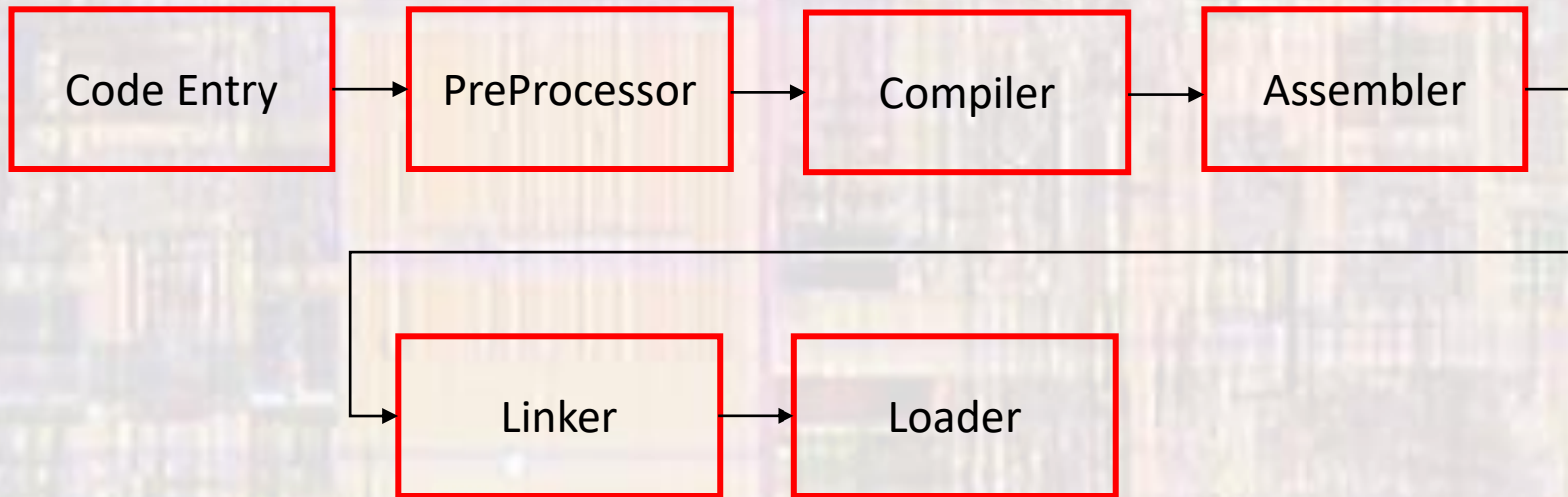


# Processors - Basics

Last modified 4/20/20

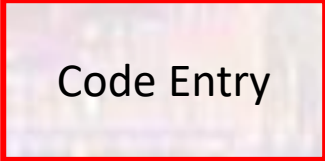
# Program Elements

- Tool Chain



# Program Elements

- Tool Chain
  - CodeEntry
    - filename.c
    - Text editor
    - Integrated Development Environment
      - Code Composer
      - Eclipse



Code Entry

# Program Elements

- Tool Chain

PreProcessor

- Preprocessor

- Deals with any commands starting with #
- Tells the tool chain to include additional libraries
- Replaces any “defines” throughout the code
- Expands macros throughout the code
- Manages any conditional defines



# Program Elements

- Tool Chain

Compiler

- Compiler

- Converts c-code to assembly language
- Assembly language
  - Architecture specific programming language
  - Direct access to specific registers, commands, memory

```
ldi R2, 5;           // load register R2 with the value 5
sts R2, 0x0200;      // copy the value in R2 to memory location 0x200
add R2, R1;          // add the values of R2 and R1 and store in R2
```

# Program Elements

- Tool Chain

Assembler

- Assembler

- Converts assembly language to machine language
- Result is an object file (file.o)

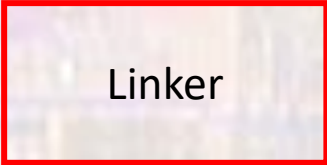
- Machine language

- Part specific programming language
- Binary representation that the processor understands

```
1001 1000 1010 1101      // load register R2 with the value 5
1100 1011 1001 1100      // copy R2 to memory location 0x200
1100 1010 1100 0011      // add R2, R1 and store in R2
```

# Program Elements

- Tool Chain



Linker

- Linker

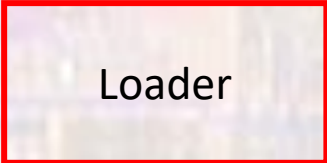
- Combines the machine language code from your program with all included libraries
- Configures all the code in memory
  - Aligns code segments
  - Makes connections where necessary (function calls)
  - Assigns variables spots in memory
- Creates an executable file - file.out (file.exe for windows systems)

# Program Elements

- Tool Chain

- Loader (programmer)

- Creates whatever environment is necessary on the executing machine
    - Loads the executable program
    - Starts the program

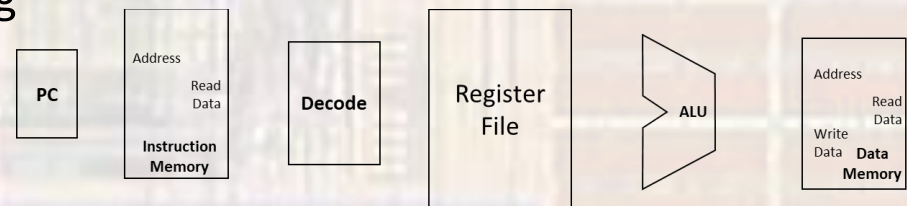


Loader



# Simple Data Path

- Simplified Processor Structure
  - Program Counter
    - Holds the memory address for the next instruction
  - Program Memory
    - Holds the program instructions
  - Decoder
    - Converts instructions (machine code) into control signals
  - Register File
    - Local working memory (for ALU)
  - ALU
    - Arithmetic Logic Unit
    - Does “calculations”
  - Data Memory
    - Holds data for future processing



# Simple Data Path

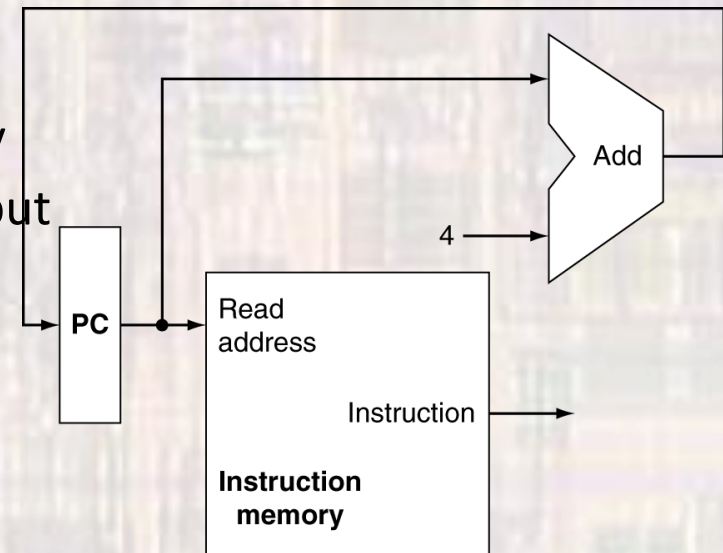
- 5 Stages of Instruction Execution
  - Fetch (IF)
  - Decode / Register Access (ID)
  - Execute (EX)
  - Memory Access (MEM)
  - Write Back (WB)

