

CS 222 Computer Organization & Architecture

Lecture 23 [18.03.2019]

Pipeline Hazards



John Jose

Assistant Professor

**Department of Computer Science & Engineering
Indian Institute of Technology Guwahati, Assam.**

Pipelined RISC Data path

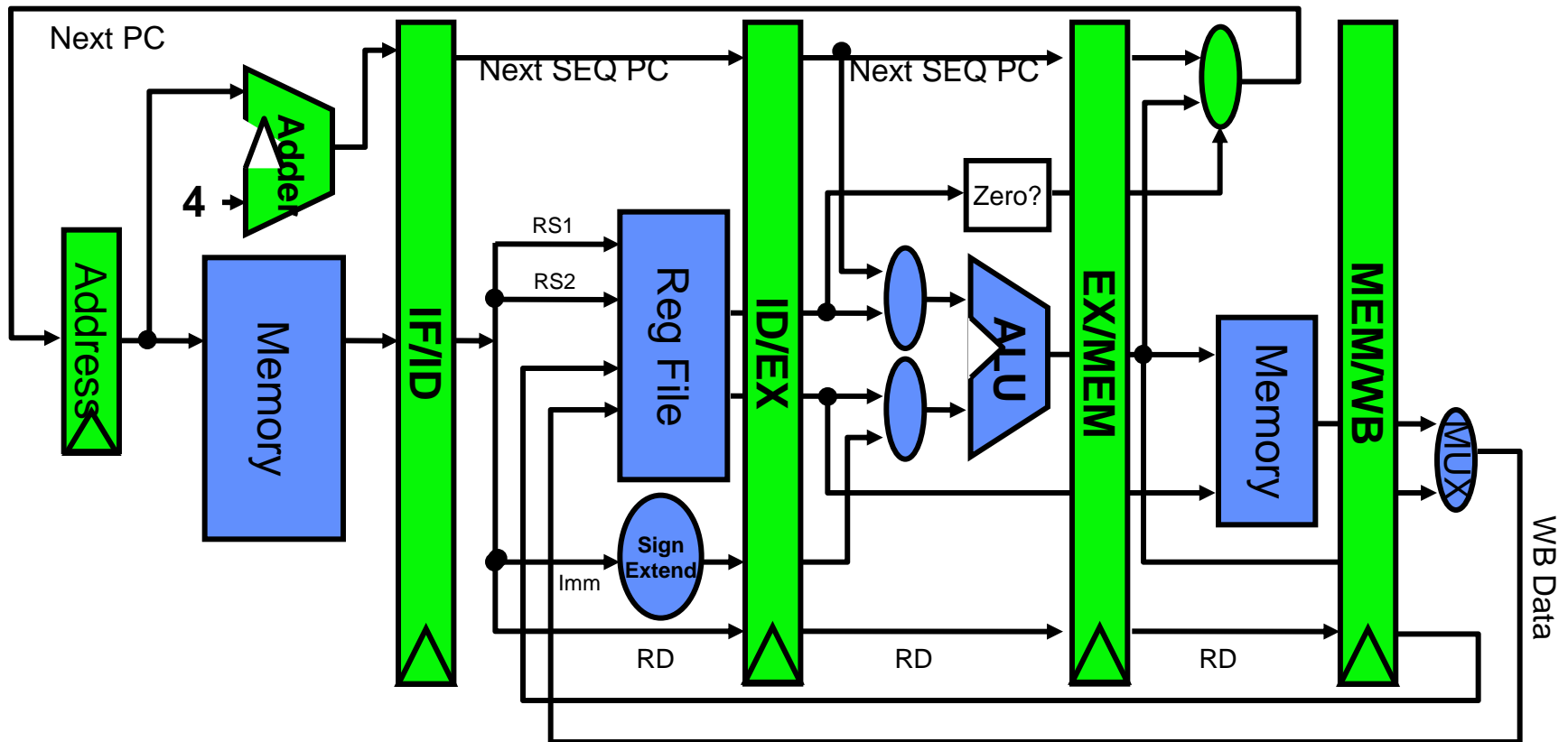
Instruction
Fetch

Instr. Decode
Reg. Fetch

Execute
Addr.
Calc

Memory
Access

Write
Back



Visualizing Pipelining

Instruction number	Clock number							
	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
$i+1$		IF	ID	EX	MEM	WB		
$i+2$			IF	ID	EX	MEM	WB	
$i+3$				IF	ID	EX	MEM	WB
$i+4$					IF	ID	EX	MEM

5 Steps of MIPS Data path

- ❖ Each instruction can take at most 5 clock cycles
- ❖ **Instruction fetch cycle (IF)**
 - ❖ Based on PC, fetch the instruction from memory
 - ❖ Update PC
- ❖ **Instruction decode/register fetch cycle (ID)**
 - ❖ Decode the instruction + register read operation
 - ❖ Fixed field decoding
 - ❖ Equality check of registers
 - ❖ Computation of branch target address if any

5 Steps of MIPS Data path

- ❖ **Execution/Effective address cycle (EX)**
 - ❖ Memory reference: Calculate the eff. address
 - ❖ Register-register ALU instruction
 - ❖ Register-immediate ALU instruction
- ❖ **Memory access cycle (MEM)**
 - ❖ Load instruction: Read from memory using effective address
 - ❖ Store instruction: Write the data in the register to memory using effective address

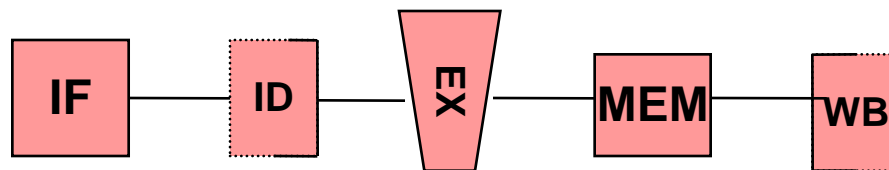
5 Steps of MIPS Data path

❖ Write-back cycle (WB)

- ❖ Register-register ALU instruction or load instruction
- ❖ Write the result into the register file

❖ Cycles to implement different instructions

- ❖ Branch instructions – 4 cycles
- ❖ Store instructions – 4 cycles
- ❖ All other instructions – 5 cycles



Pipelining Issues

❖ **Ideal Case: Uniform sub-computations**

- ❖ The computation to be performed can be evenly partitioned into uniform-latency sub-computations

❖ **Reality: Internal fragmentation**

- ❖ Not all pipeline stages may have the uniform latencies

❖ **Impact of ISA**

- ❖ Memory access is a critical sub-computation
- ❖ Memory addressing modes should be minimized
- ❖ Fast cache memories should be employed

Pipelining Issues

❖ Ideal Case : Identical computations

- ❖ The same computation is to be performed repeatedly on a large number of input data sets

❖ Reality: External fragmentation

- ❖ Some pipeline stages may not be used

❖ Impact of ISA

- ❖ Reduce the complexity and diversity of the different instruction types
- ❖ RISC use uniform stage simple instructions

Pipelining Issues

❖ Ideal Case : Independent computations

- ❖ All the instructions are mutually independent

❖ Reality: Pipeline stalls – cannot proceed.

