ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Pipelined Microprocessor

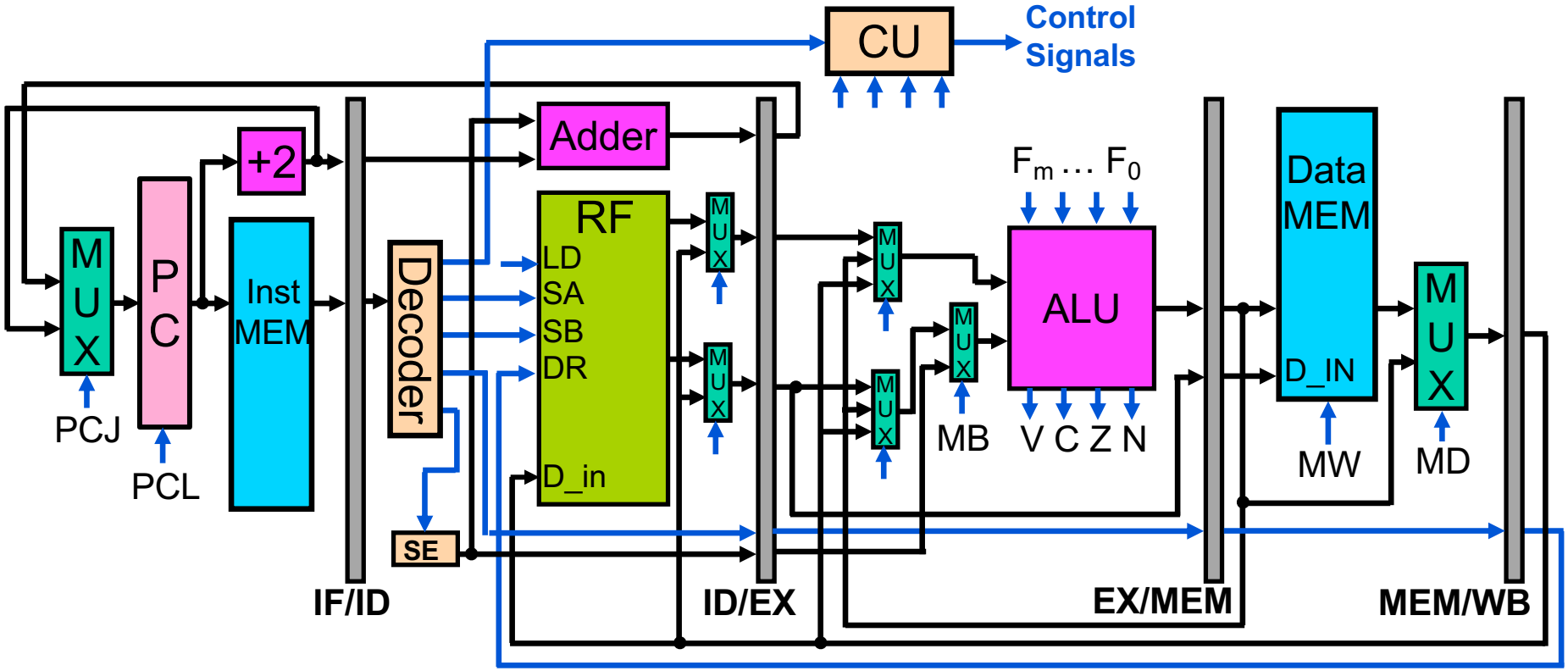
Cornell University

Announcements

- **Prelim 2 tonight in G76 Goldwin Smith Hall (GSH), 7:30-9:00pm**
  - **Closed book, closed notes, closed Internet**
  - **Arrive early by 7:25pm**

- **Prelab 4a due Monday**
  - **Form groups on CMS by Friday**

Review: Pipelined Processor w/ Forwarding



HW Forwarding: True or False

- For a pipeline processor with forwarding, data hazard may still occur in the following scenario

