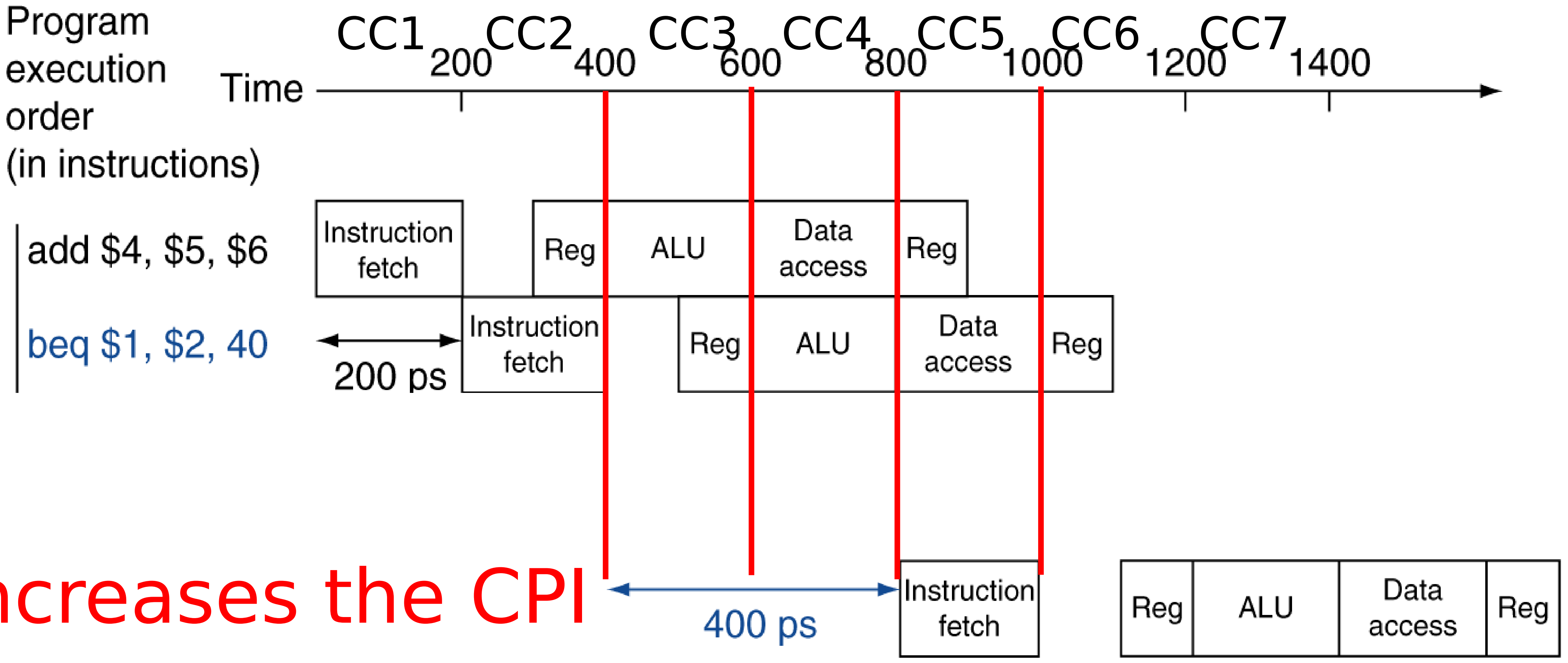
Where are we?

- We have explored a single clock cycle processor design
- We have introduced pipelining and established its performance benefits
- We have explored pipelining hazards and identified solutions

Stall on Branch

- Wait until branch outcome is determined before fetching next instruction

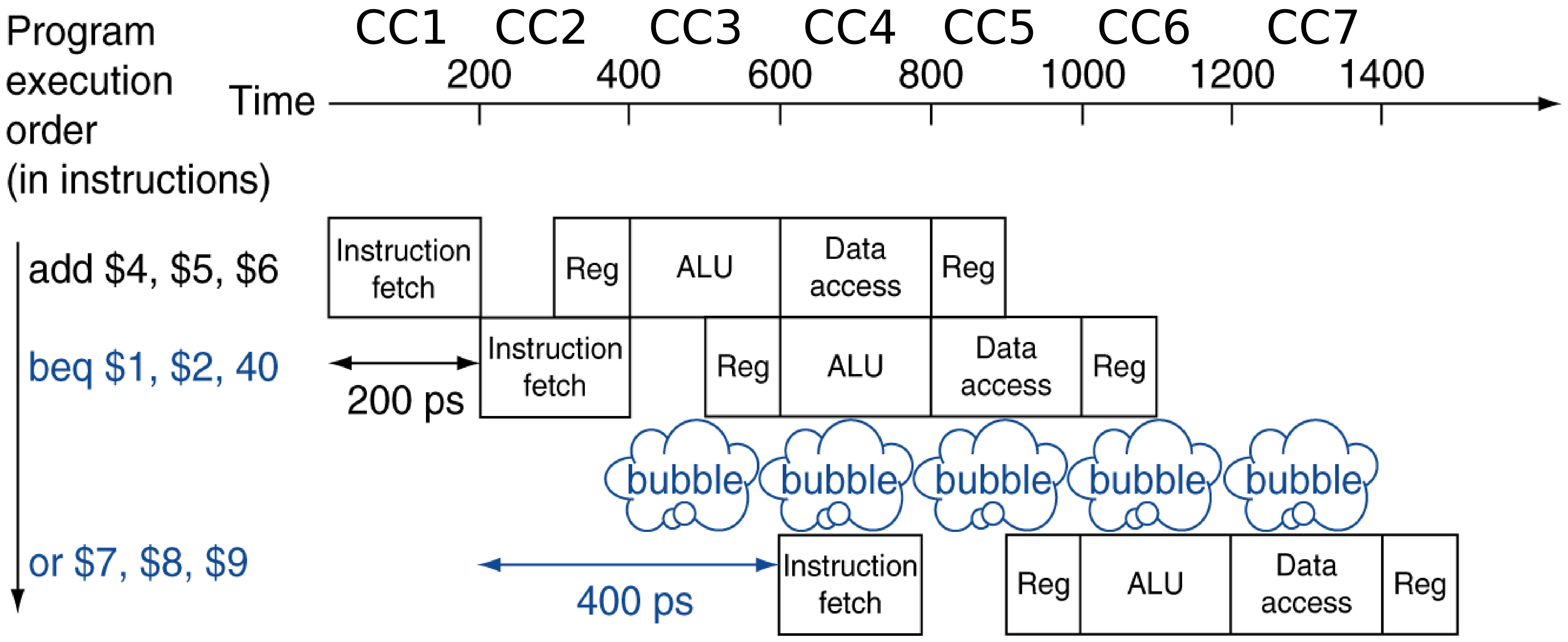


Increases the CPI

Optimized Stall on Branch

Optimized handling using additional hardware

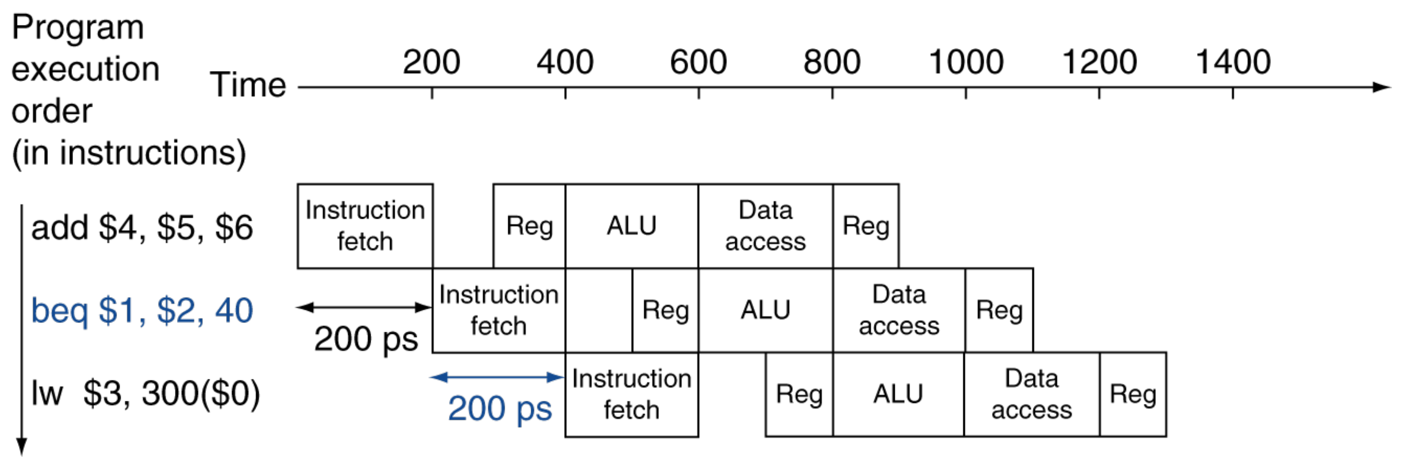
- Decode and decide



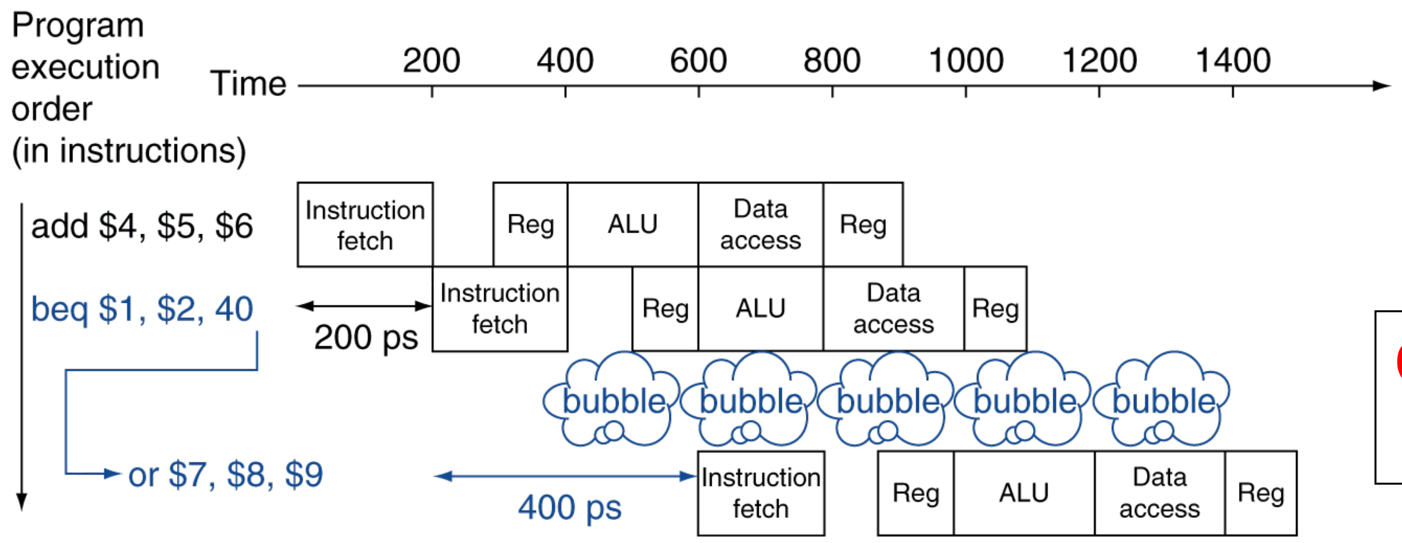
Branch Prediction

- Usually adopted because Stall penalty becomes unacceptable, ***<u>especially for Longer pipelines</u>***
- Predict outcome of branch
  - Only stall if prediction is wrong
  - Accurate prediction → high performance
- In MIPS pipeline
  - Can predict branches not taken

Simple Policy: Predict Branch Not Taken



Prediction correct



Prediction incorrect

Optimized hardware –> only one bubble

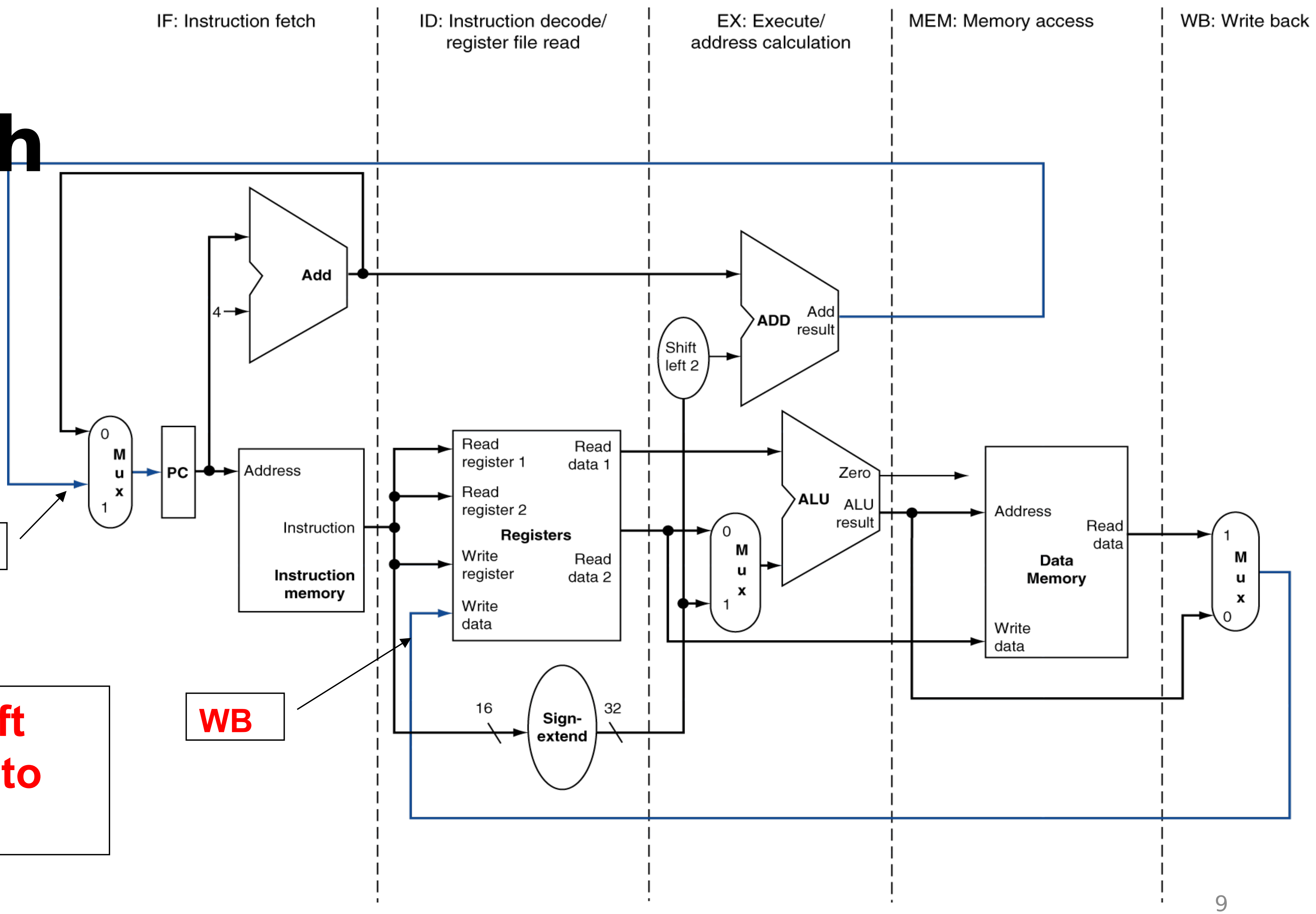
More-Realistic Branch Prediction

- Static branch prediction
    - Based on typical branch behaviour
    - Example: loop and if-statement branches
        - Predict backward branches taken
        - Predict forward branches not taken
- Dynamic branch prediction
    - Hardware measures actual branch behaviour
        - e.g., record recent history of each branch
    - Assume future behaviour will continue the trend

Pipelined Datapath Hardware

- Instructions divided into five stages
  - IF: Instruction fetch
  - ID: Instruction decode and register file read
  - EX: Execution or address calculation
  - MEM: Data memory access
  - WB: Write back
- Dividing instruction into five stages mean five-stage pipeline
  - Up to five instructions could be in execution during any single clock cycle
  - Instruction and data move from left to right through the stages
- Two exceptions to this left-to-right flow of instructions
  - The write back stage
  - Branching

