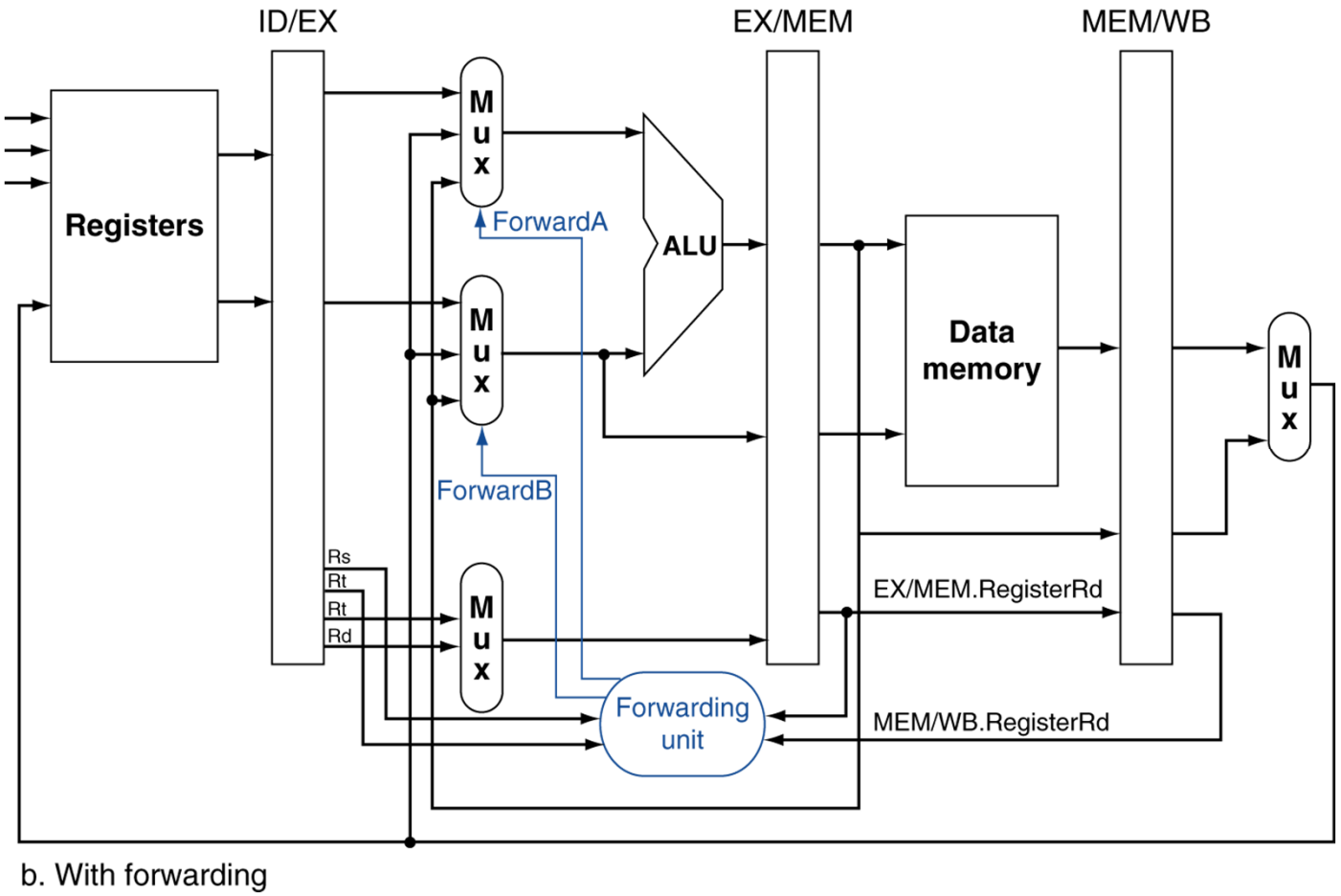
COMPUTER ORGANIZATION (IS F242)

LECT 44: PIPELINING

Forwarding Paths



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

- ForwardA=00
  - The first ALU operand comes from the register file
- ForwardA=10
  - The first ALU operand is forwarded from the prior ALU result
