CSE261: Computer Architecture

10. Processor (3): Multicycle Implementation

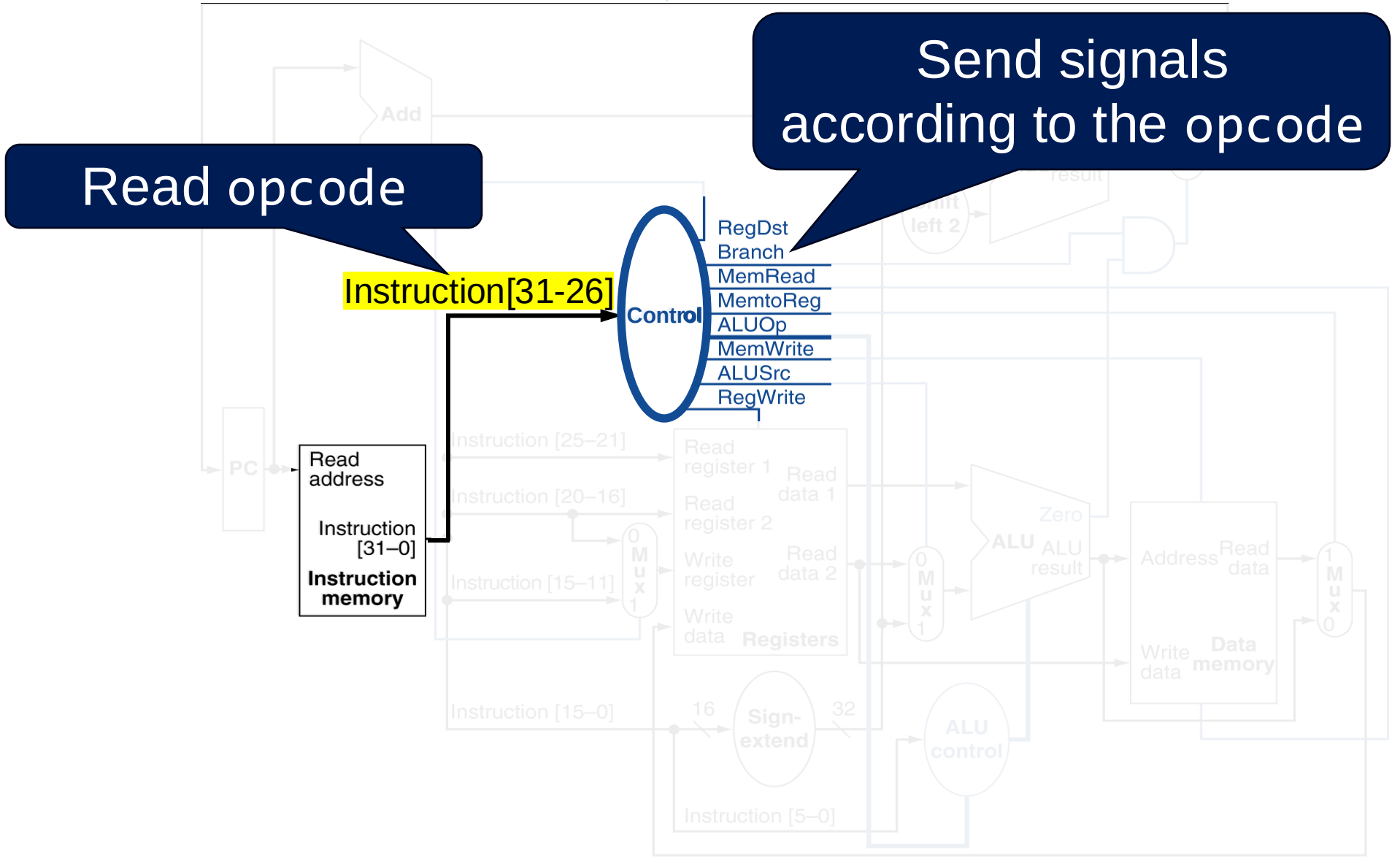
Seongil Wi

Department of Computer Science and Engineering

HW2 will be Released Soon!

Implementing single-cycle datapath and control

Recap: Datapath with Control



Recap: Introduction to ALU Control

- The ALU operation signals must be provided differently based on the instruction

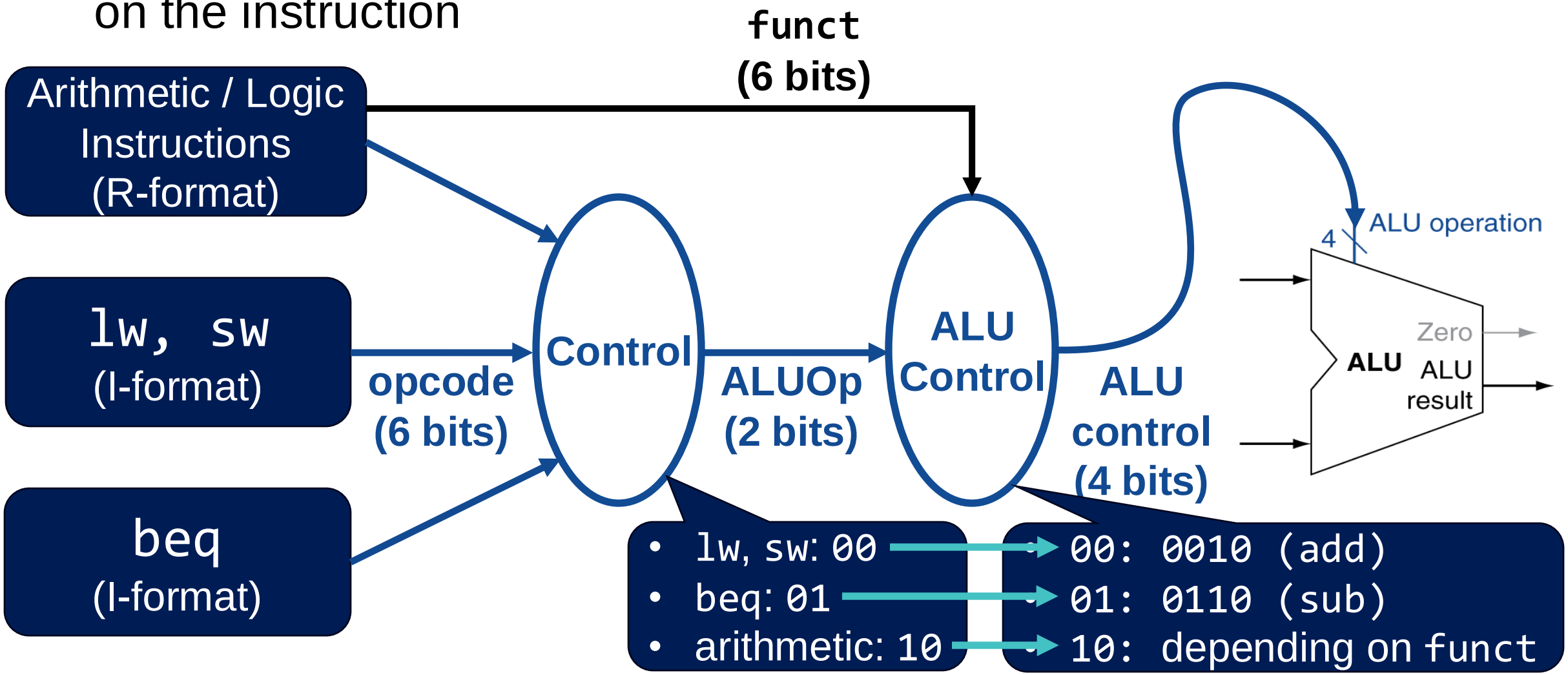
Arithmetic / Logic Instructions (R-format) → Depending on the instruction

`lw, sw` (I-format) → Addition

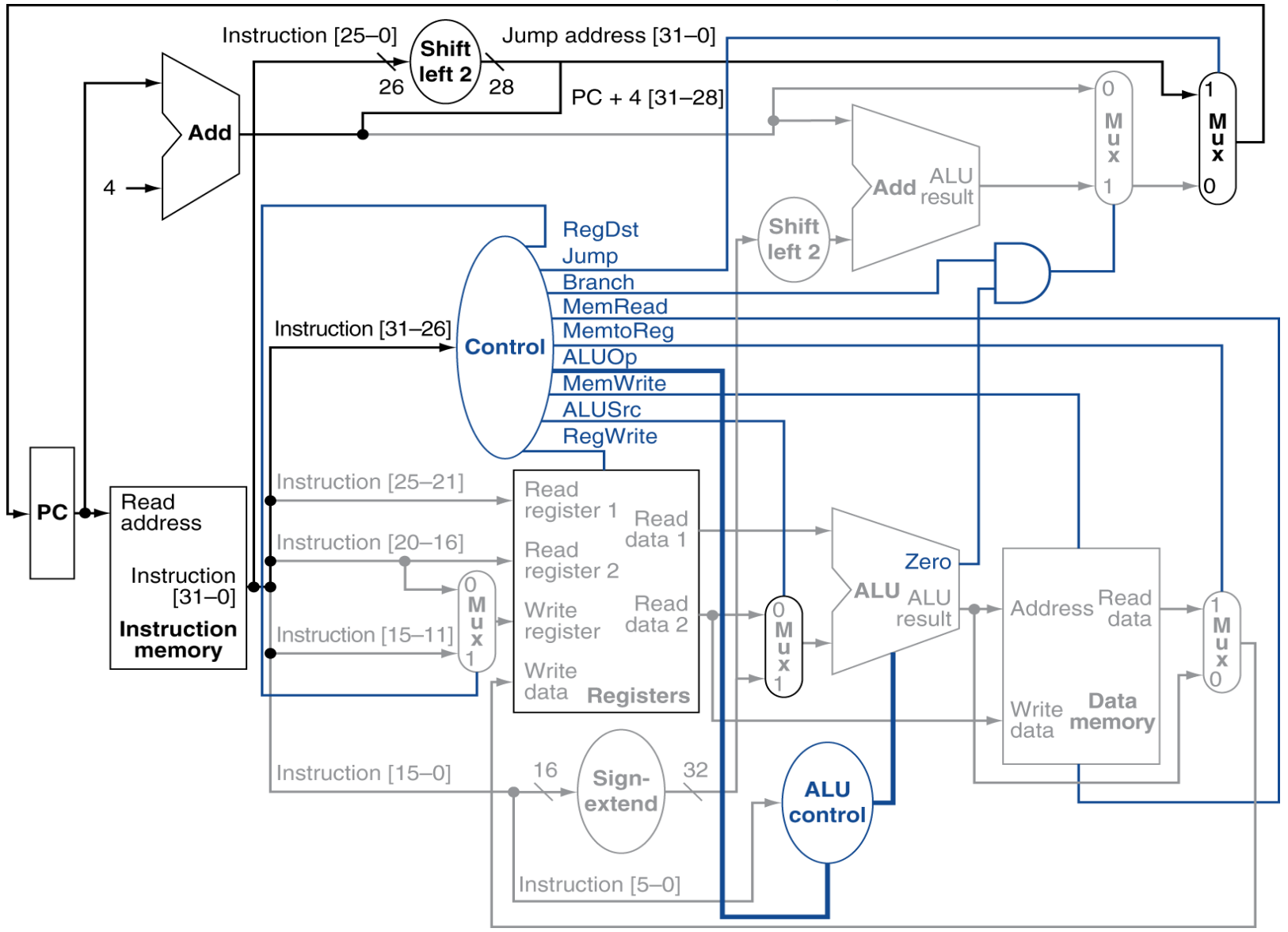
`beq` (I-format) → Subtraction

Recap: Introduction to ALU Control

- The ALU operation signals must be provided differently based on the instruction



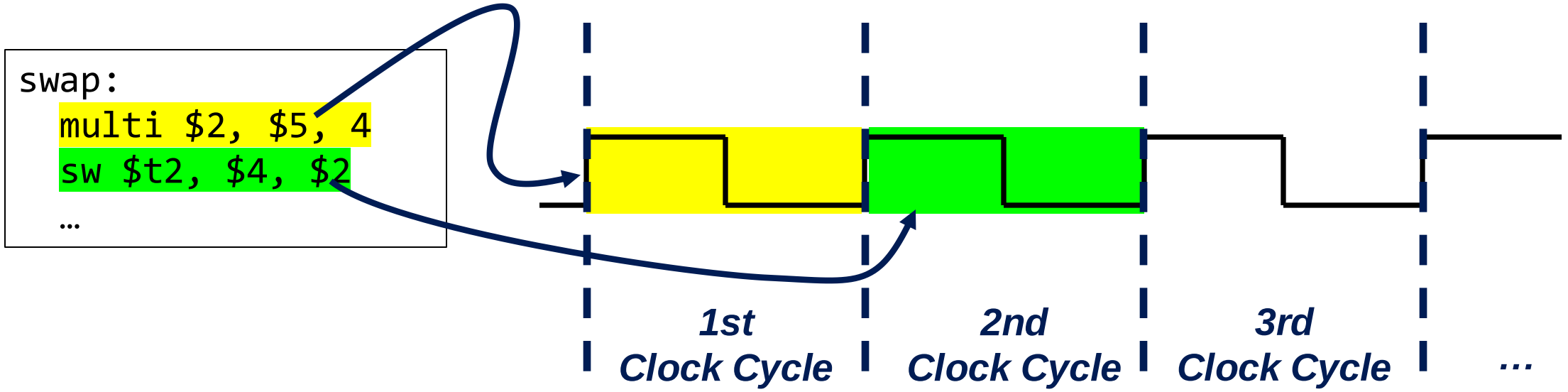
Recap: Single Cycle Implementation



Recap: Single Cycle Datapath

We have considered a ***single clock cycle datapath***

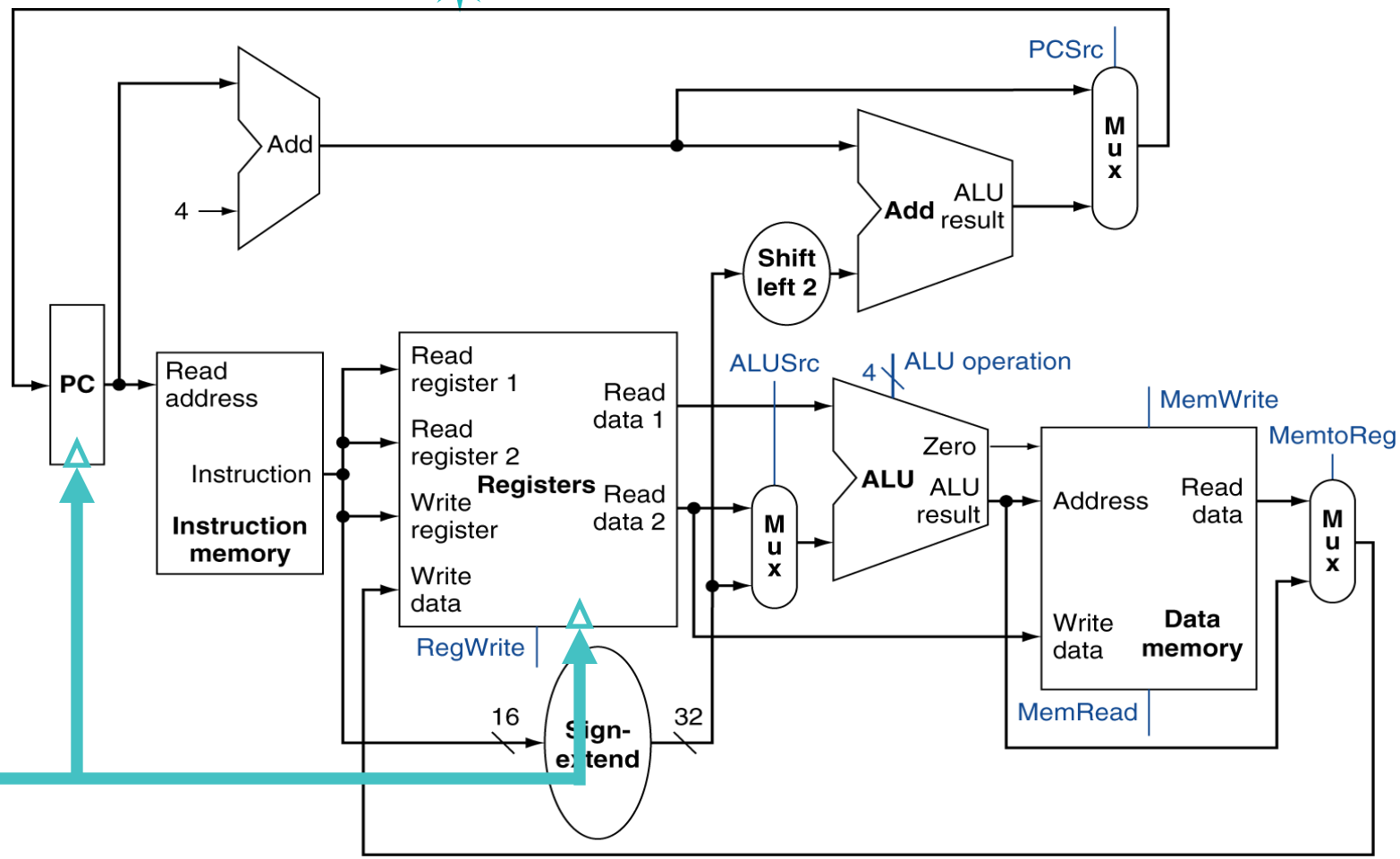
- Each instruction is executed in one clock cycle in the CPU



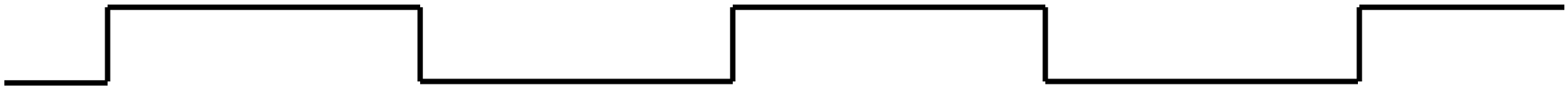
Let's look at the detailed scenario

Recap: Single Cycle Datapath

```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```

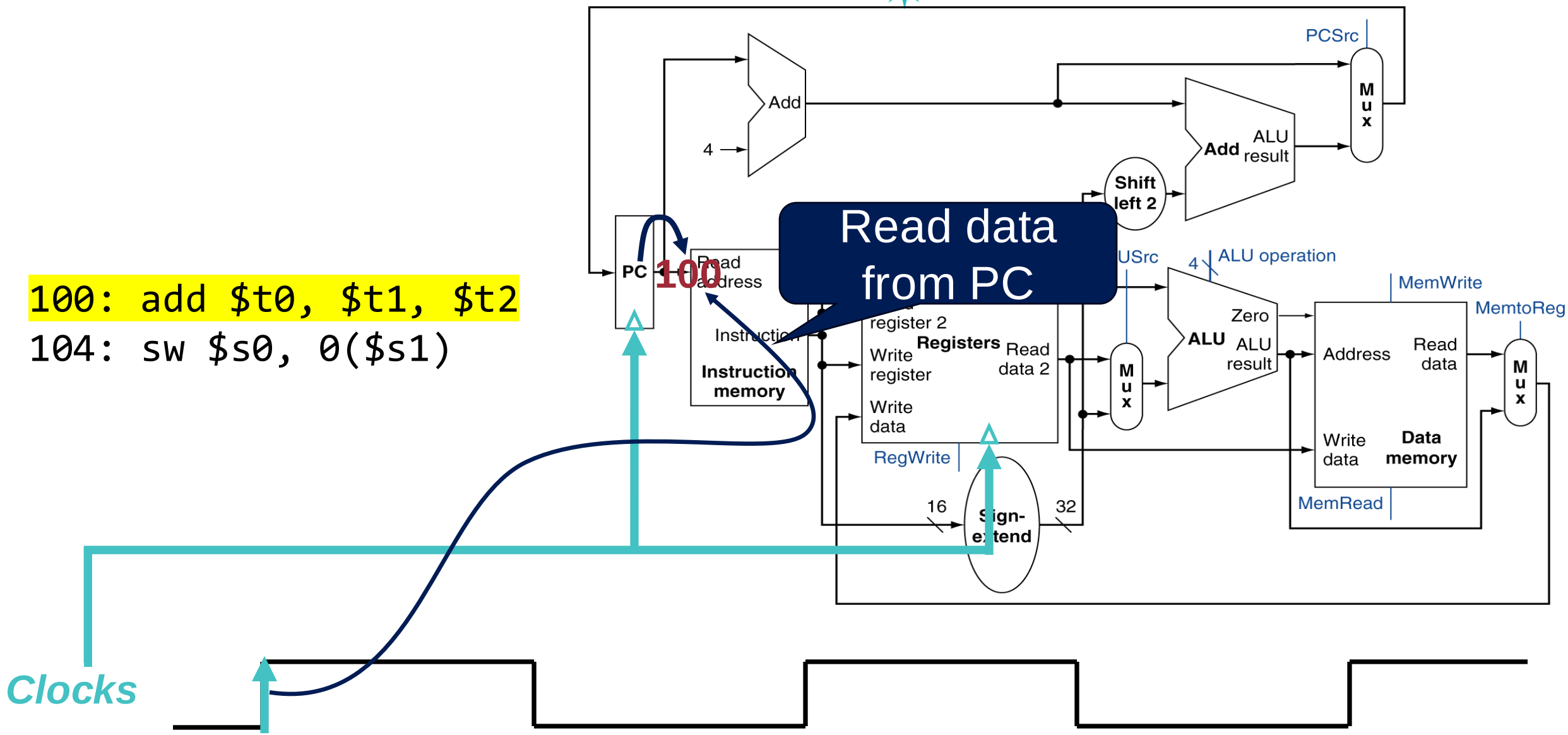


Clocks



(Before Execution) add $t0, $t1, $t2

```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```



(Execution) `add $t0, $t1, $t2`

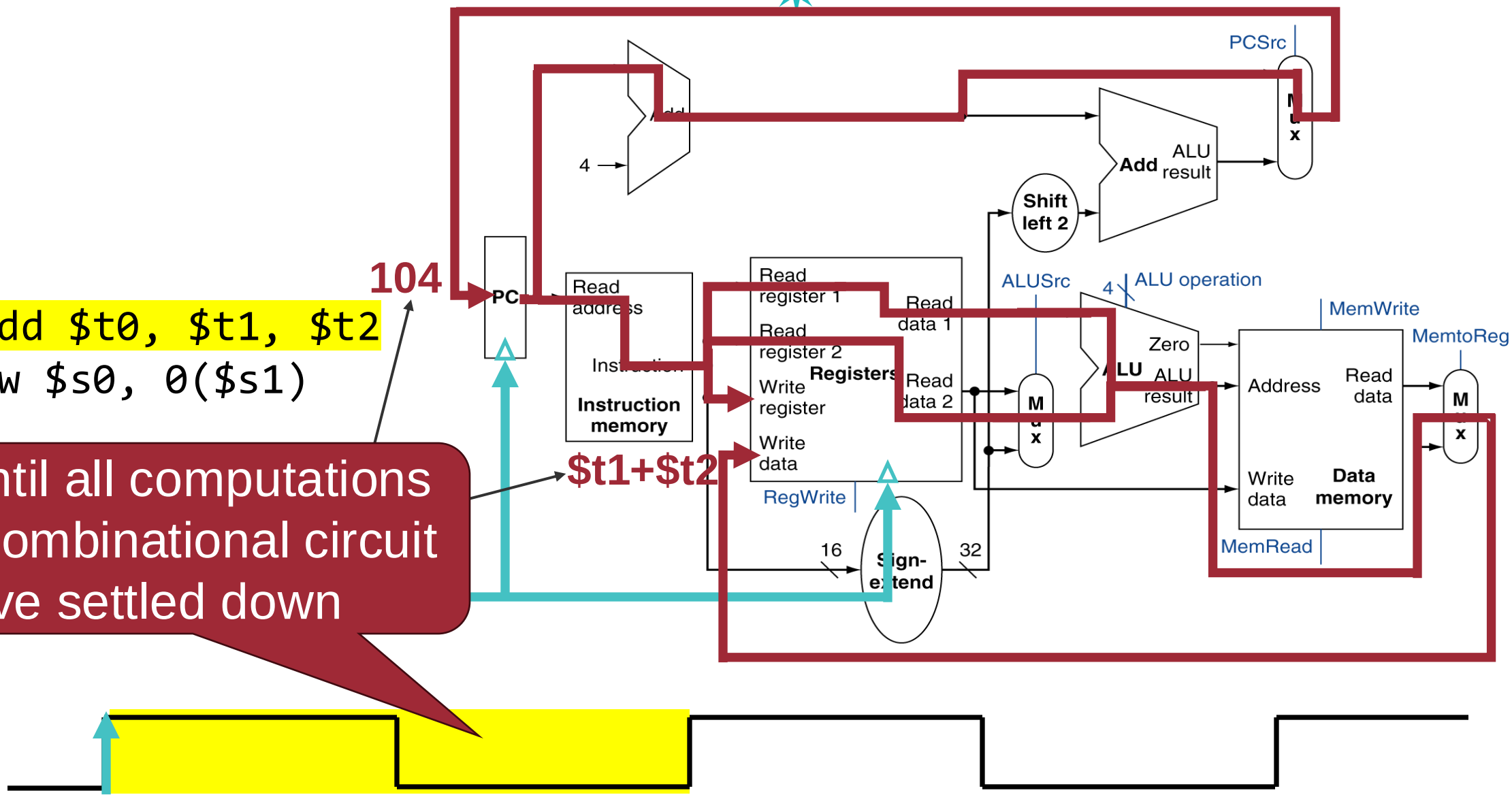
```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```

104

$t1+$t2

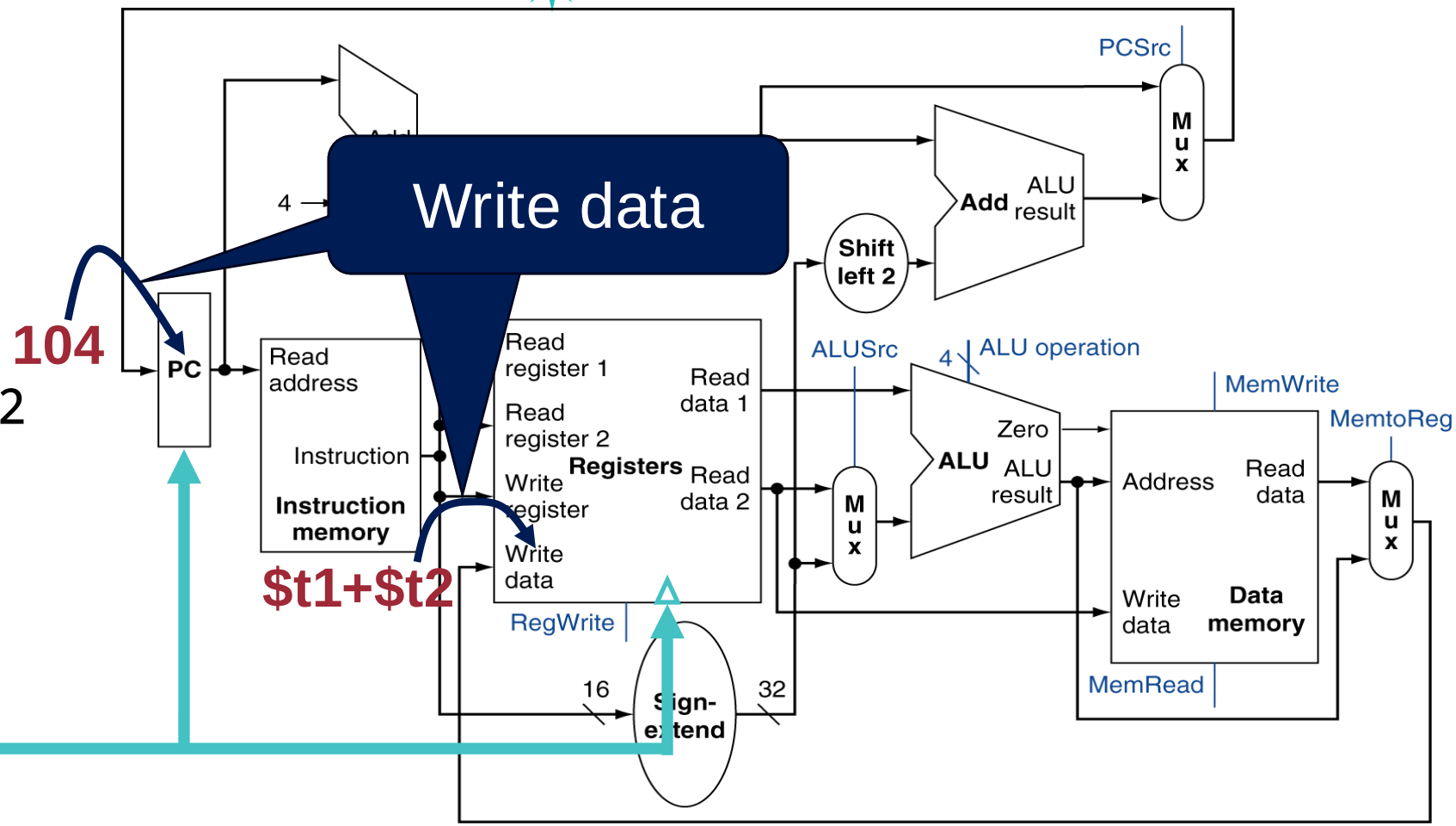
Wait until all computations in the combinational circuit have settled down

Clocks



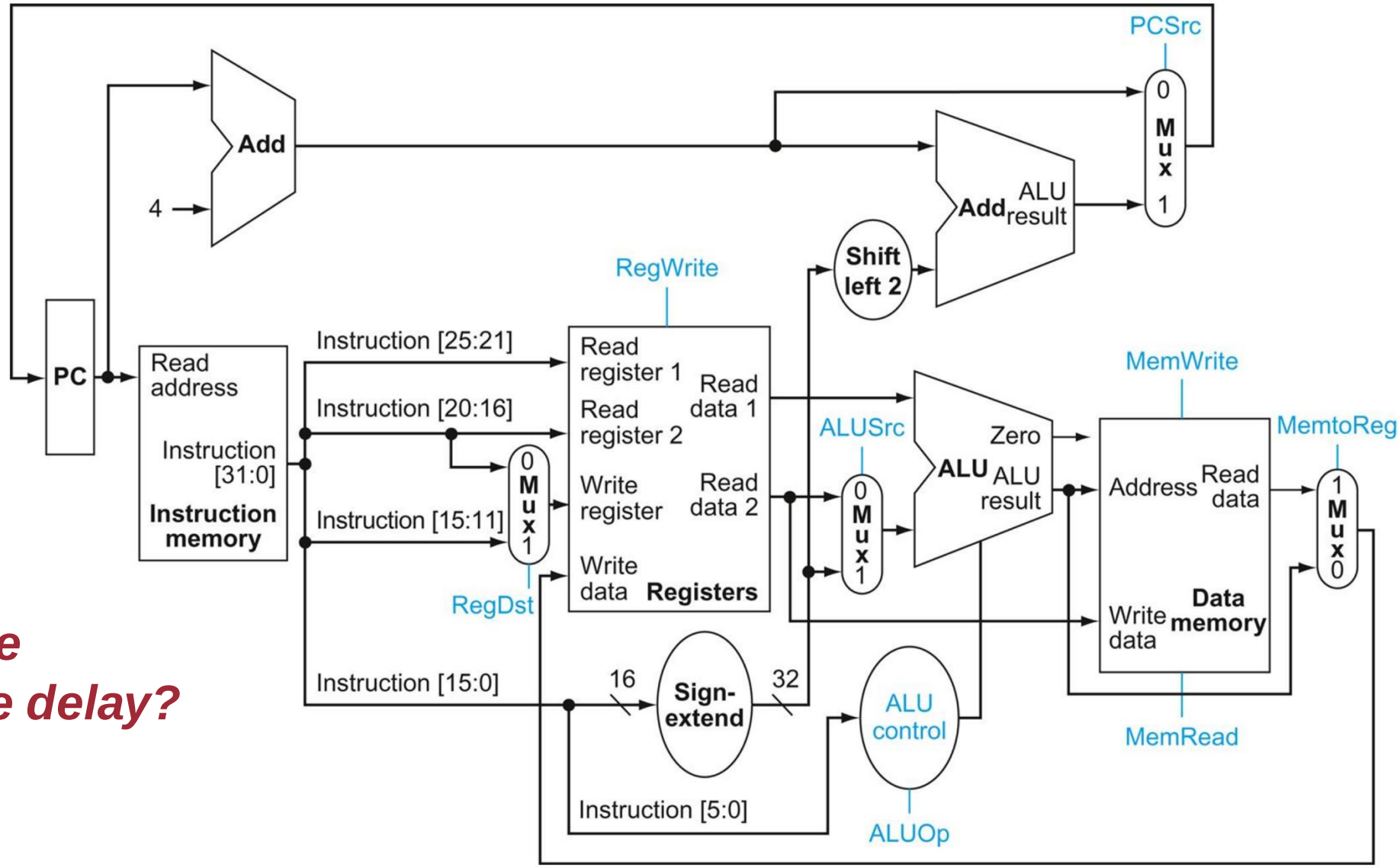
(Before Execution) `sw $s0, 0($s1)`

```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```



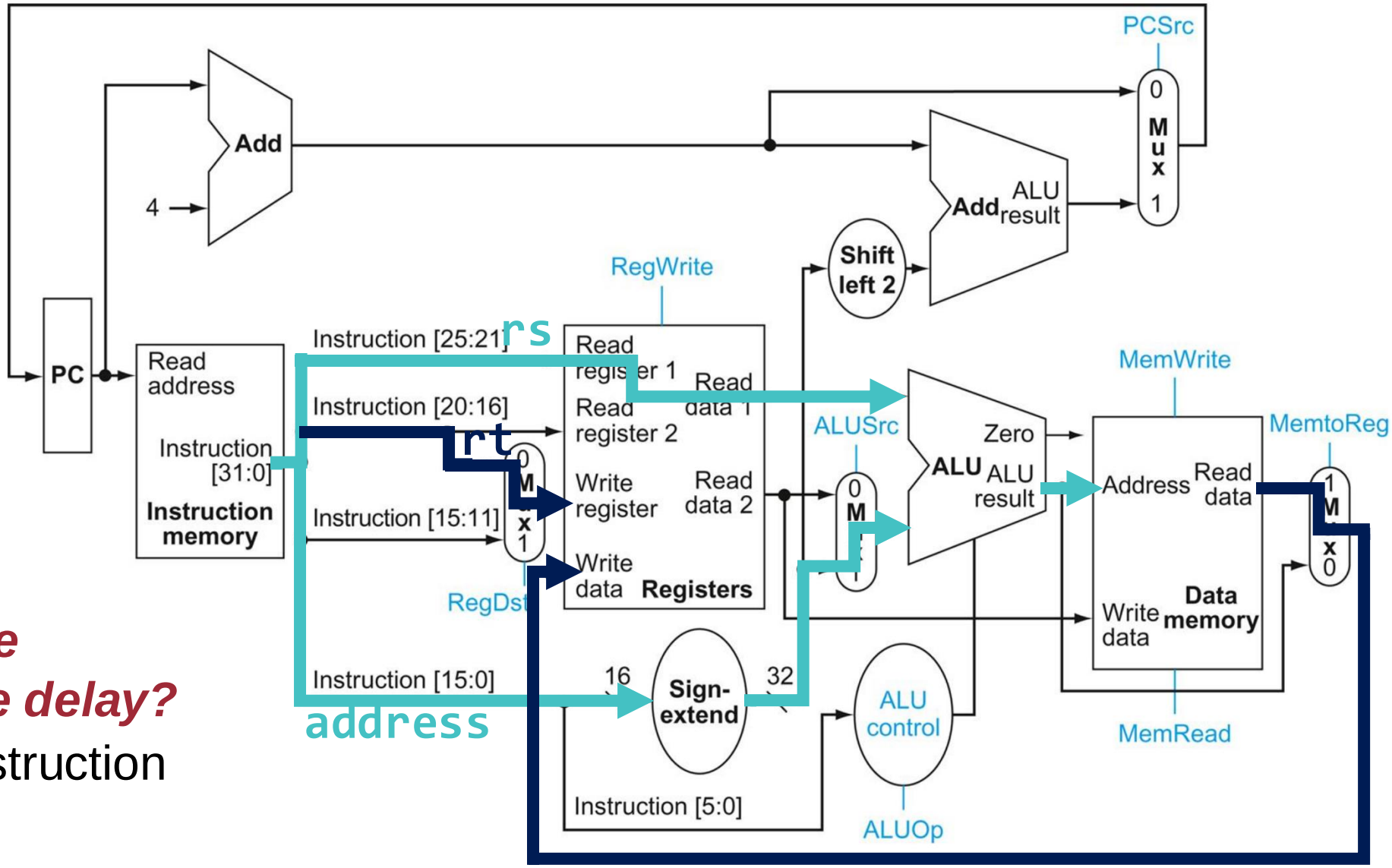
Clocks

Recap: Critical Path



What is the worst case delay?

