Computer Organization (IS F242)

Lect 43: Pipelining

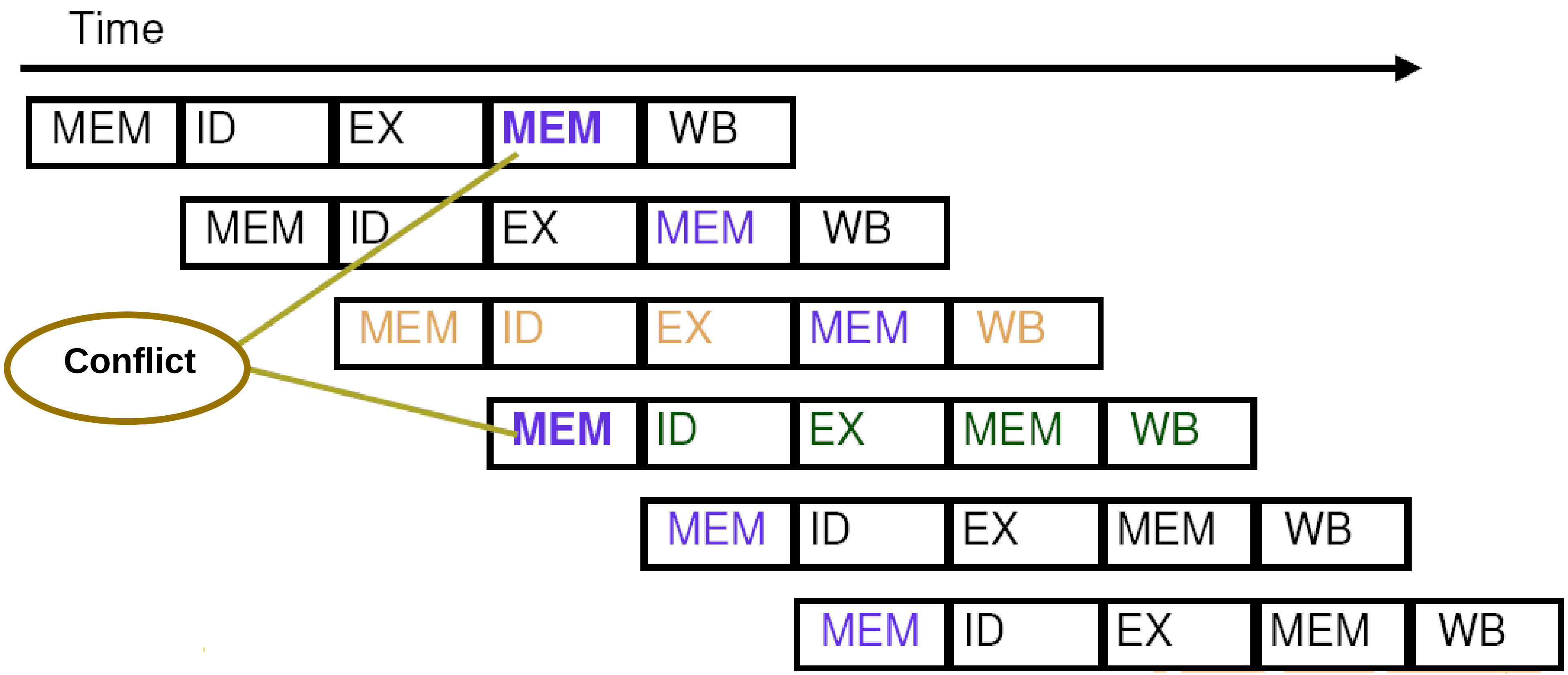
Pipeline Hazards

- Situations that prevent starting the next instruction in the next cycle
- Hazards reduce the ideal speedup gained from pipelining and are classified into three classes:
  - *Structural hazards: Arise from hardware resource conflicts when* the available hardware cannot support all possible combinations of instructions (A required resource is busy).
  - *Data hazards: Arise when an instruction depends on the results* of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
    - Need to wait for its previous instruction to complete its data read/write
  - *Control hazards: Arise from the pipelining of conditional* branches and other instructions that change the PC.
    - Deciding on control action depends on previous instruction

