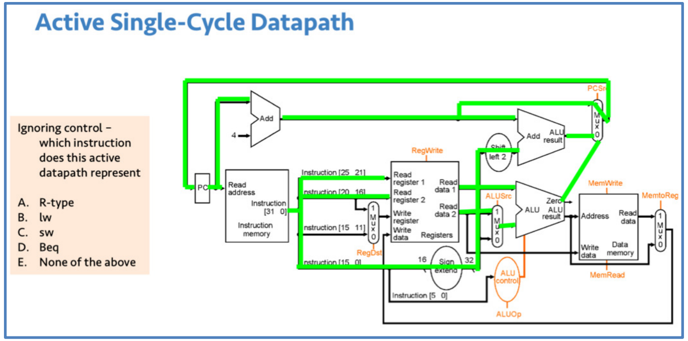
“The Control Path”

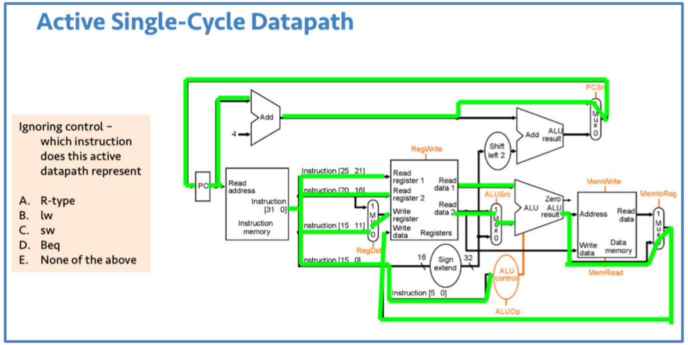
aka, what controls which wires are green?



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

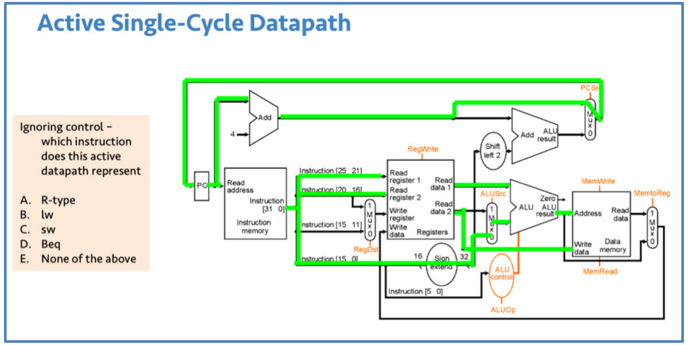
A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

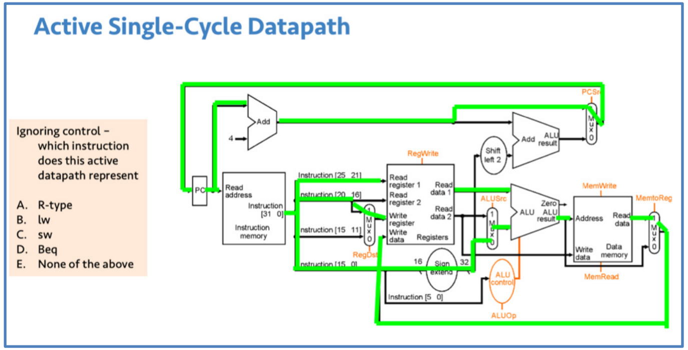
A. R-type
B. lw
C. sw
D. Beq
E. None of the above



Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

A. R-type
B. lw
C. sw
D. Beq
E. None of the above

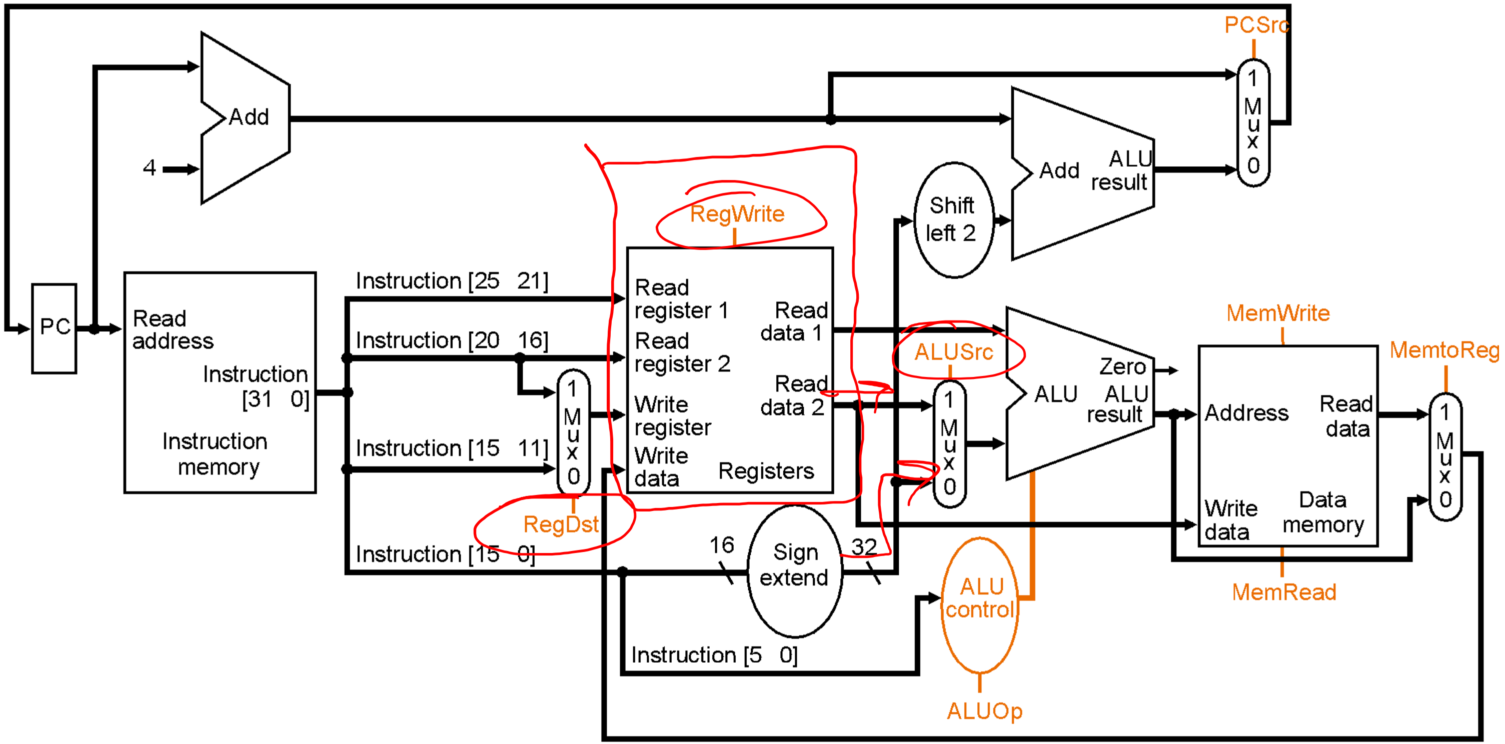


Active Single-Cycle Datapath

Ignoring control – which instruction does this active datapath represent

A. R-type
B. lw
C. sw
D. Beq
E. None of the above

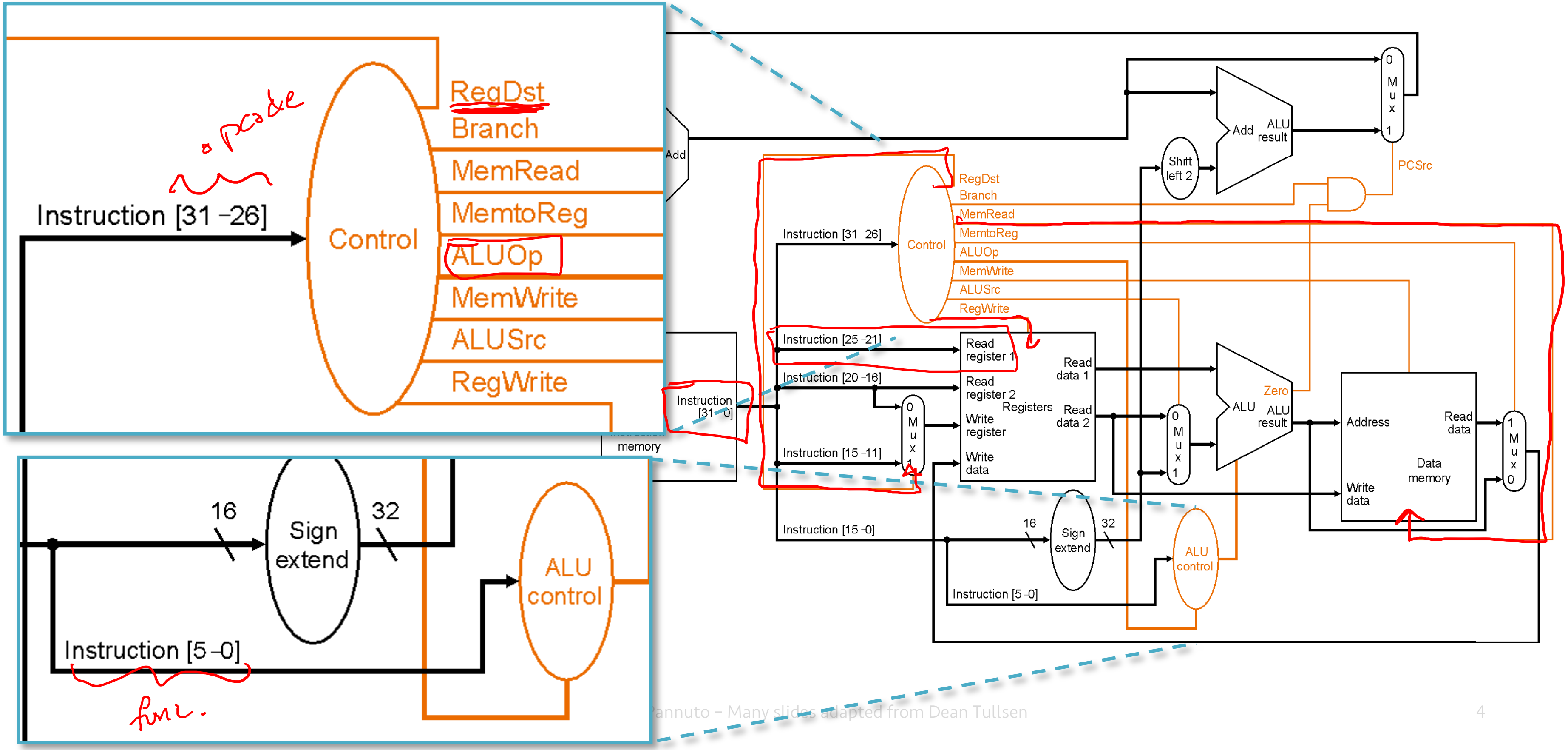
Control signals are all the parts in red

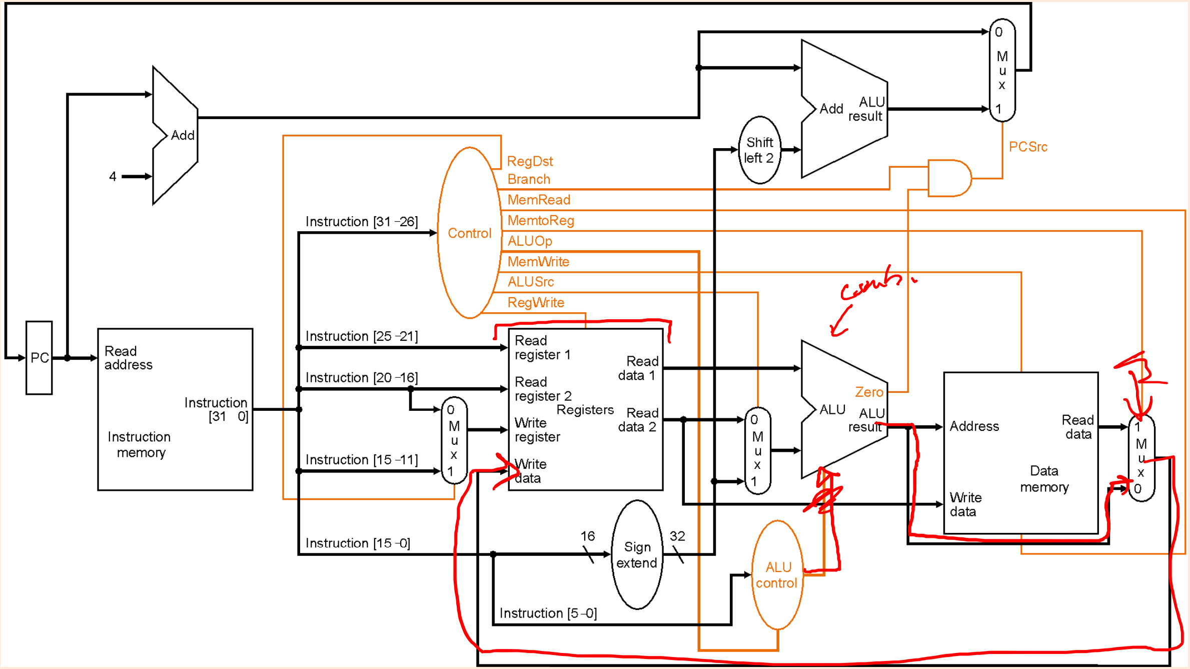


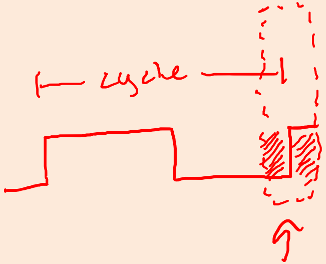
Where might we get control signals?

