CSCI 210: Computer Architecture
Lecture 30: Data Hazards

Stephen Checkoway

Slides from Cynthia Taylor

Previously: Hazards

Situations that (if not handled) 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
  - Strategies for data hazards: forwarding, stalling, and code rewriting
- Control hazard
  - Deciding on control action depends on previous instruction
  - Strategy for control hazards: Move branch logic as early as possible

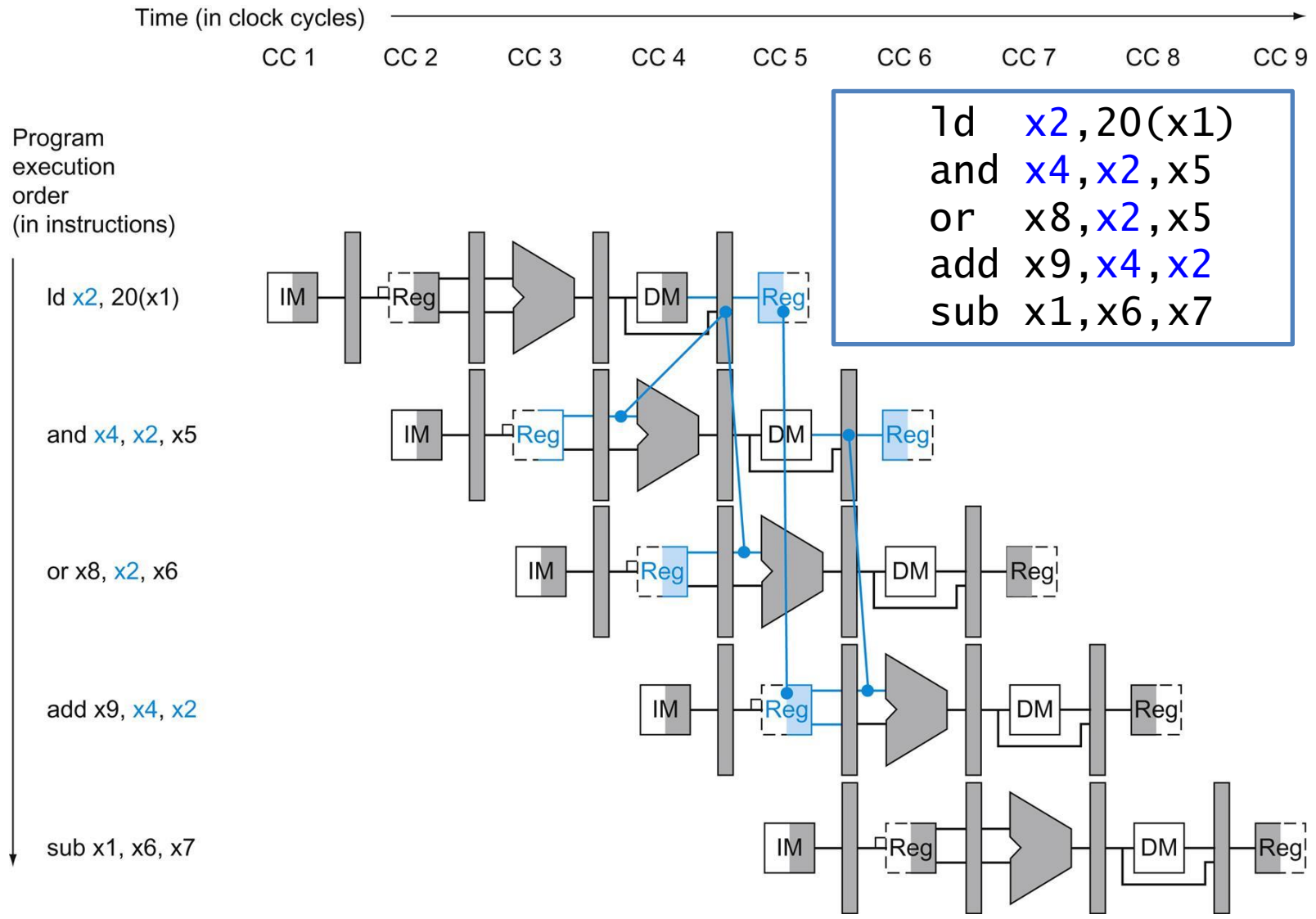
We can best solve **these** data hazards

