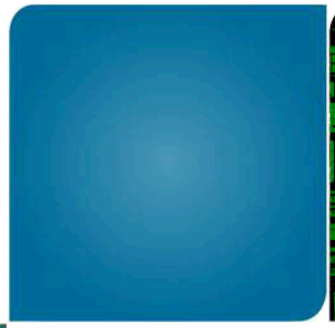
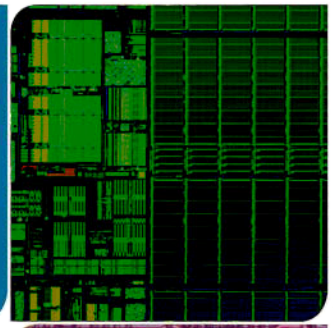
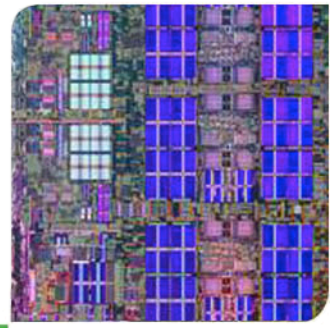
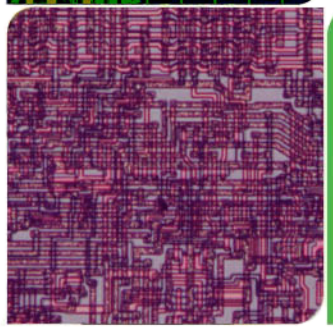
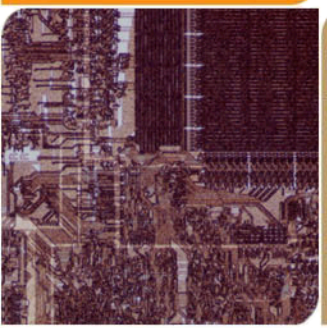
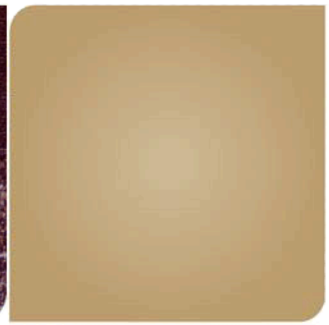







CS3220 LG ISA
(ISA 1.1)

Spring 2013

Georgia Tech

comparch

ISA Format

Note 1: Upper 8 bits are dedicated to Opcode.
Note 2: Lower 24 bits have different format depending on Opcode

Use **2's complement number for Immediate field.*

Opcode Format

Opcode[7:6]: *Category*

0 0 Arithmetic/Memory operations
0 1 GPU operation
1 0 GPU setup property
1 1 Control flow

Opcode[2:0] *Data format* other than LD/ST instructions

100: Floating-point, Register
101: Floating-point, Immediate
000: Integer, Register
001: Integer, Immediate
010: Vector
111: Vector, Immediate (Floating-point)

Opcode[2:0] *Data format* for LD/ST instructions

001: Byte (8-bit)
010: Word (16-bit)

Opcode Format (Cont'd)

Opcode [5:3]

Opcode[7:6]	000	001	010	011	100	101	110	111
00	add+	and+	mov+	cmp+	compmov	ld+	st	
01	setvertex	setcolor	rotate	translate	scale			
10	pushmatrix	popmatrix	beginprimitive	endprimitive	loadidentity		flush	draw
11				br	jmp		jsr	jsrr

Registers

- Scalar Registers (16bits)
  - R0 ~ R6 : 7 integer registers.
  - R8 ~ R14 : 7 floating-point registers.
  - R15 (Program Counter or PC)
    - Indicates where a computer is in its program sequence.
  - R7 (Link register or LR)
    - Hold the address to return to when a function call completes.
- Vector Registers (64bits = 16bits * 4)
  - There are 64 vector registers.
  - Each vector register holds 4 elements (Scalar Registers).
  - All vectors store only floating-point numbers.
  - Use *idx* field to indicate an element.

