

# Chapter 4

## The Processor

Fall 2018

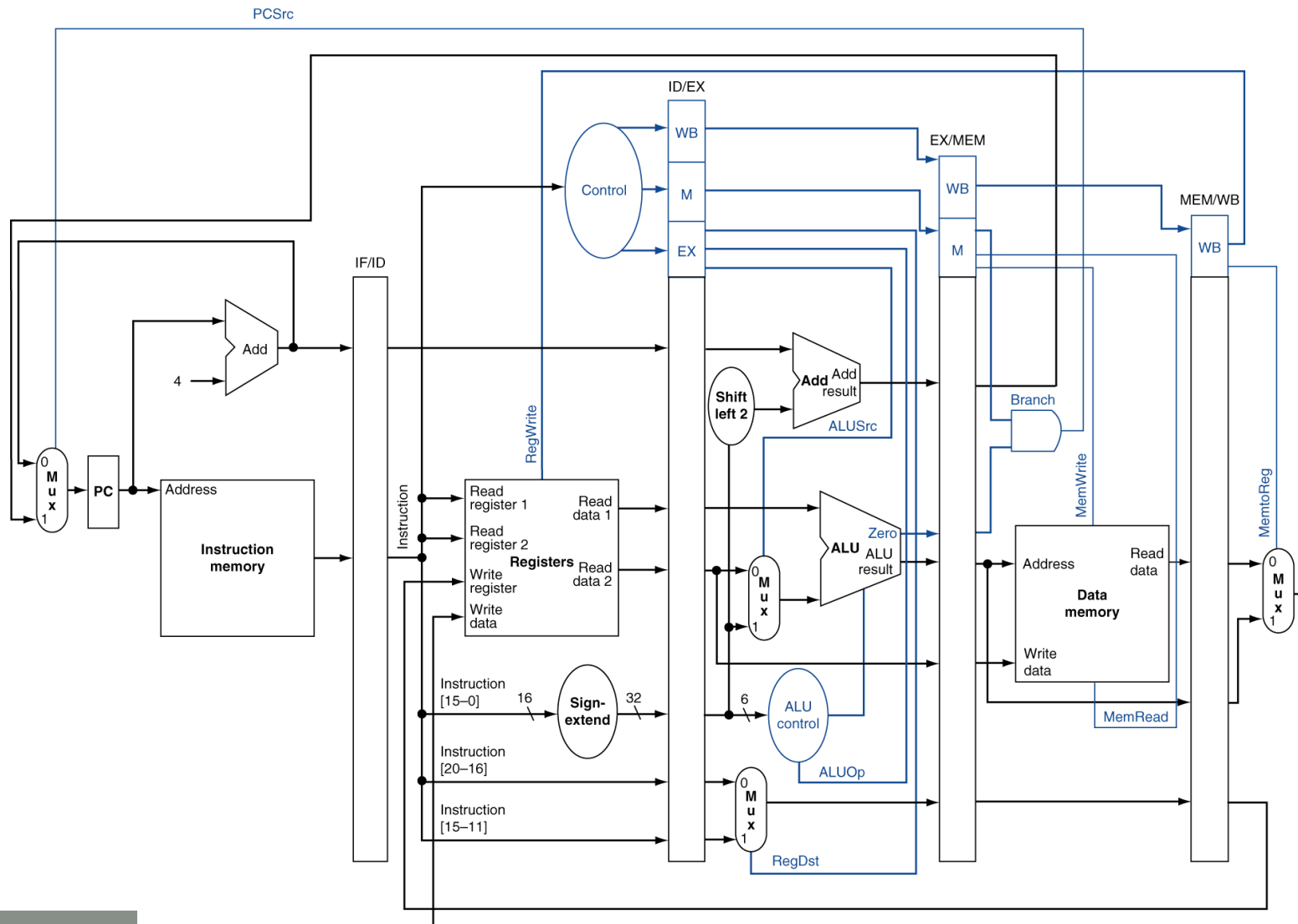
Soontae Kim

School of Computing, KAIST

# Announcement

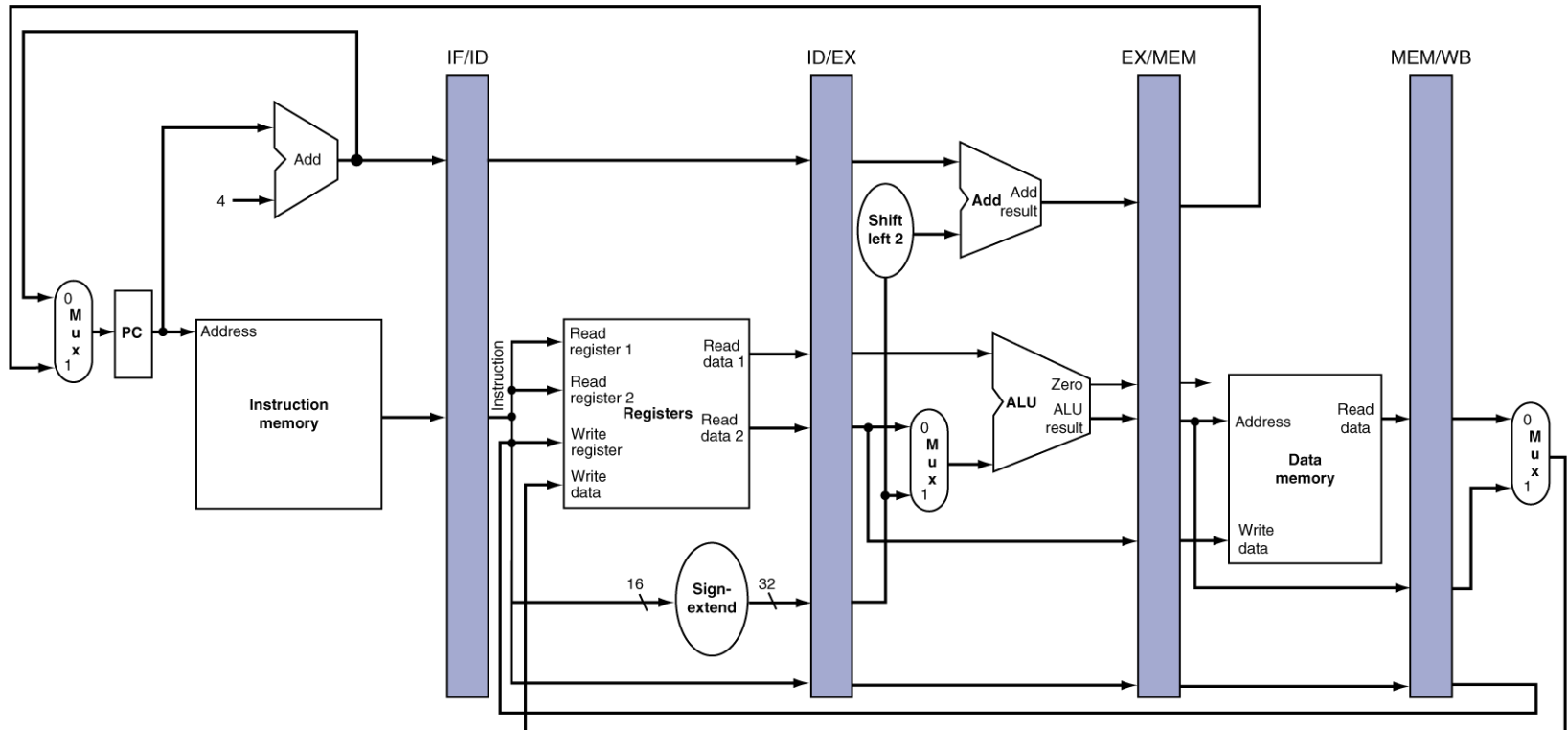
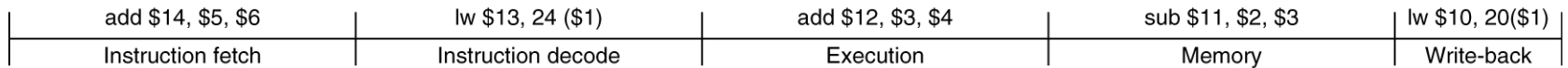
- Project #2
  - Due on Nov. 9
  - Studying pipelining with simulator
- Cheating on HW#1
  - Copying from Internet
  - Copying friends' HW
  - Modifying friends' HW
    - Changing variable names
    - Using slightly different algorithms
  - Start HW by yourselves and discuss later