A. By stalling

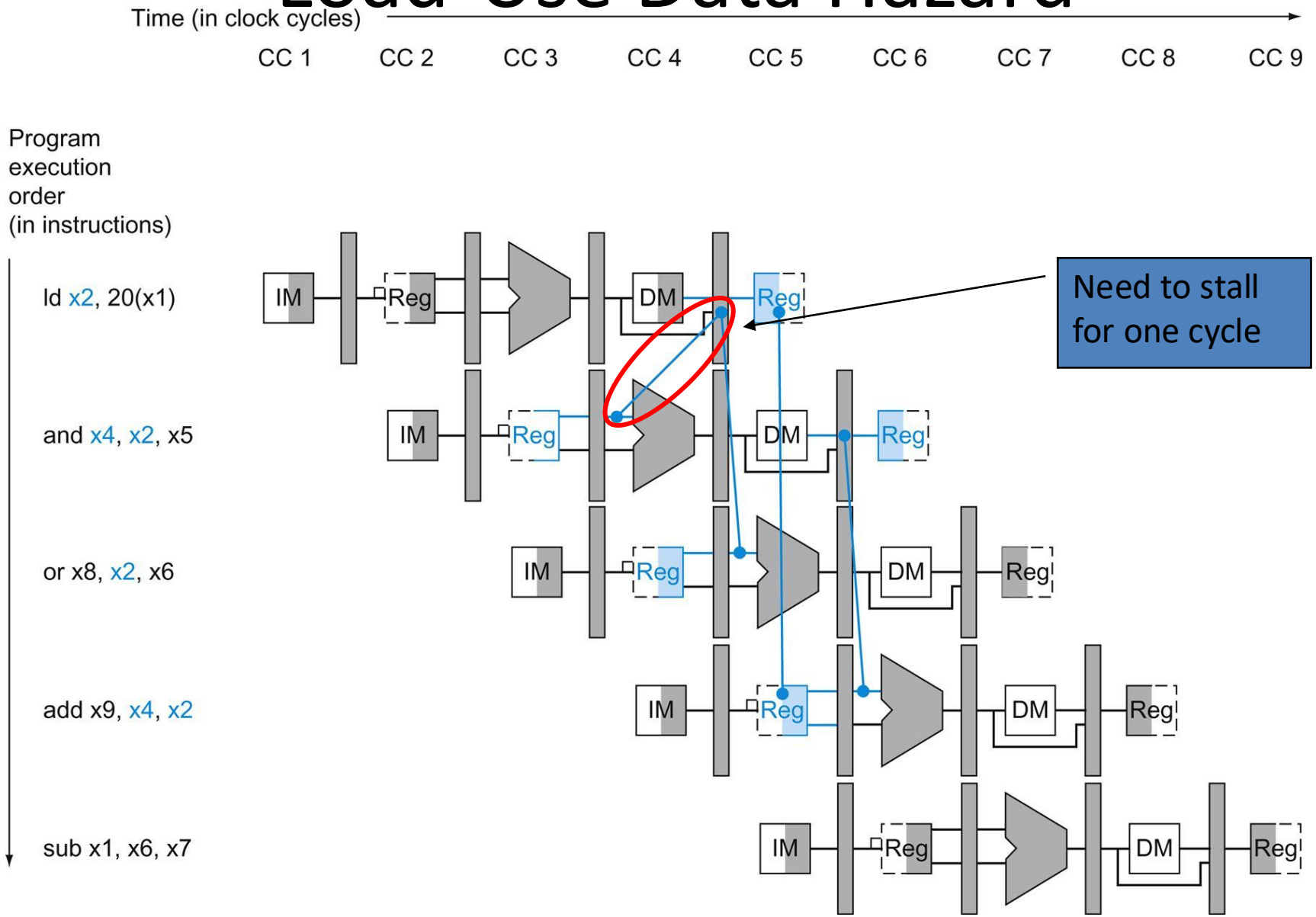
B. By forwarding

C. By combining forwards and stalls

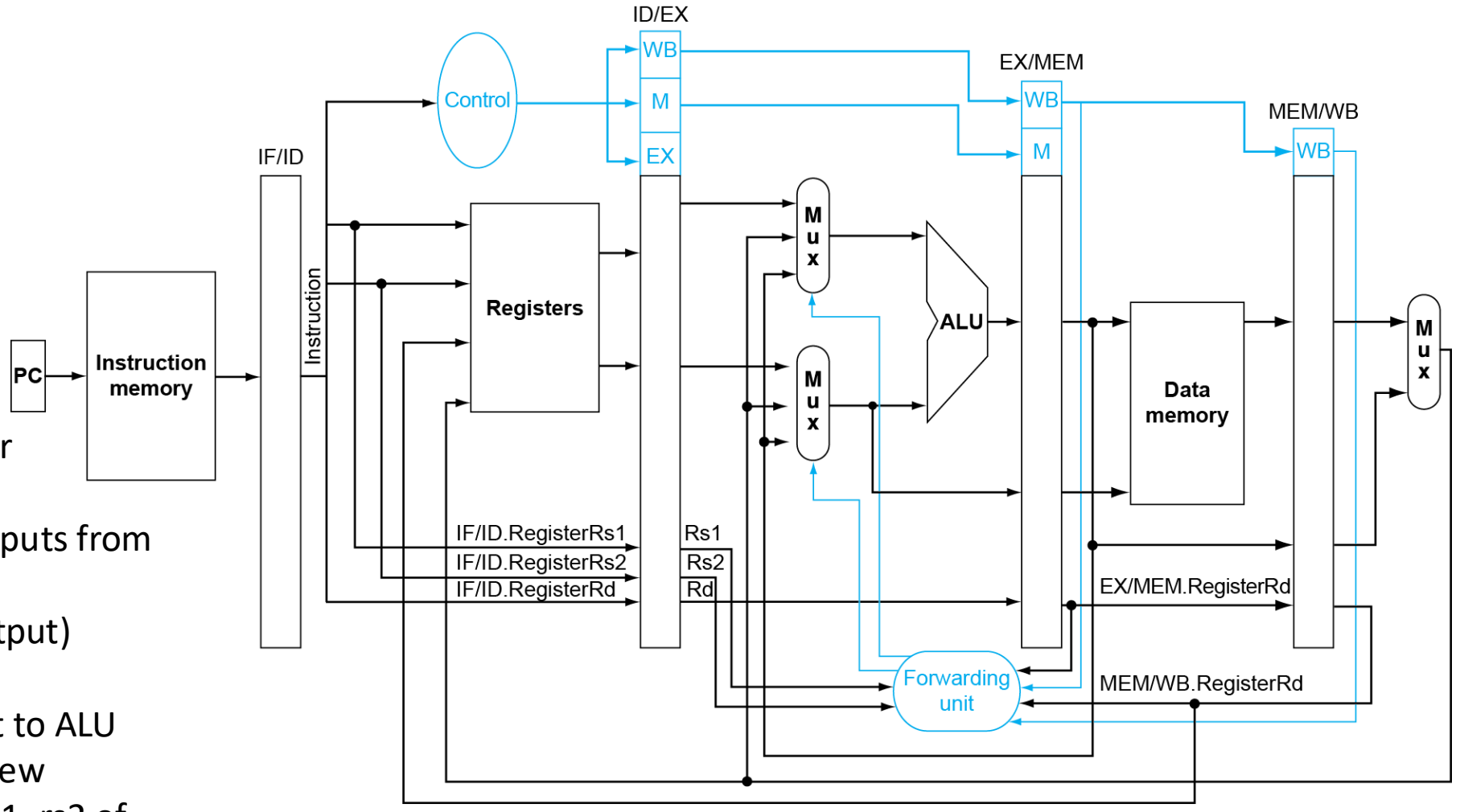
D. By doing something else



Load-Use Data Hazard



Data path with Forwarding



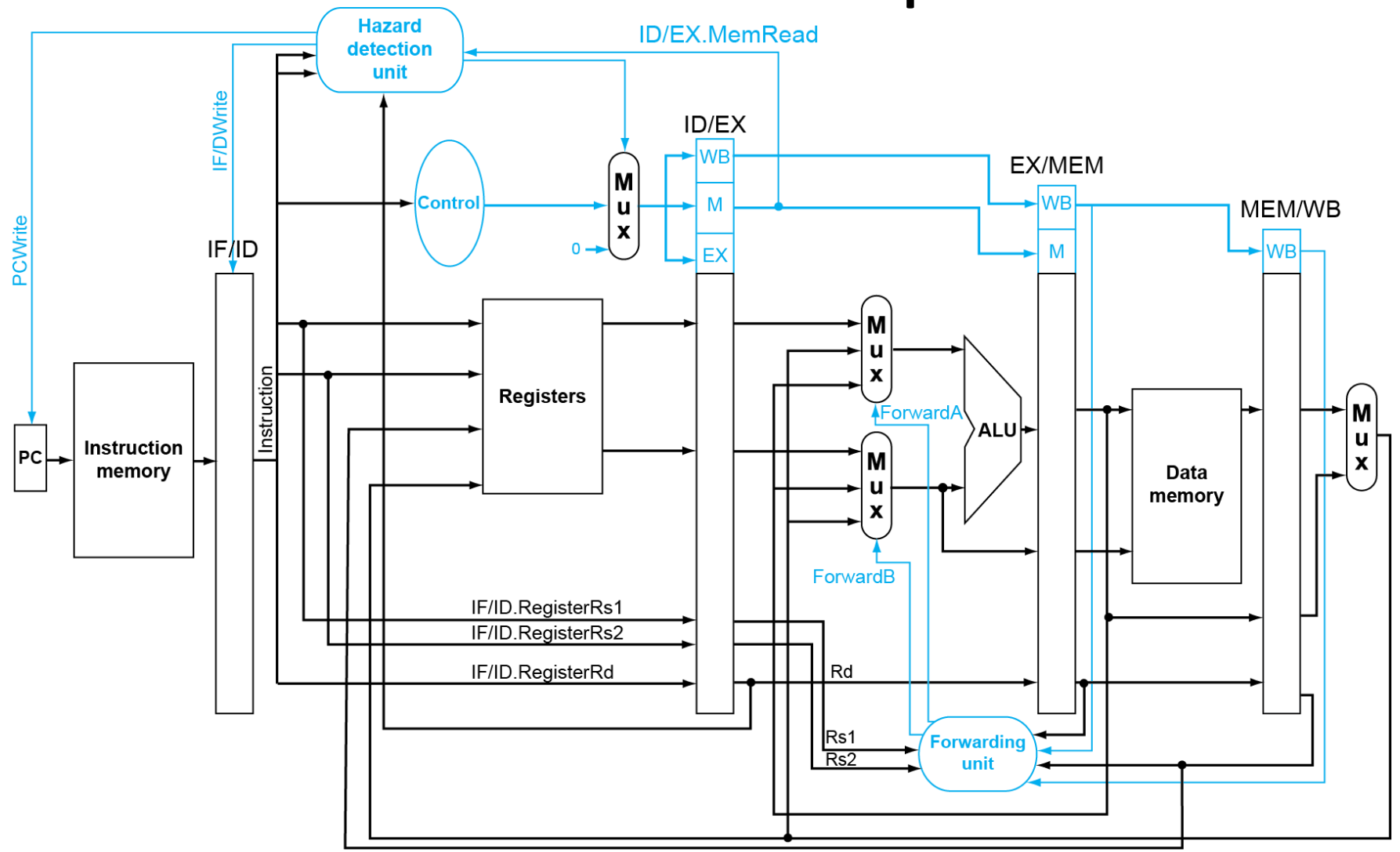
Key points:

- Two more possible inputs for each ALU input
- Use multiplexers to select inputs from
  - ID/EX (RegFile)
  - EX/MEM (prev. ALU output)
  - MEM/WB
- Not shown: Imm. Gen. input to ALU
- Multiplexers controlled by new Forwarding Unit based on rs1, rs2 of the instruction in EX and rd of the previous two instructions

