

CSE261: Computer Architecture

12. Processor (3): Pipelining

Seongil Wi

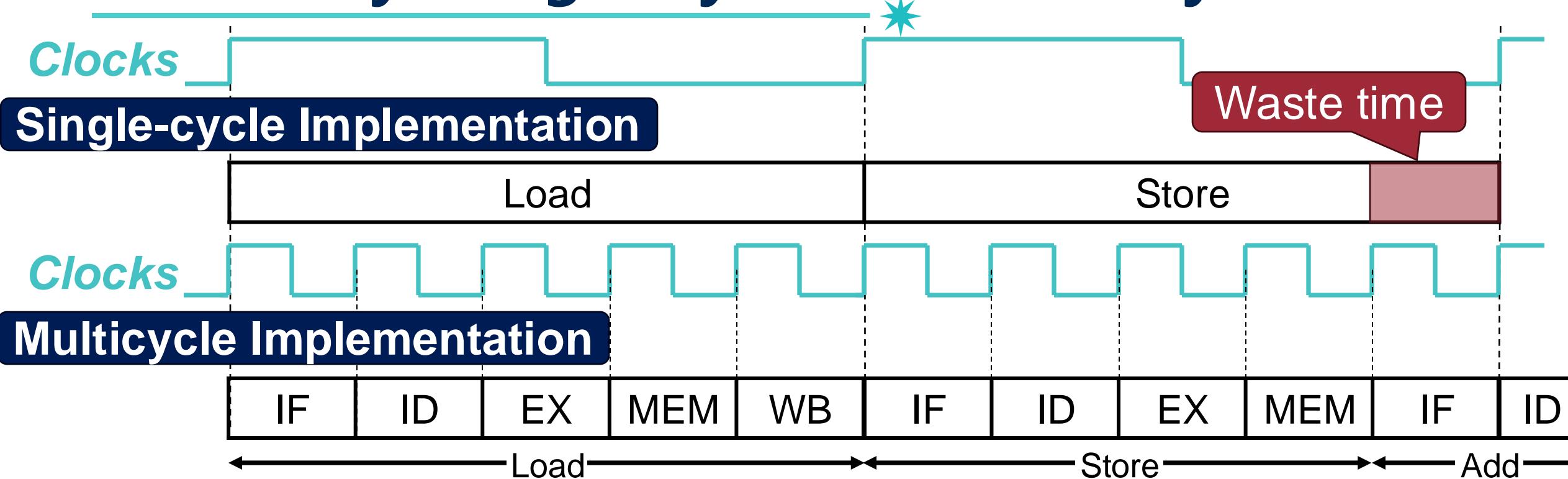
HW2 will be Released Soon!



Implementing single-cycle datapath and control

- Due: Nov. 21, 11:49PM

Summary: Single-cycle and Multicycle CPU



Is there a way to increase the speed further?

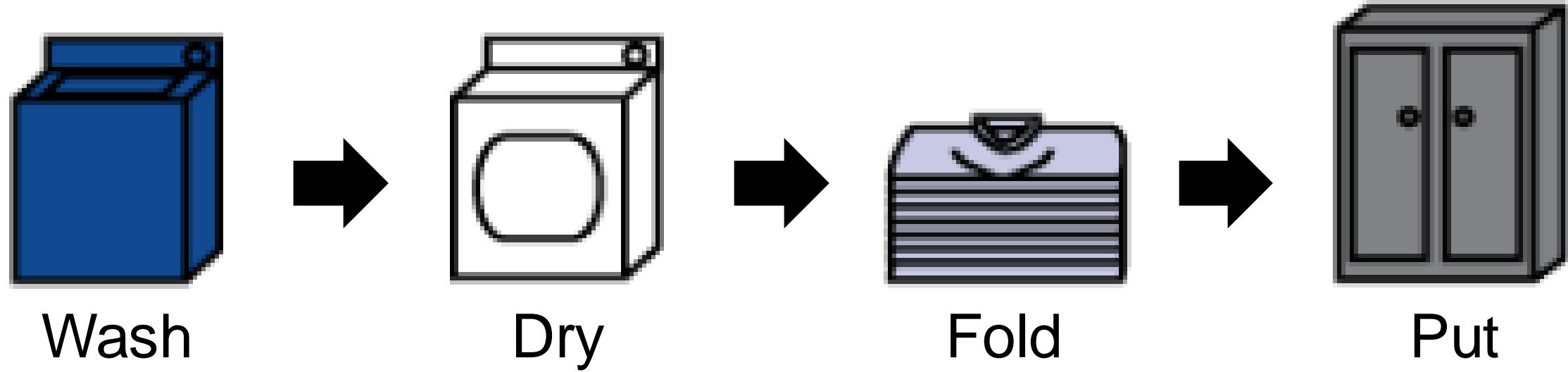
Pipelining Concept

Pipelining



Several instruction are executed at the same time to enhance the performance

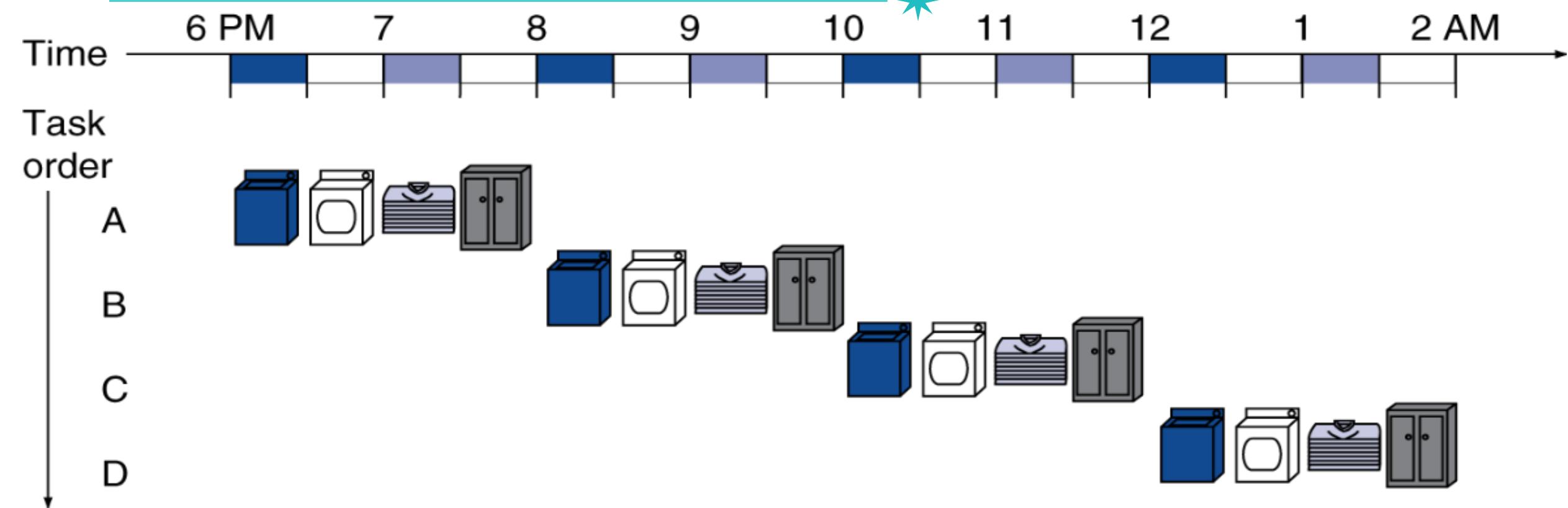
Motivation Example: Sequential Laundry Processing⁶



Assume you have a set of clothes to wash: SET A, B, C, D

Motivation Example: Sequential Laundry Processing

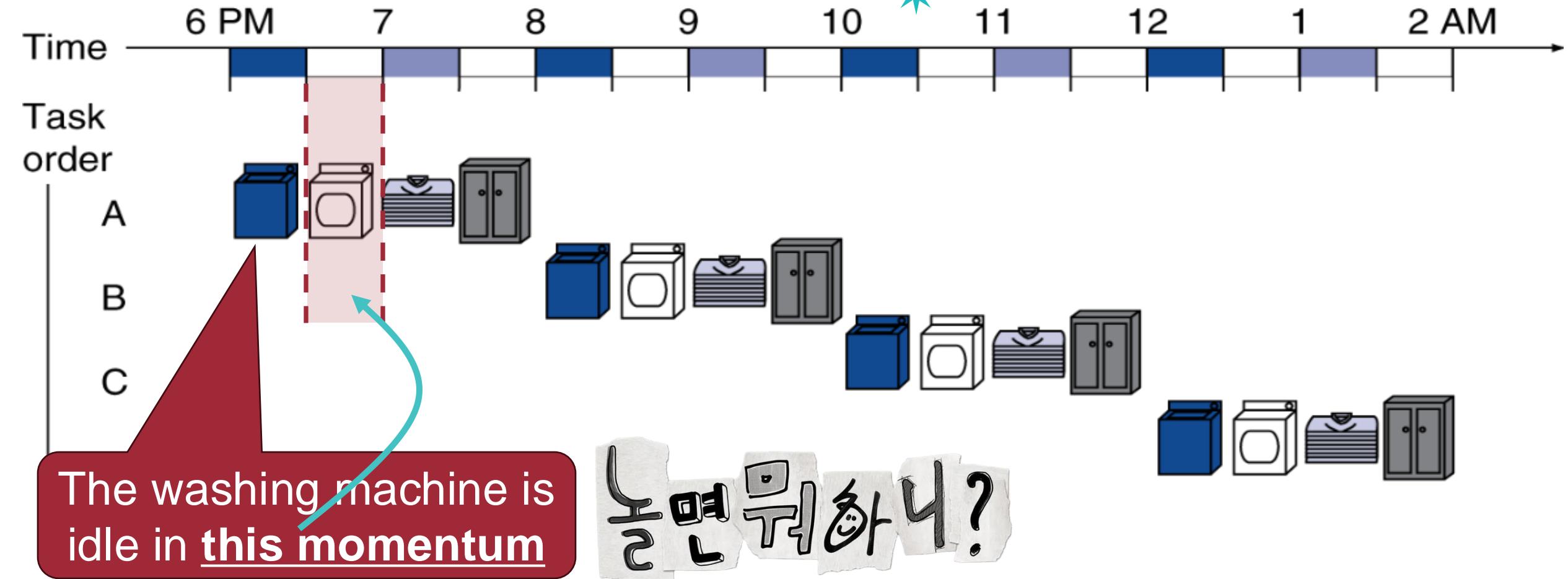
7



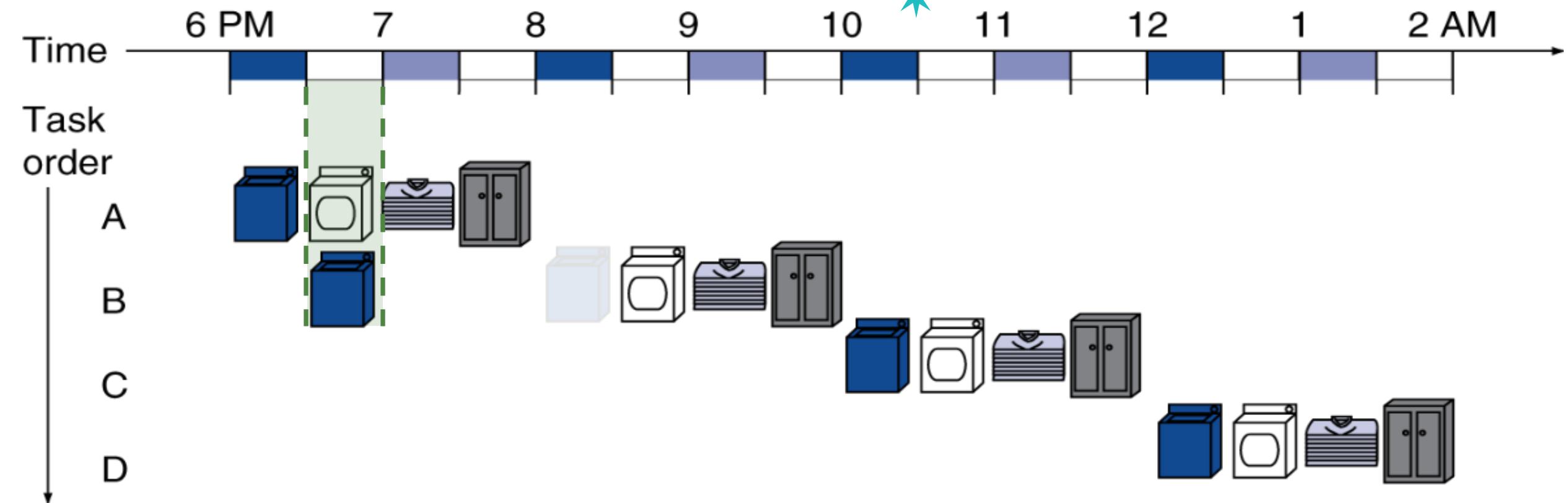
Single- and multi-cycle implementation start the next task only after the previous task has been completed

Execution time:
 $2 + 2 + 2 + 2$
= 8 hours

Let's Think about it in a More Natural Way!



Solution: Execute Washing at the Same Time



Problem Again...



10

Task
order

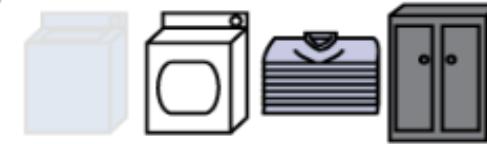
A



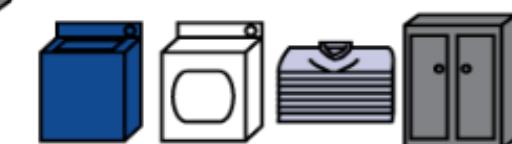
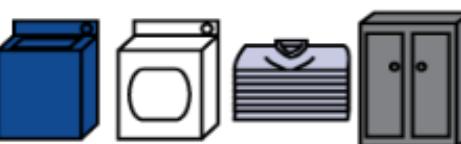
B



C

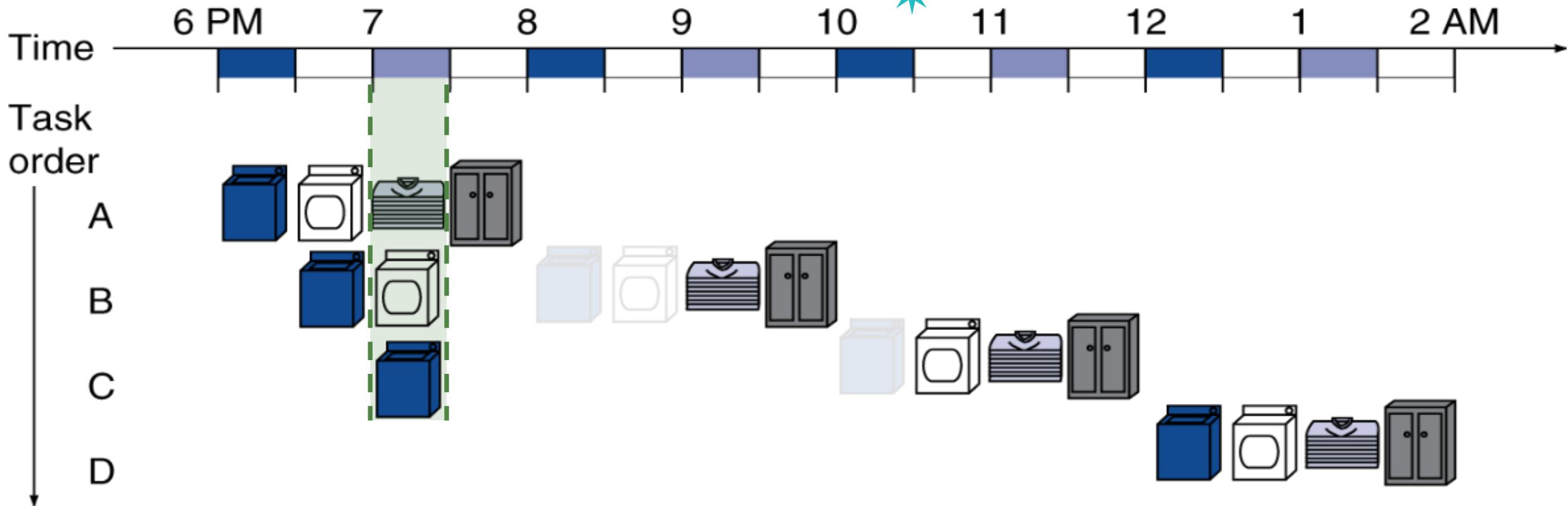


D



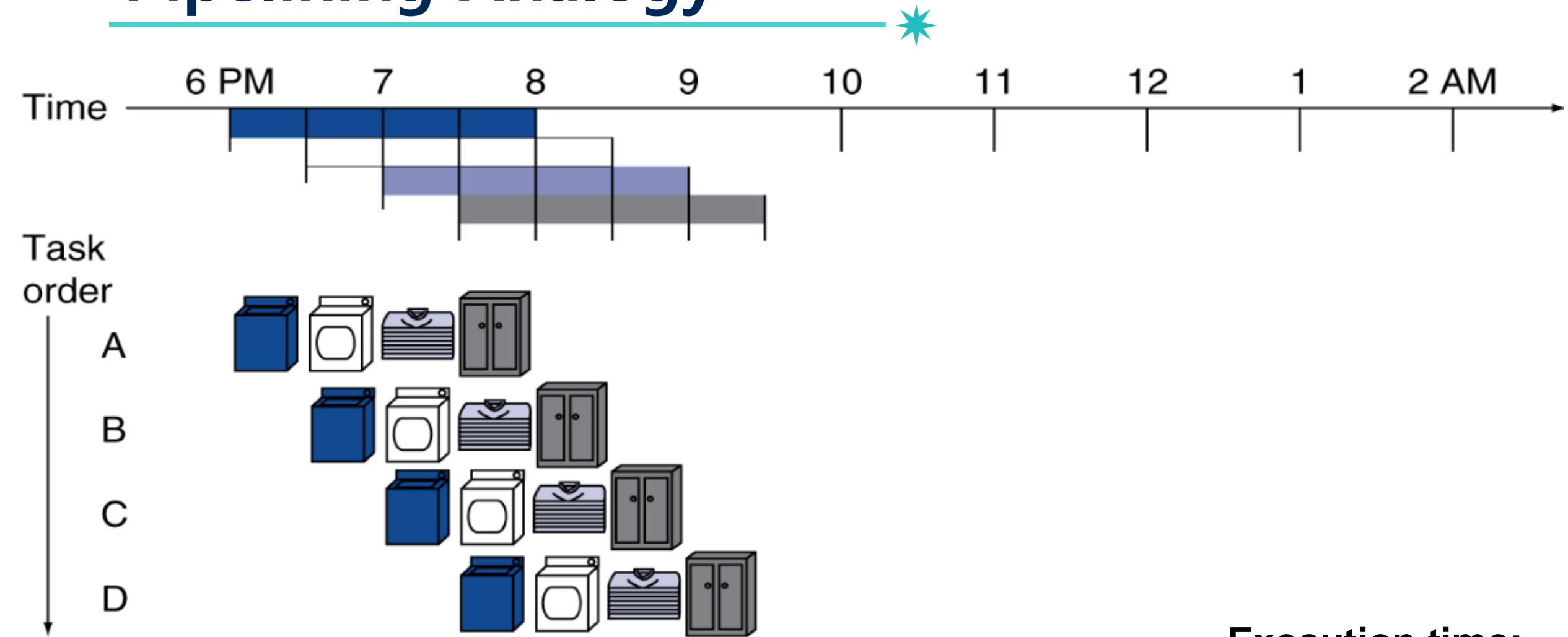
The washing machine and dry machine is idle in this momentum

Solution: Execute Washing/drying at the Same Time



Pipelining Analogy

12

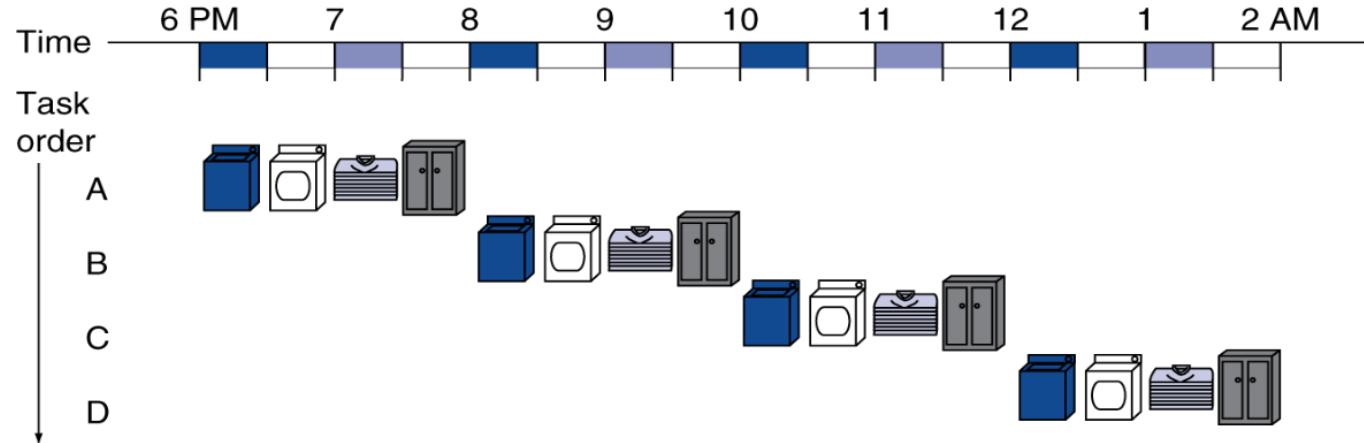


Pipelining overlaps execution

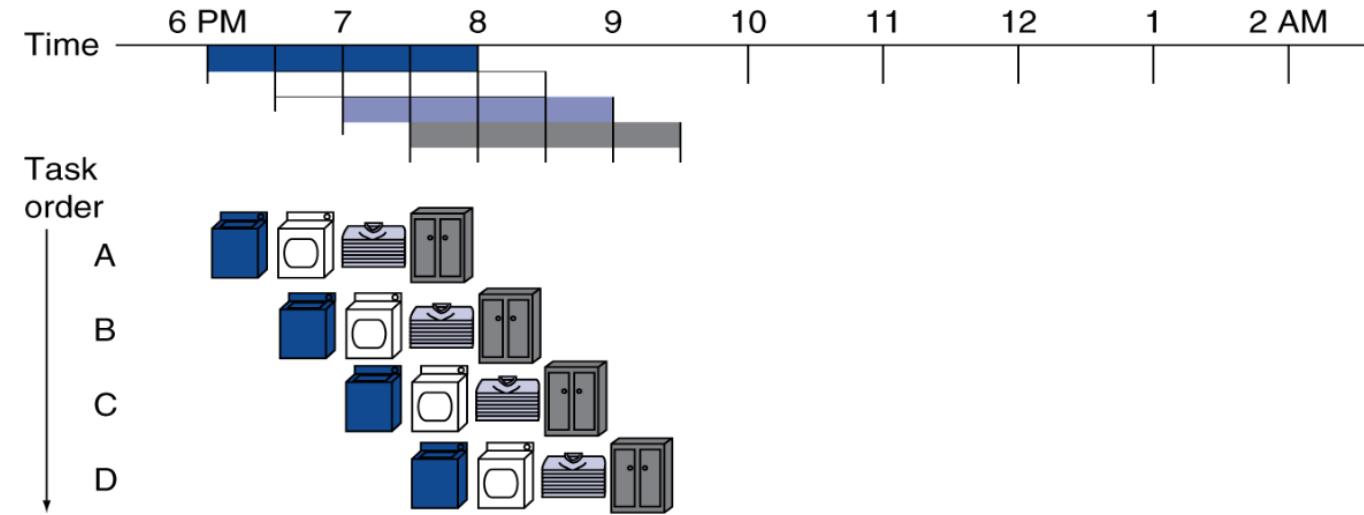
Execution time:
 $2 + 0.5 + 0.5 + 0.5 = 3.5 \text{ hours}$

