Chapter 4

The Processor

Dr. Randa Mohamed

Agenda

- MIPS Pipeline:
  - Hazards
    - Structural Hazards
    - Data Hazards

Hazards

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

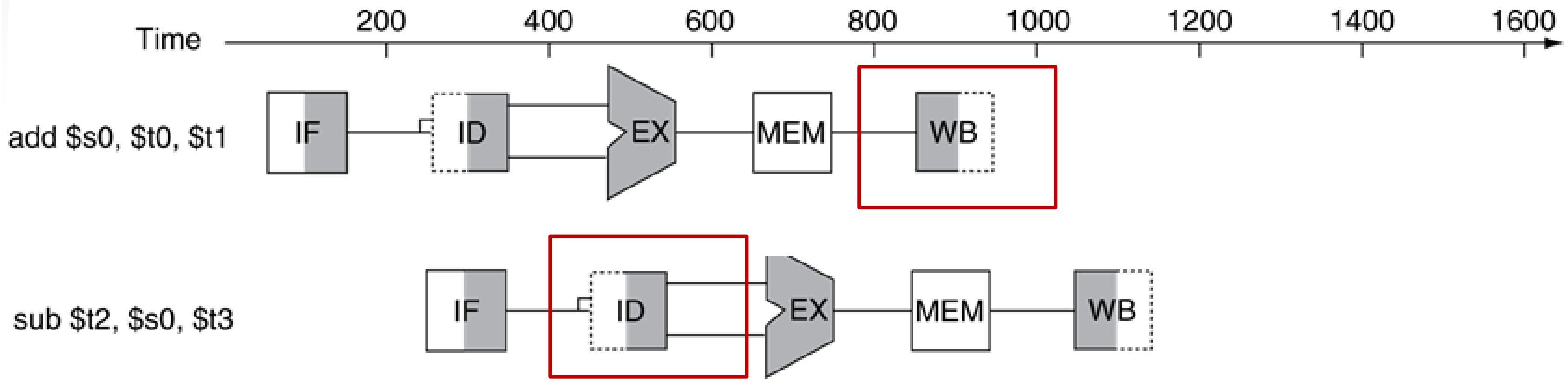
Structural Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
- Hence, pipelined datapaths require separate instruction/data memories

Data Hazards

- An instruction depends on completion of data access by a previous instruction

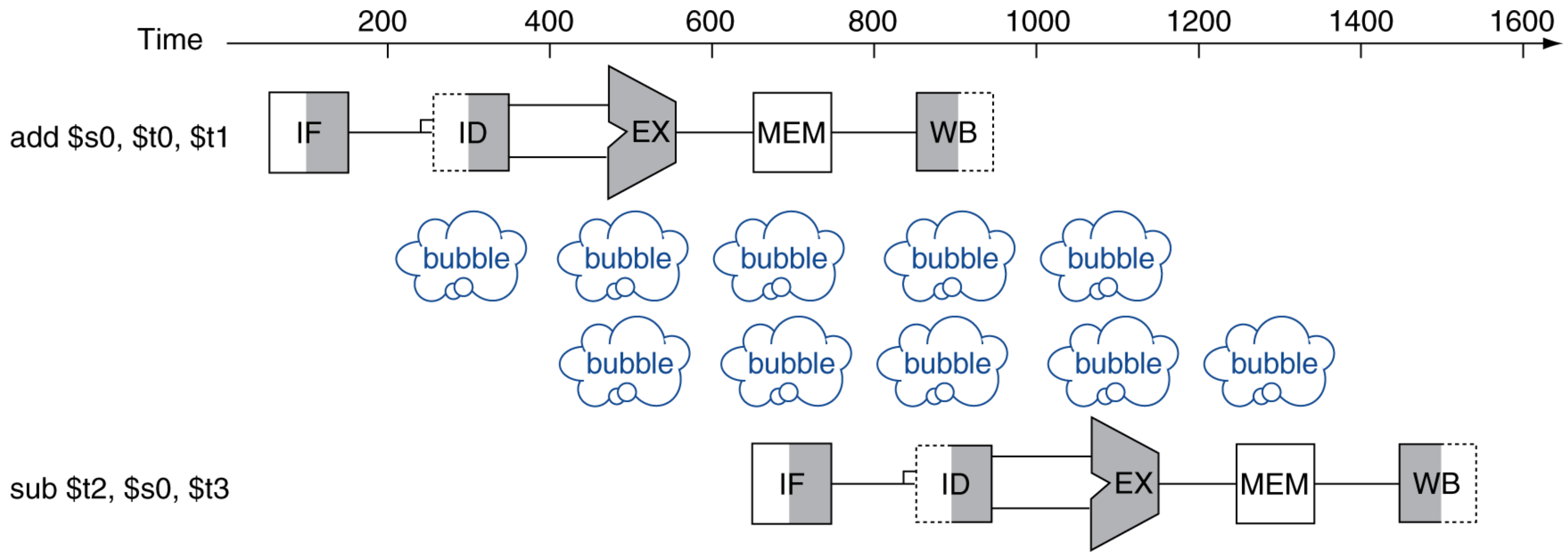
  -
    ```
