

# IT5002

## Computer Systems and Applications

### Pipelining

[colintan@nus.edu.sg](mailto:colintan@nus.edu.sg)



**NUS**  
National University  
of Singapore

School *of* Computing



## Q & A

- **DO NOT** use the Zoom chat for questions. It doesn't appear in the video recordings.
- Please ask questions at <https://sets.netlify.app/module/61597486a7805d9fb1b4accd>



← OR scan this QR code (may be obscured on some slides)

## IT5002 Midterms

- **IT5002 Midterms:**
  - Saturday 1 October 2.30 pm to 3.30 pm
  - Tutorial Groups 1 – 5, I3 Auditorium
  - Tutorial Groups 6 – 8, LT15
  - Coverage: Lectures 1 to 10



# Lecture #14: Pipelining

- 1. Introduction**
- 2. MIPS Pipeline Stages**
- 3. Pipeline Datapath**
- 4. Pipeline Control**
- 5. Pipeline Performance**



# 1. Introduction: Inspiration

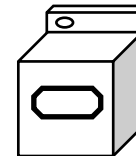
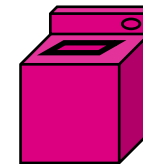
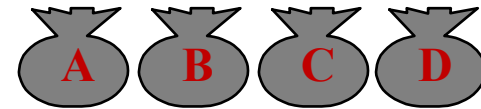
Assembly Line

**Simpler station tasks → more cars per hour.  
Simple tasks take less time, clock is faster.**

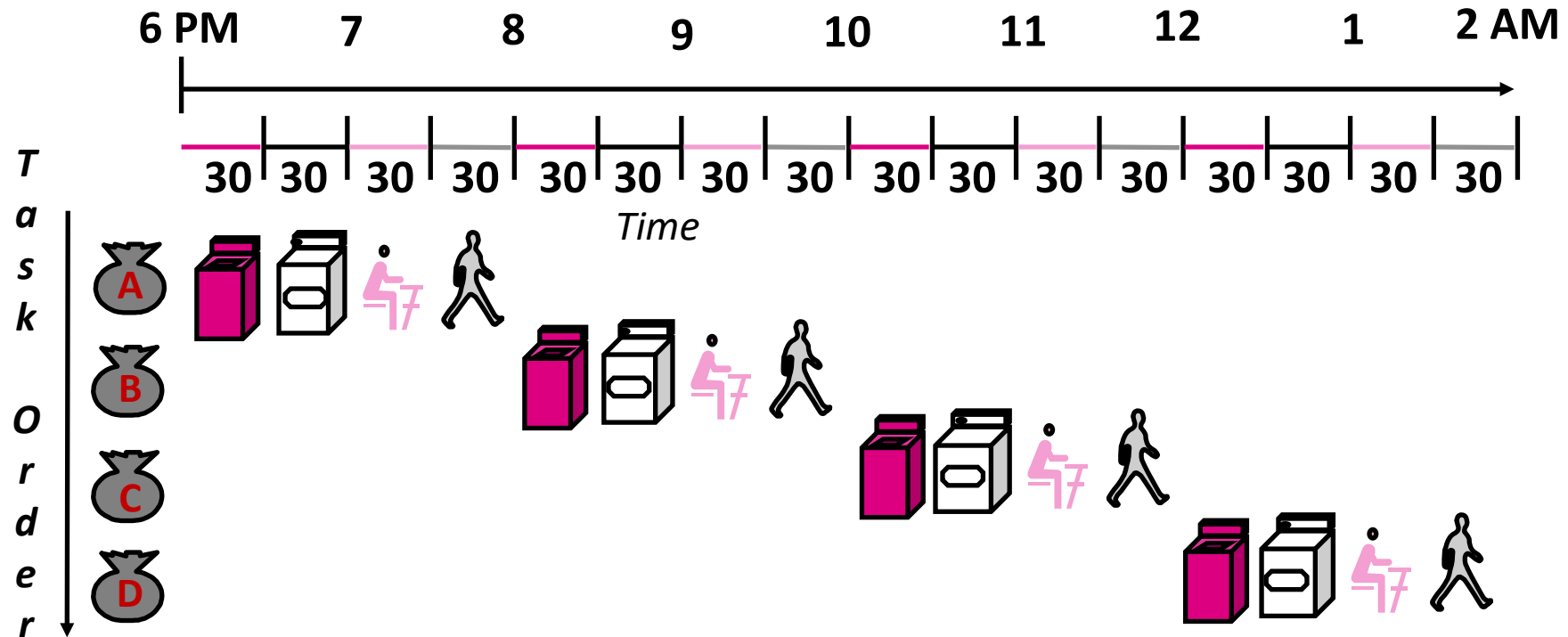


# 1. Introduction: Laundry

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold and stash
- **Washer** takes 30 minutes
- **Dryer** takes 30 minutes
- “**Folder**” takes 30 minutes
- “**Stasher**” takes 30 minutes to put clothes into drawers



# 1. Introduction: Sequential Laundry

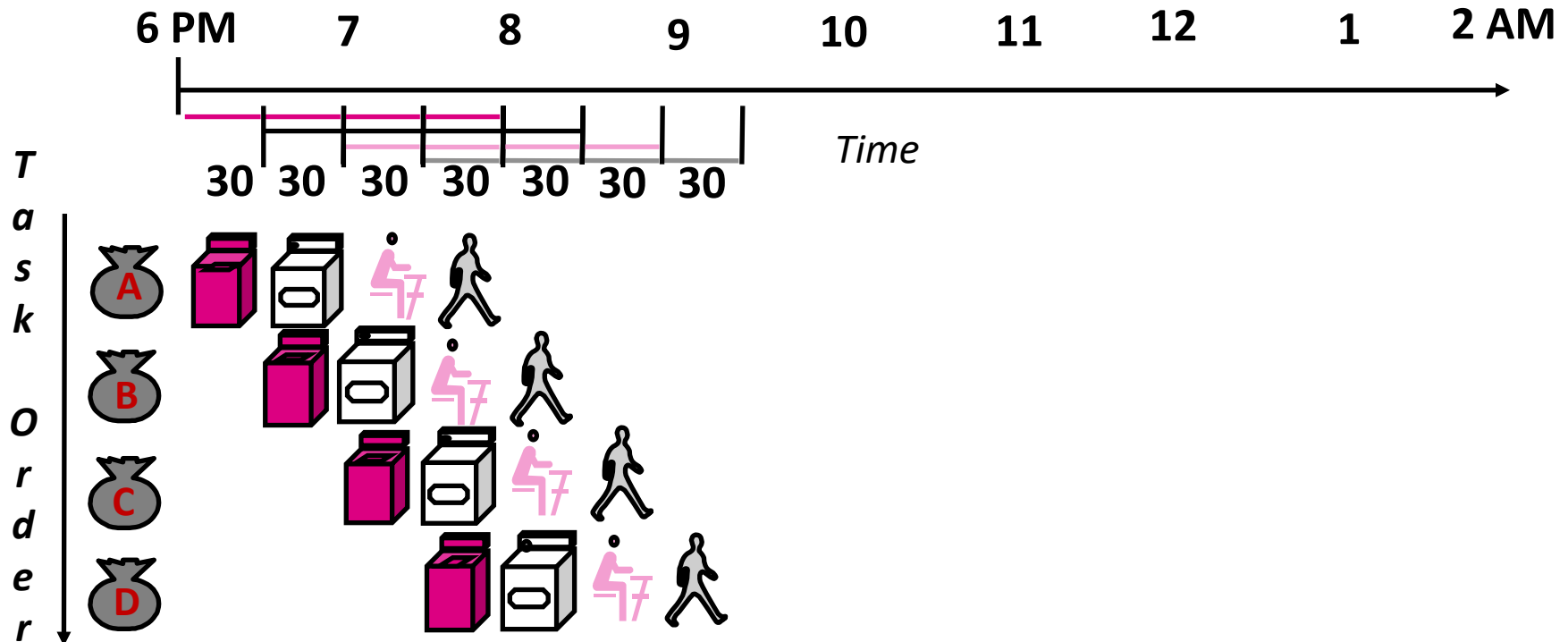


- Sequential laundry takes 8 hours for 4 loads
- Steady state: 1 load every 2 hours

If they learned **pipelining**, how long would laundry take?



