

PIPELINING: HAZARDS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Announcement

- Homework 1 submission deadline: Jan. 30th

- This lecture

- Impacts of pipelining on performance

- The MIPS five-stage pipeline

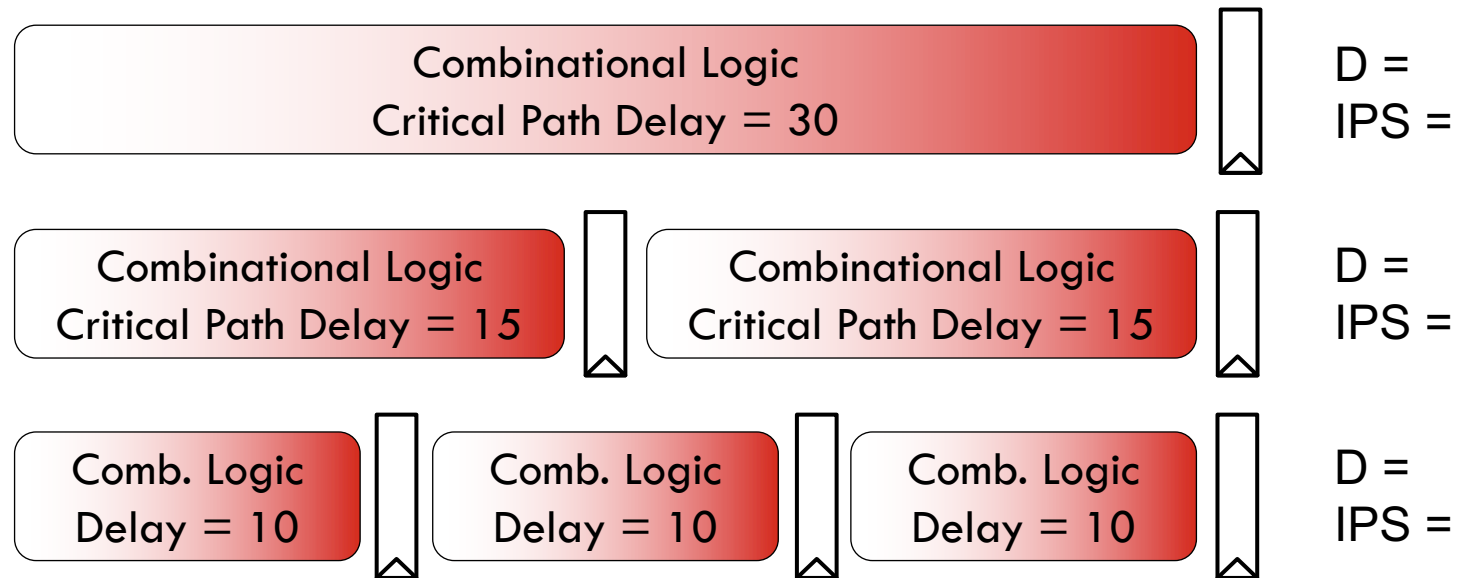
- Pipeline hazards

- Structural hazards

- Data hazards

Pipelining Technique

- Improving throughput at the expense of latency
 - ▣ Delay: $D = T + n\delta$
 - ▣ Throughput: $IPS = n/(T + n\delta)$

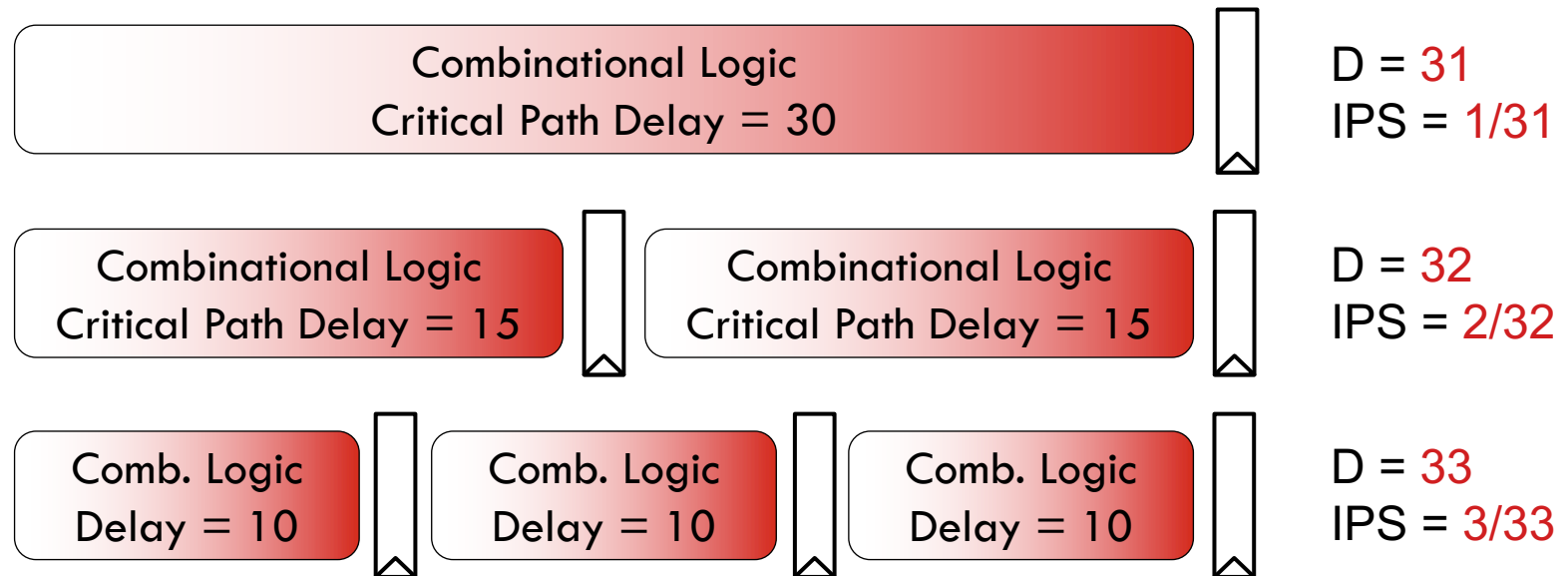


Pipelining Technique

- Improving throughput at the expense of latency

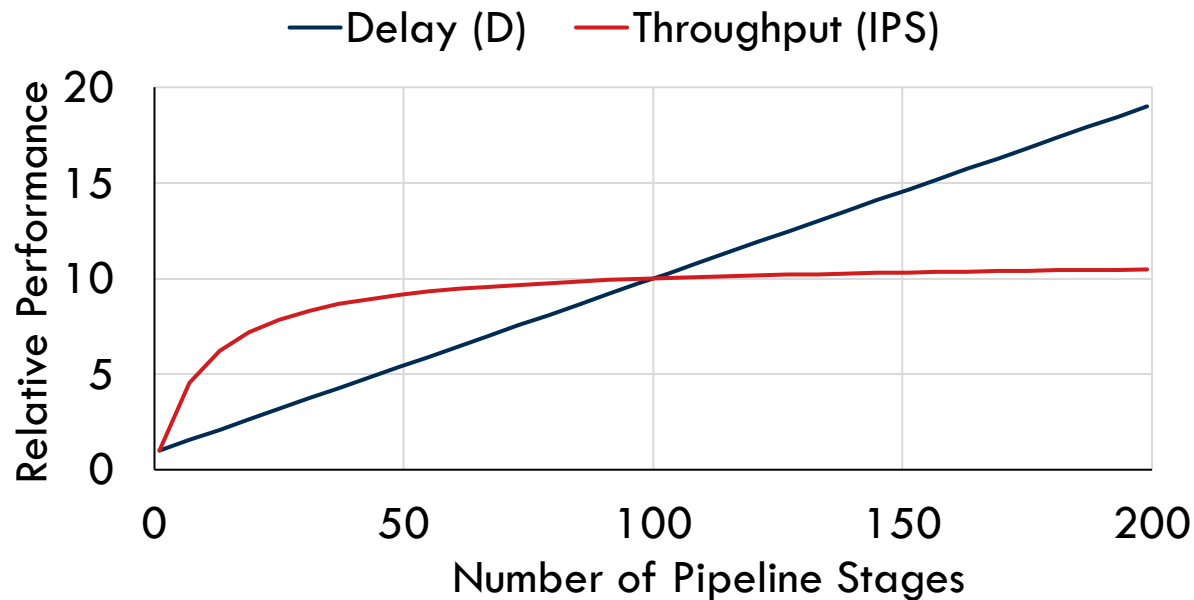
- ▣ Delay: $D = T + n\delta$

- ▣ Throughput: $IPS = n/(T + n\delta)$



Pipelining Latency vs. Throughput

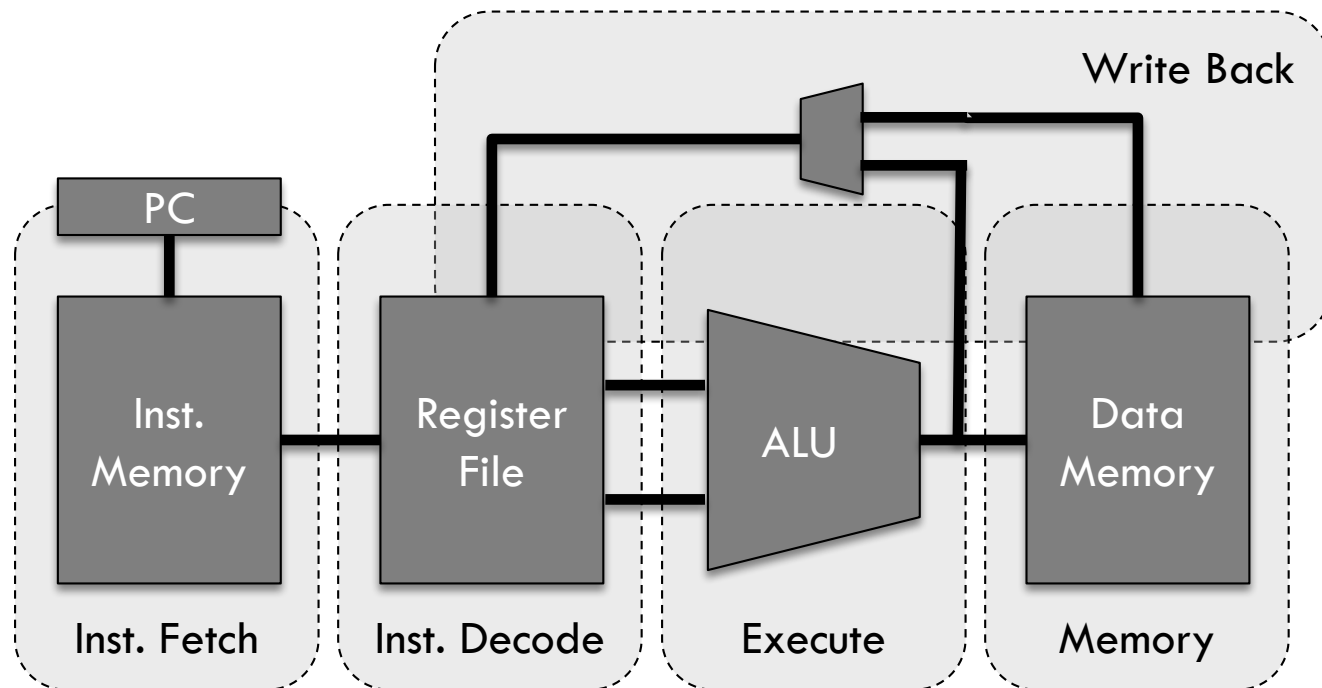
- Theoretical delay and throughput models for perfect pipelining



