

CS/ECE 5381/7381
Computer Architecture
Spring 2023

Dr. Manikas
Computer Science
Lecture 7: Feb. 14, 2023

Project 1

- Due Tue., Feb. 21 (11:59 pm)
- First programming project – MARS tool
- Assignment:
 - Run tutorial to get familiar with MARS tool
 - You will use this tool in upcoming projects

MARS

- MARS (**MIPS Assembler and Runtime Simulator**)
- Free JAR file (Java) to download and run on PC
- Run tutorial with sample program
 - MIPS assembly code (recall App. A)

Pipelining: Basic and Intermediate Concepts

(Appendix C, Hennessy and Patterson)

Note: some course slides adopted from
publisher-provided material

Notes

- This section will be a review of basic pipelining principles covered in CS 4381/ECE 3382 or equivalent course

Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations

Introduction

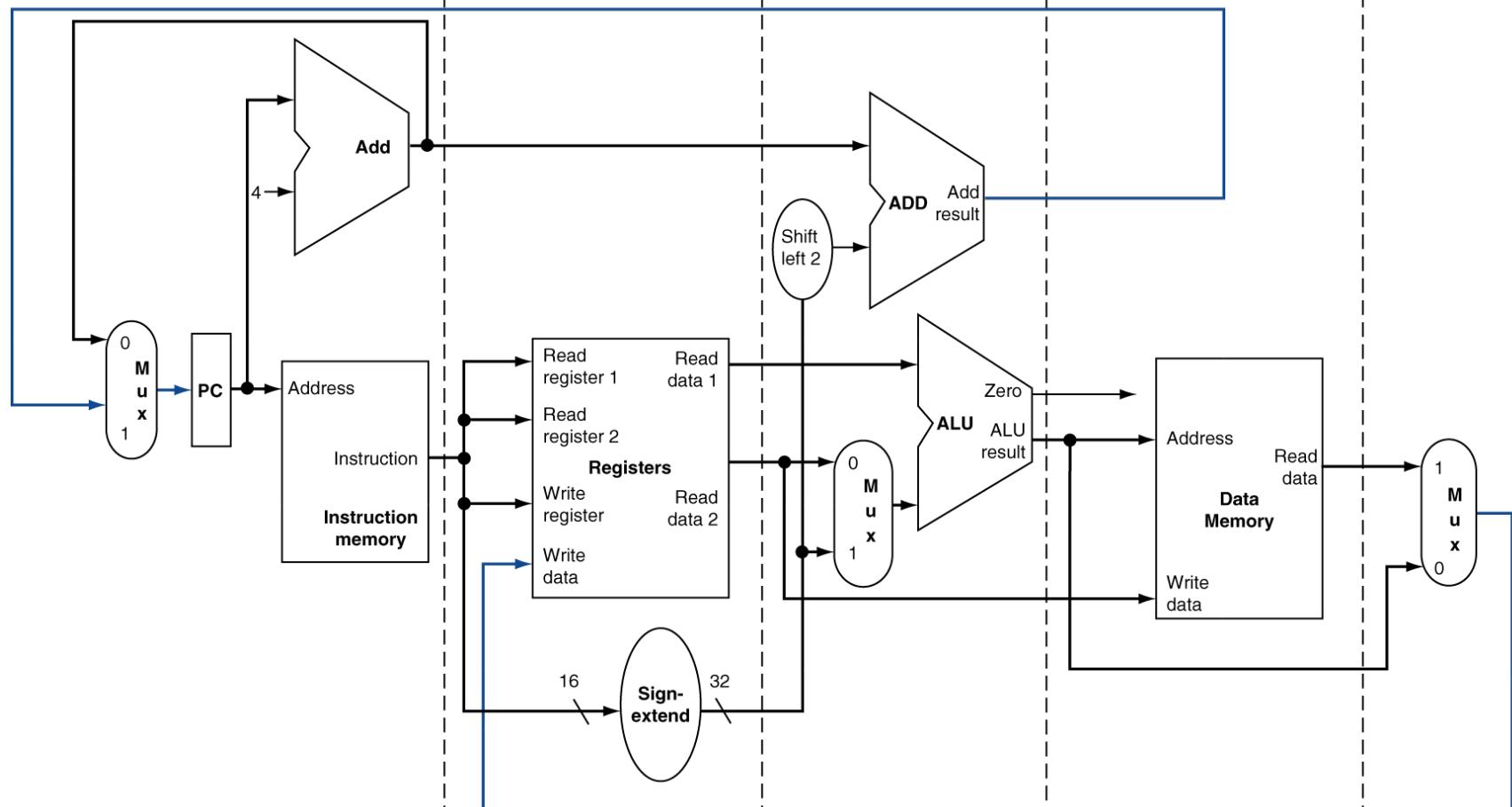


- Pipelining: multiple instructions overlapped during execution
- Uses parallelism of execution operations
- Speeds up CPU operation

MIPS Datapath Stages

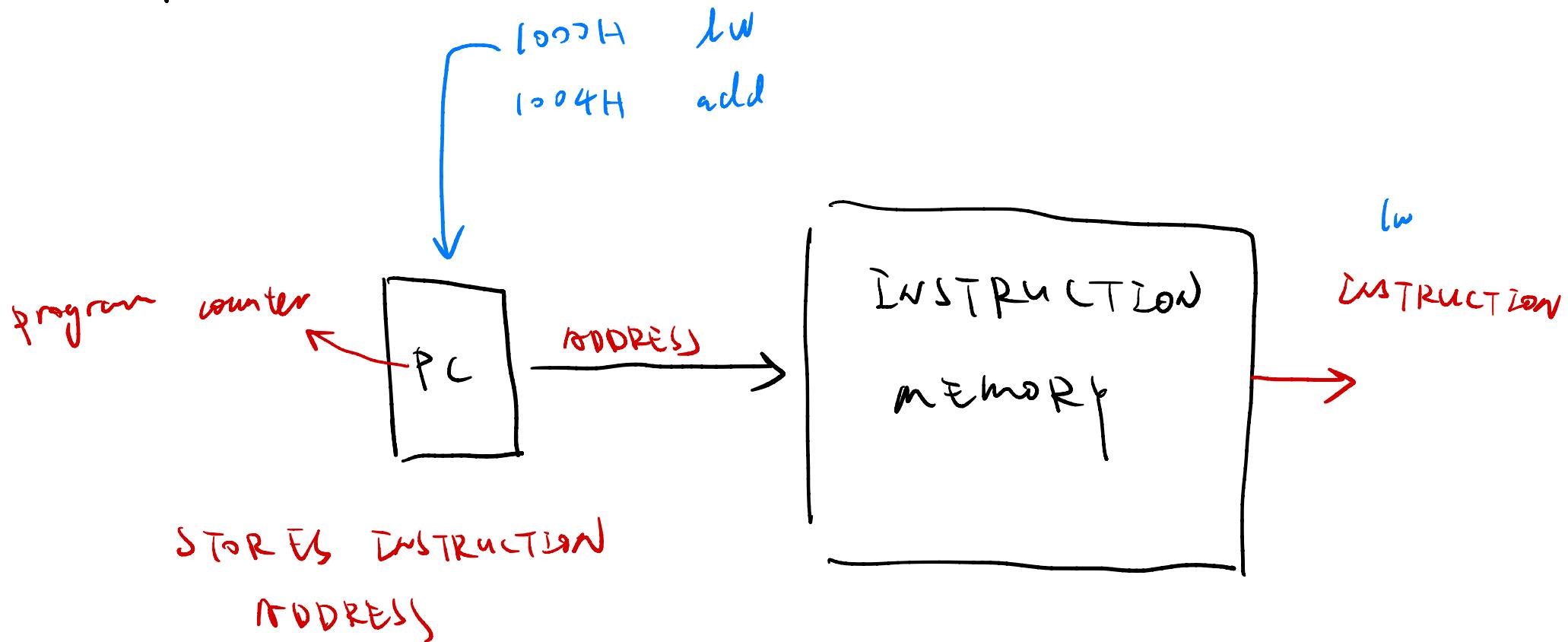
- Five stages, one step per stage
 1. IF: Instruction fetch from memory
取指
 2. ID: Instruction decode & register read
 3. EX: Execute operation or calculate address
 4. MEM: Access memory operand
 5. WB: Write result back to register

IF ID EX MEM WB



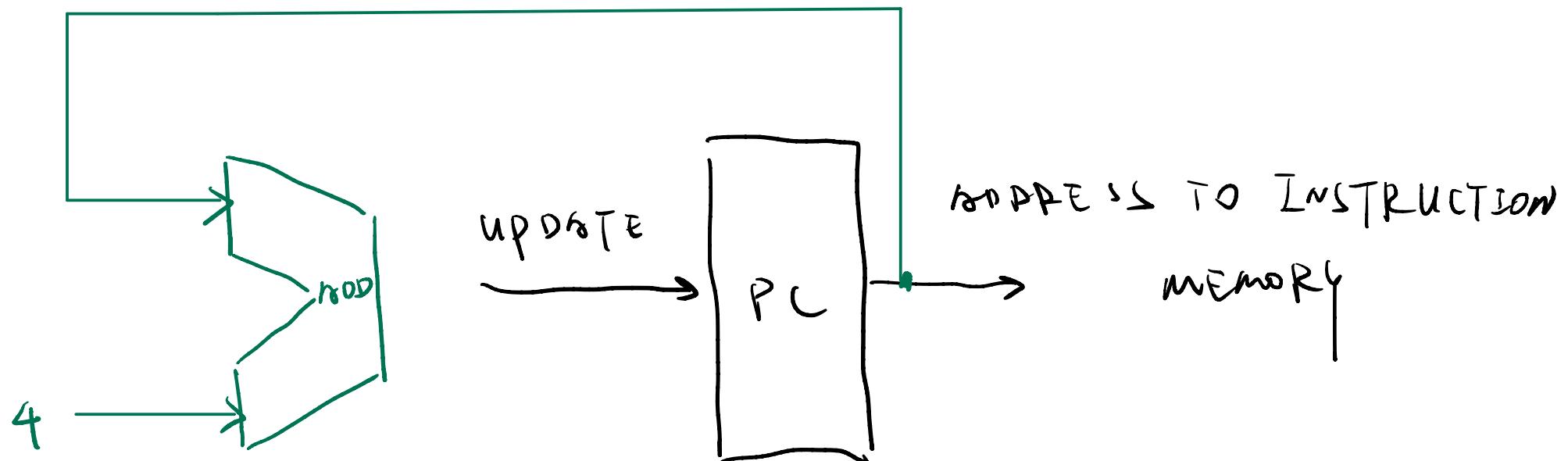
FIRST STAGE: IF (INSTRUCTION FETCH)

- Task? FETCH INSTRUCTIONS.



