CSE261: Computer Architecture

9. Processor (1)

Seongil Wi

Department of Computer Science and Engineering

Where Are We?

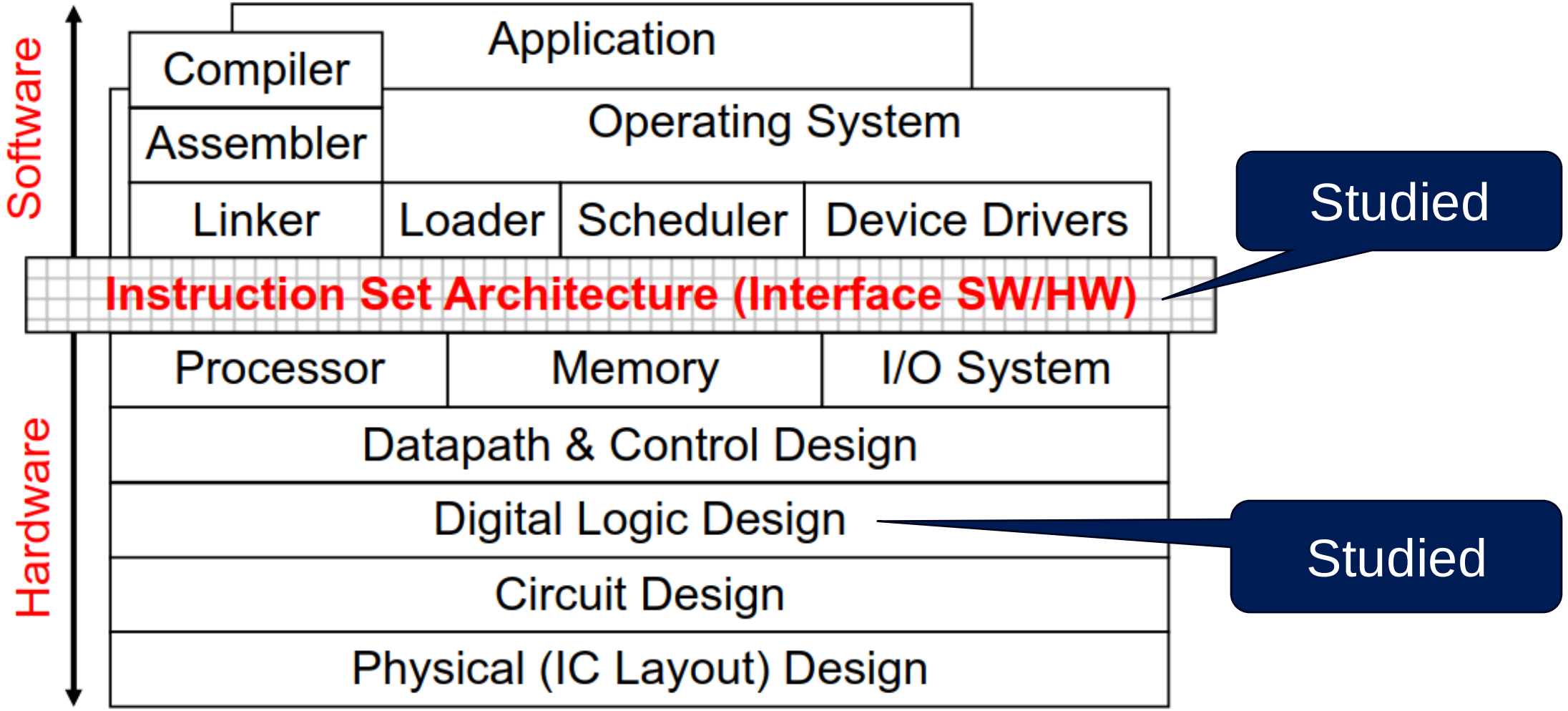


Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

We are ready to look at an implementation of the MIPS!

Today's Topic

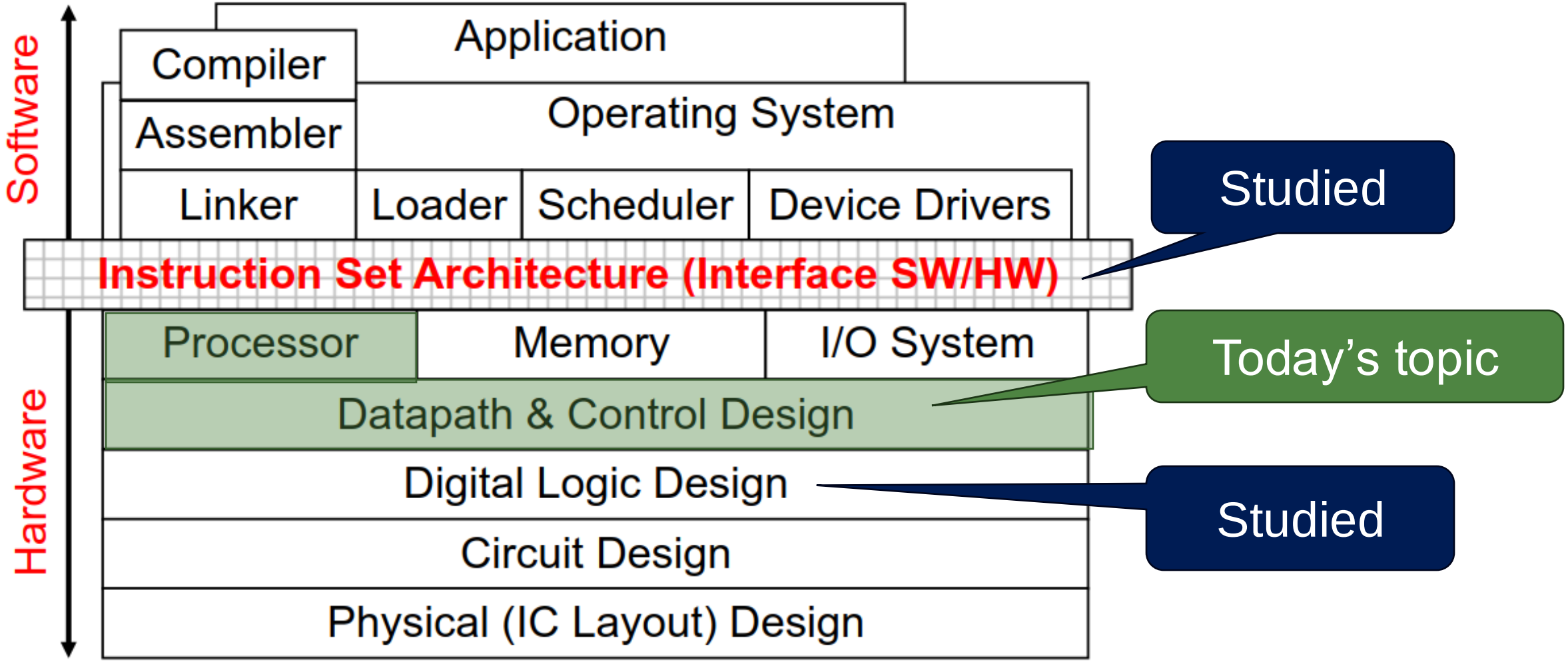
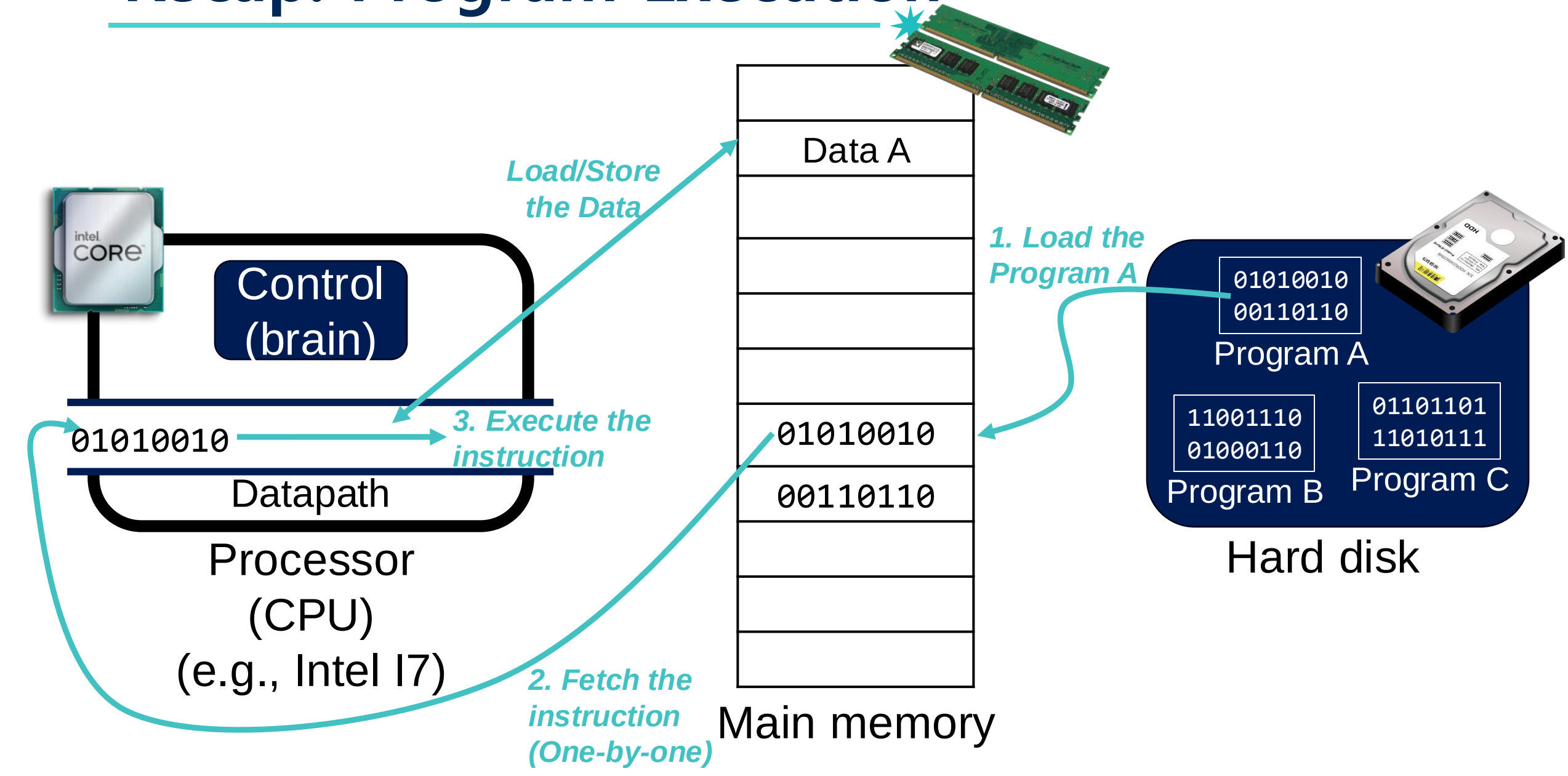
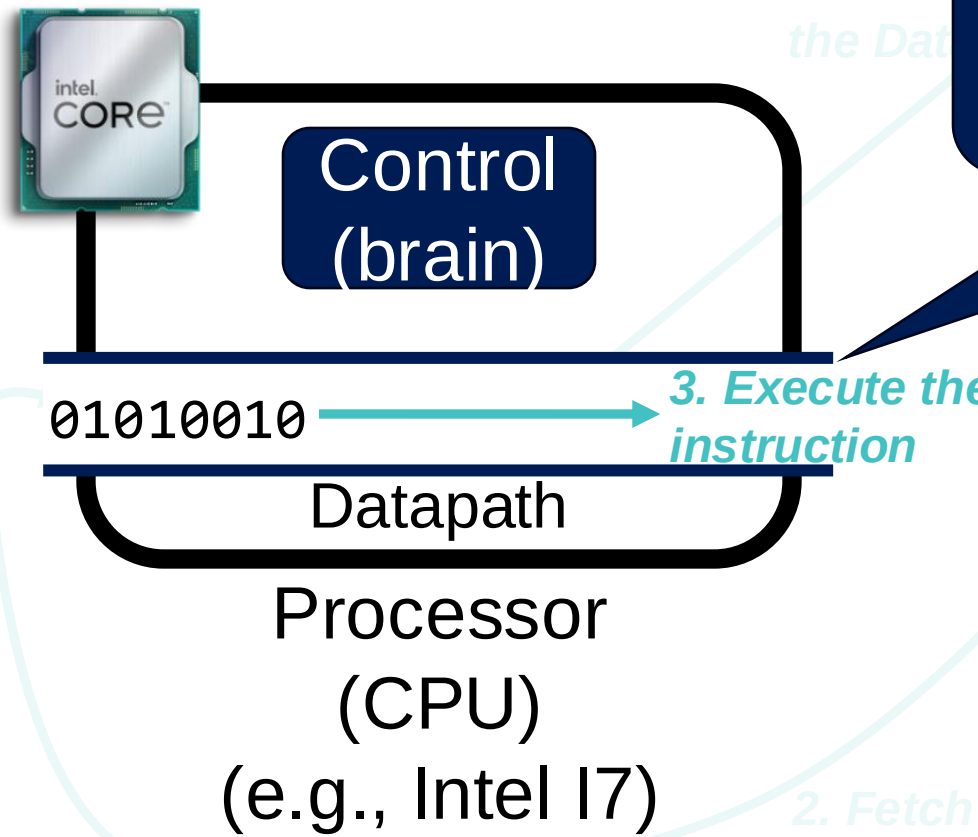


Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

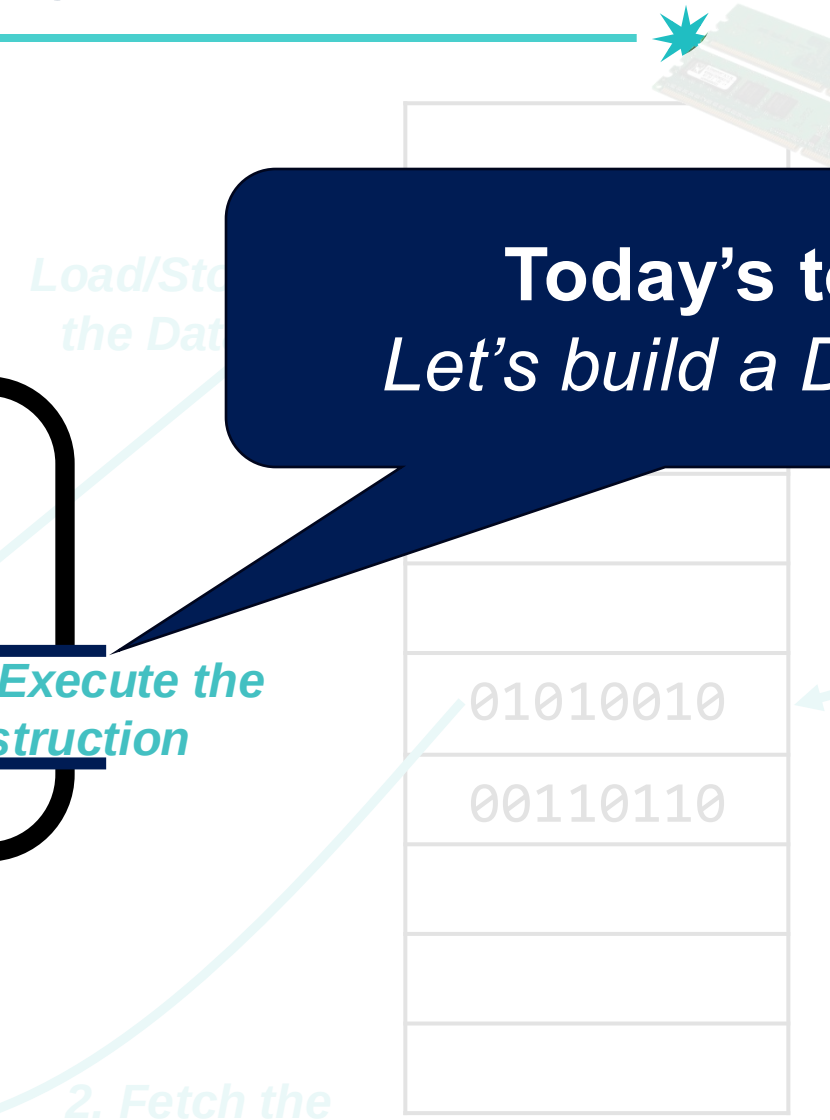
Recap: Program Execution

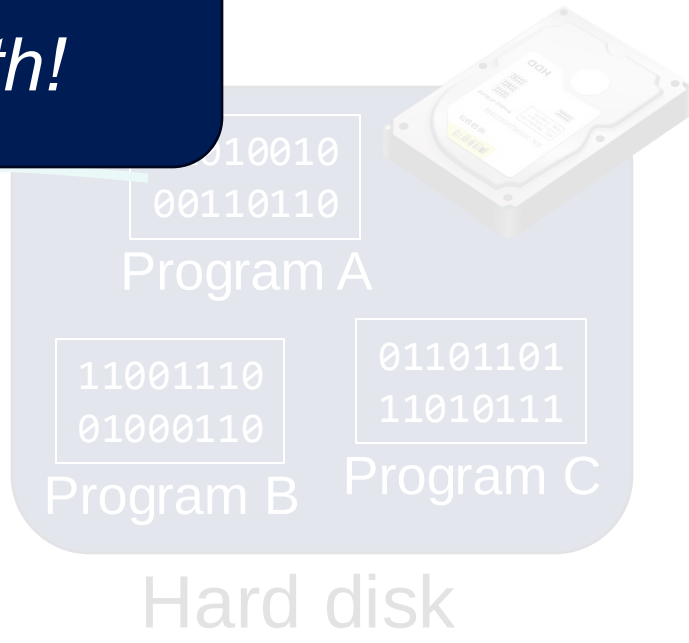


Today's Topic



Today's topic:
Let's build a Datapath!



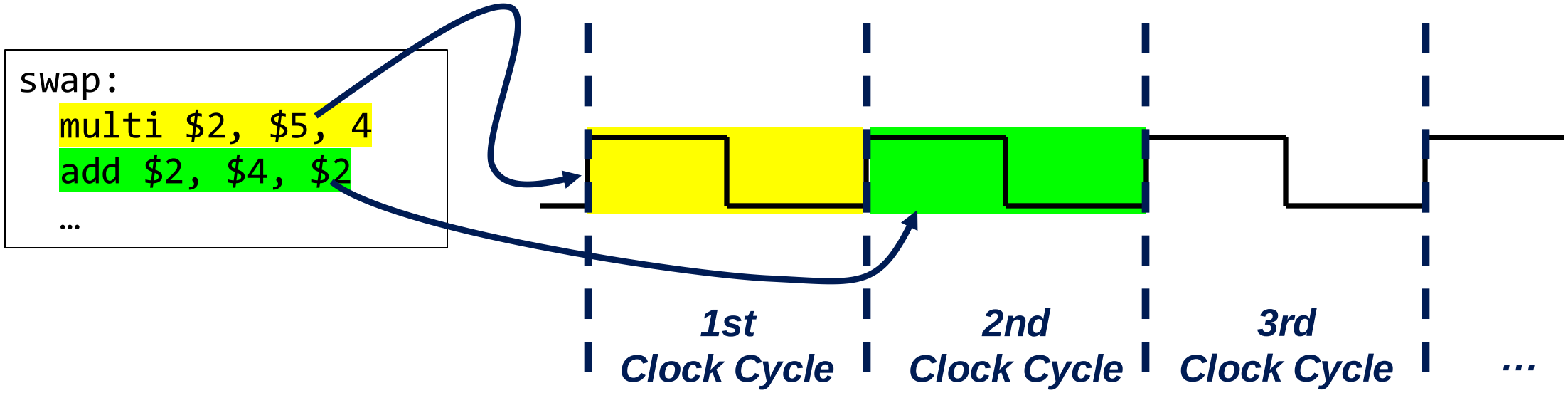


Building a Datapath

Our Assumption From Now On

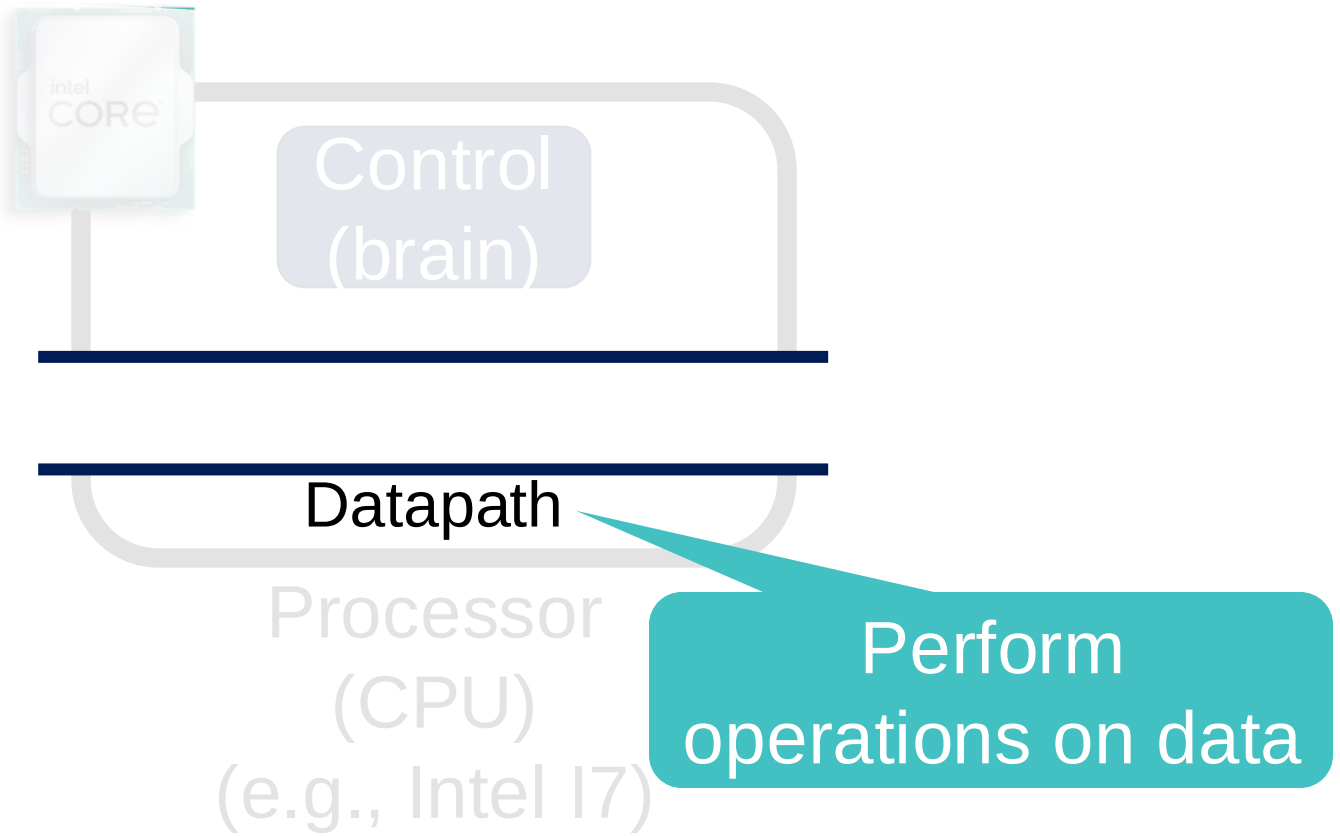
We will consider ***single clock cycle datapath***