Five Stage MIPS Pipeline

Simple Five Stage Pipeline

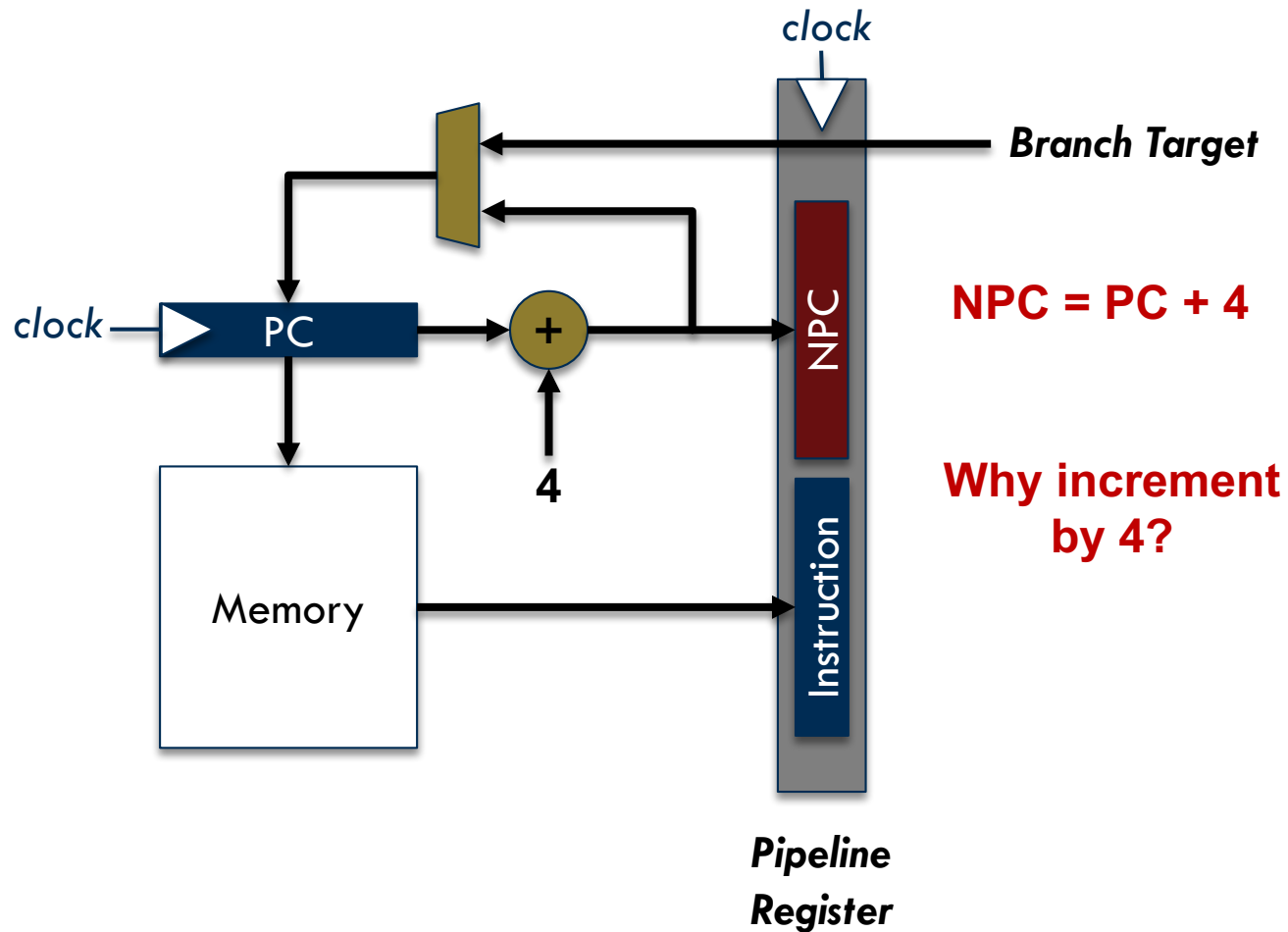
- A pipelined load-store architecture that processes up to one instruction per cycle



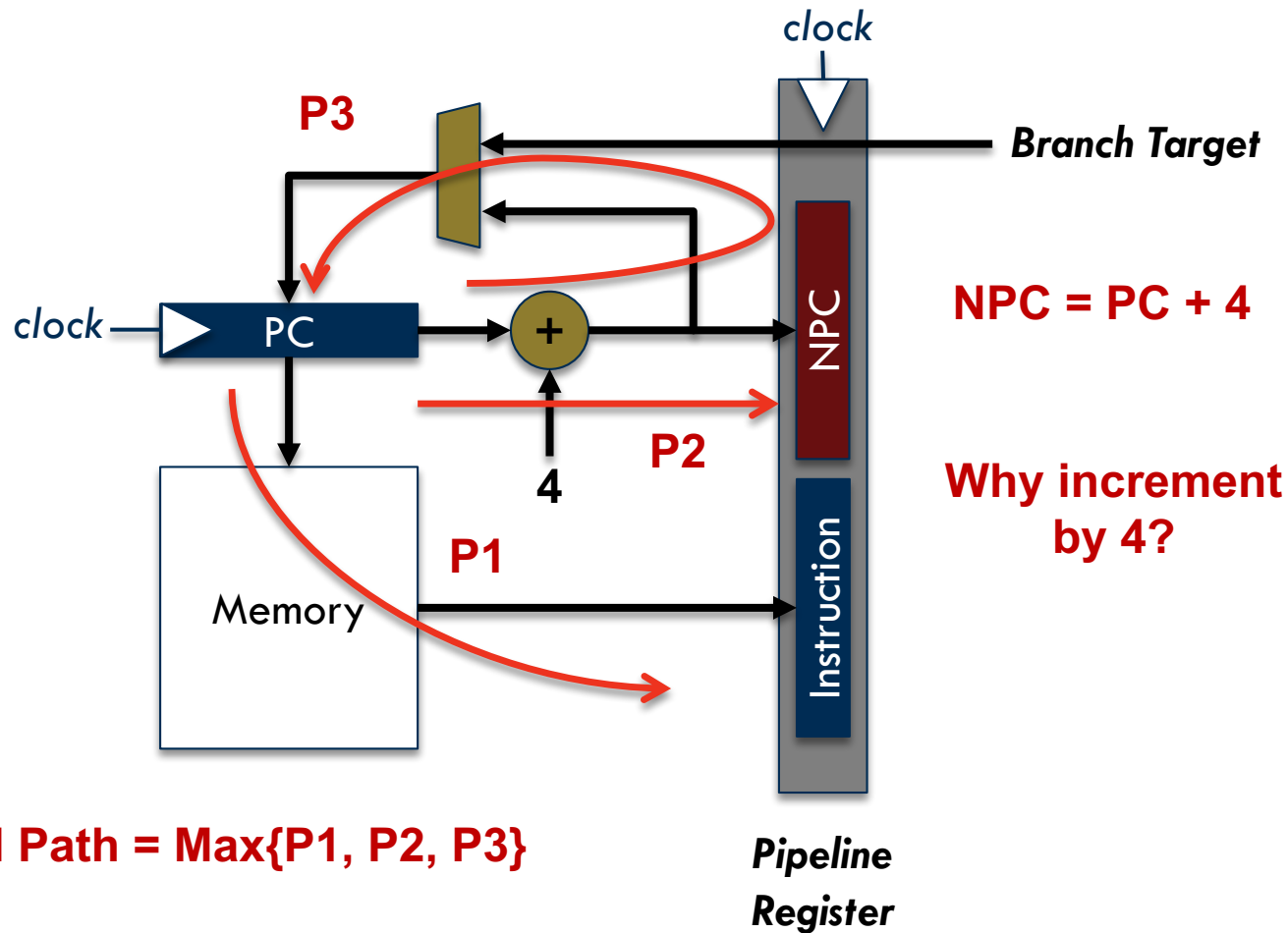
Instruction Fetch

- Read an instruction from memory (I-Cache)
 - ▣ Use the program counter (PC) to index into the I-Memory
 - ▣ Compute NPC by incrementing current PC
 - What about branches?
- Update pipeline registers
 - ▣ Write the instruction into the pipeline registers

Instruction Fetch



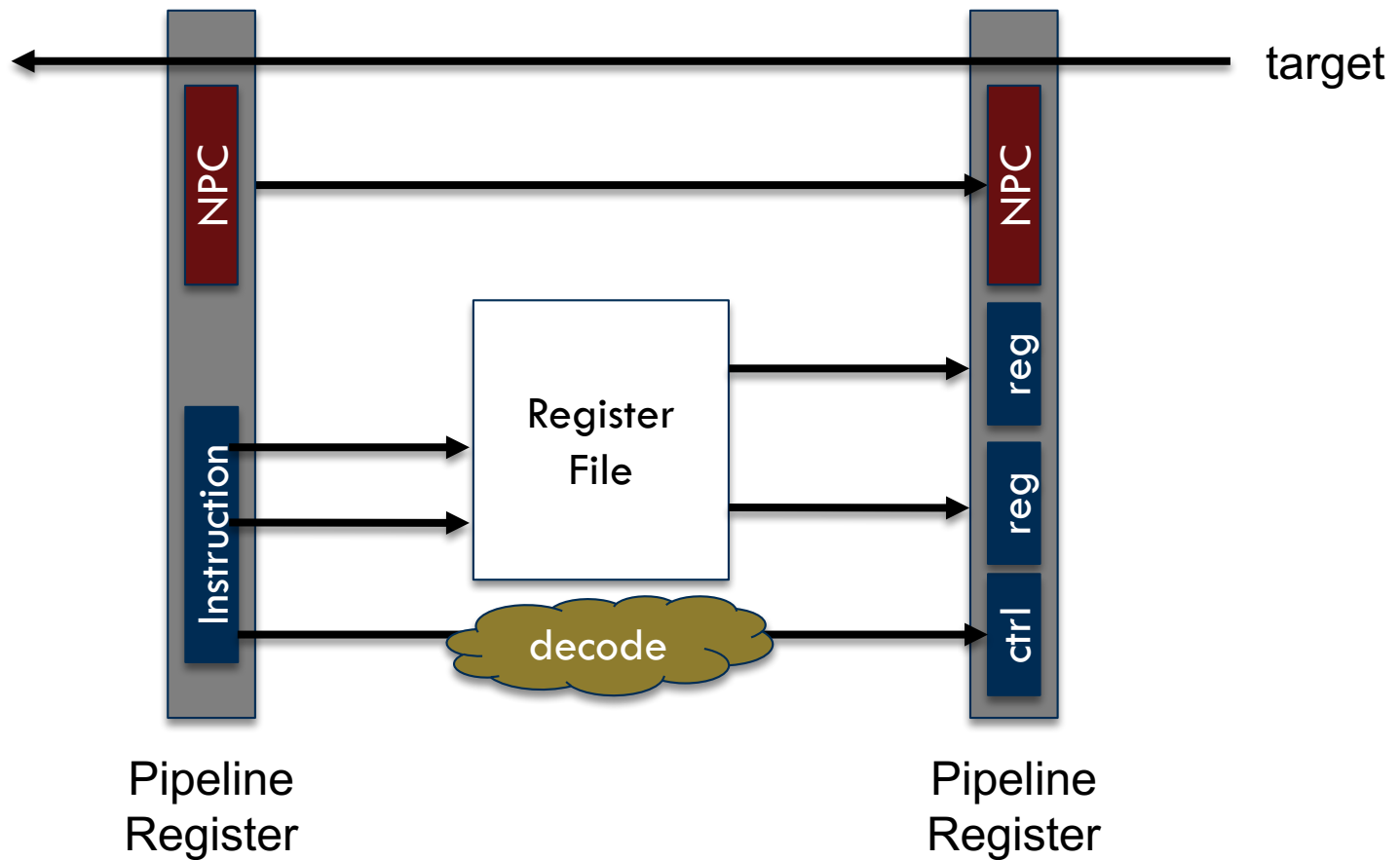
Instruction Fetch



Instruction Decode

- Generate control signals for the opcode bits
- Read source operands from the register file (RF)
 - ▣ Use the specifiers for indexing RF
 - How many read ports are required?
- Update pipeline registers
 - ▣ Send the operand and immediate values to next stage
 - ▣ Pass control signals and NPC to next stage

