

Computer Architecture

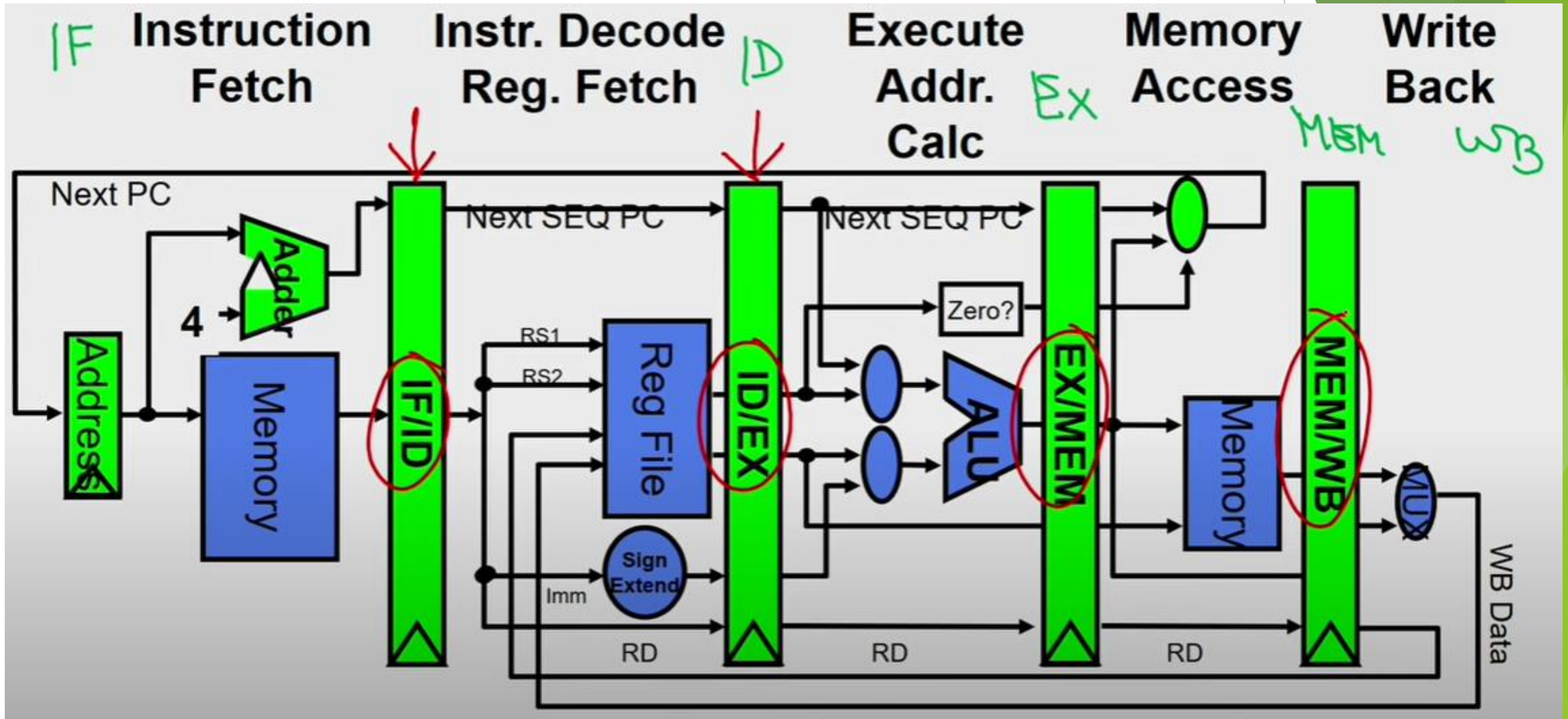
Pipeline Hazards

Dr. Mohammad Reza Selim

Lecture Outline

- ▶ Pipeline in ideal condition
- ▶ What is pipeline Hazard?
- ▶ Structural Hazards
- ▶ Data Hazards
- ▶ Control Hazards

Pipelined RISC Datapath



Instruction Execution Cycle

- ❖ Each instruction can take at most 5 clock cycles
- ❖ **Instruction fetch cycle (IF)**
- ❖ **Instruction decode/register fetch cycle (ID)**
- ❖ **Execution/Effective address cycle (EX)**
- ❖ **Memory access cycle (MEM)**
- ❖ **Write-back cycle (WB)**