Vector Register			
elmt 3	elmt 2	elmt 1	elmt 0

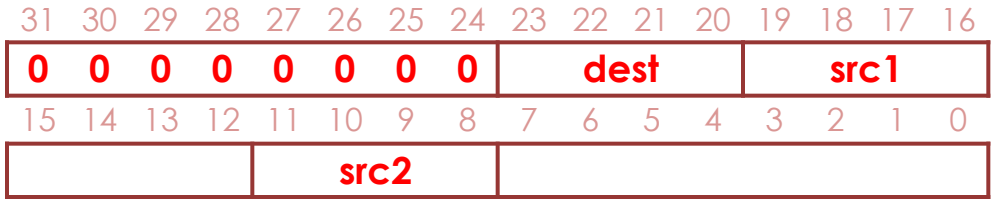
Registers (Cont'd)

- Condition Code (CC) Register
  - Bit 0: Negative (*N*)
  - Bit 1: Zero (*Z*)
  - Bit 2: Positive (Greater than zero) (*P*)
- Instructions that have (+) symbol set/clear conditional code bits
  - E.g., ADD, AND, MOV, CMP, LD
  - Let's say this *Set*CC operation.
- The condition code register is set based on whether the value produced is negative, zero or positive.

Note: Please keep in mind that in our course vector instructions are used only for graphics operations. Hence we don't have to set Condition Code Register for those vector instructions.

ADD.D instruction

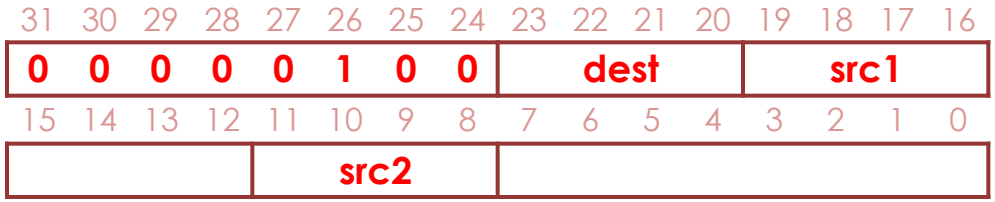
- Assembly Formats
  - add.d dest src1 src2
- Encoded to



- Operation
  - dest <- src1 + src2
  - SetCC
- Description
  - Integer Addition
- Examples
  - add.d r0 r1 r2

ADD.F instruction

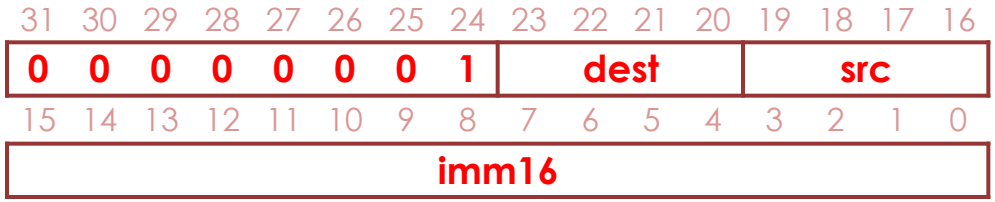
- Assembly Formats
  - add.f dest src1 src2
- Encoded to



- Operation
  - dest <- src1 + src2
  - SetCC
- Description
  - Floating-point Addition
- Examples
  - add.f r10 r8 r9

ADDI.D instruction

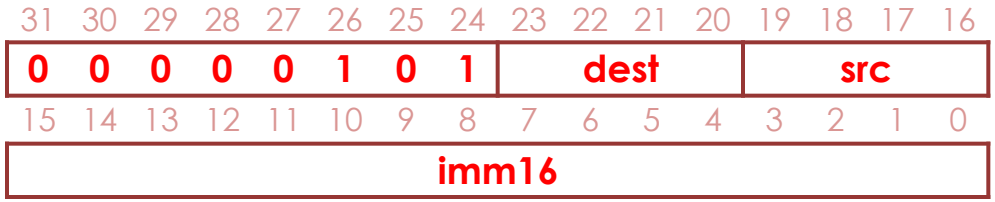
- Assembly Formats
  - addi.d dest src imm16
- Encoded to