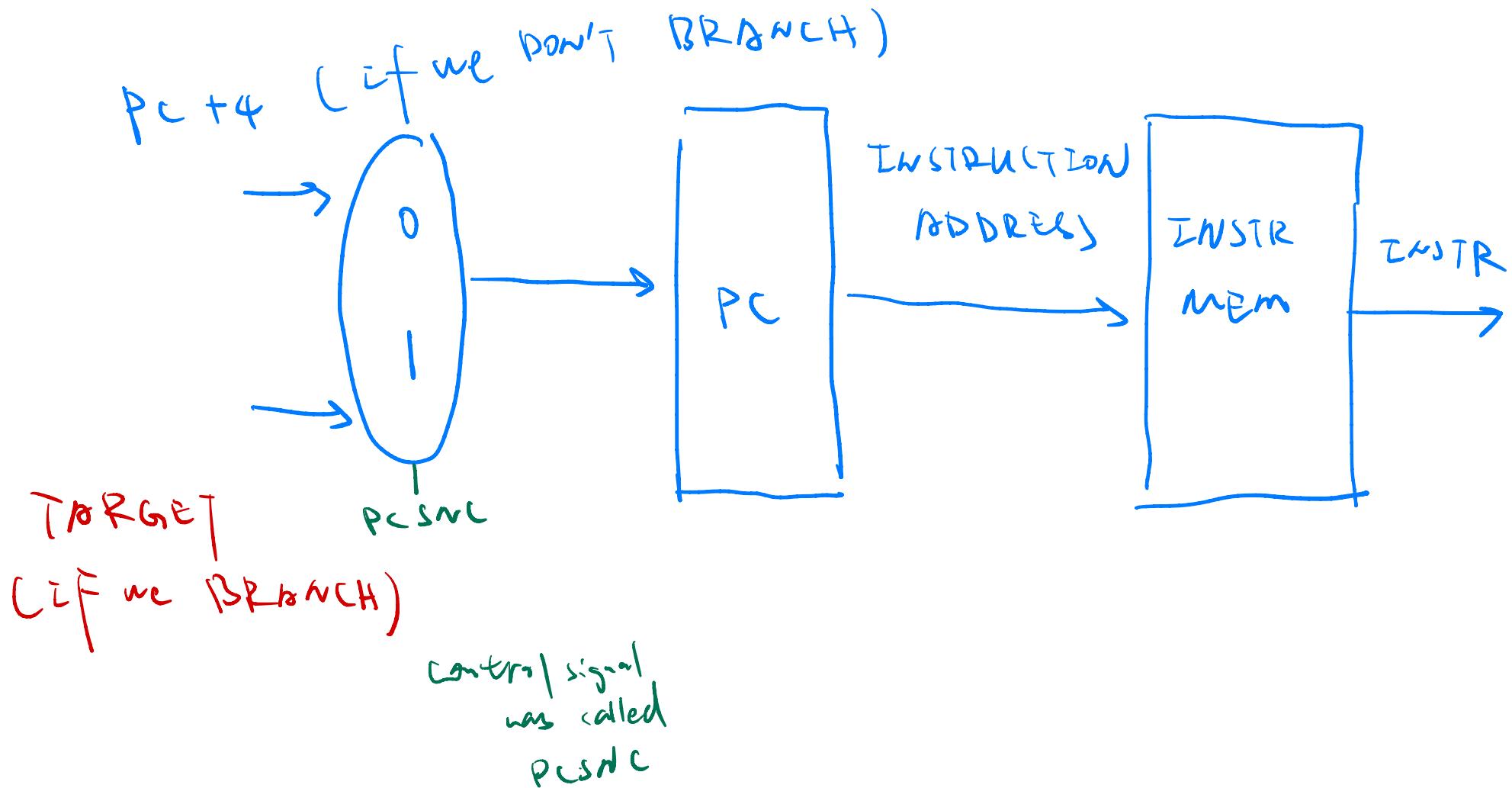
VALUE IN PC? NEXT INSTR IN seq
OR BRANCH TARGET sequence

NEXT IN SEQUENCE? $PC \leftarrow PC + 4$



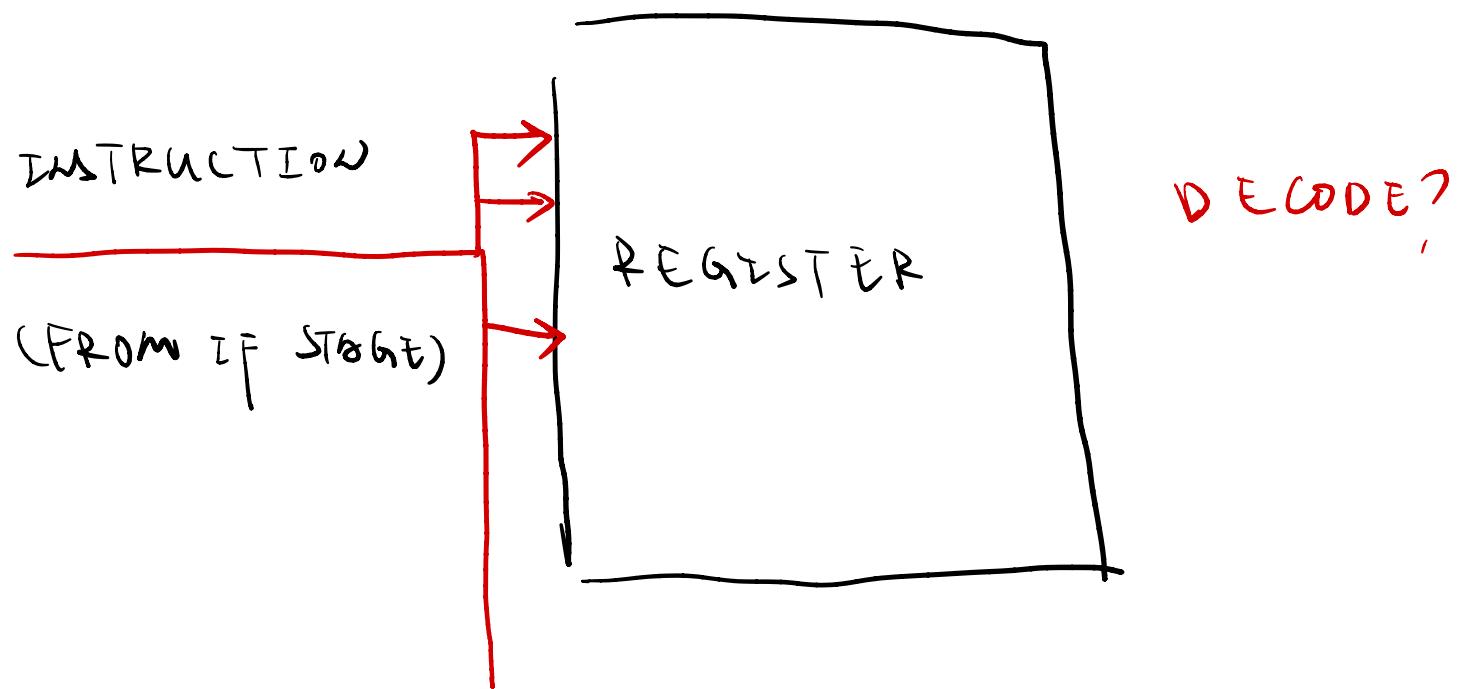
What if we BRANCH?

$$\text{RECALL: TARGET} = \text{UPDATE PC} + \underbrace{4 \text{ (OFFSET)}}_{\text{CALCULATED ELSEWHERE}}$$



SECOND STAGE : ID

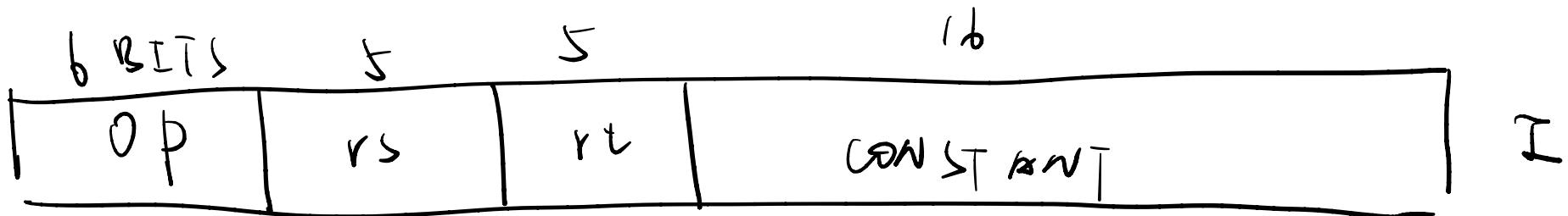
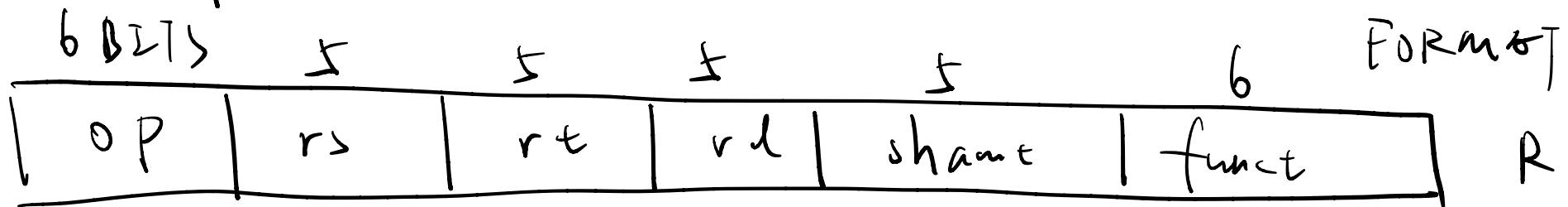
Task : INSTRUCTION DECODE
AND READ REGISTER



INSTRUCTION DECODE? You DID This!