Visualizing the Pipeline

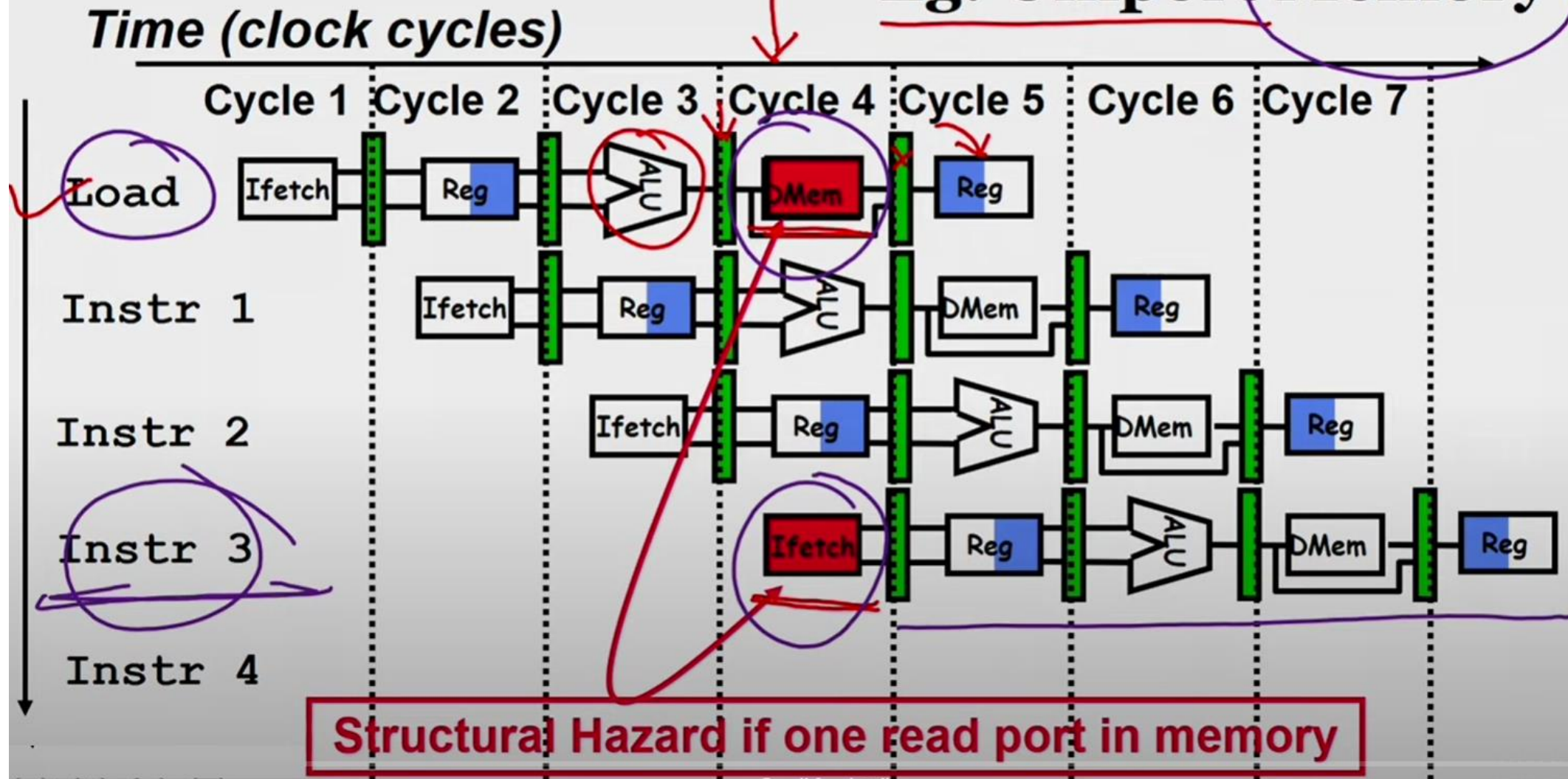
Clock number								
Instruction number	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
$i+1$		IF	ID	EX	MEM	WB		
$i+2$			IF	ID	EX	MEM	WB	
$i+3$				IF	ID	EX	MEM	WB
$i+4$					IF	ID	EX	MEM

Pipeline Hazards

- ❖ **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 resource
 - ❖ **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



