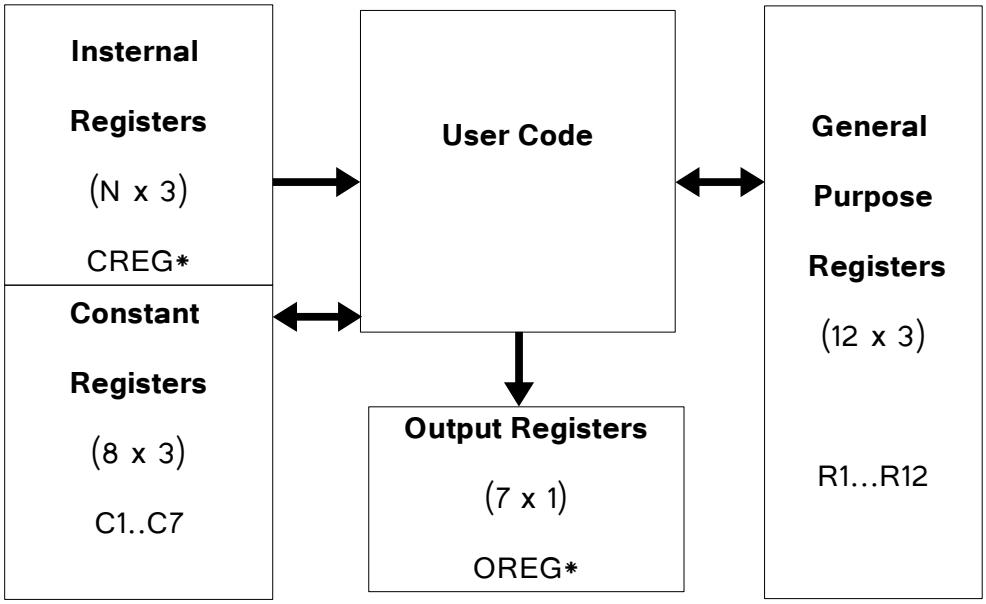
THEIA Programming Manual

Table of Contents

Index of Tables

1. Overview

THEIA's pixels shaders are written using the THEIA programming language. This is an assembly programming language consisting of a series of instructions that operate on a series of registers. There is no stack.
The following diagram illustrates THEIA's programming model.



Registers: THEAI has 4 types of registers.

- **Internal Registers**:
  Accessibility: Read-Only.
  Scope: The values of these registers can change from one pixel iteration to the next.
  Purpose: They are special registers used by the internal GPU routines. Sometimes you may want to read from these registers. Examples: `CREG_CAMERA_POSITION`, `CREG_RESOLUTION`, `CREG_PIXEL_2D_POSITION`, etc.

- **General purpose Registers**:
  Accessibility: Read-Write.
  Scope: The values of these registers can change from one pixel iteration to the next. This means that even though you can write any value you want, you are **not** guaranteed that this value will be persistent from one pixel iteration to the next.
  Purpose: You can basically store whatever you want in these registers, but again, the values are only kept for the duration of your shader.

- **Constant Registers:**

Accessibility: Read-Write.
Scope: The values of these registers are kept for the entire duration of the GPU execution. This means that if you store a value, this value is kept until you replace it with a different value or reset the GPU.
Purpose: Although you can use these registers as you like, the idea is to store constants during a special execution stage, so that you can reuse these values for each pixel iteration.

- **Output Registers:**
  Accessibility: Write-Only.
  Scope: The values of these registers can change from one pixel iteration to the next.
  Purpose: These are special registers that indicate an output from a particular stage of THEIA's execution. Examples: `OREG_PIXEL_COLOR`, `OREG_TEX_COORD1`, `OREG_TEXWEIGHT1`, etc.

Instructions:

THEIA has arithmetic, logic and flow control instructions. There are two flavors of instructions:

Type 1: **OPERATION** DESTINATION SOURCE_REGISTER1 SOURCE_REGISTER2

Type 2: **OPERATION** DESTINATION IMMEDIATE_VALUE

2. Registers

Configuration Registers.

The configuration registers are Read Only for EXE and Write Only for IO.

The configuration registers are written once by IO during the machine's initial configuration stage. These same values are used for the rest of the life of the GPU or until a new configuration event happens.

