EECS151/251A
Introduction to Digital Design and ICs

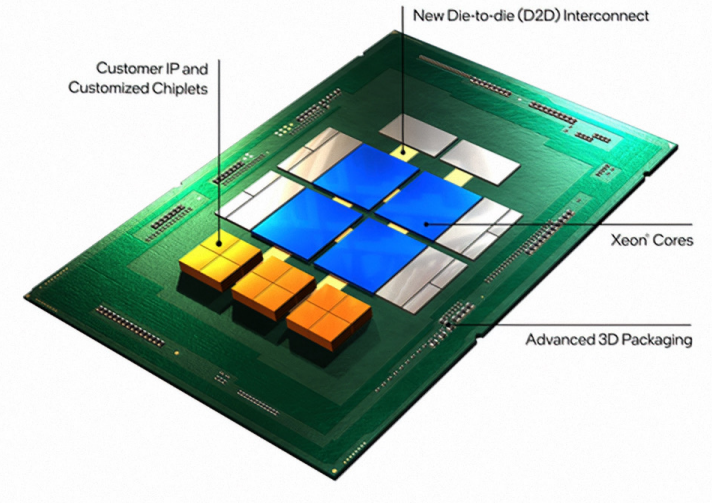
Lecture 10: RISC-V Pipelining

Sophia Shao

Intel's plan to license x86 cores for chips with Arm, RISC-V and more inside

Intel will license its most important asset, the x86 architecture, to those who want to make custom silicon. Depending on the application, customers will be able to mix up x86, Arm and RISC-V CPU cores as well as hardware acceleration units in a custom-designed chip that Intel fabricates.

"We have what we call a multi ISA strategy. That's the first time in Intel's history we'll license x86 soft cores and hard cores to customers who would like to develop chips," Bob Brennan, vice president of customer solutions engineering at Intel's Foundry Services, told *The Register*.

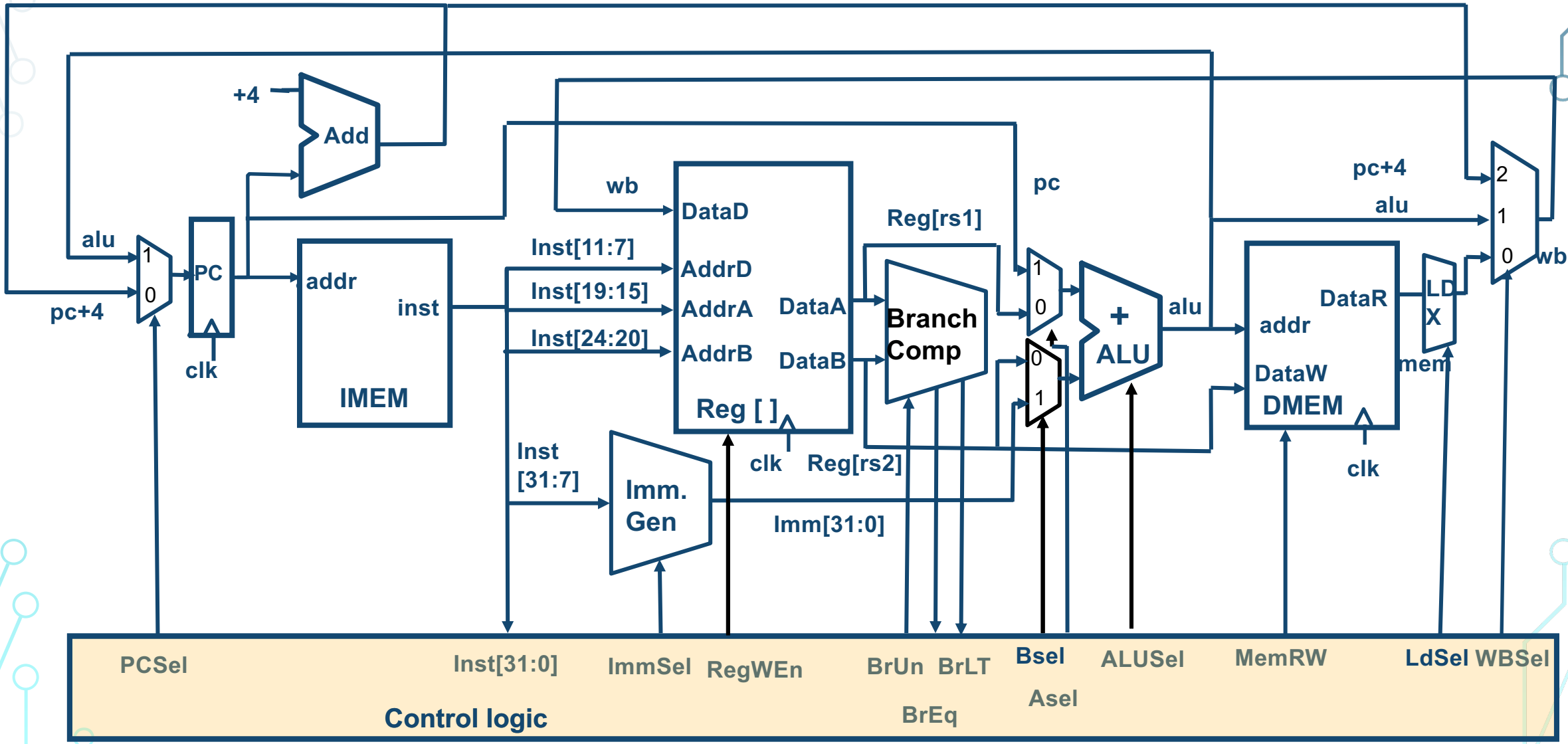


https://www.theregister.com/2022/02/14/intel_x86_licensing/

Review

- We have covered the implementation of the base ISA for RV32I!!!
  - Get yourself familiar with the ISA Spec.
- Instruction type:
  - R-type
  - I-type
  - S-type
  - B-type
  - J-type
  - U-type
- Implementation suggested is straightforward, yet there are modalities in how to implement it well at gate level.
- Single-cycle datapath is slow – need to pipeline it

