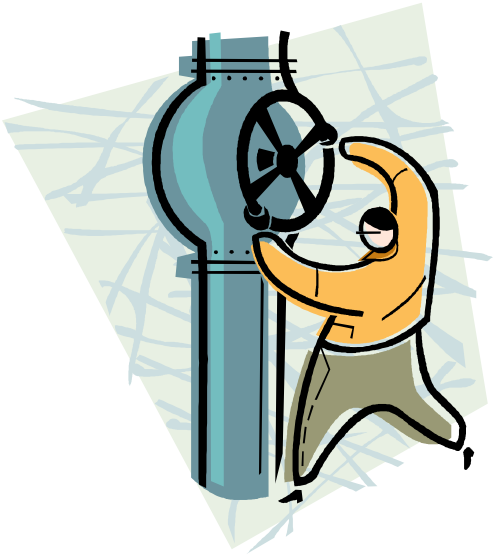
Pipeline Data Hazards



Warning, warning, warning!

Data Hazards

- Data Hazards are caused by data dependences.
- Not all data dependences result in data hazards
- A data hazard results when there is a data dependence between two instructions that appear too close together in the pipeline.

- More specifically, we will define a data hazard as any data dependence that requires either the software or hardware to take special action to get correct.

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

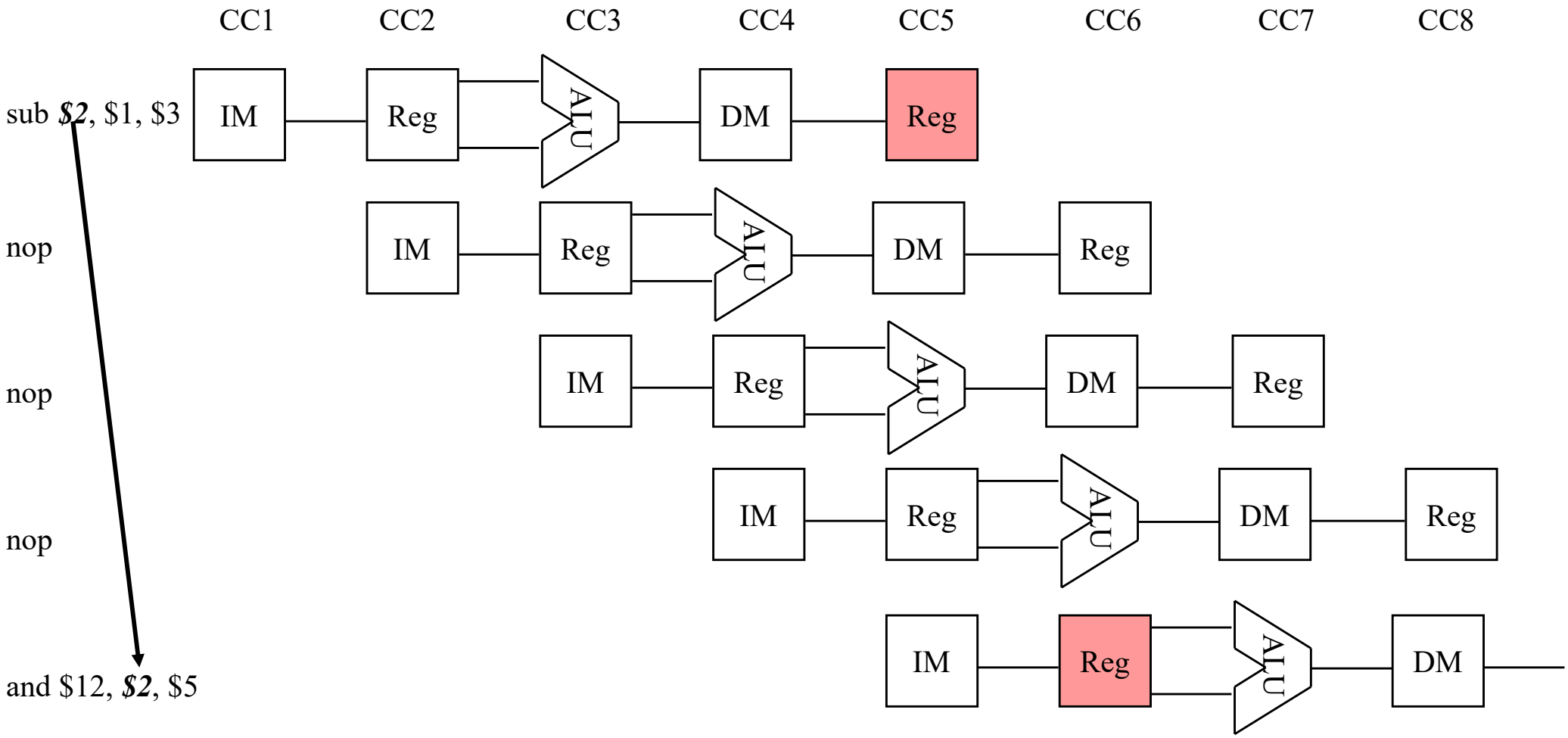
Dealing With Data Hazards

- In Software
  - –
- In Hardware
  - –
  - –

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```
subi  $5, $4, #45
add   $8, $5, $2
```

Dealing with Data Hazards in Software



How Many No-ops?

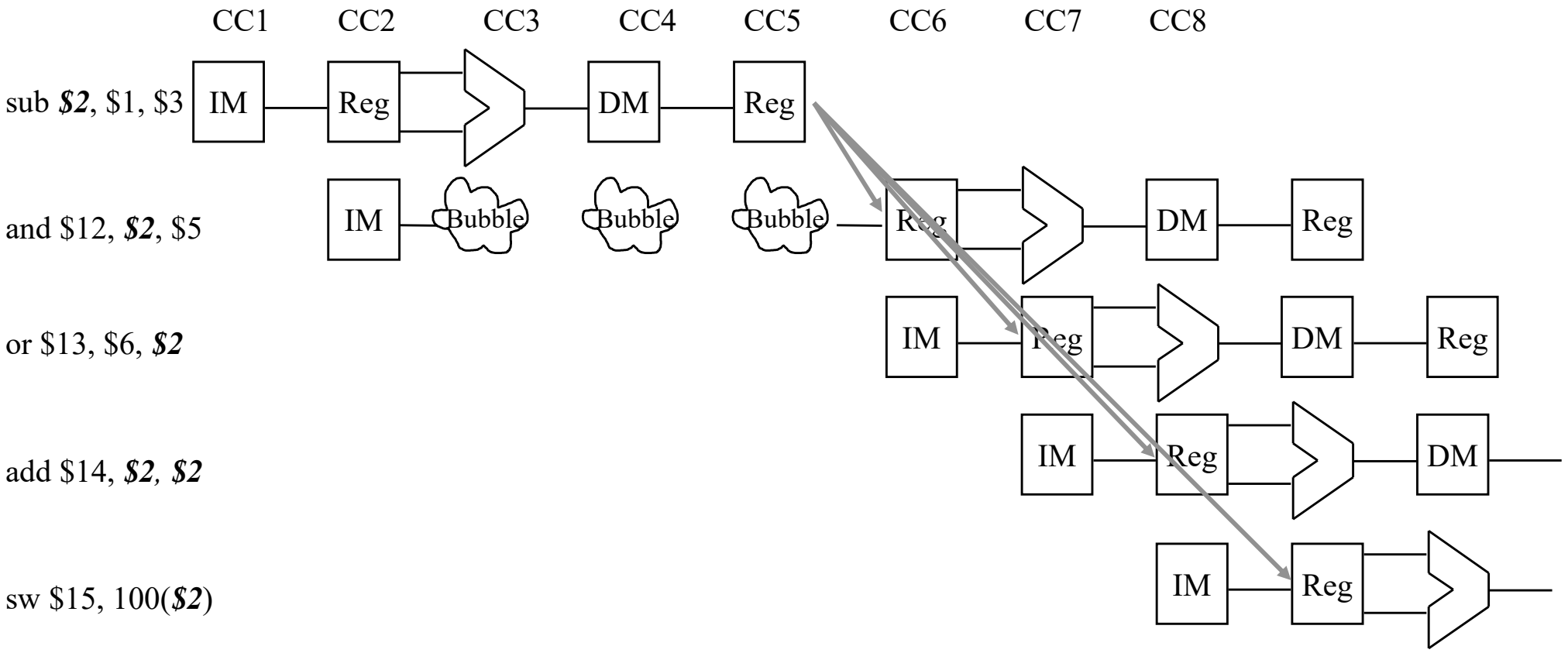
```
sub $2, $1,$3
and $4, $2,$5
or   $8, $2,$6
add $9, $4,$2
slt   $1, $6,$7
```

Are No-ops Really Necessary?