- Ideas?

Where do we get control signals?

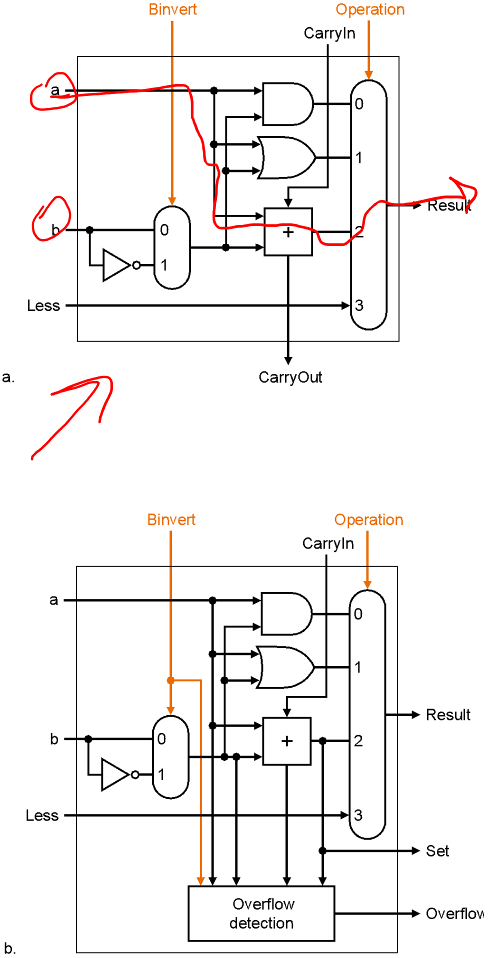


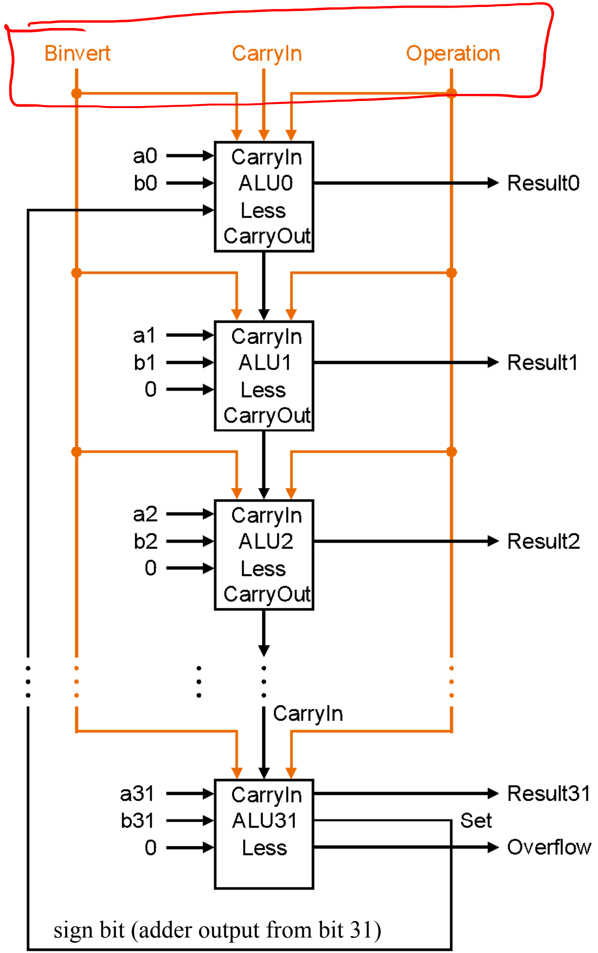




	Select the true statement for MIPS
A	Registers can be read in parallel with control signal generation
B	Instruction Read can be done in parallel with control signal generation
C	Registers can be written in parallel with control signal generation
D	The main ALU can execute in parallel with control signal generation
E	None of the above

Recall: The full ALU





	B_{invert}	$Carry_{In}$	Oper-ation
and	0	x	0
or	0	x	1
add	0	0	2
sub	1	1	2
beq	1	1	2
slt	1	1	3

1

0-3

2bit

3bits

ALU ctl

ALU control bits

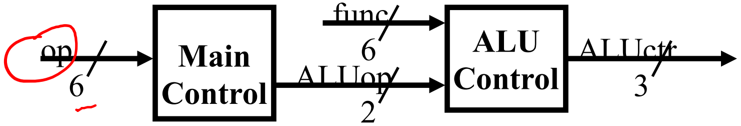
Note – book presents a 6-function ALU and a fourth ALU control input bit that never gets used (in simplified MIPS machine).

Don't let that confuse you.

- Recall: 5-function ALU

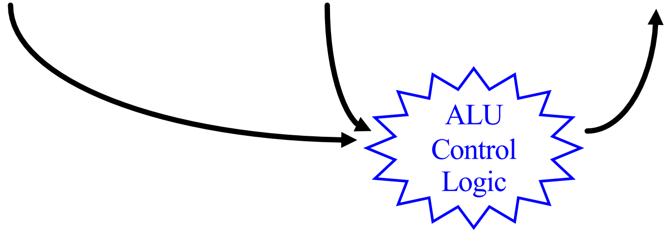
ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

- based on opcode (bits 31-26) and function code (bits 5-0) from instruction
- ALU doesn't need to know all opcodes!
  - Can summarize opcode with ALUOp (2 bits): 00 - lw,sw 01 - beq 10 - R-format



Generating ALU control

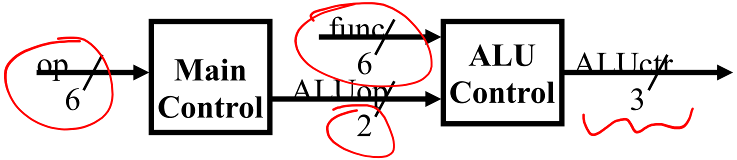
Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
lw	00	load word	xxxxxx	add	010
sw	00	store word	xxxxxx	add	010
beq	01	branch eq	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	slt	101010	slt	111



Generating individual ALU signals

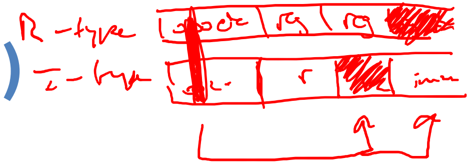
ALUop	Function	ALUCtr signals
00	xxxx	010
01	xxxx	110
10	0000	010
10	0010	110
10	0100	000
10	0101	001
10	1010	111