# Simple Data Path

- Instruction Fetch

- Clock the PC

- New address is provided to the memory
- Memory provides instruction to its output
- Next address is provided to PC input
  - Memory is Byte Addressed
  - Instructions are 4 bytes wide
  - → increment by 4



# Simple Data Path

- Instruction format (MIPS)

## BASIC INSTRUCTION FORMATS

Register - Register

<b>R</b>	opcode	rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5
						0

Immediate / Load/Store

<b>I</b>	opcode	rs	rt	immediate
	31	26 25	21 20	16 15
				0

Jump

<b>J</b>	opcode	address
	31	26 25
		0



# Simple Data Path

- Instruction format (MIPS)

**REGISTER NAME, NUMBER, USE, CALL CONVENTION**

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes

# Simple Data Path

- Instruction format (MIPS)

CORE INSTRUCTION SET				OPCODE / FUNCT (Hex)
NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)		
Add            add	R	$R[rd] = R[rs] + R[rt]$	(1)	0 / 20 <sub>hex</sub>
Add Immediate   addi	I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2)	8 <sub>hex</sub>
Add Imm. Unsigned   addiu	I	$R[rt] = R[rs] + \text{SignExtImm}$	(2)	9 <sub>hex</sub>
Add Unsigned   addu	R	$R[rd] = R[rs] + R[rt]$		0 / 21 <sub>hex</sub>
And            and	R	$R[rd] = R[rs] \& R[rt]$		0 / 24 <sub>hex</sub>
And Immediate   andi	I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3)	0 <sub>hex</sub>
Branch On Equal   beq	I	if( $R[rs] == R[rt]$ ) PC = PC + 4 + BranchAddr	(4)	4 <sub>hex</sub>
Branch On Not Equal   bne	I	if( $R[rs] != R[rt]$ ) PC = PC + 4 + BranchAddr	(4)	5 <sub>hex</sub>

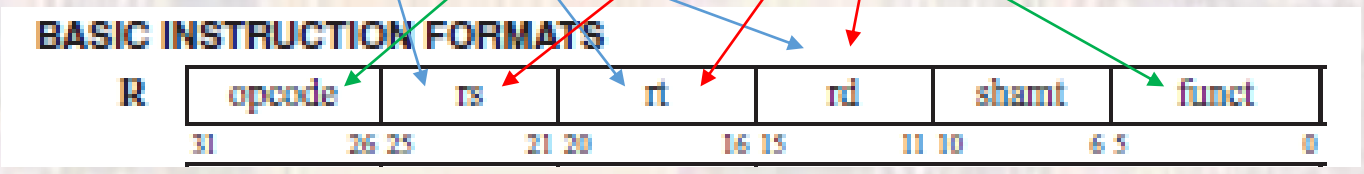
# Simple Data Path

- Instruction format (MIPS)

add \$S0, \$T2, \$S3 -- add register T2 to register S3 and store result in register S0

CORE INSTRUCTION SET			OPCODE / FUNCT (Hex)
NAME, MNEMONIC	FOR MAT	OPERATION (in Verilog)	
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0/20 <sub>hex</sub>

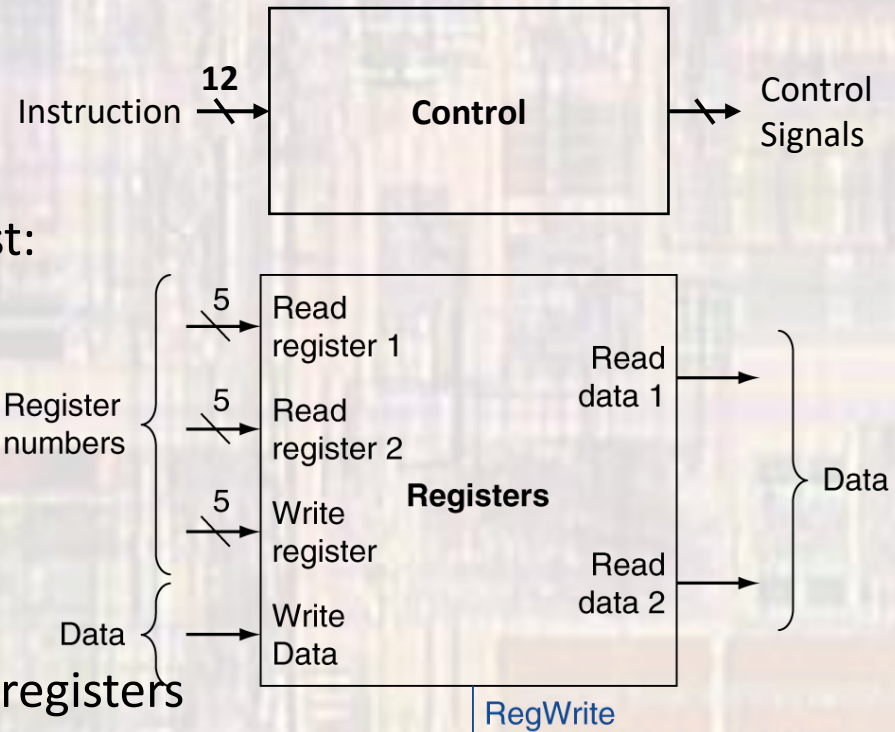
REGISTER NAME, NUMBER, USE, CALL CONVENTION			
NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes



000000 01010 10011 10000 00000 100000  
 0000 0001 0101 0011 1000 0000 0010 0000  
 0x01538020

# Simple Data Path

- Decode / Register Access
  - Decode
    - Use first and last 6 bits of the instruction
  - Register Access
    - R format instructions use at most:
      - 2 source registers and 1 destination register
    - I format instructions use:
      - immediate: 1 src, 1 dest
      - load/store: 1 src or 1 dest
      - branch: 2 src
    - J format instructions do not use registers