Complete RV32I Datapath with Control



- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

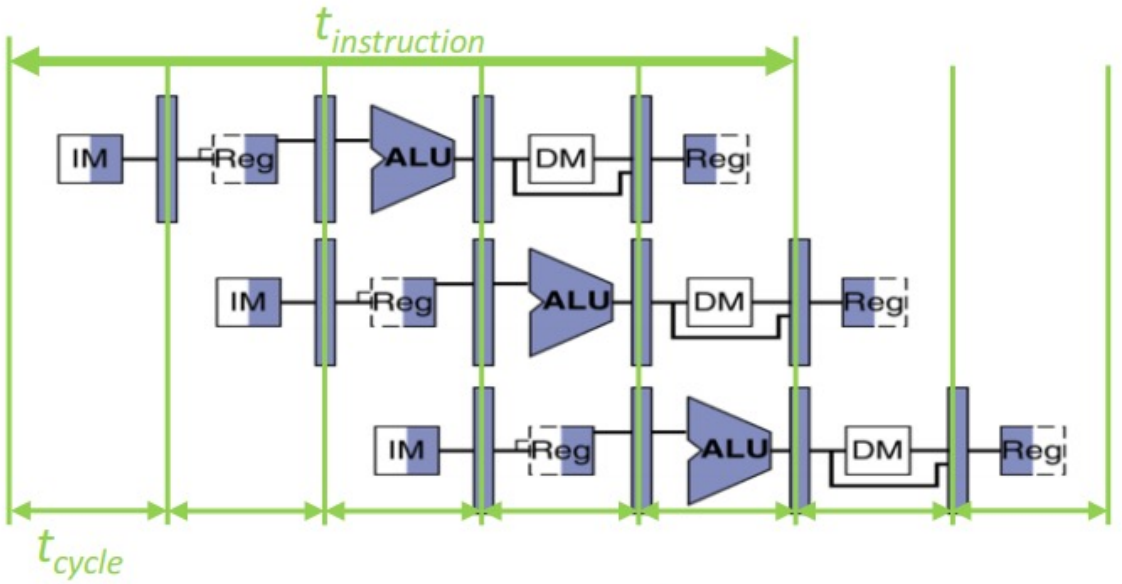
Pipelining with RISC-V

instruction sequence

add t0, t1, t2

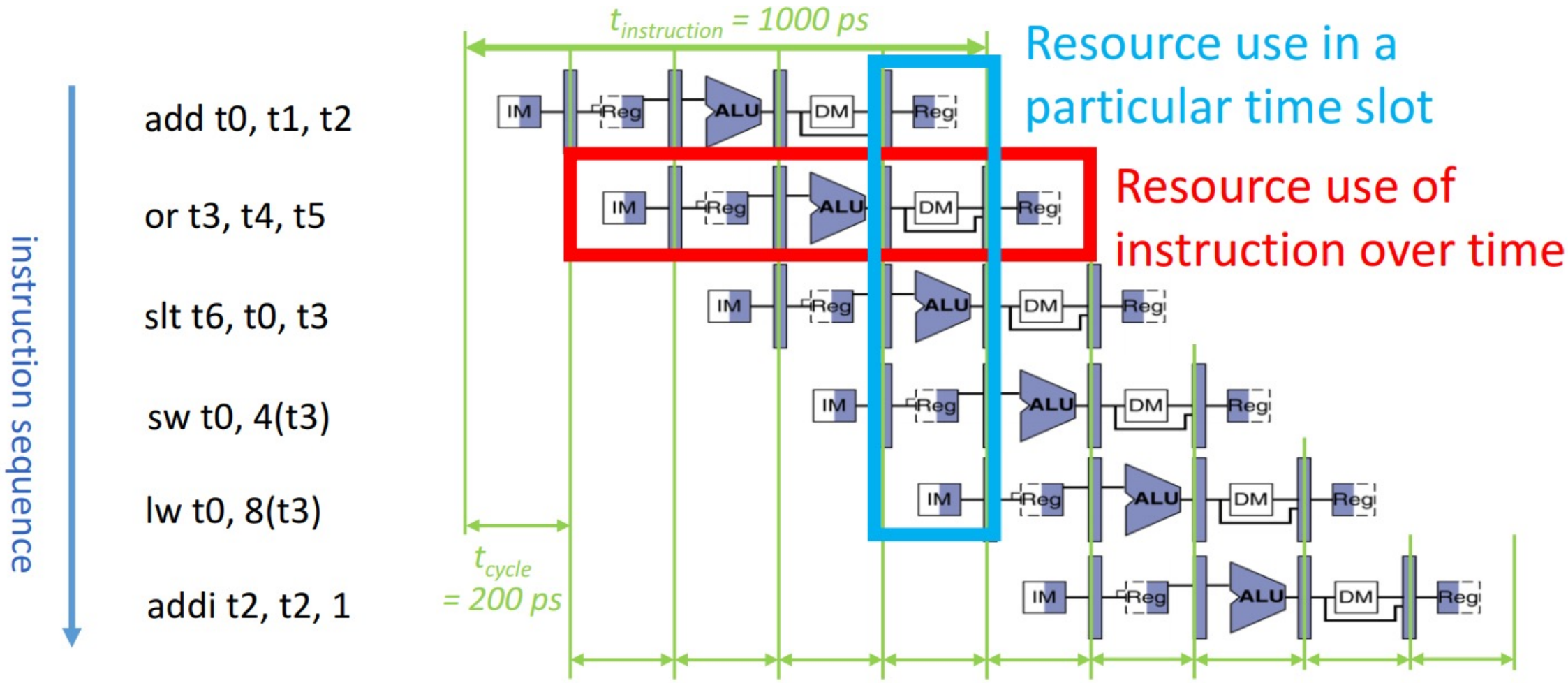
or t3, t4, t5

sll t6, t0, t3

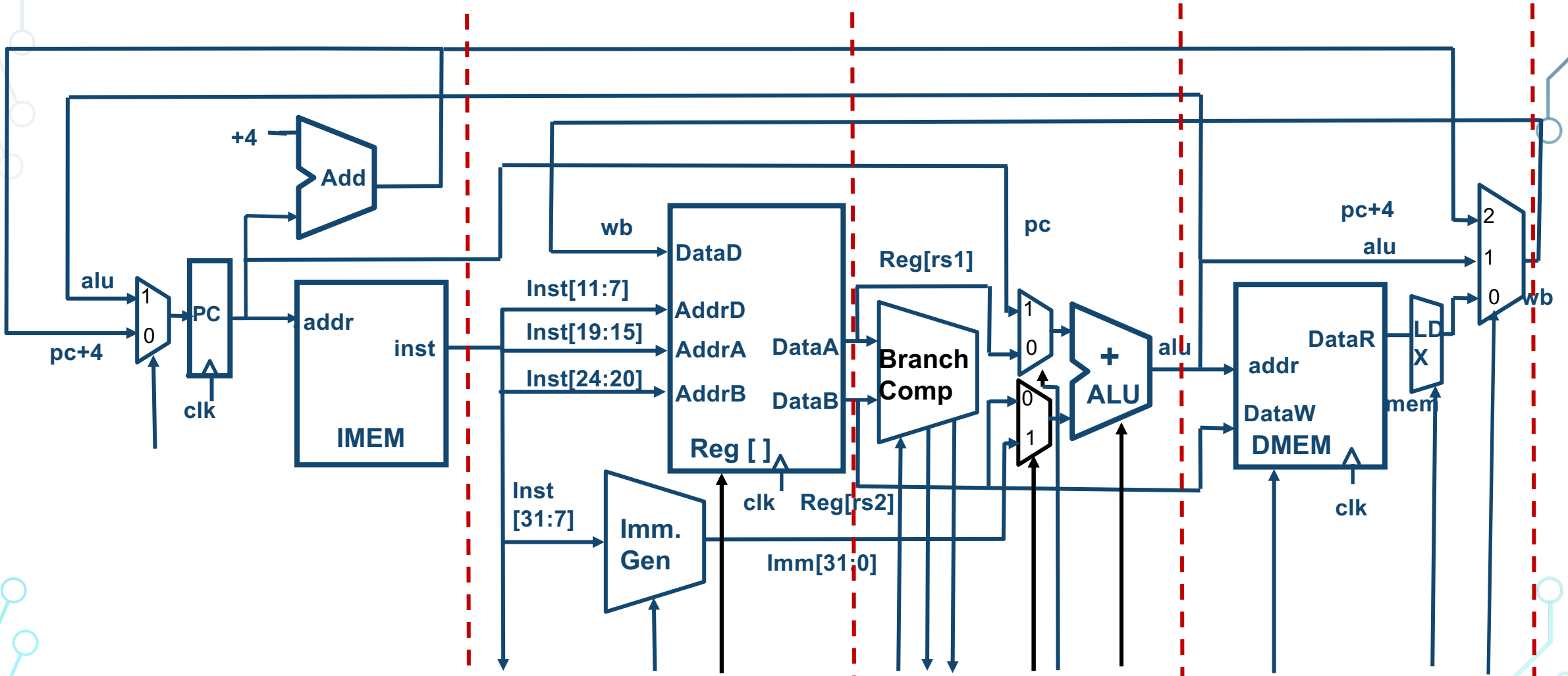


	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz
Relative speed	1 x	4 x

Pipelining with RISC-V



Complete RV32I Datapath with Control



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

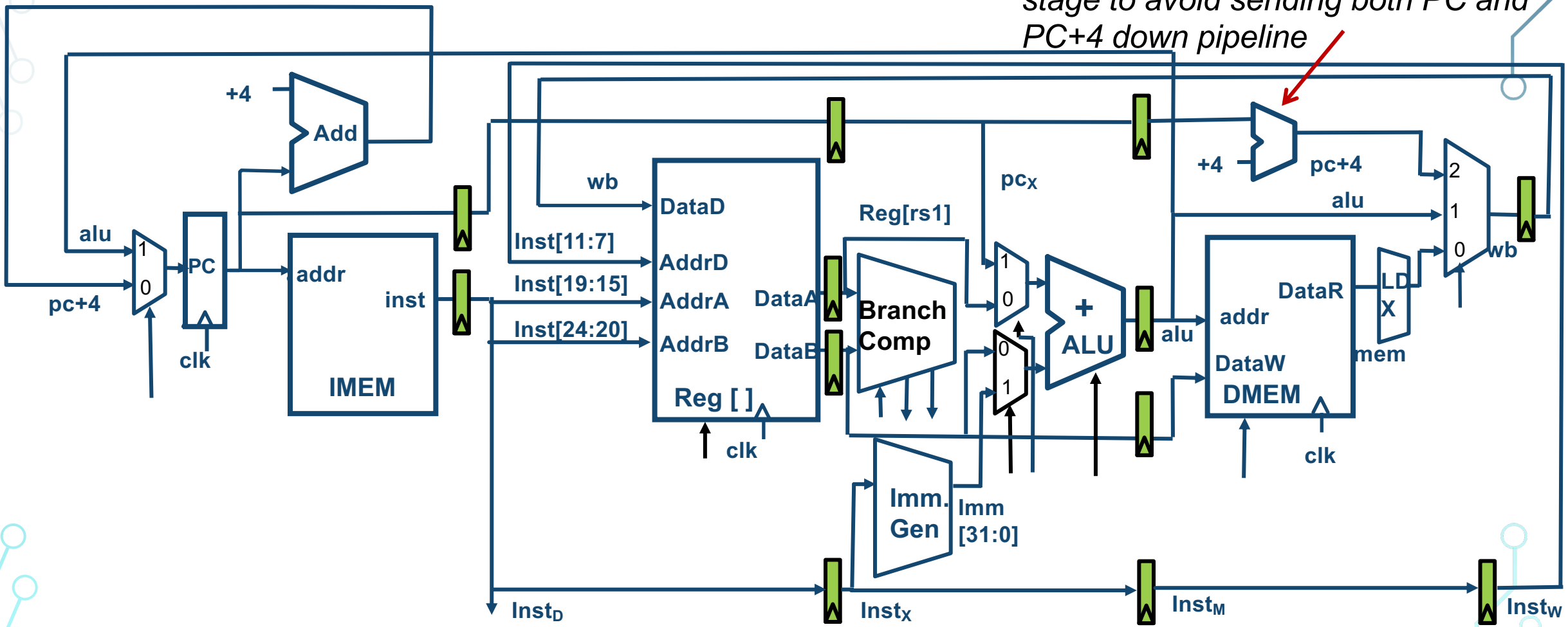
ALU Execute (X)