The configuration register stored important values that are user to set up the variables of the various internal algorithms, such as the screen resolution, axis aligned bounding box boundaries, and light configuration.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x0000 (0)	**CREG_LIGHT_INFO** Configuration information regarding the lights in the scene.	Not used	Number of lights in the scene	Number of configuration packets
0x0001 (1)	**CREG_CAMERA_POSITION** The current position of the camera.	Camera position X	Camera position Y	Camera position Z
0x00002 (2)	**CREG_PROJECTION_WINDOW_MIN**	Minimum 2D X coordinate of the projection window	Minimum 2D Y coordinate of the projection window	Not used
0x00003 (3)	**CREG_PROJECTION_WINDOW_MAX**	Maximum 2D X coordinate of the projection window	Maximum 2D Y coordinate of the projection window	Not used
0x00004 (4)	**CREG_RESOLUTION** The resolution of the output image.	Screen resolution Width	Screen resolution Height	Not used
0x00005 (5)	**CREG_TEXTURE_SIZE**[1] [2] The size of the texture in memory.	Texture Widht	Texture Height	Not used
0x00006 (6)	**CREG_PIXEL_2D_INITIAL_POSITION**[3] Determines where the row and column where the current core will start casting rays at the projection window.	Initial X position on the projection window	Initial Y position on the projection window	Not used
0x00007 (7)	**CREG_PIXEL_2D_FINAL_POSITION**[4] Determines where the row and column where the current core will stop casting rays at the projection window.	Final X position on the projection window	Final Y position on the projection window	Not used
0x00007 (8)	**CREG_FIRST_LIGTH** Pointer to the first light. Registers 8-42 are reserved to store Light structures.	Light Position X	Light Position Y	Light Position Z

1 Only square textures are currently supported. Ie Width = Height.
2 Texture Size must be a power of 2.
3 In single core operation, this value is always zero.
4 In single core operation, this value is always equal to CREG_RESOLUTION.

0x0002A (42)	**CREG_AABBMIN** AABB (Axis Aligned Bounding Box) Minimum coordinate	AABB Minimum X coordiante	AABB Minimum Y coordiante	AABB Minimum Z coordiante
0x0002B (42)	**CREG_AABBMAX** AABB (Axis Aligned Bounding Box) Maximum coordinate	AABB Minimum X coordiante	AABB Minimum Y coordiante	AABB Minimum Z coordiante

Table 1: Configuration registers

Swap Registers.

The Swap registers are Read Only for EXE and Write Only for IO. There are 2 copies of this memory region. While EXE is reading from one of copies, IO is writing into the other copy, then on the next iteration the memory areas get swapped with each other and the process repeats once again.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x002C (44)	**CREG_V0** The first vertex of the current triangle for this iteration.	Vertex X Coordinate	Vertex Y Coordinate	Vertex Z Coordinate
0x002D (45)	**CREG_UV0** The texture UV coordinate of the first vertex of the current triangle for this iteration.	Texture U coordinate.	Texture V coordinate.	Not used
0x002E (46)	**CREG_V1** The second vertex of the current triangle for this iteration.	Vertex X Coordinate	Vertex Y Coordinate	Vertex Z Coordinate
0x002F (47)	**CREG_UV1** The texture UV coordinate of the second vertex of the current triangle for this iteration.	Texture U coordinate.	Texture V coordinate.	Not used
0x0030 (48)	**CREG_V2** The third vertex of the current triangle for this iteration.	Vertex X Coordinate	Vertex Y Coordinate	Vertex Z Coordinate
0x0031 (49)	**CREG_UV2** The texture UV coordinate of the third vertex of the current triangle for this iteration.	Texture U coordinate.	Texture V coordinate.	Not used
0x0032 (50)	**CREG_TRI_DIFFUSE** The diffuse color of the current triangle for this iteration.	Red color component	Green color component	Blue color component
0x0035 (53)	**CREG_TEX_COLOR1** The first texture color fetched from the external texture memory for this iteration. A total of 6 texture color components are fetch in order to use bi-linear filtering.	Red color component	Green color component	Blue color component
0x0036 (54)	**CREG_TEX_COLOR2** The second texture color fetched from the external texture memory for	Red color component	Green color component	Blue color component

	this iteration. A total of 4 texture color components are fetch in order to use bi-linear filtering.			
0x0037 (55)	**RESERVED** Internally used by IO to store temporary values during texture color fetching.	RESERVED	RESERVED	RESERVED
0x0038 (56)	**CREG_TEX_COLOR3** The third texture color fetched from the external texture memory for this iteration. A total of 4 texture color components are fetch in order to use bi-linear filtering.	Red color component	Green color component	Blue color component
0x0039 (57)	**CREG_TEX_COLOR4** The last texture color fetched from the external texture memory for this iteration. A total of 4 texture color components are fetch in order to use bi-linear filtering.	Red color component	Green color component	Blue color component
0x003A (58)	**RESERVED** Internally used by IO to store temporary values during texture color fetching.	RESERVED	RESERVED	RESERVED

Table 2: Swap registers

Constant Registers.

