

Computer System Architecture - CSEN 601

Module 4: *Pipelining and Execution Techniques* **Lecture 10: *ALU Control & Pipelining***

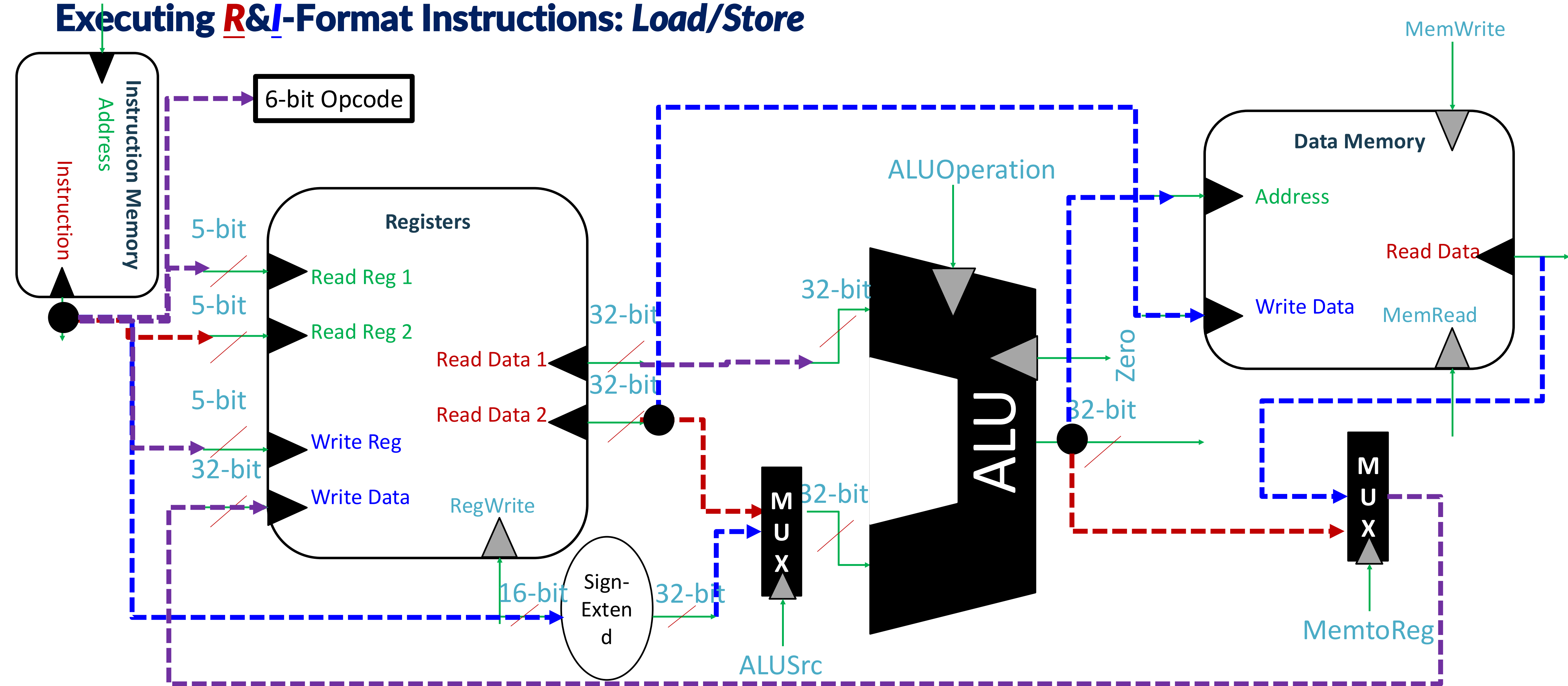
Dr. Eng. Catherine M. Elias

catherine.elias@guc.edu.eg

*Lecturer, Computer Science and Engineering,
Faculty of Media Engineering and Technology, German University in Cairo*

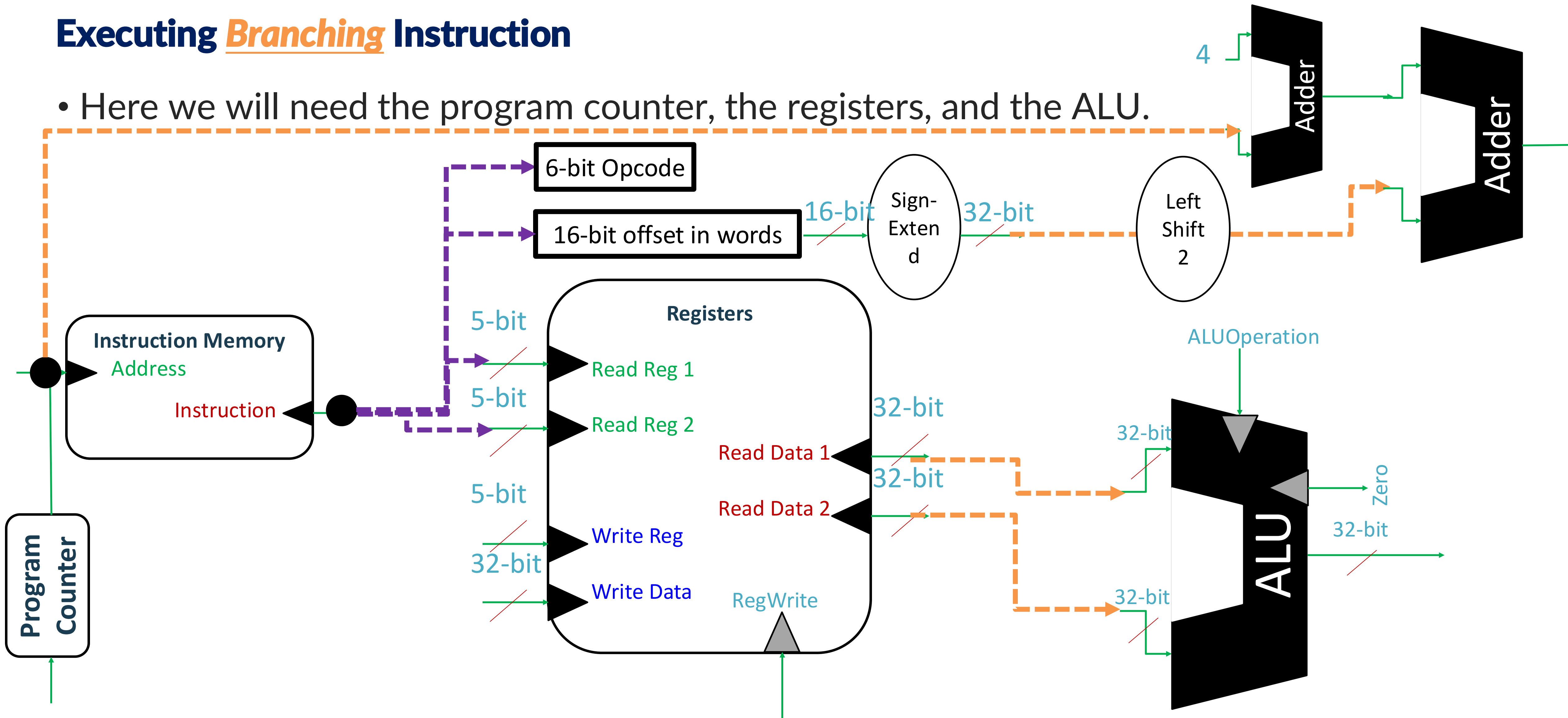
- Let's Recap 😊
- ALU Control Signals
- Updated Datapath (1)
- The Control Signals
- Updated Datapath (2)
- The Single-Cycle Machine Performance
- Pipelining

