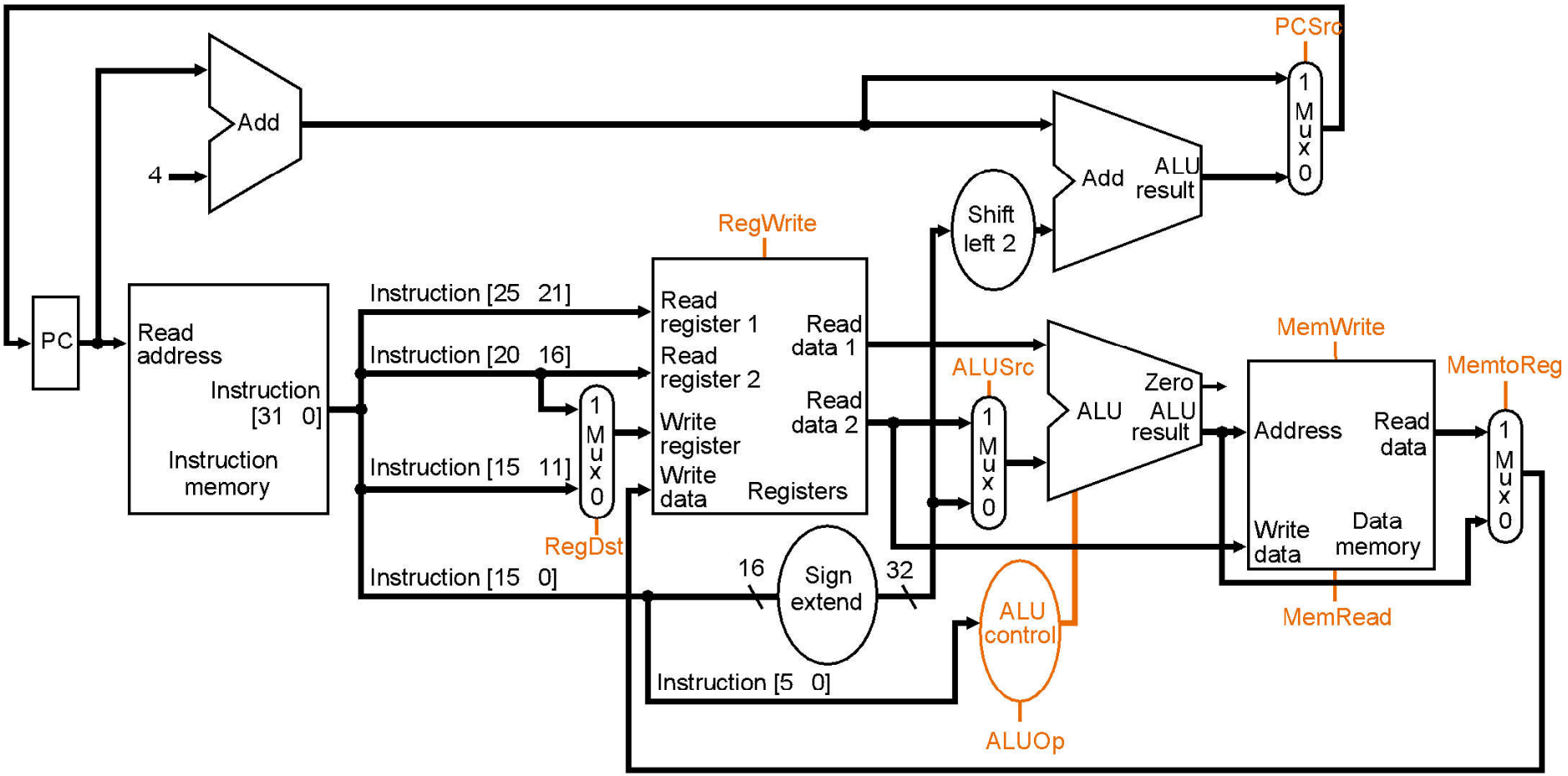
Control Logic for the Single-Cycle CPU

or

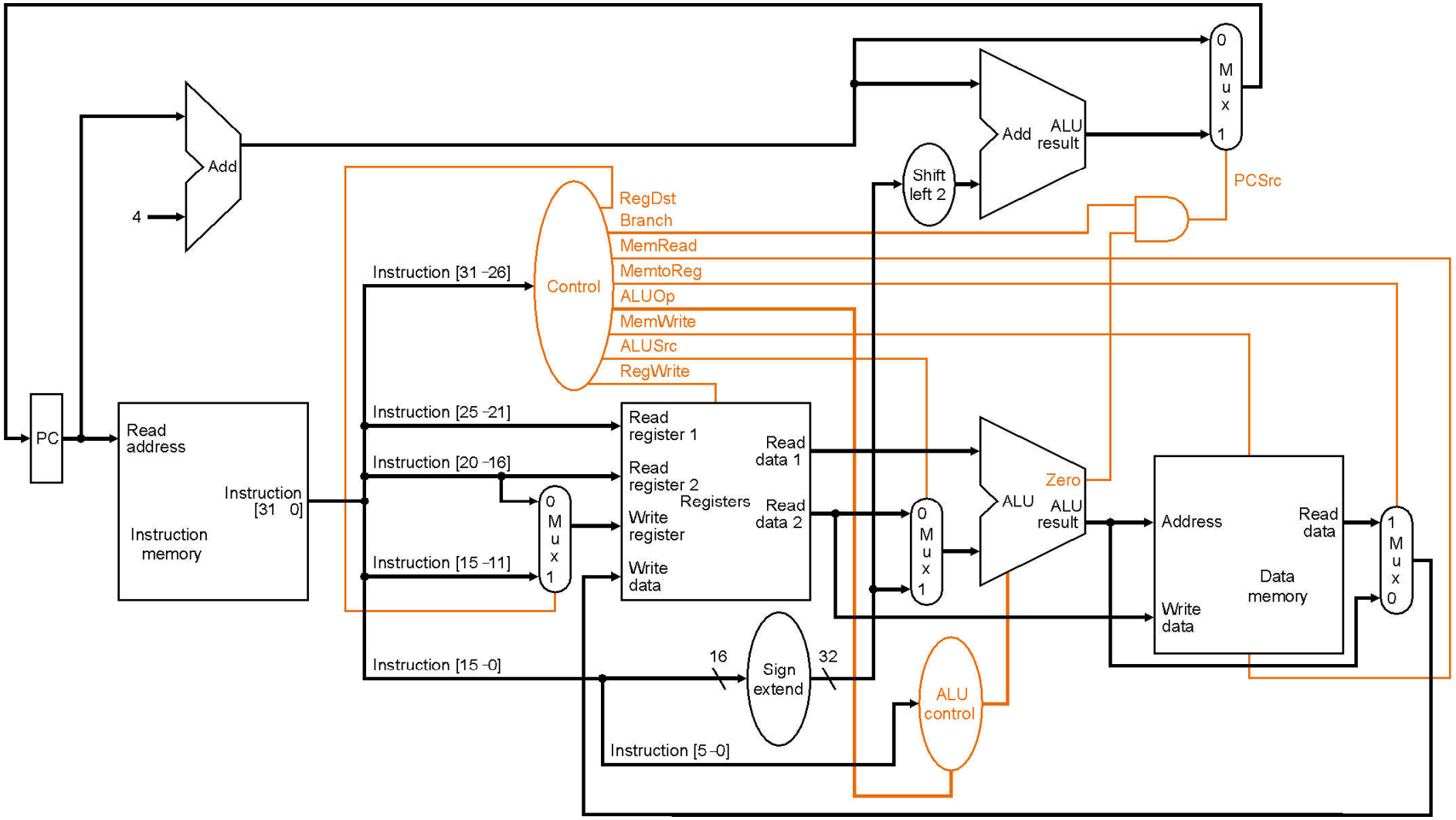
Who's in charge here?