    add  $s0, $t0, $t1
    sub  $t2, $s0, $t3
    ```



Data Hazards

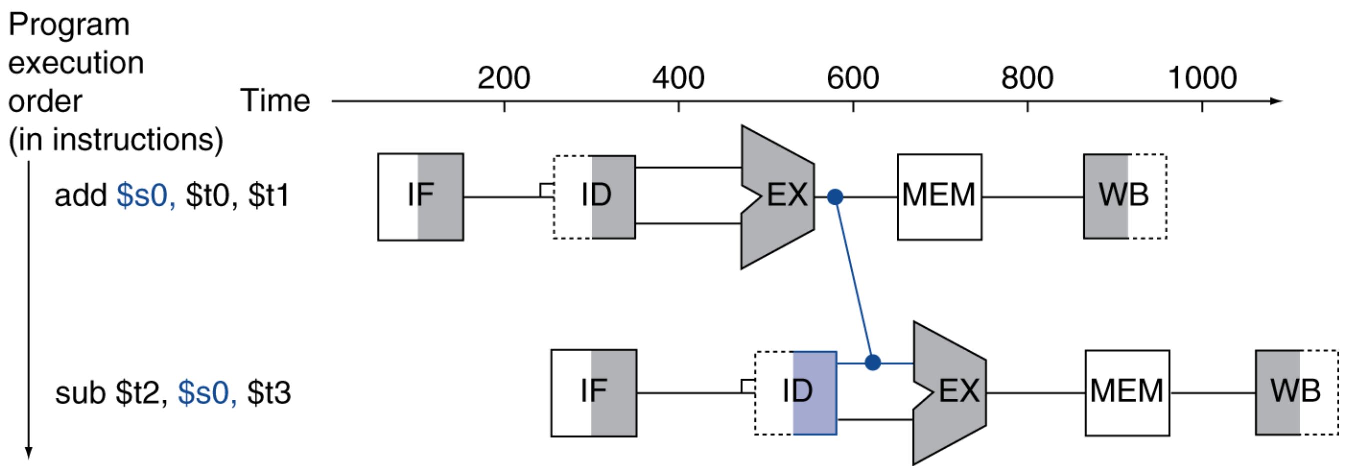
- An instruction depends on completion of data access by a previous instruction

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



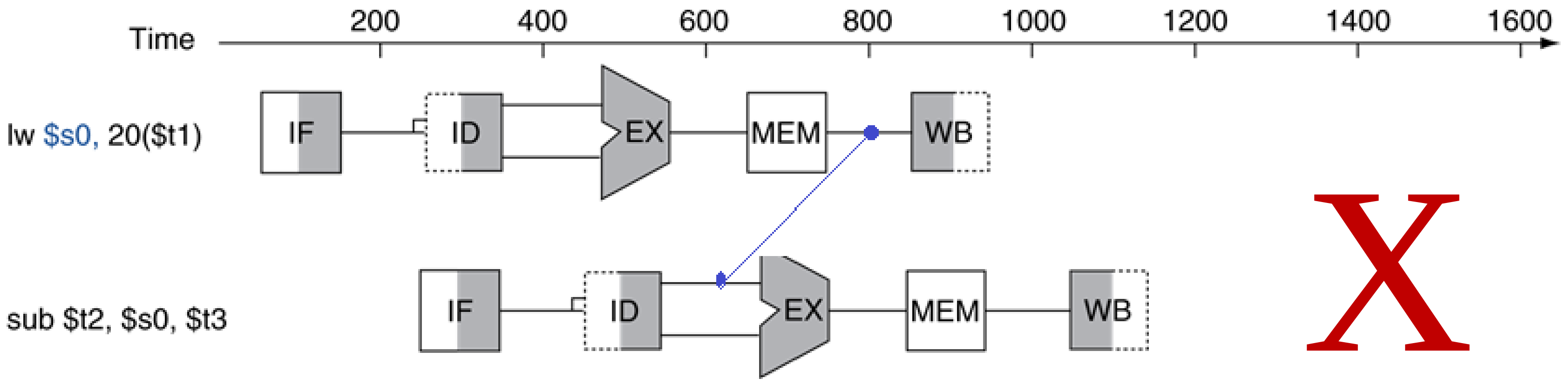
Forwarding (aka Bypassing)

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



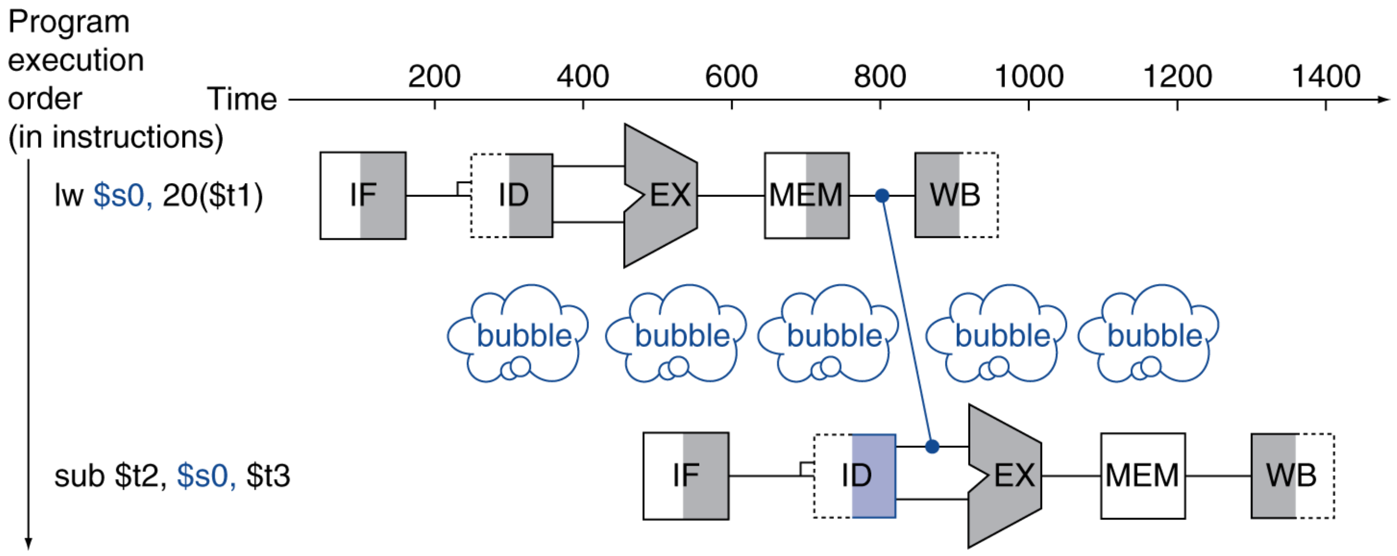
Load-Use Data Hazard

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



Load-Use Data Hazard

