



## **Topic 8**

---

# **Data Hazards**

# Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Decision on control action depends on previous instruction

# Structure Hazards

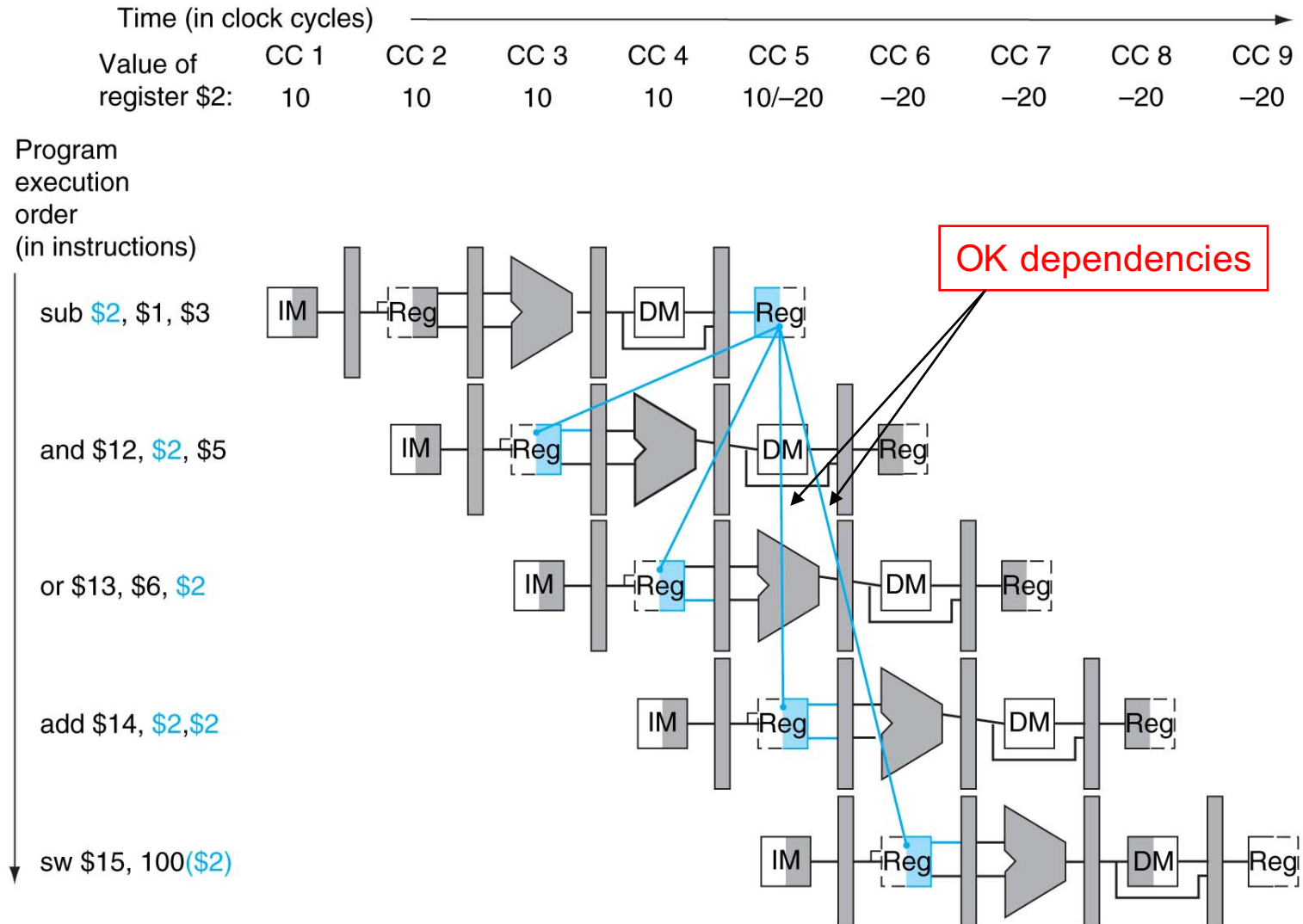
- Conflict for use of a resource
- If common memory for program and data
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - separate instruction/data caches (Harvard)
  - Common low-level memory (Von Neumann)

# 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)
```

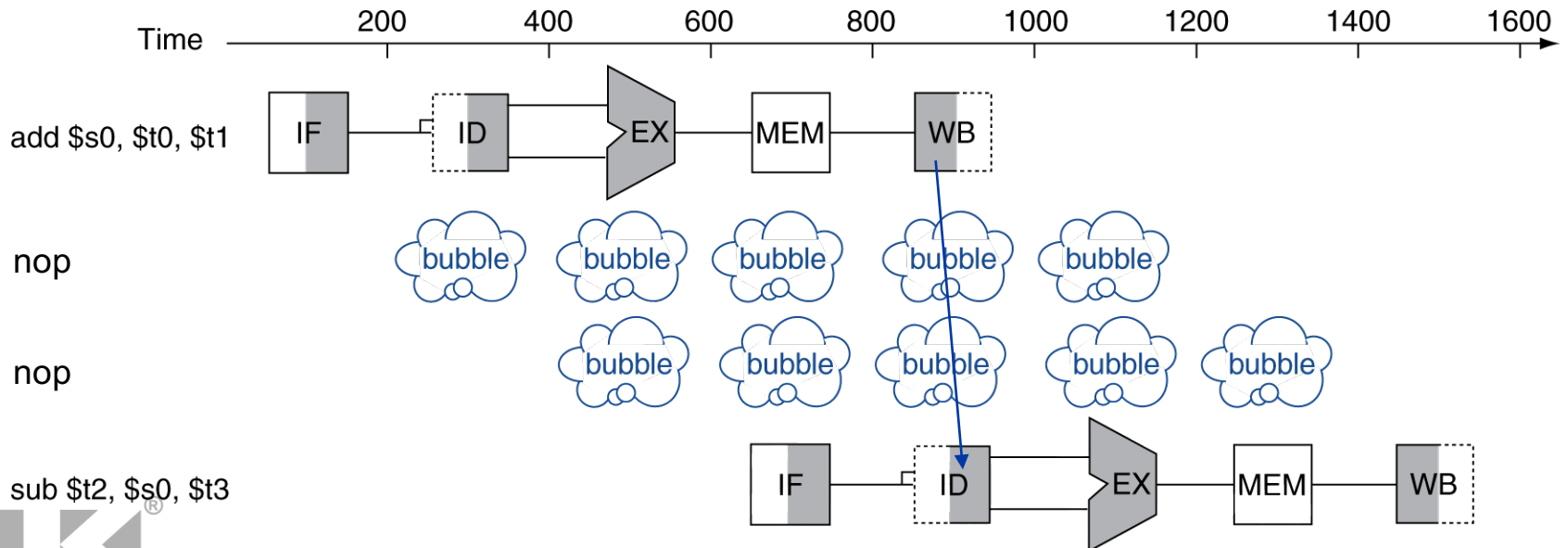
# Data Dependencies




# Data Hazards

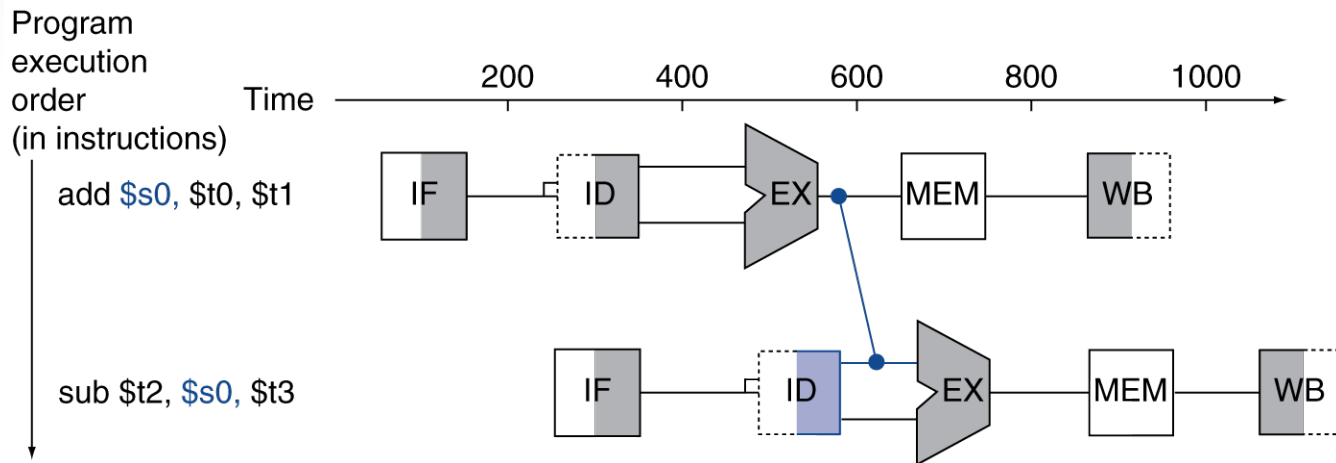