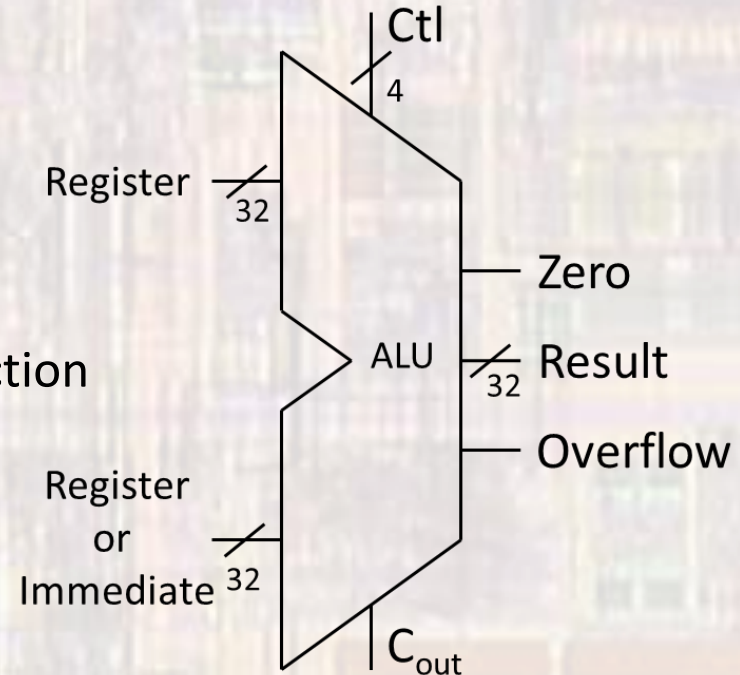




# Simple Data Path

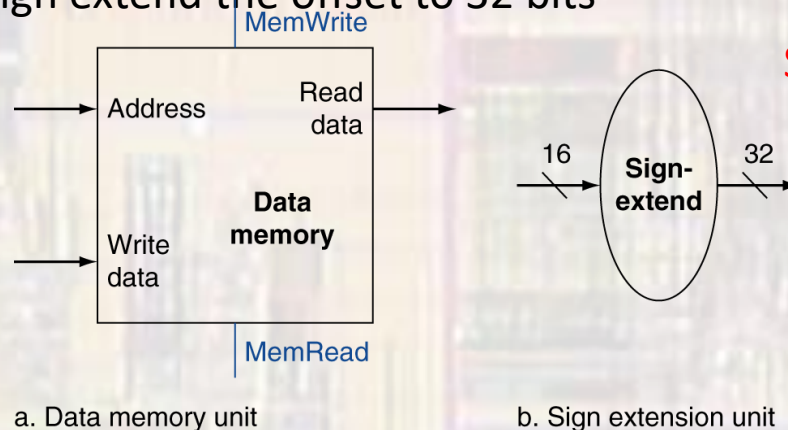
- Execute
  - ALU executes all arithmetic and logical instructions

- Inputs are Registers or Immediates
  - **Immediates** are contained in the instruction  
add \$S0, \$T1, **24**



# Simple Data Path

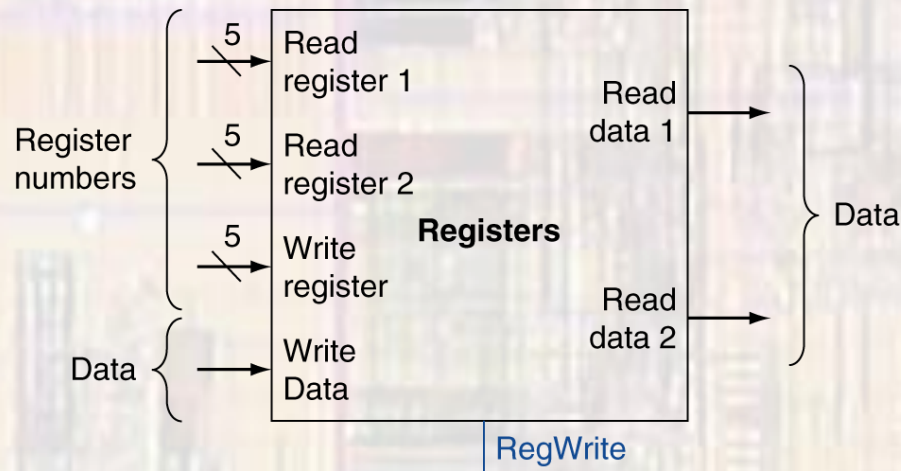
- Memory Access
  - Load / Store Instructions  
`lw $t4,4($t0) # load $t4 from memory location ($t0)+4`
  - Address is calculated by adding the offset to the value in a register
    - Use the ALU to add register value to the offset
    - Since the offset is only 16 bits and is in 2's complement format
      - Must sign extend the offset to 32 bits



Sign extension is trivial – Why?

# Simple Data Path

- Write Back
  - Write results or memory value back to a register
  - Write data comes from ALU (result)  
or
  - Write data comes from data memory

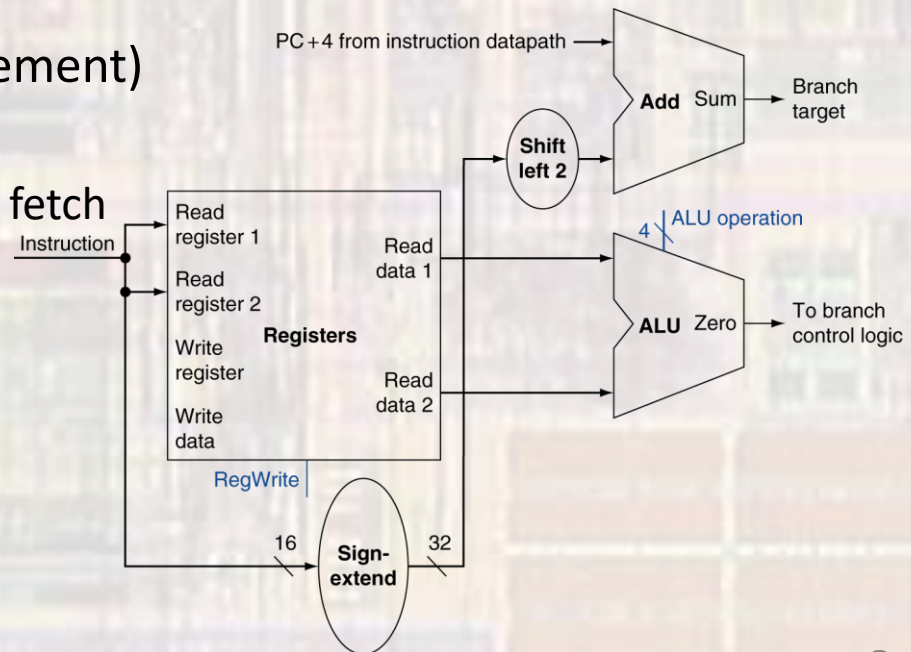


# Simple Data Path

- Missing Pieces – branches
  - Read register operands
  - Compare operands
    - Use ALU, subtract and check Zero output
  - Calculate target address
    - Sign-extend displacement
    - Shift left 2 places (word displacement)
    - Add to PC + 4
      - Already calculated by instruction fetch

shift left 2 is trivial – why?