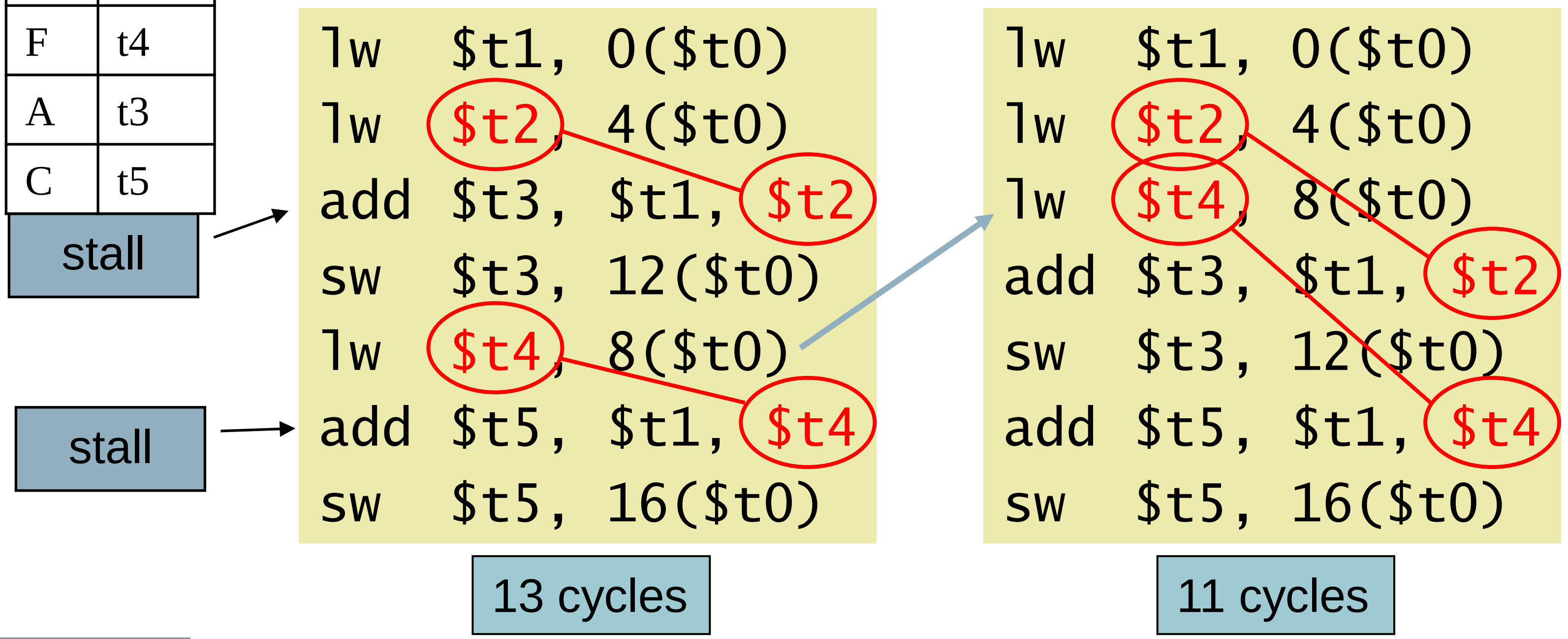
- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



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;`

t0	B	t1
t0+4	E	t2
t0+8	F	t4
t0+12	A	t3
t0+16	C	t5



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

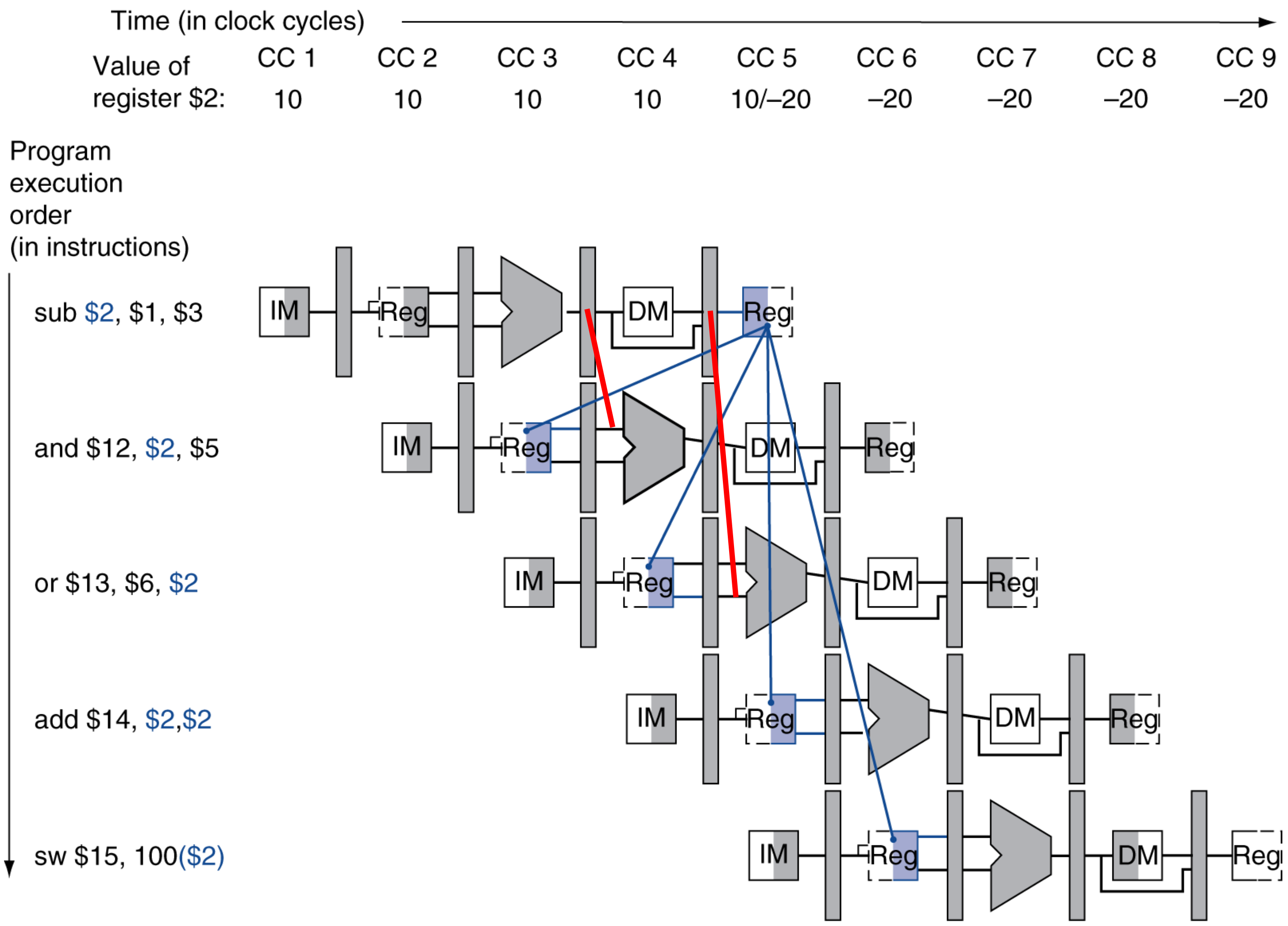
Data Hazards in ALU Instructions

- Consider this sequence:

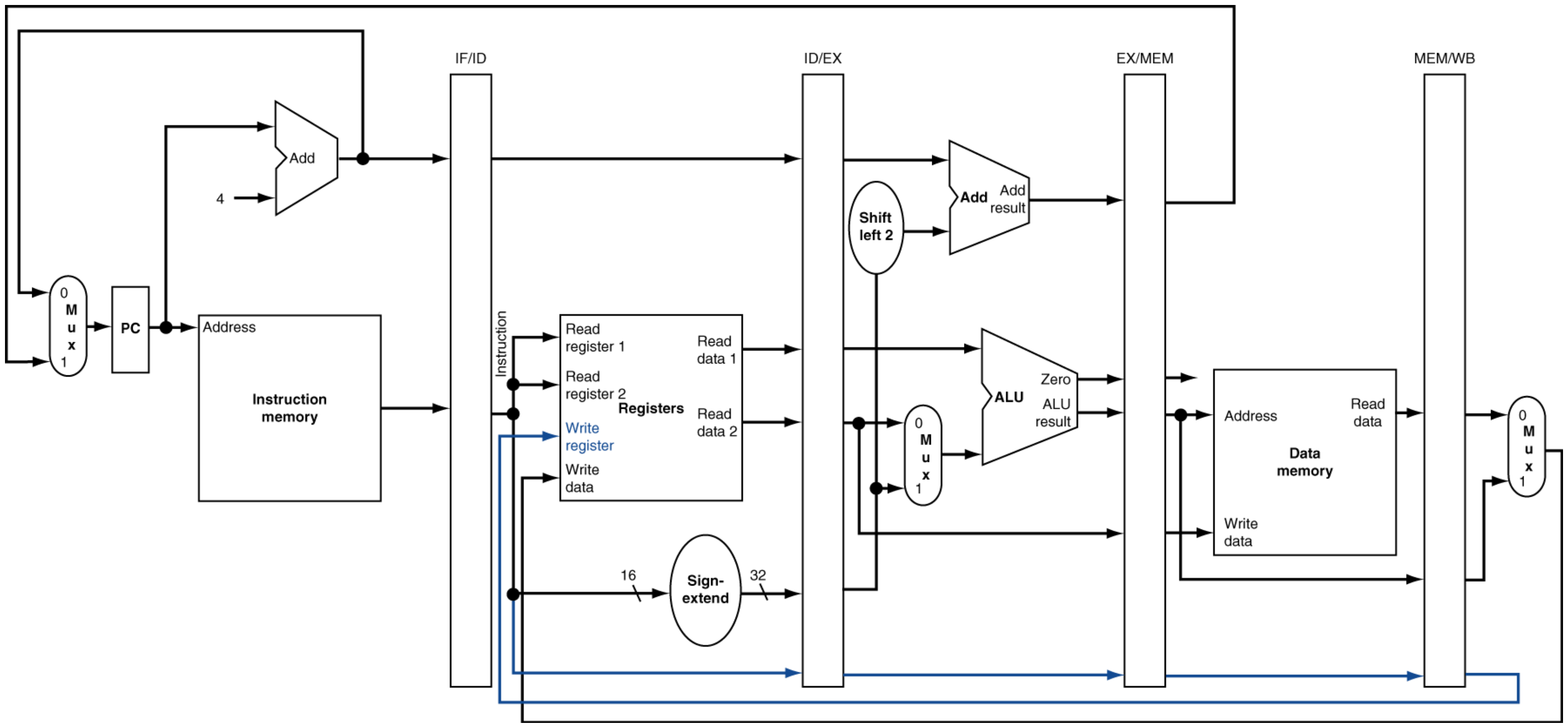
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- We can resolve hazards with forwarding
    - How do we detect when to forward?