- An instruction must be delayed
  - If an instruction depends on completion of data access by a previous instruction
  - By inserting *bubbles* or *stalls*
  - Example

```
add    $s0, $t0, $t1
sub     $t2, $s0, $t3
```



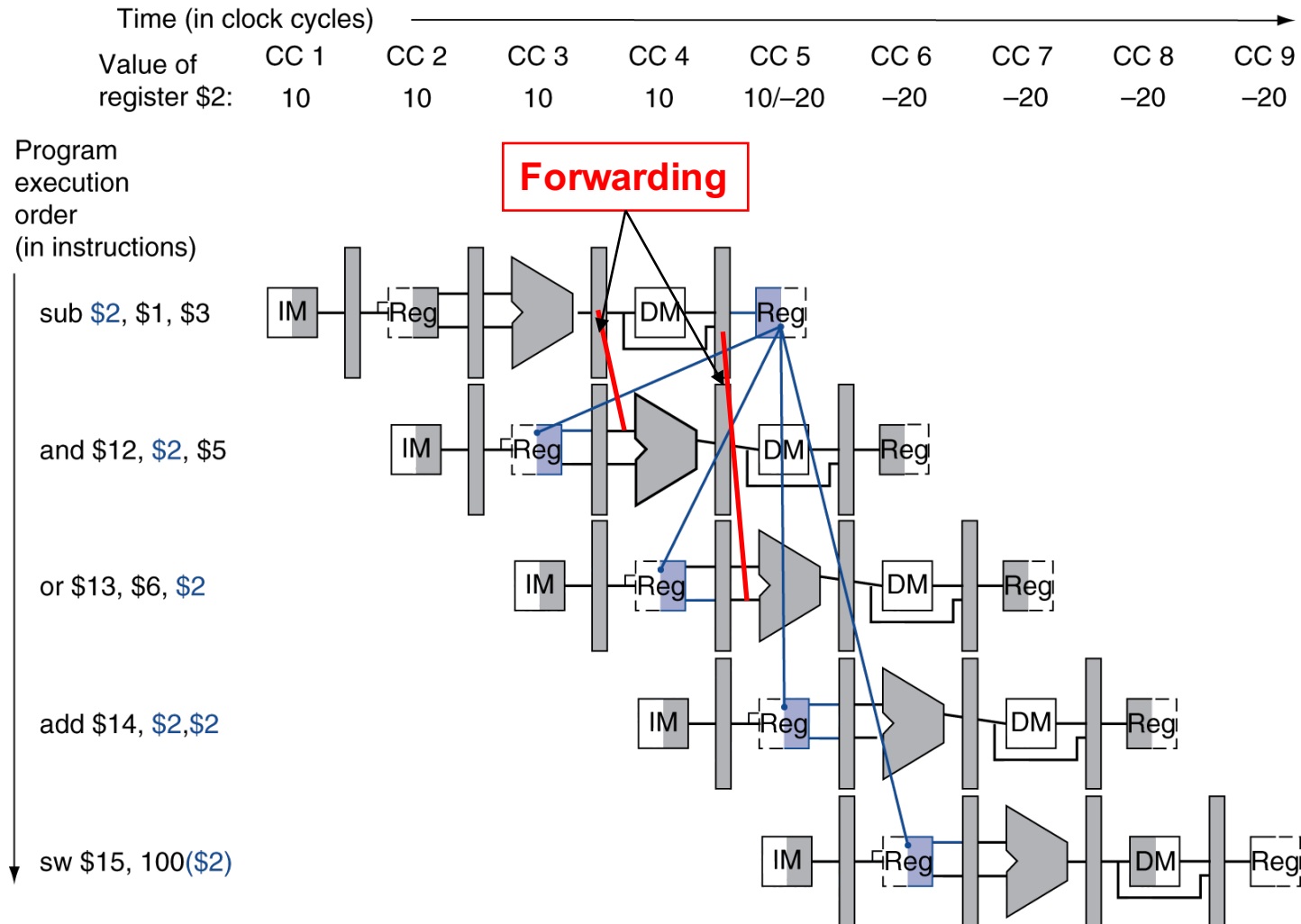
# Forwarding (aka Bypassing)

- Data hazard can be solved with **Forwarding**
  - To avoid delaying an instruction
- Use result as soon as it is computed 
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



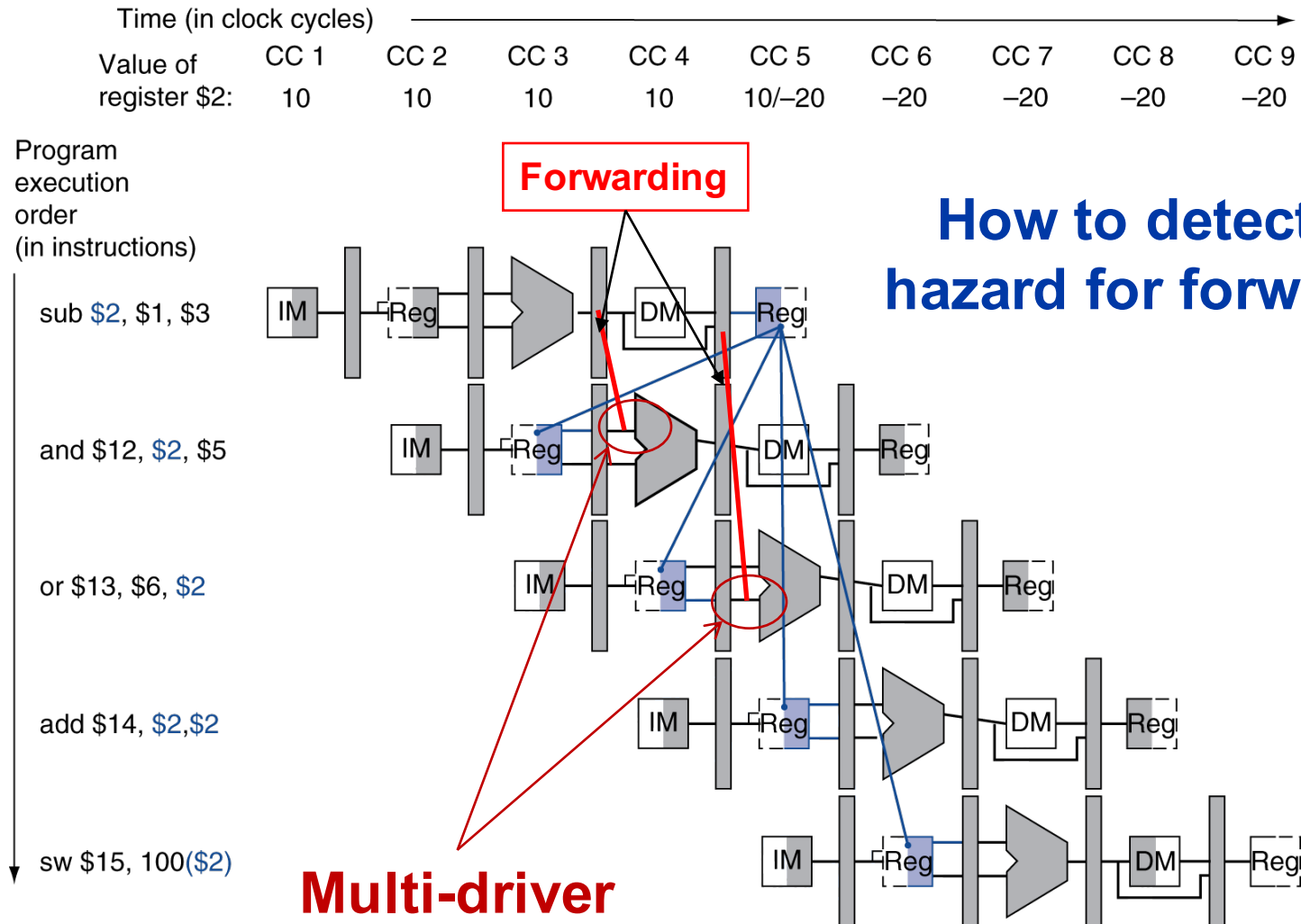
**Can't connect input and output of a component directly**

# Forwarding from Registers



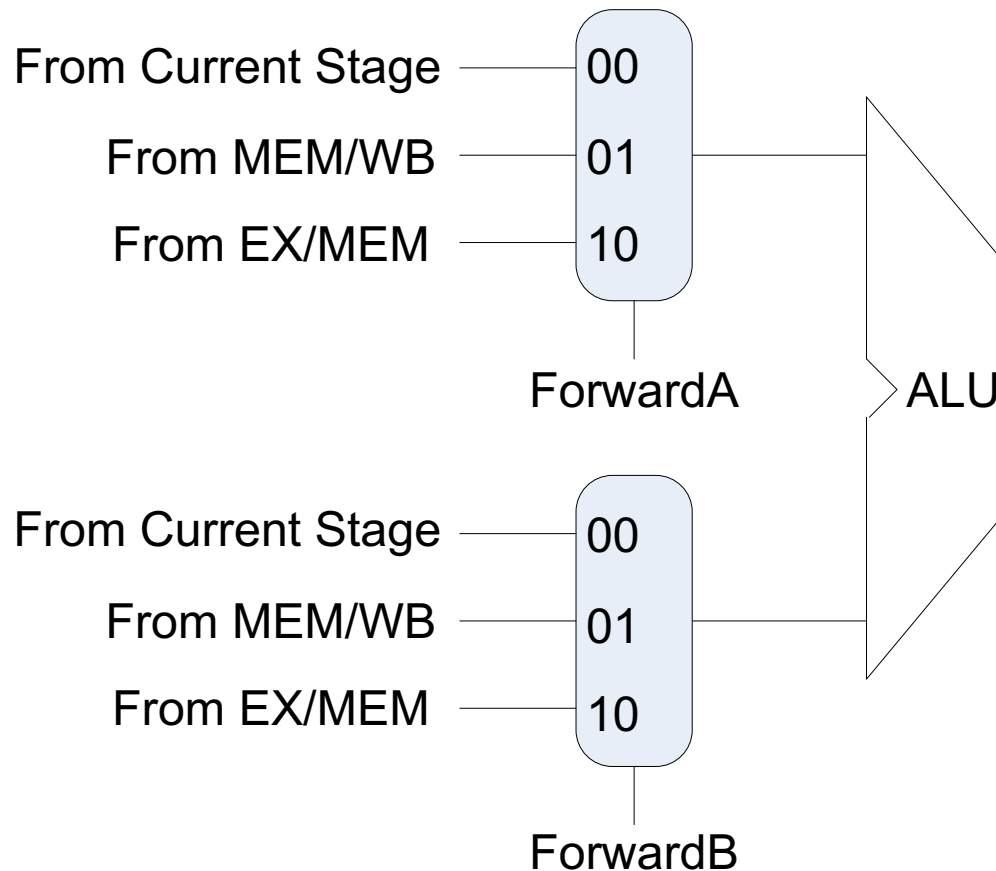


