# Where are we?

- We have explored a single clock cycle processor design [datapath and control]
- We have introduced pipelining and established its performance benefits and operational challenges
- We are exploring pipelined processor implementation [datapath and control]

# The Processor: Pipelined Control

Dr. Ahmed Zahran

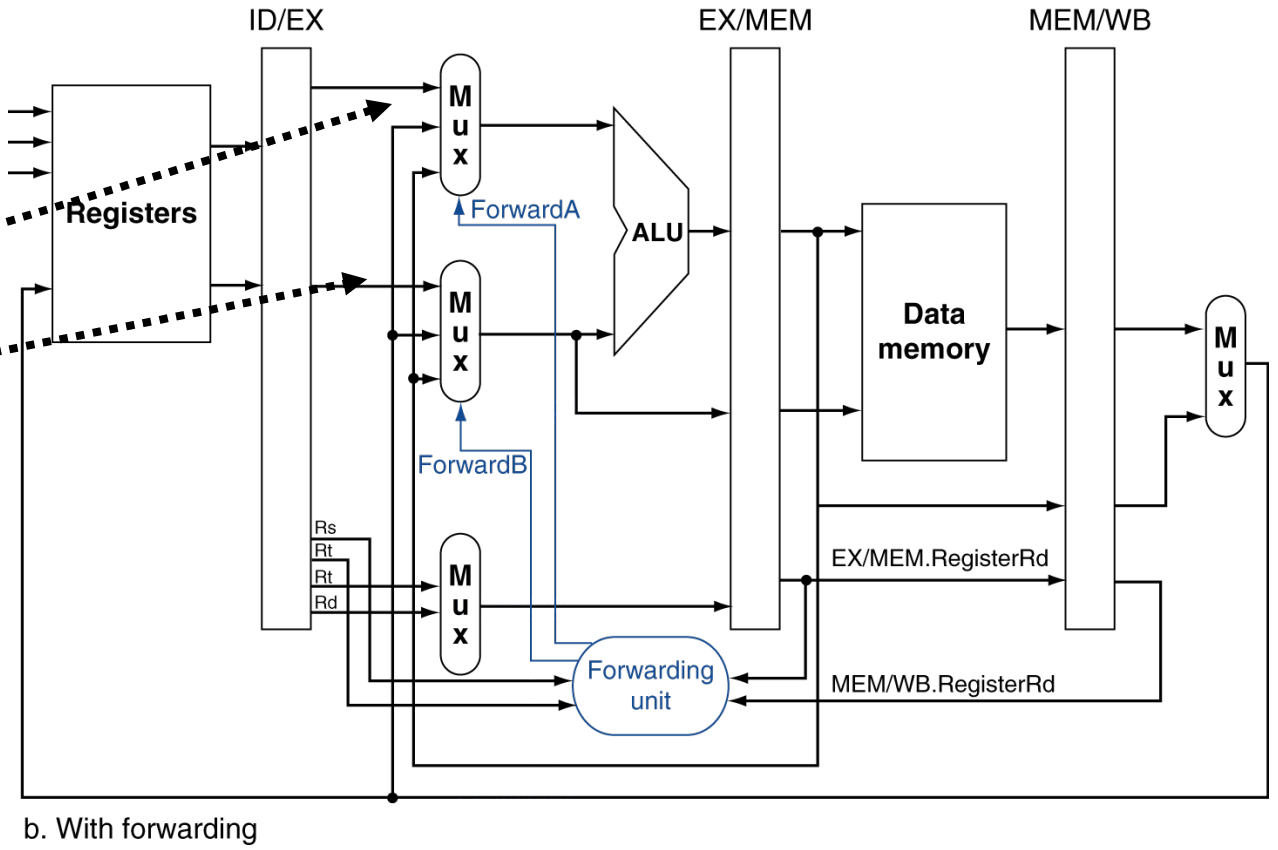
WGB 182

a.zahran@cs.ucc.ie

# Forwarding

***Register Naming convention:*** 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



b. With forwarding

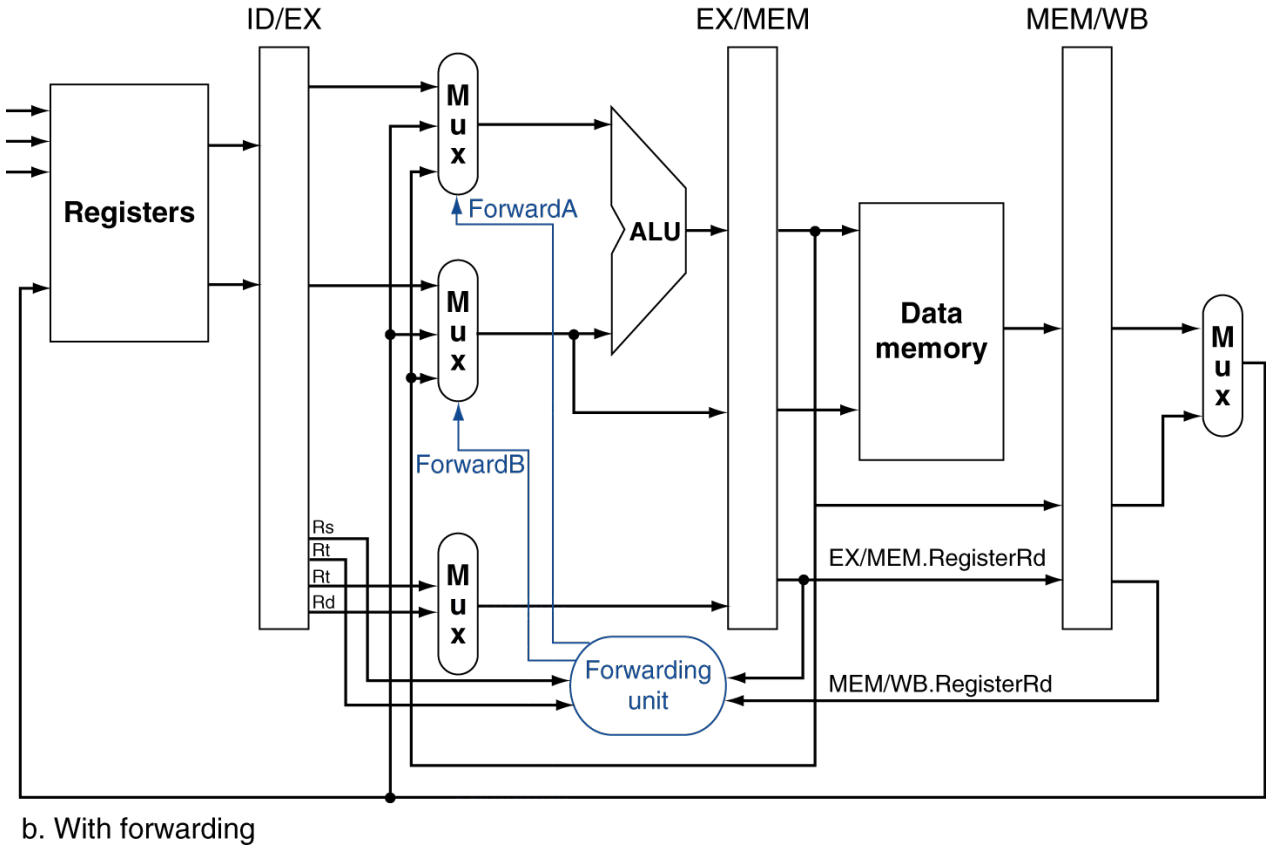
# When to Forward? (1/2)