- Operation
  - dest <- src + imm16
  - SetCC
- Description
  - Integer Addition
- Examples
  - addi.d r0 r1 1024

ADDI.F instruction

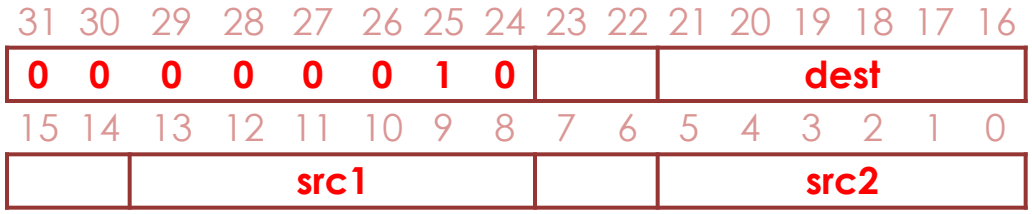
- Assembly Formats
  - addi.f dest src imm16
- Encoded to



- Operation
  - dest <- src + imm16
  - SetCC
- Description
  - Floating-point Addition
- Examples
  - addi.f r7 r8 1.50f

VADD instruction

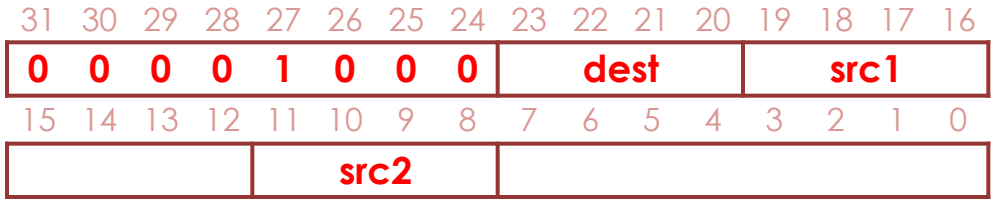
- Assembly Formats
  - vadd dest src1 src2
- Encoded to



- Operation
  - dest[0] <- src1[0] + src2[0]
  - dest[1] <- src1[1] + src2[1]
  - dest[2] <- src1[2] + src2[2]
  - dest[3] <- src1[3] + src2[3]
- Examples
  - vadd v0 v1 v2

AND.D instruction

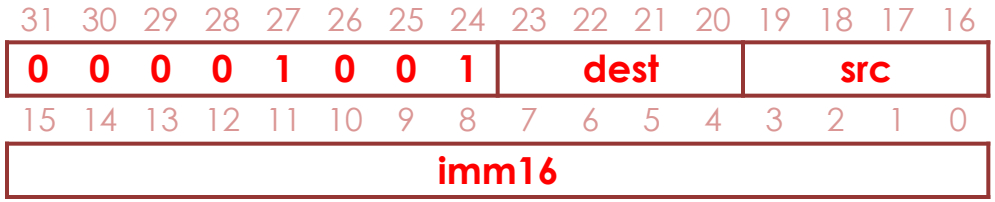
- Assembly Formats
  - and.d dest src1 src2
- Encoded to



- Operation
  - dest <- src1 & src2
  - SetCC
- Description
  - Bitwise AND operation
- Examples
  - and.d r0 r1 r2