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

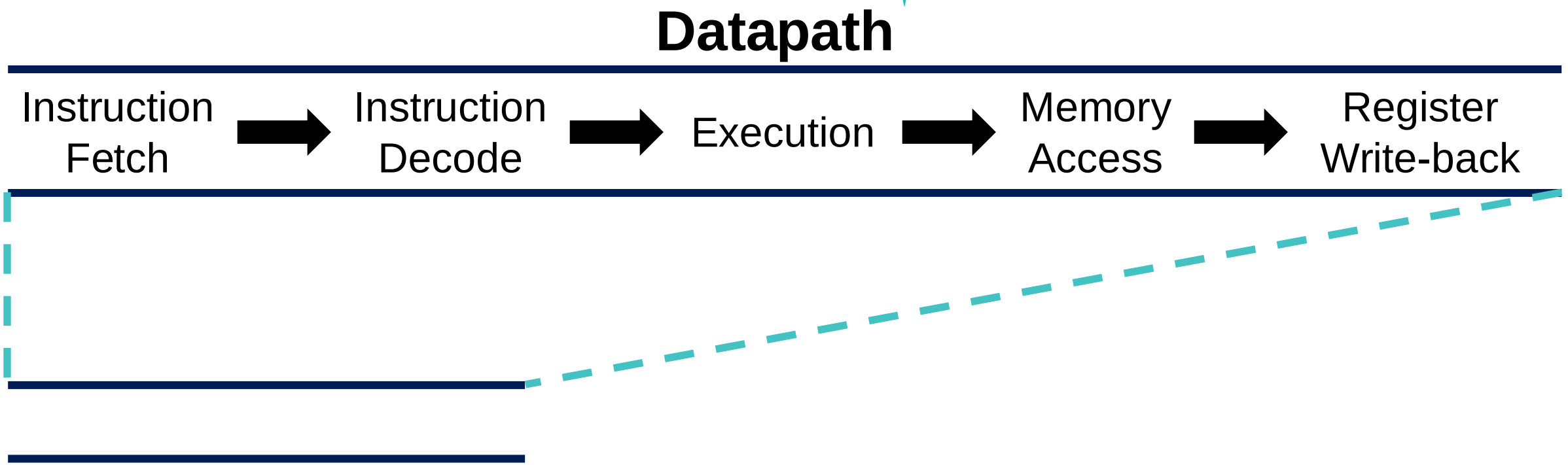


Datapath

- Elements that process data and addresses in the CPU



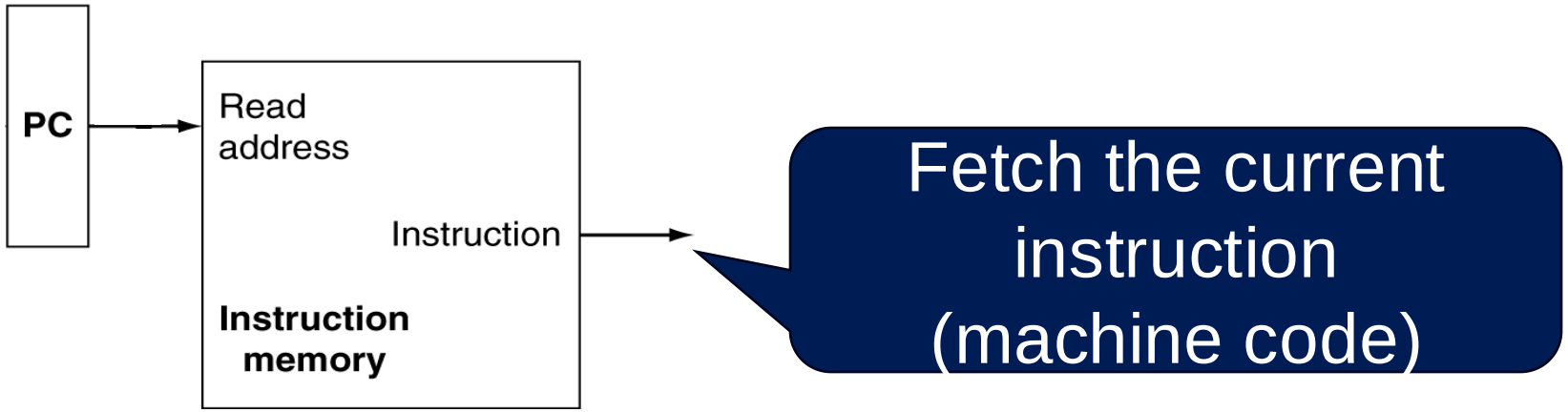
Datapath: (1) Fetch Instruction



Datapath: (1) Fetch Instruction

Datapath

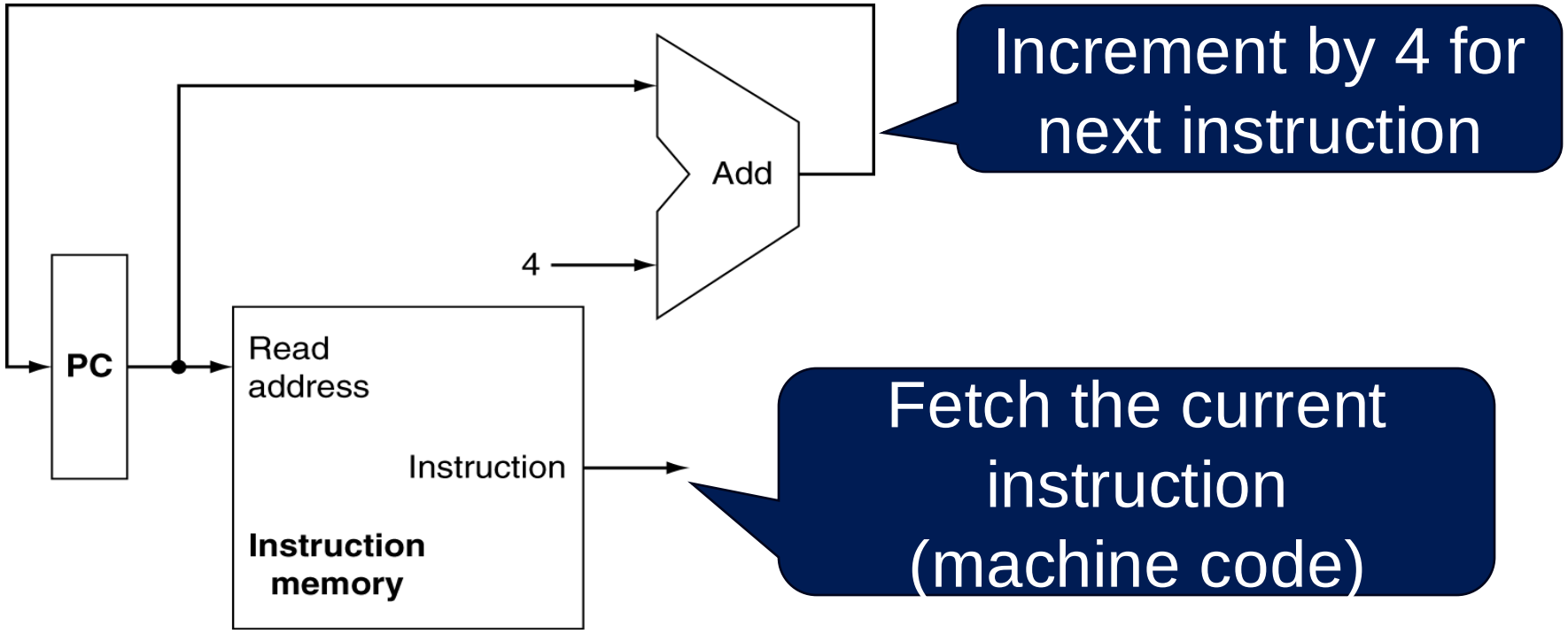
Instruction
Fetch



Datapath: (1) Fetch Instruction

Datapath

Instruction
Fetch

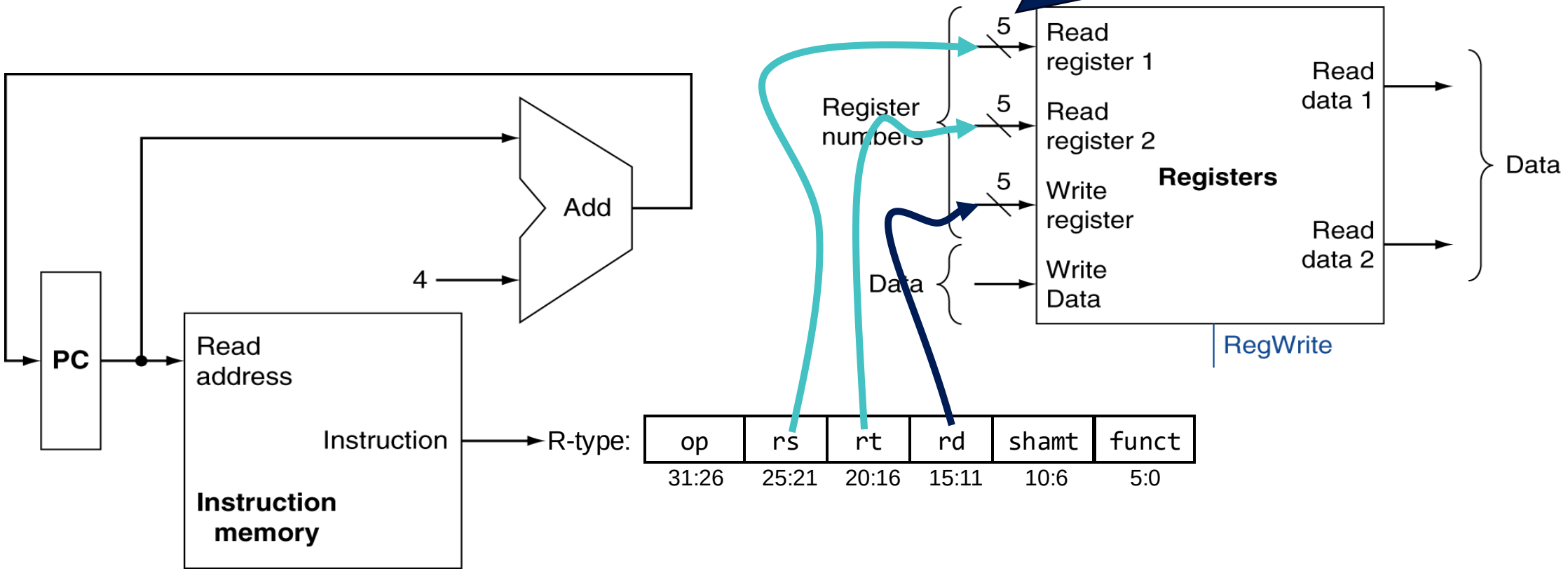


Datapath: (2) Instruction Decode

Datapath

Instruction Fetch → Instruction Decode

5 bits are enough to identify each of the 32 registers

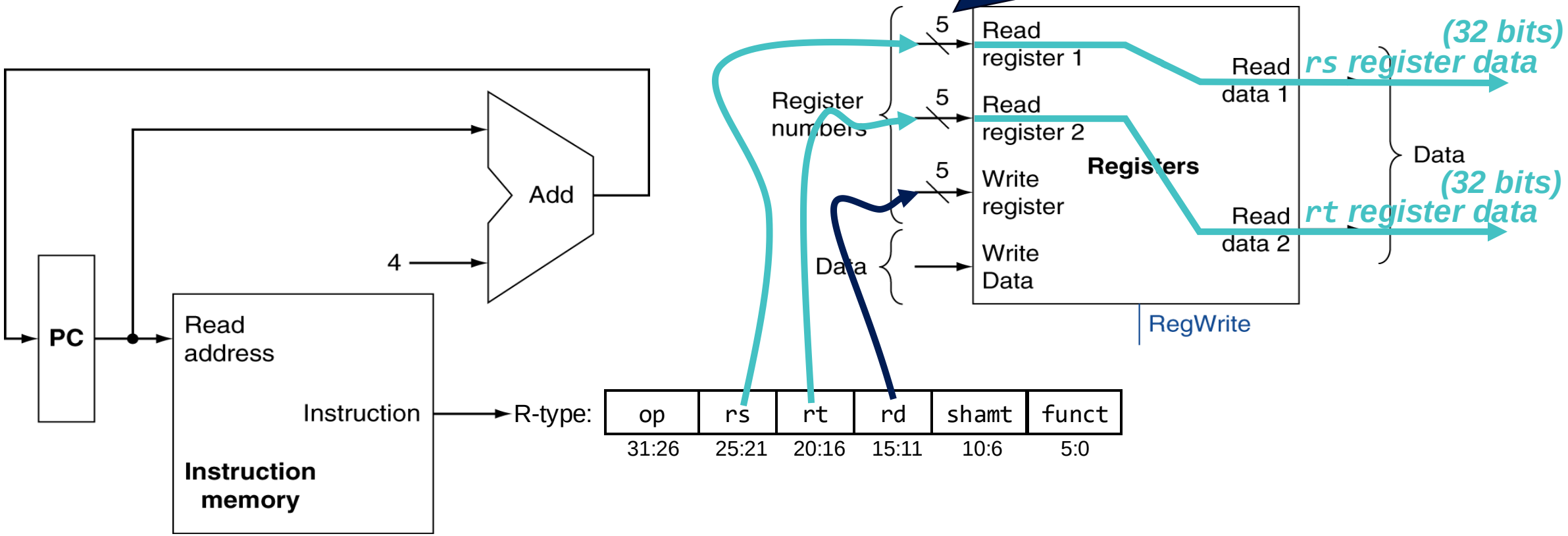


Datapath: (2) Instruction Decode

Datapath

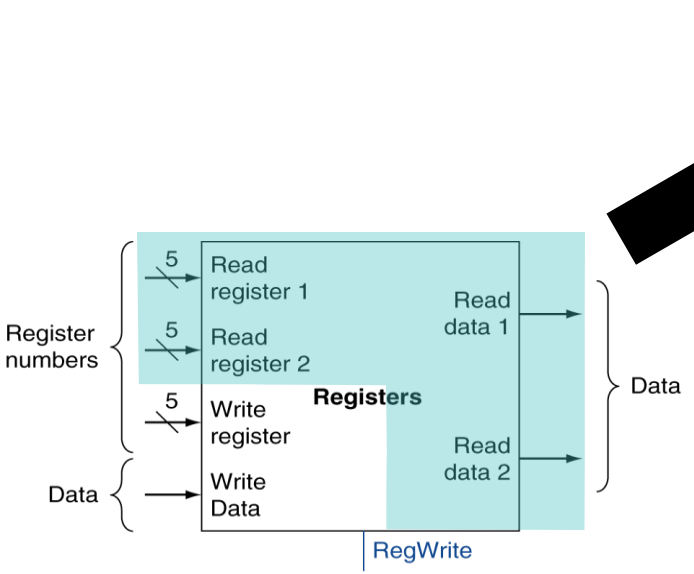
Instruction Fetch → Instruction Decode

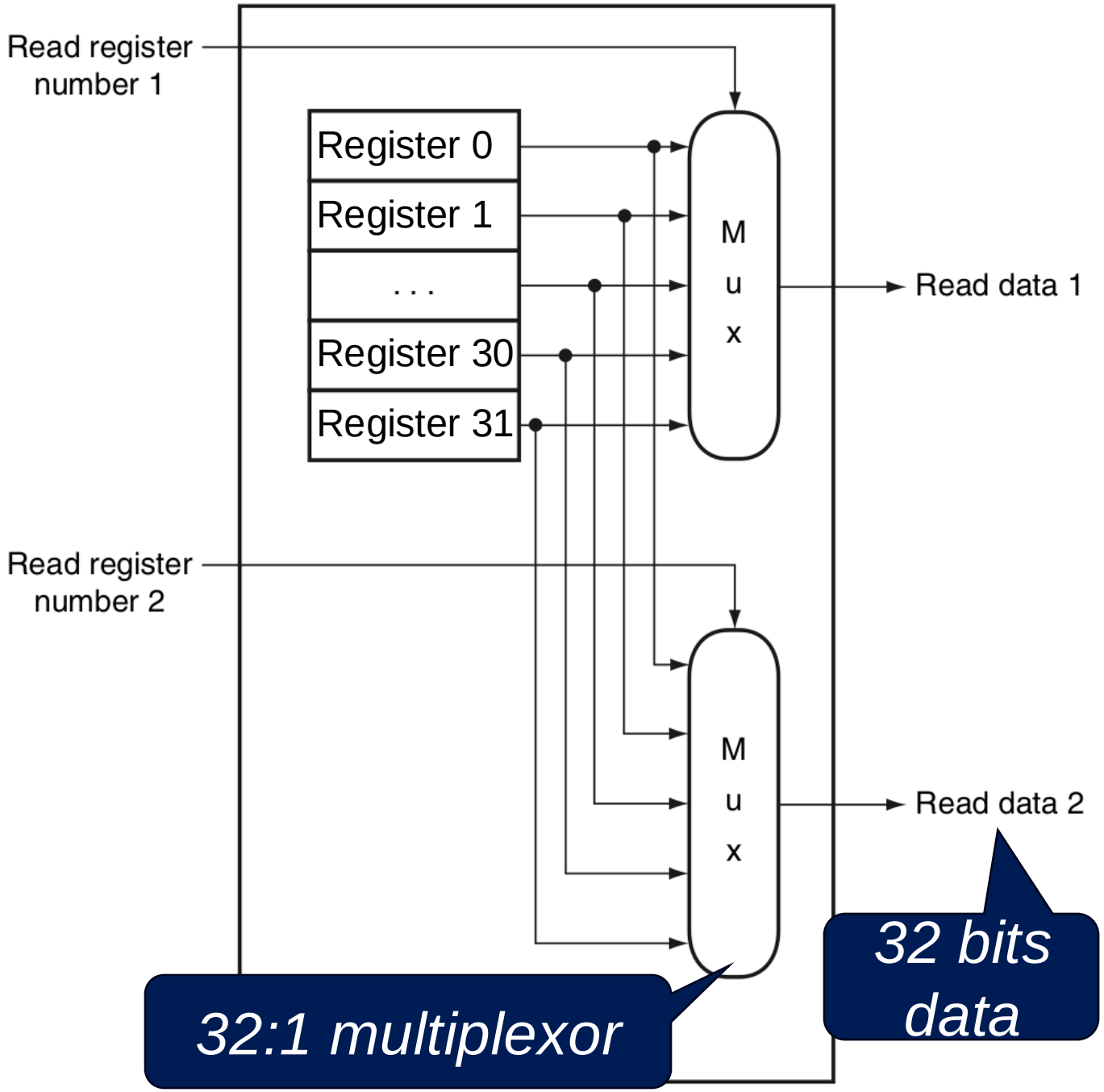
5 bits are enough to identify each of the 32 registers



Register File Read

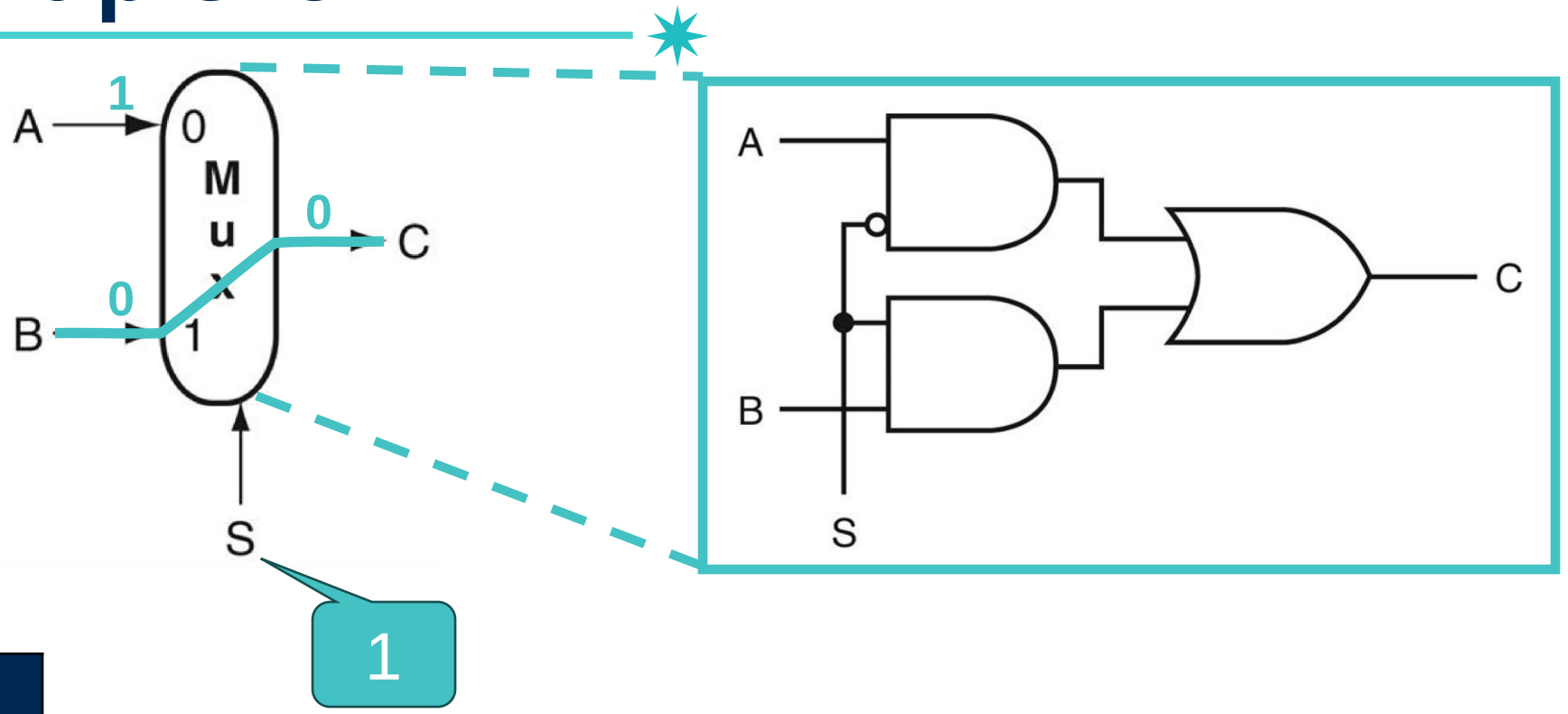
- Two register numbers select two register outputs





Recap: Multiplexor

Multiplexor
(a.k.a., MUX)

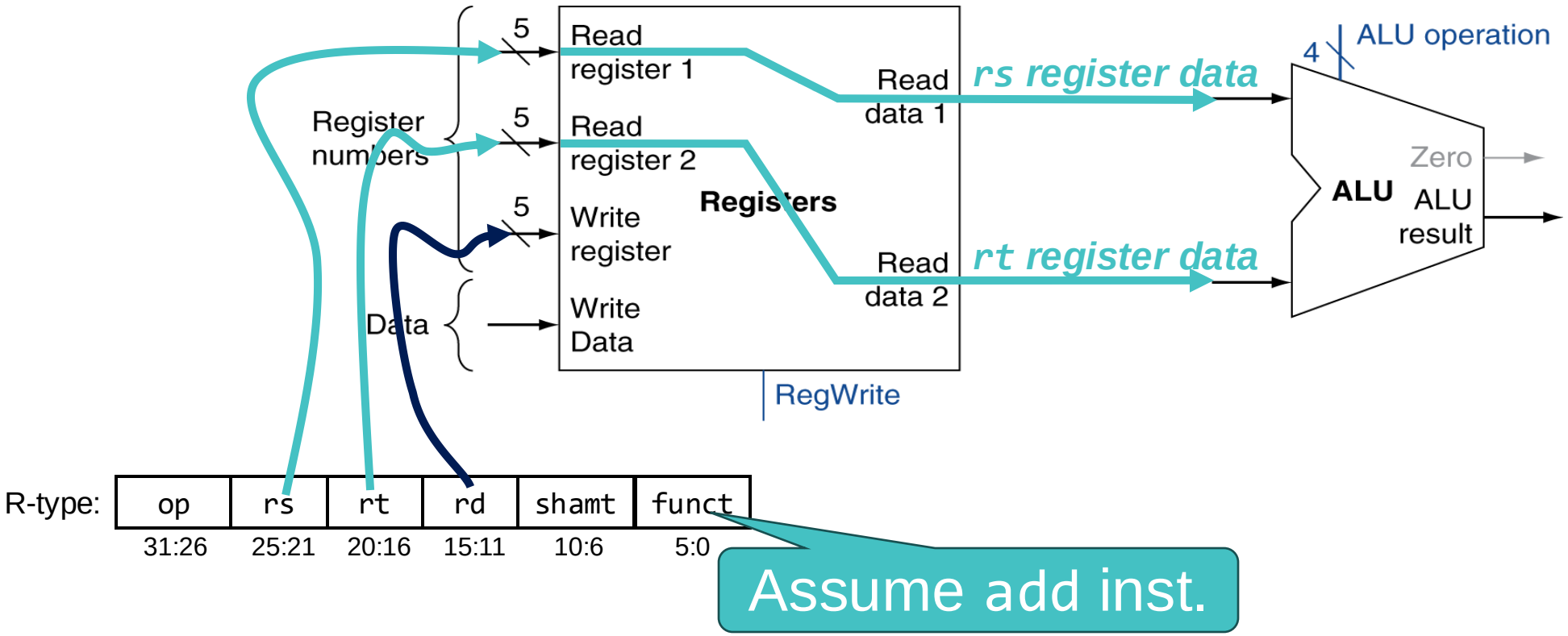


Input S	Output
0	A's Input
1	B's Input

Datapath: (3) Execution

Datapath

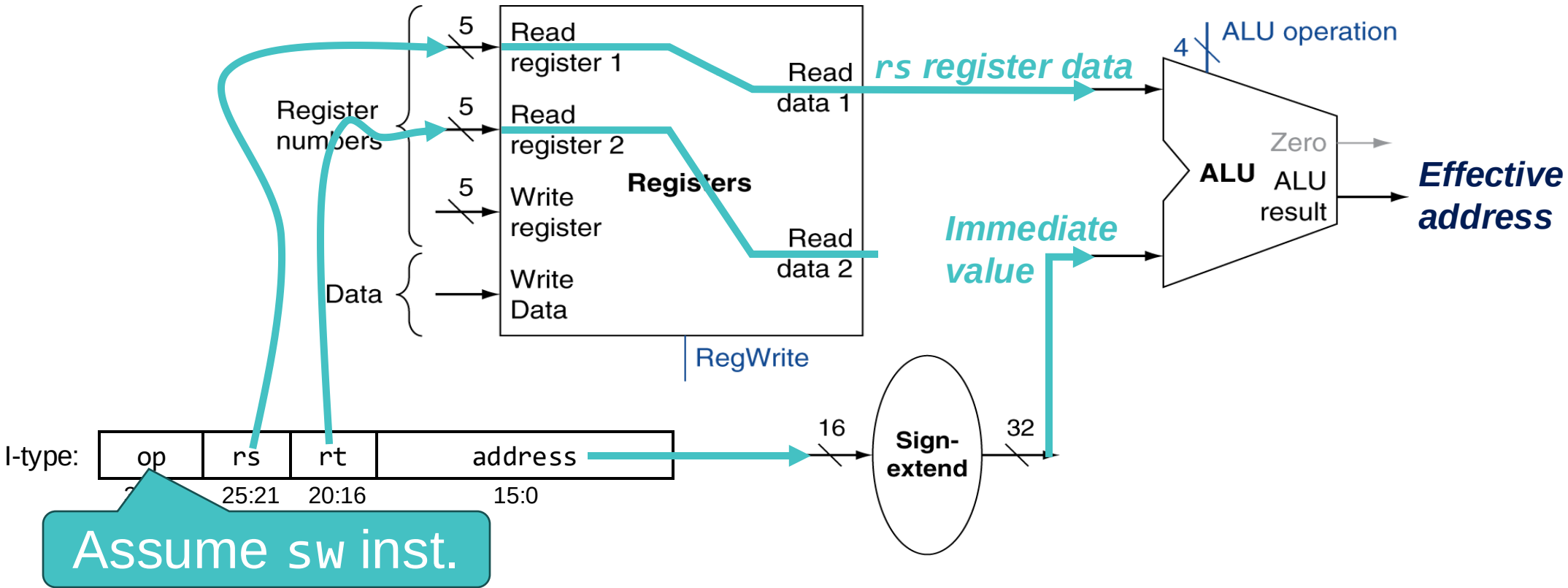
Instruction Fetch → Instruction Decode → Execution



