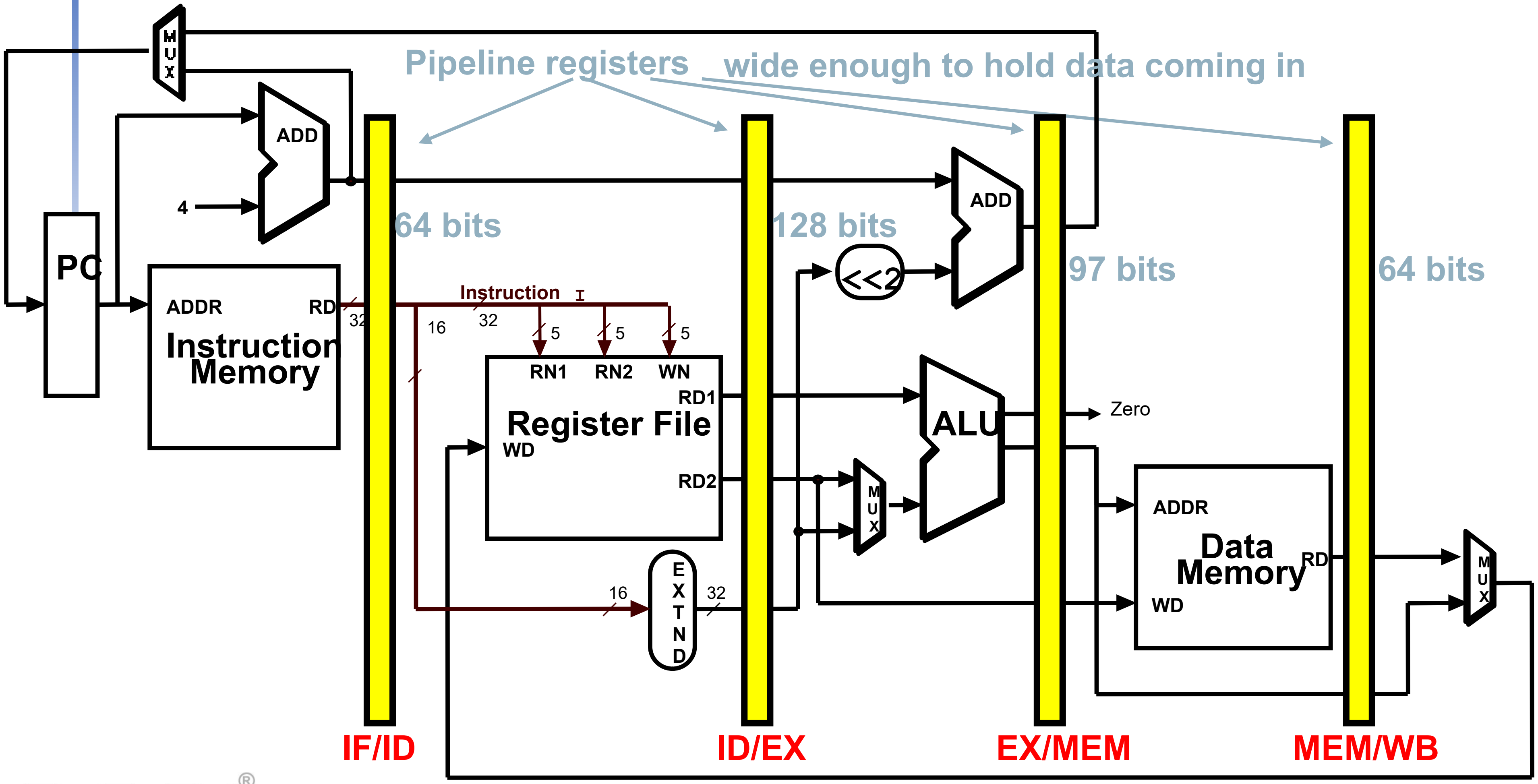
COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface

FOURTH 4 EDITION

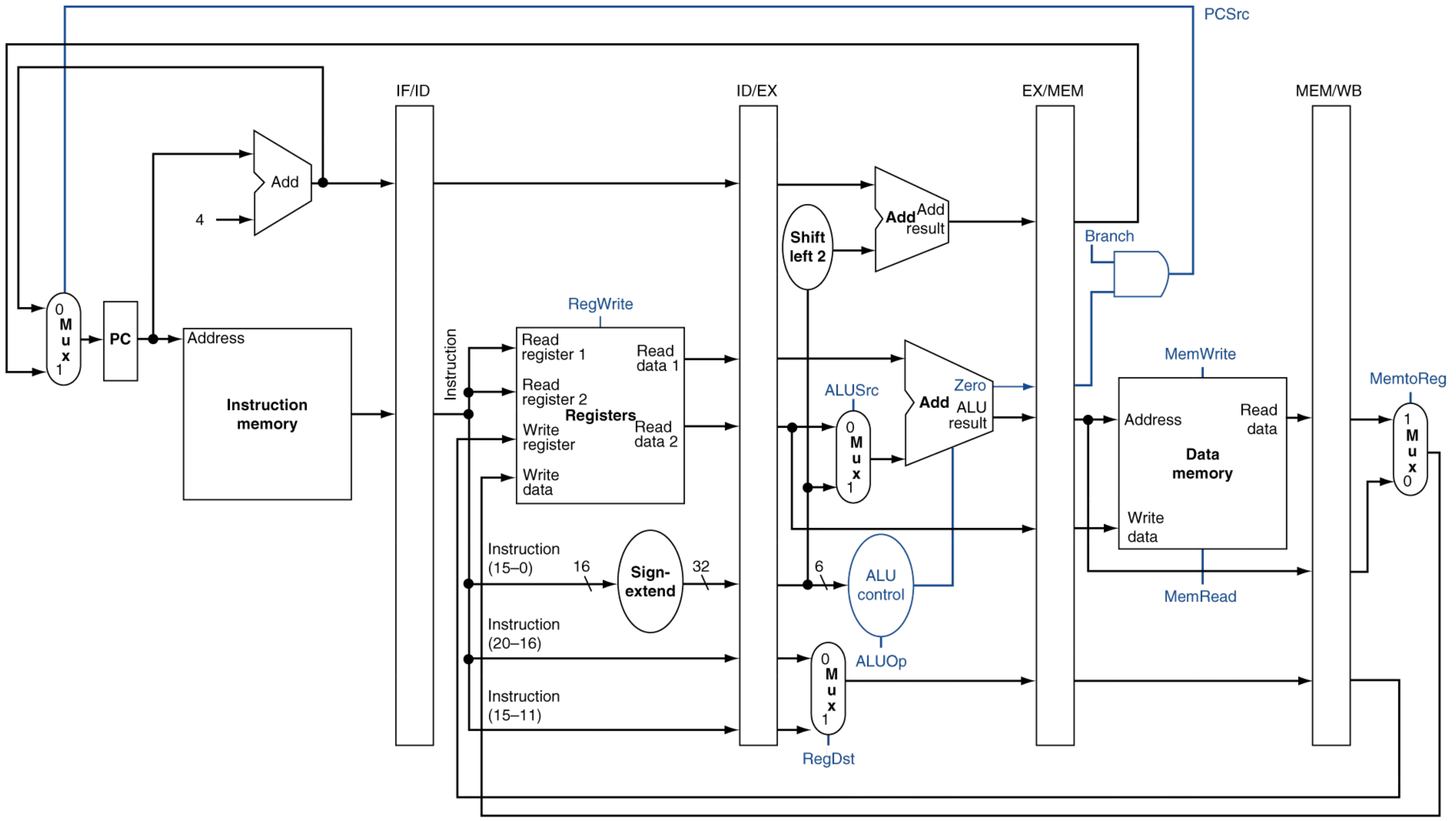
# Chapter 4

## Pipeline Datapath & Control

# Pipelined Datapath

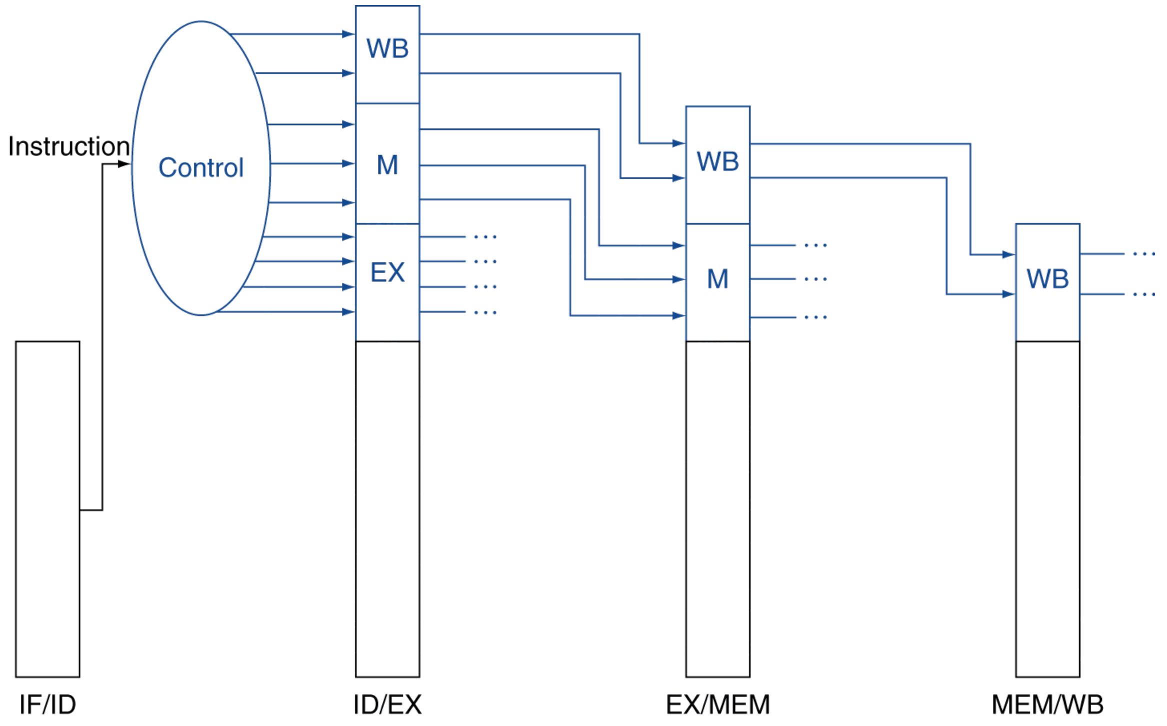


# Pipelined Control (Simplified)

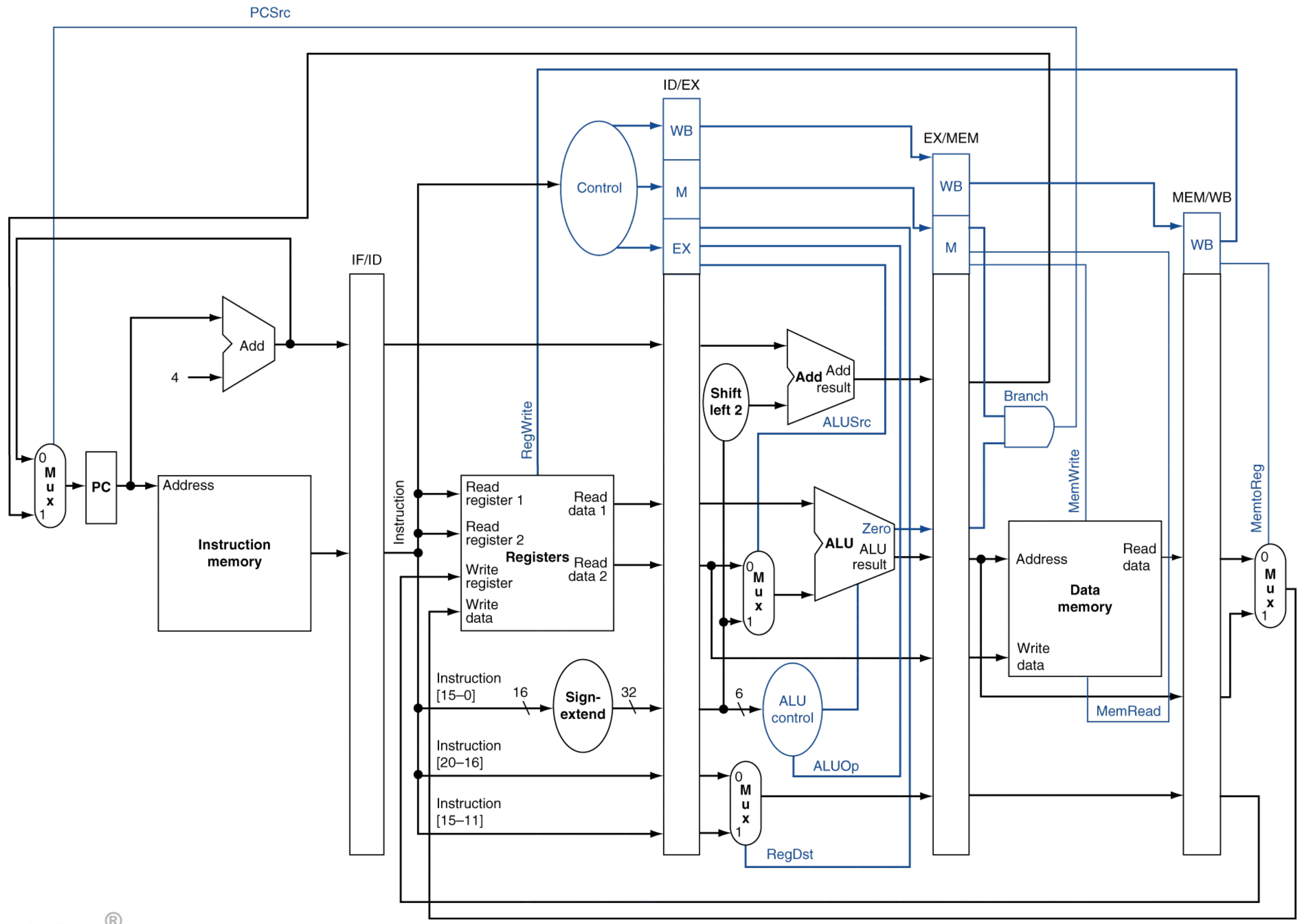


## Pipelined Control

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



# Pipelined Control



## 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

## Data Hazard : Detailed Analysis

## Revisiting Hazards

- So far our datapath and control have ignored hazards
- We shall revisit *data hazards* and *control hazards* and enhance our datapath and control to handle them in *hardware*…

## Data Hazards in ALU Instructions

- Consider this sequence:

```
I1: sub $s2, $s1,$s3
I2: and $t8,$s2,$s5
I3: or  $s7,$s6,$s2
I4: add $s8,$s2,$s2
I5: sw  $t9,100($s2)
```