ANDI.D instruction

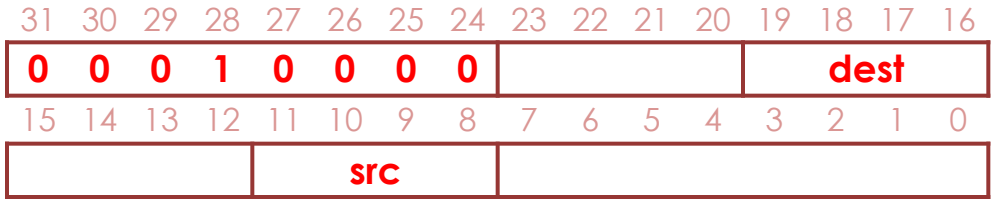
- Assembly Formats
  - andi.d dest src imm16
- Encoded to



- Operation
  - dest <- src & imm16
  - SetCC
- Description
  - Bitwise AND operation with immediate value
- Examples
  - andi.d r0 r1 1024

MOV instruction

- Assembly Formats
  - mov dest src
- Encoded to



- Operation
  - dest <- src
  - SetCC
- Description
  - Move Instruction
- Examples
  - mov r0 r1

MOVI.D instruction

- Assembly Formats
  - movi.d dest imm16
- Encoded to

- Operation
  - dest <- imm16
  - SetCC
- Description
  - Move Instruction for Integer Registers
- Examples
  - movi.d r1 1000

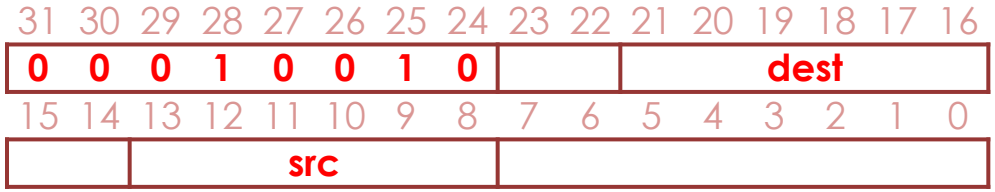
MOVI.F instruction

- Assembly Formats
  - movi.f dest imm16
- Encoded to

- Operation
  - dest <- imm16
  - SetCC
- Description
  - Move Instruction for Floating-point Registers
- Examples
  - movi.f r9 1.5f

VMOV instruction

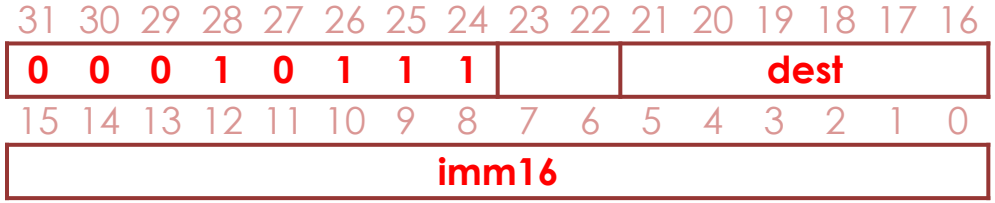
- Assembly Formats
  - vmov dest src
- Encoded to



- Operation
  - dest <- src
- Description
  - Move Instruction for Vector Registers
- Examples
  - vmov v1 v2

VMOVI instruction

- Assembly Formats
  - vmovi dest imm16
- Encoded to



- Operation
  - dest[0], dest[1], dest[2], dest[3] <- imm16
- Description
  - Move Instruction for Vector Registers (Floating-point).
  - *imm16* will be copied to all four elements of the *dest* vector register
- Examples
  - vmovi v1 1000.f

CMP instruction

- Assembly Formats
  - cmp src1 src2
- Encoded to

- Operation
  - if (src1 == src2) CC = Z;
  - else if (src1 < src2) CC = N;
  - else /* src1 > src2 */ CC = P;
- Description
- Examples
  - cmp r0 r1