Putting it All Together: A Single Cycle Datapath

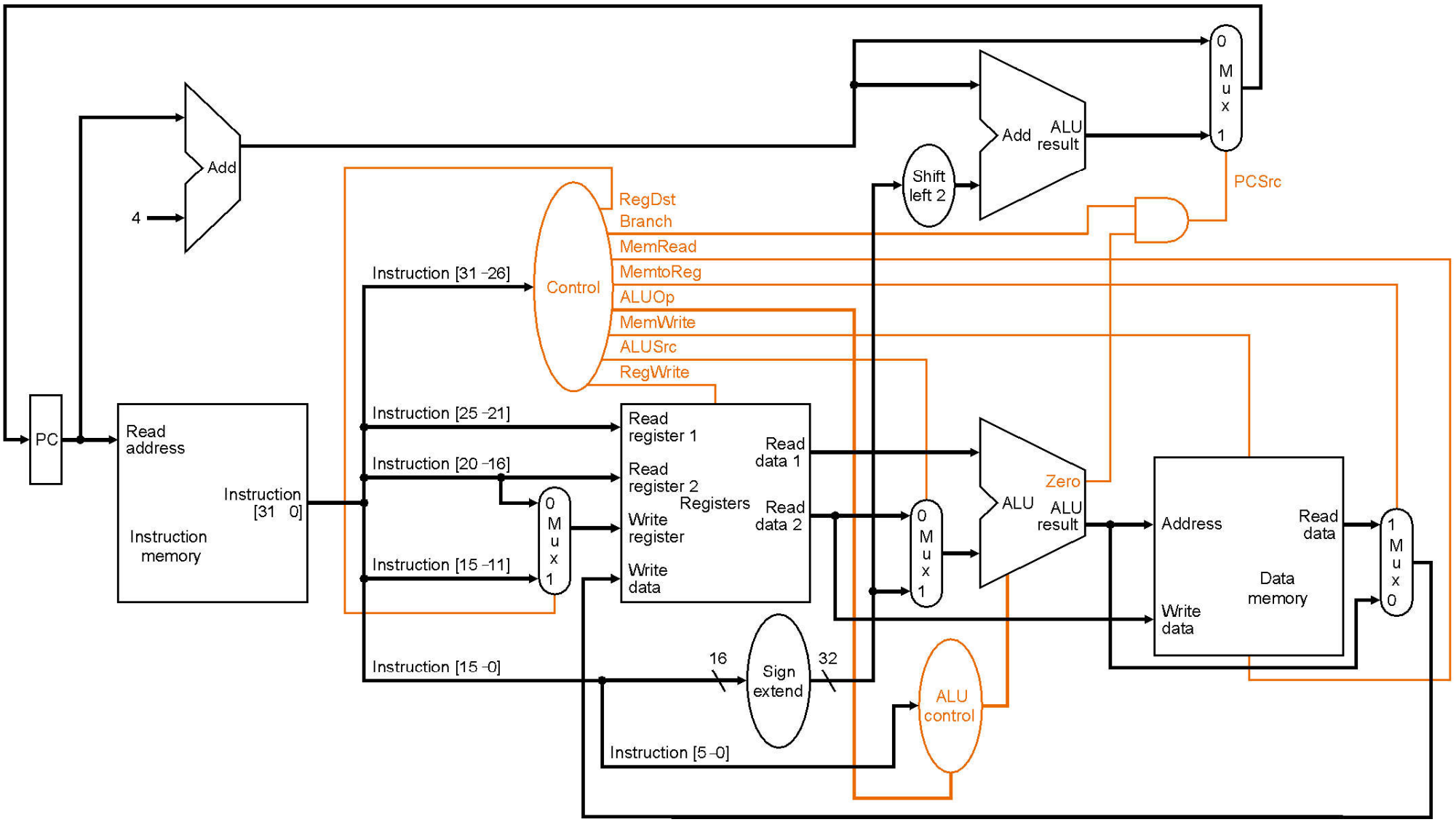
- We have everything except **control signals**



