Lecture #20

Pipelining
Part I: Introduction

NUS | School of
National University | Computing
of Singapore

Questions?

Ask at
https://sets.netlify.app/module/676ca3a07d7f5ffc1741dc65

OR

Scan and ask your questions here!
(May be obscured in some slides)

Lecture #20: Pipelining I

1. Introduction
2. MIPS Pipeline Stages
3. Pipeline Datapath
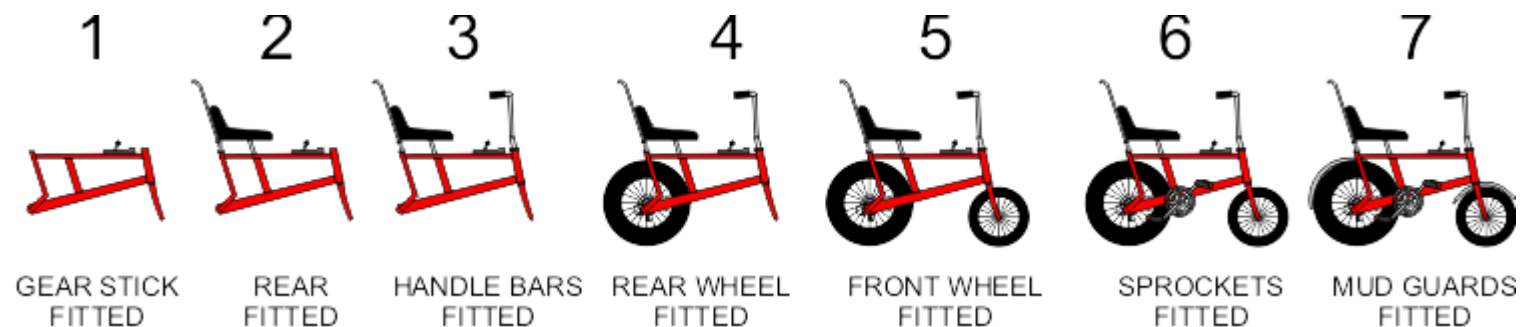4. Pipeline Control
5. Pipeline Performance
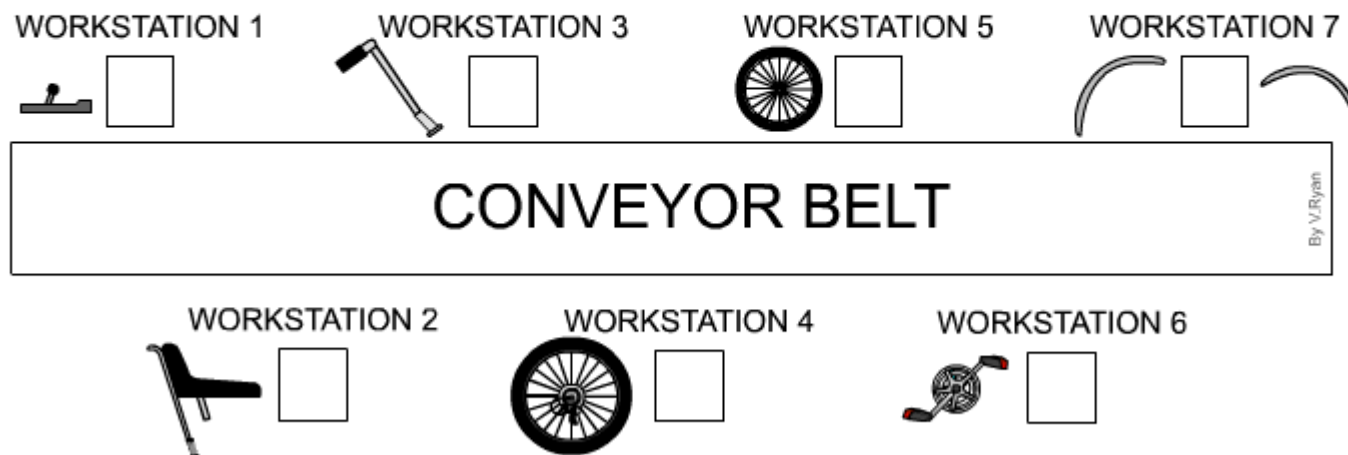
1. Introduction: Inspiration

Assembly Line

Simpler station tasks → more cars per hour.
Simple tasks take less time, clock is faster.
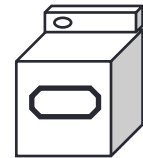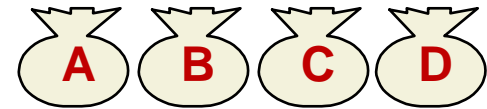
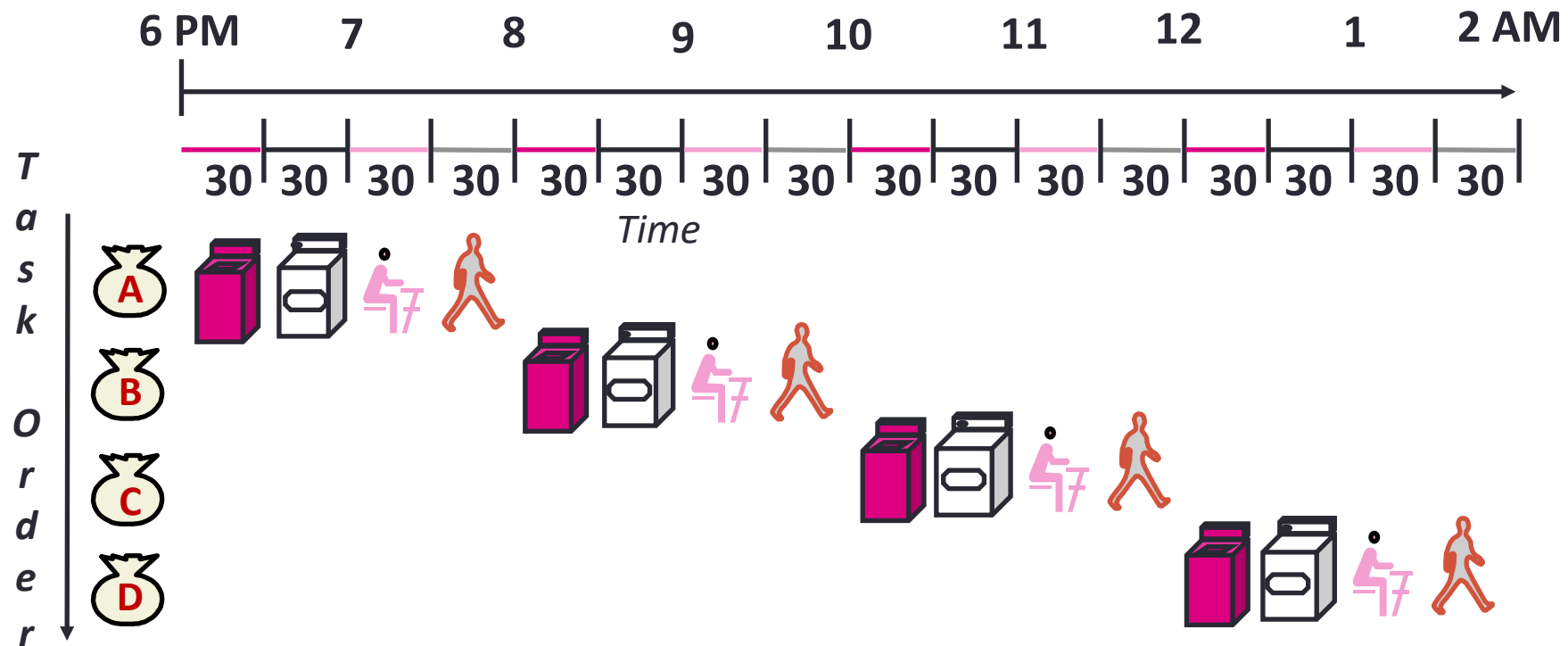# 1. Introduction: Inspiration