# Dependencies & Forwarding



# Forwarding Path

- Forwarding paths are created between stage pipeline registers and ALU inputs
  - By using MUXes



# Detect the Condition for Forwarding

- Denotation:
  - E.g.  $ID/EX.RegisterRs$  = register number for  $Rs$  residing in  $ID/EX$  pipeline register
- ALU operand register numbers in  $EX$  stage are given by
  - $ID/EX.RegisterRs$ ,  $ID/EX.RegisterRt$
- Data hazard conditions include
  - 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

- Only if an earlier instruction will write back to a register, determined by
  - EX/MEM.RegWrite
  - MEM/WB.RegWrite
- And only if Rd for that instruction is not \$zero (constant register), determined by
  - EX/MEM.RegisterRd  $\neq$  0
  - MEM/WB.RegisterRd  $\neq$  0

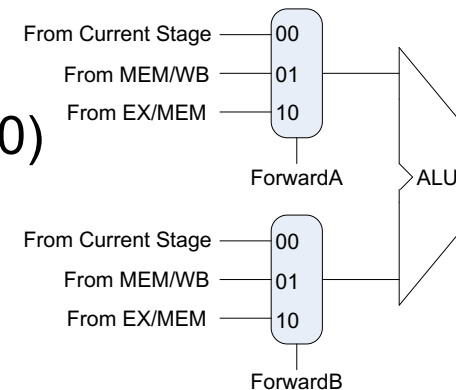
# Revised Forwarding Conditions

## EX hazard

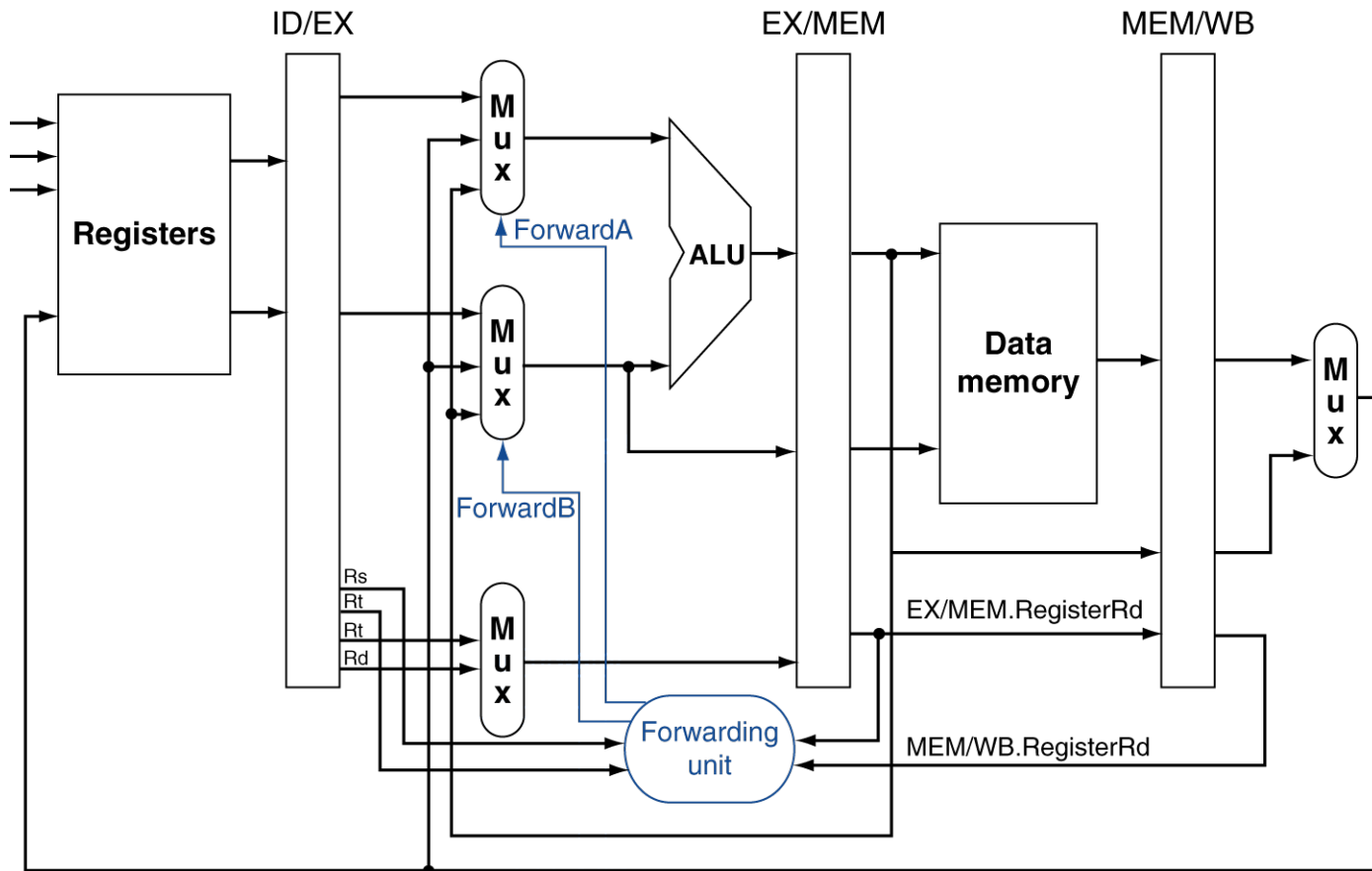
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))  
MUX select signal ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd  $\neq$  0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))  
MUX select signal ForwardB = 10

## MEM hazard

- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))  
MUX select signal ForwardA = 01
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd  $\neq$  0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))  
MUX select signal ForwardB = 01

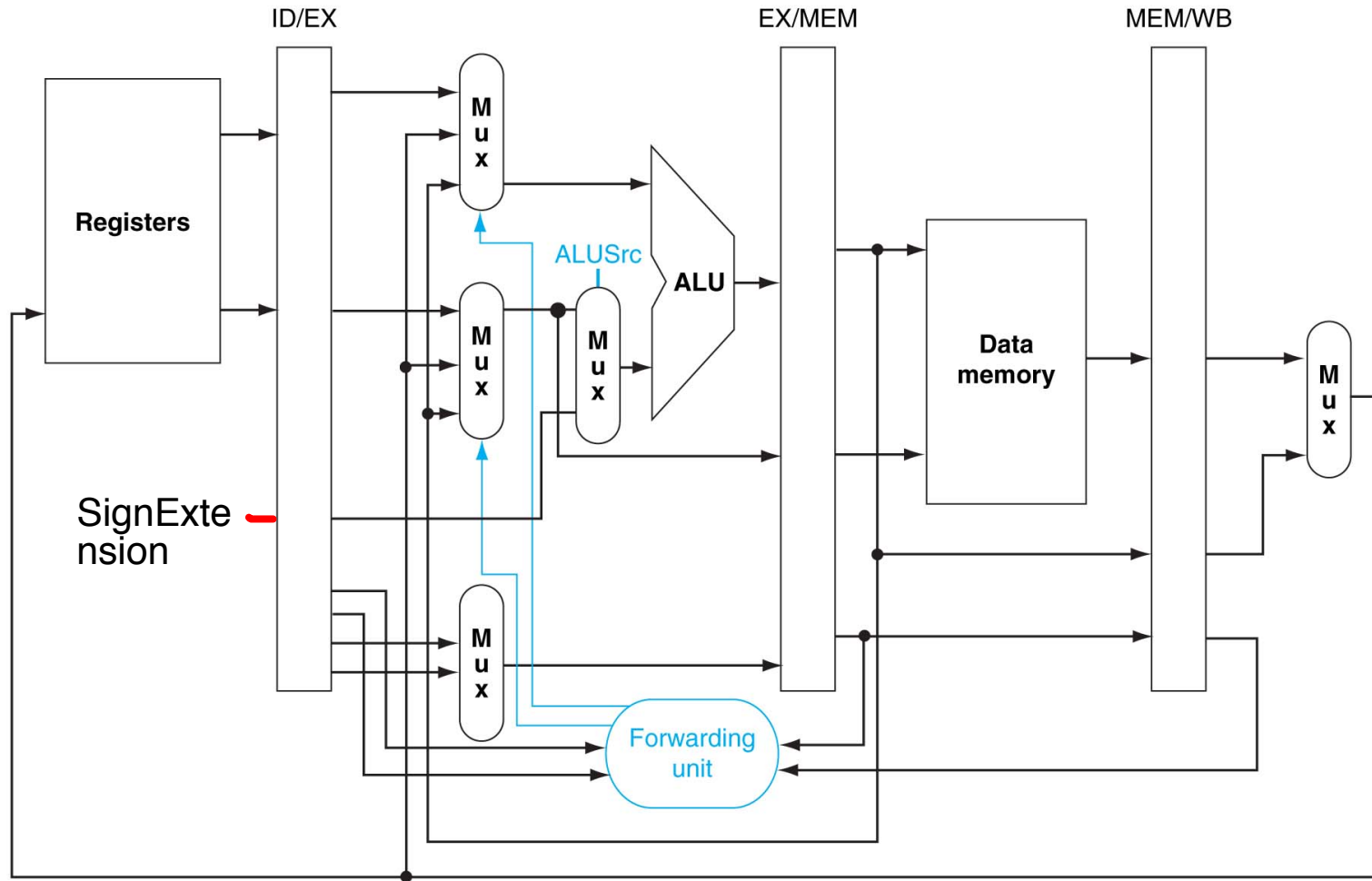


# Forwarding Paths



Forwarding Unit in **EX** stage

# Forwarding Paths with ALUSrc



# Double Data Hazard

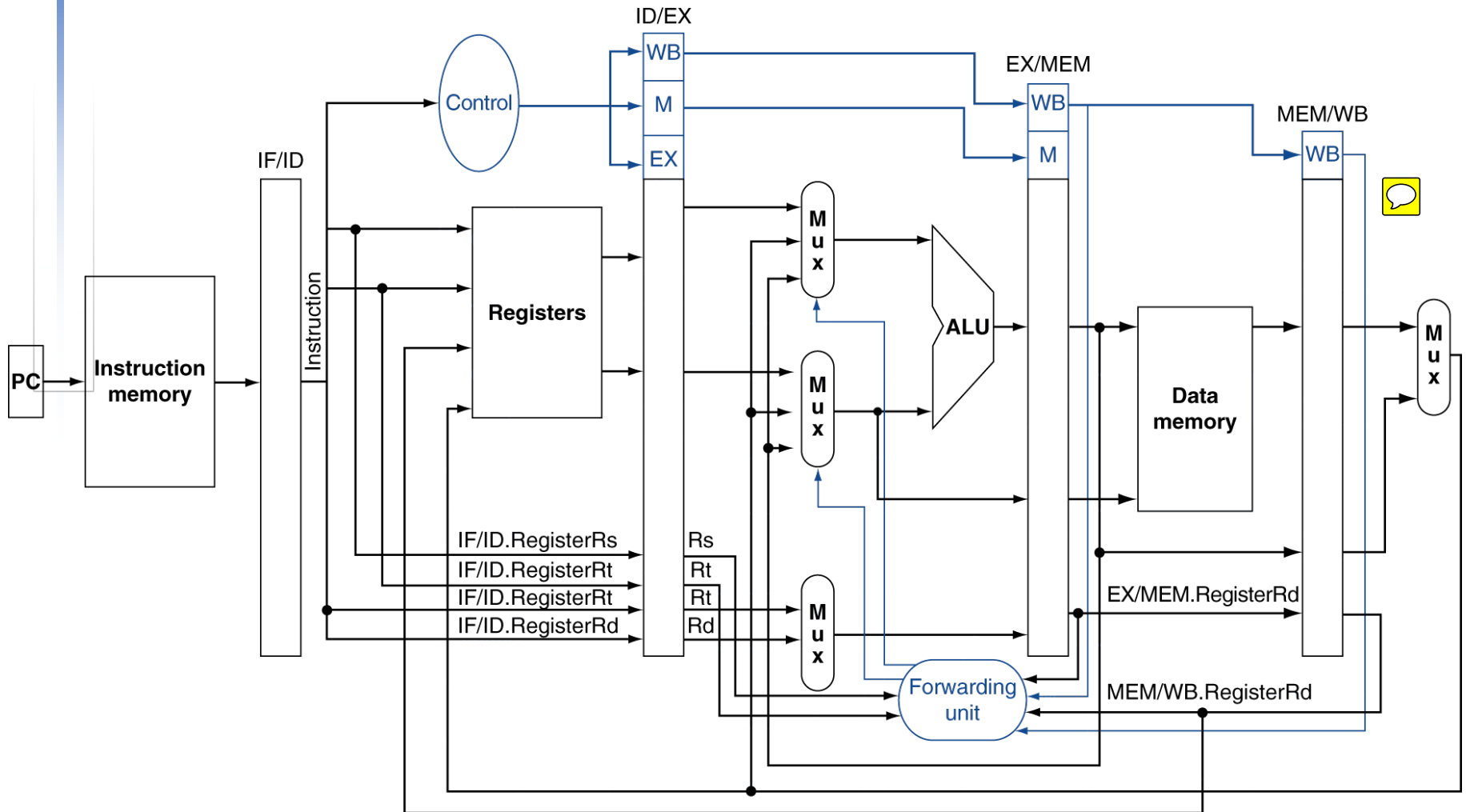
- Consider the sequence:
  - add \$1, \$1, \$2
  - add \$1, \$1, \$3
  - add \$1, \$1, \$4
- Conditions for both hazards are satisfied
  - Want to use the most recent
  - Forwarding between 1<sup>st</sup> and 3<sup>rd</sup> – MEM hazard condition should be disabled
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true



# Revised Forwarding Condition

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

# Datapath with Forwarding Unit



# Load-Use Data Hazard

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

# Load-Use Data Hazard

Program  
execution  
order  
(in instructions)

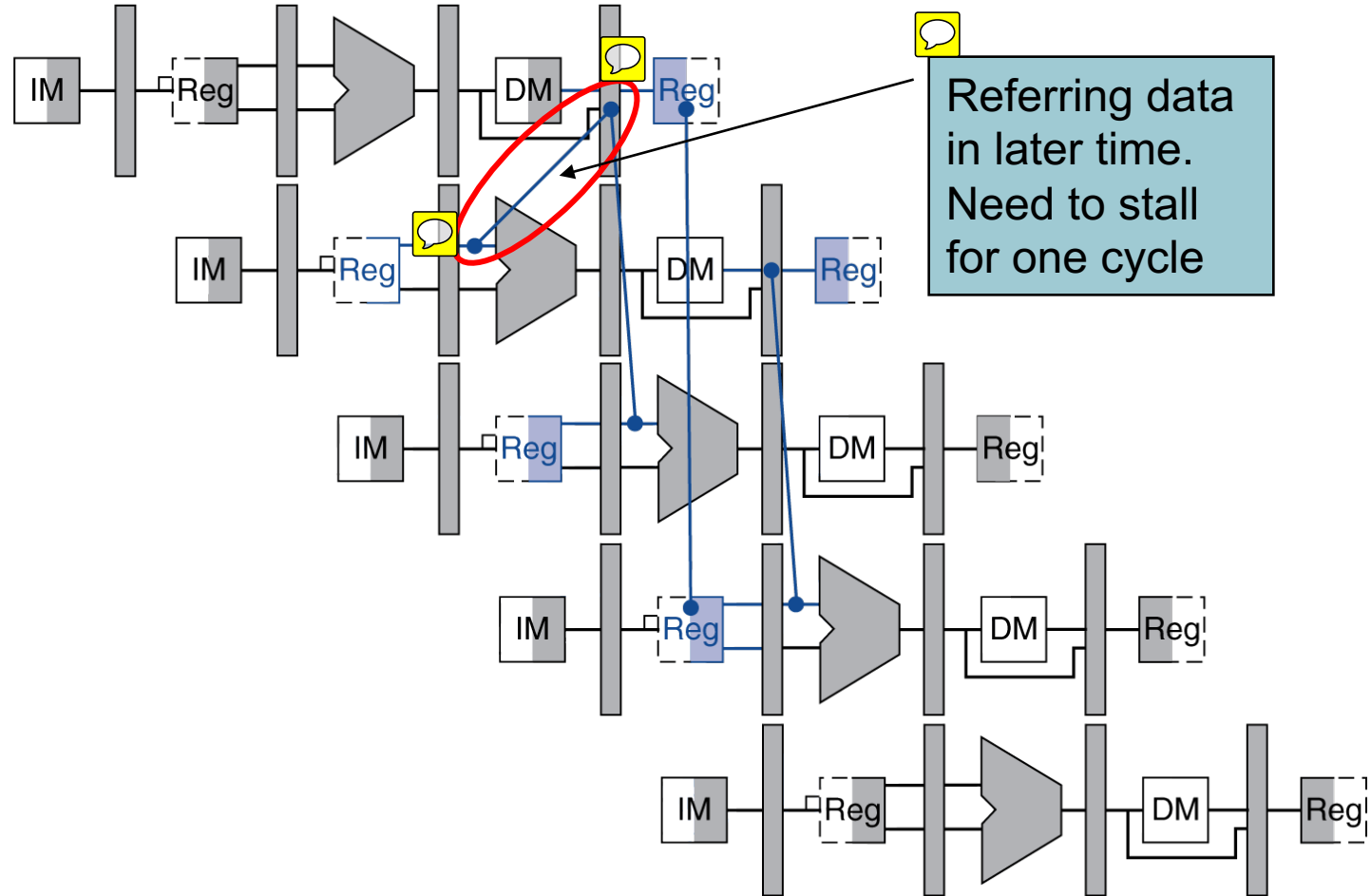
lw \$2, 20(\$1)

and \$4, \$2, \$5

or \$8, \$2, \$6

add \$9, \$4, \$2

slt \$1, \$6, \$7

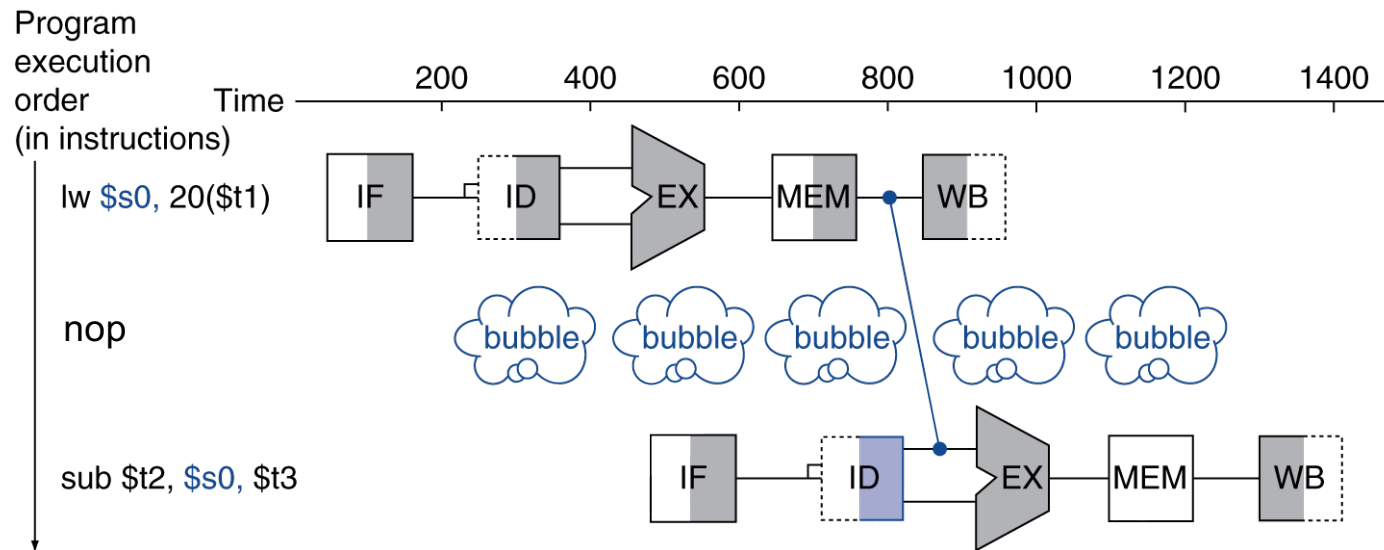


# Load-Use Hazard Detection

- Need to check only for a ***load*** instruction, determined by
  - ID/EX.MemRead
- Load-use hazard when
  - ID/EX.MemRead and  
((ID/EX.RegisterRt == IF/ID.RegisterRs) or  
(ID/EX.RegisterRt == IF/ID.RegisterRt))
- Hazard Detection is in **ID** stage
- If detected, stall and insert bubble

# Load-Use Data Hazard

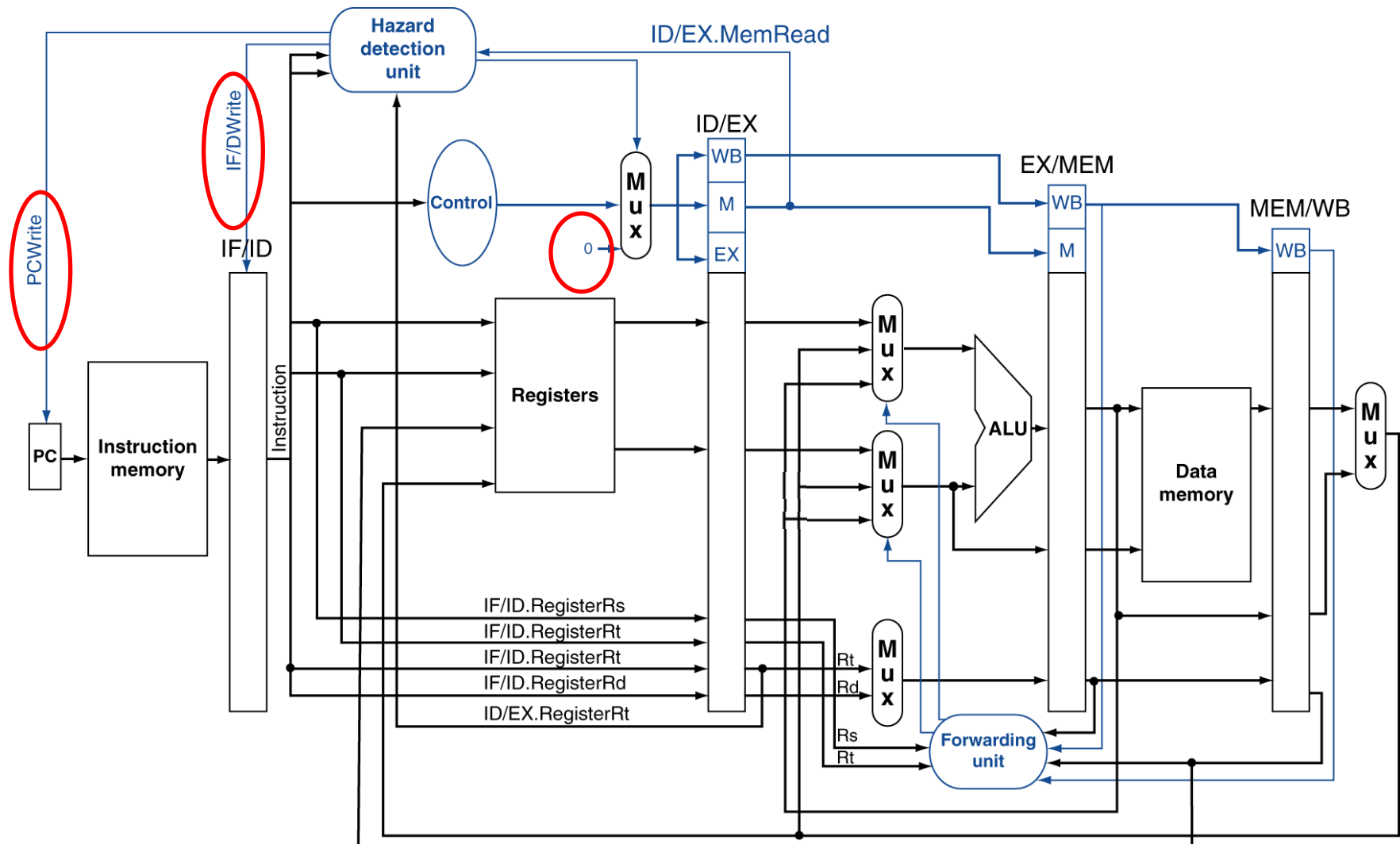
- Resolve by inserting stalls



# How to Stall the Pipeline?

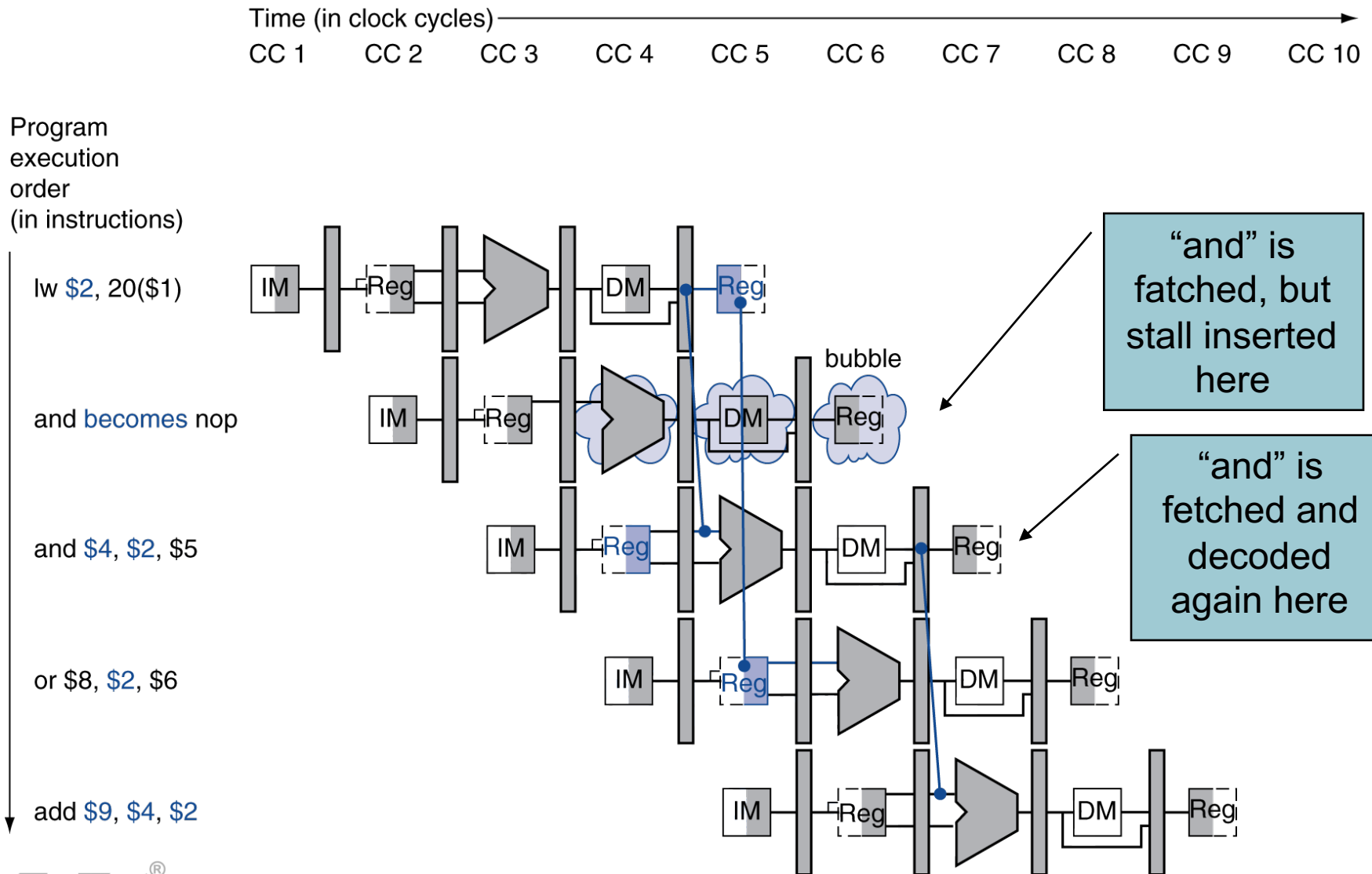
- Force control signals in ID/EX register to 0's
  - EX, MEM and WB in following cycles do no-operation (nop) with those 0 control signals
- Prevent updates of PC and IF/ID registers
  - Instruction in IF/ID is held and decoded again
  - PC is held, so the same instruction is fetched again

