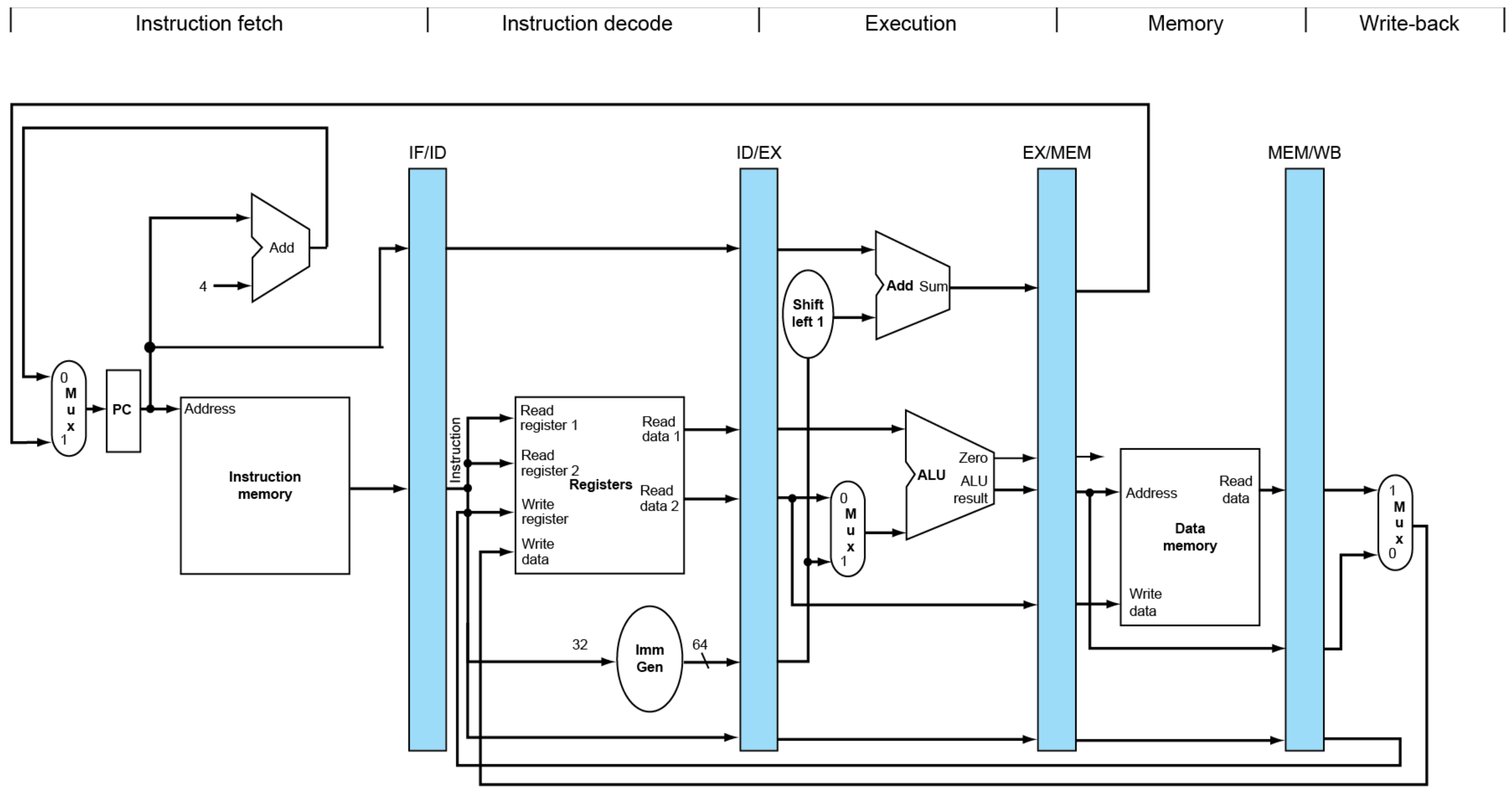
Pipelined CPU

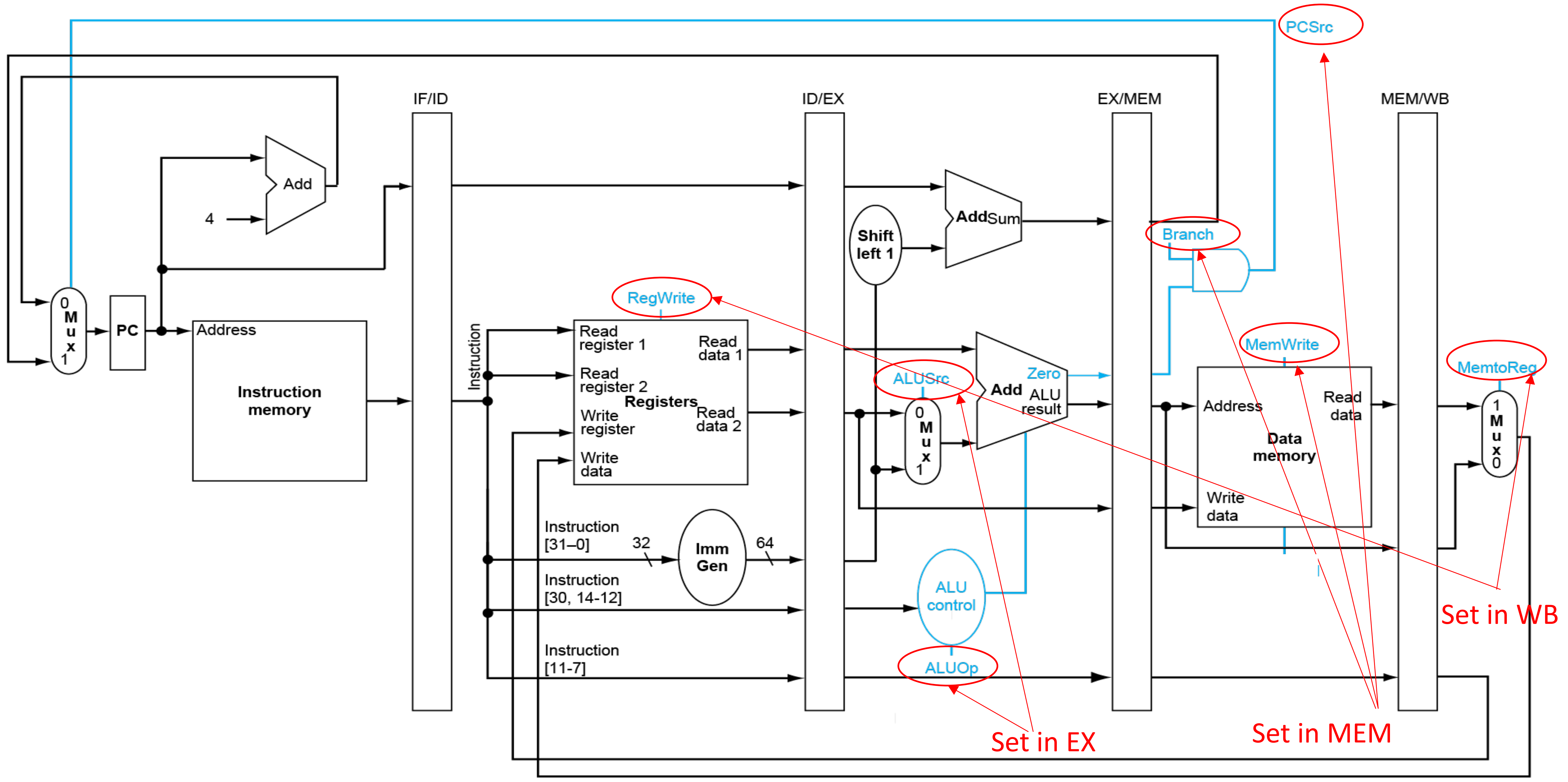
Jinyang Li

Based on the slides of Patterson and Hennessy

What we've learnt: basic 5-stage pipeline (datapath)