## Hardware Solution: Forwarding

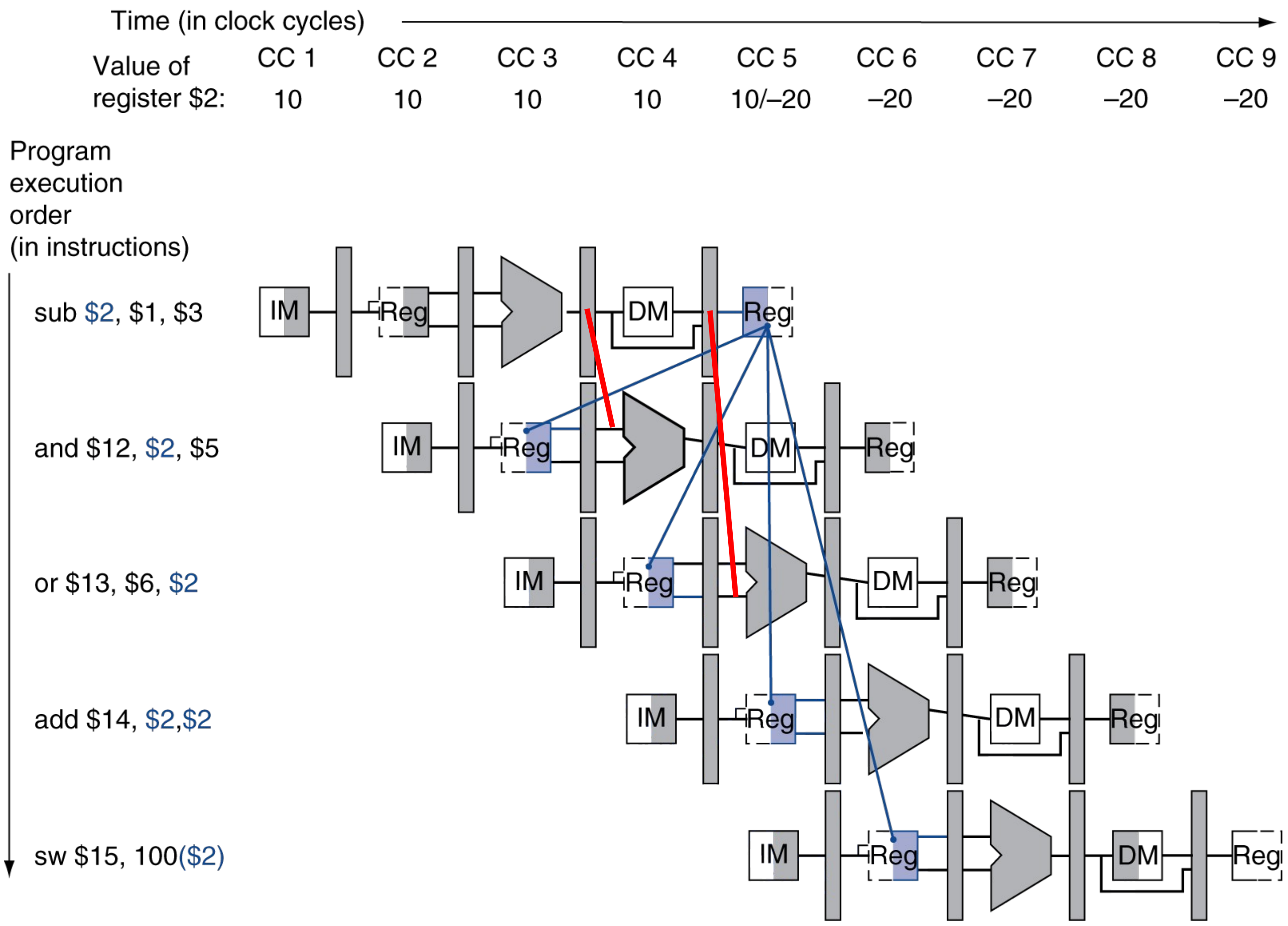
Idea: *use intermediate data*, do not wait for result to be finally written to the destination register. Two steps:

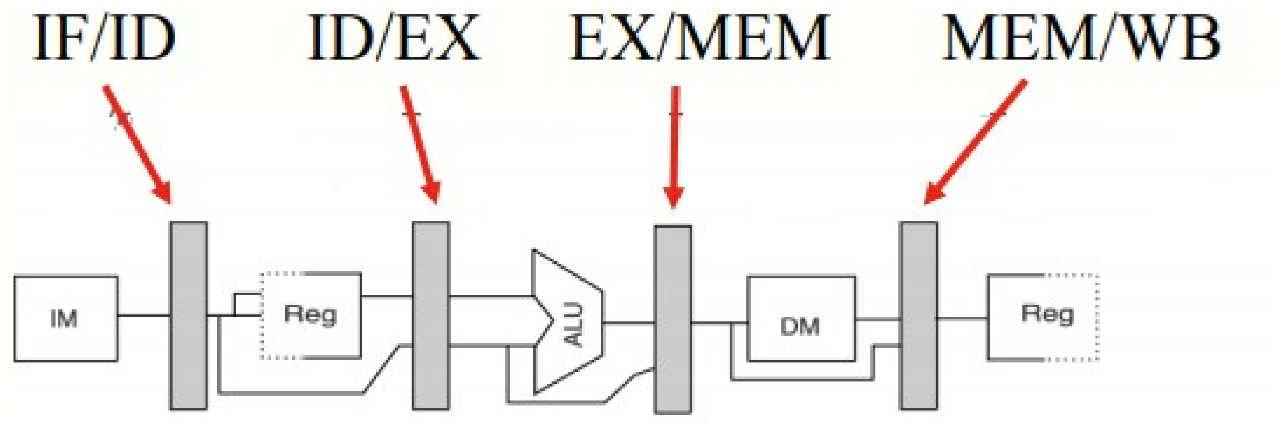
1. *Detect* data hazard
2. *Forward* intermediate data to resolve hazard

We can resolve hazards with forwarding

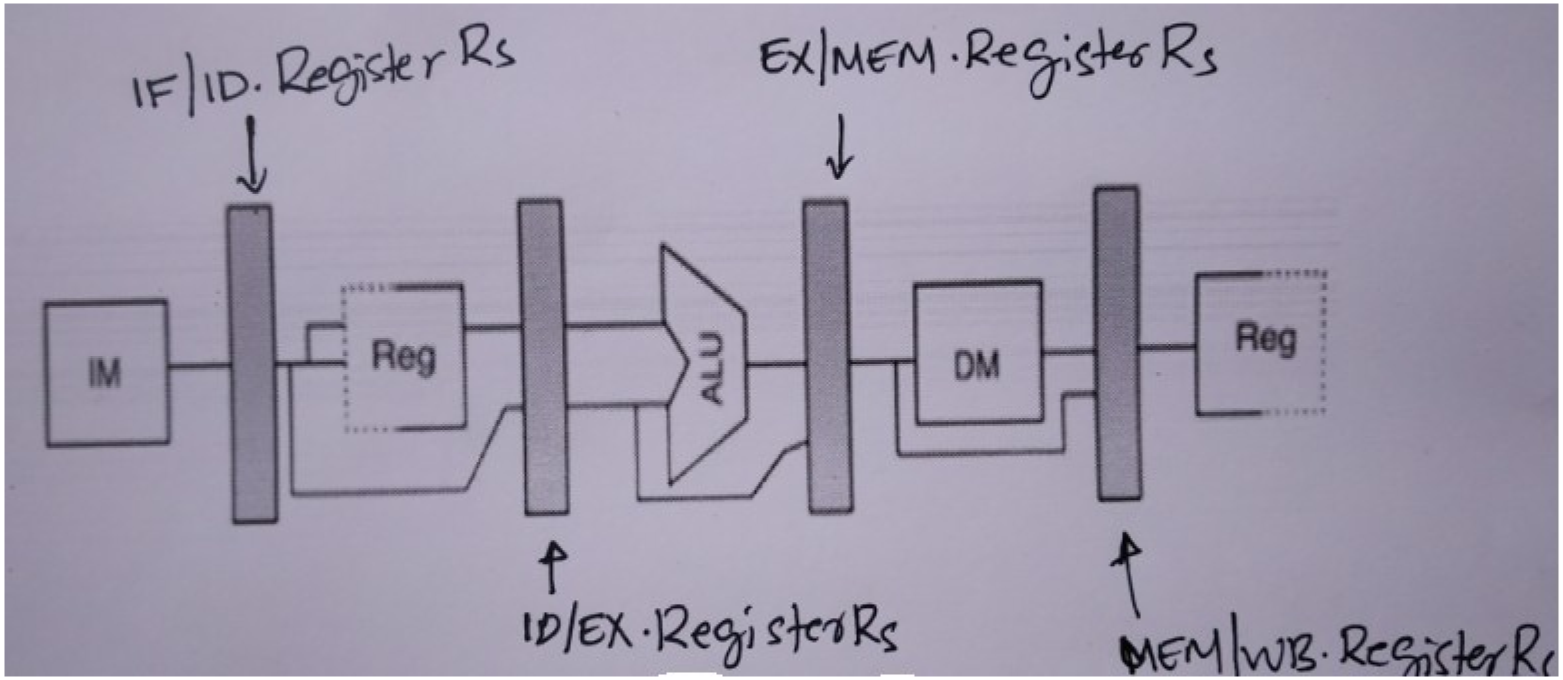
- How do we detect when to forward?