Executing R&I-Format Instructions: *Load/Store*



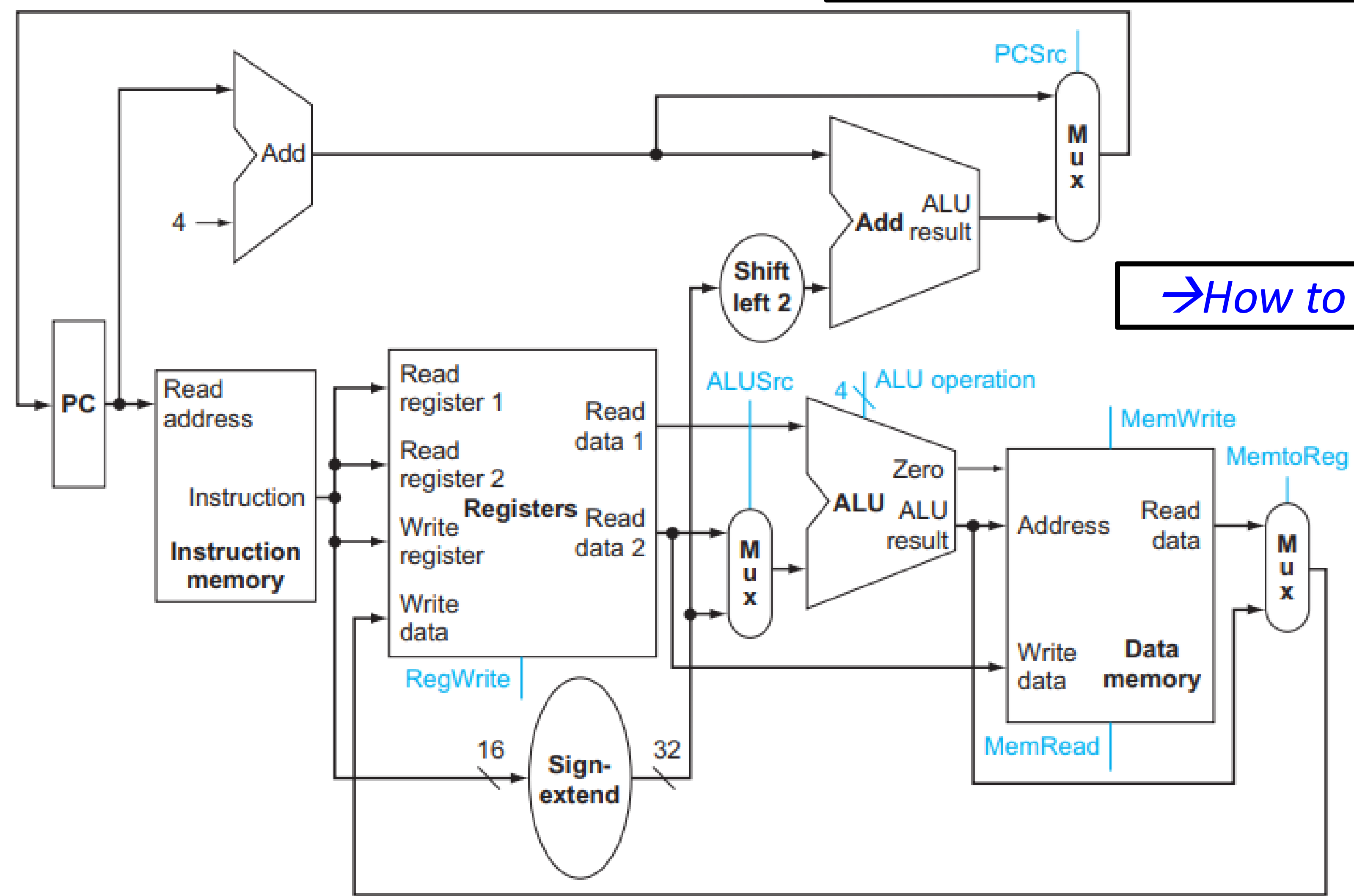
Executing Branching Instruction

- Here we will need the program counter, the registers, and the ALU.