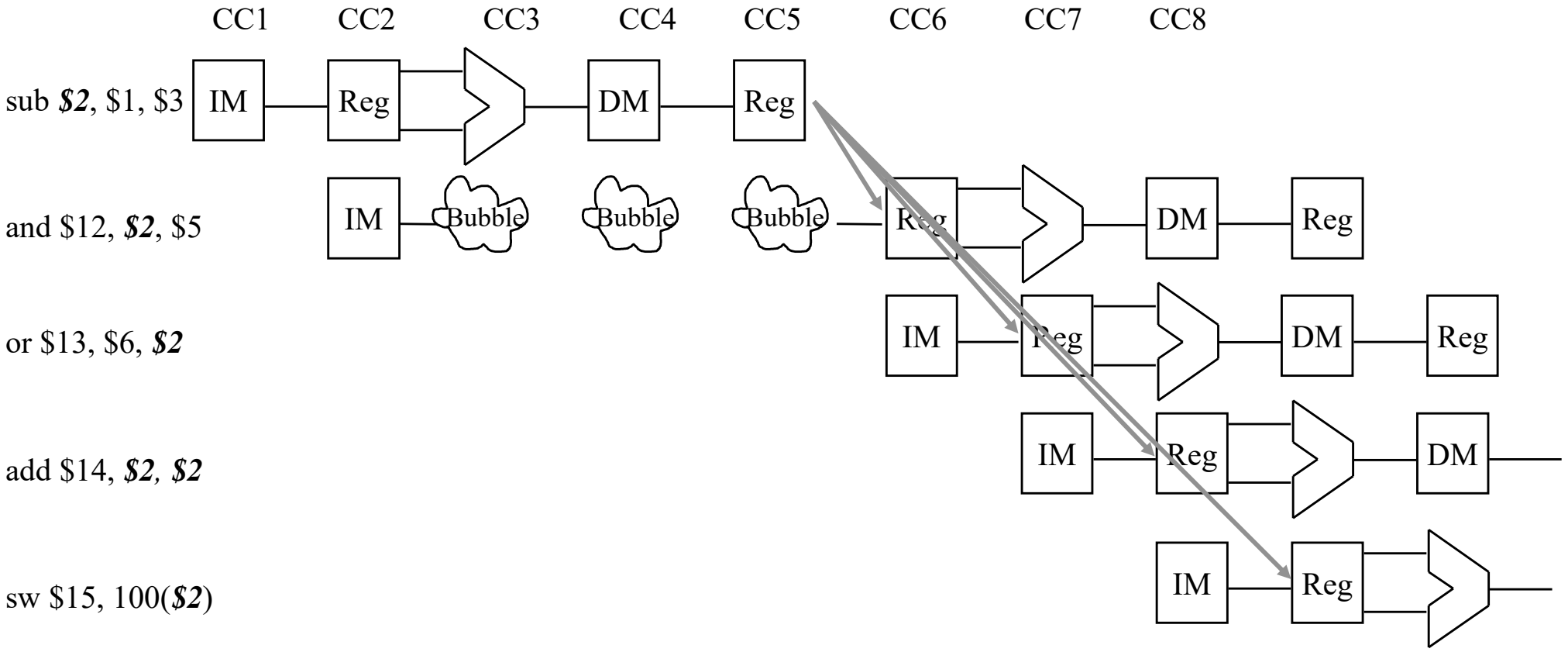
```
sub $2, $1,$3
and $4, $2,$5
or  $8, $3,$6
add $9, $2,$8
slt  $1, $6,$7
```

"Microprocessor without Interlocking Pipeline Stages"

Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls

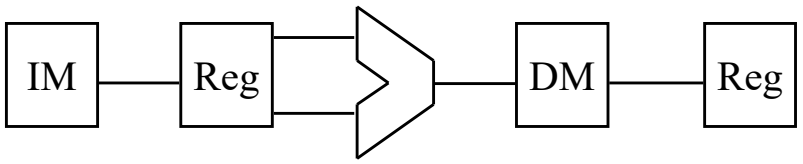


Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls (alt. View)



Pipeline Stalls

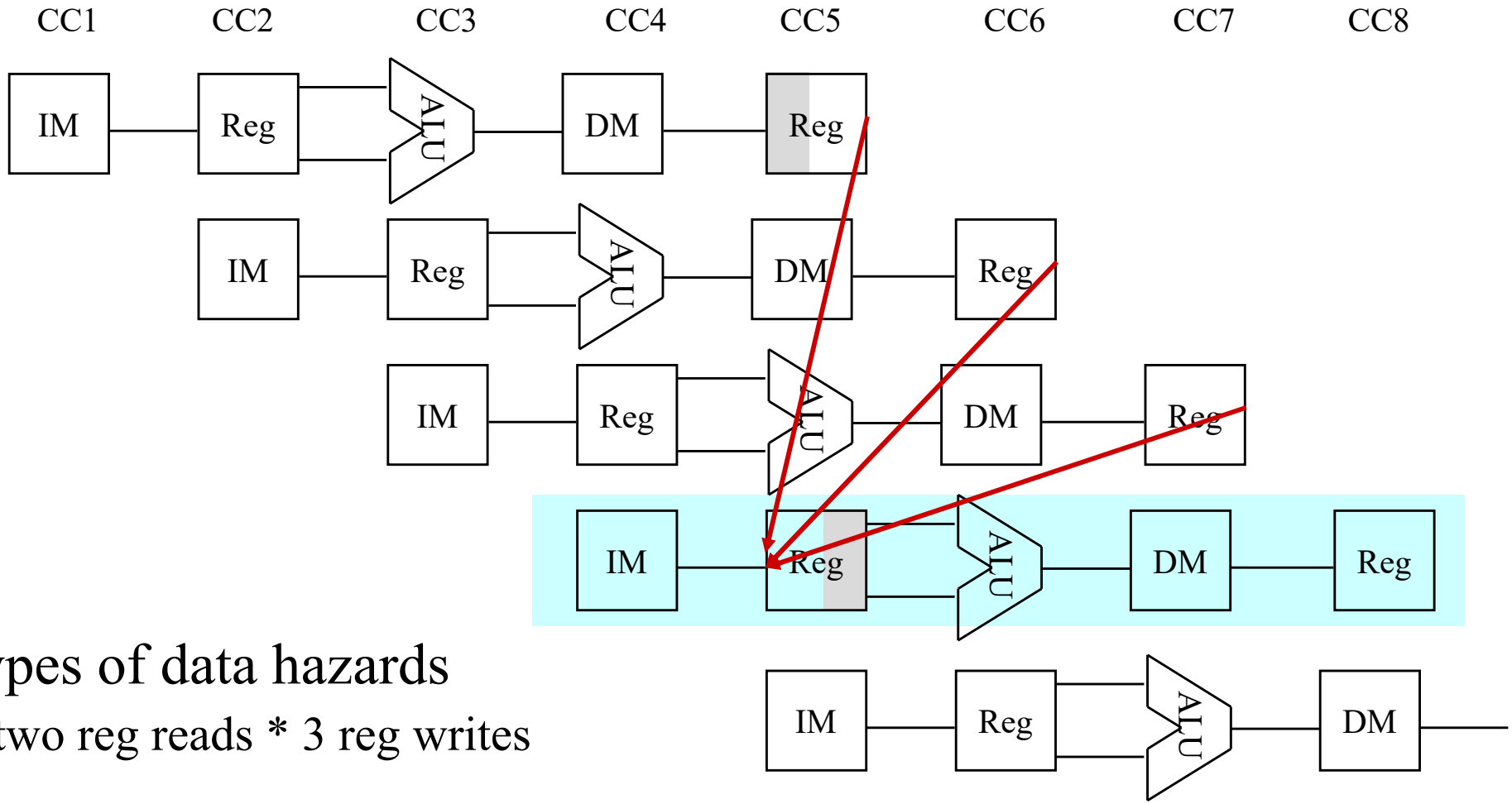
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
sub $2, $1, $3	IF	ID	EX	M	WB			
add $12, $3, $5								
or $13, $6, $2								
add $14, $12, $2								
sw $14, 100($2)								



Pipeline Stalls

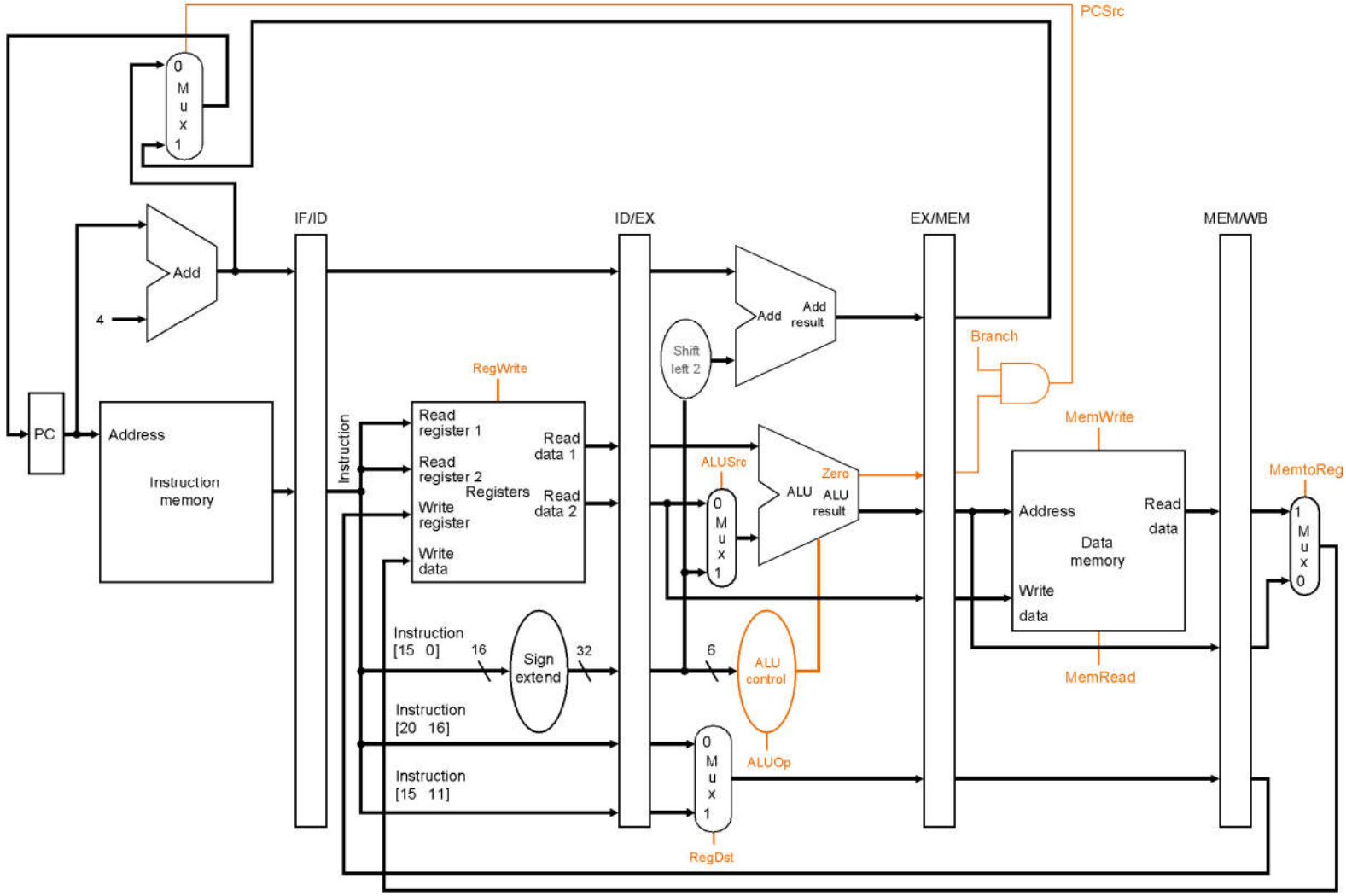
- To insure proper pipeline execution in light of register dependences, we must:
  - detect the hazard
  - stall the pipeline

Knowing When to Stall



- 6 types of data hazards
  - two reg reads * 3 reg writes

The Pipeline



- What comparisons tell us when to stall?