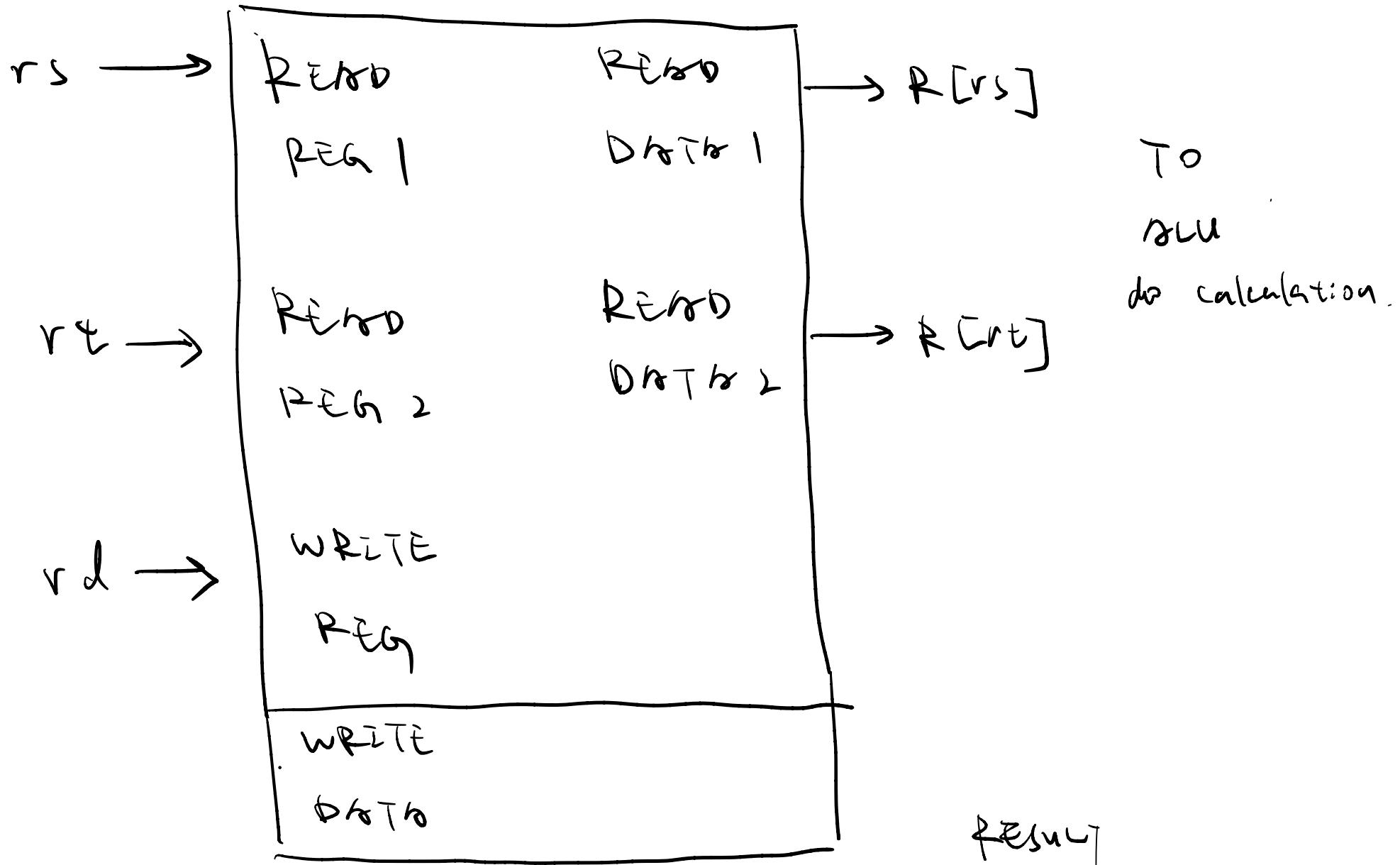
RECALL FROM RPP #: 32-BIT INSTR

⇒ SPLIT INTO FIELDS

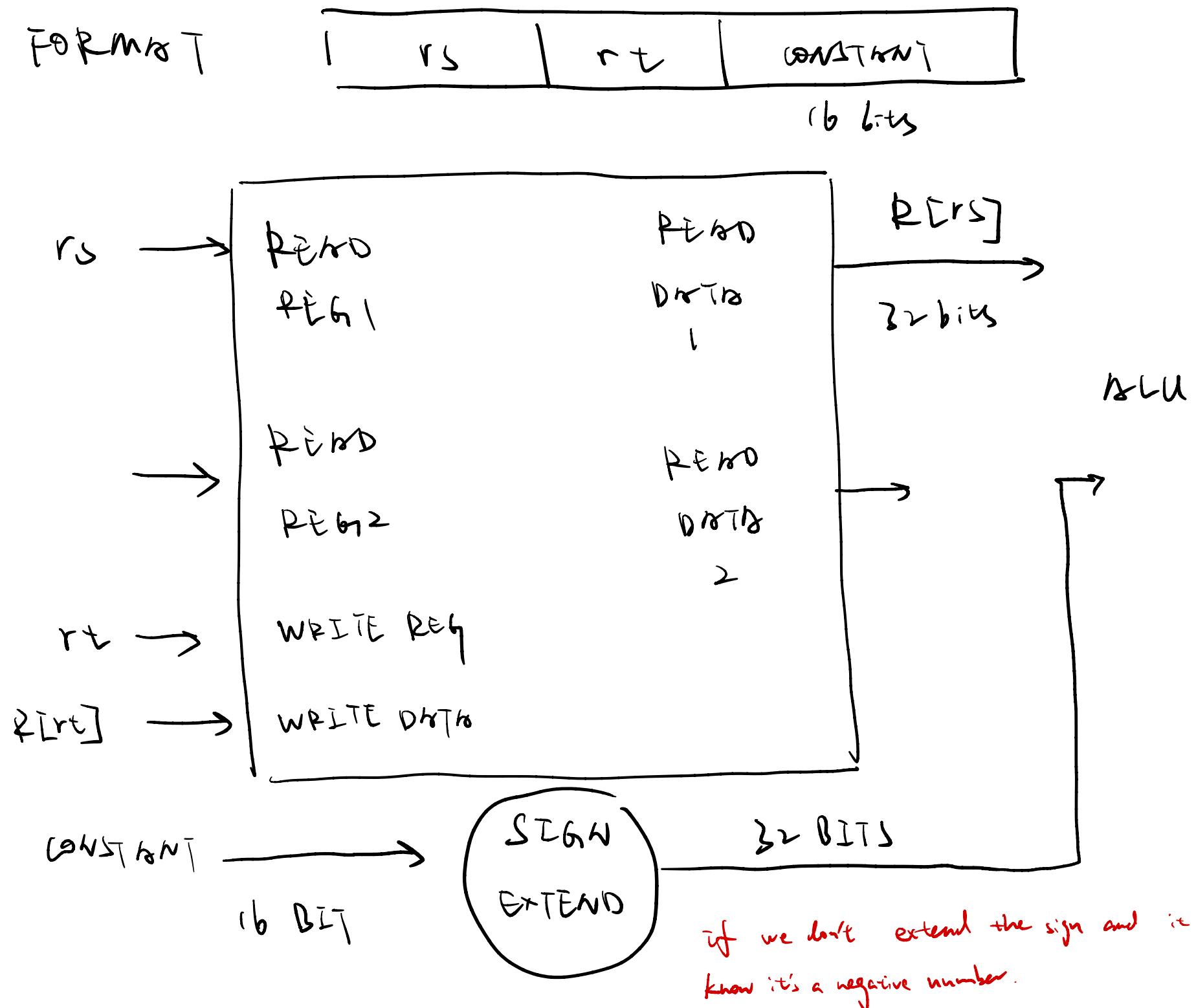


R FORMAT

rs	rt	rd
----	----	----

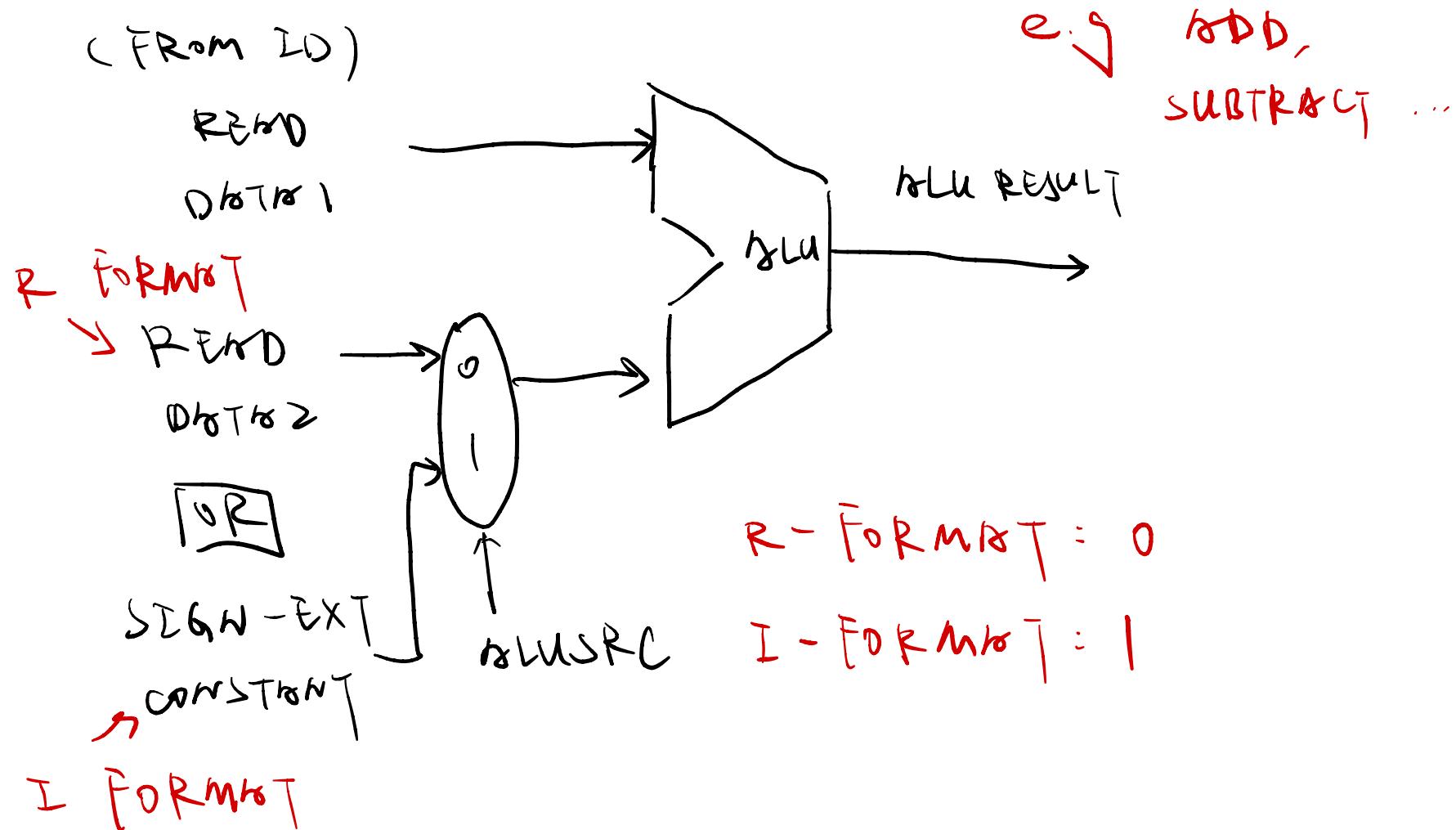


I Format



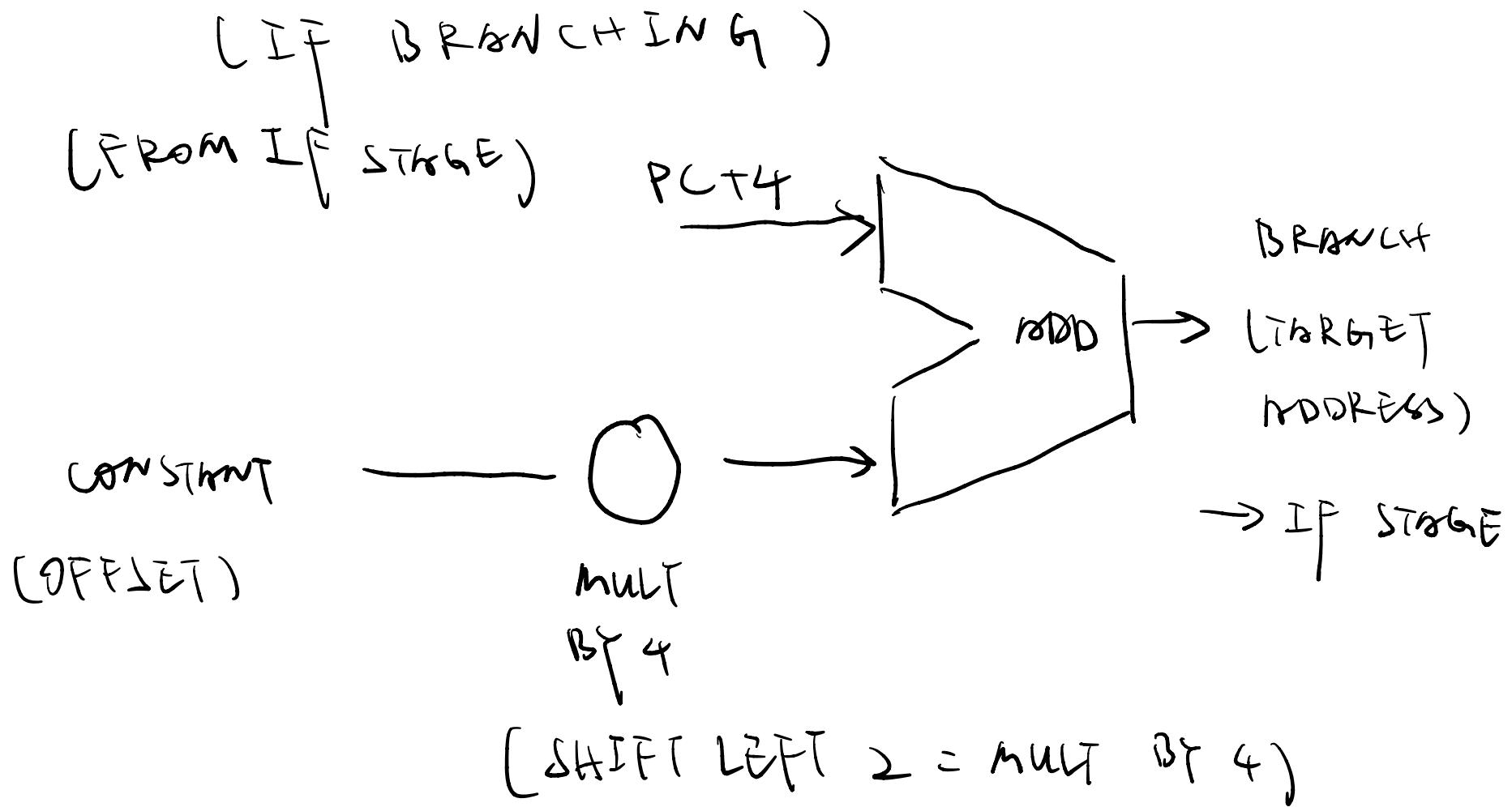
3rd STAGE = EX (EXECUTE)

- "DO THE MATH"



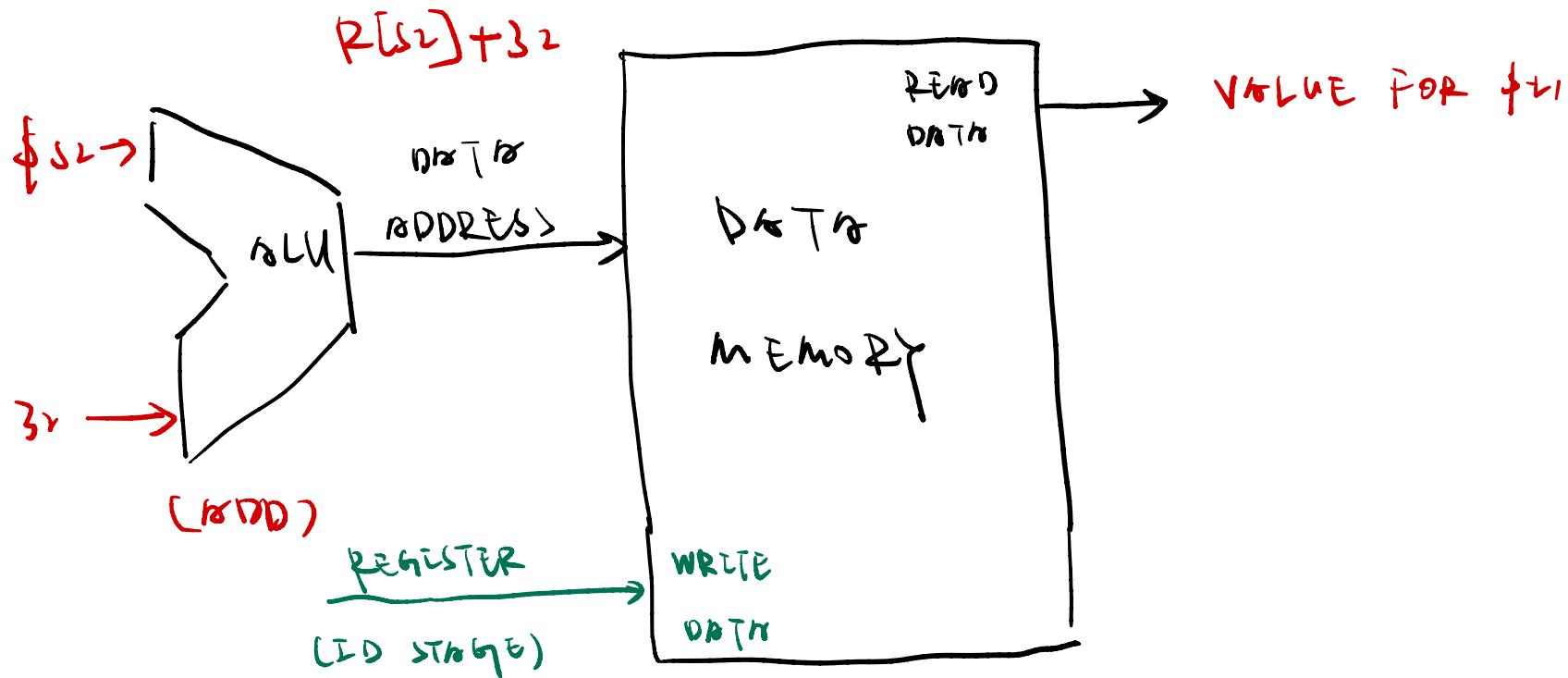
What else in EX STAGE?

