UNIVERSITY of WASHINGTON

Review of Last Lecture

- Implementing controller for your datapath
  - Take decoded signals from instruction and generate control signals
- Pipelining improves performance by exploiting Instruction Level Parallelism
  - 5-stage pipeline for RISC-V: IF, ID, EX, MEM, WB
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
  - What can go wrong???

UNIVERSITY of WASHINGTON

Agenda

Hazards Ahead!

- RISC-V Pipeline
- Hazards
  - Structural
  - Data
    - R-type instructions
    - Load
  - Control
- Superscalar processors

UNIVERSITY of WASHINGTON

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 write

3) Control hazard

- Flow of execution depends on previous instruction

UNIVERSITY of WASHINGTON

Agenda

- RISC-V Pipeline
- Hazards
  - Structural
  - Data
    - R-type instructions
    - Load
  - Control
- Superscalar processors

UNIVERSITY of WASHINGTON

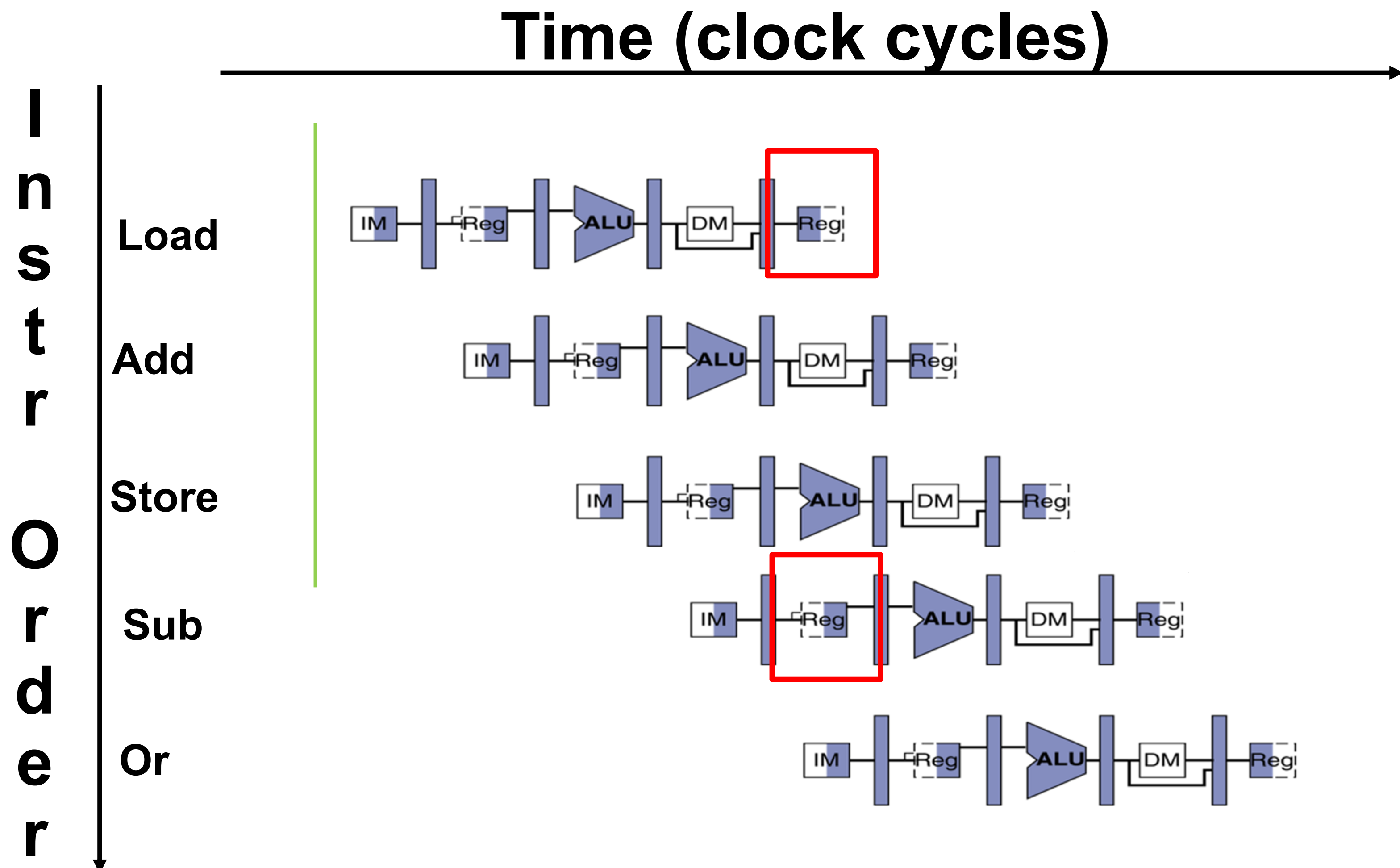
Structural Hazard

- **Problem:** Two or more instructions in the pipeline compete for access to a single physical resource
- **Solution 1:** Instructions take turns using resource, some instructions have to stall (wait)
- **Solution 2:** Add more hardware to machine
- <u>Can always solve a structural hazard by adding more hardware</u>

UNIVERSITY of WASHINGTON

Structural Hazard: Regfile!

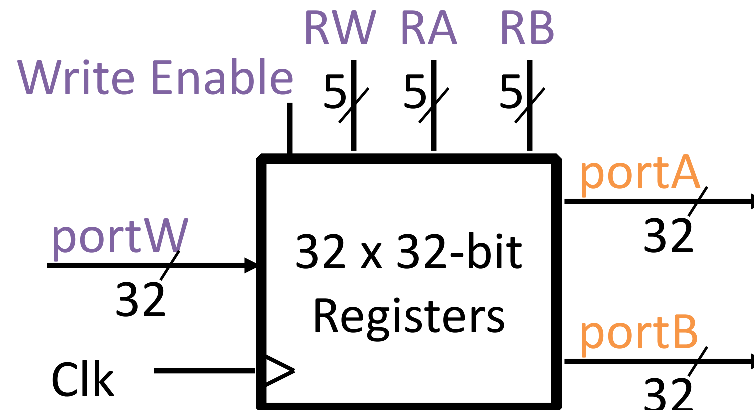
- RegFile: Used in ID and WB!