Let's put R/I/Branch/PC increment together

This is called a single-cycle machine

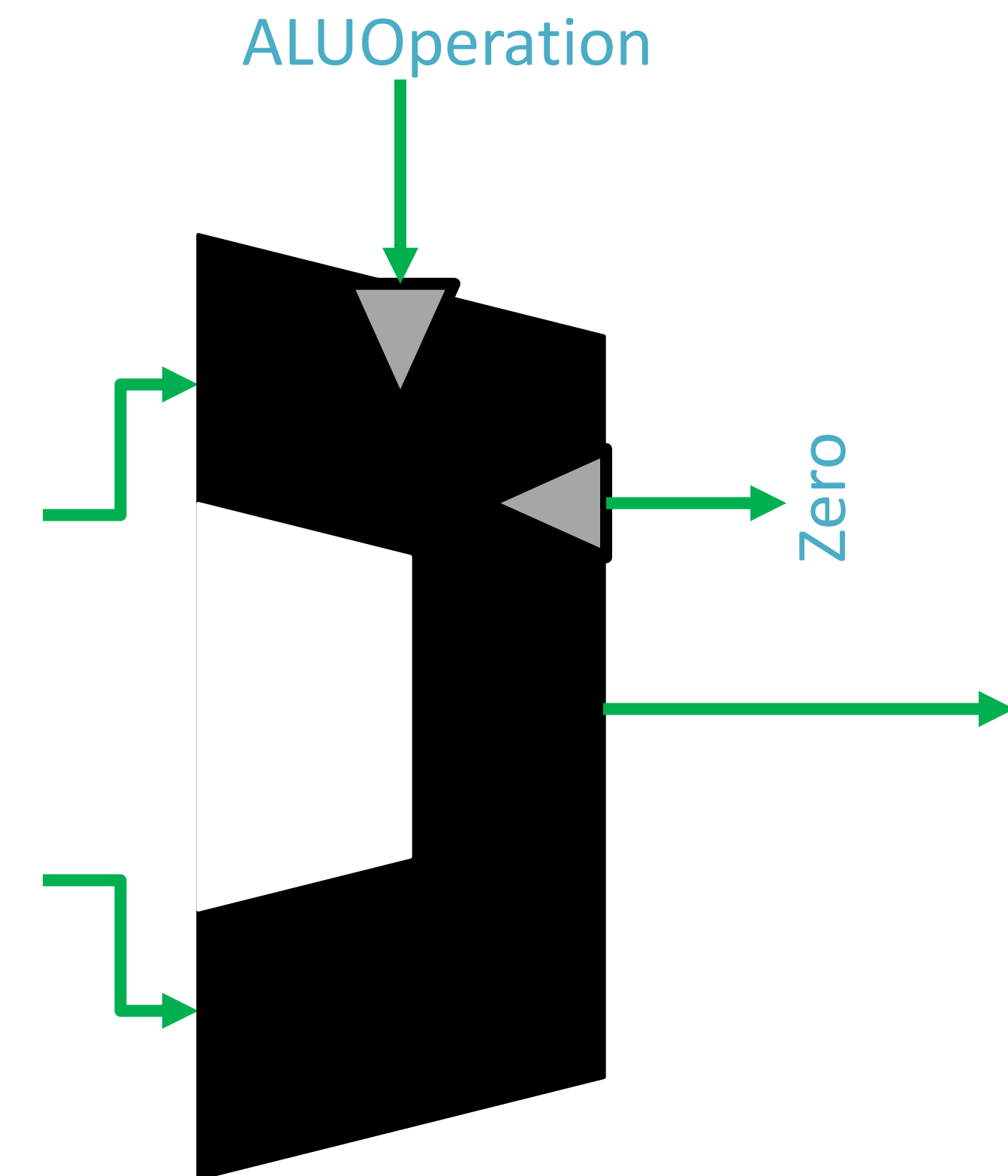


→ How to control the ALU?

The Operations

- The operation to be performed by the ALU is controlled with the ALU operation signal, which will be **4 bits wide**.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



How to control?

• Back to the performed instructions, we have 3 types of instructions that we study:

- The SW/LW
- The Branching
- The R-Type

- Step 1: Check the Opcode Field for the instruction type

→ Step 2: Identify the Instruction type:

 - I-Type (Branch + SW/LW) instructions have unique Opcode
 - All R-Type instructions have 0 Opcode field

→ Step 3: Perform the operation through the ALU Operation Signal

I-Format Instructions

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
beq	I	4	1	2	25			beq \$1, \$2, 100
lw	I	35	2	1	100			lw \$1, 100(\$2)
sw	I	43	2	1	100			sw \$1, 100(\$2)

R-Format Instructions

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add \$1, \$2, \$3
addu	R	0	2	3	1	0	33	addu \$1, \$2, \$3
sub	R	0	2	3	1	0	34	sub \$1, \$2, \$3
subu	R	0	2	3	1	0	35	subu \$1, \$2, \$3
and	R	0	2	3	1	0	36	and \$1, \$2, \$3
or	R	0	2	3	1	0	37	or \$1, \$2, \$3
nor	R	0	2	3	1	0	39	nor \$1, \$2, \$3

The SW/LW

- Add (base + Offset)
 - ALU Control Line = 0010

The Branching

- Subtract
 - ALU Control Line = 0010

The R-Type

- We check the **Funct** field (Least Significant 6 bits in the instruction)

→ How many bits are needed to determine the type of instruction?

→ 2 bits

I-Format Instructions

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
beq	I	4	1	2	25			beq \$1, \$2, 100
lw	I	35	2	1	100			lw \$1, 100(\$2)
sw	I	43	2	1	100			sw \$1, 100(\$2)

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

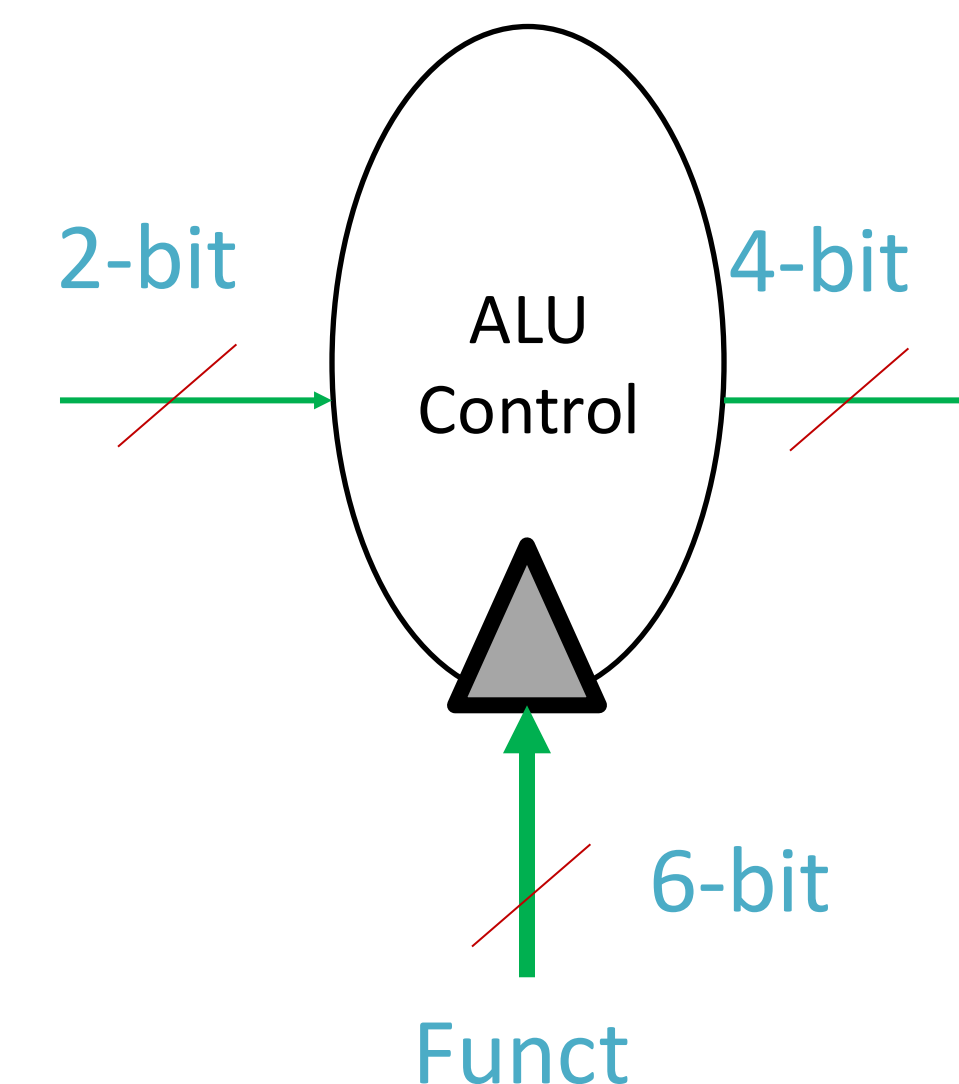
R-Format Instructions

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add \$1, \$2, \$3
addu	R	0	2	3	1	0	33	addu \$1, \$2, \$3
sub	R	0	2	3	1	0	34	sub \$1, \$2, \$3
subu	R	0	2	3	1	0	35	subu \$1, \$2, \$3
and	R	0	2	3	1	0	36	and \$1, \$2, \$3
or	R	0	2	3	1	0	37	or \$1, \$2, \$3
nor	R	0	2	3	1	0	39	nor \$1, \$2, \$3