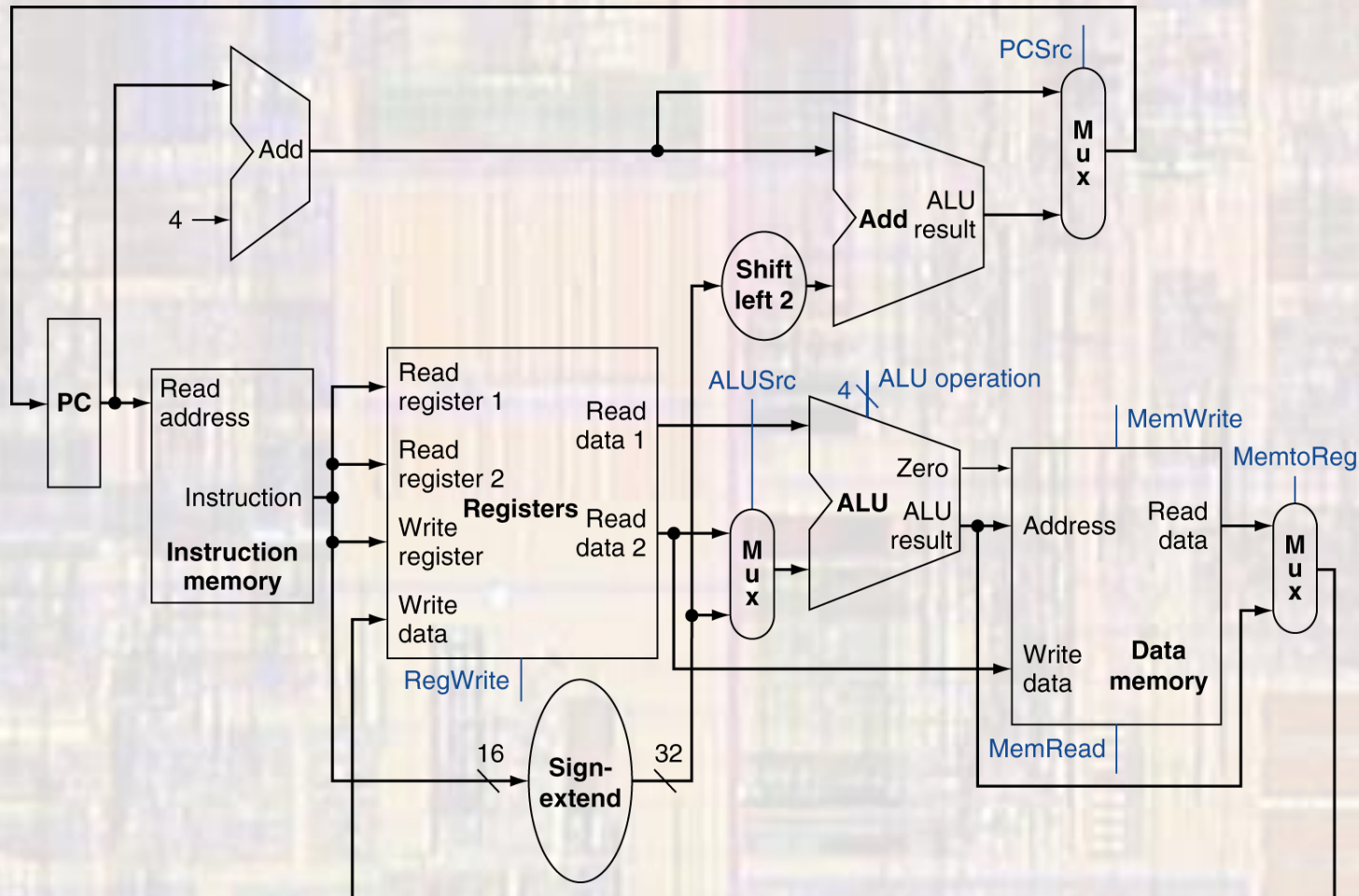
What about the bits that shift off the end?





