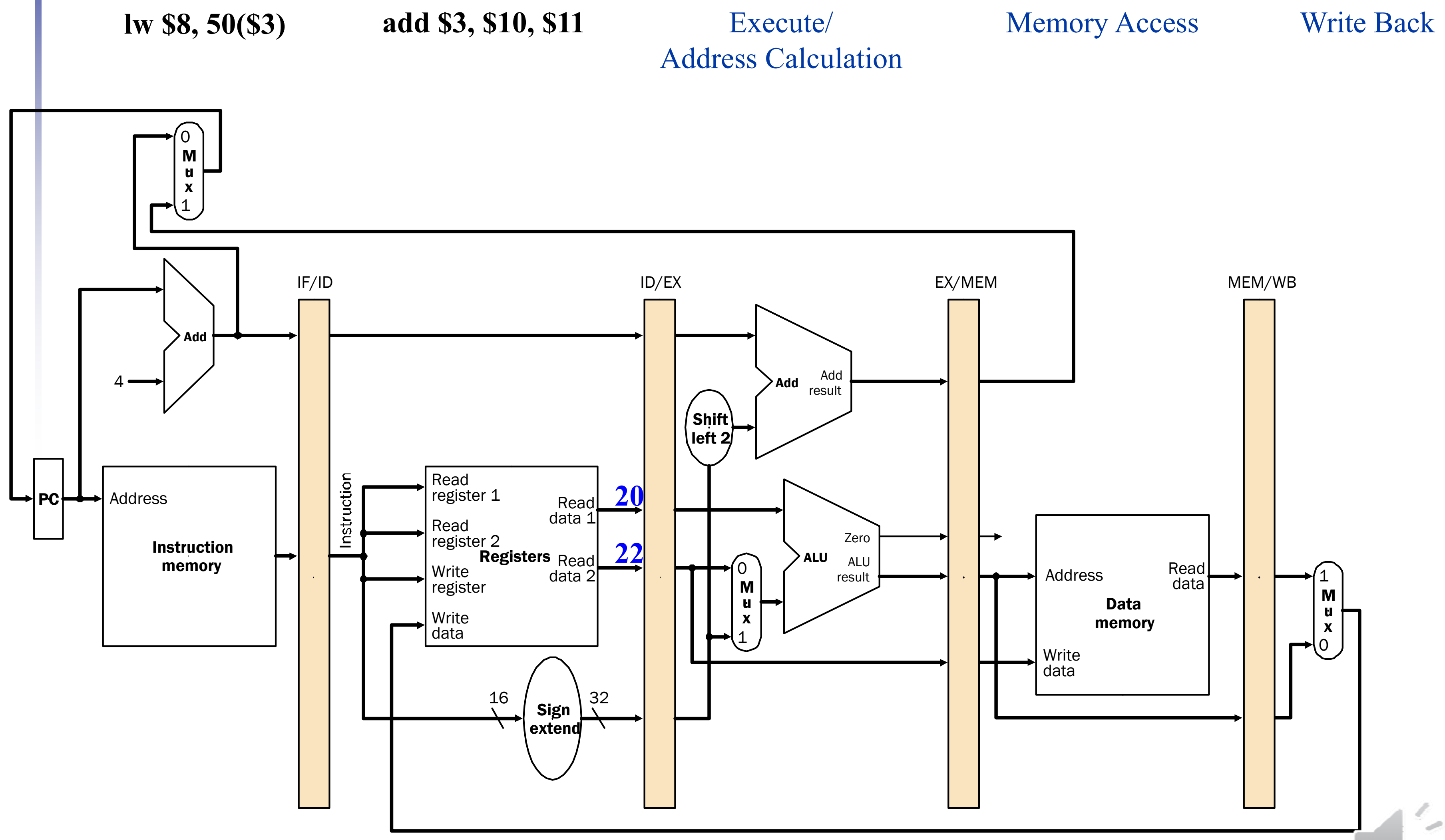
Chapter 4

The Processor

Hazards

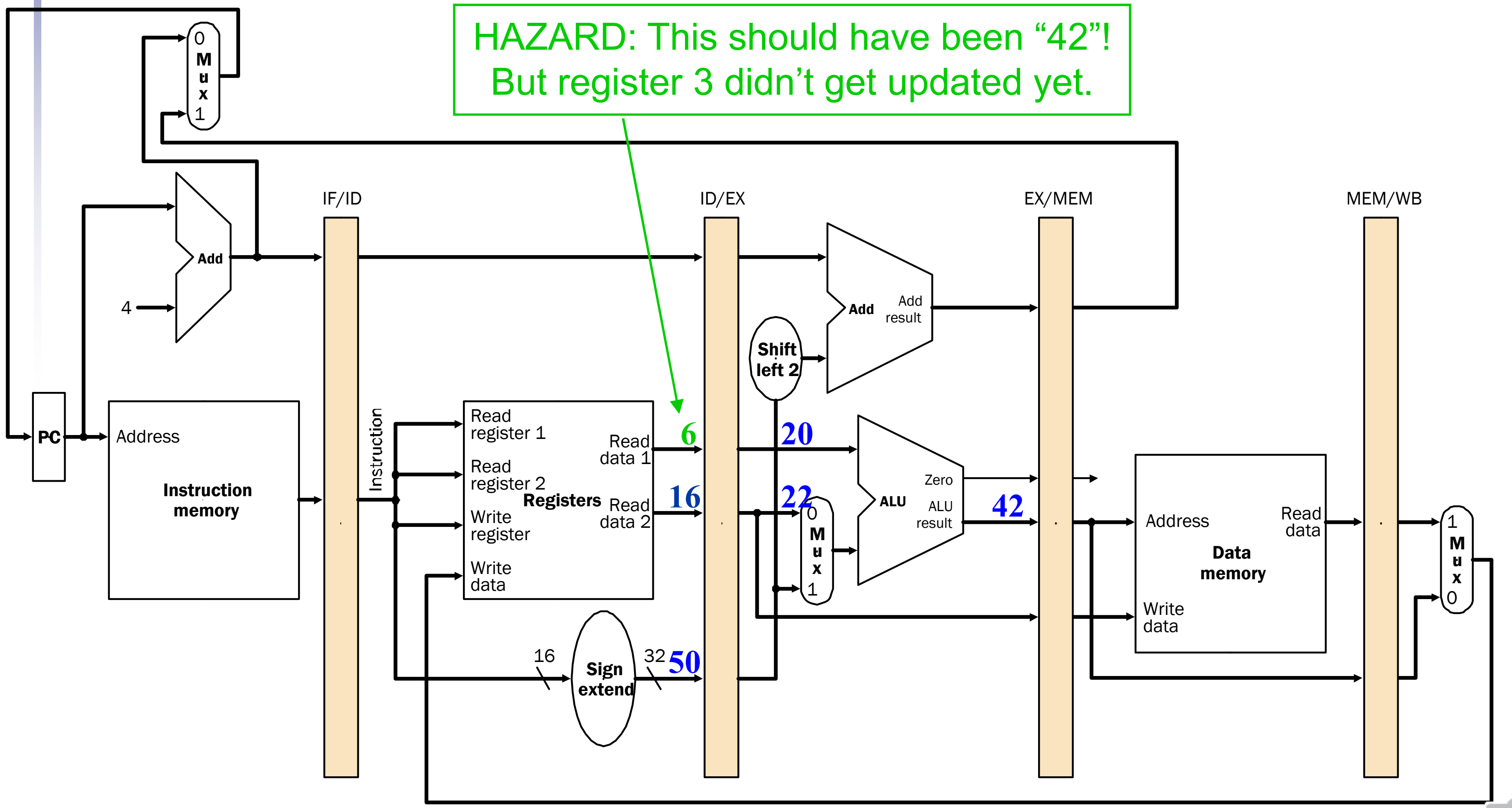
- Suppose initially, register $i holds the number 2i
- What happens when we see the following dynamic instruction sequence:
  - **add $3, $10, $11**
    - this should add 20 + 22, putting result 42 into $3
  - **lw $8, 50($3)**
    - this should load memory location 92 (42+50) into $8
  - **sub $11, $8, $7**
    - this should subtract 14 from that just-loaded val

