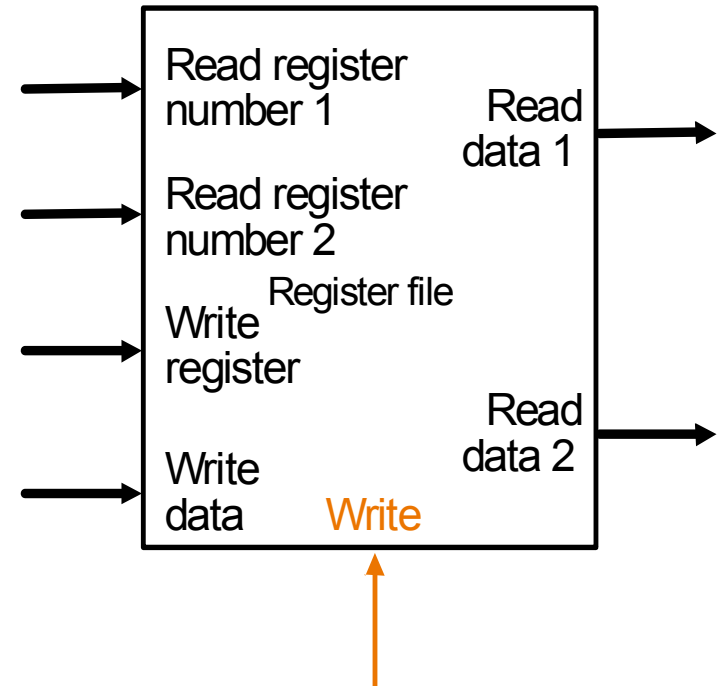
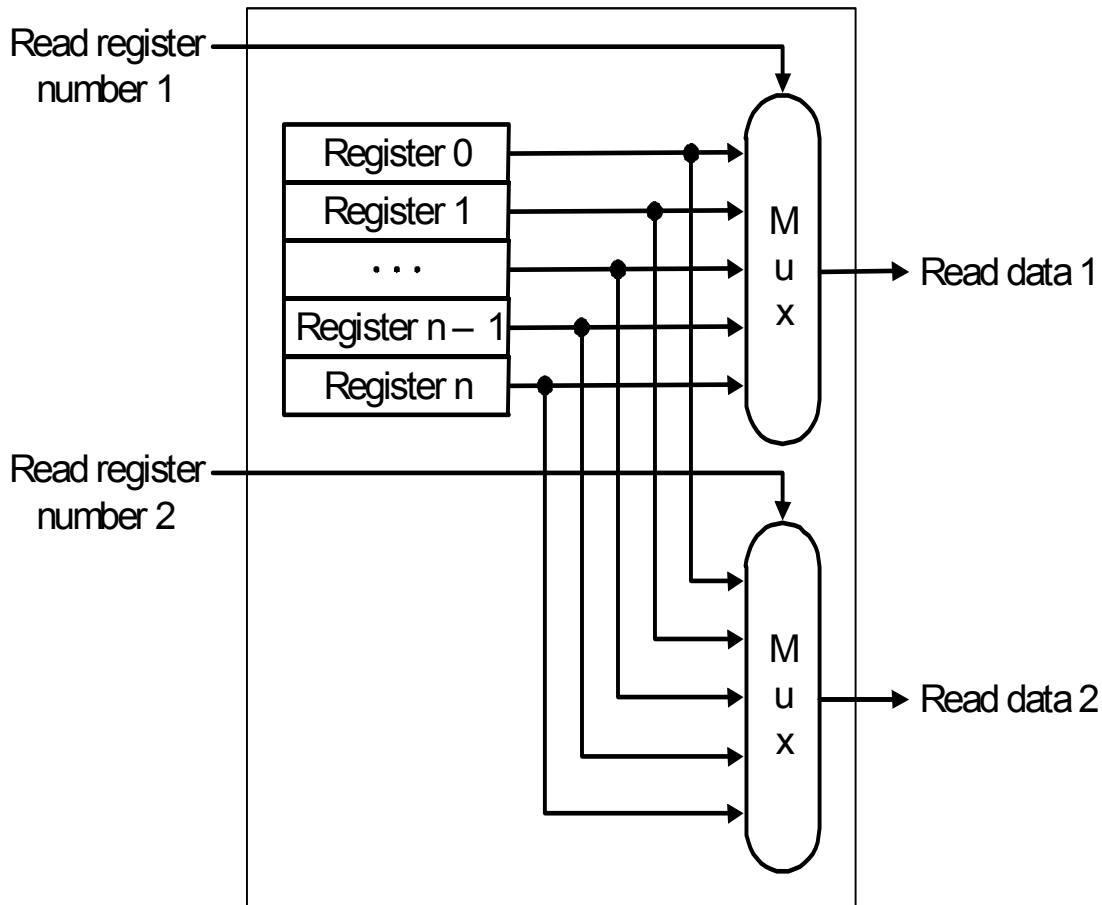


