Pipelining The Datapath

Part 2

Trouble in the Pipeline

- Structural Hazards:
  - MULTIPLE INSTRUCTIONS ARE TRYING TO USE THE SAME HARDWARE
  - Memory was a Structural hazard: Multi-cycle Datapath
    - Solution: Split up instruction memory and data memory
    - What Else was a possible structural Hazard?
- Data Hazards:
  - Some data computation needed for a subsequent instruction: **lw $2 100($1), add $1, $2, $8**
  - Or **add $1, $2, $3** followed by **sub $4, $1, $2**
- Control Hazards:
  - Flow of Control Change: PC updated in beq or jump instruction, when do we start next instruction?

Structural Hazard:
Is Register File Access a Structural hazard

- A) Yes
- B) No
- C) Yes but we designed this out. Allowed Write and Read to Register File in one clock cycle.
- D) Yes, and we solved this by allow Reads to occur First, Writes to occur Second.
- E)None of above

Data Hazards

- R-format instructions/ any ALU computation can be successfully forwarded

To the instruction immediately following. Thereby avoiding a stall in the pipeline

- LW data coming from memory needs to also be forwarded to subsequent instruction: However one stall cannot be avoided. Load-Use Hazard

What Conditions am I looking for to detect a Data Hazard?

- add \$1, \$2, \$3
- and \$4, \$2, \$1   **Instruction uses destination of previous instruction**
- sub \$4 ,\$3, \$1   **Instruction uses destination of instruction 2 before it**

What Conditions am I looking for to detect a Data Hazard?

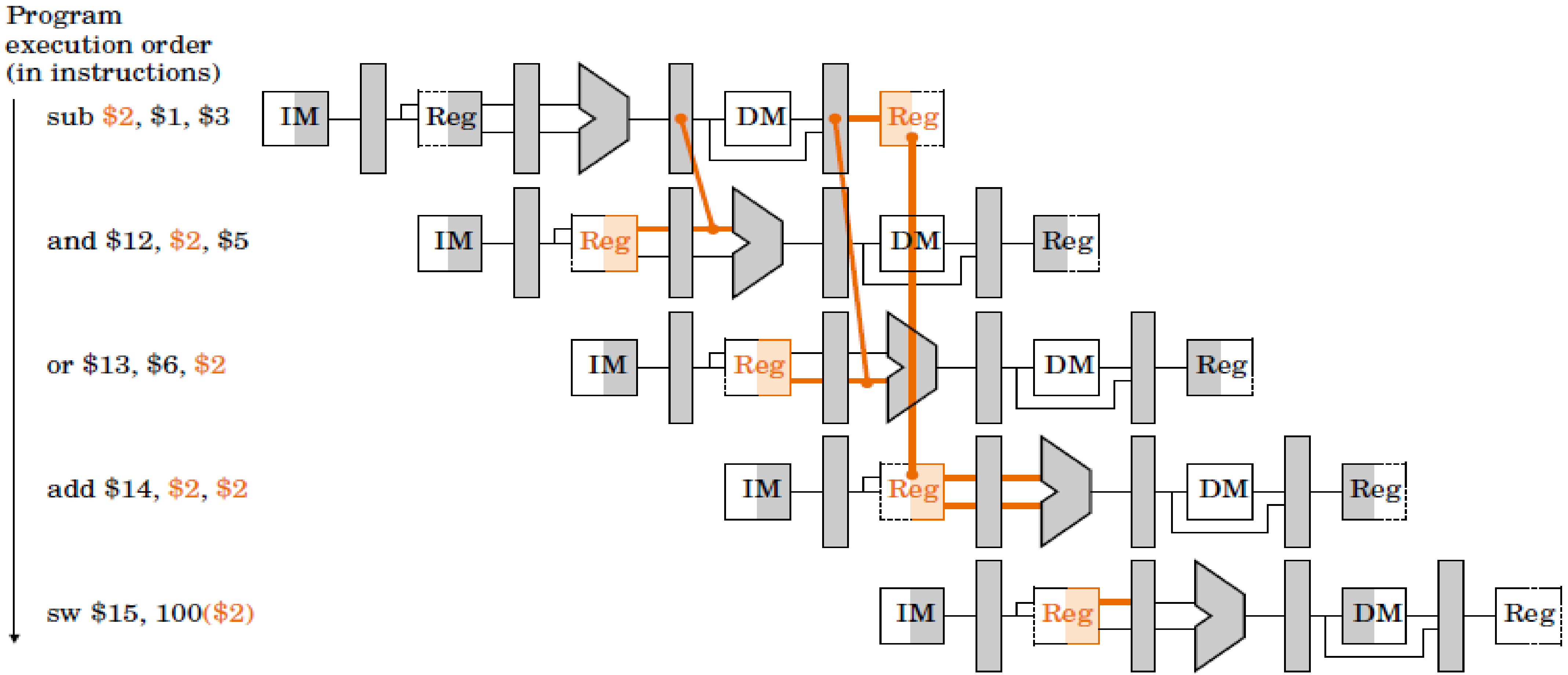
- add $1, $2, $3
- and $4, $2, $1   **Instruction uses destination of previous instruction**
- sub $4 ,$3, $1   **Instruction uses destination of instruction 2 before it**

- **Therefore to detect a data hazard : Check instructions $rs and $rt registers: did they depend on the $rd or $rt of the instruction before it**
- **OR 2 instructions before it**

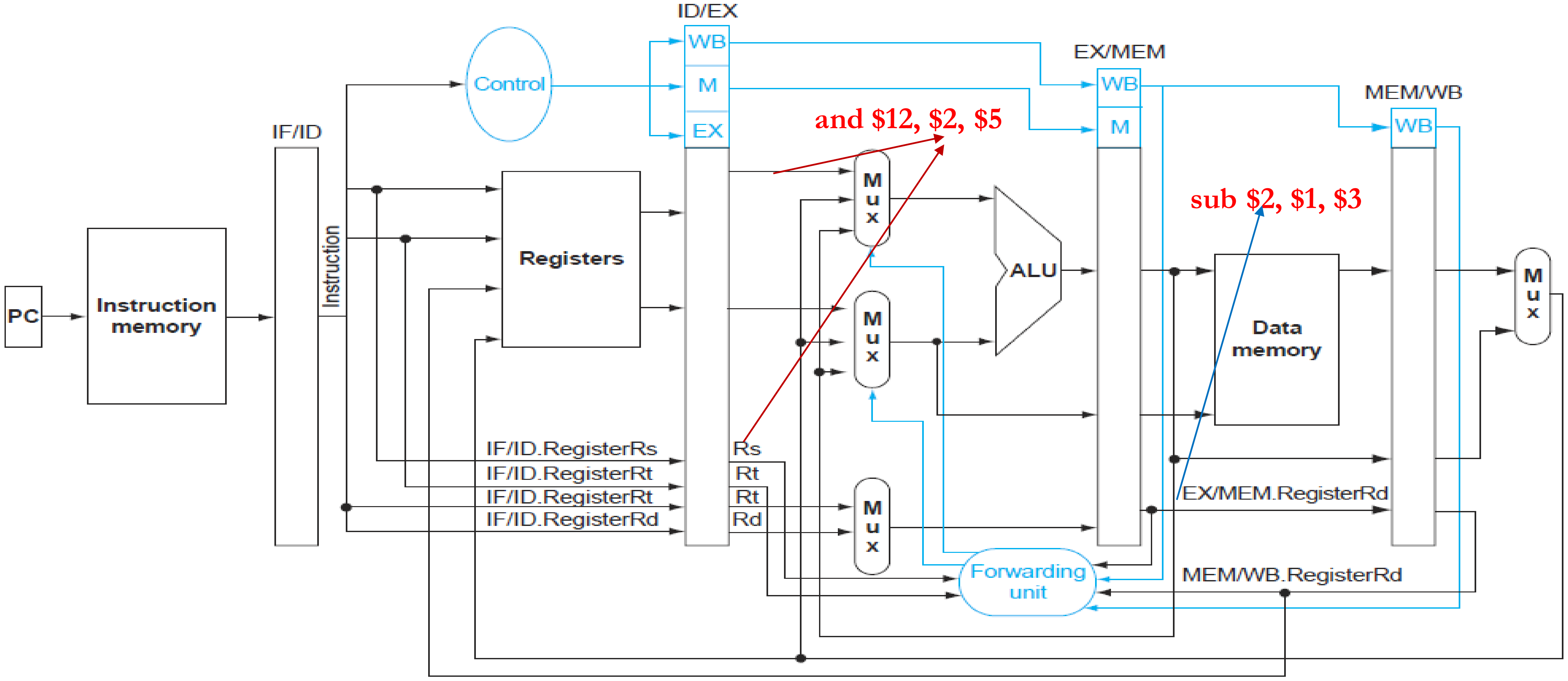
Pipeline Register Forwarding

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM :	X	X	X	–20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	–20	X	X	X	X



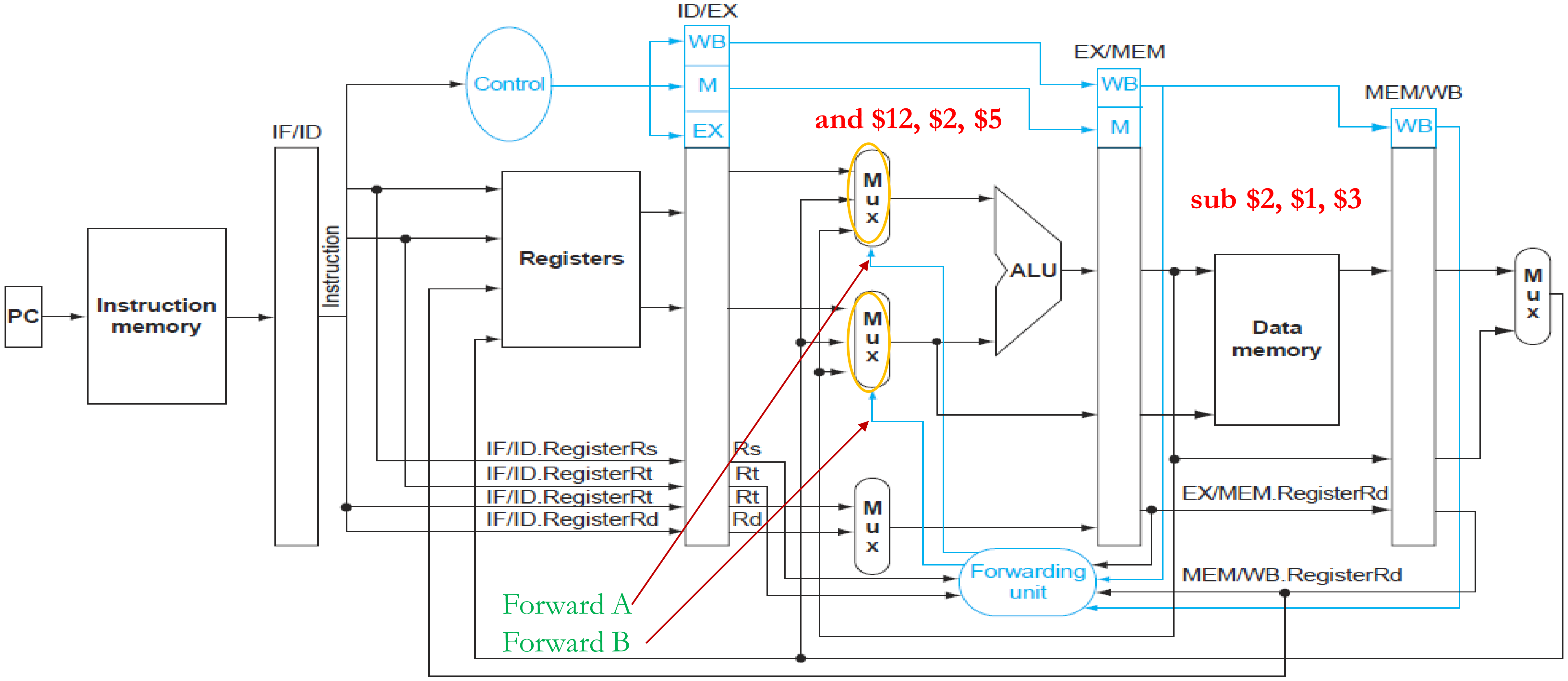
Datapath to handle forwarding



Data hazard: detection

Checking if 1st instruction is writing to $rd and 2nd instruction uses $rd as a source $rs.

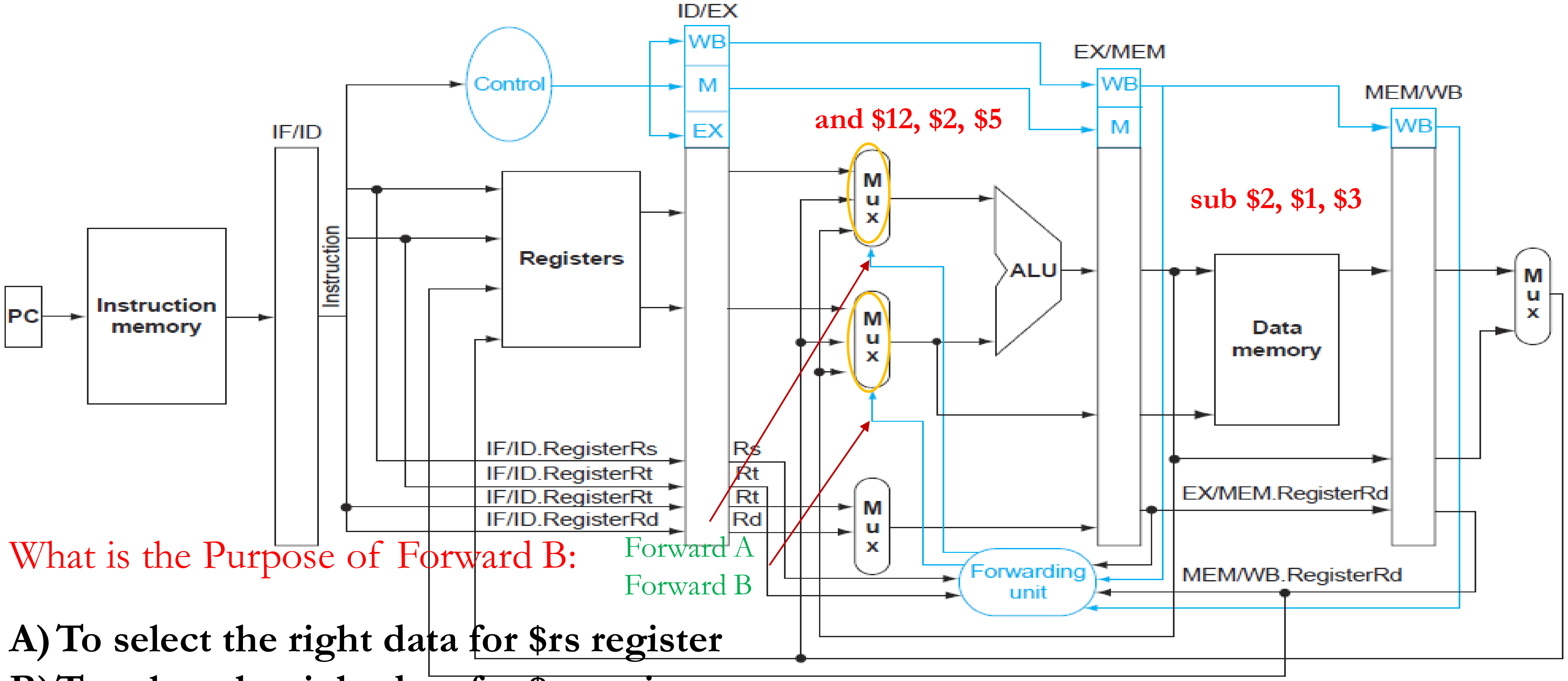
Datapath to handle forwarding



Data hazard: detection

If that is the case, Two New Multiplexors coming into ALU supply the Correct Forwarded Data. Forward A and Forward B need to be set accordingly.

Datapath to handle forwarding



What is the Purpose of Forward B:

A) To select the right data for $rs register

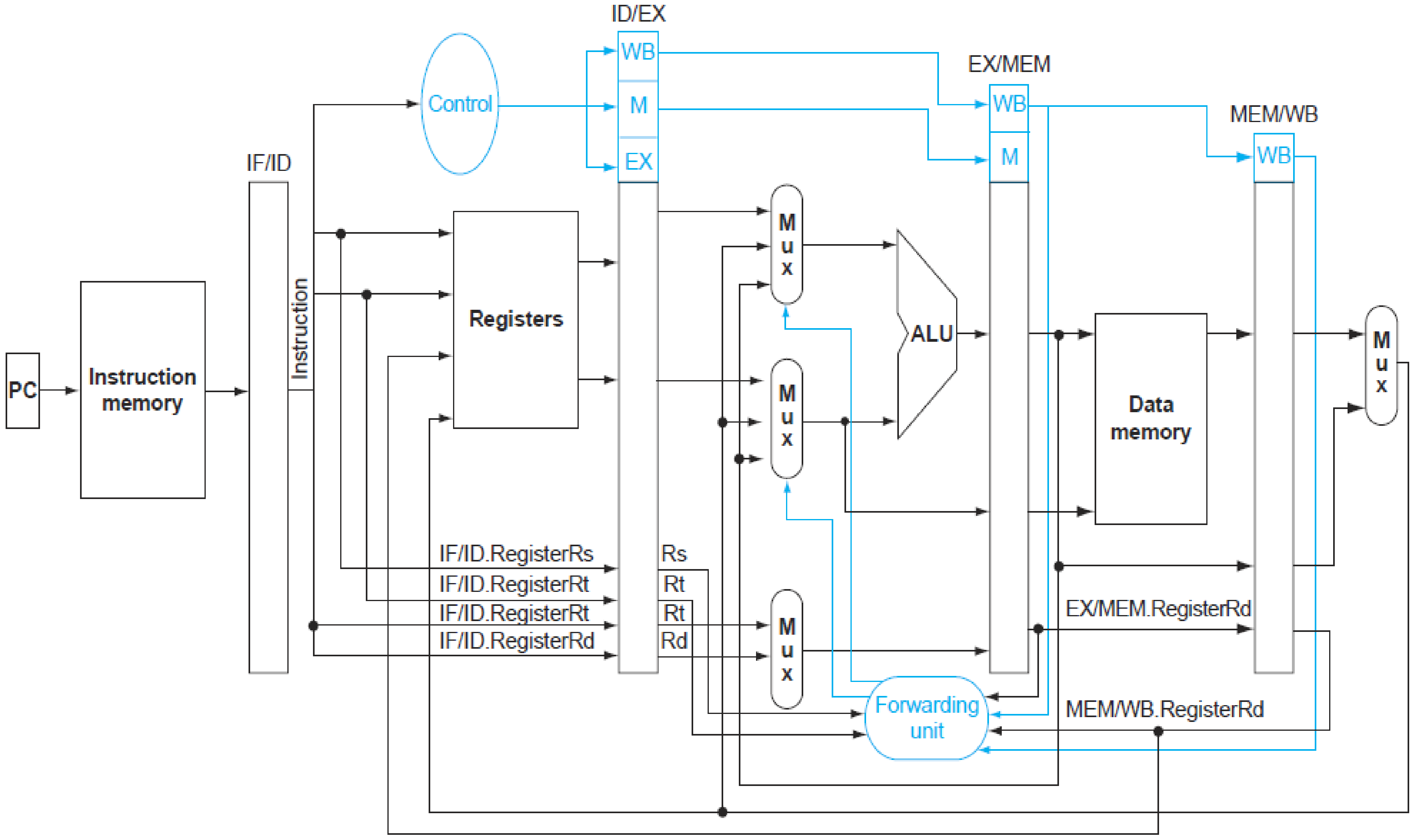
B) To select the right data for $rt register

C) To allow $rs and $rt to be sources into the ALU

D) To make sure Forwarded data comes from the Mem Stage also

E) Data hazard was with current $rd register- Forward B selects the correct forwarded data

Conditions That Need to be Checked in Hardware:



- EX Hazard 1a

  if (EX/MEM.RegWrite

  and (EX/MEM.RegisterRd $\neq$ 0)

  and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 10

- MEM Hazard 2a

  if (MEM/WB.RegWrite

  and (MEM/WB.RegisterRd $\neq$ 0)

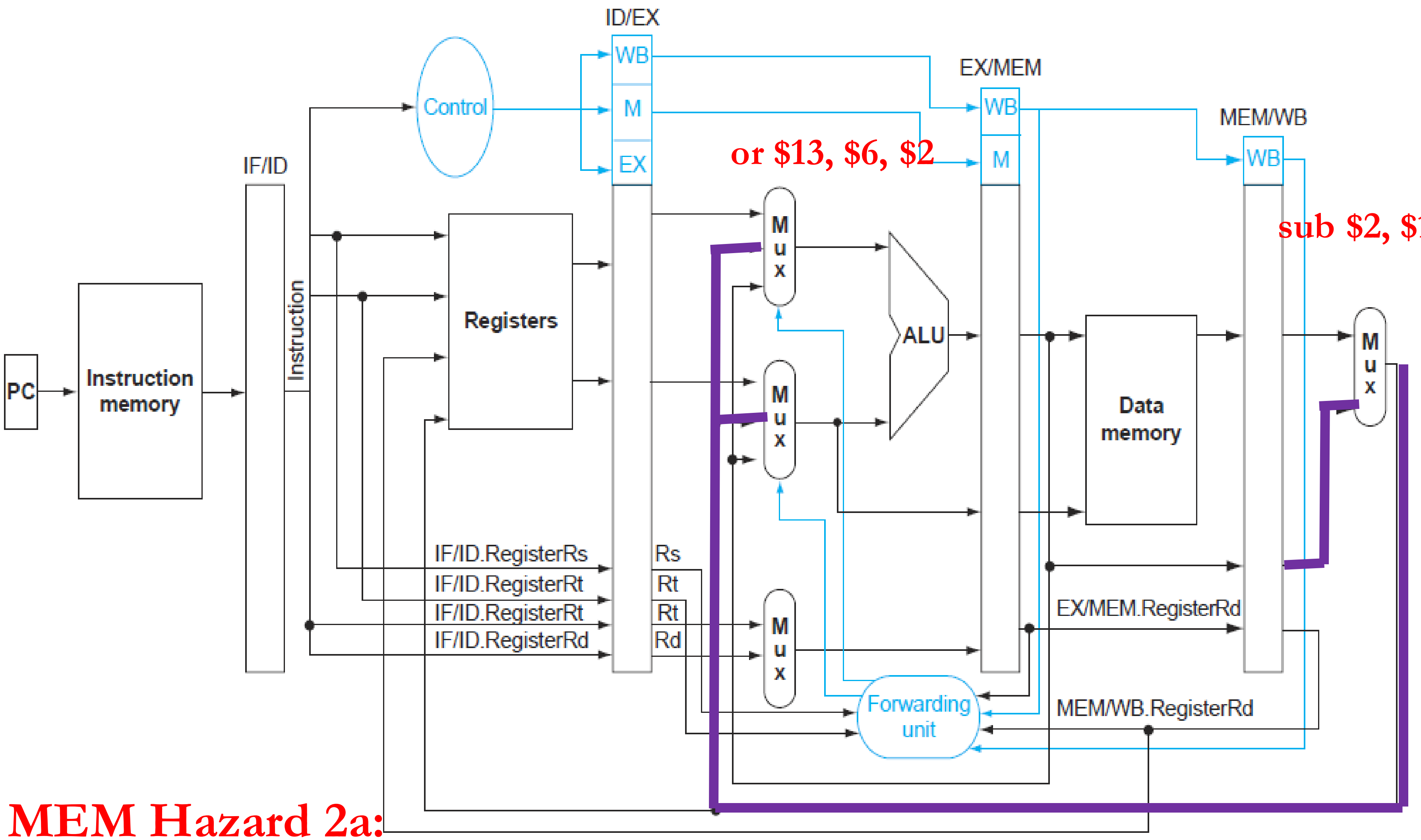
  and (EX/MEM.RegisterRd $\neq$ ID/EX.RegisterRs)

  and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 01

- 1b,2b are similar, but with Rt, ForwardB

Conditions That Need to be Checked in Hardware:



MEM Hazard 2a:

Checking if 1st instruction is writing to $rd and 3rd instruction uses $rd as a source $rs. If instruction between 1st and 3rd is also writing to $rd, then this forwarding was already done in Hazard 1a detection

- EX Hazard 1a

  if (EX/MEM.RegWrite

  and (EX/MEM.RegisterRd ≠ 0)

  and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 10

- MEM Hazard 2a

  if (MEM/WB.RegWrite

  and (MEM/WB.RegisterRd ≠ 0)

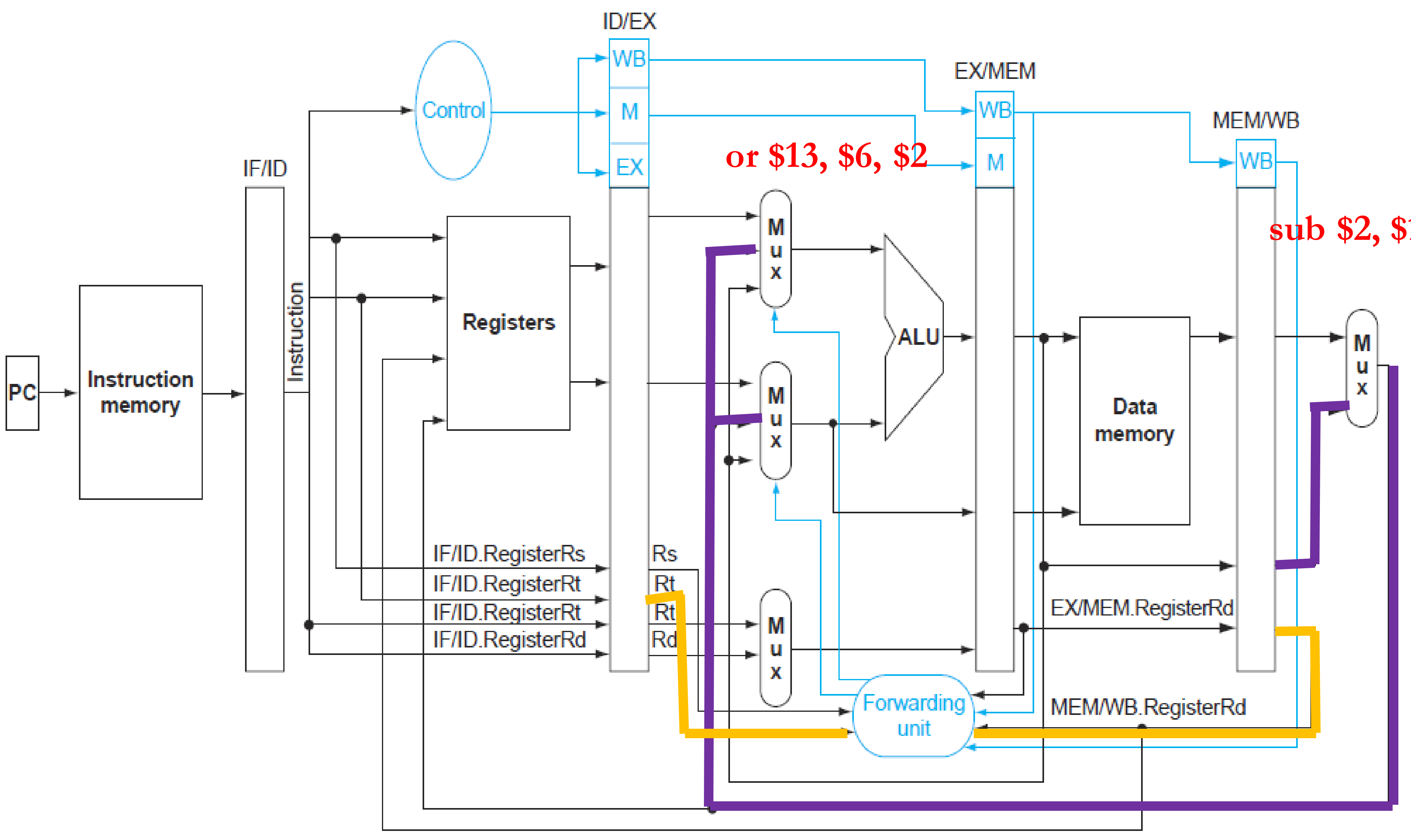
  and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)

  and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 01

- 1b,2b are similar, but with Rt, ForwardB

Conditions That Need to be Checked in Hardware:



- EX Hazard 1a

  if (EX/MEM.RegWrite

  and (EX/MEM.RegisterRd ≠ 0)

  and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 10

- MEM Hazard 2a

  if (MEM/WB.RegWrite

  and (MEM/WB.RegisterRd ≠ 0)

  and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)

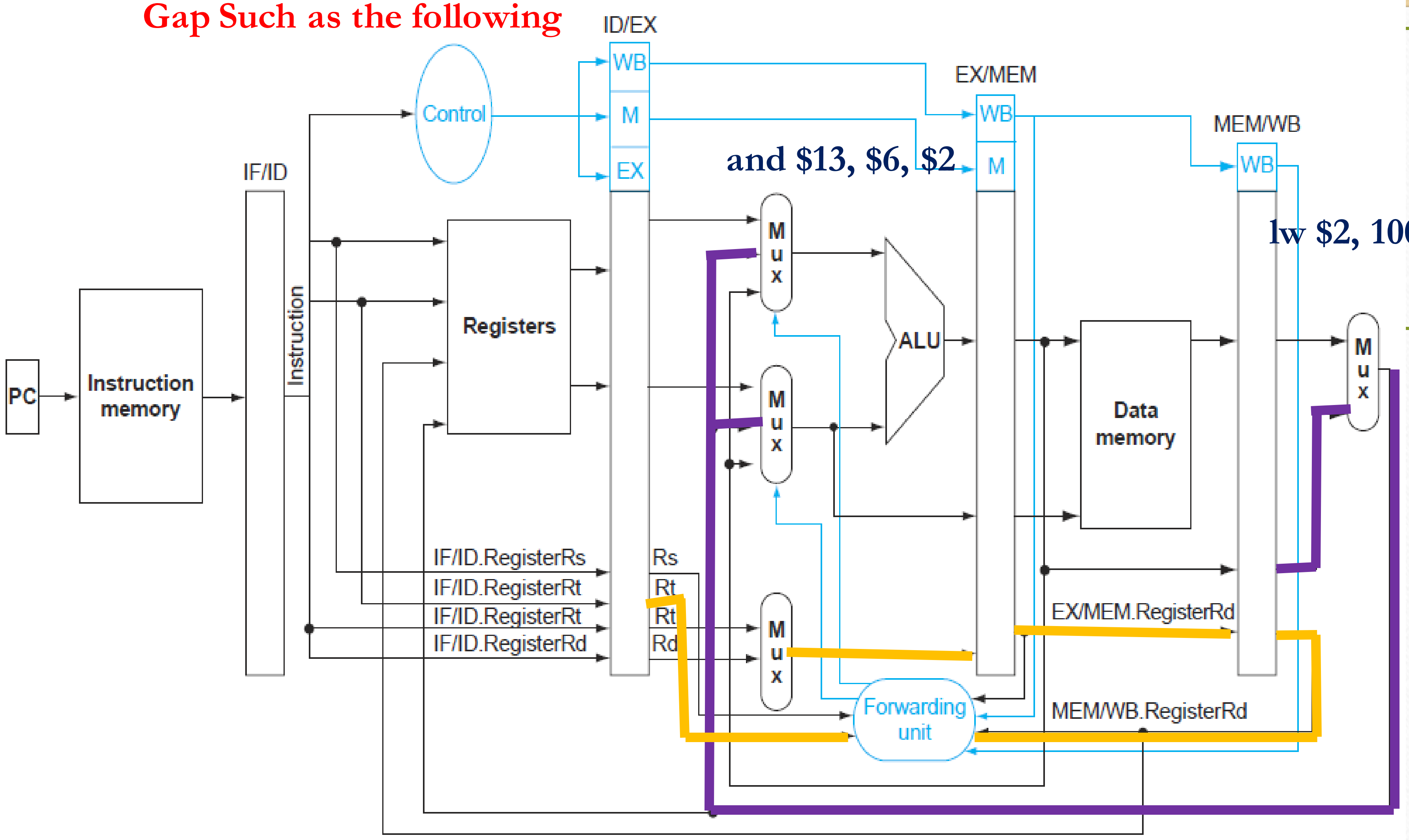
  and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

  then ForwardA = 01

  Rt

- 1b,2b are similar, but with Rt, ForwardB

LW instruction : Forwarding would work with One Instruction Gap Such as the following



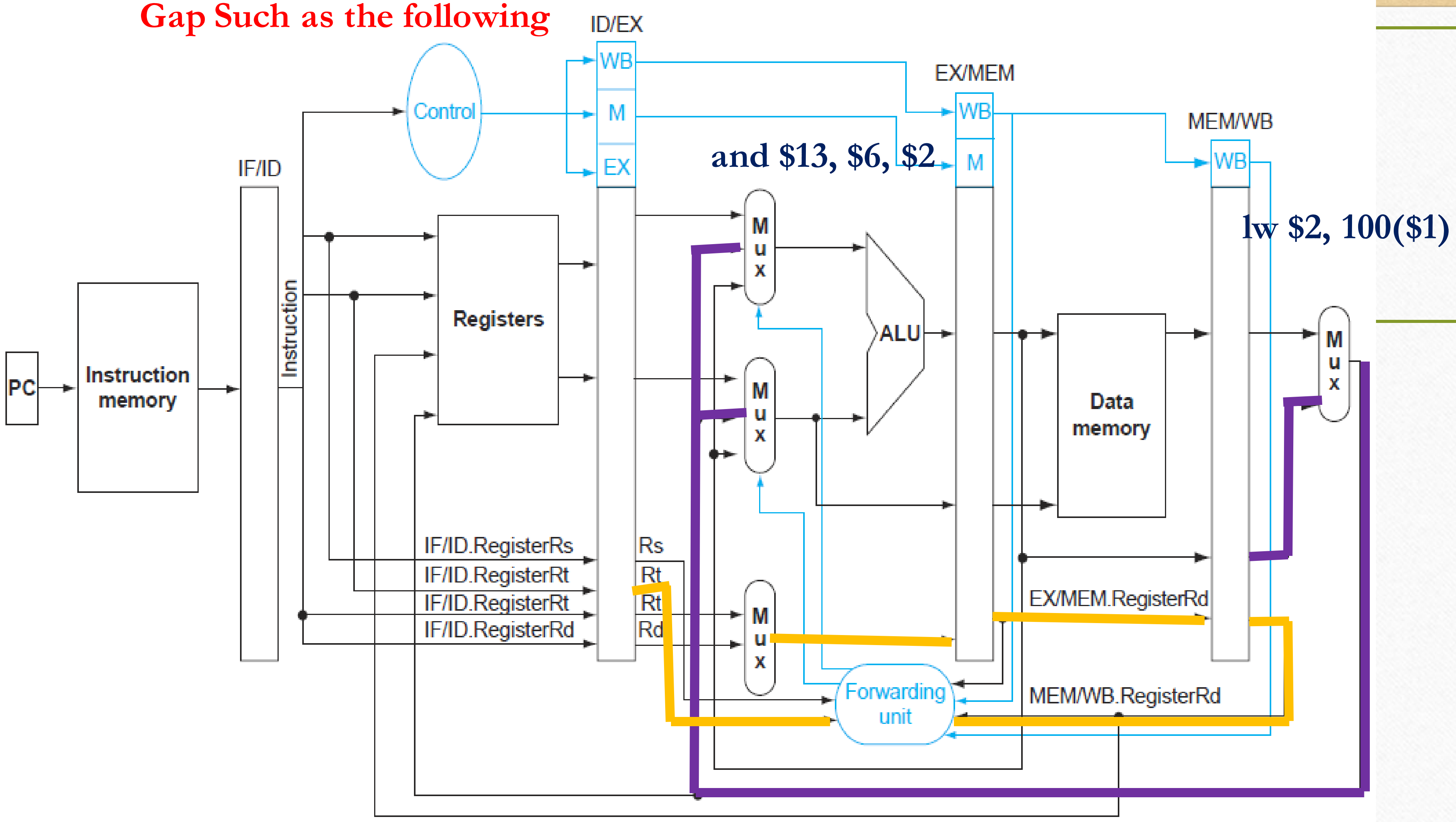
Instructions:

100 lw $2, 100($1)

104 add $6, $3, $3

108 and $13, $6, $2

LW instruction : Forwarding would work with One Instruction Gap Such as the following



Instructions:

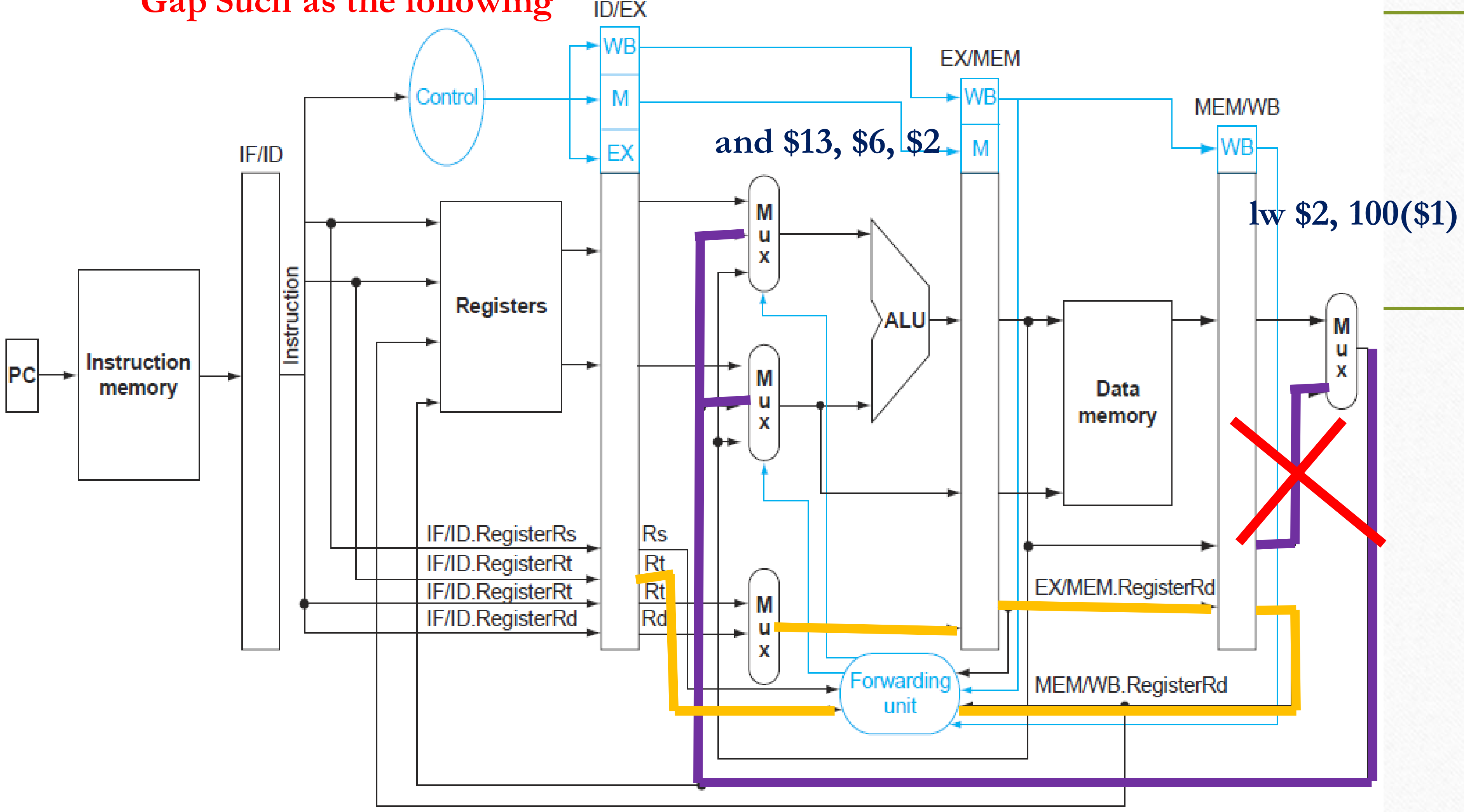
100 lw $2, 100($1)

104 add $6, $3, $3

108 and $13, $6, $2

Data for lw is not Ready until MEM stage So it can be forwarded back to an instruction TWO behind.

LW instruction : Forwarding would work with One Instruction Gap Such as the following



Instructions:

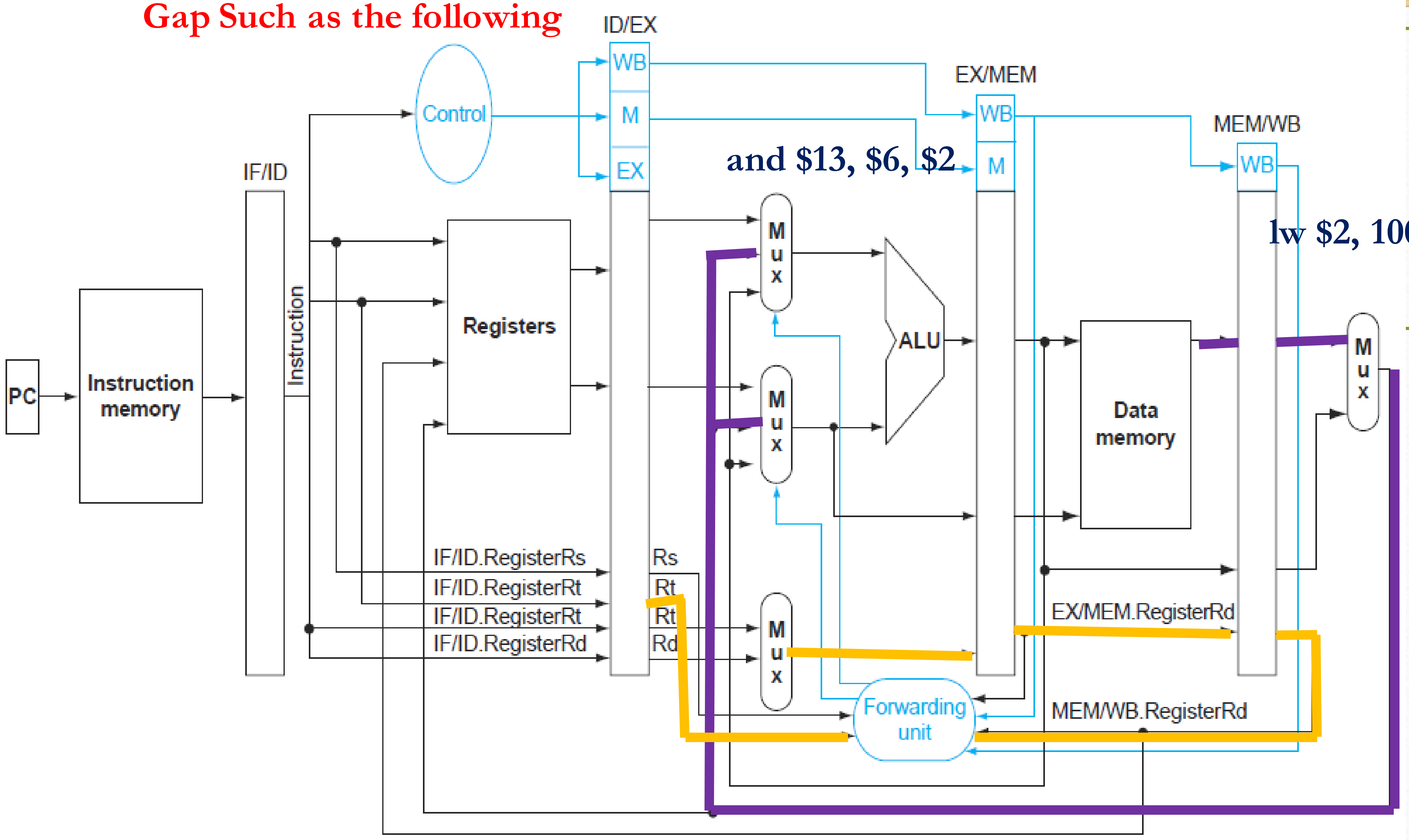
100 lw $2, 100($1)

104 add $6, $3, $3

108 and $13, $6, $2

Data for lw is not Ready until MEM stage So it can be forwarded back to an instruction TWO behind.

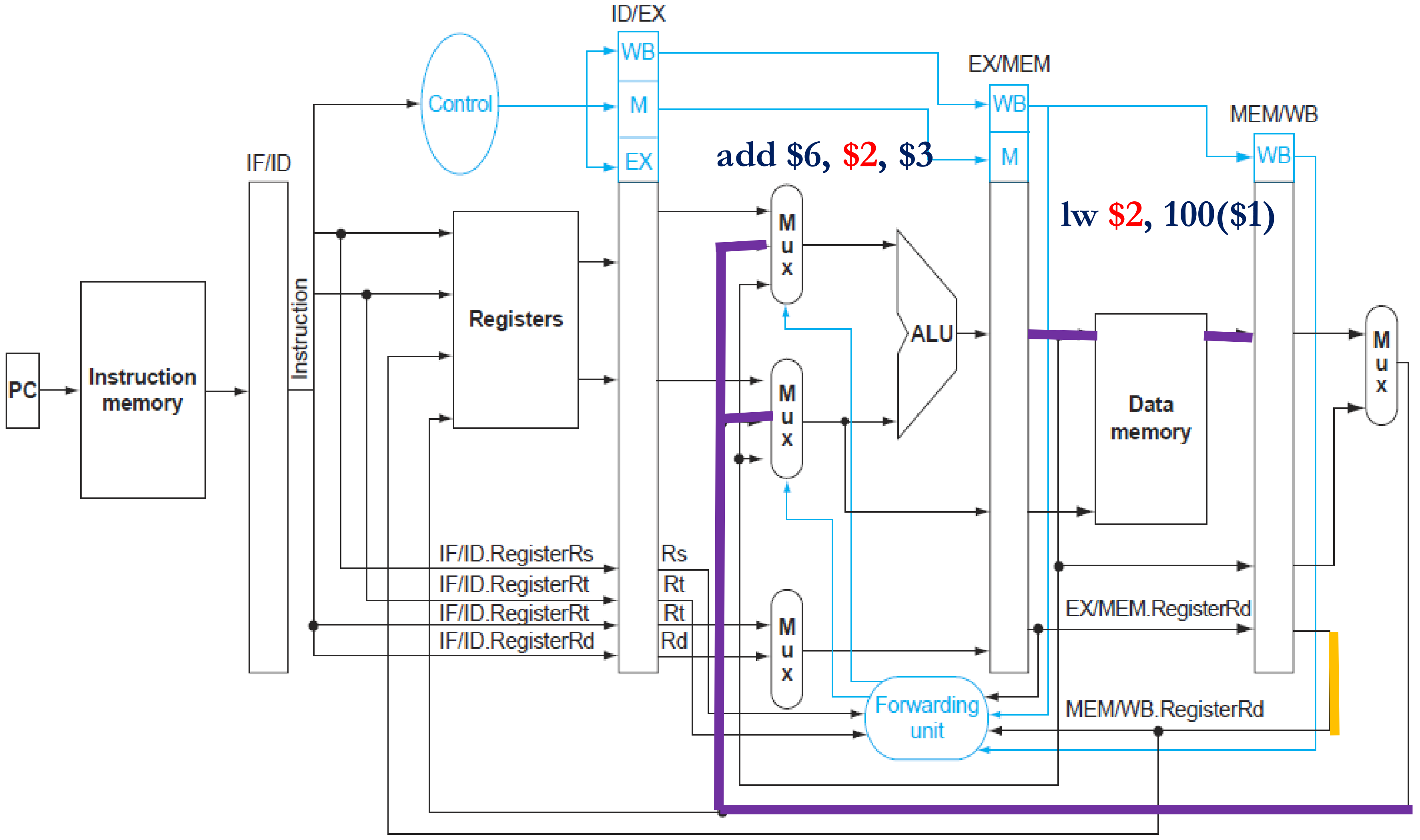
LW instruction : Forwarding would work with One Instruction Gap Such as the following



Instructions:
100 lw $2, 100($1)
104 add $6, $3, $3
108 and $13, $6, $2

Data for lw is not Ready until MEM stage So it can be forwarded back to an instruction TWO behind.

LW instruction : What about if Next instruction after LW needs Data

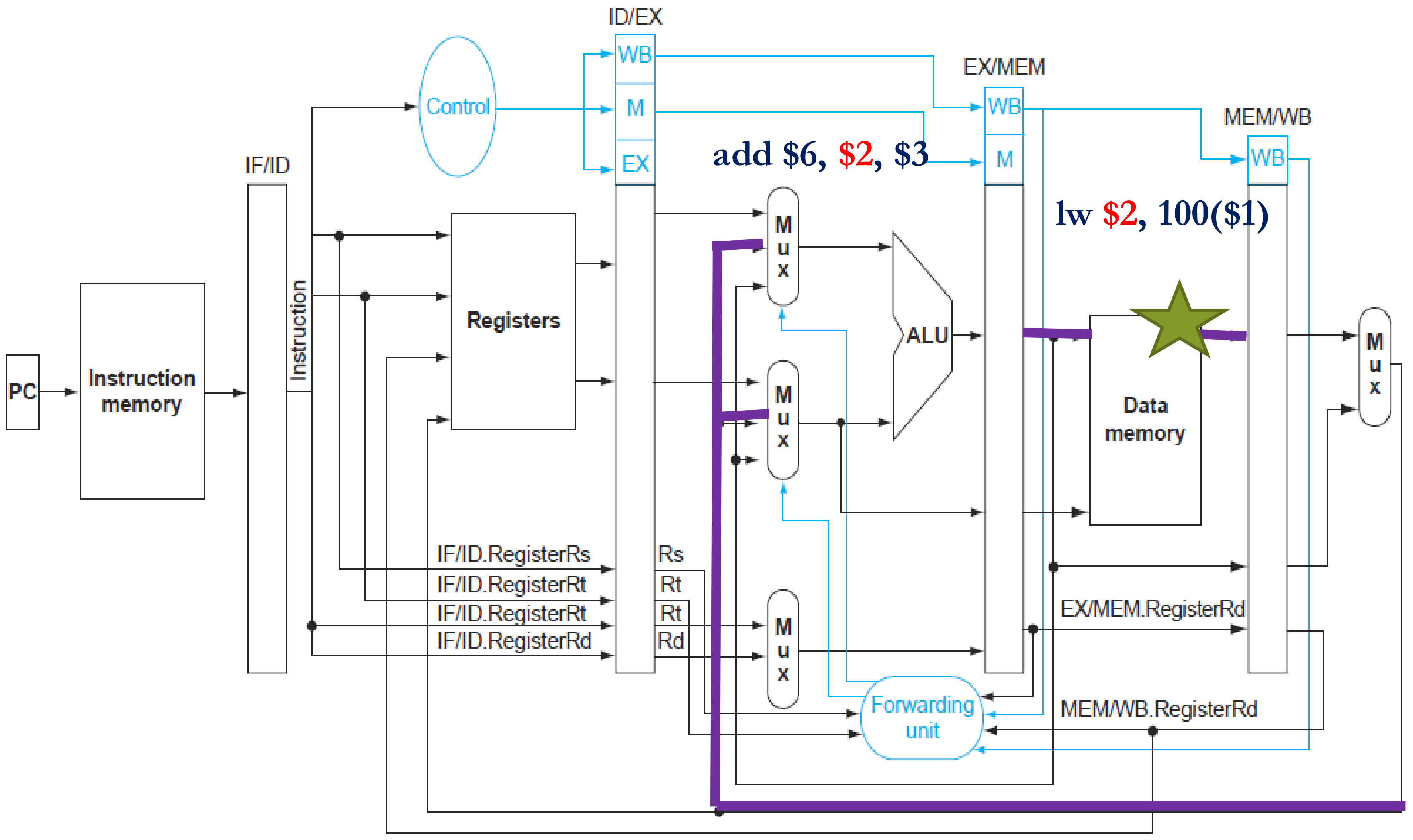


Instructions:
100 lw $2, 100($1)
104 add $6, $2, $3
108 and $13, $6, $2

Data is not available For Instruction Directly Behind The Lw

NEED ONE Stall.

LW instruction : What about if Next instruction after LW needs Data

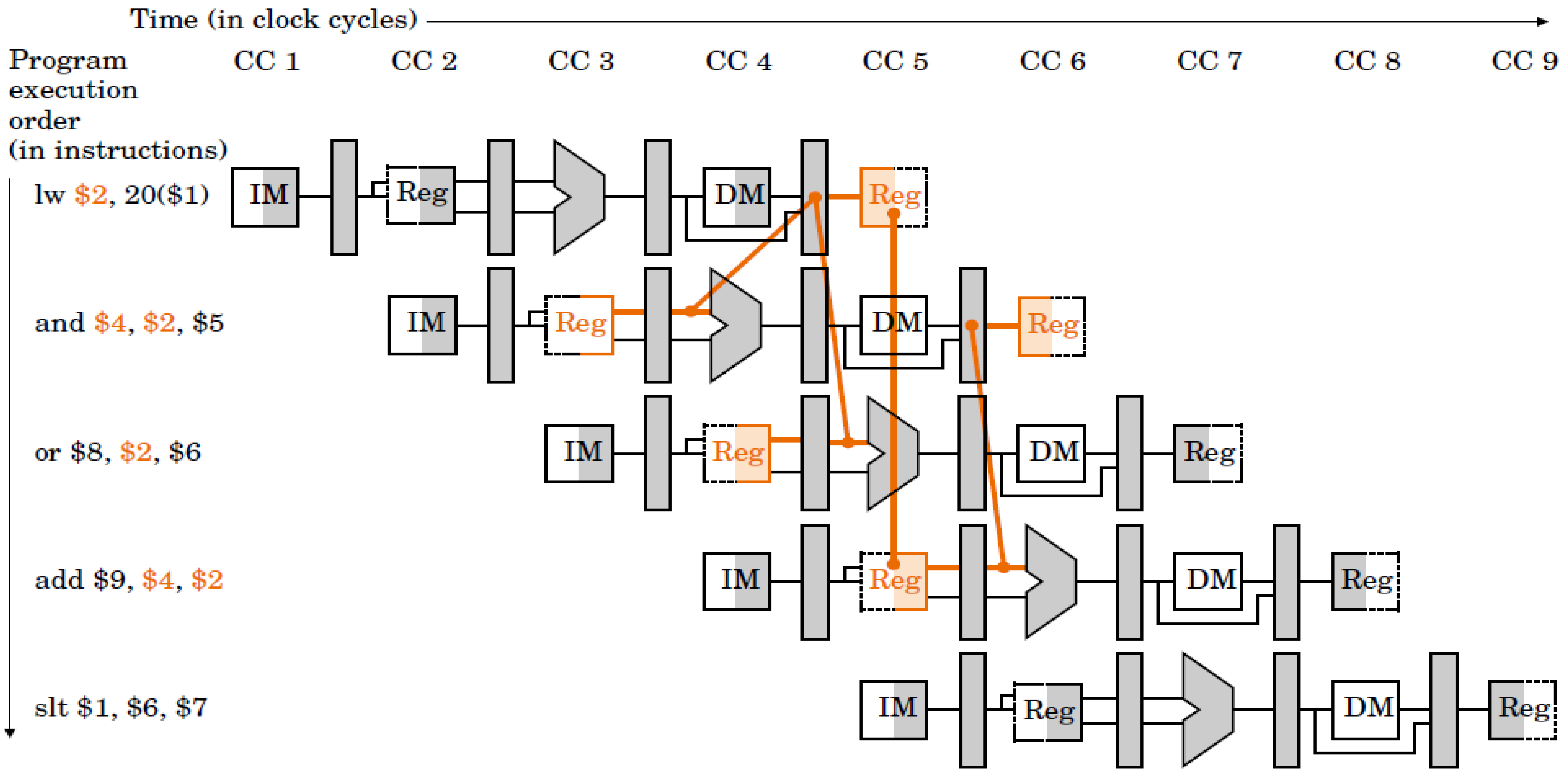


Instructions:
100 lw $2, 100($1)
104 add $6, $2, $3
108 and $13, $6, $2

Load-Use Hazard.

Does the Instruction directly after a lw, use the lw destination Register?

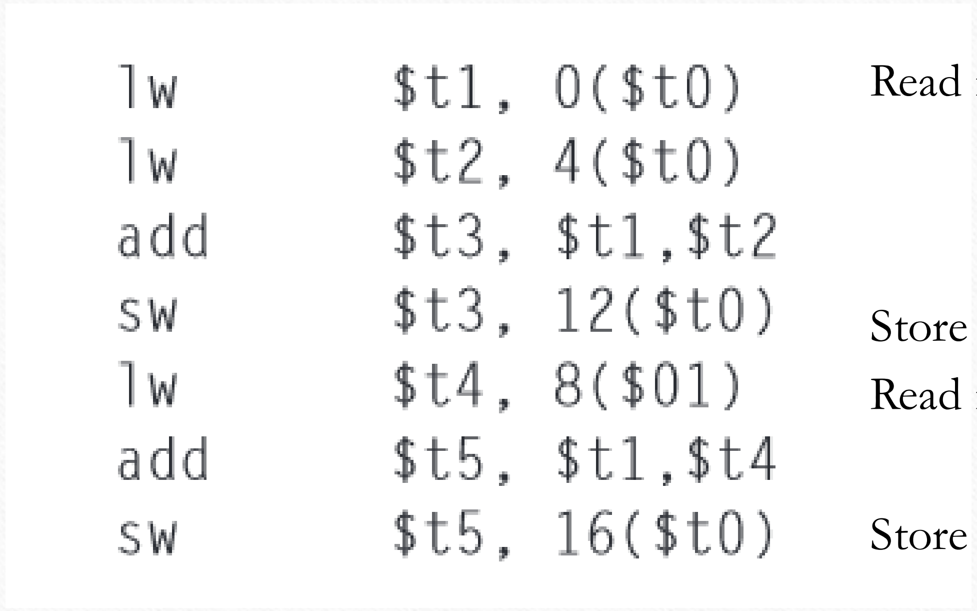
Data Hazards and Stalls



Load-use hazard requires a stall

- A= B + E
- C= B + F

- **Sample Implementation in MIPSB +E**



```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

Read from B, E in Memory

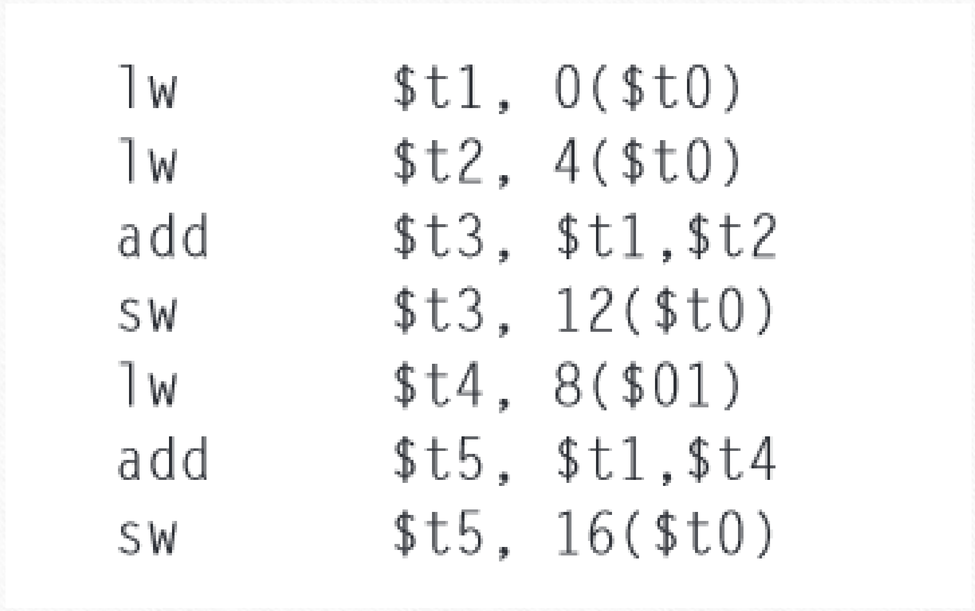
Store back B+E into location A

Read from F in Memory

Store back B+F into location C

- A= B + E
- C= B + F

- **Sample Implementation in MIPS**

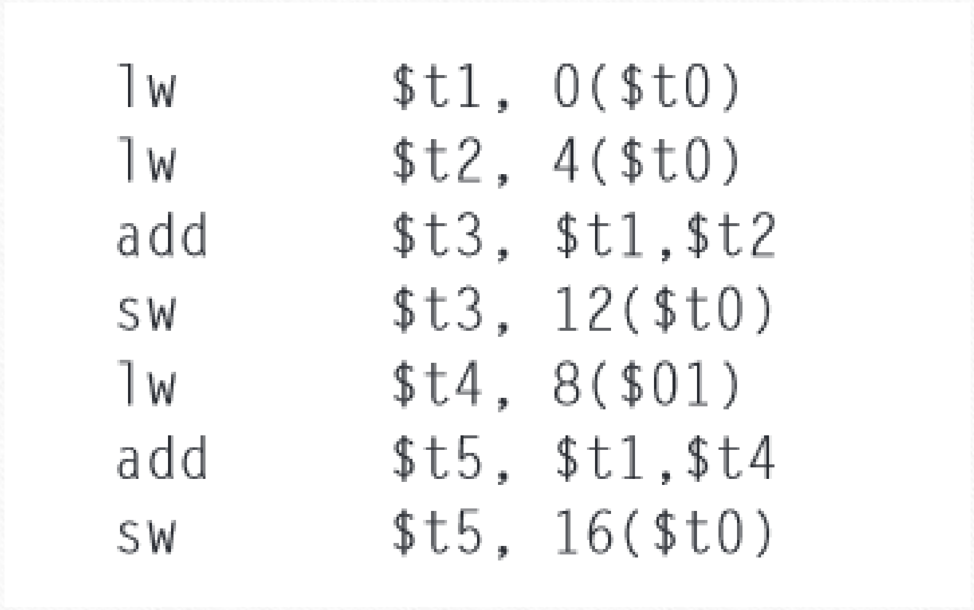


```
lw     $t1, 0($t0)
lw     $t2, 4($t0)
add    $t3, $t1,$t2
sw     $t3, 12($t0)
lw     $t4, 8($01)
add    $t5, $t1,$t4
sw     $t5, 16($t0)
```

Do we have Load-Use Hazard?

- A= B + E
- C= B + F

- **Sample Implementation in MIPS**



```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

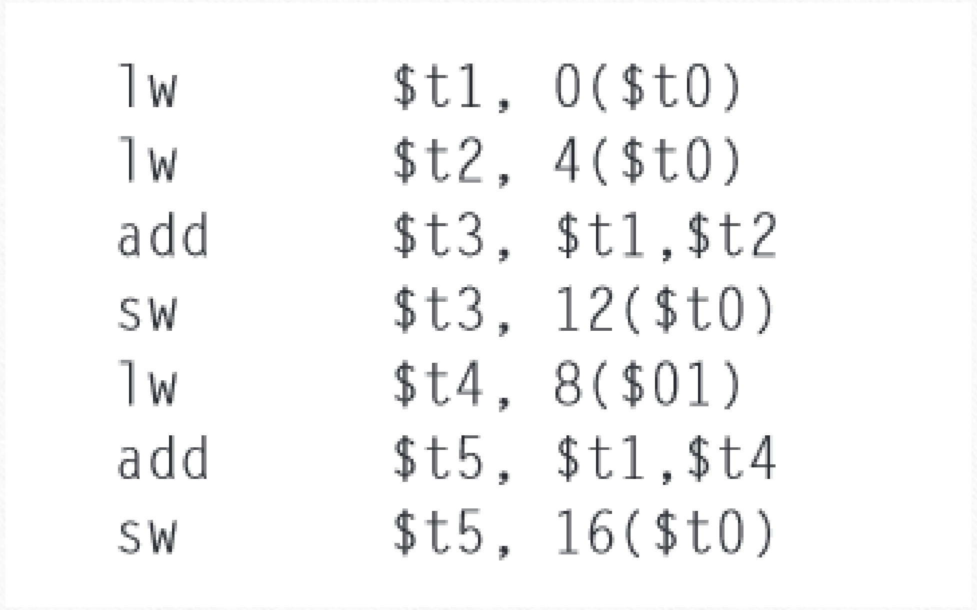
Do we have Load-Use Hazard?

How many Load-Use Hazards

1) 1
2) 2
3) 3
4) none

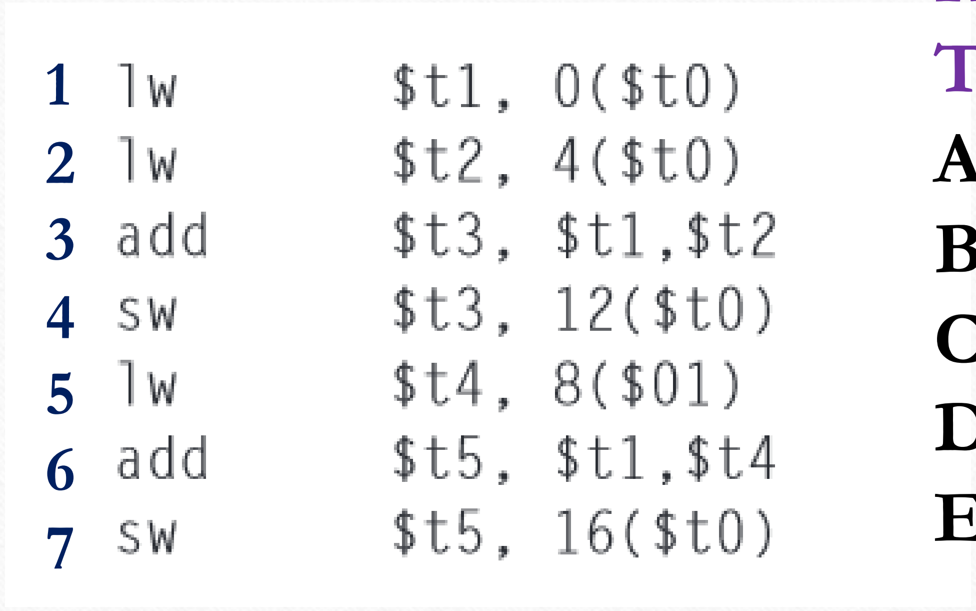
- A= B + E
- C= B + F

- **Sample Implementation in MIPS**



```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

Rearrange Code: Place another Instruction below LW That would need to execute anyways



```
1 lw      $t1, 0($t0)
2 lw      $t2, 4($t0)
3 add     $t3, $t1,$t2
4 sw      $t3, 12($t0)
5 lw      $t4, 8($01)
6 add     $t5, $t1,$t4
7 sw      $t5, 16($t0)
```

How might I rearrange the code
To remove Load-Use Hazard?

A) Swap line 5 and 6
B) Swap line 4 and 5
C) Insert line 5 between lines 2 and 3
D) Swap line 3 and 5
E) Insert line5 between line 3 and 4

- A= B + E
- C= B + F

- **Sample Implementation in MIPS**

```
1 lw      $t1, 0($t0)
2 lw      $t2, 4($t0)
3 add     $t3, $t1,$t2
4 sw      $t3, 12($t0)
5 lw      $t4, 8($01)
6 add     $t5, $t1,$t4
7 sw      $t5, 16($t0)
```

Code Rearrangement

```
1 lw      $t1, 0($t0)
2 lw      $t2, 4($t1)
3 lw      $t4, 8($01)
4 add     $t3, $t1,$t2
5 sw      $t3, 12($t0)
6 add     $t5, $t1,$t4
7 sw      $t5, 16($t0)
```

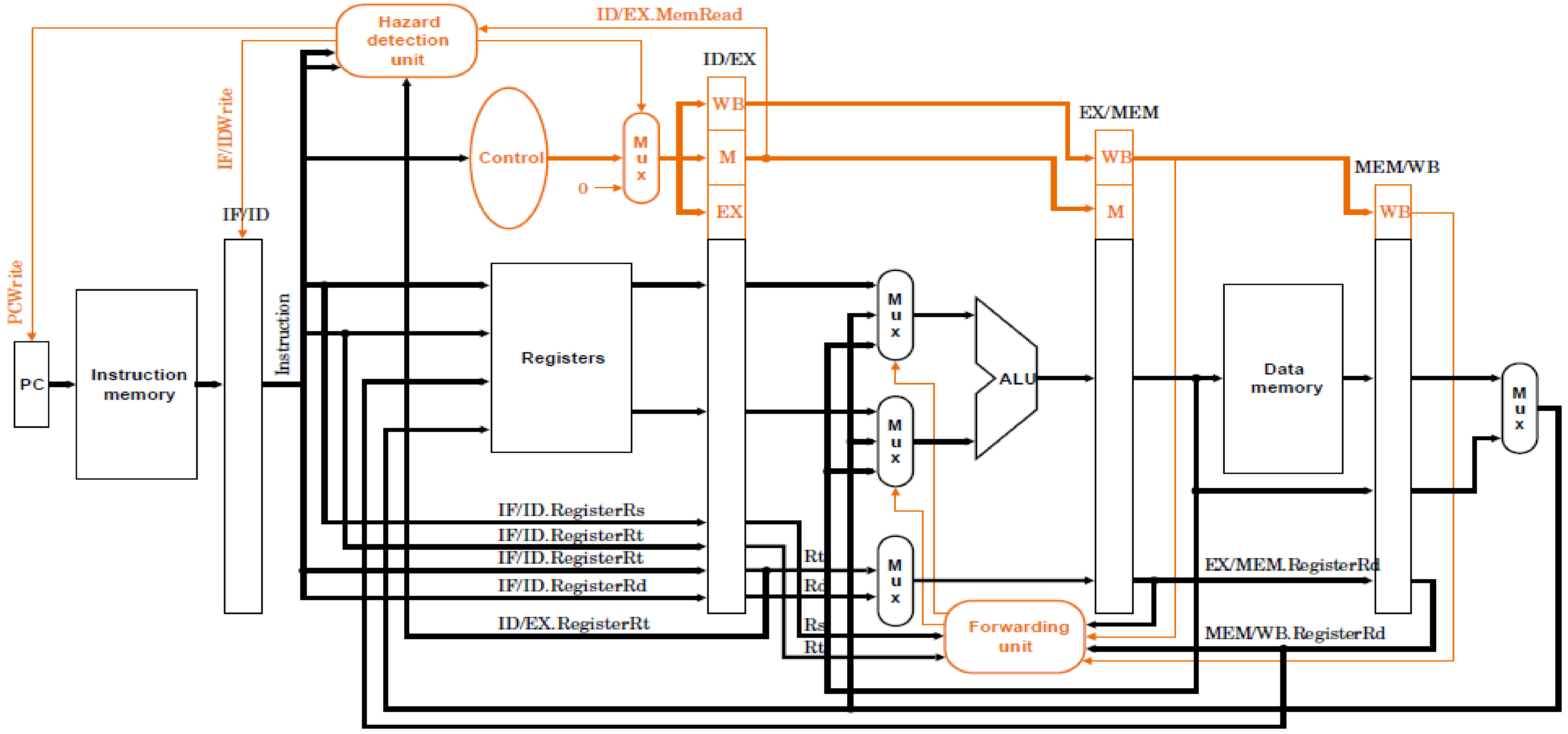
Data Forwarding and other instructions

```
add $2, $3, $4
sw $2, 100($3)
```

We assume that sw instruction is also okay because of data forwarding. The hardware to implement this is missing from our diagrams of the datapath for simplicity and clarity.

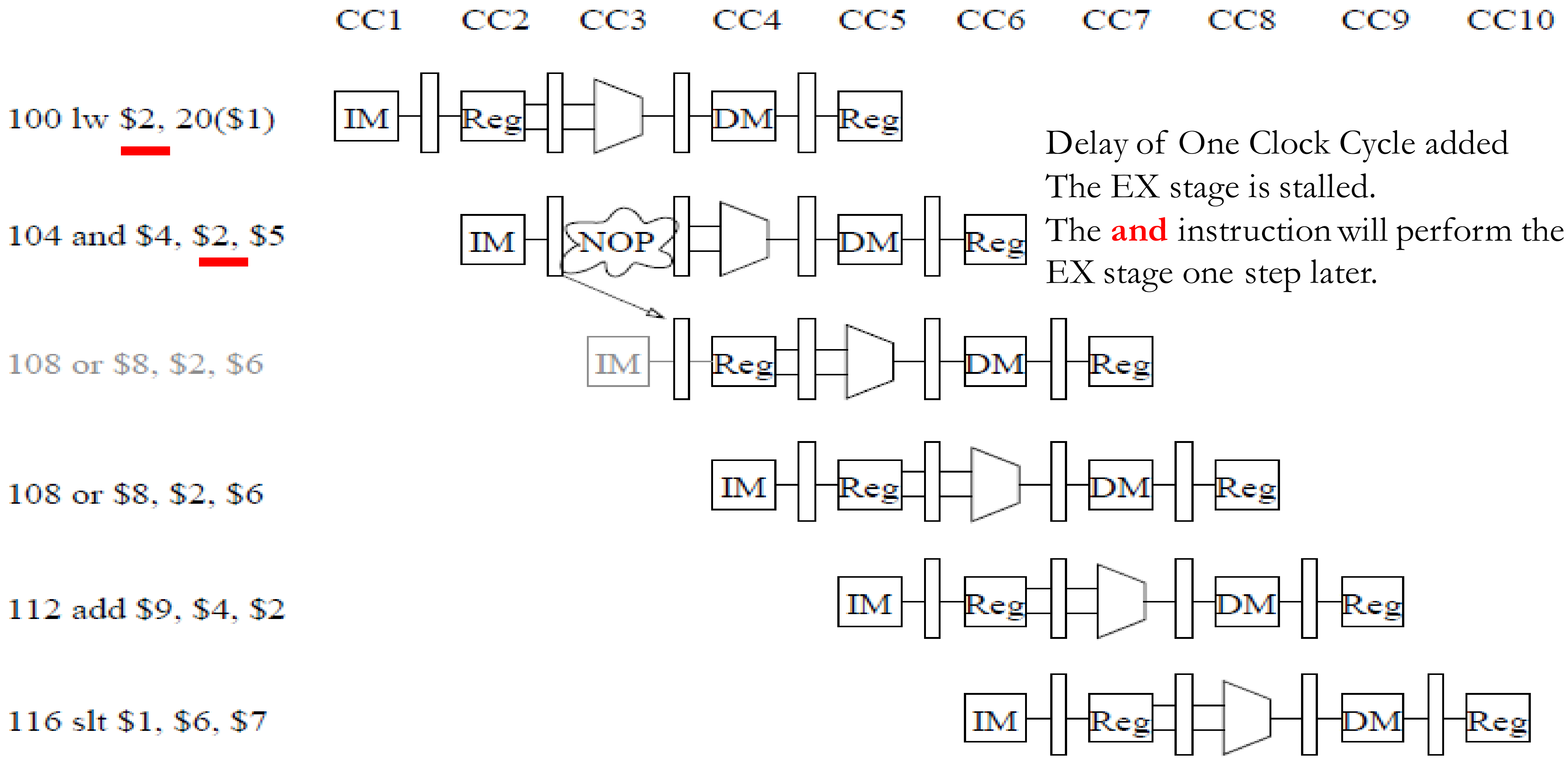
However, we assume forwarding for the **sw** instruction is in place

Hardware for Stall

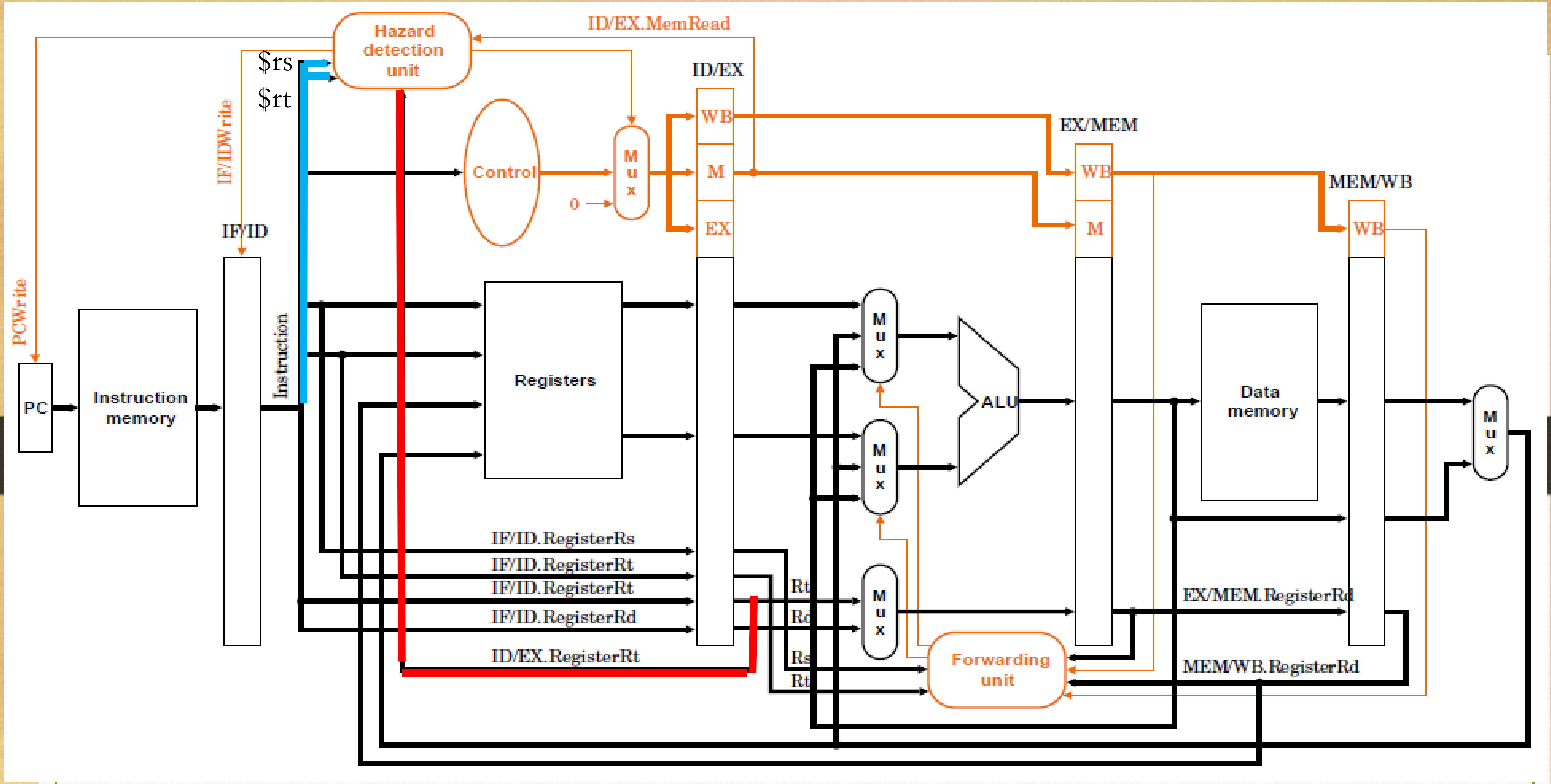


Signed-immediate and branch hardware omitted for clarity

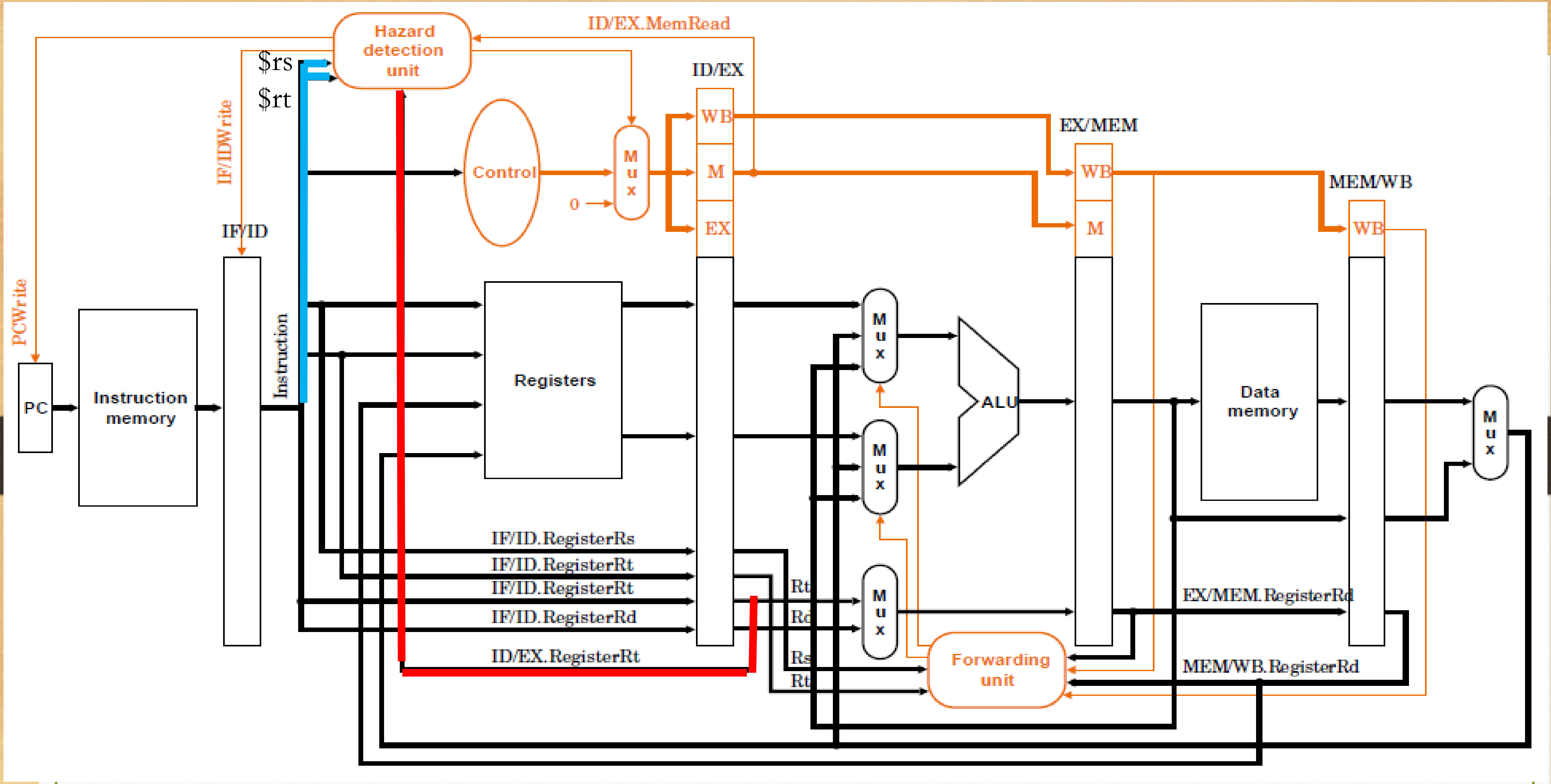
The Real Effect of a Stall



Delay of One Clock Cycle added
The EX stage is stalled.
The **and** instruction will perform the EX stage one step later.

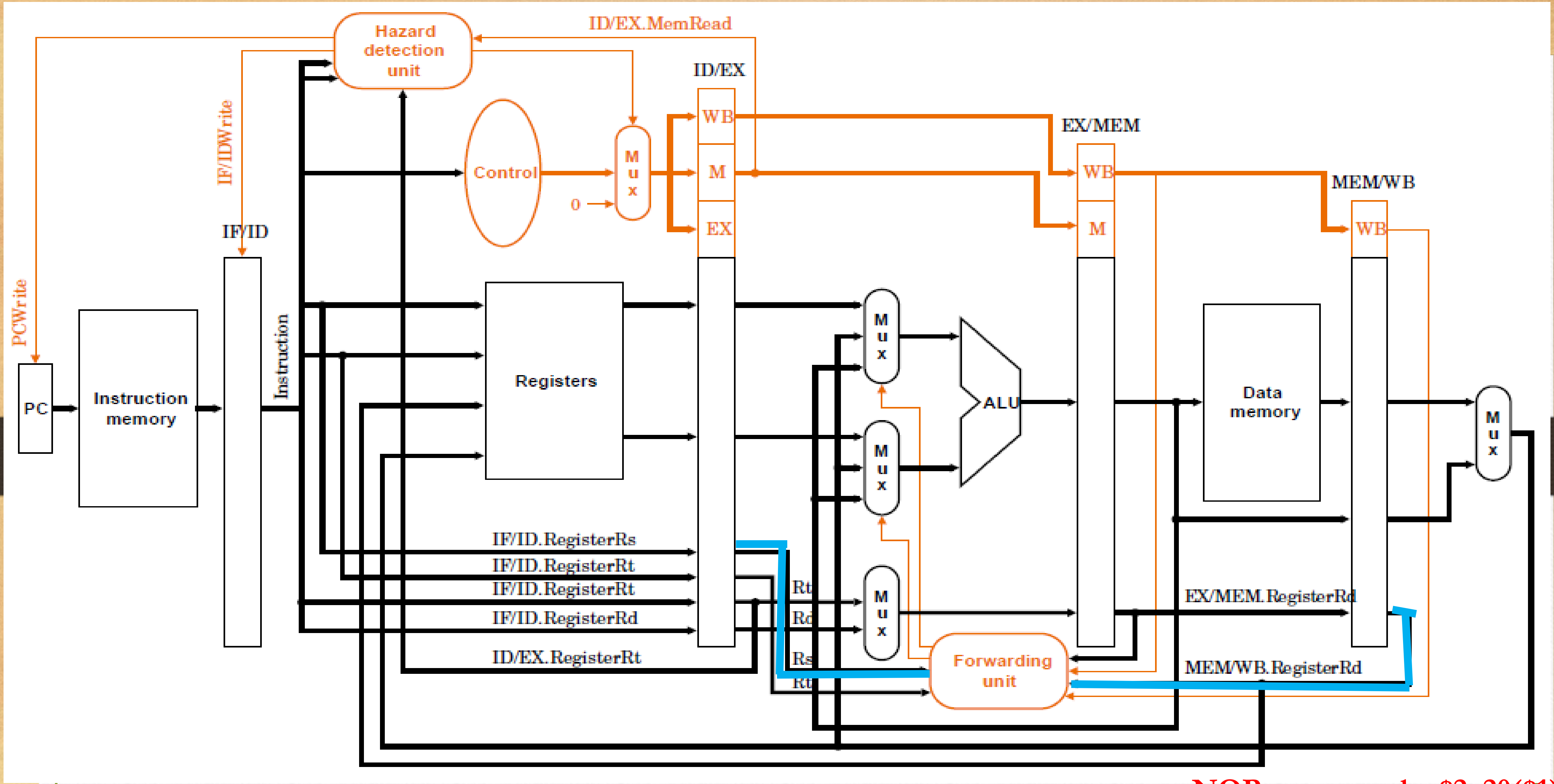


or $8, $2 , $6 and $4, $2, $5 lw $2, 20($1)



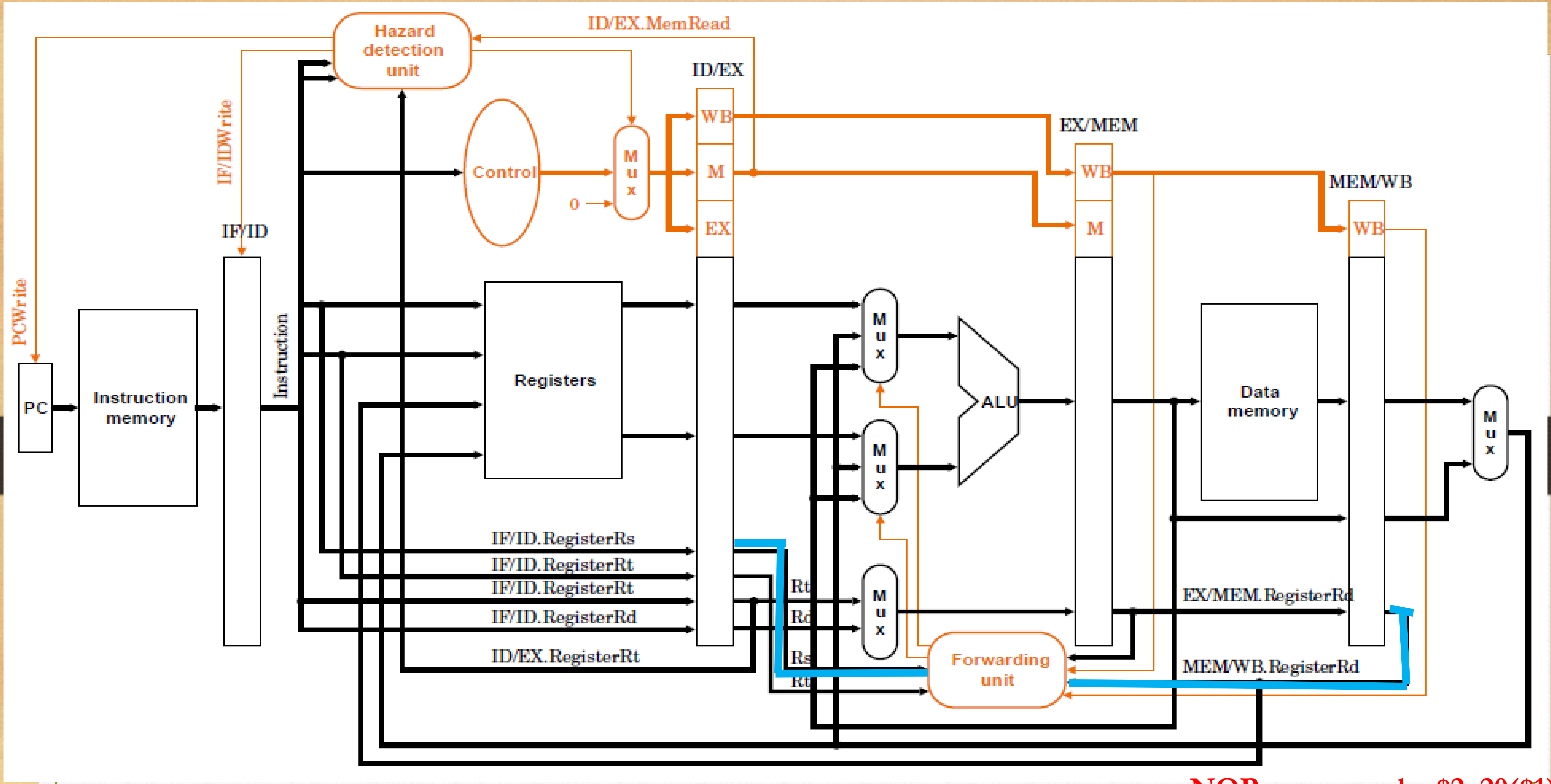
or \$8, \$2 , \$6 and \$4, \$2, \$5 NOP lw \$2, 20(\$1)

And and Or instructions stay where they are: No New Instruction is Fetched



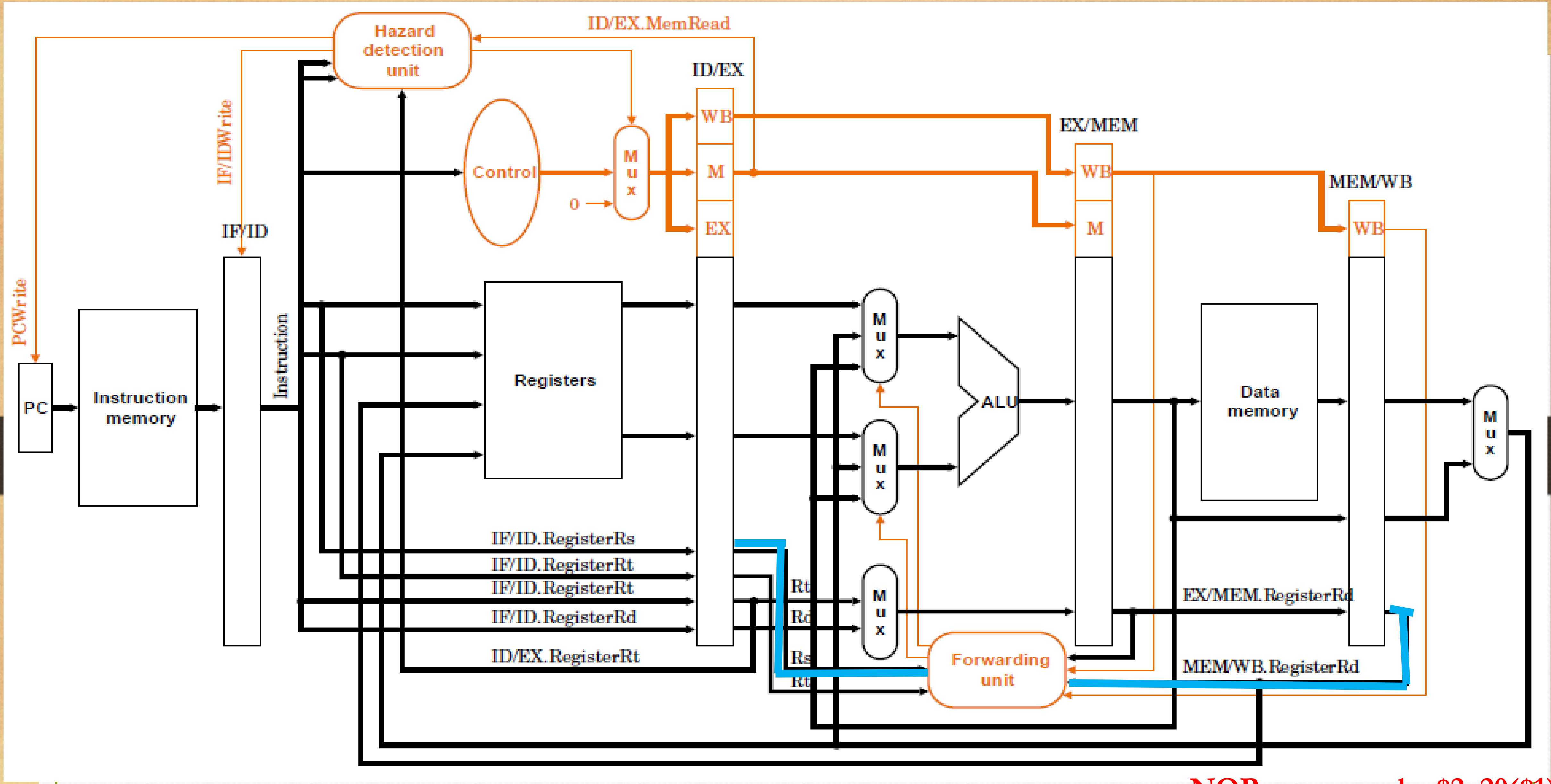
add \$9, \$4 , \$2 or \$8, \$2 , \$6 and \$4, \$2, \$5 NOP lw \$2, 20(\$1)

Now there is One Space Between **and** instruction and **lw** instruction



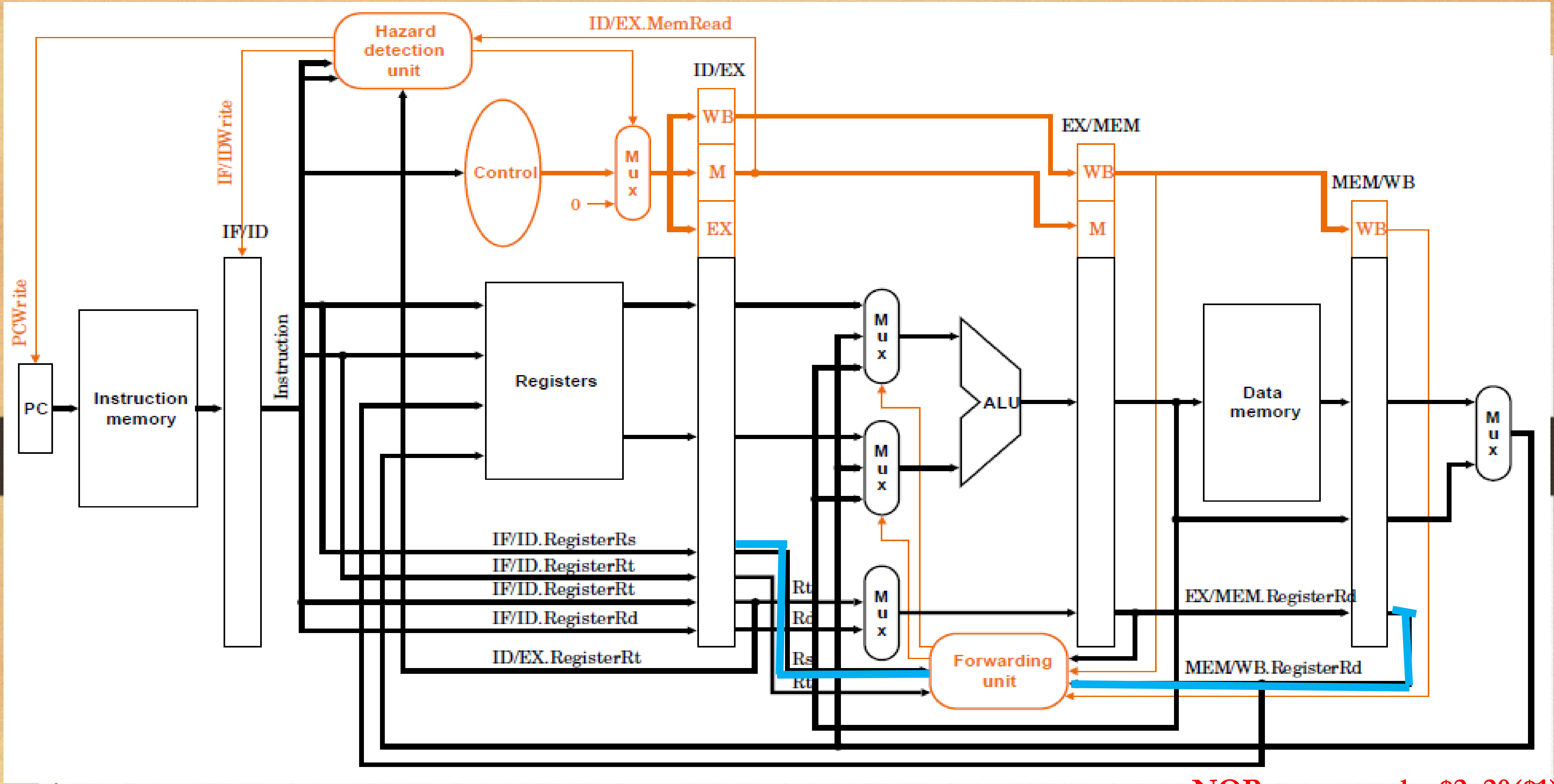
add $9, $4 , $2   or $8, $2 , $6   and $4, $2, $5   NOP   lw $2, 20($1)

Forwarding Hardware will take care of this



add \$9, \$4 , \$2 | or \$8, \$2 , \$6 | and \$4, \$2, \$5 | NOP | lw \$2, 20(\$1)

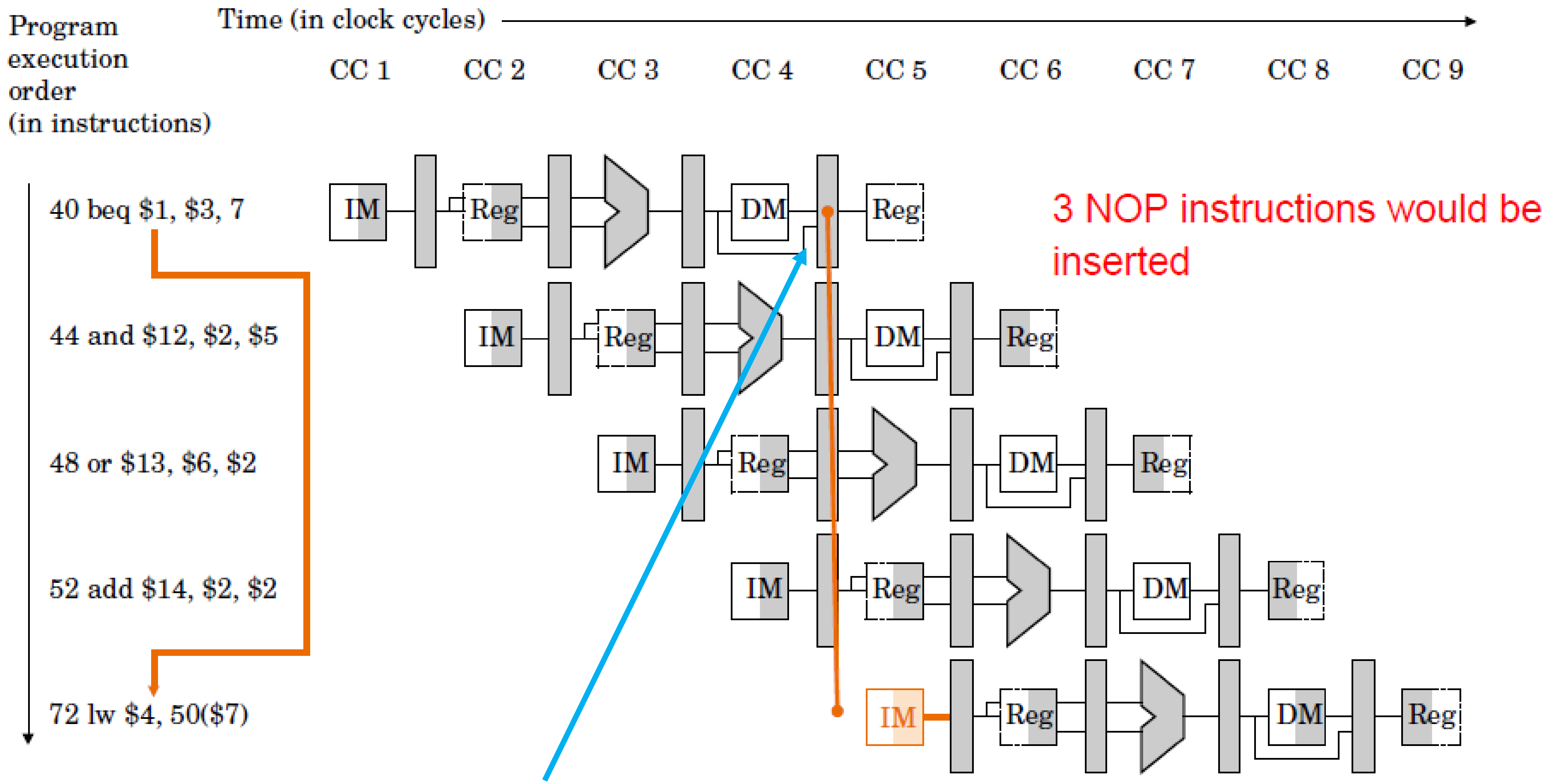
IS the OR instruction okay? Will there be forwarding needed?



add $9, $4 , $2 | or $8, $2 , $6 | and $4, $2, $5 | NOP | lw $2, 20($1)

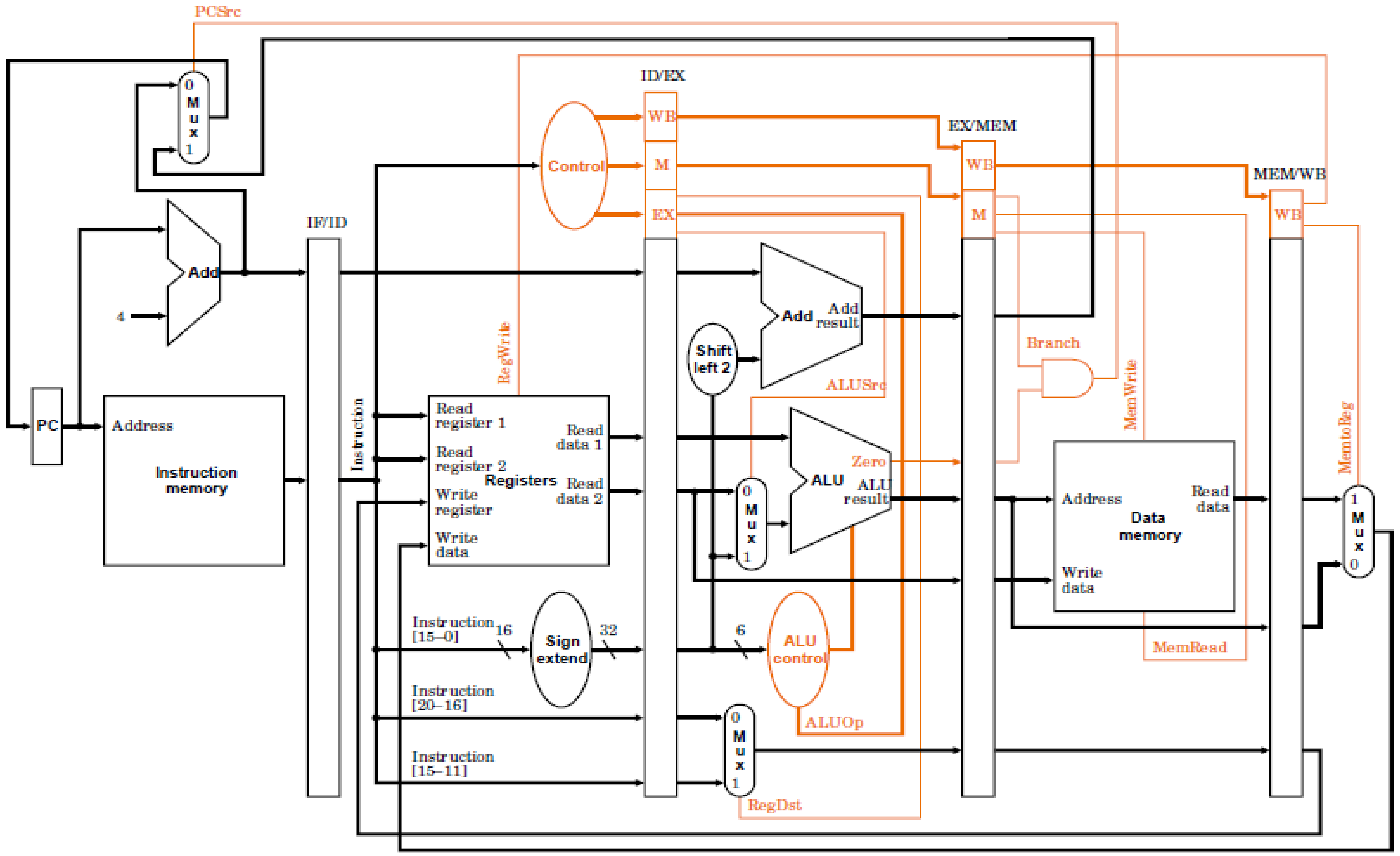
The OR instruction is okay. Will there be forwarding needed? (A) Yes (B) NO

Branch Hazards



3 NOP instructions would be inserted

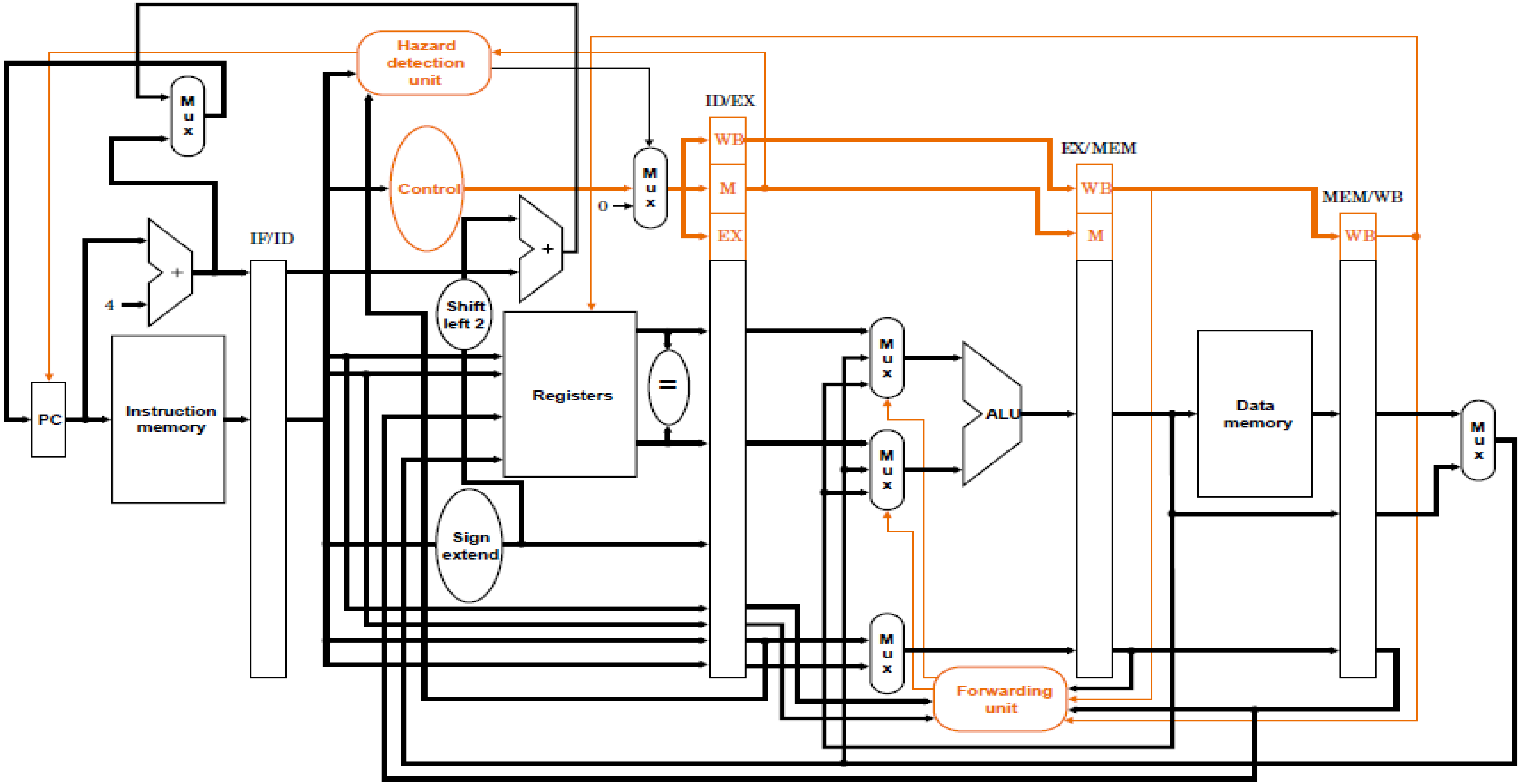
PC is updated at the end of 4th clock cycle. Therefore 3 instructions began that were not needed


PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
WB
M
WB
Mux
0
1
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
Add
Add result
Shift left 2
ALUSrc
ALU
Zero
ALU result
Branch
MemWrite
MemtoReg
Address
Read data
Data memory
Write data
MemRead
Instruction [15–0]
16
Sign extend
32
6
ALU control
ALUOp
Instruction [20–16]
Instruction [15–11]
RegDst

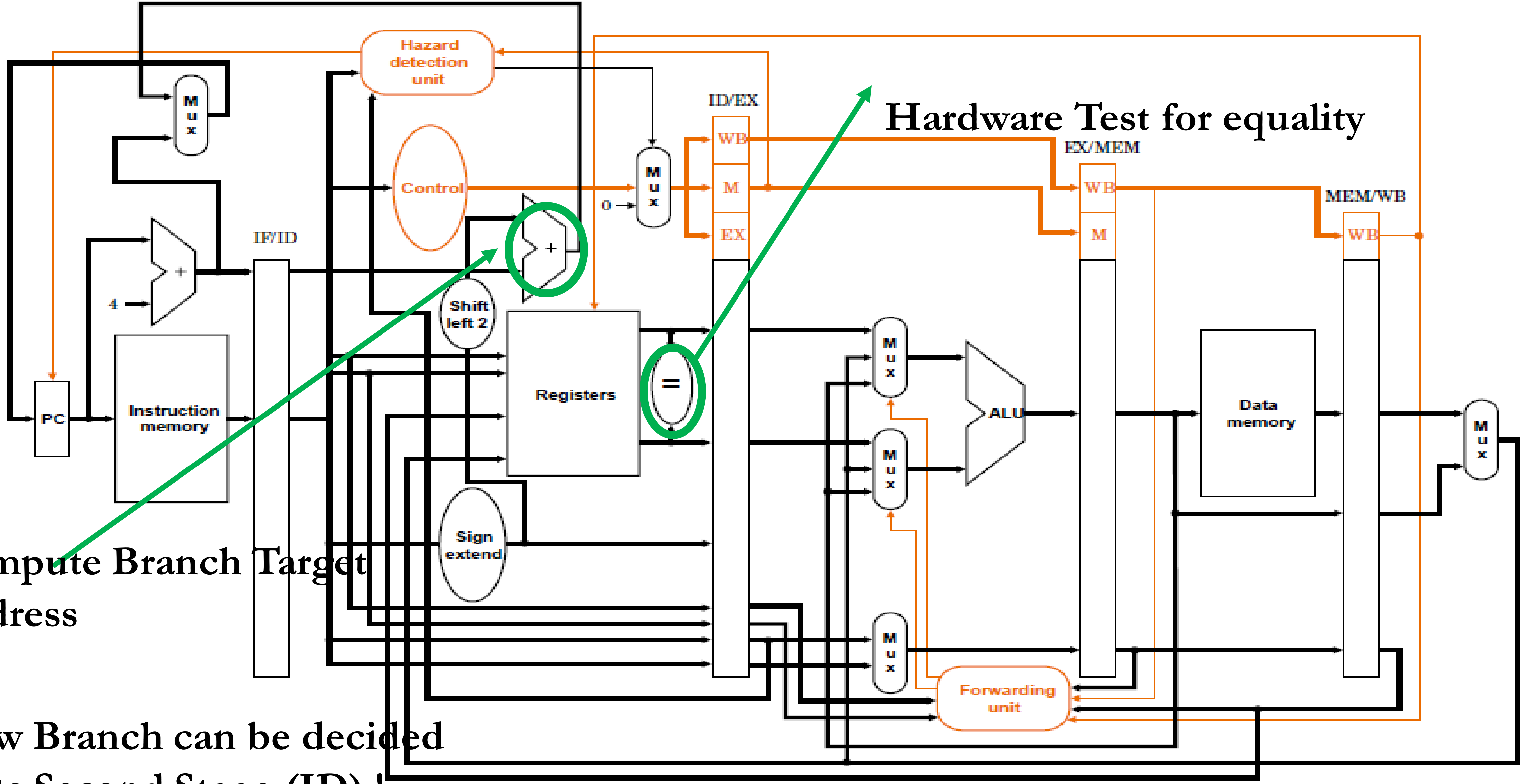
Branching Earlier in Pipeline

- Branch success decided in MEM stage
- At this point instructions in IF, ID, EX stages must be flushed
- Can move branch execution earlier, to ID stage
- Easy to move branch address calculation to ID stage
- Need to move equality test to ID stage, avoid ALU

Hardware to Execute Branch in ID Stage



Hardware to Execute Branch in ID Stage

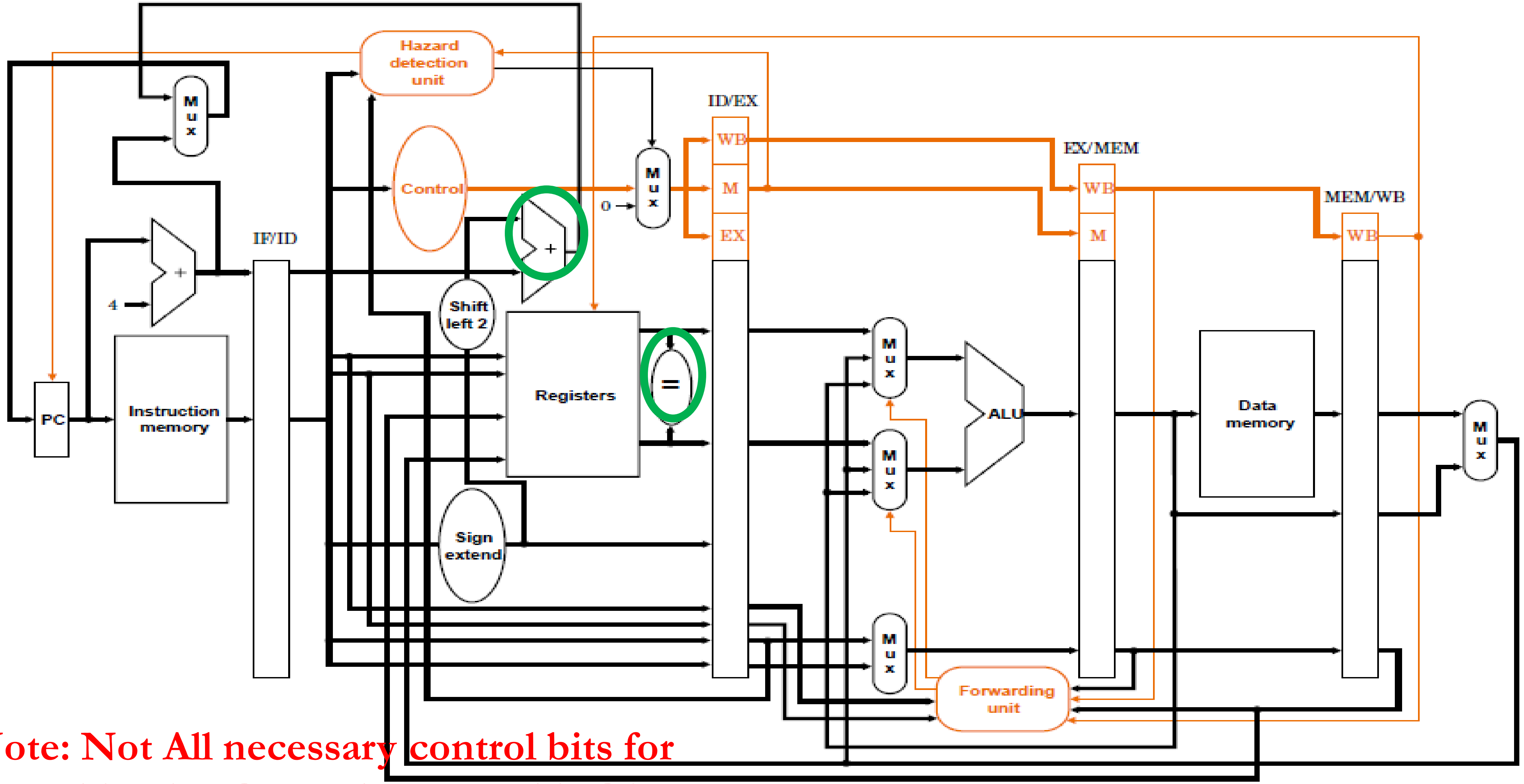


Hardware Test for equality

Compute Branch Target Address

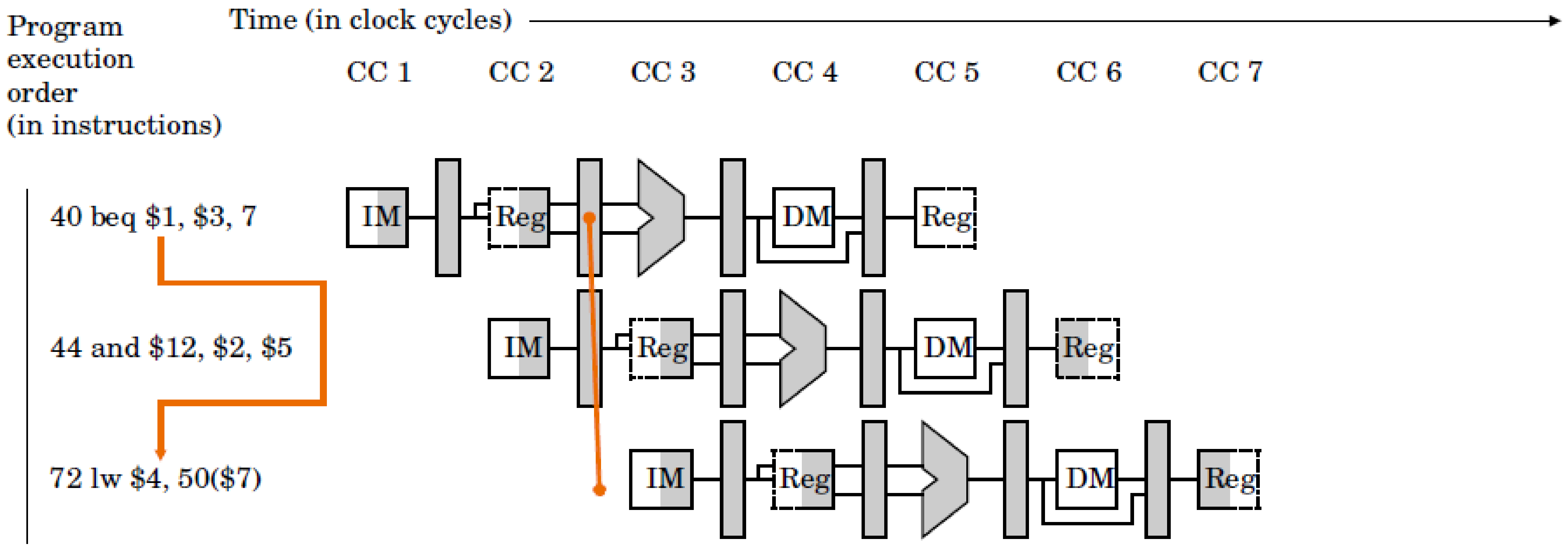
Now Branch can be decided in its Second Stage (ID) ! Great !

Hardware to Execute Branch in ID Stage



Note: Not All necessary control bits for Branching in ID are shown In this Diagram

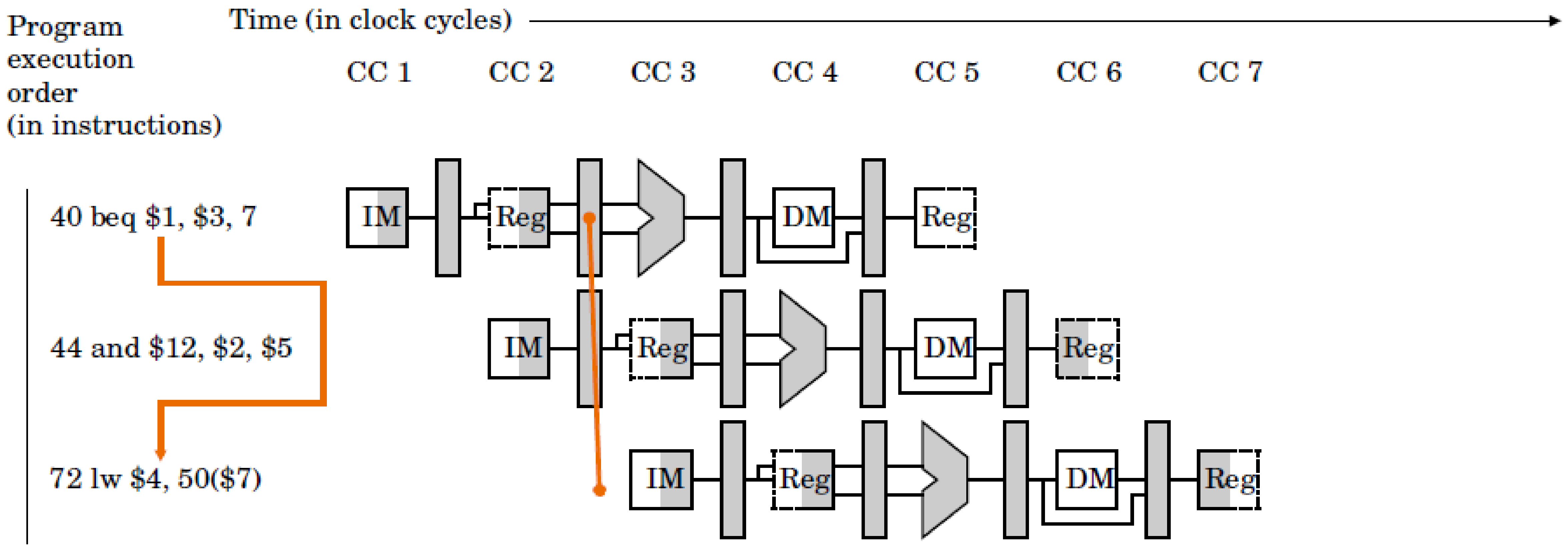
Branch Hazards



Note: this figure is not in the text.

If branch taken, should Instruction 44 be executed?

Branch Hazards



Note: this figure is not in the text.

If branch taken, should Instruction 44 be executed?

Instruction at 44 will have started: Now it will need to be Flushed in the Datapath. Ie) We do not want it to complete

Branch Flushing

- After moving Branch up to ID stage, instruction after branch will still always execute.

  How to handle this (possibly) errant instruction?

(1) • MIPS forces compiler to handle errant instruction

  Code rearrangement, NOP

(2) • Alternatively, use Branch Flushing

  Zeroing control bits of instruction in IF/ID register will flush IF stage

We will do (1) OR (2) to deal with Errant instruction
Not Both!

Branch Flushing

- After moving Branch up to ID stage, instruction after branch will still always execute.

  How to handle this (possibly) errant instruction?

(1) - MIPS forces compiler to handle errant instruction

  Code rearrangement, NOP

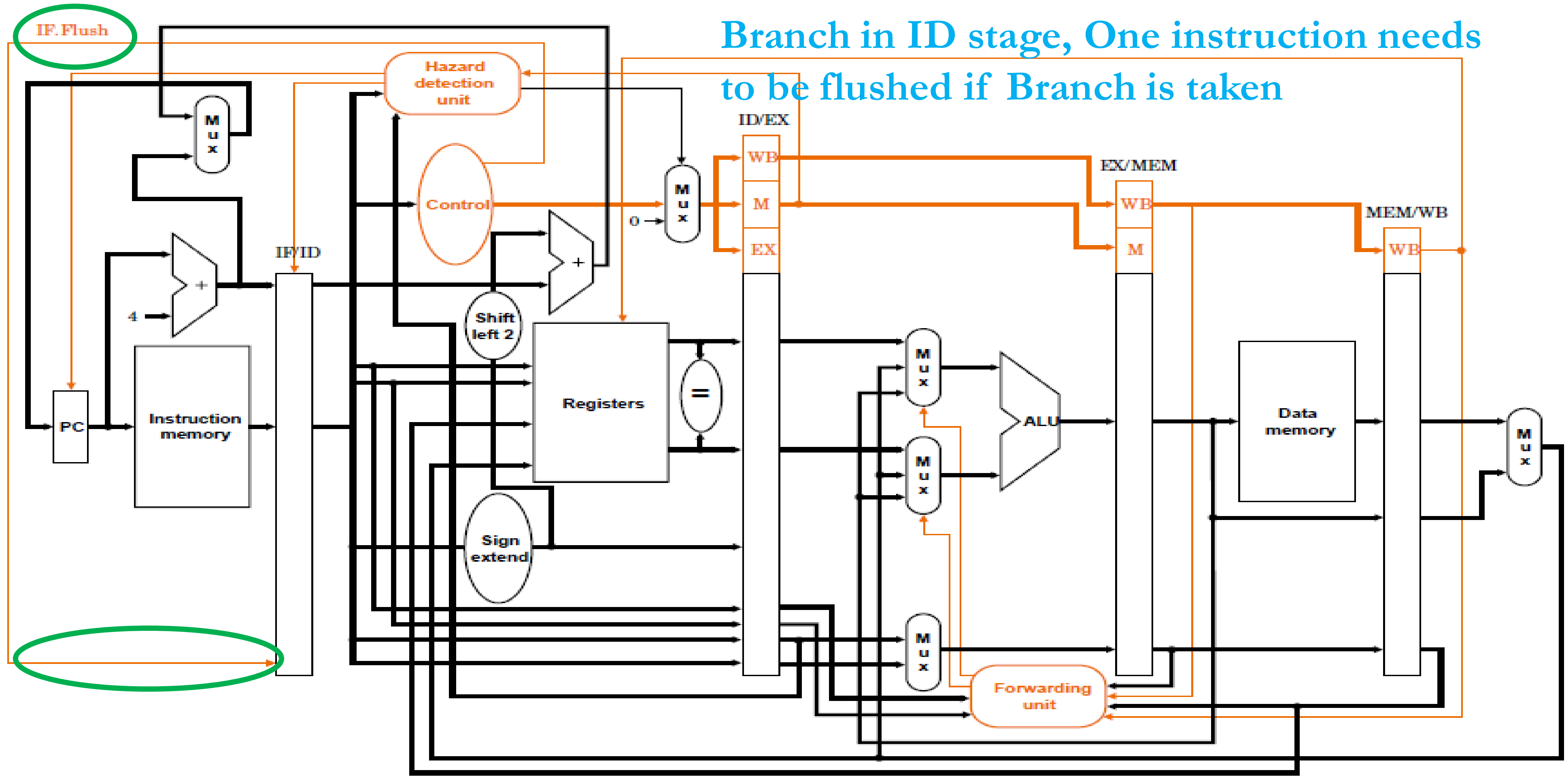
(2) - Alternatively, use Branch Flushing

  Zeroing control bits of instruction in IF/ID register will flush IF stage

Works similarly if we had 3 Errant instructions in datapath, They would all need to be Flushed

Hardware for Branch Flushing

Branch in ID stage, One instruction needs to be flushed if Branch is taken



The instruction that was just fetched: Clear out all of the Instruction Bits in IF/ID