The Pipeline in Execution



The Pipeline in Execution

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

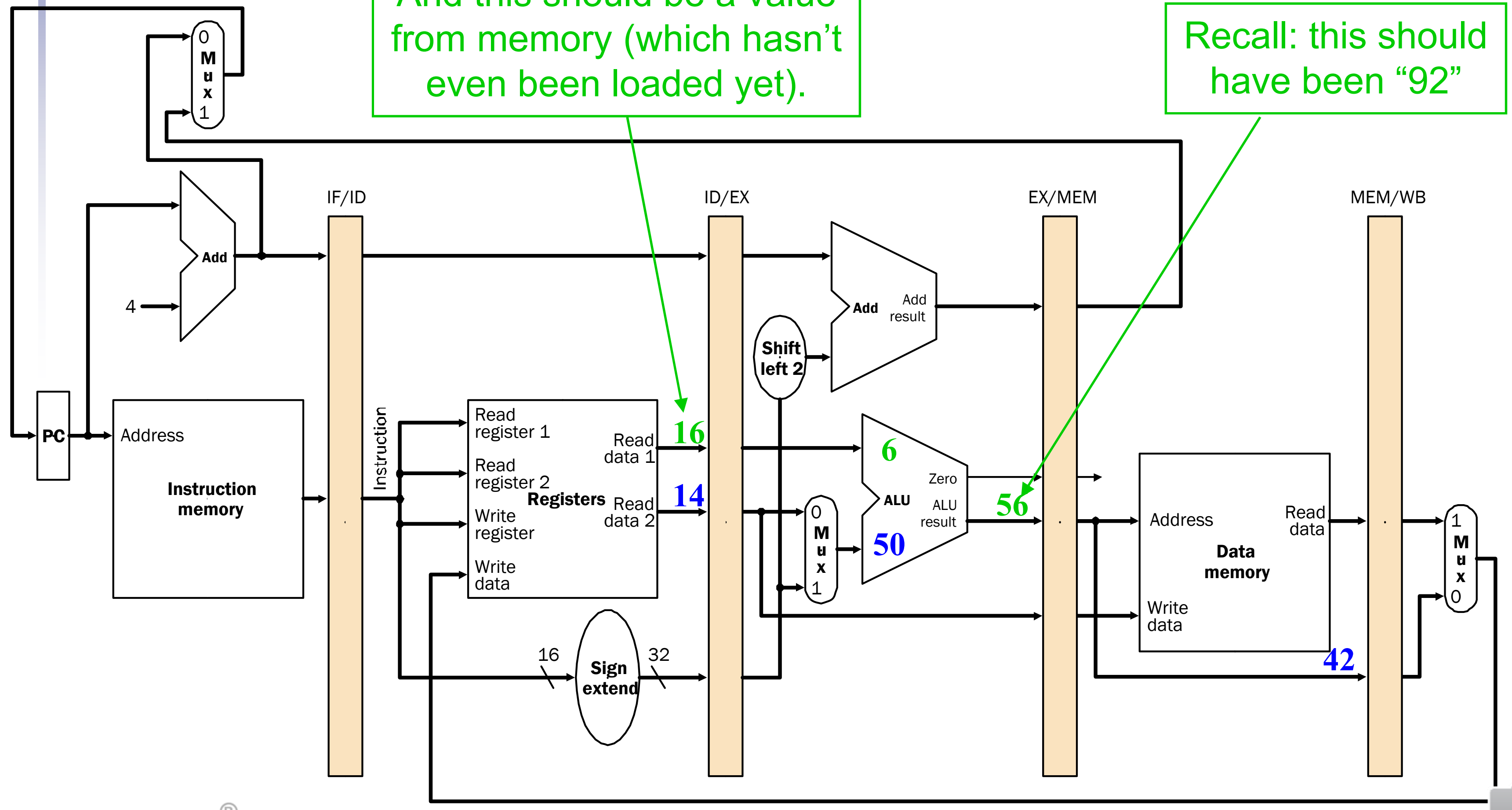


The Pipeline in Execution

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

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

Recall: this should have been "92"



Hazards

- Situations that 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
- Control hazard
  - Deciding on control action depends on previous instruction

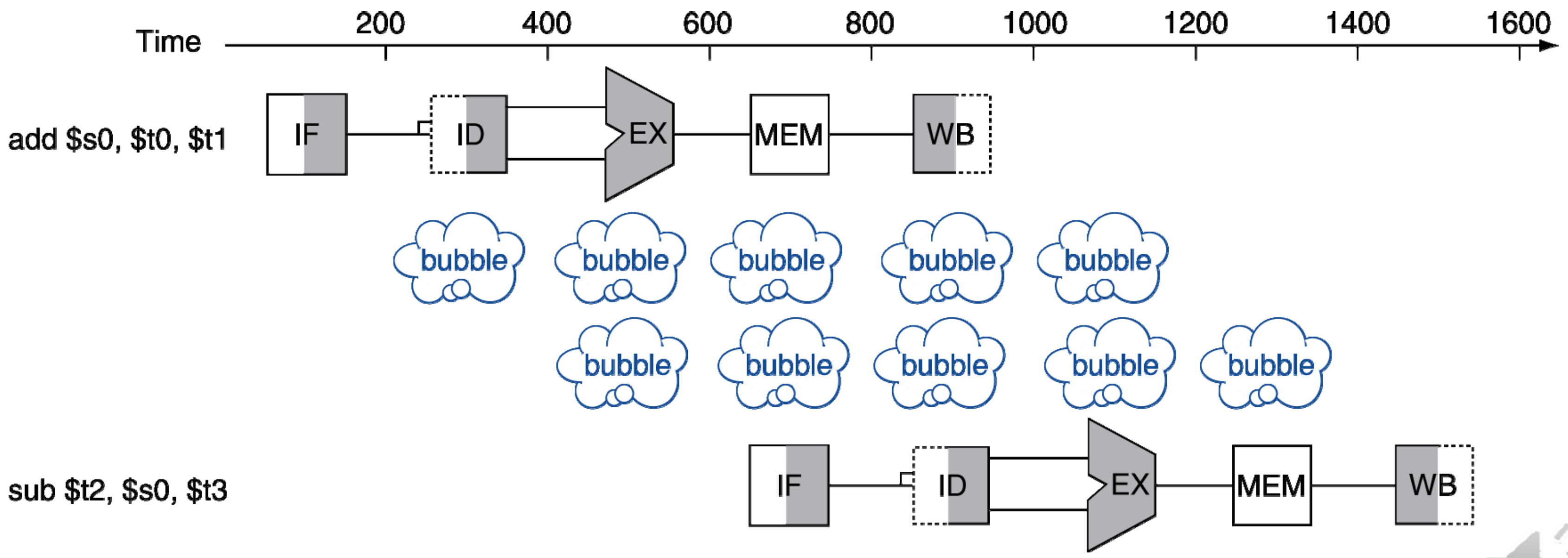
Structure Hazards

- 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
    - Would cause a pipeline "bubble"
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches

Data Hazards

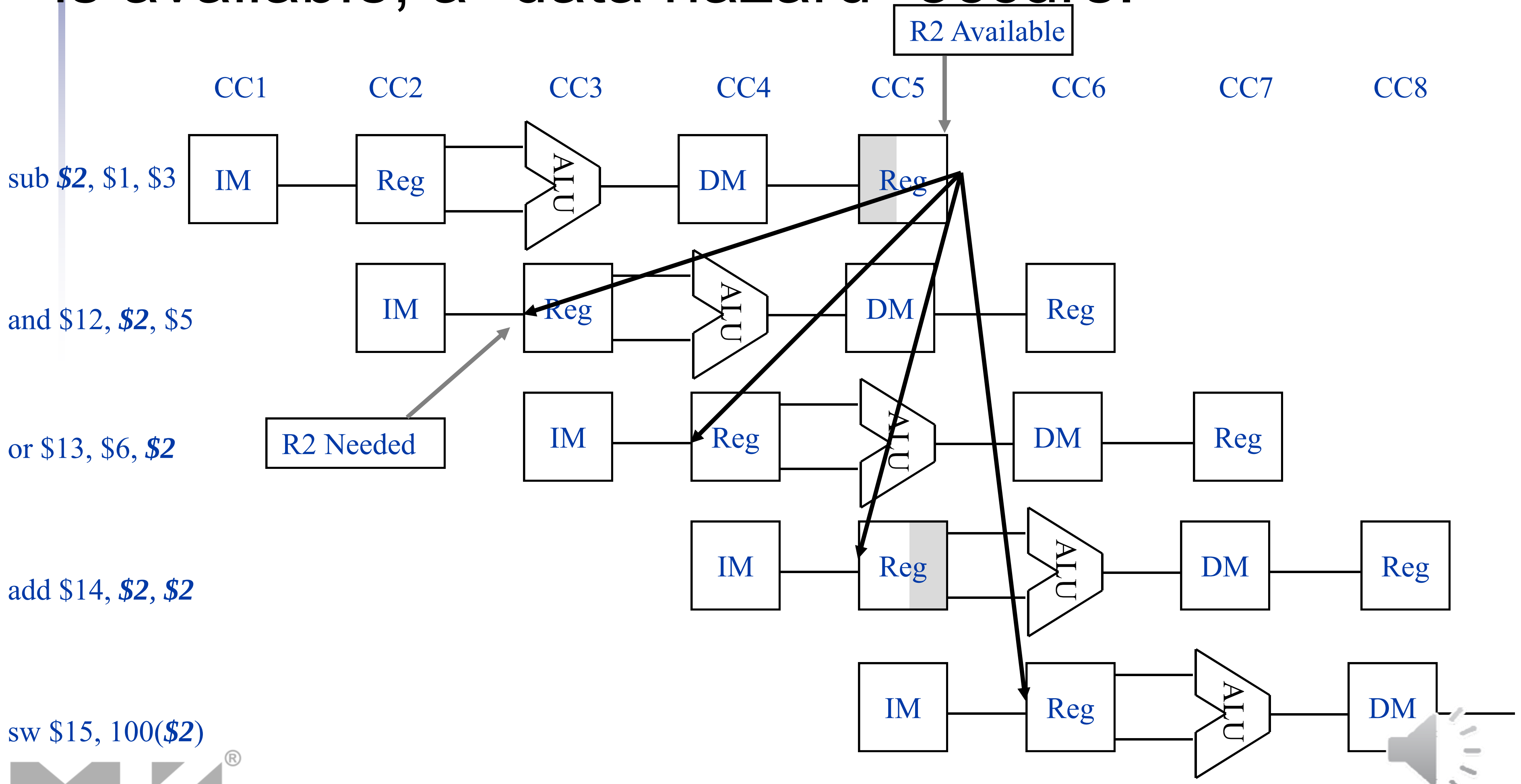
- An instruction depends on completion of data access by a previous instruction