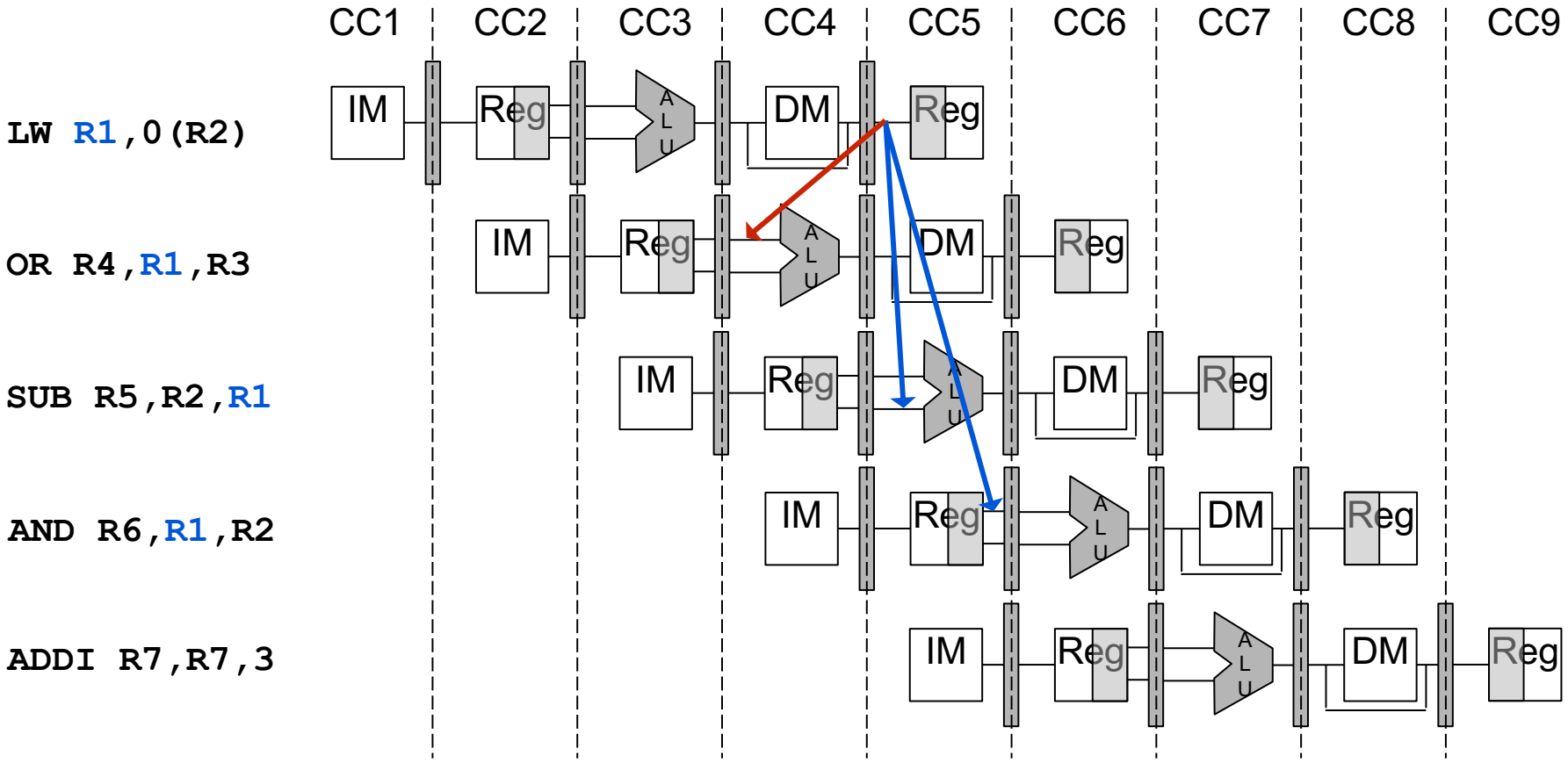
  ADD followed by ADD

  ADD followed by LW

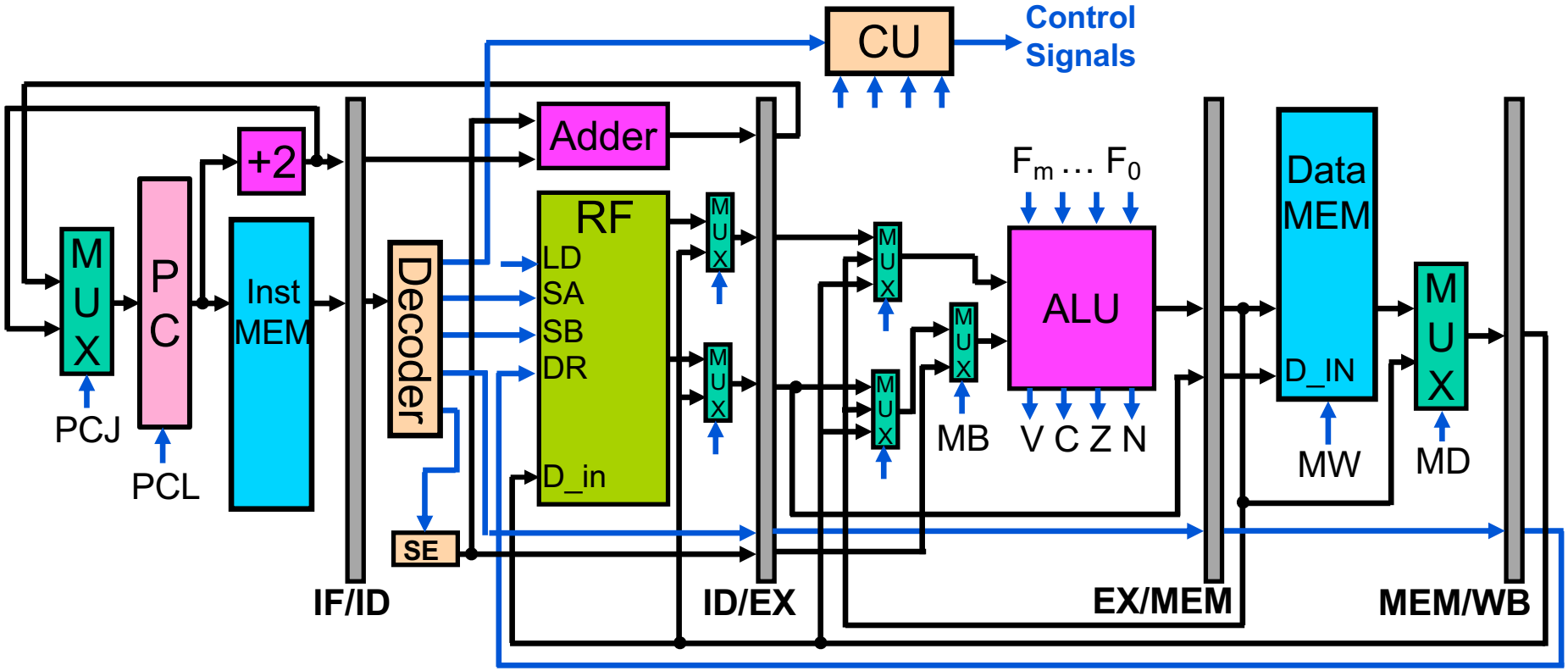
  LW followed by ADD

  LW followed by LW

Review: HW Forwarding for Load

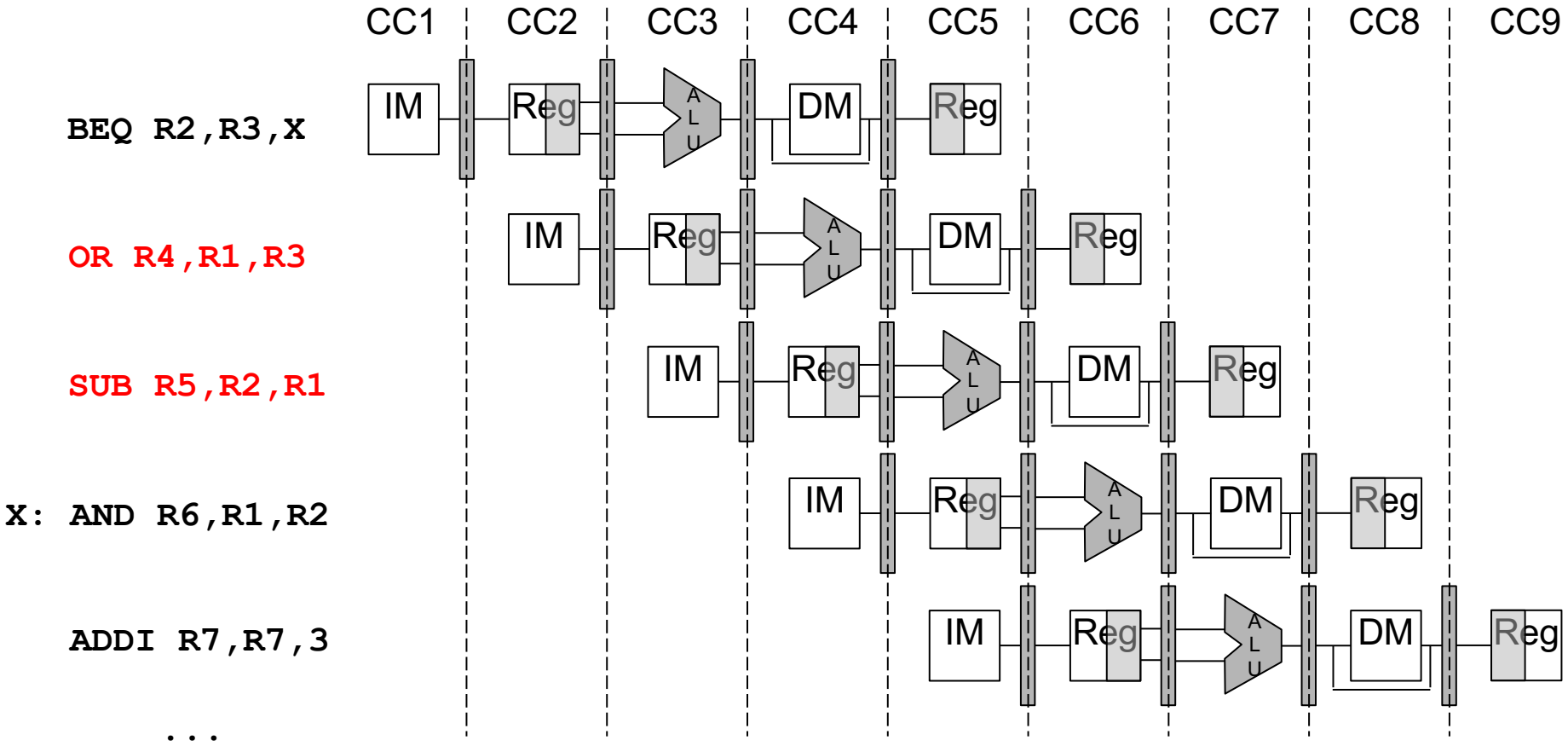


The Problem with Branches



If the condition is met, the PC is updated at the end of EX, after we've already fetched the next two instructions

The Problem with Branches

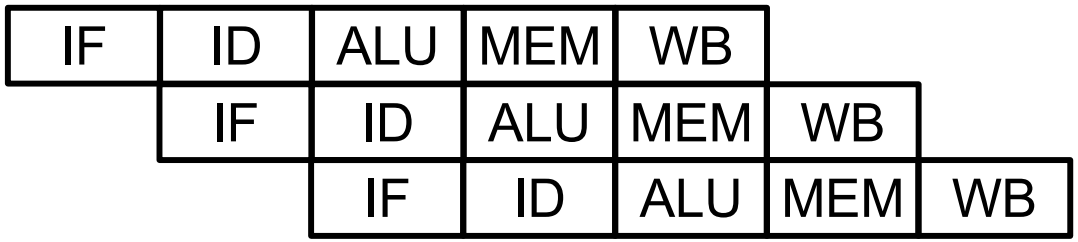


OR and SUB are fetched before the branch condition is evaluated

Control Hazard

- Occurs when instructions following a branch are fetched before the branch outcome is known

```
BEQ R2, R3, X
OR R4, R1, R3
SUB R5, R2, R1
X: AND R6,R1,R2
```

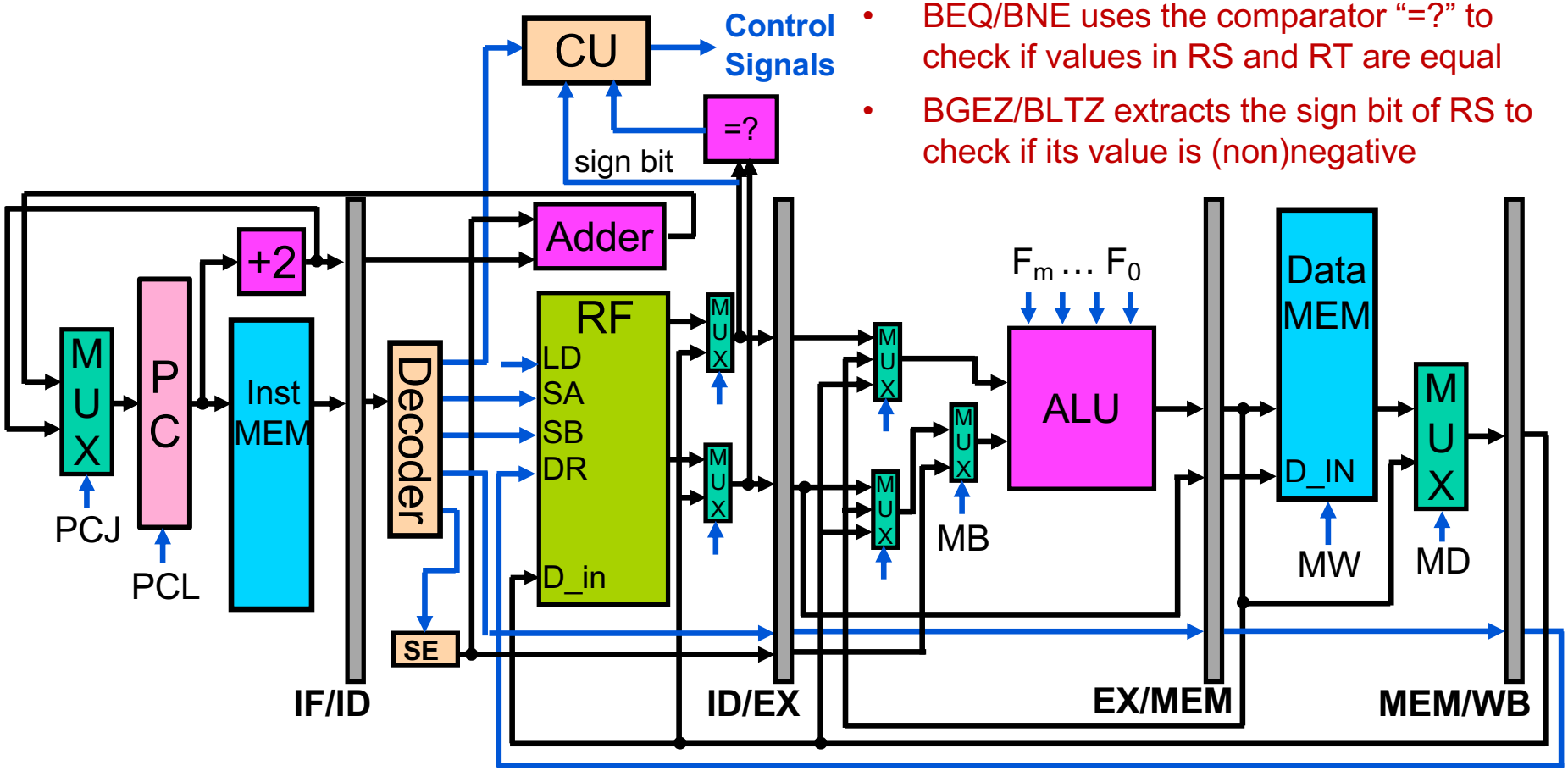


- What should happen
  - If branch is not taken, next fetched instruction should be at address PC+2 (OR)
  - If branch is taken, next fetched instruction should be at address X (AND)
- What actually happens
  - Instructions at PC+2 and PC+4 are fetched before branch outcome is known