- **Data hazards** when
  - 1a. ID/EX.RegisterRs = EX/MEM.RegisterRd
  - 1b. ID/EX.RegisterRt = EX/MEM.RegisterRd

**Fwd from EX/MEM pipeline reg**

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

Fwd from MEM/WB pipeline reg



b. With forwarding

sub **$2**, $1,$3
and $12,$2,$5
or $13,$6,$2

# When to Forward? (2/2)

- 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***

- Recall in MIPS every use of $zero ($0) as operand yields an operand value of 0
  - Example: sll $0, $0, 0  (NOP)

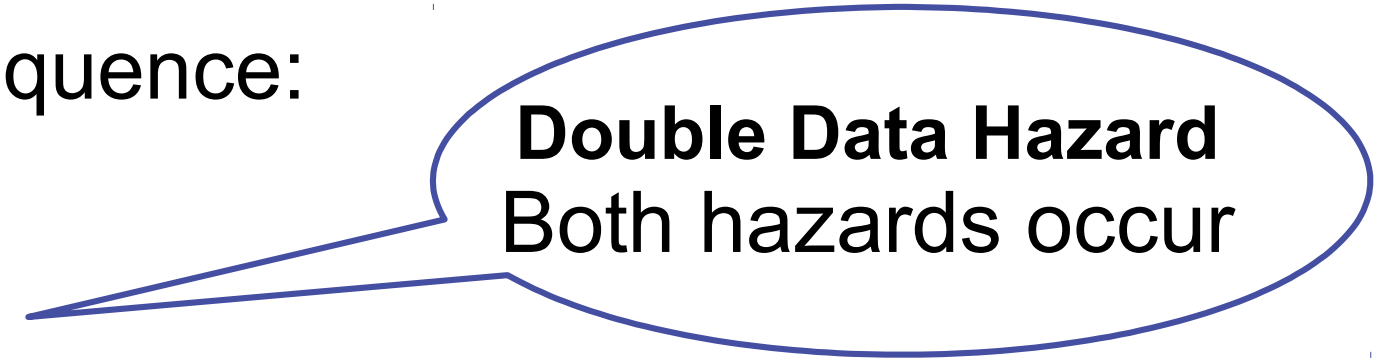
# Forwarding Conditions

- **EX/MEM 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/WB 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

# Double Data Hazard

- Consider the sequence:

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

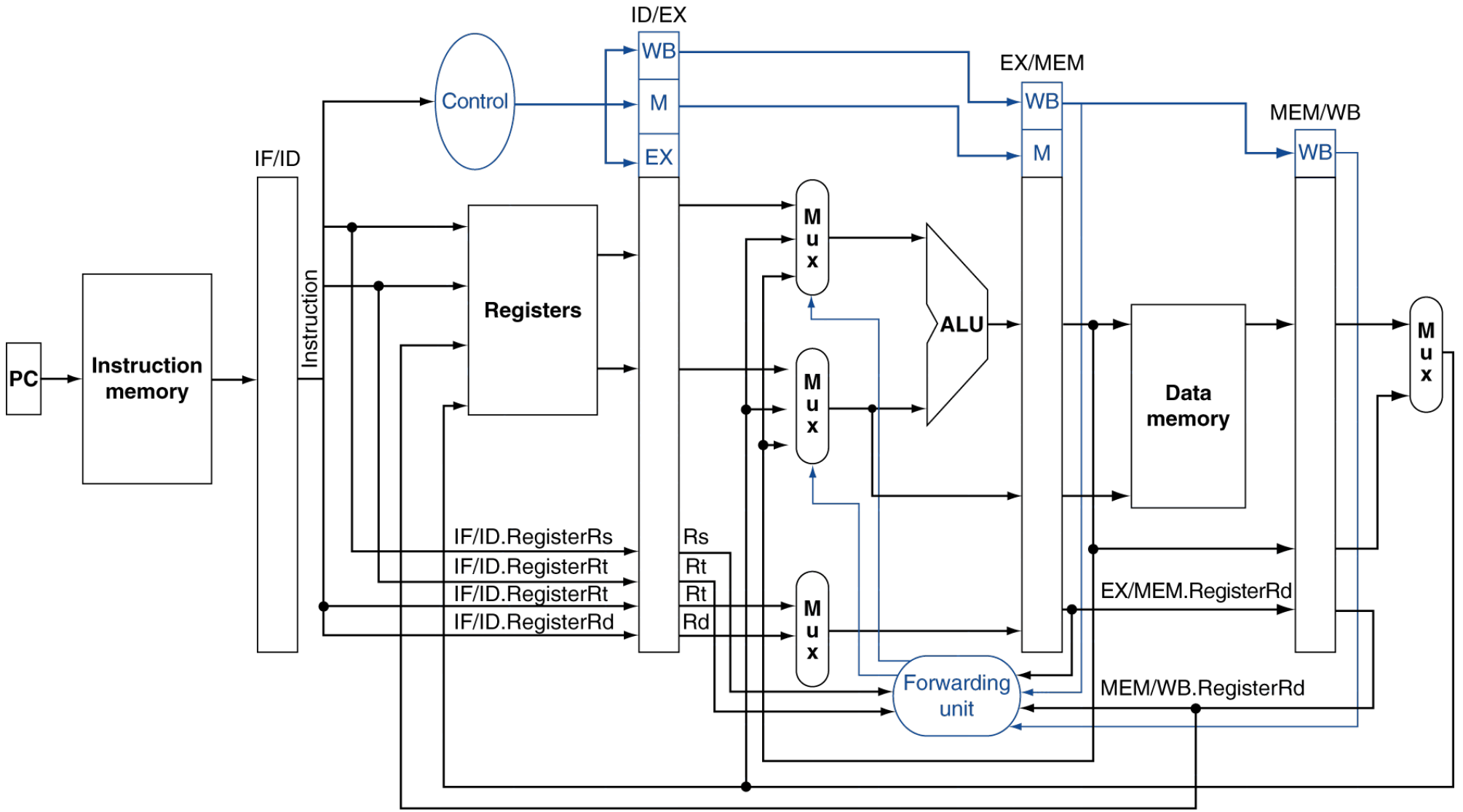



- In this case, the processor should use the most recent result at MEM stage
  - Revise MEM hazard condition to Only fwd if EX hazard condition isn't true

# Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 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 ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

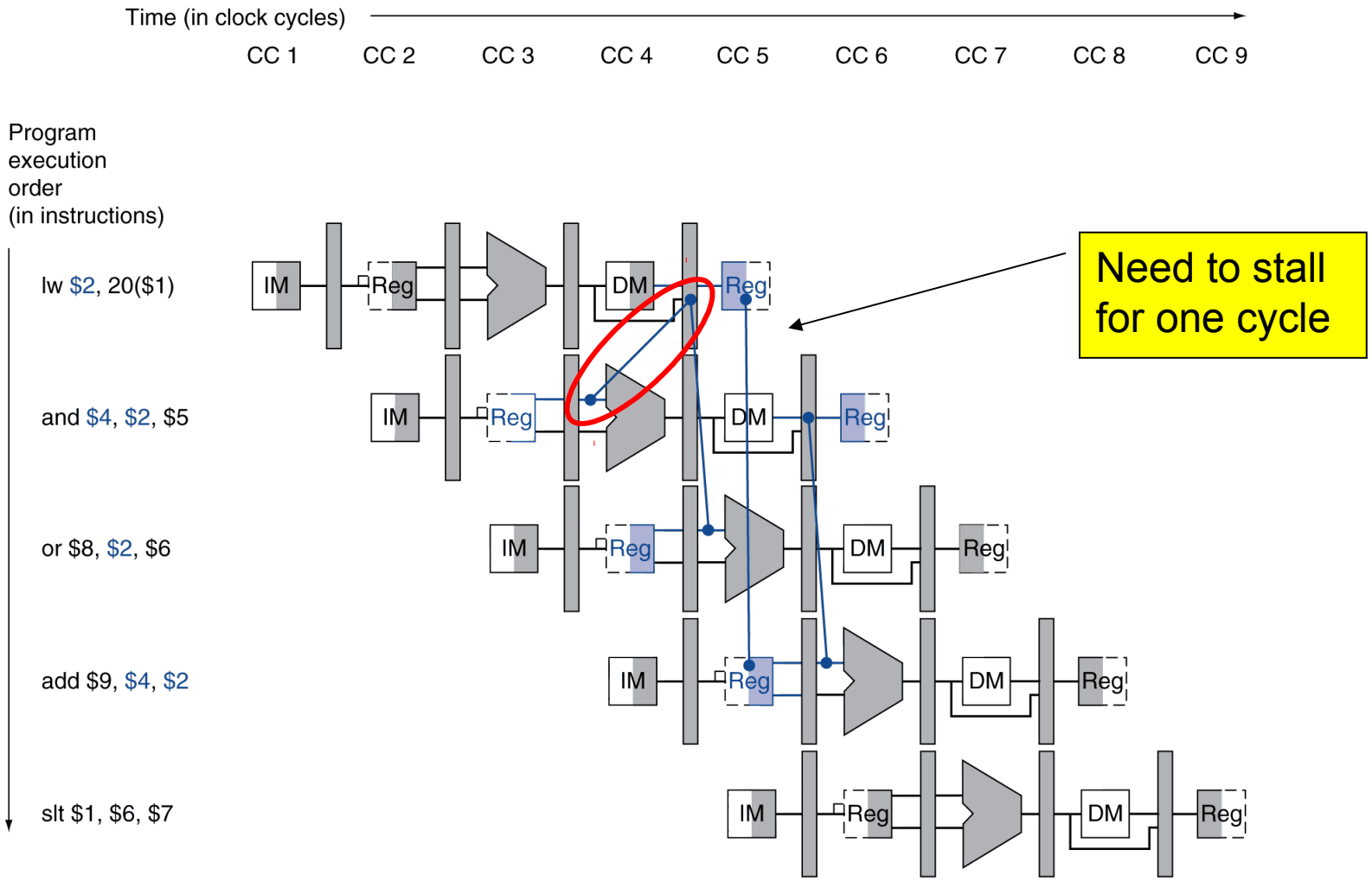
# Datapath with Forwarding and Control



# Memory Load Data Hazard

- Data forwarding does not work when
  - Instruction tries to read a register following a load instruction that writes the same register.
- Something must stall in the pipeline for a combination of load followed by an instruction that reads its result.
- Therefore, we need ***a hazard detection unit*** in the pipeline to implement a stall.