# Pipelined Control



# Single-Cycle Pipeline Diagram

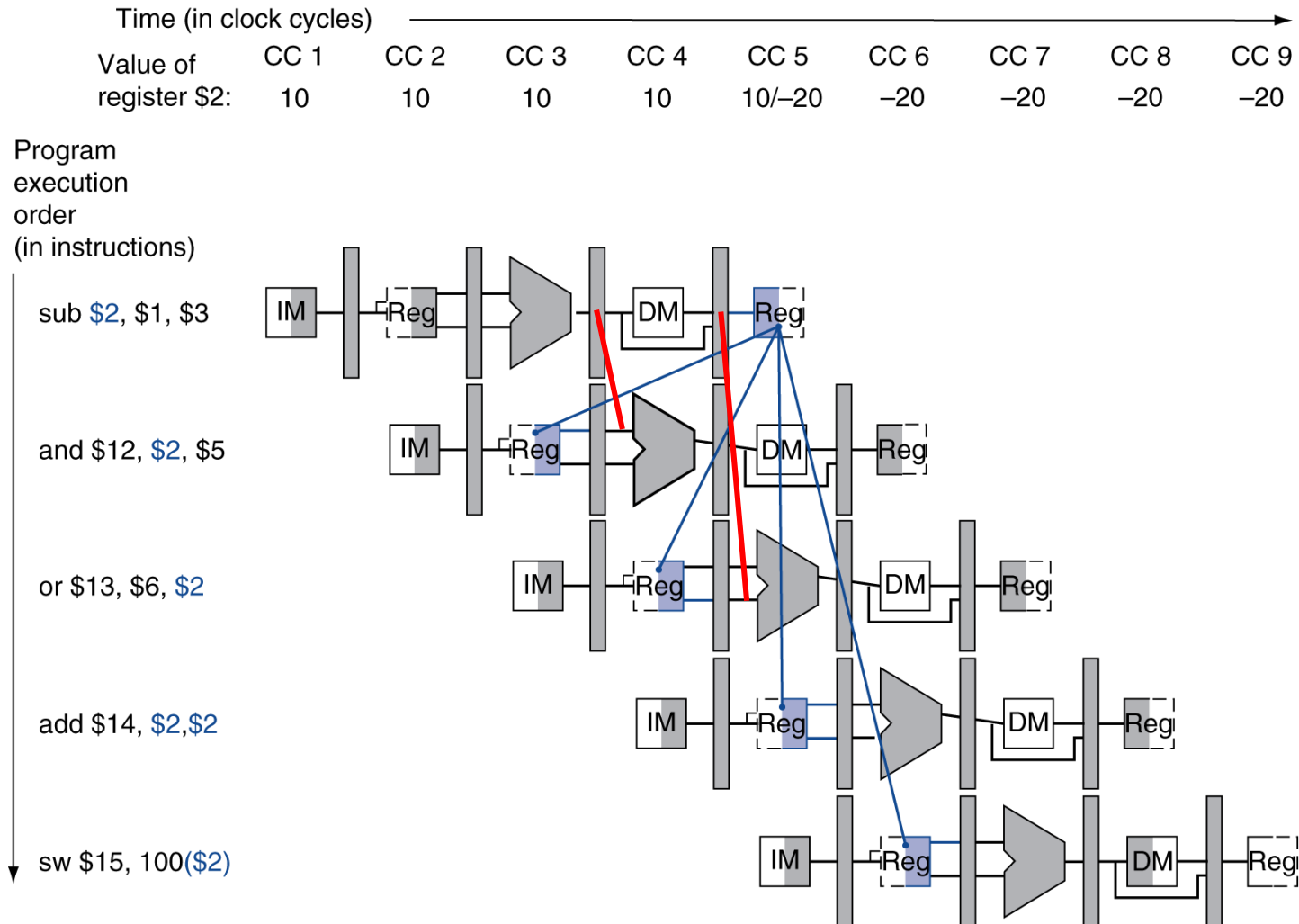
- Different instructions in stages



# 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

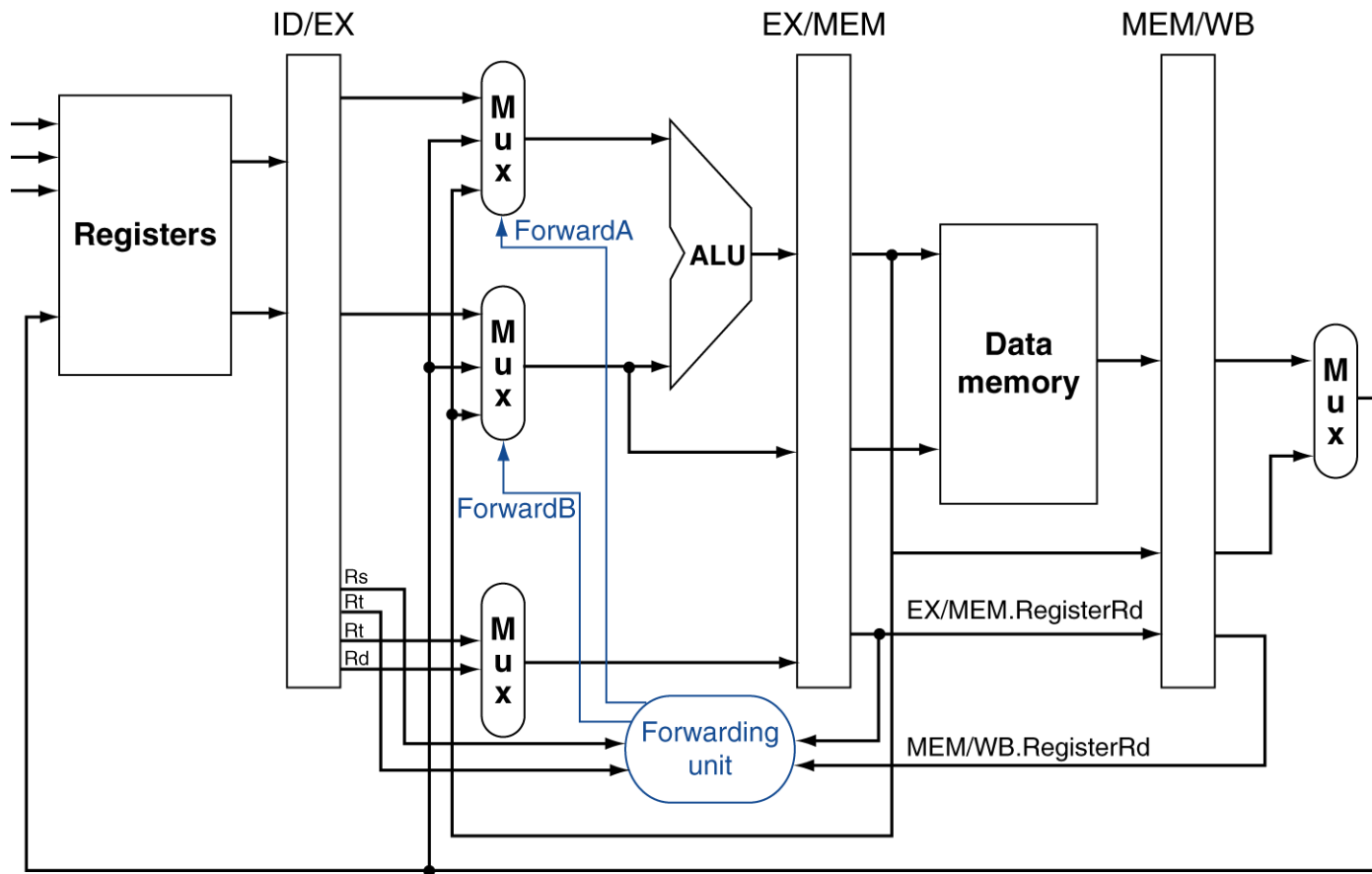
Fwd from  
MEM/WB  
pipeline reg

# 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  $\neq$  0,  
MEM/WB.RegisterRd  $\neq$  0



# Forwarding Paths



b. With forwarding

# Forwarding Conditions

## ■ EX hazard

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

ForwardA = 10

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))

ForwardB = 10

## ■ MEM hazard

- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

ForwardA = 01

- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))