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



Data Hazards

- When a result is needed in the pipeline before it is available, a “data hazard” occurs.



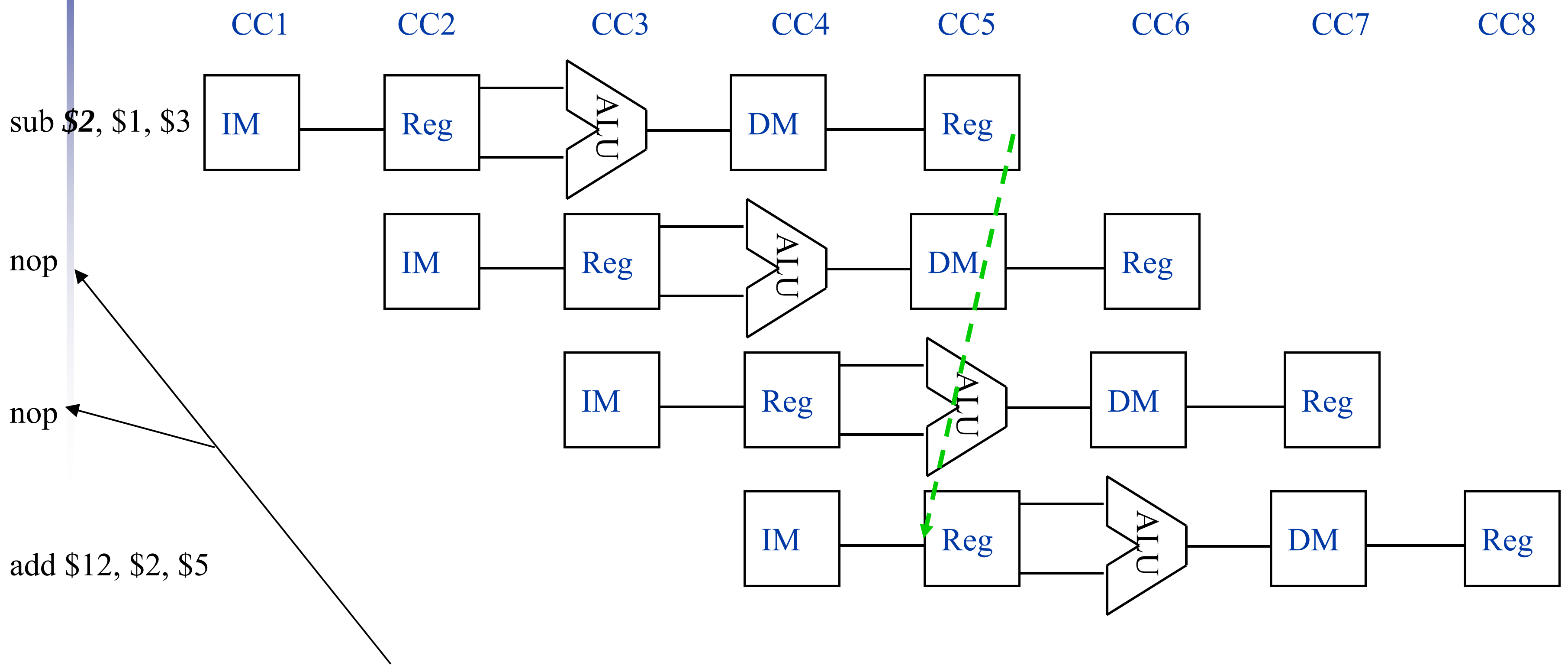
Chapter 4

The Processor

Dealing with Data Hazards

- In Software
  - insert independent instructions (or no-ops)
- In Hardware
  - insert bubbles (i.e. stall the pipeline)
  - data forwarding

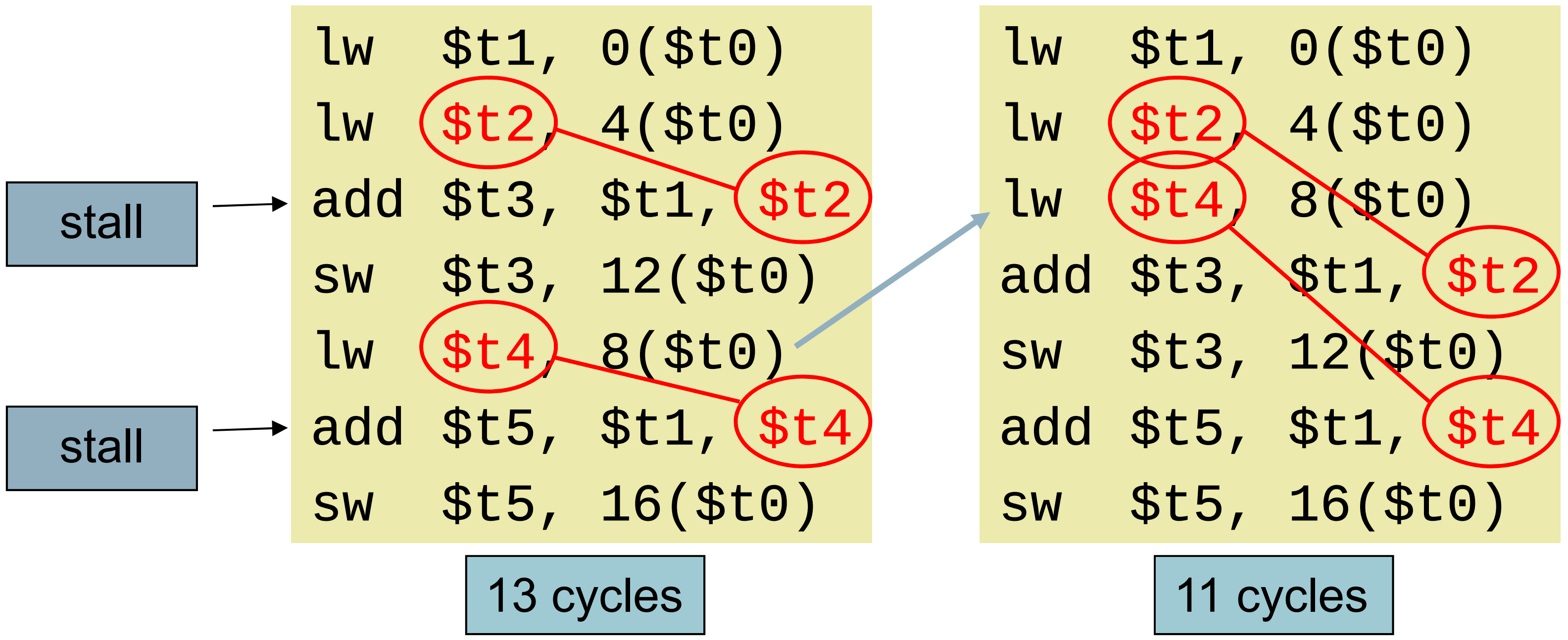
Dealing with Data Hazards in Software



Insert enough no-ops (or other instructions that don't use register 2) so that data hazard doesn't occur,

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for A `A = B + E; C = B + F;`



```
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)
```