# Dependencies & Forwarding




IF/ID
ID/EX
EX/MEM
MEM/WB
IM
Reg
ALU
DM
Reg

# Register notation in pipeline



## Workout Slide

- Consider this sequence:

```
I1: sub $s2, $s1,$s3
I2: and $t8,$s2,$s5
I3: or  $s7,$s6,$s2
I4: add $s8,$s2,$s2
I5: sw  $t9,100($s2)
```

# 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

Fwd from EX/MEM pipeline reg

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from MEM/WB pipeline reg

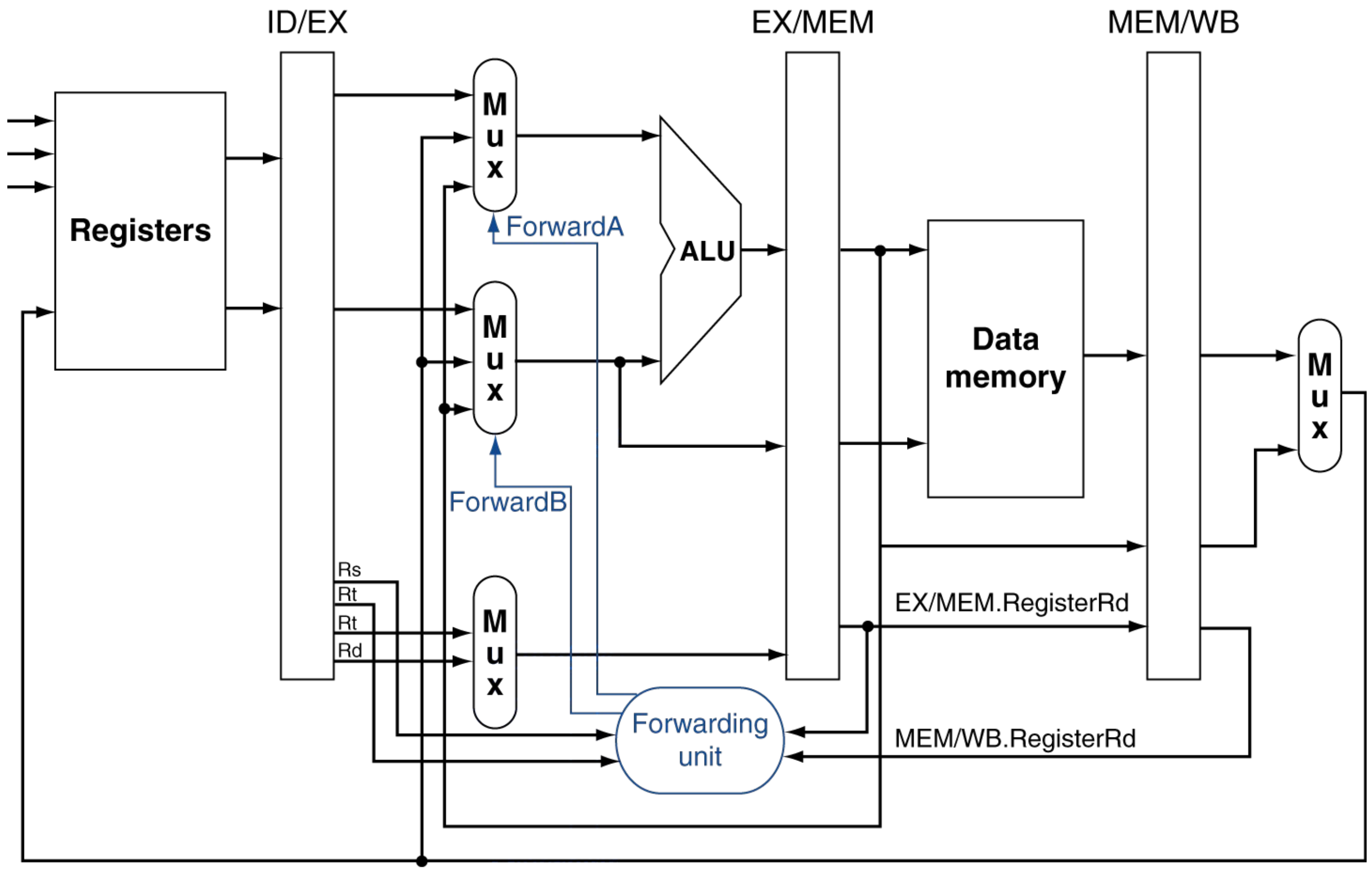
## Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not $zero
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