Structural Hazards

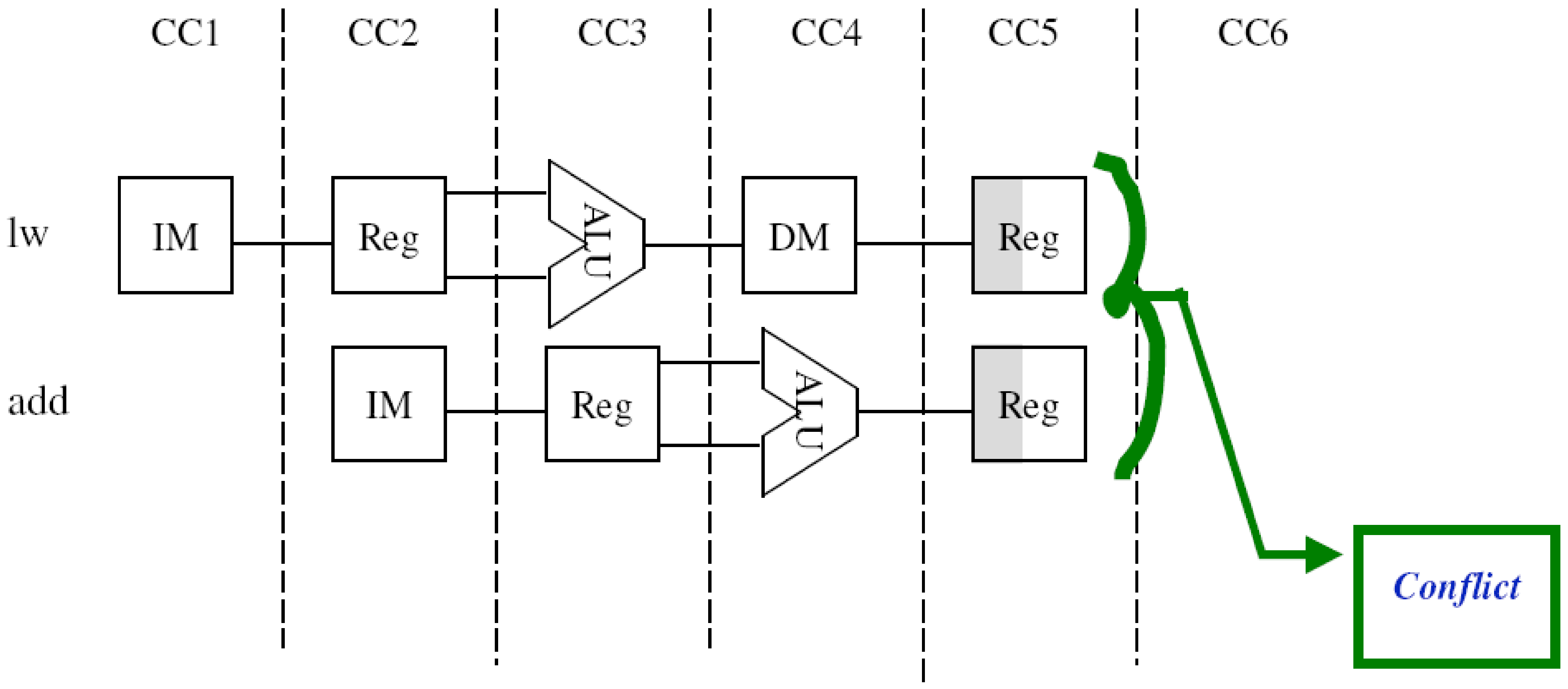
- Example: In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches

If a single memory unit is used



- This conflict is known as Structural Hazard
  - Memory, Execution, Evaluate Address

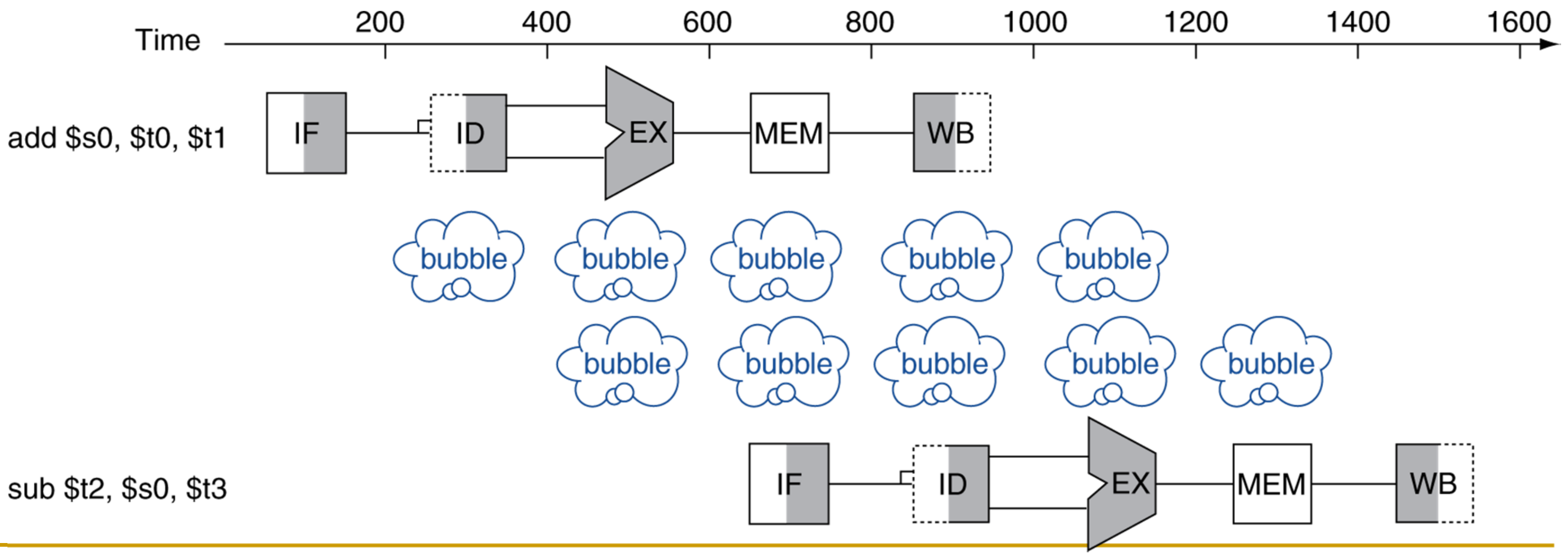
Mixed instructions in pipeline



Data Hazards

- A planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.
- An instruction depends on completion of data access by a previous instruction