Forwarding logic

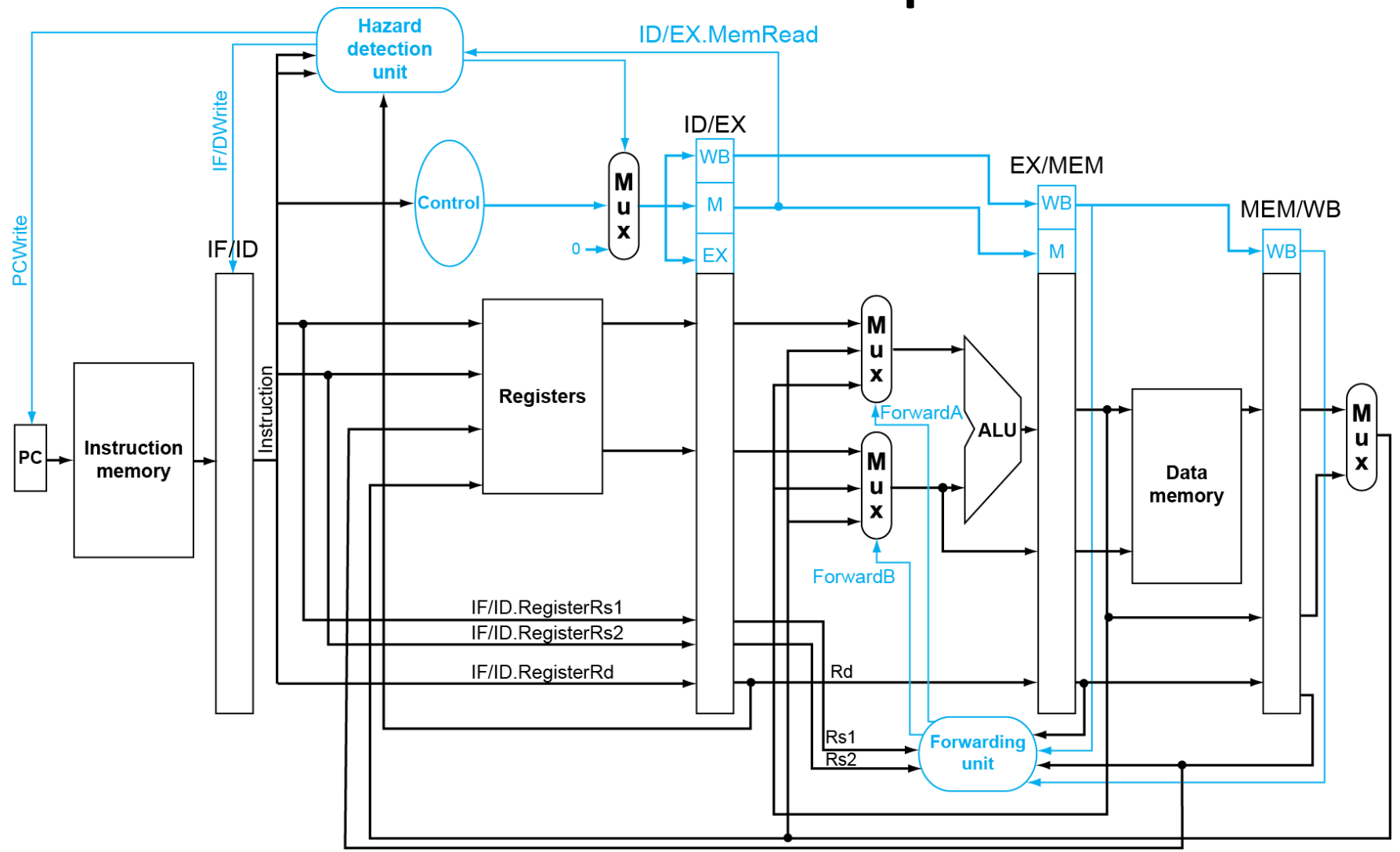
- Let rs1 = ID/EX.RegisterRs1
  rs2 = ID/EX.RegisterRs2
  prev_rd = EX/MEM.RegisterRd
  prev_prev_rd = MEM/WB.RegisterRd
- If rs1 = prev_rd and rs1 ≠ x0, use value in EX/MEM as ALU input 1
- Otherwise, if rs1 = prev_prev_rd and rs1 ≠ 0, use value in MEM/WB as ALU input 1
- Similar logic for rs2 and ALU input 2

How to Stall the Pipeline



- Detect hazard in ID stage using new Hazard detection unit
  - Check if instruction in EX stage is a load with destination rs1 or rs2

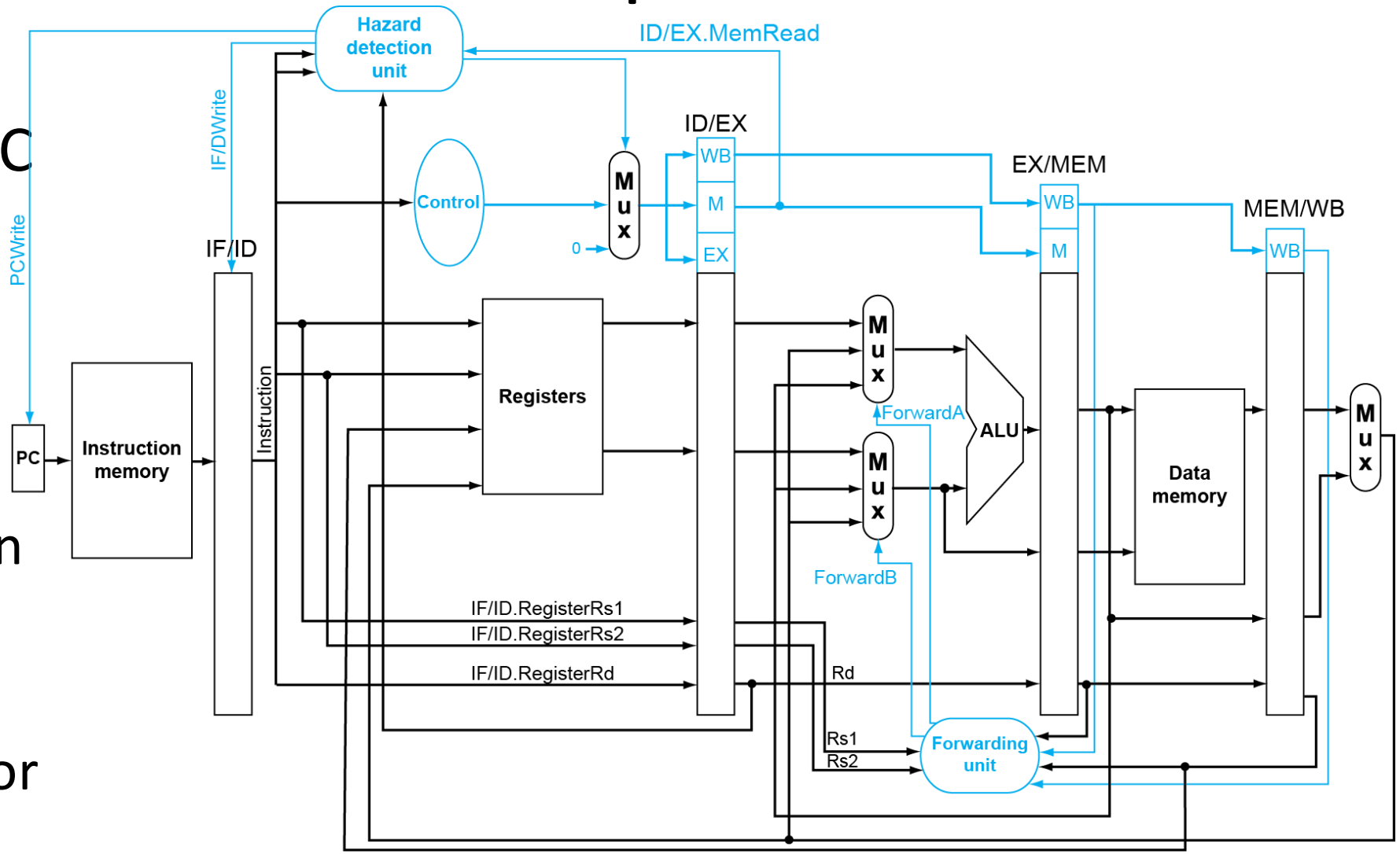
How to Stall the Pipeline



- Force control values in ID/EX register to 0
  - EX, MEM and WB do nop (no-operation)

How to Stall the Pipeline

- Prevent update of PC and IF/ID register
  - Instruction with dependency is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for ld

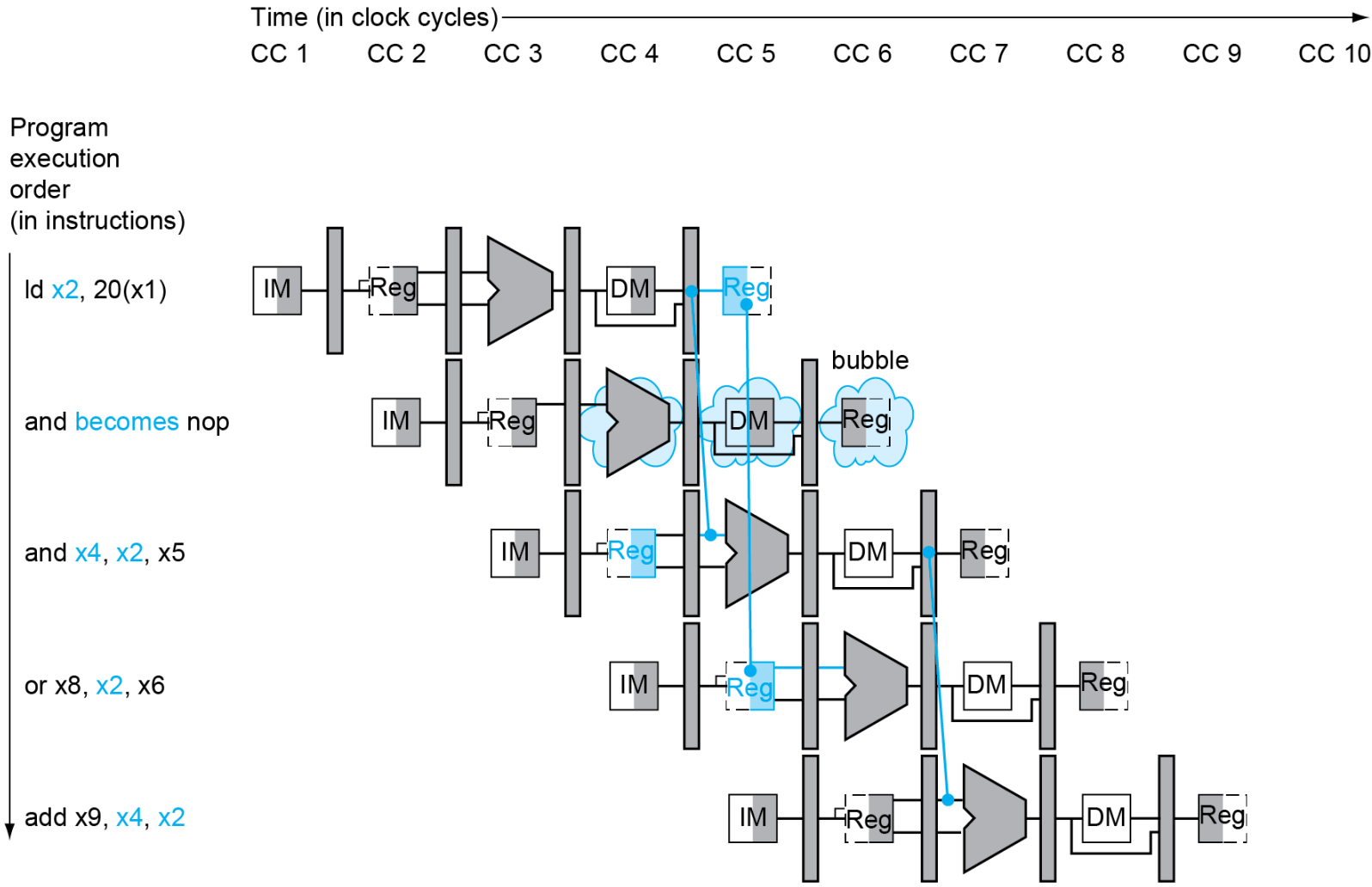


After we add the stall

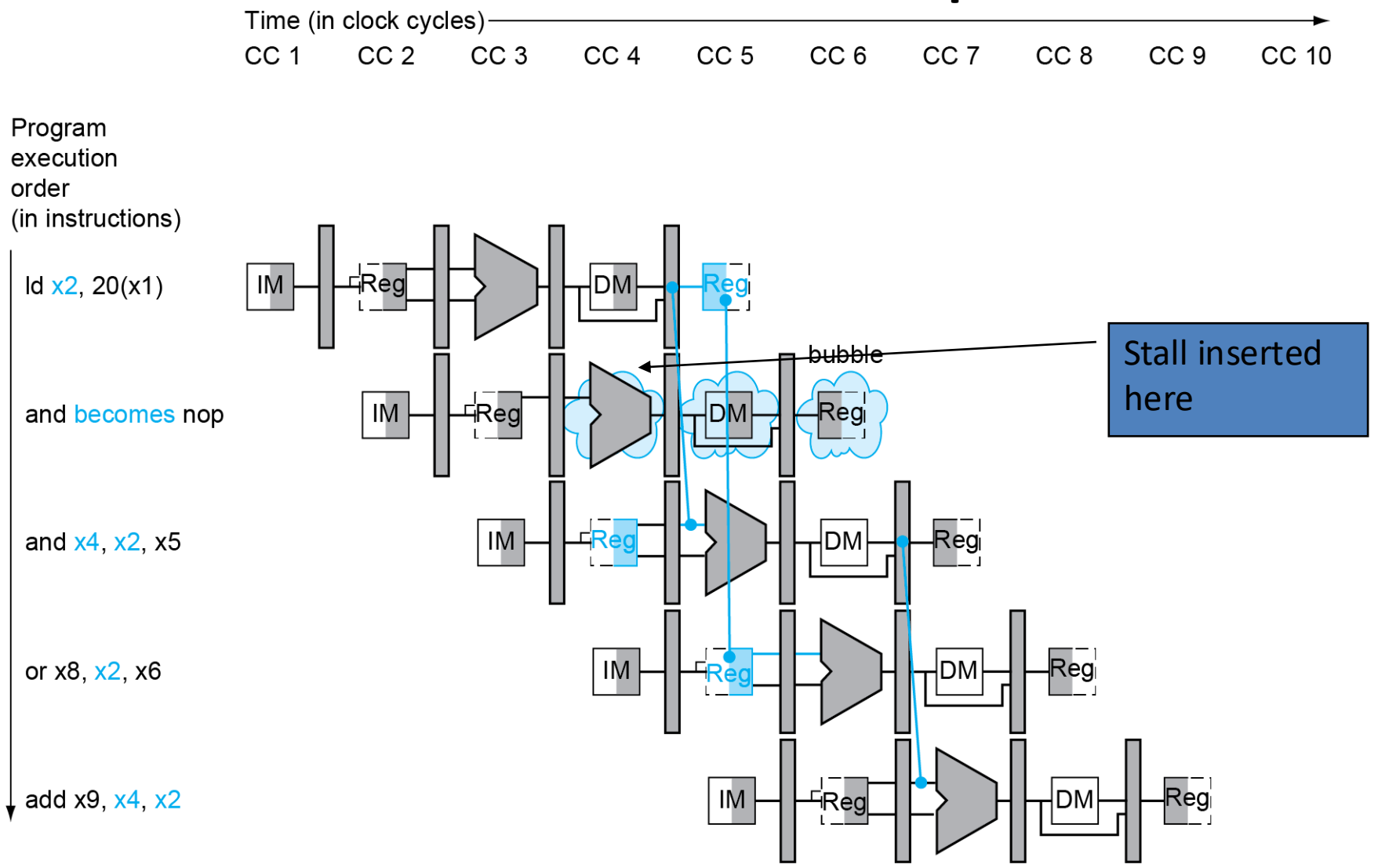
A. Everything works with our existing forwarding

B. We need to forward between the register files to solve the 2nd hazard

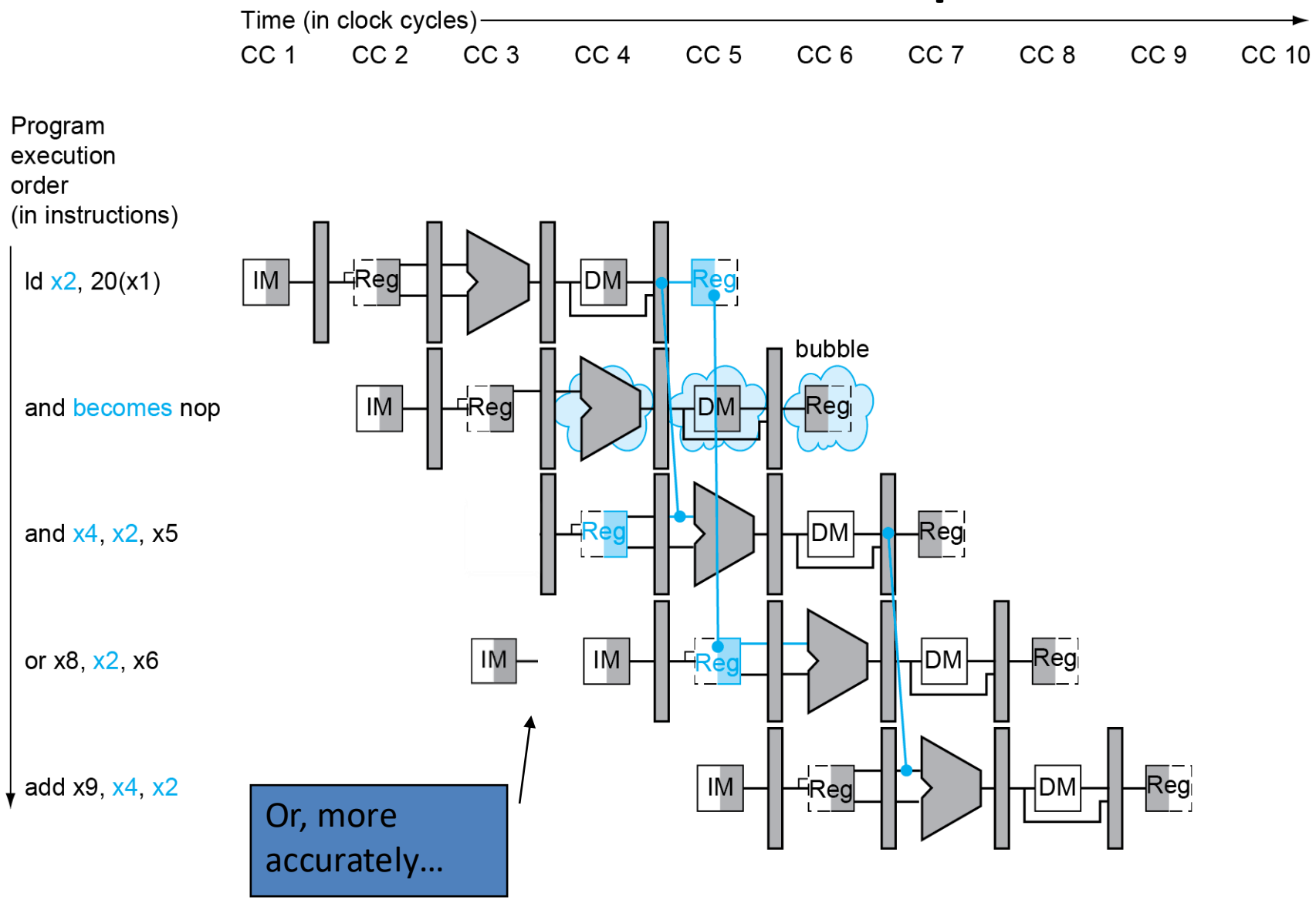
C. We need to do something else



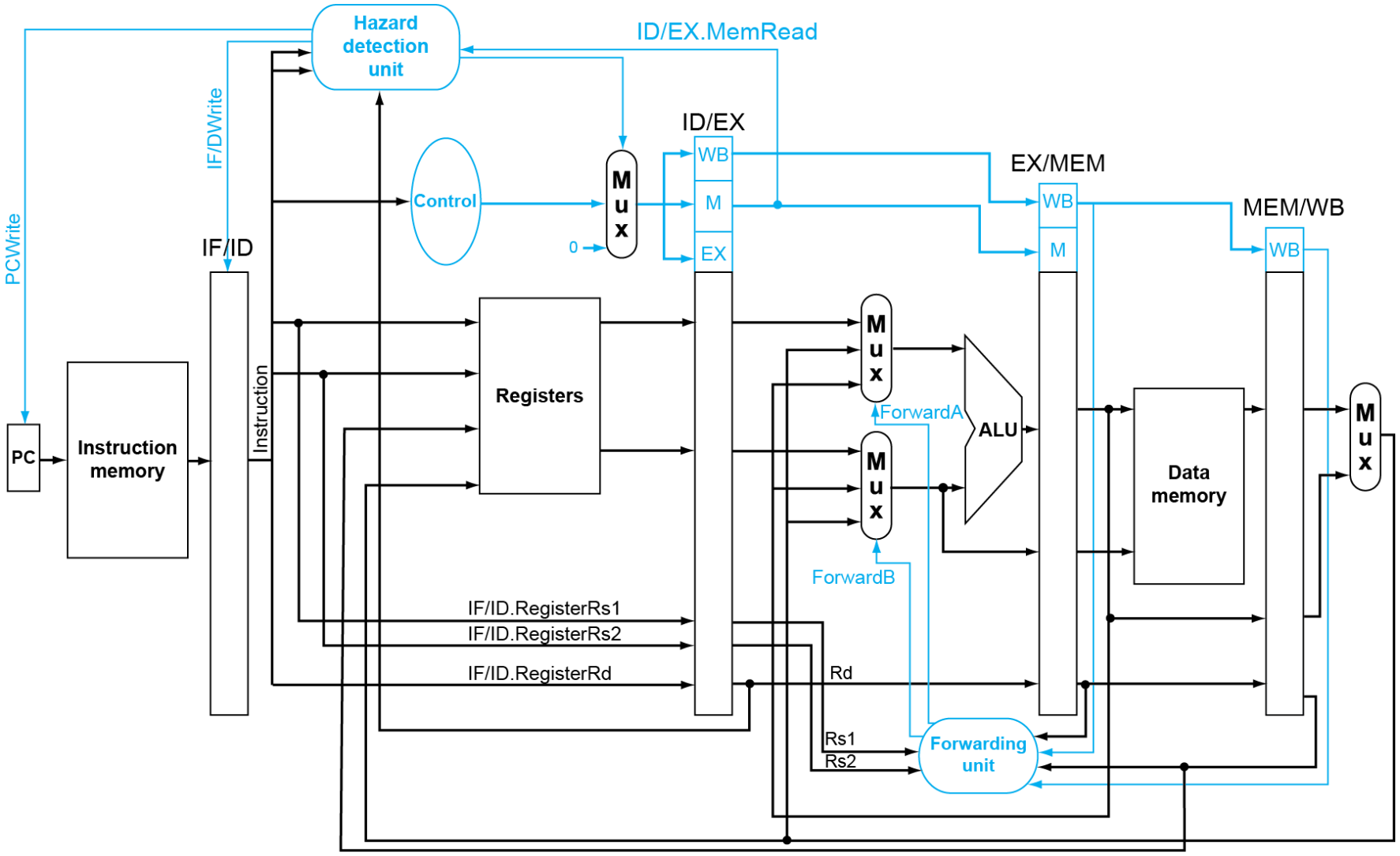
Stall/Bubble in the Pipeline



Stall/Bubble in the Pipeline



Questions about Data Hazards?



Consider the code

```
addi    s0, s0, 8
ld      t0, 0(s0)
sub     t1, t2, t2
add     t0, t0, t1
```

Does this code require a forward, a stall, both, or neither?

A. Forward
B. Stall
C. Both
D. Neither

Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results

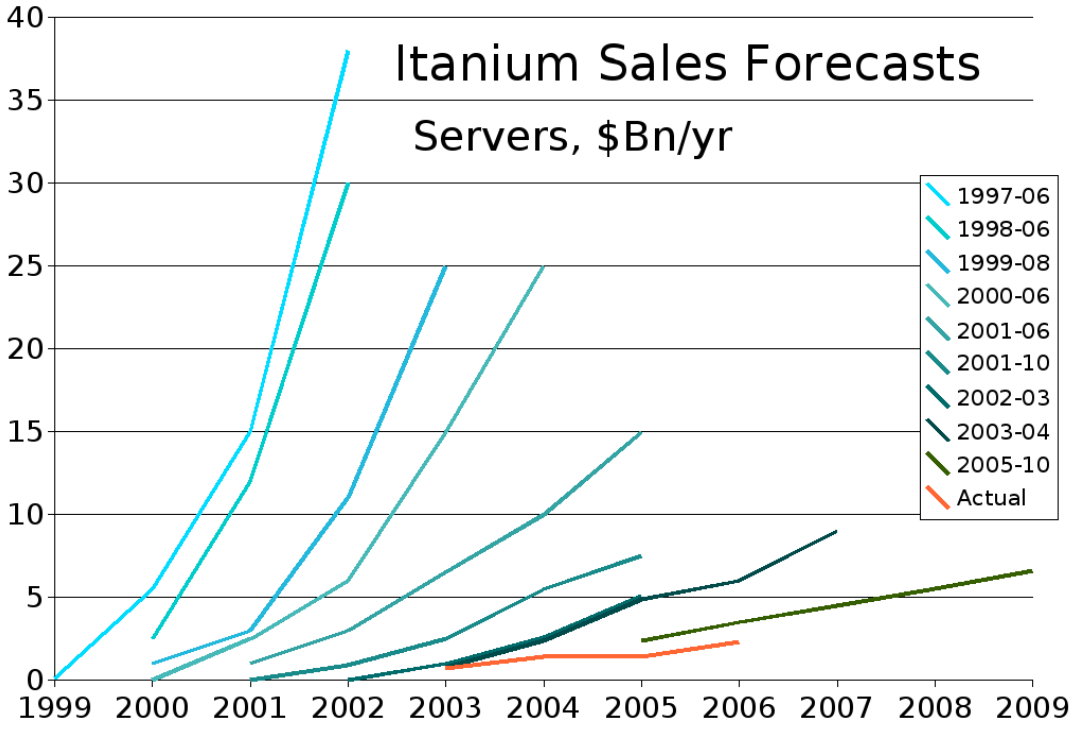
- Can rearrange code to avoid hazards and stalls

Dealing with Data Hazards

- As an ISA designer, you have a choice between reordering instructions in software or hardware. Which might you choose and why?

Selection	HW or SW	
A	Software	Compilers have a large window of instructions available to do reordering to eliminate hazards
B	Software	Detecting data hazards in hardware can be difficult and expensive
C	Hardware	Hardware knows at runtime the actual dependencies and can exploit that knowledge for better reordering
D	Hardware	Exposing the number of required stalls violates the abstraction between hardware and software

CS History: Intel Itanium Chip



Arch dude, CC BY-SA, via Wikimedia Commons

- Intel Chip launched in 2001 that used the VLIW (Very Long Instruction Word) ISA
- This ISA was designed to do all code reordering at compile time, rather than at runtime
- Designed for servers/high-performance, goal eventually desktop market
- Performance was disappointing, especially when emulating x86
- “Itanium's promise ended up sunken by a lack of legacy 32-bit support and difficulties in working with the architecture for writing and maintaining software” - Techspot

Reading

- Next lecture: Control Hazards
  - Section 4.8