Constant R/W for EXE. Constant register keep their values for the entire execution of the GPU or until Reset. This means if the user stores a value in a constant register, this value is preserved for all the iterations unlike the user registers.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x0040 (64)	**C1** User constant register	User defined.	User defined.	User defined.
0x0041 (65)	**C2** User constant register	User defined.	User defined.	User defined.
0x0042 (66)	**C3** User constant register	User defined.	User defined.	User defined.
0x0043 (67)	**C4** User constant register	User defined.	User defined.	User defined.
0x0044 (68)	**C5** User constant register	User defined.	User defined.	User defined.
0x0045 (69)	**C6** User constant register	User defined.	User defined.	User defined.
0x0046 (70)	**C7** User constant register	User defined.	User defined.	User defined.

Table 3: Constant registers

User Registers.

The user register are general Purpose registers. The user may store whatever he/she wants in here at these memory locations.
The user registers are R/W for EXE, IO can not read or write to there registers. Unlike constant registers, the value stored in a user register is on only preserved for the duration of a single iteration. This means for example, that if the register R1 is set to the value 0xCAFE at iteration I_i, then at iteration I_{i+1} the value stored at R1 may have changed. This happens because the internal code of THEIA can also use the general purpose registers, therefor potentially overwriting the previous values stored by the user.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x0047 (71)	**R1** User register	User defined.	User defined.	User defined.
0x0048 (72)	**R2** User register	User defined.	User defined.	User defined.
0x0049 (73)	**R3** User register	User defined.	User defined.	User defined.
0x004A (74)	**R4** User register	User defined.	User defined.	User defined.
0x004B (75)	**R5** User register	User defined.	User defined.	User defined.
0x004C (76)	**R6** User register	User defined.	User defined.	User defined.
0x004D (77)	**R7** User register	User defined.	User defined.	User defined.
0x004E (78)	**R8** User register	User defined.	User defined.	User defined.
0x004F (79)	**R9** User register	User defined.	User defined.	User defined.
0x0050 (80)	**R10** User register	User defined.	User defined.	User defined.
0x0051 (81)	**R11** User register	User defined.	User defined.	User defined.
0x0052 (82)	**R12** User register	User defined.	User defined.	User defined.

Table 4: User registers

Internal Registers.

Internal registers are meant for the machine's internal code to use, since those registers are R/W for EXE the user is allowed to modify them. However modification of the internal registers by user code is highly discouraged since this may result in unexpected behavior.

Note: At the present time, the Internal register design is not fully optimized. Many of these registers can simply be replaced by general purpose register at the RTL level.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x0053 (83)	**CREG_PROJECTION_WINDOW_SCALE** The scale of the projection window. This is calculated as: (CREG_PROJECTION_WINDOW_MAX – CREG_PROJECTION_WINDOW_MIN)/ CREG_SCREEN_RESOLUTION	Projection Window scale.	Projection Window scale.	Projection Window scale.
0x0054 (84)	**CREG_UNORMALIZED_DIRECTION** The un-normalized direction of the ray for the current iteration. This is calculated as: (CREG_PIXEL_2D_POSITION * CREG_PROJECTION_WINDOW_SCALE – CREG_PROJECTION_WINDOW_MIN) -CREG_CAMERA_POSITION	Direction X	Direction Y	Direction Z
0x0054 (85)	**CREG_RAY_DIRECTION** The direction of the current ray. This is the normalized vector of CREG_UNORMALIZED_DIRECTION	Direction X	Direction Y	Direction Z
0x0055 (86)	**CREG_E1_LAST** This is the last value of CREG_V1 – CREG_V0 that is closest to the camera for the current iteration.	X coordinate	Y coordinate	Z coordinate
0x0056 (87)	**CREG_E2_LAST** This is the last value of CREG_V2 – CREG_V0 that is closest to the camera for the current iteration.	X coordinate	Y coordinate	Z coordinate
0x0057 (88)	**CREG_T**[5] Partial result used by the default ray-triangle intersection algorithm. This is calculated as:	T value	T value	T value