UNIVERSITY of WASHINGTON

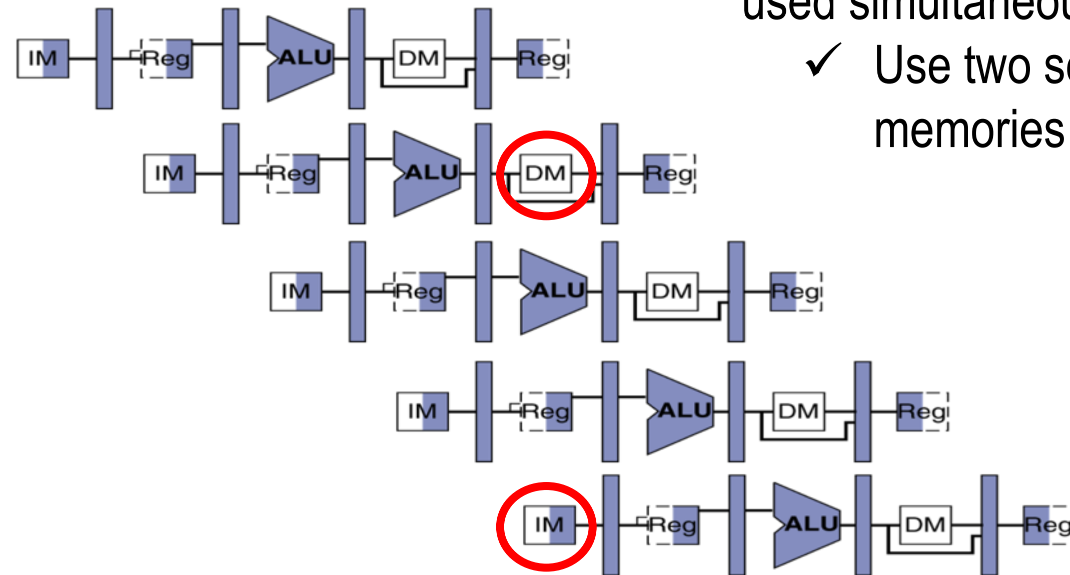
Regfile Structural Hazards

- 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



UNIVERSITY of WASHINGTON

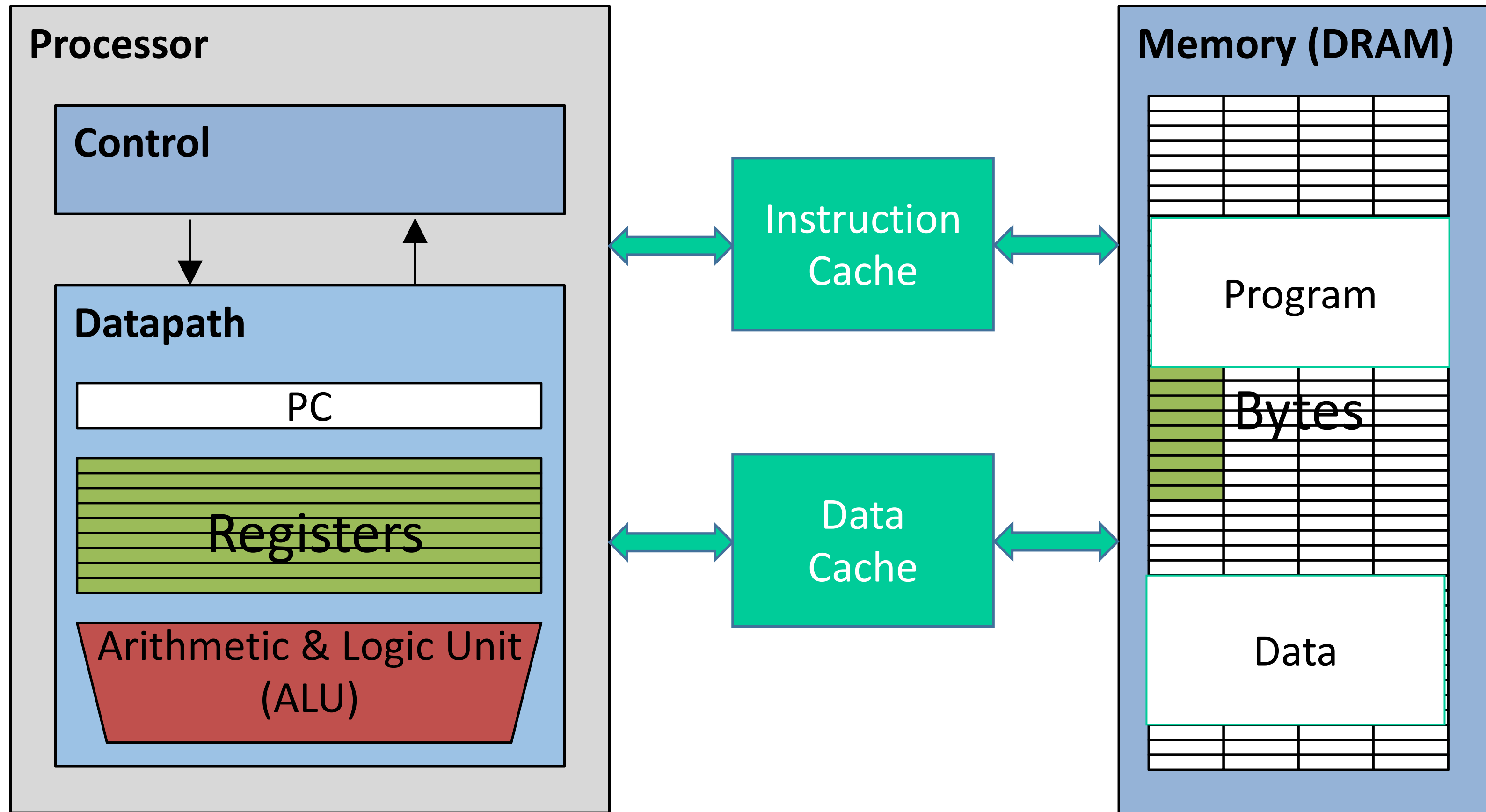
Structural Hazard: Memory Access



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

UNIVERSITY of WASHINGTON

Instruction and Data Caches



Caches: small and fast "buffer" memories

UNIVERSITY of WASHINGTON

Structural Hazards – Summary

- Conflict for use of a resource

- In RISC-V pipeline with a single memory unit
  - Load/store requires data access
  - Without separate memory units, instruction fetch would have to *stall* for that cycle
    - All other operations in pipeline would have to wait

- Pipelined datapaths require separate instruction/data memory units
  - Or separate instruction/data caches

- RISC ISAs (including RISC-V) designed to avoid structural hazards
  - e.g. at most one memory access/instruction

UNIVERSITY of WASHINGTON

Agenda

- RISC-V Pipeline
- Hazards
  - Structural
  - Data
    - R-type instructions
    - Load
  - Control
- Superscalar processors

UNIVERSITY of WASHINGTON

2. Data Hazards (1/2)

- Consider the following sequence of instructions:

```
add  s0, s1, s2
sub  s4, s0, s3
and  s5, s0, s6
or   s7, s0, s8
xor  s9, s0, s10
```

Stored during WB (destination register, e.g. s9)

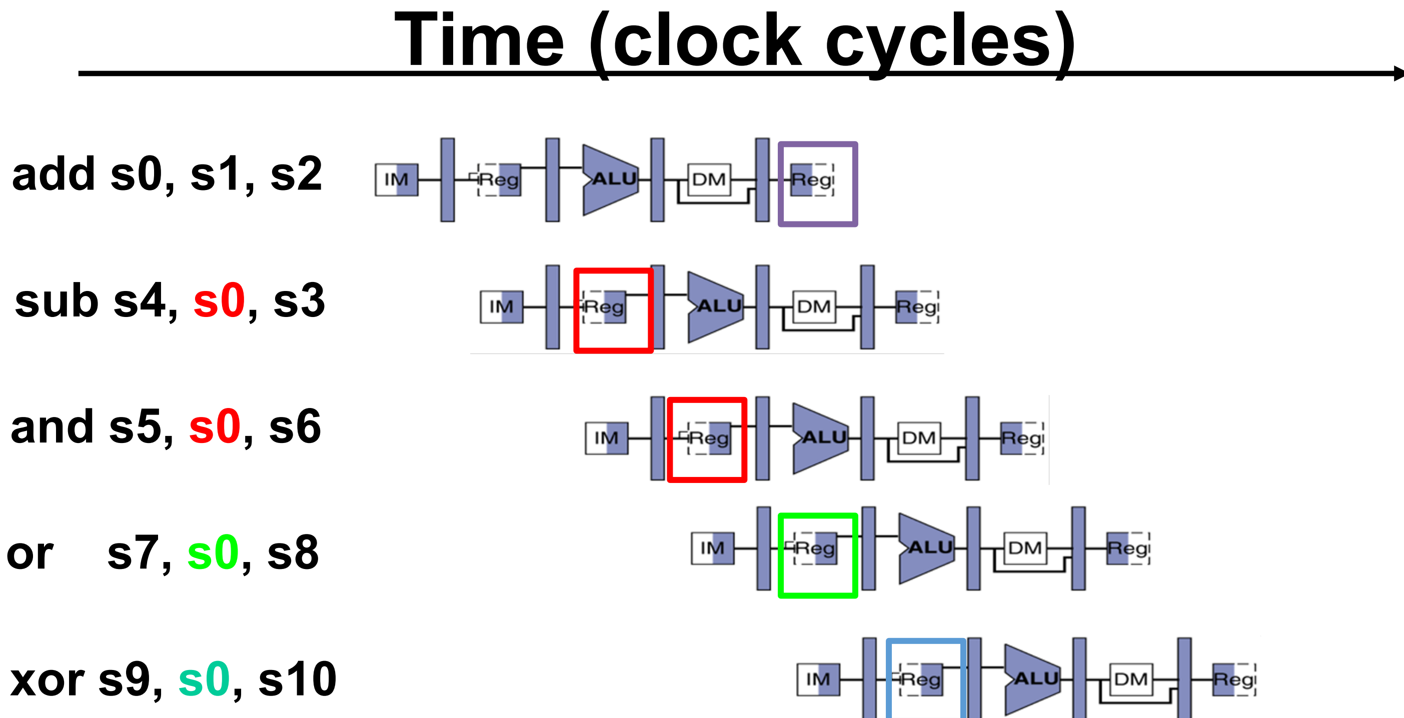
Read during ID (source registers, e.g. s0, s10)

UNIVERSITY of WASHINGTON

2. Data Hazards (2/2)

Identifying data hazards:

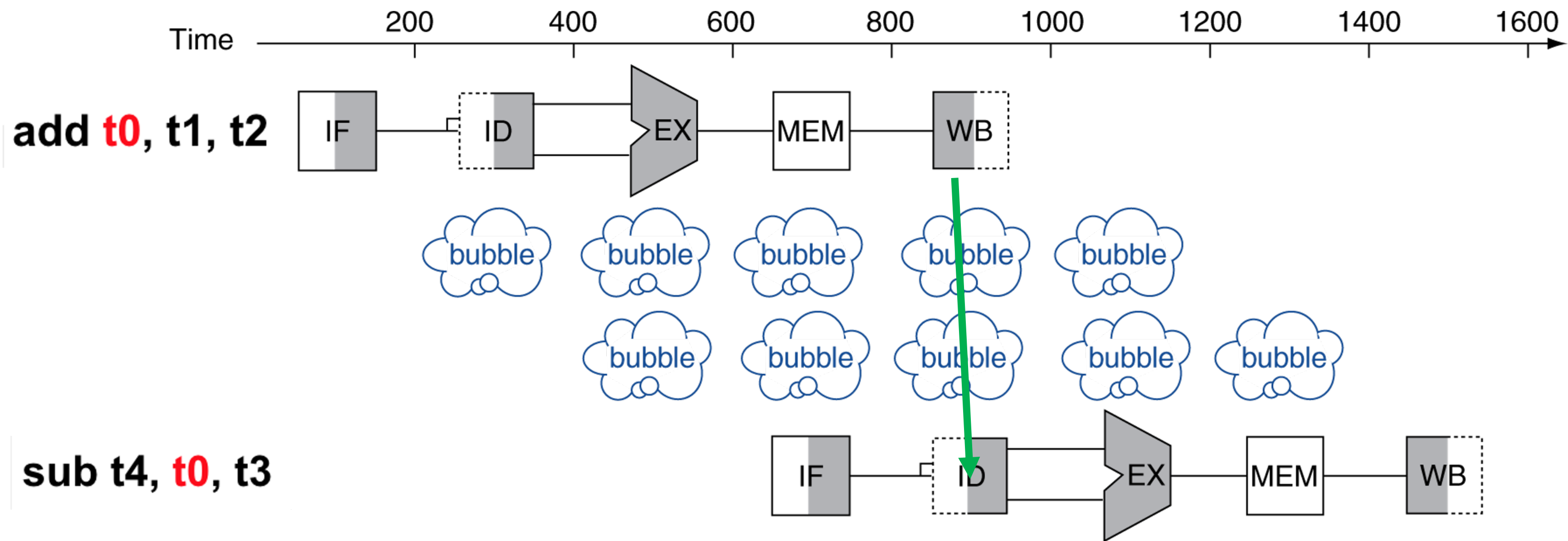
- Where is data **WRITTEN**?
- Where is data **READ**?
- Does the WRITE happen AFTER the READ?



