CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Rajat Subhra Chakraborty (*RSC*)

Bhargab B. Bhattacharya (*BBB*)

Lecture #39

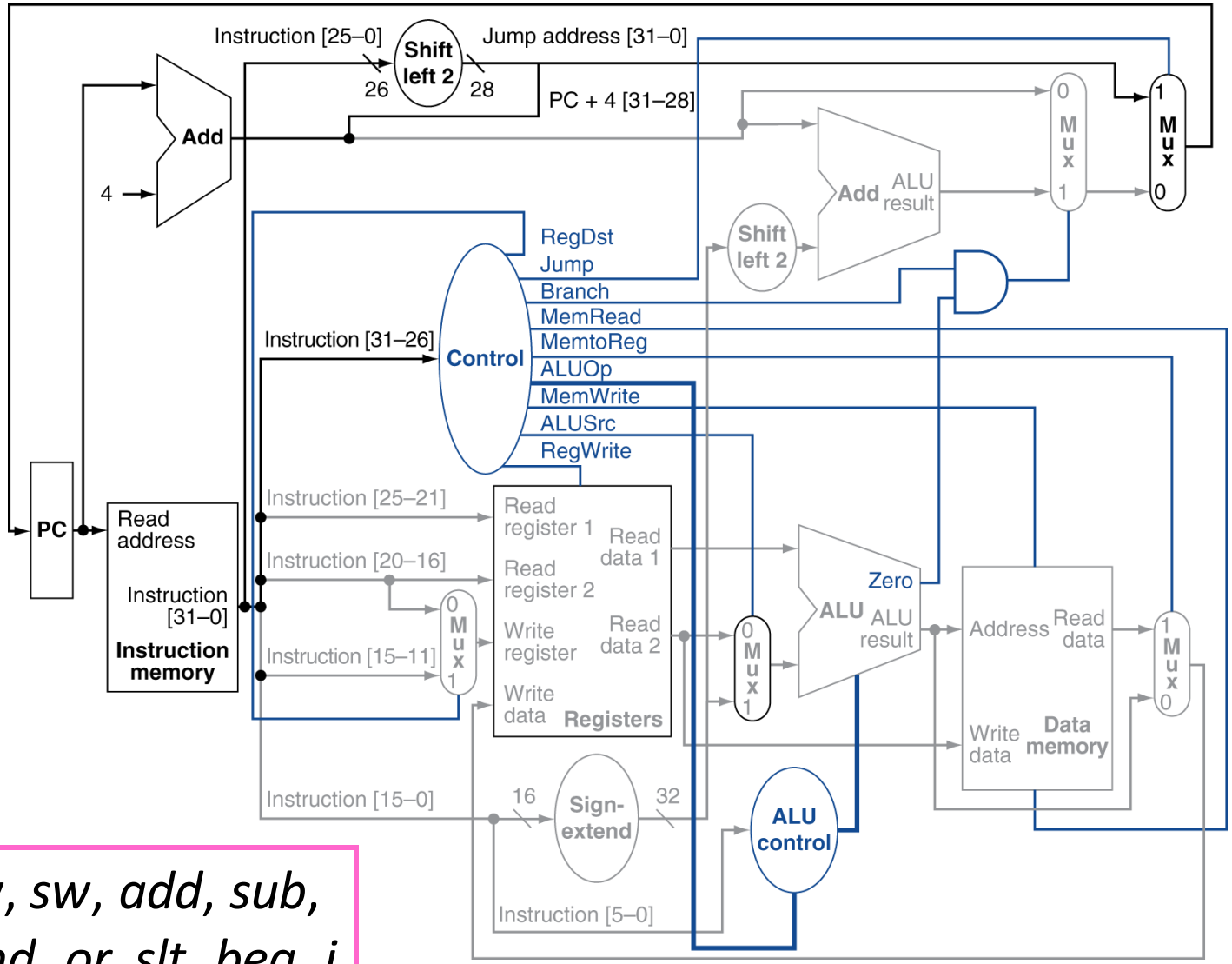
Processor Design: Pipelining

01 November 2021

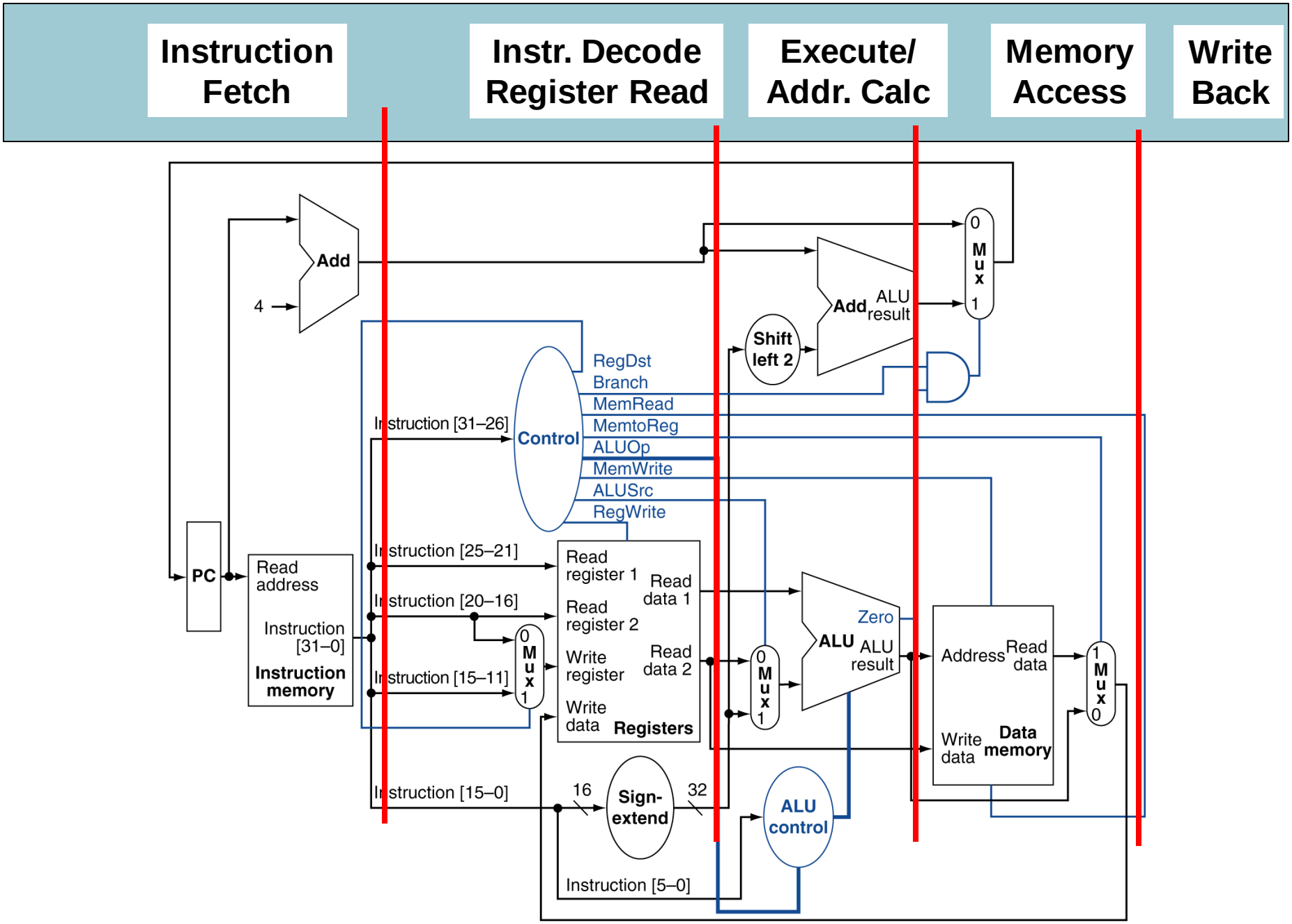
Indian Institute of Technology Kharagpur

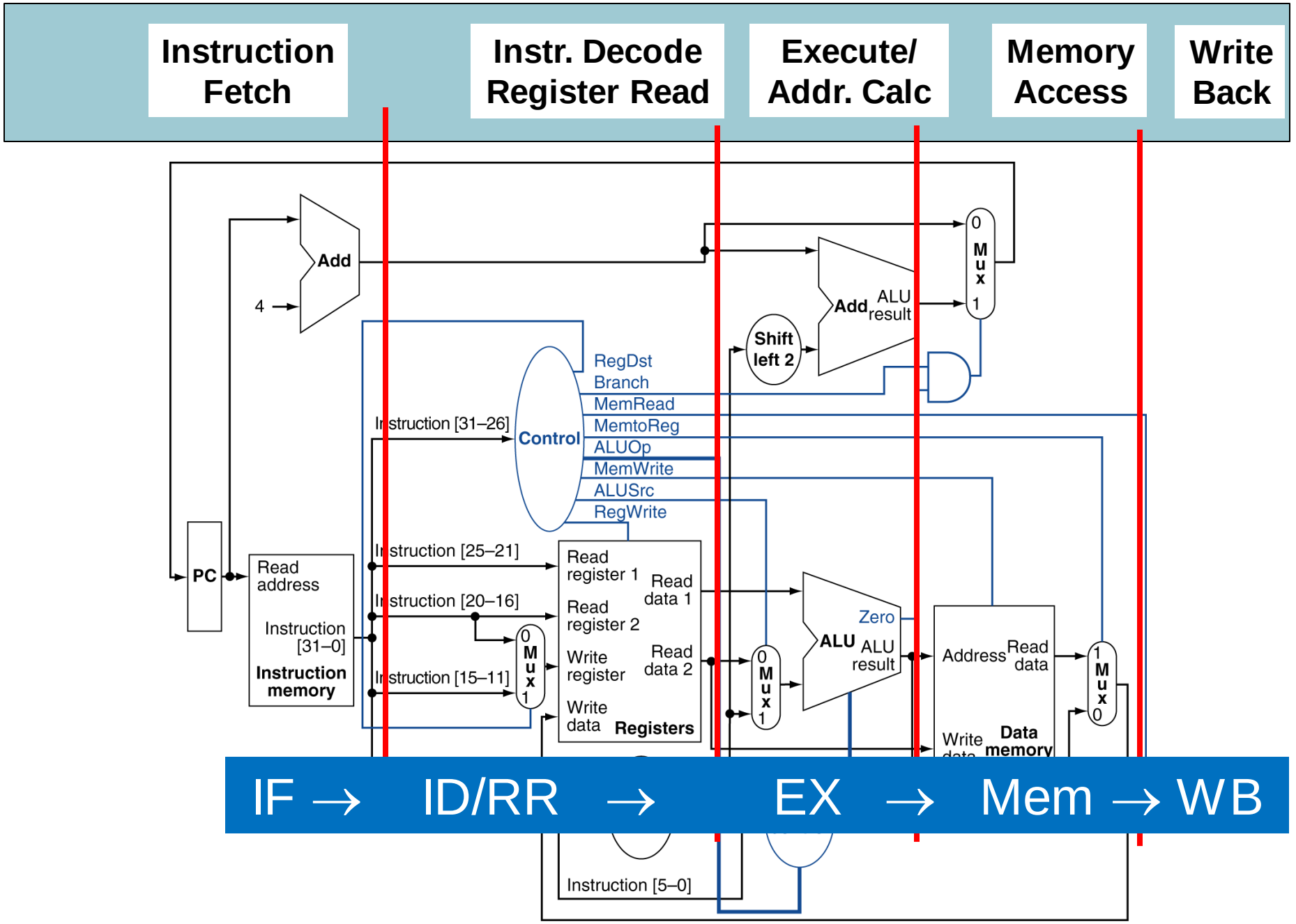
Computer Science and Engineering

Recap: What is it?



lw, sw, add, sub, and, or, slt, beq, j

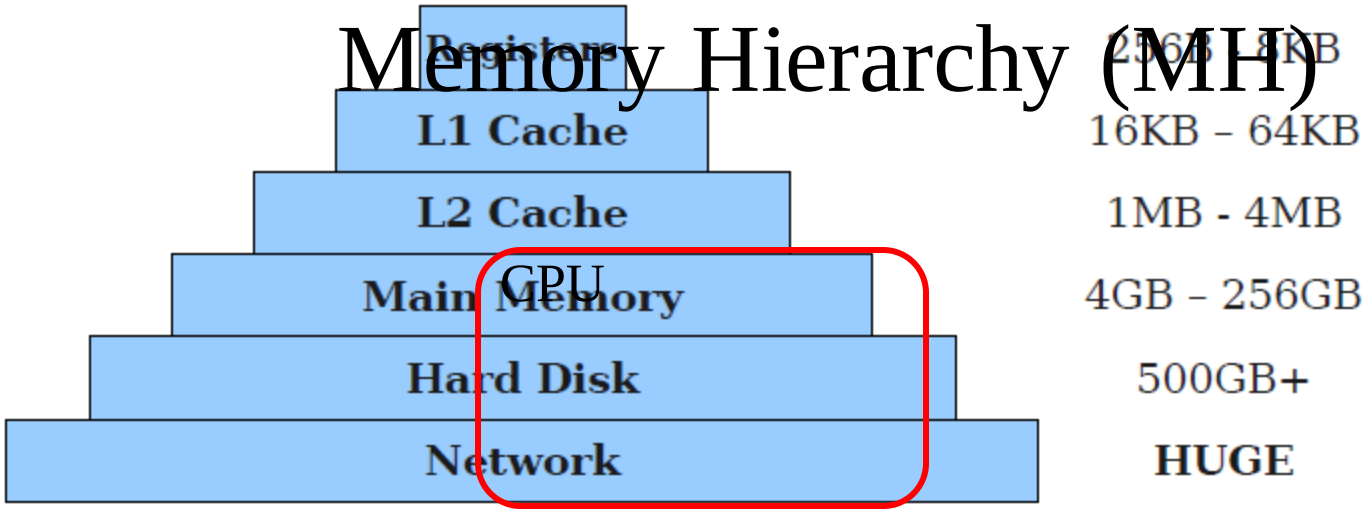

Instruction Fetch
Instr. Decode Register Read
Execute/ Addr. Calc
Memory Access
Write Back
Add
4
Add
ALU result
Shift left 2
M u x
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
16
Sign-extend
32
ALU control

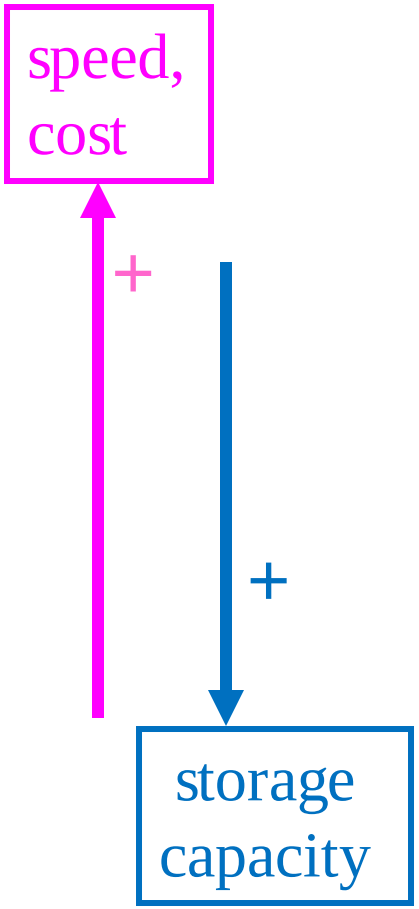

Instruction Fetch
Instr. Decode Register Read
Execute/ Addr. Calc
Memory Access
Write Back
Add
4
Add
ALU result
Shift left 2
M u x
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
PC
Read address
Instruction [31–0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [5–0]
IF → ID/RR → EX → Mem → WB

Single-Cycle Processor Design

What are the design-characteristics/ consequences of having “single-cycle” based processor?

Memory Hierarchy (MH)

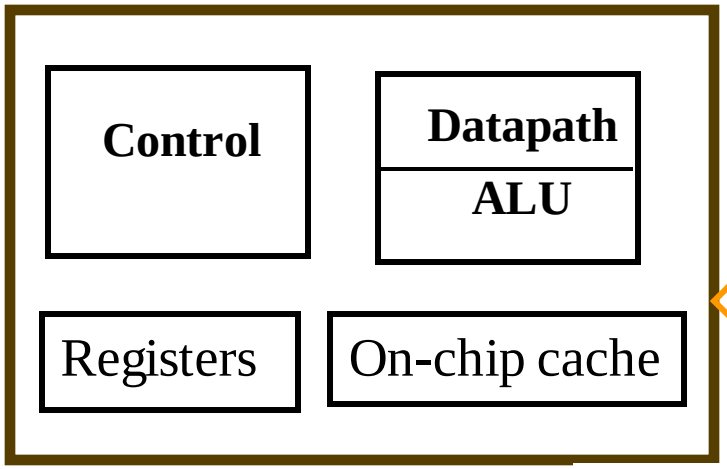




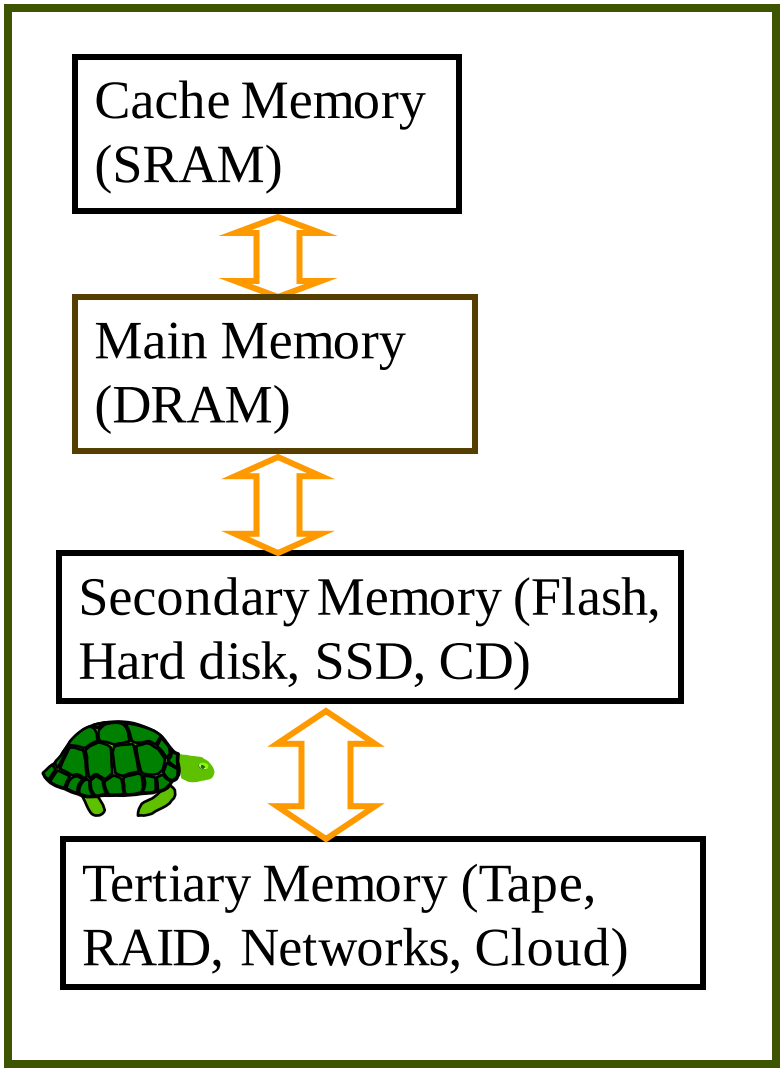
Complete View of Computer Architecture

Memory transaction is inherently slow compared to logic

Use of memory hierarchy reduces the transaction time across CPU-memory interface



Processor (CPU)