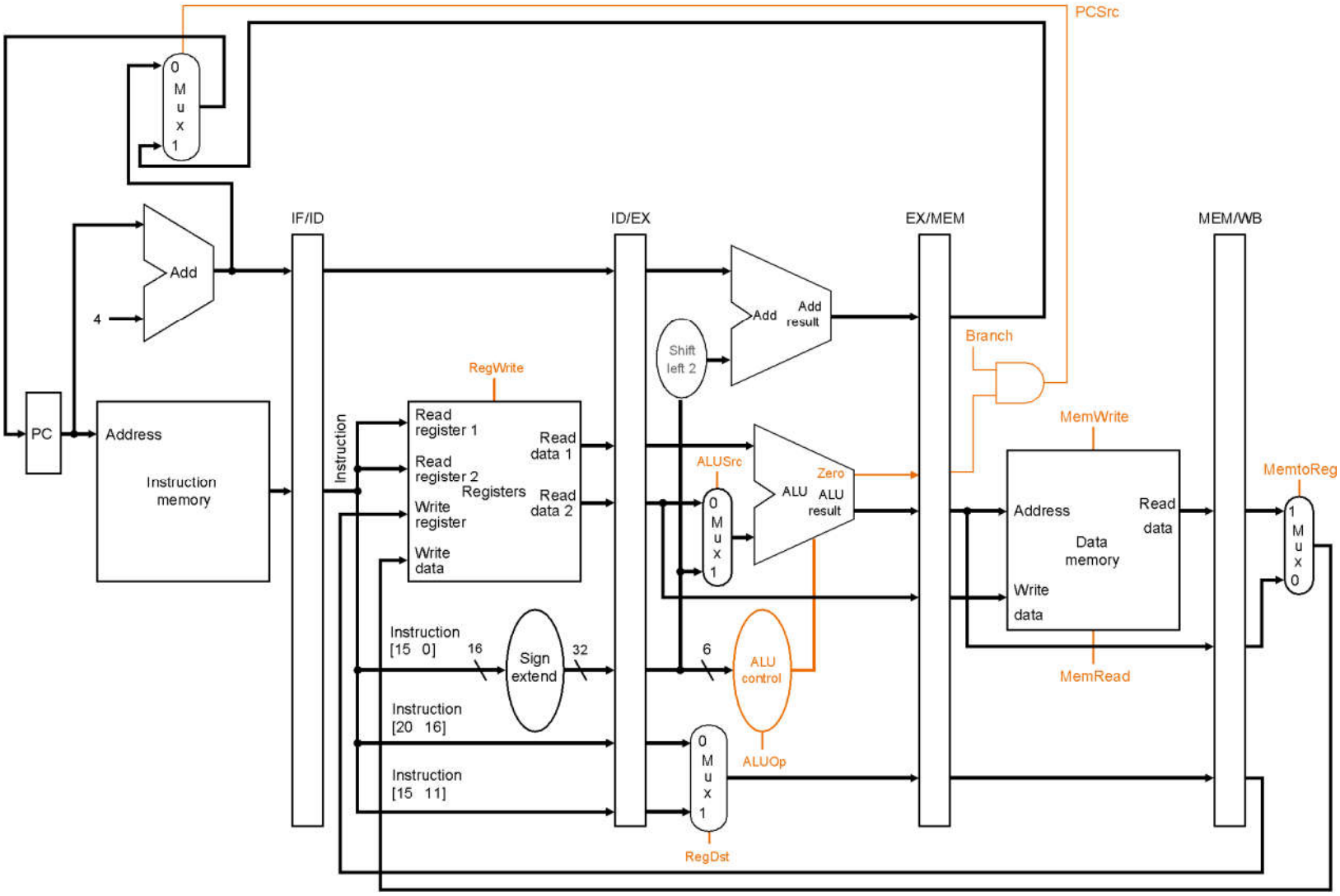
Stalling the Pipeline

- Once we detect a hazard, then we have to be able to stall the pipeline (insert a *bubble*).
- Stalling the pipeline is accomplished by
  - (1) preventing the IF and ID stages from making progress
    - the ID stage because it cannot proceed until the dependent instruction completes
    - the IF stage because we do not want to lose any instructions.
  - (2) essentially, inserting "nops" in hardware

Stalling the Pipeline

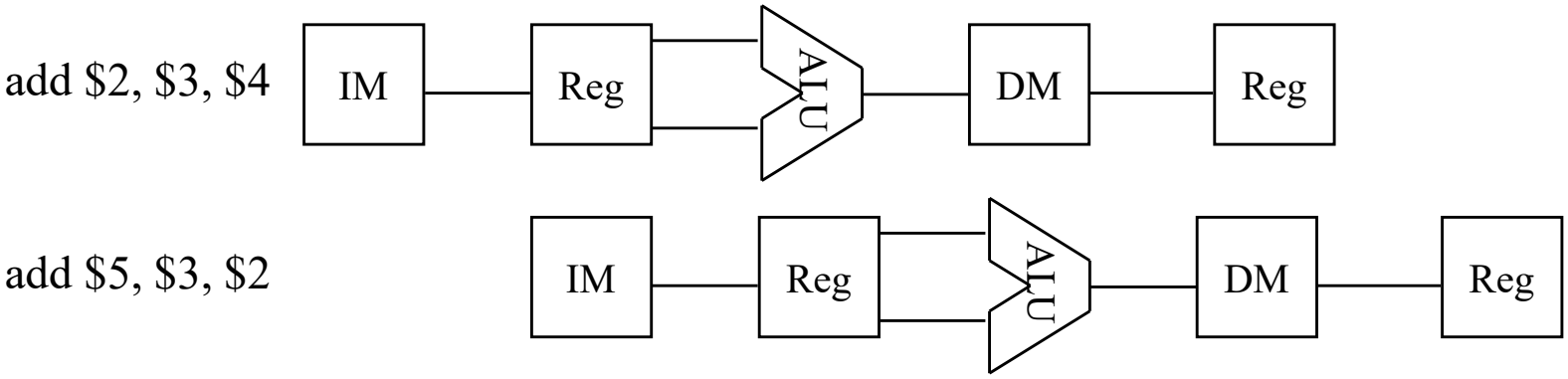
- Preventing the IF and ID stages from proceeding