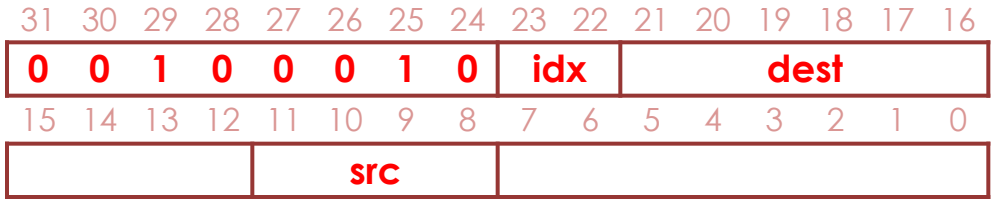
CMPI instruction

- Assembly Formats
  - cmpi src imm16
- Encoded to

- Operation
  - if (src == imm16) CC = Z;
  - else if (src < imm16) CC = N;
  - else /* src > imm16 */ CC = P;
- Description
- Examples
  - cmpi r0 10

VCOMPMOV instruction

- Assembly Formats
  - vcompmov dest idx src
- Encoded to



- Operation
  - dest[idx] <- src
  - *Note: dest - vector register, src - scalar register*
- Description
  - Move Instruction for Vector Register's element.
- Examples
  - vcompmov v1 0 r1

Georgia Tech comparch

VCOMPMOVI instruction

- Assembly Formats
  - vcompmovi dest idx imm16
- Encoded to

- Operation
  - dest[idx] <- imm16
- Description
  - Move Instruction for Vector Register's element (Floating-point).
- Examples
  - vcompmovi v1 0 1.5f

LDB instruction

- Assembly Formats
  - ldb dest base offset
- Encoded to

- Operation
  - dest <- mem[base+offset]
  - SetCC
- Description
  - Load a byte (8 bits) from the memory location
- Examples
  - ldb r1 r2 1000

LDW instruction

- Assembly Formats
  - ldw dest, base, offset
- Encoded to

- Operation
  - dest <- mem[base+offset+1: base+offset]
  - SetCC
- Description
  - Load a word (16 bits) from the memory location
- Examples
  - ldw r1, r2, 1000

STB instruction

- Assembly Formats
  - stb src base offset
- Encoded to

- Operation
  - mem[base+offset] <- src
- Description
  - Store a byte (8 bits) to the memory location
- Examples
  - stb r1 r2 1000

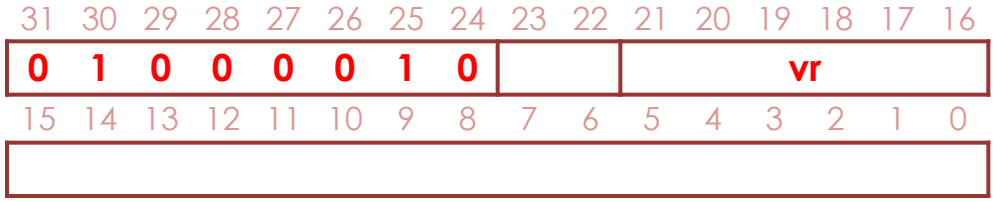
STW instruction

- Assembly Formats
  - stw src base offset
- Encoded to

- Operation
  - mem[base+offset+1: base+offset] <- src
- Description
  - Store a word (16 bits) to the memory location
- Examples
  - stw r1 r2 1000

SETVERTEX instruction

- Assembly Formats
  - setvertex vr
- Encoded to

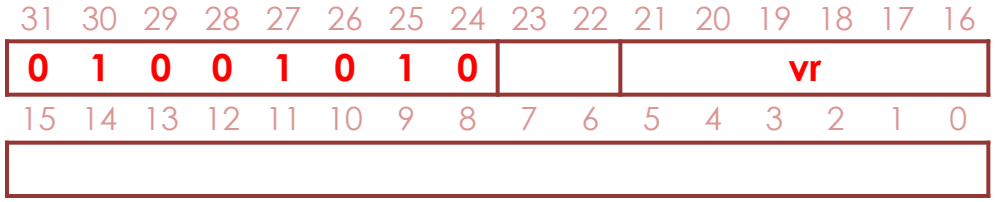


- Operation
  - The value of vr[1]: X coordinate
  - The value of vr[2]: Y coordinate
  - The value of vr[3]: Z coordinate
  - Ignore the value of vr[0]
- Description
  - Set the current vertex using *vr* register
- Examples
  - setvertex v1

GPU-ISA: Becomes NOP for the phase-I

SETCOLOR instruction

- Assembly Formats
  - setcolor vr
- Encoded to

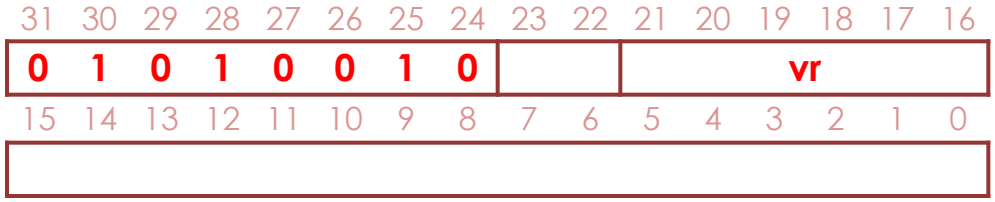


- Operation
  - The value of vr[0]: R component
  - The value of vr[1]: G component
  - The value of vr[2]: B component
  - Ignore the value of vr[3]
- Description
  - Set the color of the current vertex using *vr* register
- Examples
  - setcolor v1

GPU-ISA: Becomes NOP for the phase-I

ROTATE instruction

- Assembly Formats
  - rotate vr
- Encoded to

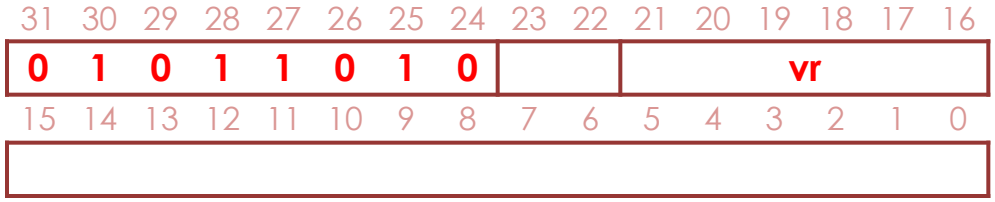


- Operation
  - The value of vr[0]: Angle
  - The value of vr[3]: Z-coordinate
  - Ignore the value of vr[1] & vr[2]
- Description
  - Rotate the current matrix using *vr* register
- Examples
  - rotate v1

GPU-ISA: Becomes NOP for the phase-I

TRANSLATE instruction

- Assembly Formats
  - translate vr
- Encoded to



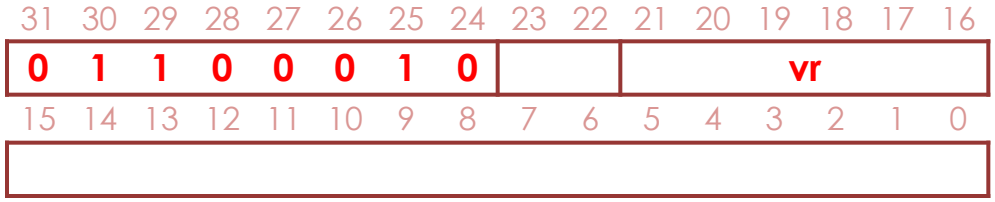
GPU-ISA: Becomes NOP for the phase-I

- Operation
  - The value of vr[1]: distance to move on x-axis
  - The value of vr[2]: distance to move on y-axis
  - Ignore the values of vr[0] & vr[3]
- Description
  - Translate the current matrix using *vr* register
- Examples
  - translate vr1

SCALE instruction

- Assembly Formats
  - scale vr
- Encoded to

GPU-ISA: Becomes NOP for the phase-I



- Operation
  - The value of vr[1]: scale value on x-axis
  - The value of vr[2]: scale value on y-axis
  - Ignore the values of vr[0] & vr[3]
- Description
  - Scale the current matrix using *vr* register
- Examples
  - scale v1

PUSHMATRIX instruction

- Assembly Formats
  - pushmatrix
- Encoded to

- Operation
- Description
  - Push the current matrix into the stack.
- Examples
  - pushmatrix

GPU-ISA: Becomes NOP for the phase-I

POPMATRIX instruction

- Assembly Formats
  - popmatrix
- Encoded to

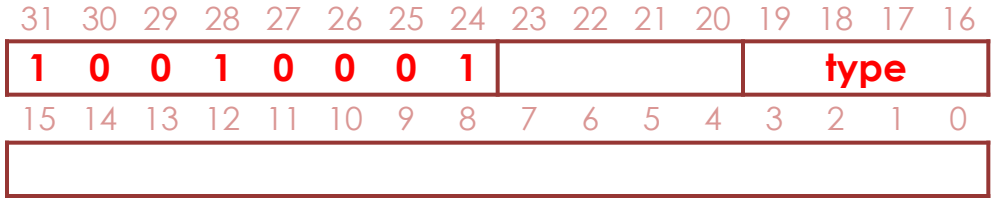
- Operation
- Description
  - Pop a matrix from the stack to the current matrix.
- Examples
  - popmatrix

GPU-ISA: Becomes NOP for the phase-I

BEGINPRIMITIVE instruction

- Assembly Formats
  - beginprimitive type
- Encoded to

GPU-ISA: Becomes NOP for the phase-I



- Pre-defined Types
  - Points: 0, Lines: 1, Line strip: 2, Line loop: 3, Triangles: 4, Triangle strip: 5, Triangle fan: 6, Quads: 7, Quad strip: 8, Polygon: 9
  - Refer to http://www.jmp.com/support/help/Graphics_Primitives.shtml
- Description
  - Delimit the vertices that define a primitive or a group of primitives.
- Examples
  - beginprimitive 4 -> Begin describing "triangle" primitive

ENDPRIMITIVE instruction

- Assembly Formats
  - endprimitive
- Encoded to

GPU-ISA: Becomes NOP for the phase-I

- Operation
- Description
  - Delimit the vertices that define a primitive or a group of primitives.
- Examples
  - endprimitive

LOADIDENTITY instruction

- Assembly Formats
  - loadidentity
- Encoded to

- Operation
- Description
  - Replace the current matrix with the identity matrix.
- Examples
  - loadidentity

GPU-ISA: Becomes NOP for the phase-I

FLUSH instruction

- Assembly Formats
  - flush
- Encoded to

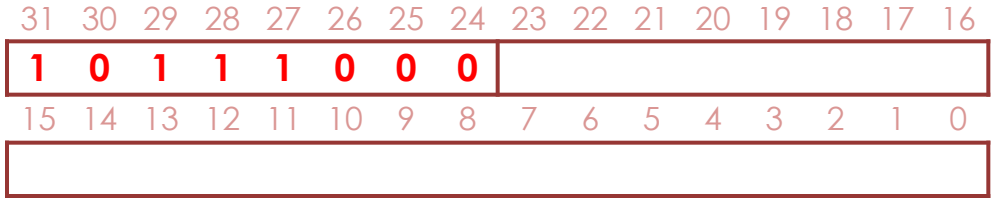
GPU-ISA: Becomes NOP for the phase-I

- Operation
- Description
  - Flush (*Empty*) the contents of the frame buffer.
- Examples
  - flush