CALCULATE BRANCH ADDRESS



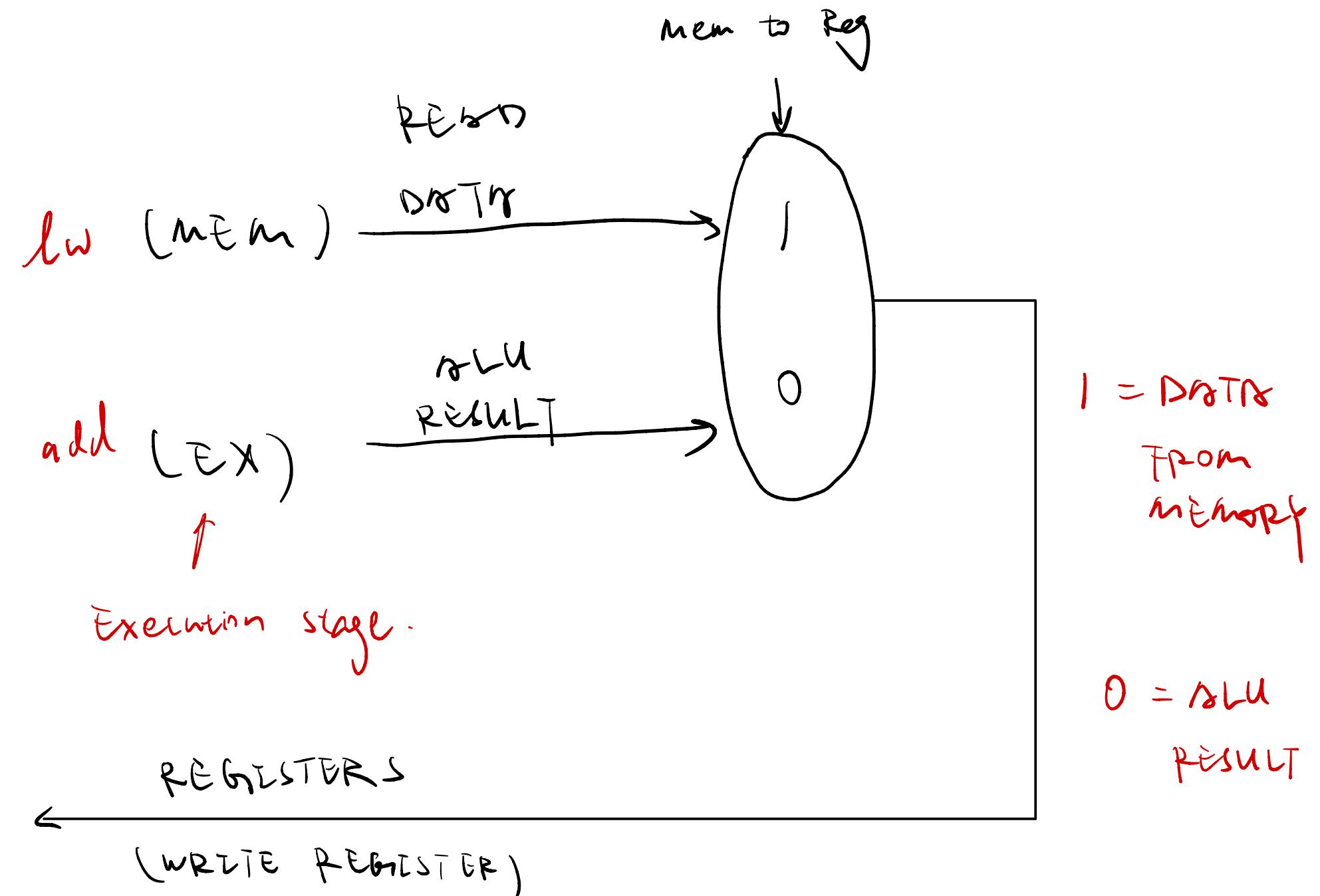
Mem Store GE (4th STAGE)

→ DTR Memory Access (lw, sw)
(w & t1, 32(\$32))



sw \$t1, 32(\$32)

5th Stage: WB (WRITE Back)



Example: assume we have the following MIPS instructions:

lw \$t0, 100(\$s0)

lw \$t1, 200(\$s1)

lw \$t2, 300(\$s2)

These instructions are essentially
independent here.