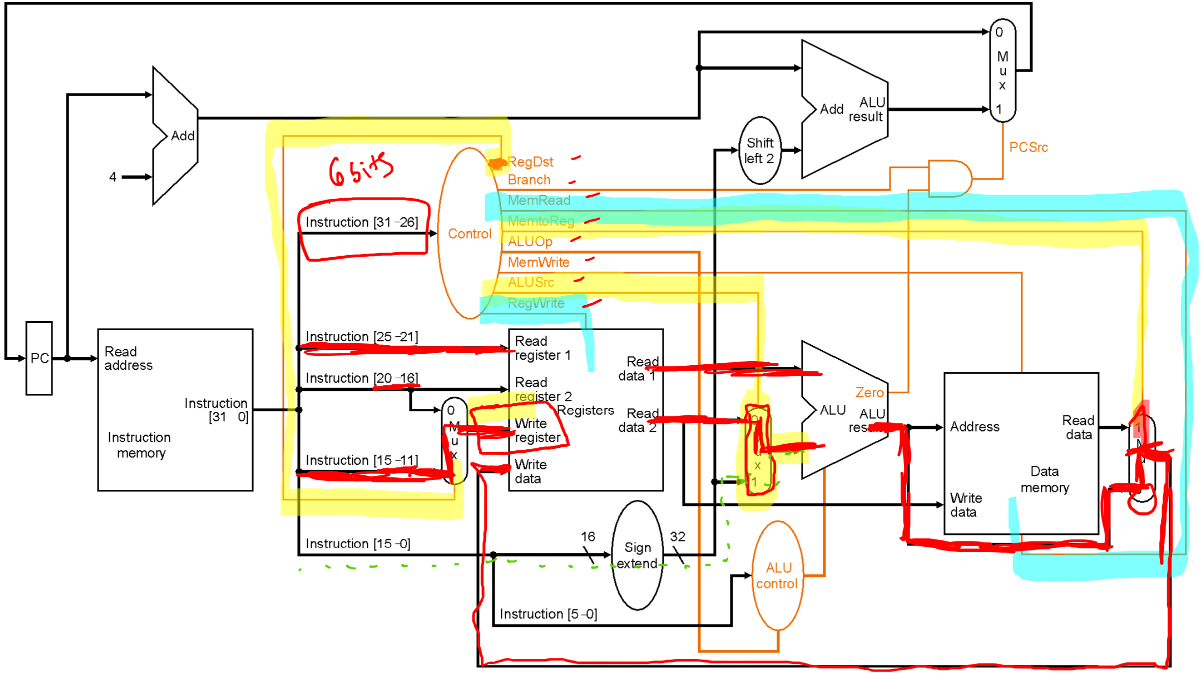
opcode func



```
ALUctr2 = (!ALUop1 & ALUop0) | (ALUop1 &  Func1)
ALUctr1 =        !ALUop1       | (ALUop1 & !Func2)
ALUctr0 =         ALUop1       & ( Func0 |  Func3)
```

R-Format Instructions (e.g., `add $d, $s0, $s1`)

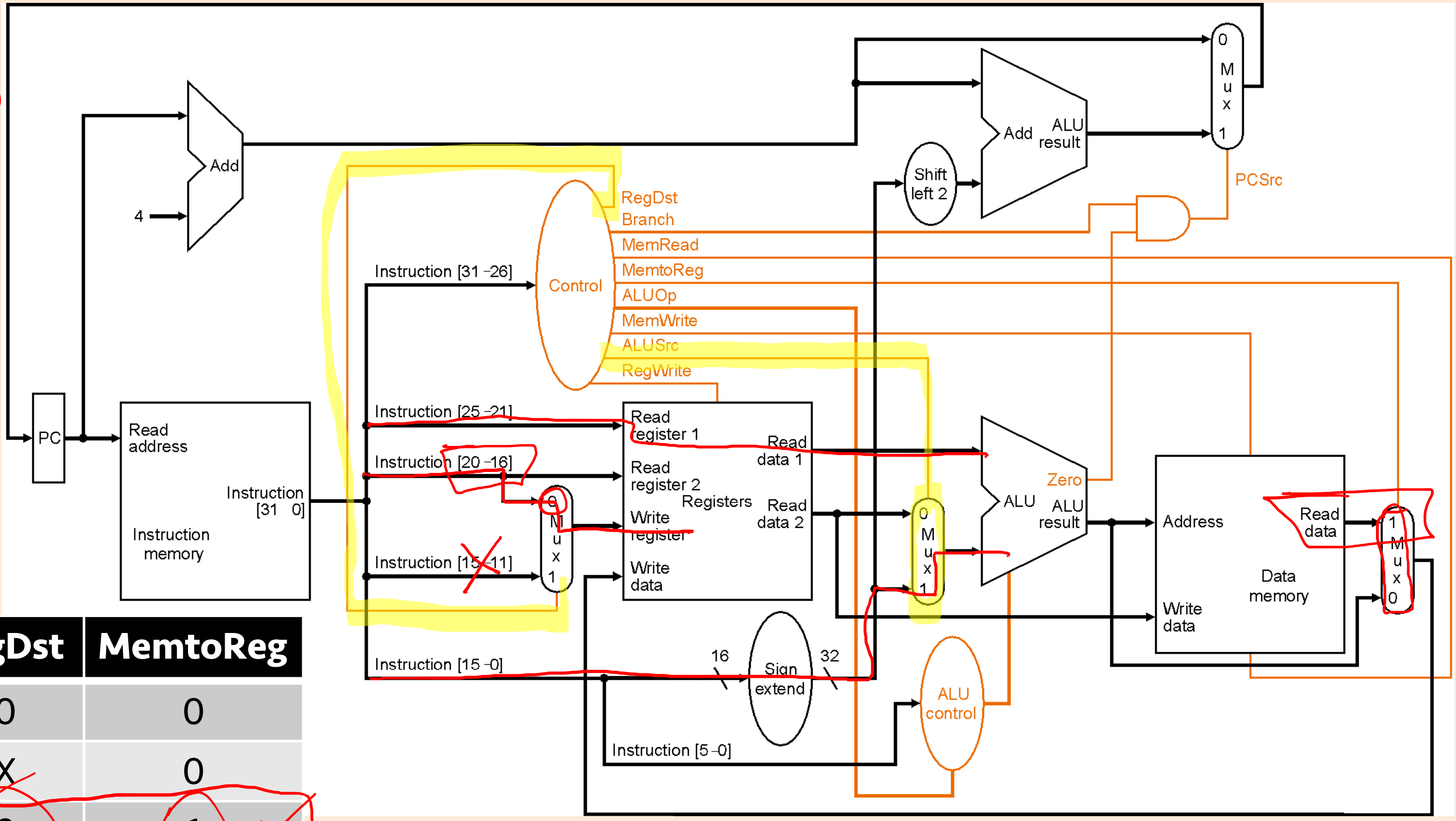




Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw								0	0
sw								0	0
beq								0	1