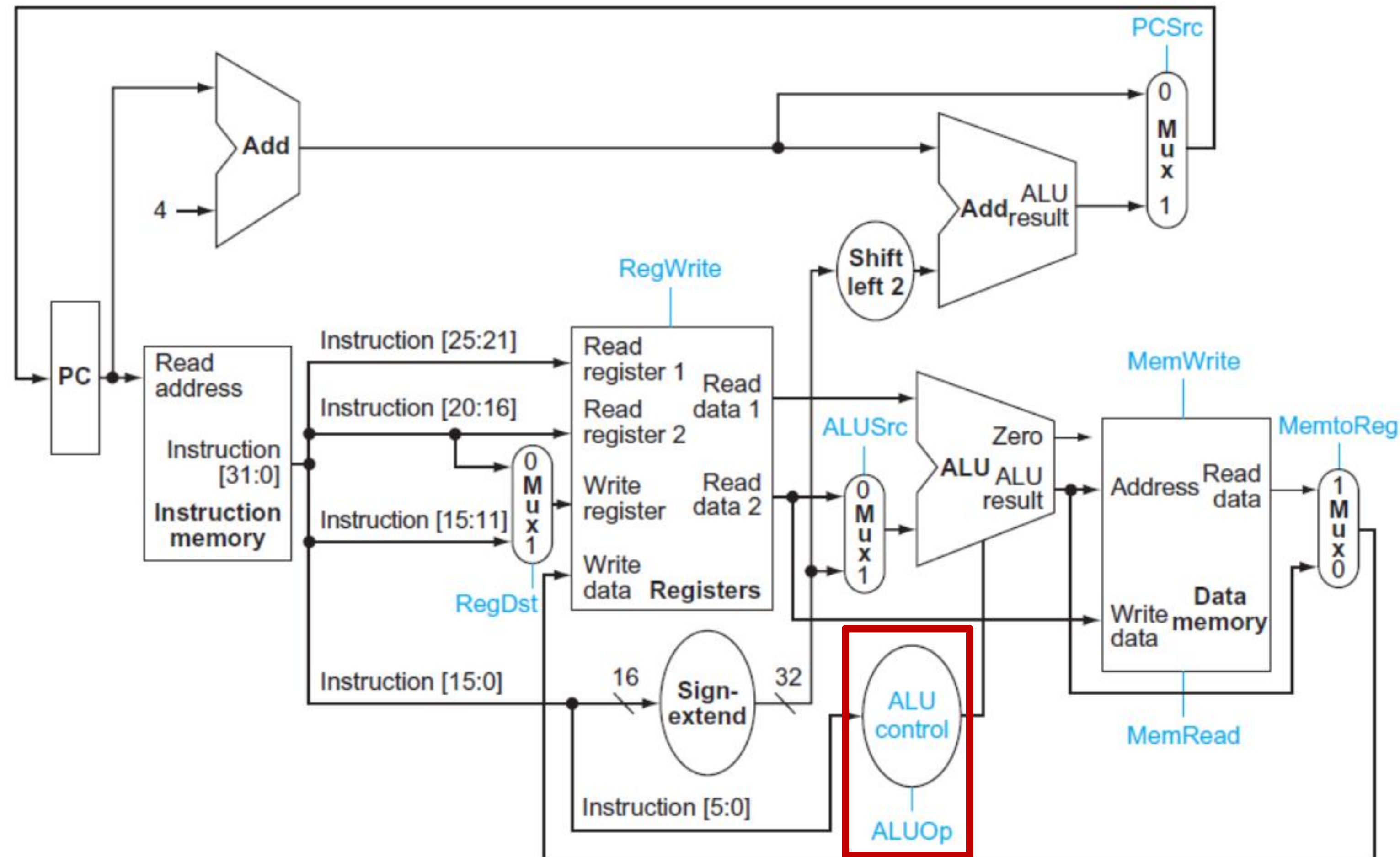
The control Unit

- The ALU Control unit is an element that takes 2 bits through which we determine the operation needed to be performed in the ALU.
- It results in a 4-bit signal that is inputted to the ALU Operation signal to select the operation that will eventually be performed in the ALU.

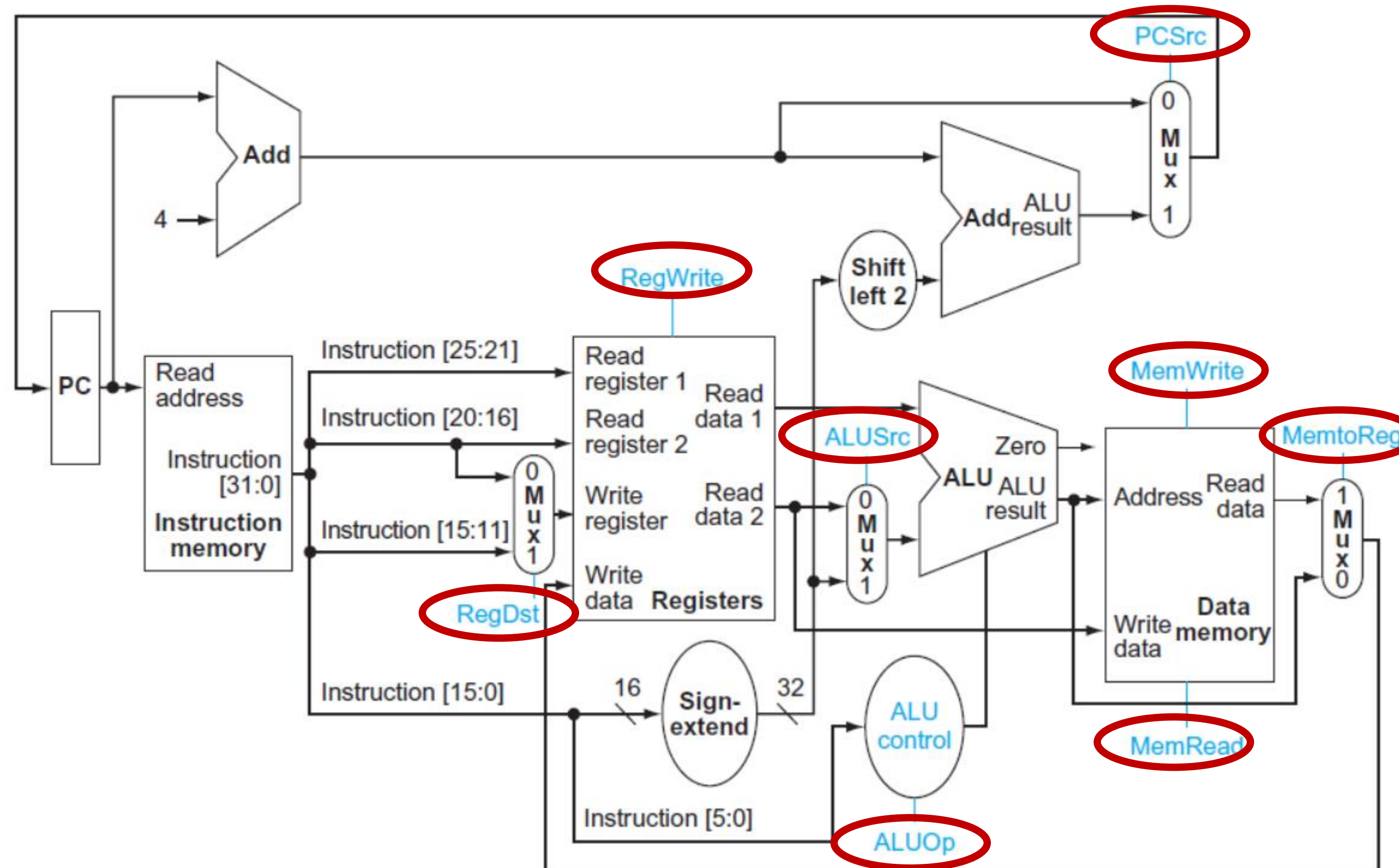
ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111



Datapath + ALU Control



Where are the Control Signals?