Detecting and Resolving Structural Hazard

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

- ❖ ~~Solution 1: Wait~~

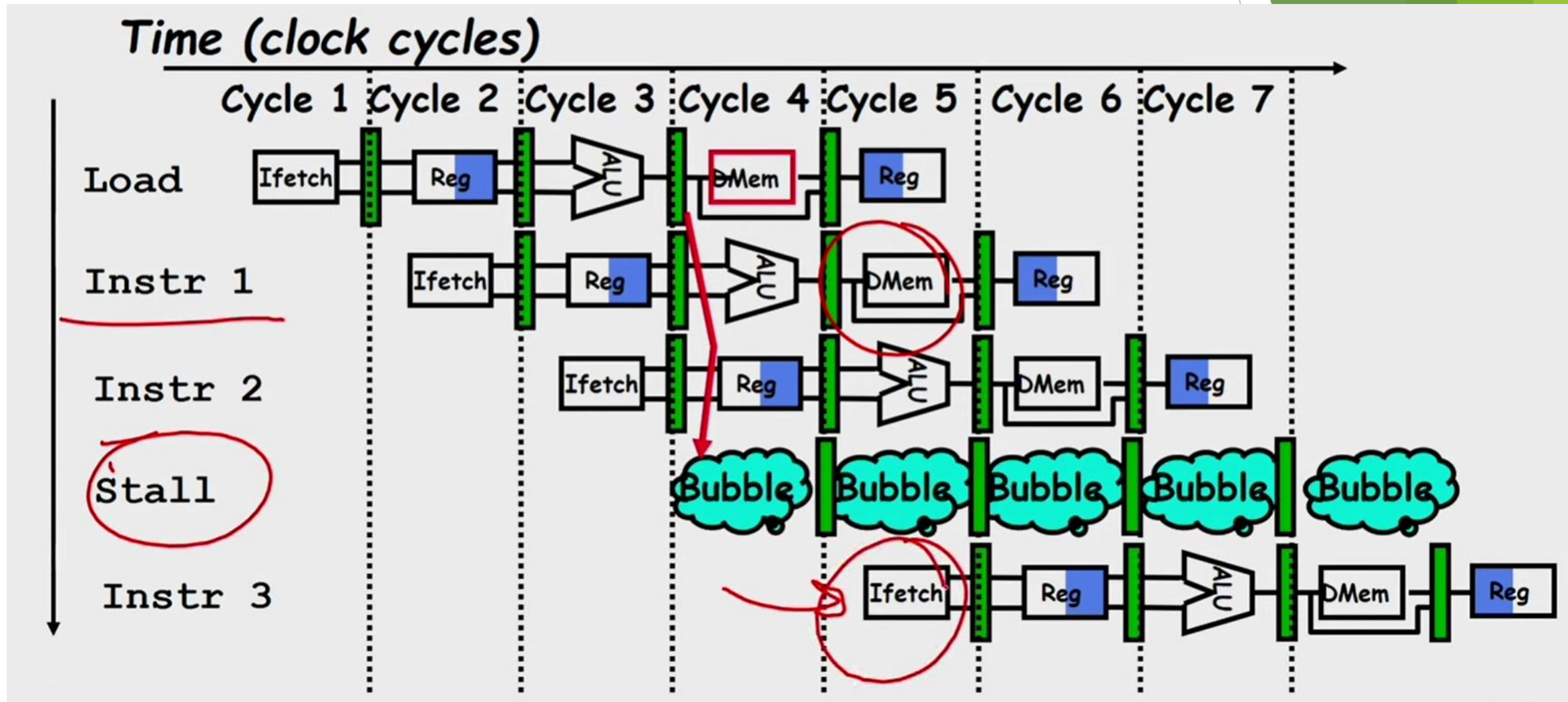
 - ❖ must detect the hazard

 - ❖ must have mechanism to stall

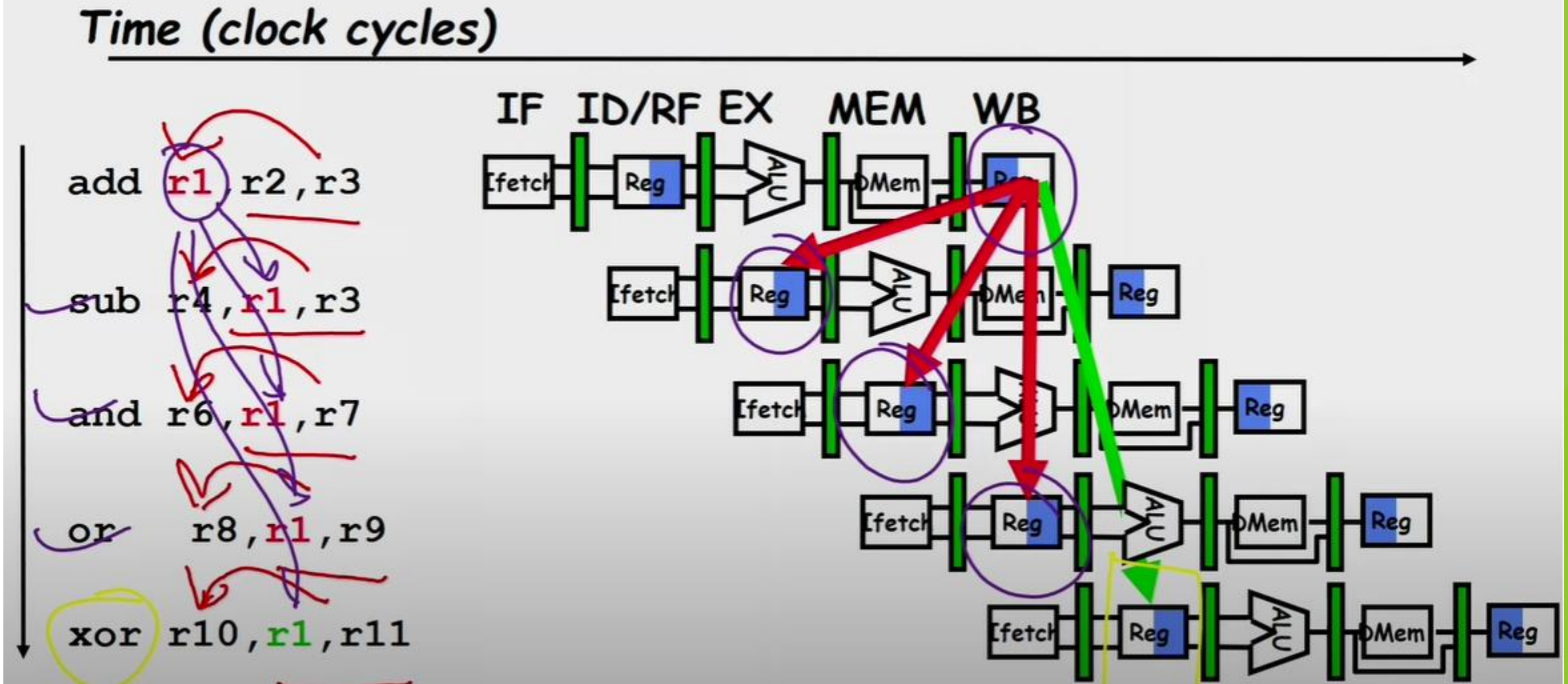
- ❖ ~~Solution 2: Duplicate hardware~~

 - ❖ Multiple such units will help both instruction to progress

Detecting and Resolving Structural Hazard



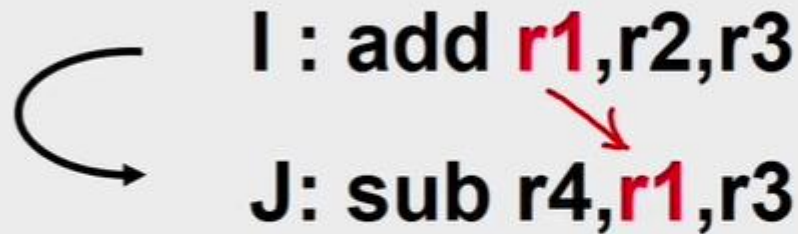
Data Hazards



Three Types of Data Hazards (1)