Recap: Critical Path



What is the worst case delay?

`lw` instruction

Recap: Critical Path

- Calculate cycle time assuming negligible delays except:
  - Memory access (200ps), ALU (100ps), Register access (50ps)

Instruction	Functional units used by the instruction class					
	Instruction fetch	Instruction decode	Execution	Memory Access	Register Write-back	
R-type	200ps	50ps	100ps		50ps	400ps
Load word	200ps	50ps	100ps	200ps	50ps	600ps
Store word	200ps	50ps	100ps	200ps		550ps
Conditional Branch	200ps	50ps	100ps			350ps
Jump	200ps					200ps

Discussion Points (1): Critical Path

- Calculate cycle time assuming negligible delays except

Critical path!
The clock cycle time (period) with single clock will be determined by the longest instruction, which is 600ps

		decode	Execution	Access	Write-back	
R-type	200ps	50ps	100ps		50ps	400ps
Load word	200ps	50ps	100ps	200ps	50ps	600ps
Store word	200ps	50ps	100ps	200ps		550ps
Conditional Branch	200ps	50ps	100ps			350ps
Jump	200ps					200ps

Limitation of a Single-Cycle Datapath

- Clock Cycle Time = 600ps

Even though it could finish early, it has to wait for 600ps

Instruction	Functional units used by t				
	Instruction fetch	Instruction decode	Execution	ory Access	Register Write-back
R-type	200ps	50ps	100ps		50ps
Load word	200ps	50ps	100ps	200ps	50ps
Store word	200ps	50ps	100ps	200ps	
Conditional Branch	200ps	50ps	100ps		
Jump	200ps				

Limitation of a Single-Cycle Datapath

- Clock Cycle Time = 600ps

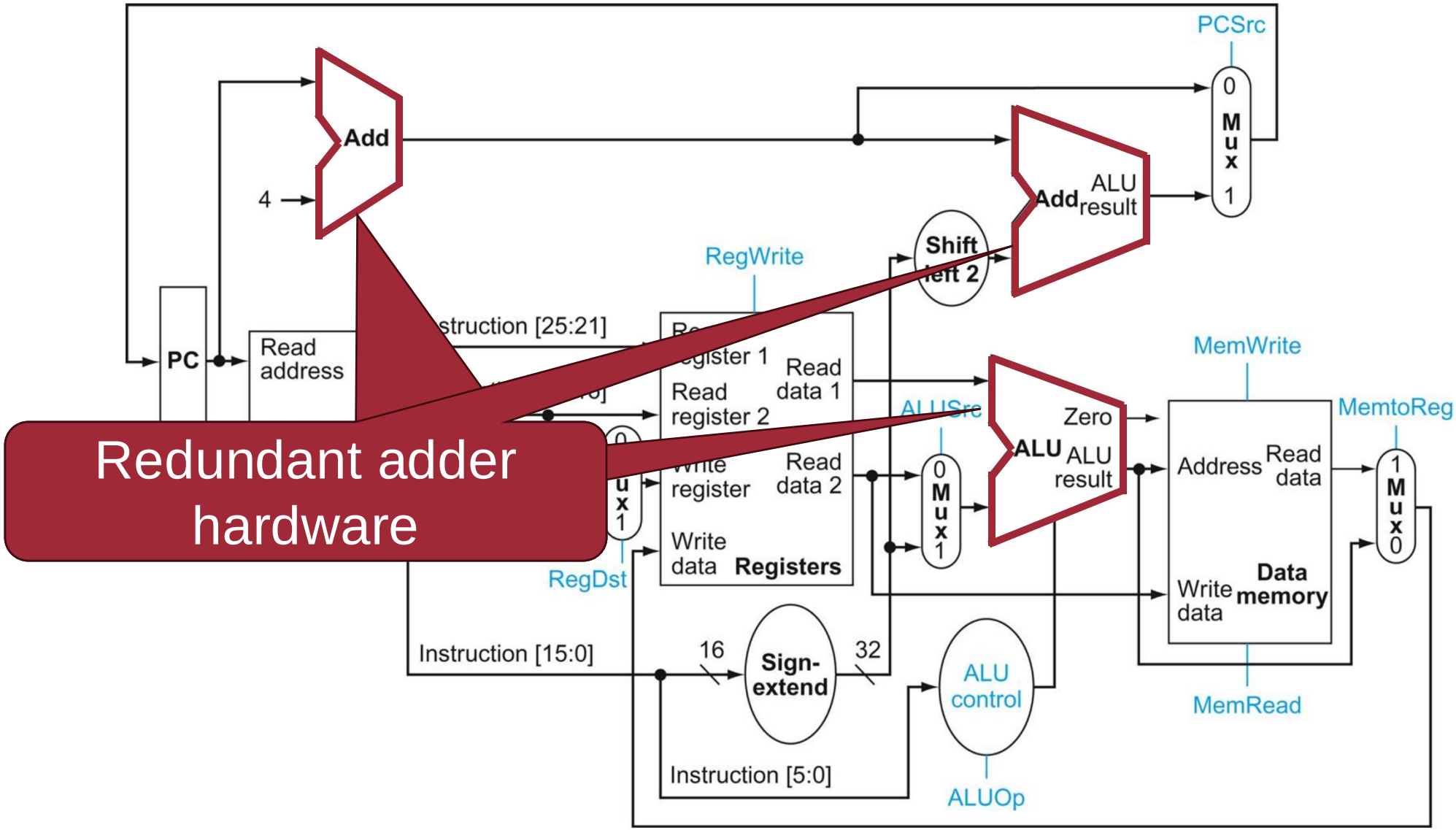
Even though it could finish early, it has to wait for 600ps

Instruction	Functional units used by t				
	Instruction fetch	Instruction decode	Execution	ory Access	Register Write-back
R-type	200ps	50ps	100ps		50ps
Load word	200ps	50ps	100ps	200ps	50ps
Store word	200ps	50ps	100ps	200ps	

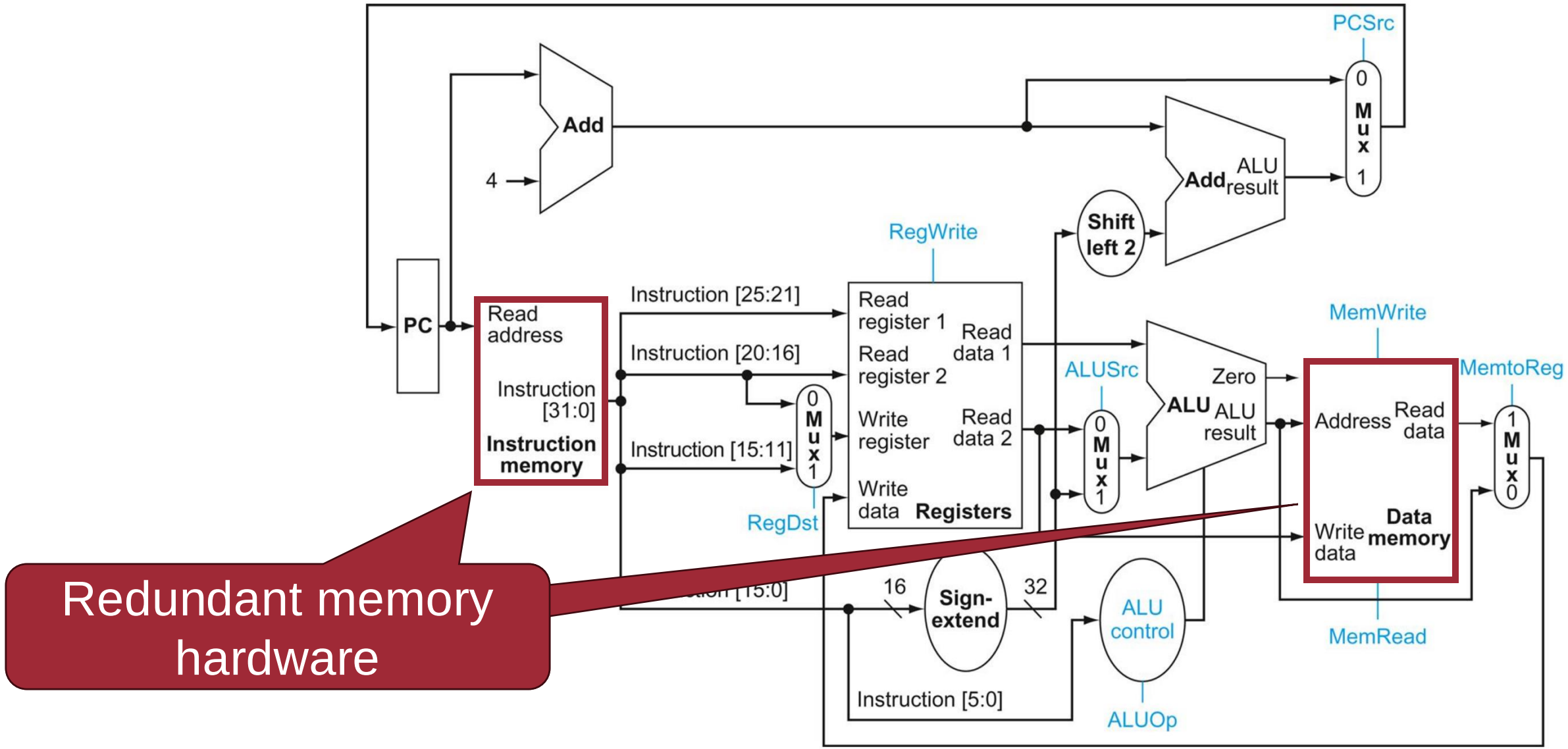
Why a Single-Cycle Implementation is Not Used Today?

Although the CPI is 1, the overall performance of a single-cycle implementation is likely to be poor, since the clock cycle is too long.

Another Disadvantage of Single-Cycle Implementation



Another Disadvantage of Single-Cycle Implementation



Redundant memory hardware

How can we address this issue?

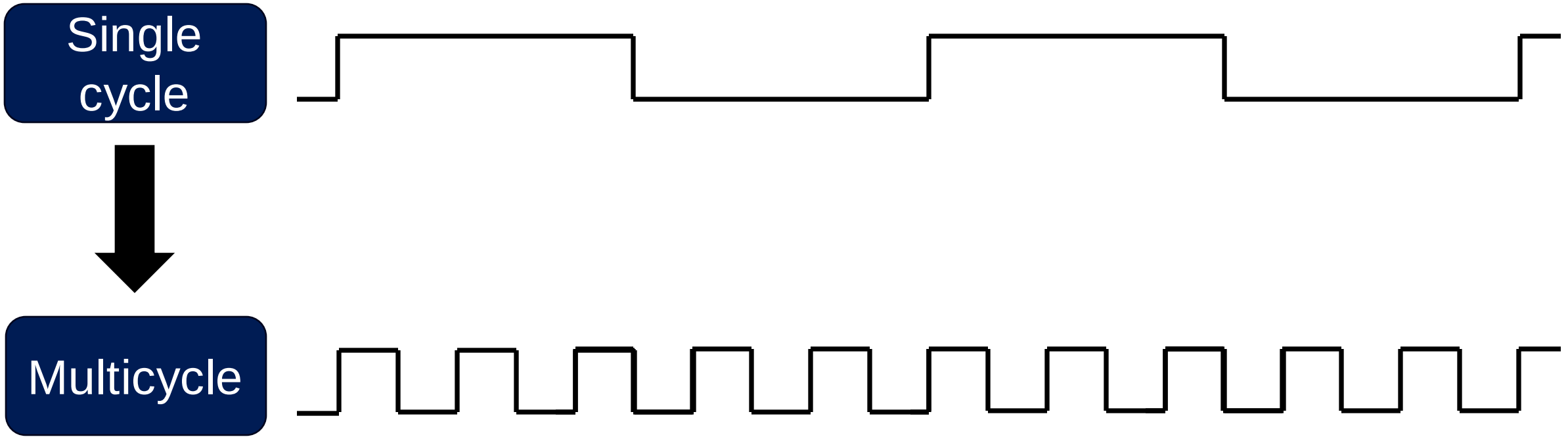
Single-Cycle Implementation

Multicycle Implementation

Today's topic

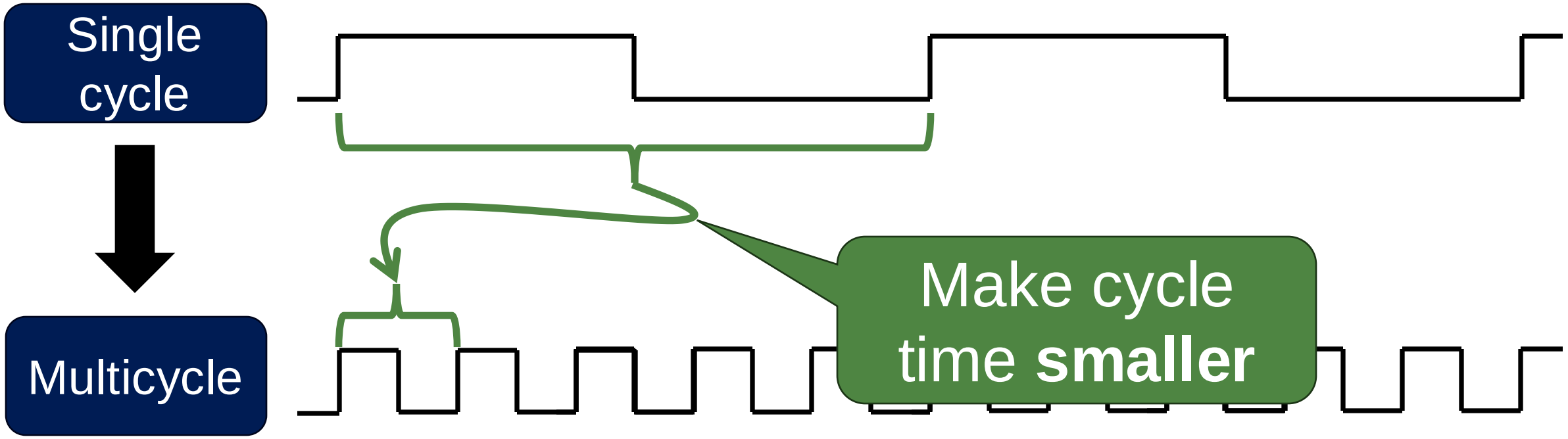
Multicycle Implementation: Intuition

- An instruction is executed in multiple clock cycles



Multicycle Implementation: Intuition

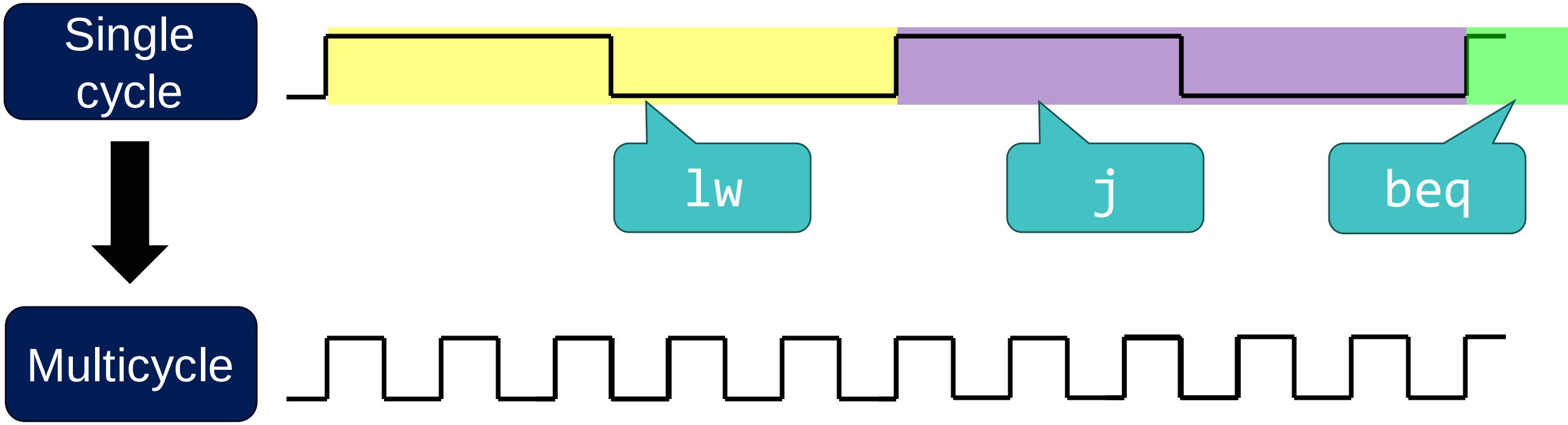
- An instruction is executed in multiple clock cycles



Multicycle Implementation: Intuition

- An instruction is executed in multiple clock cycles

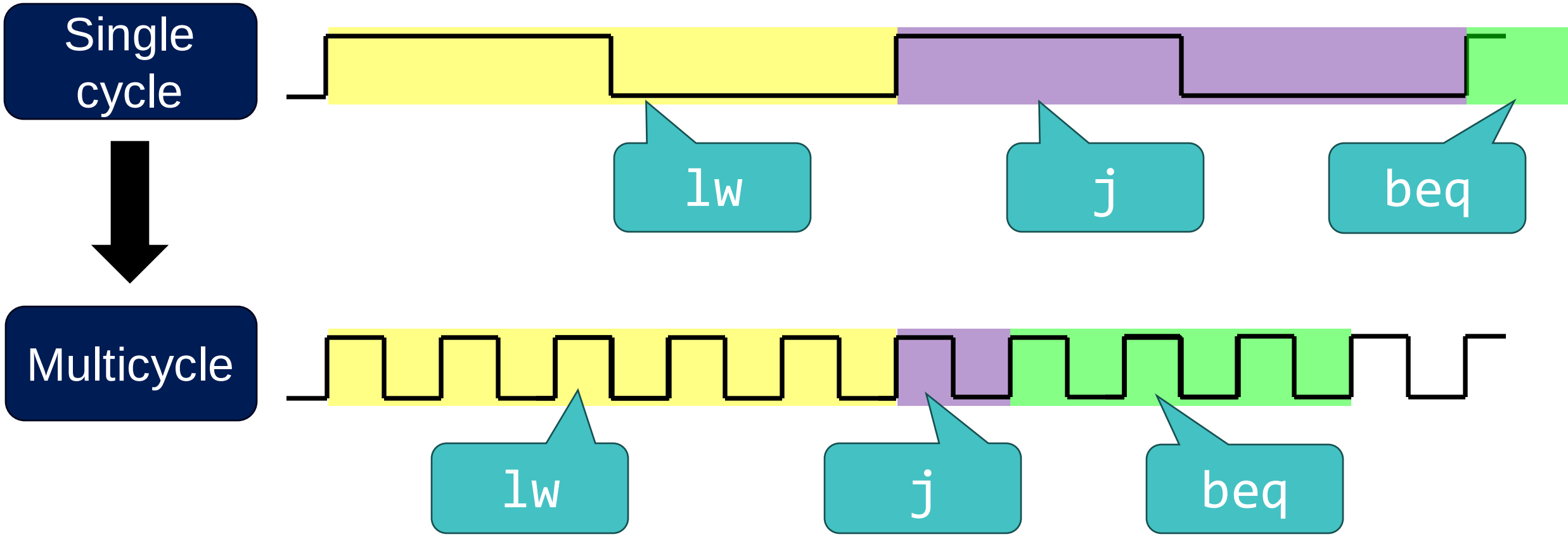
```
lw $2, 4($1)
j ELSE
beq $2, $3, ELSE
```



Multicycle Implementation: Intuition

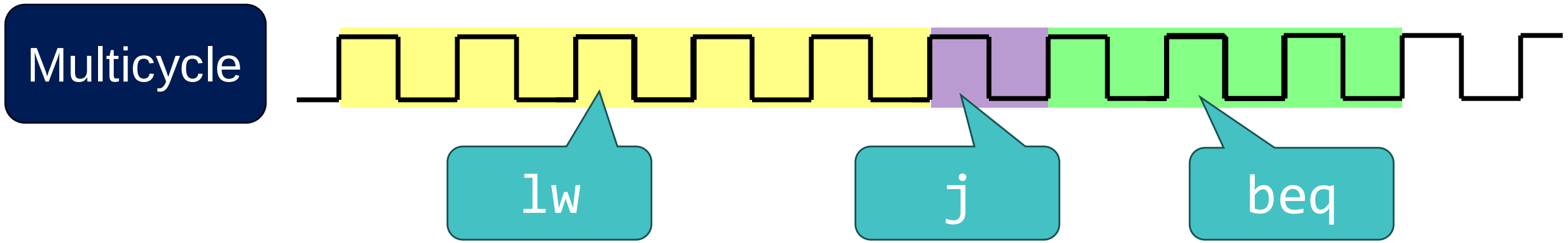
- An instruction is executed in multiple clock cycles

```
lw $2, 4($1)
j ELSE
beq $2, $3, ELSE
```

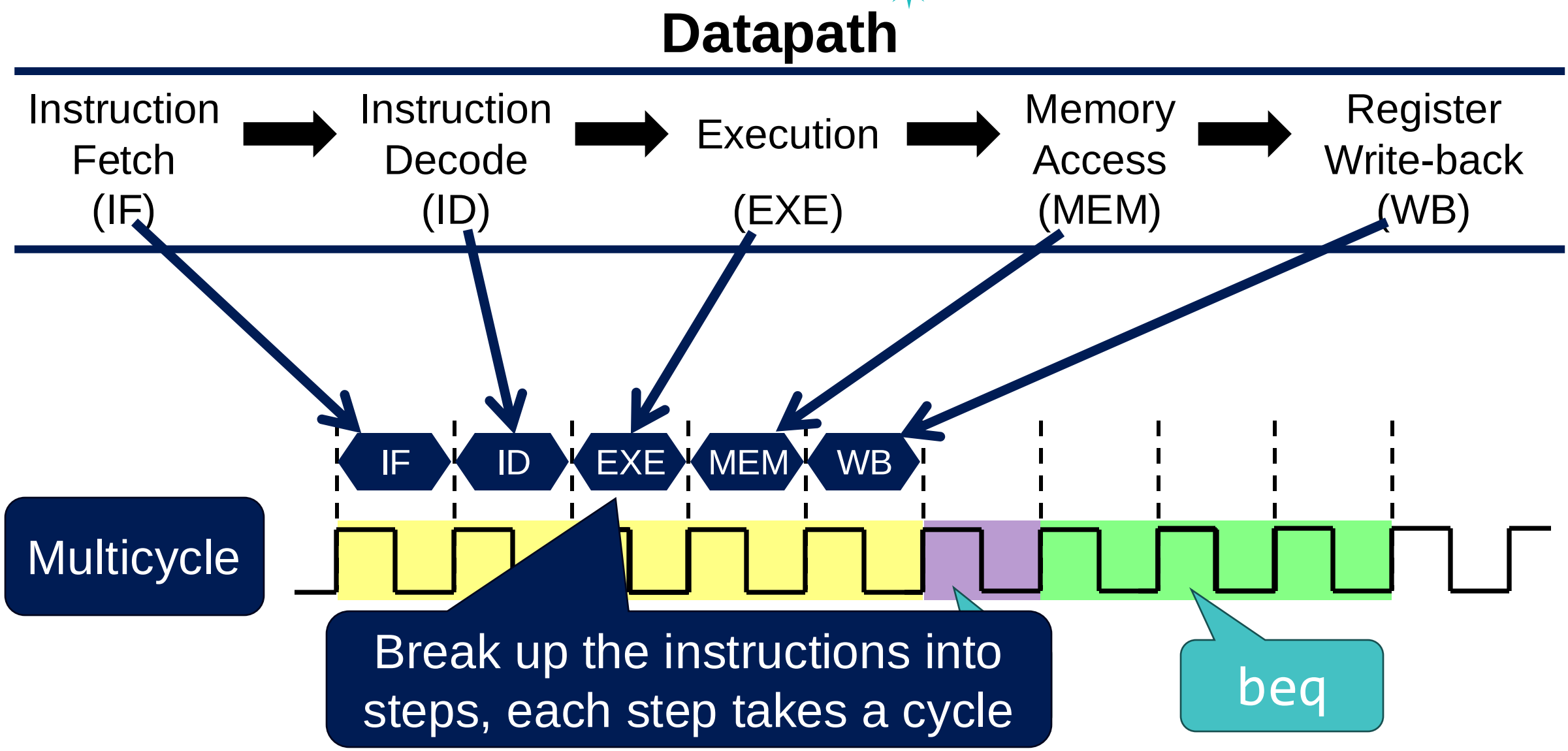


Multicycle Implementation: Intuition

Different instructions take different numbers of cycles!



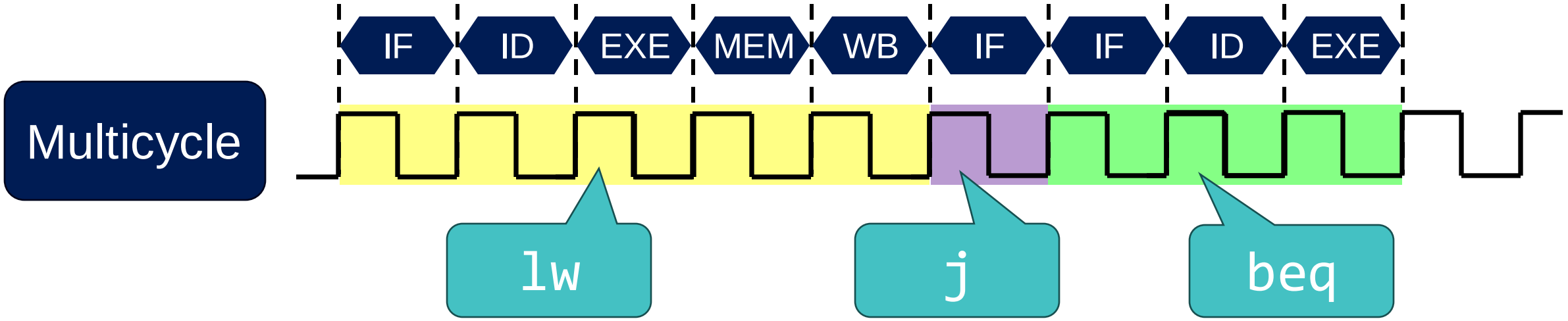
Multicycle Implementation: Intuition



Multicycle Implementation: Intuition

Datapath

Instruction Fetch (IF) → Instruction Decode (ID) → Execution (EXE) → Memory Access (MEM) → Register Write-back (WB)



Multicycle Implementation: Overview

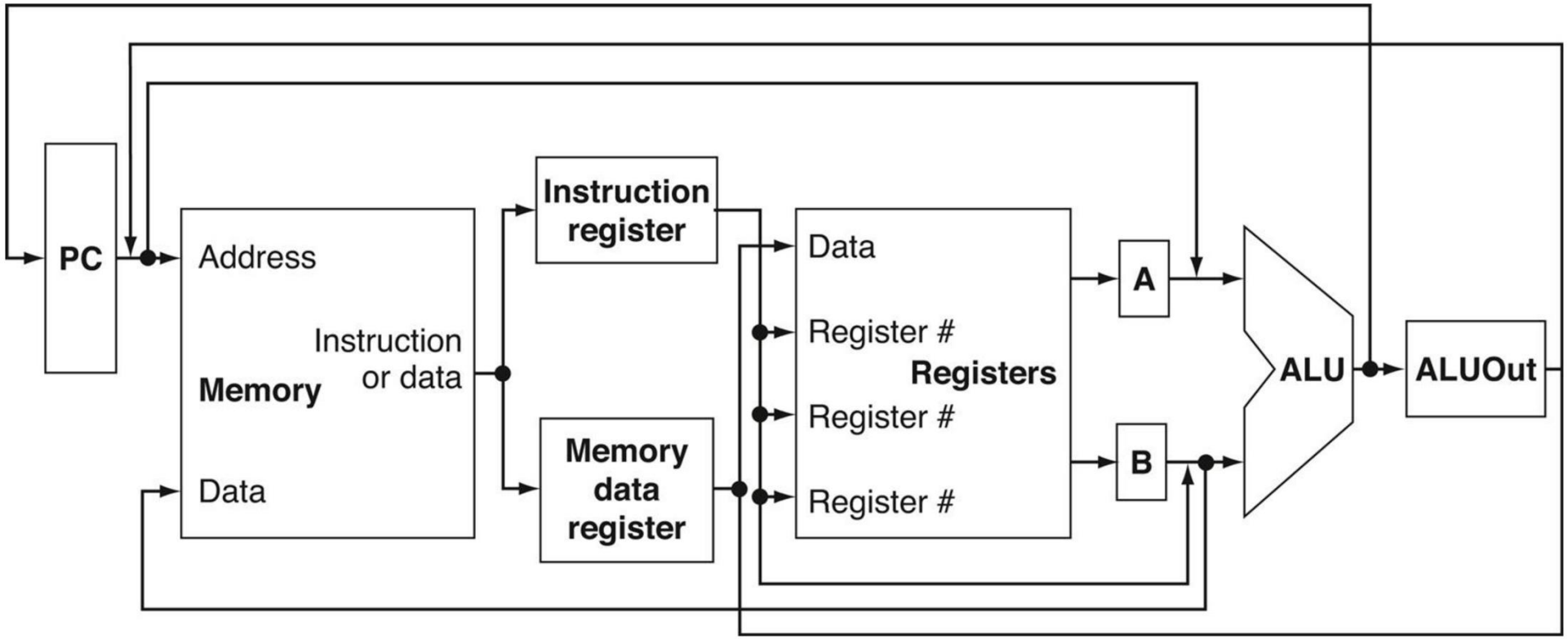
- **An instruction is executed in multiple clock cycles**
  - Different instructions take different numbers of cycles
  - Make cycle time smaller

How it can be possible?
→ Introduce additional internal registers

Multicycle Implementation: Overview

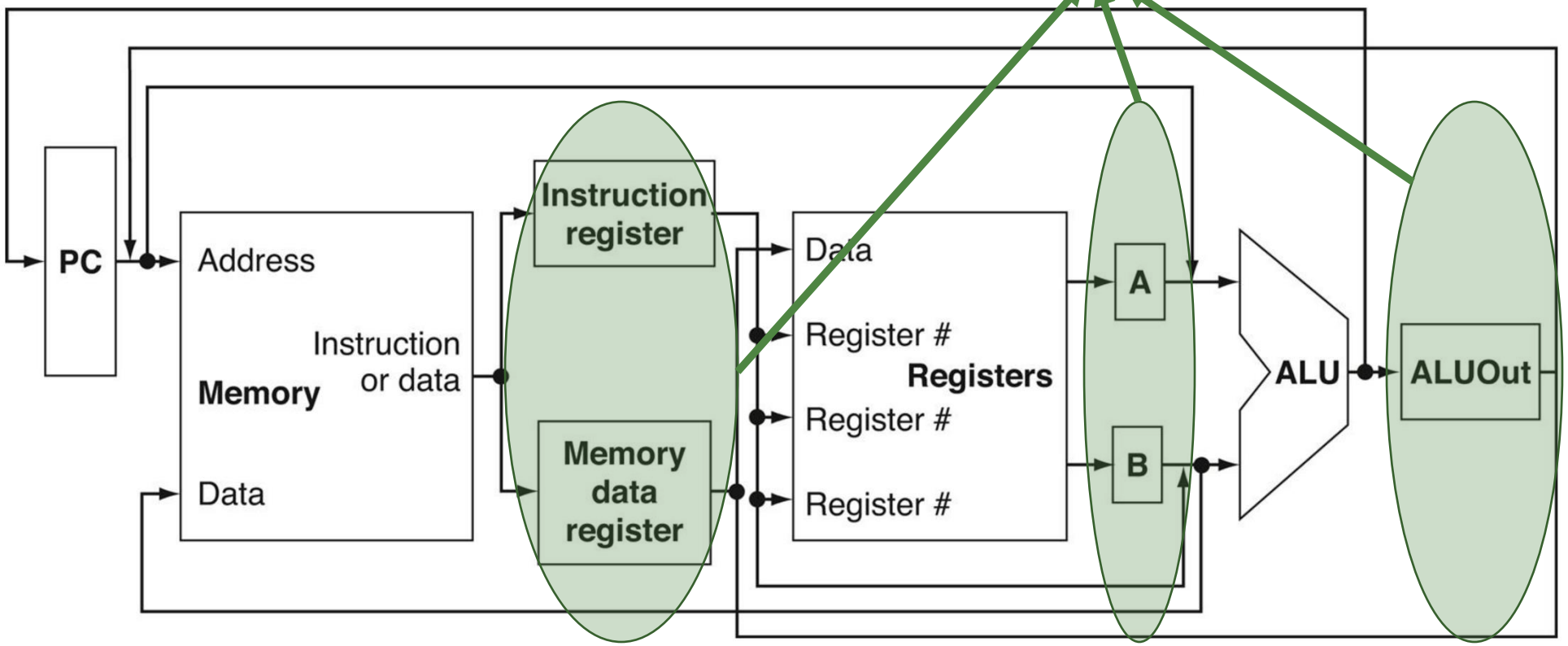
- **An instruction is executed in multiple clock cycles**
  - Different instructions take different numbers of cycles
  - Make cycle time smaller