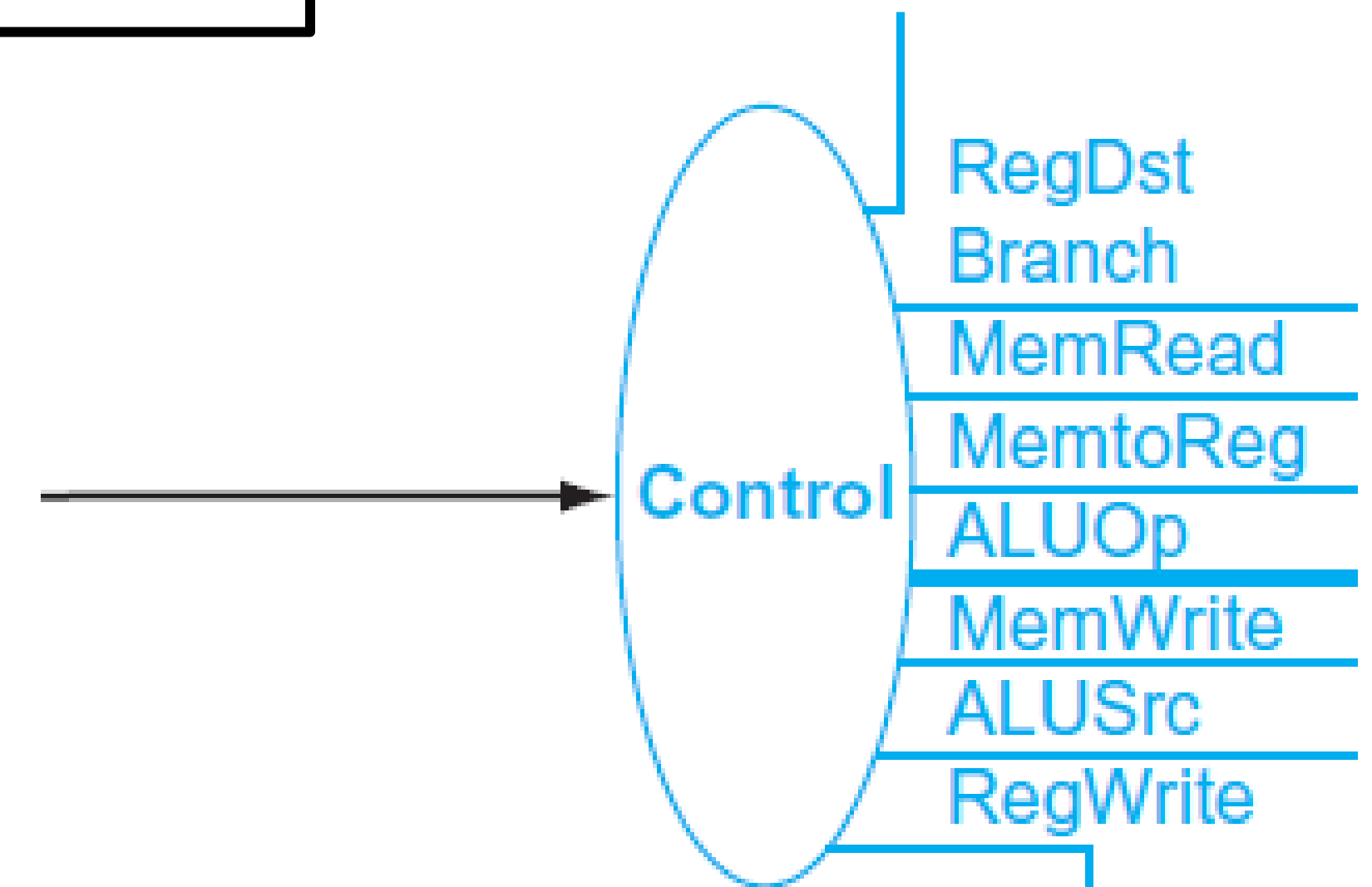


Why do we need them?

- These control signals busses are responsible for coordinating the data flow in the datapath and to avoid any conflicts or clashes between the shared data.
- They are controlled via a control unit that results in the 8 needed busses.

→ How many bits are needed to for each of these control signal busses?
→ Do they all have the same size?

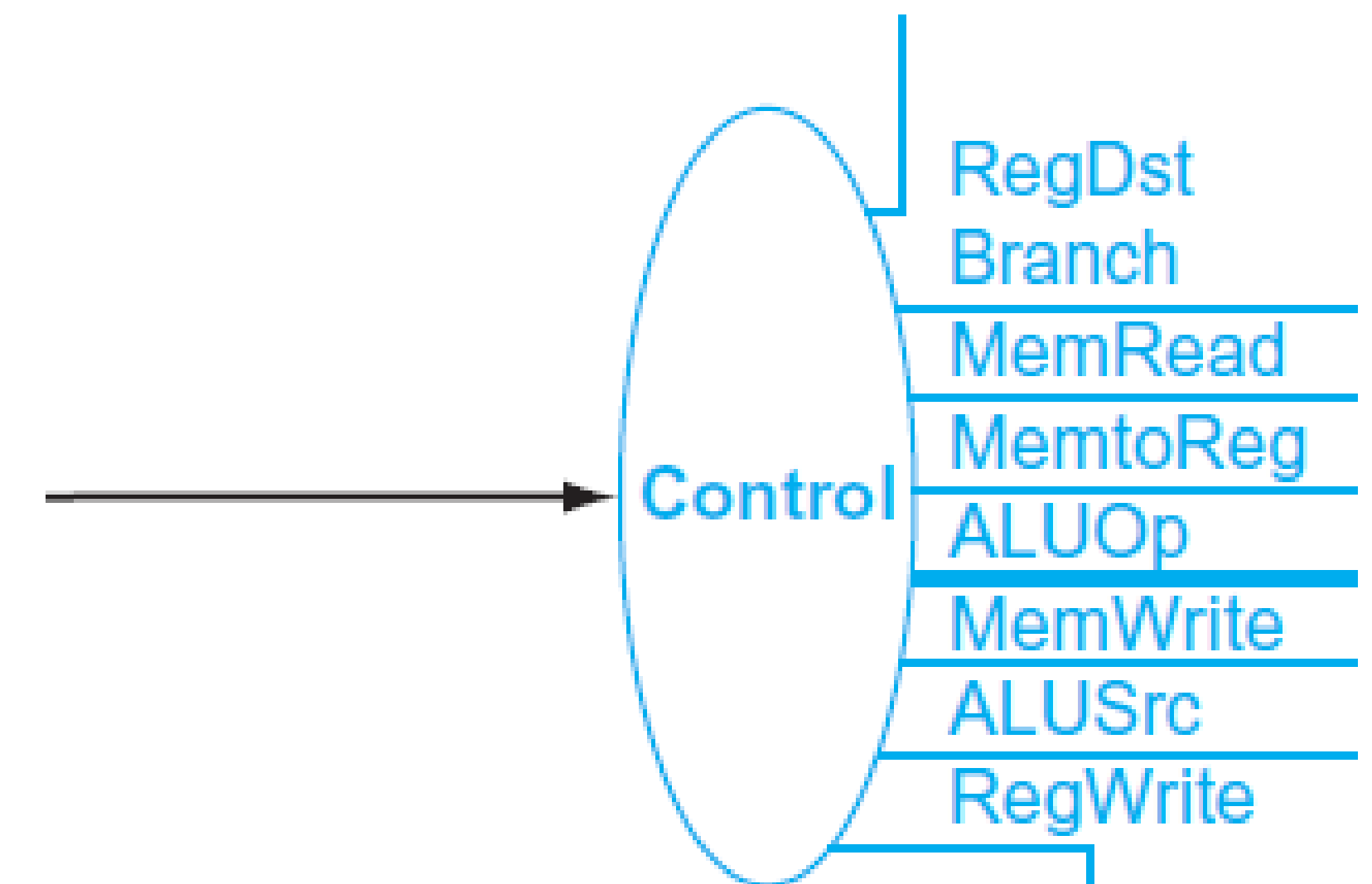
→ 1 bit except for the ALUOp we need 2-bits