❖ 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 Types of Data Hazards (2)

❖ Write After Read (WAR)

Instr_j writes operand **before** Instr_i reads it

- ❖ 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

I: sub r4, r1, r3
J: add r1, r2, r3
K: mul r6, r1, r7

Three Types of Data Hazards (3)

❖ Write After Write (WAW)

Instr_j writes operand before Instr_i writes it.

❖ Called an output dependence

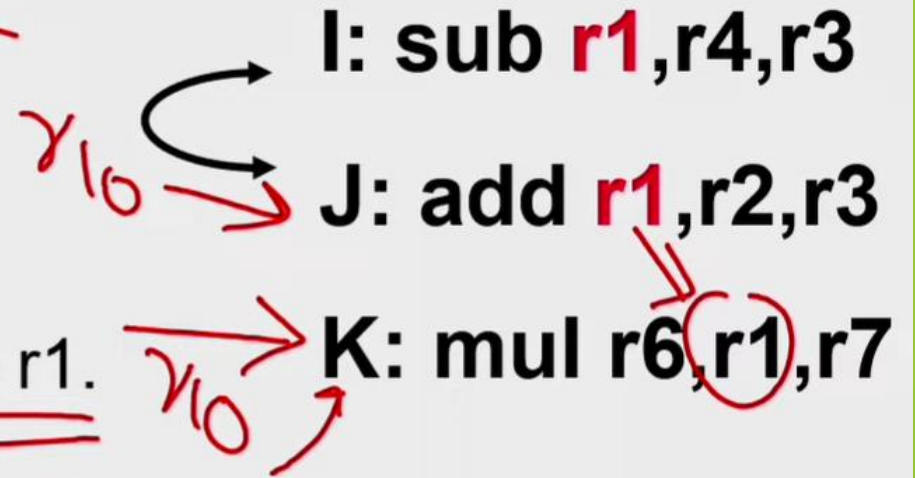
❖ This also results from the reuse of name r1.