Memory Access (M)

Write Back (W)

Pipelining RV32I Datapath

In this design, recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

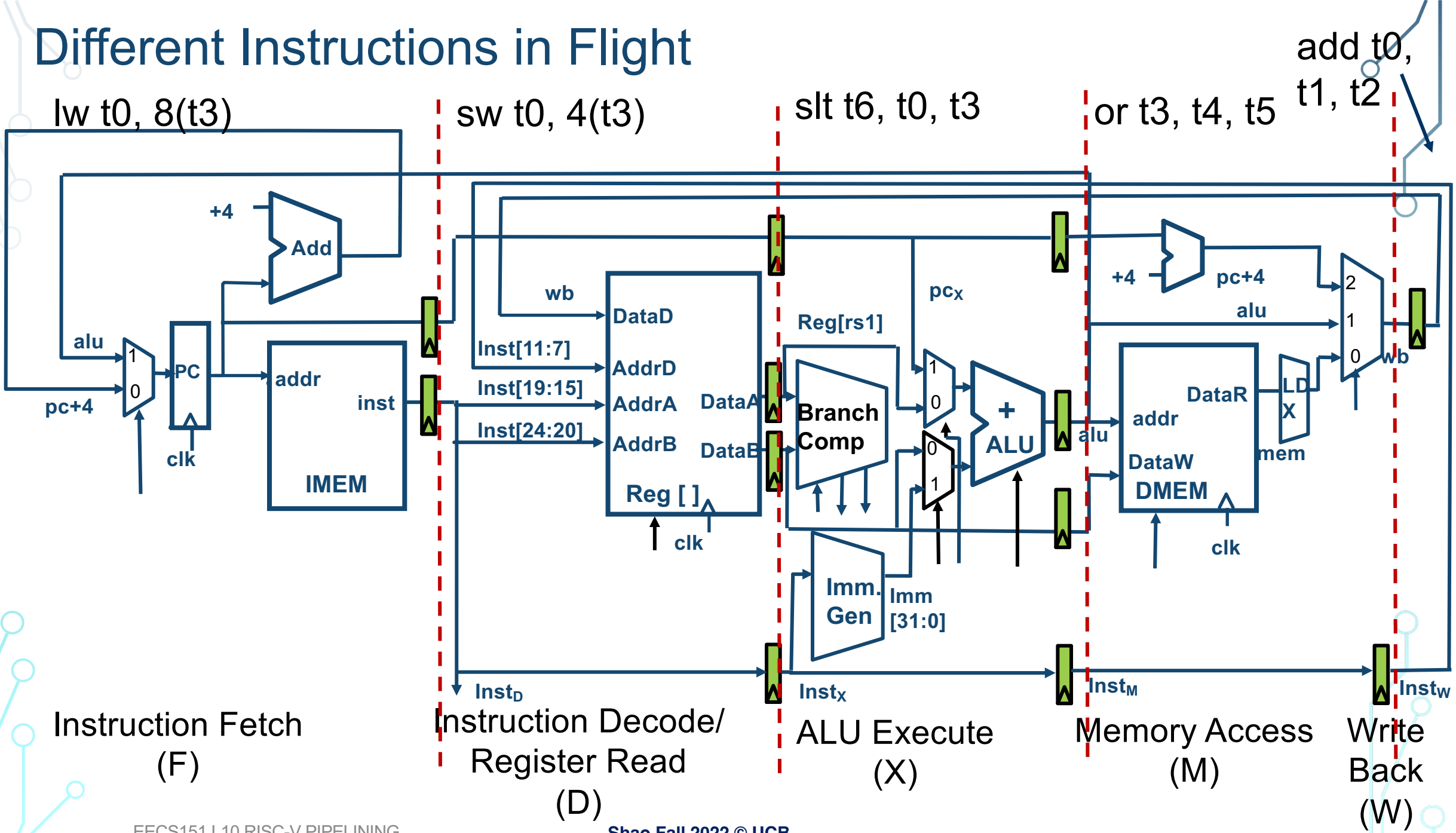
ALU Execute (X)

Memory Access (M)

Write Back (W)

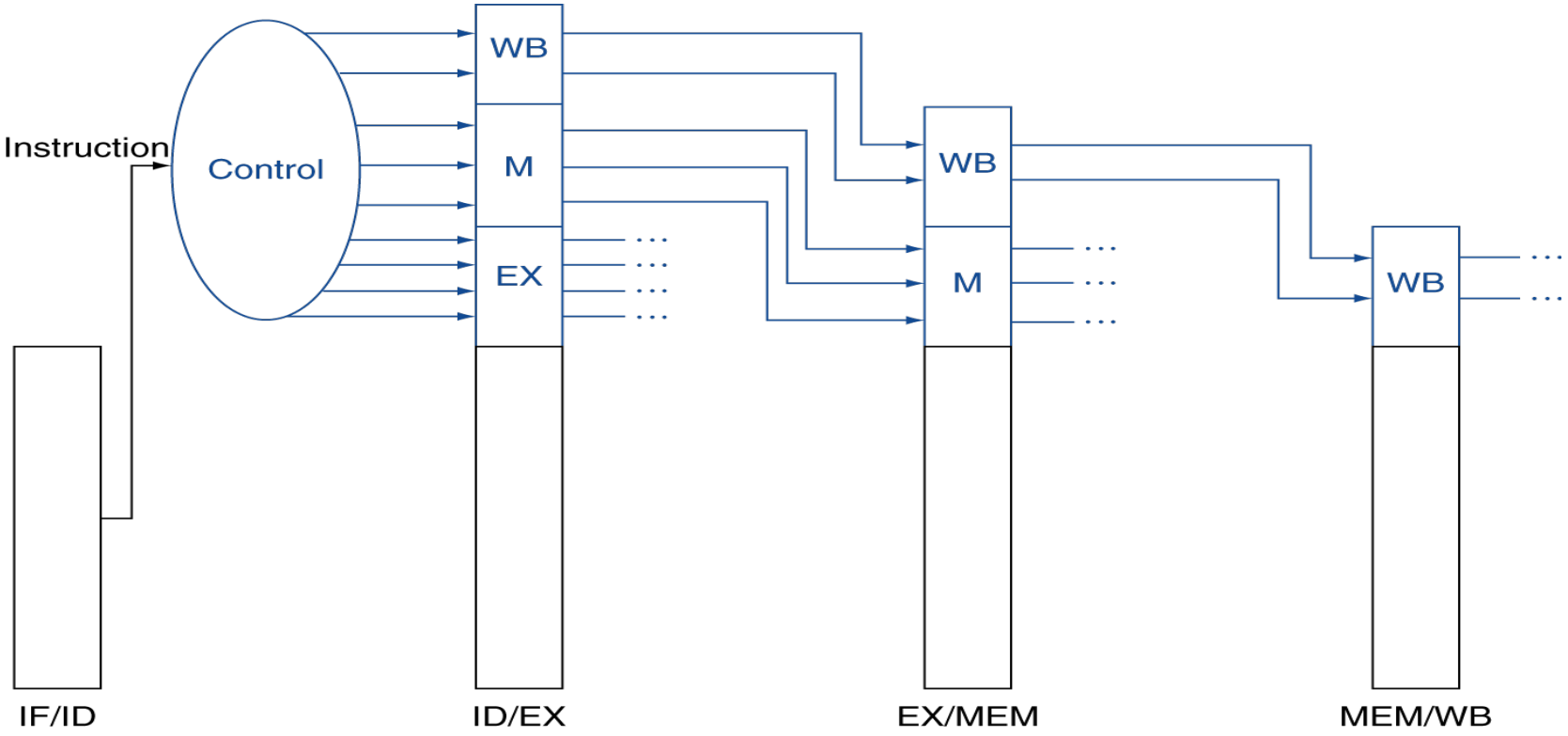
Must pipeline instruction along with data, so control operates correctly in each stage

Different Instructions 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



- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - 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*

Structural Hazards I: Regfile

- 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
  - We will see a circuit design for multi-ported register files later in the class
- Processors with multiple execution paths have larger number of ports

