CSE261: Computer Architecture

10. Processor (2)

Seongil Wi

Department of Computer Science and Engineering

Where Are We?

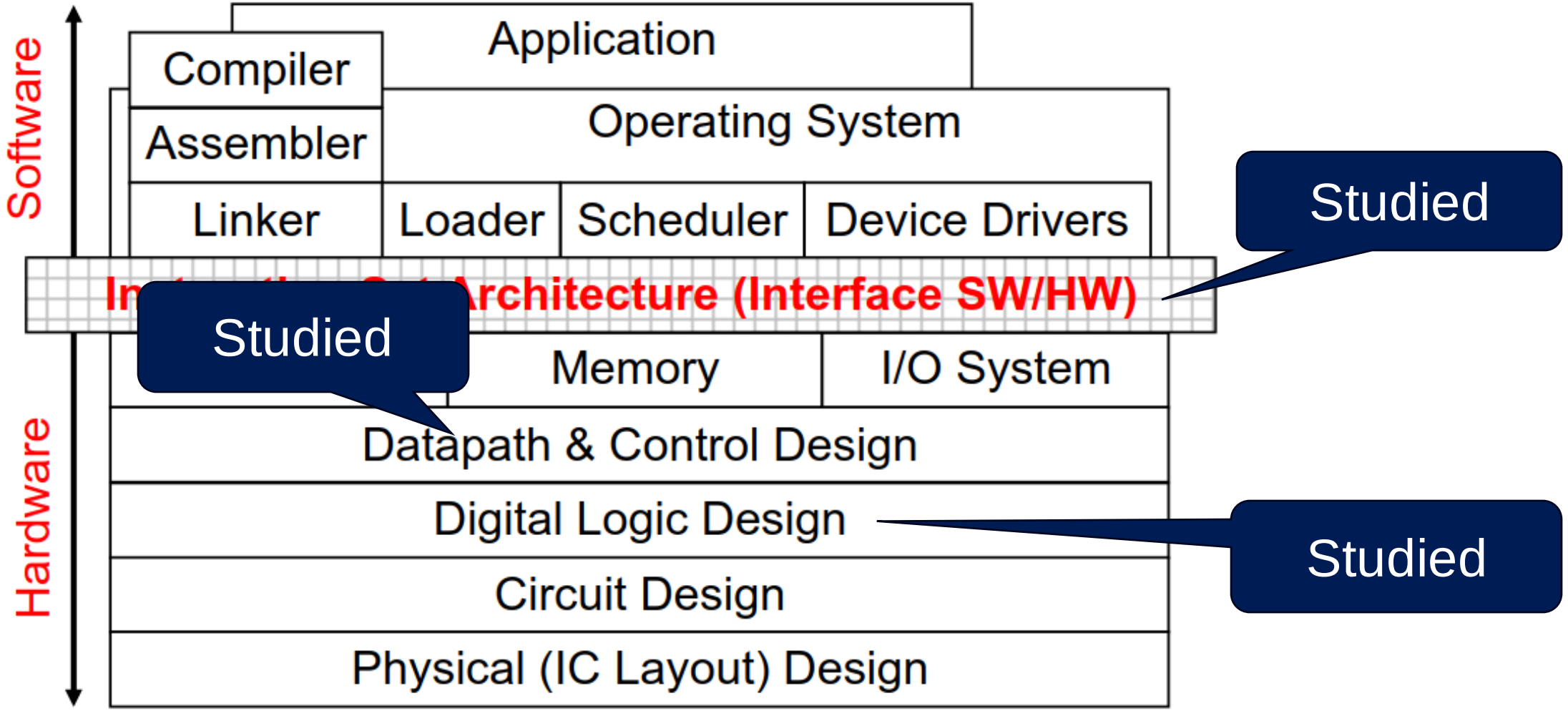


Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

Today's Topic

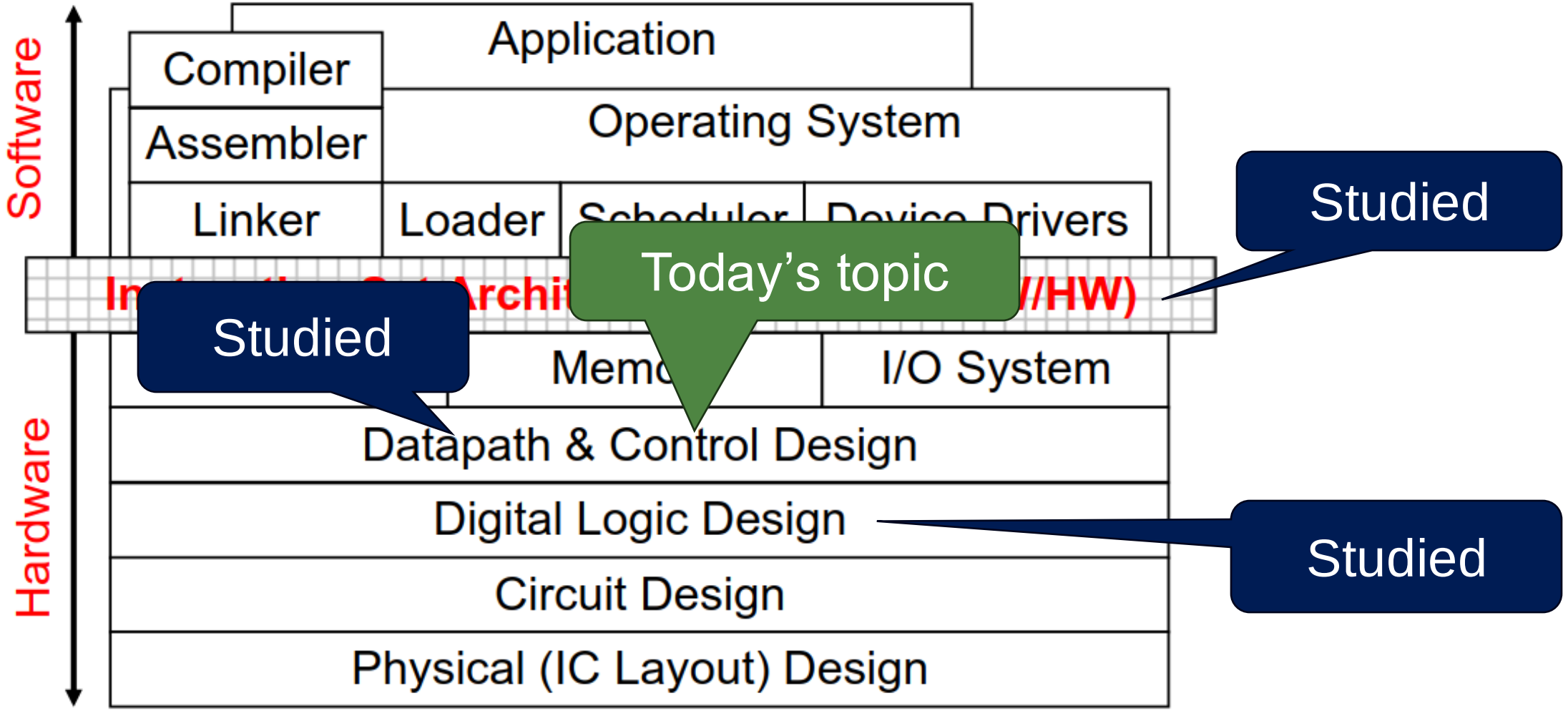
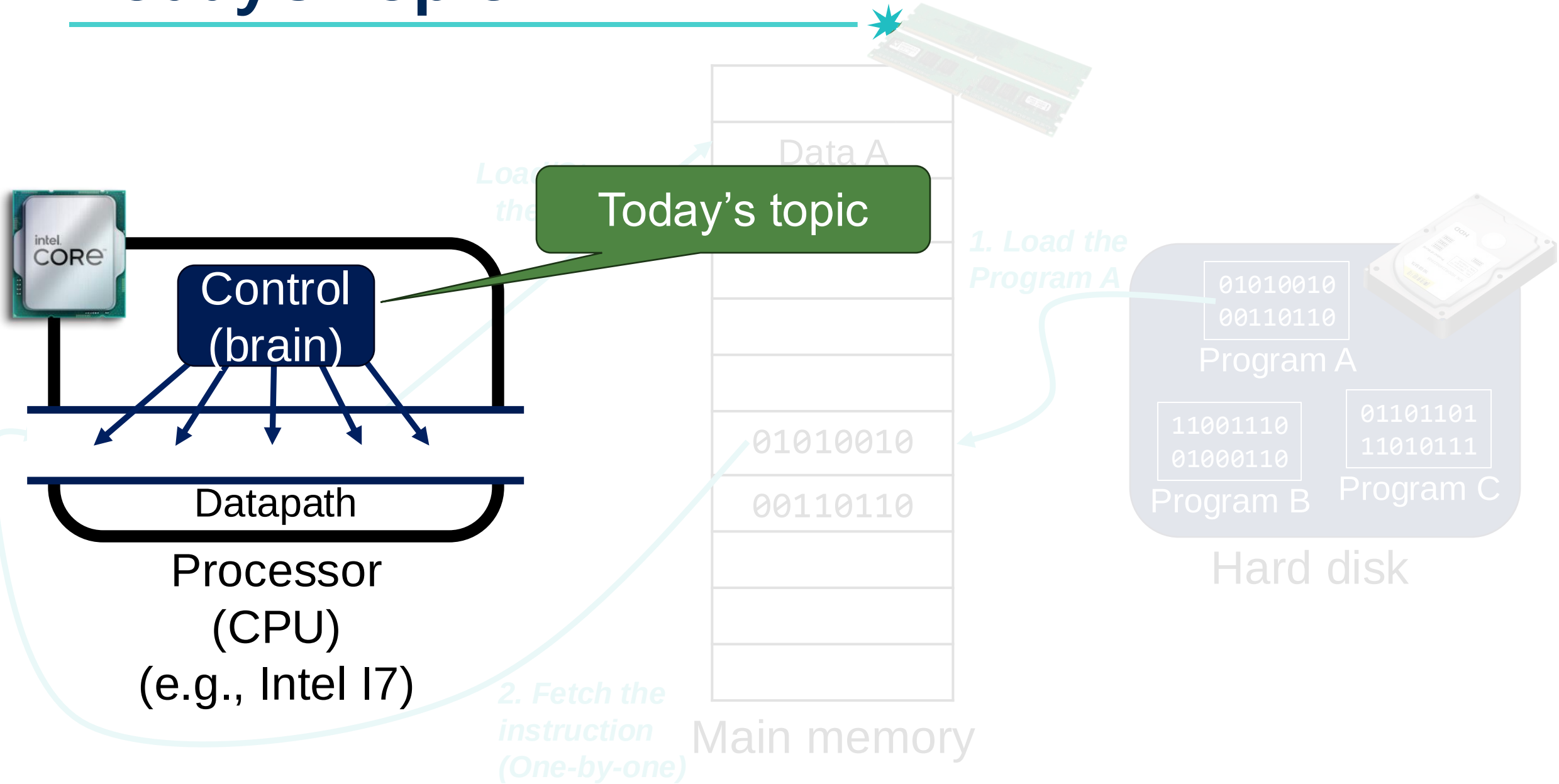


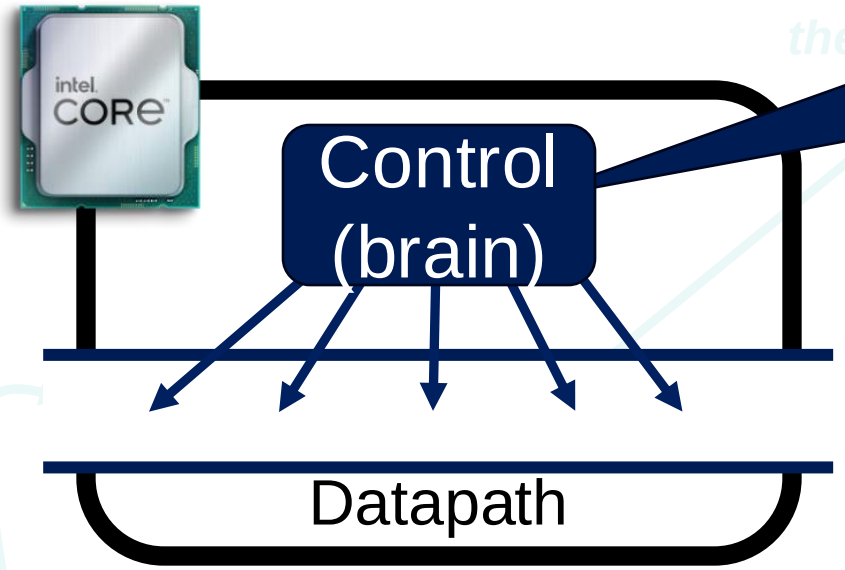
Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

Today's Topic



Building a Control Unit

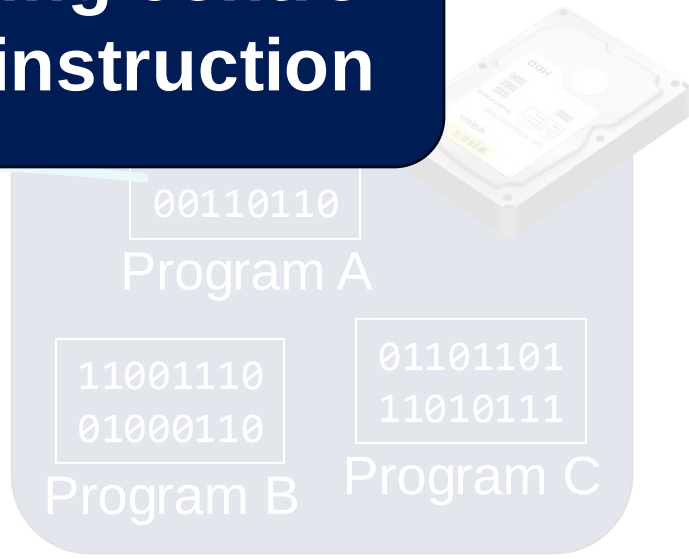
Introduction to Control



Processor
(CPU)
(e.g., Intel I7)

Tell what to do by **sending control signals derived from instruction**

Main memory



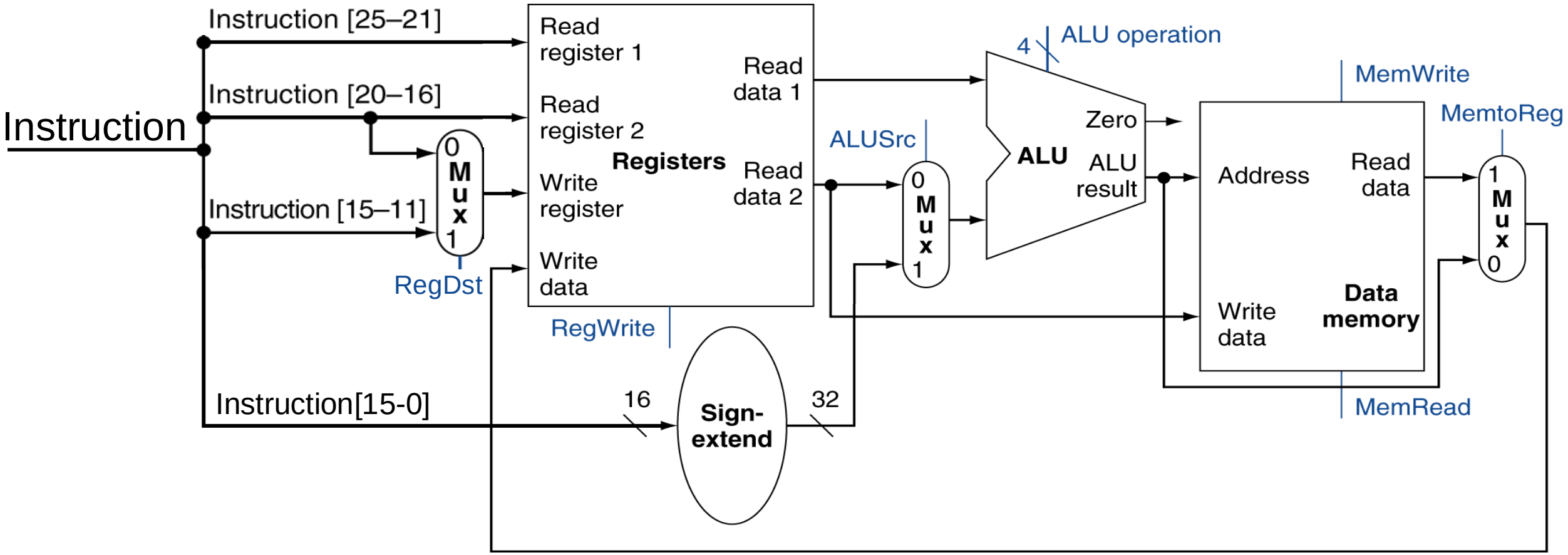
Hard disk

Introduction to Control

add instruction

R-type:

op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

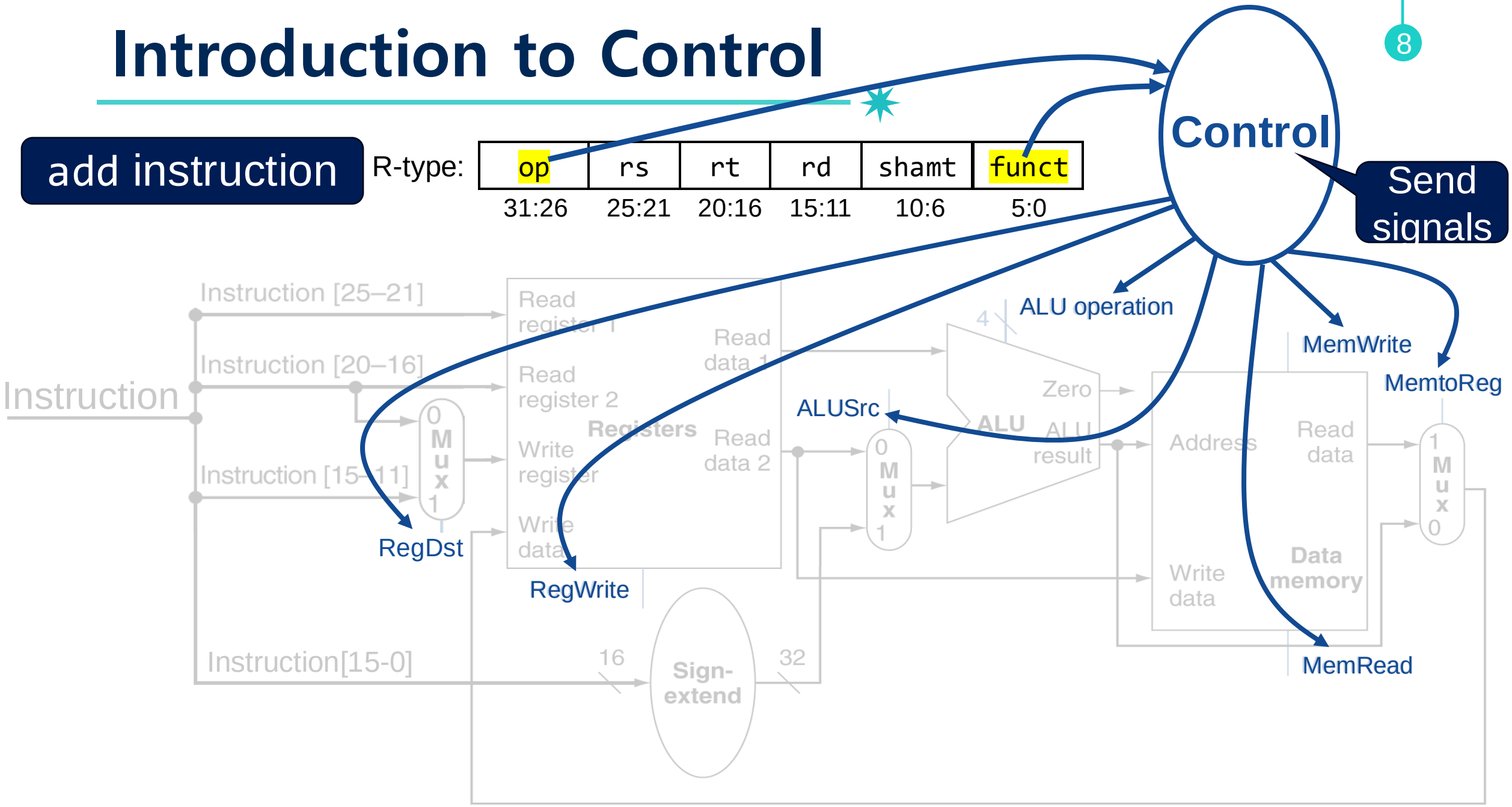


Introduction to Control

add instruction

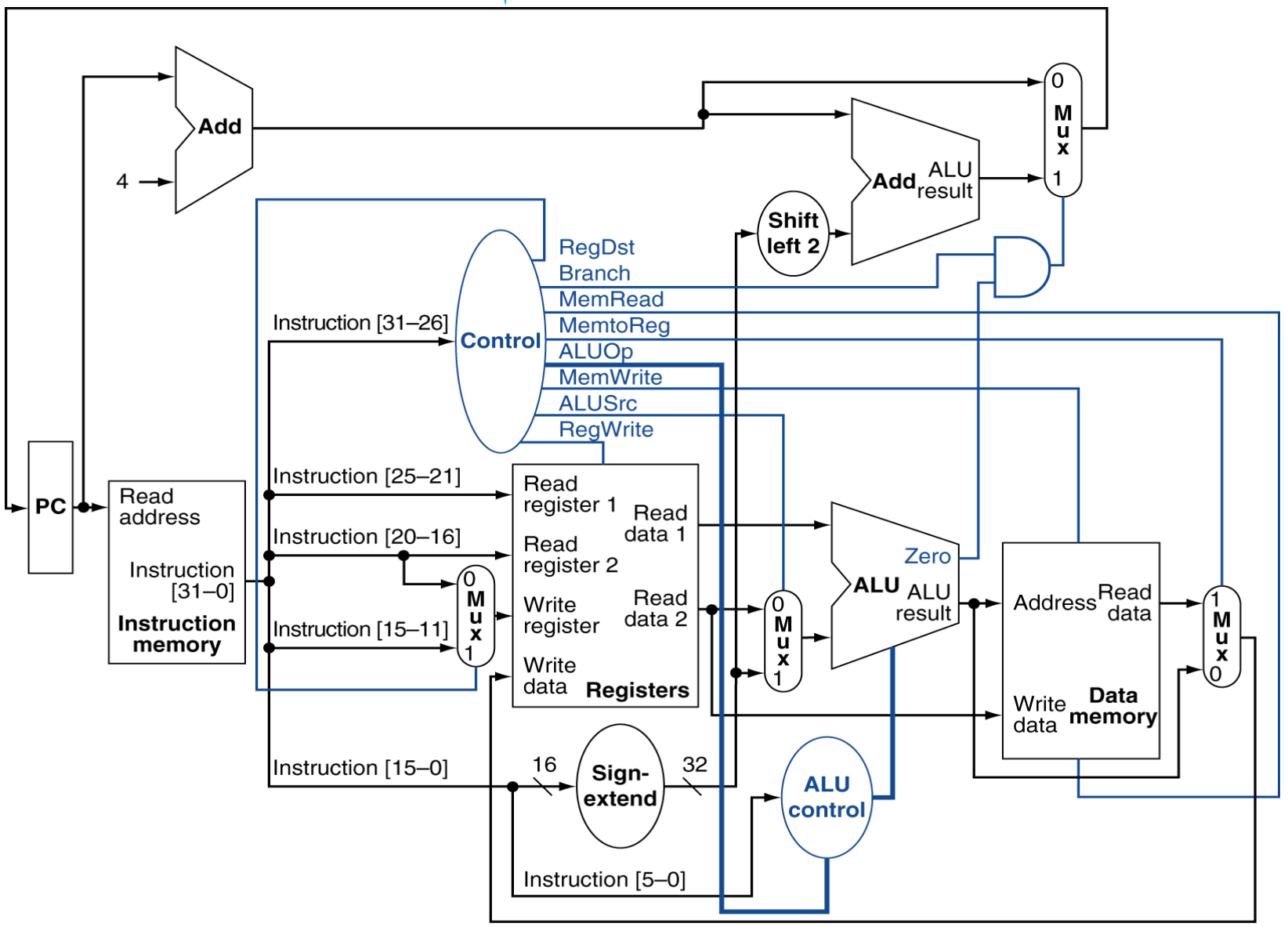
R-type:

op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

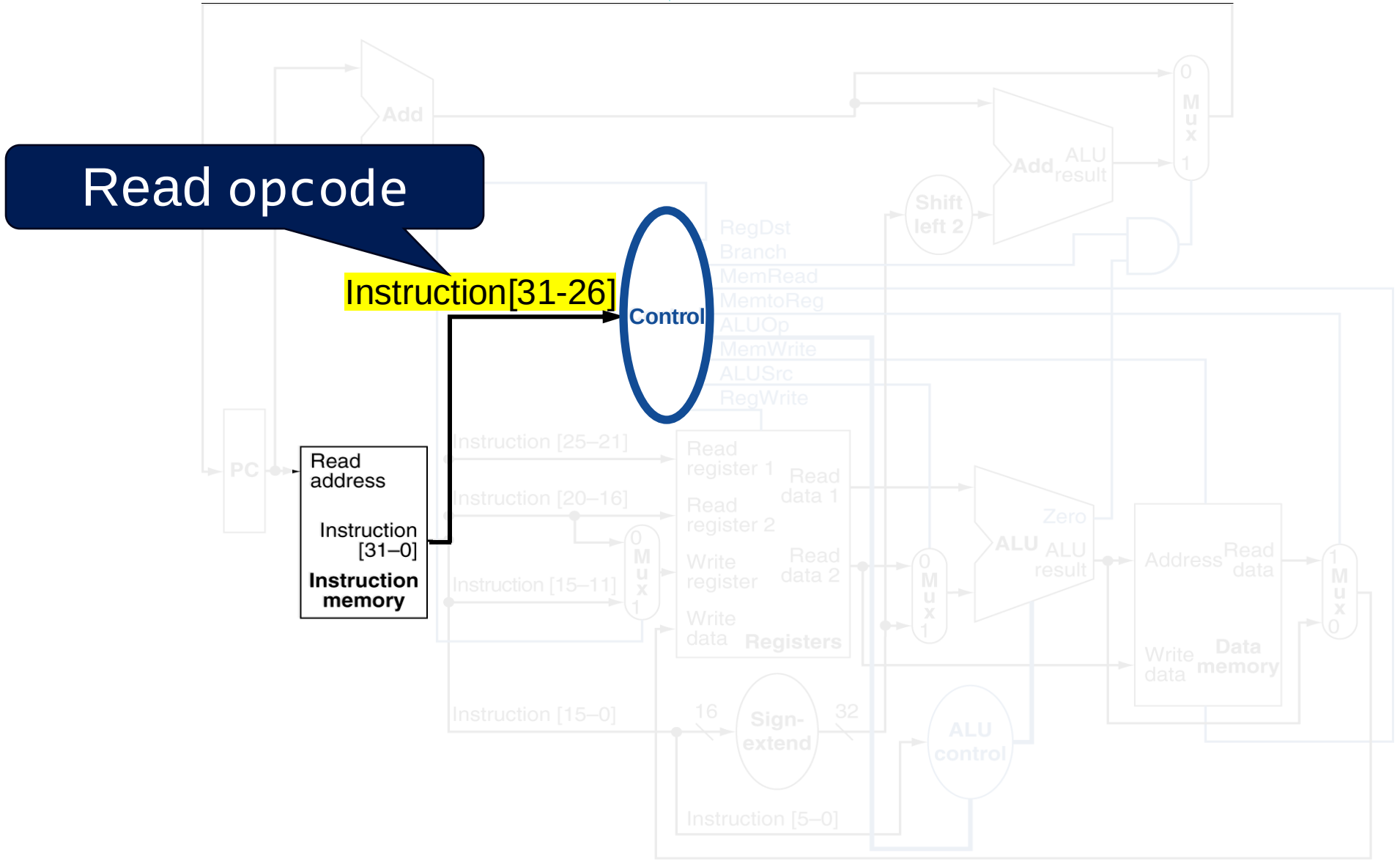


Let’s add the `Control’ to complete the implementation

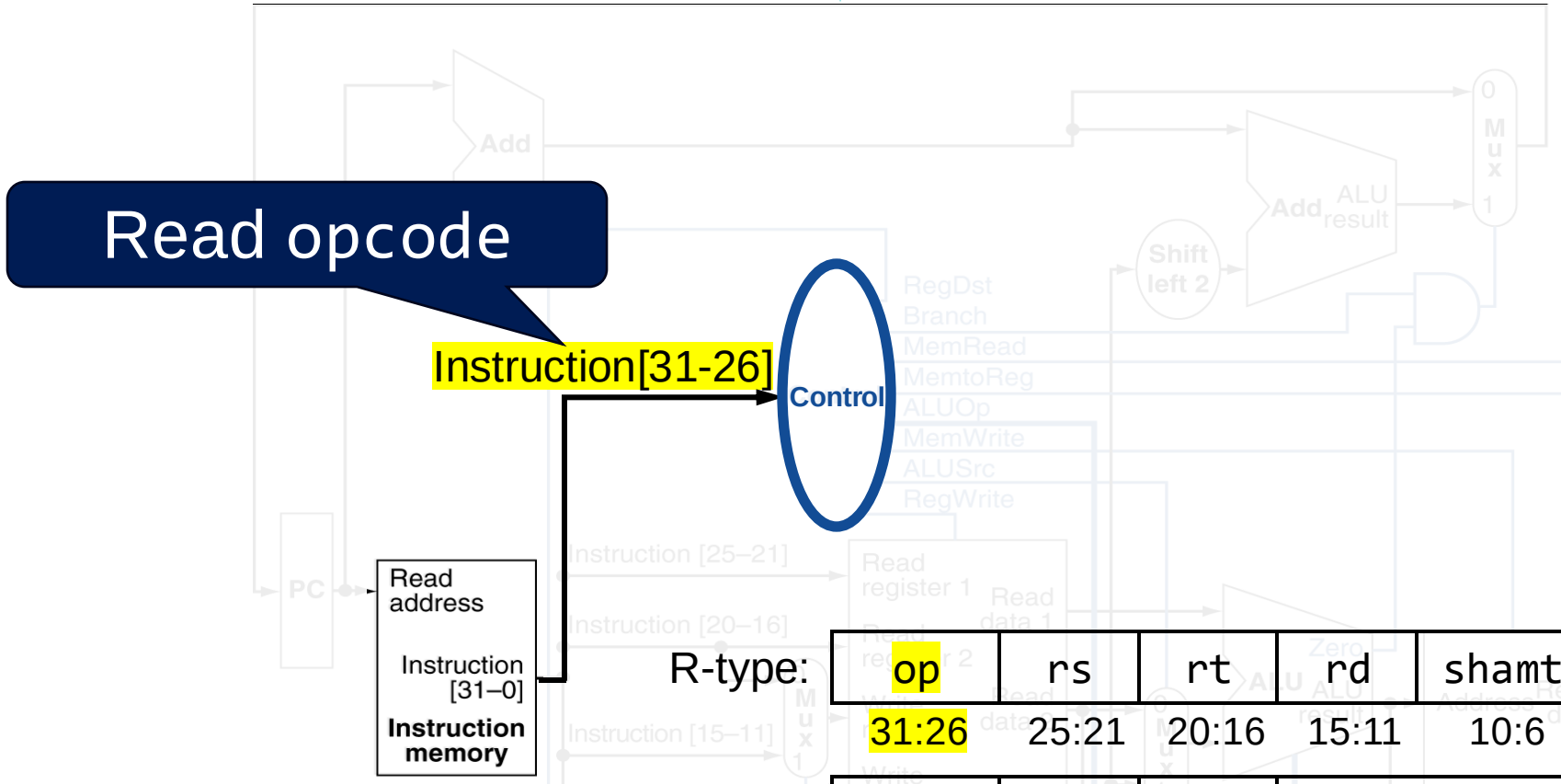
Datapath with Control



Datapath with Control: Read opcode

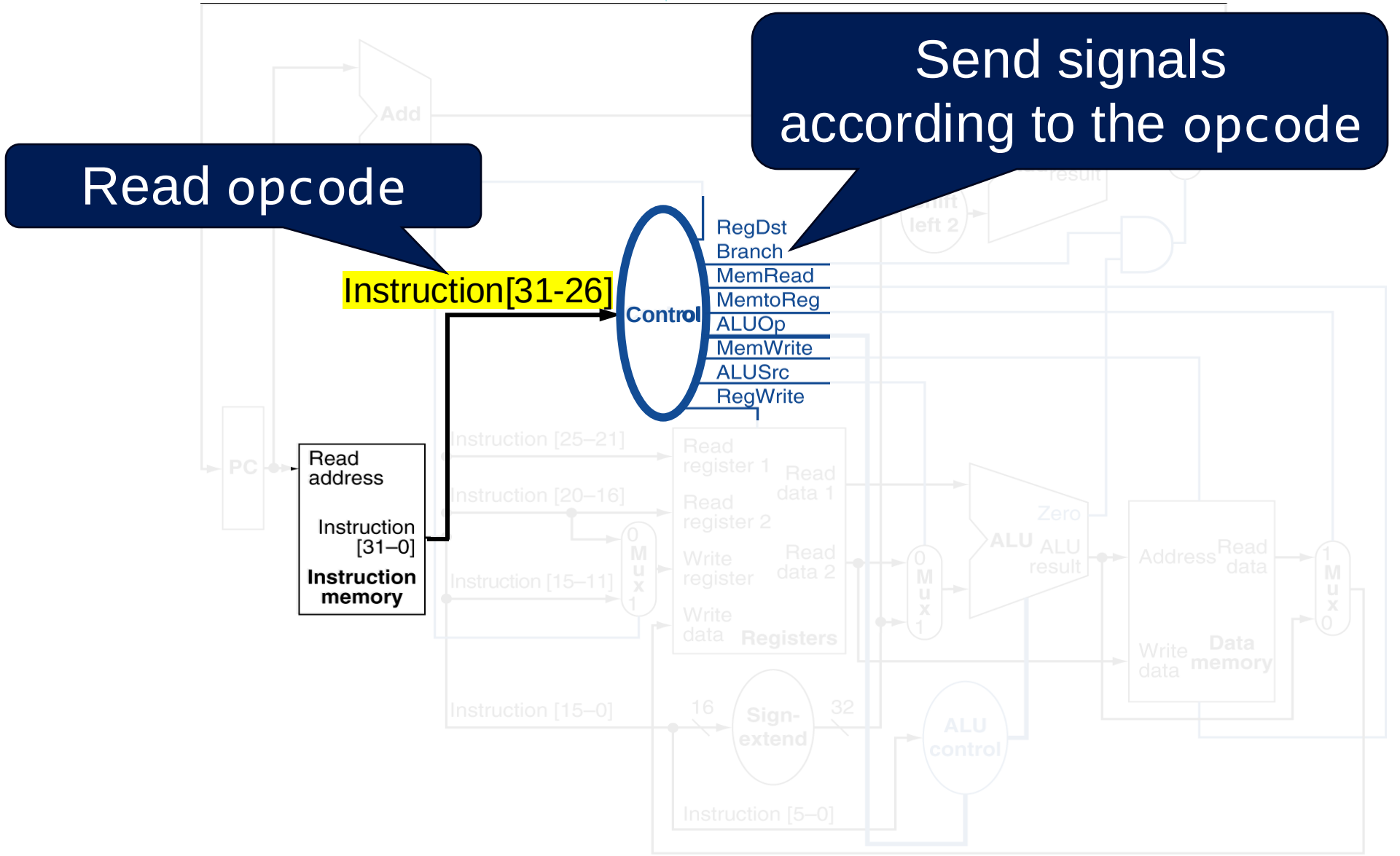


Datapath with Control: Read opcode

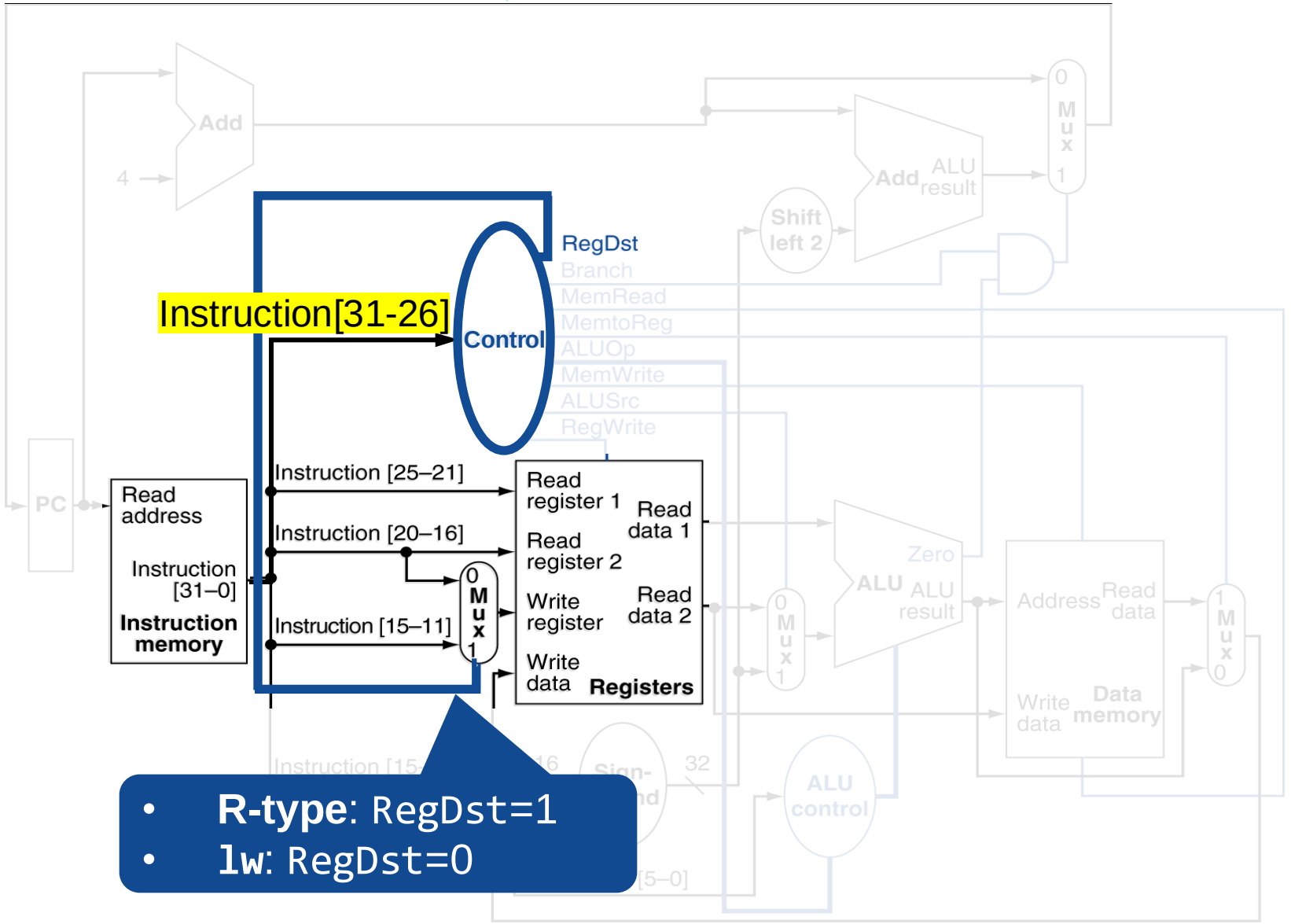


R-type:	op	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
I-type:	op	rs	rt	Address		
	31:26	25:21	20:16	16:0		
J-type:	op	address				
	31:26	26:0				

Datapath with Control: Send Signals



Control Signal Example: RegDst



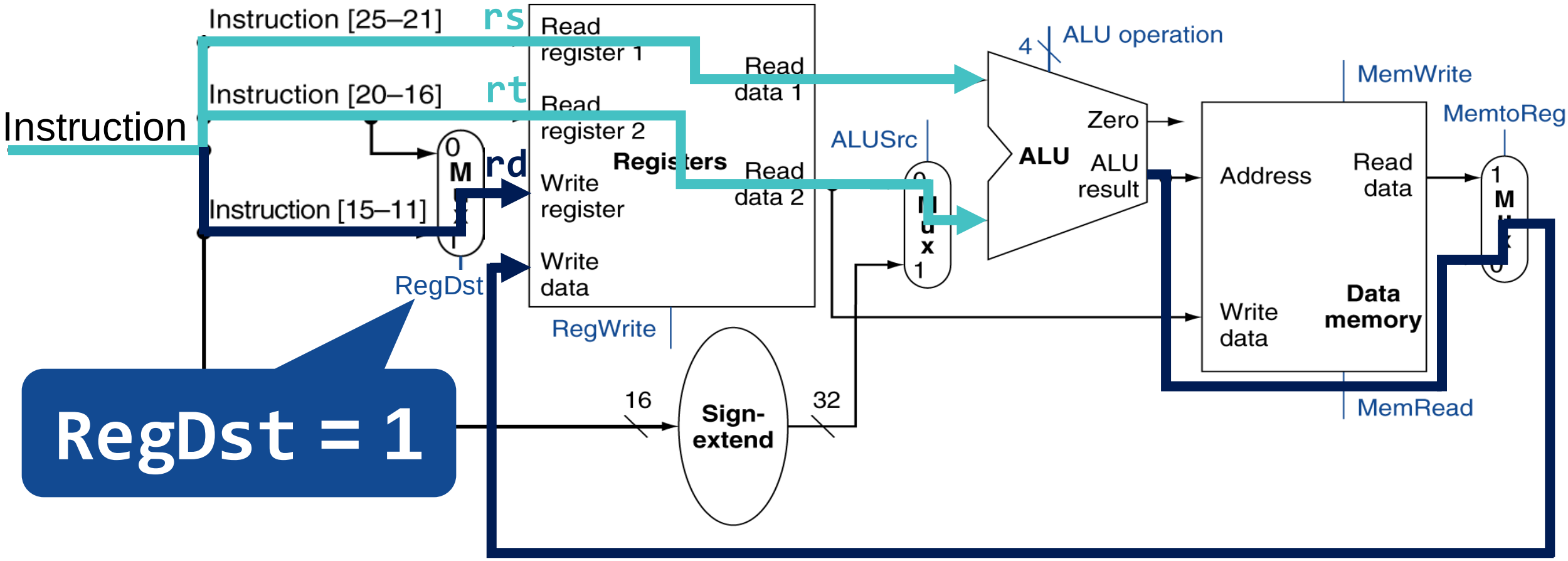
Example: RegDst with R-type Instruction

add instruction

`R[rd] =R[rs]+R[rt]`

R-type:

op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0



RegDst = 1

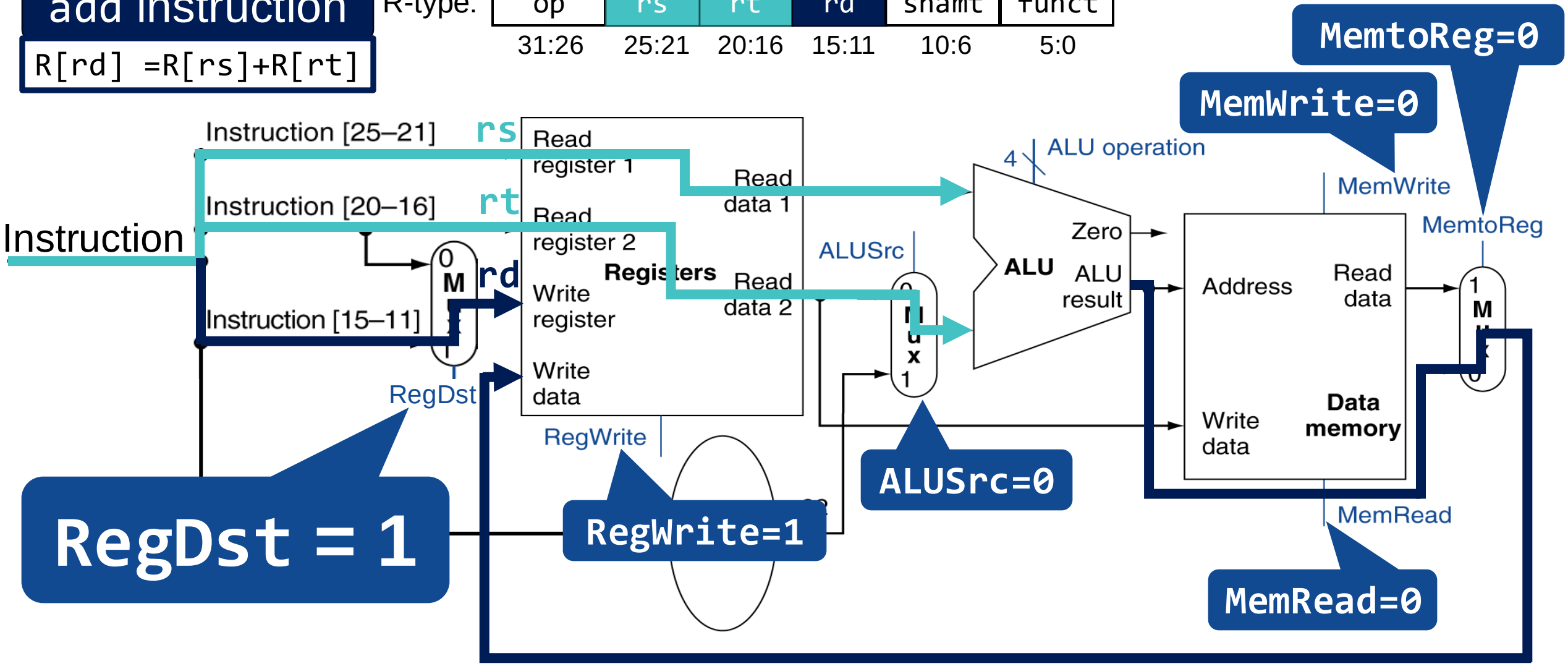
Example: RegDst with R-type Instruction

add instruction

`R[rd] =R[rs]+R[rt]`

R-type:

op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0



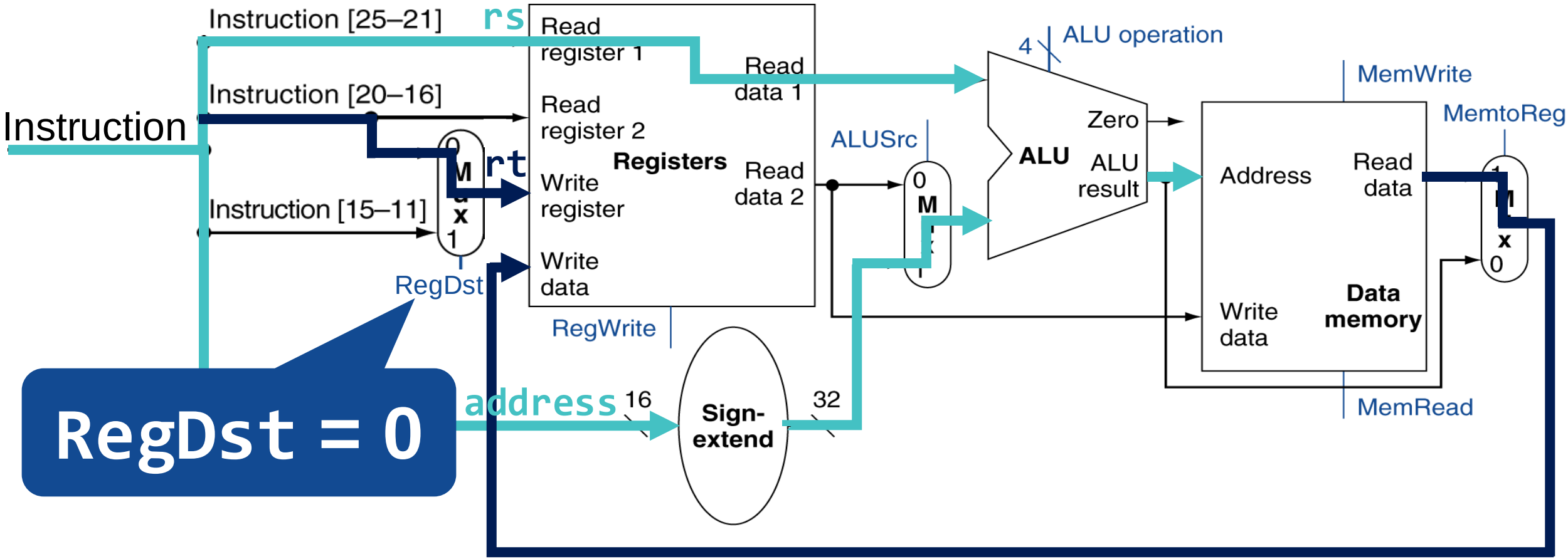
Example: RegDst with Load Instruction

lw instruction

I-type:

op	rs	rt	address
31:26	25:21	20:16	15:0

```
R[rt] = M[R[rs]+SignExtImm]
```

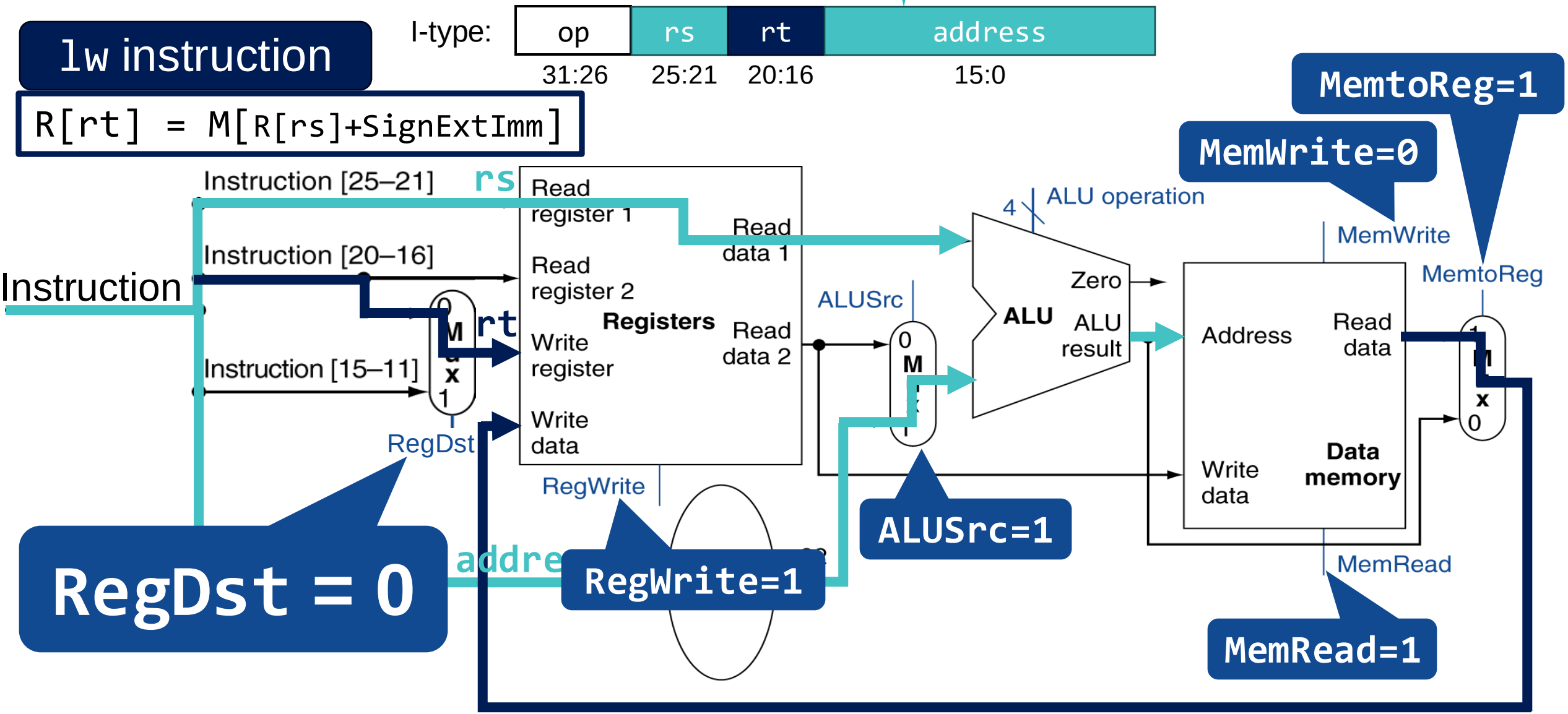


RegDst = 0

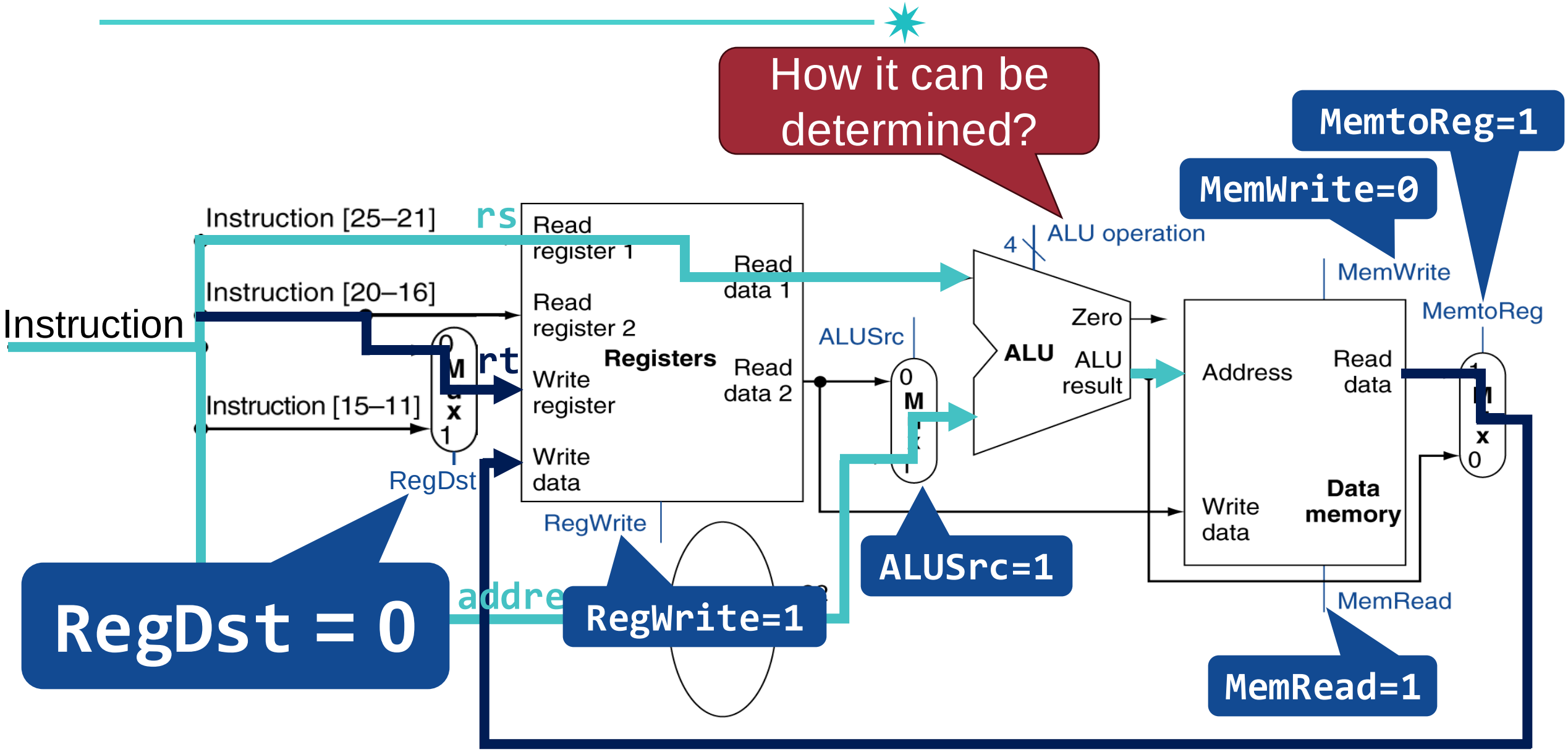
Example: RegDst with Load Instruction

`lw` instruction

`R[rt] = M[R[rs]+SignExtImm]`



Introduction to ALU



First of all, let's take a look at the details of the ALU

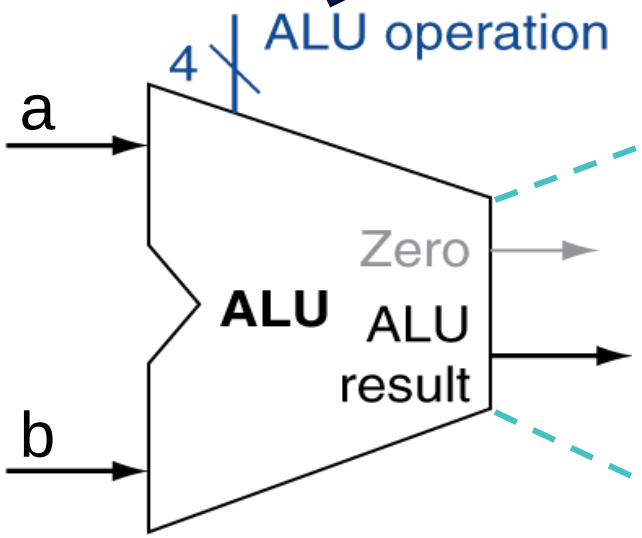
Arithmetic Logic Unit (ALU)

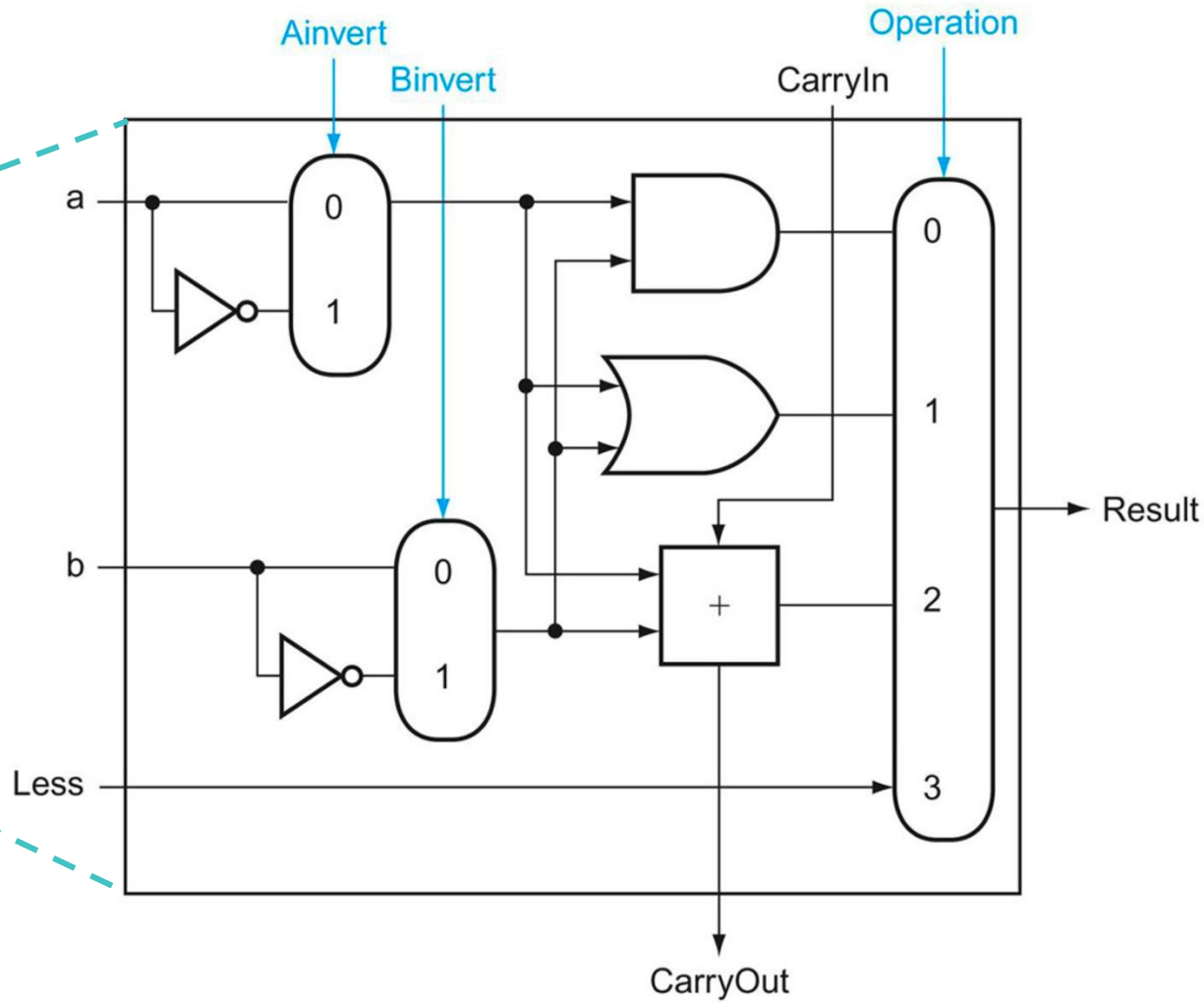
Perform arithmetic and logic operations on binary numbers

ALU with 4-bit Control Signals

Perform arithmetic and logic operations on binary numbers

4-bit control signals

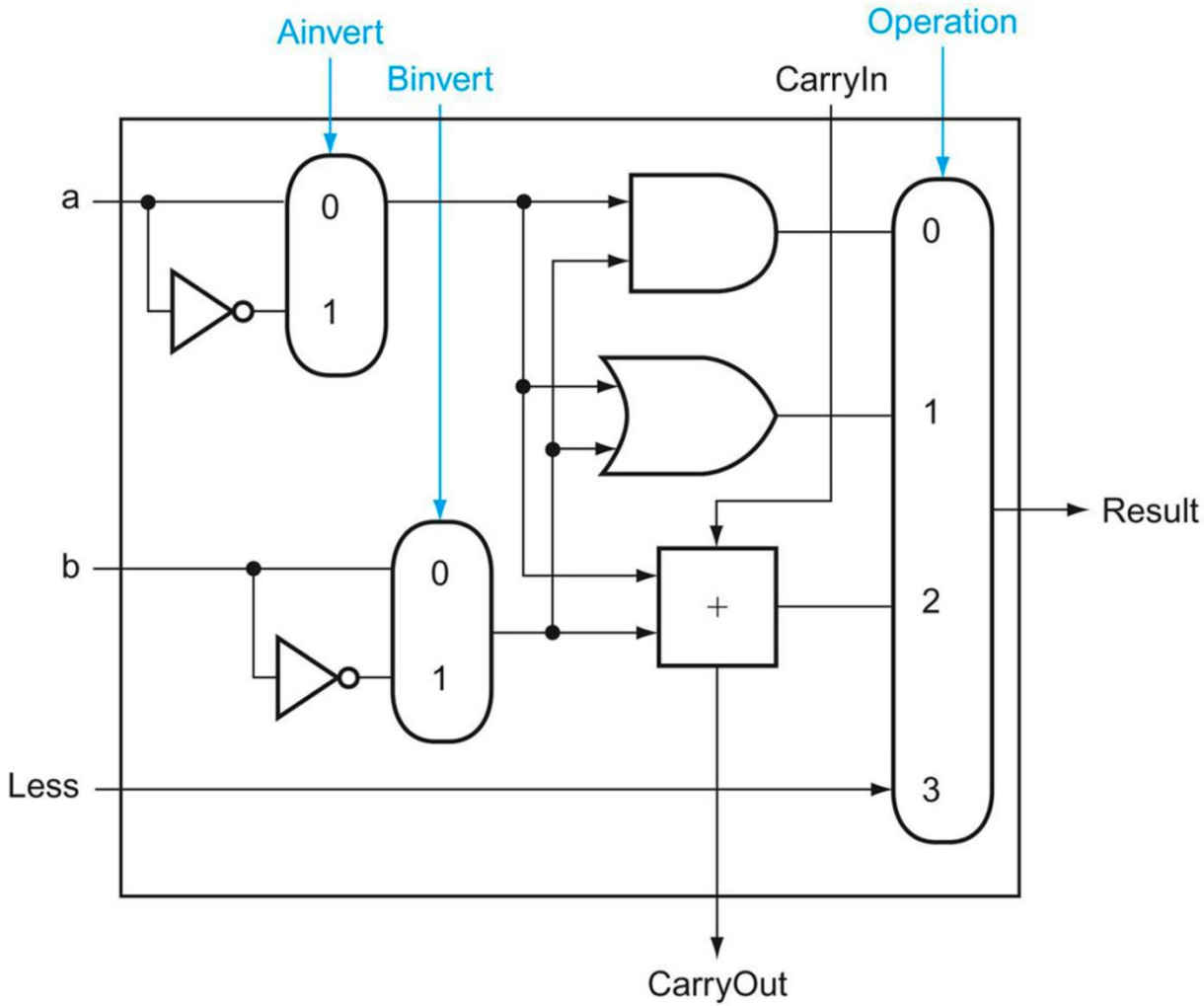




ALU with 4-bit Control Signals

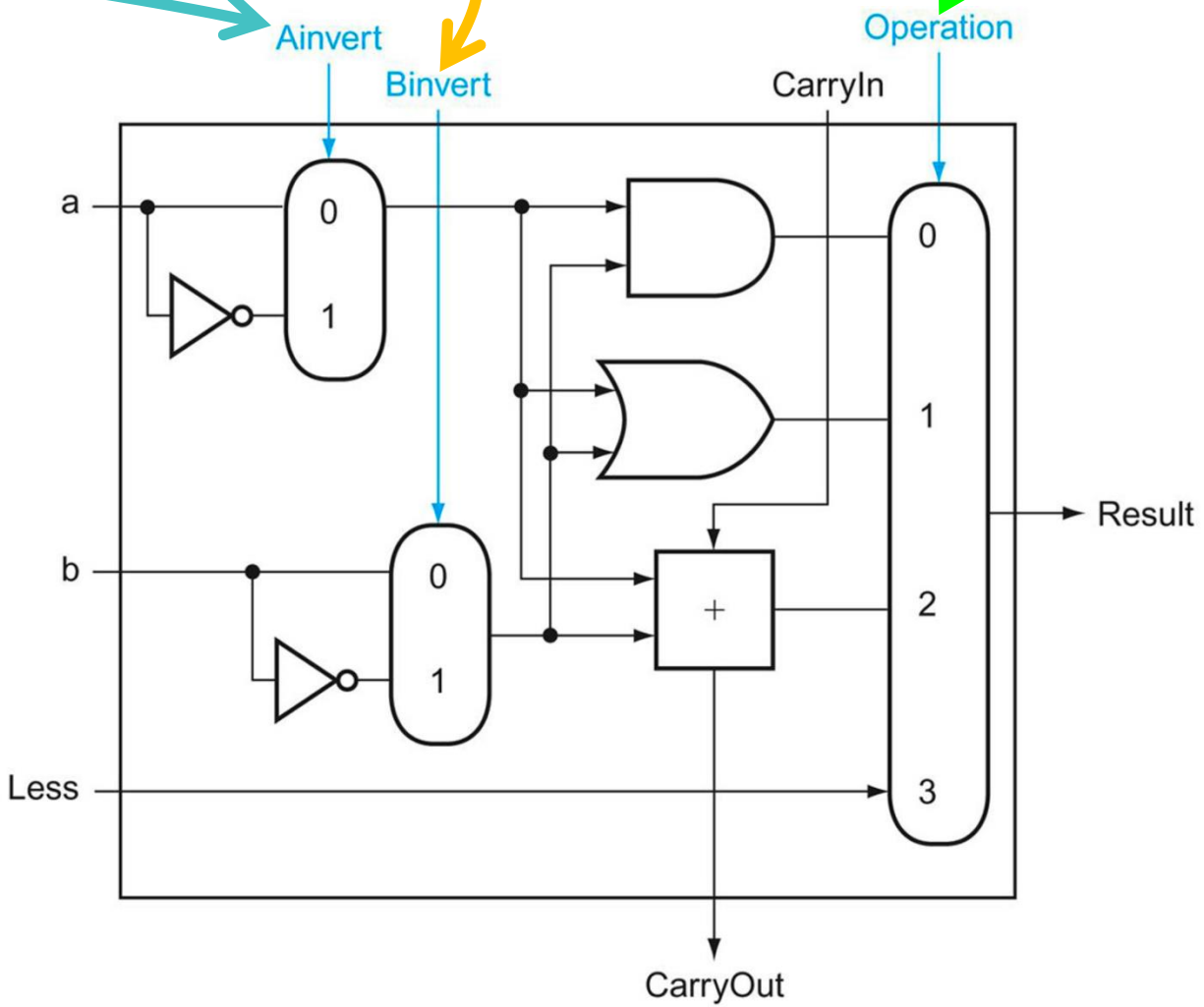
Perform arithmetic and logic operations on binary numbers

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



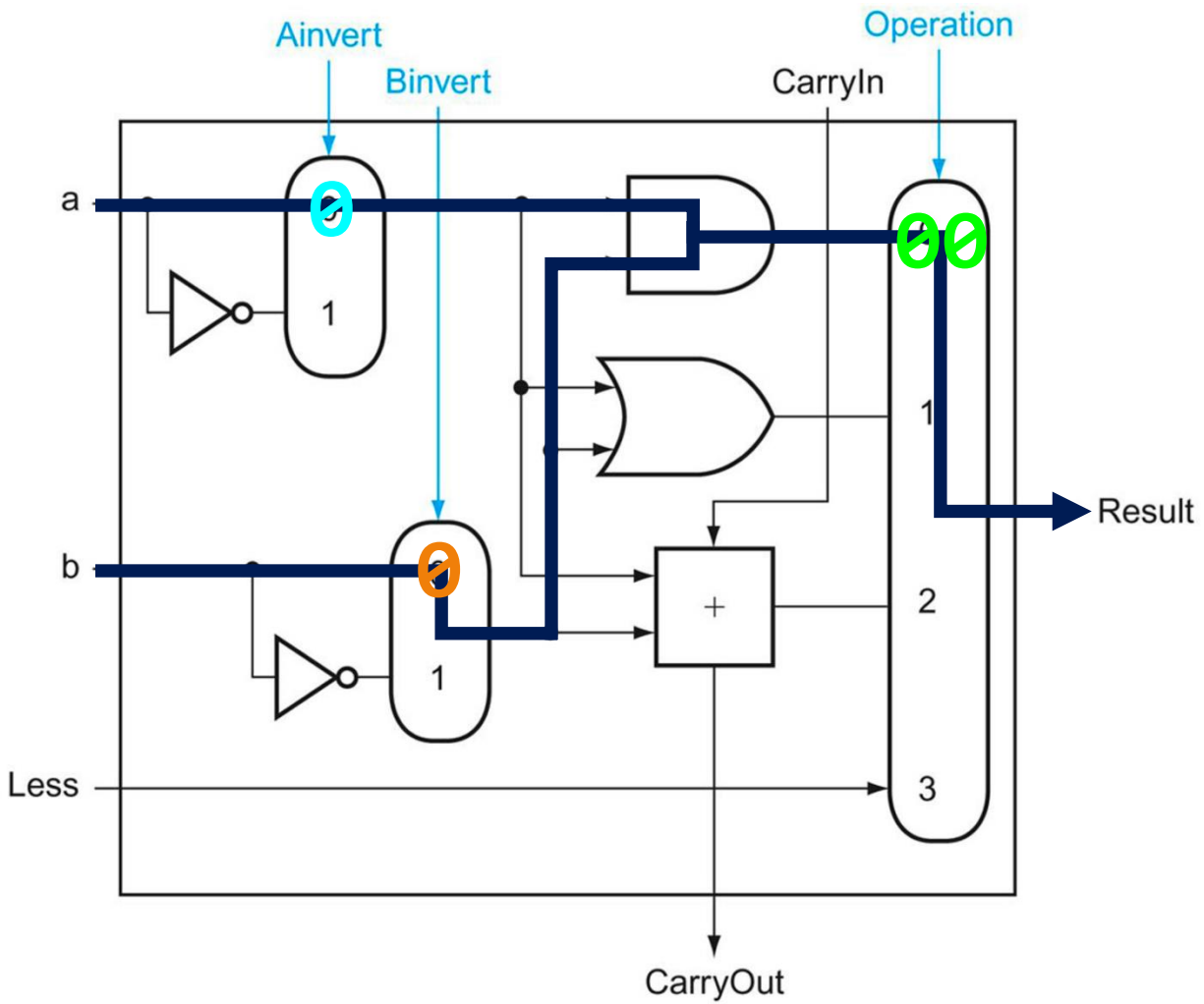
ALU with 4-bit Control Signals

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



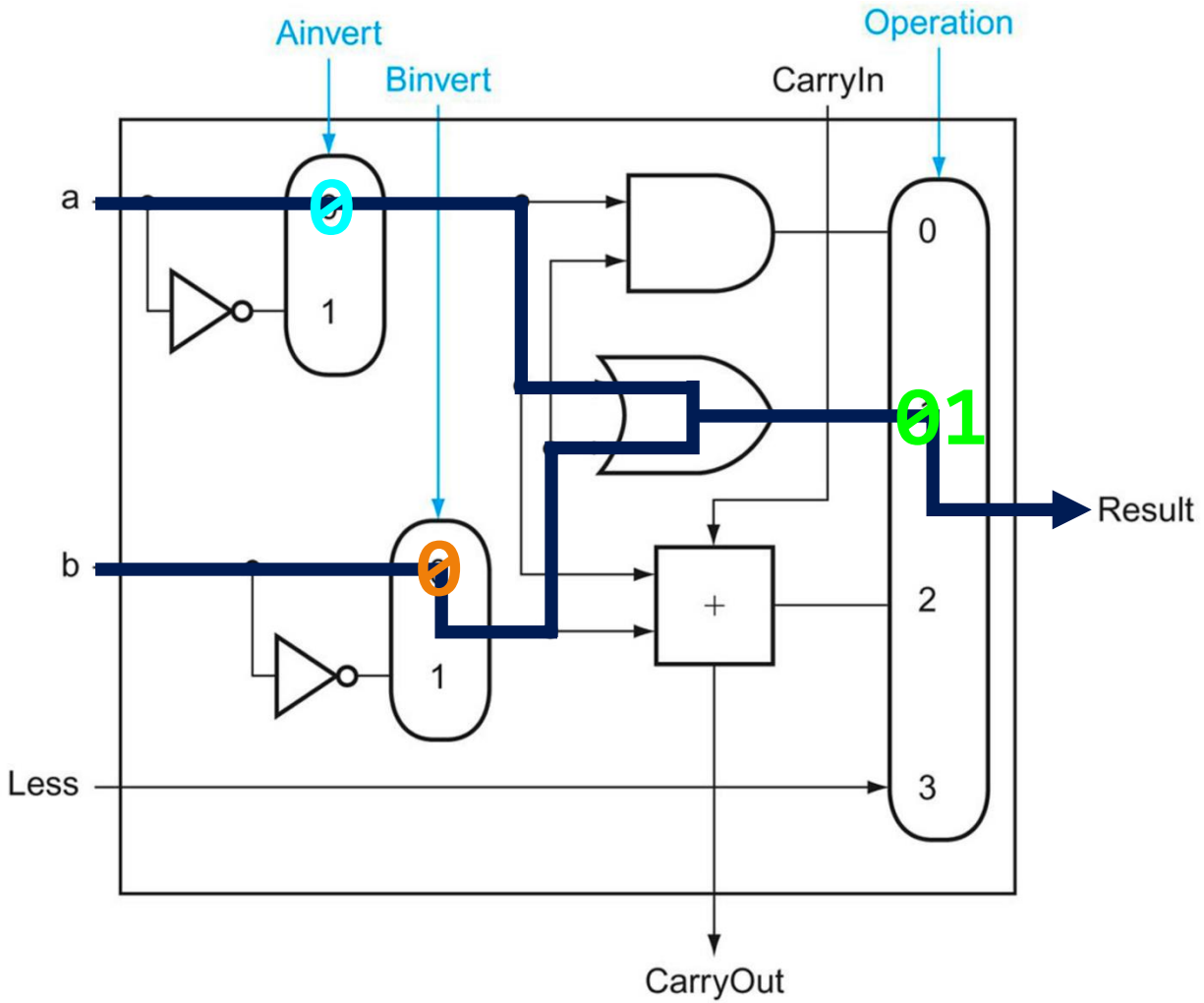
ALU with 4-bit Control Signals: And

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



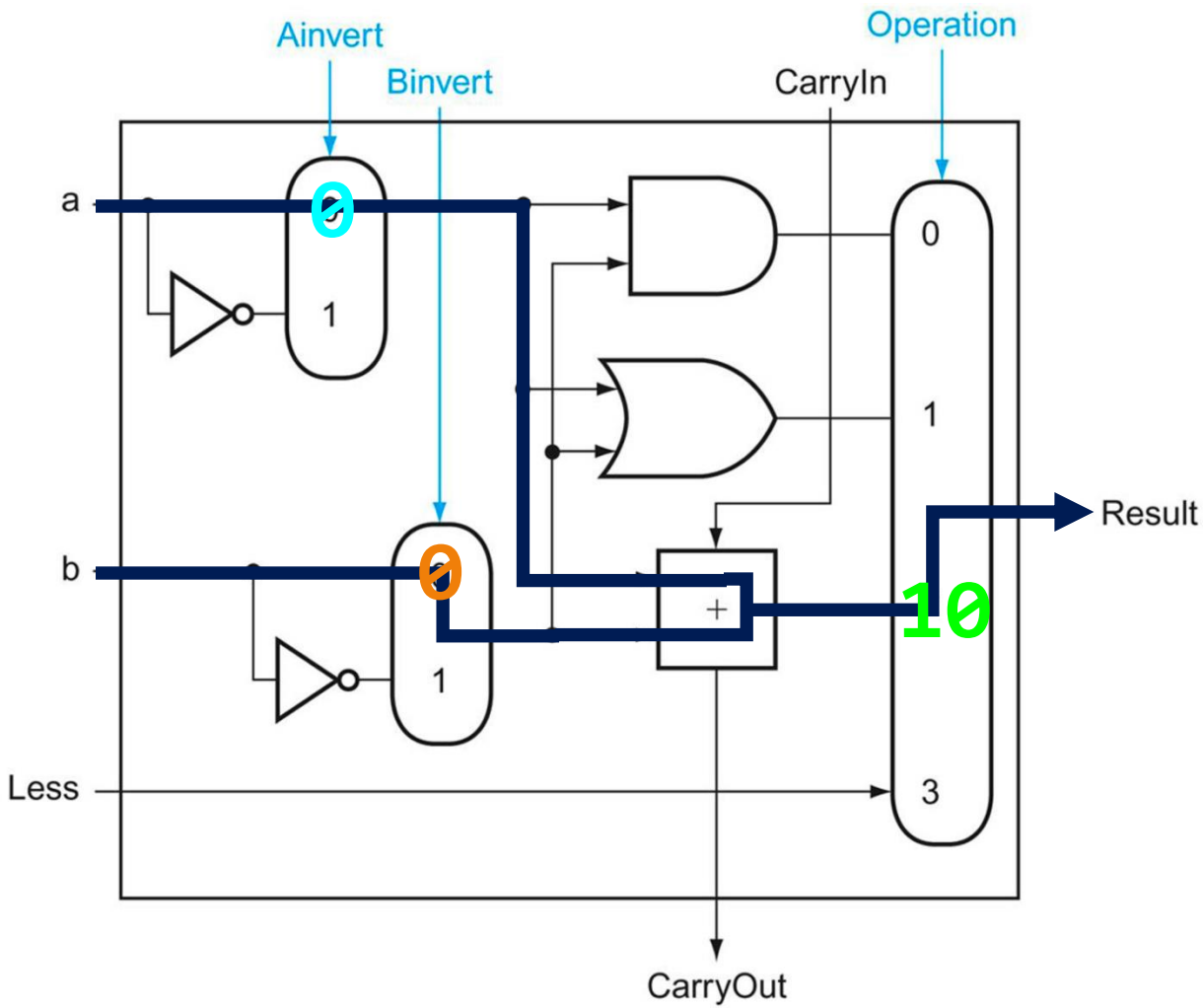
ALU with 4-bit Control Signals: Or

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



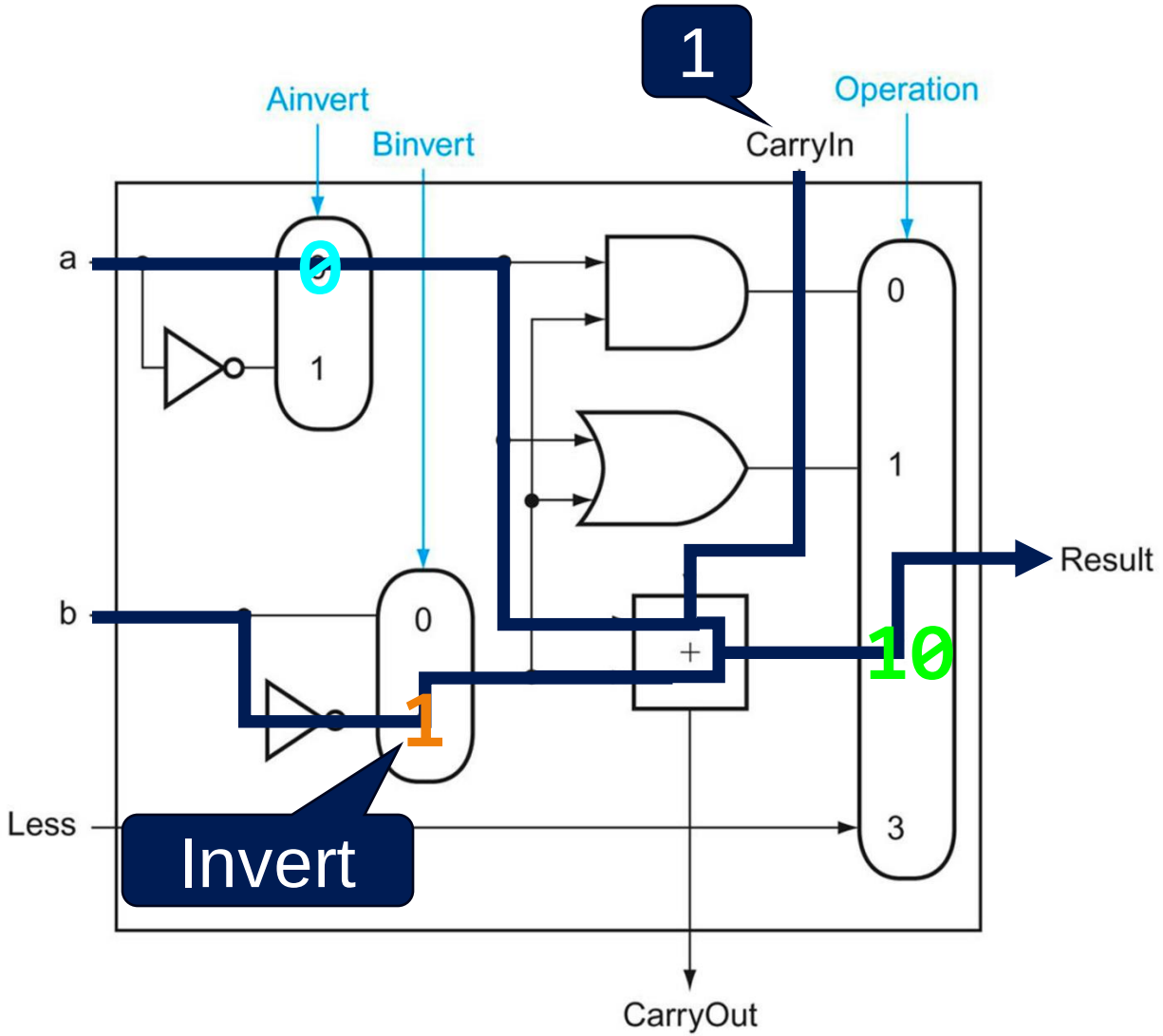
ALU with 4-bit Control Signals: Add

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



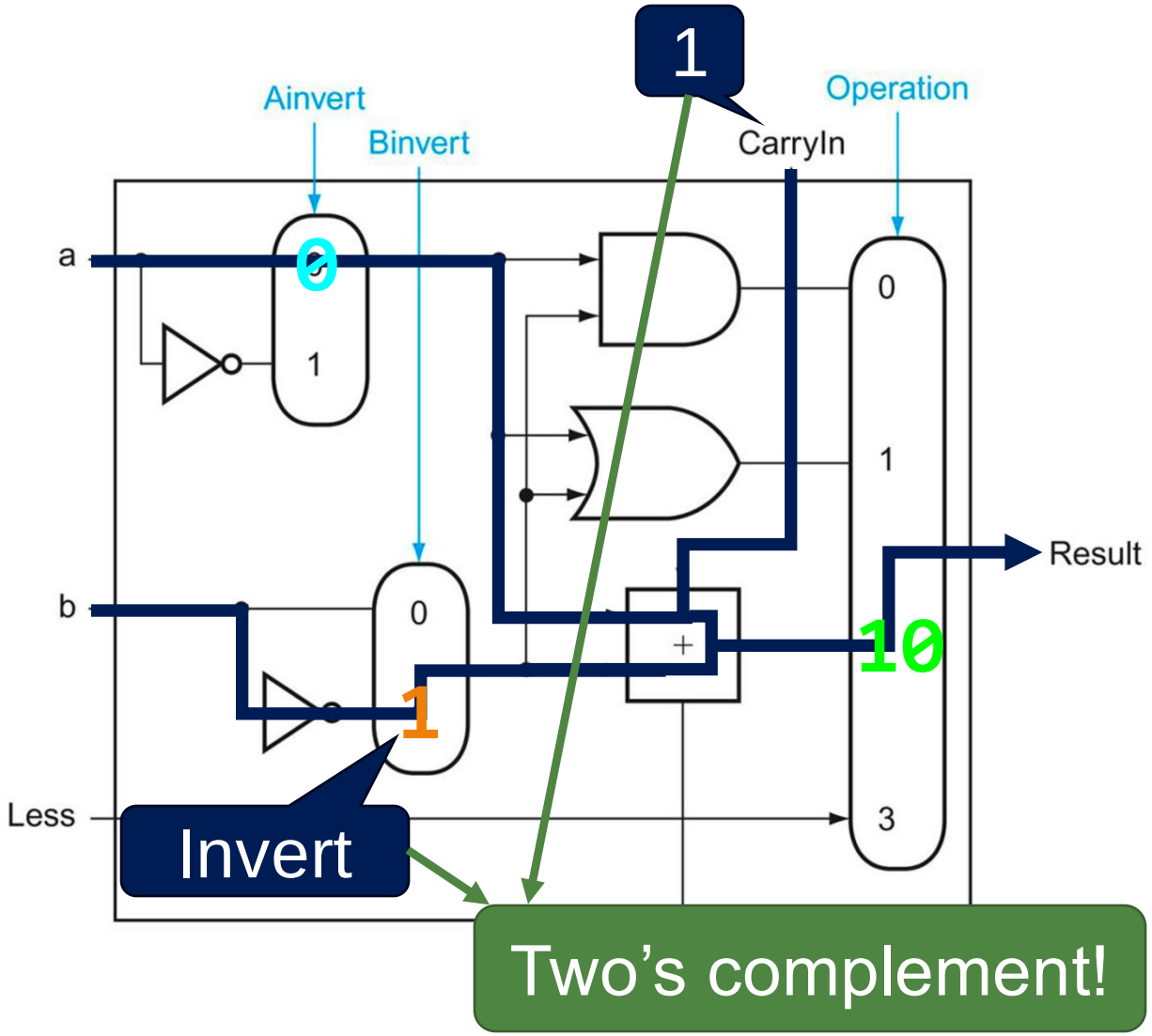
ALU with 4-bit Control Signals: Subtract

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



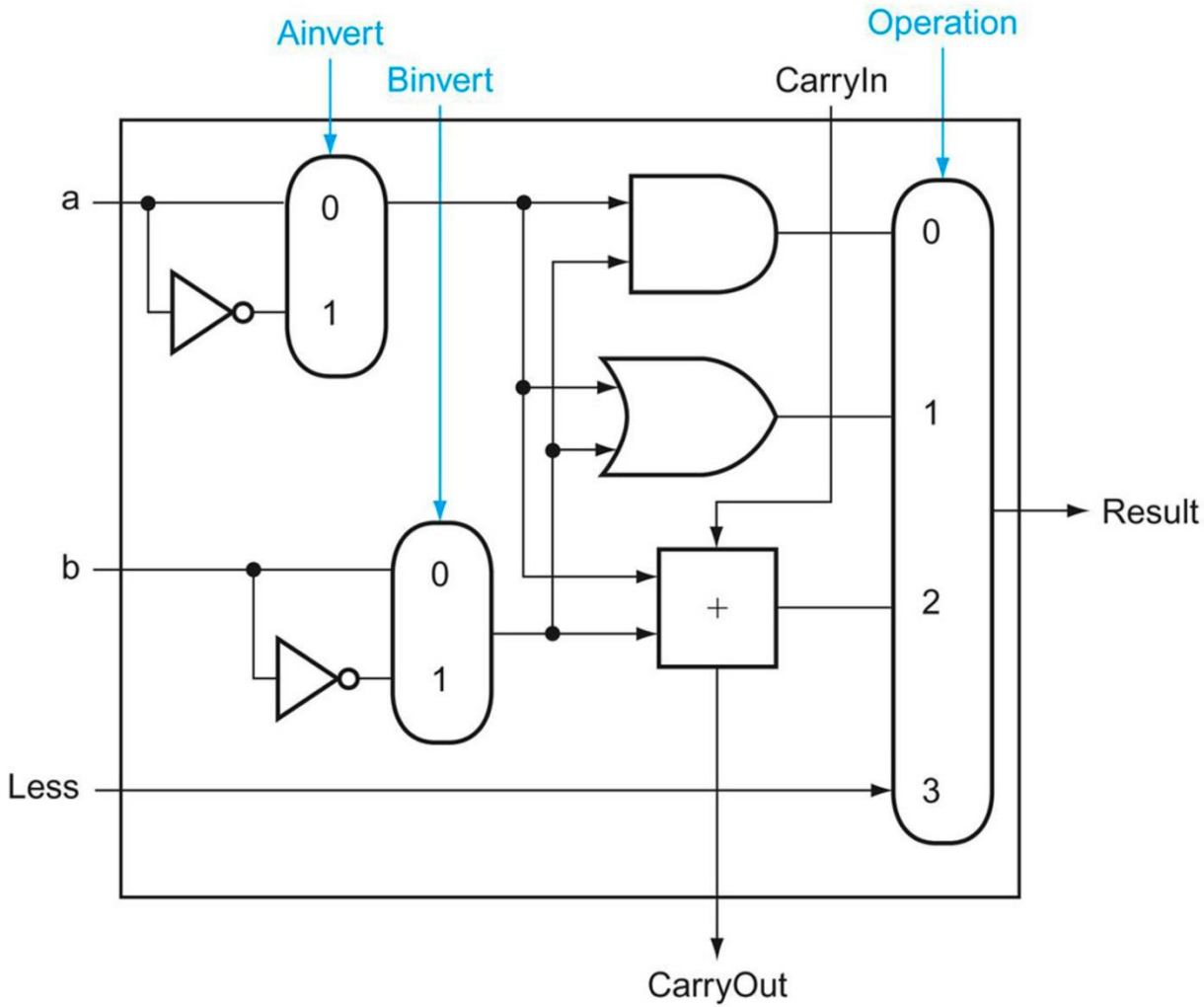
ALU with 4-bit Control Signals: Subtract

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor



ALU with 4-bit Control Signals

ALU control (4-bit control signal)	Function
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	Nor

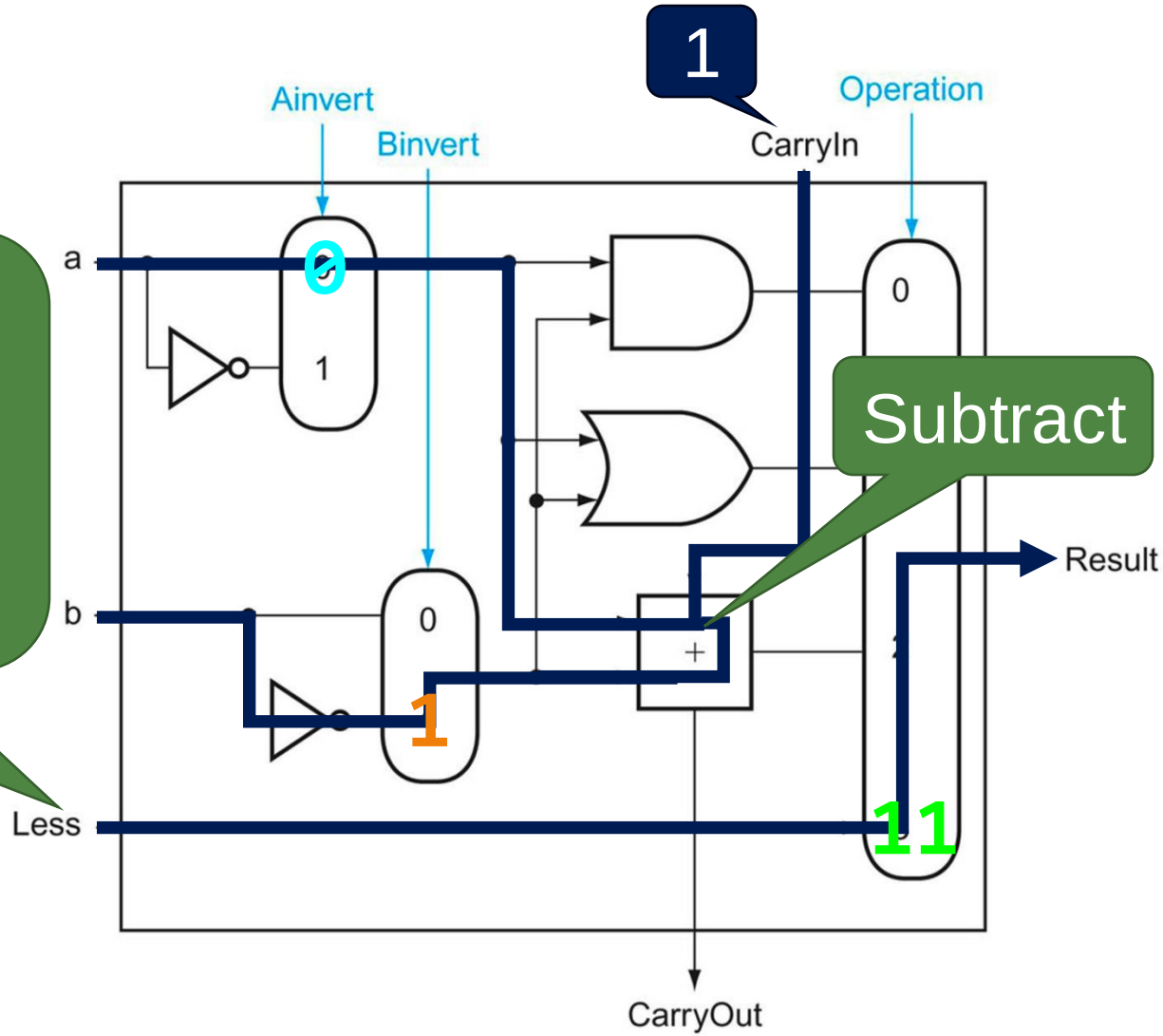


ALU with 4-bit Control Signals: SLT

ALU control (4-bit control signal)	Function
0111	Set on less than

(Skip the detailed HW)

```
if a - b < 0:    # a < b
    Less = 1
else:
    Less = 0
```



Introduction to ALU Control

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

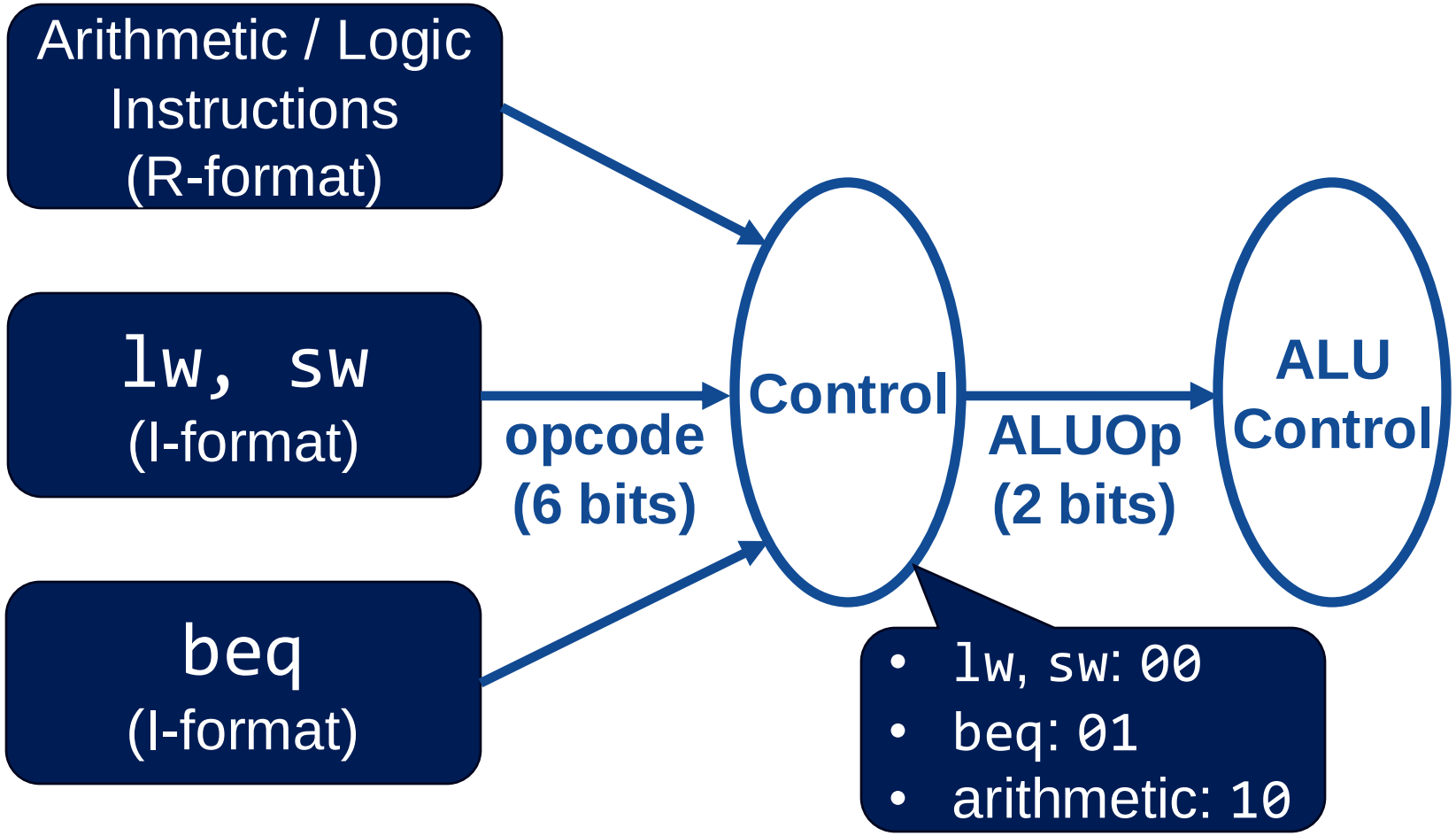
Arithmetic / Logic Instructions (R-format) → Depending on the instruction

`lw, sw` (I-format) → Addition

`beq` (I-format) → Subtraction

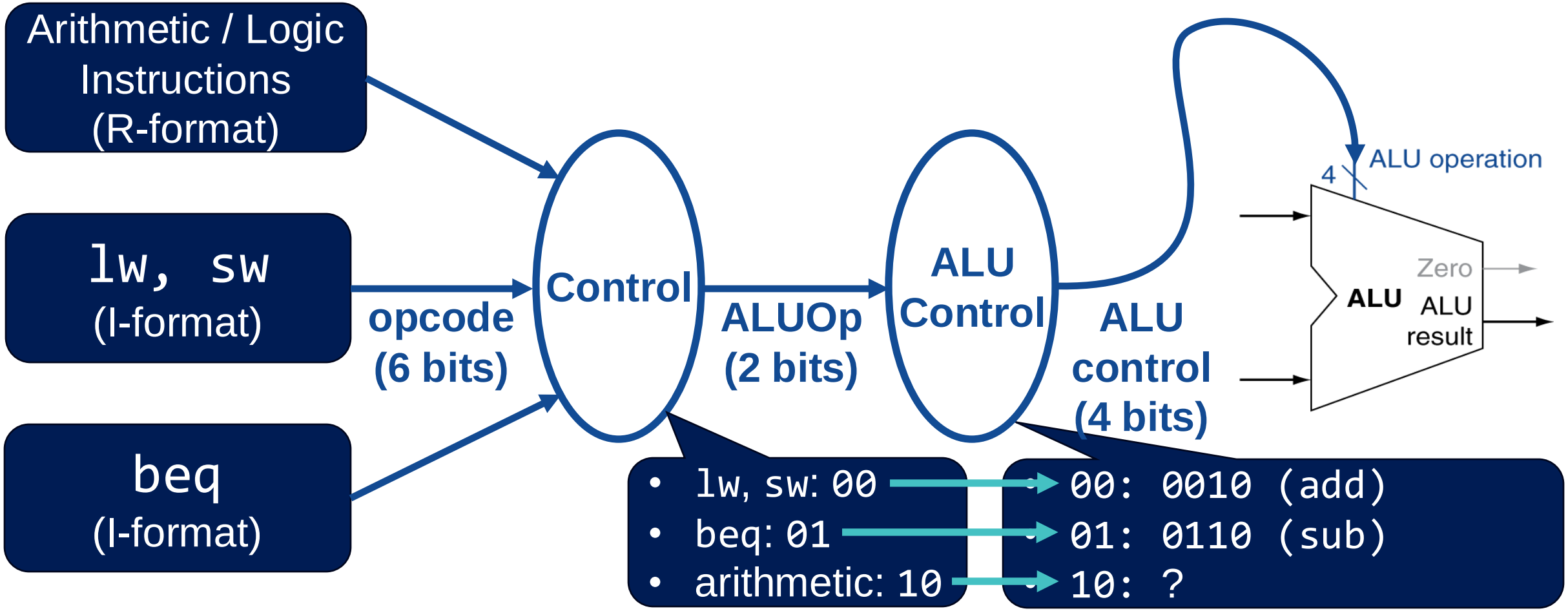
Introduction to ALU Control

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



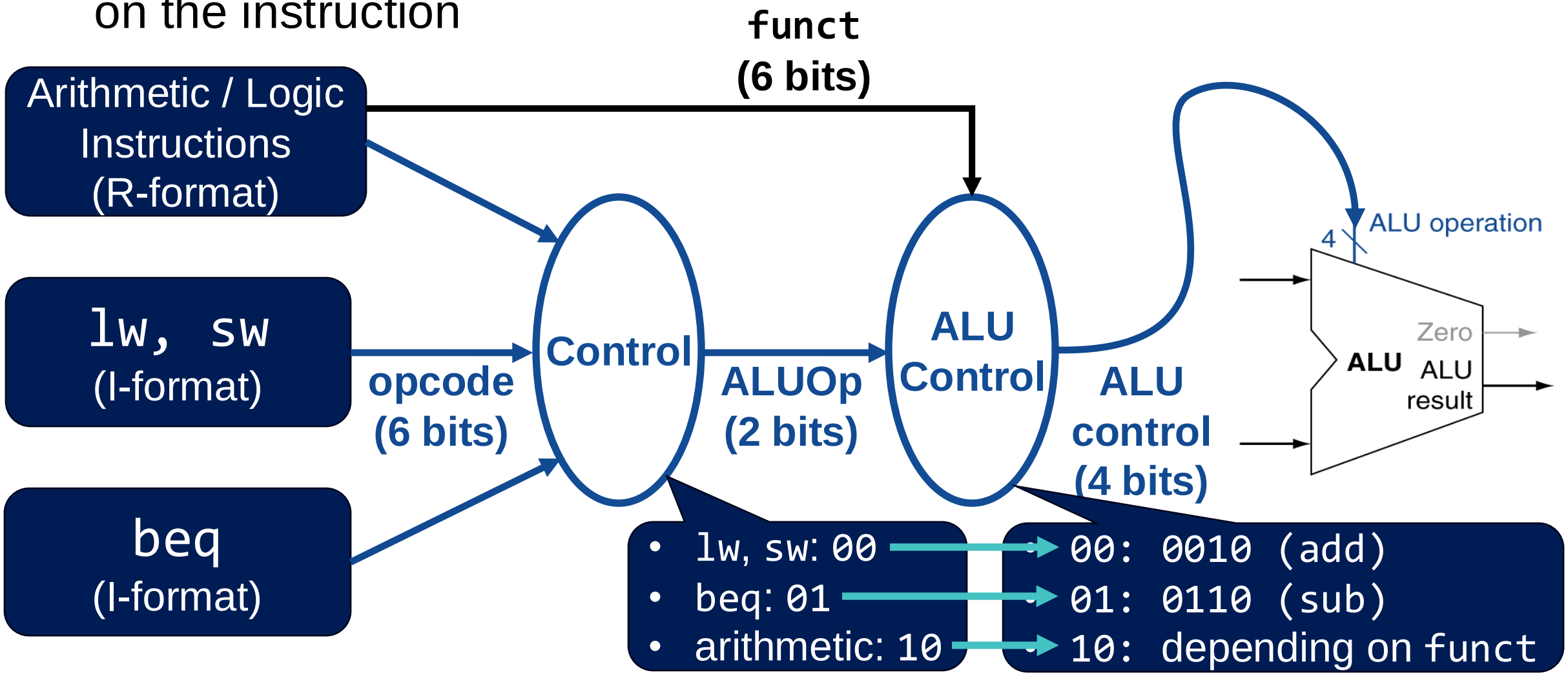
Introduction to ALU Control

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

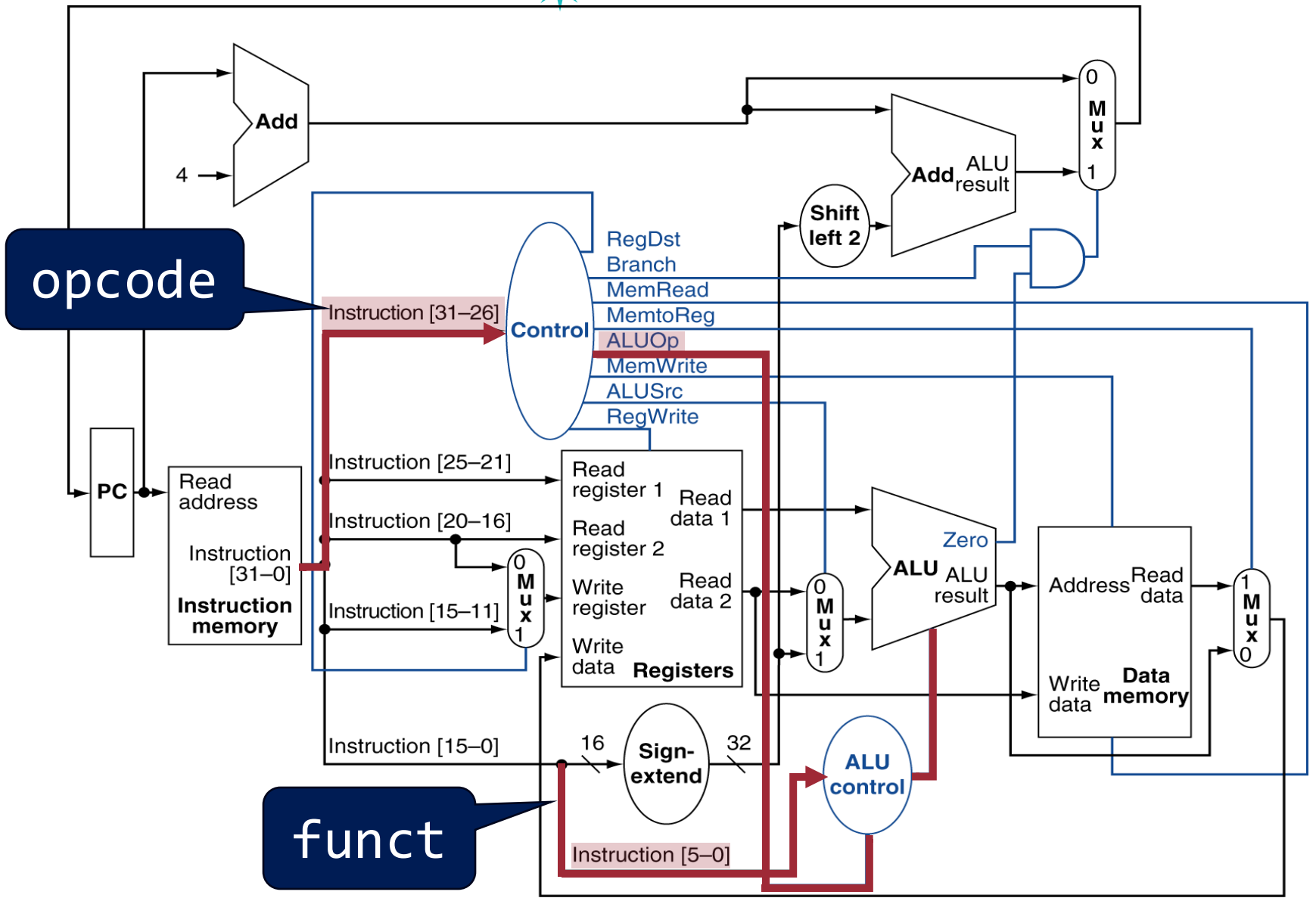


Introduction to ALU Control

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



Datapath with ALU Control



Summary: ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control
  - What's the benefit of multi-level decoding for ALU control?

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

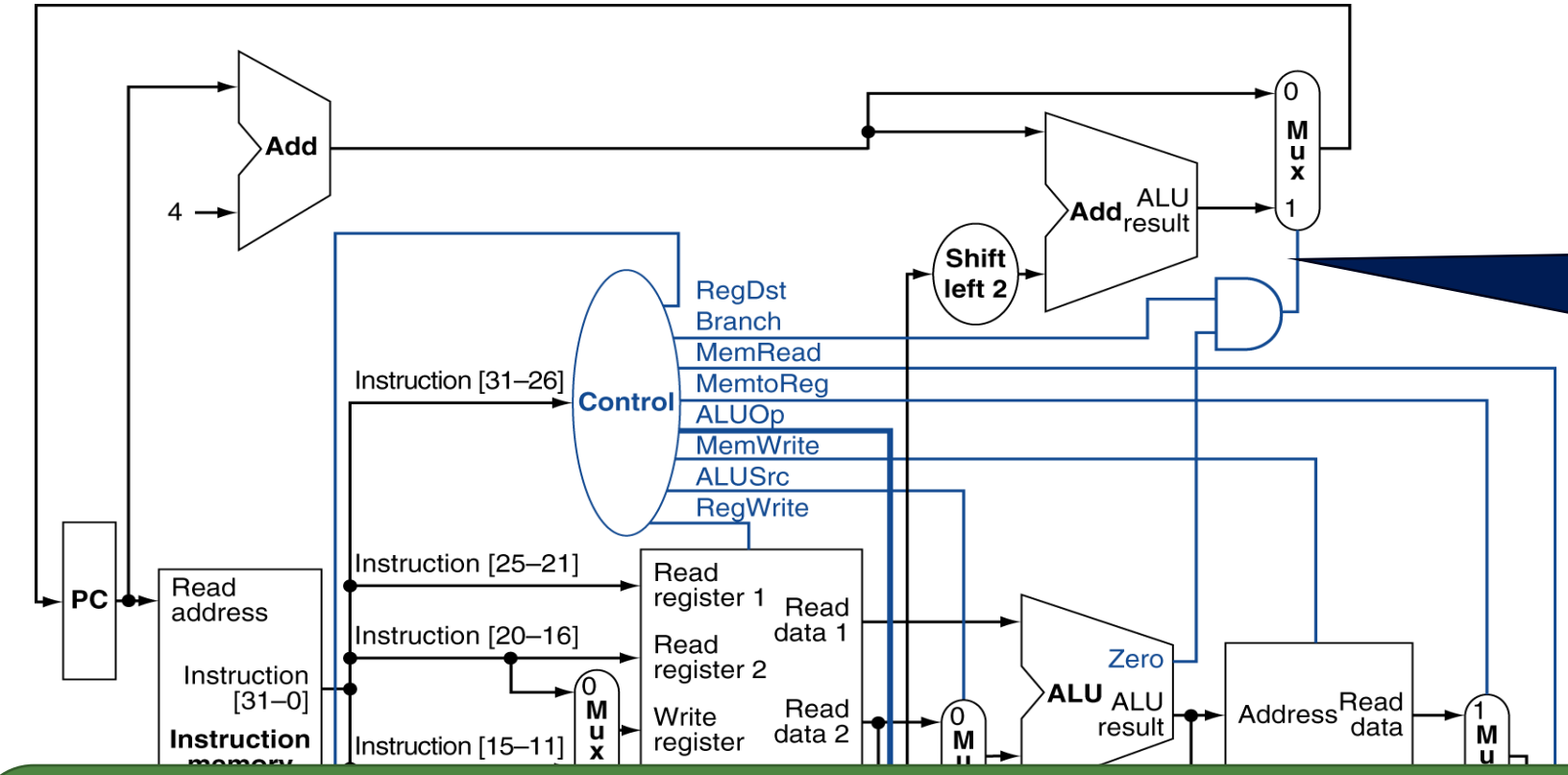
Summary: ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control
  - What's the benefit of multi-level decoding

X: "Don't care" about the value

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

One More Thing! Datapath for a Branch



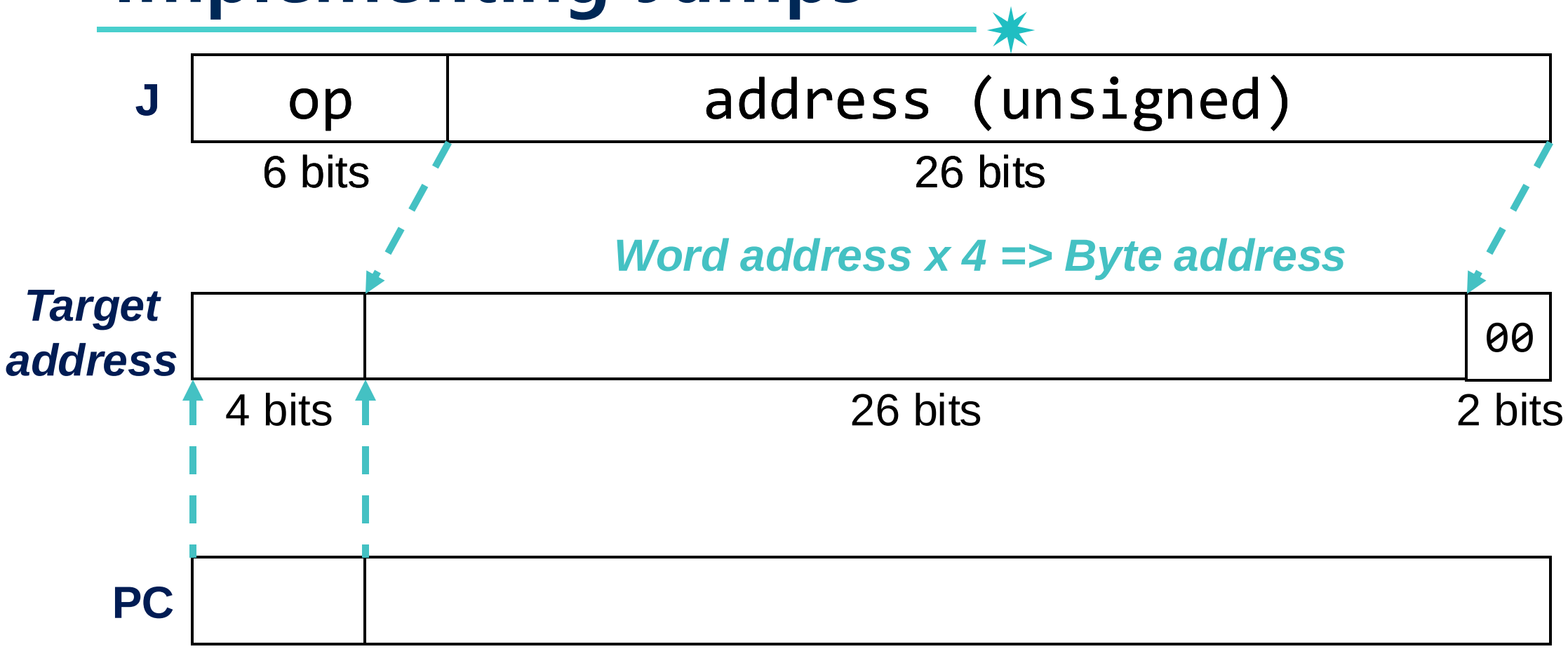
PCsrc = Branch && Zero

sub can also enable the Zero flag. Therefore, it's essential to check the opcode of the instruction to determine **whether it is a branch instruction or not**!

Summary: Control Signals

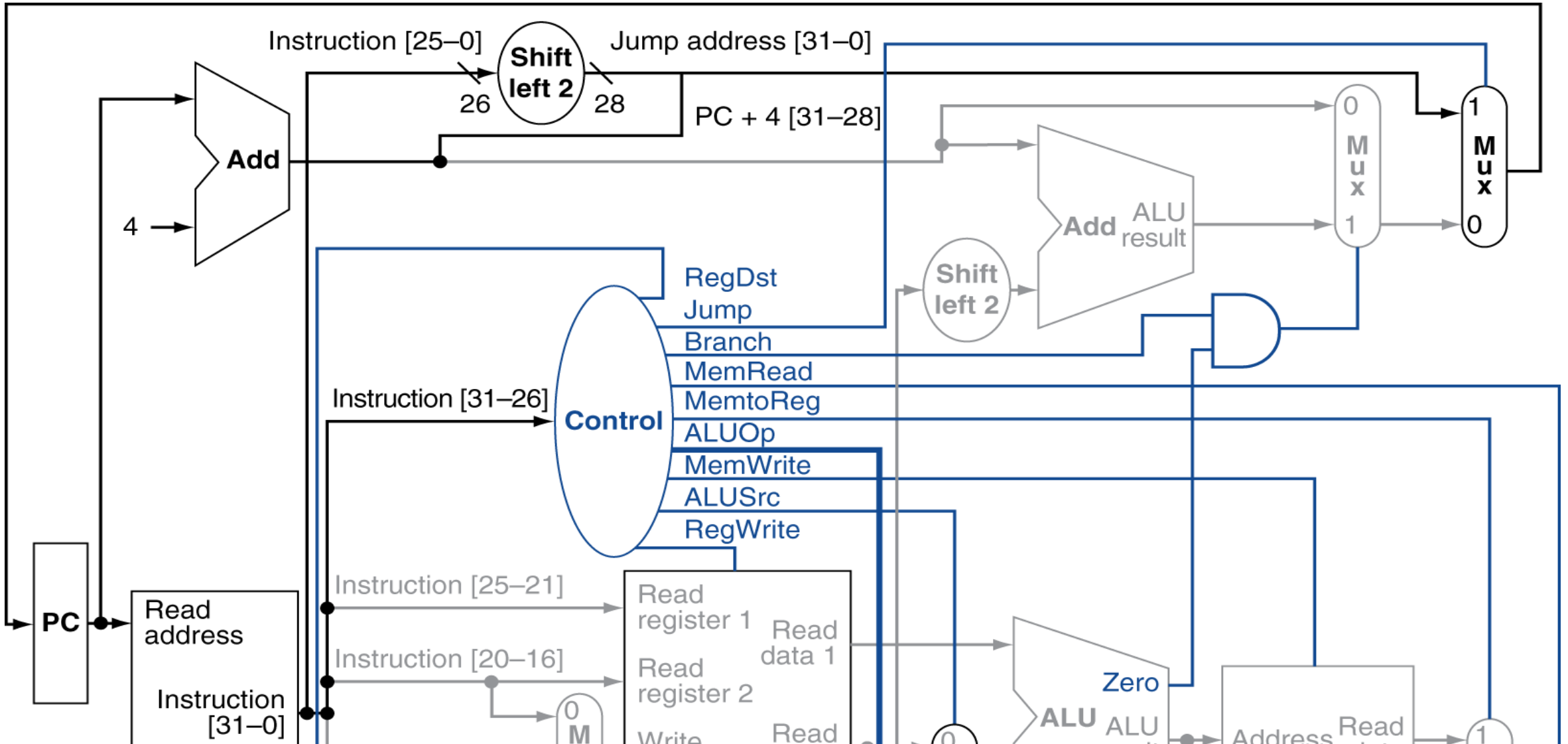
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input 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.

Implementing Jumps



Need an **extra control signal** decoded from opcode

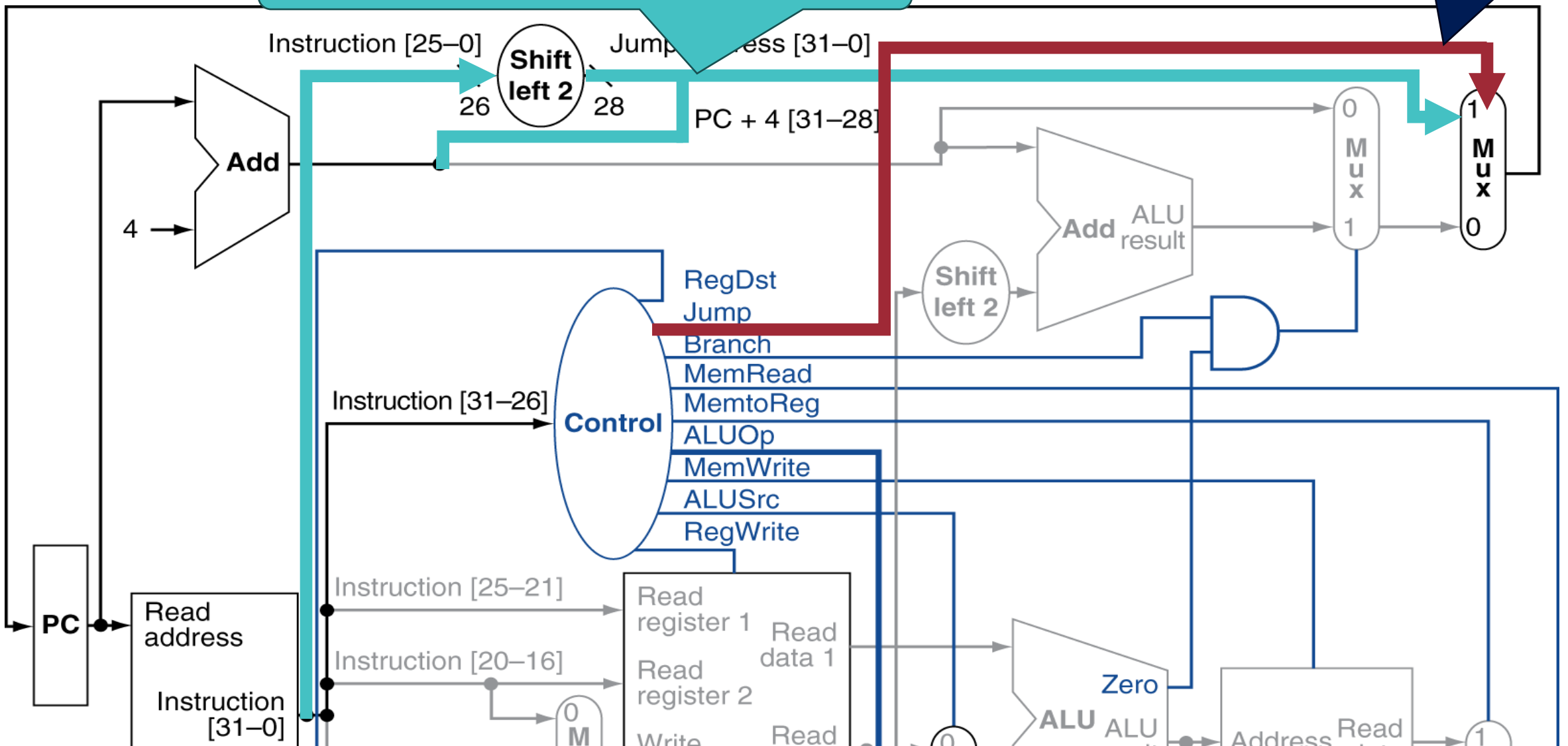
Datapath with Jumps Added



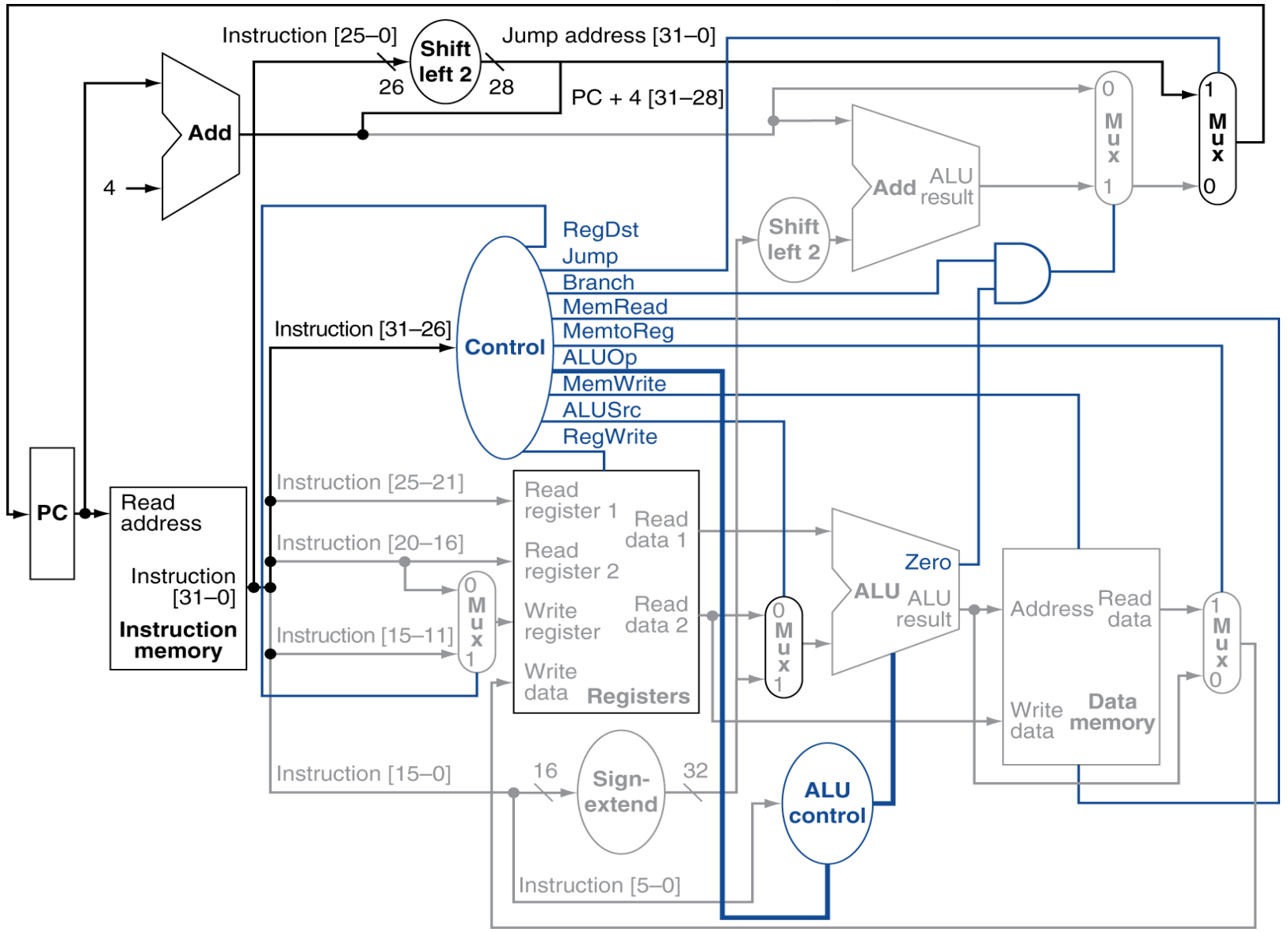
Datapath with Jumps Added

PC[31:28] && (address<<2)

Extra control signal: Jump



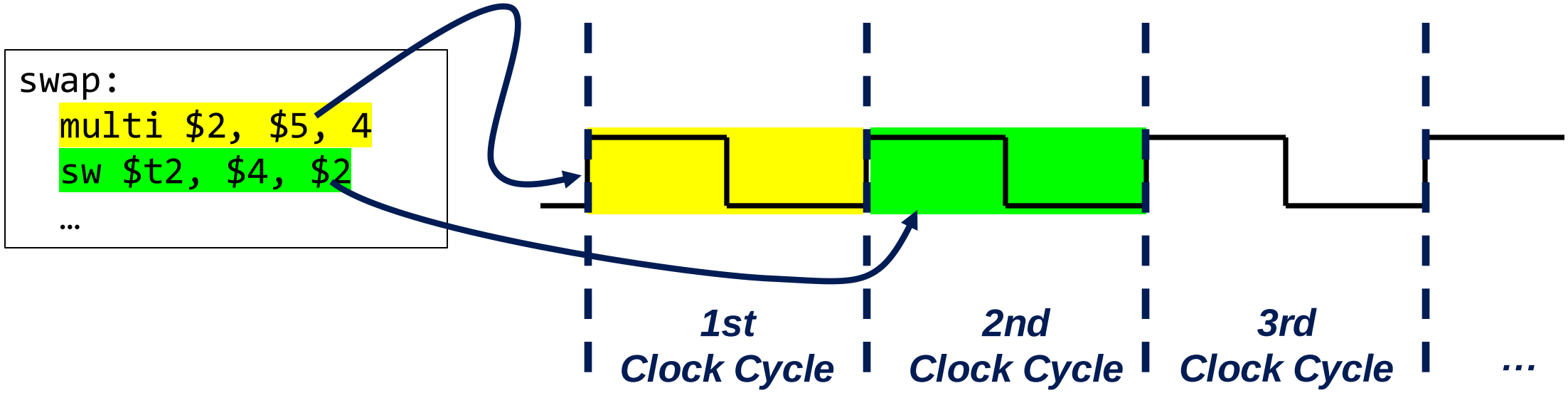
Final Version of Single Cycle Datapath



Our Assumption: Single Cycle Datapath

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

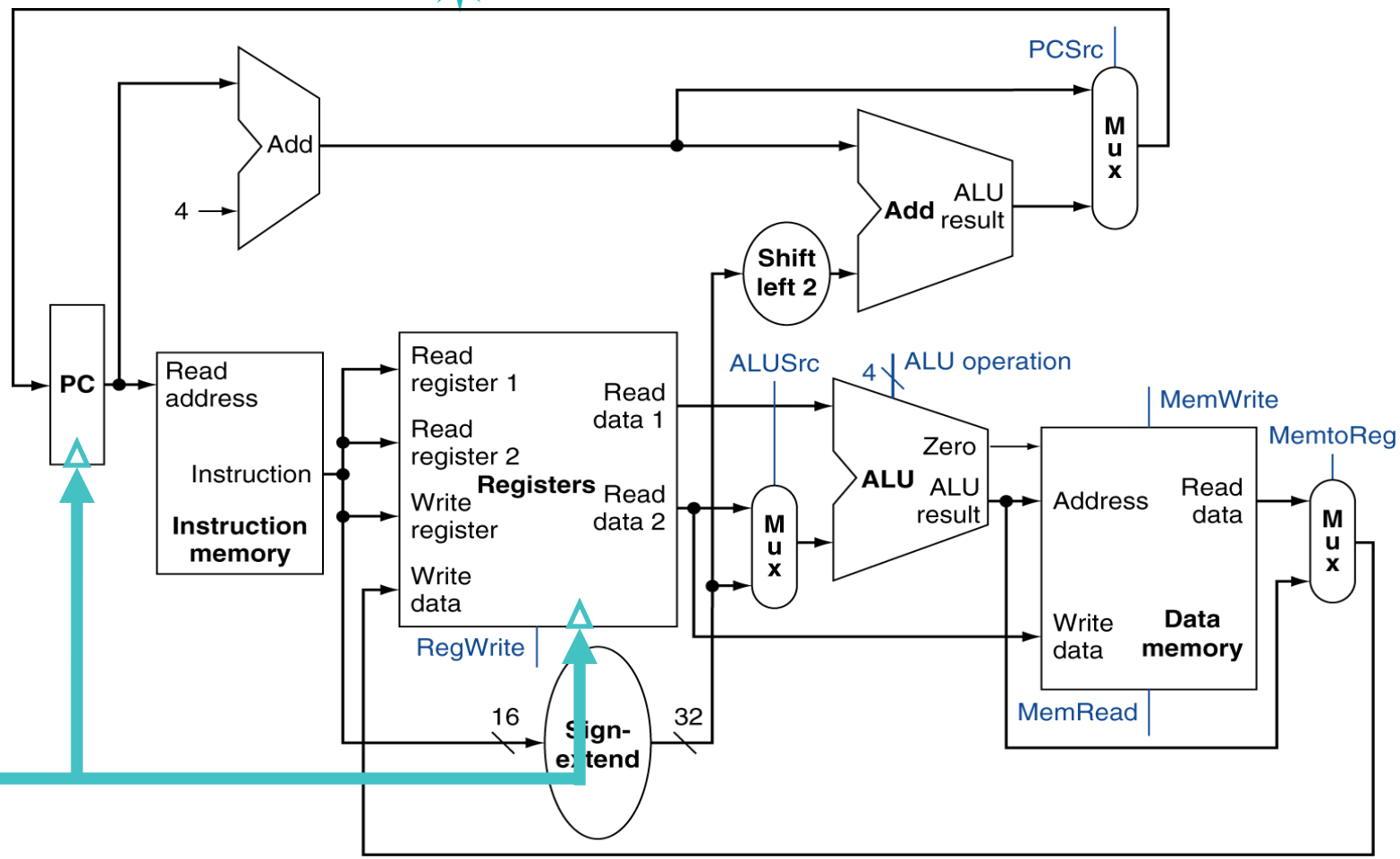
- Each instruction is executed in one clock cycle in the CPU



Let's look at the detailed scenario

Our Assumption: Single Cycle Datapath

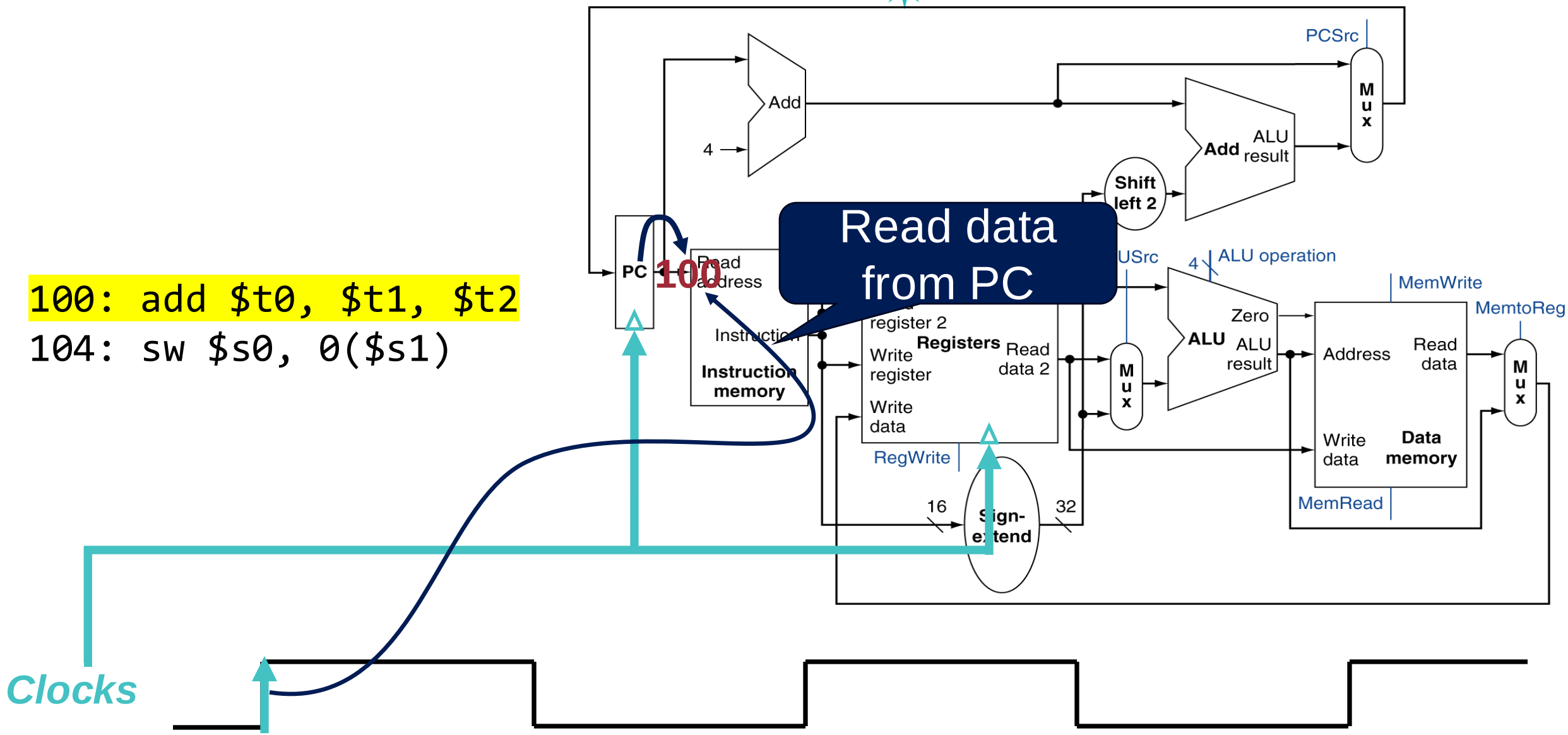
```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```



Clocks

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

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



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

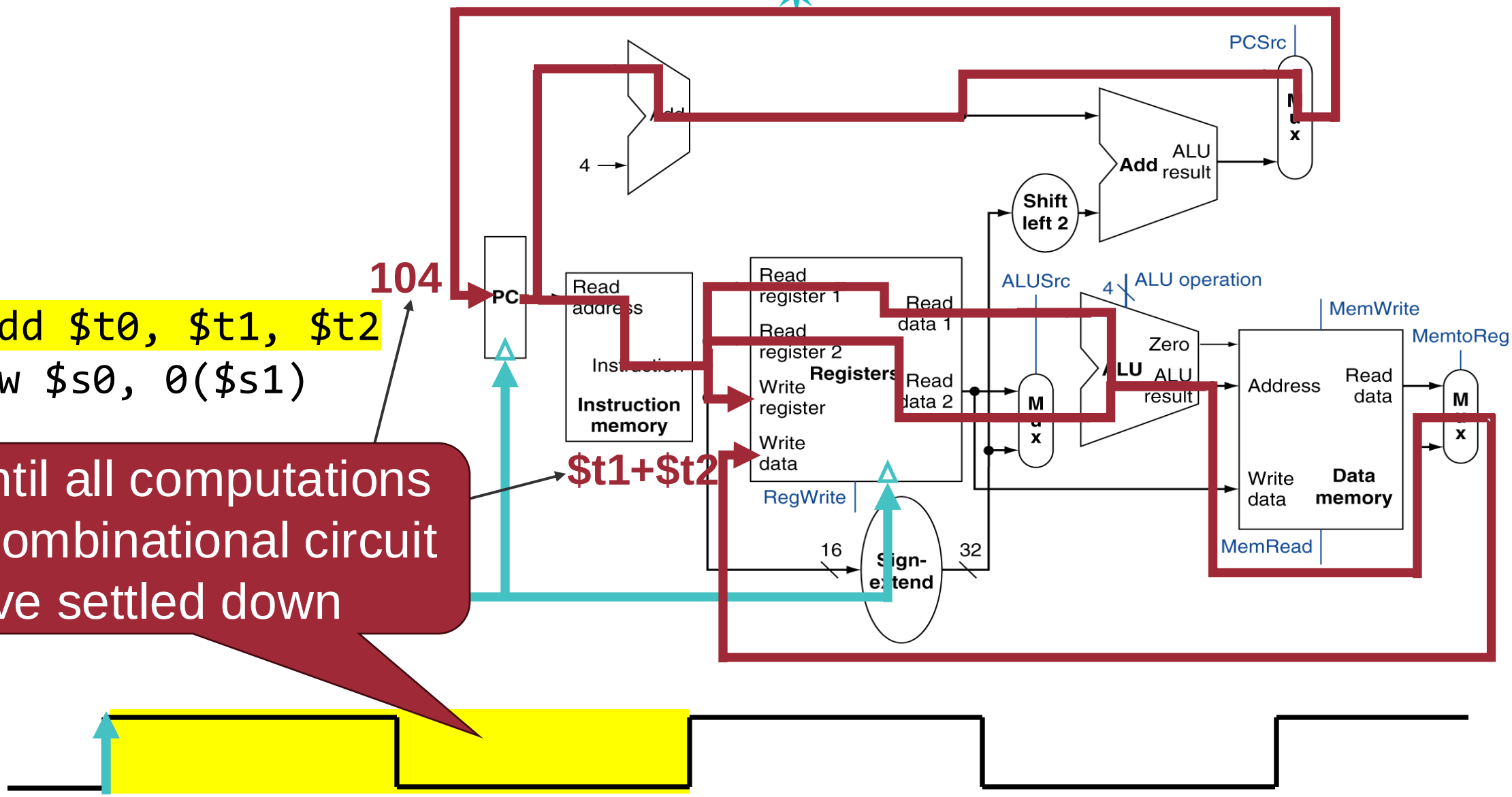
```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```

104

$t1+$t2

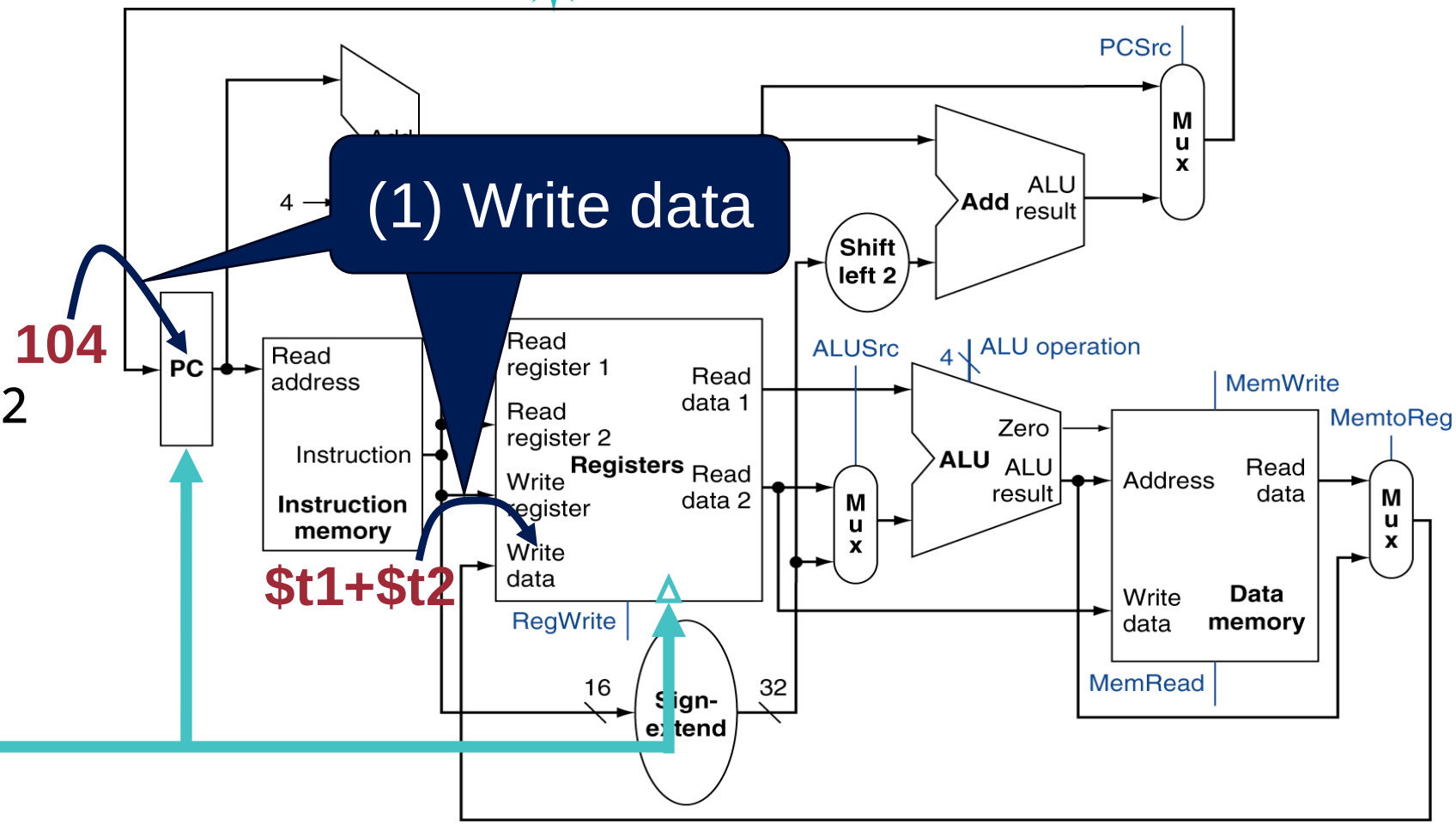
Wait until all computations in the combinational circuit have settled down

Clocks



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

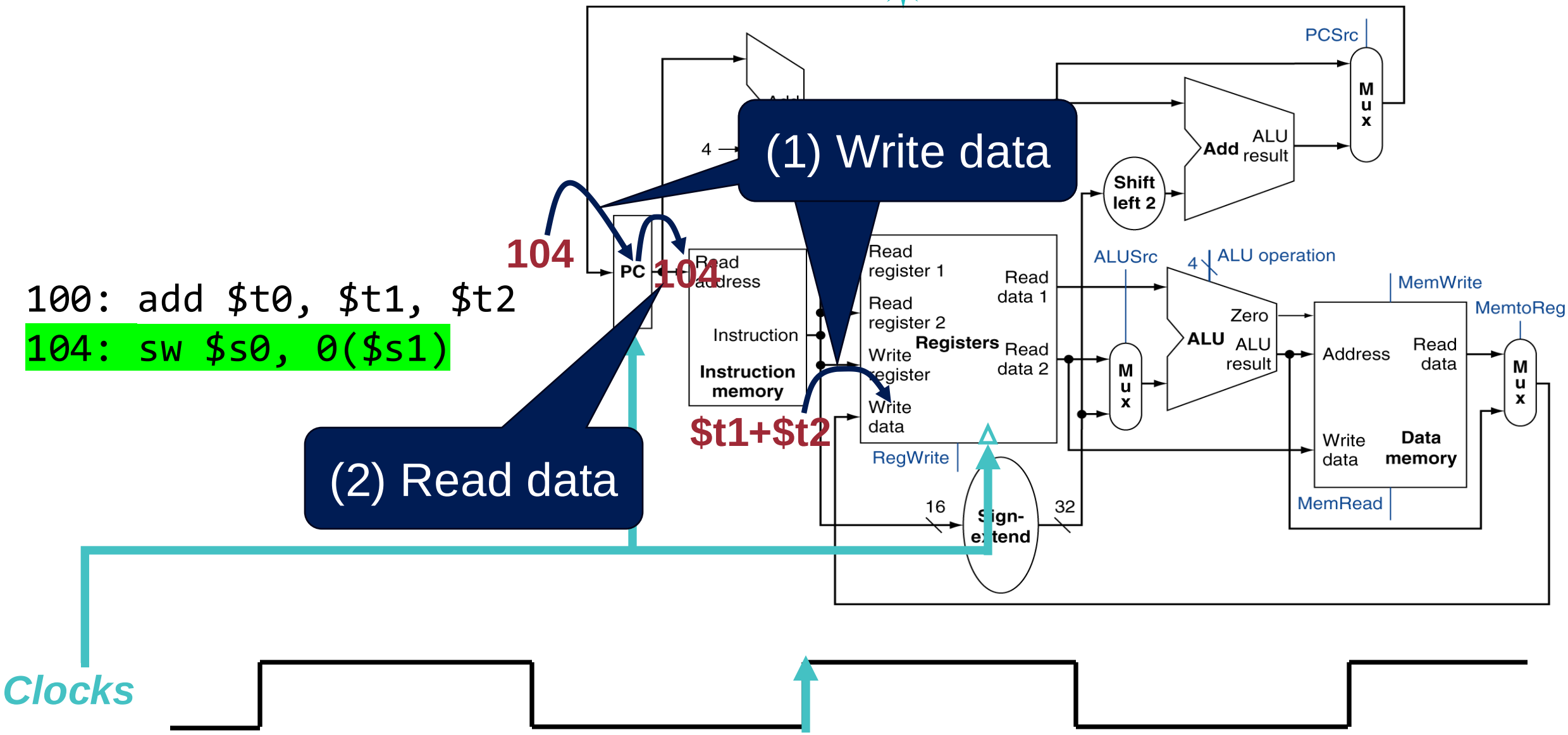
```
100: add $t0, $t1, $t2
104: sw $s0, 0($s1)
```



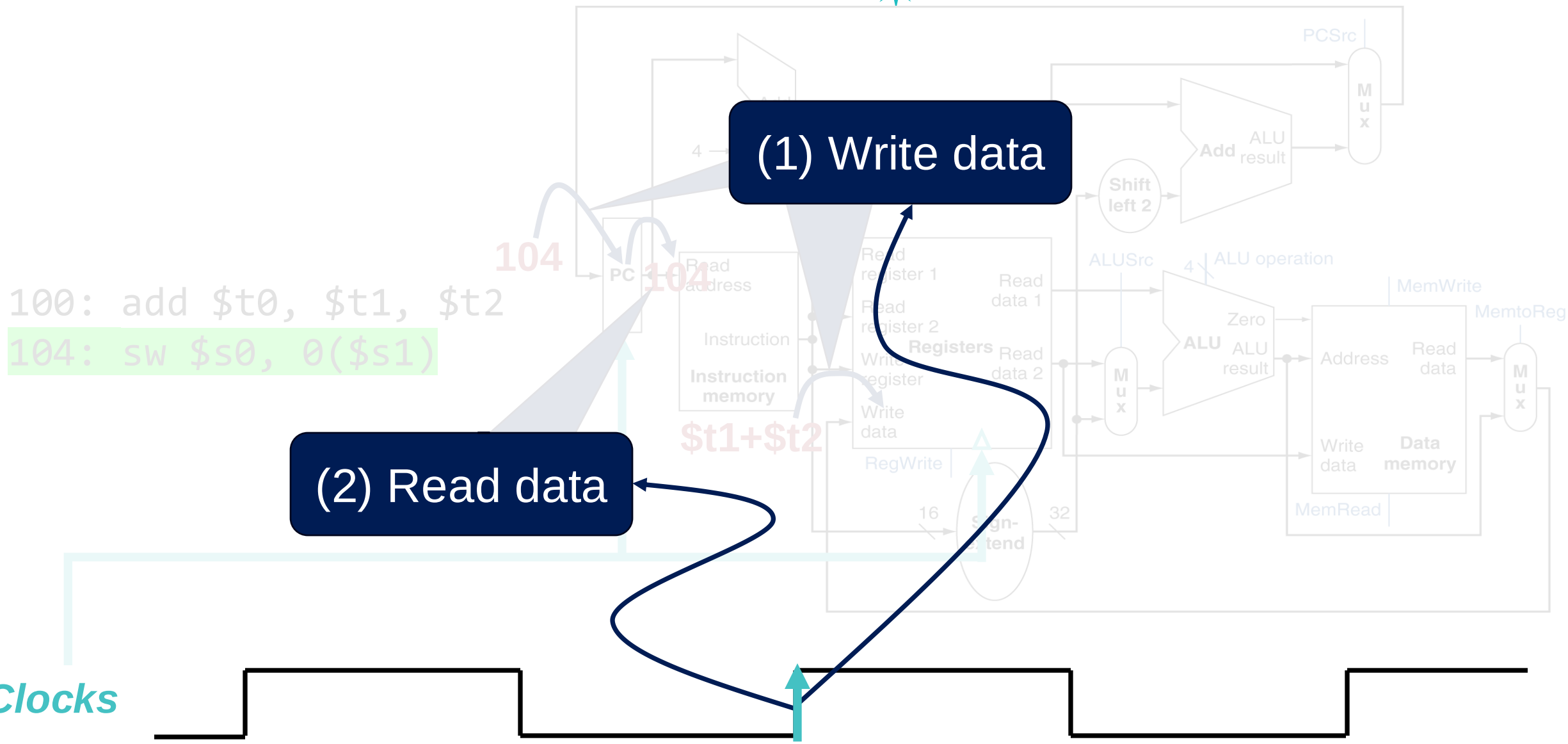
Clocks

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

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

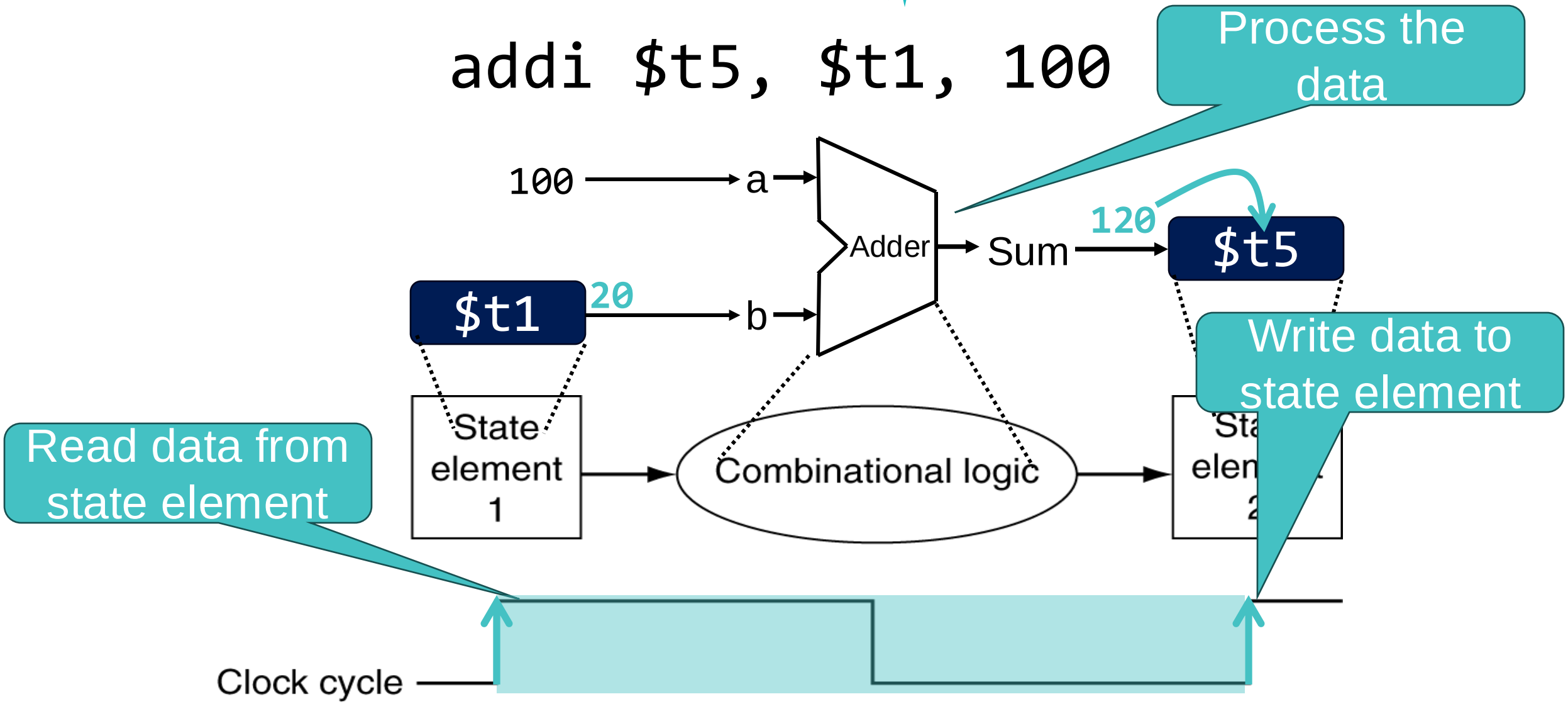


(Before Execution) `sw $s0, 0($s1)`

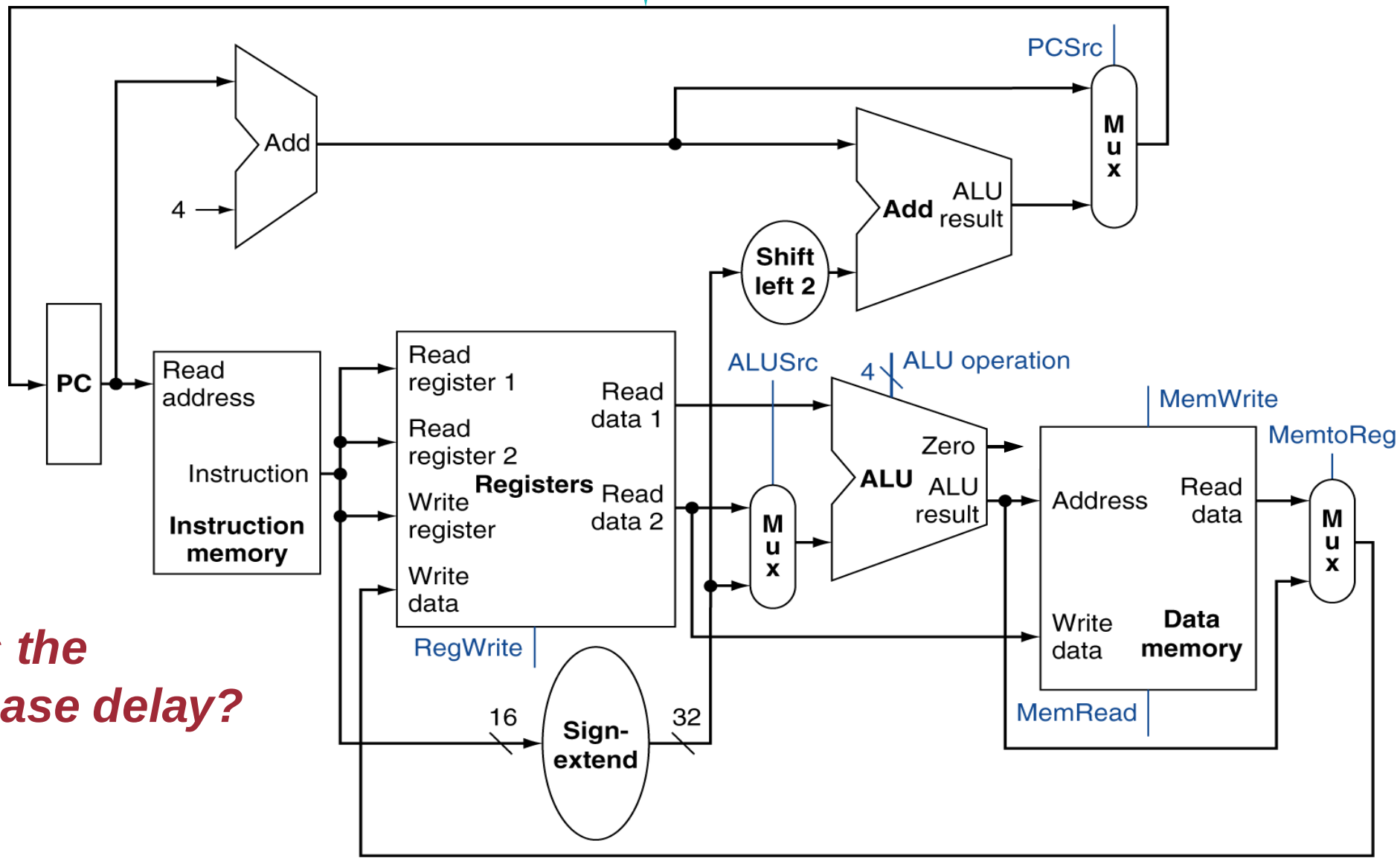


Recap: Clocking Methodology Example

```
addi $t5, $t1, 100
```



Discussion Points (1): Critical Path



What is the worst case delay?

Discussion Points (1): 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

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

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

CPI = 1

Execution Time = Instruction # * CPI * Clock Cycle Time
= Instruction # * 600ps

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

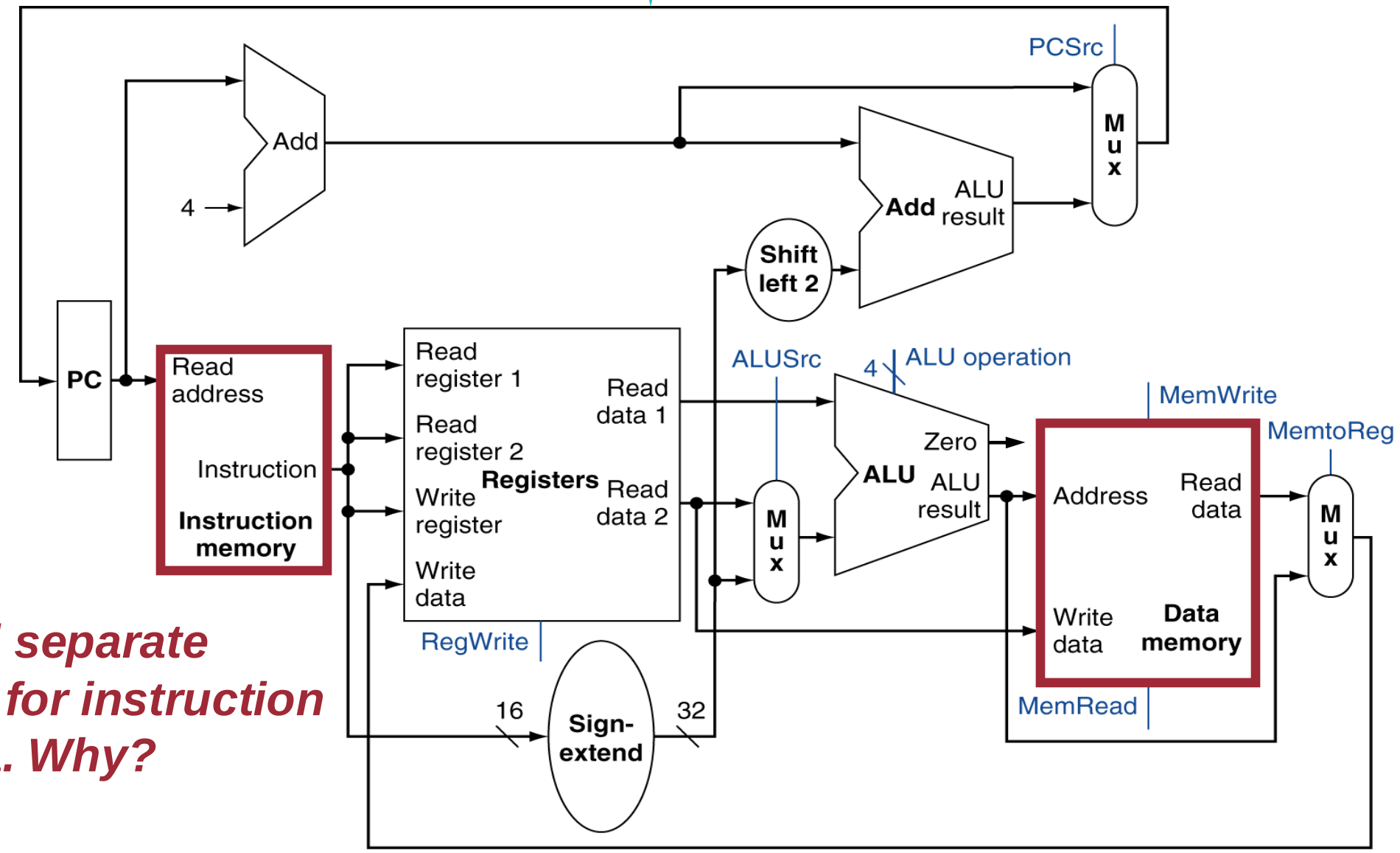
Even though it could finish early, it has to wait for 600ps

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.

Discussion Points (2): Memory Separation



We need separate memory for instruction and data. Why?

Discussion Points (2): Memory Separation

- Why do we need a memory for instructions separate from one for data?
  - No datapath resource can be ***used more than once per instruction***, so any element needed more than once must be duplicated

Question?