Reducing the Branch Delay

- We already calculate the branch target in ID
- Put dedicated hardware to also evaluate the condition in ID

Evaluating Branch Condition in ID Stage



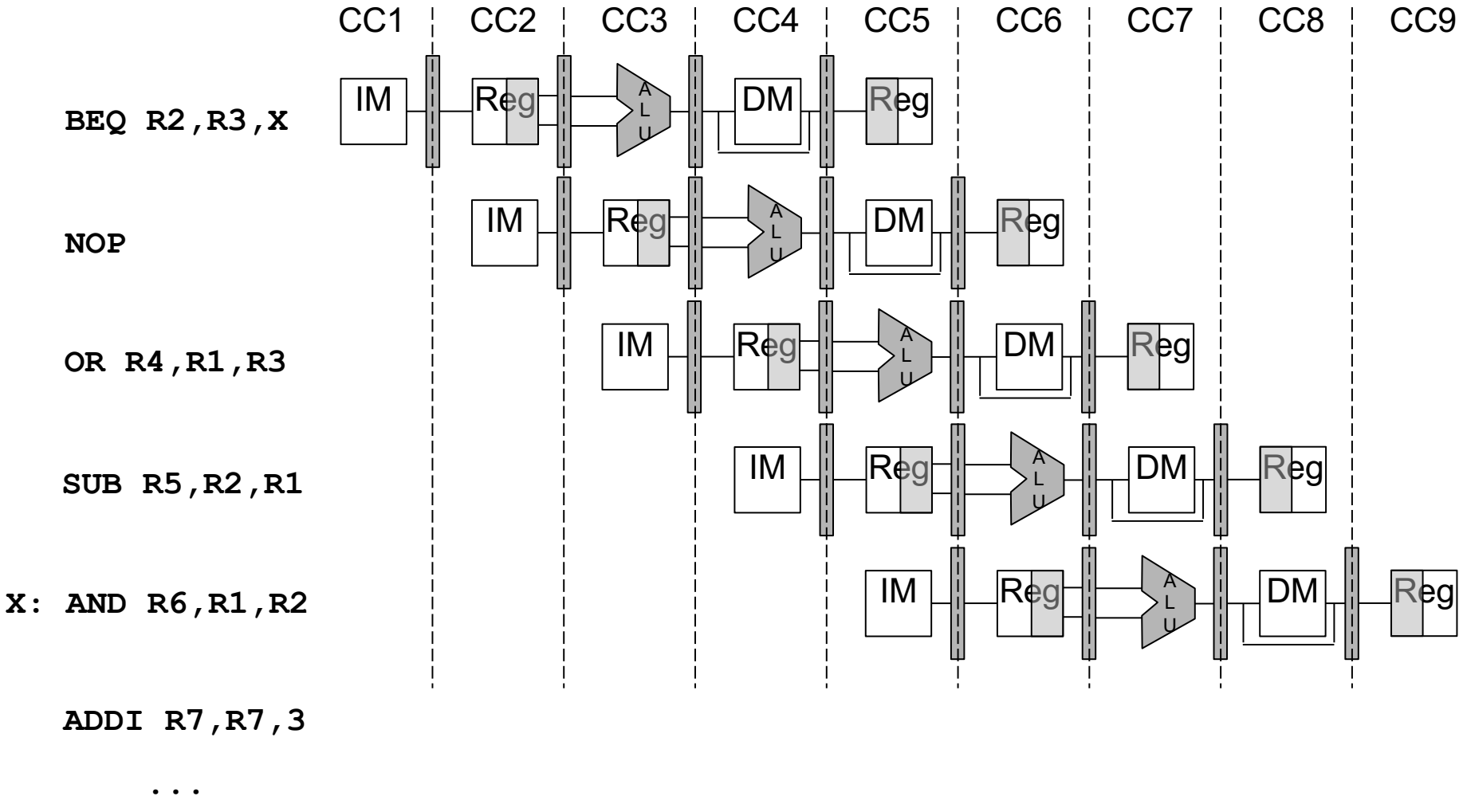
- BEQ/BNE uses the comparator "=?" to check if values in RS and RT are equal
- BGEZ/BLTZ extracts the sign bit of RS to check if its value is (non)negative

Instruction	Format	OP	Operation
BEQ rt,rs,*target*	I	1000	if(R[rs] == R[rt]) PC = PC + sext({imm,1'b0})
BNE rt,rs,*target*	I	1001	if(R[rs] ≠ R[rt]) PC = PC + sext({imm,1'b0})
BGEZ rs,*target*	I	1010	if(R[rs] >= 0) PC = PC + sext({imm,1'b0})
BLTZ rs,*target*	I	1011	if(R[rs] < 0) PC = PC + sext({imm,1'b0})

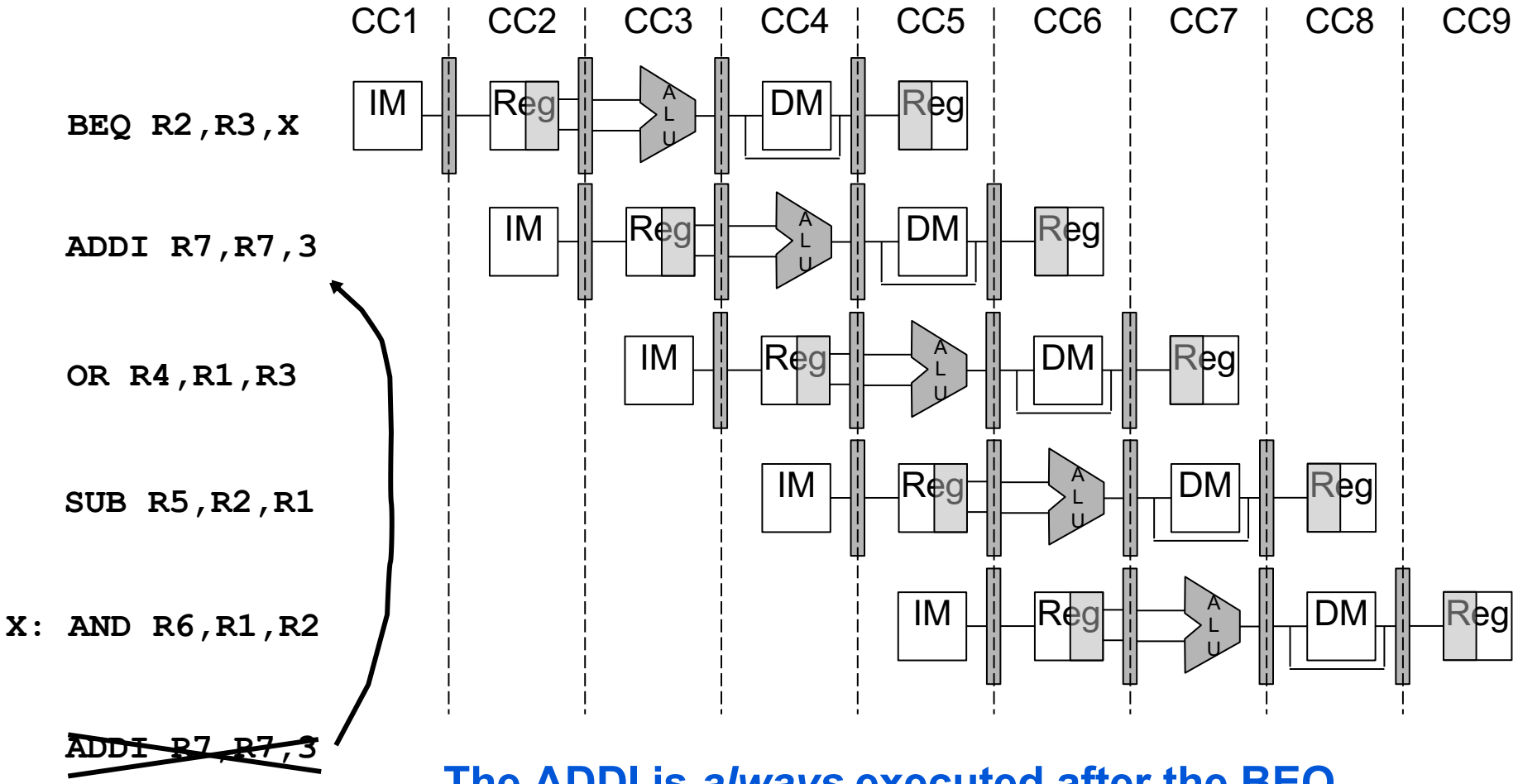
Branch Delay Slot

- If the ISA defines a branch delay slot, the instruction immediately following a branch is *always* executed after the branch
- The compiler finds an instruction to put there, or puts in a NOP
- The hardware *must* execute the instruction immediately following the branch, regardless of whether the branch is taken or not

Filling the Branch Delay Slot with a NOP

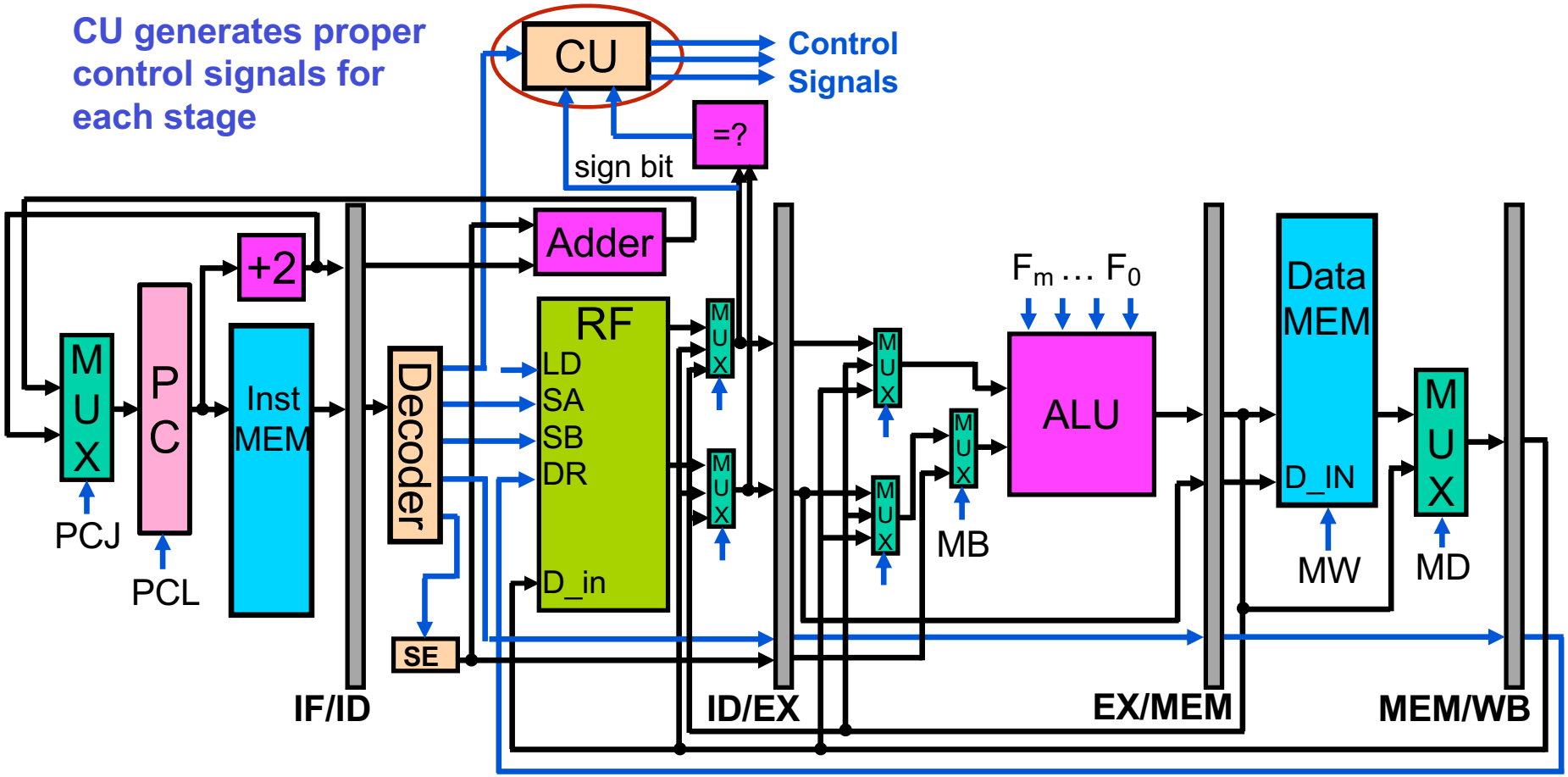


Filling the Branch Delay Slot



The ADDI is *always* executed after the BEQ.
If the BEQ is taken, executing the ADDI must not cause incorrect behavior

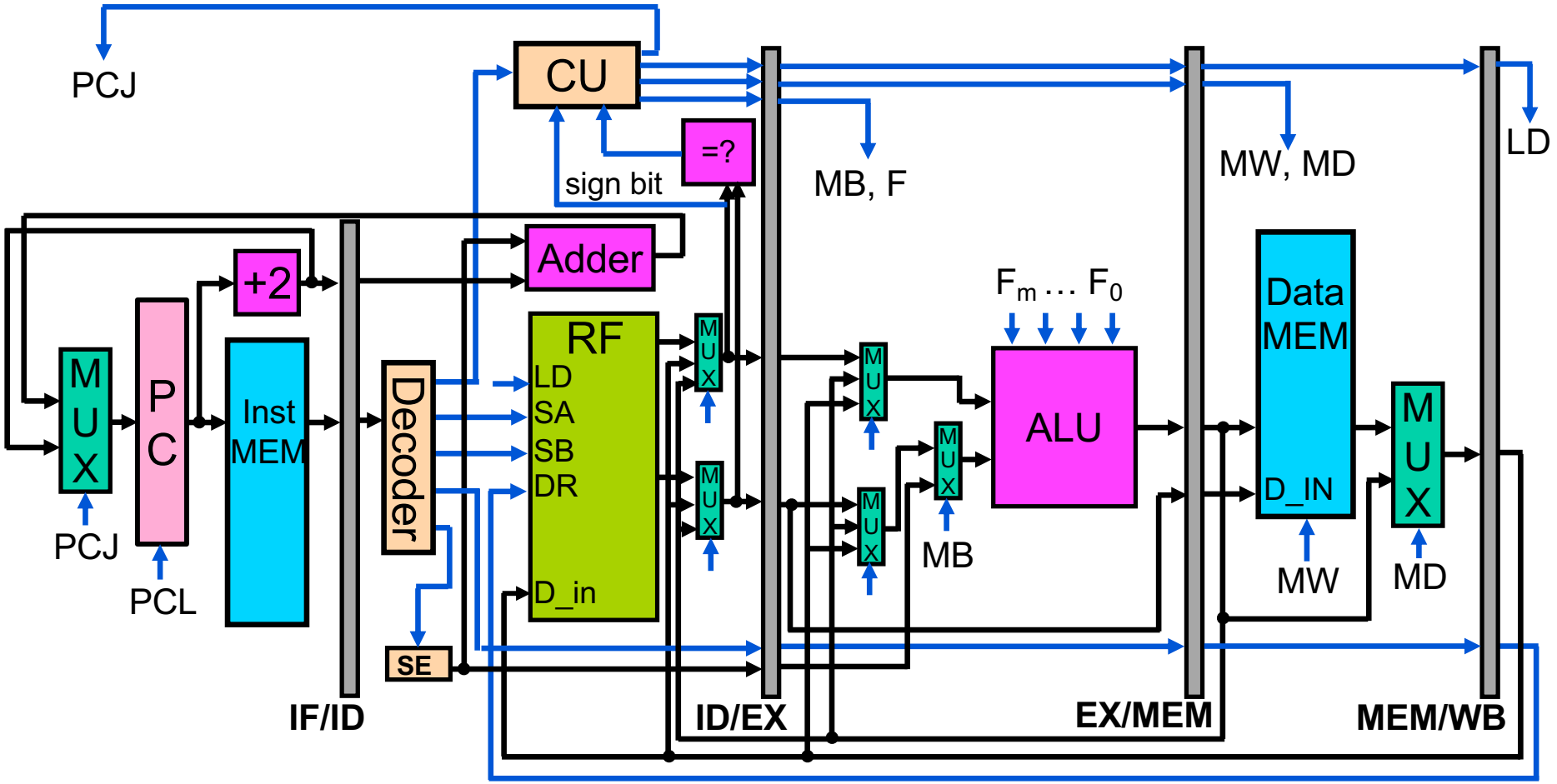
Pipeline Control Unit (CU)



