

# Pipeline Hazards

## Contents of Lecture:

- ❖ Introduction
- ❖ Resource Hazards
- ❖ Data Hazards
- ❖ Control Hazard

## References for This Lecture:

- ✓ William Stallings, Computer Organization and Architecture Designing For Performance, 9<sup>th</sup> Edition, Chapter 14: *Processor Structure and Function*

## Introduction:

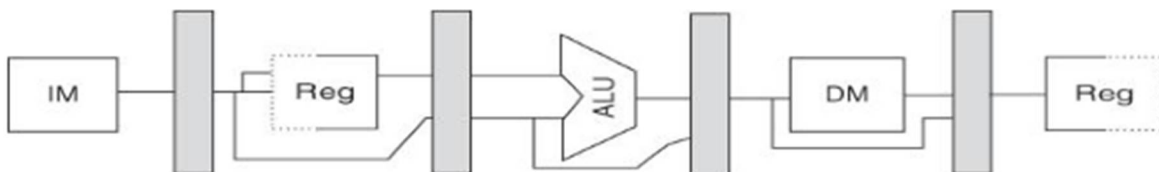
- ❖ A **pipeline hazard** occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution.
- ❖ Such a **pipeline stall** is also referred to as a **pipeline bubble**.
  
- ❖ There are three types of hazards:
  - ✓ Resource hazards.
  - ✓ Data hazards.
  - ✓ Control hazards.



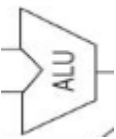


## Resource Hazards:

- ❖ A resource hazard is sometimes referred to as a **structural hazard**
- ❖ A resource hazard occurs when two or more instructions that are already in the pipeline need the **same resource**
- ❖ The result is that the instructions must be executed in serial rather than **parallel** for a **portion** of the pipeline.

## Example of resource hazard:

- ❖ Have the following 5-stage pipeline and the resources that used to perform the stage:

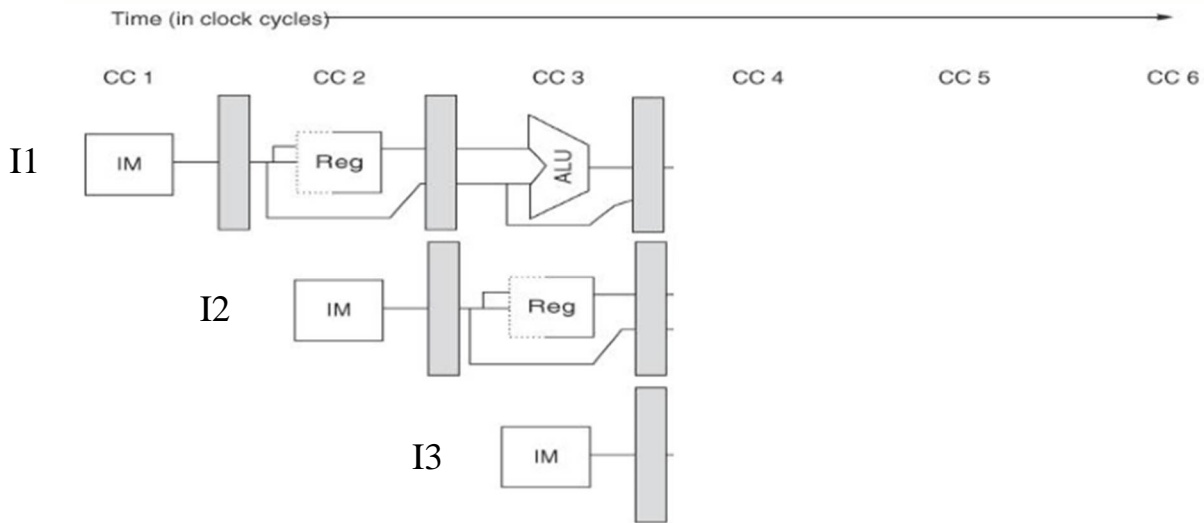


Symbol	Resource	usage
	Memory	<b>IM:</b> Instruction Memory used to Fetch Instructions
	Register	<b>Reg:</b> is divided into two stages: ✓ Decode the Instructions ✓ Read data from Register
	ALU	<b>ALU:</b> used to execute Instructions
	Memory	<b>DM:</b> Data Memory used to fetch data from the memory
	Register	<b>Reg:</b> is divided into two stages: ✓ Read data from Register ✓ Write data to Register