- ForwardA=01
  - The first ALU operand is forwarded from data memory or an earlier ALU result
- ForwardB=00
  - The second ALU operand comes from the register file
- ForwardB=10
  - The second ALU operand is forwarded from the prior ALU result
- ForwardB=01
  - The second ALU operand is forwarded from data memory or an earlier ALU result

Double Data Hazard

- Consider the sequence:

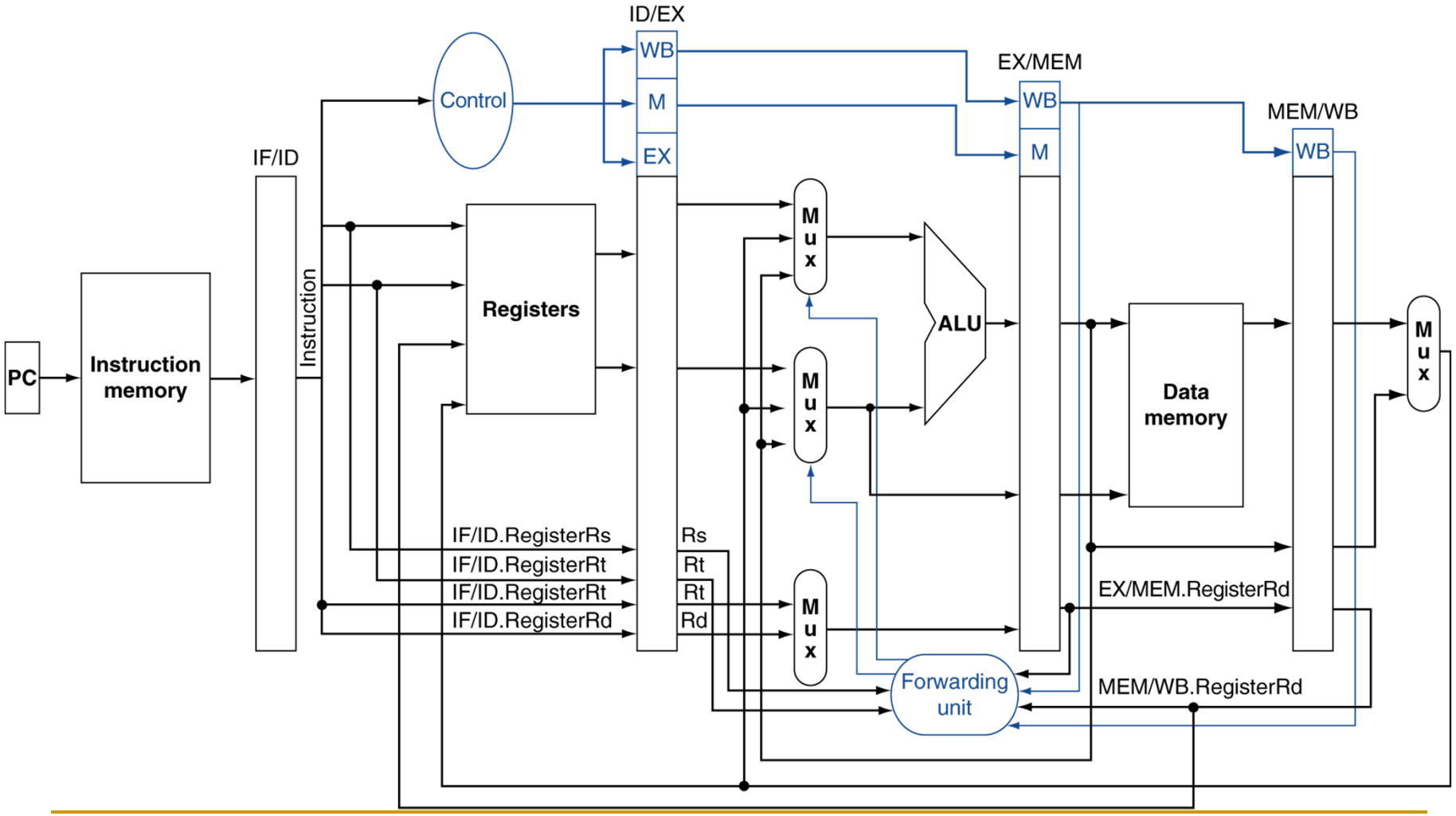
```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- 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

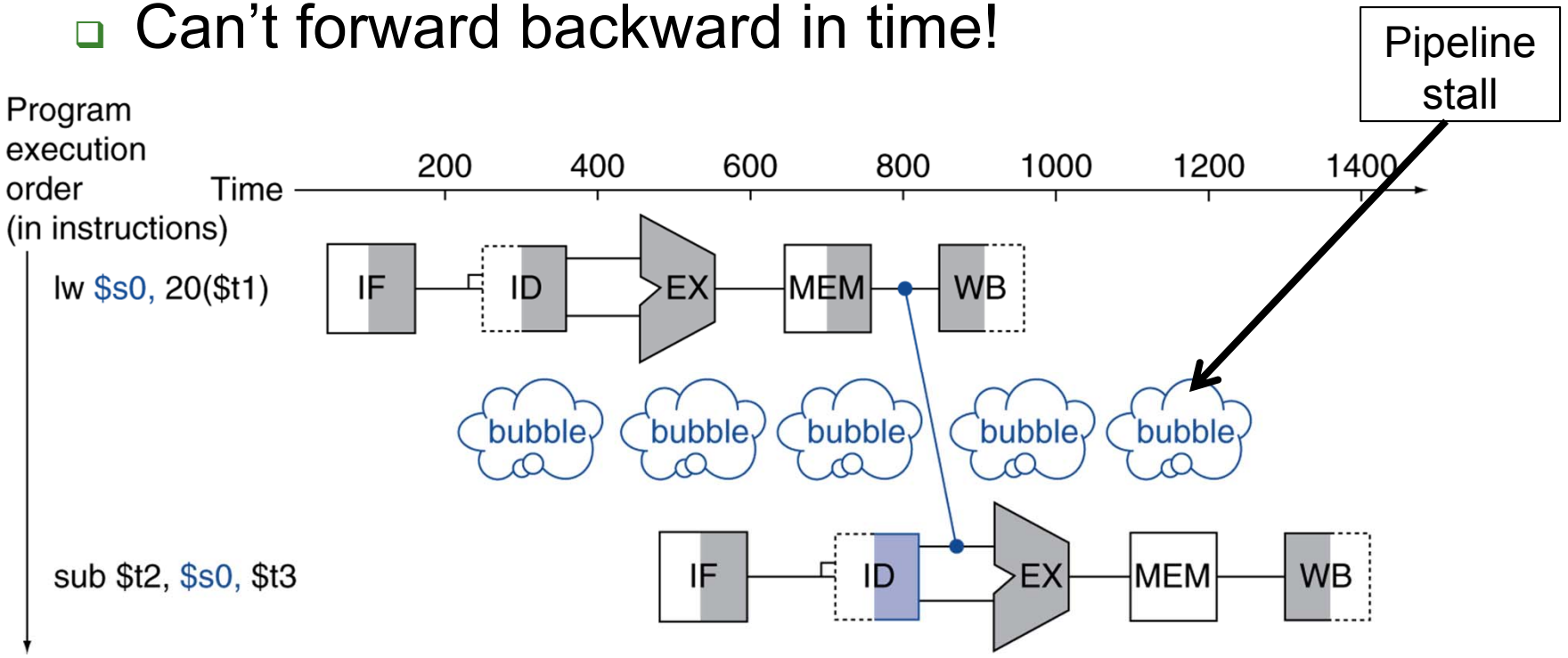