But what if somebody is changing
variables? → pipeline hazards.

We have three load operations in sequence: recall load
operation for first instruction does $R[t0] \leftarrow \text{Mem}[R[s0] + 100]$

Stages for first instruction are:

1. **IF** – fetch lw instruction from instruction memory
2. **ID** – decode lw instruction and read register $R[s0]$
3. **EX** – calculate address $R[s0] + 100$
4. **MEM** – read $\text{Mem}[R[s0] + 100]$
5. **WB** – write this value to register $R[t0]$

Executing the first two instructions in sequence uses the following instruction cycles (assume 1 stage per cycle):

Cycle	1	2	3	4	5	6	7	8	9	10
lw \$t0, 100(\$s0)	IF	ID	EX	MEM	WB					
lw \$t1, 200(\$s1)						IF	ID	EX	MEM	WB

How many cycles to complete all 3 instructions? 15 (5 cycles per instruction)

$$3 \times 5 = 15$$

For n instructions, 5n cycles

Is there a more efficient way to process instructions?

Note that if the first instruction is decoding (ID), we can fetch the next instruction (IF). After the next instruction is in ID stage, we can fetch the third instruction. So try this approach:

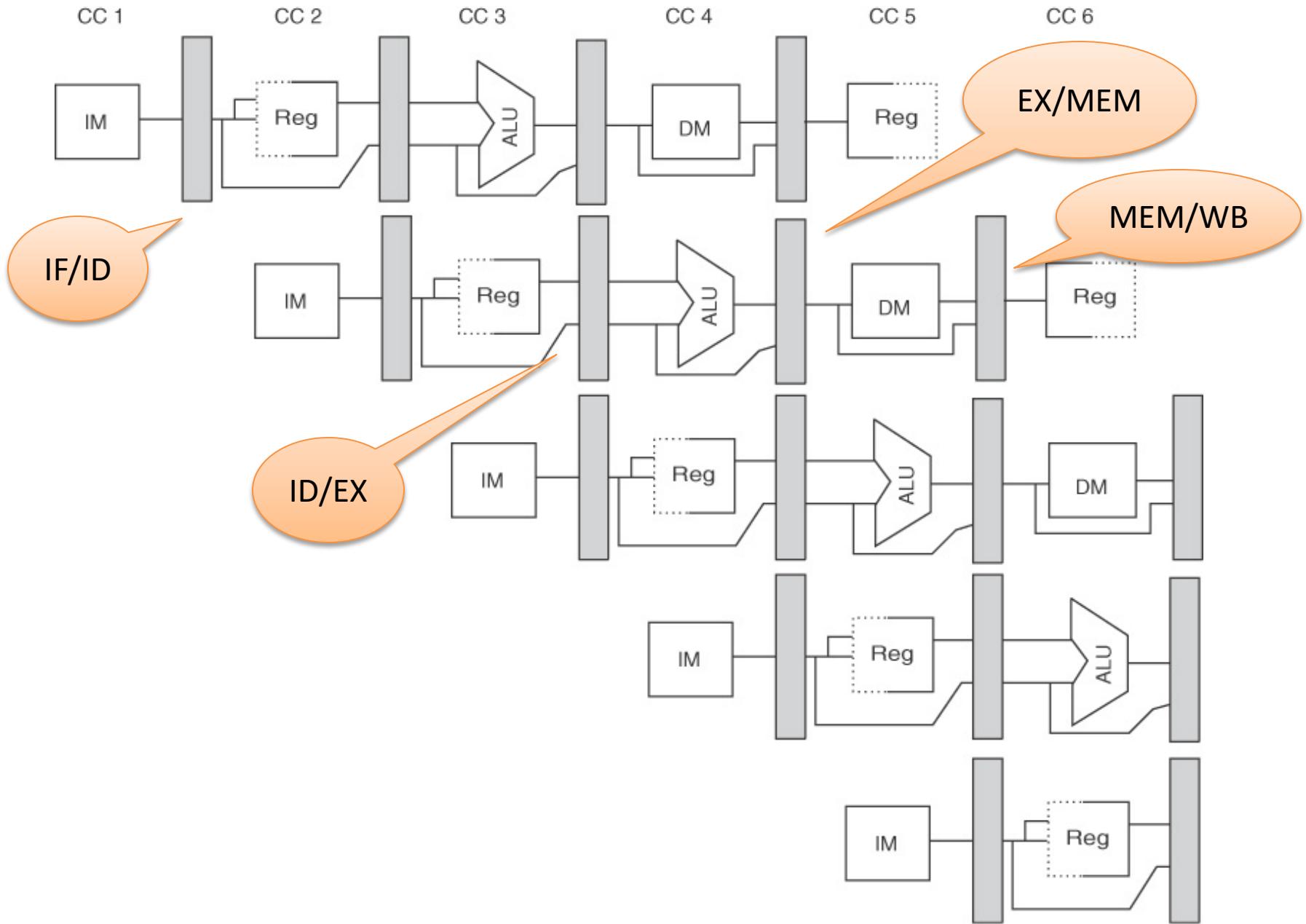
Cycle	1	2	3	4	5	6	7	8	9	10
lw \$t0, 100(\$s0)	IF	ID	EX	MEM	WB					
lw \$t1, 200(\$s1)		IF	ID	EX	MEM	WB				
lw \$t2, 300(\$s2)			IF	ID	EX	MEM	WB			

In this approach, the instructions are being processed in **parallel** – this is called ***pipelining***

How many cycles to complete all 3 instructions? 7 cycles
(instead of 15 with the sequential approach)

Pipeline Registers

- Holds data between pipeline stages
- 5 stages, so need 4 registers
 1. IF/ID – between IF and ID stages
 2. ID/EX
 3. EX/MEM
 4. MEM/WB



Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations

Pipeline Hazards

- Hazard – prevents next instruction from executing during designated clock cycle
 1. Structural hazards
 2. Data hazards
 3. Control hazards
- Use *stalls* to address pipeline hazards

Pipeline Stalls

- A stall delays instructions from entering the pipeline
 - Wait until hazard is addressed
- This will affect processing speed

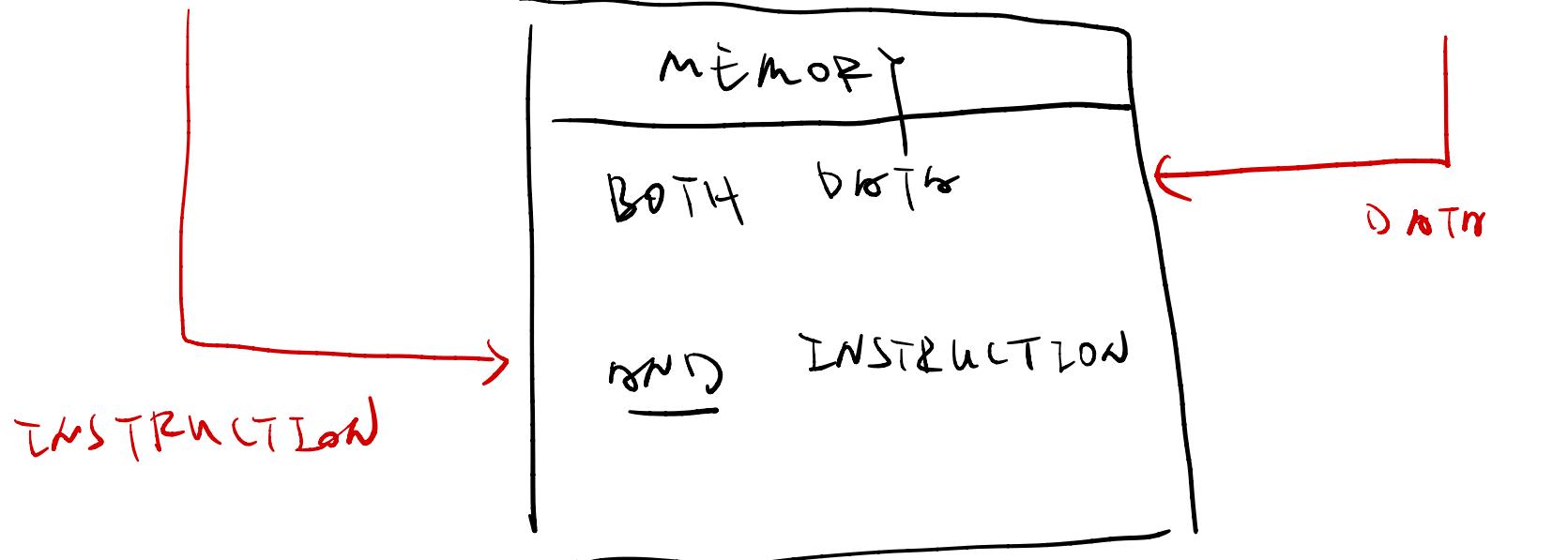
Structural Hazards

- Due to hardware limitations
 - Functional unit cannot handle pipelining, or
 - Insufficient duplication of resources
- Stall instructions until affected unit is available

STRUCTURAL

HAZARDS

IF STAGE



TWO PIPELINE STAGES ACCESSING SAME

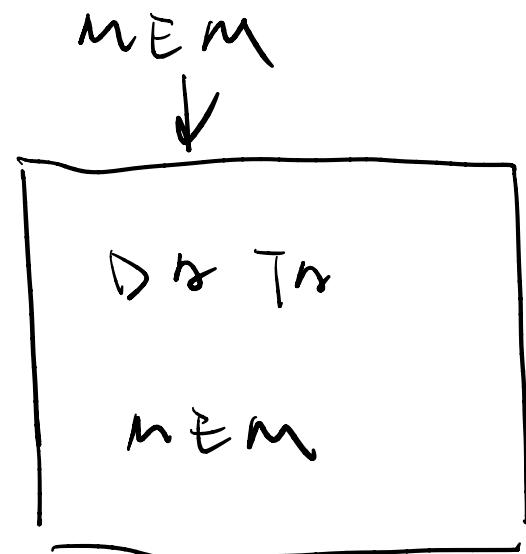
MEMORY UNIT: STRUCTURAL HAZARDS

How does MIPS solve?

"ADD" HWRDW ARE

- SPLIT MEMORY UNITS.

IF : Instruction fetch



OTHER POSSIBLE HAZARDS?

add R_0, R_1, R_2

OPERATION

$$R_0 \leftarrow R_1 + R_2$$

sub R_4, R_0, R_3

$$R_4 \leftarrow R_0 - R_3$$

WHAT IF WE PIPELINE?

CYCLE	1	2	3	4	5	6
add	IF	ID	EX	MEM	WB	
sub		IF	ID READ R_0, R_3	EX	MEM R0 UPDATE HERE	WB

This is a data hazard.

How to address data hazard?

"STALL" sub INSTRUCTION UNTIL
R0 VALUE UPDATED BY add

R0 UPDATED IN CYCLE 5

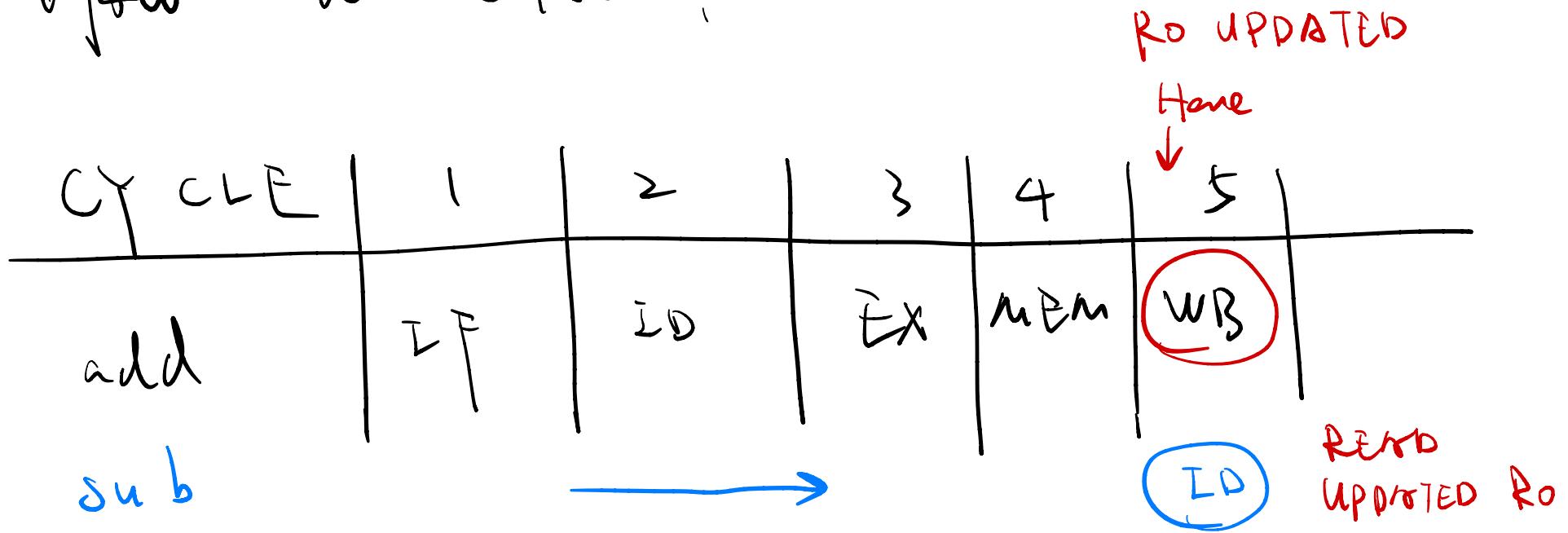
NEED TO "STALL" sub UNTIL ID

occurs in cycle 5

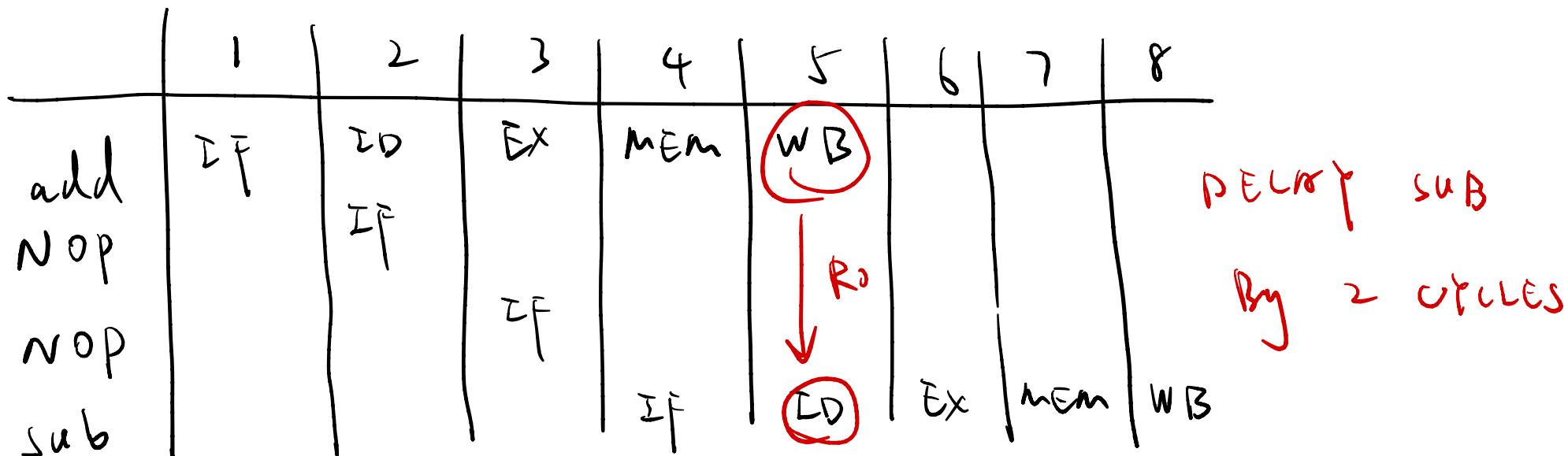
Now, ID occurs in cycle 3 NEED TO "STALL"

sub 2 CYCLES

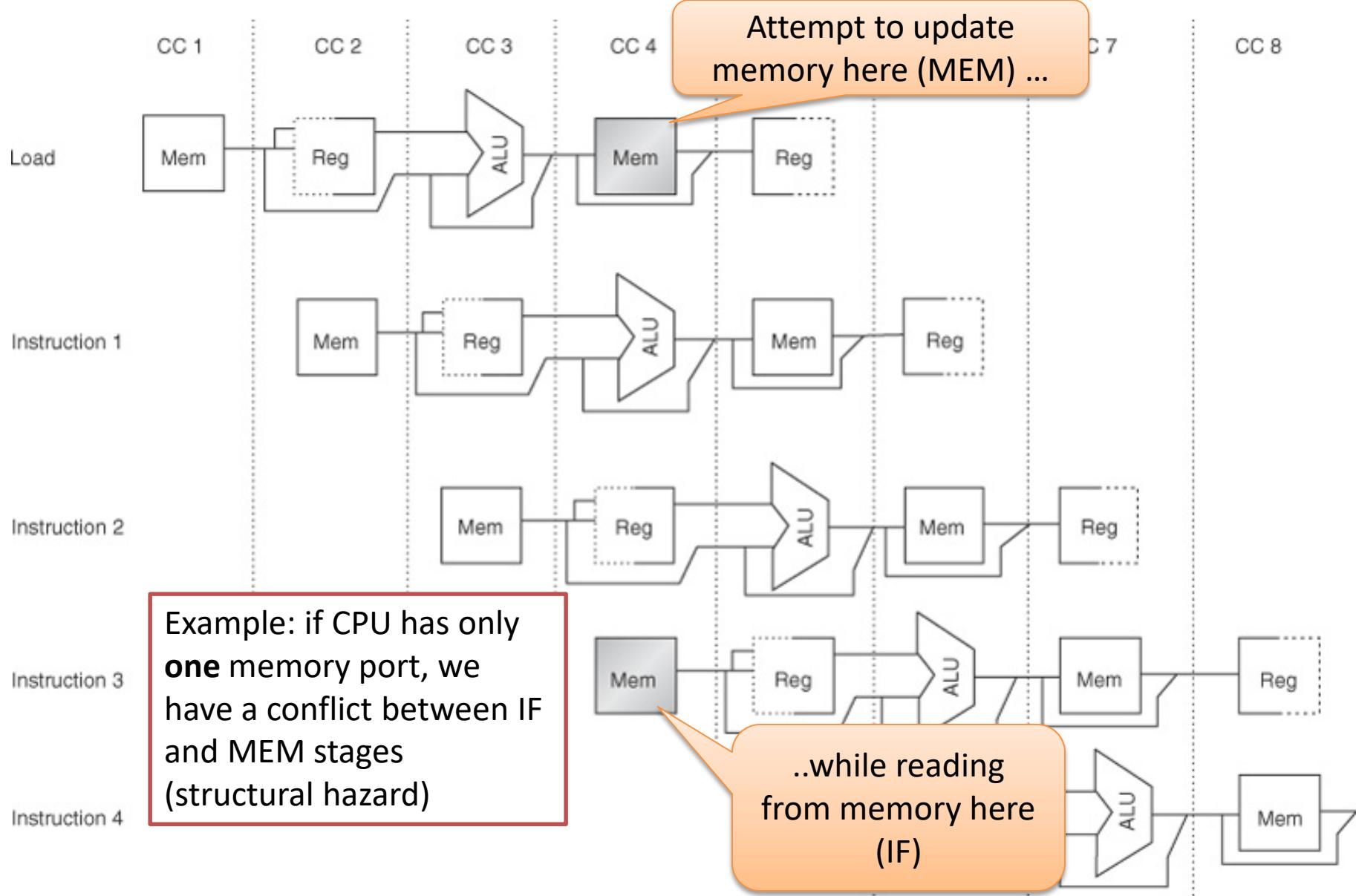
How to stall?