Structural Hazard II : Memory Access

instruction sequence

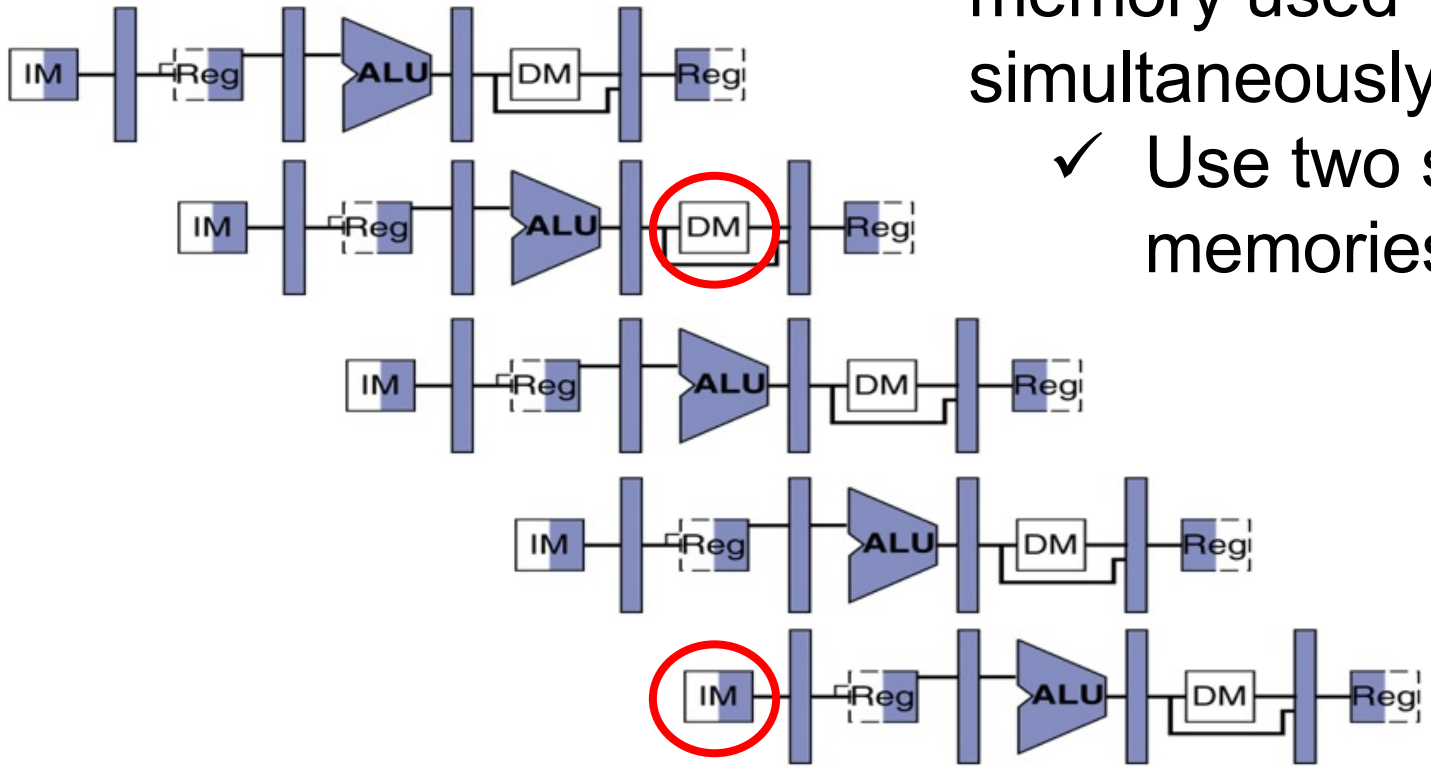
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)



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

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

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - 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?

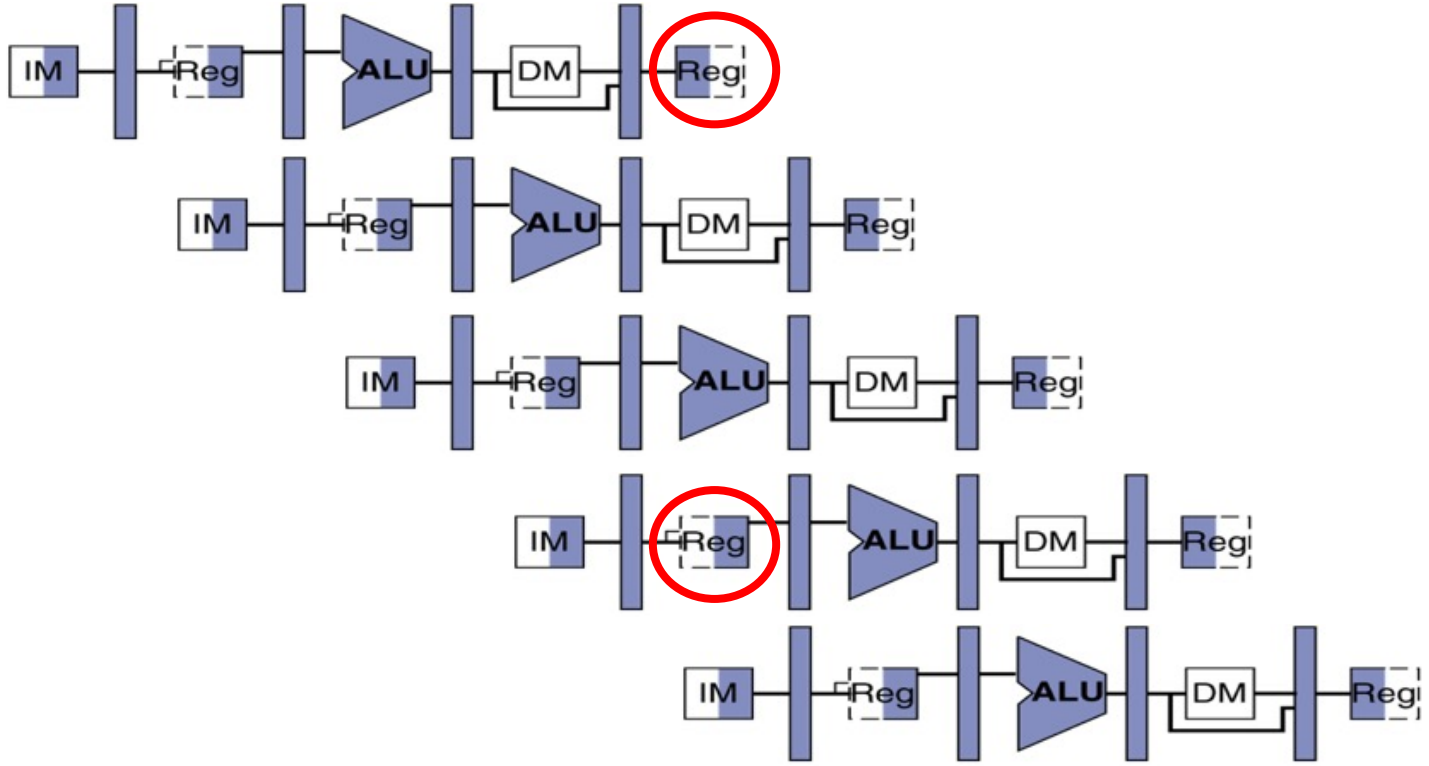
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)



instruction sequence

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

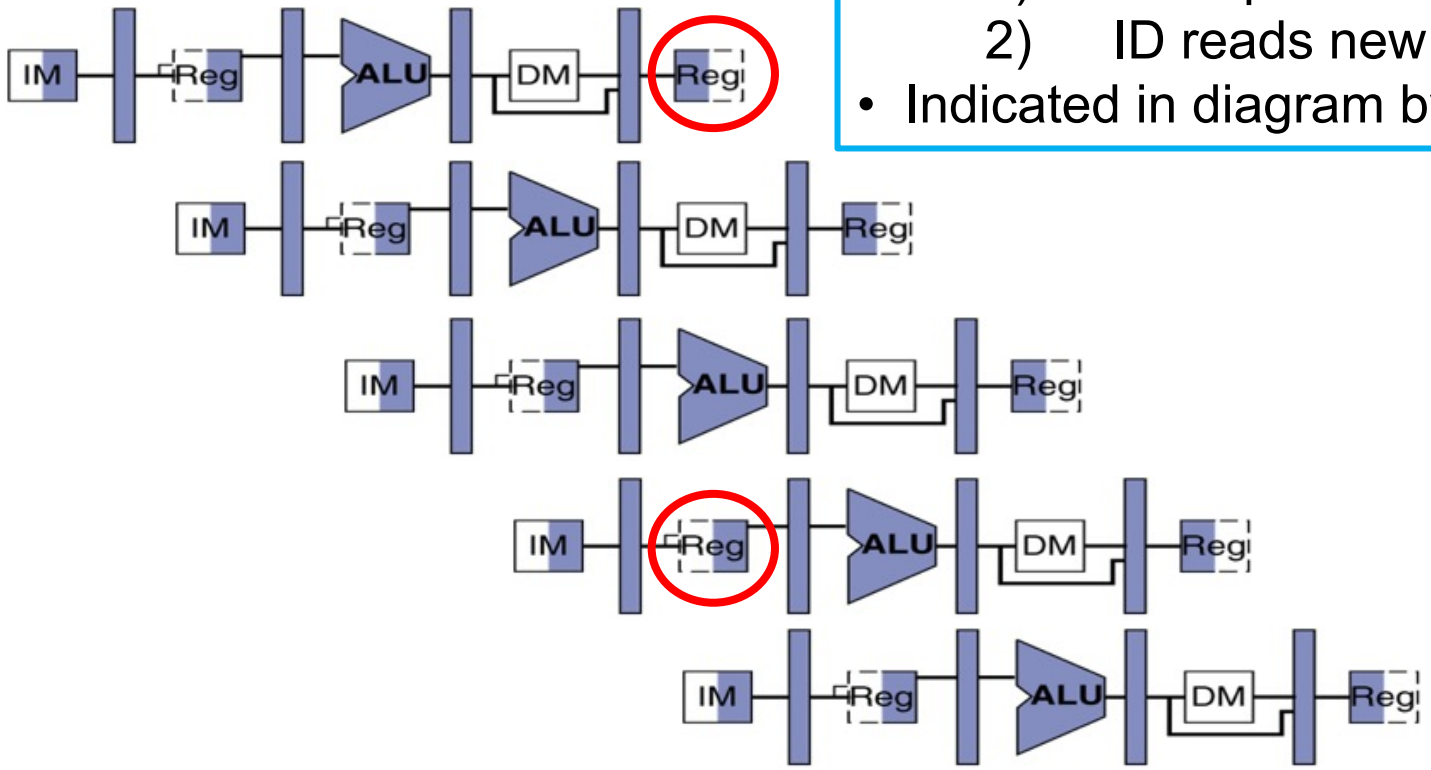
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