EXAMPLE BASIC PRODUCTION LINE

1. Introduction: Laundry

- **A**nn, **B**rian, **C**athy, **D**ave 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
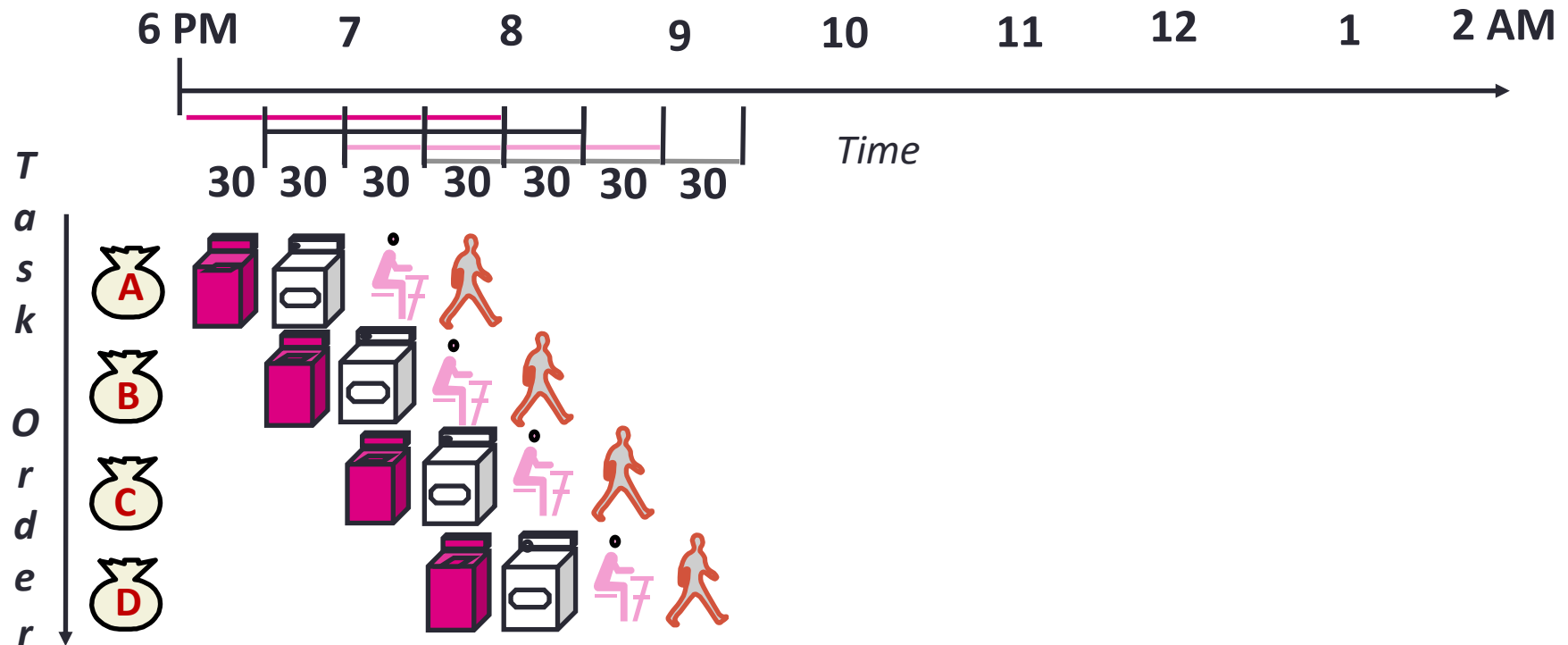

- Sequential laundry takes <u>8 hours</u> for 4 loads
- Steady state: 1 load every 2 hours
- If they learned pipelining, how long would laundry take?

1. Introduction: Pipelined Laundry
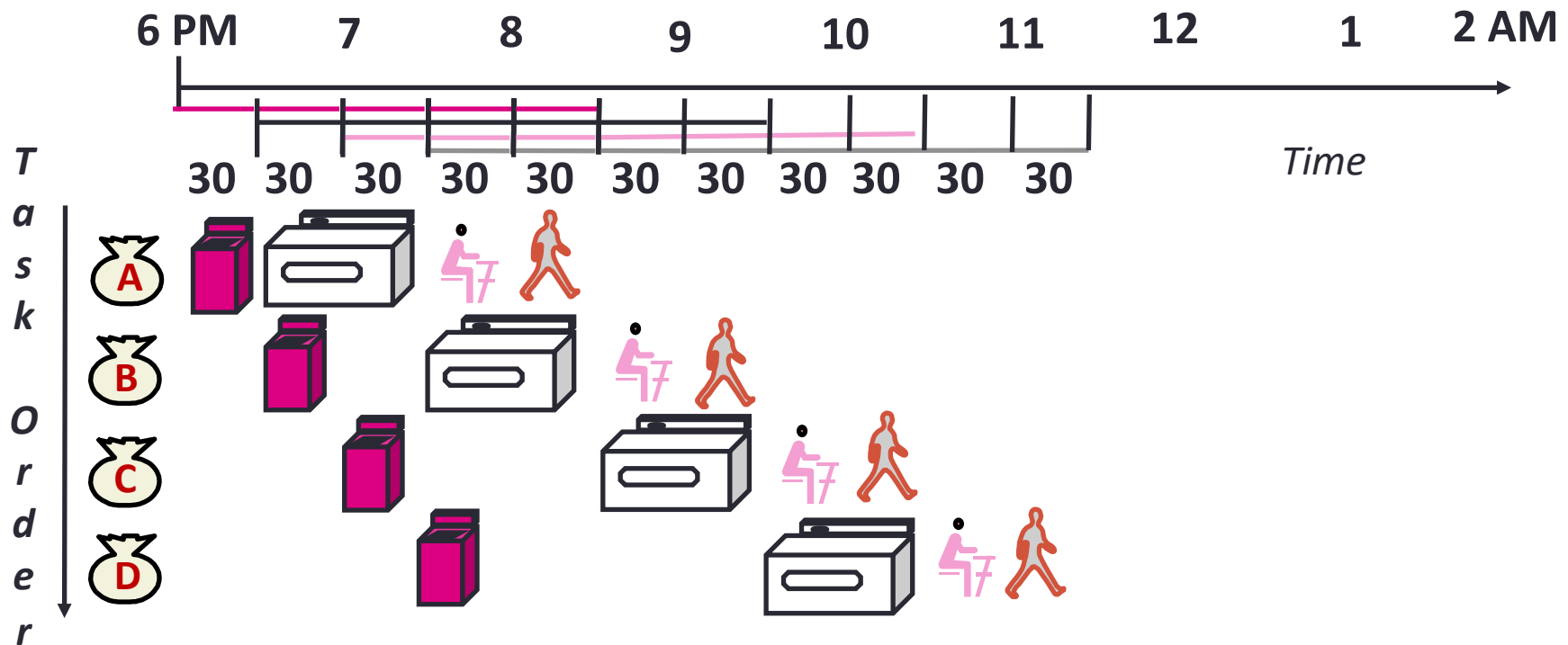

- Pipelined laundry takes <u>3.5 hours</u> for 4 loads!
- Steady state: 1 load every 30 minutes
- Potential speedup = 2 hr/30 min = 4 (no. of stages)
- Time to fill pipeline takes 2 hours → speedup ↓

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
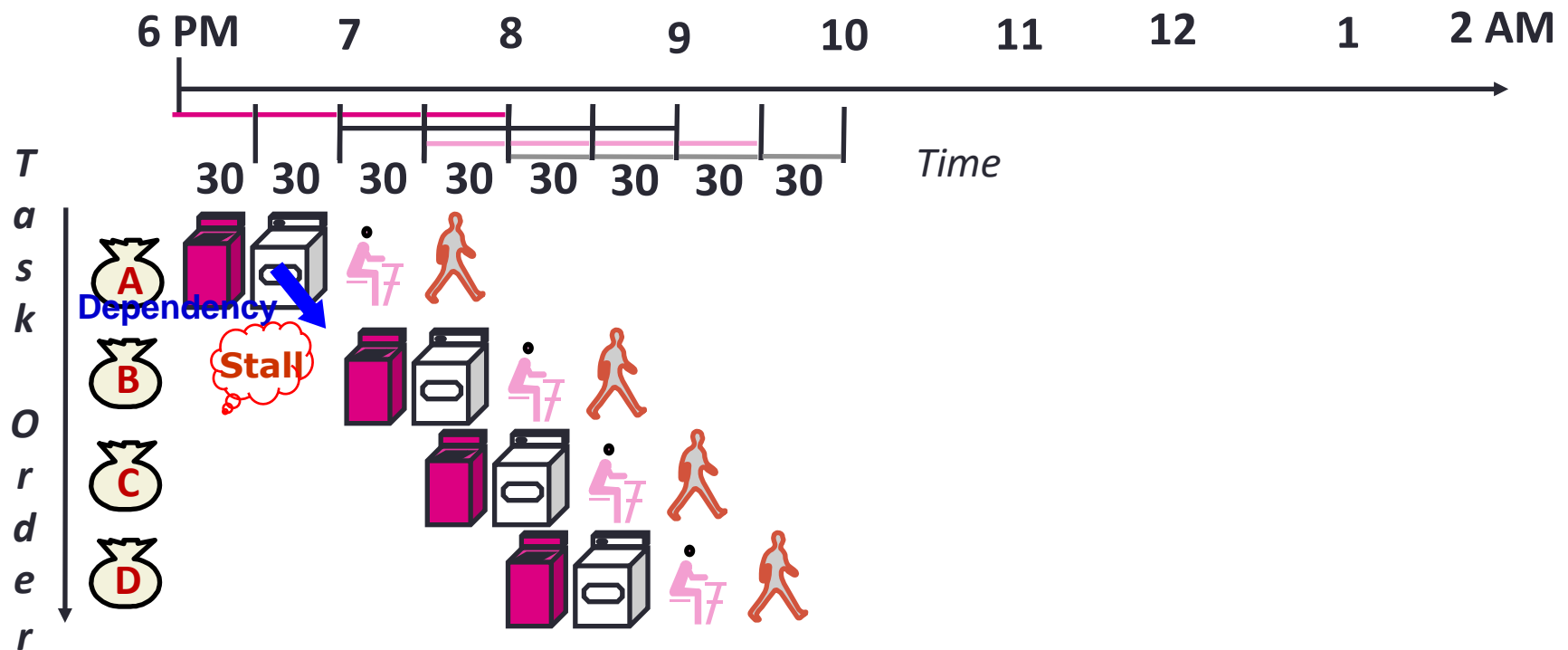
- Brian is using the laundry for the first time; he wants to see the outcome of one wash + dry cycle first before putting in his clothes
- Pipelined laundry now takes 4 hours

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)
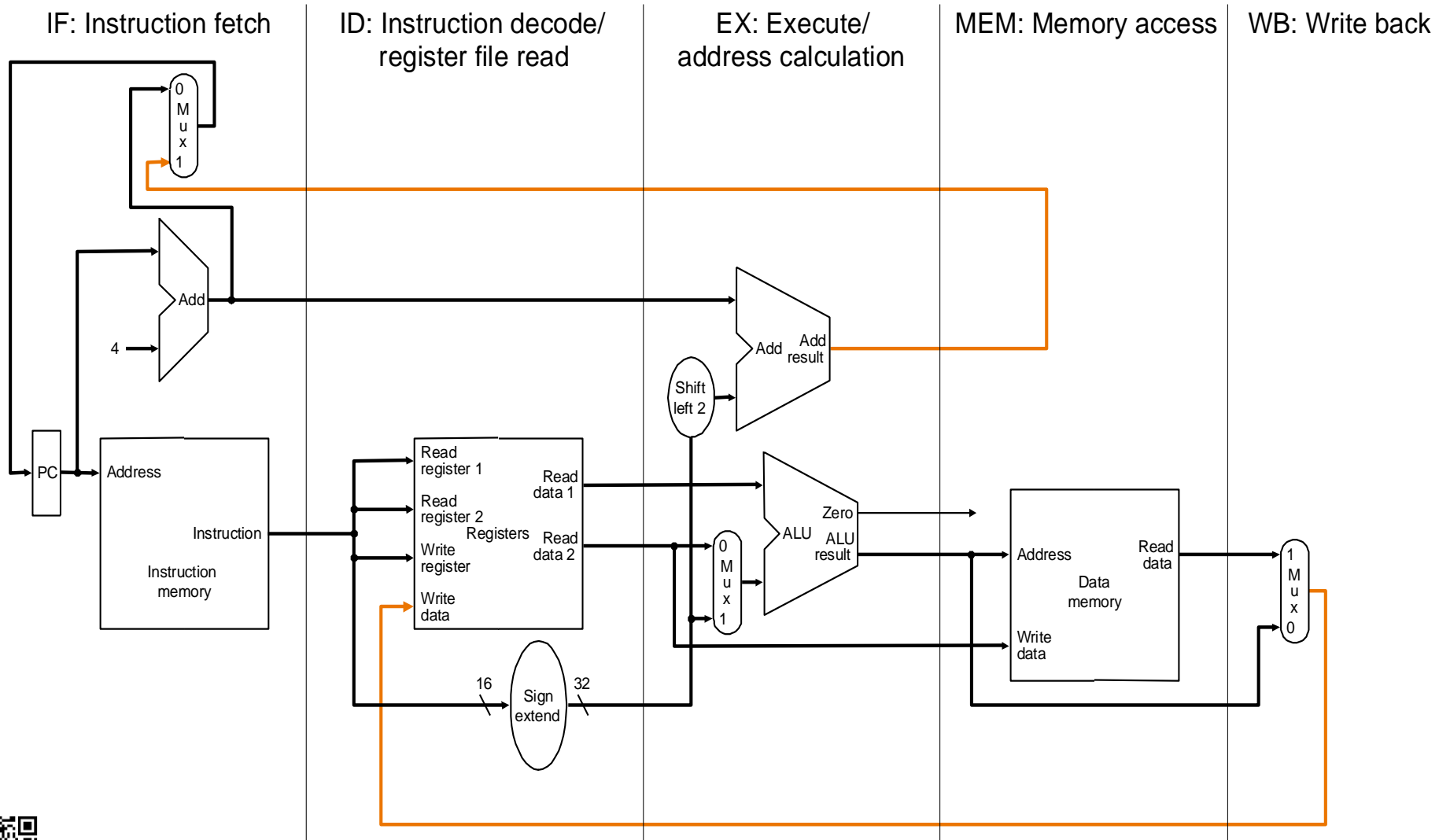
2. Pipelined Execution: Illustration

Time

IF	ID	EX	MEM	WB

| | IF | ID | EX | MEM | WB |

| | | IF | ID | EX | MEM | WB |

| | | | IF | ID | EX | MEM | WB |

Program Flow (Instructions)