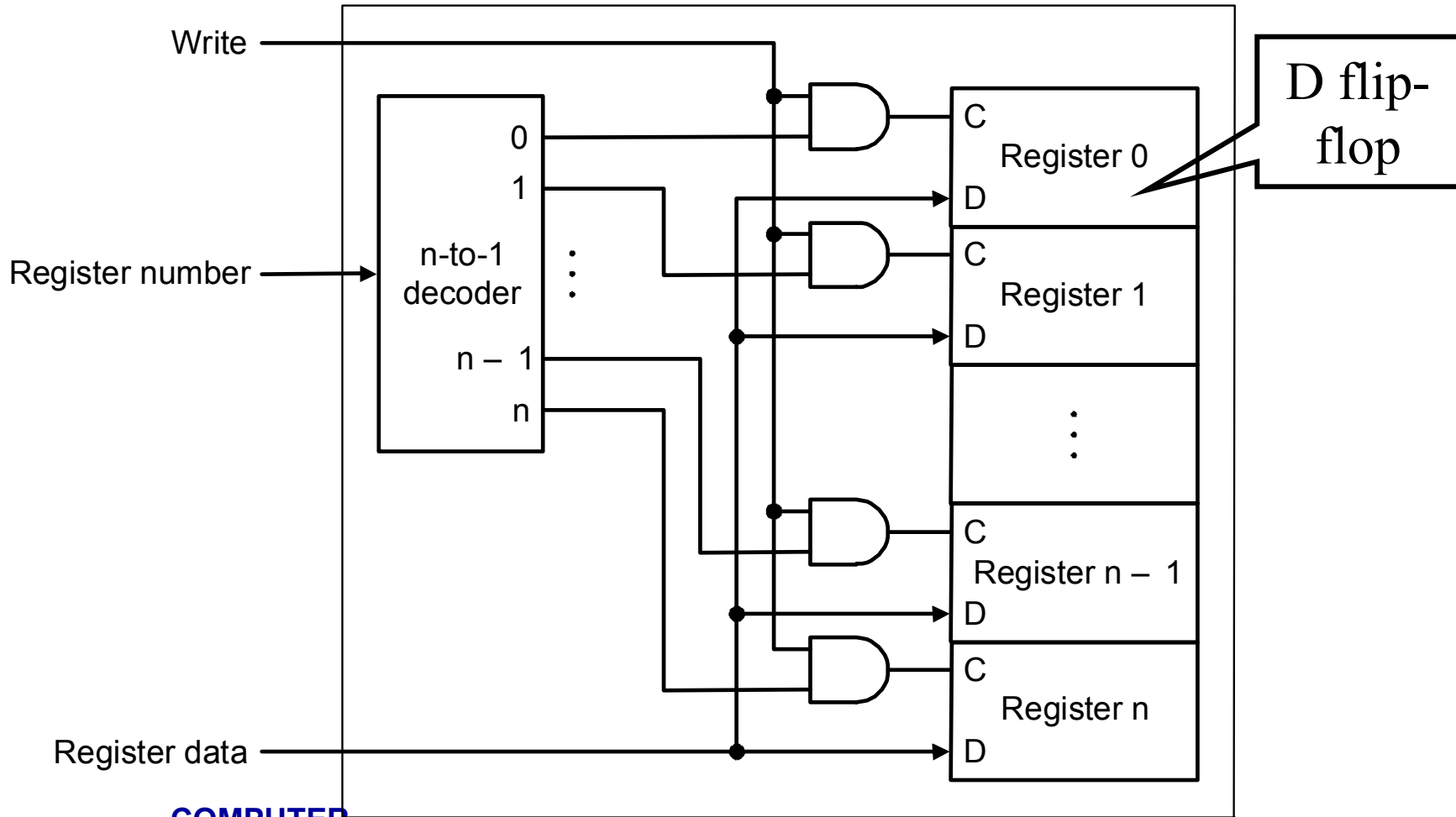
Register File

□ Built using D flip-flops



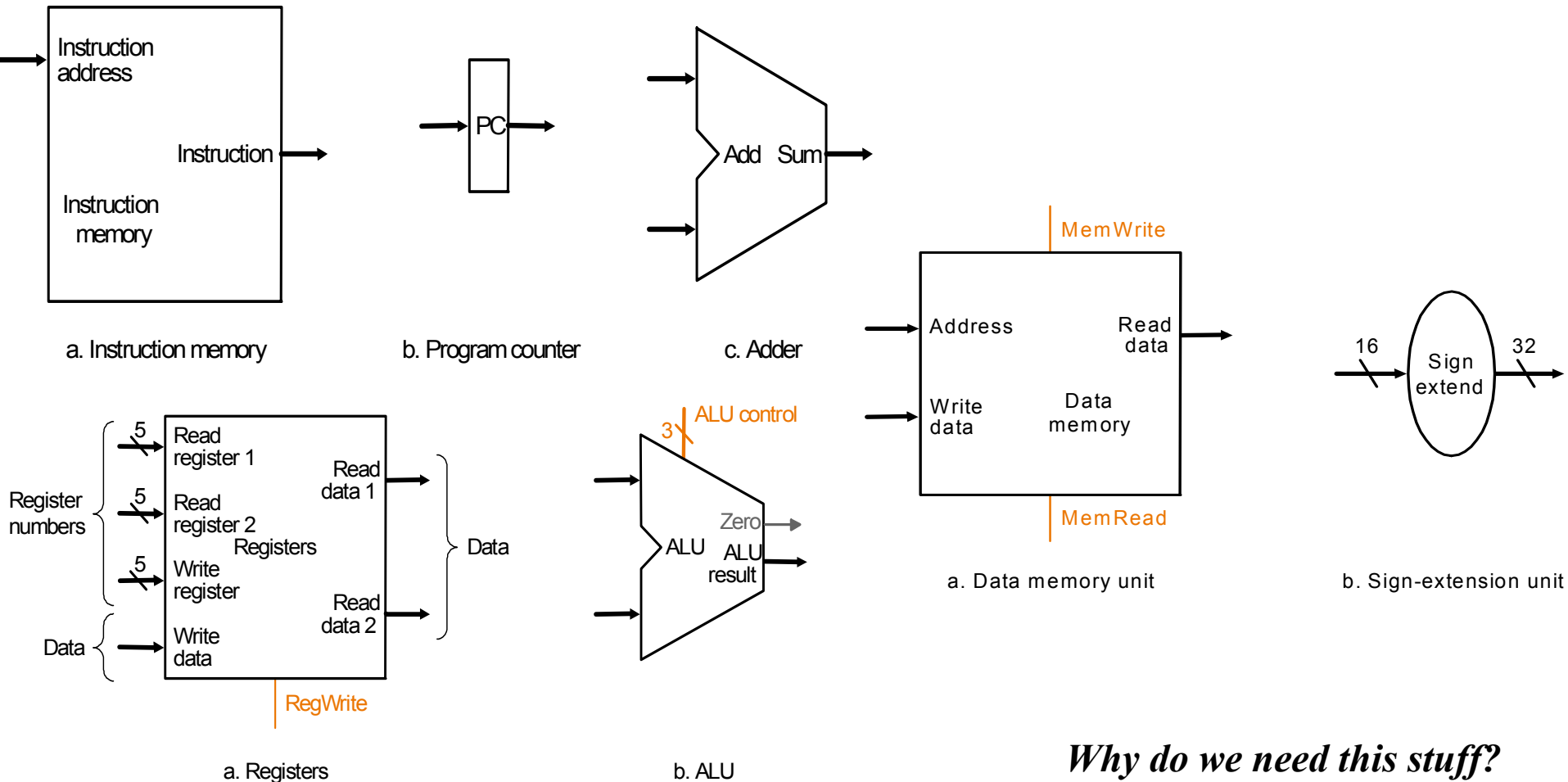
Register File

- Note: we still use the real clock to determine when to write



Simple Implementation

□ Include the functional units we need for each instruction



Why do we need this stuff?