How it can be possible?
→ Introduce additional internal registers



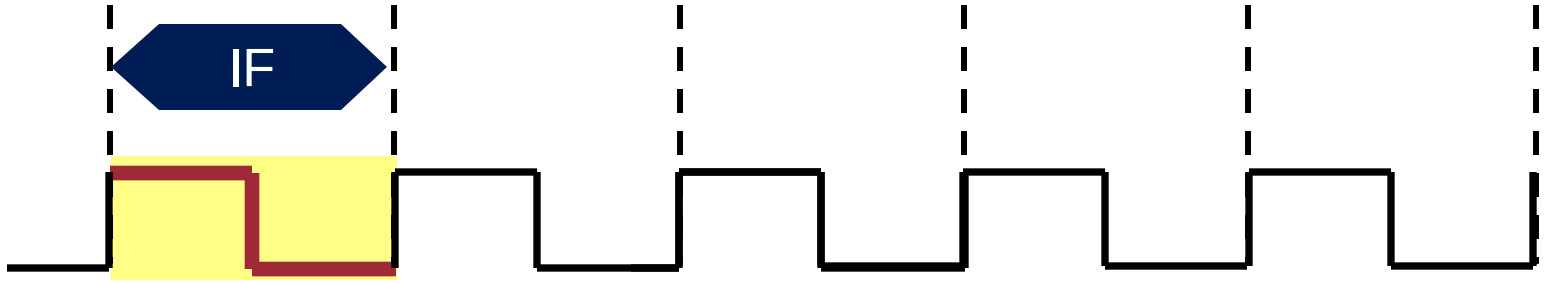
Multicycle Implementation: Overview

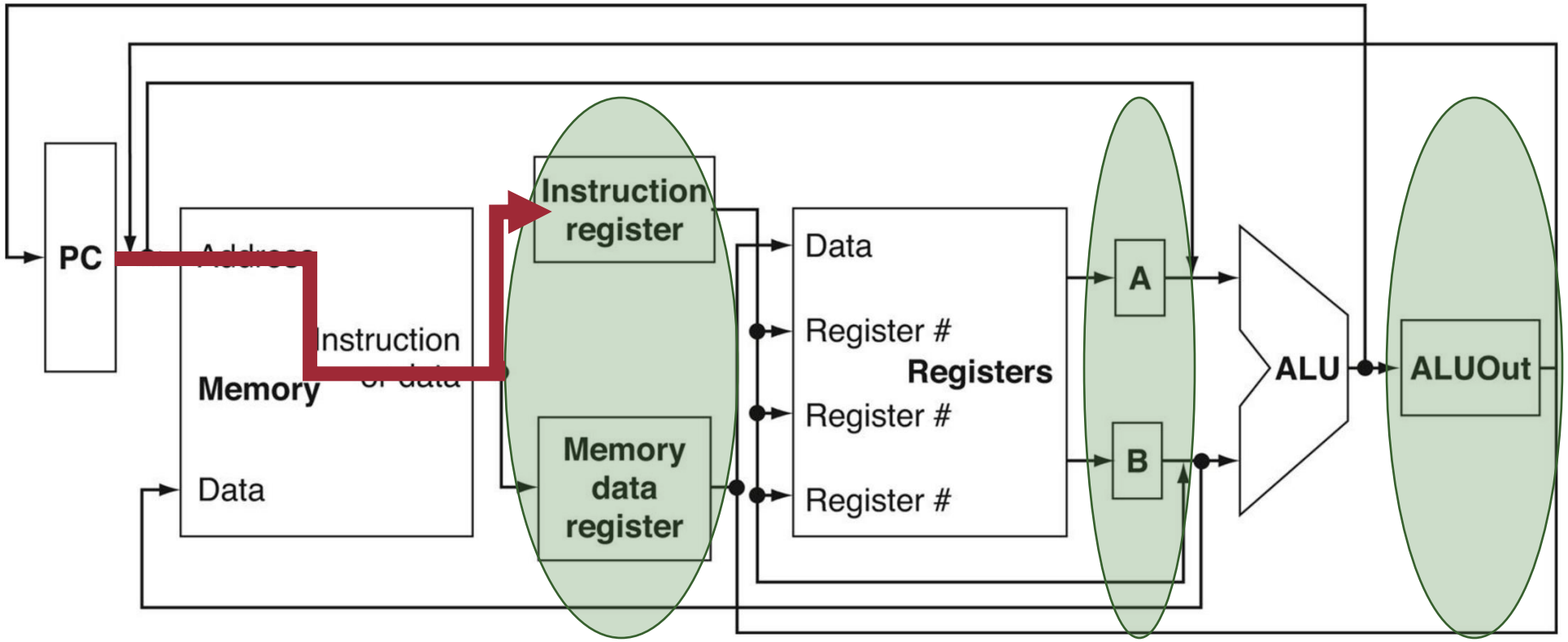
How it can be possible?
→ Introduce additional internal registers



Multicycle Implementation: Overview

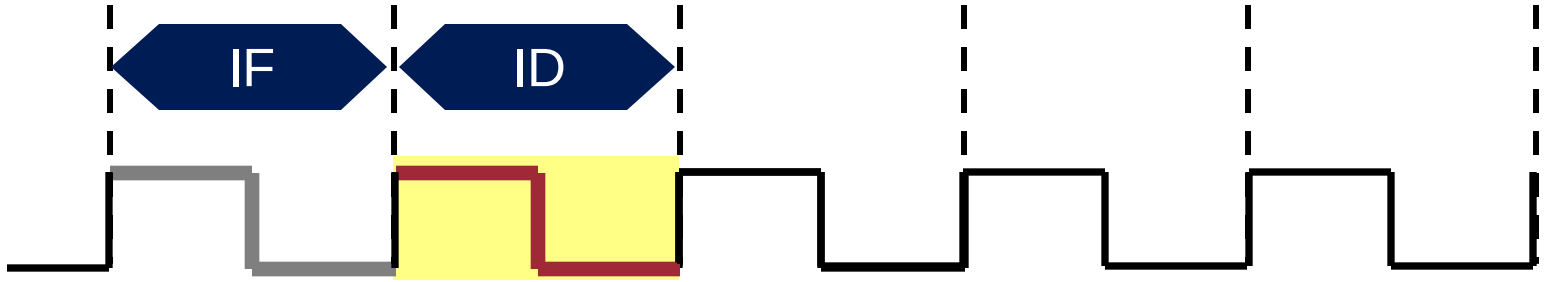
lw instruction

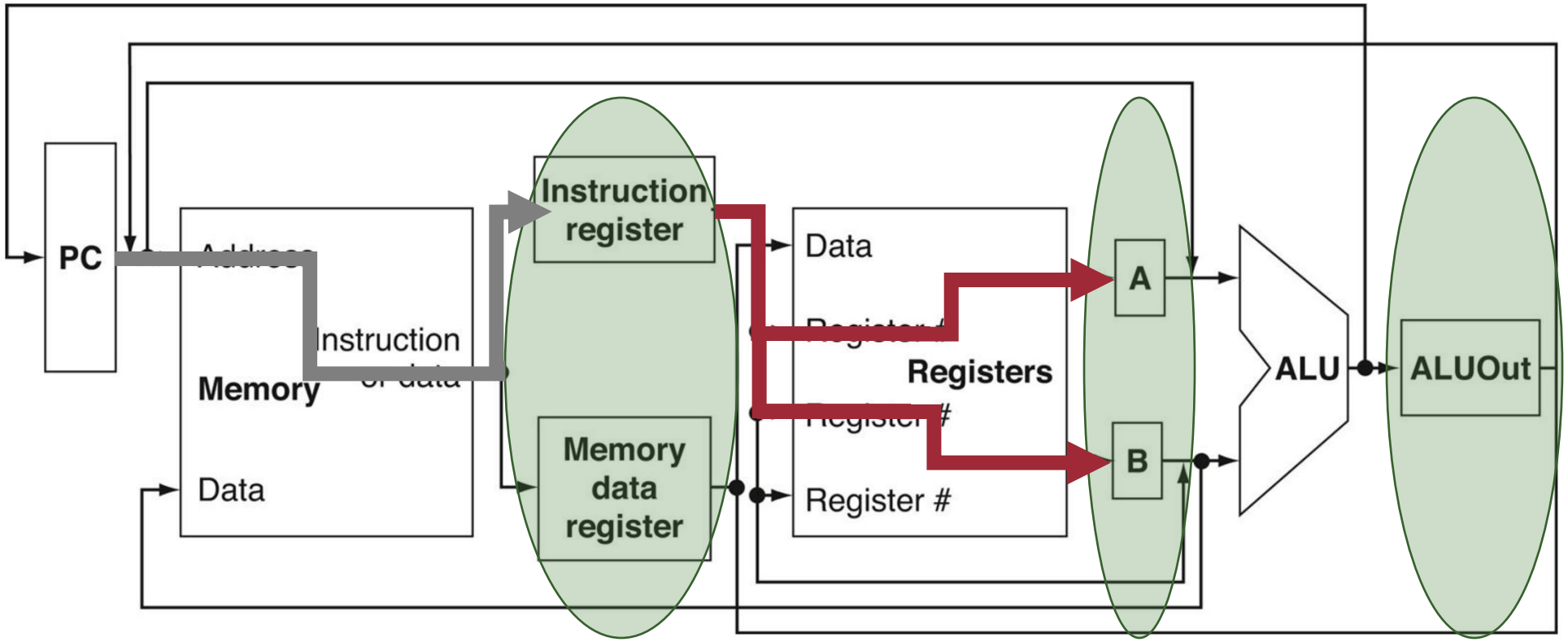




Multicycle Implementation: Overview

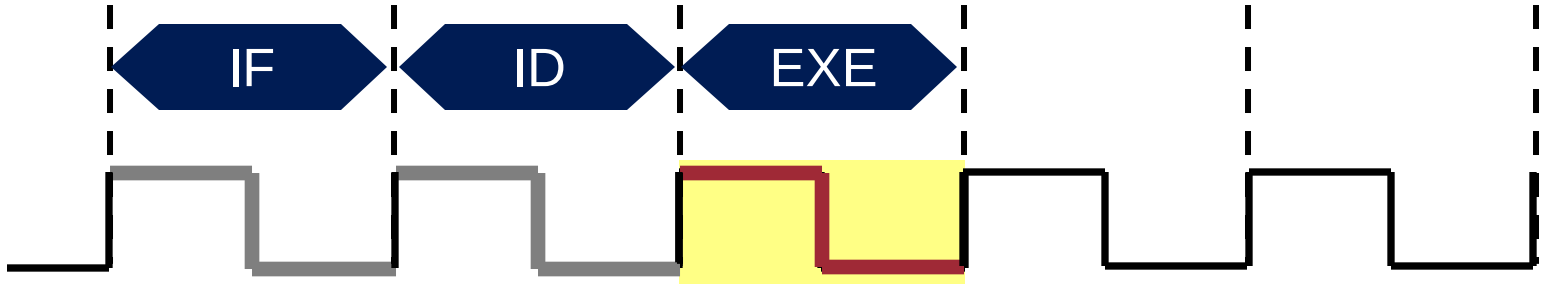
lw instruction

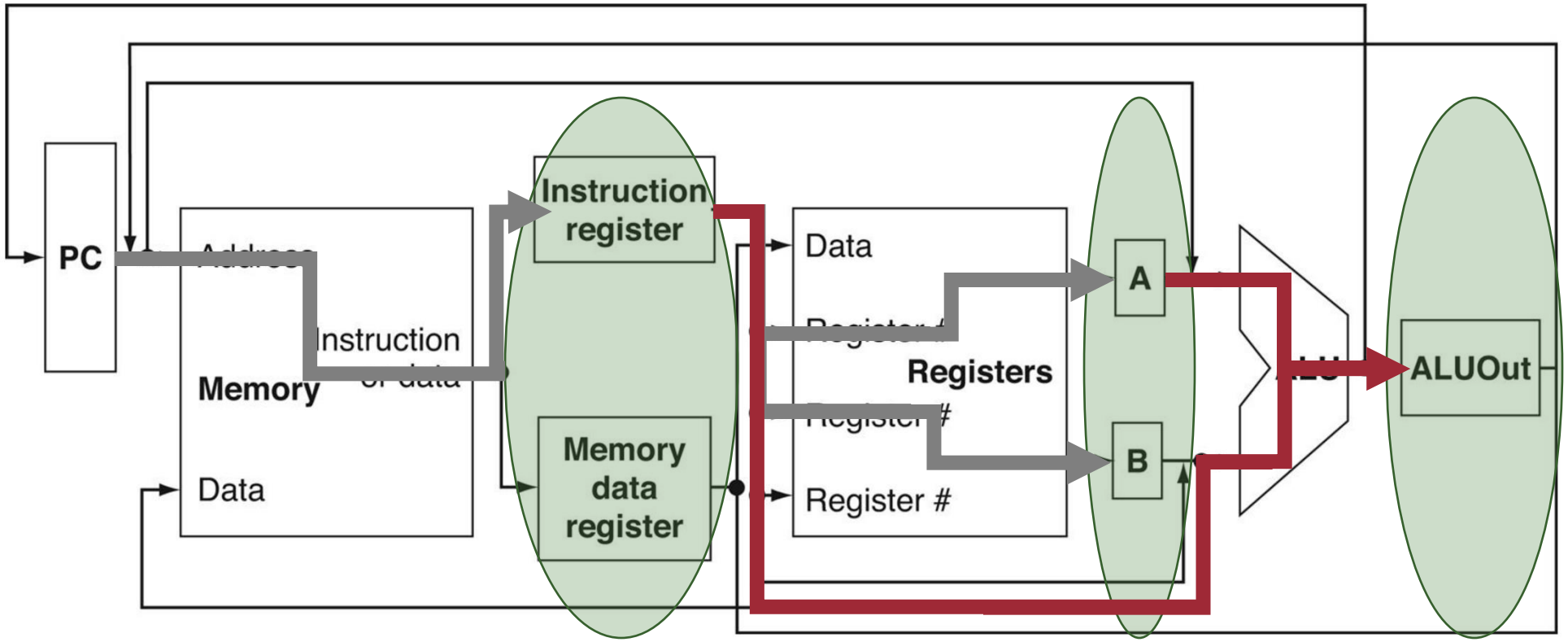




Multicycle Implementation: Overview

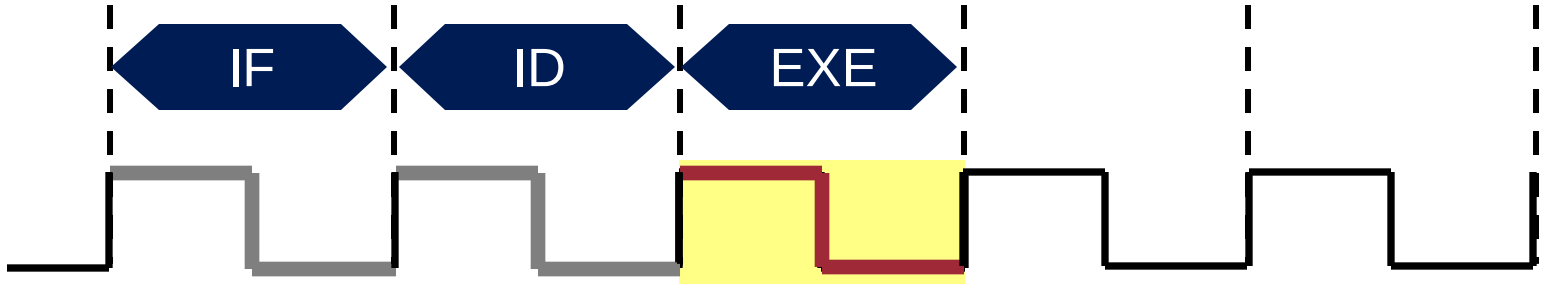
lw instruction



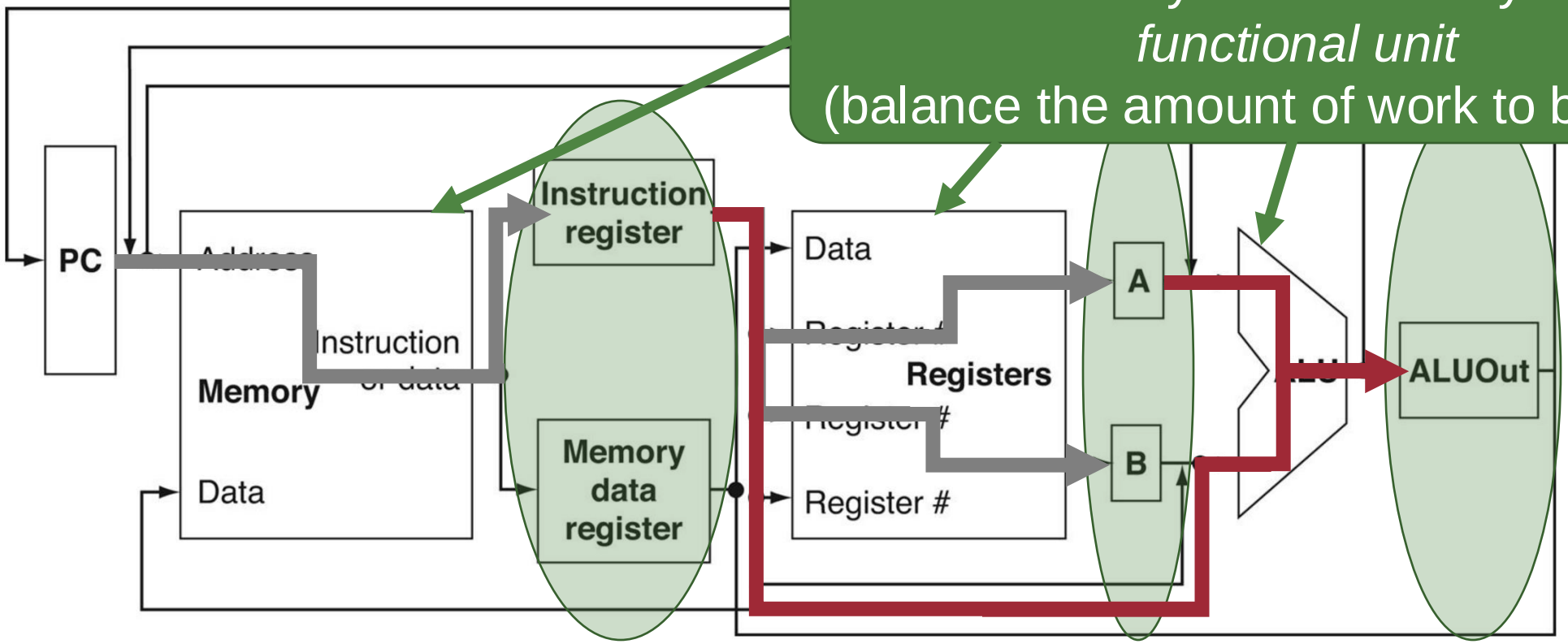


Multicycle Implementation: Overview

lw instruction

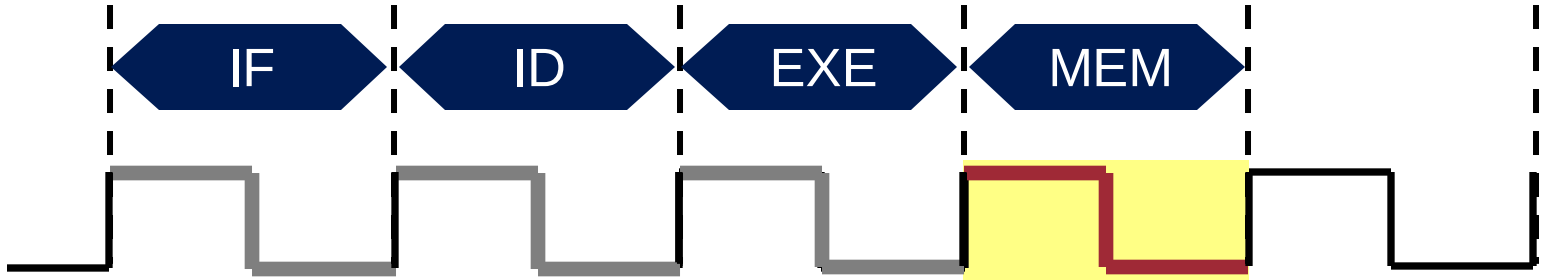


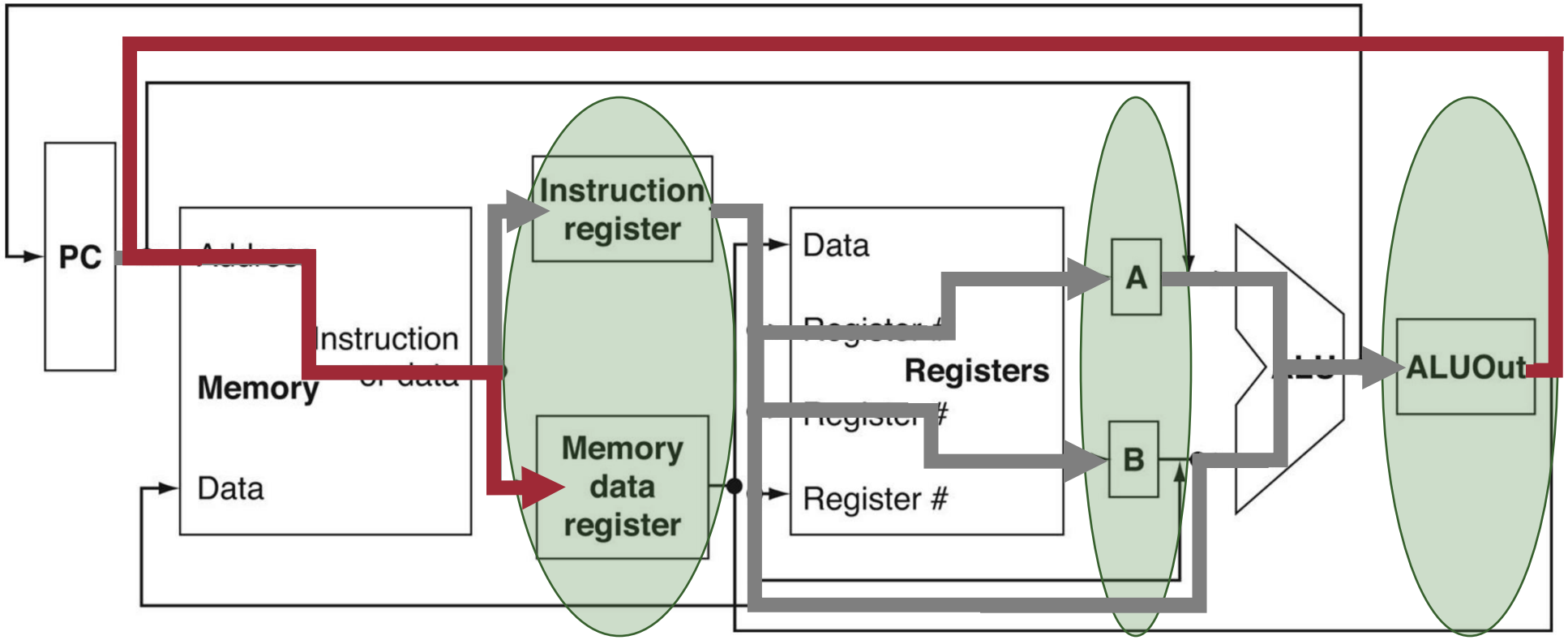
Restrict each cycle to use only *one major functional unit*
(balance the amount of work to be done!)



Multicycle Implementation: Overview

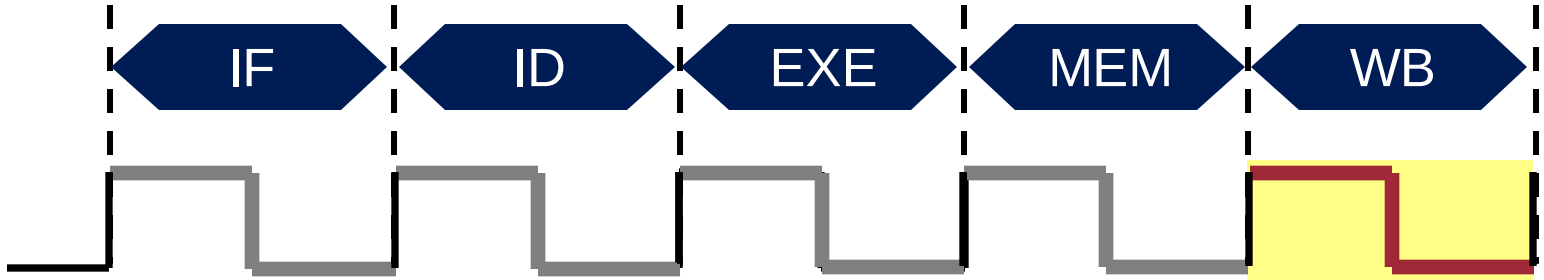
lw instruction

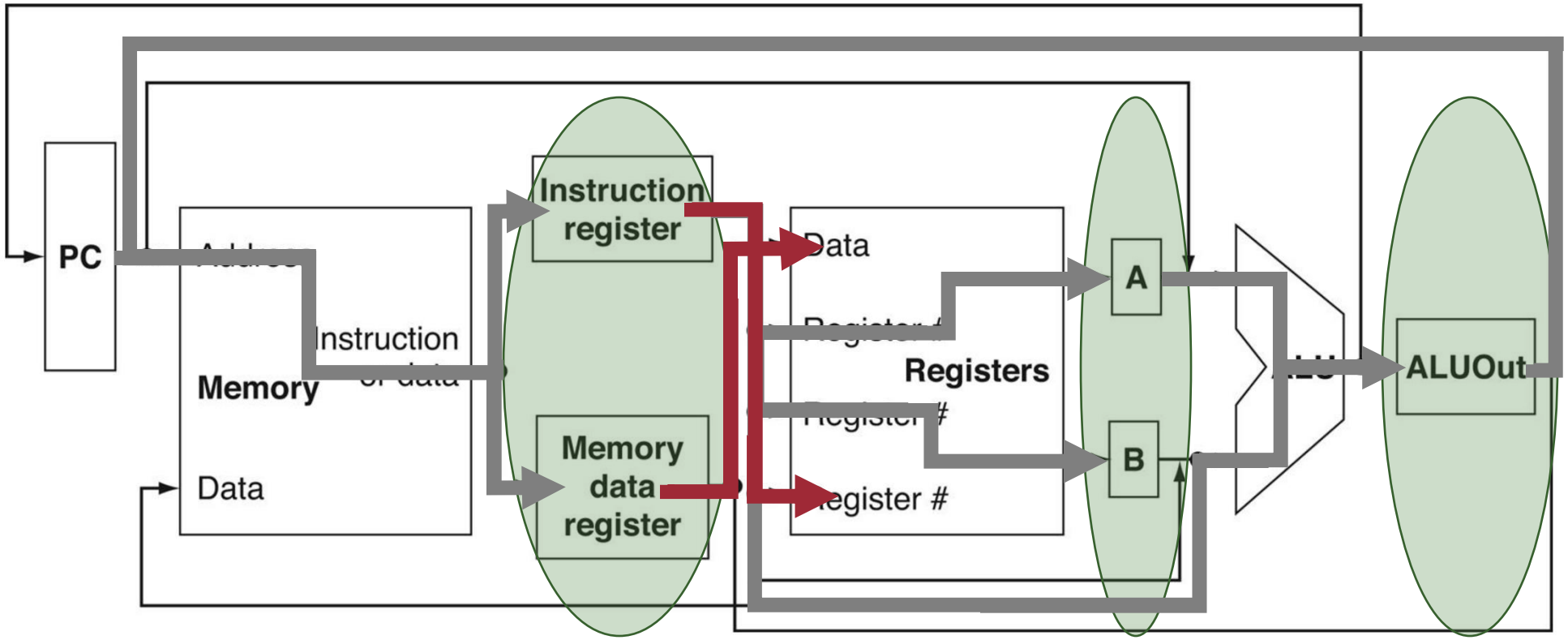




Multicycle Implementation: Overview

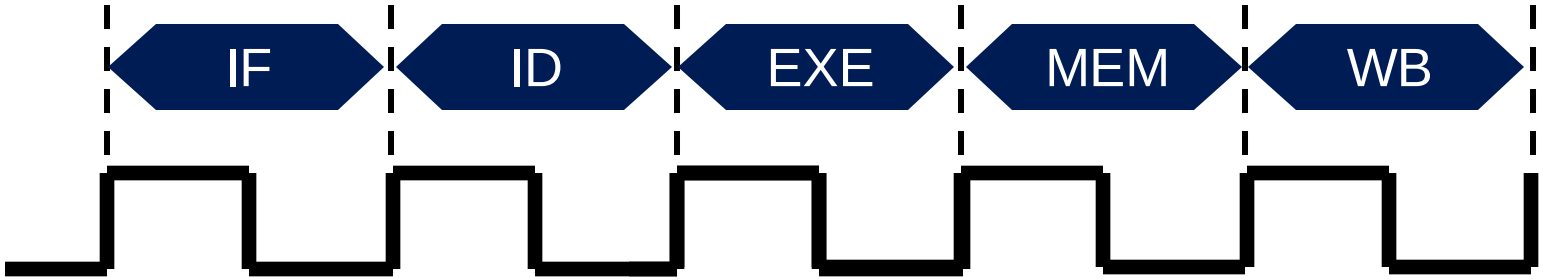
lw instruction

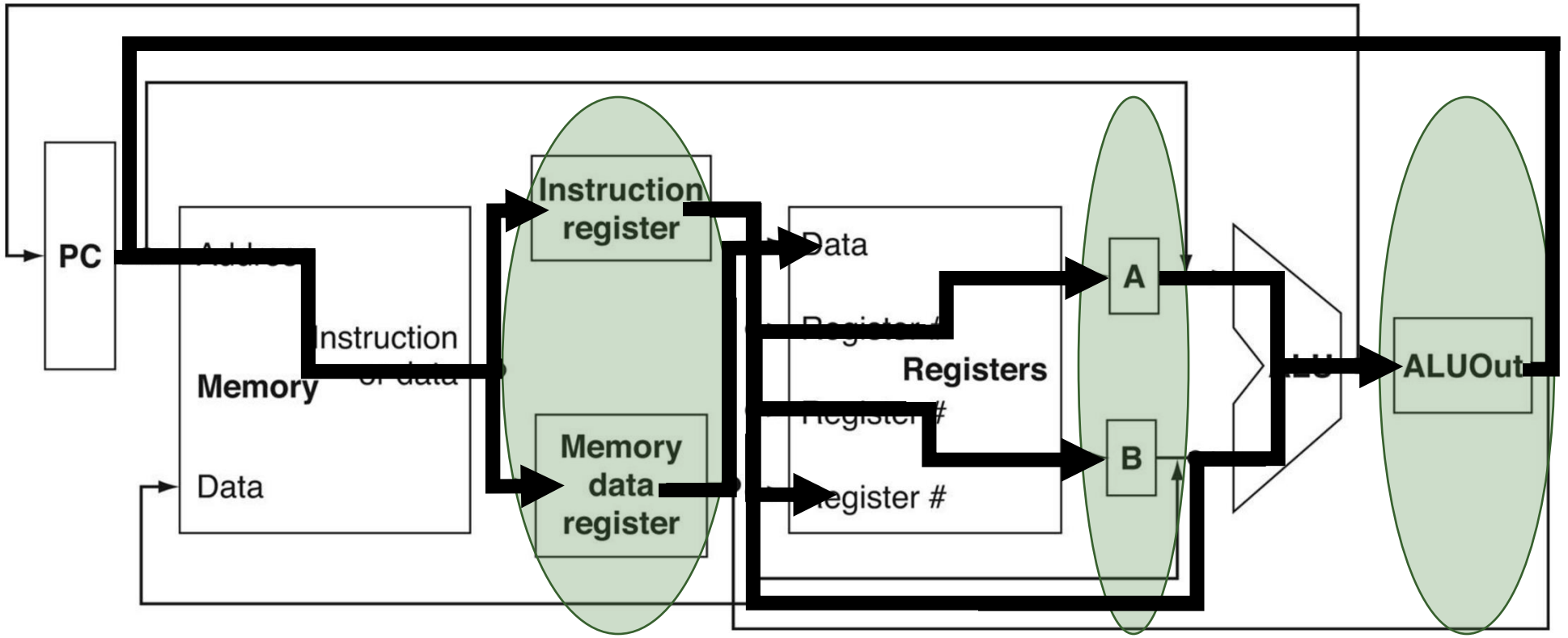




Multicycle Implementation: Overview

lw instruction





Multicycle Implementation: Overview

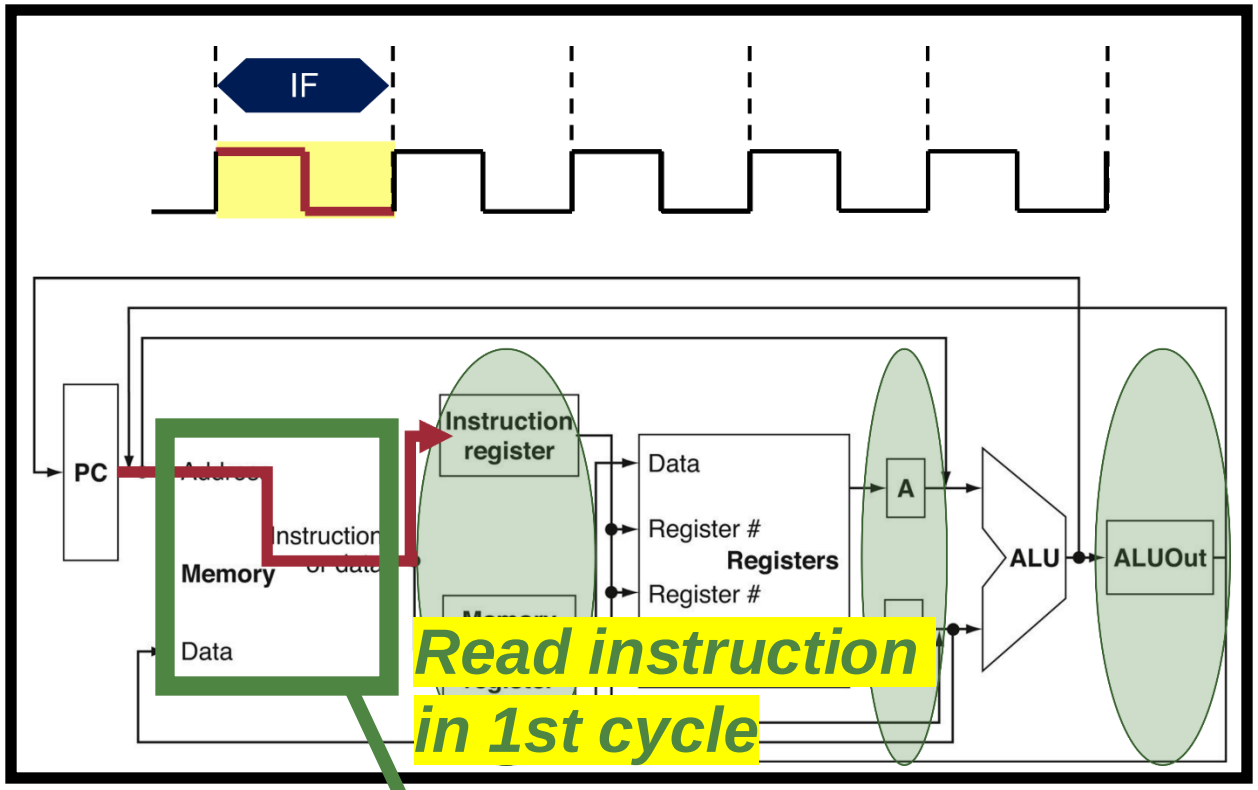
- **An instruction is executed in multiple clock cycles**
  - Different instructions take different numbers of cycles
  - Make cycle time smaller
- **Break up the instructions into steps, each step takes a cycle**
  - Balance the amount of work to be done
  - Restrict each cycle to use only one major functional unit
- **At the end of a cycle**
  - (On rising-edge) store values for use in later cycles
  - Introduce additional internal registers