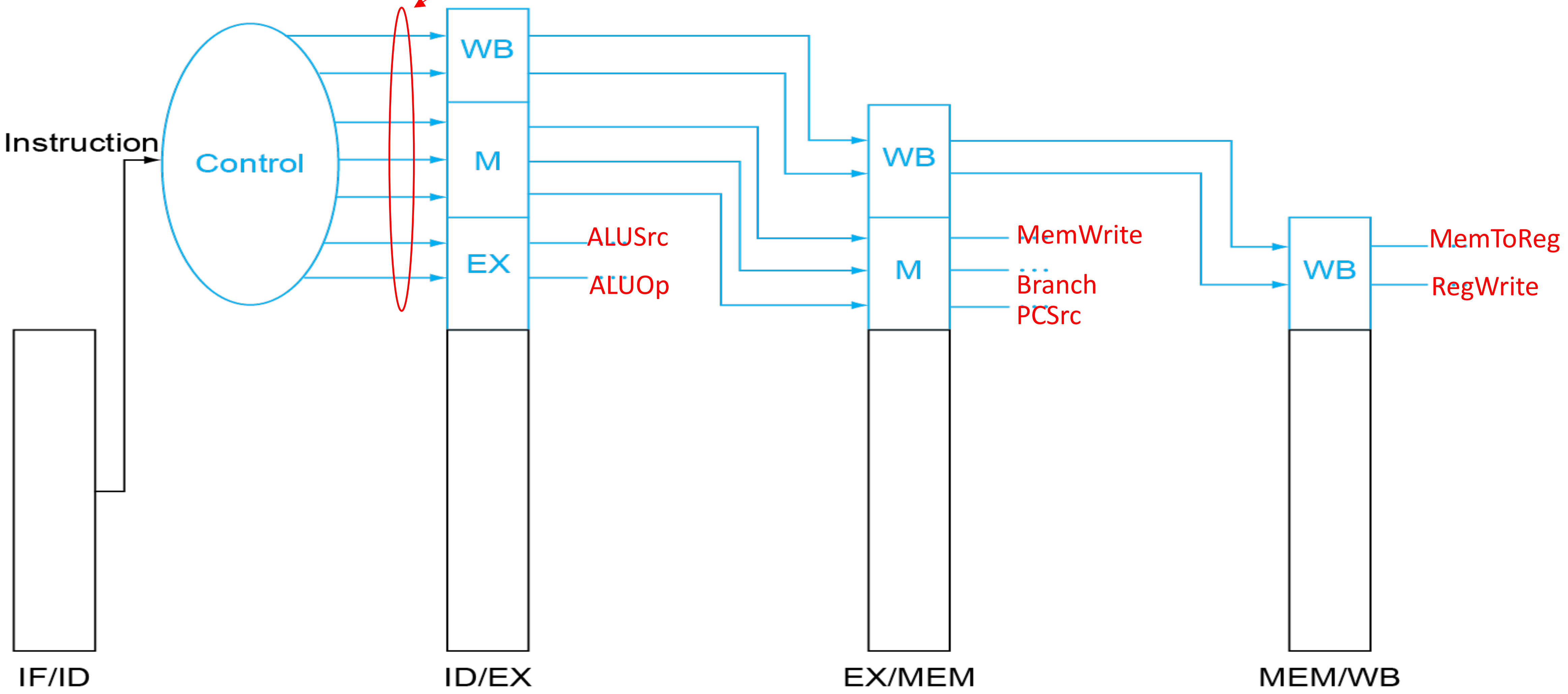


Pipelined Control (Simplified)



Pipelined Control

Control signals derived from instruction, same as in single-cycle implementation



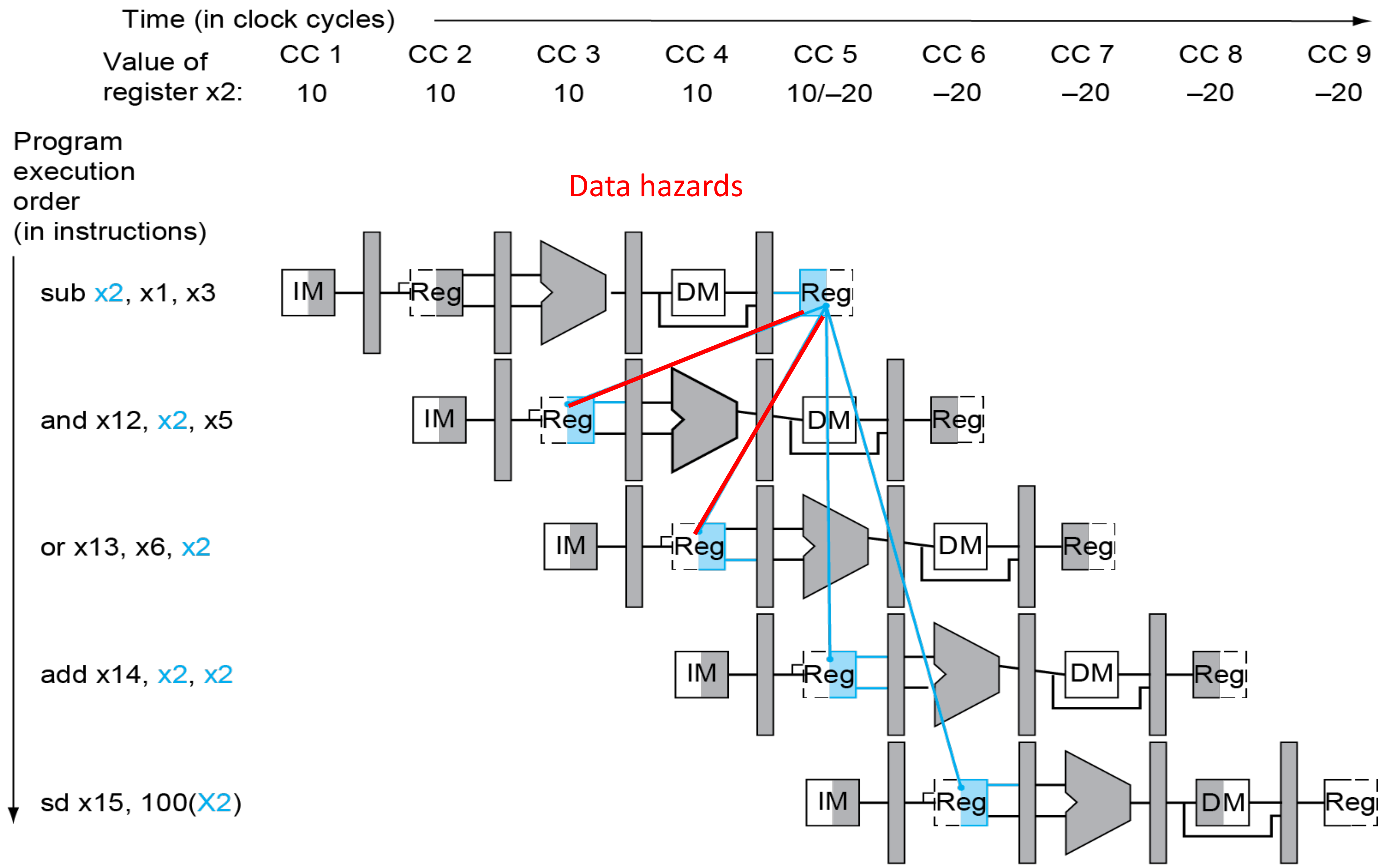
Data Hazards in ALU Instructions

- Example instruction sequence:

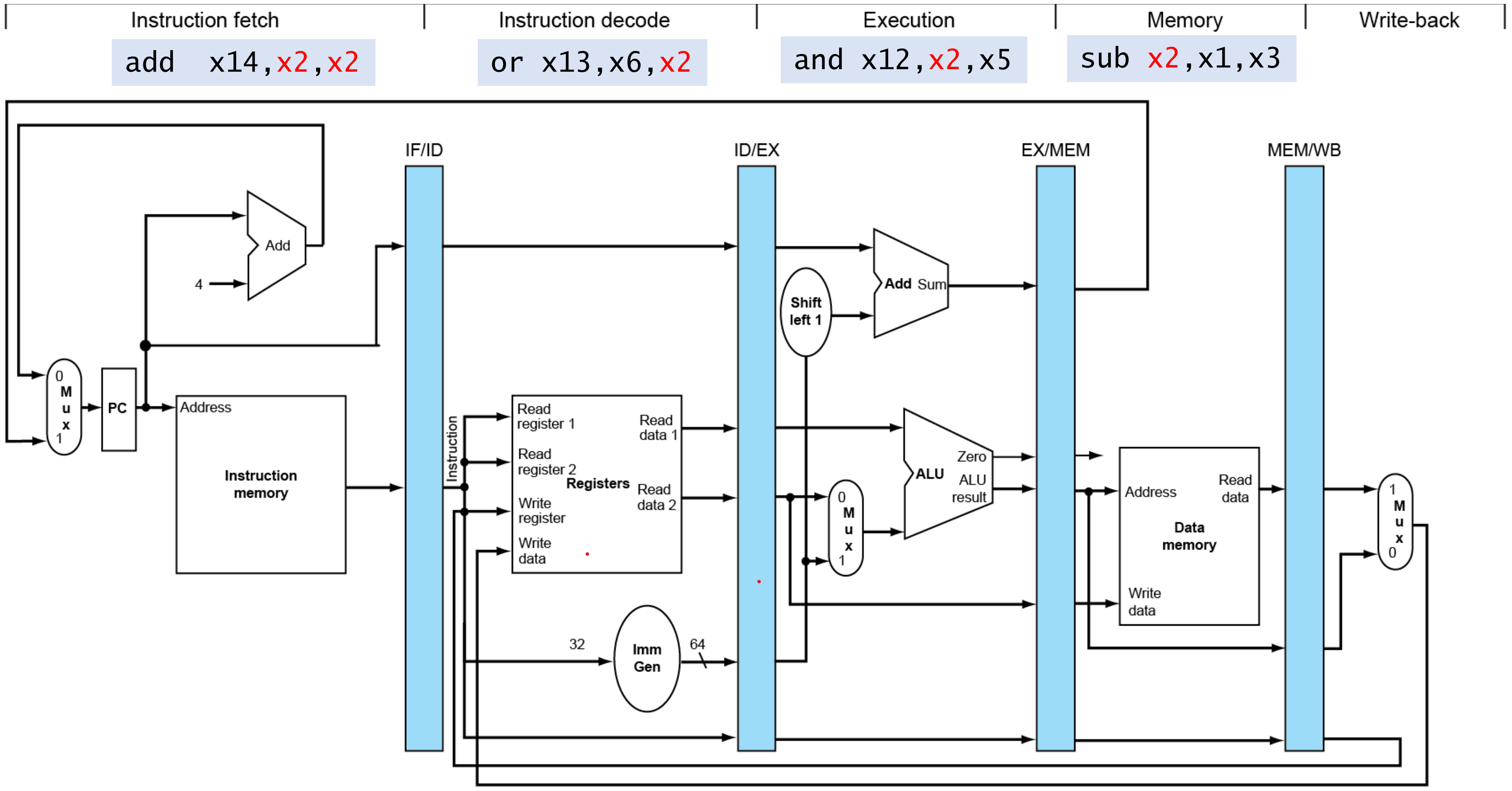
```
sub   x2, x1,x3
and   x12,x2,x5
or    x13,x6,x2
add   x14,x2,x2
sd    x15,100(x2)
```

- Solution: forwarding (aka bypassing)
  - Use result after it’s computed; don’t wait for it to be stored in register

Data hazard in ALU instructions

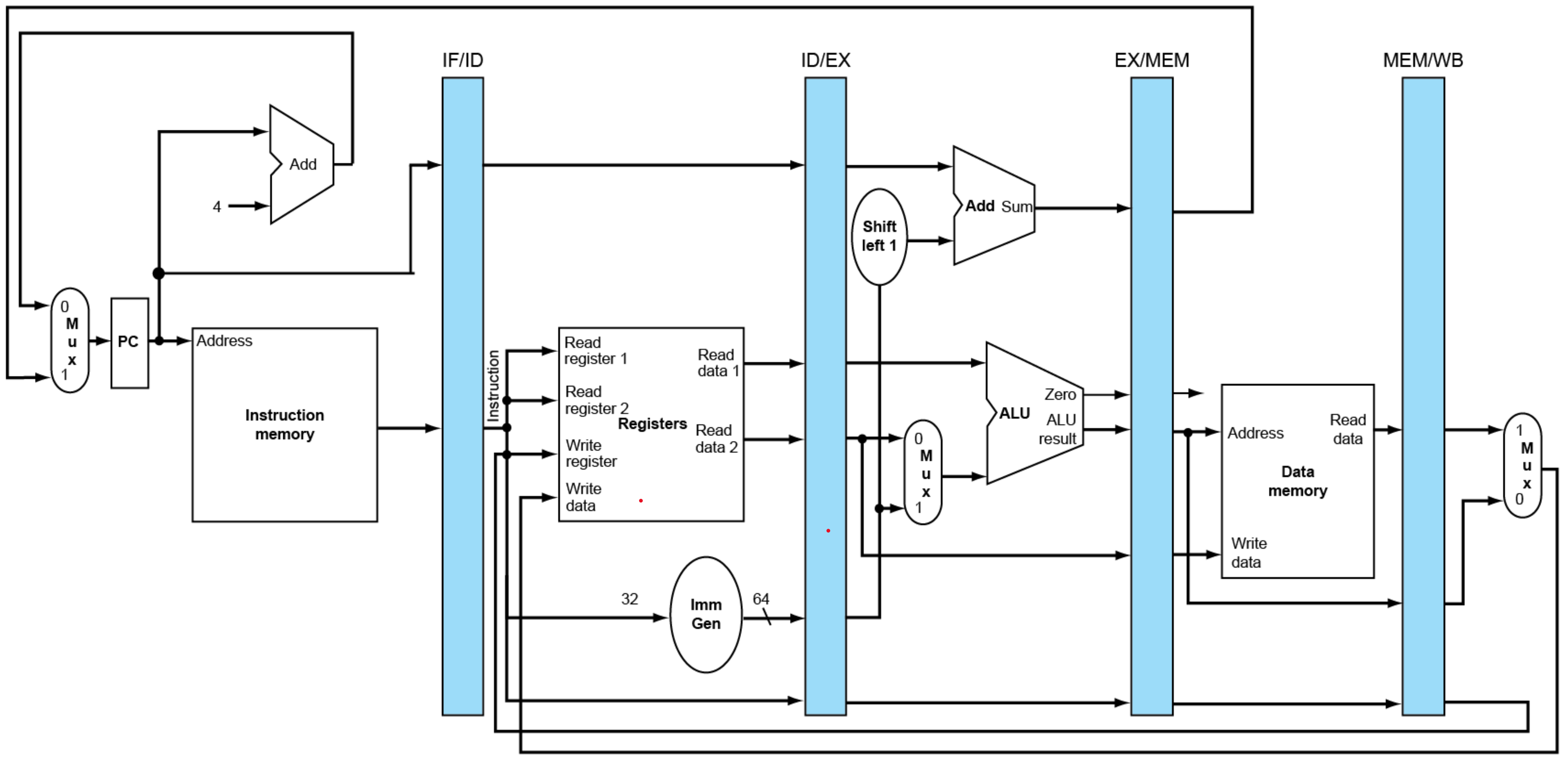


Using forwarding to resolve data hazard (cycle-4)