Georgia Tech comparch

DRAW instruction

- Assembly Formats
  - draw
- Encoded to



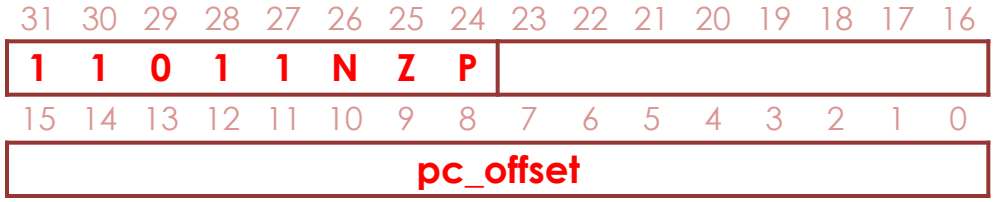
- Operation

GPU-ISA: Becomes NOP for the phase-I

- Description
  - Draw the contents of the frame buffer on a screen (if available).
  - Also, this instruction indicates the end of frame.
- Examples
  - draw

BR (Conditional branch) instruction

- Assembly Formats
  - brnzp pc_offset
- Encoded to



- Operation
  - If (CC = NZP)
    Next PC = PC + pc_offset << 2;
    Else
    Next PC = PC;
- Description
  - Conditional jump to the target address which is made by adding offset to the PC value.
- Examples
  - brnz 8

Note: In multi-stage pipeline architecture, once an instruction is fetched, PC value is updated even before the instruction is executed. Hence, PC value of the RHS of this operation is updated, newer value. Please keep in mind that pc_offset value is a distance not from the older PC value but from the newer PC value.

JMP (Unconditional branch) instruction

- Assembly Formats
  - jmp baseR
- Encoded to

- Operation
  - Next PC = baseR;
- Description
  - Unconditional jump to the contents of the *baseR* register.
- Examples
  - jmp r1

RET (Return from Subroutine) instruction

- Assembly Formats
  - ret
- Encoded to

- Operation
  - Next PC = R7;
- Description
  - The RET instruction is a special case of the JMP instruction. The PC is loaded with the value in R7. This causes a return from a previous JSR instruction.
- Examples
  - ret

JSR (Jump to Subroutine) in

- Assembly Formats
  - jsr pc_offset
- Encoded to

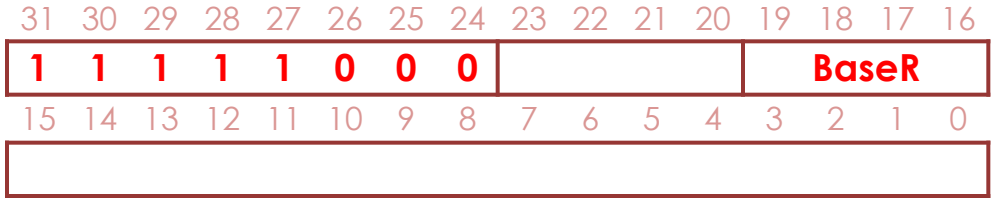
- Operation
  - R7 = PC;
  - Next PC = PC + pc_offset << 2;
- Description
  - First, the incremented PC is saved in R7. This is the linkage back to the calling routine. Then, the PC is loaded with the address of the first instruction of the subroutine, causing an unconditional jump to that address. The address of the subroutine is computed by adding offset to current PC.
- Examples
  - jsr 6

Note: In multi-stage pipeline architecture, once an instruction is fetched, PC value is updated even before the instruction is executed. Hence, PC value of the RHS of this operation is updated, newer value. Please keep in mind that pc_offset value is a distance not from the older PC value but from the newer PC value.

Georgia Tech comparch

JSRR (Jump to Subroutine) i

- Assembly Formats
  - Jsrr baseR
- Encoded to



- Operation
  - R7 = PC;
  - Next PC = baseR;
- Description
  - First, the incremented PC is saved in R7. This is the linkage back to the calling routine. Then, the PC is loaded with the address of the first instruction of the subroutine, causing an unconditional jump to that address. The address of the subroutine is obtained from the *baseR* register.
- Examples
  - jsrr r1

Note: In multi-stage pipeline architecture, once an instruction is fetched, PC value is updated even before the instruction is executed. Hence, PC value of the RHS of this operation is updated, newer value. Please keep in mind that pc_offset value is a distance not from the older PC value but from the newer PC value.

Georgia Tech comparch