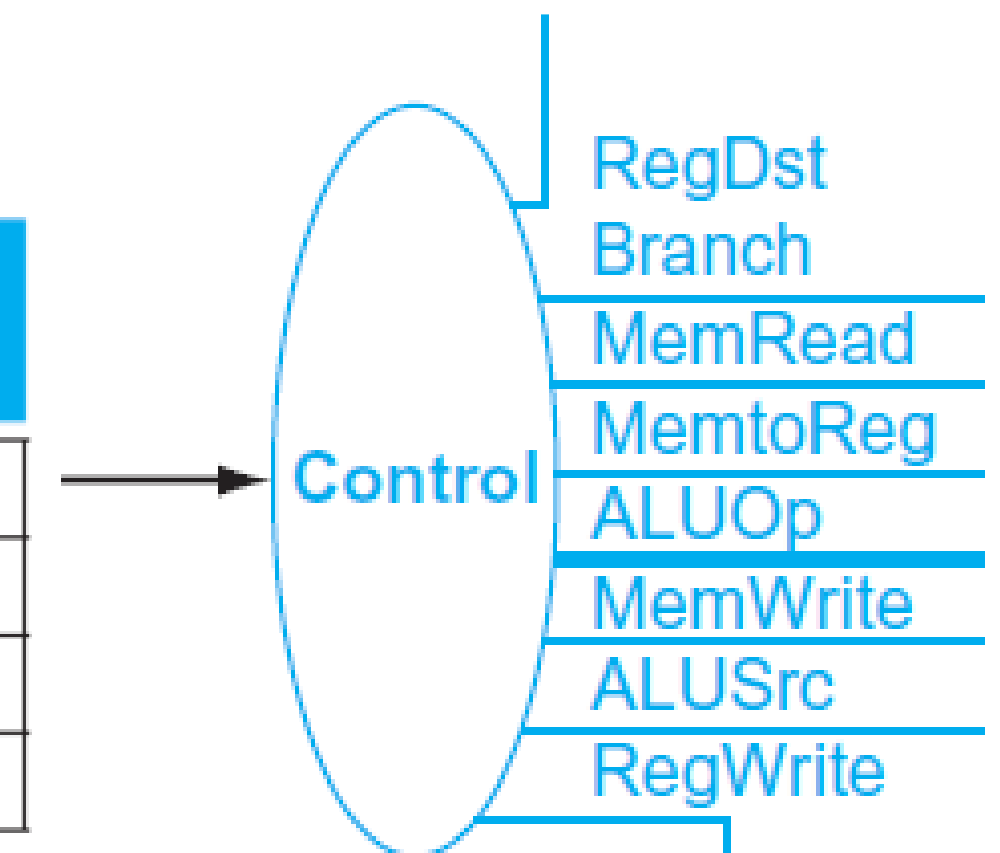
How to Operate?

- When the 1-bit control to a 2-way multiplexor is asserted, the multiplexor selects the input corresponding to 1.
- Otherwise, if the control is deasserted, the multiplexor selects the 0 input.

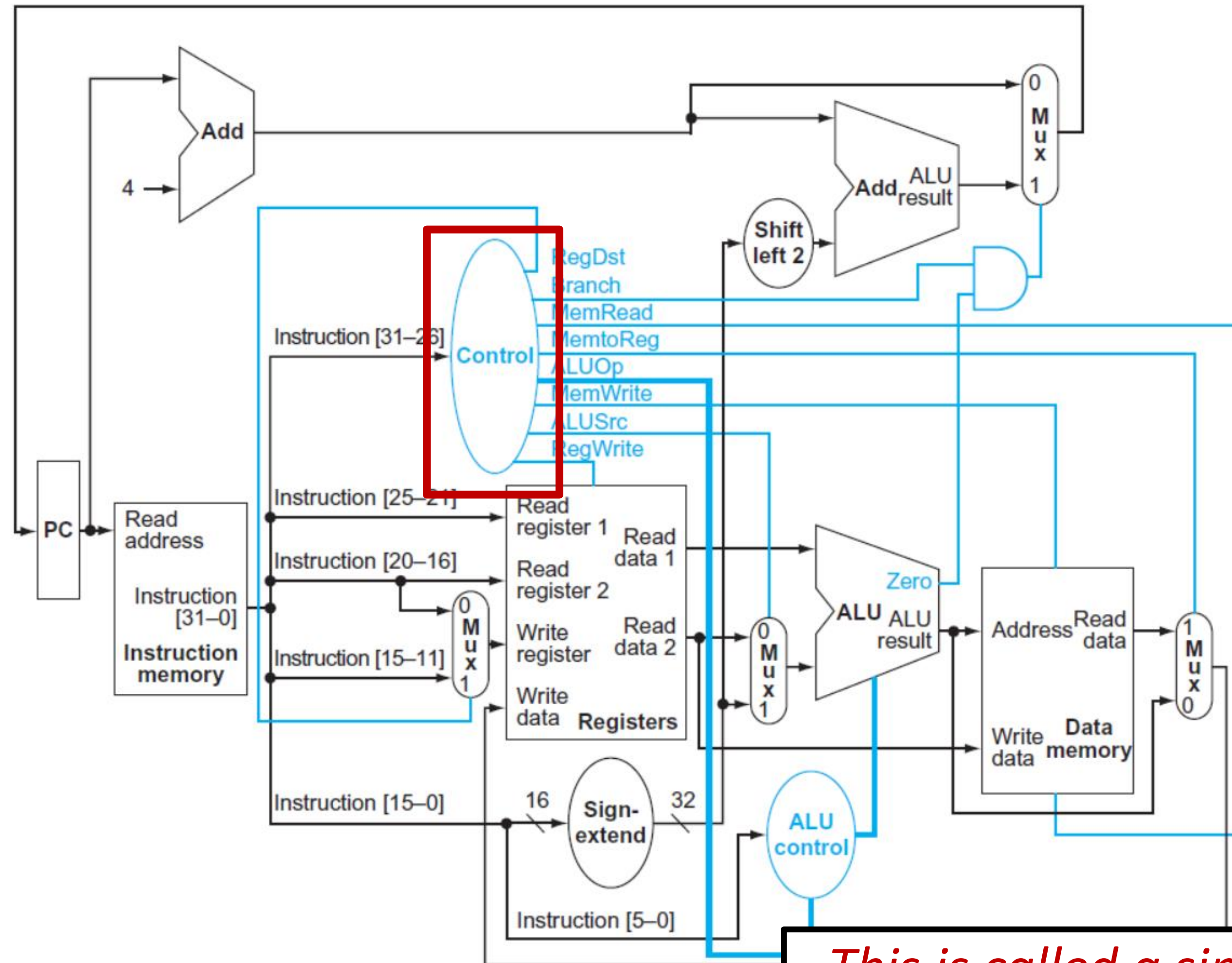
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
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



Datapath + ALU Control + Control Unit

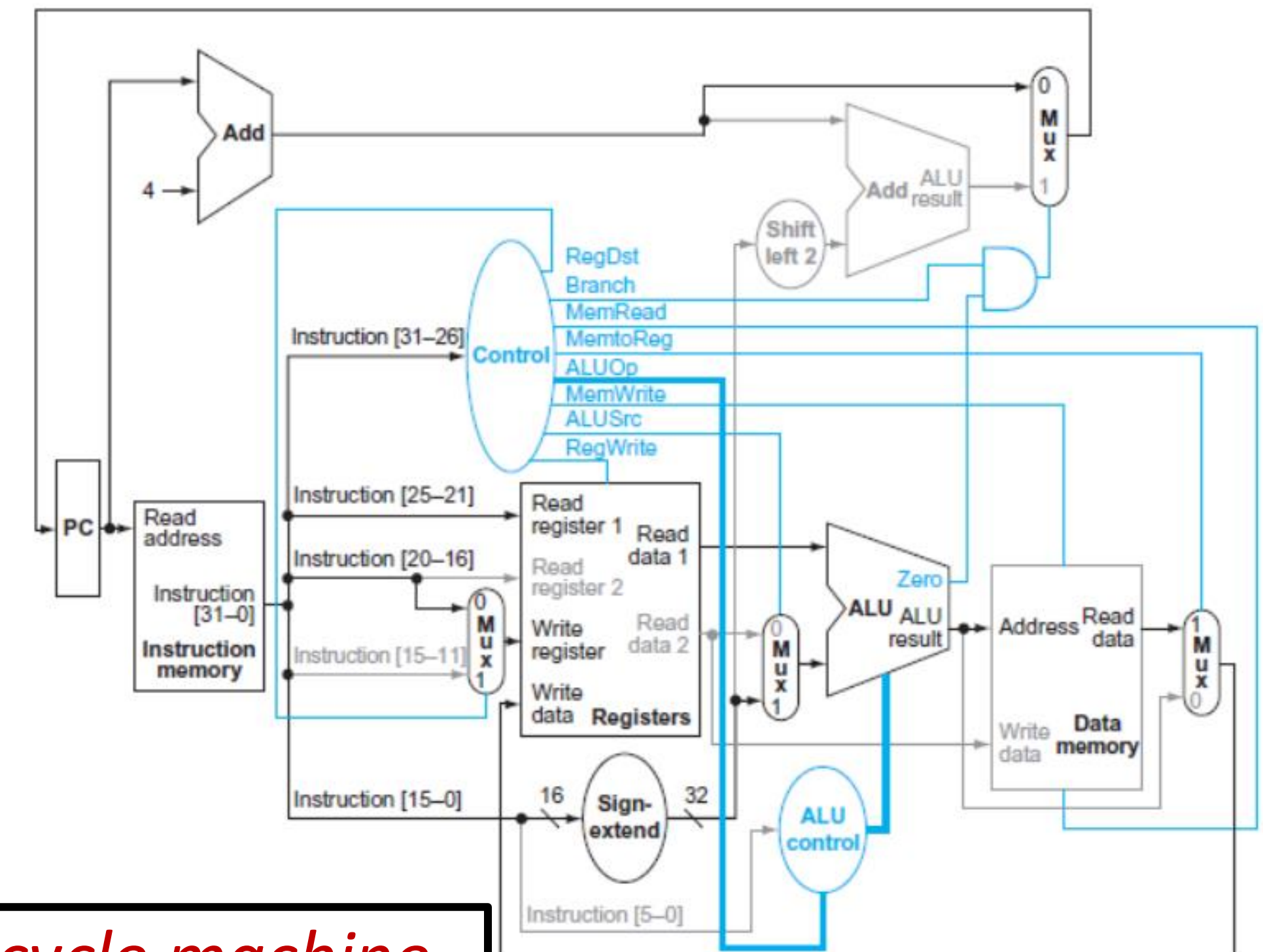


This is called a single-cycle machine

→ Try to trace these busses for each studied instructions ☺ such as:

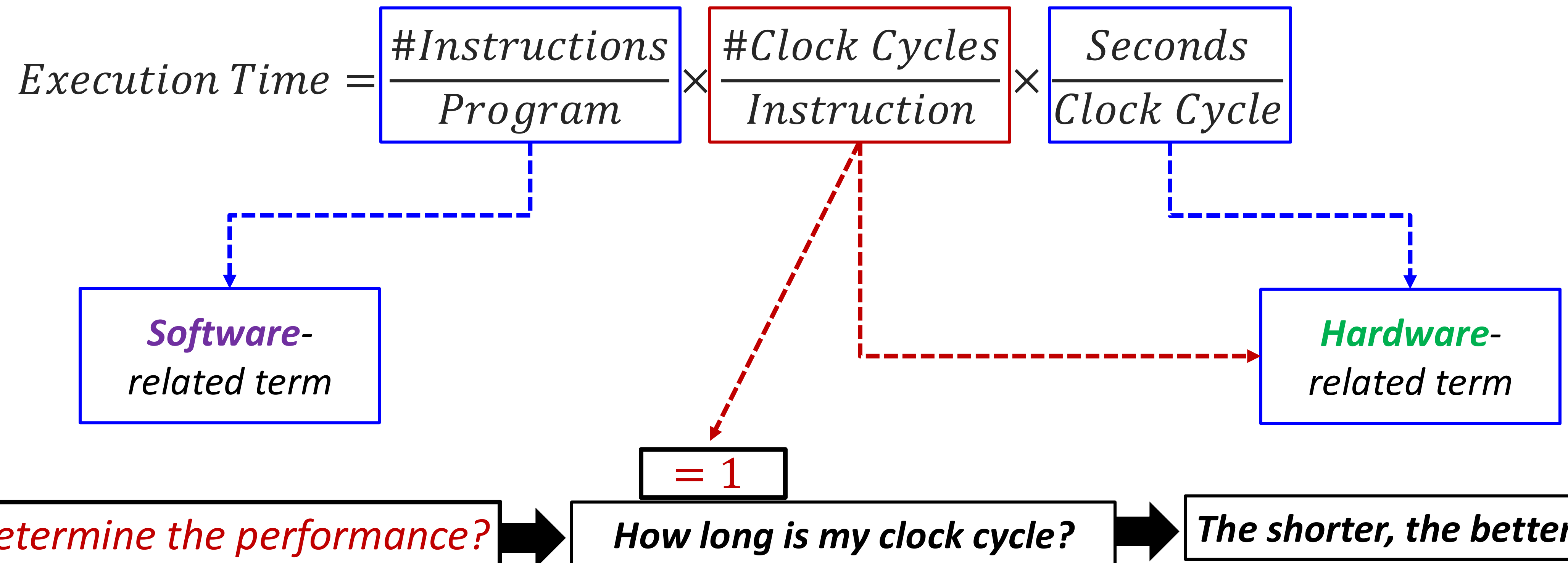
- `add $t1,$t2,$t3`
- `lw $t1, 100($t2)`
- `beq $t1, $t2, 100`

→ Below is an example for which instruction?