Building the Datapath

- Use multiplexors to stitch them together

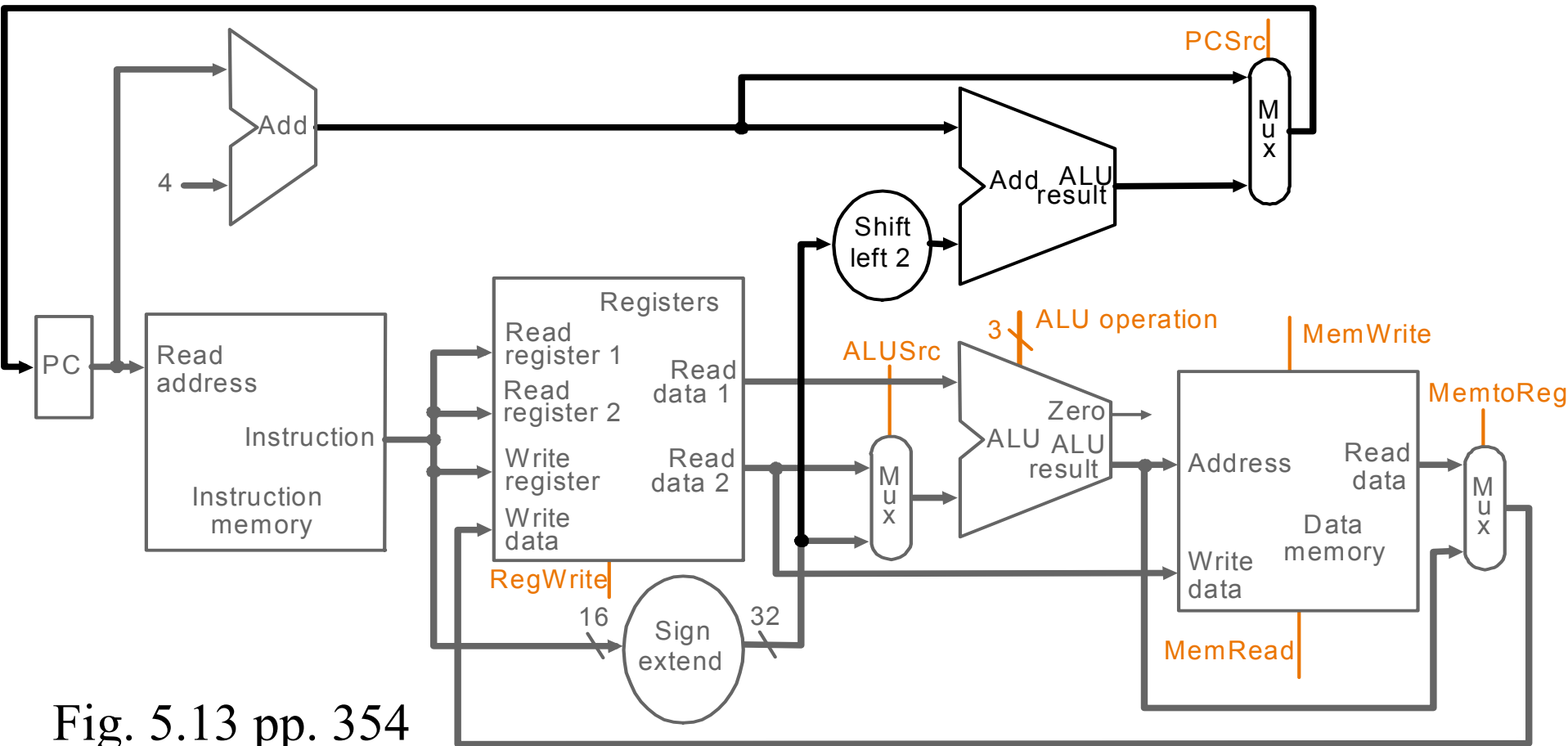