# Pipelined Load Data Hazard



# Memory Load Hazard Detection

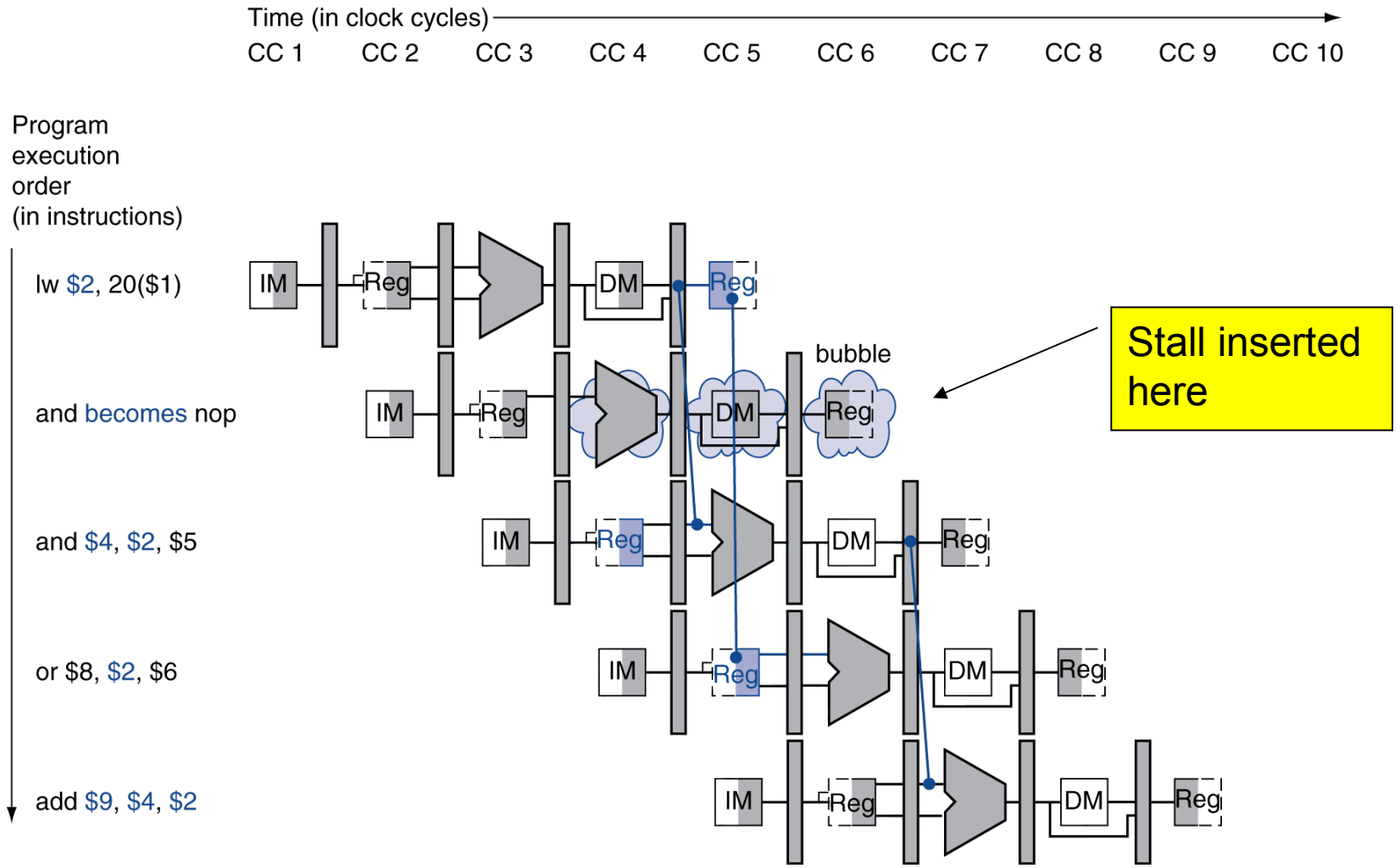
- Check when using instruction is decoded ***in ID stage***
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load hazard occurs when

  ID/EX.MemRead **and**
  ((ID/EX.RegisterRt = IF/ID.RegisterRs) **or**
  (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

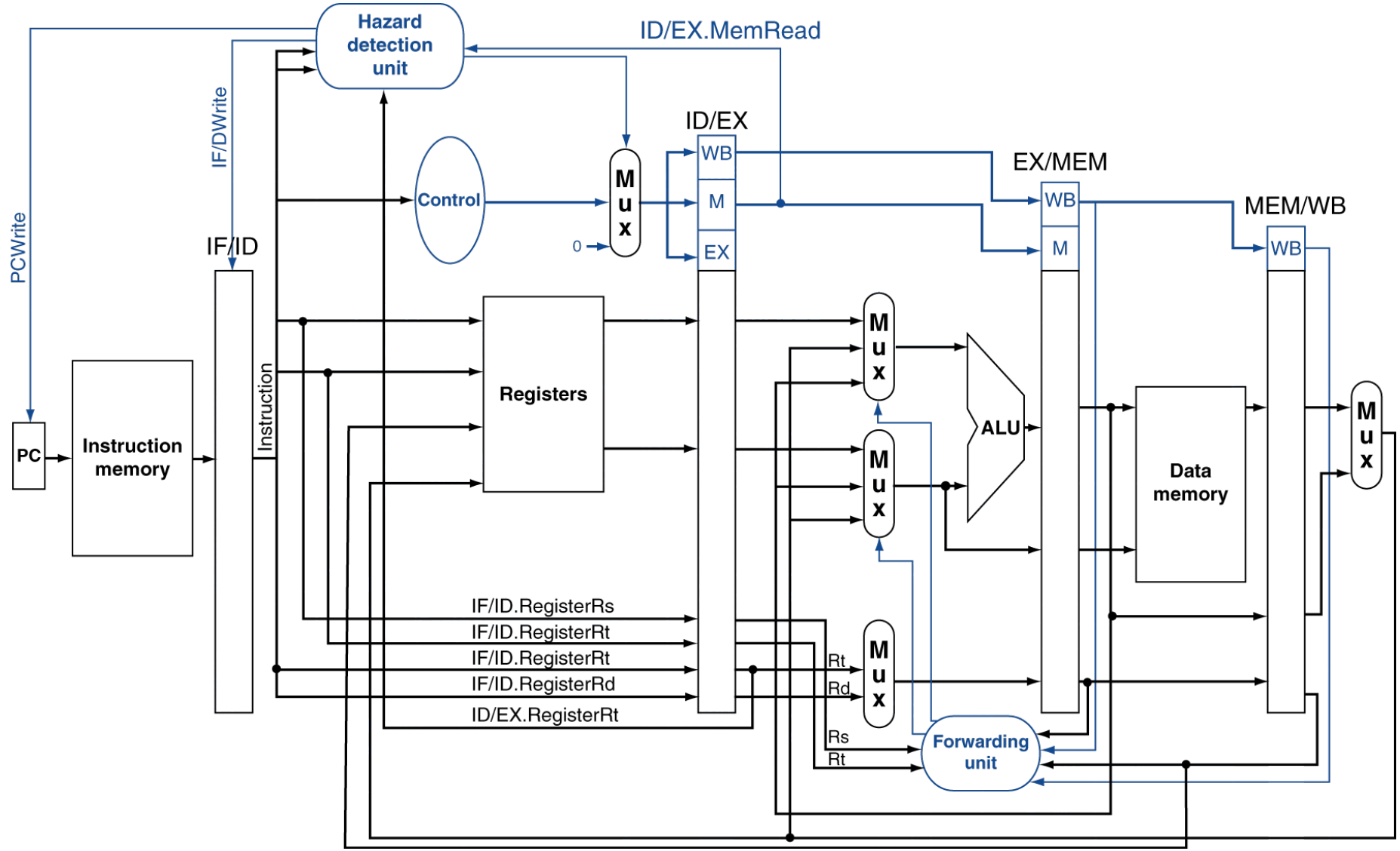
# How to Stall the Pipeline

- 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
  - 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