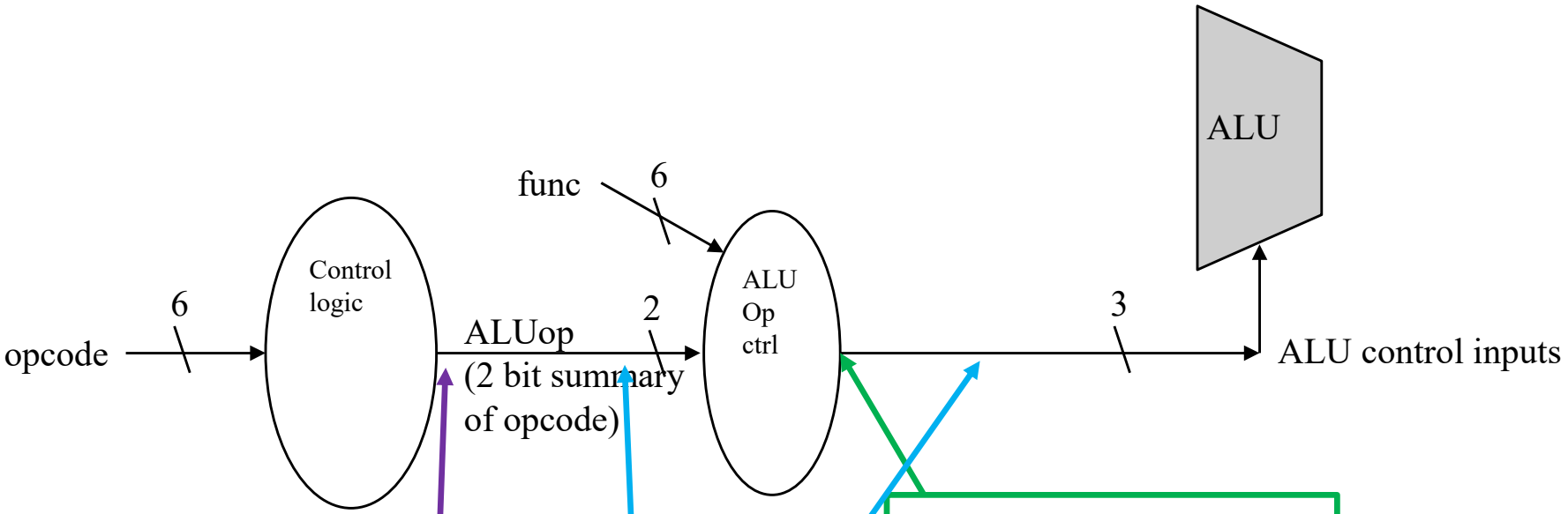
Okay, then, what about those Control Signals?



Let's start with the ALU control



Let's start with the ALU control



"truth table"

ALUop	func	output
00	X	010 (add)
01	X	110 (sub)
10	20	010 (add)
10	22	110 (sub)
10	24	000 (and)
10	25	001 (or)
10	2A	111 (slt)

"truth table"

lw, sw =>	00
Beq =>	01
R-format =>	10

Note – if we were supporting the entire MIPS ISA, we'd need more bits here and here.

ALU control bits

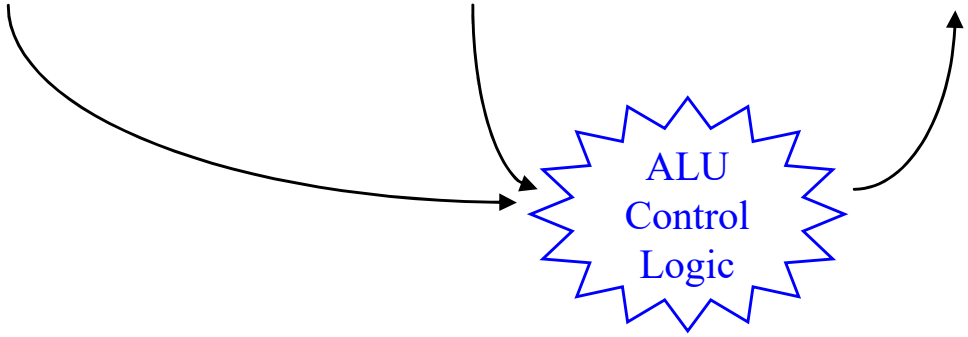
- Recall: 5-function ALU

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

Note – these are somewhat arbitrary, but actually map exactly to our Binvert/Carryin and Oper signals we derived last class.