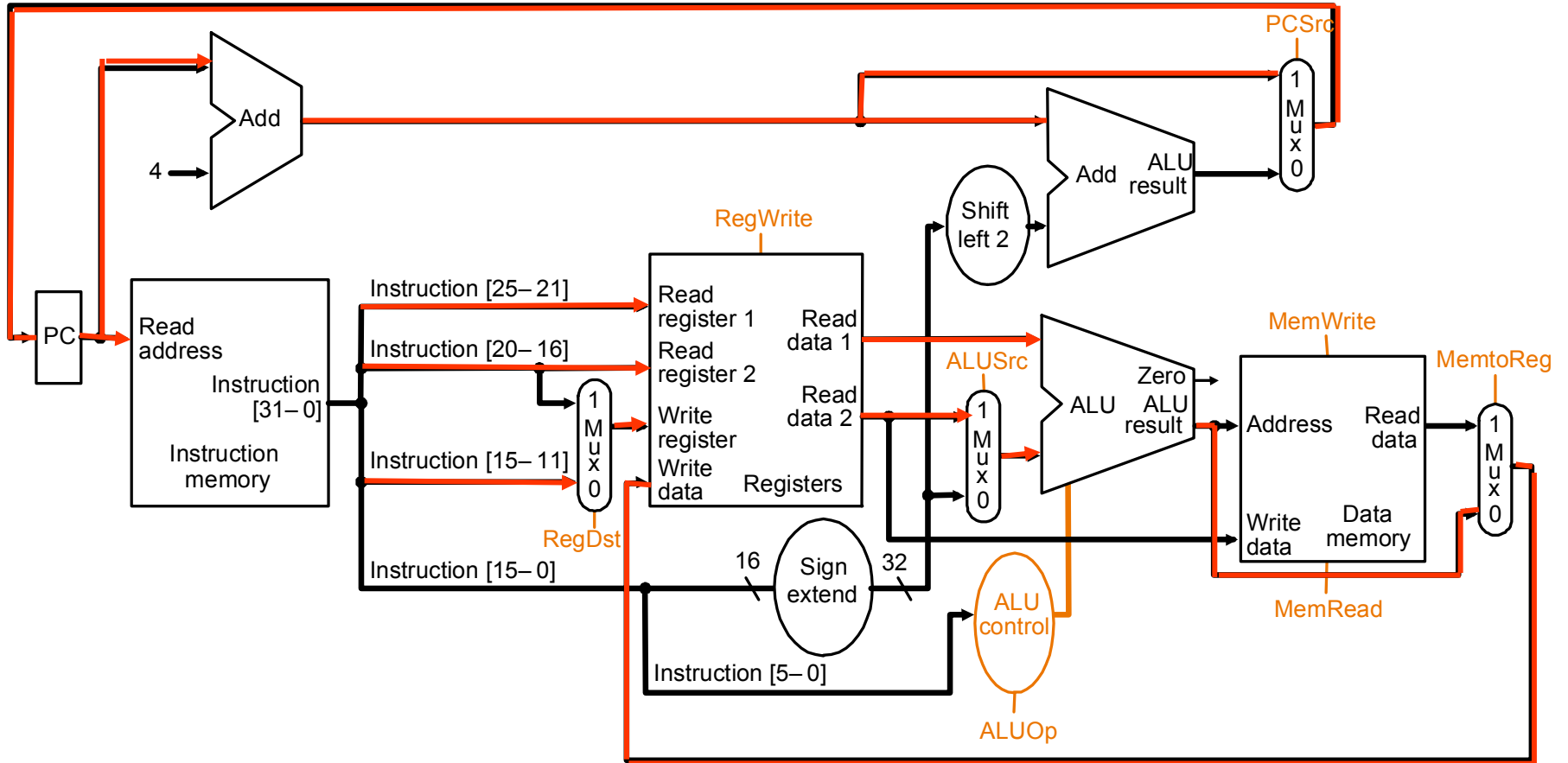
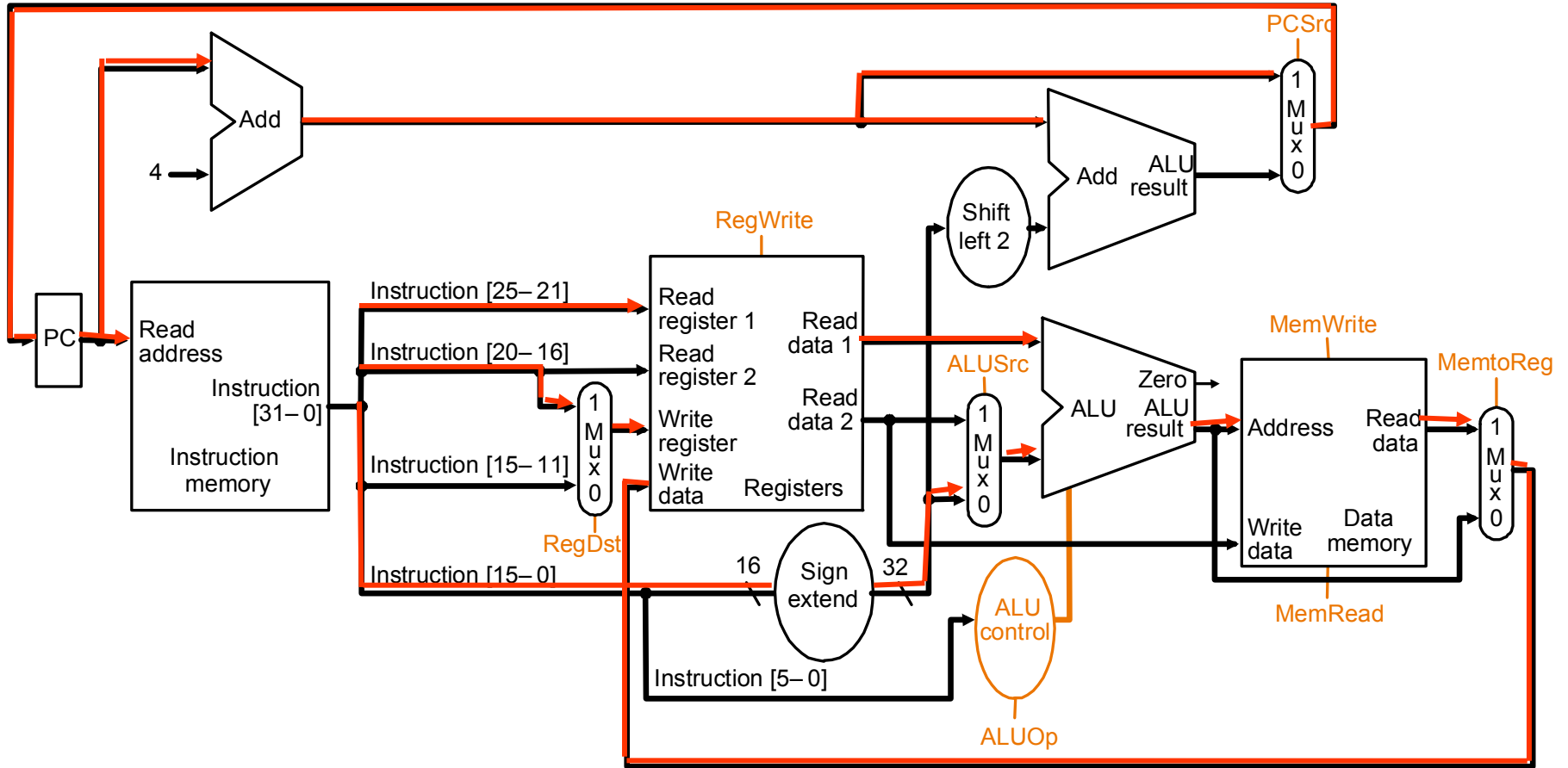