MIPS Datapath

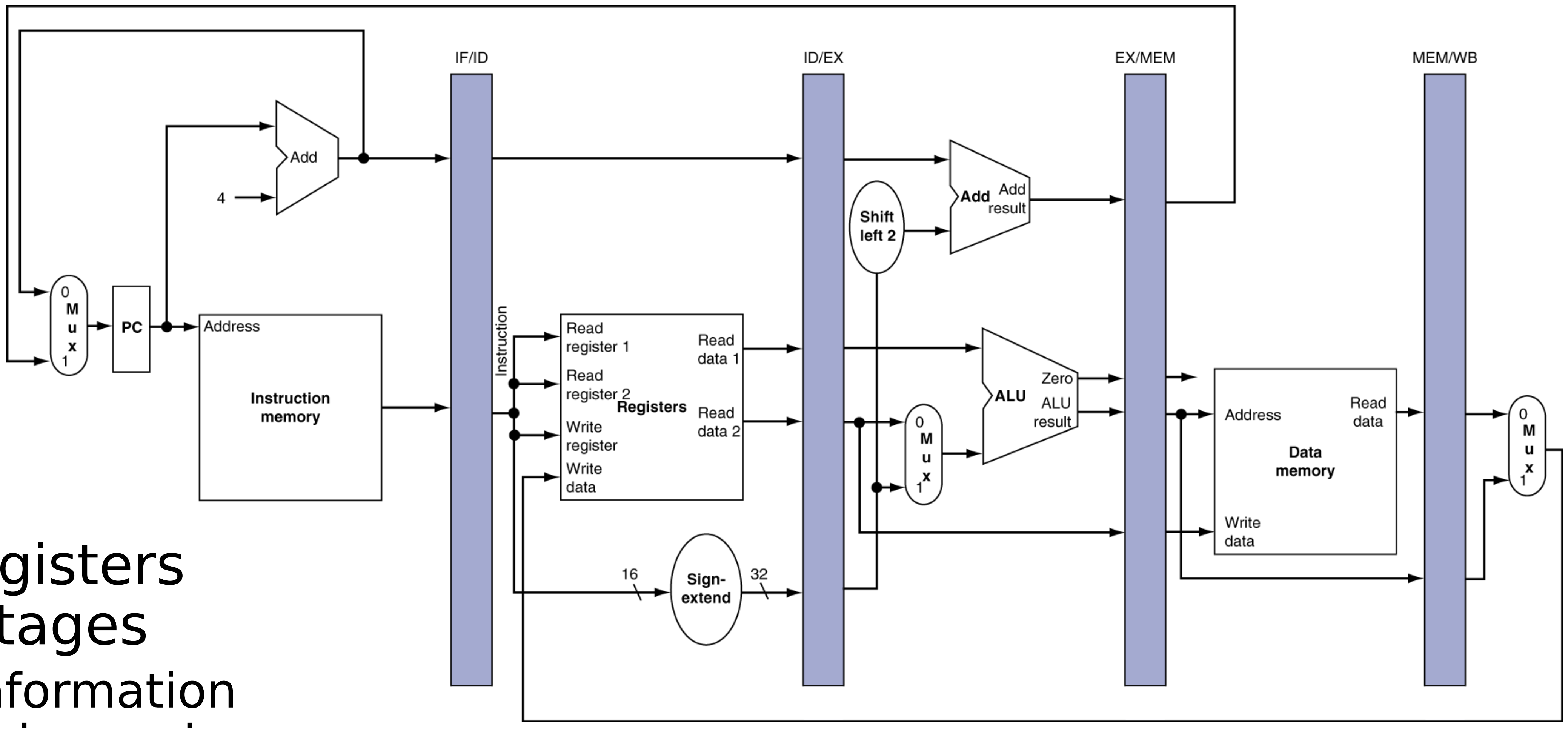


MEM

WB

Right-to-left flow leads to hazards

MIPS Pipelined Datapath

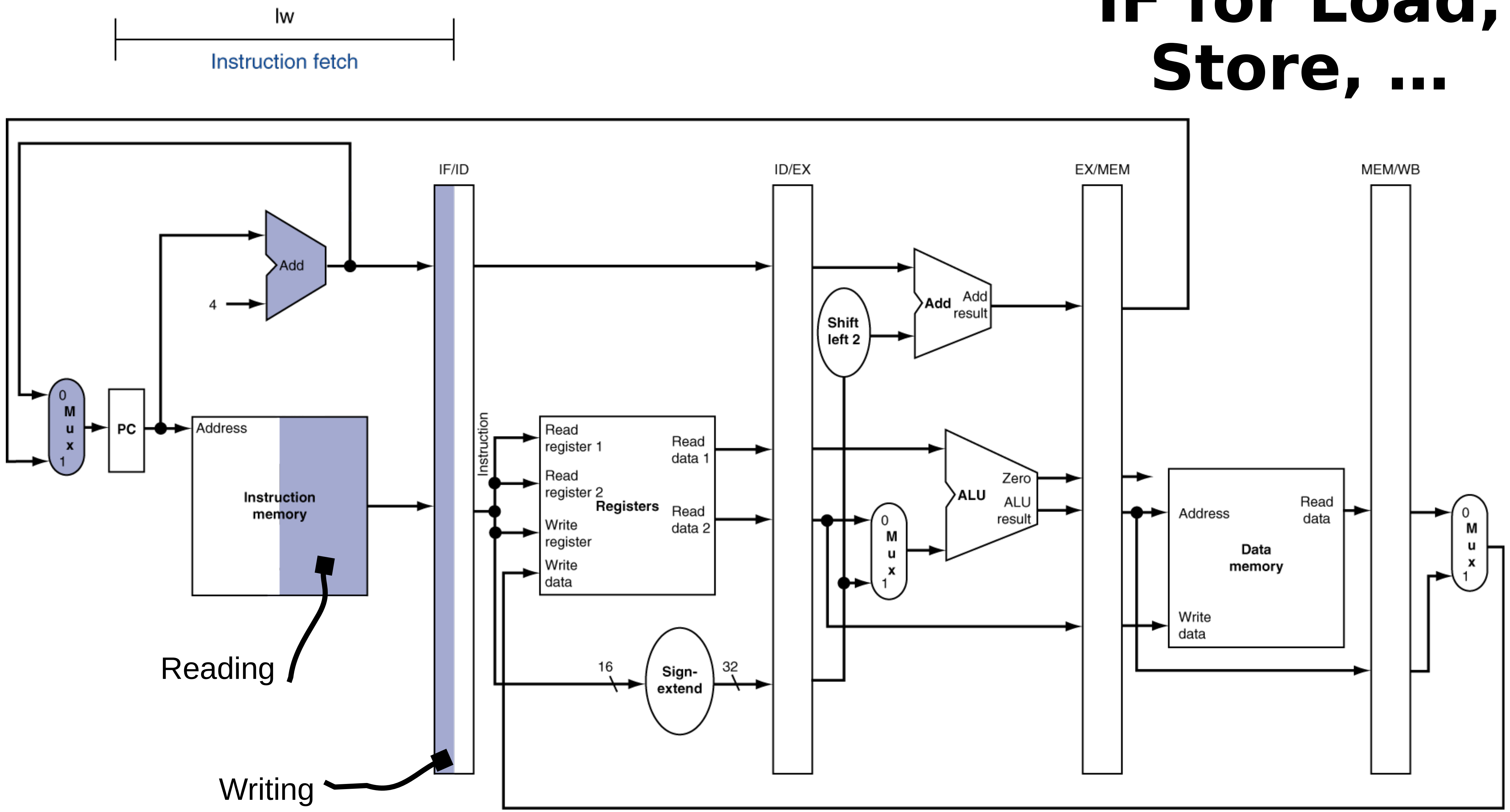


- Pipeline registers between stages
  - To hold information produced in previous cycle

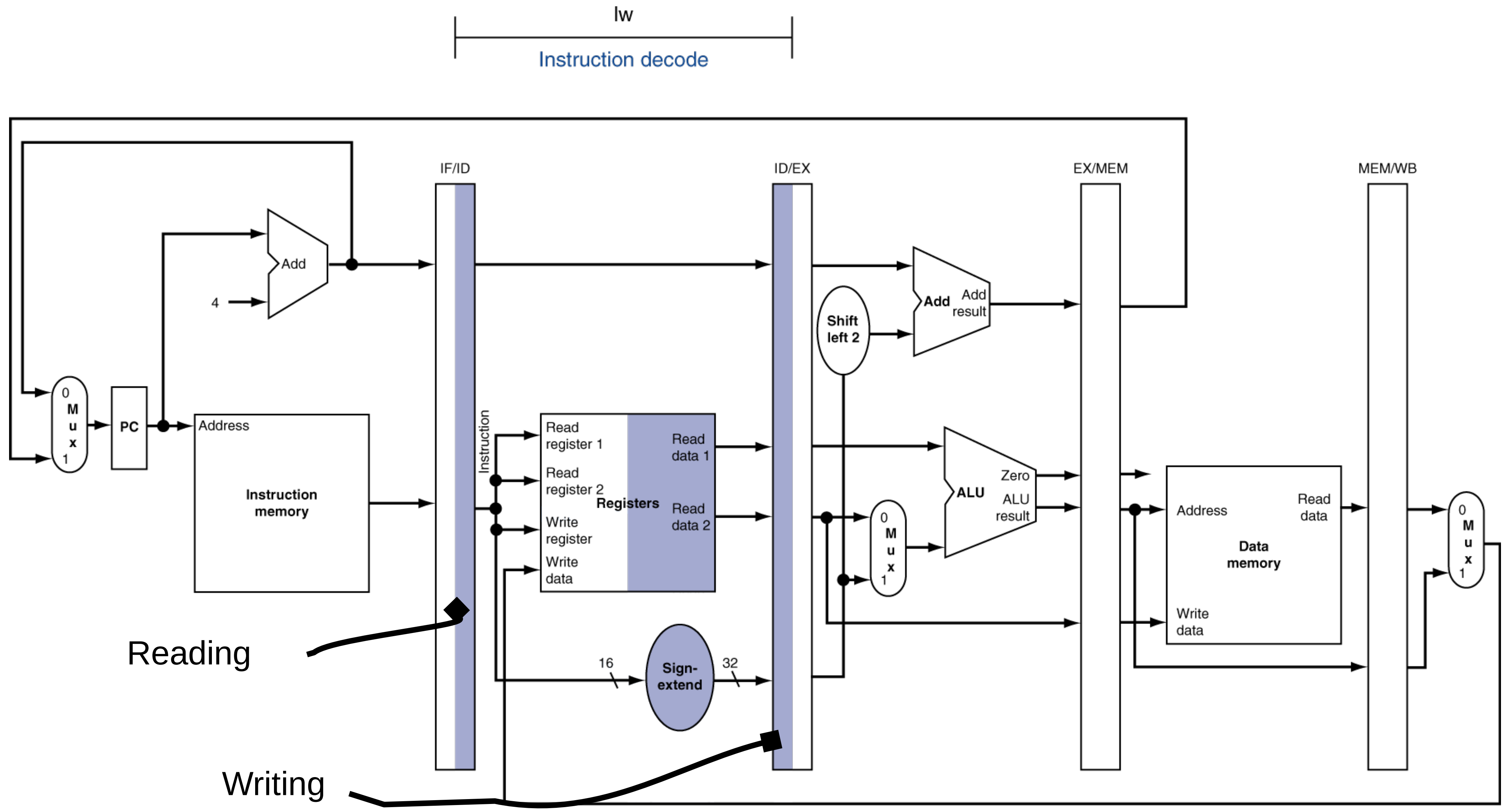
Pipeline Operation Example

- We’ll look at “single-clock-cycle” diagrams for load
  - We highlight the right half of register or memory when read operation
  - Highlight left half when they are being written