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

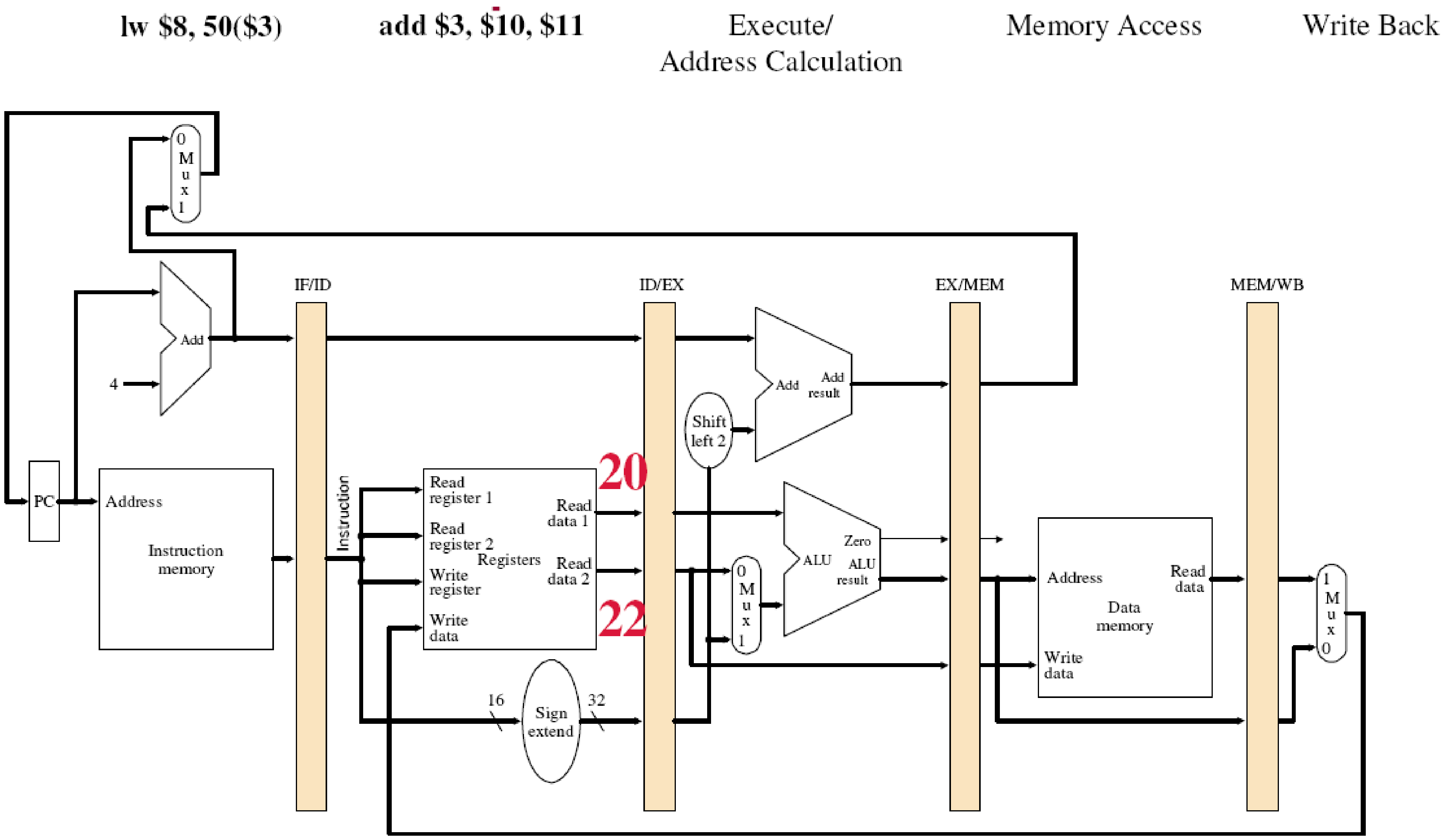


Data Hazards

- Suppose initially, register 10 holds the number 20, register 11 holds the number 22 and register 7 holds the number 14.
- What happens when...

```
add $3, $10, $11        - this should add 20 + 22,
                          putting result 42 into r3
lw $8, 50($3)           - this should load r8 from
                          memory location 42+50 = 92
sub $11, $8, $7         - this should subtract 14
                          from that just-loaded value
```