- ❖ A later computation may require the result of an earlier computation

❖ Impact of ISA

- ❖ Reduce Memory addressing modes - dependency detection difficult

- ❖ Use register addressing mode - easy dependencies check

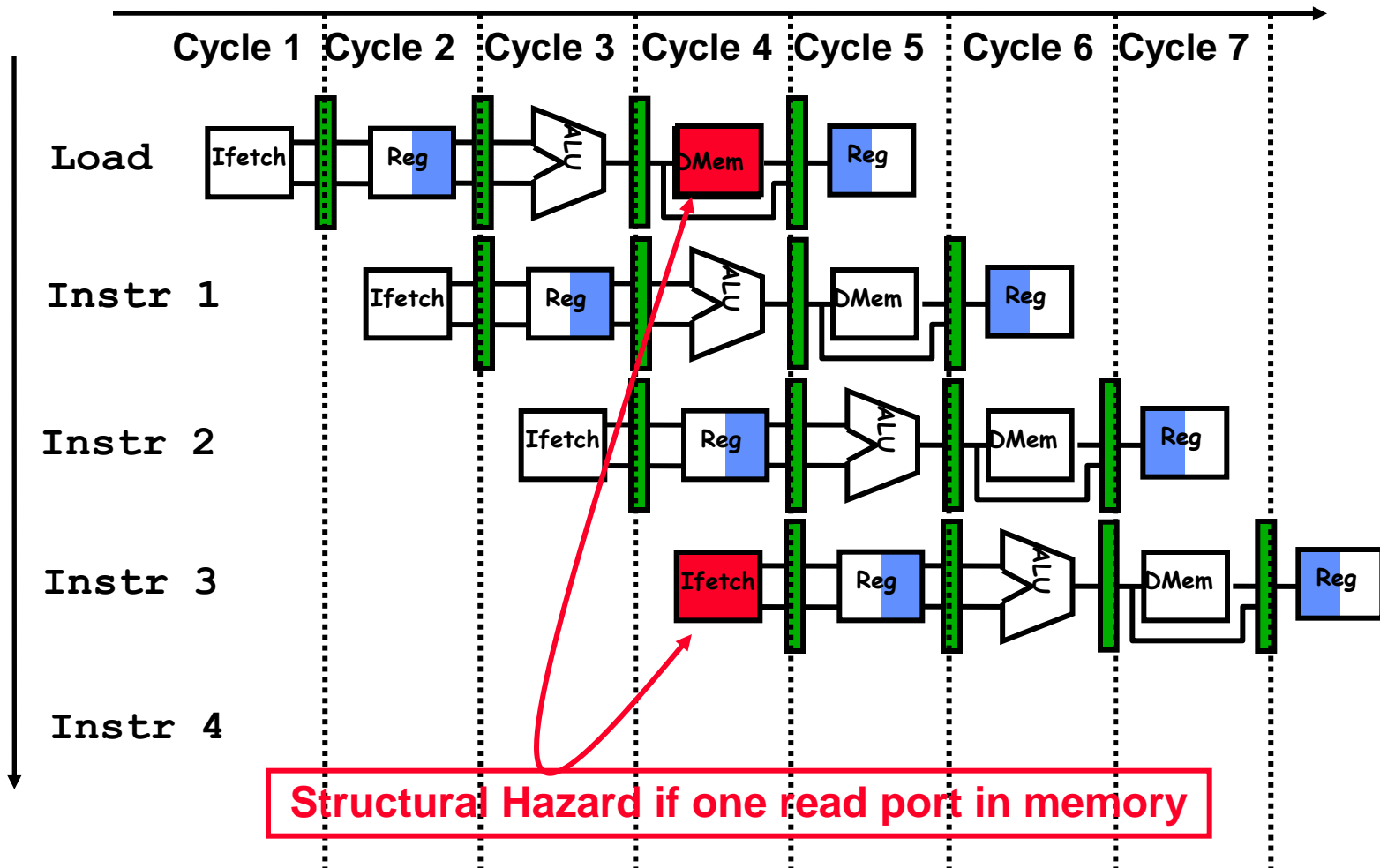
Limits to Pipelining

- ❖ **Hazards:** circumstances that would cause incorrect execution if next instruction is fetched and executed
- ❖ **Structural hazards:** Different instructions, at different stages, in the pipeline want to use the same hardware
- ❖ **Data hazards:** An instruction in the pipeline requires data to be computed by a previous instruction still in the pipeline
- ❖ **Control hazards:** Succeeding instruction, to put into pipeline, depends on the outcome of a previous branch instruction, already in pipeline

Structural Hazard

Eg: Uniport Memory

Time (clock cycles)