ForwardB = 01

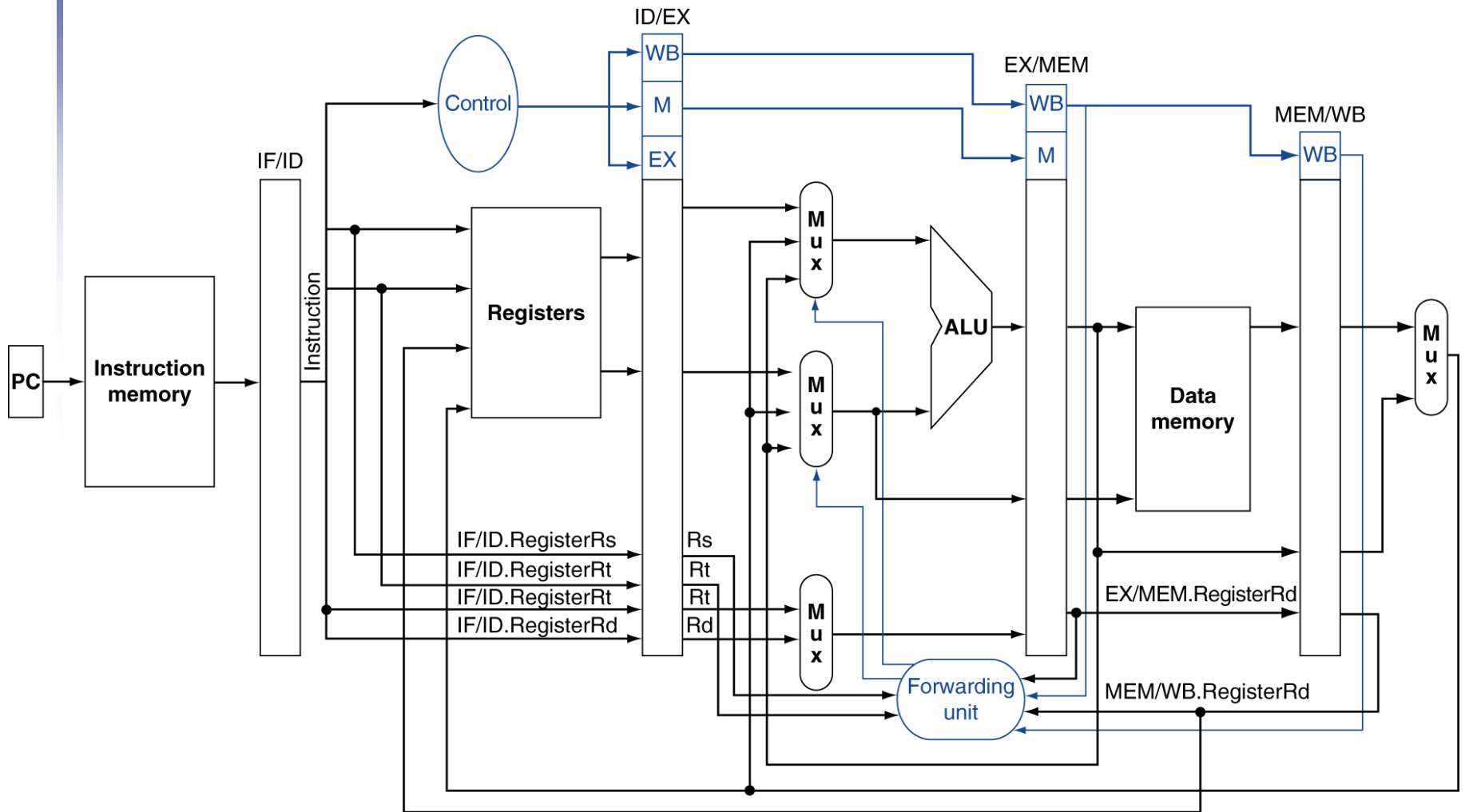
# Double Data Hazard

- Consider the sequence:
  - add \$1, \$1, \$2
  - add \$1, \$1, \$3
  - add \$1, \$1, \$4
- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

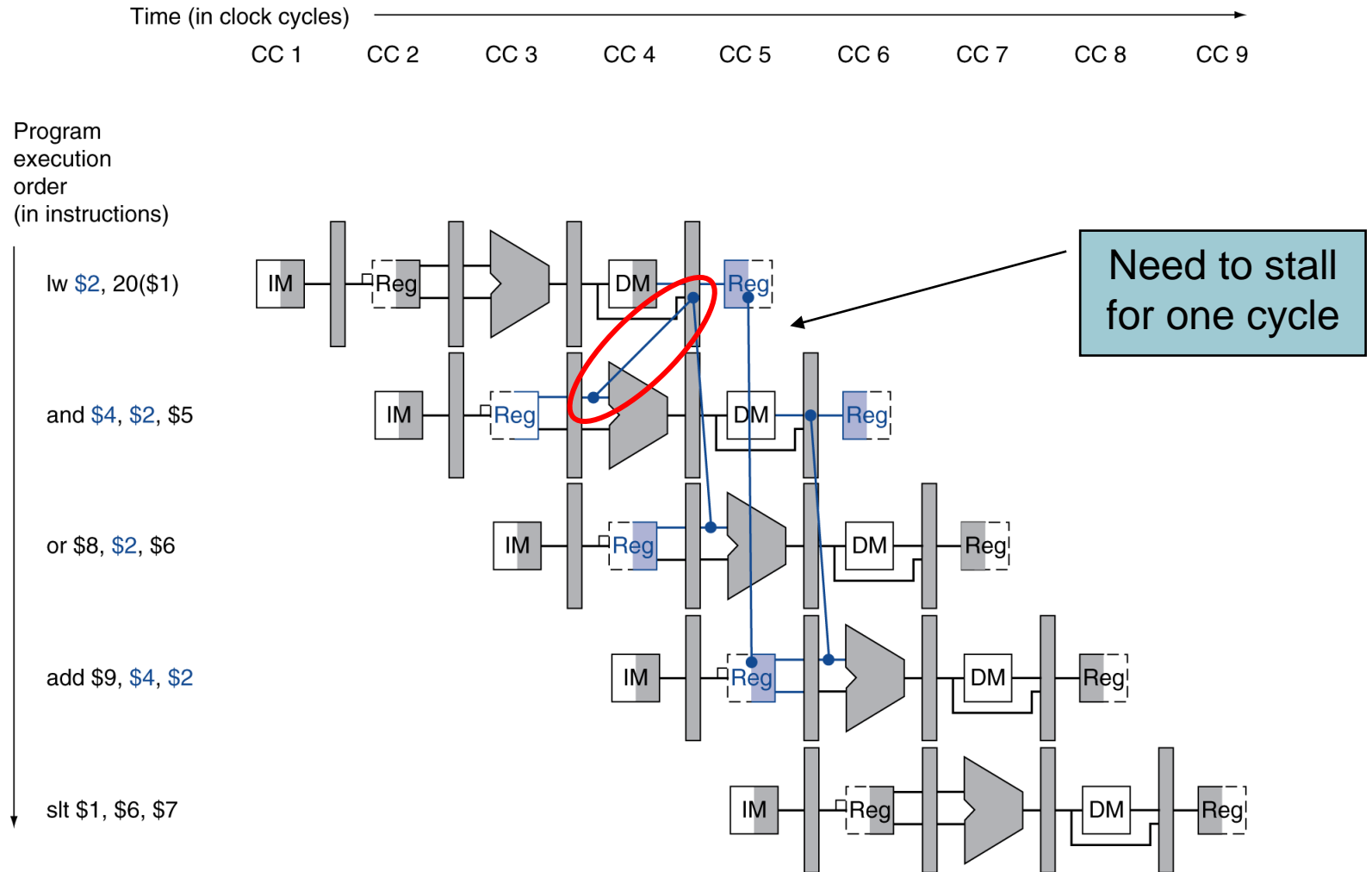
# Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))  
ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))  
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))  
ForwardB = 01