$\text{ADD} \geq \text{"NOP"} \text{ INSTRUCTIONS (No operations)}$



Time (in clock cycles) →



Stall for Structural Hazard

- Add stall to avoid memory access conflict
 - Note: assume that instruction $j+1$ does *not* reference memory

instruction	1	2	3	4	5	6	7	8
j	IF	ID	EX	MEM	WB			
j+1		IF	ID	EX	MEM	WB		
j+2			IF	ID	EX	MEM	WB	
j+3				stall	IF	ID	EX	MEM

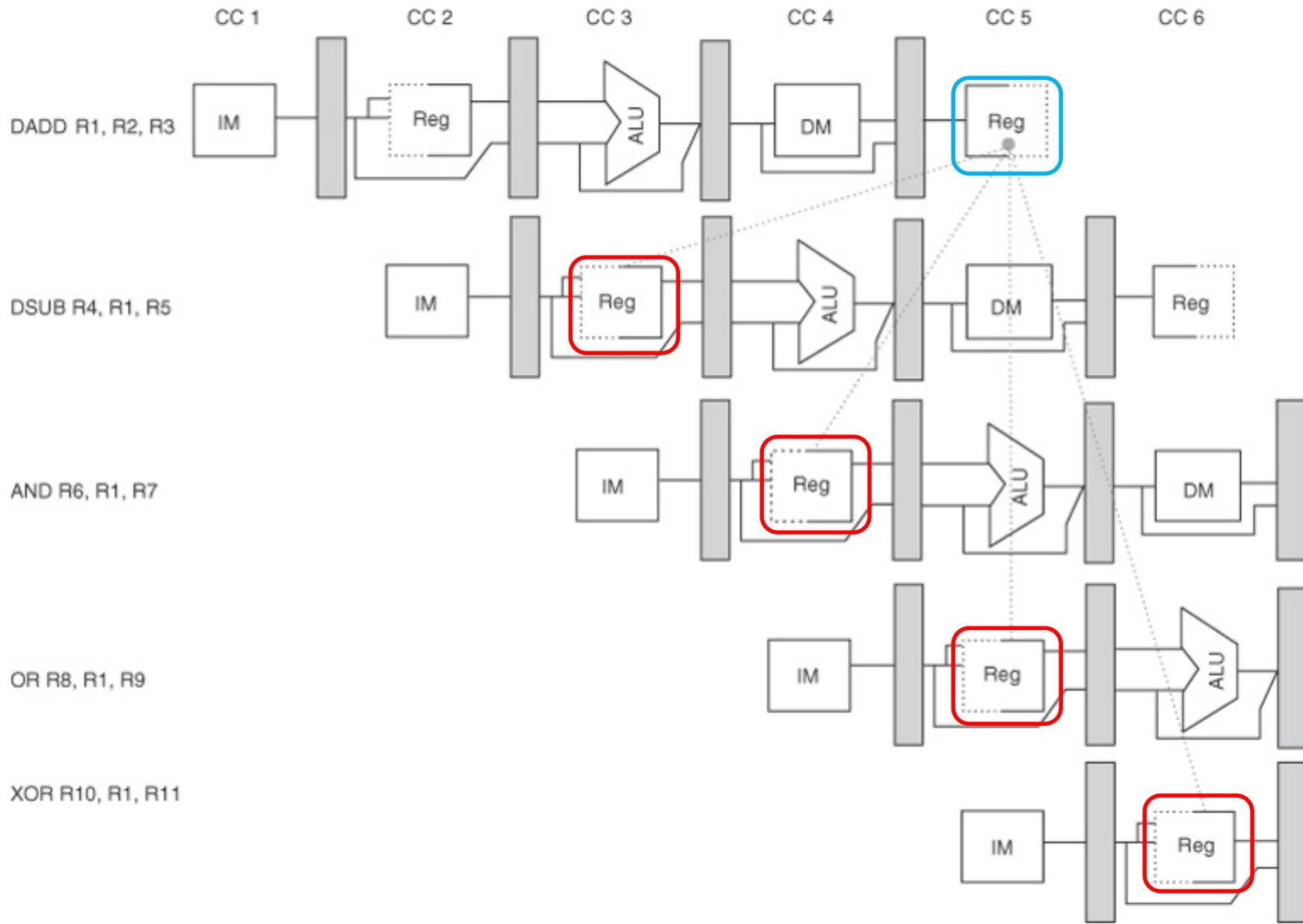
Data Hazards

- | | | |
|----|------|------------|
| 1. | DADD | R1,R2,R3 |
| 2. | DSUB | R4,R1,R5 |
| 3. | AND | R6,R1,R5 |
| 4. | OR | R8,R1,R9 |
| 5. | XOR | R10,R1,R11 |

operand	instructions	dependence
R1	1,2,3,4,5	value not valid for future instructions until DADD updates value

Time (in clock cycles) →

Program execution order (in instructions)



Alternate Representation of Pipeline: Table Method

	1	2	3	4	5	6
DADD R1,R2,R3	IF	ID	EX	MEM	WB	
DSUB R4,R1,R5		IF	ID	EX	MEM	WB
AND R6,R1,R7			IF	ID	EX	MEM
OR R8,R1,R9				IF	ID	EX
XOR R10,R1,R11					IF	ID

Stalls for Data Hazards

	1	2	3	4	5	6
DADD R1,R2,R3	IF	ID	EX	MEM	WB	
DSUB R4,R1,R5		IF	ID	EX	MEM	WB
AND R6,R1,R7			IF	ID	EX	MEM
OR R8,R1,R9				IF	ID	EX
XOR R10,R1,R11					IF	ID

Data hazard for R1: DSUB needs updated value at CC 3, but not updated by DADD until CC 5

Stalls for Data Hazards

	1	2	3	4	5	6
DADD R1,R2,R3	IF	ID	EX	MEM	WB	
NOP		IF				
NOP			IF			
DSUB R4,R1,R5				IF	ID	EX
AND R6,R1,R7					IF	ID
OR R8,R1,R9						IF
XOR R10,R1,R11						

Solution: delay DSUB by 2 clock cycles by adding 2 stalls (NOP)

Stalls for Data Hazards

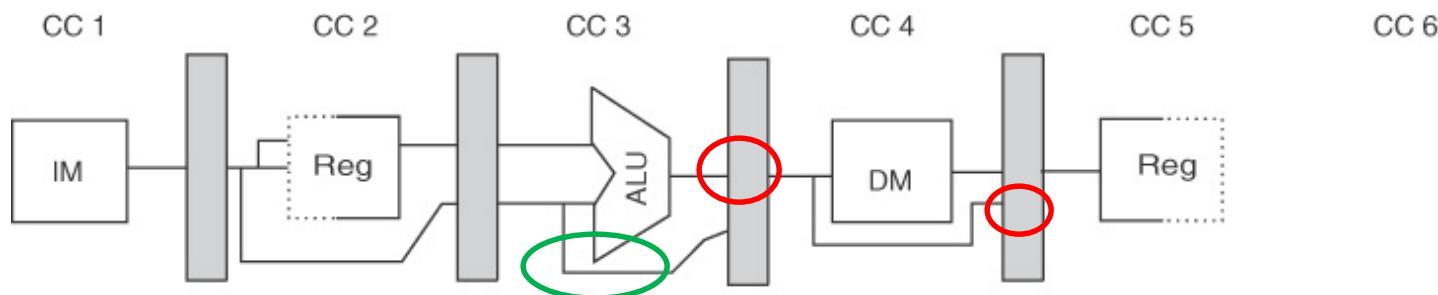
	1	2	3	4	5	6
DADD R1,R2,R3	IF	ID	EX	MEM	WB	
NOP		IF				
NOP			IF			
DSUB R4,R1,R5				IF	ID	EX
AND R6,R1,R7					IF	ID
OR R8,R1,R9						IF
XOR R10,R1,R11						

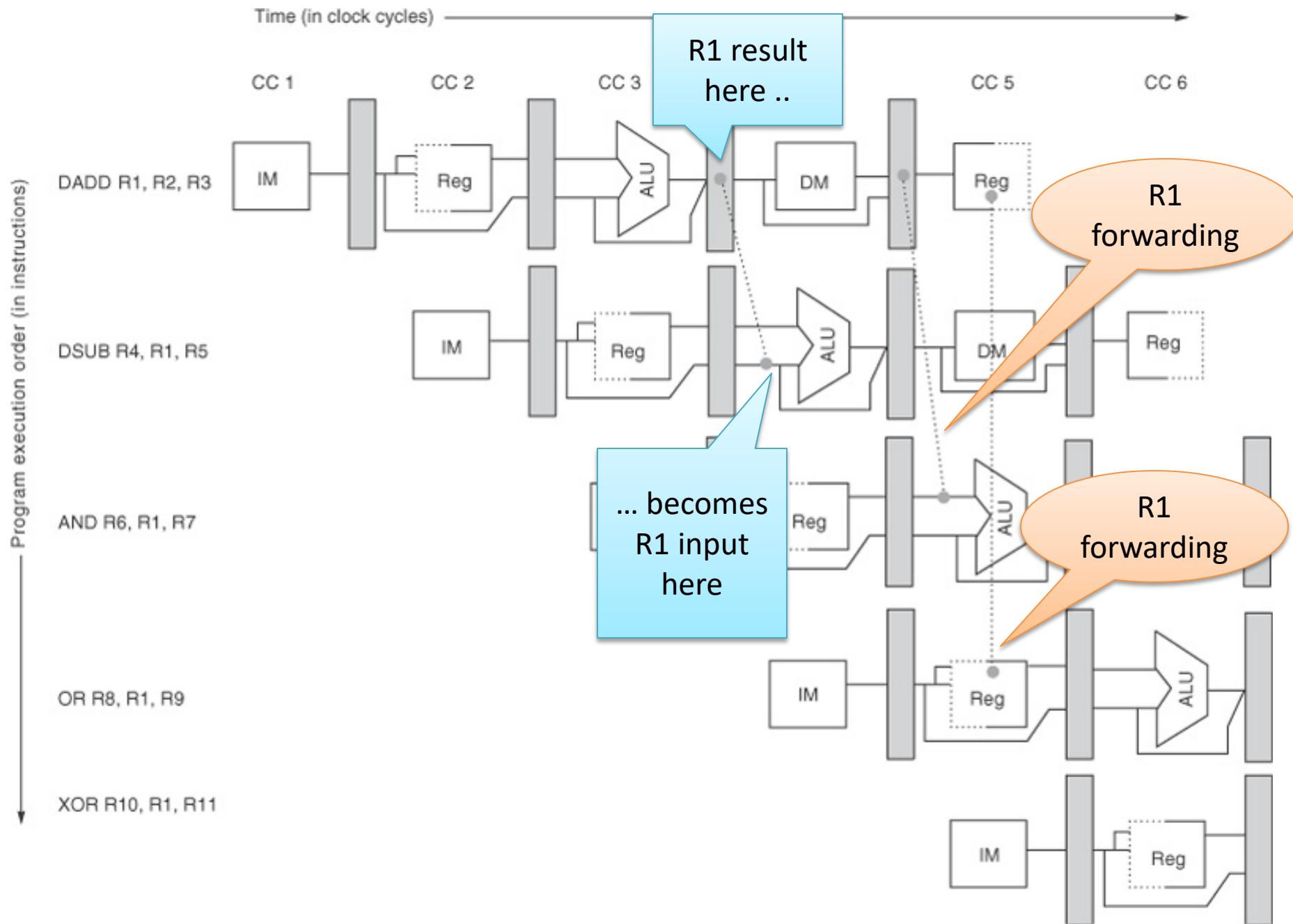
We only need stall the first time.