- 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

# Forwarding Paths



b. With forwarding

# Forwarding Hardware: Multiplexor Control

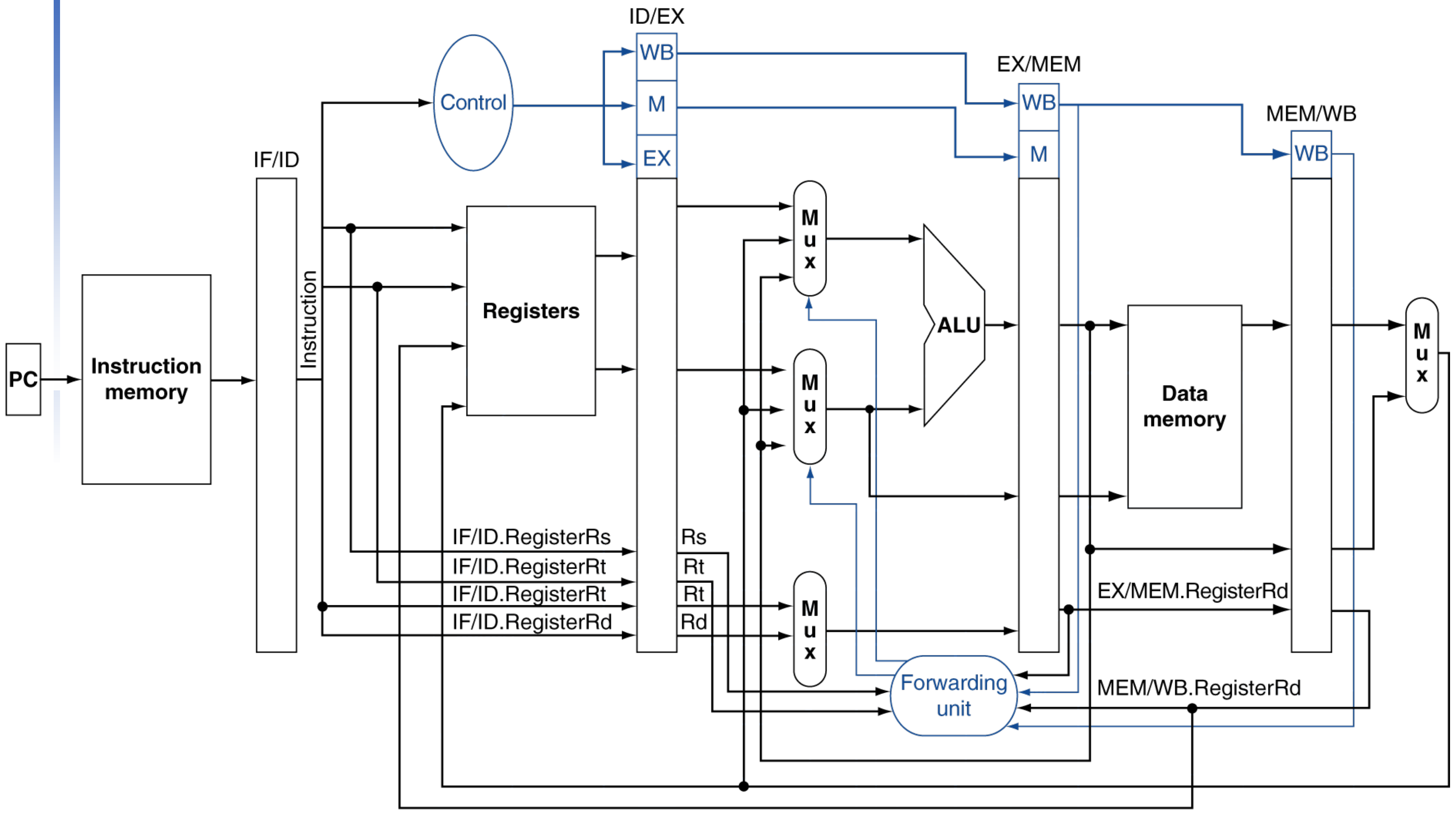
| Mux control | Source | Explanation |
|---|---|---|
| ForwardA = 00 | ID/EX | The first ALU operand comes from the register file |
| ForwardA = 10 | EX/MEM | The first ALU operand is forwarded from prior ALU result |
| ForwardA = 01 | MEM/WB | The first ALU operand is forwarded from data memory or an earlier ALU result |
| ForwardB = 00 | ID/EX | The second ALU operand comes from the register file |
| ForwardB = 10 | EX/MEM | The second ALU operand is forwarded from prior ALU result |
| ForwardB = 01 | MEM/WB | The second ALU operand is forwarded from data memory or an earlier ALU result |

Depending on the selection in the rightmost multiplexor (see datapath with control diagram)

## Forwarding Conditions

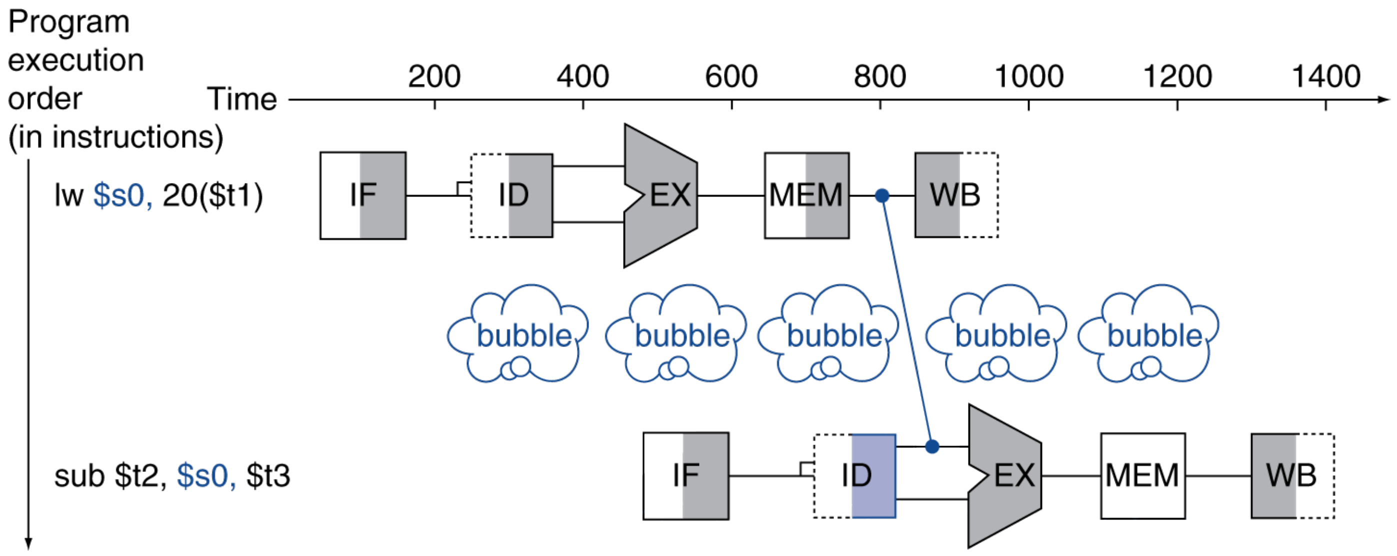
- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