Resolving Structural Hazard

- ❖ Eliminate the use same hardware for two different things at the same time

- ❖ **Solution 1: Wait**

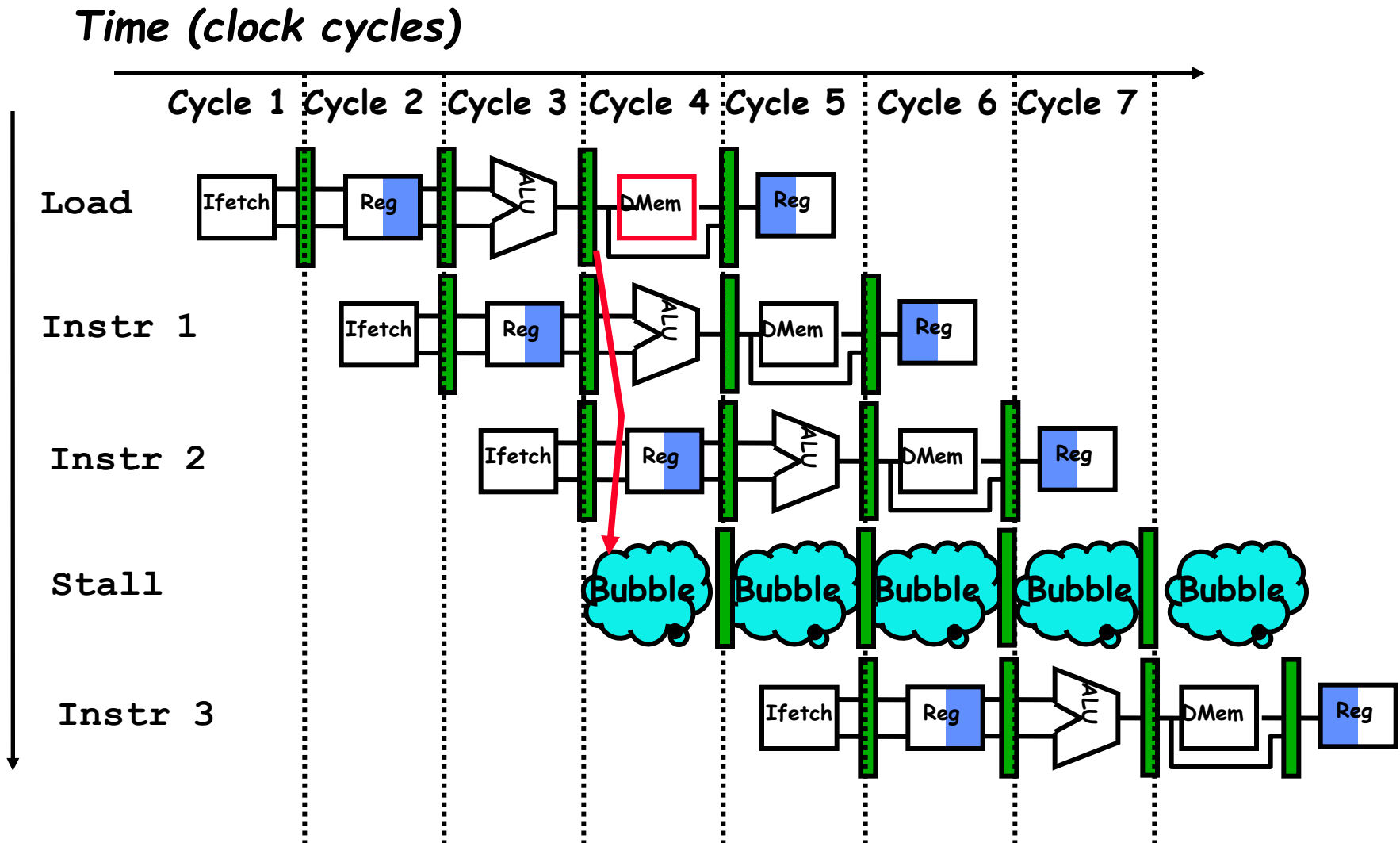
 - ❖ must detect the hazard

 - ❖ must have mechanism to stall

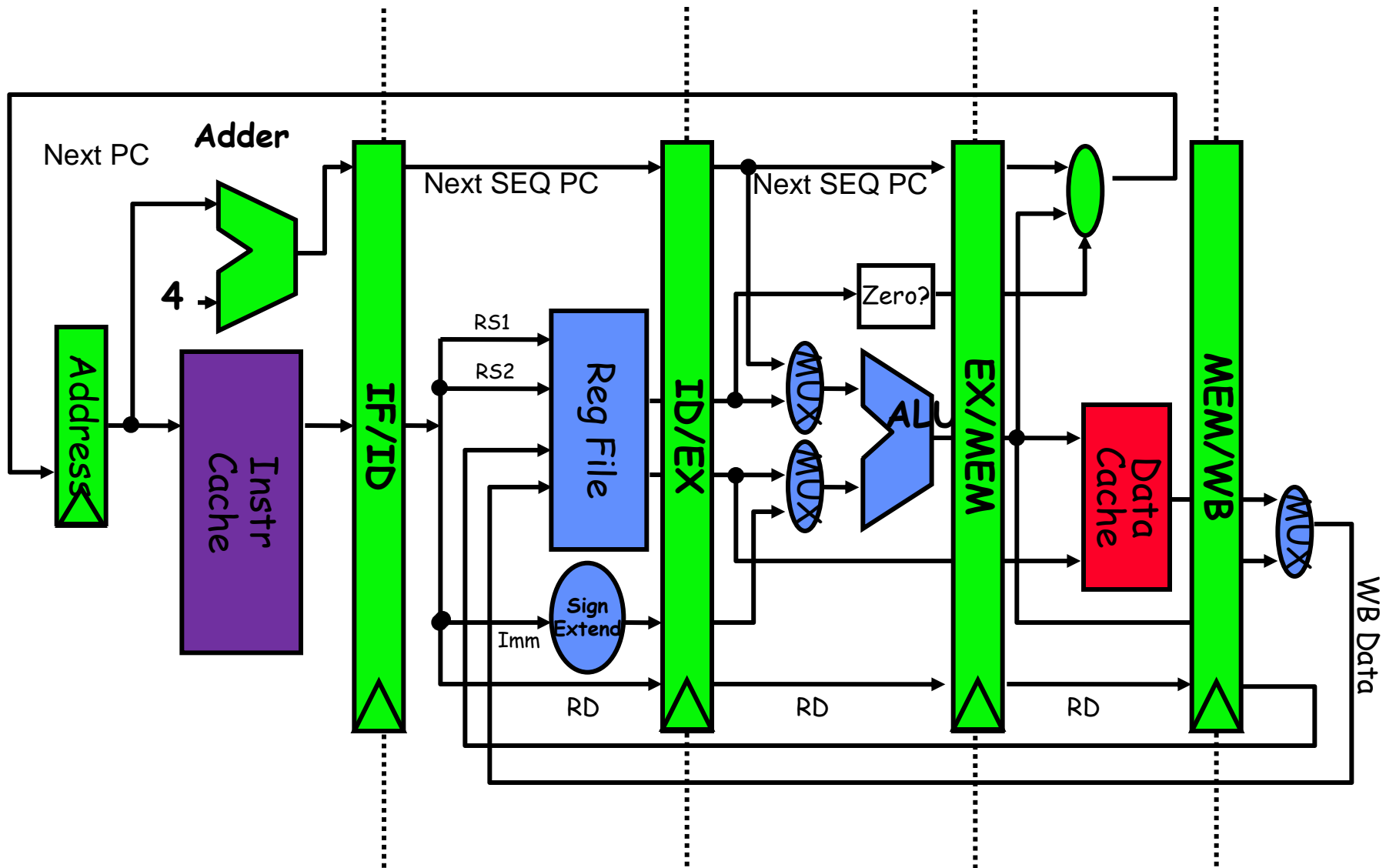
- ❖ **Solution 2: Duplicate hardware**

 - ❖ Multiple such units will help both instruction to progress

Detecting & Resolving Structural Hazard

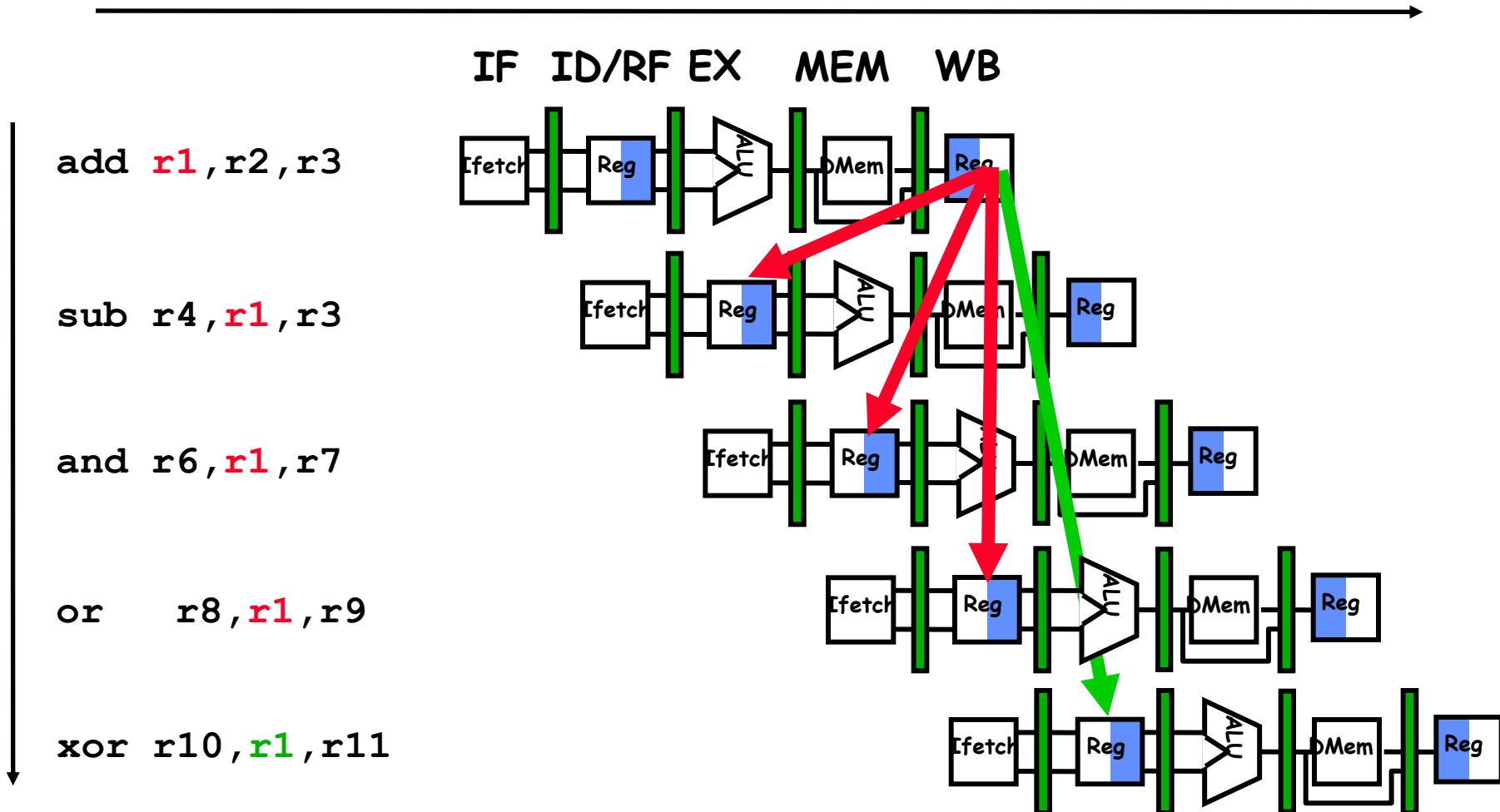


Eliminating Structural Hazards at Design



Data Hazard


Time (clock cycles)



Three Generic Data Hazards

❖ Read After Write (RAW)

Instr_j tries to read operand **before** Instr_i writes it

 I : add **r1**,r2,r3
J: sub r4,**r1**,r3

❖ Caused by a data dependence

❖ This hazard results from an actual need for communication.

Three Generic Data Hazards

❖ Write After Read (WAR)

Instr_j writes operand **before** Instr_i reads it

I: sub r4,**r1**,r3

J: add **r1**,r2,r3

K: mul r6,r1,r7

- ❖ Called an anti-dependence by compiler writers.
- ❖ This results from reuse of the name r1
- ❖ Can't happen in MIPS 5 stage pipeline because:
 - ❖ All instructions take 5 stages, and
 - ❖ Reads are always in stage 2, and
 - ❖ Writes are always in stage 5

Three Generic Data Hazards

❖ Write After Write (WAW)

Instr_j writes operand **before** Instr_i writes

I: sub **r1**,r4,r3

J: add **r1**,r2,r3

K: mul r6,r1,r7

❖ Called an output dependence

❖ This also results from the reuse of name r1.

❖ Can't happen in MIPS 5 stage pipeline because:

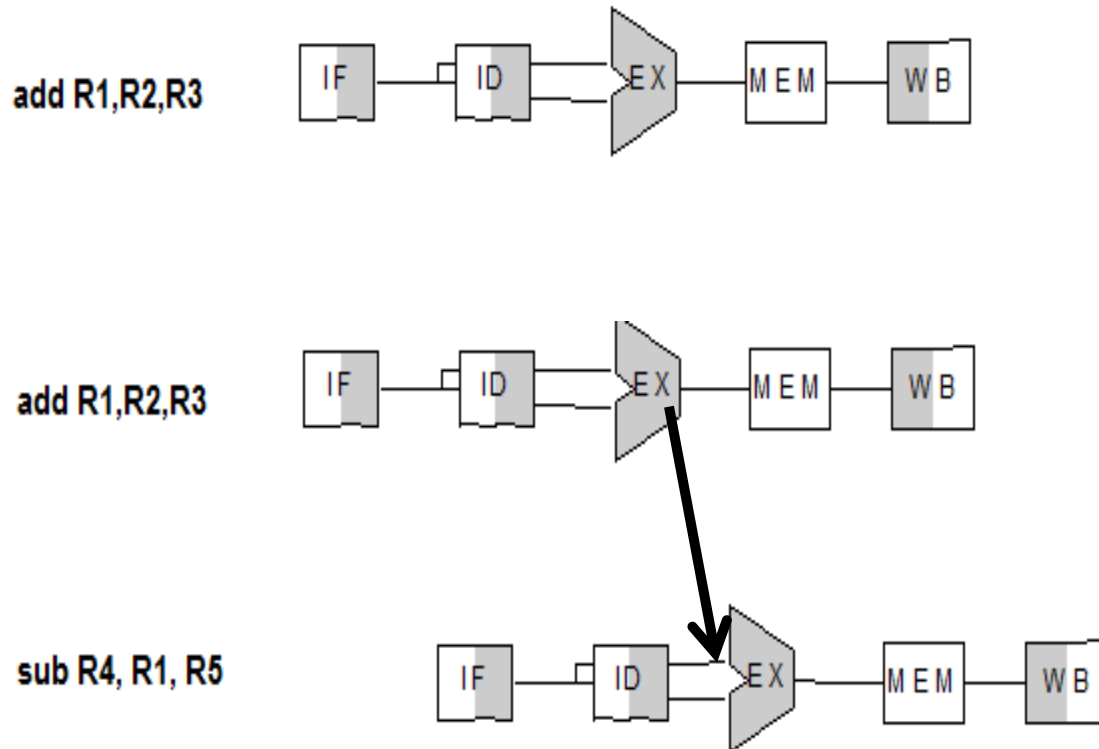
❖ All instructions take 5 stages, and

❖ Writes are always in stage 5

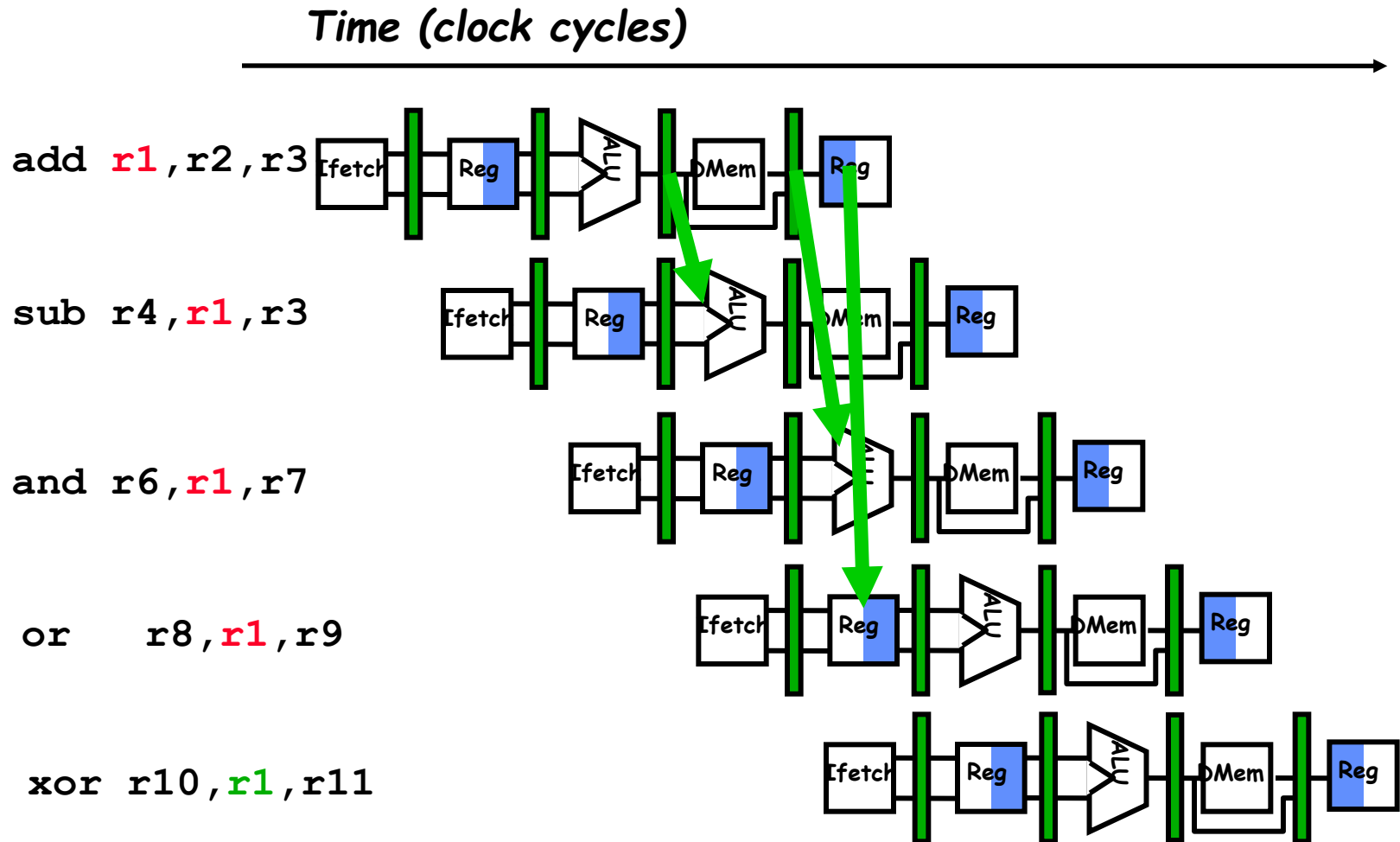
❖ WAR and WAW happens in out of order pipes

How to Handle Data Hazard ?

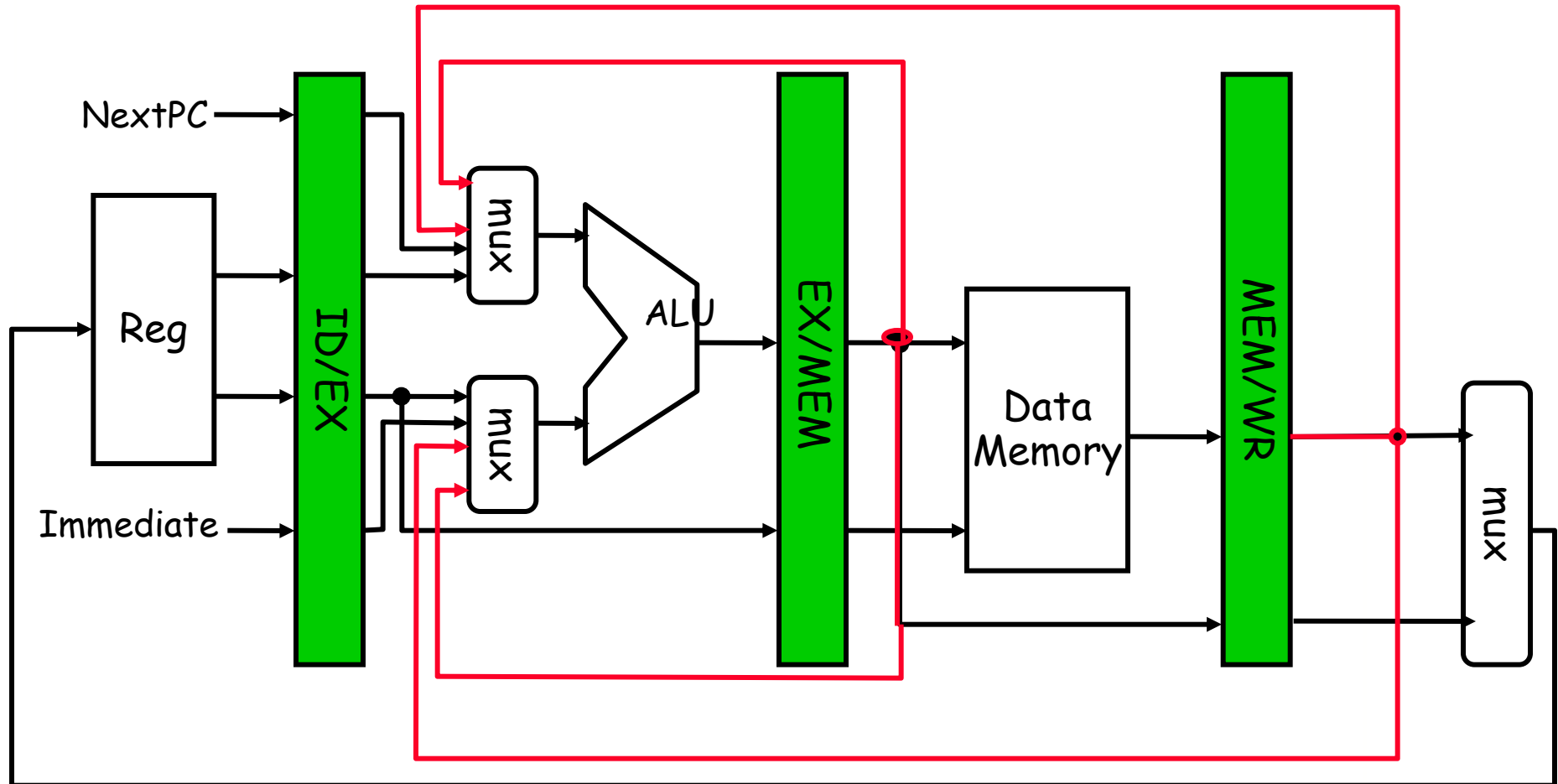
- ❖ Data hazard: instruction needs data from the result of a previous instruction still executing in pipeline
- ❖ **Solution:** Forward data if possible.



Operand Forwarding to Avoid Data Hazard

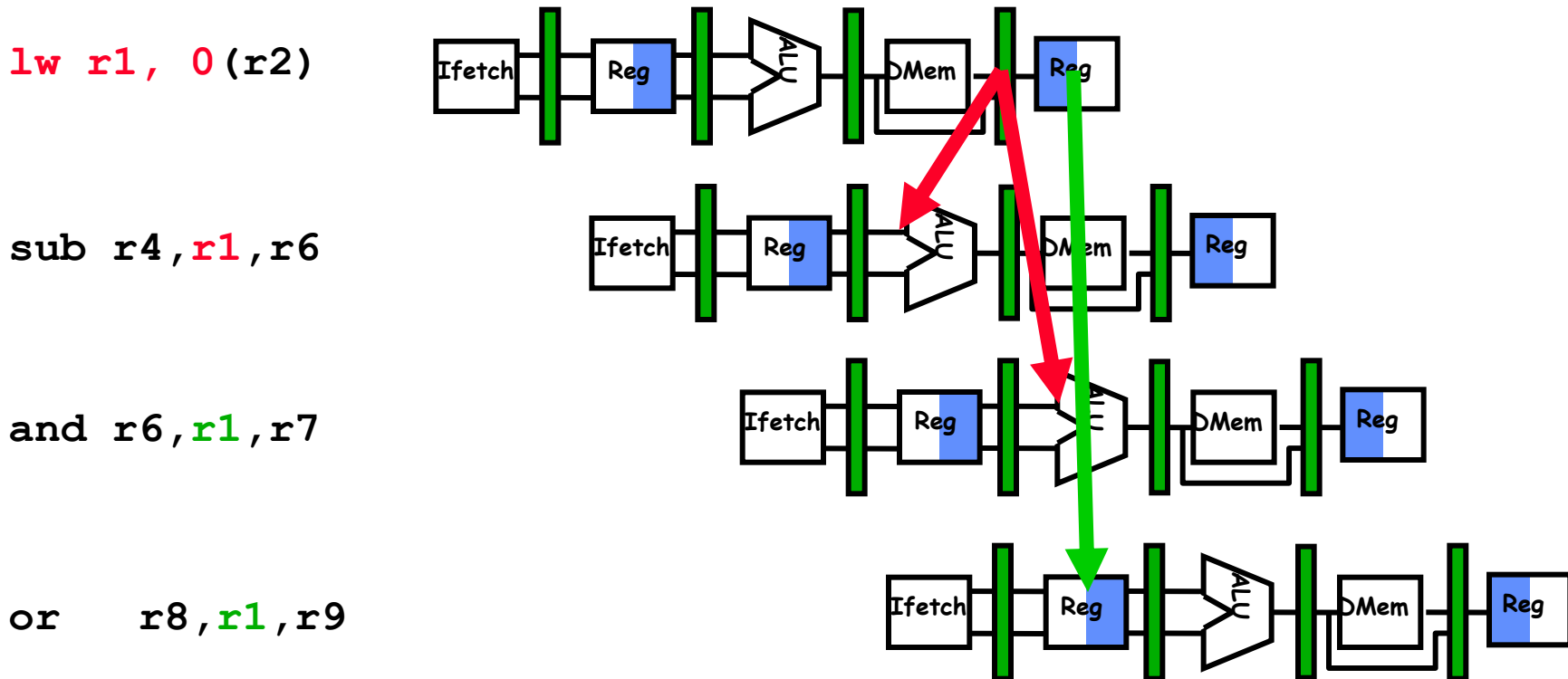


Hardware Change for Forwarding



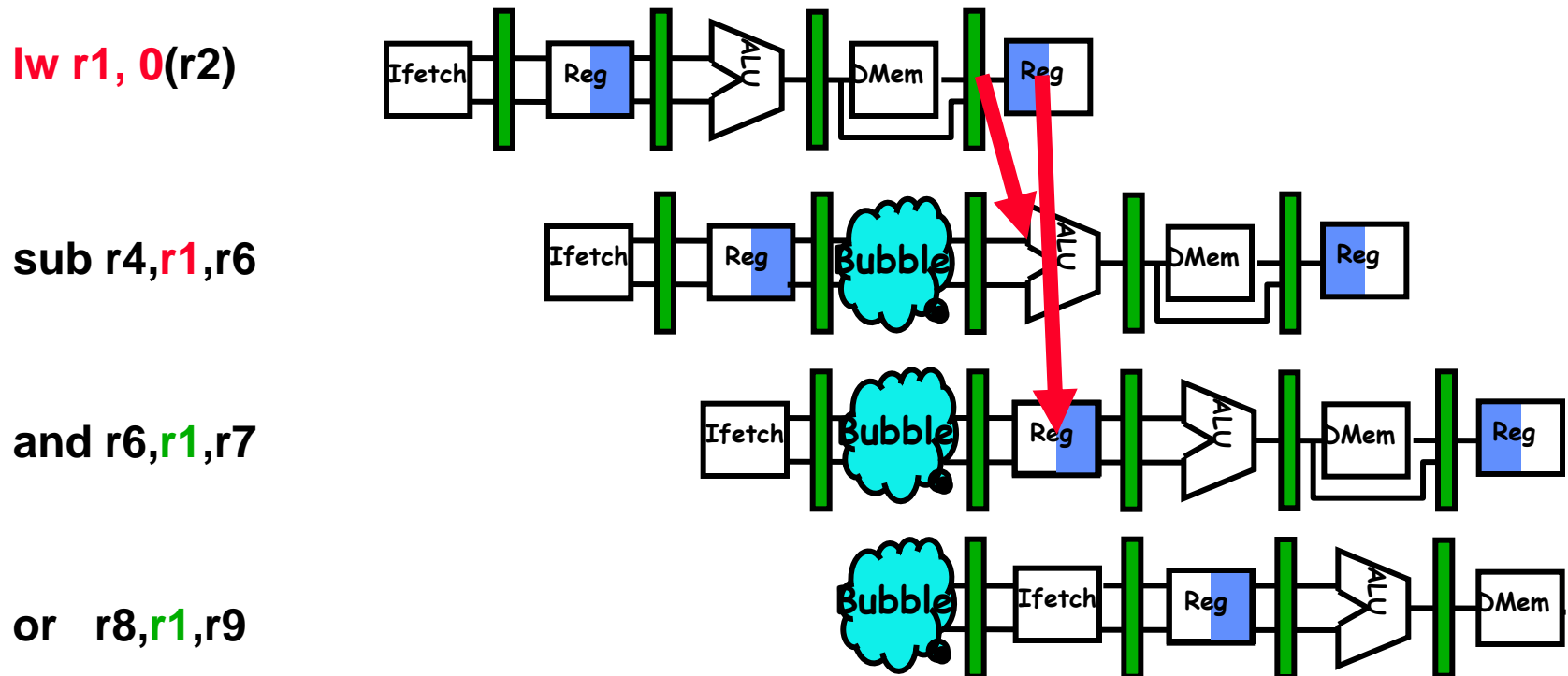
Data Hazard even with Operand Forwarding

Time (clock cycles)



Resolving the Load-ALU Hazard

Time (clock cycles) →



Software Scheduling for Load Hazards

Assume a, b, c, d, e, and f in memory.

a = b + c;

d = e - f;