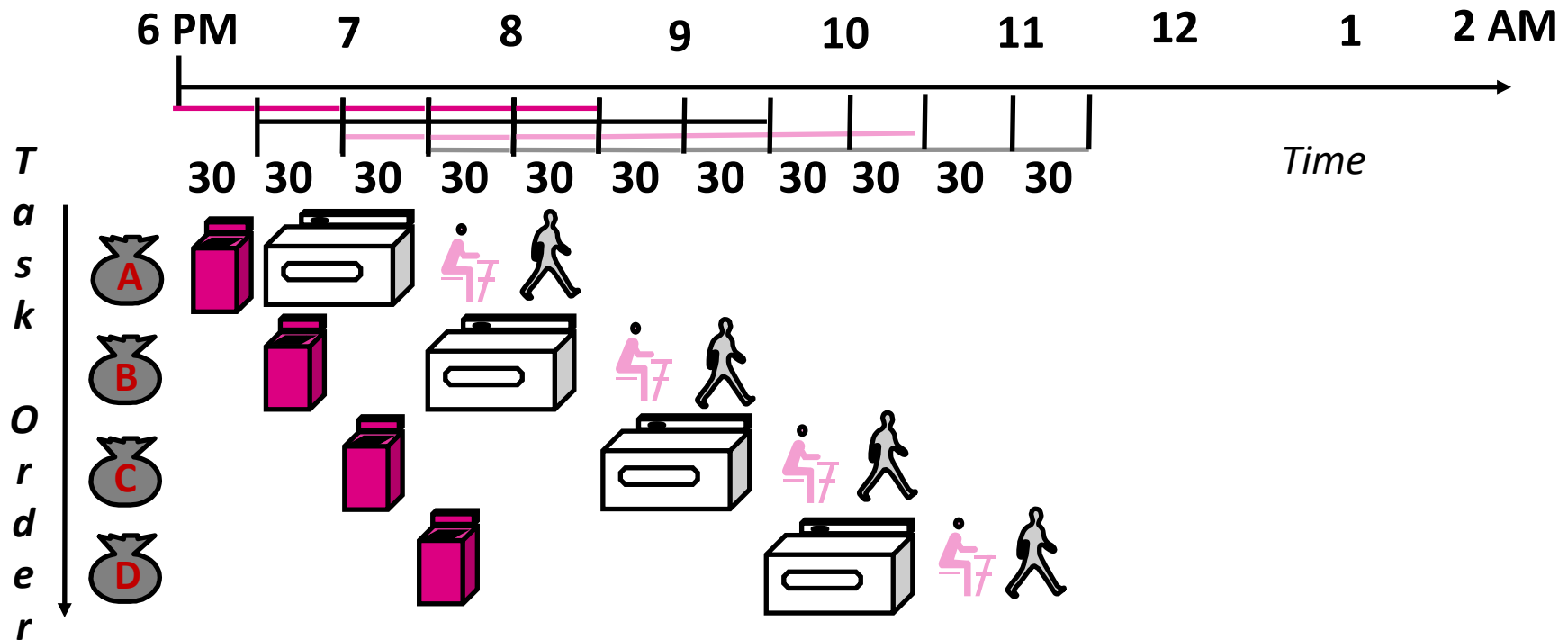
# 1. Introduction: Pipelined Laundry



- Pipelined laundry takes 3.5 hours for 4 loads!
- Steady state: 1 load every 30 minutes
- Potential speedup =  $2 \text{ hr} / 30 \text{ min} = 4$  (no. of stages)
- Time to fill pipeline takes 2 hours  $\rightarrow$  speedup  $\downarrow$



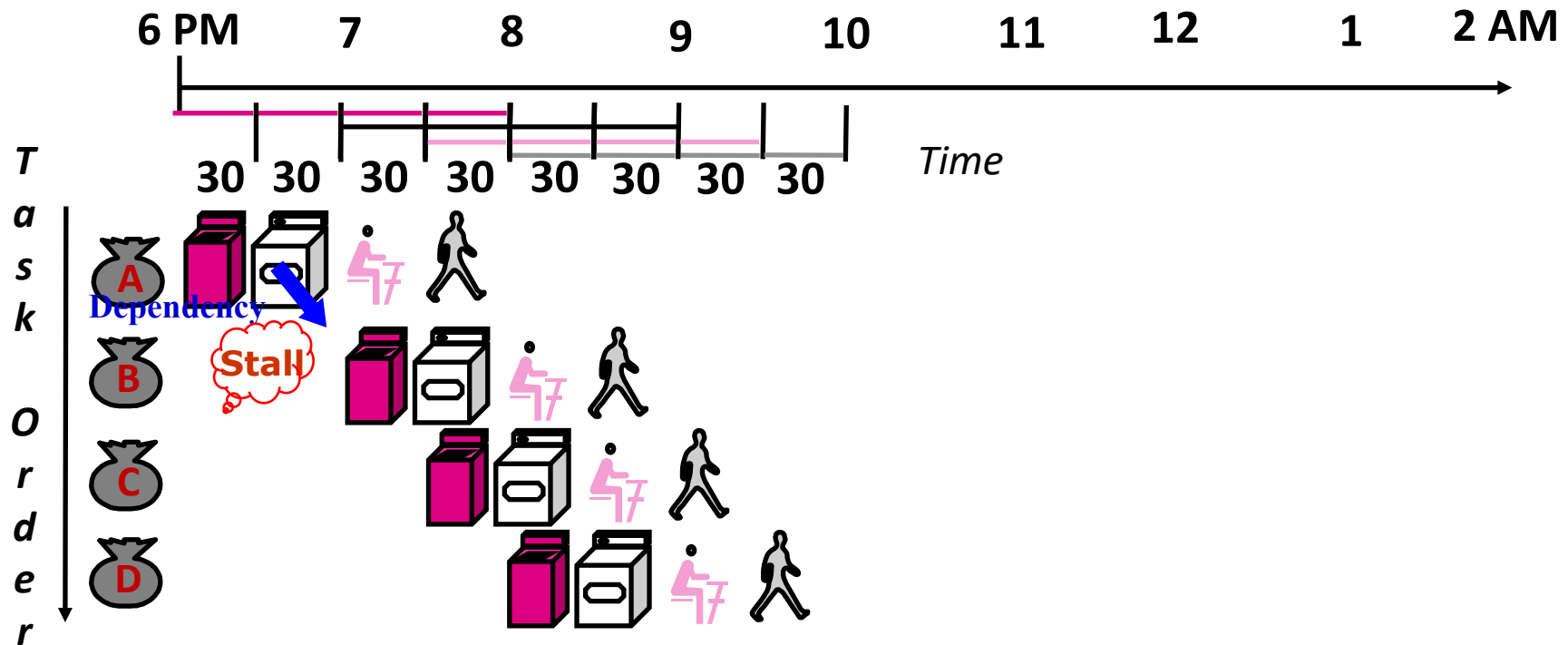
# 1. Introduction: What If: Slow Dryer



- **Pipelined laundry now takes 5.5 hours!**
- Steady state: One load every 1 hour (dryer speed)

**Pipeline rate is limited by the slowest stage**

# 1. Introduction: What If: Dependency



- Brian is using the laundry for the first time; he wants to see the outcome of one wash + dry cycle first before putting in his clothes

**Pipelined laundry now takes 4 hours**

# 1. Introduction: Pipelining Lessons

- Pipelining doesn't help **latency** of single task:
  - It helps the **throughput** of entire workload
- **Multiple** tasks operating simultaneously using different resources
- Possible delays:
  - Pipeline rate limited by **slowest** pipeline stage
  - Stall for **dependencies**



## 2. MIPS Pipeline Stages (1/2)

- **Five** Execution Stages

- **IF**: Instruction Fetch
- **ID**: Instruction Decode and Register Read
- **EX**: Execute an operation or calculate an address
- **MEM**: Access an operand in data memory
- **WB**: Write back the result into a register

- **Idea:**

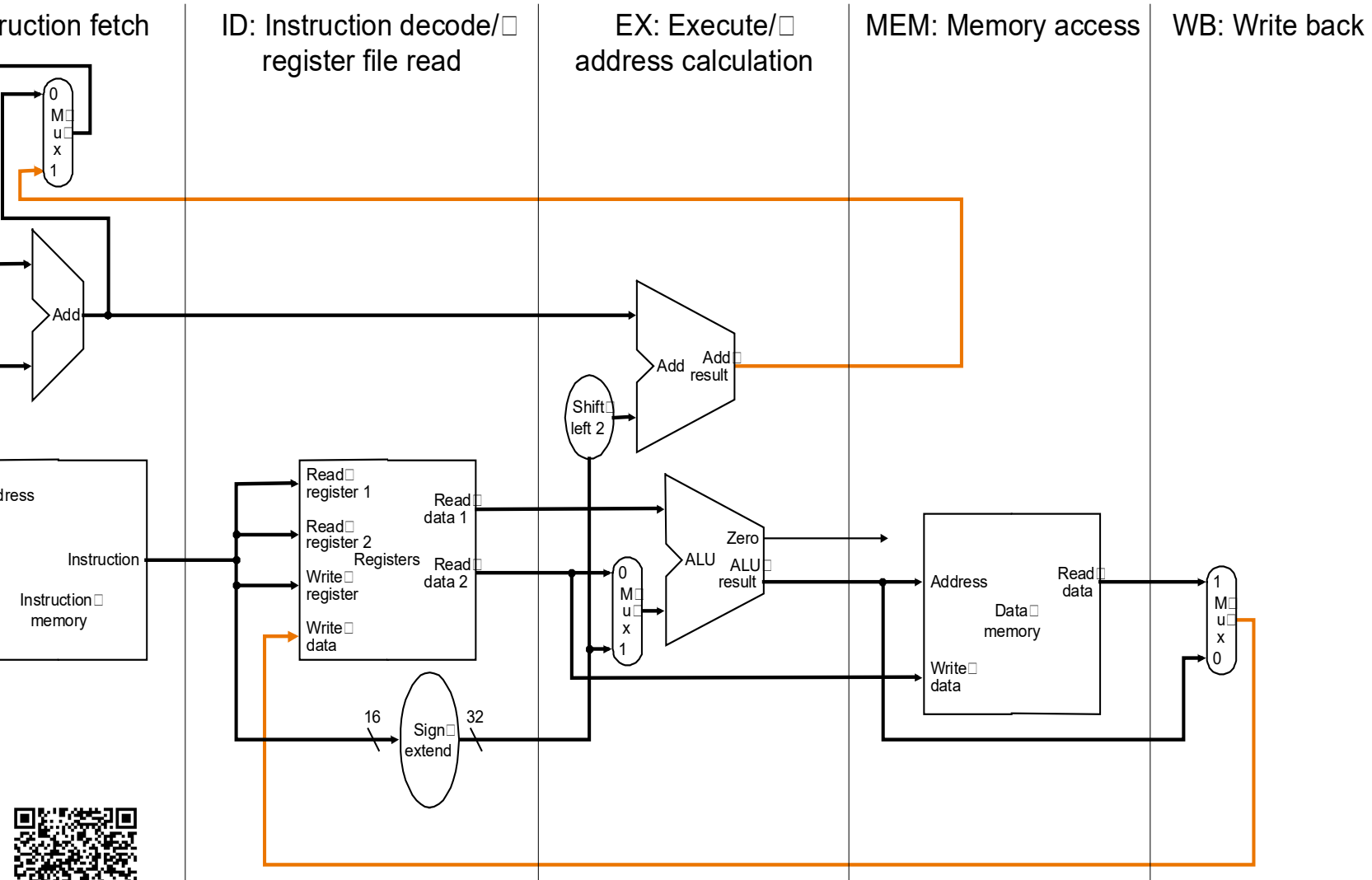
- **Each execution stage takes 1 clock cycle**
- General flow of data is from one stage to the next

- **Exceptions:**

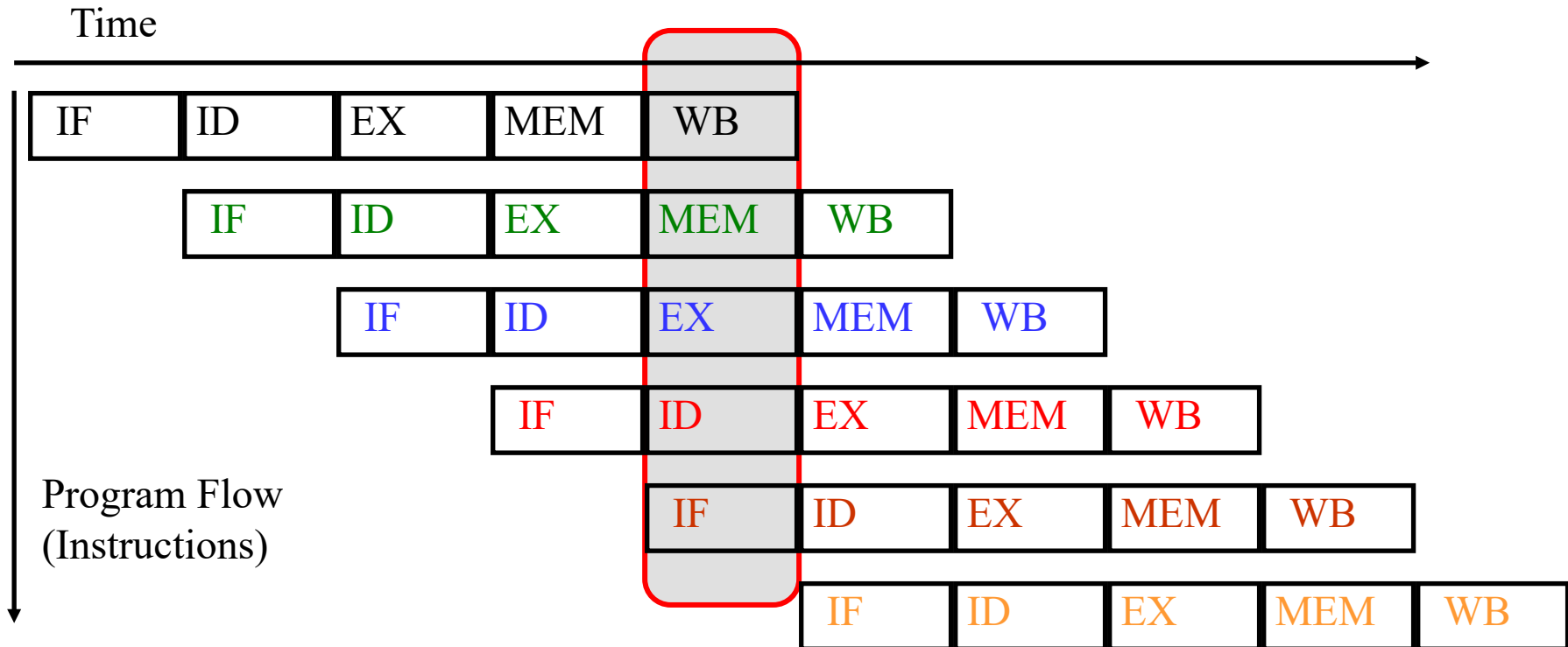
- Update of PC and write back of register file – more about this later...



## 2. MIPS Pipeline Stages (2/2)



## 2. Pipelined Execution: Illustration



Several instructions  
are in the pipeline  
simultaneously!

### 3. MIPS Pipeline: Datapath (1/3)

- **Single-cycle** implementation:
  - Update all state elements (**PC**, register file, data memory) at the end of a clock cycle
- **Pipelined** implementation:
  - One cycle per pipeline stage
  - Data required for each stage needs to be stored separately (why?)



### 3. MIPS Pipeline: Datapath (2/3)

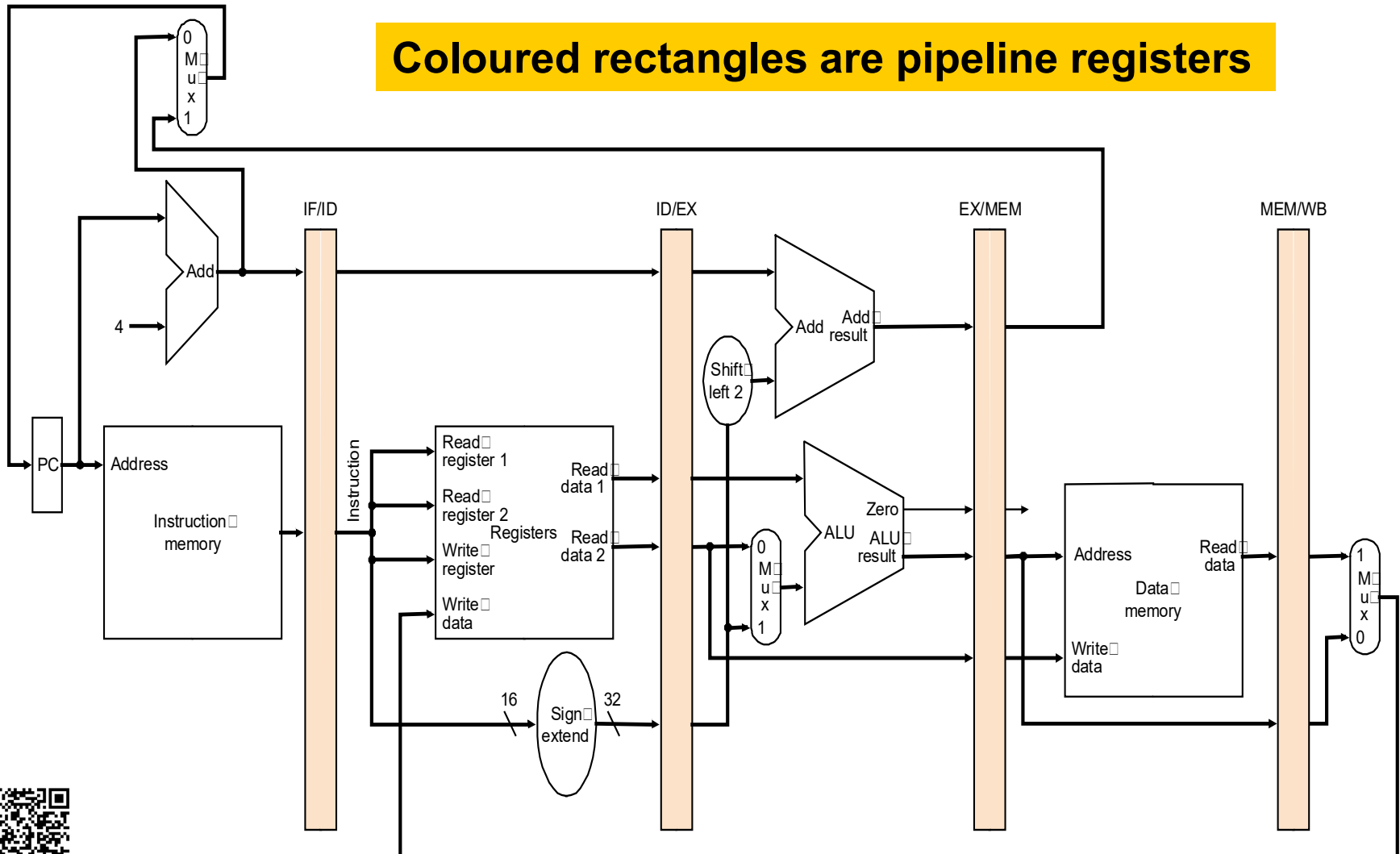
- Data used by **subsequent instructions**:
  - Store in programmer-visible state elements: **PC**, register file and memory
- Data used by **same instruction** in later pipeline stages:
  - Additional registers in datapath called **pipeline registers**
  - **IF/ID**: register between **IF** and **ID**
  - **ID/EX**: register between **ID** and **EX**
  - **EX/MEM**: register between **EX** and **MEM**
  - **MEM/WB**: register between **MEM** and **WB**
- Why no register at the end of **WB** stage?