❖ 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 Hazards

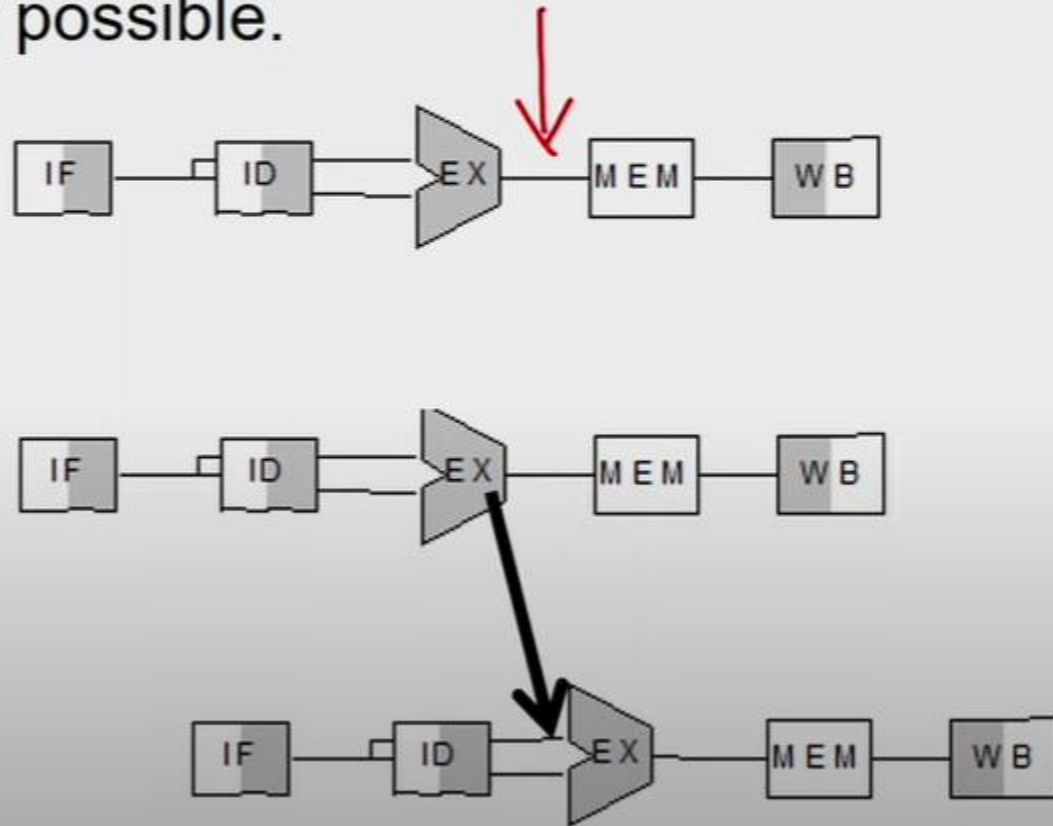
- ❖ Data hazard: instruction needs data from the result of a previous instruction still executing in pipeline
- ❖ **Solution:** Forward data if possible.