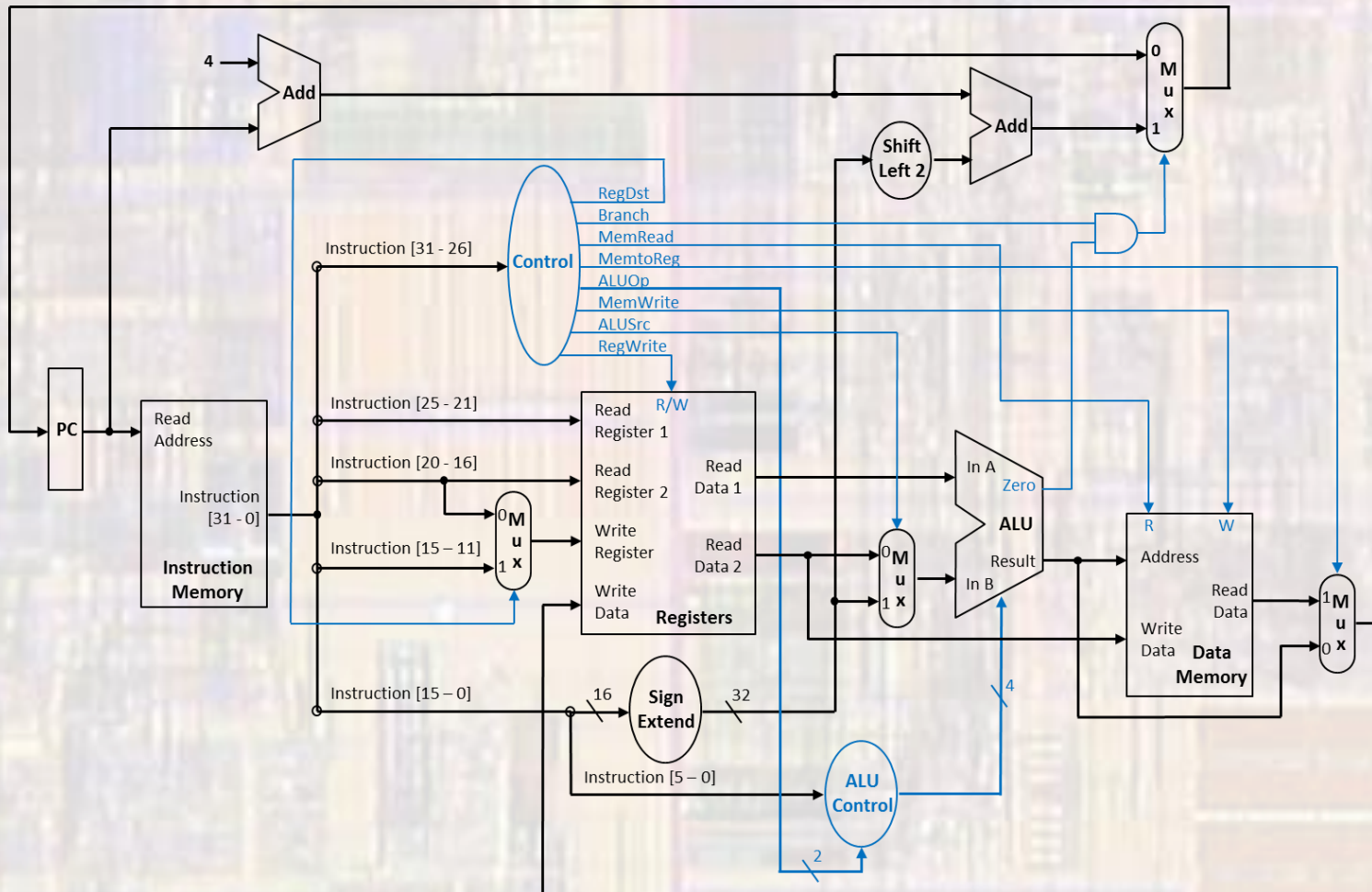
# Simple Data Path

- Full Datapath



# Simple Data Path

- Datapath Control

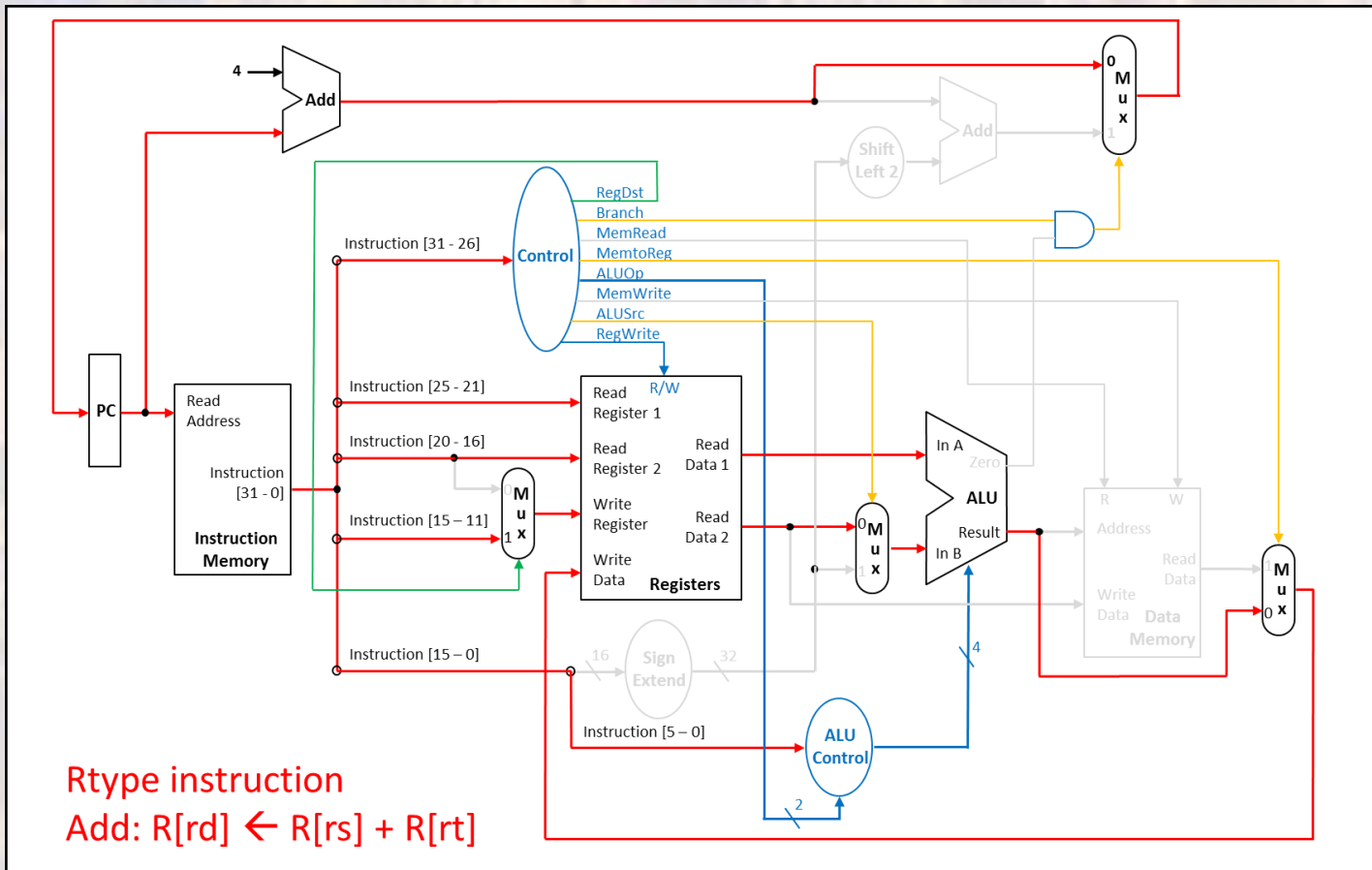


# Simple Data Path

- Datapath Control – Rtype Instruction

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0

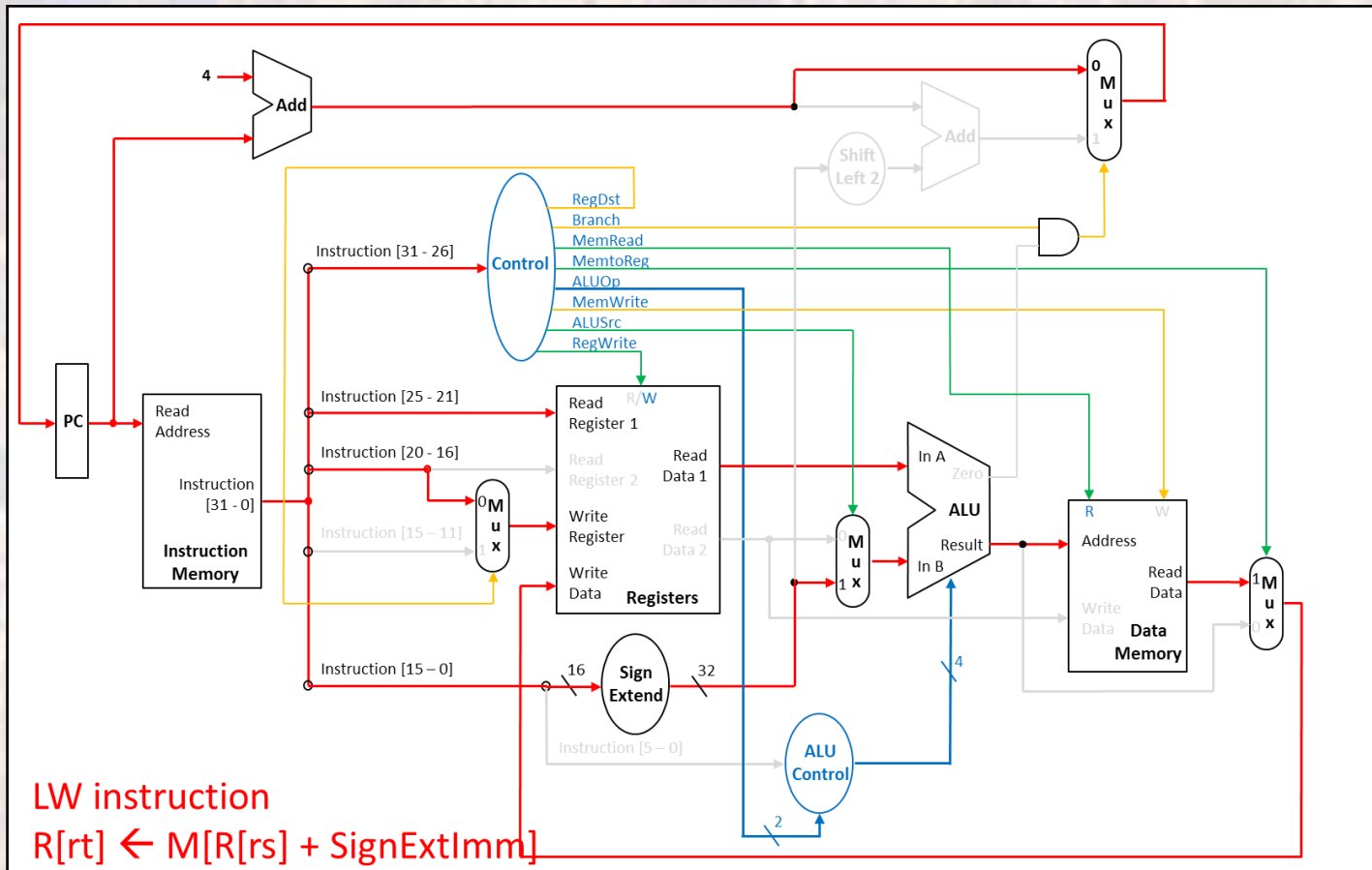


# Simple Data Path

- Datapath Control – LW Instruction

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