# Datapath with Forwarding



# Load-Use Data Hazard



# Load-Use Hazard Detection

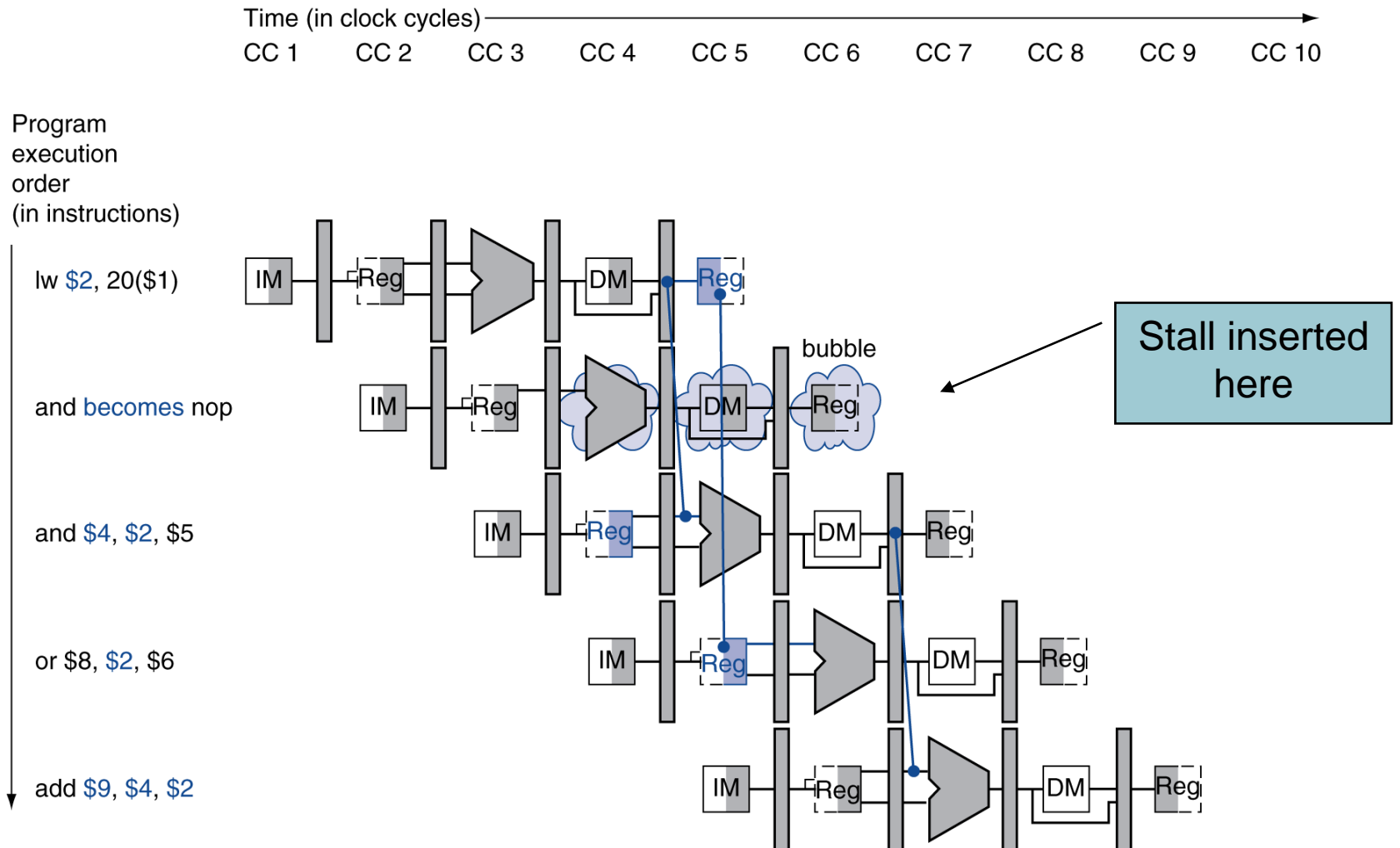
- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    - ((ID/EX.RegisterRt = IF/ID.RegisterRs) or (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall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

# How to Stall the Pipeline

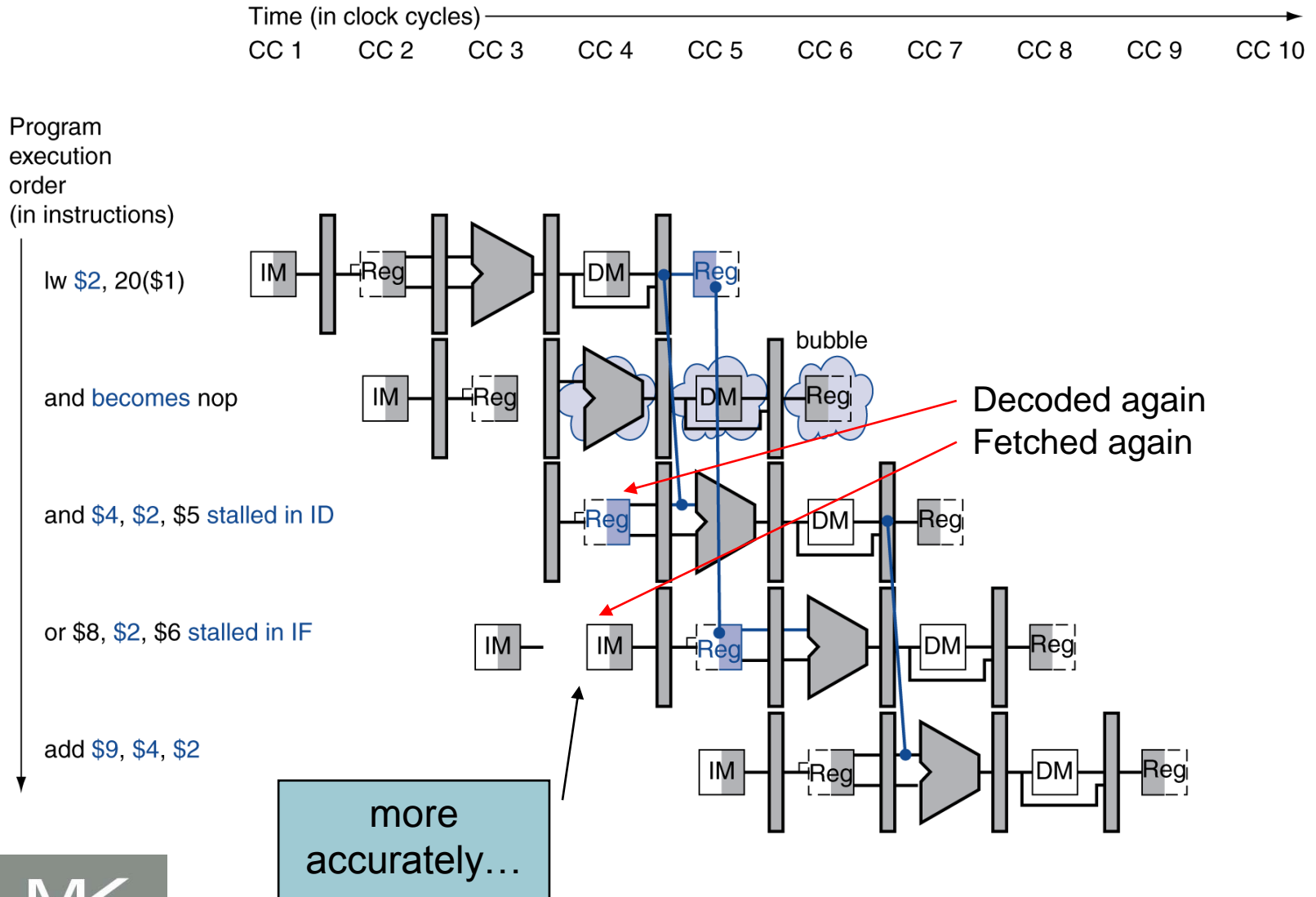
- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID registers
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for  $T_w$ 
    - Can subsequently forward to EX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