5 This register is not used if the default triangle ray intersection algorithm gets overwritten by the user.

	CREG_CAMERA_POSITION / CREG_V0			
0x0058 (89)	**CREG_P** [6] Partial result used by the default ray-triangle intersection algorithm. This is calculated as: CrossProduct(CREG_RAY_DIRECTION ,CREG_E2)	Px	Py	Pz
0x0059 (90)	**CREG_Q** [7] Partial result used by the default ray-triangle intersection algorithm. This is calculated as: CrossProduct(CREG_T ,CREG_E1)	Qx	Qy	Qz
0x005A (91)	**CREG_UV0_LAST** The texture UV coordinates from the first vertex of the triangle closest to the camera for the current iteration.	U coordinate	V coordinate	Not used
0x005B (92)	**CREG_UV1_LAST** The texture UV coordinates from the second vertex of the triangle closest to the camera for the current iteration.	U coordinate	V coordinate	Not used
0x005C (93)	**CREG_UV2_LAST** The texture UV coordinates from the third vertex of the triangle closest to the camera for the current iteration.	U coordinate	V coordinate	Not used
0x005D (94)	**CREG_TRI_DIFFUSE_LAST** The diffuse color of the triangle closest to the camera for the current iteration.	Red component	Green component	Blue component
0x005E (95)	**CREG_LAST_t** Final result from default ray-triangle	t	t	t

6 This register is not used if the default triangle ray intersection algorithm gets overwritten by the user.
7 This register is not used if the default triangle ray intersection algorithm gets overwritten by the user.

	intersection algorithm of the triangle closest to the camera for the current iteration.			
0x0060 (96)	**CREG_LAST_u** Final result from default ray-triangle intersection algorithm of the triangle closest to the camera for the current iteration.	u	u	u
0x0061 (97)	**CREG_LAST_v** Final result from default ray-triangle intersection algorithm of the triangle closest to the camera for the current iteration.	v	v	v
0x0062 (98)	**CREG_COLOR_ACC** Color accumulator register. Stores the addition of the color contributions for a pixel across all the iterations.	Red component	Green component	Blue component
0x0063 (99)	**CREG_t** Final result from default ray-triangle intersection algorithm. This is calculated as: CREG_H1 / CREG_DELTA	t	t	t
0x0064 (100)	**CREG_E1** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: CREG_V1 – CREG_V0	X coordinate	Y coordinate	Z coordinate
0x0065 (101)	**CREG_E2** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: CREG_V2 - CREG_V0	X coordinate	Y coordinate	Z coordinate

0x0066 (102)	**CREG_DELTA** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: DotProduct(CREG_P, CREG_E1)	Delta	Delta	Delta
0x0067 (103)	**CREG_u** Final result from default ray-triangle intersection algorithm. This is calculated as: CREG_H2 / CREG_DELTA	u	u	u
0x0068 (104)	**CREG_v** Final result from default ray-triangle intersection algorithm. This is calculated as: CREG_H3 / CREG_DELTA	v	v	V
0x0069 (105)	**CREG_H1** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: DotProduct(CREG_Q , CREG_E2)	H1	H1	H1
0x0069 (106)	**CREG_H2** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: DotProduct(CREG_P , CREG_T)	H2	H2	H2
0x006A (107)	**CREG_H3** Partial result used by the default ray-triangle intersection algorithm. This is calculated as: DotProduct(CREG_Q , CREG_DIRECTION)	H3	H3	H3
0x0094 (108)	CREG_PIXEL_PITCH			

0x0095 (109)	**CREG_LAST_COL** the last valid column, simply CREG_RESOLUTIONX - 1	Last colum	Not used	Not used
0x0096 (110)	**CREG_TEXTURE_COLOR** The color of the texture for the current iteration.	Red component	Green component	Blue component
0x0097 (111)	**CREG_PIXEL_2D_POSITION** The current position on the projection plane for the current iteration.	Pixel row	Pixel column	Not used
0x0098 (112)	**CREG_TEXWEIGHT1** The weight used for the first texture color fetch external texture memory for this iteration. A total of 4 weights are used by the bi-linear filtering algorithm.	Texture weight	Texture weigh	Texture weigh
0x0099 (113)	**CREG_TEXWEIGHT2** The weight used for the second texture color fetch external texture memory for this iteration. A total of 4 weights are used by the bi-linear filtering algorithm.	Texture weigh	Texture weigh	Texture weigh
0x009A (114)	**CREG_TEXWEIGHT3** The weight used for the third texture color fetch external texture memory for this iteration. A total of 4 weights are used by the bi-linear filtering algorithm.	Texture weigh	Texture weigh	Texture weigh
0x009B (115)	**CREG_TEXWEIGHT4** The weight used for the forth texture color fetch external texture memory for this iteration. A total of 4 weights are used by the bi-linear filtering algorithm.	Texture weigh	Texture weigh	Texture weigh

Table 5: Internal Registers

Output Registers.