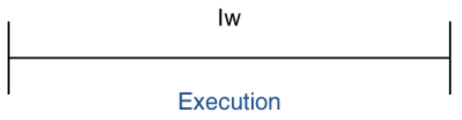
IF for Load, Store, …

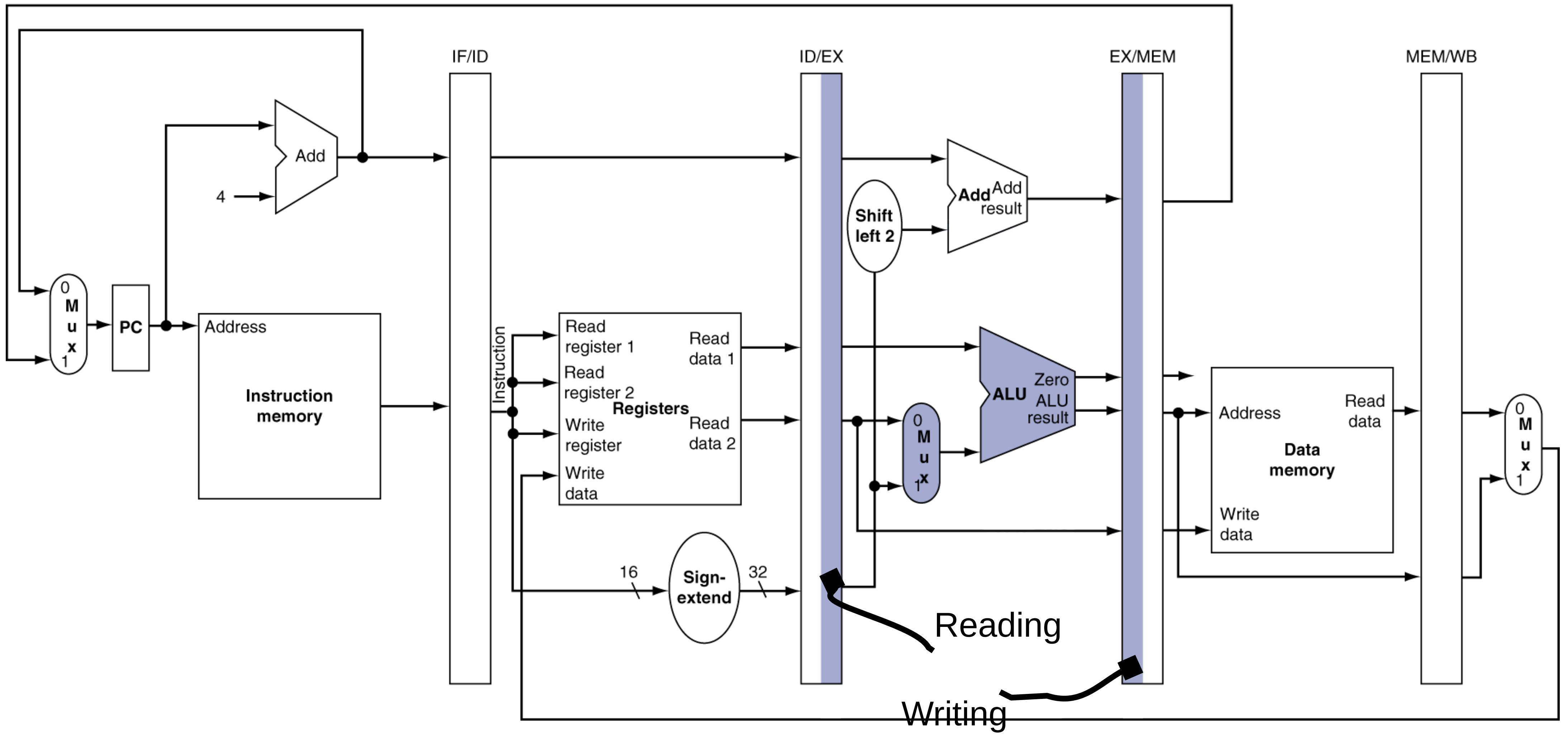


ID for Load, Store, …

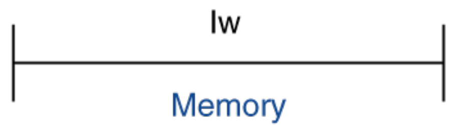


EX for Load

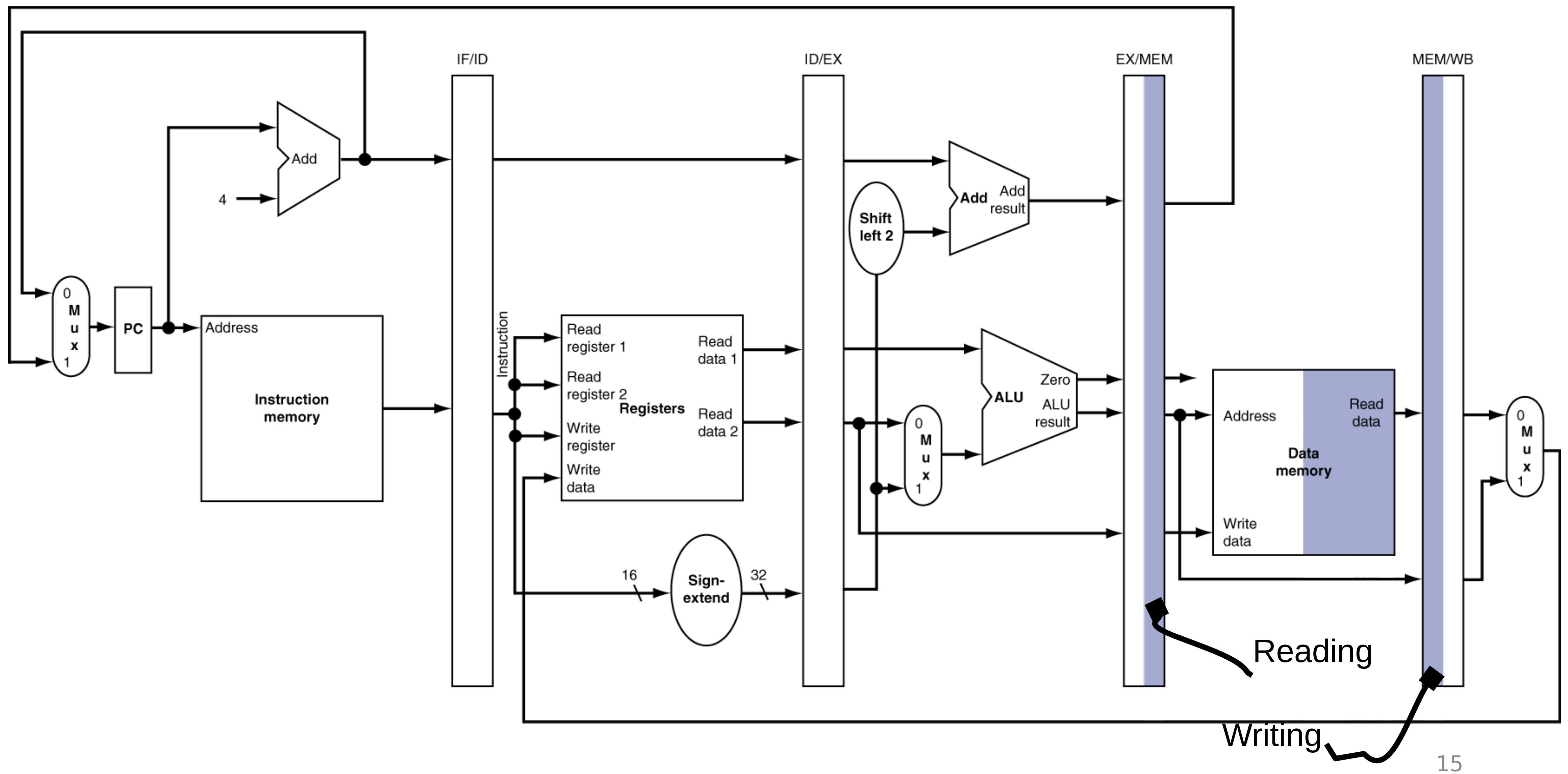

lw
Execution


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
0
M
u
x
1
Reading
Writing

MEM for Load

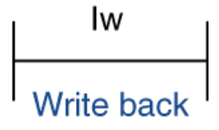


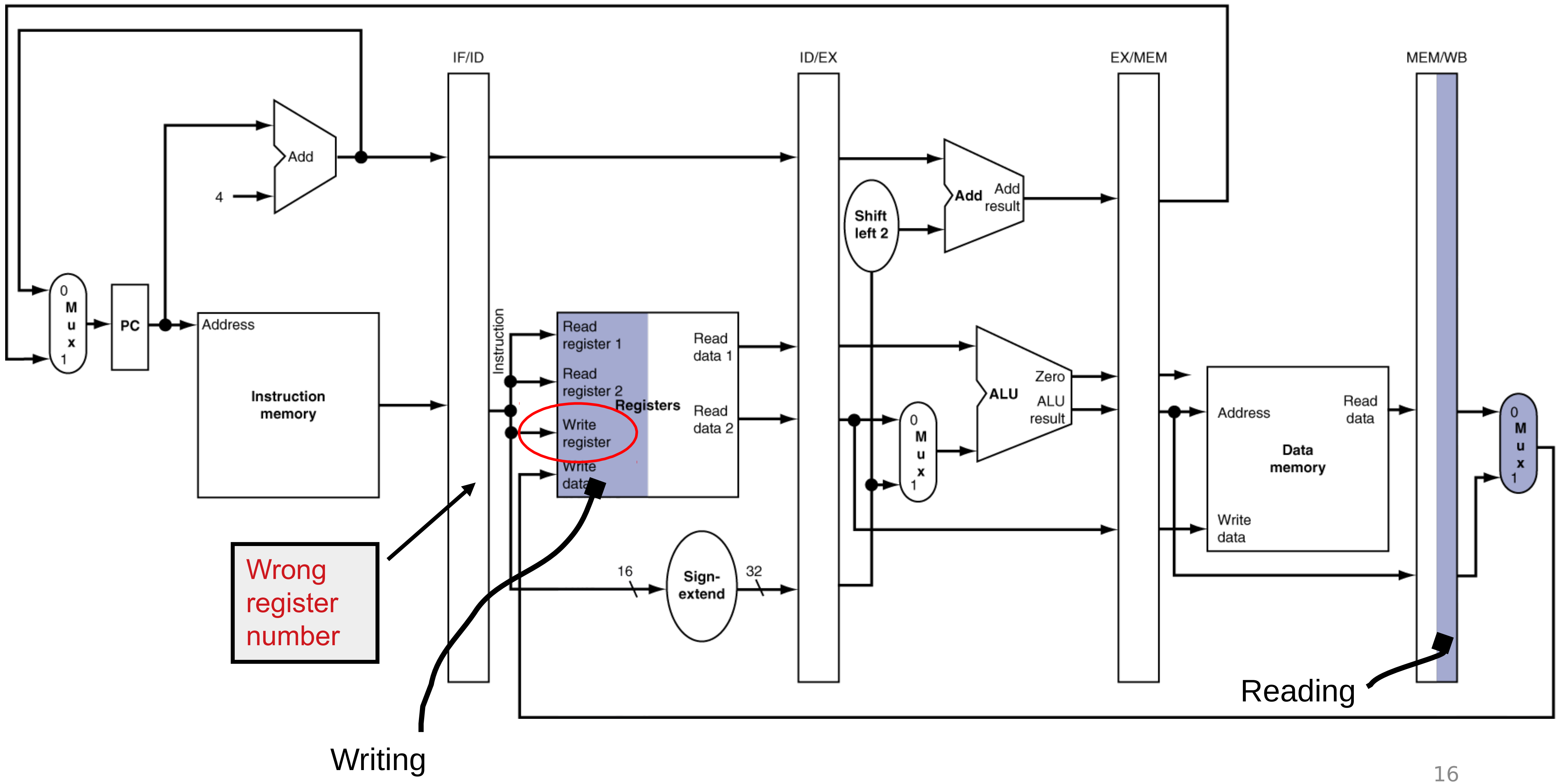
lw
Memory



IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
Reading
Writing

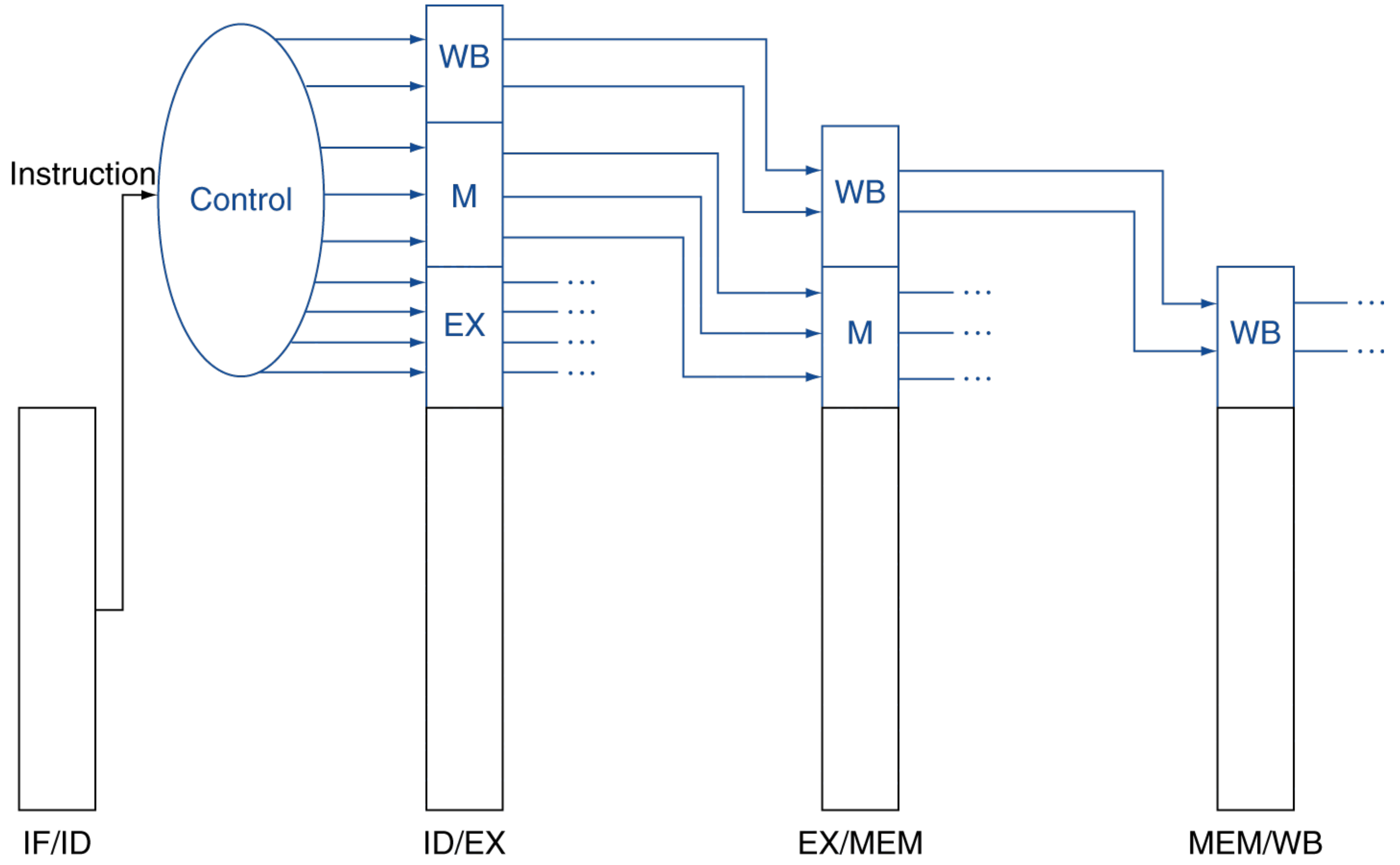
WB for Load


lw
Write back


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
Wrong register number
Writing
Reading

Pipelined Control

- Control signals derived from instruction
- Control lines start with execution stage
- Relevant control should propagate with the instruction → Extended stage registers

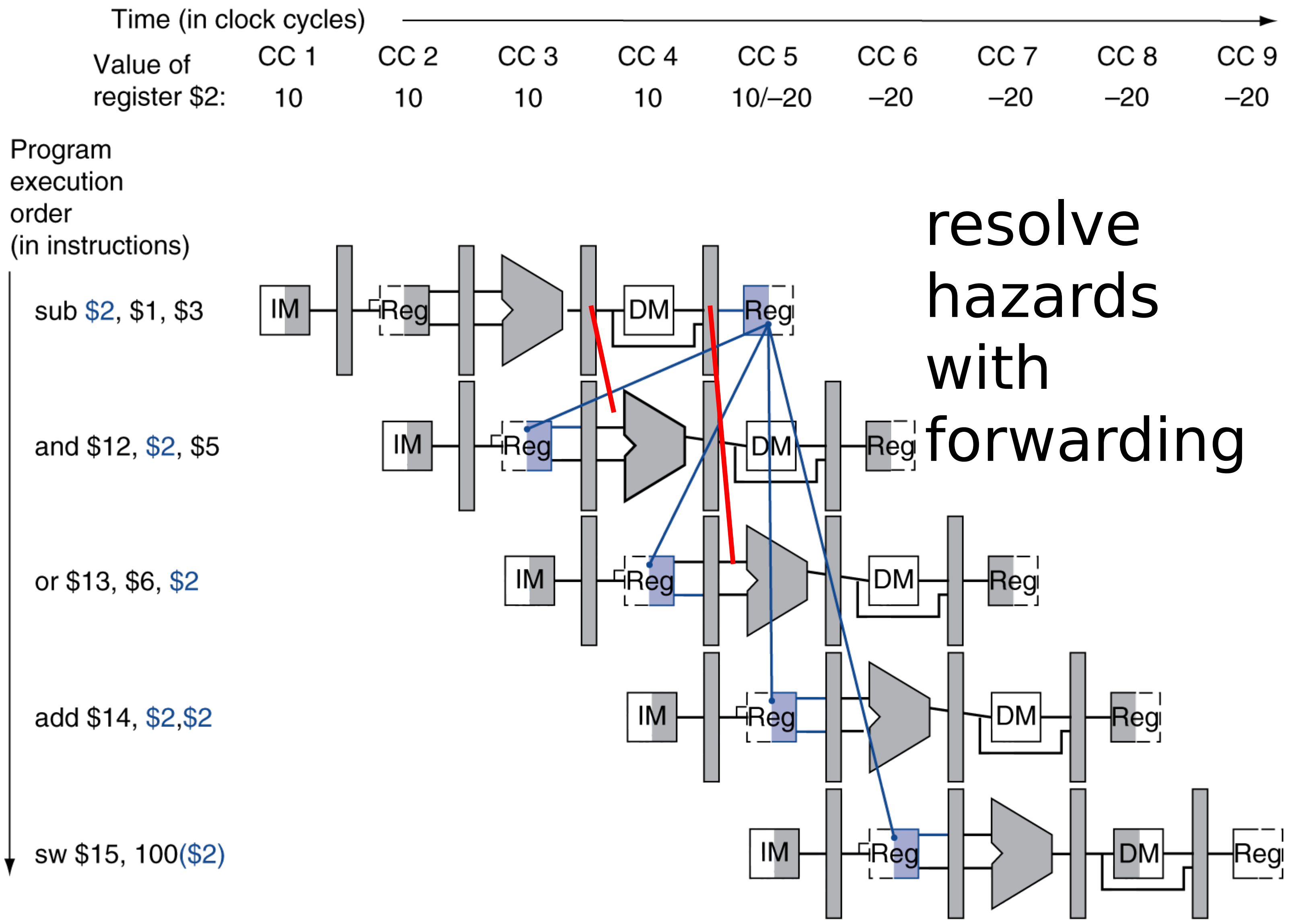


Pipelined Control Design `

Data Hazards in ALU Instructions

- Consider this sequence with many dependences:

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```



resolve hazards with forwarding

Assuming $2 has the value

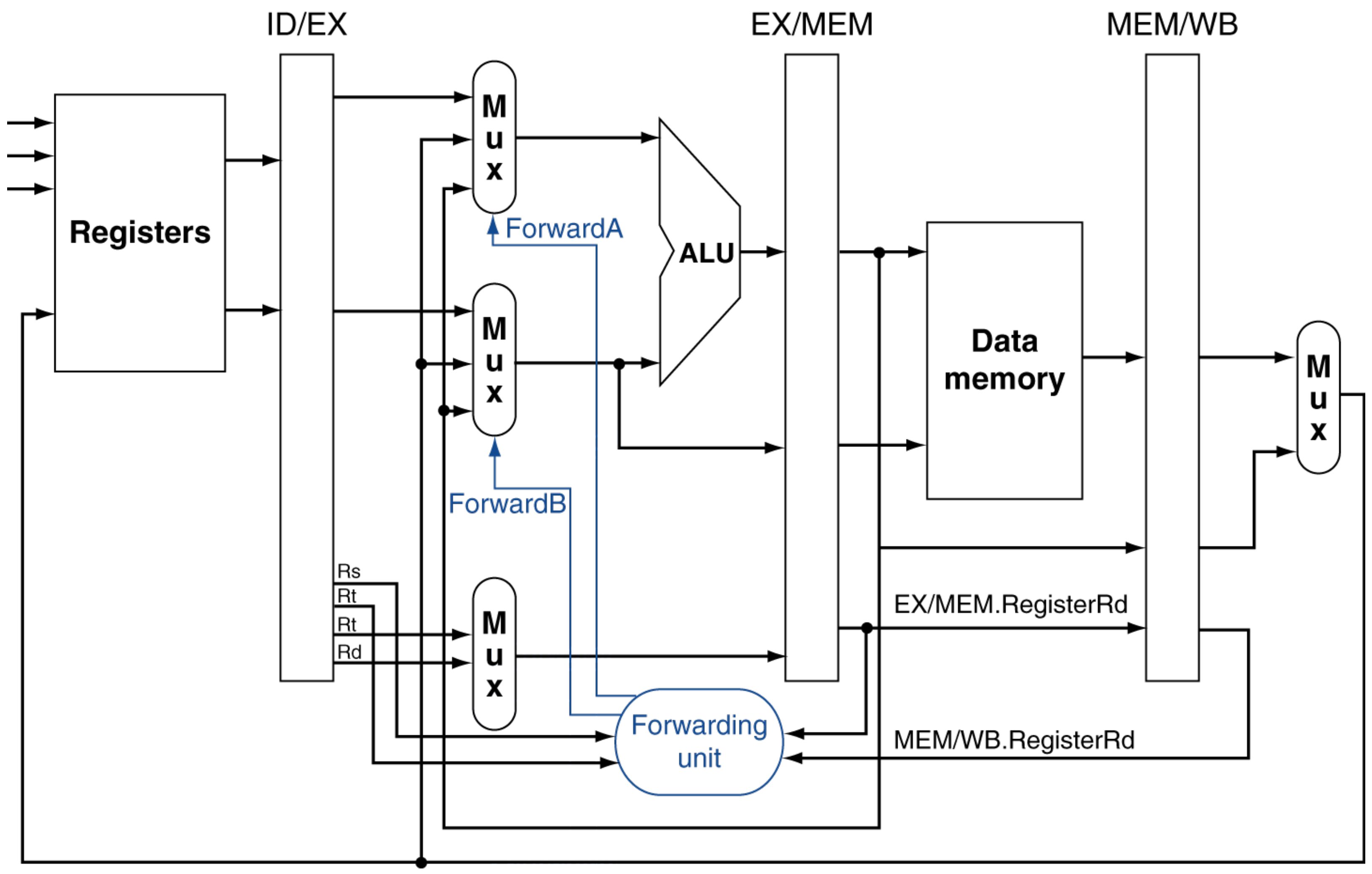
10 before the subtraction

-20 after the subtraction

Forwarding Hardware

- Forward implies reading ALU operands from a source other than the register file registers
  - ***Multiplexers*** are needed at the ALU I/P
  - A ***forwarding unit*** determines the multiplexor control signal

Forwarding Paths



b. With forwarding

Forwarding Unit Output

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Reading

Section 4.6