How to access the performance our CPU?

- The speed at which all programs and processes run depends on the CPU's performance capacity.
- The **higher** the performance of a CPU, the **faster** the PC will run.

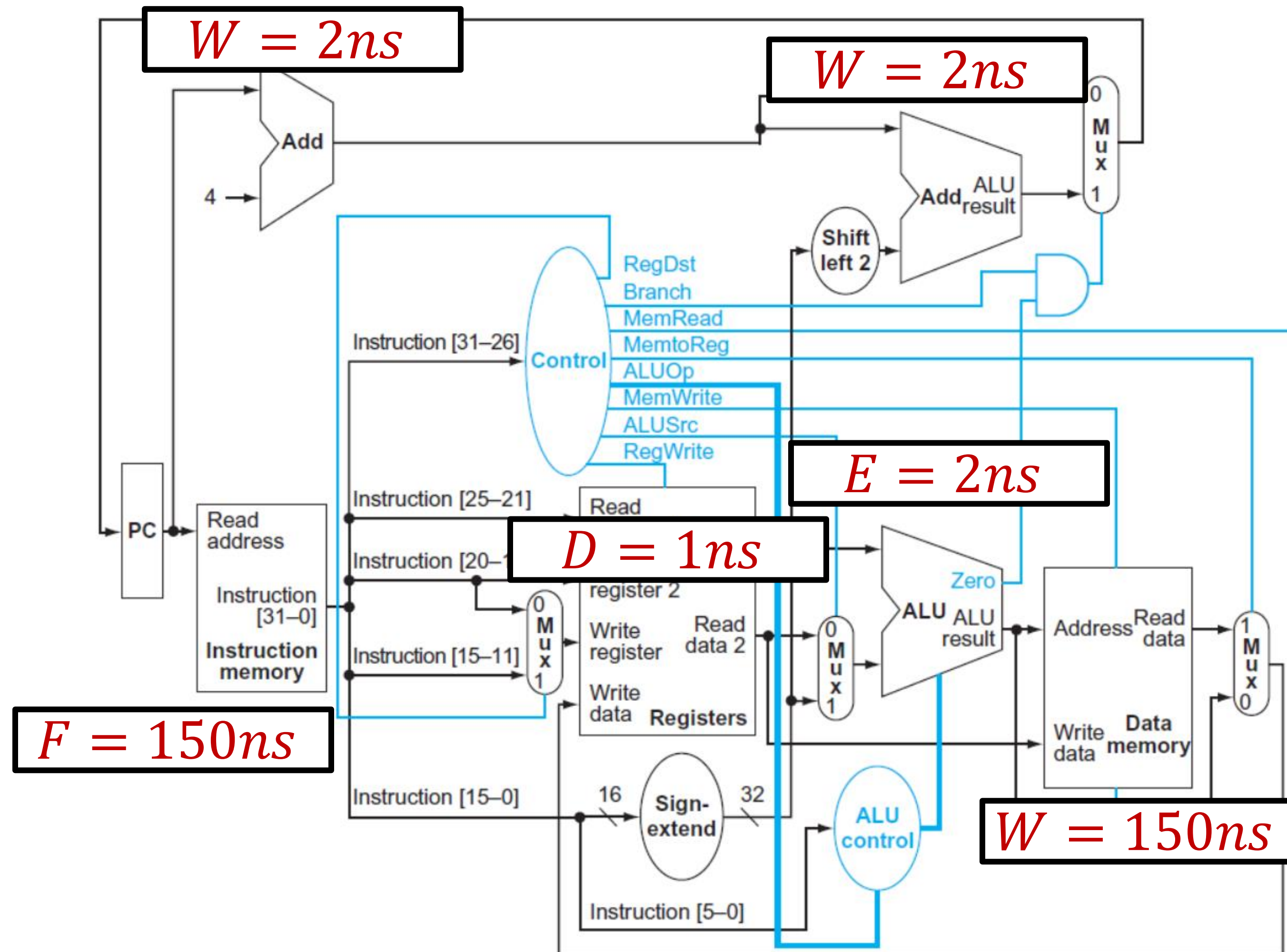


Observations

- Single-cycle design will work **correctly** but **inefficiently**
- Clock cycle must have the **same length for every instruction**
- The **longest possible path** in the processor **determines the clock cycle**
- Which instruction is the **slowest**?

The Single-Cycle Machine Performance

What is the slowest instruction?



→ Try to find the slowest out of the following:

- `add $t1,$t2,$t3`
- `lw $t1, 100($t2)`
- `sw $t1, 100($t2)`
- `beq $t1, $t2, 100`

The load = 304 ns

The clock cycle = 304 ns

The frequency
$$= \frac{1}{304 \times 10^{-9}}$$

= 3MHz

Single-cycle Machine

Stage	CLK 1				CLK 2			
	Instruction 1				Instruction 2			
F								
D								
E								
W								

Pipelining

	Stage	CLK 1				CLK 2			
Instruction 1	F								
	D								
	E								
	W								
Instruction 2	F								
	D								
	E								
	W								

The Analysis

- What are the instruction stages:
 - Fetch
 - Decode (+Read registers)
 - Execute (+ calculate an address)
 - Read memory
 - Write result into a register
- In single cycle: instructions must wait for their turn
- Pipelining: eliminate the wait and overlap instructions!
- Single-Cycle versus Pipelined Performance?

Stage	CLK 1				CLK 2			
	Instruction 1				Instruction 2			
F								
D								
E								
W								

	Stage	CLK 1				CLK 2			
Instruction 1	F								
	D								
	E								
	W								
Instruction 2	F								
	D								
	E								
	W								

The Limitation

- Consider the following code, and try to pipeline it:

- sub \$s1, \$t1, \$t2
- add \$t0, \$s1, \$s0

Data Hazards


- Consider the following code, and try to pipeline it:

- bne \$s3, \$s4, else
- addi \$t0, \$zero, 1
- ...
- else: addi \$s1, \$zero, 2

Control Hazards

- What if instruction fetching and data reading were not 'separated'?

Structure Hazards

For Further Inquiries, Please
 ***send an email***

Catherine.elias@guc.edu.eg,
Catherine.elias@ieee.org

Thank you for your attention!

See you next time 😊