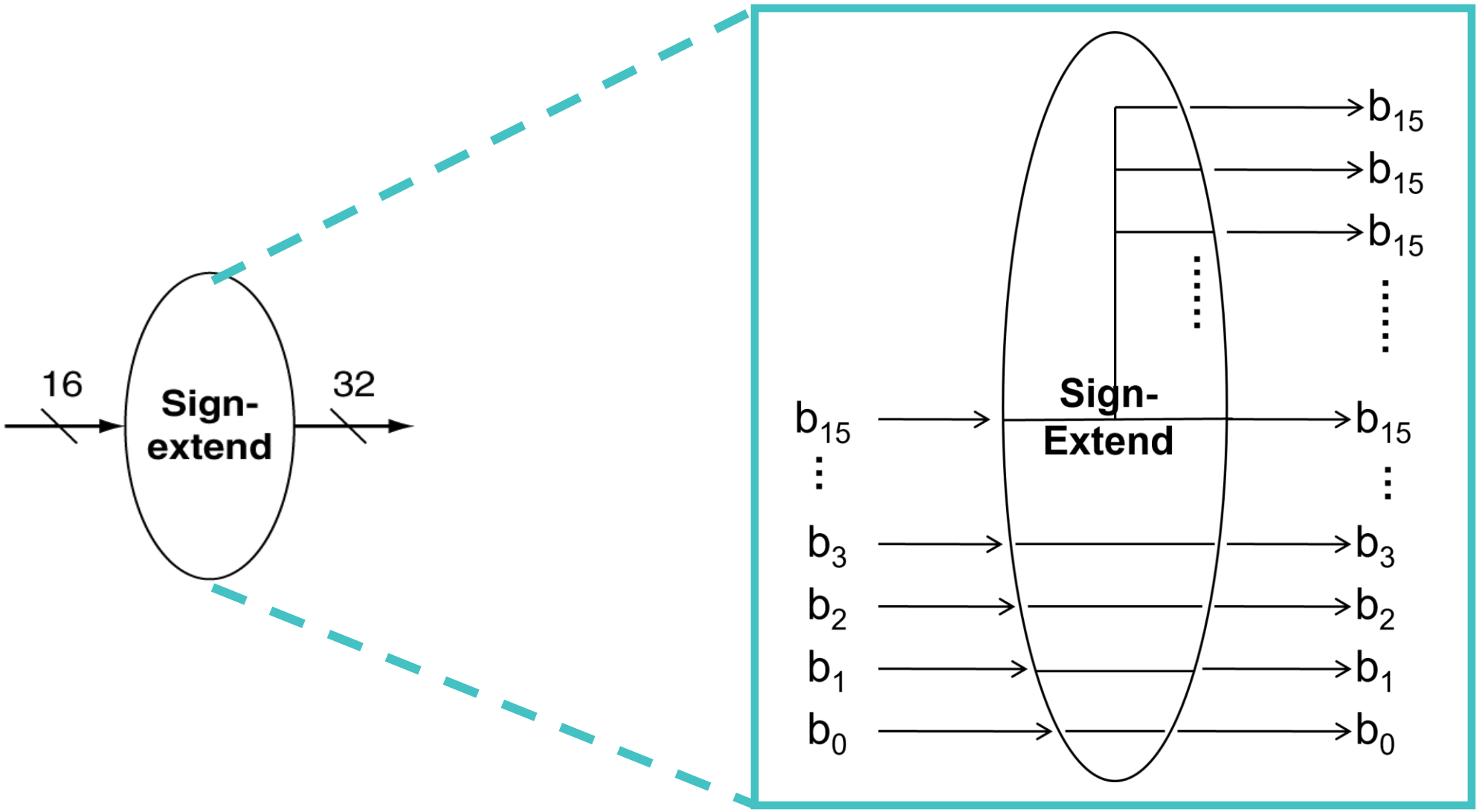
Datapath: (3) Execution

Datapath

Instruction Fetch → Instruction Decode → Execution



FYI: Sign-Extend

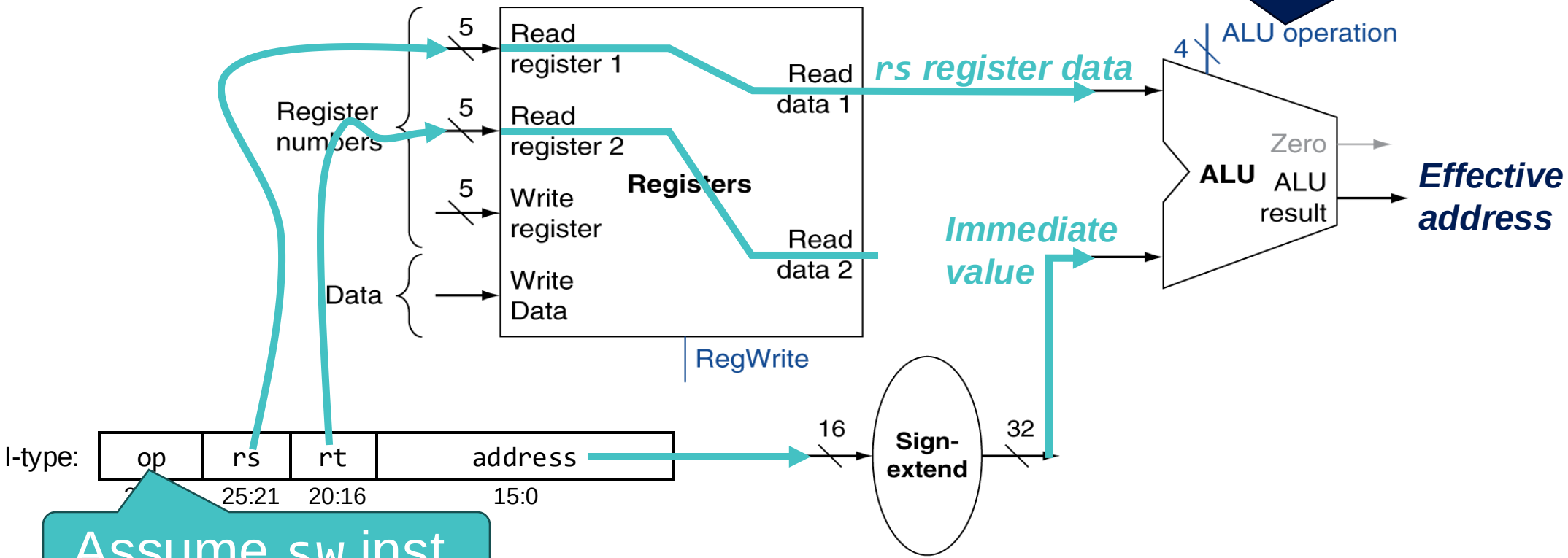


Datapath: (3) Execution

Datapath

Instruction Fetch → Instruction Decode → Execution

Signal from control unit: and, or, add, subtract, set less than, nor



Assume `sw` inst.

Arithmetic Logic Unit (ALU)

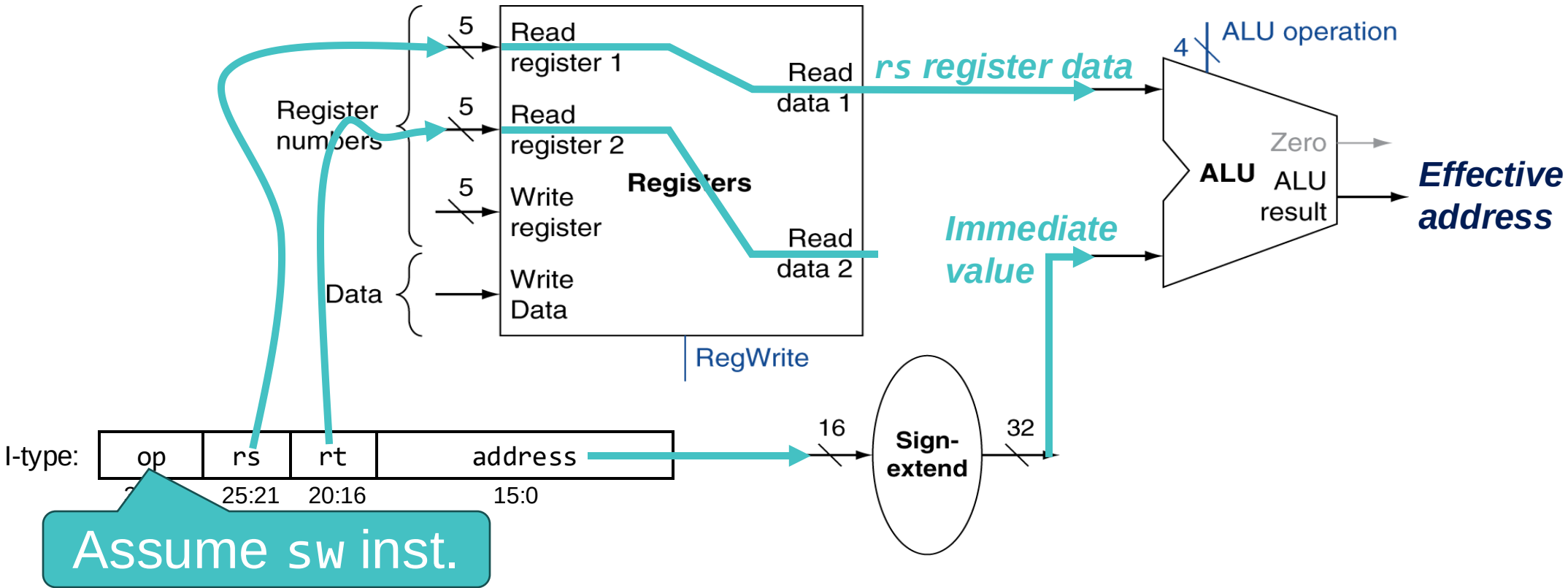
Performs arithmetic and logic operations on binary numbers

we will cover about ALU later ☺

Datapath: (3) Execution

Datapath

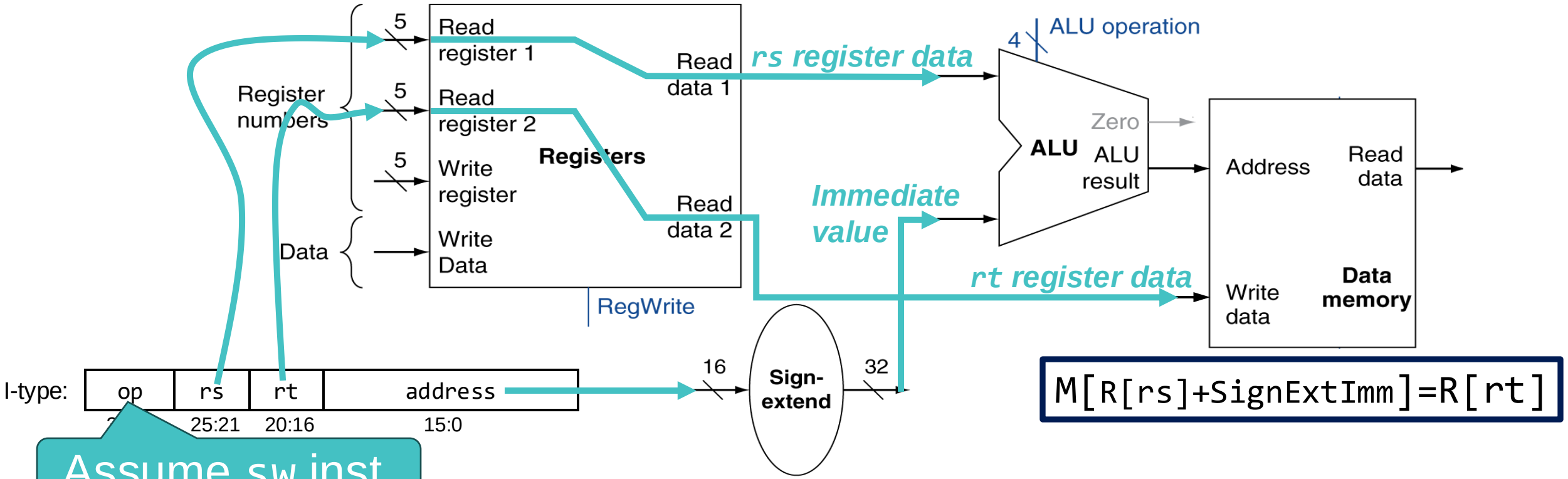
Instruction Fetch → Instruction Decode → Execution



Datapath: (4) Memory Access

Datapath

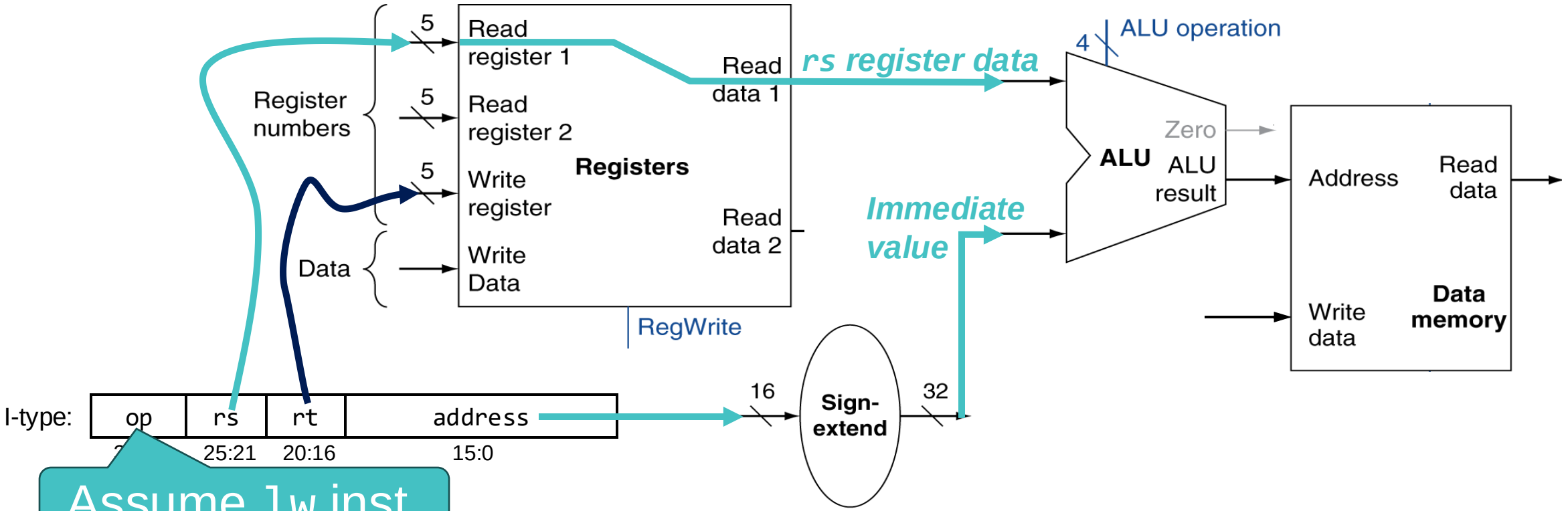
Instruction Fetch → Instruction Decode → Execution → Memory Access



Datapath: (4) Memory Access

Datapath

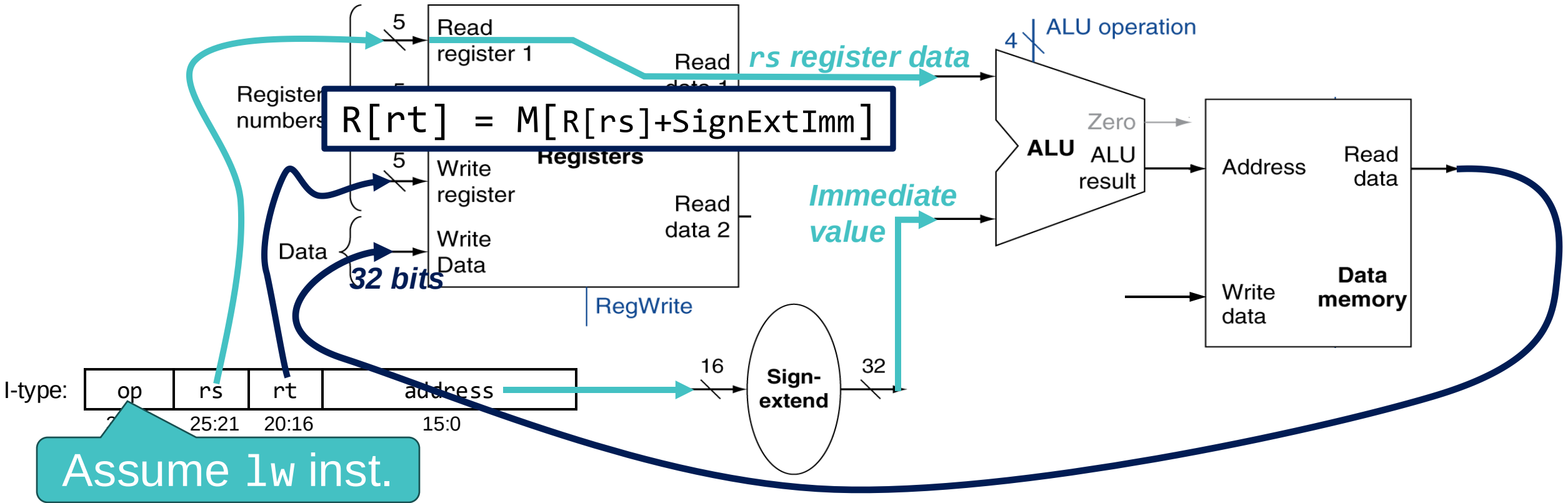
Instruction Fetch → Instruction Decode → Execution → Memory Access



Datapath: (5) Register Write-back

Datapath

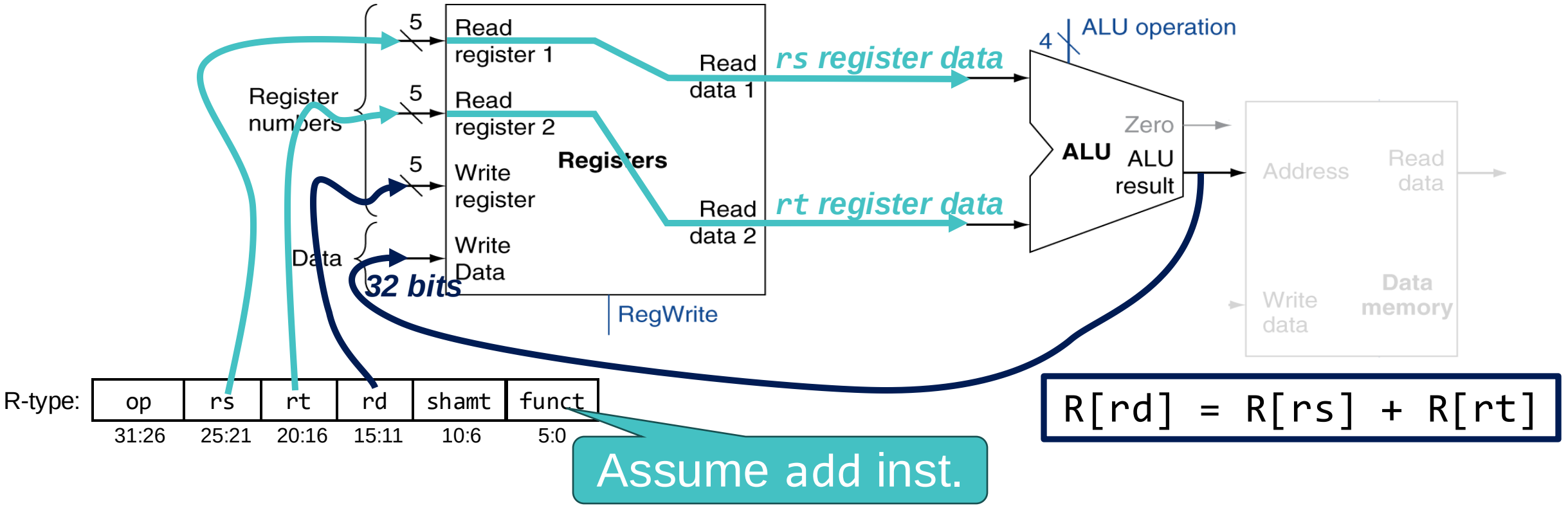
Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back



Datapath: (5) Register Write-back

Datapath

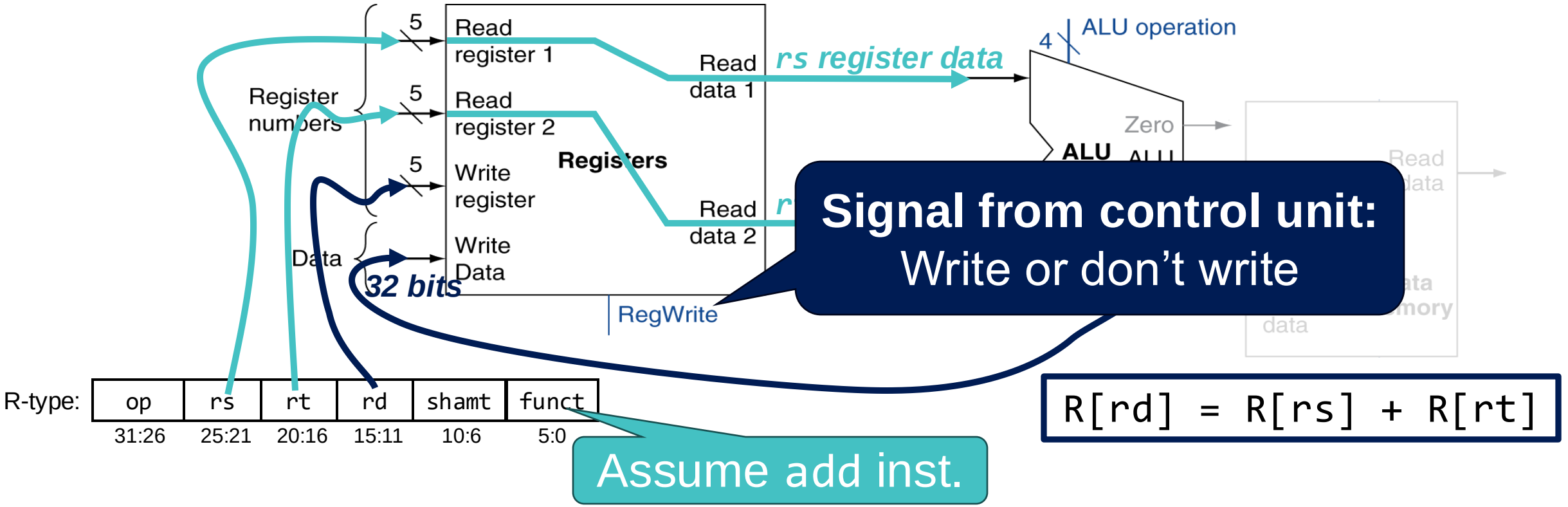
Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back



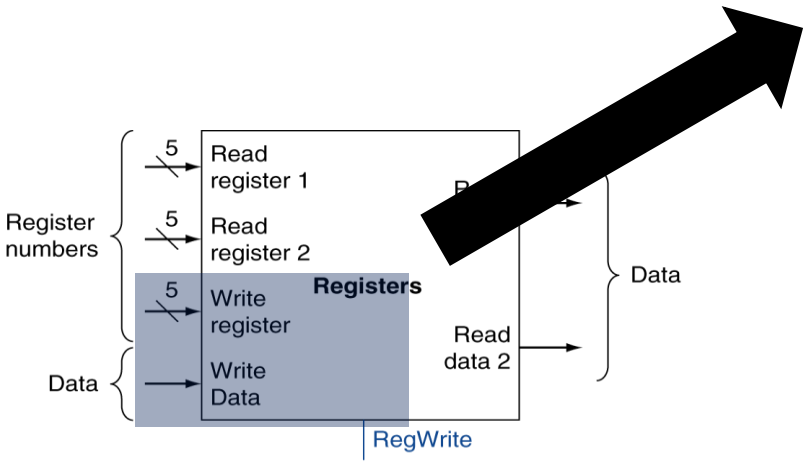
Datapath: (5) Register Write-back

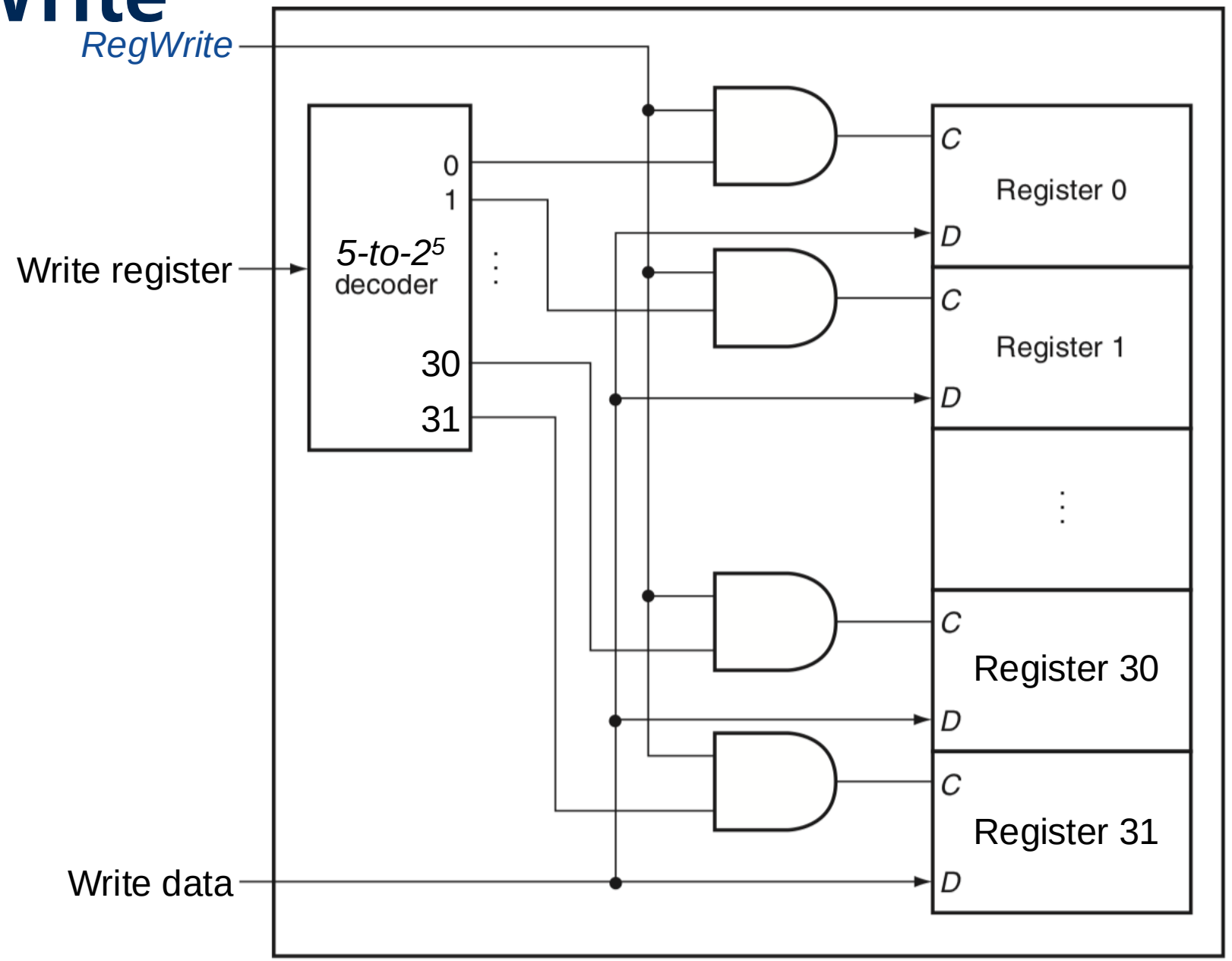
Datapath

Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back

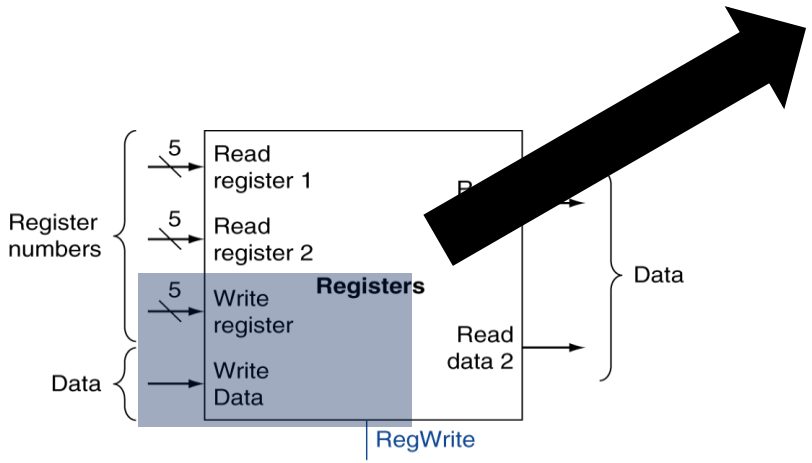


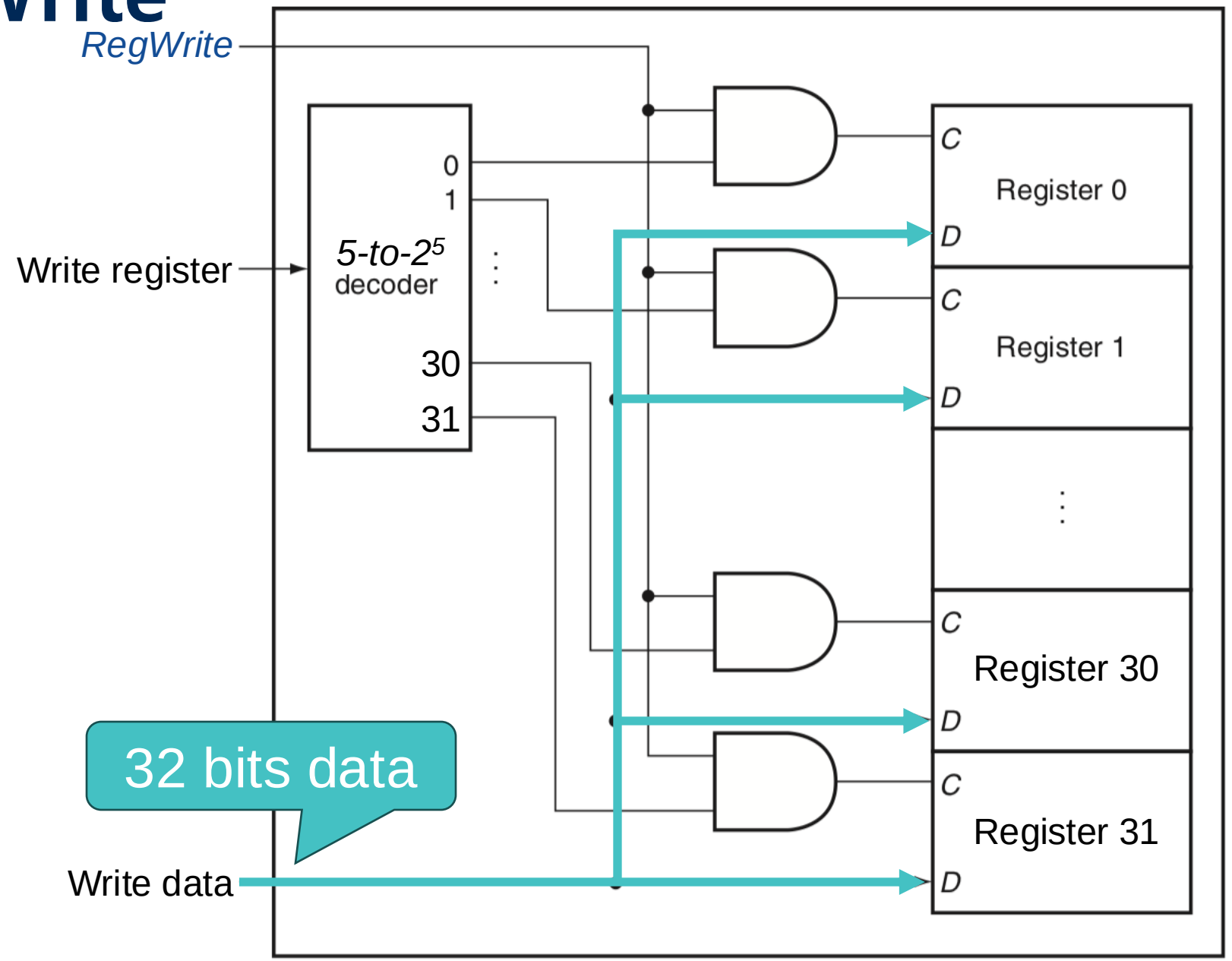
Register File Write



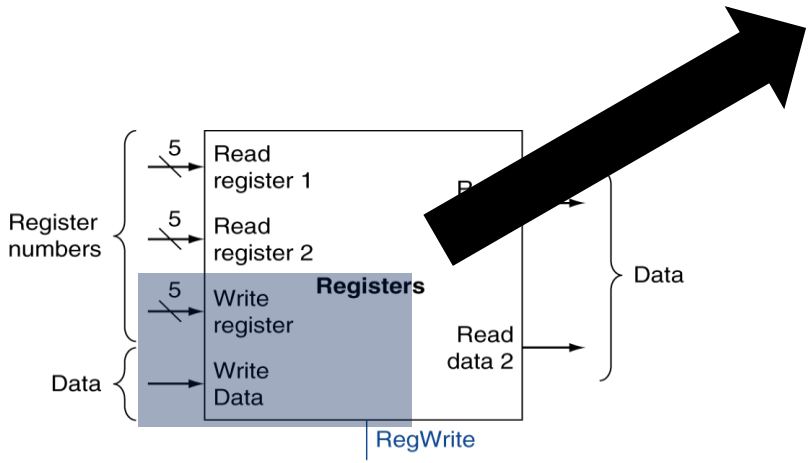


Register File Write





Register File Write



RegWrite

00001
Write register

Register number
(5 bits)

5-to-2^5 decoder

0
1
30
31

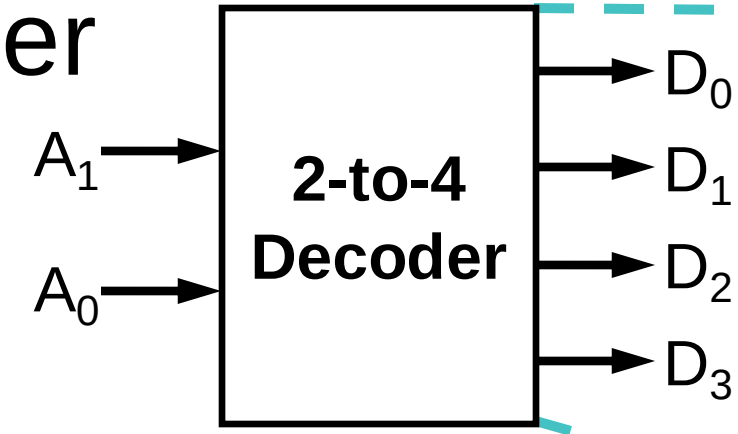
0
1
0
0

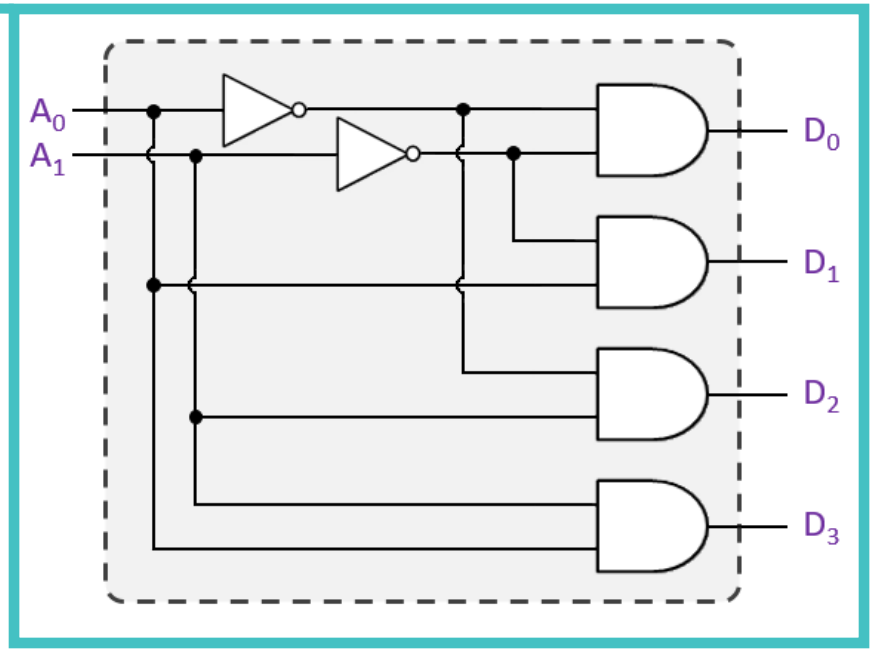
C
D
Register 0
Register 1
Register 30
Register 31

Write data

Recap: Decoder

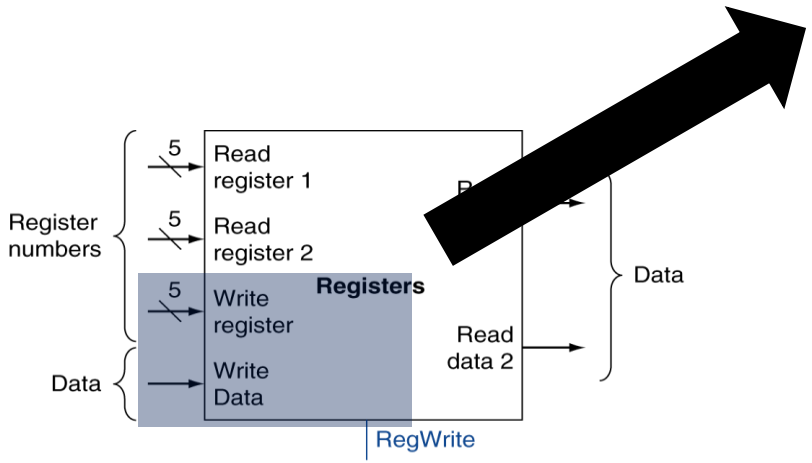
Decoder





Input		Output			
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	**1**
0	1	0	0	**1**	0
1	0	0	**1**	0	0
1	1	**1**	0	0	0

Register File Write



RegWrite

00001
Write register

Register number (5 bits)

5-to-2^5 decoder

0
1
30
31

0
1
0
0

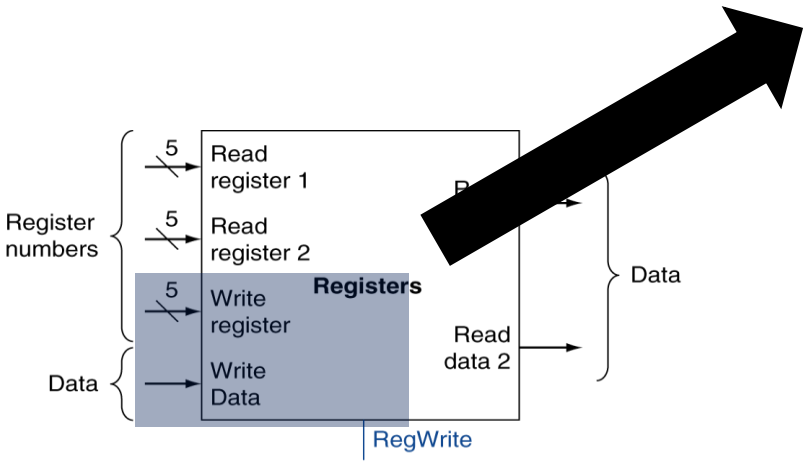
C
D
Register 0
Register 1
Register 30
Register 31

Write data

Register File Write

Signal from control unit:
- 1: write
- 2: don't write

Register number (5 bits)



RegWrite 1

00001 Write register

5-to-2^5 decoder

0
1
30
31

Write data

Register 0
Register 1
Register 30
Register 31

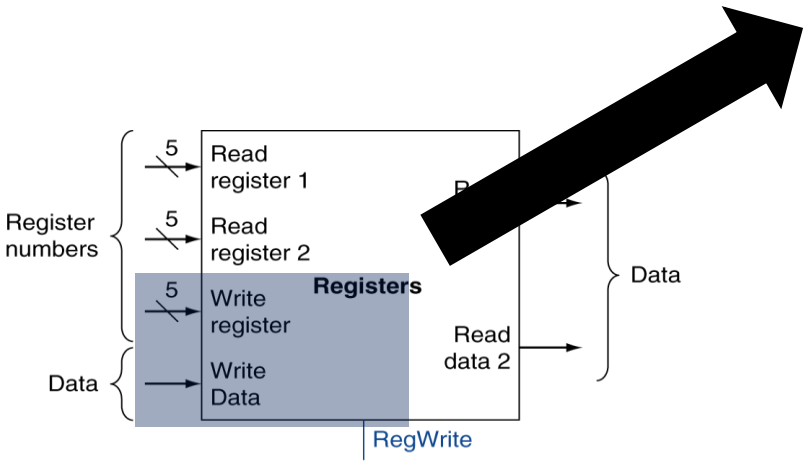
C
D

Register File Write

Signal from control unit:
- 1: write
- 2: don't write

Register number (5 bits)

32 bits data



RegWrite 1

00001 Write register

5-to-2^5 decoder

0
1
30
31

C
Register 0
D

1

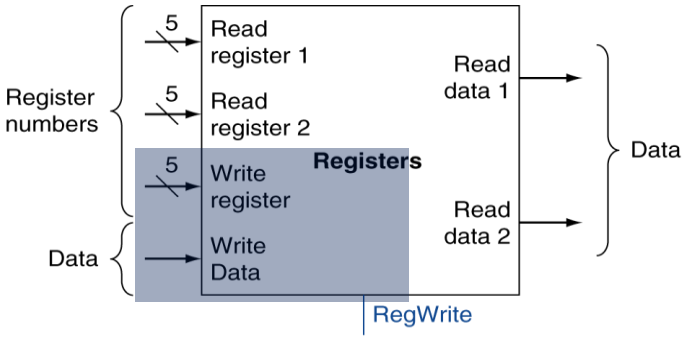
C
Register 1
D

C
Register 30
D

C
Register 31
D

Write data

Use the CPU Clock Signal to Write Data!



RegWrite

Write register

5-to-2^5 decoder

0
1
30
31

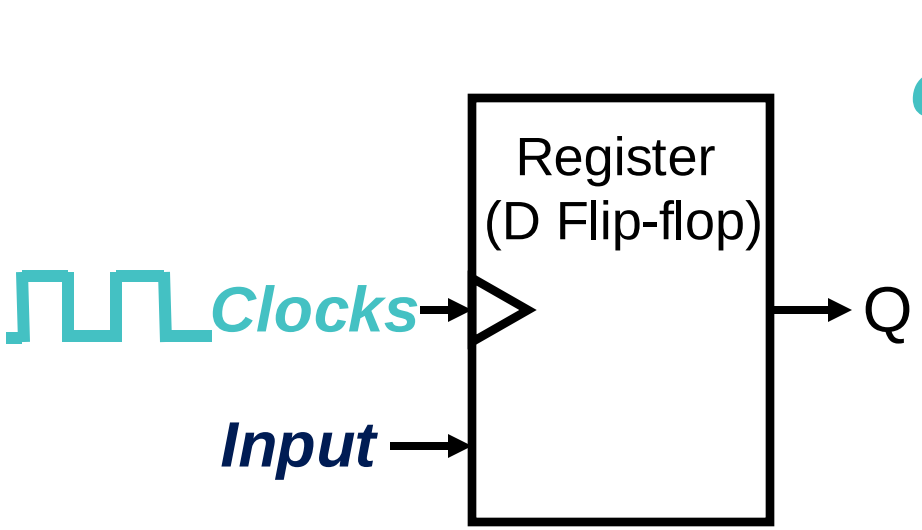
Clocks

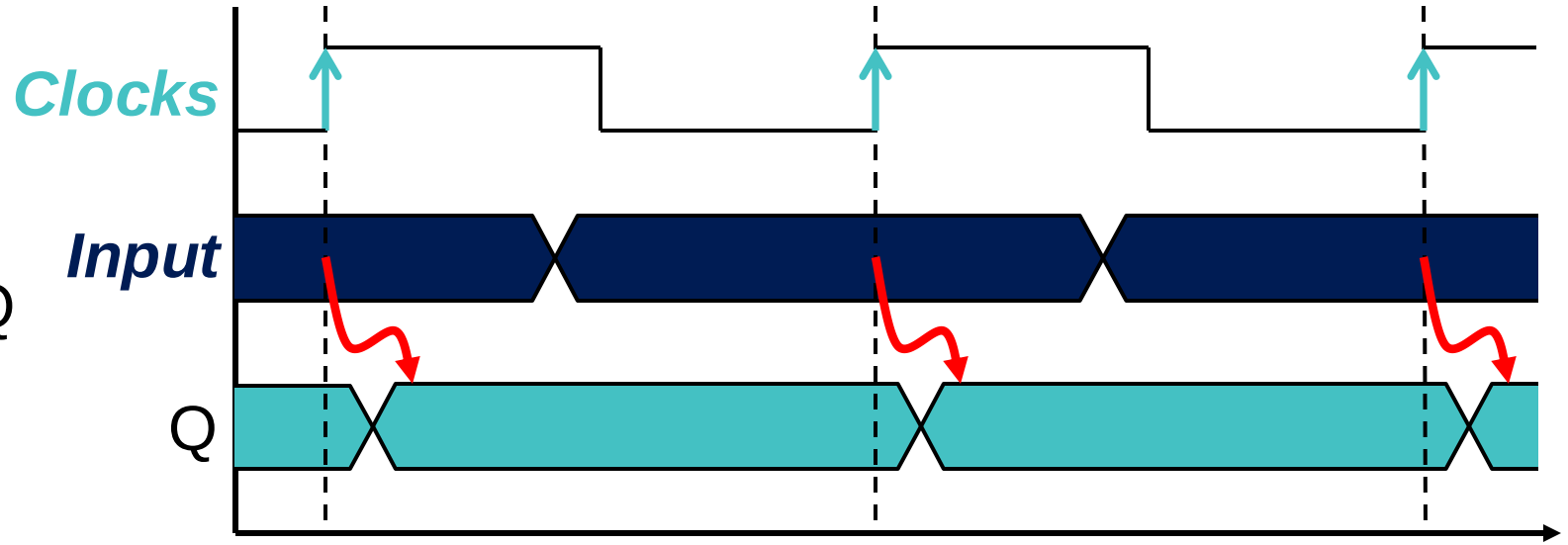
Write data

C
D
Register 0
Register 1
Register 30
Register 31

State Element: Register

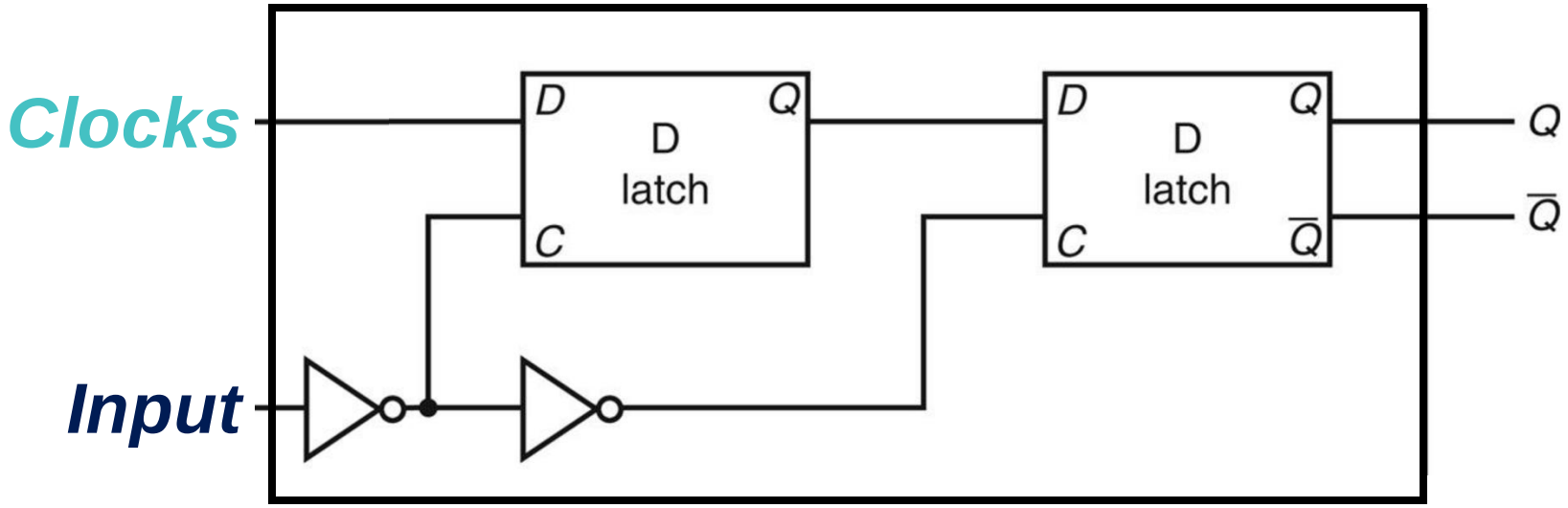
- **Register**: stores data in a circuit, i.e., *D flip-flop*
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when clocks change from 0 to 1