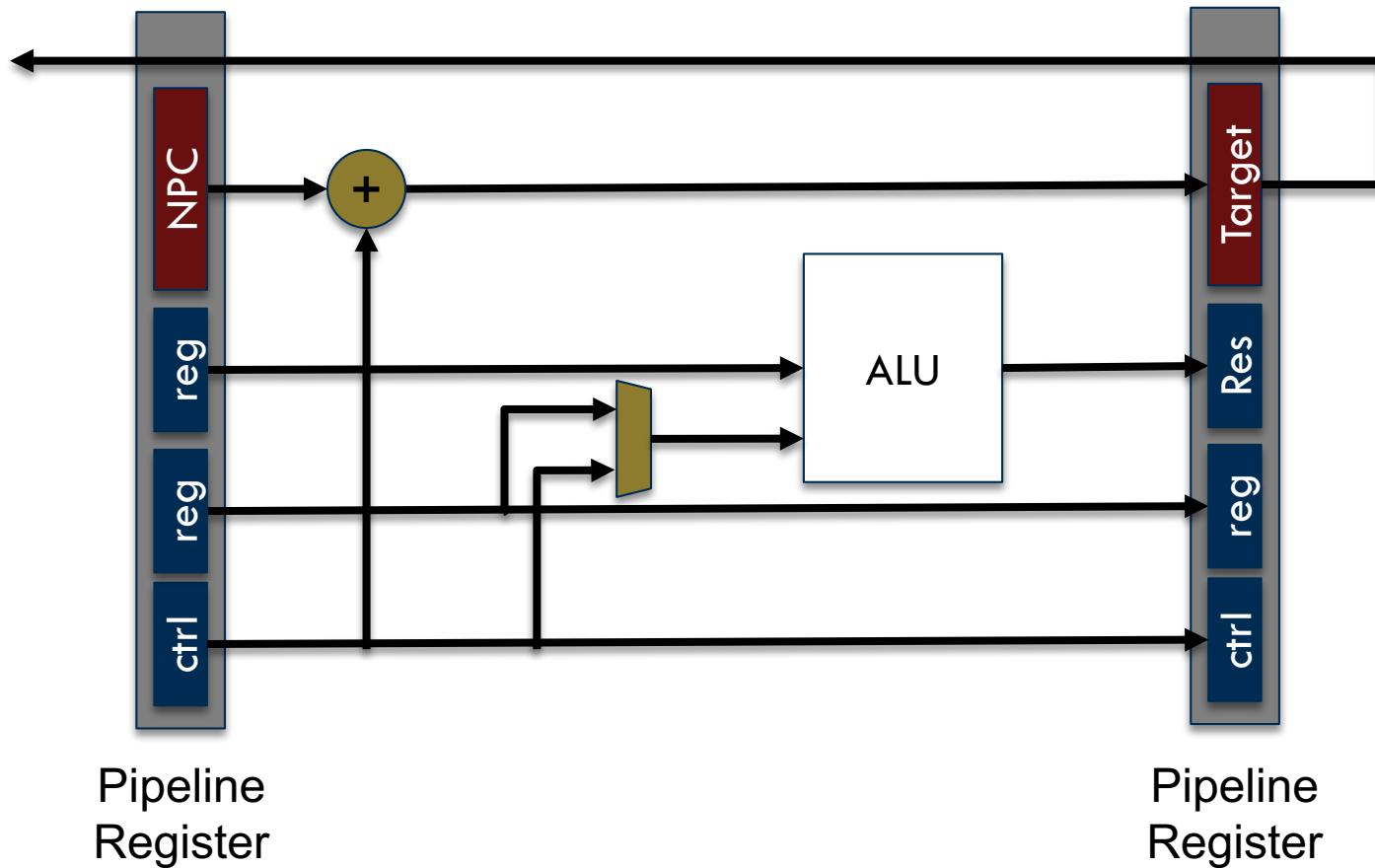
Instruction Decode



Execute Stage

- Perform ALU operation
 - ▣ Compute the result of ALU
 - Operation type: control signals
 - First operand: contents of a register
 - Second operand: either a register or the immediate value
 - ▣ Compute branch target
 - $\text{Target} = \text{NPC} + \text{immediate}$
- Update pipeline registers
 - ▣ Control signals, branch target, ALU results, and destination

Execute Stage

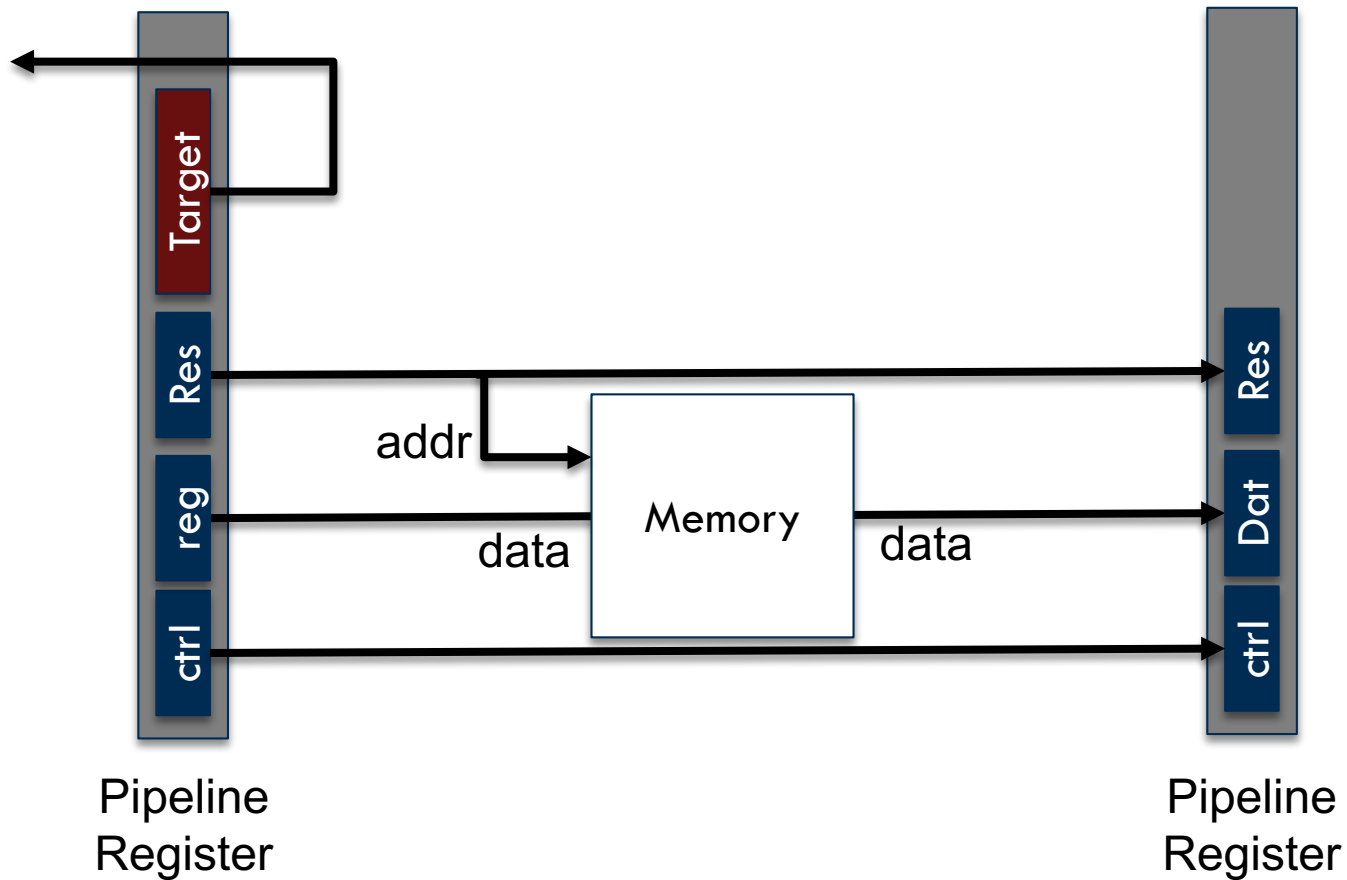


Memory Access

- Access data memory
 - ▣ Load/store address: ALU outcome
 - ▣ Control signals determine read or write access

- Update pipeline registers
 - ▣ ALU results from execute
 - ▣ Loaded data from D-Memory
 - ▣ Destination register

Memory Access



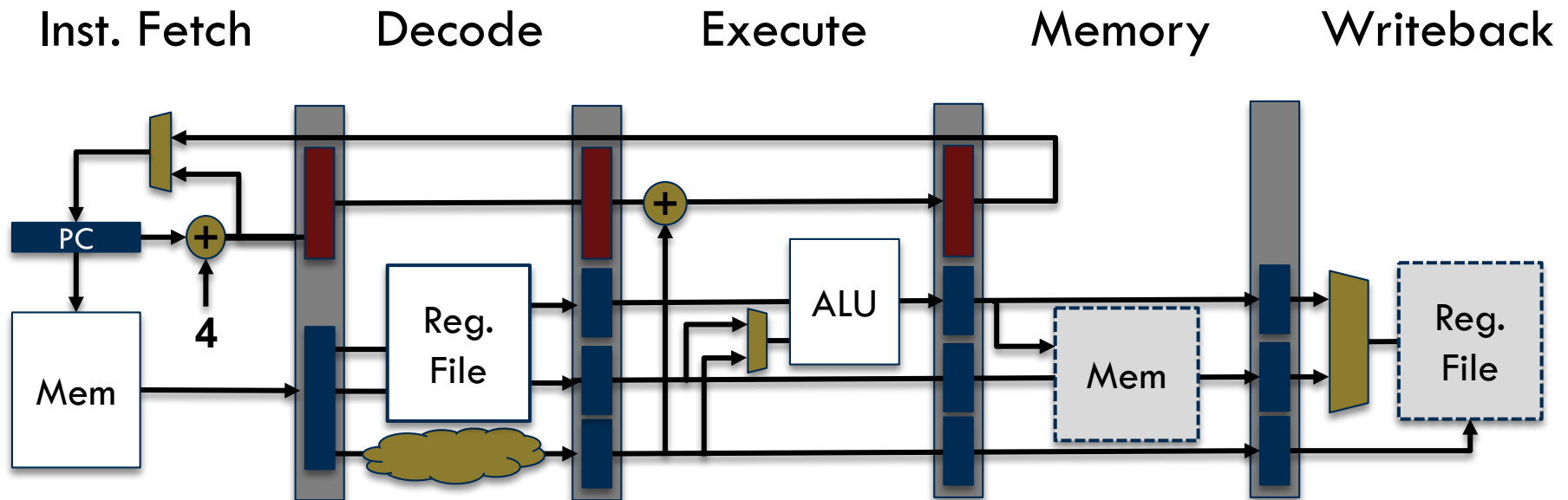
Register Write Back

- Update register file
 - ▣ Control signals determine if a register write is needed
 - ▣ Only one write port is required
 - Write the ALU result to the destination register, or
 - Write the loaded data into the register file

Five Stage Pipeline

□ Ideal pipeline: $IPC=1$

▣ Is there enough resources to keep the pipeline stages busy all the time?



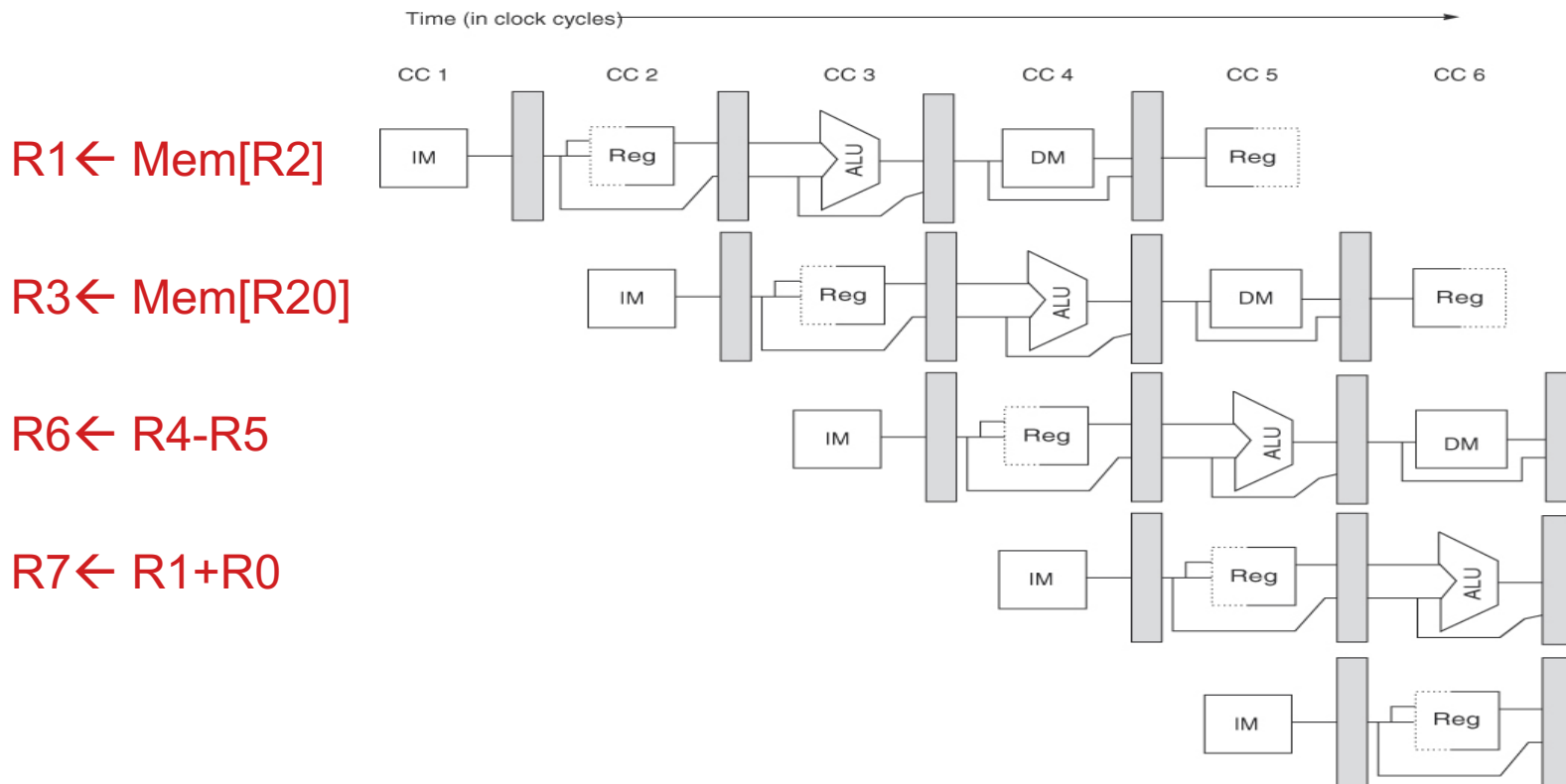
Pipeline Hazards

Pipeline Hazards

- **Structural hazards:** multiple instructions compete for the same resource
- **Data hazards:** a dependent instruction cannot proceed because it needs a value that hasn't been produced
- **Control hazards:** the next instruction cannot be fetched because the outcome of an earlier branch is unknown

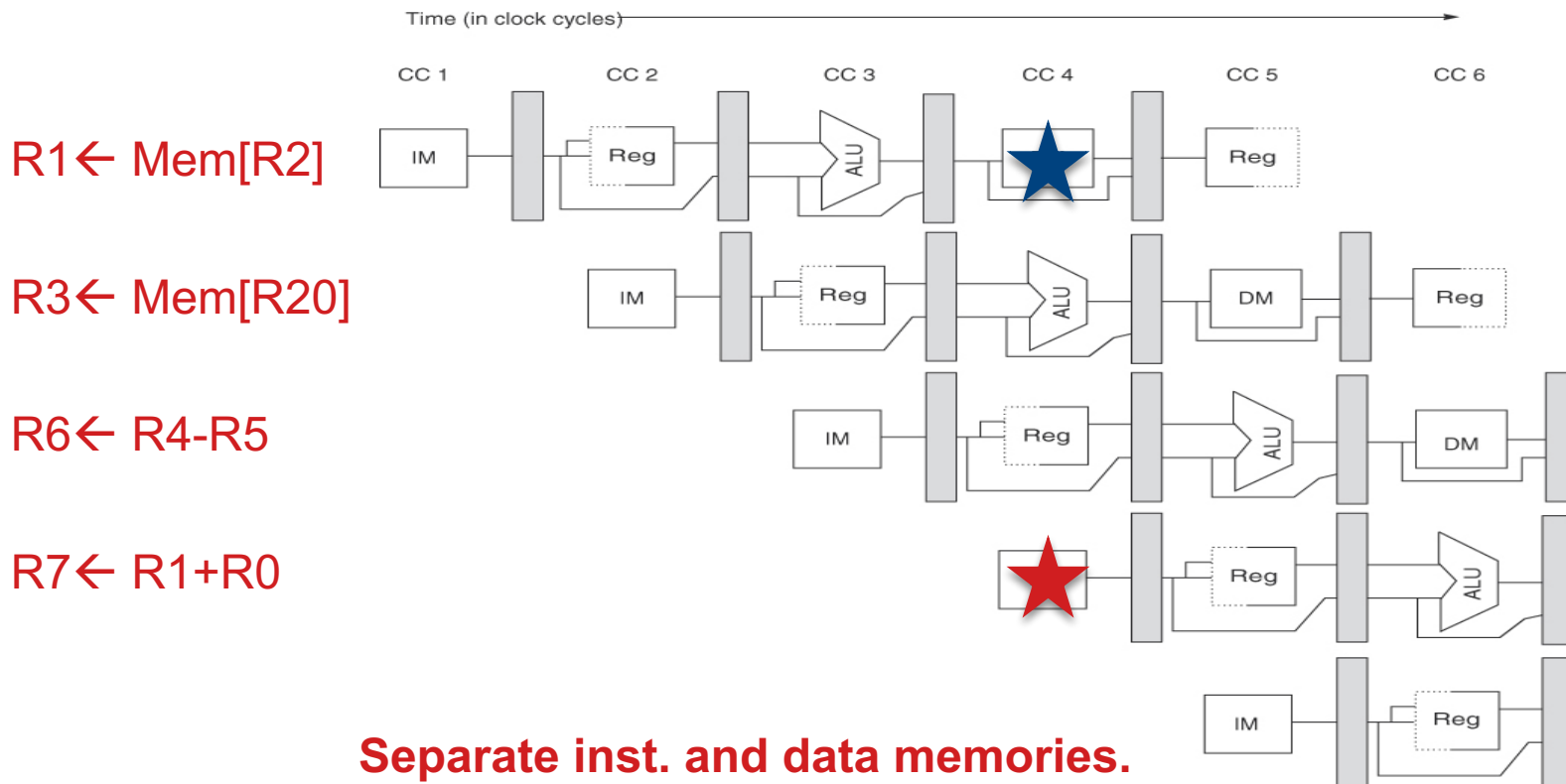
Structural Hazards

- 1. Unified memory for instruction and data



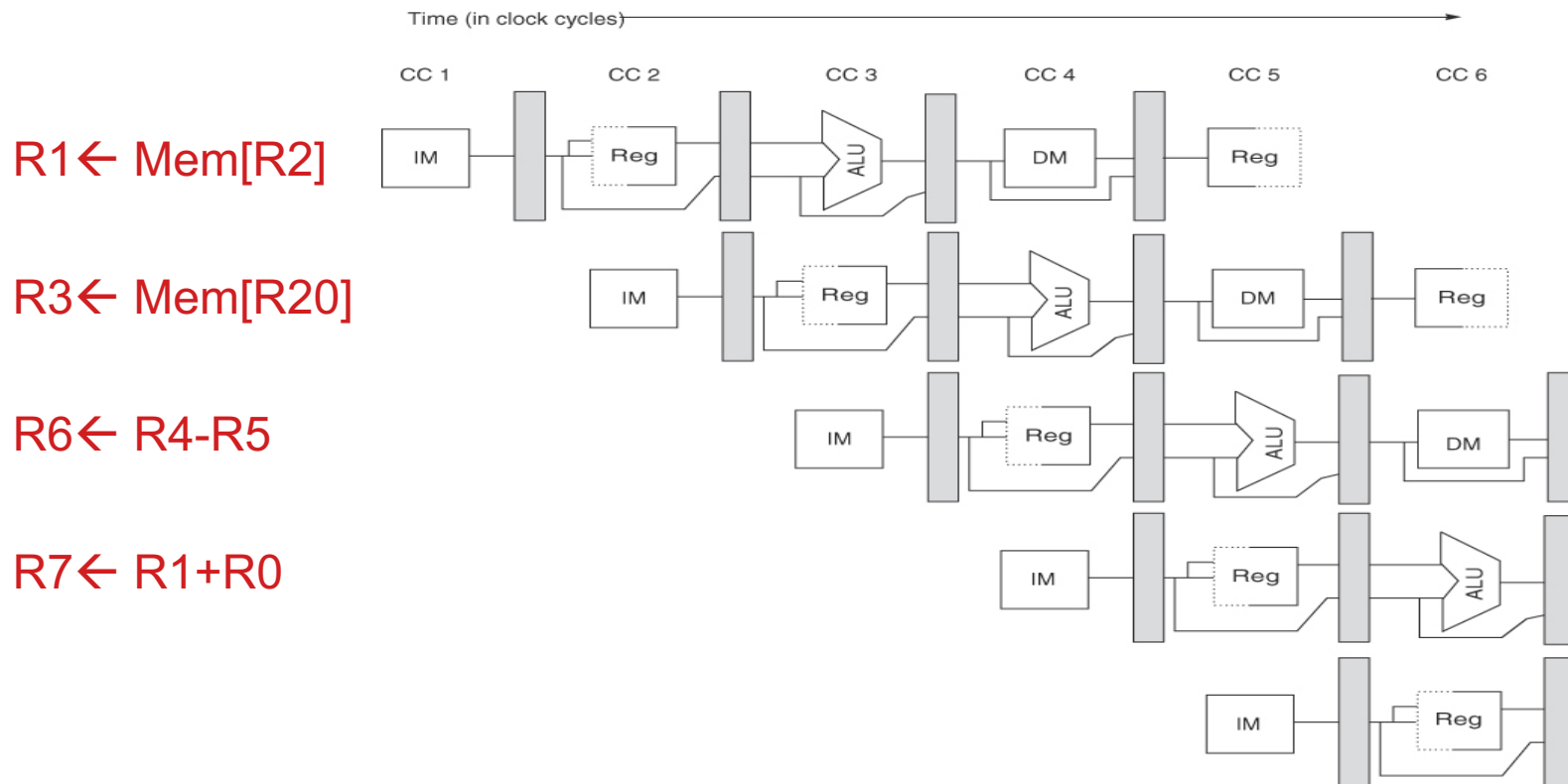
Structural Hazards

- 1. Unified memory for instruction and data



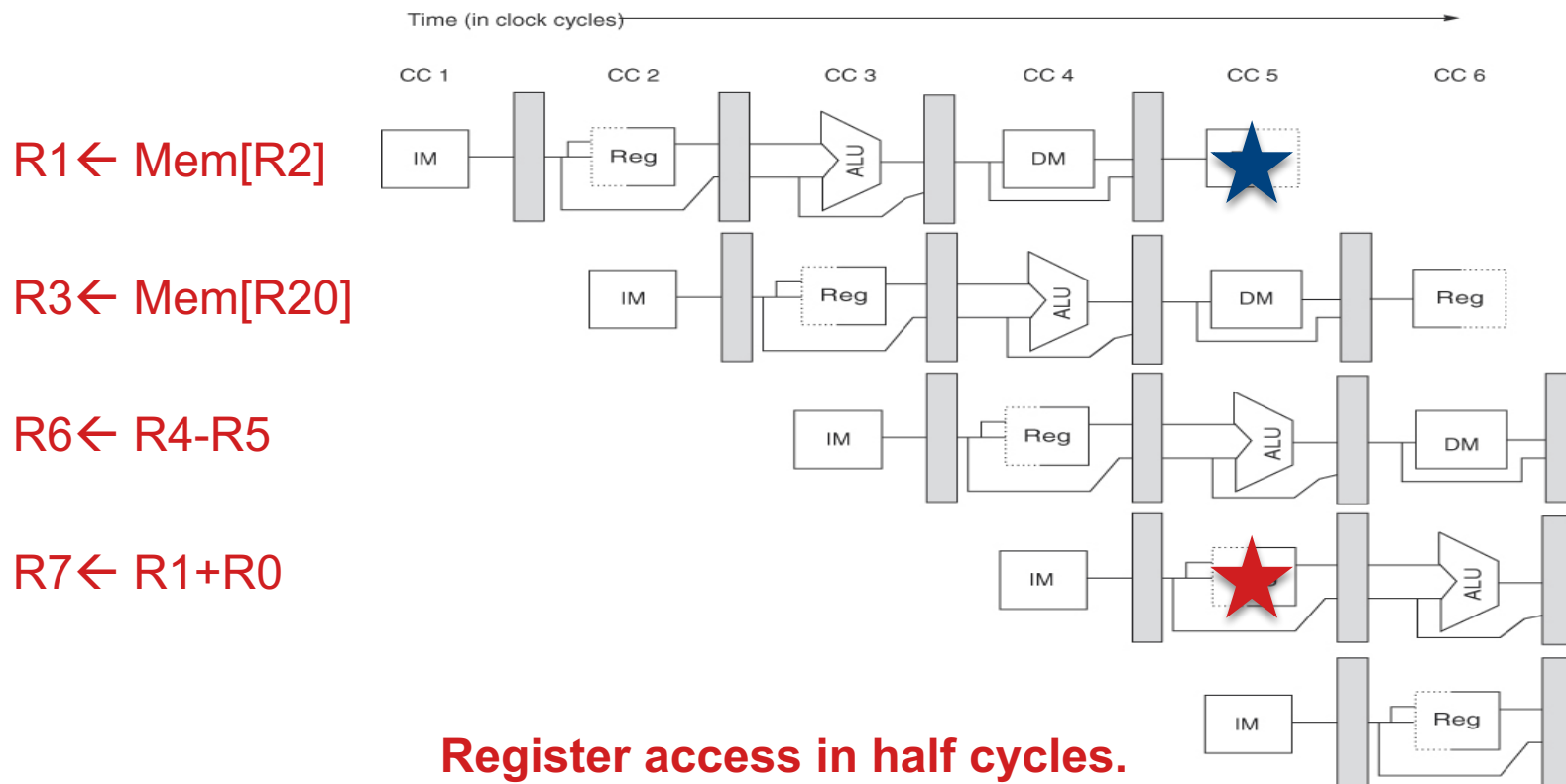
Structural Hazards

- 1. Unified memory for instruction and data
- 2. Register file with shared read/write access ports



Structural Hazards

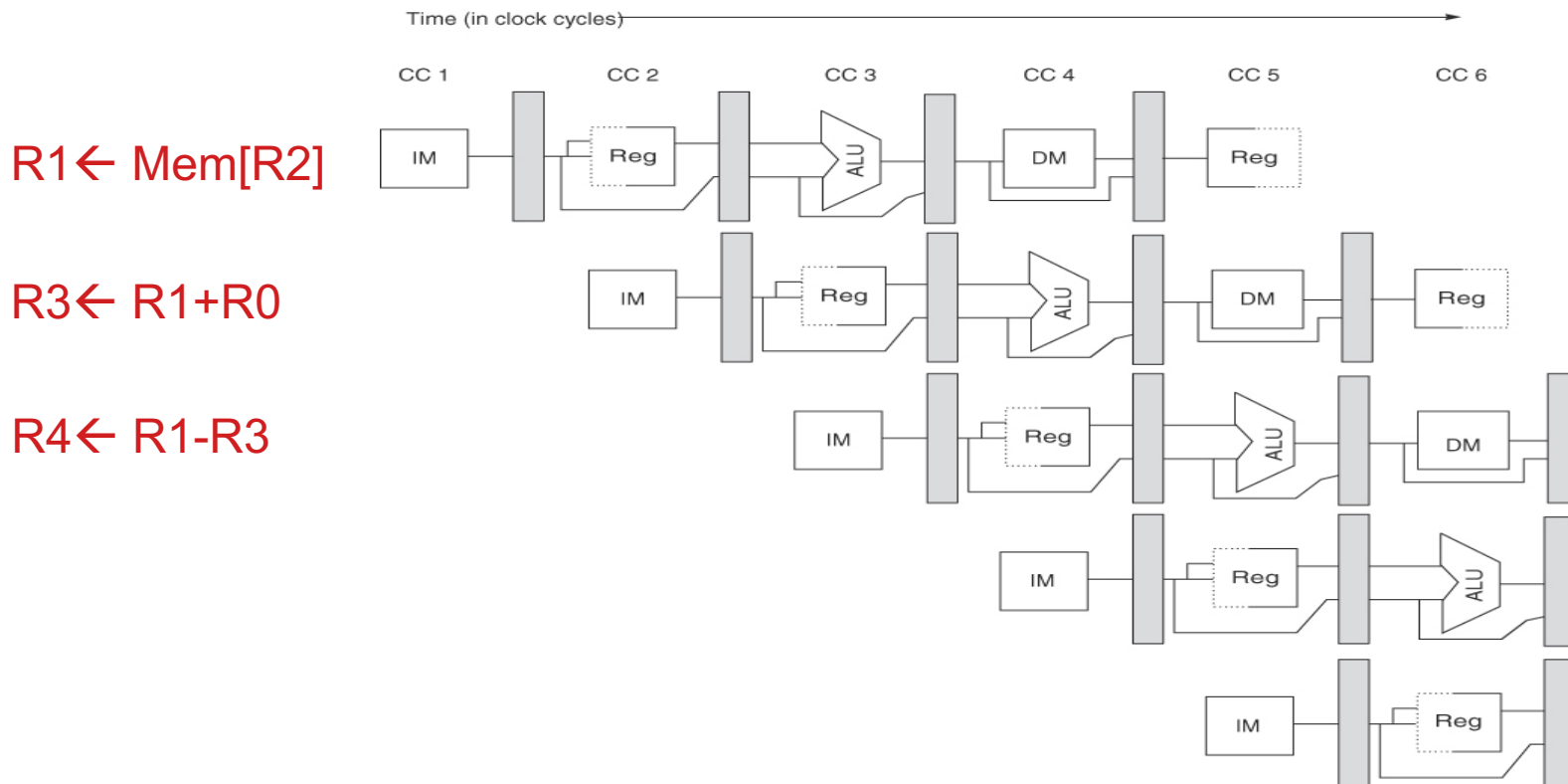
- 1. Unified memory for instruction and data
- 2. Register file with shared read/write access ports



Data Hazards

- ❑ True dependence: read-after-write (RAW)
 - ▣ Consumer has to wait for producer

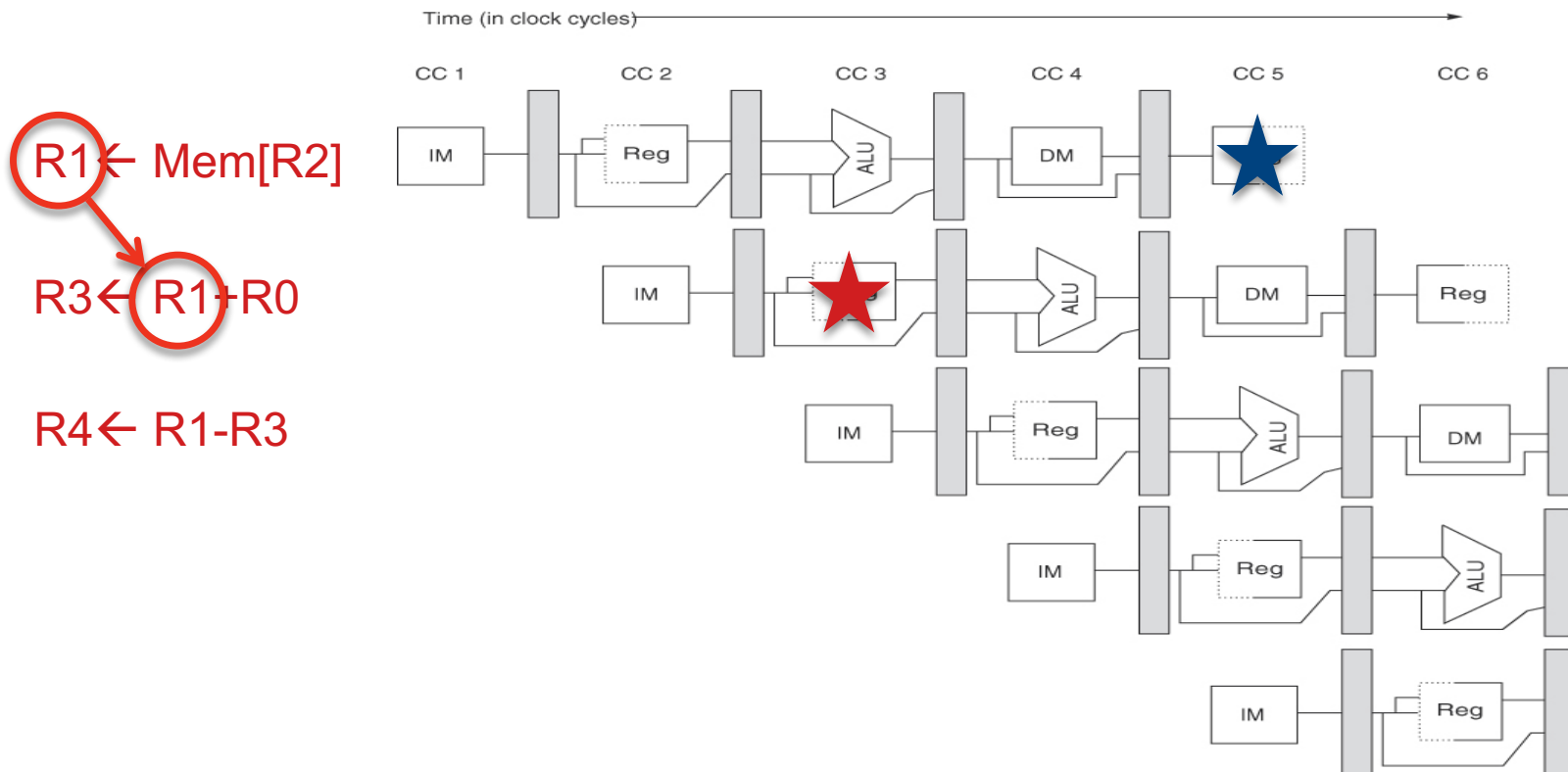
Loading data from memory.



Data Hazards

- True dependence: read-after-write (RAW)
 - ▣ Consumer has to wait for producer

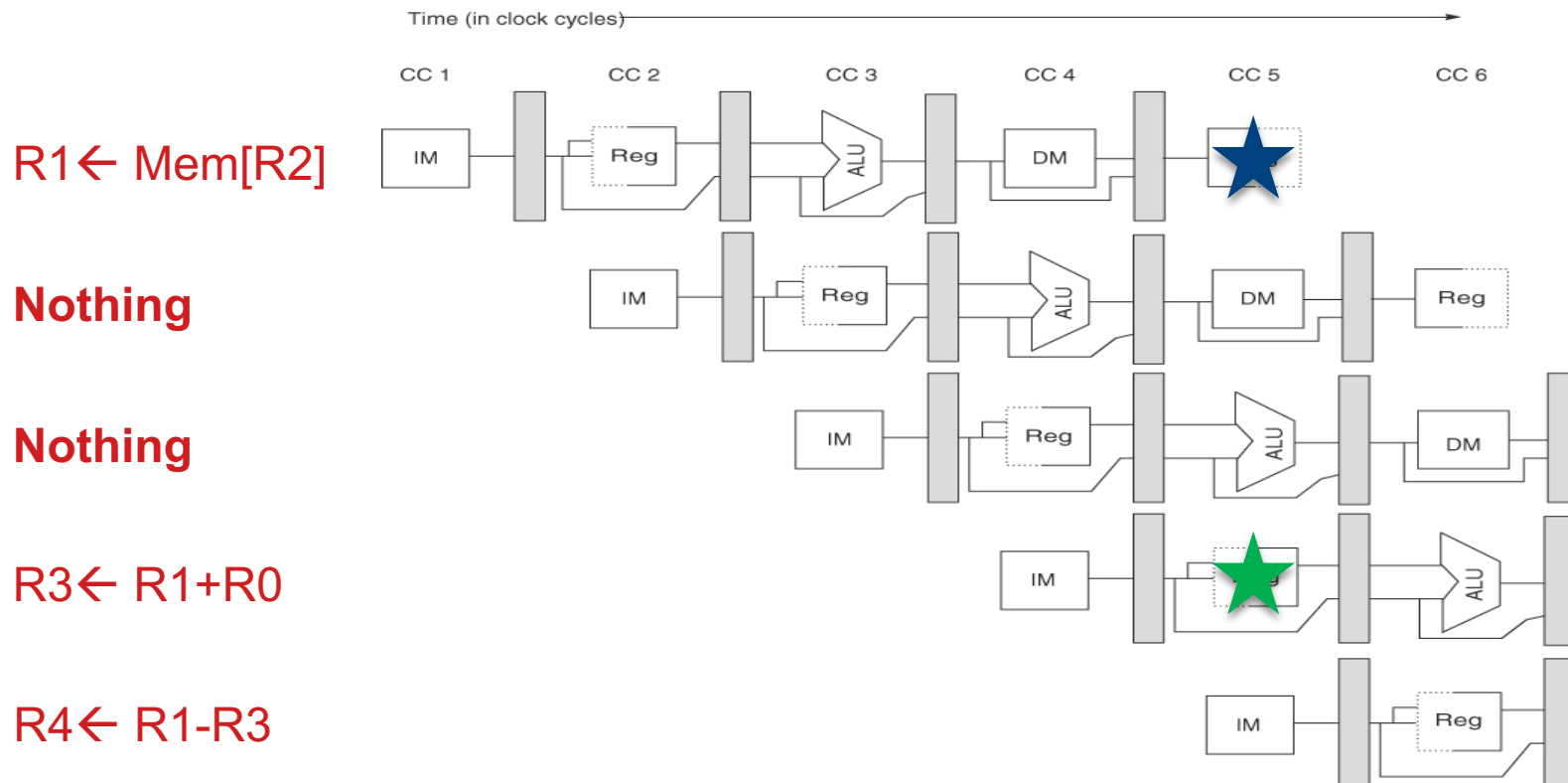
Loaded data will be available two cycles later.



Data Hazards

- True dependence: read-after-write (RAW)
- ▣ Consumer has to wait for producer

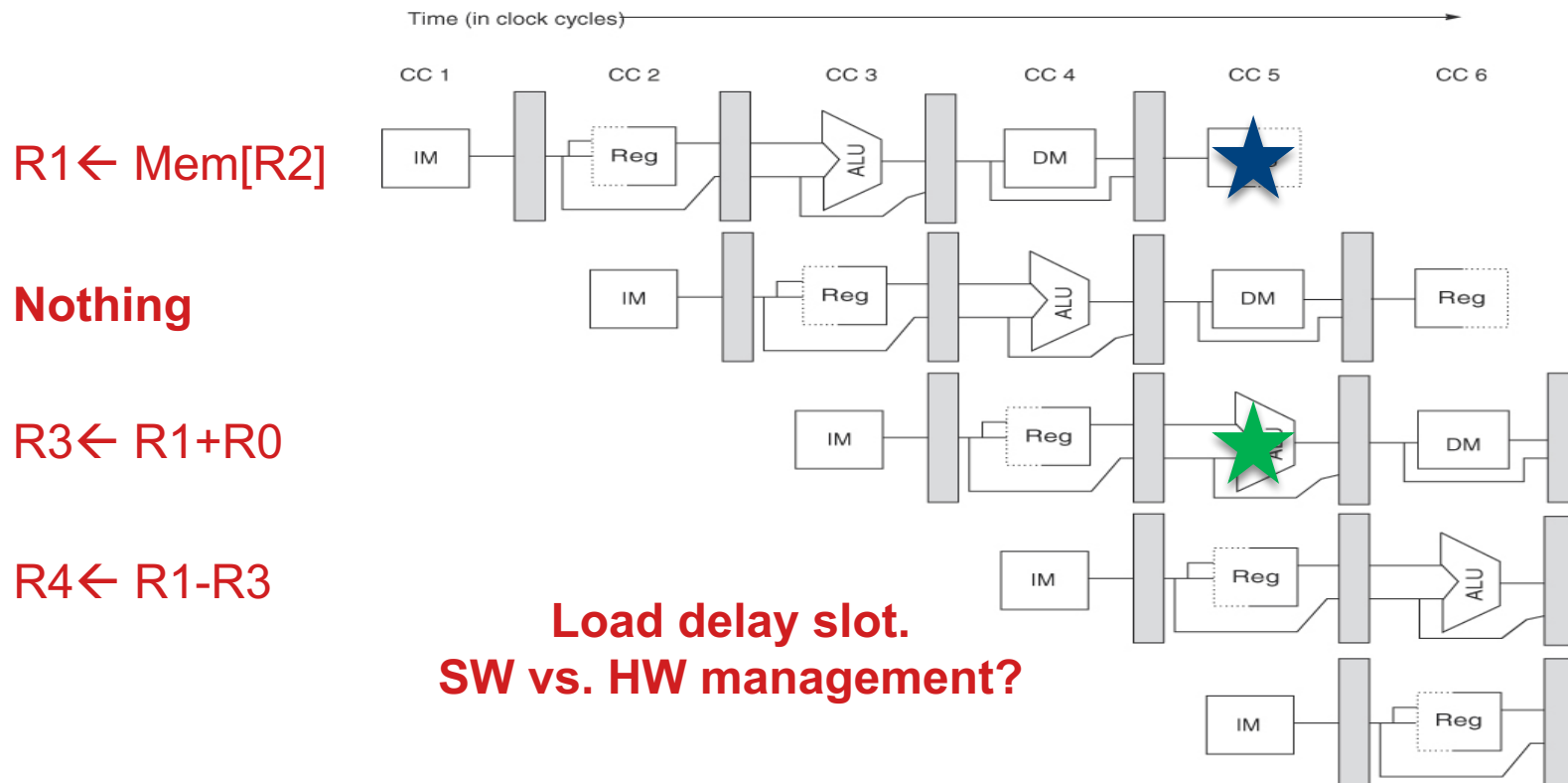
Inserting two bubbles.



Data Hazards

- True dependence: read-after-write (RAW)
 - ▣ Consumer has to wait for producer

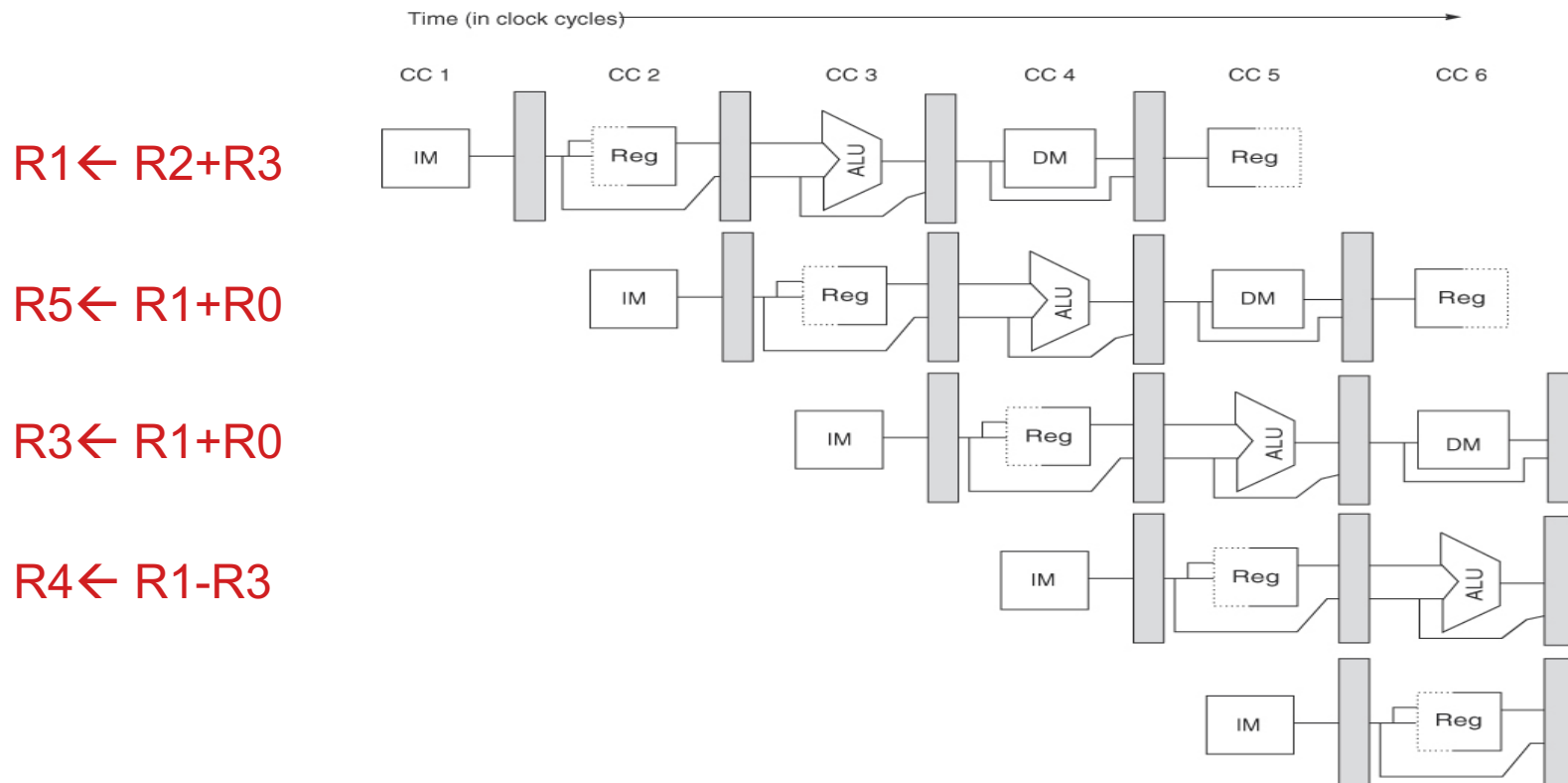
Inserting single bubble + RF bypassing.