LW	0	1	1	1	1	0	0	0	0



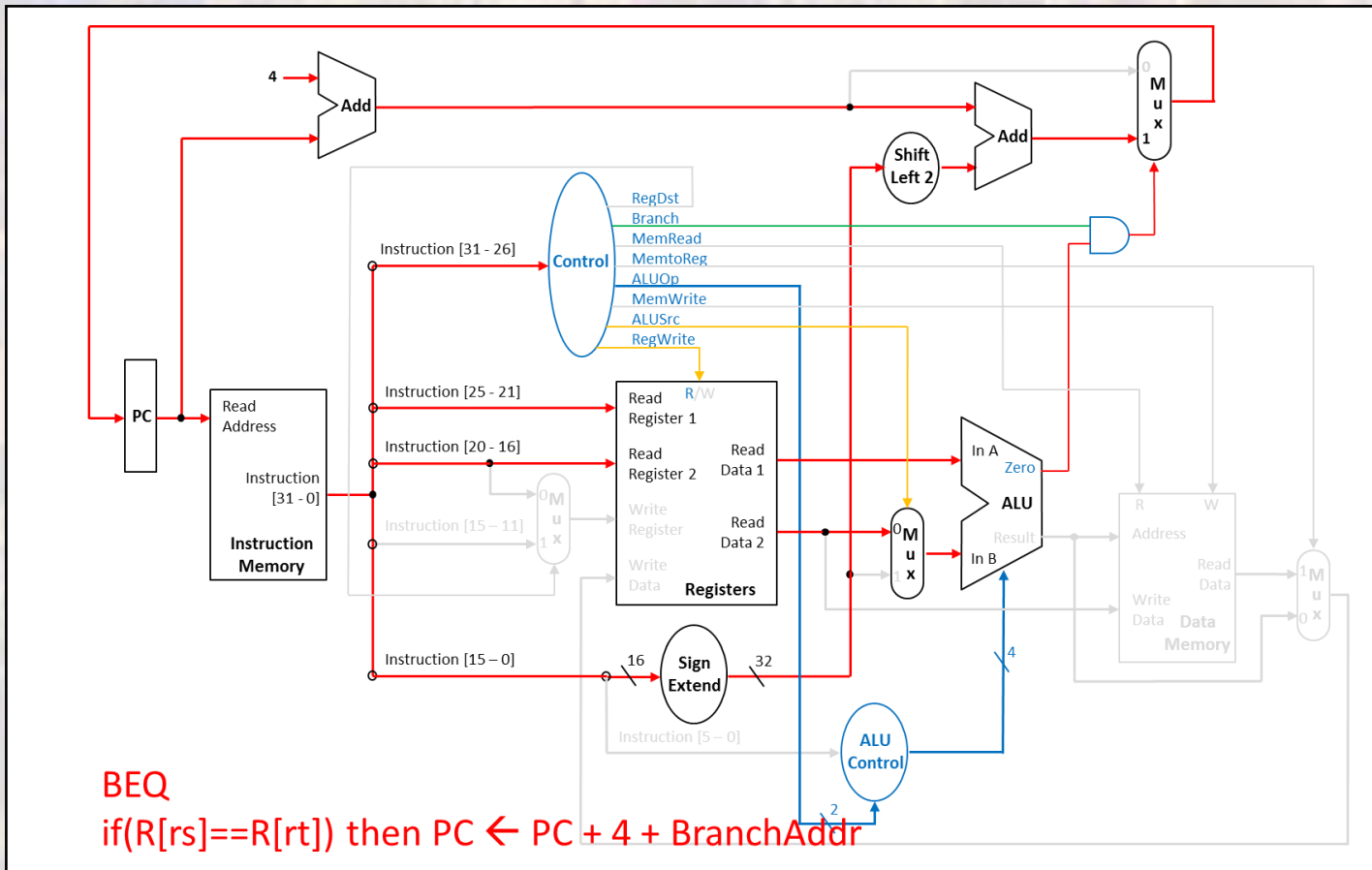


# Simple Data Path

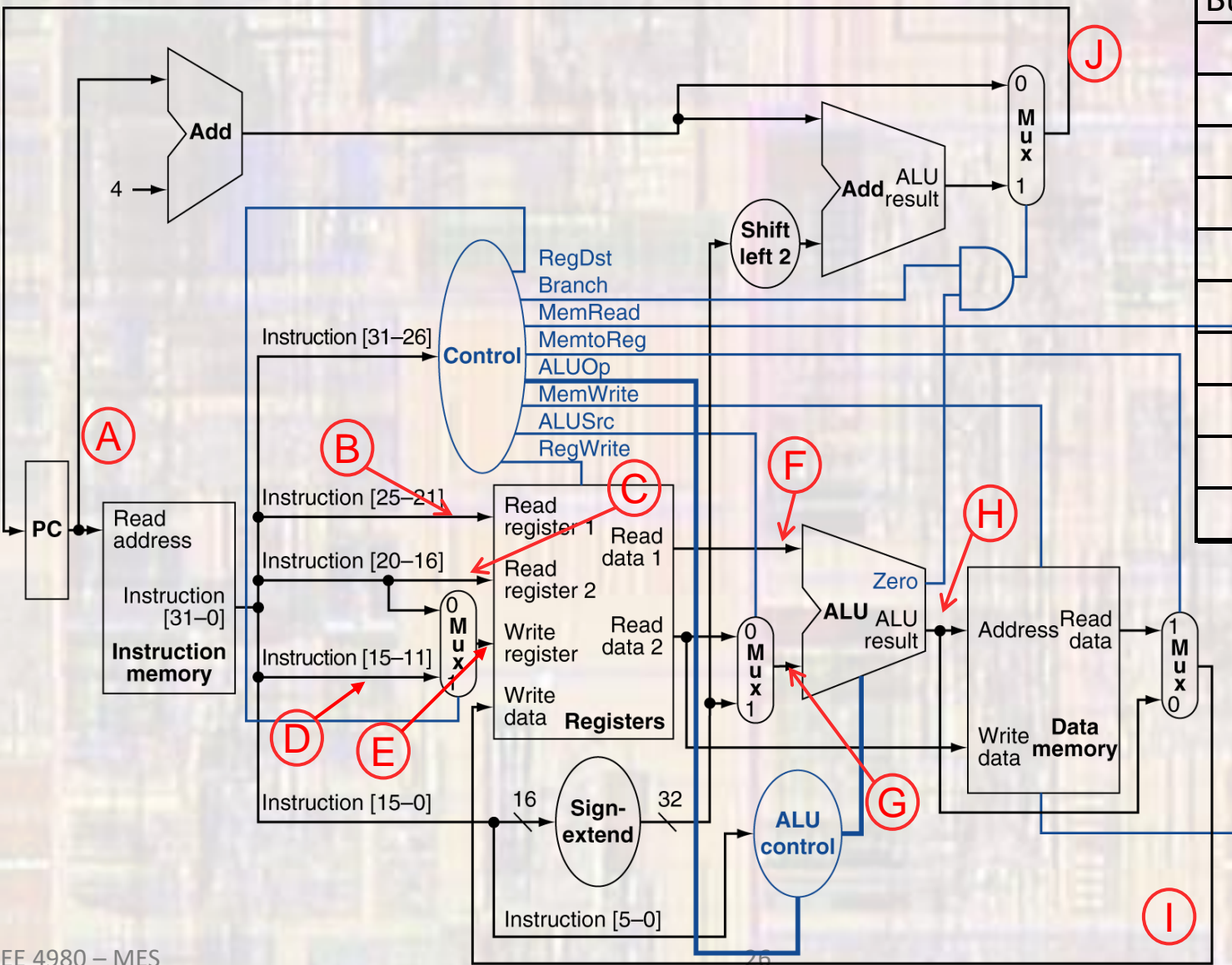
- Datapath Control – BEQ

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
LW	X	0	X	0	0	0	1	0	1

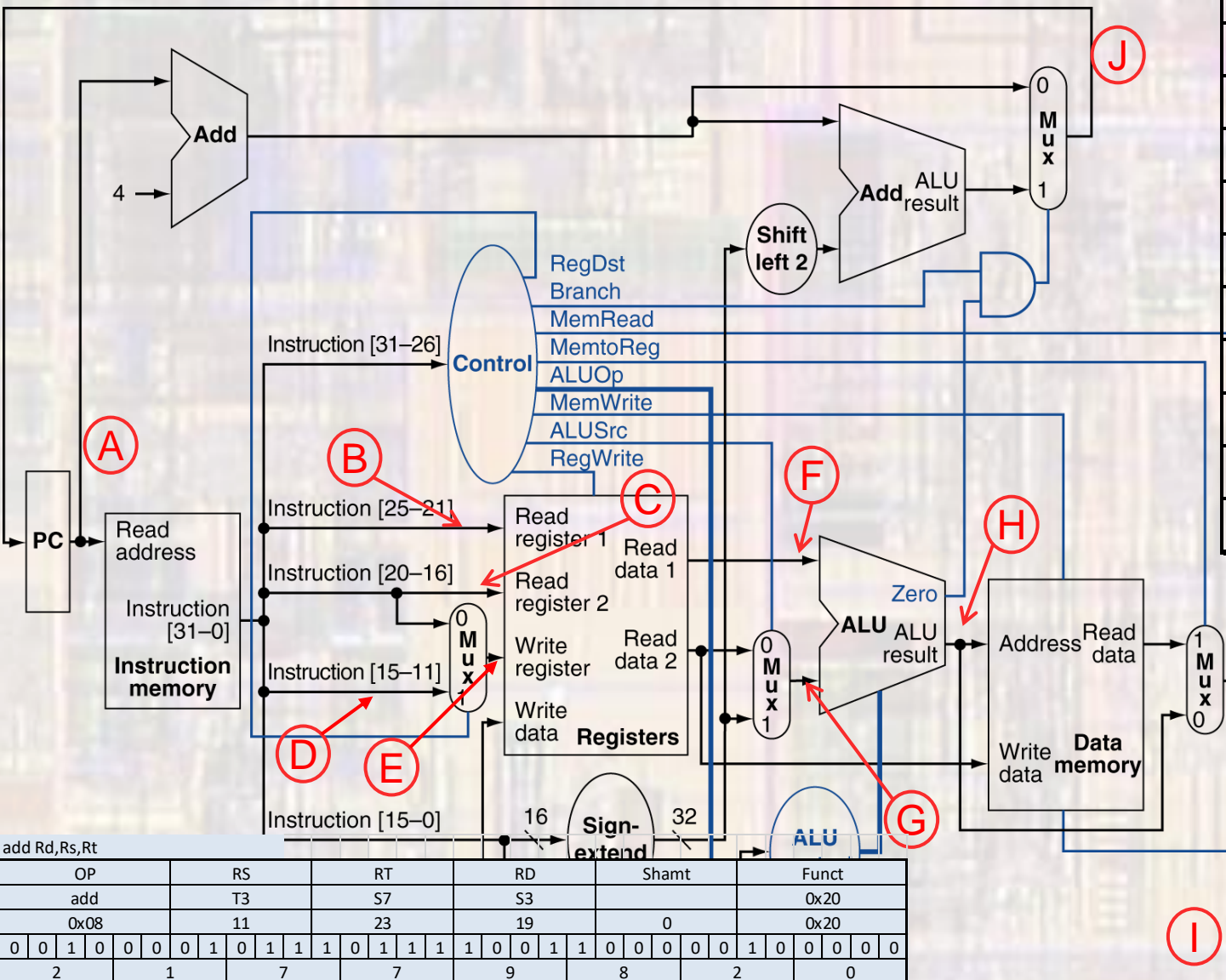


After completion of the instruction “add \$s3,\$t3,\$s7” indicate the value of each data bus.  