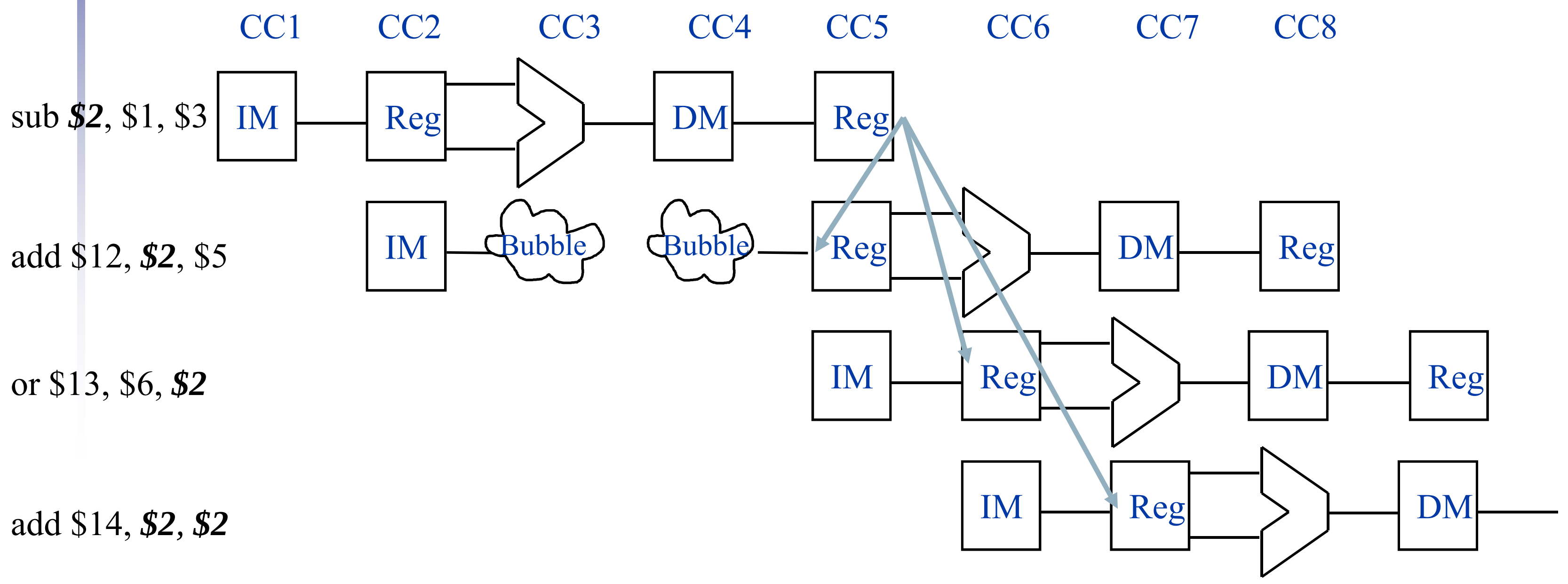
Where are No-ops needed?

- sub $2, $1,$3
- and $4, $2,$5
- or $8, $2,$6
- add $9, $4,$2
- slt $1, $6,$7

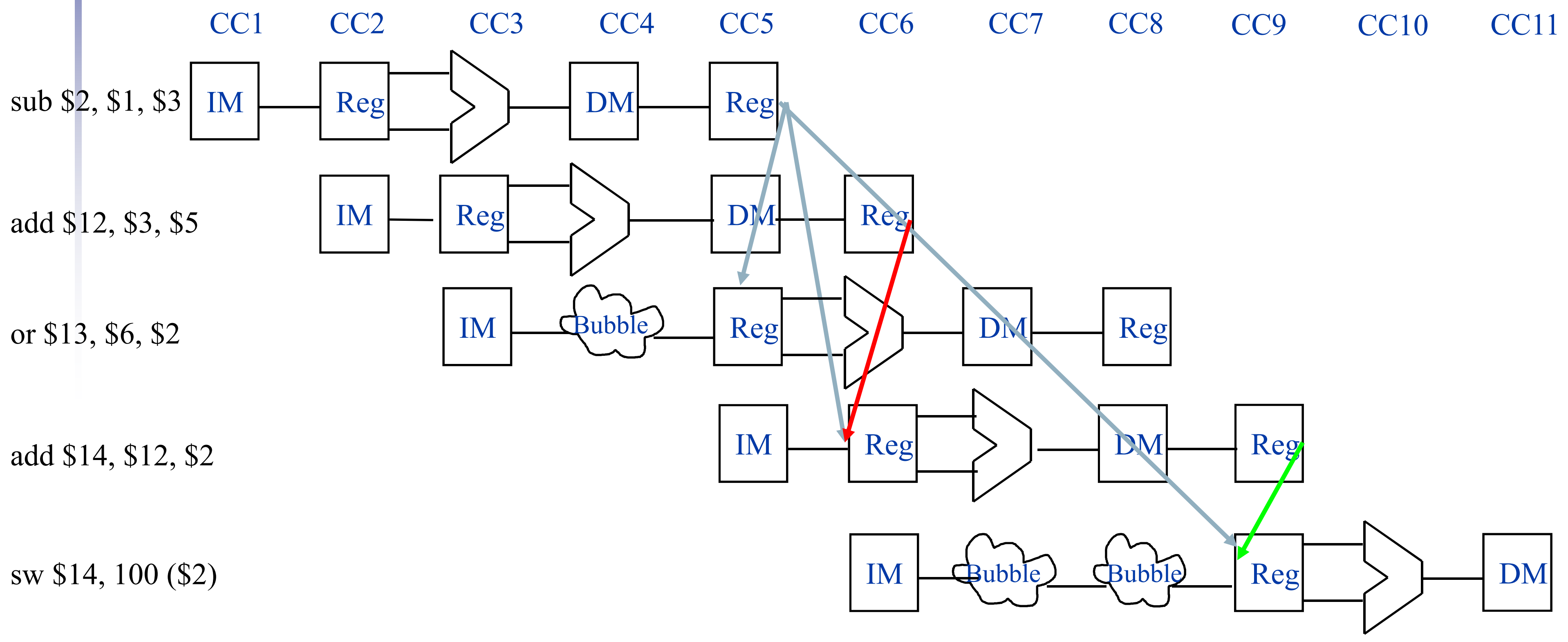
- Are no-ops really necessary?

Handling Data Hazards in Hardware

- Stall the pipeline



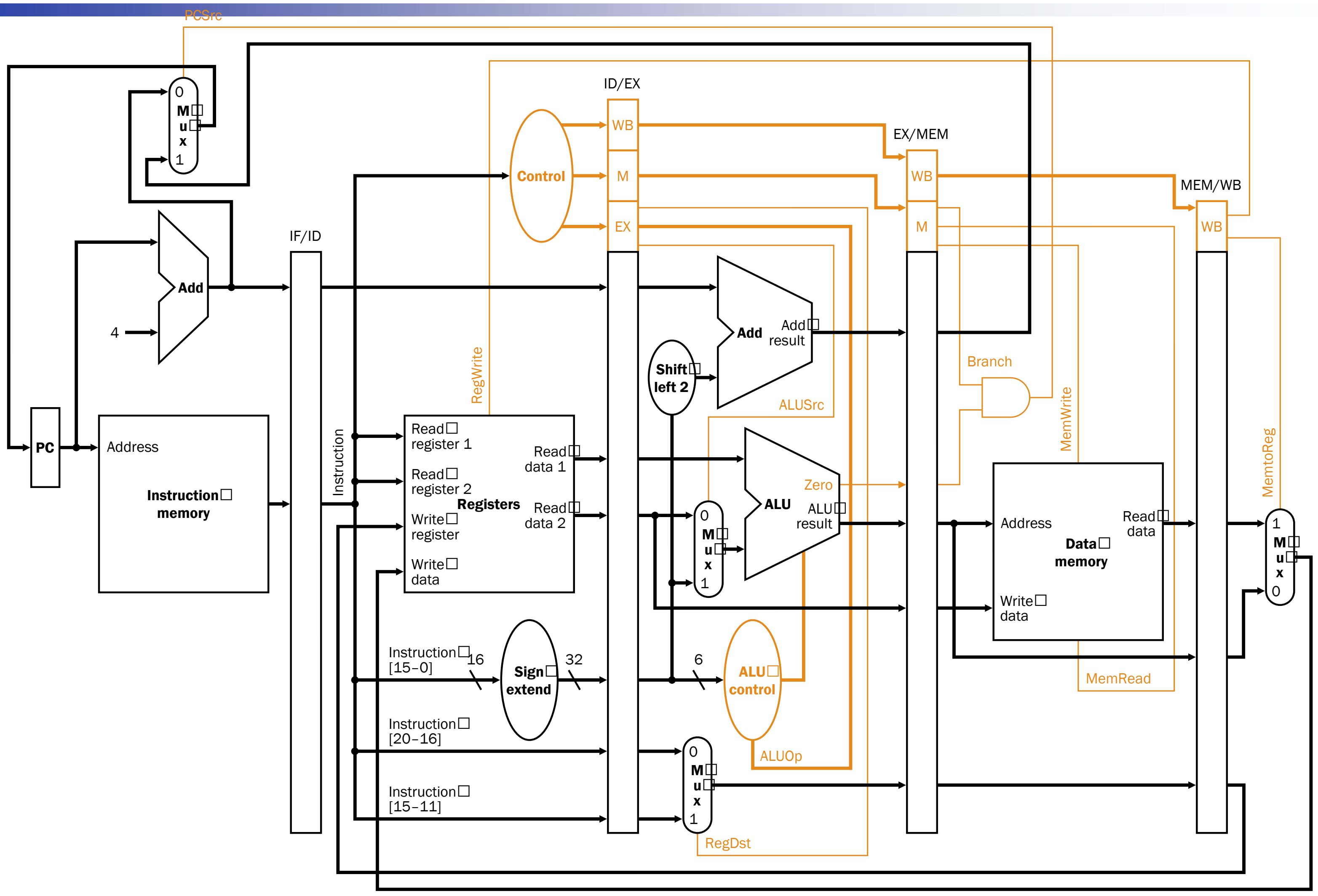
Handling Data Hazards in Hardware



Pipeline Stalls

- To insure proper pipeline execution in light of register dependences, we must:
  - Detect the hazard
  - Stall the pipeline
    - prevent the IF and ID stages from making progress
      - the ID stage because we can't go on until the dependent instruction completes correctly
      - the IF stage because we do not want to lose any instructions.

The Pipeline



What comparisons tell us when to stall?

Stalling the Pipeline

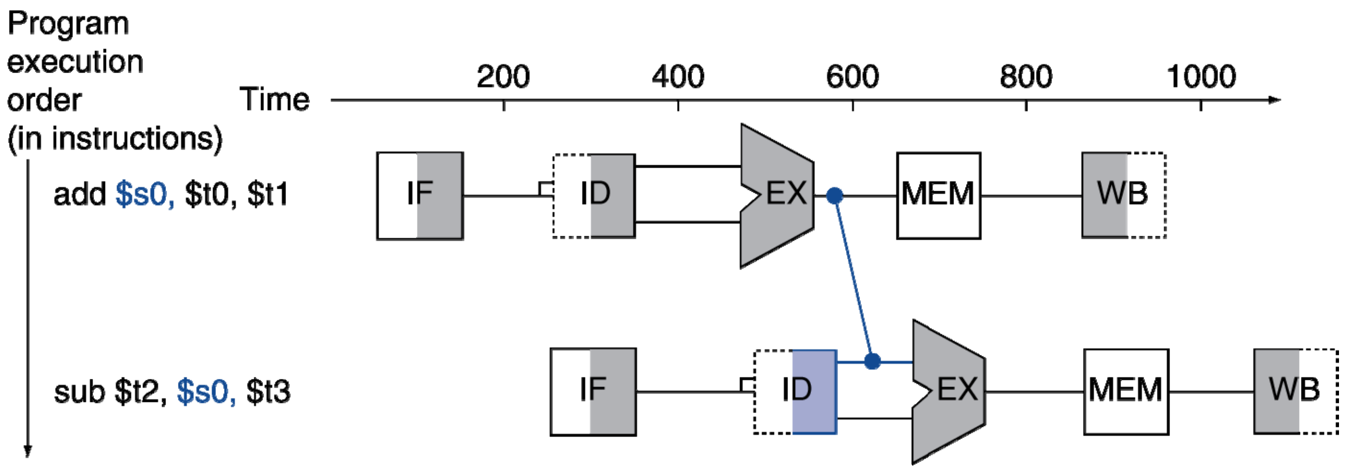
- Prevent the IF and ID stages from proceeding
  - don’t write the PC (PCWrite = 0)
  - don’t rewrite IF/ID register (IF/IDWrite = 0)
- Insert “nops”
  - set all control signals propagating to EX/MEM/WB to zero