Address	Name	X bits 95 - 64	Y bits 63 - 32	Z bits 31 - 0
0x0053 (128)	**OREG_PIXEL_COLOR** The pixel color calculated for the current iteration.	Red Component	Green component	Blue component
0x0054 (129)	**OREG_TEX_COORD1** The first 2 texture address coordinates.	Texture memory location 1	Texture memory location 2	Not used
0x0055 (130)	**OREG_TEX_COORD2** [8] The last 2 texture address coordinates.	Texture memory location 3	Texture memory location 4	Not used
0x0056 (131)	**OREG_ADDR_O** Generic output address this gets mapped by IO to the ADR_O output pin.	Memory location	Not used	Not used

Table 6: Output registers

8 Used for bilinear filtering

3. Instruction set

This is full list of Theia's instruction set. Please be aware that arithmetic Operations use fixed point arithmetic unless otherwise specified.

Type I Instructions.

NOP

Operation	NOP: No operation
Destination (16 bits)	n/a
Source register 1 (16 bits)	n/a
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	No operation, the ALU stalls until completion of operation. The macro RT_FALSE is used to set the destination and Immendiate value to zero.
Example	`//No operation` `NOP RT_FALSE`

ADD

Operation	ADD: Addition
Destination (16 bits)	Destination of the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	Second source of the arithmetic operation.
Latency (Clock Cycles)	2
Description	Adds the contents of Source register 1 and source resgister 2 and stores result into destination register. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x + S2_x \\ S1_y + S2_y \\ S1_z + S2_z \end{pmatrix}$$
Example	`//R1 = R1 + R2` `ADD R1 R1 R2`

SUB

Operation	SUB: Substraction
Destination (16 bits)	Destination of the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	Second source of the arithmetic operation.
Latency (Clock Cycles)	2
Description	Substracts the contents of source register 1 to source resgister 2 and stores result into destination register. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x - S2_x \\ S1_y - S2_y \\ S1_z - S2_z \end{pmatrix}$$
Example	`//R1 = R1 - R2` `SUB R1 R1 R2`

DIV

Operation	DIV: Division
Destination (16 bits)	Destination of the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	Second source of the arithmetic operation.
Latency (Clock Cycles)	Variable.
Description	Divides the contents of source register 1 from source resgister 2 and stores result into destination register. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} \frac{S1_x}{S2_x} \\ \frac{S1_y}{S2_y} \\ \frac{S1_z}{S2_z} \end{pmatrix}$$
Example	`//R1 = R1 / R2` `DIV R1 R1 R2`

MUL

Operation	MUL: Multiplication
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	Second source of the arithmetic operation.
Latency (Clock Cycles)	2
Description	Multiplies the contents of source register 1 to source resgister 2 and stores result into destination register. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x * S2_x \\ S1_y * S2_y \\ S1_z * S2_z \end{pmatrix}$
Example	`//R1 = R1 - R2` `SUB R1 R1 R2`

MAG

Operation	MAG: Vector Magnitude
Destination (16 bits)	Destination of the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	Variable.
Description	Caculates the vector magnitud of source register 1, ignores source resgister 2 and stores result into destination register. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} \sqrt{SI_x^2 + SI_y^2 + SI_z^2} \\ \sqrt{SI_x^2 + SI_y^2 + SI_z^2} \\ \sqrt{SI_x^2 + SI_y^2 + SI_z^2} \end{pmatrix}$
Example	`//R1 = Magnitude(CREG_UNORMALIZED_DIRECTION )` `MAG R1 CREG_UNORMALIZED_DIRECTION VOID`

COPY

Operation	COPY: Copy
Destination (16 bits)	Destination of the arithmetic operation.
Source register 1 (16 bits)	First source of the arithmetic operation.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Copies contents of Source register 1 into destination register.
Example	`//R1 = CREG_PIXEL_PITCH` `COPY R1 CREG_PIXEL_PITCH VOID`

JGX

Operation	JGX: Jump if greater than X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is greater than Source 2 x component .

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t > GREG_LAST_t`
	`JGX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JGY

Operation	JGY: Jump if greater than Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is greater than Source 2 y component .