| | | | | IF | ID | EX | MEM | WB |

| | | | | | IF | ID | EX | MEM | WB |
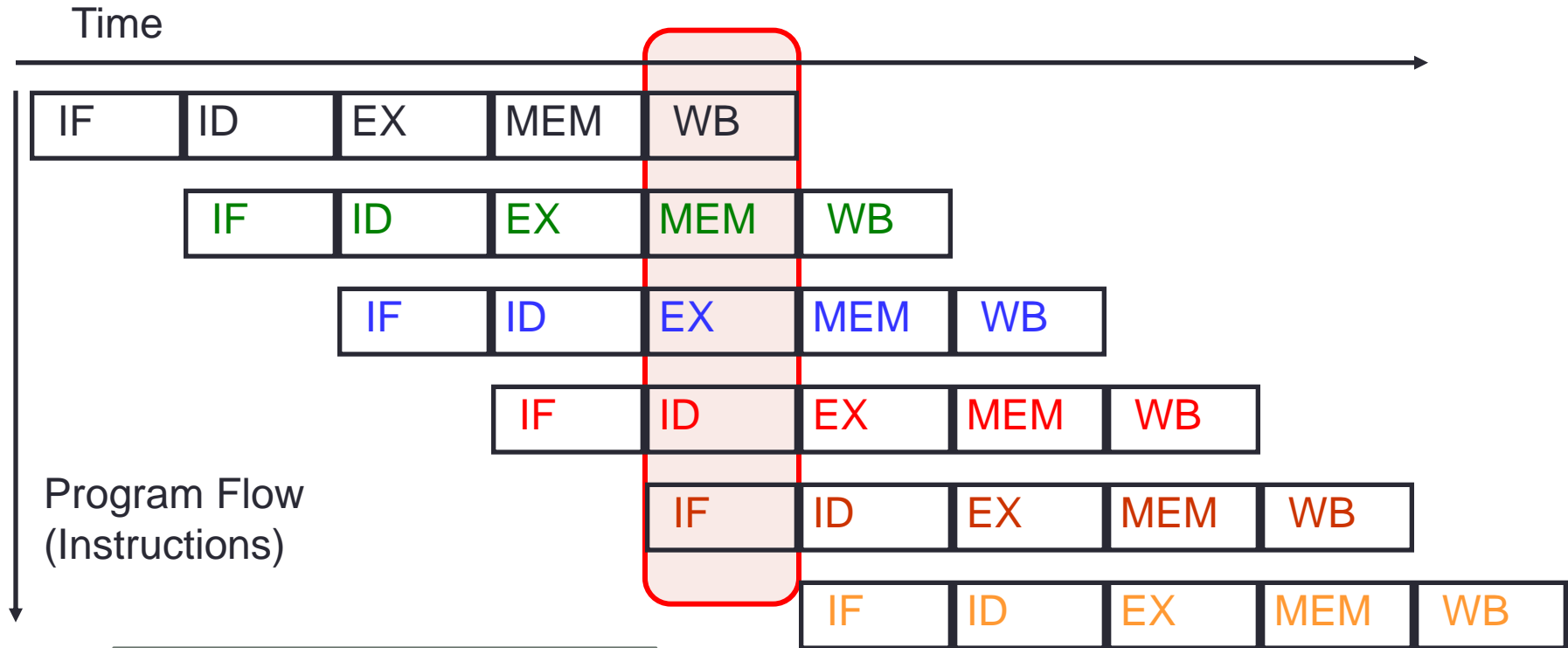
Several instructions are in the pipeline simultaneously!

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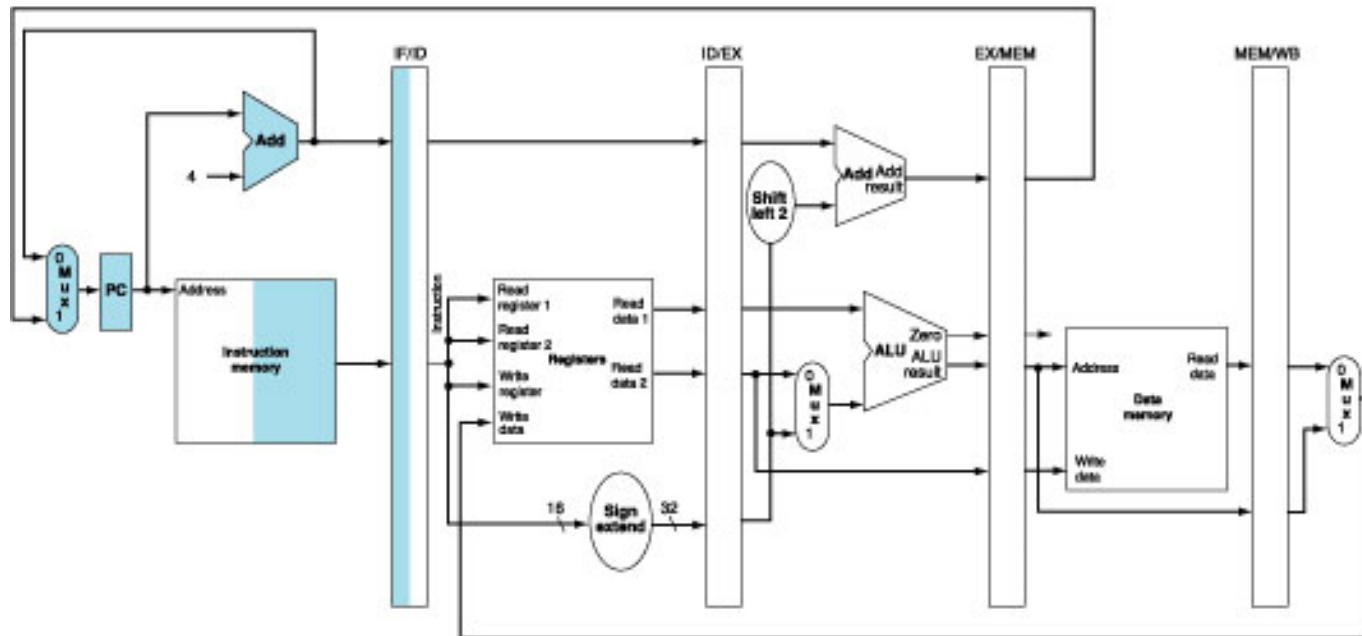
- At the end of a cycle, **IF/ID** receives (stores):
  - Instruction read from InstructionMemory[PC]
  - PC + 4
- PC + 4
  - Also connected to one of the MUX's inputs (another coming later)

3. Pipeline Datapath: **ID Stage**
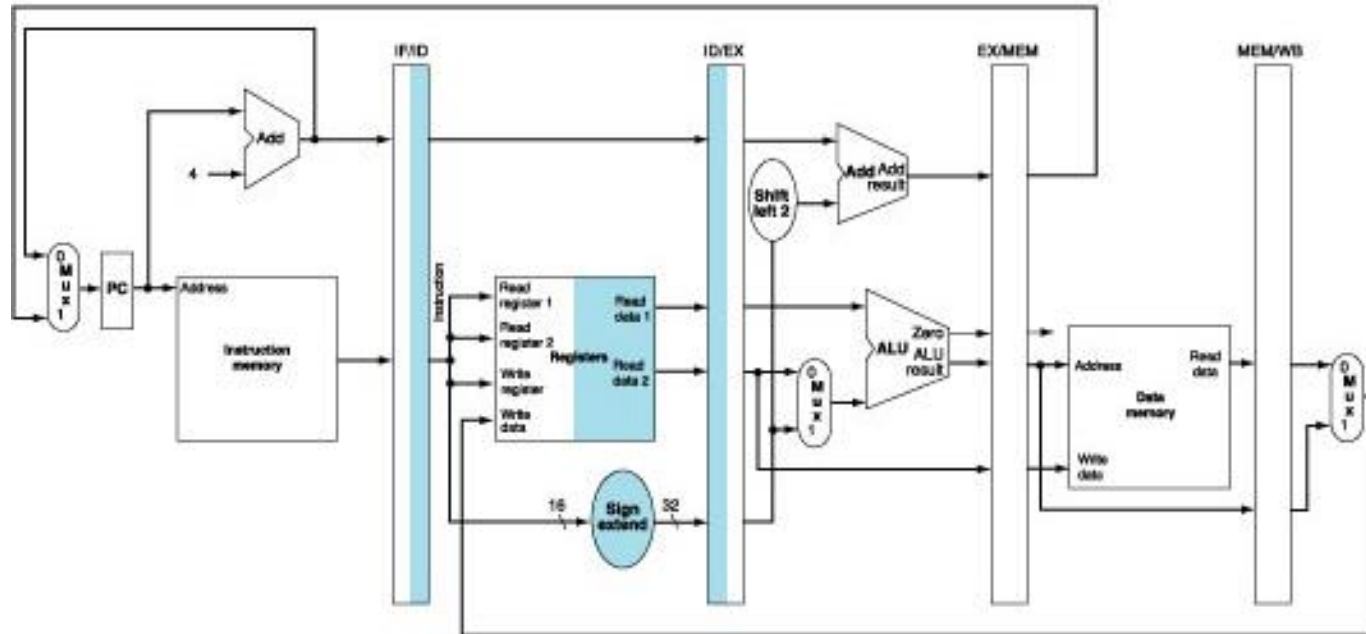
At the beginning of a cycle **IF/ID register supplies:**	At the end of a cycle **ID/EX receives:**
❖ Register numbers for reading two registers ❖ 16-bit offset to be sign-extended to 32-bit ❖ `PC + 4`	❖ Data values read from register file ❖ 32-bit immediate value ❖ `PC + 4`

3. Pipeline Datapath: **EX Stage**
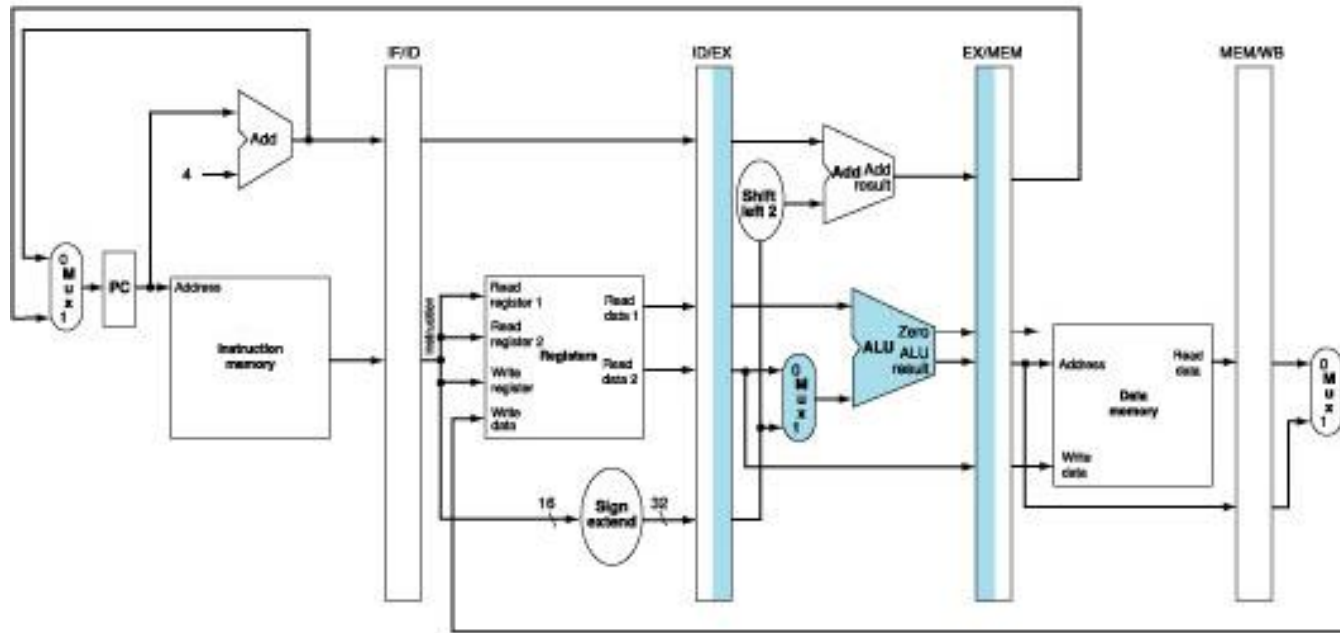
At the beginning of a cycle **ID/EX register supplies:**	At the end of a cycle **EX/MEM receives:**
❖ Data values read from register file ❖ 32-bit immediate value ❖ **PC + 4**	❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ **isZero?** signal ❖ Data Read 2 from register file

3. Pipeline Datapath: MEM Stage
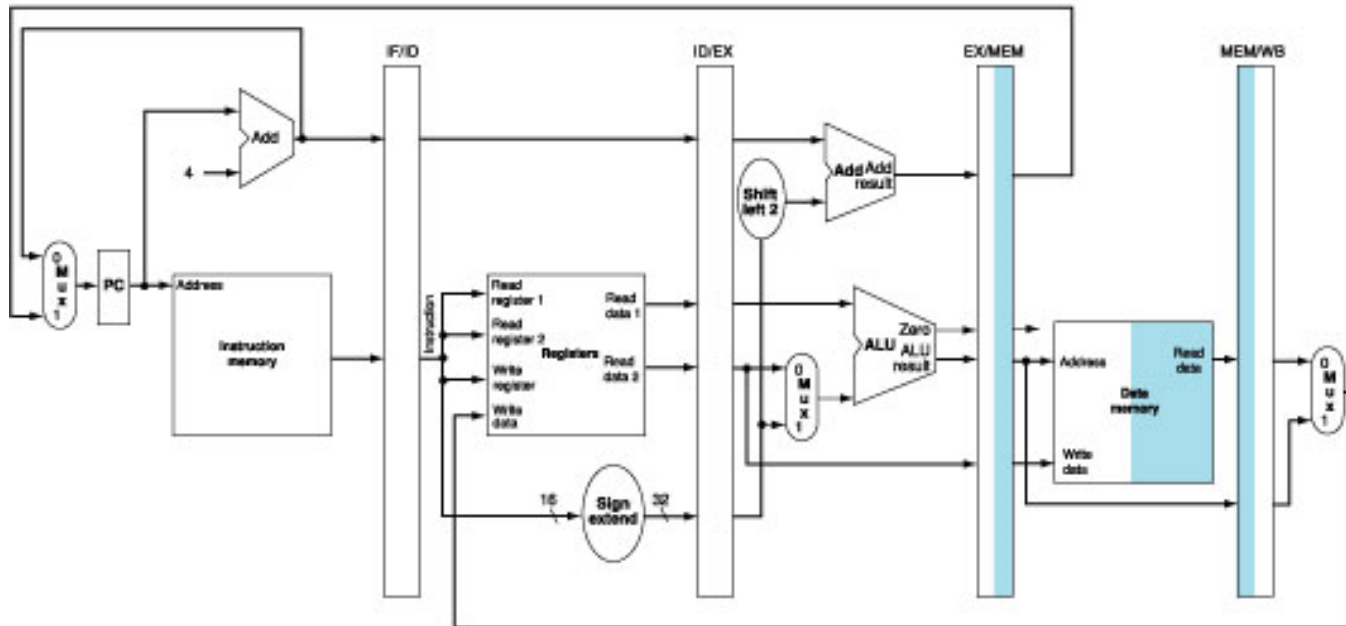
At the beginning of a cycle EX/MEM register supplies:	At the end of a cycle MEM/WB receives:
❖ (PC + 4) + (Immediate x 4) ❖ ALU result ❖ isZero? signal ❖ Data Read 2 from register file	❖ ALU result ❖ Memory read data

3. Pipeline Datapath: WB Stage
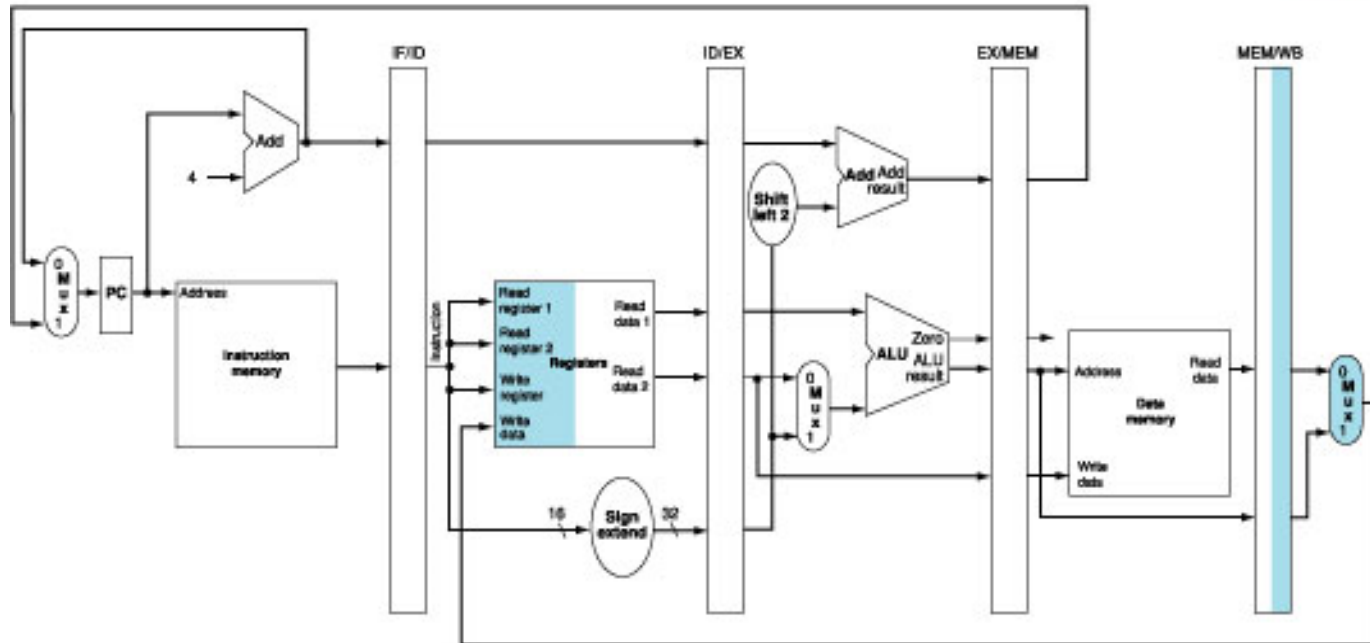
At the beginning of a cycle MEM/WB register supplies:	At the end of a cycle
❖ ALU result ❖ Memory read data	❖ Result is written back to register file (if applicable) ❖ **There is a bug here…….**

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)
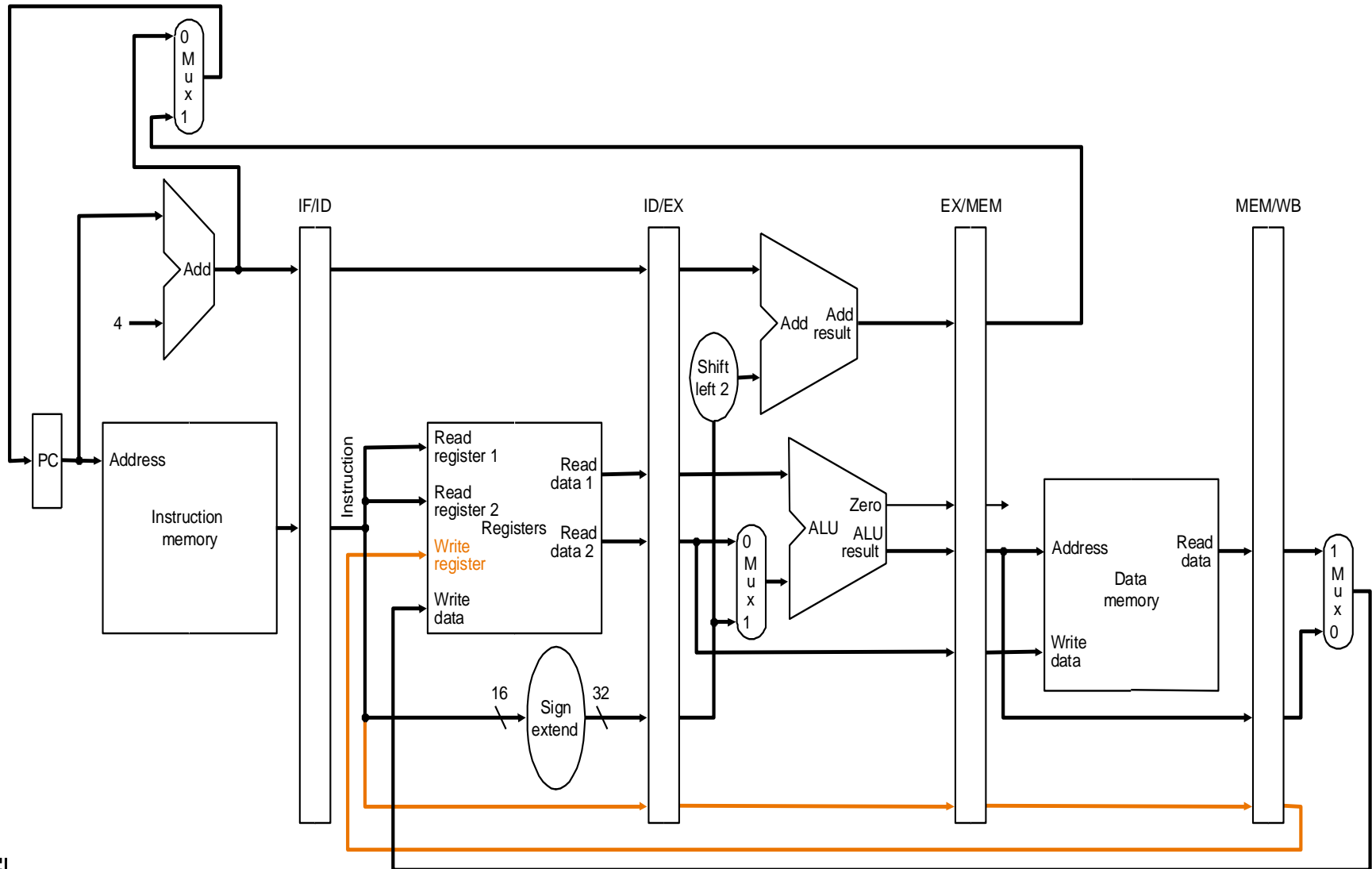
4. Pipeline Control: **Main Idea**

- 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

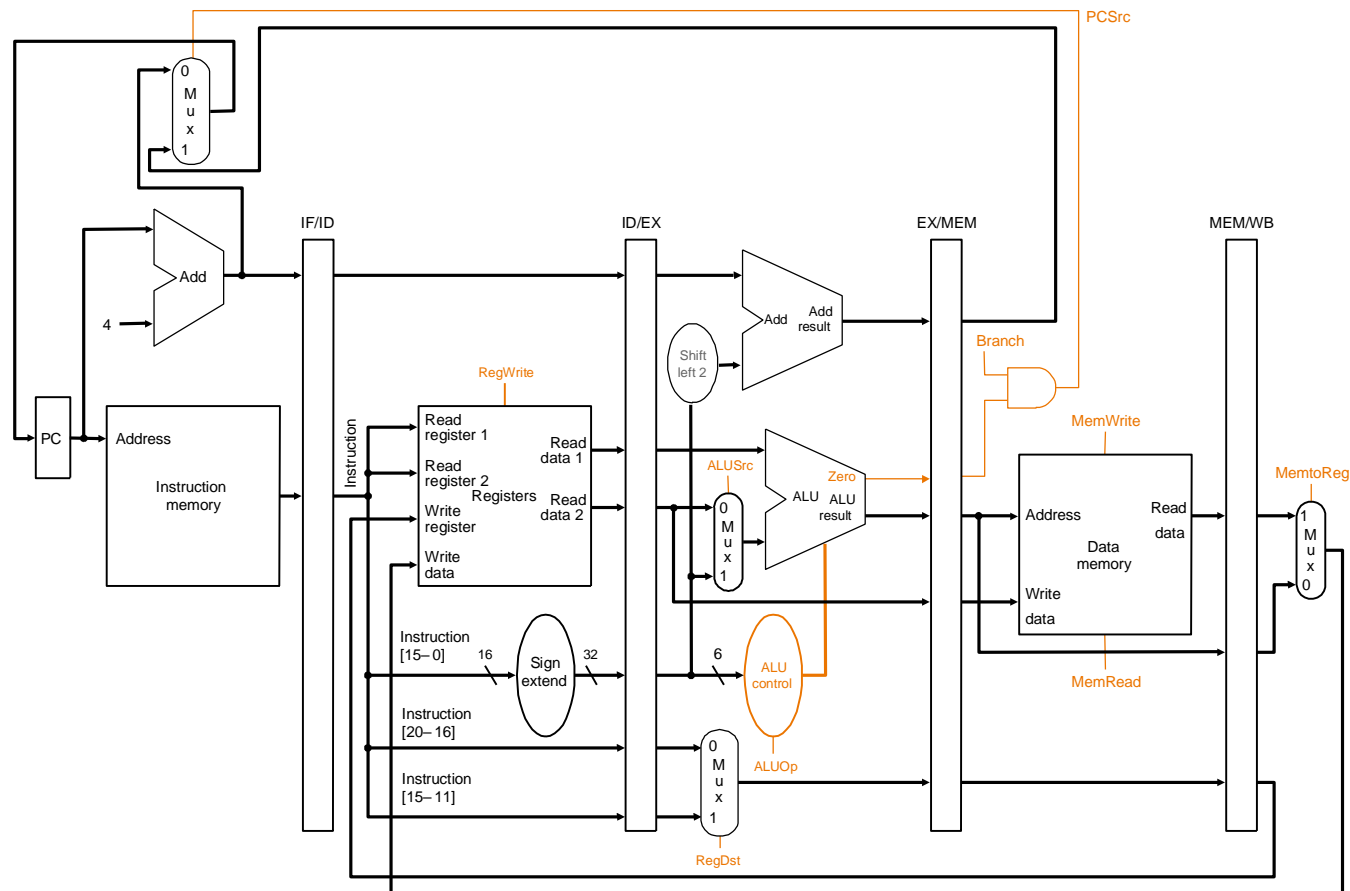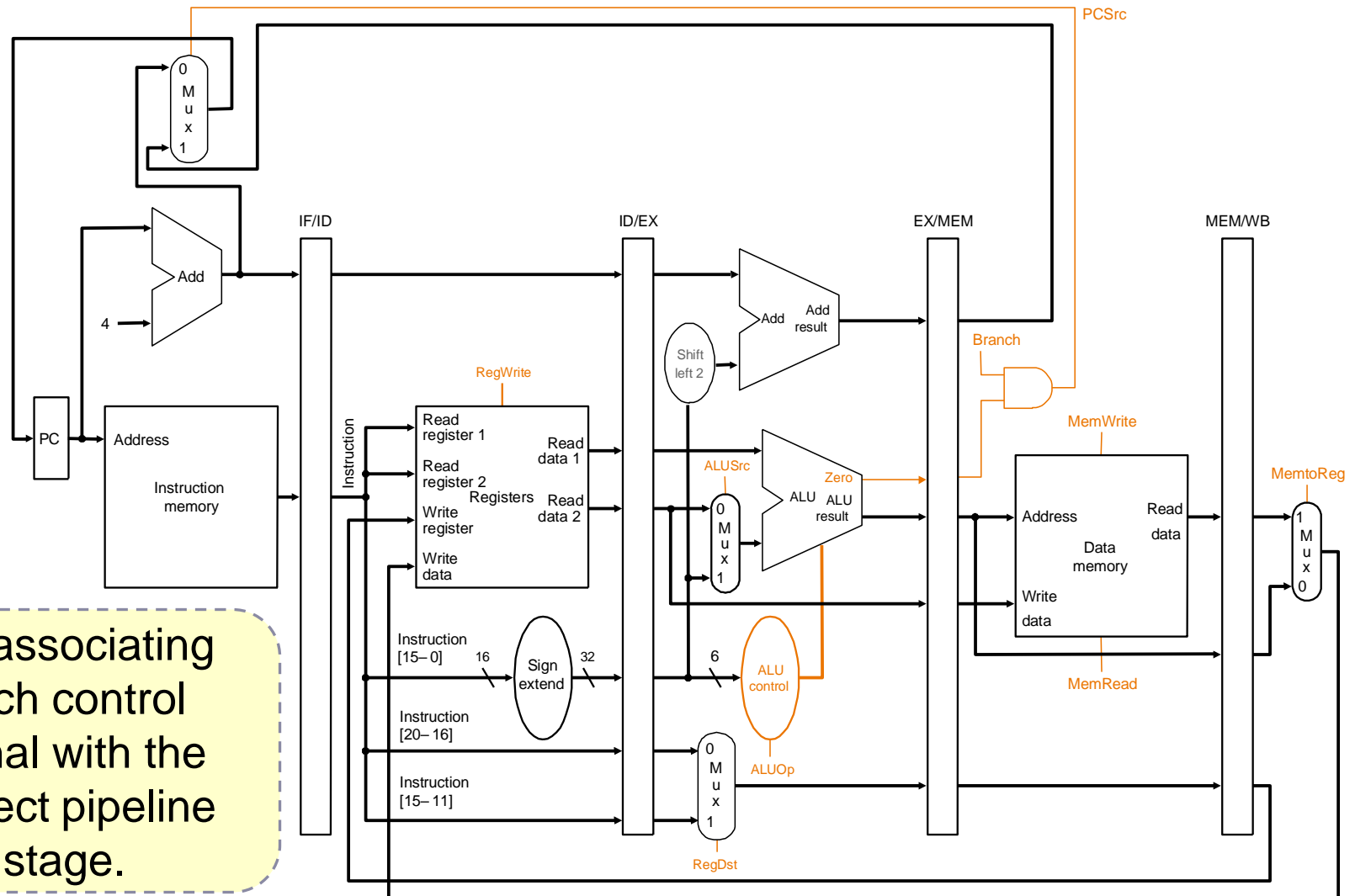|	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

	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**
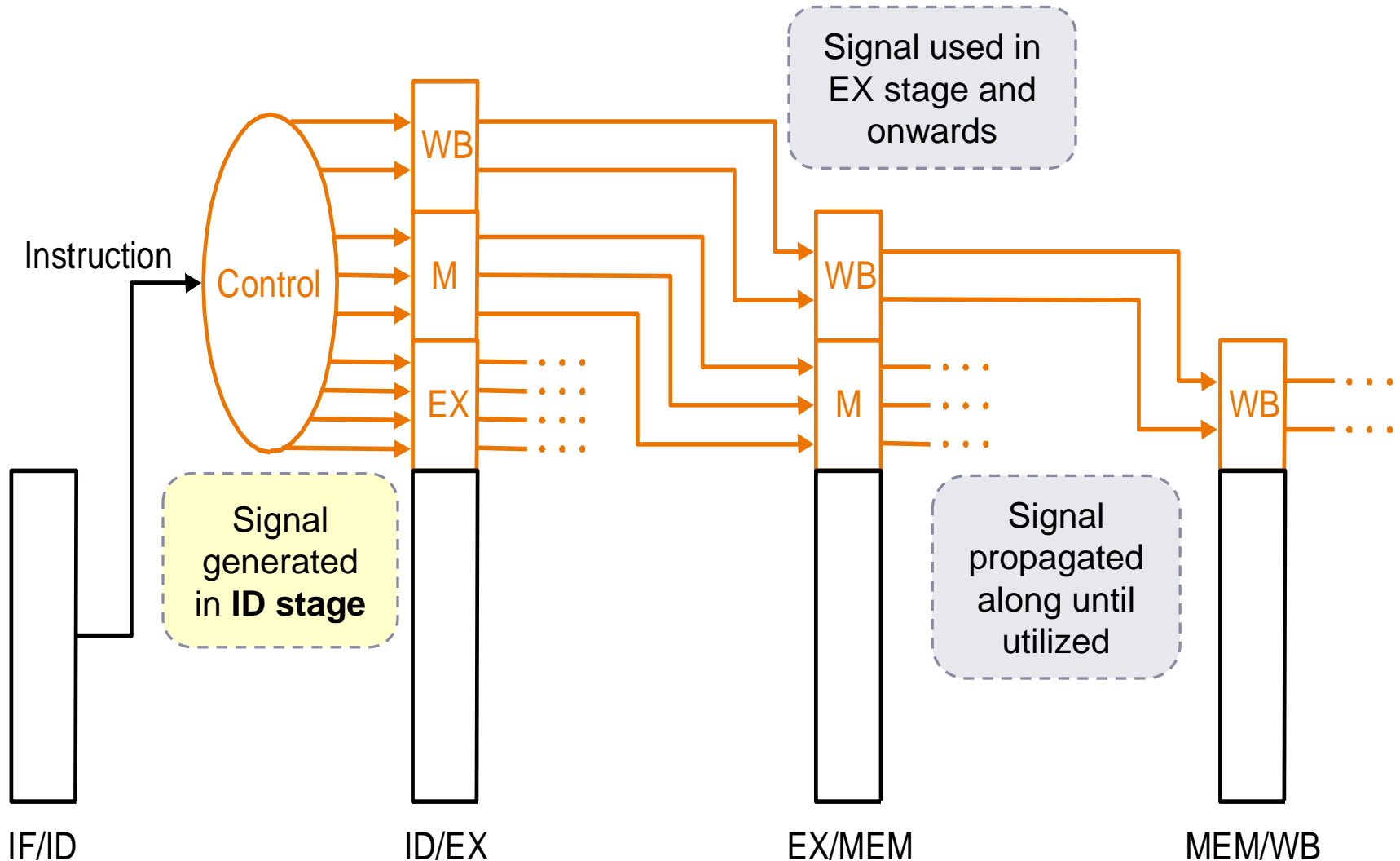
4. Pipeline Control: **Datapath** and **Control**
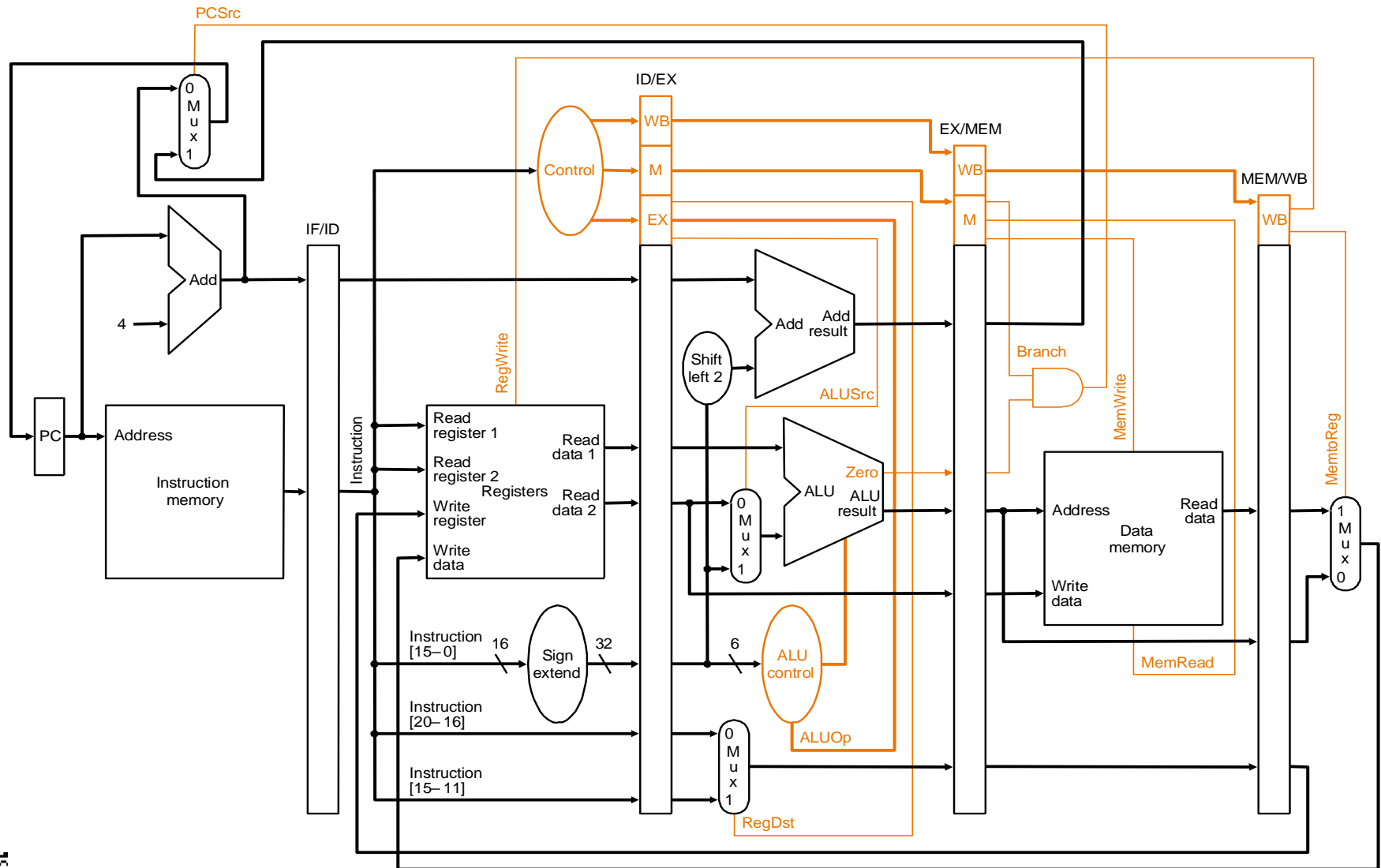
5. Different Implementations

Single-Cycle

Cycle 1	Cycle 2

Clk

Load	Store	Waste

Multi-Cycle

Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9	Cycle 10

Clk

Load   **Store**   **R-type**

IF	ID	EX	MEM	WB	IF	ID	EX	MEM	IF

Pipeline

Load

IF	ID	EX	MEM	WB

Store

IF	ID	EX	MEM	WB

R-type

IF	ID	EX	MEM	WB

5. Comparison of **Performance** (1/2)

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU (eg: add)	2	1	2		1	**6**
lw	2	1	2	2	1	**8**
sw	2	1	2	2		**7**
beq	2	1	2			**5**

Assume 100 instructions.

Single-Cycle Processor

- Cycle time: $CT_{seq} = \max(\sum_{k=1}^{N} T_k)$
  - T_k = Time for operation in stage *k*
  - *N* = Number of stages
- Execution Time for **I** instructions:
  - $Time_{seq} = I \times CT_{seq}$

Cycle time = 8ns

Time to execute 100 instructions =100×8ns=800ns

Multi-Cycle Processor

- Cycle time: $CT_{multi} = \max(T_k)$
- Execution Time for **I** instructions:
  - $Time_{multi} = I \times Average\ CPI \times CT_{multi}$
  - Average CPI needed as each instruction takes different number of cycles

Cycle time = 2ns

Given average CPI = 4.6

Time to execute 100 instructions = 100 × 4.6 × 2ns = 920ns

5. Comparison of **Performance** (2/2)

Single-Cycle Processor

- Cycle time: $CT_{seq} = \max(\sum_{k=1}^{N} T_k)$
  - T_k = Time for operation in stage k
  - N = Number of stages
- Execution Time for **I** instructions:
  - $Time_{seq} = I \times CT_{seq}$

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU (eg: add)	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

Assume 100 instructions.

Cycle time = 8ns

Time to execute 100 instructions =100×8ns=800ns.

Pipelining Processor

- Cycle time: $CT_{pipeline} = \max(T_k) + T_d$
  - T_d = Overhead for pipelining, e.g. pipeline register
- Cycles needed for **I** instructions:
  - $I + N - 1$ (need $N - 1$ cycles to fill up the pipeline)
- Execution Time for **I** instructions:
  - $Time_{pipeline} = (I + N - 1) \times CT_{pipeline}$

Assume pipeline register latency = 0.5ns

Cycle time = 2ns + 0.5ns = 2.5ns

Time to execute 100 instructions = (100+5–1) × 2.5ns = 260ns.

5. Pipeline Processor: **Ideal Speedup**

- Assumptions for ideal case:
  - Every stage takes the same amount of time $\rightarrow \sum_{k=1}^{N} T_k = N \times T_1$

  - No pipeline overhead $\rightarrow T_d = 0$

  - $I >> N$ (Number of instructions is much larger than number of stages)

- $Speedup_{pipeline} = \dfrac{Time_{seq}}{Time_{pipeline}}$

$= \dfrac{I \times \sum_{k=1}^{N} T_k}{(I+N-1) \times (\max(T_k)+T_d)} = \dfrac{I \times N \times T_1}{(I+N-1) \times T_1}$

$\approx \dfrac{I \times N \times T_1}{I \times T_1}$

$= N$

> **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) $4/(100 \times 10^6) = 40$ ns

c) $4 + 4 = 8$ cycles

d) $8/(500 \times 10^6) = 16$ 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?

End of File