

CS 110

Computer Architecture

Lecture 12:

Pipelining

Instructors:

Sören Schwertfeger & Chundong Wang

<https://robotics.shanghaitech.edu.cn/courses/ca/20s/>

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C



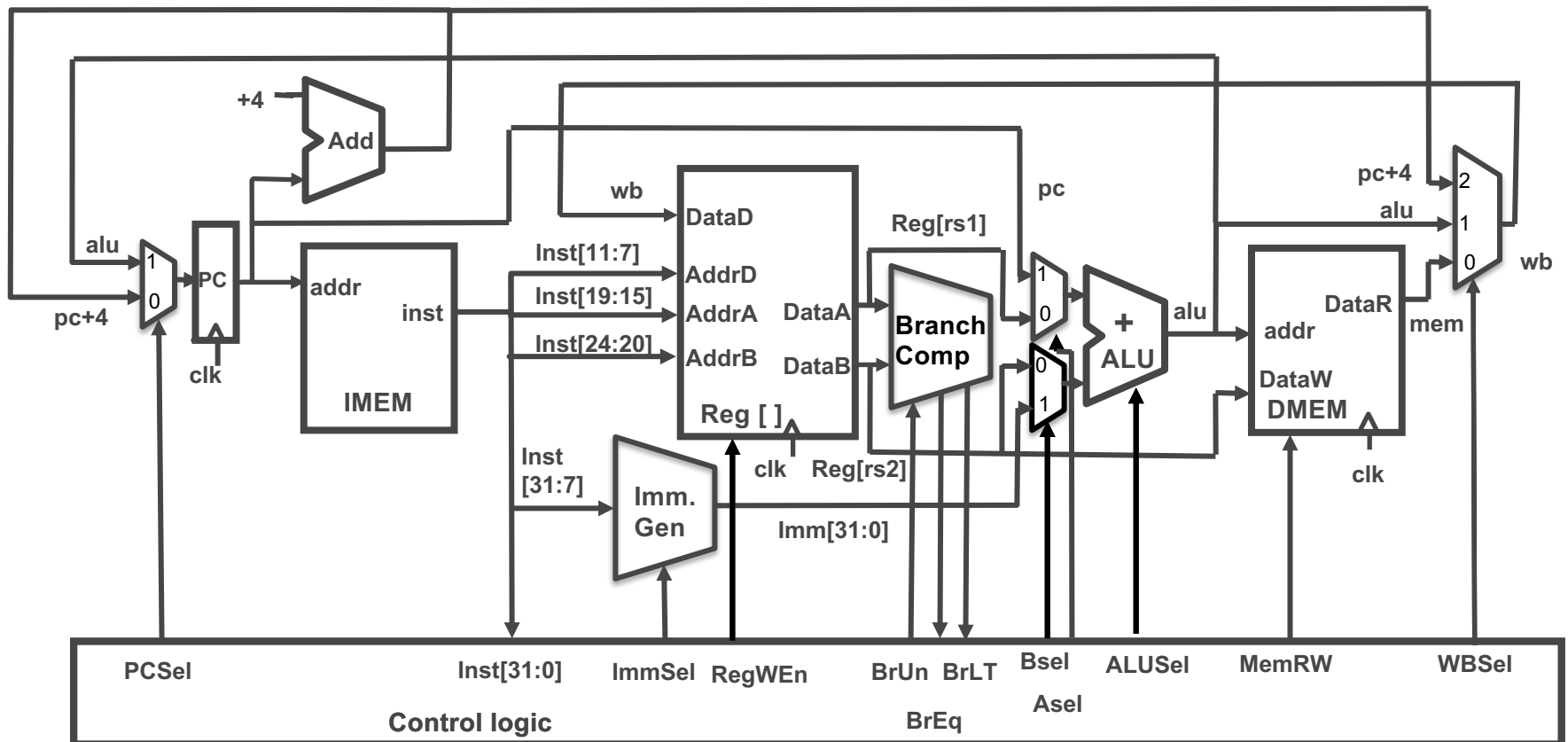
Admin

- Update your zoom app!
- From Thursday on we will use a password – find it on piazza
- HW4 is due this Friday!

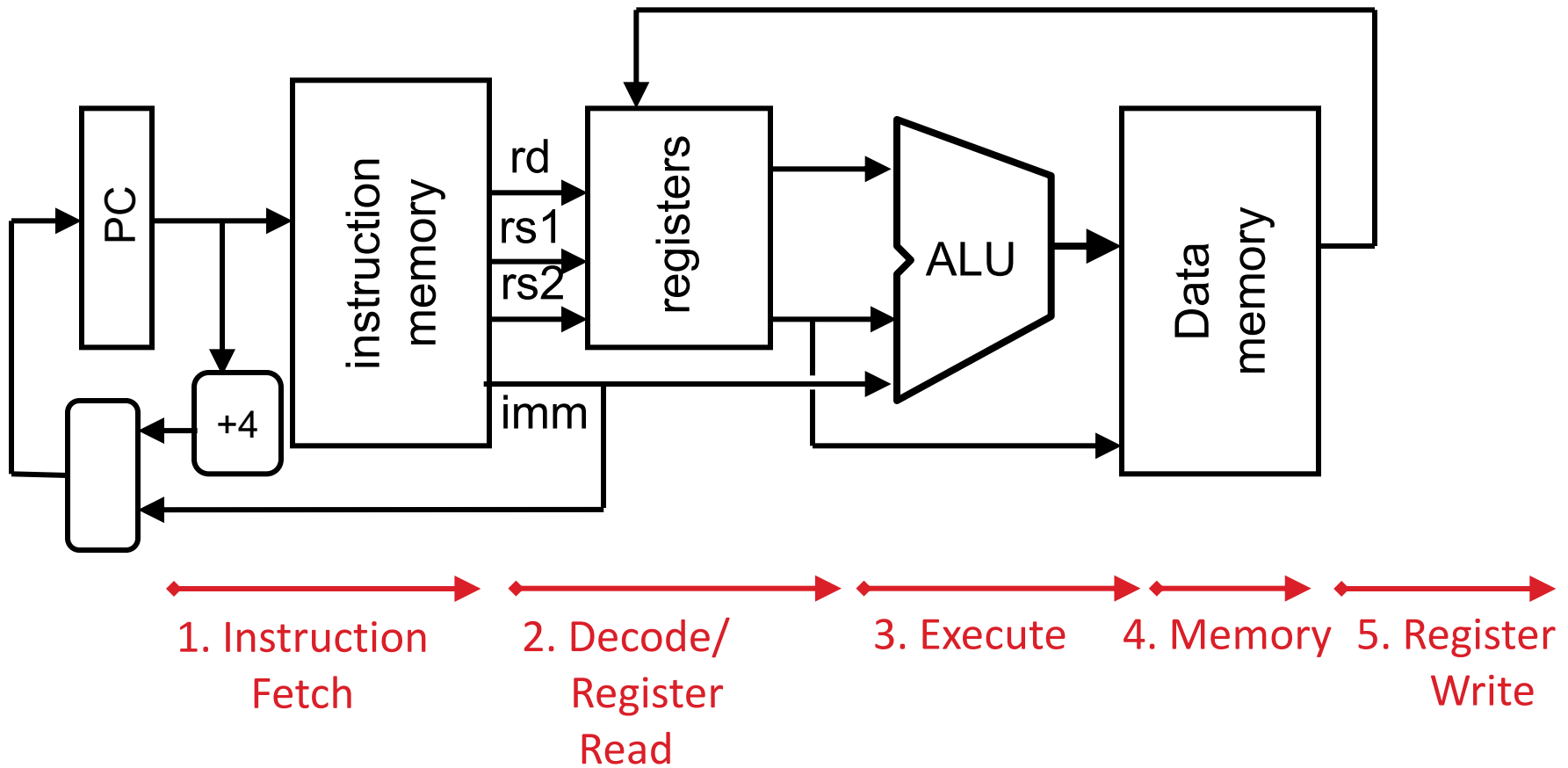
Agenda

- Pipelining
- Hazards
 - Structural
 - Data
 - R-type instructions
 - Load
 - Control

Complete Single-Cycle RV32I Datapath!



Stages of Execution on Datapath



Single Cycle Performance

- Assume time for actions are
 - 100ps for register read or write; 200ps for other events
- Clock period is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- Clock rate (cycles/second = Hz) = $1/\text{Period (seconds/cycle)}$

Single Cycle Performance

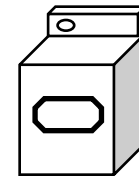
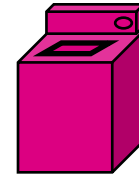
- Assume time for actions are
 - 100ps for register read or write; 200ps for other events
- Clock period is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

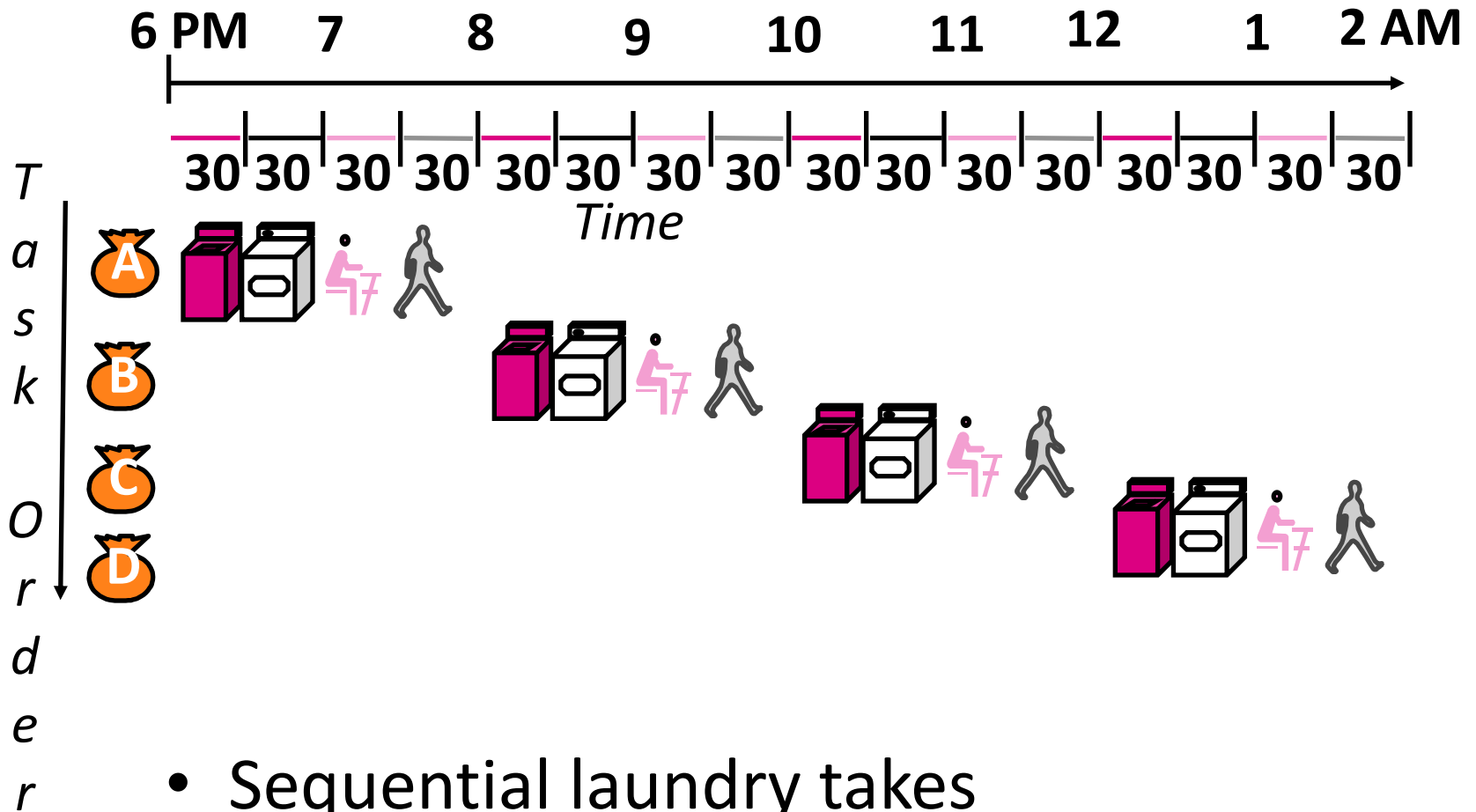
- What can we do to improve clock rate?
- Will this improve performance as well?
 - Want increased clock rate to mean faster programs

Gotta Do Laundry

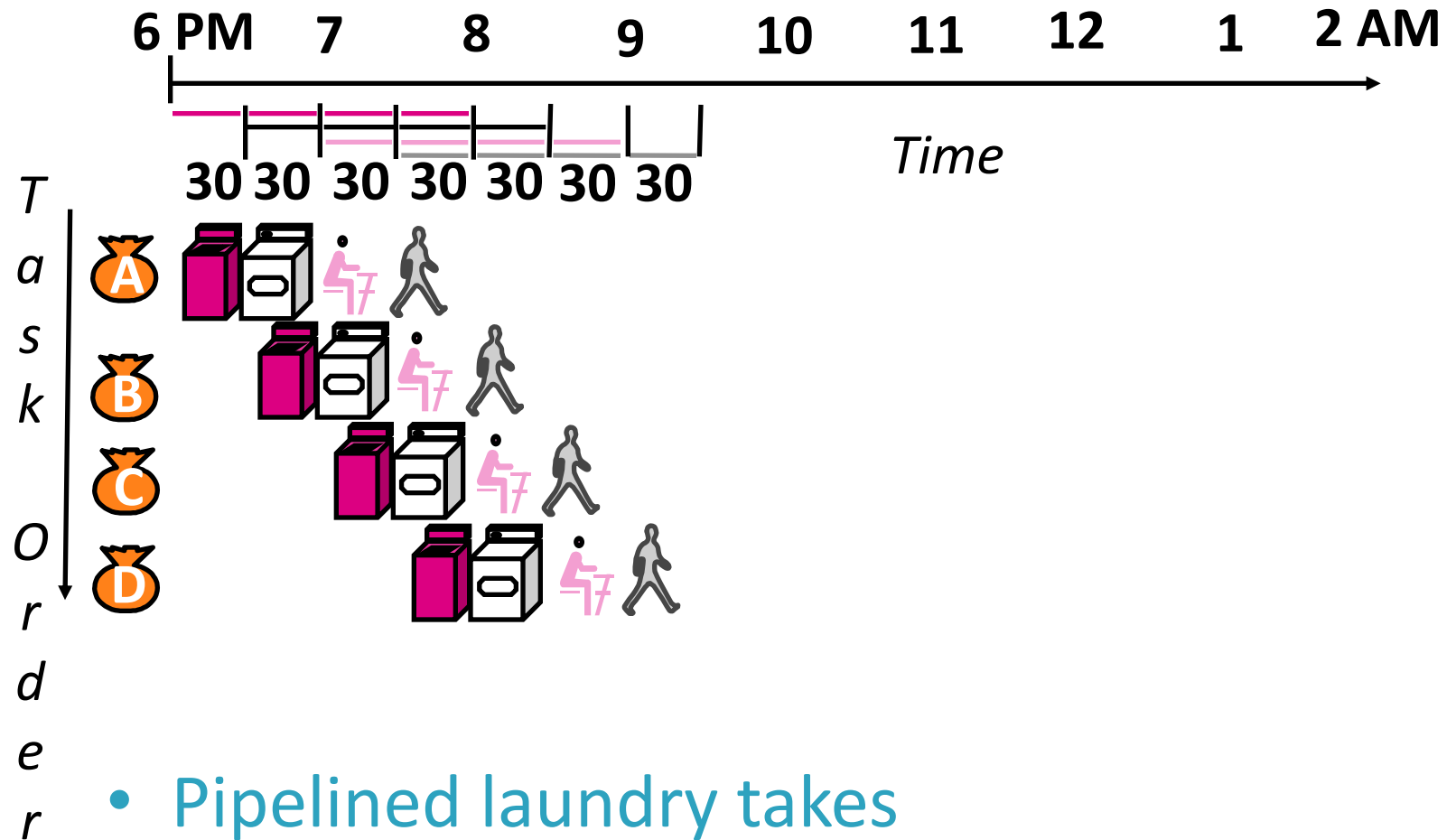
- Students 阿安 (A An), 鲍伯 (Bao Bo), 陈晨 (Chen Chen) and 丁丁 (Ding Ding) each have one load of clothes to wash, dry, fold, and put away
 - Washer takes 30 minutes
 - Dryer takes 30 minutes
 - “Folder” takes 30 minutes
 - “Stasher” takes 30 minutes to put clothes into drawers



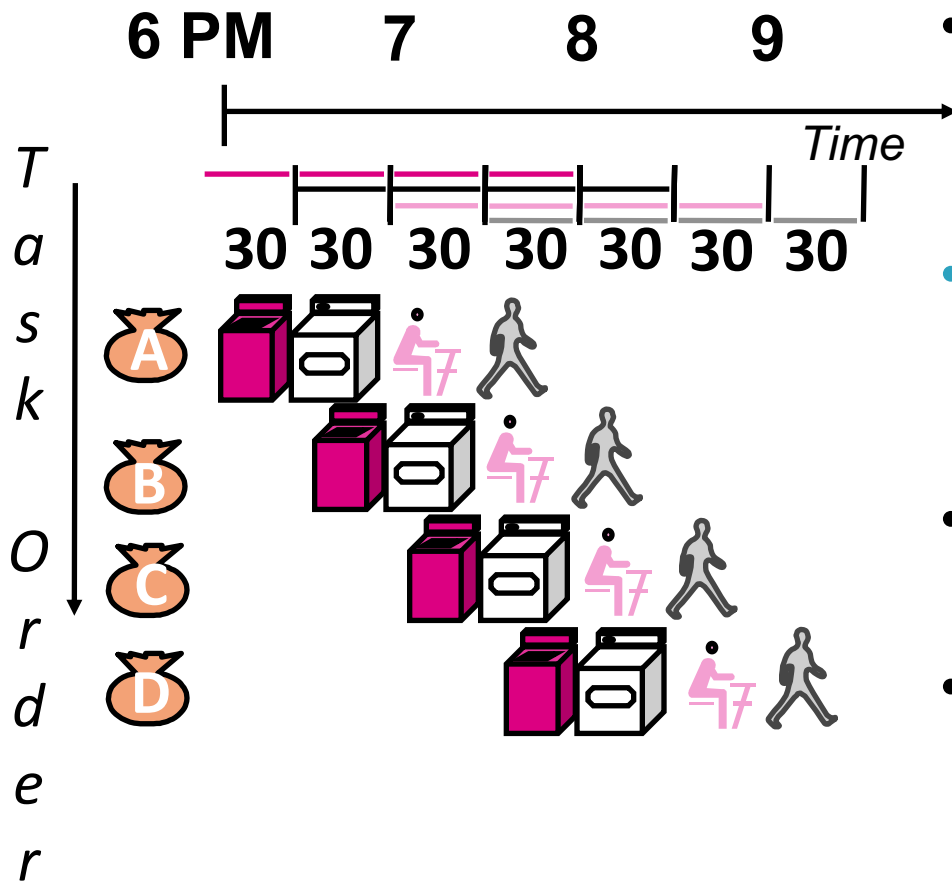
Sequential Laundry



Pipelined Laundry

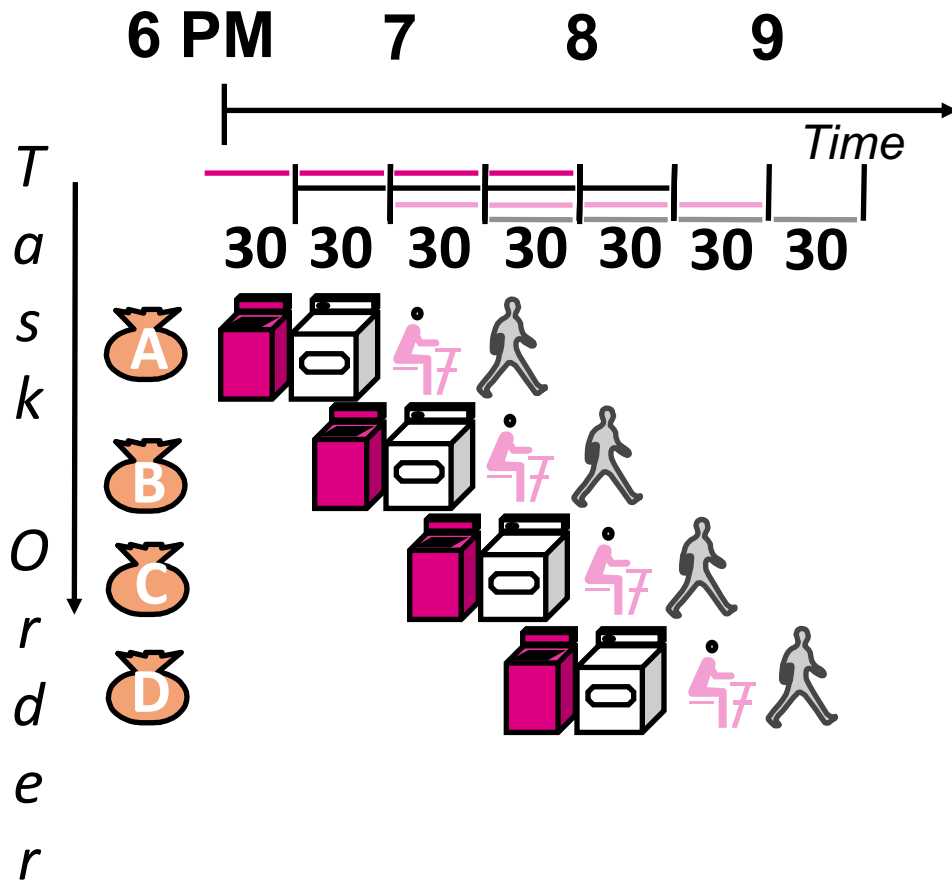


Pipelining Lessons (1/2)



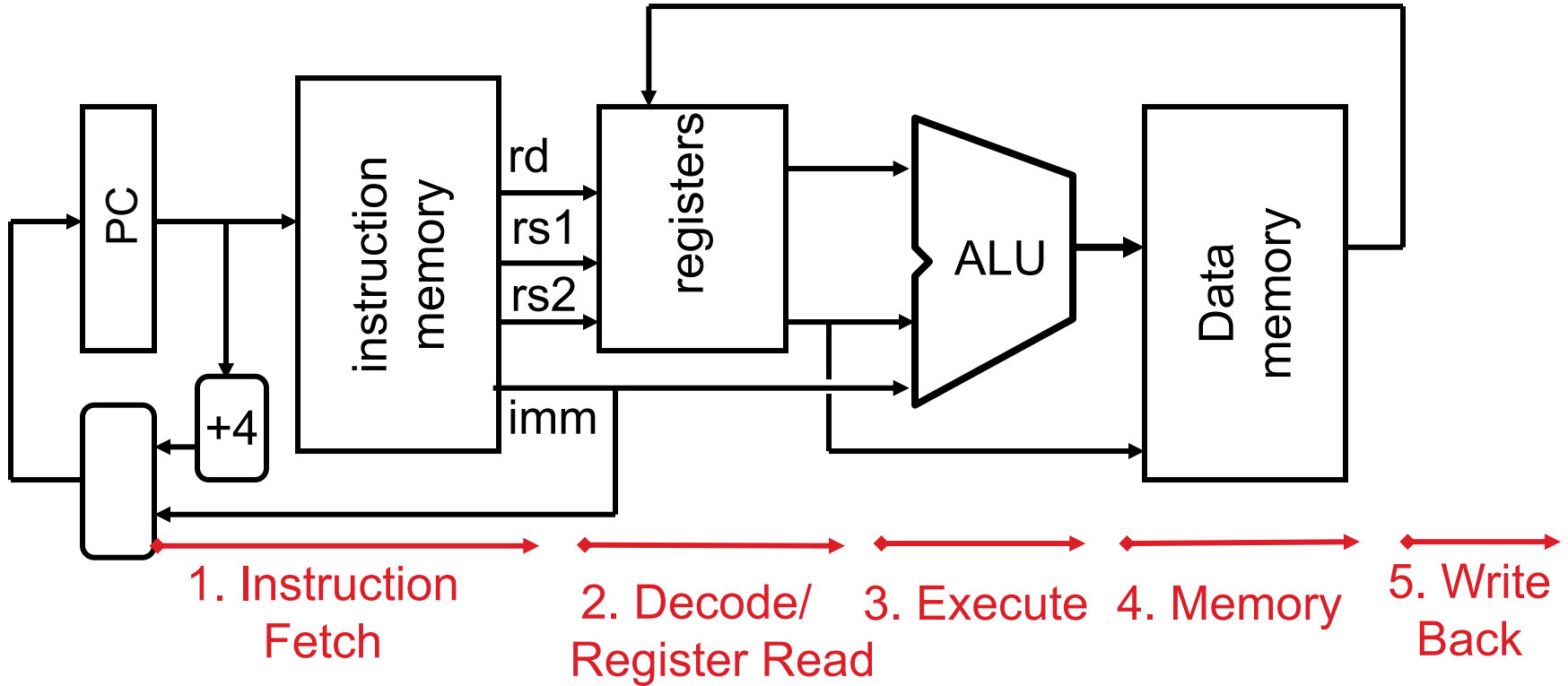
- Pipelining doesn't help [latency](#) of single task, it helps [throughput](#) of entire workload
- [Multiple](#) tasks operating simultaneously using different resources
- Potential speedup = [Number pipe stages](#)
- Time to “[fill](#)” pipeline and time to “[drain](#)” it reduces speedup

Pipelining Lessons (2/2)









- Suppose new Dryer takes 20 minutes, new Folder takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

Single Cycle Datapath



Pipelining with RISC-V

Phase	Pictogram	t_{step} Serial	t_{cycle} Pipelined
Instruction Fetch		200 ps	200 ps
Reg Read		100 ps	200 ps
ALU		200 ps	200 ps
Memory		200 ps	200 ps
Register Write		100 ps	200 ps
$t_{instruction}$		800 ps	1000 ps

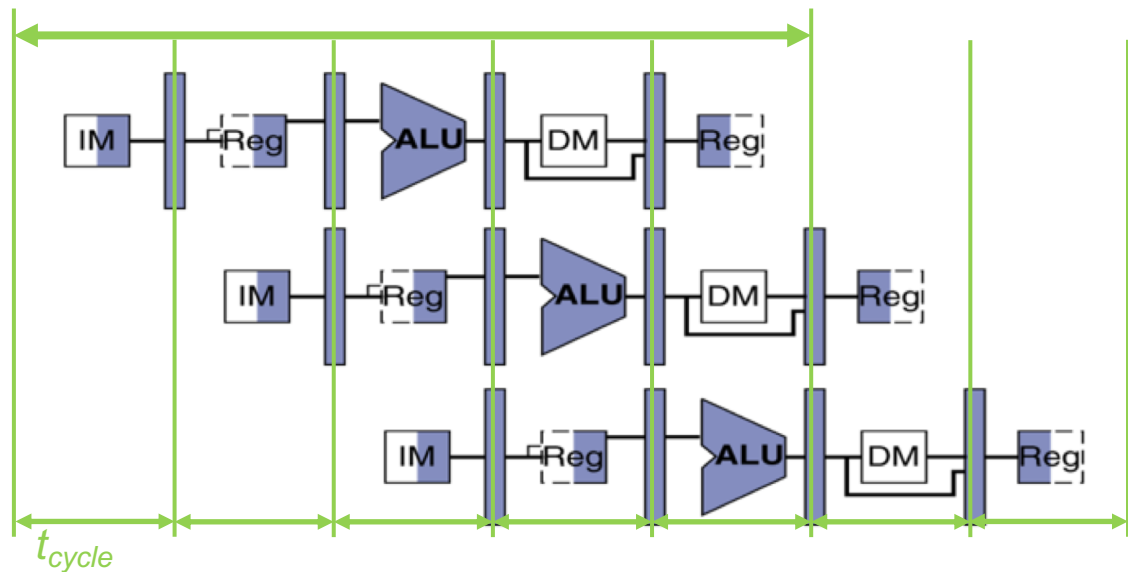
instruction sequence



add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

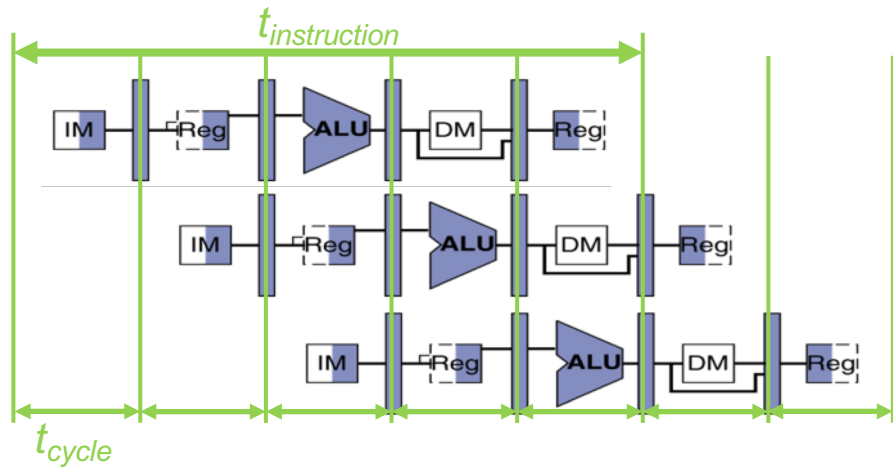


Pipelining with RISC-V

instruction sequence



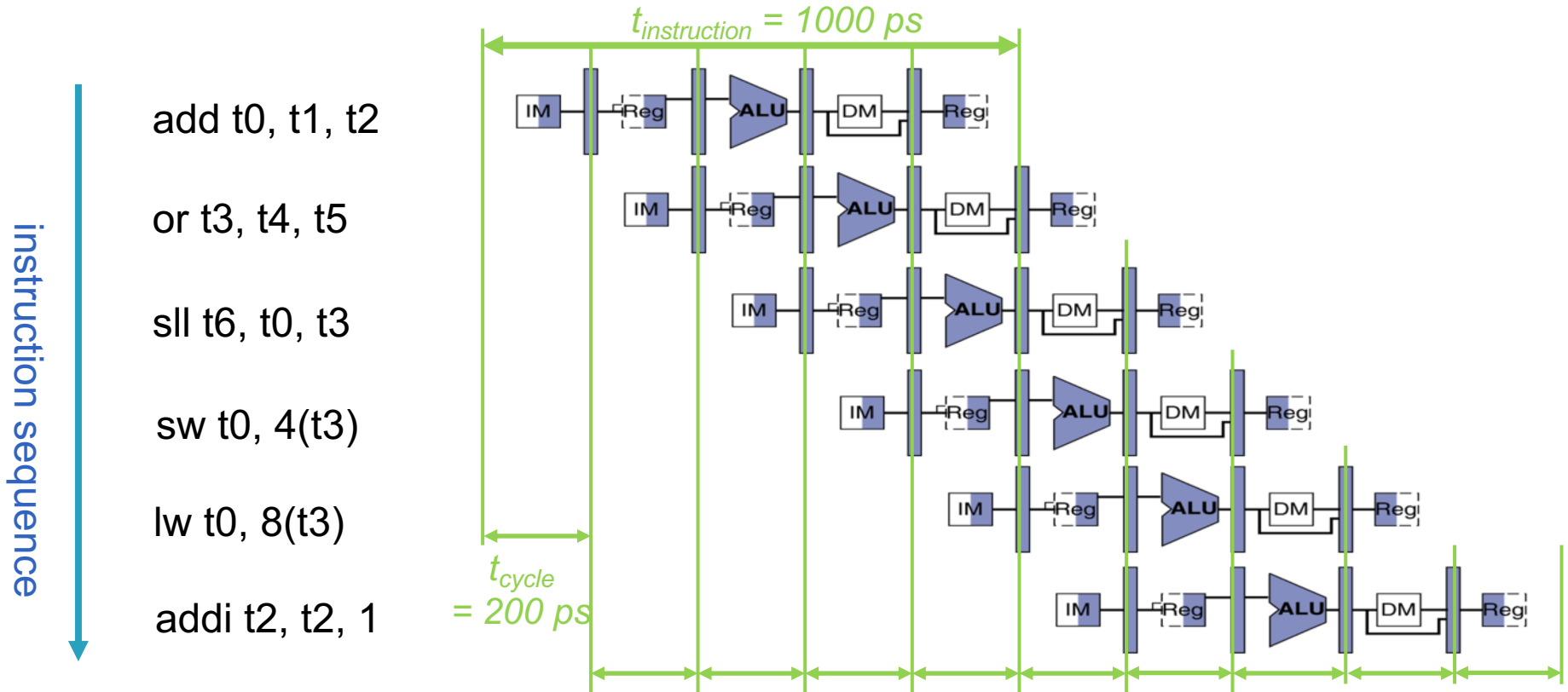
add t0, t1, t2
or t3, t4, t5
sll t6, t0, t3



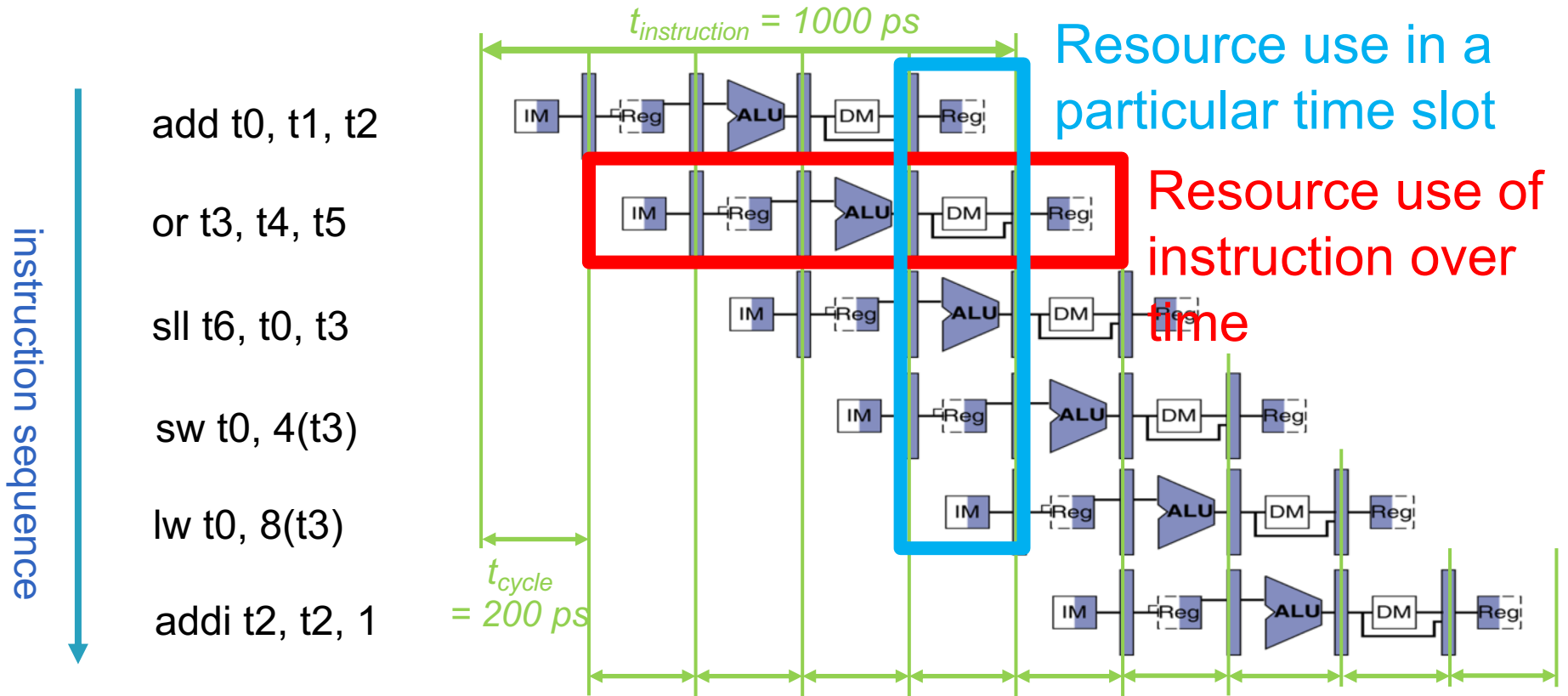
	Single Cycle	Pipelining
Timing	$t_{step} = 100 \dots 200 \text{ ps}$	$t_{cycle} = 200 \text{ ps}$
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	$= t_{cycle} = 800 \text{ ps}$	1000 ps
CPI (Cycles Per Instruction)	~ 1 (ideal)	~ 1 (ideal), > 1 (actual)
Clock rate, f_s	$1/800 \text{ ps} = 1.25 \text{ GHz}$	$1/200 \text{ ps} = \mathbf{5 \text{ GHz}}$
Relative speed	1 x	4 x

Sequential vs Simultaneous

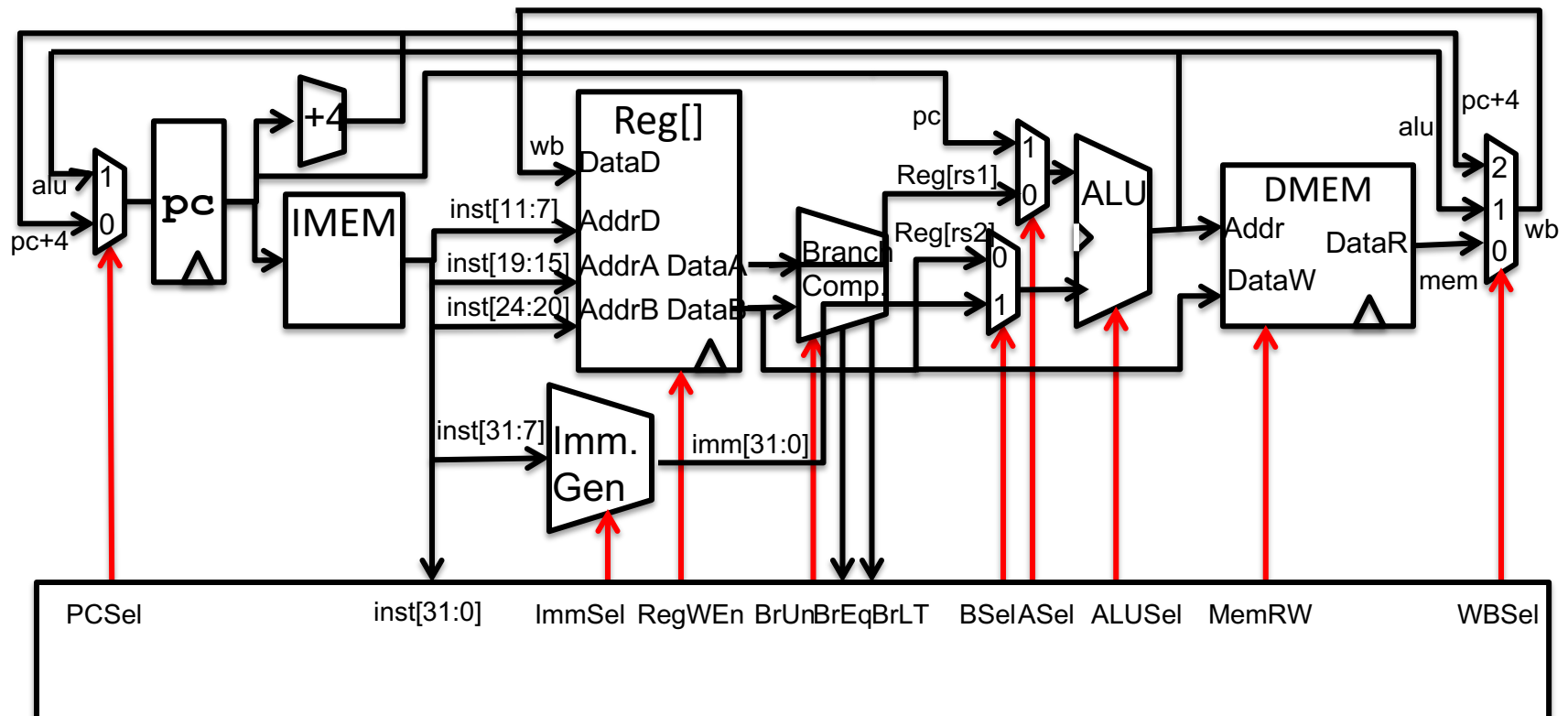
What happens sequentially, what happens simultaneously?



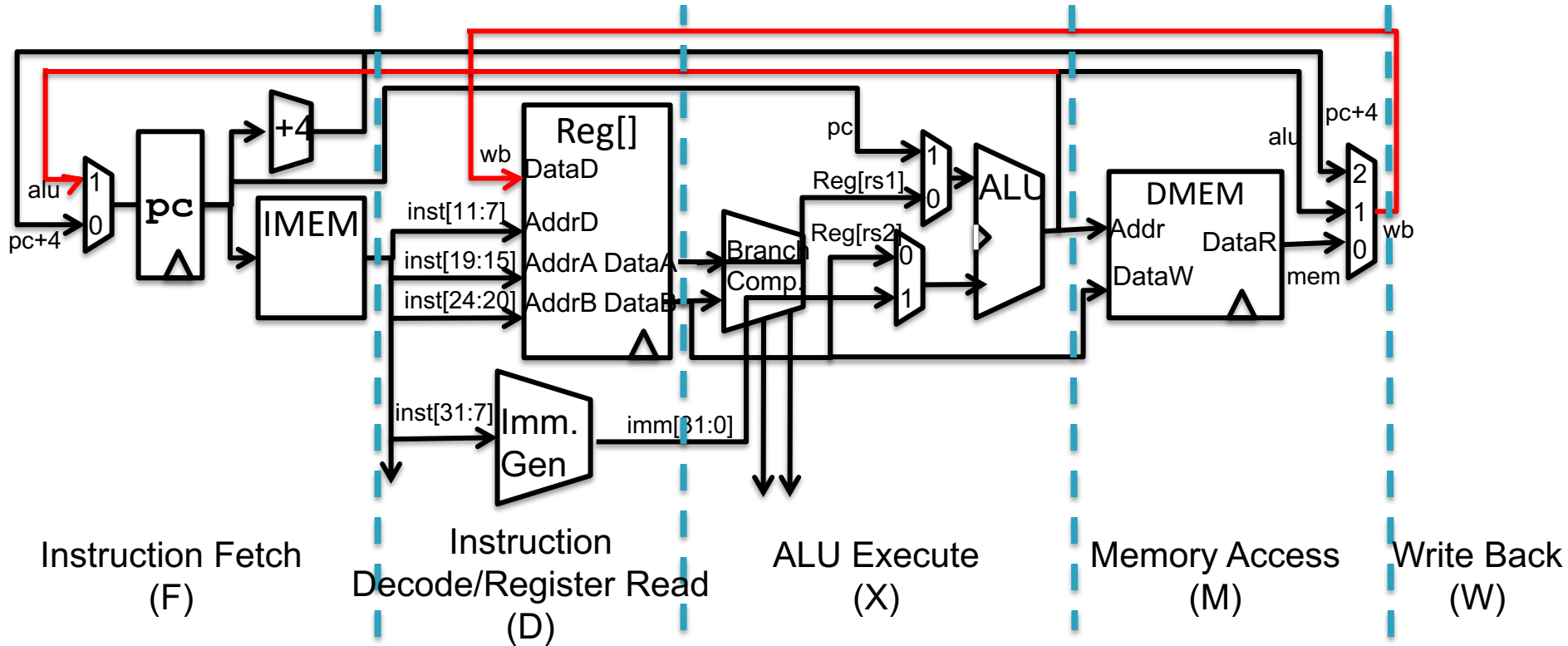
RISC-V Pipeline



Single-Cycle RISC-V RV32I Datapath

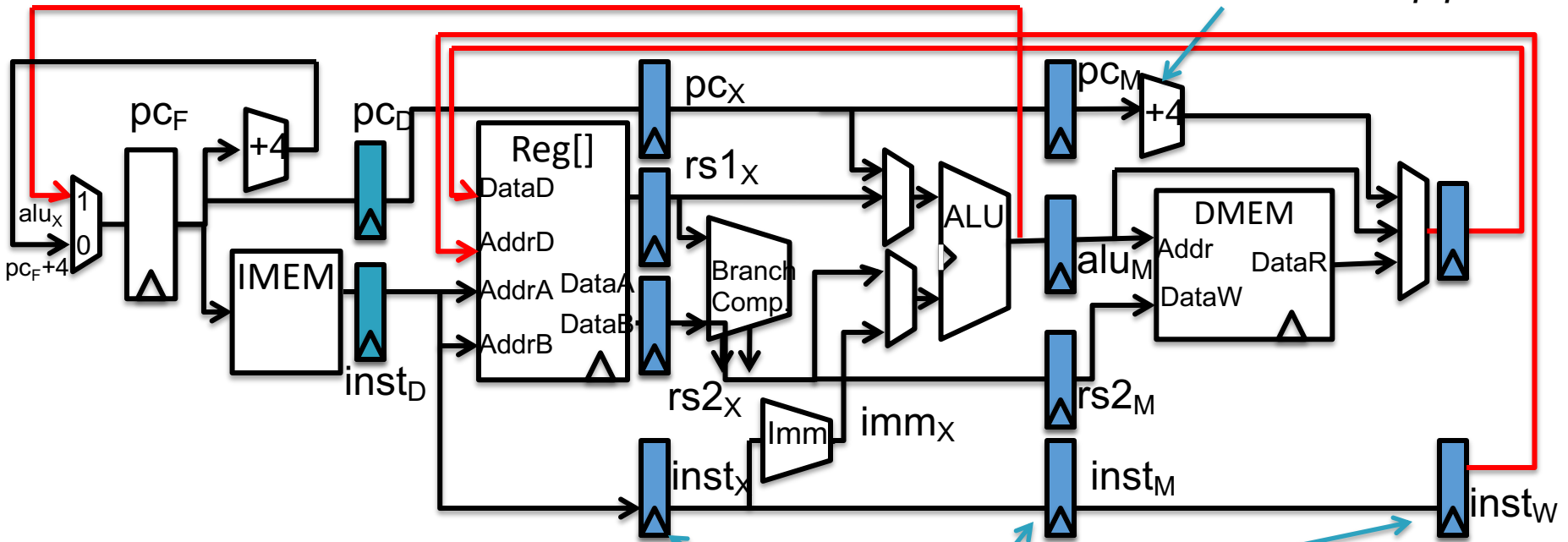


Pipelining RISC-V RV32I Datapath



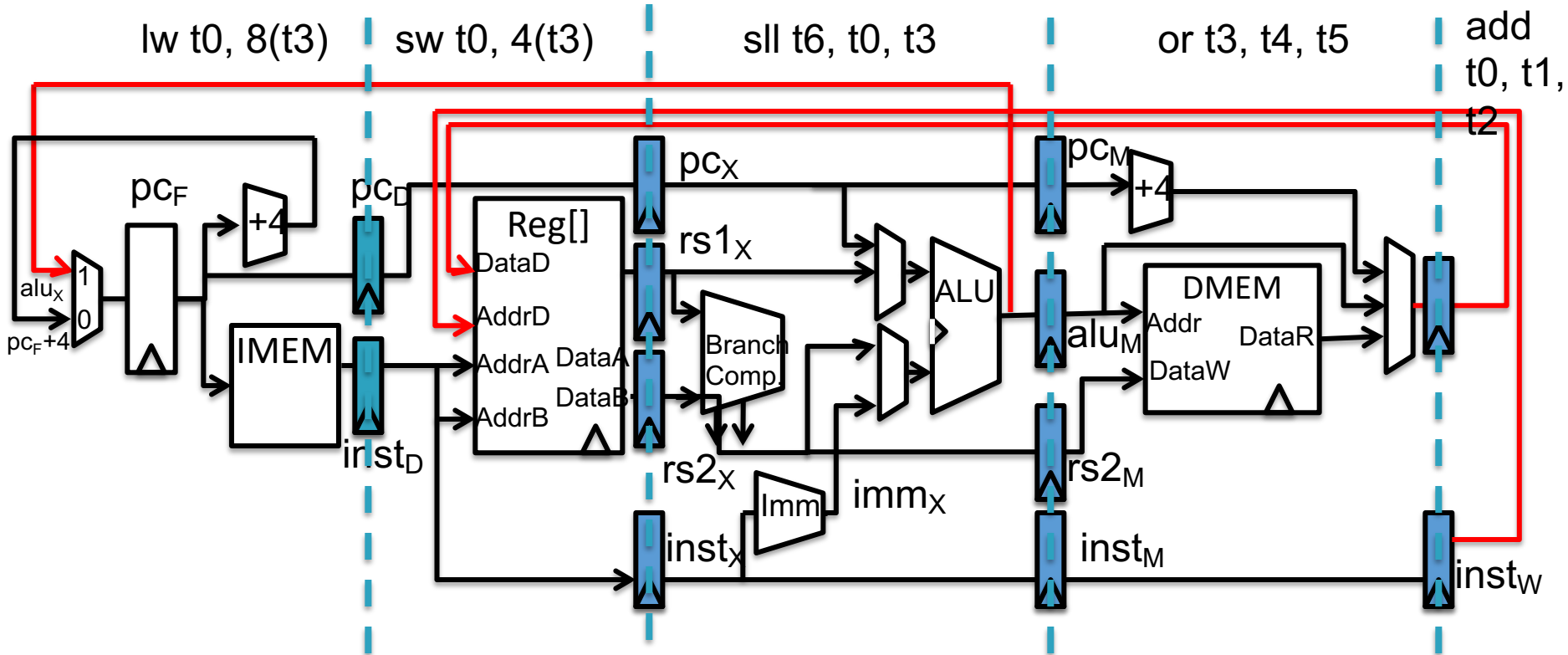
Pipelined RISC-V RV32I Datapath

Recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline



Must pipeline instruction along with data, so control operates correctly in each stage

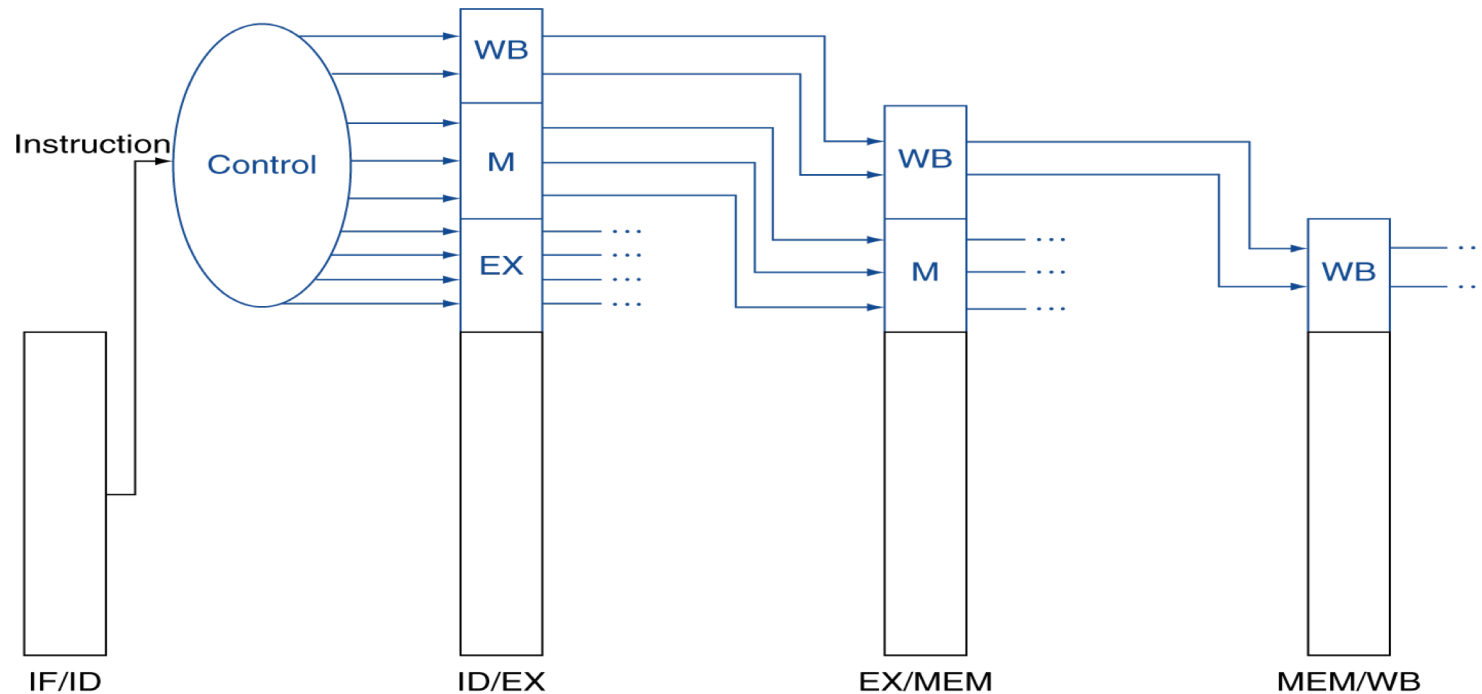
Each stage operates on different instruction



Pipeline registers separate stages, hold data for each instruction in flight

Pipelined Control

- Control signals derived from instruction
 - As in single-cycle implementation
 - Information is stored in pipeline registers for use by later stages



Question

Logic in some stages takes 200ps and in some 100ps. Clk-Q delay is 30ps and setup-time is 20ps. What is the maximum clock frequency at which a pipelined design with 5 stages can operate?

- A: 10GHz
- B: 5GHz
- C: 6.7GHz
- D: 4.35GHz
- E: 4GHz





TA Discussion

Video Anqi Pang

Watch After watching all videos of today...



Q & A



Quiz



Quiz

Piazza: "Online Lecture 12 Pipelining Poll"

- Select the statements that are TRUE:
 - A. Pipelining increases instruction throughput
 - B. Pipelining increases instruction latency
 - C. Pipelining increases clock frequency
 - D. Pipelining decreases number of components

Also: Select the make of the car that Prof. Schwertfeger likes to rent to drive fast on German Highways:

- E. BMW (宝马)
- F. Audi (奥迪)
- G. Mercedes-Benz (奔驰)
- H. Porsche (保时捷)

CS 110
Computer Architecture
Lecture 12:
Pipelining
Video 2: Hazards

Instructors:
Sören Schwertfeger & Chundong Wang

<https://robotics.shanghaitech.edu.cn/courses/ca/20s/>

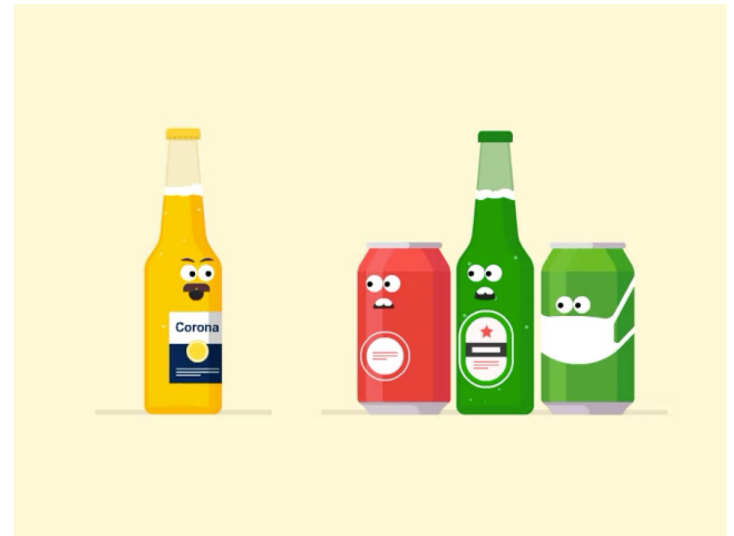
School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Agenda

- Pipelining
- Hazards
 - Structural
 - Data
 - R-type instructions
 - Load
 - Control



Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

1) *Structural hazard*

- A required resource is busy (e.g. needed in multiple stages)

2) *Data hazard*

- Data dependency between instructions
- Need to wait for previous instruction to complete its data read/write

3) *Control hazard*

- Flow of execution depends on previous instruction

Structural Hazard

- **Problem:** Two or more instructions in the pipeline compete for access to a single physical resource
- **Solution 1:** Instructions take it in turns to use resource, some instructions have to stall
- **Solution 2:** Add more hardware to machine
- Can always solve a structural hazard by adding more hardware

Regfile Structural Hazards

- Each instruction:
 - can read up to two operands in decode stage
 - can write one value in writeback stage
- Avoid structural hazard by having separate “ports”
 - two independent read ports and one independent write port
- Three accesses per cycle can happen simultaneously

Structural Hazard: Memory Access

instruction sequence

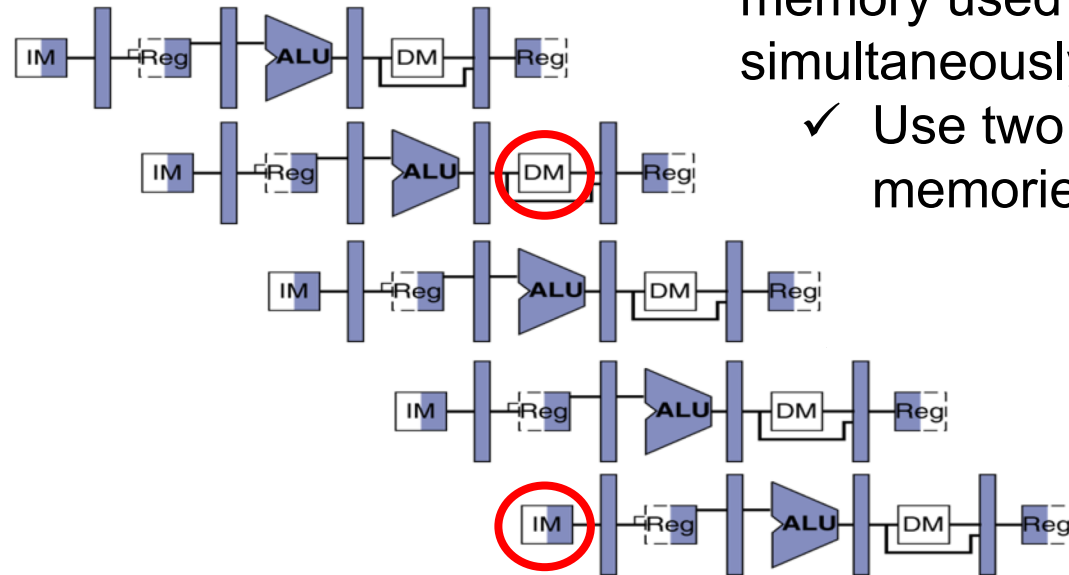
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

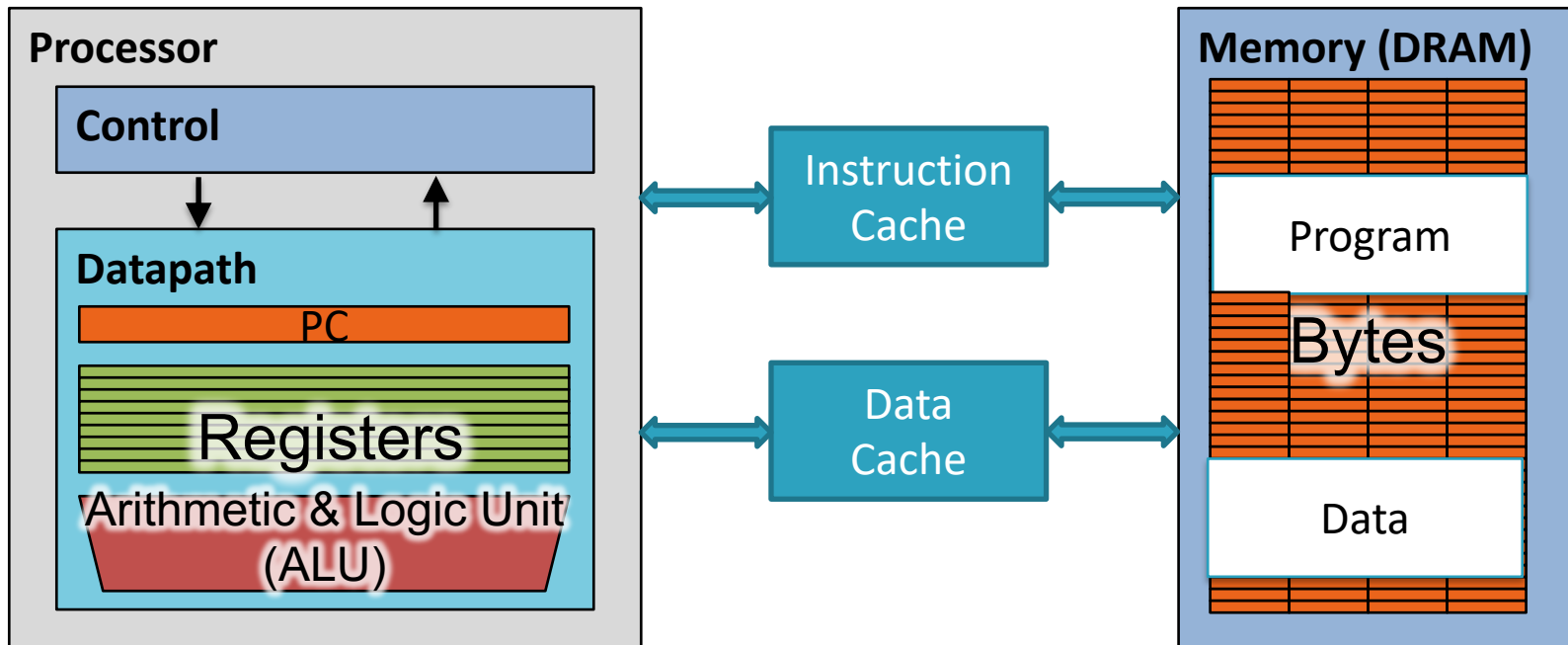
sw t0, 4(t3)

lw t0, 8(t3)



- Instruction and data memory used simultaneously
 - ✓ Use two separate memories

Instruction and Data Caches



Structural Hazards – Summary

- Conflict for use of a resource
- In RISC-V pipeline with a single memory
 - Load/store requires data access
 - Without separate memories, instruction fetch would have to *stall* for that cycle
 - All other operations in pipeline would have to wait
- Pipelined datapaths require separate instruction/data memories
 - Or separate instruction/data caches
- RISC ISAs (including RISC-V) designed to avoid structural hazards
 - e.g. at most one memory access/instruction

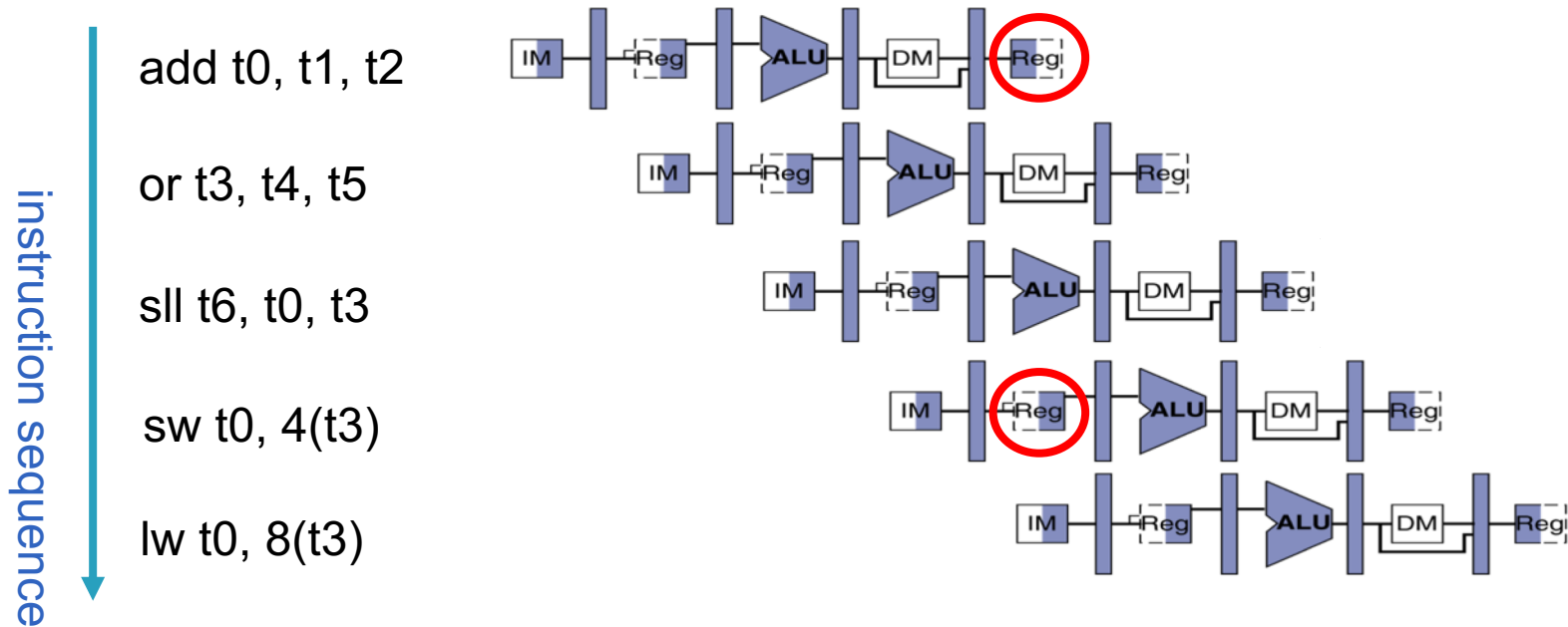
Agenda

- Pipelining
- Hazards
 - Structural
 - Data
 - R-type instructions
 - Load
 - Control



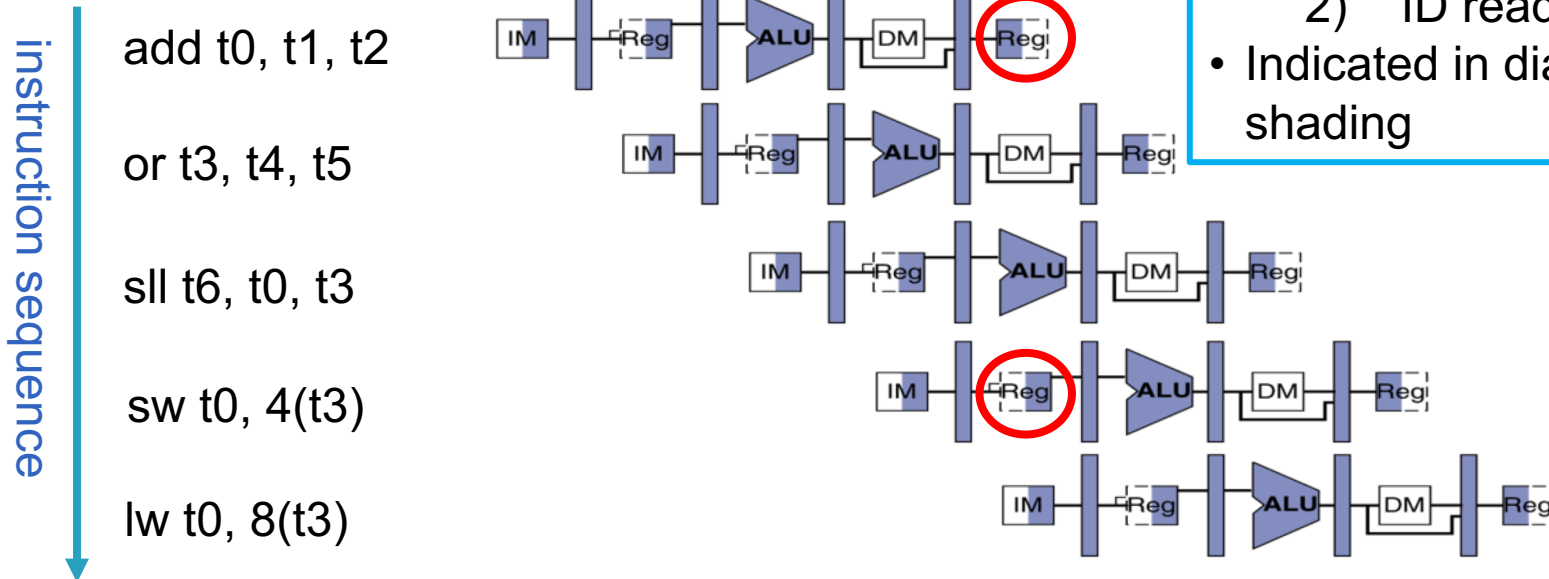
Data Hazard: Register Access

- Separate ports, but what if write to same value as read?
- Does `sw` in the example fetch the old or new value?



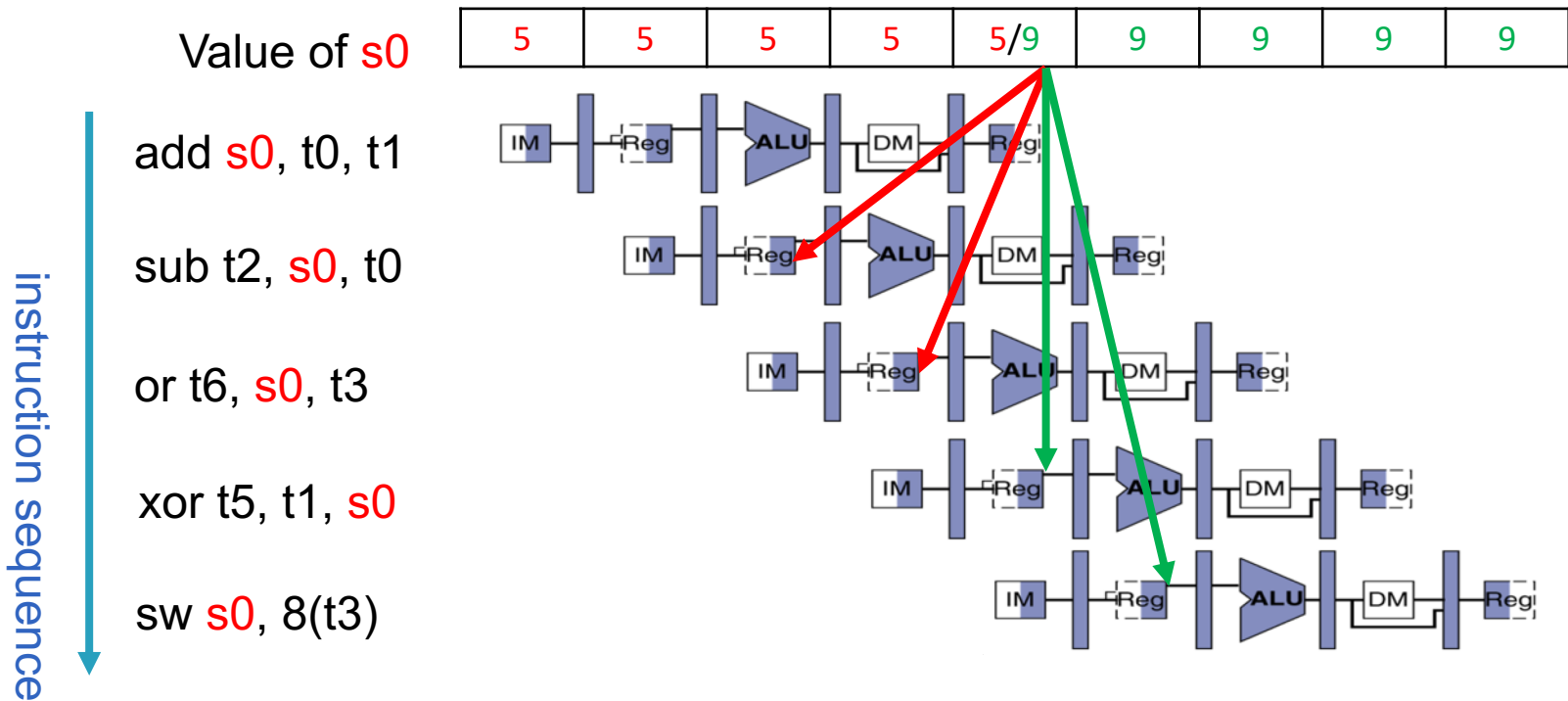
Register Access Policy

- Exploit high speed of register file (100 ps)
 - 1) WB updates value
 - 2) ID reads new value
- Indicated in diagram by shading



Might not always be possible to write then read in same cycle, especially in high-frequency designs. Always check assumptions!

Data Hazard: ALU Result

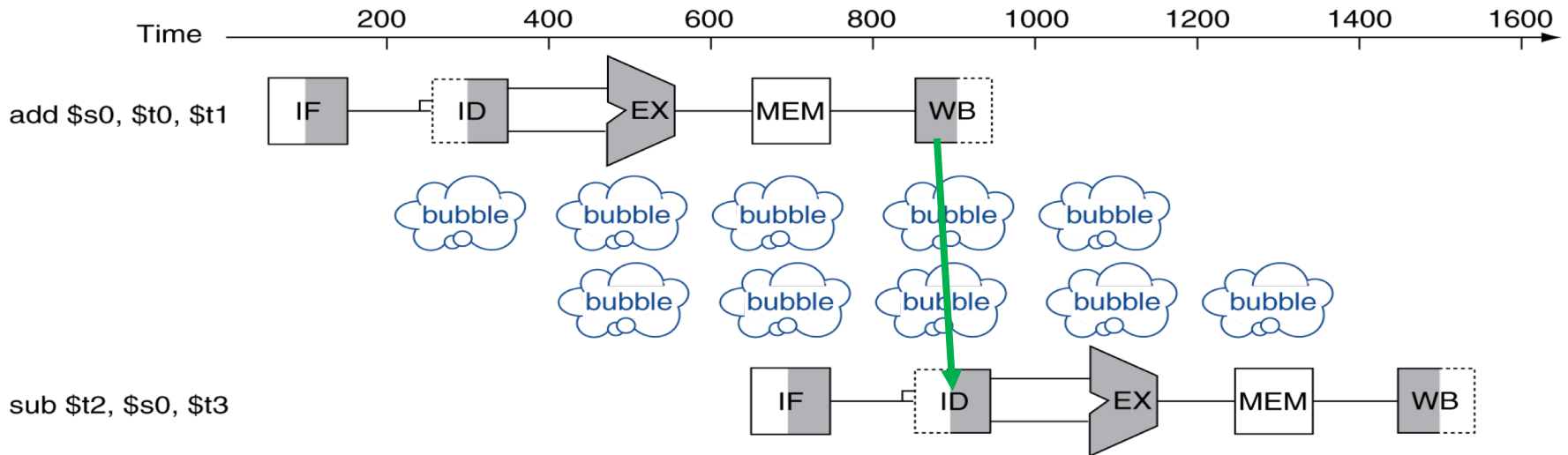


Without some fix, **sub** and **or** will calculate wrong result!

Solution 1: Stalling

- Problem: Instruction depends on result from previous instruction

- add **s0**, t0, t1
- sub t2, **s0**, t3



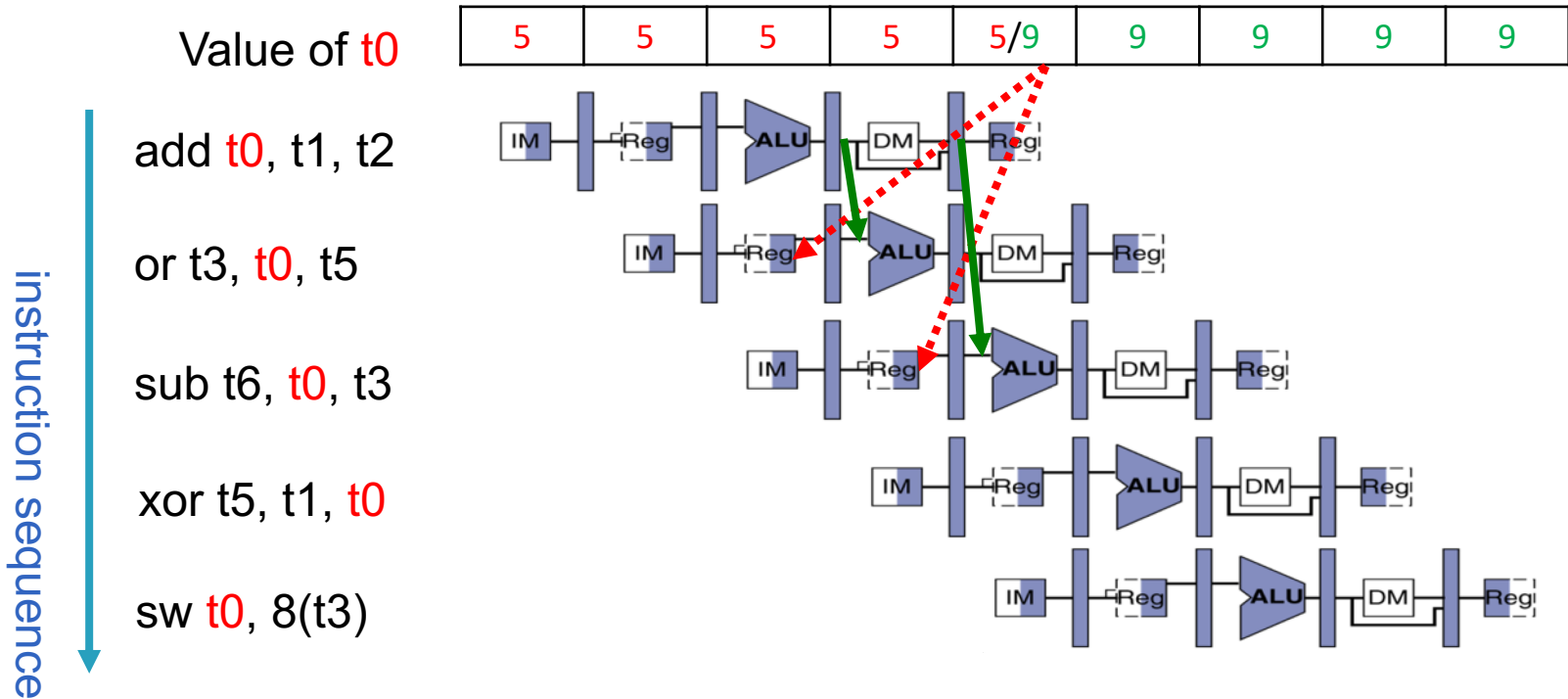
- Bubble:

- effectively NOP: affected pipeline stages do “nothing”

Stalls and Performance

- Stalls reduce performance
 - But stalls are required to get correct results
- Compiler can arrange code or insert NOPs (writes to register x0) to avoid hazards and stalls
 - Requires knowledge of the pipeline structure

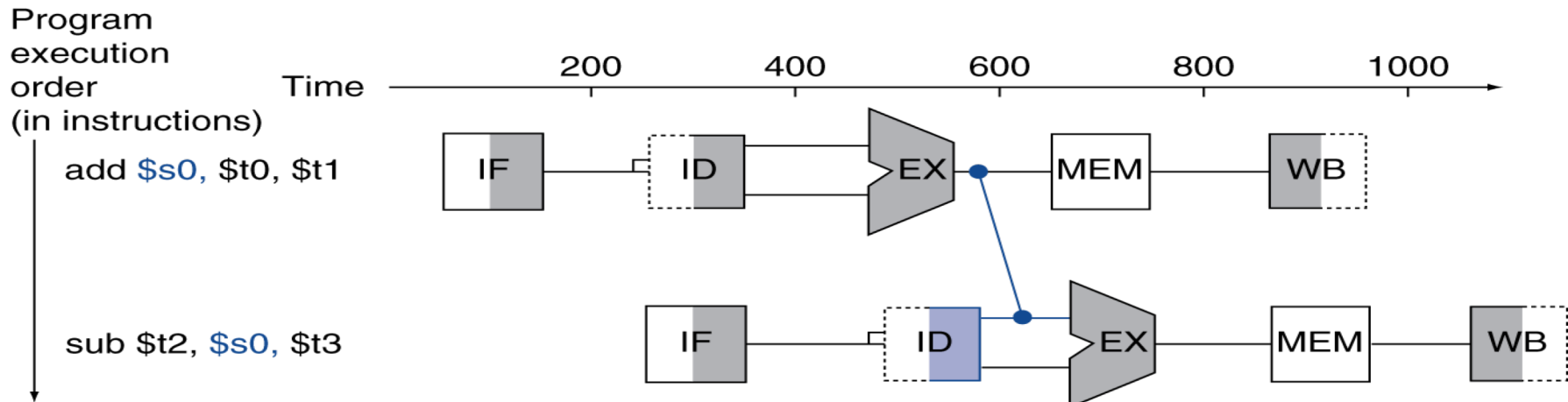
Solution 2: Forwarding



Forwarding: grab operand from pipeline stage, rather than register file

Forwarding (aka Bypassing)

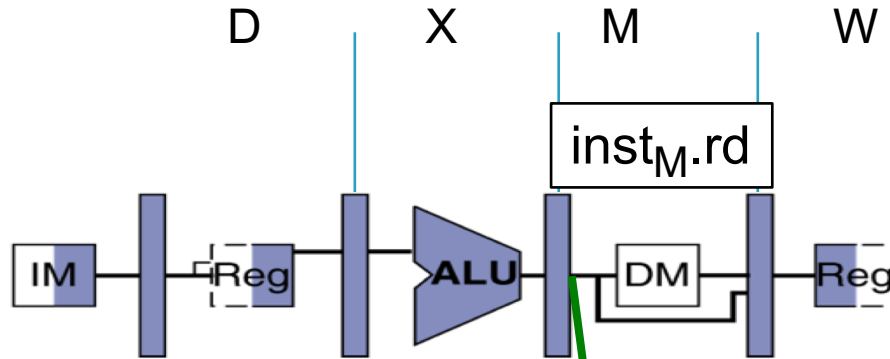
- Use result when it is computed
 - Don't wait for it to be stored in a register
 - Requires extra connections in the datapath



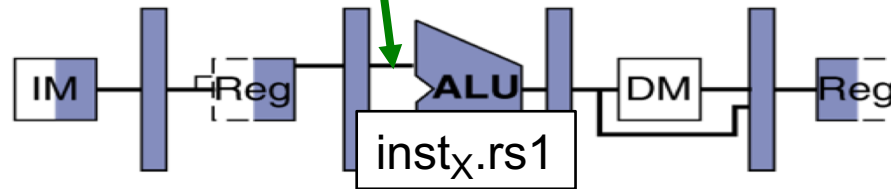
Detect Need for Forwarding (example)

*Compare destination of older instructions in pipeline with sources of new instruction in decode stage.
Must ignore writes to x0!*

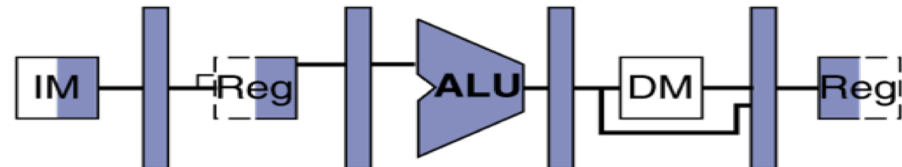
add t0, t1, t2



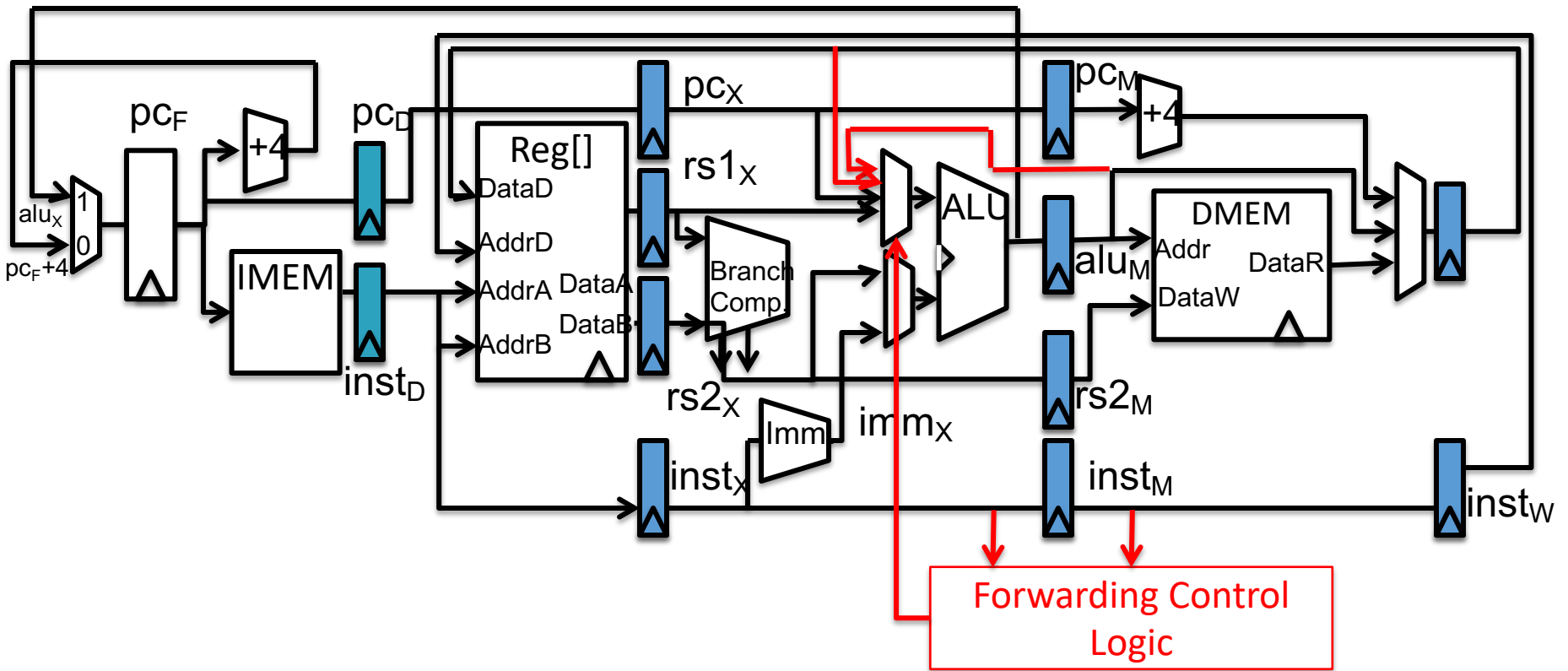
or t3, t0, t5



sub t6, t0, t3



Forwarding Path



CS 110
Computer Architecture
Lecture 12:
Pipelining
Video 3: More Hazards

Instructors:

Sören Schwertfeger & Chundong Wang

<https://robotics.shanghaitech.edu.cn/courses/ca/20s/>

School of Information Science and Technology SIST

ShanghaiTech University

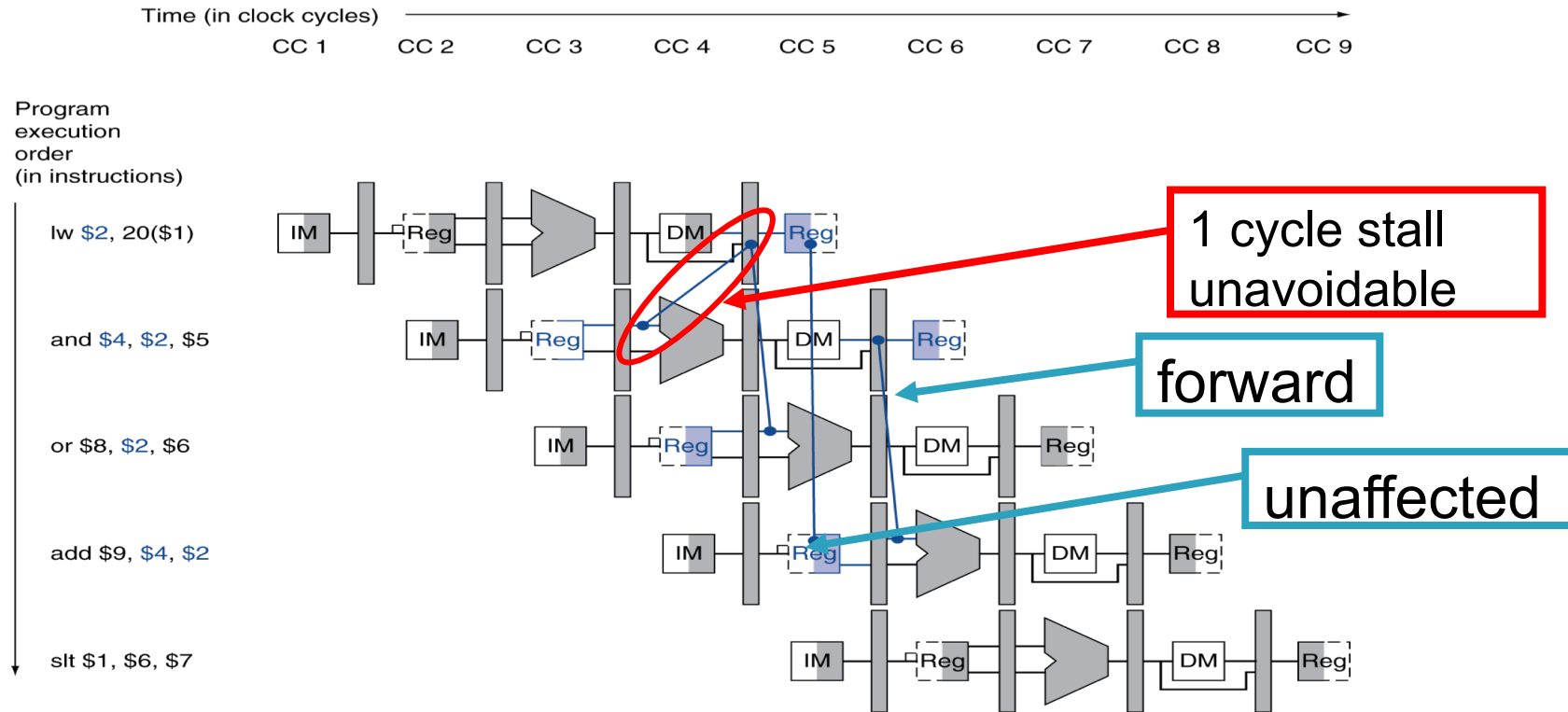
Slides based on UC Berkley's CS61C

Agenda

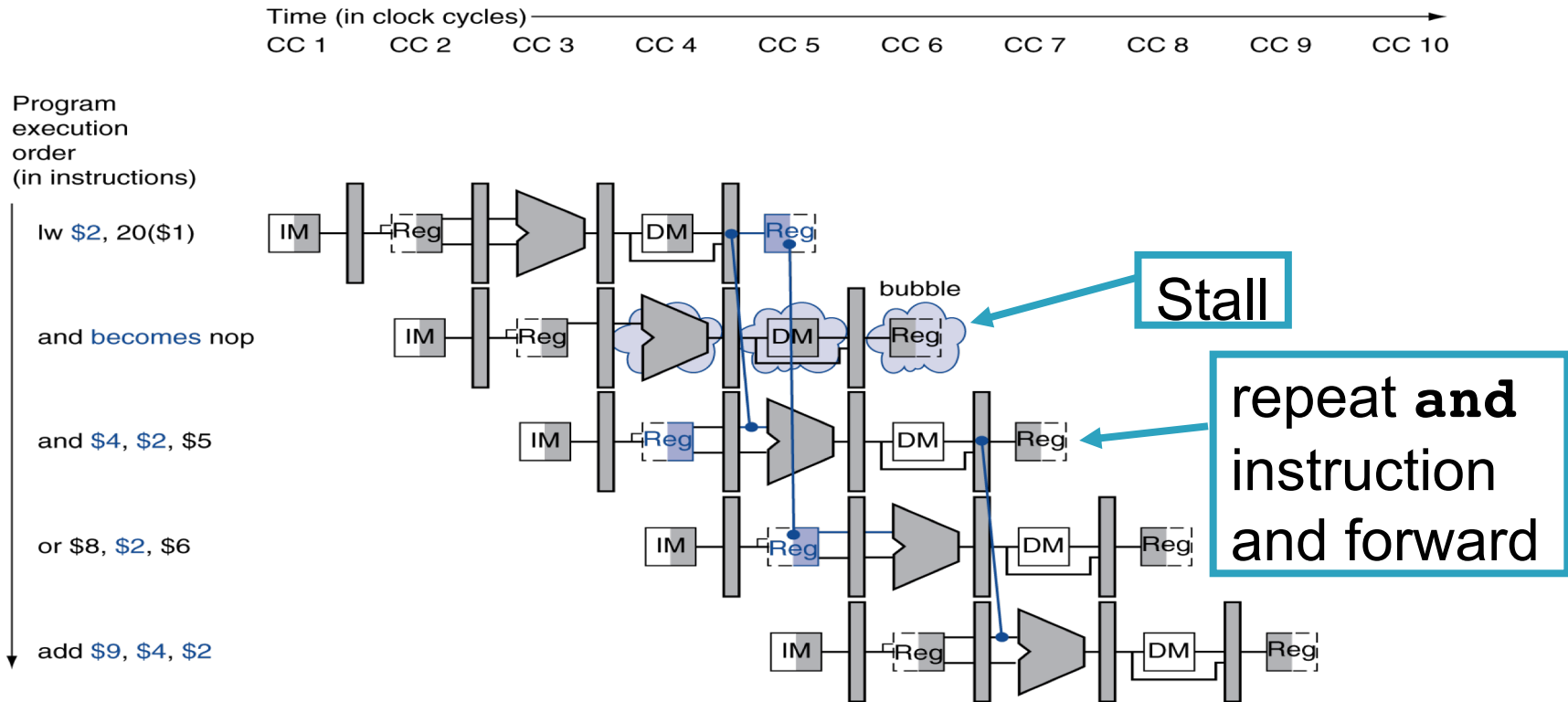
- Pipelining
- Hazards
 - Structural
 - Data
 - R-type instructions
 - Load
 - Control



Load Data Hazard



Stall Pipeline

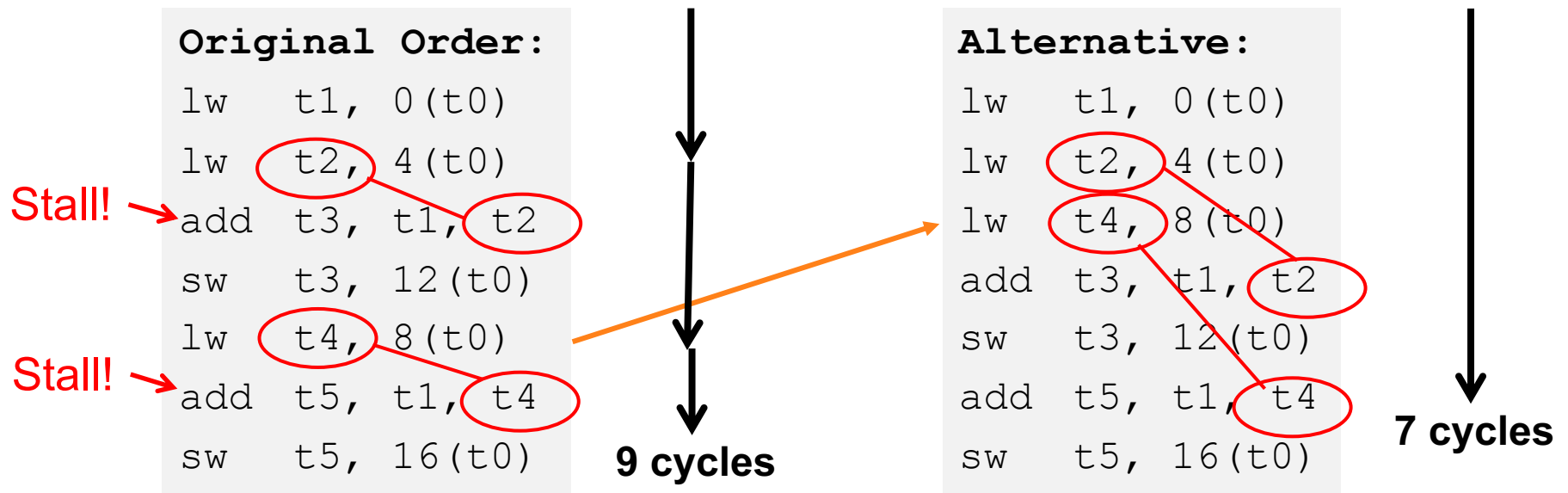


lw Data Hazard

- Slot after a load is called a *load delay slot*
 - If that instruction uses the result of the load, then the hardware will stall for one cycle
 - Equivalent to inserting an explicit **nop** in the slot
 - except the latter uses more code space
 - Performance loss
- Idea:
 - Put unrelated instruction into load delay slot
 - No performance loss!

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instr!
- RISC-V code for $\mathbf{A[3]=A[0]+A[1]}$; $\mathbf{A[4]=A[0]+A[2]}$



Agenda

- Pipelining
- Hazards
 - Structural
 - Data
 - R-type instructions
 - Load
 - Control
- Instruction-Level Parallelism

Control Hazards

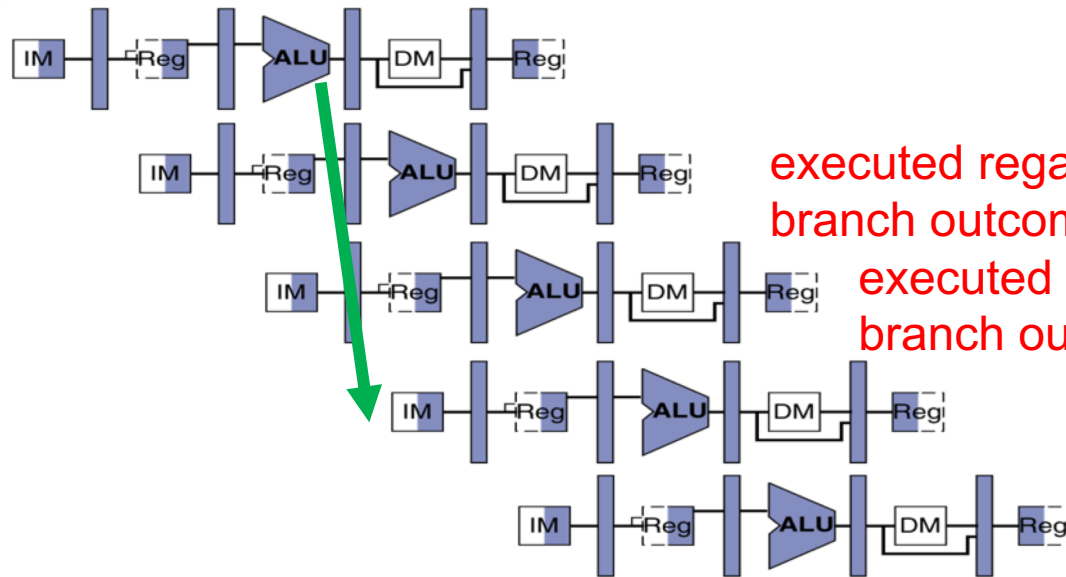
beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

xor t5, t1, s0

sw s0, 8(t3)



executed regardless of
branch outcome!

executed regardless of
branch outcome!!!

PC updated
reflecting branch
outcome

Observation

- If branch not taken, then instructions fetched sequentially after branch are correct
- If branch or jump taken, then need to flush incorrect instructions from pipeline by converting to NOPs

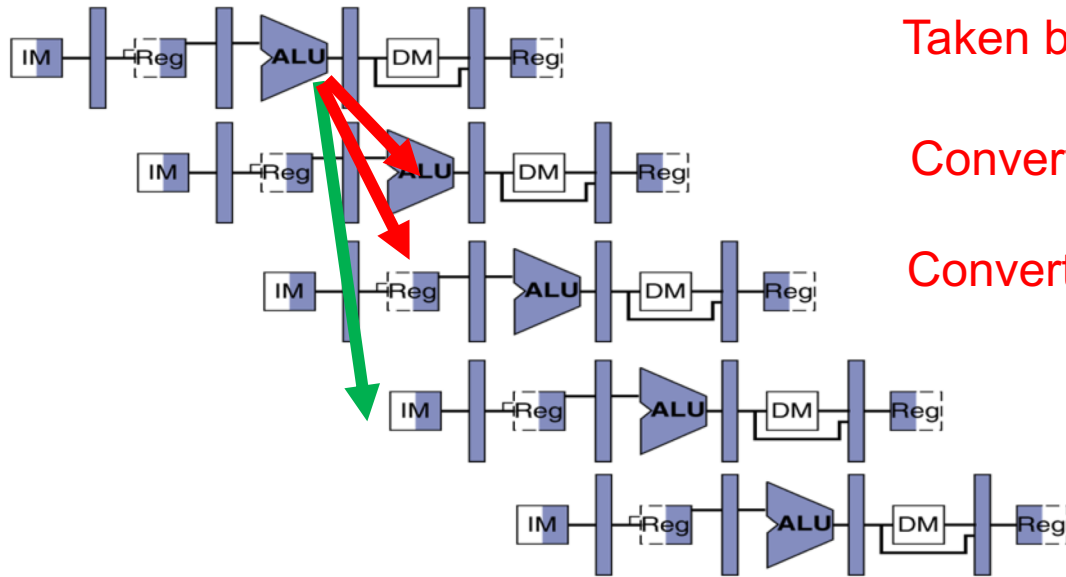
Kill Instructions after Branch if Taken

beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

label: xxxxxx



Taken branch

Convert to NOP

Convert to NOP

PC updated
reflecting branch
outcome

Reducing Branch Penalties

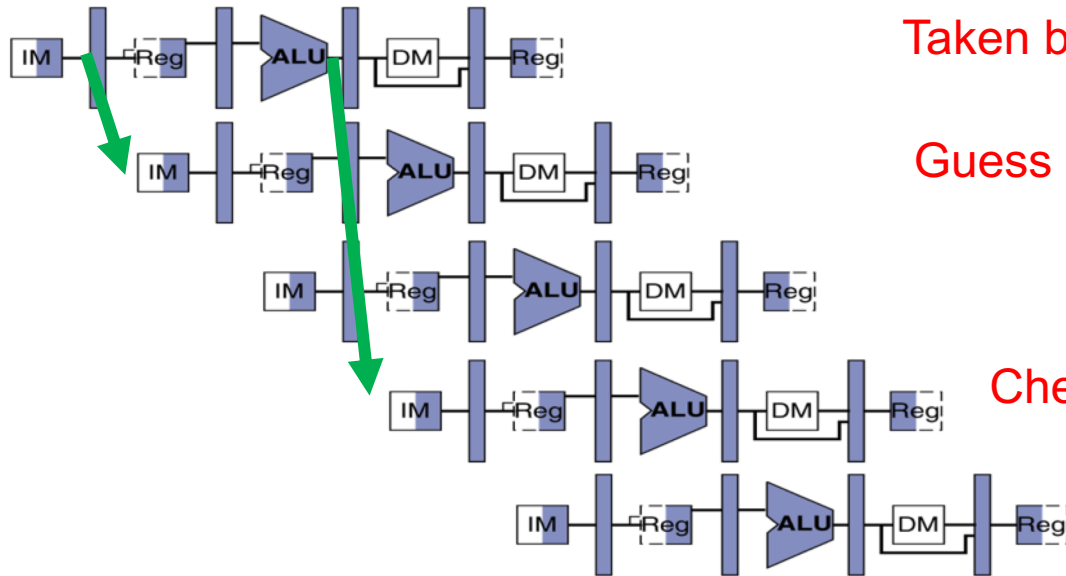
- Every taken branch in simple pipeline costs 2 dead cycles
- To improve performance, use “branch prediction” to guess which way branch will go earlier in pipeline
- Only flush pipeline if branch prediction was incorrect

Branch Prediction

beq t0, t1, label

label:

.....



Taken branch

Guess next PC!

Check guess correct

In Conclusion

- Pipelining increases throughput by overlapping execution of multiple instructions
- All pipeline stages have same duration
 - Choose partition that accommodates this constraint
- Hazards potentially limit performance
 - Maximizing performance requires programmer/compiler assistance

Quiz

Piazza: "Video Lecture 12 Pipelining Poll"

Question: For each code sequences below, choose one of the statements below:

I:

```
addi t1, t0, 1
addi t2, t0, 2
addi t3, t0, 2
addi t3, t0, 4
addi t5, t1, 5
```

II:

```
add t1, t0, t0
addi t2, t0, 5
addi t4, t1, 5
```

III:

```
lw t0, 0(t0)
add t1, t0, t0
```

- A) No stalls as is
- B) No stalls with forwarding
- C) Must stall

Check the numbers with are correct:

I: A: 1

I: B: 2

I: C: 3

II: A: 4

II: B: 5

II: C: 6

III: A: 7

III: B: 8

III: C: 9