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



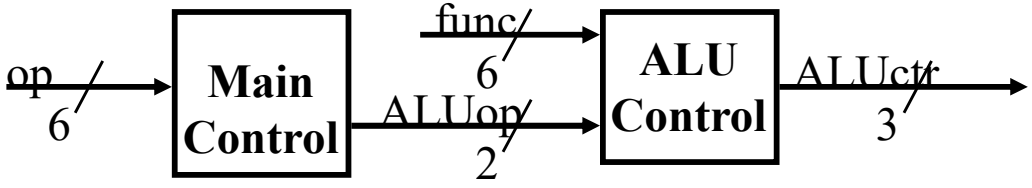
And just in case you really wanted the logic equations.

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

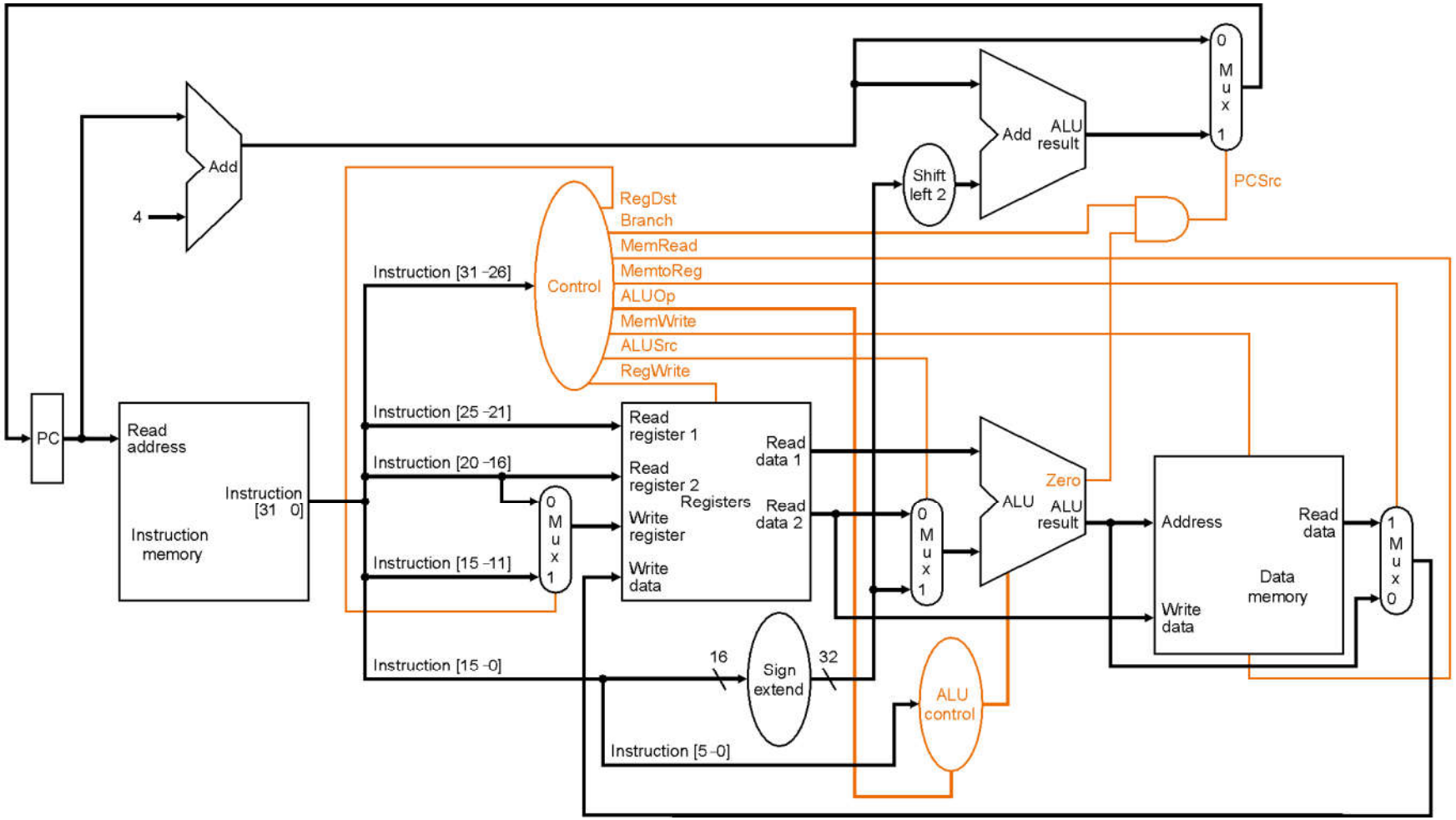
ALUctr2 = (!ALUop1 & ALUop0) | (ALUop1 & Func1)

ALUctr1 = !ALUop1 | (ALUop1 & !Func2)

ALUctr0 =ALUop1 & (Func0 | Func3)

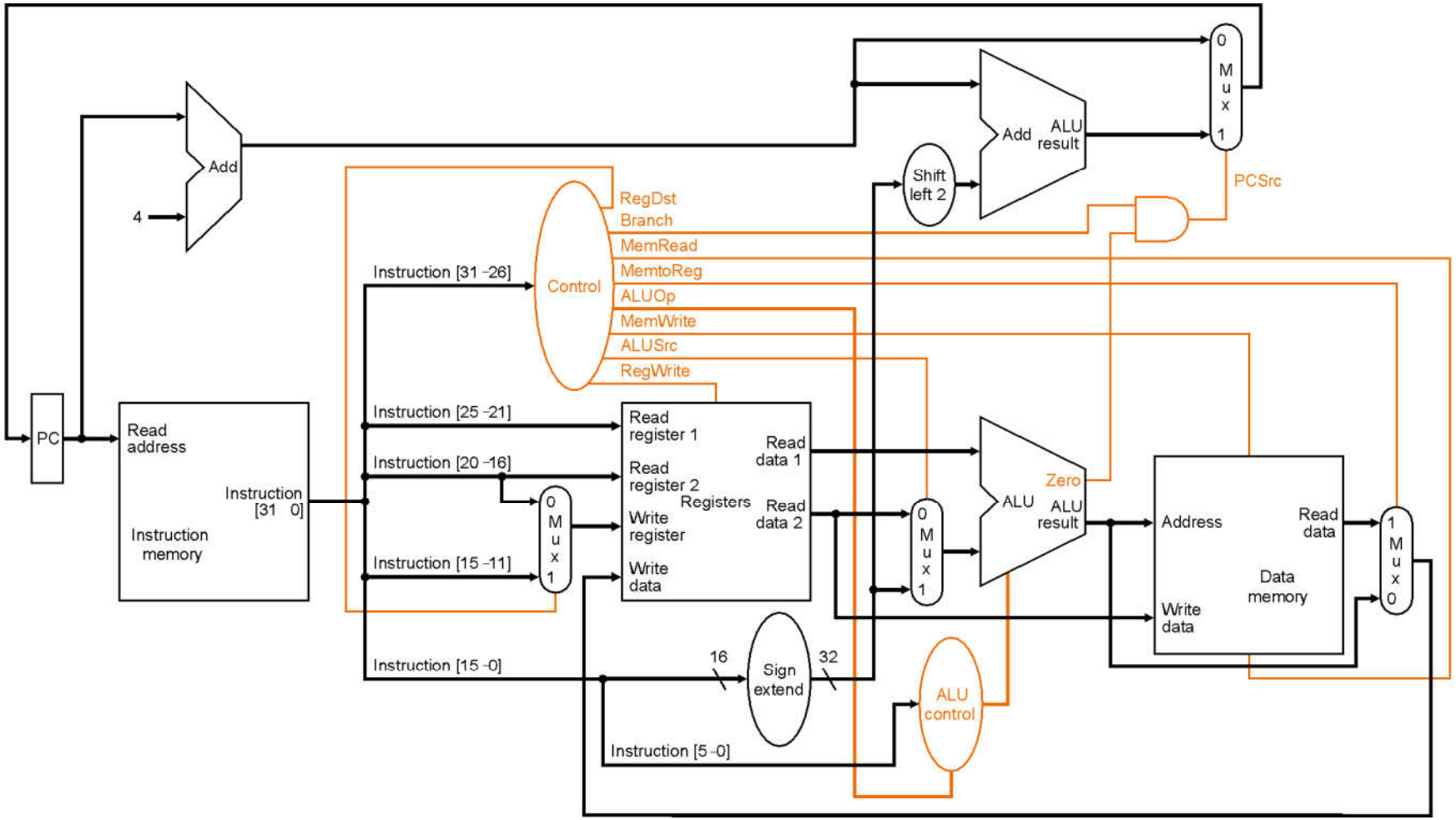


R-Format Instructions (e.g., Add)



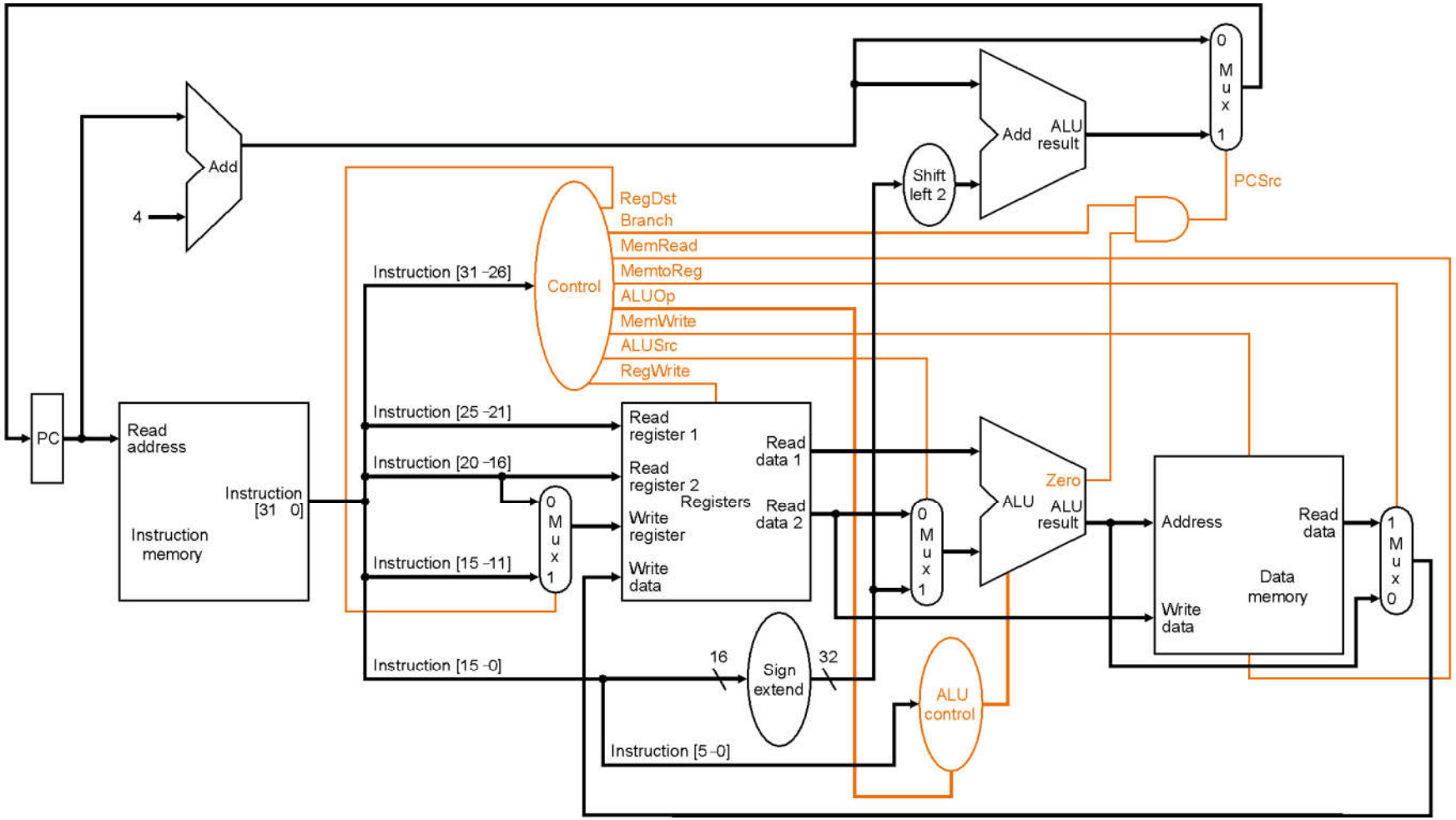
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format								1	0
`lw`								0	0
`sw`								0	0
`beq`								0	1

lw Control



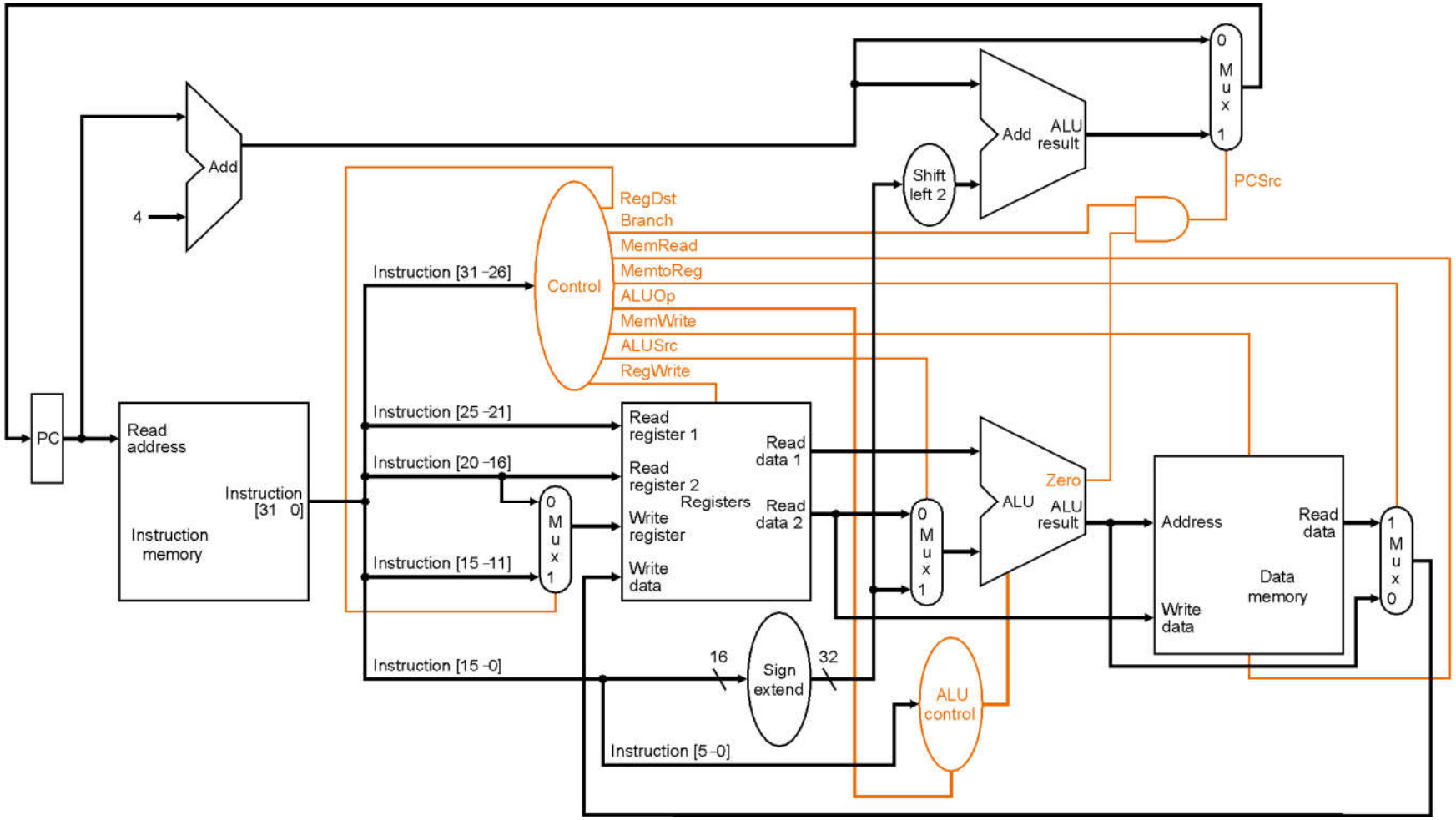
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`								0	0
`sw`								0	0
`beq`								0	1

sw 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



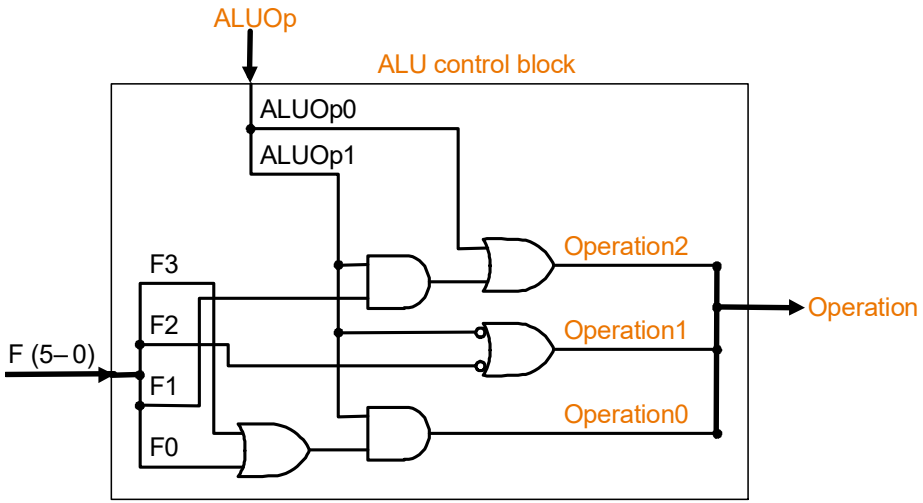
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`								0	1

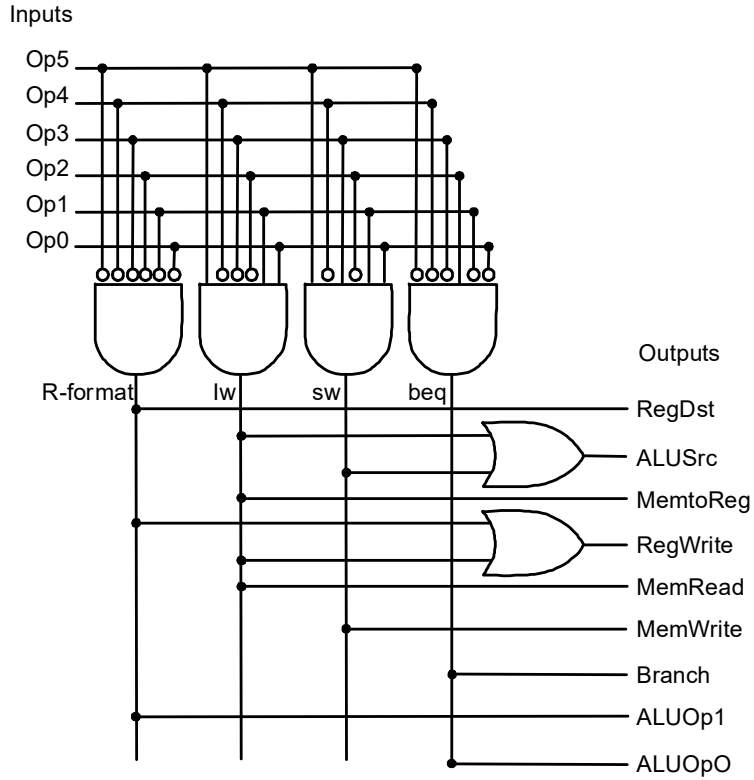
Control Truth Table

		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

Control

- Simple combinational logic (truth tables)
- And again, just in case you wanted to see the logic:





Single-Cycle CPU Summary

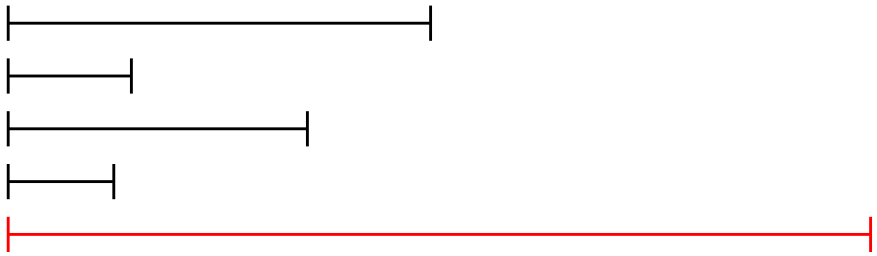
- Easy, particularly the control
- Which instruction takes the longest? By how much? Why is that a problem?
- ET = IC * CPI * CT
- What else can we do?
- When does a multi-cycle implementation make sense?
  - e.g., 70% of instructions take 75 ns, 30% take 200 ns?
  - suppose 20% overhead for extra latches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

Let's think about this multicycle processor…

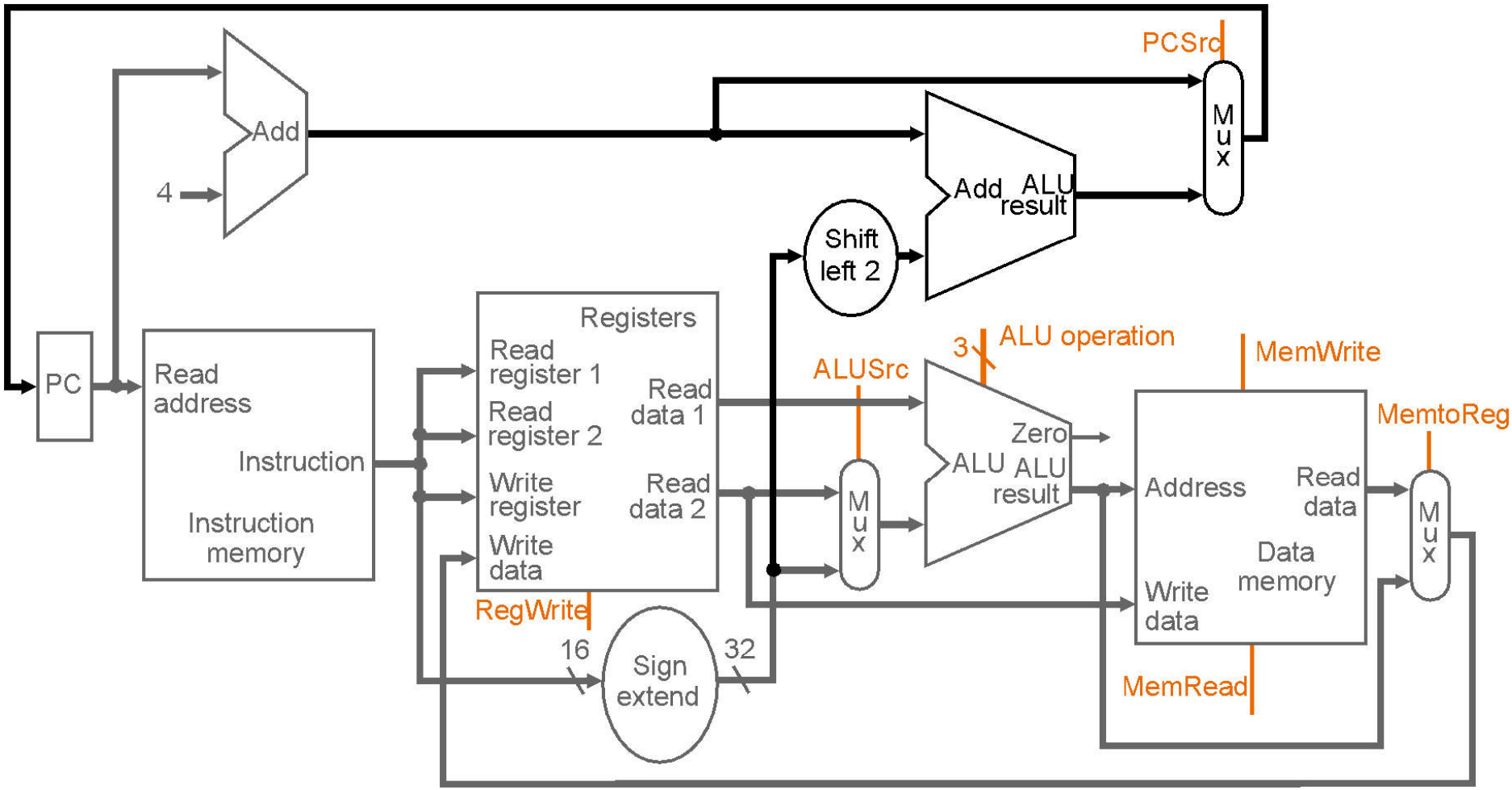
- (a very brief introduction…)

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 execution into smaller tasks, each task taking a cycle, different instructions requiring different numbers of cycles or tasks
- 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 execution steps

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	

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:    ...
```

- What is going on during the 8th cycle of execution?

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

Hint

- Some of the questions from this week's homework assume (or, at least make more sense if you assume) a multicycle machine.
  - 1 instruction at a time
  - Each takes a variable number of cycles
  - Cpi can be calculated if you know the instruction mix

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 early1990s, and dominant for many decades before that.