Data Hazards

- True dependence: read-after-write (RAW)
 - ▣ Consumer has to wait for producer

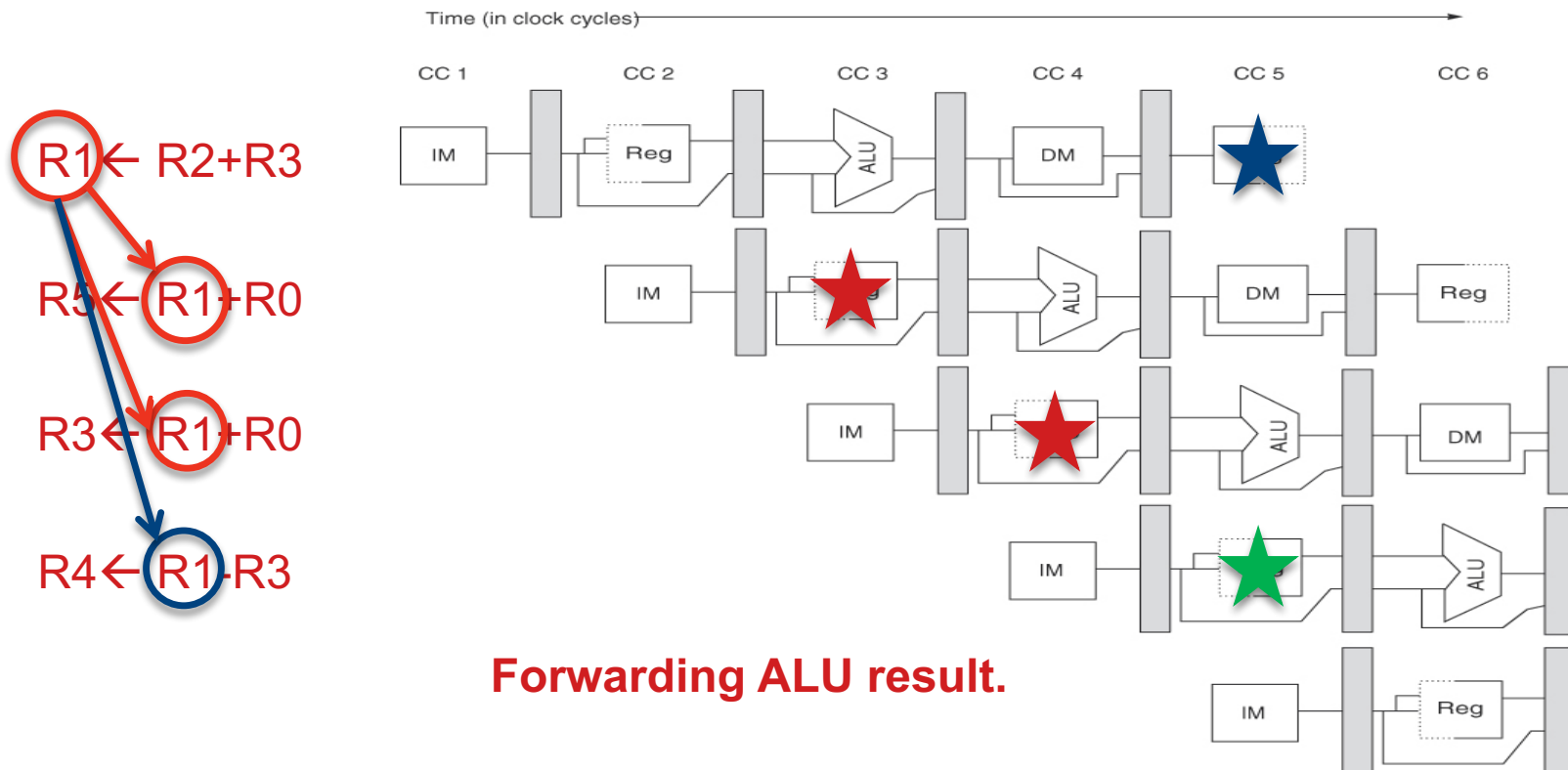
Using the result of an ALU instruction.



Data Hazards

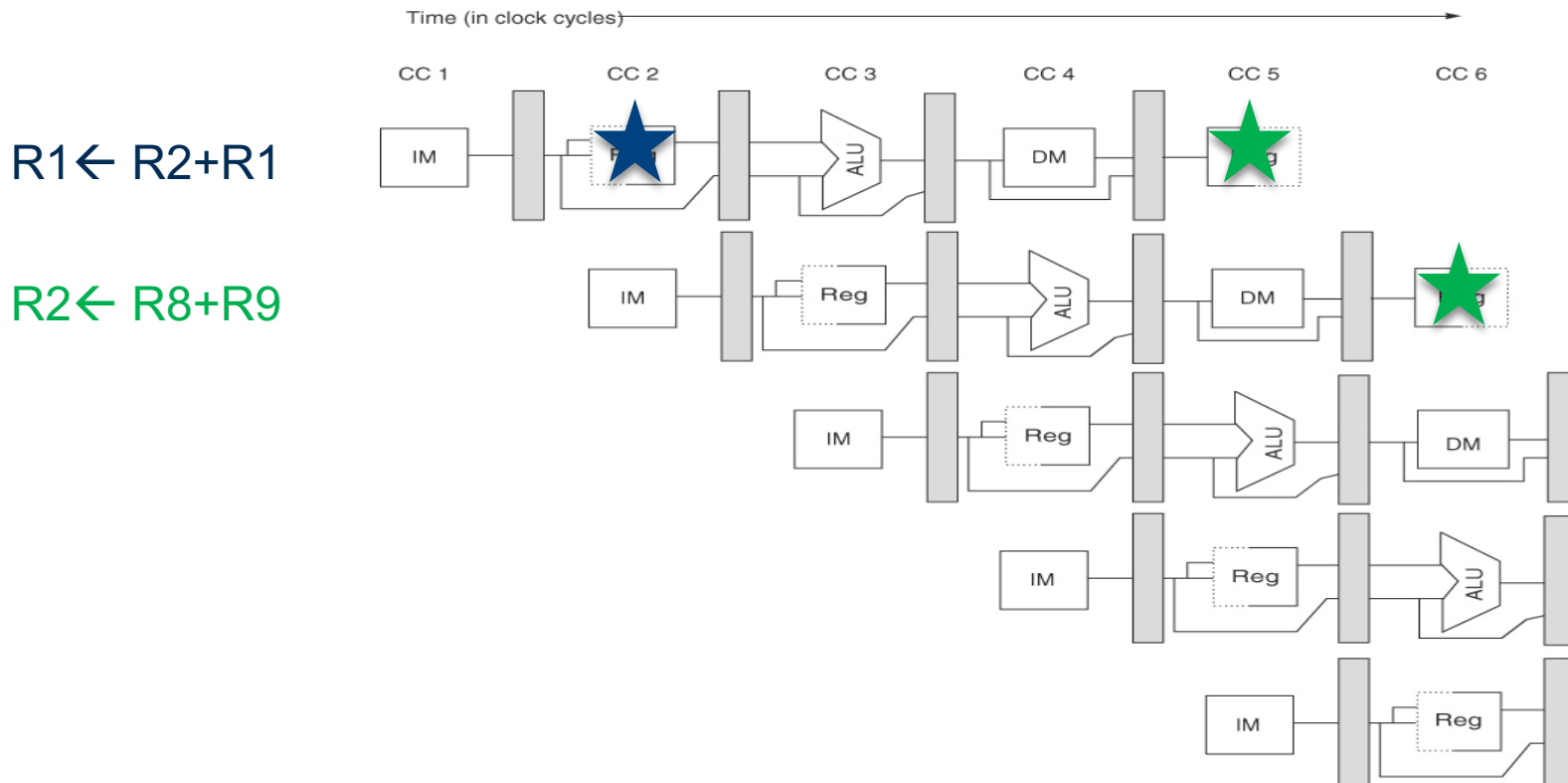
- True dependence: read-after-write (RAW)
 - ▣ Consumer has to wait for producer

Using the result of an ALU instruction.



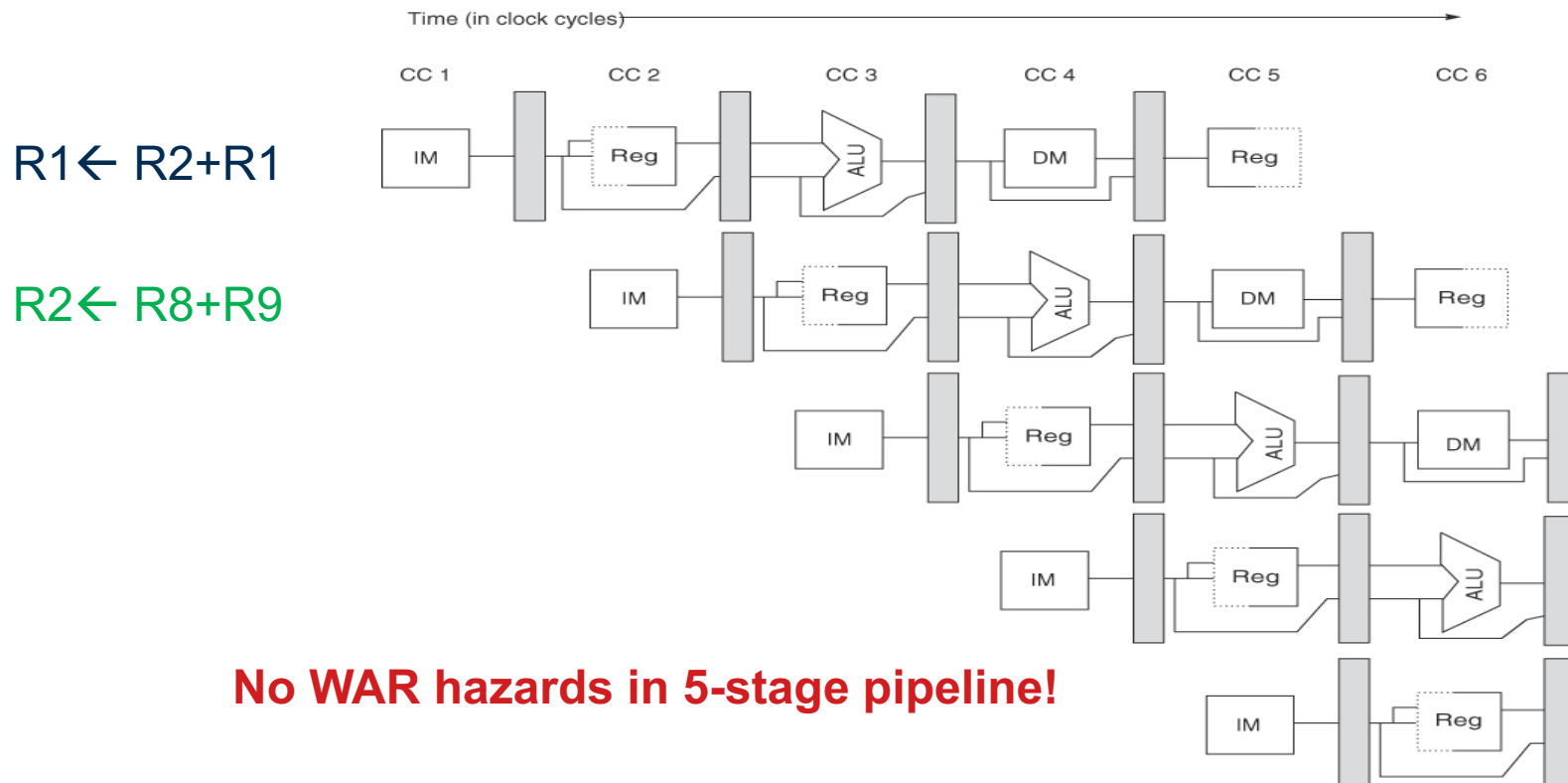
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- ▣ Write must wait for earlier read