 Assume \$t3=0xDCBA, \$s7=0x4321, and the instruction was located at memory location 0x1220,  
 use x for unknown



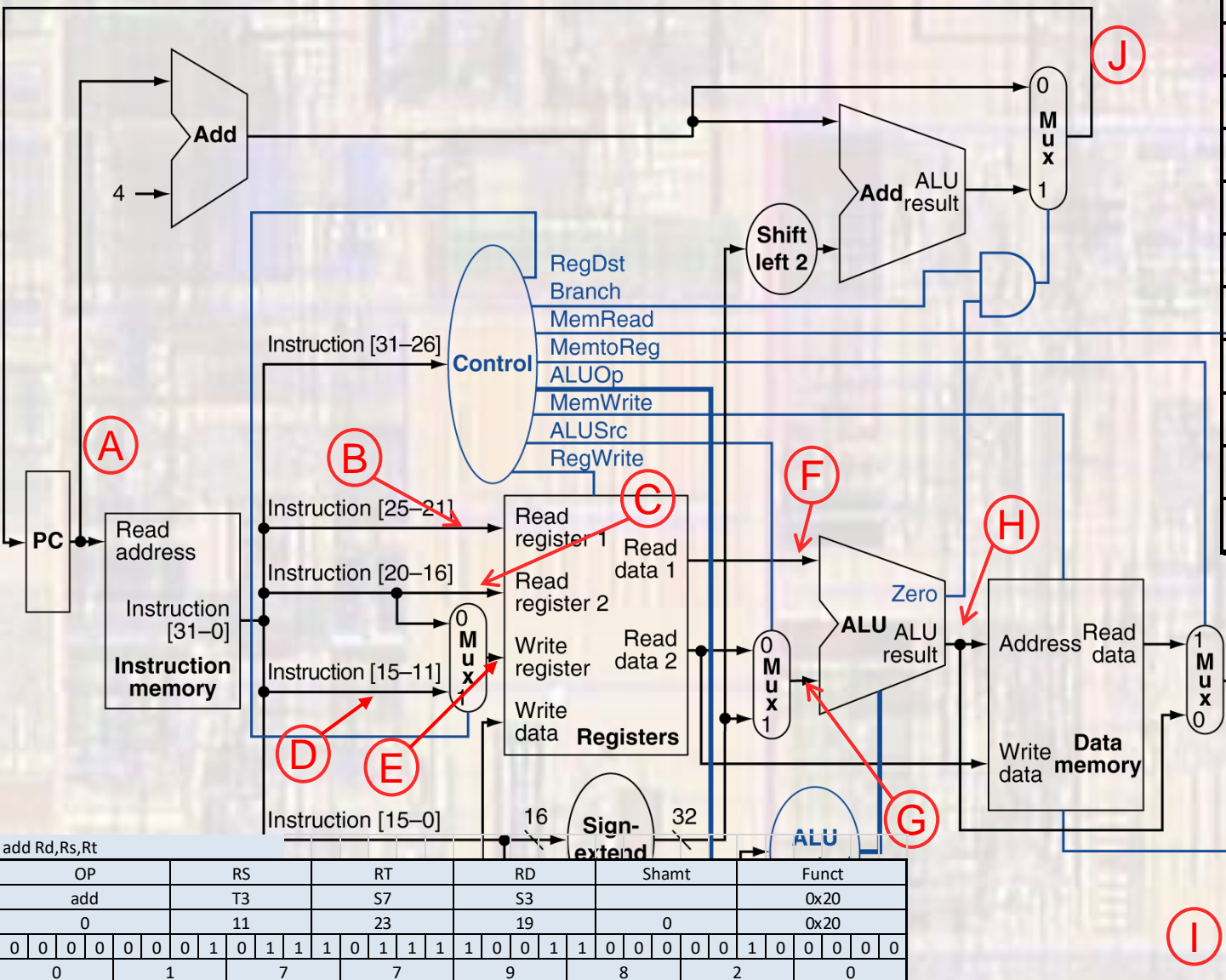
Bus/Wire	Value (hex)
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	

After completion of the instruction “add \$s3,\$t3,\$s7” indicate the value of each data bus.  
 Assume \$t3=0xDCBA, \$s7=0x4321, and the instruction was located at memory location 0x1220,  
 use x for unknown