# Datapath with Hazard Detection



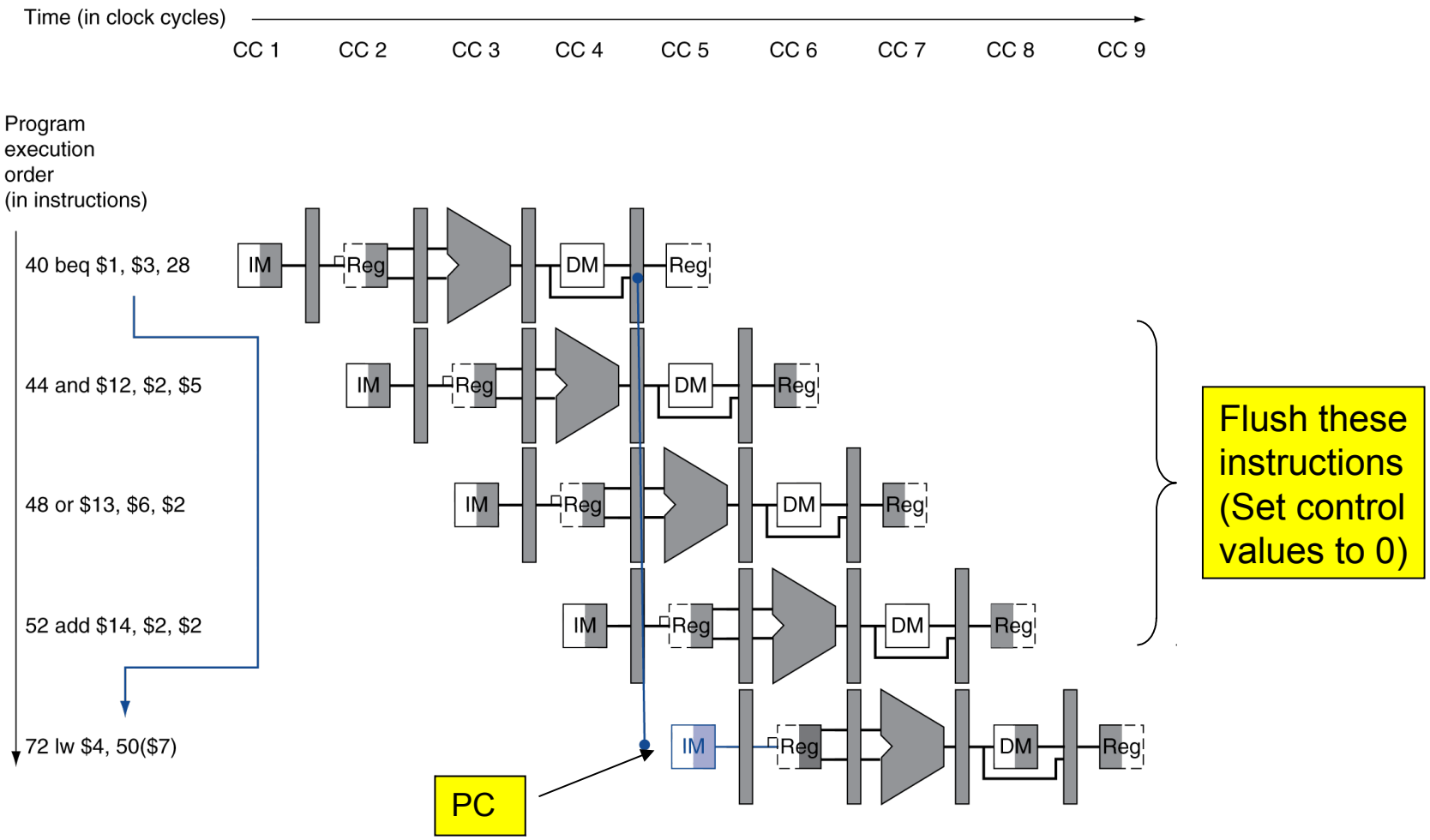
- Explored hardware implementation of bypassing (datapath and control)

# NEXT: Branch Hazards

- Speeding using hardware optimization
- Data hazards for branch operands
- Dynamic branch prediction

# Branch Hazards

- If branch outcome determined in MEM

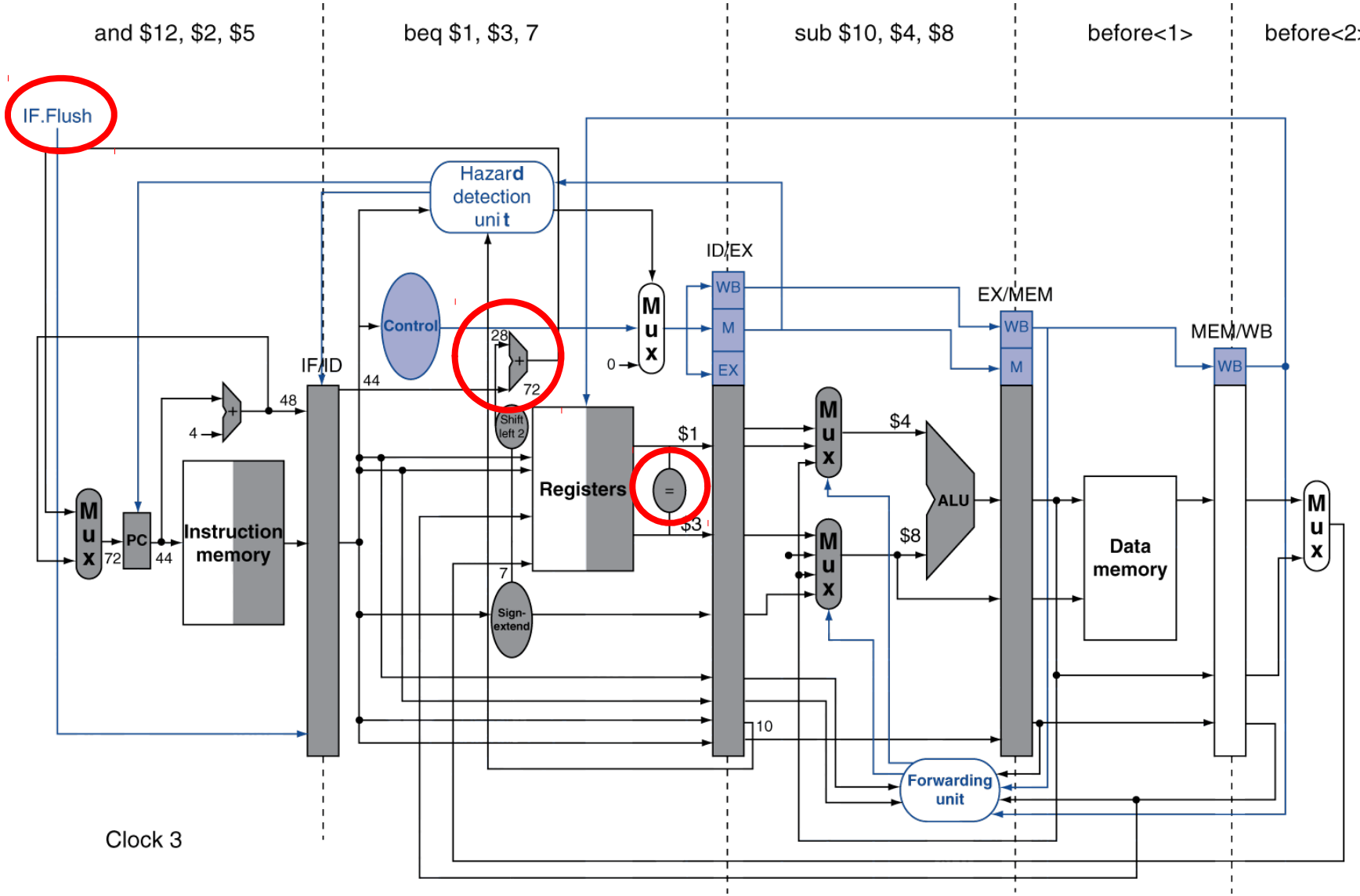


# Reducing Branch Delay

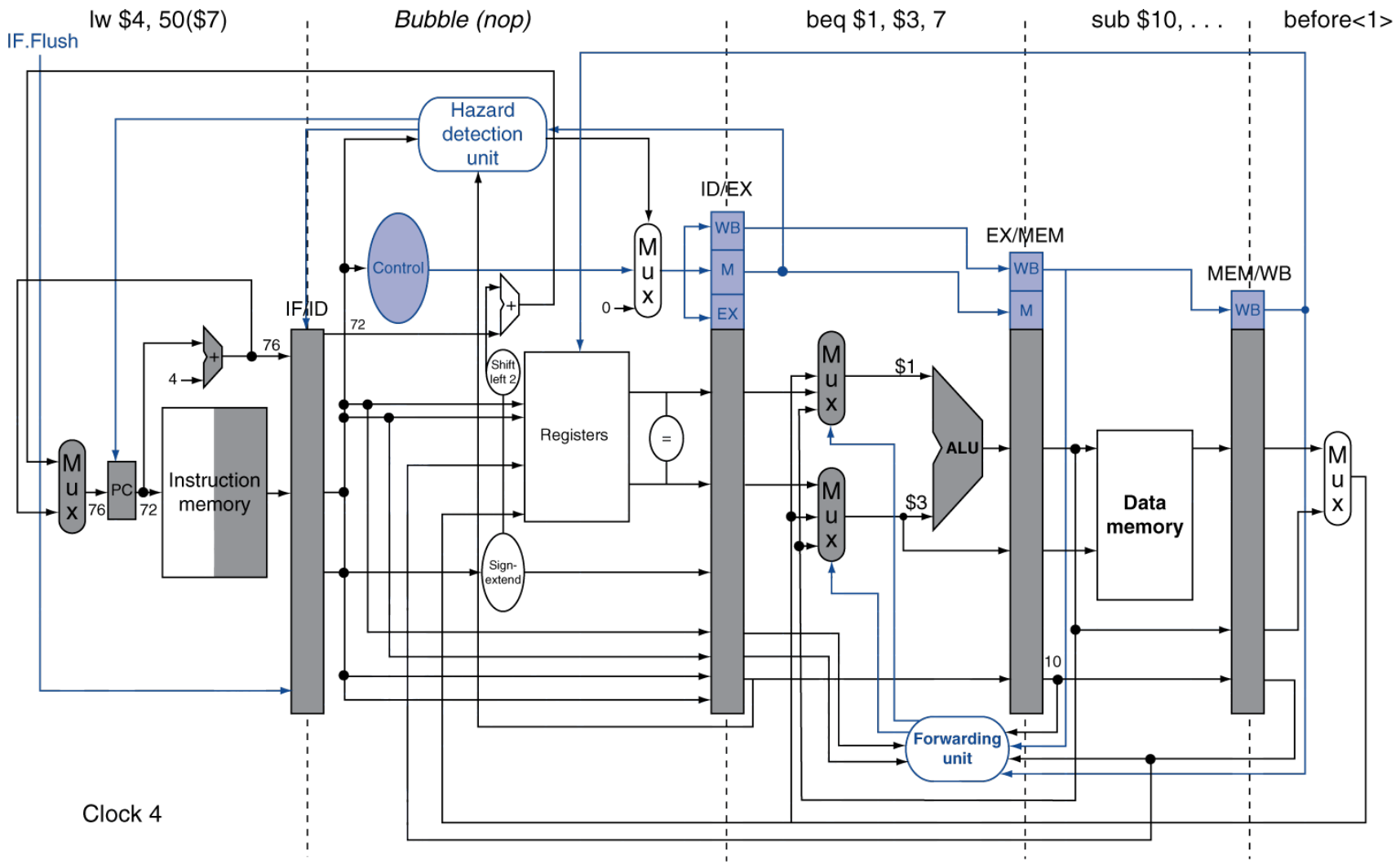
- Determine branching decision at the ID stage using additional hardware [hazard detection]
  - Target address adder
  - Register comparator (**XOR then ORing bits for equality**)
- Early branch detection reduces the penalty to one cycle
  - IF.flush to flush the fetched instruction
- Example: branch 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)
```

# Branch Taken Implementation (1/2)

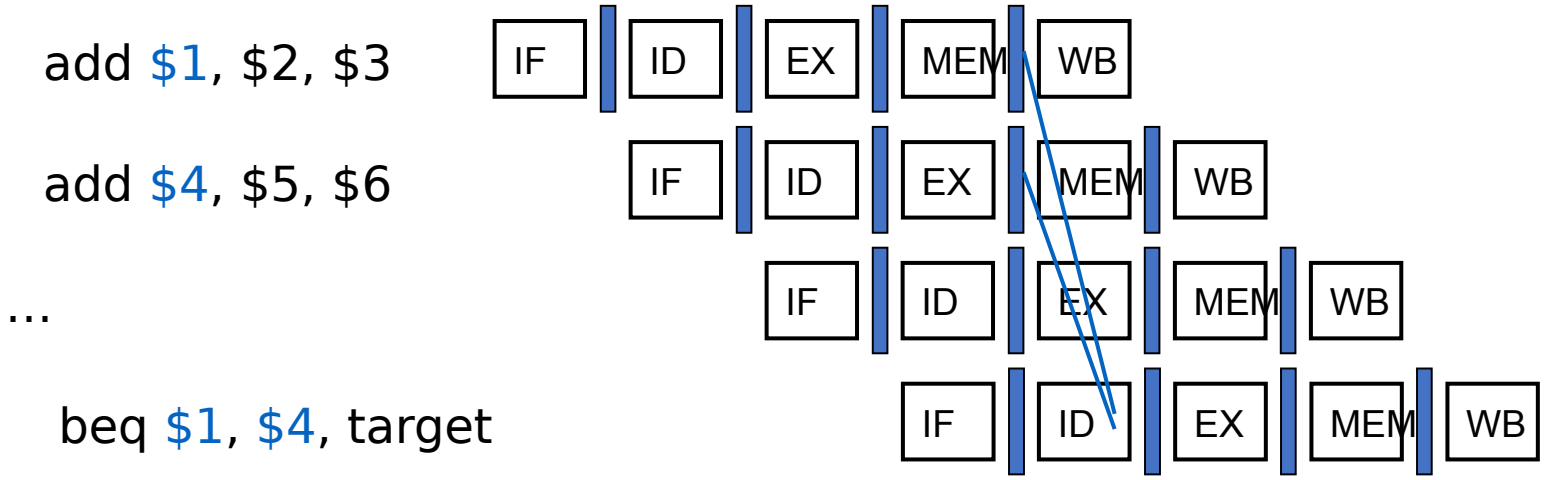


# Branch Taken Implementation (2/2)



# Data Hazards for Branches

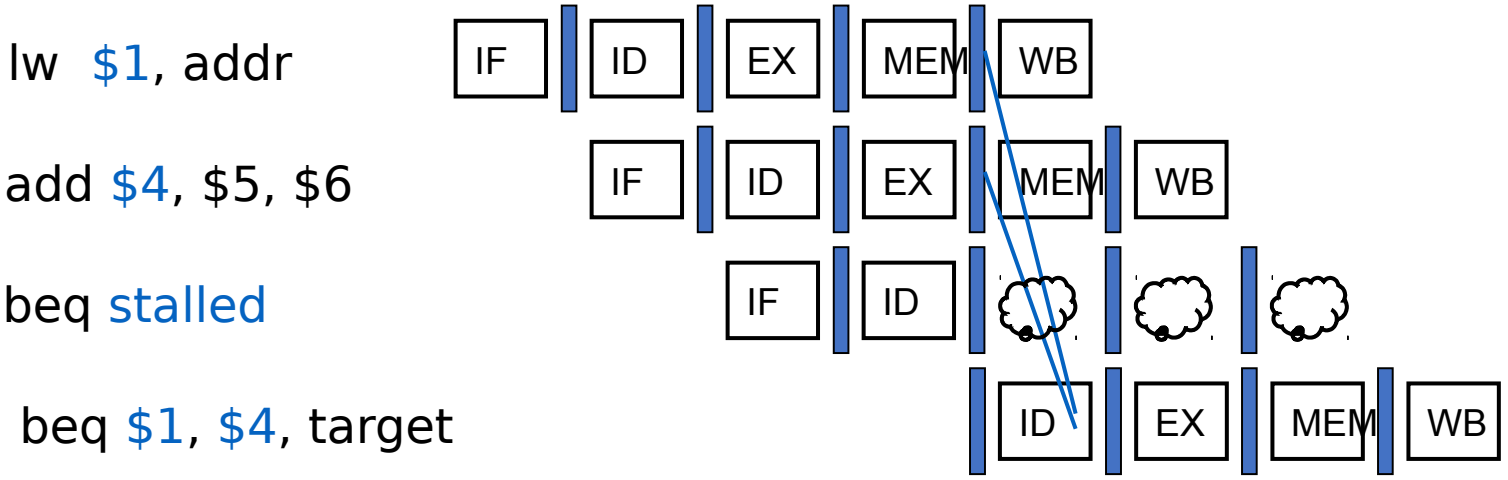
- If a comparison register is a destination of **2nd** or **3rd** preceding ALU instruction




- Can be resolved using forwarding

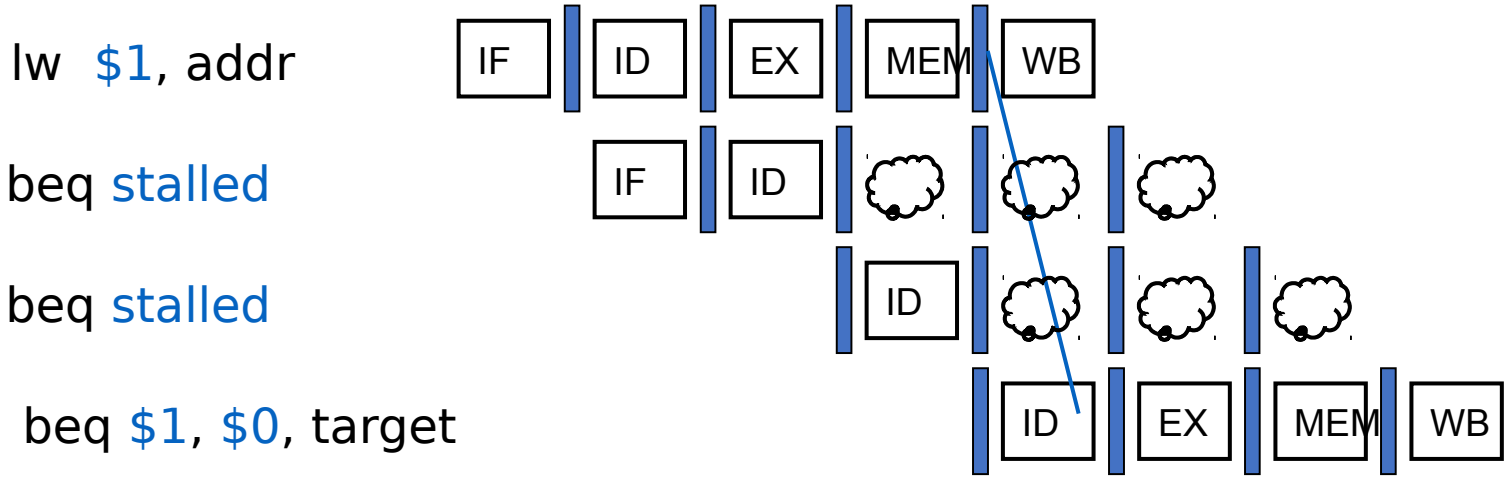
# Data Hazards for Branches

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



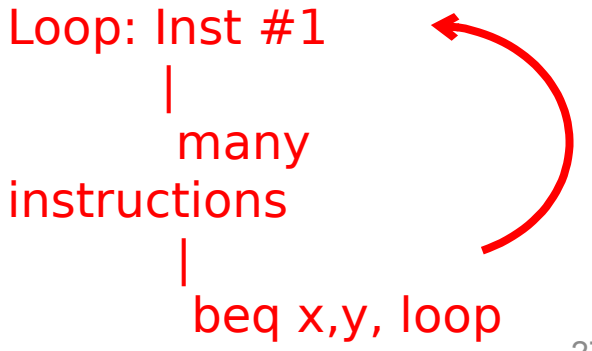
# Data Hazards for Branches

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



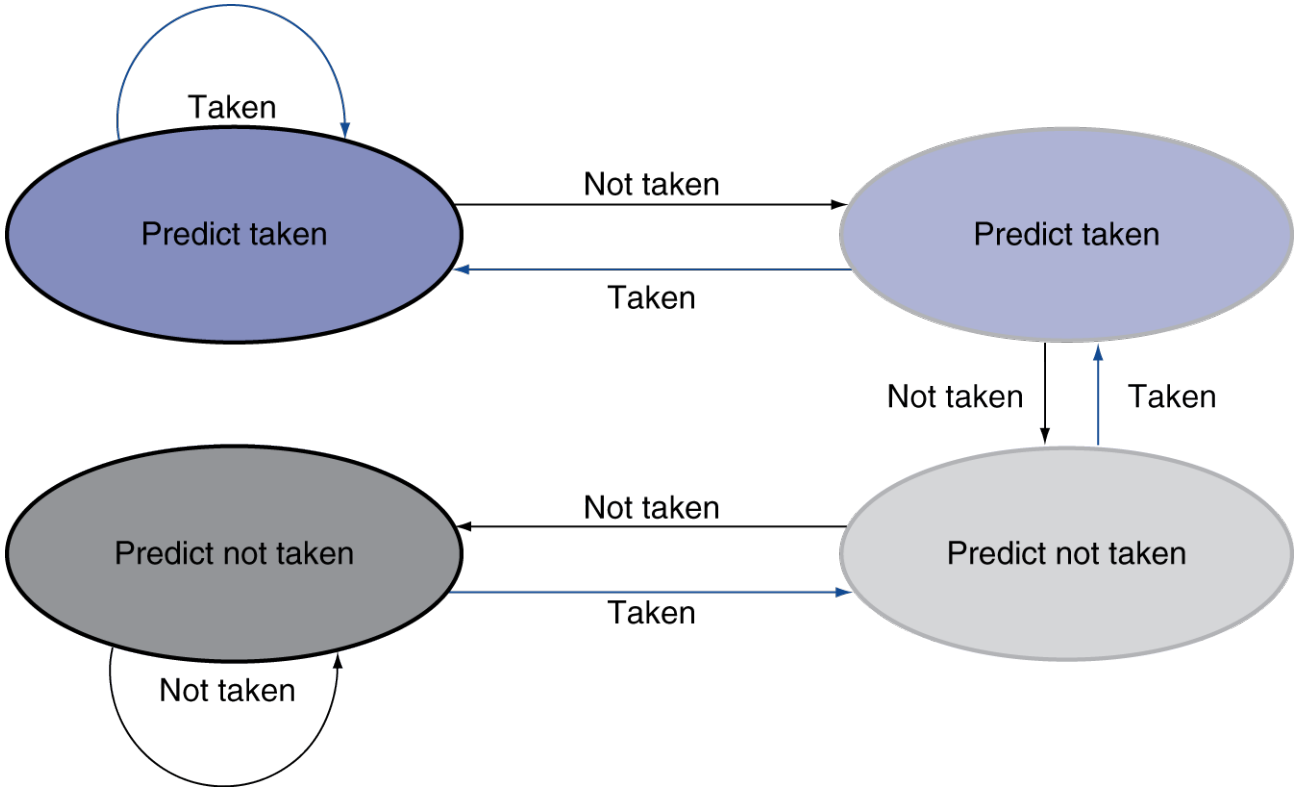
# Dynamic Branch Prediction

- In deeper and superscalar pipelines, the branch hazard penalty is significant
- ***Dynamic branch hazard prediction***
  - Branch prediction buffer (aka branch history table)
    - Indexed by recent branch instruction addresses
    - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, predict the outcome [repeat the last action]
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction
- What would happen for a loop?



# 2-Bit Predictor

- Only change prediction on two successive mispredictions



# Calculating the Branch Target

- We still need to calculate the target address

- Branch target buffer
  - ***Cache*** of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately

# Reading

- Sections 4.7 and 4.8