CS 110 Computer Architecture Lecture 12: *Pipelining*

Instructors:
Sören Schwertfeger & Chundong Wang

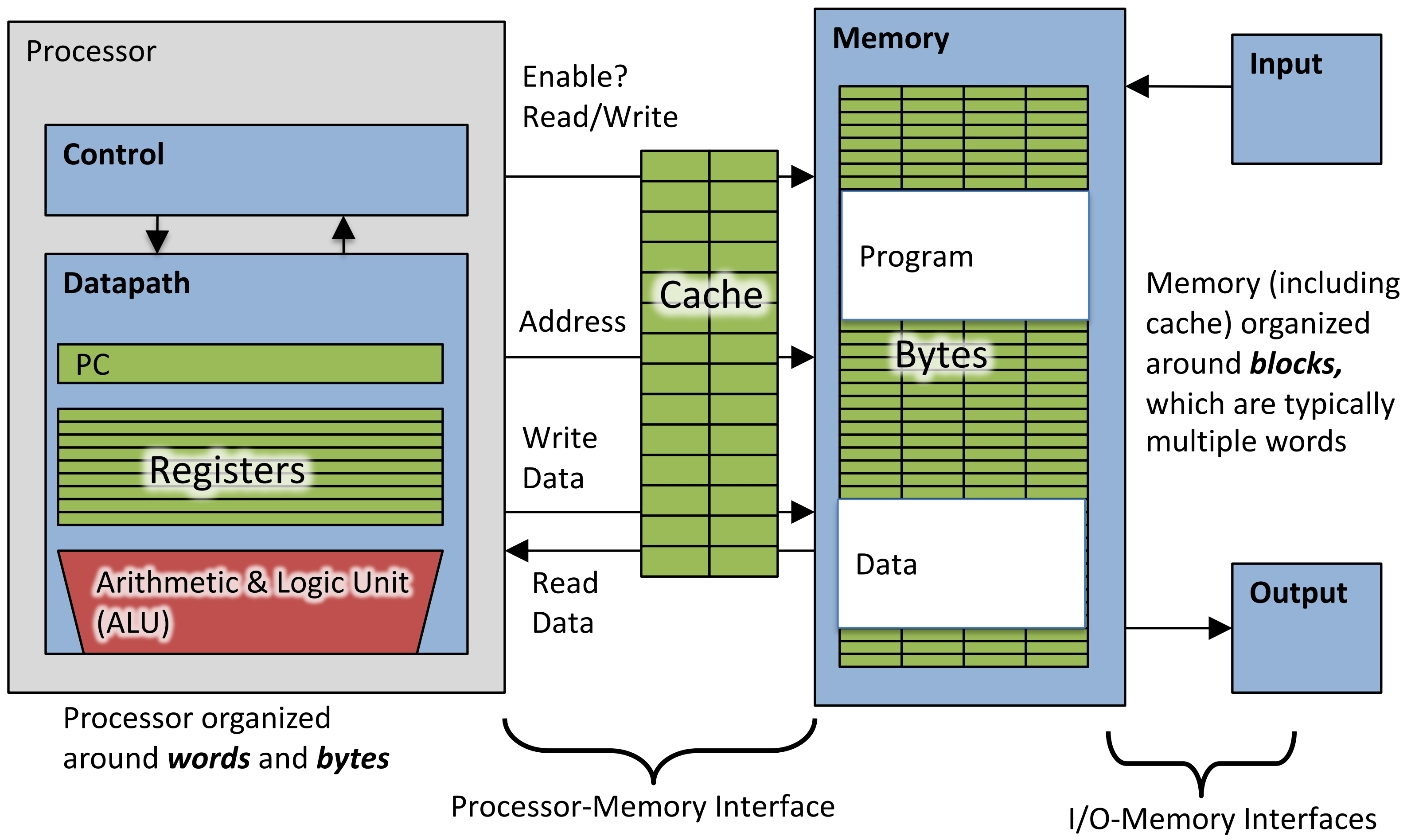
https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

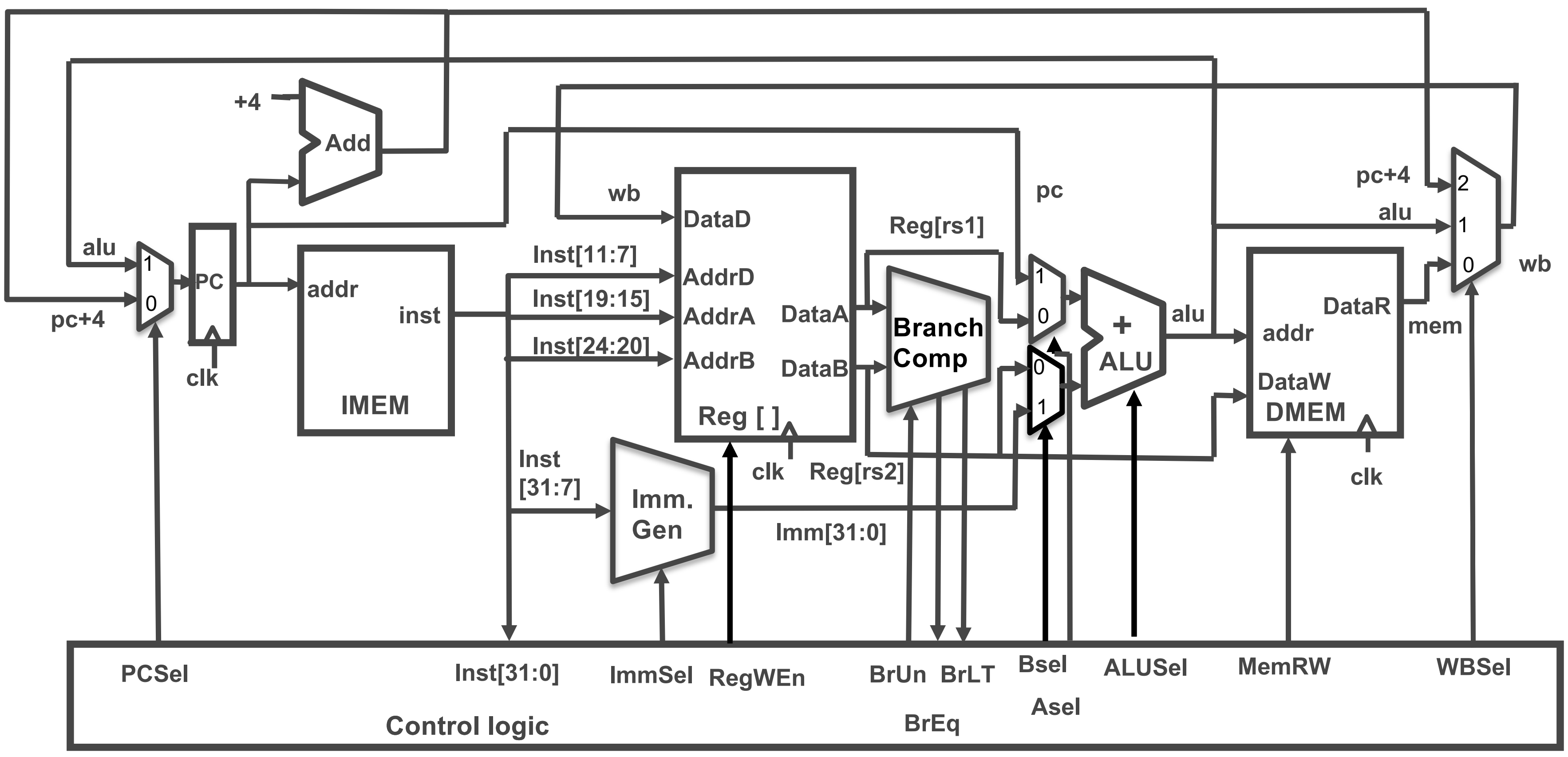
ShanghaiTech University

Slides based on UC Berkley's CS61C

Our Single-Core Computer


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Enable? Read/Write
Address
Write Data
Read Data
Cache
Memory
Program
Bytes
Data
Input
Output
Memory (including cache) organized around ***blocks,*** which are typically multiple words
Processor organized around ***words*** and ***bytes***
Processor-Memory Interface
I/O-Memory Interfaces

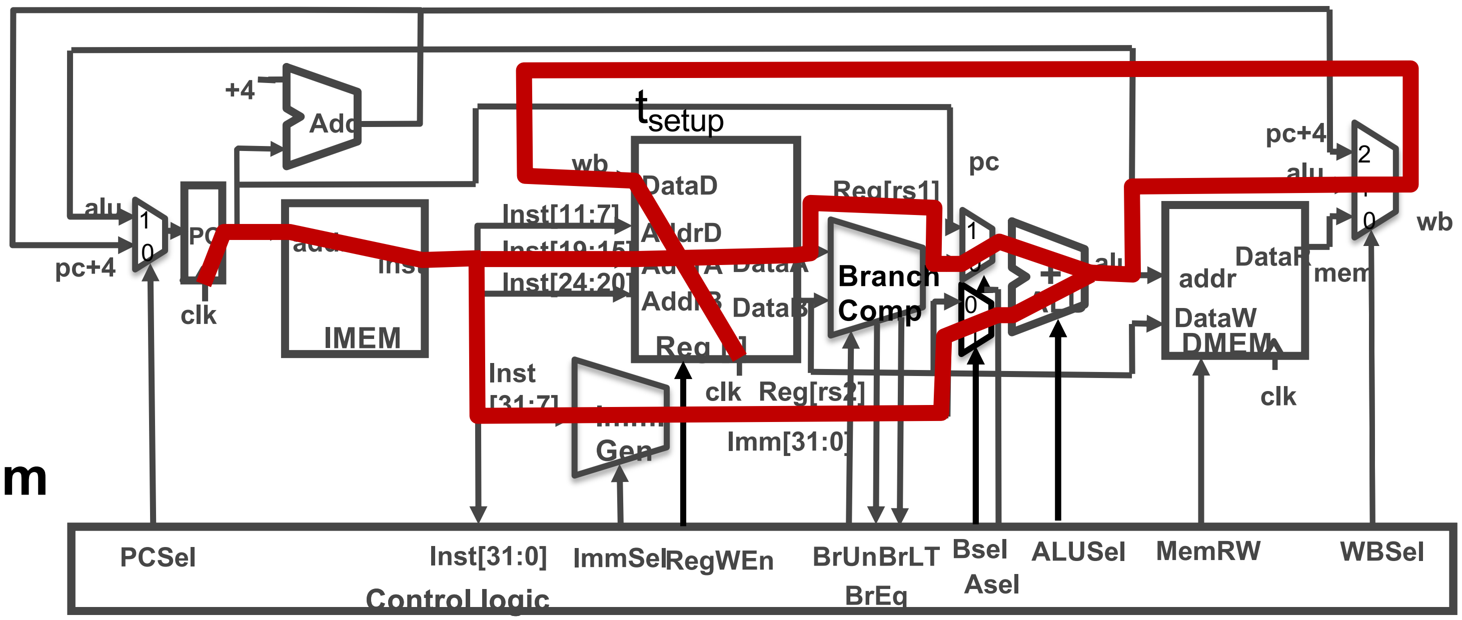
Complete RV32I Datapath!



Critical Path

Critical path for xori

R[rd] = R[rs1]+imm



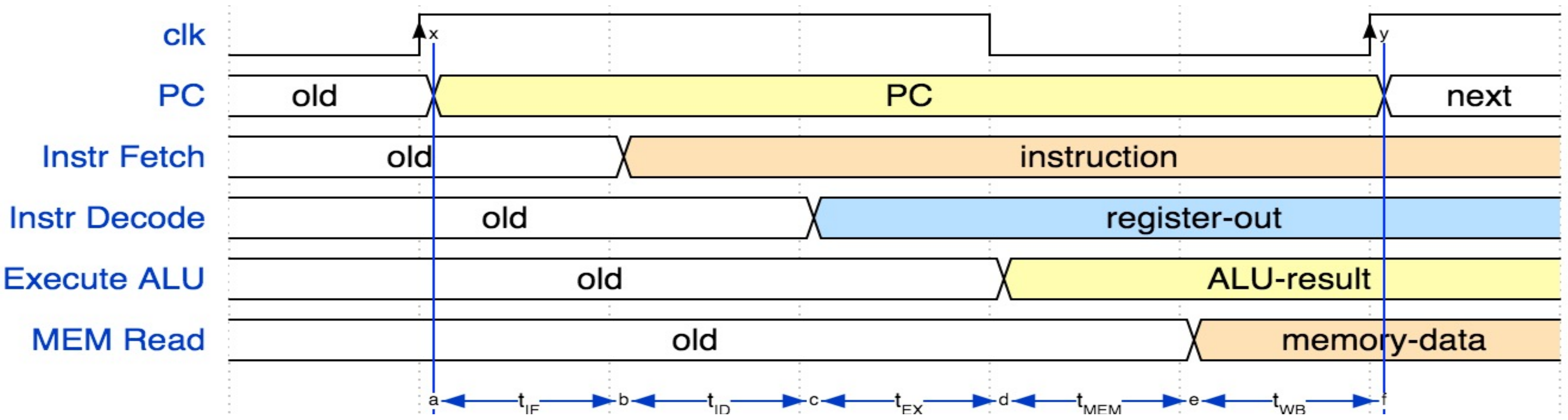
A. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2*t_{mux} + t_{Setup}$

B. $t_{clk\text{-}q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$

C. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3*t_{mux} + t_{DMEM} + t_{Setup}$

D. None of the above

Instruction Timing



IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**

Instruction Timing

Instr	IF = 200ps	ID = 100ps	ALU = 200ps	MEM=200ps	WB = 100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X		X	600ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

- Maximum clock frequency
  - f_{max} = 1/800ps = 1.25 GHz
- Most blocks idle most of the time
  - E.g. $f_{max,ALU}$ = 1/200ps = 5 GHz!

Performance

- “Our” RISC-V executes instructions at 1.25 GHz
  - 1 instruction every 800 ps
- Can we improve its performance?
  - What do we mean with this statement?
  - Not so obvious:
    - Quicker response time, so one job finishes faster?
    - More jobs per unit time (e.g. web server returning pages)?
    - Longer battery life?

Transportation Analogy

	Sports Car	Bus
Passenger Capacity	2	50
Travel Speed	250 km/h	100 km/h
Fuel consumption	20 l/100km	20 l/100km

Schwerin => Berlin trip: 200 km

=>

Transportation Analogy

	Sports Car	Bus
Passenger Capacity	2	50
Travel Speed	250 km/h	100 km/h
Fuel consumption	20 l/100km	20 l/100km

Schwerin => Berlin trip: 200 km

	Sports Car	Bus
Travel Time	48 min	120 min
Time for 100 passengers	40 h	4 h
Fuel per passenger	2000 l	80 l

Computer Analogy

Transportation	Computer
Travel Time	Program execution time (**latency**) e.g. time to update display
Time for 100 passengers	**Throughput**: e.g. number of server requests handled per hour
Fuel per passenger	Energy per task*: e.g.: - how many movies can you watch per battery charge - energy bill for datacenter

* Note: power is not a good measure, since low-power CPU might run for a long time to complete one task consuming more energy than faster computer running at higher power for a shorter time

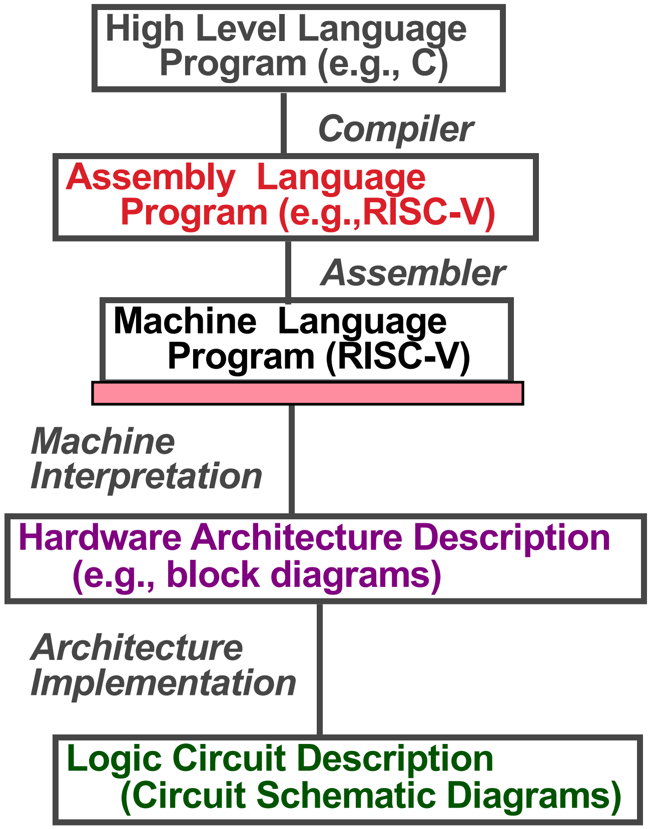
This Lecture:

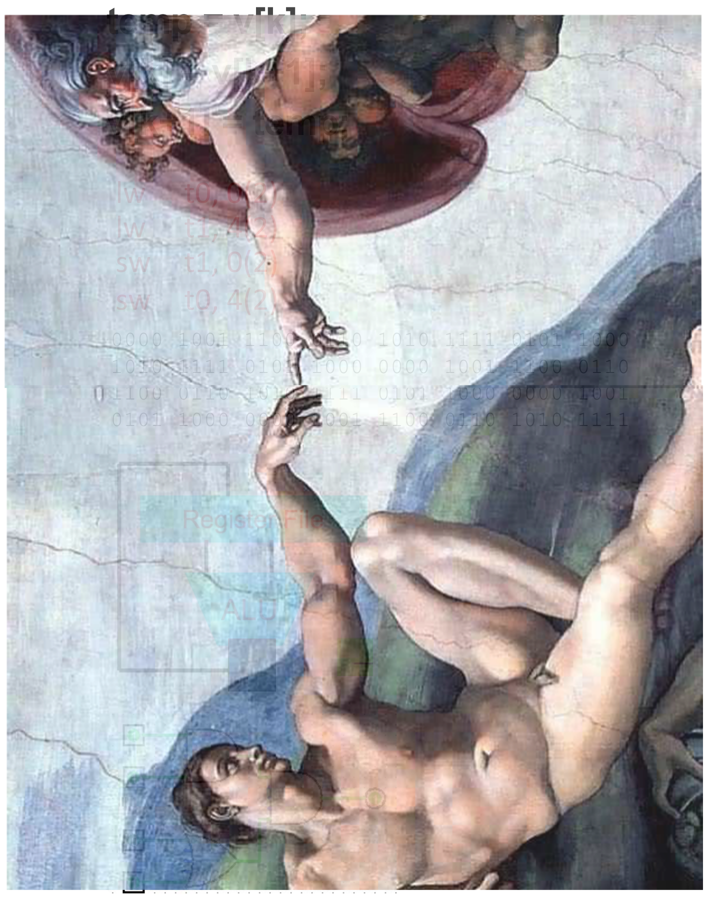
- Improve performance through pipelining:
  - One stage per clock cycle!
  - Need 5 clock cycles to complete one instruction
  - Clock cycles much shorter now

  => Higher throughput ☺

  => Higher latency ☹

Call home, we’ve made HW/SW contact!

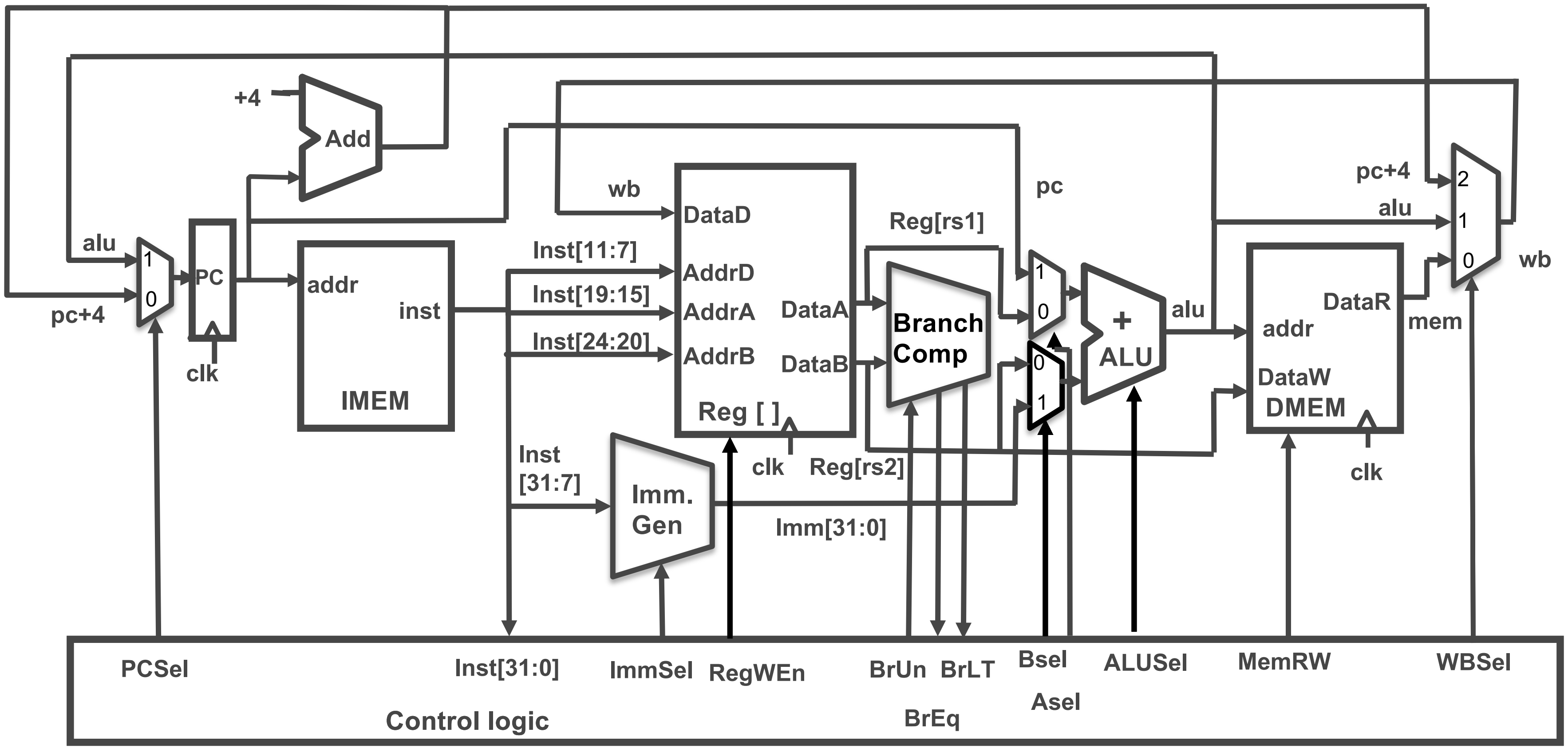

High Level Language Program (e.g., C)
Compiler
Assembly Language Program (e.g.,RISC-V)
Assembler
Machine Language Program (RISC-V)
Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)
Architecture Implementation
Logic Circuit Description (Circuit Schematic Diagrams)



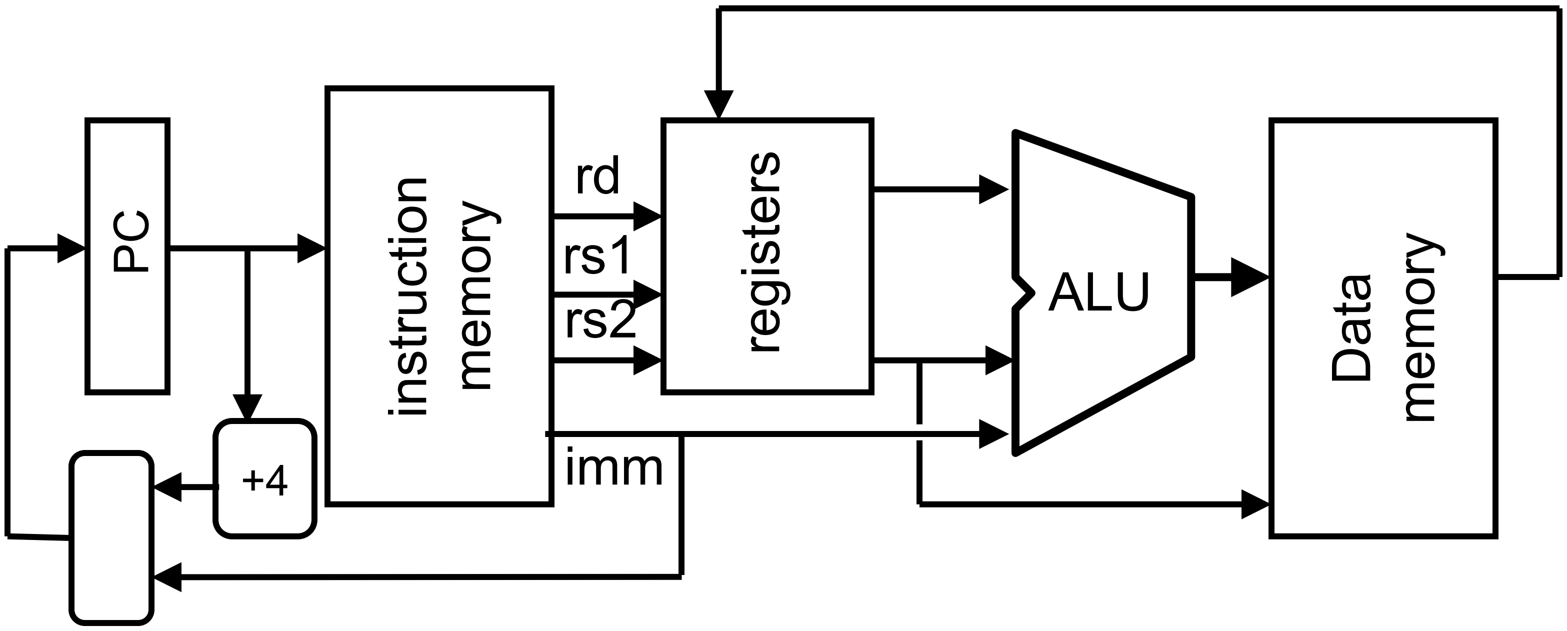
Agenda

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

Complete Single-Cycle RV32I Datapath!



Stages of Execution on Datapath



1. Instruction Fetch

2. Decode/ Register Read

3. Execute

4. Memory

5. Register Write

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/Period (seconds/cycle)

Single Cycle Performance

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

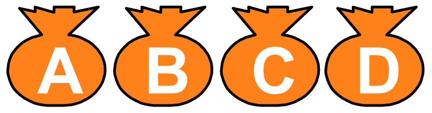
Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

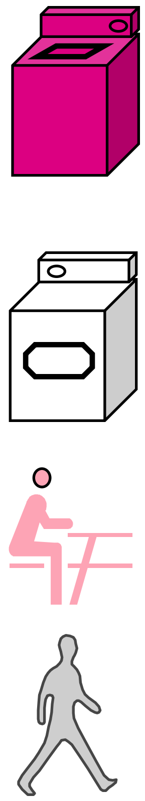
- What can we do to improve clock rate?
- Will this improve performance as well?