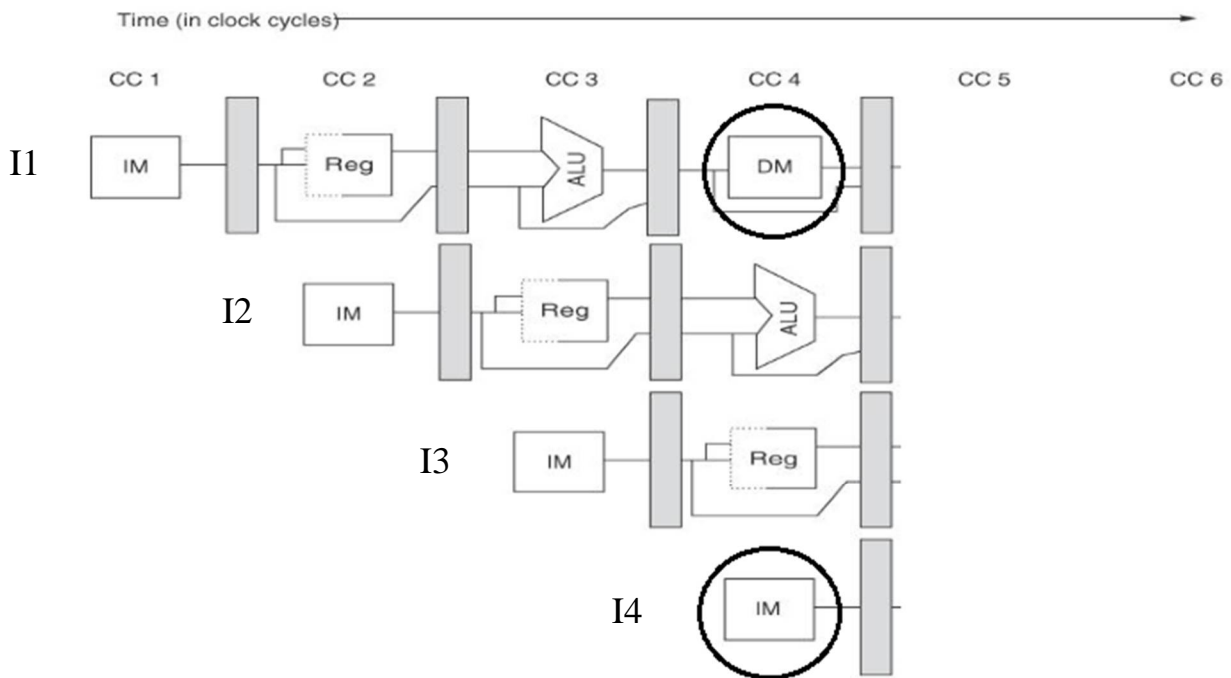
- ❖ In the following example will suppose that there is single memory for both instructions and data.
- ❖ In the **first cycle** (CC1) instruction1 ( I1 ) enter the cycle and start Fetch Instructions from the memory. Shown in the following figure.