Chapter 4

The Processor

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



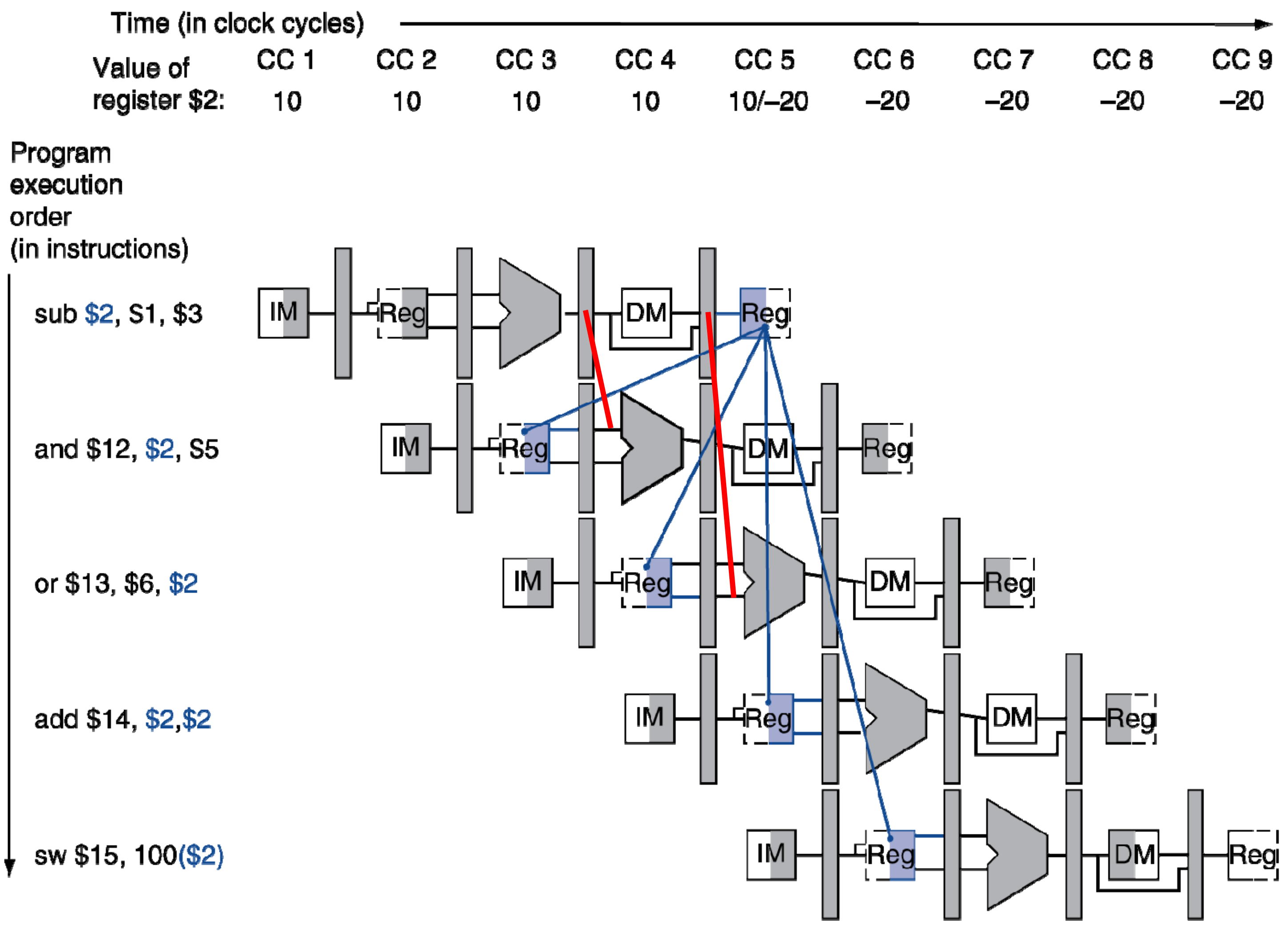
Data Hazards in ALU Instructions

- 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



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

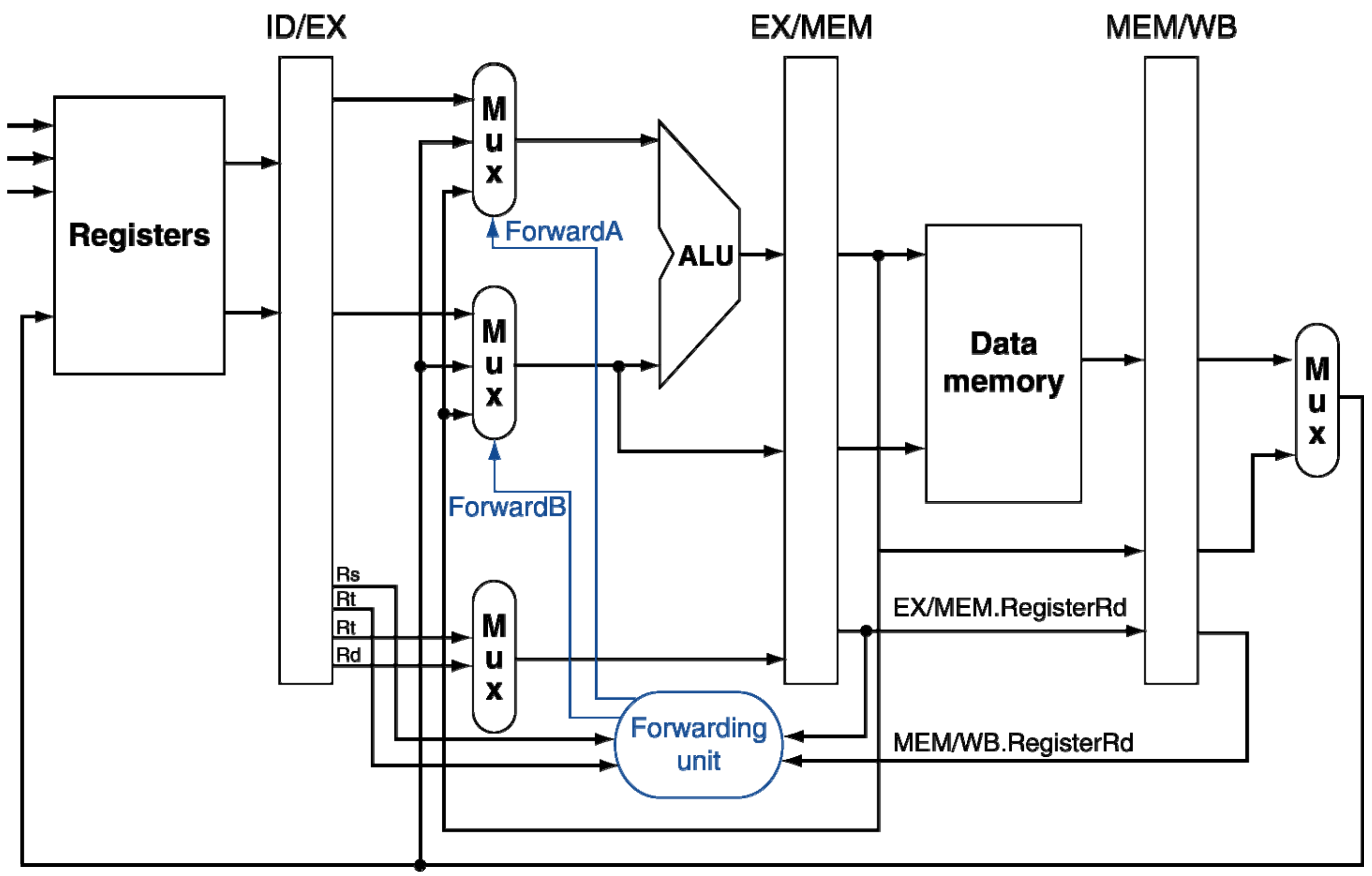
Fwd from EX/MEM pipeline reg (1a, 1b)

Fwd from MEM/WB pipeline reg (2a, 2b)

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 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