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite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

Datapath with Forwarding

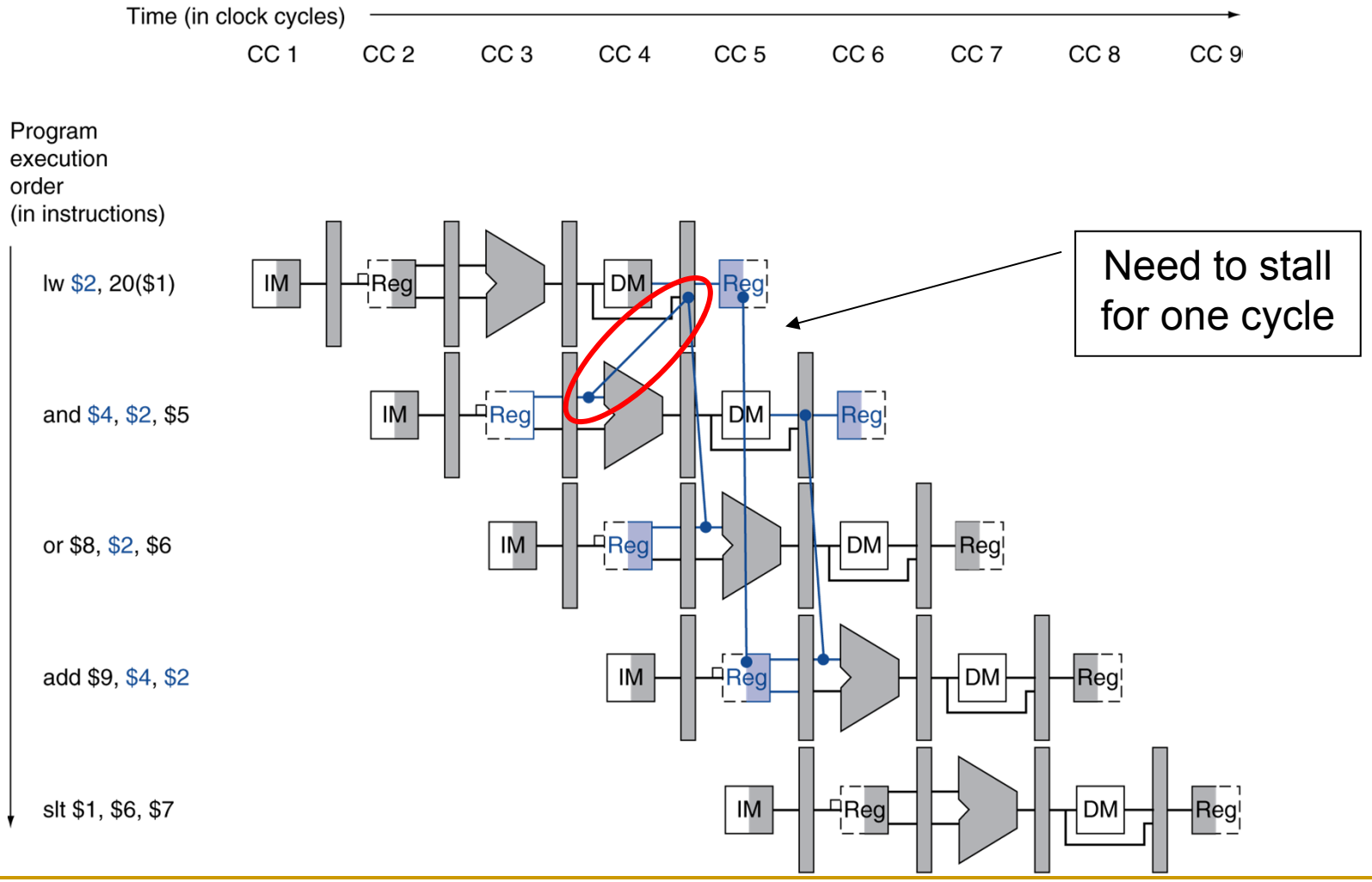


Load-Use Data Hazard

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



Load-Use Data Hazard



Load-Use Hazard Detection

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

    Stall the pipeline

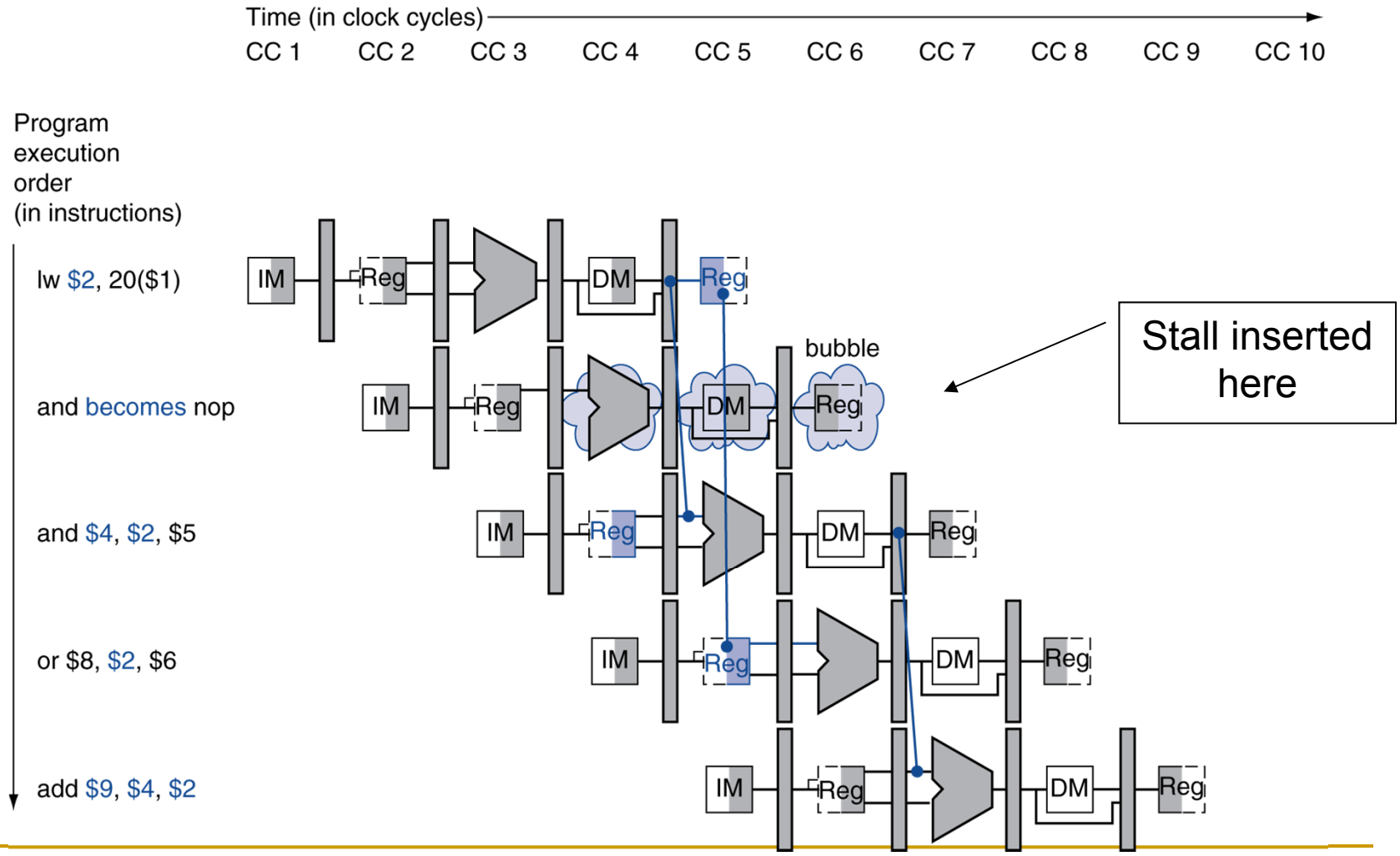
Stall an instruction in ID stage

- If the instruction in ID stage is stalled the instruction in IF stage must also be stalled
  - Otherwise we will lose the fetched instruction
- How can we achieve this??

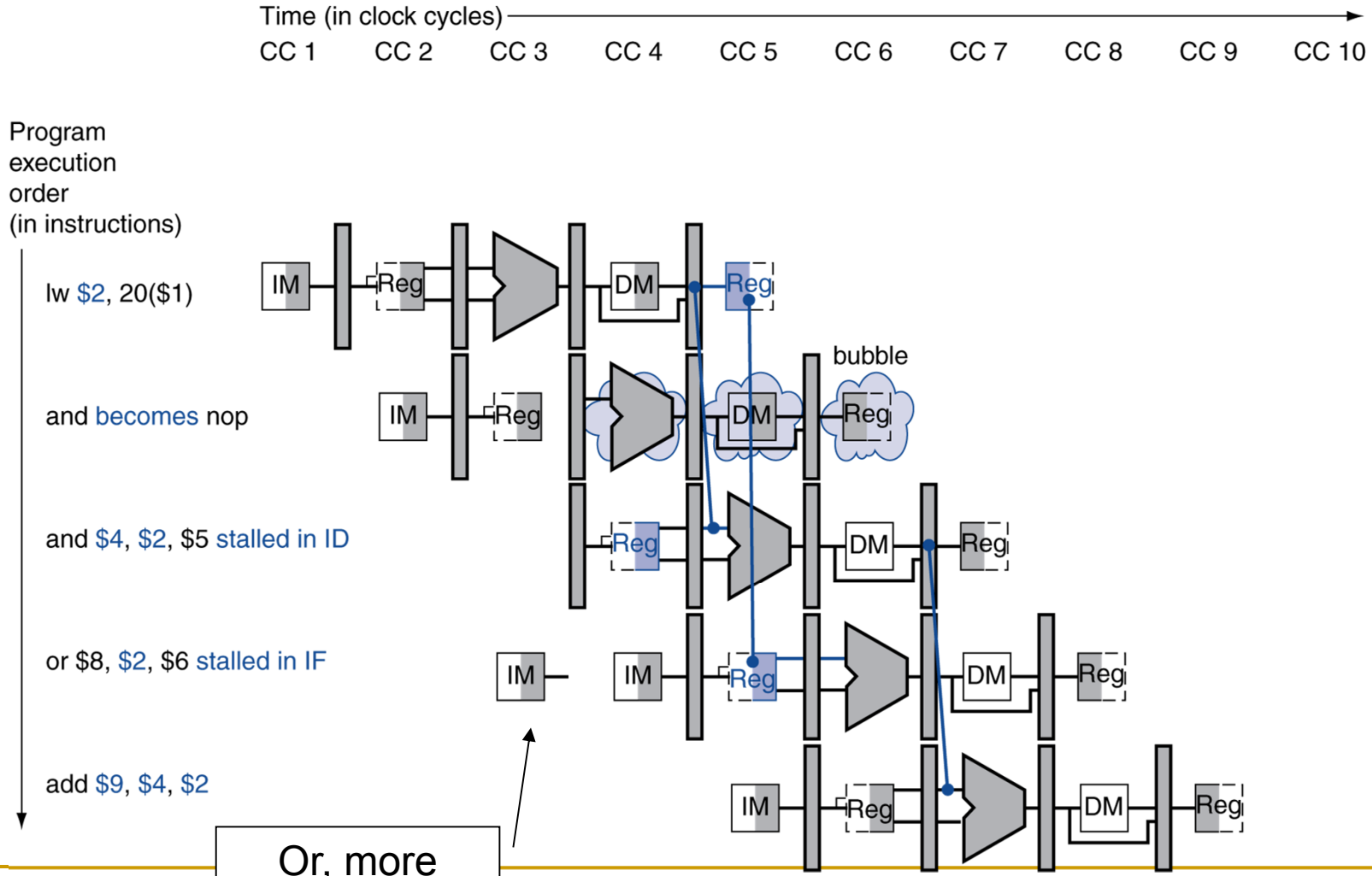
How to Stall the Pipeline

- 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

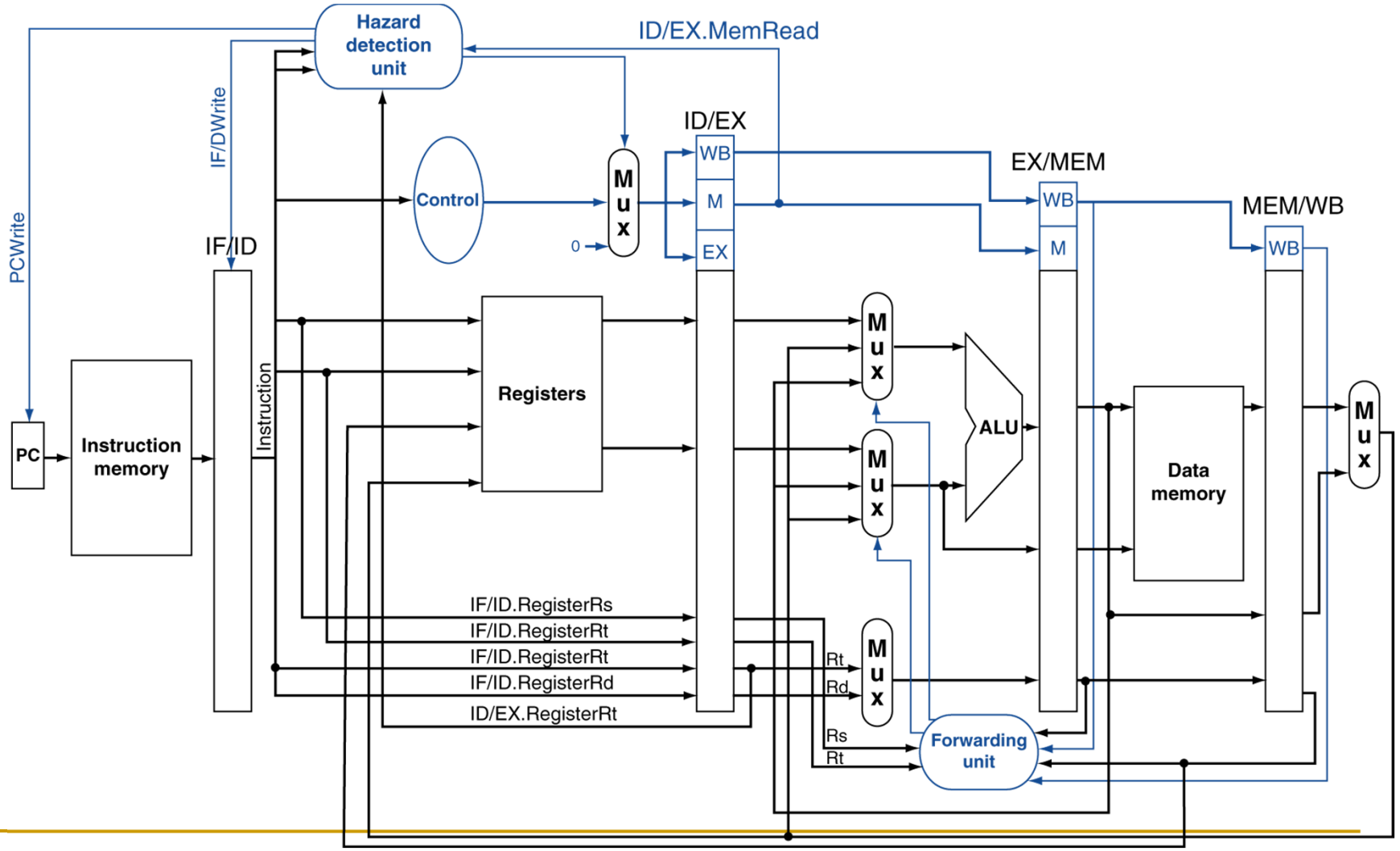
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline



Datapath with Hazard Detection



Stalls and Performance

- 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

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;`