  Want increased clock rate to mean faster programs

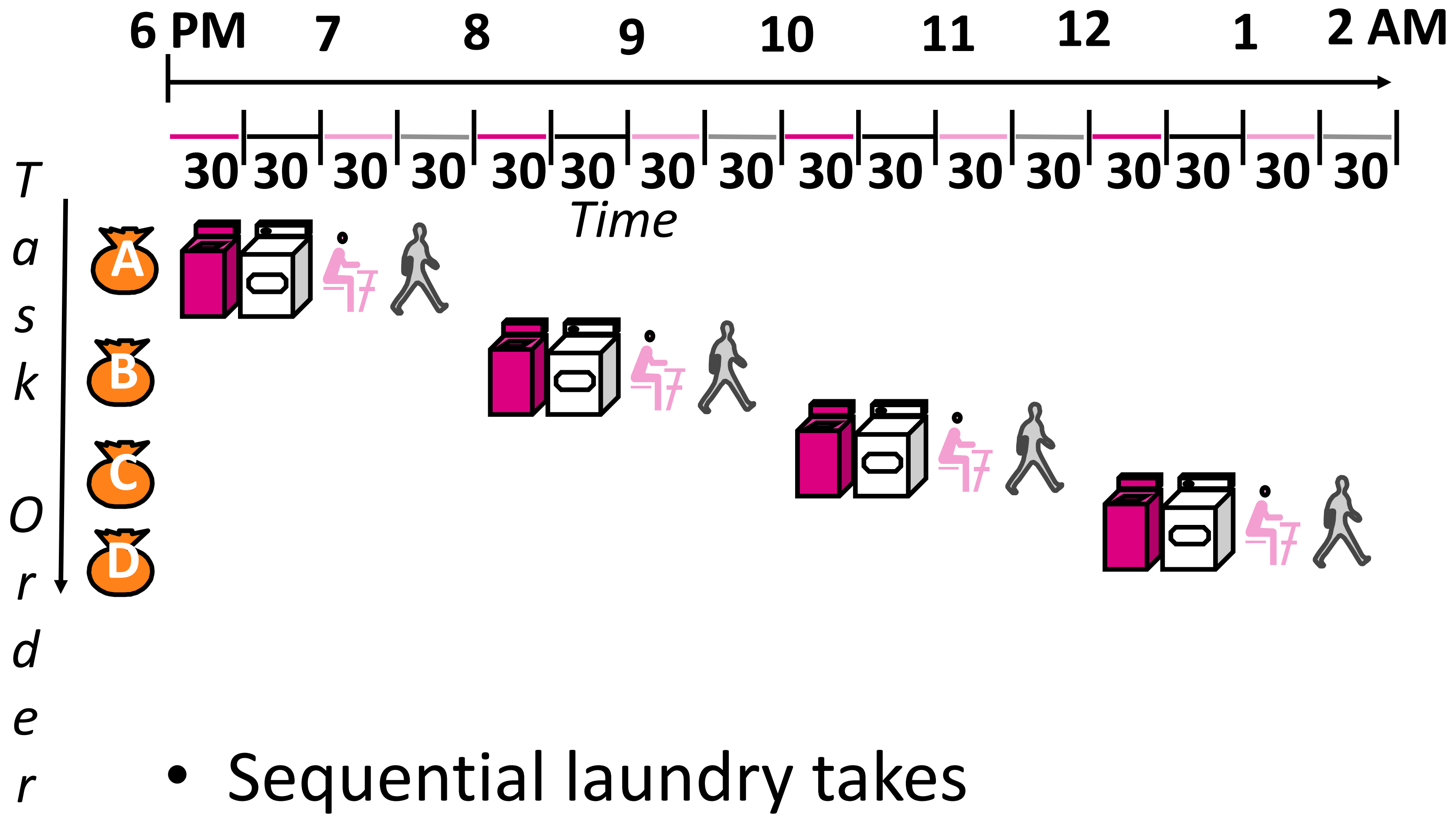
Gotta Do Laundry

- 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



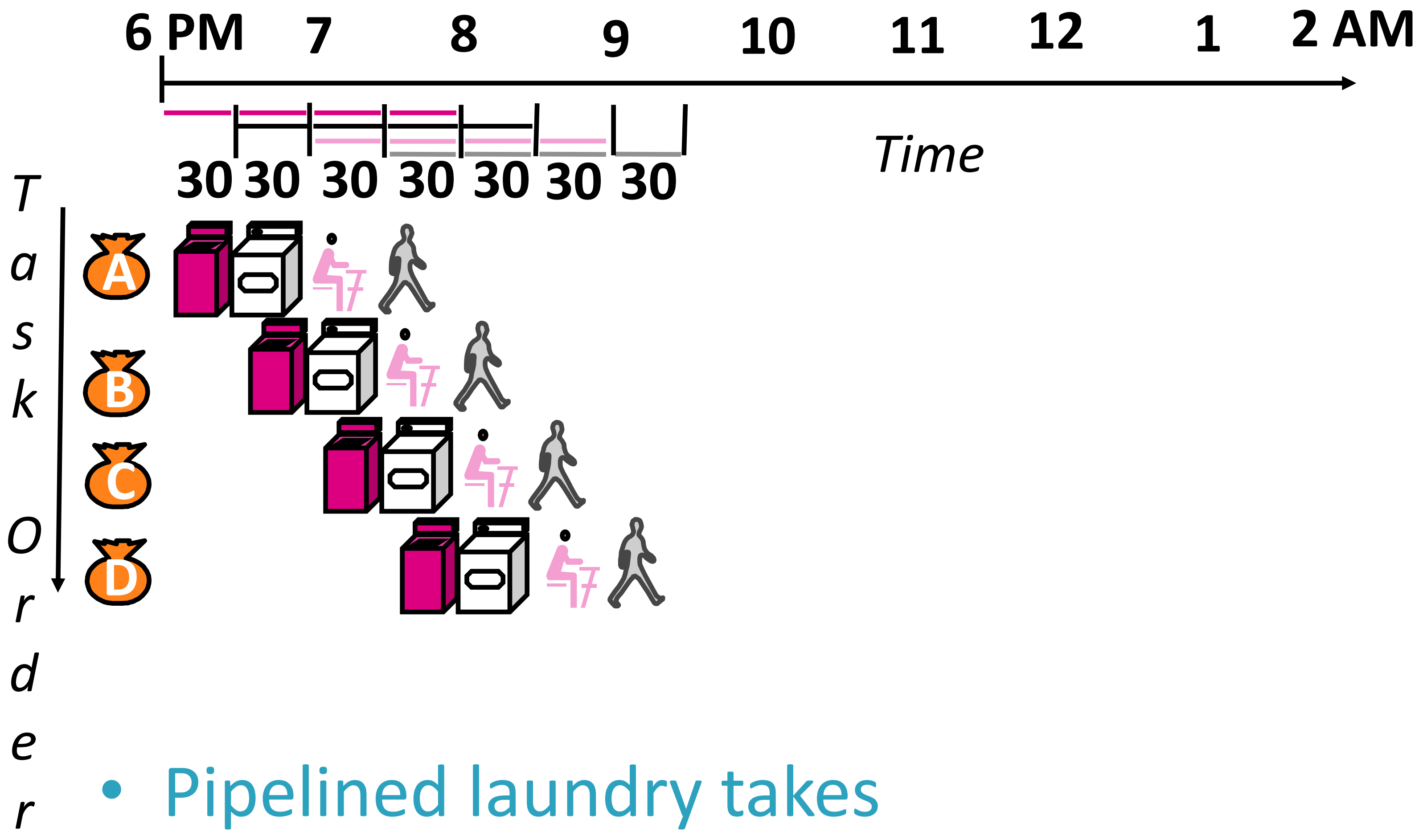


Sequential Laundry



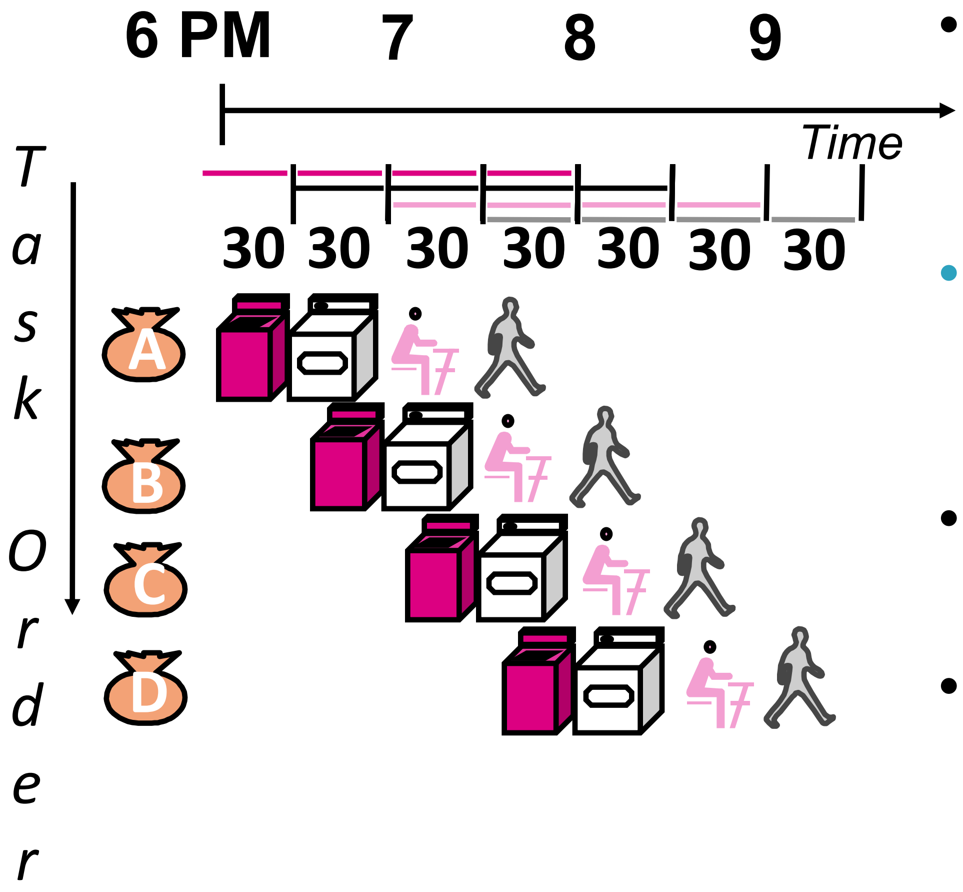
- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



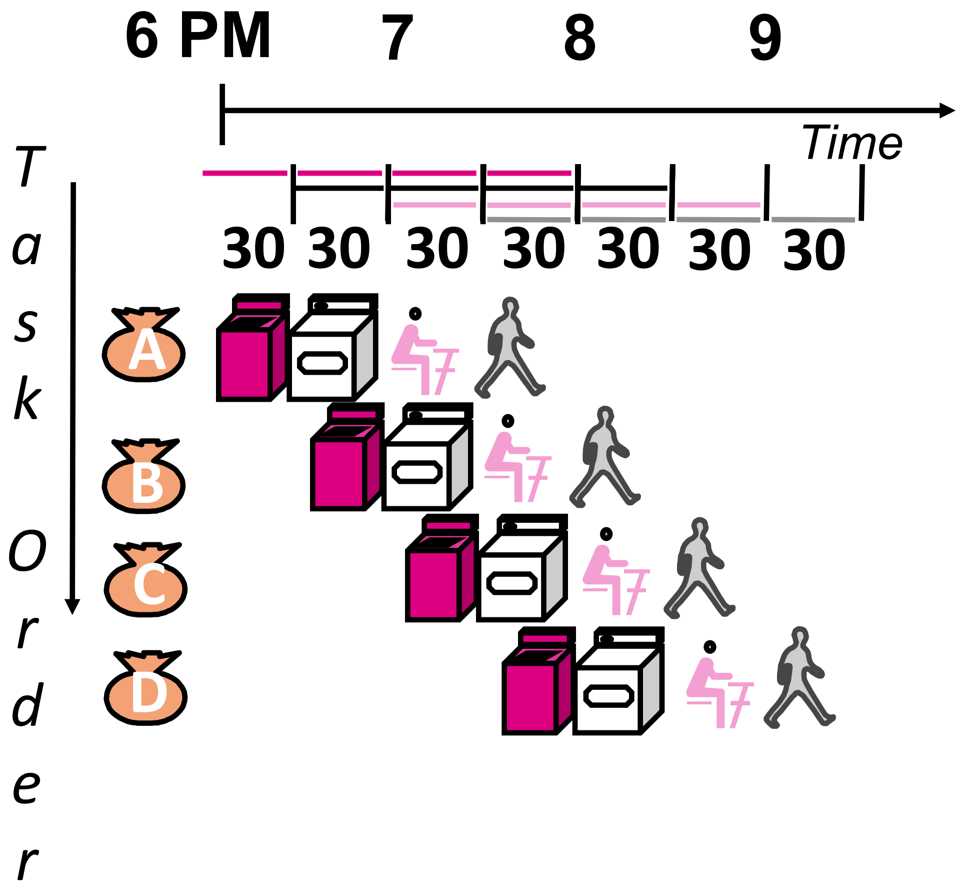
- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons (1/2)



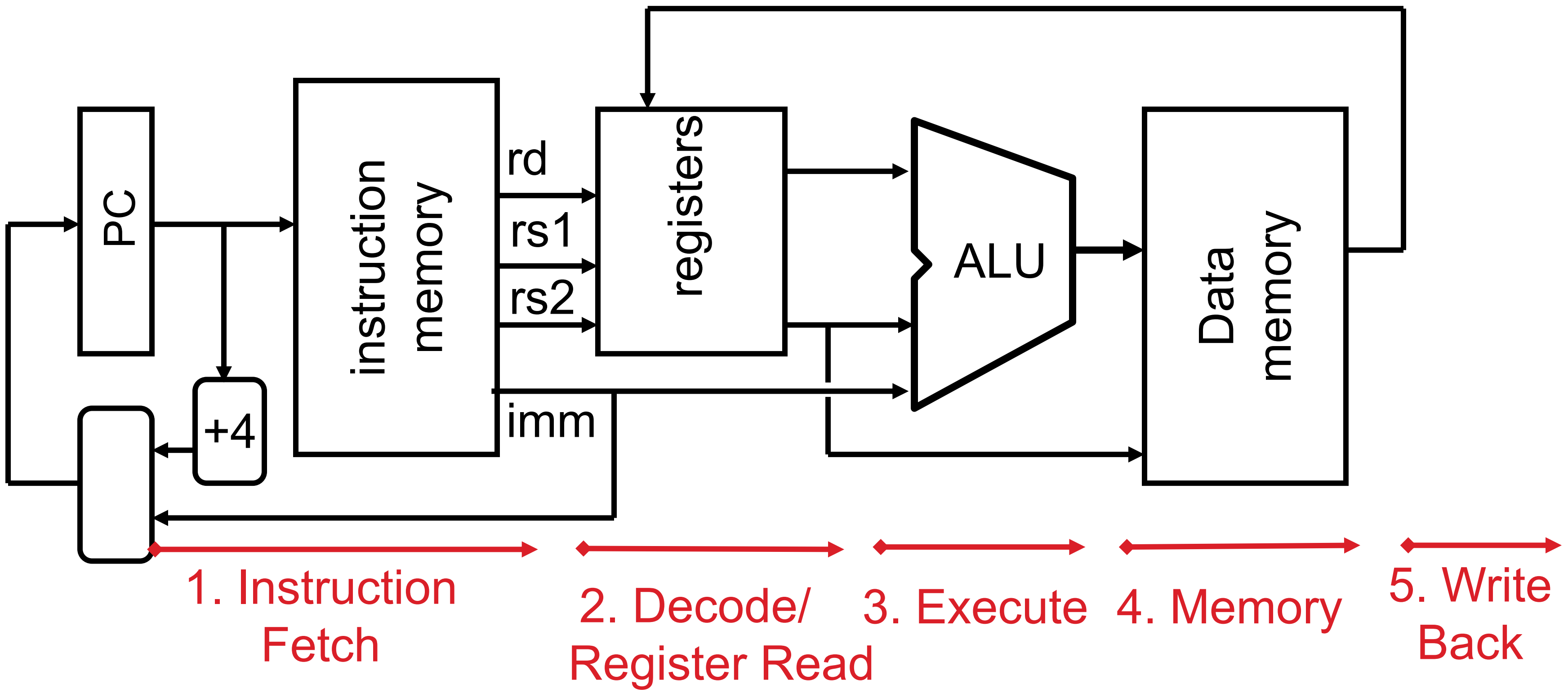
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Time to "fill" pipeline and time to "drain" it reduces speedup

Pipelining Lessons (2/2)

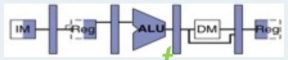


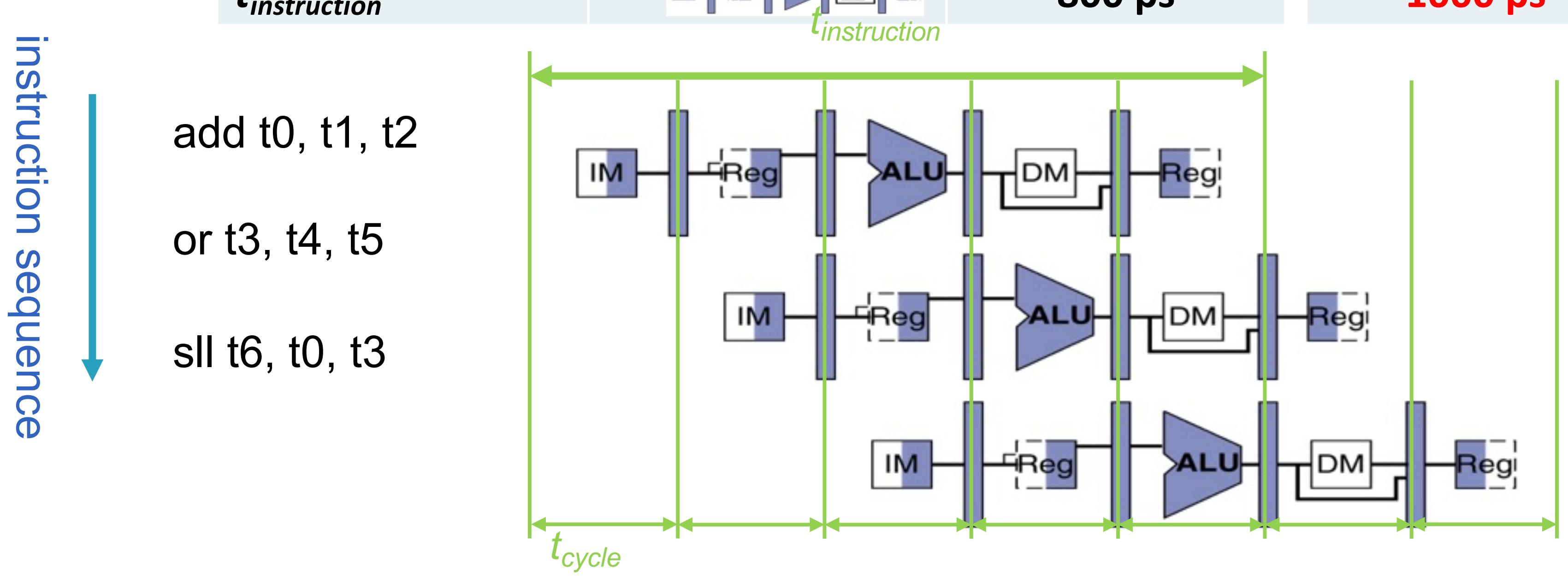
- Suppose new Dryer takes 20 minutes, new Folder takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

Single Cycle Datapath

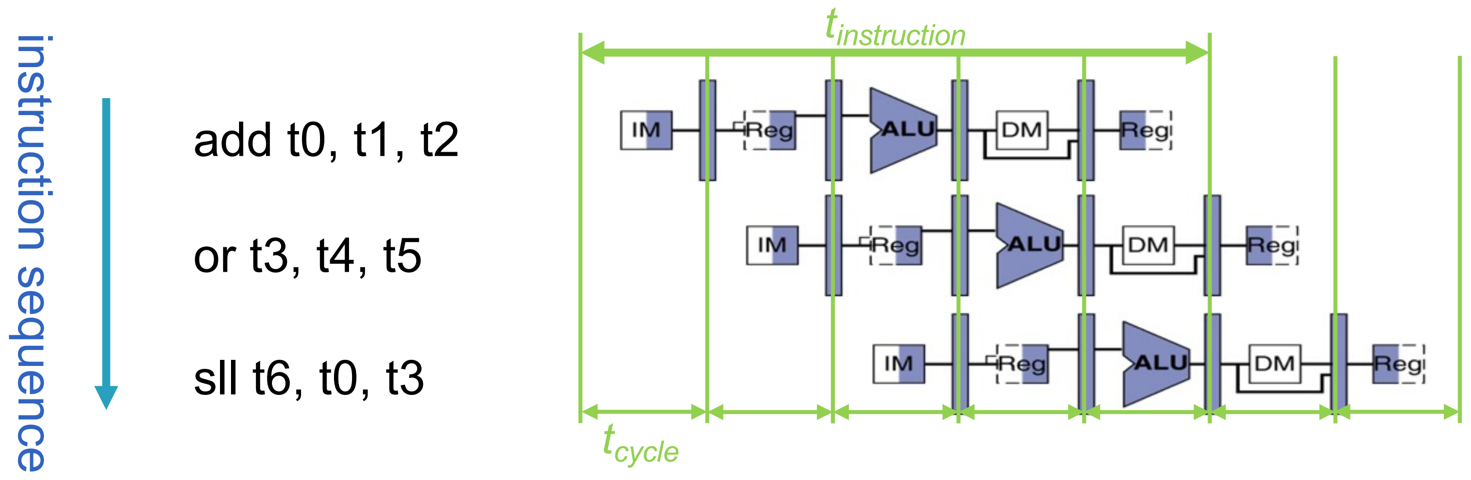


Pipelining with RISC-V

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**



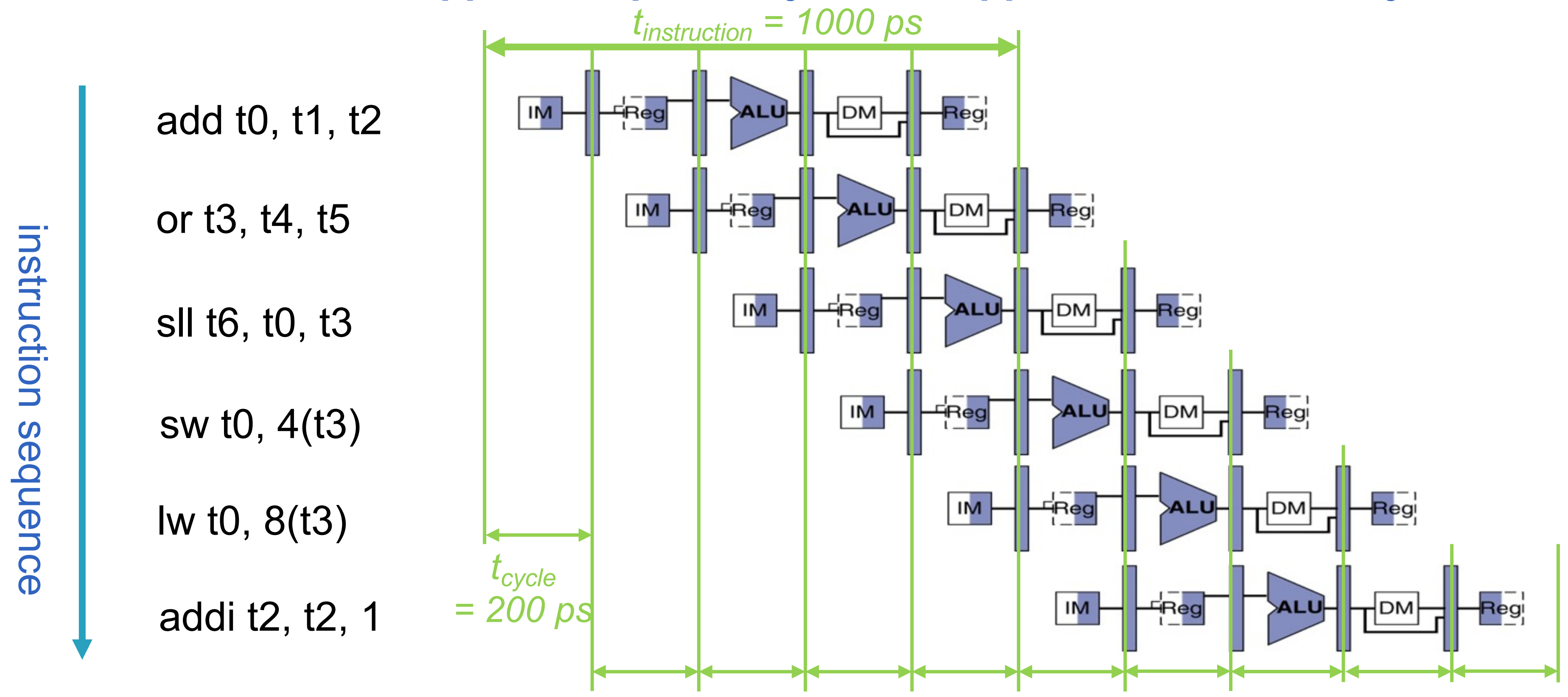
Pipelining with RISC-V



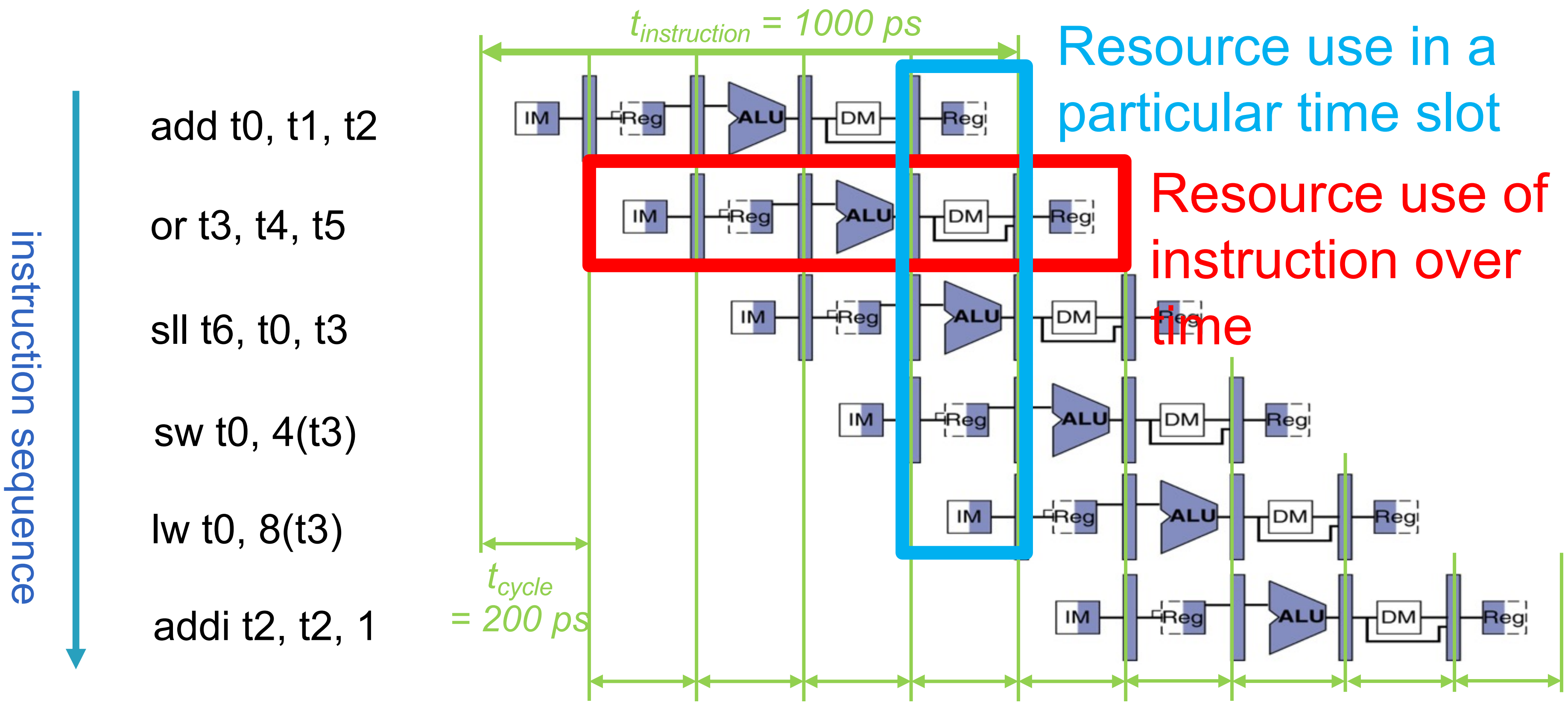
	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
CPI (Cycles Per Instruction)	~1 (ideal)	~1 (ideal), >1 (actual)
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps =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

Sequential vs Simultaneous

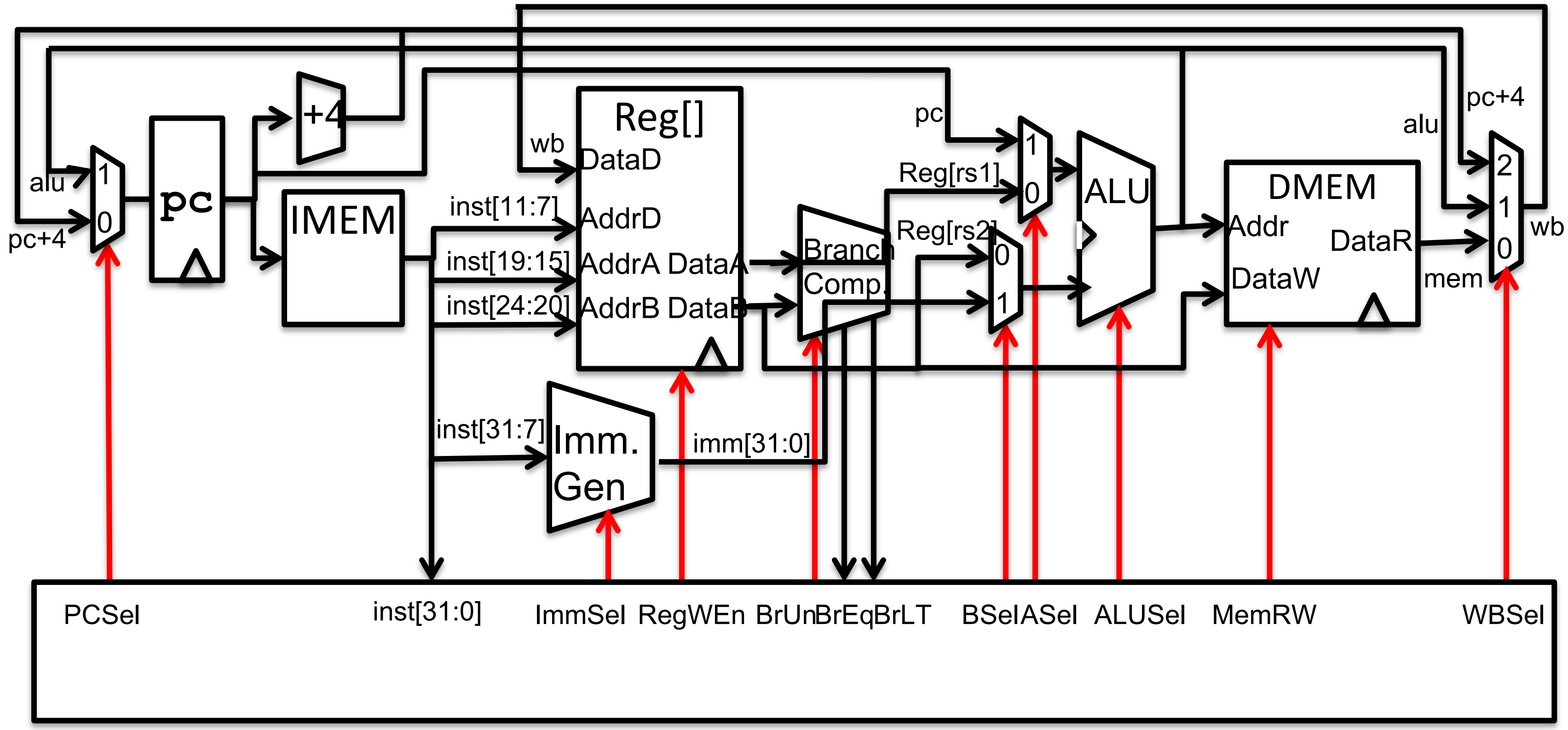
What happens sequentially, what happens simultaneously?



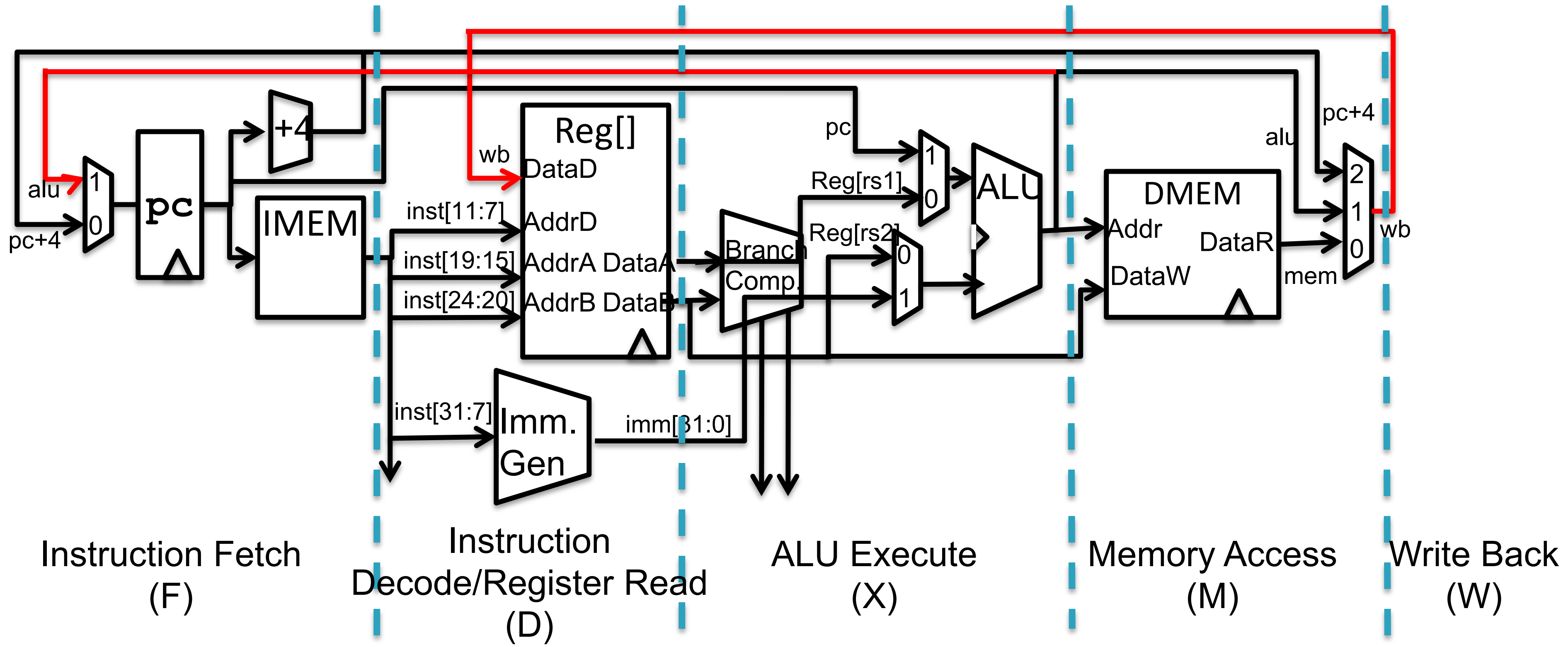
RISC-V Pipeline



Single-Cycle RISC-V RV32I Datapath

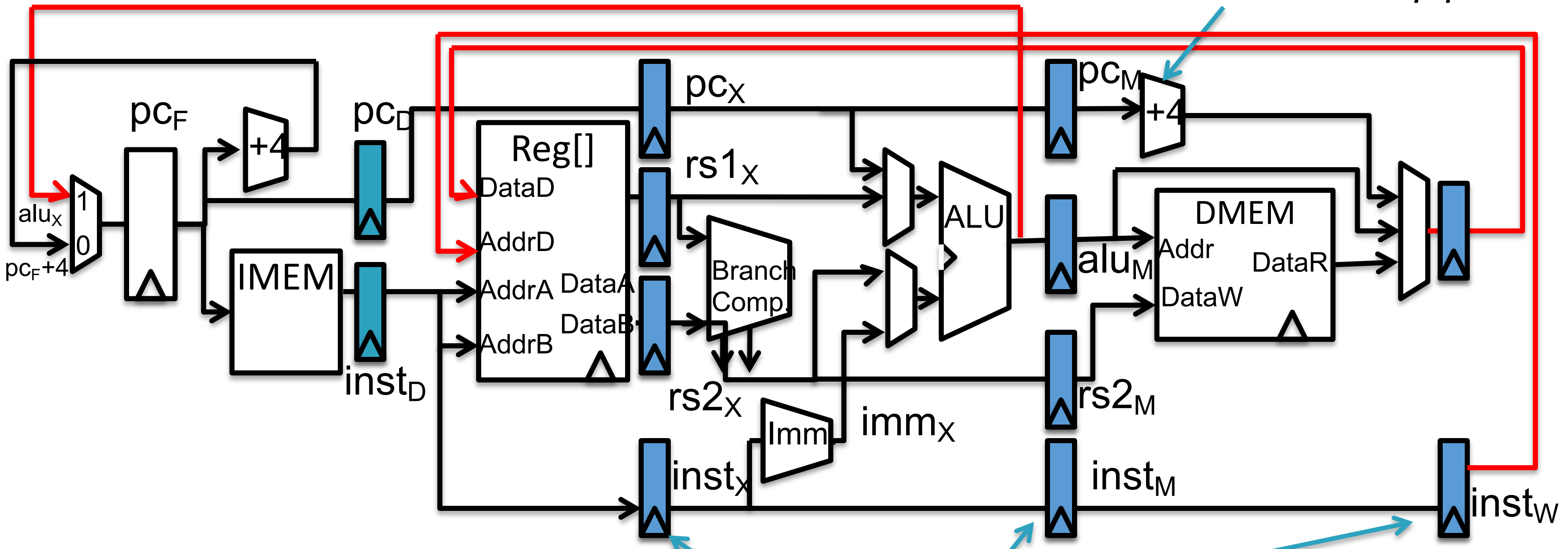


Pipelining RISC-V RV32I Datapath



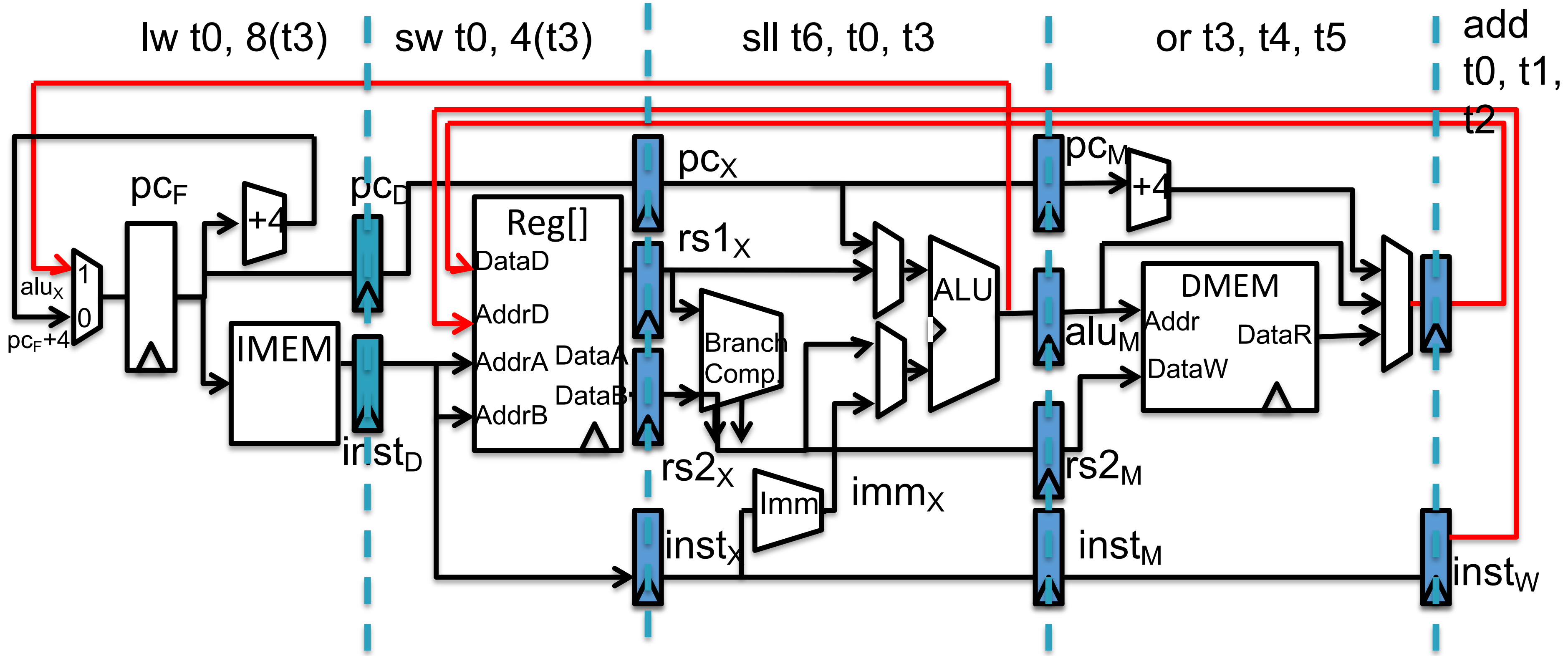
Pipelined RISC-V RV32I Datapath

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

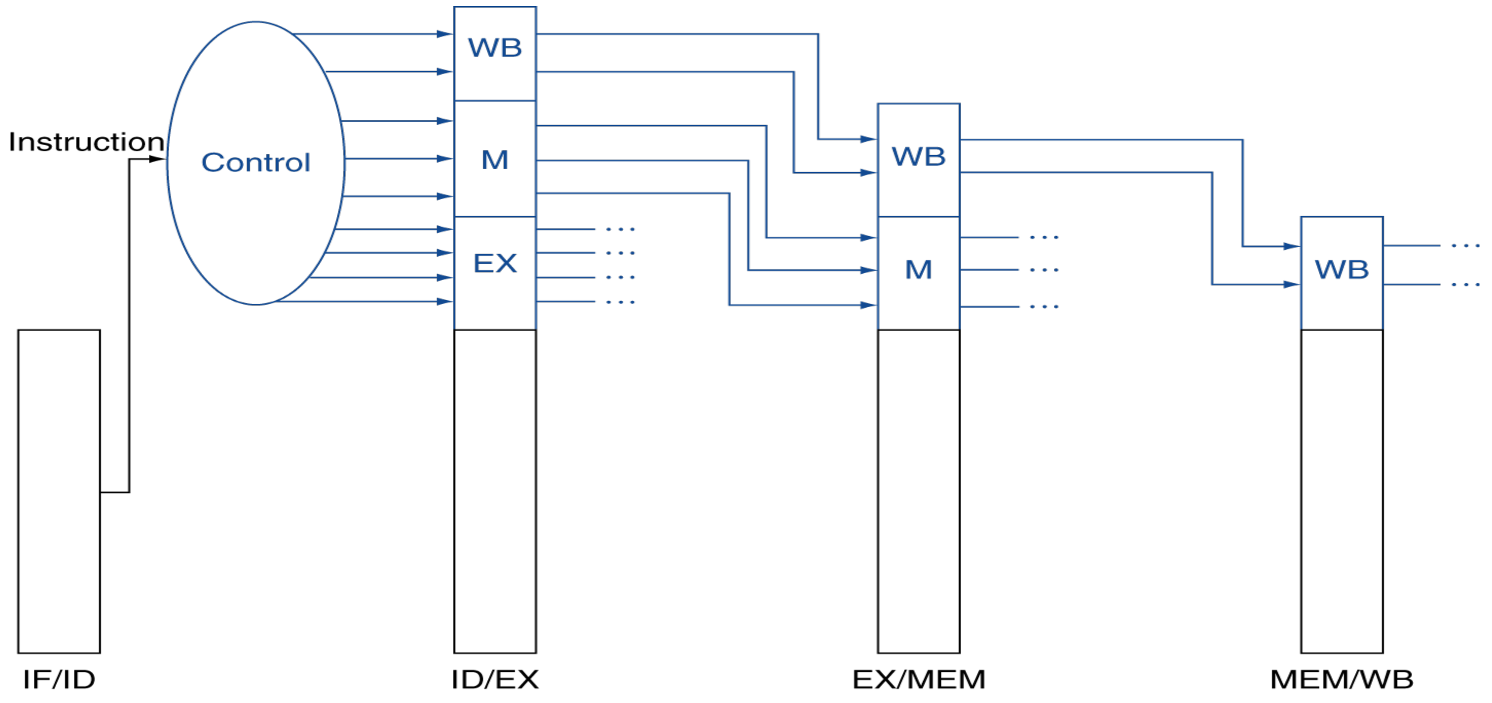
Each stage operates on different instruction



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



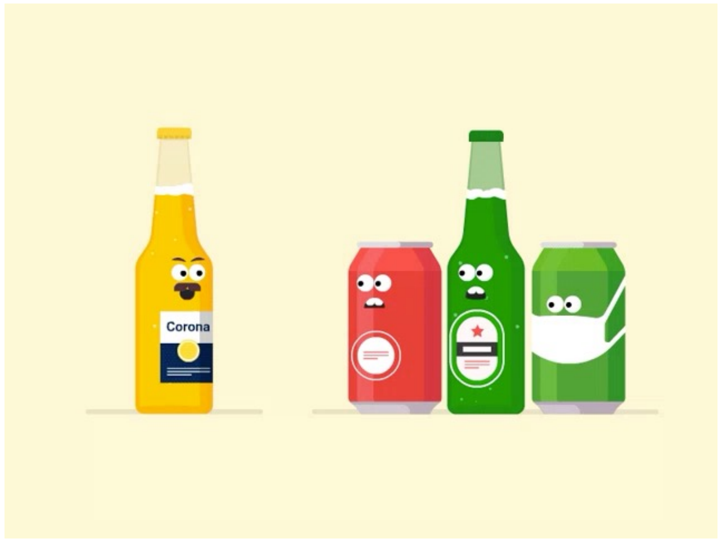
Question

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

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

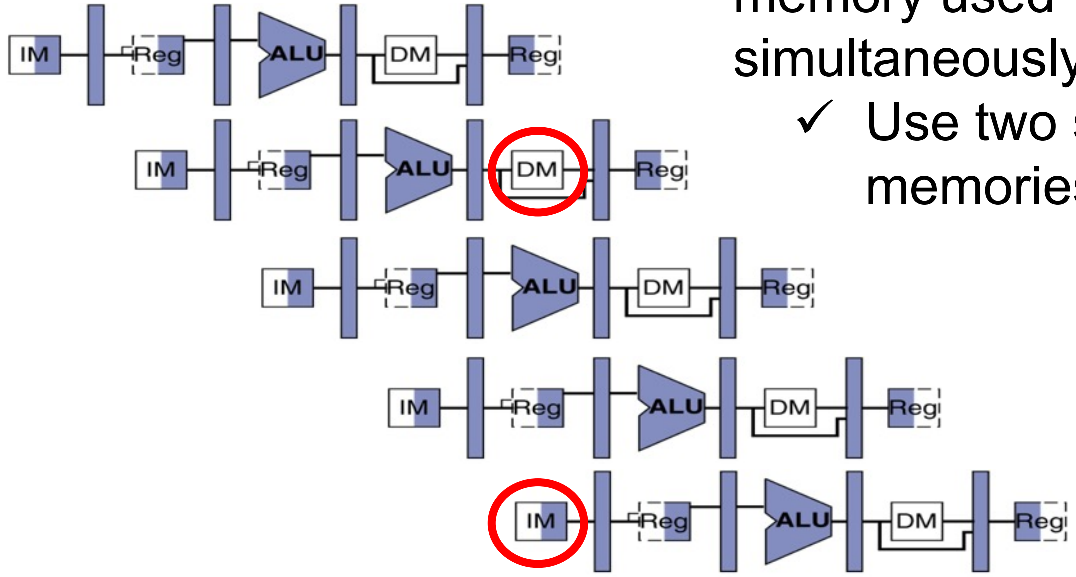
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

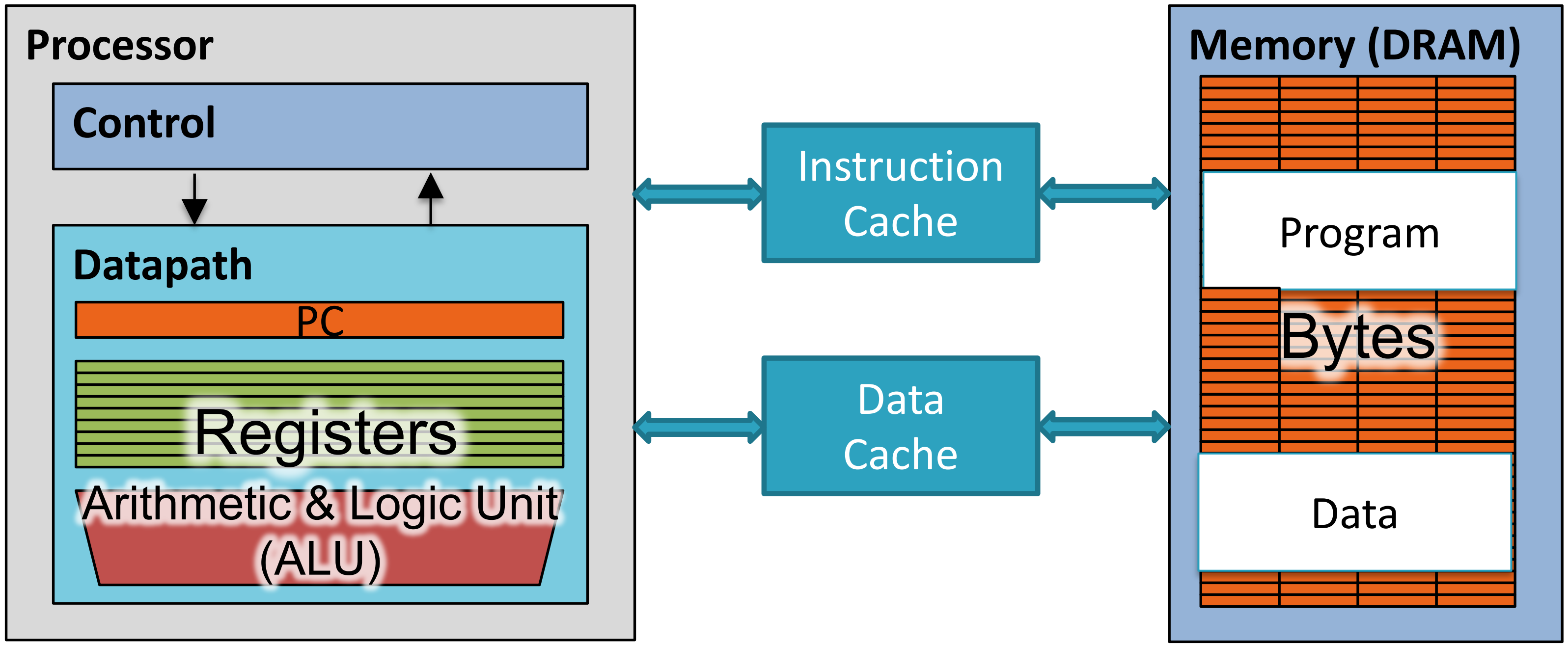
sw t0, 4(t3)

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?

instruction sequence

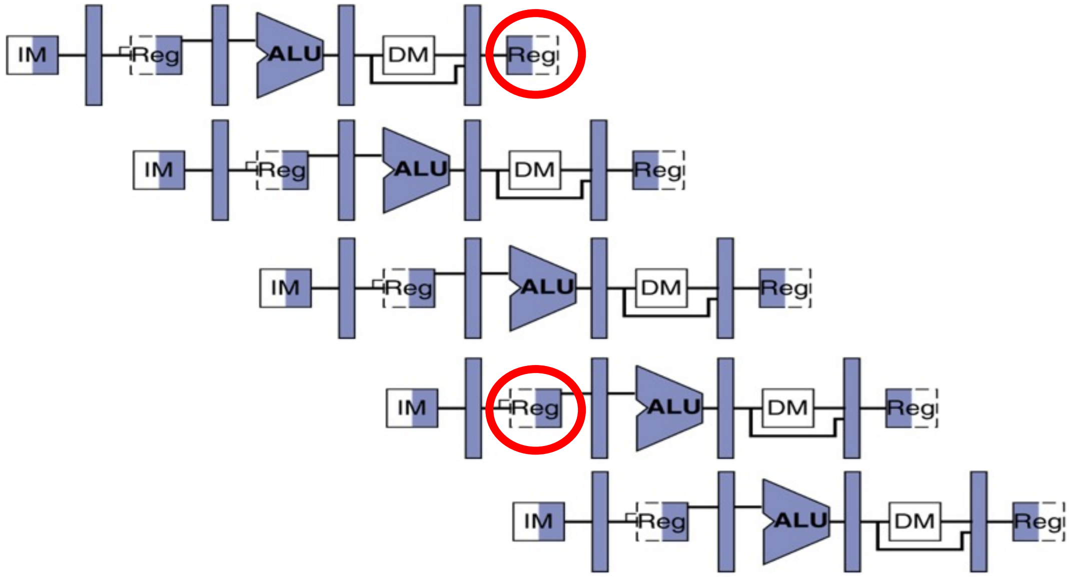
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)



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

instruction sequence

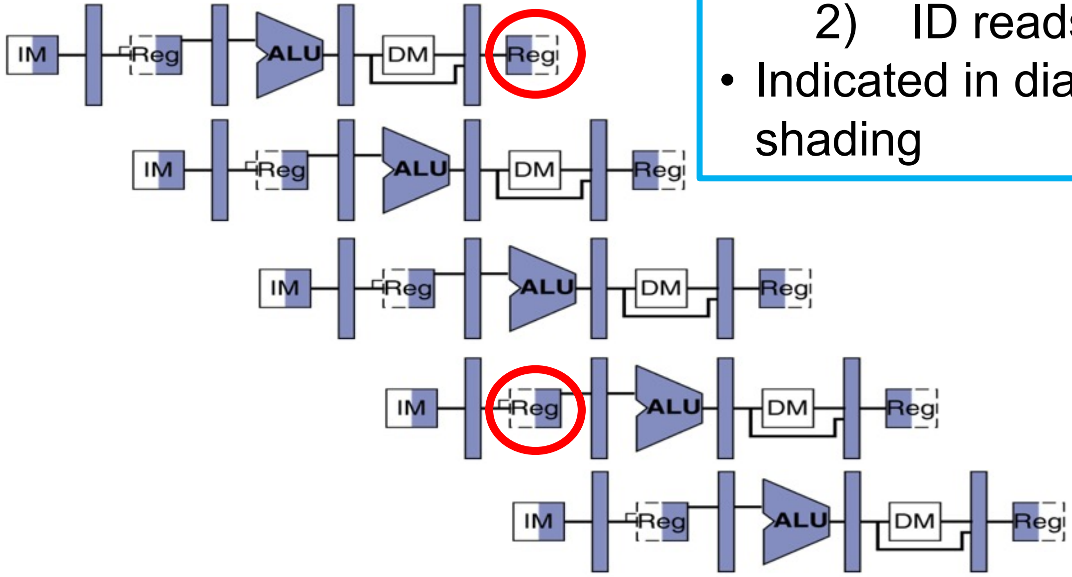
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

sw t0, 4(t3)

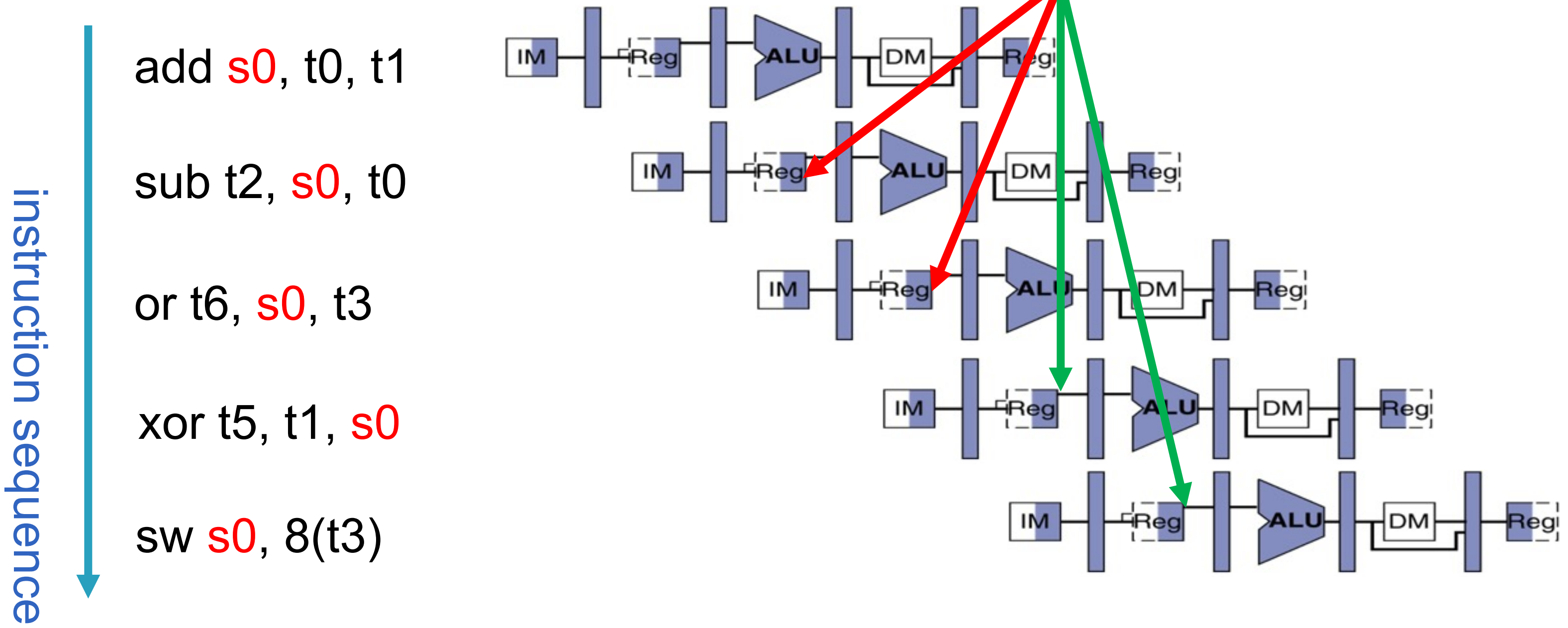
lw t0, 8(t3)



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

Data Hazard: ALU Result

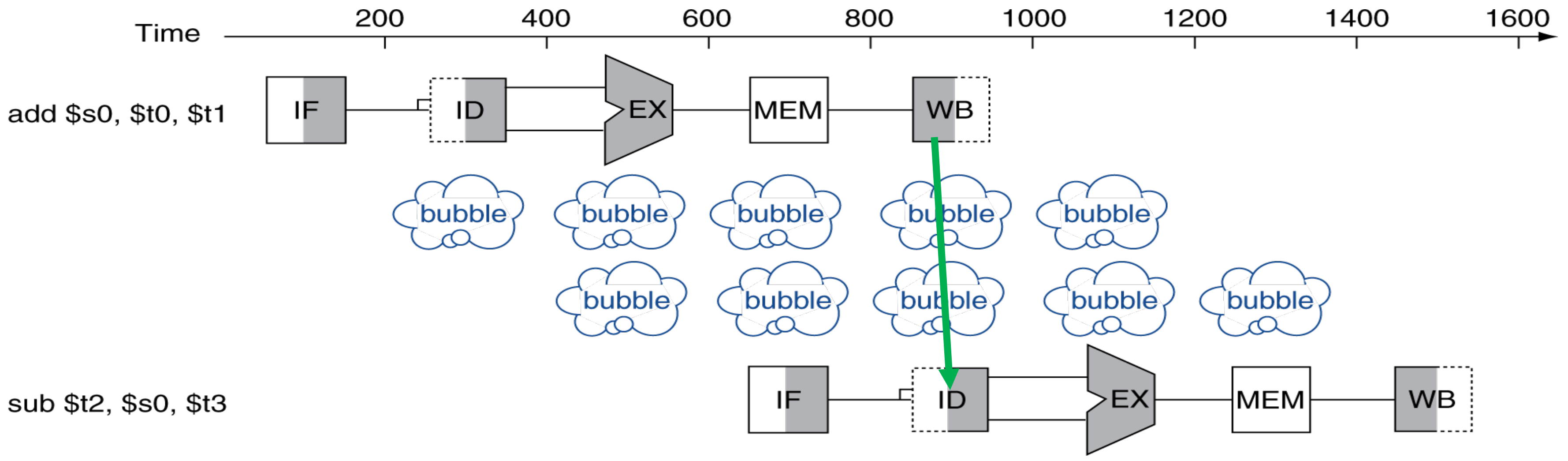
Value of s0	5	5	5	5	5/9	9	9	9	9



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

Value of t0									
	5	5	5	5	5/9	9	9	9	9

instruction sequence

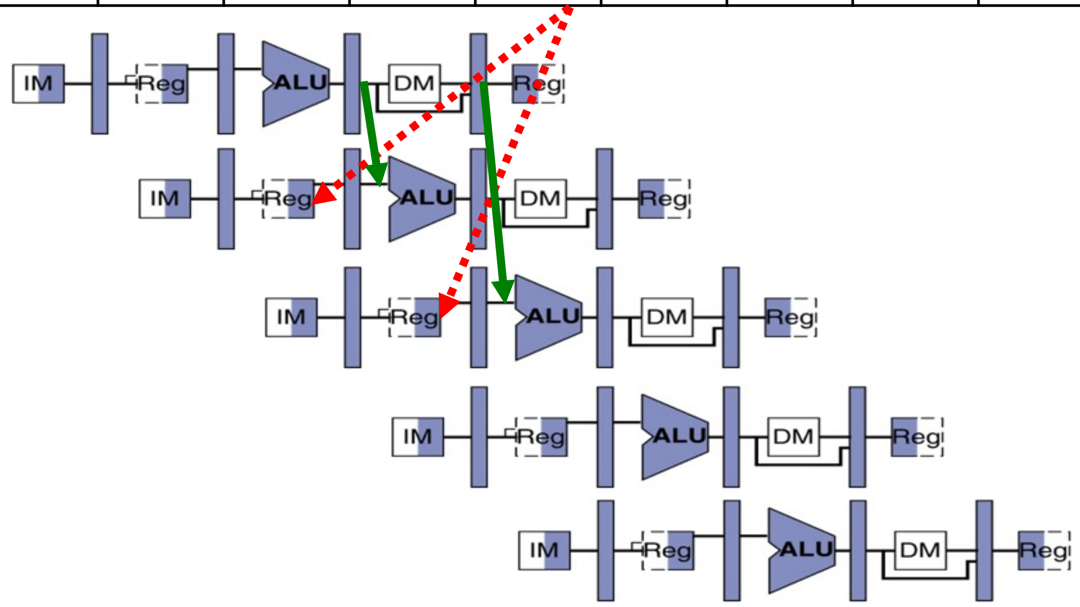
add t0, t1, t2

or t3, t0, t5

sub t6, t0, t3

xor t5, t1, t0

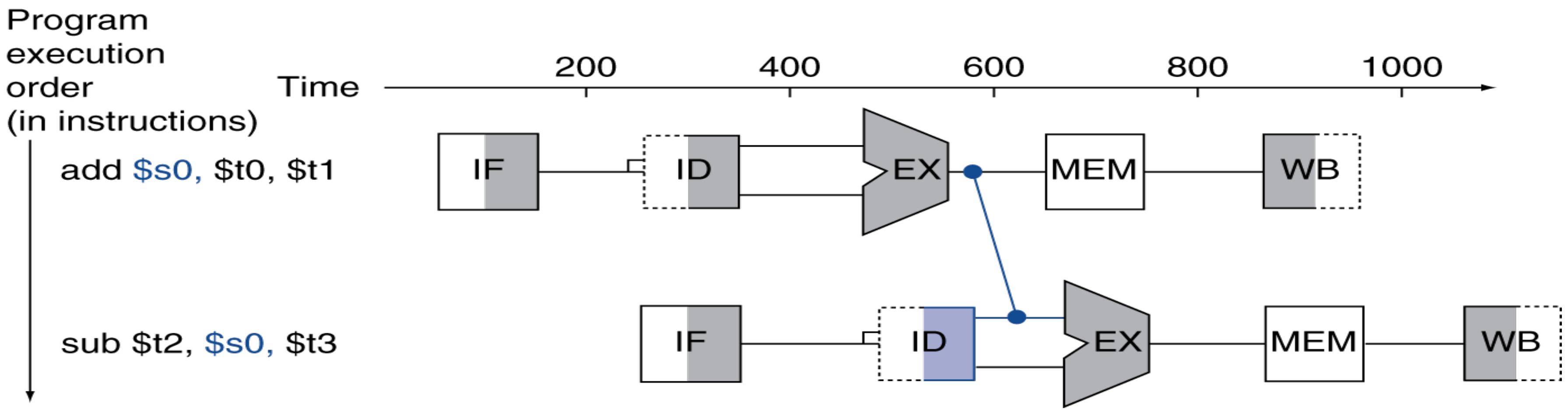
sw t0, 8(t3)



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



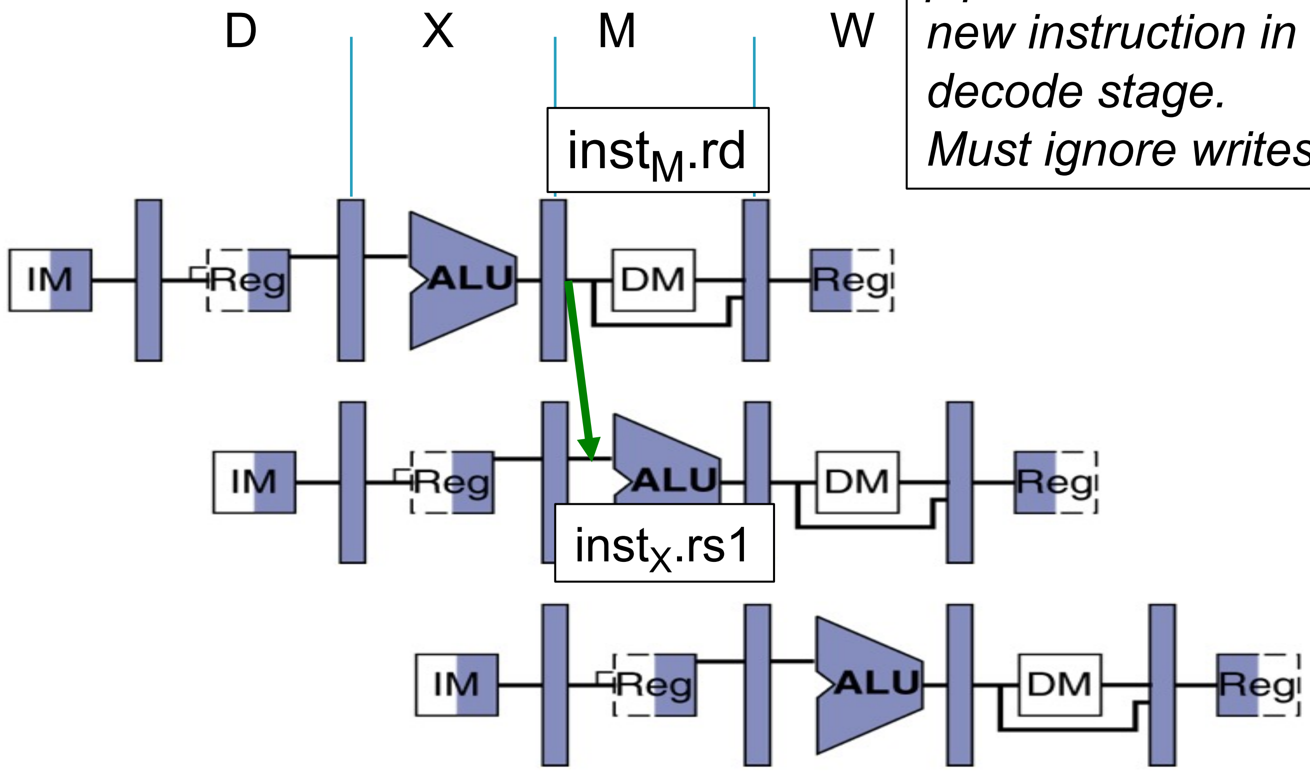
Detect Need for Forwarding (example)

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

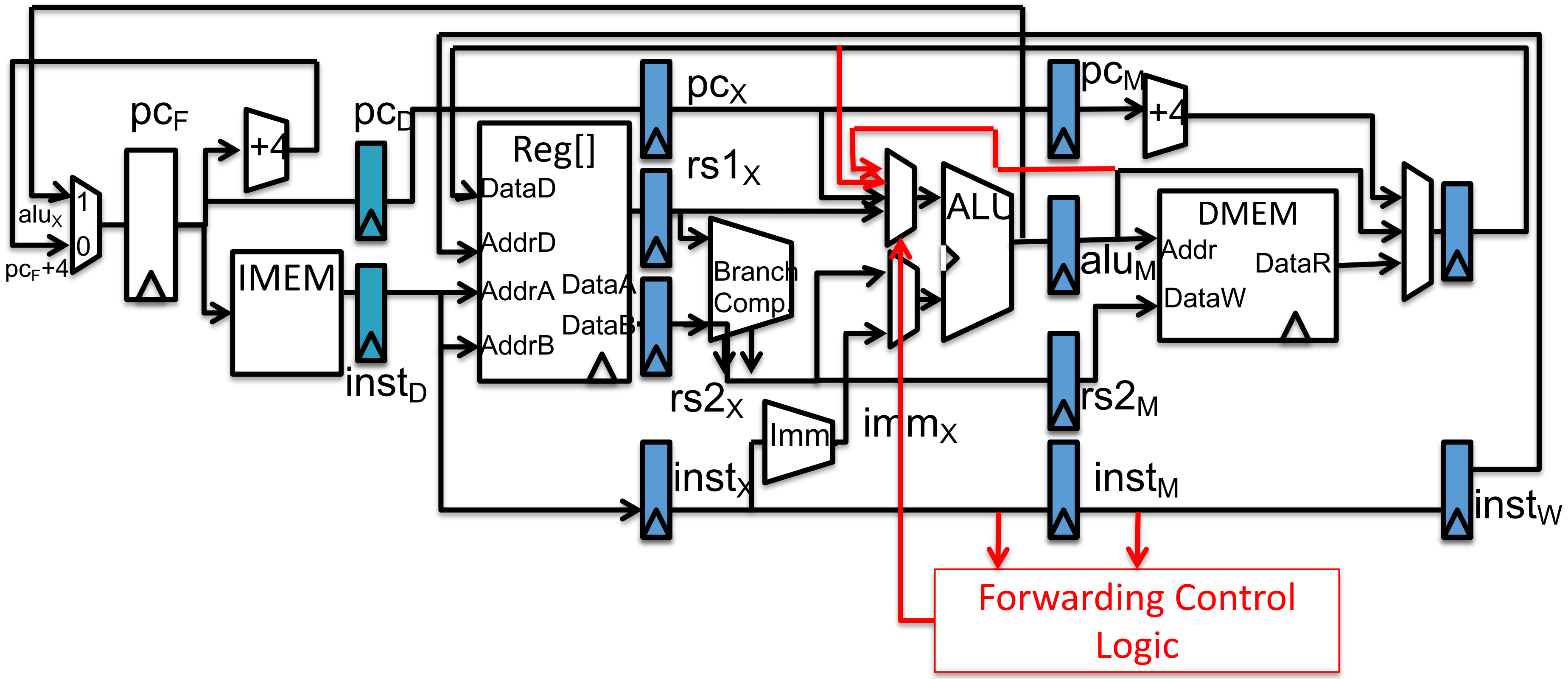
add t0, t1, t2

or t3, t0, t5

sub t6, t0, t3



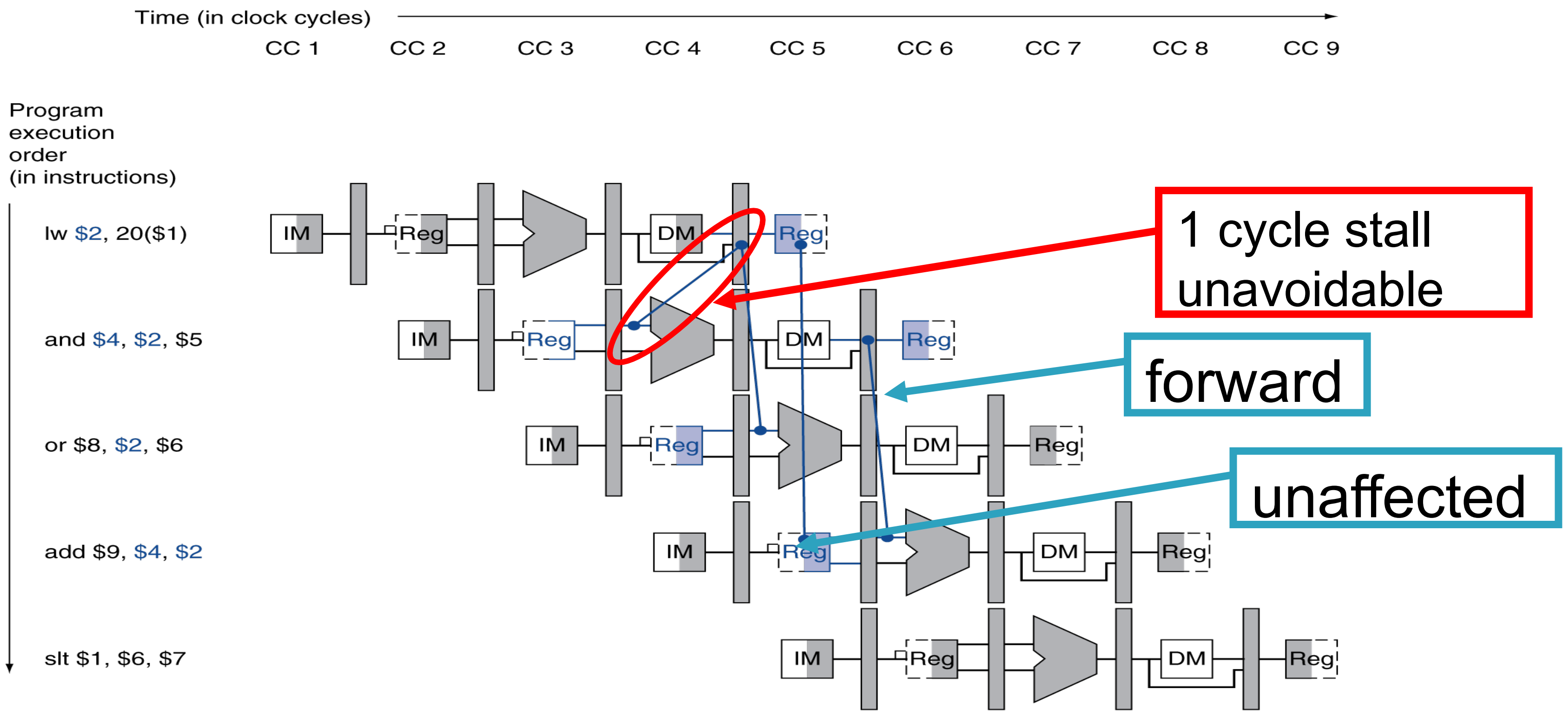
Forwarding Path



Agenda

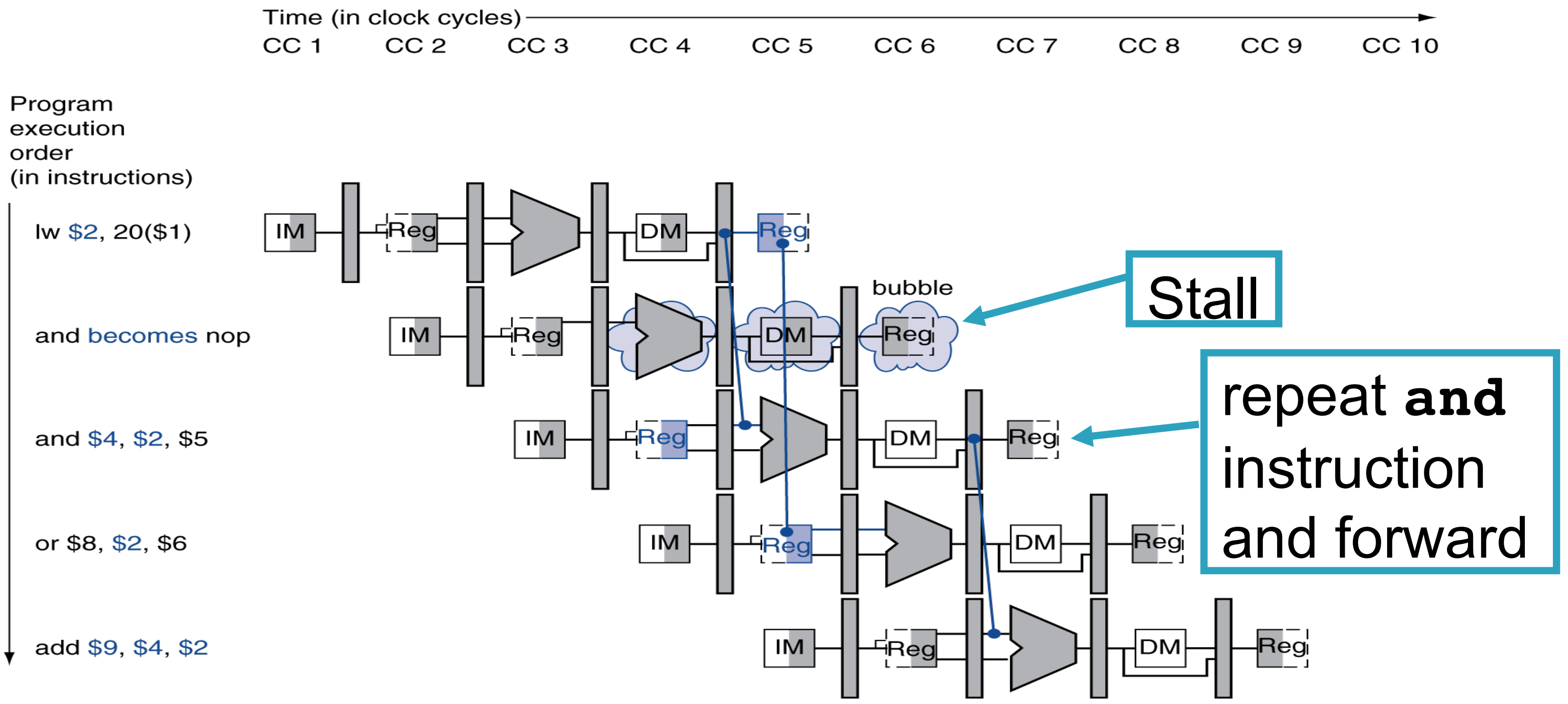
- Pipelining
- Hazards
  - Structural
  - Data
    - R-type instructions
    - Load
  - Control

Load Data Hazard



Above image from MIPS – in your head: change ‘$’ to ‘x’

Stall Pipeline

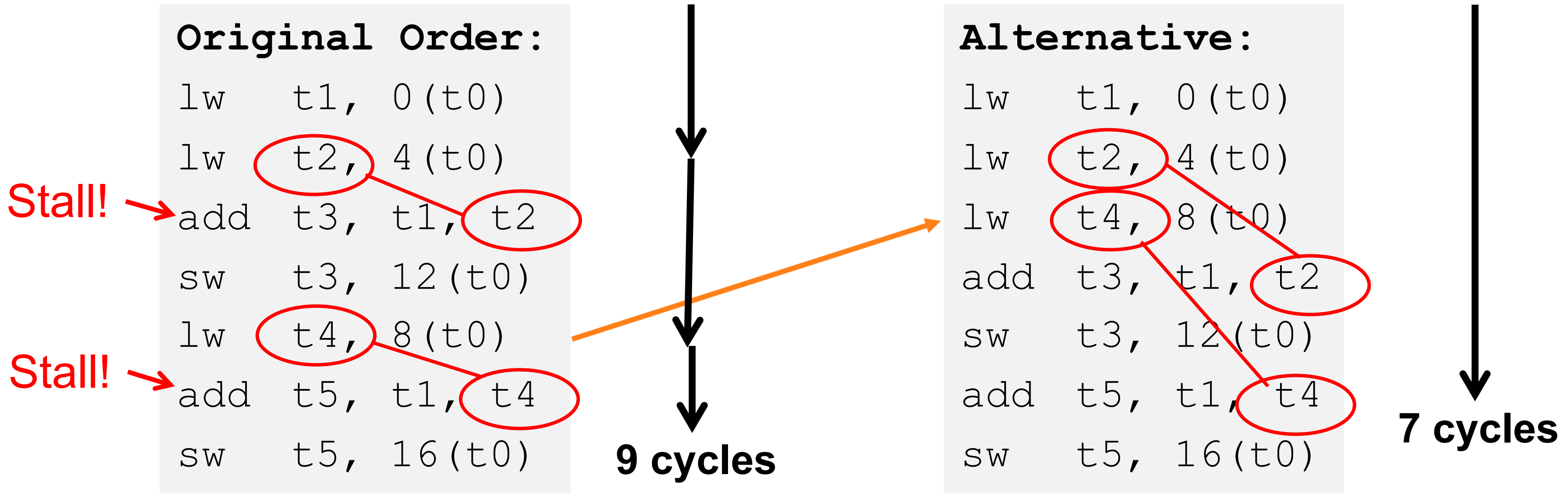


`lw` Data Hazard

- 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 **`A[3]=A[0]+A[1]; A[4]=A[0]+A[2]`**



```
Original Order:
lw   t1, 0(t0)
lw   t2, 4(t0)
add  t3, t1, t2
sw   t3, 12(t0)
lw   t4, 8(t0)
add  t5, t1, t4
sw   t5, 16(t0)
```

```
Alternative:
lw   t1, 0(t0)
lw   t2, 4(t0)
lw   t4, 8(t0)
add  t3, t1, t2
sw   t3, 12(t0)
add  t5, t1, t4
sw   t5, 16(t0)
```

Agenda

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

Control Hazards

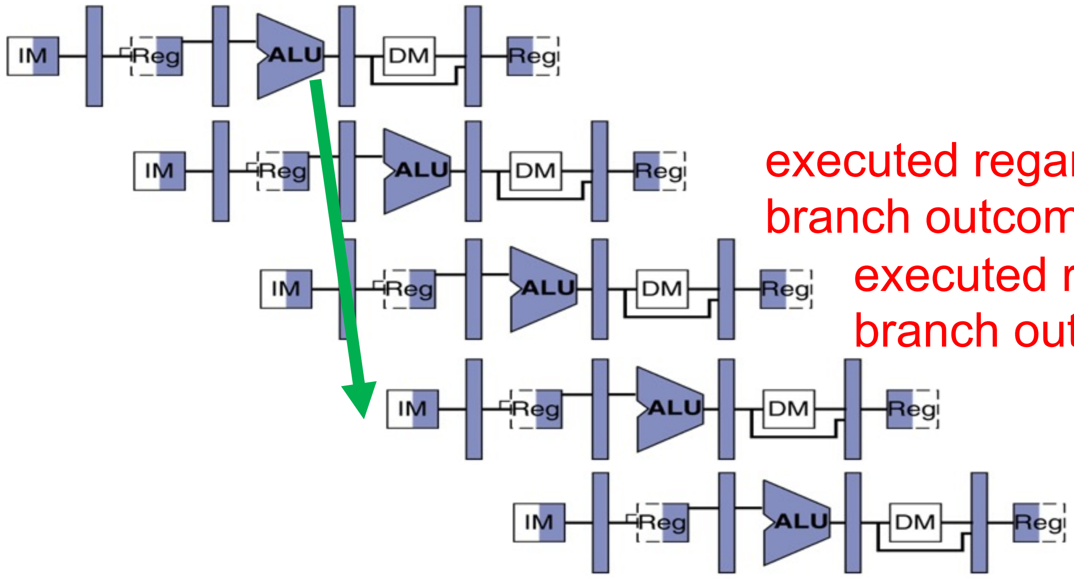
beq t0, t1, label

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

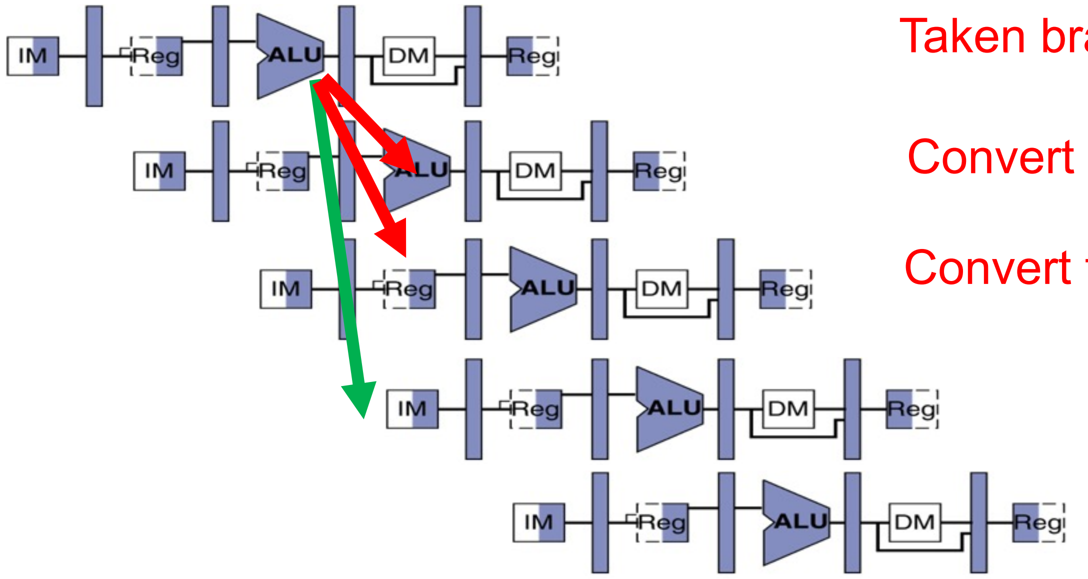
Kill Instructions after Branch if Taken

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

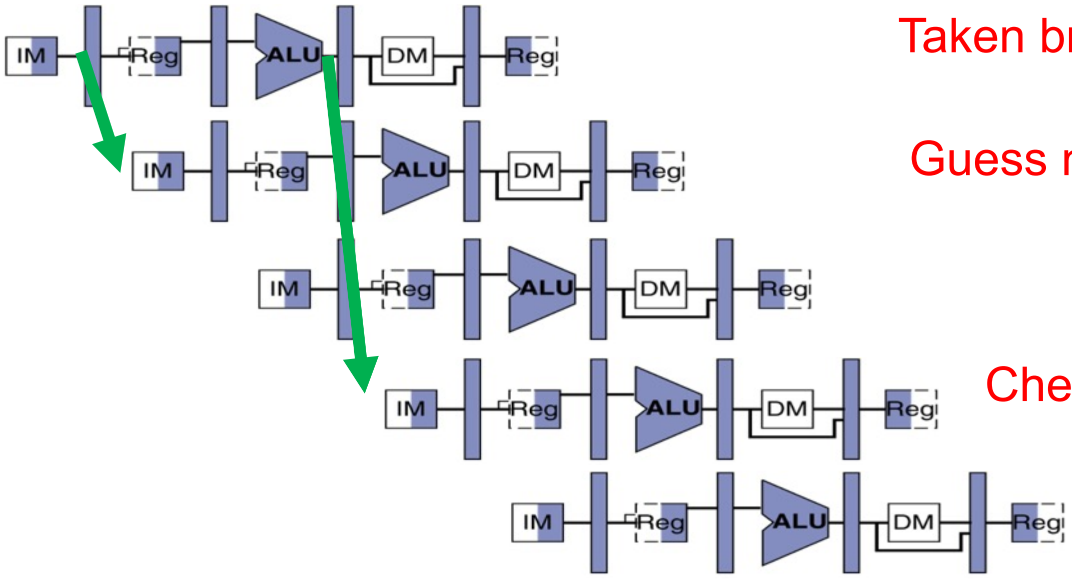
- Every taken branch in simple pipeline costs 2 dead cycles
- To improve performance, use “branch prediction” to guess which way branch will go earlier in pipeline
- Only flush pipeline if branch prediction was incorrect

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