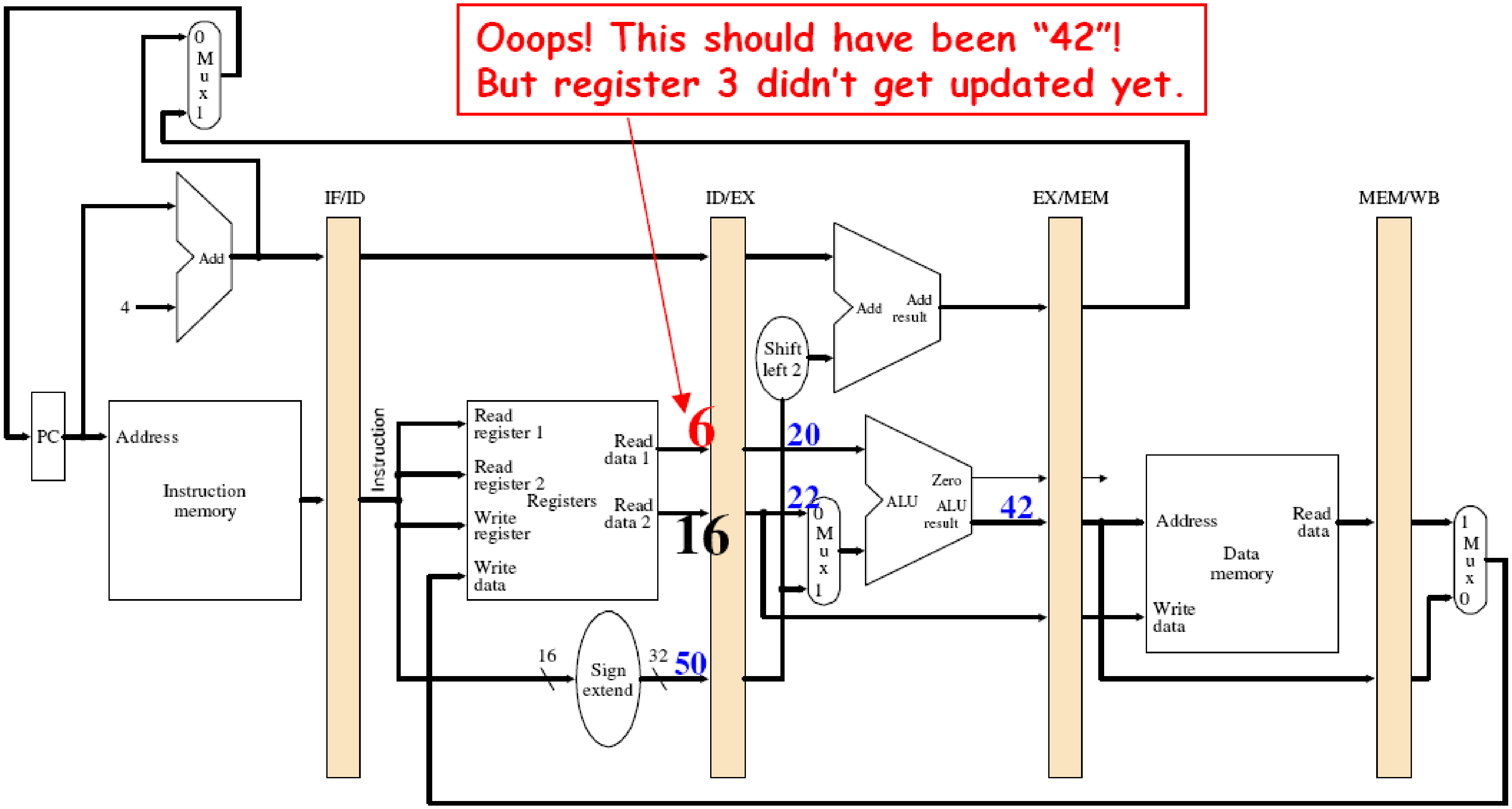
Pipeline in Execution



Pipeline in Execution

sub $11, $8, $7 lw $8, 50($3) add $3, $10, $11 Memory Access Write Back

Ooops! This should have been "42"!
But register 3 didn't get updated yet.