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t > GREG_LAST_t`
	`JGY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JGZ

Operation	JGZ: Jump if greater than Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is greater than Source 2 z component .

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t > GREG_LAST_t`
	`JGZ LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLX

Operation	JLX: Jump if less than X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is less than Source 2 x component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t < GREG_LAST_t`
	`JLX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLY

Operation	JLY: Jump if less than Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is less than Source 2 y component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t < GREG_LAST_t`
	`JLY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLZ

Operation	JLX: Jump if less than Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is less than Source 2 z component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t < GREG_LAST_t`
	`JLX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JEQX

Operation	JEQX: Jump if equal X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is equal than Source 2 x component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t = GREG_LAST_t`
	`JEQX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JEQY

Operation	JEQY: Jump if equal Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is equal than Source 2 y component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t = GREG_LAST_t`
	`JEQY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JEQZ

Operation	JEQY: Jump if equal Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is equal than Source 2 z component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t = GREG_LAST_t`
	`JEQZ LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JNEX

Operation	JNEX: Jump if not equal X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is not equal than Source 2 x component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t != GREG_LAST_t`
	`JNEX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JNEY

Operation	JNEX: Jump if not equal Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is not equal than Source 2 y component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t != GREG_LAST_t`
	`JNEY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JNEZ

Operation	JNEZ: Jump if not equal Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is not equal than Source 2 z component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t != GREG_LAST_t`
	`JNEZ LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JGEX

Operation	JGEX: Jump if greater or equal X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is greater or equal than Source 2 x component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t >= GREG_LAST_t`
	`JGEX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JGEY

Operation	JGEY: Jump if greater or equal Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is greater or equal than Source 2 y component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t >= GREG_LAST_t`
	`JGEY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JGEZ

Operation	JGEZ: Jump if greater or equal Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is greater or equal than Source 2 z component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t >= GREG_LAST_t`
	`JGEZ LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLEX

Operation	JLEX: Jump if less or equal X
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 x component is less or equal than Source 2 x component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t <= GREG_LAST_t`
	`JLEX LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLEY

Operation	JLEY: Jump if less or equal Y
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 y component is less or equal than Source 2 y component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t <= GREG_LAST_t`
	`JLEY LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

JLEZ

Operation	JLEZ: Jump if less or equal Z
Immediate value (16 bits)	Immediate value representing the jump destination.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	Second source for arithmetic comparison.
Latency (Clock Cycles)	2
Description	Jumps to the Immediate address if Source 1 z component is less or equal than Source 2 z component.

Example	`//Jumps to LABEL_TCC_EXIT if CREG_t <= GREG_LAST_t`
	`JLEZ LABEL_TCC_EXIT CREG_t CREG_LAST_t`
	…
	`LABEL_TCC_EXIT:`
	`RETURN RT_TRUE`

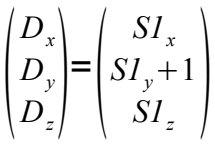
INC

Operation	INC: Increment
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Increments the contents of Source 1 and store into destination. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x + 1 \\ S1_y + 1 \\ S1_z + 1 \end{pmatrix}$
Example	`//R2 = R1 + 1` `INC R2 R1 VOID`

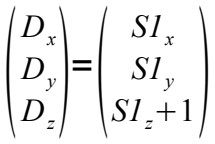
INCX

Operation	INCX: Increment X
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Increments the contents of Source 1 X and store into destination. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x + 1 \\ S1_y \\ S1_z \end{pmatrix}$
Example	`//R2 = R1.x + 1` `INCX R2 R1 VOID`

INCY

Operation	INCY: Increment Y
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Increments the contents of Source 1 Y and store into destination. 
Example	`//R2 = R1.y + 1` `INCY R2 R1 VOID`

INCZ

Operation	INCZ: Increment Z
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Increments the contents of Source 1 Z and store into destination. 
Example	`//R2 = R1.z + 1` `INCZ R2 R1 VOID`

DEC

Operation	DEC: decrement
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	First source for arithmetic comparison.
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	decrements the constents of Source 1 and store into destination. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x - 1 \\ S1_y - 1 \\ S1_z - 1 \end{pmatrix}$
Example	`//R2 = R1 + 1` `INC R2 R1 VOID`

ZERO

Operation	ZERO: Store Zero into register
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	n/a
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Stores zero into the register $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Example	`//R7 = (0, 0, 0)` `ZERO R7 VOID VOID`