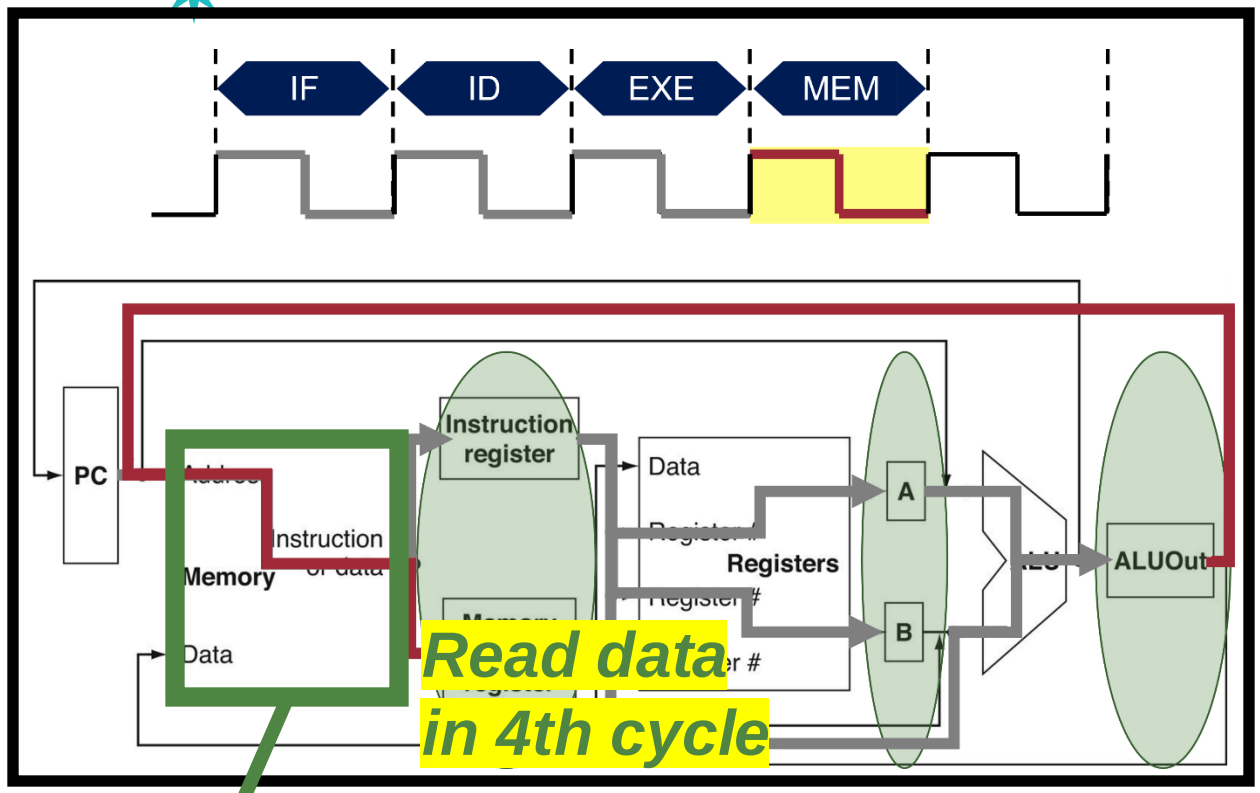
Multicycle Implementation: Advantages

Compared with single-cycle implementation, performance can be ***increased*** due to:

- Clock cycle period ↓
- Different instructions take different numbers of cycles
- Hardware sharing: single ALU and single memory
  - Reduce the circuit area
  - They can be used for **different purposes** *in different clock cycle*

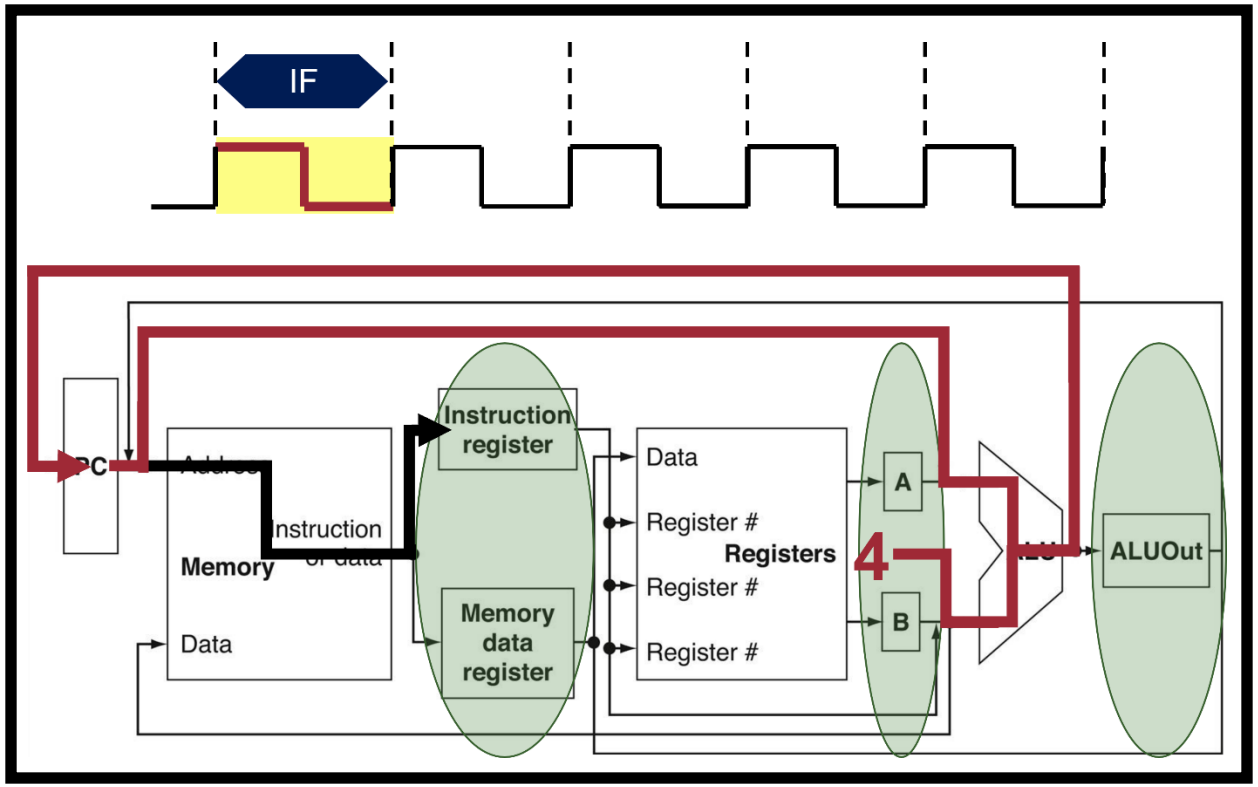
Hardware Sharing: Single Memory

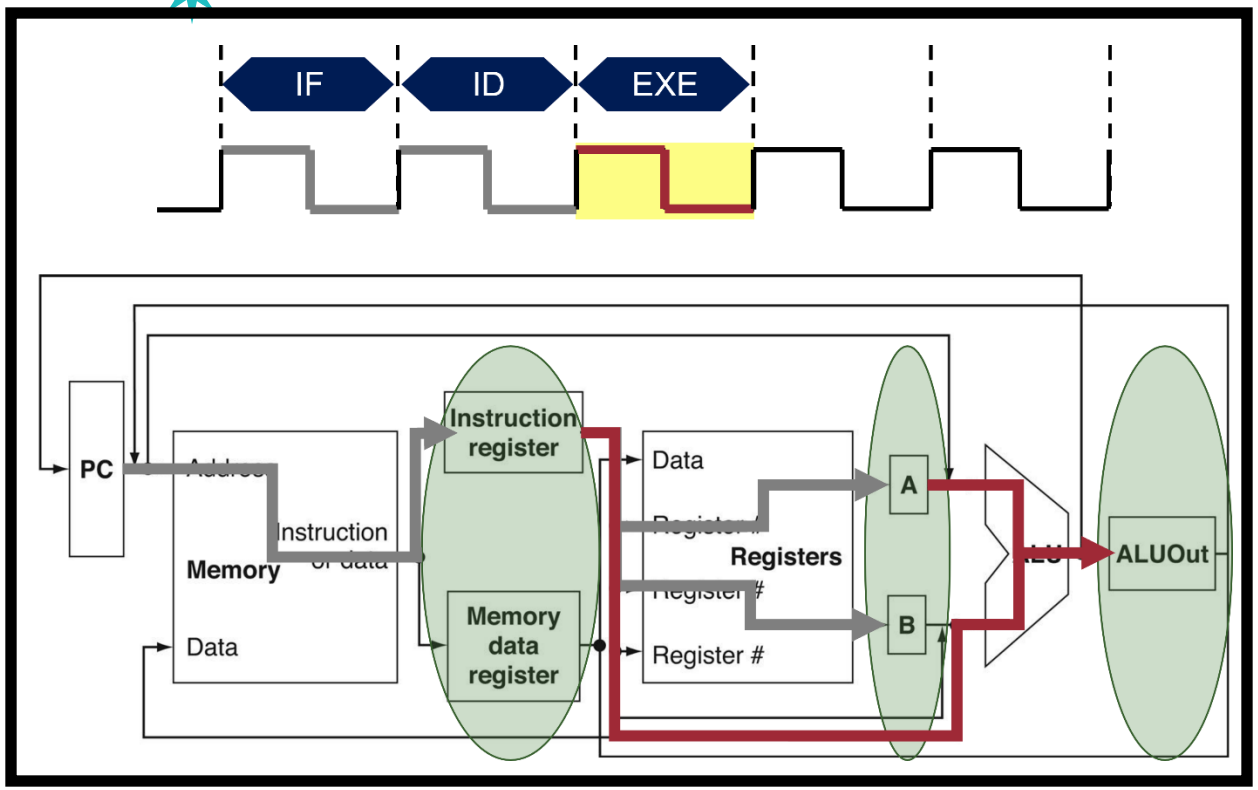




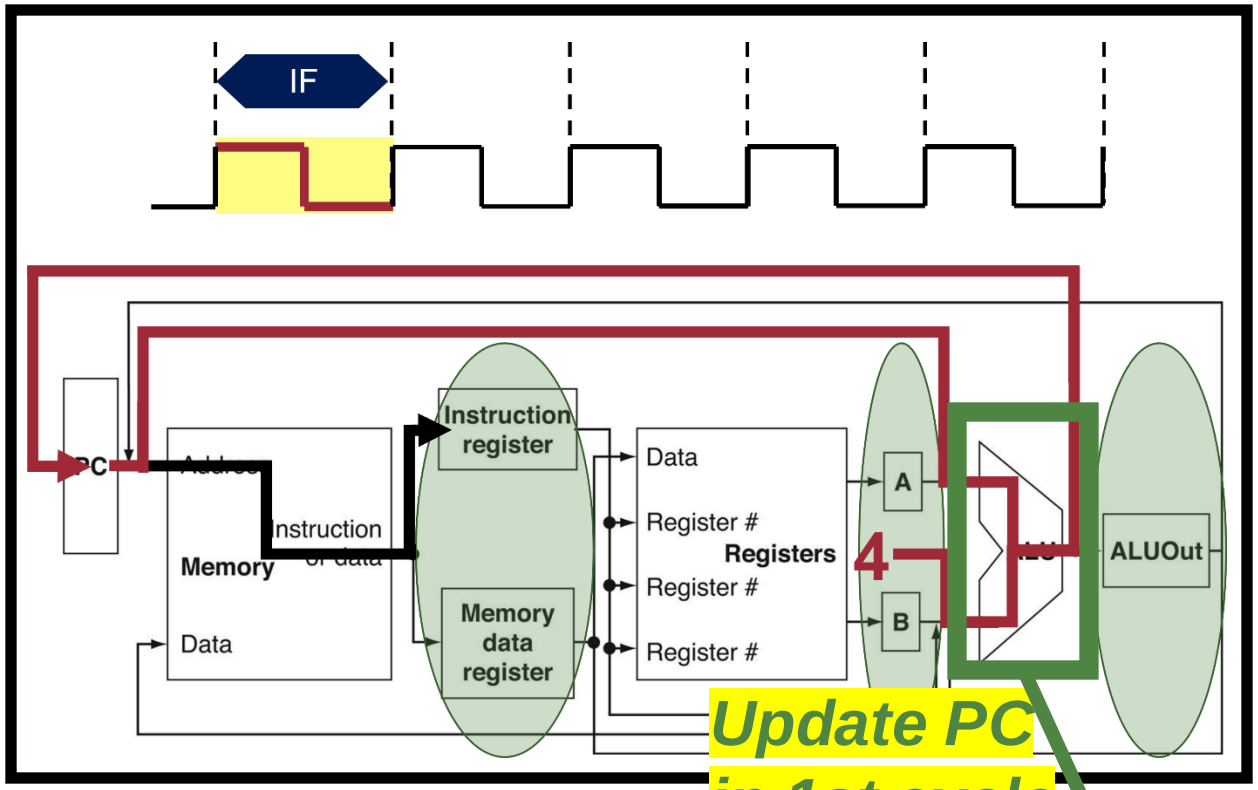
Memory can be shared because it is used
in different clock cycle

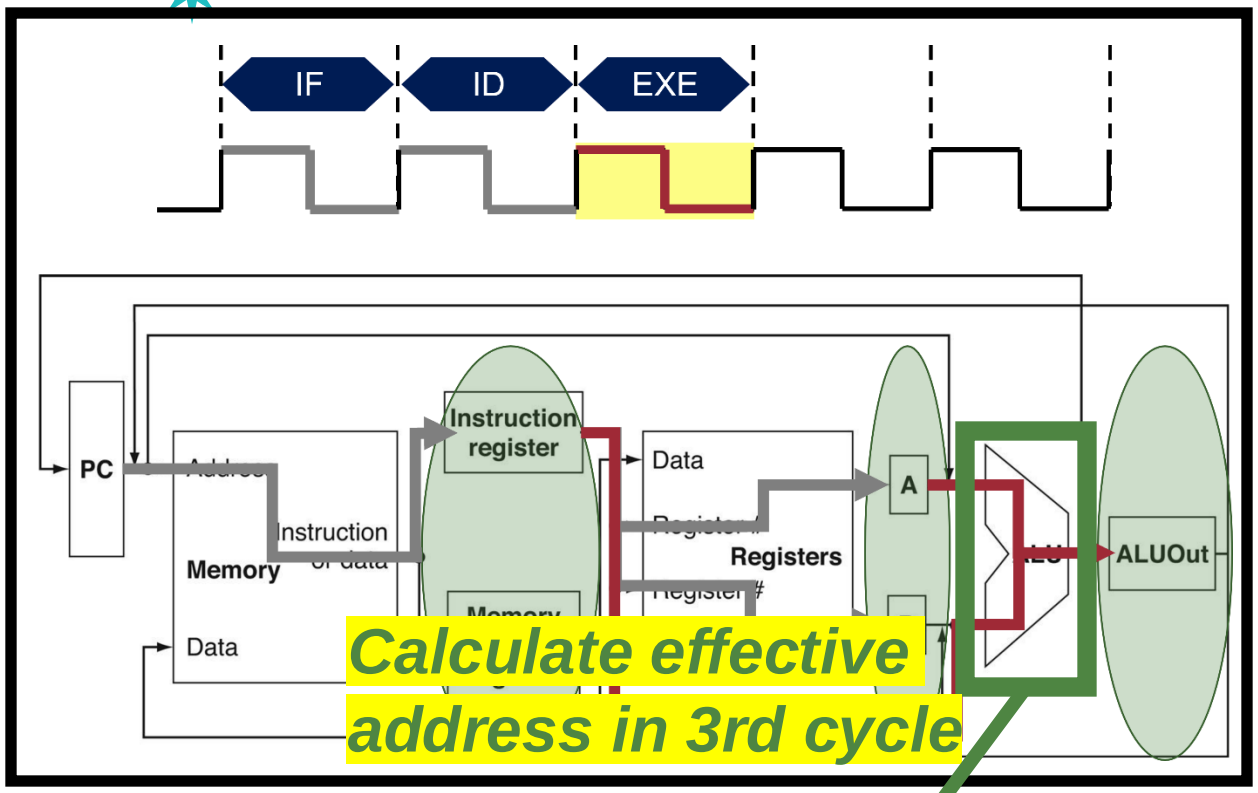
Hardware Sharing: Single ALU





Hardware Sharing: Single ALU





Update PC in 1st cycle

Calculate effective address in 3rd cycle

ALU can be shared because it is used *in different clock cycle*

Multicycle Implementation: Advantages

Compared with single-cycle implementation, performance can be ***increased*** due to:

- Clock cycle period ↓
- Different instructions take different numbers of cycles
- Hardware sharing: single ALU and single memory
  - Reduce the circuit area
  - They can be used for **different purposes** *in different clock cycle*

Multicycle Implementation: Disadvantages

Compared with single-cycle implementation, performance can be ***decreased*** due to:

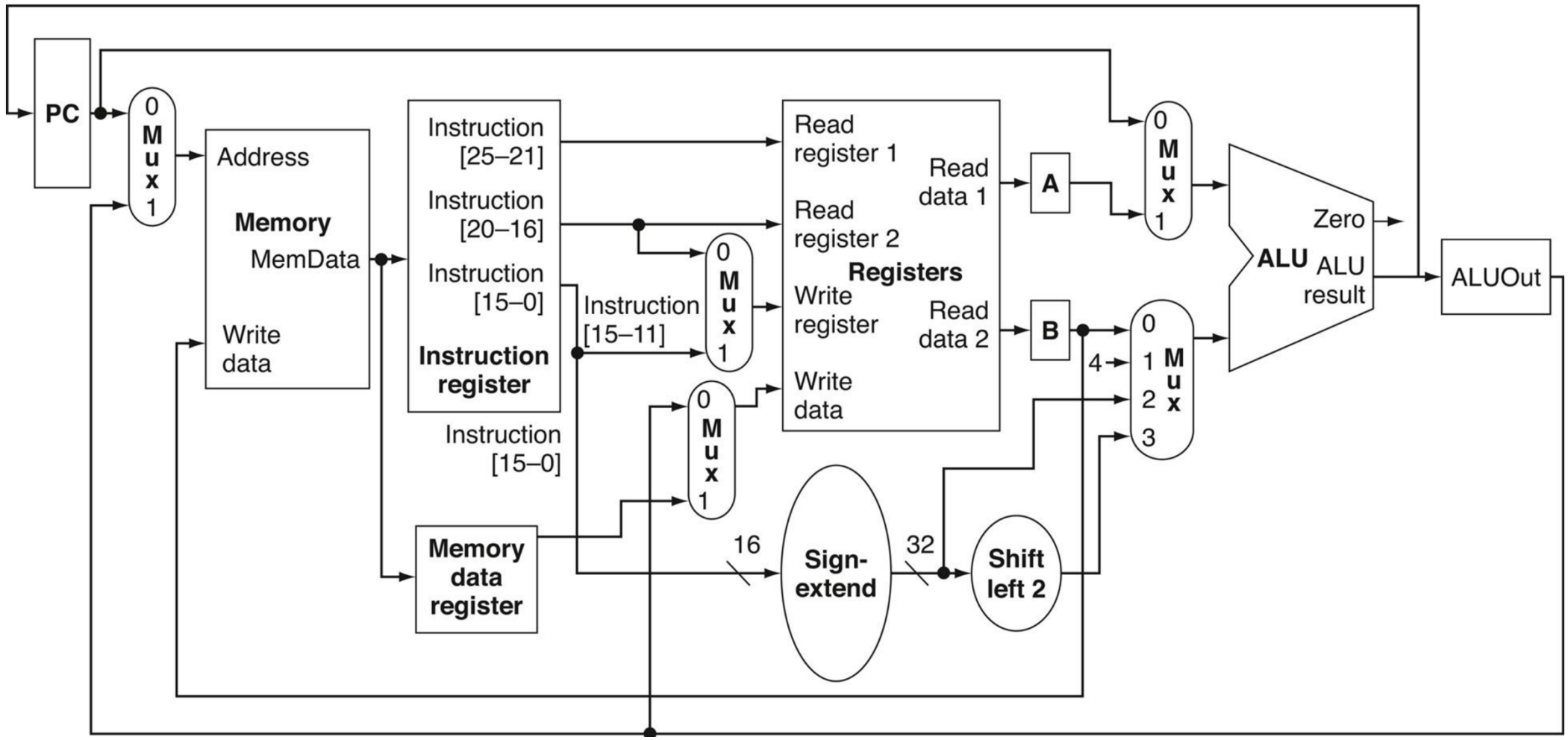
- Additional internal registers
- More multiplexors to shared functional units (We will cover about it)
- More complicated control to consider about *time* (We will cover about it)

Single-cycle vs multicycle: which one is better?

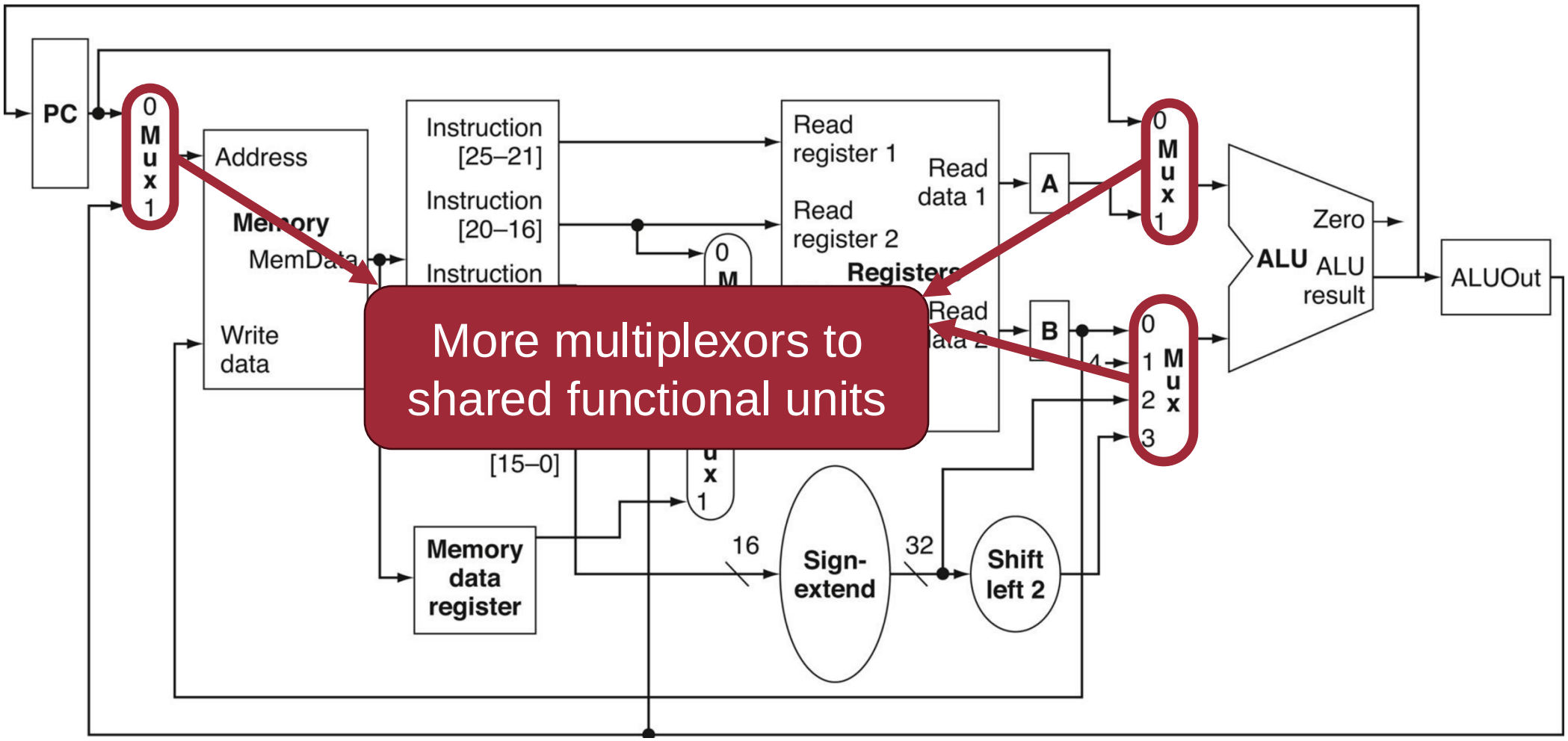
There are trade-offs in terms of time, design complexity, and hardware area

Now, let's look at the details of the multicycle datapath!

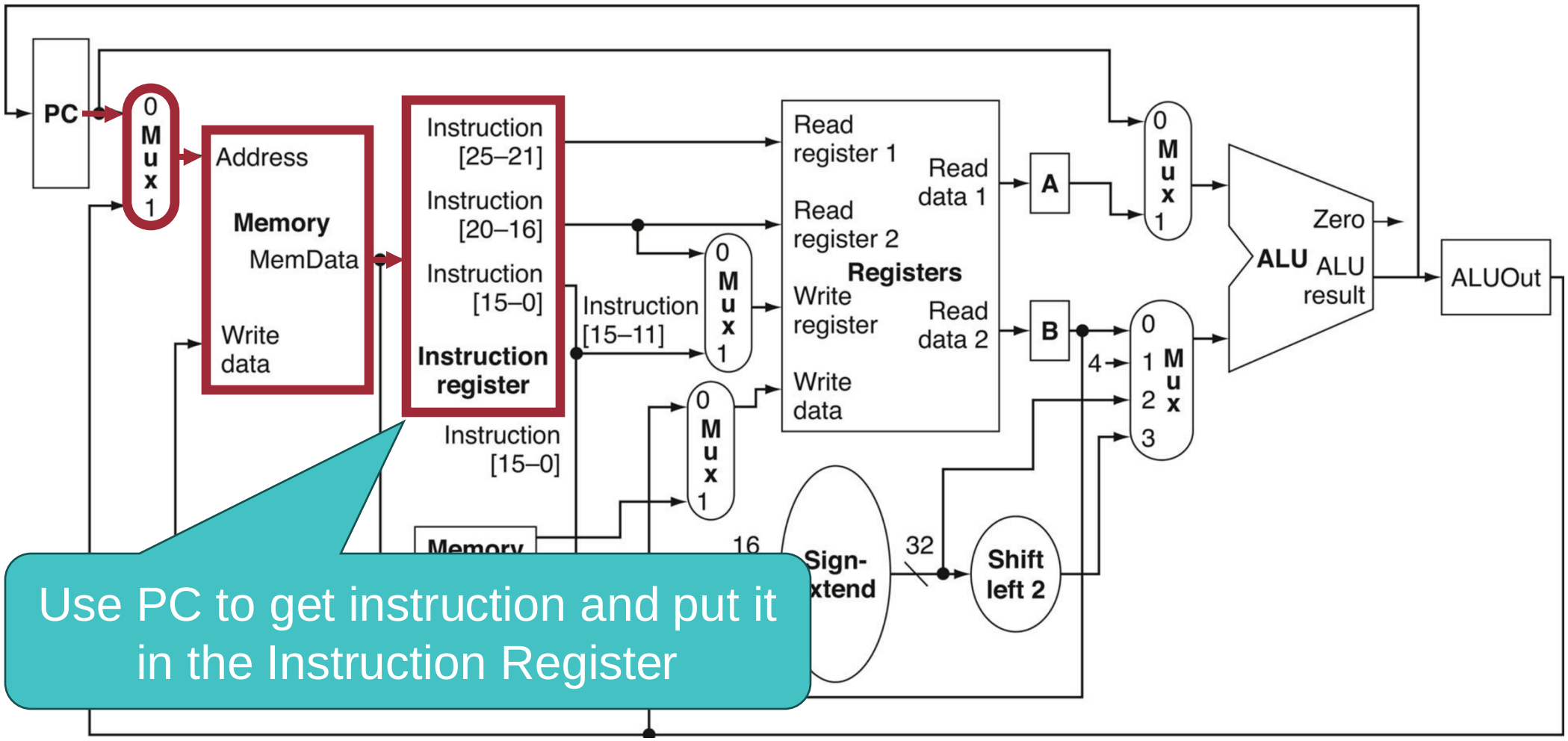
Multicycle Datapath for MIPS



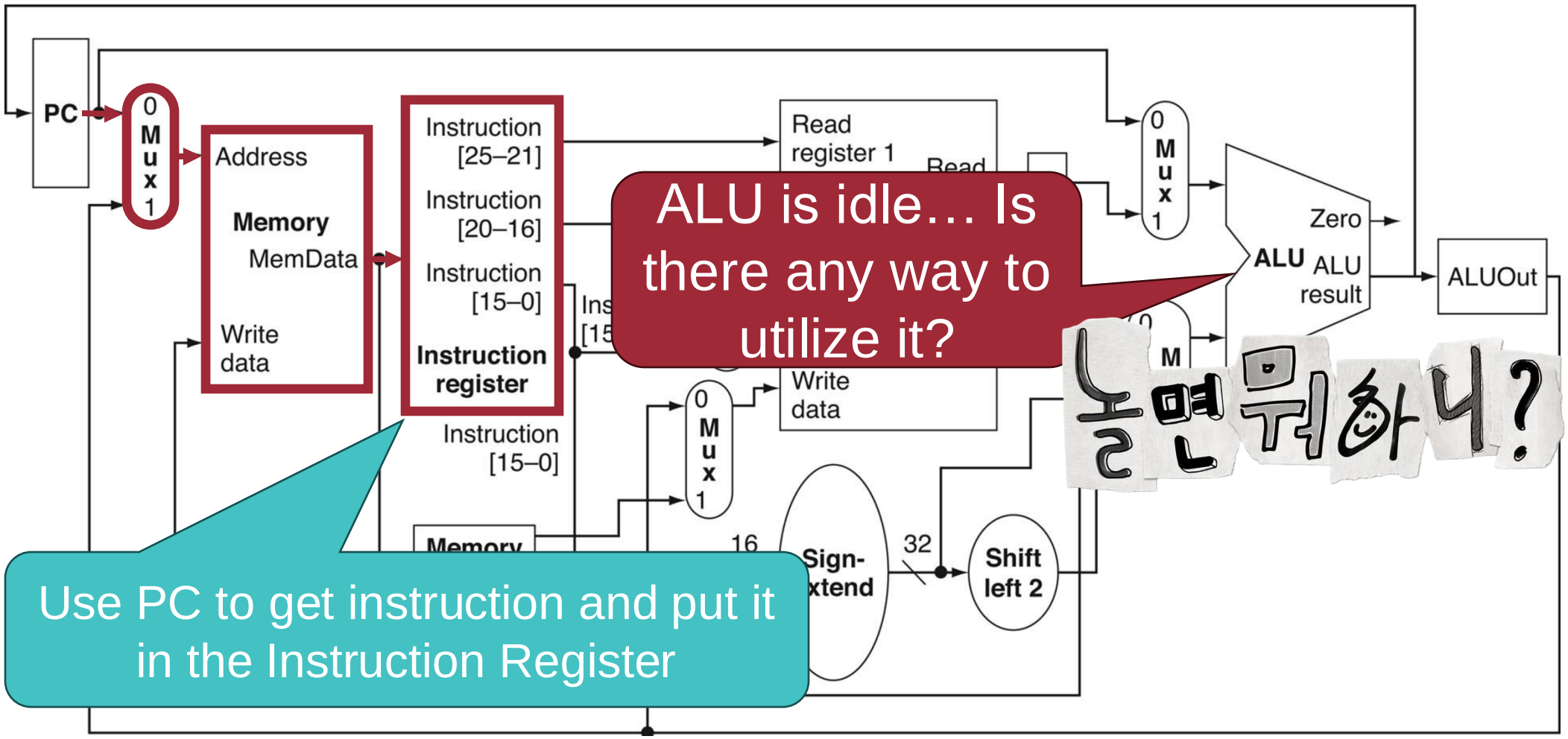
Multicycle Datapath for MIPS



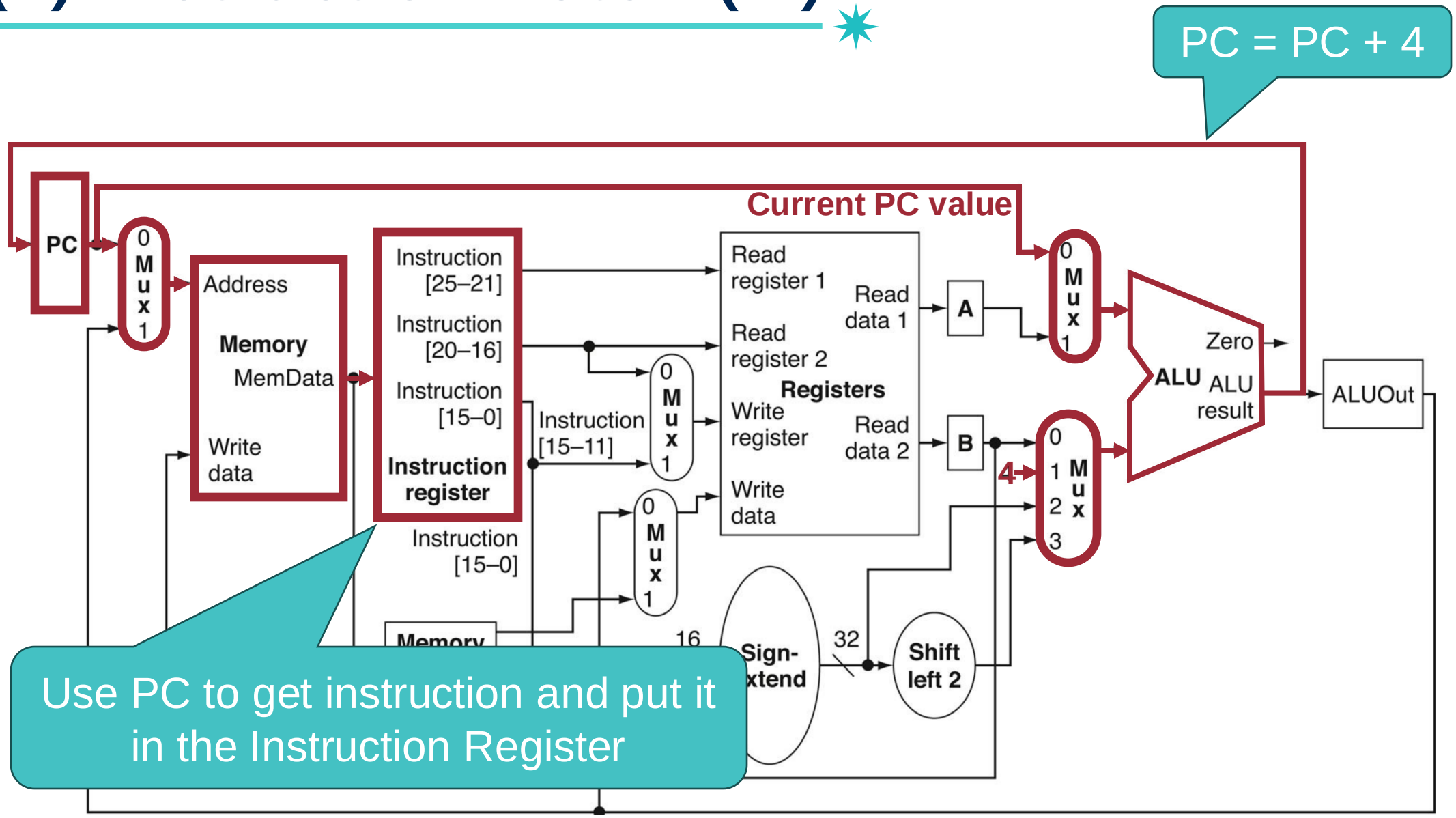
(1) Instruction Fetch (IF)



(1) Instruction Fetch (IF)