Dependencies & Forwarding



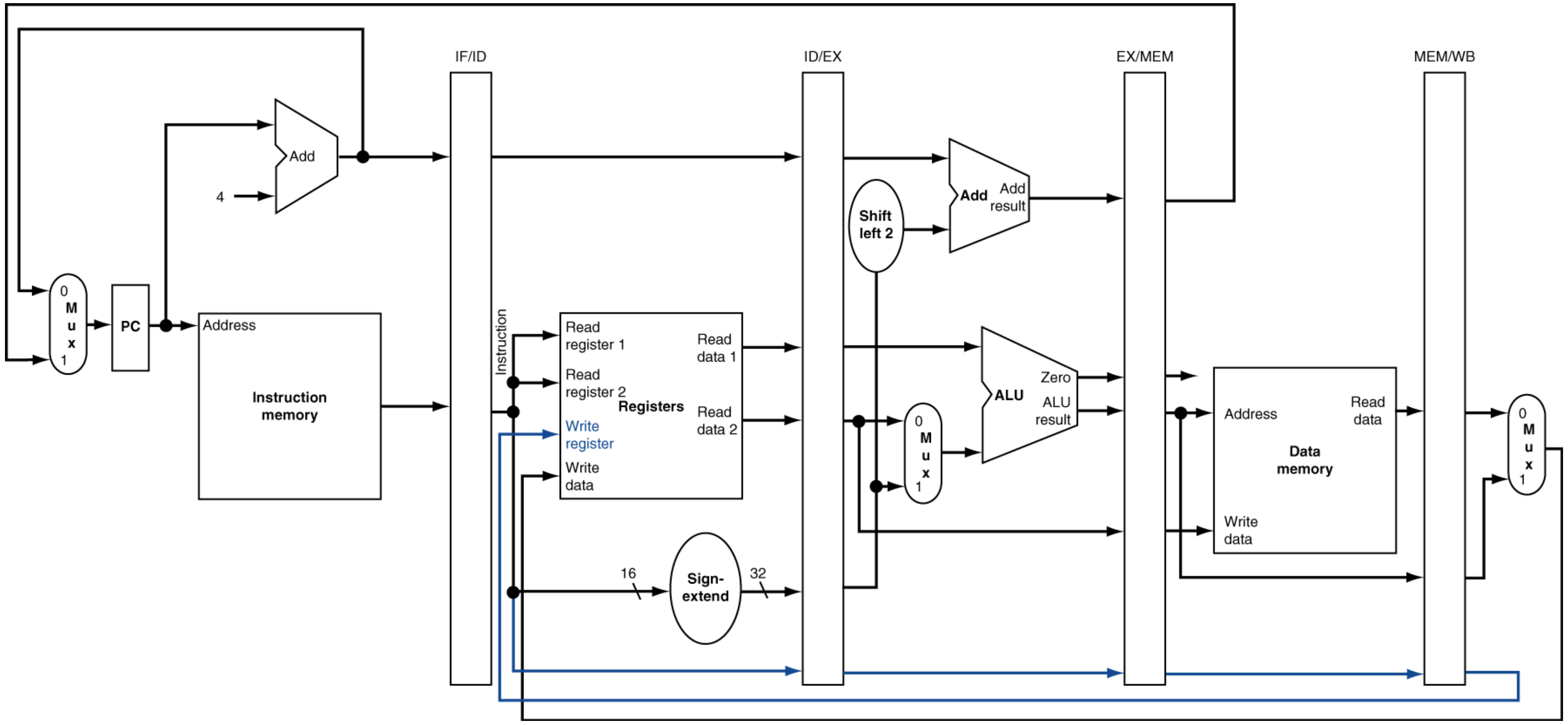
Remember: Pipelined Datapath



Remember: Pipelined Datapath

Cycle #3

OR AND SUB



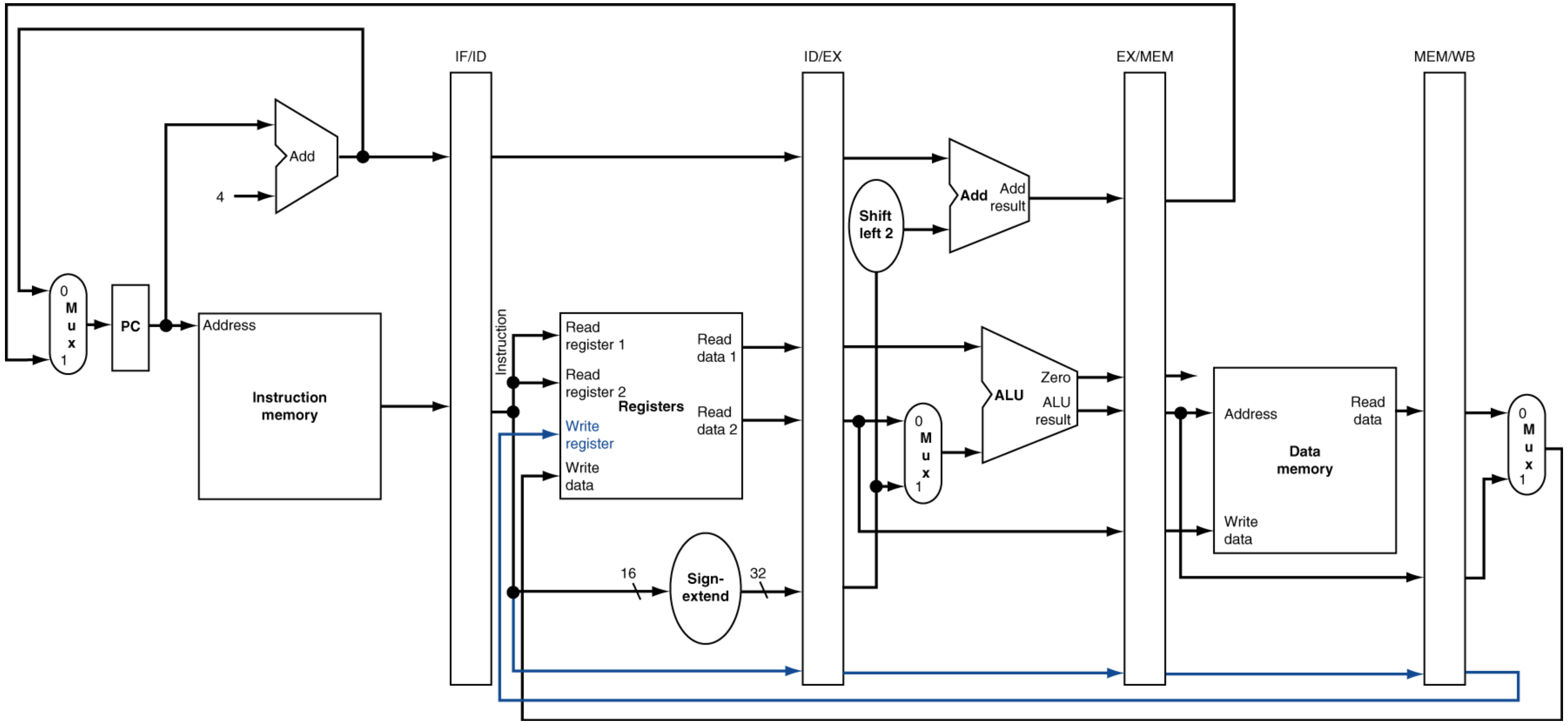
Remember: Pipelined Datapath

What to forward?