CROSS

Operation	CROSS: Cross product (vector product or Gibbs vector product)
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	Source 2 Vector
Latency (Clock Cycles)	2
Description	Calculates the cross product of Source 1 and Source 2 and stores into destination. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \\ S1_y \\ S1_z \end{pmatrix} \times \begin{pmatrix} S2_x \\ S2_y \\ S2_z \end{pmatrix} = \begin{pmatrix} S1_y * S2_z - S1_z * S2_y \\ S1_z * S2_x - S1_x * S2_z \\ S1_x * S2_y - S1_y * S2_x \end{pmatrix}$
Example	`//P = RayDirection Cross E2` `CROSS CREG_P CREG_RAY_DIRECTION CREG_E2`

DOT

Operation	DOT: Dot product (scalar product)
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	Source 2 Vector
Latency (Clock Cycles)	2
Description	Calculates the scalar product of Source 1 and Source 2 and stores into destination. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \\ S1_y \\ S1_z \end{pmatrix} * \begin{pmatrix} S2_x \\ S2_y \\ S2_z \end{pmatrix} = \begin{pmatrix} S1_y * S2_x + S1_y * S2_y + S1_z * S2_z \\ S1_y * S2_x + S1_y * S2_y + S1_z * S2_z \\ S1_y * S2_x + S1_y * S2_y + S1_z * S2_z \end{pmatrix}$
Example	`//H1 = Q dor E2` `DOT CREG_H1 CREG_Q CREG_E2`

MULP

Operation	MULP: Special Multiplication
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Calculates the multiplication of the x and y components of S1 and stores the result back into Dz, leaving the contents of Dx and Dy unchanged. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} D1_x \\ D1_y \\ S1_x * S1_y \end{pmatrix}$
Example	`//TextureWeight.z = R5.x * R5.y` `MULP CREG_TEXWEIGHT1 R5 VOID`

MOD

Operation	MOD: Modulus of the power of 2
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	Source 2 Vector [9]
Latency (Clock Cycles)	2
Description	Calculates the multiplication of the x and y components of S1 and stores the result back into Dz, leaving the contents of Dx and Dy unchanged. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} SI_x \wedge 2^{S2_x} \\ SI_y \wedge 2^{S2_y} \\ SI_x \wedge 2^{S2_z} \end{pmatrix}$
Example	`//R2 = R2 modulo texture_size` `MOD R2 R2 CREG_TEXTURE_SIZE`

FRAC

9 Argument to operation has to be power of 2

Operation	FRAC: Fractional part
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Calculates the fractional part of S1. This is the last n bits of the Fixed point representation with scale n. Stores the result into the dstination register. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} fractional\ part\ of\ S1_x \\ fractional\ part\ of\ S1_y \\ fractional\ part\ of\ S1_z \end{pmatrix}$
Example	`//R4 = fractional part of R7` `FRAC R4 R7 VOID`

INTP

Operation	INTP: Integer part
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Calculates the integer part of S1. This is the first 32-n bits of the Fixed point representation with scale n. Stores the result into the dstination register. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} integer\ part\ of\ S1_x \\ integer\ part\ of\ S1_y \\ integer\ part\ of\ S1_z \end{pmatrix}$
Example	`//R4 = integer part of R7` `INTP R4 R7 VOID`

NEG

Operation	NEG: Sign Change
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Calculates 2's complent of S1. Stores the result into the dstination register. $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} -S1_x \\ -S1_y \\ -S1_z \end{pmatrix}$
Example	`//R5 = -R5` `INTP R5 R5 VOID`

XCHANGEX

Operation	XCHANGEX: Exchange X components
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Exchange X components of S1 for S2 and store into Destination $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S2_x \\ S1_y \\ S1_z \end{pmatrix}$
Example	`//R1 = (R3.x,R2.y,R2,z)` `XCHANGEX R1 R2 R3`

XCHANGEY

Operation	XCHANGEY: Exchange y components
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Exchange y components of S1 for S2 and store into Destination $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \\ S2_y \\ S1_z \end{pmatrix}$
Example	`//R1 = (R2.x,R3.y,R2,z)` `XCHANGEY R1 R2 R3`

XCHANGEZ

Operation	XCHANGEZ: Exchange Z components
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Exchange Z components of S1 for S2 and store into Destination $\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \\ S1_y \\ S2_z \end{pmatrix}$
Example	`//R1 = (R2.x,R2.y,R3,z)` `XCHANGEX R1 R2 R3`

IMUL

Operation	IMUL: Integer multiplication
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	Source 1 Vector
Latency (Clock Cycles)	2
Description	Multiplies S1 and S2 using integer arithmetic instead of fixed point arithmetic. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x * S2_x \\ S1_y * S2_y \\ S2_z * S2_z \end{pmatrix}$$
Example	`//R1 = R2 * R3` `IMUL R1 R2 R3`

UNSCALE

Operation	UNSCALE: Converts a fixed point into an integer
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Shifts right a Fixed point number converting it into a integer. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \gg n \\ S1_y \gg n \\ S2_z \gg n \end{pmatrix} S1_x, S1_y, S1_z \in Q_{32n}$$
Example	`//R1 = Integer( R1 )` `UNSCALE R1 R1 VOID`

RESCALE

Operation	RESCALE: Converts an integer into a fixed point number
Destination (16 bits)	Destination for the arithmetic operation.
Source register 1 (16 bits)	Source 1 Vector
Source register 2 (16 bits)	n/a
Latency (Clock Cycles)	2
Description	Shifts left an integerr converting it into a Fixed point number. $$\begin{pmatrix} D_x \\ D_y \\ D_z \end{pmatrix} = \begin{pmatrix} S1_x \ll n \\ S1_y \ll n \\ S2_z \ll n \end{pmatrix} S1_x, S1_y, S1_z \in I_{32}$$
Example	`//R1 = R2 * R3` `RESCALE R1 R1 VOID`

Type I I Instructions.

RETURN

Operation	RETURN: Returns from subroutine
Destination (16 bits)	n/a
Immediate	Immediate value: RT_TRUE or RT_FALSE
Latency (Clock Cycles)	2
Description	Returns from subroutine. Can return of the following values: RT_TRUE RT_FALSE;

Example	`//Return here`
	`RETRURN RT_TRUE`
	...
	`//or here`
	`RETRURN RT_FALSE`

JMP

Operation	JMP: Jump
Destination (16 bits)	Jump destination address
Immediate	VOID
Latency (Clock Cycles)	2
Description	Jumps to address
Example	`//Just jump` `JMP LABEL_FOO VOID VOID`

SETX

Operation	SETX: Set register's X value
Destination (16 bits)	Destiantion register
Immediate	Value to store
Latency (Clock Cycles)	2
Description	Store immediate value in Destination X part.
Example	`//R1.x = 0xCAFE` `SETX R1 0xCAFE`

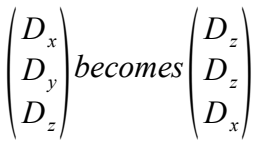
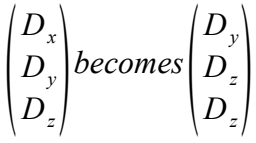
SETY

Operation	SETY: Set register's Y value
Destination (16 bits)	Destiantion register
Immediate	Value to store
Latency (Clock Cycles)	2
Description	Store immediate value in Destination Y part.
Example	`//R1.y = 0xCAFE` `SETY R1 0xCAFE`

SETZ

Operation	SETZ: Set register's Z value
Destination (16 bits)	Destiantion register
Immediate	Value to store
Latency (Clock Cycles)	2
Description	Store immediate value in Destination Z part.
Example	`//R1.z = 0xCAFE` `SETZ R1 0xCAFE`

SWIZZLE3D

Operation	SWIZZLE3D: Re-arrange the X,Y and Z components of destiantion in many different possible combinations.
Destination (16 bits)	Destiantion register
Immediate	Swizzle operation. Can be one of: SWIZZLE_XXX SWIZZLE_YYY SWIZZLE_ZZZ SWIZZLE_XYY SWIZZLE_XXY SWIZZLE_XZZ SWIZZLE_XXZ SWIZZLE_YXX SWIZZLE_YYX SWIZZLE_YZZ SWIZZLE_YYZ SWIZZLE_ZXX SWIZZLE_ZZX SWIZZLE_ZYY SWIZZLE_ZZY SWIZZLE_XZX SWIZZLE_XYX SWIZZLE_YXY SWIZZLE_YZY SWIZZLE_ZXZ SWIZZLE_ZYZ SWIZZLE_YXZ
Latency (Clock Cycles)	2
Description	Swizzle is one of the most handy operations. It lets re-arrage the components of the vector. For example SWIZZLE_XXX will have the following result:  On the other hand, SWIZZLE_ZZX will have the following result:  SWIZZLE_YZZ does this:  you get the idea...

Example	`//R6.x = R6.x`
	`//R6.y = R6.x`
	`//R6.z = R6.x`