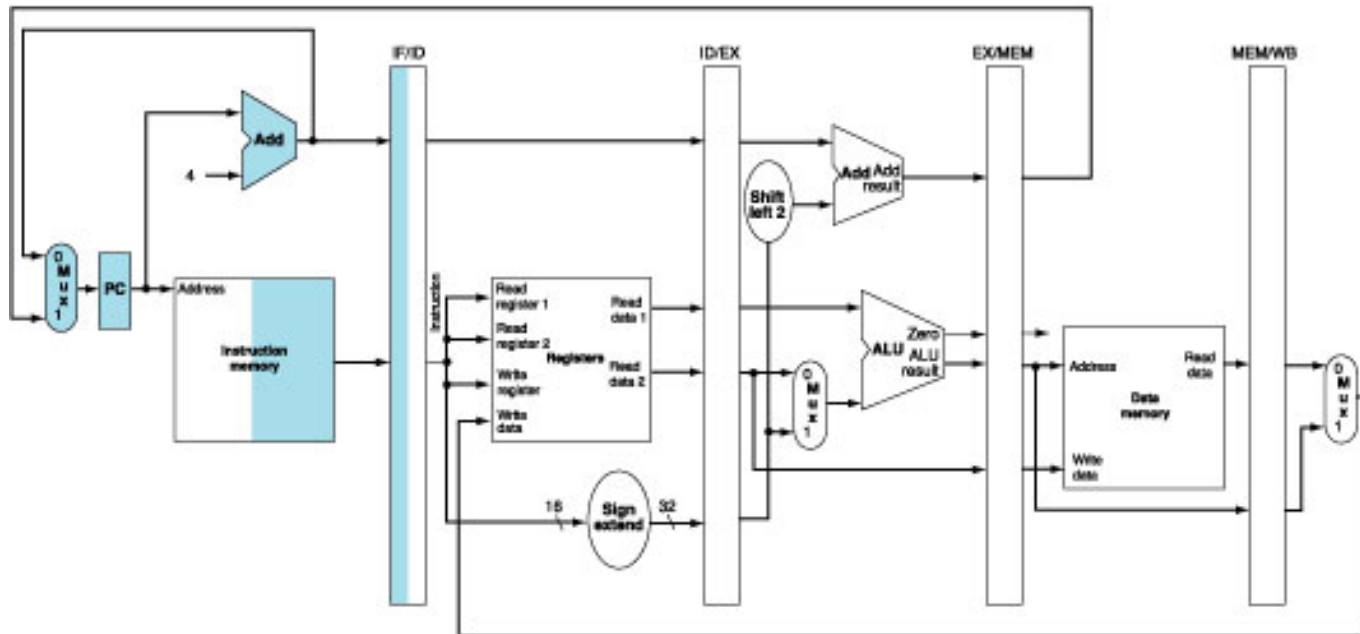


# 3. MIPS Pipeline: Datapath (3/3)

**Coloured rectangles are pipeline registers**

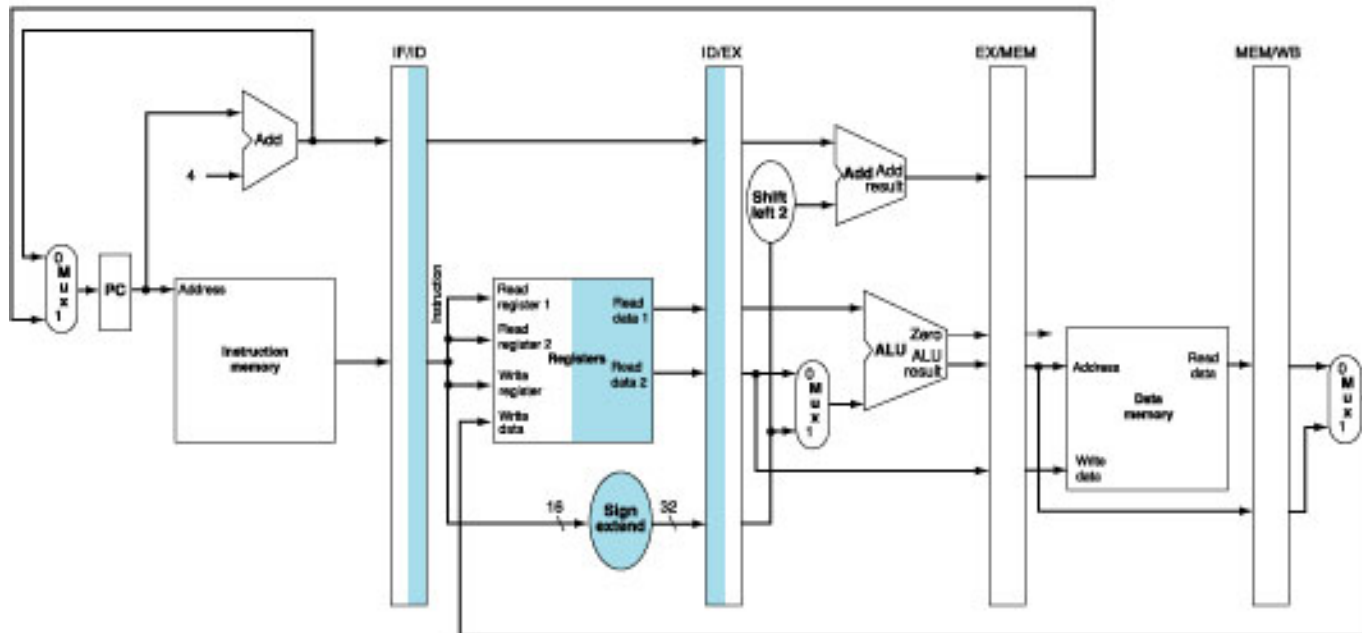


### 3. Pipeline Datapath: **IF** Stage



- At the end of a cycle, **IF/ID** receives (stores):
  - Instruction read from InstructionMemory[ PC ]
  - PC + 4
- PC + 4
  - Also connected to one of the MUX's inputs (another coming later)

### 3. Pipeline Datapath: ID Stage



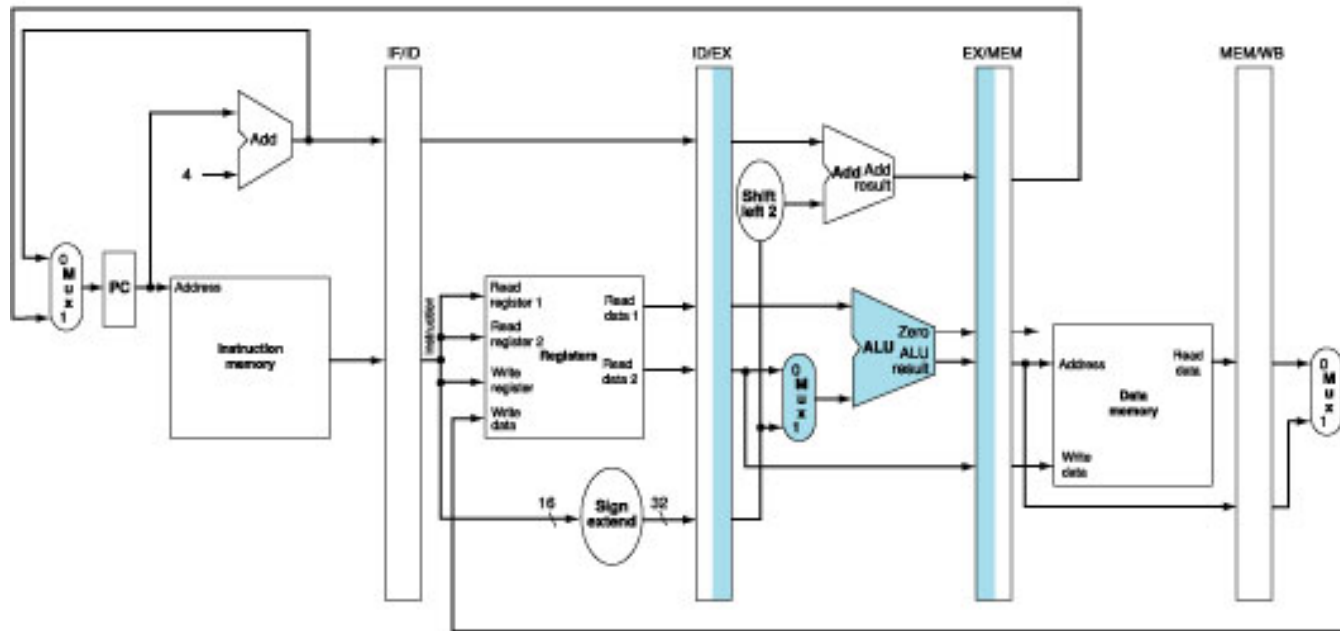
**At the beginning of a cycle**  
**IF/ID** register supplies:

- ❖ Register numbers for reading two registers
- ❖ 16-bit offset to be sign-extended to 32-bit

**At the end of a cycle**  
**ID/EX** receives:

- ❖ Data values read from register file
- ❖ 32-bit immediate value
- ❖  $PC + 4$

### 3. Pipeline Datapath: **EX** Stage



**At the beginning of a cycle**  
**ID/EX** register supplies:

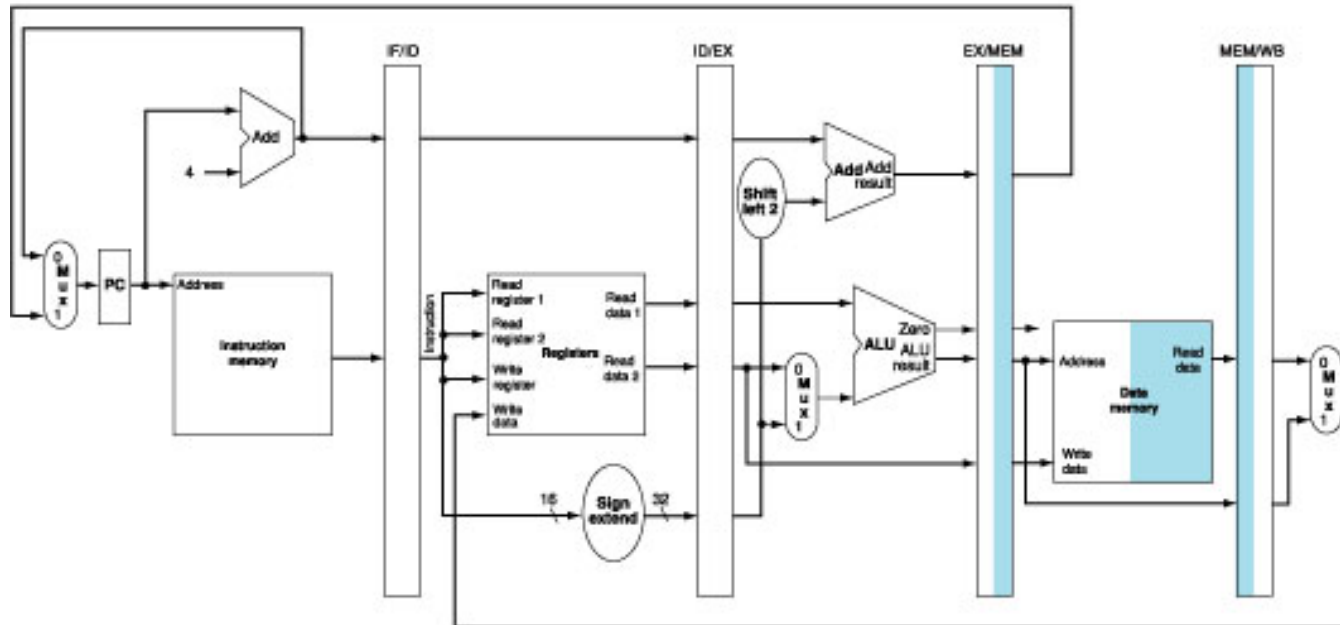
- ❖ Data values read from register file
- ❖ 32-bit immediate value
- ❖  $PC + 4$

**At the end of a cycle**  
**EX/MEM** receives:

- ❖  $(PC + 4) + (Immediate \times 4)$
- ❖ ALU result
- ❖ **isZero?** signal
- ❖ Data Read 2 from register file



### 3. Pipeline Datapath: **MEM** Stage



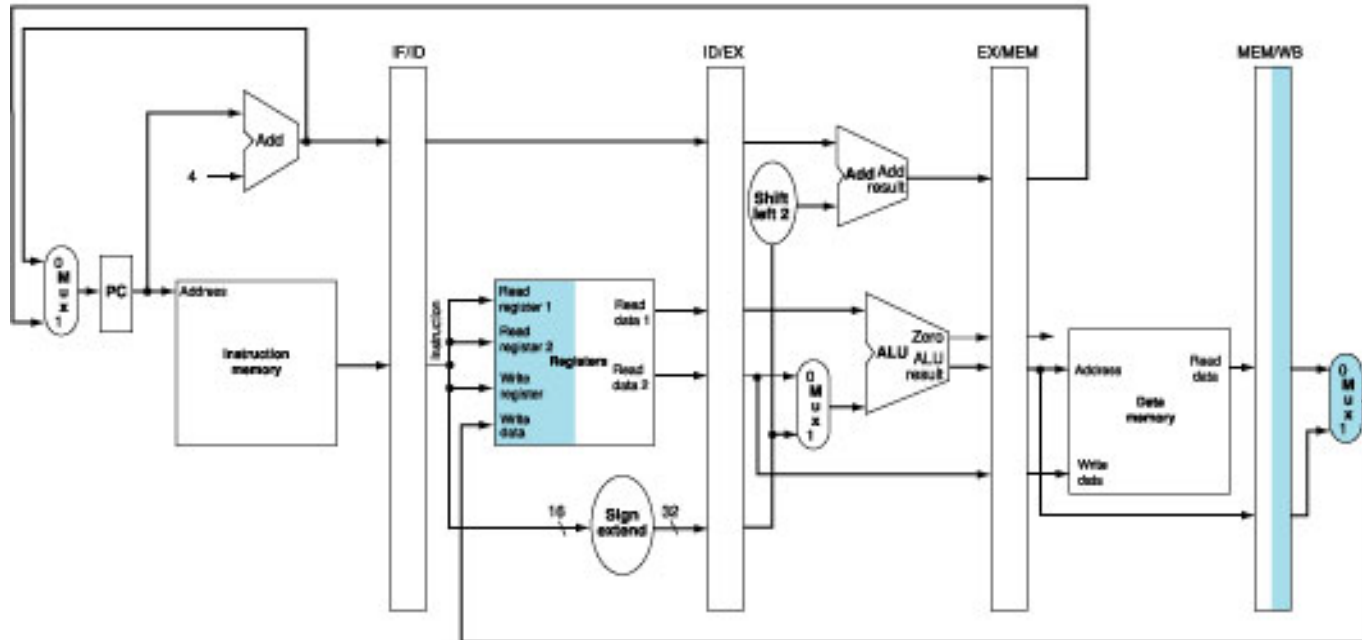
At the beginning of a cycle  
**EX/MEM** register supplies:

- ❖  $(PC + 4) + (Immediate \times 4)$
- ❖ ALU result
- ❖ **isZero?** signal
- ❖ Data Read 2 from register file

At the end of a cycle  
**MEM/WB** receives:

- ❖ ALU result
- ❖ Memory read data

### 3. Pipeline Datapath: **WB** Stage



**At the beginning of a cycle**  
**MEM/WB** register supplies:

- ❖ ALU result
- ❖ Memory read data

**At the end of a cycle**

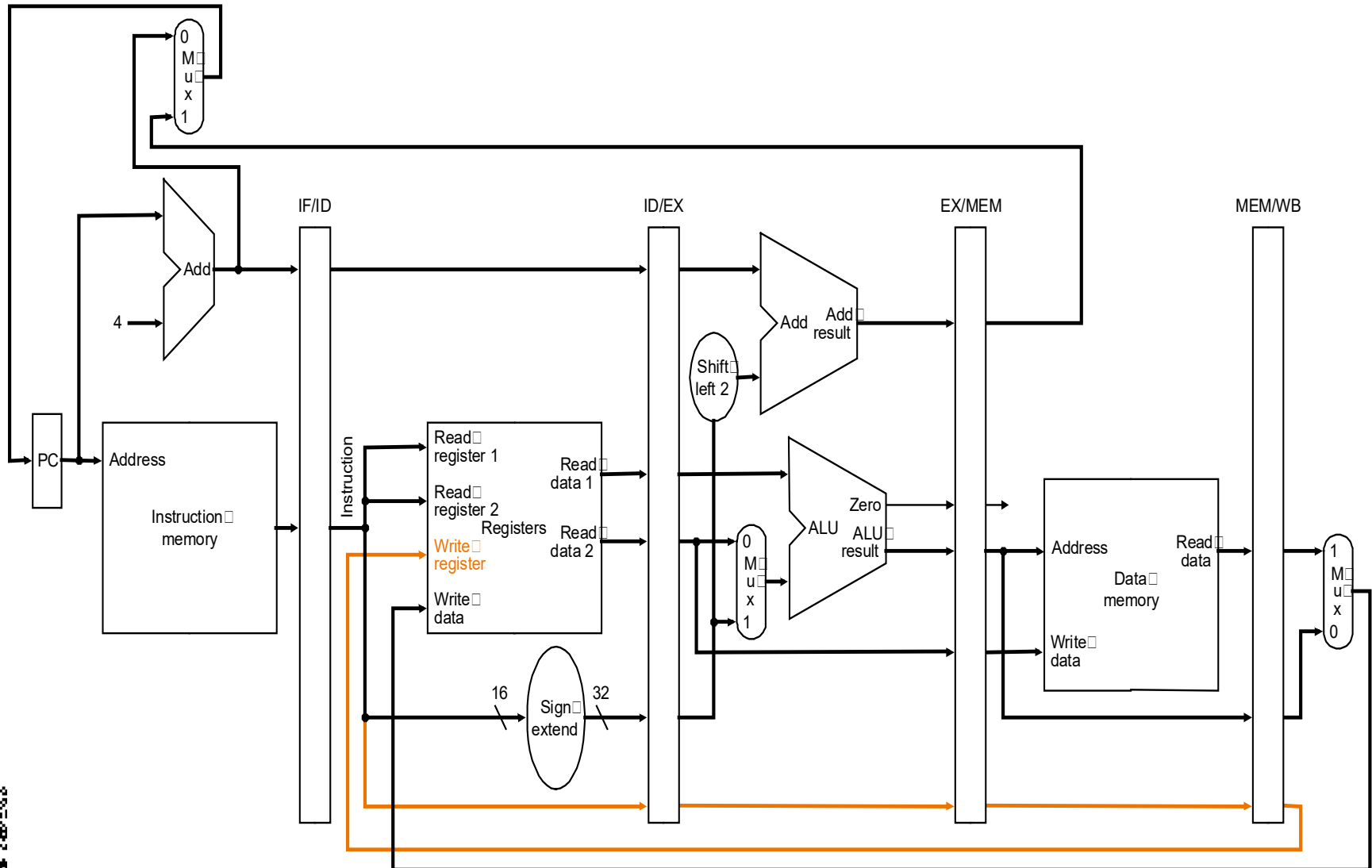
- ❖ Result is written back to register file (if applicable)
- ❖ **There is a bug here.....**

### 3. Corrected Datapath (1/2)

- Observe the “Write register” number
  - Supplied by the **IF/ID** pipeline register
  - It is NOT the correct write register for the instruction now in **WB** stage!
- **Solution:**
  - Pass “Write register” number from **ID/EX** through **EX/MEM** to **MEM/WB** pipeline register for use in **WB** stage
  - i.e. let the "Write register" number follows the instruction through the pipeline until it is needed in WB stage



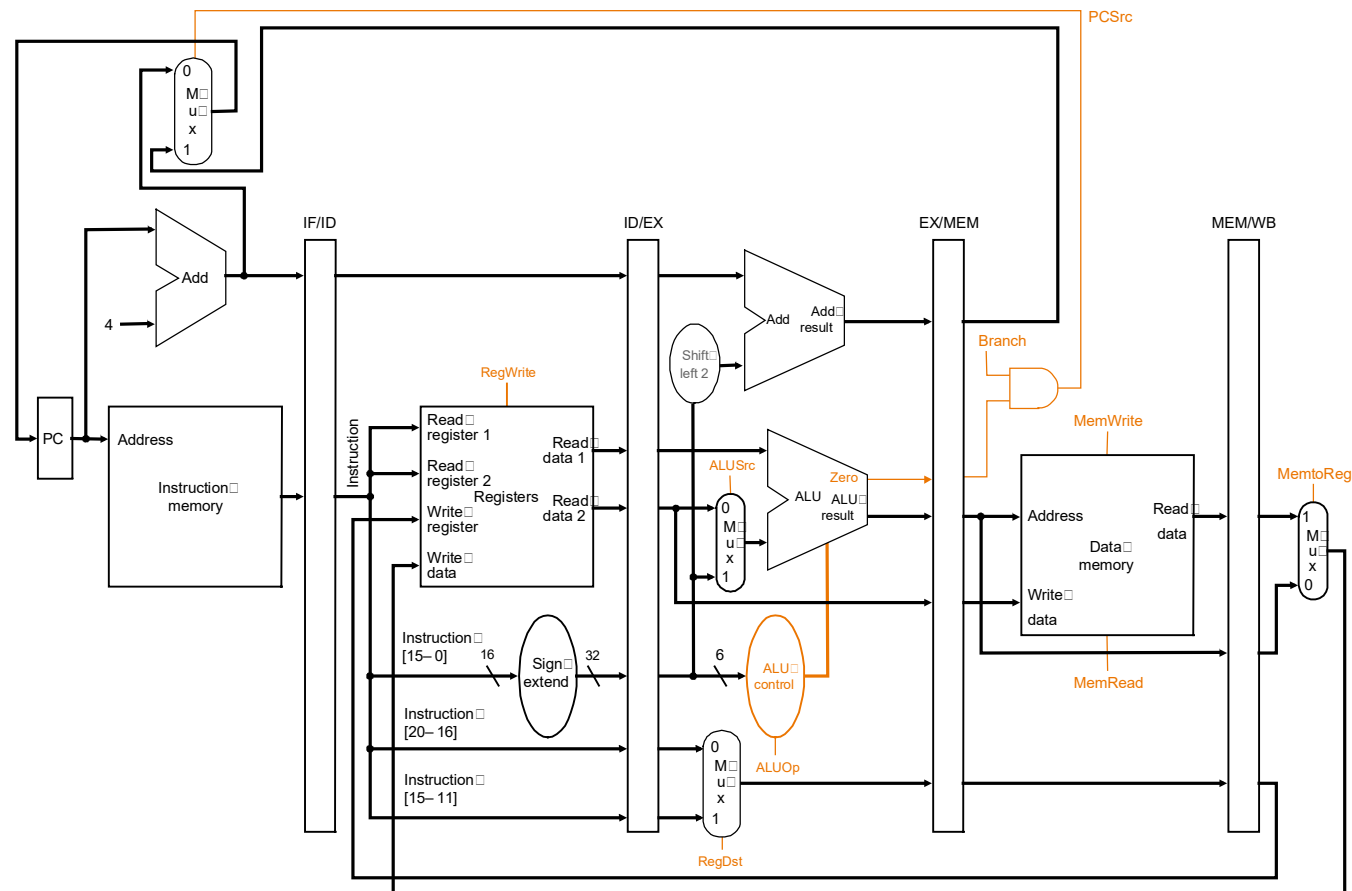
# 3. Corrected Datapath (2/2)



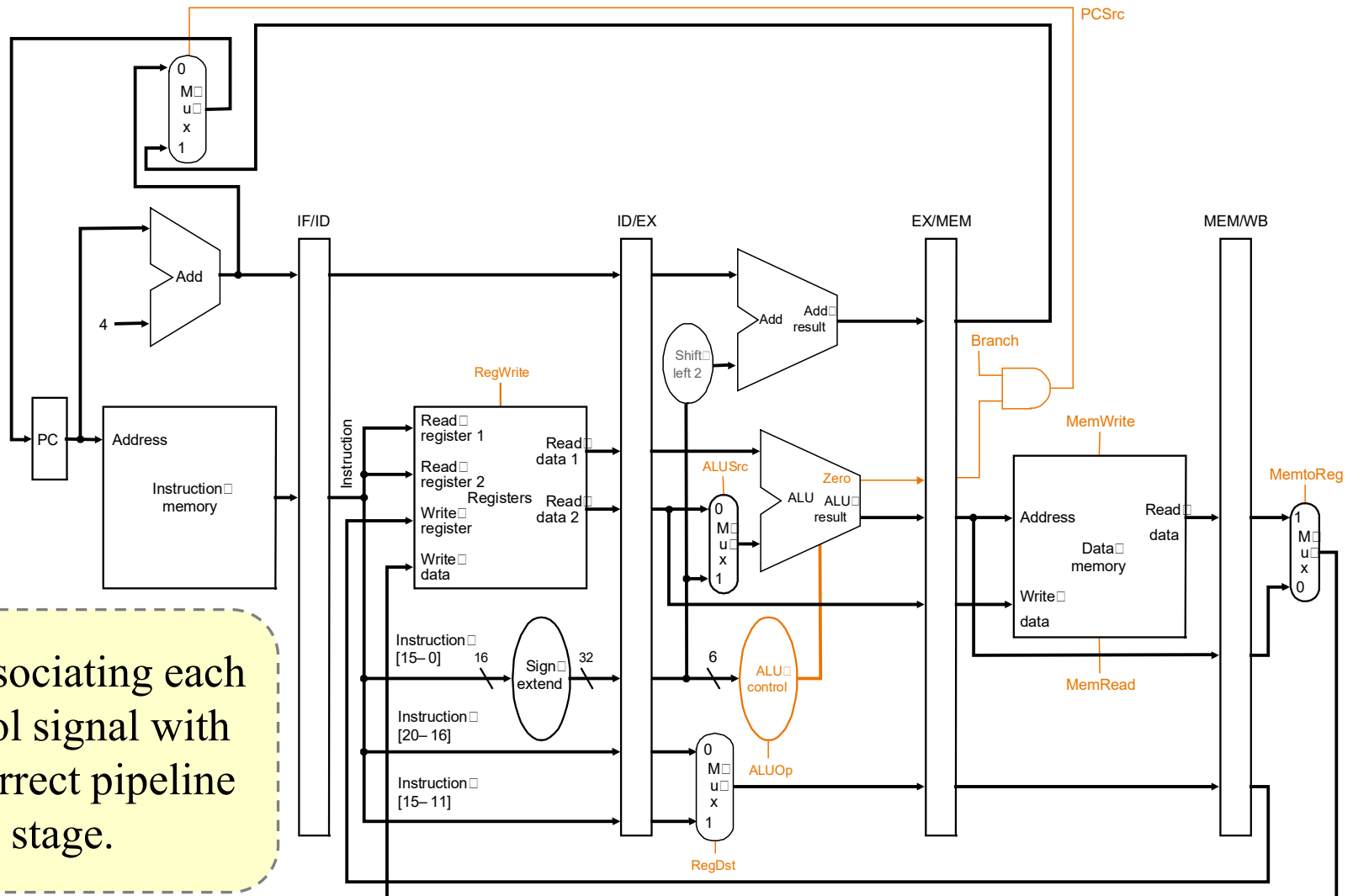


# 4. Pipeline Control: Main Idea

- Same control signals as single-cycle datapath
- Difference:** Each control signal belongs to a particular pipeline stage



# 4. Pipeline Control: Try it!



Try associating each control signal with the correct pipeline stage.

## 4. Pipeline Control: Grouping

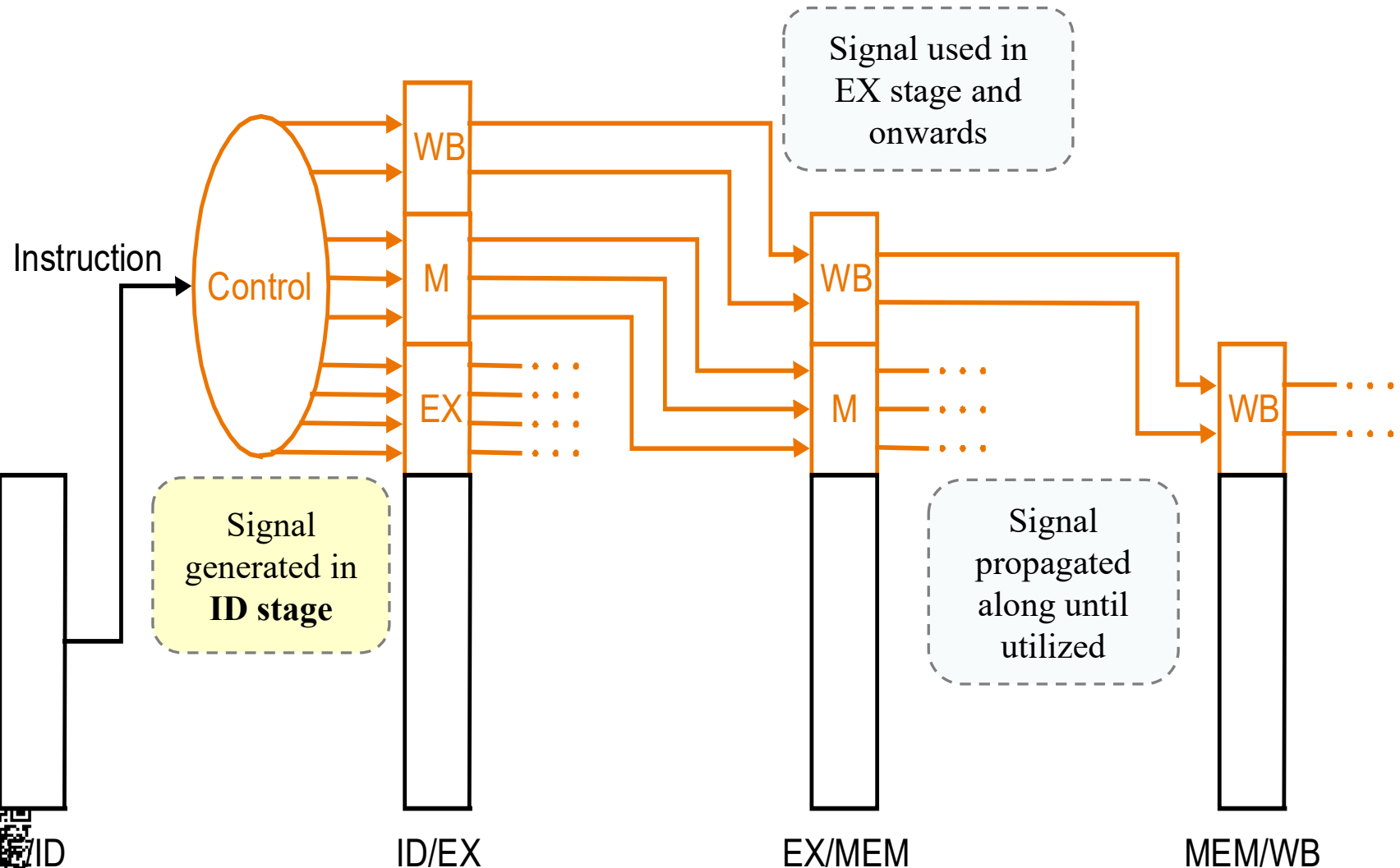
### ■ Group control signals according to pipeline stage

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

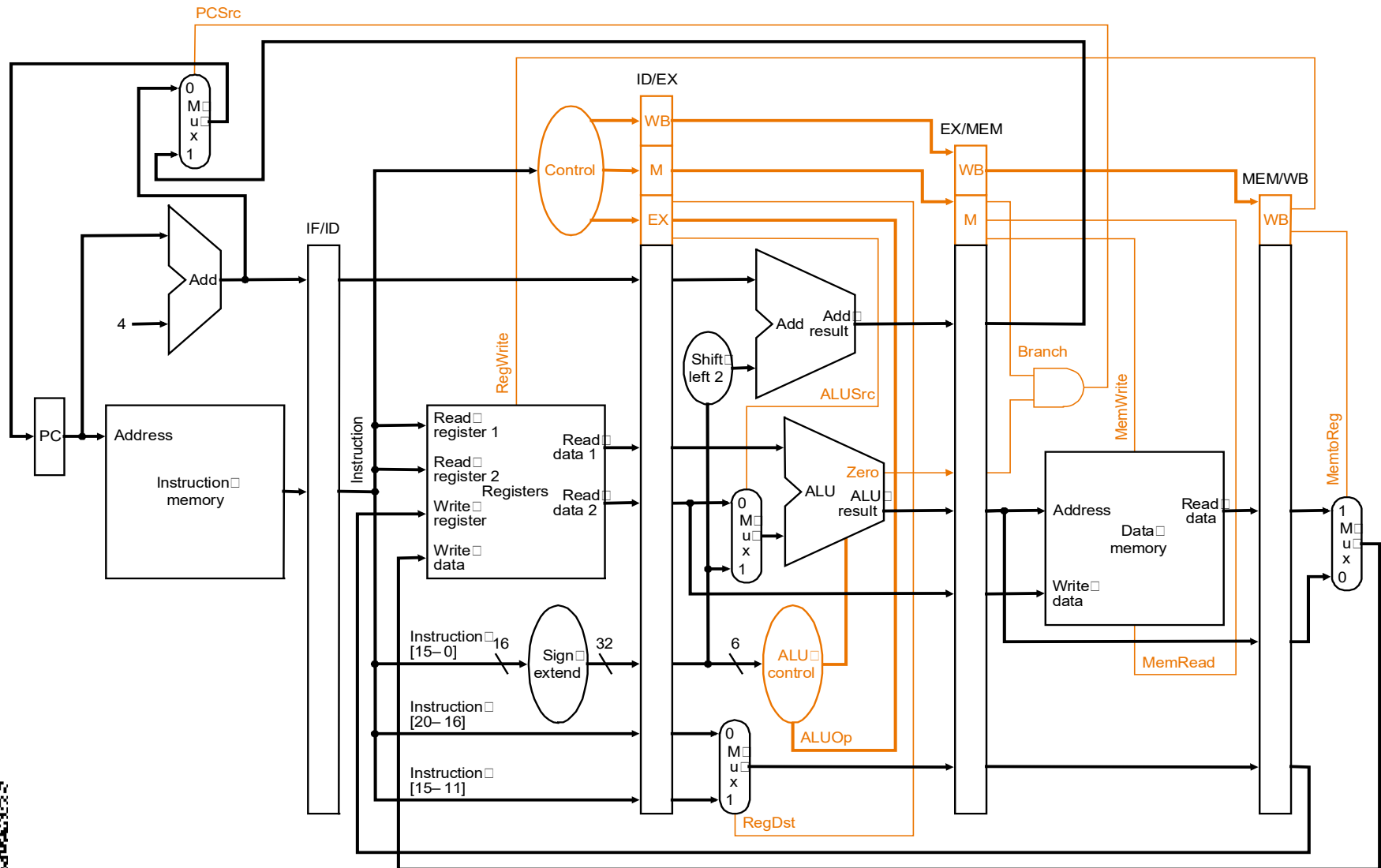
	EX Stage				MEM Stage			WB Stage	
	RegDst	ALUSrc	ALUop		Mem Read	Mem Write	Branch	MemTo Reg	Reg Write
			op1	op0					
R-type	1	0	1	0	0	0	0	0	1
lw	0	1	0	0	1	0	0	1	1
sw	X	1	0	0	0	1	0	X	0
beq	X	0	0	1	0	0	1	X	0



# 4. Pipeline Control: Implementation

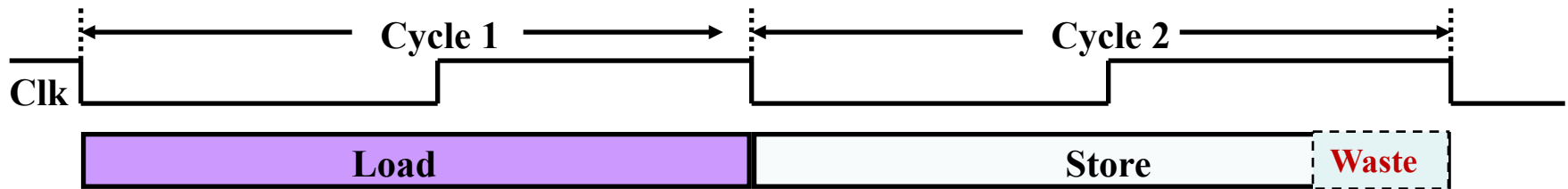


# 4. Pipeline Control: Datapath and Control

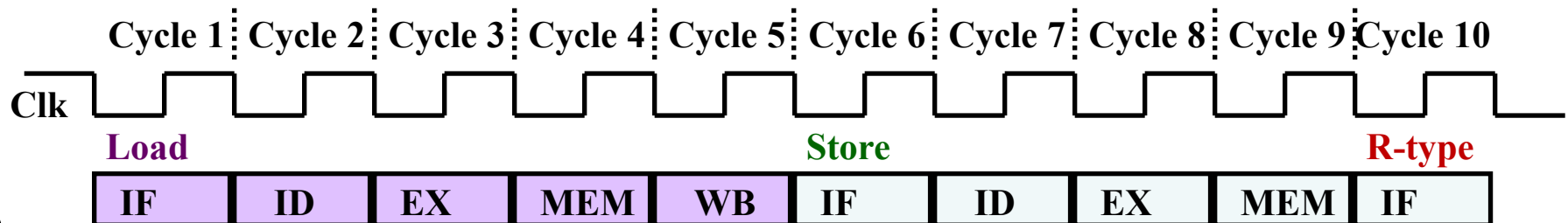


# 5. Different Implementations

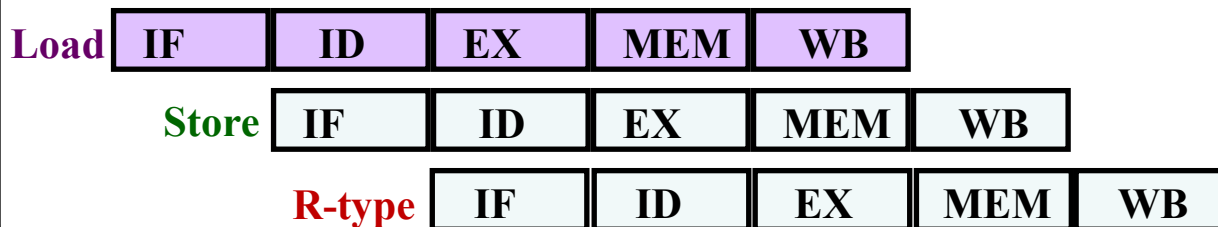
## Single-Cycle



## Multi-Cycle



## Pipeline



## 5. Single Cycle Processor: Performance

- **Cycle time:**

- $CT_{seq} = \sum_{k=1}^N T_k$
- $T_k$  = Time for operation in stage k
- N = Number of stages

- **Total Execution Time for  $I$  instructions:**

- $Time_{seq} = \text{Cycles} \times \text{CycleTime}$   
 $= I \times CT_{seq} = I \times \sum_{k=1}^N T_k$



## 5. Single Cycle Processor: Example

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

- **Cycle Time**
  - Choose the longest total time = **8ns**
- To execute **100 instructions**:
  - $100 \times 8\text{ns} = 800\text{ns}$





## 5. Multi-Cycle Processor: Performance

- **Cycle time:**
  - $CT_{multi} = \max(T_k)$
  - $\max(T_k)$  = longest stage duration among the N stages
- **Total Execution Time for  $I$  instructions:**
  - $Time_{multi} = Cycles \times CycleTime$   
 $= I \times \textit{Average CPI} \times CT_{multi}$
  - Average CPI is needed because each instruction takes different number of cycles to finish



## 5. Multi-Cycle Processor: Example

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

### ■ Cycle Time

- Choose the longest stage time = **2ns**
- To execute **100 instructions**, with a given **average CPI of 4.6**
  - $100 \times 4.6 \times 2\text{ns} = 920\text{ns}$



## 5. Pipeline Processor: Performance

### ■ Cycle Time:

- $CT_{pipeline} = \max(T_k) + T_d$
- $\max(T_k)$  = longest time among the N stages
- $T_d$  = Overhead for pipelining, e.g. pipeline register

### ■ Cycles needed for **I** instructions:

- $I + N - 1$
- $N - 1$  is the cycles wasted in filling up the pipeline

### ■ Total Time needed for **I** instructions :

- $Time_{pipeline} = Cycle \times CT_{pipeline}$   
 $= (I + N - 1) \times (\max(T_k) + T_d)$



## 5. Pipeline Processor: Example

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

### ■ Cycle Time

- assume pipeline register delay of **0.5ns**
- longest stage time + overhead = **2 + 0.5 = 2.5ns**

### ■ To execute **100 instructions**:

- **$(100 + 5 - 1) \times 2.5\text{ns} = 260\text{ns}$**



## 5. Pipeline Processor: Ideal Speedup (1/2)

- **Assumptions for ideal case:**

- Every stage takes the same amount of time:

- $\sum_{k=1}^N T_k = N \times T_k$

- No pipeline overhead →  $T_d = 0$

- Number of instructions **I**, is much larger than number of stages, **N**

- **Note: The above also shows how pipeline processor loses performance**



## 5. Pipeline Processor: Ideal Speedup (2/2)

$$\blacksquare \text{Speedup}_{\text{pipeline}} = \frac{\text{Time}_{\text{seq}}}{\text{Time}_{\text{pipeline}}}$$

$$= \frac{I \times \sum_{k=1}^N T_k}{(I+N-1) \times (\max(T_k) + T_d)}$$

$$= \frac{I \times N \times T_k}{(I+N-1) \times T_k}$$

$$\approx \frac{I \times N \times T_k}{I \times T_k}$$

$$\approx N$$

### Conclusion:

Pipeline processor can gain **N** times speedup, where **N** is the number of pipeline stages



# Review Question

- Given this code:

```
add $t0, $s0, $s1
sub $t1, $s0, $s1
sll $t2, $s0, 2
srl $t3, $s1, 2
```

a) 4 cycles

b)  $4/(100 \times 10^6) = 40 \text{ ns}$

c)  $4 + 4 = 8 \text{ cycles}$

d)  $8/(500 \times 10^6) = 16 \text{ ns}$

- a) How many cycles will it take to execute the code on a single-cycle datapath?
- b) How long will it take to execute the code on a single-cycle datapath, assuming a 100 MHz clock?
- c) How many cycles will it take to execute the code on a 5-stage MIPS pipeline?
- d) How long will it take to execute the code on a 5-stage MIPS pipeline, assuming a 500 MHz clock?