  - don't write the PC (PCWrite = 0)
  - don't rewrite IF/ID register (IF/IDWrite = 0)
- Inserting "nops"
  - set all control signals propagating to EX/MEM/WB to zero

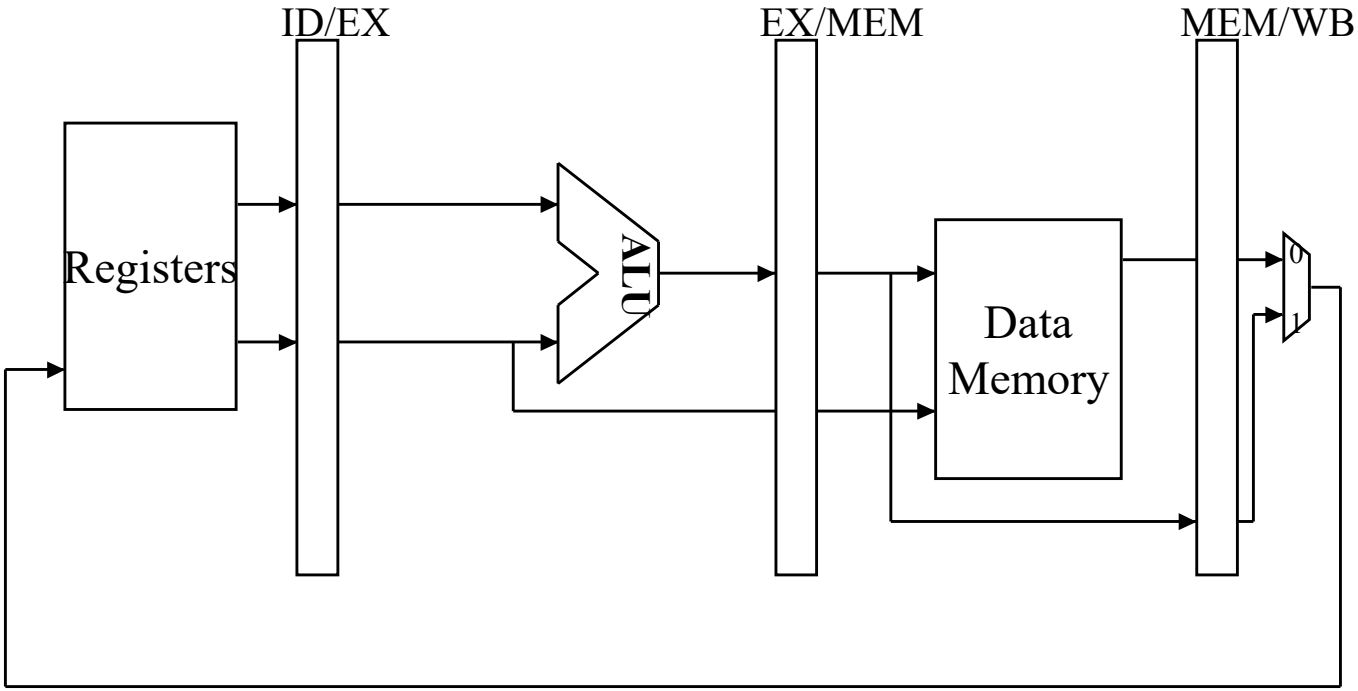
The Pipeline



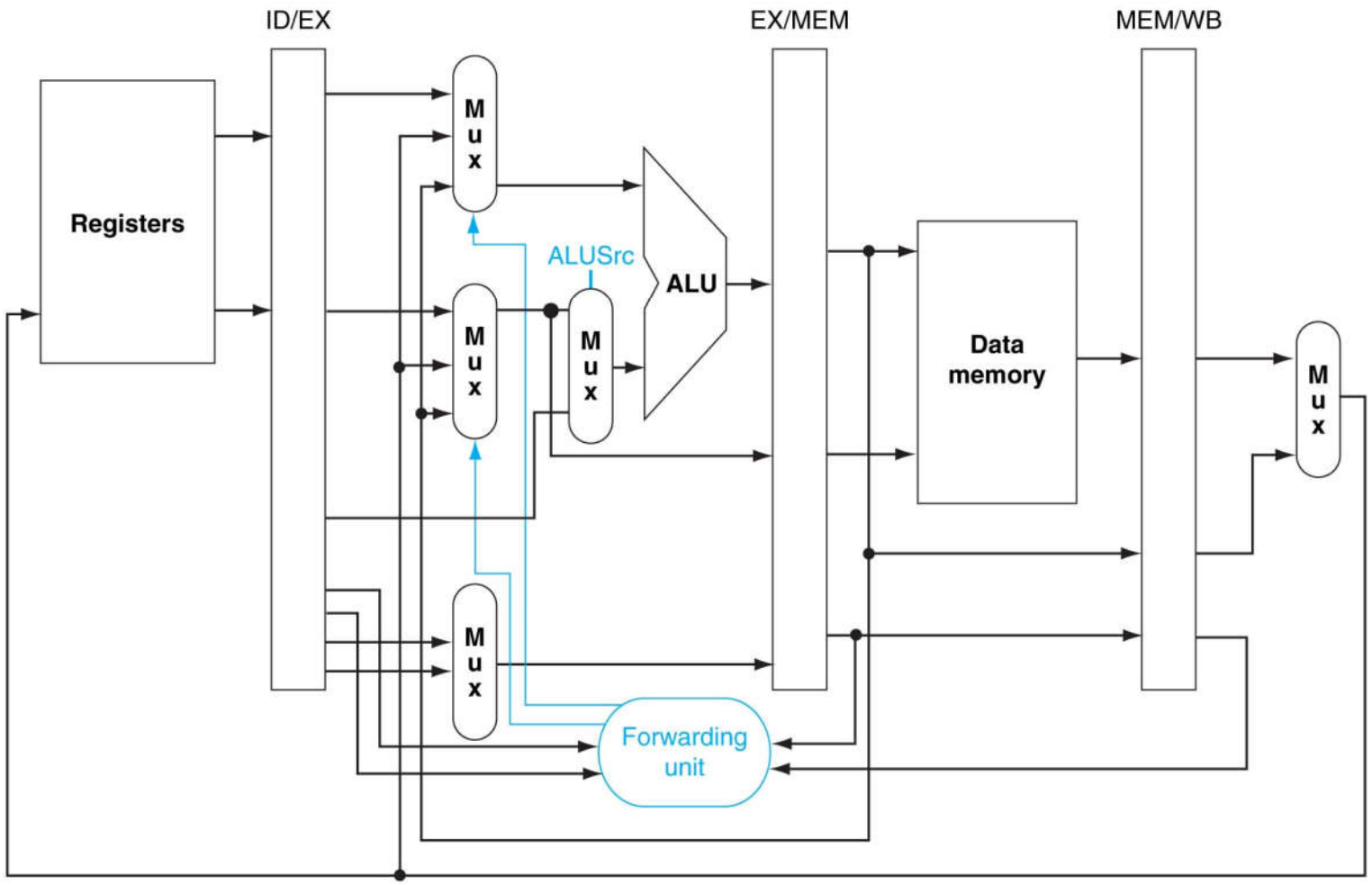
How else can we deal with data hazards?

Reducing Data Hazards Through Forwarding

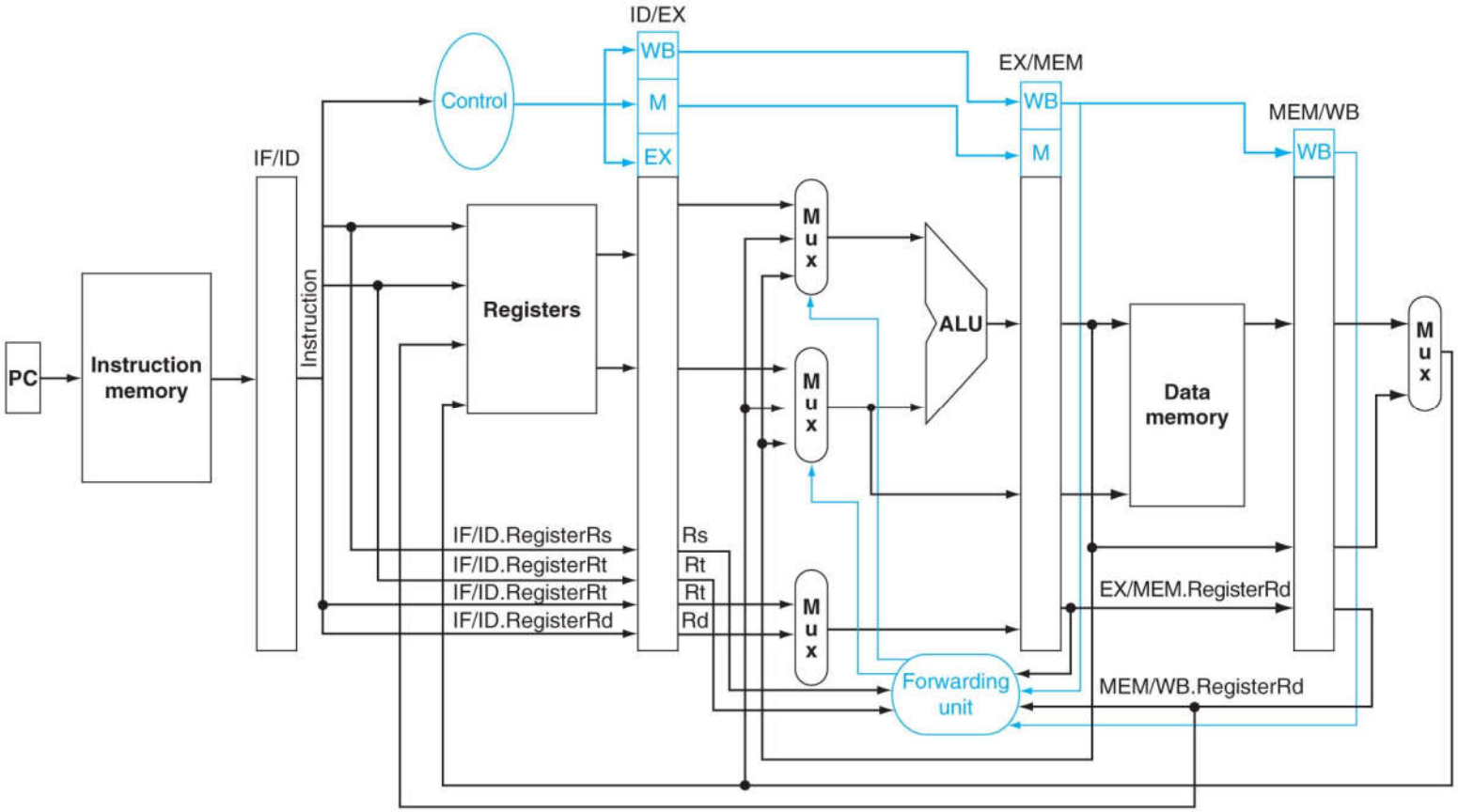




Reducing Data Hazards Through Forwarding



Reducing Data Hazards Through Forwarding



EX Hazard:

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