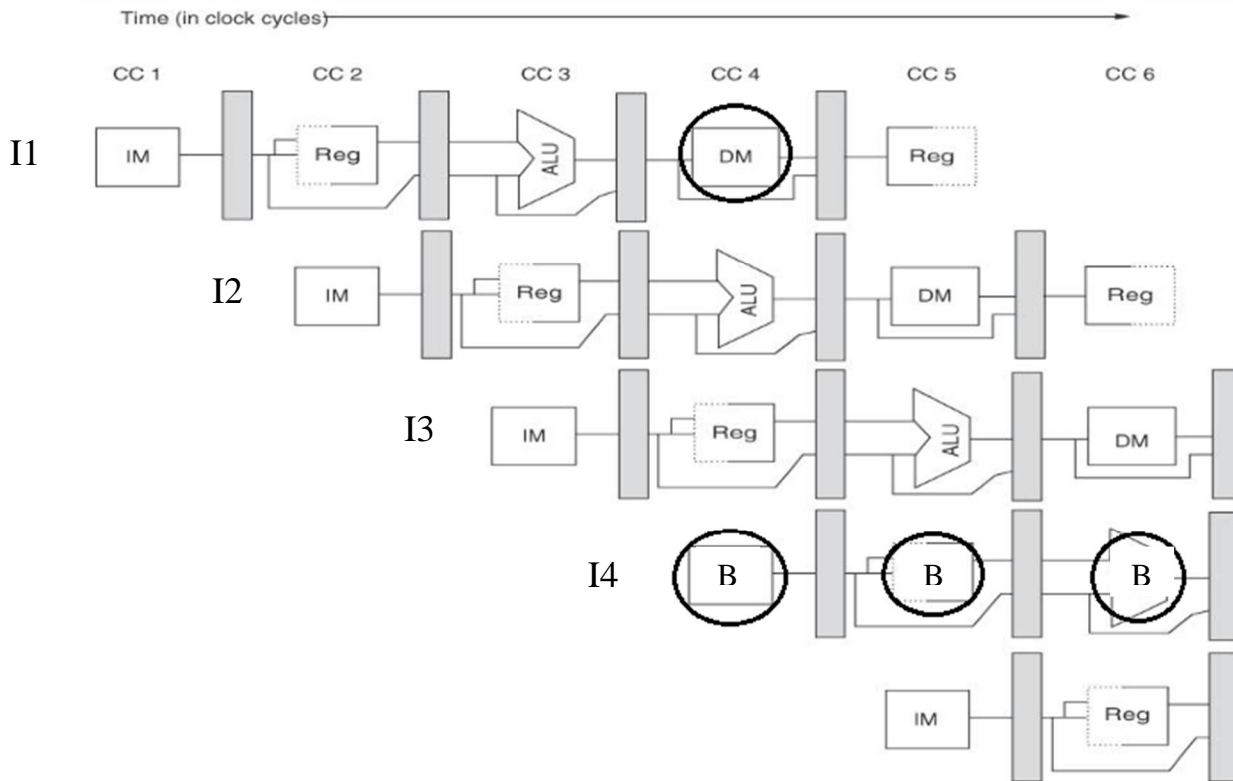
- ❖ At (CC2) instruction1 ( I1 ) enter the next stage and instruction2 ( I2 ) enter the cycle and start Fetch Instructions from the memory.
- ❖ At (CC3) new instruction ( I3 ) enter the cycle and the other two instructions enter the next stages. Shown in the following figure.



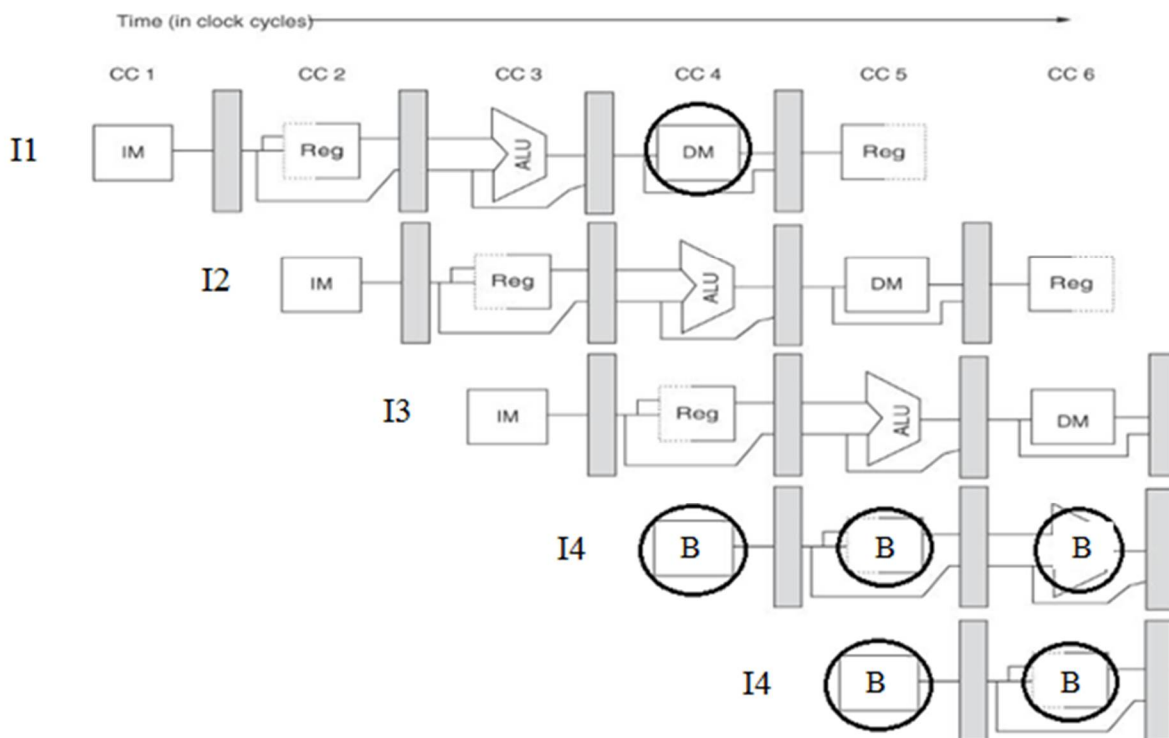
- ❖ When (I4) enter the pipeline at CC4 to Fetch Instruction from memory, it will find that (I1) is used the memory resource to fetch data. So, the same resource will lead to resource hazard.



- ❖ So, in this case control unit must stop one of the two instructions.
- ❖ I4 will be stopped as hazard or bubble, which mean all the following stage of I4 will be stop as hazard or bubble also.



- ❖ At the next cycle (CC5), I4 will try to enter the pipeline, but again the same hazard will happened because I2 at the DM stage, so I4 will be bubble all the stage.
- ❖ At CC6, I4 will try to enter the pipeline, but again the same hazard will happened because I3 at the DM stage, so I4 will be bubble all the stage.
- ❖ At CC7, I4 will enter the pipeline and start the first stage after I1, I2 and I3 will be end all the stages.



## Data Hazards:

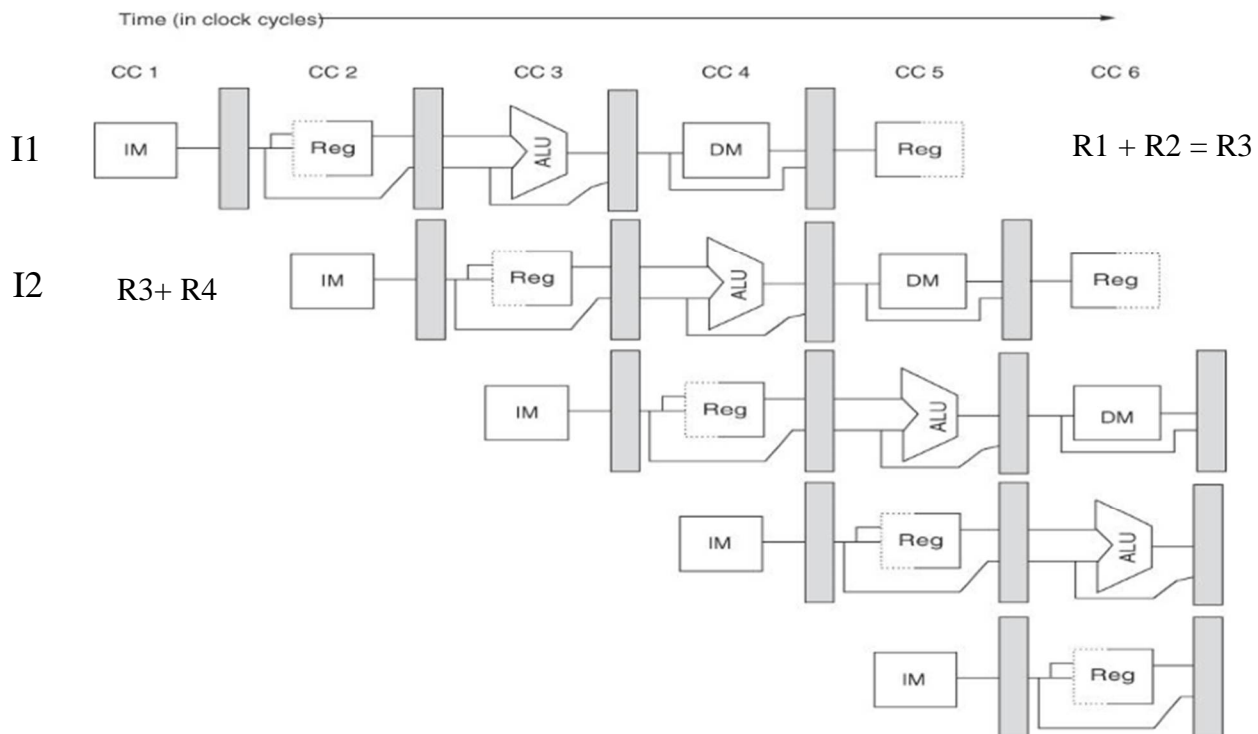
- ❖ A data hazard occurs when there is a conflict in the access of an operand location
- ❖ In other word, Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction

### ❖ Types of Data Hazard:

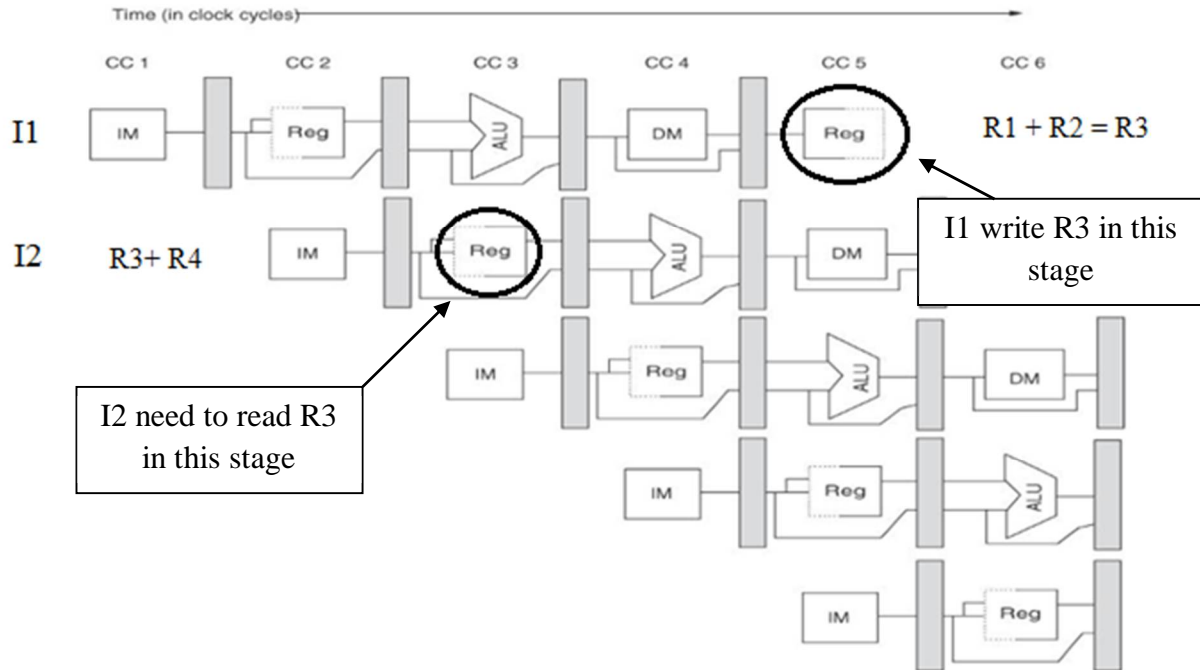
- ✓ **Read after write (RAW), or true dependency**
  - An instruction modifies a register or memory location
  - Succeeding instruction reads data in memory or register location
  - Hazard occurs if the read takes place before write operation is complete
- ✓ **Write after read (WAR), or antidependency**
  - An instruction reads a register or memory location
  - Succeeding instruction writes to the location
  - Hazard occurs if the write operation completes before the read operation takes place
- ✓ **Write after write (WAW), or output dependency**
  - Two instructions both write to the same location
  - Hazard occurs if the write operations take place in the reverse order of the intended sequence

### Example of Data hazard (RAW):

- ❖ At the same 5-stage cycle pipeline, execute the following instructions:
  - ✓ I1:  $R1 + R2 = R3$
  - ✓ I2:  $R3 + R4$



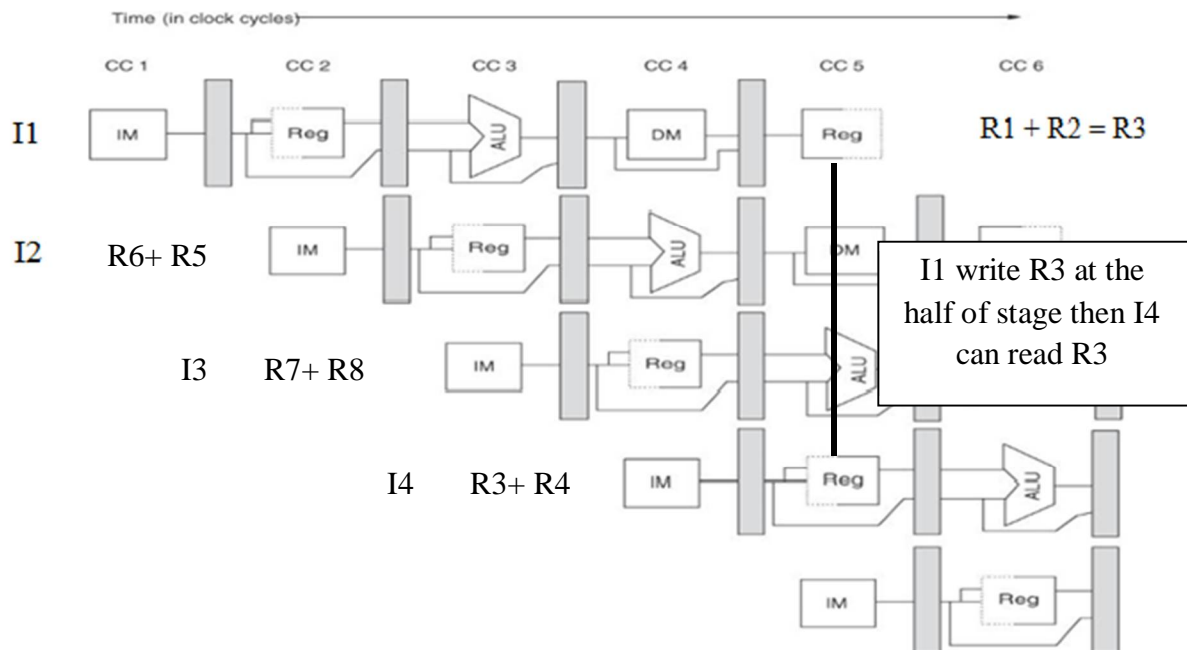
- ❖ In this case the hazard happened as following:
  - ✓ I2 need to read the value of R3 in the CC3, but I1 will write the value of R3 in the CC5.
  - ✓ So. If I2 read R3 , its will read an old value or wrong value.
  - ✓ I2 will wait until I1 write the value of R3.



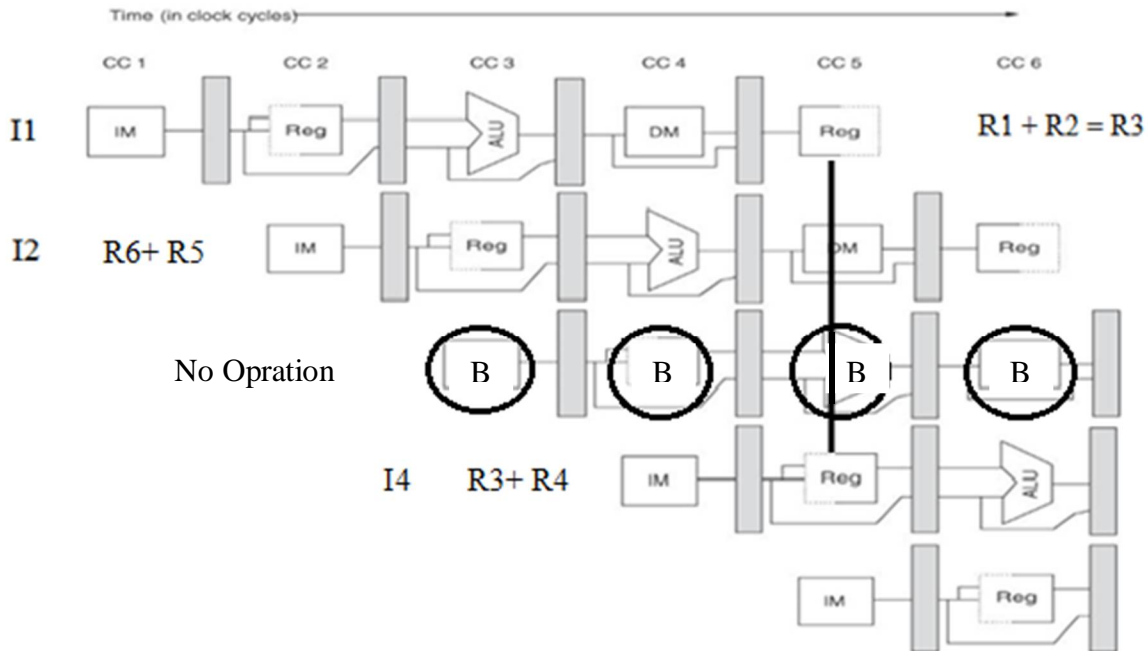
**In this case we have two way to deal with this hazard:**

**First:**

- ❖ I2 will wait until I1 write the value of R3, at this time CPU will execute other instruction, as following.

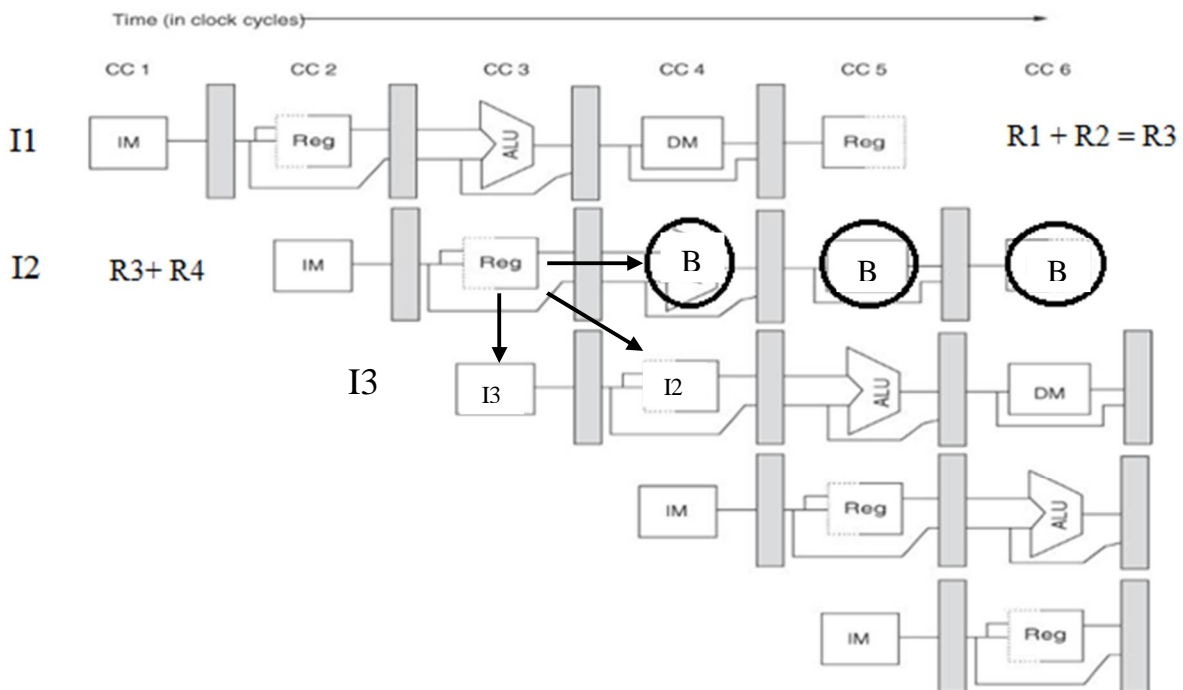


❖ If there is no other operation that will lead to bubble, as following.



**Second:**

❖ Enter new instruction to the pipeline and let I2 execute with the new instruction, as following.



❖ In this case I2 send three signals by the decoder in decode stage:

- ✓ First signal: send to the following stage to stop (bubble)
- ✓ Second signal: send to next instruction I3 to stay at the first stage (IM) because I2 will execute with I3.
- ✓ Third signal: send to Reg stage in I3 (the stage I2 will start with it)

## **Control Hazard:**

- ❖ Also known as a branch hazard
- ❖ Occurs when the pipeline makes the wrong decision on a branch prediction
- ❖ Brings instructions into the pipeline that must subsequently be discarded
- ❖ Dealing with Branches:
  - ✓ Multiple streams
  - ✓ Prefetch branch target
  - ✓ Loop buffer
  - ✓ Branch prediction
  - ✓ Delayed branch

## **Multiple Streams:**

- ❖ A simple pipeline suffers a penalty for a branch instruction because it must choose one of two instructions to fetch next and may make the wrong choice
- ❖ A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams
- ❖ **Drawbacks (disadvantage):**
  - ✓ With multiple pipelines there are contention delays for access to the registers and to memory
  - ✓ Additional branch instructions may enter the pipeline before the original branch decision is resolved

## **Prefetch Branch Target:**

- ❖ When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch
- ❖ Target is then saved until the branch instruction is executed
- ❖ If the branch is taken, the target has already been prefetched

## **Branch Prediction:**

- ❖ Various techniques can be used to predict whether a branch will be taken:
- ❖ **Static approaches**
  - ✓ They do not depend on the execution history up to the time of the conditional branch instruction
    1. Predict never taken
    2. Predict always taken
    3. Predict by opcode
- ❖ **Dynamic approaches**
  - ✓ They depend on the execution history
    1. Taken/not taken switch
    2. Branch history table



### **Static Branch Prediction:**

- ❖ Predict never taken
  - ✓ Assume that jump will not happen
  - ✓ Always fetch next instruction
- ❖ Predict always taken
  - ✓ Assume that jump will happen
  - ✓ Always fetch target instruction
- ❖ Predict by Opcode
  - ✓ Some instructions are more likely to result in a jump than others
  - ✓ conditional branches are taken more than 50%

### **Dynamic Branch Prediction:**

- ❖ Taken/Not taken switch
  - ✓ Based on previous history
  - ✓ Good for loops

### **Delayed Branch:**

- ❖ Delayed Branch
  - ✓ Do not take jump until you have to Rearrange instructions