LW Rb,b

LW Rc,c

ADD Ra,Rb,Rc

SW a,Ra

LW Re,e

LW Rf,f

SUB Rd,Re,Rf

SW d,Rd

LW Rb,b

LW Rc,c

LW Re,e

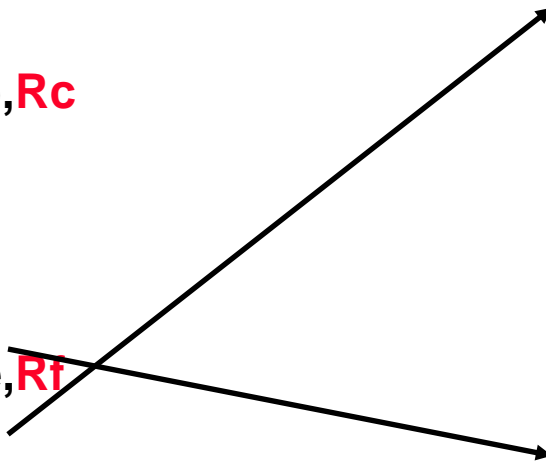
ADD Ra,Rb,Rc

LW Rf,f

SW a,Ra

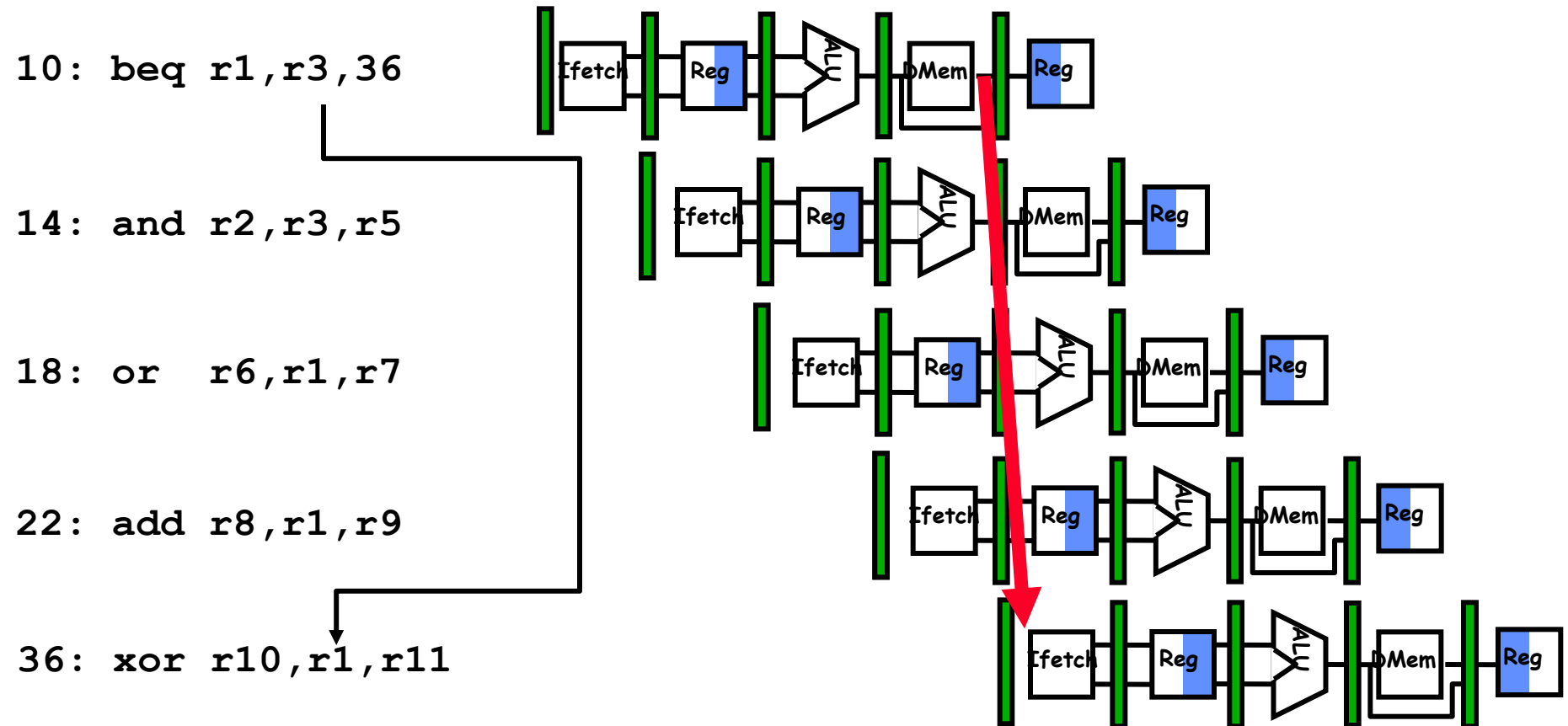
SUB Rd,Re,Rf

SW d,Rd



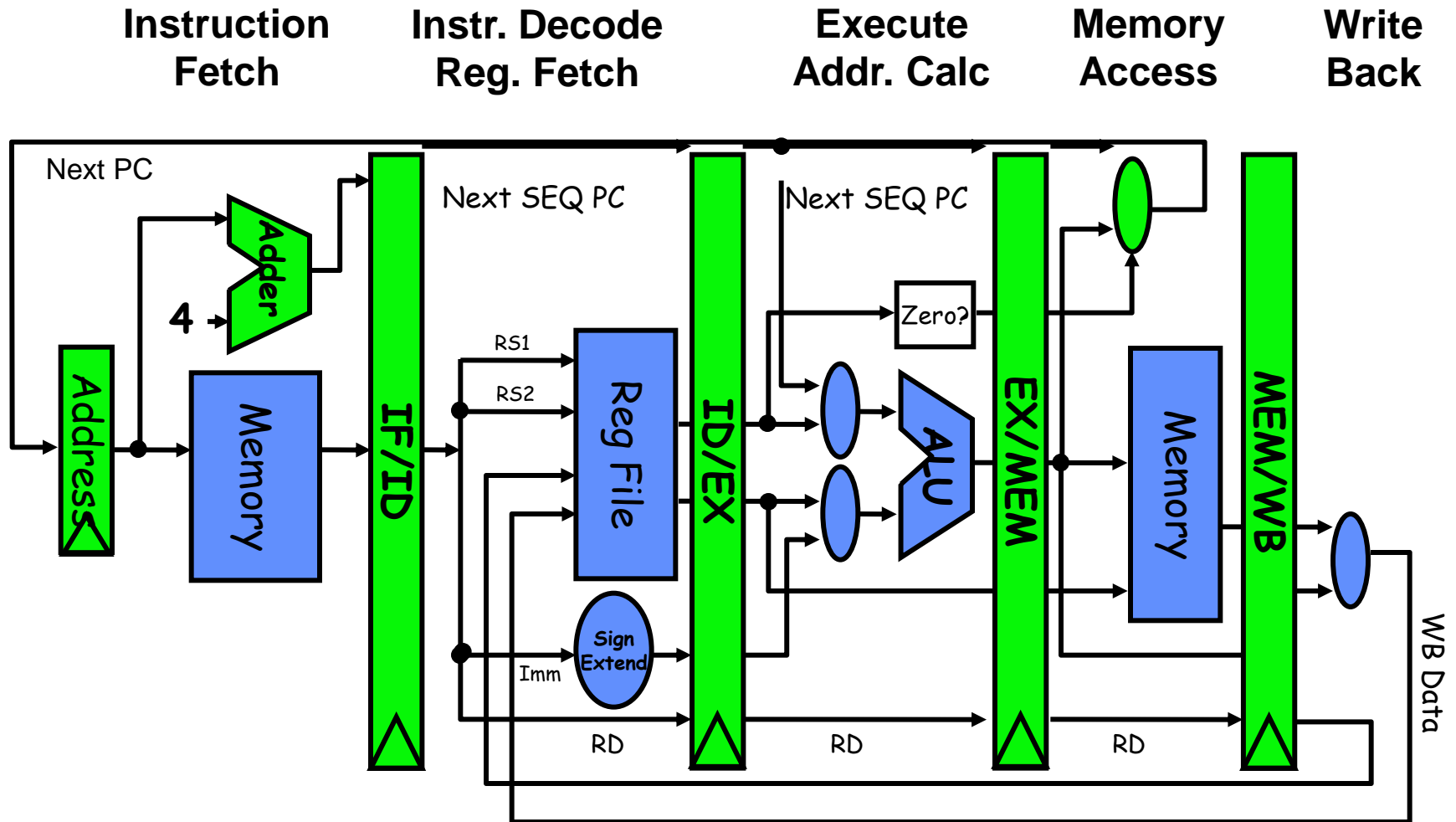
Control Hazard on Branches

=> Three Stage Stall



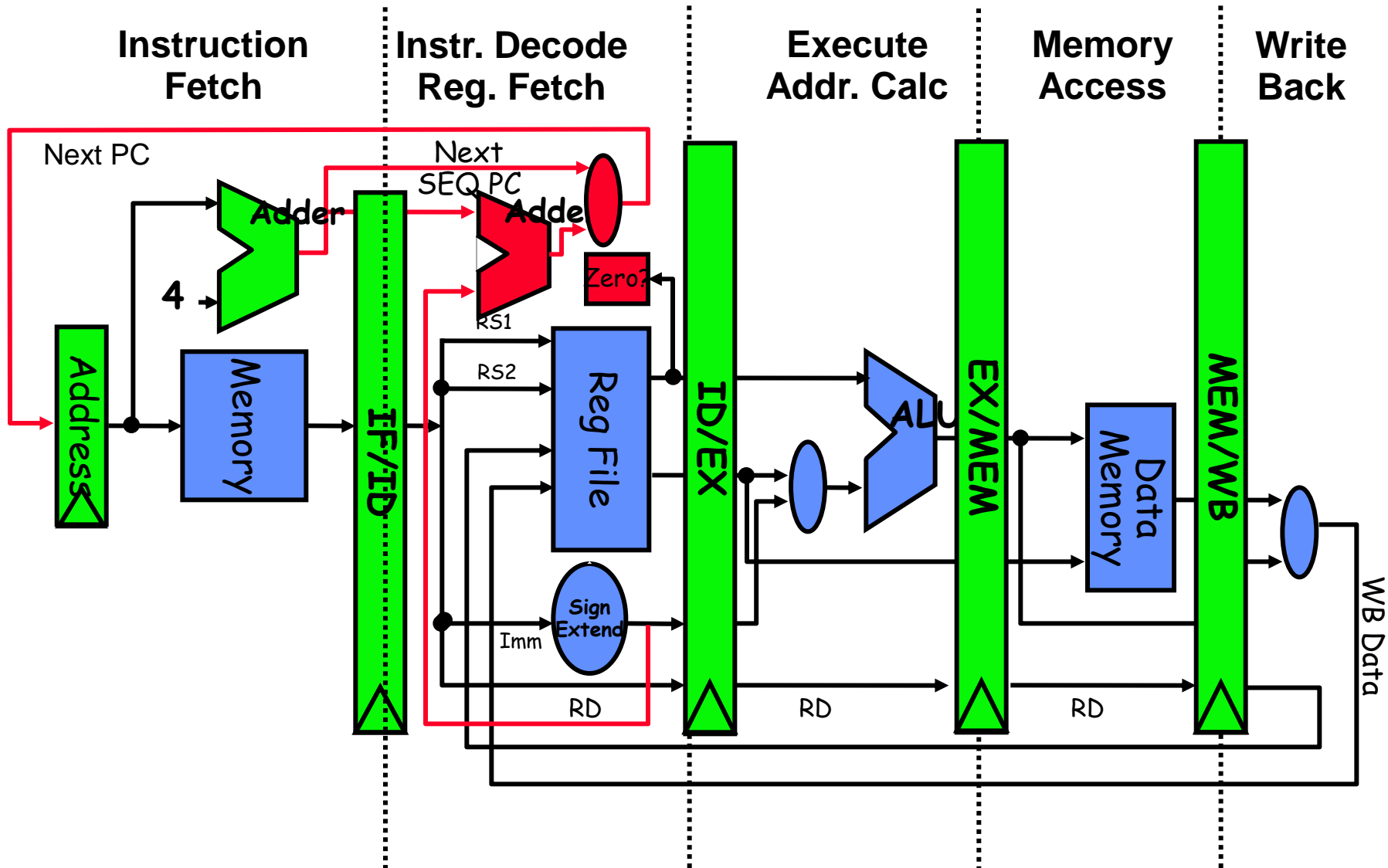
Conventional MIPS Pipeline