op code r r r

op code r r imm



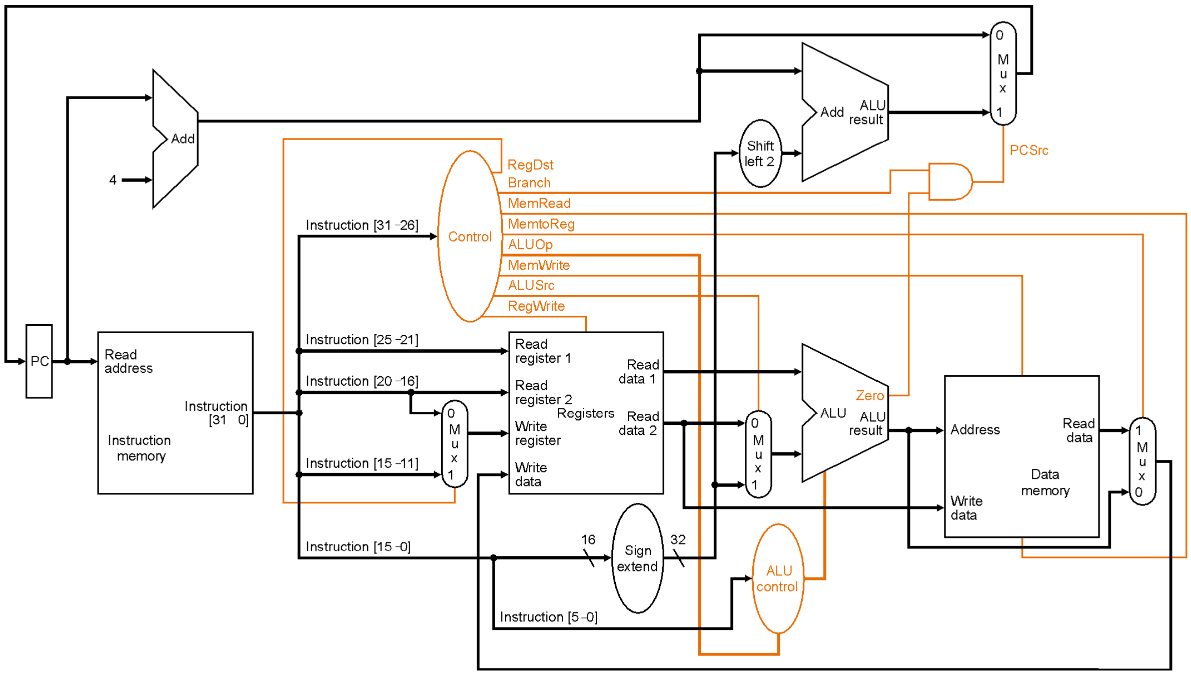
lw instruction control signals?

