pass kernels as a stream of 32-bit tokens organized as variable-length instruction packets. These tokens include information about the client language, shader type, and instruction packets that describe the operation of the kernels. Table 2.1 indicates the ordering of packets.

Table 2.1 IL Stream Instruction Packet Ordering

Instruction Packet	Description
1	IL_Lang token. See Section 2.2.1, on page 2-2.
2	IL_Version token. See Section 2.2.2, on page 2-2.
3	IL_OpCode token describing the operation of the first instruction in the stream and the beginning of the first IL instruction packet.
...	*... More IL instruction packets*. See Chapter 3.
n (number of 32-bit tokens in the stream)	*IL instruction packet for an* END *instruction*. See page 6-13.

Instruction Packets contain all the information needed to perform the single instruction specified in the IL_OpCode token. This information can include data about source and destination operands, destination or target labels, and additional data needed to perform the instruction.

There are assorted IL statements that can be used to declare resources, samplers, or registers. Any declaration of an object must appear before all uses of the object. There is no requirement to group all declarations at the start of the program.

Most IL statements and types can be used in any kind of shader. However, as noted below, some statements and types are restricted to specific kinds of shaders.

2.2 IL Token Descriptions

This section describes the generic tokens used in the IL stream. There are additional tokens for use in single specific instruction packets. Those tokens are

described under the instruction packet description for the instruction in which they can be used.

2.2.1 Language Token

This token indicates the type of client generating the IL. This token must be at the beginning of every IL stream passed to the compiler.

Table 2.2 IL_Lang: Source Language Information

Field Name	Bits	Description
client_type	7:0	Specifies the client API. Can be any value of the enumerated type ILLanguage-Type. This value is not used, but can allow IL compilers to make API specific workarounds and optimizations.
reserved	31:8	Must be zero.

2.2.2 Version Token

This token specifies the version of IL used in this IL stream. It also specifies the type of kernel the IL stream represents (pixel or vertex).

Table 2.3 IL_Version: Source Version Information

Field Name	Bits	Description
minor_version	7:0	The minor version.
major_version	15:8	The major version.
shader_type	23:16	Specifies the type of shader described by this binary token stream. See Section 5.1, “ILShader,” page 5-1.
multipass	24	0 = Outputs are not ignored. 1 = This shader is for multipass use only (output are ignored).
realtime	25	0 = This shader is not for real-time use. 1 = This shader is for real-time use. This can be set only when the shader type is IL_PIXEL_SHADER.
reserved	31:26	Must be zero.

2.2.3 Opcode Token

This token specifies the current operation and information required to perform the operation.

Table 2.4 IL_Opcode: Instruction Opcode Details

Field Name	Bits	Description
code	15:0	The operation for the current instruction. This value can be any of the enumerated type ILOpCode (see Section 5.27, “ILOpCode,” page 5-13).
control	29:16	Opcode specific control. Possible values for this depend on the value of *code*. Specifies further instruction behavior. This field must be zero for all instructions not using it.

Table 2.4 IL_Opcode: Instruction Opcode Details (Cont.)

Field Name	Bits	Description
sec_modifier_present	30	Specifies whether an opcode-specific token describing further instruction behavior follows the primary modifier token. 0 An opcode specific token does not follow. 1 An opcode specific token follows. This field must be zero for all instructions not using it.
pri_modifier_present	31	Specifies whether an opcode-specific token describing further instruction behavior follows this token. 0 An opcode specific token does not follow. 1 An opcode specific token follows. This field must be zero for all instructions not using it.

2.2.4 Destination Token

This token specifies the register to which the hardware passes the result of the current instruction and other information pertaining to this result. This token can only be issued after an IL_OpCode token has been issued as part of an instruction packet. By default, all components of the register specified are written unless the modifier_present field is 1 and an IL_Dst_Mod token follows.

DX10 allows an additional kind of indexing: some temporary objects can be indexed by a register; thus, the IL now allows an additional modifier (register relative modifier). Two kinds of destinations can be indexed: IL_TEMP_ARRAY and IL_OUTPUT.

The immediate_present field is used for indexed data types: itemp, cb, etc. It can be used if the data type uses absolute, reg-relative, loop, or addr relative addressing. See Section 2.2.6, “Source Token,” page 2-4, for examples.

Table 2.5 IL_Dst: Destination Operand Information

Field Name	Bits	Description
register_num	15:0	Register number to which the result of the current instruction is written. See Chapter 4, “Register Types,” for acceptable values.
register_type	21:16	This value can be any of the enumerated type ILRegType (see Section 5.3, “ILRegType,” page 5-2).
modifier_present	22	Specifies whether an IL_Dst_Mod token follows this token: 0 An IL_Dst_Mod token does not follow. 1 An IL_Dst_Mod token follows.
relative_address	24:23	Specifies whether register_num represents an offset from the address register specified the following IL_Rel_Addr token: 0 Absolute addressing is used. 1 Relative addressing is used (an IL_Rel_Addr token follows). See Section 2.2.8, on page 2-8. 2 Register relative addressing is used (tokens describing the index follows) Dimension specifies the number of source tokens that need to follow.
dimension	25	Number of additional following dimensions (0 == 1 dimension). so v[3][5] is src_token 1: il_regtype_vertex dim = 1 num = 3 src_token 2: il_regtype_vertex dim = 0 num = 5

Table 2.5 IL_Dst: Destination Operand Information (Cont.)

Field Name	Bits	Description
immediate_present	26	0 There is no immediate value. 1 A 32-bit value containing the immediate value follows this token, modifier tokens, and src tokens used in register relative addressing.
reserved	27:30	Must be zero.
extended	31	0 No extended register addressing. 1 The register_number is a 32-bit value. The low 16-bits is represented by the register_num field, and following this token is a 32-bit word containing the high 16-bits and 16-bits reserved.

2.2.5 Destination Modifier Token

This token specifies modifications to the destination operand.

The modifiers precedence is:

1. shift_scale
2. clamp
3. destination comp

An example of all these operations performed on the x-component of a destination operand is:

```
dstCmpMod(clamp(shift_scale(dst.x)))
```

Table 2.6 IL_Dst_Mod: Destination Modification Information

Field Name	Bits	Description
component_x_r	1:0	This value can be any of the enumerated type ILModDstComp[1].
component_y_g	3:2	This value can be any of the enumerated type ILModDstComp[1].
component_z_b	5:4	This value can be any of the enumerated type ILModDstComp[1].
component_w_a	7:6	This value can be any of the enumerated type ILModDstComp[1].
clamp	8	Specifies whether to clamp the value to 0.0 and 1.0: 0 Do not clamp. 1 Clamp. Clamp(NaN) returns 0.
shift_scale	12:9	This value can be any of the enumerated type ILShiftScale[2].
reserved	31:13	Must be zero.

1. See Section 5.7, “ILModDstComp,” page 5-4.
2. See Section 5.11, “ILShiftScale,” page 5-7.

2.2.6 Source Token

This token specifies the register that the instruction uses as a source operand. This token can only be issued after an IL_OpCode token as part of an instruction packet. If an IL_Src_Mod token does not follow, then:

- the first component is set to the x component,

- the second component is set to the y component,
- the third component is set to the z component, and
- the fourth component is set to the w component.

Starting with IL 2.0, some source tokens are allowed to use register relative indexing. Only one level of indexing is allowed. The type of register that has indexed sources must be one of the following: IL_TEMP_ARRAY, IL_CONST_BUFFER, or IL_INPUT. Since the index must be a scalar value, a modifier field must be used to replicate a single component into four slots.

Table 2.7 lists and briefly describes the IL_src source operands. IL version 2.0 also allows a source token to refer to a literal defined in a dcl_literal statement. DX10 statements such as:

```
add r1, r2, float4 (1.0f, 2.0f, 3.0f, 4.0f)
```

can be translated into:

```
dcl_literal_float4, l1, 1.0f, 2.0f, 3.0f, 4.0f
add r1, r2, l1
```

Table 2.7 IL_Src: Source Operand Information

Field Name	Bits	Description
register_num	15:0	Register number from which to retrieve the operand. See Chapter 4, “Register Types,” for acceptable values of this field.
register_type	21:16	This value can be any of the enumerated type ILRegType. See Chapter 4, “Register Types,” for a description of these types.
modifier_present	22	Specifies whether an IL_Src_Mod token follows this token: 0 An IL_Src_Mod token does not follow. 1 An IL_Src_Mod token follows.
relative_address	24:23	Specifies whether the register_num represents an offset from the address register specified in the following IL_Rel_Addr token or an absolute address: 0 Absolute addressing is used. 1 Relative addressing is used (an IL_Rel_Addr token follows). See Section 2.2.8, on page 2-8. 2 Register relative addressing is used (source tokens follows). Dimension specifies the number of source tokens that need to follow.
dimension	25	Number of additional following dimensions (0 == 1 dimension). so v[3][5] is src_token 1: il_regtype_vertex dim = 1 num = 3 src_token 2: il_regtype_vertex dim = 0 num = 5
immediate_present	26	0 There is no immediate value. 1 A 32-bit value containing the immediate value follows this token, modifier tokens and src tokens used in register relative addressing.
reserved	27:30	Must be zero.
extended	31	0 There is no extended register addressing. 1 The register_number is a 32-bit value. The low 16-bits is represented by the register_num field and following this token is a 32-bit word containing the high 16-bits and 16-bits reserved.

2.2.7 Source Modifier Token

This token specifies modifications to the source operand. It can be issued only immediately following an IL_Src token, and only if the preceding IL_Src token has the modifier_present field set to 1.

When this token is used in conjunction with an IL_Rel_Addr token, modifiers are applied to the register referenced by the relative address, not to the relative-address register itself. Also, this token always precedes the IL_Rel_Addr token if both are used.

Notes about this token:

- If this token is not present, the x, y, z, w corresponds to the first, second, third, and fourth components, respectively.
  - For floating point arithmetic instructions, the negate modifier simply flips the sign of the number(s) in the source operand, including on INF values. Applying negate on NaN preserves NaN, although the particular NaN bit pattern that results is not defined.
  - For integer instructions, the negate modifier takes the 2's complement of the number(s) in the source operand.
  - For floating point arithmetic instructions: the abs modifier simply forces the sign of the number(s) on the source operand positive, including on INF values. Applying abs on NaN preserves NaN, although the NaN bit pattern that results is not defined.
  - The 1 swizzle inserts a floating point 1.0f, even if the opcode is an integer operation. This can lead to unexpected results. For example, when evaluating the second source of iadd r1, r1, r1.1_neg(xyzw), implementations take a floating point 1.0f and treat it as an integer. To negate an integer, use the INEGATE function.
- The modifiers use the following precedence:
  1. swizzle
  2. invert
  3. bias
  4. x2
  5. sign
  6. divComp
  7. abs
  8. negate (per component)
  9. clamp

An example of all of these operations performed on the x-component of a source operand is:

```
clamp(negate (abs (divComp(sign(x2(bias(invert(swizzle(src's))))))))))
```

The modifiers *bias*, *x2*, *divComp*, and *clamp* cannot be used when the opcode of the instruction specifies an integer or logical operation.

Table 2.8 IL_Src_Mod: Source Operand Modification Information

Field Name	Bits	Description
swizzle_x_r	2:0	This value can be any of the enumerated type ILComponentSelect[1]. If this token is not present, the IL behaves as if this field is set to IL_COMPSEL_X_R.
negate_x_r	3	Specifies whether to negate the first component: 0 Do not negate. 1 Negate.
swizzle_y_g	6:4	This value can be any of the enumerated type ILComponentSelect[1]. If this token is not present, the IL behaves as if this field is set to IL_COMPSEL_Y_G.
negate_y_g	7	Specifies whether to negate the second component: 0 Do not negate. 1 Negate.
swizzle_z_b	10:8	This value can be any of the enumerated type ILComponentSelect[1]. If this token is not present, the IL behaves as if this field is set to IL_COMPSEL_Z_B.
negate_z_b	11	Specifies whether to negate the third component: 0 Do not negate. 1 Negate.
swizzle_w_a	14:12	This value can be any of the enumerated type ILComponentSelect[1]. If this token is not present, the IL behaves as if this field is set to IL_COMPSEL_W_A.
negate_w_a	15	Specifies whether to negate the fourth component: 0 Do not negate. 1 Negate.
invert	16	Specifies whether to invert each value of the source register prior to the operation. (s = 1.0 – s). The 'invert' modifier has higher precedence then bias, x2, negate sign, divComp ops, or abs: 0 Do not invert. 1 Invert.
bias	17	Specifies whether to bias each value of the source register prior to the operation. (s = s – 0.5). This takes precedence over x2, negate, sign, divComp ops, and abs: 0 Do not bias. 1 Bias.
x2	18	Specifies whether to multiply each value by 2.0 prior to the operation (s = 2.0 * s). This takes precedence over negate, sign, divComp ops, and abs: 0 Do not perform x2. 1 Perform x2.
sign	19	Specifies whether to sign each value of the source register prior to the operation (s = (s < 0) ? –1 : ((s == 0) ? 0 : 1)). This takes precedence before divComp ops, and abs: 0 Do not sign. 1 Sign.

Table 2.8 IL_Src_Mod: Source Operand Modification Information (Cont.)

Field Name	Bits	Description
abs	20	Specifies whether to take the absolute value of each value of the source register prior to the operation. (s = (s < 0) ? –s : s): 0 Do not take the absolute value. 1 Take the absolute value.
divComp	23:21	Specifies component divide modifier. This can be any value of the enumerated type ILDivComp. This takes precedence over abs. See Section 5.12, “ILDiv-Comp,” page 5-8. This field indicates one of the following: • No component divide necessary. • The first component is divided by the second component. • The first and second component is divided by the third component. • The first, second, and third component is divided by the fourth component. Any one of these four are performed based on this field.
clamp	31:24	Specifies whether or not to clamp the value to [0.0, 1.0]. 0 Do not clamp 1 Clamp clamp(NaN) returns 0

1. See Section 5.6, “ILComponentSelect,” page 5-3.

2.2.8 Relative Address Token

This token specifies data to use for relative addressing. This token can be issued after an IL_Src, IL_Src_Mod, IL_Dst, or IL_Dst_Mod token. It follows an IL_Src token if *relative_address* is 1 and *modifier_present* is 0 in the preceding IL_Src token. It follows an IL_Src_Mod token if *relative_address* is 1 and *modifier_present* is 1 in the preceding IL_Src token. It follows an IL_Dst token if *relative_address* is 1 and *modifier_present* is 0 in the preceding IL_Dst token. It follows an IL_Dst_Mod token if *relative_address* is 1 and *modifier_present* is 1 in the preceding IL_Dst token. See the Section 3.3, “Relative Addressing,” page 3-2, for more details on using this token.

Table 2.9 IL_Rel_Addr: Relative Addressing Descriptor

Field Name	Bits	Description
address_register	15:0	Specifies the address register number to use for base relative addressing. (In the future this can specify the loop counter to use for loop relative addressing. In the current version of the IL, only the loop counter for the inner-most loop can be used).
loop_relative	16	Specifies whether loop or base relative addressing is used: 0 Base relative addressing is used. 1 Loop relative addressing is used. This field must be 1 if relative addressing is used on a destination.
component	19:17	Specifies which component of the address register to use as the base address. If *loop_relative* is 0, this field can be set to any value of the enumerated type ILComponentSelect. If *loop_relative* is 1, this field must be set to IL_COMPSEL_X_R
reserved	31:20	Must be zero.

2.2.9 Source Token Examples

Some source token encodings can be complicated, as shown in the following examples.

Example 1: *Source Operand:* x5[6].y

- IL_Src
  - register_num = 5
  - register_type = IL_REGTYPE_ITEMP (see Section 6.4, “Declaration and Initialization Instructions,” page 6-22)
  - modifier_present = 1
  - relative_address = IL_ADDR_ABSOLUTE
  - dimension = 0
  - immediate_present = 1
  - extended = 0
- IL_Src_Mod
  - swizzle_x_r = y
  - swizzle_y_g = y
  - swizzle_z_b = y
  - swizzle_w_a = y
- IL_Literal
  - val = 6

Example 2: *Source Operand:* x5[r2.x+6].y

Here are the IL tokens for this:

- IL_Src x5
  - register_num = 5
  - register_type = IL_REGTYPE_ITEMP
  - modifier_present = 1
  - relative_address = IL_ADDR_REG_RELATIVE
  - dimension = 0
  - immediate_present = 1
  - extended = 0
- IL_Src_Mod
  - swizzle_x_r = y
  - swizzle_y_g = y
  - swizzle_z_b = y
  - swizzle_w_a = y

- IL_Src
  - register_num = 2
  - register_type = IL_REGTYPE_TEMP
  - modifier_present = 1
  - relative_address = IL_ADDR_ABSOLUTE
  - dimension = 0
  - immediate_present = 0
  - extended = 0
- IL_Src_Mod .x for r2
  - swizzle_x_r = x
  - swizzle_y_g = x
  - swizzle_z_b = x
  - swizzle_w_a = x
- IL_Literal
  - val = 6

Example 3: *Source Operand:* v[1][2] (all fields set to zero, unless otherwise stated)

- IL_Src
  - Register_num = 1
  - Register_type = IL_REGTYPE_VERTEX
  - Dimension = 1
- IL_Src
  - Register_num = 2
  - Register_type = IL_REGTYPE_VERTEX

Example 4: *Source Operand:* v[1][2].xyxx (all fields set to zero unless otherwise stated)

- IL_Src
  - Register_num = 1
  - Register_type = IL_REGTYPE_VERTEX
  - Dimension = 1
  - Modifier_present = 1
- IL_Src_Mod
  - swizzle_x_r = x
  - swizzle_y_g = y
  - swizzle_z_b = x

  - swizzle_w_a = x
- IL_Src
  - Register_num = 2
  - Register_type = IL_REGTYPE_VERTEX

Chapter 3
Text Instruction Syntax

IL Text syntax is designed to closely match the IL specification, so that there is an almost complete one-to-one mapping.

Below is a simple vertex and pixel shader pair written in IL Text syntax that renders green stripes.

```
il_vs
dclv_elem(0) v0                    ; Declare position
dclv_elem(1) v1                    ; Declare color
dclv_elem(2) v2                    ; Declare texture coordinates

mmul_matrix(4x4) oPos, v0, c[0]    ; Transform position to clip space

mov oPriColor0, v1                 ; Export vertex color
mov oT0, v2                        ; Export texture coordinates

end

il_ps
dclpi_x(1)_y(1)_z(1)_w(1) vPriColor0 ; Declare primary color import
dclpi_x(1)_y(1)_z(*)_w(*) vT0      ; Declare vs import texture
                                     coordinates

def c0, 0.5, 1, 0, 0
def c1, 0.0, 1.0, 0.0, 1.0        ; Green color

mod r0.x, vT0.x, c0.y              ; x = mod( s, 1.0 )

ifc_relop(lt) r0.x, c0.x           ; if ( x < 0.5 )
   mov oC0, vPriColor0             ; Output surface color
else ; else
   mov oC0.rgb1, c1                ; Output green color
endif

end
```

3.1 Version

The first two tokens in an IL binary stream are IL_Lang and IL_Version. IL Text syntax combines these into a single version instruction. The IL translator sets the language type to IL_LANG_VERSION and disables the language defaults. The IL_Version token has the following syntax:

```
il_ps_major_minor_mp_rt
il_vs_major_minor_mp_rt
```

If the major and minor version are not specified, the IL translator inserts them based on its own version. Typically, the version information is omitted, unless a specific optimization made in the compiler is wanted. If a shader represents a multipass shader, append _mp to the statement. If this shader represents a real-time shader, append _rt to the statement.

3.2 Registers

Registers that are prefixed with the letter “v” are read-only (import) buffers, registers prefixed with the letter “o” are write-only (export) buffers. This section only lists the registers. See Chapter 4 for more information on register types.

Common Registers:

b#, c#, i#, a#, aL, and r#

Vertex Shader:

v#, oPos#, oPriColor#, oSecColor#, oT#, oInterp#, oFog, oSprite, vBaryCood, vPrimIndex, vQuadIndex

Pixel Shader:

vPriColor#, vSecColor#, vT#, vInterp#, vFog, vSprite, vFace, vWinCoord, oC#, oDepth

3.3 Relative Addressing

The [ ] syntax can be used on register types that can be relatively addressed.

Absolute Register Index

reg[m]	Example: mov r[0], r[1]
reg[m0+...+mn]	Example: mov r0, r[1+2]

Note that r[0] is the same as r0.

Base Relative Addressing

The address register is used to address another register file.

reg[A+m0+...+mj]	Example: mov r0, r[a0.x+5]
reg[m0+...+mn+A+n0+...+nn]	Example: mov r0, r[1+3+a0.y+2]

Loop Relative Addressing

The loop register aL is used to address another register file.

reg[A+m0+...+mj]	Example: mov r0 r[aL+5]
reg[m0+...+mn+A+n0+...+nn]	Example: mov r0, r[1+3+aL+2]

3.4 Control Specifiers

Control specifiers affect the behavior of the instruction. Binary control specifiers, which typically indicate there are two possible actions, have the following syntax:

<instr>[_ctrl]

If the control specifier _ctrlspec is left off the instruction, the default action is performed.

Control specifiers that have more than two modes of operation have the following syntax:

instruction_ctrl(*value*)]

For these specifiers, the parenthesis is mandatory, and the value specifies the mode of operation. Sections 3.5, 3.6, 3.7, and 3.8 describe the control specifiers in the IL spec. See Chapter 6, “Instructions,” to determine which specifiers go with which instructions.

3.5 Destination Modifiers

Instruction modifiers are applied after the instruction runs but before writing the result to the destination register.

Table 3.1 Destination Modifiers

Modifier	Description	Example
_x2 _x4 _x8 _d2 _d4 _d8	Shift scale modifiers.	add_x2 r0, r1, r2
_sat	Saturate or clamp result to [0,1].	add_sat r0, r1, r2 add_x2_sat r0, r1, r2

3.6 Write Mask

Component-wise write masks on the destination have the following syntax in IL Text:

```
reg.{x|_|0|1}{y|_|0|1}{z|_|0|1}{w|_|0|1}
reg.{r|_|0|1}{g|_|0|1}{b|_|0|1}{a|_|0|1}
```

A letter specifies a vector component to be written, while an underscore specifies that an component is not written. An component can also be forced to zero or one.

Examples:

```
mov r0.x, r1
mov r0.y r1
mov r0.z r1
mov r0.x_zw, r1
mov r0.y_w, r1
mov r0.__w, r1
mov r0._1_w, r1
mov r0.01z, r1
mov r0.01z1, r1
mov r0.11, r1
mov r0._11, r1
mov r0.__11, r1
```

```
mov r0.xyz1, r1
mov r0_1_0, r1
mov r0.11_0, r1
mov r0.y00, r1
mov r0.yz1, r1
mov r0.y_1, r1
```

3.7 Source Modifiers

Table 3.2 Source Modifiers

Modifier	Description	Example
swizzle	Rearrange and/or replicate components.	mov r0, r1.zyx1 mov r0, r1.xxyy
_abs	Takes the absolute value of components.	mov r0, r1_abs
_bias	Components are biased (x – 0.5).	add r0, r1, r2_bias
_bx2	Signed scaling. Combined bias and x2 modifiers.	add r0, r1, r2_bx2
_divcomp(*type*)	Performs division based on *type*. *type*: y, z, w, unknown	texId_stage(0) r0, vT0_divcomp(y)
_invert	Invert components (1.0 - x).	add r0, r1_invert, r2
_neg(*comp*)	Provides per component negate.	mov r0 r1_neg(xw)
_sign	Signs Components: Components less then 0 become -1. Components equal to 0 become 0. Components greater then 0 become 1.	mov r0, r1_sign
_x2	Multiply components by 2.0.	add r0, r1, r2_x2

3.8 Comments

Only single-line comments are supported using a semicolon as the delimiter. Other commenting styles, such as /* */ matching pairs, also are supported. A comment can start from any position on the line.

Example:

```
; The following instruction moves the contents of r1 into r0
mov r0, r1 ; mov instruction
```

Chapter 4
Register Types

This chapter describes the ATI IL valid register types.

Table 4.1 Registers and Their Restrictions

Name	IL Regtype_	Syntax	Num Comps	Read	Write	Relative Address			Notes
						Loop	A0	Index	
Boolean	CONST_BOOL	b#	1	No	Defb only	No	No	No	Can be used only as the source for static control flow.
Prim id	VPRIM	vPrim	1	Yes	No	No	No	No	Used only in geometry shader.
buffer	CONST_BUFF	C#[#]	4	Yes	No	No	No	Yes	Max 4096 elements.
float	CONST_FLOAT	C#	4	Yes	Def only	No	Yes	No	
Int	CONST_INT	I#	4	Loop only	Def only	No	No	No	
Addr	ADDR	A0	4	Yes	Mova	No	No	No	
Temp	TEMP	R#	4	Yes	Write	No	No	No	Max 4096 elements.
Index temp	INDEXED_TEMP	X#	4	Yes	Yes	No	No	Yes	Max 4096 elements.
Vertex	VERTEX	V#	4	Yes	Dclv or initv	No	No	No	Cannot be used in pixel shader.
index	INDEX	vINDEX	4	Yes					
Output	OUTPUT	o#	4	No	Yes	No	No	No	To support DX10.
Input	INPUT	v#	4	Yes	No	No	No	No	To support DX10.

4.1 INDEX

Enum: IL_REGTYPE_INDEX

Text Syntax: `vIndex`

Common Name: Index Register

Number of Components per Register: 4

Description:

- Vertex shader import for the index from the index buffer of the current vertex processed.

- Not guaranteed to be incremental.
- For non-HOS (high-order surface) rendering, the first component is the index of the current vertex processed.
- For HOS rendering (when HOS is enabled) the first, second, third, and fourth components represent the indices for the superprim vertices for the current tessellated vertex being processed. The superprim indices are output by the tessellation engine based on the relevant HOS state.
- This register cannot be used with relative addressing.
- It is an error to use this register in a pixel shader.
- This is a read-only register.

4.2 OBJECT_INDEX

Enum: IL_REGTYPE_OBJECT_INDEX

Text Syntax: `vObjIndex`

Common Name: Object Index Register

Number of Components per Register: 1

Description:

- Vertex shader import for the ordered index for the current vertex processed (ordered vertex shader instance).
- Pixel shader import for the ordered index for the current pixel processed (ordered pixel shader instance).
- This value starts at 0 and is incremented for each successive pixel/vertex.
- The first component of this register contains the current vertex processed.
- This register cannot be used with relative addressing.
- This is a read-only register.
- It is an error to use the second, third, and fourth components register.

4.3 BARYCENTRIC_COORD

Enum: IL_REGTYPE_BARYCENTRIC_COORD

Text Syntax: `vBaryCoord`

Common Name: HOS Barycentric Coordinates

Number of Components per Register: 4

Description:

- This register type is valid only for HOS rendering.

- For non-HOS rendering, this register type is invalid and its contents are undefined.
- For HOS rendering, this is a vertex shader import for the barycentric coordinates of the current, tessellated vertex.
- This register cannot be used with relative addressing.
- This is a read-only register.
- It is an error to use this register in a pixel shader.

4.4 PRIMITIVE_INDEX

Enum: IL_REGTYPE_PRIMITIVE_INDEX

Text Syntax: `vPrimIndex`

Common Name: HOS Primitive Index

Number of Components per Register: 4

Description:

- This register type is valid only for HOS rendering.
- For normal, non-HOS rendering, this register type is invalid, and its contents are undefined.
- For HOS rendering, this is the incremental index of current tessellation primitive generated by the tessellation engine for certain types of HOS rendering.
- This register cannot be used with relative addressing.
- This is a read-only register.
- It is an error to use this register in a pixel shader.

4.5 QUAD_INDEX

Enum: IL_REGTYPE_QUAD_INDEX

Text Syntax: `vQuadIndex`

Common Name: HOS QUAD Index

Number of Components per Register: 4

Description:

- This register type is valid only for HOS rendering.
- For normal, non-HOS rendering, this register type is invalid and its contents are undefined.
- For HOS rendering, the first component is a quad index generated by the tessellation engine for certain types of HOS rendering.

- This register cannot be used with relative addressing.
- This is a read-only register. It is an error to use this register in a pixel shader.

4.6 SPRITE

Enum: IL_REGTYPE_SPRITE

Text Syntax: `oSprite`

Common Name: Sprite Register

Number of Components per Register: 1

Description:

- Vertex shader export for point size.
- The first component contains the point size.
- This is a write-only register. This register cannot be the source of an instruction.
- This register cannot be used with relative addressing.
- It is an error to use this register if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_POINTSIZE.
- It is an error to use this register in a pixel shader.
- The second, third and fourth components of this register are undefined.

4.7 POS

Enum: IL_REGTYPE_POS

Text Syntax: oPos

Common Name: Export Position Register

Number of Components per Register: 4

Description:

- Vertex shader export for position data.
- The position of the vertex in clip space.
- This is a write-only register. This register cannot be the source of an instruction.
- It is an error to use this register if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_POS.
- This register cannot be used with relative addressing.

- It is an error to use this register in a pixel shader.

4.8 PINPUT

Enum: IL_REGTYPE_PINPUT

Text Syntax: `vPixIn#`

Common Name: Pixel Shader Input Register

Number of Components per Register: 4

Number Per Shader: 16

- Pixel shader input data.
- It is an error to use this register in a vertex shader.
- These are read and write registers.
- This register can be used with loop-relative addressing as a source only.
- A DCLPIN or DCLPP instruction must be issued on a register of this type before it is used.
- It is an error to use this register if an INTERP, TEXCOORD, PRICOLOR, SECCOLOR, or FOG register is used.

4.9 VOUTPUT and OUTPUT

Enum: IL_REGTYPE_VOUTPUT

Text Syntax: `oVtxOut#`

Common Name: Vertex Shader Output Register

Number of Components per Register: 4

Number Per Shader: 18

Description:

- Vertex shader output data.
- These are write-only registers. They cannot be the source of an instruction.
- It is an error to use this register in a pixel shader.
- This register can be used with loop-relative addressing as a destination only.
- A DCLVOUT instruction must be issued on a register of this type before it is used.
- It is an error to use this register if a POS, SPRITE, INTERP, TEXCOORD, PRICOLOR, SECCOLOR, or FOG register is used.

4.10 LITERAL

Enum: IL_REGTYPE_LITERAL

Text Syntax: `l#` where # is the literal register number.

Common Name: Literal Constant Register

Number of Components per Register: 4

Description:

This four-components, constant, typeless, read-only register can be used in place of a GPR. The format of this register can be either integer, floating point, or four-byte hex value.

4.11 INTERP

Enum: IL_REGTYPE_INTERP

Text Syntax (VS): `oInterp#`

Text Syntax (PS): `vInterp#`

Common Name: General-Purpose Interpolator Register

Number of Components per Register: 4

Description:

- General-purpose vertex shader export and pixel shader import for interpolated data.
- Perspective correct interpolation is performed on the values of these registers when passed from the vertex shader to the pixel shader.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- It is an error to use a register of this type in a vertex shader if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on an VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_GENERIC.
- It is an error to use a register of this type in a pixel shader if a PINPUT register is used. You can achieve similar functionality by using the DCLPIN instruction on a PINPUT register and declaring its usage as IL_IMPORTUSAGE_GENERIC.
- This register can be used only with loop relative addressing.
- In a vertex shader, these are write-only registers. They cannot be the source of an instruction.
- In a pixel shader, these are read-only registers. They cannot be the destination of an instruction.

4.12 TEXCOORD

Enum: IL_REGTYPE_TEXCOORD

Text Syntax (VS): oT#

Text Syntax (PS): vT#

Common Name: Texture Coordinate Interpolator Register

Number of Components per Register: 4

Description:

- Vertex shader export and pixel shader import interpolated for texture coordinate data.
- Perspective correct interpolation is performed on the values of these registers when passed from the vertex shader to the pixel shader.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- It is an error to use a register of this type in a vertex shader if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_GENERIC.
- It is an error to use a register of this type in a pixel shader if a PINPUT register is used. You can achieve similar functionality by using the DCLPIN instruction on a PINPUT register and declaring its usage as IL_IMPORTUSAGE_GENERIC.
- In a vertex shader, these are write-only registers. They cannot be the source of an instruction.
- In a pixel shader, these are read and write registers.
- In a vertex shader, this register can be used with loop-relative addressing as a destination only.
- In a pixel shader, this register can be used with loop-relative addressing as a source only.

4.13 PRICOLOR

Enum: IL_REGTYPE_PRICOLOR

Text Syntax (VS): oPriColor#

Text Syntax (PS): vPriColor#

Common Name: Primary Color Interpolator Register

Number of Components per Register: 4

Description:

- Vertex shader export and pixel shader import for interpolated primary color data.
- By convention register number 0 represents the front-facing color, while register number 1 represents the back-facing color.
- When flat shading is used (AS_SHADE_MODE is set to FLAT), no interpolation is performed on the values of these registers when passed from the vertex shader to the pixel shader. Instead, only the value of the provoking vertex is passed to the pixel shader. When smooth shading is used (AS_SHADE_MODE is set to SMOOTH), perspective correct interpolation is performed on the values.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- This register cannot be used with relative addressing.
- It is an error to use a register of this type in a vertex shader if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_COLOR or IL_IMPORTUSAGE_BACKCOLOR.
- It is an error to use a register of this type in a pixel shader if a PINPUT register is used. You can achieve similar functionality by using the DCLPIN instruction on a PINPUT register and declaring its usage as IL_IMPORTUSAGE_COLOR or IL_IMPORTUSAGE_BACKCOLOR.
- In a vertex shader, these are write-only registers. They cannot be the source of an instruction.
- In a pixel shader, these are read and write registers.

4.14 SECCOLOR

Enum: IL_REGTYPE_SECCOLOR

Text Syntax (VS): oSecColor#

Text Syntax (PS): vSecColor#

Common Name: Secondary Color Interpolator Register

Number of Components per Register: 4

Description:

- Vertex shader export and pixel shader import for interpolated secondary color data.
- By convention, register number 0 represents the front-facing color, while register number 1 represents the back-facing color.
- When flat shading is used (AS_SHADE_MODE is set to FLAT), no interpolation is performed on the values of these registers when passed from the vertex shader to the pixel shader. Instead, only the value of the provoking vertex is passed to the pixel shader. When smooth shading is used

(AS_SHADE_MODE is set to SMOOTH), perspective correct interpolation is performed on the values.

- This register cannot be used with relative addressing.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- It is an error to use a register of this type in a vertex shader if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_COLOR or IL_IMPORTUSAGE_BACKCOLOR.
- It is an error to use a register of this type in a pixel shader if a PINPUT register is used. You can achieve similar functionality by using the DCLPIN instruction on a PINPUT register and declaring its usage as IL_IMPORTUSAGE_COLOR or IL_IMPORTUSAGE_BACKCOLOR.
- In a vertex shader, these are write-only registers. They cannot be the source of an instruction.
- In a pixel shader, these are read and write registers.

4.15 PSOUTFOG

Enum: IL_REGTYPE_PS_OUT_FOG

Text Syntax: oPsFog

Common Name: Pixel Shader Fog Output

Number of Components per Register: 1

Description:

- Pixel shader output for fog factor.
- The first component contains a fog factor.
- The second, third, and fourth components of this register are ignored.
- This is a write-only register.
- This register cannot be used with relative addressing.
- It is an error to use this register in a vertex or geometry shader.

4.16 FOG

Enum: IL_REGTYPE_FOG

Text Syntax (VS): oFog

Text Syntax (PS): vFog

Common Name: Fog Interpolator Register

Number of Components per Register: 1

Description:

- Vertex shader export and pixel shader import for interpolated fog data.
- This is a scalar register where the value is contained in the first component.
- In a vertex shader, the second, third, and fourth components must be masked (cannot be written to).
- In a pixel shader the second, third, and fourth components are undefined.
- Perspective correct interpolation is performed on the values of this registers when passed from the vertex shader to the pixel shader.
- Can only use a total of 16 of INTERP, TEXCOORD, PRICOLOR, SECCOLOR, and FOG registers in a single shader.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- This register cannot be used with relative addressing.
- It is an error to use a register of this type in a vertex shader if a VOUTPUT register is used. You can achieve similar functionality by using the DCLVOUT instruction on a VOUTPUT register and declaring its usage as IL_IMPORTUSAGE_FOG.
- It is an error to use a register of this type in a pixel shader if a PINPUT register is used. You can achieve similar functionality by using the DCLPIN instruction on a PINPUT register and declaring its usage as IL_IMPORTUSAGE_FOG.
- In a vertex shader, this is a write-only register. It cannot be the source of an instruction.
- In a pixel shader, this is a read-only register. It cannot be the destination of an instruction.

4.17 SPRITECOORD

Enum: IL_REGTYPE_SPRITECOORD

Text Syntax: `vSpriteCoord`

Common Name: Sprite Texture Coordinate Register

Number of Components per Register: 2

Description:

- Pixel shader input for sprite texture coordinate.
- The first and second components contain the pixel's S and T coordinate for the point primitive rendered.
- The values of this register are undefined if the primitive rendered is not a point.
- The third and fourth components of this register are undefined.
- This is a read-only register. It cannot be the destination of an instruction.
- This register cannot be used with relative addressing.
- It is an error to use this register in a vertex shader.