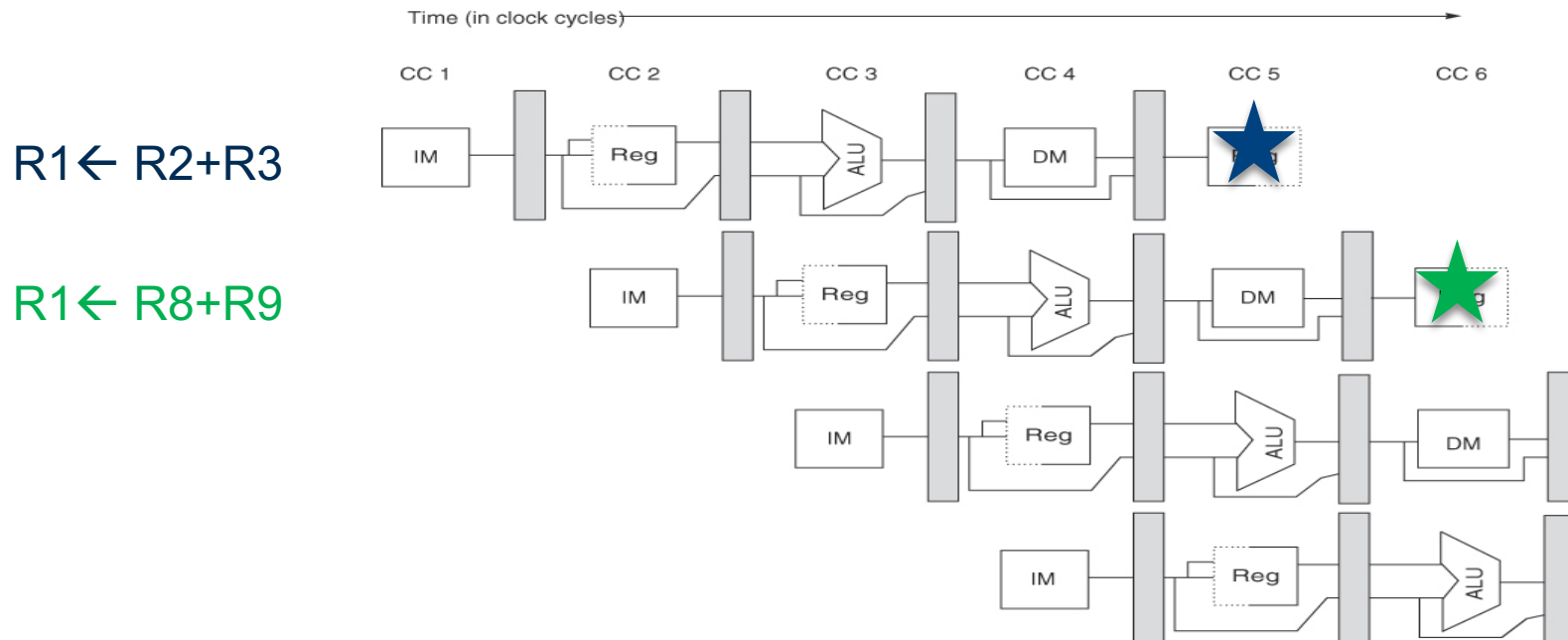
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- ▣ Write must wait for earlier read



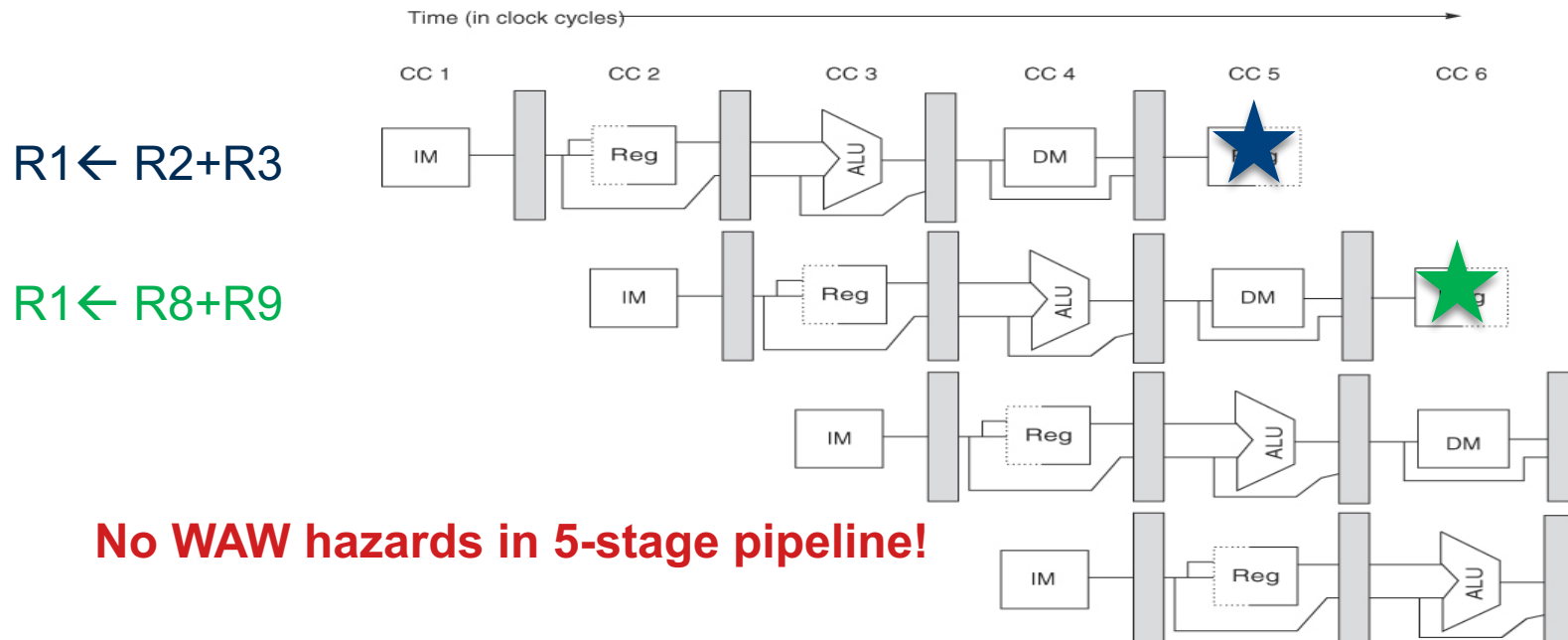
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)
 - ▣ Old writes must not overwrite the younger write



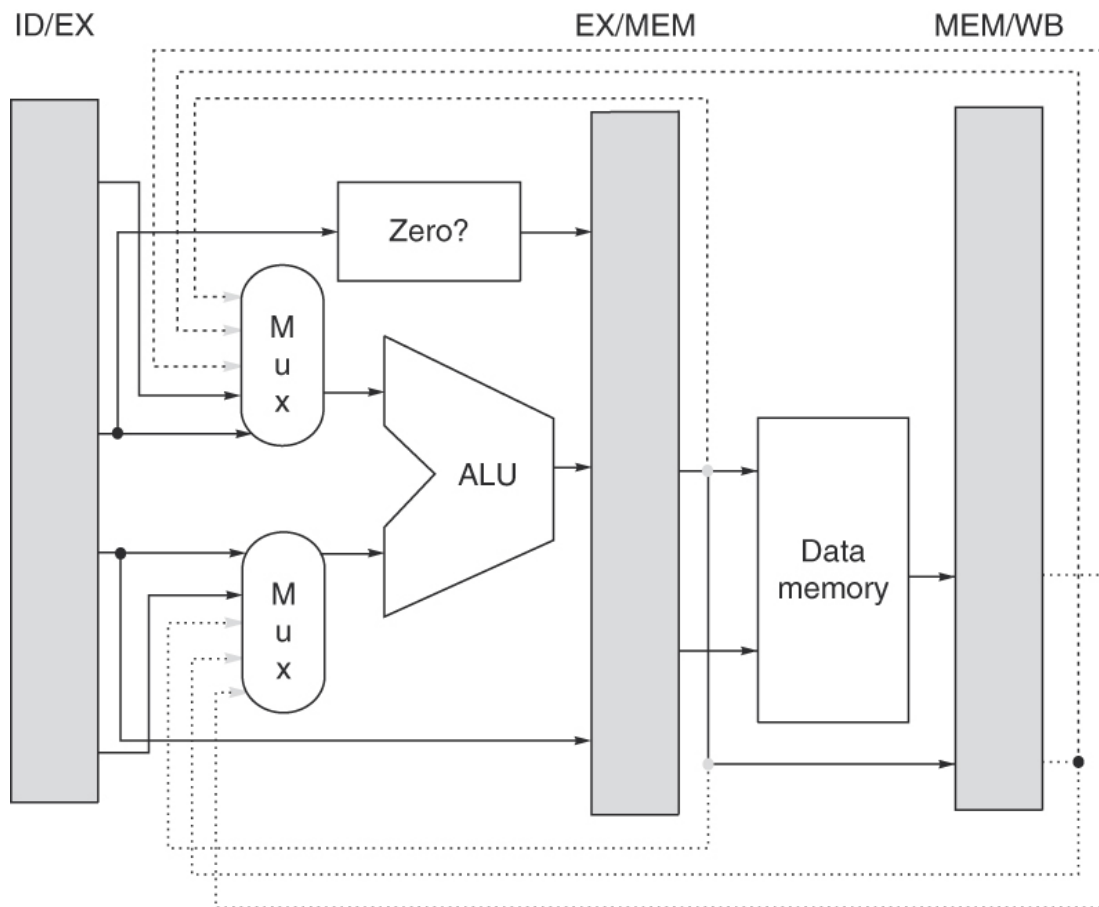
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)
 - ▣ Old writes must not overwrite the younger write



Data Hazards

- Forwarding with additional hardware



Data Hazards

- How to detect and resolve data hazards
 - ▣ Show all of the data hazards in the code below

$R1 \leftarrow \text{Mem}[R2]$

$R2 \leftarrow R1 + R0$

$R1 \leftarrow R1 - R2$

$\text{Mem}[R3] \leftarrow R2$

Data Hazards

- How to detect and resolve data hazards
 - ▣ Show all of the data hazards in the code below