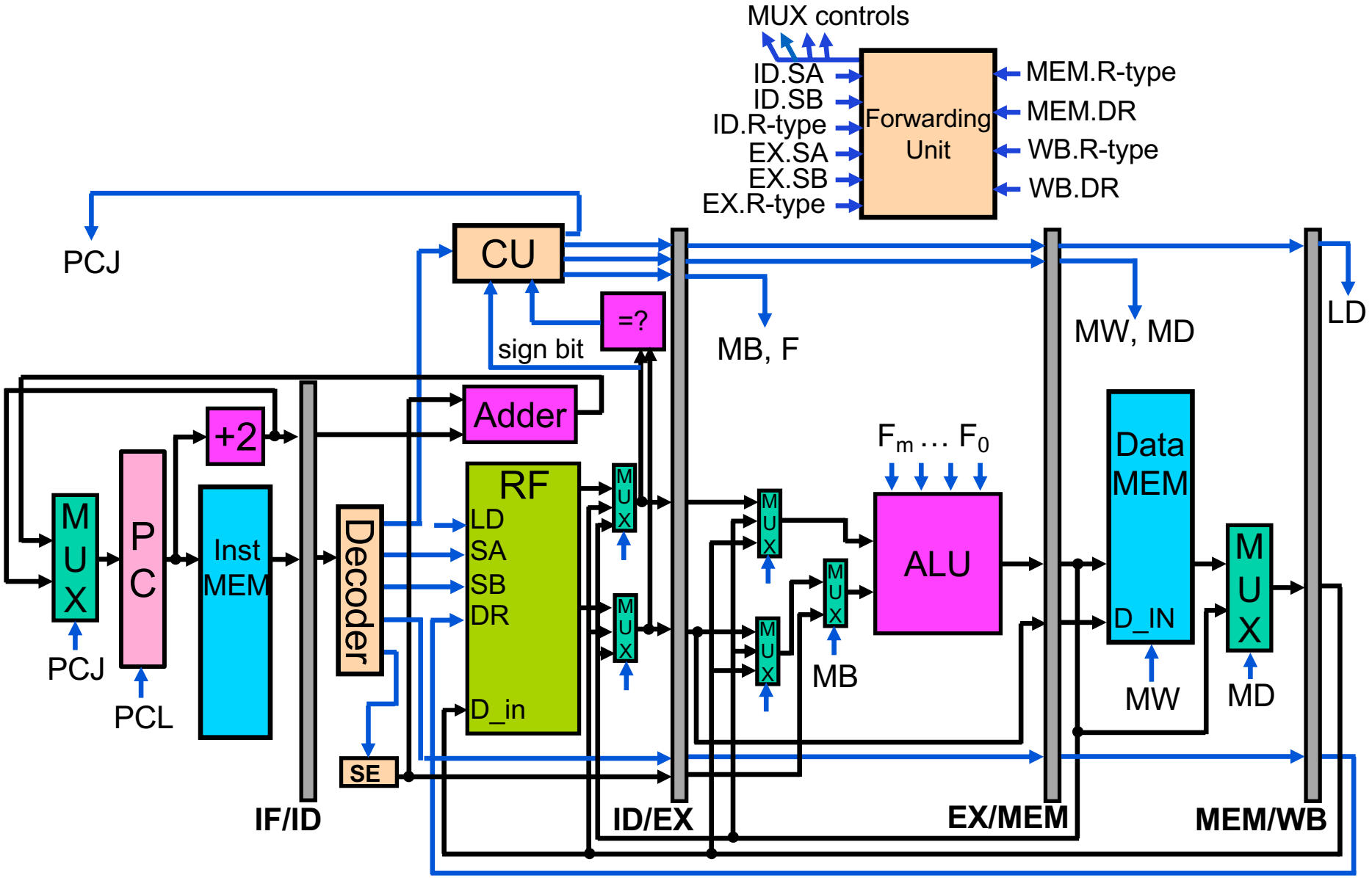
Pipeline Control Requirements

- Generate control signals for each stage
  - IF: PCJ
  - EX: MB, F
  - MEM: MW, MD
  - WB: LD
- Detect forwarding conditions and generate MUX control signals
- Detect data hazards and insert pipeline bubbles
  - Assume no load delay slot defined in ISA
- Assume one branch delay slot defined in ISA

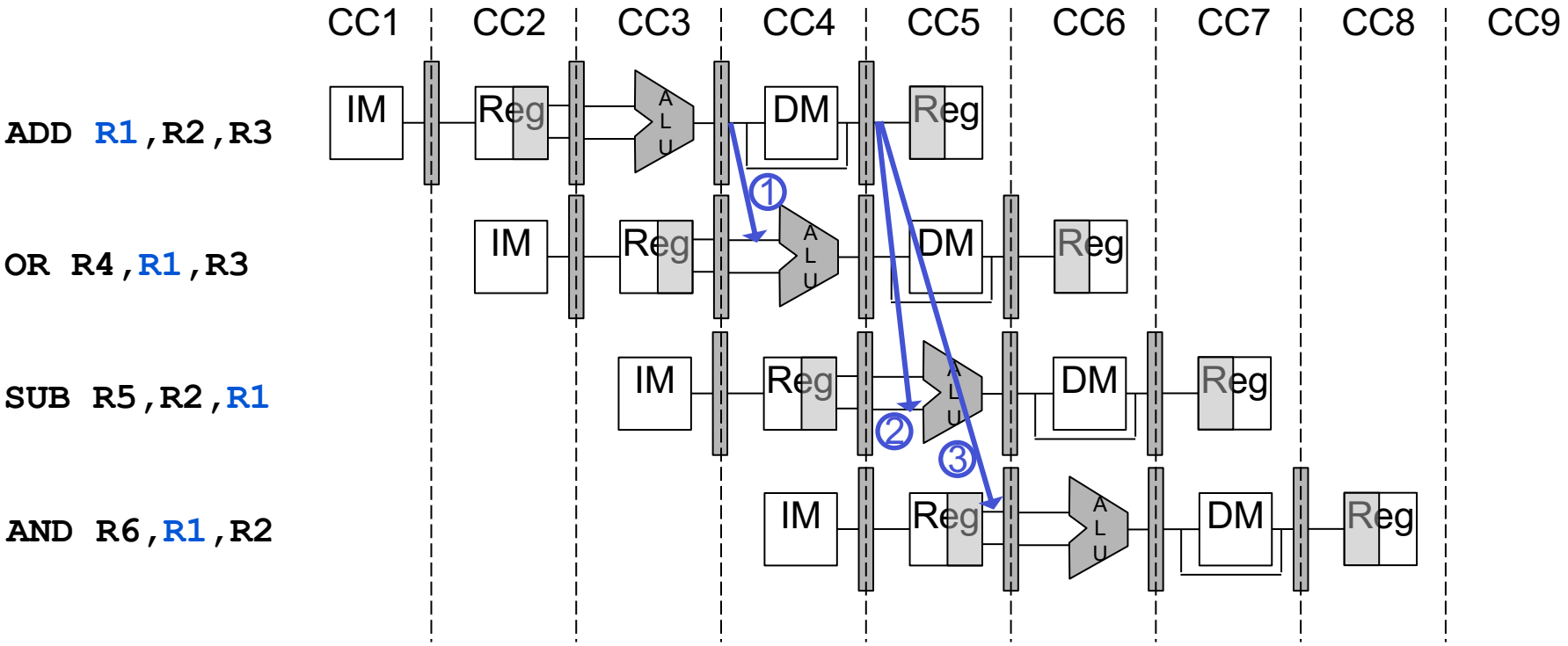
Generating Control for Each Stage



Forwarding Unit (Partial)



R-type to R-type Forwarding



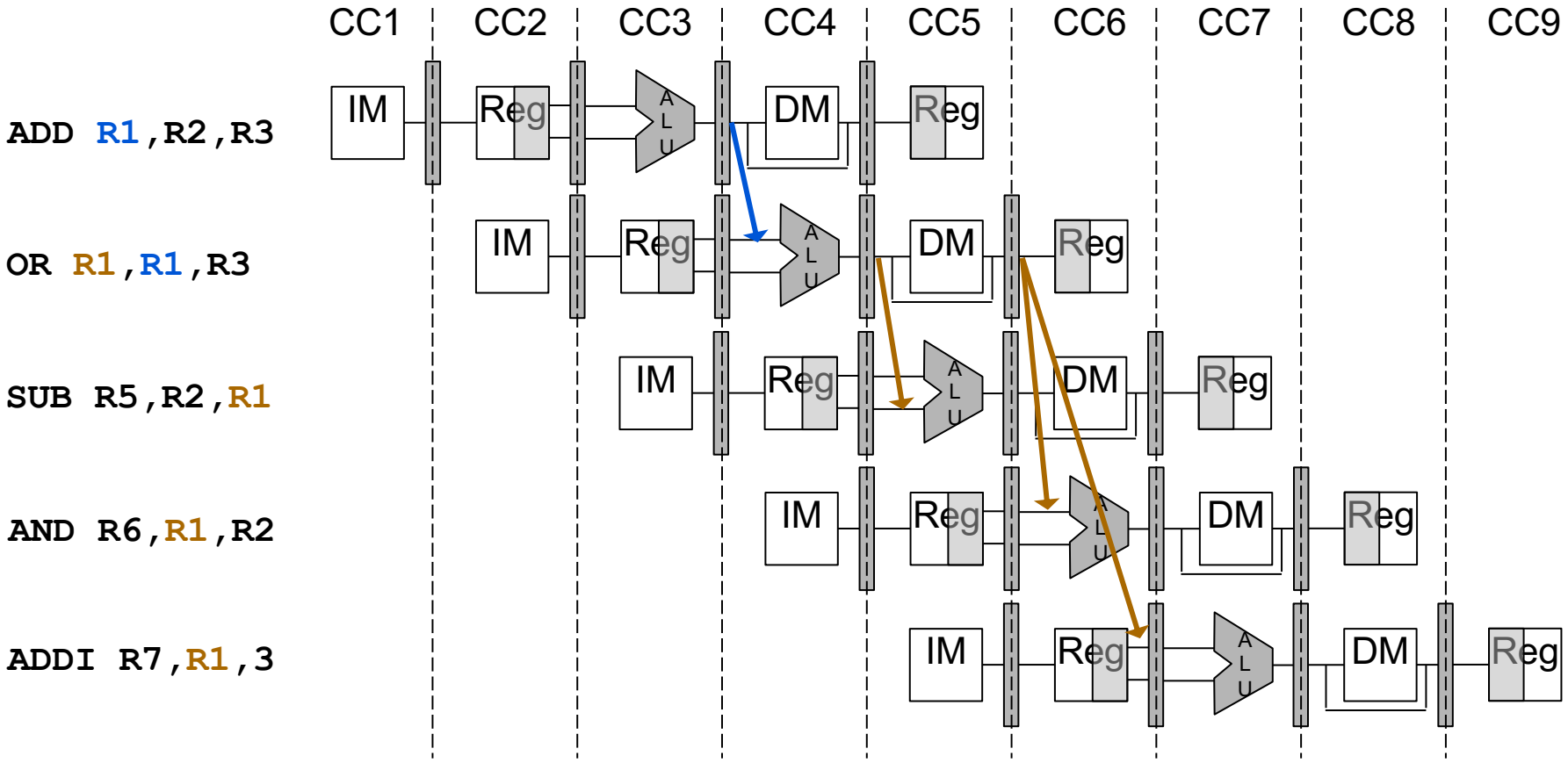
Forwarding path **Conditions to enable the forwarding**

① **MEM→EX :** **MEM.DR == (EX.SA or EX.SB)**

② **WB→EX :** **(1) WB.DR == (EX.SA or EX.SB) and**
(2) there is no forwarding from MEM→EX in this cycle

③ **WB→ID :** **WB.DR == (ID.SA or ID.SB)**

Another Example: R-type to R-type Forwarding



Forwarding path from ADD to SUB (via WB→EX) must be disabled so the updated R1 produced by OR is forwarded to SUB (via MEM→EX)

Summary: R-type to R-type Forwarding Conditions

① MEM→EX

- MEM.DR == (EX.SA or EX.SB)

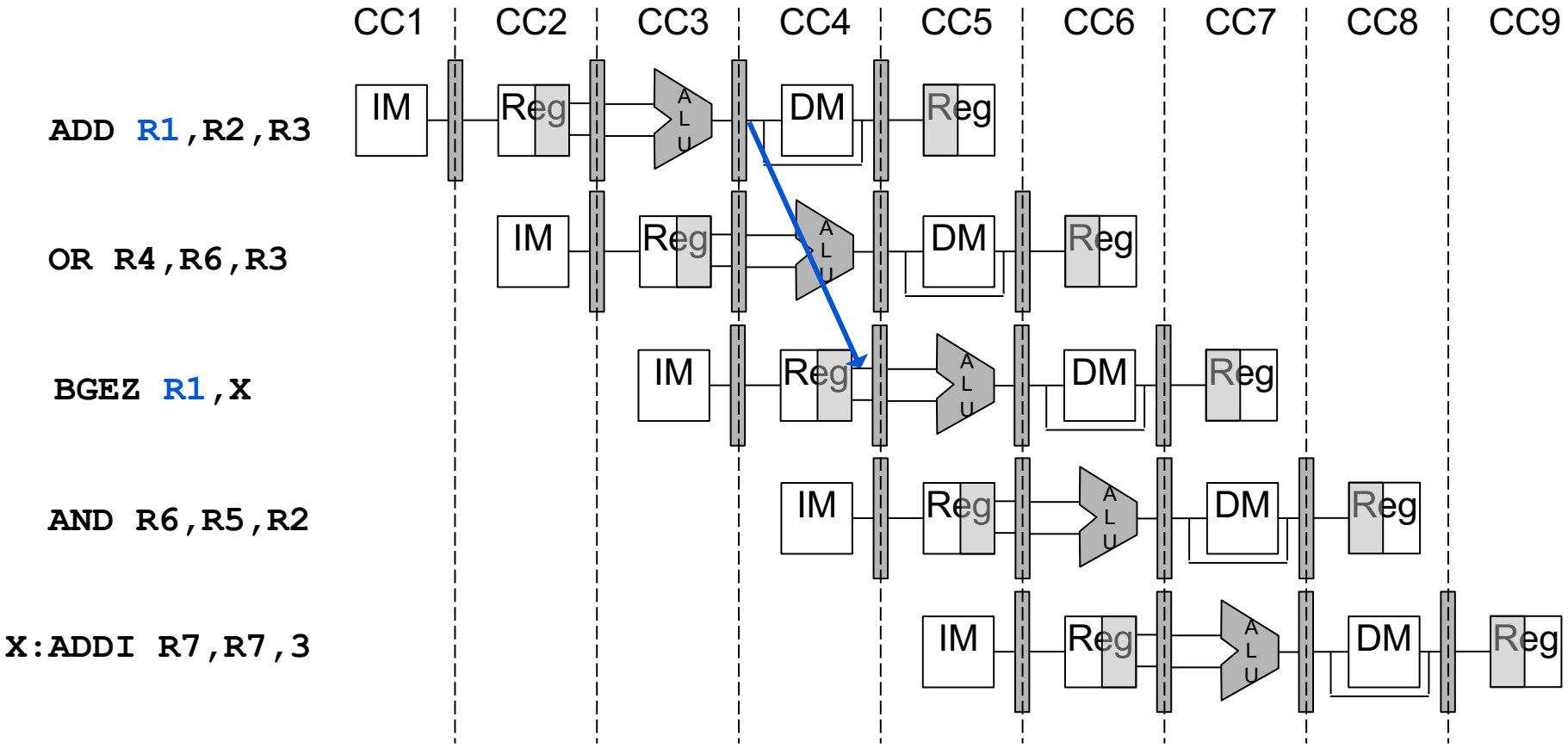
② WB→EX

- WB.DR == (EX.SA or EX.SB) *and*
- MEM.DR != (EX.SA or EX.SB)

③ WB→ID

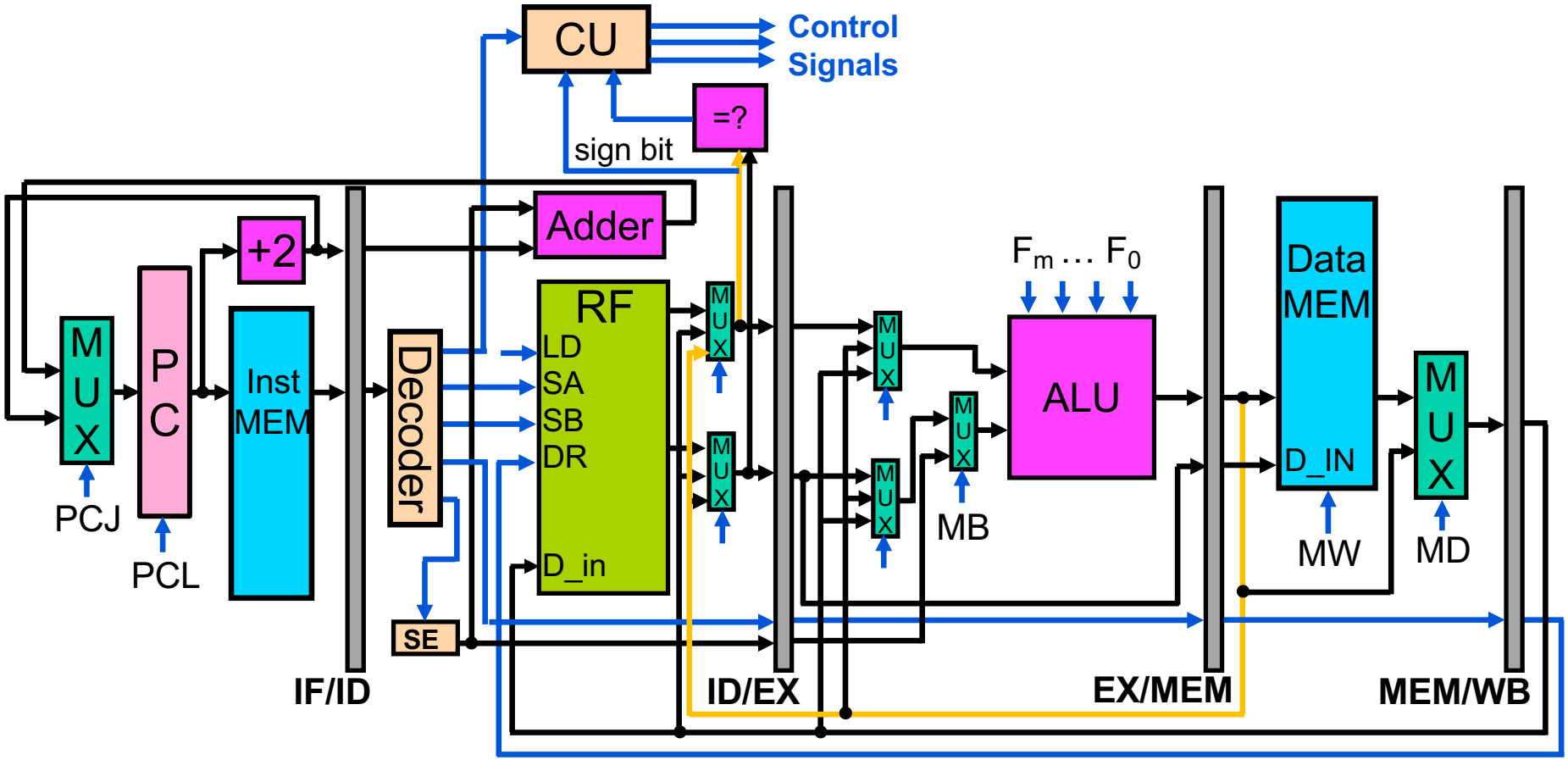
- WB.DR == (ID.SA or ID.SB)

R-type to Branch Forwarding



- **There are other forwarding conditions to handle as well (such as WB→ID)**

Pipeline with Fwding + Branch HW in ID

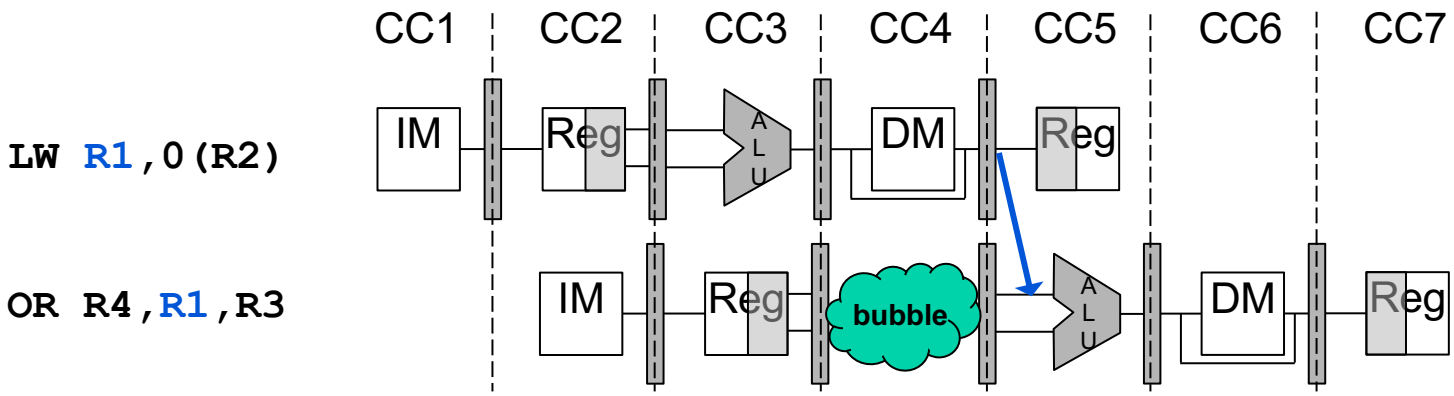



```
AND R6,R5,R2        BGEZ R1,X        OR R4,R6,R3        ADD R1,R2,R3
```

Data Hazards Requiring Bubbles

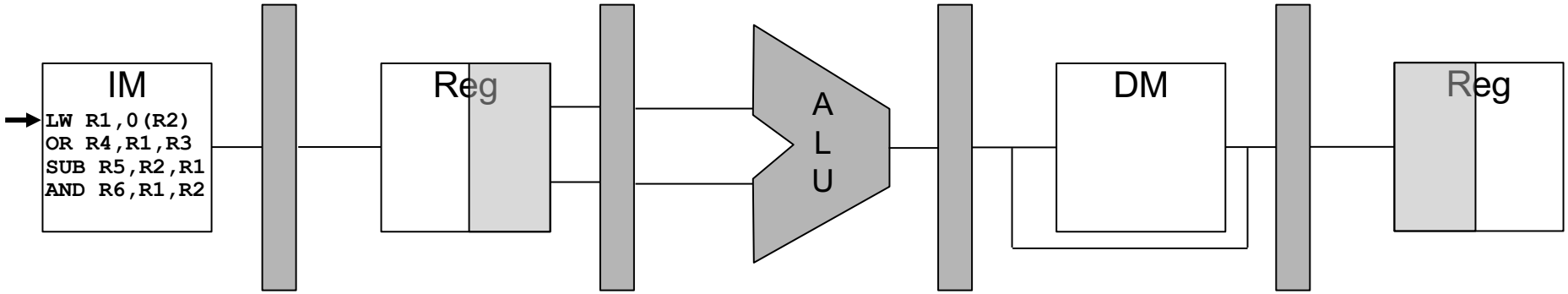
- Occur when instructions are too close together for forwarding to work
- Requires adding bubbles in the pipeline
- Data hazard conditions to detect and handle
  - Load followed by R-type
  - Load followed by I-type ALU instruction
  - Load followed by Load
  - Load followed by Store (two cases)
  - Load followed by Branch
  - ALU instruction followed by Branch

Load Followed by R-type Instruction



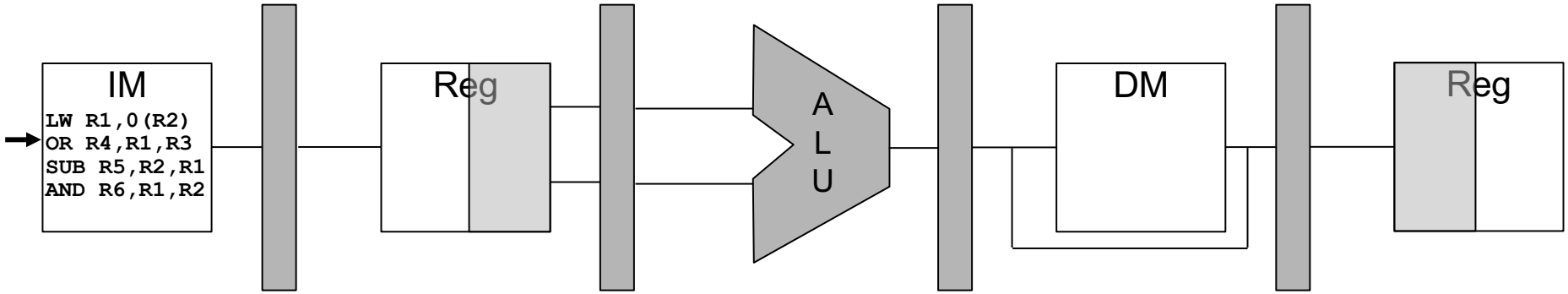

```
if (EX.Load && ID.R-type) {                  // Inst in EX is a Load and inst in ID is R-type
    if (EX.DR == (ID.SA || ID.SB)) {         // DR of Load matches SA or SB of inst in ID
        Insert NOP into EX in next cycle // Insert bubble
        Don't Load IF/ID in next cycle     // Hold instruction in ID for a cycle
        Don't Load PC in next cycle        // Hold instruction in IF for a cycle
    }
}
```

Load Followed by R-type Instruction



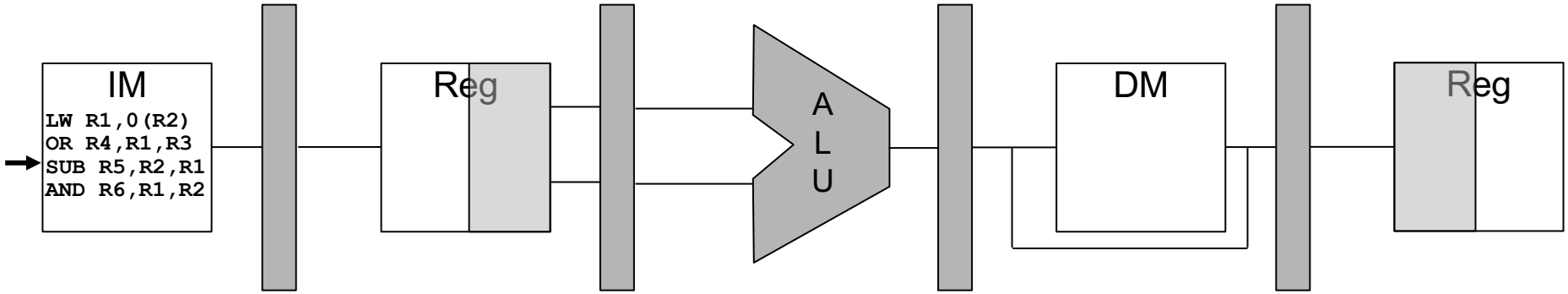
LW R1,0(R2)

Load Followed by R-type Instruction



OR R4,R1,R3 **LW R1,0(R2)**

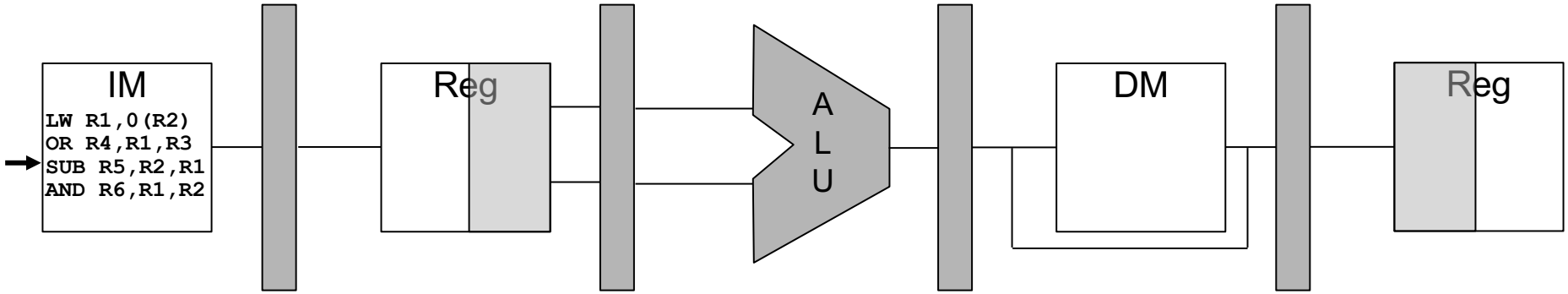
Load Followed by R-type Instruction



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

Detect hazard

Load Followed by R-type Instruction

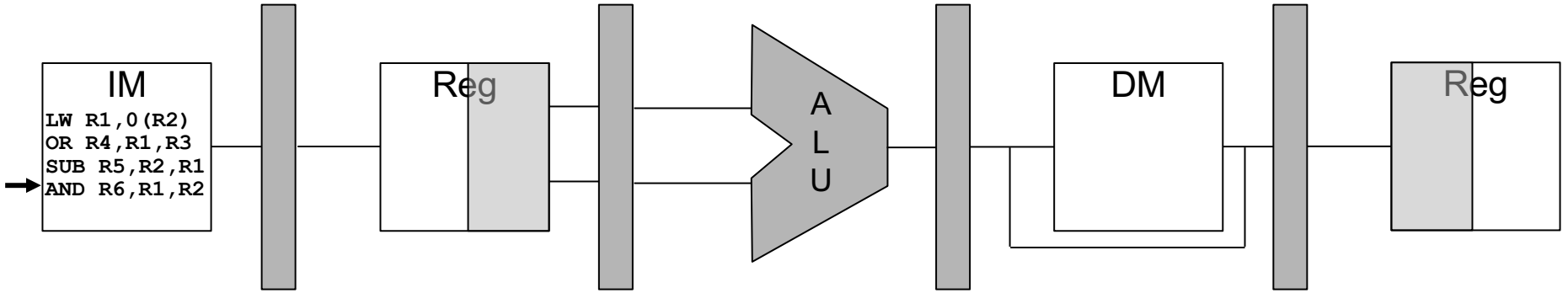


SUB R5,R2,R1 OR R4,R1,R3 NOP LW R1,0(R2)

Hold these instructions

Insert bubble

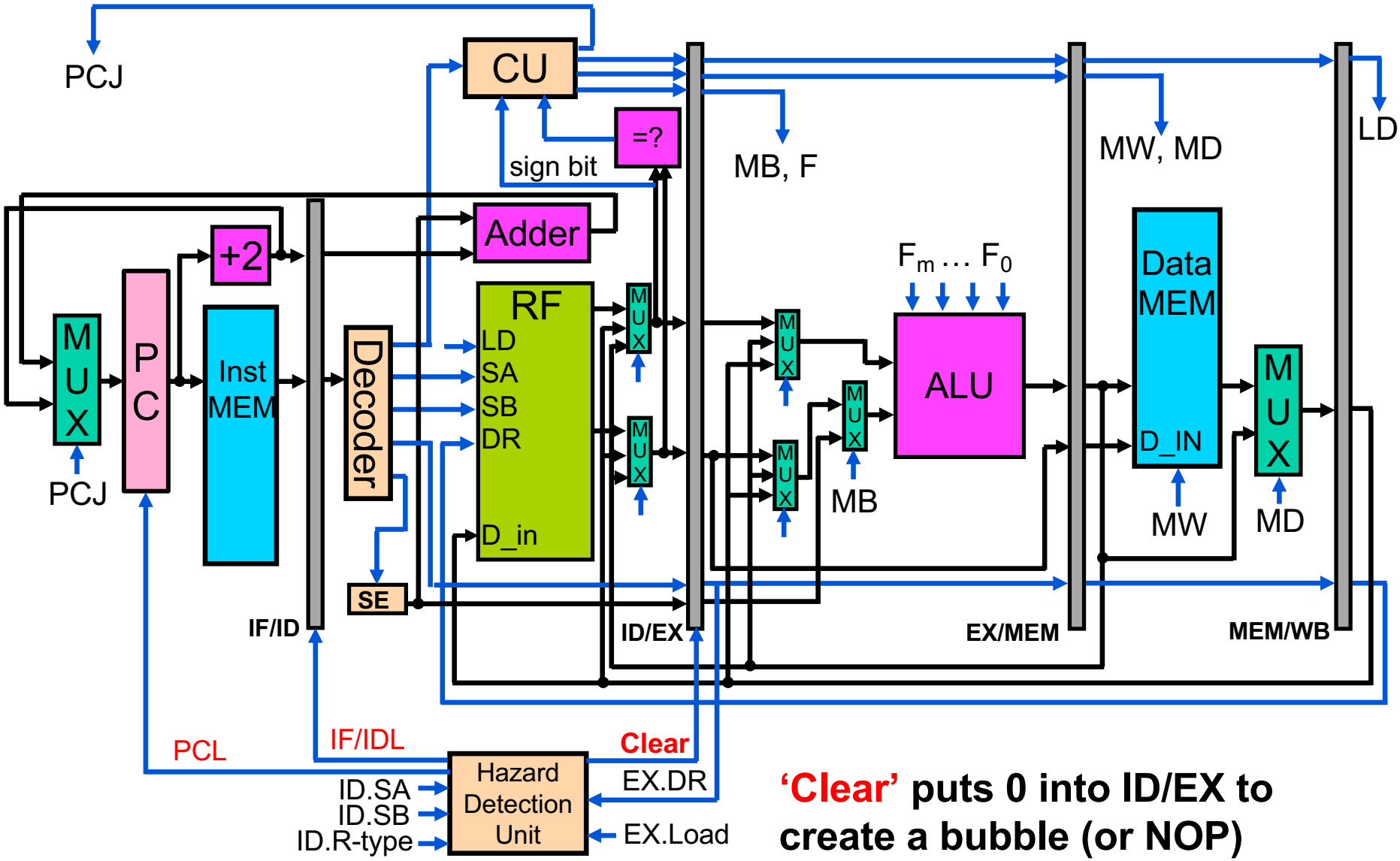
Load Followed by R-type Instruction



AND R6,R1,R2 SUB R5,R2,R1 OR R4,R1,R3 NOP LW R1,0(R2)

Pipeline continues normally
R1 value forwarded from WB to EX and ID

Hazard Detection Unit (Partial)

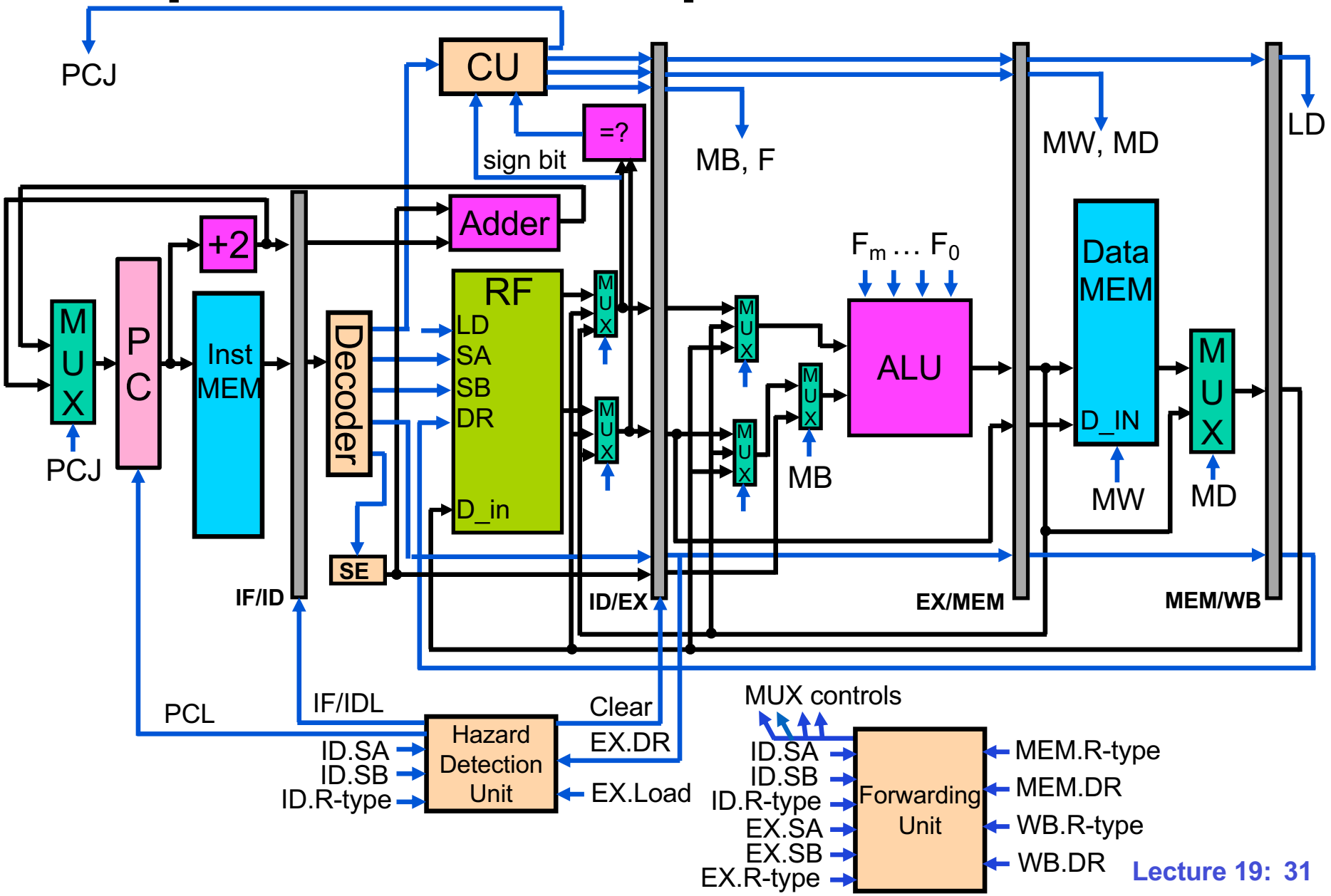


'Clear' puts 0 into ID/EX to create a bubble (or NOP)

'PCL' is the write enable for PC

'IF/IDL' is the write enable for the IF/ID register

Pipeline with Datapath and Control



Next Class

Caches
(H&H 8-8.3)