Any other data hazards? No, R1 will have updated value for following instructions

Forwarding

- Note: output of ALU (EX stage) is stored in **both EX/MEM and MEM/WB pipeline registers**
- Result is also fed back to **ALU inputs**
- We can *forward* this result to the next operation

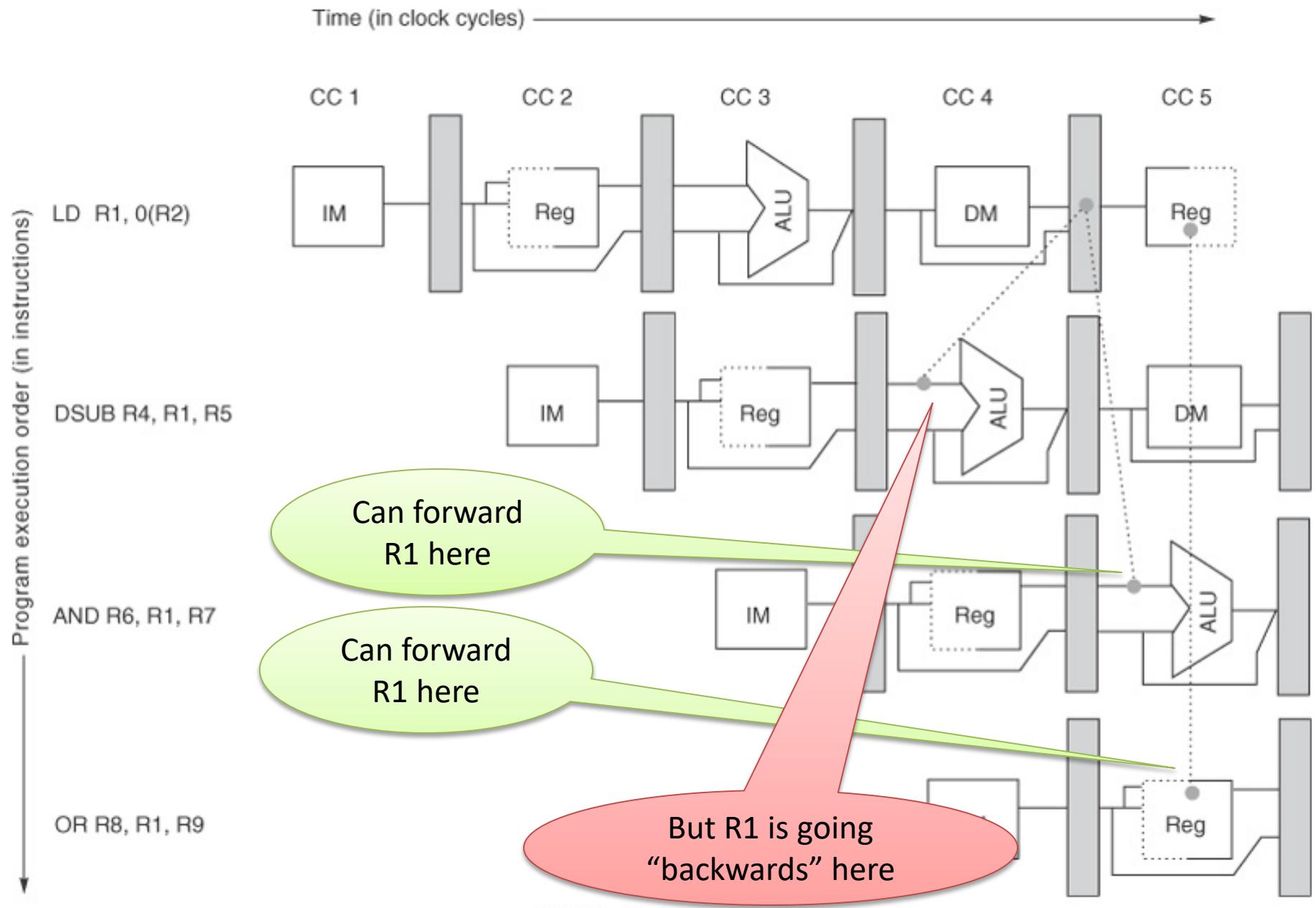




Data Hazards Requiring Stalls

1. LD R1,0(R2)
2. DSUB R4,R1,R5
3. AND R6,R1,R5
4. OR R8,R1,R9

operand	instructions	dependence
R1	1,2,3,4	value not valid for future instructions until LD updates value



Stall for Data Hazard

LD R1,0(R2)	IF	ID	EX	MEM	WB		
DSUB R4,R1,R5		IF	ID	Stall	EX	MEM	WB
AND R6,R1,R7			IF	Stall	ID	EX	MEM
OR R8,R1,R9				Stall	IF	ID	EX

R1 updated
here...

Forwarding doesn't really
buy anything sometimes, and
it makes the design harder,
therefore, stall for data
Hazards plays the easier way to go.

... so add a
stall here

Example C.2-1

We are given the following instruction sequence for our 5-stage MIPS pipeline:

LW R2, 20(R1)

AND R4, R2, R5

OR R8, R2, R6

ADD R9, R4, R2

SUB R1, R6, R7

We will assume that there is no data forwarding or hazard detection hardware in our pipeline. This means that you will need to identify data hazards in the instruction section and insert stalls (NOP instructions) to ensure correct program execution. Determine the data hazards and insert NOP's into the given instruction sequence as needed.

LW	R2, R0 (R1)
AND	R4, R2, R5
OR	R8, R2, R6
AND	R9, R4, R2
SUB	R1, R6, R7

R2
R4
FIRST HANDLE R2
<u>THEN</u> HANDLE R4

STEP 1. DO WE HAVE ANY DATA HAZARDS?

(IF NO, WE DON'T NEED TO INSERT STALLS
— WE ARE DONE)



WE HAVE ②

Control Hazards

- If a program branches, the PC is changed
- Methods to handle?
 1. *Freeze* or *flush* pipeline
 2. Predicted-not-taken
 3. Predicted-taken
 4. Delayed branch

Freeze or flush pipeline

- Perform either operation on pipeline until branch destination is known
 - Freeze: hold instructions
 - Flush: delete instructions
- Simple, but requires refetch of instructions

Predicted-not-taken

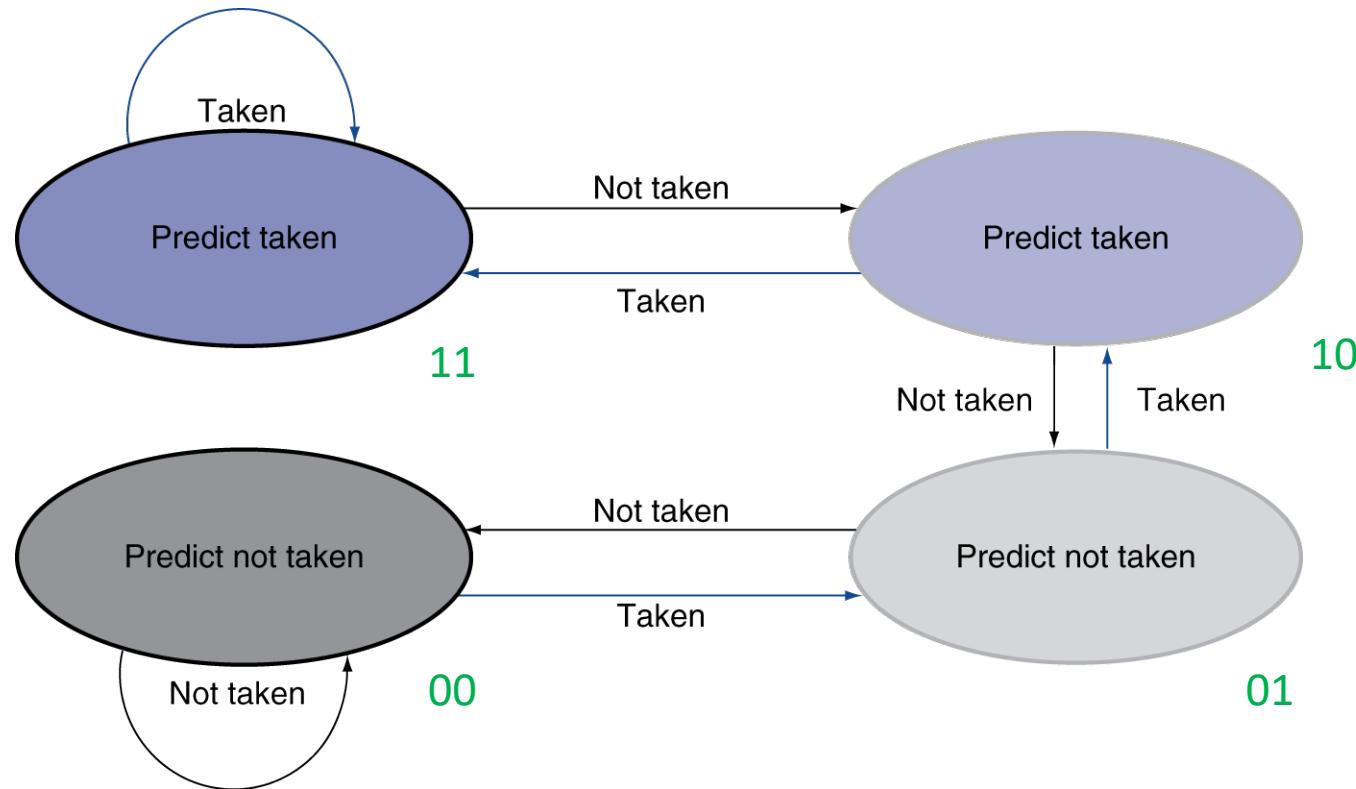
- Assume branches are *not* taken (i.e., no change in PC)
- If branch *is* taken, then refetch last instruction

Predicted-taken

- Assume branches *always* taken (PC changes)
 - Start next instruction from updated PC
- If branch is *not* taken, then refetch next instruction

2-Bit Predictor

- Only change prediction on two successive mispredictions



Example C.2-2

- We have the following repeating pattern of branch outcomes: NT, NT, NT, T, T
 - Where T = branch taken, NT = branch not taken
1. What is the accuracy of **always-taken** predictors for this sequence of branch outcomes?
 2. What is the accuracy of **always-not-taken** predictors for this sequence of branch outcomes?
 3. What is the accuracy of the **two-bit predictor** for this sequence, assuming that the predictor starts off in the bottom left state of the figure (predict not taken)?

Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations

Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations