11/11/26 (Make a wish…)

COMP 311: *Computer Organization!*

Lecture 23: More Hazards!



The University of North Carolina at Chapel Hill

https://tinyurl.com/comp311-fa26

Logistics Update



- Lab 4 is out
  - Due 11/23
- WA5 released today.
  - I will drop 1 WA
  - Due 11/16
- Will add more quiz corrections dates!

Plan for Class Today



- 3:35-4:00: Finish going over single-cycle RISC-V Datapath
- 4:00-4:15: Pipelining the RISC-V CPU?
- 4:15-4:20: BREAK
- 4:20-4:50: Pipeline Hazards

Guiding Question ☺

What happens when there are dependencies between instructions being computed at the same time in a CPU?



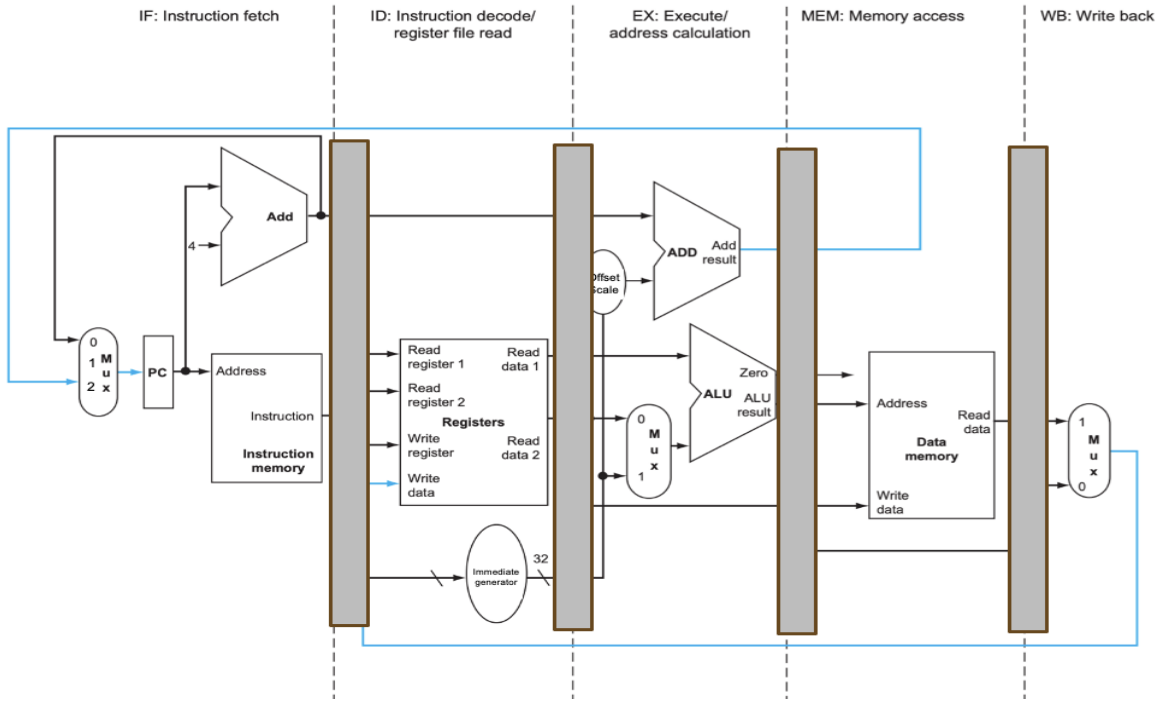
Pipeline Hazards



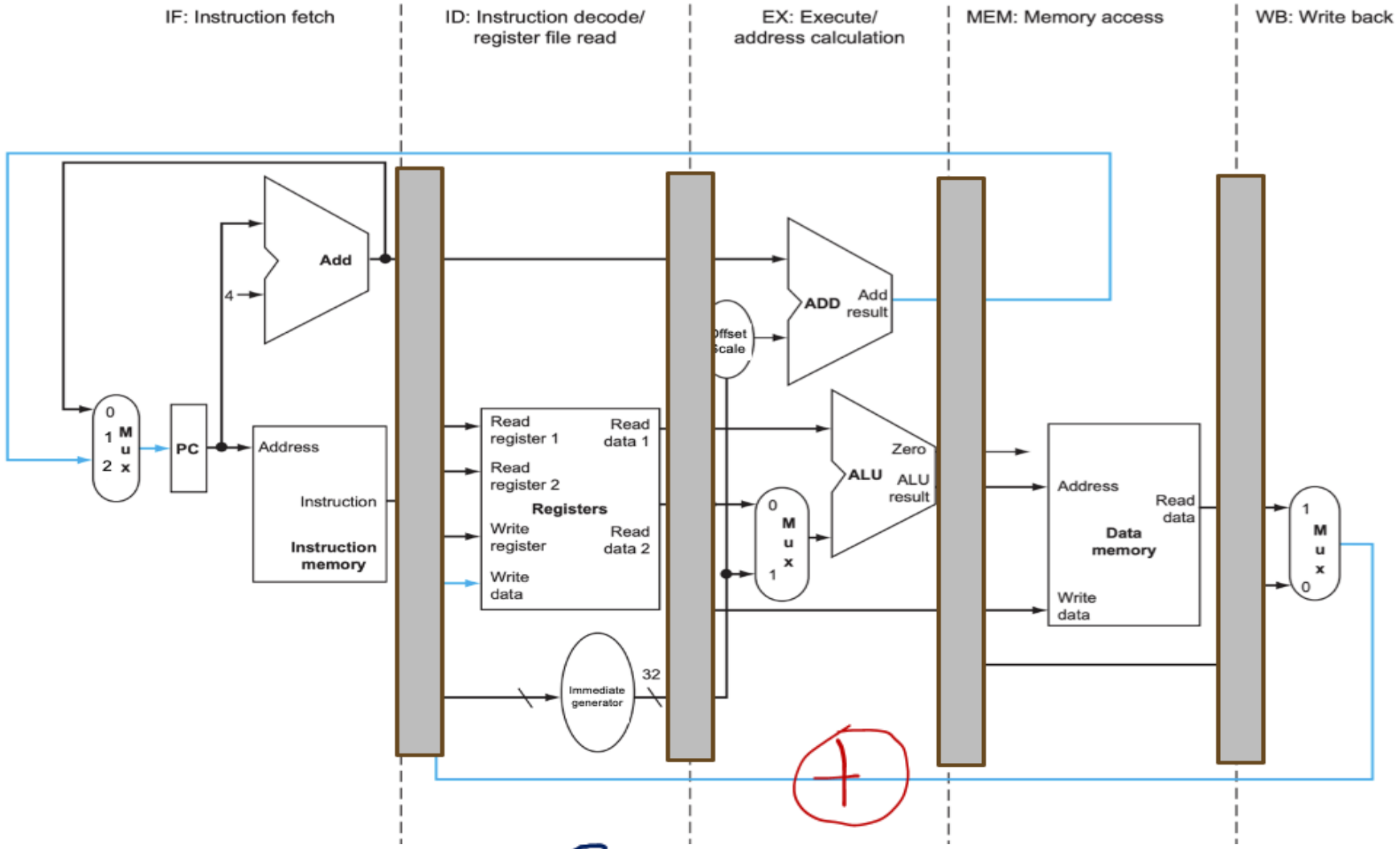
Data Hazard Tracing

Let's say we want to execute the following two instructions sequentially

```
add x5, x3, x4
sub x7, x5, x2
```