# Datapath with Hazard Detection





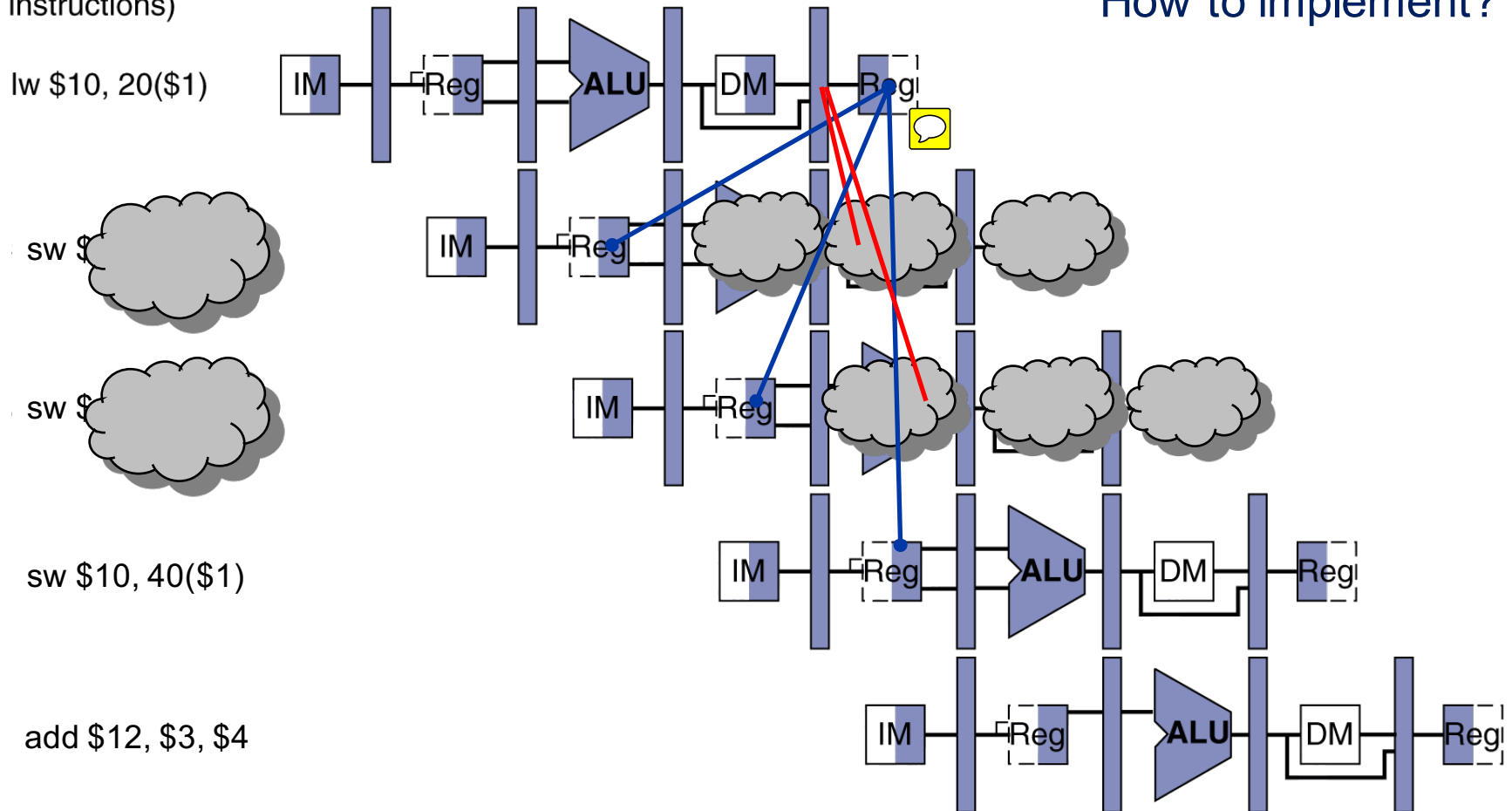
# Stall/Bubble in the Pipeline



# Question

(in instructions)

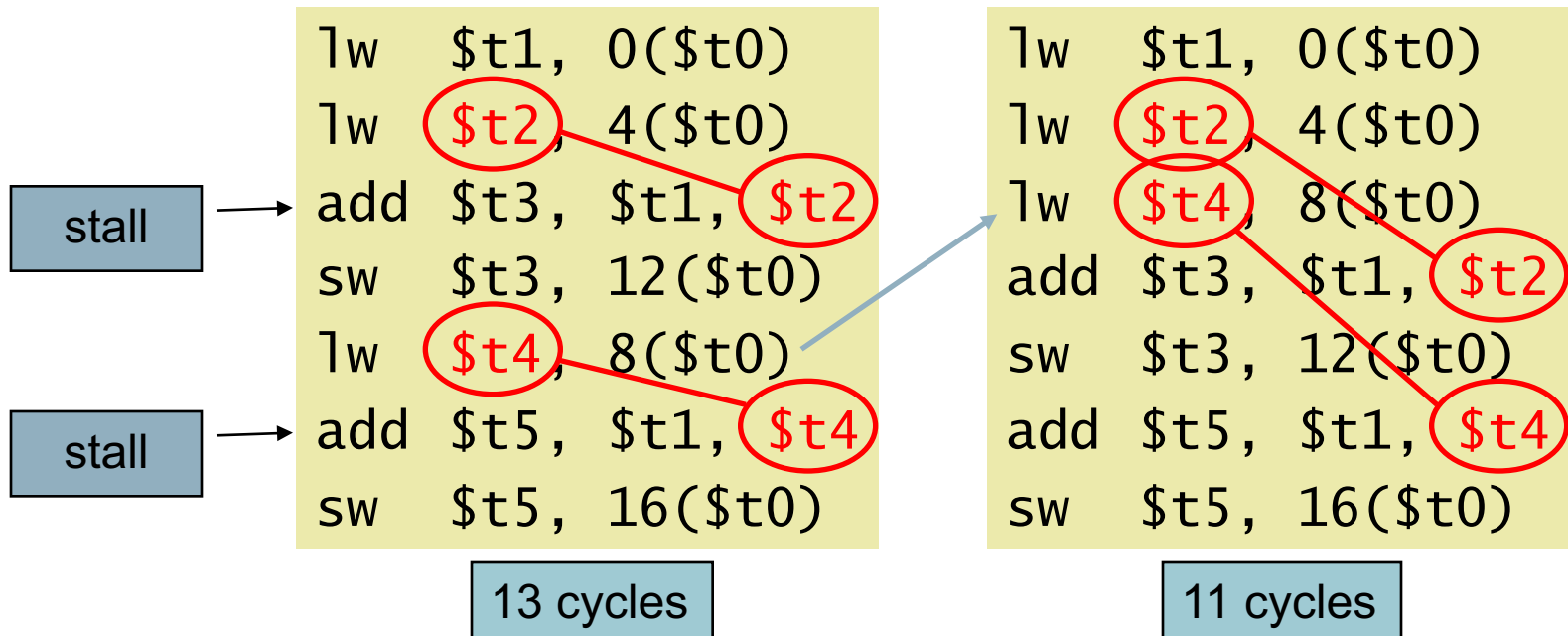
How to implement?



This delays the execution too much, how to fix?

# Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for  $A = B + E$ ;  $C = B + F$ ;



# Stalls and Performance

## The BIG Picture

- Stalls reduce performance
  - But are required to get correct results
- Hardware can be improved to take care of the hazards automatically
  - forwarding
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure