18-447 Lecture 7:
Pipelined Implementation

James C. Hoe

Department of ECE

Carnegie Mellon University
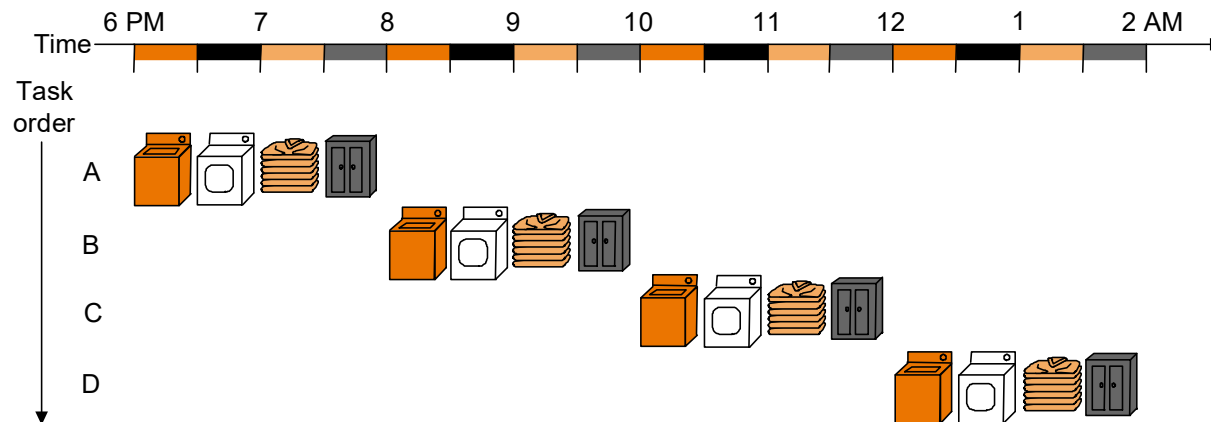
Housekeeping

- Your goal today
  - getting started on pipelined implementations
- Notices
  - Lab 1, Part B, due this week
  - HW 2, due Mon 2/21

 *Look for* Handout #7: Lab 2 *on Friday*

- Readings
  - P&H Ch 4

# Doing laundry: by the book (P&H 4.6)

1. "place one load of <u>dirty clothes</u> in **washer**"
2. "when washer is finished, place <u>washed clothes</u> in **dryer**"
3. "when dryer is finished, **you** fold <u>dried clothes</u>"
4. "when folding is finished, **friend** put <u>away folded</u> clothes"

- steps to do a load are sequentially dependent
- no dependence between different loads
- different steps do not share **resources**

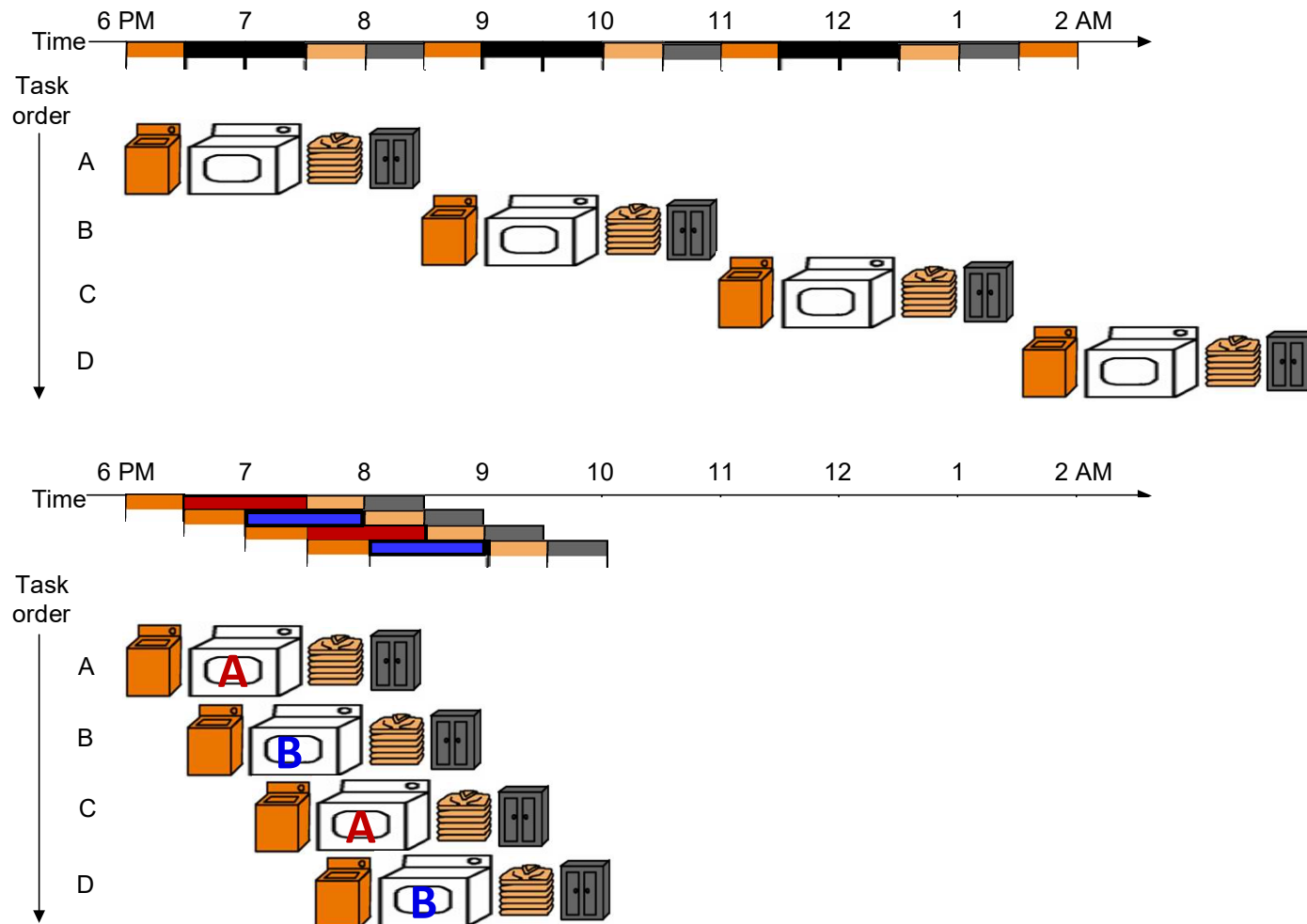# Doing laundry more quickly: in theory
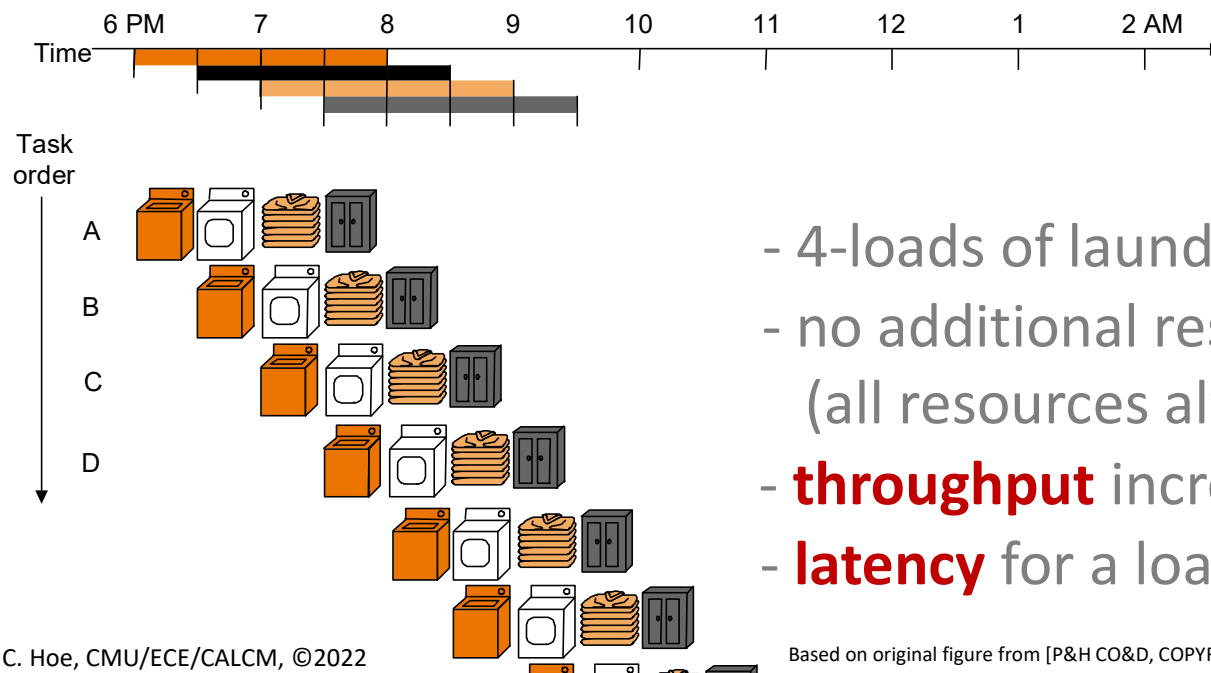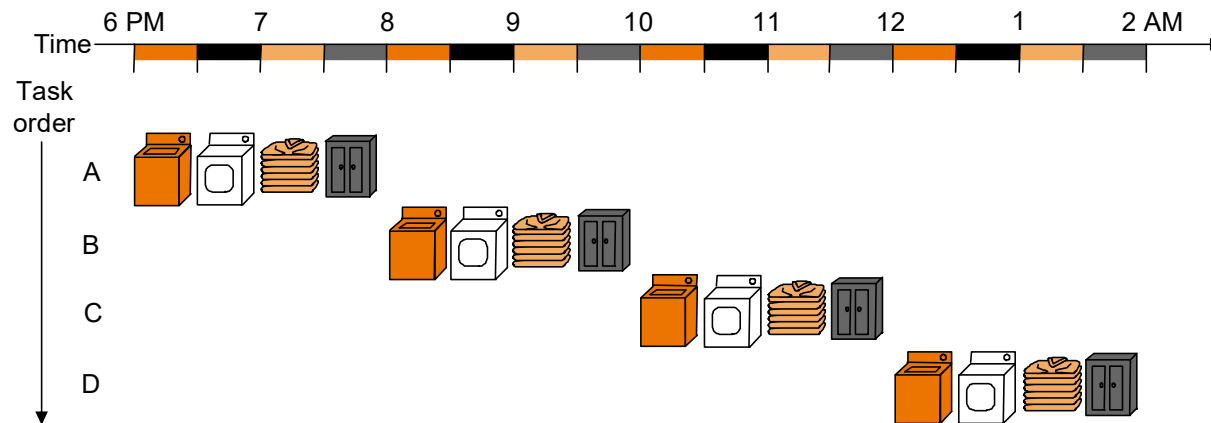
- 4-loads of laundry in parallel
- no additional resources
  (all resources always busy!)
- **throughput** increased by 4
- **latency** for a load is the same

Doing laundry more quickly: in practice
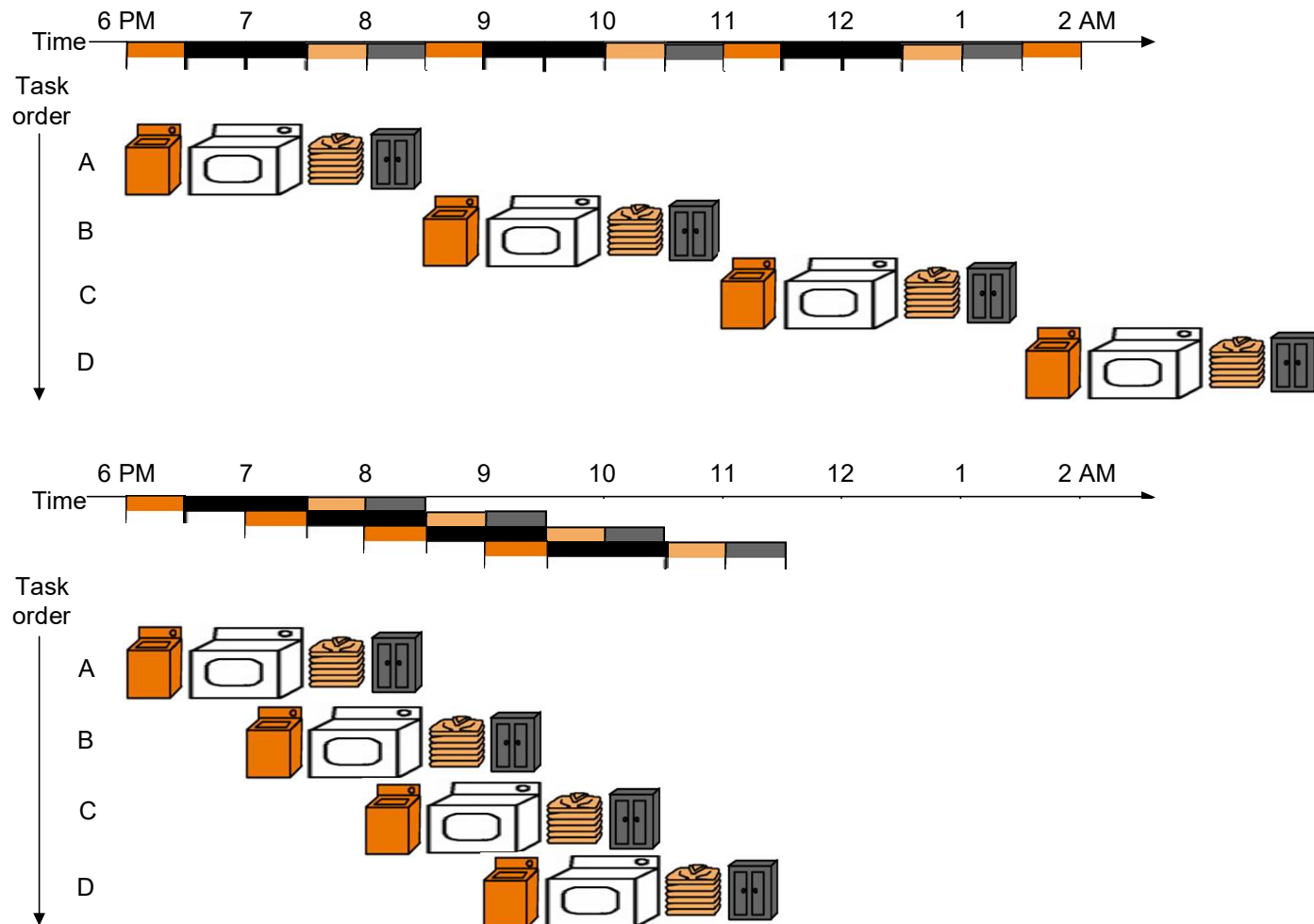
the slowest step decides **throughput**

Doing laundry more quickly: in practice

Throughput restored (2 loads per hour) using 2 dryers

Pipeline Idealism

Motivation: Increase throughput without adding hardware cost

- **Repetition of identical tasks**

  same task repeated for many different inputs

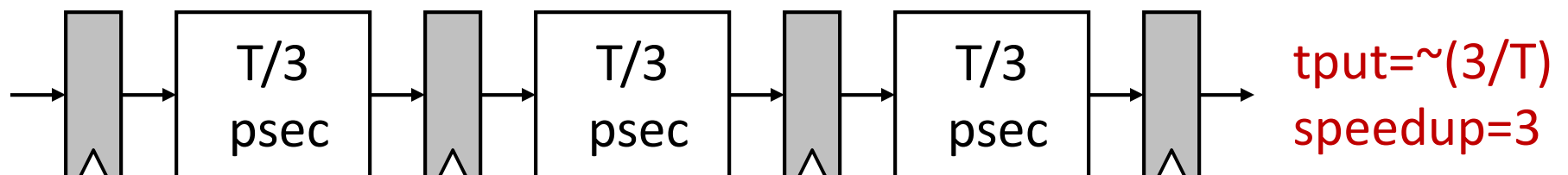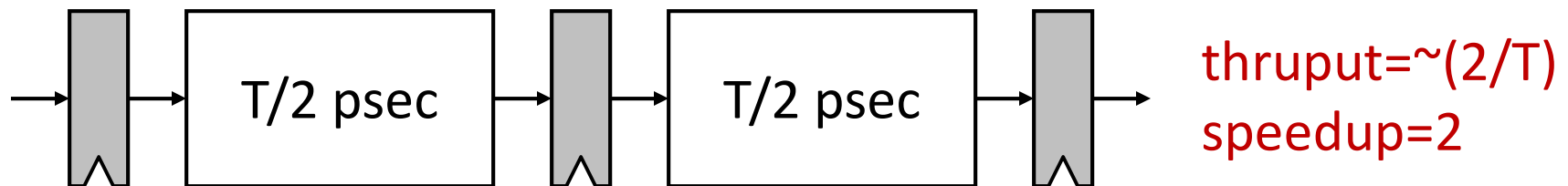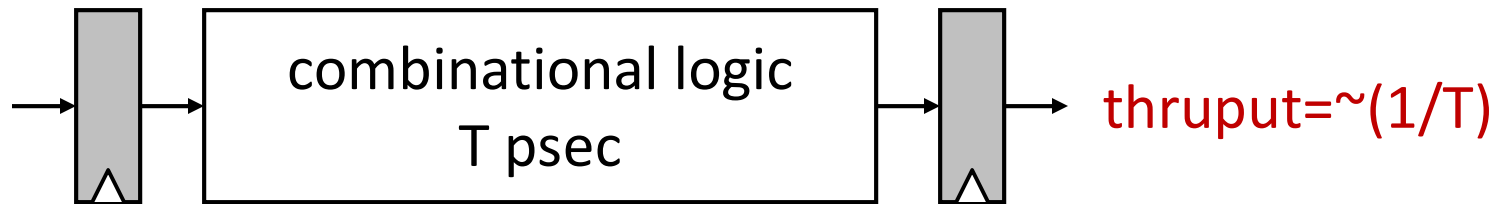- **Repetition of independent tasks**

  no ordering dependencies between repeated tasks

- **Uniformly partitionable suboperations**

  arbitrary number and placement of boundaries

Good examples: automobile assembly line, doing laundry, but instruction execution???
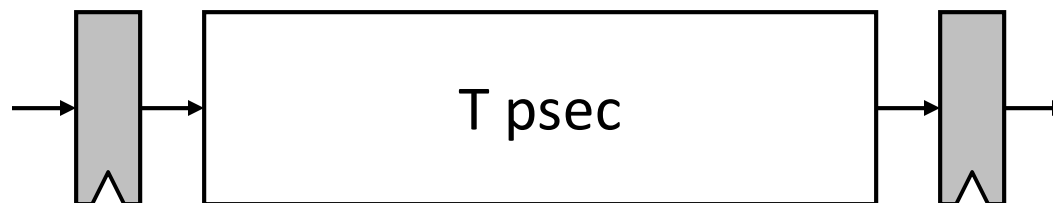
(Ideal) HW Pipelining

thruput=~(1/T)

thruput=~(2/T)
speedup=2

tput=~(3/T)
speedup=3

Notice: evenly divisible; no feedback wires

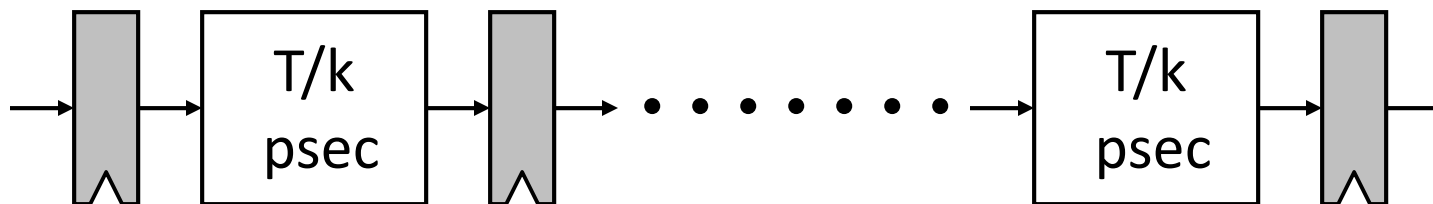# Performance Model

- Nonpipelined version with delay T

  throughput = $1/(T+S)$ where S = latch delay

- k-stage pipelined version

  $\text{throughput}_{k\text{-stage}} = 1 / (T/k + S)$

  $\text{throughput}_{max} = 1 / (1 \text{ gate delay} + S)$

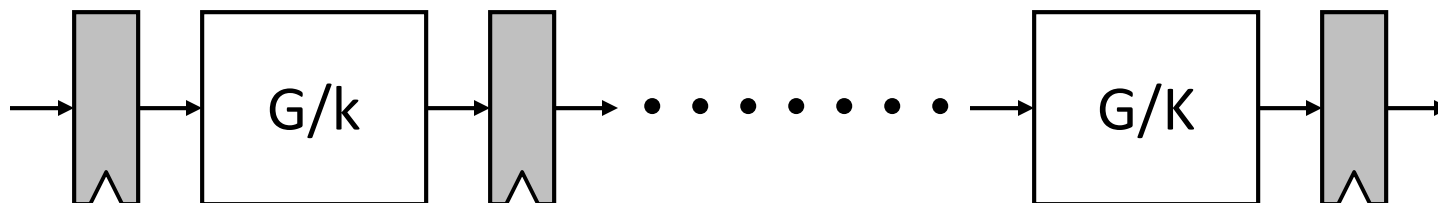

per-task latency became longer: $T+kS$

Cost Model

- Nonpipelined version with combinational cost G

  Cost = G+L where L = latch cost

  G gates

- k-stage pipelined version

  $\text{Cost}_{k\text{-stage}} = G + Lk$

  G/k · · · · · · · G/K

Cost/Performance Trade-off
[Peter M. Kogge, 1981]
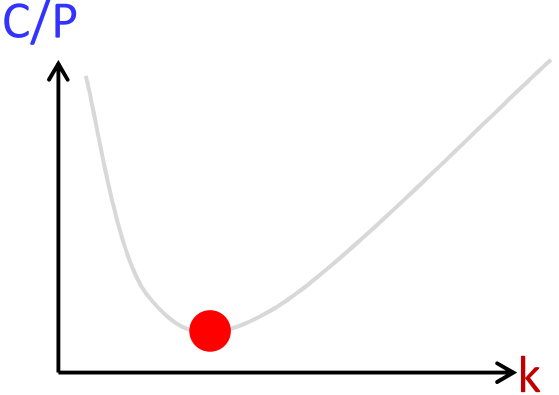
Cost/Performance:

$$C/P = [Lk + G] / [1/(T/k + S)] = (Lk + G)(T/k + S)$$

$$= LT + GS + LSk + GT/k$$
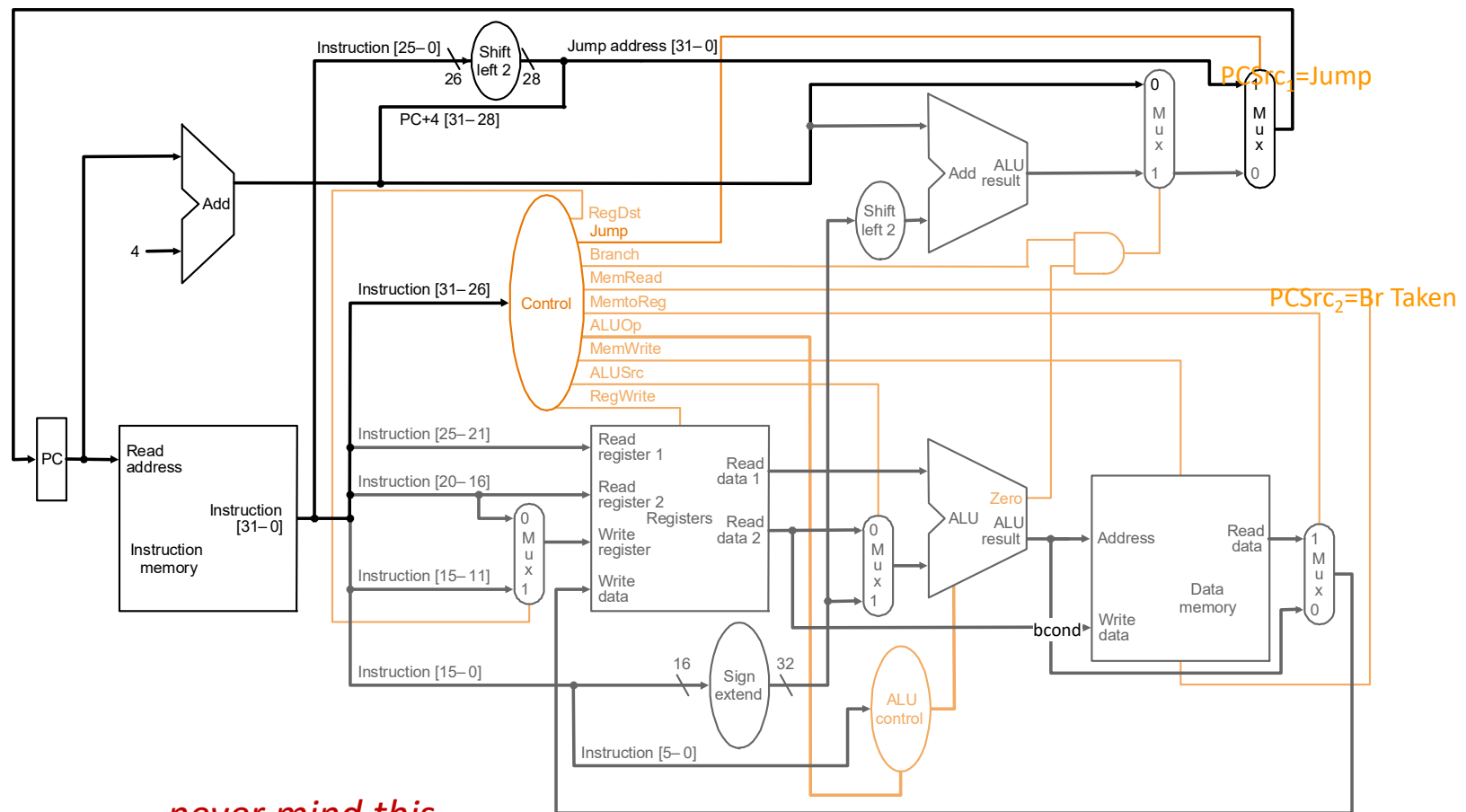
Optimal Cost/Performance: find min. C/P w.r.t. choice of k

$$\frac{d}{dk}\left(\frac{Lk + G}{\frac{1}{\frac{T}{k} + S}}\right) = 0 + 0 + LS - \frac{GT}{k^2}$$

$$LS - \frac{GT}{k^2} = 0$$

$$k_{opt} = \sqrt{\frac{GT}{LS}}$$

C/P

k

Reality of Instruction Pipelining . . . .

never mind this complication today

$$tput = \sim(1/T)$$

18-447-S22-L07-S12, James C. Hoe, CMU/ECE/CALCM, ©2022
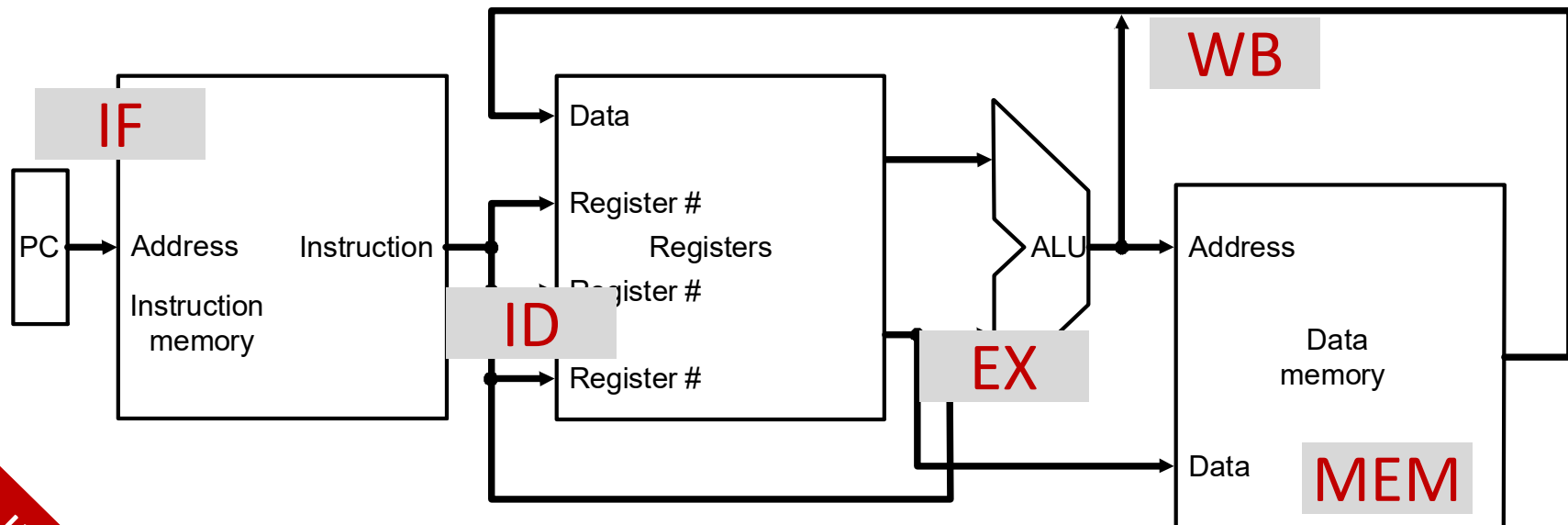
RISC Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access
  - write-back

WB

IF

PC | Address | Instruction | Data

Register #

Registers

Register #

Register #

ID

EX

ALU

Instruction memory

Address

Data memory

Data

MEM

Recall

Coalescing and "External Fragmentation"

steps	IF	ID	EX	MEM	WB
R-type	√	√	√		√
I-type	√	√	√		√
LW	√	√	√	√	√
SW	√	√	√	√	
Bxx/JALR	√	√	√		--/√
JAL	√		√		√