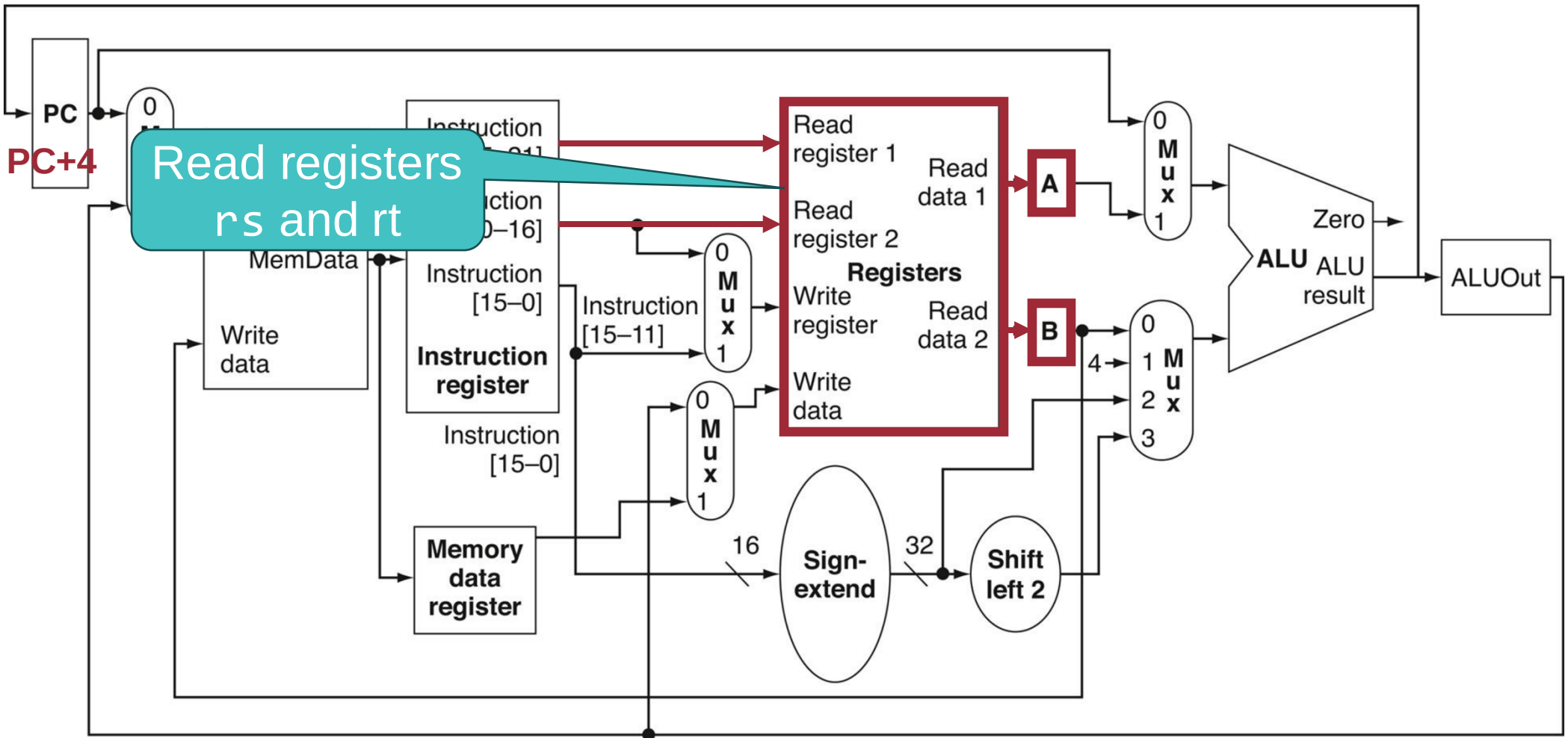
(1) Instruction Fetch (IF)



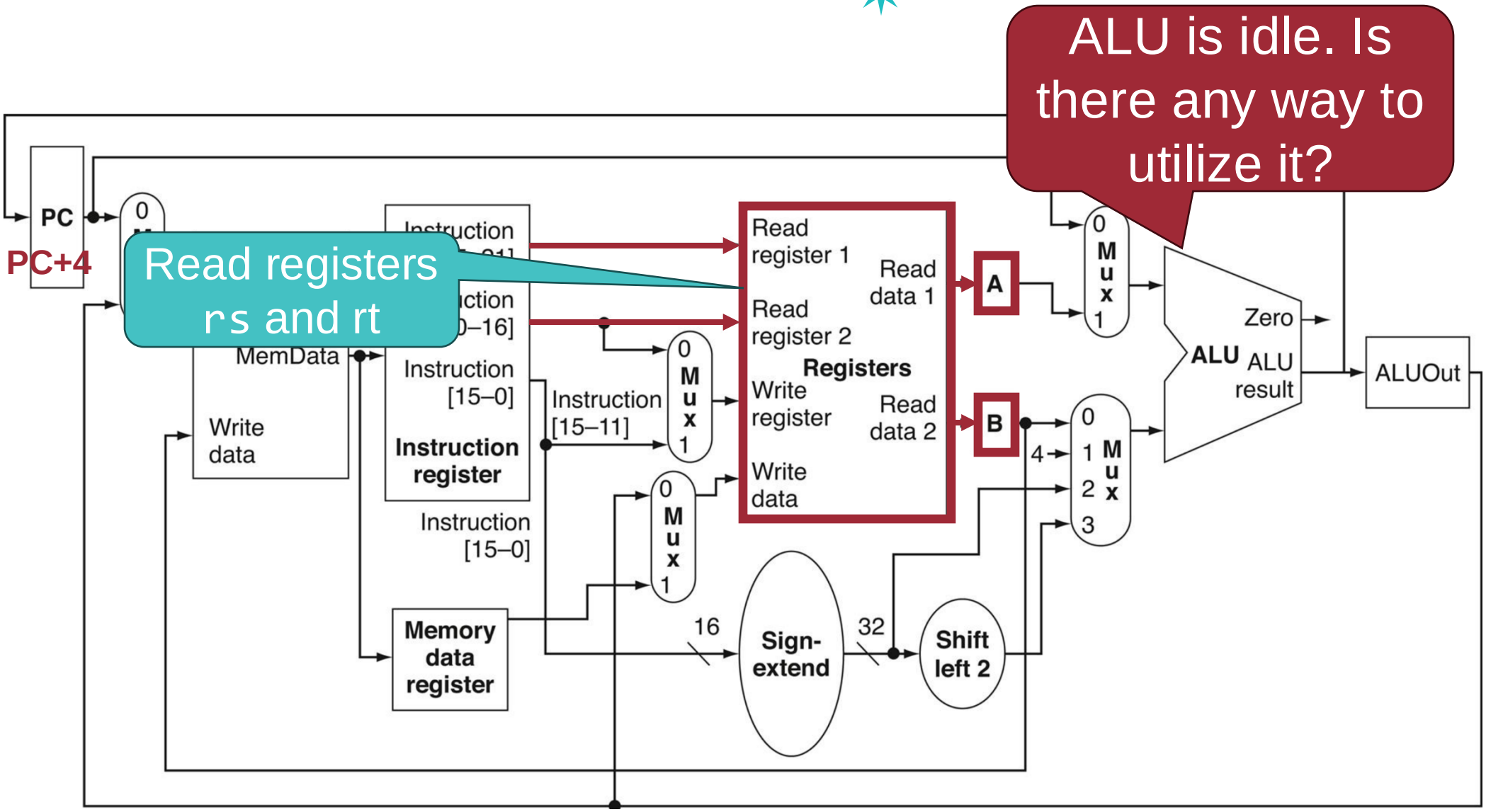
(1) Instruction Fetch (IF): Summary

- Use PC to get instruction and put it in the Instruction Register
- Increment the PC by 4 and put the result back in the PC

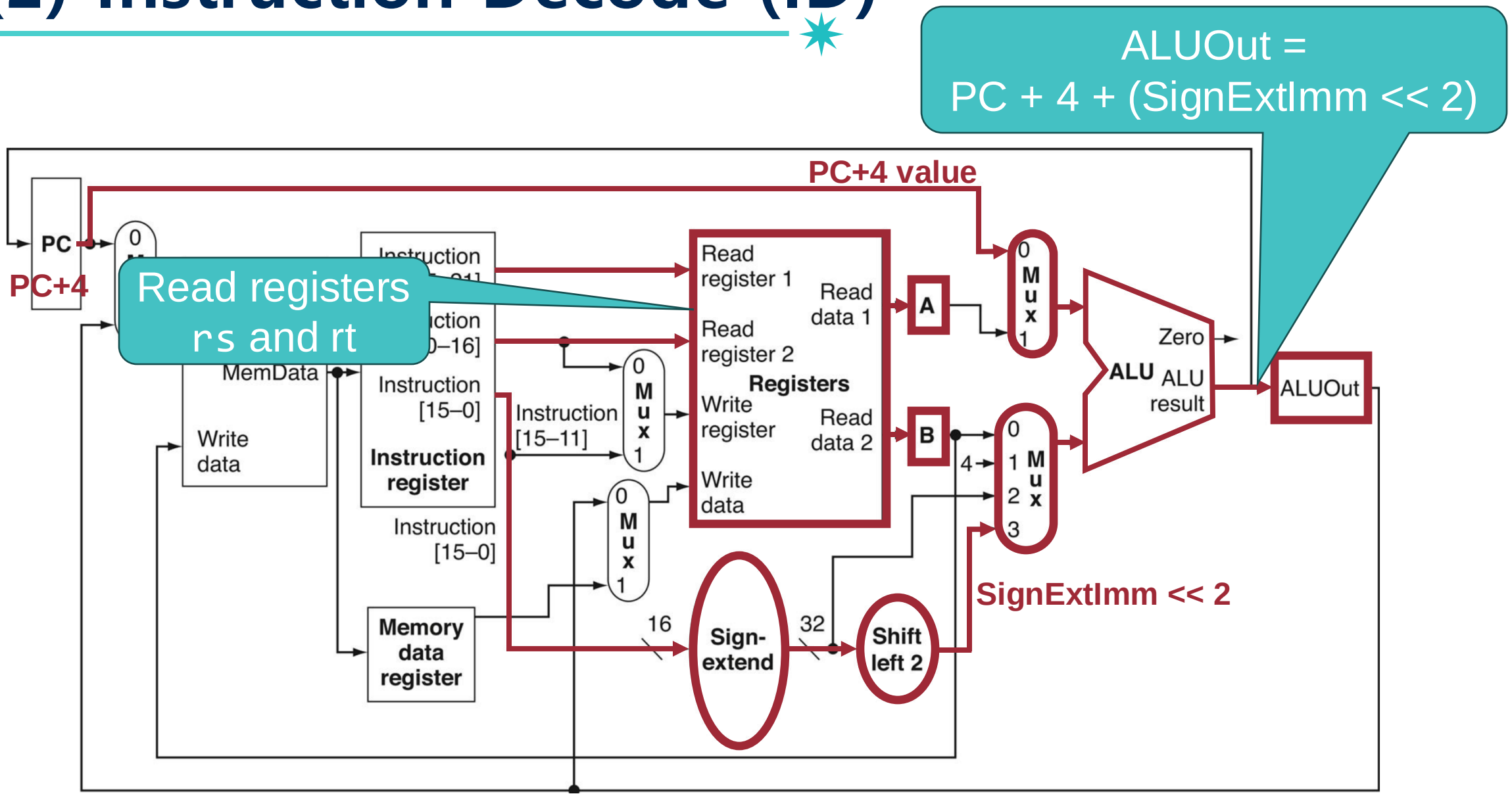
(2) Instruction Decode (ID)



(2) Instruction Decode (ID)



(2) Instruction Decode (ID)



(2) Instruction Decode (ID): Summary

- Read registers `rs` and `rt`

- Compute the branch address in advance

IF and ID stages are common for all types of instructions

From the next stage, the actions vary depending on the type of instruction

Execution (EXE) Stage Summary

ALU is performing one of three functions, based on instruction type

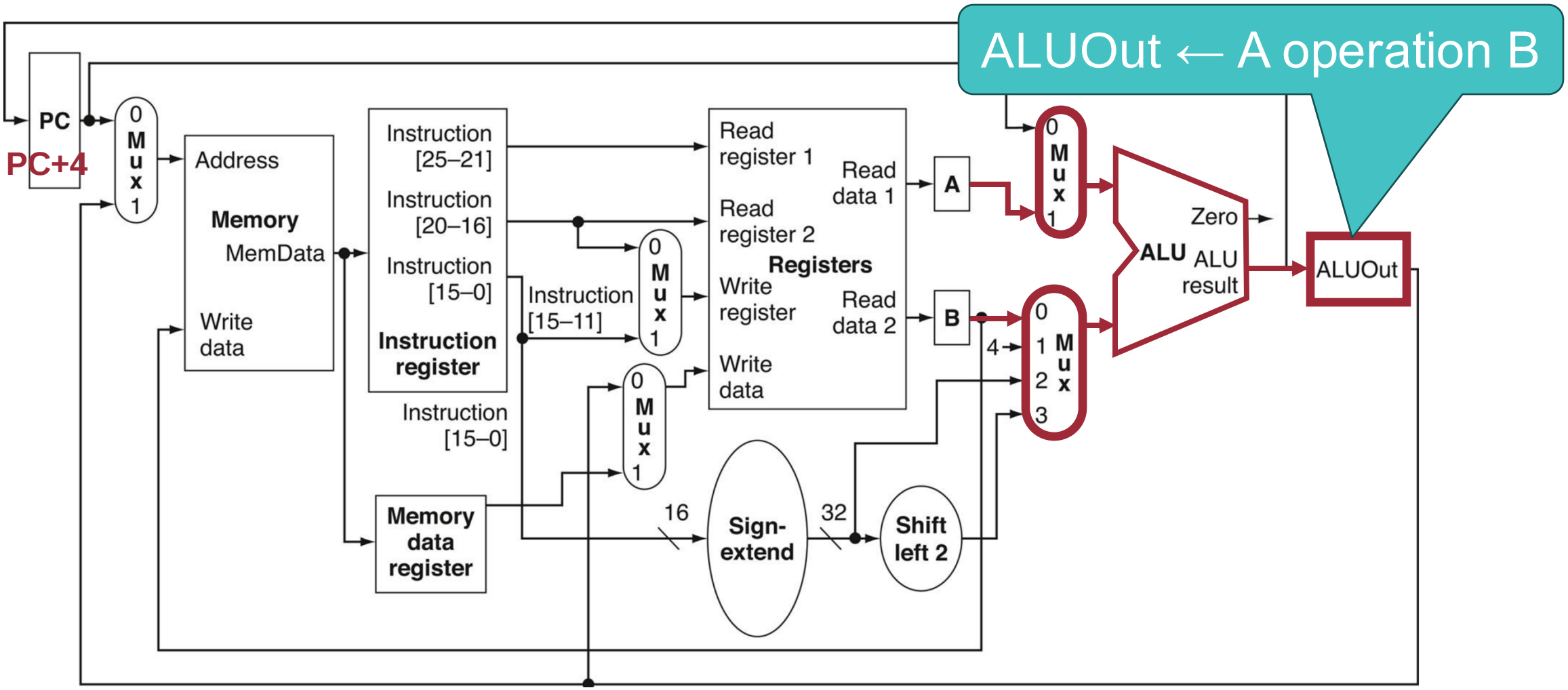
Arithmetic / Logic Instructions (R-format)

ALUOut ← A operation B

`lw, sw` (I-format)

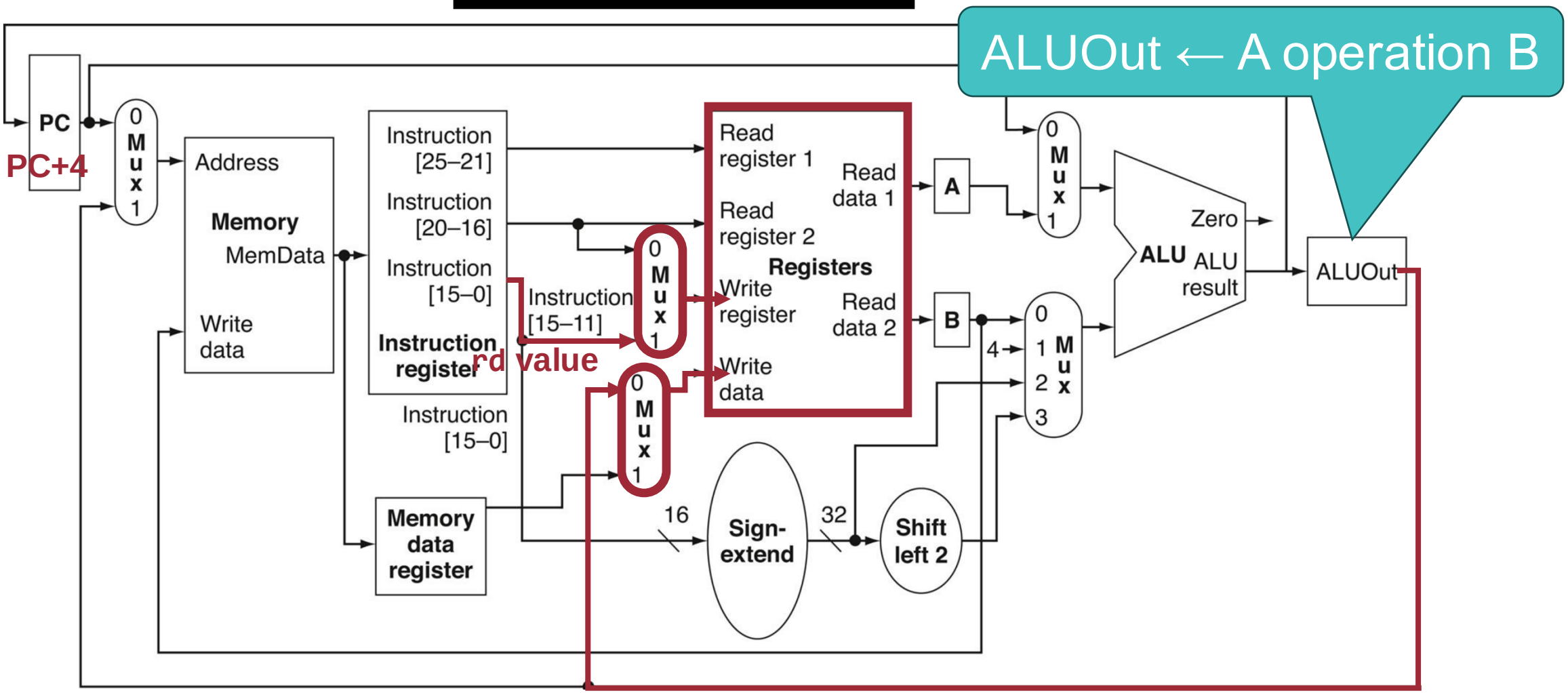
`beq` (I-format)

R-formation – (3) Execution (EXE)



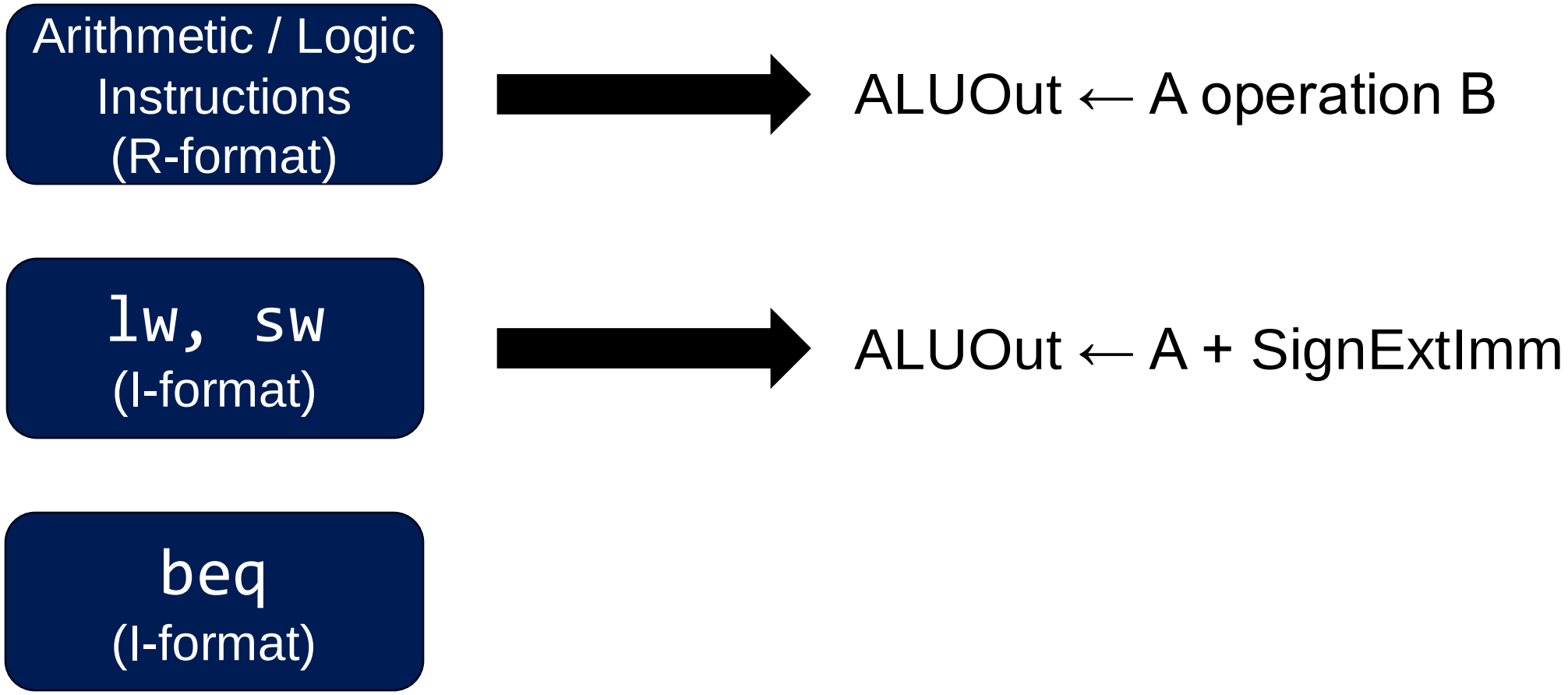
R-formation – (4) Register Write-back (WB)

Completion!

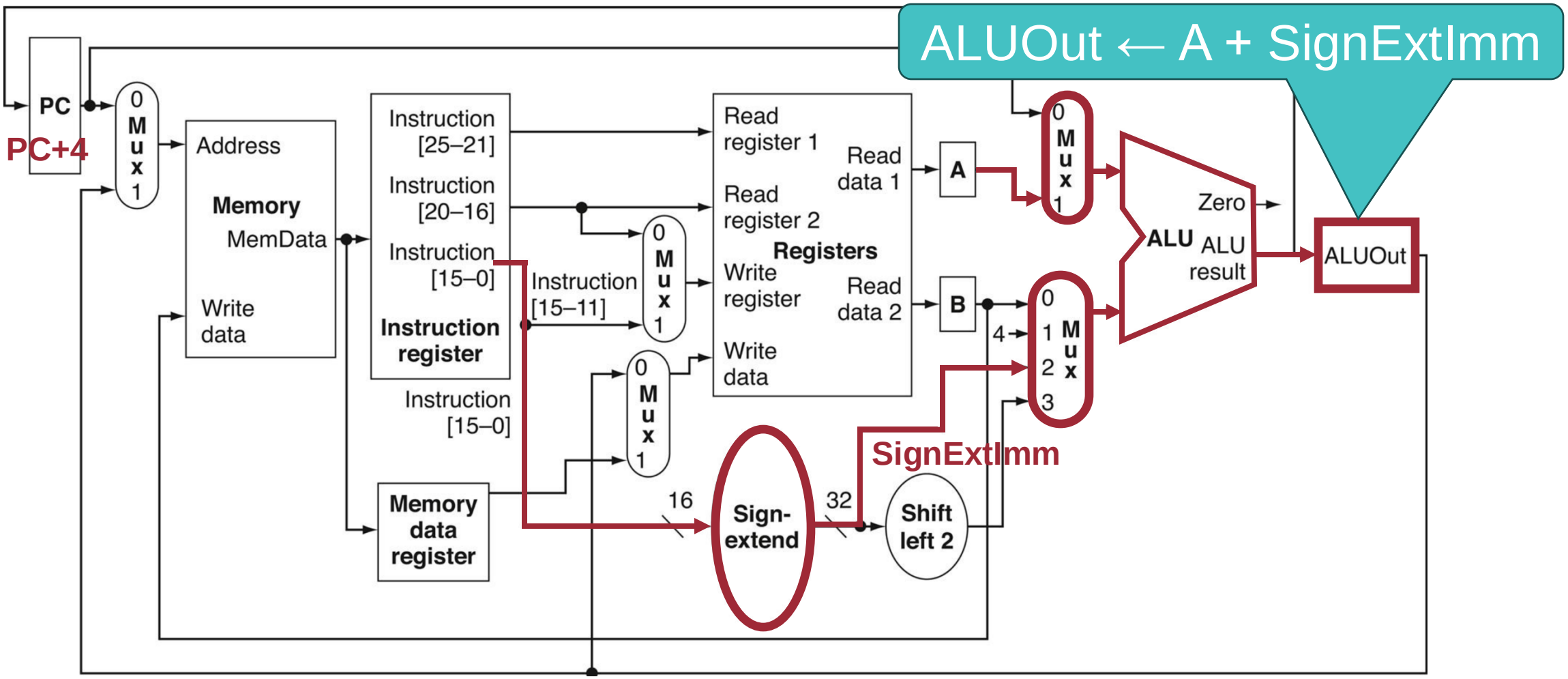


Execution (EXE) Stage Summary

ALU is performing one of three functions, based on instruction type

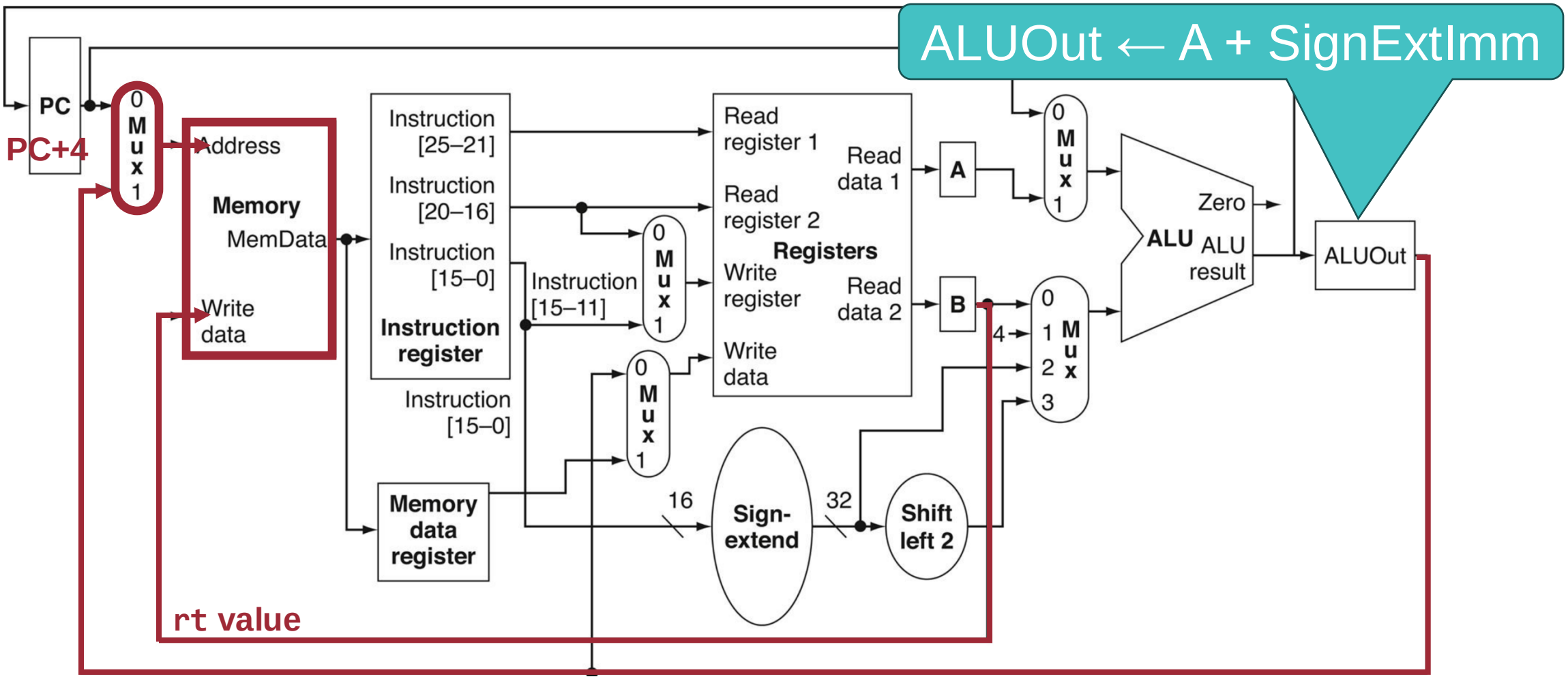


lw and sw – (3) Execution (EXE)

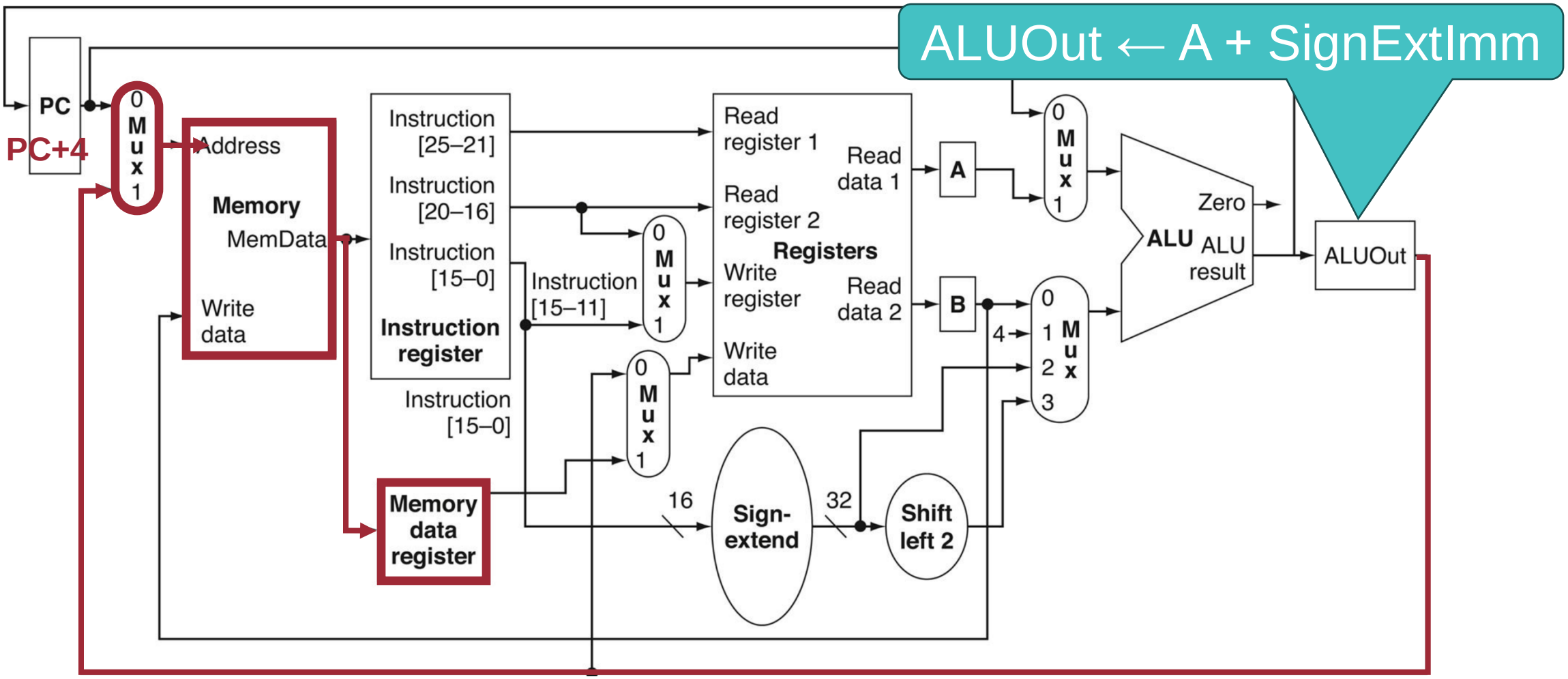


sw – (4) Memory Access (MEM)

Completion!

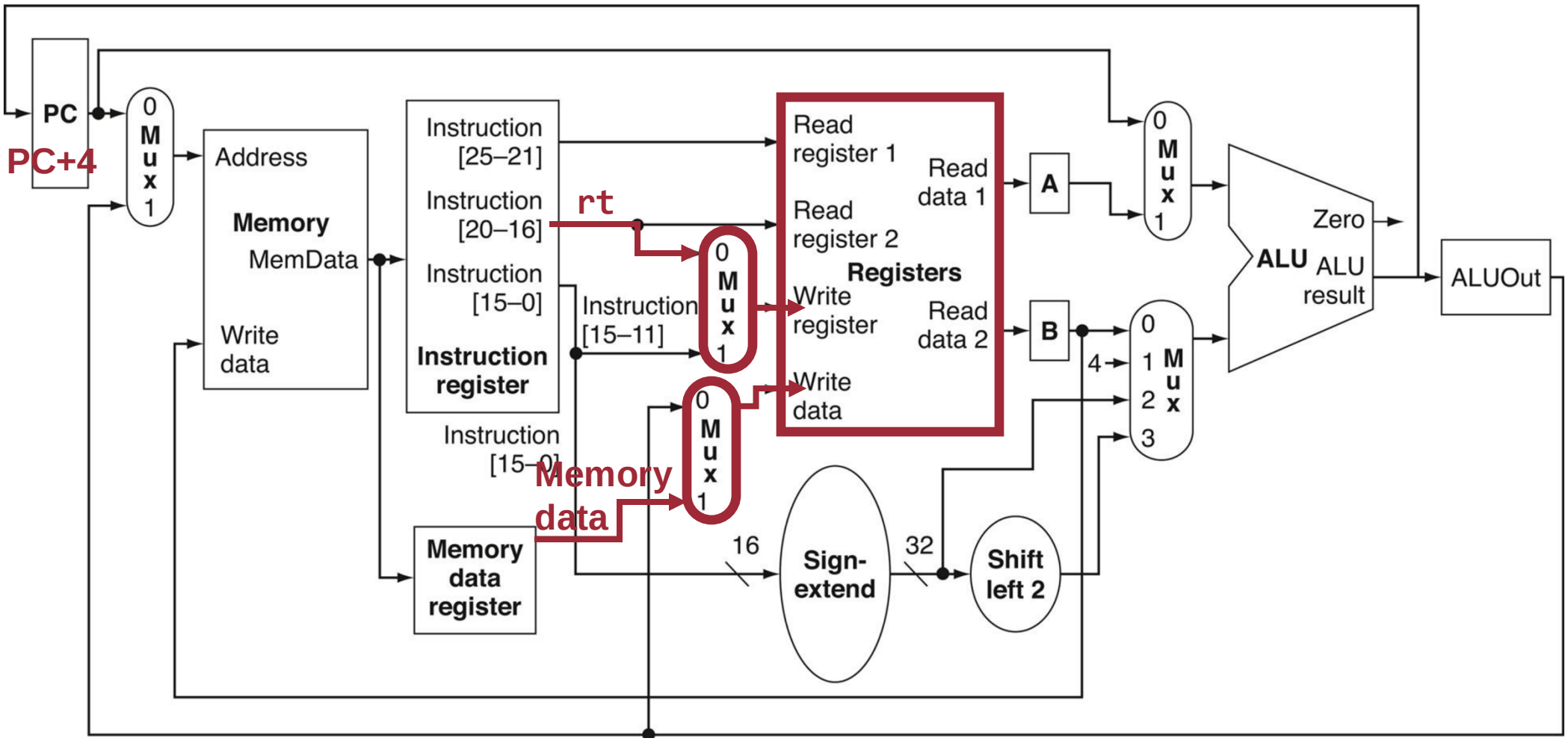


lw – (4) Memory Access (MEM)



lw – (5) Register Write-back (WB)

Completion!