Branching at 4th stage



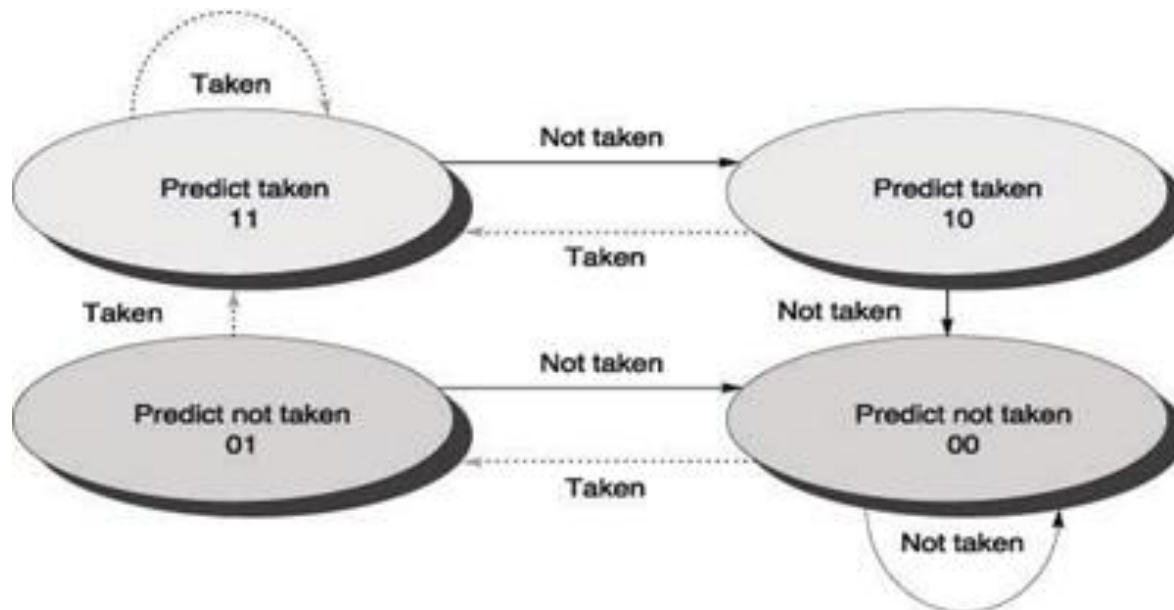
Branch Optimized MIPS Pipeline

Branching at 2nd stage



Dynamic Branch Prediction

- ❖ Use a **Branch Prediction Buffer (BPB)**
- ❖ Records previous outcomes of the branch instruction.
- ❖ Refer BPB at IF stage and act on it at ID stage.
- ❖ A FSM can model prediction





johnjose@iitg.ac.in
<http://www.iitg.ac.in/johnjose/>