	ALUSrc	RegDst	MemtoReg
A	0	0	0
B	1	X	0
C	1 ✓	0	1 ✓
D	1	1	1
E	None of the above		

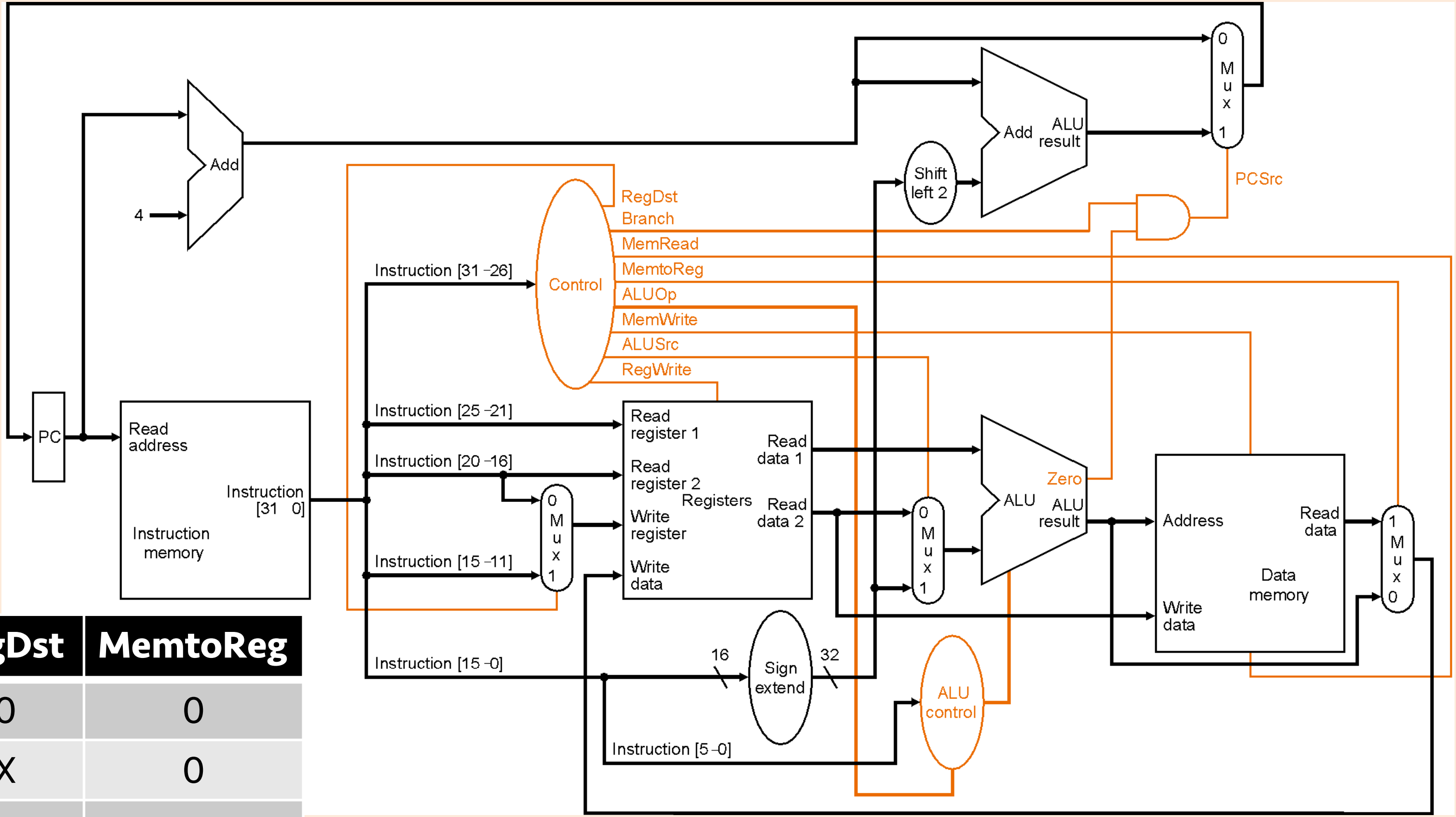
I-type lw $d, $s(10) – addr

imm

lw Control



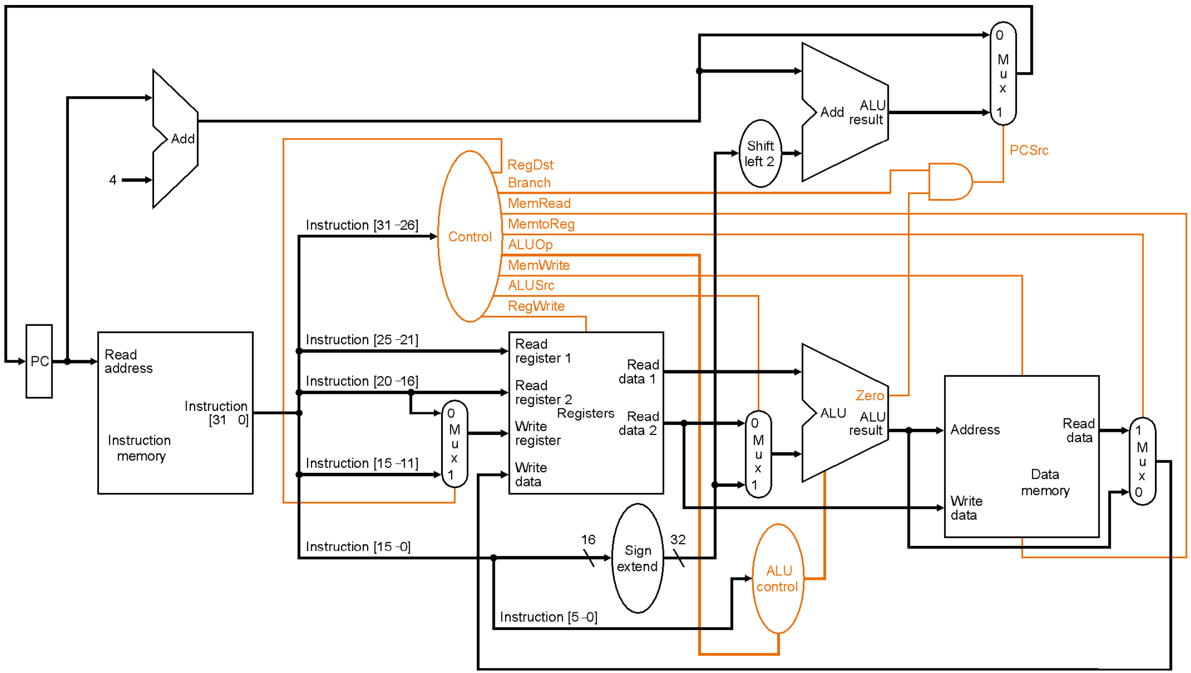
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw								0	0
sw								0	0
beq								0	1



`sw` instruction control signals?

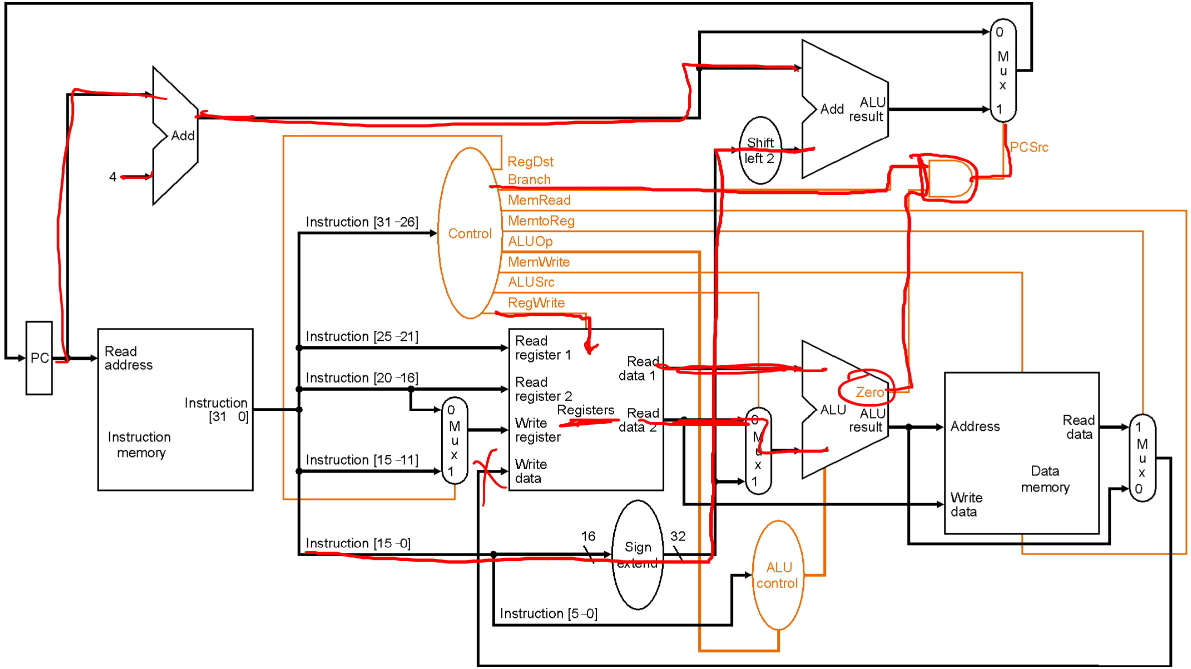
	ALUSrc	RegDst	MemtoReg
A	0	0	0
B	1	X	0
C	0	0	X
D	1	X	1
E	None of the above		

sw Control



Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw								0	0
beq								0	1

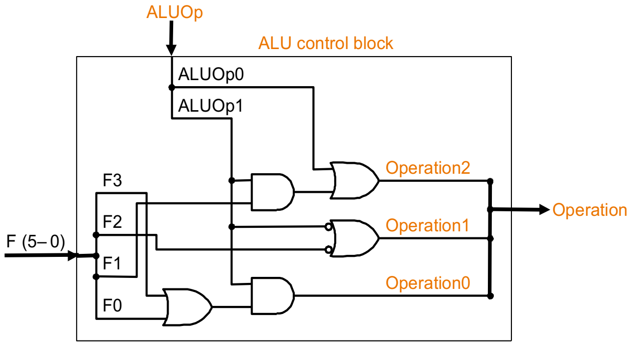
beq Control

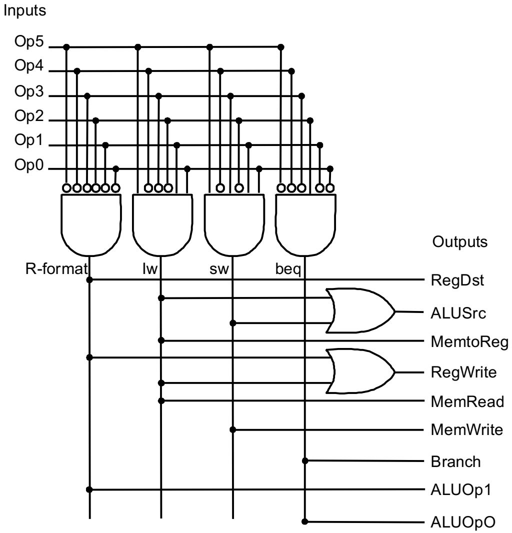


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Control Truth Table

		R-format	lw	sw	beq
Opcode		000000	100011	101011	000100
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