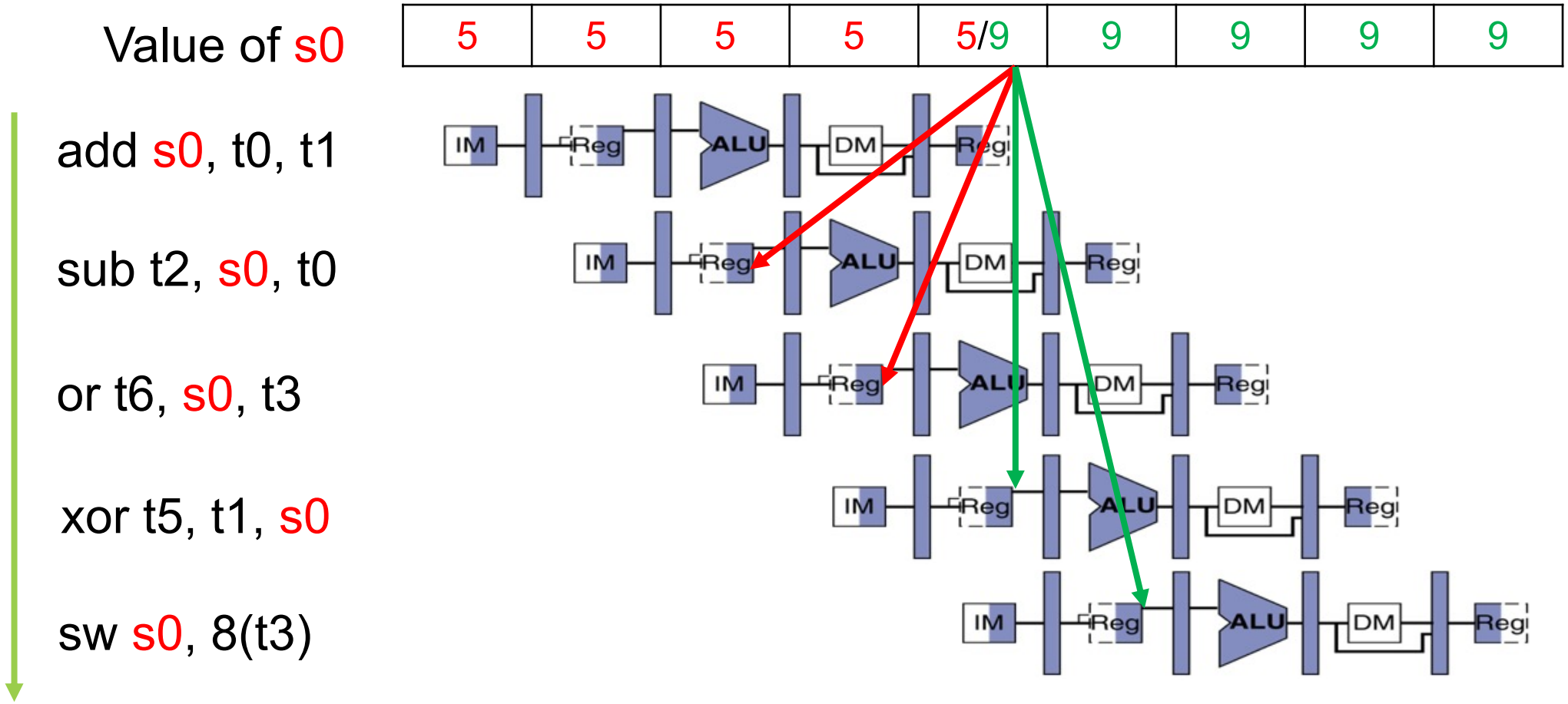
sw t0, 4(t3)

lw t0, 8(t3)



Might not always be possible to write then read in same cycle, especially in high-frequency designs. Check assumptions in any question.

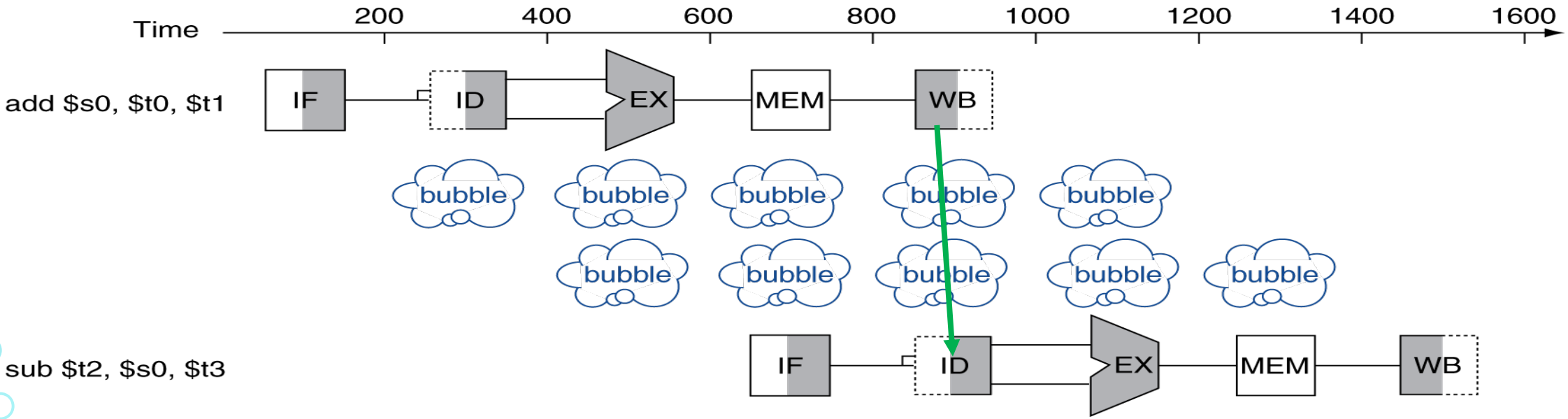
Data Hazard: ALU Result



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



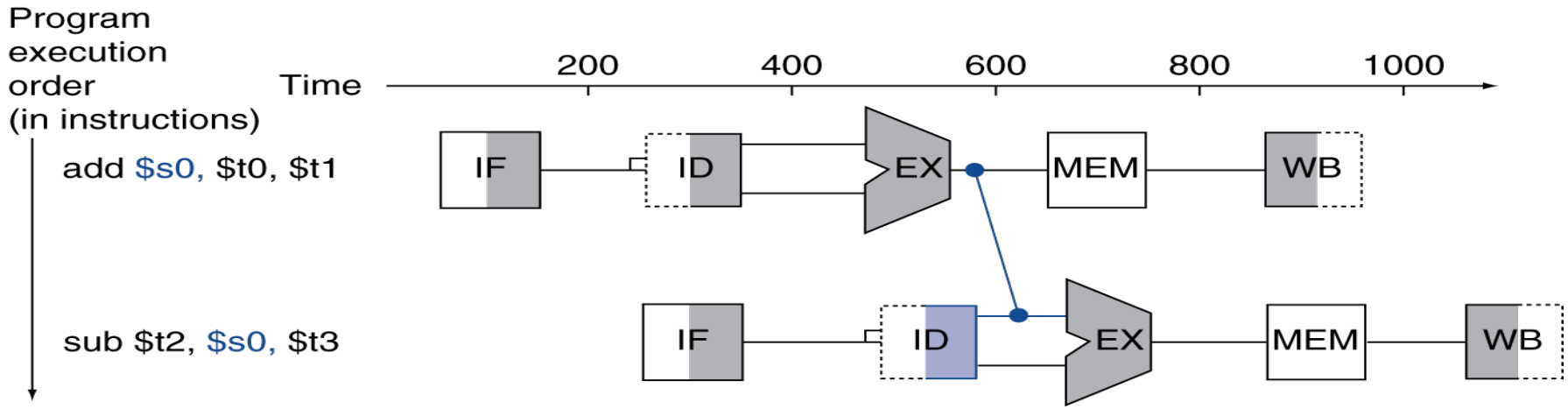
- Bubble:
  - effectively NOP: affected pipeline stages do “nothing”