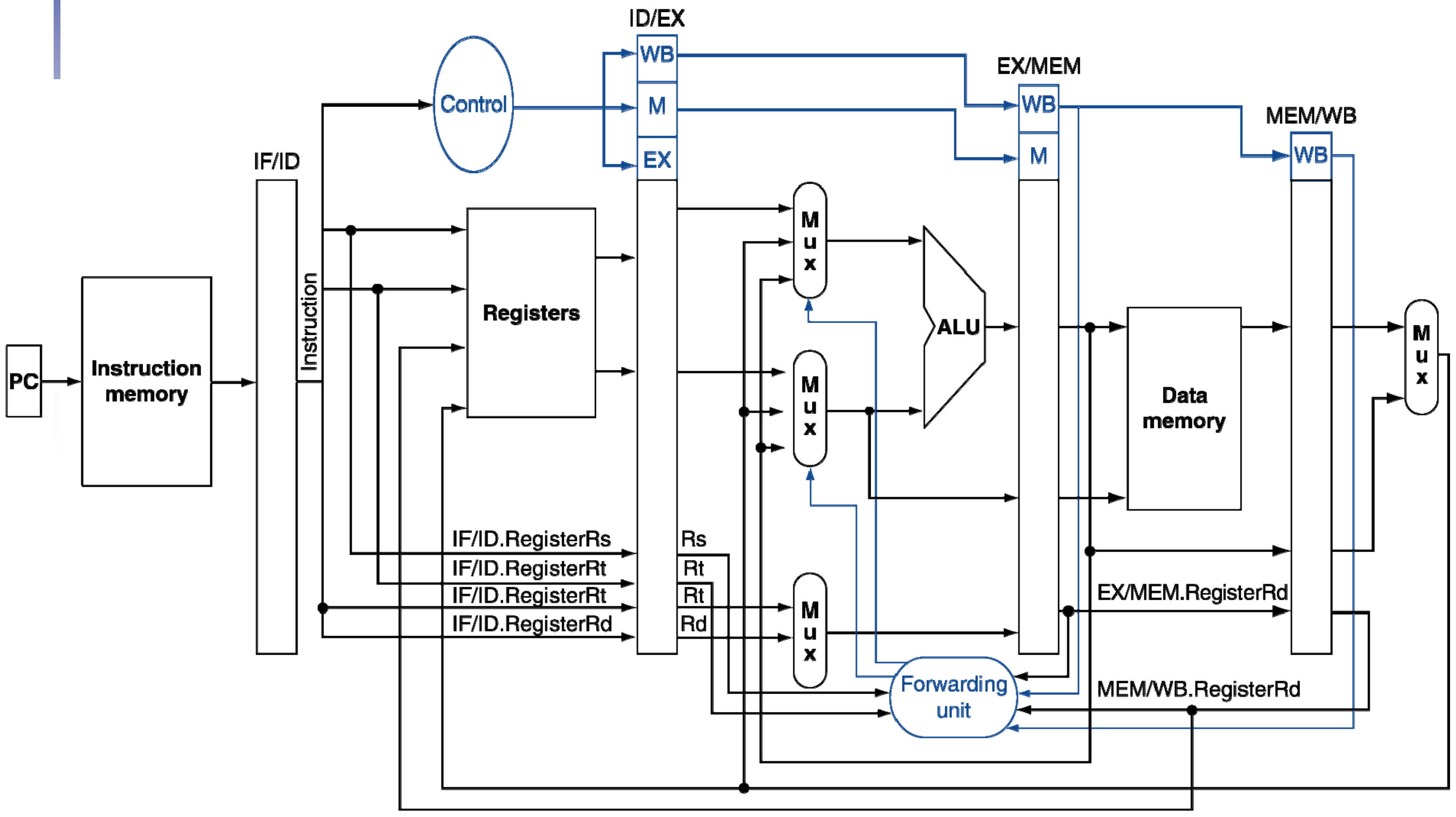
Revised Forwarding Condition

- 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

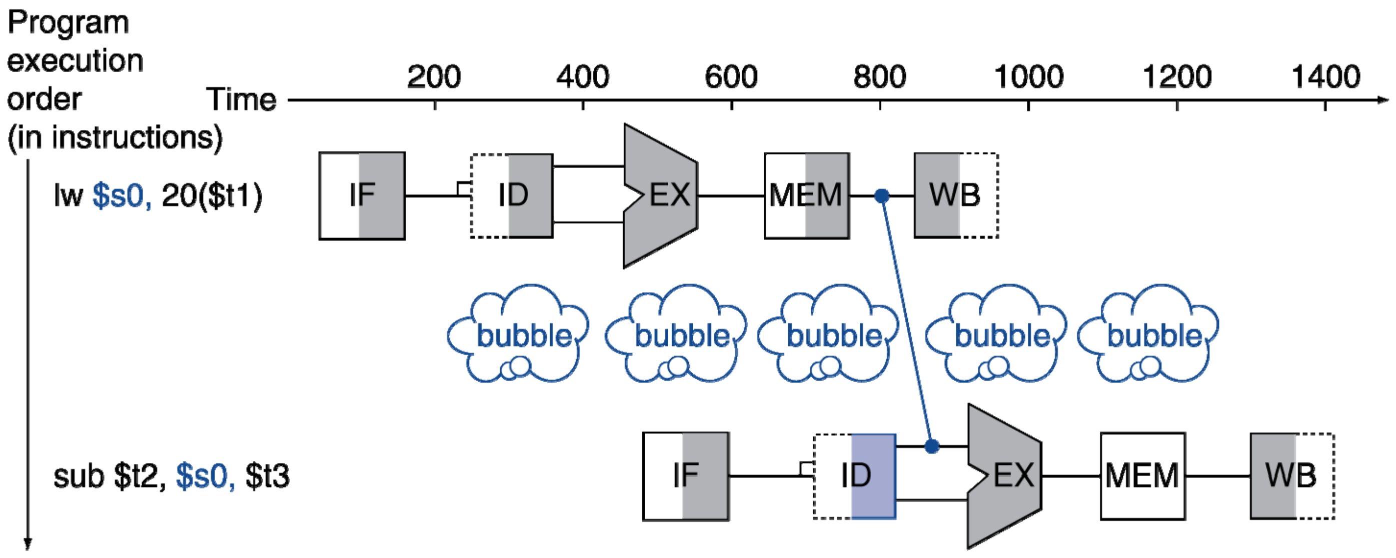


Chapter 4

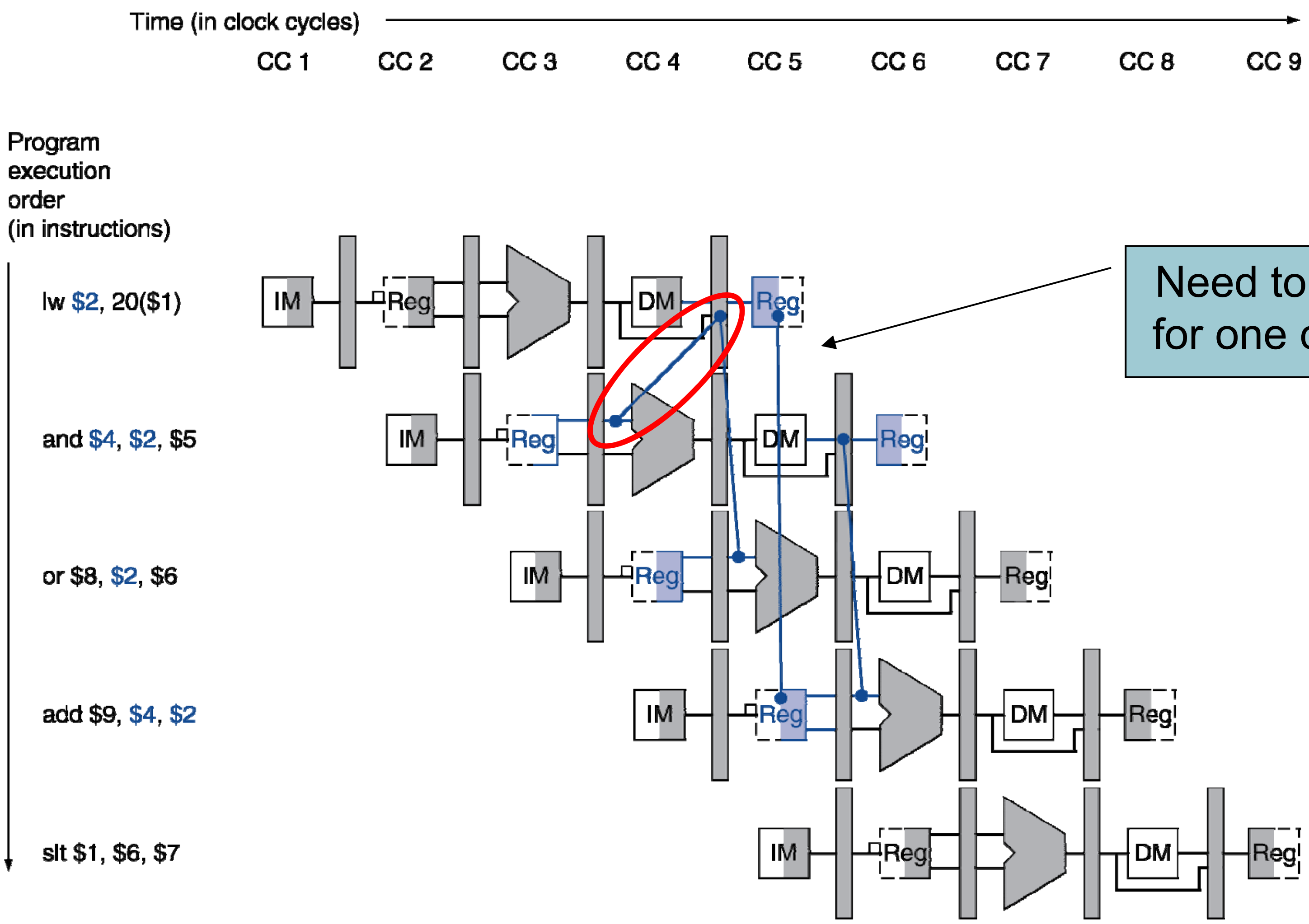
The Processor

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



Need to stall for one cycle

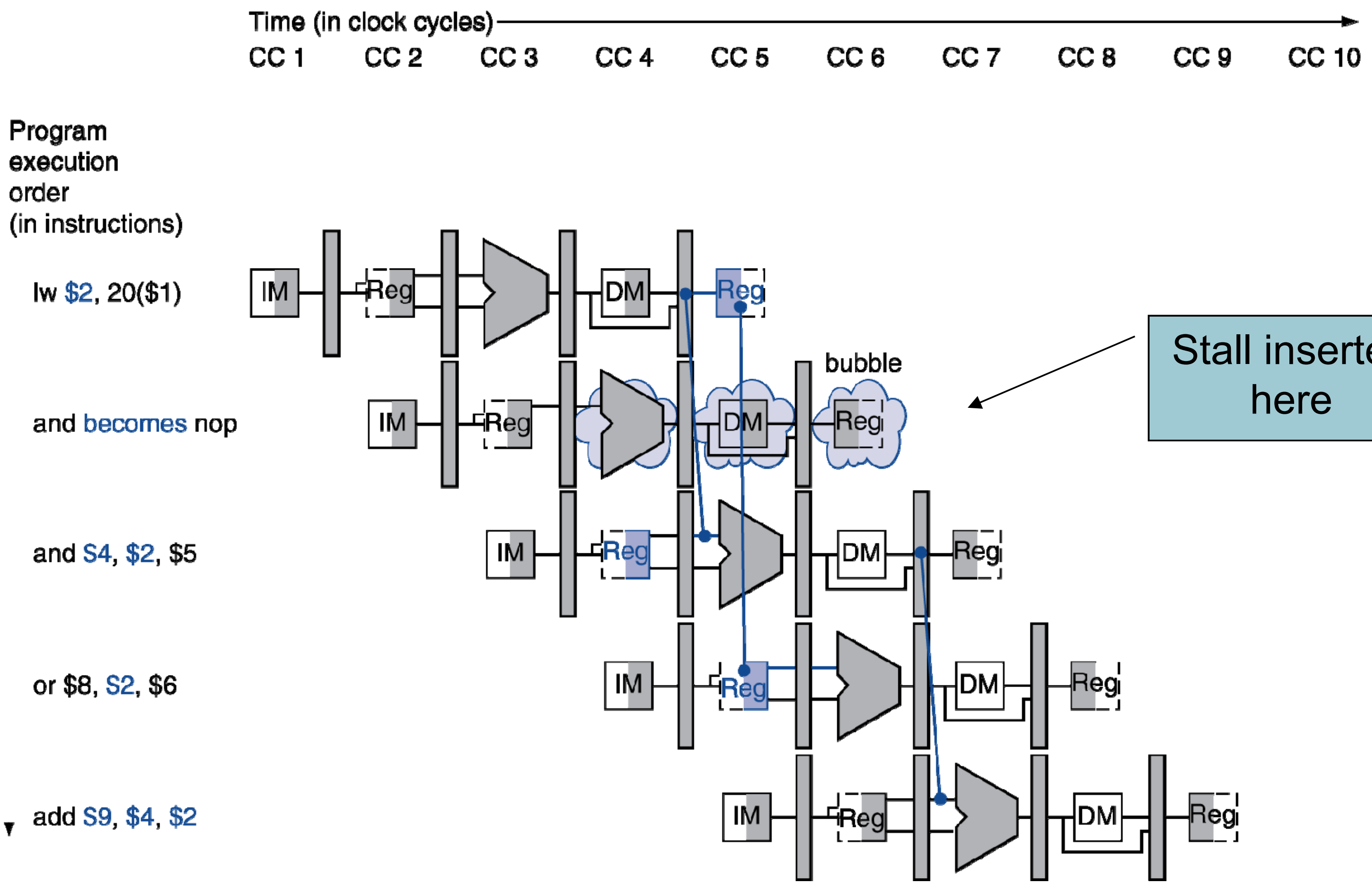
Load-Use Hazard Detection

- 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

How to Stall the Pipeline

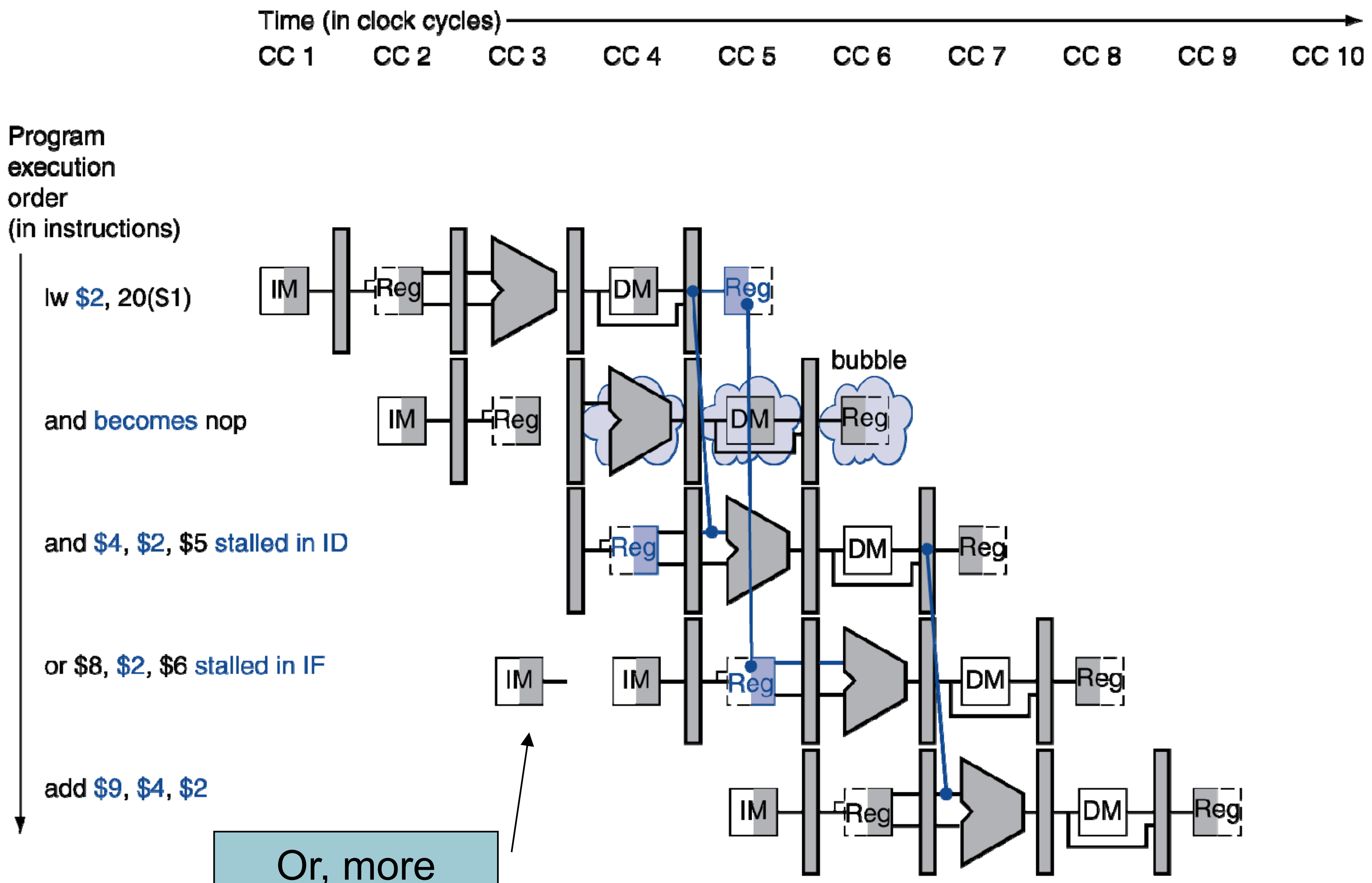
- Force control values in ID/EX register to 0
  - EX, MEM and WB do `nop` (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `lw`
    - Can subsequently forward to EX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

Stall/Bubble in the Pipeline



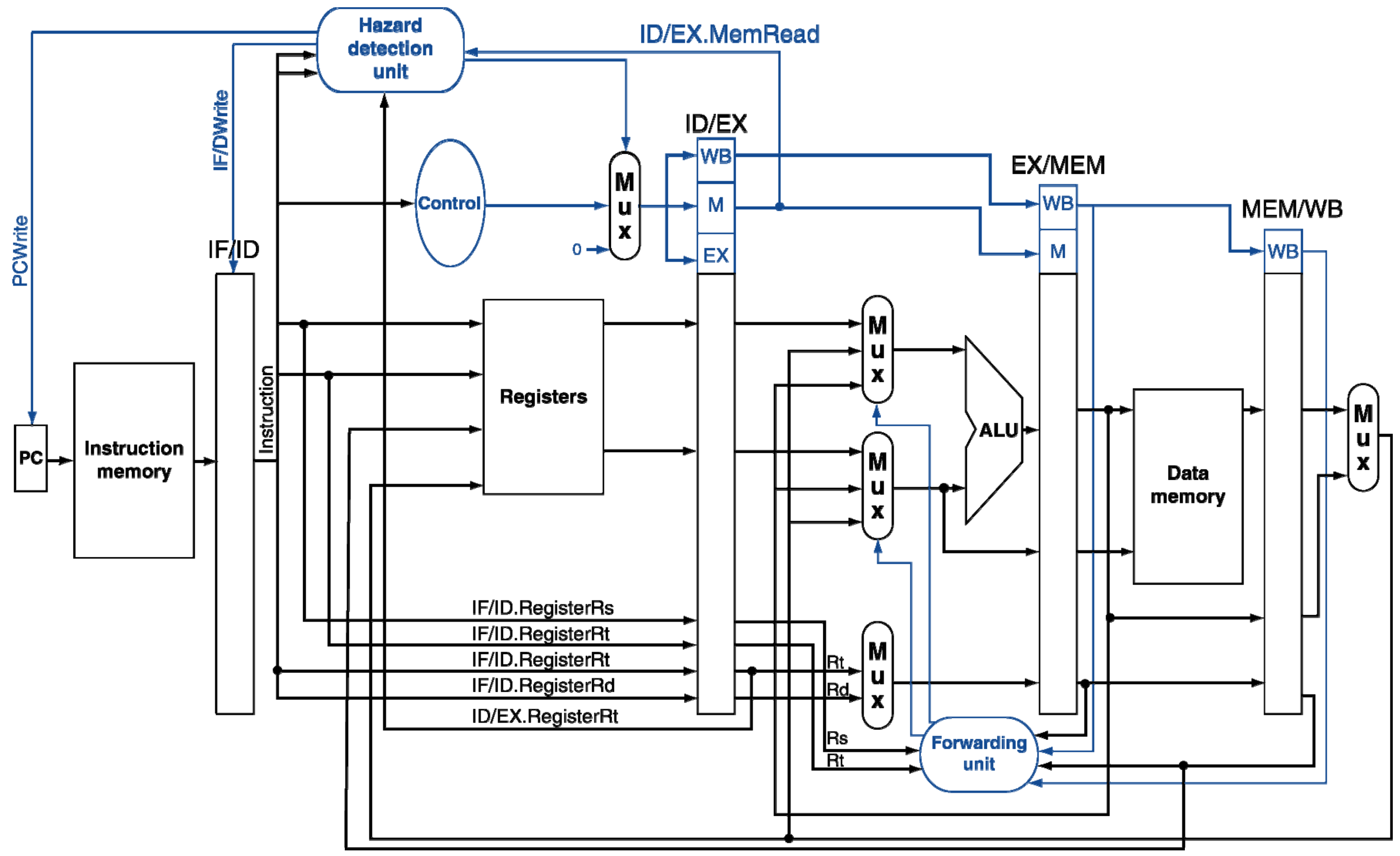
Stall inserted here

Stall/Bubble in the Pipeline



Or, more accurately…

Datapath with Hazard Detection


Hazard detection unit
ID/EX.MemRead
PCWrite
IF/DWrite
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Control
Mux
0
Registers
PC
Instruction memory
Instruction
ALU
Data memory
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
ID/EX.RegisterRt
Rt
Rd
Rs
Rt
Forwarding unit