Bus/Wire	Value (hex)
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	

After completion of the instruction “add \$s3,\$t3,\$s7” indicate the value of each data bus. Assume \$t3=0xDCBA, \$s7=0x4321, and the instruction was located at memory location 0x1220, use x for unknown



Bus/Wire	Value (hex)
A	1220
B	B
C	17
D	13
E	13
F	0000 DCBA
G	0000 4321
H	0001 1FDB
I	0001 1FDB
J	1224



# Simple Data Path

- MIPS Greed Card

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data



### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0/20 <sub>hex</sub>
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0/21 <sub>hex</sub>
And	and R	$R[rd] = R[rs] \& R[rt]$	0/24 <sub>hex</sub>
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c <sub>hex</sub>
Branch On Equal	beq I	$\text{if}(R[rs] == R[rt]) \text{PC} = \text{PC} + 4 + \text{BranchAddr}$	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne I	$\text{if}(R[rs] != R[rt]) \text{PC} = \text{PC} + 4 + \text{BranchAddr}$	(4) 5 <sub>hex</sub>
Jump	j J	$\text{PC} = \text{JumpAddr}$	(5) 2 <sub>hex</sub>
Jump And Link	jal J	$R[31] = \text{PC} + 8; \text{PC} = \text{JumpAddr}$	(5) 3 <sub>hex</sub>
Jump Register	jr R	$\text{PC} = R[rs]$	0/08 <sub>hex</sub>
Load Byte Unsigned	lbu I	$R[rt] = (24'b0, M[R[rs]] + \text{SignExtImm})(7:0)$	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu I	$R[rt] = (16'b0, M[R[rs]] + \text{SignExtImm})(15:0)$	(2) 25 <sub>hex</sub>
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui I	$R[rs] = (\text{imm}, 16'b0)$	f <sub>hex</sub>
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 <sub>hex</sub>
Nor	nor R	$R[rd] = \sim (R[rs]   R[rt])$	0/27 <sub>hex</sub>
Or	or R	$R[rd] = R[rs]   R[rt]$	0/25 <sub>hex</sub>
Or Immediate	ori I	$R[rt] = R[rs]   \text{ZeroExtImm}$	(3) d <sub>hex</sub>
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0/2a <sub>hex</sub>
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b <sub>hex</sub>
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0/2b <sub>hex</sub>
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0/00 <sub>hex</sub>
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0/02 <sub>hex</sub>
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$	(2) 28 <sub>hex</sub>
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt]; R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 <sub>hex</sub>
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0)$	(2) 29 <sub>hex</sub>
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b <sub>hex</sub>
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0/22 <sub>hex</sub>
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0/23 <sub>hex</sub>

- (1) May cause overflow exception
- (2)  $\text{SignExtImm} = \{ 16(\text{immediate}[15]), \text{immediate} \}$
- (3)  $\text{ZeroExtImm} = \{ 16(1'b'0), \text{immediate} \}$
- (4)  $\text{BranchAddr} = \{ 14(\text{immediate}[15]), \text{immediate}, 2'b'0 \}$
- (5)  $\text{JumpAddr} = \{ \text{PC} + 4[31:28], \text{address}, 2'b'0 \}$
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair;  $R[rt] = 1$  if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31	26-25	21-20	16-15	11-10	6-5
I	opcode	rs	rt	immediate		
	31	26-25	21-20	16-15		
J	opcode	address				
	31	26-25				

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

### ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bc1t F	$\text{if}(FPcond) \text{PC} = \text{PC} + 4 + \text{BranchAddr}$	(4) 11/8/1-
Branch On FP False	bc1f F	$\text{if}(\sim FPcond) \text{PC} = \text{PC} + 4 + \text{BranchAddr}$	(4) 11/8/0-
Divide	div R	$R[Lo-Rs]/R[Rt]; \text{Hi} = R[Rs] \% R[Rt]$	0/-/-/1a
Divide Unsigned	divu R	$R[Lo-Rs]/R[Rt]; \text{Hi} = R[Rs] \% R[Rt]$	(6) 0/-/-/1b
FP Add Single	add.s F	$F[rd] = F[rs] + F[rt]$	11/10/-0
FP Add Double	add.d F	$\{F[rd], F[rd+1]\} = \{F[rs], F[rs+1]\} + \{F[rt], F[rt+1]\}$	11/11/-0
FP Compare Single	c.x.s F	$FPcond = (F[rs] op F[rt]) ? 1 : 0$	11/10/-y
FP Compare Double	c.x.d F	$FPcond = ((F[rs], F[rs+1]) op (F[rt], F[rt+1])) ? 1 : 0$	11/11/-y
FP Divide Single	div.s F	$F[rd] = F[rs] / F[rt]$	11/10/-3
FP Divide Double	div.d F	$\{F[rd], F[rd+1]\} = \{F[rs], F[rs+1]\} / \{F[rt], F[rt+1]\}$	11/11/-3
FP Multiply Single	mul.s F	$F[rd] = F[rs] * F[rt]$	11/10/-2
FP Multiply Double	mul.d F	$\{F[rd], F[rd+1]\} = \{F[rs], F[rs+1]\} * \{F[rt], F[rt+1]\}$	11/11/-2
FP Subtract Single	sub.s F	$F[rd] = F[rs] - F[rt]$	11/10/-1
FP Subtract Double	sub.d F	$\{F[rd], F[rd+1]\} = \{F[rs], F[rs+1]\} - \{F[rt], F[rt+1]\}$	11/11/-1
Load FP Single	lwc1 I	$F[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 31/-/-/1
Load FP Double	lwc2 I	$\{F[rt], F[rt+1]\} = M[R[rs] + \text{SignExtImm}]; \{F[rt+1], F[rt+2]\} = M[R[rs+1] + \text{SignExtImm} + 4]$	(2) 35/-/-/1
Move From Hi	mthi R	$R[rd] = \text{Hi}$	0/-/-/10
Move From Lo	mtlo R	$R[rd] = \text{Lo}$	0/-/-/12
Move From Control	mfc0 R	$R[rd] = \text{CR}[rs]$	10/00/-0
Multiply	mult R	$\{\text{Hi}, \text{Lo}\} = R[rs] * R[rt]$	0/-/-/18
Multiply Unsigned	multu R	$\{\text{Hi}, \text{Lo}\} = R[rs] * R[rt]$	(6) 0/-/-/19
Shift Right Arith.	sra R	$R[rd] = R[rt] \gg \text{shamt}$	0/-/-/3
Store FP Single	swc1 I	$M[R[rs] + \text{SignExtImm}] = F[rt]$	(2) 39/-/-/1
Store FP Double	swc2 I	$M[R[rs] + \text{SignExtImm}] = F[rt]; M[R[rs+1] + \text{SignExtImm} + 4] = F[rt+1]$	(2) 3d/-/-/1

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31	26-25	21-20	16-15	11-10	6-5
FI	opcode	fmt	ft	immediate		
	31	26-25	21-20	16-15		

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	b1t	$\text{if}(R[rs] < R[rt]) \text{PC} = \text{Label}$
Branch Greater Than	bgt	$\text{if}(R[rs] > R[rt]) \text{PC} = \text{Label}$
Branch Less Than or Equal	b1e	$\text{if}(R[rs] <= R[rt]) \text{PC} = \text{Label}$
Branch Greater Than or Equal	bge	$\text{if}(R[rs] >= R[rt]) \text{PC} = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$l8-\$l9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	Yes