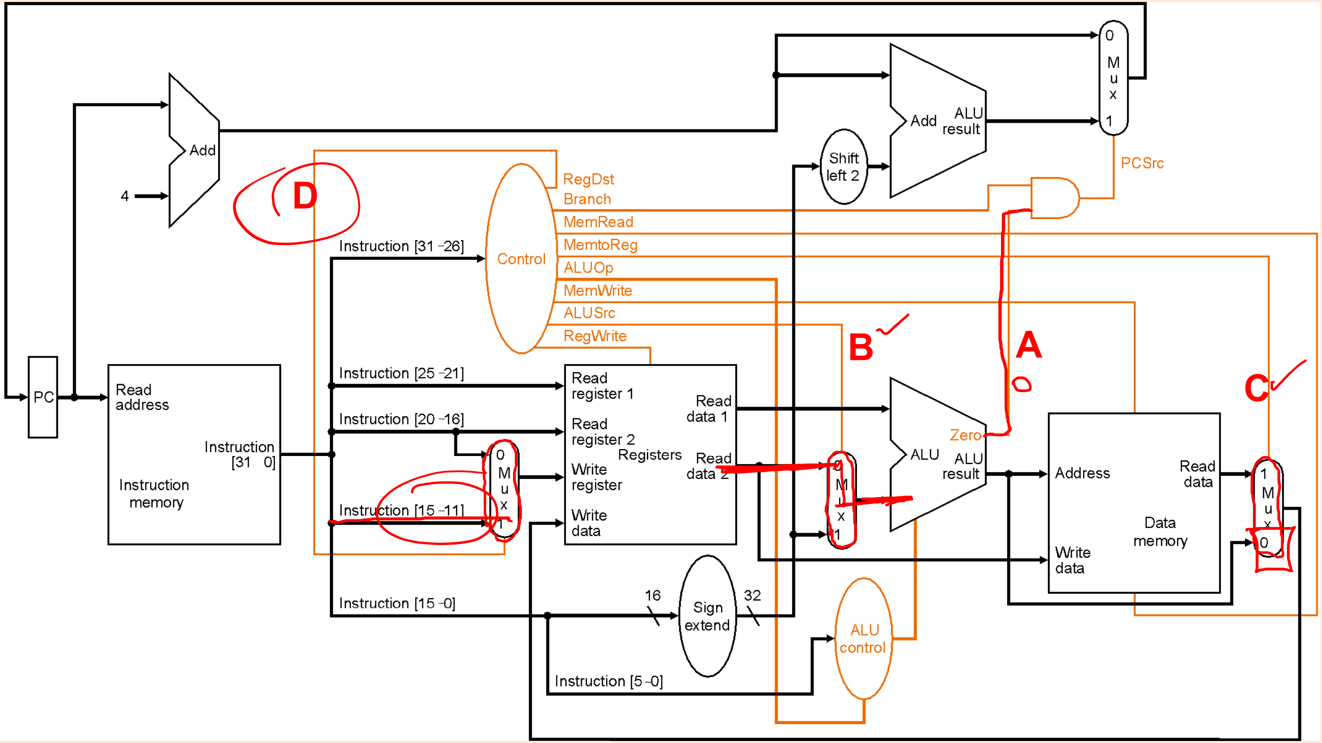




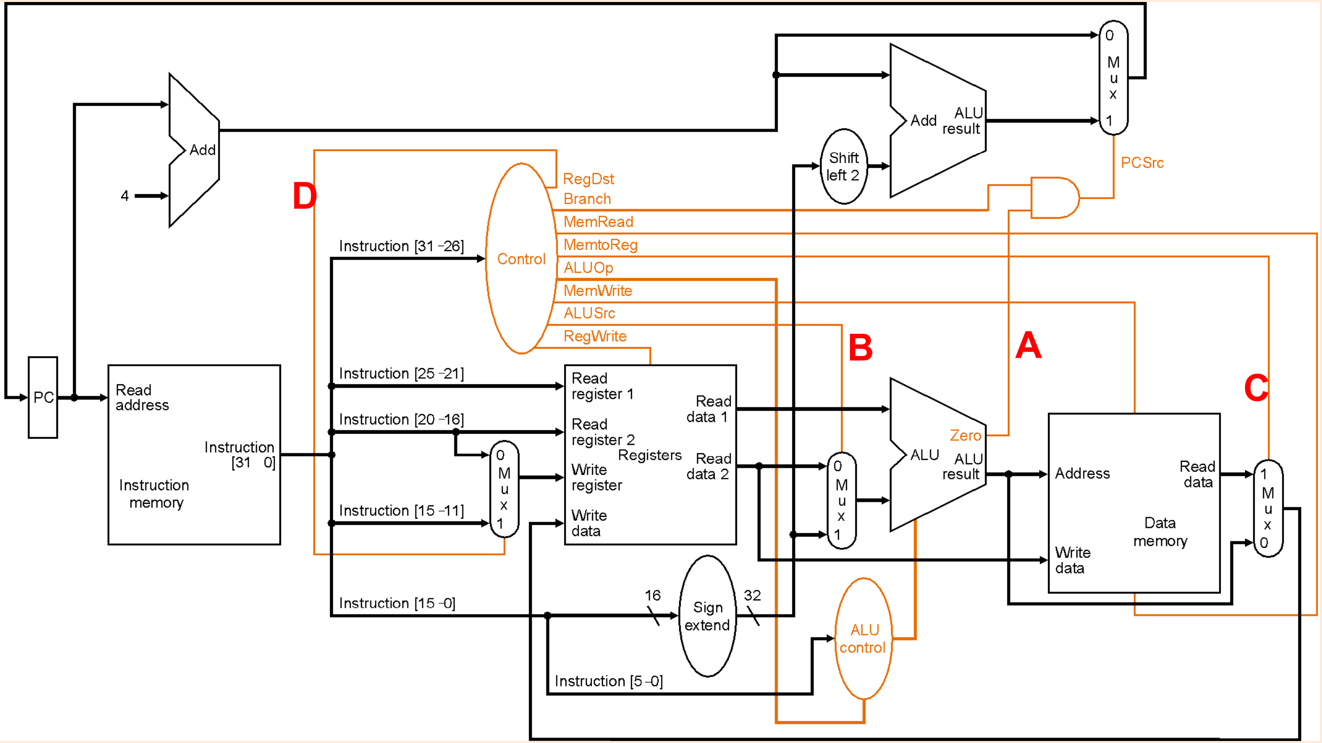
addi

Which wire – if always ZERO – would break add?

↳ R-type



Which wire – if always ONE – would break lw?



multi

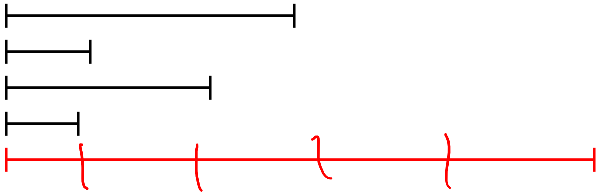
~~Single~~-Cycle CPU Summary

- Easy, particularly the control
- Which instruction takes the longest?
  - By how much? Why is that a problem?
- ET = IC * CPI↑ * CT↓
- What else can we do?
- When does a multi-cycle implementation make sense?
  - e.g., 70% of instructions take 75 ns, 30% take 200 ns?
  - suppose 20% overhead for extra latches
- Real machines have much *more* variable instruction latencies than this.

Let’s think about this multicycle processor…

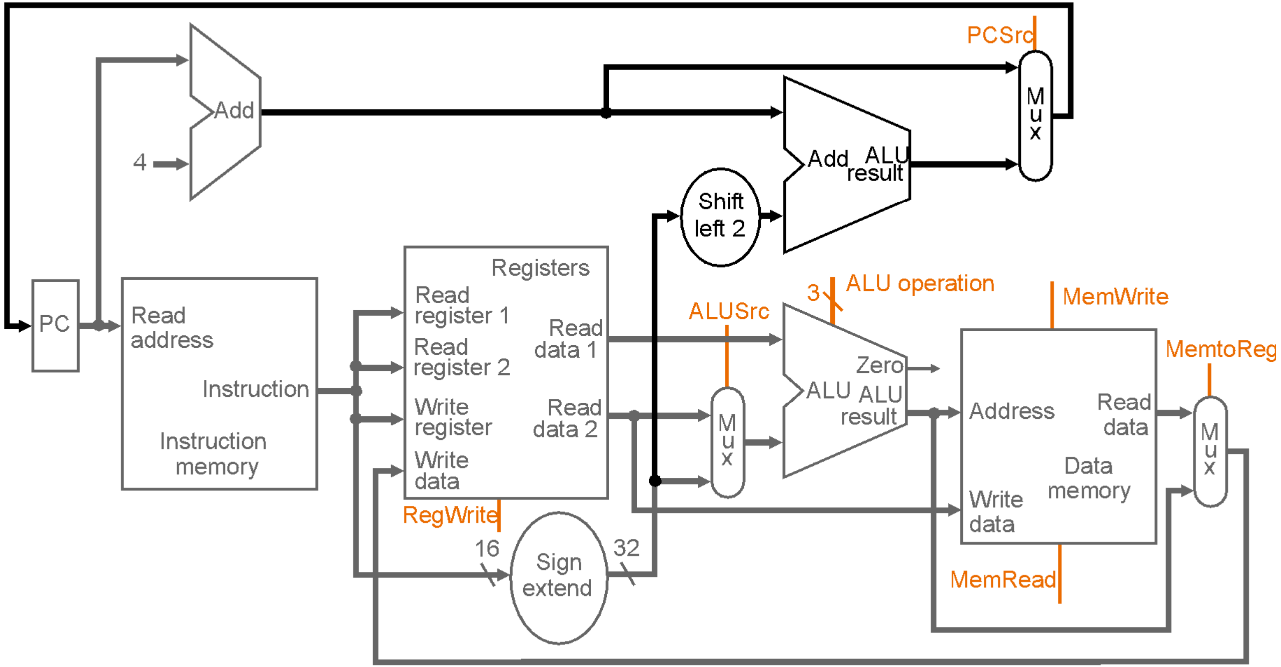
- (a very brief introduction…)

Why a Multiple Clock Cycle CPU?



- the problem => single-cycle cpu has a cycle time long enough to complete the longest instruction in the machine
- the solution => break up instruction execution into smaller tasks, each task taking one cycle
  - different instructions require different numbers of tasks (of cycles)
- other advantages => reuse of functional units (e.g., alu, memory)

High-level View



So Then,

- How many cycles does it take to execute
  - Add
  - BNE ?
  - LW
  - SW

- What about control logic?

- ET = IC * CPI * CT

Summary of instruction execution steps

“step” == “task” == “________”

4 5 3

Step	R-type	Memory	Branch

Instruction Fetch ✓	IR = Mem[PC]
	PC = PC + 4		

Instruction Decode/ register fetch ✓	A = Reg[IR[25-21]]
	B = Reg[IR[20-16]]
	ALUout = PC + (sign-extend(IR[15-0]) << 2)		

Execution, address computation, branch completion ✓	ALUout = A op B	ALUout = A + sign-extend(IR[15-0])	if (A==B) then PC=ALUout

Memory access or R-type completion ✓	Reg[IR[15-11]] = ALUout	memory-data = Mem[ALUout]
		or
		Mem[ALUout]= B
			✗

Write-back		Reg[IR[20-16]] = memory-data	✗

What is the fastest, slowest class of instruction in this MC machine?

Multicycle Questions

- How many cycles will it take to execute this code?

```
 lw  $t2, 0($t3)
 lw  $t3, 4($t3)
 beq $t2, $t3, Label  #assume not taken
 add $t5, $t2, $t3
 sw  $t5, 8($t3)
Label:  ...
```

	How many cycles to execute these 5 instructions?
A	5
B	25
C	22
D	21
E	None of the above

Multicycle Implications

```
    lw  $t2, 0($t3)
    lw  $t3, 4($t3)
    #assume not taken
    beq $t2, $t3, Label
    add $t5, $t2, $t3
    sw  $t5, 8($t3)
Label: ...
```

- What is the CPI of this program?
- What about a program that is 20% loads, 10% stores, 50% R-type, and 20% branches?

Single-Cycle, Multicycle CPU Summary

- Single-cycle CPU
  - CPI = 1, CT = LONG, simple design, simple control
  - No one has built a single-cycle machine in many decades
- Multi-cycle CPU
  - CPI > 1, CT = fairly short, complex control
  - Common up until maybe early 1990s, and dominant for many decades before that.

$$\downarrow ET = CPI^{\uparrow} * CT \Downarrow$$