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
  - But requires knowledge of the pipeline structure

Solution 2: Forwarding

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



Solution 2: Forwarding

Value of t0

5	5	5	5	5/9	9	9	9	9

instruction sequence

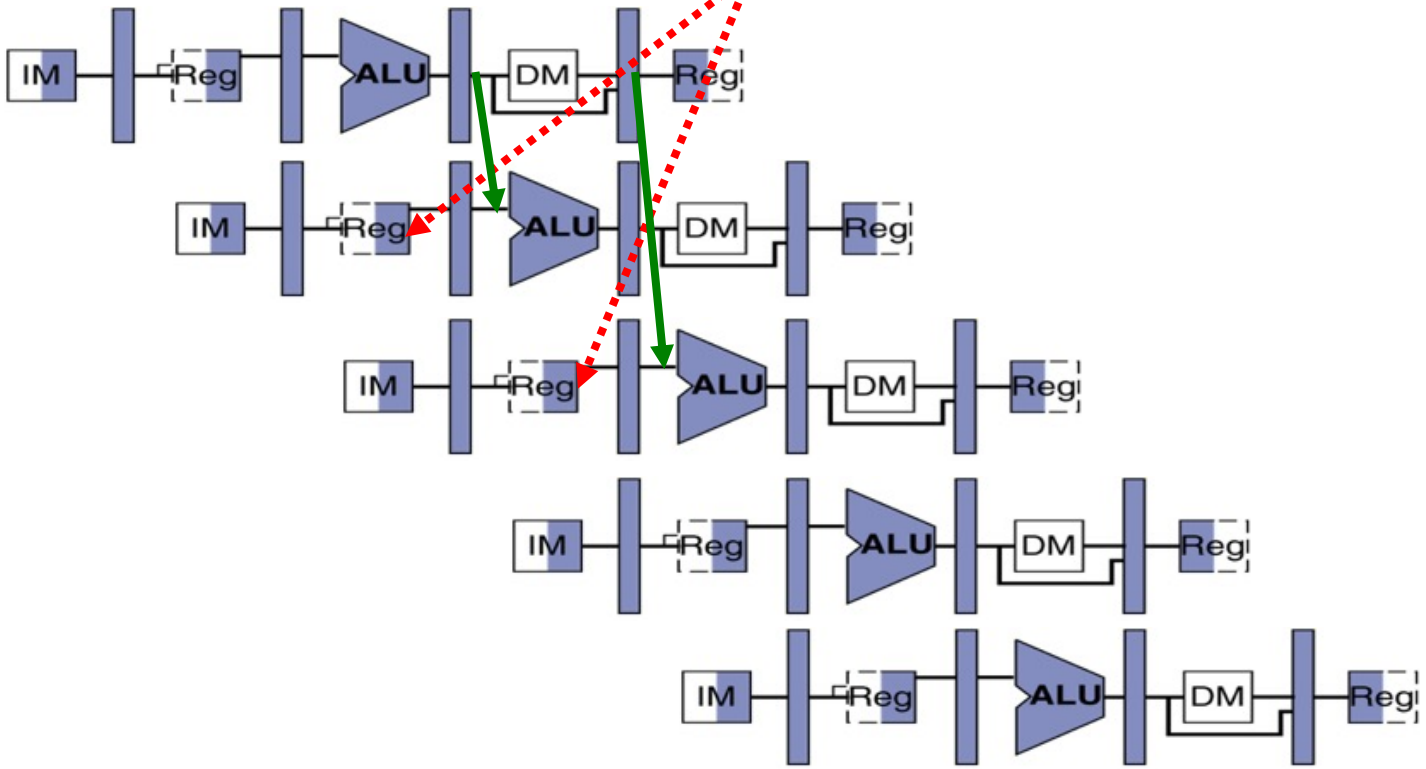
add t0, t1, t2

or t3, t0, t5

sub t6, t0, t3

xor t5, t1, t0

sw t0, 8(t3)



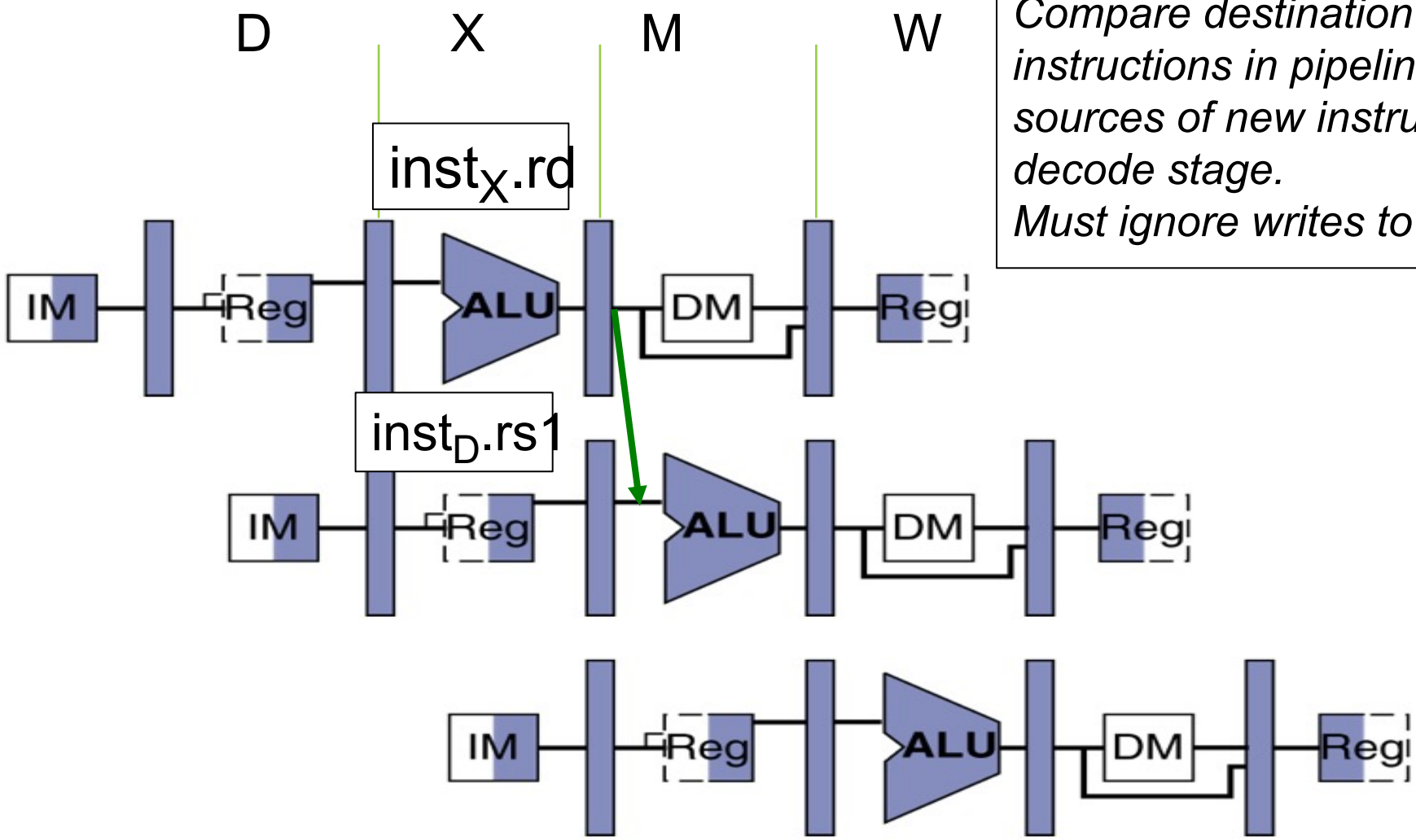
Forwarding: grab operand from pipeline stage, rather than register file

Detect Need for Forwarding (example)

add t0, t1, t2

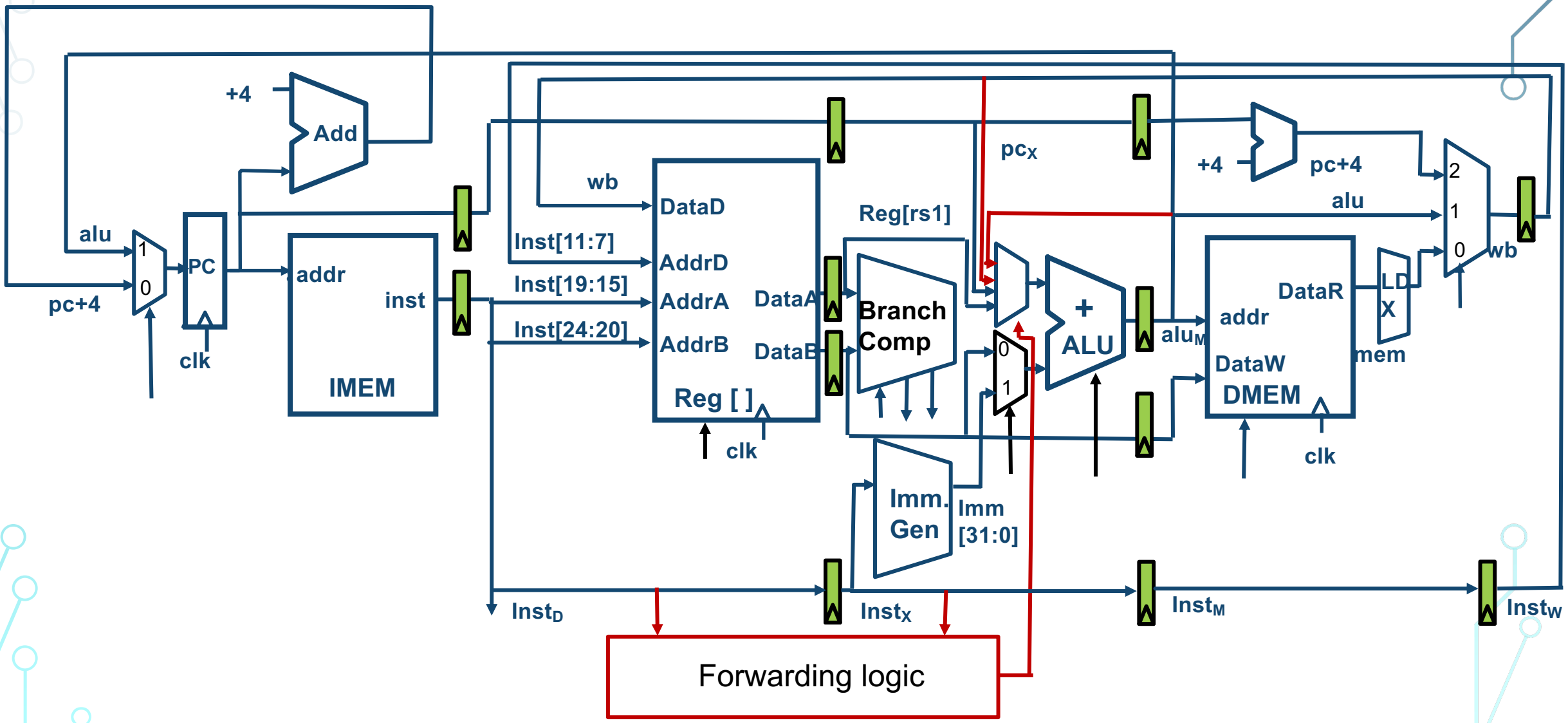
or t3, t0, t5

sub t6, t0, t3



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

ForwardingA Path

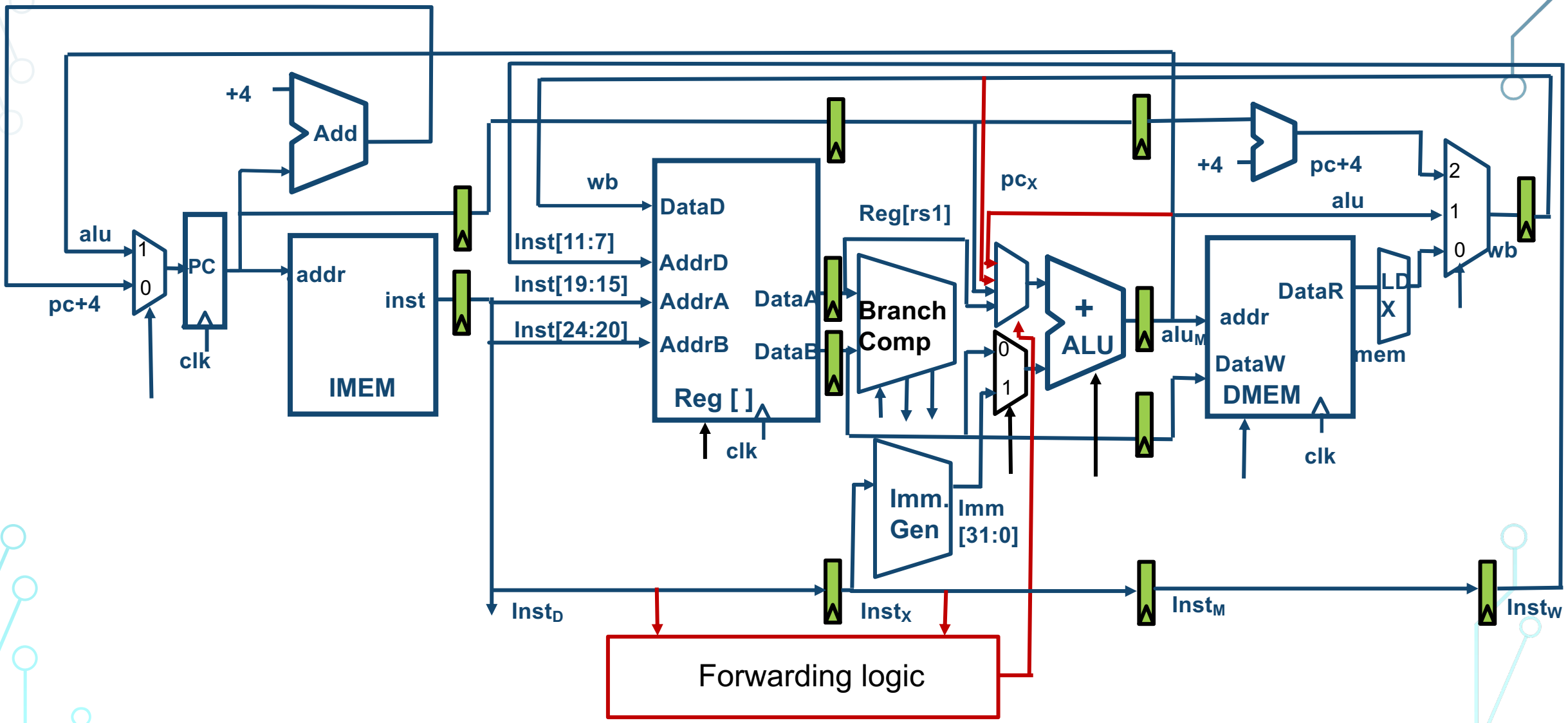


Administrivia

- Midterm: 3/1, in class
  - Covers material up to this week.
  - One double-sided page of hand-written notes
    - RISC-V green card will be provided.
  - Review session next Tuesday 5-7pm.
    - Cory 540
- No new lab this week.
- New homework.

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Pipelined Datapath: When is the target PC calculated?



Control Hazards

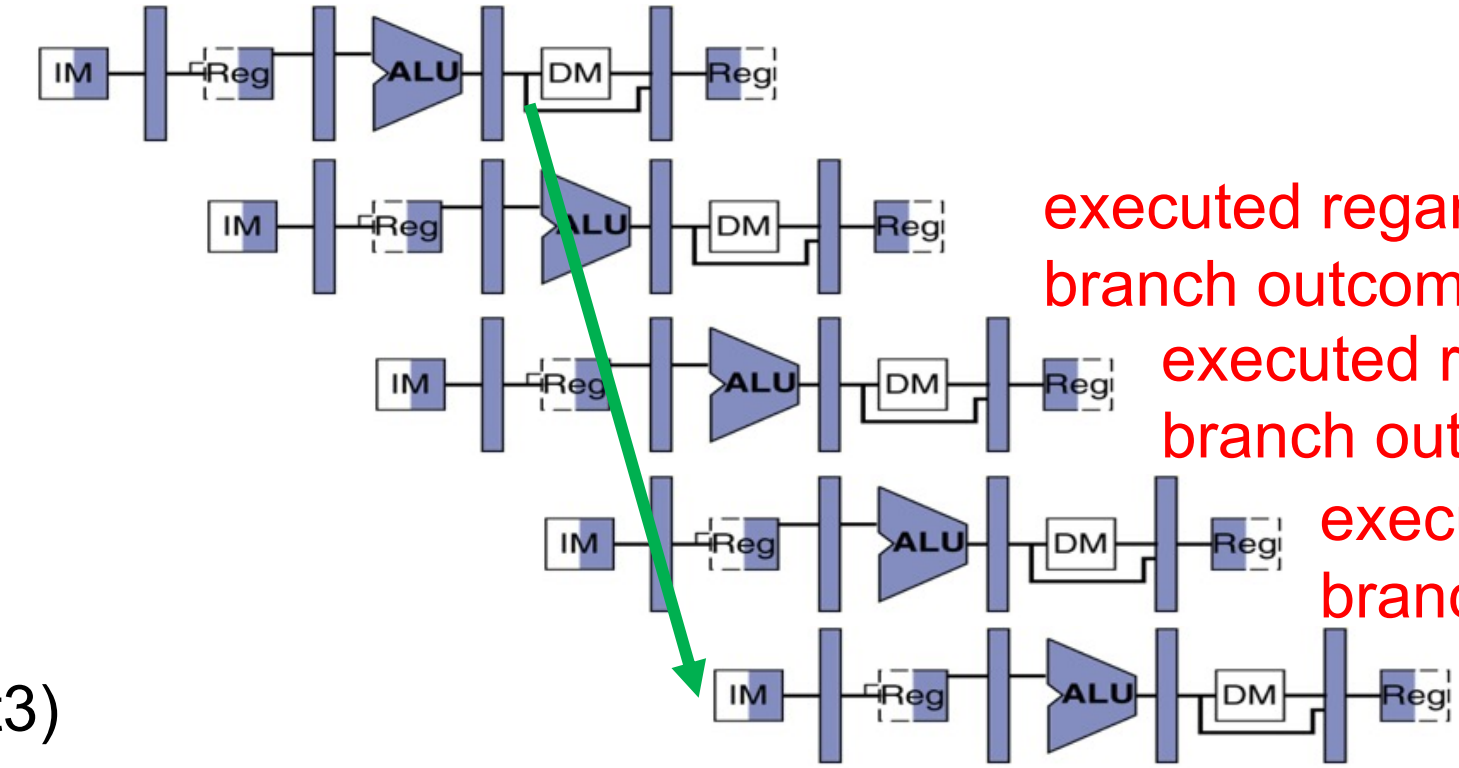
beq t0, t1, Label

sub t2, s0, t5

or t6, s0, t3

xor t5, t1, s0

Label: sw s0, 8(t3)



executed regardless of branch outcome!

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

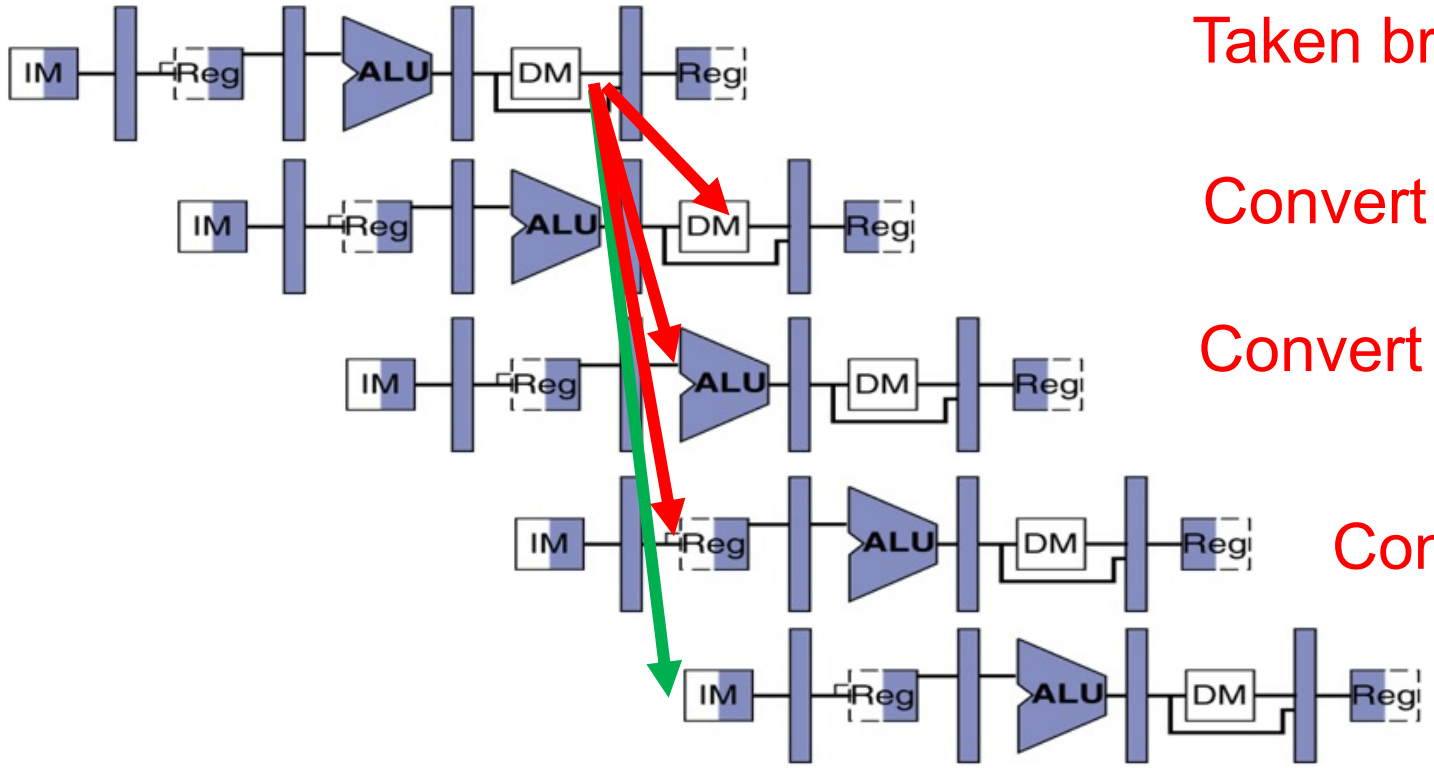
beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

Add t2, t3, s0

label: xxxxxx



Taken branch

Convert to NOP

Convert to NOP

Convert to NOP

PC updated reflecting branch outcome

Reducing Branch Penalties

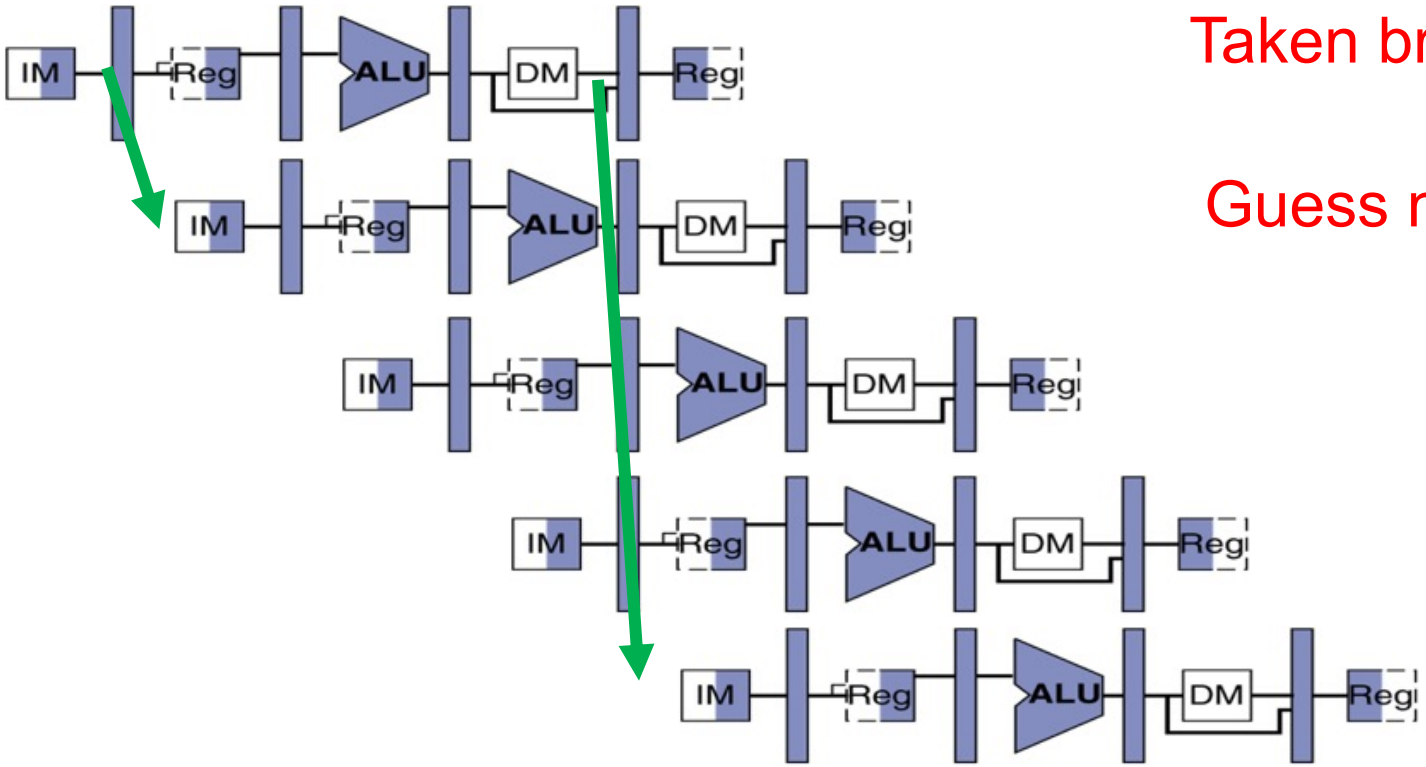
- In our datapath, every taken branch in simple pipeline costs 3 dead cycles
- To improve performance, use “branch prediction” to guess which way branch will go earlier in pipeline
- Only flush pipeline if branch prediction was incorrect

Branch Prediction

beq t0, t1, label

label: …..

…..



Taken branch

Guess next PC!

Check guess correct

Summary

- We have covered single-cycle, multi-cycle and pipelined datapaths
- Pipeline Hazards:
  - Structural
  - Data
  - Control hazards
- Need to be understood and handled appropriately
  - More resources
  - Forwarding
  - Stall