Fig. 5.13 pp. 354

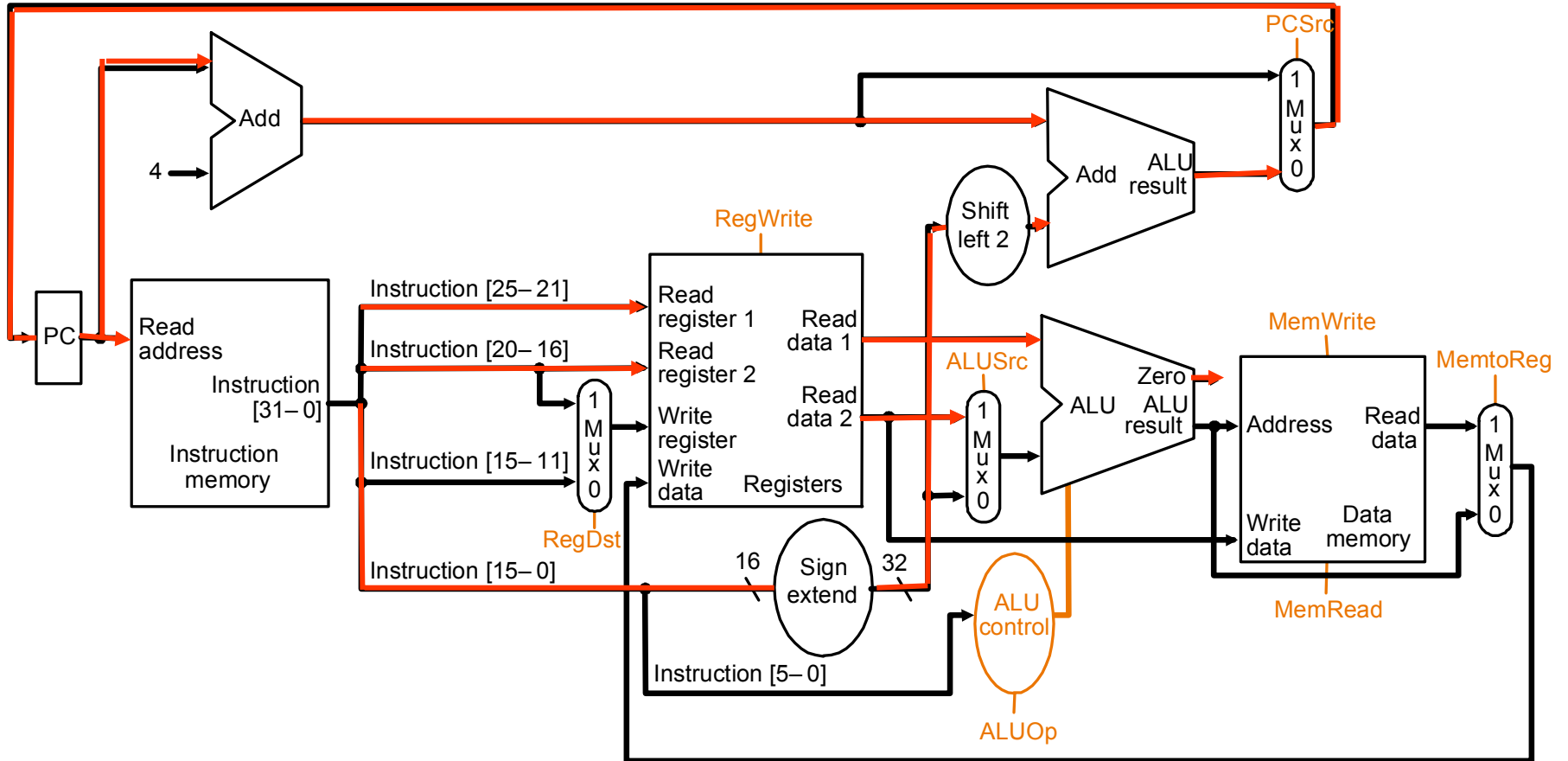
R-type Instructions



Lw Instruction



Branch Instructions



Control

- ❑ Selecting the operations to perform (ALU, read/write, etc.)
- ❑ Controlling the flow of data (multiplexor inputs)
- ❑ Information comes from the 32 bits of the instruction
- ❑ Example:

add \$8, \$17, \$18

Instruction Format:

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

- ❑ ALU's operation based on instruction type and function code

Control

- ❑ e.g., what should the ALU do with this instruction
- ❑ Example: lw \$1, 100(\$2)

35	2	1	100
----	---	---	-----

op	rs	rt	16 bit offset
----	----	----	---------------

- ❑ ALU control input

000	AND
001	OR
010	add
110	subtract
111	set-on-less-than

- ❑ Multi-level control is common: main control unit generates ALUop (load/store, branch, arithmetic) bits, which are then used as input to the ALU control that generates the actual signals (3 bits shown above) to control the ALU unit

Control

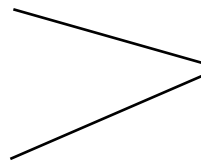
❑ Must describe hardware to compute 3-bit ALU control input