Dividing into Stages

200ps	100ps	200ps	200ps	100ps
IF: Instruction fetch	ID: Instruction decode/ register file read	EX: Execute/ address calculation	MEM: Memory access	WB: Write back

ignore for today
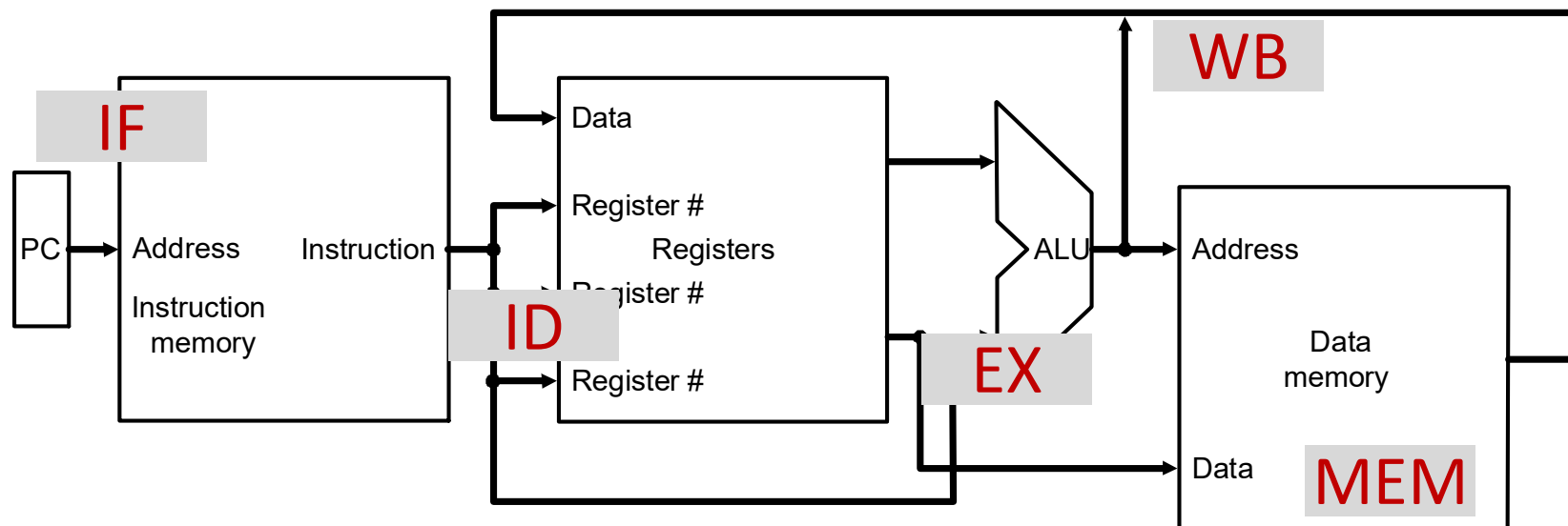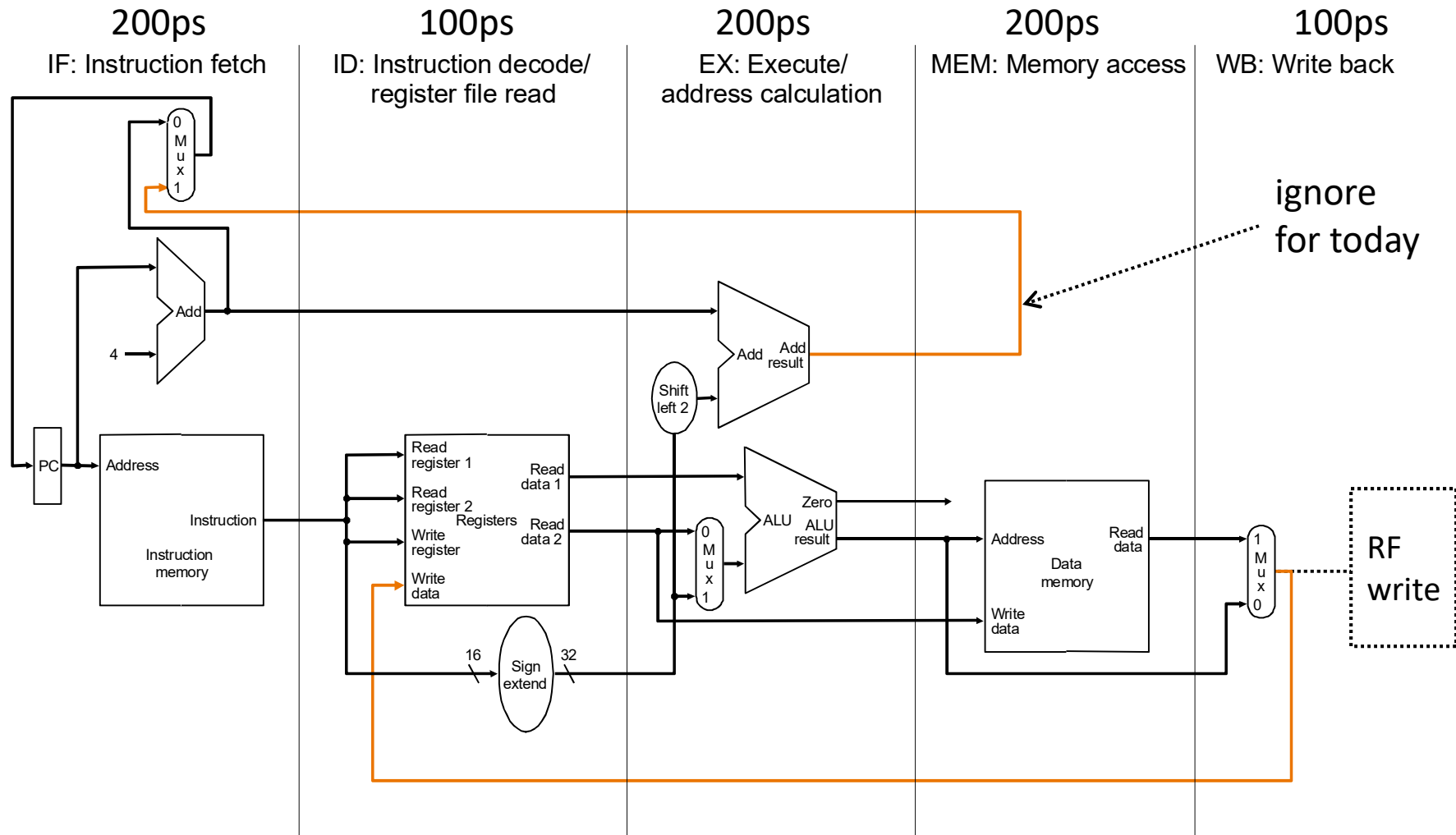
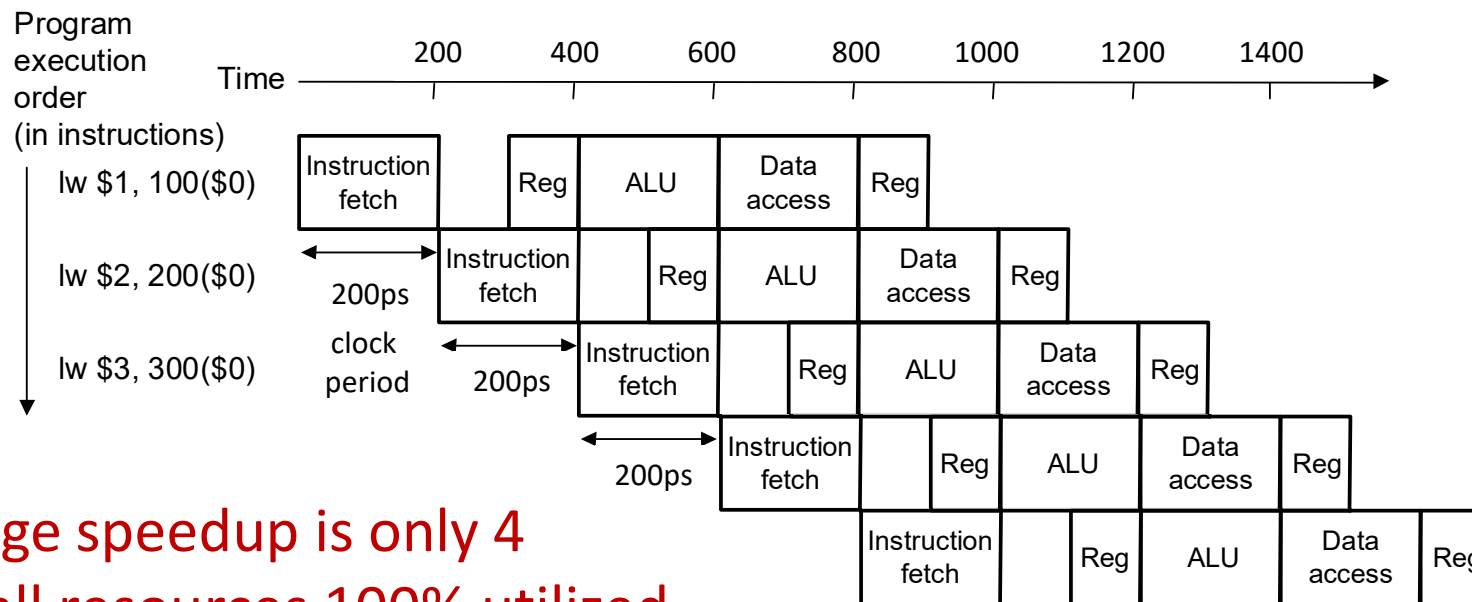Is this the correct partitioning?
Why not 4 or 6 stages? Why not different boundaries