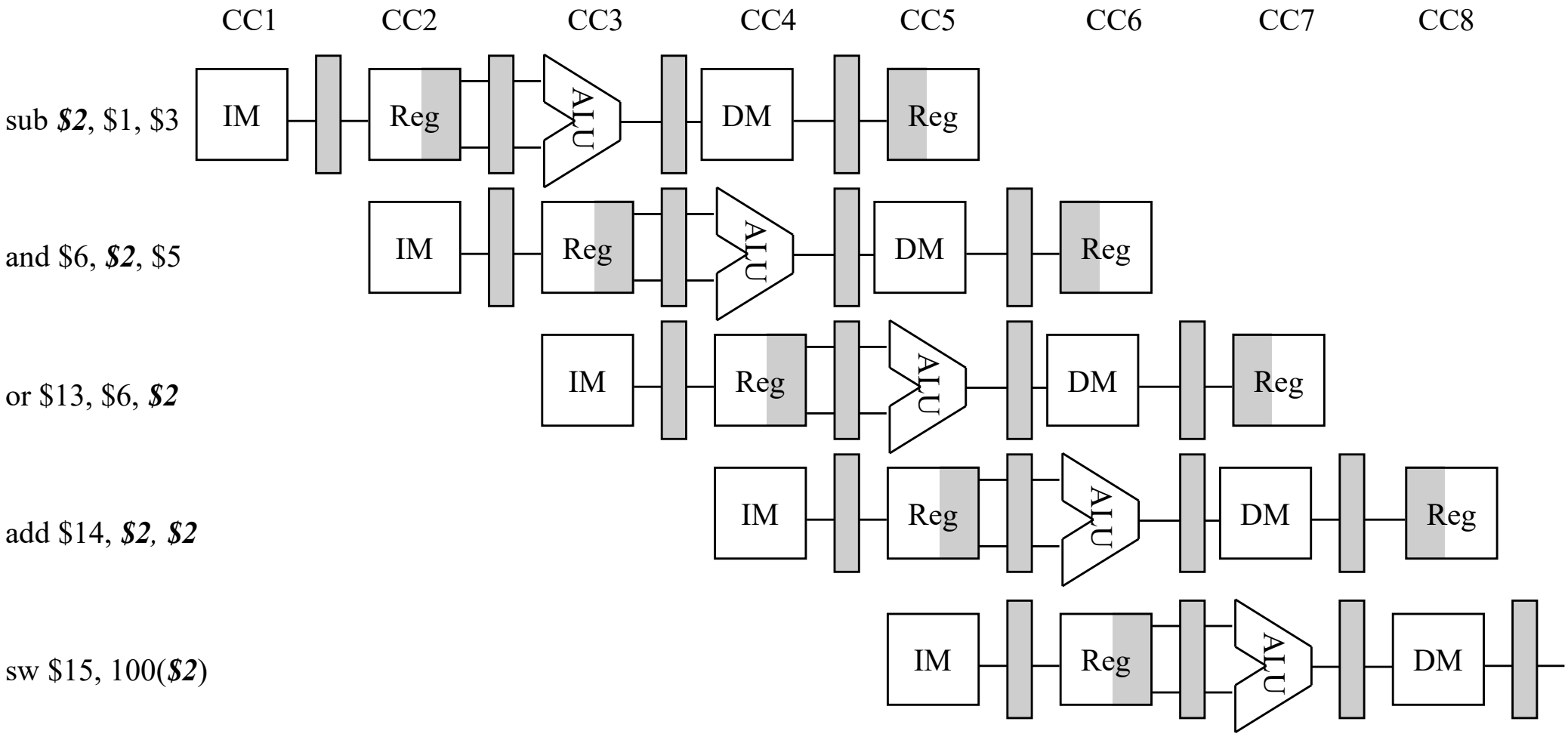
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10

(similar for the MEM stage)

Data Forwarding

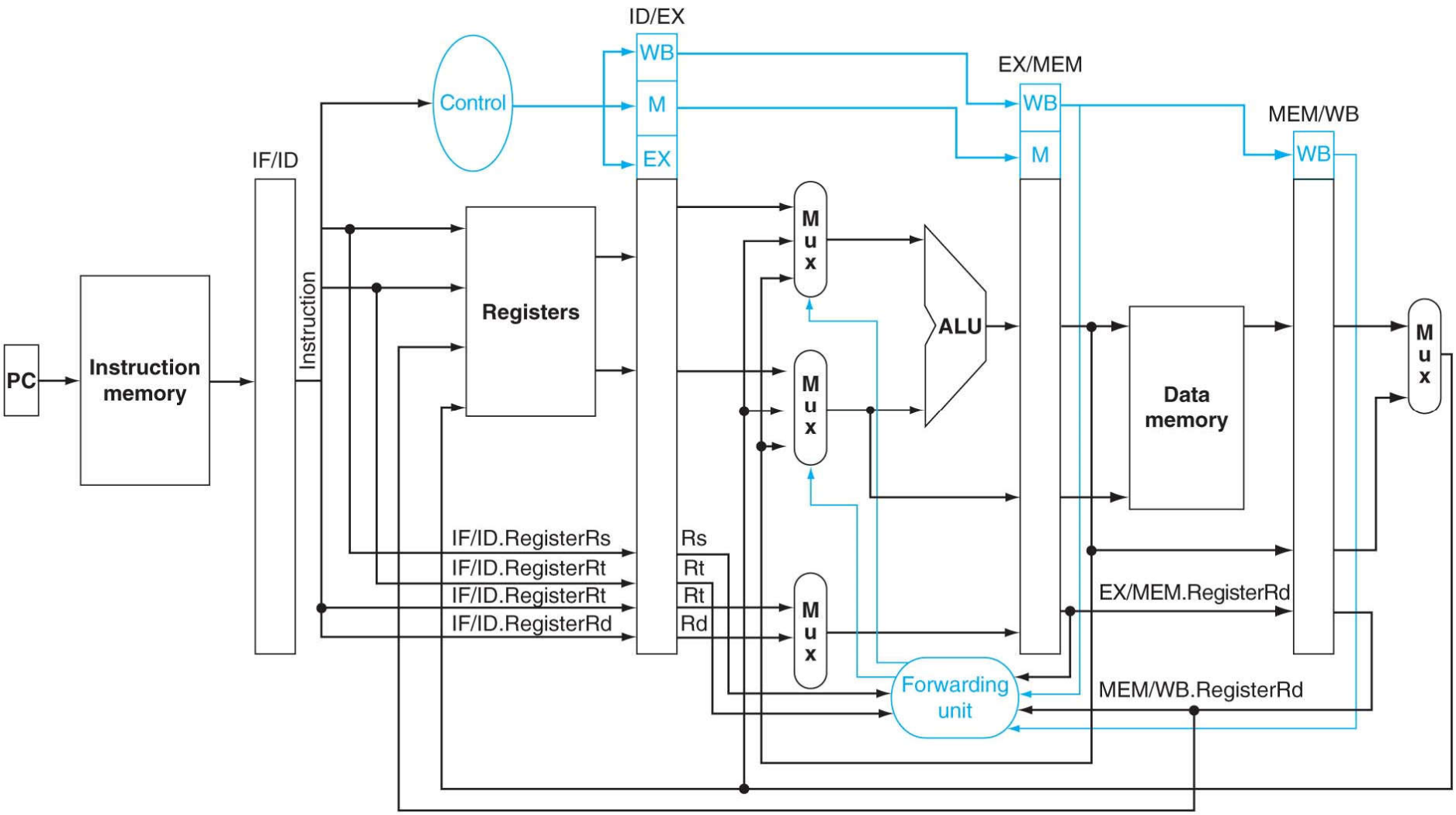
- The Previous Data Path handles two types of data hazards
  - EX hazard
  - MEM hazard
- We assume the register file handles the third (WB hazard)
  - if the register file is asked to read and write the same register in the same cycle, we assume that the reg file allows the write data to be forwarded to the output
  - We're still going to call that forwarding.

Eliminating Data Hazards via Forwarding



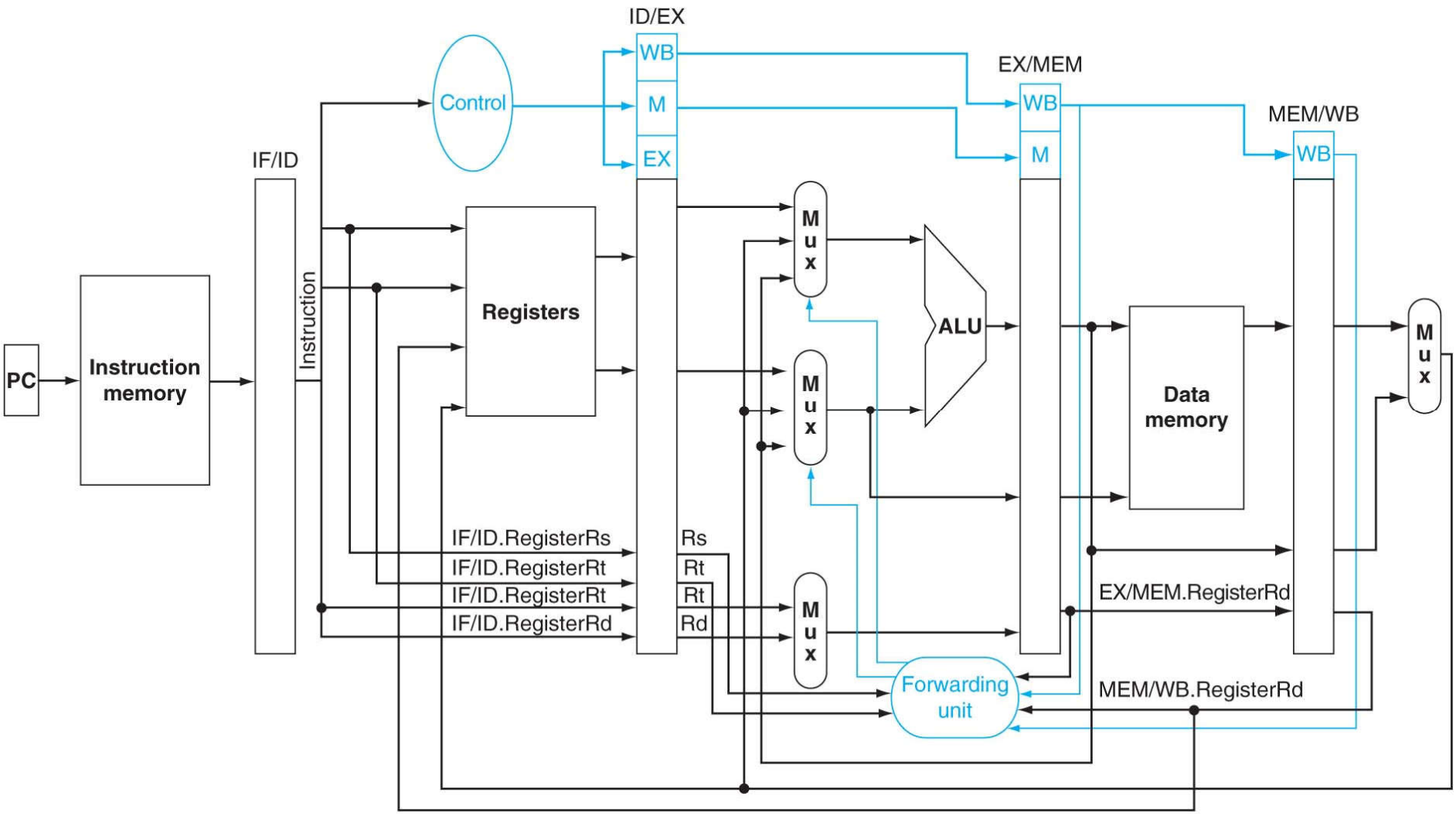
Forwarding in Action

add $1, $12, $3 **sub $12, $3, $4** **add $3, $10, $11** Memory Access Write Back



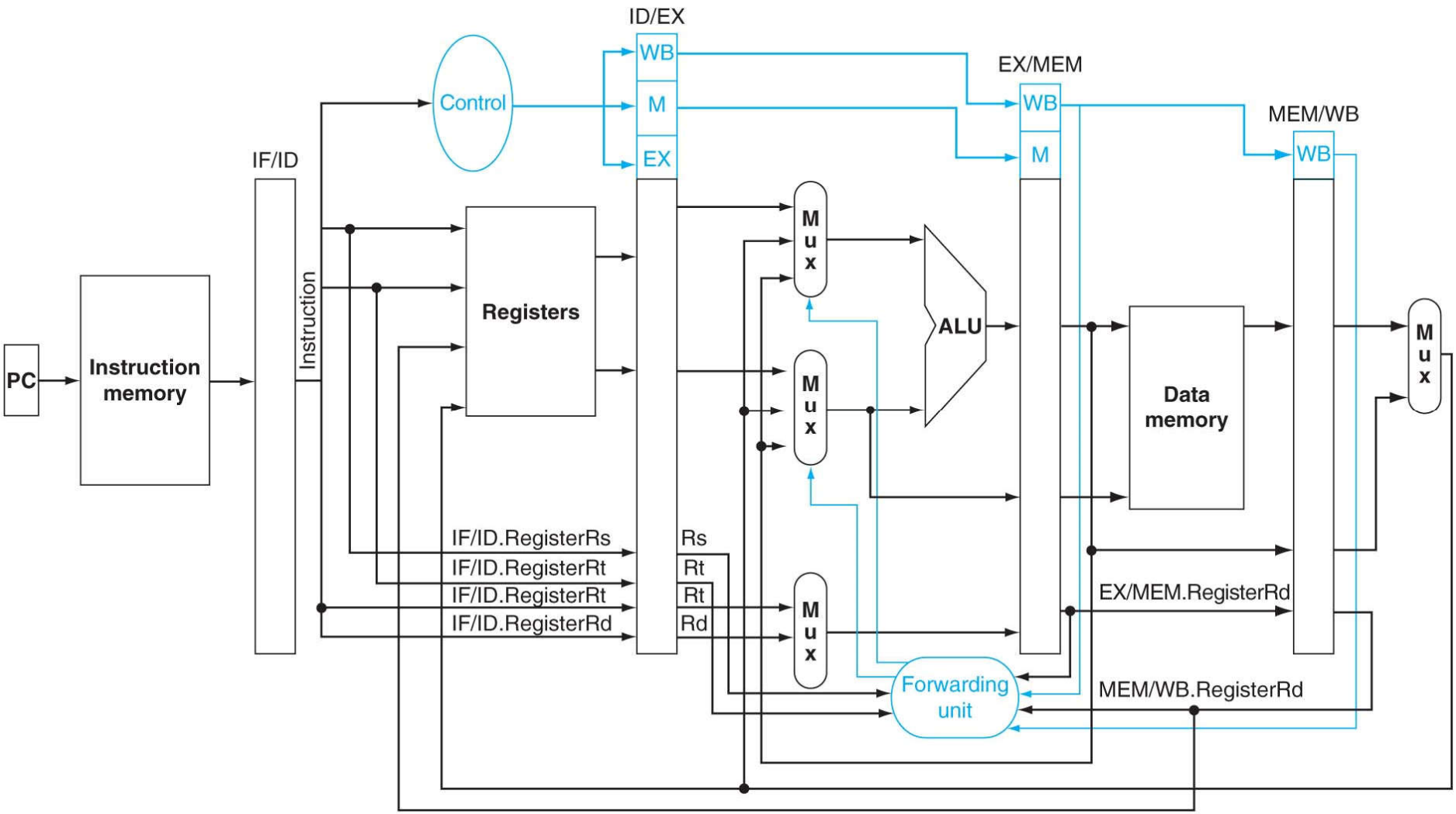
Forwarding in Action

Instruction Fetch **add $1, $12, $3** **sub $12, $3, $4** **add $3, $10, $11** Write Back

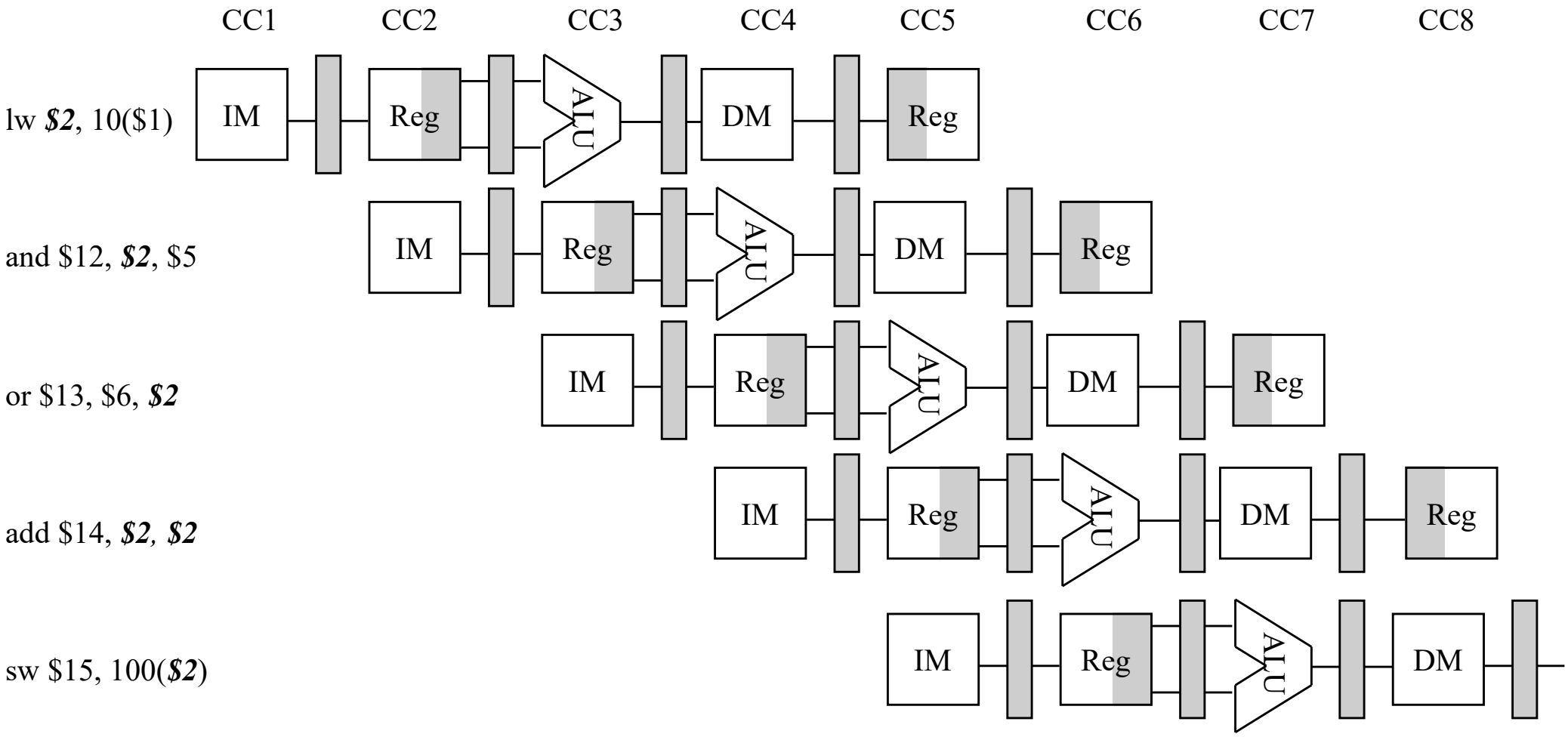


Forwarding in Action

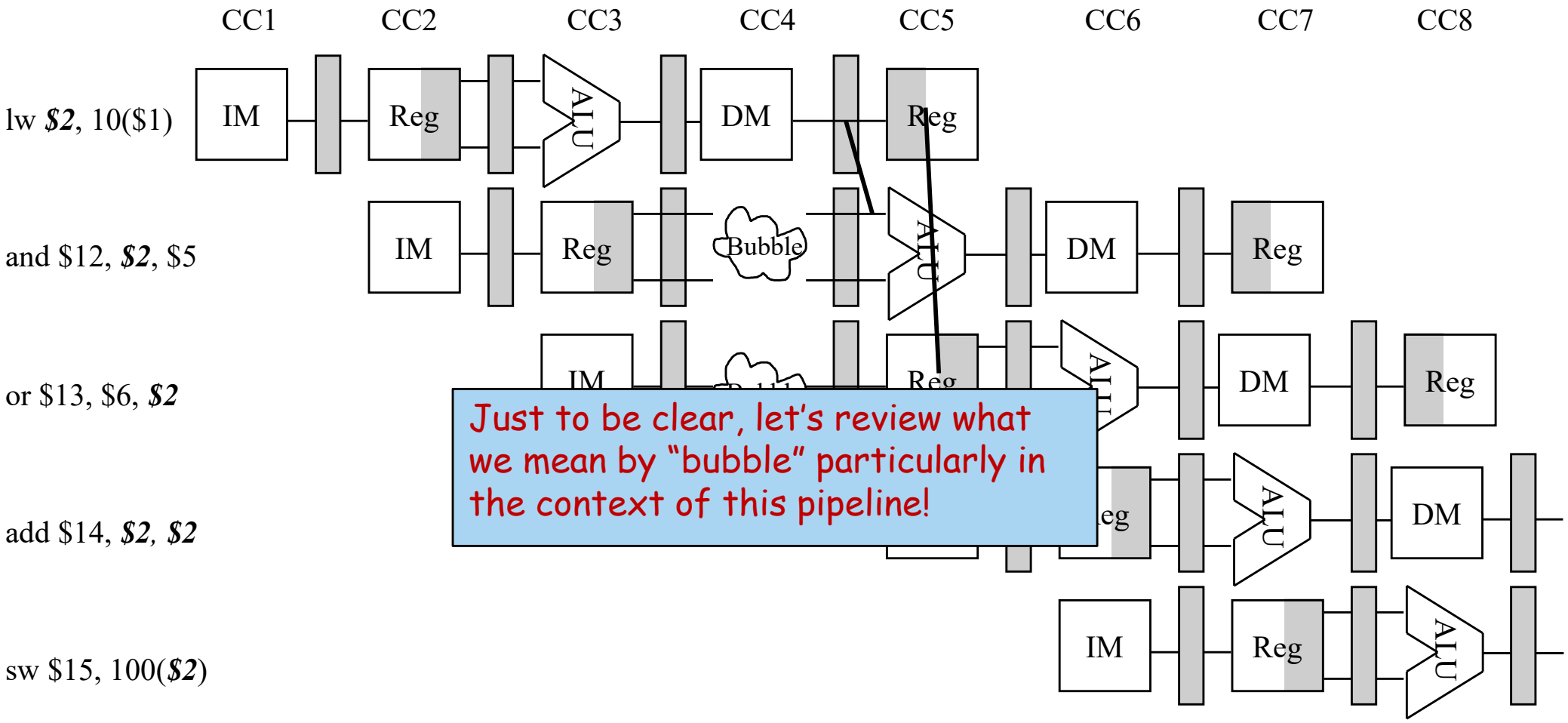
Instruction Fetch Instruction Decode **add $1, $12, $3** **sub $12, $3, $4** **add $3, $10, $11**



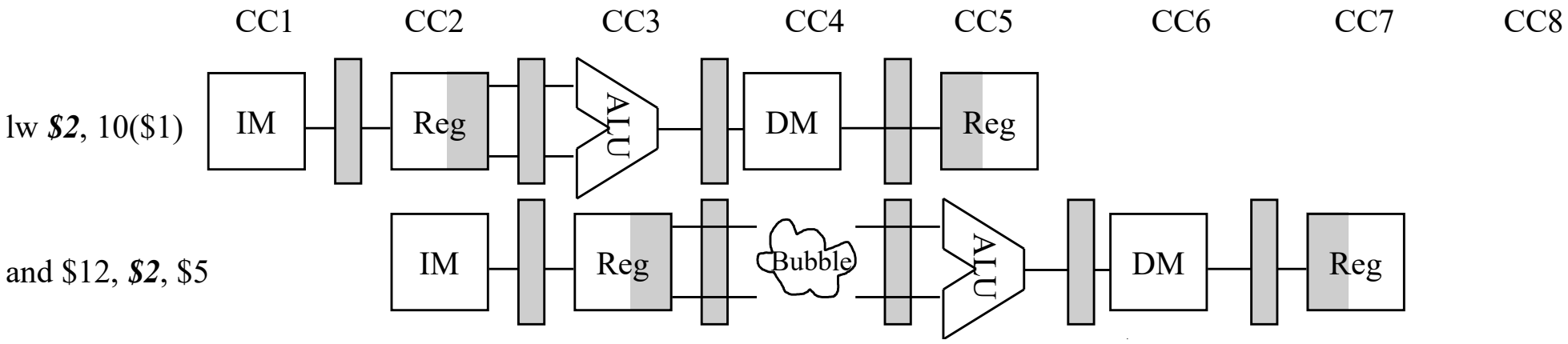
Eliminating Data Hazards via Forwarding??



Eliminating Data Hazards via Forwarding and stalling



Eliminating Data Hazards via Forwarding and stalling



What is really happening during the bubble (for this particular pipeline)?

- While ***lw*** moves to the Mem stage in CC4, the ***and*** instruction repeats the ID stage (important because the values the ***and*** reads in CC4 are the ones it will carry forward).
- There is now *no instruction* in the EX stage. So we better make sure that whatever is in the EX stage is safe.
  - Safe = no state changes (PC, reg, memory), now or as it moves through the pipeline.

Try this one...

Show bubbles and forwarding for this code

```
add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
add $9, $3, $6
```

Another one...

Show bubbles and forwarding for this code

```
lw $9, 100($6)     IF   ID   EX   M   WB
addi $6, $9, #26
sub $7, $6, $9
add $6, $3, $6
add $3, $2, $6
```

Suppose EX is the longest (in time) pipeline stage. To reduce CT, we split it in half. Given the following pipeline:

IF ID EX1 EX2 M WB

Assume the input data must be available at the start of EX1 and the output is available after EX2. How many hardware stalls would be required in the following code (assuming hardware forwarding wherever possible)?

```
add r1, r2, r3
add r4, r1, r3
```

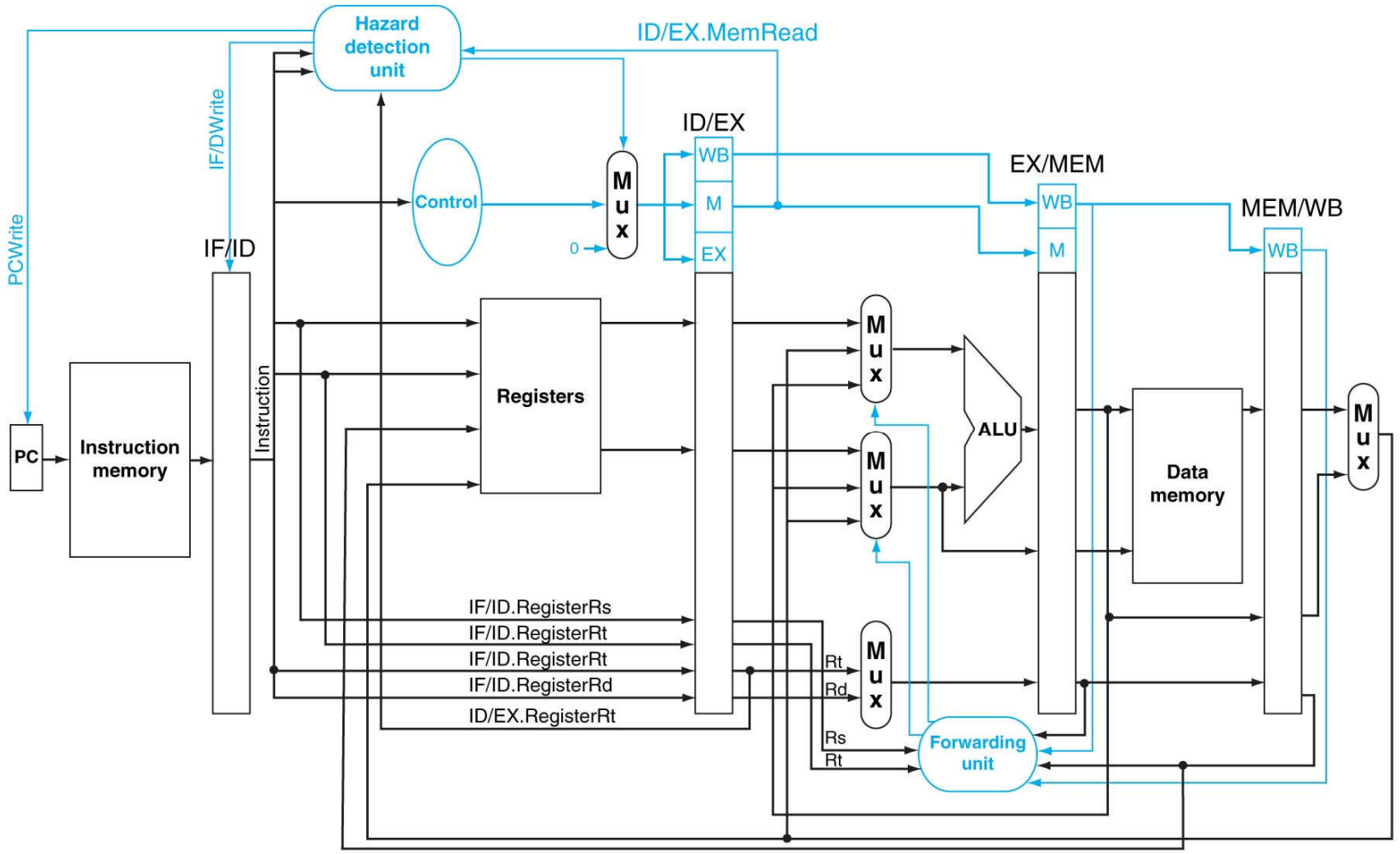
Suppose EX is the longest (in time) pipeline stage. To reduce CT, we split it in half. Given the following pipeline:

IF ID EX1 EX2 M WB

Assume the input data must be available at the start of EX1 and the output is available after EX2.How many hardware stalls would be required in the following code (assuming hardware forwarding wherever possible)?

```
lw r1, 0(r3)
add r2, r1, r3
```

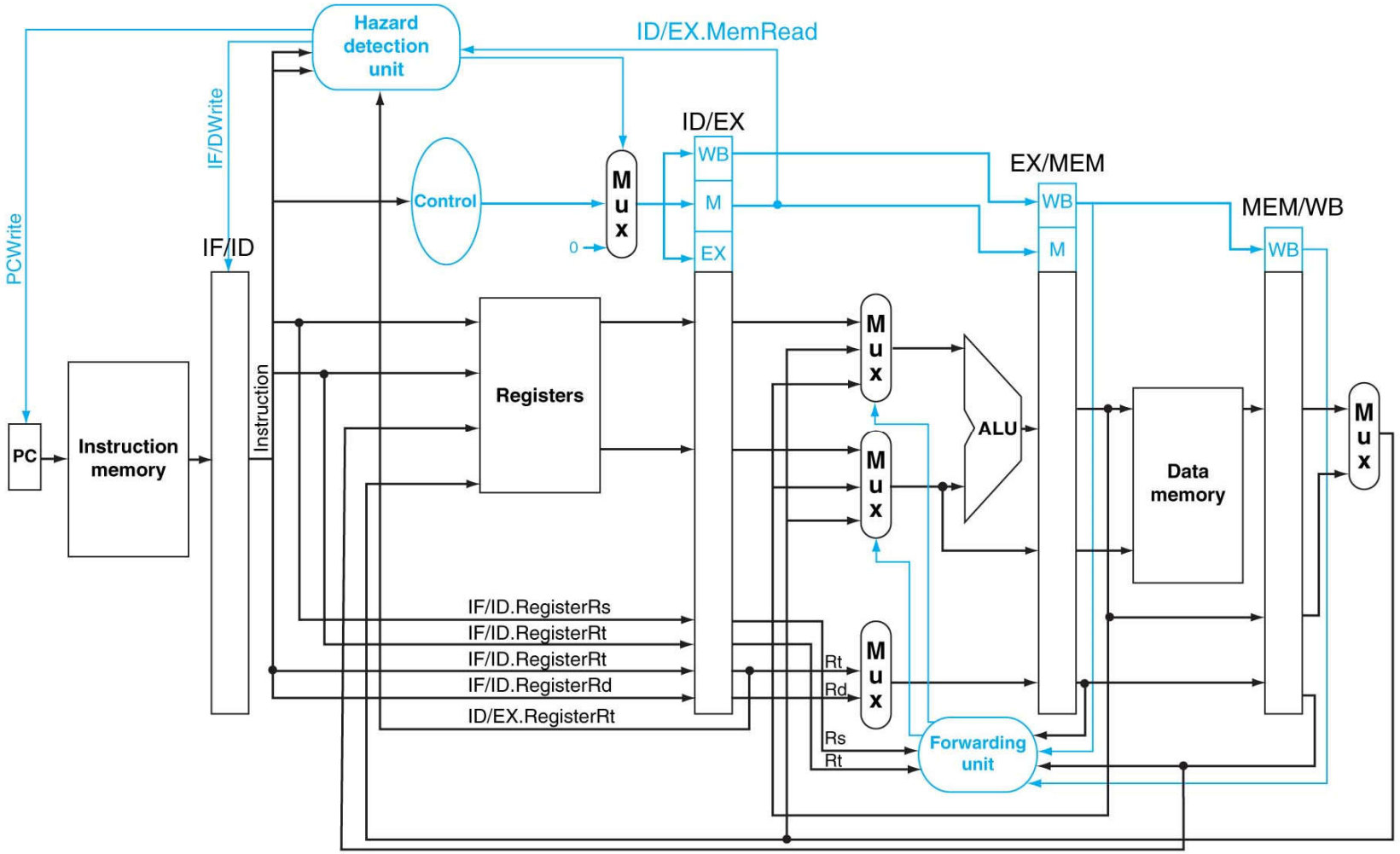
Datapath with Hazard-Detection



if (ID/EX.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)))
then stall the pipeline

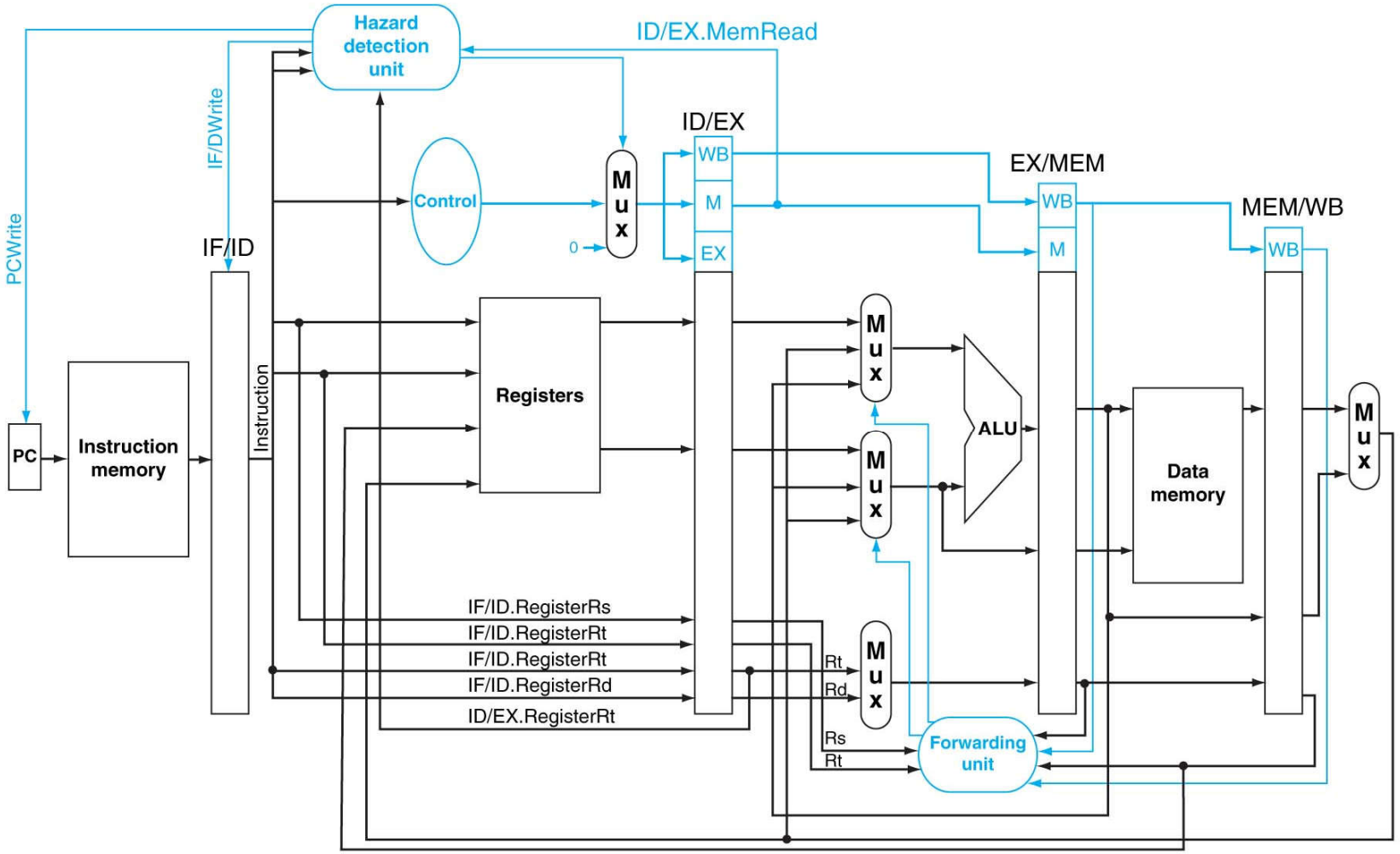
Hazard Detection

and $4, $2, $5 ***lw $2, 20($1)***



Hazard Detection

and $4, $2, $5 ***(bubble)*** ***lw $2, 20($1)***



Data Hazard Key Points

- Pipelining provides high throughput, but does not handle data dependences easily.
- Data dependences cause *data hazards*.
- Data hazards can be solved by:
  - software (nops)
  - hardware stalling
  - hardware forwarding
- Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
- ET = IC * CPI * CT