4.18 PRIMCOORD

Enum: IL_REGTYPE_PRIMCOORD

Text Syntax: `vPrimCoord`

Number of Components per Register: 2

Description:

- This is a graphics-only feature.
- Pixel shader input for point-aa or line-aa texture coordinates.
- SpriteCoord cannot be used in any shader that uses primcoord.
- The first and second components contain the pixel's S and T coordinate for the point/line primitive rendered.
- The values of this register are undefined if the primitive rendered is not a point or a line.
- The third and fourth components of this register are undefined.
- This is a read-only register. It cannot be the destination of an instruction.
- This register cannot be used with relative addressing.
- It is an error to use this register in a vertex or geometry shader.

4.19 PRIMTYPE

Enum: IL_REGTYPE_PRIMTYPE

Text Syntax: `vPrimType`

Number of Components per Register: 2

Description:

- This is a graphics only feature.
- It is a pixel shader input for a primitive type.
- The first component has sign bit = 1, this is a point. Values in other bits are undefined.
- The second component has sign bit = 1, this is a line. Values in other bits are undefined.
- The third and fourth components of this register are undefined.
- This is a read-only register. It cannot be the destination of an instruction.
- This register cannot be used with relative addressing.
- It is an error to use this register in a vertex or geometry shader.

4.20 vWinCoord

Enum: IL_REGTYPE_WINCOORD

Text Syntax: `vWinCoord.xy`

Common Name: Window Coordinate Register

Number of Components per Register: 2

Description:

- Pixel shader import for screen position data.
- The first and second components are the X and Y position of the pixel in the domain of execution. The third component is the Z coordinate of the pixel in window space.
- The fourth component is W.
- A DCLPI instruction must be issued on this shader type before is it used in any other instruction.
- The DCLPI instruction specifies whether the X and Y coordinate is relative to the lower-left or upper-left corner of the window and whether it represents the center or upper-left corner of the pixel.

This read-only register cannot be the destination of an instruction. It is an error to use this register in a vertex shader. This register cannot be used with relative

addressing. The second, third, and fourth components of this register are undefined. This register is not valid in compute shaders.

4.21 FACE

Enum: IL_REGTYPE_FACE

Text Syntax: vFace

Common Name: Face Register

Number of Components per Register: 1

Description:

- Pixel shader import for primitive facing.
- The x component is negative if the pixel is the back-face of the primitive. The x component is positive if the pixel is the front-face of the primitive.
- Point and Line primitives are always front-facing. Points and lines rendered as a result of polygons using point or line fill mode inherit the facing of the polygon.
- This is a read-only register. It cannot be the destination of an instruction.
- It is an error to use this register in a vertex shader.
- It is an error to use this register in a real-time pixel shader.
- This register cannot be used with relative addressing.
- The second, third, and fourth components of this register are undefined.

4.22 PCOLOR

Enum: IL_REGTYPE_PCOLOR

Text Syntax: oC#

Common Name: Pixel Color Register

Number of Components per Register: 4

Description:

- Pixel shader export for color data.
- The register number corresponds to the color buffer to which data is output.
- These are write-only registers. They cannot be the source of an instruction.
- It is an error to use this register in a vertex shader.
- This register cannot be used with relative addressing.

4.23 DEPTH

Enum: IL_REGTYPE_DEPTH

Text Syntax: `oDepth`

Common Name: Pixel Depth Register

Number of Components per Register: 1

Description:

- Pixel shader export for depth data.
- This is a scalar register where the depth values is contained in the first component.
- This is a write-only register. It cannot be the source of an instruction.
- It is an error to use this register in a vertex shader.
- This register cannot be used with relative addressing.
- The second, third, and fourth components of this register are unused and undefined.

4.24 Mask

Enum: IL_REGTYPE_OMASK

Text Syntax: `oMask`

Common Name: Pixel Coverage Mask

Number of Components per Register: 1

Description:

When the pixel shader runs at sample-frequency, the coverage mask is ANDed with a mask that selects the sample currently being processed. As a result, sample N is always masked by bit N of oMask. This allows a shader to run at either sample-frequency or pixel-frequency with identical oMask behavior. The same rule applies to Alpha and to Coverage when the shader runs at sample-frequency.

- This is a scalar register where the mask value is contained in the first component.
- Values assigned to oMask are treated as integer.
- This is a write-only register. It cannot be the source of an instruction.
- It is an error to use this register in a vertex shader.
- This register cannot be used with relative addressing.
- The second, third, and fourth components of this register are unused and undefined.

4.25 Stencil

Enum: IL_REGTYPE_STENCIL

Text Syntax: oSTENCIL

Common Name: STENCIL ref value

Number of Components per Register: 1

Description:

- This is a scalar register where the stencil value is contained in the first component.
- Values assigned to Stencil are treated as integer.
- This is a write-only register. It cannot be the source of an instruction.
- It is an error to use this register in a vertex shader.
- This register cannot be used with relative addressing.
- The second, third, and fourth components of this register are unused and undefined.

4.26 Global

Enum: IL_REGTYPE_GLOBAL

Text Syntax: g[address]
Note that the address must be a single-component integer.

Common Name: Global Memory Register

Number of Components per Register: 4

Description:

This is a read/write register that can be used to address global memory in the text form of the shader. Global variables are indicated by a g[*address*].

For example:

```
add g[2].x, r4, g[4]
```

reads address 4, adds r4 to the contents of that address, and scatters the result to address 2. Each address corresponds to a 128-bit, four dword location. Most address locations are indexed.

Global g registers can be indexed using the temp (r) registers. For example, `add g [r5.x].x r4, g[r6.y]`

4.27 Shared Temp

Enum: IL_REGTYPE_SHARED_TEMP

Text Syntax: `sr#` (where # is any shared register number)

Common Name: Shared General-Purpose Register

Number of Components per Register: 4

Description:

This read/write register is shared by all wavefronts running on a SIMD. Only absolute address mode is allowed; for example: `sr2`. It is used only in compute shaders.

For example:

```
iadd sr2.xy_w, sr2.xyzw, r0.xxxx
```

4.28 Thread_ID

Enum: IL_REGTYPE_THREAD_ID

Text Syntax: `vTid`

Common Name: Input Register for a Thread ID Within a Block

Number of Components per Register: 1

Description:

This read-only input register contains the thread ID within a thread block. It is used only in compute shaders as an index or in integer operations.

For example:

```
mov r1.x___,  vTid.x
mov g[vTid.x], r1.x
```

4.29 Absolute Thread_ID

Enum: IL_REGTYPE_ABSOLUTE_THREAD_ID

Text Syntax: vaTid

Common Name: Input Register for Absolute Thread ID

Number of Components per Register: 1

Description:

This read-only input register contains an absolute thread ID. It is used only in compute shaders. It is used as an index or in integer operations.

For example:

```
mov r1.x___,  vaTid.x
mov g[vaTid.x], r1.x
```

4.30 Thread_Group_ID

Enum: IL_REGTYPE_THREAD_GROUP_ID

Text Syntax: vTGroupid

Common Name: Input Register for Thread Group ID

Number of Components per Register: 1

Description:

This is a read-only input register that contains the thread group ID. It is used only in compute shaders. It is used as an index or in integer operations.

For example:

```
mov r1.x___,  vTGroupid.x
mov g[vTGroupid.x], r1.x
```

4.31 Generic_Memory

Enum: IL_REGTYPE_GENERIC_MEM

Text Syntax: `mem`

Common Name: Generic Memory Type

Number of Components per Register: 4

Description:

This is register provides a mask or swizzle. It is used by instructions like write_lds_vec, which does not have a dst register but requires a dst mask.

For example:

```
write_lds_vec mem.x_z_, r0.xyzw
```

Chapter 5
Enumerated Types

This chapter lists and briefly describes the ATI IL enumerated types.

5.1 ILShader

Table 5.1 ILShader Enumeration Types

Enumeration	Description
IL_SHADER_PIXEL	This code describes a kernel that uses a thread creation pattern optimized for graphics (pixel kernel).
IL_SHADER_VERTEX	This code describes a vertex shader.
IL_SHADER_GEOMETRY	This code describes a geometry shader.

5.2 ILLanguageType

Table 5.2 ILLanguageType Enumeration Types

Enumeration	Description
IL_LANG_DX10_GS	
IL_LANG_DX10_PS	
IL_LANG_DX10_VS	
IL_LANG_DX8_PS	DX 1.x shader (DX 8)
IL_LANG_DX8_VS	DX vertex shader (DX 8)
IL_LANG_DX9_PS	
IL_LANG_DX9_VS	
IL_LANG_GENERIC	
IL_LANG_OPENGL	

5.3 ILRegType

See Chapter 4, “Register Types,” for more information.

Table 5.3 ILRegType Enumeration Types

Register Type	Description
IL_REGTYEP_OBJECT_INDEX	
IL_REGTYPE_ADDR	
IL_REGTYPE_BARYCENTRIC_COORD	
IL_REGTYPE_CLIP	
IL_REGTYPE_CONST_BOOL	
IL_REGTYPE_CONST_BUFF	A four component element of a constant buffer.
IL_REGTYPE_CONST_FLOAT	
IL_REGTYPE_CONST_INT	
IL_REGTYPE_DEPTH	
IL_REGTYPE_FACE	
IL_REGTYPE_FOG	
IL_REGTYPE_INDEX	
IL_REGTYPE_INTERP	
IL_REGTYPE_LITERAL	A four component register containing explicit values.
IL_REGTYPE_PCOLOR	
IL_REGTYPE_PINPUT	
IL_REGTYPE_POS	
IL_REGTYPE_PRICOLOR	
IL_REGTYPE_PRIMITIVE_INDEX	
IL_REGTYPE_QUAD_INDEX	
IL_REGTYPE_SECCOLOR	
IL_REGTYPE_SPRITE	
IL_REGTYPE_SPRITECOORD	
IL_REGTYPE_TEMP	A four component virtual register.
IL_REGTYPE_TEXCOORD	
IL_REGTYPE_VERTEX	
IL_REGTYPE_VOUTPUT	
IL_REGTYPE_VPRIM	
IL_REGTYPE_WINCOORD	A Windows coordinate register.

5.4 ILTopologyType

Table 5.4 IL_TOPOLOGY Enumeration Types (Input to a Geometry Shader)

Enumeration	Description
IL_TOPOLOGY_LINE	
IL_TOPOLOGY_LINE_ADJ	
IL_TOPOLOGY_POINT	
IL_TOPOLOGY_TRIANGLE	
IL_TOPOLOGY_TRIANGLE_ADJ	

Table 5.5 IL_OUTPUT_TOPOLOGY Enumeration Types (Output from a Geometry Shader)

Enumeration	Description
IL_OUTPUT_TOPOLOGY_LINESTRIP	
IL_OUTPUT_TOPOLOGY_POINTLIST	
IL_OUTPUT_TOPOLOGY_TRIANGLE_STRIP	

5.5 ILMatrix

Table 5.6 ILMatrix Enumeration Types

Enumeration	IL Text
IL_MATRIX_3X2	_matrix(*3x2*)
IL_MATRIX_3X3	_matrix(*3x3*)
IL_MATRIX_3X4	_matrix(*3x4*)
IL_MATRIX_4X3	_matrix(*4x3*)
IL_MATRIX_4X4	_matrix(*4x4*)

5.6 ILComponentSelect

See Section 2.2.7, “Source Modifier Token,” page 2-6 for usage. IL Text details for component selection can be found in Chapter 3, “Text Instruction Syntax.”

Table 5.7 ILComponentSelect Enumeration Types

Enumeration	Description
IL_COMPSEL_0	Force this component to 0.0.
IL_COMPSEL_1	Force this component to 1.0.
IL_COMPSEL_W_A	Select the fourth component (w/alpha) for the component.
IL_COMPSEL_X_R	Select the first component (x/red) for the component.
IL_COMPSEL_Y_G	Select the second component (y/green) for the component.
IL_COMPSEL_Z_B	Select the third component (z/blue) for the component.

5.7 ILModDstComp

See Section 3.5, “Destination Modifiers,” page 3-3 for usage. IL Text details for write mask can be found in Section 3.6, “Write Mask,” page 3-3.

Table 5.8 ILModDstComp Enumeration Types

Enumeration	Description
IL_MODCOMP_0	Write 0.0 to this component.
IL_MODCOMP_1	Write 1.0 to this component.
IL_MODCOMP_NOWRITE	Do not write to this component. The contents of this component of the destination register are not changed.
IL_MODCOMP_WRITE	Write the result of the instruction to this component.

5.8 ILImportUsage

See the DCLVOUT and DCLPIN instructions for more information:

Table 5.9 ILImportUsage Enumeration Types

Enumeration	Text Syntax	Description
IL_IMPORTUSAGE_BACKCOLOR	_usage(*backcolor*)	When used to declare a vertex shader output, this usage indicates the register contains a color value to be interpolated across the primitive and passed to the pixel shader once the shader terminates. The processed value can be read from the PINPUT register declared with this usage and the matching *usageIndex.* When used to declare a pixel shader input, this usage indicates the register contains a color value that has been interpolated across the primitive. The value originates from the VOUTPUT register in the vertex shader declared with this *usage* and matching *usageIndex*. Hardware can use lower-precision interpolators for colors.
IL_IMPORTUSAGE_COLOR	_usage(*color*)	When used to declare a vertex shader output, this usage indicates the register contains a color value to be interpolated across the primitive and passed to the pixel shader once the shader terminates. The processed value can be read from the PINPUT register declared with this usage and the matching *usageIndex.* When used to declare a pixel shader input, this usage indicates the register contains a color value that has been interpolated across the primitive. The value originates from the VOUTPUT register in the vertex shader declared with this *usage* and matching *usageIndex*. Hardware can use lower-precision interpolators for colors.

Table 5.9 ILImportUsage Enumeration Types (Cont.)

Enumeration	Text Syntax	Description
IL_IMPORTUSAGE_FOG	_usage(*fog*)	When used to declare a vertex shader output, this usage indicates that the x component of the register has the vertices' fog value at shader termination. The value is interpolated across the primitive and passed to the pixel shader. The processed value can be read in the pixel shader from the PINPUT register declared with this *usage*. When used to declare a pixel shader input, indicates that the x component of the register contains the fog value at shader execution. The value of the VOUTPUT register in the vertex shader declared with the DCLVOUT instruction as having this *usage* is interpolated across the primitive and passed to the PINPUT register declared with this *usage*. For any register declared with this *usage*, the *usageIndex* must be set to 0.
IL_IMPORTUSAGE_GENERIC	_usage(*generic*)	When used to declare a vertex shader output, this usage indicates the register contains a generic value to be interpolated across the primitive and passed to the pixel shader once the shader terminates. The processed value can be read from the PINPUT register declared with this usage and the matching *usageIndex.* When used to declare a pixel shader input, this usage indicates the register contains a generic value that has been interpolated across the primitive. The value originates from the VOUTPUT register in the vertex shader declared with this *usage* and matching *usageIndex*.
IL_IMPORTUSAGE_POINTSIZE	_usage(*pointsize*)	When used to declare a vertex shader output, this usage indicates the x component of the register contains the vertices' point size when the shader terminates. *usageIndex* must be zero when this *usage* is set. Only VOUTPUT register 1 can have this *usage*. This *usage* cannot be used in a pixel shader input.
IL_IMPORTUSAGE_POS	_usage(*pos*)	When used to declare a vertex shader output, this usage indicates the register contains the vertices' position when the shader terminates. *usageIndex* must be zero when this *usage* is set. Only VOUTPUT register 0 can have this *usage*. This *usage* cannot be used in a pixel shader input.
IL_IMPORTUSAGE_WINCOORD	_usage(*wincoord*)	Can only be used in a pixel shader. The x, y, z, w values correspond to the screen position. This has been added to IL2.0 for OpenGL support.

5.9 ILImportComponent

See the “DCLPI”, “DCLPIN”, “DCLV”, and “DCLVOUT” instructions for usage.

Table 5.10 ILImportComponent Enumeration Types

Enumeration	Text Syntax	Description
IL_IMPORTSEL_DEFAULT0	_<*comp*>(0) where <*comp*> is *x, y, z, or w*	This component is enabled and can be used in a vertex shader. If this register or this component of this register is not written to, the component defaults to 0.0 when used as a source or when the shader terminates. In a pixel shader, if this register or this component of this register is not exported in the vertex shader, the component is set to 0.0. If used in with the DCLVOUT instruction in the vertex shader, this component will default to 0.0 if it is not written to. The component is considered to be exported in the vertex shader in this case; thus, any component in a pixel shader mapped to this component will be set to 0.0 regardless of its default value set by the DCLPIN instruction.
IL_IMPORTSEL_DEFAULT1	_<*comp*>(1) where <*comp*> is *x, y, z, or w*	This component is enabled and can be used. In a vertex shader, if this register or this component of this register is not written to, the component defaults to 1.0 when used as a source or when the shader terminates. In a pixel shader, if this register or this component of this register is not exported in the vertex shader, set the component to 1.0. If used with the DCLVOUT instruction in the vertex shader, this component defaults to 1.0 if it’s not written to. The component is considered to be exported in the vertex shader in this case; thus, any component in a pixel shader mapped to this component is set to 1.0, regardless of its default value set by the DCLPIN instruction.
IL_IMPORTSEL_UNDEFINED	_<*comp*>(*) where <*comp*> is *x, y, z, or w*	This component is enabled and can be used. If this register or this component of the register is not exported in the vertex shader, the value of the component is undefined (the value of the component does not matter).
IL_IMPORTSEL_UNUSED	_<*comp*>(-) where <*comp*> is *x, y, z, or w* Example: `dclpi_z(-)_w(-)`	This component is disabled and cannot be used in the shader. It is an error to reference the component in the shader.

5.10 ILDefaultVal

See the DCLDEF instruction for usage.

Table 5.11 ILDefaultVal Enumeration Types

Enumeration	Text Syntax	Description
IL_DEFVAL _0	_<*comp*>(0) where <*comp*> is *x, y, z, or w*	Indicates that the default value for this component of the register type is 0.0.
IL_DEFVAL_1	_<*comp*>(1) where <*comp*> is *x, y, z, or w*	Indicates that the default value for this component of the register type is 1.0.
IL_DEFVAL_NONE	_<*comp*>(*) where <*comp*> is *x, y, z, or w* Example: `dcldef_z(*)_w(*)`	Indicates there is no default value for this component of the register type.

5.11 ILShiftScale

See Section 3.5, “Destination Modifiers,” page 3-3 for usage.

Table 5.12 ILShiftScale Enumeration Types

Enumeration	Text Syntax	Description
IL_SHIFT_D2	_d2	Shift value right by 1 bit (divide by 2).
IL_SHIFT_D4	_d4	Shift value right by 2 bits (divide by 4).
IL_SHIFT_D8	_d8	Shift value right by 3 bits (divide by 8).
IL_SHIFT_NONE		Do not shift.
IL_SHIFT_X2	_x2	Shift value left by 1 bit (multiply by 2).
IL_SHIFT_X4	_x4	Shift value left by 2 bits (multiply by 4).
IL_SHIFT_X8	_x8	Shift value left by 3 bits (multiply by 8).

5.12 ILDivComp

See Section 3.7, “Source Modifiers,” page 3-4 for usage.

Table 5.13 ILDivComp Enumeration Types

Enumeration	Text Syntax	Description
IL_DIVCOMP_NONE	_divcomp(*none*)	None
IL_DIVCOMP_UNKNOWN	_divcomp(*unknown*)	The component used to divide is not known at shader create time. The component is determined by a state setting at shader run time. This value can only be used in a TEXLD, TEXLDD, TEXLDB, TEXLDMS, TEXWEIGHT, PROJECT, or LOD instruction.
IL_DIVCOMP_W	_divcomp(*w*)	Divide the first, second, and third components by the fourth component. If the fourth component is 0.0, the first, second, and third component become ±FLT_MAX.
IL_DIVCOMP_Y	_divcomp(*y*)	Divide the first component by the second component. If the second component is 0.0, the first component becomes ±FLT_MAX.
IL_DIVCOMP_Z	_divcomp(*z*)	Divide the first and second components by the third component. If the third component is 0.0, the first and second component become ±FLT_MAX.

5.13 ILRelOp

See IFC, CONTINUEC, BREAKC, CMP, and SET for usage.

Table 5.14 ILRelOp Enumeration Types

Enumeration	Text Syntax	Description
IL_RELOP_EQ	_relop(*eq*)	Equal.
IL_RELOP_GE	_relop(*ge*)	Greater than or equal.
IL_RELOP_GT	_relop(*gt*)	Greater than.
IL_RELOP_LE	_relop(*le*)	Less than or equal.
IL_RELOP_LT	_relop(*lt*)	Less than.
IL_RELOP_NE	_relop(*ne*)	Not equal.

5.14 ILLogicOp

Can be used in IF_LOGICAL*, CONTINUE_LOGICAL*, and BREAK_LOGICAL*.

Table 5.15 ILLogicOp Enumeration Types

Enumeration	Text Syntax	Description
IL_LOG_EQ	_log(*eq*)	Equal (32-bit integer compare)
IL_LOG_NE	_log(*ne*)	Not equal (32-bit integer compare)

5.15 ILZeroOp

See the RSQ, RCP, LOG, LOGP, LN, NRM, and DIV instructions for more information:.

Table 5.16 ILZeroOp Enumeration Types

Enumeration	Text Syntax	Description
IL_ZEROOP_0	_zeroop(*zero*)	Return 0.0 when the instruction operates on 0.0. Cannot be used with LOG, LOGP, or LN instructions.
IL_ZEROOP_FLTMAX	_zeroop(*fltmax*)	Return the max floating point value when the instruction operates on 0.0.
IL_ZEROOP_INF_ELSE_MAX	_zeroop(*inf_else_max*)	Implementation-dependant: Return IEEE infinity if the implementation can support it; otherwise, use FLT_MAX when the instruction operates on 0.0.
IL_ZEROOP_INFINITY	_zeroop(*infinity*)	Return IEEE Infinity when the instruction operates on 0.0.

5.16 ILCmpValue

See the CMP Instruction for usage.

Table 5.17 ILCmpValue Enumeration Types

Enumeration	Text Syntax	Description
IL_CMPVAL_0_0	_cmpval(*0.0*)	Compare vs. 0.0.
IL_CMPVAL_0_5	_cmpval(*0.5*)	Compare vs. 0.5.
IL_CMPVAL_1_0	_cmpval(*1.0*)	Compare vs. 1.0.
IL_CMPVAL_NEG_0_5	_cmpval(*-0.5*)	Compare vs. -0.5.
IL_CMPVAL_NEG_1_0	_cmpval(*-1.0*)	Compare vs. -1.0.

5.17 ILTexCoordMode

See DCLPT instruction for more information:

Table 5.18 ILTexCoordMode Enumeration Types

Enumeration	Text Syntax	Description
IL_TEXCOORDMODE_NORMALIZED	_coordmode(*normalized*)	The texture coordinates given in the texture load instructions are non-parametric. (the coordinate range [0.0-1.0] spans the entire texture)
IL_TEXCOORDMODE_UNKNOWN	_coordmode(*unknown*)	At shader create time, it is not known if the texture coordinates given in the texture load instructions are normalized. Instead, this is determined at shader run time based on a state value.
IL_TEXCOORDMODE_UNNORMALIZED	_coordmode(*unnormalized*)	The texture coordinates given in the texture load instructions are para-metric. (the coordinate range [0.0, dimension of the texture] spans the entire dimension of the texture)

5.18 ILPixTexUsage

There are a maximum of eight values. See DCLPT instruction for more information.

Table 5.19 ILPixTexUsage Enumeration Types

Enumeration	Text Syntax
IL_USAGE_PIXTEX_1D	_type(*1d*)
IL_USAGE_PIXTEX_2D	_type(*2d*)
IL_USAGE_PIXTEX_2DMS_ARRAY	_type(*2dms_array*)
IL_USAGE_PIXTEX_2DMSAA	_type(*2dmsaa*)
IL_USAGE_PIXTEX_3D	_type(*3d*)
IL_USAGE_PIXTEX_CUBEMAP	_type(*cubemap*)
IL_USAGE_PIXTEX_CUBEMAPARRAY	_type(*cubemaparray*)
IL_USAGE_PIXTEX_UNKNOWN	_type(*unknown*)

5.19 ILTexShadowMode

See the TEXLD (page 6-67), TEXLDB (page 6-70), and TEXLDD (page 6-74) instructions for more information.

Table 5.20 ILTexShadowMode Enumeration Types

Enumeration	Text Value	Description
IL_TEXSHADOWMODE_NEVER	_shadowmode(*never*)	Never do a shadow map comparison.
IL_TEXSHADOWMODE_UNKNOWN	_shadowmode(*unknown*)	Do a compare vs. the third component of the texture coordinate.
IL_TEXSHADOWMODE_Z	_shadowmode(*z*)	Always do a compare vs. the third component of the texture coordinate.

5.20 ILTexFilterMode

See the TEXLD, TEXLDB, and TEXLDD instructions for more information.

Table 5.21 ILTexFilterMode Enumeration Types

Enumeration	Text Syntax	Description
IL_TEXFILTER_ANISO	_<filter>(*aniso*) where <filter> is min, mag, volmag, or volmin	Always use anisotropic filtering if aniso filtering is enabled in state or based on the aniso parameter in the instruction.
IL_TEXFILTER_LINEAR	_<filter>(*linear*) where <filter> is min, mag, volmag, or volmin	Always use linear filtering.
IL_TEXFILTER_POINT	_<filter>(*point*) where <filter> is min, mag, volmag, or volmin	Always use point filtering.
IL_TEXFILTER_UNKNOWN	_<filter>(*unknown*) where <filter> is min, mag, volmag, or volmin	Use the filtering mode specified by state.

5.21 ILAnisoFilterMode

See the TEXLD, TEXLDB, and TEXLDD instructions for more information.

Table 5.22 ILAnisoFilterMode Enumeration Types

Enumeration	Text Syntax	Description
IL_ANISOFILTER_DISABLED	_aniso(*disabled*)	Never use anisotropic filtering.
IL_ANISOFILTER_MAX_1_TO_1	_aniso(*1*)	Use a maximum of 1 sample for anisotropic filtering.
IL_ANISOFILTER_MAX_16_TO_1	_aniso(*16*)	Use a maximum of 16 samples for anisotropic filtering.
IL_ANISOFILTER_MAX_2_TO_1	_aniso(*2*)	Use a maximum of 2 samples for anisotropic filtering.
IL_ANISOFILTER_MAX_4_TO_1	_aniso(*4*)	Use a maximum of 4 samples for anisotropic filtering.
IL_ANISOFILTER_MAX_8_TO_1	_aniso(*8*)	Use a maximum of 8 samples for anisotropic filtering.
IL_ANISOFILTER_UNKNOWN	_aniso(*unknown*)	Use the anisotropic filtering mode specified by state.

5.22 ILMipFilterMode

See the TEXLD, TEXLDB, and TEXLDD instructions for more information.

Table 5.23 ILMipFilterMode Enumeration Types

Enumeration	Text Syntax	Description
IL_MIPFILTER_BASE	_mip(*base*)	Always sample the base mipmap only.
IL_MIPFILTER_LINEAR	_mip(*linear*)	Always use linear filtering across mipmaps.
IL_MIPFILTER_POINT	_mip(*point*)	Always sample the nearest mipmap only.
IL_MIPFILTER_UNKNOWN	_mip(*unknown*)	Use the filtering mode specified by state.

5.23 ILNoiseType

Table 5.24 ILNoiseType Enumeration Types

Enumeration	Text Syntax	Description
IL_NOISETYPE_PERLIN1D	_type(*perlin1D*)	Compute 1D Perlin noise function
IL_NOISETYPE_PERLIN2D	_type(*perlin2D*)	Compute 2D Perlin noise function
IL_NOISETYPE_PERLIN3D	_type(*perlin3D*)	Compute 3D Perlin noise function
IL_NOISETYPE_PERLIN4D	_type(*perlin4D*)	Compute 4D Perlin noise function

5.24 ILInterpolation

Table 5.25 ILInterpolation Enumeration Types

Enumeration	Text Syntax
IL_INTERPMODE_CONSTANT	_interp(*constant*)
IL_INTERPMODE_LINEAR	_interp(*linear*)
IL_INTERPMODE_LINEAR_CENTROID	_interp(*centroid*)
IL_INTERPMODE_LINEAR_NOPERSPECTIVE	_interp(*noperspective*)
IL_INTERPMODE_LINEAR_NOPERSPECTIVE_CENTROID	_interp(*noper_centroid*)
IL_INTERPMODE_LINEAR_NOPERSPECTIVE_SAMPLE	_interp(*noper_sample*)
IL_INTERPMODE_LINEAR_SAMPLE	_interp(*sample*)
IL_INTERPMODE_NOTUSED = 0	_interp(*notused*)

5.25 ILAddressing

See IL_Dst and IL_Src for more information.

Table 5.26 ILAddressing Enumeration Types

Enumeration	Description
IL_ADDR_ABSOLUTE (no = 0)	Absolute addressing is used.
IL_ADDR_REG_RELATIVE	Register-relative addressing is used.
IL_ADDR_RELATIVE	Relative addressing is used.

5.26 ILElementFormat

Table 5.27 ILElementFormat Enumeration Types

Enumeration	Description
IL_ELEMENTFORMAT_FLOAT	Element uses a single-precision, floating point format.
IL_ELEMENTFORMAT_MIXED	Element uses more than one format.
IL_ELEMENTFORMAT_SINT	Element uses a signed, integer format.
IL_ELEMENTFORMAT_SNORM	Element uses a signed, normalized format.
IL_ELEMENTFORMAT_SRGB	Element uses an integer format in the sRGB color space.
IL_ELEMENTFORMAT_UINT	Element uses an unsigned, integer format.
IL_ELEMENTFORMAT_UNKNOWN	Element uses an unknown or user-defined format.
IL_ELEMENTFORMAT_UNORM	Element uses an unsigned, normalized format.

5.27 ILOpCode

Table 5.28 ILOpCode Enumeration Types

Enumeration	Description
IL_OP_ABS	Computes the absolute value of each element of a vector.
IL_DCL_CONST_BUFFER	Declares a constant buffer.
IL_DCL_INDEXED_TEMP_ARRAY	Declares an indexed array.
IL_DCL_INPUT	Declares an input register.
IL_DCL_INPUT_PRIMITIVE	Used to declare the input primitive type.
IL_DCL_LITERAL	Declares a literal value.
IL_DCL_MAX_OUTPUT_VERTEX_COUNT	Used to declare the maximum number of vertices that will be emitted by a shader
IL_DCL_ODEPTH	Used to declare that the pixel shader intends to write to its scalar output oDepth register.
IL_DCL_OUTPUT	Declares an output register.
IL_DCL_OUTPUT_TOPOLOGY	Used to declare the output topology of a primitive.
IL_DCL_PERSIST	Used to declare the amount of persistent storage used by a shader.
IL_DCL_RESOURCE	Declares an input buffer.
IL_DCL_VPRIM	Used to declare an input register.
IL_OP_ACOS	Used to compute the inverse cosine of a component.
IL_OP_ADD	Computes the single-precision, floating point addition of the values in each element of a vector to the values in the corresponding element in another vector.
IL_OP_AND	Performs a logical-AND on two vectors.
IL_OP_ASIN	Used to compute the inverse sine of a component.
IL_OP_ATAN	Used to compute the inverse tangent of a component.
IL_OP_BREAK	Unconditionally terminates a LOOP or SWITCH block.
IL_OP_BREAK_LOGICALNZ	Conditionally terminates a LOOP or SWITCH block if the parameter is not zero.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_BREAK_LOGICALZ	Conditionally terminates a LOOP or SWITCH block if the parameter is zero.
IL_OP_BREAKC	Conditionally terminates a LOOP or SWITCH block.
IL_OP_CALL	Unconditionally calls a FUNC block.
IL_OP_CALL_LOGICALNZ	Calls a FUNC block if the parameter is not zero.
IL_OP_CALL_LOGICALZ	Calls a FUNC block if the parameter is zero.
IL_OP_CALLNZ	Used to call a FUNC block if the contents of the Constant Boolean register are not zero.
IL_OP_CASE	Compares <case-value> to <switch-value>, and conditionally executes the CASE instruction block if they are equal.
IL_OP_CLAMP	
IL_OP_CLG	Computes the ceiling of the value in each element of a vector.
IL_OP_CMOV	Performs a single-precision, floating point comparison of the value in each element of a vector to 0.0f, and conditionally moves the value in each element of another vector to the corresponding element of a third vector if the comparison evaluates FALSE.
IL_OP_CMOV_LOGICAL	Performs an integer comparison of the value in each element of a vector to zero, and moves the value in each element of a second vector to the corresponding element of a third vector if the comparison evaluates TRUE; otherwise, moves the corresponding element of a fourth vector to the corresponding element of the third vector.
IL_OP_CMP	Performs a logical comparison of the value in each element of a comparison value, and conditionally moves the value in each element of another vector to the corresponding element of a third vector if the comparison evaluates TRUE; otherwise, the value in the corresponding element of a fourth vector is moved.
IL_OP_COLORCLAMP	
IL_OP_COMMENT	
IL_OP_CONTINUE	Unconditionally continues execution at the next LOOP or WHILELOOP instruction.
IL_OP_CONTINUE_LOGICALNZ	Conditionally continues execution at the next LOOP or WHILELOOP instruction if the parameter is not zero.
IL_OP_CONTINUE_LOGICALZ	Conditionally continues execution at the next LOOP or WHILELOOP instruction if the parameter is zero.
IL_OP_CONTINUEC	Conditionally continues execution at the next LOOP or WHILELOOP instruction.
IL_OP_COS	Computes the cosine of a component.
IL_OP_COS_VEC	Computes the cosine of each element of a vector.
IL_OP_CRS	Used to compute the cross product of two 3-vectors.
IL_OP_CUT	Used to close a primitive topology and open a new one.
IL_OP_D_2_F	Converts a double-precision float value to a single-precision float value.
IL_OP_D_ADD	Performs a double-precision, floating point addition of the values in each element of a vector to the values in the corresponding element in another vector.
IL_OP_D_EQ	Performs a double-precision float equality comparison.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_D_FRAC	Returns a double-precision, floating point fraction (mantissa).
IL_OP_D_FREXP	Splits a double-precision, floating point value into fraction (mantissa) and exponent values.
IL_OP_D_GE	Performs a double-precision float greater than or equal comparison.
IL_OP_D_LDEXP	Combines fraction (mantissa) and exponent values into a double-precision, floating point value.
IL_OP_D_LT	Performs a double-precision float less than comparison.
IL_OP_D_MAD	Performs a double-precision, floating point multiplication of two values, then performs a double-precision addition on the result with a third value.
IL_OP_D_MUL	Performs a double-precision, floating point multiplication of two values.
IL_OP_D_NE	Performs a double-precision float inequality comparison.
IL_OP_DCLARRAY	Used to declare a range of registers to be included in an array.
IL_OP_DCLDEF	Not used.
IL_OP_DCLPI	Used to declare the interpolator properties.
IL_OP_DCLPIN	Used to declare an input register.
IL_OP_DCLPP	Not used.
IL_OP_DCLPT	Used to declare a texture's properties.
IL_OP_DCLV	Used to declare the mapping for a vertex shader's inputs.
IL_OP_DCLVOUT	Used to declare the output register for a vertex shader.
IL_OP_DEF	Used to declare the constant integer or float value for a register.
IL_OP_DEFAULT	Starts the default instruction block within a SWITCH instruction block.
IL_OP_DEFB	Used to declare the constant Boolean value for a register.
IL_OP_DET	Used to calculate the determinant of a 4x4 matrix.
IL_OP_DISCARD_LOGICALNZ	Logically compares a parameter to zero, and conditionally stops execution and discards the results of a kernel invocation if the parameter is not zero. When used in the compute kernel, this instruction produces undefined results.
IL_OP_DISCARD_LOGICALZ	Logically compares a parameter to zero, and conditionally stops execution and discards the results of a kernel invocation if the parameter is zero. When used in the compute kernel, this instruction produces undefined results.
IL_OP_DIST	Used to compute the vector distance between two 3-vectors.
IL_OP_DIV	Performs a single-precision, floating point division of the values in each element of a vector by the values in the corresponding element in another vector.
IL_OP_DP2	Computes the dot product of two 2-vectors.
IL_OP_DP2ADD	Used to compute the dot product of two 2-vectors and scalar adds a 32-bit value to the result.
IL_OP_DP3	Computes the dot product of two 3-vectors.
IL_OP_DP4	Computes the dot product of two 4-vectors.
IL_OP_DST	

