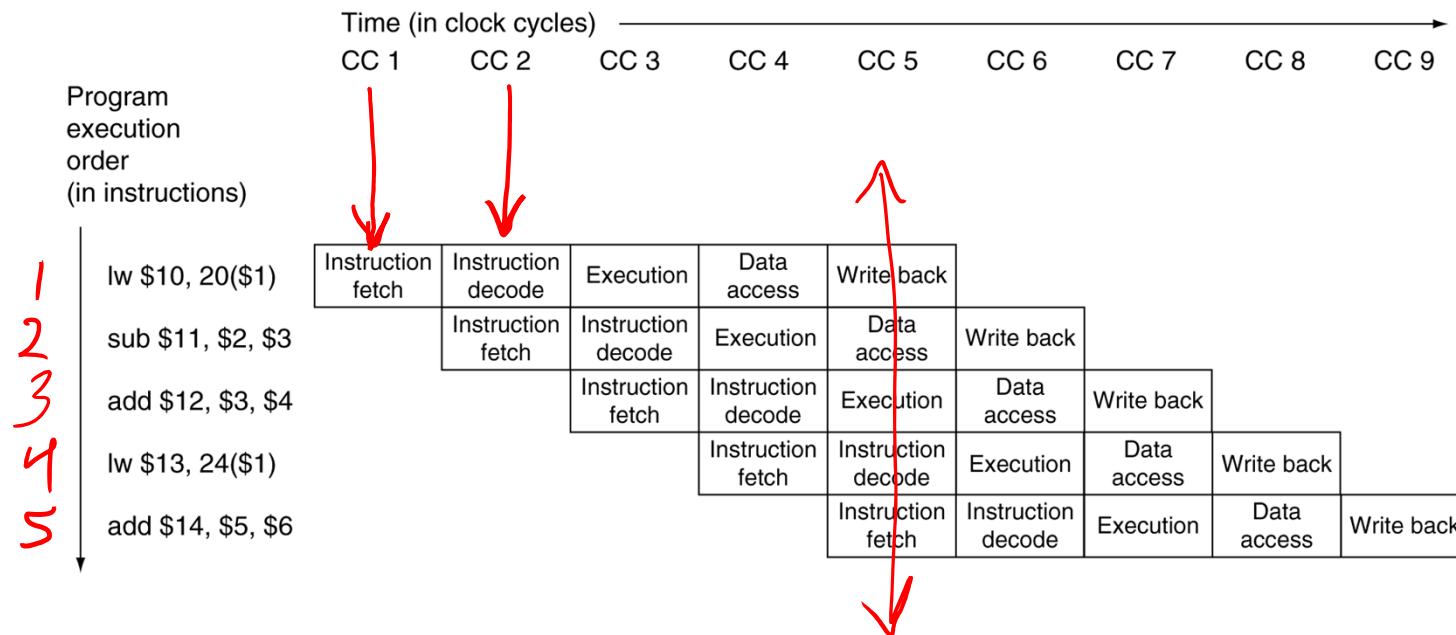


# Multi-Cycle Pipeline Diagram

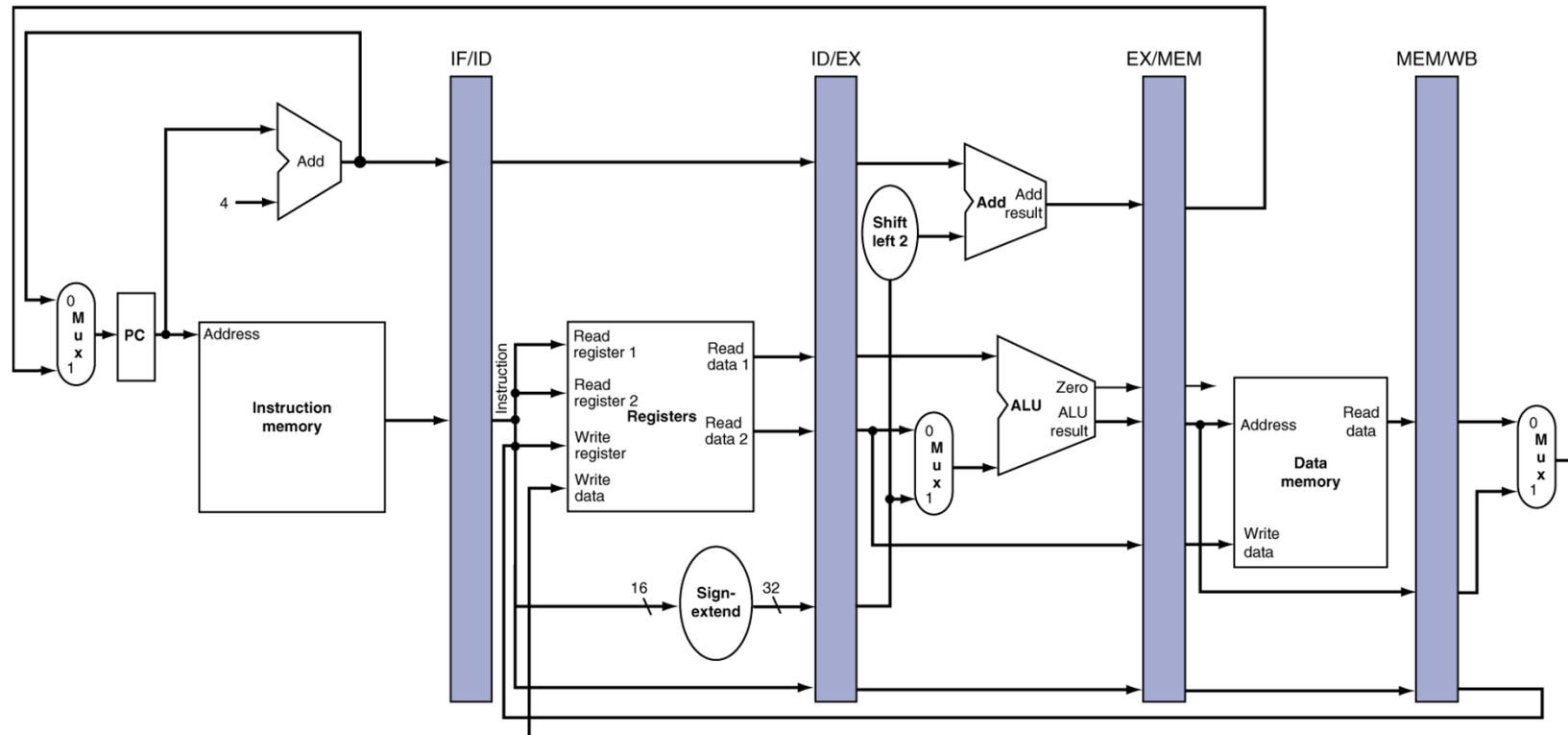
## ■ Traditional form



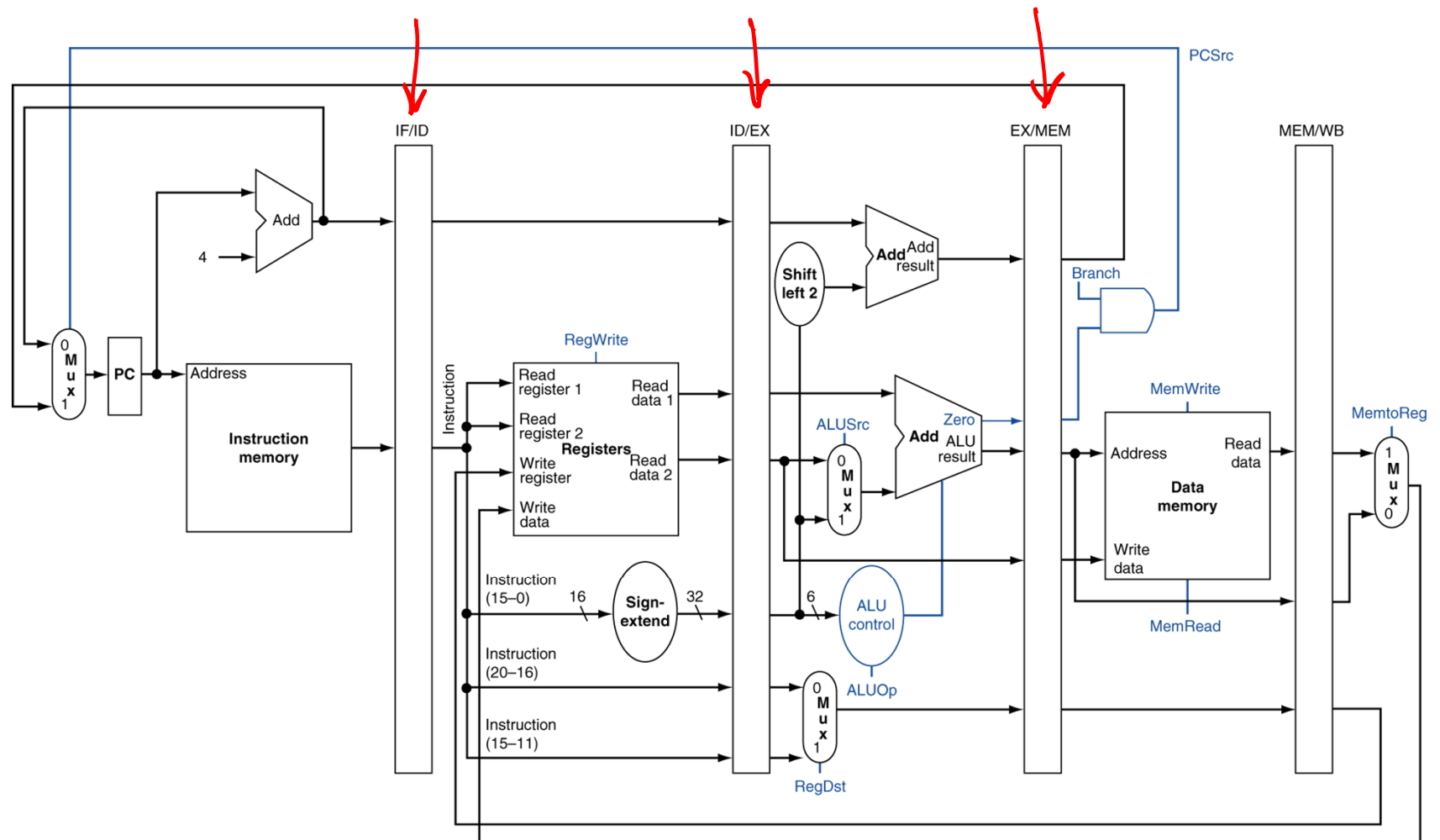
# Single-Cycle Pipeline Diagram

## ■ State of pipeline in a given cycle

add \$14, \$5, \$6	lw \$13, 24 (\$1)	add \$12, \$3, \$4	sub \$11, \$2, \$3	lw \$10, 20(\$1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back

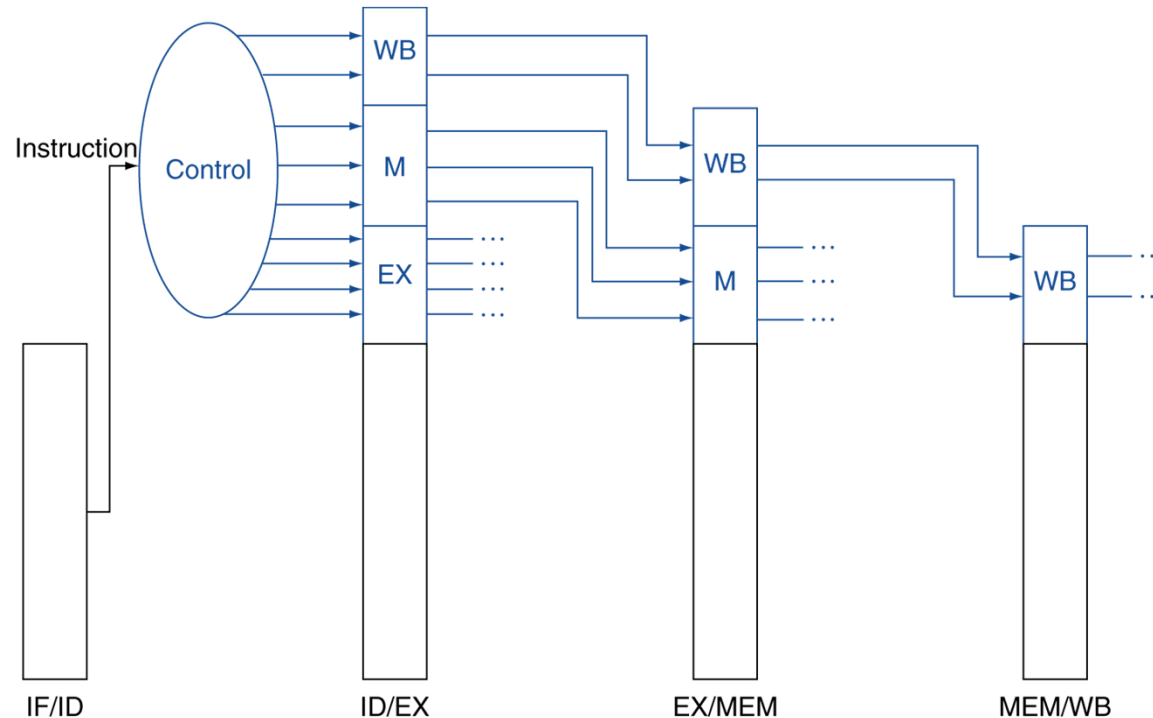


# Pipelined Control (Simplified)

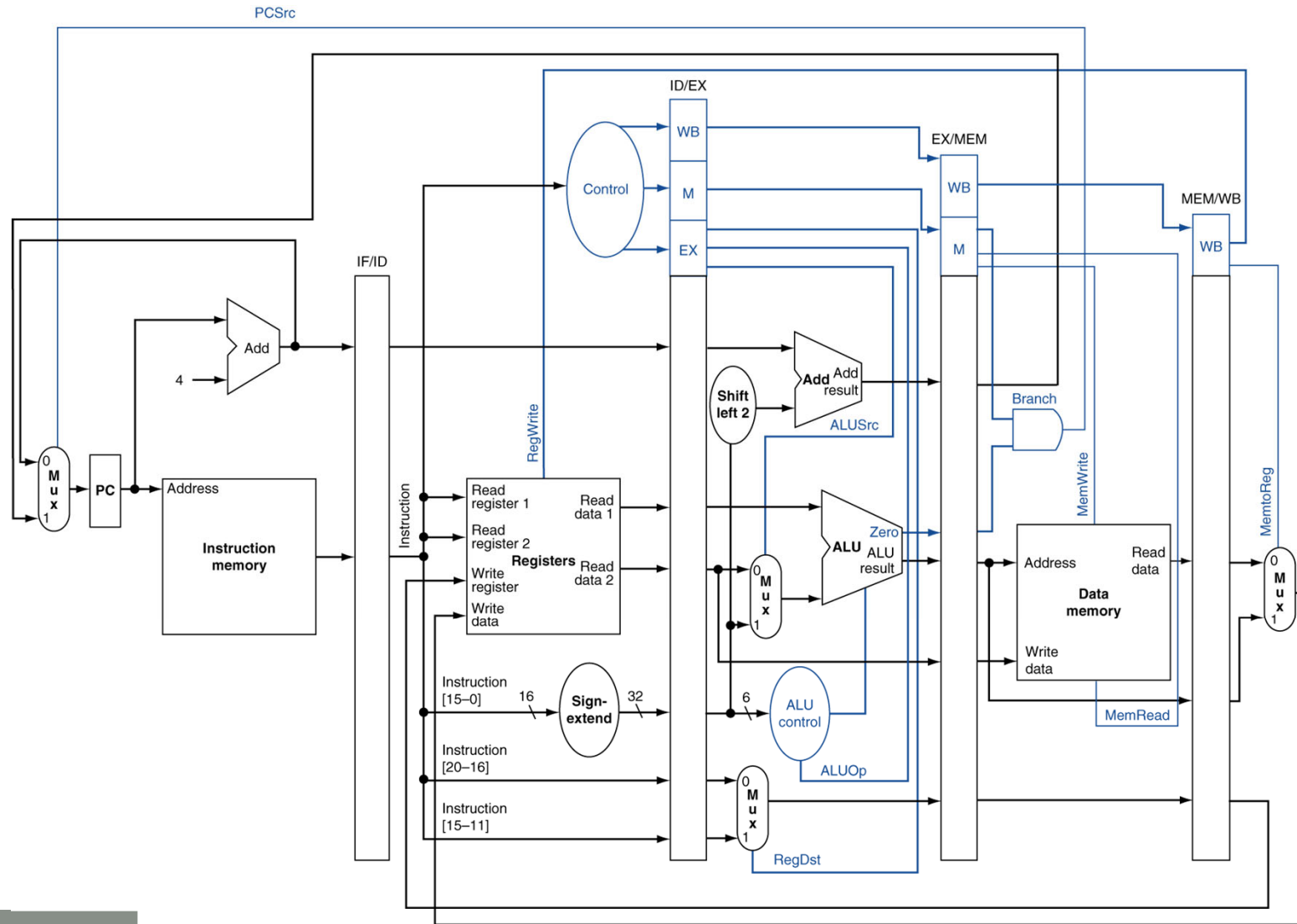


# Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation



# Pipelined Control



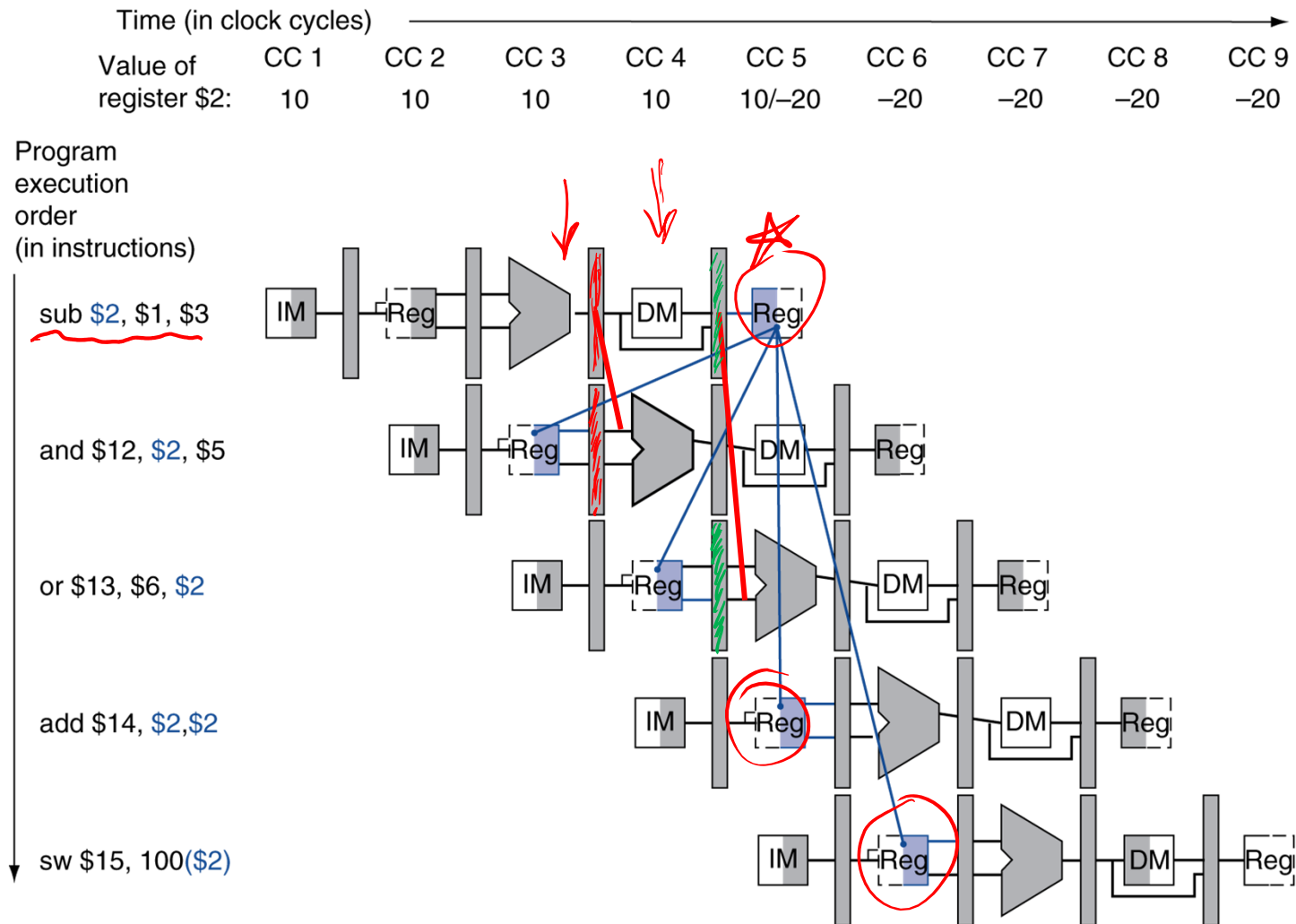
# Data Hazards in ALU Instructions

- Consider this sequence:

```
sub $2, $1, $3  
and $12, $2, $5  
or  $13, $6, $2  
add $14, $2, $2  
sw  $15, 100($2)
```

- We can resolve hazards with forwarding
  - How do we detect when to forward?

# Dependencies & Forwarding



# Detecting the Need to Forward

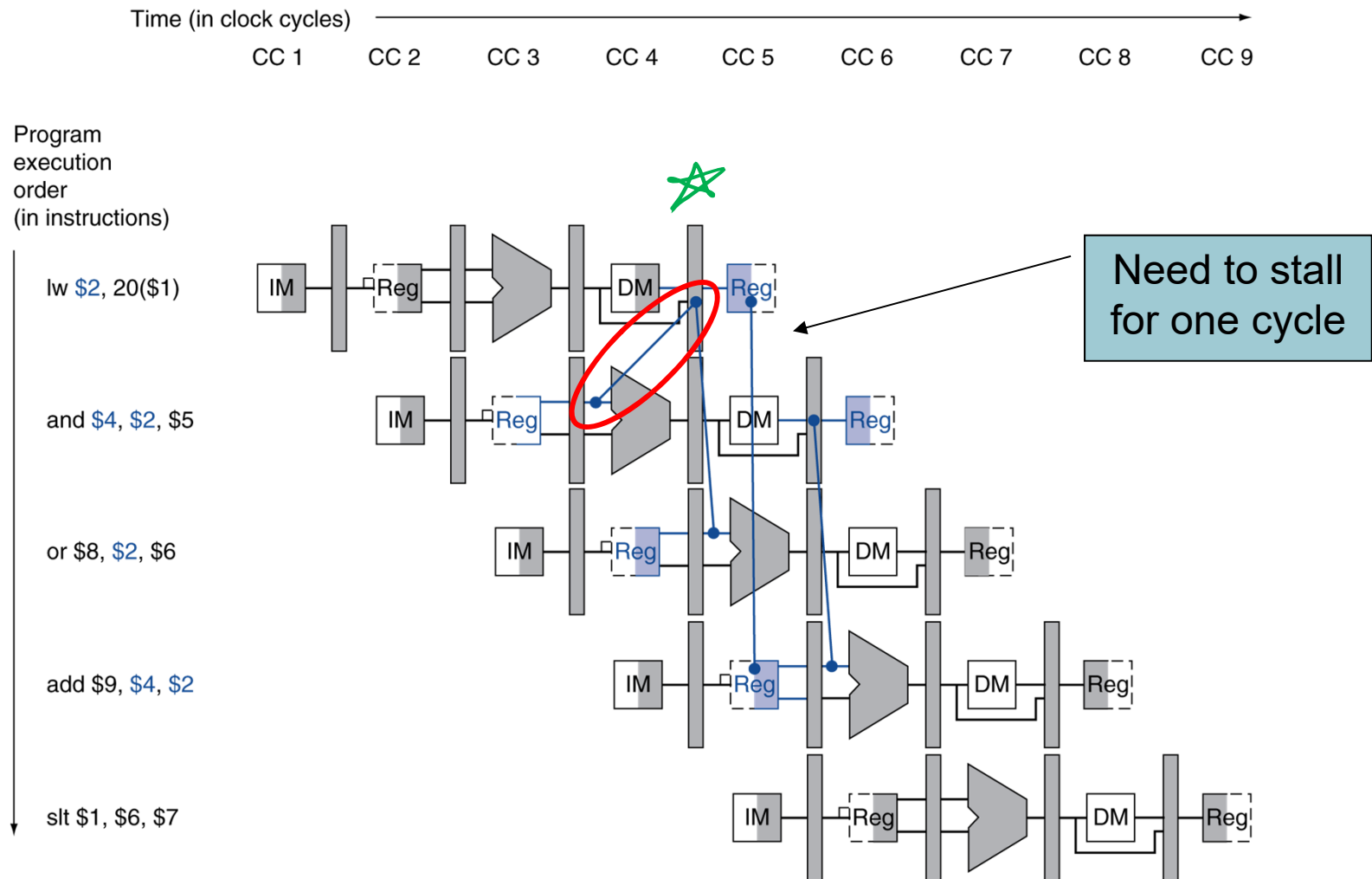
- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Data hazards when
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from  
EX/MEM  
pipeline reg

Fwd from  
MEM/WB  
pipeline reg



# Load-Use Data Hazard



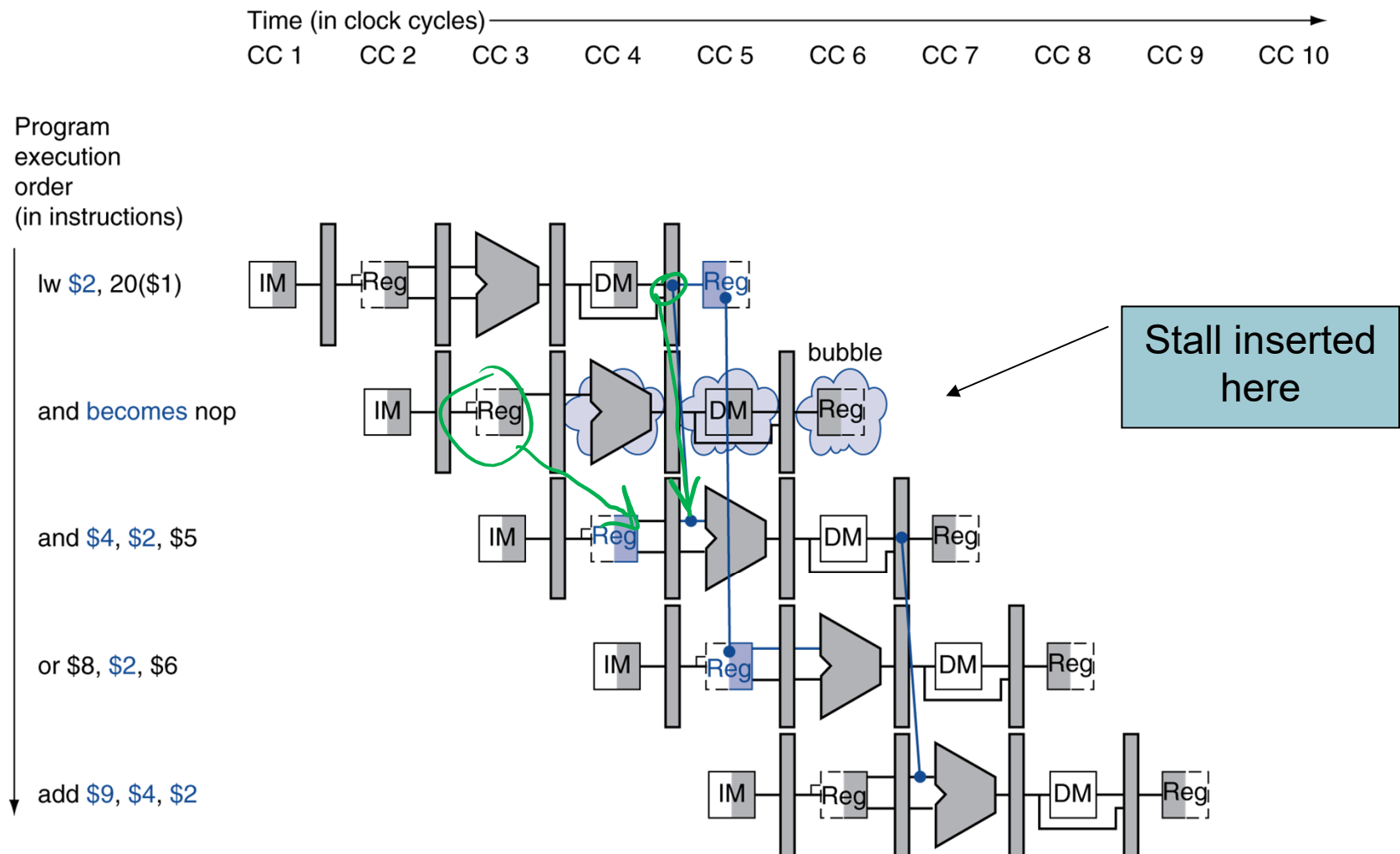
# Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and  
((ID/EX.RegisterRt = IF/ID.RegisterRs) or  
(ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

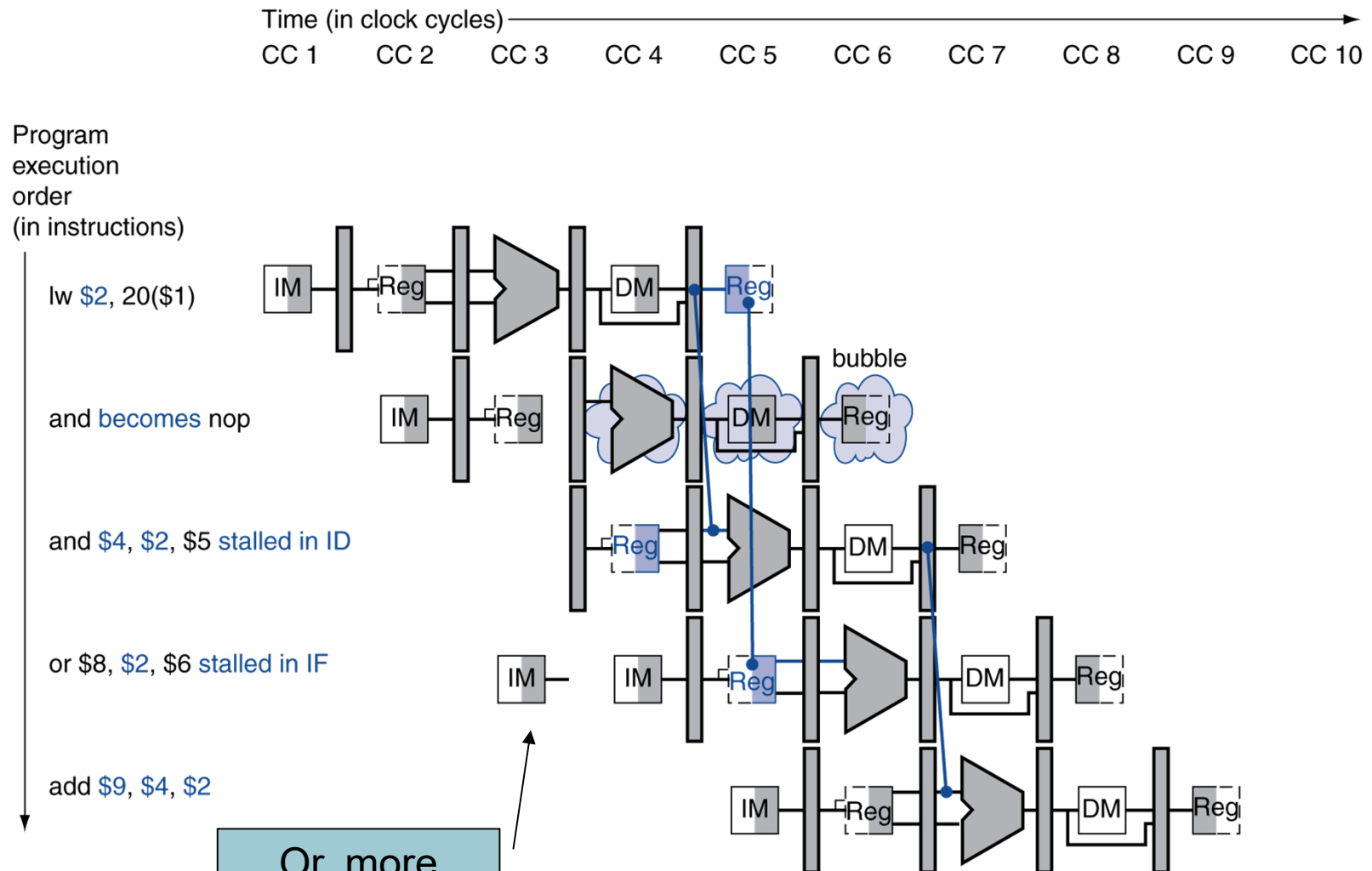
# How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for 1w
    - Can subsequently forward to EX stage

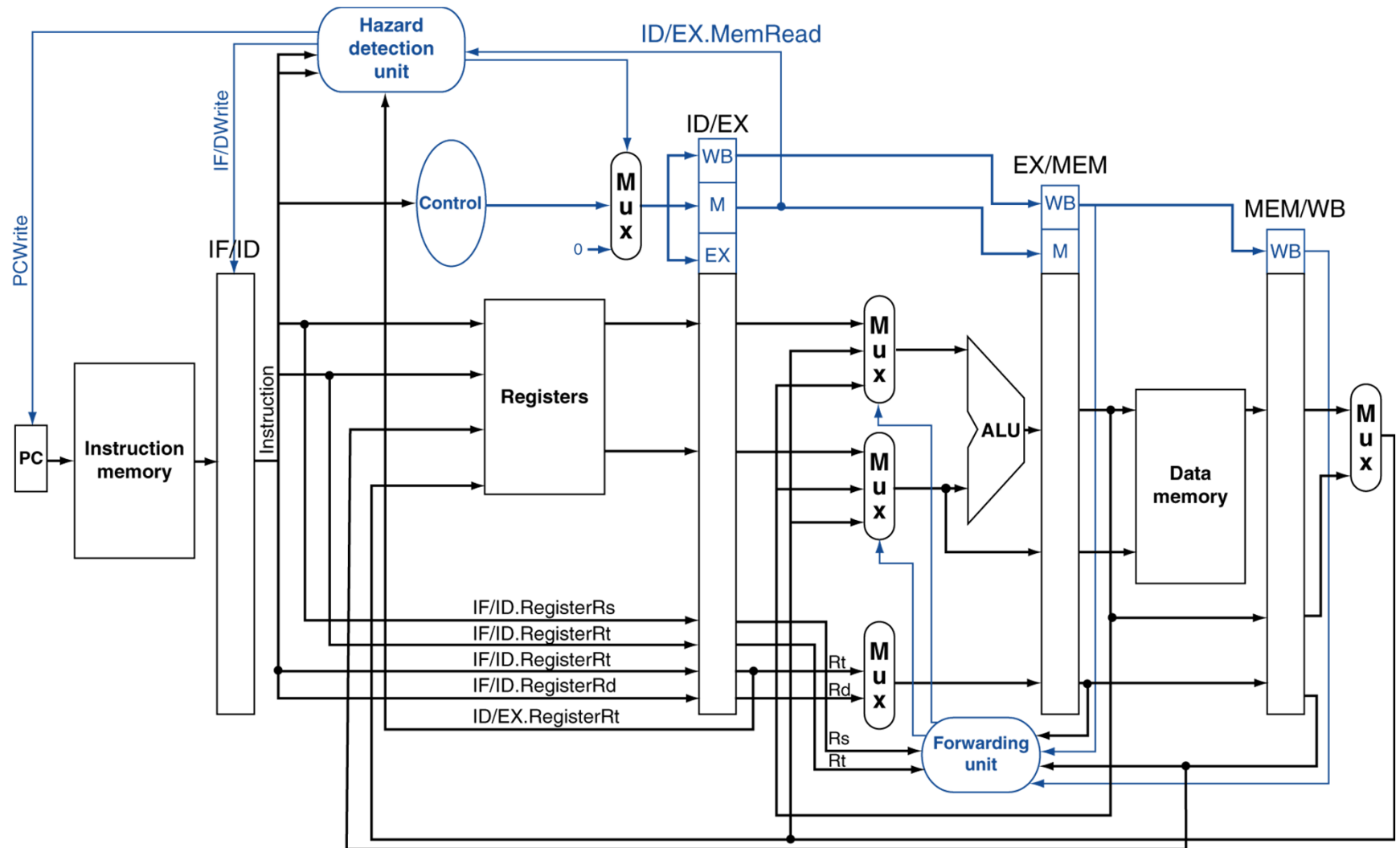
# Stall/Bubble in the Pipeline



# Stall/Bubble in the Pipeline



# Datapath with Hazard Detection



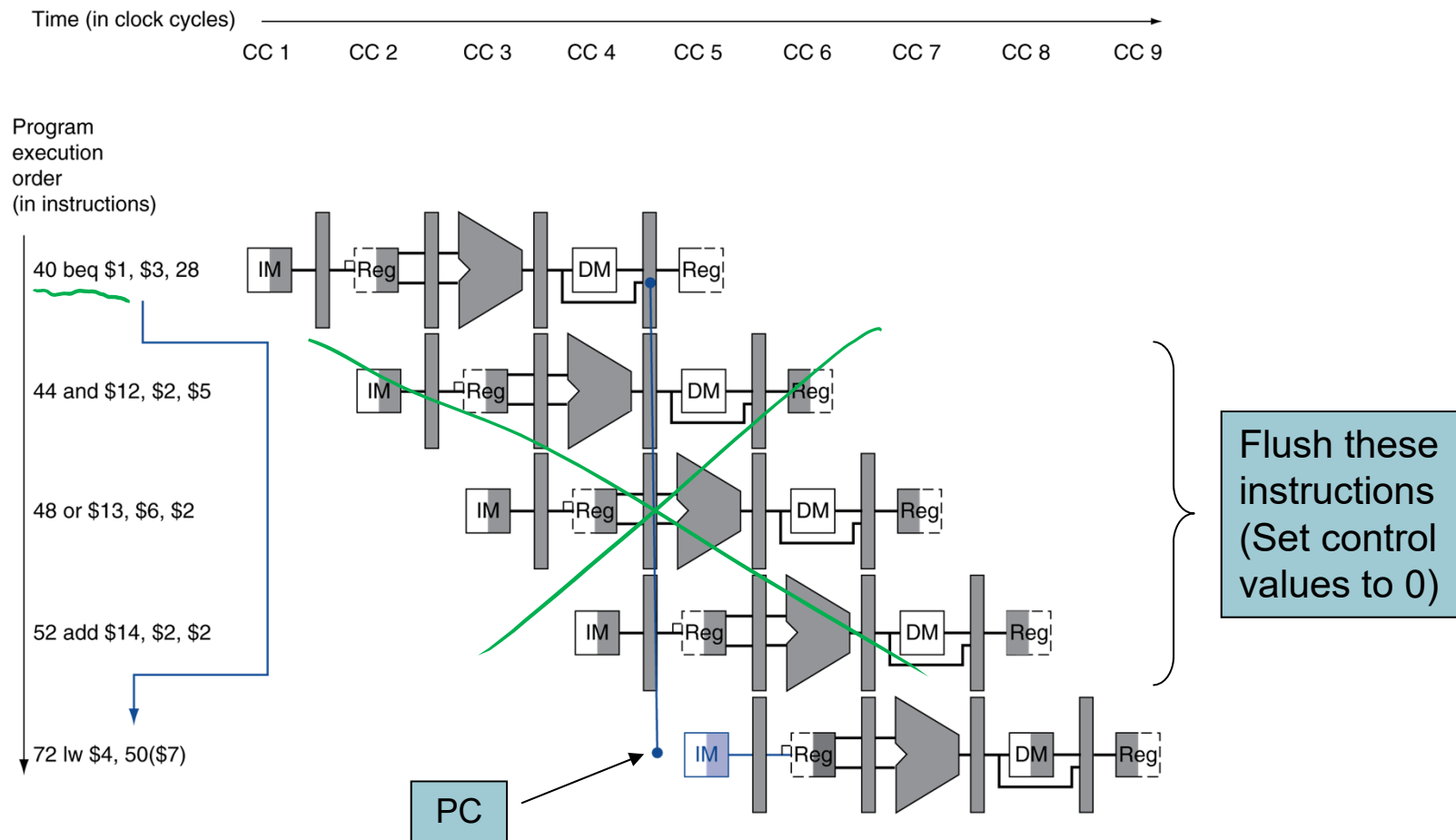
# Stalls and Performance

## The BIG Picture

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

# Branch Hazards

- If branch outcome determined in MEM



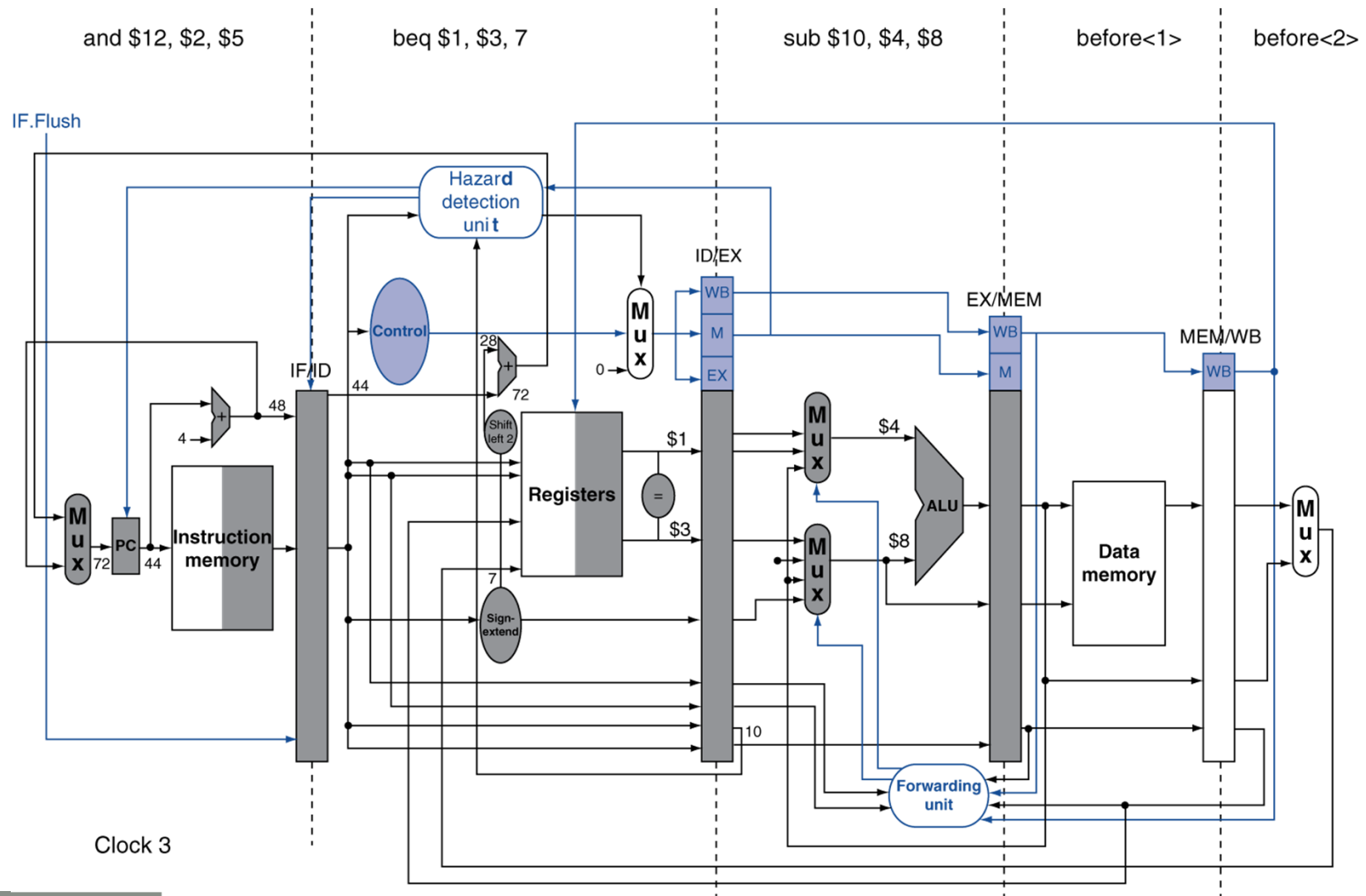


# Reducing Branch Delay

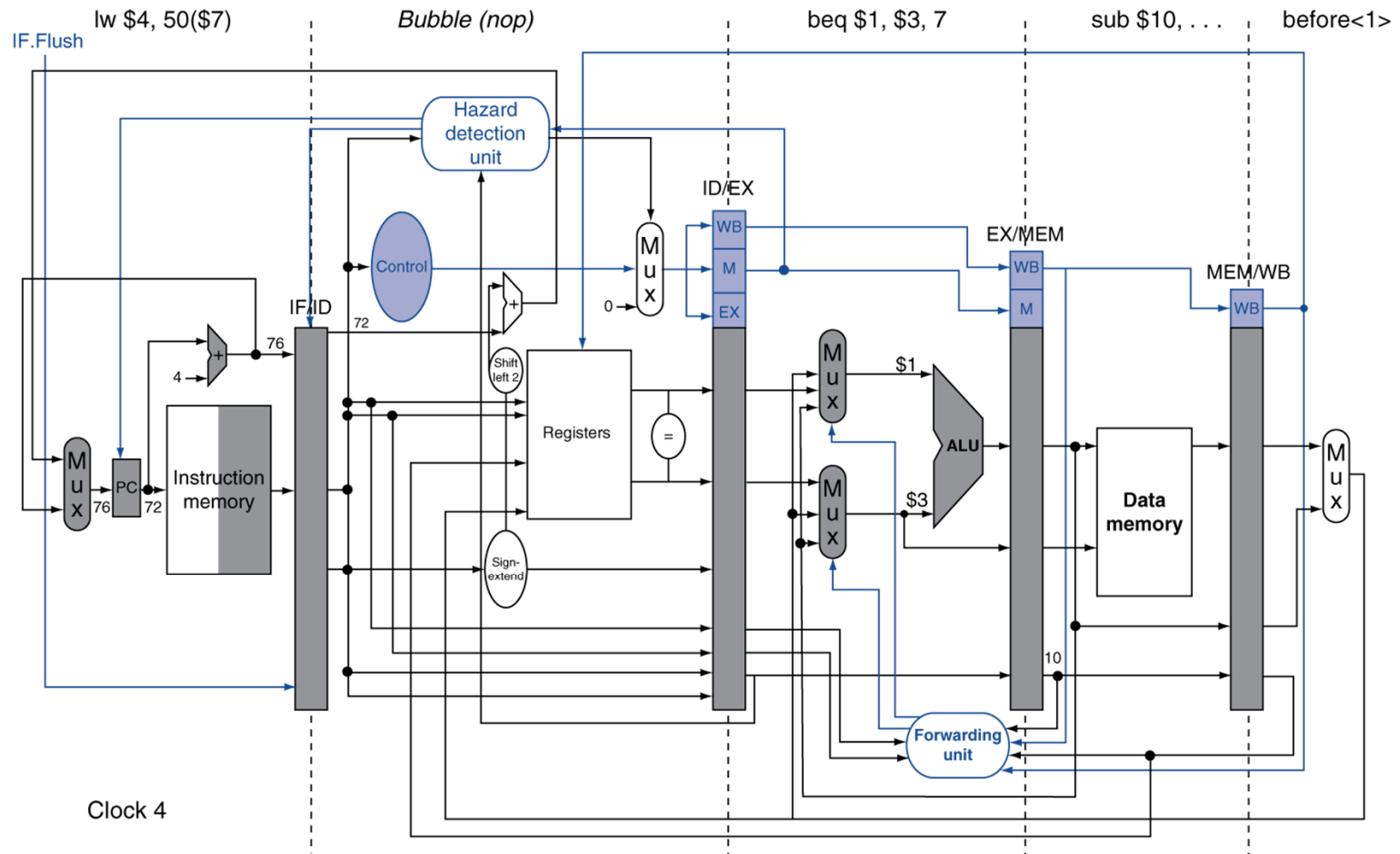
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub    $10, $4, $8
40:  beq    $1,  $3, 7
44:  and    $12, $2, $5
48:  or     $13, $2, $6
52:  add    $14, $4, $2
56:  slt    $15, $6, $7
    ...
72:  lw     $4, 50($7)
```

# Example: Branch Taken

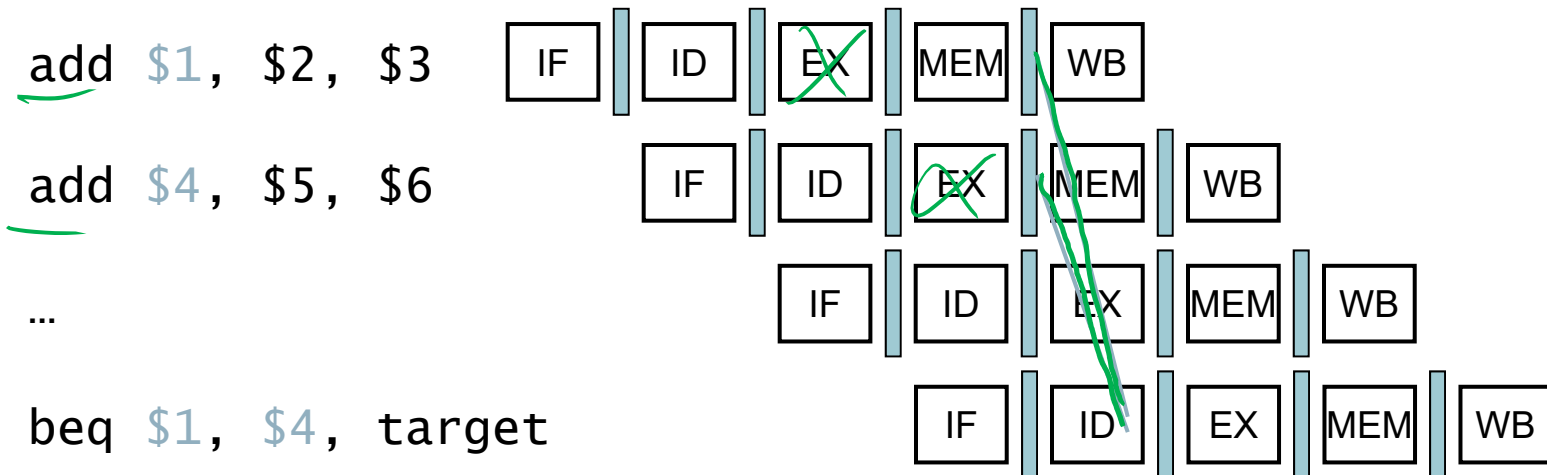


# Example: Branch Taken



# Data Hazards for Branches

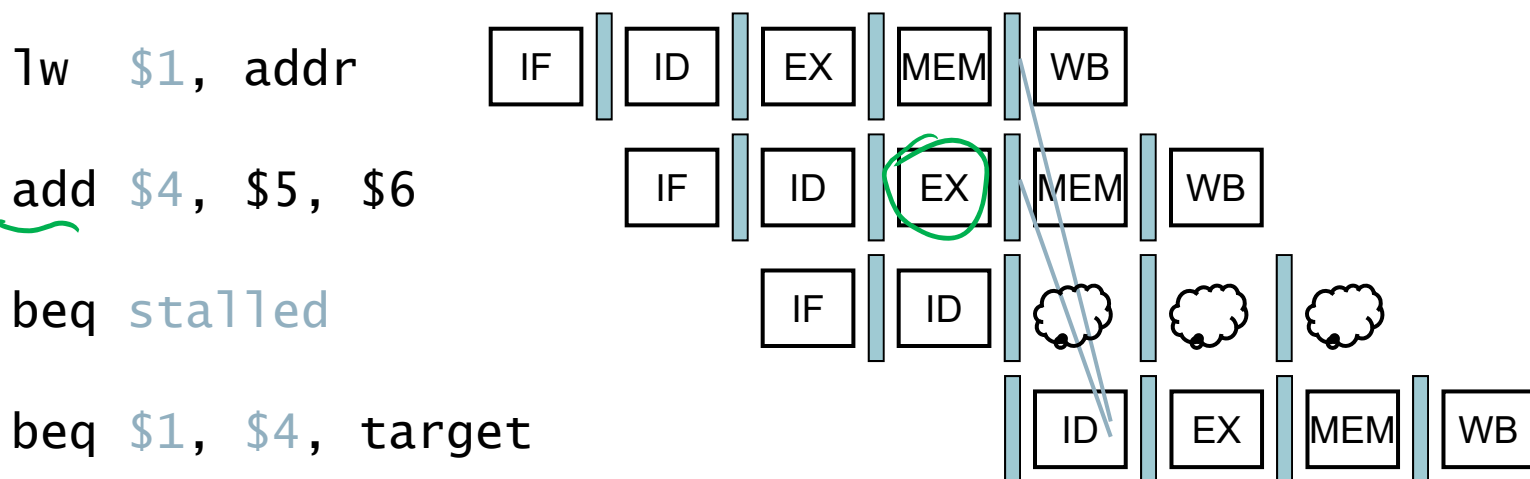
- If a comparison register is a destination of 2<sup>nd</sup> or 3<sup>rd</sup> preceding ALU instruction



- Can resolve using forwarding

# Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2<sup>nd</sup> preceding load instruction
  - Need 1 stall cycle



# Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