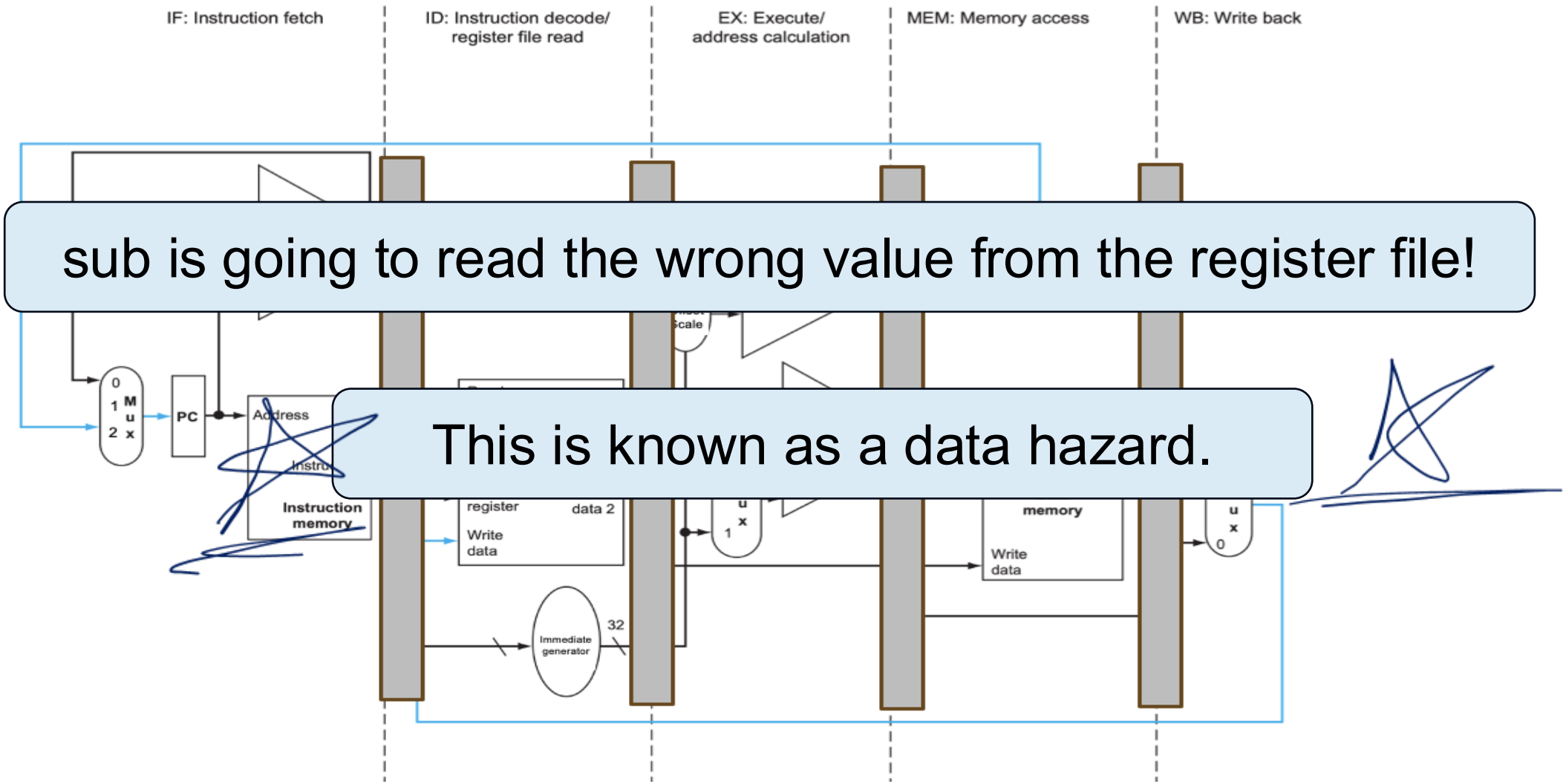
Cycle 2



sub x7 x5 x2 add x5, x3, x4

Cycle 2

RAW data



How can we fix this?



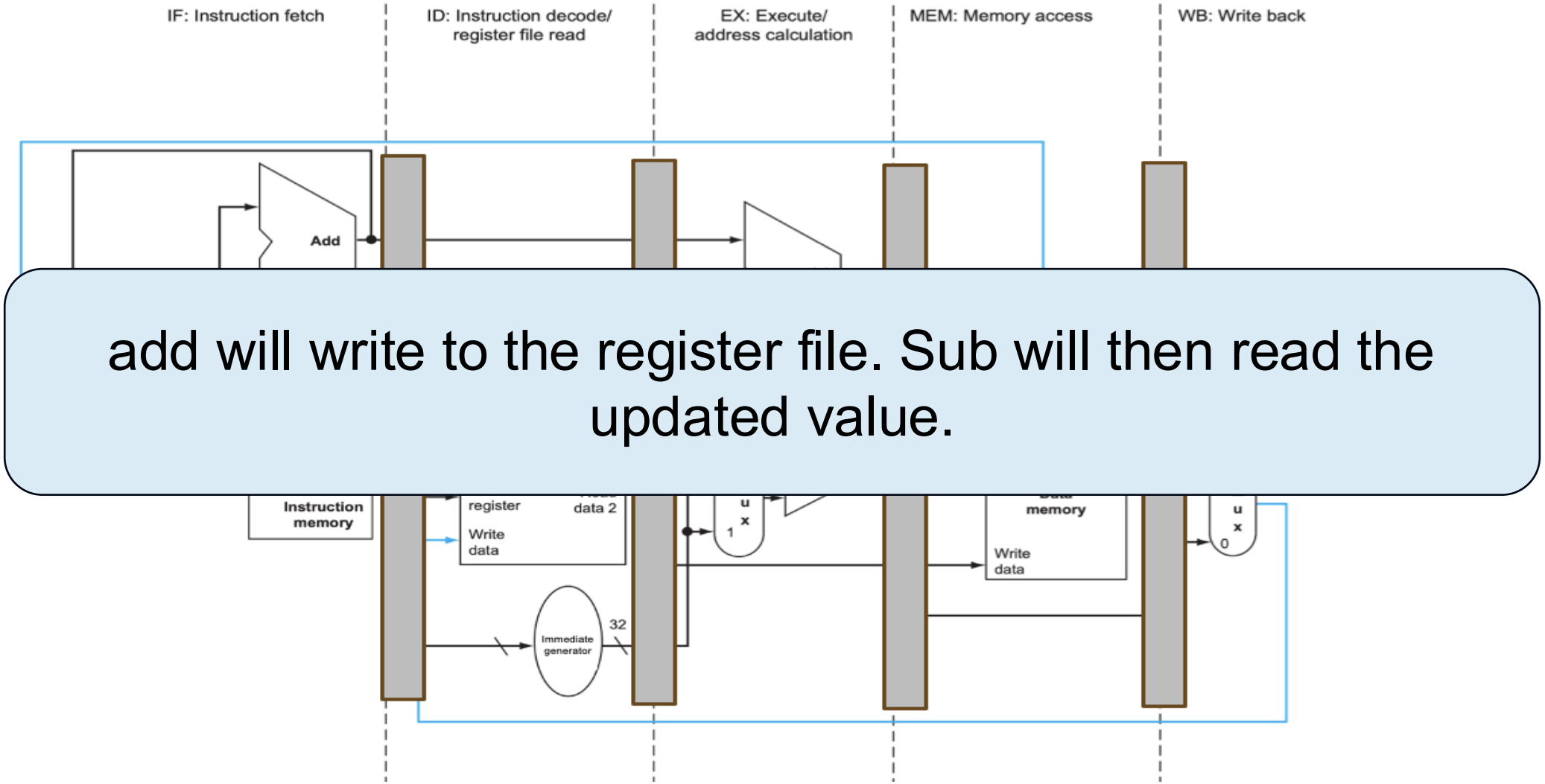
Introducing pipeline stalls!
(aka bubbles or nops)

Pipeline Stalls



- Also known as
  - bubbles
  - nops (no operation)
- An instruction that does nothing
- Provides time for the correct value to be written to the register file

Cycle 4



add will write to the register file. Sub will then read the updated value.

sub x7, x5, x2 ***STALL*** ***STALL*** add x5, x3, x4

Would we need to add any stalls for this program?



```
add x5, x3, x4
or x8, x9, x10
xor x14, x15, x16
sub x7, x5, x2
```



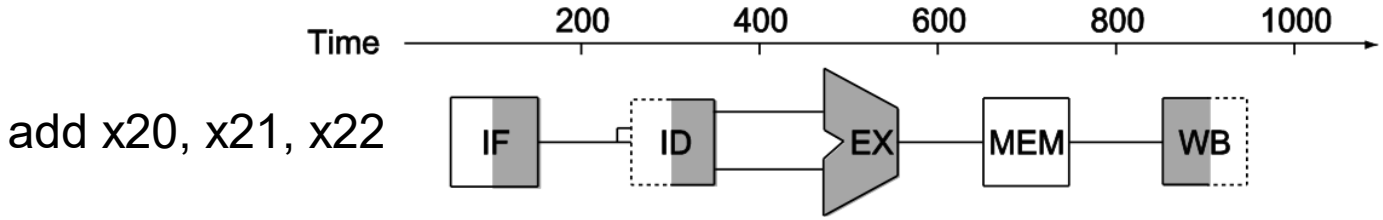
Pipeline Diagram

Provides a way for us to visualize instructions moving through the pipeline.

Instruction	Cycle #													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add x5, x3, x4	F	D	X	M	W										
or x8, x9, x10		F	D	X	M	W									
xor x14, x15, x16			F	D	X	M	W								
sub x7, x5, x2				F	D	X	M	W							

F = Fetch
D = Decode
X = Execute
M = Memory
W = Writeback

Question from last time: Why can Decode & WriteBack happen in the same clock cycle?



The bold/dark parts are showing when the stage is actively using its inputs / producing outputs within the cycle.

Light parts are when we access memory.

Instruction

Cycle #

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add x20, x21, x22	F	D	X	M	W										
sub x28, x20, x15		F	D	D	D	X	M	W							

2 stalls

RISC-V design:
Register file **writes** in the **first half** of the cycle.
Allows for **reads** in **second half**

Performance Metric: CPI (Cycles per instruction)

$$CPI = \frac{total\ number\ of\ cycles\ to\ execute\ program}{number\ of\ instructions\ in\ program}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add x20, x21, x22	F	D	X	M	W										
sub x28, x20, x15		F	D	D	D	X	M	W							
and x27, x5, x18			F	F	F	D	X	M	W						
or x9, x28, x3						F	D	D	X	M	W				

$$CPI = \frac{11}{4} = 2.75$$

Performance Metric: CPI (Cycles per instruction)

$$CPI = \frac{total\ number\ of\ cycles\ to\ execute\ program}{number\ of\ instructions\ in\ program}$$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add x20, x21, x22	F	D	X	M	W										
sub x28, x20, x15		F	D	D	D	X	M	W							
and x27, x5, x18			F	F	F	D	X	M	W						
or x9, x28, x3						F	D	D	X	M	W				

$$CPI = \frac{11}{4} = 2.75$$

Ideal CPI (Cycles per instruction) = 1

$$CPI = \frac{\#\ of\ cycles}{\#\ of\ instructions}$$

Ideal CPI (Cycles per instruction) = 1

In the ideal case, we will have one instruction finishing on every cycle.
If one instruction finishes every cycle, the total number of cycles needed to execute the program is (4 + # of instructions).

$$CPI = \frac{\#\ of\ cycles}{\#\ of\ instructions} = \frac{4+\#\ of\ instructions}{\#\ of\ instruciotns}$$

(this equation assumes no stalls)

Stalls are going to hurt our CPI. We are going to see later in the lecture how to reduce the number of stalls.

Pipeline Diagram with Stalls



1. Fill out the pipeline diagram for the following set of instructions.
2. What is the CPI?
3. What is the speedup compared to a single-cycle datapath?
   - Assume pipelined clock period = 220ps and single-cycle clock period = 820ps

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
add x3, x2, x2															
or x4, x2, x5															
and x9, x4, x10															
sub x11, x4, x14															
add x11, x15, x18															

Pipeline Diagram with Stalls



Writeback and decode may use the same cycle.

Instruction	0	1	2	3	4	5	6	7	8	9	10
add x3, x2, x2	F	D	X	M	W						
or x4, x2, x5		F	D	X	M	W					
and x9, x4, x10			F	D	D	D	X	M	W		
sub x11, x4, x14				F	F	F	D	X	M	W	
add x11, x15, x18							F	D	X	M	W

F = Fetch D = Decode X = Execute M = Memory W = Writeback

The and waits for x4. The following instructions wait behind it.

CPI and Speedup



- 5 instructions + 4 cycles to fill/drain + 2 stalls = 11 cycles
- CPI = 11 / 5 = 2.2
- Pipelined time = 11 × 220 ps = 2420 ps
- Single-cycle time = 5 × 820 ps = 4100 ps
- Speedup = 4100 / 2420 ≈ 1.69

Do We Really Have to Wait for Writeback?



Writeback and decode may use the same cycle.

Instruction	0	1	2	3	4	5	6	7	8	9	10
add x3, x2, x2	F	D	X	M	W						
or x4, x2, x5		F	D	X	M	W					
and x9, x4, x10			F	D	D	D	X	M	W		
				F	F	F	D	X	M	W	
							F	D	X	M	W

Memory W = Writeback

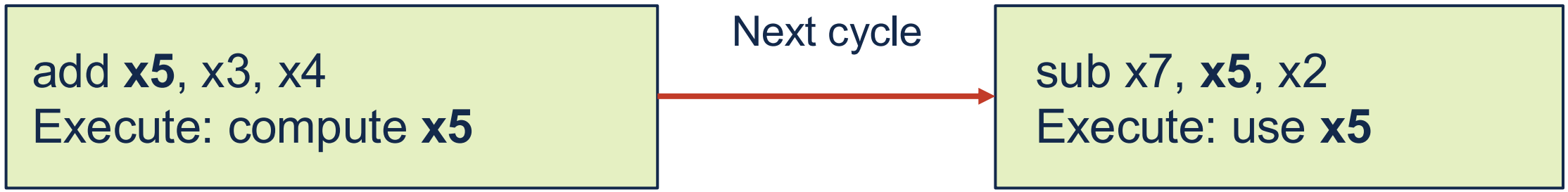
wing instructions wait behind it.

Question:
When does add x5,x3,x4 calculate x5?

Forwarding



Send a result directly to the instruction that needs it.



Also called bypassing: the value bypasses the register file.

Forwarding: No Stall Needed (!)



The result from add is available for sub's Execute stage.

Instruction	0	1	2	3	4	5
add x5, x3, x4	F	D	X	M	W	
sub x7, x5, x2		F	D	X	M	W

F = Fetch D = Decode X = Execute M = Memory W = Writeback

Forward from EX/MEM to the ALU input.

[The register file may still contain the old value of x5 during decode.]

Forwarding to the ALU

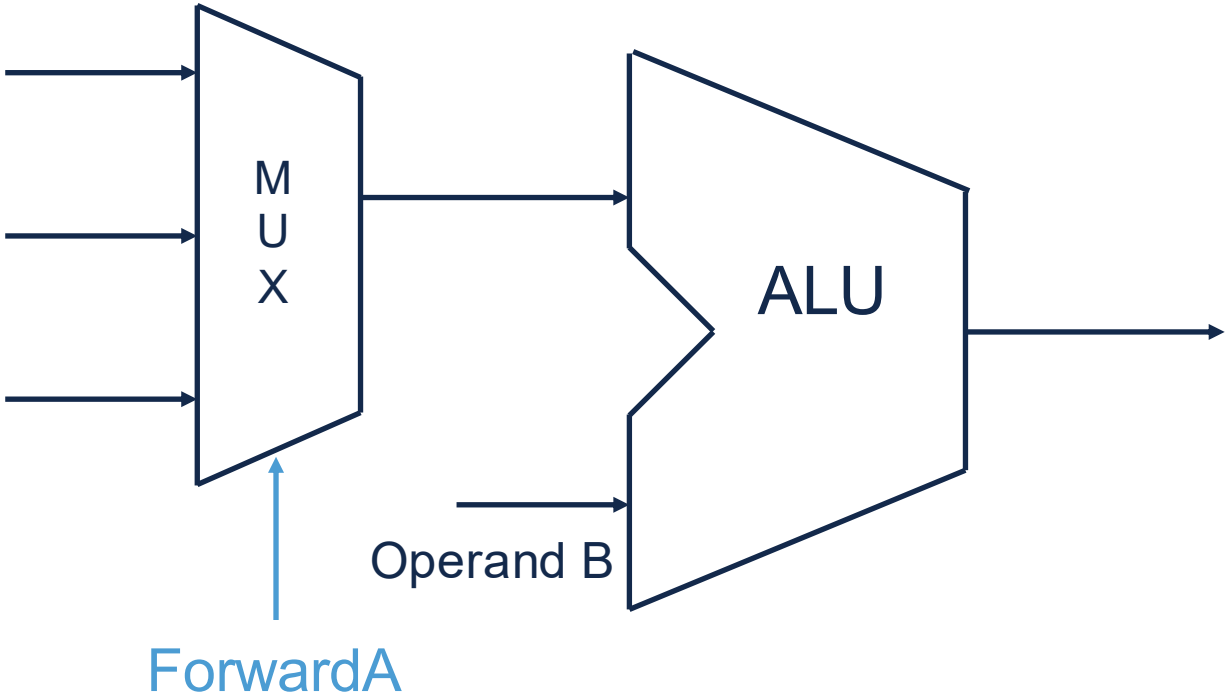


A mux selects the value used by each ALU operand.

ID/EX: register value
(no forwarding)

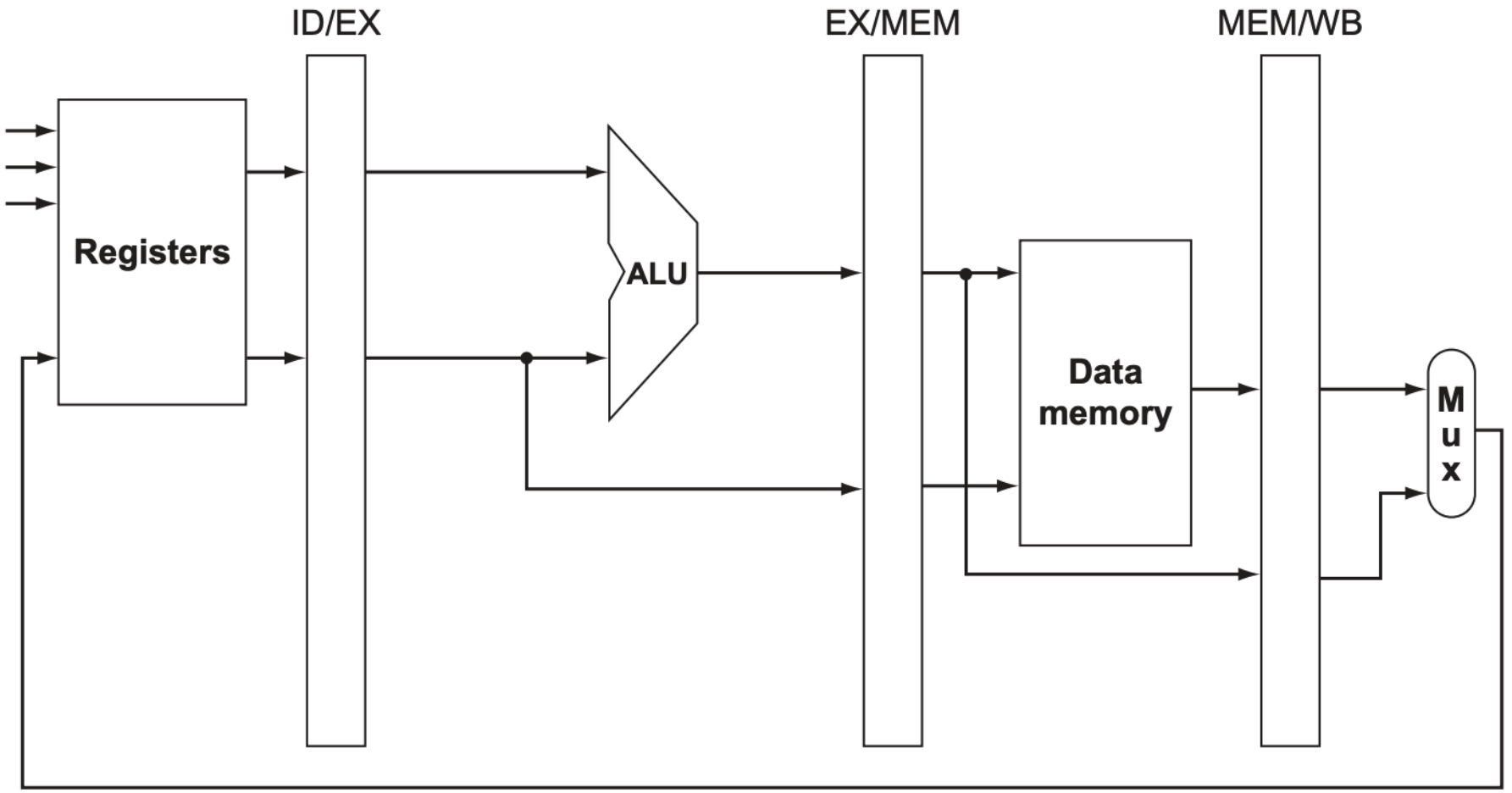
EX/MEM: ALU result
(result from previous instruction)

MEM/WB: writeback value
(result from two instructions ago)

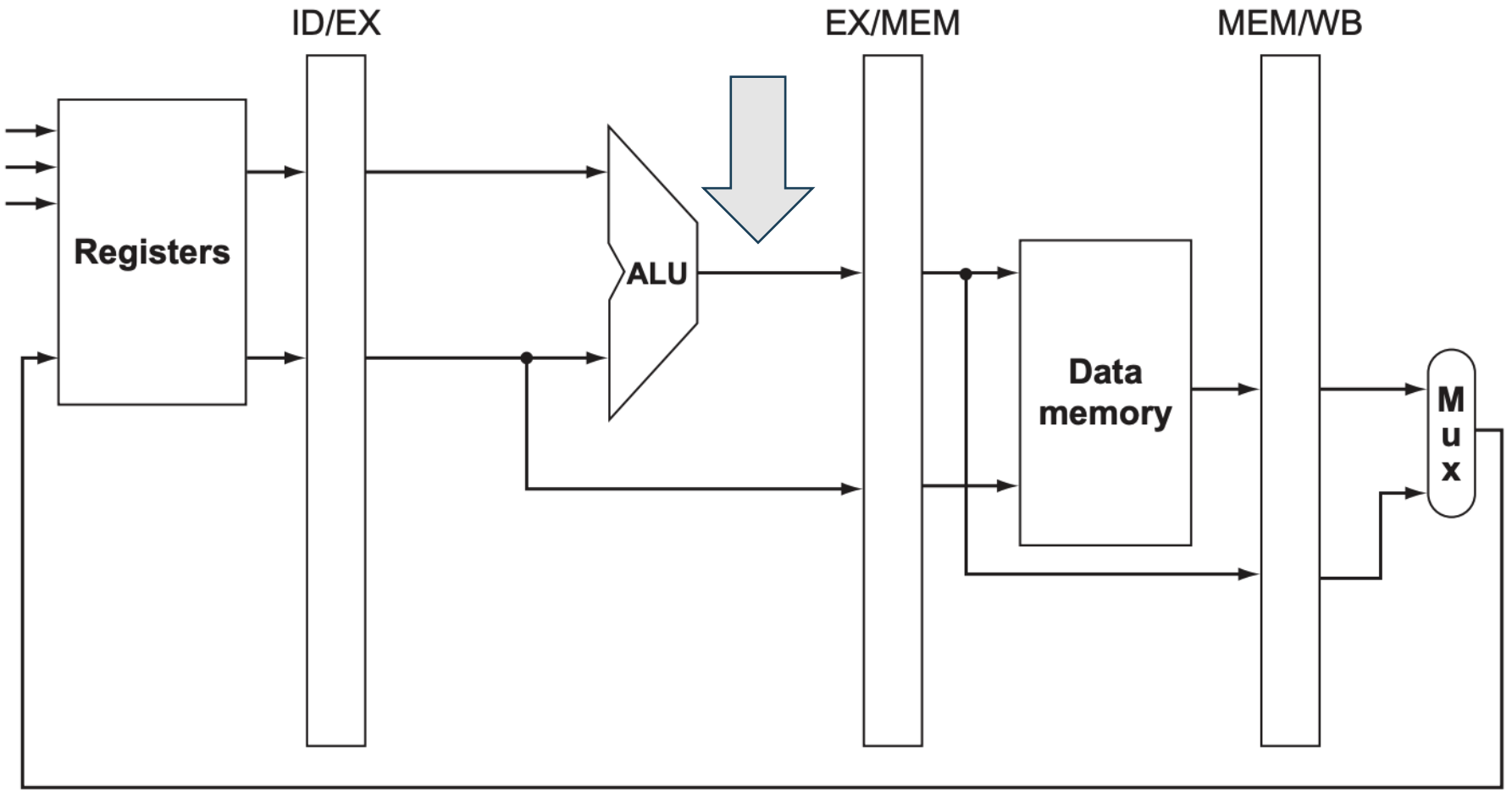


We will apply the same ideas to Operand B in a moment.

Simplified datapath to focus on forwarding for r-format



The value that is to be written to x5 is available at the ALU output

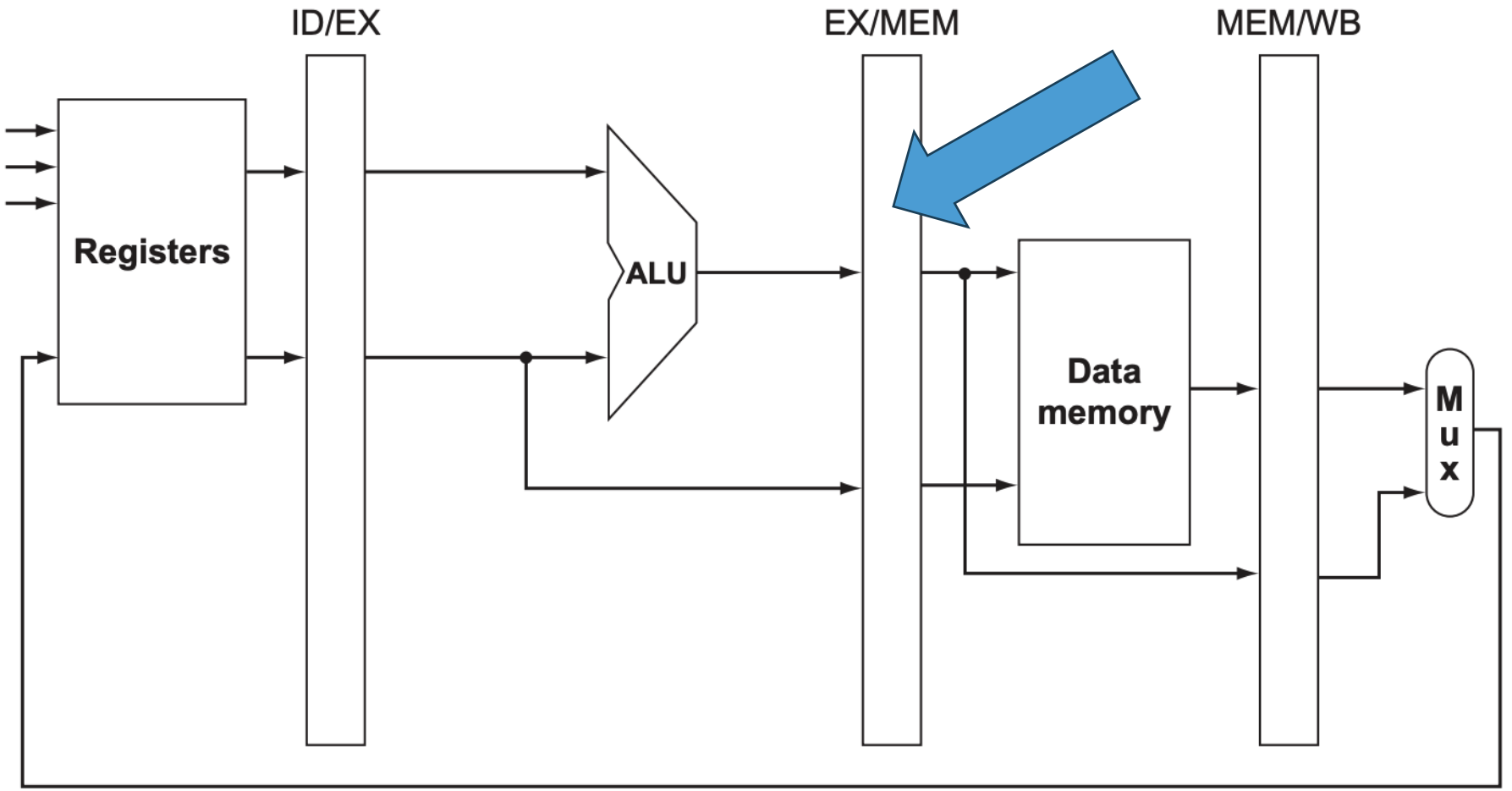


sub x7, x5, x2

add x5, x3, x4

Forwarding rs1

The value to be written to x5 is here

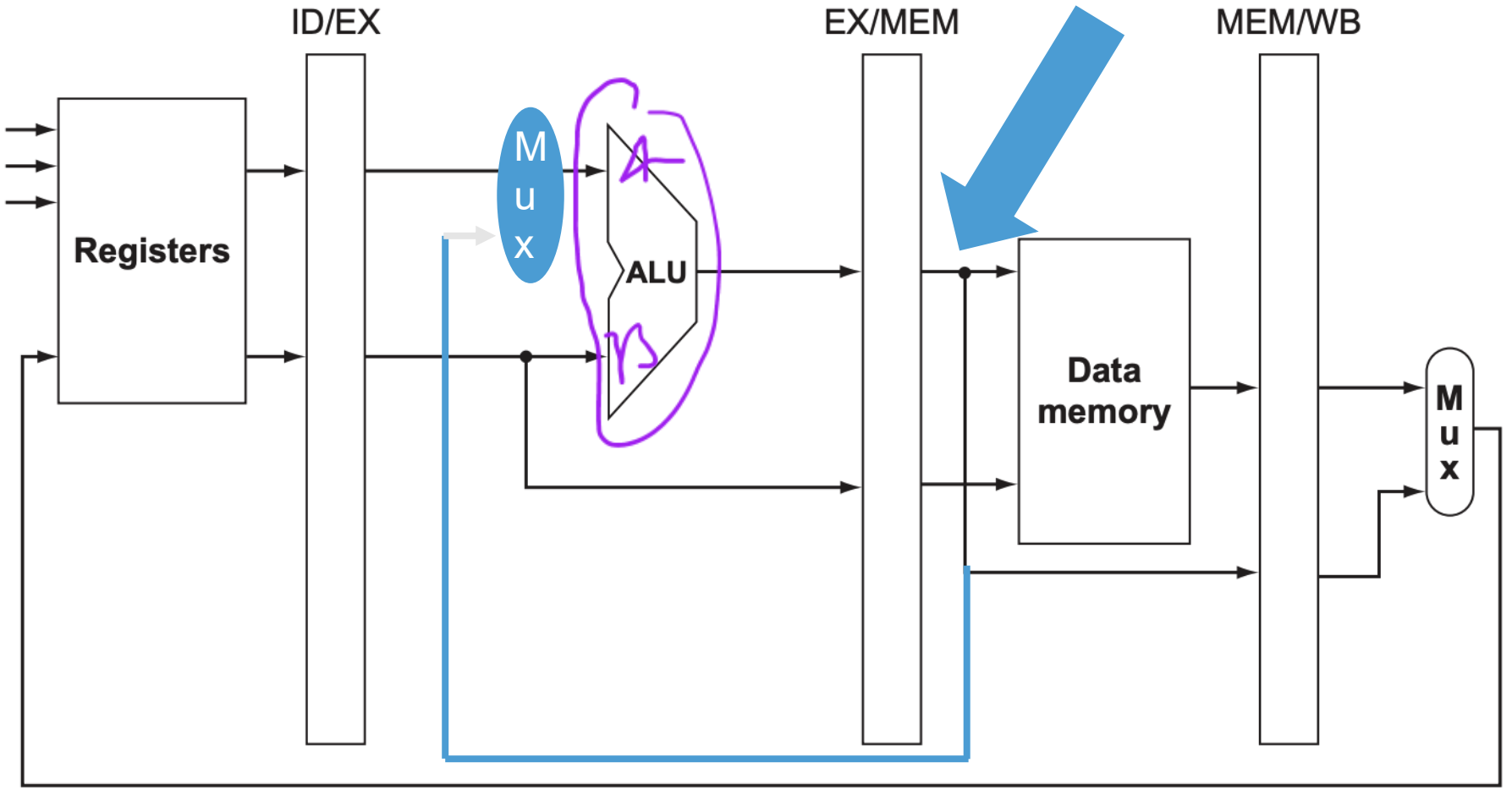


sub x7, x5, x2

add x5, x3, x4

Forwarding rs1

We are forwarding the value to be written to x5 to the dependent instruction.

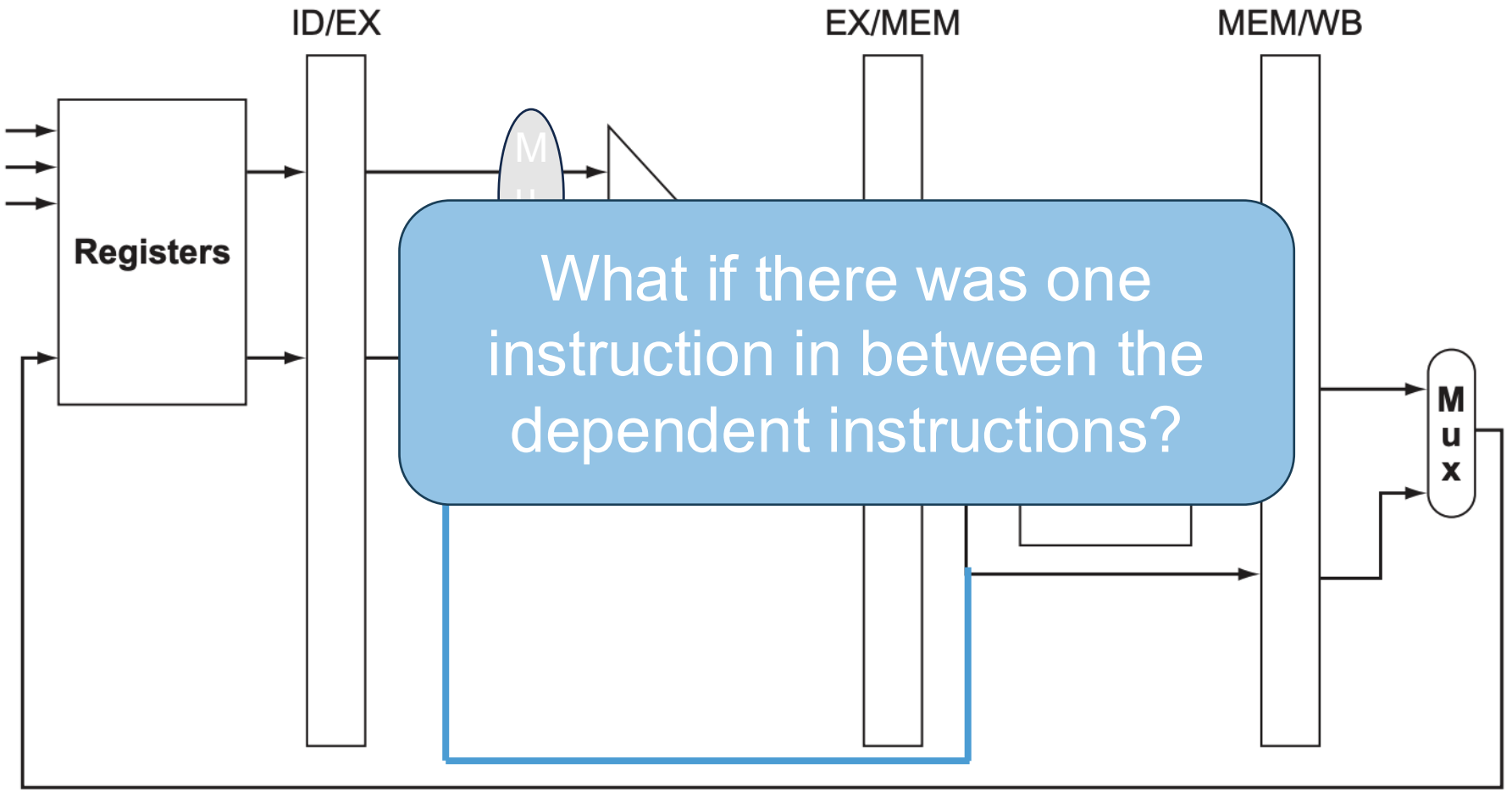


We add “Bypass MUXes”

sub x7, x5, x2

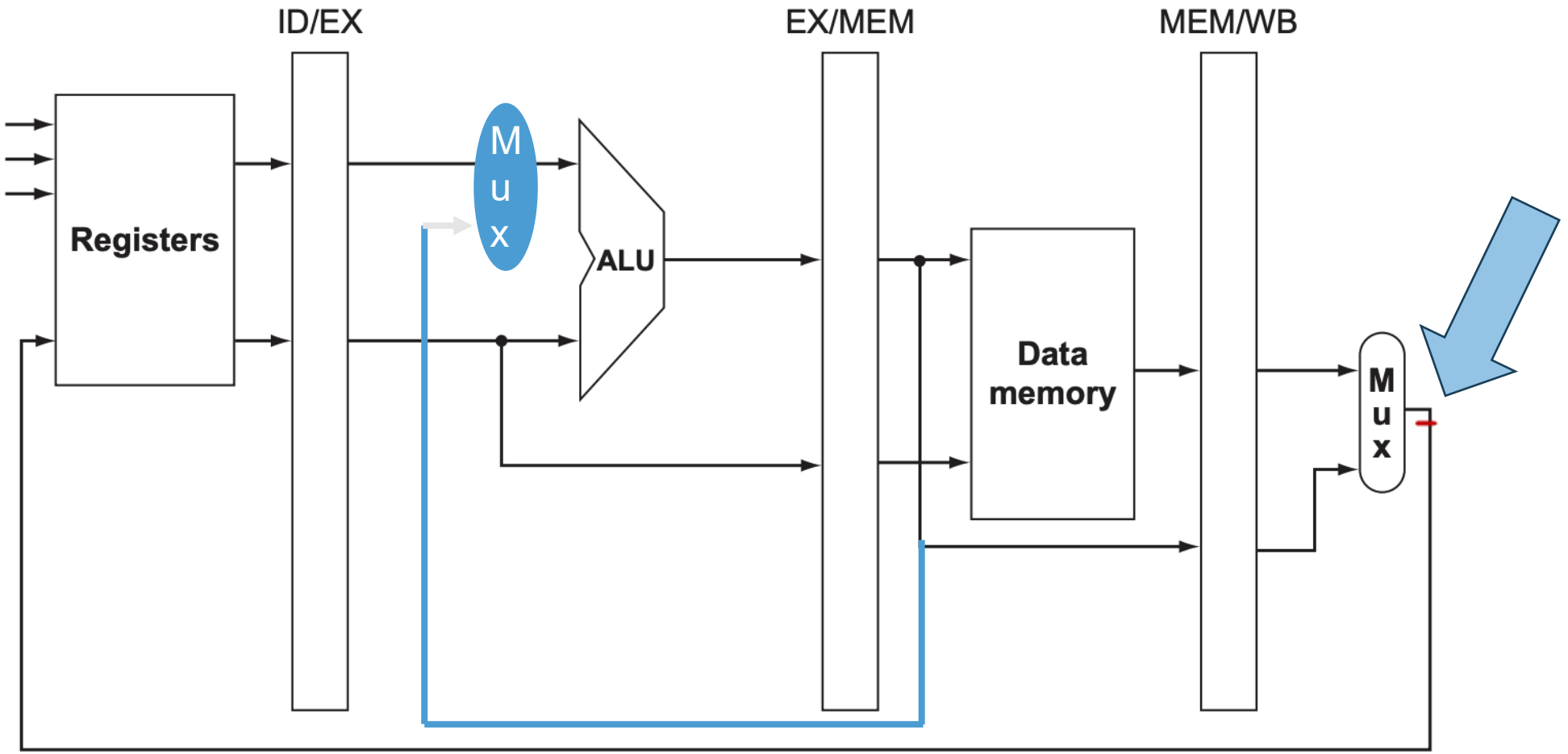
add x5, x3, x4

Forwarding rs1



Forwarding rs1

The value to be written to x5 is at the output of the writeback mux.

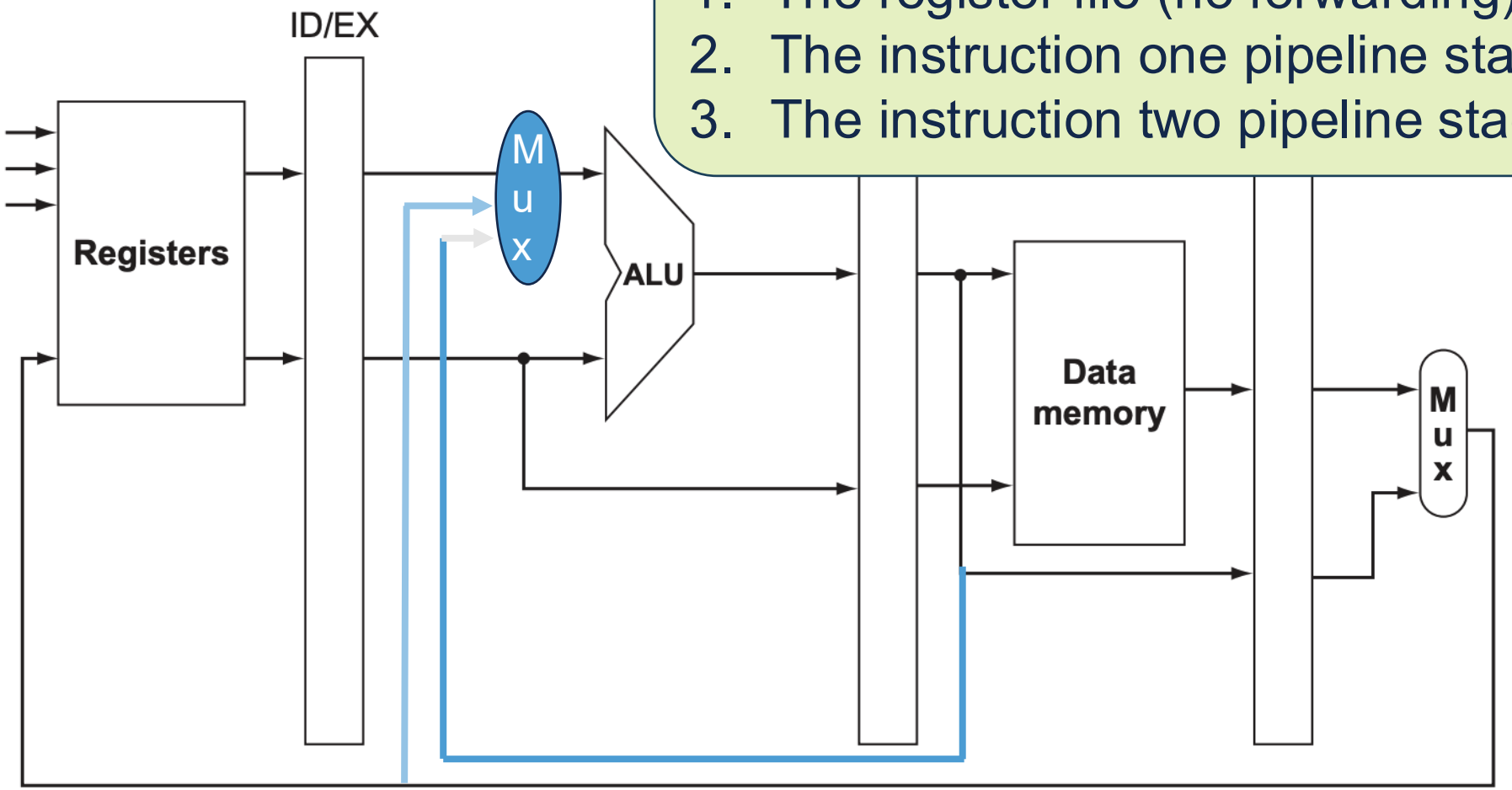


sub x7, x5, x2 or x8, x9, x10 add x5, x3, x4

Forwarding rs1

Now, we can choose if the top input of the ALU should come from

1. The register file (no forwarding)
2. The instruction one pipeline stage ahead
3. The instruction two pipeline stage ahead



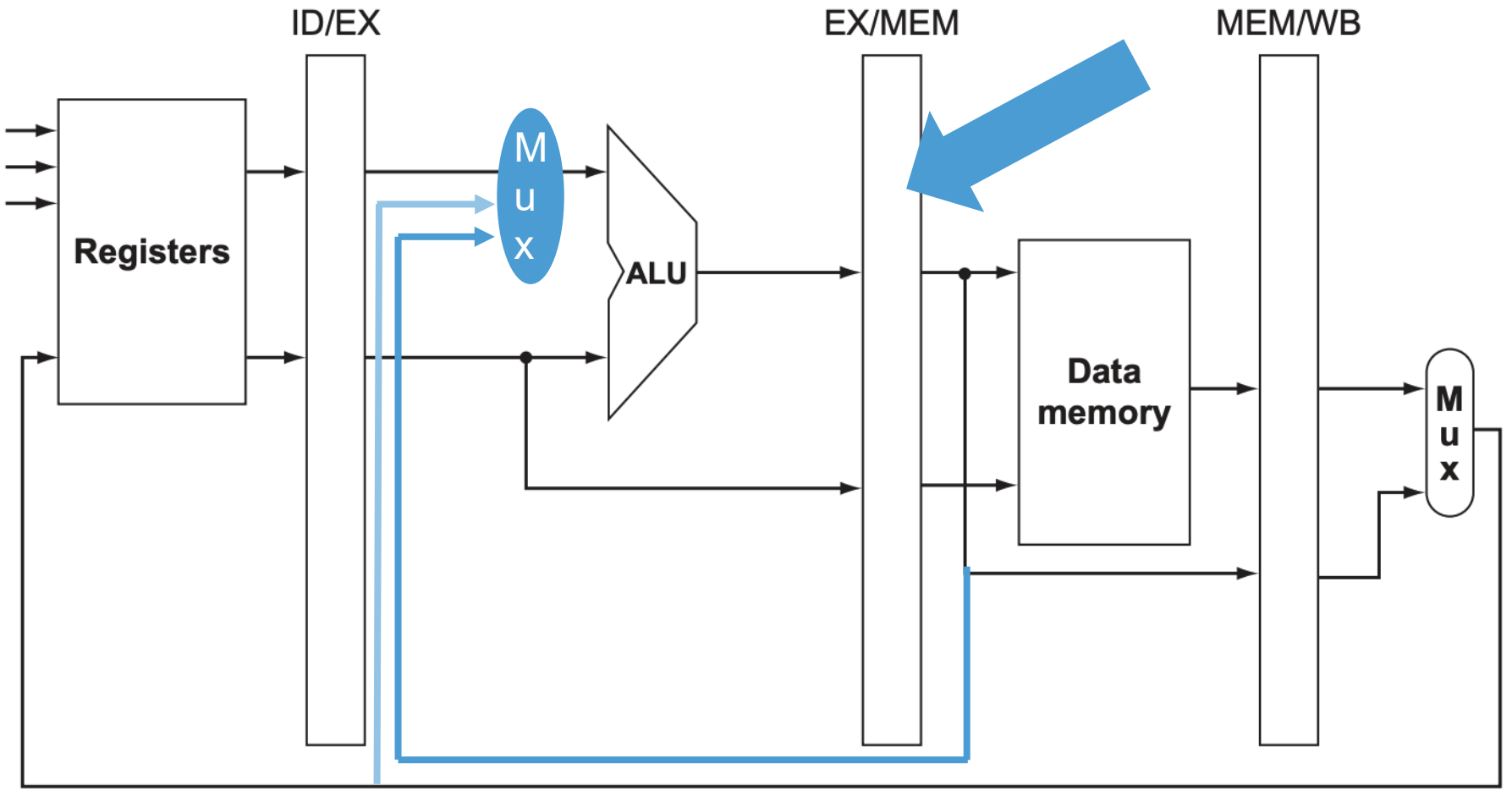
sub x7, x5, x2

or x8, x9, x10

add x5, x3, x4

Forwarding rs2

The value to be written to x5 is here

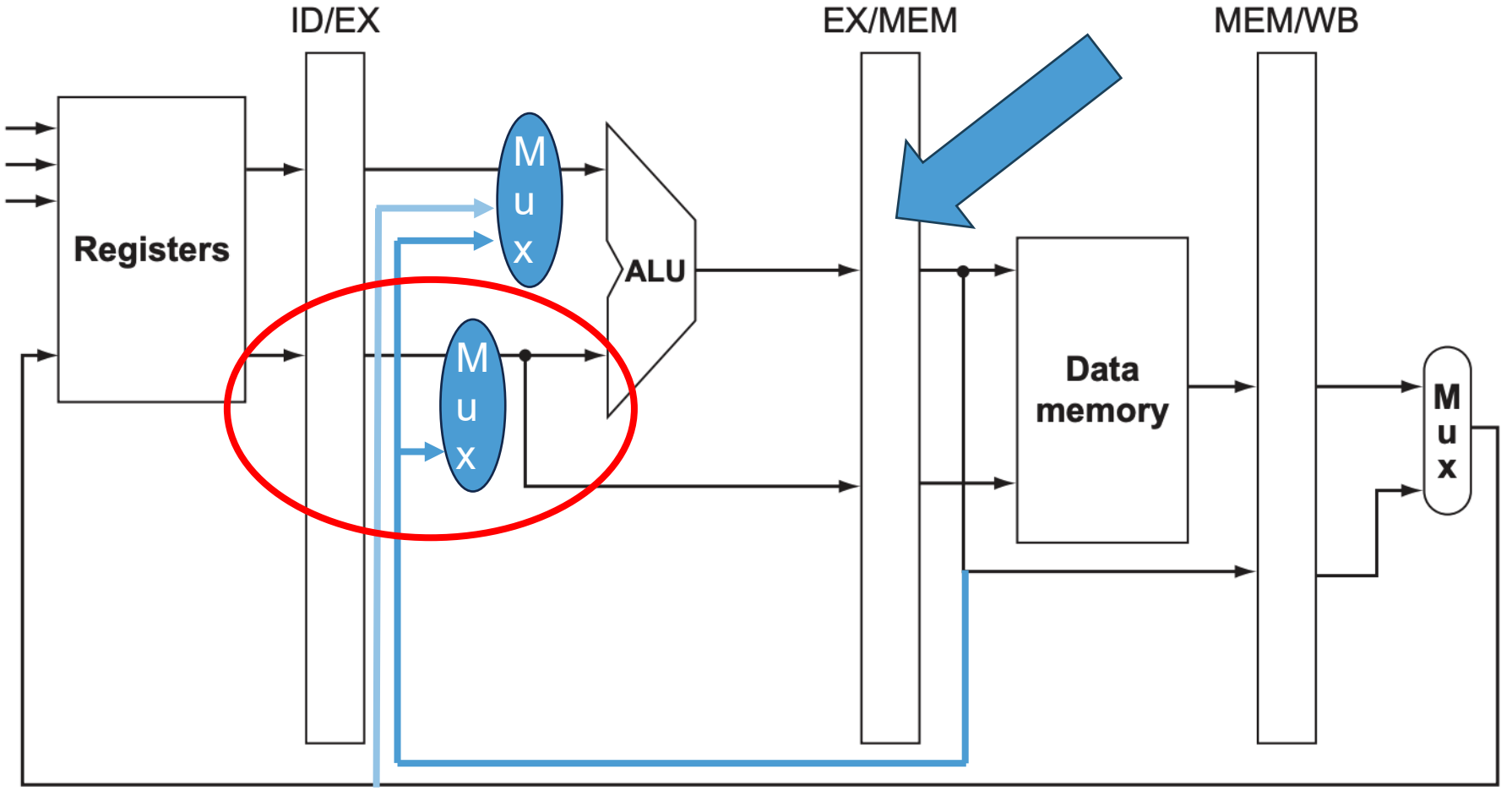


sub x7, x2, x5

add x5 x3, x4

Forwarding rs2

We are forwarding the value to be written to x5 to the dependent instruction.



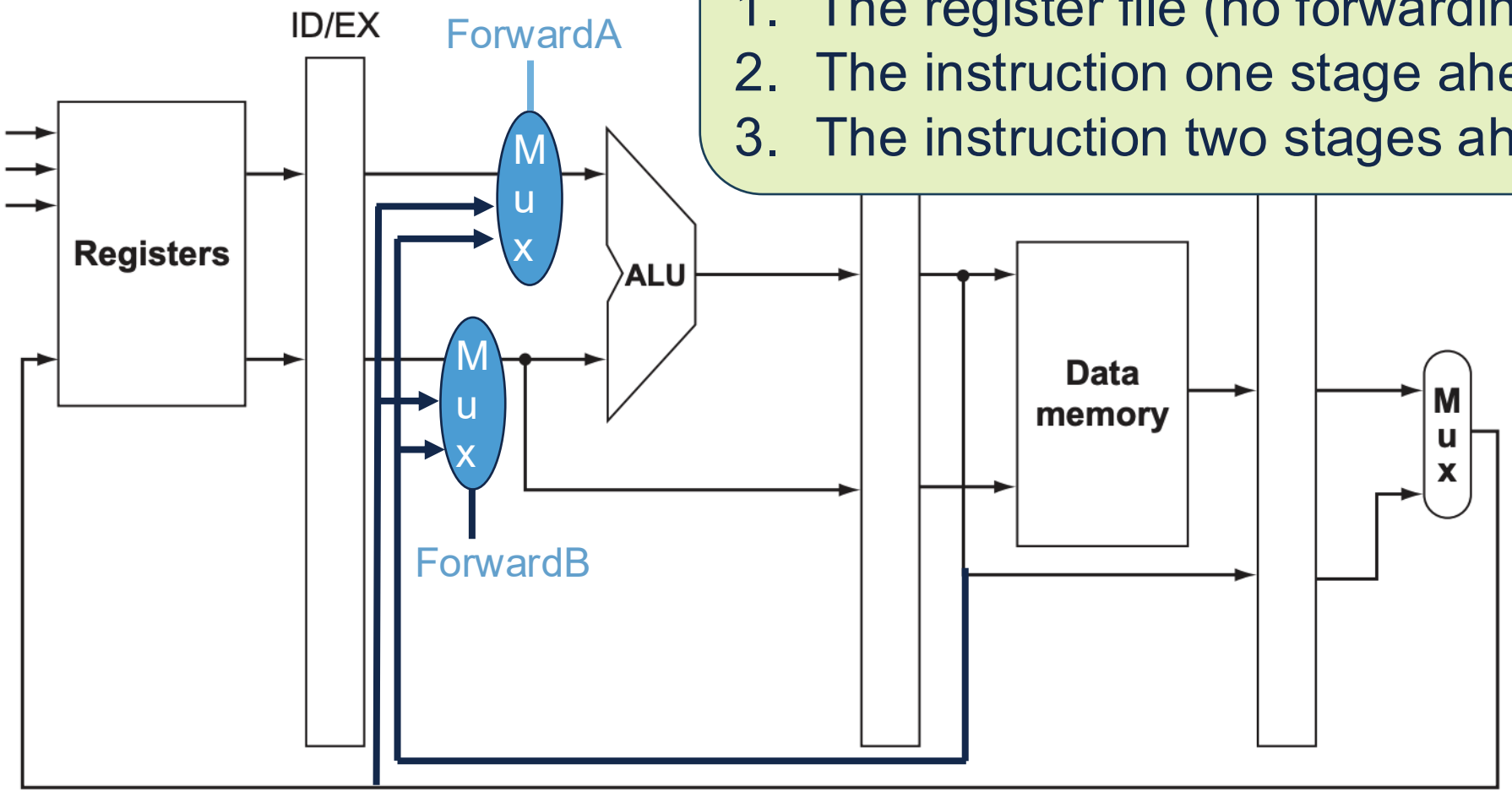
sub x7, x2, x5

add x5 x3, x4

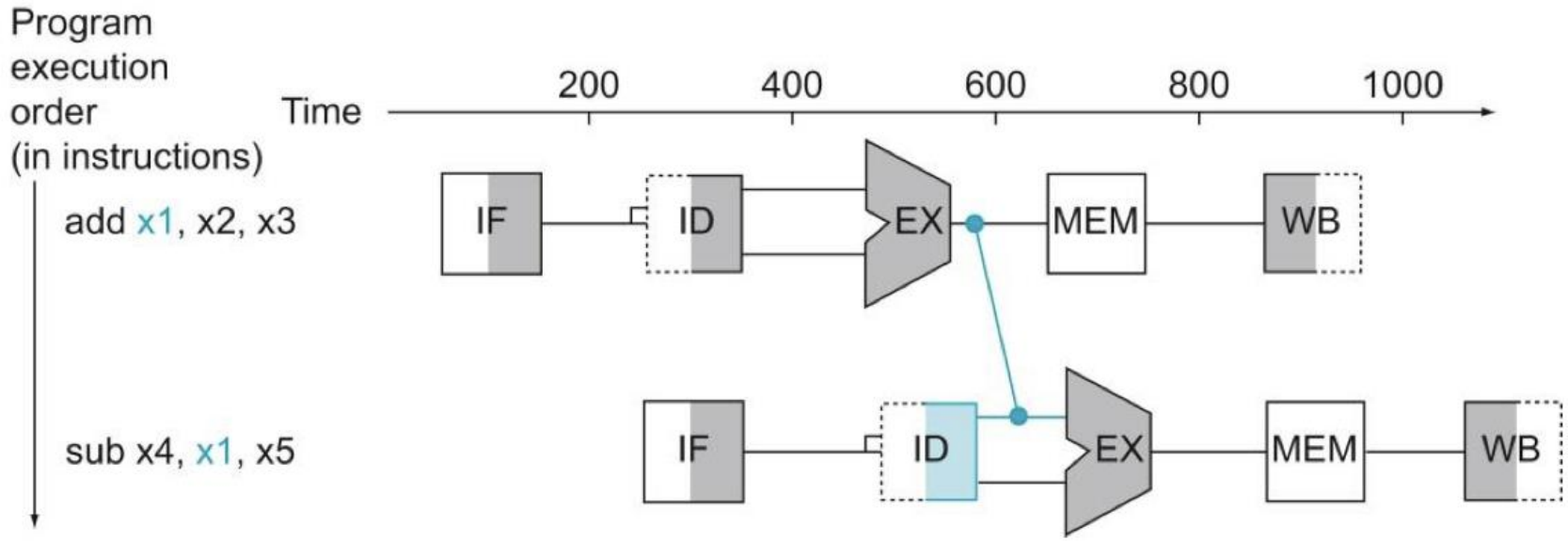
Source Bypass Muxes: 3 c

Now, we can choose if the bottom input of the ALU should come from

1. The register file (no forwarding) -- 00
2. The instruction one stage ahead – 10
3. The instruction two stages ahead -- 10



Graphical Representation of Forwarding



Source: https://csweb.wooster.edu/jmusgrave/cs210/Chapter_04.pdf

Which Operand Needs Forwarding?



Instruction pair	Dependent operand
add x5, x3, x4 sub x7, x5, x2	
add x5, x3, x4 sub x7, x2, x5	

Check both source registers against the destination register.

Which Operand Needs Forwarding?



Instruction pair	Dependent operand
add **x5**, x3, x4 sub x7, **x5**, x2	A (rs1)
add **x5**, x3, x4 sub x7, x2, **x5**	B (rs2)

Check both source registers against the destination register.

Forwarding Controls



Select	ALU input comes from
00	ID/EX: value read from the register file
10	EX/MEM: result from the previous instruction
01	MEM/WB: value being written back

ForwardA and ForwardB select the two operands independently.

Practice: ForwardA and ForwardB



For each sub in Execute, choose ForwardA and ForwardB.

Instruction sequence	ForwardA	ForwardB
add x5, x3, x4 sub x7, x5, x2		
add x5, x3, x4 sub x7, x2, x5		

add x5, x3, x4
or x8, x9, x10
sub x7, x5, x2		

10: EX/MEM 01: MEM/WB 00: register value

ForwardA and ForwardB



Instruction sequence	ForwardA	ForwardB
add x5, x3, x4 sub x7, x5, x2	10	00
add x5, x3, x4 sub x7, x2, x5	00	10

add x5, x3, x4
or x8, x9, x10
sub x7, x5, x2
	01	00

10: EX/MEM 01: MEM/WB 00: register value

What About Loads?

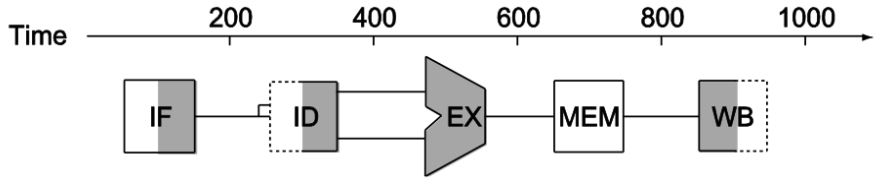


```
lw x4, 0(x9)
add x5, x4, x3
```

Execute	Memory	Writeback
Compute address	Read the loaded value	Write the register

The load produces its data later than an arithmetic instruction.

Load-Use Hazard



What happens if we do not stall?

Instruction	0	1	2	3	4	5
lw x4, 0(x9)	F	D	X	M	W	
add x5, x4, x3		F	D	X	M	W

F = Fetch D = Decode X = Execute M = Memory W = Writeback

In cycle 3, add needs x4 at the start of Execute.
The load's data is ready at the *end* of Memory.

Forwarding cannot send a value backward in time.

Load-Use Hazard: One Stall



Hold add in Decode for one extra cycle.

Instruction	0	1	2	3	4	5	6
lw x4, 0(x9)	F	D	X	M	W		
add x5, x4, x3		F	D	D	X	M	W

F = Fetch D = Decode X = Execute M = Memory W = Writeback

Now add executes in cycle 4.
Forward the loaded value from MEM/WB to the ALU.

Practice: Pipeline with Forwarding



Use forwarding; register writes occur before reads in the same cycle.

Instruction	0	1	2	3	4	5	6	7	8	9
lw x3, 0(x8)										
add x11, x3, x3										
lw x4, 0(x11)										
add x9, x11, x9										
sub x10, x4, x9										

F = Fetch D = Decode X = Execute M = Memory W = Writeback

Mark stalls and forwarding. How many cycles? What is the CPI?

Control Hazards



Q: Which instruction should we fetch after a branch?

```
beq  x1, x2, target
addi x3, x3, 1
addi x4, x4, 1
target:
     sub x5, x6, x7
```

Assume the branch is resolved at the end of Execute.
By then, we have already fetched two more instructions.

Taken Branch: Flush the Wrong Path



Assume we fetch sequentially until the branch is resolved.

Instruction	0	1	2	3	4	5	6	7
beq x1, x2, target	F	D	X					
addi x3, x3, 1		F	D					
addi x4, x4, 1			F					
target: sub x5, x6, x7								

If the target address resolves to **`TAKE_BRANCH`** … We have done two incorrect instructions ☹

F = Fetch D = Decode X = Execute M = Memory W = Writeback

At cycle 2: the branch instruction is in the execute stage! Our branch target address is being calculated.

Question: Do we see a problem here??

Taken Branch: Flush the Wrong Path



Assume we fetch sequentially until the branch is resolved.

Instruction	0	1	2	3	4	5	6	7
beq x1, x2, target	F	D	X	M	W			
addi x3, x3, 1		F	D	–				
addi x4, x4, 1			F	–				
target: sub x5, x6, x7				F	D	X	M	W

F = Fetch D = Decode X = Execute M = Memory W = Writeback

Flush both wrong-path instructions (–). Taken-branch penalty: 2 cycles.

Handling Control Hazards

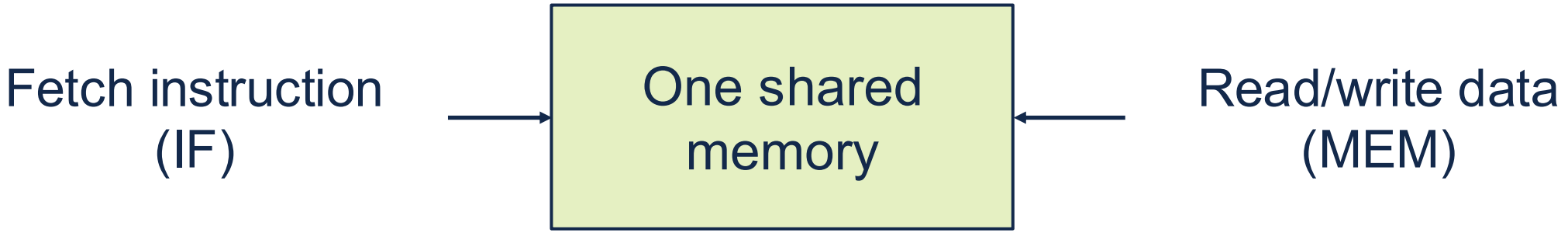


- Determine whether the branch is taken and where it goes.
- Redirect the PC if the fetched path was wrong.
- Flush wrong-path instructions before they change state.
- Branch prediction guesses the path so fetching can continue.

Structural Hazards



Two instructions need the same hardware in the same cycle.



A single memory port cannot serve both requests at once.
Separate **instruction** and **data** memories avoid this conflict!

Three Types of Pipeline Hazards



Hazard	What is the conflict?	Possible response
Data	An operand is not ready.	Forward or stall
Control	The next PC is not yet known.	Try to predict; flush if wrong
Structural	Two instructions need one resource.	Add resources or stall

Pipelining overlaps instructions. Hazards limit that overlap.

Hazards and CPI



For N instructions in our five-stage pipeline:

Cycles = N + 4 + stall cycles + flush penalty cycles

CPI = total cycles / N

Where does the 4 come from in the cycle formula?

Count each lost cycle once.
Forwarding and prediction help reduce the extra cycles.

Practice: CPI with Hazards



- 100 instructions
- 8 load-use stalls (1 cycle each)
- 6 taken branches (2-cycle penalty each)

How many cycles? What is the CPI?

Performance Analysis + Pipelining



Assume a 1 GHz clock. One clock cycle is 1 ns

1×10^{-9}

- Assume a pipelined implementation where:
  - Taken branches take 2 cycles.
  - Loads and stores take 2 cycles.
- Assuming approximately 10% of instructions executed are branches, and of those 80% of the time they are taken, and 15% of instructions executed are loads or stores, what sort of speed-up do we expect compared to an ideal pipeline? What about unpipelined?

Performance Analysis + Pipelining



- Assume a pipelined implementation where:
  - Taken branches take 2 cycles.
  - Loads and stores take 2 cycles.
- Assuming approximately 10% of instructions executed are branches, and of those 80% of the time they are taken, and 15% of instructions executed are loads or stores, what sort of speed-up do we expect?

Performance Analysis + Pipelining



Some takeaways…



- Design strategy: Focus our attention on placing pipelining registers around the slowest circuit elements (bottlenecks!)
- You can pipeline an RISC-V CPU even more. There exist implementations with 7, 8, and 9 pipeline stages. But the overhead of bypass paths and the number of stall cases increase .
- Memory access time is our real bottleneck!