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_DSX	Used to compute the single-precision, floating point rate of change of a vector in the x-direction.
IL_OP_DSY	Used to compute the single-precision, floating point rate of change of a vector in the y-direction.
IL_OP_DXSINCOS	
IL_OP_ELSE	Starts the ELSE clause within an IF instruction block.
IL_OP_EMIT	Used to generate a primitive.
IL_OP_EMIT_THEN_CUT	Used to generate and close a primitive then start a new primitive.
IL_OP_END	Indicates the end of an IL stream.
IL_OP_ENDFUNC	Indicates the end of a FUNC instruction block.
IL_OP_ENDIF	Indicates the end of an IF or ELSE instruction block.
IL_OP_ENDLOOP	Indicates the end of a LOOP or WHILELOOP instruction block.
IL_OP_ENDMAIN	Indicates the end of a main program in a kernel.
IL_OP_ENDSWITCH	Indicates the end of a SWITCH instruction block.
IL_OP_EQ	Performs a float equality comparison.
IL_OP_EXN	Used to compute the single-precision, floating point value of *e* raised to a power.
IL_OP_EXP	Computes the single-precision, floating point value of 2 raised to a power.
IL_OP_EXP_VEC	Computes the single-precision, floating point value of 2 raised to a power for each element in a vector.
IL_OP_EXPP	Used to compute the partial-precision, floating point value of 2 raised to a power.
IL_OP_F_2_D	Converts a single-precision float value to a double-precision float value.
IL_OP_FACEFORWARD	
IL_OP_FLR	Computes the floor of the value in each element of a vector.
IL_OP_FRC	Returns a single-precision, floating point fraction (mantissa).
IL_OP_FTOI	Converts a single-precision float value to an integer value.
IL_OP_FTOU	Converts a single-precision float value to an unsigned integer value.
IL_OP_FUNC	Indicates the start of a FUNC instruction block.
IL_OP_FWIDTH	
IL_OP_GE	Performs a float greater than or equal comparison.
IL_OP_I_ADD	Computes the integer addition of the values in each element of a vector to the values in the corresponding element in another vector.
IL_OP_I_EQ	Performs an integer equality comparison.
IL_OP_I_GE	Performs an integer greater than or equal comparison.
IL_OP_I_LT	Performs an integer less than comparison.
IL_OP_I_MAD	Performs an integer multiplication of the values in each element of a vector by the values in the corresponding elements of another vector; it then adds the value in the corresponding elements of a third vector to the results.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_I_MAX	Performs an integer comparison of the value in each element of a vector to the value in the corresponding element of another vector, and returns the larger of the two values in the corresponding element of a third vector.
IL_OP_I_MIN	Performs an integer comparison of the value in each element of a vector to the value in the corresponding element of another vector, and returns the smaller of the two values in the corresponding element of a third vector.
IL_OP_I_MUL	Performs an integer multiplication of the values in each element of a vector by the values in the corresponding element in another vector, and returns the lower 32-bits of the result.
IL_OP_I_MUL_HIGH	Performs an integer multiplication of the values in each element of a vector by the values in the corresponding element in another vector, and returns the upper 32-bits of the result.
IL_OP_I_NE	Performs an integer inequality comparison.
IL_OP_I_NEGATE	Computes the two's complement negation of the values in each element in a vector.
IL_OP_I_NOT	Performs a bit-wise one's complement on a vector.
IL_OP_I_OR	Performs a logical-OR on two vectors.
IL_OP_I_SHL	Shifts integer values in each element of a vector the specified number of bits to the left.
IL_OP_I_SHR	Shifts the integer values in each element of a vector a the specified number of bits to the right through sign extension.
IL_OP_I_XOR	Performs a logical-XOR on two vectors.
IL_OP_IF_LOGICALNZ	Logically compares a parameter to zero, and executes the IF instruction block if the comparison evaluates FALSE.
IL_OP_IF_LOGICALZ	Logically compares a parameter to zero, and executes the IF instruction block if the comparison evaluates TRUE.
IL_OP_IFC	Logically compares two parameters, and conditionally executes the IF instruction block if the comparison evaluates TRUE.
IL_OP_IFNZ	Used to conditionally execute the IF instruction block if the parameter is not zero.
IL_OP_INITV	
IL_OP_ITOF	Converts an integer value to a single-precision float value.
IL_OP_KILL	Used to perform a float comparison on a parameter and conditionally stop execution and discard the results of a pixel shader invocation if any element in the parameter is less that 0.0f.
IL_OP_LEN	Used to compute the length of a 3-vector.
IL_OP_LIT	
IL_OP_LN	Computes the single-precision, floating point natural logarithm of a component.
IL_OP_LOAD	Used to fetch data from a specified Buffer or Texture without filtering.
IL_OP_LOD	
IL_OP_LOG	Computes the single-precision, floating point binary logarithm of a component.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_LOG_VEC	Computes the single-precision, floating point binary logarithm of the values in each element of a vector.
IL_OP_LOGP	Used to compute the partial-precision, floating point binary logarithm of a component.
IL_OP_LOOP	Indicates the start of a LOOP instruction block.
IL_OP_LRP	
IL_OP_LT	Performs a float less-than comparison.
IL_OP_MAD	Performs a single-precision, floating point multiplication of the values in each element of a vector by the values in the corresponding elements of another vector; it then adds the value in the corresponding elements of a third vector to the results.
IL_OP_MAX	Performs a single-precision, floating point comparison of the value in each element of a vector to the value in the corresponding element of another vector, and returns the larger of the two values in the corresponding element of a third vector.
IL_OP_MEMEXPORT	Used to export the value in *src1* to a memory stream.
IL_OP_MEMIMPORT	Used to import a value in memory to a TEMP register.
IL_OP_MIN	Performs a single-precision, floating point comparison of the value in each element of a vector to the value in the corresponding element of another vector, and returns the smaller of the two values in the corresponding element of a third vector.
IL_OP_MMUL	
IL_OP_MOD	Computes the single-precision, floating point division of the values in each element of a vector by the values in the corresponding elements of another vector, and returns the remainder of each division.
IL_OP_MOV	Unconditionally moves the value in each element of a vector to the corresponding element of another vector.
IL_OP_MOVA	Used to unconditionally move the rounded or truncated value in each element of a vector to the corresponding element of another vector.
IL_OP_MUL	Performs a single-precision, floating point multiplication of the values in each element of a vector by the values in the corresponding element in another vector, and returns the lower 32-bits of the result.
IL_OP_NE	Performs a float inequality comparison.
IL_OP_NOISE	
IL_OP_NOP	Used to pad IL streams so that they are the same length in memory without unnecessary computations.
IL_OP_NRM	Computes the single-precision, floating point, normalized value of the value in each element of a vector.
IL_OP_PIREDUCE	
IL_OP_POW	Computes the value in a component raised to the power of the value in another component.
IL_OP_PRECOMP	
IL_OP_PROJECT	
IL_OP_RCP	Computes the single-precision, floating point reciprocal of the value in a component.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_REFLECT	
IL_OP_RESINFO	Used to get information about a resource.
IL_OP_RET	Used to unconditionally transfer control from the end of a FUNC instruction block back to the caller.
IL_OP_RET_DYN	Unconditionally transfers control from anywhere in a FUNC instruction block back to the caller.
IL_OP_RET_LOGICALNZ	Compares a parameter to zero, and conditionally transfers control from anywhere in a FUNC instruction block back to the caller if the parameter is not zero.
IL_OP_RET_LOGICALZ	Compares a parameter to zero, and conditionally transfers control from anywhere in a FUNC instruction block back to the caller if the parameter is zero.
IL_OP_RND	Rounds the single-precision, floating point value in each element of a vector to the nearest integer.
IL_OP_ROUND_NEAR	Rounds the single-precision, floating point value in each element of a vector to the nearest even integer.
IL_OP_ROUND_NEG_INF	Rounds the single-precision, floating point value in each element of a vector towards $-\infty$.
IL_OP_ROUND_POS_INF	Rounds the single-precision, floating point value in each element of a vector towards ∞.
IL_OP_ROUND_ZERO	Rounds the single-precision, floating point value in each element of a vector towards zero.
IL_OP_RSQ	Computes the single-precision, floating point square root of the reciprocal of the value in a component.
IL_OP_RSQ_VEC	Computes the single-precision, floating point square root of the reciprocal of the value in each element of a vector.
IL_OP_SAMPLE	Performs a single-precision, floating point sample of a resource.
IL_OP_SAMPLE_B	Used to perform a single-precision, floating point sample of a resource with filtering and bias.
IL_OP_SAMPLE_C	Used to perform a single-precision, floating point sample of a resource with filtering and comparison.
IL_OP_SAMPLE_C_LZ	Used to perform a single-precision, floating point sample of a resource with filtering and comparison.
IL_OP_SAMPLE_G	Used to perform a single-precision, floating point sample of a resource with filtering and gradient.
IL_OP_SAMPLE_L	Performs a single-precision, floating point sample of a resource with filtering.
IL_OP_SET	Performs a single-precision, floating point comparison of the value in each element of a vector with the value in each element of another vector, and places a 1.0f in the corresponding element of a third vector if the comparison evaluates TRUE; otherwise, it places a 0.0f in the corresponding element of the third vector.
IL_OP_SGN	Computes the single-precision, floating point sign of the value in each element of a vector.
IL_OP_SIN	Computes the sine of the value in a component.
IL_OP_SIN_VEC	Computes the sine of the value in each element of a vector.
IL_OP_SINCOS	Used to compute the sine and cosine of a component.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_SQRT	Computes the single-precision, floating point square root of the value in a component.
IL_OP_SQRT_VEC	Computes the single-precision, floating point square root of the value in each element of a vector.
IL_OP_SUB	Computes the single-precision, floating point subtraction of the values in each element of a vector from the values in the corresponding element in another vector.
IL_OP_SWITCH	Indicates the start of a SWITCH instruction block, and provides a <switch-value> to later CASE instructions.
IL_OP_TAN	Used to compute the tangent of a component.
IL_OP_TEXLD	
IL_OP_TEXLDB	
IL_OP_TEXLDD	
IL_OP_TEXLDMS	
IL_OP_TEXWEIGHT	
IL_OP_TRANSPOSE	
IL_OP_TRC	Performs a single-precision, floating point truncation of the value in each element of a vector.
IL_OP_U_DIV	Performs an unsigned integer division of the values in each element of a vector by the values in the corresponding element in another vector.
IL_OP_U_GE	Performs an unsigned integer greater than or equal comparison.
IL_OP_U_LT	Performs an unsigned integer less than comparison.
IL_OP_U_MAD	Performs an unsigned integer multiplication of the values in each element of a vector by the values in the corresponding elements of another vector; it then adds the value in the corresponding elements of a third vector to the results.
IL_OP_U_MAX	Performs an unsigned integer comparison of the value in each element of a vector to the value in the corresponding element of another vector; it then returns the larger of the two values in the corresponding element of a third vector.
IL_OP_U_MIN	Performs an unsigned integer comparison of the value in each element of a vector to the value in the corresponding element of another vector; it then returns the smaller of the two values in the corresponding element of a third vector.
IL_OP_U_MOD	Computes the unsigned integer division of the values in each element of a vector by the values in the corresponding elements of another vector; it then returns the remainder of each division.
IL_OP_U_MUL	Performs an unsigned integer multiplication of the values in each element of a vector by the values in the corresponding element in another vector; it then returns the lower 32-bits of the result.
IL_OP_U_MUL_HIGH	Performs an unsigned integer multiplication of the values in each element of a vector by the values in the corresponding element in another vector; it then returns the upper 32-bits of the result.

Table 5.28 ILOpCode Enumeration Types (Cont.)

Enumeration	Description
IL_OP_U_SHR	Shifts without sign extension unsigned integer values in each element of a vector the specified number of bits to the right.
IL_OP_UTOF	Converts an unsigned integer value to a single-precision float value.
IL_OP_WHILE	Indicates the start of a WHILELOOP instruction block.

Chapter 6
Instructions

An IL stream consists mainly of IL instruction packets. Each packet begins with an IL_OpCode token. The type and number of tokens that follow depends upon the value of the *code* field in the initial IL_OpCode token, as well as the `modifier_present` field in each of the following IL_Dst tokens and IL_Src tokens. This chapter describes each instruction packet, including its operation and usage within the IL stream. Some instructions are defined by pseudo code. In this chapter, the symbol `V[i]` refers to source `i` post swizzle.

6.1 Formats

Most instructions use a standardized format. Rather than repeat the format in each instruction, the common format is given here. Tokens have the following order:

1. Opcode token.
2. Destination information (zero or one set of tokens: *dst* token, any relative addressing, any modifiers).
3. Source information (zero or more sets of tokens), src_token, any relative addressing information, any modifiers.

Formats include: zero or one destination, followed by any number of sources. See the specific opcode to determine the number of destinations/sources allowed.

6.2 Instruction Notes

6.2.1 Notes on Comparison Instructions

Comparison instructions support four comparison types: float, double, integer, and unsigned integer, with different instructions for each type. Instructions return either TRUE (comparison condition met) or FALSE (comparison condition not met). Integer comparison instructions return 0xFFFFFFFF (TRUE) and 0x00000000 (FALSE). Float comparison instructions can be configured to return 1.0f (TRUE) and 0.0f (FALSE).

Integer instructions use signed arithmetic when comparing operands; unsigned integer instructions use unsigned arithmetic. Float instructions use signed arithmetic when comparing operands; however, denorms are flushed before all float instructions (the original source registers remain untouched). Thus, +0 and

-0 are equivalent for float comparisons. Also, float instructions return FALSE when either operand holds NaN. See the IEEE 754 documentation for more information on floating point rules.

6.2.2 Notes on Flow Control Instructions

Flow control instructions determine the order in which instructions are executed by the hardware.

A *flow-control-block* is code within:

- A subroutine: code between FUNC and RET, or between FUNC and ENDFUNC.
- Between IFNZ and ENDIF.
- Between IFC and ENDIF.
- Between LOG_IF and ENDIF.
- Between ELSE and ENDIF.
- Between IFNZ and ELSE.
- Between IFC and ELSE.
- Between LOOP and ENDLOOP.
- Between WHILELOOP and ENDLOOP.

There are two forms of loop: LOOP/ENDLOOP used for DX9-style counted loops, and WHILELOOP/ENDLOOP used for DX10-style while loops.

The following are the restrictions on when particular control flow instructions are allowed.

- LOOPs and WHILELOOPs must terminate in the same *flow-control-block* in which they begin.
- END, ENDMAIN, and ENDFUNC cannot be placed within a *flow-control-block.*

6.2.3 Notes on Input/Output Instructions

Instructions in this section are expected to be used only by graphic clients. Consult the DX10 documentation or other graphics programming reference for details.

6.2.4 Notes on Conversion Instructions

Even though IL is an untyped language, conversion instructions are required to convert between different data formats so that source data for an instruction appears in the proper format. Without format conversion, instructions could provide incorrect results.

6.2.5 Notes on Double Precision Instructions

Double precision values are represented by a pair of registers. Outputs are either the pair yx or to the pair wz, where the msb is stored in y/w. For example:

`Idata 3.0 => (0x4008000000000000)` in register `r` looks like:

```
r.w =  0x40080000  ;high dword
r.z =   0x00000000 ;low  dword
```

Or by:

```
r.y =  l0x40080000 ;high dword
r.x =  l0x00000000 ;low  dword
```

All source double inputs must be in xy (after swizzle operations). For example:

```
d_add r1.xy, r2.xy, r2.xy
```

Or

```
d_add r1.zw, r2.xy, r2.xy
```

Each computes twice the value in r2.xy, and places the result in either xy or zw.

The user can set the output mask to either xy or zw. The msb is in y/w, so users can test the sign of the result with single precision operations.

All inputs are in the first two components xy of each source.

6.3 Flow Control Instructions

Unconditional BREAK out of Loop or Switch Constructs

Instructions **BREAK**

Syntax BREAK

Description BREAK is used only in a loop or switch statement. It terminates the smallest enclosing loop or switch. Control passes to the statement following the terminated statement, if any.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_BREAK
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related BREAKC, BREAK_LOGICALZ, BREAK_LOGICALNZ.

Conditional BREAK Using Floating Point

Instructions **BREAKC**

Syntax `breakc_relop(op) src0, src1`

Description BREAKC compares two registers using a float comparison on the x component of the sources. It is used only in a loop or switch statement. It terminates the smallest enclosing loop or switch. Control passes to the statement following the terminated statement, if any.

Operation:

```
if (v0.x relop v1.x) {
    Break;
}
```

Format 2-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_BREAKC
control	29:16	relop(*op*). See Table 5.14 on page 5-8.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related BREAK, BREAK_LOGICALZ, BREAK_LOGICALNZ.

Conditional BREAK Using Integers

Instructions **BREAK_LOGICALNZ, BREAK_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_BREAK_LOGICALNZ	break_logicalnz *src0*	If *src0*.x ≠ 0 break.
IL_OP_BREAK_LOGICALZ	break_logicalz *src0*	If *src0*.x == 0 break.

Description This form of Break compares a register component using integer compare to zero. The break is done if all bits of *src0* are zero. *src0* must have a swizzle that selects a single component. The break statement occurs only in a loop-*statement* or a switch statement. It causes termination of the smallest enclosing loop or switch statement; control passes to the statement following the terminated statement, if any.

Format 1-input.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related BREAK, BREAKC.

Unconditional CALL

Instructions **CALL**

Syntax `call <integer label>`

Description Call a subroutine. The subroutine is identified by a label id, corresponding to the ID in a function statement. Subroutines can nest up to 32 levels deep. Direct or indirect recursion is permitted. If there are already 32 entries on the return address stack, and a “call” is issued, the call is skipped.

.Ordinal	Token
1	IL_OpCode token with code set to IL_OP_CALL (control field ignored).
3	Unsigned integer representing the subroutine label.

Format 0-input.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_CALL
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	Must be zero.		
3	Unsigned integer representing label of the subroutine.		

Related CALLNZ, CALL_LOGICALZ, CALL_LOGICALNZ.

CALL if Boolean register is not zero

Instructions **CALLNZ**

Syntax `callnz src0, <integer label>`

Description CALLNZ compares *src0.*x to zero using an integer comparison. If any bits of *src0.*x are not zero, CALLNZ pushes the address of the next instruction in the shader onto the return address stack and transfers control to the FUNC block identified by <*integer label*>. CALLs can be nested up to 32 levels deep. If all bits of *src0.*x are zero, or the return address stack already contains 32 addresses, the CALL is skipped and execution continues at the next instruction in the shader. Recursion is permitted either directly, using another CALL instruction, or indirectly using relative addressing.

src0 must be a CONST_BOOL register. See Section 4 on page 4-1 for information on the register types.

CALLNZ cannot be used with relative addressing or a source modifier. Both the `modifier_present` and `relative_address` fields of the IL_Src token must be zero.

Operation:

```
if (v0.x != 0) {
Call label
}
```

Format 1-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_CALLNZ
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.

2 IL_Src token (*src0)* with `register_type` set to IL_REGTYPE_CONST_BOOL. The `modifer_present` and `relative_address` field must be set to 0.

Field Name	Bits	Description
register_num	15:0	Any available register.
register_type	21:16	CONST_BOOL
Remaining fields	31:22	Must be zero.

3 Unsigned integer representing the label of the subroutine.

Related CALL, CALL_LOGICALZ, CALL_LOGICALNZ.

Conditional CALL Based on Boolean

Instructions **CALL_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_CALL_LOGICALZ	call_logicalnz *src0* <*integer label*>	If *src0*.x == 0, call <integer label>.

Description

Performs conditional call based on a Boolean constant. Scr0 must be a CONST_BOOL register. This instruction does not allow the use of a source modifier. The modifier_present field of the IL_Src token must be set to 0. This instruction does not allow relative addressing. The relative_address field of the IL_Src token must be set to 0.

All four forms call a subroutine. The subroutine is identified by a label id, corresponding to the ID in a function statement. Subroutines can nest up to 32 levels deep. Recursion is permitted, directly or indirectly. If there are already 32 entries on the return address stack and a “call” is issued, the call is skipped.

.Ordinal	Token
1	IL_OpCode token with code set to IL_OP_CALL_LOGICALZ (control field ignored).
2	(*src0*) representing any valid IL Source (unspecified number of tokens).
3	Unsigned integer following all source tokens representing the subroutine label.

Operation:

```
if (v0.x eq 0) {
      Call subroutine;
}
```

Format 1-input.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	See *Syntax*, above.
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Src token and IL_Src_Mod token (if required) representing any valid IL Source. See Section 2.2.6, “Source Token,” page 2-4.		
3	Unsigned integer representing the label of the subroutine.		

Related CALL, CALLNZ.

Conditional CALL Based on Integer

Instructions **CALL_LOGICALNZ**

Syntax

Opcode	Syntax	Description
IL_OP_CALL_LOGICALNZ	call_logicalnz, *src0*	If *src0*.x ≠ 0, call <integer label>.

Description Conditional call subroutine at the label. The 32-bit register component (*src0*) is tested. If any bits are not zero, the instruction performs the call. It checks only the x component (after swizzling) of *src0* is checked. All four forms call a subroutine. The subroutine is identified by a label id, corresponding to the id in a function statement. Subroutines can nest up to 32 levels deep. Recursion is permitted, directly or indirectly. If there are already 32 entries on the return address stack, and a “call” is issued, the call is skipped.

Operation:

```
if (v0.x ne 0) {
       Call subroutine;
}
```

Format 1-input.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	See *Syntax*, above.
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Src token and IL_Src_Mod token (if required) representing any valid IL Source. See Section 2.2.6, “Source Token,” page 2-4.		
3	Unsigned integer representing the label of the subroutine.		

Related CALL, CALLNZ.

CASE Statement

Instructions **CASE**

Syntax `case case-value`

Description This case statement is used in a switch. This instruction is followed by a 32 integer that identifies the case. Compares are done using integer arithmetic. Falling through cases is valid, as in C. This can be used to implement a DX10 case instruction.
Operation:

see SWITCH.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CASE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DEFAULT, ENDSWITCH, SWITCH.

Unconditional CONTINUE

Instructions **CONTINUE**

Syntax `continue`

Description CONTINUE causes shader execution to continue at the previous LOOP or WHILELOOP instruction. This is used only in a LOOP - ENDLOOP instruction block.

Operation:

```
If this is a counted loop {
  LoopIterationCount = LoopIterationCount - 1;
  LoopCounter = LoopCounter + LoopStep;
  LoopCounter = (LoopCounter > 0) ? LoopCounter : 0;
  if (LoopIterationCount > 0)
    Continue execution at the StartLoopOffset;
} else {
      Continue execution at the StartLoopOffset;
}
```

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CONTINUE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CONTINUEC, CONTINUE_LOGICALZ, CONTINUE_LOGICALNZ.

CONTINUE Using Floating Point

Instructions **CONTINUEC**

Syntax continuec_relop(*op*) *src0*, *src1*

Description Conditionally continues execution at previous LOOP or WHILELOOP instruction if the condition is true. Can only be used in a LOOP - ENDLOOP instruction block.

Operation:

```
if (v0.x relop v1.x) {
      Continue;
}
```

Format 2-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CONTINUEC
control	29:16	relop(*op*). See Table 5.14 on page 5-8.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CONTINUE, CONTINUE_LOGICALZ, CONTINUE_LOGICALNZ.

Conditional CONTINUE Using Integers

Instructions **CONTINUE_LOGICALNZ, CONTINUE_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_CONTINUE_LOGICALNZ	continue_logicalnz *src0*	If *src0*.x ≠ 0 call <integer label>.
IL_OP_CONTINUE_LOGICALZ	continue_logicalnz *src0*	If *src0*.x == 0 call <integer label>.

Description Conditionally continues execution at the beginning of the current loop:
CONTINUE_LOGICAL_Z continues the loop if all bits of *src0*.x are zero.
CONTINUE_LOGICAL_NE continues the loop if any bit of *src0*.x is not zero.

Can only be within a LOOP - ENDLOOP switch block.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CONTINUE, CONTINUEC.

DEFAULT Statement

Instructions **DEFAULT**

Syntax `default`

Description DEFAULT starts an instruction block within a SWITCH instruction block (see page 6-19). Unlike a CASE label, a DEFAULT label does not provide a value for comparison.

This is like the default in C. Falling through or into a DEFAULT section is valid. There can be only one DEFAULT statement in each SWITCH block.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DEFAULT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CASE, ENDSWITCH, SWITCH.

ELSE

Instructions **ELSE**

Syntax `else`

Description This instruction is the start of the ELSE clause of an IFNZ-ELSE-ENDIF or IFC-ELSE-ENDIF or LOG_IF-ELSE-ENDIF block. ELSE must be after an IFC, IFNZ, or LOG_IF instruction in the stream.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ELSE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related IFNZ, IFC, ENDIF.

ENDSWITCH

Instructions **ENDSWITCH**

Syntax endswitch

Description ENDSWITCH indicates the end of a SWITCH instruction block.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ENDSWITCH
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CASE, DEFAULT, SWITCH.

End of Main Program

Instructions **ENDMAIN**

Syntax endmain

Description ENDMAIN indicates the end of the shader source for the main program. Only FUNC instruction blocks and the END statement (see page 6-13) can follow an ENDMAIN statement. An ENDMAIN statement is only required if the shader contains subroutines.

Operation:

End of shader execution, and beginning of subroutine definitions.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ENDMAIN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

End of Stream

Instructions **END**

Syntax end

Description END indicates the end of an IL stream and must be the last statement in the stream. All shader programming, including subroutines, must be placed before this instruction.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_END
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

End of FUNC (subroutine)

Instructions **ENDFUNC**

Syntax endfunc

Description ENDFUNC indicates the end of a shader subroutine. Only FUNC blocks and the END statement can follow an ENDFUNC statement. This instruction is required only if the shader contains subroutines.

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ENDFUNC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related FUNC.

End of IF block

Instructions **ENDIF**

Syntax endif

Description Indicates the end of an IFNZ - ENDIF, IFC - ENDIF, IFNZ - ELSE - ENDIF, or IFC - ELSE - ENDIF block. The ENDIF statement must follow an ELSE, IFC, IFNZ or LOG_IF instruction.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ENDIF
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related ELSE, IFC, IFNZ, IF_LOGICAL*.

End of a LOOP or WHILELOOP block

Instructions **ENDLOOP**

Syntax endloop

Description Indicates the end of a LOOP - ENDLOOP or WHILELOOP - ENDLOOP instruction block. The instruction must follow a LOOP/WHILELOOP instruction. This instruction cannot be with a FUNC - RET, IFNZ - ENDIF, IFC - ENDIF, IFNZ - ELSE, IFC - ELSE, or ELSE - ENDIF block unless its corresponding LOOP or WHILELOOP is also in that block.

Operation:

(For counted loops)

```
LoopIterationCount = LoopIterationCount - 1;
LoopCounter = LoopCounter + LoopStep;
LoopCounter = (LoopCounter > 0) ? LoopCounter : 0;
if (LoopIterationCount > 0)
      Continue execution at the StartLoopOffset;
```

For While loops

```
   Transfer control back to the test of the while loop.
```

Format 0-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ENDLOOP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LOOP, WHILELOOP.

Start of a FUNC Instruction Block

Instructions **FUNC**

Syntax `func <integer label>`

Description Indicates the start of a subroutine. Each FUNC statement must have a unique label, *<integer-label>*. A RET instruction must be before the END instruction. This instruction can be used only after an ENDMAIN instruction. The code between FUNC and RET is executed if a CALL or CALLNZ with the same label value is executed.

Operation:

Defines the lexical start of a subroutine.

Format 0-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_FUNC
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	Must be zero.		
3	Unsigned integer representing the label of the subroutine.		

Related ENDFUNC.

Conditional IF Using Floating Point

Instructions **IFC**

Syntax `ifc_relop(op) src0, src1`

Description This is the start of an IFC - ENDIF or IFC - ELSE - ENDIF block. It skips a block of code based on the value of *src0* compared to *src1*. The IFC block must end with and ELSE or ENDIF instruction.

Operation:

```
if (v0.x relop v1.x ) {
    Execute following instructions;
} else {
    Jump to the instruction following the next ELSE or ENDIF instruction;
}
```

Format 2-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_IFC
control	29:16	relop(*op*). See Table 5.14 on page 5-8.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related IF_LOGICALNZ, IF_LOGICALZ, IFNZ.

Execute instruction block if Boolean register is not zero

Instructions **IFNZ**

Syntax `ifnz` *`src0`*

Description This starts an IFNZ - ENDIF or IFNZ - ELSE - ENDIF block. IFNZ cannot be used with relative addressing or a source modifier. Both the `relative_address` and `modifier_present` fields of the IL_Src token must be zero. An IFNZ block must end with an ELSE or ENDIF instruction.

Operation:

```
if (v0.x) {
    Execute following instructions;
} else {
    Jump to the instruction following the next ELSE or ENDIF instruction;
}
```

Format 1-input.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_IFNZ
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related IFC, IF_LOGICALNZ, IF_LOGICALZ.

Conditional IF Using Integers

Instructions **IF_LOGICALNZ, IF_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_IF_LOGICALNZ	if_logicalnz *src0.x*	If *src0*.x ≠ 0 execute instruction block.
IL_OP_IF_LOGICALZ	if_logicalz *src0.x*	If *src0*.x == 0 execute instruction block.

Description These are integer versions of the IF statement. They skip a block of code based on the value of *src0*.x. The LOG_IF block must end with and ELSE or ENDIF instruction. The source selector must replicate the component to be tested into all four components. The test uses integer tests, so values like Nan or -0 are not equal to zero.

Operation for IF_LOGICALNZ:

```
if (v0.x has any bit non-zero) {
    Execute following instructions;
} else {
    Jump to the instruction following the next ELSE or ENDIF instruction;
}
```

Operation for IF_LOGICALZ:

```
if (v0.x has all bits zero) {
    Execute following instructions;
} else {
    Jump to the instruction following the next ELSE or ENDIF instruction;
}
```

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related IFC, IFNZ.

Start of a Counted LOOP Block

Instructions **LOOP**

Syntax loop [*repeat*] *src0*

Description LOOP indicates the start of a LOOP instruction block. *src0* must be a register of type IL_REGTYPE_CONST_INT. *src0.*x specifies the iteration count used for the loop. If repeat is set to 0, *src0.*y specifies the initial value for the current *loop-counter* used for relative addressing. If repeat is set to 1, *src0.y* and *src0.z* are not used, and the current auto-increment loop counter is not incremented during this loop. This instruction starts a LOOP - ENDLOOP block.

If this instruction is within a FUNC - RET, IFNX - ENDIF, IFX - ENDIF, IFNZ - ELSE, IFC - ELSE, or ELSE - ENDIF block, its corresponding ENDLOOP must also be within that block.

This instruction is provided for iteration. It only increments the current auto-incremented loop counter if repeat is set to 0.

ENDLOOP must follow the last instruction of a loop block. The ENDLOOP instruction offset must be greater than the LOOP instruction offset.

If repeat is set to 0, the loop initial value and the loop iteration count cannot be negative.

LOOP cannot be used with relative addressing or a source modifier. Both the `relative_address` and `modifier_present` fields of the IL_Src token must be zero.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LOOP
control	29:16	*repeat* flag 0 *src0*.y holds the initial value for the current loop-counter used for relative addressing. *src0*.z holds the loop step. *src0*.y and *src0*.z cannot be negative. 1 *src0*.y and *src0*.z are not used and the current auto-increment loop-counter is not incremented during this loop. *src0*.y and *src0*.z can be negative.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related ENDLOOP, WHILELOOP.

Start of a WHILELOOP Block

Instructions **WHILELOOP**

Syntax whileloop

Description WHILELOOP indicates the start of a WHILELOOP instruction block. A WHILELOOP block can iterate indefinitely, exiting only when a BREAK instruction is executed. It can be used as the translation of a DX10 loop statement. There is no limit to the amount of loop nesting. Although a WHILELOOP can iterate indefinitely, overall execution of the shader can be terminated after some number of instructions are executed.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_WHILE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related ENDLOOP, LOOP.

Start of a SWITCH Block

Instructions **SWITCH**

Syntax switch *src0*

Description A switch/endswitch construct behaves exactly as a switch construct in the C language. The *src0* must be a 32-bit register component or immediate quantity. Compares are done using integer arithmetic. Falling through cases are valid, as in C. This instruction can be used to implement DX10 case instruction. Switch statements can be nested without limits.

Operation:

Same as a C switch statement.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SWITCH
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CASE, DEFAULT, ENDSWITCH.

RETURN from a FUNC Block (end of subroutine)

Instructions **RET**

Syntax `ret`

Description Returns from a subroutine, and indicates the end of the subroutine. It marks the end of a subroutine and can be present only at the end (it cannot be within a flow-control block). This instruction can implement DX9 returns.

Operation:

Continue execution after the call statement which executed this subroutine.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RET
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related CALL, CALLNZ, CALL_LOGICALZ, CALL_LOGICALNZ, FUNC, RET_DYN, RET_LOGICALNZ, RET_LOGICALZ.

RETURN from a FUNC Block (not end of subroutine)

Instructions **RET_DYN**

Syntax `ret_dyn`

Description Returns from a subroutine to the instruction after the call. It can appear anywhere in a subroutine, any number of times.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RET_DYN
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related CALL, CALLNZ, CALL_LOGICALZ, CALL_LOGICALNZ, FUNC, RET, RET_LOGICALNZ, RET_LOGICALZ.

Conditional RETURN from FUNC Block Using Integer

Instructions **RET_LOGICALNZ, RET_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_IF_LOGICALNZ	if_logicalnz *src0.x*	If *src0*.x ≠ 0 execute instruction block.
IL_OP_IF_LOGICALZ	if_logicalz *src0.x*	If *src0*.x == 0 execute instruction block.

Description These instructions conditionally return tot he instruction after the call. *src1*.x is tested after swizzle. The instructions can appear anywhere in a subroutine, any number of times.

The 32-bit value supplied by *src1* is tested at the bit level:

For RET_LOGICALNZ, if any bit is non-zero, the statement returns.
For RET_LOGICALZ, if all bits are zero, the statement returns.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	logic_op
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CALL, CALLNZ, CALL_LOGICALZ, CALL_LOGICALNZ, FUNC, RET, RET_DYN.

6.4 Declaration and Initialization Instructions

Declare Registers as Part of an Array

Instructions **DCLARRAY**

Syntax dclarray *src0, src1*

Description Declares a range of registers as part of an array indexed access (to be accessed through base/loop relative addressing). Only the INTERP and TEXCOORD registers can be used. *src0* and *src1* must be of the same register type. Base and loop relative addressing cannot be used on TEMP, INTERP, and TEXCOORD registers not declared within the range of *src1* and *src2*. If registers outside the range of registers in the array declared with this instruction are accessed, the compiler cannot guarantee the stability of the shader.

Base and loop relative addressing perform indexing based on the register number, not on the register's position in the array. (Declaring an array of registers 2 to 6, and indexing when the loop counter is 3, accesses register number 3, not register number 5.)

There can be numerous DCLARRAYs for each type, so long as they do not overlap in range.

DCLARRAY cannot be used with relative addressing or a source modifier. Both the `modifier_present` and `relative_address` fields of the IL_Src tokens must be zero.

Format 2-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DCLARRAY
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related DCL_INDEXED_TEMP_ARRAY.

Declare a Constant Buffer

Instructions **DCL_CB**

Syntax dcl_cb *cbm[n]*

Description Declares that the shader can use the constant buffer. This instruction must occur before any use of the constant buffer. The source of this instruction must be of type IL_REGTYPE_CONSTANT_BUFFER. The cb index m can be in the range of 0-14. The index n can be 4k.

DCL_CB cannot be used with relative addressing or a source modifier. Both the modifier_present and relative_address fields of the IL_Src token must be zero.

If the instruction is being used to specify an immediate constant buffer, set the pri_modifier_present bit to 1. The subsequent 32 bits are an unsigned integer specifying the number of elements in the constant buffer, followed by one 32-bit floating point value specifying each member of the immediate constant buffer.

Note that the HD4000-family of devices does not support both constant buffer and constant file operations; drivers must select only one mode.

Format 1-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_DCL_CONST_BUFFER
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	0: Instruction declares a constant buffer. 1: Does not declare a constant buffer. Instruction declares an Immediate constant buffer.
2	pri_modifier_present == 0		IL_Src token (*src0*).
	pri_modifier_present == 1		Number of elements, n, in an immediate constant buffer.
3	pri_modifier_present == 0		Not used.
	pri_modifier_present == 1		First (index 0) 32-bit element of the immediate constant buffer.
4 to n+2	pri_modifier_present == 0		Not used.
	pri_modifier_present == 1		Second (index 1) through nth (index n-1) 32-bit elements of the immediate constant buffer.

Related None.

Declare register defaults

Instructions **DCLDEF**

Syntax dcldef_x(*comp*)_y(*comp*)_z(*comp*)_w(*comp*) *dst*

Description Declares the default value for register *dst* and cannot be used more than once per register type in a shader. *dst* must be a TEMP or ADDR register type (see Section 5.3 on page 5-2). Each component in the register is set individually to an ILDefaultVal enumerated type. See Table 5.11 on page 5-7 for ILDefaultVal values. The xdefault, ydefault, zdefault, and wdefault describe the default value for each component of the register.

DCLDEF cannot be used with relative addressing or a destination modifier. Both the modifier_present and relative_address fields of the IL_Dst token must be zero.

Format 0-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DCLDEF
	xdefault	17:16	x-component default value.
	ydefault	19:18	y-component default value.
	zdefault	21:20	z-component default value.
	wdefault	23:22	w-component default value.
	reserved	31:24	Must be zero.