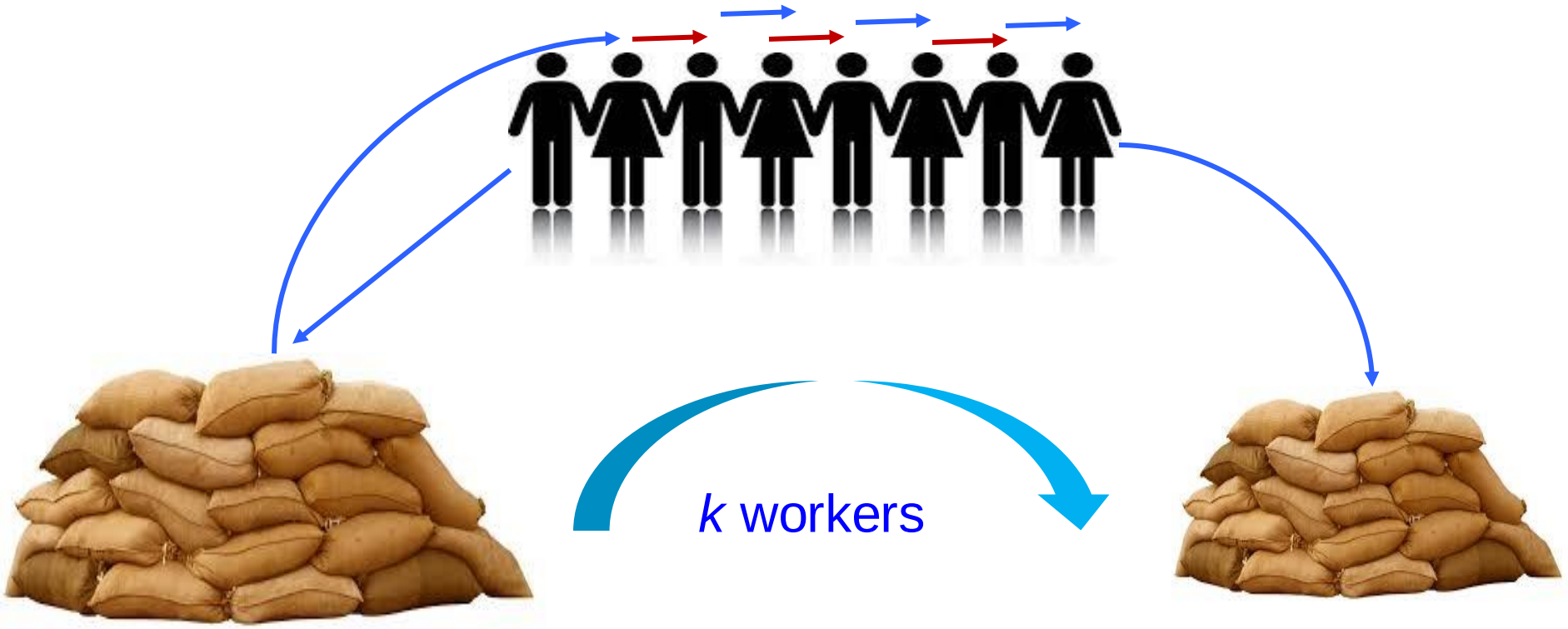


Memory

Logical address *vs.* physical address;

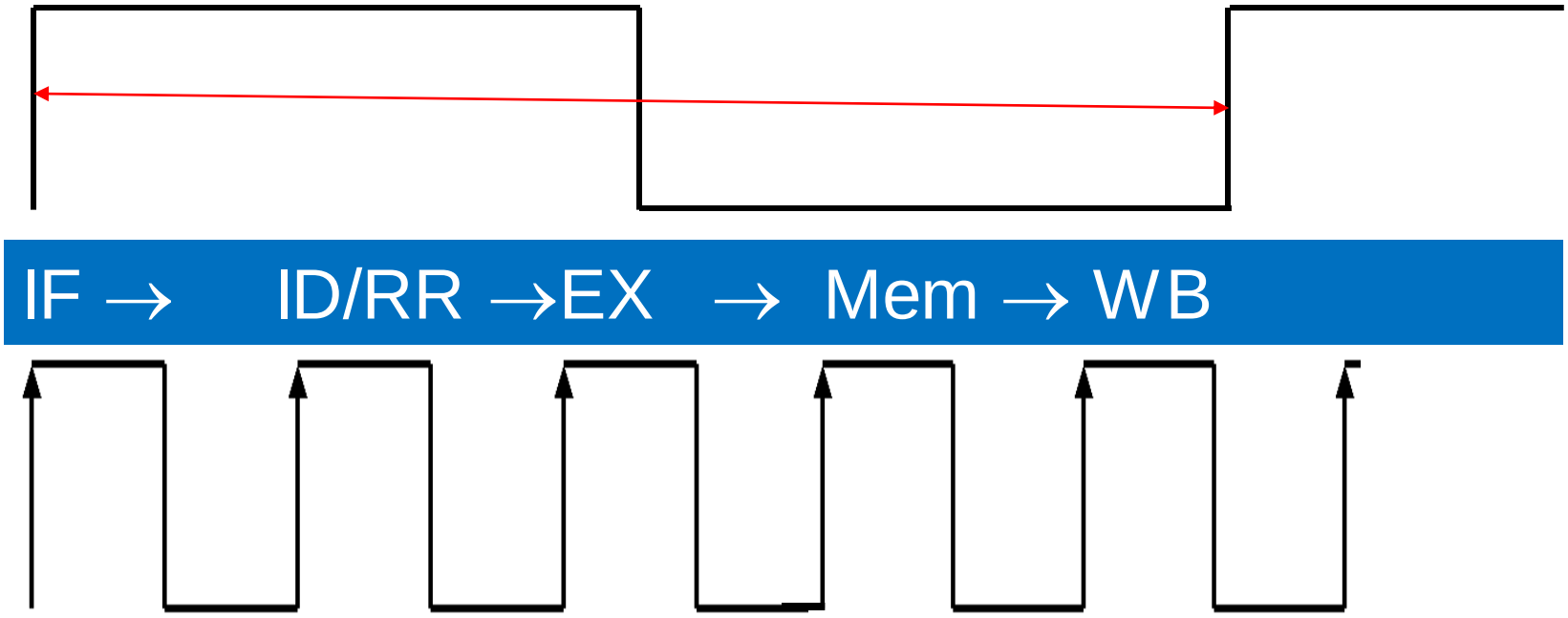
Consequences of having single-cycle processor design?

Bag transportation via human chain



Processor Design

Single-Cycle Processor

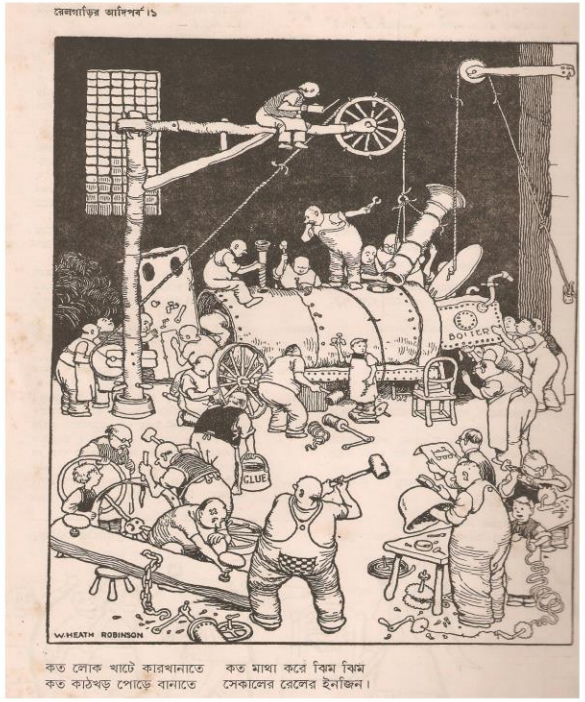


Multi-Cycle Processor

Description

Job: Manufacture *n* cars or locomotive engines

Naïve Approach: Engage all labor and machinery to finish building one car, and proceed to produce the next;



Courtesy: Satyajit Ray

Assembly Line

Assembly Line

Basic Idea

- Chassis is moved through different stations (phases) in sequence, comprising relevant workforce
- A specific sub-task is performed in each station
- Once finished job at this station, the next chassis arrives
- Multiple stations are busy at any time instance (sub-tasks overlapped in time)
- Each station performs only one specific job every time (non-overlapped in space)
- Certain precedence among tasks should be honored
- Phases should preferably be balanced in time

The concept of *pipelining* is very old

It had been in practice in industrial assembly-lines (e.g., automobiles) for a long time

Description: Assembly Line

Job: Manufacture *n* cars or locomotive engines

Pipelined Approach: Divide, Forward, and Conquer

Multiple phases are needed to complete production;

Each phase needs specific expertise/ instruments/time

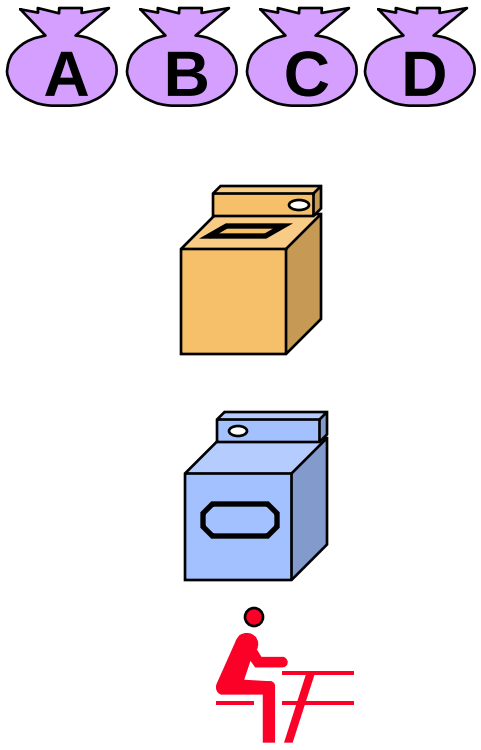
Goal: How to optimize the total completion time (cost) for producing *n* cars

What is Pipelining?

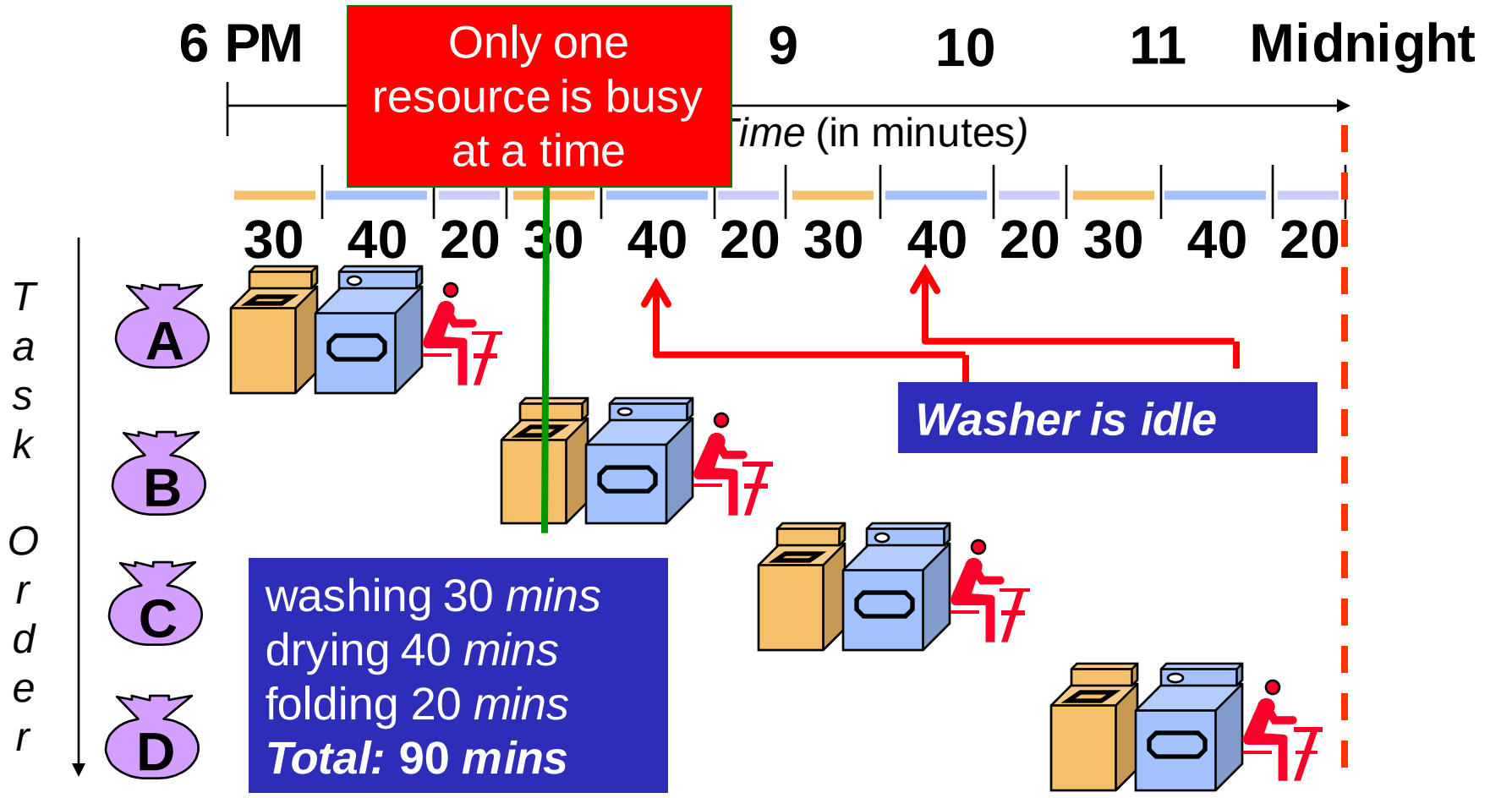
"Tell me...what IS in the pipeline?"

What is Pipelining

- Laundry Room
- Four students A, B, C, D each has one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- Folder takes 20 minutes



What Is Pipelining: Laundry Example

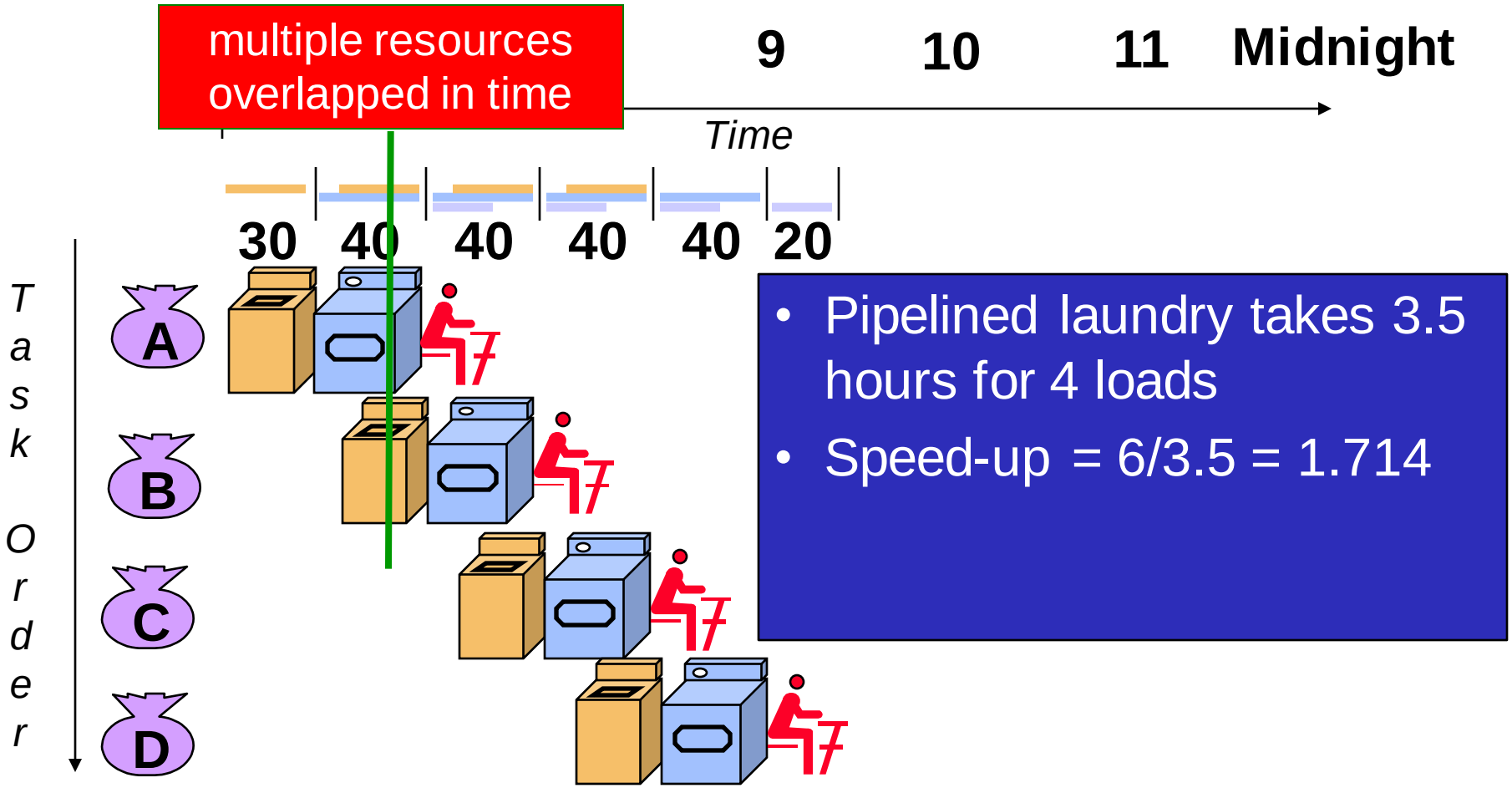


Sequential laundry takes 6 hours for four users; Work finishes at 00:00

If they learned pipelining, how long would laundry take?

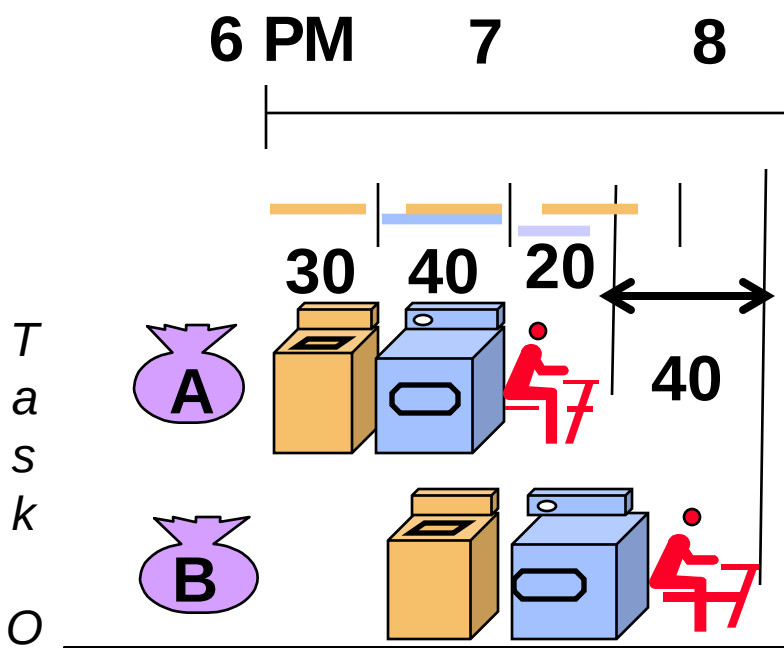
Basic Idea of Pipelining

Minimize idle-time for resources



- Pipelined laundry takes 3.5 hours for 4 loads
- Speed-up = 6/3.5 = 1.714

Impact of Pipeline Stages



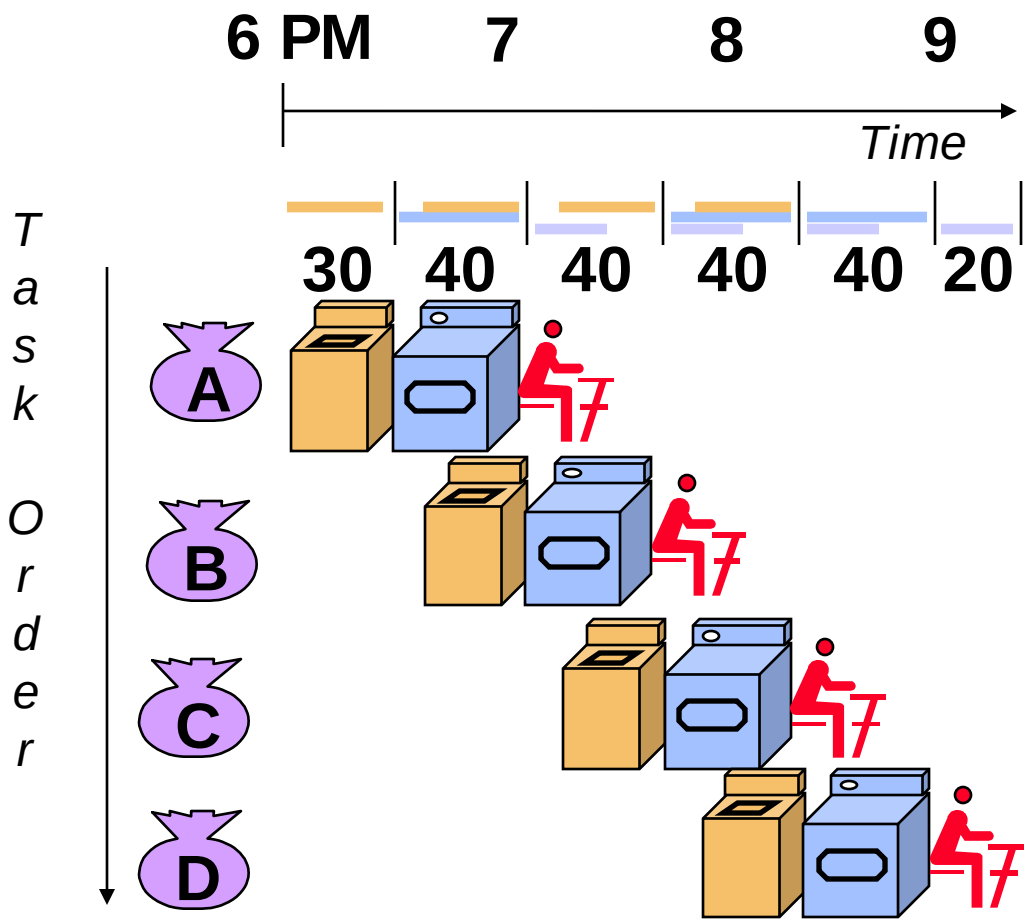
- Q1. What is the time difference between two consecutive finish-times?
- time required to move an instruction one step forward
- → longest time (slowest) among pipe-stages

Q2. What is the maximum speed-up compared to non-pipelined approach??

→ limited by the number of pipe-stages

- Speed-up when $n \rightarrow \infty$
  $\approx 90/40 = 2.25$
  < 3 (number of pipe-stages)

Pipelining Lessons



- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Pipeline rate limited by **slowest** pipeline stage
- **Multiple** tasks operating simultaneously
- Potential speed-up ≤ **Number pipe stages (pipe-depth)**
- Unbalanced lengths of pipe stages reduces speed-up
- Time to "**fill**" pipeline and time to "**drain**" it reduces speedup
- **Stall for Dependences**

Learned So Far ….

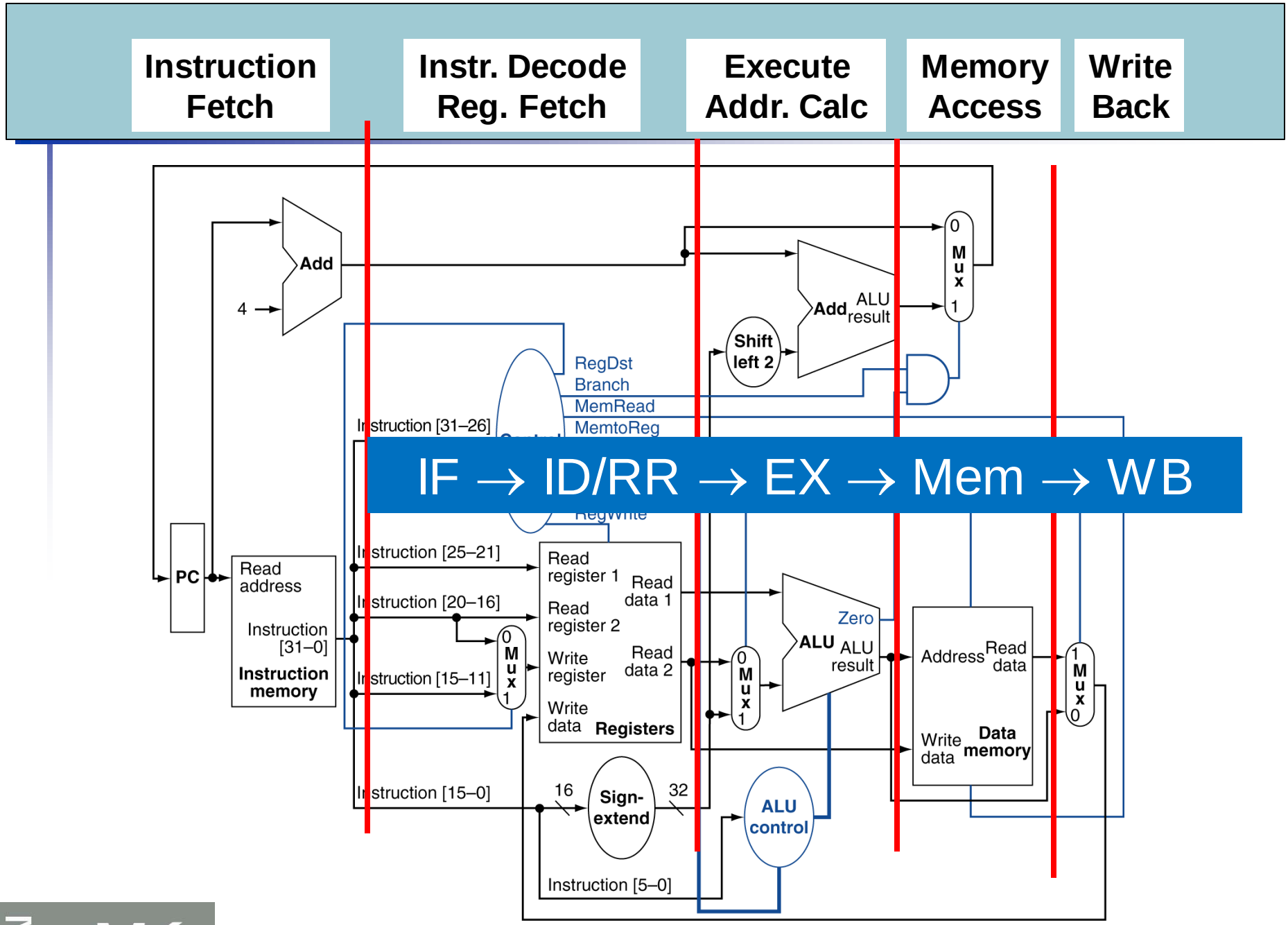
- Pipelining is widely used in modern processors.
- Pipelining improves system performance in terms of throughput.
- Pipelined organization requires sophisticated compilation techniques.

Making Program Execution Faster

- Arrange the hardware so that more than one operation can be overlapped at the same time.
- The number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

MIPS Pipeline

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


Instruction Fetch
Instr. Decode Reg. Fetch
Execute Addr. Calc
Memory Access
Write Back
IF → ID/RR → EX → Mem → WB
Add
4
M u x
0
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
Instruction [31–26]
RegWrite
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
16
Sign-extend
32
ALU control

CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Rajat Subhra Chakraborty (*RSC*)

Bhargab B. Bhattacharya (*BBB*)

Lecture #40, #41

Processor Design: Pipelining

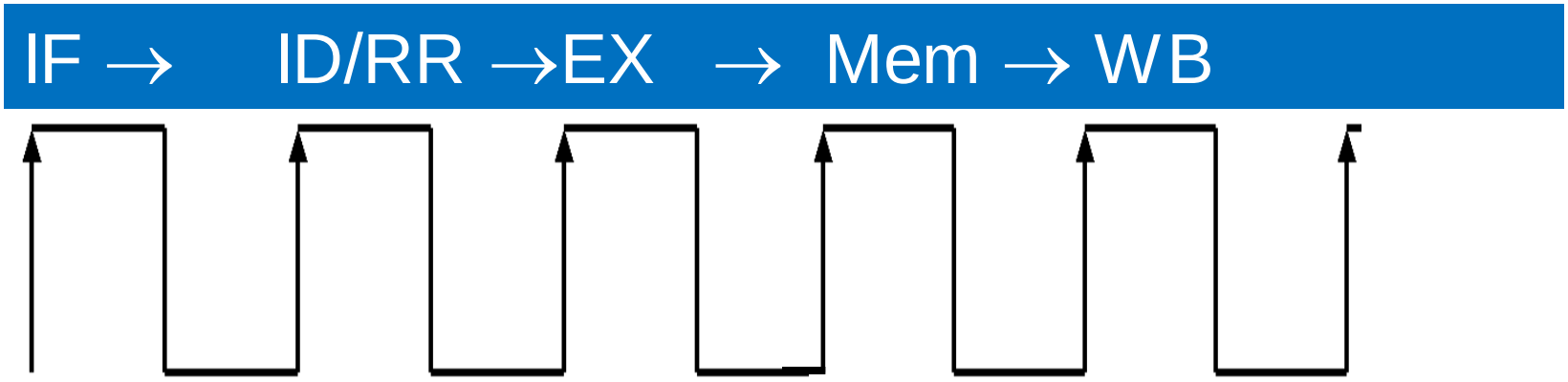
02 November 2021

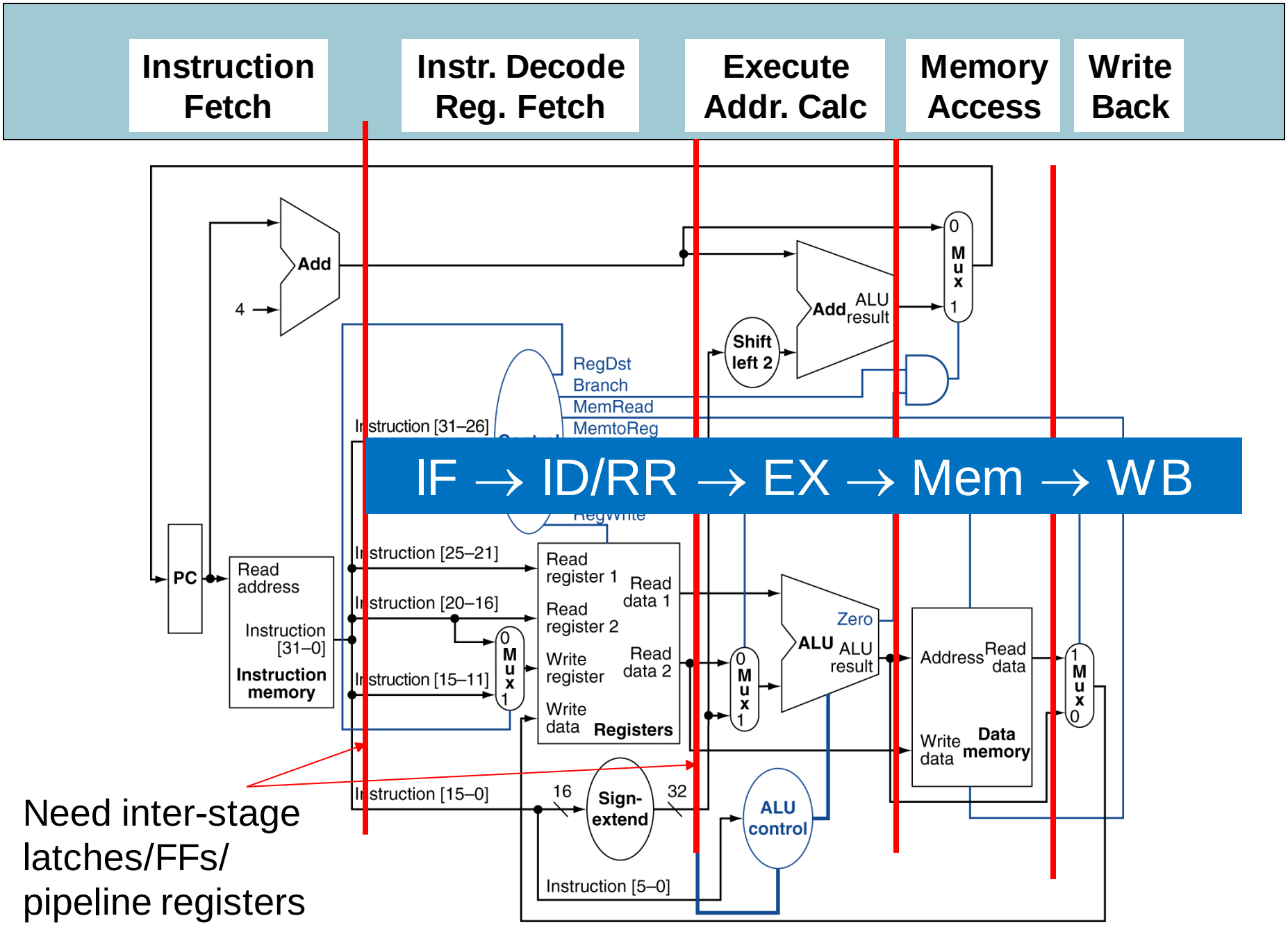
Indian Institute of Technology Kharagpur

Computer Science and Engineering

MIPS Pipeline

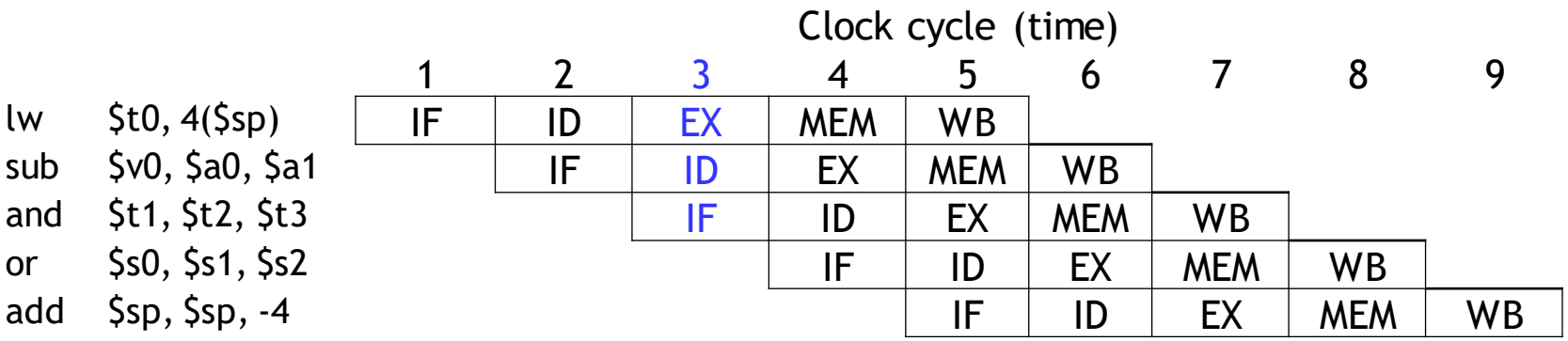
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID/RR: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register





Need inter-stage latches/FFs/ pipeline registers

A pipeline diagram

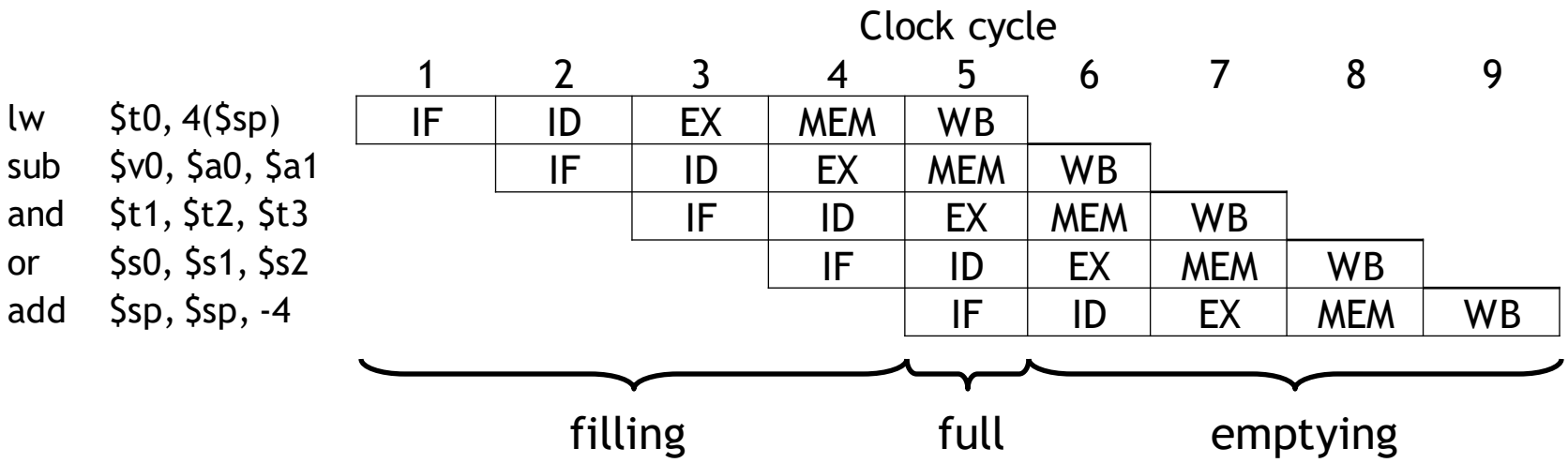


- A pipeline diagram shows the execution of a series of instructions.
  - The instruction sequence is shown vertically, from top to bottom.
  - Clock cycles are shown horizontally, from left to right.
  - Each instruction is divided into its component stages.
- Overlapping of instructions: five instructions are active in the fifth cycle:
- Why balanced stages??

- logistics;

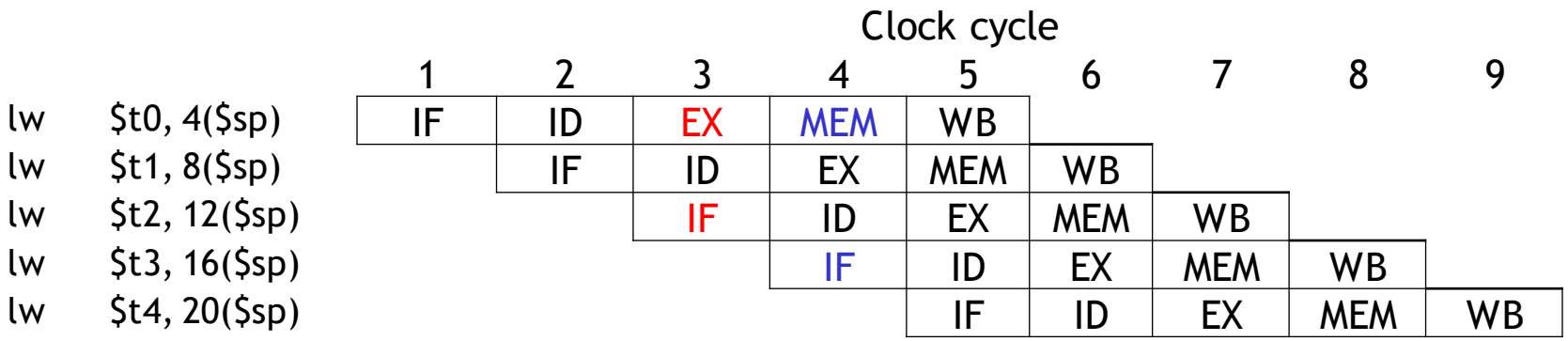
- speed-up is determined by the time needed by the longest stage, anyway;

Pipeline terminology



- The pipeline depth is the number of stages—in this case, five.
- In cycle 5, the pipeline is full. Five instructions are being executed simultaneously, so all hardware units are in use => speed-up $\leq$ 5

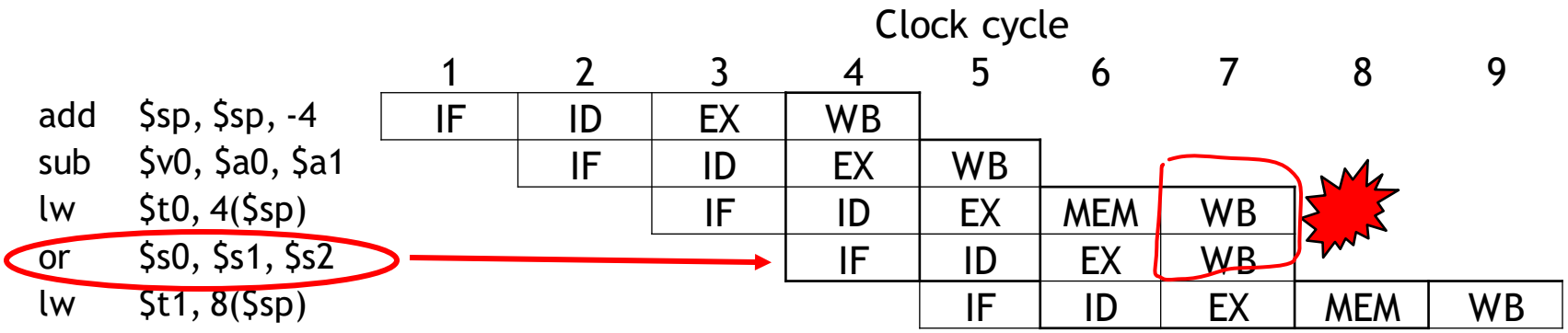
Pipeline Datapath: Resource Requirements



- We need to perform several operations in the same cycle.
  - Increment the PC and add registers at the same time.
  - Fetch one instruction while another one reads or writes data.
- What does this mean for our hardware?

Pipelining Multiple Instruction-Types

- R-type instructions only require 4 stages: IF, ID, EX, and WB
  - We don't need the MEM stage
- What happens if we try to pipeline **loads** with **R-type** instructions?



Solution??

A solution: Insert NOP (No-Operation) stages

- Enforce uniformity
  - Ensure that all instructions take 5 cycles/stages
  - Make them have the same stages, in the same order
    - Some stages will do nothing for some instructions

R-type	IF	ID	EX	NOP	WB

Clock cycle

		1	2	3	4	5	6	7	8	9
add	$sp, $sp, -4	IF	ID	EX	NOP	WB				
sub	$v0, $a0, $a1		IF	ID	EX	NOP	WB			
lw	$t0, 4($sp)			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	NOP	WB	
lw	$t1, 8($sp)					IF	ID	EX	MEM	WB

Mem: NOP

- Stores and Branches have NOP stages, too…

store	IF	ID	EX	MEM	NOP
branch	IF	ID	EX	NOP	NOP

WB: NOP
Mem: NOP,
WB: NOP

Learned So Far ….

- Pipelining attempts to maximize instruction throughput by overlapping, in time, the execution of multiple instructions.

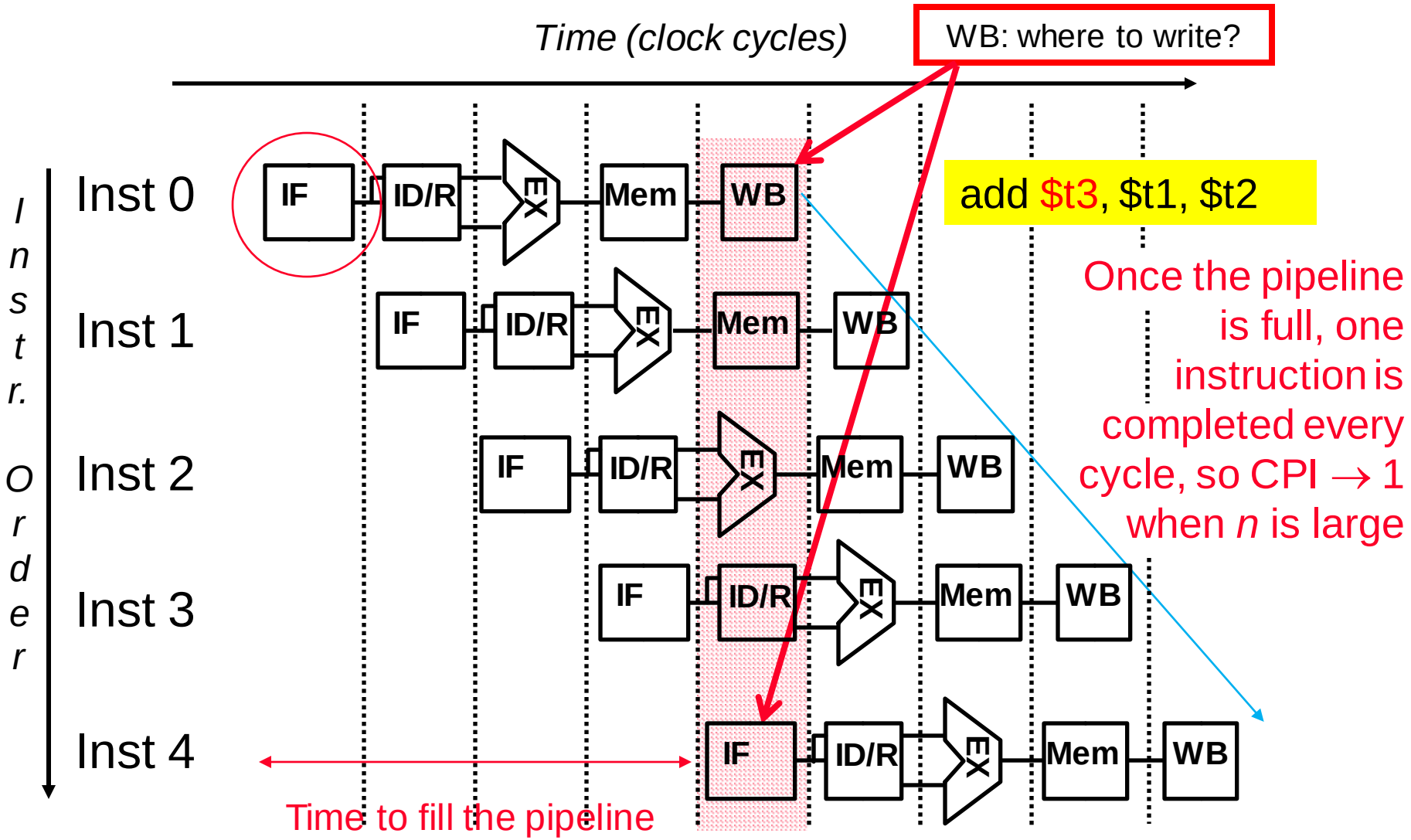
- All instructions must pass through the same number of stages (required plus NOPs)

- Pipelining offers speed-up that is bounded by the number of pipe-stages
  - In the best case, one instruction finishes on every cycle, and the speed-up is equal to the pipeline depth.
  - Each stage needs is own functional units

Pipeline Speed-up

- If all stages are balanced
  - i.e., all take the same time
  - Speed-up ≤ *k* (pipe-depth)
- If not balanced, speed-up is less
- However, life is not that simple; actual speed-up might be further affected due to:
- -- pipeline overhead
- -- pipe hazards

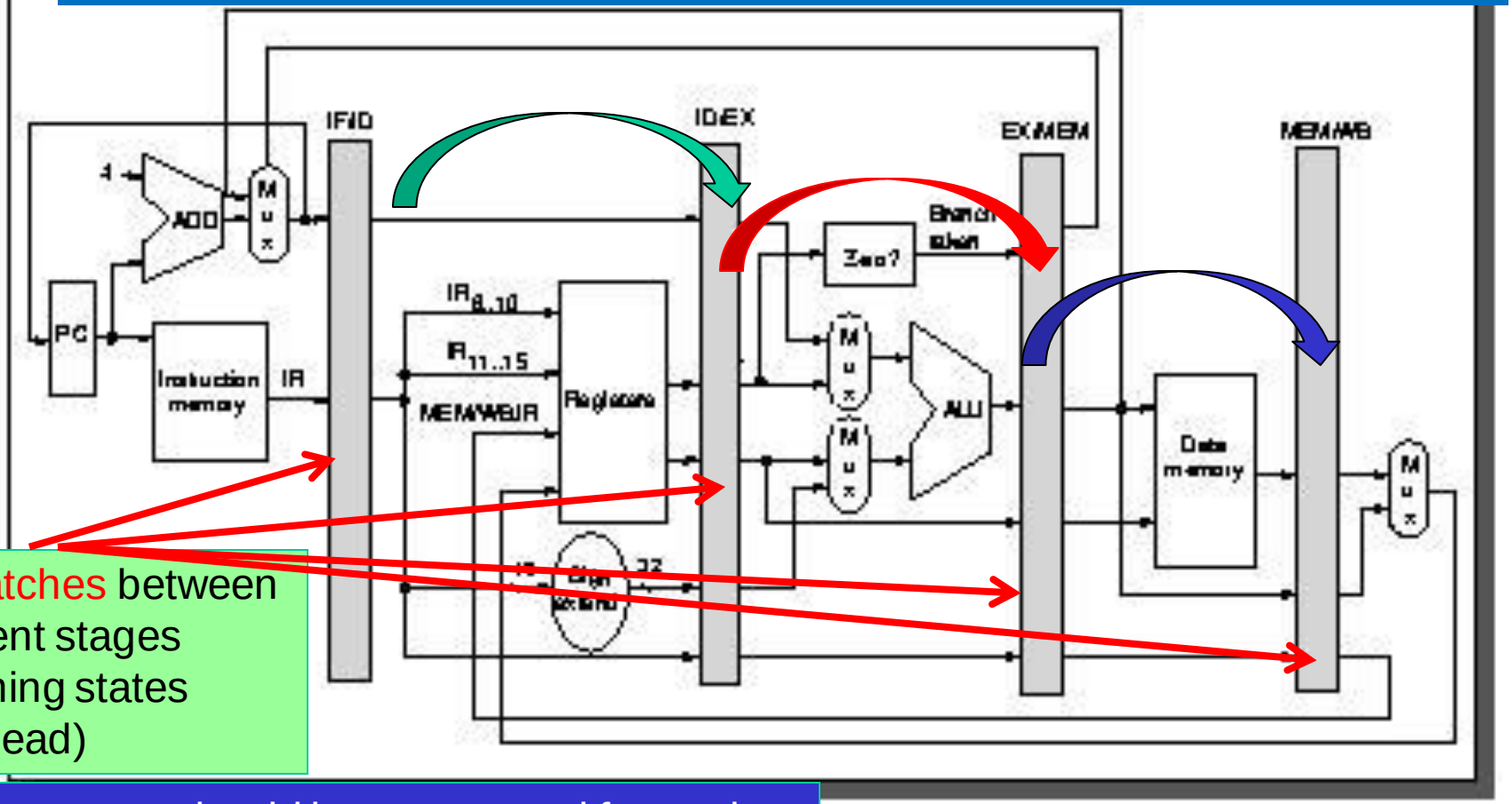
Pipeline issues



Information regarding destination-register-ID for Inst-0 is no longer available, because Inst-4 has already entered into the pipeline

Solution??