18-447-S22-L07-S15, James C. Hoe, CMU/ECE/CALCM, ©2022

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
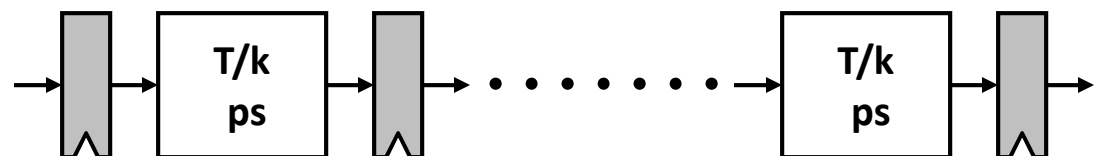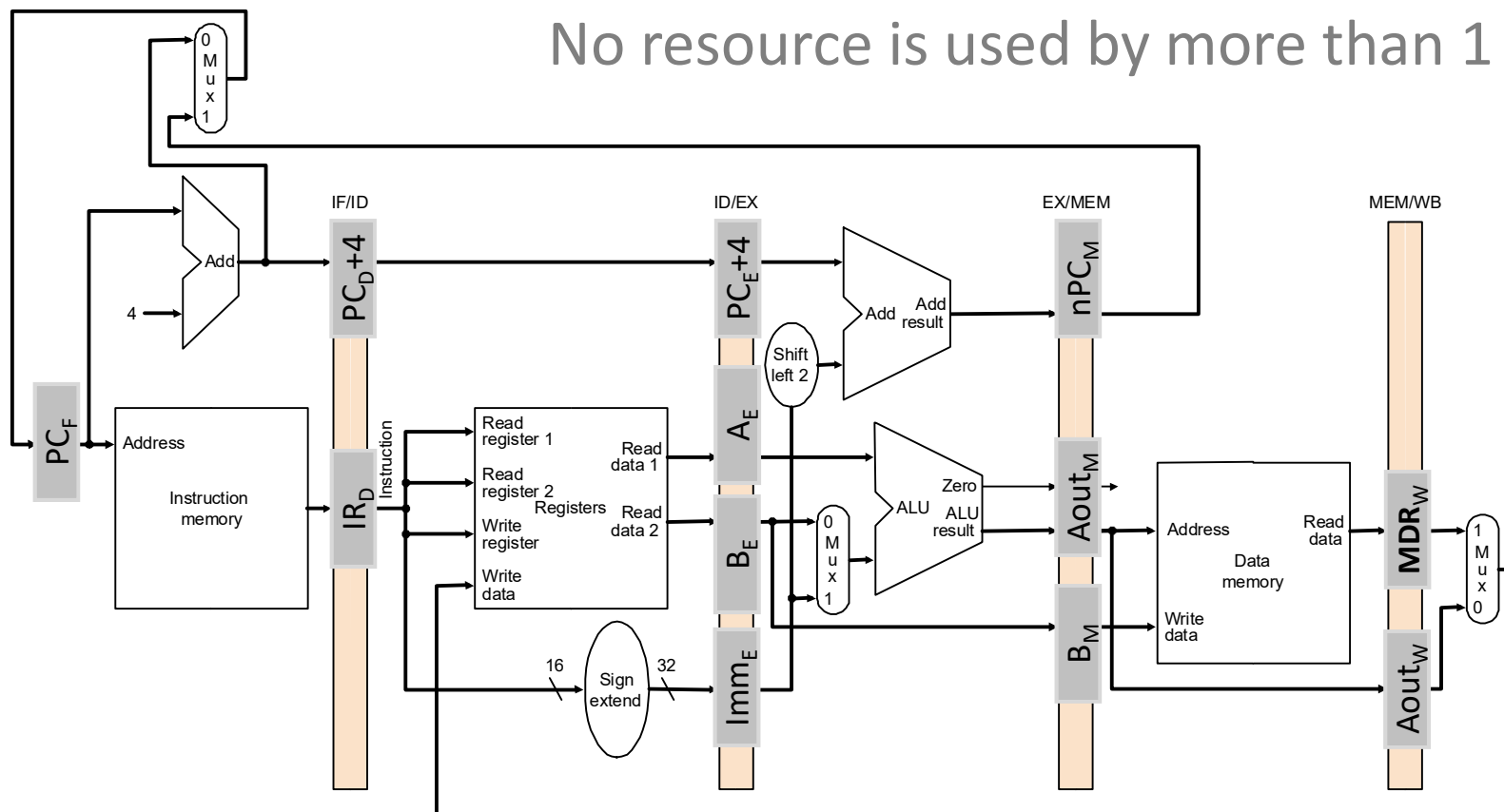
Internal and External Fragmentation
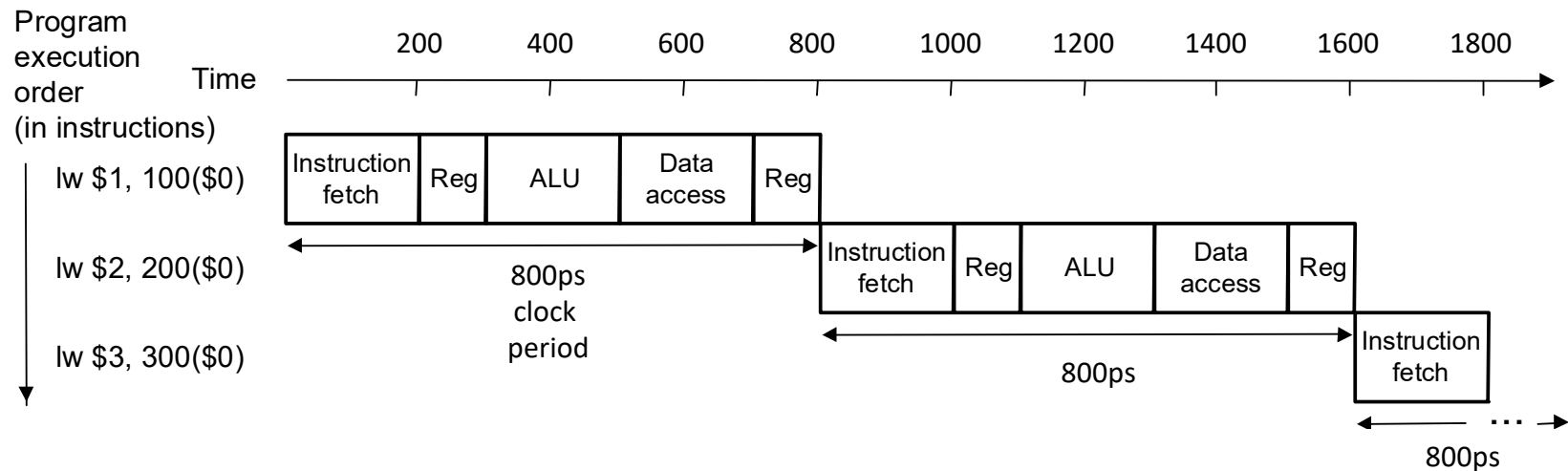
- 5-stage speedup is only 4
- Not all resources 100% utilized

Pipeline Registers

No resource is used by more than 1 stage!

18-447-S22-L07-S17, James C. Hoe, CMU/ECE/CALCM, ©2022

Pipelined Operation

Pipelined Operation

sub $11, $2, $3   lw $10, 20($1)

Clock 5
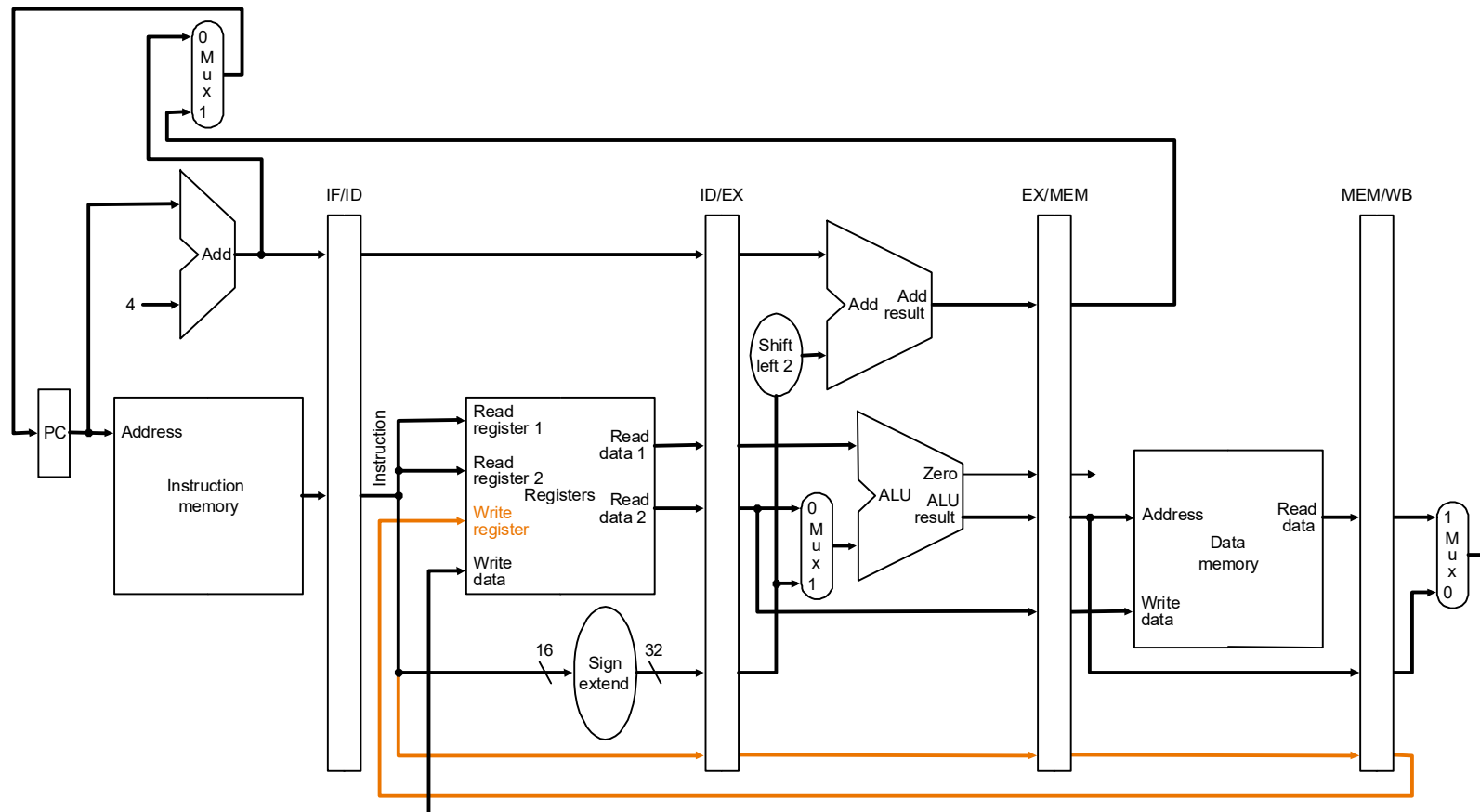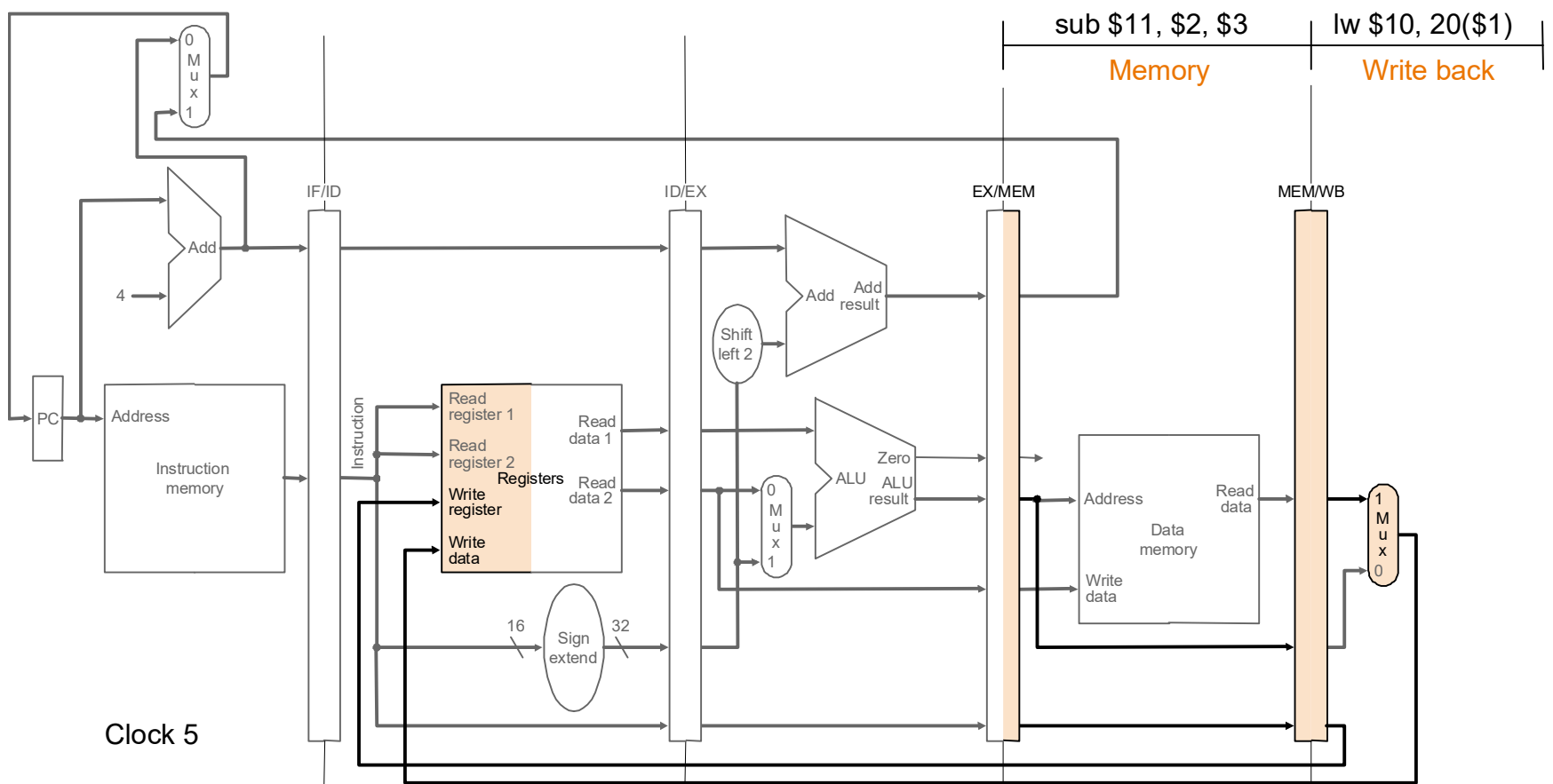
What if LW dest is $2?

Optimize Latency of ALU Insts?

steps	IF	ID	EX	MEM	WB
R-type	√	√	√	**?**	√
I-type	√	√	√		√
LW	√	√	√	√	√
SW	√	√	√	√	
Bxx/JALR	√	√	√		--/√
JAL	√		√		√
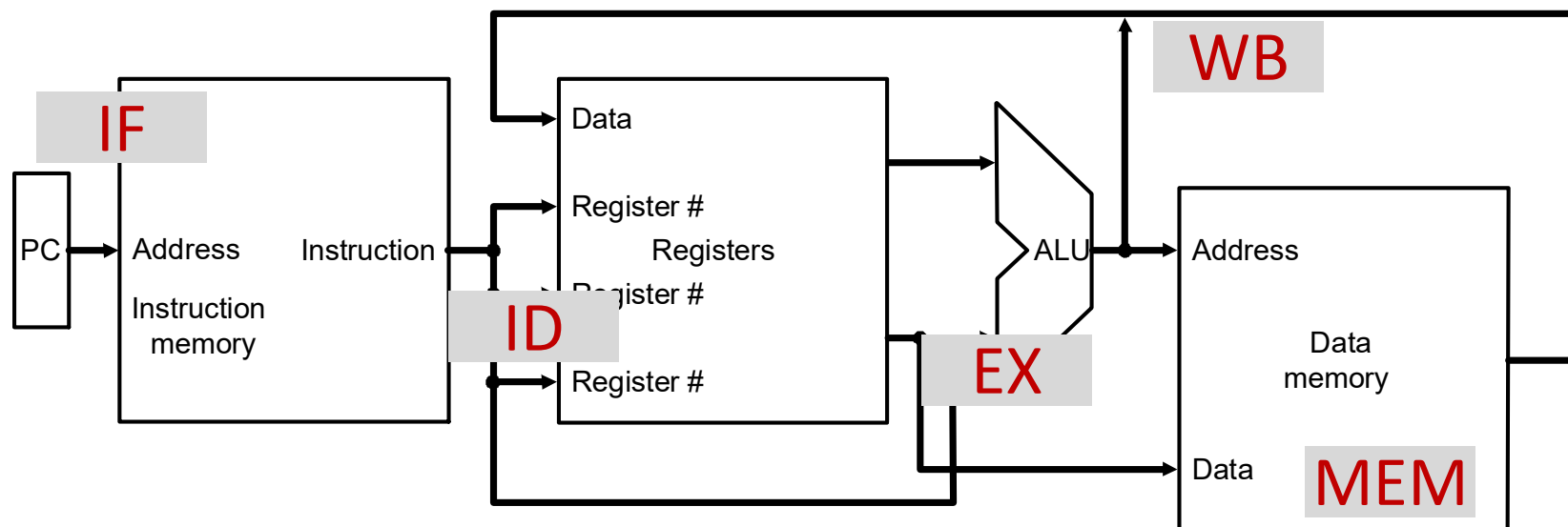
Illustrating Pipeline Operation: Resource View

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
ID		I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
EX			I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
MEM				I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
WB					I_0	I_1	I_2	I_3	I_4	I_5	I_6

Illustrating Pipeline Operation: Operation View
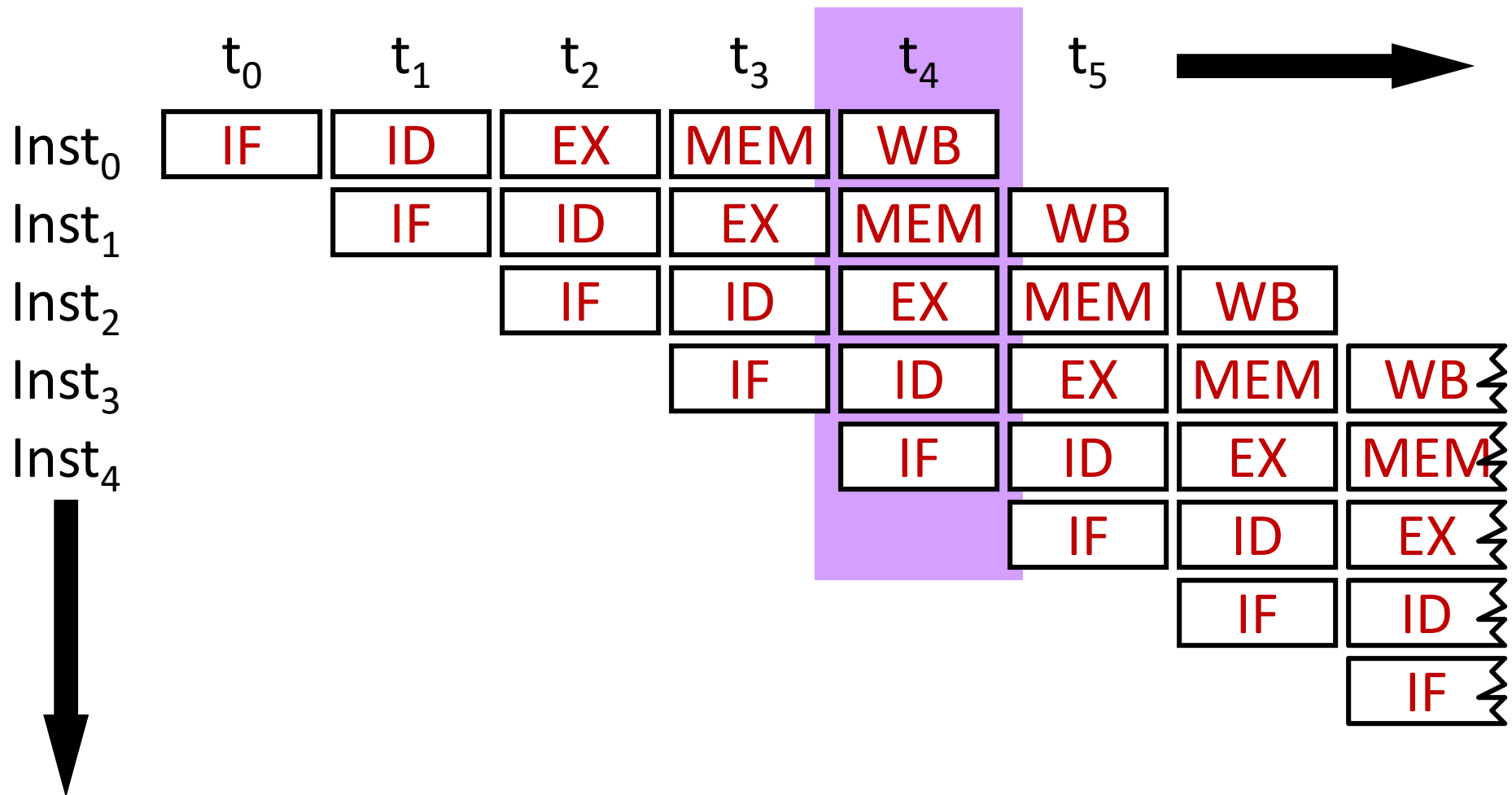
Example: Read-after-Write Hazard

	t_0	t_1	t_2	t_3	t_4	t_5
addi **x1**, x0, 0	IF	ID	EX	MEM	WB	
addi x2, **x1**, 0		IF	ID	EX	MEM	WB
addi x3, **x1**, 0			IF	ID	EX	MEM
addi x4, **x1**, 0				IF	ID	EX
addi x5, **x1**, 0					IF	?ID
addi x6, **x1**, 0						IF
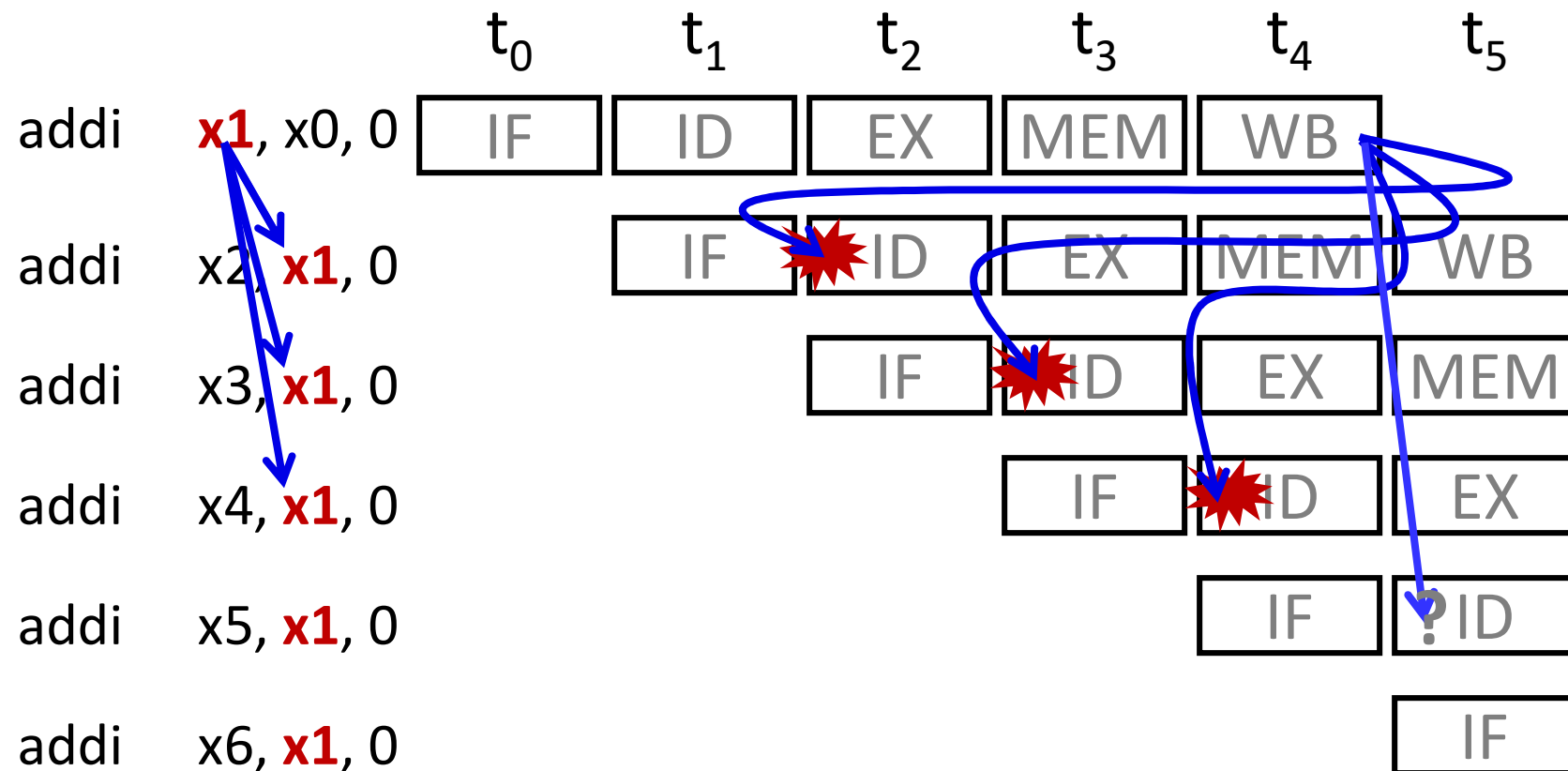
Example: Pipeline Stalls

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	I_0	I_1	I_2	I_3	I_4	I_4	I_4	I_4	I_5	I_6	I_7
ID		I_0	I_1	I_2	I_3	I_3	I_3	I_3	I_4	I_5	I_6
EX			I_0	I_1	I_2	$\varnothing$	$\varnothing$	$\varnothing$	I_3	I_4	I_5
MEM				I_0	I_1	I_2	$\varnothing$	$\varnothing$	$\varnothing$	I_3	I_4
WB					I_0	I_1	I_2	$\varnothing$	$\varnothing$	$\varnothing$	I_3

I_2=addi x1, x0, 0 ; I_3=addi x2, x1, 0 ;

Control Points
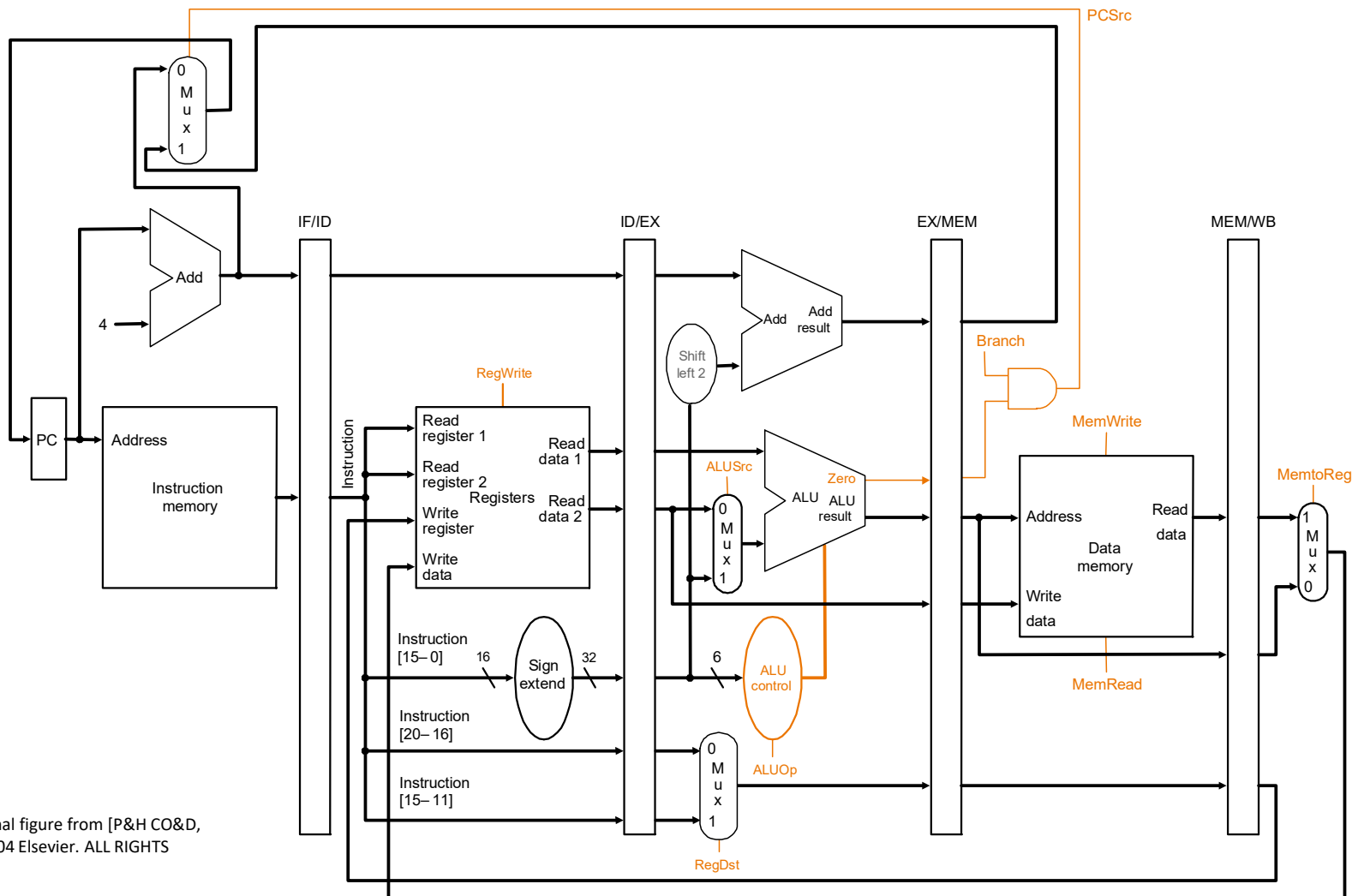
Identical set of control points as the single-cycle datapath!!

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Sequential Control: Special Case

- For a given instruction
  - same control settings as single-cycle, but
  - control signals required at different cycles, depending on stage
  - decode once using the same logic as single-cycle and buffer control signals until consumed
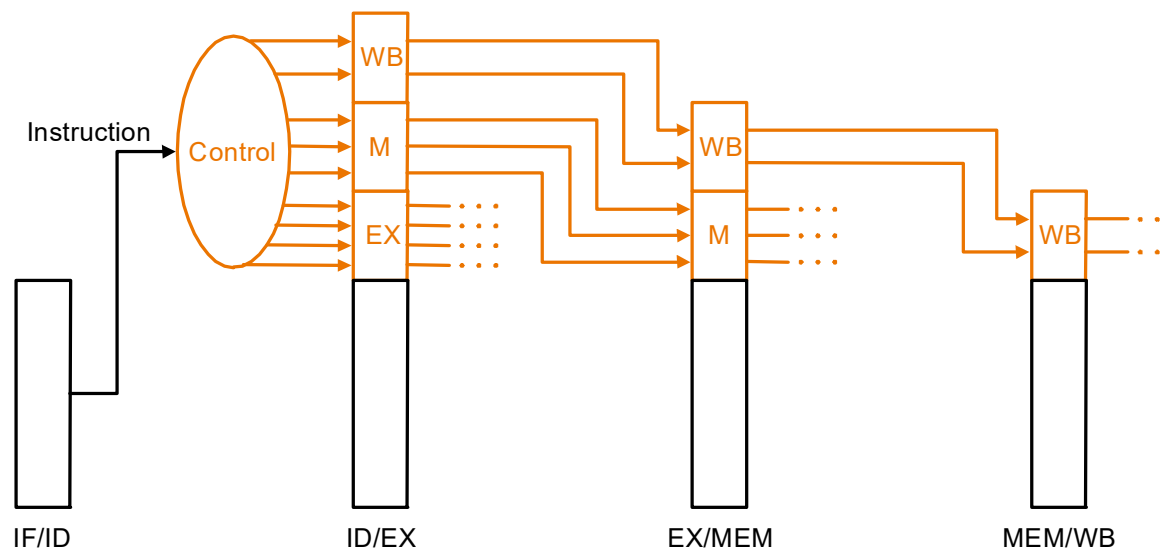