- given instruction type

00 = lw, sw

01 = beq,

10 = arithmetic



ALUOp

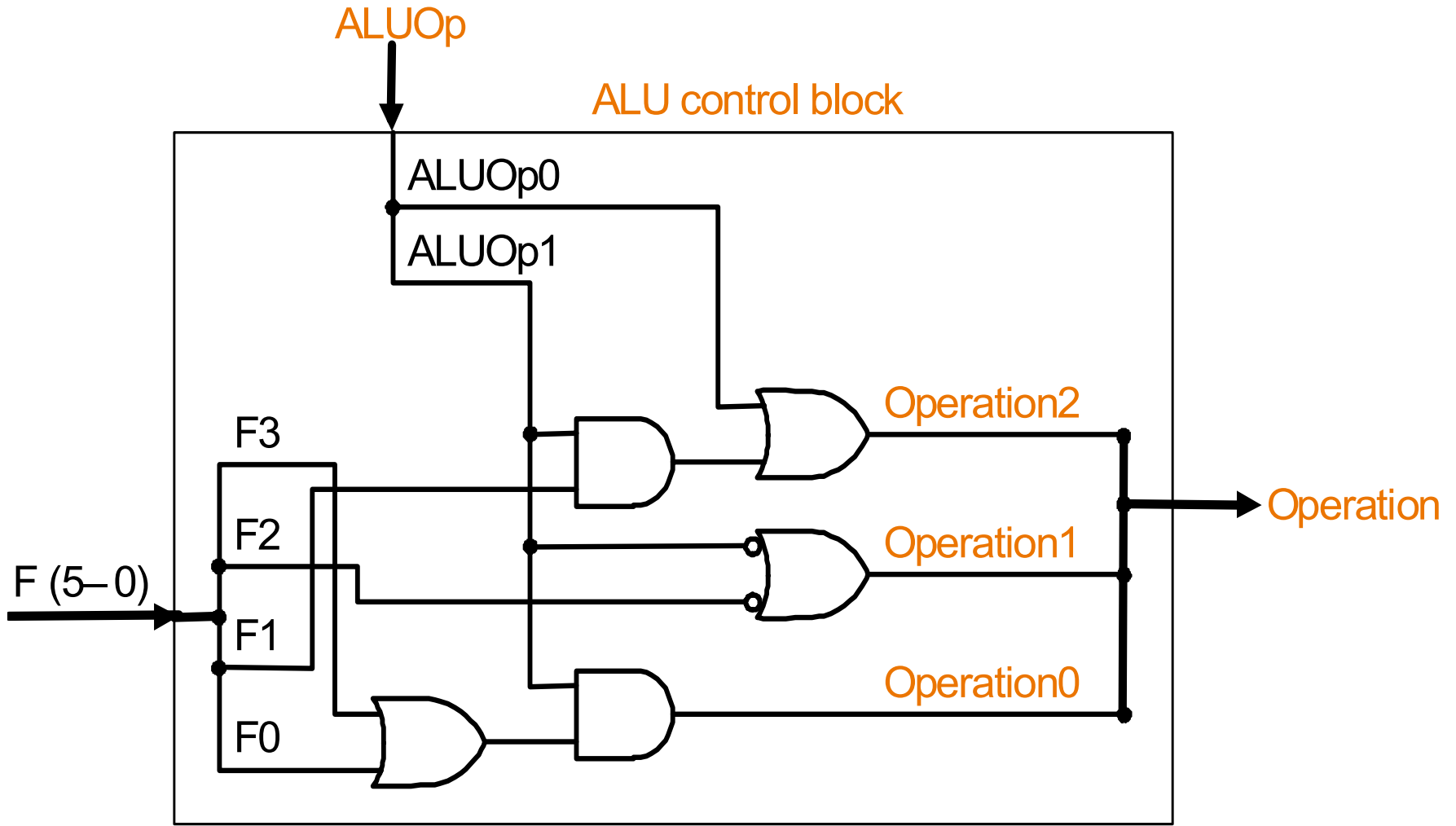
computed from instruction type

- function code for arithmetic

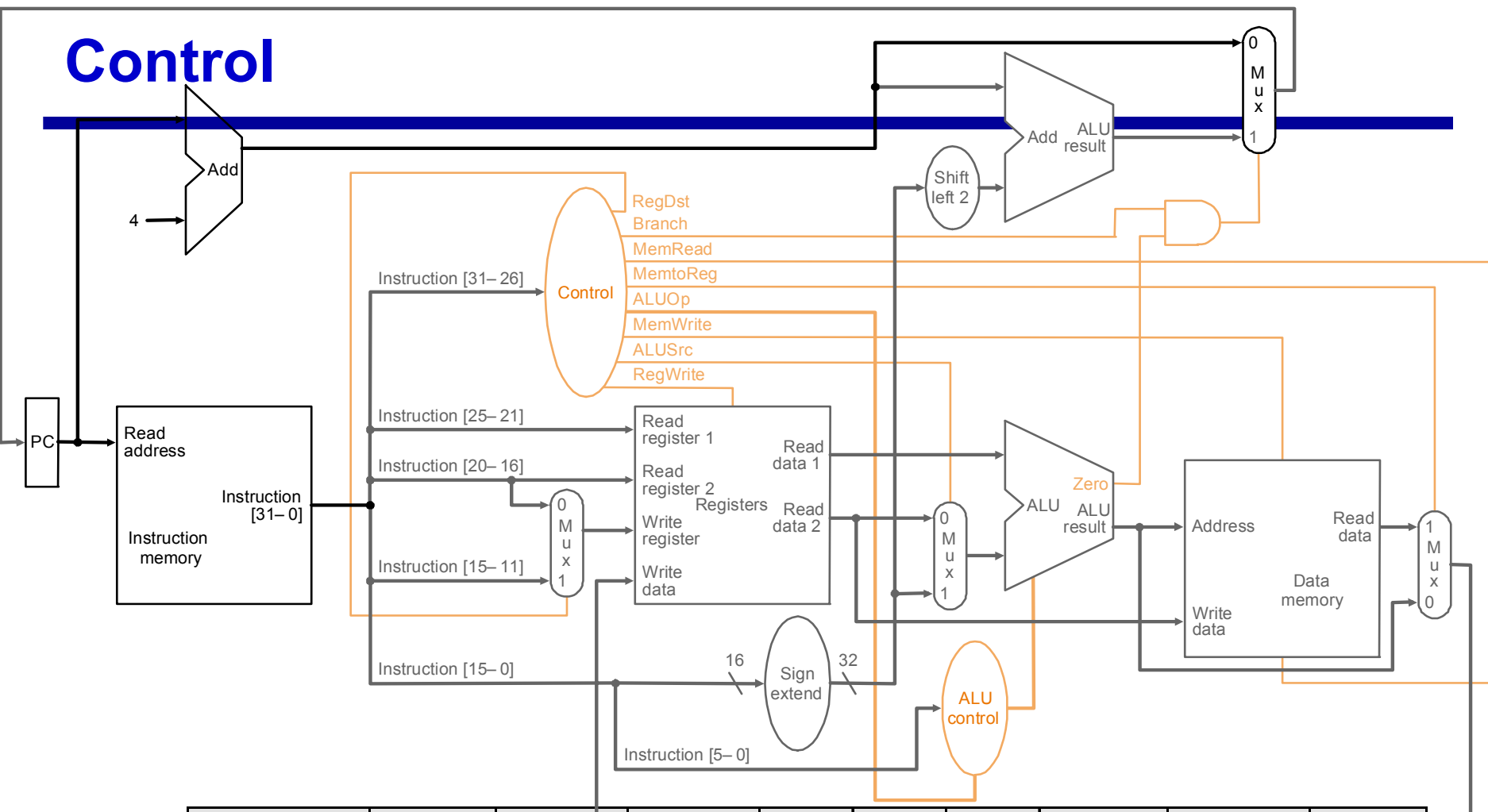
❑ Describe it using a truth table (can turn into gates):

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

ALU Control Schematic Diagram

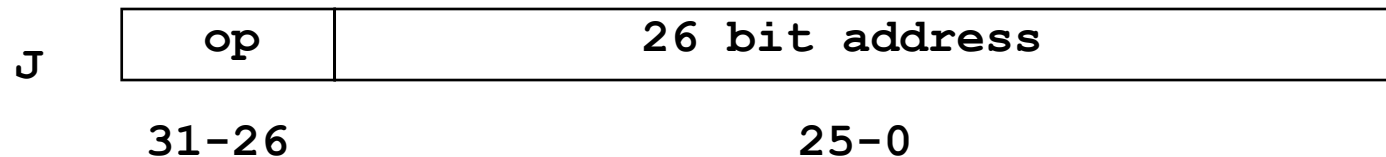
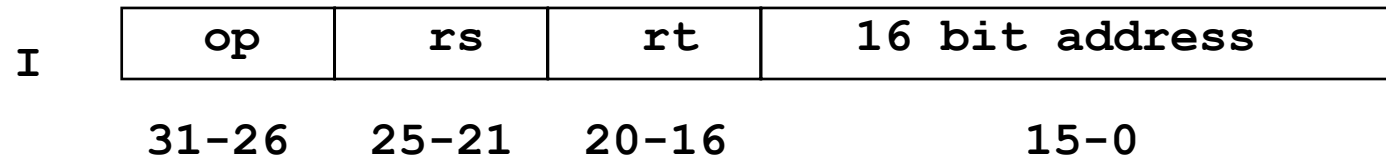
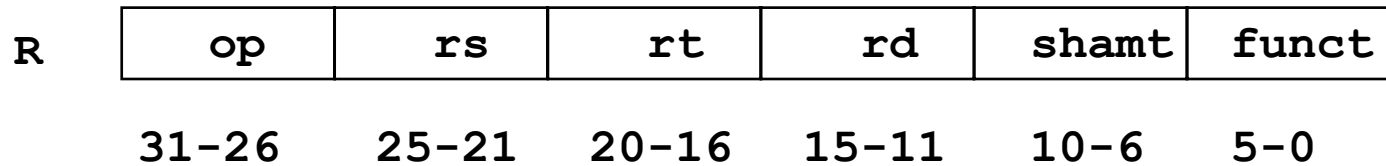


Control



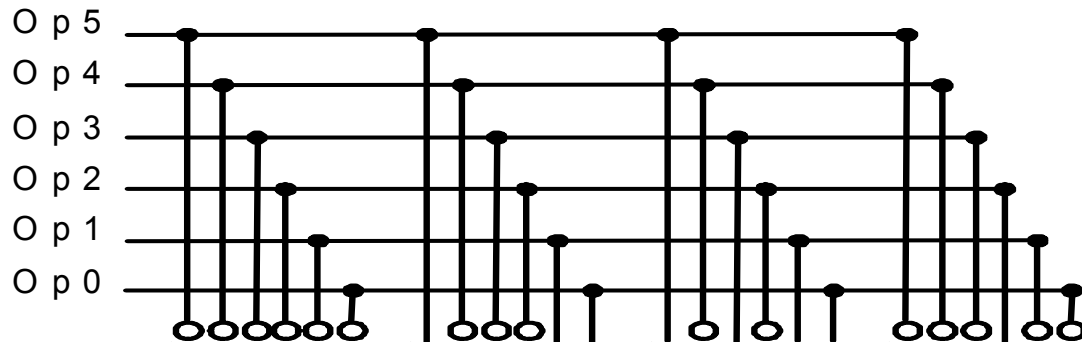
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Recall Instruction Format