Execution (EXE) Stage Summary

ALU is performing one of three functions, based on instruction type

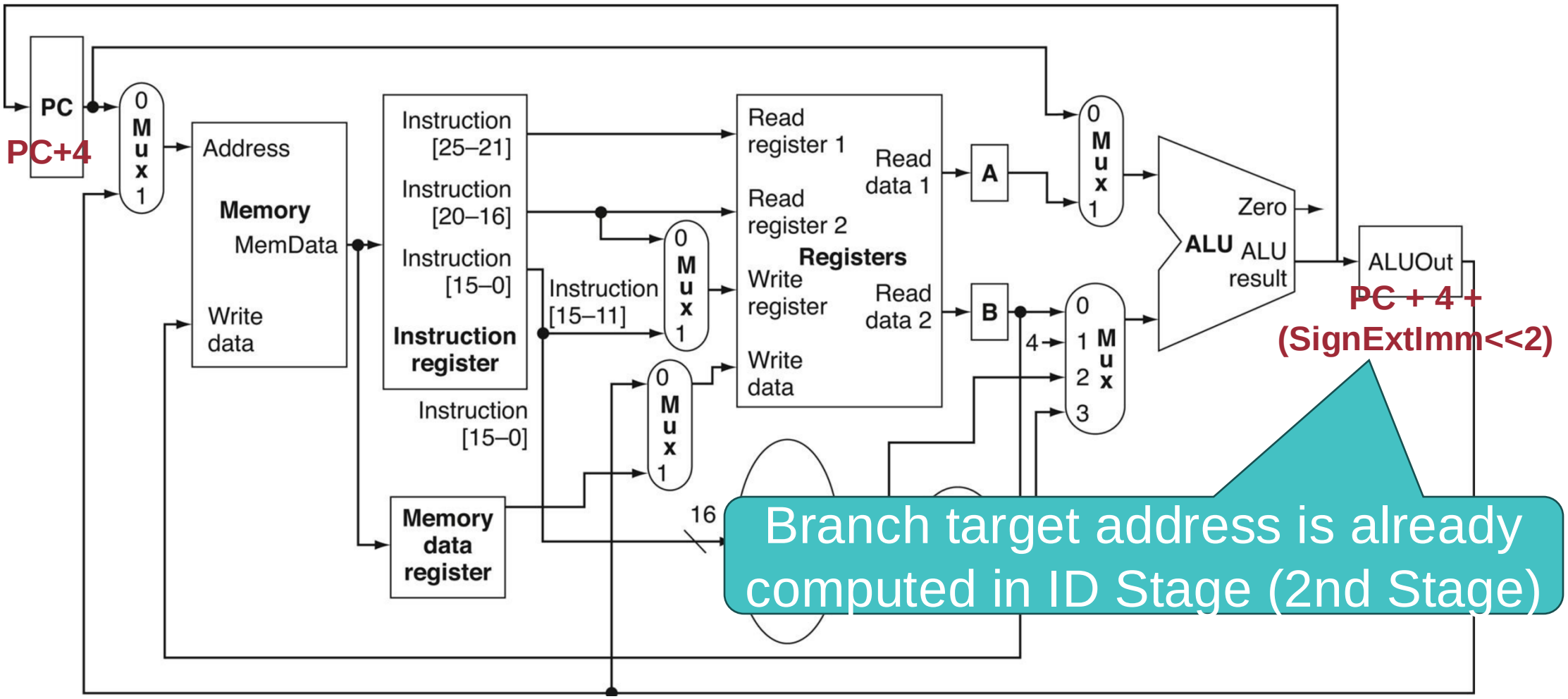
Arithmetic / Logic Instructions (R-format) → ALUOut ← A operation B

`lw, sw` (I-format) → ALUOut ← A + SignExtImm

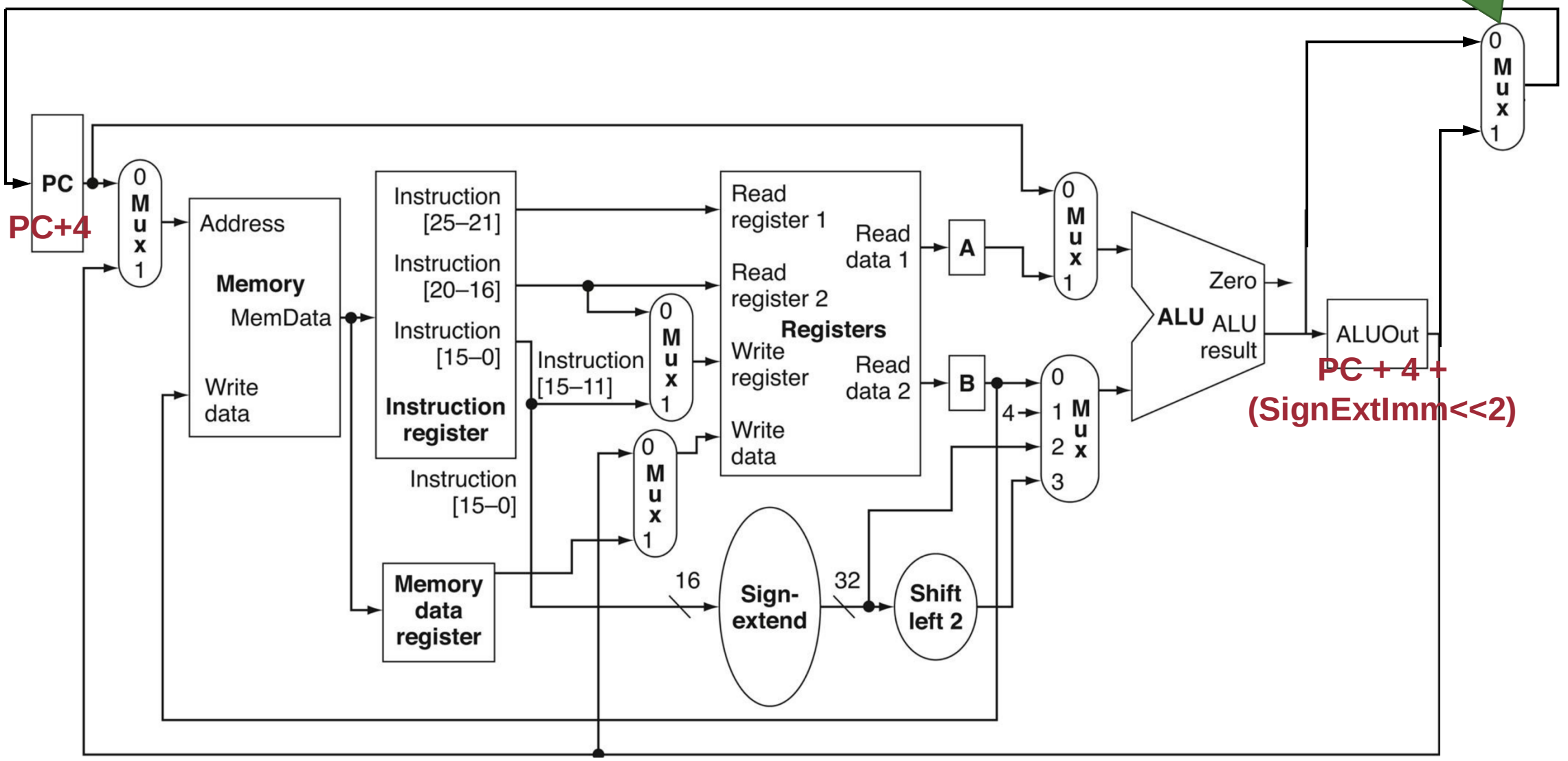
`beq` (I-format) →

```
if (zero):
    PC ← ALUOut
```

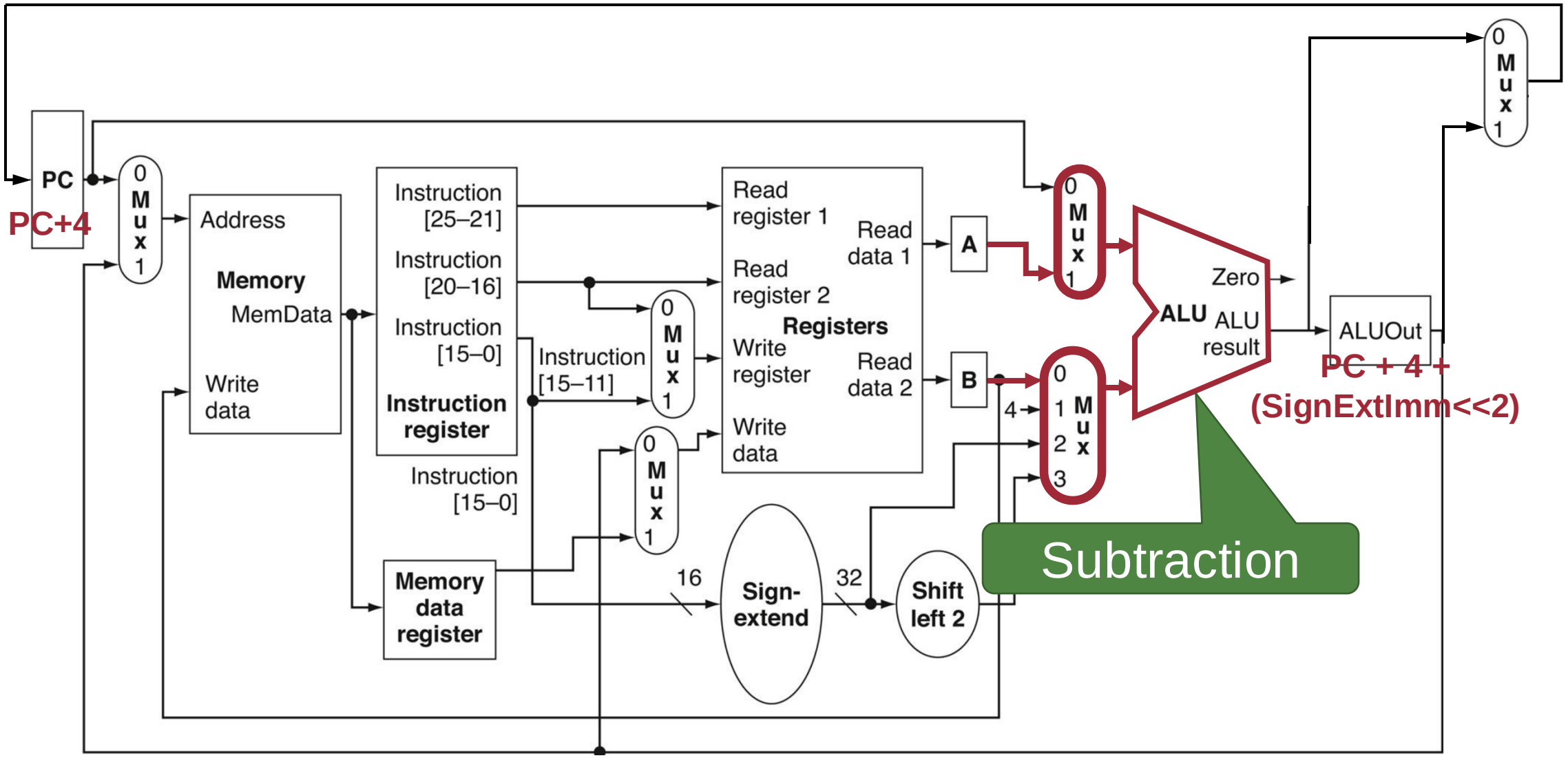
beq – (3) Execution (EXE)



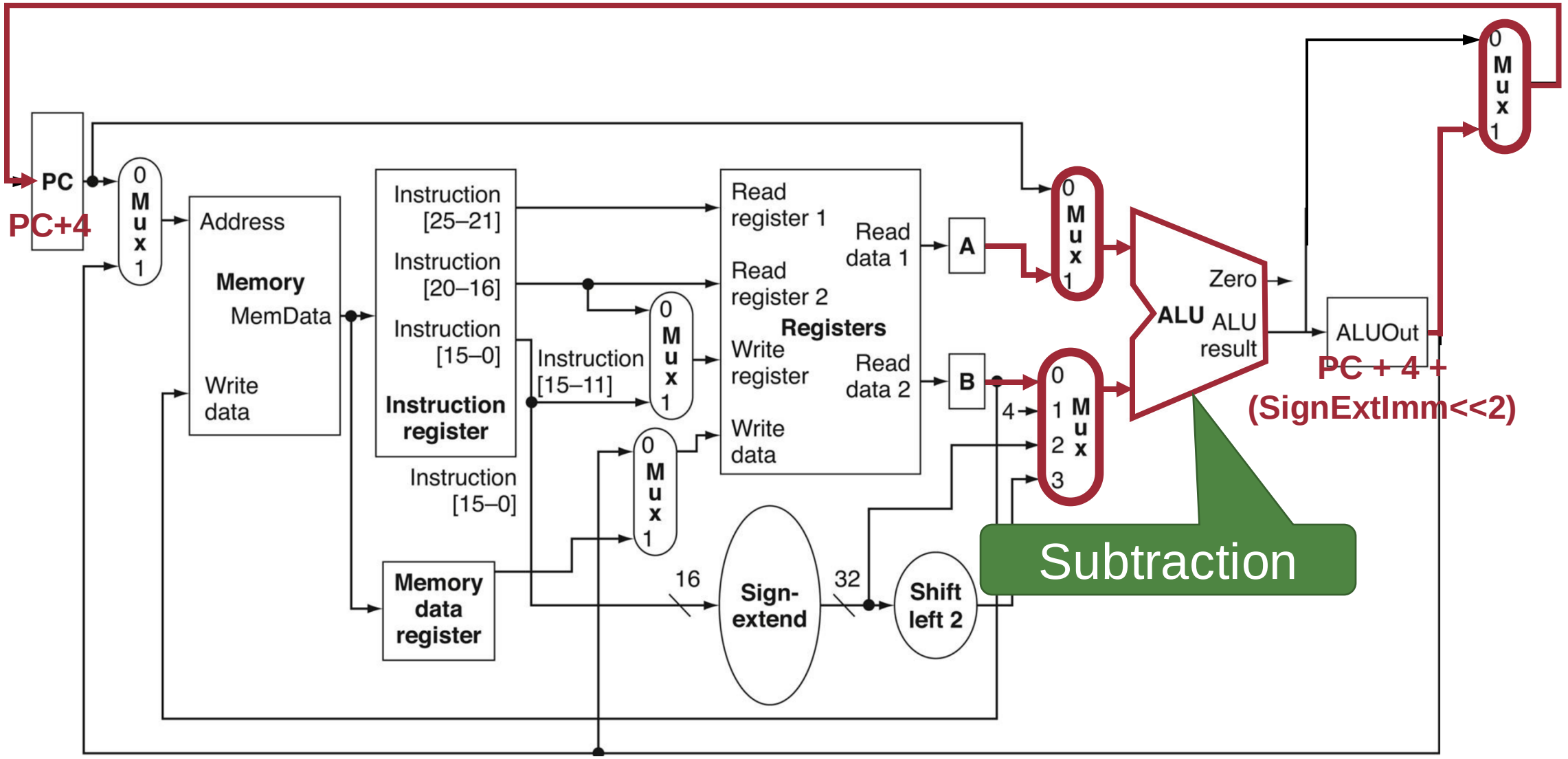
beq – (3) Execution (EXE)



beq – (3) Execution (EXE)

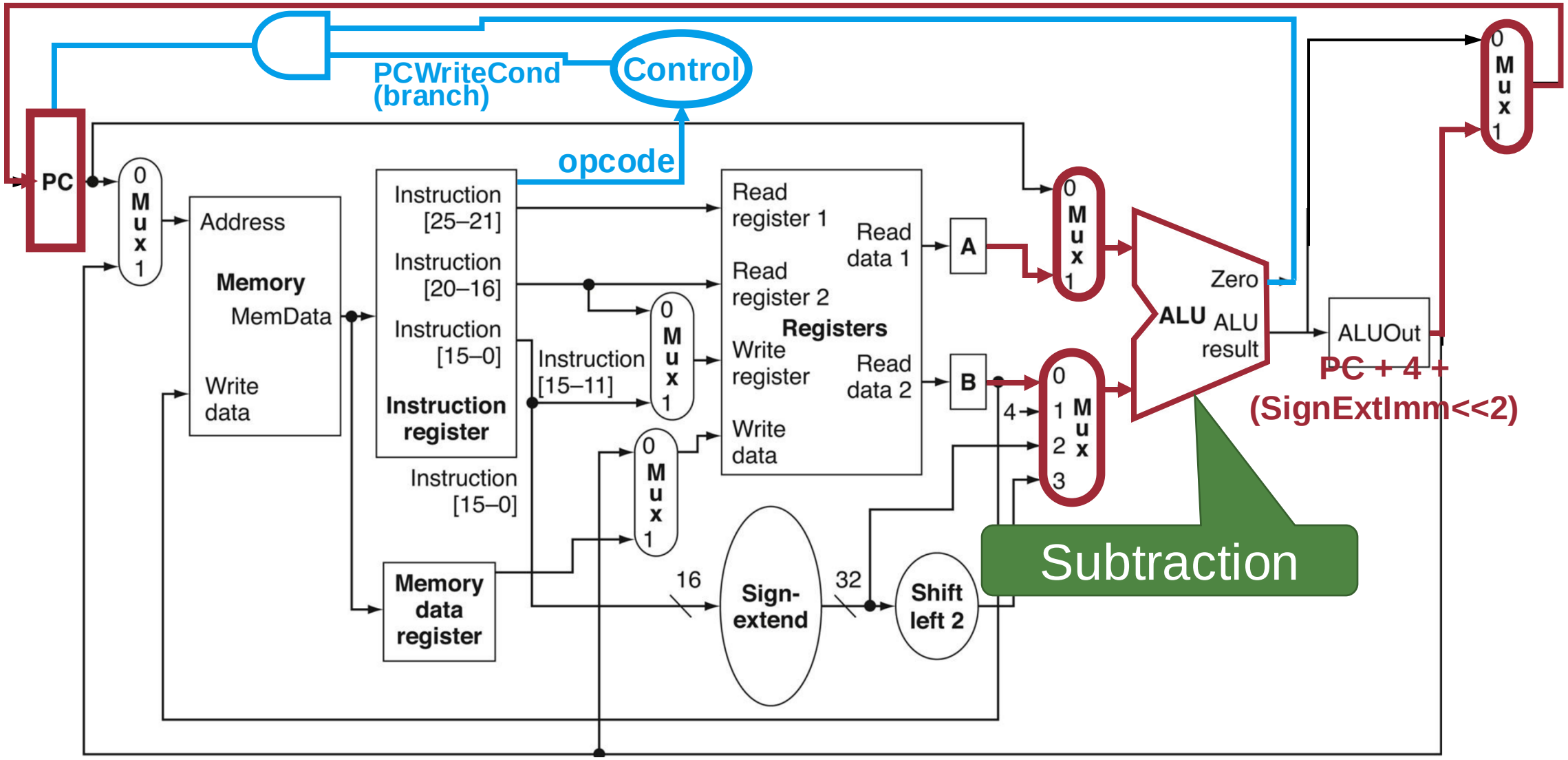


beq – (3) Execution (EXE)

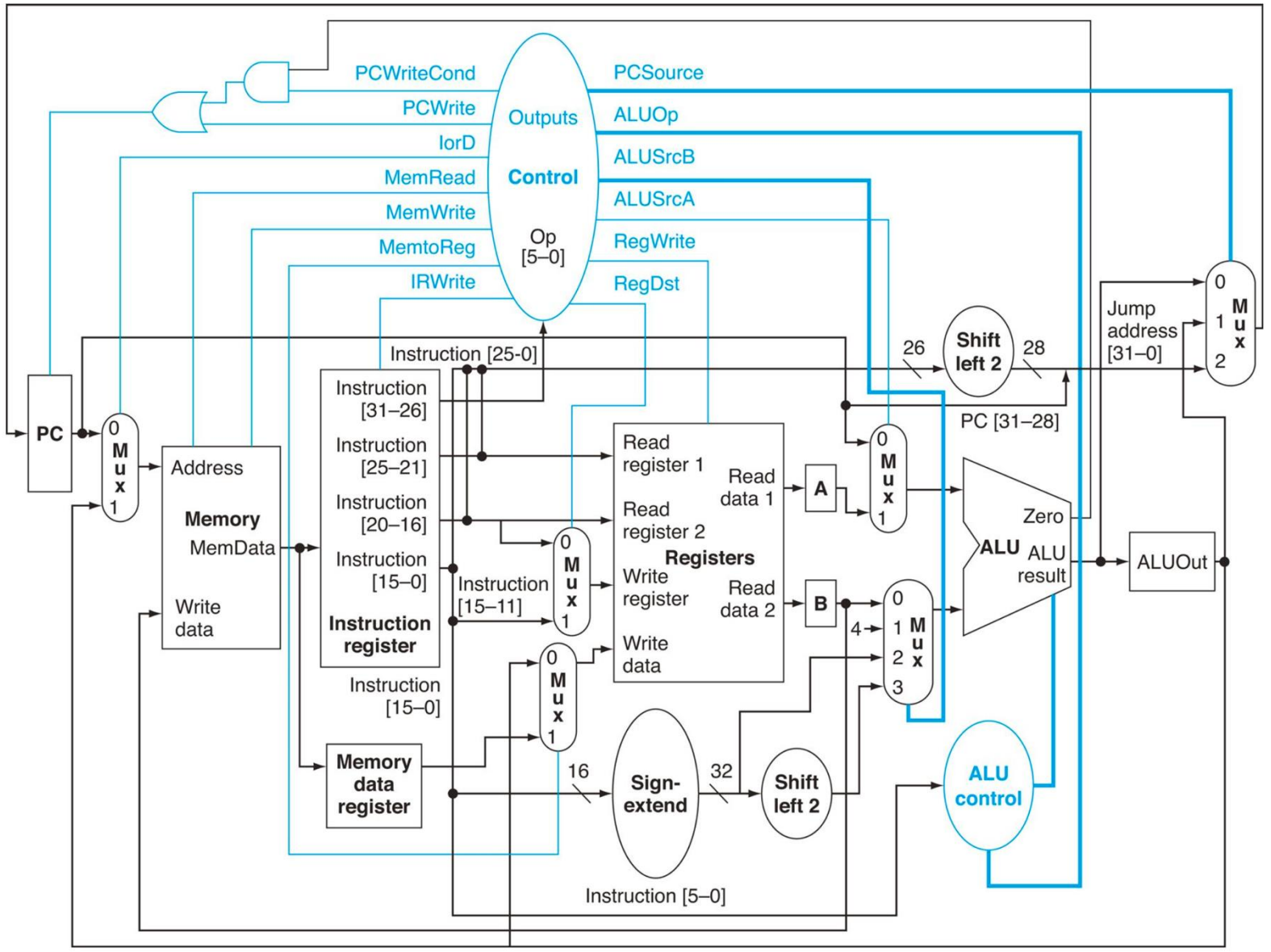


beq – (3) Execution (EXE)

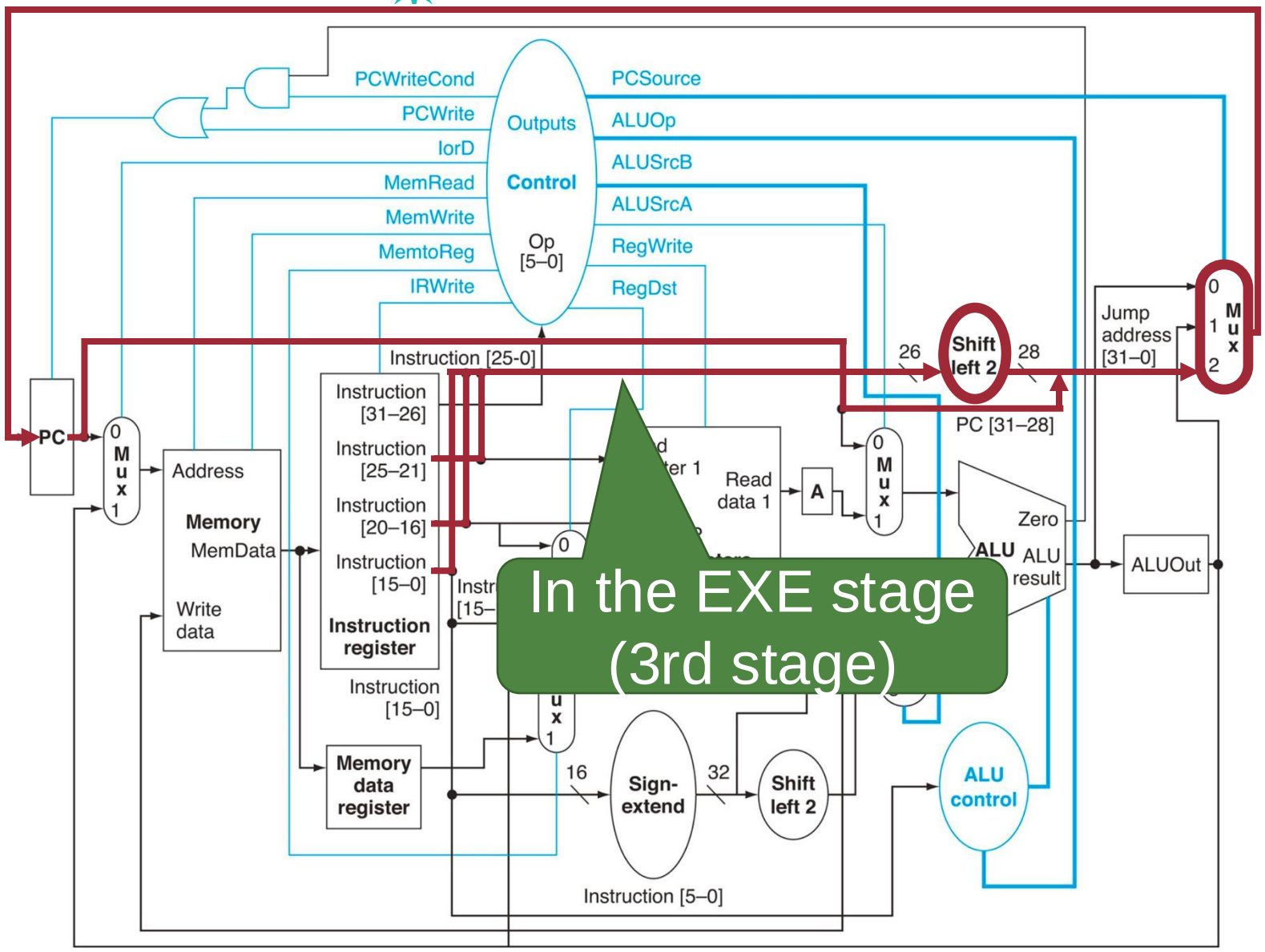
Completion!



Multicycle Datapath: Complete View



Multicycle Datapath: Jump Instruction



Multicycle Datapath: Summary

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			

Multicycle Datapath: Summary

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[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)			

Multicycle Datapath: Summary

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[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/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}

Multicycle Datapath: Summary

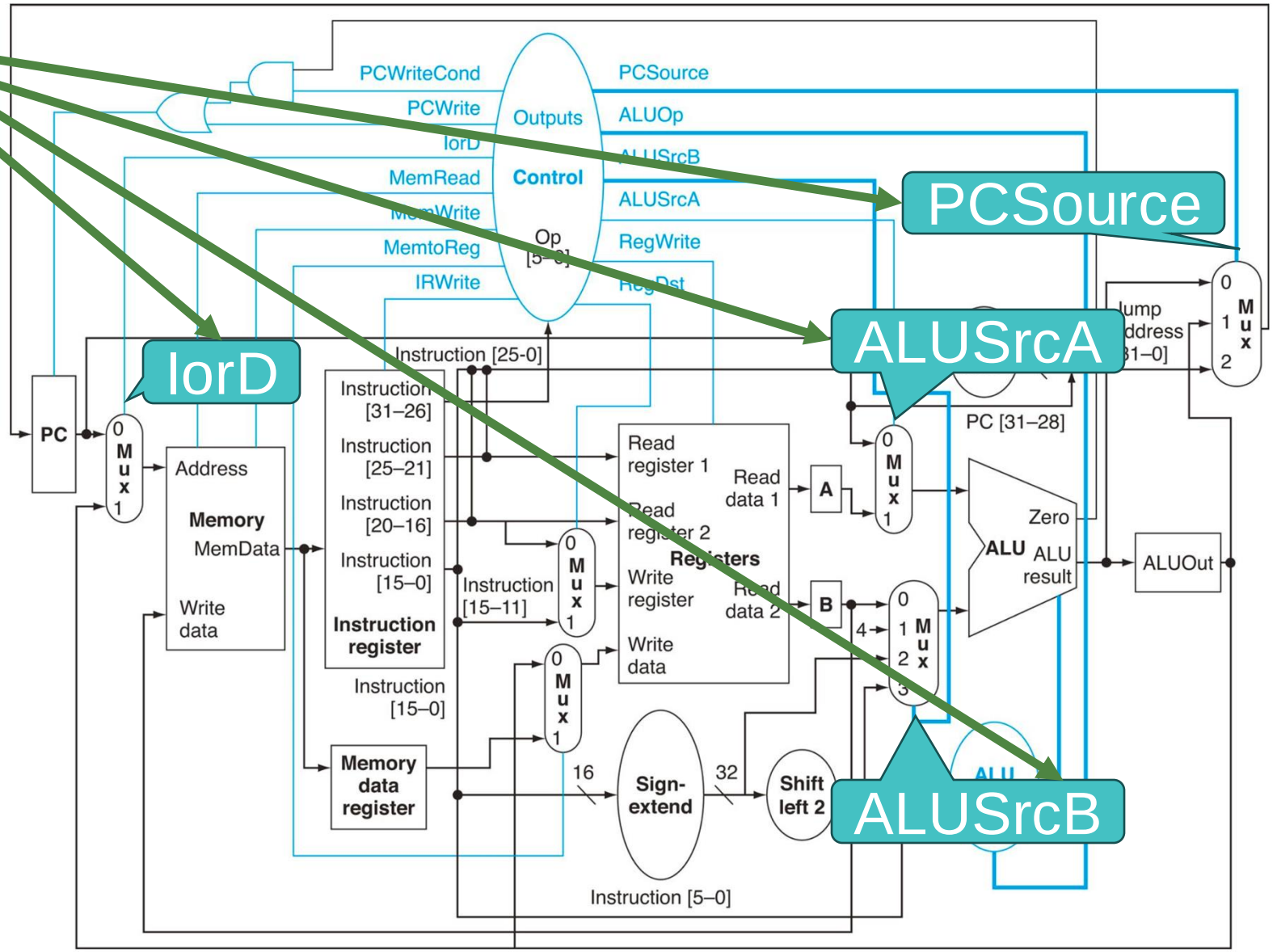
Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		

Multicycle Datapath: Summary

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[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/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

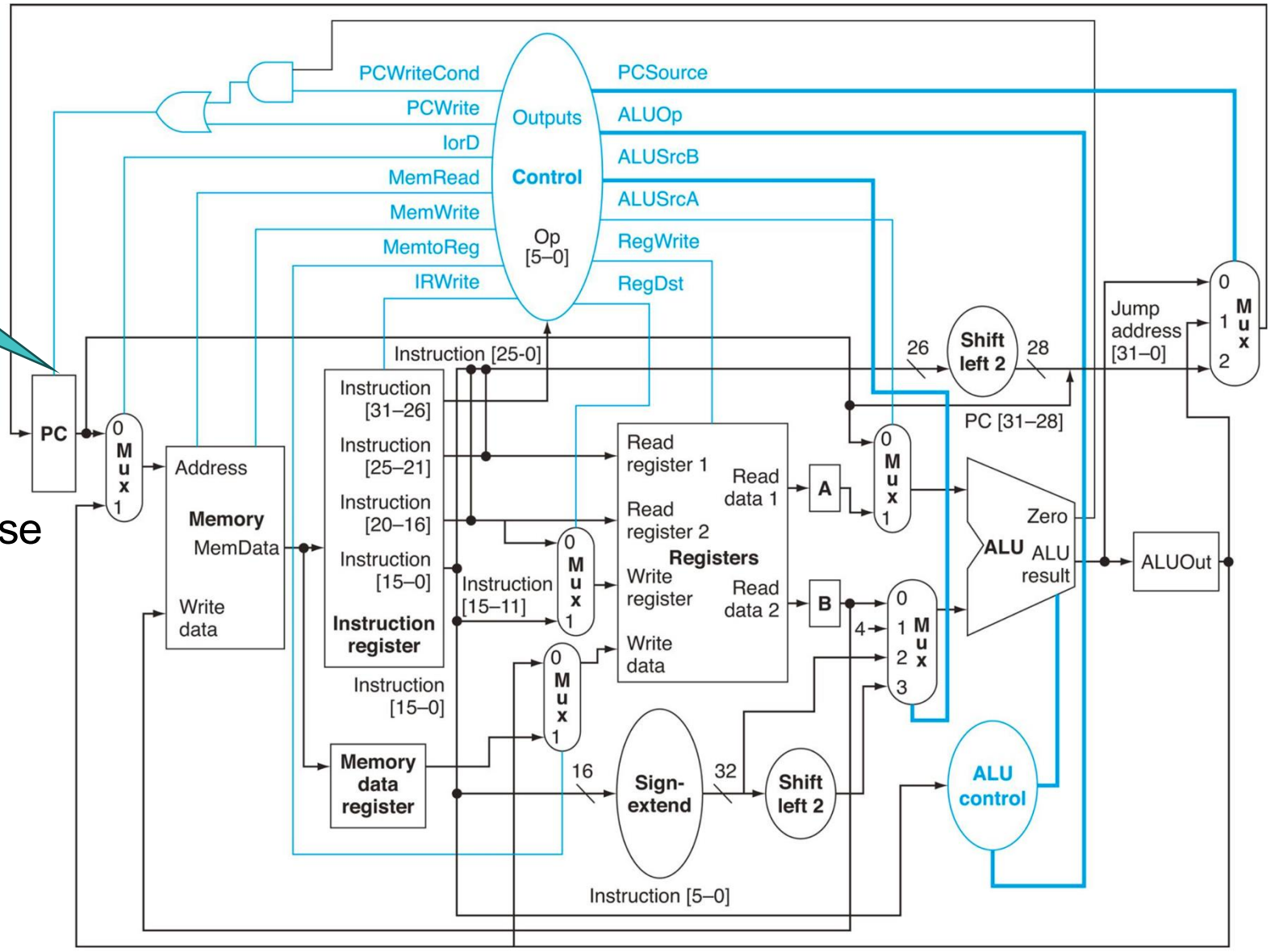
Multicycle Implementation: Control Signals

Introduced by the newly added mux



Multicycle Implementation: Control Signals

PCWrite

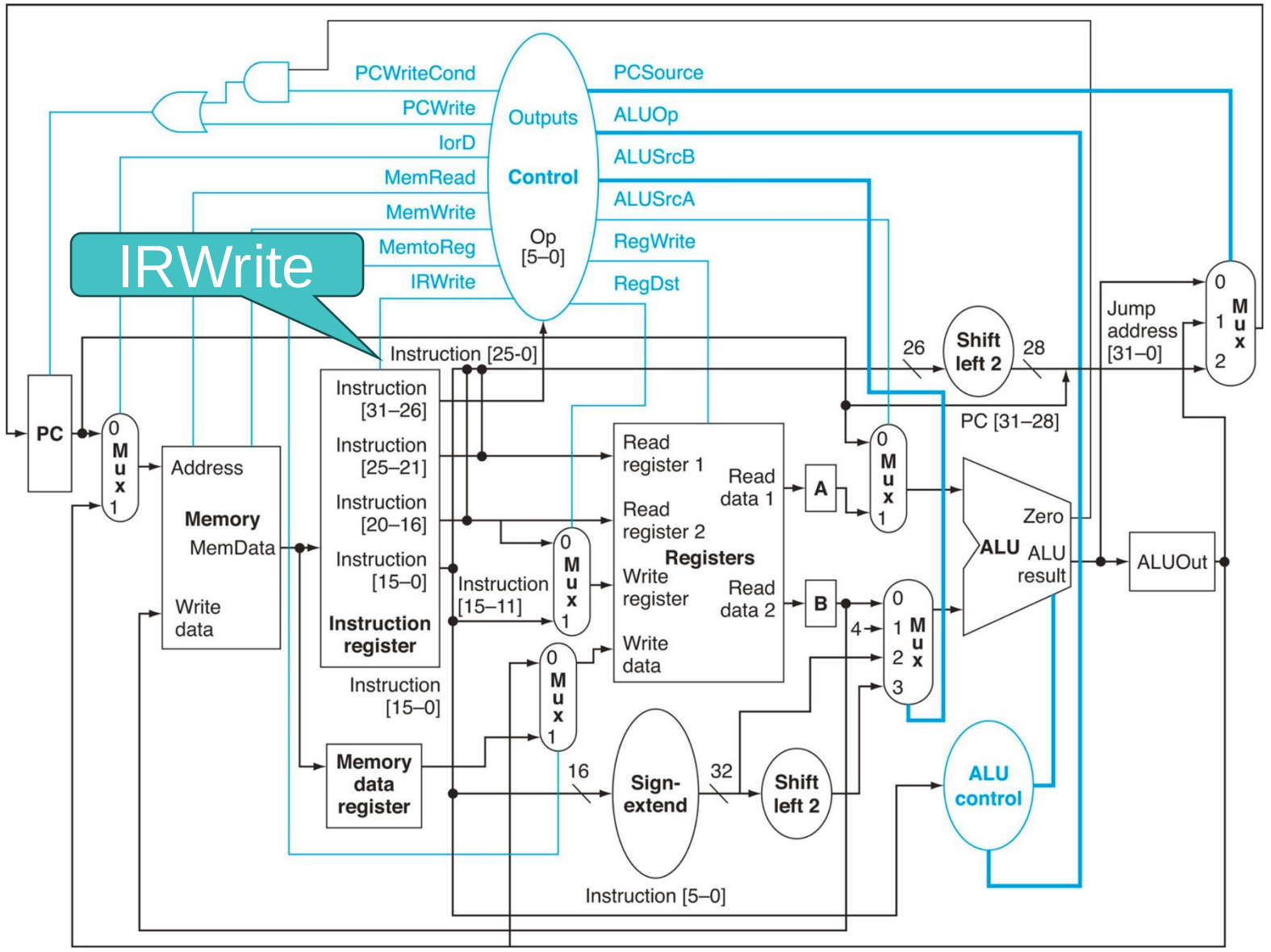


Single-cycle implementation:
we don't need a PC write signal because *it is updated on every clock cycle*

Multi-cycle implementation:
The PC should be updated *only in a specific cycle* → Introduce a PC write signal

Multicycle Implementation: Control Signals

The instruction register should be updated only in the IF stage →
Introduce a IR write signal



1-bit Control Signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by the value on the Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

2-bit Control Signals

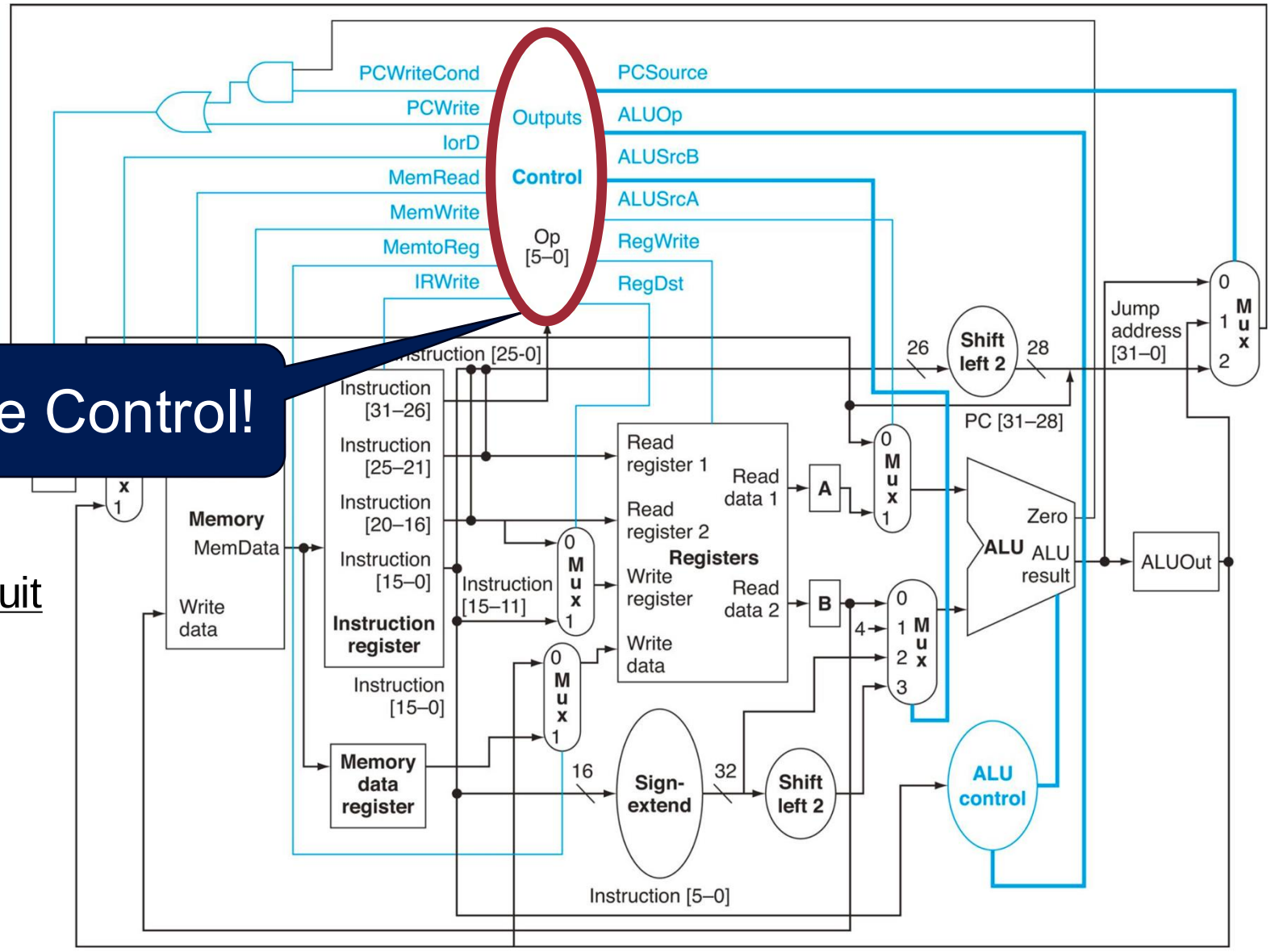
Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

Multicycle Implementation: Control

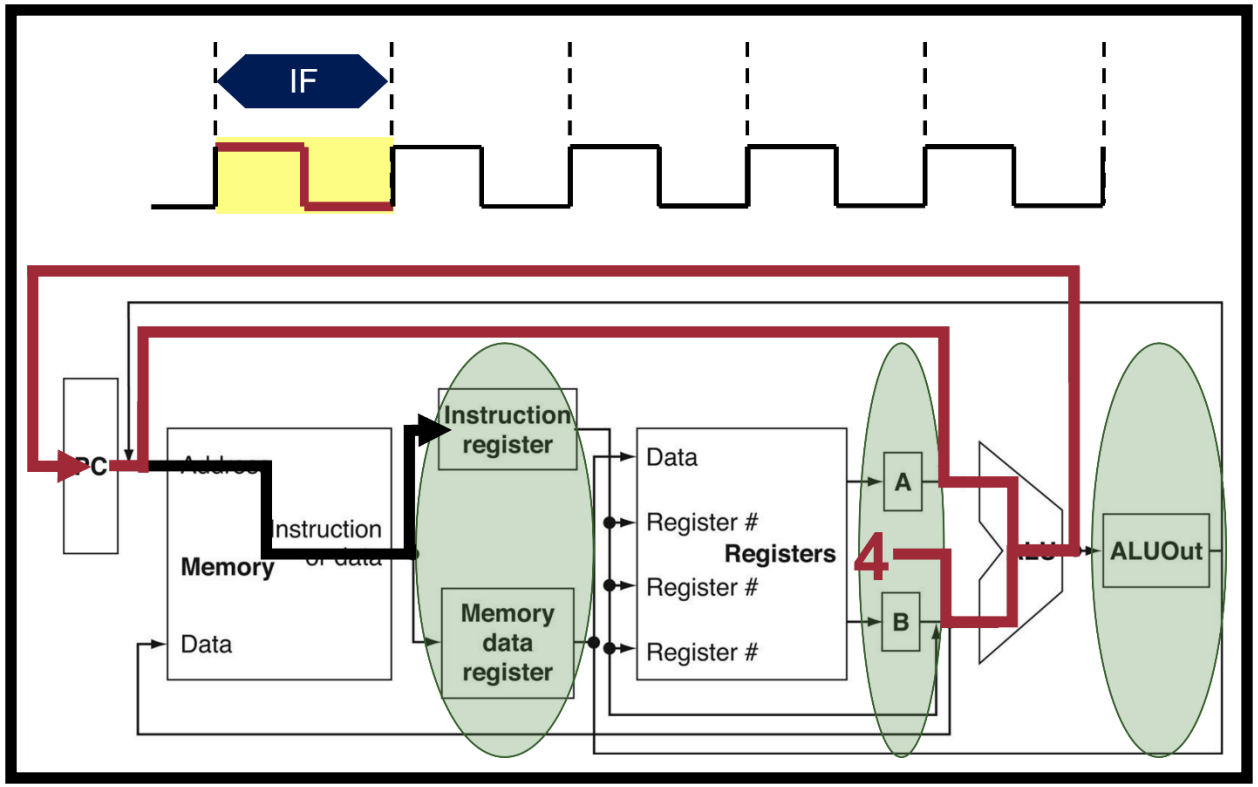
Let's look at the Control!

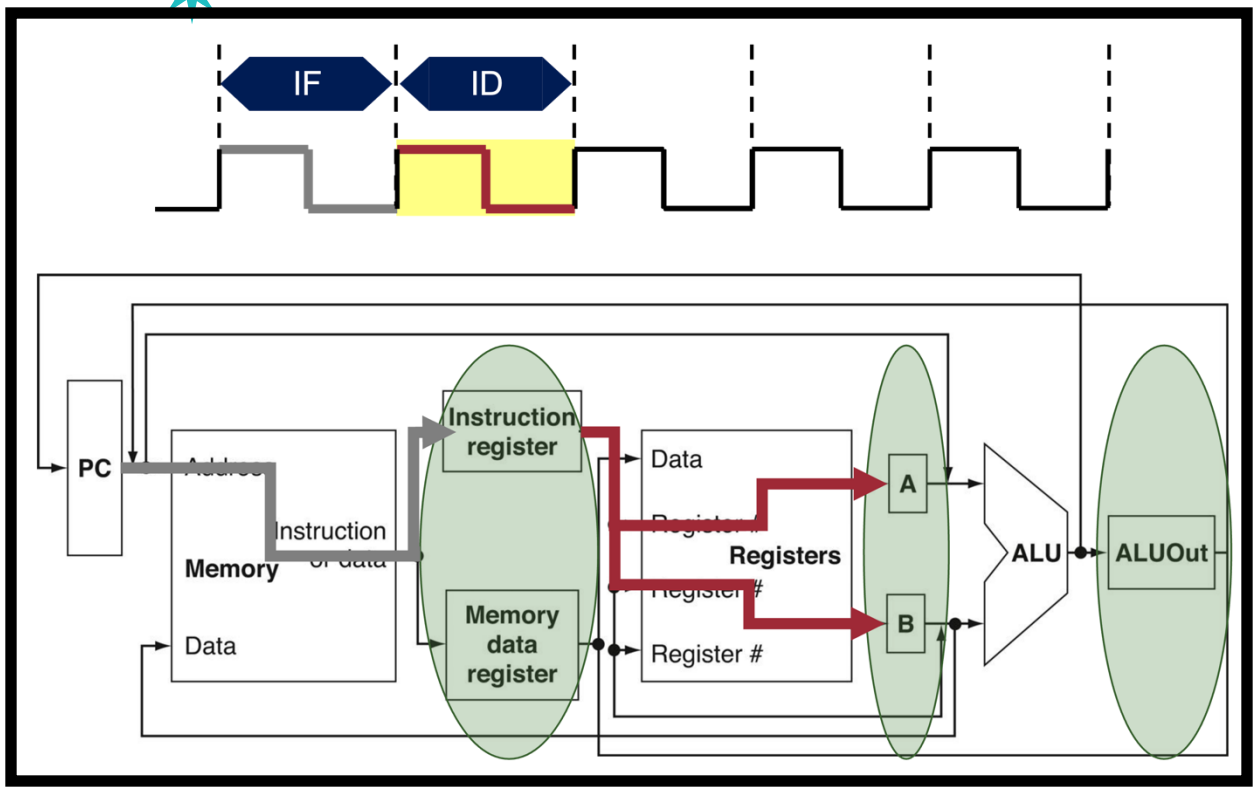
Single-cycle implementation:
The control unit is a combinational circuit

Multi-cycle implementation:
The control unit is a sequential circuit

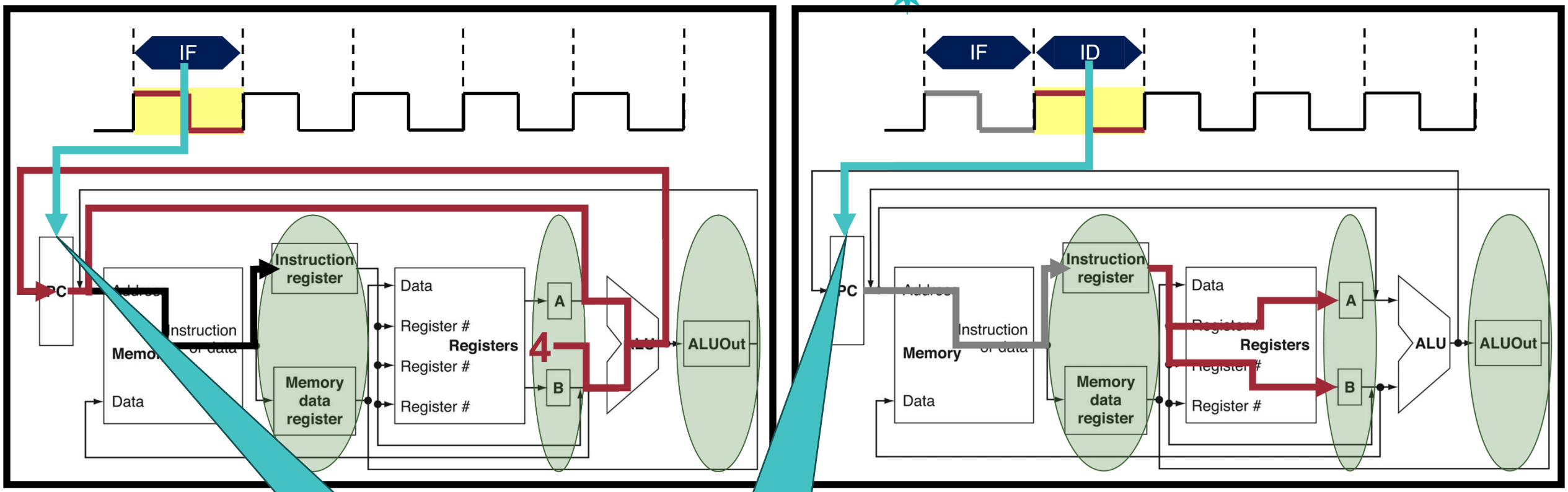


Motivation: Multicycle Control Unit





Motivation: Multicycle Control Unit



PCWrite=1

PCWrite=0

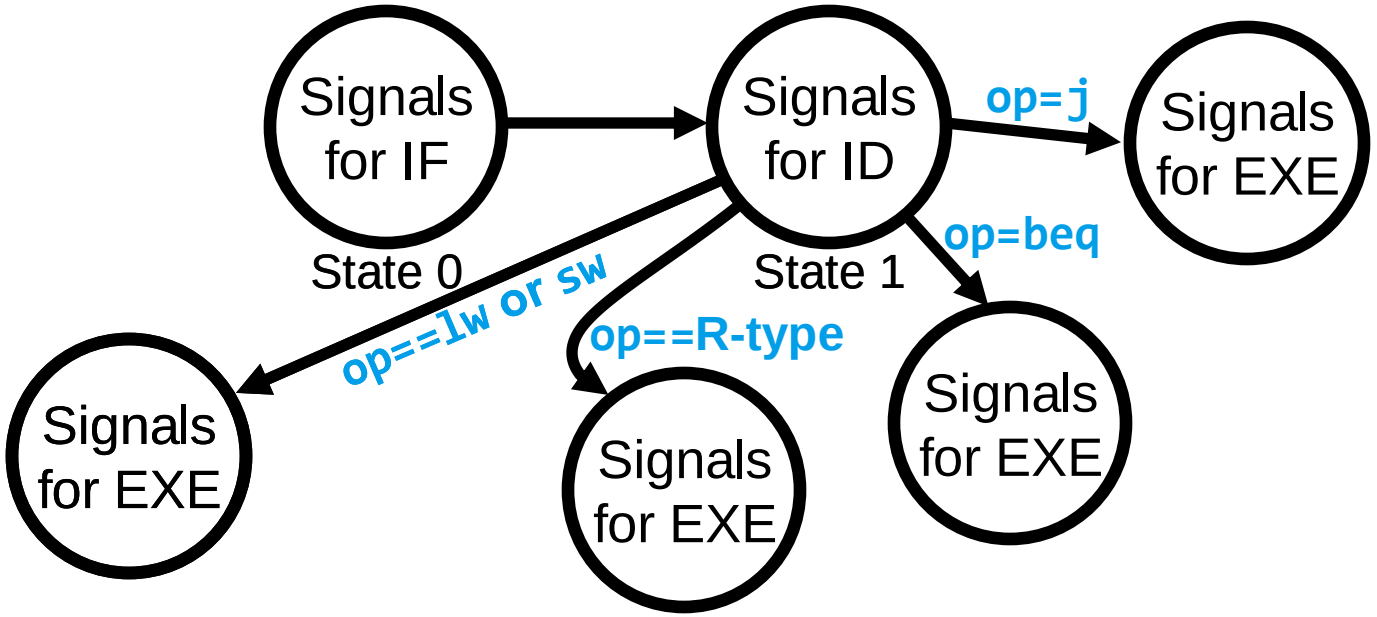
Even for the same instruction, the required control signals vary depending on the cycle

→

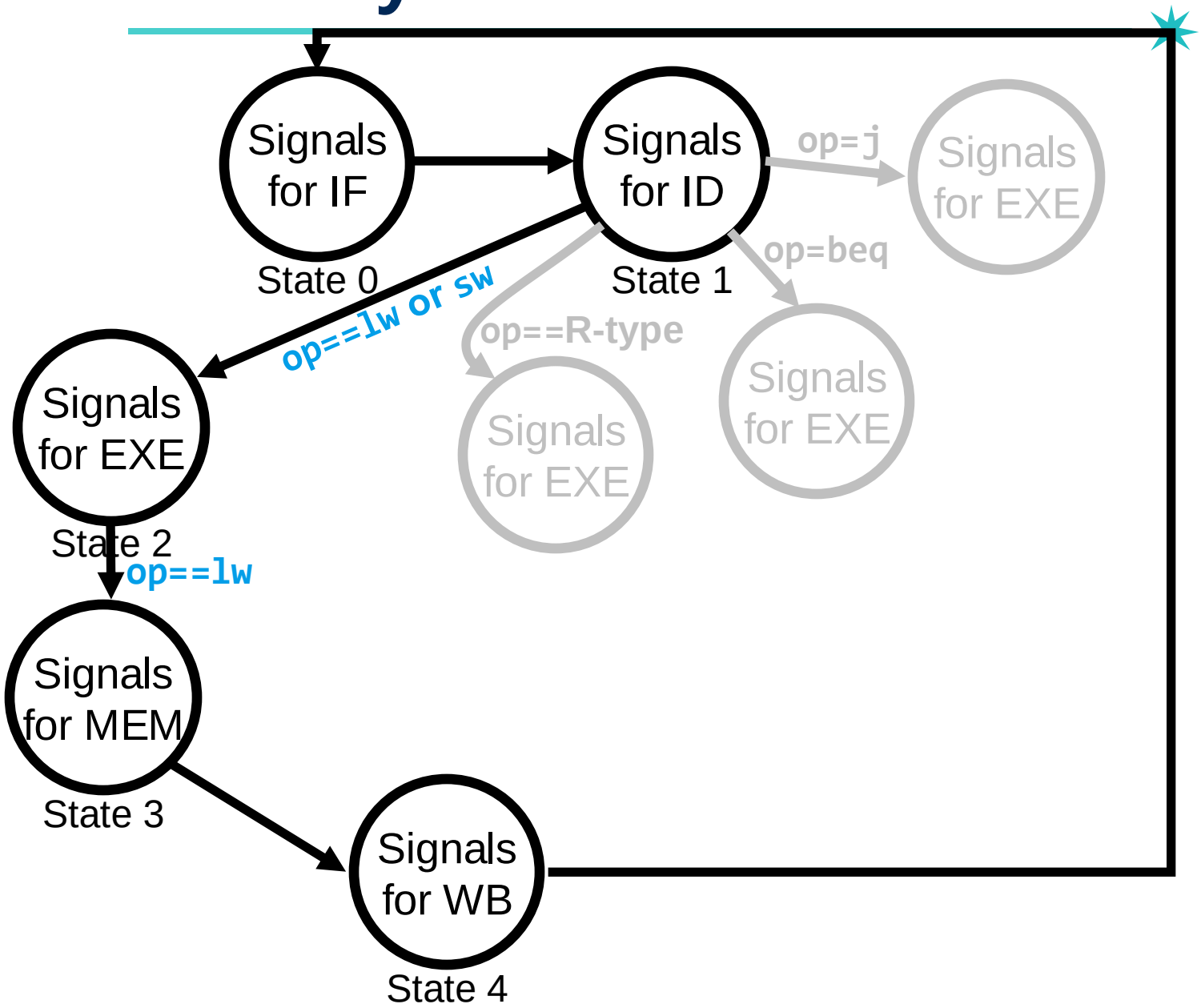
Idea: maintain states (signals) for each cycle

Multicycle Control Unit: Use Finite State Machine

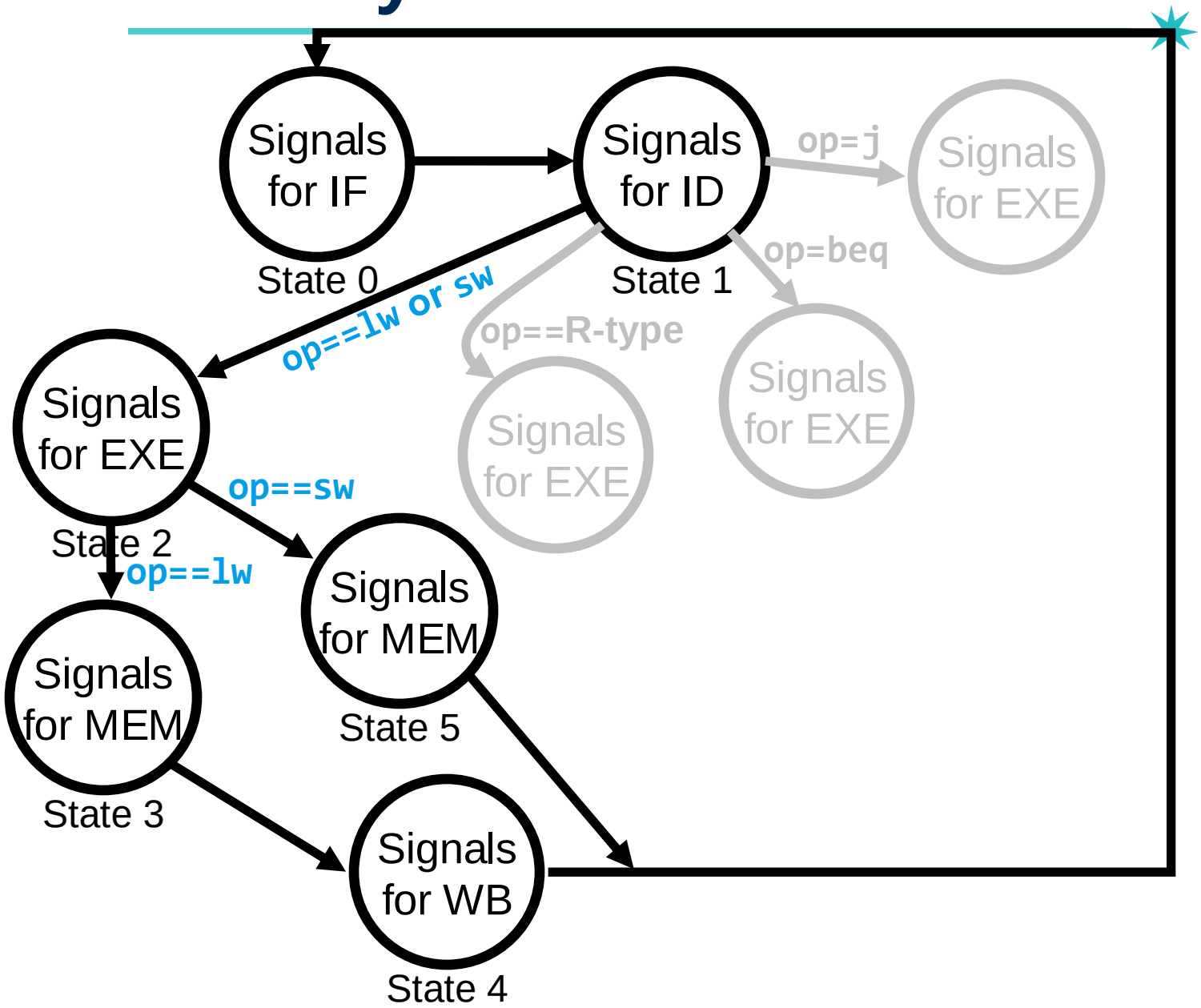
Multicycle Control Unit: Use Finite State Machine



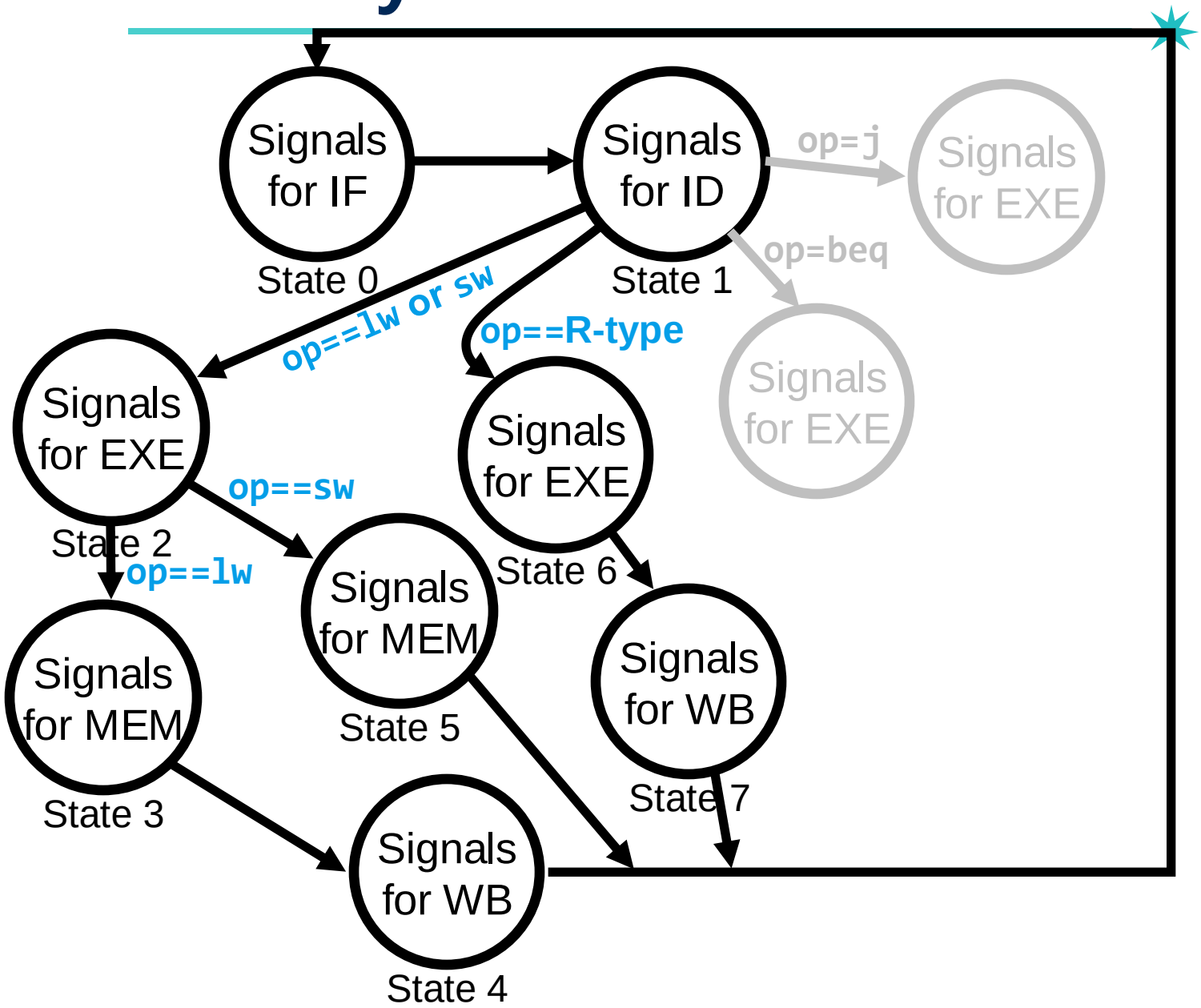
Multicycle Control Unit: Use Finite State Machine



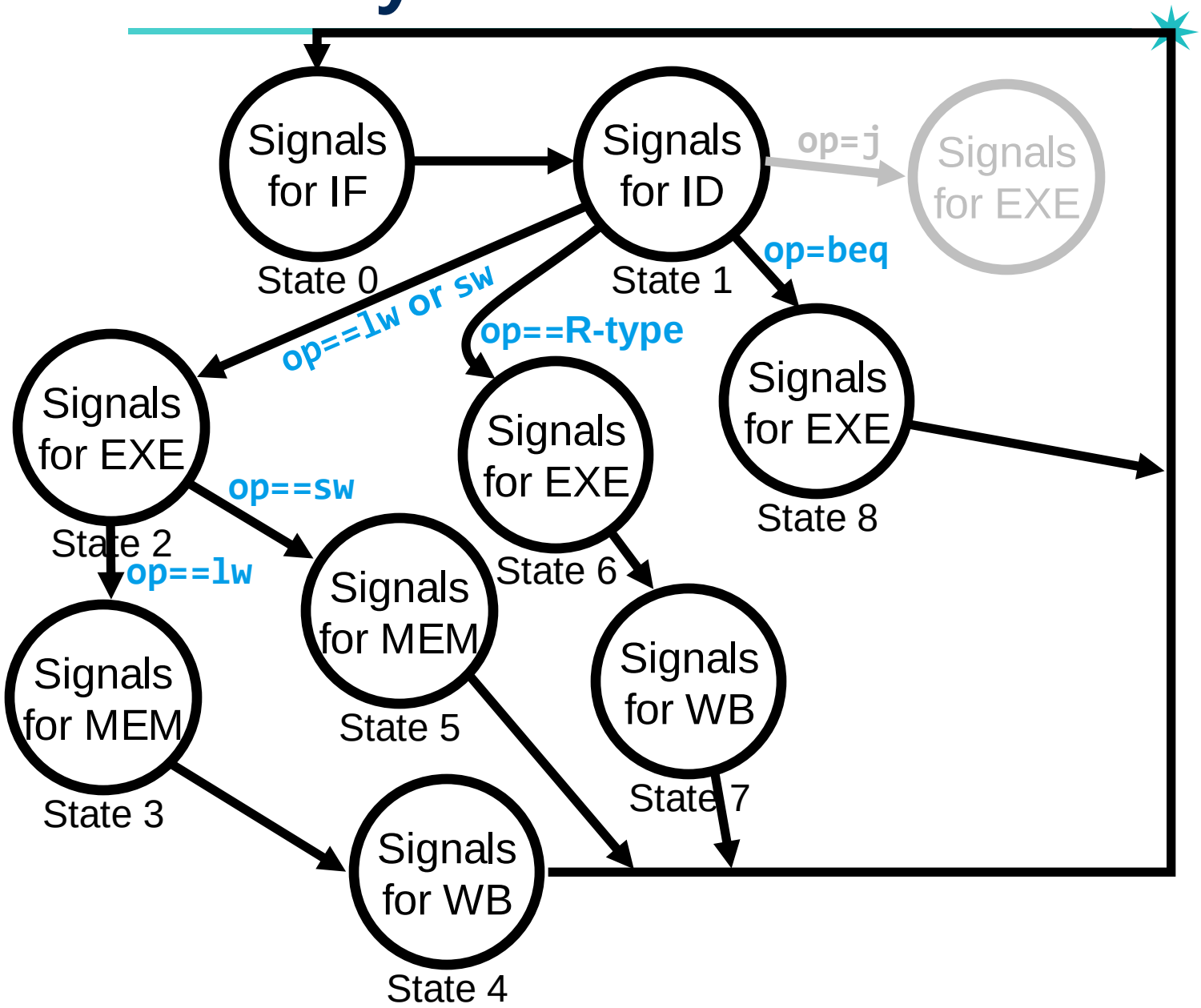
Multicycle Control Unit: Use Finite State Machine



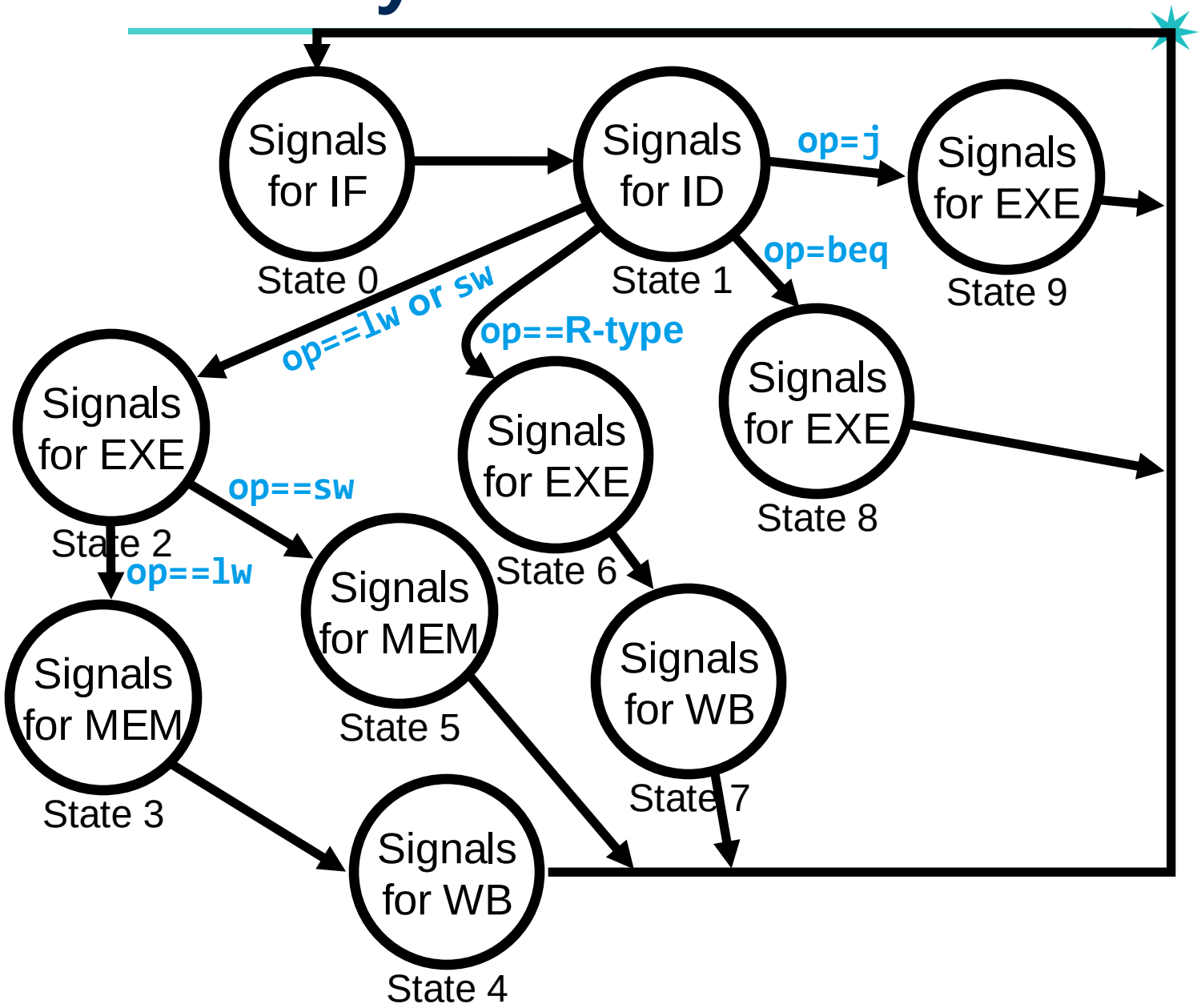
Multicycle Control Unit: Use Finite State Machine



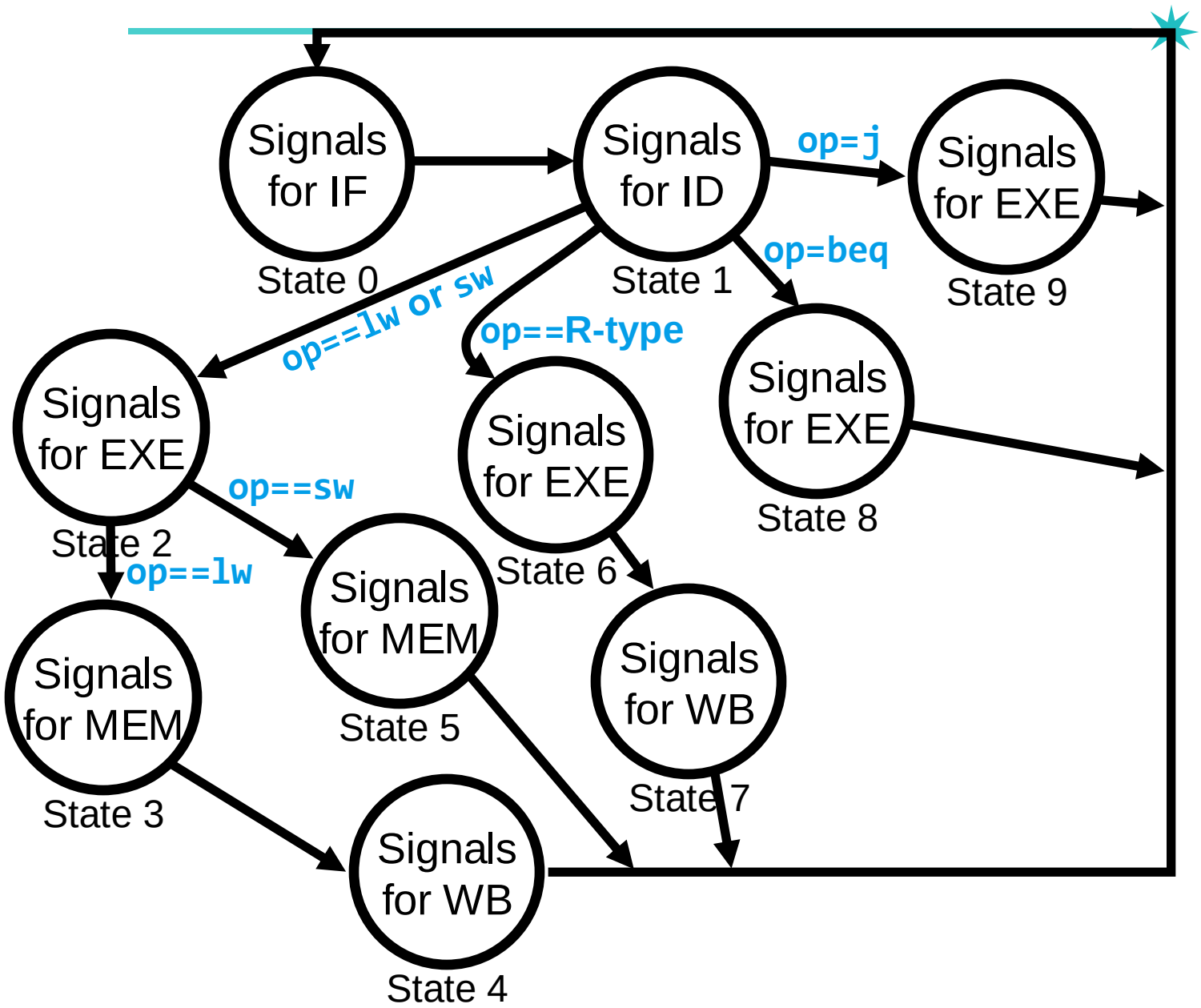
Multicycle Control Unit: Use Finite State Machine

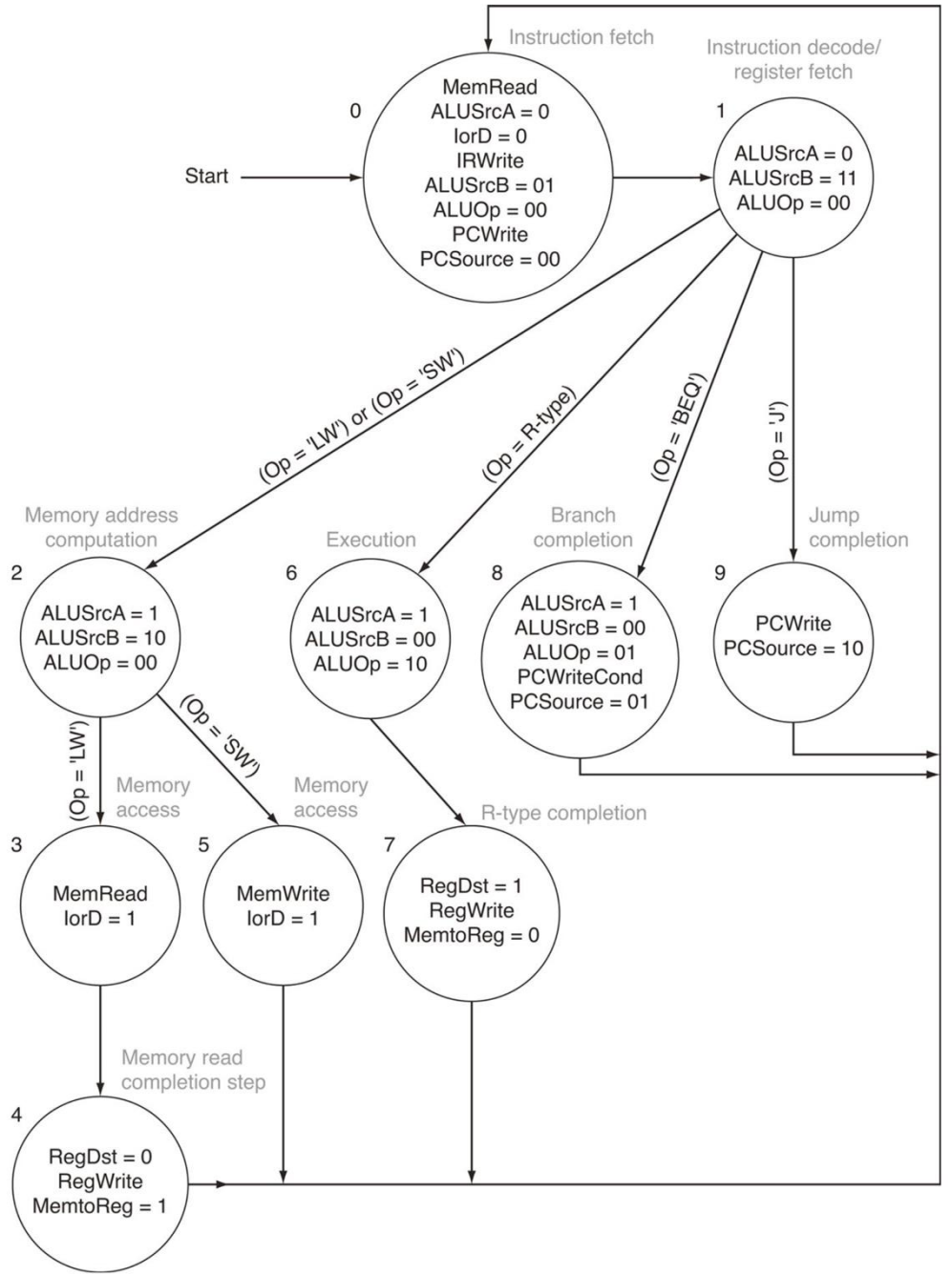


Multicycle Control Unit: Use Finite State Machine

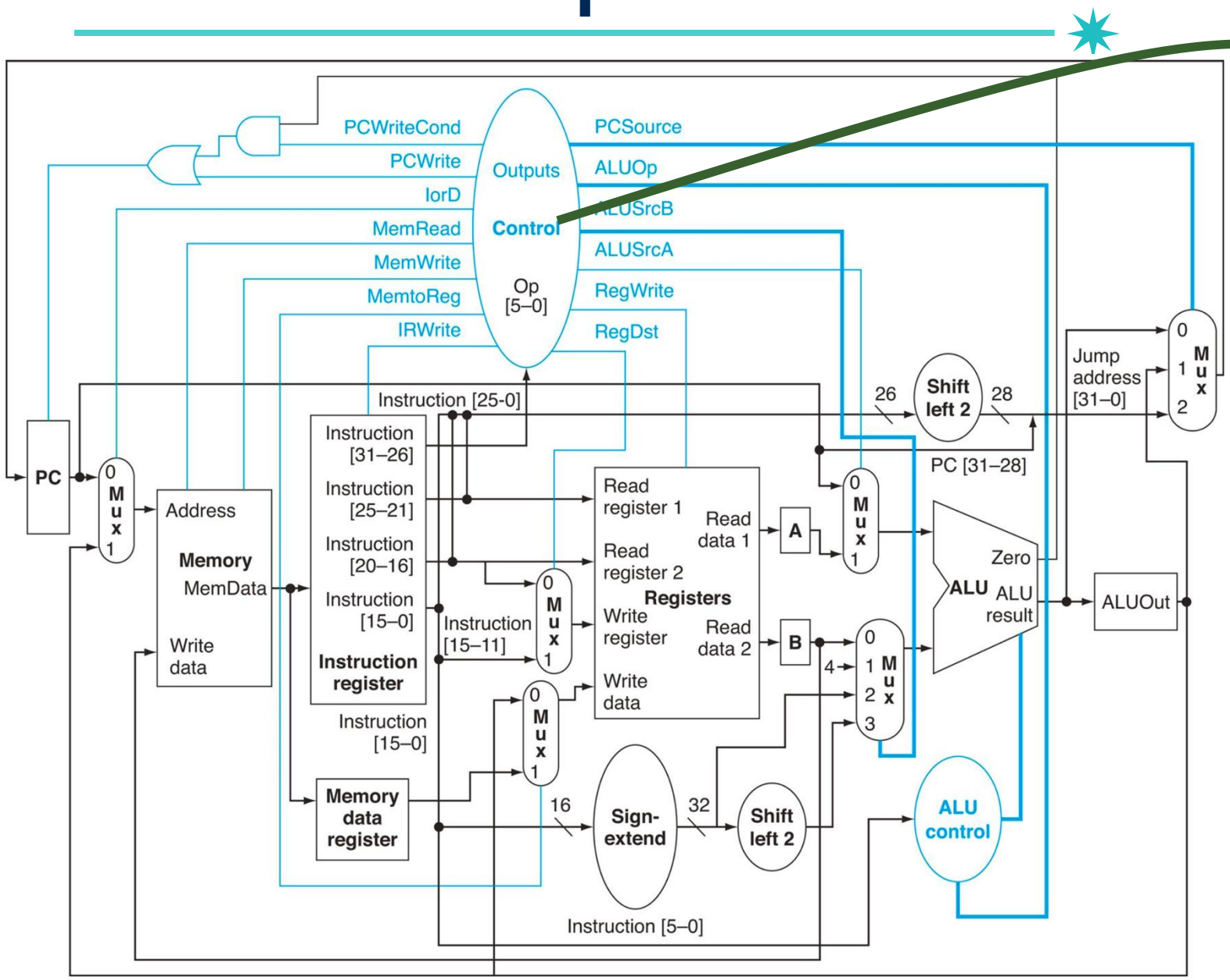


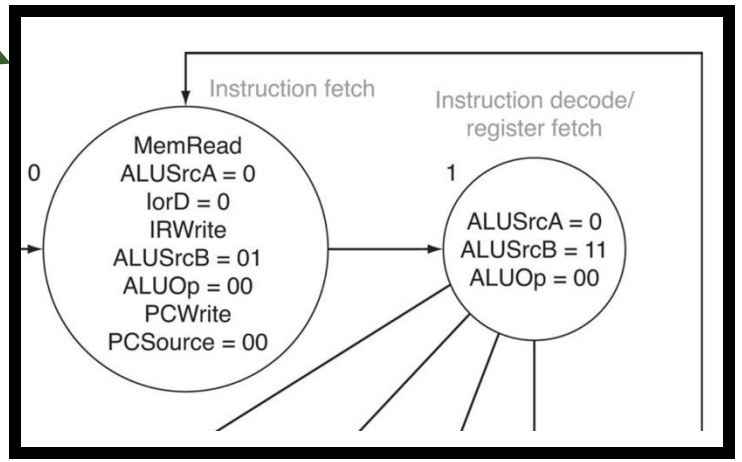
Control Unit Details



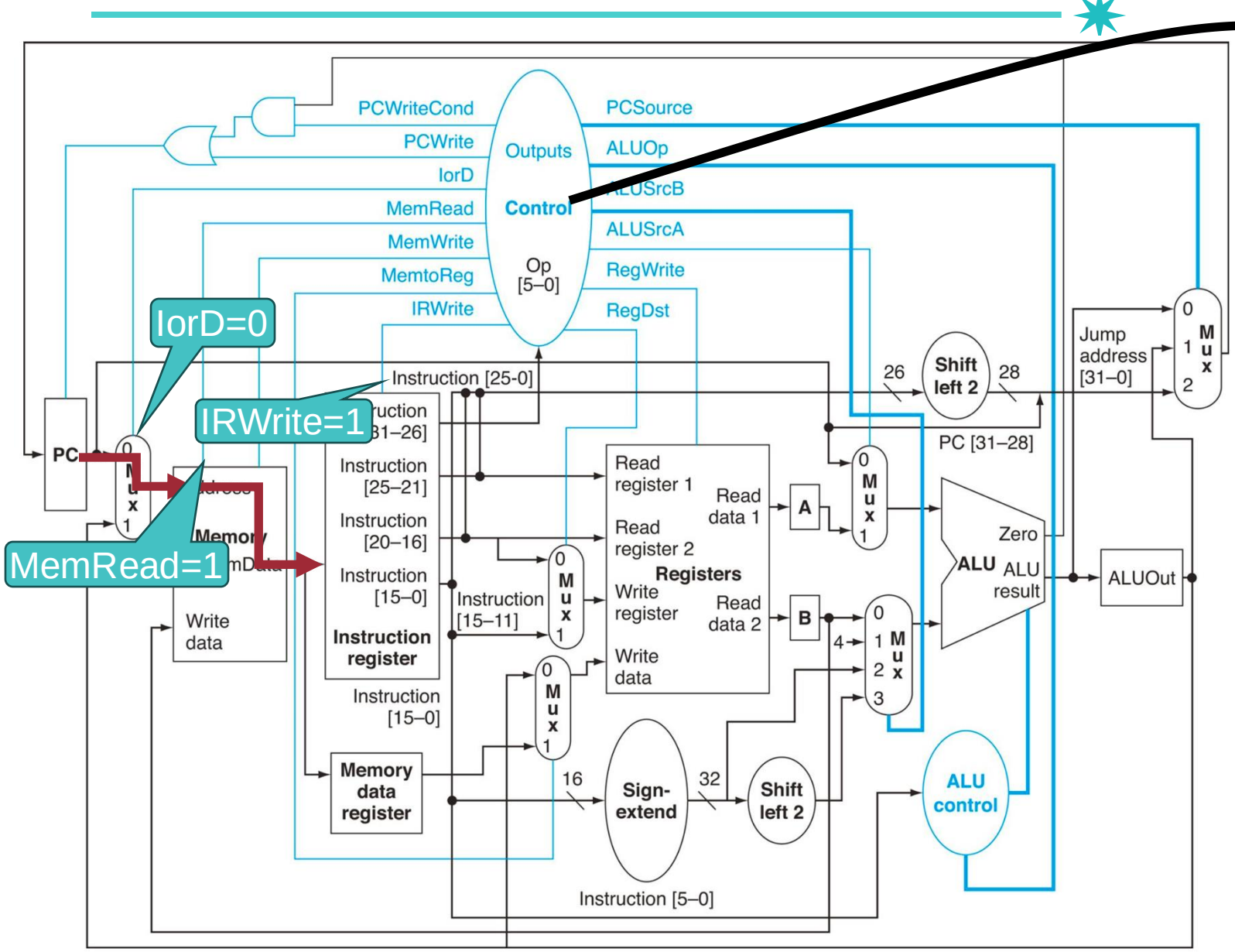


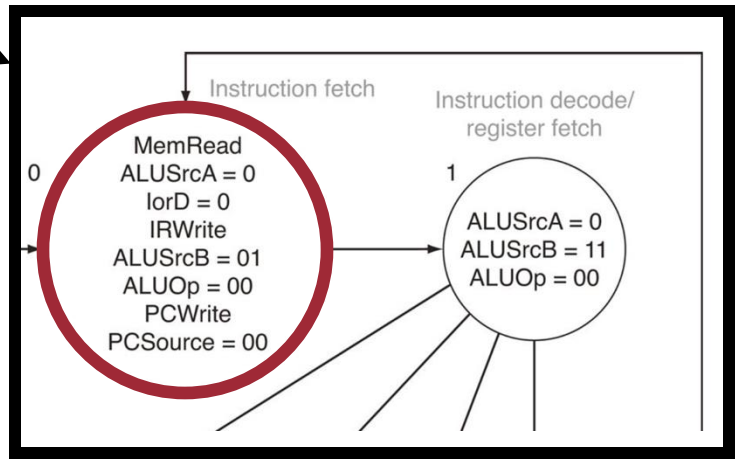
Control Example



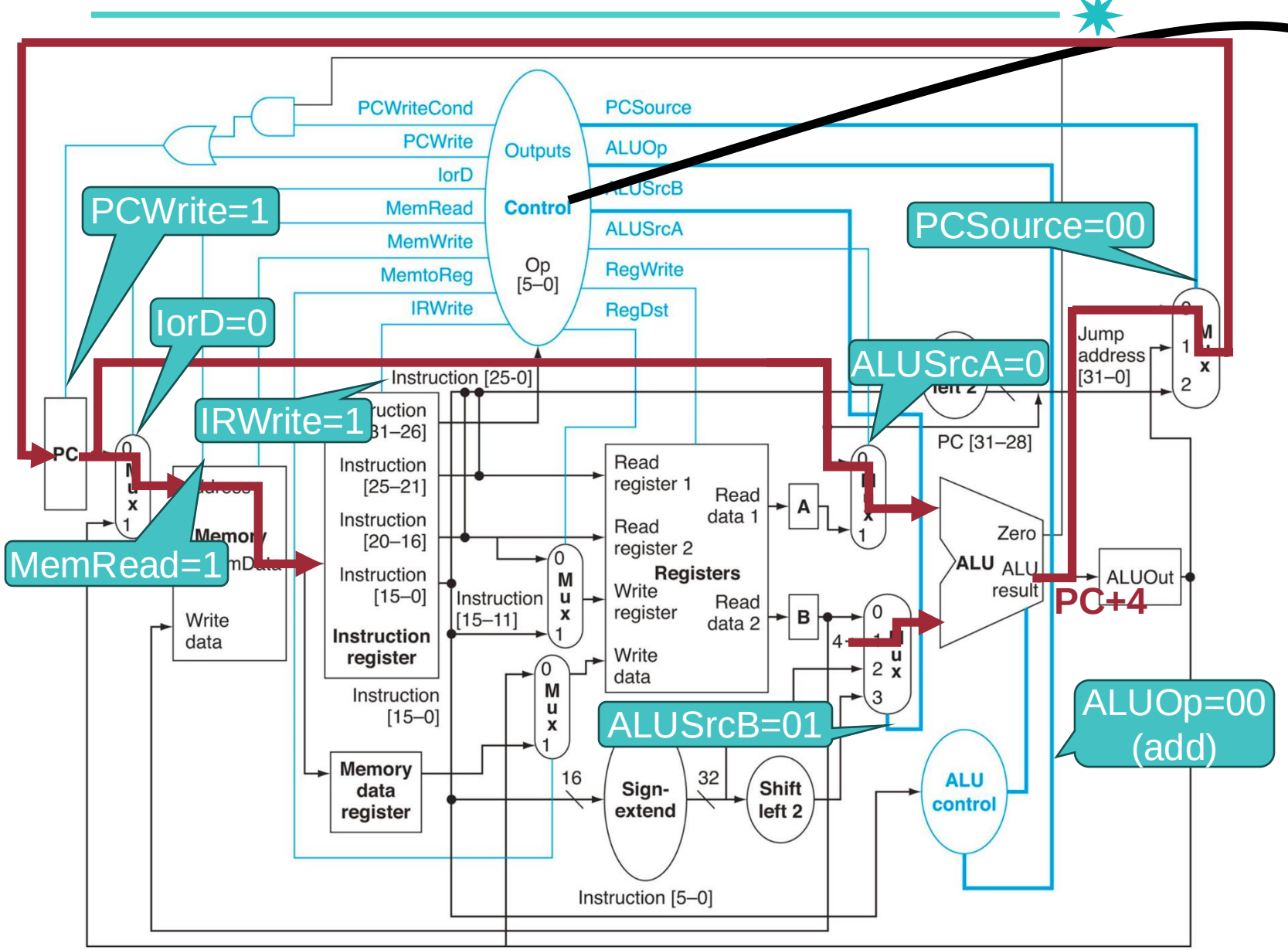


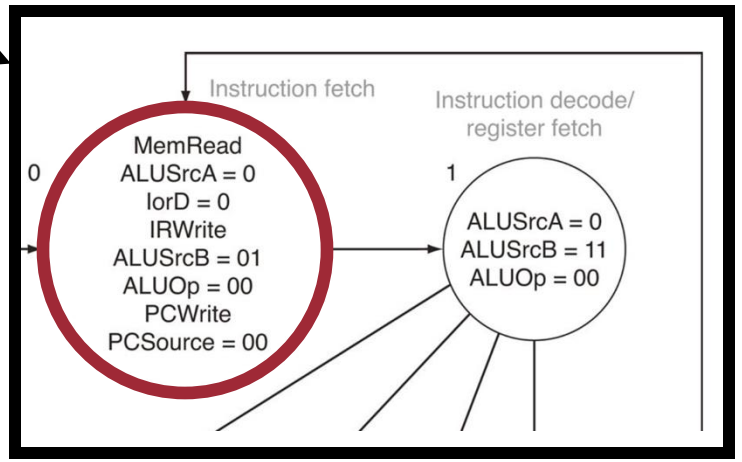
Example: 1st Cycle (Instruction Fetch)



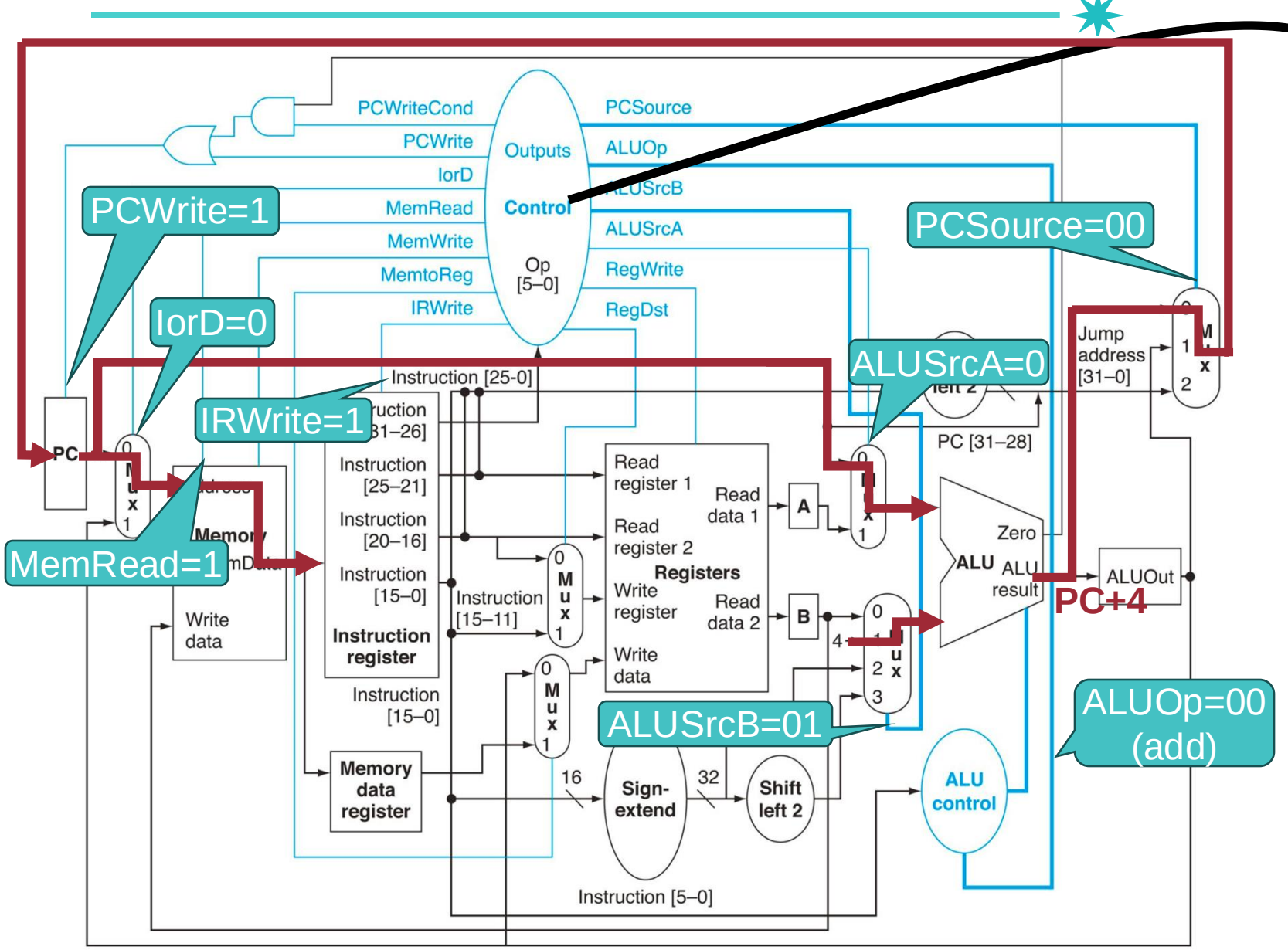


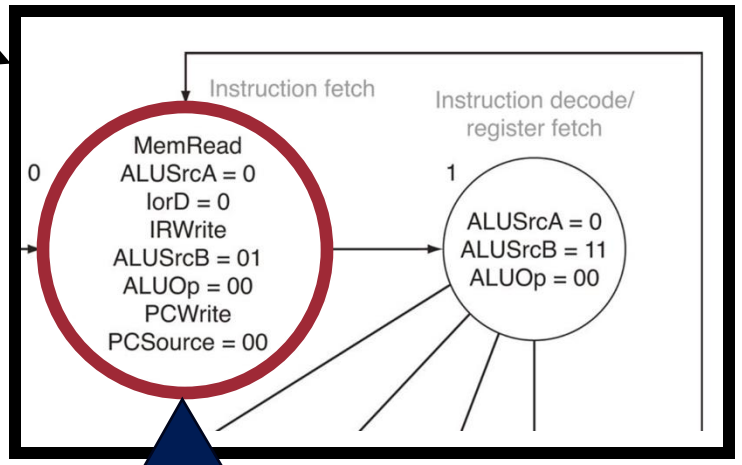
Example: 1st Cycle (Instruction Fetch)





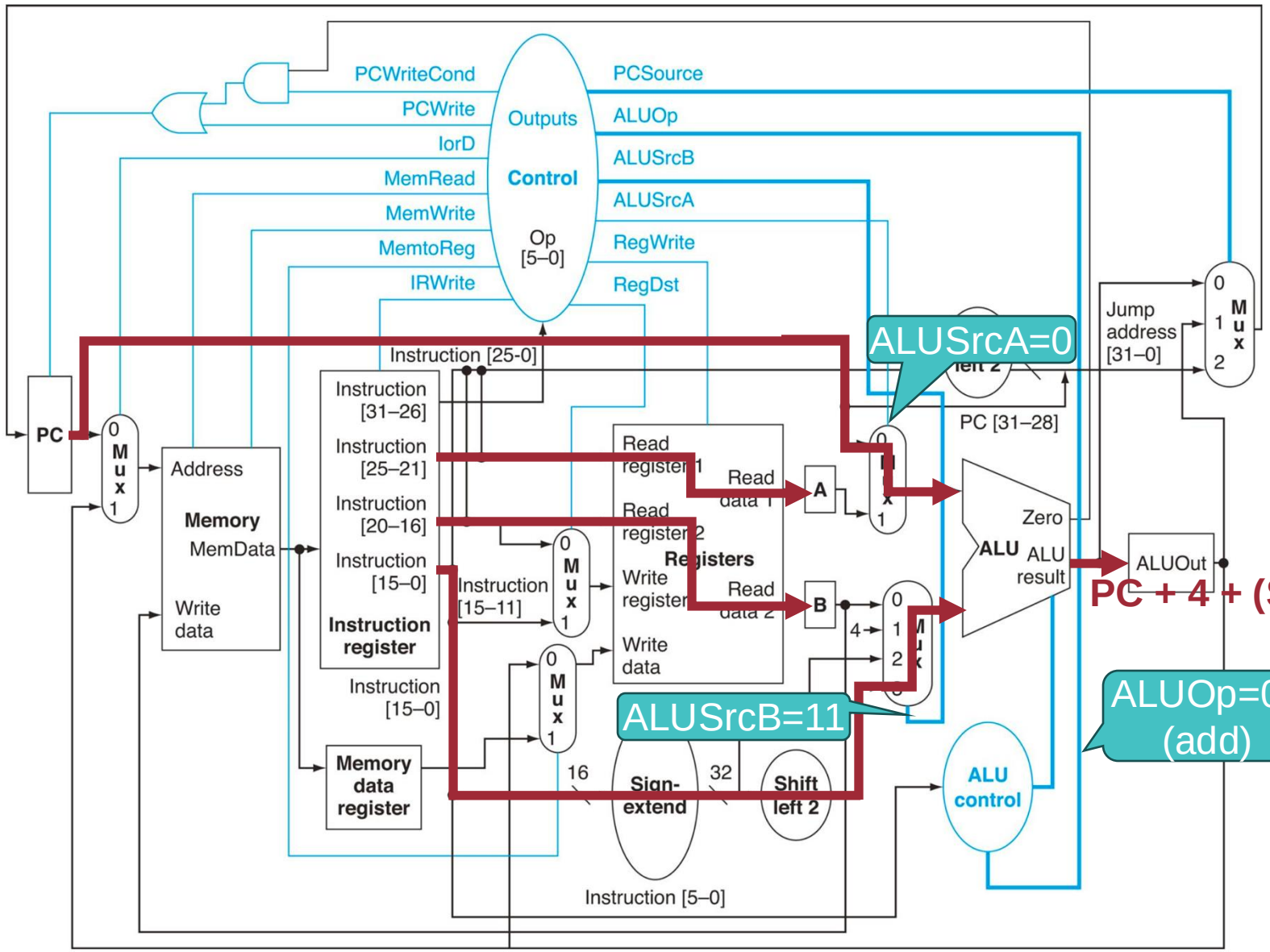
Example: 1st Cycle (Instruction Fetch)

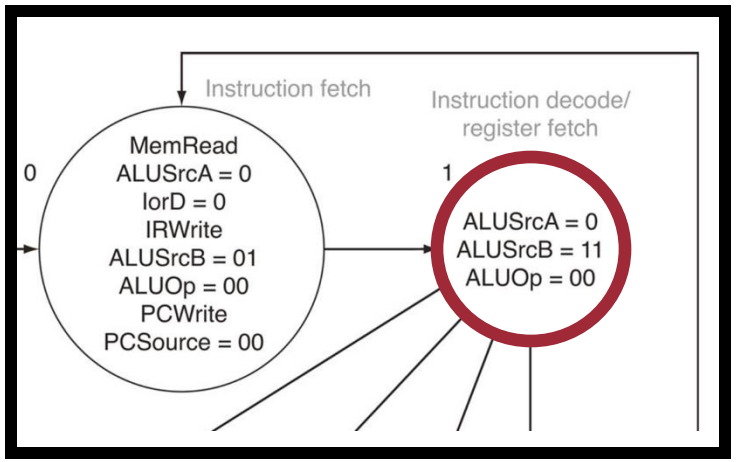




Signals that are not specified as 0 (deasserted)

Example: 2nd Cycle (Instruction Decode)





Multicycle Implementation: Advantages

Compared with single-cycle implementation, performance can be ***increased*** due to:

- Clock cycle period ↓
- Different instructions take different numbers of cycles
- Hardware sharing: single ALU and single memory
  - Reduce the circuit area
  - They can be used for **different purposes** *in different clock cycle*

Multicycle Implementation: Disadvantages

Compared with single-cycle implementation, performance can be ***decreased*** due to:

- Additional internal registers
- More multiplexors to shared functional units
- More complicated control to consider about *time*

Question?