IF/ID   ID/EX   EX/MEM   MEM/WB

Pipelined Control
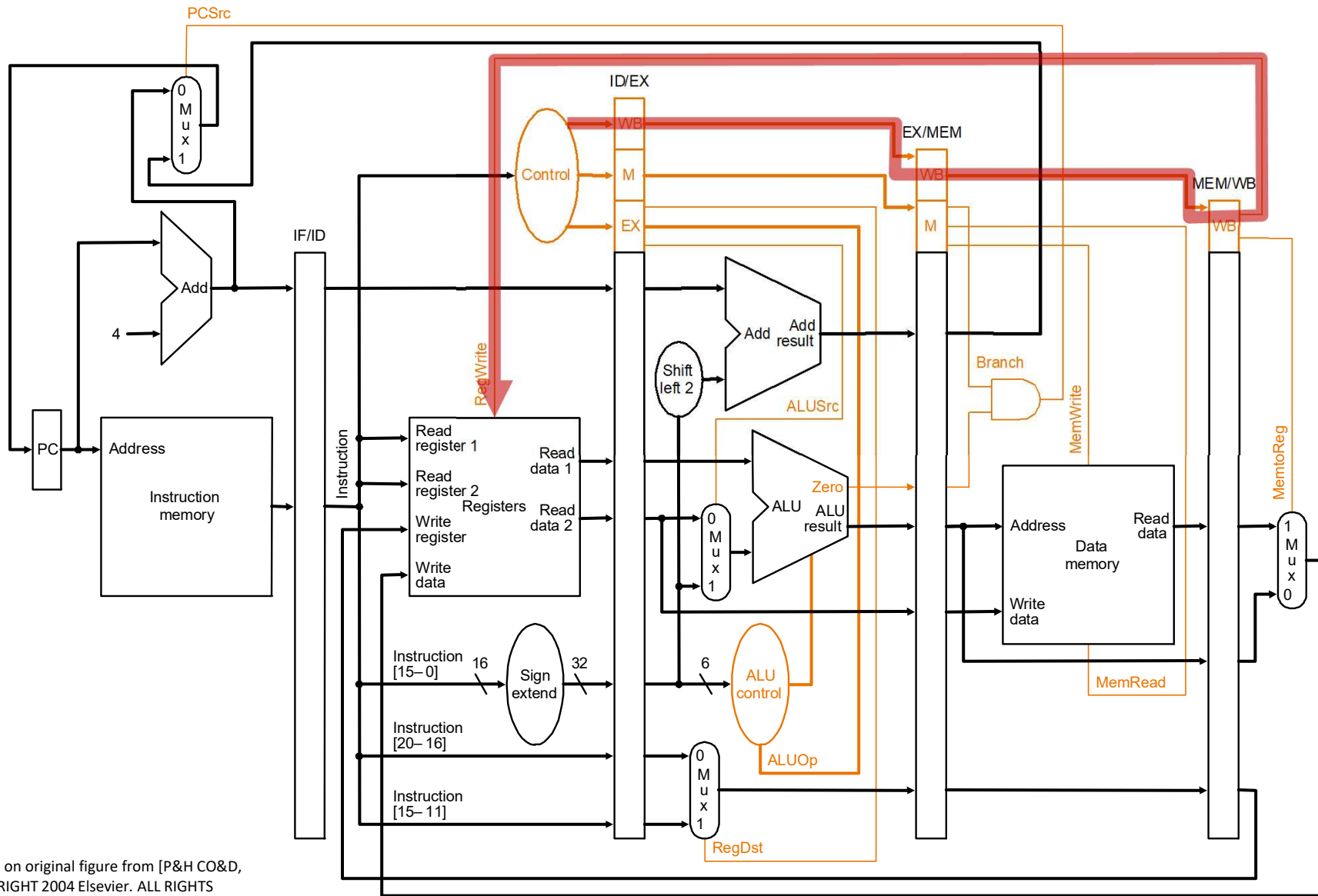
18-447-S22-L07-S27, James C. Hoe, CMU/ECE/CALCM, ©2022

This is all there is to it (without hazards)!!

Instruction Pipeline Reality

- **Not identical tasks**

  – coalescing instruction types into one "multi-function" pipe

  – external fragmentation (some idle stages)

- **Not uniform suboperations**

  – group or sub-divide steps into stages to minimize variance

  – internal fragmentation (some too-fast stages)

- **Not independent tasks**

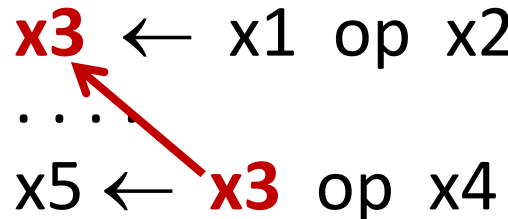 – dependency detection and resolution

  – next lecture(s)
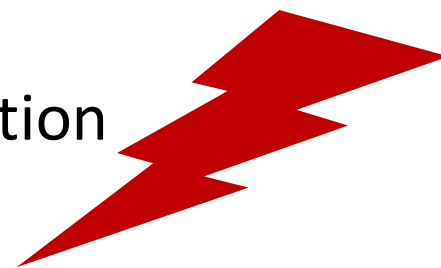
Even more messy if not RISC

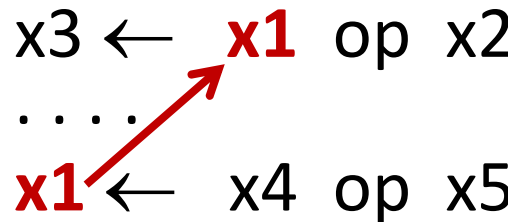# Data Dependence

Data dependence

$$x3 \leftarrow x1 \ op \ x2$$

Read-after-Write (RAW)

. . . .

$$x5 \leftarrow x3 \ op \ x4$$
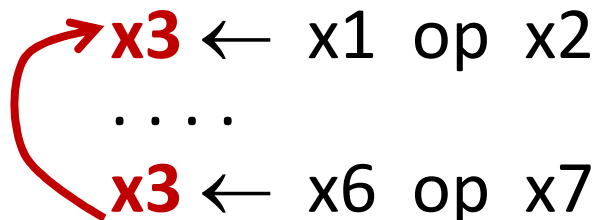
Anti-dependence

$$x3 \leftarrow x1 \ op \ x2$$

Write-after-Read (WAR)

. . . .

$$x1 \leftarrow x4 \ op \ x5$$

Output-dependence

$$x3 \leftarrow x1 \ op \ x2$$

Write-after-Write (WAW)

. . . .

$$x3 \leftarrow x6 \ op \ x7$$

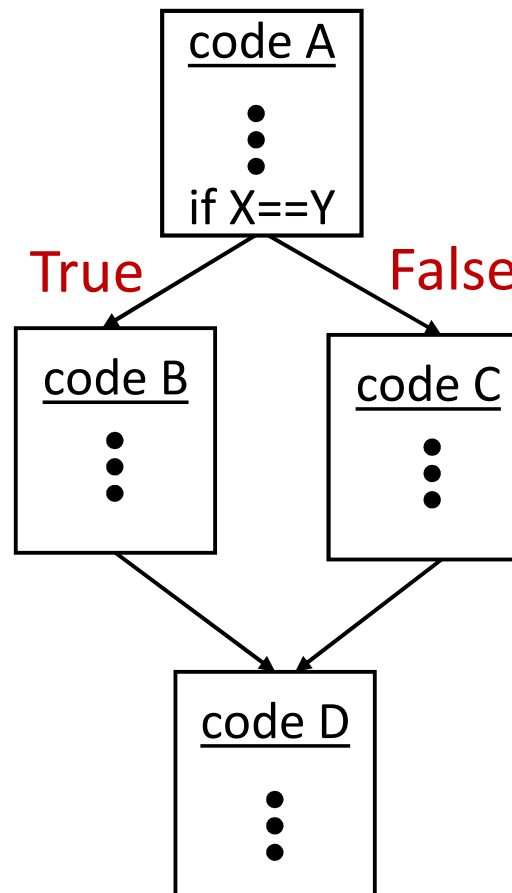Don't forget memory instructions

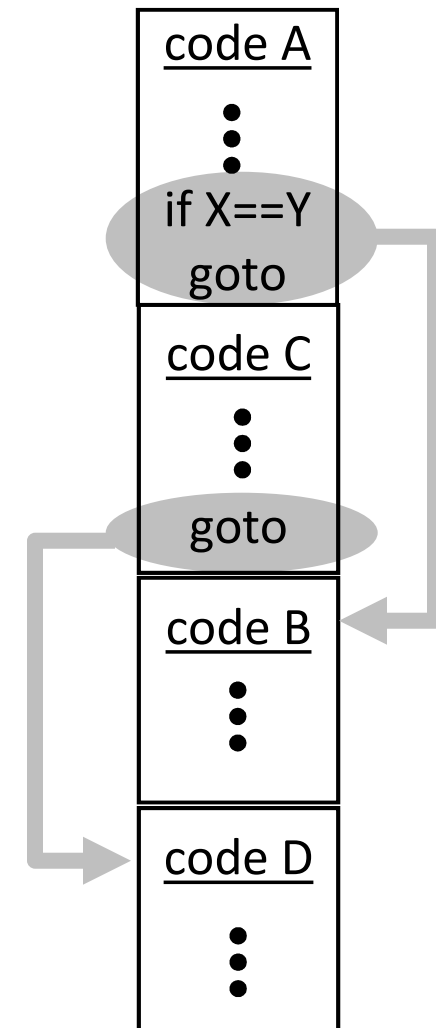# Control Dependence

- C-Code

```
{ code A }
if X==Y then
    { code B }
else
    { code C }
{ code D }
```

Control Flow Graph

Assembly Code (linearized)

Does B or C come after A?