ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Pipelined Microprocessor

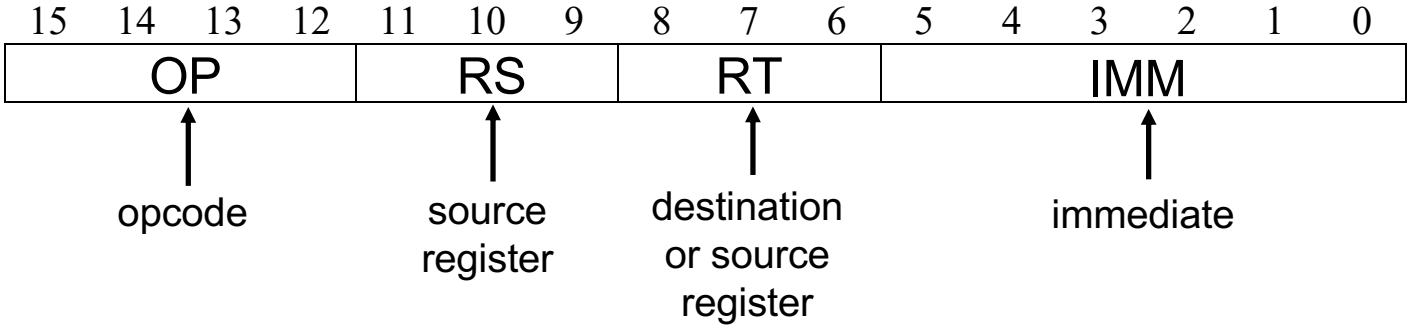
Cornell University

Announcements

- Lab 4 released
- First batch of quiz scores released
- Solutions to HW5 & HW6 released
- Recording of the Prelim 2 review session available in Ed post #173

Review: Load and Store Instructions

Immediate (*I-Type*): ALU operations with immediate, load/store, branch

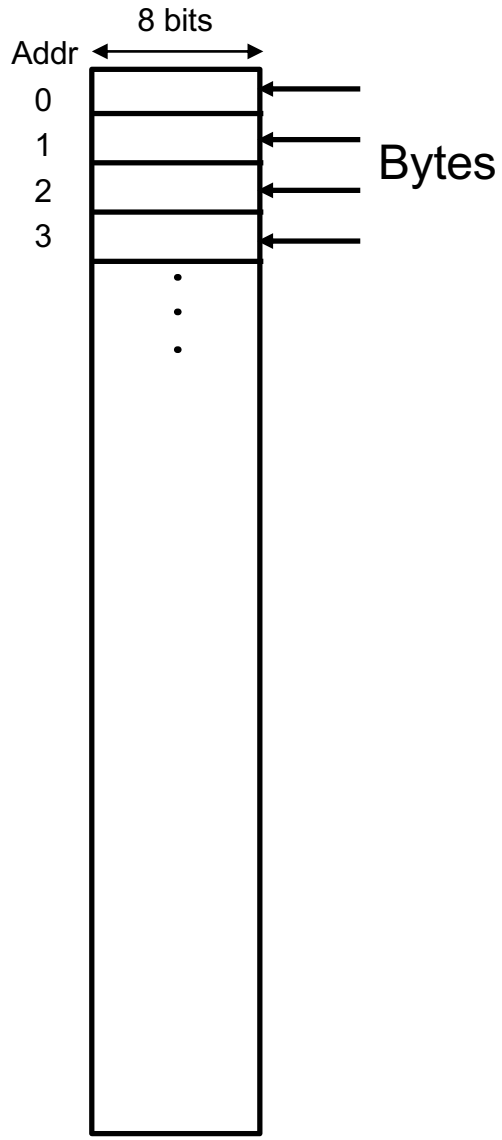


Load and Store Instructions

OP	Instruction	
0001	LW RT, IMM(RS)	(Load Word)
0010	LB RT, IMM(RS)	(Load Byte)
0011	SW RT, IMM(RS)	(Store Word)
0100	SB RT, IMM(RS)	(Store Byte)

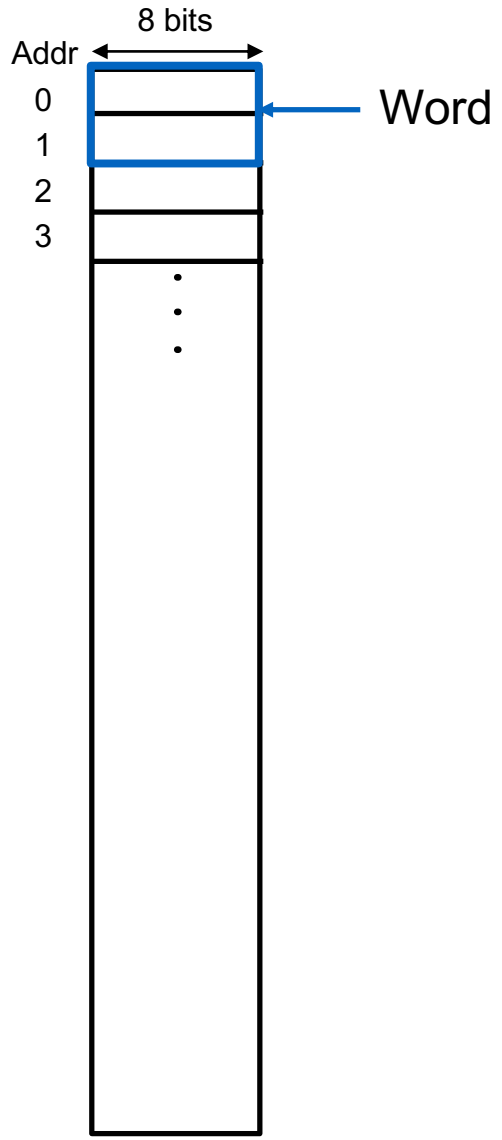
- **Important assumptions in our ISA**
  - **Main memory is byte addressable**
  - **2 bytes per word**

Main Memory Organization



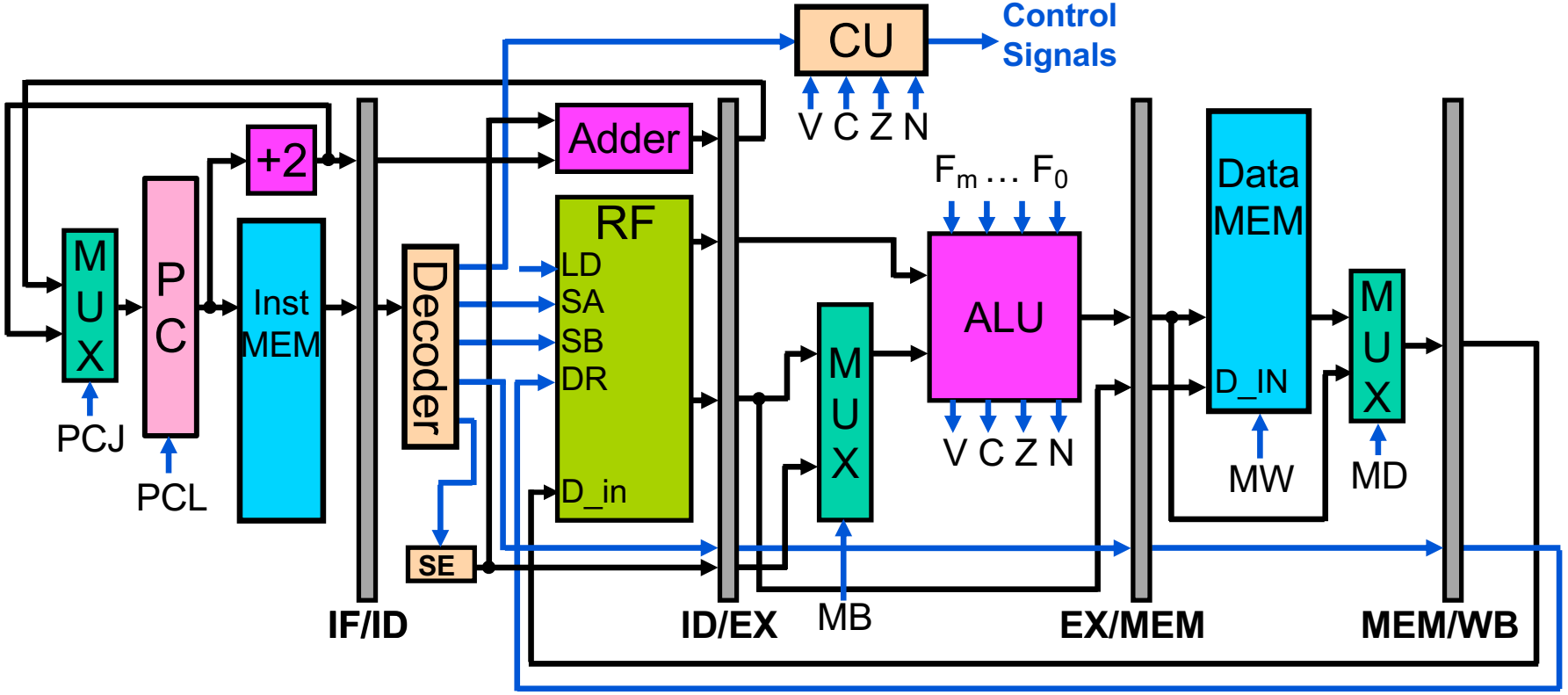
- **Assumptions**
  - **Byte addressable**

Main Memory Organization



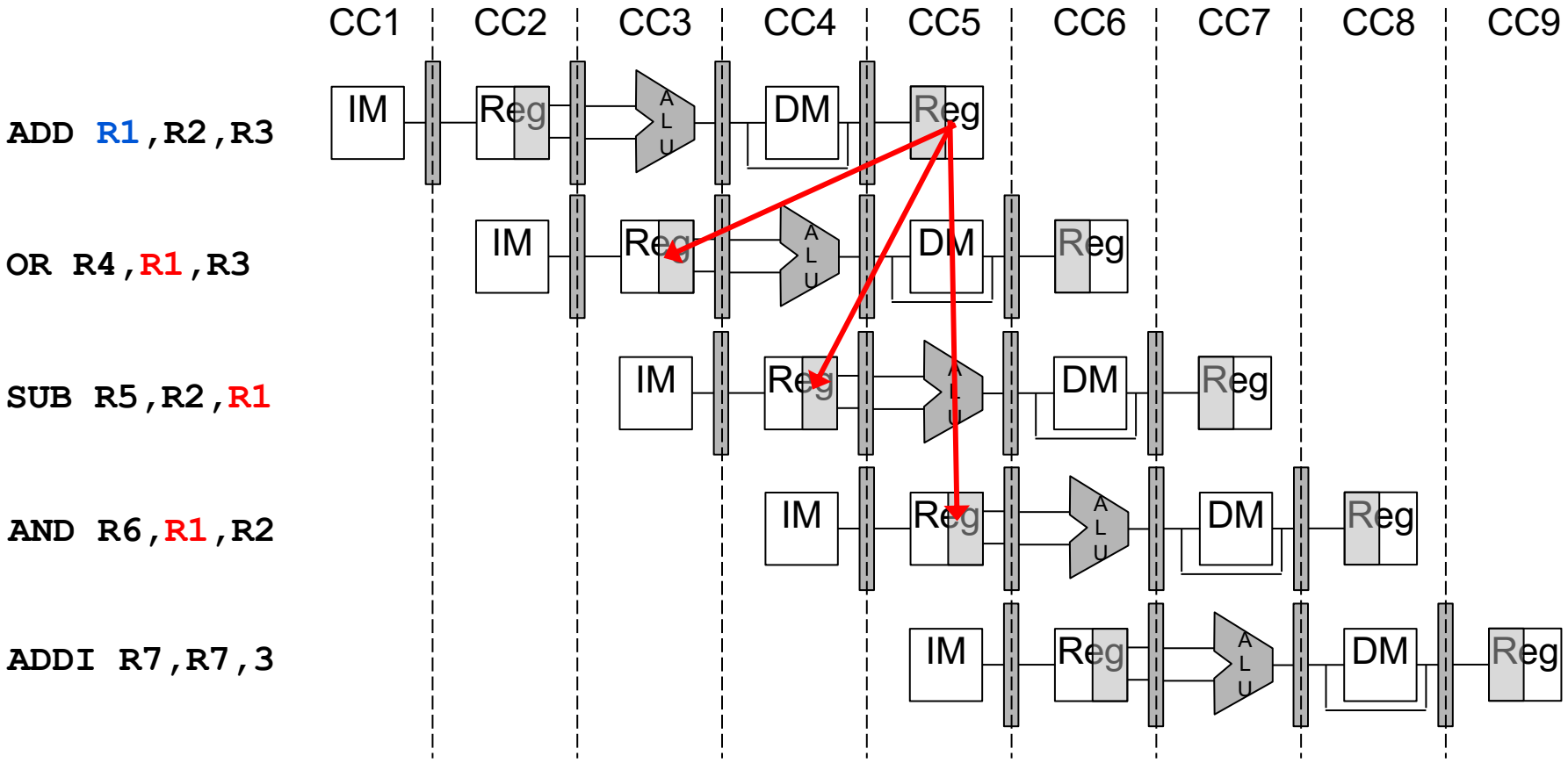
- **Assumptions**
  - **Byte addressable**
  - **2 bytes per word**

Review: Pipelined Microprocessor



- Faster clock frequency than single cycle processor
- In the ideal case, ~1 instruction completed every cycle, where all pipeline stages are busy all the time

Review: What About This Sequence?

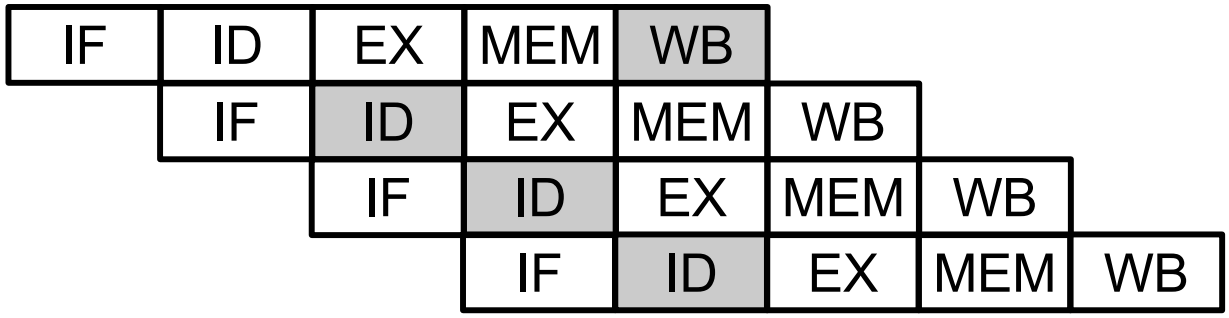


The OR, SUB, and AND instructions are *data dependent* on the ADD instruction

Data Hazard

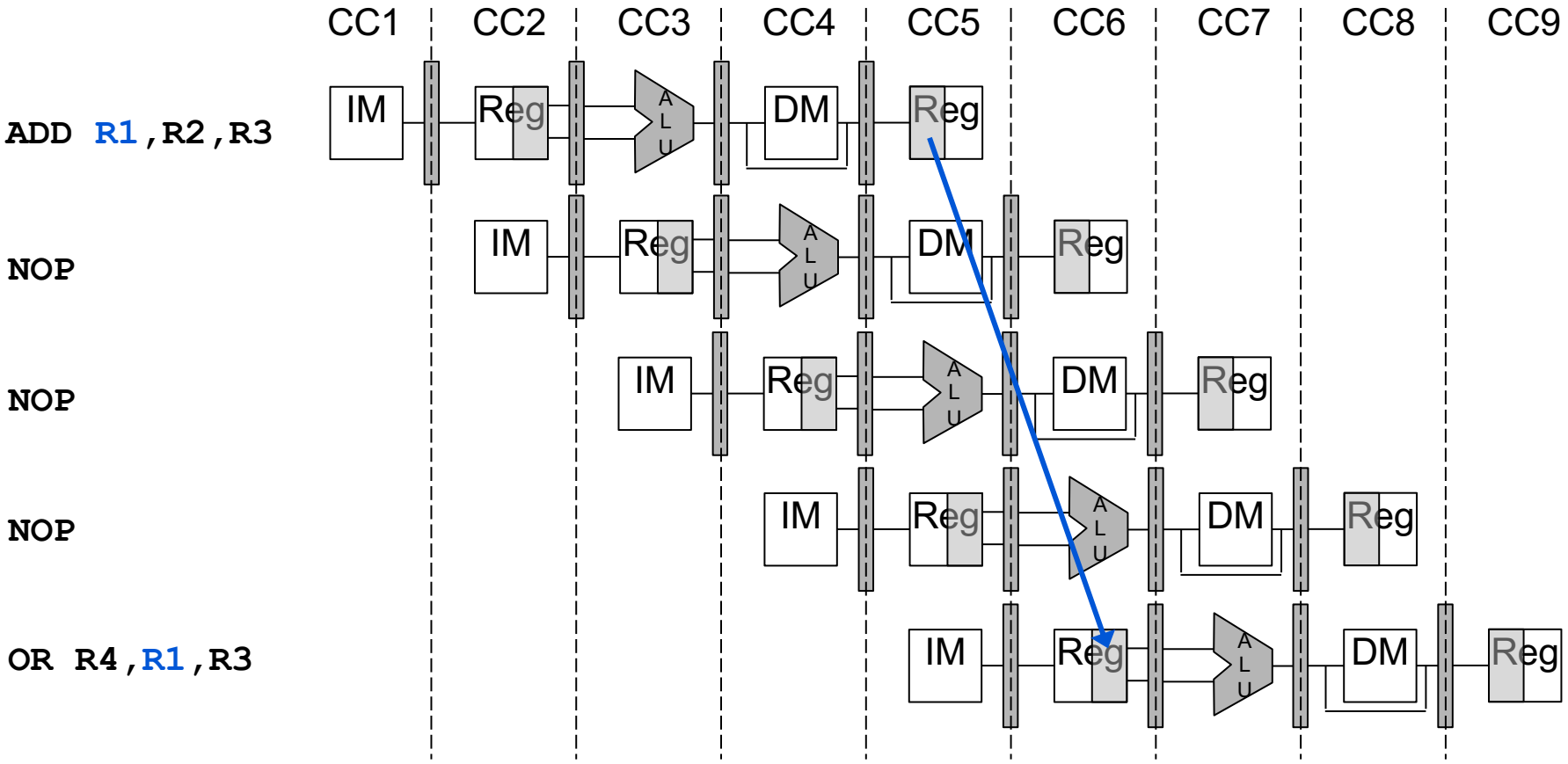
- Occurs when a register is read before the write back of a value to that register

ADD R1,R2,R3
OR R4,R1,R3
SUB R5,R2,R1
AND R6,R1,R2



- What should happen
  - The 1st instruction calculates a new value for R1
  - The 2nd, 3rd, and 4th instructions use this new value
- What actually happens
  - The 2nd, 3rd, and 4th instructions read the old value of R1
  - The first instruction then writes the new value into R1

Solution 1: Software (Compiler) Inserts NOPs



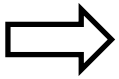
NOP (No Operation) does nothing when executed, which is commonly used as a dummy padding instruction to delay execution in a programs.

Exercise: Data Hazards

- Identify all data hazards in the following instruction sequences by **circling each source register** that is read before the updated value is written back

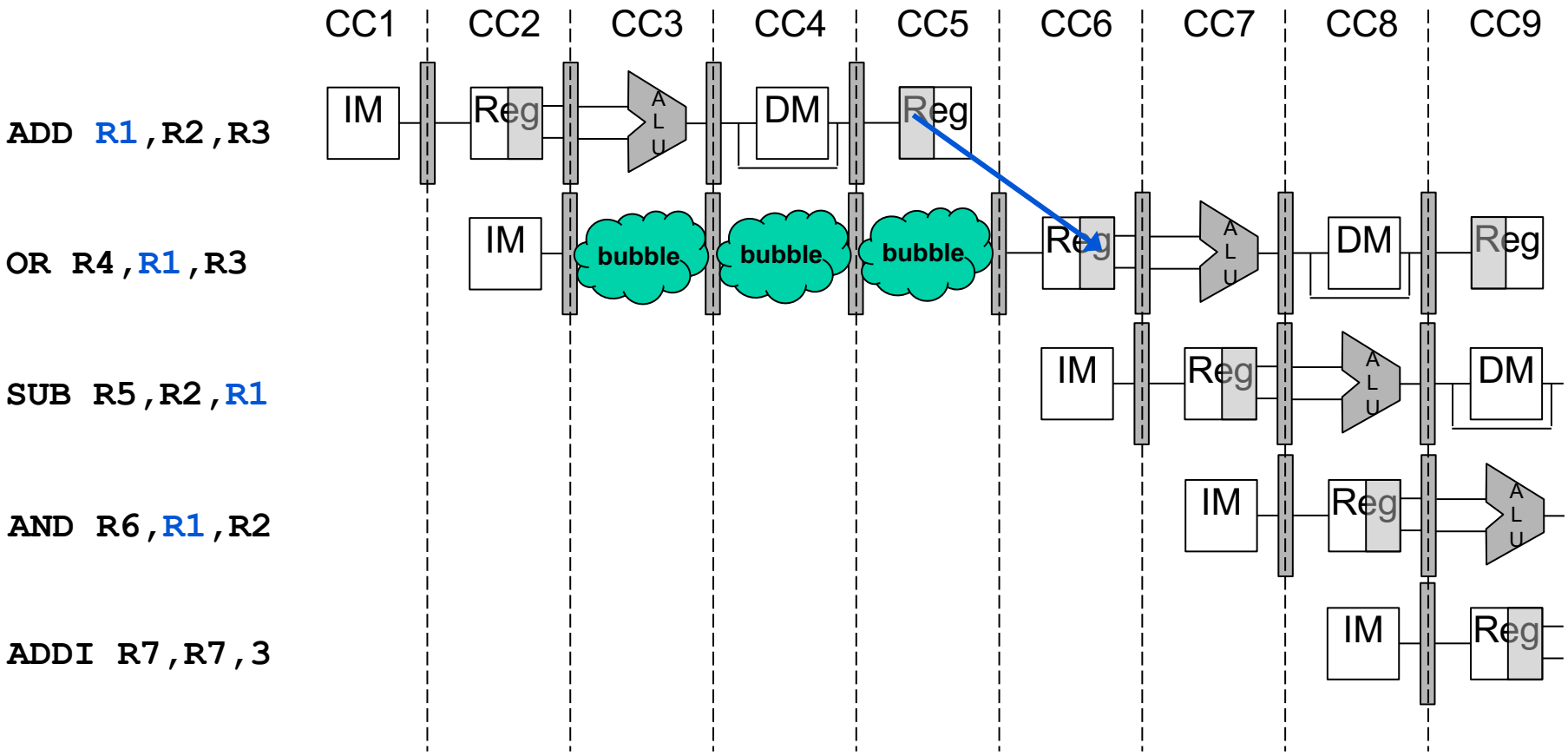
- How many NOPs need to be inserted to avoid data hazards in the following instruction sequence

```
SUB R4, R2, R3
ADD R1, R1, R0
ADDI R2, R4, 1
OR R5, R3, R4
```



```
SUB R4, R2, R3
NOP
NOP
ADD R1, R1, R0
ADDI R2, R4, 1
OR R5, R3, R4
```

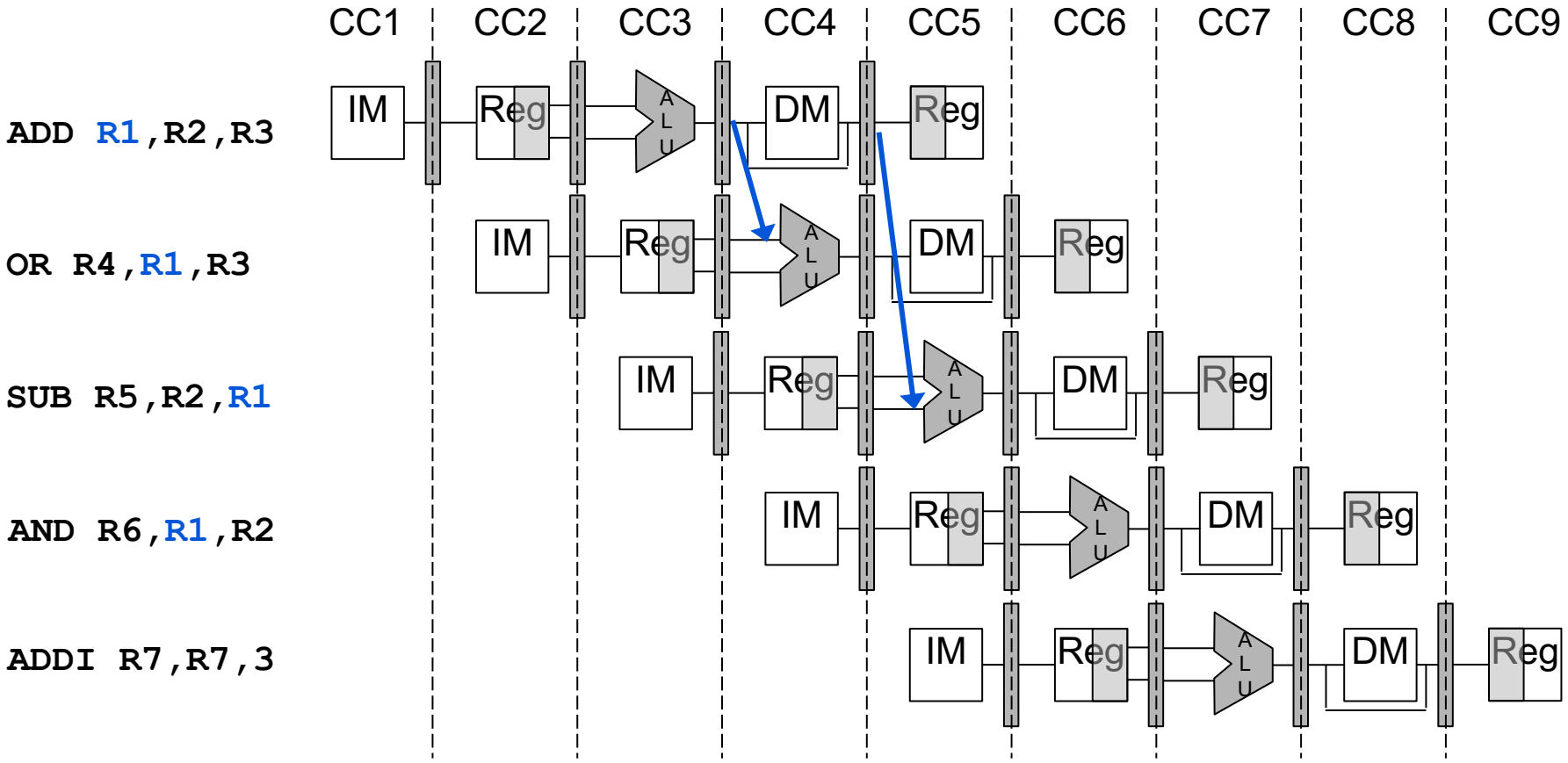
Solution 2: Hardware (HW) *Stalls* the Pipeline



Bubbles are inserted by HW to stall the pipeline for three cycles[1]

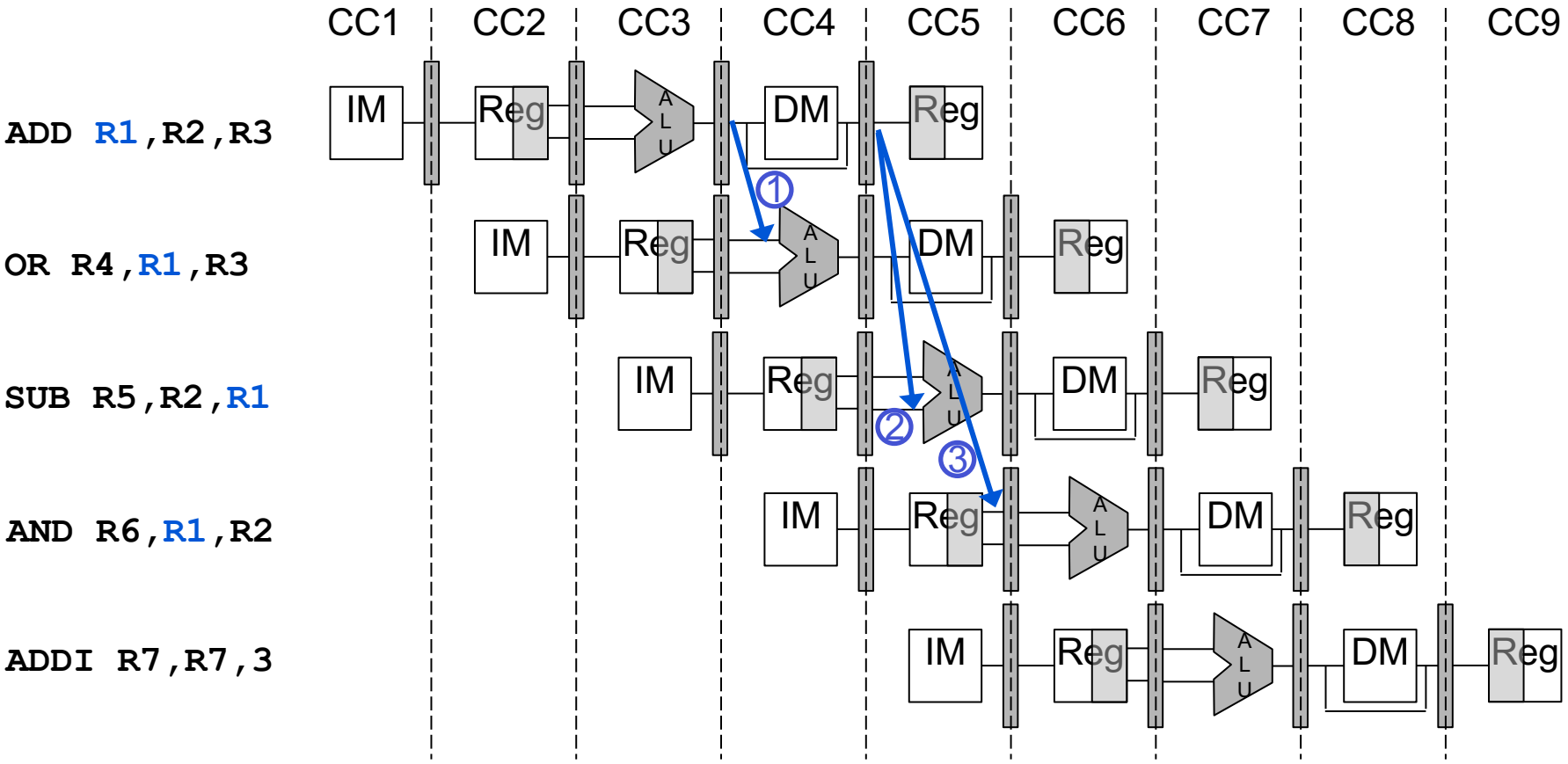
[1] A bubble is basically a NOP instruction inserted by the processor hardware during program execution, in contrast to a NOP generated by the compiler prior to program execution. Just like other instructions, a bubble is pushed through the pipeline from the stage where it is inserted.

Solution 3: HW Forwarding (Bypassing)



How to forward to the AND instruction?

Solution 3: HW Forwarding (Bypassing)

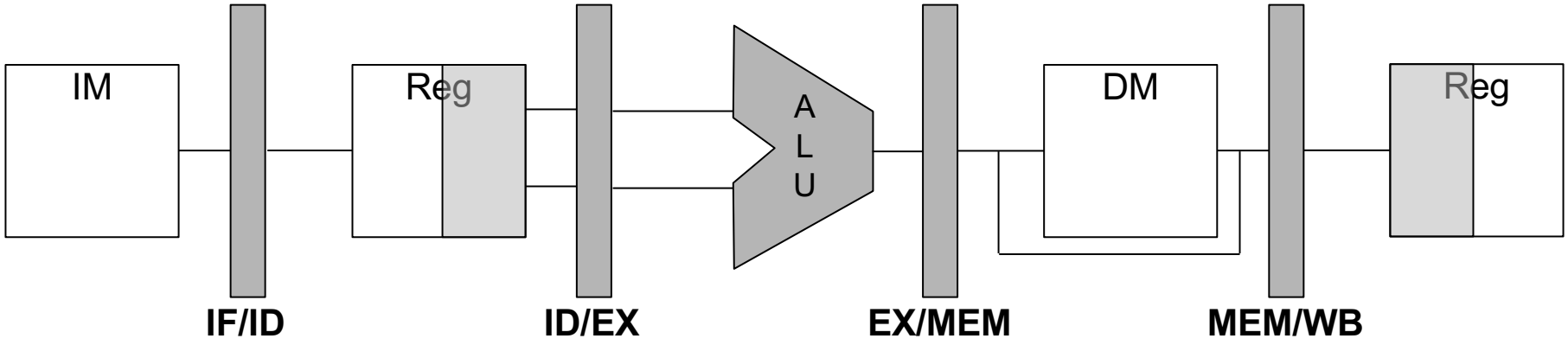


① MEM→EX for ADD to OR

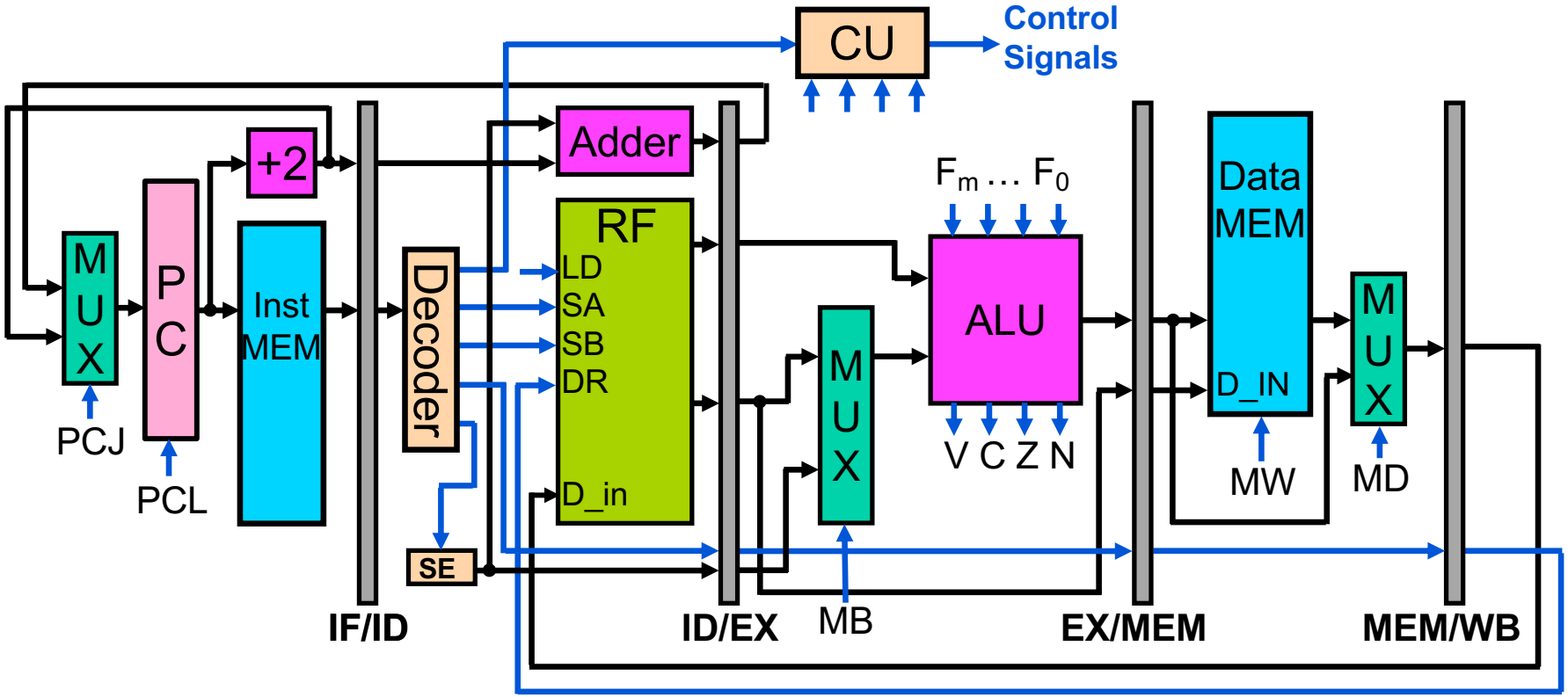
② WB→EX for ADD to SUB

③ WB→ID for ADD to AND

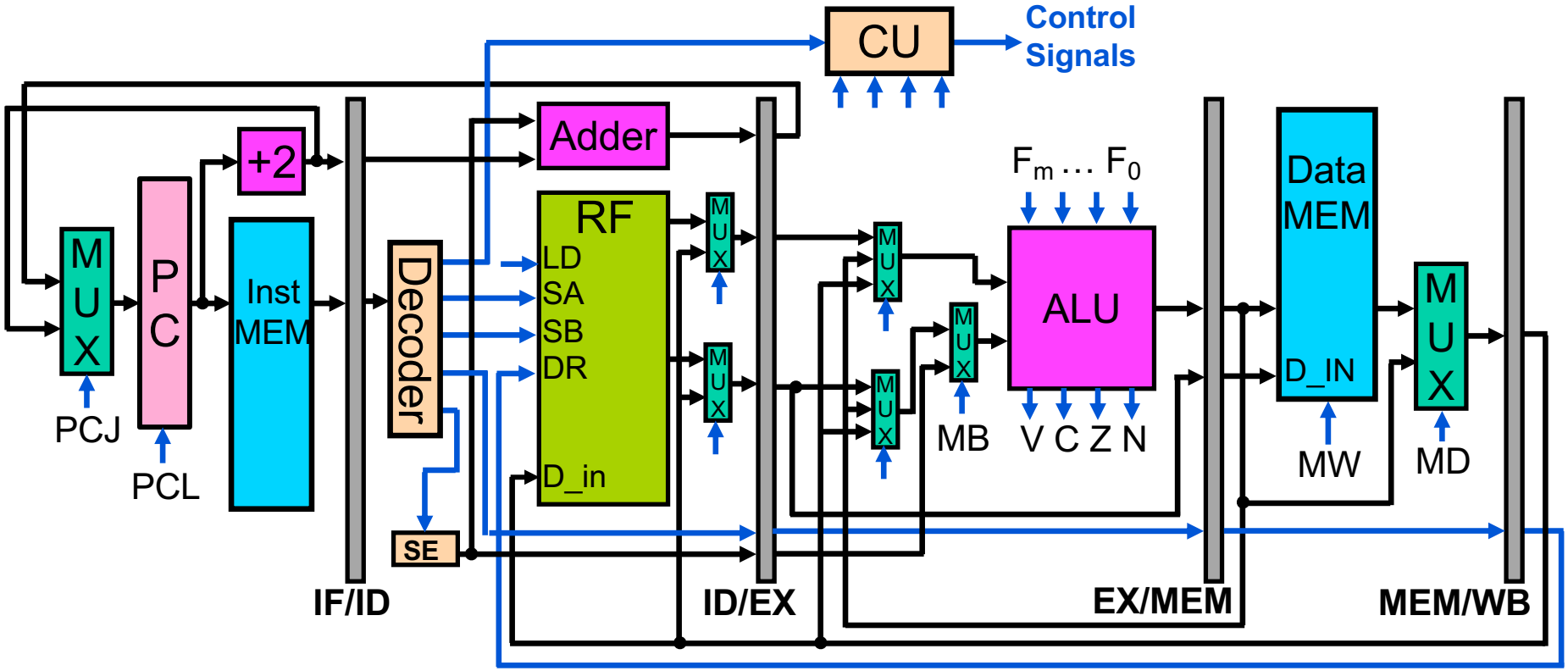
Pipeline Modifications for Forwarding?



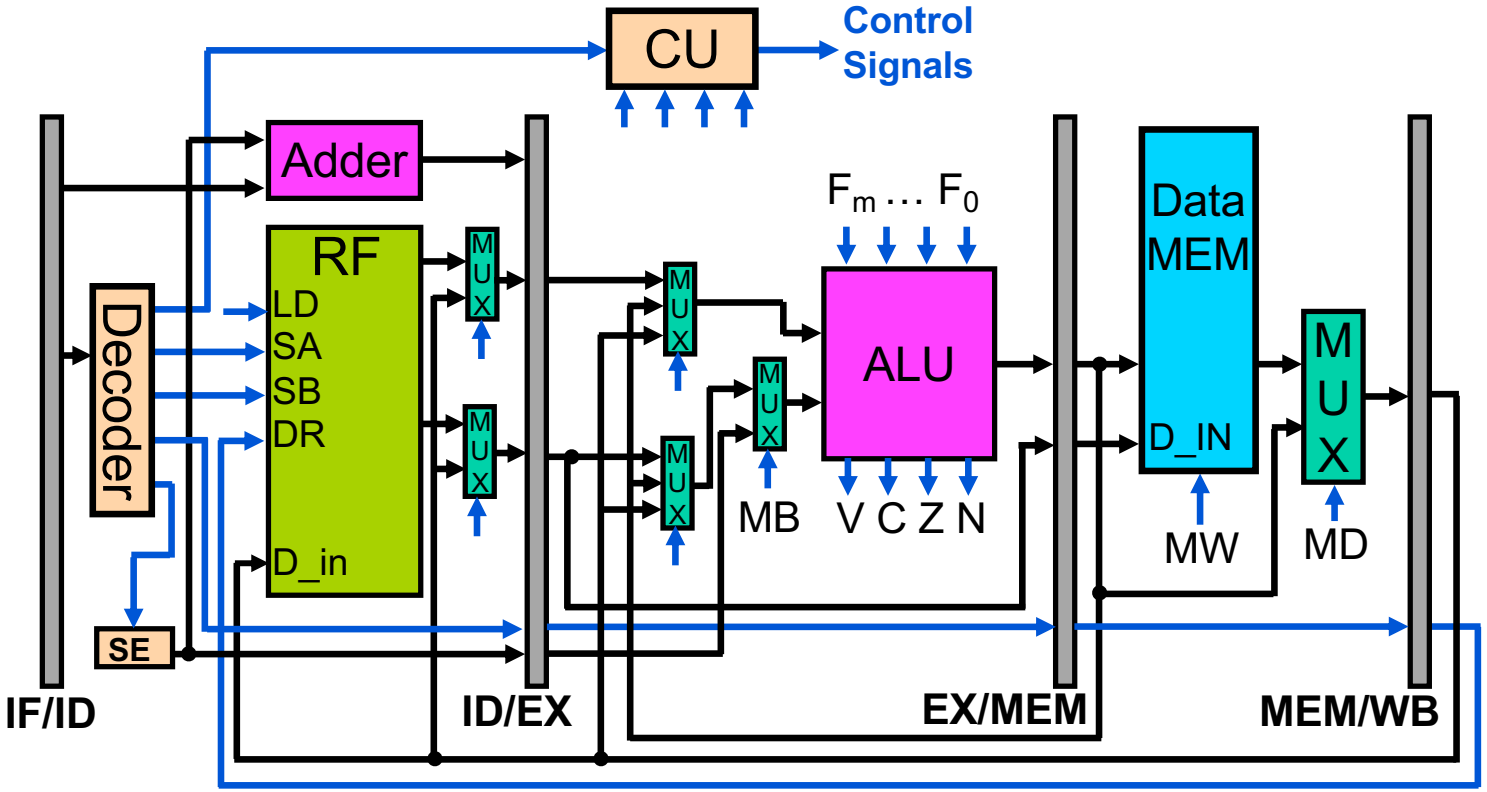
Pipelined Microprocessor w/o Forwarding



Pipelined Processor with Forwarding

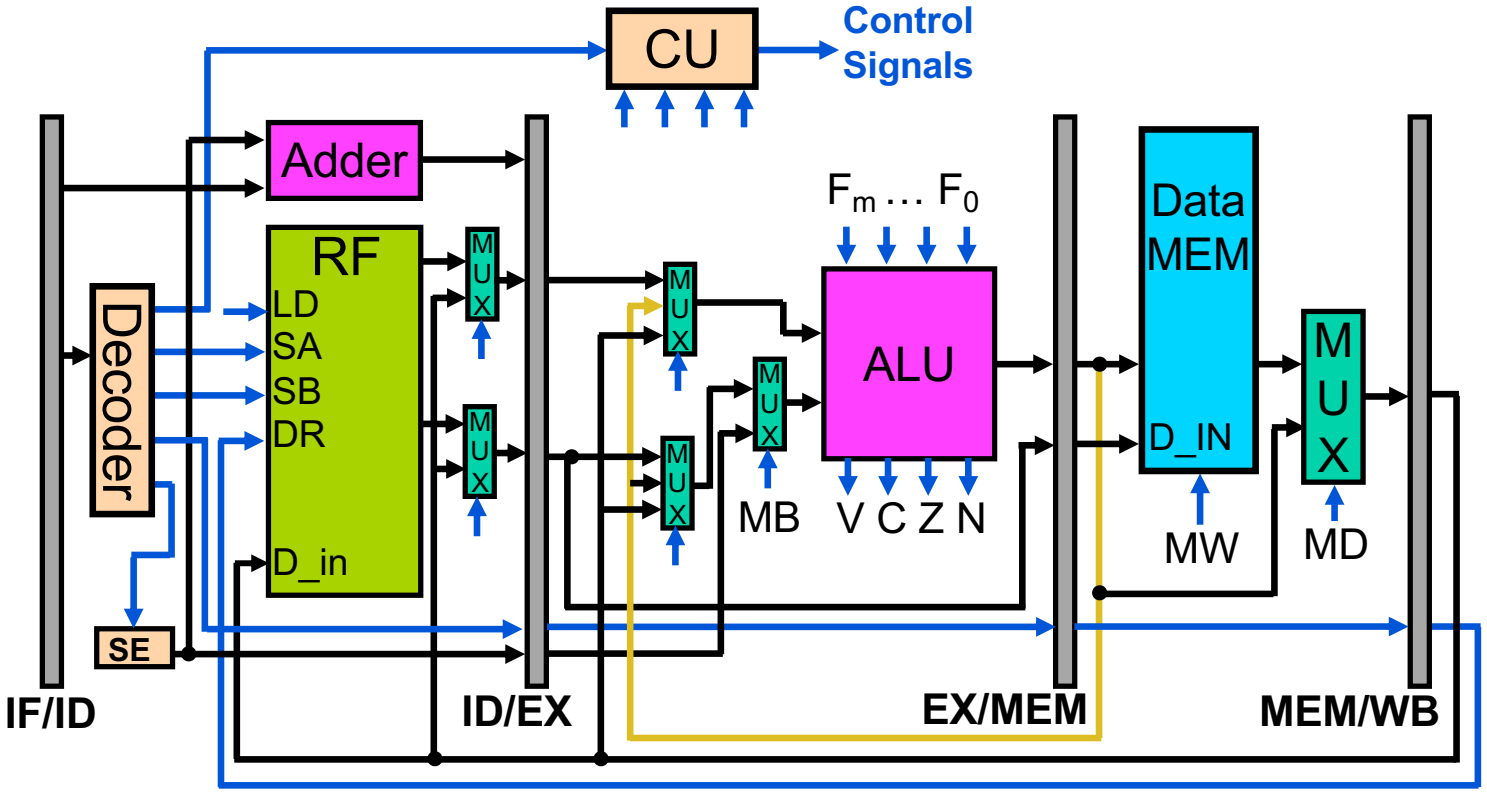


Forwarding in Action



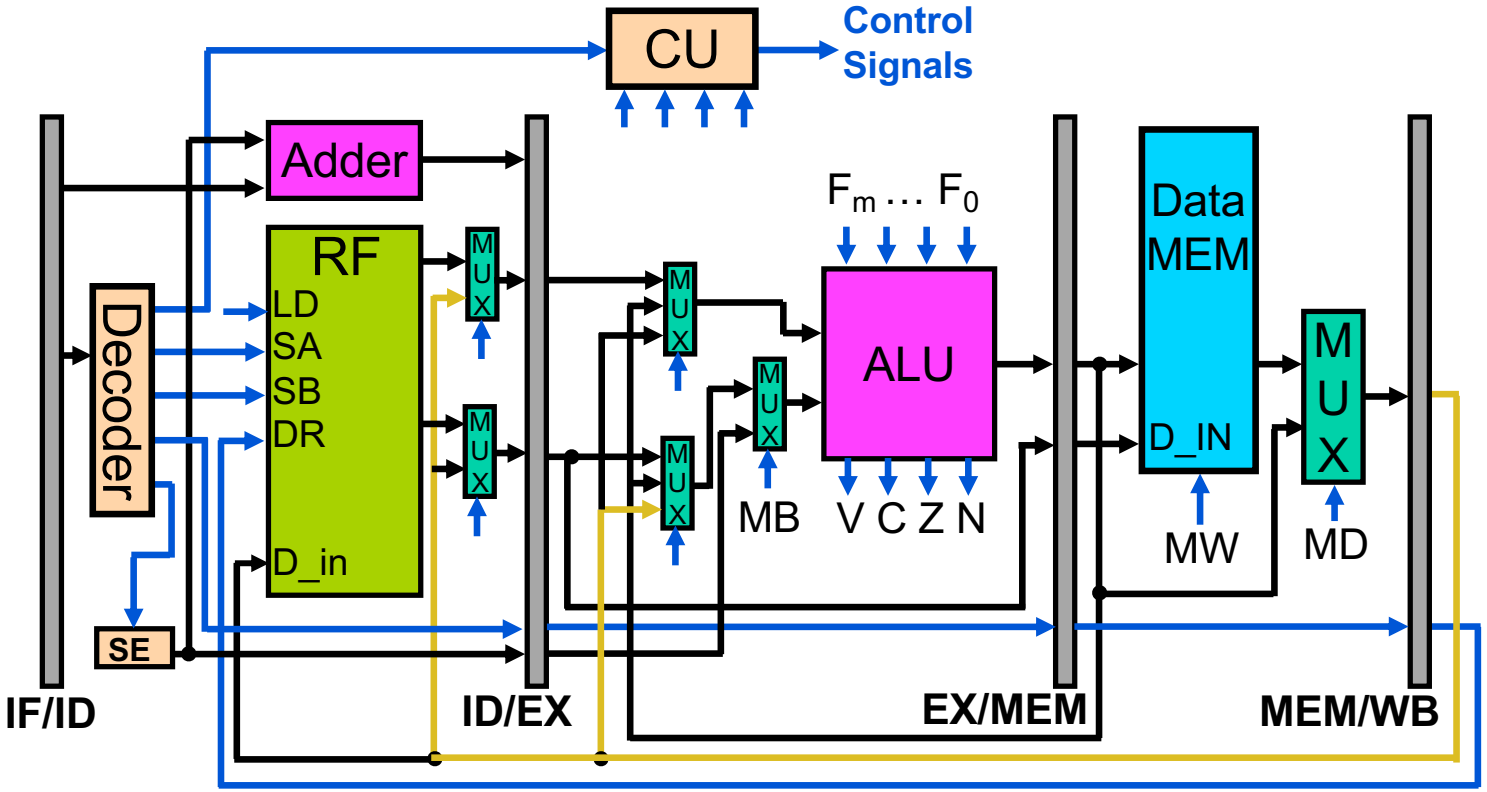
`OR R4,R1,R3` `ADD R1,R2,R3`

Forwarding in Action



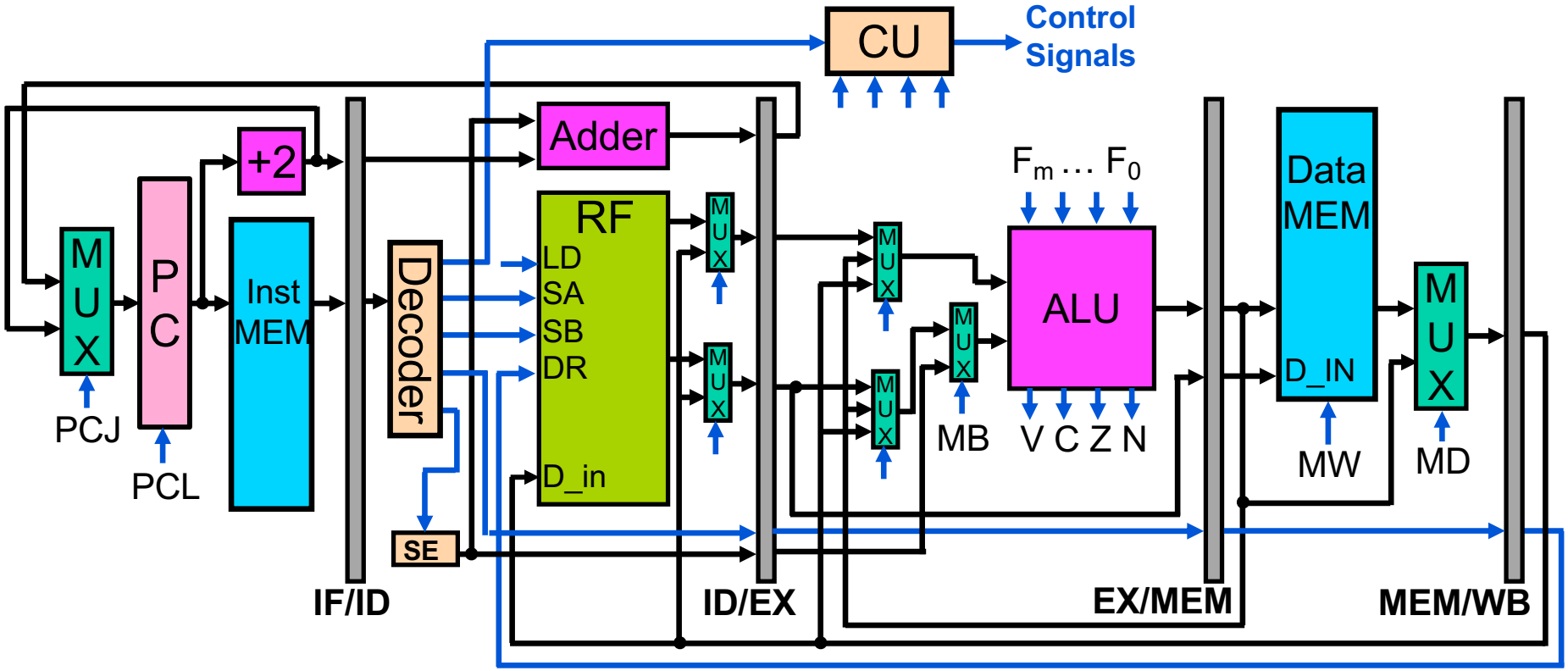

```
SUB R5,R2,R1        OR R4,R1,R3        ADD R1,R2,R3
```

Forwarding in Action




```
AND R6,R1,R2        SUB R5,R2,R1        OR R4,R1,R3        ADD R1,R2,R3
```

HW Forwarding



Trade-off between performance and cost

Example: Data Hazards w/o Forwarding

- Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back

```
ADD R1, R2, R3
OR R4, R1, R3
SUB R5, R2, R1
AND R6, R1, R2
```

Example: Data Hazards w/ Forwarding

- Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back

```
ADD R1, R2, R3
OR R4, R1, R3
SUB R5, R2, R1
AND R6, R1, R2
```

Data hazards resolved by R-type to R-type forwarding

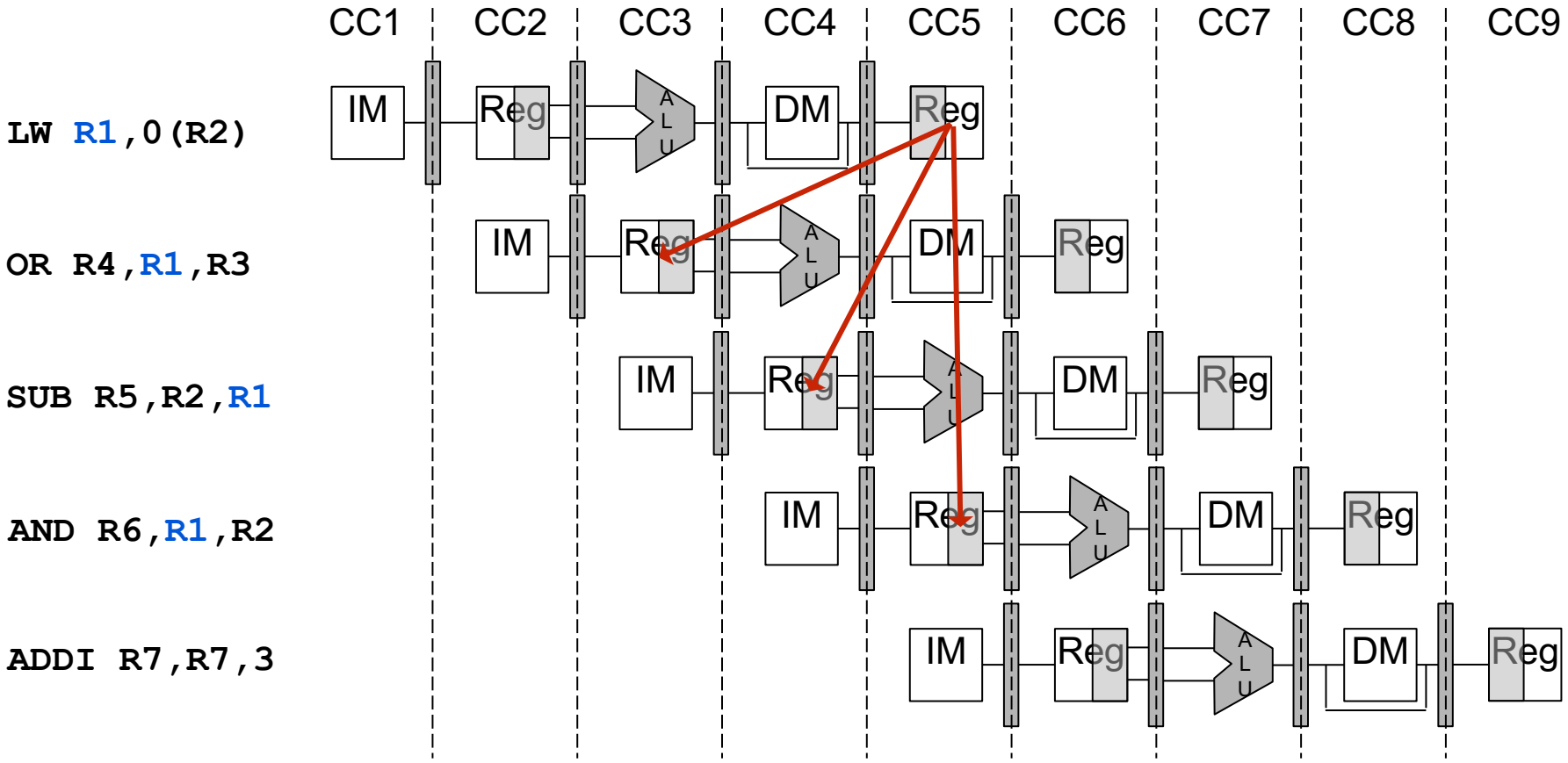
Another Example: Data Hazards w/ Forwarding

- **Identify all data hazards in the following instruction sequences by circling each source register that is read before the updated value is written back**

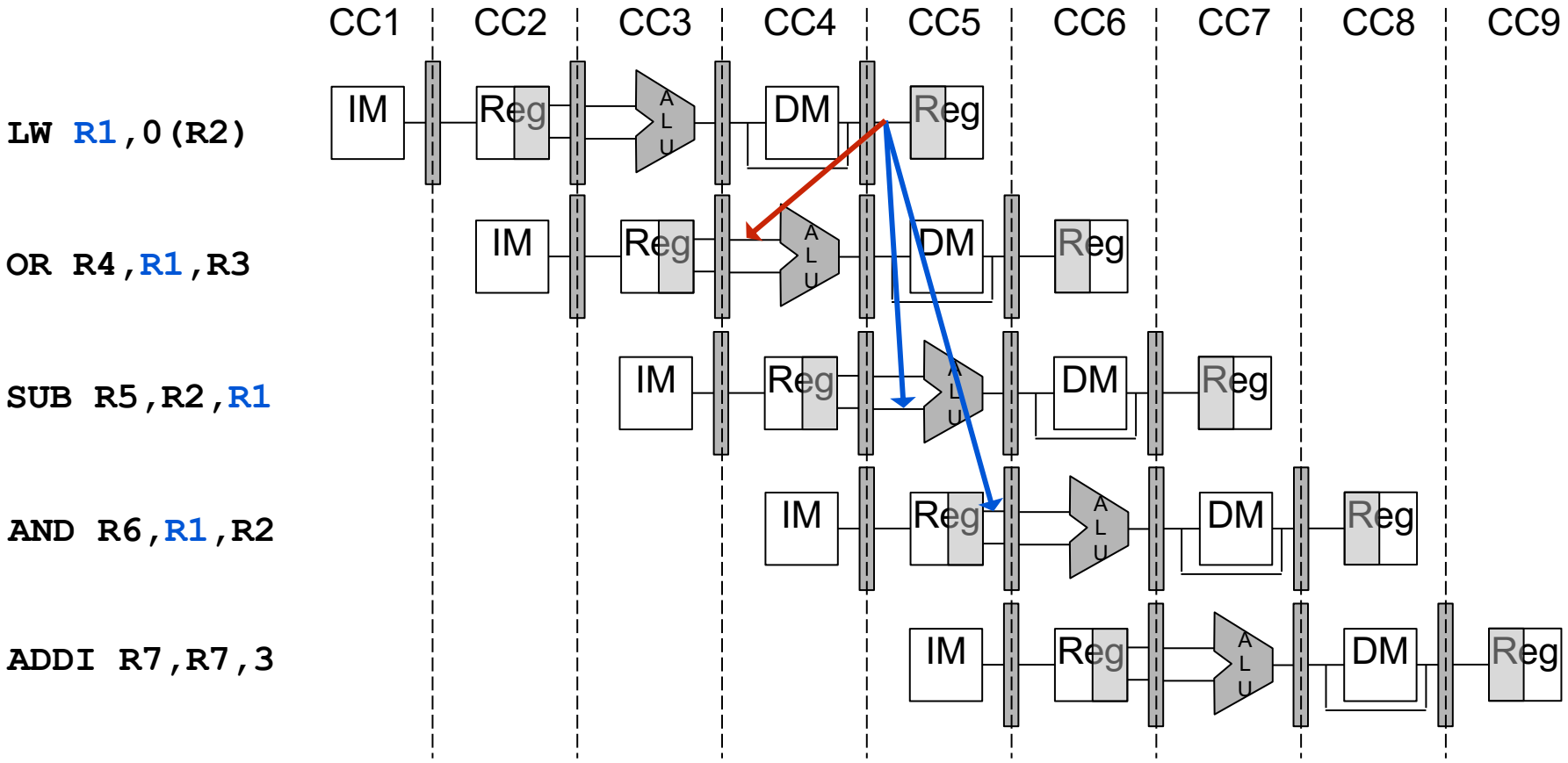
```
LW R1, 0(R2)
OR R4, R1, R3
SUB R5, R2, R1
```

Data hazard not resolved by R-type to R-type forwarding

Data Hazards Caused by Load

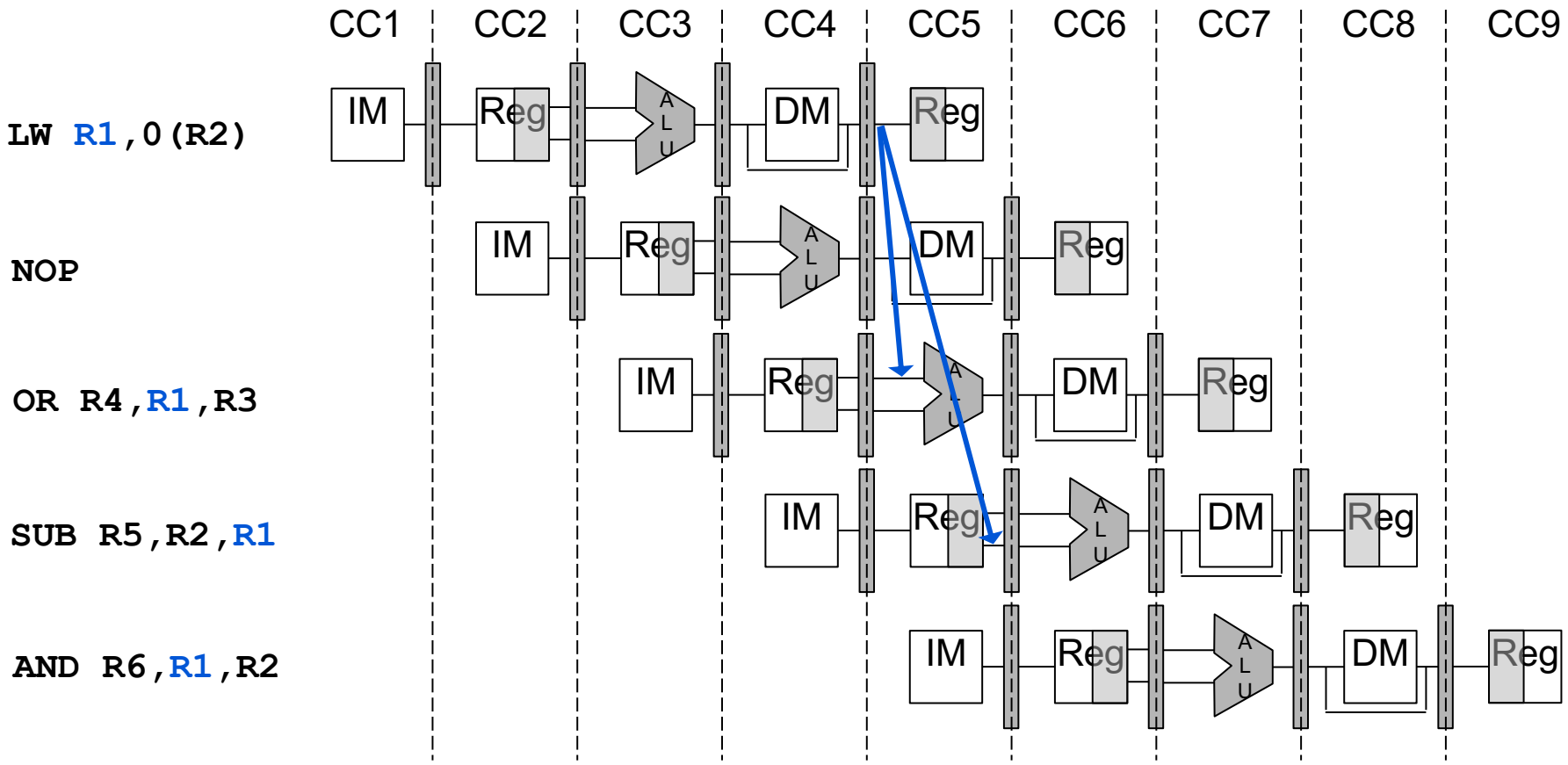


Load Instructions and Forwarding

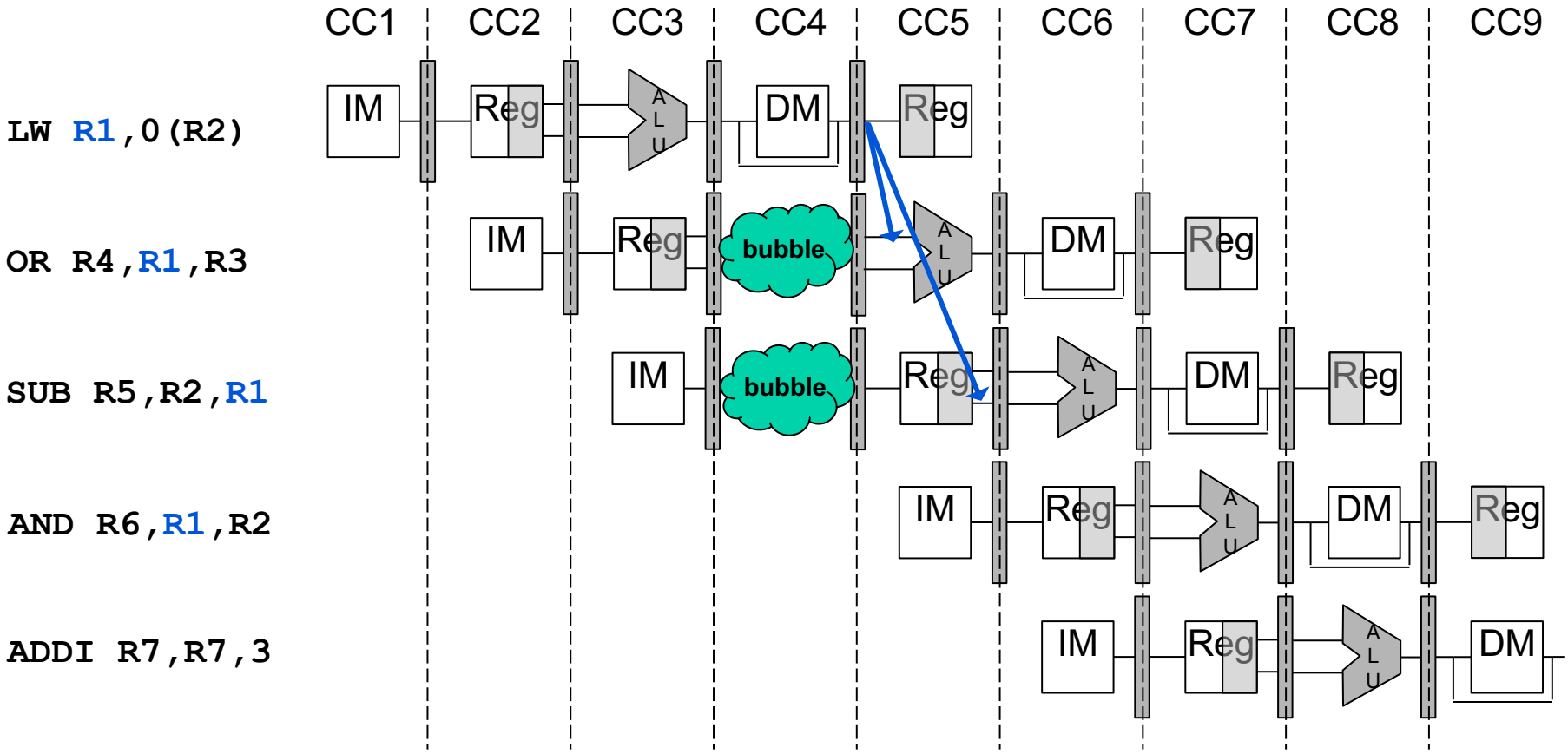


Forwarding can resolve the hazards for SUB and AND, but it's not possible to forward to the OR instruction

Solution 1: Compiler Inserts NOP Instruction



Solution 2: HW Stalls the Pipeline



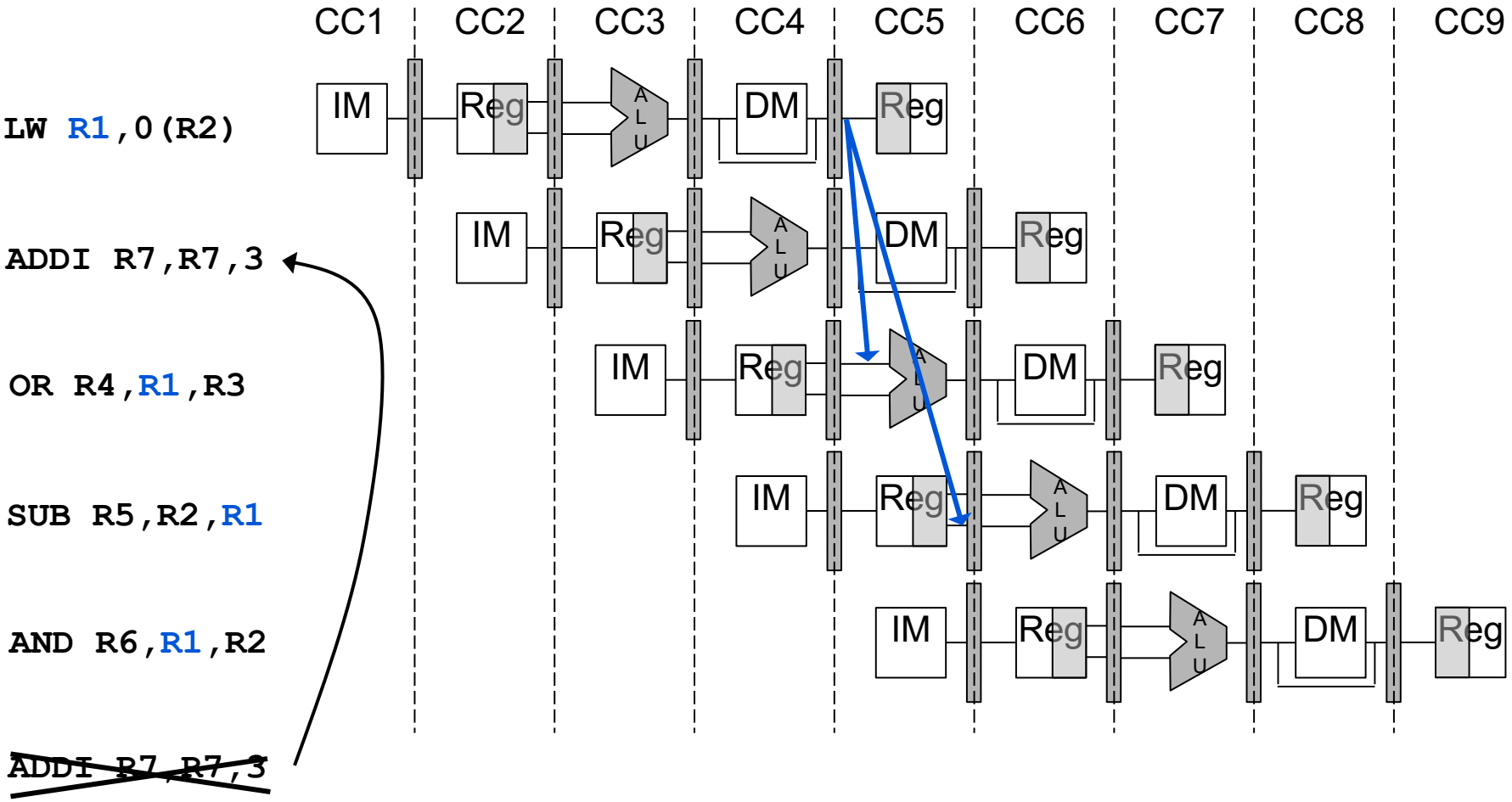
Solution 3: Delay Slots

- A *delay slot* is a location in the program where the compiler is required to insert an instruction between dependent instructions

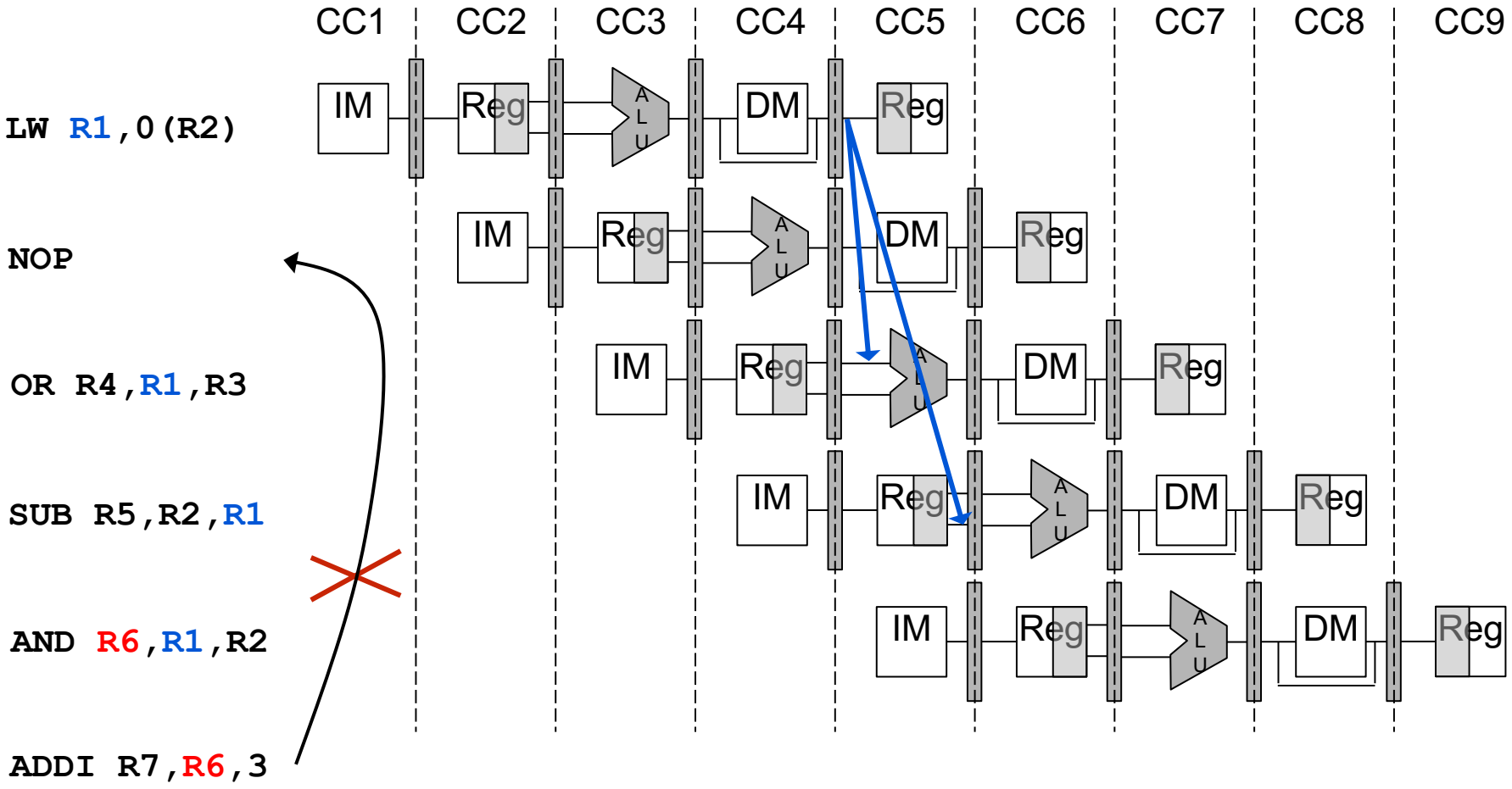
- The ISA defines the delay slots (for specific instructions)

- The compiler can fill a delay slot in two ways:
  (1) by inserting NOP, or
  (2) preferably, by moving *non-dependent instruction* from elsewhere in the program into the delay slot
  - Doing so must not change the function of the program

Example 1: Filling the Load Delay Slot



Example 2: Filling the Load Delay Slot ?



The ADDI instruction cannot be moved to the delay slot due to *data dependence on AND*

Next Class

More Pipelined Microprocessor