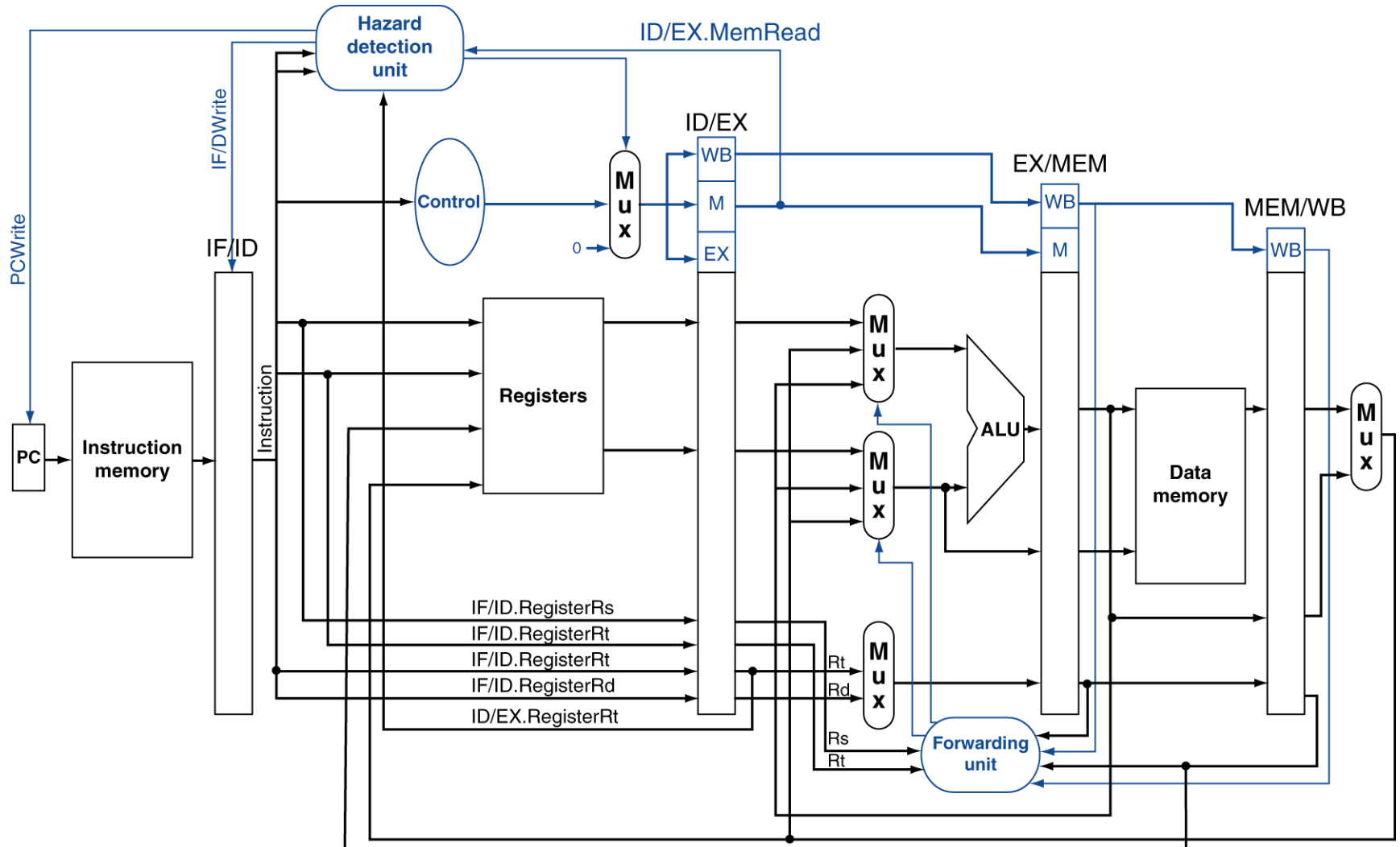
# Stall/Bubble in the Pipeline



# 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