$R_1 \leftarrow R_2 + R_3$

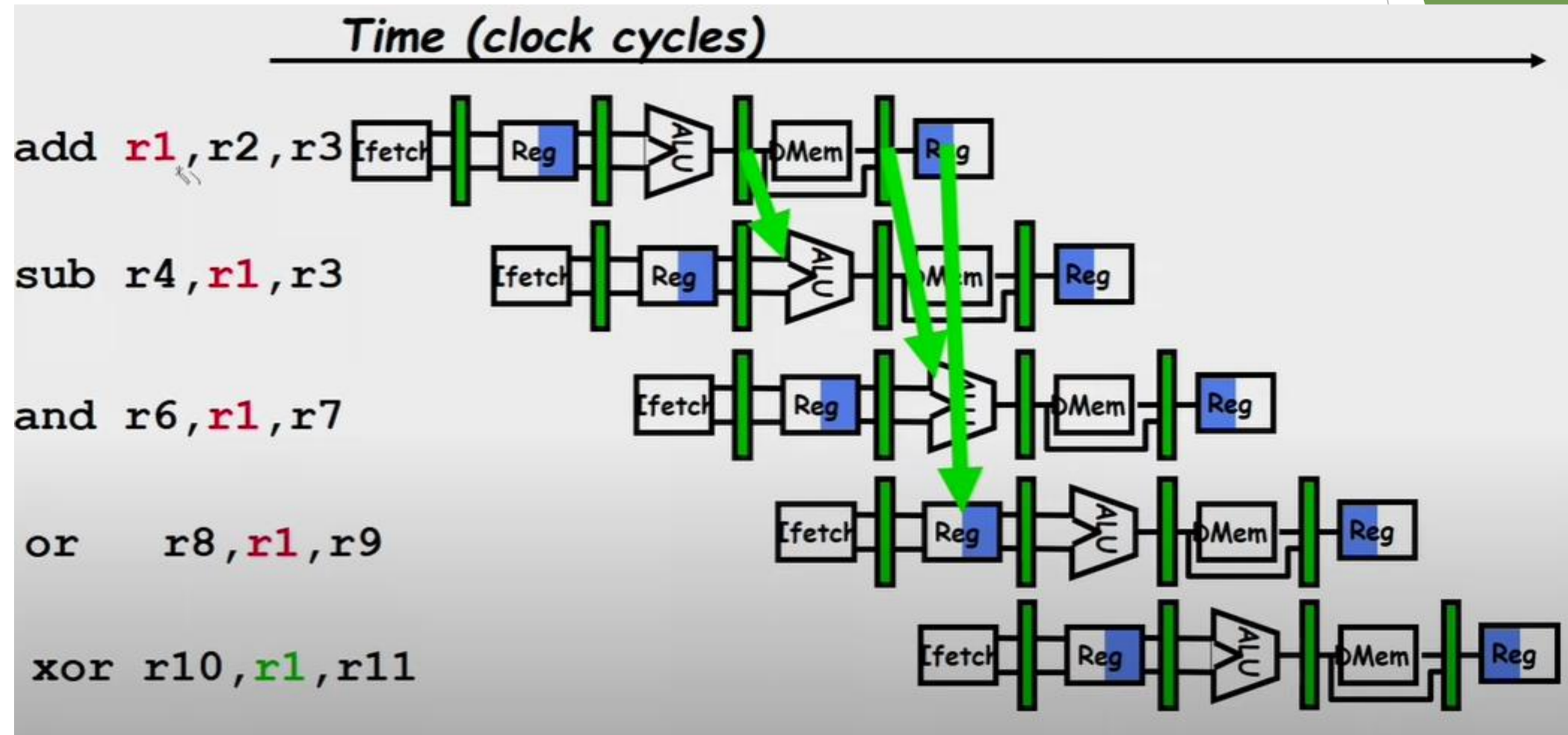
add R1, R2, R3

add R1, R2, R3

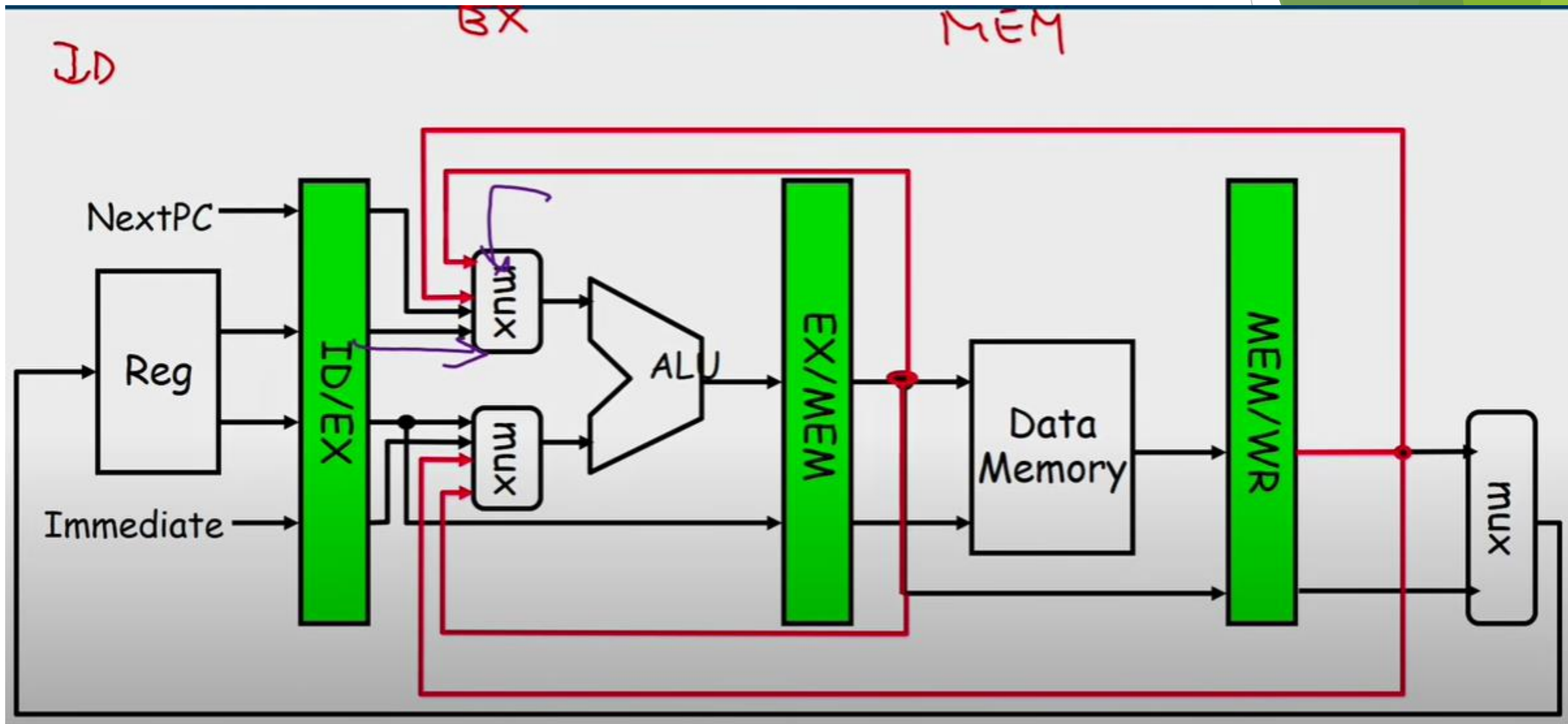
sub R4, R1, R5



Operand Forwarding to Avoid Data Hazards



Hardware Change for Operand Forwarding



Data Hazards even with Operand Forwarding

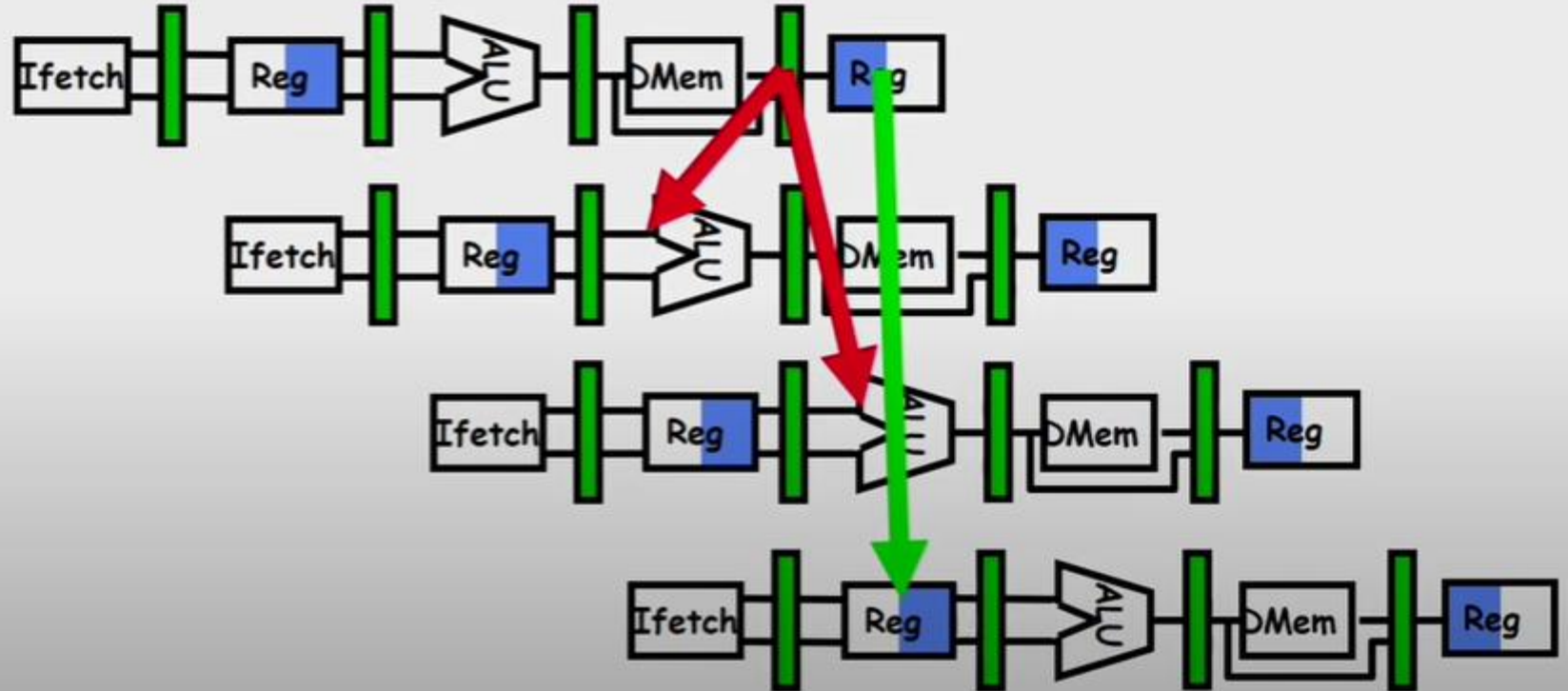
Time (clock cycles) →

lw r1, 0(r2)

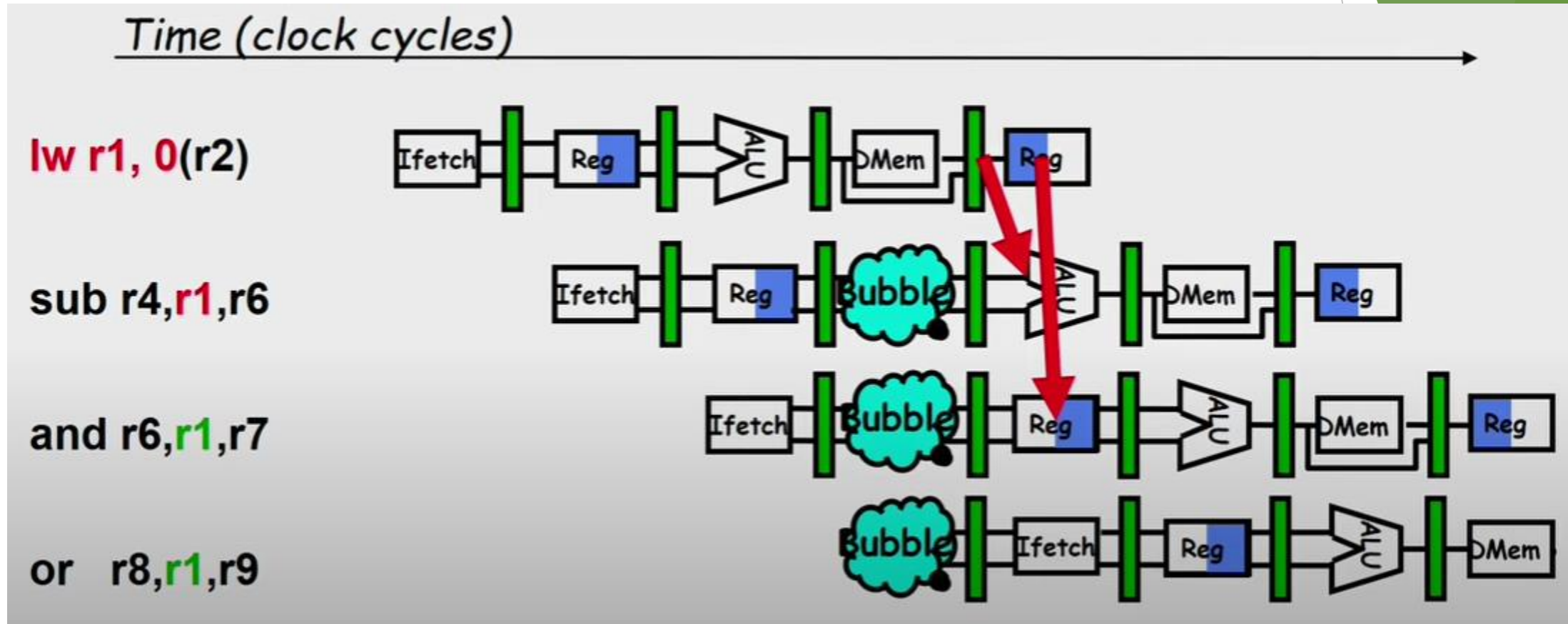
sub r4, r1, r6

and r6, r1, r7

or r8, r1, r9



Resolving the Load-ALU Hazard



Software Scheduling for Load Hazards

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

✓ $a = b + c;$ $b = 8(Rx)$
✓ $d = e - f;$