## Datapath with Forwarding

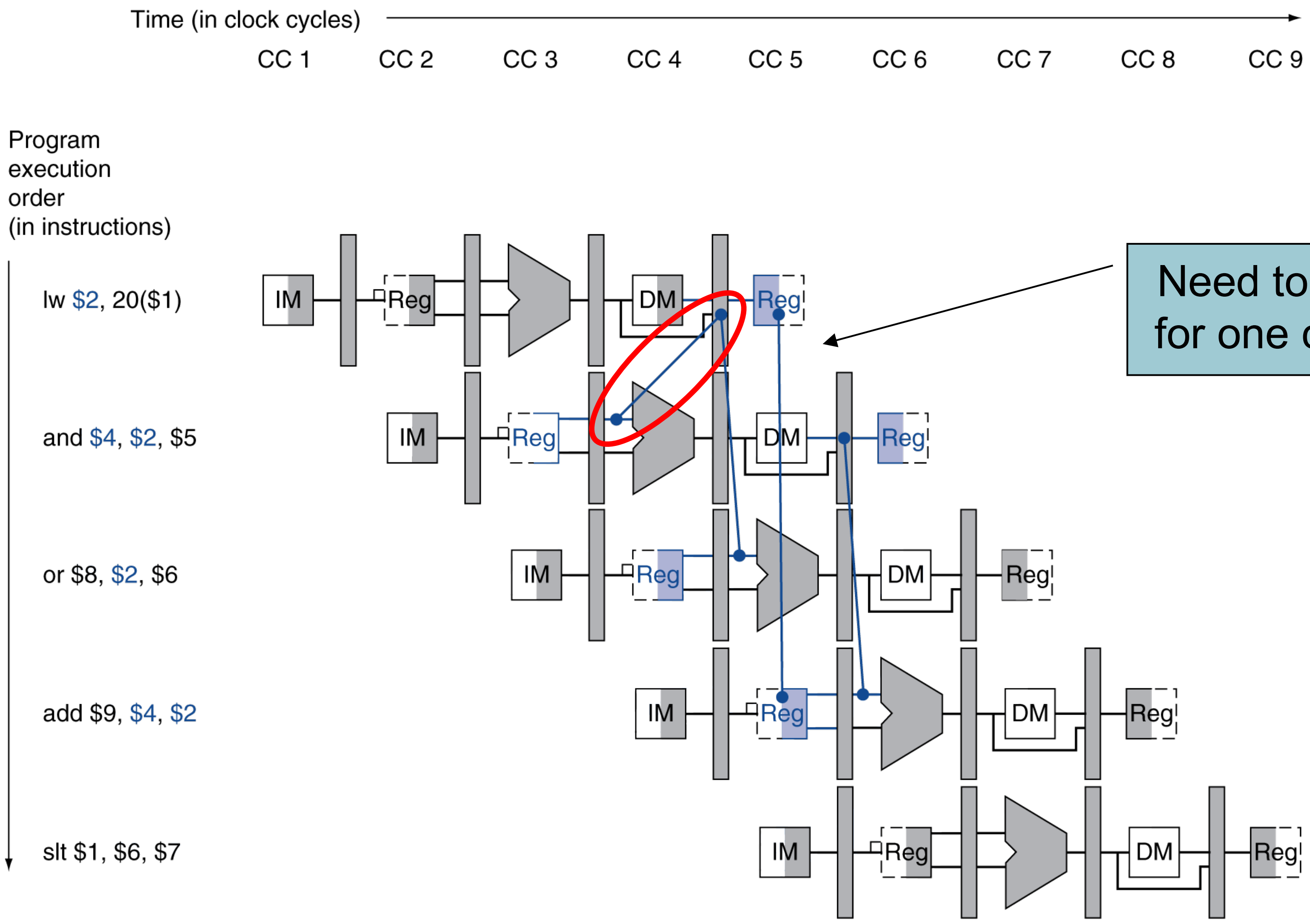


# Load-Use Data Hazard

- Can’t always avoid stalls by forwarding
  - If value not computed when needed
  - Can’t forward backward in time!

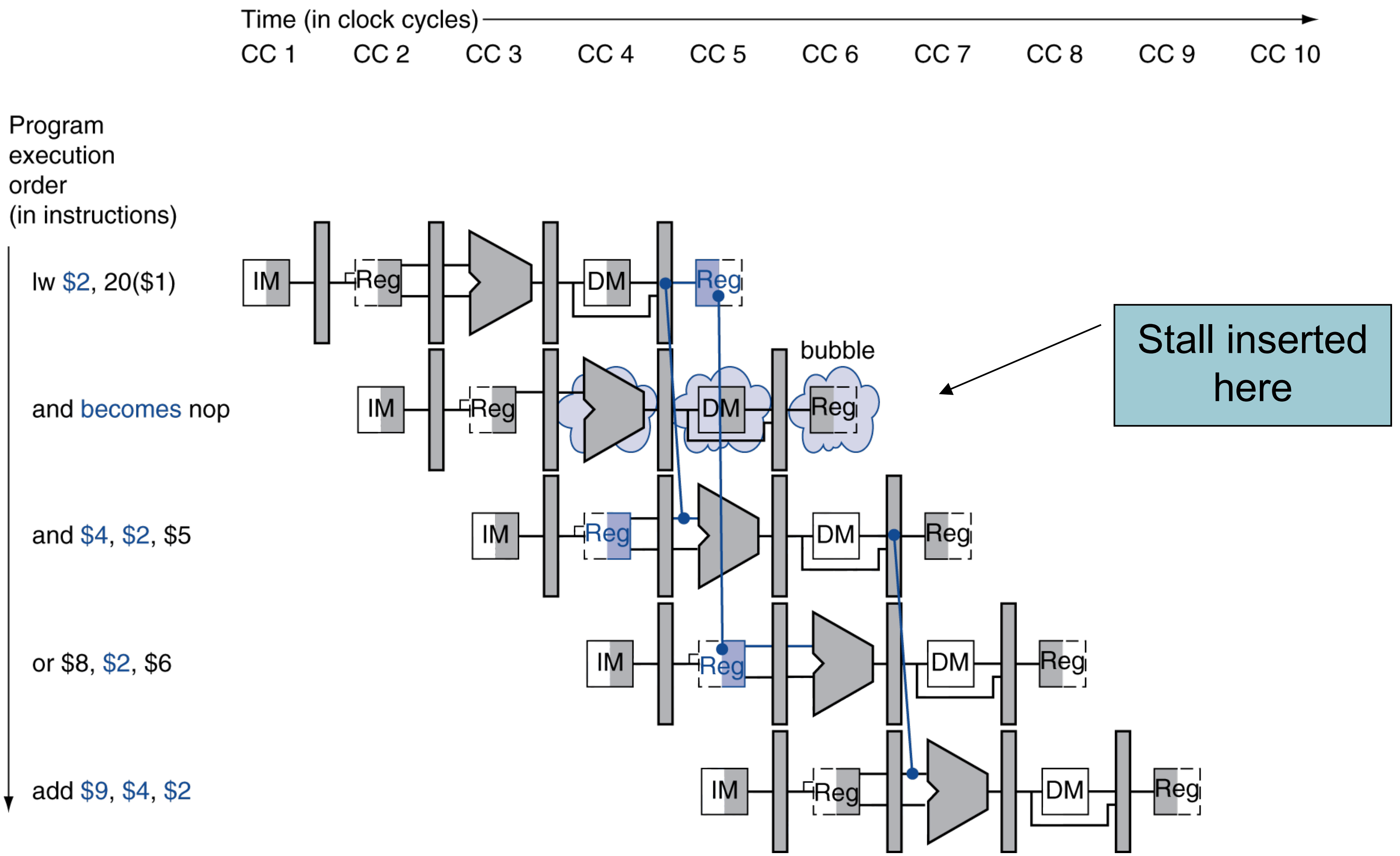


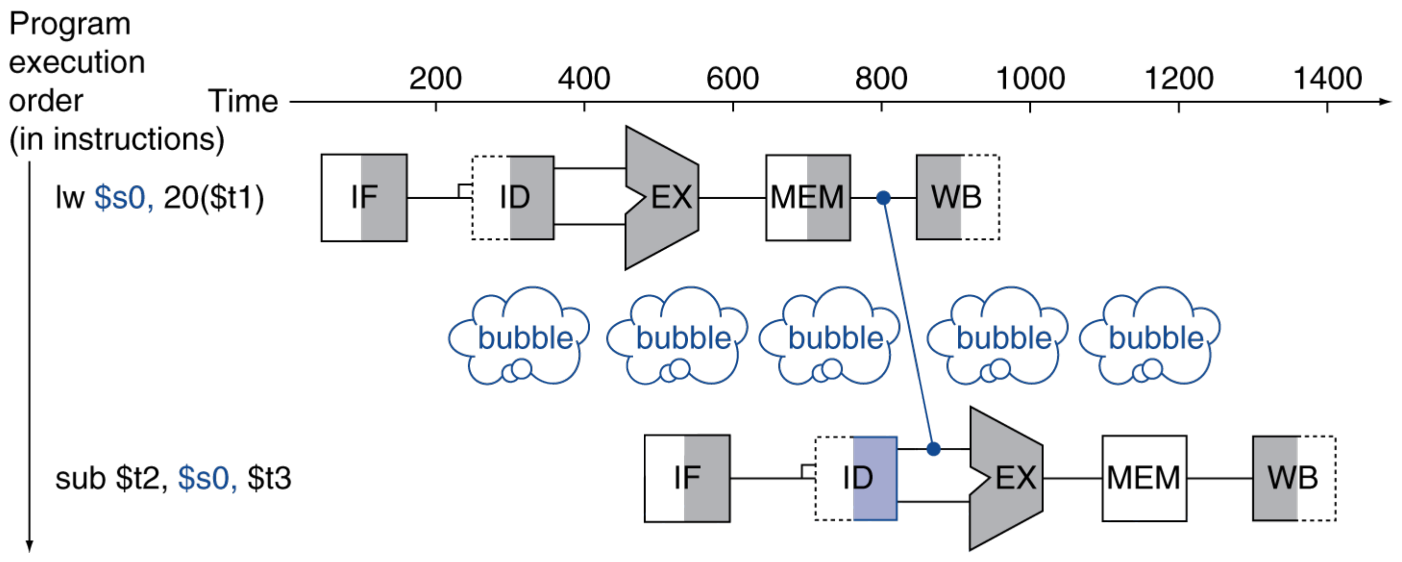
# Load-Use Data Hazard



Need to stall for one cycle

# Stall/Bubble in the Pipeline

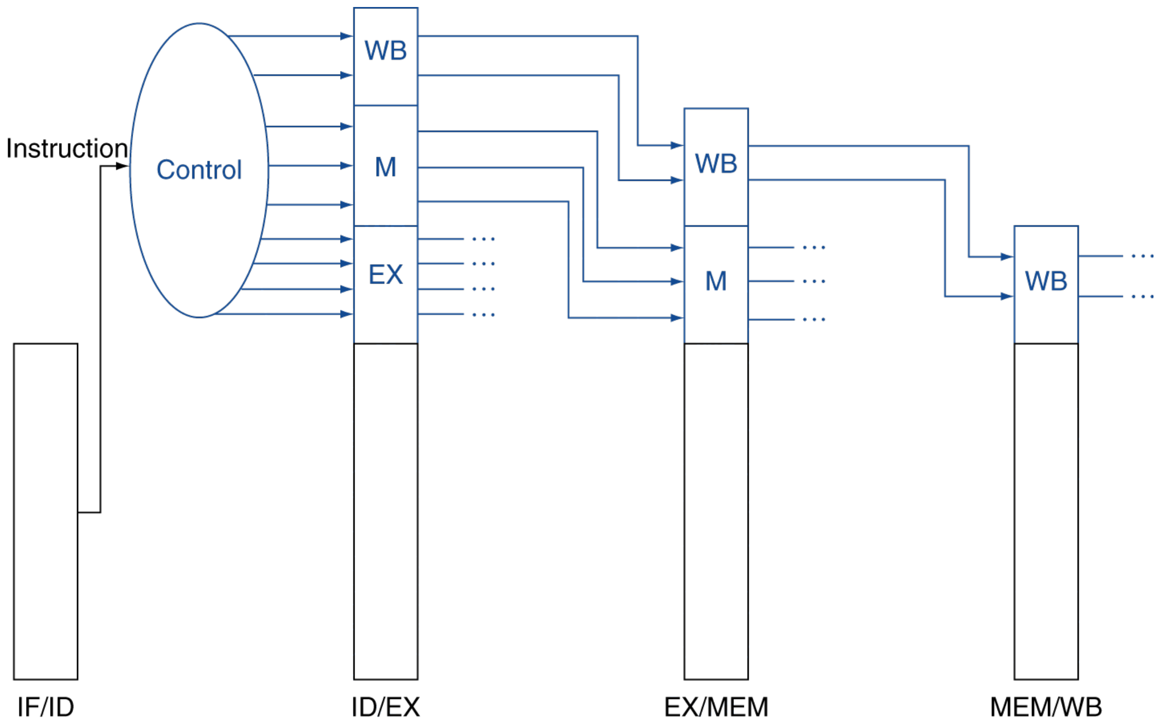



Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
lw $s0, 20($t1)
IF
ID
EX
MEM
WB
bubble
bubble
bubble
bubble
bubble
sub $t2, $s0, $t3
IF
ID
EX
MEM
WB

## 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))

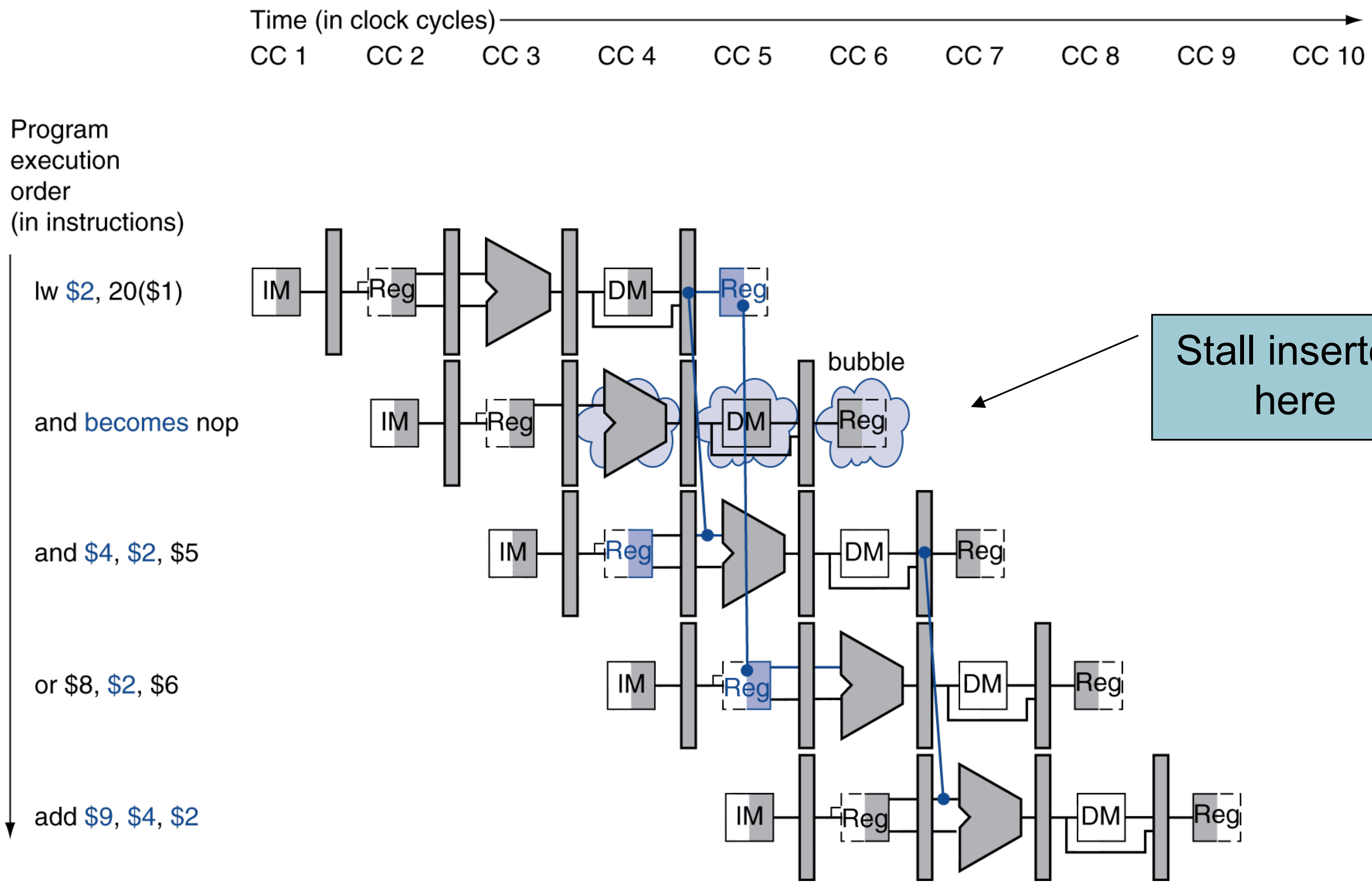
stall the pipeline


Instruction
Control
WB
M
EX
WB
M
WB
IF/ID
ID/EX
EX/MEM
MEM/WB

## How to Stall the Pipeline

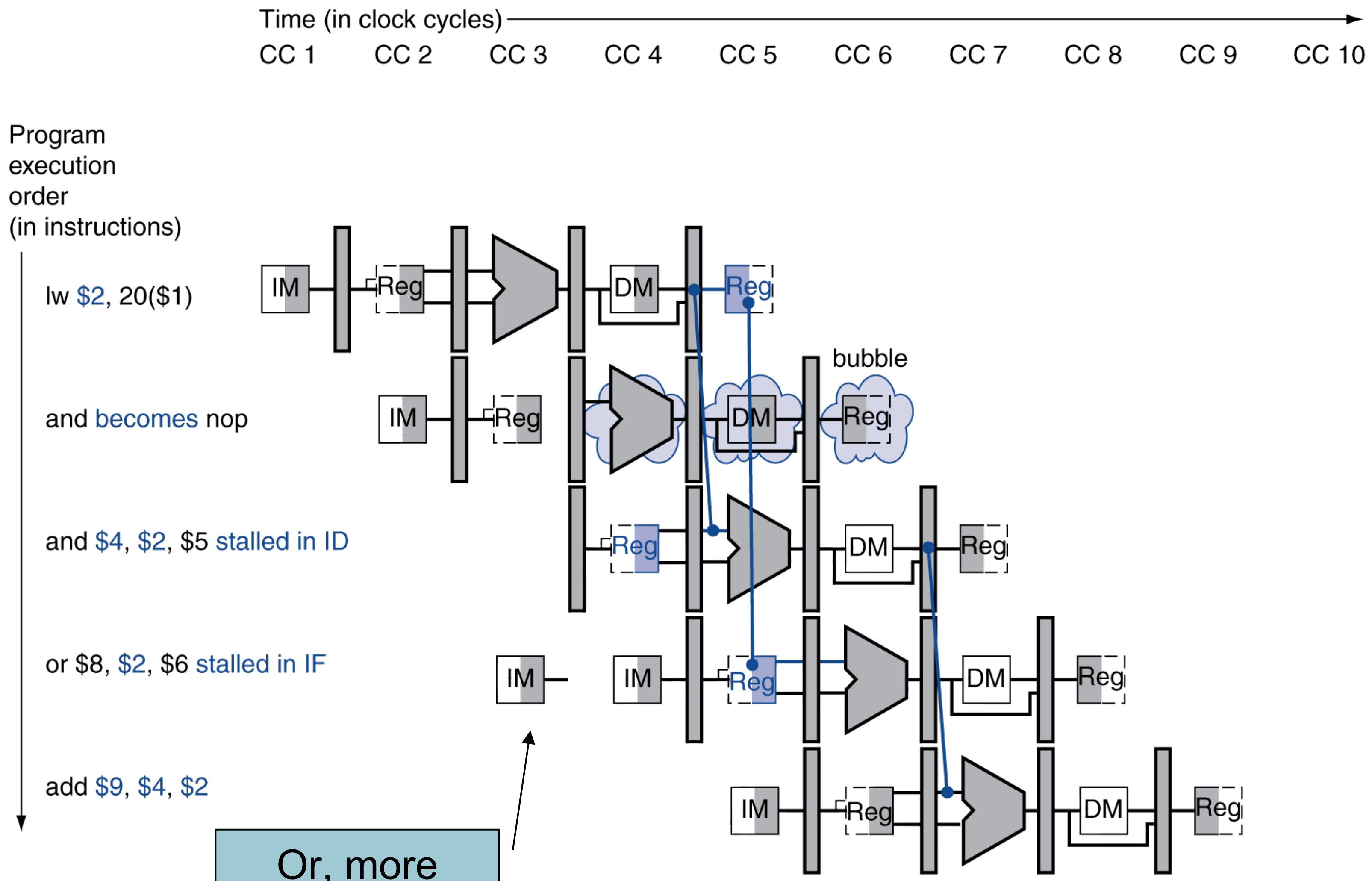
- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `lw`
    - Can subsequently forward to EX stage

# Stall/Bubble in the Pipeline



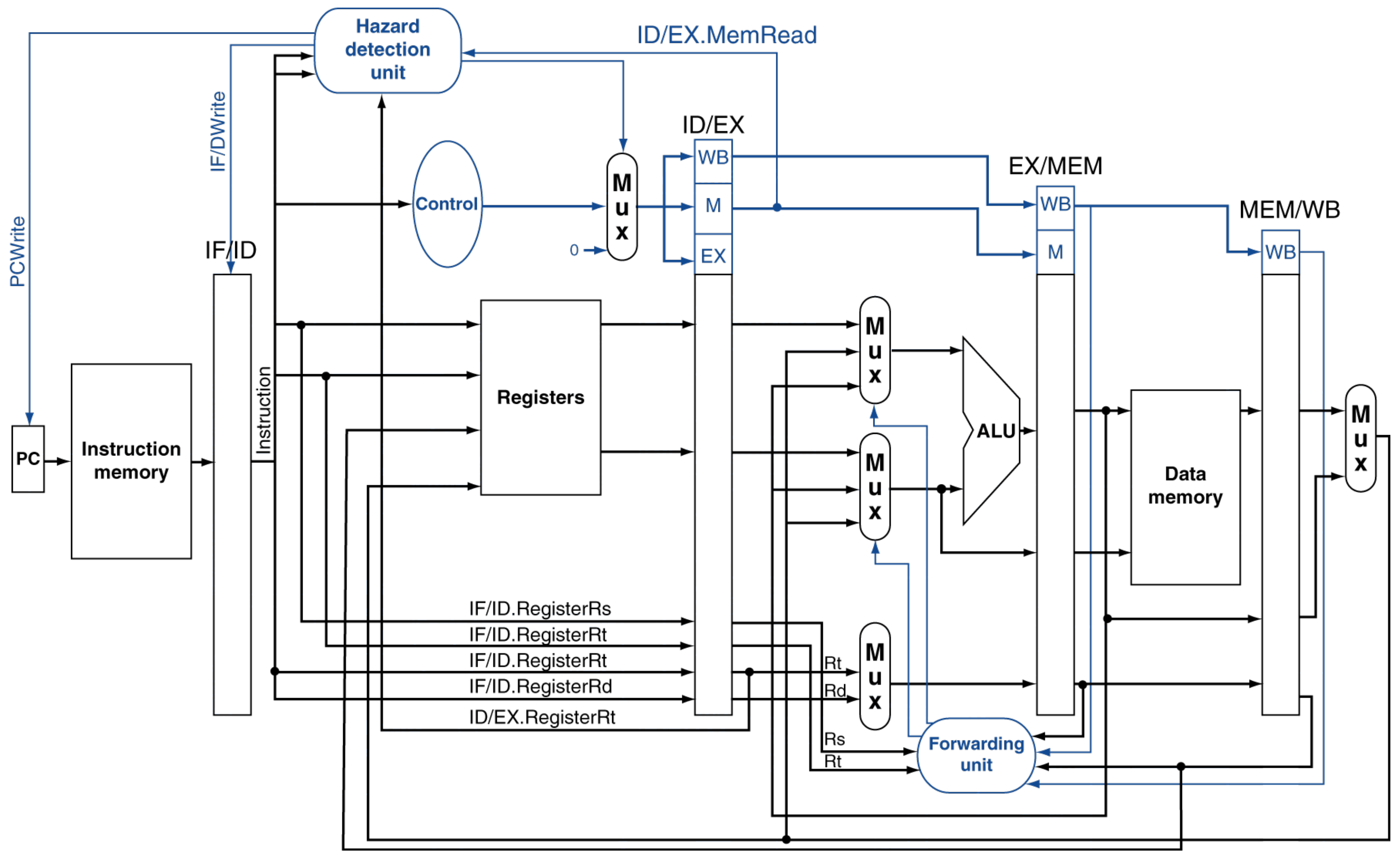
Stall inserted here

## Stall/Bubble in the Pipeline



Or, more accurately…

# Datapath with Hazard Detection


Hazard detection unit
ID/EX.MemRead
IF/DWrite
PCWrite
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
Control
Mux
0
IF/ID
PC
Instruction memory
Instruction
Registers
Mux
ALU
Mux
Data memory
Mux
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Mux
Rs
Rt
Forwarding unit