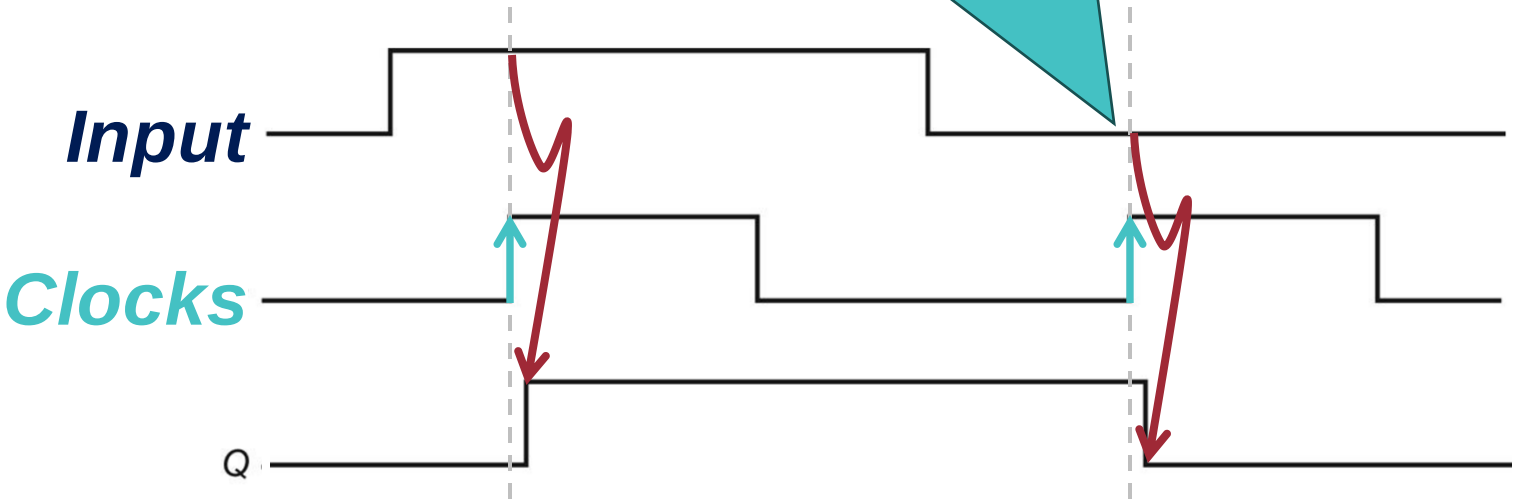


Recap: D Flip-flop

- Output changes ***only on the clock edge***

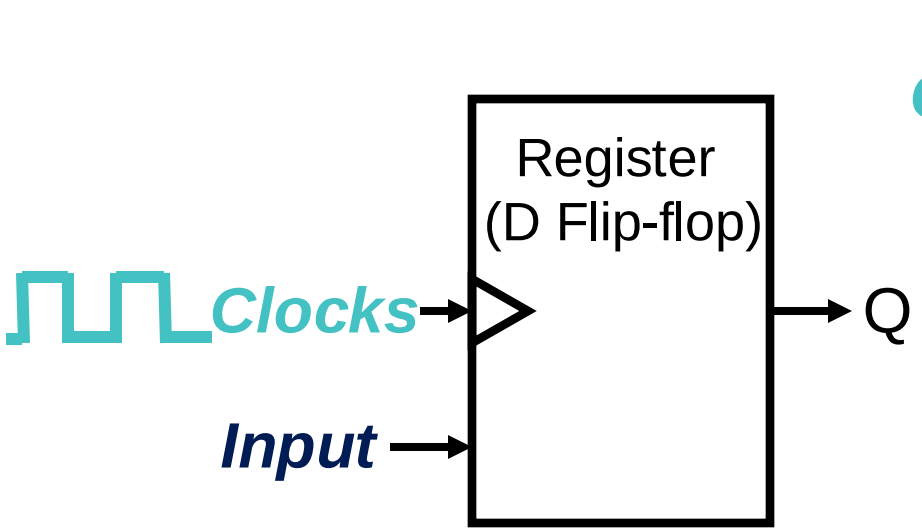


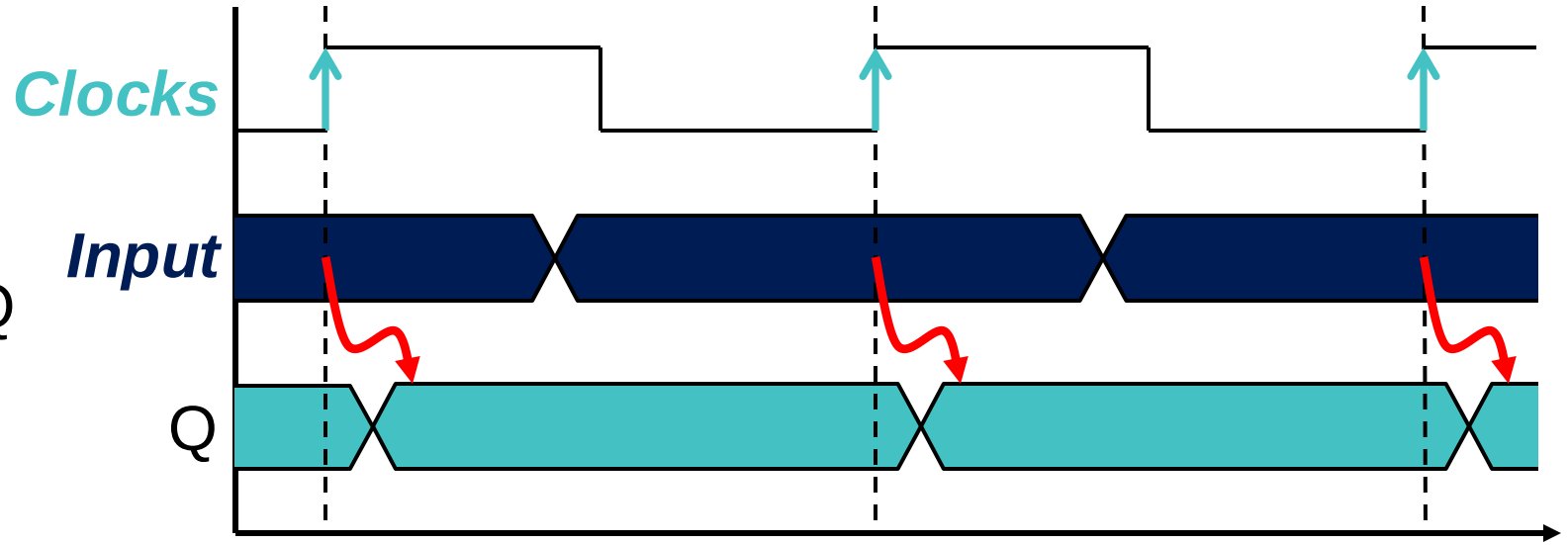
*Output changes **only on the rising edge***



State Element: Register

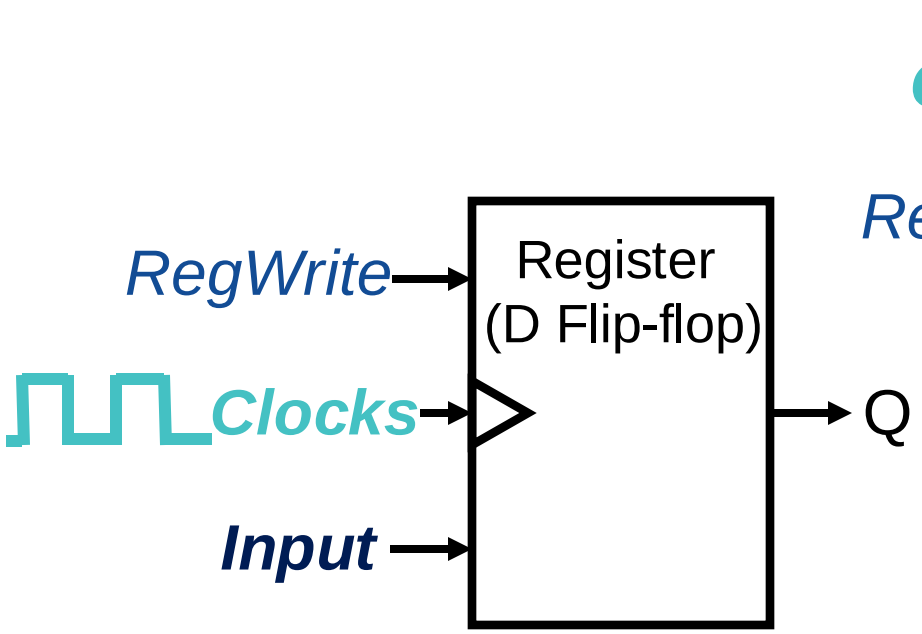
- **Register**: stores data in a circuit, i.e., *D flip-flop*
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when clocks change from 0 to 1

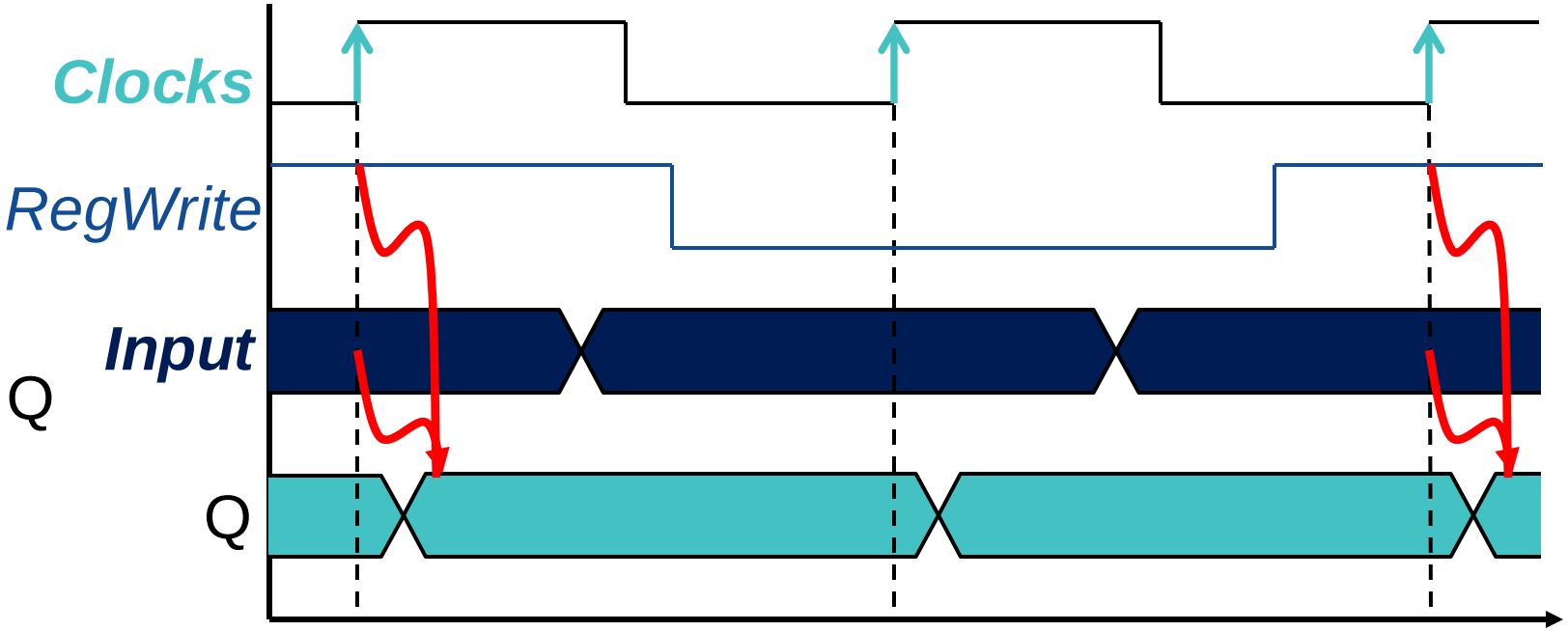




Register with *RegWrite* Control

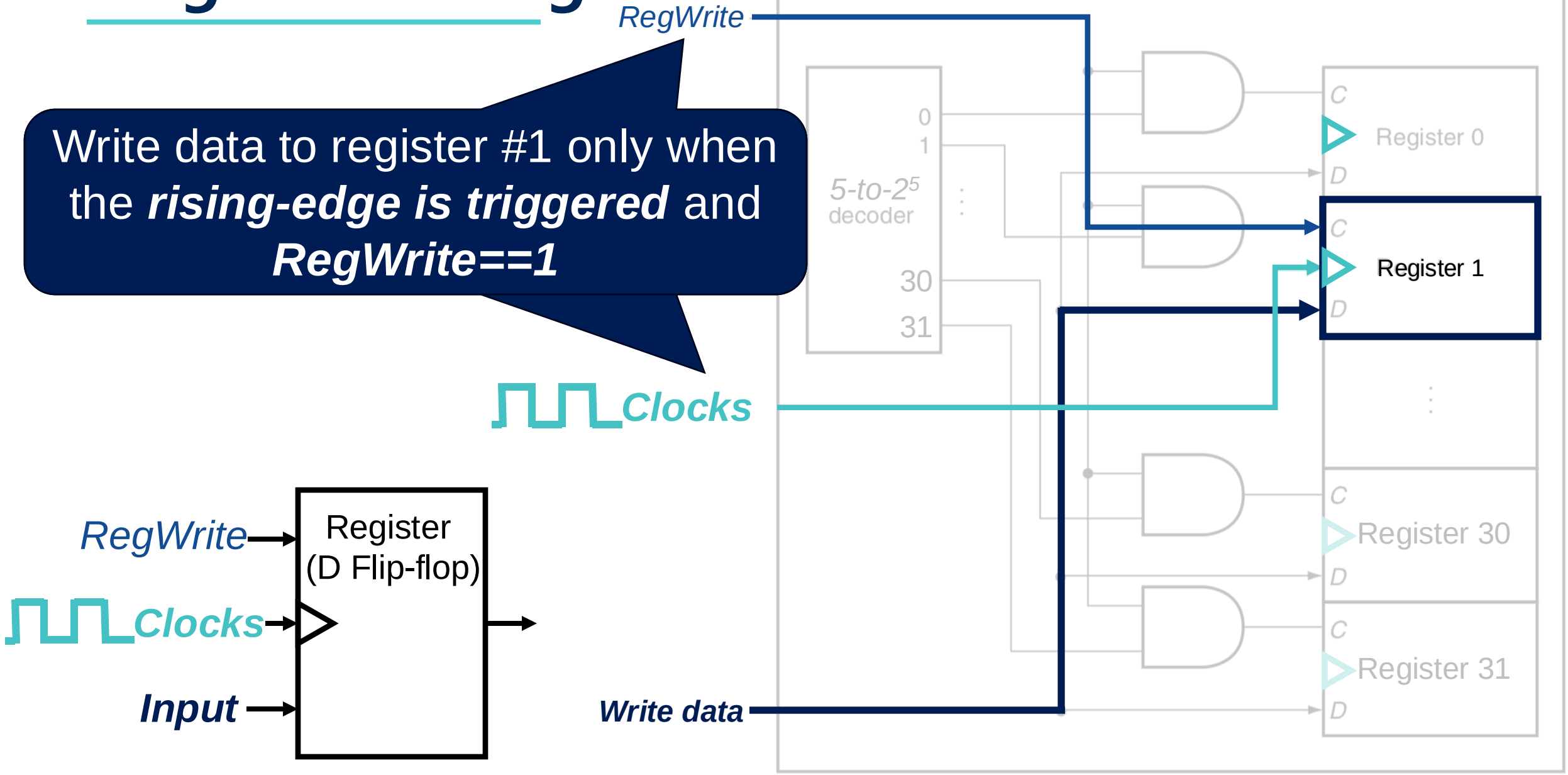
- **Register**: stores data in a circuit, i.e., *D flip-flop*
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when clocks change from 0 to 1
  - Only updates on rising-edge when *write control signal (i.e., RegWrite)* is 1
    - Registers are not written every cycle (e.g. sw), so we need an explicit write control signal for the registers



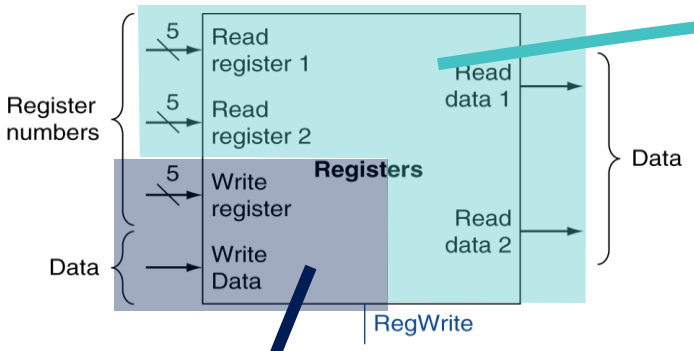


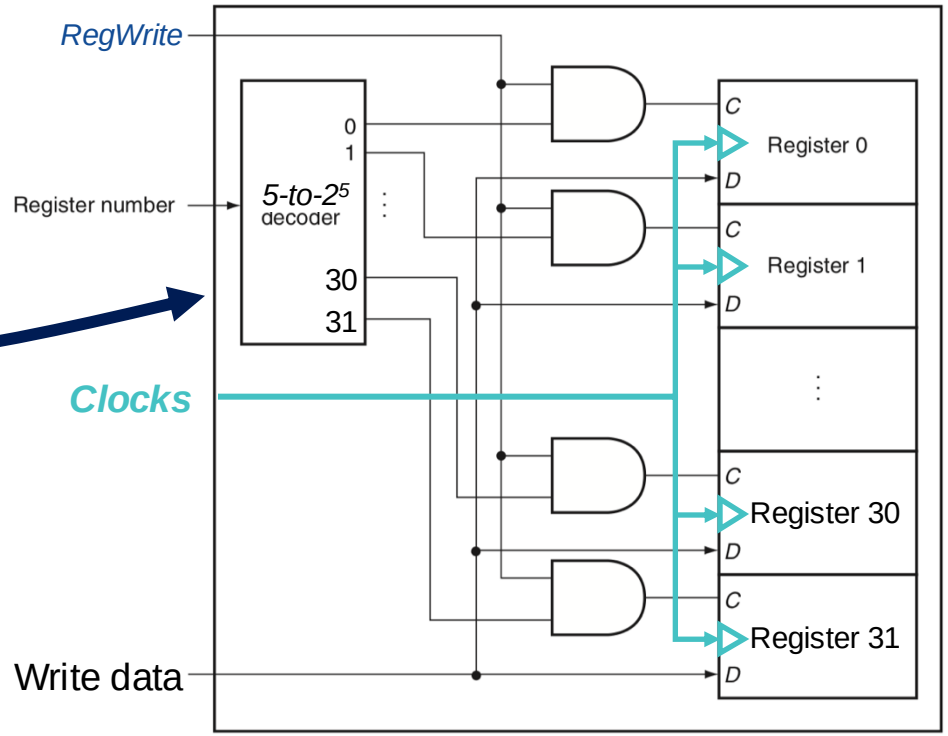
Register in Register File

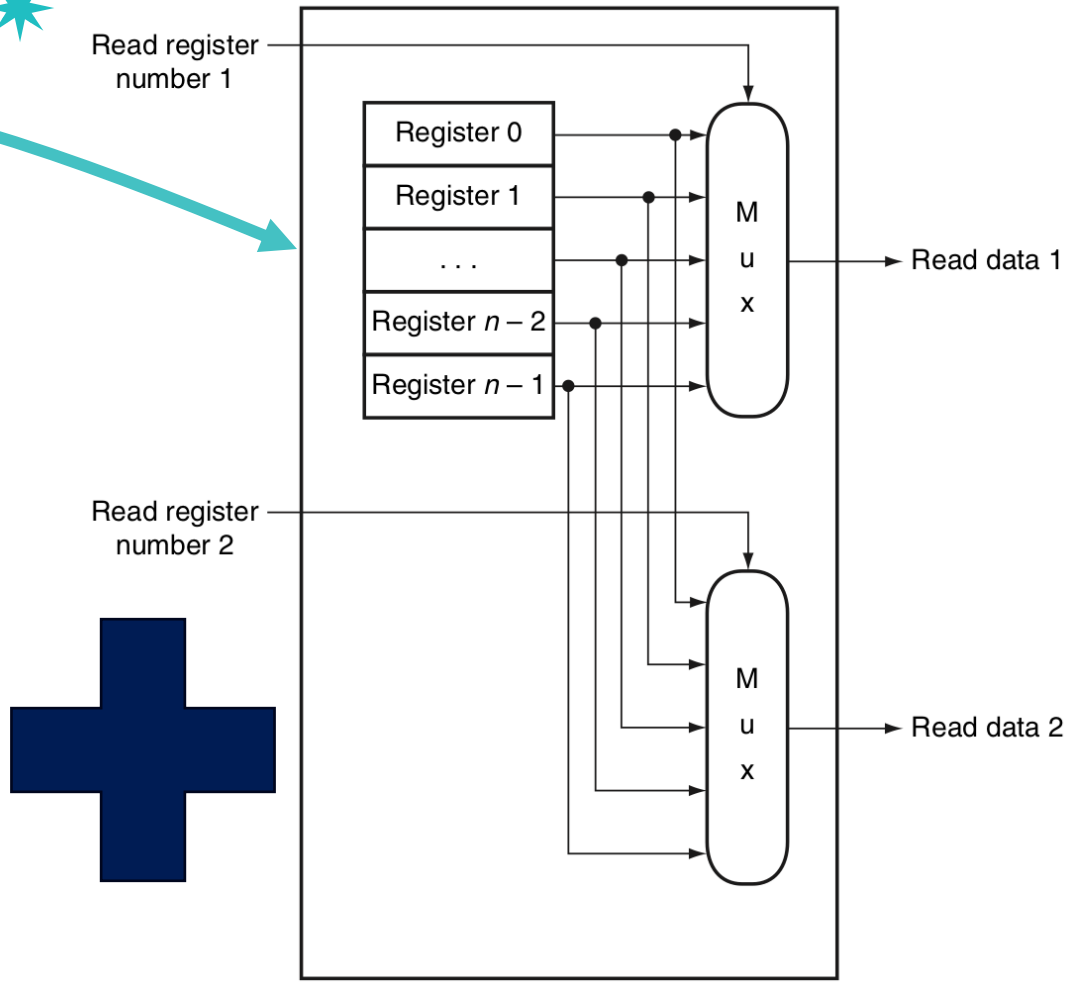
Write data to register #1 only when the ***rising-edge is triggered*** and ***RegWrite==1***



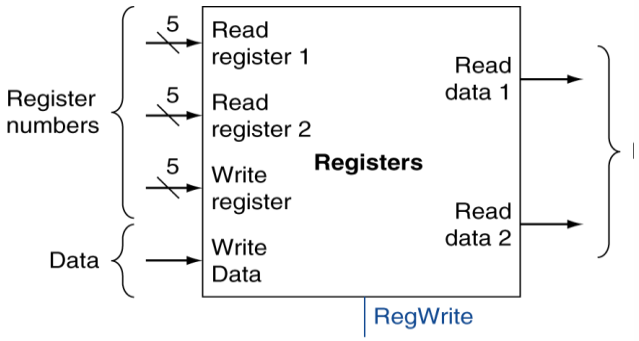
Register File: Read and Write

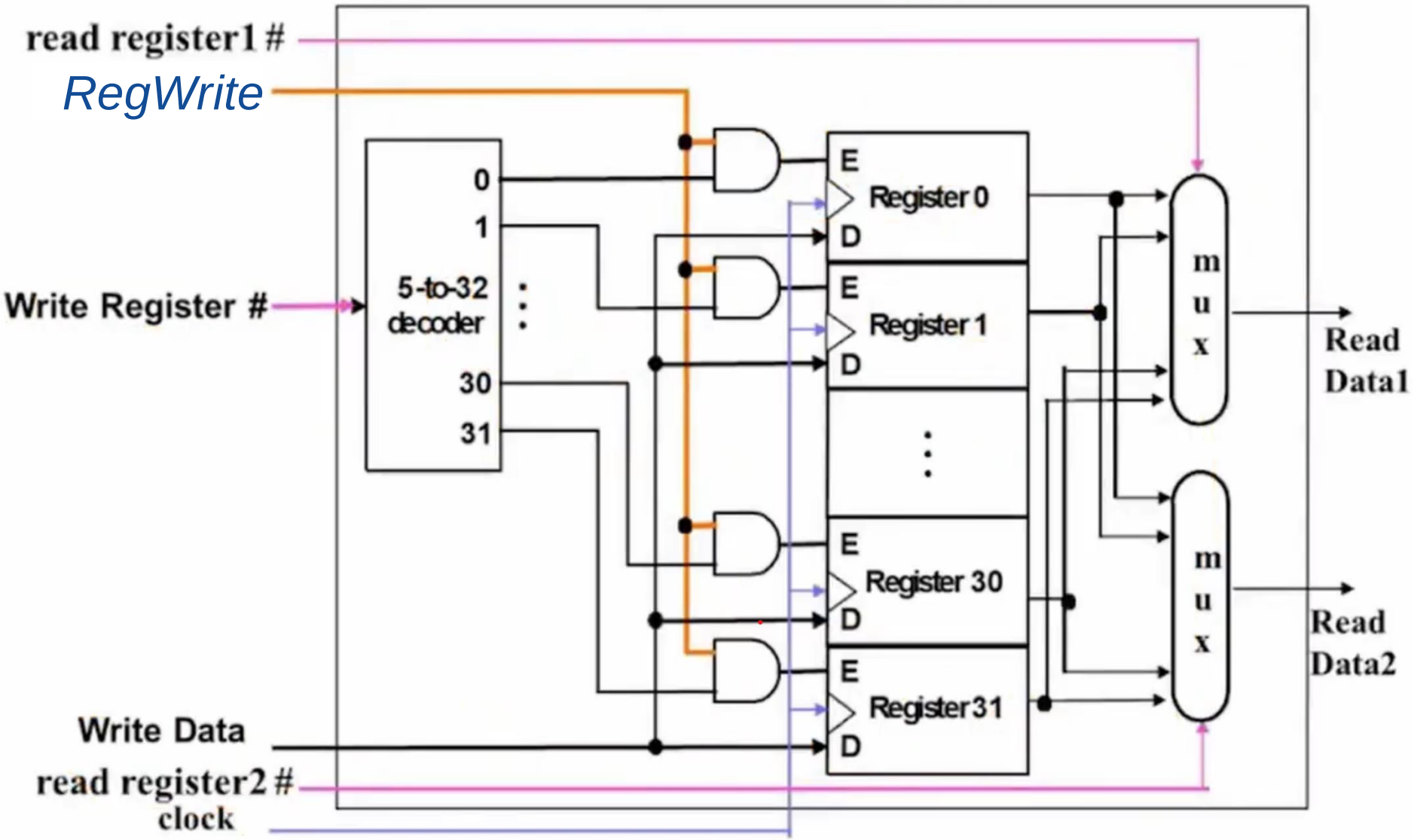




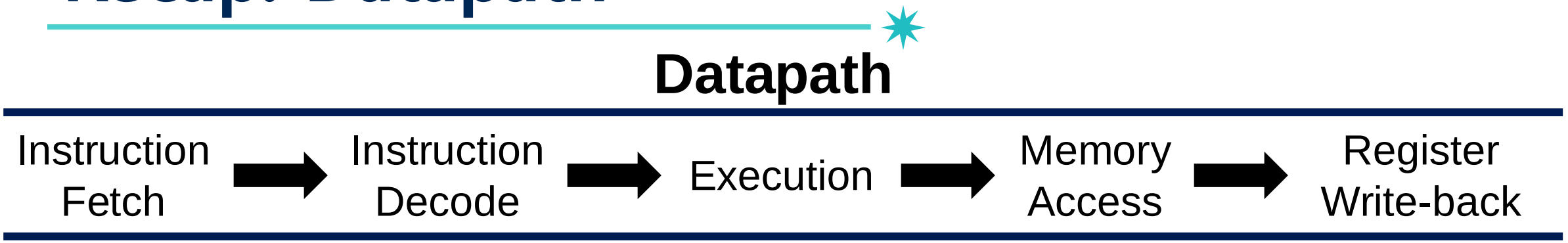


Register File Read and Write: Final View





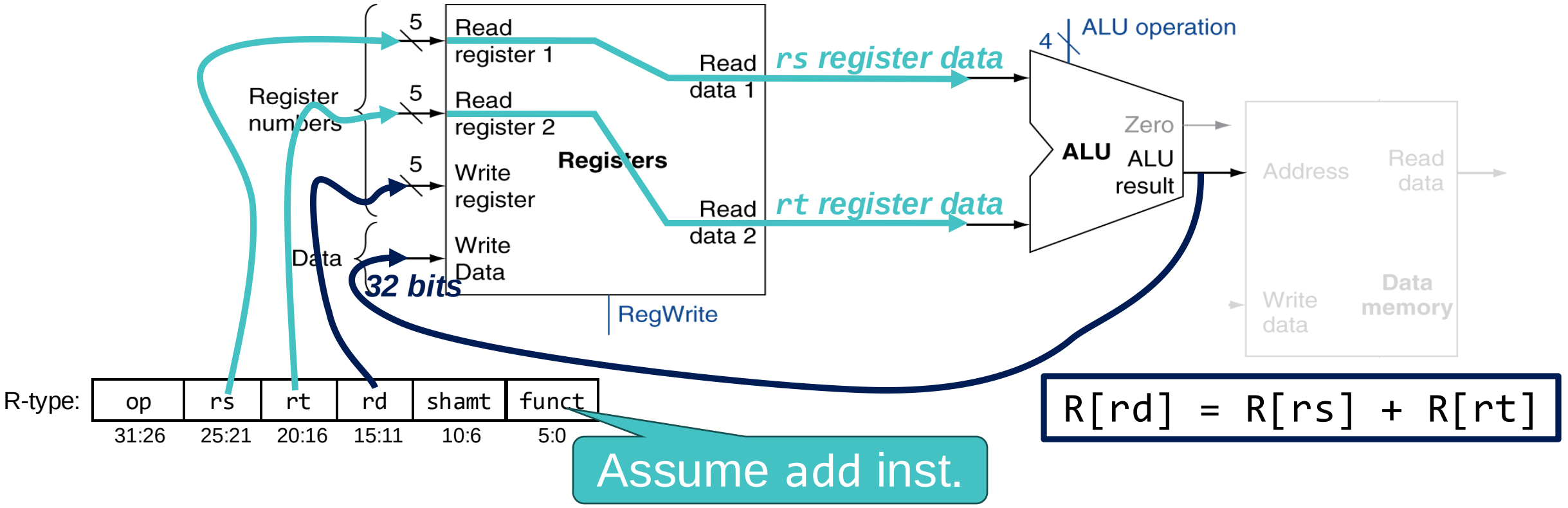
Recap: Datapath



Summary: R-type Instruction in Datapath

Datapath

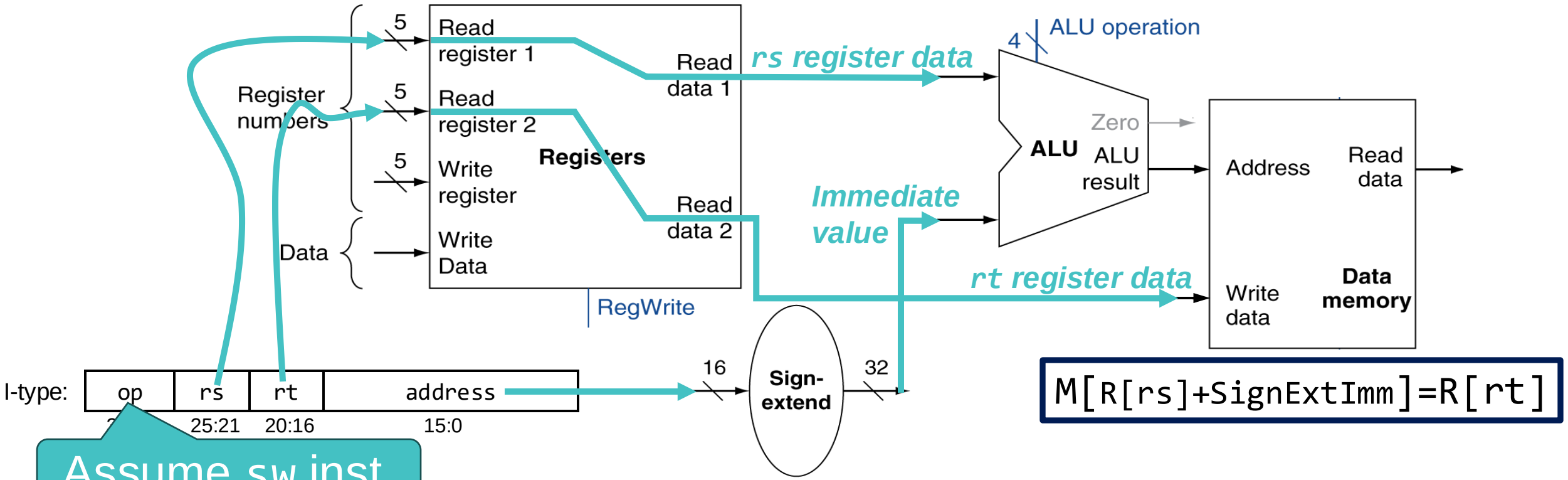
Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back



Summary : sw Instruction in Datapath

Datapath

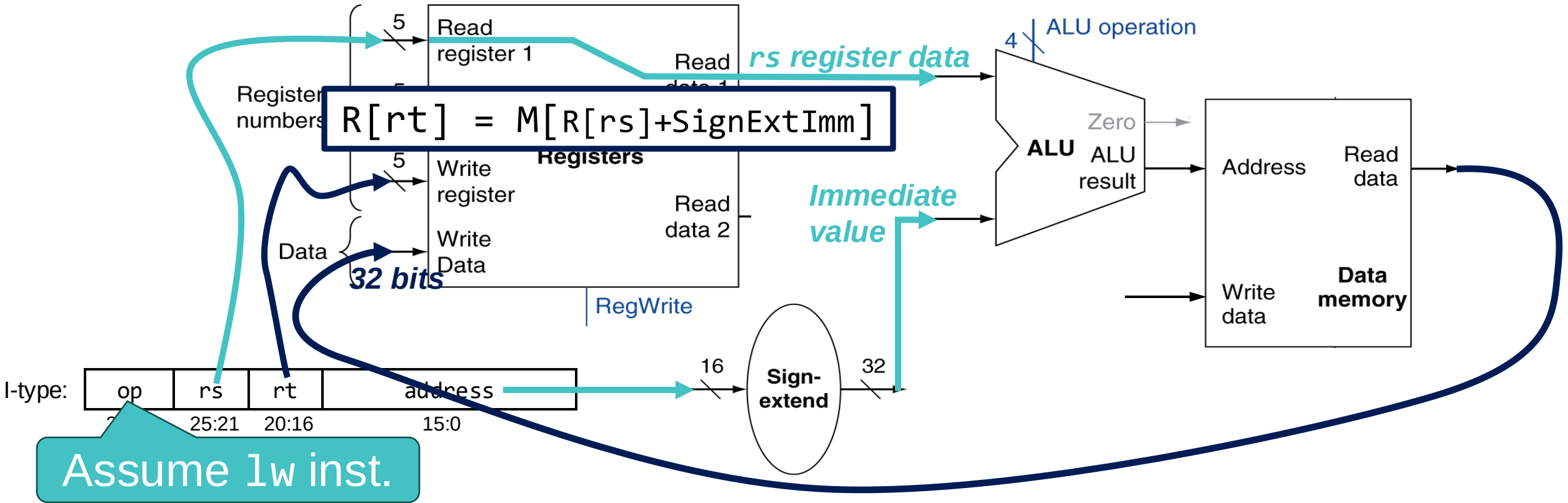
Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back



Summary : lw Instruction in Datapath

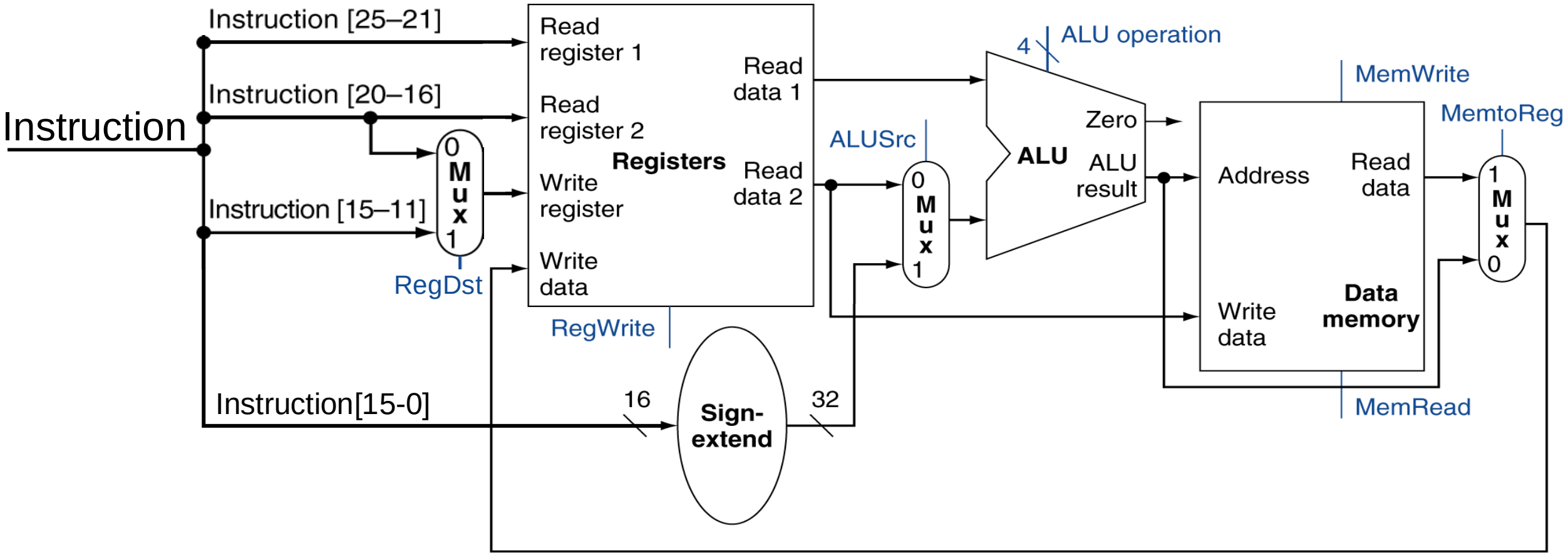
Datapath

Instruction Fetch → Instruction Decode → Execution → Memory Access → Register Write-back



Let’s Combine Datapath Elements!

R-type/Load/Store Datapath

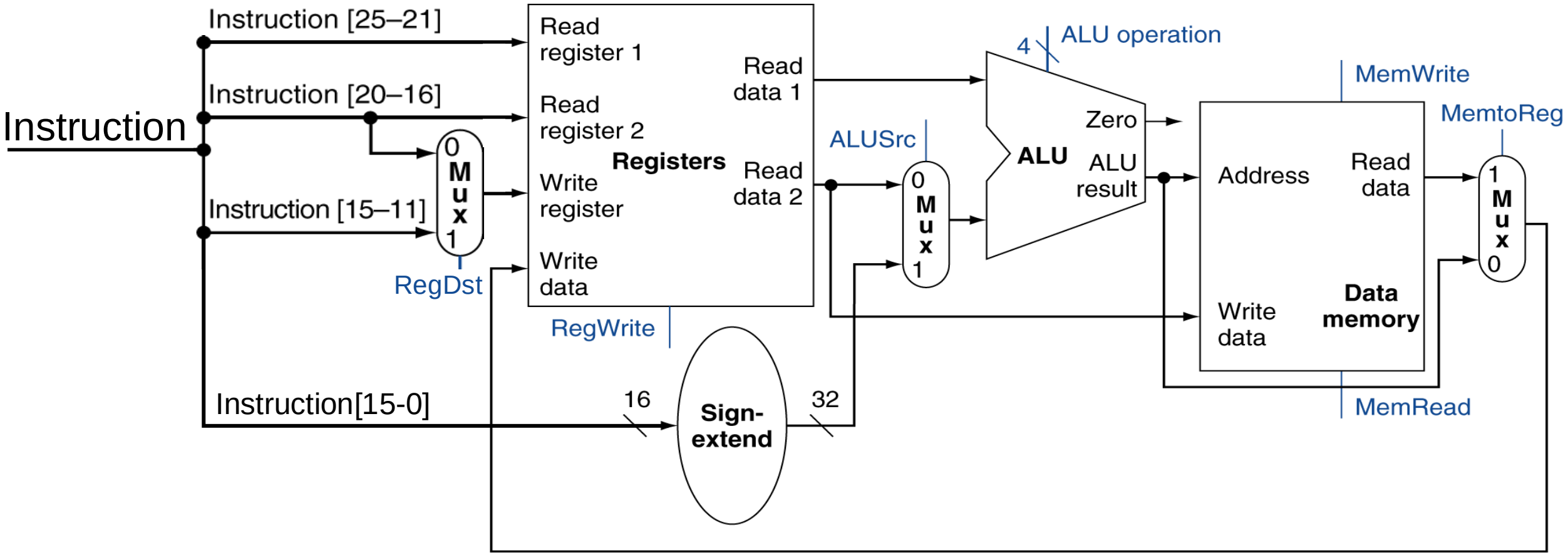


R-type Instruction: add

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

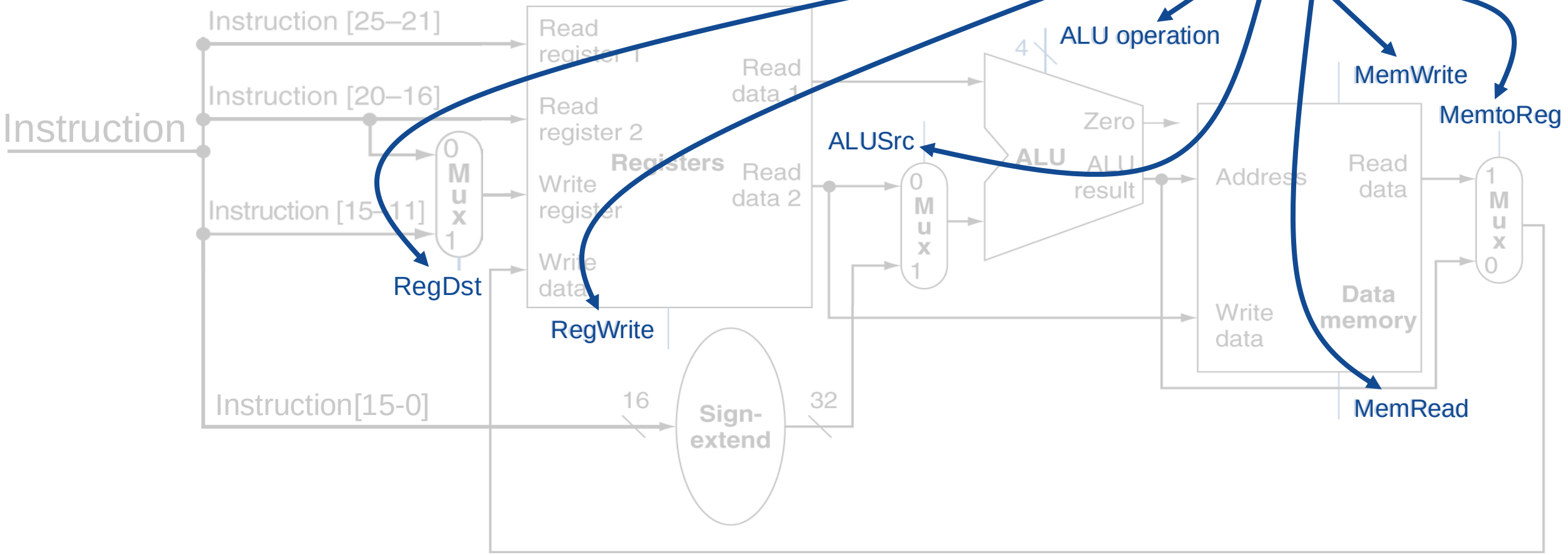
add instruction

R-type:

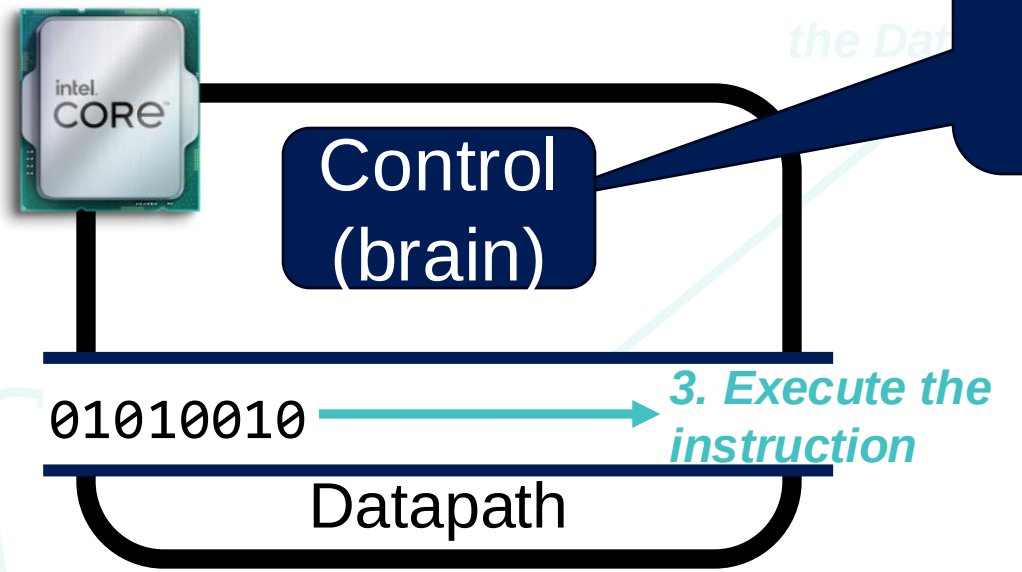
op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Control

Send signals

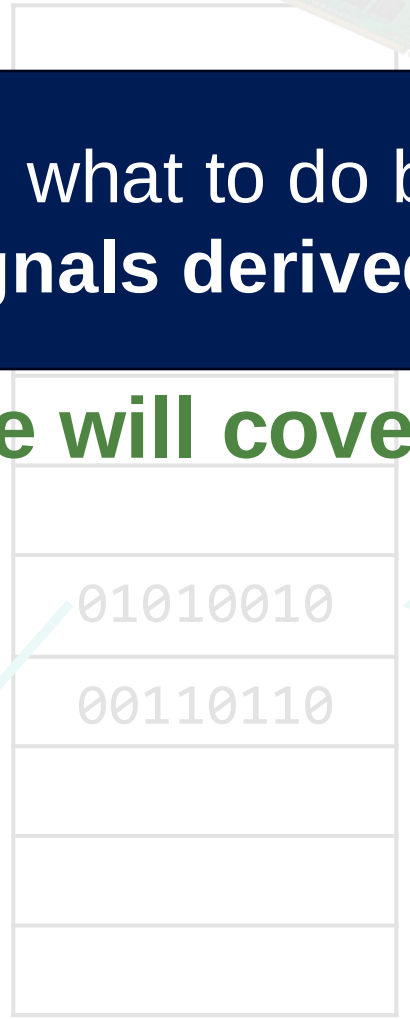


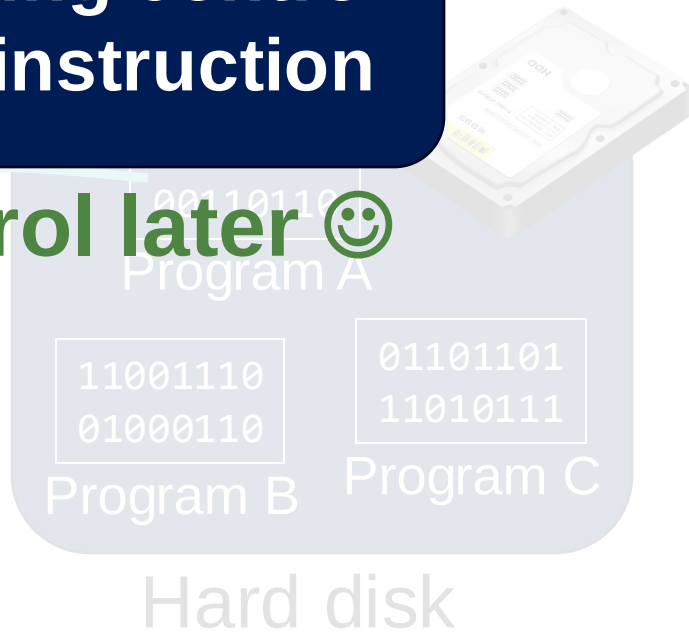
Introduction to Control



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

We will cover control later ☺





2. Fetch the instruction (One-by-one)

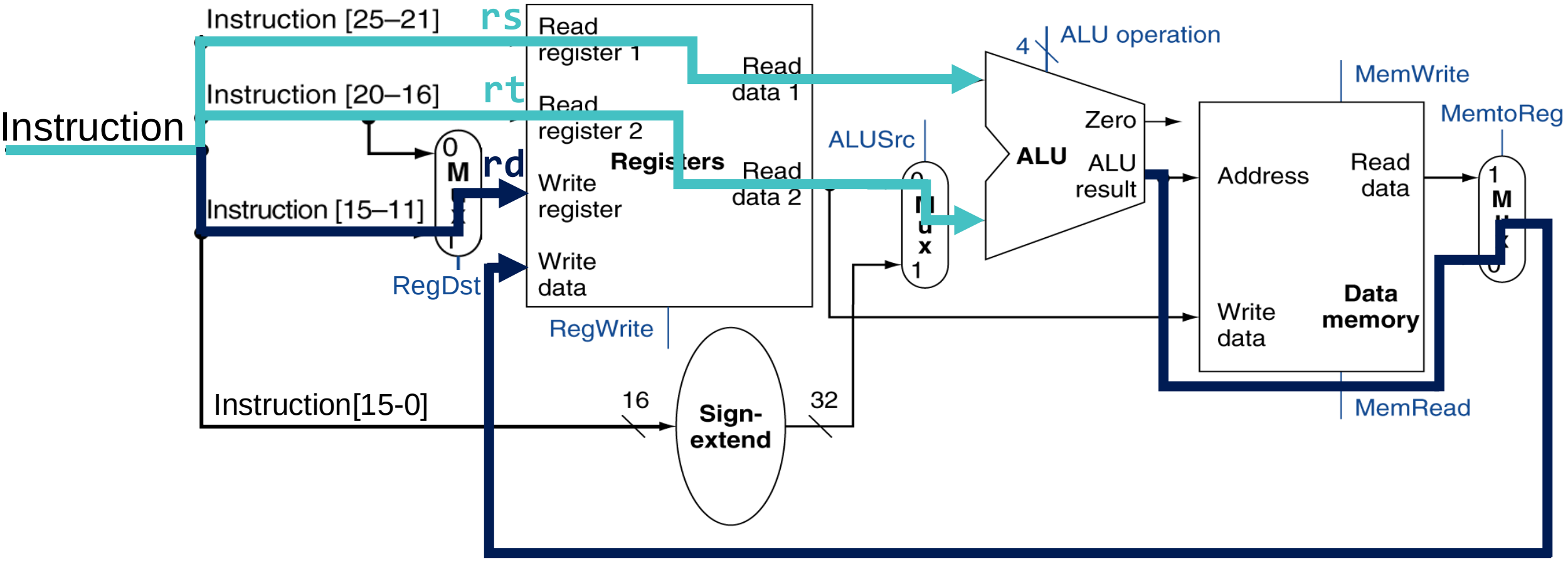
R-type Instruction: add

add instruction

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

R-type:

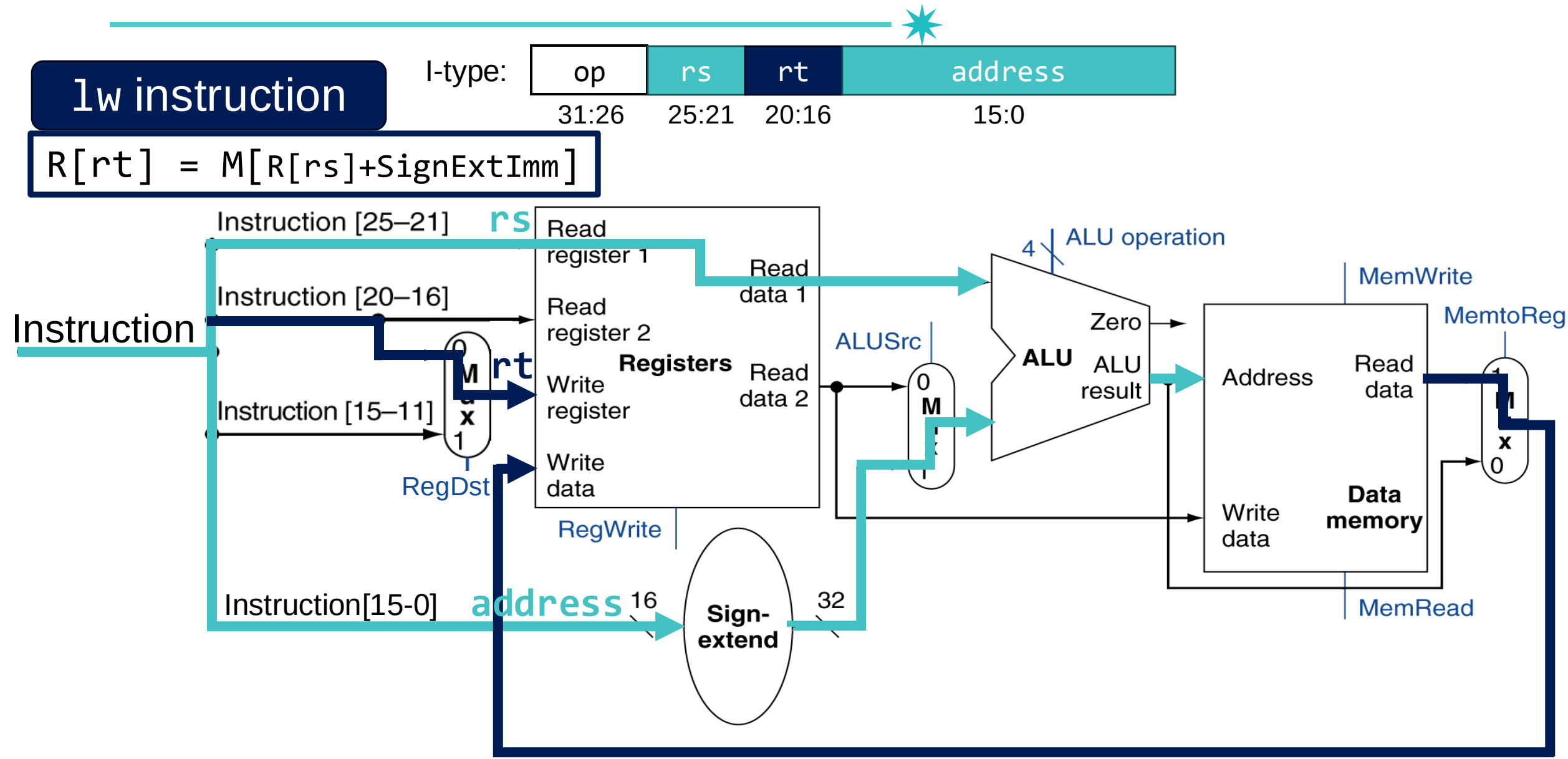
op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0



Load Instruction: lw

lw instruction

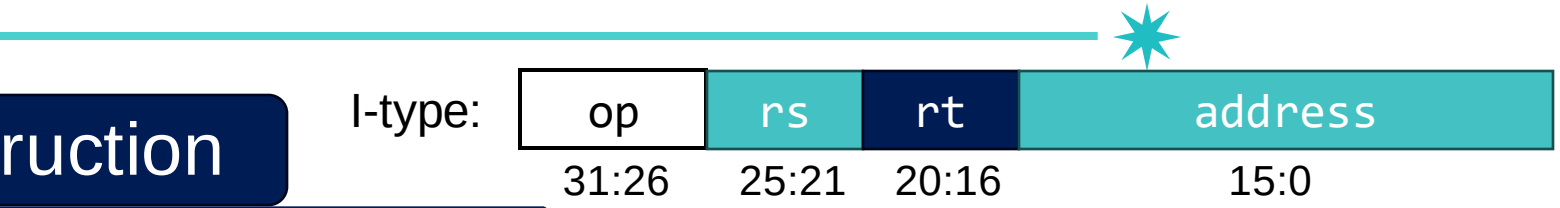
```
R[rt] = M[R[rs]+SignExtImm]
```

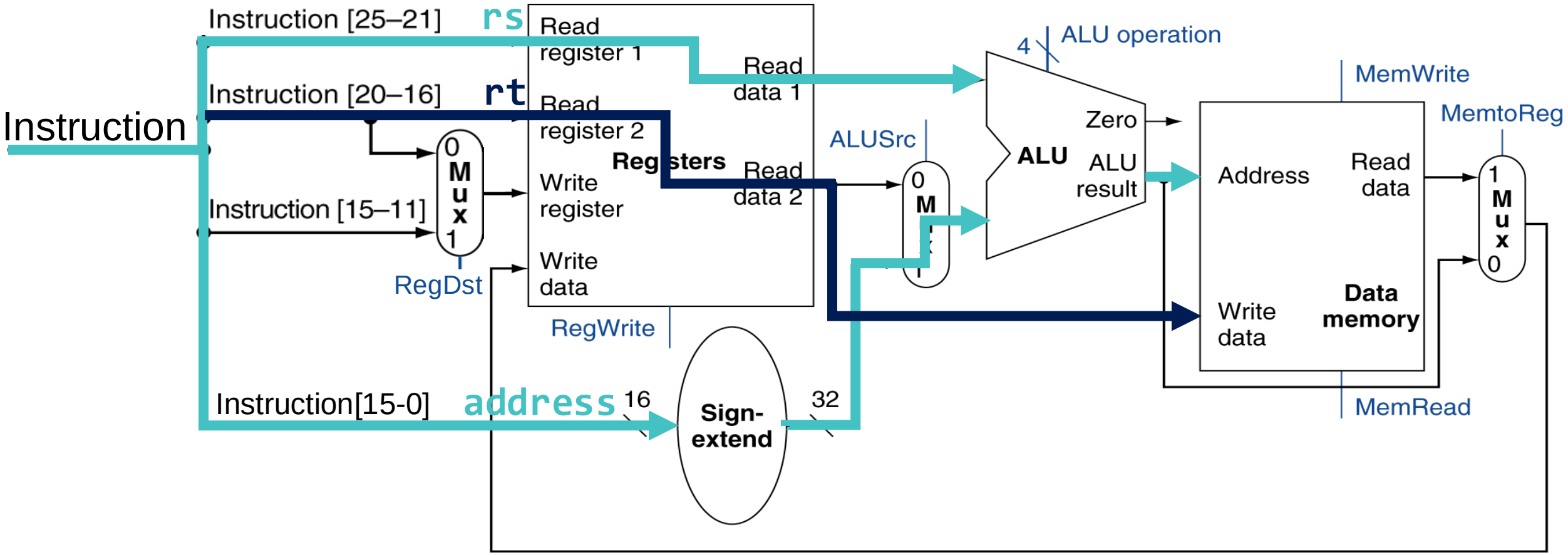


Store Instruction: sw

sw instruction

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

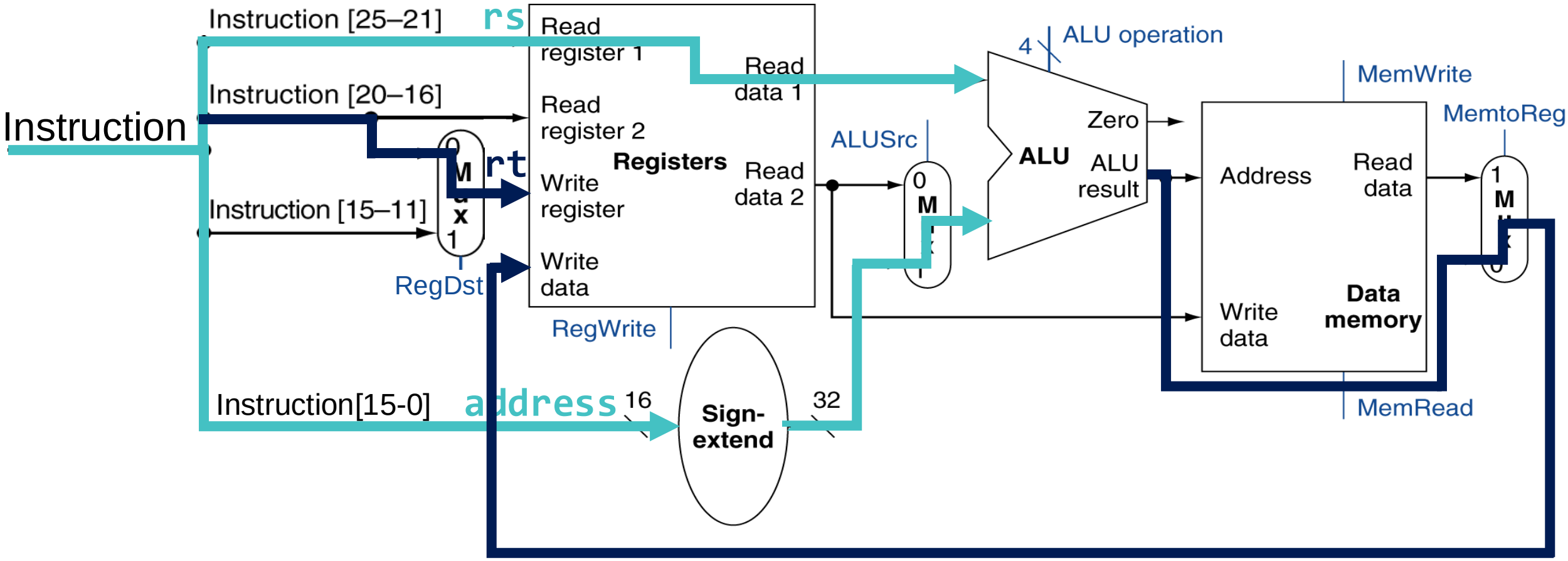




Store Instruction: addi

addi instruction

R[rt]=R[rs]+SignExtImm



Question

addi instruction

R[rt]=R[rs]+SignExtImm

I-type: op | rs | rt | address

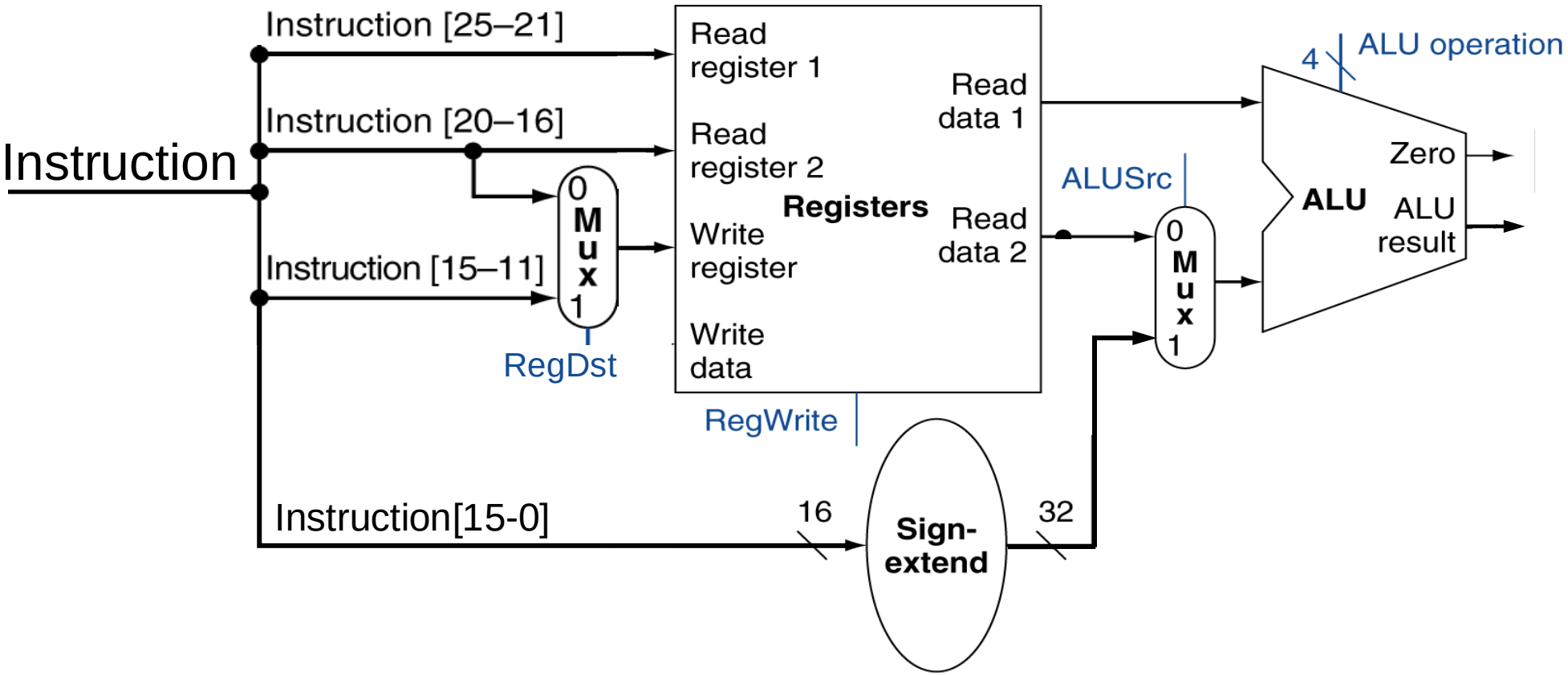
31:26 25:21 20:16 15:0

How about branch instructions (e.g., beq, bne)?

Datapath for a Branch

beq instruction

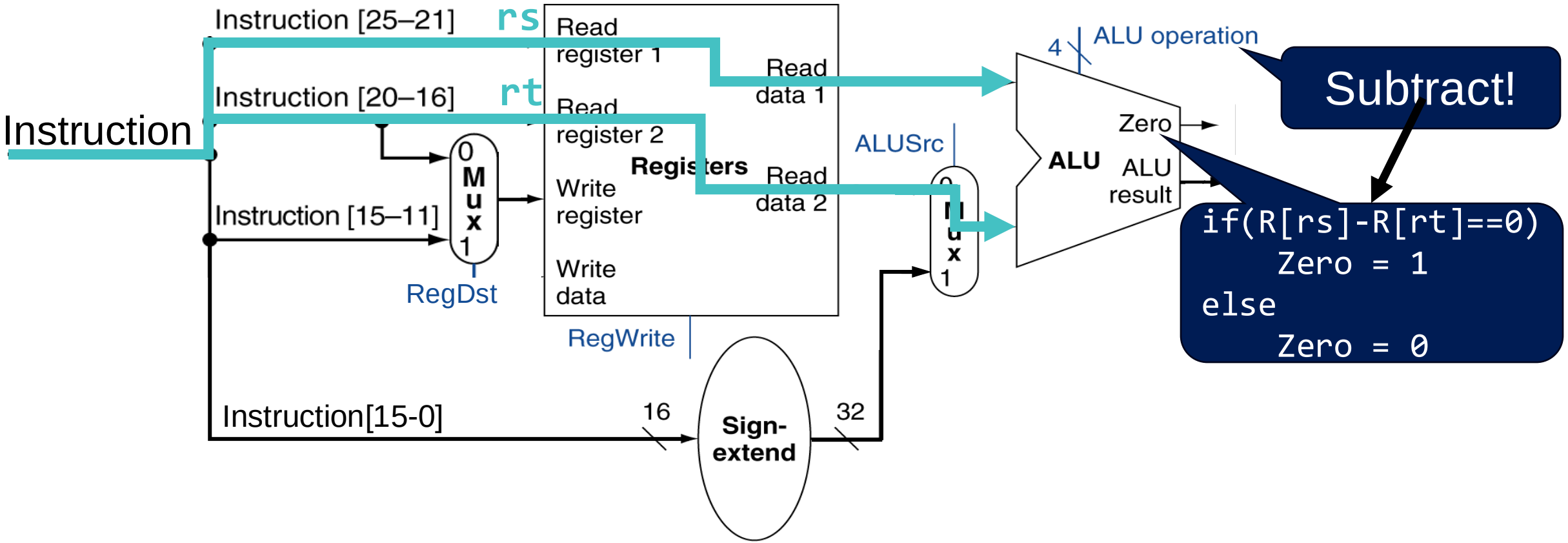
```
if(R[rs]==R[rt])
    PC = PC+4+BranchAddr
```



Datapath for a Branch

beq instruction

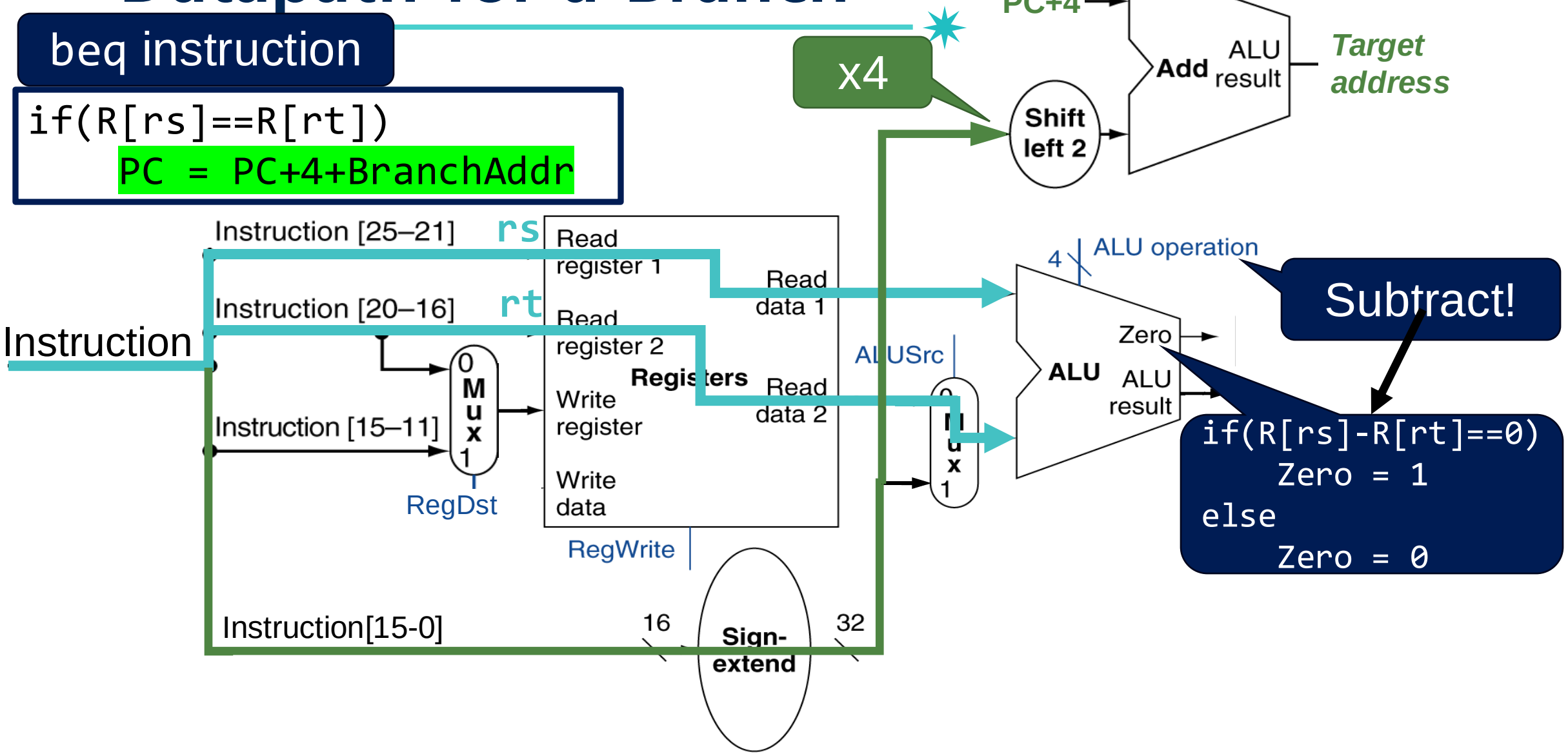
```
if(R[rs]==R[rt])
    PC = PC+4+BranchAddr
```