UNIVERSITY of WASHINGTON

Solution 1: Stalling

- Problem: Instruction depends on result from previous instruction
  - add s0, s1, s2
    sub s4, s0, s3



- Bubble:
  - effecsively NOP: affecsed pipeline ssages do “nothing” (add x0 x0 x0)

UNIVERSITY of WASHINGTON

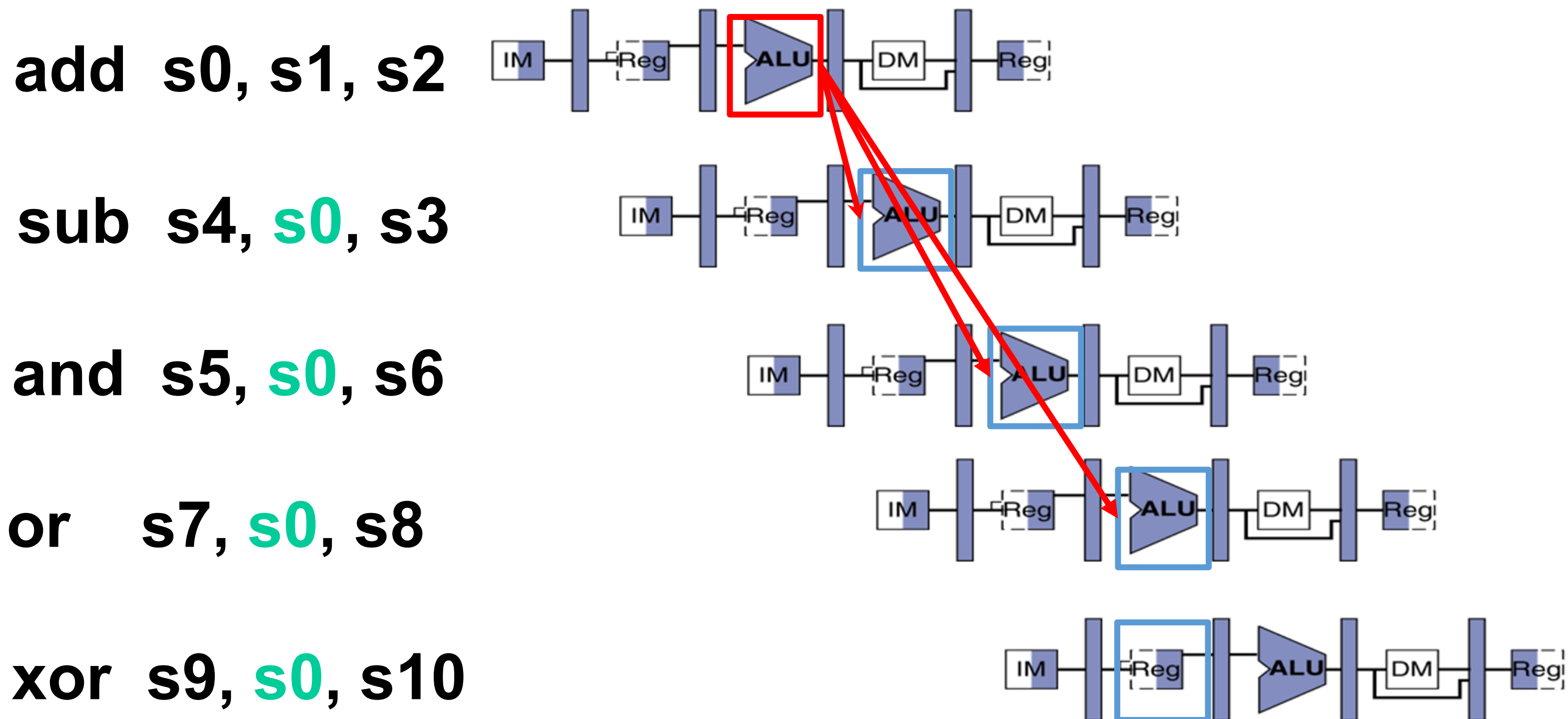
Stalls and Performance

- Stalls reduce performance
  - Decrease throughput of "valid" or useful instructions
  - Can also be seen as increasing the latency of our stalled instruction
- But stalls are required to get correct results
- Compiler can arrange code to avoid hazards and stalls!
  - Requires knowledge of the pipeline structure, and knowledge of instruction interactions

UNIVERSITY of WASHINGTON

Data Hazard Solution: Forwarding

- Forward result as soon as it is available, even though it's not stored in RegFile yet

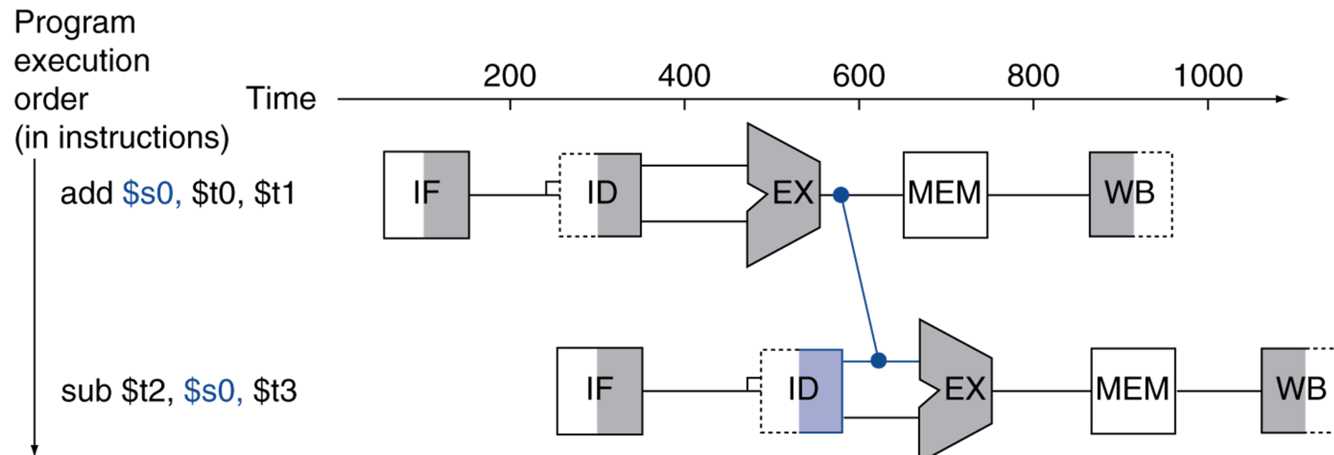


Forwarding: grab operand from pipeline stage, rather than register file

UNIVERSITY of WASHINGTON

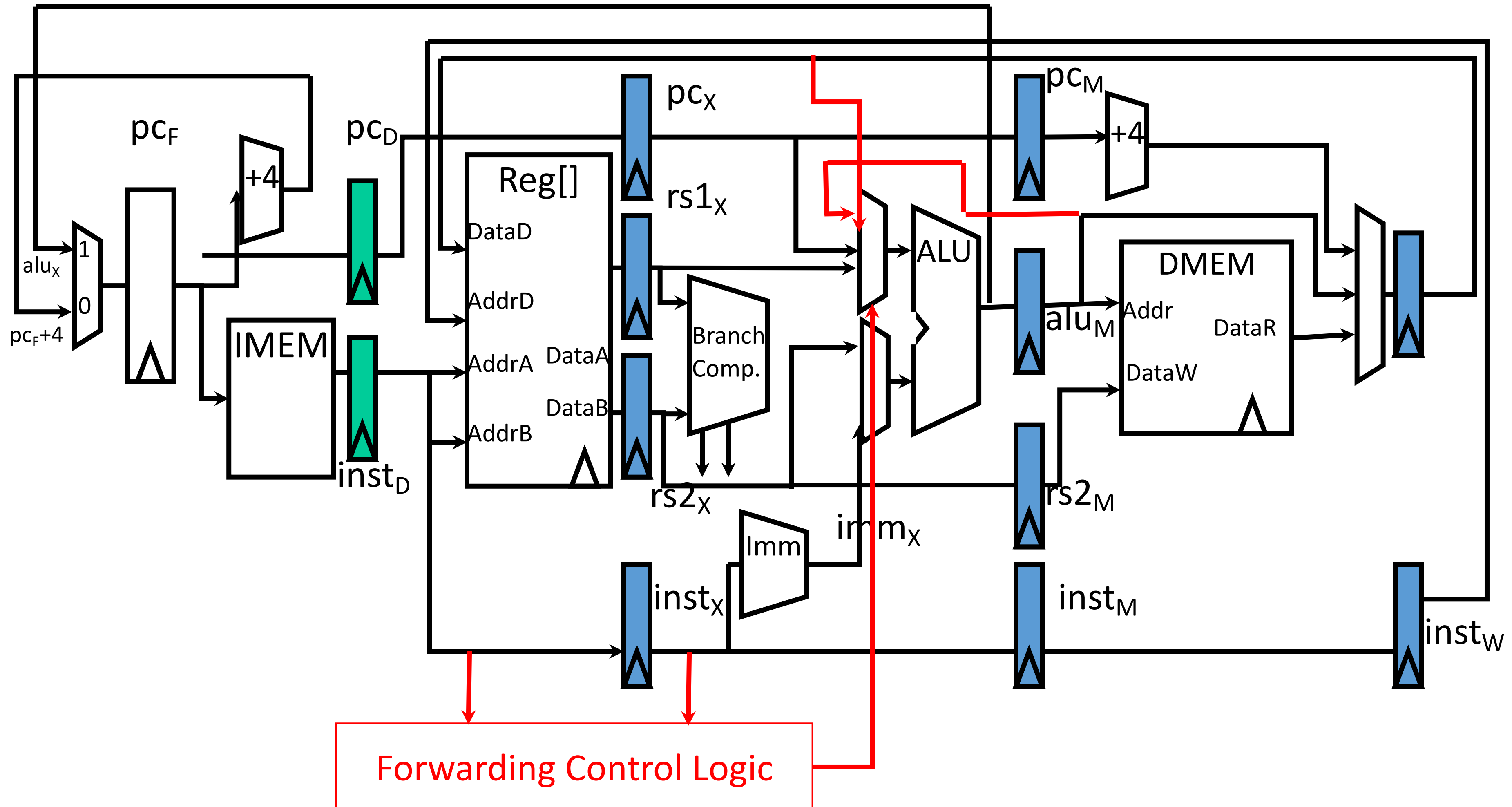
Forwarding (aka Bypassing)

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra hardware in the datapath (and extra control!)



UNIVERSITY of WASHINGTON

Forwarding Path



UNIVERSITY of WASHINGTON

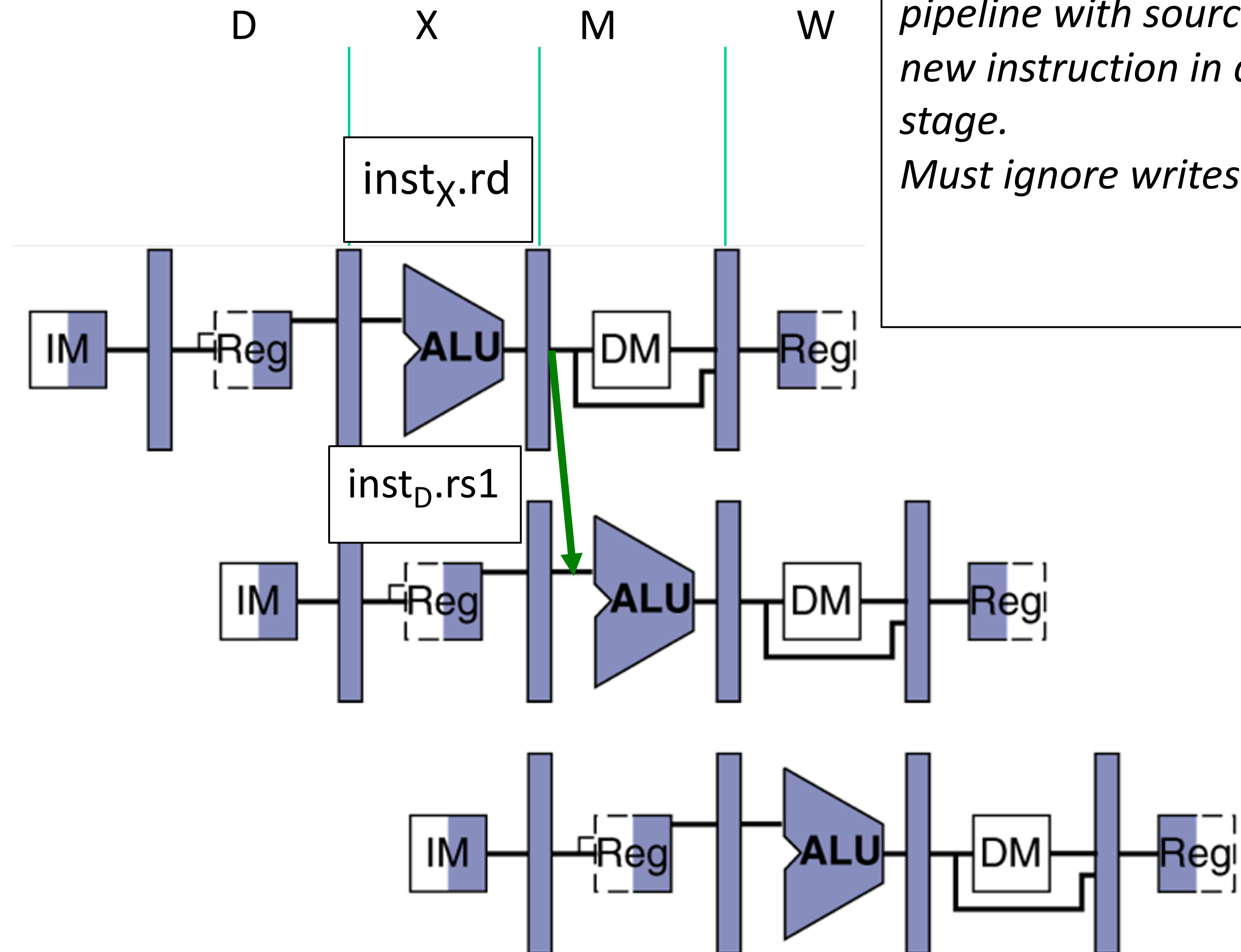
Detect Need for Forwarding (example)

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

add s0, s1, s2

or s3, s0, s5

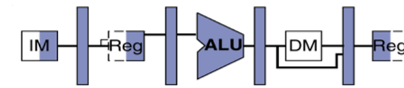
sub s6, s0, s3

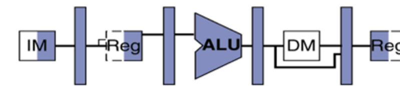


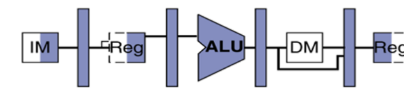
UNIVERSITY of WASHINGTON

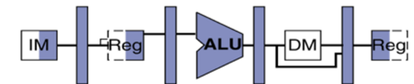
In our 5-stage pipeline, how many subsequent instructions do we need to look at to detect data hazards for this **add**? Assume we have double-pumping.

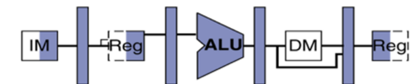
A) 1 instruction
B) 2 instructions
C) 3 instructions
D) 4 instructions
E) 5 instructions

add s0, s1, s2


sub s4, s0, s3


and s5, s0, s6


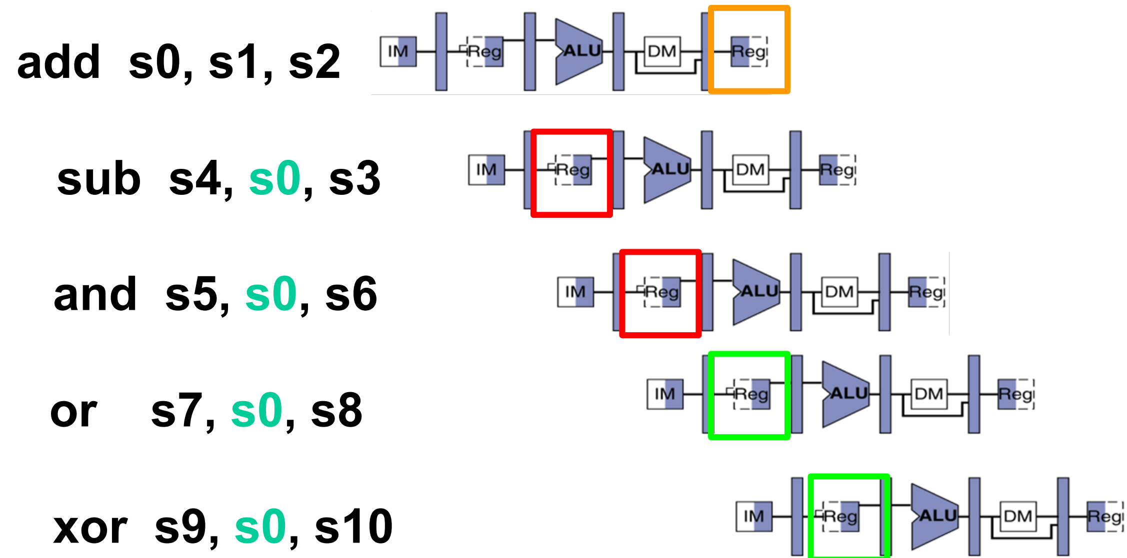
or s7, s0, s8


xor s9, s0, s10


UNIVERSITY of WASHINGTON

In our 5-stage pipeline, how many subsequent instructions do we need to look at to detect data hazards for this add? Assume we have double-pumping.

A) 1 instruction
B) 2 instructions
C) 3 instructions
D) 4 instructions
E) 5 instructions



UNIVERSITY of WASHINGTON

Agenda

- RISC-V Pipeline
- Hazards
  - Ssructural
  - Data
    - R-type instructions
    - Load
  - Control
- Superscalar processors

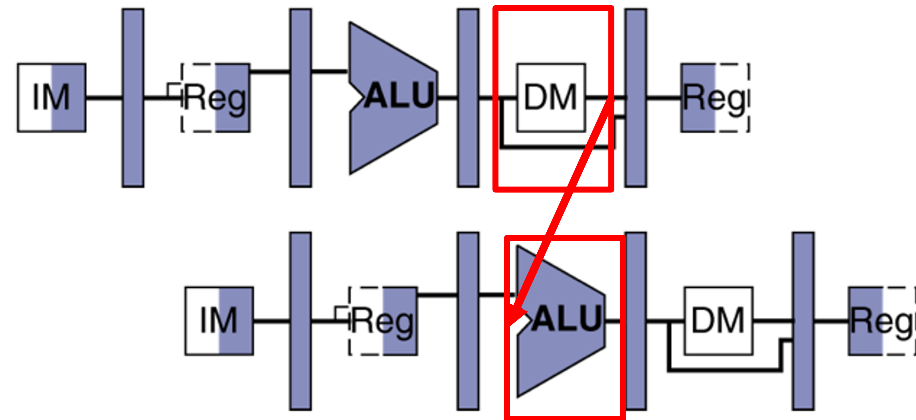
UNIVERSITY of WASHINGTON

Data Hazard: Loads (1/4)

- **Recall:** Dataflow backwards in time are hazards

lw t0, 0(t1)

sub t3, t0, t2



- Can't solve all cases with forwarding
  - Must *stall* instruction dependent on load (sub), then forward after the load is done (more hardware)

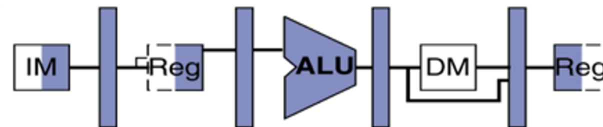
UNIVERSITY of WASHINGTON

Data Hazard: Loads (2/4)

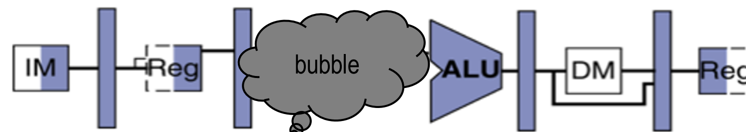
- *Hardware* stalls pipeline
  - Called "hardware interlock"

This is what happens in hardware in a "hardware interlock"

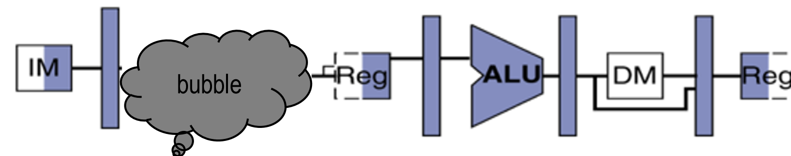
lw t0, 0(t1)



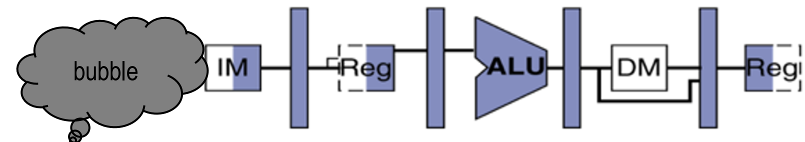
sub t3, t0, t2



and t5, t0, t4



or t7, t0, t6

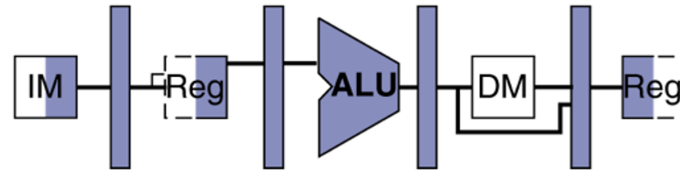


UNIVERSITY of WASHINGTON

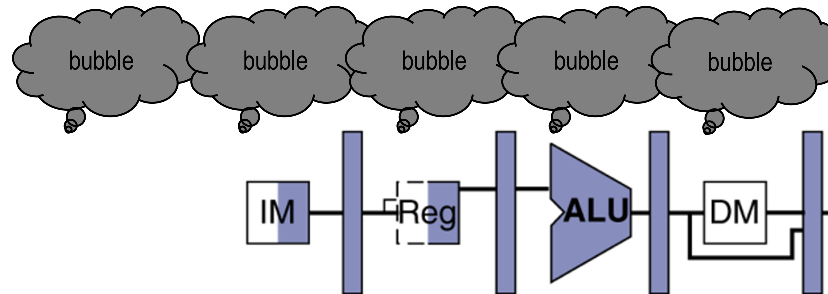
Data Hazard: Loads (3/4)

- Stall is equivalent to `nop`

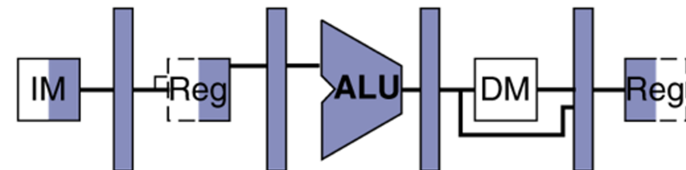
lw t0, 0(t1)



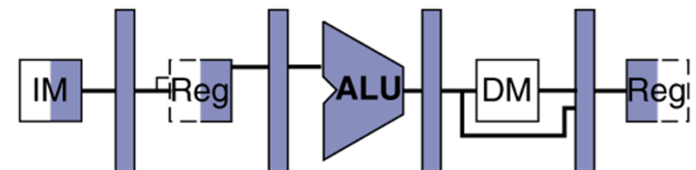
nop



sub t3, t0, t2



and t5, t0, t4



or t7, t0, t6

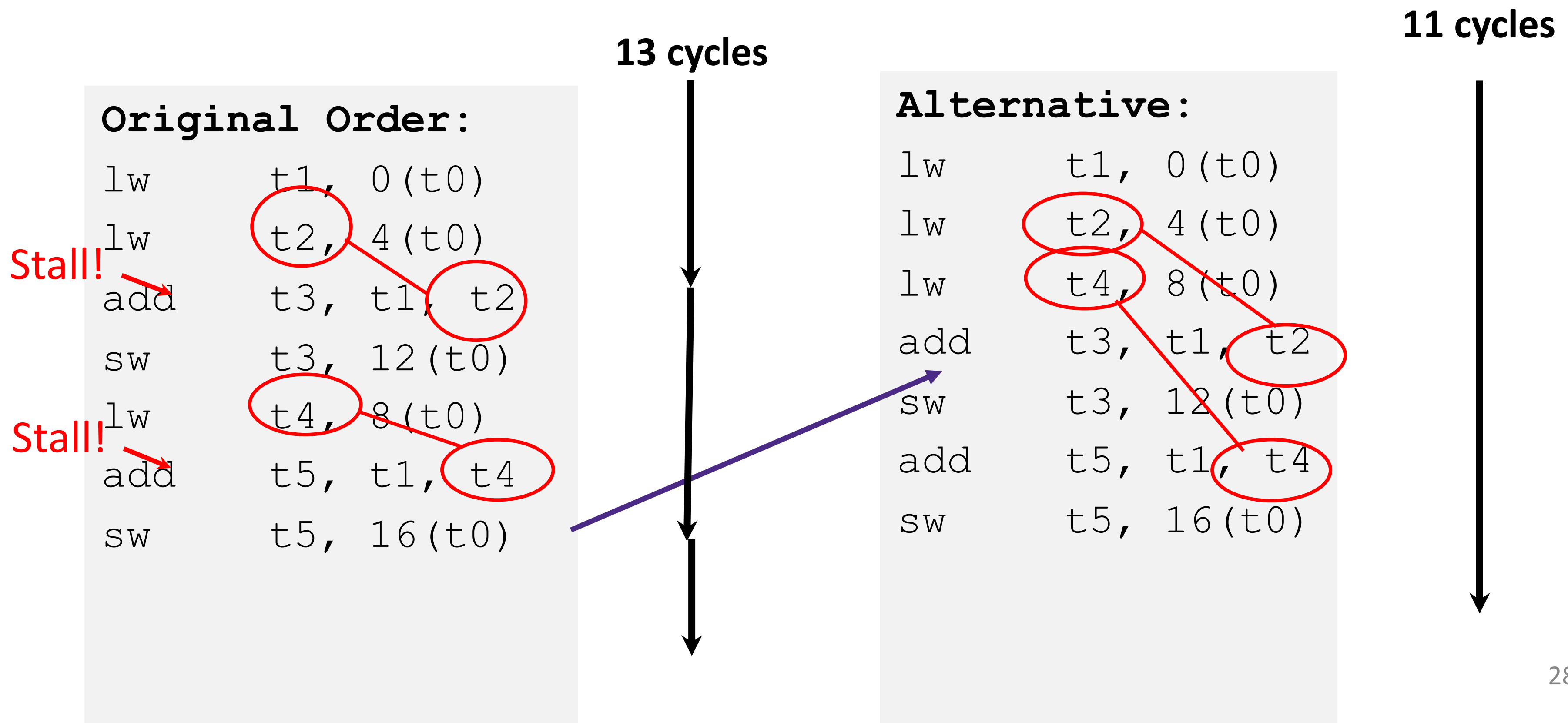
Data Hazard: Loads (4/4)

- 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:** Let the compiler/assembler put an unrelated instruction in that slot → no stall!

UNIVERSITY of WASHINGTON

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction!
- RISC-V code for D=A+B; E=A+C;



UNIVERSITY of WASHINGTON

Break!

UNIVERSITY of WASHINGTON

Agenda

- RISC-V Pipeline
- Hazards
  - Structural
  - Data
    - R-type instructions
    - Load
  - Control
- Superscalar processors

UNIVERSITY of WASHINGTON

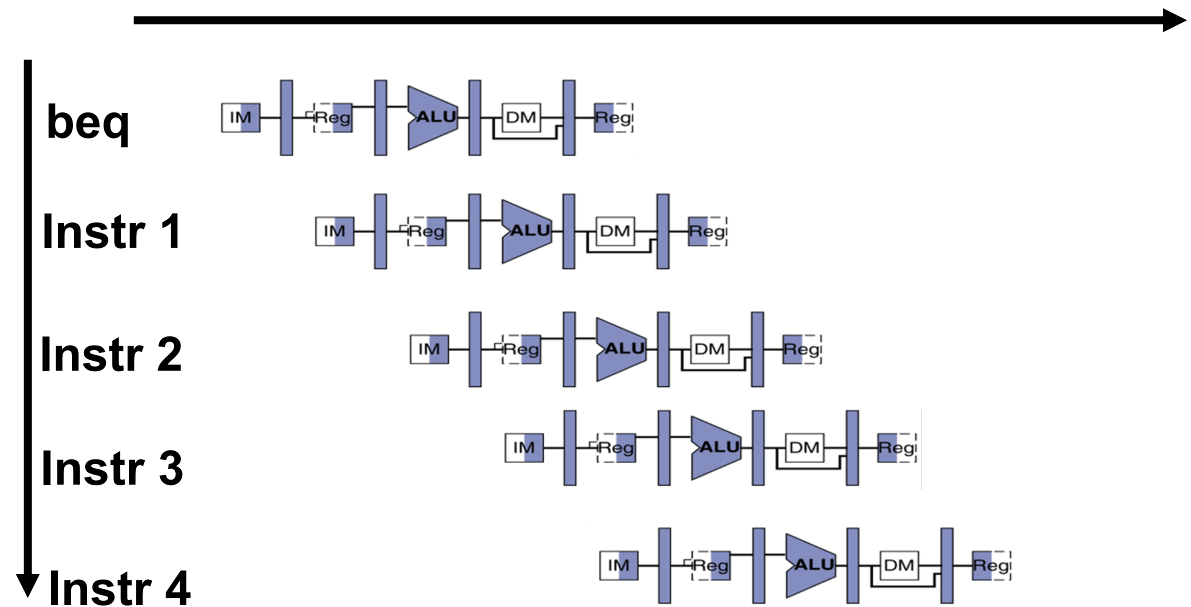
3. Control Hazards

- Branch (`beq`, `bne, ...`) determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Result isn't known until end of execute
- **Simple Solution:** Stall *or flush* on *every* branch until we have the new PC value
  - How long must we stall?

UNIVERSITY of WASHINGTON

- How many instructions after **beq** are affected by the control hazard?

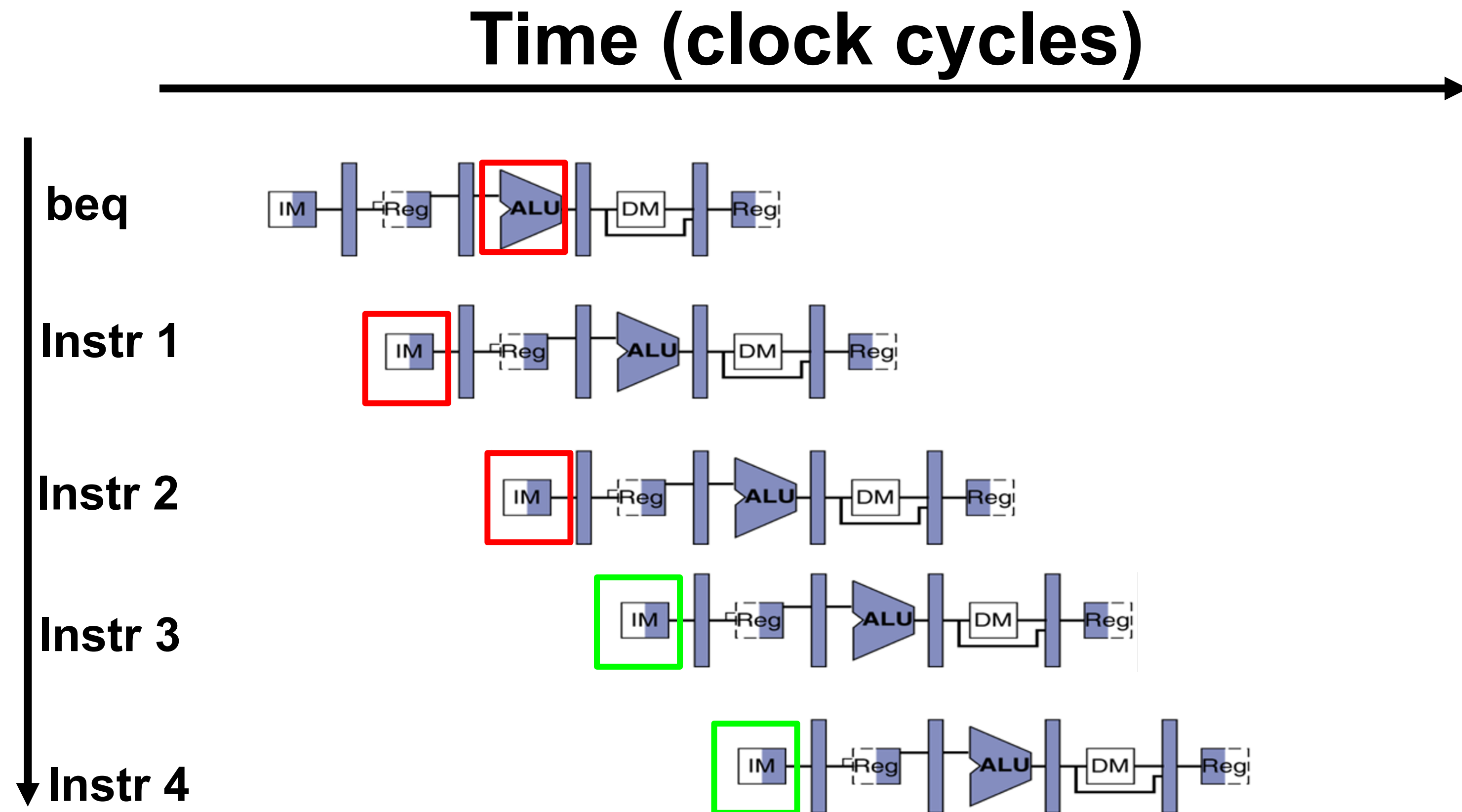
A) 1
B) 2
C) 3
D) 4
E) 5



UNIVERSITY of WASHINGTON

Branch Stall

- How many bubbles required for branch?



UNIVERSITY of WASHINGTON

3. Control Hazard: Branching

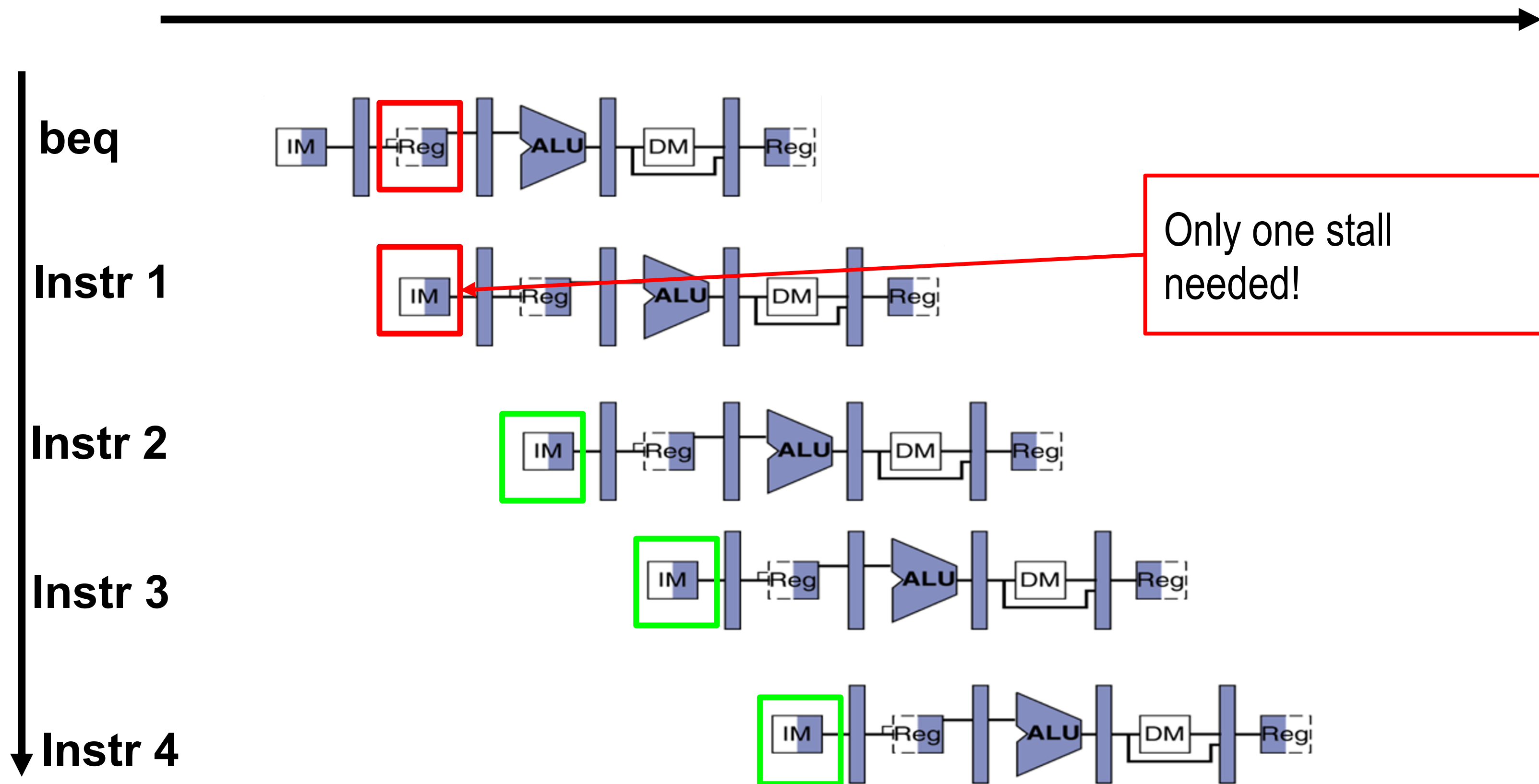
- **Option #1:** Move **branch comparator** to ID stage

- As soon as instruction is decoded, immediately make a decision and set the new value of PC
  - **Benefit:** Branch decision made in 2nd stage, so only one `nop` is needed instead of two
  - **Side Note:** Have to compute new PC value (PC + imm) in ID instead of EX
    - Adds extra copy of new-PC logic in ID stage
    - Branches are idle in EX, MEM, and WB

UNIVERSITY of WASHINGTON

Improved Branch Stall

- When is comparison result available?

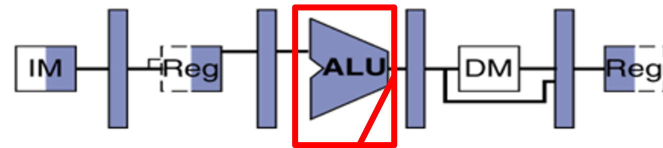


UNIVERSITY of WASHINGTON

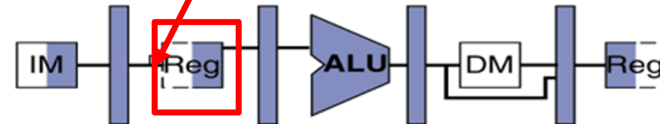
Data Hazard: **Branches!**

- **Recall:** Dataflow backwards in time are hazards

add t0, t0, t1



beq x0, t0, foo



- Now that t0 is needed earlier (ID instead of EX), we can't forward it to the beq's ID stage
  - Must *stall* after add, then forward (more hardware)

UNIVERSITY of WASHINGTON

Observations

- **Takeaway**: Moving branch comparator to ID stage would add redundant hardware and introduce new problems
- Can we work with the nature of branches?
  - 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

UNIVERSITY of WASHINGTON

Agenda

- Structural Hazards
- Data Hazards
  - Forwarding
- Administrivia
- Data Hazards (Continued)
  - Load Delay Slot
- Control Hazards
  - Branch and Jump Delay Slots
  - Branch Prediction

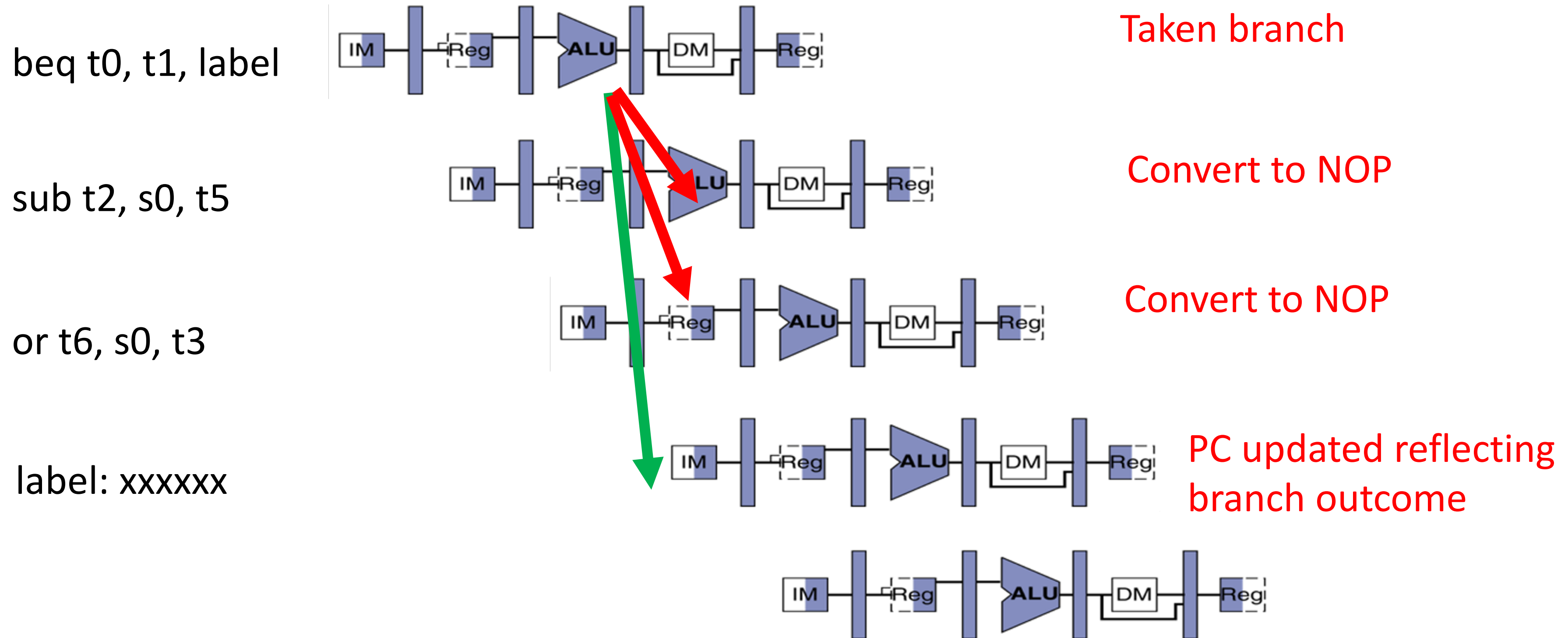
UNIVERSITY of WASHINGTON

3. Control Hazard: Branching

- **RISC-V Solution:** *Branch Prediction* – guess outcome of a branch, fix afterwards if necessary
  - Must cancel (*flush*) all instructions in pipeline that depended on guess that was wrong
  - How many instructions do we end up flushing?

UNIVERSITY of WASHINGTON

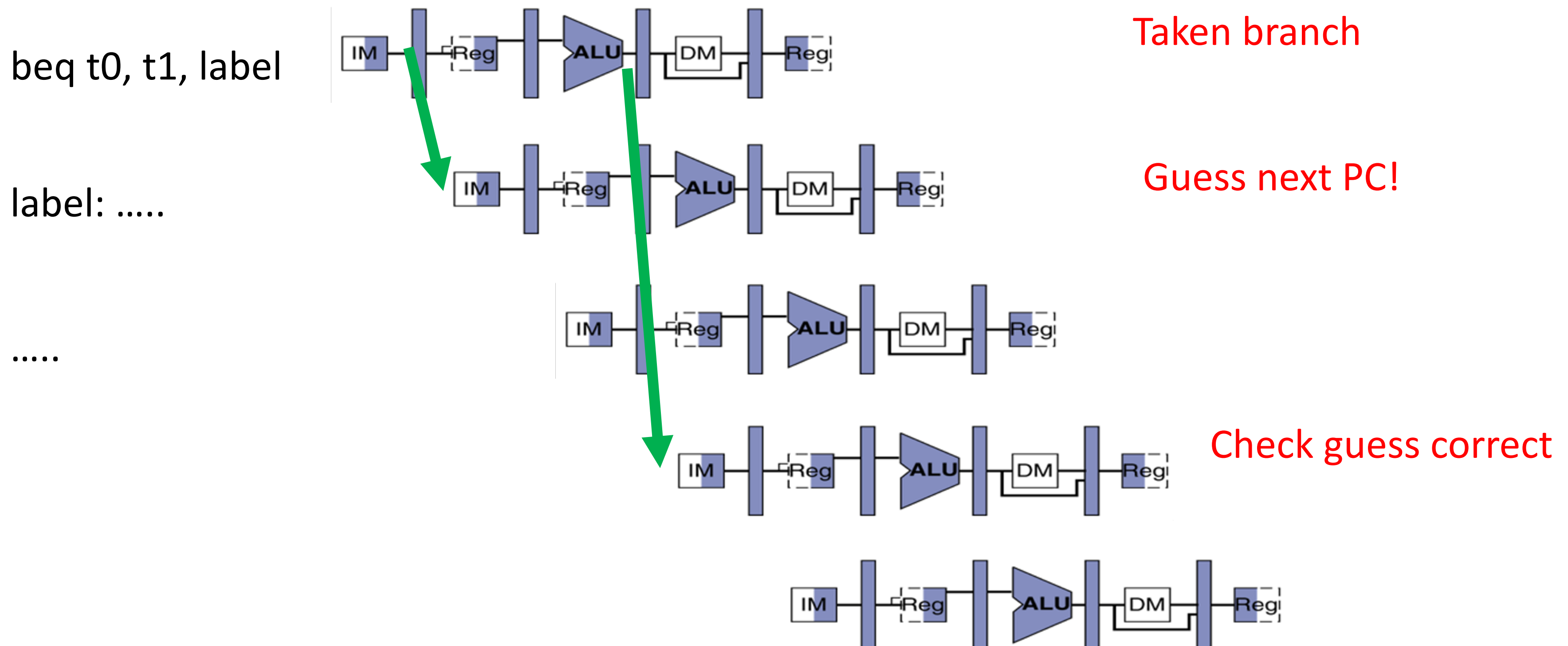
Kill Instructions after Branch if Taken



Two instructions are affected by an incorrect branch, just like we'd have to insert two NOP's/stalls in the pipeline to wait on the correct value!

UNIVERSITY of WASHINGTON

Branch Prediction



In the correct case, we don't have any stalls/NOP's at all!

Prediction, if done correctly, is better on average than stalling

UNIVERSITY of WASHINGTON

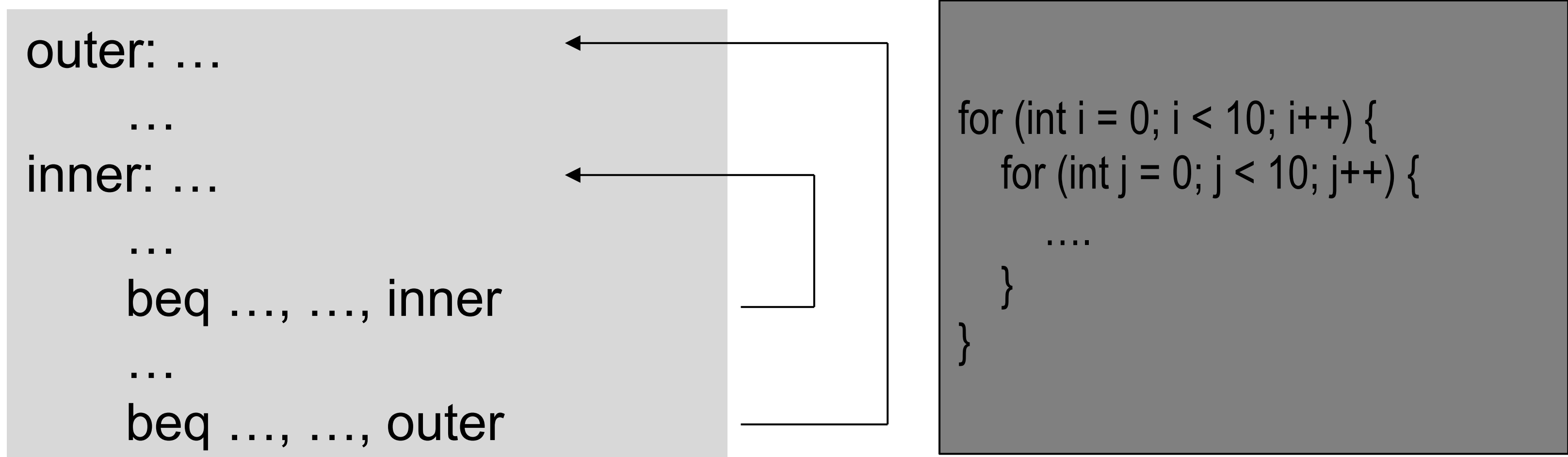
Dynamic Branch Prediction

- Branch penalty is more significant in deeper pipelines
- Use *dynamic branch prediction*
  - Have branch prediction mechanism (a.k.a. branch history table) that stores outcomes (taken/not taken) of previous branches
  - To execute a branch
    - Check table and predict the same outcome for next fetch
    - If wrong, flush pipeline and flip prediction

UNIVERSITY of WASHINGTON

1-Bit Predictor: Shortcoming

- Examine the code below, assuming both loops will be executed multiple times:



- Inner loop branches are predicted wrong twice!
  - Predict as taken on last iteration of inner loop
  - Then predict as not taken on first iteration of inner loop next time around

Question: For the code sequence below, choose the statement that best describes requirements for correctness

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

A **No stalls as is**

B **No stalls with forwarding**

C **Must stall**

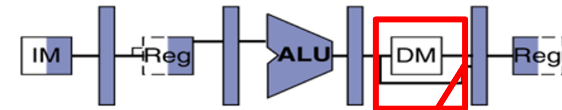
UNIVERSITY of WASHINGTON

Code Sequence 1

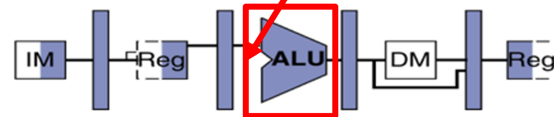
Time (clock cycles)

I n s t r O r d e r

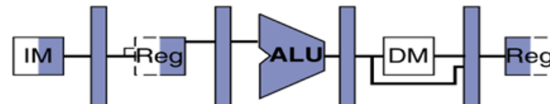
lw



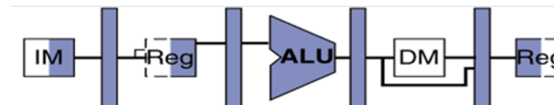
add



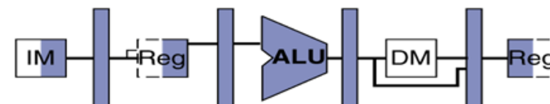
instr



instr



instr



Must stall at least once! Forwarding doesn't help us here!

Question: For the code sequence below, choose the statement that best describes requirements for correctness

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

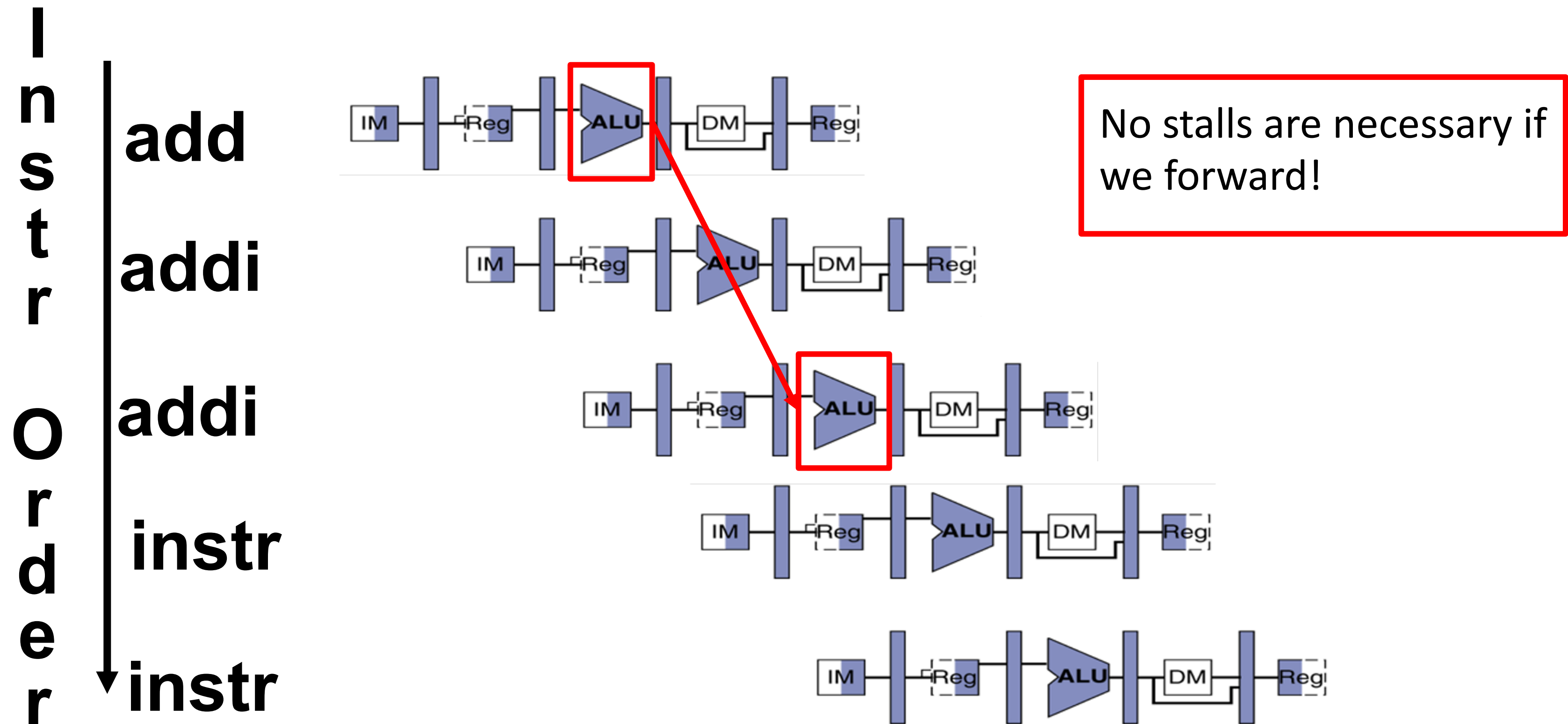
A No stalls as is

B No stalls with forwarding

C Must stall

UNIVERSITY of WASHINGTON

Code Sequence 2



Question: For the code sequence below, choose the statement that best describes requirements for correctness

3:

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

A **No stalls as is**

B **No stalls with forwarding**

C **Must stall**

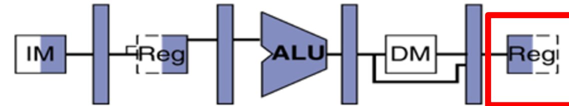
UNIVERSITY of WASHINGTON

Code Sequence 3

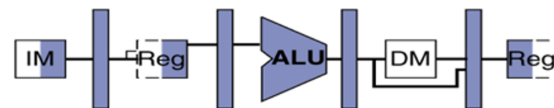
Time (clock cycles)

Instr Order

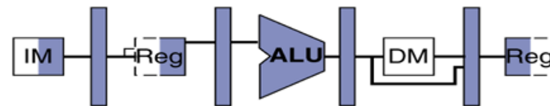
addi



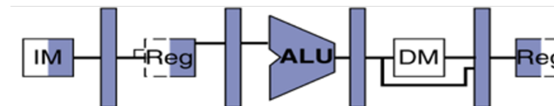
addi



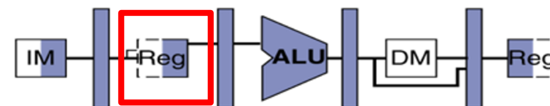
addi



addi



addi



No stalls as is! Our reading takes place after our write has finished!

UNIVERSITY of WASHINGTON

Summary

- Hazards reduce effectiveness of pipelining
  - Cause stalls/bubbles
- Structural Hazards
  - Conflict in use of a datapath component
- Data Hazards
  - Need to wait for result of a previous instruction
- Control Hazards
  - Address of next instruction uncertain/unknown
- Superscalar processors use multiple execution units for additional instruction level parallelism
  - Performance benefit highly code dependent