Performance Comparison: Speedup

13



Without pipelining
execution time:
 $2 + 2 + 2 + 2$
= 8 hours



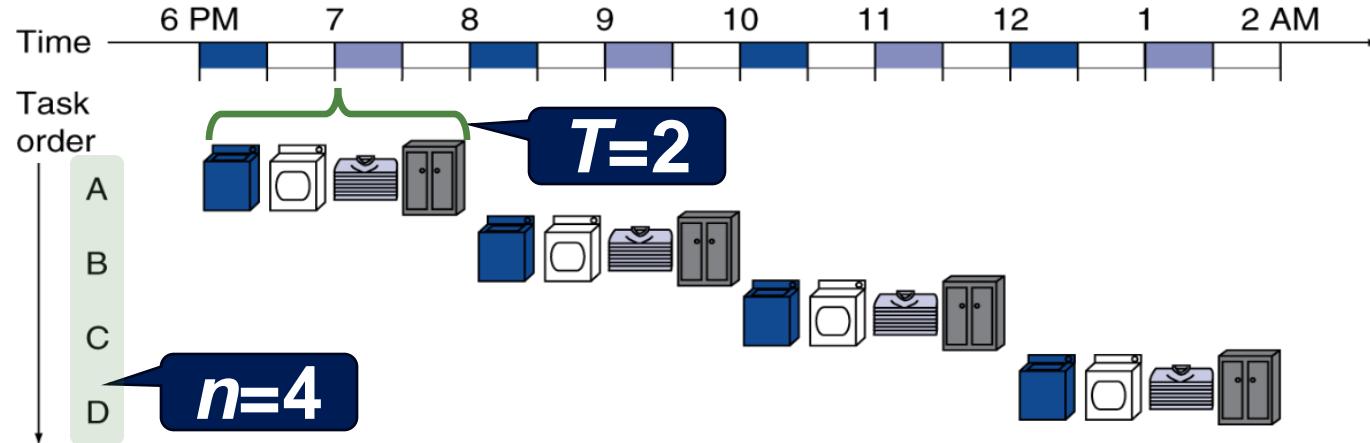
Pipelining
execution time:
 $2 + 0.5 + 0.5 + 0.5$
= 3.5 hours

Speedup?
 $8/3.5 = 2.3$

Let's Think about Speedup in General Terms

14

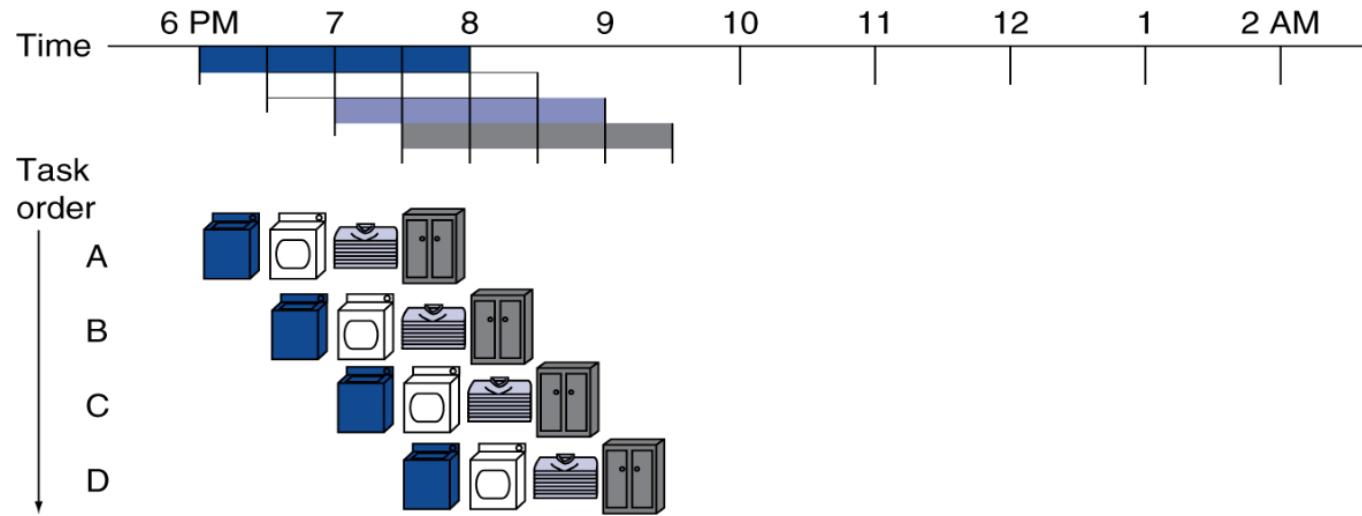
Suppose there are n jobs to execute and each job takes T hours



Without pipelining
execution time:

$$2 + 2 + 2 + 2
= 8 \text{ hours}$$

$$n*T$$



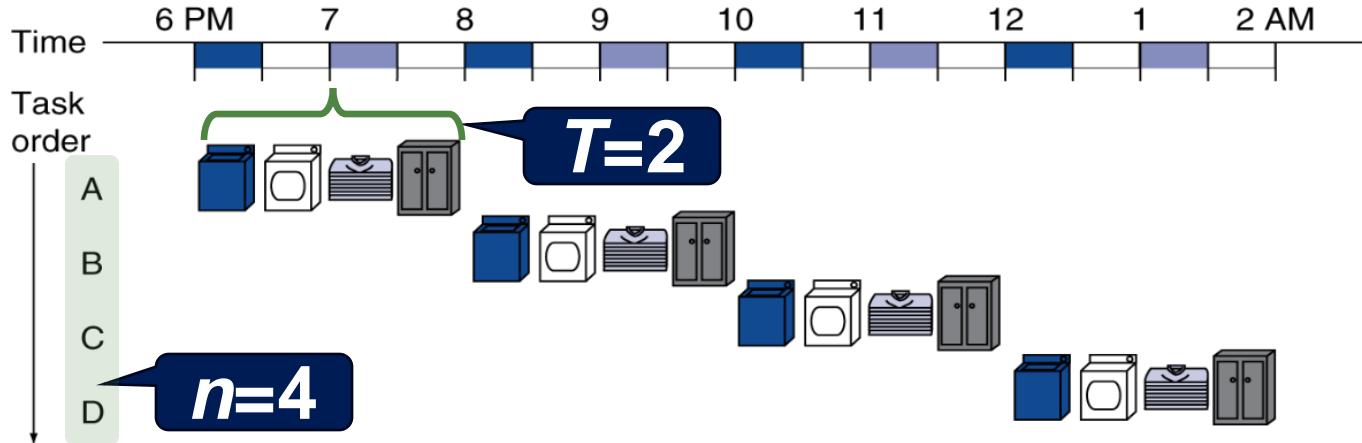
Pipelining
execution time:

$$2 + 0.5 + 0.5 + 0.5
= 3.5 \text{ hours}$$

Let's Think about Speedup in General Terms

15

Suppose there are n jobs to execute and each job takes T hours

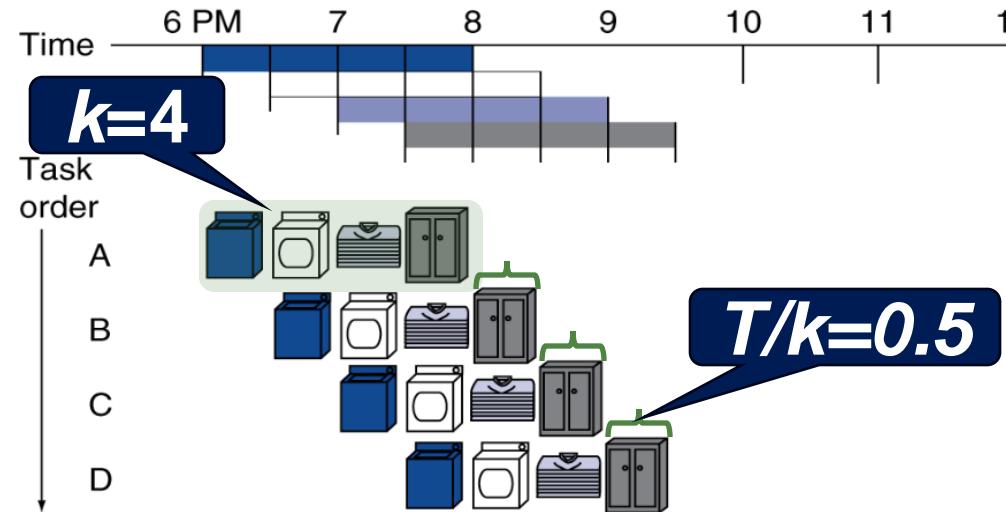


Without pipelining
execution time:

$$2 + 2 + 2 + 2 = 8 \text{ hours}$$

$$n * T$$

Suppose we have k stage



Pipelining
execution time:

$$2 + 0.5 + 0.5 + 0.5 = 3.5 \text{ hours}$$

$$(n+k-1) * T/K$$

Pipelining Speedup

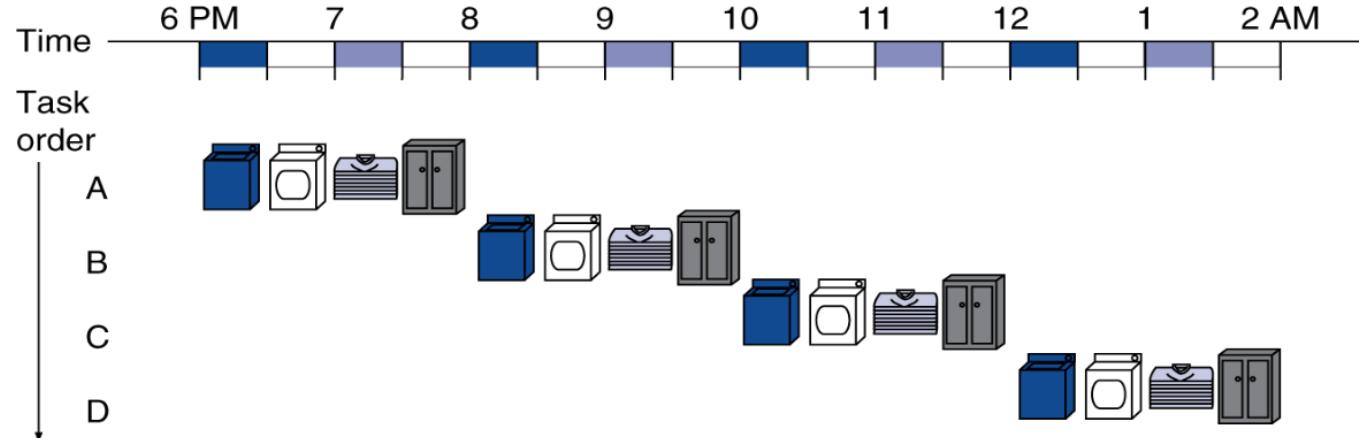


- Execution time *without pipelining*: $n*T$
- Execution time *with pipelining*: $(n+k-1)*T/k$
- **Pipelining speedup:** $\frac{n*T}{(n+k-1)*T/k} = \frac{n*k}{n+k-1}$
- **If n goes infinity, pipelining speedup becomes k**

$$\lim_{n \rightarrow \infty} \frac{n*k}{n+k-1} = k$$

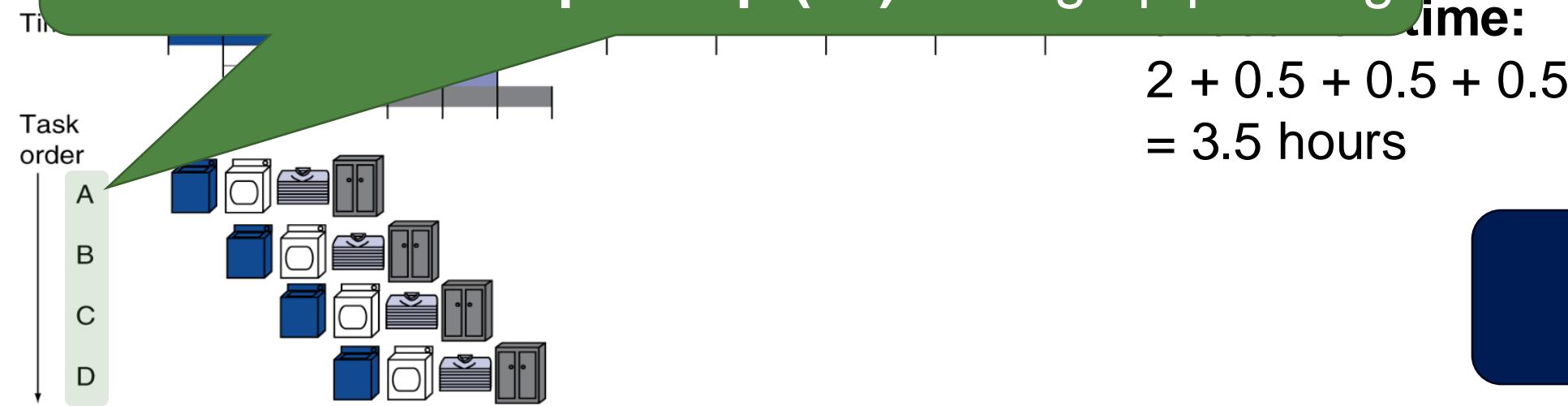
If we have a large number of jobs, we can achieve a **speedup of about k times** through pipelining

Performance Comparison: Speedup



Without pipelining execution time:
 $2 + 2 + 2 + 2 = 8 \text{ hours}$

If we have a large number of jobs, we can achieve **about 4 speedup (x4)** through pipelining



With pipelining execution time:
 $2 + 0.5 + 0.5 + 0.5 = 3.5 \text{ hours}$

Speedup?
 $8/3.5 = 2.3$

Pipeline Lessons

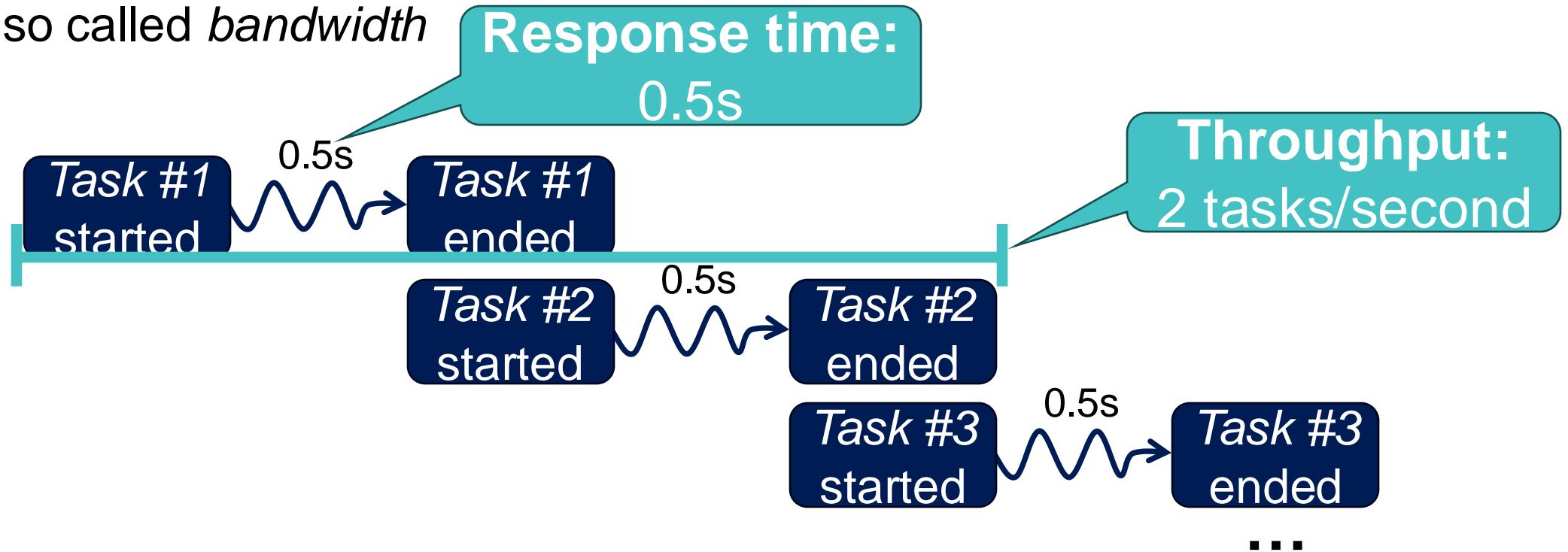


- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload

Performance Metrics



- **Response time:** the time between the start and completion of a task
 - Also called *execution time, latency*
- **Throughput:** total work done per unit time
 - E.g., # of tasks/transactions/... per hour
 - Also called *bandwidth*



Pipeline Lessons

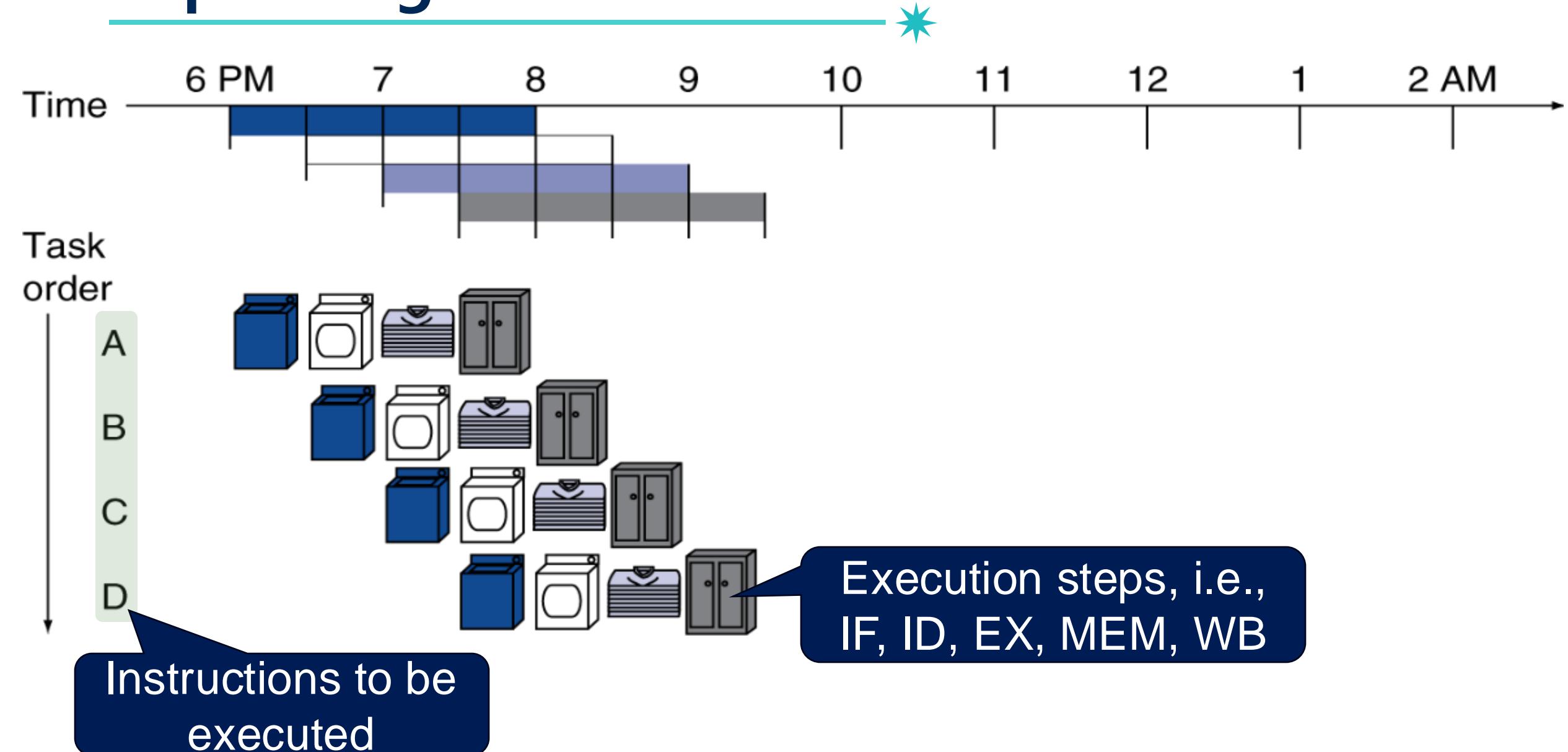


- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = # pipe stages

Pipelining Overview

Pipelining Overview

22



MIPS Pipelining

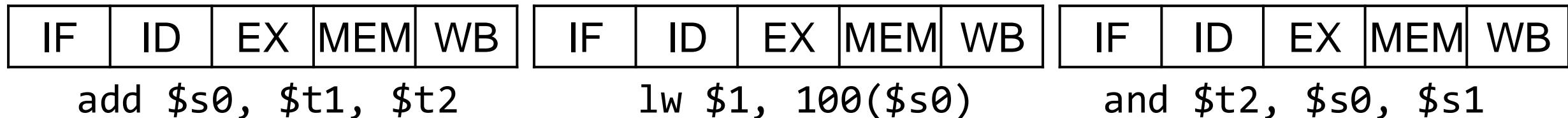


- Five stages ($k=5$), one step per stage
 1. **IF**: Instruction fetch from memory
 2. **ID**: Instruction decode & register read
 3. **EX**: Execute operation or calculate address
 4. **MEM**: Access memory operand
 5. **WB**: Write result back to register

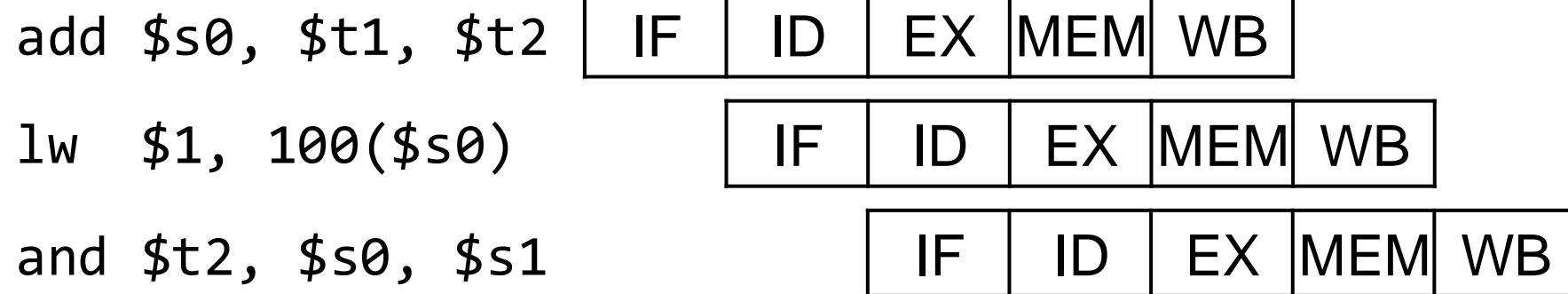
MIPS Pipelining



- **Sequential execution**



- **Pipelined execution**



Single-cycle vs. Multicycle vs. Pipeline

25

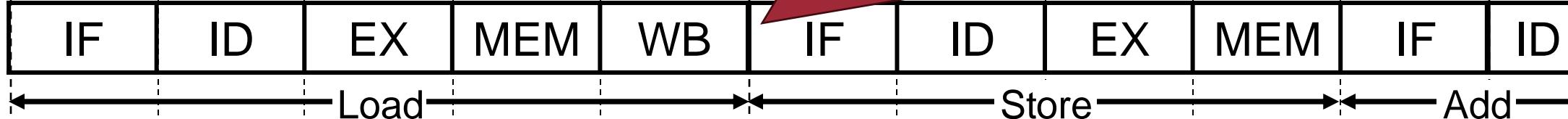
Clocks

Single-cycle Implementation



Clocks

Multicycle Implementation



Load

Store

Add

Single-cycle vs. Multicycle vs. Pipeline

26

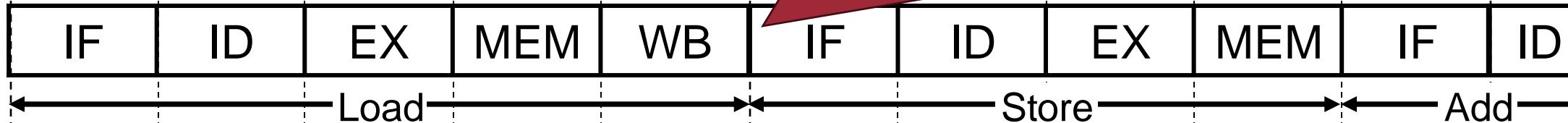
Clocks

Single-cycle Implementation

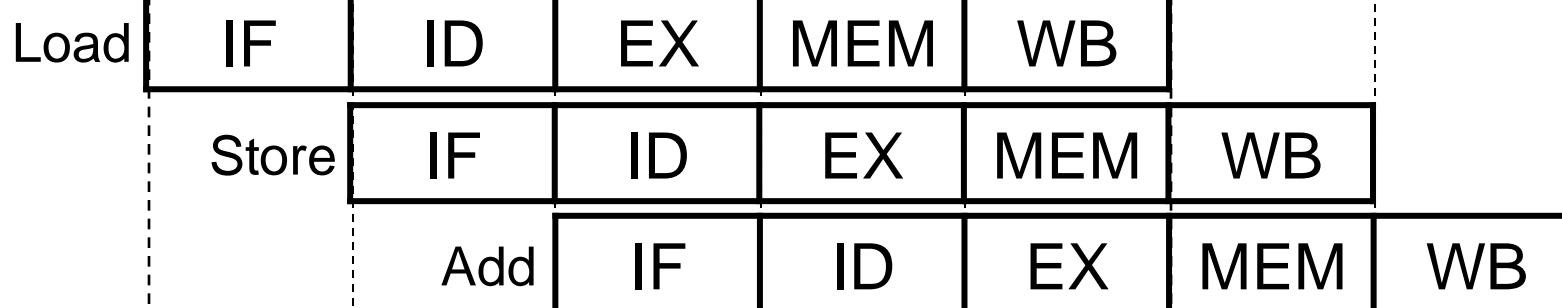


Clocks

Multicycle Implementation



Pipeline Implementation



Pipeline Performance

27

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

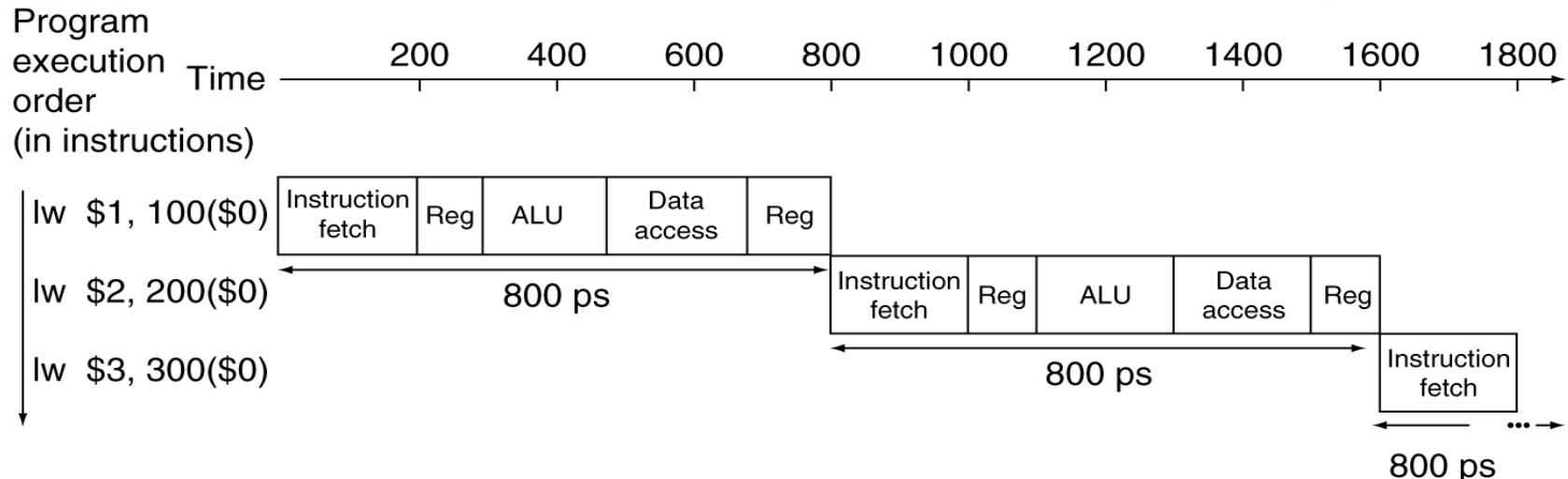
Instruction	Functional units used by the instruction class				
	Instruction fetch	Instruction decode	Execution	Memory Access	Register Write-back
R-type	200ps	100ps	200ps		100ps
Load word	200ps	100ps	200ps	200ps	100ps
Store word	200ps	100ps	200ps	200ps	
Conditional Branch	200ps	100ps	200ps		
Jump	200ps				

The diagram shows the total cycle time for each instruction type, calculated as the sum of the times for each functional unit. The results are:

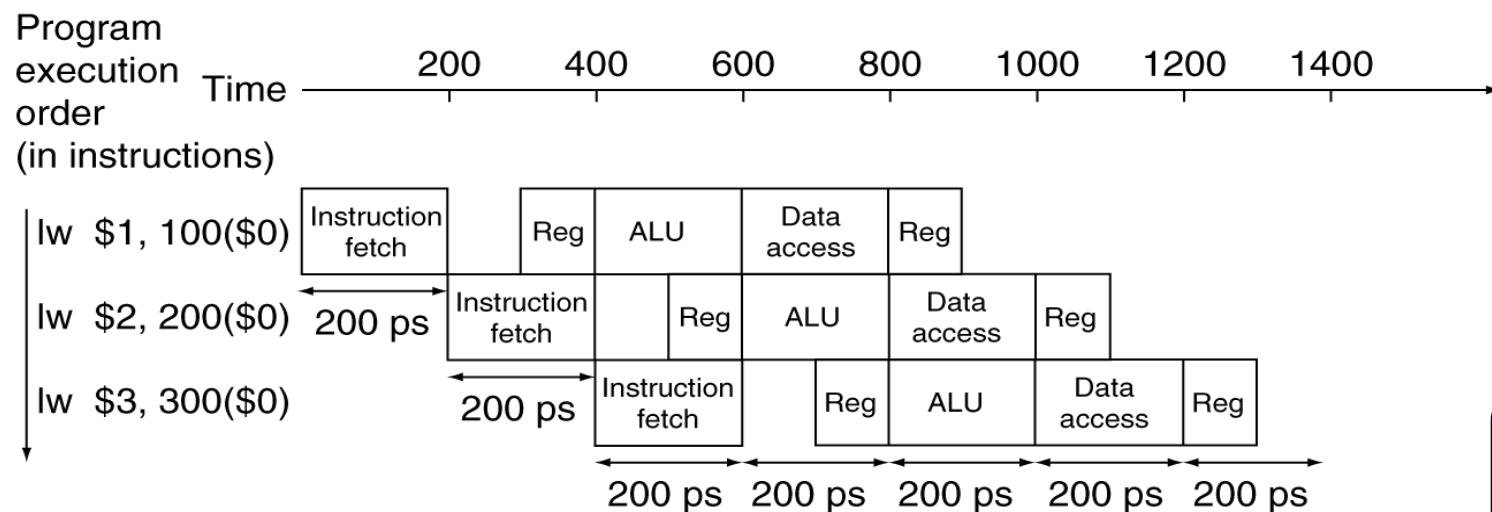
- R-type: 600ps
- Load word: 800ps
- Store word: 700ps
- Conditional Branch: 500ps
- Jump: 200ps

Pipeline Performance: Speed Up?

28



Without pipelining execution time:
 $800 + 800 + 800 = 2400 \text{ ps}$

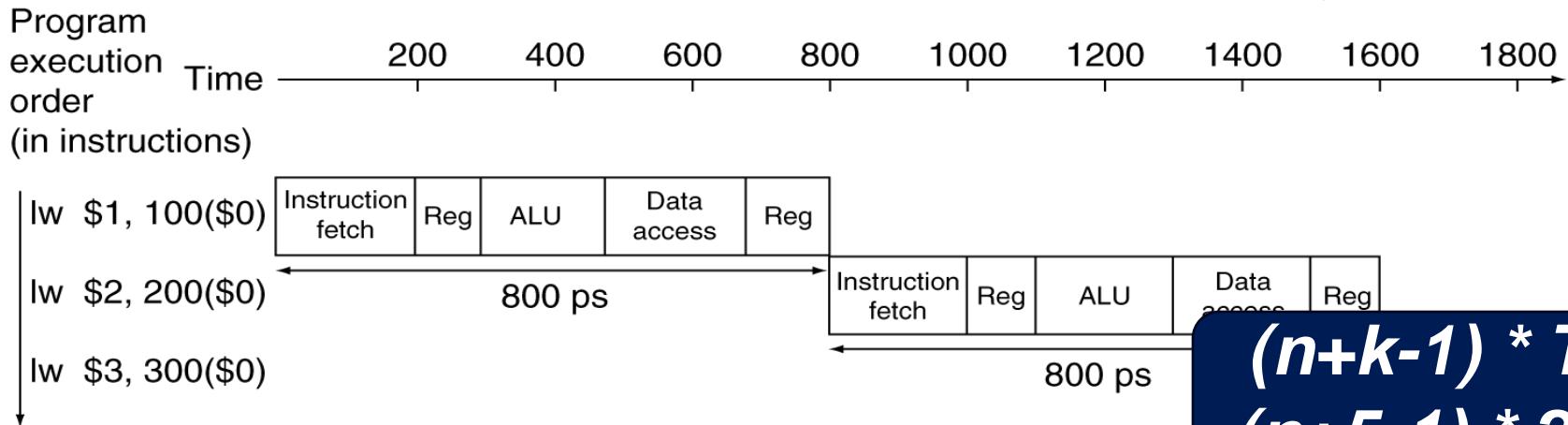


Pipelining execution time:
 $800 + 200 + 200 + 200 = 1400 \text{ ps}$

Speedup?
 $2400/1400 \approx 1.7$

What if We have 1,000,003 Instructions?

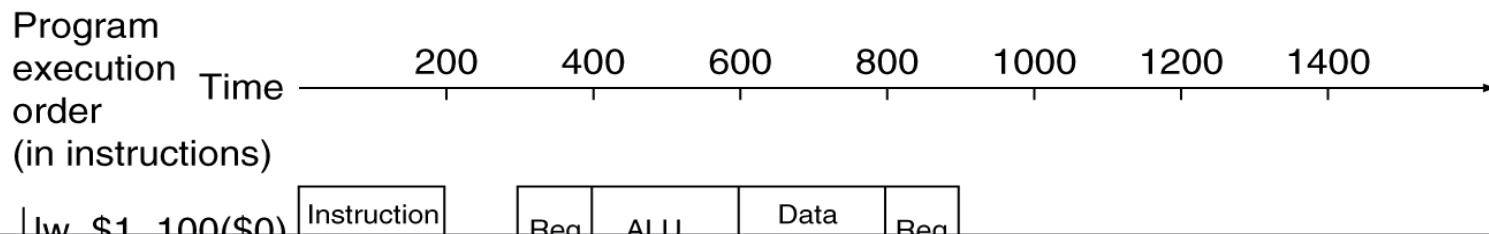
29



Without pipelining
execution time:

$$n \cdot T = n \cdot 800\text{ps}$$

$$(n+k-1) * T/K = \\ (n+5-1) * 200\text{ps}$$



Pipelining
execution time:

$$\text{Speedup} = \frac{n \cdot 800\text{ps}}{(n+4) \cdot 200\text{ps}} \quad (n=1,000,003)$$

$$= 800,002,400 \text{ ps} / 200,01,400 \text{ ps} \cong 4.00$$

MIPS ISA is Designed to Work Well with Pipelining



- **All instructions are 32-bits**
 - Consistent and short instruction fetch/decode times
- **Few and regular instruction formats**
 - Consistent and short instruction decode time
- **Load/store addressing**
 - Fixed sequence: calculate address in 3rd stage, access memory in 4th stage

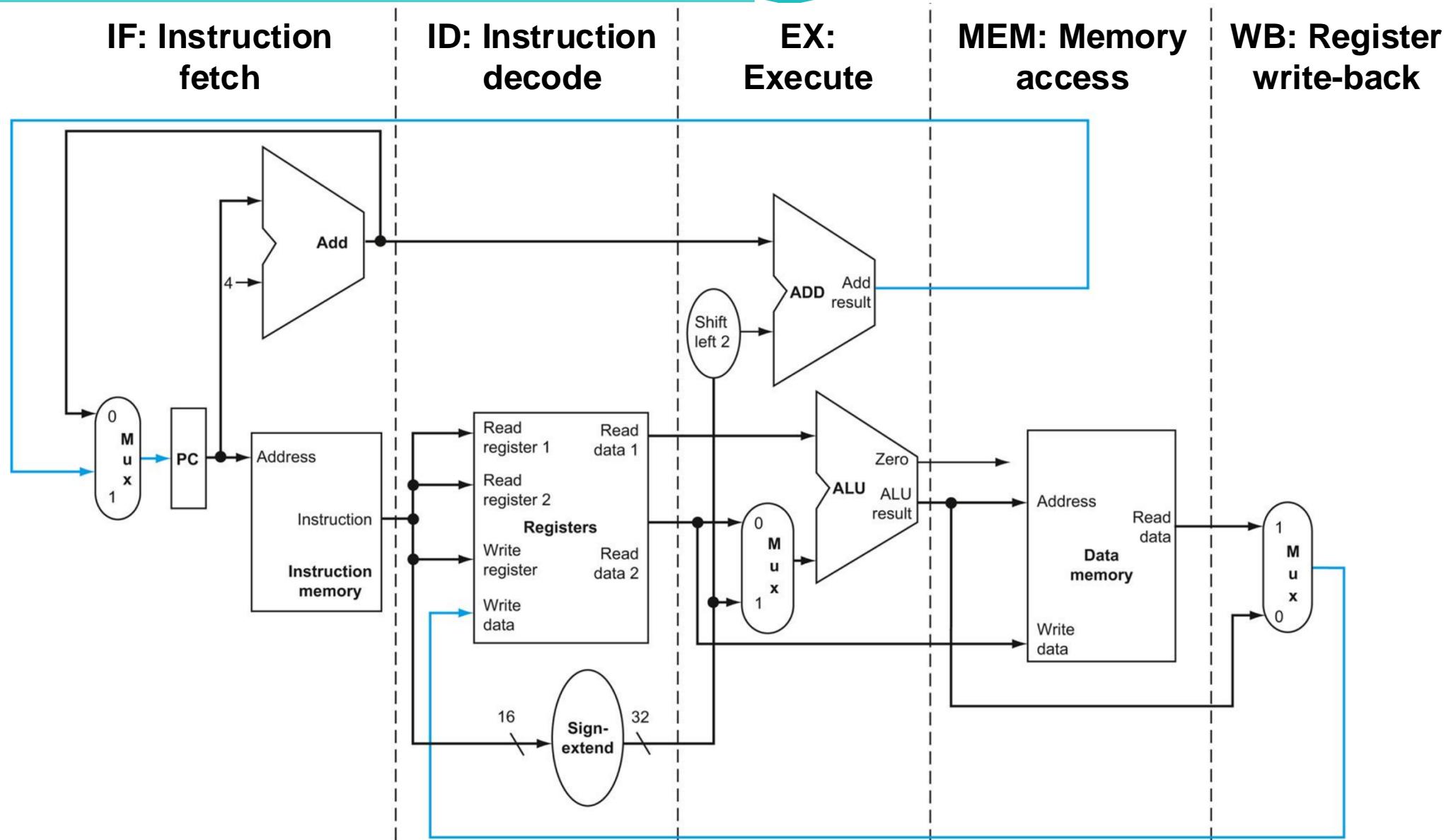
Pipeline Summary



- Pipelining improves performance by increasing instruction throughput
 - Executes multiple instructions in parallel
 - Each instruction has the same latency
- The speedup depends on longest pipe stage and the number of pipe stages
 - Length of pipe stage $\downarrow \rightarrow$ speedup \uparrow
 - # of pipe stage $\uparrow \rightarrow$ speedup \uparrow
 - Instruction set design affects complexity of pipeline implementation

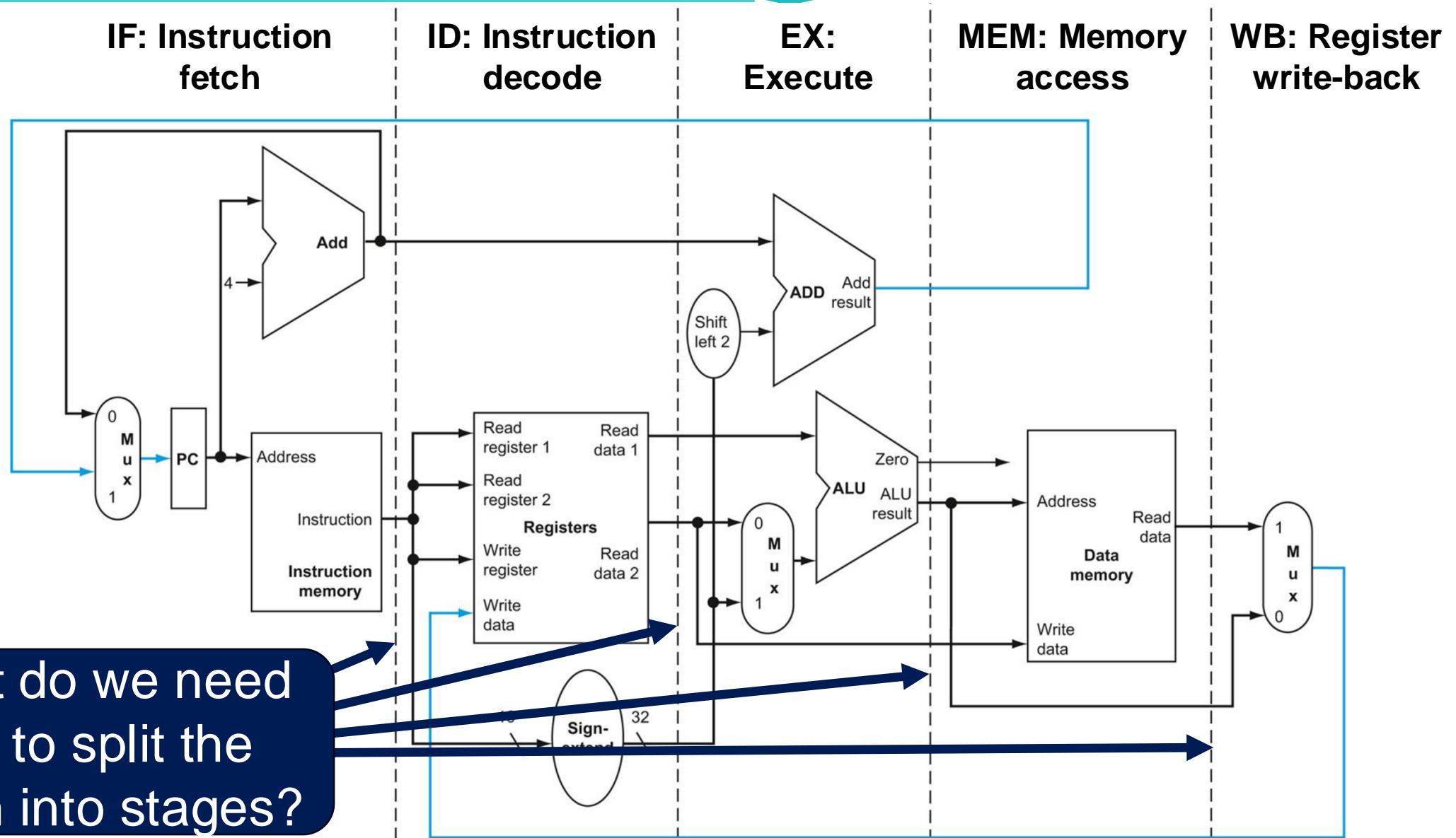
Pipelined Datapath

MIPS Pipelined Datapath



MIPS Pipelined Datapath

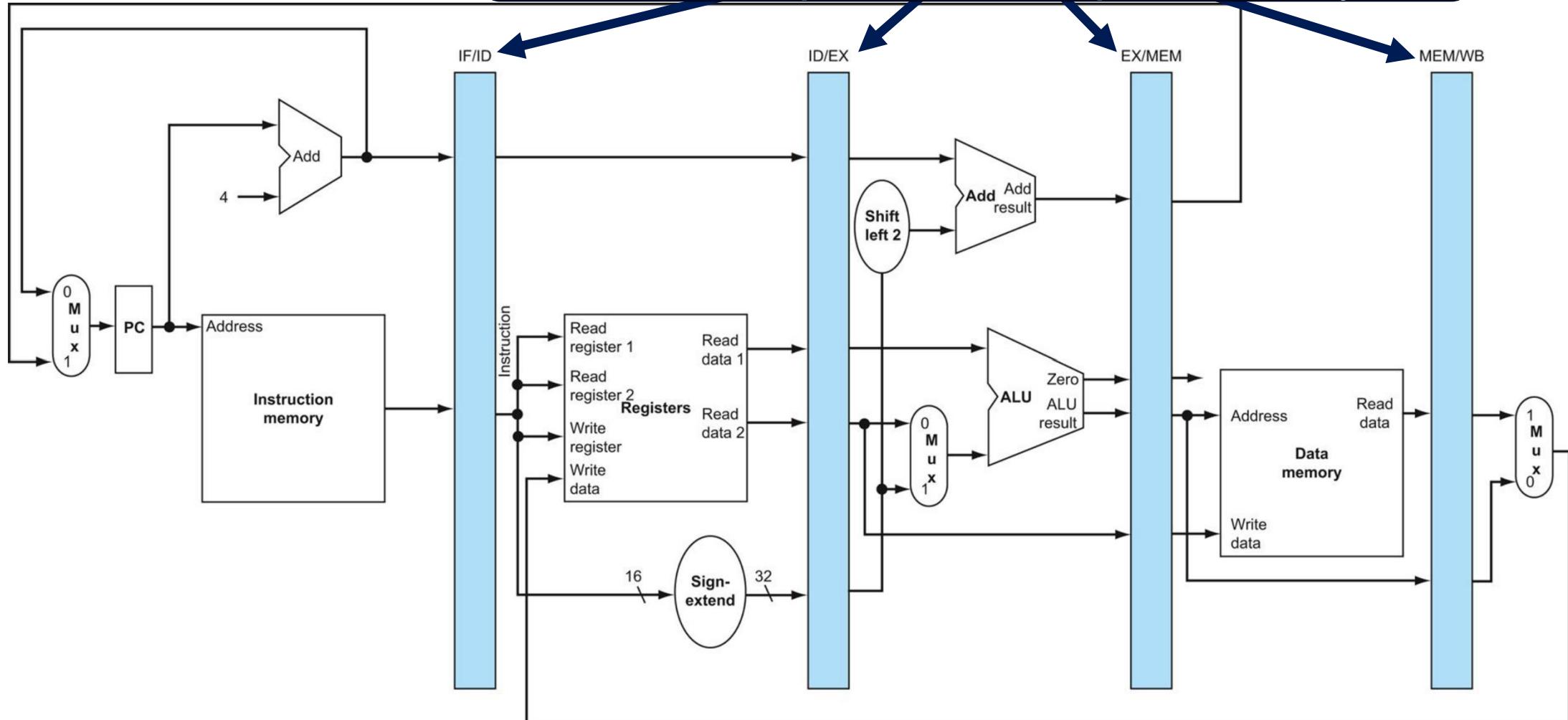
34



The Pipelined Version of the Datapath

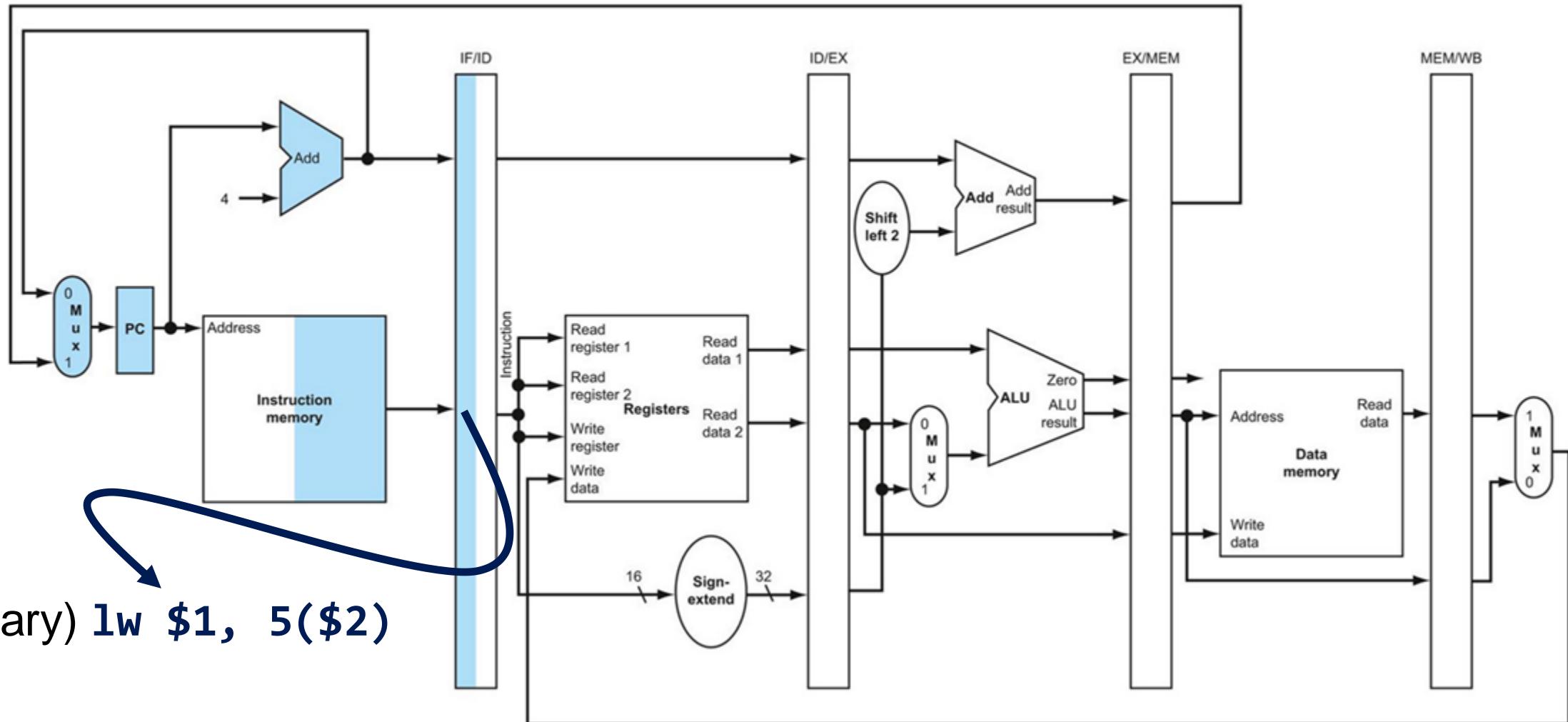
35

Registers between stages to hold information produced in previous cycle



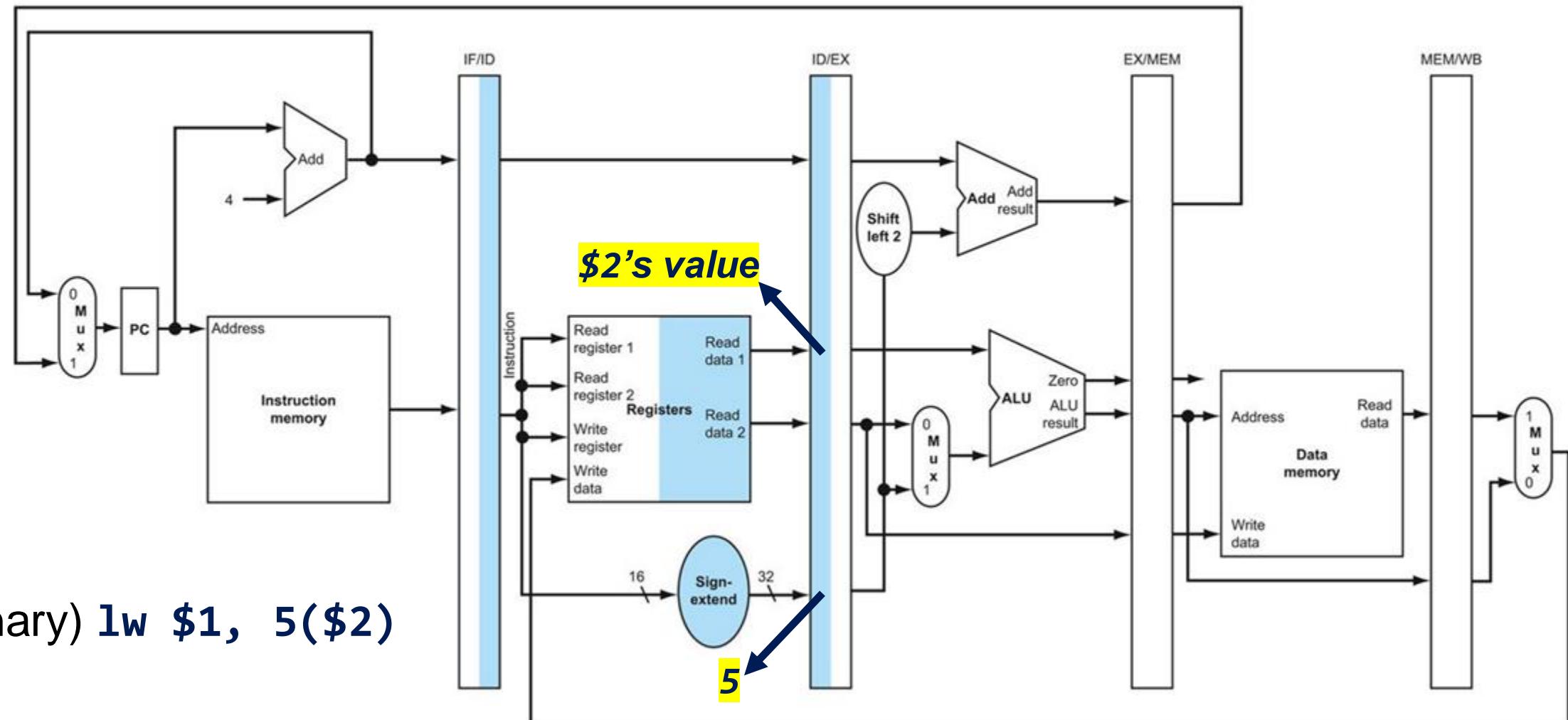
The Pipelined Version of the Datapath

36



The Pipelined Version of the Datapath

37

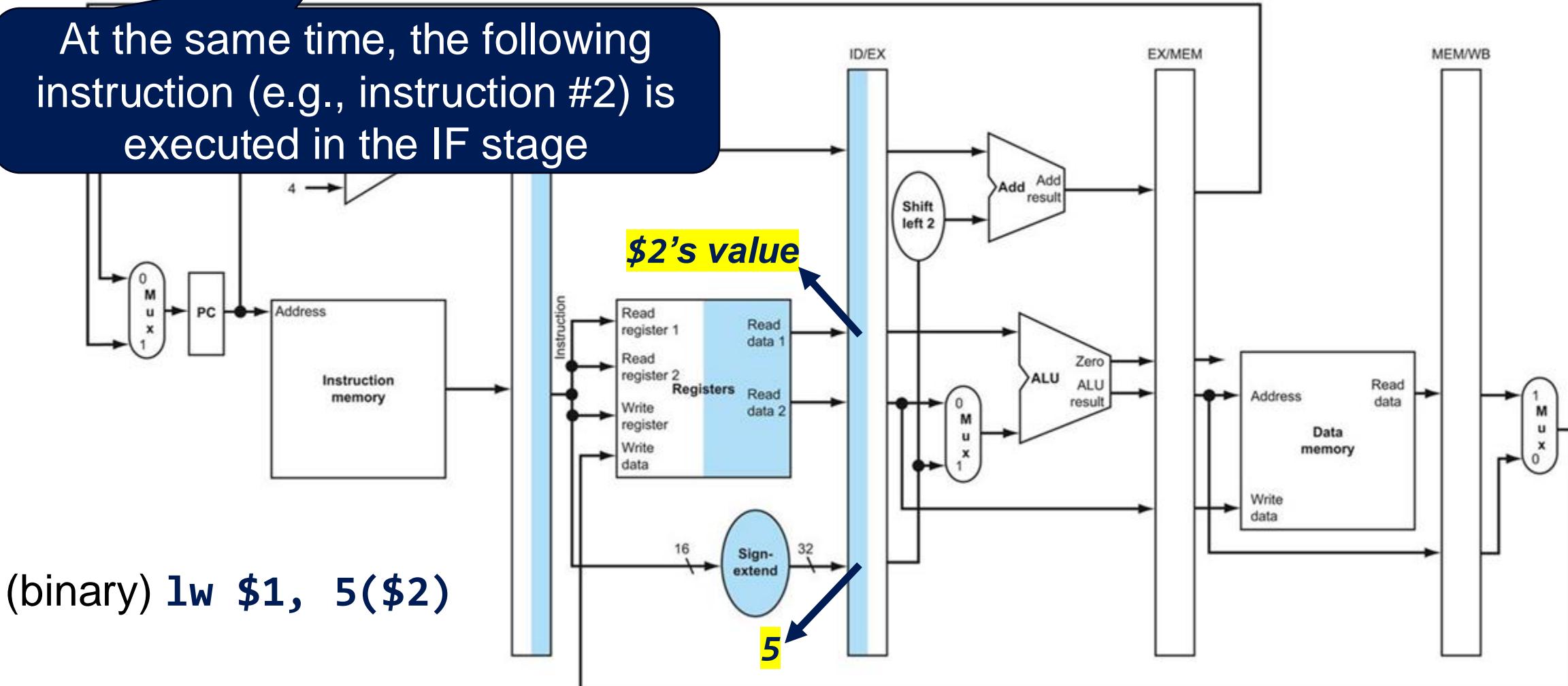


The Pipelined Version of the Datapath

38

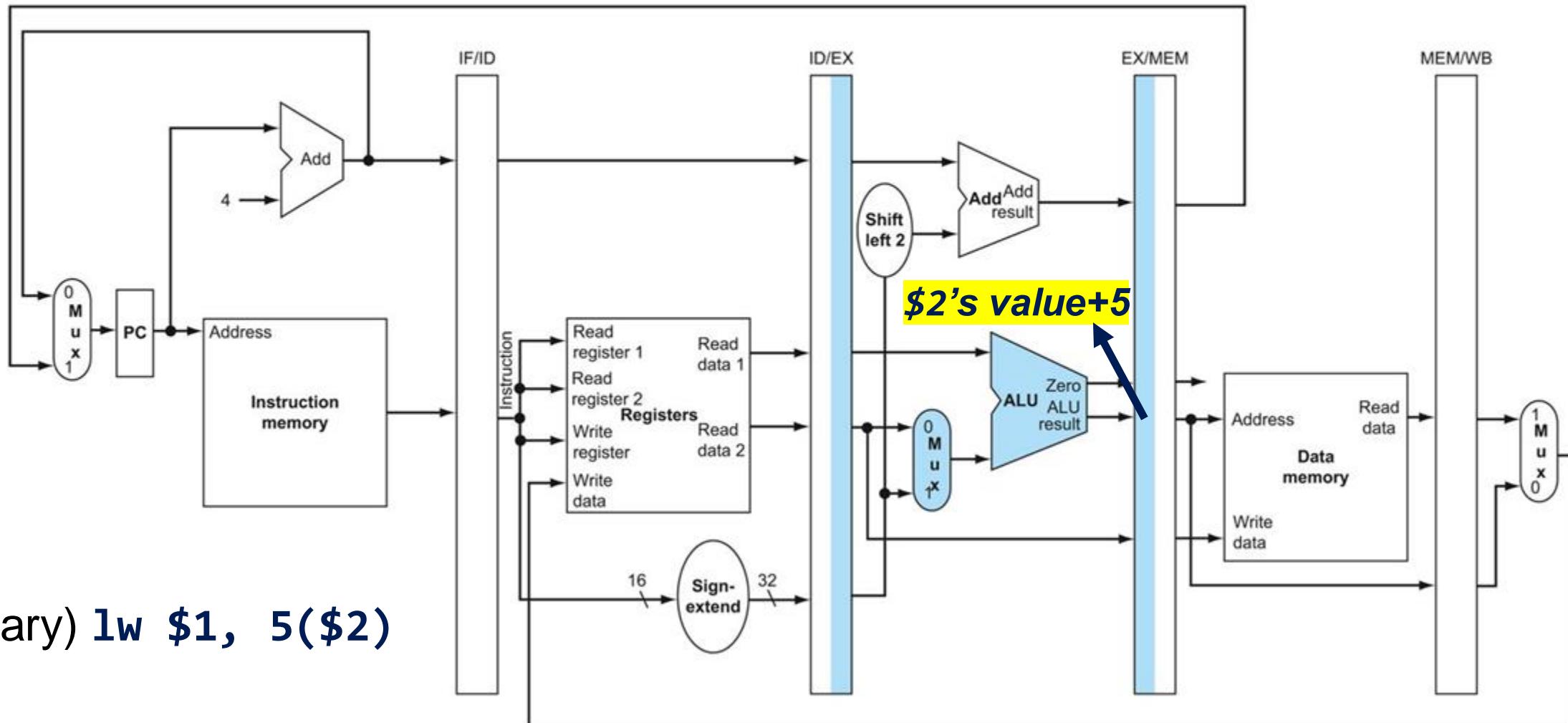


At the same time, the following instruction (e.g., instruction #2) is executed in the IF stage



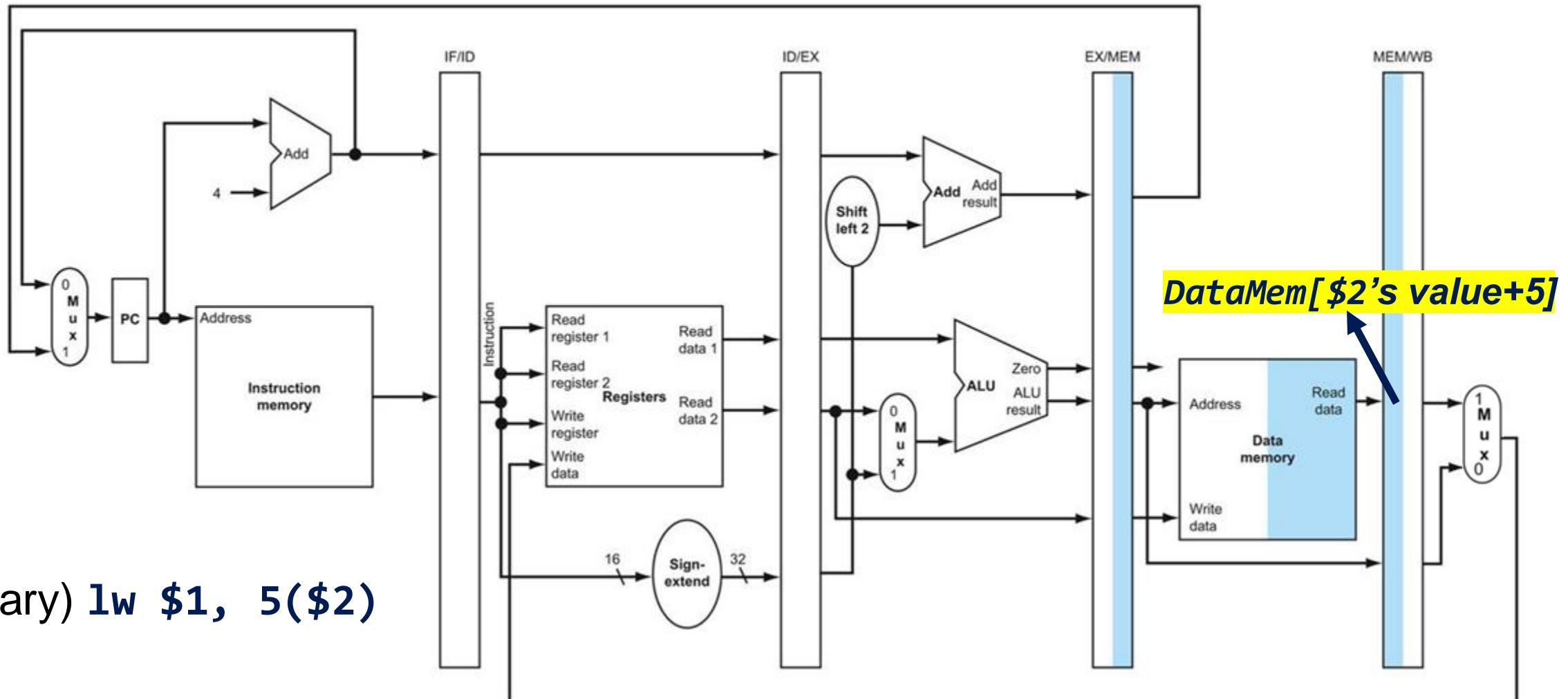
The Pipelined Version of the Datapath

39



The Pipelined Version of the Datapath

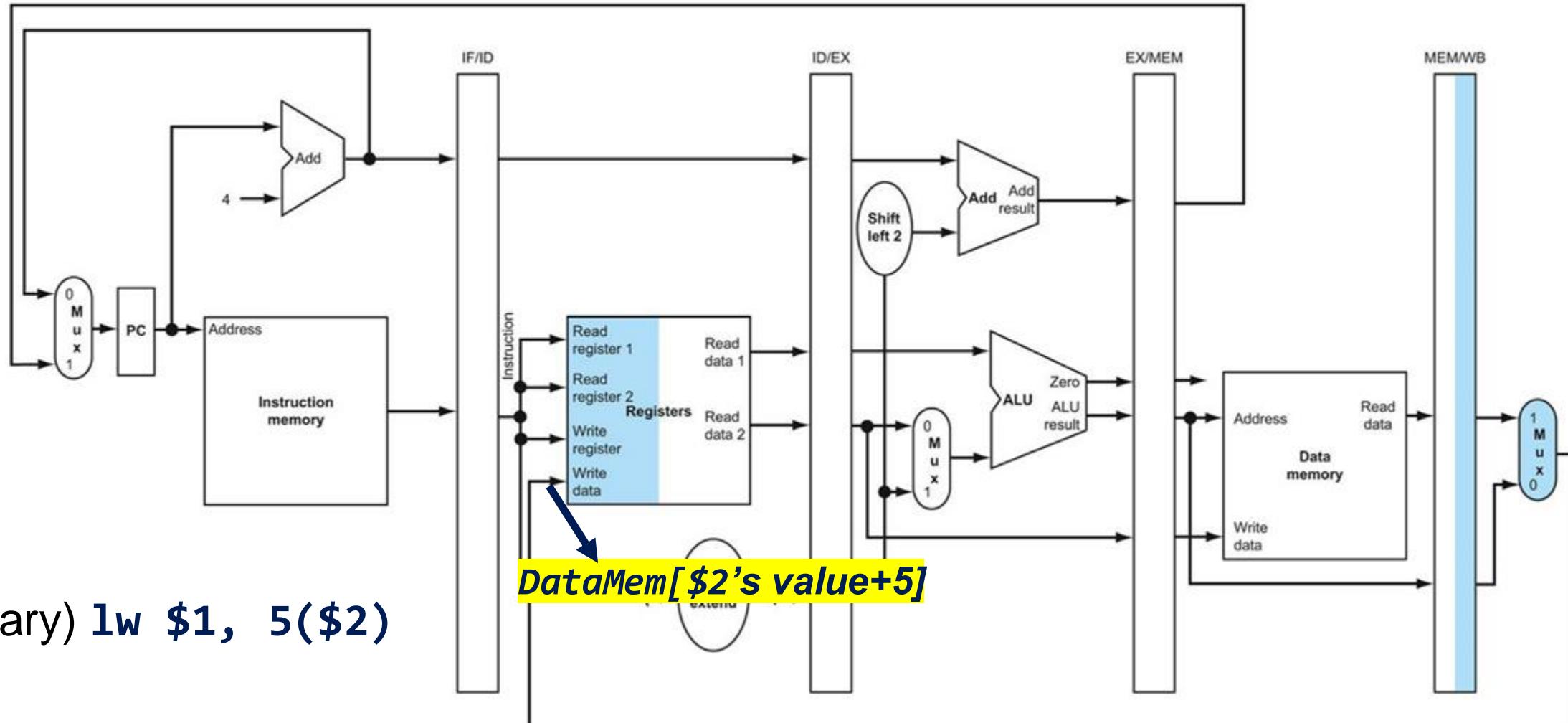
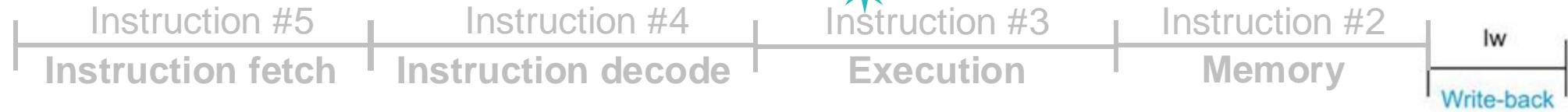
40



(binary) `Iw $1, 5($2)`

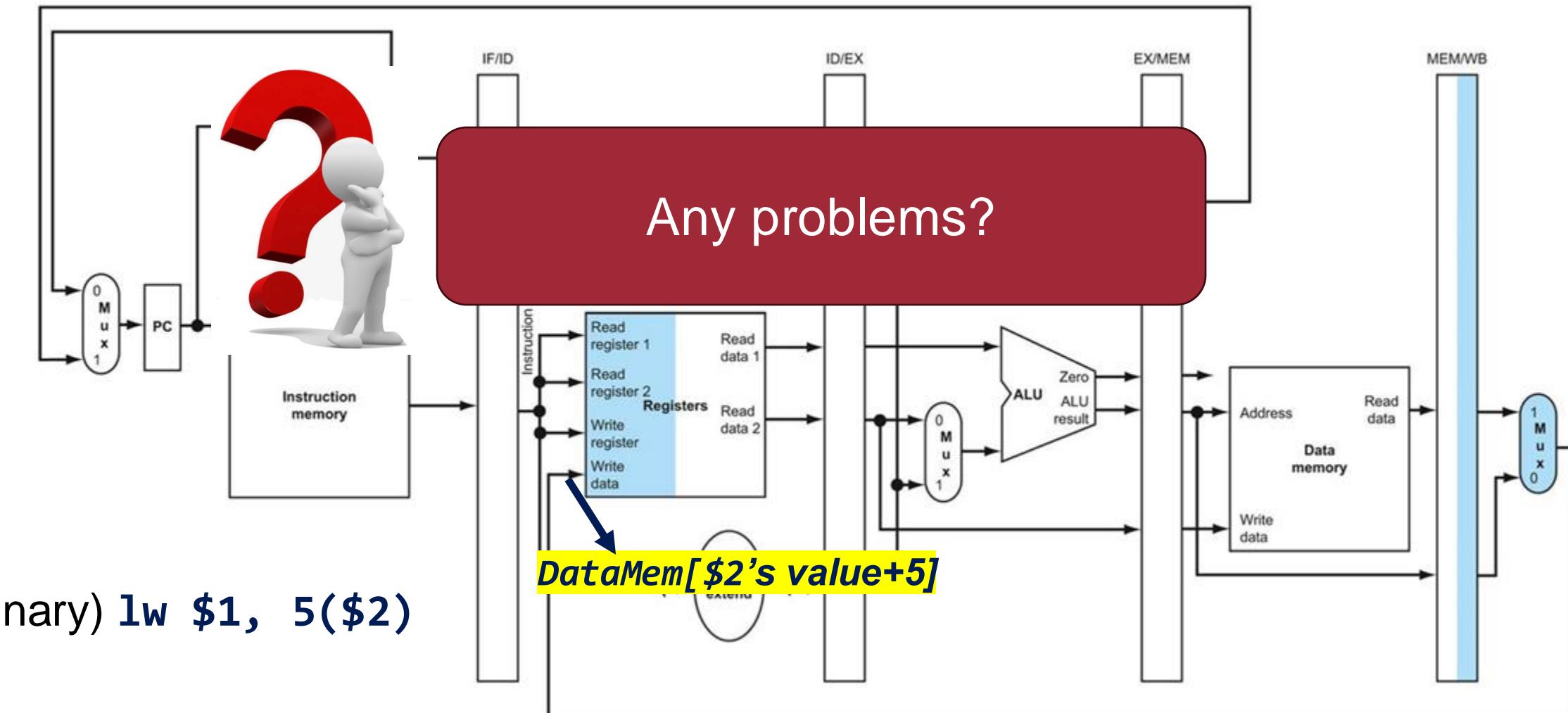
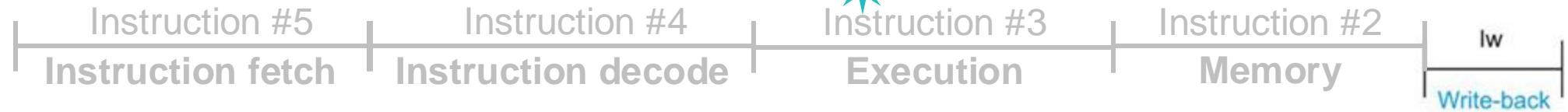
The Pipelined Version of the Datapath

41



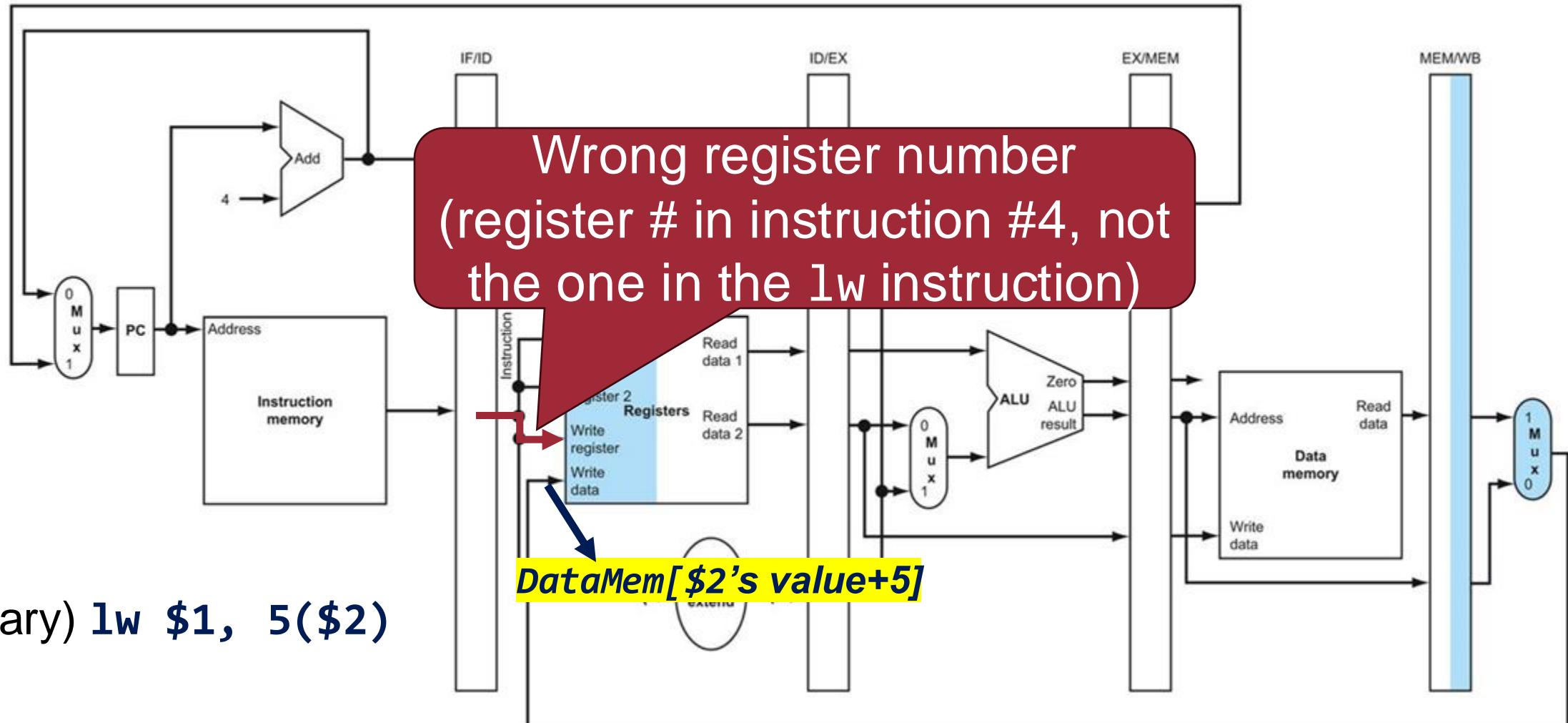
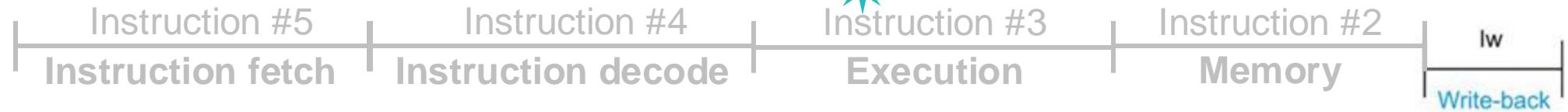
The Pipelined Version of the Datapath

42



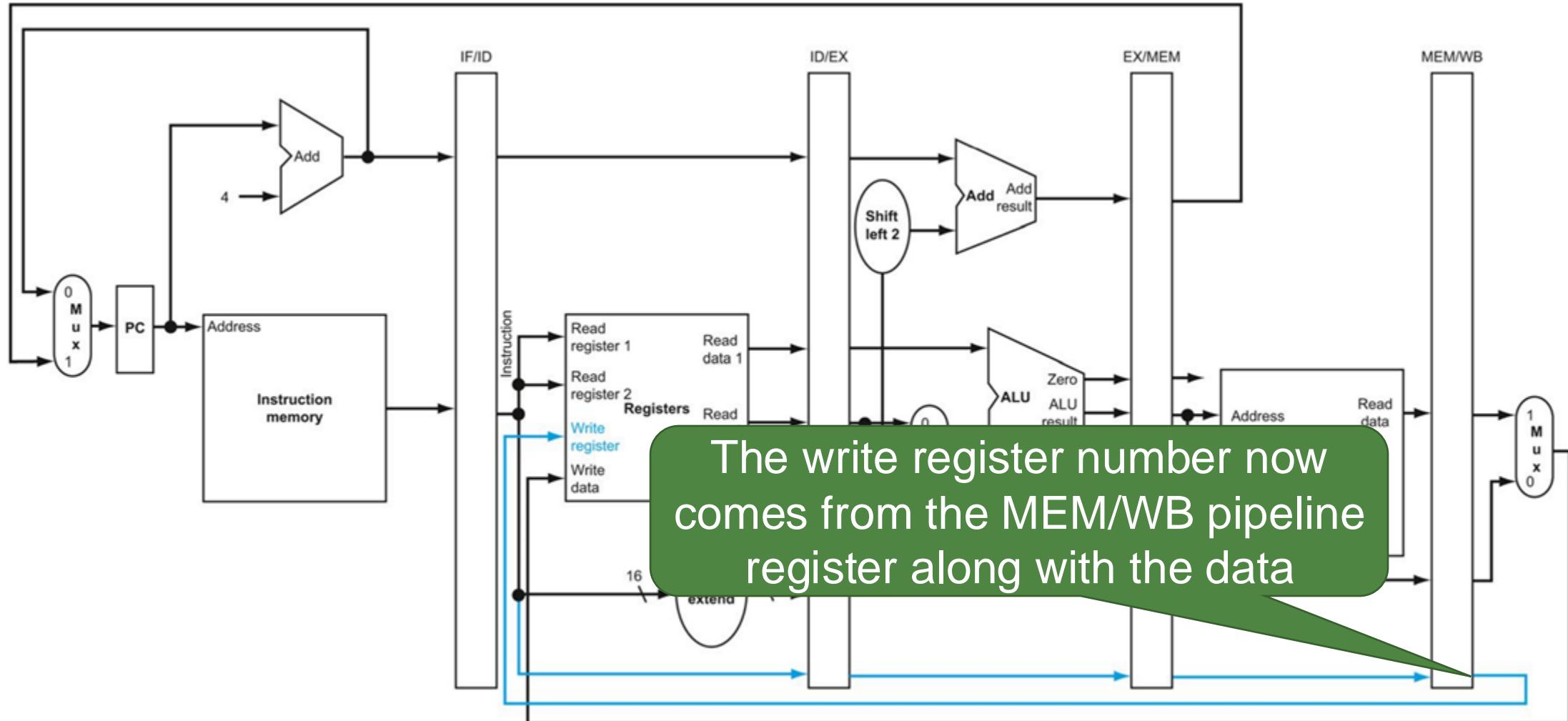
The Pipelined Version of the Datapath

43



Solution: Corrected Datapath for Load

44



Pipelining Example

45

```
lw    $10, 20($1)
sub   $11, $2, $3
add   $12, $3, $4
lw    $13, 24($1)
add   $14, $5, $6
```

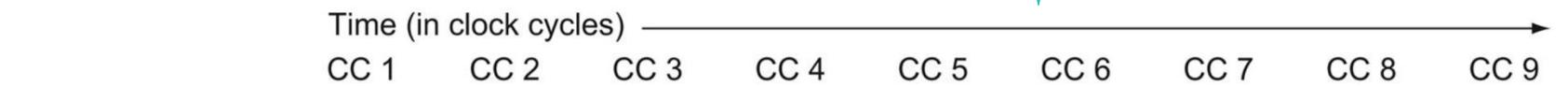


Assumption: there is no hazard

Later, we will cover about hazards!

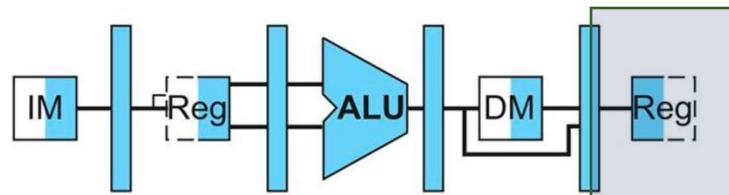
Pipeline Diagram of Five Instructions

46

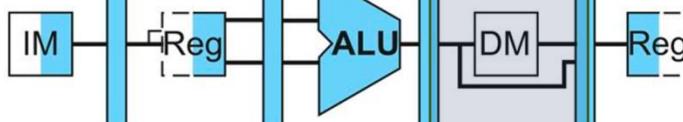


Program
execution
order
(in instructions)

lw \$10, 20(\$1)



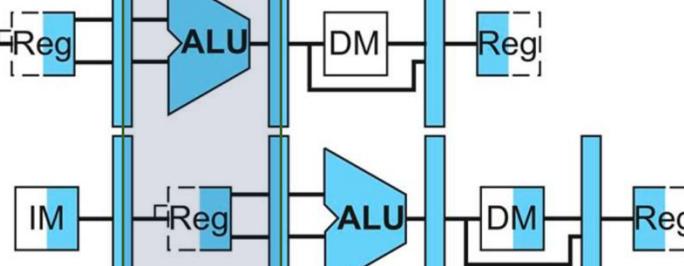
sub \$11, \$2, \$3



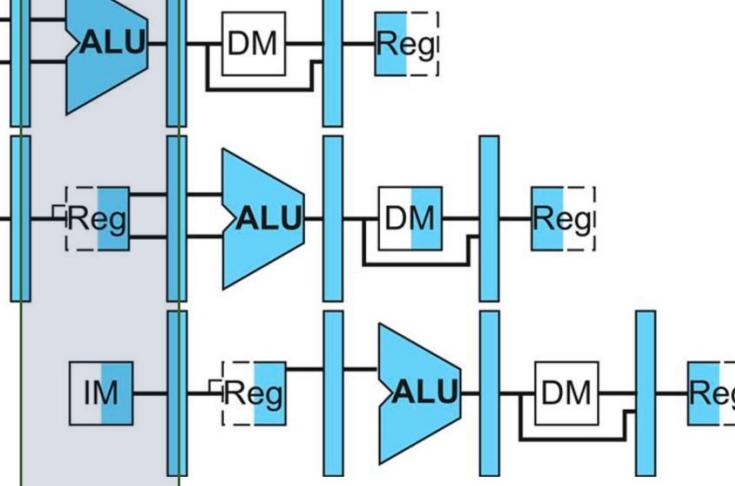
add \$12, \$3, \$4



lw \$13, 24(\$1)



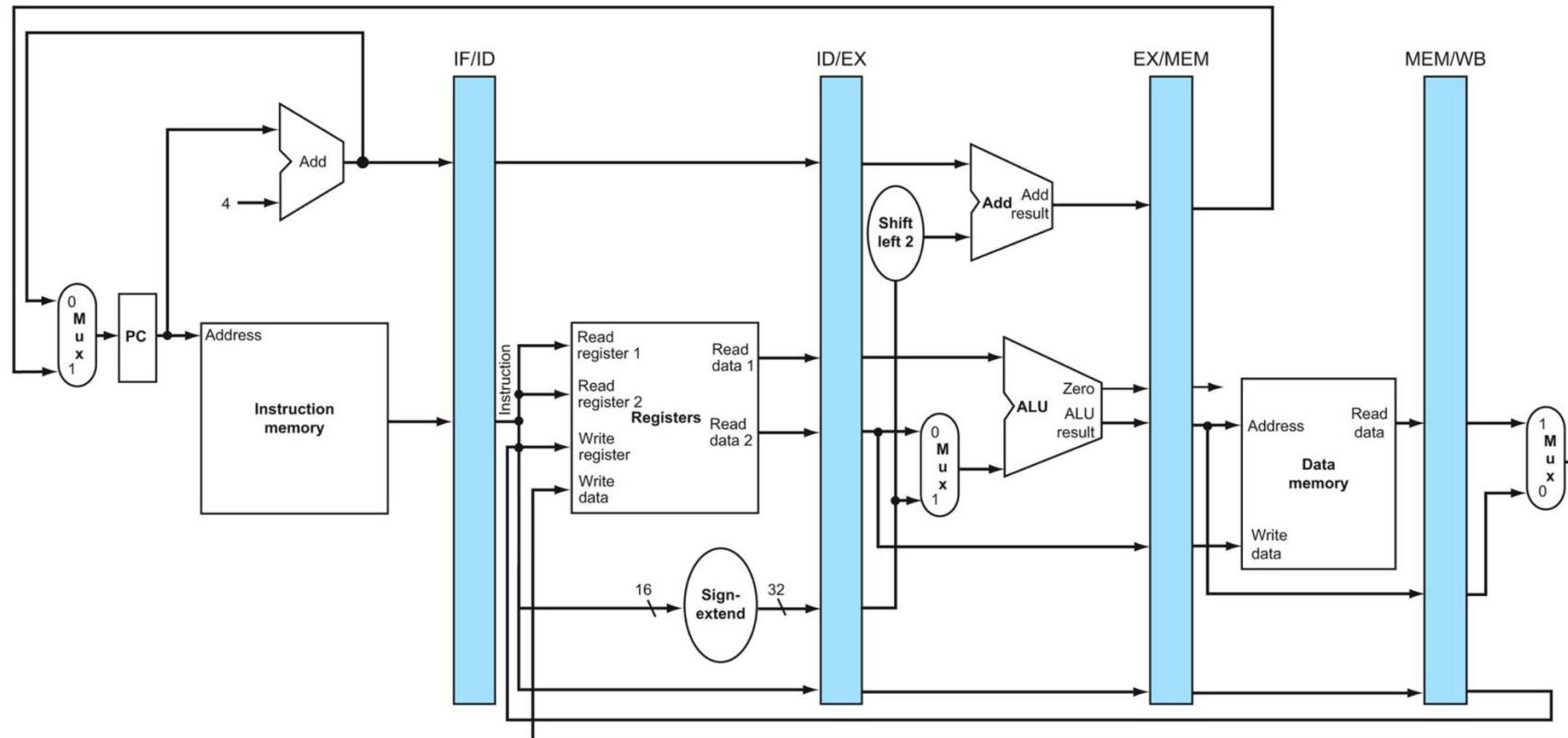
add \$14, \$5, \$6



At the cycle 5, we can fully
utilize the hardware resources

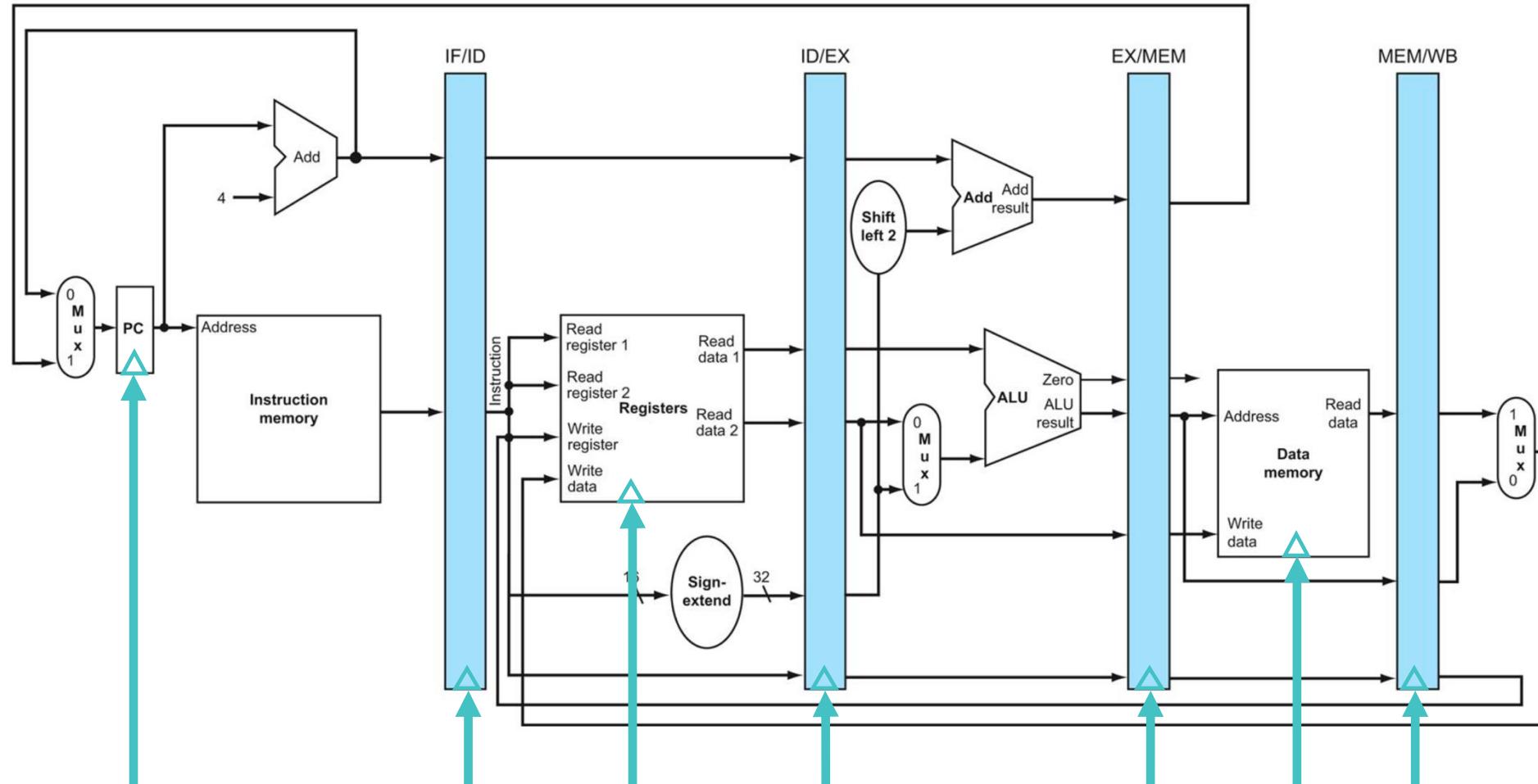
Clock Cycle 5 of the Pipeline

47



Clock Cycle 5 of the Pipeline

48

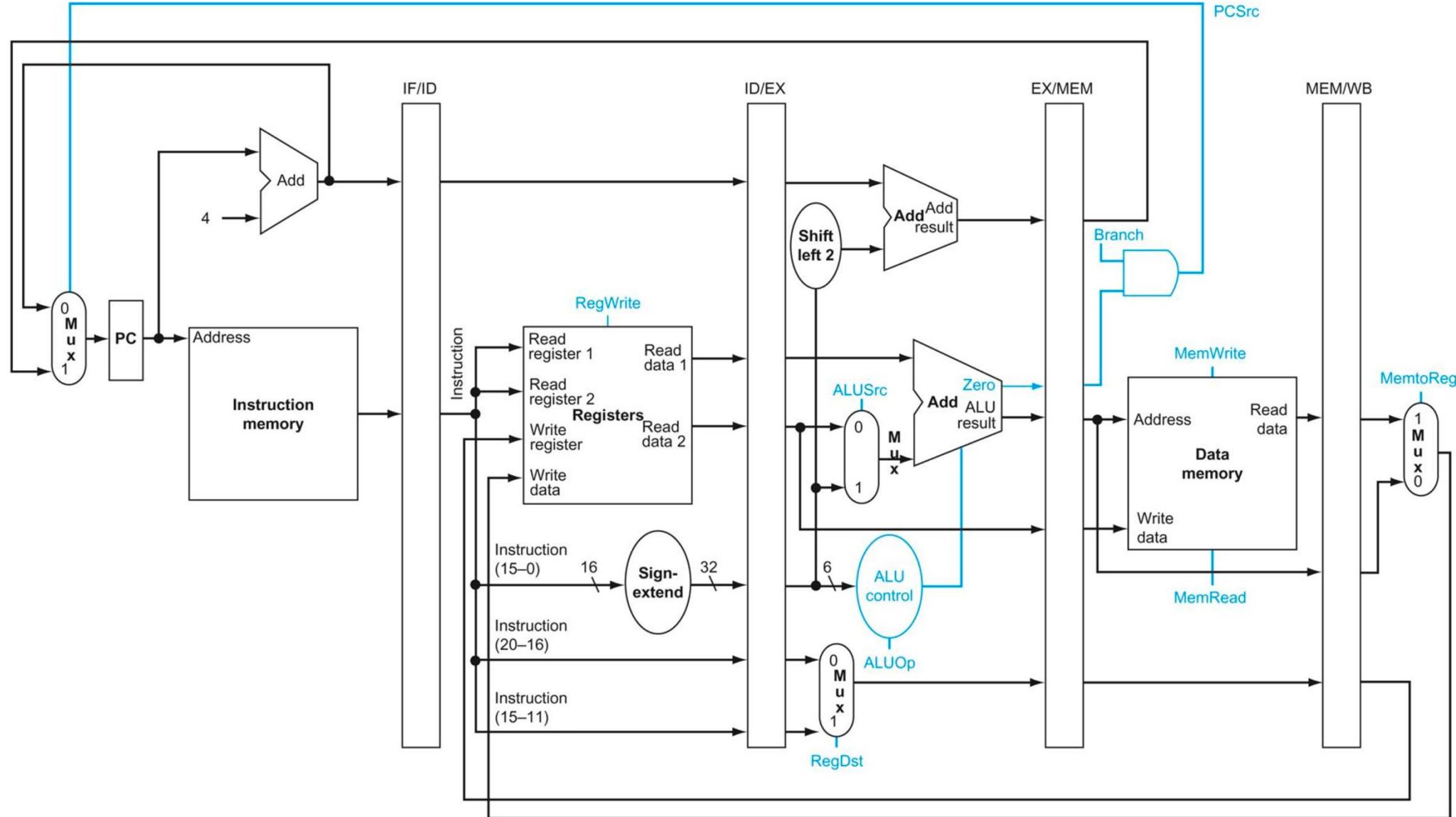


Clocks

Pipelined Control

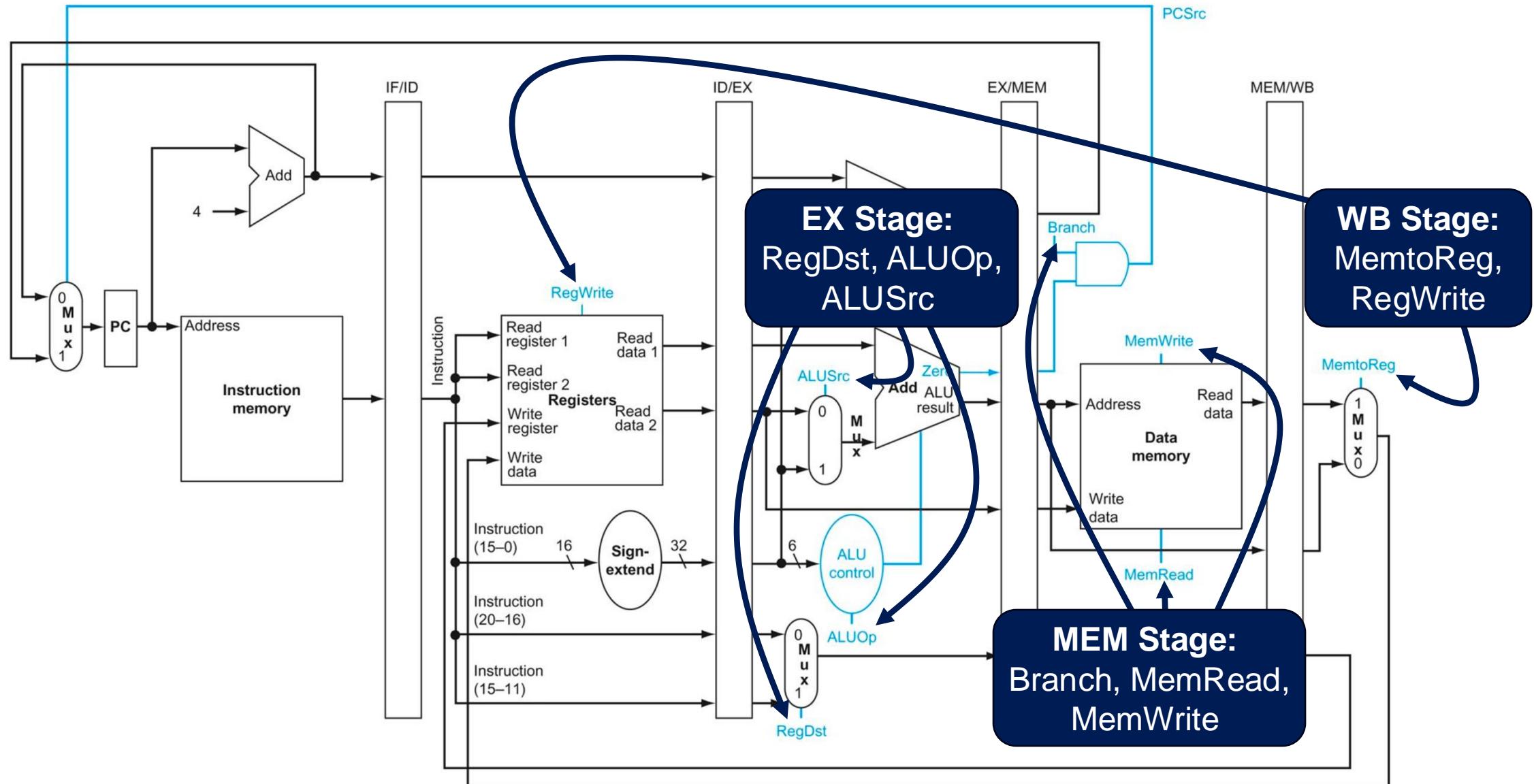
Pipeline Control Signals

50



Pipeline Control Signals

51



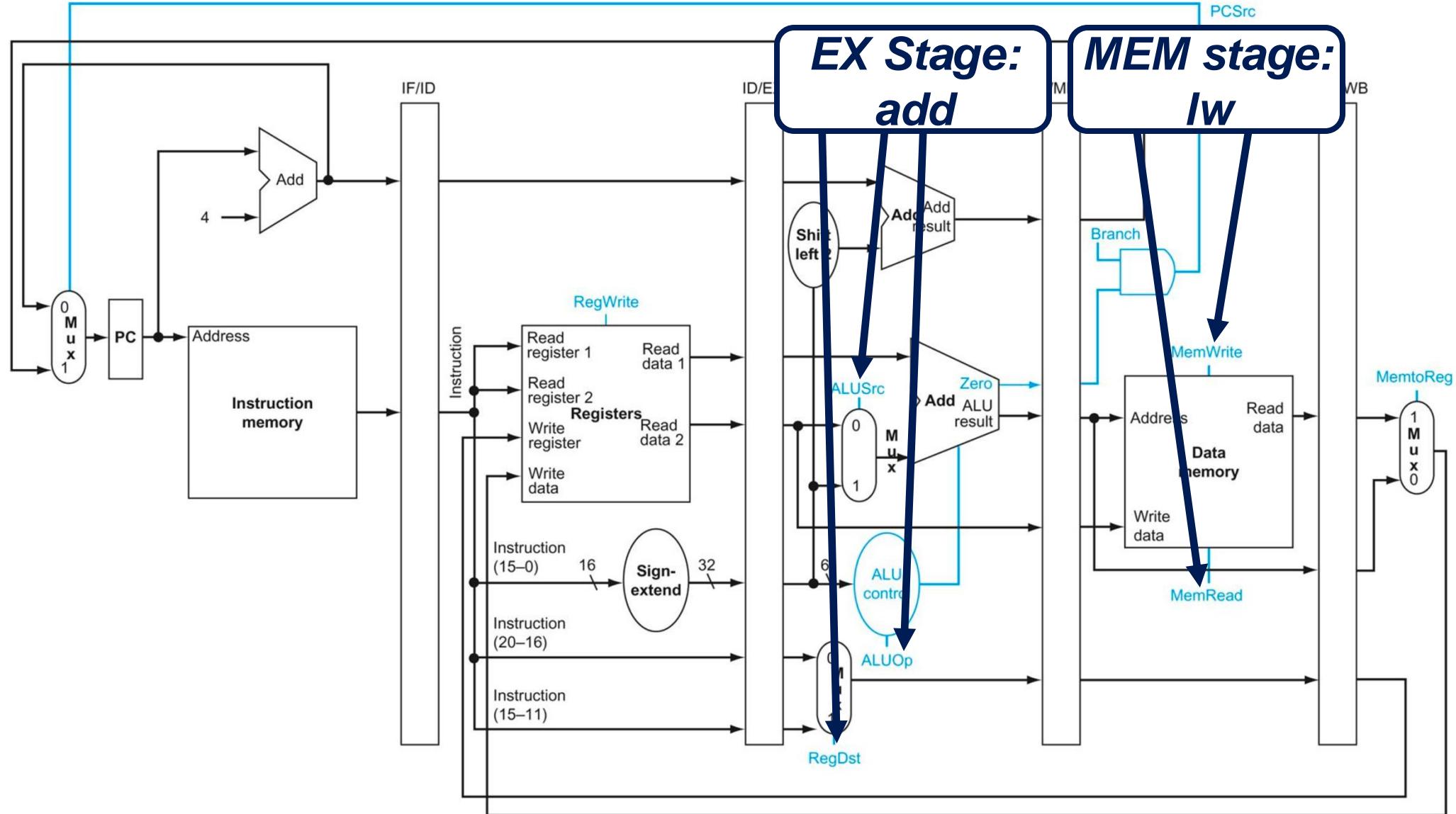
Pipeline Control Signals

52

Signal name	Effect when deasserted (0)	Effect when asserted (1)
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.

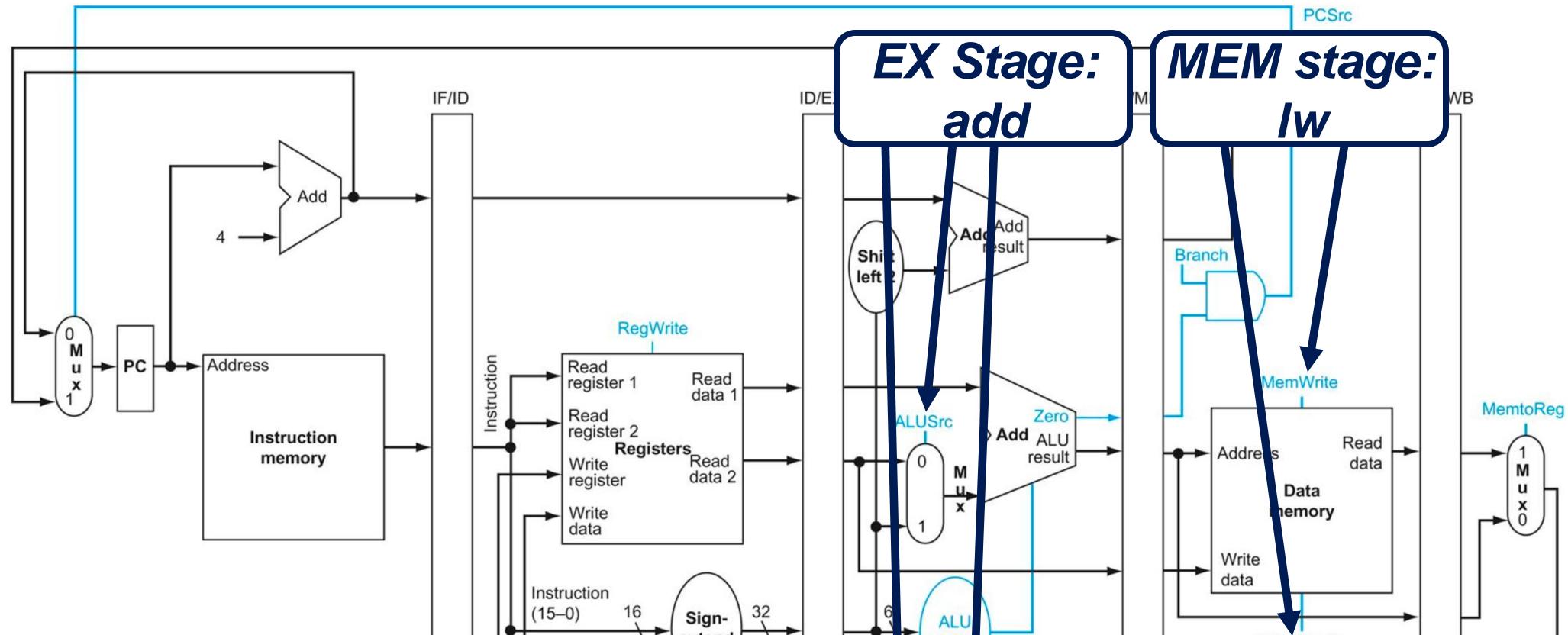
Motivation: Pipelined Control

53



Motivation: Pipelined Control

54



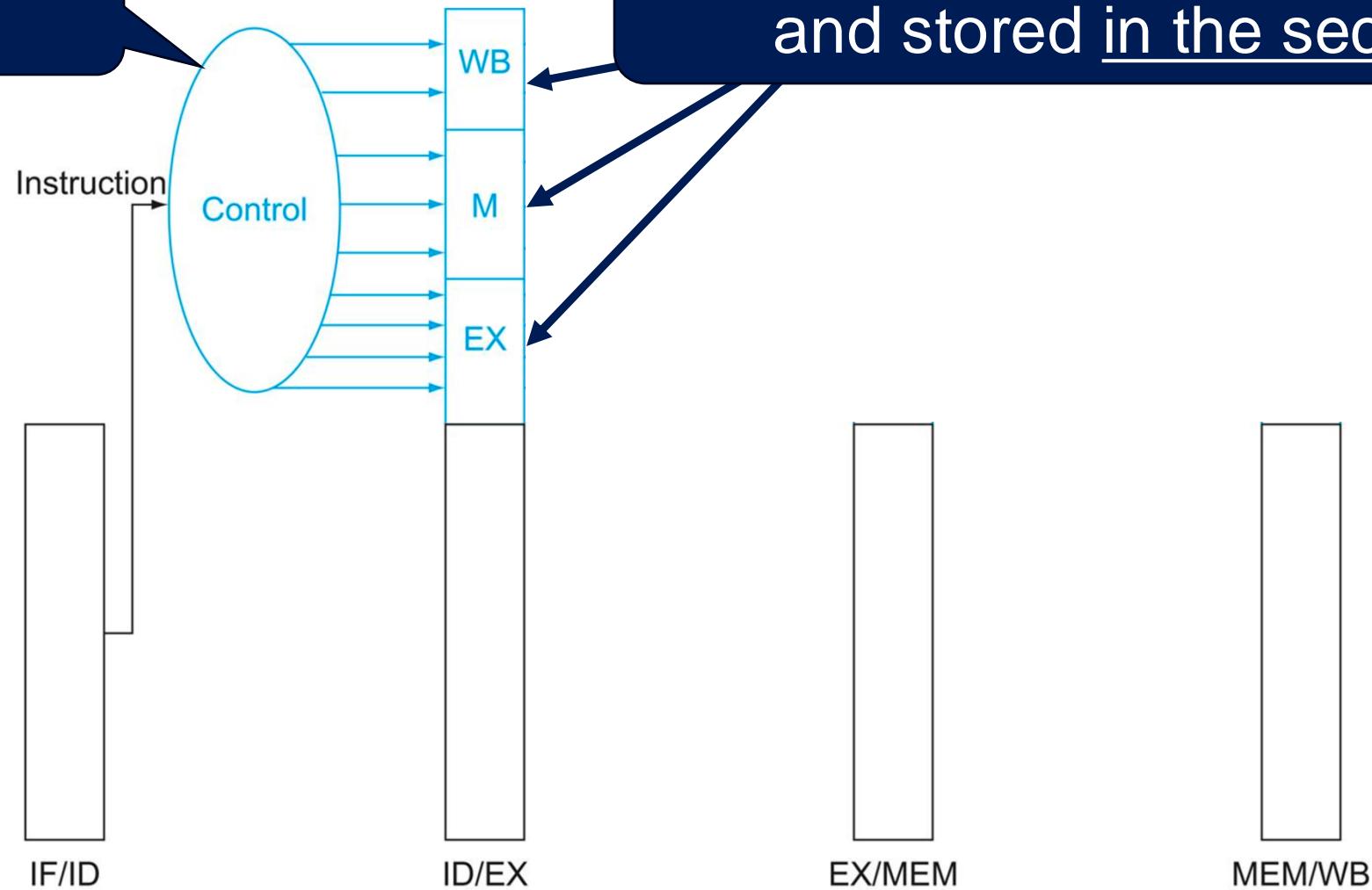
- Instructions are executed simultaneously at each stage
 - The required control signals differ for each stage
- So, How do we generate and deliver these signals?

Idea: Pipelined Control

55

Combinational circuit

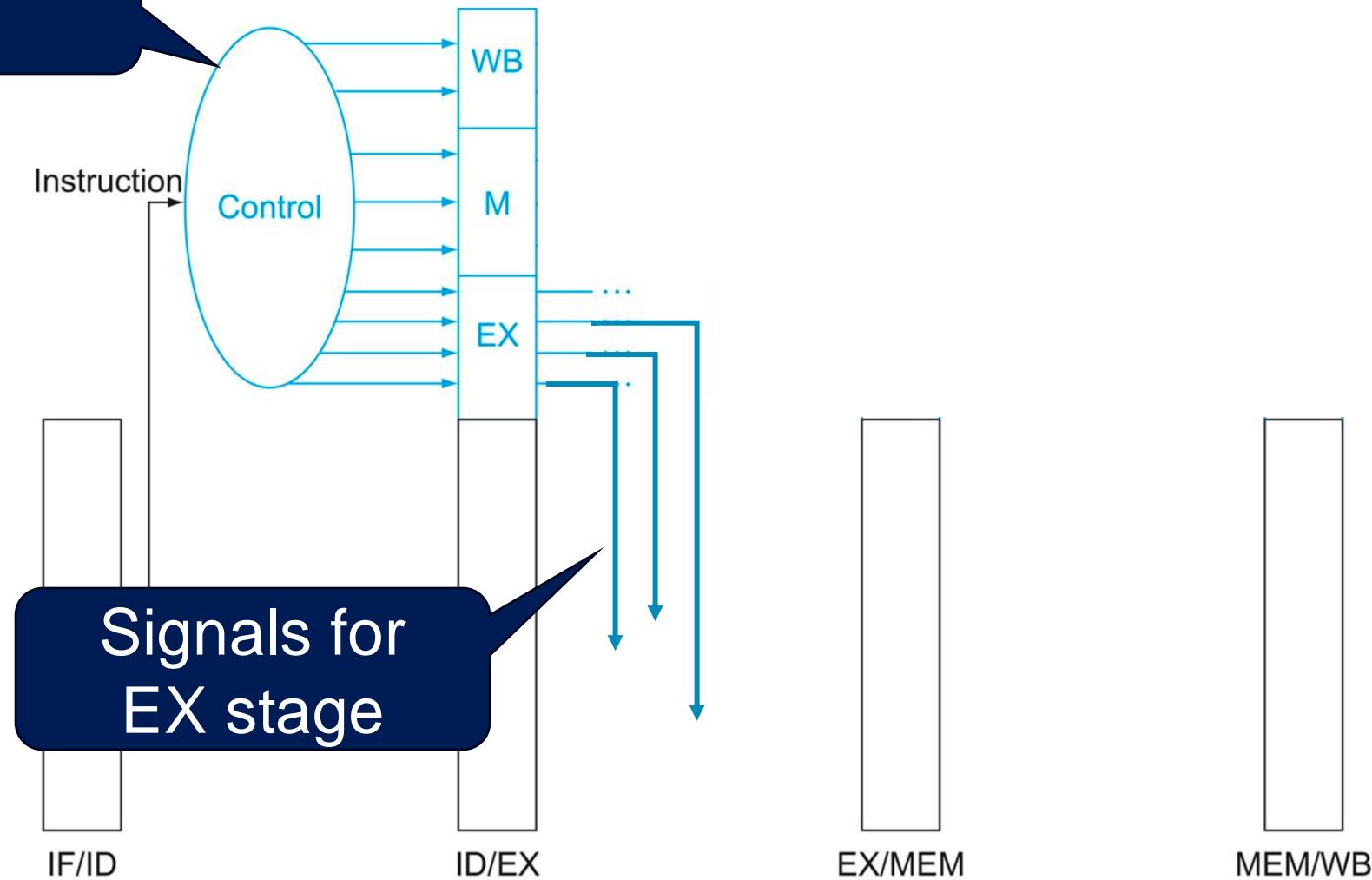
The signals for all stages are generated and stored in the second stage



Idea: Pipelined Control

56

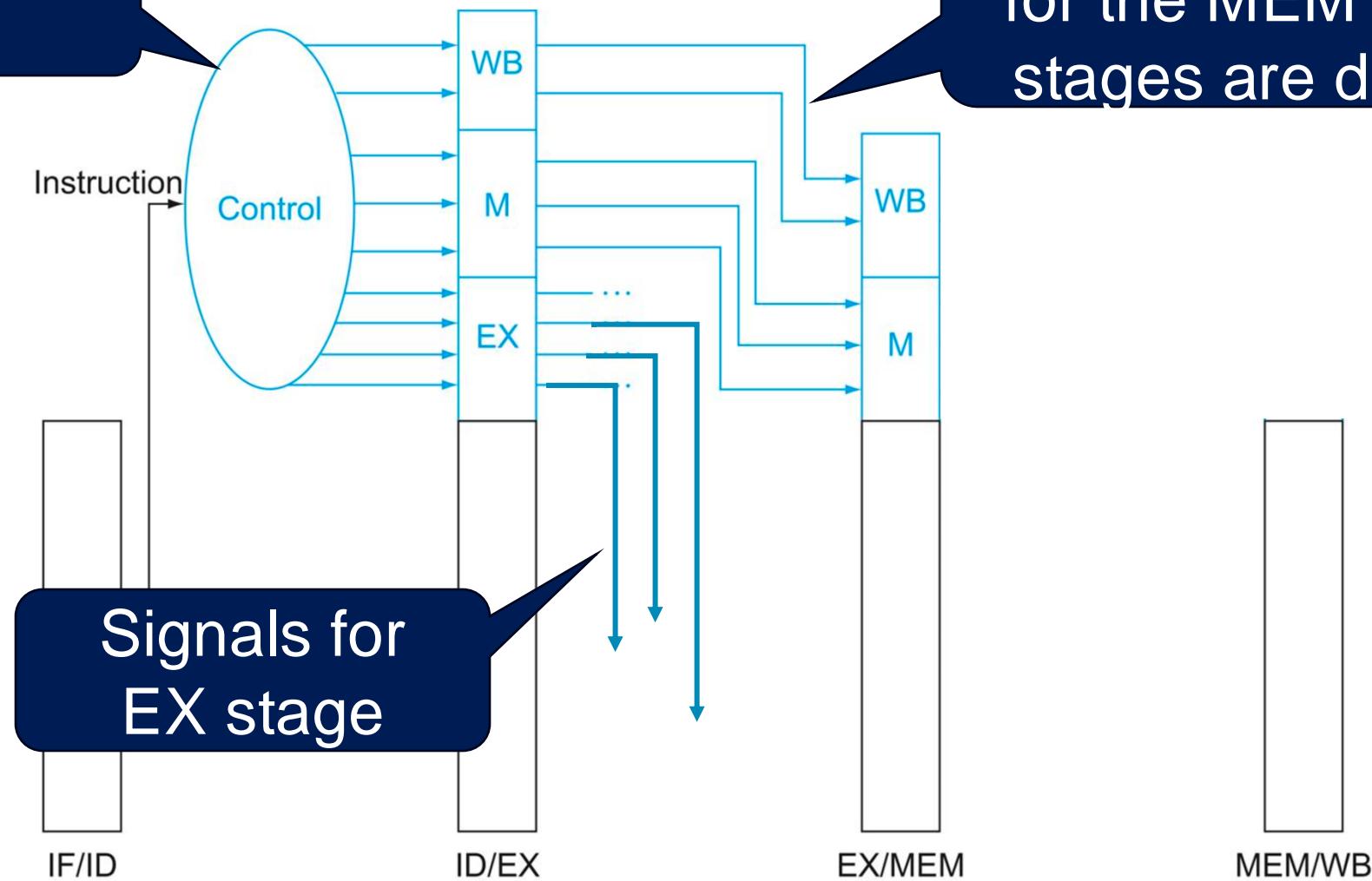
Combinational
circuit



Idea: Pipelined Control

57

Combinational circuit



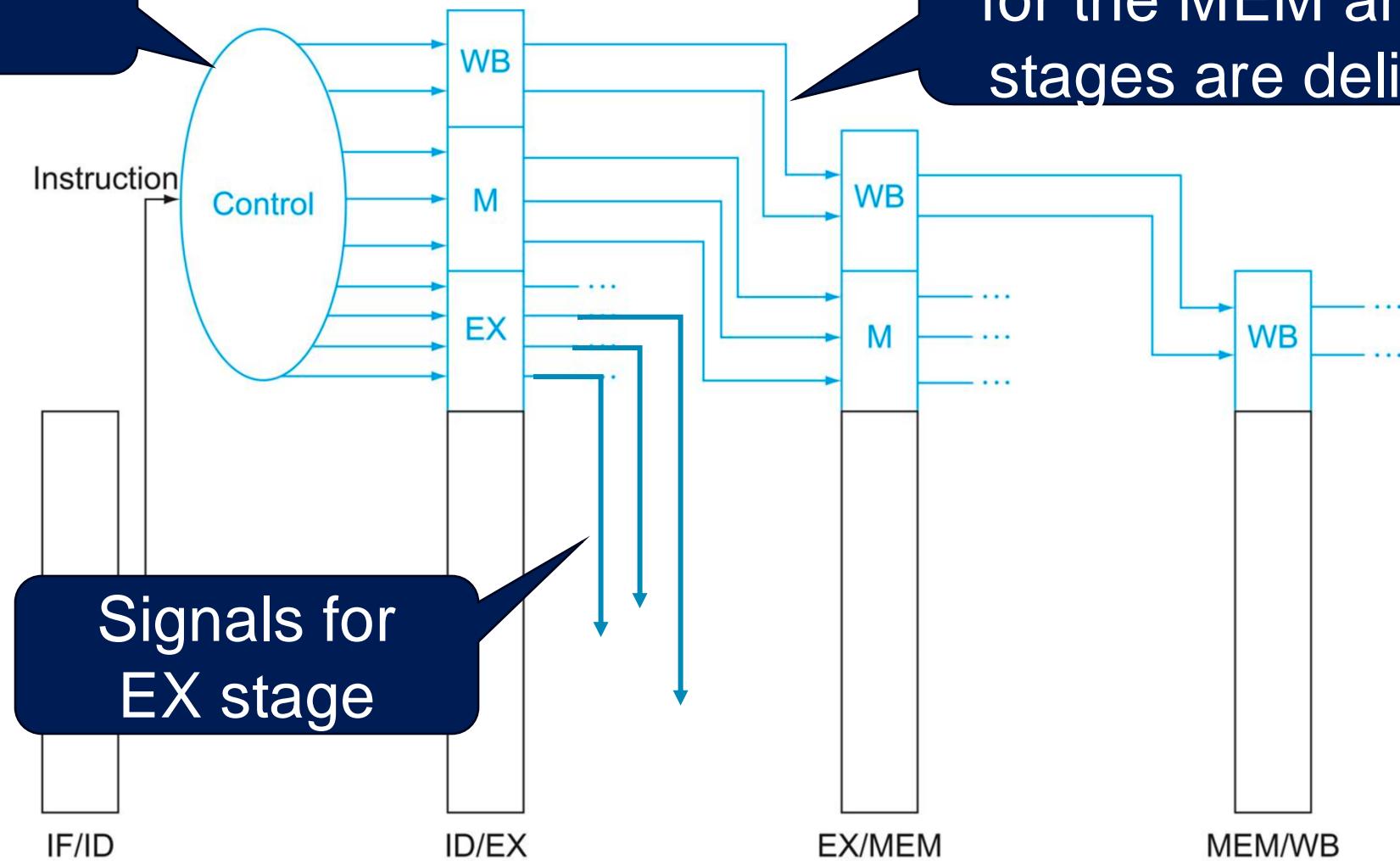
The signals required for the MEM and WB stages are delivered

Signals for EX stage

Idea: Pipelined Control

58

Combinational circuit



The signals required for the MEM and WB stages are delivered

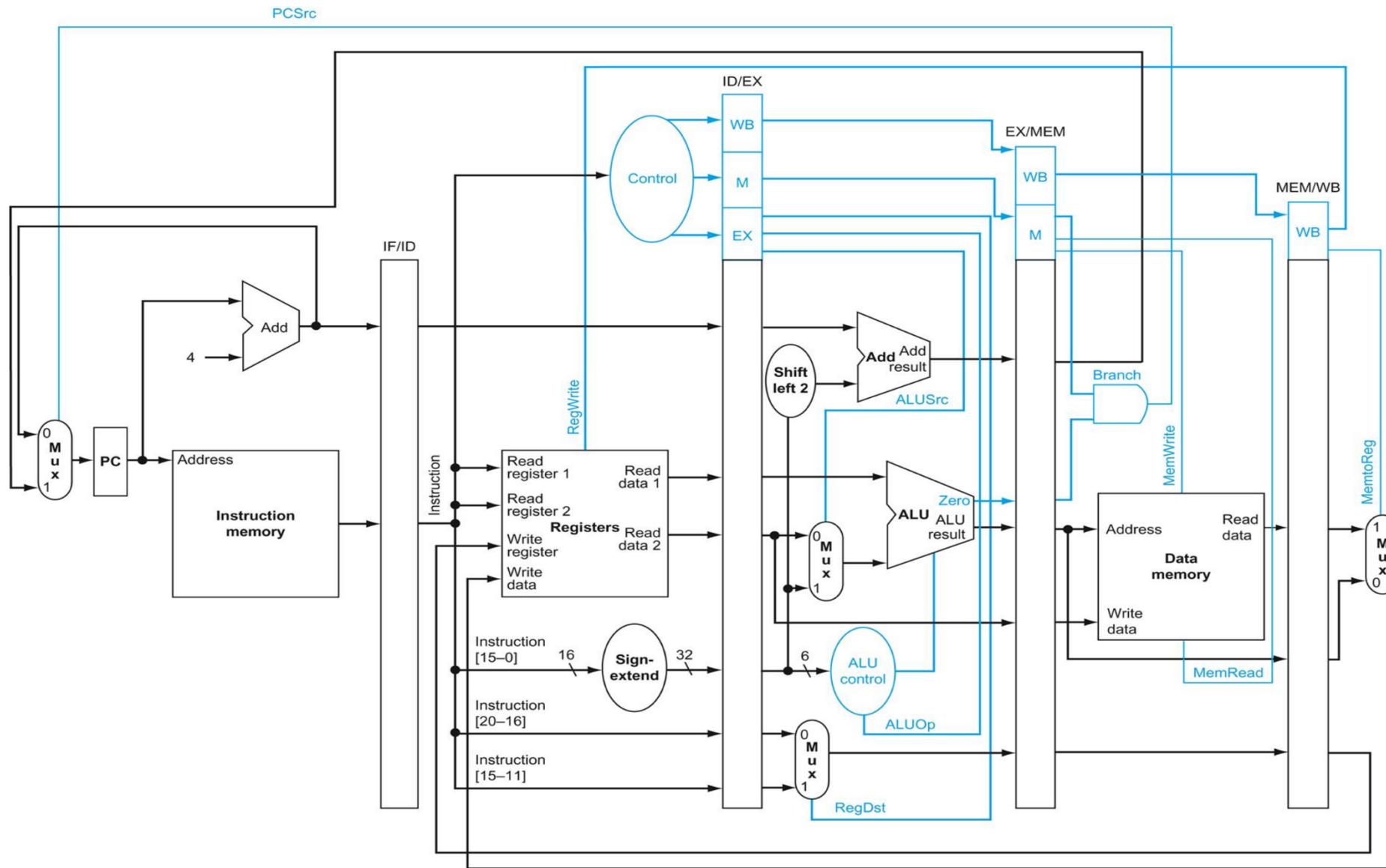
Pipelined Control Signals

59

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

The Pipelined Datapath with the Control

60



Pipelining Example

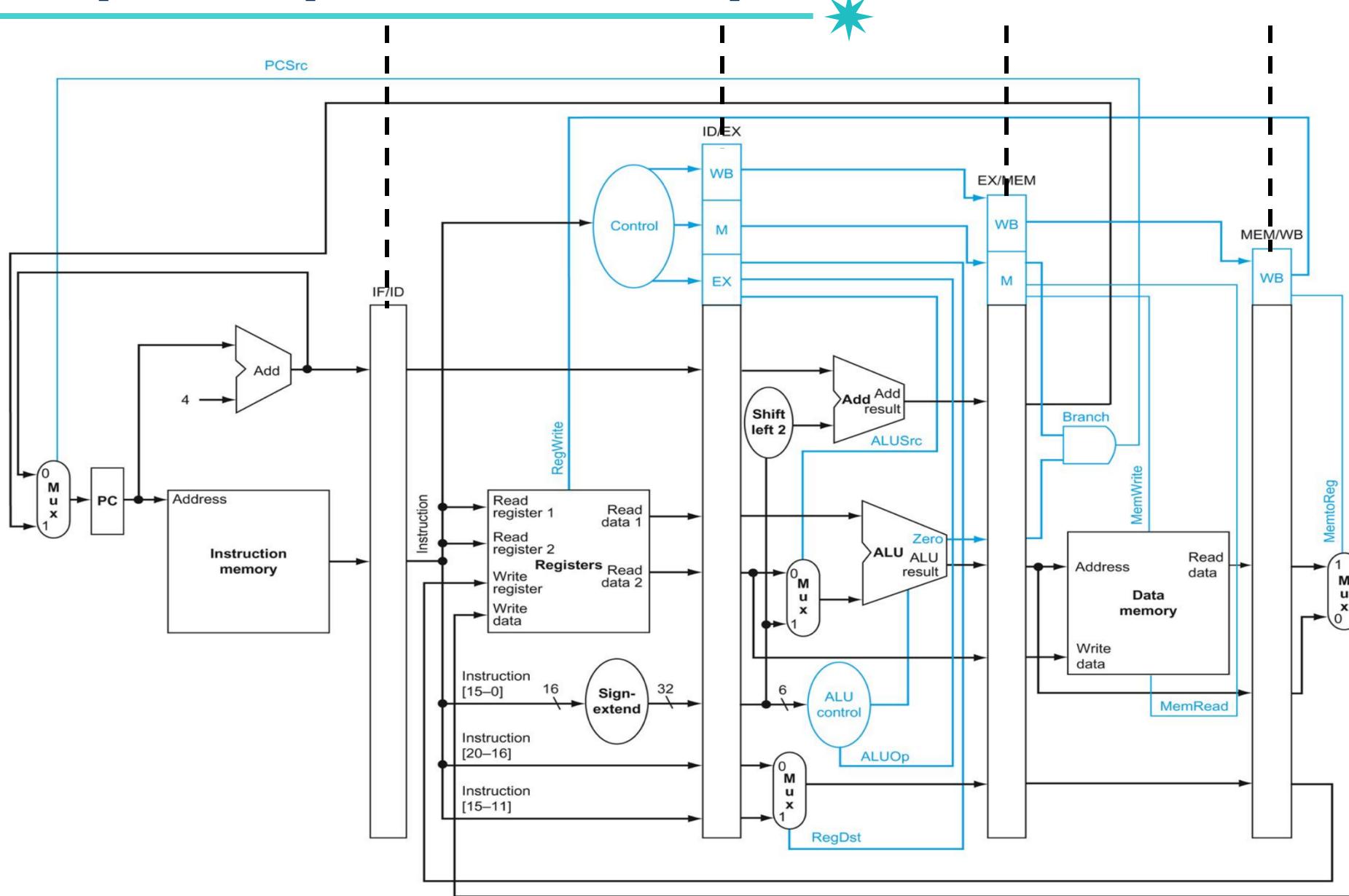


```
lw    $10, 20($1)
sub   $11, $2, $3
add   $12, $4, $5
or    $13, $6, $7
add   $14, $8, $9
```

Assumption: there is no hazard

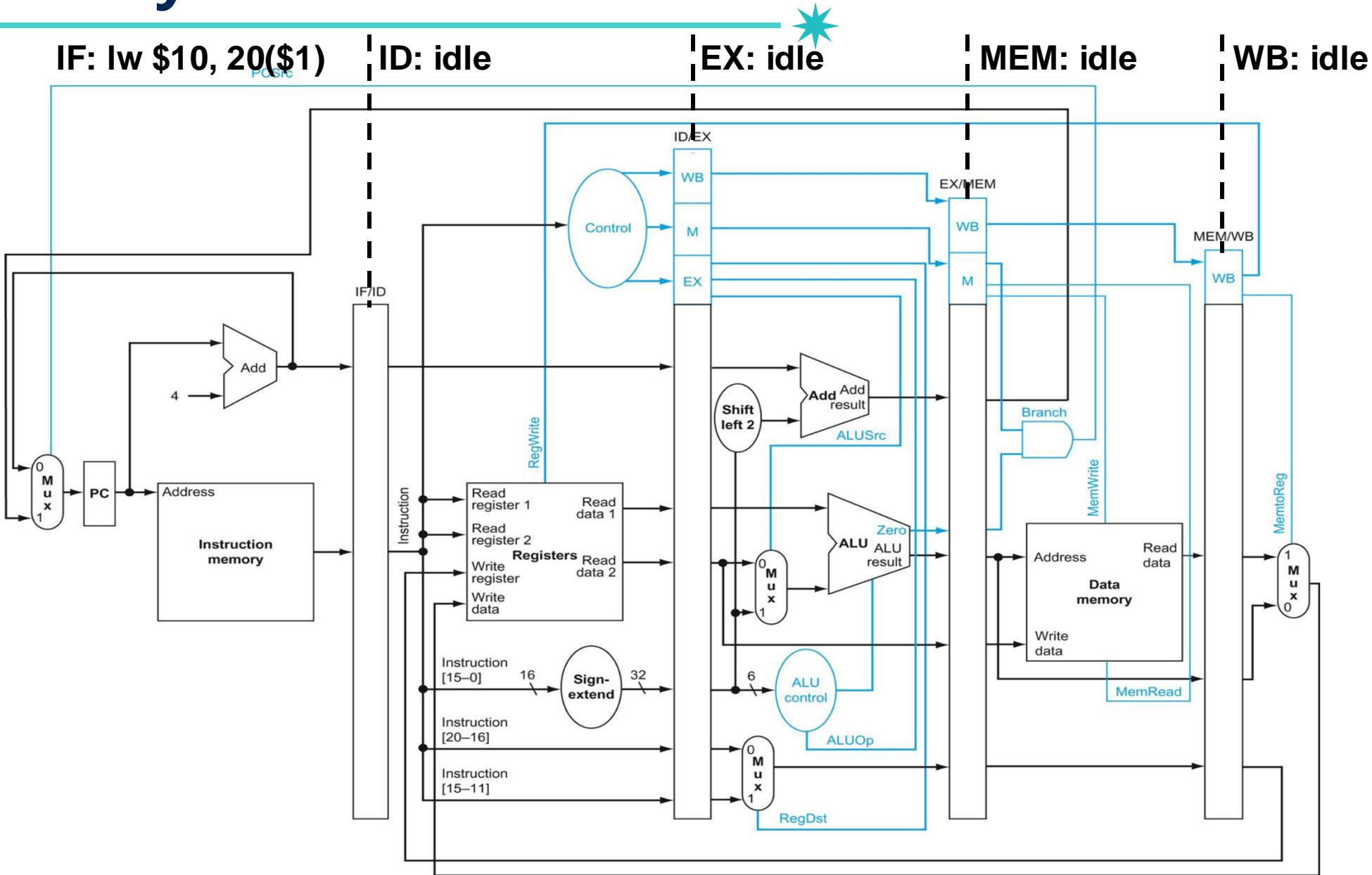
Later, we will cover about hazards!

Example: Pipelined Datapath with the Control



Clock Cycle #1

63



Clock Cycle #2

64

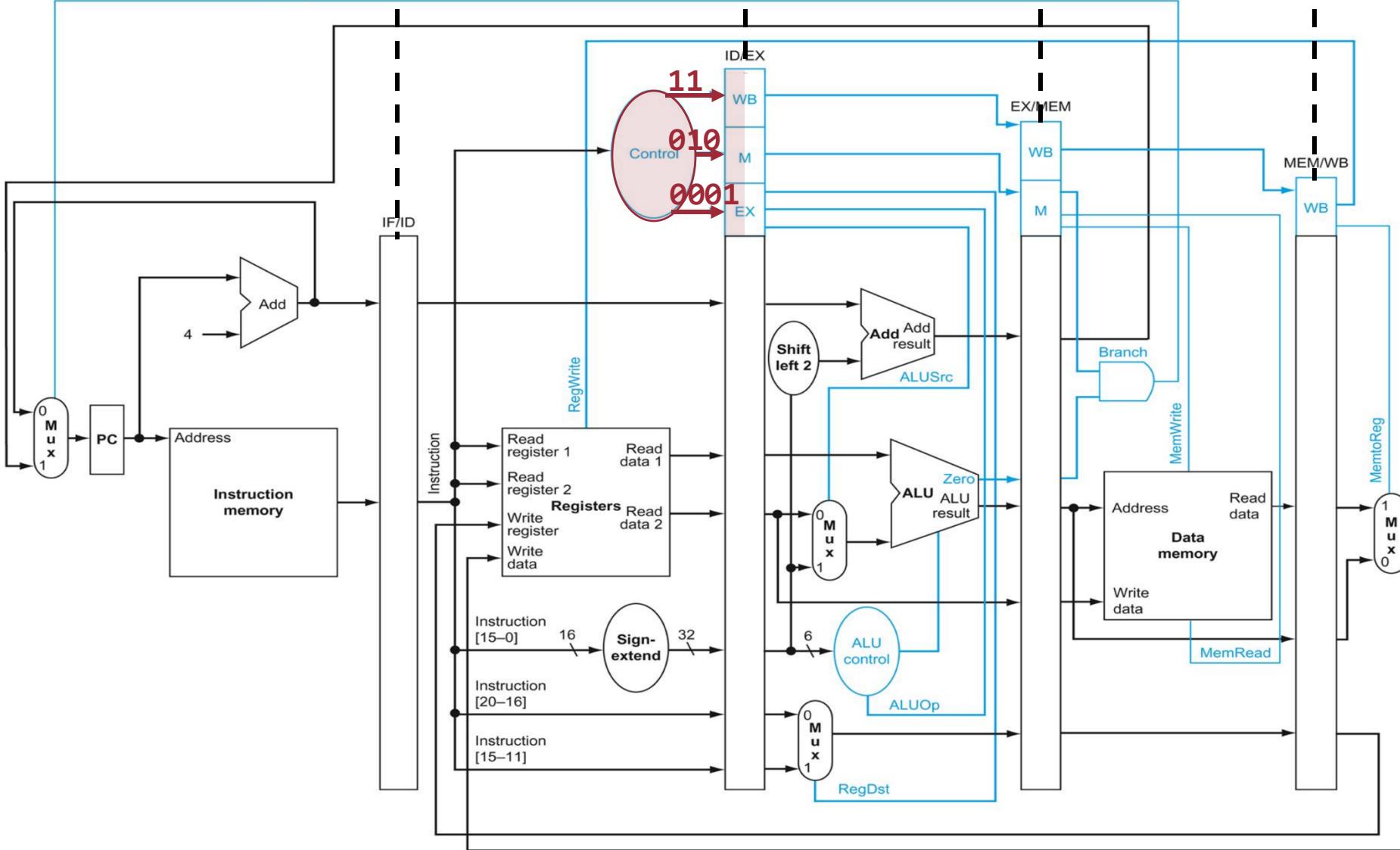
IF: sub \$11, \$2, \$3

ID: lw \$10, 20(\$1)

EX: idle

MEM: idle

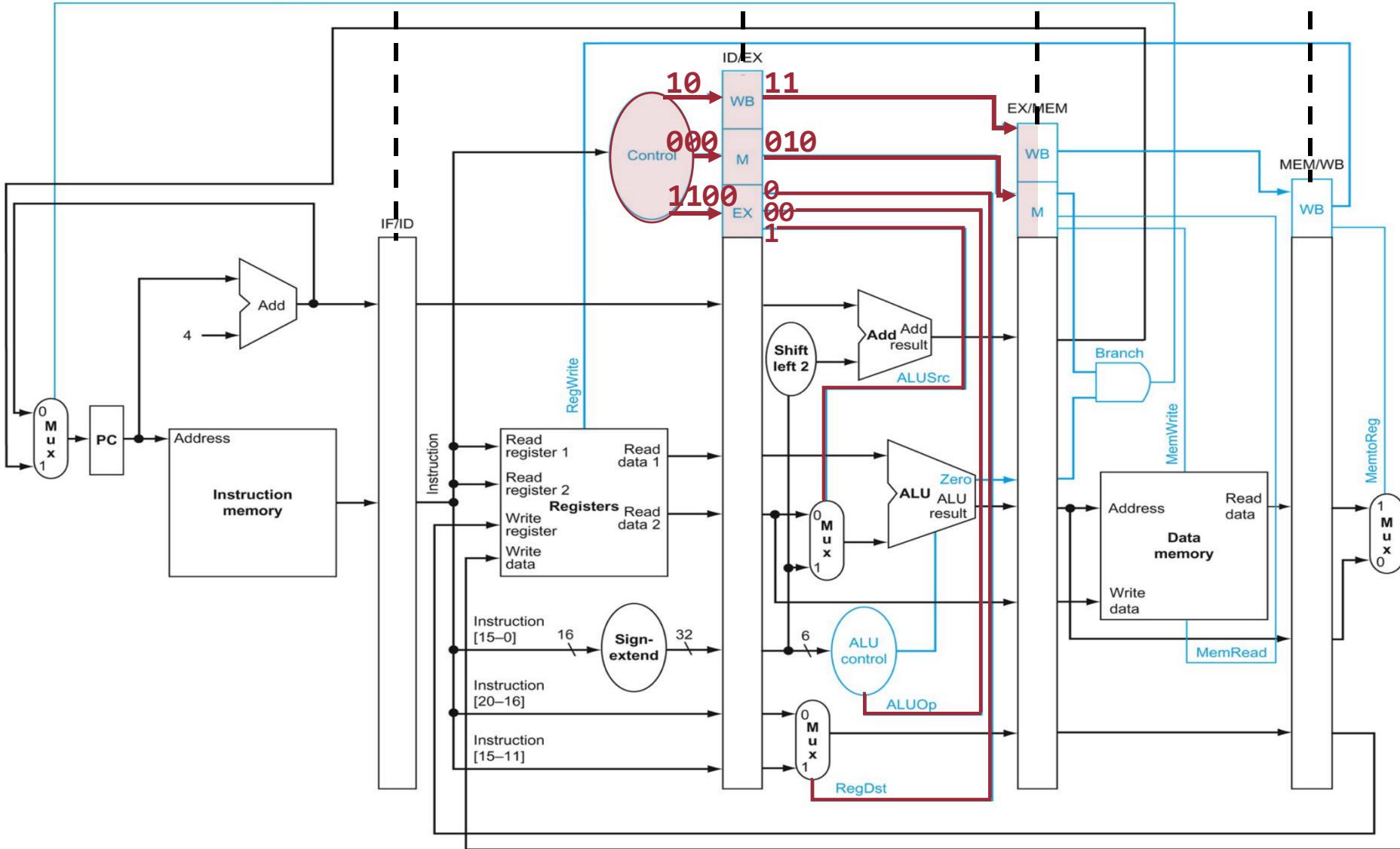
WB: idle



Clock Cycle #3

65

IF: and \$12, \$4, \$5
 PC_{src}
 ID: sub \$11, \$2, \$3
 EX: lw \$10, 20(\$1)
 MEM: idle
 WB: idle



Clock Cycle #4

66

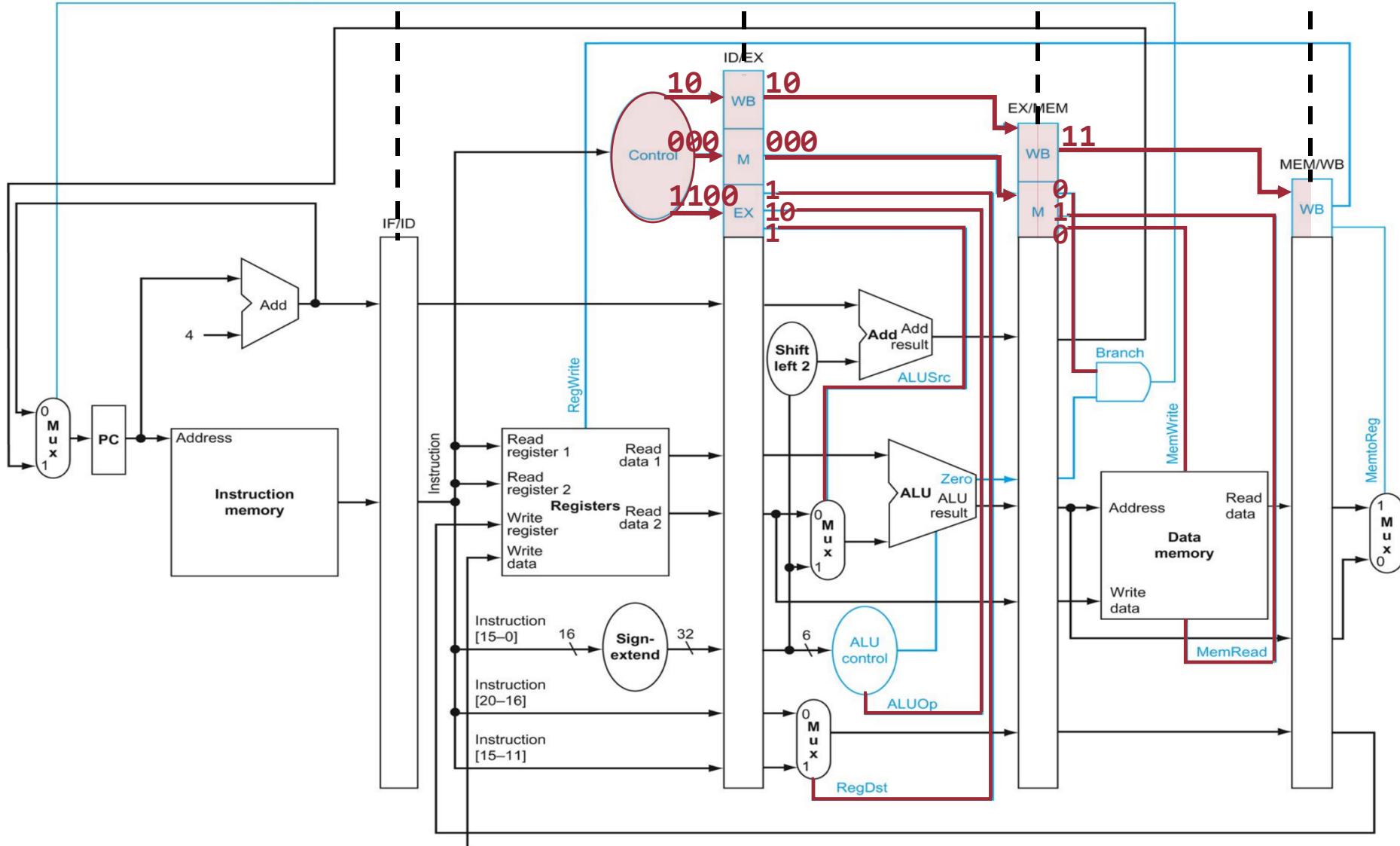
IF: or \$13, \$6, \$7
PCSrc

ID: and \$12, \$4, \$5

EX: sub\$11, \$2, \$3

MEM: lw\$10,20(\$1)

WB: idle

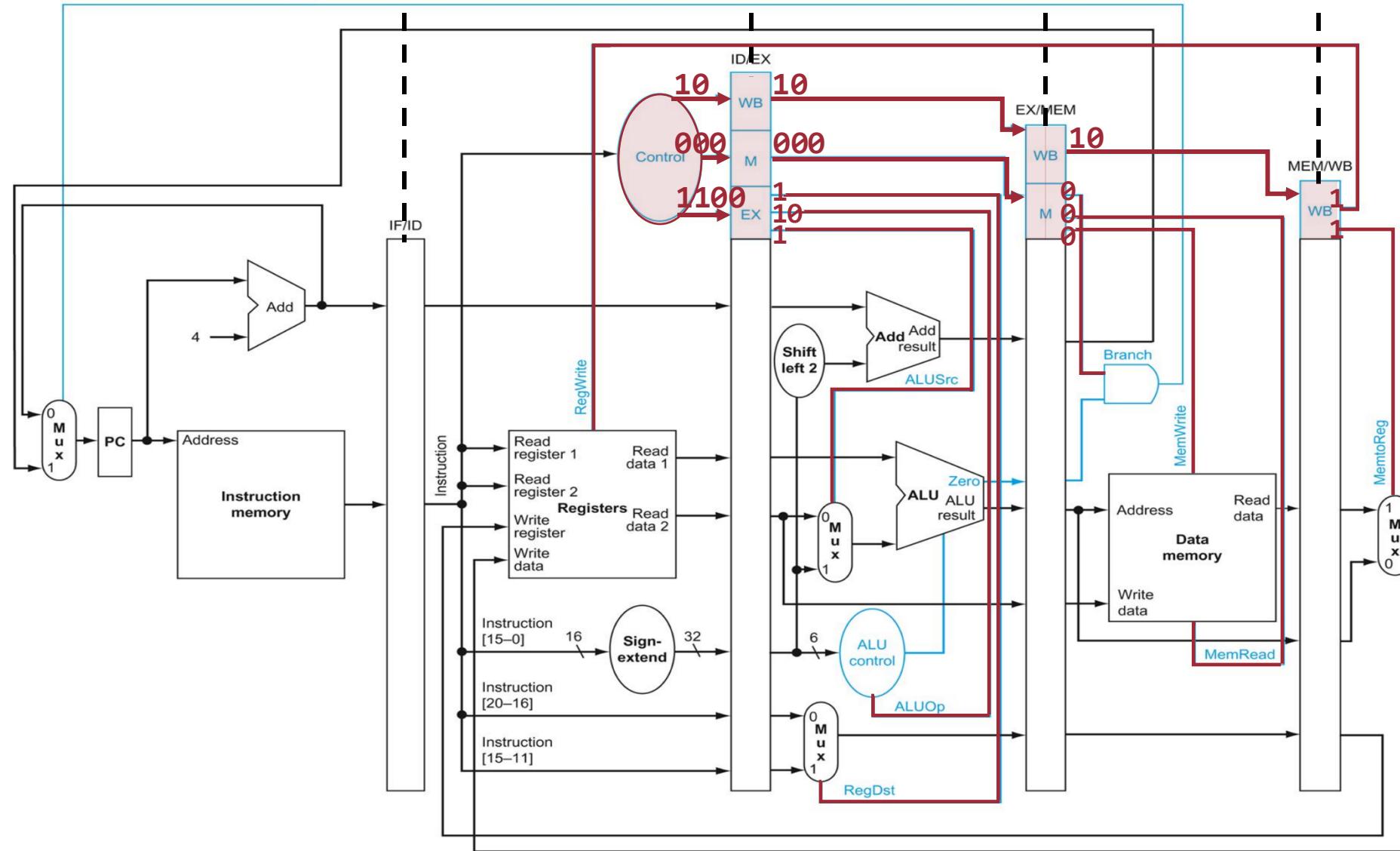


Clock Cycle #5

IF: add \$14, \$8, \$9
PCsrc

ID: or \$13, \$6, \$7

EX: and \$12, \$4, \$5
MEM: sub \$11, \$2, \$3
WB: lw \$10, 20(\$1)



Question?