Datapath for a Branch

beq instruction

```
if(R[rs]==R[rt])
    PC = PC+4+BranchAddr
```



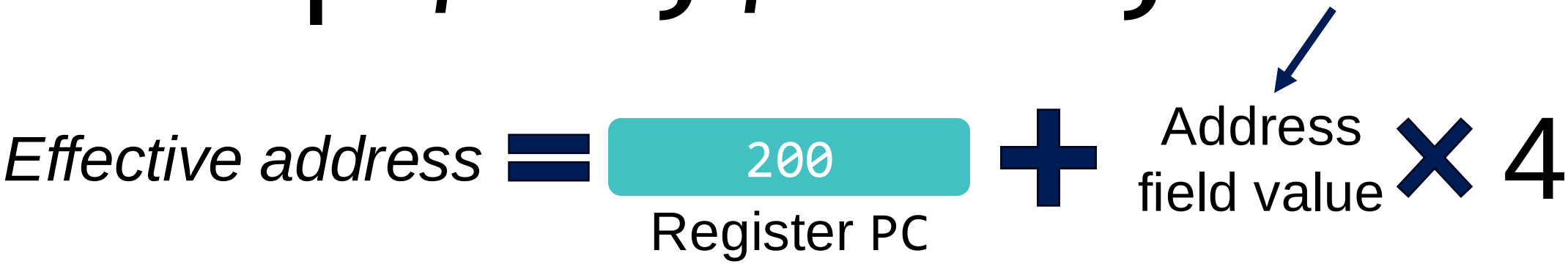
Recap: PC-Relative Mode

The content of PC is added to the address part of instruction to obtain the *effective address* (branch type instructions)

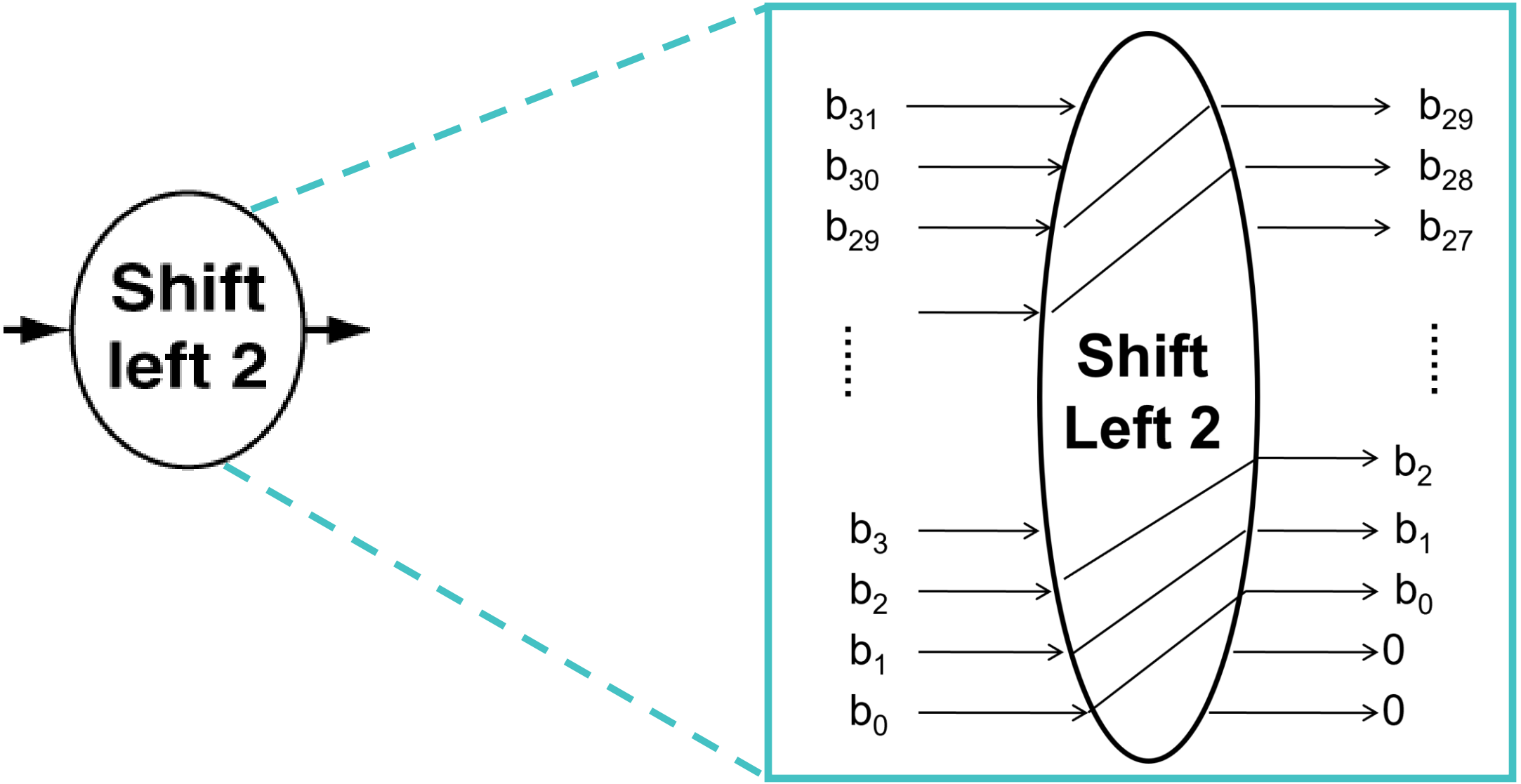
- *Effective address*: PC + the address part of instruction*4
- Operand value: memory[effective address]

Example: MIPS instruction

```
beq $t0,$zero,else
```



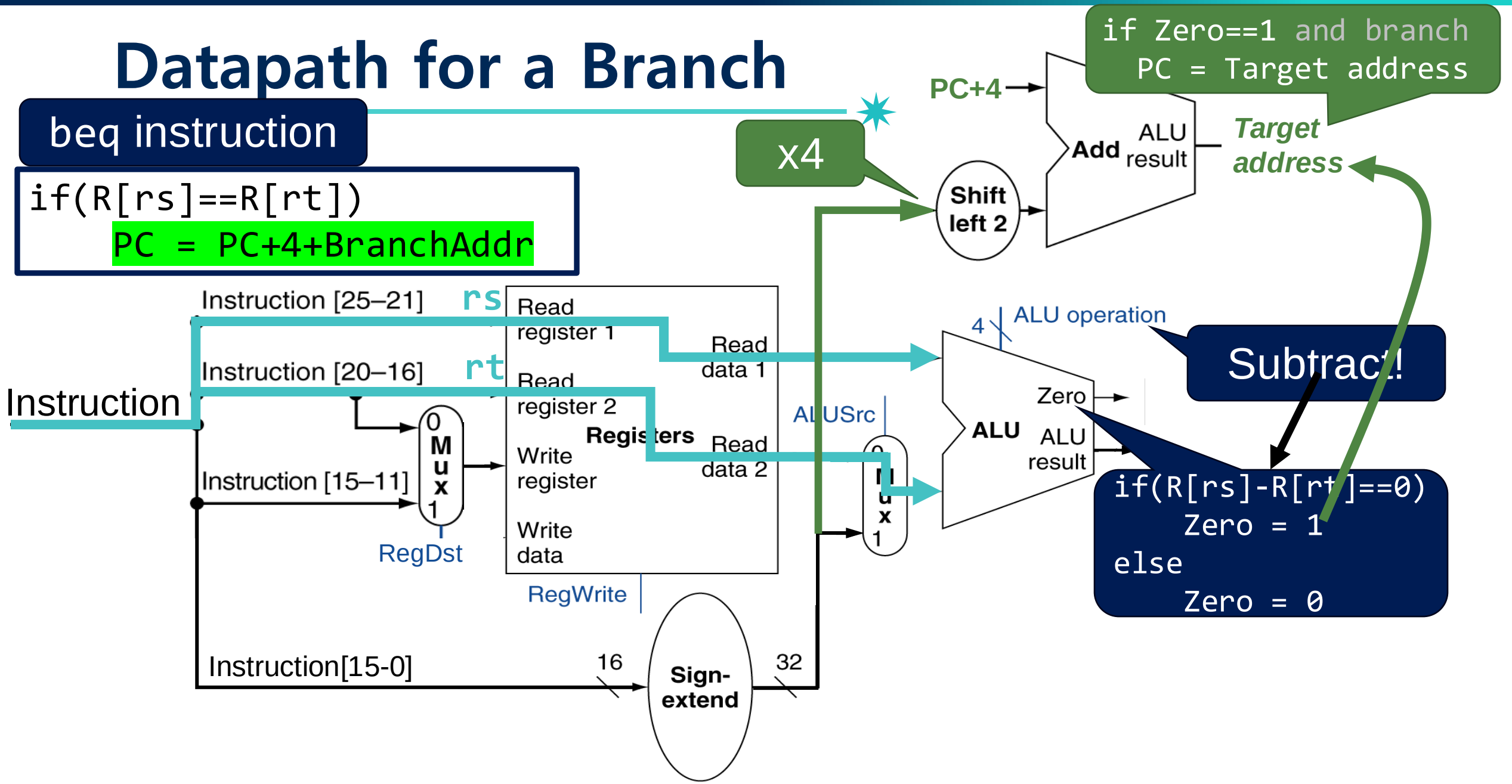
FYI: Shift Left 2



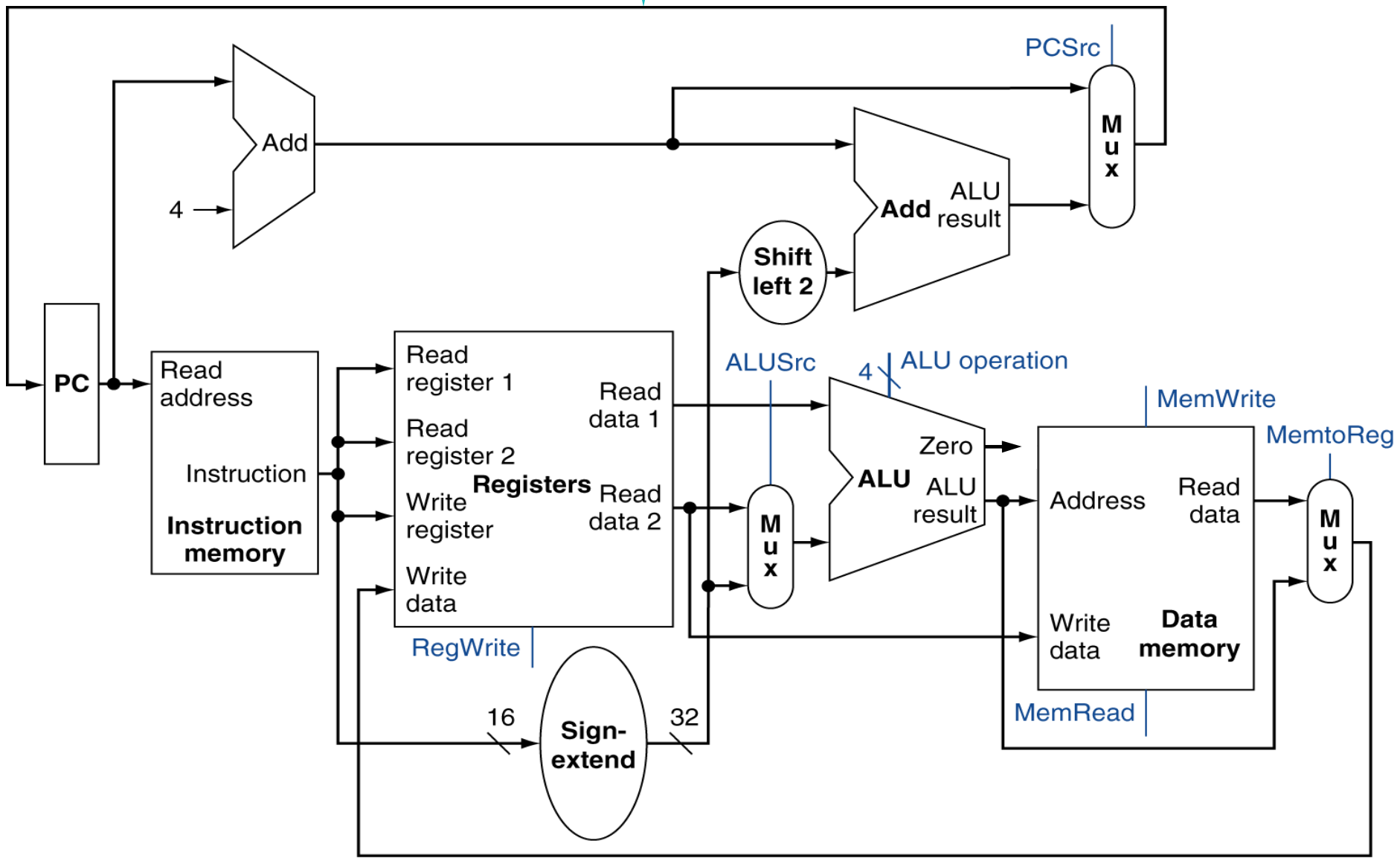
Datapath for a Branch

beq instruction

```
if(R[rs]==R[rt])
    PC = PC+4+BranchAddr
```

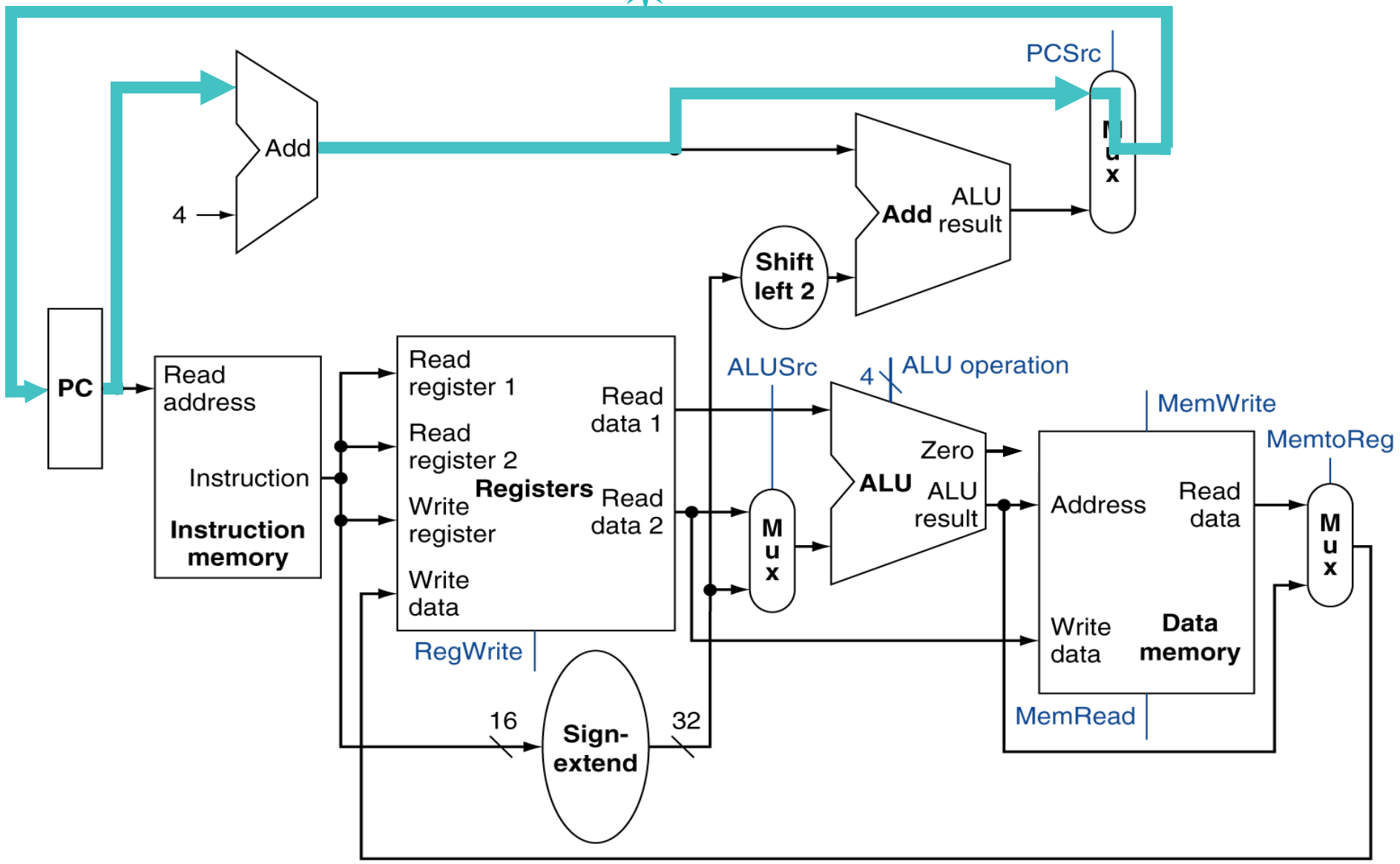


Full Datapath (R-Type and I-Type)



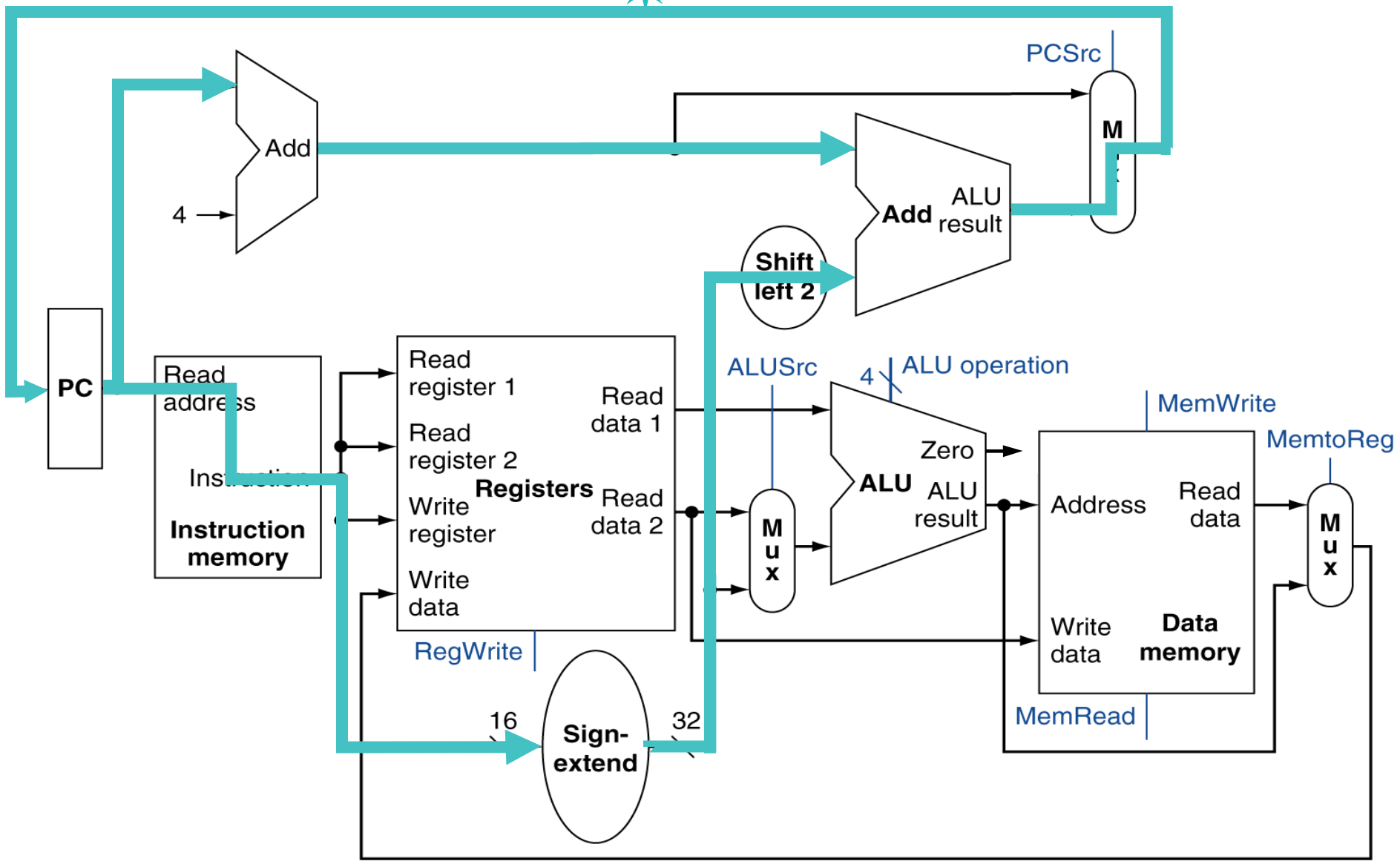
Full Datapath (R-Type and I-Type)

*PC update path for **non-branch** instructions*

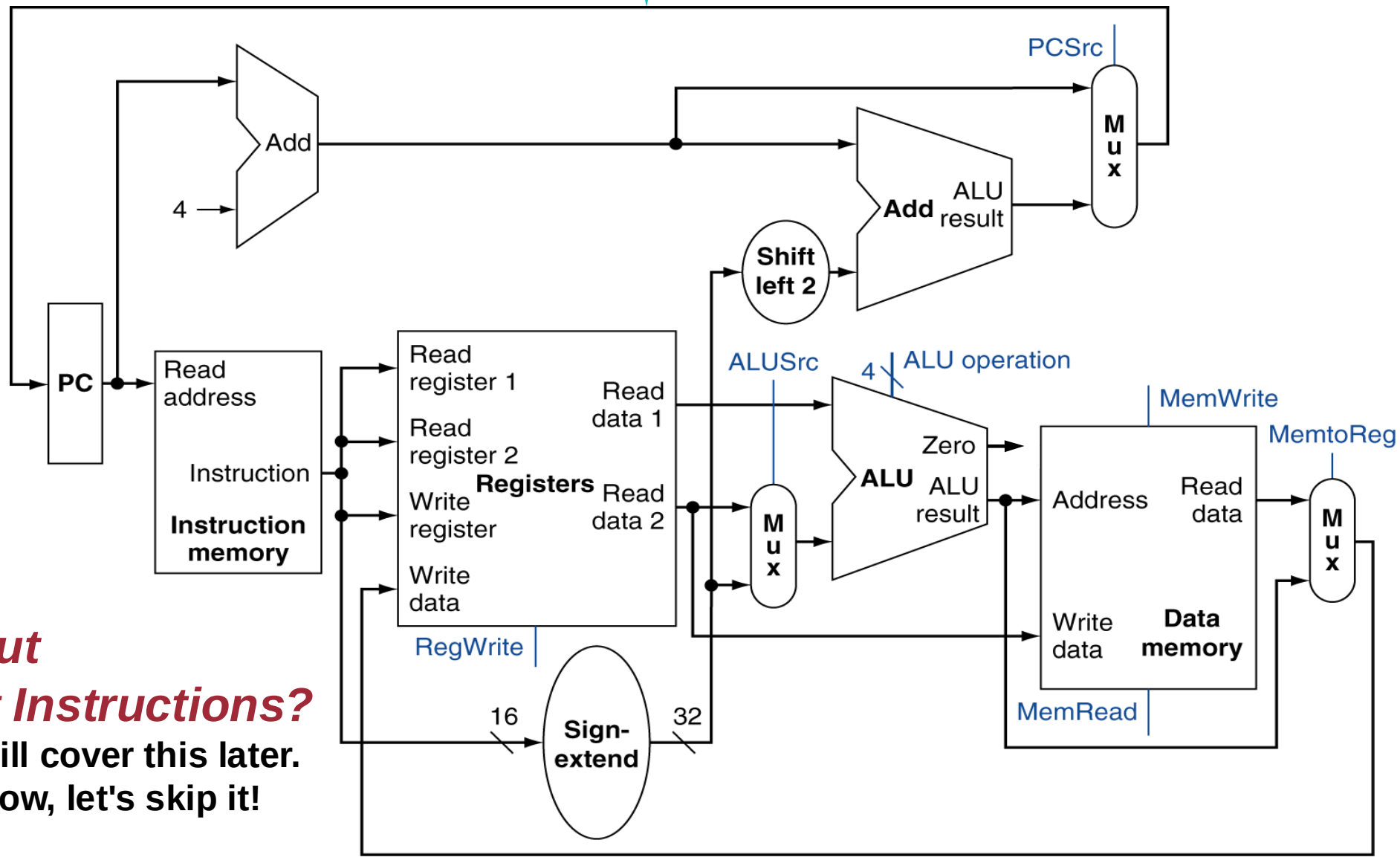


Full Datapath (R-Type and I-Type)

*PC update path for **branch** instructions*



Discussion Points (1): J-Format Instructions



How about J-Format Instructions?

We will cover this later.
For now, let's skip it!

Question?