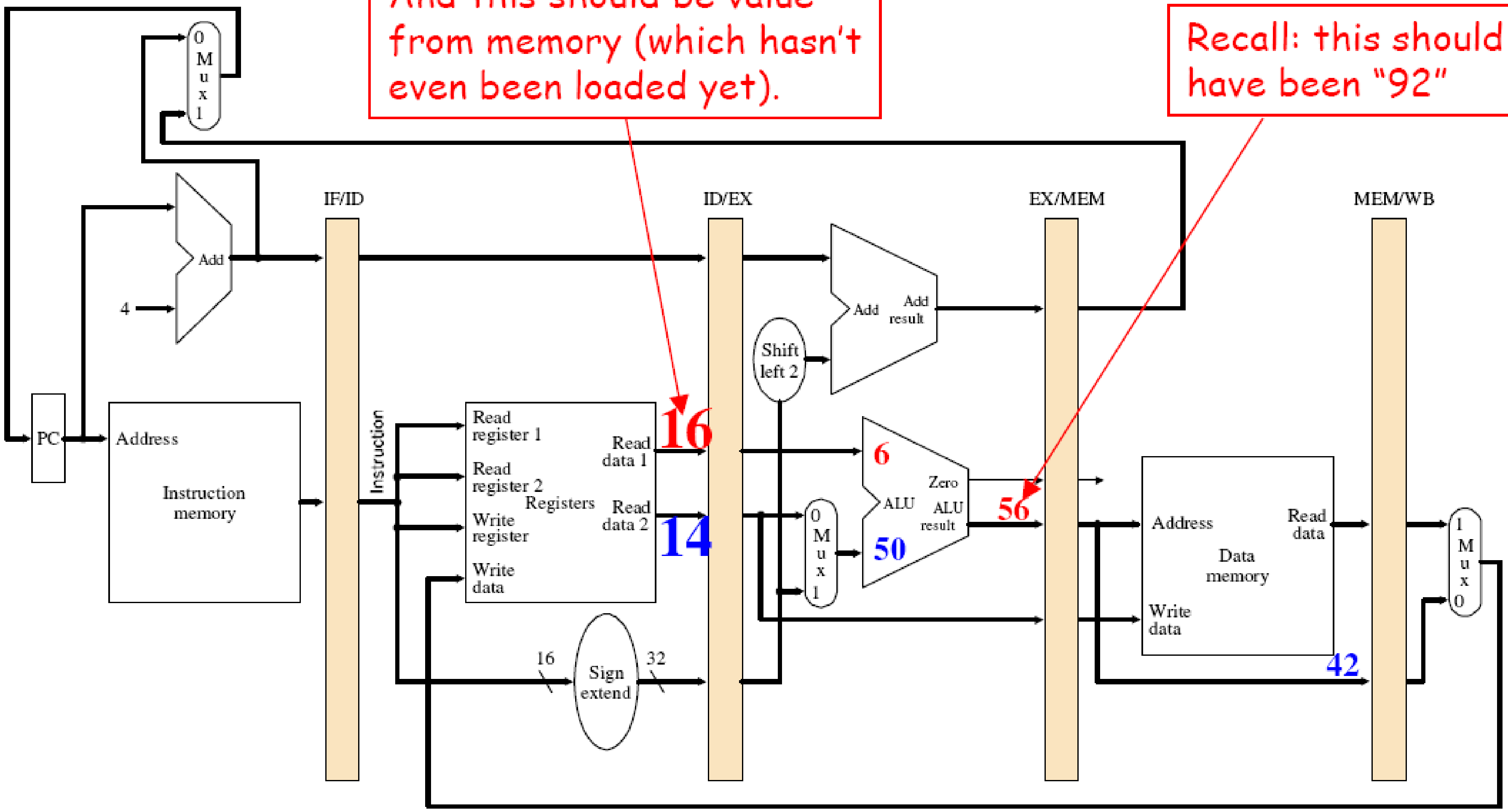


Pipeline in Execution

add $10, $1, $2 **sub $11, $8, $7** **lw $8, 50($3)** **add $3, $10, $11** Write Back

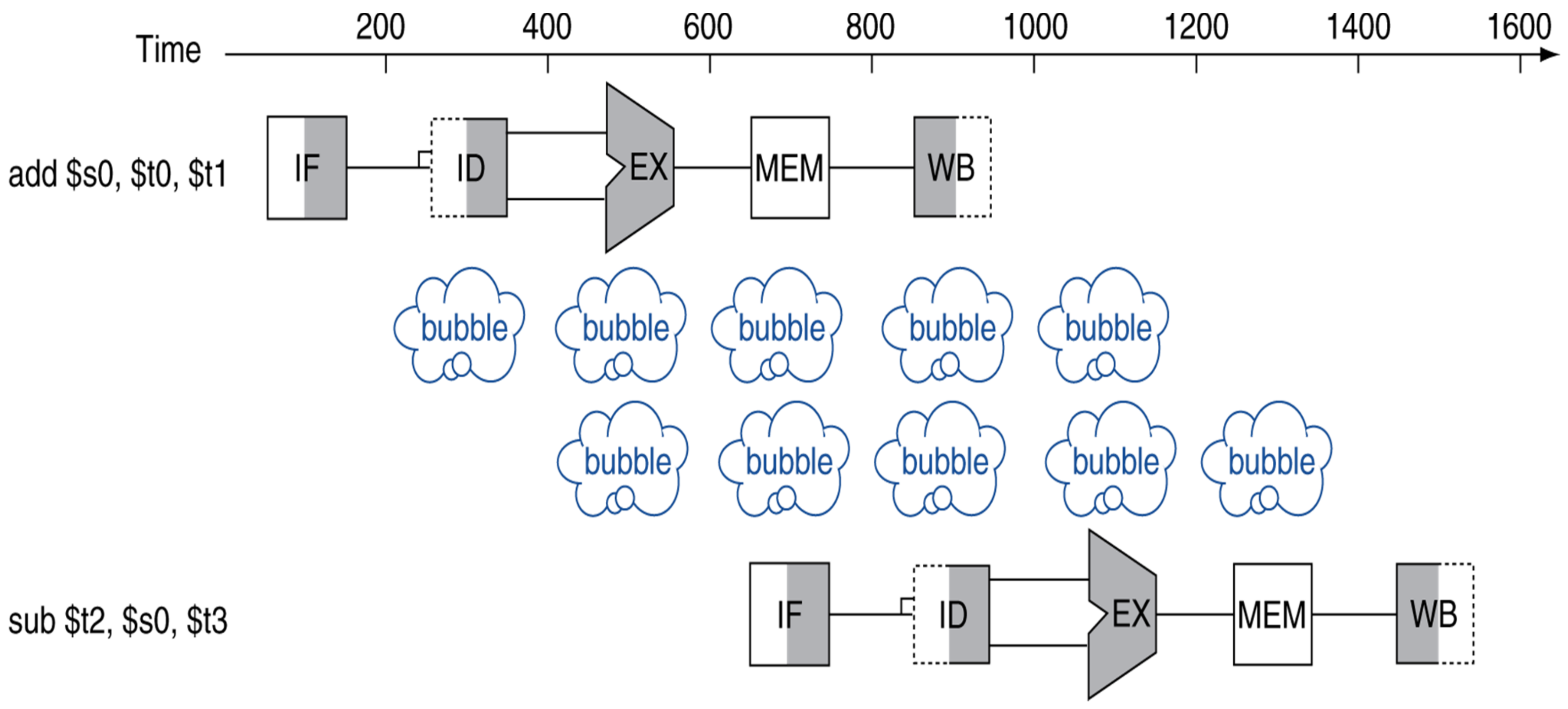
And this should be value from memory (which hasn't even been loaded yet).

Recall: this should have been "92"



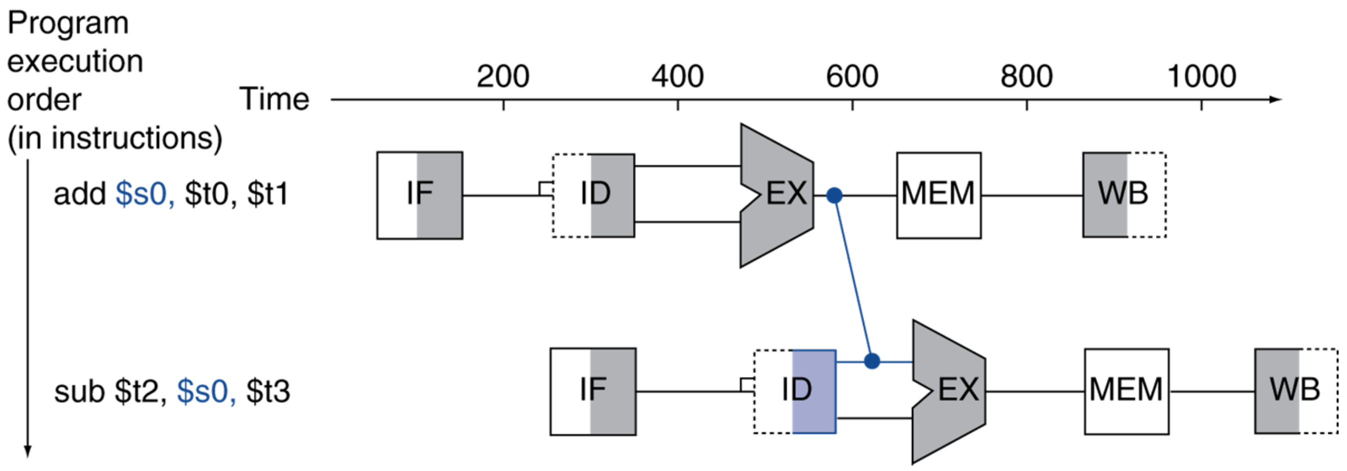
Data Hazards

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



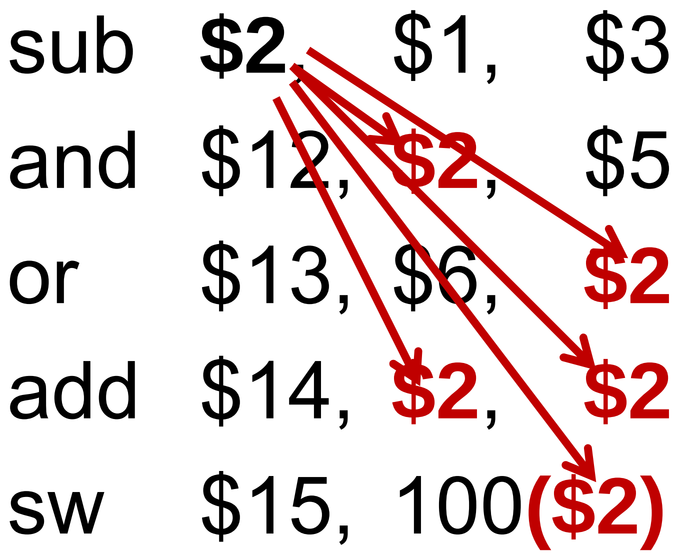
Forwarding (aka Bypassing)

- Adding extra hardware to retrieve the missing item early from the internal resources
- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



Data Hazards

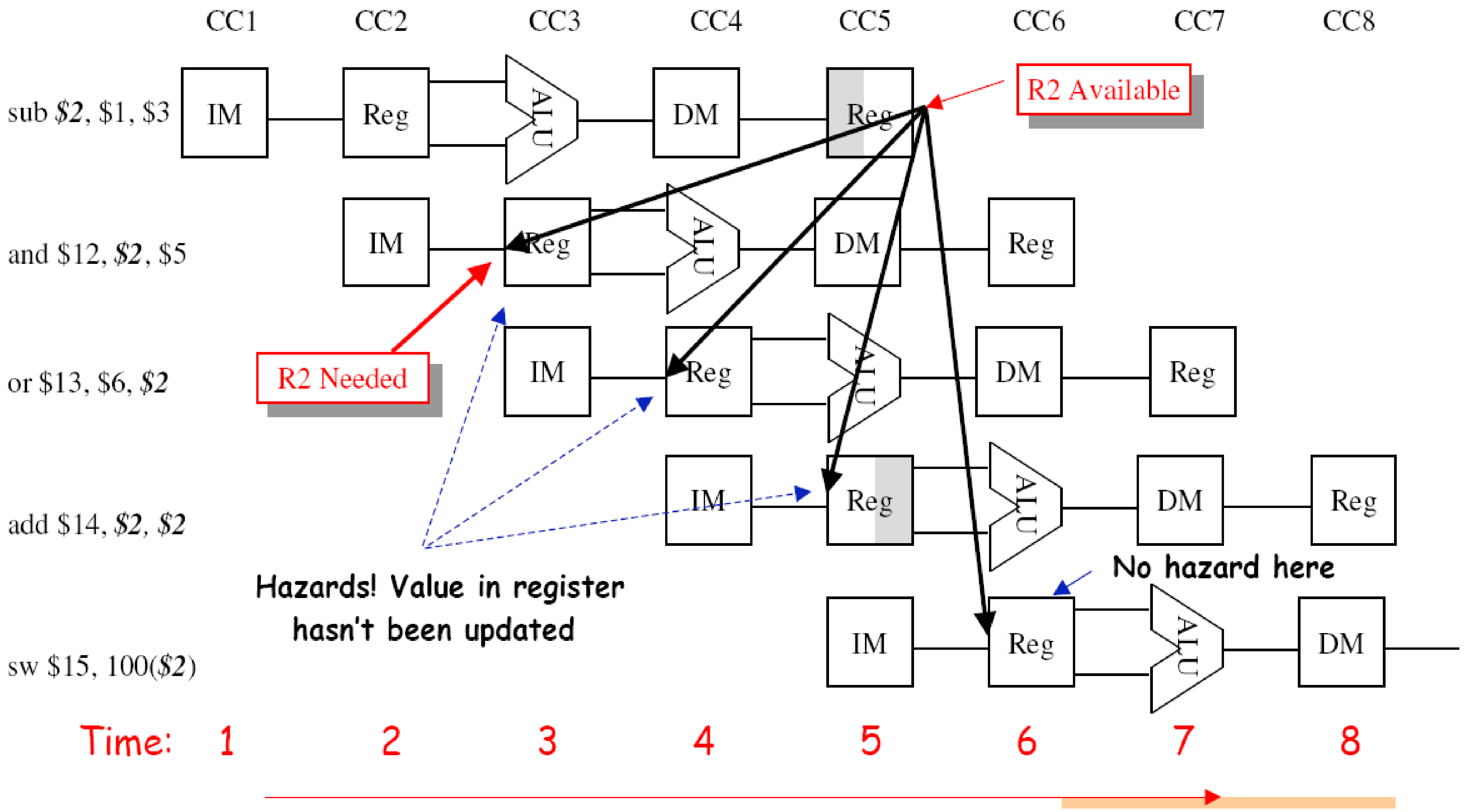
- Problem with starting the next instruction before first is finished
  - Data dependencies here that “go backward in time” create data hazards.



```
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?

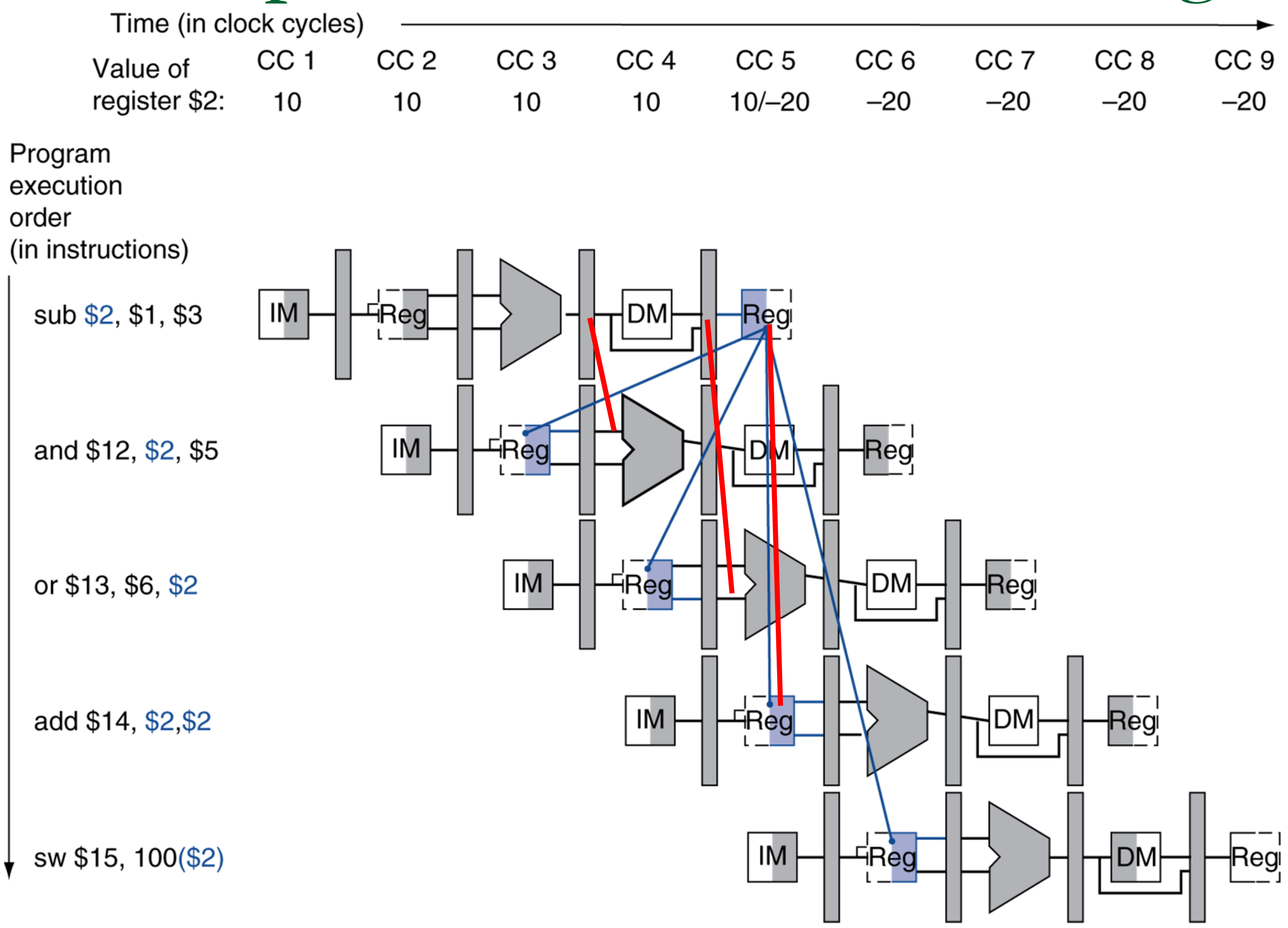
Data Hazards



Observation

- Register Write is in the first half of the clock cycle
- Register Read in the second half of the clock cycle.
  - Read delivers what is written
- To work properly, it should read $2 value during CC5 or later
  - Happens when dependence line goes backward in time

Dependencies & Forwarding



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

$\}$ Fwd from EX/MEM pipeline reg

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

$\}$ Fwd from MEM/WB pipeline reg

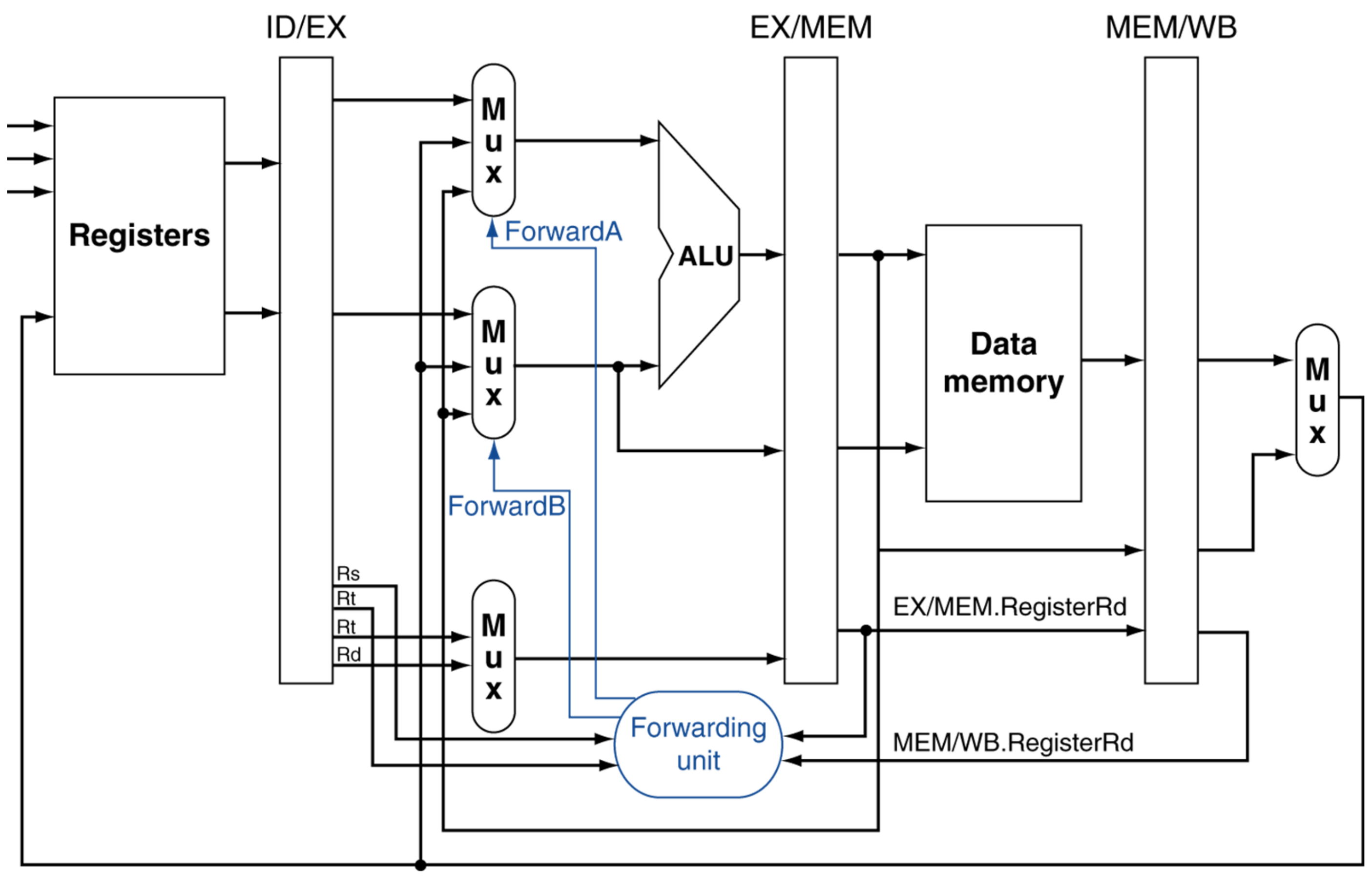
- sub – and is a type 1a hazard
- sub – or is a type 2b hazard
- The dependence on sub – add are not hazards because the register file supplies the proper data during the ID stage of add
- No hazard between sub and sw
  - sw reads $2 after sub writes $2

Detecting the Need to Forward

- Forward only if forwarding instruction will write to a register!
  - Only if RegWrite signal is active
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not $zero
  - In MIPS every use of $0 as an operand must yield an operand value of 0
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

- Can forward the proper data if
  - We can take inputs to the ALU from any pipeline register rather than just ID/EX
- We can achieve this By adding MUX to the input of ALU with proper control signals.
- Forwarding control will be in EX stage
  - Pass operand register number from ID stage via ID/EX pipeline register to determine whether to forward values

Forwarding Paths



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

- 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