IF → ID/RR → EX → Mem → WB



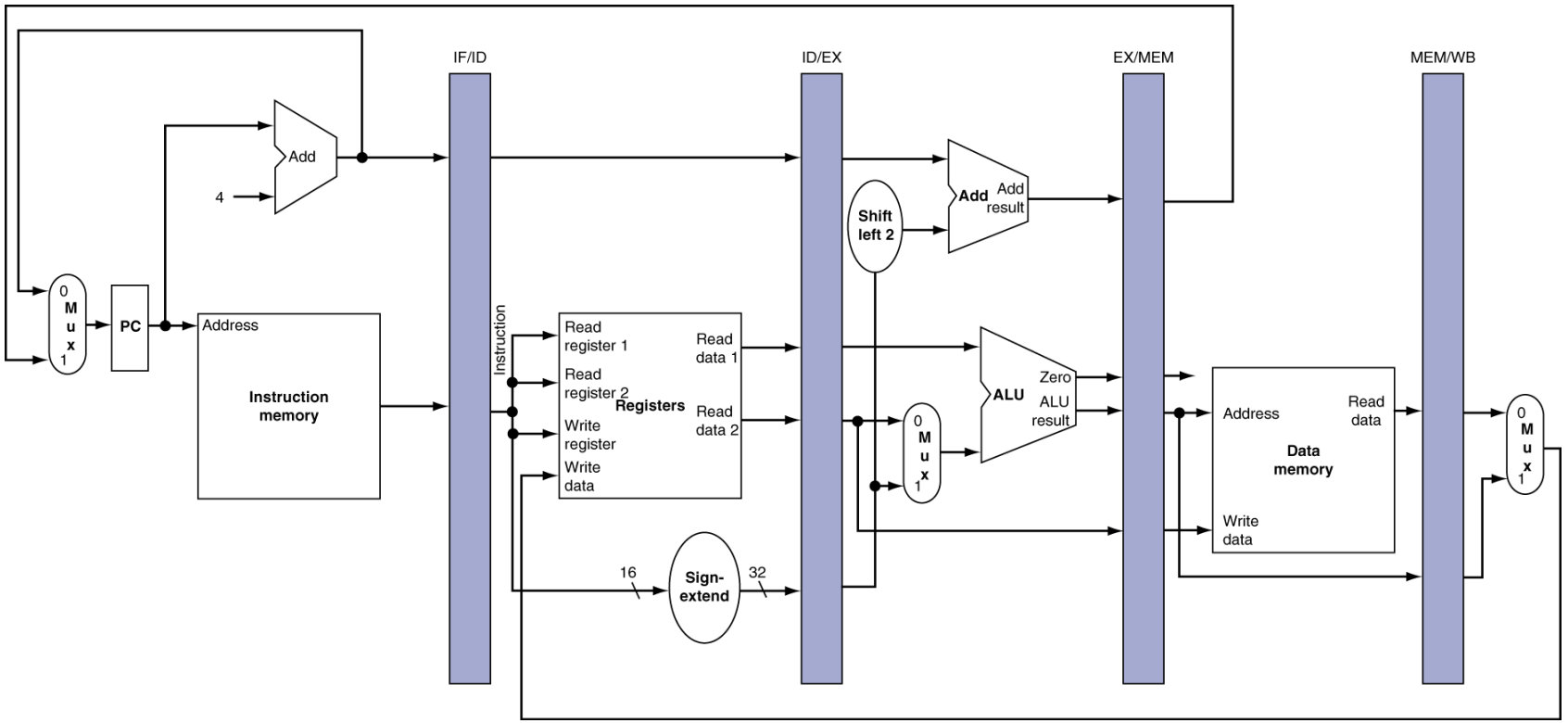
Use latches between adjacent stages pipelining states (overhead)

Latch-contents should be propagated forward as the pipeline advances

Cons: Adds pipeline overhead; slows down the pipeline

Pipeline registers

- Need registers between stages
  - to hold information concerning the previous cycles
  - to pass information forward in every clock cycle



ISA-design impacts pipelining

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step (ID/RR)
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle (assuming ideal cache)

Pipeline Hazards

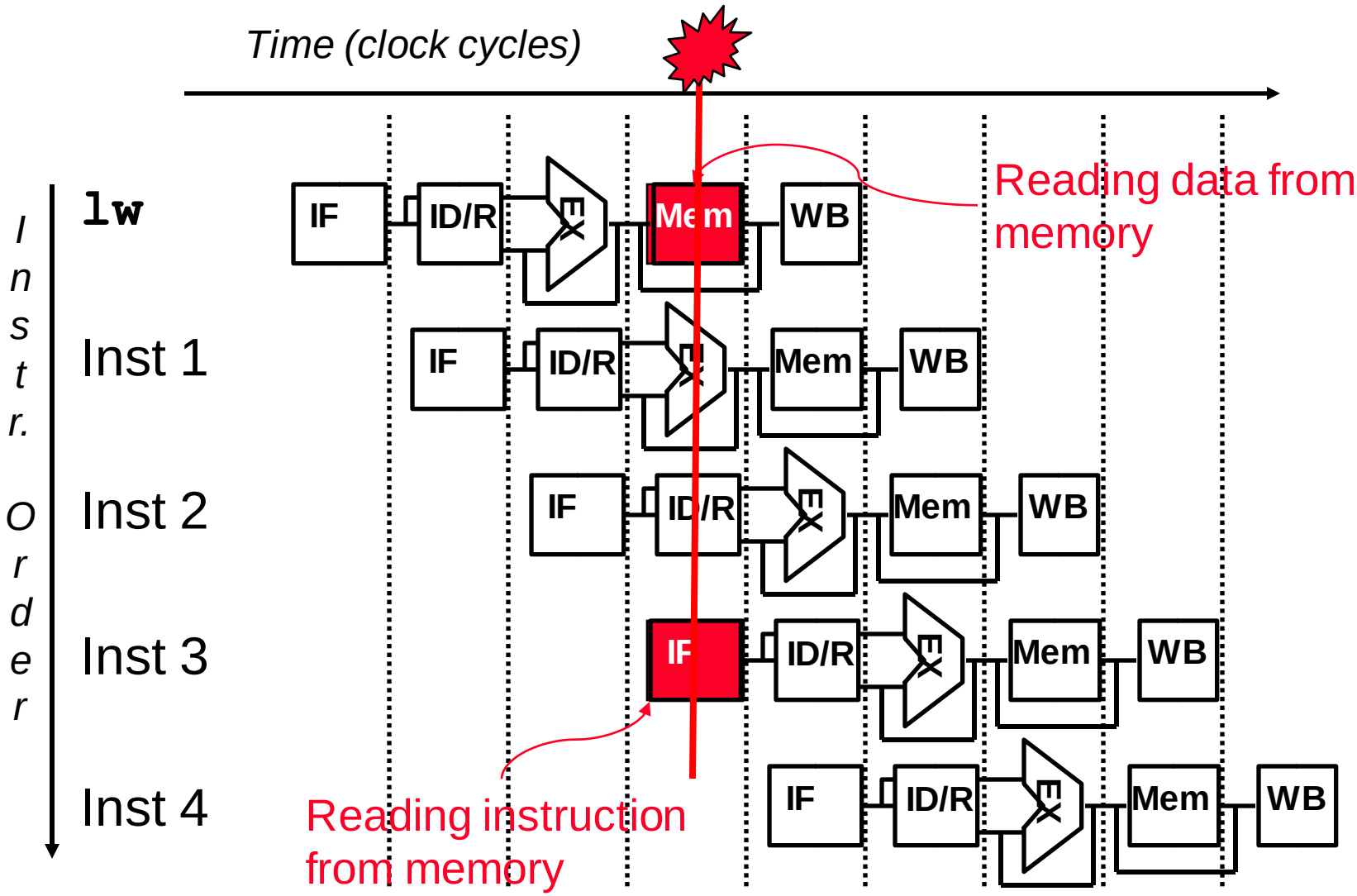
- Situations that prevent starting the next instruction in the next cycle; slows down the pipeline!
- Structural hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

Hazards - Elaborated

- structural hazards: attempt to use the same resource by two different instructions at the same time
- data hazards: attempt to use data before it is ready
  - An instruction's source operand(s) are yet to be produced by a prior instruction, which is still in the pipeline
- control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
  - branch instructions

- Can always resolve hazards by waiting (inserting "bubbles" in the pipeline) – pipeline stall
  - pipeline controller must detect the hazard
  - and take action to resolve hazards
  - Bubbles may be needed to handle cache misses as well (may not be truly termed as hazard)!

Structural Hazard due to Memory Access Conflict



- *Solution*: Use separate instruction and data memory modules (Recall single-cycle case as well)

Structural Hazards

- Conflict for use of a resource
- In MIPS pipeline, with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble” (stalls)
- Hence, pipelined datapaths require separate instruction/data memories
  - Or, separate instruction/data caches

Data Hazards

These occur when at any time, there are active instructions that need to access the same data (memory or register) locations

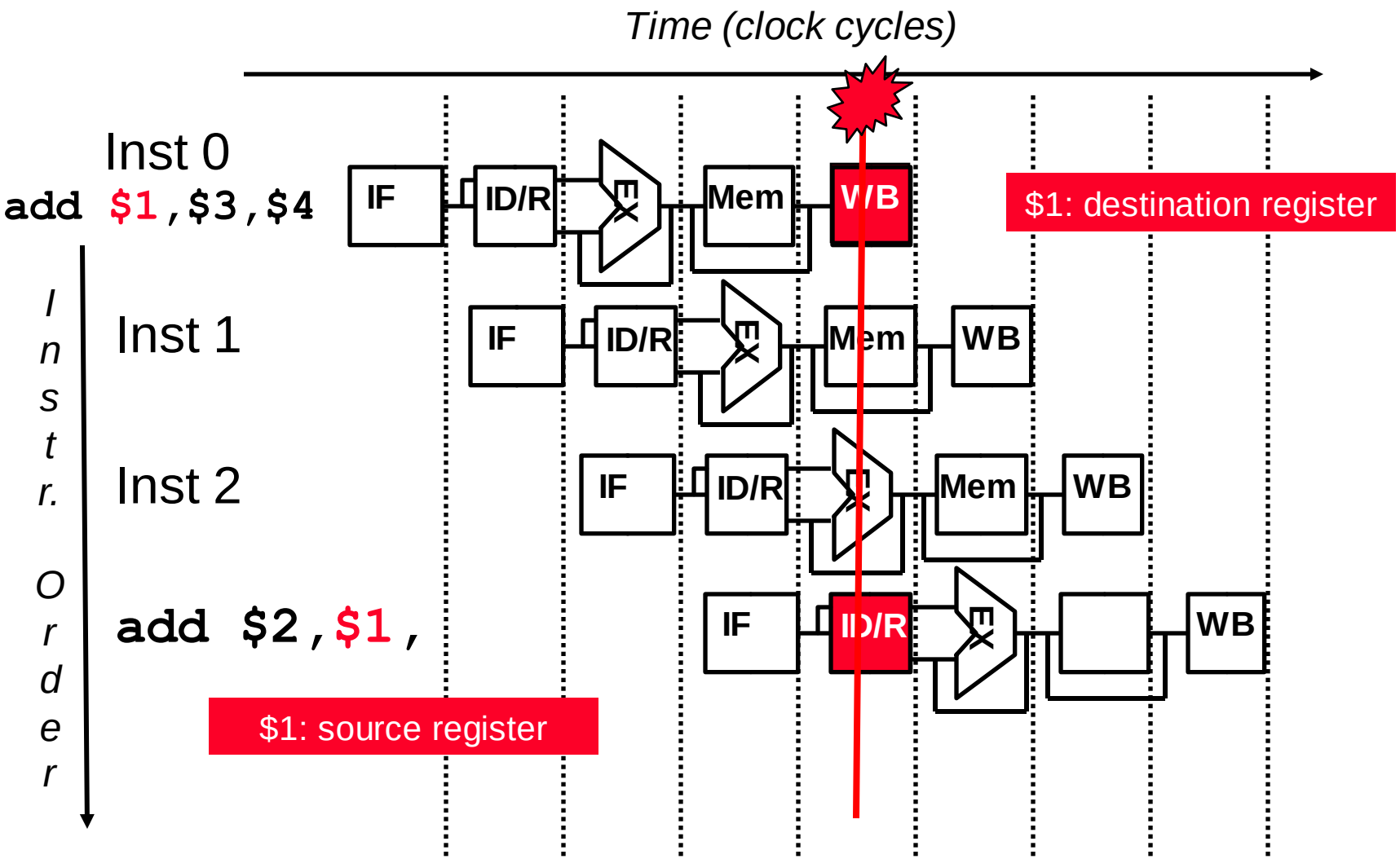
Where there's real trouble is when we have:

instruction A

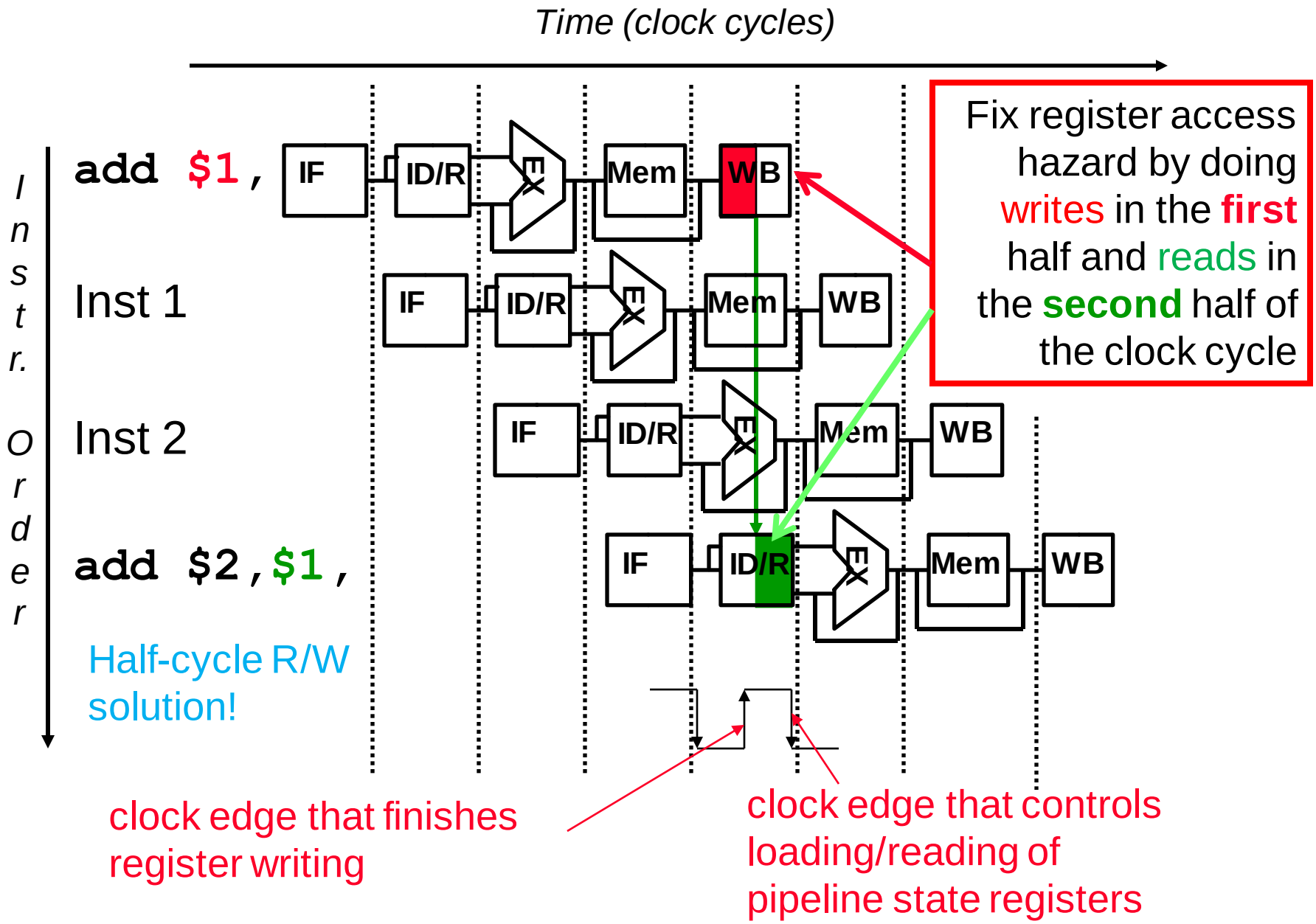
instruction B

and B wants to manipulate (read or write) data before A does. This violates the order of the instructions, since the architecture implies that A completes entirely before B is executed.

Example: Hazards due to Register File Access

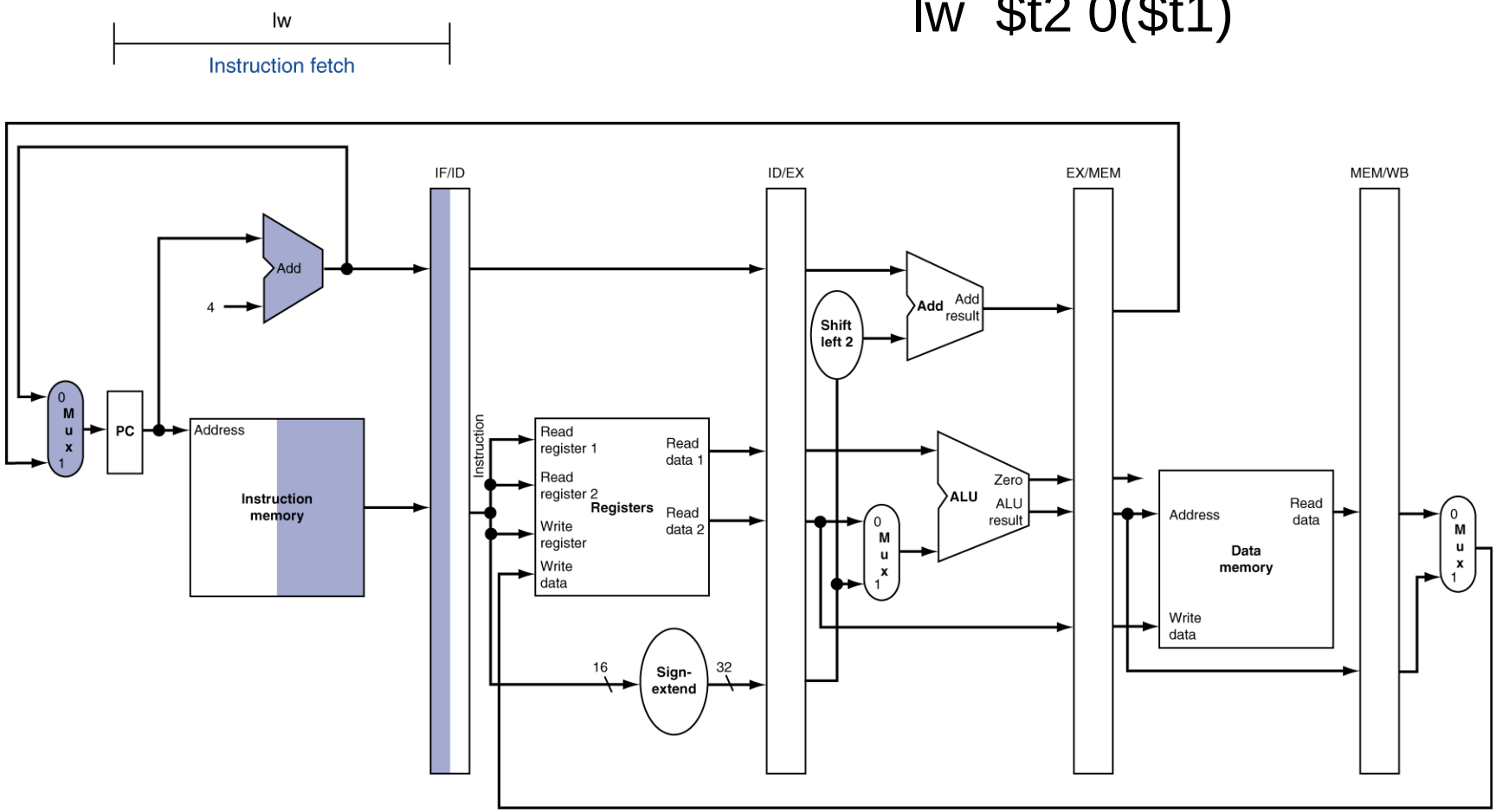


Solution: Register Write-Read Hazard?

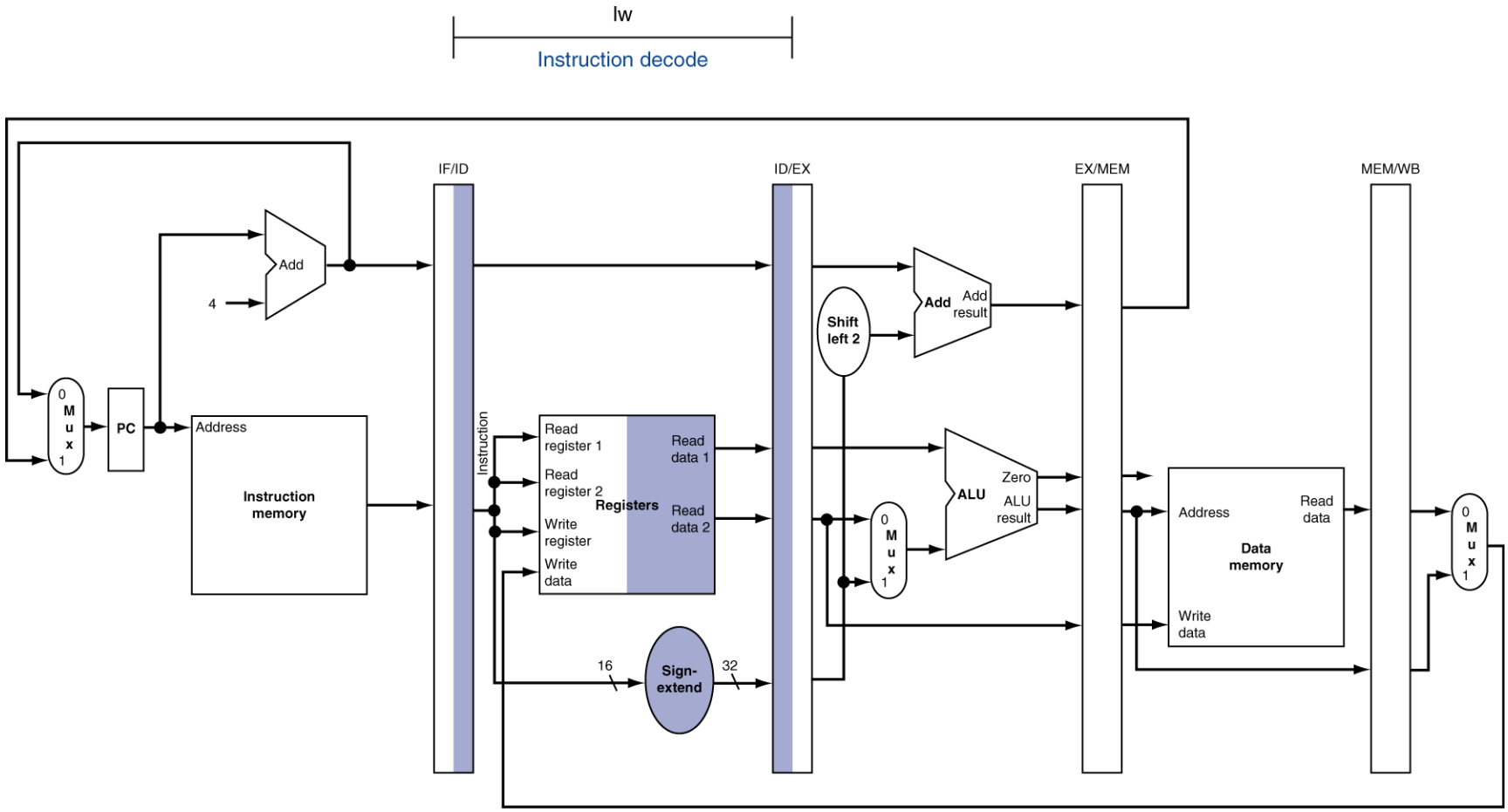


IF for Load, Store, …

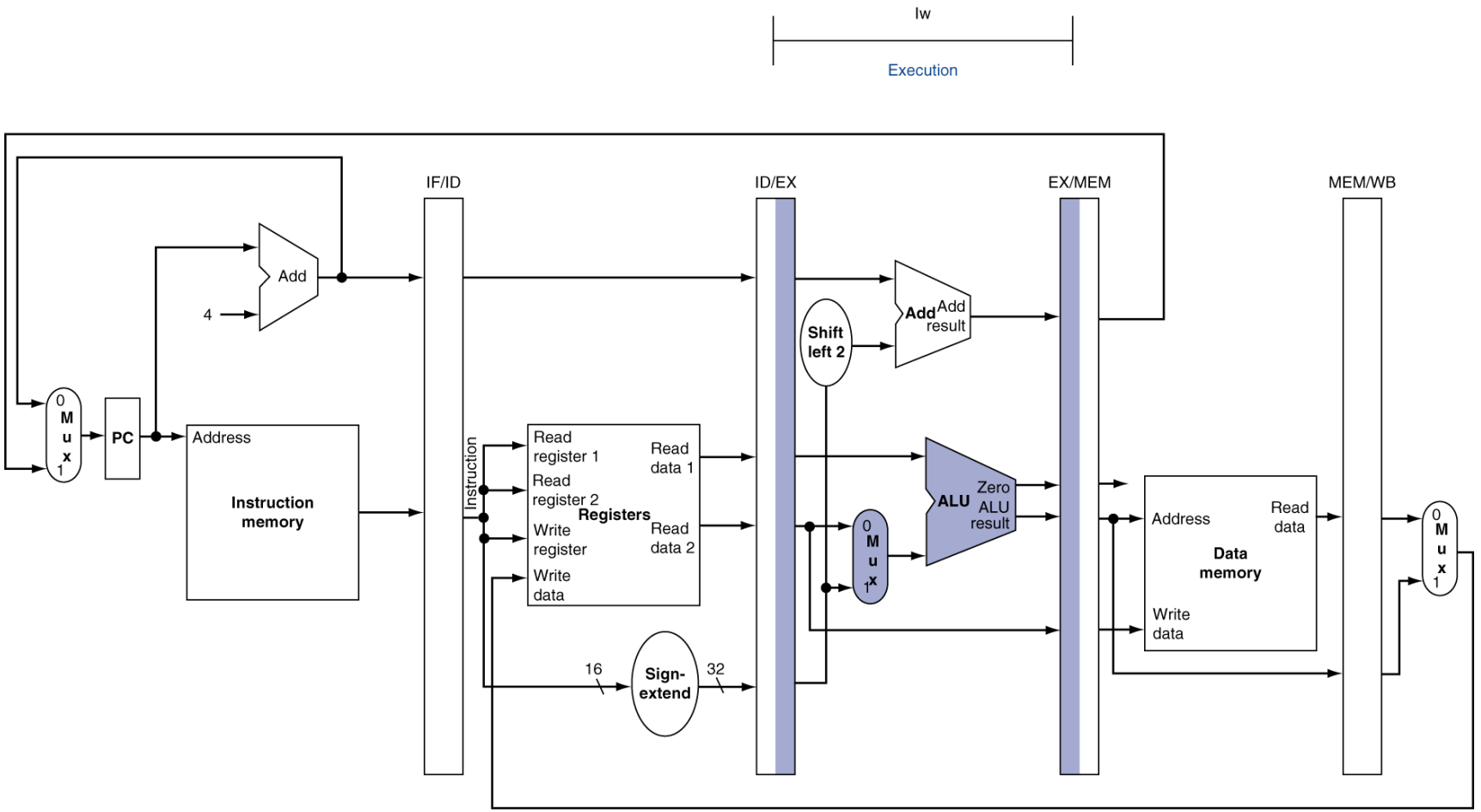
lw $t2 0($t1)



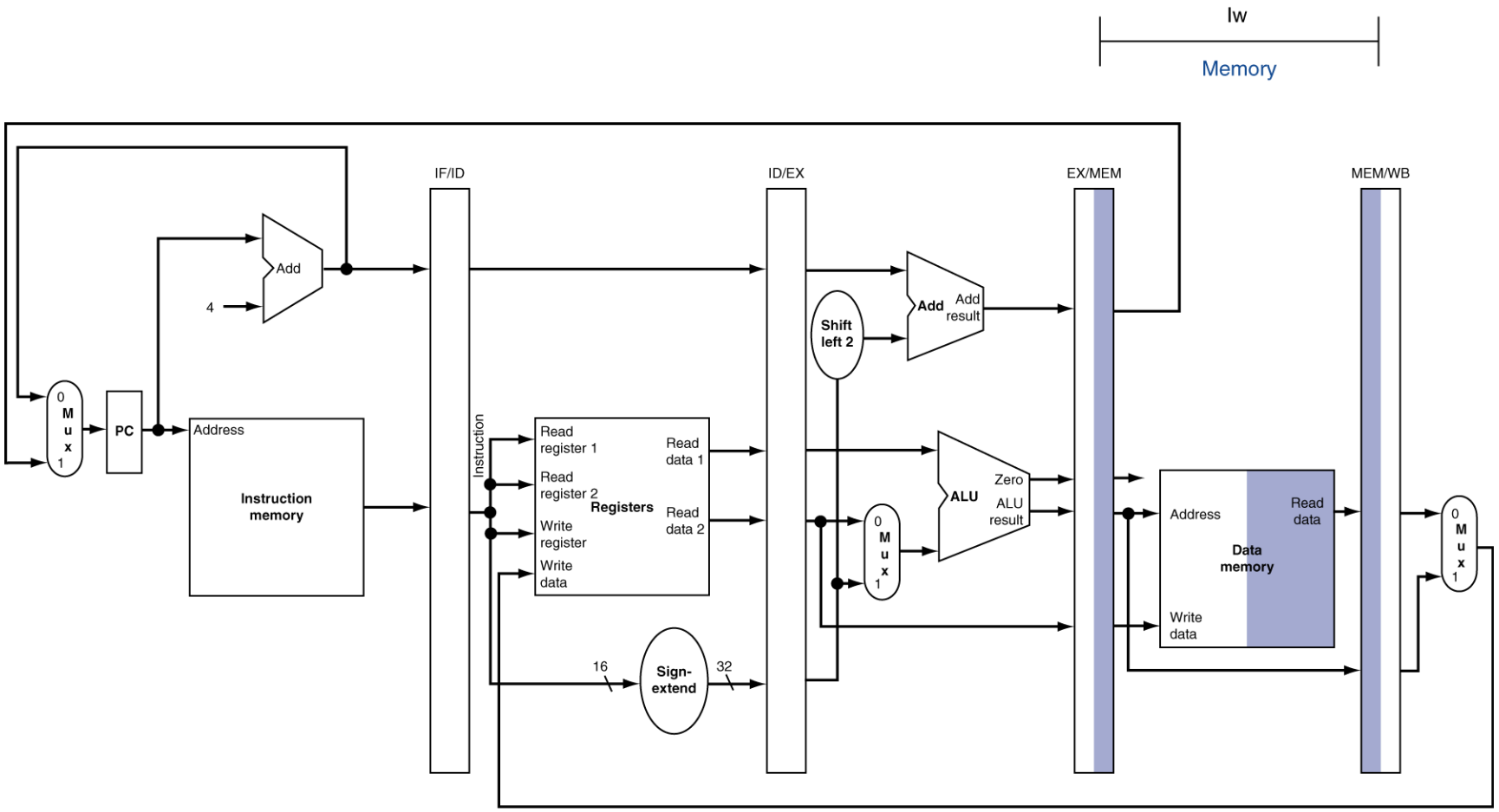
ID for Load, Store, …



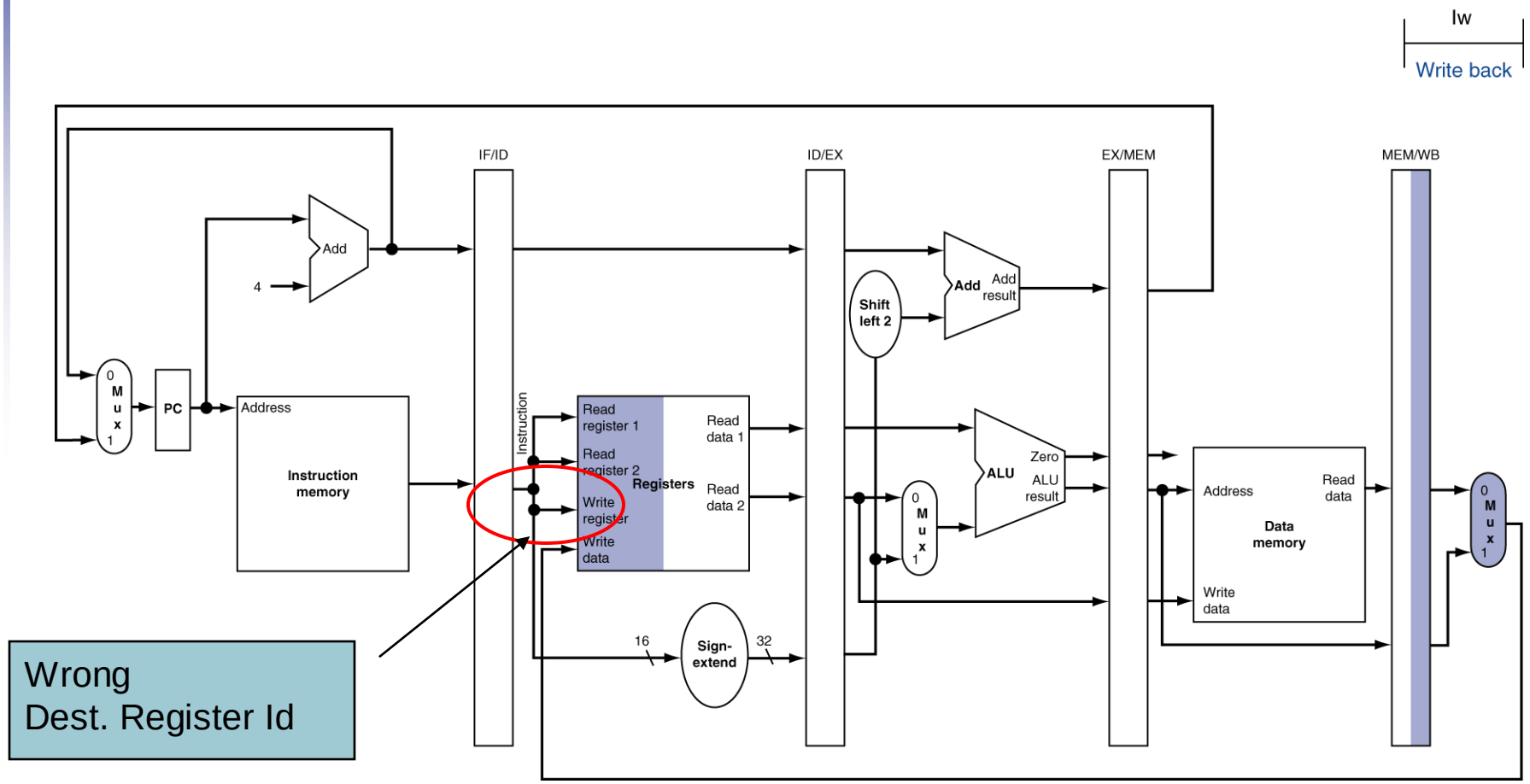
EX for Load


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

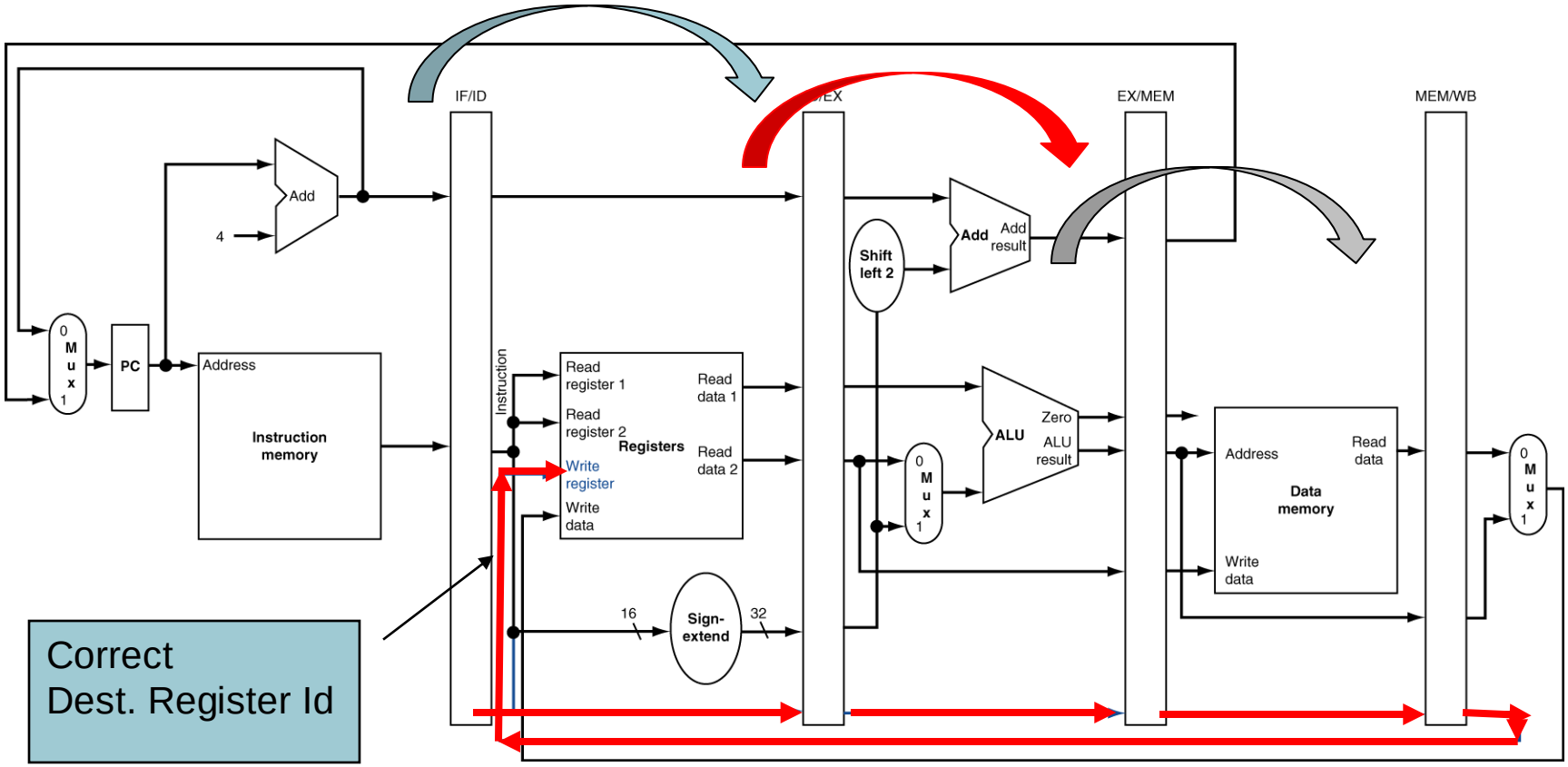
MEM for Load


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
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
ALU
Zero
ALU result
Address
Data memory
Read data
Write data

WB for Load

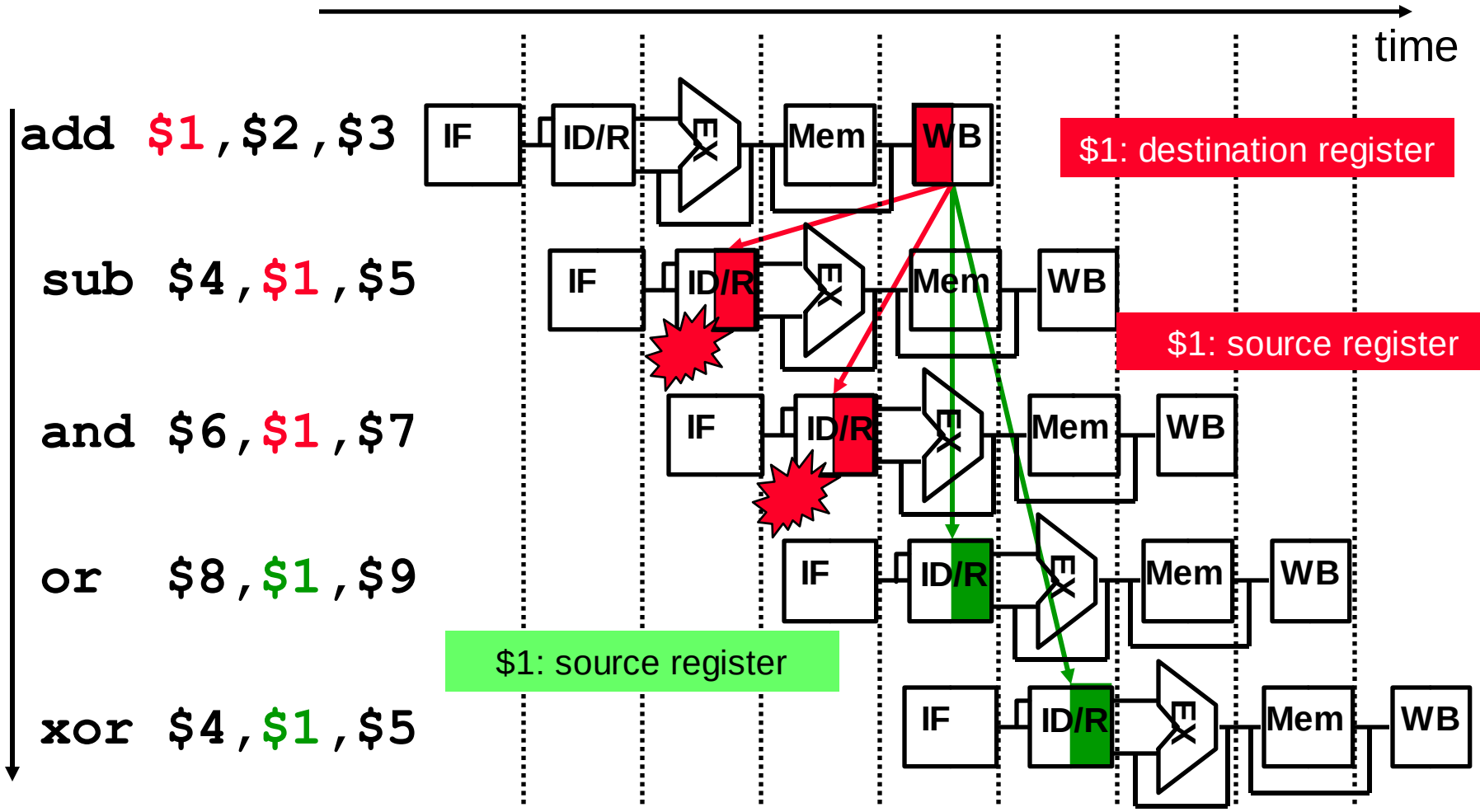

lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Wrong
Dest. Register Id

Corrected Datapath for Load



Use of Same Register Can Cause Data Hazards

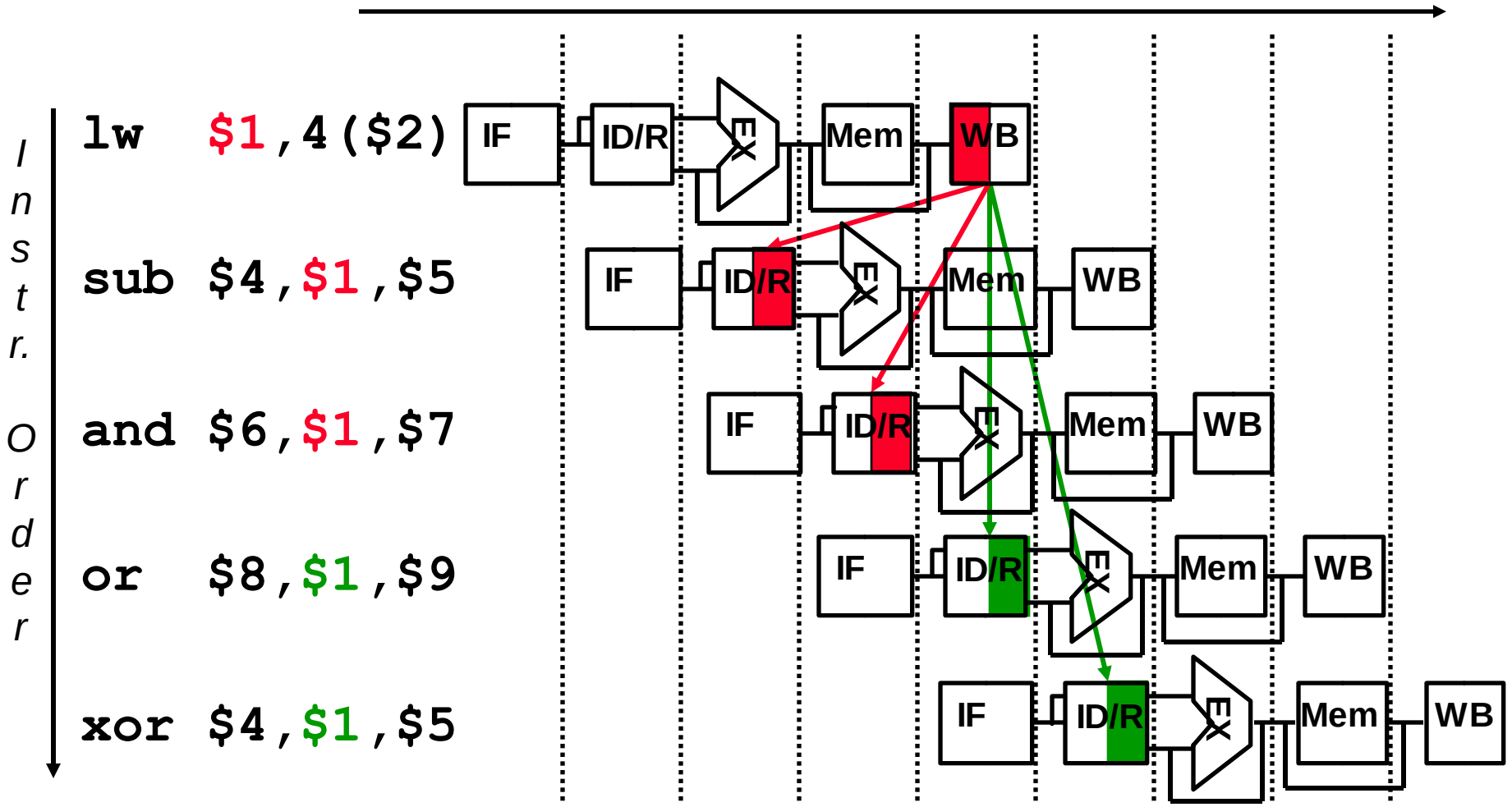
- Dependencies that are *backward* in time, cause hazards



- Read-after-write (RAW) data hazard (trying to read before)

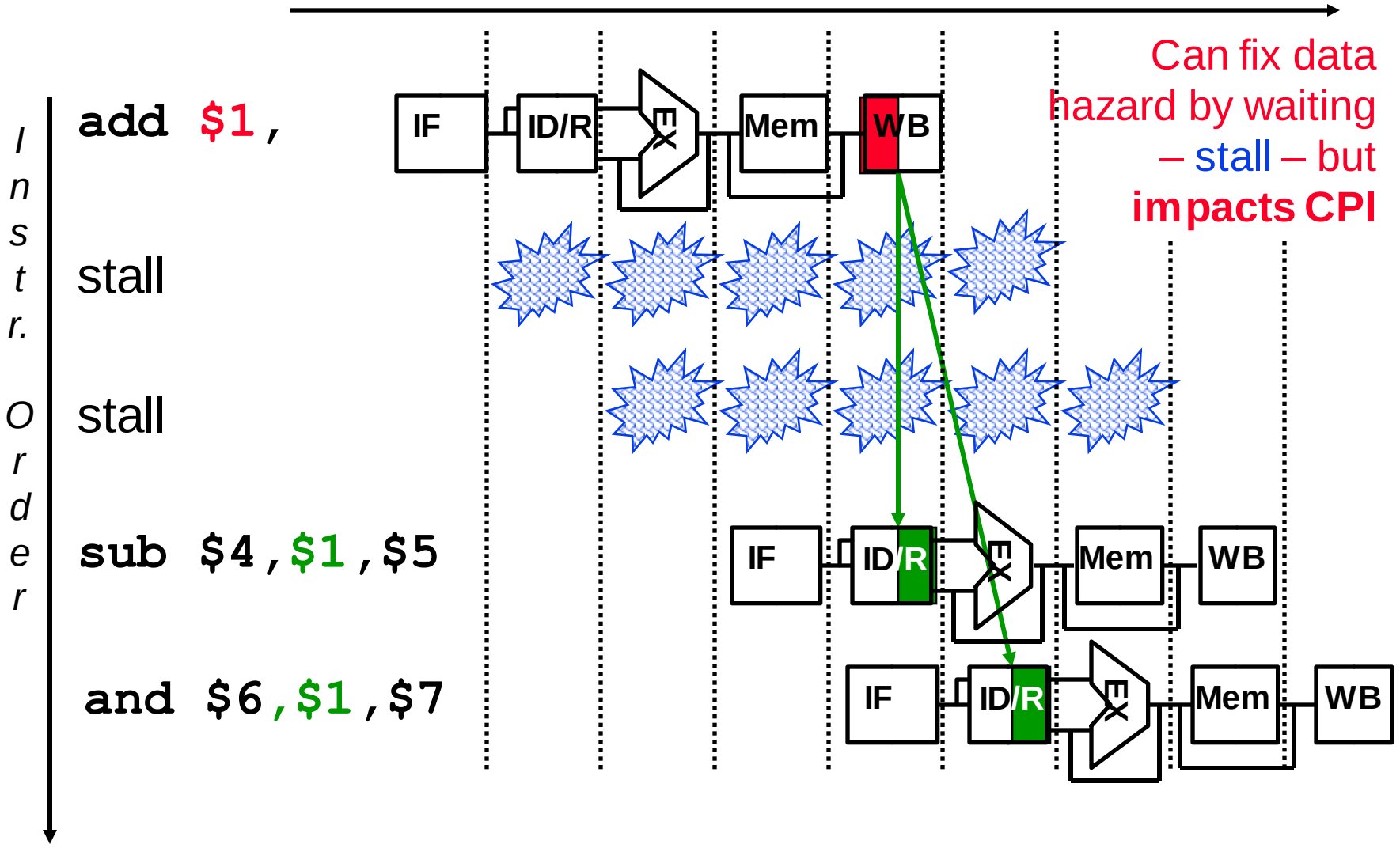
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards



- Load-use data hazard

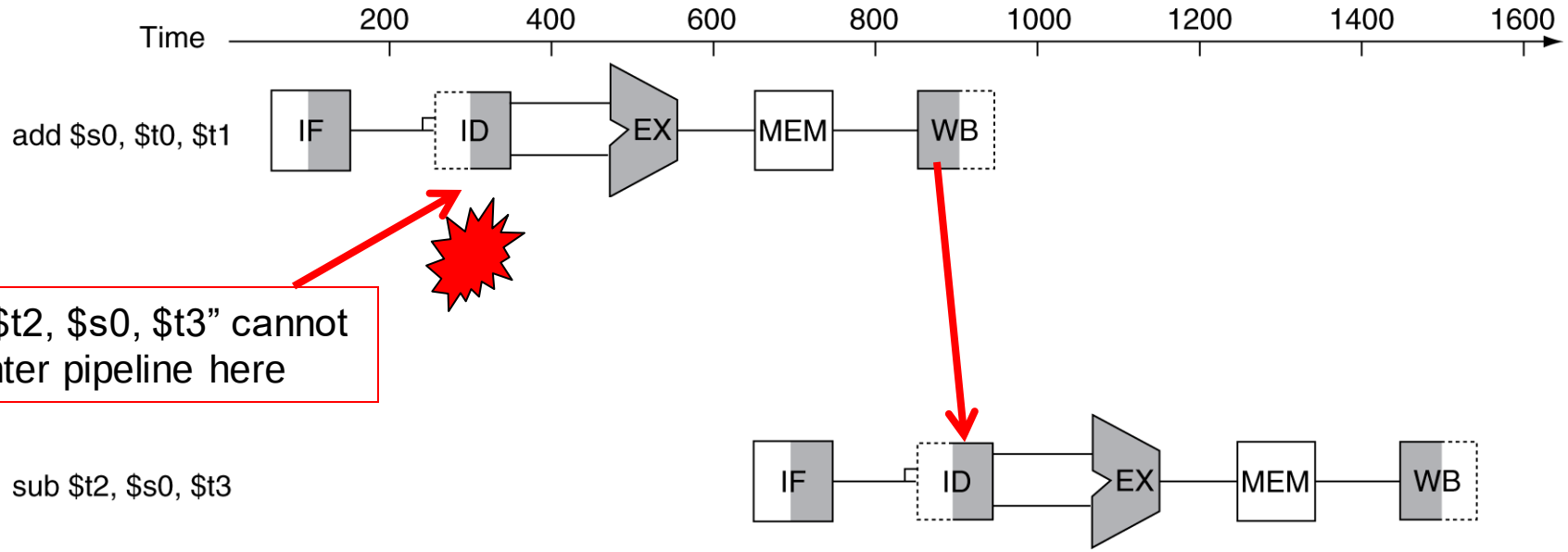
One Way to "Fix" Data Hazard: Use Bubbles



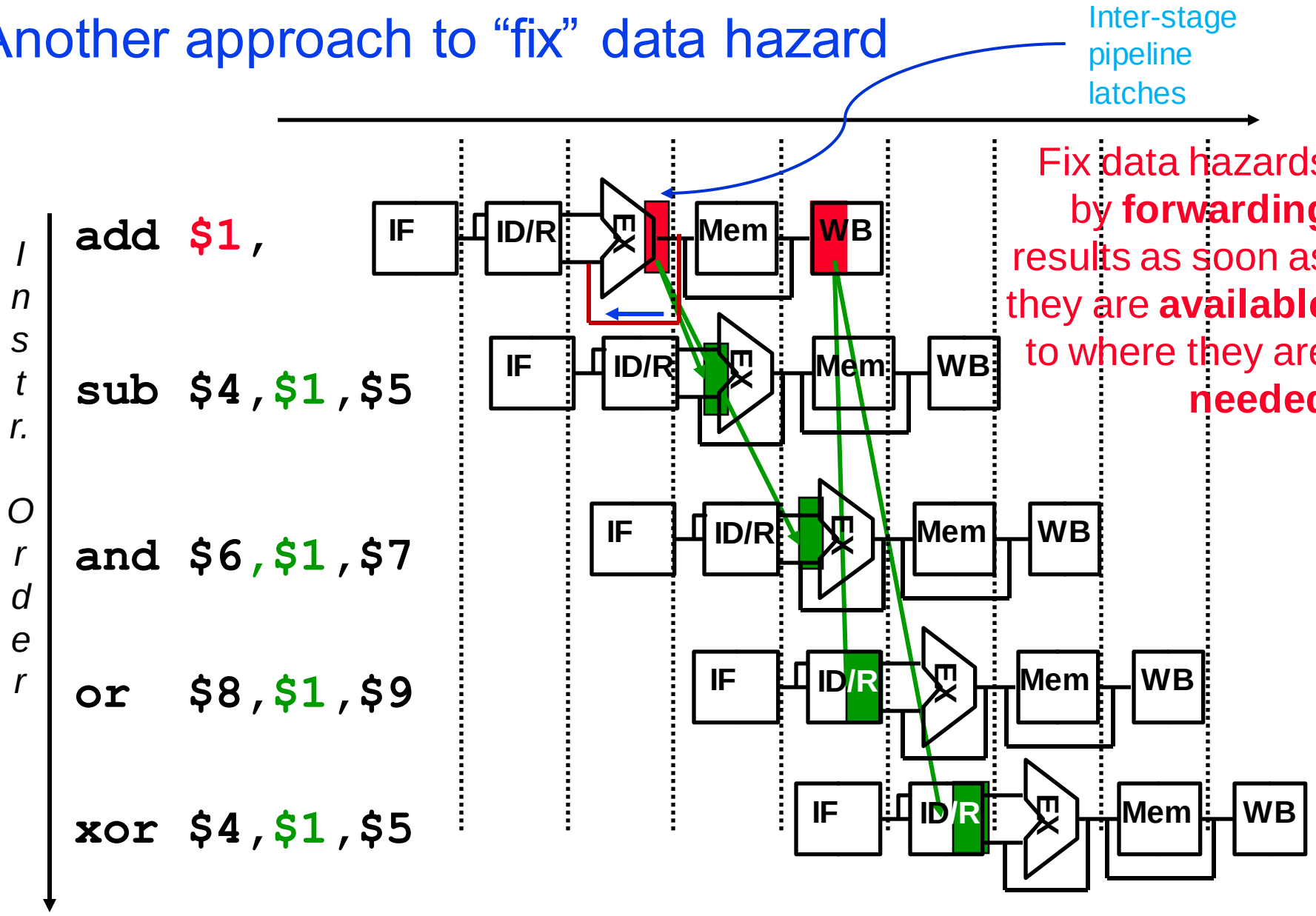
Q. How does the machine decide that two stalls are needed and how it could be implemented?

Data Hazards (contd..)

- An instruction depends on completion of data access by a previous instruction
  - `add $s0, $t0, $t1`
    `sub $t2, $s0, $t3`



Another approach to "fix" data hazard

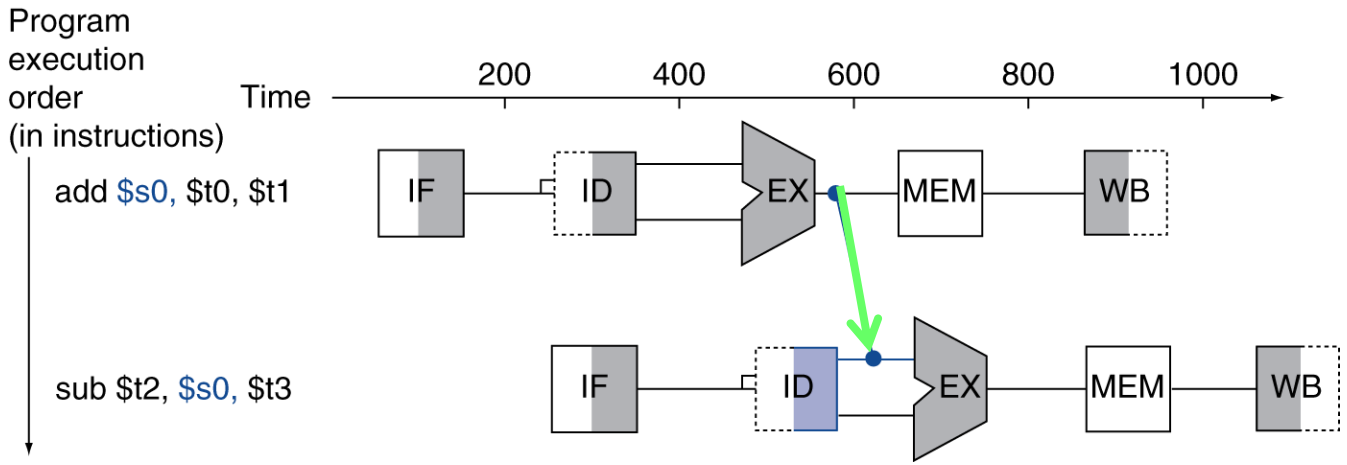


Data forwarding (in time); data-path set backward (in space);

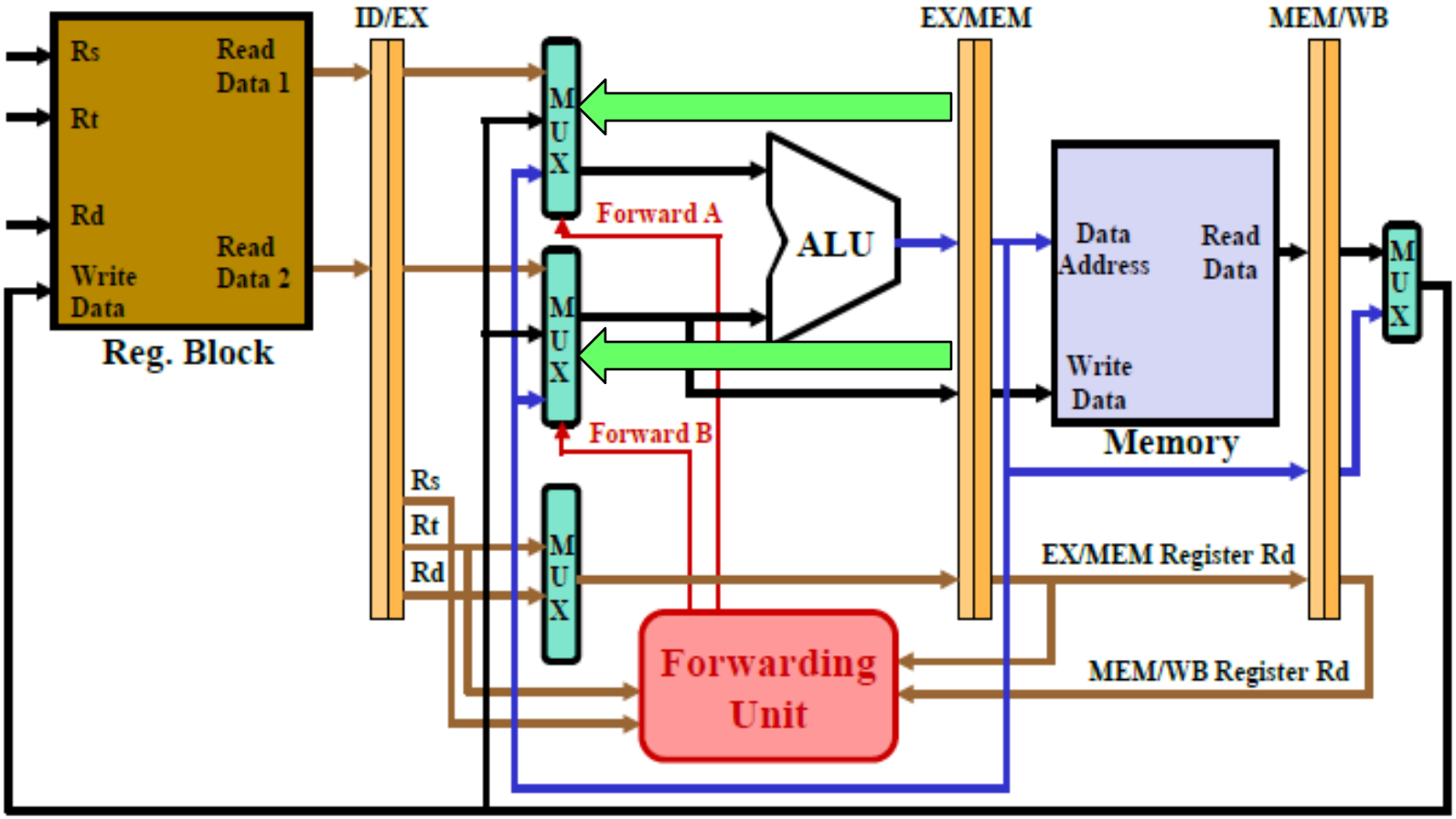
No stalls are needed (saving two)!

Forwarding (*aka* By-passing)

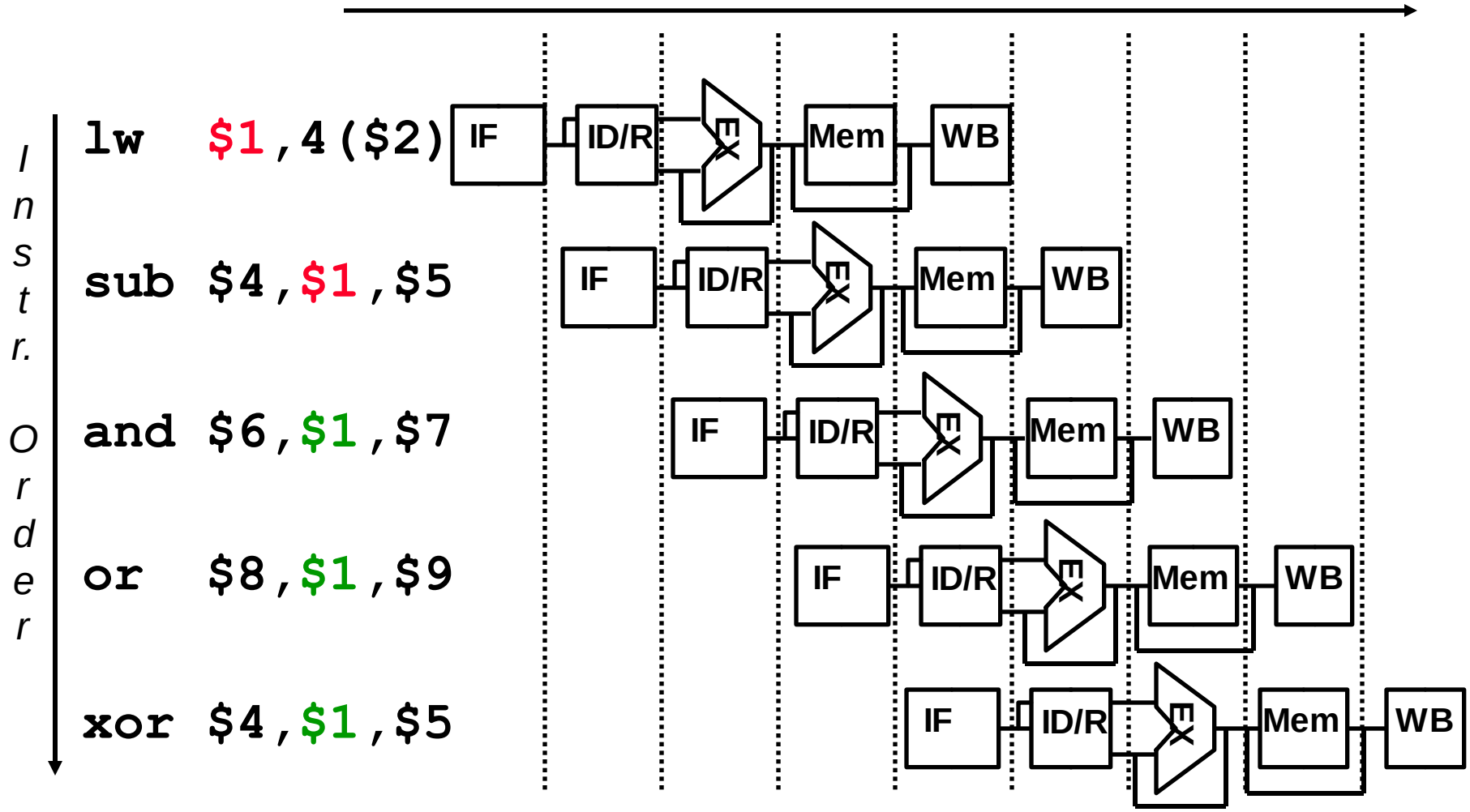
- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath from the output of ALU (EX) to the input of the same ALU!



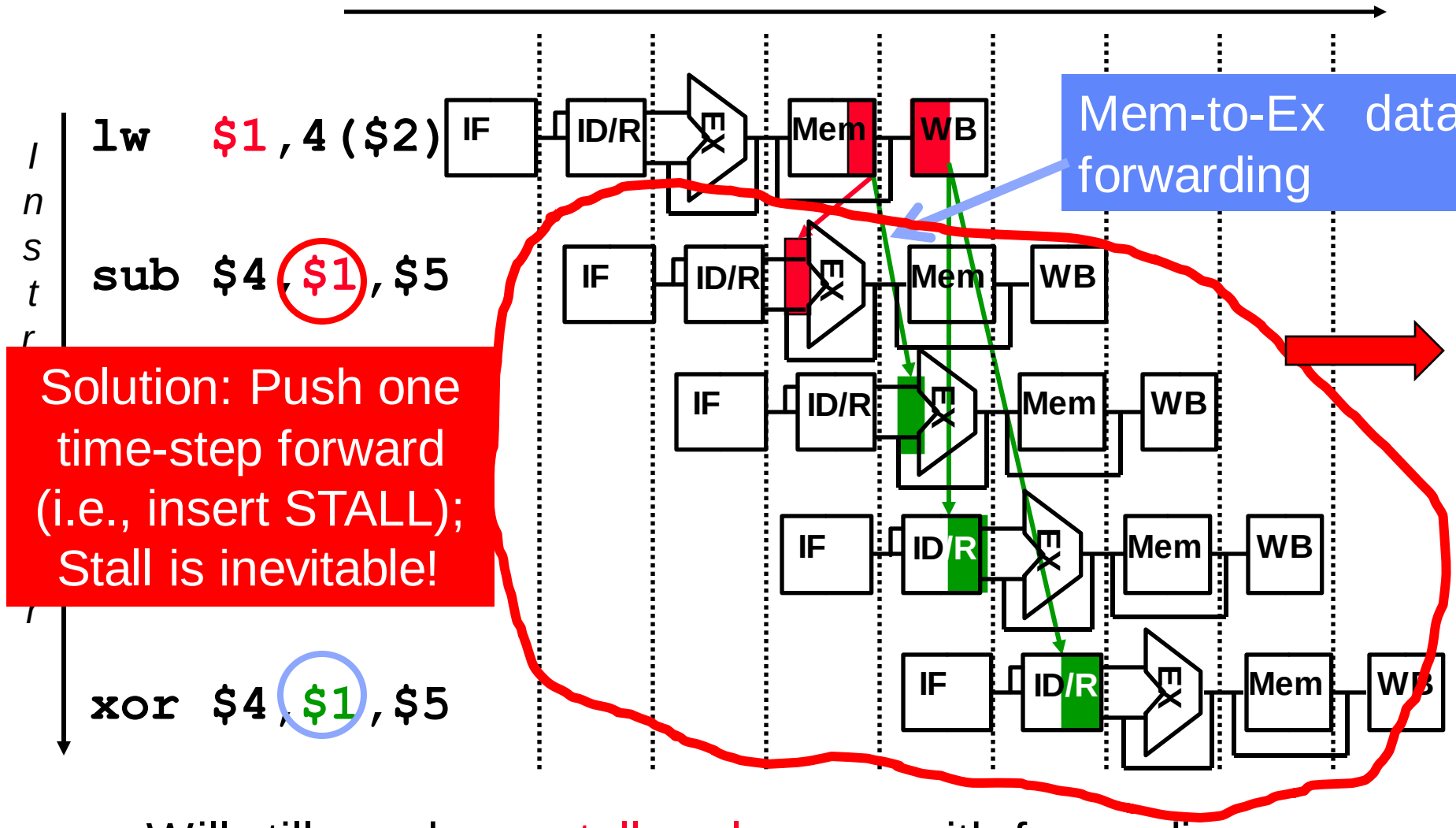
Data Forwarding Needs Extra Hardware (Ex→Ex)



Forwarding with Load-Use Data Hazards



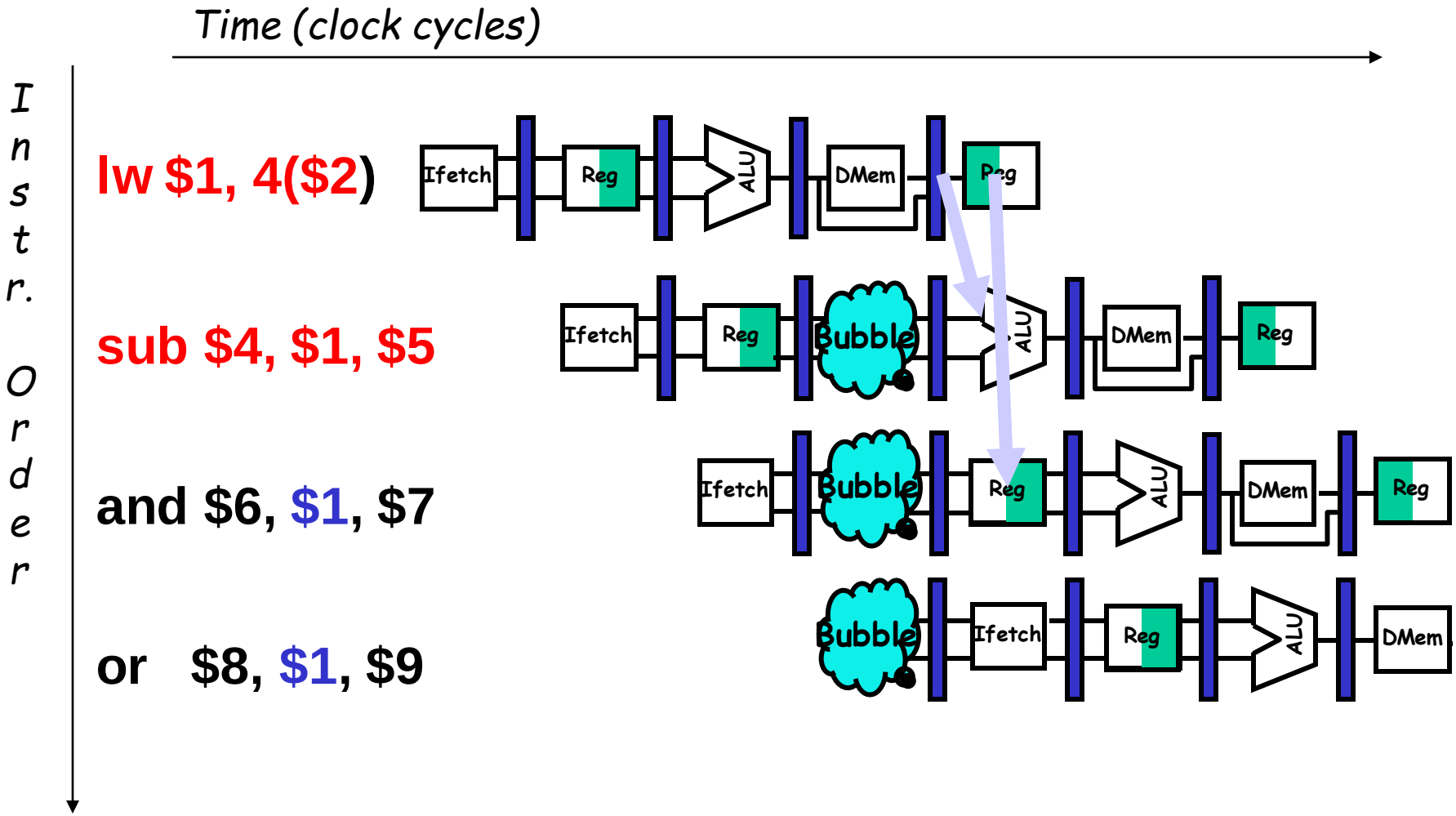
Forwarding with Load-Use Data Hazards



- Will still need one stall cycle even with forwarding
- We cannot forward data “backward in time”!

Data Hazards

The stall is necessary as shown here.



There are some instances where hazards occur, even with forwarding.

Tabular Representation of Pipeline

LW R1, 4(R2)	IF	ID	EX	MEM	WB			
SUB R4, R1, R5		IF	ID	EX	MEM	WB		
AND R6, R1, R7			IF	ID	EX	MEM	WB	
OR R8, R1, R9				IF	ID	EX	MEM	WB

data hazard

LW R1, 4(R2)	IF	ID	EX	MEM	WB			
SUB R4, R1, R5		IF	ID	stall	EX	MEM	WB	
AND R6, R1, R7			IF	stall	ID	EX	MEM	WB
OR R8, R1, R9				stall	IF	ID	EX	MEM	WB

Inserting vertical bubbles is equivalent to inserting horizontal bubbles

Announcement of Class Test 03 (last one)

- Monday, 15 November 2021
- Time: 12:00 noon – 02:00 pm
- Credit: 40%
- Coverage: Processor Design, Memory: Cache, Main Memory, Virtual Memory, TLB, Hard Disks, Exceptions/Interrupts, and Pipelining