Main Control Schematic Diagram

Inputs



R-format

lw

sw

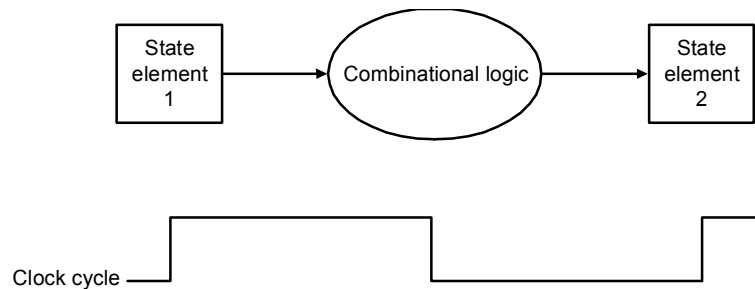
beq

Outputs

- RegDst
- ALUSrc
- MemtoReg
- RegWrite
- MemRead
- MemWrite
- Branch
- ALUOp1
- ALUOp0

Our Simple Control Structure

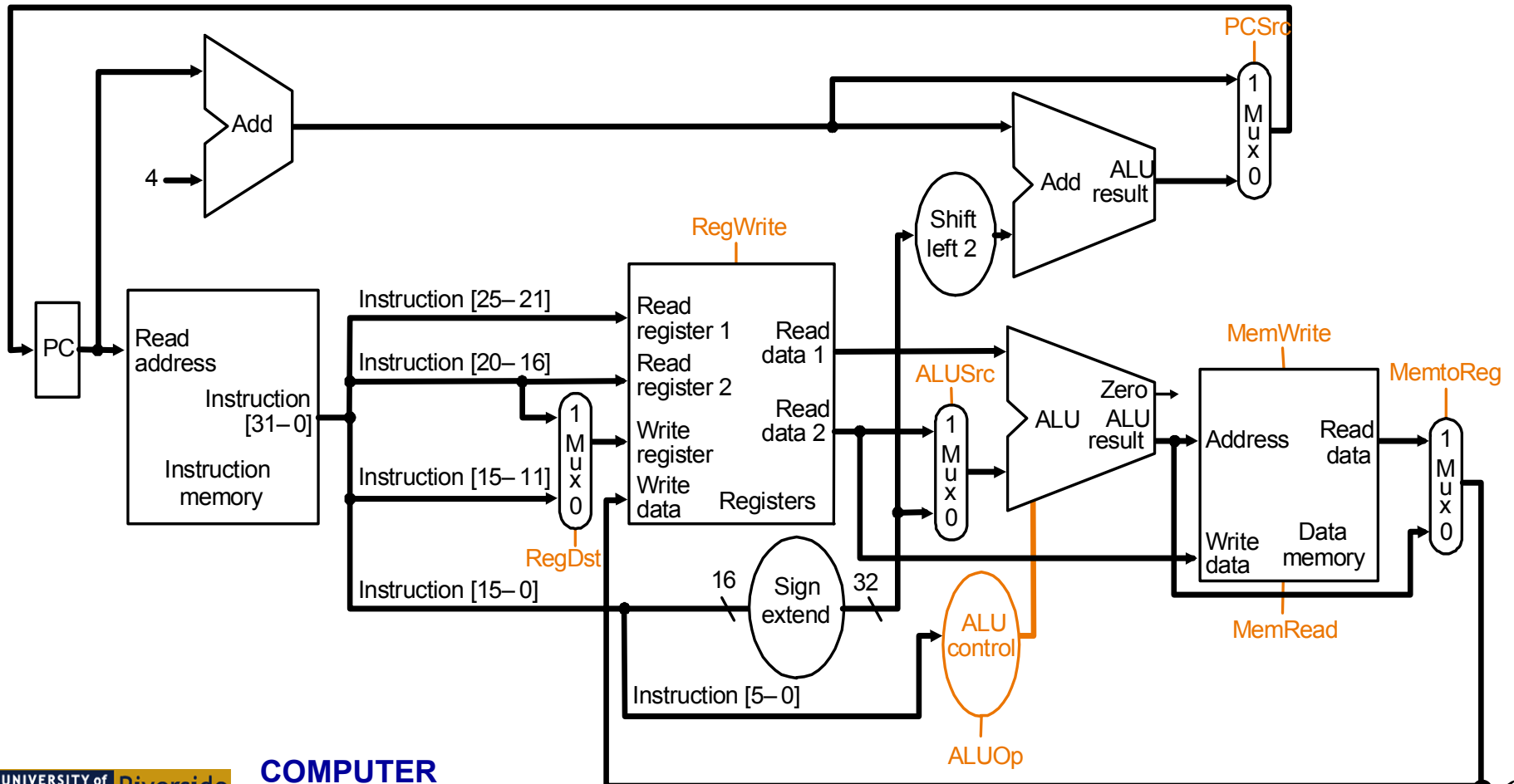
- ❑ All of the logic is combinational
- ❑ We wait for everything to settle down, and the right thing to be done
 - ALU might not produce “right answer” right away
 - we use write signals along with clock to determine when to write
- ❑ Cycle time determined by length of the longest path



We are ignoring some details like setup and hold times

Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
 - memory (2ns), ALU and adders (2ns), register file access (1ns)



Single Cycle – Steps of each instruction

Inst. Type	Functional Units Used				
R-type	Instruction fetch	Register read	ALU	Register write	
Load	Instruction fetch	Register read	ALU	Memory access	Register write
Store	Instruction fetch	Register read	ALU	Memory access	
Branch	Instruction fetch	Register read	ALU		
Jump	Instruction fetch				

Single Cycle – How long is the cycle?

Inst. Type	Inst. Mem.	Reg. File (read)	ALU (s)	Data Mem.	Reg. File (write)	Total	Inst. %
R-type	2	1	2	0	1	6 ns	44
Load	2	1	2	2	1	8 ns	24
Store	2	1	2	2	0	7 ns	12
Branch	2	1	2	0	0	5 ns	18
Jump	2	0	0	0	0	2 ns	2

The cycle time must accommodate the longest operation: *lw*.
 Cycle time \geq 8 ns but the CPI = 1.

If we can accommodate variable number of cycles for each instruction and a cycle time of 1ns.

$$\text{CPI} = 6 \cdot 44\% + 8 \cdot 24\% + 7 \cdot 12\% + 5 \cdot 18\% + 2 \cdot 2\% = 6.3$$

How much faster would this machine be?