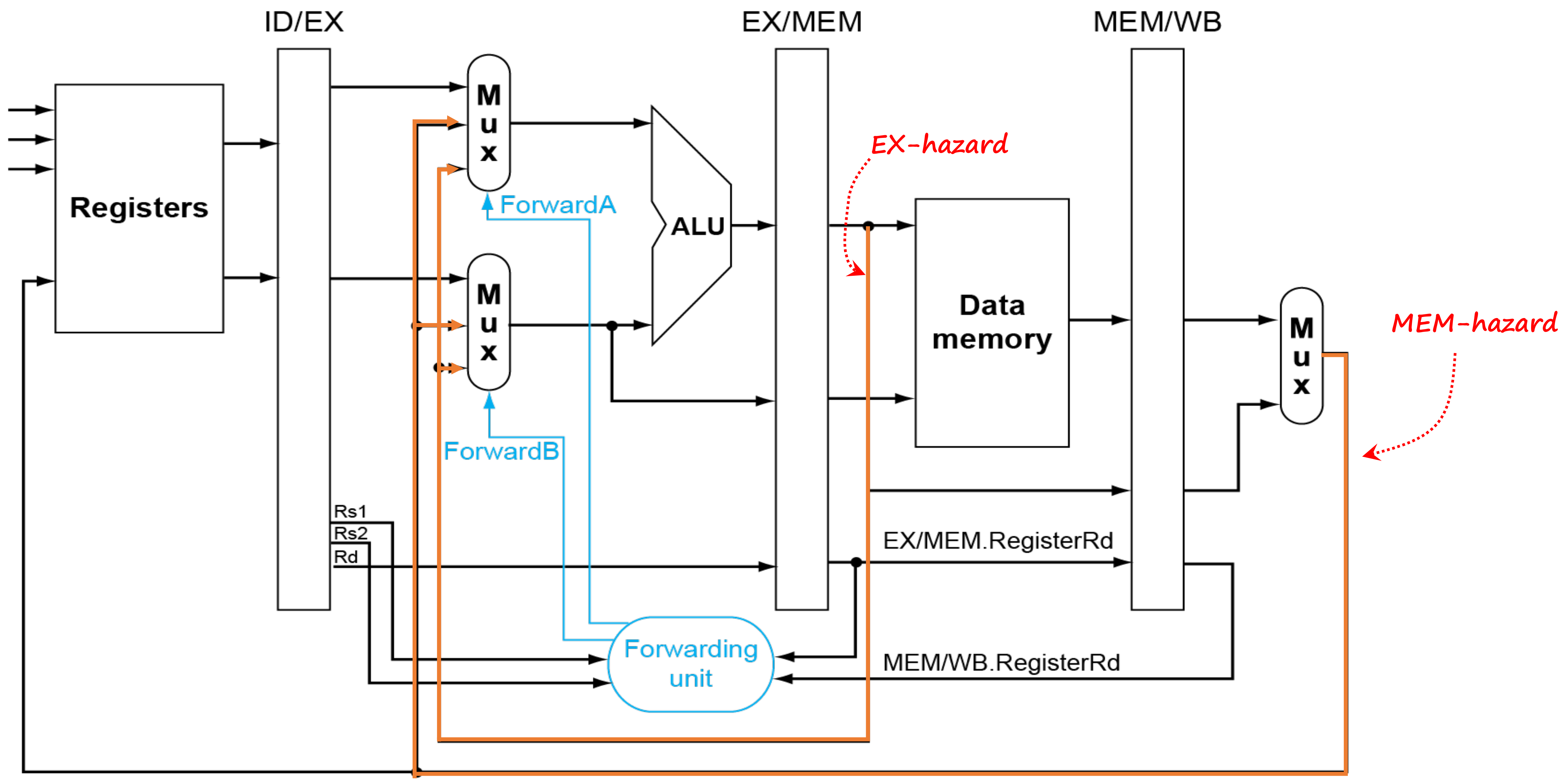


Using forwarding to resolve data hazard (cycle-5)

Instruction fetch	Instruction decode	Execution	Memory	Write-back
`sd x15,100(x2)`	`add x14,x2,x2`	`or x13,x6,x2`	`and x12,x2,x5`	`sub x2,x1,x3`



Forwarding Paths



Detecting when to use forwarded data

rd rs1 rs2

- Pass register numbers along pipeline
  - e.g., ALU operand register numbers (in EX stage) are:
    - ID/EX.RegisterRs1, ID/EX.RegisterRs2 Rd
- Data hazards when

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1
1b. EX/MEM.RegisterRd = ID/EX.RegisterRs2

} EX-hazard: Fwd from EX/MEM pipeline reg

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs1
2b. MEM/WB.RegisterRd = ID/EX.RegisterRs2

} Mem-hazard: Fwd from MEM/WB pipeline reg

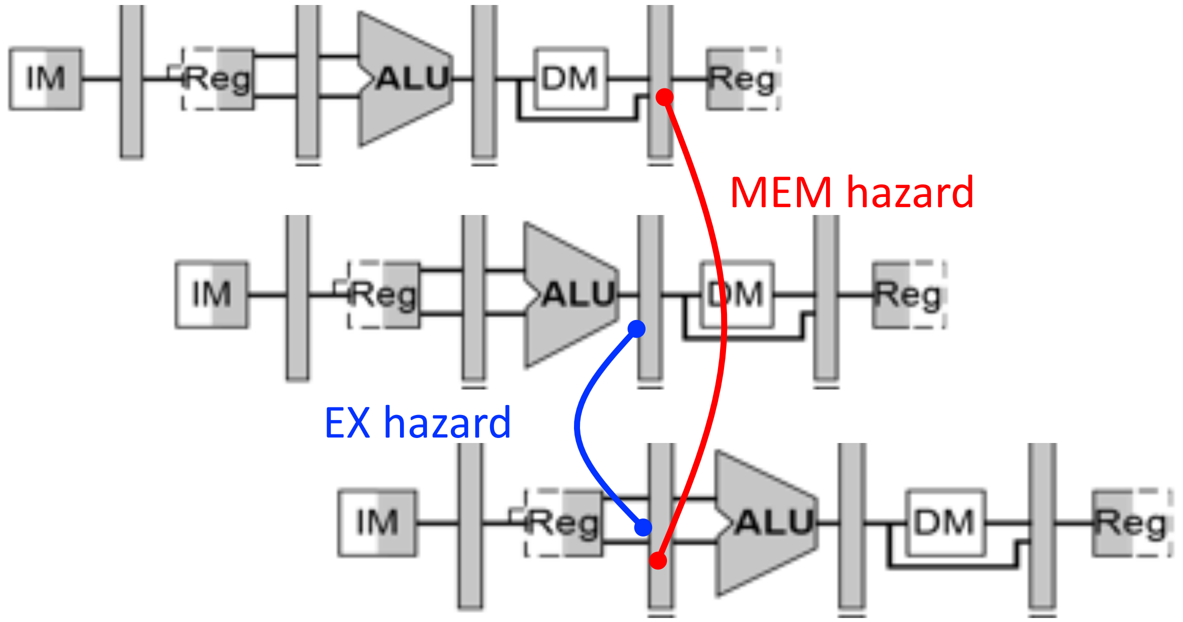
Detecting when to use forwarded data

- 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 x0
  - EX/MEM.RegisterRd ≠ 0,
    MEM/WB.RegisterRd ≠ 0

X0 is always 0 in RISC-V

Double Data Hazard



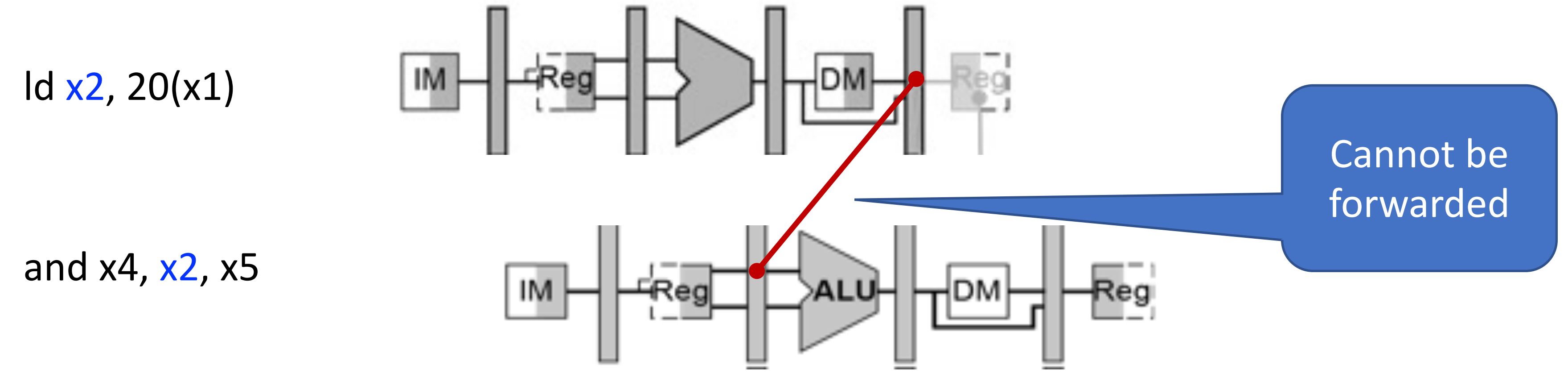
- Consider the sequence:

```
add  x1,x1,x2
add  x1,x1,x3
add  x1,x1,x4
```

- Both EX- Mem-hazards occur
  - use the most recent
  - aka, only fwd Mem-hazard if EX hazard condition isn't true

Load-Use Data hazard Detection

- Load-use hazard cannot be resolved using forwarding alone

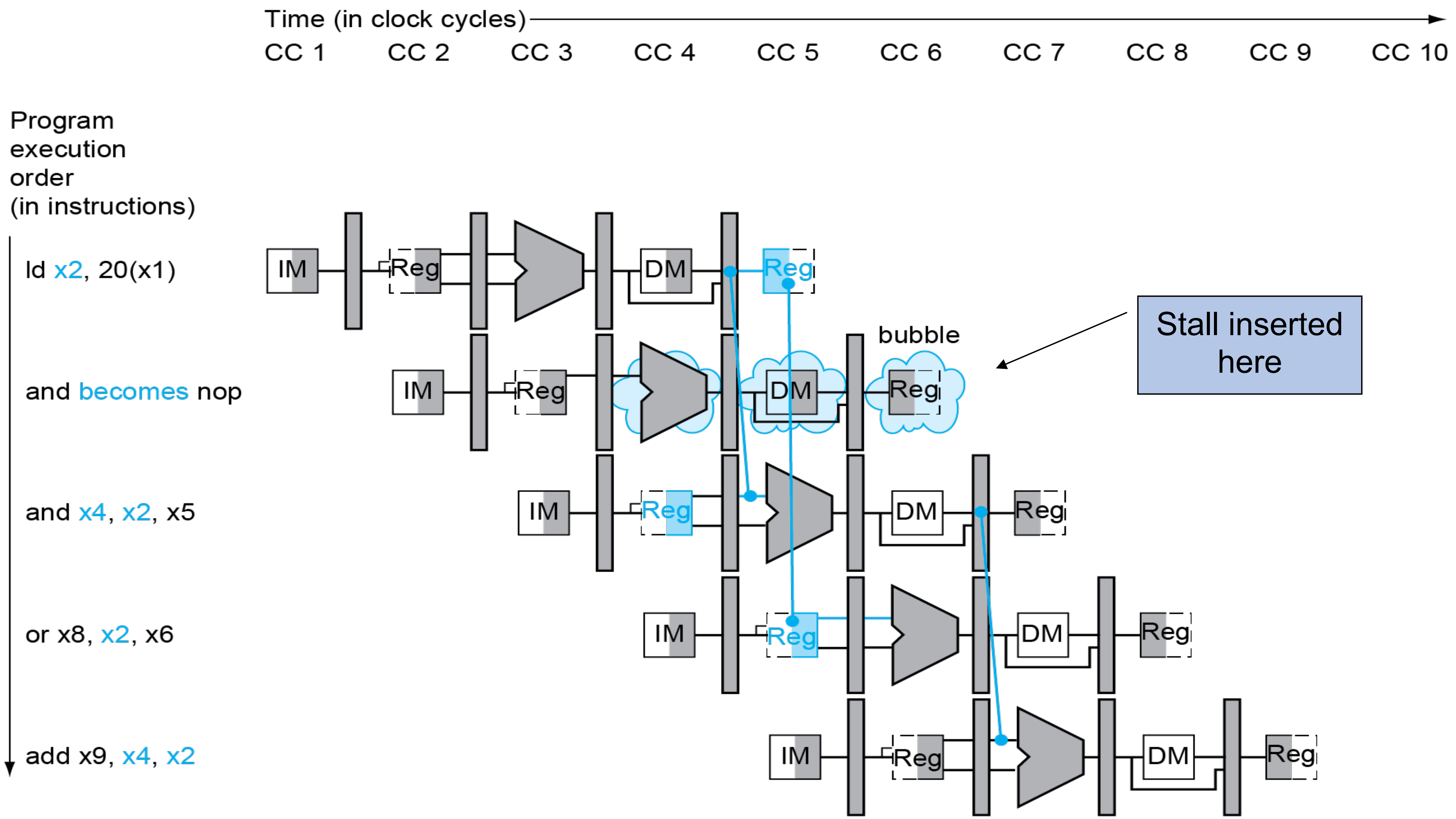


- Check load-use hazard using condition
  - ID/EX.MemRead and ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or (ID/EX.RegisterRd = IF/ID.RegisterRs1))
- If detected, stall and insert bubble

How to Stall the Pipeline (aka insert bubble)

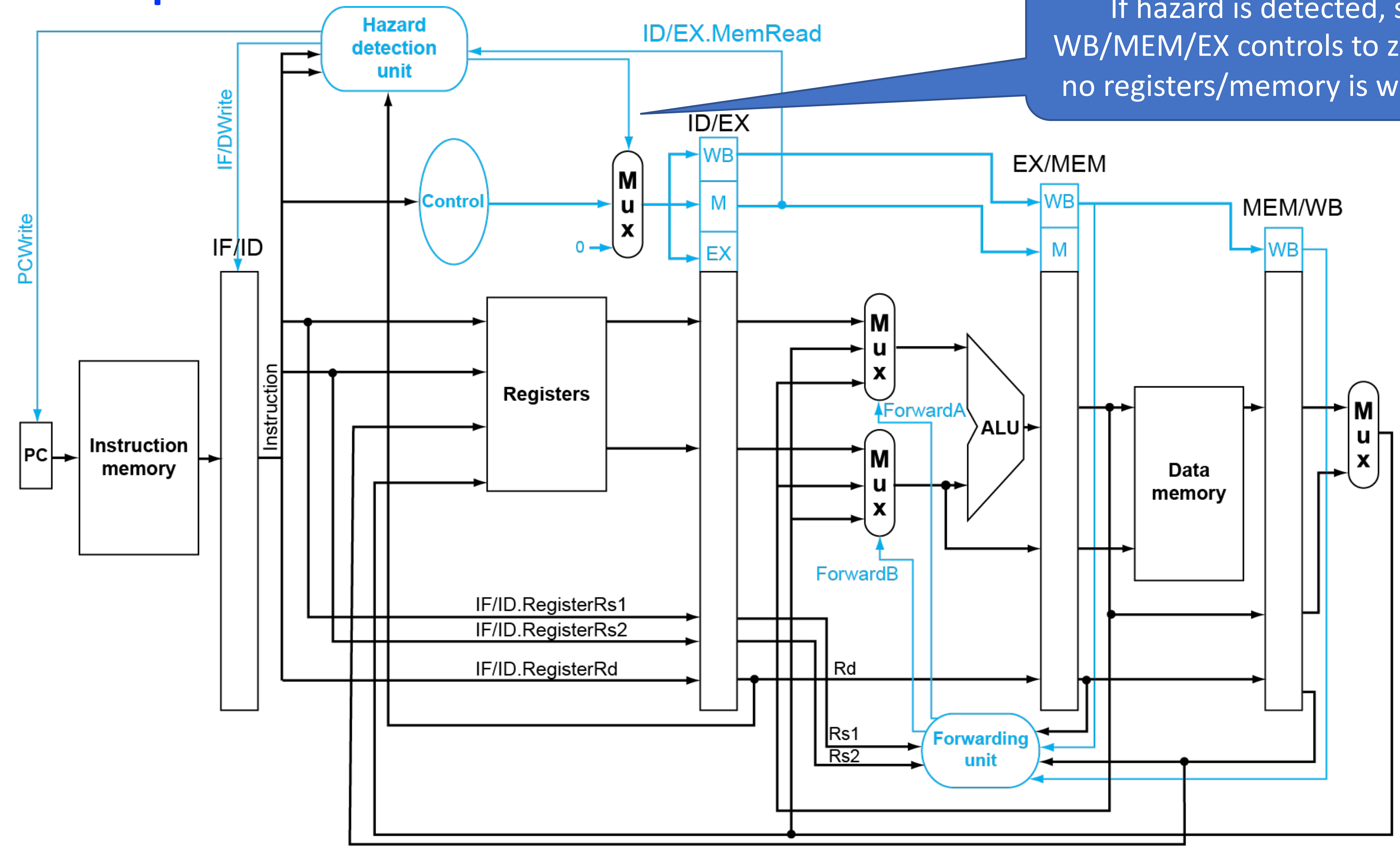
- Force control values in ID/EX register to 0
  - EX, MEM and WB do `nop` (no-operation)
- Prevent update of PC and IF/ID register
  - Current instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `ld`
    - Can subsequently forward to EX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

Load-Use Data Hazard



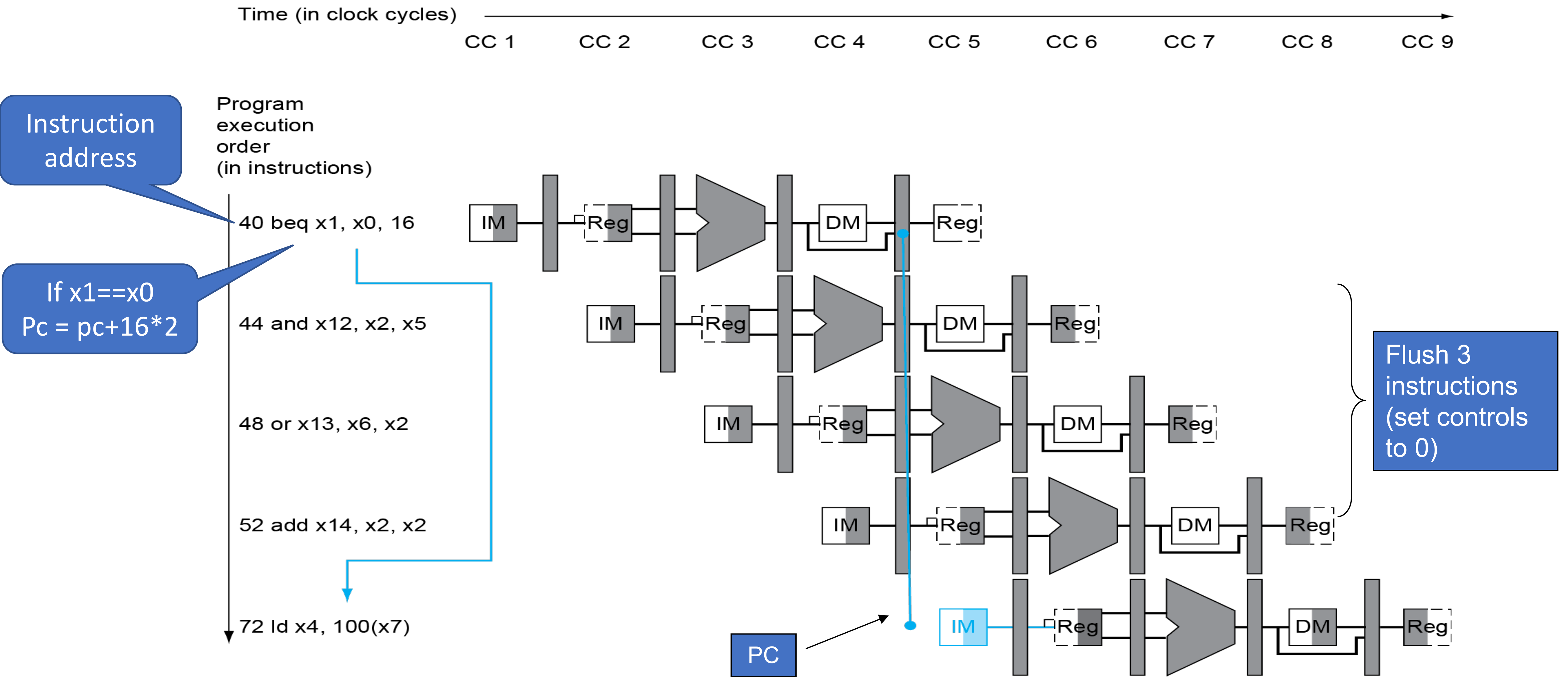
Datapath with Hazard Detection

If hazard is detected, set WB/MEM/EX controls to zero (so no registers/memory is written)



Branch Hazard

- If branch outcome is determined in MEM

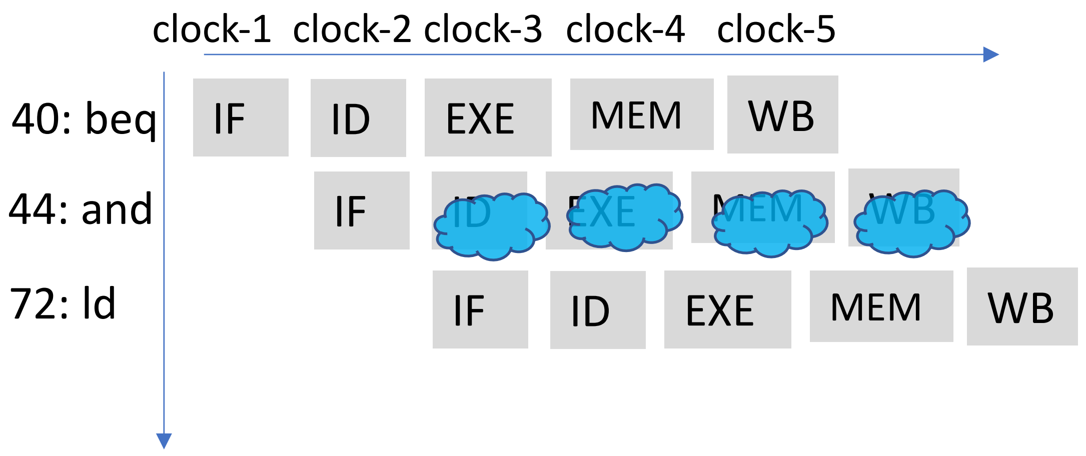


Reducing Branch Delay

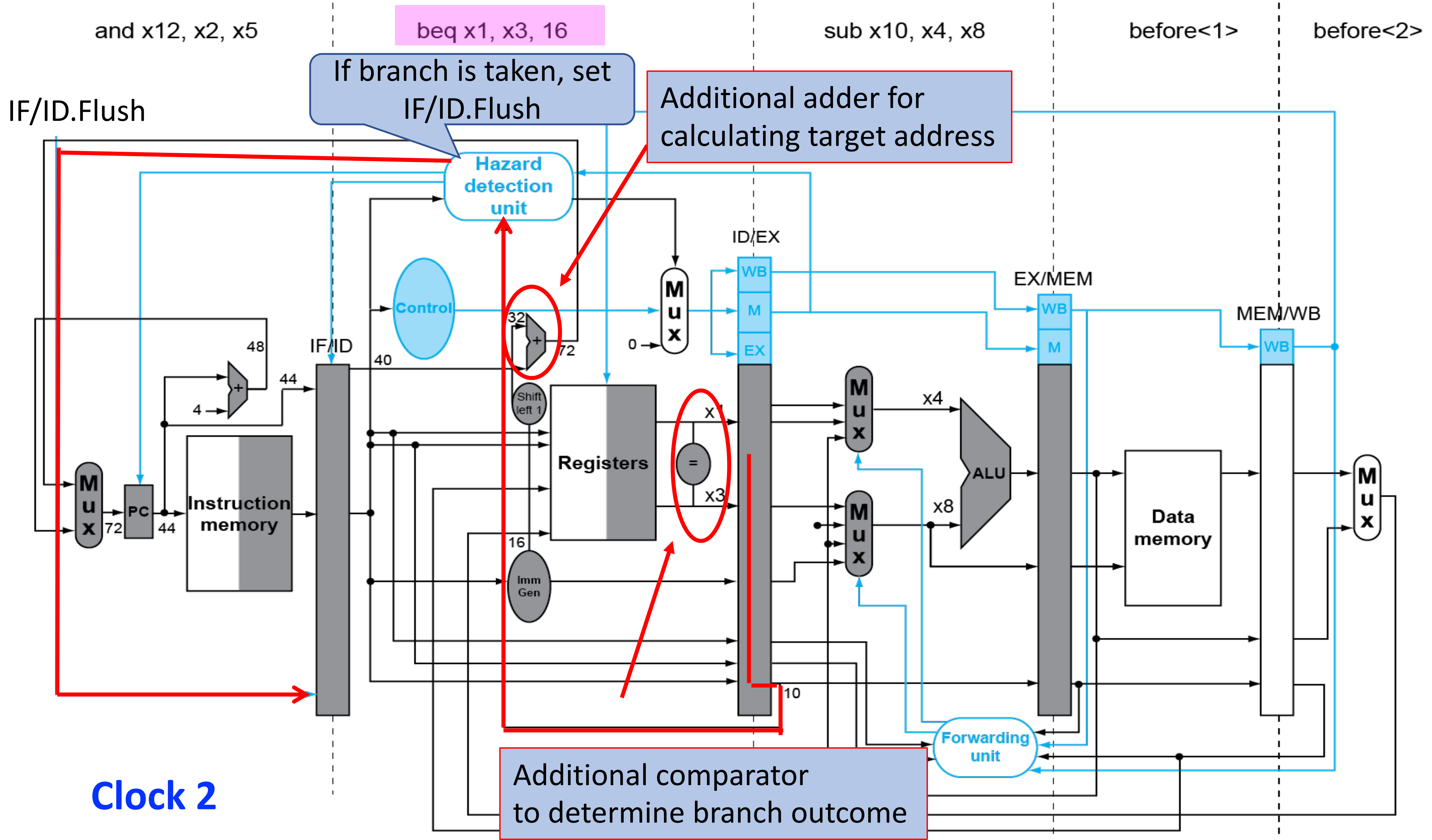
- Add hardware to determine branch outcome earlier (e.g. ID instead of MEM) → fewer instructions to flush

```
36:   sub  x10, x4, x8
40:   beq  x1,  x3, 16  // PC-relative branch
                        // to 40+16*2=72
44:   and  x12, x2, x5
48:   orr  x13, x2, x6
52:   add  x14, x4, x2
56:   sub  x15, x6, x7
      ...
72:   ld   x4, 50(x7)
```

How many instructions to flush if branch outcome is known in ID?

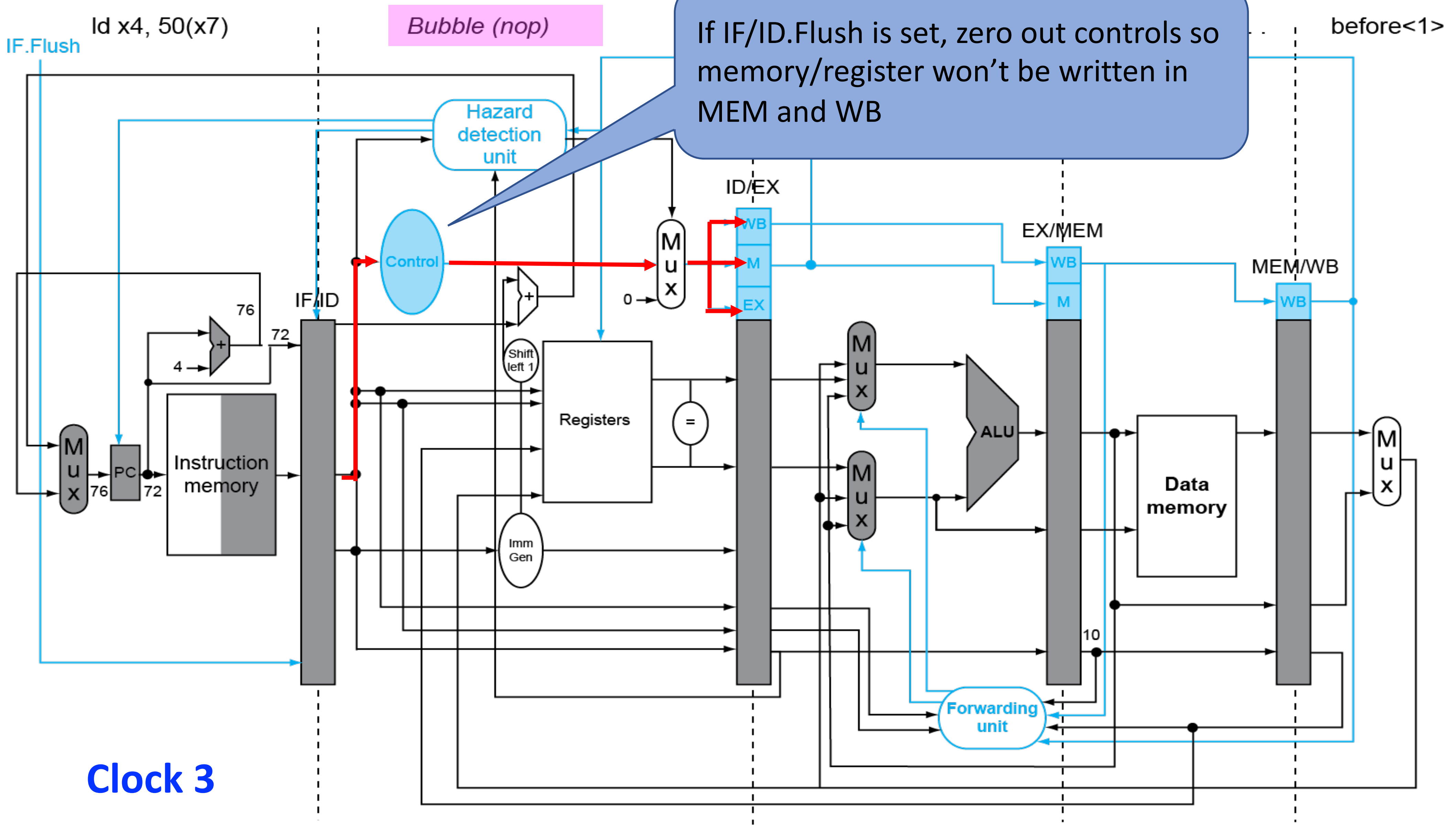


Branch determined in ID and is taken



Clock 2

Branch determined in ID and is taken



If IF/ID.Flush is set, zero out controls so memory/register won't be written in MEM and WB

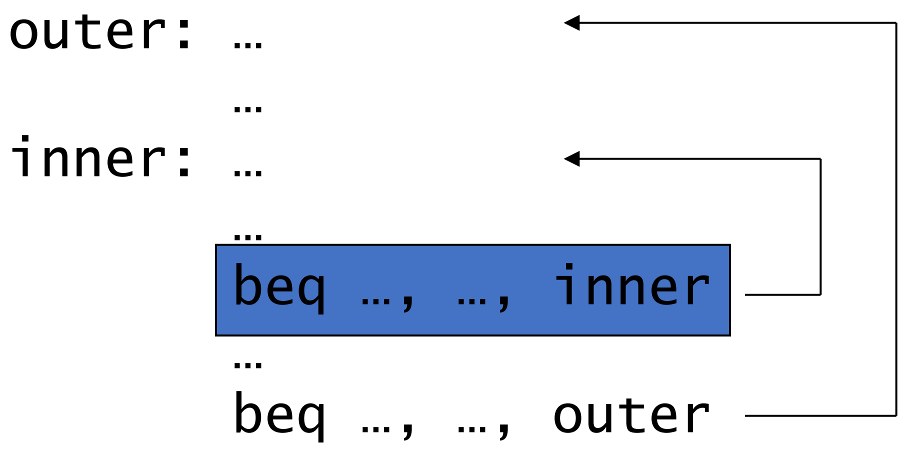
Clock 3

Dynamic Branch Prediction

- Our simple 5-stage pipeline’s branch penalty is 1 bubble, but
  - In deeper pipelines, branch penalty is more significant
- Solution: dynamic 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, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!

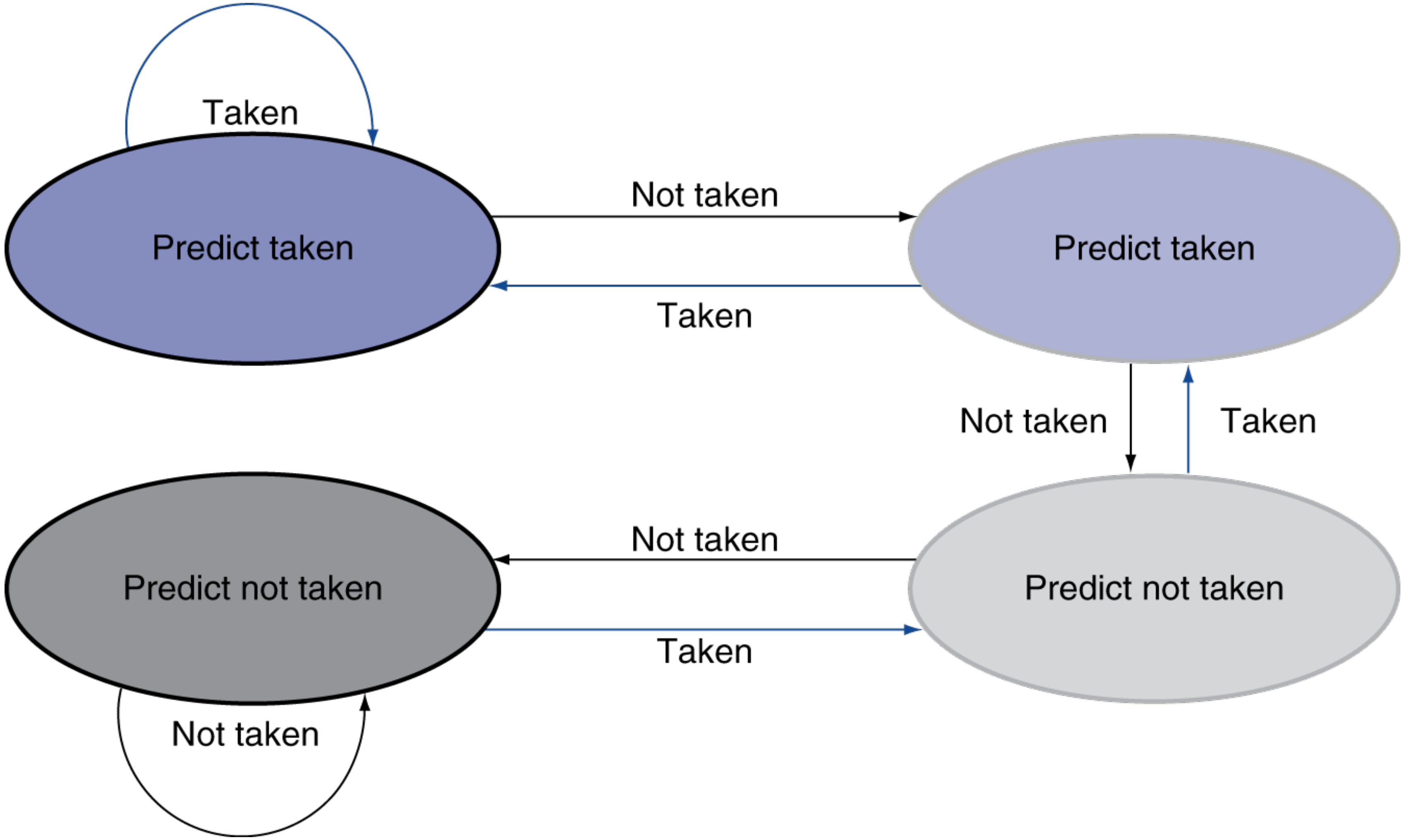


```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

- Mispredict as taken on last iteration of inner loop
- Then mispredict as not taken on first iteration of inner loop next time around

2-Bit Predictor

- Only change prediction on two successive mispredictions



Calculating Branch Target (needed if branch is predicted taken)

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- 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

Summary

- Pipeline increases throughput by overlapping execution of multiple instructions
- Pipeline hazard
  - Structure (solution: add resources)
  - Data (solution: forwarding)
  - Control (next class)
- Pipeline stalls