The component default types can be any value of the enumerated type ILDefaultVal. See Table 5.11 on page 5-7.

2 IL_Dst token (*dst*) where the register_type field is set to IL_REGTYPE_TEMP or IL_REGTYPE_ADDR. The modifer_present and relative_address fields must be set to 0.

Related None.

Constant Integer or Float Register Definition

Instructions **DEF**

Syntax `def dst, <number>, <number>, <number>, <number>`

Description DEF declares the constant value for register *dst* and cannot be used more than once on the same register. It indicates that a floating point or integer constant contains a given value when the shader is executed. This instruction is followed by four words that contain the bit values of the constant. *dst* must be a CONST_INT or CONST_FLOAT register type (See Section 4 on page 4-1). Each component of a register can be set individually. CONST_INT registers do not use the w-component.

Clients must ensure that the constant values set through state match the values indicated in this instruction.

DEF cannot be used with relative addressing or a destination modifier. Both the `modifier_present` and `relative_address` fields of the IL_Dst token must be zero.

Operation:

```
VECTOR v;
v[0] = x;
v[1] = y;
v[2] = z;
v[3] = w;
WriteResult(v, dst);
```

Format 0-input, 1-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DEF
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Dst token (*dst*) where the register_type field is set to IL_REGTYPE_CONST_INT or IL_REGTYPE_CONST_FLOAT. The modifer_present and relative_address fields must be set to zero.		
3	x-component, 32-bit integer or float		
4	y-component, 32-bit integer or float		
5	z-component, 32-bit integer or float		
6	w-component, 32-bit float only (not used for a CONST_INT register)		

Related DEFB.

Constant Boolean Register Definition

Instructions **DEFB**

Syntax defb *dst*, <unsigned integer>

Description Indicates that a boolean constant contains a given value when the shader is executed. Clients must ensure that the constant values set through state match the values indicated in this instruction. A shader cannot use this instruction on the same register more than once.

If the value of the given unsigned integer is 0, the boolean is set to false. If the value the given unsigned integer is non-zero, set the boolean to true.

DEFB cannot be used with relative addressing or a destination modifier. Both the modifier_present and relative_address fields of the IL_Dst token must be zero.

Operation:

```
CONST_BOOL b;
b = TRUE;
if(val == 0) {
    b=FALSE;
}
WriteResult(b, dst);
```

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DEFB
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Dst token (*dst*) where the register_type field is set to IL_REGTYPE_CONST_BOOL. The modifer_present and relative_address fields must be set to zero.		
3	Boolean value, 32-bit unsigned integer • 0: FALSE • not 0: TRUE		

Related DEF.

Declare an Indexed Array

Instructions **DCL_INDEXED_TEMP_ARRAY**

Syntax `dcl_indexed_temp_array` *`src0`*`[`*`n`*`]`

Description Declares a temporary array.

Each indexed temp array to be used in the Shader must be declared. *src0* must be of type IL_REGTYPE_ITEMP. The modifier_present and relative_address fields must be zero. n is the maximum size in 128-bit units.

DX10 limits the total storage for temps (indexed plus non-indexed) to be ≤ 4096 registers (each a four-component vector).

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_DCL_INDEXED_TEMP_ARRAY
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None. DCLARRAY.

Declare an Input Register

Instructions **DCL_INPUT**

Syntax `dcl_input[_usage(`*`usage`*`)][_interp(`*`mode`*`)]` *`dst`*`[.mask]`

Description Declares an input register for the shader. *`dst`* must be an IL_REGTYPE_INPUT register type (see Chapter 4, “Register Types”) and can have a mask. It is legal to declare a subset of the component mask from what is output from the previous shader in the pipeline; however, mutually exclusive masks are not allowed (for example: VS outputting o3.xy means PS declaring v3.z input is invalid, but v3.x, v3.y, or v3.xy is legal). Interpolation mode, _interp(mode), is only applicable to pixel shaders; it is an error to use this in GS or VS.

Possible values for _interp(mode) are:

IL_INTERPMODE_NOTUSED
IL_INTERPMODE_CONSTANT
IL_INTERPMODE_LINEAR
IL_INTERPMODE_LINEAR_CENTROID
IL_INTERPMODE_LINEAR__NOPERSPECTIVE
IL_INTERPMODE_LINEAR_SAMPLE
IL_INTERPMODE_LINEAR_NOPERSPECTIVE_CENYTROID
IL_INTERPMODE_LINEAR_NOPERSPECTIVE_SAMPLE

Declare an Input Register (Cont.)

Format	**Field Name**	**Bits**	**Description**
	code	15:0	IL_DCL_INPUT
	control	20:16	*usage*. Unless using system value (SV), must be IL_IMPORTUSAGE_GENERIC (see Table 5.9 on page 5-4). The usage index is specified by the register number.
	interp mode	23:21	Interpolation mode.
	center	24	This field is always ignores. Center vs centroid must be specified through _interp(mode).
	bias	25	This is ignored, unless the *dst* type is wincoord. For the window coordinate register, this bit specifies if the compiler generates code that adds a bias supplied in constants that are named SC_CONS_SRC_VIEWPORT_BIAS_{X,Y}. Without a bias, the origin is located at the upper-left corner of the screen (DX Style). A bias can be used to move the origin. For example, to move the origin to the lower-left corner of a window (OGL style), DX must set this to 0, OGL must set this to 1.
	invert	26	This is ignored, unless the *dst* type is wincoord. If the register_type field of the following IL_DstToken is IL_REGTYPPE_WINCOORD, and this bit is set, then the compiler inverts (multiplies by -1) the y coordinate of the position. By default, the y axis goes down (DX style), but this bit can be used to change the direction to up (OGL style).
	centered	27	This is ignored, unless the *dst* type is wincoord. For the window coordinate register, this bit controls (value 0) defines the origin as the upper-left corner of the RenderTarget. Thus, pixel centers are offset by (0.5f,0.5f) from integer locations on the RenderTarget. This choice of origin makes rendering screen-aligned textures trivial, as the pixel coordinate system is aligned with the texel coordinate system. This choice matches OGL and DX10. The second value (1) defines the origin as the center of the upper-left pixel in the RenderTarget. In other words, the origin is (0.5,0.5) away from the upper left corner of the RenderTarget. Thus, pixel centers are at integer locations. This choice of origin matches DX9. 0 – center of the pixel is 0.5,0.5 1 – center of the pixel is 0,0
	reserved	29:28	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
Related	DCL_OUTPUT, DCLPIN, DCLVPOUT.		

Declare a Primitive

Instructions **DCL_INPUTPRIMITIVE**

Syntax `dcl_input_primitive prim_type`

Description Declare the type of primitive that can be accepted by a geometry shader. Must appear in a geometry shader. Only valid in a geometry shader. The prim_type must be an element of the enumeration IL_TOPOLOGY.

Format 0-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_DCL_INPUTPRIMITIVE
	`control`	29:16	prim_type
	`sec_modifier_present`	30	Must be zero.
	`pri_modifier_present`	31	Must be zero.
2	IL_Src token (*src0*) where the `register_type` field is set to IL_REGTYPE_LITERAL.		
3	x-bits, 32-bit untyped literal.		
4	y-bits, 32-bit untyped literal.		
5	z-bits, 32-bit untyped literal.		
6	w-bits, 32-bit untyped literal.		

Related None.

Declare a Literal

Instructions **DCL_LITERAL**

Syntax `dcl_literal` *src0*, <x-bits>, <y-bits>, <z-bits>, <w-bits>

Description DCL_LITERAL declares the literal to be used in the following instruction. The instruction is followed by four words containing the actual bits of the literal, in order x, y, z, w.

value for register *src0* and can be used more than once on the same register in a shader. The lexically nearest preceding value is used. The 32-bit component literals (x-bits, y-bits, z-bits, and w-bits) are untyped, so that integer and float literals can be initialized with this instruction. *src0* must be a IL_REGTYPE_LITERAL register type (see Section 4.10, “LITERAL,” page 4-6). This instruction cannot be placed in an unreachable code block such as after an unconditional break or return instruction. No modifiers are allowed.

Format 1-input, 0-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_DCL_LITERAL
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Src token (*src0*) where the register_type field is set to IL_REGTYPE_LITERAL.		
3	x-bits, 32-bit untyped literal		
4	y-bits, 32-bit untyped literal		
5	z-bits, 32-bit untyped literal		
6	w-bits, 32-bit untyped literal		

Related None.

Declare the Maximum Number of Vertices

Instructions **DCL_MAX_OUTPUT_VERTEX_COUNT**

Syntax `dcl_max_output_vertex_count` *n*

Description DCL_MAX_OUTPUT_VERTEX_COUNT declares the maximum number of vertices that a single invocation of a geometry shader can emit. A geometry shader can emit a maximum of 1024 32-bit values. Thus, n can be no larger than floor (1024/number of 32-bit values per vertex). For example, if a geometry shader emits one 4-component vector (four 32-bit values) per vertex then n must be 256 or less.

Some implementations may be able to make optimizations by knowing the maximum number of vertices a single geometry shader invocation emits (for a single input primitive). The upper bound on the number of vertices that a geometry shader can produce depends on how large each vertex is. The sum of the number of components in each declared geometry shader output register defines how many 32-bit values are present in a single vertex. For example, if a geometry shader declares that it outputs a single four-component position, plus a three-component color per vertex, then the maximum number of vertices that can be declared for output by a single invocation is floor(1024 / 7). Or, if a Geometry Shader declares that it outputs 32 four-component vectors, the maximum number of vertices that can be declared for output by a single invocation is floor(1024 / 128).

DCL_MAX_OUTPUT_VERTEX_COUNT sets an upper limit on the number of vertices that can be emitted and, a geometry shader invocation can terminate after emitting fewer vertices than the maximum number allowed. An invocation terminates when DCL_MAX_OUTPUT_VERTEX_COUNT is reached. There is no requirement on the minimum number of vertices a geometry shader invocation must emit. The amount of vertices generated by a geometry shader invocation is simply the total number of `emit*` instructions executed in an invocation.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_DCL_MAX_OUTPUT_VERTEX_COUNT
`control`	29:16	*n*, maximum number of vertices.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related EMIT, EMIT_THEN_CUT.

Declare that the Pixel Shader intends to write to its scalar output oDepth register

Instructions **DCL_ODEPTH**

Syntax `dcl_odepth`

Description Declare that the pixel shader intends to write to its scalar output oDepth register. DX10 has some rules for what happens if oDepth is declared, but the shader does not write it.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_DCL_ODEPTH
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related None.

Declare Primitive Topology of Geometry Shader Output

Instructions **DCL_OUTPUT_TOPOLOGY**

Syntax `dcl_output_topology` *n*

Description The geometry shader can only emit a single primitive topology from a given shader; the choices are: pointlist, linestrip or trianglestrip. Geometry shaders must contain this declaration. Note that for strip topologies, a single invocation of the geometry shader can emit multiple strips by using the cut instruction. This instruction is required in a geometry shader.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_DCL_OUTPUT_TOPOLOGY
`control`	29:16	Any value of the enumerated type ILTopologyType. See Table 5.4 on page 5-3.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related None.

Declare an Output Register

Instructions **DCL_OUTPUT**

Syntax `dcl_output[_usage(`*`type`*`)] dst`

Description Declares the usage of a shader output register. `dst` must be of type IL_REGTYPE_OUTPUT and can have a modifier that specifies a component mask (see Section 3.6, “Write Mask,” page 3-3). The component mask can be any subset of [xyzw]. DCL_OUTPUT can declare a superset of the component mask declared for input by the next shader stage, indicating the current shader writes more registers than the next shader stage reads; however, mutually exclusive masks are not allowed. For example, a vertex shader that writes o3.xy means the pixel shader reading only v3.z is invalid, but reading v3.x or v3.y or v3.xy would be valid. An output variable can be declared more than once, so that different components can be given different output types. Also, it is possible to have multiple clip distances. DX10 specifies that any component which is of type “none” must appear in xyzw order after components with non-none types. The component mask must be appropriate to the particular type. For example, the current set of system-generated values are all scalars, so the mask must have only one component.

A system-generated value cannot be output from a stage that is before the place in the pipeline where the hardware normally generates the value. For example, a geometry shader cannot output `IsFrontFace`, and VS cannot output `PrimitiveID`.

Output type is:

Type	IL Enumeration (No EXPORTUSAGE in IL Headers)
Generic	IL_IMPORTUSAGE_GENERIC
Position	IL_IMPORTUSAGE_POSITION
ClipDistance	IL_IMPORTUSAGE_CLIPDISTANCE
CullDistance	IL_IMPORTUSAGE_CULLDISTANCE
PrimitiveID	IL_IMPORTUSAGE_PRIMITIVEID
VertexID	IL_IMPORTUSAGE_VERTEXID
InstanceID	IL_IMPORTUSAGE_INSTANCEID
RenderTargetArrayIndex	IL_IMPORTUSAGE_RENDERTARGET_ARRAY_INDEX
ViewportArrayIndex	IL_IMPORTUSAGE_VIEWPORT_ARRAY_INDEX

Format 0-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_DCL_ODEPTH
`control`	29:16	Any value of the enumerated type ILImportUsage. See Table 5.9 on page 5-4.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related None.

Declare a Primitive ID

Instructions **DCL_VPRIM (Input Primitive)**

Syntax `dcl_vprim`

Description Declares that the geometry shader intends to use its scalar input register vPrim. For the geometry shader, input primitive data only comes in the form of a scalar (vPrim, no mask). Also, there is no Primitive Data for adjacent primitives available in a geometry shader invocation.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DCL_VPRIM
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related None.

Declare Shared Registers

Instructions **DCL_SHARED_TEMP**

Syntax `dcl_shared_temp src#`

Description Declares the number of shared GPRs used by this kernel (shared for each SIMD). *src0* must be of type IL_REGTYPE_SHARED_TEMP, with the number (#) indicating one more than the maximum used index. For example, `dcl_shared_temp sr4` indicates that sr0, sr1, sr2, and sr3 are used in the shader. The number declared must be smaller than the maximum available in hardware (for example: 128 for the HD4000-family of devices).

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DCL_SHARED_TEMP
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related DCL_OUTPUT.

Declare LDS Size Used in a Shader

Instructions **DCL_LDS_SIZE_PER_THREAD**

Syntax `dcl_lds_size_per_thread n`

Description Declare the space or size of LDS memory (in dwords) to be used in a compute shader. The value must be:

- in dwords
- no greater than 64, and
- a factor of 4.

Format 0-input, 0-output, no additional token.

Opcode

Field Name	Bits	Description
code	15:0	Must be set to IL_OP_DCL_LDS_SIZE_PER_THREAD.
controls	29:16	n = size in dwords
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Example

```
dcl_lds_size_per_thread 8
```

Related None.

Declare the LDS Sharing Mode

Instructions **DCL_LDS_SHARING_MODE**

Syntax `dcl_lds_sharing_mode _wavefrontRel` or `_wavefrontAbs`

Description Local data share (LDS) memory has two sharing mode: wavefront relative or absolute. Relative means each wavefront has its private LDS memory. Absolute means all wavefronts share the same piece of LDS memory. Only used in a compute kernel.

Format 0-input, 0-output, 0 additional token.

Opcode

Field Name	Bits	Description
code	15:0	Must be set to IL_OP_ DCL_LDS_SHARING_MODE.
controls	29:16	Mode: 0 _wavefrontRel 1 _wavefrontAbs
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Examples

```
dcl_lds_sharing_mode _wavefrontRel
dcl_lds_sharing_mode _wavefrontAbs
```

Related None.

Declare the Thread Group Size

Instructions **DCL_NUM_THREAD_PER_GROUP**

Syntax `dcl_num_thread_per_group n1, n2, n3`

Description Specifies the umber of threads per group. The product of the three sizes can be at most 1024. Only used in a compute kernel. For the HD4000-family of devices, n2 and n3 must be 1 because they allow only one dimension.

Format 0-input, 0-output, 1 to 3 additional tokens: up to three unsigned integers representing the literal value for n (n1, n2, n3).

Opcode

Field Name	Bits	Description
code	15:0	Must be set to IL_OP_DCL_NUM_THREAD_PER_GROUP.
controls	29:16	Number of dimensions (1 to 3).
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Examples

```
dcl_num_thread_per_group 5, 10, 1
dcl_num_thread_per_group 5 (which is the same as dcl_num_thread_per_group 5, 1, 1)
```

Related None.

Declare Interpolator Properties

Instructions **DCLPI**

Syntax `dclpi_x(`*comp*`)_y(`*comp*`)_z(`*comp*`)_w(`*comp*`)_center_bias_invert_centered] `*dst*

Description Declares properties of pixel shader named interpolator inputs and the window coordinate register.

A shader cannot use this instruction on the same register more than once. There can be at most one DCLPI per register.

`ximport`, `yimport`, `zimport`, and `wimport` each describe what components of the register are used by the pixel shader. See the enumerated type for more information.

In shaders where PINPUT registers are used, this instruction can only be used to declare a WINCOORD register.

You cannot use a destination modifier with this instruction. The `modifier_present` field of the IL_Dst token must be set to 0.

You cannot use relative addressing with this instruction. The `relative_address` field of the IL_Dst token must be set to 0.

To use both bias and centered, set just bias and adjust the bias values: when invert is not set, add 0.5 to both bias.x and bias.y; when invert is set, add 0.5 to bias.x and -0.5 to bias.y. This optimization reduces instructions generated to support these control fields.

A DCLPI token can be used to declare interpolated outputs in a vertex shader and interpolated inputs in a pixel shader.

Declare Interpolator Properties (Cont.)

Format	Token	Field Name	Bits	Description
	1	code	15:0	IL_OP_DCLPI
		ximport	17:16	Any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		yimport	19:18	Any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		zimport	21:20	Any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		wimport	23:22	Any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		center	24	For interpolator registers, this field specifies if center or centroid sampling is used. If the register_type field of the following IL_Dst token is IL_REGTYPE_WINCOORD, this bit determines whether to use the center (1), or the nearest sample (0), called centroid, within the pixel as the interpolation value when multisampling, since the center might be outside the polygon. For DX10 drivers, this value can be zero or one, depending on the shader. For all other drivers this value must be one.
		bias	25	This field has no effect if the register_type field of the following IL_Dst token is not IL_REGTYPE_WINCOORD. For the window coordinate register, specifies if the compiler generates instructions to add a bias supplied in constants named SC_CONS_SRC_VIEWPORT_BIAS_{X,Y}. Without a bias, the origin is located at the upper-left corner of the screen (DX Style). A bias can be used to move the origin. For example, to move the origin to the lower-left corner of a window (OGL style): • DX must set this to 0 • OGL must set this to 1.
		invert	26	This field does nothing if the register_type field of the following IL_Dst token is not IL_REGTYPE_WINCOORD. If the register_type field of the following IL_DstToken is IL_REGTYPPE_WINCOORD and this bit is set, the compiler inverts (multiplies by -1) the y coordinate of the position. By default, the y axis goes down (DX style), but this bit can be used to change the direction to up (OGL style).

Declare Interpolator Properties (Cont.)

	`centered`	27	This field does nothing if the register_type field of the following IL_Dst token is not IL_REGTYPE_WINCOORD. If the register_type field of the following IL_Dst token is IL_REGTYPE_WINCOORD, this bit specifies the following: 0 defines the origin as the upper-left corner of the RenderTarget. Thus, pixel centers are offset by (0.5f,0.5f) from integer locations on the RenderTarget. This origin makes rendering screen-aligned textures trivial, as the pixel coordinate system is aligned with the texel coordinate system. This choice matches OGL and DX10. 1 defines the origin as the center of the upper-left pixel in the RenderTarget: the origin is (0.5,0.5) away from the upper-left corner of the RenderTarget. Thus, pixel centers are at integer locations. This choice of origin matches DX9. 0 center of the pixel is (0.5,0.5) 1 center of pixel is (0,0)
	reserved	31:28	Must be zero.
2	IL_Dst token (*dst*) where the `register_type` field is set to IL_REGTYPE_INTERP, IL_REGTYPE_FOG, IL_REGTYPE_PRIMCOORD, IL_REGTYPE_TEXCOORD, IL_REGTYPE_PRICOLOR, or IL_REGTYPE_SECCOLOR, or IL_REGTYPE_WINCOORD. The `modifer_present` and `relative_address` fields must be set to 0.		

Related None.

Declare Pixel Shader Input Register

Instructions **DCLPIN**

Syntax dclpin_usage(*op*)_usageIndex(*n*)_x(*comp*)_y(*comp*)_z(*comp*)_w_(comp)[_centroid] *dst*

Description Declares a mapping of a vertex shader output to a pixel shader input. This instruction or a DCLPP instruction must be issued on each PINPUT register before the register is used in a shader. There can be at most one DCLPIN instruction per *usage-usageIndex* pair.

An *enabled* component is a component set to IL_IMPORTSEL_UNDEFINED, IL_IMPORTSEL_DEFAULT0, or IL_IMPORTSEL_DEFAULT1.

A *disabled* component is a component set to IL_IMPORTSEL_UNUSED.

Packed Registers: a PINPUT register can be declared multiple times with this instruction; thus, a single register can have multiple unique *usage-usageIndex* pairs. However, the same component of a register cannot be *enabled* in both declarations. Thus, if in one declaration a component is *enabled*, the component must be *disabled* in the other declaration(s). For example, if vIN3 is declared as having *usage(interp)_usageIndex(1)_x (*),* and vIN3 is also declared as having *usage(interp)_usageIndex(2),* then that declaration must set x*(-).*

If an IL_PrimaryDCLPIN_Mod token is not preset, the shader behaves as if ximport, yimport, zimport, and wimport are set to IL_IMPORTSEL_UNDEFINED and centroid is set to 0.

Only one register can be declared to have the usage IL_IMPORTUSAGE_FOG. In this case, usageIndex must be zero.

It is an error to use this instruction in a vertex shader or a real-time pixel shader.

Note that a shader using a PINPUT register cannot use the INTERP, TEXCOORD, PRICOLOR, SECCOLOR, or FOG registers.

A source modifier cannot be used with this instruction. The `modifier_present` field of the IL_Dst token must be set to 0.

You cannot use relative addressing with this instruction. The `relative_address` field of the IL_Dst token must be set to 0.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DCLPIN The lower five bits of the control field must be set to an acceptable value of the enumerated type ILImportUsage(usage). The next eight bits of the control field must be a unique number for usage that specifies a mapping of usage type to a vertex shader output (usageIndex).
	usage	20:16	Any value of the enumerated type ILImportUsage. See Table 5.9 on page 5-4.
	usageIndex	28:21	Unique number for *usage* which specifies a mapping of *usage* type to a vertex shader output (*usageIndex*), 0 to 255.
	reserved	30:29	Must be zero.
	pri_instruction_modifier	31	If `pri_modifier_present` is set to 1, an IL_PrimaryDCLPIN_Mod token immediately follows this token.

Declare Pixel Shader Input Register (Cont.)

2 Primary pixel shader input register declaration modifier. IL_PrimaryDCLPIN_Mod token described below: Note that the IL_PrimaryDCLPIN_Mod is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token.

Field Name	Bits	Description
ximport	1:0	Specifies if the x component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
yimport	3:2	Specifies if the y component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
zimport	5:4	Specifies if the z component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
wimport	7:6	Specifies if the w component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
centroid	8	Specifies if this value represents a value at the pixel centroid or the center 0 Value at the center. 1 Value at the centroid.
origin	9	This field is ignored if the import usage is not IL_REGTYPE_WINCOORD. For the window coordinate register, specifies if the value is relative to the upper-left corner of the window or the lower-left corner of the window. If the import usage is IL_IMPORTUSAGE_WINCOORD, then: 0 The value of the IL_REGTYPE_WINCOORD register is relative to the upper-left-hand corner of the window (DX style). 1 The value of the IL_REGTYPE_WINCOORD register is relative to the lower-left-hand corner of the window (OpenGL style).
constant	10	Constant interpolation.
no_perspective	11	Do not perform perspective divide during interpolation.
reserved	31:12	Must be zero.

3 IL_Dst token (*dst*) where the register_type field is set to IL_REGTYPE_PINPUT. The modifier_present and relative_address fields must be set to 0.

Related None.

Map Interpolator Register to Realtime Interpolator Parameter

Instructions **DCLPP**

Syntax `dclpp_param(`*`n`*`)` *`dst`*

Description Maps a PINPUT register to a realtime interpolator parameter. This instruction can be used only in a real-time pixel shader. It is an error to use this instruction in a normal pixel shader or vertex shader. This instruction must be issued on each PINPUT register before the register is used in a real-time pixel shader. There can be at most one DCLPP per register.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DCLPP
	`param`	23:16	Specifies the realtime state register to which the register is mapped (`param`).
	reserved	31:24	Must be zero.
2	IL_Dst toke (*dst*), where *register_type* is set to IL_REGTYPE_PINPUT. The *modifier_present* and *relative_address* fields must be set to 0.		

Related None.

Declare Texture Properties

Instructions **DCLPT**

Syntax `dclpt_stage(`*n*`)_type(`*op*`)_coordmode(`*mode*`)`

Description Declares properties of a texture stage. Do not use this instruction for DX10.

This instruction must be issued on each stage if a TEXLD, TEXLDD, TEXLDB, TEXLDMS, TEXWEIGHT, PROJECT, or LOD instruction is issued on the stage.

There can be at most one DCLPT per *stage* value.

A shader cannot use this instruction on the same stage more than once.

If *type* is IL_USAGE_PIXTEX_UNKNOWN, this instruction indicates that the texture type of the texture on the stage/unit indicated by *stage* is not known at shader-create time

If coordmode is set to IL_TEXCOORDMODE_NORMALIZED, this instruction indicates that the texture coordinate is normalized in any subsequent TEXLD, TEXLDB, or LOD instruction for the texture on *stage*.

If coordmode is set to IL_TEXCOORDMODE_UNNORMALIZED, this instruction indicates that the texture coordinate is not normalized in any subsequent TEXLD, TEXLDD, TEXLDB, TEXLDMS, TEXWEIGHT, PROJECT, or LOD instruction for the texture on *stage* when the wrap mode set in state for the stage/unit is set to clamp-to-border, clamp-half-way-to-border, or clamp-to-edge. In this case, the x texture coordinate ranges from 0.0 to the width of the texture, the y texture coordinate ranges from 0.0 to the height of the texture, and the z texture coordinate ranges from 0.0 to the depth of the texture.

If coordmode is set to IL_TEXCOORDMODE_UNKNOWN, the value of AS_TEX_DENORM_N (*stage*) determines if the texture coordinate used in any subsequent TEXLD, TEXLDD, TEXLDB, TEXLDMS, TEXWEIGHT, PROJECT, or LOD instructions are normalized at shader run time.

If cleartype mode is set, the compiler multiplies the result of any fetch on this texture by 1/(kernel height * width). This is used for DX9 only.

Format

Field Name	Bits	Description
code	15:0	IL_OP_DCLPT
stage	23:16	Stage or unit number.
type	26:24	Any value of the enumerated type ILPixTexUsage. See Table 5.19 on page 5-10.
coordmode	28:27	Any value of the enumerated type ILTexCoordMode. See Table 5.18 on page 5-10.
cleartype mode	29	1 = yes, 0 = no.
reserved	31:30	Must be zero.

Related None.

Declare an Input Buffer

Instructions **DCL_RESOURCE**

Syntax dcl_resource_id(*n*)_type(*pixtexusage*[,unnorm])_fmtx(*fmt*)_fmty(*fmt*)_fmtz(*fmt*)_fmtw(*fmt*)

Description Declares an input buffer, specifies its dimension, and assigns it to a resource number. It identifies the resource type as a Buffer, Texture1D, Texture1DArray, Texture2D, Texture2DArray, Texture3D or TextureCube. Resources of type buffer can be used in an ld instruction. Resources of type texture * can as used in both ld and sample* instructions.

Use the ILPixTexUsage enumeration for this field. The instruction can also indicate if a texture resource has been normalized. Return types identify the data type fetched from the input buffer on a per-component basis.

Declares an input buffer, assigns it a number, and specifies its dimension.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DCL_RESOURCE
	id	23:16	Resource id number, 0 to 255.
	type	27:24	Must be set to any value of the enumerated type ILPixTexUsage (see Section 5.18, “ILPixTexUsage,” page 5-10. Possible types: Buffer, Texture1D, Texture1DArray, Texture2D, Texture2DArray, Texture3D, or TextureCube. Use the ILPixTexUsage enumeration for this field. Resources of type buffer can be used in an ld instruction. Resources of type texture * can as used in both ld and sample* instructions.
	reserved	30:24	Must be zero.
	unnorm	31	0: Texture is normalized. 1: Texture is not normalized. The optional _unnorm option in the textural source can be used to specify this bit. The _unnorm option in the textural source can be used to specify this bit.
2	The return type identifies the data type fetched from the input buffer. Return types are four groups, each of three bits that must be set to any value of the enumerated type ILElementFormat. See Section 5.26, “ILElementFormat,” page 5-13. Return-types are specified on a per-component basis, DX10 specification has no need to repeat identical return types; however, IL requires the type to be repeated all four times.		
	reserved	19:0	Must be zero.
	fmtx	22:20	x-component format.
	fmty	25:23	y-component format.
	fmtz	28:26	z-component format.
	fmtw	31:29	w-component format.

Example

```
dcl_resource_id(1)_type(1d,unnorm)_fmtx(float)_fmty(float)_fmtz(float)_fmtw(float)
```

Related None.

Declare Mapping for Vertex Shader Inputs

Instructions **DCLV**

Syntax `dclv_elem(`*n*`)_x(`*comp*`)_y(`*comp*`)_z(`*comp*`)_w(`*comp*`)` *dst*

Description Declares a mapping between vertex buffer elements (logical streams) set through state and vertex shader inputs. It is an error to use this instruction in a pixel shader.

The INITV and DCLV instruction are mutually exclusive for a given VERTEX register. There can be at most one DCLV per register. Use only registers of type TEMP and VERTEX. Each VERTEX register must either be declared at least once with a DCLV, or initialized with an INITV before it is used as a source in any instruction.

The value of `elem` corresponds to vertex buffer element *n.*

An *enabled* component is a component set to IL_IMPORTSEL_UNDEFINED, IL_IMPORTSEL_DEFAULT0, or IL_IMPORTSEL_DEFAULT1.

A *disabled* component is a component set to IL_IMPORTSEL_UNUSED.

Packed Registers: A VERTEX register can be initialized multiple times with this instruction; thus, a single register can be mapped to multiple unique vertex buffer elements. However, the same component of a register cannot be used in both declarations: if in one declaration a component is *enabled*, the component must be *disabled* in the other declaration(s) of the register.

The i^{th}-enabled component receives the i^{th} piece of data of the vertex buffer element specified by `elem` if it exists. If the data does not exist (the dimension of the vertex buffer element specified in state is less than i), the i^{th} component receives the default value specified in this instruction.

Do not use a destination modifier with this instruction. The `modifier_present` field of the IL_Dst token must be set to 0.

Do not use relative addressing with this instruction. The `relative_address` field of the IL_Dst token must be set to 0.

Operation:

```
VECTOR v;
for (i=0; i < 4; i++)
{
    if(i < ElementDimension(elem)) # AS_VS_DECL_TYPE_DIMENSION_N(elem)
    v[i]=FetchData(elem,currentIndex,i);
else
{
    v[i]=Default(i);
}}
WriteResult(v, dst);
```

Declare Mapping for Vertex Shader Inputs (Cont.)

Format	Token	Field Name	Bits	Description
	1	code	15:0	IL_OP_DCLV
		elem	21:16	Vertex buffer element, 0 to 63. These bits specify the vertex buffer element from which vertex data is loaded/fetched (elem).
		reserved	30:22	Must be zero.
		pri_modifier_present	31	If pri_modifier_present is set to 1, an IL_PrimaryDCLV_Mod token immediately follows this token.
	2	Primary vertex shader input register declaration modifier. IL_PrimaryDCLV_Mod token described below. The IL_PrimaryDCLV_Mod is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token.		
		ximport	1:0	Specifies if the x component is enabled for the vertex buffer element specified by elem for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		yimport	3:2	Specifies if the y component is enabled for the vertex buffer element specified by elem for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		zimport	5:4	Specifies if the z component is enabled for the vertex buffer element specified by elem for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		wimport	7:6	Specifies if the w component is enabled for the vertex buffer element specified by elem for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
		reserved	31:8	Must be zero.
	3	IL_Dst token with register_type set to IL_REGTYPE_TEMP or IL_REGTYPE_VERTEX (*dst*). The modifier_present and relative_address field must be set to 0.		
Related	None.			

Declare Vertex Shader Output Register

Instructions **DCLVOUT**

Syntax `dclvout_usage(op)_usageIndex(n)_x(comp)_y(comp)_z(comp)_w(comp) dst`

Description Declares the usage of a vertex shader output register, as well as a mapping of a vertex shader output to a pixel shader input.

This instruction must be issued on each VOUTPUT register before the register is used in a shader. A shader using a VOUTPUT register cannot use the POS, SPRITE, INTERP, TEXCOORD, PRICOLOR, SECCOLOR, or FOG registers.

There can be at most one DCLOUT instruction per usage-usageIndex pair.

An *enabled* component is a component set to IL_IMPORTSEL_UNDEFINED, IL_IMPORTSEL_DEFAULT0, or IL_IMPORTSEL_DEFAULT1.

A *disabled* component is a component set to IL_IMPORTSEL_UNUSED.

Packed Registers: A VOUTPUT register can be declared multiple times with this instruction; thus, a single register can have multiple unique usage-usageIndex pairs. However, the same component of a register cannot be enabled in both declarations: if in one declaration a component is enabled, the component must be disabled in the other declaration(s). For example, if oOUT3 is declared as having usage(interp)_usageIndex(1)_x (*), and oOUT3 is also declared as having usage(interp)_usageIndex(2), then the second declaration must set x(-).

If an IL_PrimaryDCLVOUT_Mod token is not preset, the shader behaves as if xexport, yexport, zexport, and wexport are set to IL_IMPORTSEL_UNDEFINED.

Only one register can be declared to have the usage IL_IMPORTUSAGE_POS. If loop relative addressing is used on vertex shader outputs, that register can only be the VOUTPUT register number 0.

usageIndex must be if IL_IMPORTUSAGE_POINTSIZE is used.

Only one register can be declared to have the usage IL_IMPORTUSAGE_POINTSIZE. If loop relative addressing is used on vertex shader outputs, that register can only be the VOUTPUT register number 1. usageIndex must be zero if IL_IMPORTUSAGE_POINTSIZE is used.

Only one register can be declared to have the usage IL_IMPORTUSAGE_FOG. usageIndex must be zero in this case if IL_IMPORTUSAGE_FOG is used.

It is an error to use this instruction in a pixel shader.

You cannot use a source modifier with this instruction. The modifier_present field of the IL_Dst token must be set to 0.

You cannot use relative addressing with this instruction. The relative_address field of the IL_Dst token must be set to 0.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_DCLVOUT
	usage	20:16	Any value of the enumerated type ILImportUsage. See Table 5.9 on page 5-4. The first five bits of the must be set to an acceptable value of the enumerated type ILImportUsage (usage). The next eight bits must be a unique number for usage which specifies a mapping of usage type to a pixel shader input (usageIndex).
	usageIndex	28:21	Unique number for *usage* which specifies a mapping of *usage* type to a pixel shader input (*usageIndex*), 0 to 255.
	reserved	30:29	Must be zero.
	pri_modifier_present	31	If pri_modifier_present is set to 1, an IL_PrimaryDCLOUT_Mod token immediately follows this token.

Declare Vertex Shader Output Register (Cont.)

Primary vertex shader output register declaration modifier. IL_PrimaryDCLVOUT_Mod token[1] described below.

Field Name	Bits	Description
xexport	1:0	Specifies if the x component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
yexport	3:2	Specifies if the y component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
zexport	5:4	Specifies if the z component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
wexport	7:6	Specifies if the w component is enabled for the usage-usageIndex for this declaration. Also specifies the default value if the component is enabled. Can be any value of the enumerated type ILImportComponent. See Table 5.10 on page 5-6.
reserved	31:8	Must be zero.

3 IL_Dst token (*dst*) where the register_type field is set to IL_REGTYPE_VOUTPUT. The modifier_present and relative_address fields must be set to 0.

Related None.

1. The IL_PrimaryDCLOUT_Mod is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token.

6.5 Input/Output Instructions

Finish Current Topology

Instructions **CUT**

Syntax `cut`

Description Completes the current primitive topology (if any vertices have been emitted), and any previous primitive topology, then starts a new topology of the type declared by the geometry shader. No vertices are leftover since a geometry shader can only emit pointlist, linestrip and trianglestrip topologies. When CUT is executed, any previously emitted topology by the Geometry Shader invocation is completed. If not enough vertices were emitted for the previous primitive topology, the extra vertices are discarded.

After the previous topology (if any) is completed, CUT causes a new topology to begin, using the topology declared as the geometry shader's output.

CUT can be used only in a geometry shader.This instruction can appear any number of times in a geometry shader and is executed implicitly at the end of an invocation of the geometry shader.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CUT
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related None.

Pixel Kill

Instructions **KILL**

Syntax `kill[_stage(n)_sample] src0`

Description Component-wise test to terminate current pixel shader execution and discard all results.

If sample is set to 1, KILL does not use the actual value of *src0* to test. Instead, the shader performs a KILL based upon a texture sample at the coordinate specified by *src0* on the texture stage/unit specified by stage.

It is an error to use this instruction in a vertex or geometry shader.

To kill based on a subset of the four components, set the swizzle for the component not used for the test to IL_COMPSEL_1.

Operation:

```
VECTOR v;
If(sample == 1)
{
    v = Sample(src0, stage);
}
else
{
    v = EvalSource(src0);
}
if((v[0] < 0.0) || (v[1] < 0.0) || (v[2] < 0.0) || (v[3] < 0.0))
{
    Discard outputs;
    Terminate pixel shader;
}
```

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_KILL
control	29:16	See below
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

control Field Name	Bits	Description
stage	23:16	Texture stage or unit number, 0 to 255, if sample is 1; otherwise, must be zero.
sample	24	0: *src0* is the test value. 1: Test value is sampled from texture stage or unit *stage*. *src0* is the sample coordinate, and the resulting sample is used as the test value.
reserved	29:25	Must be zero.

`IL_Src token (src0)`

`IL_Src_Mod` token: only present if modifier_present field is 1 in previous IL_Src token.

`IL_Src token (src0)`: only present if relative_address field is 1 in the preceding IL_Src or IL_Dst token.

Related DISCARD_LOGICALNZ, DISCARD_LOGICALZ.

Discard Results Based on Integer

Instructions **DISCARD_LOGICALNZ, DISCARD_LOGICALZ**

Syntax

Opcode	Syntax	Description
IL_OP_DISCARD_LOGICALNZ	discard_logicalnz *src0.select_component*	Discard results if all bits in *src0*.{x|y|z|w} ≠ 0.
IL_OP_DISCARD_LOGICALZ	discard_logicalz *src0.select_component*	Discard results if all bits in *src0*.{x|y|z|w} == 0.

Description

Conditionally flags results of pixel shader to be discarded when the end of the program is reached. This instruction flags the current pixel as terminated, while continuing execution, so that other pixels executing in parallel can obtain gradients, if necessary. Although execution continues, all pixel shader output writes before or after the discard_* instruction are discarded.

The discard_* instruction can be present inside any flow control construct.

Multiple discard instructions can be present in a pixel shader, and if any is executed, the pixel is terminated.

Can be used only in a pixel shader.

Format 1-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	See Opcode part of *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None. KILL.

Emit a Vertex

Instructions **EMIT**

Syntax emit

Description Causes all declared o# registers to be read out of a geometry shader to generate a vertex. Multiple EMIT instructions are used to generate a primitive. Any number of EMIT instructions can appear in a geometry shader, including within flow control. This instruction can be used only in a geometry shader.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EMIT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Emit Followed by Cut

Instructions **EMIT_THEN_CUT**

Syntax `emitcut`

Description Operation is the same as an EMIT instruction immediately followed by a CUT instruction. It has the same restrictions as the union of restrictions for the EMIT and CUT commands.

Format 0-input, 0-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EMIT_THEN_CUT
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related CUT, EMIT.

LOAD from Buffer

Instructions **LOAD**

Syntax `load_resource(`*n*`)[_aoffimmi(`*u,v,w*`)]` *dst, src0*

Description Simplified alternative to the “sample” instruction. Using the provided signed integer address, LOAD fetches data from the specified buffer or texture without any filtering (for example, point sampling). The source data can come from any resource type except TextureCube.

src0 must specify a single component used as the address.

DX10 allows an output swizzle on the ld instruction. IL requires an additional move if the swizzle is used.

Unlike SAMPLE, LOAD can fetch data from buffers. The buffer with the data being fetched is identified by the resource id stored in the control field.

src0 provides the set of texture coordinates needed to sample the texture in the form of signed integers.

If the *src*Address is out of the range [0, (#texels in dimension -1)], the load returns 0.

src0.a (post-swizzle) always provides a signed integer mipmap level. If the value is out of range [0, (num miplevels in resource-1)], the load returns 0. If the resource has no mipmaps, *src0*.a is ignored.

src0.gb (post-swizzle) are ignored for buffers and Texture1D (non-Array). *src*Address.b (post-swizzle) is ignored for Texture1D Arrays and Texture2Ds.

For Texture1D Arrays, *src0*.g (post-swizzle) provides the array index as a signed integer. If the value is out of range of the available array (indices [0, (Array size-1)]), the load returns 0.

For Texture2D Arrays, *src0*.b (post-swizzle) provides the array index; otherwise, it has the same semantics as Texture1D, described above.

LOAD from Buffer (Cont.)

Address Offset

- If the optional `_aoffimmi` suffix is included (bit 29 of the Opcode token is set), a 32-bit value containing the packed offsets immediately follows the Opcode token (and modifier, if present).
- The `optional` field indicates that the texture coordinates for the load are to be offset by the provided immediate texel space integer constant values. The literal values must be a set of unnormalized (texel space) values represented as signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5]. *src1* can be used with all resources, including Texture1D/2D Arrays and Texture3D; however, it cannot be used with a resource of type TextureCube.
- The offsets are added to the texture coordinates, in texel space, relative to the miplevel being accessed by the load.
- Address offsets are not applied along the array axis of Texture1D/2D Arrays.
- `_aoffimmi` v,w components are ignored for Buffers, and Texture1Ds.
- `_aoffimmi` w component is ignored for Texture2Ds.

Return Type Control

The data format returned by LOAD to the *dst* register is determined in the same way as described for the sample instruction; it is based on the format bound to the Resource parameter (n).

As with the SAMPLE instruction, returned values for load are four-component vectors (with format-specific defaults for components not present in the format).

DX10 allows a swizzle on the resource that is used to determine how to swizzle the four-component result back from the texture load, after which `.mask` on *dst* is used to determine which components in *dst* is updated. The equivalent effect can be achieved in IL by adding an extra move after LOAD.

The resource cannot be a TextureCube or a Constant Buffer.

Out-of-bounds access on any axis always results in 0 as the result of the memory fetch.

Formats 1-input, 1-output.

Opcode

Field Name	**Bits**	**Description**		
code	15:0	IL_OP_LOAD		
`control`	29:16	**Field Name**	**Bits**	**Description**
		`resource`	23:16	resource_id.
		`sampler`	27:24	sampler_id.
		reserved	28:24	Must be zero.
		`aoffimmi`	29	0 = `aoffimmi` does not exist.
				1 = `aoffimmi` exists.
`sec_modifier_present`	30	Must be zero.		
`pri_modifier_present`	31	Must be zero.		

Related SAMPLE.

Get Texture Mipmap Level of Detail

Instructions **LOD**

Syntax `lod_stage(n) dst, src0`

Description Uses the provided texture coordinate to determine the computed mipmap level(s) sampled from at *src0*. The LOD value returned includes any clamping and biasing defined by the texture currently bound to the specified texture unit. This instruction cannot be used on a stage that has not been declared with a DCLPT instruction.

You can project using the `divComp` source modifier on *src0*. If the `divComp` source modifier is used, and the component to divide by is negative, the result of this instruction is undefined (if IL_DIVCOMP_Y is used, and the second component of *src0* is negative, the results of this texture load is undefined).

Operation:

src0 is the texture coordinate with which the mipmap LOD is determined. The control field of the IL_OpCode token specifies the texture stage to determine the LOD. Associated with the sampler specified by control are 1) a texture, 2) the texture's dimension, and 3) all filter, wrap, bias, and clamp states. The LOD value is replicated on each component of *dst*.

Format 1-input, 1-output.

Opcode

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_LOD
			The first eight bits specify the texture stage/unit from which to determine the texture's level of detail (stage).
	control	29:16	**Field Name Bits** **Description**
			stage 23:16 Texture stage or unit number.
			reserved 29:24 Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.
2	IL_Dst token (*dst*)		
3	IL_Dst_Mod token is present only if the modifier_present field is 1 in previous IL_Dst token.		
4	IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.		
5	IL_Src token (*src0*).		
6	IL_Src_Mod token is present only if the modifier_present field is 1 in previous IL_Src token.		
7	IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.		

Related None.

Export Data to Memory Stream

Instructions **MEMEXPORT**

Syntax `memexport_exportStream(`*`n`*`)_elemOffset(`*`i`*`)` *`src0`*`, `*`src1`*

Description Exports a value in *src1* to a memory stream. *exportStream* corresponds to the number of the export stream buffer to which data is written. *elemOffset* specifies the offset in terms of elements (not bytes or dwords) to which data is written. *src0*.x contains the index to which data is written. *src1* contains the data to write to the export stream.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_MEMEXPORT
	`exportStream`	21:16	Export stream number, 0 to 16
	reserved	22	Must be zero.
	`elemOffset`	28:23	Offset from the start of the export stream in elements.
	reserved	31:29	Must be zero.

2 IL_Src token (*src0*).

3 IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.

4 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src token.

5 IL_Src token (*src1*).

6 IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.

7 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src token.

Related MEMEXPORT, SCATTER_QUAD.

Import Memory from Stream

Instructions **MEMIMPORT**

Syntax `memimport_elem(`*`n`*`)` *`dst, src0`*

Description Imports data from buffers set externally. This instruction provides an alternate means of fetching vertex element data using an arbitrary index. Refer the DCLV instruction for normal operation. This instruction typically is used only for higher-order surface shaders (see the INITV instruction).

In a vertex shader:

- the value of *elem* corresponds to vertex buffer element *n*.
- if the vertex buffer corresponding to *elem* is disabled in state, apply defaults when this instruction is executed.

In a pixel shader

- the value of *elem* corresponds to pixel buffer element *n*.
- if the pixel buffer corresponding to *elem* is disabled in state, apply defaults when this instruction is executed.

The first component of *src0* (*src0.*x) contains the index for the element from which to fetch. The value is floored before it is used.

dst contains the data to be fetched from memory.

dst can only be a TEMP register.

Operation:

```
VECTOR v;
v = Fetch(importElement, FLOOR(src0.x));
WriteResult(v, dst);
```

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_MEMIMPORT The lower six bits are set to a number between 0 and 16 inclusive, specifying the import element from which memory is loaded/fetched (`elem`).
	`elem`	21:16	Element number, 0 to 16
	reserved	31:22	Must be zero.
2	IL_Dst token (*dst*) where `register_type` is set to IL_REGTYPE_TEMP.		
3	IL_Dst_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Dst token.		
4	IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.		
5	IL_Src token (*src0*).		
6	IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.		
7	IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.		

Related MEMEXPORT, SCATTER_QUAD.

Query Information from a Resource

Instructions **RESINFO**

Syntax `resinfo_resource(n)[_uint] dst, src0`

Description *src0.*x is the 2's complement integer scalar.

DX10 supports both destination swizzles and repFloat modifiers on this instruction. Clients must insert additional IL instructions to achieve the same effect.

dst receives <*width*, *height*, *depth*, *total mip count*>.

If *src0.*x is out of the range of the available number of miplevels in the resource, [0, (*numMipLevels* – 1)], then resinfo returns [0, 0, 0, *total mip count*].

The returned width, height, and depth values are for the mip-level selected by *src0.*x; they are in number of texels, independent of texel data size.

Returned values are all floating point, unless the control field is negative (_uint modifier is used), in which case the returned values are integers.

dst.w always receives total-mip-count (if .w is included in write mask). Thus, the total-mip-count is independent of *src0.*x.

DX10 allows a destination swizzle. IL, however, requires an extra move to allow the returned values to be swizzled arbitrarily before they are written to the destination.

If the resource is not a Texture3D or Texture1D/2D with Array > 1, then depth is zero.

For a Texture1D/2D with Array > 1 or Texture3D, depth always represents the number of array slices.

If the resource is a Buffer or Texture1D, the height is zero, unless it is a Texture1D Array, in which case height is the number of array slices.

If the resource is a TextureCube, then width and height represent individual cube face dimensions, and depth is zero.

If *src0.*x is out of the range of the available number of miplevels in the resource, then resinfo returns [0, 0, 0, *total mip count*].

Formats 1-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_RESINFO		
control	29:16	**Field Name**	**Bits**	**Description**
		resource_id	23:16	Resource ID
		return type	24	0x0: Return type is Float.
				0x1: Return type is Integer.
		reserved	29:25	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related None.

Query Information from a Resource

Instructions **SAMPLEINFO**

Syntax `sampleinfo_resource(n)[_uint] dst, src0`

Description *dst* receives a number in components x. The returned value is floating point, unless the control field is negative (_uint modifier is used), in which case the returned value is integers. If the resource is not a multi-sample resource and not a render target, the result is 0.

Formats 0-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLEINFO		
control	29:16	**Field Name**	**Bits**	**Description**
		resource_id	23:16	Resource ID
		return type	24	0x0: Return type is Float.
				0x1: Return type is Integer.
		reserved	29:25	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related None.

Query Position Information from a Resource

Instructions **SAMPLEPOS**

Syntax `samplepos_resource(n)[_uint] dst, src0`

Description The returned value is a float4 (x,y,0,0) indicating where the sample is located. If the resource is not a multi-sample resource and not a render target, the result is 0.

Formats 1-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLEPOS		
control	29:16	**Field Name**	**Bits**	**Description**
		resource_id	23:16	Resource ID
		return type	24	0x0: Return type is Float.
				0x1: Return type is Integer.
		reserved	29:25	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related None.

Sample Data from Resource with Filter

Instructions **SAMPLE**

Syntax `sample_resource(n)_sampler(m)[_aoffimmi(u, v, w)] dst, src0`

Description Samples data from the specified Element/texture using the filtering mode identified by the given sampler. The source data can come from any resource type other than buffers. *src0* provides the set of texture coordinates needed sampling as floating point values, referencing normalized space in the texture. Address wrapping modes (wrap, mirror, clamp, border, etc.) are applied to texture coordinates outside [0...1] range, taken from the sampler state, and applied after any address offset is applied to texture coordinates (see Address Offset section, below).

For Texture2D Arrays, *src0*.b (post-swizzle) selects the Array Slice from which to fetch and uses the same semantics described for Texture1D Arrays.

Address Offset

The optional `_aoffimmi` suffix is specified by the `addroff_present` bit in IL_SampleOpCode. If set, a 32 bit value containing the packed offsets immediately follows the opcode token and modifier, if present.

The optional *address offset by immediate integer* indicates that the texture coordinates for the sample are offset by the provided immediate texel space integer constant values. The literal values must be a set of unnormalized (texel space) values represented as Signed fixed point with 1 bit of fraction (7.1) that allows ranges from [–64,63.5]. This modifier is defined for all resources, including Texture1D/2D Arrays and Texture3D; but it is undefined for TextureCube.

The offsets are added to the texture coordinates, in texel space, relative to each mip level being accessed. Thus, even though texture coordinates are provided as normalized float values, the offset applies a texel-space integer offset. Address offsets are not applied along the array axis of Texture1D/2D Arrays.

The v and w components are ignored for buffers and Texture1Ds.

The w component is ignored for Texture2Ds.

Address wrapping modes (wrap, mirror, clamp, border, etc.) from the sampler state are applied after any address offset is applied to texture coordinates.

Return Type Control

The data format returned by sample to the destination register is determined by the resource format (`WGFFMT*`). For example, if the resource format is `WGFFMT_A8B8G8R8_UNORM_SRGB`, the sampling operation converts sampled texels from gamma 2.0 to 1.0, applies filtering, and the result are written to the destination register as floating point values in the range [0,1].

Returned values are 4-vectors (with format-specific defaults for components not present in the format).

DX10 supports an output swizzle in this instruction. To achieve the same effect in IL, use an extra move.

Sample Data from Resource with Filter (Cont.)

LOD Calculation

See the deriv_rtx and deriv_rty instructions on how derivatives are calculated while determining LOD for filtering. The sample instruction implicitly computes derivatives on the texture coordinates using the same definition that the deriv* Shader instructions use. This does not apply to sample_l or sample_g instructions; for these, LOD or gradients are provided directly by the application.

Miscellaneous Details

src0.w (post-swizzle) must be zero. *src0* provides the set of texture coordinates to perform the sample as floating point values referencing normalized space within the texture. Address wrapping modes (wrap, mirror, clamp, border, etc.) are applied for texture coordinates outside the [0, 1] range, taken from the sampler state, and applied after any address offset (see subsection, above) is applied to texture coordinates.

The information required for the hardware to perform sampling is split into two orthogonal parts. The first part (texture register), provides source data type information, including if the texture contains SRGB data; also, it references the memory being sampled. The second part (sampler register), defines the filtering mode to apply to the sample operation.

For Texture1D Arrays, *src0*.y (post-swizzle) selects which Array Slice to fetch from. This is treated as a scaled float value, as opposed to the normalized space for standard texture coordinates; and a round-to-nearest even is applied on the value, followed by a clamp to the available BufferArray range.

Formats 1-input, 1-output or 3-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id, 0 to 255.
		sampler	27:24	sampler_id, 0 to 15.
		reserved	28	Must be zero.
		aoffimmi	29	0 = aoffimmi does not exist.
				1 = aoffimmi exists.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE_B, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_L.SAMPLE_L.

Sample Data from Resource with Filter and Bias

Instructions **SAMPLE_B**

Syntax `sample_b_resource(`*`n`*`)_sampler(`*`m`*`)[_aoffimmi(`*`u, v, w`*`)] dst, src0 , src1`

Description Samples data from the specified element/texture using the filtering mode identified by the given sampler. The source data can come from any resource type except for buffers. An additional bias is applied to the level of detail computed as part of the instruction execution.

SAMPLE_B is like the SAMPLE instruction with the addition of applying the specified srcLOBBias (in *src1*) value to the level of detail value prior to selecting the mip map.

src1 must be either a literal value or a replicated single component.

SAMPLE_B has the same restrictions as the SAMPLE instruction.

Formats 2-input, 1-output or 4-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_B		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	28	Must be zero.
		aoffimmi	29	0 = aoffimmi does not exist.
				1 = aoffimmi exists.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_L.

Sample Data from Resource with Filter and Gradient

Instructions **SAMPLE_G**

Syntax `sample_g_resource(`*`n`*`)_sampler(`*`m`*`)[_aoffimimmi(`*`u, v, w`*`)]` *`dst, src0, src1, src2`*

Description Samples data from the specified element/texture using the filtering mode identified by the given sampler. The source data can come from any Resource Type, other than Buffers.

SAMPLE_G behaves exactly as the “sample” instruction, except that gradients for the source address in the x direction and the y direction are provided by extra parameters, *src1* and *src2*, respectively.

The x, y, and z components of *src1*/*src2* (post-swizzle) provide du/dx, dv/dx and dw/dx. The w component (post-swizzle) is ignored.

The w component (post-swizzle) is ignored.

This instruction has the same restrictions as the SAMPLE instruction.

Formats 3-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_G		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_B, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_L.

Sample Data from Specified Memory Using Given Sampler. Source Data can come from any Resource Type

Instructions **SAMPLE_L**

Syntax `sample_l_resource(`*n*`)_sampler(`*m*`)` *dst, src0, src1*

Description This is identical to the SAMPLE instruction (6-58), except that the level of detail (LOD) is provided directly by the application as a scalar value, representing no anisotropy. This instruction also is available in all programmable shader stages, not only the pixel shader (as with SAMPLE).

It samples the texture using *src1*.x to choose the LOD. If the LOD value is negative, the 0'th (biggest map) is chosen with MAGFILTER applied. Since *src1*.x is a floating point value, the fractional value is used to interpolate (if MIPFILTER is linear) between two mip levels.

This instruction ignores address gradients (filtering is isotropic).

See the description of the SAMPLE instruction for operational details of this instruction other than the LOD calculation.

Note that when used in the pixel kernel, `sample_l` implies the choice of LOD is per-pixel, with no effect from neighboring pixels.

Formats 2-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_L		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE_B, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_L, SAMPLE.

Sample Data from Resource with Filter and Comparison

Instructions **SAMPLE_C_LZ**

Syntax `sample_c_lz_resource(`*m*`)_sampler(`*n*`)_aoffimmi(`*u*`, `*v*`, `*w*`) dst, src0, src1`

Description Performs a comparison filter. SAMPLE_C mainly provides a building-block for Percentage-Closer Depth filtering. The 'c' in SAMPLE_C stands for comparison.

src0 is the index. *src1*.x contains the reference value.

Same as SAMPLE_C, except LOD is 0, and derivatives are ignored (as if they are 0).

The LZ stands for level-zero. Because derivatives are ignored, this instruction is available in vertex and geometry shaders. It can also be used inside of control flow.

Formats 2-input, 1-output or 4-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_B		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_B, SAMPLE_C, SAMPLE_G, SAMPLE_L.

Sample Data from Resource with Filter and Comparison

Instructions **SAMPLE_C**

Syntax `sample_c_resource(m)_sampler(n)_aoffimimmi(u, v, w) dst, src0, src1`

Description Performs a comparison filter. *src0* is the index. *src1* contains the reference value. The primary purpose for SAMPLE_C is to provide a building-block for percentage-closer depth filtering. The C in SAMPLE_C stands for comparison.

The operands to SAMPLE_C are identical to those of SAMPLE, except for an additional float32 source operand, *src1*.

If SAMPLE_C is used with a resource that is not a texture1D/2D/Cube, or an unsupported format, then this instruction produces undefined results. It also produces undefined results if used with texture arrays.

When the SAMPLE_C instruction is executed, the hardware uses the current sampler's comparison function to compare *src1*.x against the corresponding texture value at each filter “tap” location (texel) that the currently configured texture filter covers, based on the provided coordinates (*src0*).

The comparison occurs after the source texel's x-component is converted to float32, prior to filtering texels.

For texels that fall off the resource, the x-component value is determined by applying the address modes (and BorderColorR, if in Border mode) from the sampler.

Each comparison that passes returns 1.0f as the x-component value for the texel; each comparison that fails returns 0.0f as the x value for the texture. Then, filtering occurs as specified by the sampler states, operating only in the x-component, and returning a single scalar filter result to the shader (replicated to all masked *dst* components).

The use of SAMPLE_C is orthogonal to all other general-purpose filtering controls (SAMPLE_C works with the other general-purpose filter modes). What SAMPLE_C changes the behavior of the general-purpose filters so that the values being filtered all become 1.0f or 0.0f (comparison results).

See the SAMPLE instruction description for operation of this instruction other than those specified here.

Formats 2-input, 1-output or 4-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_C		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_B, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_L.

Sample Data From Element/Texture With Gradient Using Filter

Instructions **SAMPLE_C_G**

Syntax `sample_c_g_resource(n)_sampler(m)_aoffimimmi(u,v,w) dst, src0, src1, src2, src3`

Description Sample data from the specified element/texture using the filtering mode identified by the given sampler. The source data can come from any resource type other than buffers.

Behaves like the SAMPLE_C, except that gradients for the source address in the x direction and the y direction are provided through *src2* and *src3*, respectively.

The x, y, and z components of *src2*/*src3* (post-swizzle) provide du/dx, dv/dx, and dw/dx. The w component (post-swizzle) is ignored.

This instruction has the same restrictions as the SAMPLE_C and SAMPLE_G instructions.

Format 4-input 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_C_G		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_B, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_L.

Sample Data from Element/Texture with LOD

Instructions **SAMPLE_C_L**

Syntax sample_c_l_*resource(n)_sampler(m)_aoffimimmi(u,v,w) dst, src0 , src1, src2*

Description Sample data from the specified element/texture using the filtering mode identified by the given sampler. The source data can come from any resource type other than buffers.

Behaves like the SAMPLE_C instruction, except that LOD is provided directly through *src2*.x.

This instruction has the same restrictions as the SAMPLE_C and SAMPLE_L instructions.

Format 3-input 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_SAMPLE_C_L		
control	29:16	**Field Name**	**Bits**	**Description**
		resource	23:16	resource_id.
		sampler	27:24	sampler_id.
		reserved	29:28	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related LOAD, SAMPLE, SAMPLE_B, SAMPLE_C, SAMPLE_C_LZ, SAMPLE_G, SAMPLE_C_G.

Texture LOAD

Instructions **TEXLD**

Syntax

```
texld_stage(n)_centroid_shadowmode(op)_mag(op)_min(op)_volmag(op)_volmin(op)_mip(op)_aniso(op)_lodbias(f)_xoffset(x)_yoffset(y)_zoffset(z) dst, src0
```

Description Samples a texture specified by *stage* at the coordinates specified by the *src0* register.

This instruction cannot be used on a *stage* that has not been declared with a DCLPT instruction. Also, it cannot be used on a stage set to IL_USAGE_PIXTEX_2DMSAA by a DCLPT instruction. The value of stage corresponds to the stage/unit.

The coordinates can be projected using the divComp source modifier on *src0.*

- If the divComp source modifier is used, and the component to divide by is negative, the result of this instruction is undefined. That is, if IL_DIVCOMP_Y is used, and the second component of *src0* is negative, the results of this instruction are undefined.

If centroid is 1, sampling is done based on the pixel centroid, not center.

The lodbias value specifies a constant value to bias the mipmap from which to load for this instruction. This value is added to the bias value set in the state (the value set through AS_TEX_LODBIAS_N(*stage*)), and the bias value in the fourth component of *src2*.

The following determines the mipmap level(s) from which to sample:

- The *computed LOD* is the mipmap level-of-detail determined based on the ratio of texels in the base texture to the pixel.
- The *instruction LOD* is the value specified by the *lodbias* parameter in the IL_PrimaryTEXLD_Mod token).
- The *minLOD* is the state-based floating point minimum mipmap LOD value.
- The *maxLOD* is the state-based floating point maximum mipmap LOD value.
- The *minLevel* is the smallest mipmap level specified by state to use.
- The *maxLevel* is the largest mipmap level specified by state to use.
- The mipmap level(s) to sample from are determined by:
  - Adding the *state based LOD* to the *computed LOD.*
  - If LOD clamping is enabled in state, clamping the resulting value to *minLOD* and *maxLOD.*
  - Adding the *instruction LOD*.
  - Clamping the resulting value to *minLevel* and *maxLevel.*

The following pseudocode demonstrates the algorithm used to determine the mipmap level(s) to sample from:

- (initial bias LOD) = (state based bias) + (computed LOD)
- (clamped LOD) = (minLOD) ≤ (initial bias LOD) ≤ (maxLOD)
- (secondary bias LOD) = (clamped LOD) + (instruction LOD)
- (final LOD) = (minLevel) ≤ (secondary bias LOD) ≤ (minLevel)

The mag, min, volmag, volmin, mip, and aniso parameters specify whether (and how) to override filter settings. If the IL_PrimaryTEXLD_Mod token is not present, the filters set through external state are used.

Texture LOAD (Cont.)

The values of xoffset, yoffset, and zoffset are added to the unnormalized values of the first, second, and third components of *src0,* respectively, within the sample mipmap. These values are applied whether or not normalized texture coordinates are used. Clamping policy is obeyed as usual when sampling outside the texture's dimensions using these offset parameters. If the IL_SecondaryTEXLD_Mod token is not present, *xoffset, yoffset,* and *zoffset* default to 0.0.

The shadowMode parameter specifies if this instruction performs a shadow map load (compare the texture value to the z-component of *src0*). shadowMode indicates one of the following:

- A shadow load never occurs.
- A shadow load always occurs.

If a shadow load occurs with this instruction, the mag, min, volmag, volmin, aniso, xoffset, yoffset, and zoffset parameters are ignored.

Operation:

The *src0* provides the texture coordinates for the texture sample. The first eight bits of the control field of the IL_OpCode token specify the texture stage from which to sample. Associated with the stage specified in the control field are: 1) a texture image, 2) the texture's dimension, 3) and all format, filter, wrap, bias, and clamp settings specified in state and through the DCLPT instruction.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_TEXLD
	stage	23:16	Stage or unit number, from which to sample.
	centroid	24	0: Sample on pixel center. 1: Sample on pixel centroid.
	reserved	25	Must be zero.
	shadowmode	27:26	Any value of the enumerated type ILTexShadowMode. See Table 5.20 on page 5-11.
	reserved	29:28	Must be zero.
	sec_modifier_present	30	0: No secondary modifier token is present. 1: IL_SecondaryTEXLD_Mod token immediately follows this token or an IL_PRIMARYTEXLD_Mod token, if bit 31 is set.
	pri_modifier_present	31	0: No primary modifier token is present. 1: IL_PrimaryTEXLD_Mod token immediately follows this token.
2	Primary Texture Load Instruction Modifier token (is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token).		
	Mag	2:0	Specifies how to filter texture values in the S and T directions when a single texel maps to multiple pixels. Can be any value of the enumerated type ILTexFilterMode. See Table 5.21 on page 5-11.
	Min	5:3	Specifies how to filter texture values in the S and T directions when multiple texels map to a single pixel. Can be any value of the enumerated type ILTexFilterMode.
	volmag	8:6	Specifies how to filter texture values in the R direction when the pixel maps to an area less than one texel. Can be any value of the enumerated type ILTexFilterMode.

Texture LOAD (Cont.)

	Volmin	11:9	Specifies how to filter texture values in the R direction when the pixel maps to an area greater than one texel. Can be any value of the enumerated type ILTexFilterMode.
	Mip	14:12	Specifies how to filter values of multiple mipmaps when multiple texels of the base level maps a single pixel. Can be any value of the enumerated type ILMipFilterMode.
	Aniso	17:15	When anisotropic filtering is enabled in the min or mag filter, specifies the maximum number of samples to use for anisotropic filtering. When anisotropic filtering is used for the min and mag filter, this value is the same. Can be any value of the enumerated type ILAnisoFilterMode. See Table 5.22 on page 5-11.
	lodbias	24:18	Specifies a constant mipmap level of detail bias applied to the texture load. Signed fixed point with 4 bits of fraction (3.4), which allows a bias range from [–4, 3.9375]. This value is added to the computed texture LOD value.
	reserved	31:25	Must be zero.

3 Secondary Texture Load Instruction Modifier token (is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token).

Xoffset	7:0	Unnormalized (texel space) values added to the X texture coordinate before sampling. Signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5].
Yoffset	15:8	Unnormalized (texel space) values added to the Y texture coordinate before sampling. Signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5].
Zoffset	23:16	Unnormalized (texel space) values added to the Z texture coordinate before sampling. Signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5].
reserved	31:24	Must be zero.

4 IL_Dst token (*dst*)

5 IL_Dst_Mod token is present only if the modifier_present field is 1 in the previous IL_Dst token.

6 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

7 IL_Src token (*src0*)

8 IL_Src_Mod token is present only if the modifier_present field is 1 in the previous IL_Src token.

9 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

Related TEXLDB, TEXLDD, TEXLDMS.

Biased Texture LOAD

Instructions **TEXLDB**

Syntax

texldb_stage(*n*)[_centroid][_absolute]_shadowmode(*op*)_mag(*op*)_min(*op*)_volmag(*op*)_volmin(*op*)_mip(*op*)_aniso(*op*)_lodbias(*f*)_qualitybias_xoffset(*x*)_yoffset(*y*)_zoffset(*z*) *dst, src0, src1*

Description

Samples a texture specified by stage at coordinates specified by the *src0* register and biased by the value in the fourth component of *src1.*

This instruction cannot be used on a stage that has not been declared with a DCLPT instruction. Also, it cannot be used on a *stage* set to IL_USAGE_PIXTEX_2DMSAA by a DCLPT instruction.

The value of stage corresponds to the stage/unit defined externally.

The fourth component of *src2* (*src2*.w) is a factor in determining the mipmap level from which to sample.

The coordinates can be projected using the divComp source modifier on *src0*.

- If the divComp source modifier is used, and the component to divide by is negative, the result of this instruction is undefined. That is, if IL_DIVCOMP_Y is used, and the second (y) component of *src0* is negative, the result of this instruction is undefined.
- divComp can be set to IL_DIVCOMP_UNKNOWN in this instruction; in this case the component used to divide is specified externally.

If *centroid* is 1, sampling is done based on the pixel centroid, not center.

The *lodbias* value specifies a constant value to bias the mipmap from which to load for this instruction. This value is added to the bias value set in the in state and the bias value in the fourth component (w) of *src2*.

The following is used to determine the mipmap level(s) to sample from:

- The *computed LOD* is the mipmap level of detail determined based on the ratio of texels in the base texture to the pixel.
- The *instruction LOD* is the value specified by the lodbias parameter in the IL_PrimaryTEXLDB_Mod token)
- The state based bias is the texture bias value specified externally.
- The *minLOD* is the state based floating point minimum mipmap LOD value.
- The *maxLOD* is the state based floating point maximum mipmap LOD value.
- The *minLevel* is the smallest mipmap level specified by state to use.
- The *maxLevel* is the largest mipmap level specified by state to use.
- When *absolute* is 0, the computed LOD is a factor in determining the mipmap level(s) to sample from. The mipmap level(s) to sample from are determined by:
  - Adding the fourth component (w) of *src2*, the *state based LOD*, and the *computed LOD*
  - If LOD clamping is enabled in state, clamping the resulting value to *minLOD* and *maxLOD.*
  - Adding the *instruction LOD*.
  - Clamping the resulting value to *minLevel* and *maxLevel.*

The following pseudo code demonstrates the algorithm used to determine the mipmap level(s) to sample from:

```
(initial bias LOD) = (fourth component of src1) + (state based bias) + (computed LOD)
(clamped LOD) = (minLOD) ≤ (initial bias LOD) ≤ (maxLOD)
(secondary bias LOD) = (clamped LOD) + (instruction LOD)
(final LOD) = (minLevel) ≤ (secondary bias LOD) ≤ (minLevel)
```

Biased Texture LOAD (Cont.)

- When *absolute* is 1, the computed LOD is not a factor in determining the mipmap level(s) from which to sample. These level(s) are determined by:
  - Adding the fourth component of *src2* and the *state based LOD*.
  - If LOD clamping is enabled in state, clamping the resulting value to *minLOD* and *maxLOD.*
  - Adding the *instruction LOD*.
  - Clamping the resulting value to *minLevel* and *maxLevel.*

The following pseudo code demonstrates the algorithm used to determine the mipmap level(s) to sample from:

```
(initial bias LOD) = (fourth component of src2) + (state based bias)
(clamped LOD) = (minLOD) ≤ (initial bias LOD) ≤ (maxLOD)
(secondary bias LOD) = (clamped LOD) + (instruction LOD)
(final LOD) = (minLevel) ≤ (secondary bias LOD) ≤ (minLevel)
```

The mag, min, volmag, volmin, mip, and aniso parameters specify whether (and how) to override external filter settings. If the IL_PrimaryTEXLD_Mod token is not present, the filters are set externally.

The values of xoffset, yoffset, and zoffset are added to the unnormalized values of the first, second, and third components of *src0*, respectively, within the sample mipmap. These values are applied whether or not normalized texture coordinates are used. Clamping policy is obeyed when sampling outside the texture's dimensions using these offset parameters. If the IL_SecondaryTEXLD_Mod token is not present, xoffset, yoffset, and zoffset default to 0.0.

The shadowmode parameter specifies if this instruction performs a shadow map load, (compare the texture value to the z-component of *src0).* shadowMode indicates if a shadow load never occurs or always occurs.

See shadow texture load appendix for texture load algorithm.

If a shadow load occurs with this instruction, the *mag*, *min, volmag, volmin, aniso, xoffset, yoffset,* and *zoffset* parameters are ignored.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_TEXLDB
	stage	23:16	Stage or unit number.
	centroid	24	0: Sample on pixel center. 1: Sample on pixel centroid.
	absolute	25	0: the fourth component of *src2* is a relative mipmap. 1: the fourth component of *src2* is an absolute mipmap.
	shadowmode	27:26	Any value of the enumerated type ILTexShadowMode. See Table 5.20 on page 5-11.
	reserved	29:28	Must be zero.
	sec_modifier_present	30	0: No secondary modifier token is present. 1: IL_SecondaryTEXLD_Mod token immediately follows the IL_OP_TEXLDB token or an IL_PrimaryTEXLDB_Mod token if bit 31 is set.
	pri_modifier_present	31	0: No primary modifier token is present. 1: IL_PrimaryTEXLD_Mod token immediately follows the IL_OP_TEXLDB token.

Biased Texture LOAD (Cont.)

2 Primary Texture Load Instruction Modifier token (is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token).

Mag	2:0	Specifies how to filter texture values in the S and T directions when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
Min	5:3	Specifies how to filter texture values in the S and T directions when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
volmag	8:6	Specifies how to filter texture values in the R direction when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
Volmin	11:9	Specifies how to filter texture values in the R direction when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
Mip	14:12	Specifies how to filter values of multiple mipmaps when the pixel maps to an area greater than one texel of the base map. Can be any value of the enumerated type ILMipFilterMode.
Aniso	17:15	When anisotropic filtering is enabled in the min or mag filter, specifies the maximum number of samples to use for anisotropic filtering. When anisotropic filtering is used for the min and mag filter, this value is the same. Can be any value of the enumerated type ILAnisoFilterMode.
lodbias	24:18	Specifies a constant mipmap level of detail bias applied to the texture load. Signed fixed point with 4 bits of fraction (3.4), which allows a bias range from [–4, 3.9375].
qualitybias	25	Specifies if the bias is per pixels or per quad: 0 = Per quad. 1 = Per pixel.
reserved	31:26	Must be zero.

3 Secondary Texture Load Instruction Modifier token (is present only if the sec_modifier_present field is 1 in the previous IL_OpCode token).

Xoffset	7:0	Unnormalized (texel space) values added to the X texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
Yoffset	15:8	Unnormalized (texel space) values added to the Y texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
Zoffset	23:16	Unnormalized (texel space) values added to the Z texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
reserved	31:24	Must be zero.

Biased Texture LOAD (Cont.)

4 IL_Dst token (*dst*)

5 IL_Dst_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Dst token.

6 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.

7 IL_Src token (*src1*)

8 IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.

9 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.

10 IL_Src token (*src2*)

11 IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.

12 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.

Related TEXLDB, TEXLDD, TEXLDMS.

Gradient Texture LOAD

Instructions **TEXLDD**

Syntax `texldd_stage(`*n*`)_centroid_shadowmode(`*op*`)_mag(`*op*`)_min(`*op*`)_volmag(`*op*`)_volmin(`*op*`)_mip(`*op*`)_aniso(`*op*`)_lodbias(`*f*`)_xoffset(`*x*`)_yoffset(`*y*`)_zoffset(`*z*`) dst, src0, src1, src2`

Description *src0* contains the texture coordinate to sample from the specified texture stage.

This instruction cannot be used on a *stage* set to IL_USAGE_PIXTEX_2DMSAA by a DCLPT instruction. Also, it cannot be used on a *stage* that has not been declared with a DCLPT instruction.

The value of *stage* corresponds to the stage/unit defined in state.

The coordinates can be projected using the divComp source modifier on *src0*.

- If the divComp source modifier is used, and the component to divide by is negative, the result of this instruction is undefined. That is, if IL_DIVCOMP_Y is used, and the second component of *src0* is negative, the result of this instruction is undefined.
- divComp can be set to IL_DIVCOMP_UNKNOWN in this instruction; in this case, the component used to divide is specified externally.

If *centroid* is set to 1, sampling is done based on the pixel centroid, not center.

The *lodbias* value specifies a constant value to bias the mipmap from which to load for this instruction. This value is added to the bias value set in the in state and the bias value in the fourth component of *src2*.

The following is used to determine the mipmap level(s) from which to sample.

- The *computed LOD* is the mipmap level of detail determined based on the ratio of texels in the base texture to the pixel.
- The *instruction LOD* is the value specified by the *lodbias* parameter in the IL_PrimaryTEXLD_Mod token)
- The *state based bias* is the texture bias value specified externally.
- The *minLOD* is the state based floating point minimum mipmap LOD value.
- The *maxLOD* is the state based floating point maximum mipmap LOD value.
- The *minLevel* is the smallest mipmap level specified by state to use.
- The *maxLevel* is the largest mipmap level specified by state to use.
- The mipmap level(s) to sample from are determined by:
  - Adding the *state based LOD* to the *computed LOD.*
  - If LOD clamping is enabled in state, clamping the resulting value to *minLOD* and *maxLOD.*
  - Adding the *instruction LOD*.
  - Clamping the resulting value to *minLevel* and *maxLevel.*

The following pseudo code demonstrates the algorithm used to determine the mipmap level(s) to sample from:

- (initial bias LOD) = (state based bias) + (computed LOD)
- (clamped LOD) = (minLOD) $\leq$ (initial bias LOD) $\leq$ (maxLOD)
- (secondary bias LOD) = (clamped LOD) + (instruction LOD)
- (final LOD) = (minLevel) $\leq$ (secondary bias LOD) $\leq$ (minLevel)

The *mag*, *min, volmag, volmin, mip,* and *aniso* parameters specify whether (and how) to override filter settings. If the IL_PrimaryTEXLD_Mod token is not present, the external filters settings are used.

Gradient Texture LOAD (Cont.)

The values of *xoffset, yoffset,* and *zoffset* are added to the unnormalized values of the first, second, and third components of *src0*, respectively, within the sample mipmap. These values are applied whether or not normalized texture coordinates are used. Clamping policy is obeyed when sampling outside the texture's dimensions using these offset parameters. If the IL_SecondaryTEXLD_Mod token is not present, *xoffset, yoffset,* and *zoffset* default to 0.0.

The *shadowmode* parameter specifies if this instruction performs a shadow map load (compare the texture value to the z-component of *src0*). shadowMode indicates if a shadow load never occurs or always occurs.

If a shadow load occurs with this instruction, the *mag*, *min, volmag, volmin, aniso, xoffset, yoffset,* and *zoffset* parameters are ignored.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_TEXLDD
	stage	23:16	Stage or unit number, 0 to 255.
	centroid	24	0: Sample on pixel center. 1: Sample on pixel centroid.
	reserved	25	Must be zero.
	shadowmode	27:26	Any value of the enumerated type ILTexShadowMode. See Table 5.20 on page 5-11.
	reserved	29:28	Must be zero.
	sec_modifier_present	30	0: No secondary modifier token is present. 1: IL_SecondaryTEXLD_Mod token is present.
	pri_modifier_present	31	0: No primary modifier token is present. 1: IL_PrimaryTEXLD_Mod token is present.
2	Primary Texture Load Instruction Modifier token (is present only if the pri_modifier_present field is 1 in the previous IL_OpCode token).		
	Mag	2:0	Specifies how to filter texture values in the S and T directions when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
	Min	5:3	Specifies how to filter texture values in the S and T directions when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
	volmag	8:6	Specifies how to filter texture values in the R direction when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
	Volmin	11:9	Specifies how to filter texture values in the R direction when the pixel maps to an area than one texel. Can be any value of the enumerated type ILTexFilterMode.
	Mip	14:12	Specifies how to filter values of multiple mipmaps when the pixel maps to an area greater than one texel of the base map. Can be any value of the enumerated type ILMipFilterMode.

Gradient Texture LOAD (Cont.)

	Aniso	17:15	When anisotropic filtering is enabled in the min or mag filter, specifies the maximum number of samples to use for anisotropic filtering. When anisotropic filtering is used for the min and mag filter, this value is the same. This can be any value of the enumerated type ILAnisoFilterMode.
	lodbias	24:18	Specifies a constant mipmap level of detail bias applied to the texture load. Signed fixed point with 4 bits of fraction (3.4), which allows a bias range from [–4, 3.9375]. This value is added to the computed texture LOD value.
	reserved	31:25	Must be zero.

3 Secondary Texture Load Instruction Modifier token (is present only if the sec_modifier_present field is 1 in the previous IL_OpCode token.)

Xoffset	7:0	Unnormalized (texel space) values added to the X texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
Yoffset	15:8	Unnormalized (texel space) values added to the Y texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
Zoffset	23:16	Unnormalized (texel space) values added to the Z texture coordinate before sampling. Signed fixed point with 1 bit of fraction (7.1) which allows ranges from [–64, 63.5].
reserved	31:24	Must be zero.

4 IL_Dst token (*dst*)

5 IL_Dst_Mod token is present only if the modifier_present field is 1 in the previous IL_Dst token.

6 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

7 IL_Src token (*src1*)

8 IL_Src_Mod token is present only if the sec_modifier_present field is 1 in the previous IL_OpCode token.

9 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

10 IL_Src token (*src*2)

11 IL_Src_Mod token is present only if the sec_modifier_present field is 1 in the previous IL_OpCode token.

12 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

13 IL_Src token (*src3*)

14 IL_Src_Mod token is present only if the sec_modifier_present field is 1 in the previous IL_OpCode token.

15 IL_Rel_Addr token is present only if the relative_address field is 1 in the preceding IL_Src or IL_Dst token.

Related TEXLD, TEXLDB, TEXLDMS.

Multisample Texture LOAD

Instructions **TEXLDMS**

Syntax `texld_stage(`*n*`)_mag(`*op*`)_min(`*op*`)_aniso(`*op*`)_sample(`*n*`)_xoffset(`*x*`)_yoffset(`*y*`)` *dst, src0*

Description Samples a multi-sample texture specified by *stage* at coordinates specified by the *src0.*

This instruction cannot be used on a *stage* that has not been declared with a DCLPT instruction.

This instruction can only be executed on a stage where the *usage* of the DCLPT instruction is set to IL_USAGE_PIXTEX_2DMSAA.

The value of *stage* corresponds to the stage/unit defined in state.

The first two components of *src0* (*src0*.xy) are used as the texture coordinate to sample from the specified texture stage.

The sum of the third component of *src0*, *src0*.z, and the value specified in the *sample* parameter specifies the single sample to retrieve from the multi-sample buffer when the state based multi-sample filter is set to point filtering.

- The sum of the third component of *src0*, *src0*.z, and the value specified in the *sample* parameter specifies the single sample to retrieve when AS_MSAA_TEX_FILTER_FUNCTION_N(*stage*) is set to POINT.

The coordinates can be projected using the divComp source modifier on *src0.*

- If the divComp source modifier is used, and the component to divide by is negative, the results of this instruction are undefined. For example, if IL_DIVCOMP_Y is used and the second component of *src0* is negative, the result of this instruction is undefined.
- divComp can be set to IL_DIVCOMP_UNKNOWN in this instruction in which case the component used to divide is specified externally.

If the IL_PrimaryTEXLDMS_Mod token is not present, the filters are set externally.

The values of *xoffset* and *yoffset* are added to the unnormalized values of the first, second, and third components of *src0* respectively within the sample mipmap. These values are applied whether or not normalized texture coordinates are used. Clamping policy is obeyed when sampling outside the texture's dimensions using these offset parameters. If the IL_SecondaryTEXLDMS_Mod token is not present, *xoffset* and *yoffset* default to 0.0

Operation:

src0 provides the texture coordinates for the texture sample. The first eight bits of the control field of the IL_OpCode token specify the texture stage from which to sample. Associated with the stage specified in the control field are 1) a texture image, 2) the texture's dimension, 3) and all format, filter, wrap, bias, and clamp settings specified in state and through the DCLPT instruction.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_TEXLDMS
	stage	23:16	Stage or unit number.
	reserved	29:24	Must be zero.
	sec_modifier_present	30	0: No secondary modifier token is present. 1: IL_SecondaryTEXLD_Mod token immediately follows the IL_OP_TEXLDMS token or an IL_PrimaryTEXLDMS_Mod token if bit 31 is set.
	pri_modifier_present	31	0: No primary modifier token is present. 1: IL_PrimaryTEXLD_Mod token immediately follows the IL_OP_TEXLDMS token.

Multisample Texture LOAD (Cont.)

2 Primary Multi-sample Texture Load Instruction Modifier token (is present only if the `pri_modifier_present` field is 1 in the previous IL_OpCode token).

`Mag`	2:0	Specifies how to filter texture values in the S and T directions when the pixel maps to an area less than one texel. Can be any value of the enumerated type ILTexFilterMode.
`Min`	5:3	Specifies how to filter texture values in the S and T directions when the pixel maps to an area greater than one texel. Can be any value of the enumerated type ILTexFilterMode.
reserved	14:6	Must be zero.
`Aniso`	17:15	When anisotropic filtering is enabled in the min or mag filter, specifies the maximum number of samples to use for anisotropic filtering. When anisotropic filtering is used for the min and mag filter, this value is the same. This can be any value of the enumerated type ILAnisoFilterMode.
reserved	31:18	Must be zero.

3 Secondary Multi-sample Texture Load Instruction Modifier token (is present only if the `sec_modifier_present` field is 1 in the previous IL_OpCode token).

`Xoffset`	7:0	Unnormalized (texel space) values added to the X texture coordinate before sampling. Signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5].
`Xoffset`	15:8	Unnormalized (texel space) values added to the Y texture coordinate before sampling. Signed fixed point with one bit of fraction (7.1) that allows ranges from [–64, 63.5].
reserved	2316	Must be zero.
`Sample`	28:24	This number is added to the third component of *src0* to select the single sample to load in the case that point multi-sample filtering is used. This field only applies if the state based point multi-sample filter is set.
reserved	31:29	Must be zero.

4 IL_Dst token (*dst*)

5 IL_Dst_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Dst token.

6 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.

7 IL_Src token (*src0*)

8 IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.

9 IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.

Related TEXLD, TEXLDB, TEXLDD.

Get Texel Weight

Instructions **TEXWEIGHT**

Syntax `texweight_stage(`*`n`*`) dst, src0`

Description Retrieves the weights used by a bilinear filtered fetch based upon the texture coordinate provided in *src0.*

The *dst.*x represents the horizontal LERP factor; the *dst.*y represents the vertical LERP factor.

The *dst*.z and *dst*.w component are not written to by this instruction; however, if the *component_z_b* or *component_w_a* field of the IL_Dst_Mod token is set to IL_MODCOMP_0 or IL_MODCOMP_1, the *dst*.z and *dst*.w are written (*dst*.z and *dst*.w can be set to 0.0 or 1.0 if IL_MODCOMP_0 or IL_MODCOMP_1 is used on the *component_z_b* or *component_w_a* field of the IL_Dst_Mod token).

The coordinates can be projected using the divComp source modifier on *src0.*

- If the divComp source modifier is used, and the component to divide by is negative, the result of this instruction is undefined. That is, if IL_DIVCOMP_Y is used, and the second component of *src0* is negative, the results of this texture load is undefined.
- divComp can be set to IL_DIVCOMP_UNKNOWN in this instruction; in this case, the component used to divide is specified in state.

The value of *stage* corresponds to the stage/unit.

This instruction can be used on only a *stage* that has been declared with a DCLPT instruction.

Operation:

src0 provides the texture coordinates for the texture weight. The first eight bits of the control field of the IL_OpCode token specify the texture stage from which to retrieve the weight. Associated with the stage specified in the control field are 1) a texture, 2) the texture's dimension, 3) and all format, filter, wrap, bias, and clamp states defined externally and through the DCLPT instruction.

ALU instructions perform arithmetic and relational operations. All take in at least one source operand and output to 1 destination operand.

Format

Token	Field Name	Bits	Description
1	code	15:0	IL_OP_TEXWEIGHT
	`stage`	23:16	Stage or unit number, 0 to 255.
	reserved	31:24	Must be zero.
2	IL_Dst token (*dst*)		
3	IL_Dst_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Dst token.		
4	IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.		
5	IL_Src token (*src0*)		
6	IL_Src_Mod token is present only if the `modifier_present` field is 1 in the previous IL_Src token.		
7	IL_Rel_Addr token is present only if the `relative_address` field is 1 in the preceding IL_Src or IL_Dst token.		

Related None.

Read Local Data Share (LDS) Memory into a Vector

Instructions **LDS_READ_VEC**

Syntax `lds_read_vec (_neighborExch)(_sharingMode) dst, src0.xy`

Description Reads the LDS memory into the *dst* register (can read a vector of up to four components). This uses the ownership accessing model. Each thread owns a part of the LDS memory. The LDS location is specified using the owner thread ID and offset, such as “Read data written/owned by thread Tid at Offset.”

Register *src0* specifies the location with *src0*.x = Tid, *src0*.y = offset.

Dst can be any writable register.

The sharing modes _sharingMode(rel) and _sharingMode(abs) are relative and absolute, respectively. The flag is optional. If it is not specified, the instruction takes the default sharing mode specified by dcl_lds_sharing_mode.

The flag neighborExch is optional. If it is specified, the output of lds memory is exchanged with its neighbor threads, so that the first thread receives all values from x-components, the second thread receives all values from y-components, etc. This flag is useful for applications like FFT matrix transpose.

It is used only in a compute shader.

Formats 1-input, 1-output, 0 additional token.

Opcode

Field Name	Bits	Description
code	15:0	Must be set to IL_OP_LDS_READ_VEC.
controls	17:16	_sharingMode: 0: IL_LDS_SHARING_MODE_RELATIVE 1: IL_LDS_SHARING_MODE_ABSOLUTE 2-3: reserved
	18	_neighborExch:0: off 1: on and enabled
	29:19	Not used. Must be 0
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Examples

```
lds_read_vec r1, r0.xy
lds_read_vec_neighborExch r1, r0.xy
lds_read_vec_sharingMode(rel) r1, r0.xy
lds_read_vec_neighborExch_sharingMode(abs) r1, r0.xy
```

Related LDS_WRITE_VEC

Write a Vector to Local Data Share (LDS) Memory

Instructions **LDS_WRITE_VEC**

Syntax `lds_write_vec _lOffset(N) (_sharingMode) dst, src0`

Description Writes data (a vector of up to four components) in *src0* into LDS memory. This uses the ownership accessing model: each thread owns a part of the LDS memory. Each thread can write only to the part it owns. The location is specified by offset.

src0 can be any register.

Dst must be of type `IL_REGTYPE_GENERIC_MEM`. This is used to provide the write mask.

`_lOffset()` is optional. If not specified, the offset is 0. If it is specified, such as `_lOffset(n)`, *n* must be a value of multiples of four in the range of [0,60], and must be smaller than the declared `lds_size_per_thread`.

`_sharingMode(rel)` or `_sharingMode(abs)` are for relative or absolute sharing mode, respectively. The flag is optional. If it is not specified, the instruction takes the default sharing mode specified by `dcl_lds_sharing_mode`.

This instruction is used only in a compute kernel.

Formats 1-input, 1-output, no additional token.

Opcode

Field Name	Bits	Description
code	15:0	Must be set to lL_OP_LDS_READ_VEC.
controls	17:16	_sharingMode: 0: IL_LDS_SHARING_MODE_RELATIVE 1: IL_LDS_SHARING_MODE_ABSOLUTE 2-3: reserved
	23:18	_ lOffset: offset within its reserved space, must be one of [0,4, 8, … 60]
	29:24	Not used. Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Examples

```
lds_write_vec mem._y__, r0.yyyy
lds_write_vec_lOffset(4) mem.xy__, r0.xyzw
lds_write_vec_sharingMode(rel) mem.__zw, r0.xyzw
lds_write_vec_lOffset(4)_sharingMode(abs) mem.x_z_w, r0.xyzw
```

Related LDS_READ_VEC.

Fence for Synchronization Threads, and/or LDS and/or Global Memory or UAV

Instructions **FENCE**

Syntax `fence[_threads][_lds][_memory][_sr]`

Description It must include at least one option. It is an error if none is specified. All selected options must complete before the thread continues.

`_threads` - synchronize threads in a group such that all threads must have reached this point before any thread can go further. The instruction cannot be used inside of any control flow. It can be used only in a compute shader (except for bit 18; see below).

`_lds` - shared memory fence. It ensures that:

- no LDS read/write instructions can be re-ordered or moved across this fence instruction.
- all LDS write instructions are complete (the data has been written to LDS memory, not in internal buffers) and visible to other threads.

`_memory` - global/scatter memory fence. It ensures that:

- no memory import/export instructions can be re-ordered or moved across this fence instruction.
- all memory export instructions are complete (the data has been written to physical memory, not in the cache) and is visible to other threads.

`_sr` - shared register write/read fence. No shared register writes/reads can be re-ordered or moved across this fence instruction.

All prior writes to shared registers done by this thread become visible to all other threads.

Bit 16 (0 = this is a thread barrier point; 1 = a barrier point is required)

Bit 17 (1 = all prior LDS writes must become visible to other threads in the current thread group)

Bit 18 (1 = all prior writes to global memory and UAV surfaces by this thread in program order become visible to all other threads) Note that both compute and pixel shaders can use this bit in the FENCE instruction.

Bit 19 (1 = all prior writes to shared registers done by this thread become visible to all other threads)

Pixel kernels can only use fence instructions for global memory. Compute kernels can use all of options.

In pixel kernels, use of discard instructions implies a fence_memory.

Use of discard with a fence_threads instruction is undefined in IL, although specific implementations can select an interpolation.

Formats 0-input, 0-output, no additional token.

Examples

```
lds_write_vec  mem._y__,  r0.yyyy
fence_threads_lds
lds_read_vec r1,  r0.xy
```

Opcode

Field Name	Bits	Description
code	15:0	Must be set to IL_OP_FENCE.
_threads	16	1 indicates that this is a fence for thread synchronization.
_lds	17	1 indicates that this is a fence for the LDS.
_memory	18	1 indicates that this is a fence for global memory or UAV.
_sr	19	1 indicates that this is a fence for *sr*.
Controls	29:20	Must be zero.
sec_modifier_present	30	Must be zero.

Fence for Synchronization Threads, and/or LDS and/or Global Memory or UAV (Cont.)

	pri_modifier_present	31	Must be zero.

Examples

```
lds_write_vec mem._y__, r0.yyyy
fence_threads_lds
lds_read_vec r1, r0.xy
```

Related None.

6.6 Integer Arithmetic Instructions

Integer AND

Instructions **IAND**

Syntax iand *dst*, *src0*, *src1*

Description Component-wise logical AND of each pair of 32-bit values from *src0* and *src1*. The 32-bit result is placed in *dst*.

Format 2-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_IAND
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Integer Negate

Instructions **INOT**

Syntax inot *dst*, *src0*

Description Performs a bit-wise one's complement on each component of *src0*. The 32-bit results are placed in the corresponding components of *dst*.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_NOT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Integer inclusive-OR, Integer Exclusive-OR

Instructions **IOR, IXOR**

Syntax

Function	Opcode	Syntax	Description
IOR	IL_OP_I_OR	ior *dst*, src0, *src1*	Integer inclusive-OR, bit-wise OR of each component in *src0* with the corresponding component in *src1*
IXOR	IL_OP_I_XOR	ixor *dst*, *src0*, *src1*	Integer exclusive-OR, bit-wise XOR of each component in *src0* with the corresponding component in *src1*

Description Performs a bit-wise logical operation of each component of *src0* with the corresponding component of *src1*. The 32-bit results are placed in the corresponding components of *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Bitwise Integer Addition

Instructions **I_ADD**

Syntax **iadd *dst, src0, src1***

Description Component-wise add of 32-bit operands *src0* and *src1*. The 32-bit result is placed in *dst*. No carry or borrow is done beyond the 32-bit values of each component; thus, this instruction is not sensitive to the sign of the operand.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_ADD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DADD.

Signed Integer Multiply and Add

Instructions **I_MAD**

Syntax `imad` *`dst, src0, src1, src2`*

Description Component-wise integer multiply of 32-bit signed operands *src0* and *src1*, keeping the low 32 bits (per component) of the result, followed by an integer add of *src2* to produce the low 32-bit (per component) result. and placing it in *dst*.

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_MAD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMAD.

Integer Maximum, Integer Minimum

Instructions **IMAX, IMIN**

Syntax

Function	Opcode	Syntax	Description
IMAX	IL_OP_I_MAX	imax *dst, src0, src1*	*dst* = *src0* > *src1* ? *src0*:*src1*
IMIN	IL_OP_I_MIN	imin *dst, src0, src1*	*dst* = *src0* < *src1* ? *src0*:*src1*

Description Component-wise integer maximum (I_MAX) and minimum (I_MIN).

Compares each component of *src0* with the corresponding component of *src1*. If the comparison evaluates true, *src0* is returned in the corresponding component of *dst*; otherwise, *src1* is returned.

For the MAX and MIN instructions, if *src0*.{x|y|z|w} or *src1*.{x|y|z|w} is NaN, then the other component, *src1*.{x|y|z|w} or *src0*.{x|y|z|w}, respectively, is returned. Denorms are flushed before the comparison is performed. If a flushed denorm is the maximum value (for the MAX instruction) or minimum value (for the MIN instruction), then the flushed denorm is returned. The MAX instruction uses a greater-than-or-equal-to comparison. Thus, if min(*src0*, *src1*) = *src0*, then max(*src0*, *src1*) = *src1*, including cases with +0 and -0, such as when denorms are flushed to sign preserve zero.

If the _ieee flag is included, then MIN and MAX follow IEEE-754r rules for minnum and maxnum (except denorms are flushed before the comparison is made). Also, MIN returns -0, and MAX returns +0, for comparisons between -0 and +0.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_MAX, IL_OP_I_MIN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Signed Integer Multiplication (Low 32 Bits)

Instructions **IMUL**

Syntax imul *dst, src0, src1*

Description Component-wise multiply of 32-bit operands *src0* and *src1*. The lower 32 bits of the 64-bit result, which is placed in *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_MUL
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related IMUL_HIGH.

Signed Integer Multiplication (High 32 Bits)

Instructions **IMUL_HIGH**

Syntax `imul_high dst, src0, src1`

Description Component-wise multiply of 32-bit signed operands *src0* and *src1*. The upper 32 bits of the 64-bit result is placed in the corresponding component of *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_MUL_HIGH
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMUL, IMUL, MUL, UMUL.

Integer Compare

Instructions **IEQ, IGE, ILT, INE**

Syntax

Function	Opcode	Syntax	Description
==	IL_OP_I_EQ	ieq *dst*, *src0*, *src1*	Compares if integer vector ins scr0 is equal to the one in *src1*. If TRUE, 0xFFFFFFFF is returned for that component; otherwise, 0x00000000 is returned.
≥	IL_OP_I_GE	ige *dst*, *src0*, *src1*	Compares if integer vector in *src0* is greater or equal to the one in *src1*. If TRUE, 0xFFFFFFFF is returned for that component; otherwise, 0x00000000 is returned.
<	IL_OP_I_LT	ilt *dst*, *src0*, *src1*	Compares if integer vector in src0 is less than the one in *src1*. If TRUE, 0xFFFFFFFF is returned for that component; otherwise, 0x00000000 is returned.
≠	IL_OP_I_NE	ine *dst*, *src0*, *src1*	Compares two integer vectors, one in *src0*, the other in *src1*, to check if they are not equal. If TRUE, 0xFFFFFFFF is returned for that component; otherwise, 0x00000000 is returned.

Description Component-wise compares two vectors using an integer comparison. If *src0*.{x|y|z|w} and the corresponding component in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE; otherwise, it is set to FALSE. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the control field, pri_modifier_present, and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Float Compare, Unsigned Integer Compare.

Two's Complement Negate

Instructions **INEGATE**

Syntax inegate *dst*, *src0*

Description Computes the two's complement negation of each component in *src0.* The 32-bit results are placed in the corresponding components of *dst*.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_I_NEGATE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Integer Shift Left, Integer Shift Right

Instructions **ISHL, ISHR**

Syntax

Function	Opcode	Syntax	Description
ISHL	IL_OP_I_SHL	ishl *dst*, *src0*, *src1*	*src1* must be a scalar value; thus, all four swizzles on *src1* must be the same. Component-wise shift of each 32-bit value in *src0* left by an unsigned integer bit count provided by the LSB five bits (0-31 range) in *src1*.selected_component, inserting 0. The 32-bit per component results are placed in *dst*. The count is a scalar value applied to all components.
ISHR	IL_OP_I_SHR	ishr *dst*, *src0*, *src1*	Component-wise arithmetic shift of each 32-bit value in *src0* right by an unsigned integer bit count provided by the LSB five bits (0-31 range) in *src1*.x, replicating the value of bit 31. The 32-bit per component result is placed in *dst*. The count is a scalar value applied to all components.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

6.7 Unsigned Integer Operations

Integer Shift Left, Integer Shift Right

Instructions **USHR**

Syntax ushr *dst, src0, src1*

Description Component-wise shift of each 32-bit value in *src0* right by an unsigned integer bit count provided by the LSB five bits (0-31 range) in *src1*.selected_component, inserting 0. The 32-bit per component result is placed in *dst*. The count is a scalar value applied to all components.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_SHR
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floating Point Division, Unsigned Integer Division

Instructions **UDIV**

Syntax udiv *dst, src0, src1*

Description Component-wise unsigned division of the 32-bit operand in *src0* by the corresponding component in *src1*. The 32-bit quotient is placed in the corresponding component of *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_DIV
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Unsigned Integer Modulo

Instructions **UMOD**

Syntax umod *dst, src0, src1*

Description Component-wise unsigned division of the 32-bit operand *src0* by the 32-bit operand *src1*. The 32-bit remainder is placed in the corresponding component of *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_MOD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Unsigned Integer Multiply and Add

Instructions **UMAD**

Syntax umad *dst, src0, src1, src2*

Description Component-wise unsigned multiply of the 32-bit operand *src0* by the 32-bit operand *src1* is added to the corresponding component in *src2*. The result of the multiply-add operation is placed in the corresponding component of *dst*.

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_MAD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMAD.

Unsigned Integer Maximum, Unsigned Integer Minimum

Instructions **UMAX, UMIN**

Syntax

Function	Opcode	Syntax	Description
UMAX	IL_OP_U_MAX	umax *dst, src0, src1*	unsigned integer maximum, *dst = src0 > src1 ? src0:src1*
UMIN	IL_OP_U_MIN	umin *dst, src0, src1*	unsigned integer minimum, *dst = src0 < src1 ? src0:src1*

Description Compares each component of *src0* with the corresponding component of *src1*. If the comparison is TRUE, *src0* is returned in the corresponding component of *dst*; otherwise, *src1* is returned.

For the MAX and MIN instructions, if *src0*.{x|y|z|w} or *src1*.{x|y|z|w} is NaN, then the other component, *src1*.{x|y|z|w} or *src0*.{x|y|z|w}, respectively, is returned. Denorms are flushed before the comparison is performed. If a flushed denorm is the maximum value (for the MAX instruction) or minimum value (for the MIN instruction), then the flushed denorm is returned. The MAX instruction uses a greater-than-or-equal-to comparison. Thus, if min(*src0*, *src1*) = *src0*, then max(*src0*, *src1*) = *src1*, including cases with +0 and -0 such as when denorms are flushed to sign preserve zero.

If the _ieee flag is included, MIN and MAX follow IEEE-754r rules for minnum and maxnum (except denorms are flushed before the comparison is made). In addition, MIN returns -0 and MAX returns +0 for comparisons between -0 and +0.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	See *Syntax*, above.
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Unsigned Integer Compare

Instructions **UGE, ULT**

Syntax

Function	Opcode	Syntax
≥	IL_OP_U_GE	ige *dst, src0, src1*
<	IL_OP_U_LT	ilt *dst, src0, src1*

Description Component-wise compares two vectors using an unsigned integer comparison. For UGE: (*src0* ≥ *src1*); for ULT: (*src0* < *src1*). If *src0*.{x|y|z|w} and the corresponding component in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE and returns 0xFFFFFFFF; otherwise, it is set to FALSE and returns 0x00000000. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the control field, pri_modifier_present, and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_GE, IL_OP_U_LT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Float Compare, Integer Compare.

Unsigned Integer Multiplication

Instructions **UMUL**

Syntax umul *dst, src0, src1*

Description Component-wise multiply of 32-bit unsigned operands *src0* and *src1*. The lower 32 bits of the 64-bit result (per component) is placed in the corresponding component of *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_U_MUL
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMUL, IMUL_HIGH, UMUL_HIGH.

Unsigned Integer Multiplication

Instructions **UMUL_HIGH**

Syntax `umul_high` *`dst, src0, src1`*

Description Component-wise multiply of 32-bit unsigned operands *src0* and *src1*. The upper 32 bits of the 64-bit result (per component) is placed in the corresponding component of *dst*.

Format 2-input, 1-output.

Opcode	Field Name	Bits	Description
	code	15:0	IL_OP_U_MUL_HIGH
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.

Related DMUL, IMUL, MUL, UMUL.

6.8 Conversion Instructions

Float to Signed Integer Conversion

Instructions **FTOI**

Syntax `ftoi` *`dst`*, *`src0`*

Description Converts each component of *src0* in the range [-2147483648.999f, 2147483647.999f] to a signed integer, and puts the result in the corresponding component of *dst*. Using values outside the specified range produces undefined results.

After the instruction executes, *dst* contains a 32-bit signed integer 4-tuple. The conversion is performed per component.

Rounding is performed towards zero on each component of *src0*, following the C convention for casts from float to int. Applications that require different rounding semantics can invoke the ROUND_* instructions before using FTOI.

Format 1-input, 1-output.

Opcode	Field Name	Bits	Description
	code	15:0	IL_OP_FTOI
	control	29:16	Must be zero.
	sec_modifier_present	30	Must be zero.
	pri_modifier_present	31	Must be zero.

Related F2D, FTOU.

Float to Unsigned Integer Conversion

Instructions **FTOU**

Syntax `ftou dst, src0`

Description Converts each component of *src0* in the range [0.0f, 4294967296.999f] to an unsigned integer and puts the result in the corresponding component of *dst*. Using values outside the specified range produces undefined results.

After the instruction executes, *dst* contains a 32-bit unsigned integer 4-tuple. The conversion is performed per component.

Rounding is performed towards zero on each component of *src0*, following the C convention for casts from float to unsigned int. Applications that require different rounding semantics can invoke the ROUND_* instructions before using FTOU.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_FTOU
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related F2D, FTOI.

Signed Integer to Float Conversion

Instructions **ITOF**

Syntax `itof dst, src0`

Description Converts each component of *src0* to a float, and puts the result in the corresponding component of *dst*. Each component of *src0* is assumed to contain a signed 32-bit integer 4-tuple. After conversion, *dst* contains a floating-point 4-tuple.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ITOF
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related D2F, UTOF.

Unsigned Integer to Float Conversion

Instructions **UTOF**

Syntax `utof dst, src0`

Description Converts each component of *src0* to a float and puts the result in the corresponding component of *dst*. Each component of *src0* is assumed to contain an unsigned 32-bit integer 4-tuple.

After the instruction executes, *dst* contains a floating-point 4-tuple.

If an integer is not represented exactly, the nearest representable value is used. Rounding is performed towards zero on each component of *src0*, following the C convention for casts from unsigned int to float. Applications that require different rounding semantics can invoke the ROUND_* instructions before using UTOF.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_UTOF
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related D2F, ITOF.

Double to Float Conversion

Instructions **D2F**

Syntax `d2f dst, src0`

Description Abs and negate modifiers can be used on the source. All four output components are set to the float converted value. Nan/inf convert to nan/inf; signs are preserved. If the result does not fit as a float, inf is returned. Uses round to nearest even.

d	-NaN	-inf	-F	-1.0	-denom	-0	+0	+denom	+1.0	+F	+inf	+NaN
f	0xffc00000	-inf	-F	-1.0	-0.0	-0.0	++0.0	+0.0	+1.0	+F	+inf	0x7fc00000

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_2_F
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related ITOF, UTOF.

Float to Double Conversion

Instructions **F2D**

Syntax f2d *dst*, *src0*

Description The source is interpreted as a float in component x (after swizzles). Abs and negate modifiers can be used on the source. Output yx or output wz is set to the double converted value. Nan/inf convert to nan/inf; signs are preserved.

Rounding is performed towards zero on each component of *src0*, following the ANSI C convention for casts from float to double. Applications that require different rounding semantics can invoke the ROUND_* instructions before using F2D.

f	-inf	-F	-1.0	-denom	-0	+0	+denom	+1.0	+F	+inf	NaN
d	-inf	-F	-1.0	-0.0	-0.0	+0.0	+0.0	+1.0	+F	+inf	Nan*

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_F_2_D
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related FTOI, FTOU.

6.9 Float Instructions

Absolute Value

Instructions **ABS**

Syntax abs *dst, src0*

Description Computes the absolute value of each component in a vector (*src0*).

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (=0; i < 4; i++)
     v[i] = abs(v1[i]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ABS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floating Point Addition

Instructions **ADD**

Syntax add *dst, src0, src1*

Description Adds two float vectors: *src0*.{x|y|z|w} + *src1*.{x|y|z|w}.

Operation:

```
VECTOR v1 = EvalSource(src1);
VECTOR v2 = EvalSource(src2);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v1[i] + v2[i];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ADD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DADD.

Inverse Cosine (arccos)

Instructions **ACOS**

Syntax acos *dst, src0*

Description Computes the inverse cosine in radians of *src0*.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the w component. *src0*.w must be within the range [-1.0, 1.0]; otherwise, the result is undefined. The maximum absolute error is 0.002.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v
v[0] = v[1] = v[2] = v[3] = acos(v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ACOS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Logical-AND

Instructions **AND**

Syntax and *dst, src0, src1*

Description Performs a component-wise logical AND of each pair of 32-bit values from *src1* and *src2*. The 32-bit result is placed in *dst*.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_AND
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Inverse Sine (arcsin)

Instructions **ASIN**

Syntax asin *dst, src0*

Description Computes the inverse sine in radians of the w component of *src0*. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the w component. *src0*.w must be within the range [-1.0, 1.0]; otherwise, the result is undefined. The maximum absolute error is 0.002.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v
v[0] = v[1] = v[2] = v[3] = asin(v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ASIN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Inverse Tangent (arctan)

Instructions **ATAN**

Syntax `atan dst, src0`

Description Computes the inverse tangent in radians of *src0*.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the w component. *src0*.w must be within

the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$; otherwise, the result of this instruction is undefined.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v
v[0] = v[1] = v[2] = v[3] = atan(v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ATAN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Clamp to Given Range

Instructions **CLAMP**

Syntax `clamp dst, src0, src1, src2`

Description Clamps *src0* to *src1* and *src2*.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v3 = EvalSource(src2);
VECTOR v;
for (i=0; i < 4; i++)
{
    v[i] = v1[i];
    if(v[i] < v2[i])
    {
        v[i] = v2[i];
    }
    if(v[i] > v3[i])
    {
        v[i] = v3[i];
    }
}
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CLAMP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Ceiling

Instructions **CLG**

Syntax `clg dst, src0`

Description Calculates the ceiling of each component of *src0*, and places the 32-bit result in the corresponding component of *dst*. The ceiling of a component is defined as the smallest integer greater than or equal to the value in the component. For example, the ceiling of 2.3 is 3.0, and the ceiling of -3.6 is -3.0. The operation is identical to ROUND_PLUS_INF.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = ceiling(v1[i]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CLG
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related FLR.

Component-Wise Conditional Move

Instructions **CMOV**

Syntax cmov *dst*, *src0*, *src1*

Description Conditionally moves a value from *src1* to *dst*. If *src0*.{x|y|z|w} is not 0.0f, the value of the corresponding component in *dst* becomes the value of the corresponding component of *src1*; otherwise, the corresponding component in *dst* remains unchanged.

IL_DSTMOD_1 and IL_DSTMOD_0 in the IL_Dst_Mod token are treated as IL_DSTMOD_NOWRITE for components where *src1* is 0.0f.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
{
    v[i] = v2[i];
    if(v1[i] == 0.0)
        v[i].MASK = NOWRITE;
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CMOV
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CMOV_LOGICAL.

Component-wise Conditional Logical Move

Instructions **CMOV_LOGICAL**

Syntax cmov_logical *dst*, *src0*, *src1*, *src2*

Description For each component in *src0* (post-swizzle): if the component has any bit set, the component in *src1* (post-swizzle) is copied to the corresponding component in *dst*; otherwise, the corresponding component in *src2* is copied to the corresponding component in *dst*.

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CMOV_LOGICAL
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CMOV.

Compare with Value

Instructions **CMP**

Syntax `cmp_relop(op)_cmpval(cmpval) dst, src0, src1, src2`

Description For every component of *src0* that passes (is TRUE for) its relational comparison with *cmpval*, that component is set to the corresponding component of *src1*; otherwise, it is set to the corresponding component of *src2*.

Operation:

```
VECTOR v1 = EvalSource(src1);
VECTOR v2 = EvalSource(src2);
VECTOR v3 = EvalSource(src3);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = (v1[i] relop cmpval) ? v2[i] : v3[i];
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_CMP		
control	29:16	**Field Name**	**Bits**	**Description**
		relop	18:16	Any value of the enumerated type ILRelOp(*relop*). See Table 5.14 on page 5-8.
		reserved	20:19	Must be zero.
		cmpval	23:21	Any value of the enumerated type ILCmpValue(*cmpval*). See Table 5.17 on page 5-9.
		reserved	29:24	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related None.

Clamp Color Output Values

Instructions **COLORCLAMP**

Syntax colorclamp *dst*, *src0*

Description Clamps (or does not clamp) an output color to values specified by the frame buffer color clamp state.

Operation:

```
VECTOR v1 = EvalSource(src0);
int buffer = RegisterNum(src0);
VECTOR v;
for (i=0; i < 4; i++)
{
    if(AS_CB_CLAMP_MODE_N(buffer) == 0_1)
{
    v[i] = (v1[i] <= 0.0) ? 0.0 : v1[i];
    v[i] = (v[i] > 1.0) ? 1.0 : v[i];
}
else if (AS_CB_CLAMP_MODE_N(buffer) == NEG_1_1)
{
    v[i] = (v1[i] < -1.0) ? -1.0 : v1[i];
    v[i] = (v[i] > 1.0) ? 1.0 : v[i];
}
else if (AS_CB_CLAMP_MODE_N(buffer) == NONE)
{
    v[i]=v1[i];
}
}
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_COLORCLAMP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Cosine (cos)

Instructions **COS**

Syntax cos *dst, src0*

Description Computes the cosine of src0.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component.

The fourth component of *src0* must be in the range [-π, π]; otherwise, the result is of this instruction is undefined.

The max absolute error is 0.002.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_COS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Cross Product

Instructions **CRS**

Syntax crs *dst, src0, src1*

Description Computes a cross product. This instruction does not write to the *dst*.w component; however, the if the component_w_a field of the IL_Dst_Mod token is set to IL_MODCOMP_0 or IL_MODCOMP_1, *dst*.w is written. That is, *dst*.w can be set to 0.0 or 1.0 if IL_MODCOMP_0 or IL_MODCOMP_1 is used on the component_w_a field of the IL_Dst_Mod token.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
v[0] = v1[1] * v2[2] - v1[2] * v2[1];
v[1] = v1[2] * v2[0] - v1[0] * v2[2];
v[2] = v1[0] * v2[1] - v1[1] * v2[0];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_CRS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Determinant of a 4x4 Matrix -- WARNING: This instruction has been deleted.

Instructions **DET**

Syntax det *dst, src0*

Description Calculates the determinant of a 4x4 matrix. Relative addressing and source modifiers of *src0* are applied to each row of the matrix beginning with *src0*. The 32-bit scalar result is placed in all four components of *dst.*

Operation:

```
VECTOR v0 = EvalSource(src0);
VECTOR v1 = EvalSource(src0+1); # Register number of src0 + 1
VECTOR v2 = EvalSource(src0+2); # Register number of src0 + 2
VECTOR v3 = EvalSource(src0+3); # Register number of src0 + 3
VECTOR v;
float f;

f = v0[0]*(v1[1]*(v2[2]*v3[3]-v2[3]*v3[2])-v1[2]*(v2[1]*v3[3]-
v3[1]*v2[3])+v1[3]*(v2[1]*v3[2]-v3[1]*v2[2]))

- v0[1]*(v1[0]*(v2[2]*v3[3]-v2[3]*v3[2])-v1[2]*(v2[0]*v3[3]-
v3[0]*v2[3])+v1[3]*(v2[0]*v3[2]-v3[0]*v2[2]))

+ v0[2]*(v1[0]*(v2[1]*v3[3]-v2[3]*v3[1])-v1[1]*(v2[0]*v3[3]-
v3[0]*v2[3])+v1[3]*(v2[0]*v3[1]-v3[0]*v2[1]))

- v0[3]*(v1[0]*(v2[1]*v3[2]-v2[2]*v3[1])-v1[1]*(v2[0]*v3[2]-
v3[0]*v2[2])+v1[2]*(v2[0]*v3[1]-v3[0]*v2[1]));

v[0] = v[1] = v[2] = v[3] = f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DET
control	29:16	Must be the enumerated type ILMatrix(*IL_MATRIX_4X4*). See Table 5.6 on page 5-3.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Vector Distance

Instructions **DIST**

Syntax `dist dst, src0, src1`

Description Computes the vector distance from *src0*.[xyz] to *src1*.[xyz]. The 32-bit scalar result is placed in all four components of *dst*.

Operation:

```
VECTOR v1 = EvalSource(src0;
VECTOR v2 = EvalSource(src1;
VECTOR v;
V[0] = v[1] = v[2] = v[3] =
    sqrt((v1[0]-v2[0])*(v1[0]-v2[0]) +
         (v1[1]-v2[1])*(v1[1]-v2[1]) +
         (v1[2]-v2[2])*[v1[2]-v2[2]) +
         (v1[3]-v2[3])*(v1[3]-v2[3]));
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DIST
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floating Point Division

Instructions **DIV**

Syntax `div_zeroop(op) dst, src0, src1`

Description Floating point division *src0*.{x|y|z|w} / *src1*.{x|y|z|w}. The 32-bit quotient is placed in the corresponding component of *dst*. DX10 requires the DIV instruction to use zeroop = IL_ZEROOP_INFINITY.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
float f;
for (i=0; i < 4; i++)
{
    if(v2[i] == 0.0)
    {
        if(zeroop == IL_ZEROOP_0)
            f = 0.0;
        else if(zeroop == IL_ZEROOs_FLT_MAX)
            f = FLT_MAX;
    else if(zeroop == IL_ZEROOP_INFINITY)
            f = INFINITY;
        else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
            f = INFINITY or FLT_MAX;  # Depends on IL Implementation

        if(v1[i] < 0.0)
            f = -f;

        v[i] = f;
   }
    else
        v[i] = v1[i] / v2[i];
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DIV
control	29:16	DIV: Any value of the enumerated type ILZeroOp(*zeroop*). See Table 5.16 on page 5-9.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Two-Component Dot Product and Scalar Add

Instructions **DP2ADD**

Syntax dp2add *dst*, *src0*, *src1*, *src2*

Description Computes the two-component dot product of *src0*.[xy] and *src1*.[xy] and adds scalar *src2*.z to the result. The 32-bit scalar result is placed in all four components of *dst*. This instruction corresponds to the DX10 dp2 operation. If the control field is 0, then 0*n equals zero, even if n = NaN.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v3 = EvalSource(src2);
VECTOR v;
v[0] = v[1] = v[2] = v[3] =
    v1[0]*v2[0] + v1[1]*v2[1] + v3[2];
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DP2ADD
control	29:16	0 DX9 DP2-style multiple. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DP2, DP3, DP4.

Two-Component Dot Product

Instructions **DP2**

Syntax dp2[_ieee] *dst, src0, src1*

Description Two-component dot product of vector operands *src0*.[xy] and *src1*.[xy]. The 32-bit scalar result is placed in all four components of *dst*. If the control field is 0, then 0*n = 0, even if n = NaN.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
v[0] = v[1] = v[2] = v[3] = v1[0]*v2[0] + v1[1]*v2[1];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DP2
control	29:16	0 DX9 DP2-style multiple. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related NoneDP2ADD.

Three-Component Dot Product

Instructions **DP3**

Syntax dp3[_ieee] *dst, src0, src1*

Description Three-component dot product vector operands *src0*.[xyz] and *src1*.[xyz]. The 32-bit scalar result is placed in all four components of *dst*. If the control field is 0, then 0*n = 0, even if n = NaN.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
v[0] = v[1] = v[2] = v[3] =
    v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DP3
control	29:16	0 DX9 DP2-style multiple. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related NoneDP2ADD.

Four-Component Dot Product

Instructions **DP4**

Syntax dp4[_ieee] *dst, src0, src1*

Description Four-component dot product vector operands *src0*.[xyzw] and *src1*.[xyzw]. The 32-bit scalar result is placed in all four components of *dst*. If the control field is 0, then 0*n = 0, even if n = NaN.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
v[0] = v[1] = v[2] = v[3] =
    v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2] + v1[3]*v2[3];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DP4
control	29:16	0 DX9 DP2-style multiple. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related NoneDP2ADD.

Vector Distance

Instructions **DST**

Syntax dst *dst, src0, src1*

Description Computes a distance-related value from operands *src0* and *src1*. The result vector is placed in *dst*. If the control field is 0, then 0*n = 0, even if n = NaN.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
V[0] = 1.0;
V[1] = v1[1] * v2[1];
V[2] = v1[2];
V[3] = v2[3];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DST
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Instantaneous Derivative in X

Instructions **DSX**

Syntax dsx[_centroid] *dst, src0*

Description Instantaneous derivative in the x-direction. Computes the rate of change of each float32 component of *src0* (post-swizzle) in the RenderTarget x direction. The results are undefined if used in a vertex or geometry shader.

The data in the current pixel shader invocation might participate in the calculation of the requested derivative, since the derivative is calculated only once per 2x2 quad.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DSX
control	29:16	See below.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Field Name	Bits	Description
reserved	22:16	Must be zero.
gradient	23	0: Gradients can be computed once per quad. 1: Gradients are computed for each pixel.
centroid	24	0: Pixel center is used to determine the derivative value. 1: Pixel centroid is used to determine the derivative value.
reserved	29:25	Must be zero.

Related None.

Instantaneous Derivative in Y

Instructions **DSY**

Syntax `dsy[_centroid] dst, src0`

Description Instantaneous derivative in the y-direction. Computes the rate of change of each float32 component of *src0* (post-swizzle) in the RenderTarget y direction. The results are undefined if used in a vertex or geometry shader.

The data in the current pixel shader invocation might participate in the calculation of the requested derivative, since the derivative is calculated only once per 2x2 quad.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DSY
control	29:16	See below.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Field Name	Bits	Description
reserved	22:16	Must be zero.
gradient	23	0: Gradients can be computed once per quad. 1: Gradients are computed for each pixel.
centroid	24	0: Pixel center is used to determine the derivative value. 1: Pixel centroid is used to determine the derivative value.
reserved	29:25	Must be zero.

Related None.

Compute Sine and Cosine

Instructions **DXSINCOS**

Syntax dxsincos *dst, src0, src1, src2*

Description Computes sine and cosine values of *src0.w* for the legacy DX9 sincos instruction. The results are returned in *dst*.x and *dst*.y.

This instruction supports the DX legacy instruction *sincos dst*, *src0*, *src1*, *src2*. *src1* and *src2* must be constant float registers and are used in a Taylor series expansion. See the *DX SDK* to understand how the two constants are used in a Taylor series expansion. Devices with native sincos support ignore *src1* and *src2*.

By default this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. The fourth component of *src* must be within the range [-π, π]; otherwise, the result of this instruction is undefined.

This instruction does not write to the *dst*.z and *dst*.w components; however, if the component_z_b or the component_w_a field of the IL_Dst_Mod token is set to IL_MODCOMP_0 or IL_MODCOMP_1, the *dst*.z and *dst*.w are written. That is, *dst*.z and *dst*.w can be set to 0.0 or 1.0 if IL_MODCOMP_0 or IL_MODCOMP_1 is used on the component_z_b or component_w_a field of the IL_Dst_Mod token.

The maximum absolute error is 0.002.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
v[0] = cos(v1[3]);  // approximated by using Taylor series
v[1] = sin(v1[3]);  // approximated by using Taylor series
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_DXSINCOS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Equality Compare Floats

Instructions **EQ**

Syntax eq *dst*, *src0*, *src1*

Description Performs the float comparison *src0* == *src1* for each component, and writes the result to *dst*. If the comparison is true, 0xFFFFFFFF is returned for that component; otherwise, 0x0000000 is returned. This instruction follows DX10 Floating Point Rules. Denorms are flushed before comparison (original source registers untouched). +0 equals -0. Comparison with NaN returns false.

Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the pri_modifier_present and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EQ
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Integer Compare, Unsigned Integer Compare.

Full-Precision e to the Power of x

Instructions **EXN**

Syntax exn *dst, src0*

Description Full-precision base e power *src0*: e^{src}.

By default, this instruction operates on *src0.w*, but can operate on any component by swizzling it into the fourth component.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
v[0] = v[1] = v[2] = v[3] = exp2(v1[3] * log2(e));
WriteResult(v, dst);
```

Format 1-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EXN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

2 Raised to a Power

Instructions **EXP**

Syntax exp *dst, src0*

Description Full precision base 2 power *src0*: 2^{src}.

Computes two to the power of *src0.w*. The floating point result is placed in all four components of *dst*. By default this instruction operates on *src0.w*, but can operate on any component by swizzling it into the fourth component.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EXP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related EXP_VEC, POW.

Component-Wise Full Precision 2 to the Power of X

Instructions **EXP_VEC**

Syntax exp_vec *dst, src0*

Description EXP computes a scalar version, EXP_VEC computes the same result per component.

Computes 2 raised to the power of each component of *src0*. The result of the operation is accurate to at least 21 bits. The 32-bit floating point results are placed in the corresponding components of *dst*.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EXP_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related EXN, EXP, EXPP, POW.

Component-Wise Partial Precision 2 to the Power of X

Instructions **EXPP**

Syntax expp *dst, src0*

Description Computes partial precision of 2 raised to the power of each component of *src0*. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. The result of the operation is accurate to at least 10 bits. The 32-bit floating point results are placed in the corresponding components of *dst*.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f = floor(v1[3]);
v[0] = exp2(f);
v[1] = v1[3] – f;
v[2] = exp2(v1[3]);
v[3] = 1.0;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EXPP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related EXN, EXP, EXPP, POW.

Face Forward

Instructions **FACEFORWARD**

Syntax faceforward *dst, src0, src1, src2*

Description FACEFORWARD performs the following calculation:

d(dst = *src2**sign(dot(*src0*, *src1*))

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_FACEFORWARD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floor

Instructions **FLR**

Syntax `flr dst, src0`

Description Performs a component-wise floor operation on the operand to generate a result vector. Calculates the floor of each component of *src0*, and places the 32-bit result in the corresponding component of *dst*.The floor of a component is defined as the largest integer less-than-or -equal to the value in the component. For example, the floor of 2.3 is 2.0 and the floor of -3.6 is -4.0. The operation is identical to ROUND_NEG_INF.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = floor(v1[i]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_FLR
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CLG.

Fraction

Instructions **FRC**

Syntax `frc dst, src0`

Description Extracts the fractional portion of each component of *src0*. The results are returned in the corresponding component of *dst*. The fractional portion of a component is defined as the result after subtracting the floor of the component from the component (see FLR). The result is always in the range [0.0, 1.0).

For negative values, the fractional portion is not the number to the right of the decimal point. For example, the fractional portion of -1.7 is 0.3, not 0.7. In this case, it is produced by subtracting the floor of -1.7 - (-2.0) from 1.7.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v1[i] - (float)floor(v1[i]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_FRC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DFRAC, TRC.

Filter Width

Instructions **FWIDTH**

Syntax `fwidth dst, src0`

Description Computes the sum of the absolute derivative in x and y using local differencing for each component of *src0*. The result is returned in the corresponding component of *dst*. If used in a vertex shader, the results are undefined.

Operation:

```
VECTOR v1 = EvalSource(src0);
for (i=0; i < 4; i++)
    v[i] = abs(dPdx(v1[i])) + abs(dPdy(v1[i]));
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_FWIDTH
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Greater or Equal Floats

Instructions **GE**

Syntax ge *dst*, *src0*, *src1*

Description Compares two float vectors *src0* and *src1* for each component, and writes the result to *dst*. If the comparison is true, 0xFFFFFFFF is returned for that component; otherwise, 0x0000000 is returned. This instruction follows DX10 Floating Point Rules. Denorms are flushed before comparison (original source registers untouched). +0 equals -0. Comparison with NaN returns false.

Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the pri_modifier_present and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EQ
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Integer Compare, Unsigned Integer Compare.

Vector Length

Instructions **LEN**

Syntax len *dst, src0*

Description Computes the length of a vector. Computes the vector length of the three component vector in *src0*.[xyz]. The scalar 32-bit floating point result is placed in all four components of *dst*.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
V[0] = v[1] = v[2] = v[3] =
    sqrt(v1[0]*v1[0]+ v1[1]*v1[1] + v1[2]*v1[2] + v1[3]*v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LEN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Lighting Coefficient

Instructions **LIT**

Syntax lit *dst*, *src0*

Description Calculates lighting coefficients for ambient, diffuse, and specular light contributions.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float epsilon = 1.192092896e-07F;
float g = v1[3];
if(g < -(128.0-epsilon))
    g = -(128.0-epsilon);
else if (g > (128.0-epsilon))
    g = 128.0-epsilon;
v[0] = 1.0;
v[1] = (v1[0] > 0.0) ? v1[0] : 0.0;
v[2] = ((v1[0] > 0.0) && (V1[1] > 0)) ? EXP2(g * LOG2(v1[1])) : 0.0;
v[3] = 1.0;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LIT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Natural Logarithm

Instructions **LN**

Syntax `ln_zeroop(op) dst, src0`

Description Computes the natural logarithm of *src0*.w. The result of the operation is accurate to at least bits. The 32-bit floating point result is placed in all four components of *dst*. By default this instruction operates on *src0.w*, but can operate on any component by swizzling it into the fourth component. zeroop can be any value of the enumerated type ILZeroOp(zeroop) except IL_ZEROOP_0.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f;
if (v1[3] == 0.0)
{
    if(zeroop == IL_ZEROOP_FLT_MAX)
        f = -FLT_MAX;
   else if(zeroop == IL_ZEROOP_INFINITY)
        f = -INFINITY;
    else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
        f = -INFINITY or -FLT_MAX;  # Depends on IL Implementation
}
else if (v1[3] < 0.0)
{
    f = undefined;
}
else
{
    f = (float)(log10(v1[3])/log10(e));
}
v[0] = v[1] = v[2] = v[3] = f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LN
control	29:16	Any value of the enumerated type ILZeroOp(*zeroop*). See Table 5.16 on page 5-9.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LOG_VEC, LOGP.

Base-2 Logarithm

Instructions **LOG**

Syntax log_zeroop(*op*) *dst*, *src0*

Description Computes the base-2 logarithm of *src0*.w. The result of the operation is accurate to at least 21 bits. The 32-bit floating point result is placed in all four components of *dst*. By default this instruction operates on *src0.w*, but can operate on any component by swizzling it into the fourth component. zeroop can be any value of the enumerated type ILZeroOp(zeroop) except IL_ZEROOP_0.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f;
if (v1[3] == 0)
{
    if(zeroop == IL_ZEROOP_FLT_MAX)
        f = -FLT_MAX;
   else if(zeroop == IL_ZEROOP_INFINITY)
        f = -INFINITY;
    else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
        f = -INFINITY or -FLT_MAX;  # Depends on IL Implementation
}
else if (v1[3] < 0.0)
{
    f = undefined;
}
else
{
    f = (float)(log10(v1[3])/log10(2));
}
v[0] = v[1] = v[2] = v[3] = f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LOG
control	29:16	Any value of the enumerated type ILZeroOp(*zeroop*). See Table 5.16 on page 5-9.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LOG_VEC, LOGP.

Component-wise Base-2 Logarithm

Instructions **LOG_VEC**

Syntax log_vec *dst, src0*

Description Computes the base-2 logarithm of each component of *src0*. The result of the operation is accurate to at least 21 bits. This is the vector version of LOG with zeroop set to il_zeroop_infinity.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LOG_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LN, LOG, LOGP.

Base-2 Logarithm (partial precision)

Instructions **LOGP**

Syntax `logp_zeroop(op) dst, src0`

Description Computes the base-2 logarithm of *src0*.w using partial precision. The result of this computation is accurate to at least 10-bits. The 32-bit floating point result is placed in *dst*.z. By default, this instruction operates on *src0.w*, but can operate on any component by swizzling it into the fourth component.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f;
if (v1[3] == 0)
{
    if(zeroop == IL_ZEROOP_FLT_MAX)
        f = -FLT_MAX;
   else if(zeroop == IL_ZEROOP_INFINITY)
        f = -INFINITY;
    else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
        f = -INFINITY or -FLT_MAX;  # Depends on IL Implementation
}
else if (v1[3] < 0.0)
{
    f = undefined;
}
else
{
    f = log2(v1[3]);
}
v[0] = floor(f);
v[1] = v1[3] / exp2(floor(f));
v[2] = f
v[3] = 1.0;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LOGP
control	29:16	Any value of the enumerated type ILZeroOp(zeroop) except IL_ZEROOP_0. See Table 5.16 on page 5-9.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LN, LOG, LOG_VEC.

Linear Interpolation

Instructions **LRP**

Syntax `lrp dst, src0, src1, src2`

Description Computes the linear interpolation between two vectors.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v3 = EvalSource(src2);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v2[i] * v1[i] + v3[i] * (1 – v1[i]);
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_LRP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Less Than

Instructions **LT**

Syntax `lt dst, src0, src1`

Description Compares two float vectors, *src0* and *src1*, for each component, and writes the result to *dst*. If the comparison is true, 0xFFFFFFFF is returned for that component; otherwise, 0x0000000 is returned. This instruction follows DX10 Floating Point Rules. Denorms are flushed before comparison (original source registers untouched). +0 equals -0. Comparison with NaN returns false.

Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the `pri_modifier_present` and `sec_modifier_present` fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_EQ
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Integer Compare, Unsigned Integer Compare.

Floating Point Multiply and Add

Instructions **MAD**

Syntax `mad[_ieee] dst, src0, src1, src2`

Description Multiplies a component in *src0* by the corresponding component in *src1*. The lower 32 bits of the multiplication is added to the corresponding component in *src2*.The result of the multiply-add operation is placed in the corresponding component of *dst*.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v3 = EvalSource(src2);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v1[i] * v2[i] + v3[i];
WriteResult(v, dst);
```

Format 3-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MAD
control	29:16	MAD: 0 0*n = o, even if n is NaN. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMAD.

Floating Point Maximum

Instructions **MAX**

Syntax `max[_ieee] dst, src0, src1`

MAX	IL_OP_MAX	`max[_ieee] dst, src0, src1`	floating point maximum, *src0*.{x|y|z|w} ≥ *src1*.{x|y|z|w} ? *src0*.{x|y|z|w} : *src1*.{x|y|z|w}

Description Computes maximum value per component. Both max(+0, -0) and max(-0, +0) return +0. NaN has special handling: If one source operand is NaN, then the other source operand is returned (choice made per-component). If both are NaN, any NaN representation is returned.

Denorms are flushed to sign preserved 0s before comparison; if it is the maximum, the flushed denorm is written to *dst*.

If the _ieee flag is included then MIN and MAX follow IEEE-754r rules for minnum and maxnum except denorms are flushed before the comparison is made. In addition, MIN returns -0 and MAX returns +0 for comparisons between -0 and +0.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = (v1[i] > v2[i]) ? v1[i] : v2[i];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MAX
control	29:16	MAX: 0 No special NaN rules. 1 IEEE-style NaN rules.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floating Point Minimum

Instructions **MIN**

Syntax `min[_ieee] dst, src0, src1`

Description Computes minimum value per component. Both min(+0, -0) and min(-0, +0) return -0. NaN has special handling: If one source operand is NaN, then the other source operand is returned (choice made per-component). If both are NaN, any NaN representation is returned. Denorms are flushed to sign preserved 0s before comparison; if it is the minimum, the flushed denorm is written to *dst*.

If the _ieee flag is included then MIN and MAX follow IEEE-754r rules for minnum and maxnum except denorms are flushed before the comparison is made. In addition, MIN returns -0 and MAX returns +0 for comparisons between -0 and +0.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = (v1[i] < v2[i]) ? v1[i] : v2[i];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MIN
control	29:16	MIN: 0 No special NaN rules. 1 IEEE-style NaN rules.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Matrix Multiply by Vector

Instructions **MMUL**

Syntax `mmul_matrix(op) dst, src0, src1`

Description Performs a matrix multiply of the generic matrices *src0* and *src1*. Data must be arranged in one of the formats indicated by the enumerated type ILMatrix(*matrix*). *src1* must be of type IL_REGTYPE_CONST_FLOAT or IL_REGTYPE_TEMP. Relative addressing and source modifiers of *src1* are applied to each row of the matrix beginning with *src1.*

Certain components are not returned by this instruction, depending on the value of *matrix.* Each component is returned only if the component mask field (*component_z_b* or *component_w_a*) of the IL_Dst_Mod token for that component is set to IL_MODCOMP_0 or IL_MODCOMP_1, in which case 0.0 or 1.0 is returned.

Operation:

```
v[2] = v1[0]*v4[0] + v1[1]*v4[1] + v1[2]*v4[2] + v1[3]*v4[3];
}
else if(matrix == IL_MATRIX_3X4)
{
VECTOR v1 = EvalSource(src1);
VECTOR v2 = EvalSource(src2);
VECTOR v3 = EvalSource(src2+1); # Register number of src2 + 1
VECTOR v4 = EvalSource(src2+2); # Register number of src2 + 2
VECTOR v5 = EvalSource(src2+3); # Register number of src2 + 3
v[0] = v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2];
v[1] = v1[0]*v3[0] + v1[1]*v3[1] + v1[2]*v3[2];
v[2] = v1[0]*v4[0] + v1[1]*v4[1] + v1[2]*v4[2];
v[3] = v1[0]*v5[0] + v1[1]*v5[1] + v1[2]*v5[2];
}
else if(matrix == IL_MATRIX_3X3)
{
VECTOR v1 = EvalSource(src1);
VECTOR v2 = EvalSource(src2);
VECTOR v3 = EvalSource(src2+1); # Register number of src2 + 1
VECTOR v4 = EvalSource(src2+2); # Register number of src2 + 2
v[0] = v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2];
v[1] = v1[0]*v3[0] + v1[1]*v3[1] + v1[2]*v3[2];
v[2] = v1[0]*v4[0] + v1[1]*v4[1] + v1[2]*v4[2];
}
else if(matrix == IL_MATRIX_3X2)
{
VECTOR v1 = EvalSource(src1);
VECTOR v2 = EvalSource(src2);
VECTOR v3 = EvalSource(src2+1); # Register number of src2 + 1
v[0] = v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2];
v[1] = v1[0]*v3[0] + v1[1]*v3[1] + v1[2]*v3[2];
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MMUL
control	29:16	Any value of the enumerated type ILMatrix(*matrix*). See Table 5.6 on page 5-3.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Floating Point Modulo

Instructions **MOD**

Syntax mod *dst, src0, src1*

Description Performs the modulo operation. The sign remains intact (for example, mod(-1.7,1.0)=-0.7).

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
{
    if(v2[i] == 0.0)
        v[i] = v1[i];
    else
        v[i] = v1[i] – v2[i] * trunc(v1[i]/v2[i]);
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MOD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Move Data

Instructions **INV_MOV**

Syntax invariant_move *dst, src0*

Summary Move data between registers.

Description Moves value from *src0* to *dst*. As a result of using this instruction, the compiler ensures that any other shader that computes this source using the same instructions gets the identical answer. Generally, use of INV_MOVE prevents many compiler optimizations and lowers performance.

Operation:

```
VECTOR v = EvalSource(src0);
WriteResult(v, dst);
```

Format 1-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_INVATRIANT_MOV
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Move Data Between Registers

Instructions **MOV**

Syntax `mov dst, src0`

Description Places all four components of *src0* into the corresponding components of *dst*.

Operation:

```
VECTOR v = EvalSource(src0);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MOV
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related INV_MOV, MOVA.

Set Address Register Value

Instructions **MOVA**

Syntax `mova[_round] dst, src0`

Description Sets the values of an ADDR register to the truncated or rounded value of *src0*. Relative addressing is not possible on the destination register. The relative_address field of the IL_Dst token must be 0.

See Base Relative Addressing for more information, in Section 3.3, “Relative Addressing,” page 3-2, and Table 2.9 on page 2-8.

If the rounding mode (bit 16) is 1, the values of *src0* are rounded to the nearest integer before they are loaded into the address register. If the rounding mode (bit 16) is 0, the values of *src0* are floored before they are loaded into the address register.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for(i = 0; i < 4; i++)
{
    if(round == 1)
    {
        v[i] = round(v1[i]);
}
else
{
        v[i] = floor(v1[i]);
}
}
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MOVA
control	29:16	**Field Name** / **Bits** / **Description** rounding_mode / 16 / 0: Truncate *src0* before moving to *dst*. 1: Round *src0* before moving to *dst*. *reserved* / 29:25 / Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related INV_MOV, MOV.

Floating Point Multiplication

Instructions **MUL**

Syntax `mul[_ieee] dst, src0, src1`

Description Multiplies two vectors.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v1[i] * v2[i];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_MUL
control	29:16	MUL: 0 DX9 DP2-style multiple. 1 IEEE-style multiply.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DMUL, IMUL_HIGH, UMUL_HIGH.

Not Equal

Instructions **NE**

Syntax `nt dst, src0, src1`

Description Compares two float vectors, *src0* and *src1*, for each component, and writes the result to *dst*. If the comparison is true, 0xFFFFFFFF is returned for that component; otherwise, 0x0000000 is returned. This instruction follows DX10 Floating Point Rules. Denorms are flushed before comparison (original source registers untouched). +0 equals -0. Comparison with NaN returns false.

Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the `pri_modifier_present` and `sec_modifier_present` fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_NE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Double-precision Compare, Integer Compare, Unsigned Integer Compare.

Compute Noise Value

Instructions **NOISE**

Syntax noise_type(*op*) *dst, src0*

Description Uses *src0* as the seed of a pseudo-random number generator to compute a noise value and places it in *dst*.

Operation:

The values in *dst* has the following characteristics:

- They are in the range [-1.0, 1.0]
- Over many iterations, their average value is 0.0.
- The value is repeatable. That is, *dst* is always the same value given the same value of *src0*.
- The value is statistically invariant under rotation (no matter how the domain is rotated, it has the same statistical character).
- The value is statistically invariant under translation (no matter how the domain is translated, it has the same statistical character).
- The value is typically different under translation.
- Over many iterations, the values have a narrow band pass limit in frequency (the values have no visible features larger or smaller than a certain narrow-size range).
- Over many iterations, the values are C_1 continuous everywhere (the first derivative is continuous).

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description		
code	15:0	IL_OP_NOISE		
control	29:16	**Field Name**	**Bits**	**Description**
		noise	19:16	Any value of the enumerated type ILNoiseType(type). See Table 5.24 on page 5-12.
		reserved	29:20	Must be zero.
sec_modifier_present	30	Must be zero.		
pri_modifier_present	31	Must be zero.		

Related None.

Three or Four Component Normalization

Instructions **NRM**

Syntax nrm_zeroop(*op*) *dst*, *src0*

Description Normalize a 4D vector using three or four components. Each component of the vector in *src0* is divided by the square root of the sum of squares of the components in *src0*. This operation has the result of limiting each component to the range [-1.0, 1.0]. The result is placed in *dst*. If nrm4 is one, all four components of *src0* are used in normalization. If nrm4 is zero, only x, y, and z are used in the normalization calculation.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
Float  f;

If( nrm4 == 0)
{
    f = v1[0]*v1[0] + v1[1]*v[1] + v1[2]*v1[2];
else
{
    f = v1[0]*v1[0] + v1[1]*v[1] + v1[2]*v1[2] + v1[3]*v1[3];
}
if (f == 0.0)
{
    if(zeroop == IL_ZEROOP_0)
        f = 0.0;
    else if(zeroop == IL_ ZEROOP_FLT_MAX)
        f = FLT_MAX;
   else if(zeroop == IL_ ZEROOP_INFINITY)
        f = INFINITY;
    else if(zeroop == IL_ ZEROOP_INF_ELSE_MAX)
        f = INFINITY or FLT_MAX;  # Depends on IL Implementation
}
else
{
    f = 1.0/sqrt(f);
}
for (i=0; i < 4; i++)
    v[i] = v1[i] * f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_NRM
control	29:19	See the nested table below.
sec_modifier_present	30	Must be zero.

Field Name	Bits	Description
reserved	17:16	Any value of the enumerated type ILZeroOp(*zeroop*). See Table 5.16 on page 5-9 when the first three components of *src0* are 0.0.
nrm4	18	0: Normalize *src0.*xyz 1: Normalize *src0.*xyzw
reserved	29:19	Must be zero.

Three or Four Component Normalization (Cont.)

	pri_modifier_present 31	Must be zero.

Related None.

Reduce Vector to [–π, π]

Instructions **PIREDUCE**

Syntax pireduce *dst, src0*

Description All four components of the vector in *src0* is reduced to the range [-π, π].

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = (frac((v1[i]/(2*Pi))+0.5)* 2 * PI) – PI;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_PIREDUCE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related NRM.

X to the Power of Y

Instructions **POW**

Syntax `pow dst, src0, src1`

Description Computes *src0*.w to the power of *src1*.w (*src0.w*$^{src1.w}$). By default, this instruction operates on *src0*.w and *src1*.w, but can operate on any component of either operand by swizzling it into the fourth component.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
If(v1[3] < 0.0)
{
    v[0] = v[1] = v[2] = v[3] = undefined;
}
else
{
    v[0] = v[1] = v[2] = v[3] = exp2(v2[3] * log2(v1[3]));
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_POW
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related EXN, EXP, EXP_VEC, EXPP.

Reciprocal

Instructions **RCP**

Syntax `rcp_zeroop(op) dst, src0`

Description Computes the reciprocal of *src0*.w. By default, this instruction operates on *src0*.w and *src1*.w, but can operate on any component of either operand by swizzling it into the fourth component.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f = v1[3];
if (f == 0.0)
{
    if(zeroop == IL_ZEROOP_0)
        f = 0.0;
    else if(zeroop == IL_ZEROOP_FLT_MAX)
        f = FLT_MAX;
   else if(zeroop == IL_ZEROOP_INFINITY)
        f = INFINITY;
    else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
        f = INFINITY or FLT_MAX;  # Depends on IL Implementation
}
else if (f != 1.0)
    f = 1/f;
v[0] = v[1] = v[2] = v[3] = f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RCP
control	17:16	The value of the enumerated type ILZeroOp(*zeroop*), which controls how this instruction behaves when the value of *src0* is 0.0. See Table 5.16 on page 5-9.
	29:18	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related RSQ, RSQ_VEC.

Compute Reflection Vector

Instructions **REFLECT**

Syntax `reflect_normalize` *`dst`*, *`src0`*, *`src1`*

Description Computes the reflection direction of the vector in *src0* using the source vector in *src1*, and placing the direction vector in *dst*. `_normalize` specifies if *src1* is normalized before computing the reflection vector.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
float f = 2 * (v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2] + v1[3]*v2[3]);
if(normalize)
{
    float fnorm = (v2[0]*v2[0] + v2[1]*v2[1] + v2[2]*v2[2] + v2[3]*v2[3]);
for (i=0; i < 4; i++)
   v[i] = fnorm * v1[i] - f * v2[i];
}
else
{
for (i=0; i < 4; i++)
   v[i] = v1[i] - f * v2[i];
}
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_REFLECT
control	16	0: Do not normalize *src1*. 1: Normalize *src1* before computing the reflection vector.
	29:17	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Round

Instructions **RND**

Syntax `rnd dst, src0`

Description Rounds the float value in each component of a floating point vertex (*src0*) to the nearest integer.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = floor(v1[i] + 0.5);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RND
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related ROUND_NEAREST, ROUND_NEG_INF, ROUND_PLUS_INF, ROUND_Z.

Float Round to Nearest Even Float Integral

Instructions **ROUND_NEAREST**

Syntax `round_nearest dst, src0`

Description Rounds the float value in each component of *src0* to the nearest even integral floating-point.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ROUND_NEAR
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related RND, ROUND_NEG_INF, ROUND_PLUS_INF, ROUND_Z.

Float Round to –∞

Instructions **ROUND_NEG_INF**

Syntax round_neginf *dst, src0*

Description Rounds the float values in each component of *src0* towards -∞. This is sometimes called a floor() instruction.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ROUND_NEG_INF
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related FLR, RND, ROUND_NEAREST, ROUND_PLUS_INF, ROUND_Z.

Float Round to +∞

Instructions **ROUND_POS_INF**

Syntax round_posinf *dst, src0*

Description Rounds the float values in each component of *src0* towards ∞. This is sometimes called a ceil() instruction.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ROUND_POS_INF
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related CLG, RND, ROUND_NEAREST, ROUND_NEG_INF, ROUND_Z.

Float Round to Zero

Instructions **ROUND_ZERO**

Syntax round_z *dst, src0*

Description Rounds the float values in each component of *src0* towards zero.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_ROUND_ZERO
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related RND, ROUND_NEAREST, ROUND_NEG_INF, ROUND_PLUS_INF.

Component-wise Reciprocal Square Root

Instructions **RSQ_VEC**

Syntax rsq_vec *dst, src0*

Description Computes the reciprocal of the square root of each component of *src0*.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RSQ_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related RCP, RSQ.

Reciprocal Square Root

Instructions **RSQ**

Syntax `rsq_zeroop(op) dst, src0`

Description Computes the reciprocal of the square root (positive only) of *src0*.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component of *src0*.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
float f = v1[3];
if (f == 0.0)
{
    if(zeroop == IL_ZEROOP_0)
        f = 0.0;
    else if(zeroop == IL_ZEROOP_FLT_MAX)
        f = FLT_MAX;
   else if(zeroop == IL_ZEROOP_INFINITY)
        f = INFINITY;
    else if(zeroop == IL_ZEROOP_INF_ELSE_MAX)
        f = INFINITY or FLT_MAX;  # Depends on IL Implementation
}
else if (f < 0.0)
{
    f = undefined;
}
else
{
    f = 1.0/(float)sqrt(f);
}
v[0] = v[1] = v[2] = v[3] = f;
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_RSQ
control	29:16	Any value of the enumerated type ILZeroOp(*zeroop*). See Table 5.16 on page 5-9.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related RCP, RSQ_VEC.

Set on Comparison

Instructions **SET**

Syntax `set_relop(op) dst, src0, src1`

Description Compares each component of *src0* with the corresponding component of *src1*. The type of comparison performed is dictated by relop(*op*). If the comparison *src0*.{x|y|z|w} relop(*op*) *src1*.{x|y|z|w} evaluates TRUE, the result is 1.0; otherwise, the result is 0.0.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = (v1[i] relop v2[i]) ? 1.0 : 0.0;
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SET
control	29:16	Any value of the enumerated type ILRelOp(*relop*). See Table 5.14 on page 5-8.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Compute Sign

Instructions **SGN**

Syntax sgn *dst*, *src0*

Description Computes the sign of each component of *src0*.

Operation:

```
VECTOR v = EvalSource(src0);
for (i=0; i < 4; i++)
{
    if (v[i] < 0.0)
        v[i] = -1.0;
    else if (v[i] == 0.0)
        v[i] = 0.0;
    else
        v[i] = 1.0;
}
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SGN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Sine (sin)

Instructions **SIN**

Syntax sin *dst*, *src0*

Description Computes the sine of *src0.*w. *src0.*w is in radians for trigonometric functions. *src0.*w must be within the range [-π,π] for each function; otherwise, the results are undefined. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. The maximum absolute error is 0.002.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SIN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Compute Sine and Cosine

Instructions **SINCOS**

Syntax sincos *dst*, *src0*

Description Computes sine and cosine values of *src0*.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. The maximum absolute error is 0.002. The 32-bit floating point results are returned in *dst*.x (COS) and *dst*.y (SIN). *src0*.w must be in the range [-π, π]; otherwise, the results are undefined. *dst*.z and *dst*.w are not written by this instruction; however, the if component_z_b or component_w_a field of the IL_Dst_Mod token is set to IL_MODCOMP_0 or IL_MODCOMP_1, the *dst.z* and *dst.w* are written. That is, *dst.z* and *dst.w* can be set to 0.0 or 1.0 if IL_MODCOMP_0 or IL_MODCOMP_1 is used on the component_z_b or component_w_a field of the IL_Dst_Mod token.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
v[0] = cos(v1[3]);
v[1] = sin(v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SINCOS
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS, SIN.

Component-Wise Sine

Instructions **SIN_VEC**

Syntax sin_vec *dst*, *src0*

Description Computes the sine of each component of *src0* in radians. The maximum absolute error is 0.0008 in the range [-100*π, 100*π].

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SIN_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS, SIN.

Component-Wise Cosine

Instructions **COS_VEC**

Syntax

Function	Opcode	Syntax	Description
COS_VEC	IL_OP_COS_VEC	cos_vec *dst, src0*	cosine, cos(*src0*.xyzw)

Description Computes the cosine of each component of *src0* in radians. The maximum absolute error is 0.0008 in the range [-100*π, 100*π].

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_COS_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS, SIN.

Square Root

Instructions **SQRT**

Syntax sqrt *dst, src0*

Description Computes the square root of *src0*.w. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. If *src0*.w is less than zero, the result is undefined. The result is approximate.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SQRT
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related SQRT_VEC.

Component-Wise Square Root

Instructions **SQRT_VEC**

Syntax sqrt_vec *dst, src0*

Description Computes the square root of each component of *src0*. If a component of *src0* is less than zero, the result for that component is undefined. The result is approximate.

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SQRT_VEC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related SQRT.

Floating Point Subtraction

Instructions **SUB**

Syntax sub *dst, src0, src1*

Description Subtracts each component of *src0* from the corresponding component of *src1*. No carry or borrow beyond the 32-bit values of each component is performed.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v2 = EvalSource(src1);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = v1[i] - v2[i];
WriteResult(v, dst);
```

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_SUB
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DADD.

Tangent (tan)

Instructions **TAN**

Syntax `tan dst, src0`

Description Computes the tangent of *src0*.w. *src0*.w is in radians. *src0*.w must be within the range [-π, π] for each function; otherwise, the results are undefined. By default, this instruction operates on *src0*.w, but can operate on any component by swizzling it into the fourth component. The maximum absolute error is 0.002.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v
v[0] = v[1] = v[2] = v[3] = tan(v1[3]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_TAN
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related COS_VEC, SIN_VEC, SINCOS.

Transpose a 4x4 Matrix

Instructions **TRANSPOSE**

Syntax `transpose dst, src0`

Description Transposes the rows and columns of the 4x4 matrix indicated by *src*.0.

Operation:

```
VECTOR vsrc0 = EvalSource(src0);
VECTOR vsrc1 = EvalSource(src0+1); # Register number of src0 + 1
VECTOR vsrc2 = EvalSource(src0+2); # Register number of src0 + 2
VECTOR vsrc3 = EvalSource(src0+3); # Register number of src0 + 3
VECTOR vdst0;
VECTOR vdst1;
VECTOR vdst2;
VECTOR vdst3;
vdst0[0] = vsrc0[0];
vdst0[1] = vsrc1[0];
vdst0[2] = vsrc2[0];
vdst0[3] = vsrc3[0];
vdst1[0] = vsrc0[1];
vdst1[1] = vsrc1[0];
vdst1[2] = vsrc2[1];
vdst1[3] = vsrc3[1];
vdst2[0] = vsrc0[2];
vdst2[1] = vsrc1[2];
vdst2[2] = vsrc2[2];
vdst2[3] = vsrc3[2];
vdst3[0] = vsrc0[3];
vdst3[1] = vsrc1[3];
vdst3[2] = vsrc2[3];
vdst3[3] = vsrc3[3];
WriteResult(vdst0, dst);
WriteResult(vdst1, dst+1); # Register number of dst + 1
WriteResult(vdst2, dst+2); # Register number of dst + 2
WriteResult(vdst3, dst+3); # Register number of dst + 3
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_TRANSPOSE
control	29:16	Must be the enumerated type ILMatrix(*IL_MATRIX_4X4*). See Table 5.6 on page 5-3.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Truncate

Instructions **TRC**

Syntax `trc dst, src0`

Description Extracts the integer portion (the number to the left of the decimal point) of each component of *src0*. For example, truncate(-1.7) is -1.0, not -2.0 as it is with floor(-1.7). Note, the sign remains intact.

Operation:

```
VECTOR v1 = EvalSource(src0);
VECTOR v;
for (i=0; i < 4; i++)
    v[i] = trunc(v1[i]);
WriteResult(v, dst);
```

Format 1-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_TRC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related DFRAC, FRC.

6.10 Double-Precision Instructions

Double Not Equal Compare

Instructions **D_NE**

Syntax dne *dst*, *src0*, *src1*

Description Component-wise compares two vectors using a float comparison that follows floating point rules. If *src0*.{xy|zw} and the corresponding component pair in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE and returns 0xFFFFFFFF; otherwise, the corresponding component of *dst* is set to FALSE and returns 0x00000000. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the pri_modifier_present and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_NE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Float Compare, Integer Compare, Unsigned Integer Compare.

Double Equal Compare

Instructions **D_EQ**

Syntax deq *dst*, *src0*, *src1*

Description Component-wise compares two vectors using a float comparison that follows floating point rules. If *src0*.{xy|zw} and the corresponding component pair in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE and returns 0xFFFFFFFF; otherwise, the corresponding component of *dst* is set to FALSE and returns 0x00000000. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the pri_modifier_present and sec_modifier_present fields must be zero.

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_EQ
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Float Compare, Integer Compare, Unsigned Integer Compare.

Double Greater-Than or Equal Compare

Instructions **D_GE**

Syntax dge *dst*, *src0*, *src1*

Description Component-wise compares two vectors using a float comparison that follows floating point rules. If *src0*.{xy|zw} and the corresponding component pair in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE and returns 0xFFFFFFFF; otherwise, the corresponding component of *dst* is set to FALSE and returns 0x00000000. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the pri_modifier_present and sec_modifier_present fields must be zero.

	src1								
src0	**-inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**NaN**
-inf	T	F	F	F	F	F	F	F	F
-F	T	T/F	F	F	F	F	F	F	F
-denorm	T	T	T	T	T	T	F	F	F
-0	T	T	T	T	T	T	F	F	F
+0	T	T	T	T	T	T	F	F	F
+denorm	T	T	T	T	T	T	F	F	F
+F	T	T	T	T	T	T	T/F	F	F
+inf	T	T	T	T	T	T	T	T	F
NaN	F	F	F	F	F	F	F	F	F

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_GE
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related Float Compare, Integer Compare, Unsigned Integer Compare.

Double Less-Than Compare

Instructions **D_LT**

Syntax `dlt` *`dst`*, *`src0`*, *`src1`*

Description Component-wise compares two vectors using a float comparison that follows floating point rules. If *src0*.{xy|zw} and the corresponding component pair in *src1* satisfy the comparison condition, the corresponding component of *dst* is set to TRUE and returns 0xFFFFFFFF; otherwise, the corresponding component of *dst* is set to FALSE and returns 0x00000000. Primary and secondary opcode modifiers are not permitted, and no additional control options are supported. Thus, the `pri_modifier_present` and `sec_modifier_present` fields must be zero.

	src1								
src0	**-inf**	**-F**	**-denorm**	**-0**	**+0**	**+denorm**	**+F**	**+inf**	**NaN**
-inf	F	F	F	F	F	F	F	F	F
-F	T	T/F	F	F	F	F	F	F	F
-denorm	T	T	F	F	F	F	F	F	F
-0	T	T	F	F	F	F	F	F	F
+0	T	T	F	F	F	F	F	F	F
+denorm	T	T	F	F	F	F	F	F	F
+F	T	T	T	T	T	T	T/F	F	F
+inf	T	T	T	T	T	T	T	F	F
NaN	F	F	F	F	F	F	F	F	F

Format 2-input, 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_LT
`control`	29:16	Must be zero.
`sec_modifier_present`	30	Must be zero.
`pri_modifier_present`	31	Must be zero.

Related Float Compare, Integer Compare, Unsigned Integer Compare.

Split Double into Fraction and Exponent

Instructions **D_FREXP**

Syntax `dfrexp dst, src0`

Description Output x is zero. Output y is the exponent as an integer. Output wz is the mantissa in range (-1.0,-0.5][0.5,1.0). Abs or negate modifiers can be used on the source. If the input is +-0, all four results are zero. The size is preserved for the wz output. If the input is a NaN, y is set to -1, and wz is set to a NaN = {s, 0x7ff, 1'b1, mant[50:0]} // QNaN. If the input is an +-iff, y is set to -1, and wz is set t0 a NaN = 0xfff8000000000000.

d	-inf/+inf	-0/+0	-denorm/+denorm	NaN
f	NAN	{sign,0}	{sign,0}	d
e	0xFFFFFFFF	0	0	0xFFFFFFFF

Format 1-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_FREXP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related LDEXP.

Add Two Doubles

Instructions **D_ADD**

Syntax dadd *dst*, *src0*, *src1*

Description Abs and negate modifiers can be used on the source. Output is either wz or yx. Output is the correct IEEE result.

				src1			
	-inf	**-F**	**-0**	**+0**	**+F**	**+inf**	**NaN**
scr0							
-inf	-inf	-inf	-inf	-inf	-inf	NaN	NaN
-F	-inf	-F	*src0*	*src0*	+-F or +-0	+inf	NaN
-0	-int	*src1*	-0	+0	*src1*	+inf	NaN
+0	-inf	*src1*	+0	+0	*src1*	+inf	NaN
+F	-inf	+-F or +-0	*src0*	*src0*	-F	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Format 2-input 1-output.

Opcode

Field Name	**Bits**	**Description**
code	15:0	IL_OP_D_ADD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Multiply Two Doubles

Instructions **D_MUL**

Syntax `dmul` *dst*, *src0*, *src1*

Description Abs and negate modifiers can be used on the source. Output is either wz or yx. Output is the correct IEEE result.

	src1								
	-inf	**-F**	**-1.0**	**-0**	**+0**	**+1.0**	**+F**	**+inf**	**NaN**
src0									
-inf	+inf	+inf	+inf	NaN	NaN	inf	-inf	-inf	NaN
-F	+inf	+F	-*src0*	+0	-0	*src0*	-F	-inf	NaN
-1.0	+int	-*src1*	+1.0	+0	-0	-1.0	-*src1*	-inf	NaN
-0	NaN	+0	+0	+0	-0	-0	-0	NaN	NaN
+0	NaN	-0	-0	-0	+0	+0	+0	NaN	NaN
+1.0	-inf	*src1*	-1.0	-0	+0	+1.0	*src1*	+inf	NaN
+F	-inf	-F	-*src0*	-0	+0	*src0*	+F	+inf	NaN
+inf	-inf	-inf	-inf	NaN	NaN	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Format 2-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_MUL
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Divide Two Doubles

Instructions **D_DIV**

Syntax ddiv *dst*, *src0*, *src1*

Description Abs and negate modifiers can be used on the source. Output is either wz or yx. Output is the correct IEEE result.

Format 2-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_DIV
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Pack an EXP and Mantissa into a Double

Instructions **D_LDEXP**

Syntax dldexp *dst*, *src0*, *src1*

Description Puts an exp and a mantissa into a double. The computation is result = source1 * 2 source. Abs and negate modifiers can be used. Output.yx or output.wz are set to the double result. NaN in either input produces a NaN. Overflow produces inf. Underflow produces 0.

Format 2-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_LDEXP
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related FREXP.

Return Fractional Part of a Double

Instructions **D_FRAC**

Syntax dfrac *dst*, *src0*

Description Returns the fractional part of the source in range [0.0 – 1.0). Abs and negate modifiers can be used. Output.yx or output wz is set to the double result.

x	-inf	-F	-1.0	-denorm	-0	+0	+denorm	+1.0	+F	+inf	NaN
result	NAN	[+0.0,+1.0)*	+0	+0	d+0	+0	+0	+0	[+0.0,+1.0)*	NAN	NaN

Format 1-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_FRAC
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Double Multiply and Add

Instructions **D_MULADD**

Syntax dmad *dst*, *src0*, *src1*, *src2*

Description Computes: *src0 * src1 + src2*. Abs and negate can be used on the sources. The output in either yz or wz is the result. The compiler determines if this is identical to separate MULs and ADD based on the chipset. The operation truncates internally, so the result is not identical to doing a MUL followed by an ADD.

Format 3-input 1-output.

Opcode

Field Name	Bits	Description
code	15:0	IL_OP_D_MULADD
control	29:16	Must be zero.
sec_modifier_present	30	Must be zero.
pri_modifier_present	31	Must be zero.

Related None.

Glossary of Terms

Term	Description
*	Any number of alphanumeric characters in the name of a microcode format, microcode parameter, or instruction.
< >	Angle brackets denote streams.
[1,2)	A range that includes the left-most value (in this case, 1) but excludes the right-most value (in this case, 2).
[1,2]	A range that includes both the left-most and right-most values (in this case, 1 and 2).
{BUF, SWIZ}	One of the multiple options listed. In this case, the string *BUF* or the string *SWIZ*.
{x | y}	One of the multiple options listed. In this case, x or y.
0.0	A single-precision (32-bit) floating-point value.
0x	Indicates that the following is a hexadecimal number.
1011b	A binary value, in this example a 4-bit value.
29'b0	29 bits with the value 0.
7:4	A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first.
ABI	Application Binary Interface.
absolute	A displacement that references the base of a code segment, rather than an instruction pointer. See relative.
active mask	A 1-bit-per-pixel mask that controls which pixels in a "quad" are really running. Some pixels general-purpose not be running if the current "primitive" does not cover the whole quad. A mask can be updated with a PRED_SET* ALU instruction, but updates do not take effect until the end of the ALU clause.
address stack	A stack that contains only addresses (no other state). Used for flow control. Popping the address stack overrides the instruction address field of a flow control instruction. The address stack is only modified if the flow control instruction decides to jump.
ACML	AMD Core Math Library. Includes implementations of the full BLAS and LAPACK routines, FFT, Math transcendental and Random Number Generator routines, stream processing backend for load balancing of computations between the CPU and stream processor.
aL (also AL)	Loop register. A three-component vector (x, y and z) used to count iterations of a loop.
allocate	To reserve storage space for data in an output buffer ("scratch buffer," "ring buffer," "stream buffer," or "reduction buffer") or for data in an input buffer ("scratch buffer" or "ring buffer") before exporting (writing) or importing (reading) data or addresses to, or from that buffer. Space is allocated only for data, not for addresses. After allocating space in a buffer, an "export" operation can be done.

Term	Description
ALU	Arithmetic Logic Unit. Responsible for arithmetic operations like addition, subtraction, multiplication, division, and bit manipulation on integer and floating point values. In stream computing, these are known as *stream cores*. ALU.[X,Y,Z,W] - an ALU that can perform four vector operations in which the four operands (integers or single-precision floating point values) do not have to be related. It performs “SIMD” operations. Thus, although the four operands need not be related, all four operations execute the same instruction. ALU.Trans - An ALU unit that can perform one ALU.Trans (transcendental, scalar) operation, or advanced integer operation, on one integer or single-precision floating-point value, and replicate the result. A single instruction can co-issue four ALU.Trans operations to an ALU.[X,Y,Z,W] unit and one (possibly complex) operation to an ALU.Trans unit, which can then replicate its result across all four component being operated on in the associated ALU.[X,Y,Z,W] unit.
ATI Stream™ SDK	A complete software development suite from ATI for developing applications for ATI Stream Processors. Currently, the ATI Stream SDK includes Brook+ and CAL.
AR	Address register.
aTid	Absolute thread id. It is the ordinal count of all threads being executed (in a draw call).
b	A bit, as in *1Mb* for one megabit, or *lsb* for least-significant bit.
B	A byte, as in *1MB* for one megabyte, or *LSB* for least-significant byte.
BLAS	Basic Linear Algebra Subroutines.
border color	Four 32-bit floating-point numbers (XYZW) specifying the border color.
branch granularity	The number of threads executed during a branch. For ATI, branch granularity is equal to wavefront granularity.
brcc	Source-to-source meta-compiler that translates Brook programs (.br files) into device-dependent kernels embedded in valid C++ source code that includes CPU code and stream processor device code, which later are linked into the executable.
Brook+	A high-level language derived from C which allows developers to write their applications at an abstract level without having to worry about the exact details of the hardware. This enables the developer to focus on the algorithm and not the individual instructions run on the stream processor. Brook+ is an enhancement of Brook, which is an open source project out of Stanford. Brook+ adds additional features available on ATI Stream Processors and provides a CAL backend.
brt	The Brook runtime library that executes pre-compiled kernel routines invoked from the CPU code in the application.
burst mode	The limited write combining ability. See write combining.
byte	Eight bits.
cache	A read-only or write-only on-chip or off-chip storage space.
CAL	Compute Abstraction Layer. A device-driver library that provides a forward-compatible interface to ATI Stream processor devices. This lower-level API gives users direct control over the hardware: they can directly open devices, allocate memory resources, transfer data and initiate kernel execution. CAL also provides a JIT compiler for ATI IL.
CF	Control Flow.
cfile	Constant file or constant register.
channel	A component in a vector.

Term	**Description**
clamp	To hold within a stated range.
clause	A group of instructions that are of the same type (all stream core, all fetch, etc.) executed as a group. A clause is part of a CAL program written using the stream processor ISA. Executed without pre-emption.
clause size	The total number of slots required for an stream core clause.
clause temporaries	Temporary values stored at GPR that do not need to be preserved past the end of a clause.
clear	To write a bit-value of 0. Compare “set”.
command	A value written by the host processor directly to the stream processor. The commands contain information that is not typically part of an application program, such as setting configuration registers, specifying the data domain on which to operate, and initiating the start of data processing.
command processor	A logic block in the R700 (HD4000-family of devices) that receives host commands, interprets them, and performs the operations they indicate.
component	(1) A 32-bit piece of data in a “vector”. (2) A 32-bit piece of data in an array. (3) One of four data items in a 4-component register.
compute shader	Similar to a pixel shader, but exposes data sharing and synchronization.
constant buffer	Off-chip memory that contains constants. A constant buffer can hold up to 1024 four-component vectors. There are fifteen constant buffers, referenced as cb0 to cb14. An immediate constant buffer is similar to a constant buffer. However, an immediate constant buffer is defined within a kernel using special instructions. There are fifteen immediate constant buffers, referenced as icb0 to icb14.
constant cache	A constant cache is a hardware object (off-chip memory) used to hold data that remains unchanged for the duration of a kernel (constants). “Constant cache” is a general term used to describe constant registers, constant buffers or immediate constant buffers.
constant file	Same as constant register.
constant index register	Same as “AR” register.
constant registers	On-chip registers that contain constants. The registers are organized as four 32-bit component of a vector. There are 256 such registers, each one 128-bits wide.
constant waterfalling	Relative addressing of a constant file. See waterfalling.
context	A representation of the state of a CAL device.
core clock	See engine clock. The clock at which the stream processor stream core runs.
CPU	Central Processing Unit. Also called host. Responsible for executing the operating system and the main part of the application. The CPU provides data and instructions to the stream processor.
CRs	Constant registers. There are 512 CRs, each one 128 bits wide, organized as four 32-bit values.
CS	Compute shader. A shader type, analogous to VS/PS/GS/ES.
CTM	Close-to-Metal. A thin, HW/SW interface layer. This was the predecessor of the ATI CAL.

Term	Description
DC	Data Copy Shader.
device	A *device* is an entire ATI Stream processor.
DMA	Direct-memory access. Also called DMA engine. Responsible for independently transferring data to, and from, the stream processor's local memory. This allows other computations to occur in parallel, increasing overall system performance.
double word	Dword. Two words, or four bytes, or 32 bits.
double quad word	Eight words, or 16 bytes, or 128 bits. Also called "octword."
domain of execution	A specified rectangular region of the output buffer to which threads are mapped.
DPP	Data-Parallel Processor.
dst.X	The X "slot" of an destination operand.
dword	Double word. Two words, or four bytes, or 32 bits.
element	A component in a vector.
engine clock	The clock driving the stream core and memory fetch units on the stream processor stream processor core.
enum(7)	A seven-bit field that specifies an enumerated set of decimal values (in this case, a set of up to 27 values). The valid values can begin at a value greater than, or equal to, zero; and the number of valid values can be less than, or equal to, the maximum supported by the field.
event	A token sent through a pipeline that can be used to enforce synchronization, flush caches, and report status back to the host application.
export	To write data from GPRs to an output buffer (scratch, ring, stream, frame or global buffer, or to a register), or to read data from an input buffer (a "scratch buffer" or "ring buffer") to GPRs. The term "export" is a partial misnomer because it performs both input and output functions. Prior to exporting, an allocation operation must be performed to reserve space in the associated buffer.
FFT	Fast Fourier Transform.
flag	A bit that is modified by a CF or stream core operation and that can affect subsequent operations.
FLOP	Floating Point Operation.
flush	To writeback and invalidate cache data.
frame	A single two-dimensional screenful of data, or the storage space required for it.
frame buffer	Off-chip memory that stores a frame.
FS	Fetch subroutine. A global program for fetching vertex data. It can be called by a "vertex shader" (VS), and it runs in the same thread context as the vertex program, and thus is treated for execution purposes as part of the vertex program. The FS provides driver independence between the process of fetching data required by a VS, and the VS itself. This includes having a semantic connection between the outputs of the fetch process and the inputs of the VS.
function	A subprogram called by the main program or another function within an ATI IL stream. Functions are delineated by `FUNC` and `ENDFUNC`.

Term	Description
gather	Reading from arbitrary memory locations by a thread.
gather stream	Input streams are treated as a memory array, and data elements are addressed directly.
global buffer	Memory space containing the arbitrary address locations to which uncached kernel outputs are written. Can be read either cached or uncached. When read in uncached mode, it is known as mem-import. Allows applications the flexibility to read from and write to arbitrary locations in input buffers and output buffers, respectively.
GPGPU	General-purpose stream processor. A stream processor that performs general-purpose calculations.
GPR	General-purpose register. GPRs hold vectors of either four 32-bit IEEE floating-point, or four 8-, 16-, or 32-bit signed or unsigned integer or two 64-bit IEEE double precision data components (values). These registers can be indexed, and consist of an on-chip part and an off-chip part, called the “scratch buffer,” in memory.
GPU	Graphics Processing Unit. An integrated circuit that renders and displays graphical images on a monitor. Also called Graphics Hardware, Stream Processor, and Data Parallel Processor.
GPU engine clock frequency	Also called 3D engine speed.
GS	Geometry Shader.
HAL	Hardware Abstraction Layer.
host	Also called CPU.
iff	If and only if.
IL	Intermediate Language. In this manual, the ATI version: ATI IL. A pseudo-assembly language that can be used to describe kernels for stream processors. ATI IL is designed for efficient generalization of stream processor instructions so that programs can run on a variety of platforms without having to be rewritten for each platform.
in flight	A thread currently being processed.
instruction	A computing function specified by the *code* field of an IL_OpCode token. Compare “opcode”, “operation”, and “instruction packet”.
instruction packet	A group of tokens starting with an IL_OpCode token that represent a single ATI IL instruction.
int(2)	A 2-bit field that specifies an integer value.
ISA	Instruction Set Architecture. The complete specification of the interface between computer programs and the underlying computer hardware.
kcache	A memory area containing “waterfall” (off-chip) constants. The cache lines of these constants can be locked. The “constant registers” are the 256 on-chip constants.
kernel	A small, user-developed program that is run repeatedly on a stream of data. A parallel function that operates on every element of input streams. A device program is one type of kernel. Unless otherwise specified, an ATI Stream processor program is a kernel composed of a main program and zero or more functions. Also called Shader Program. This is not to be confused with an OS kernel, which controls hardware.
LAPACK	Linear Algebra Package.

Term	Description
LERP	Linear Interpolation.
local memory fetch units	Dedicated hardware that a) processes fetch instructions, b) requests data from the memory controller, and c) loads registers with data returned from the cache. They are run at stream processor stream core or engine clock speeds. Formerly called texture units.
LOD	Level Of Detail.
loop index	A register initialized by software and incremented by hardware on each iteration of a loop.
lsb	Least-significant bit.
LSB	Least-significant byte.
MAD	Multiply-Add. A fused instruction that both multiplies and adds.
mask	(1) To prevent from being seen or acted upon. (2) A field of bits used for a control purpose.
MBZ	Must be zero.
mem-export	An ATI IL term random writes to the global buffer.
mem-import	Uncached reads from the global buffer.
memory clock	The clock driving the memory chips on the stream processor.
microcode format	An encoding format whose fields specify instructions and associated parameters. Microcode formats are used in sets of two or four. For example, the two mnemonics, `CF_DWORD[0,1]` indicate a microcode-format pair, `CF_DWORD0` and `CF_DWORD1`.
MIMD	Multiple Instruction Multiple Data. – Multiple SIMD units operating in parallel (Multi-Processor System) – Distributed or shared memory
MRT	Multiple Render Target. One of multiple areas of local stream processor memory, such as a “frame buffer”, to which a graphics pipeline writes data.
MSAA	Multi-Sample Anti-Aliasing.
msb	Most-significant bit.
MSB	Most-significant byte.
neighborhood	A group of four threads in the same wavefront that have consecutive thread IDs (Tid). The first Tid must be a multiple of four. For example, threads with Tid = 0, 1, 2, and 3 form a neighborhood, as do threads with Tid = 12, 13, 14, and 15.
normalized	A numeric value in the range [a, b] that has been converted to a range of 0.0 to 1.0 using the formula: normalized value = value/ (b–a+ 1)
oct word	Eight words, or 16 bytes, or 128 bits. Same as “double quad word”.
opcode	The numeric value of the *code* field of an “instruction”. For example, the opcode for the CMOV instruction is decimal 16 (0x10).
opcode token	A 32-bit value that describes the operation of an instruction.
operation	The function performed by an “instruction”.
PaC	Parameter Cache.

Term	**Description**
page	A program-controlled cache, backing up processor-accessible memory.
PCI Express	A high-speed computer expansion card interface used by modern graphics cards, stream processors and other peripherals needing high data transfer rates. Unlike previous expansion interfaces, PCI Express is structured around point-to-point links. Also called PCIe.
PoC	Position Cache.
pop	Write “stack” entries to their associated hardware-maintained control-flow state. The POP_COUNT field of the CF_DWORD1 microcode format specifies the number of stack entries to pop for instructions that pop the stack. Compare “push.”
pre-emption	The act of temporarily interrupting a task being carried out on a computer system, without requiring its cooperation, with the intention of resuming the task at a later time.
processor	Unless otherwise stated, the ATI Stream Processor.
program	Unless otherwise specified, a program is a set of instructions that can run on the ATI Stream Processor. A device program is a type of kernel.
PS	Pixel Shader.
push	Read hardware-maintained control-flow state and write their contents onto the stack. Compare pop.
PV	Previous vector register. It contains the previous four-component vector result from a ALU.[X,Y,Z,W] unit within a given clause.
quad	Group of 2x2 threads in the domain. Always processed together.
rasterization	The process of mapping threads from the domain of execution to the SIMD engine. This term is a carryover from graphics, where it refers to the process of turning geometry, such as triangles, into pixels.
rasterization order	The order of the thread mapping generated by rasterization.
RB	Ring Buffer.
register	A 128-bit address mapped memory space consisting of four 32-bit components.
relative	Referencing with a displacement (also called offset) from an index register or the loop index, rather than from the base address of a program (the first control flow [CF] instruction).
render backend unit	The hardware units in a stream processor stream processor core responsible for writing the results of a kernel to output streams by writing the results to an output cache and transferring the cache data to memory.
resource	A block of memory used for input to, or output from, a kernel.
ring buffer	An on-chip buffer that indexes itself automatically in a circle.
Rsvd	Reserved.
sampler	A structure that contains information necessary to access data in a resource. Also called Fetch Unit.
SC	Shader Compiler.
scalar	A single data component, unlike a vector which contains a set of two or more data elements.

Term	Description
scatter	Writes (by uncached memory) to arbitrary locations.
scatter write	Kernel outputs to arbitrary address locations. Must be uncached. Must be made to a memory space known as the global buffer.
scratch buffer	A variable-sized space in off-chip-memory that stores some of the “GPRs”.
set	To write a bit-value of 1. Compare “clear”.
shader processor	Also called thread processor.
shader program	User developed program. Also called kernel.
SIMD	Single instruction multiple data. – Each SIMD receives independent stream core instructions. – Each SIMD applies the instructions to multiple data elements.
SIMD Engine	A collection of thread processors, each of which executes the same instruction per cycle.
SIMD pipeline	A hardware block consisting of five stream cores, one stream core instruction decoder and issuer, one stream core constant fetcher, and support logic. All parts of a SIMD pipeline receive the same instruction and operate on different data elements. Also known as “slice.”
Simultaneous Instruction Issue	Input, output, fetch, stream core, and control flow per SIMD engine.
SKA	Stream KernelAnalyzer. A performance profiling tool for developing, debugging, and profiling stream kernels using high-level stream computing languages.
slot	A position, in an “instruction group,” for an “instruction” or an associated literal constant. An ALU instruction group consists of one to seven slots, each 64 bits wide. All ALU instructions occupy one slot, except double-precision floating-point instructions, which occupy either two or four slots. The size of an ALU clause is the total number of slots required for the clause.
SPU	Shader processing unit.
src0, src1, etc.	In floating-point operation syntax, a 32-bit source operand. Src0_64 is a 64-bit source operand.
stage	A sampler and resource pair.
stream	A collection of data elements of the same type that can be operated on in parallel.
stream buffer	A variable-sized space in off-chip memory that stores an instruction stream. It is an output-only buffer, configured by the host processor. It does not store inputs from off-chip memory to the processor.
stream core	The fundamental, programmable computational units, responsible for performing integer, single, precision floating point, double precision floating point, and transcendental operations. They execute VLIW instructions for a particular thread. Each stream processor stream core handles a single instruction within the VLIW instruction.
stream operator	A node that can restructure data.
stream processor	A parallel processor capable of executing multiple threads of a kernel in order to process streams of data.
swizzling	To copy or move any component in a source vector to any element-position in a destination vector. Accessing elements in any combination.

Term	Description
thread	One invocation of a kernel corresponding to a single element in the domain of execution. An instance of execution of a shader program on an ALU.
thread group	It contains one or more thread blocks. Threads in the same thread-group but different thread-blocks might communicate to each through global per-SIMD shared memory. This is a concept mainly for global data share (GDS). A thread group can contain one or more wavefronts, the last of which can be a partial wavefront. All wavefronts in a thread group can run on only one SIMD engine; however, multiple thread groups can share a SIMD engine, if there are sufficient resources.
thread processor	The hardware units in a SIMD engine responsible for executing the threads of a kernel. It executes the same instruction per cycle. Each thread processor contains multiple stream cores. Also called shader processor.
thread-block	A group of threads which might communicate to each other through local per SIMD shared memory. It can contain one or more wavefronts (the last wavefront can be a partial wavefront). A thread-block (all its wavefronts) can only run on one SIMD engine. However, multiple thread blocks can share a SIMD engine, if there are enough resources to fit them in.
Tid	Thread id within a thread block. An integer number from 0 to Num_threads_per_block-1
token	A 32-bit value that represents an independent part of a stream or instruction.
uncached read/write unit	The hardware units in a stream processor responsible for handling uncached read or write requests from local memory on the stream processor.
vector	(1) A set of up to four related values of the same data type, each of which is an element. For example, a vector with four elements is known as a “4-vector” and a vector with three elements is known as a “3-vector”. (2) See “AR”. (3) See ALU.[X,Y,Z,W].
VLIW design	Very Long Instruction Word. – Co-issued up to 6 operations (5 stream cores + 1 FC) – 1.25 Machine Scalar operation per clock for each of 64 data elements – Independent scalar source and destination addressing
waterfall	To use the address register (AR) for indexing the GPRs. Waterfall behavior is determined by a “configuration registers.”
wavefront	Group of threads executed together on a single SIMD engine. Composed of quads. A full wavefront contains 64 threads; a wavefront with fewer than 64 threads is called a partial wavefront. For the HD4000-family of devices, there are 64. 32, 16 threads in a full wavefront.
write combining	Combining several smaller writes to memory into a single larger write to minimize any overhead associated with write commands.

Index

D

E

K

L

M

S

T

U

V

W

X

Z