When to forward?

Cycle #4

OR AND SUB



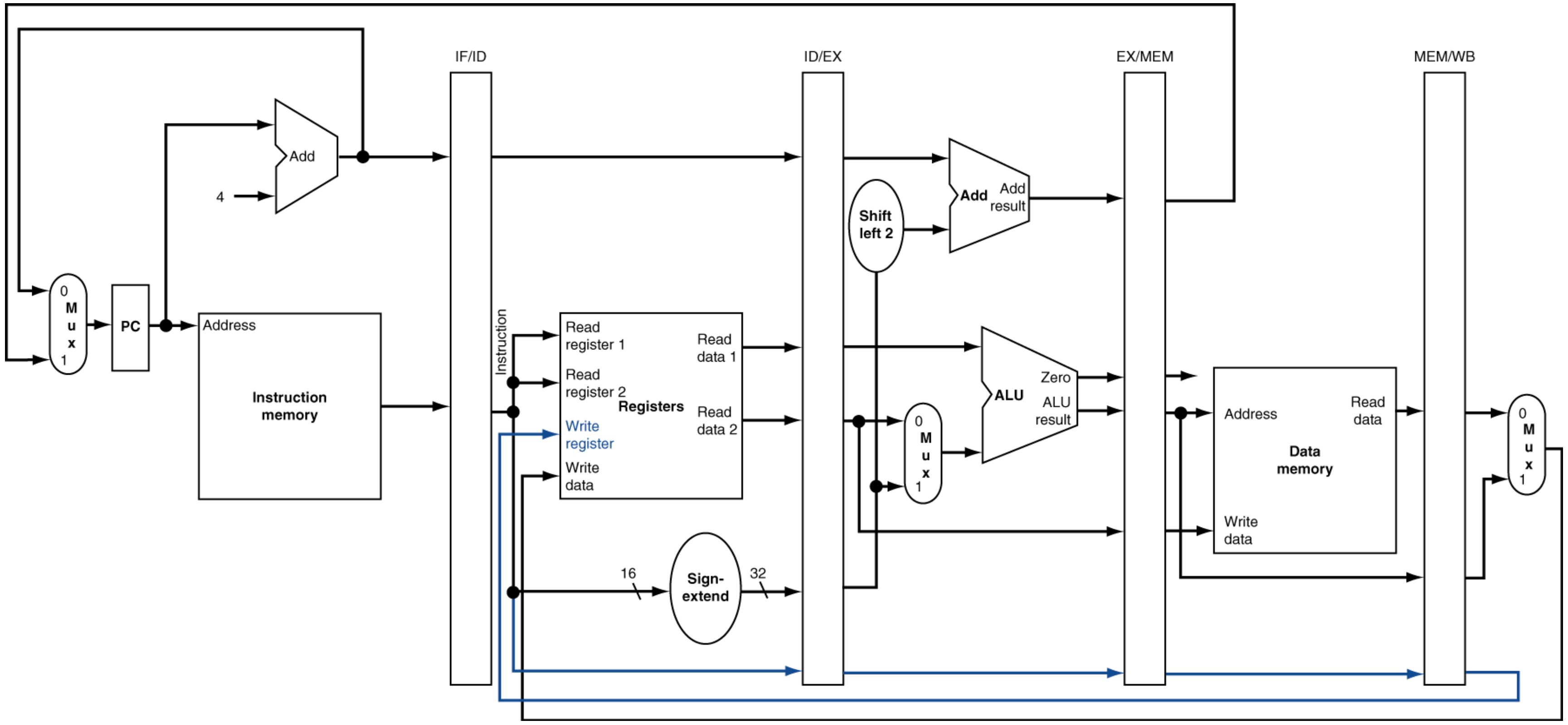
Remember: Pipelined Datapath

What to forward?

When to forward?

Cycle #5

OR AND SUB



Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., 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
- Data hazards when
  - 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

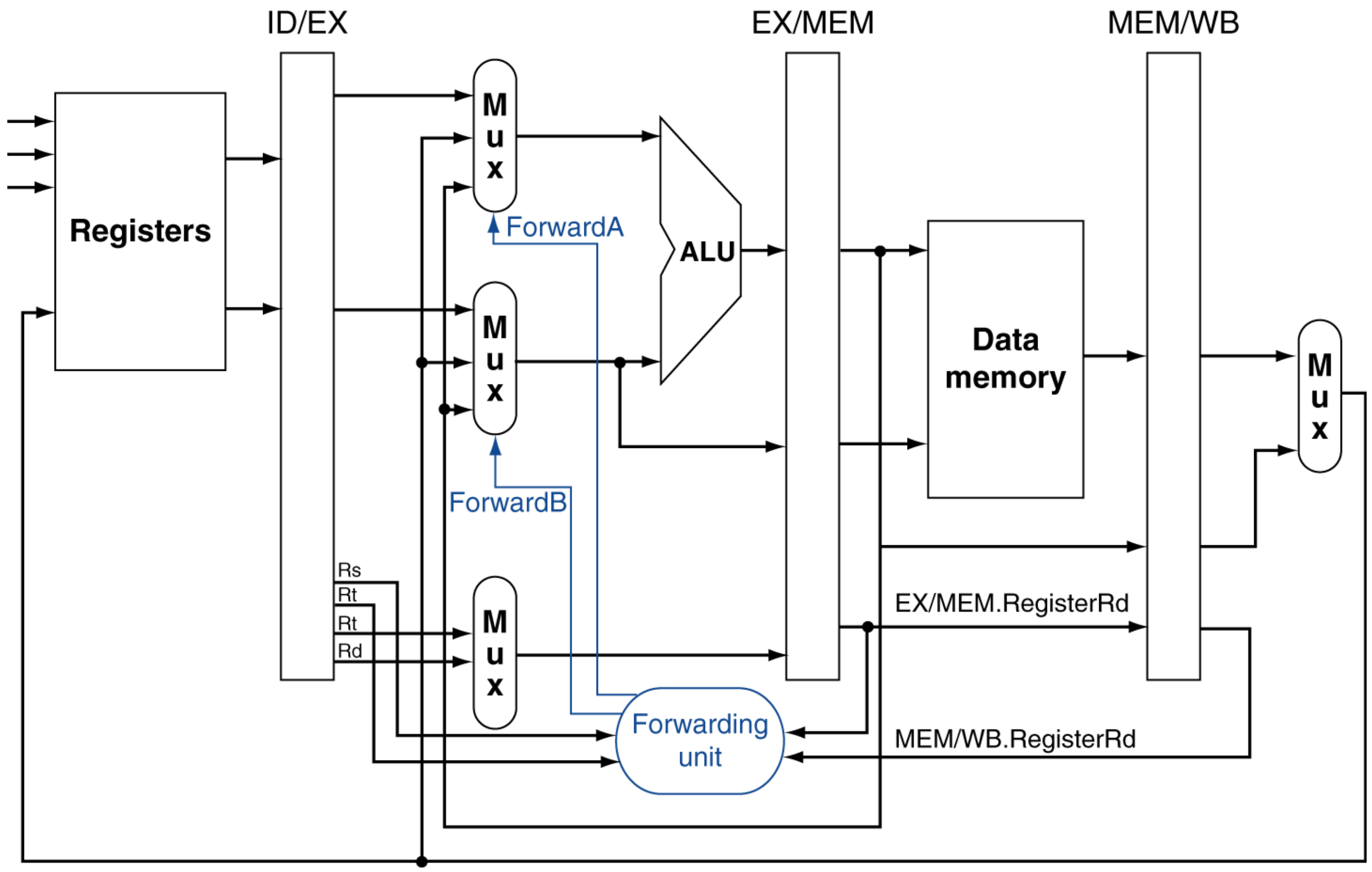
$\left.\begin{array}{l}\text{1a, 1b}\end{array}\right\}$ Fwd from EX/MEM pipeline reg

$\left.\begin{array}{l}\text{2a, 2b}\end{array}\right\}$ Fwd from MEM/WB pipeline reg

Detecting the Need to Forward

- 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

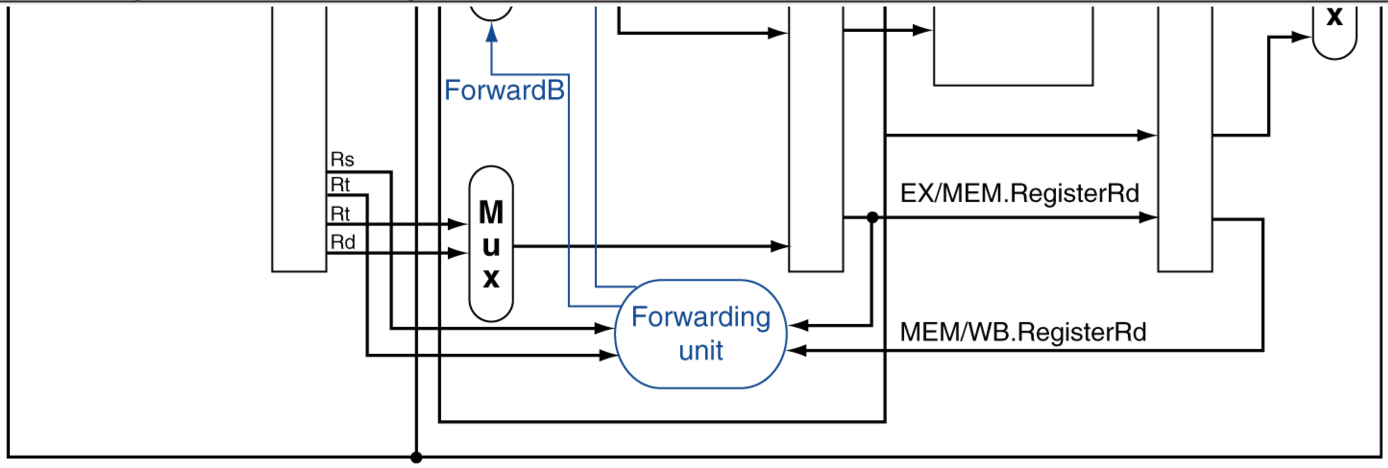
Forwarding Paths



b. With forwarding

Forwarding Paths

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.



b. With forwarding

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

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

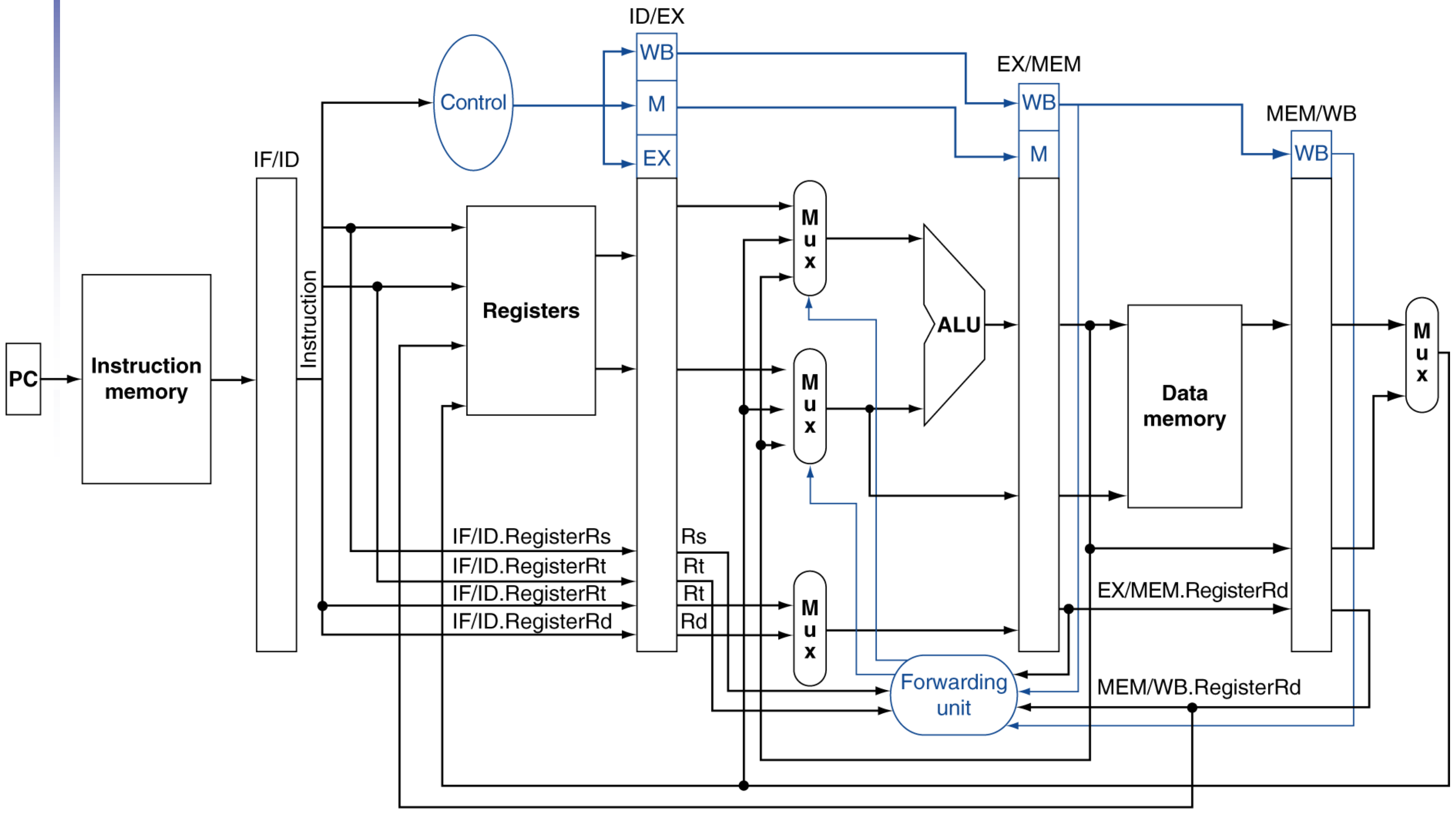
Revised Forwarding Condition

- MEM hazard

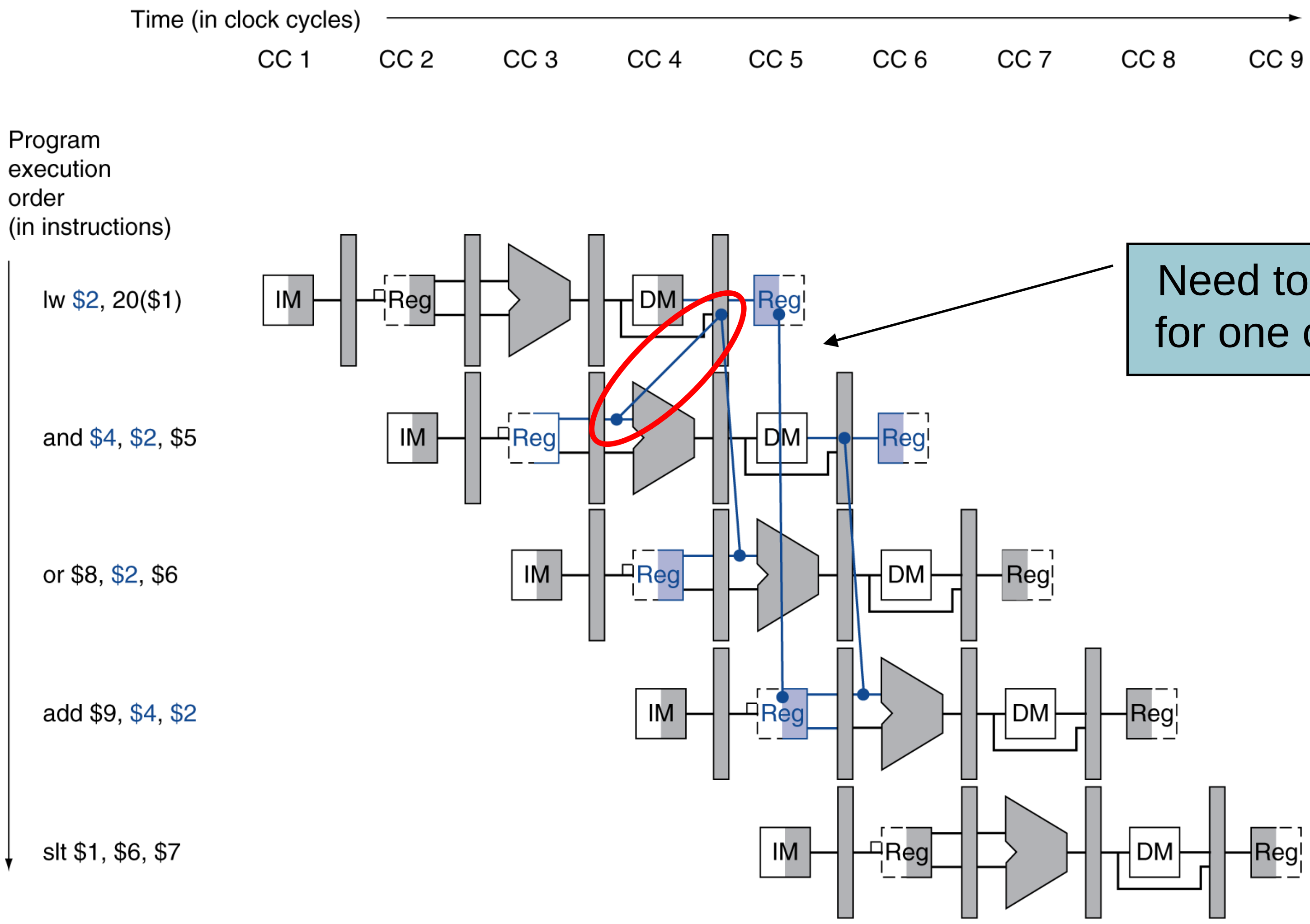
  -
    ```
    if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
            and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
     ForwardA = 01
    ```

  -
    ```
    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

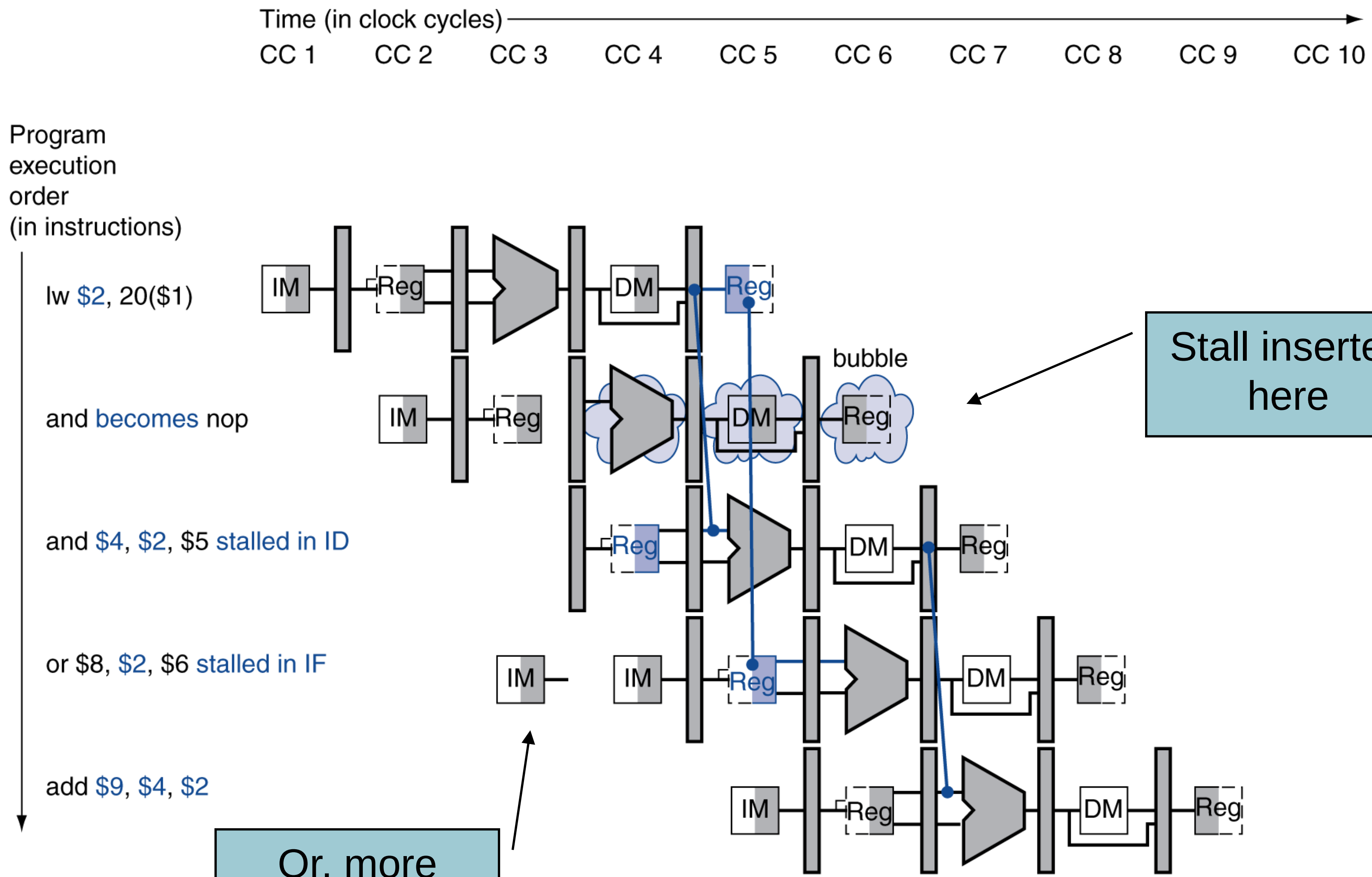


Need to stall for one cycle

Load-Use 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-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble

Stall/Bubble in the Pipeline



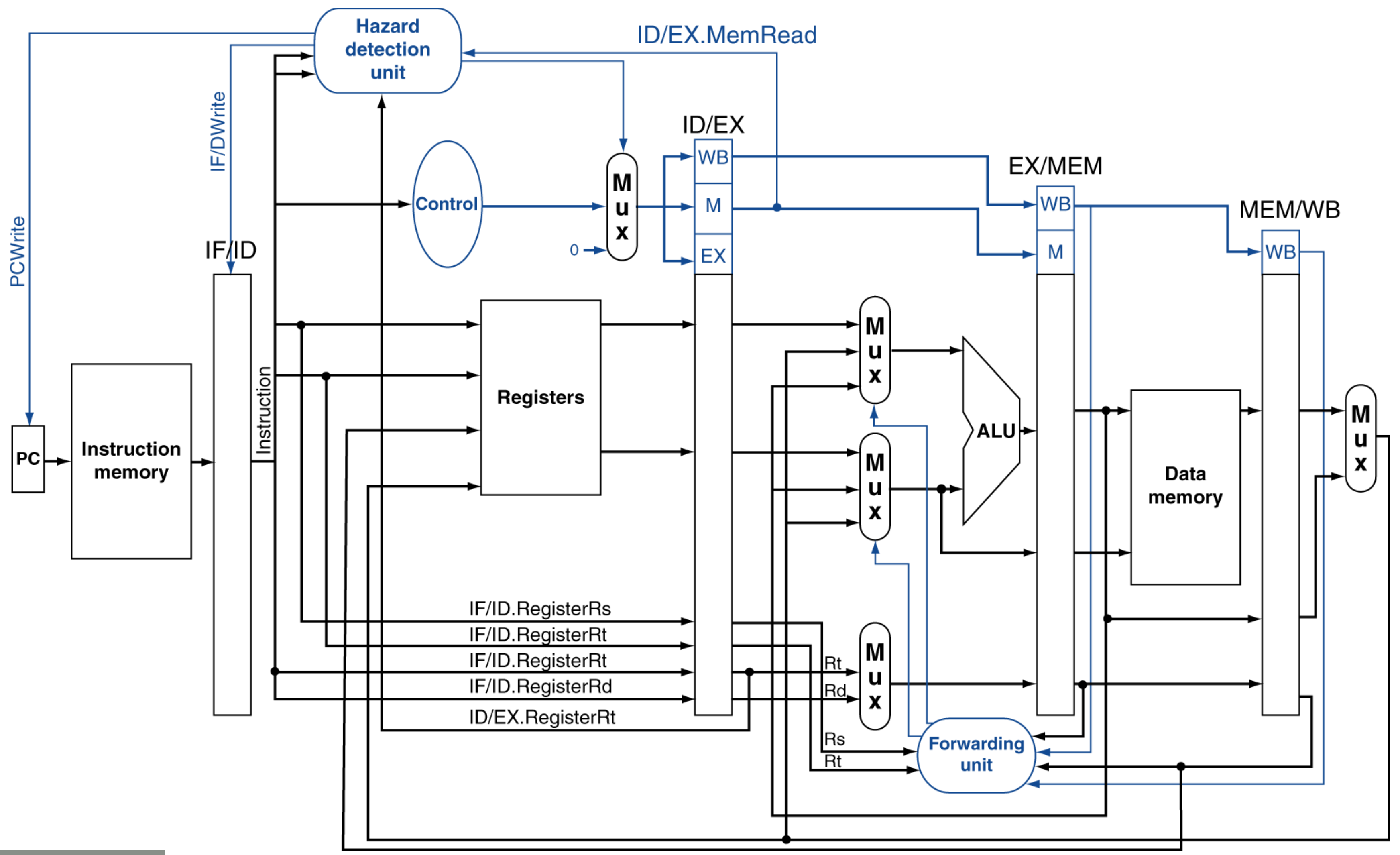
Stall inserted here

Or, more accurately…

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

Datapath with Hazard Detection



Stalls and Performance

The BIG Picture

- 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

Concluding Remarks

- Data hazards exist when an instruction depends on result of previous instruction or the one before.
- Data hazards can be solved using forwarding or forwarding combined with stall.

1. Forwarding is implemented using forwarding unit for two situations:
   - EX hazard: forward result from EX stage in previous cycle to EX stage (pipeline registers needed: EX/MEM and ID/EX)
   - MEM hazard: forward result from MEM stage in previous cycle to EX stage (pipeline registers needed: MEM/WB and ID/EX)
2. Stall is implemented using hazard detection unit.

Problems to Solve

- 4.9 (note that “With ALU-ALU Forwarding Only” means EX hazard only)
- 4.12, 4.13