

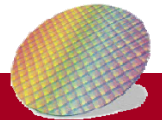


成功大學

National Cheng Kung University

---

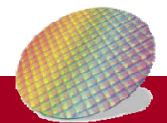
# Control Signal for Pipeline Processor



# Outline

---

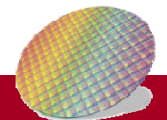
- A pipelined datapath
- **Pipelined control**
- Data hazards and forwarding
- Data hazards and stalls
- Branch hazards



# Pipeline Control

---

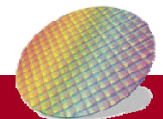
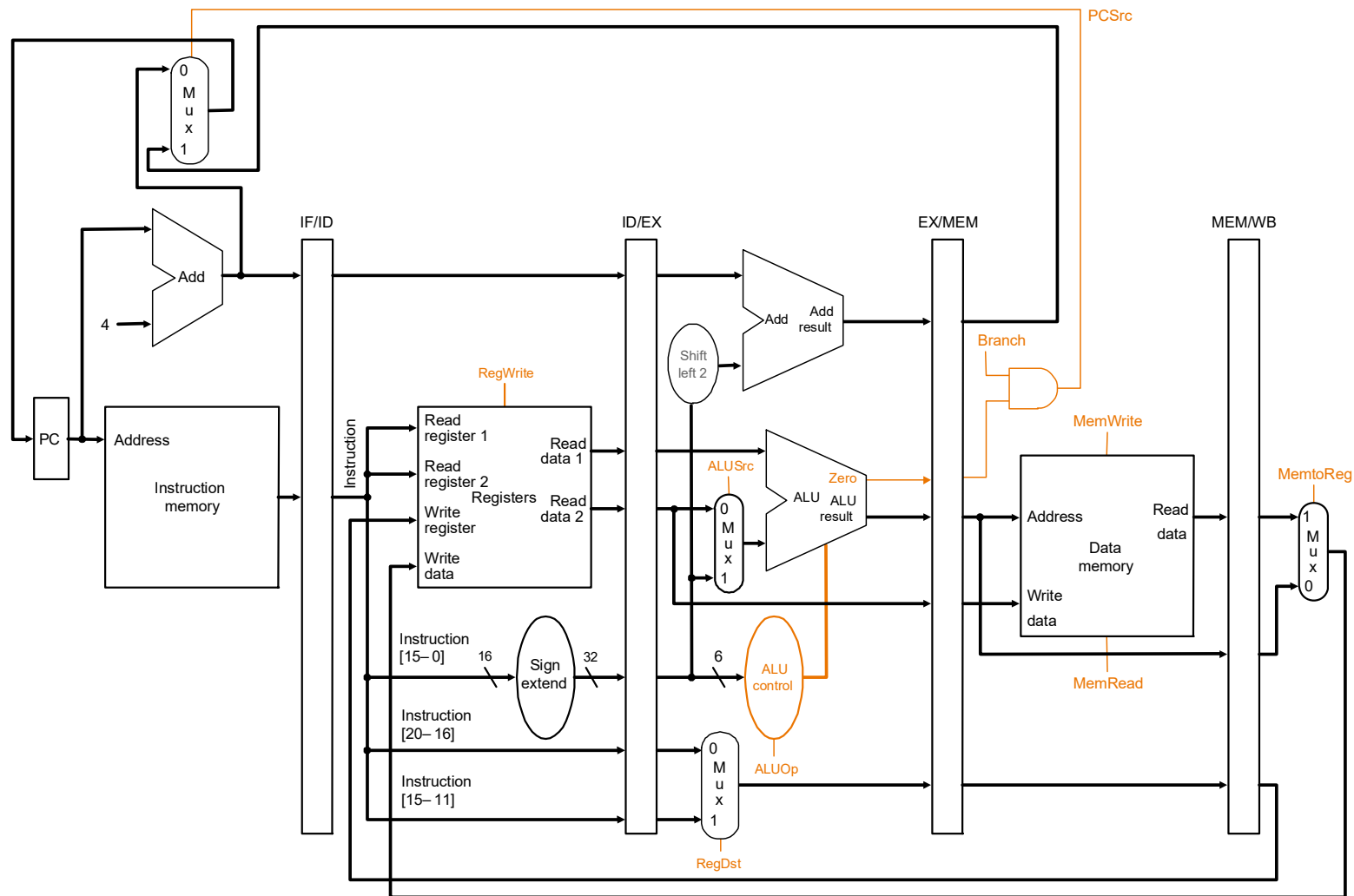
- Three steps are needed
  - Step 1: Begin with the same control signal with **single-cycle** datapath control
  - Group control lines into **five** groups according to pipeline stage
    - Since **control signals** are associated with **components** active during a single pipeline stage
  - **Set** control signals during **each** pipeline stage





# Pipelined Datapath with Control – Step 1

Start with Same control signals as the single-cycle datapath



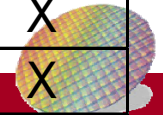
# Pipeline Control Signals- Step 2

- Group control lines into **five** groups according to pipeline stage
  - instruction fetch / PC increment (IF)
  - instruction decode / register fetch (ID)
  - execution / address calculation (**EX**)
  - memory access (**M**)
  - write back (**WB**)

} **Nothing to control as instruction memory read and PC write are always enabled**

} **(the following table)**

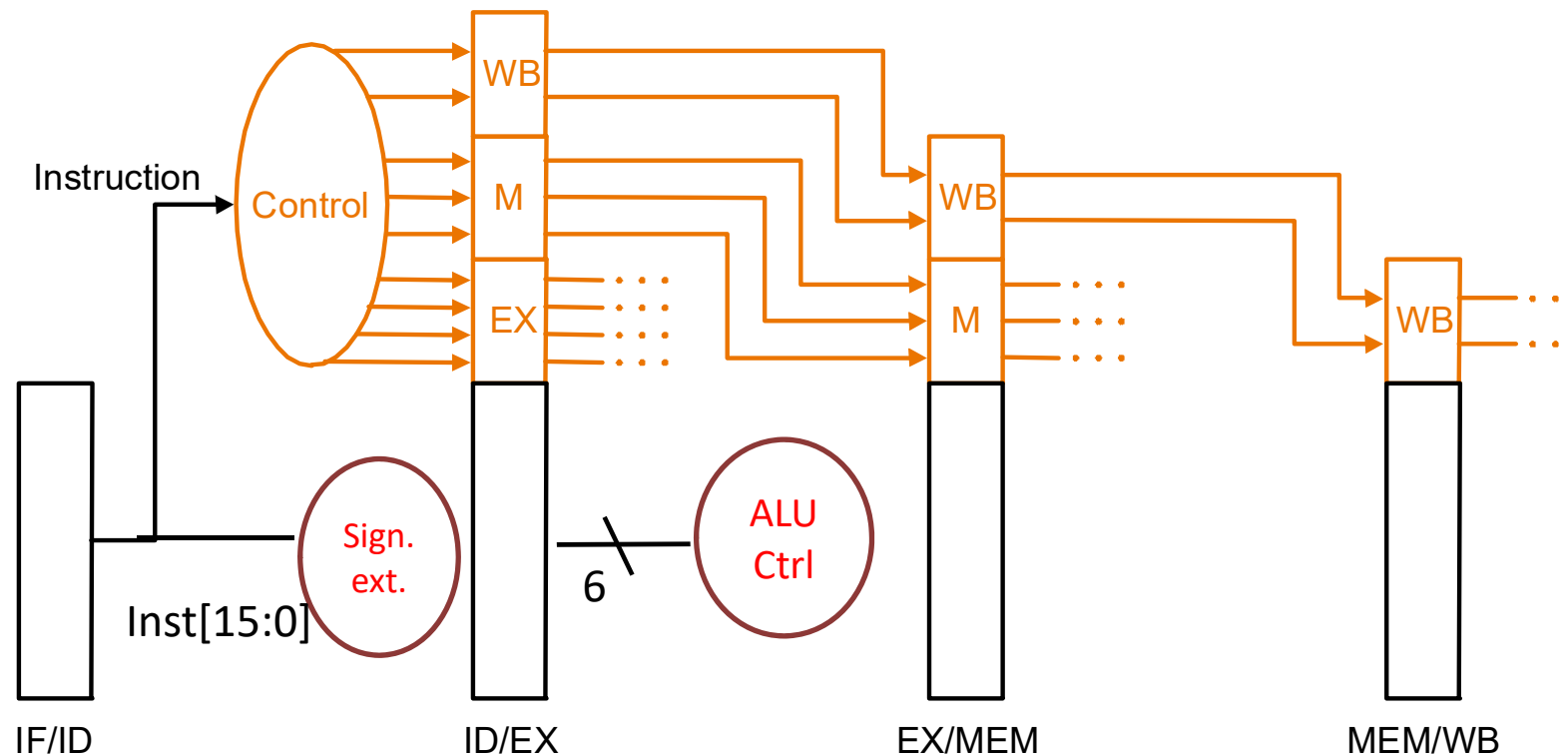
Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X



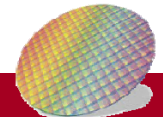


# Pipeline Control Implementation

- *Pass control signals along just like the data* – extend each pipeline register to hold needed control bits for succeeding stages

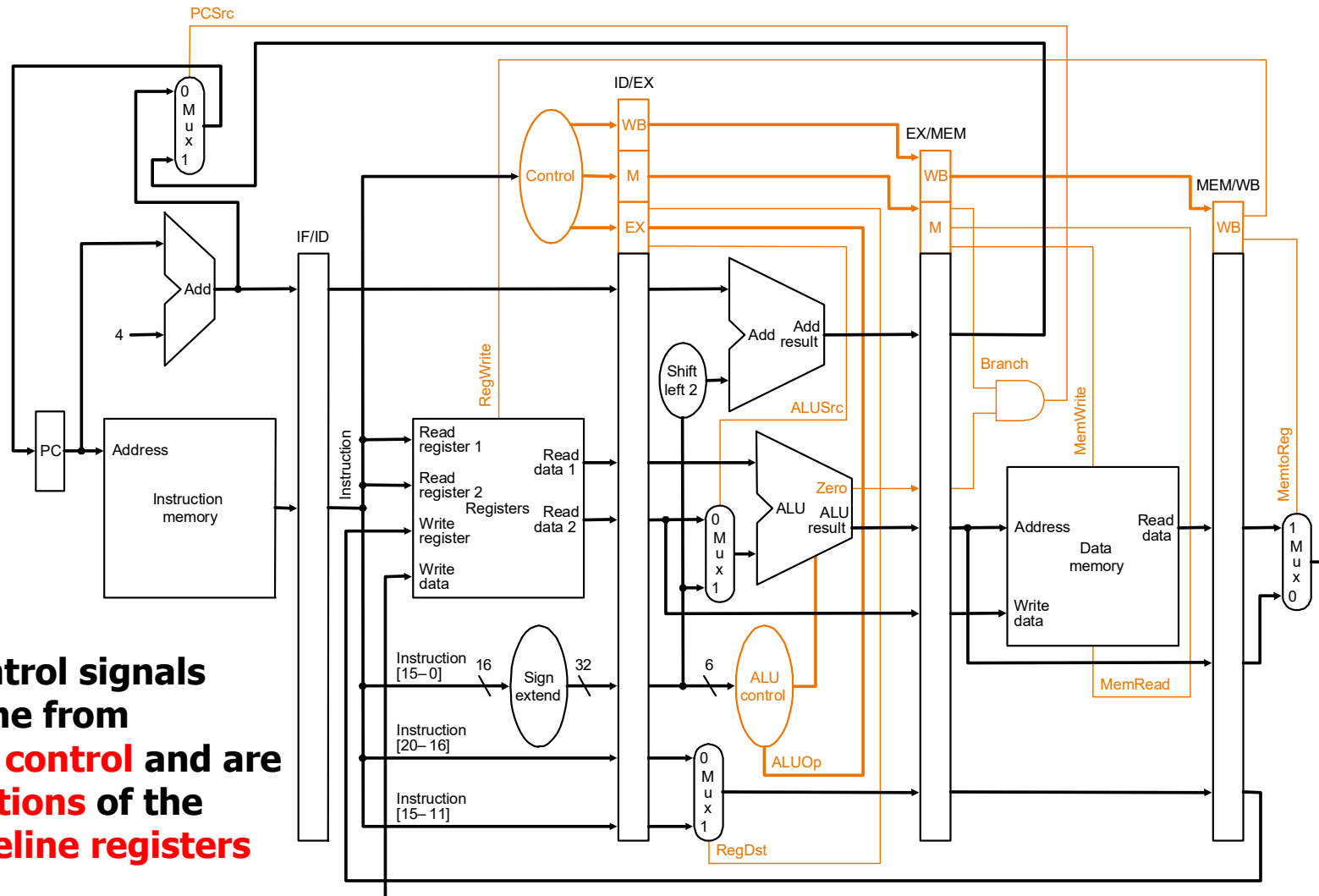


- *Note:* The 6-bit *funct* field of the instruction required in the EX stage to generate ALU control can be retrieved as the 6 least significant bits of the immediate field which is sign-extended and passed from the *IF/ID* register to the *ID/EX* register

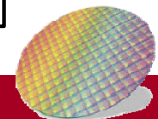




# Pipelined Datapath with Control – Step 3



Control signals come from the **control** and are **portions of the pipeline registers**





## An Example

---

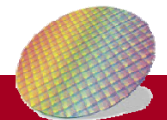
`lw $10, 20($1)`

`sub $11, $2, $3`

`and $12, $4, $5`

`or $13, $6, $7`

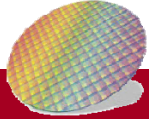
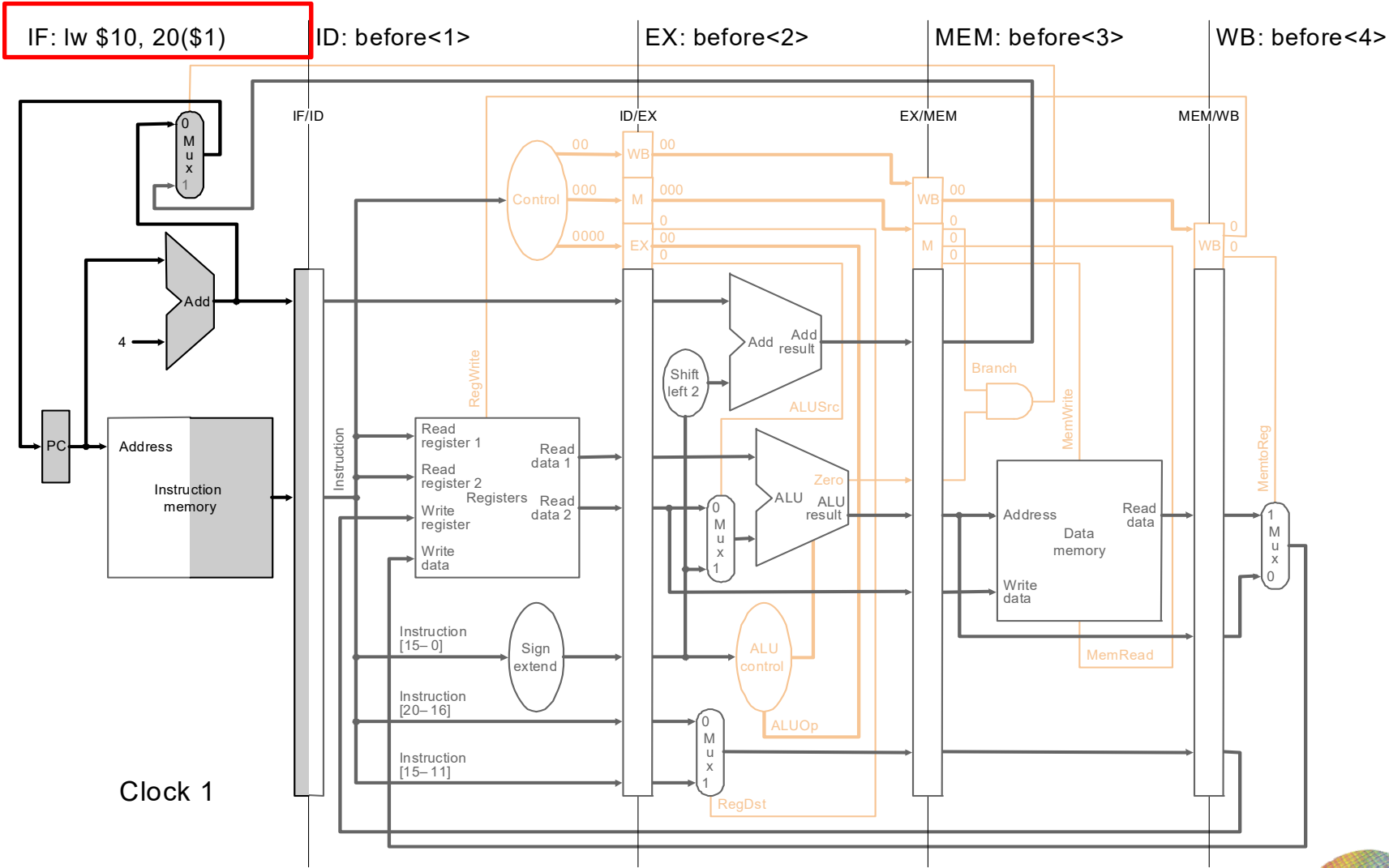
`add $14, $8, $9`





# Cycle 1

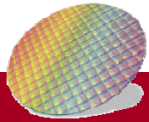
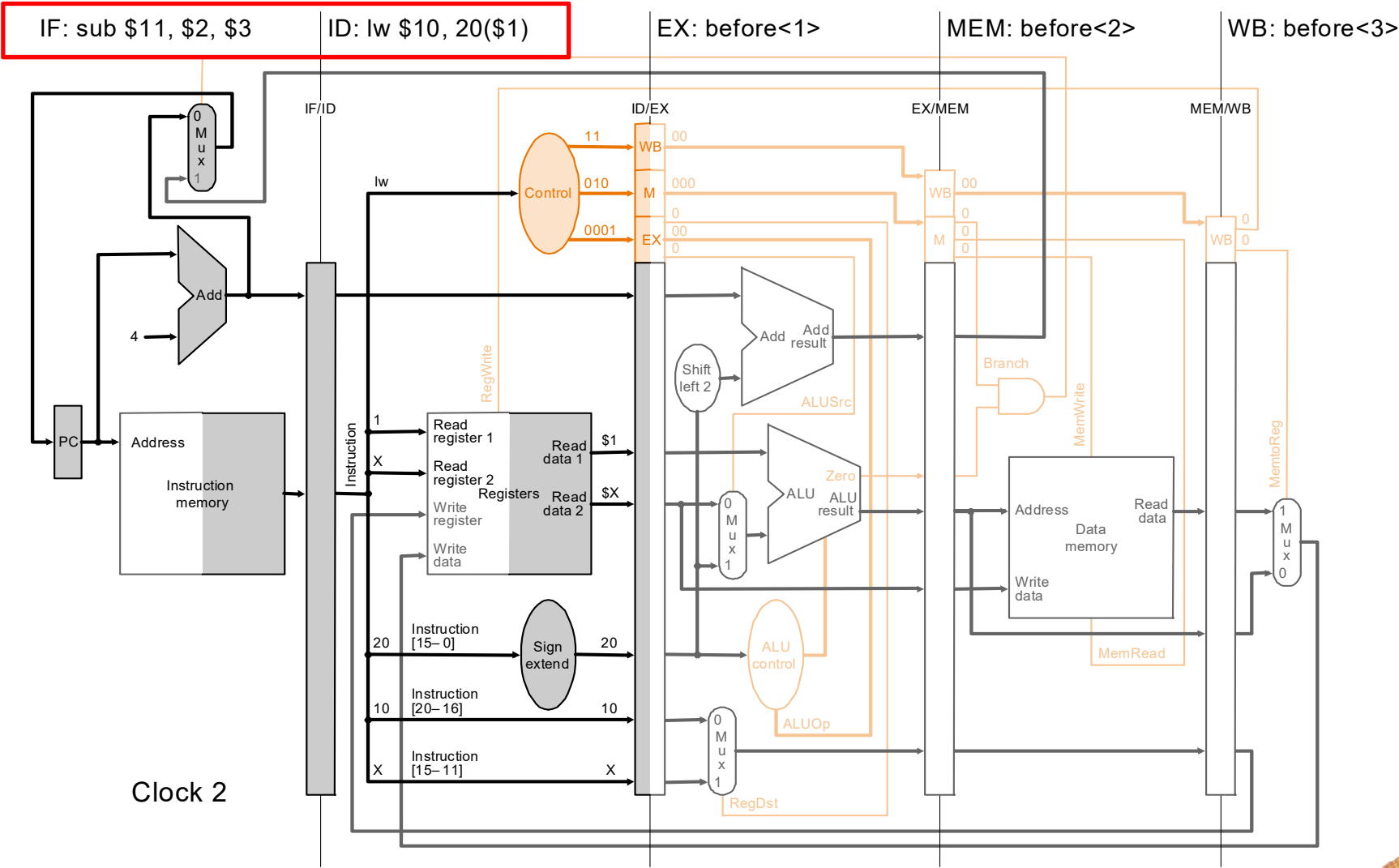
```
lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
```



# Cycle 2

```

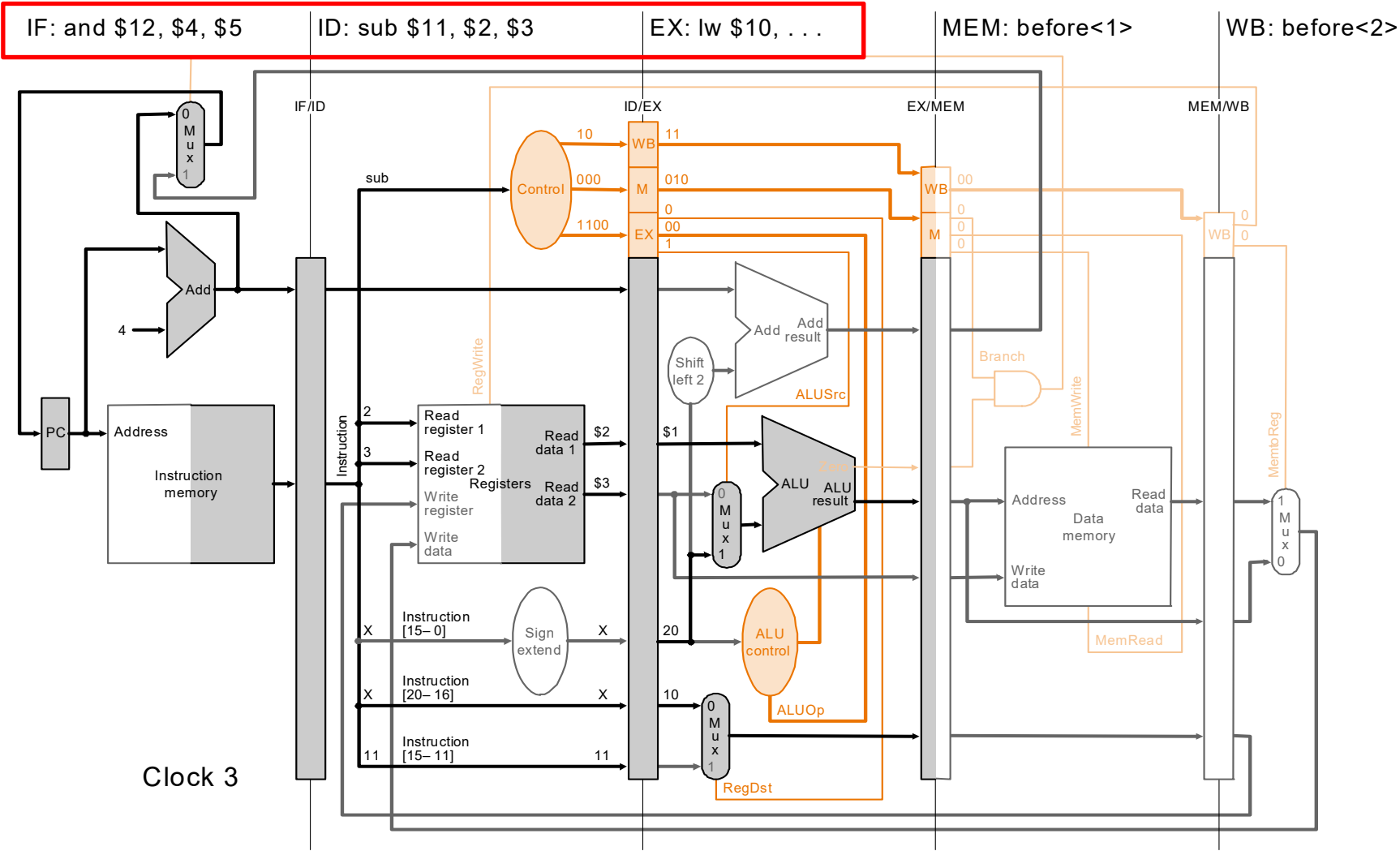
lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```



# Cycle 3

```

lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```

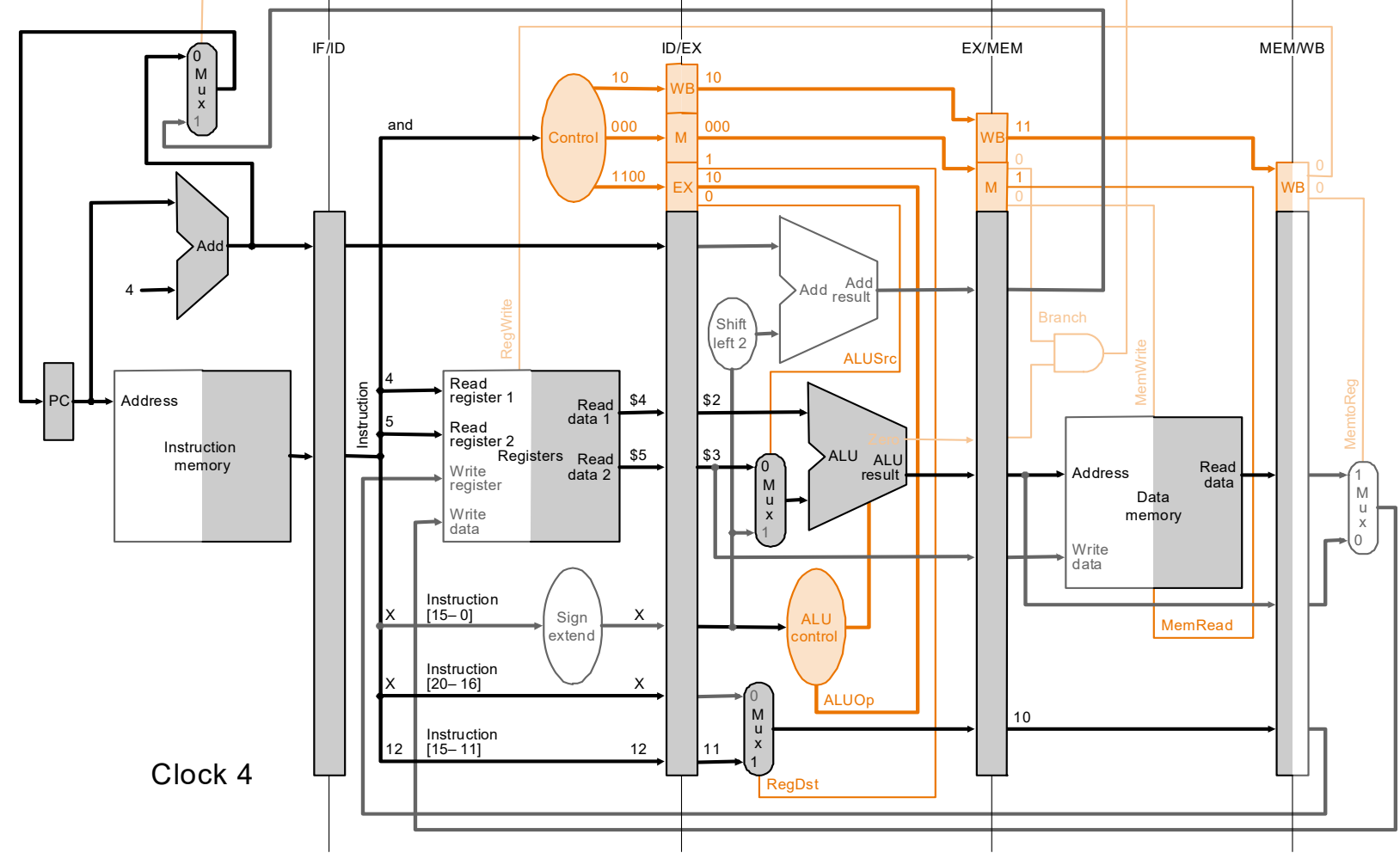


# Cycle 4

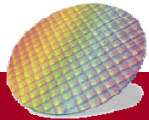
```

lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```

IF: or \$13, \$6, \$7    ID: and \$12, \$2, \$3    EX: sub \$11, ...    MEM: lw \$10, ...    WB: before <1>



Clock 4

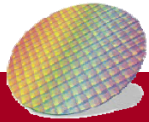
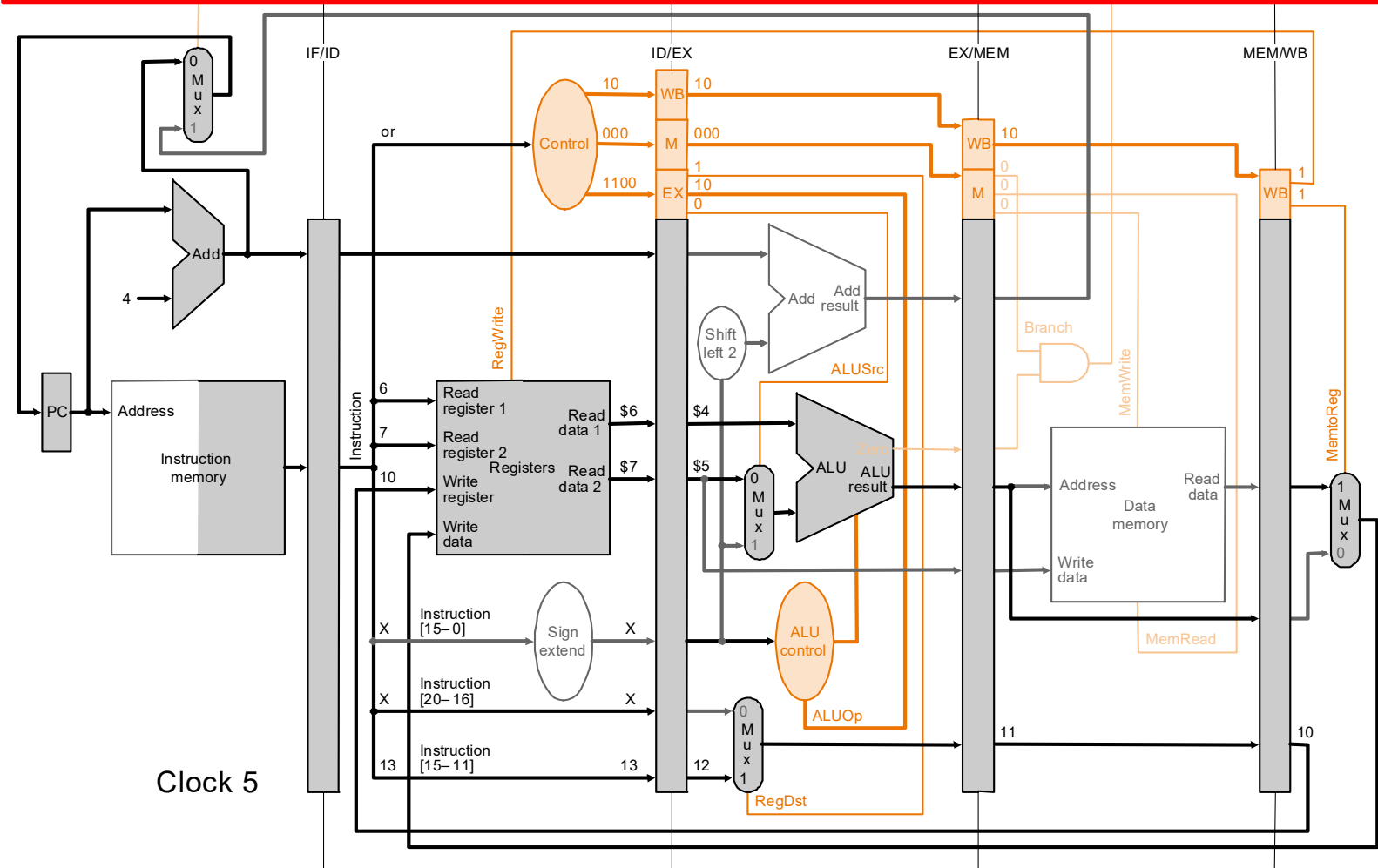


# Cycle 5

```

lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```

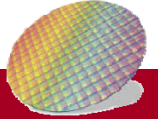
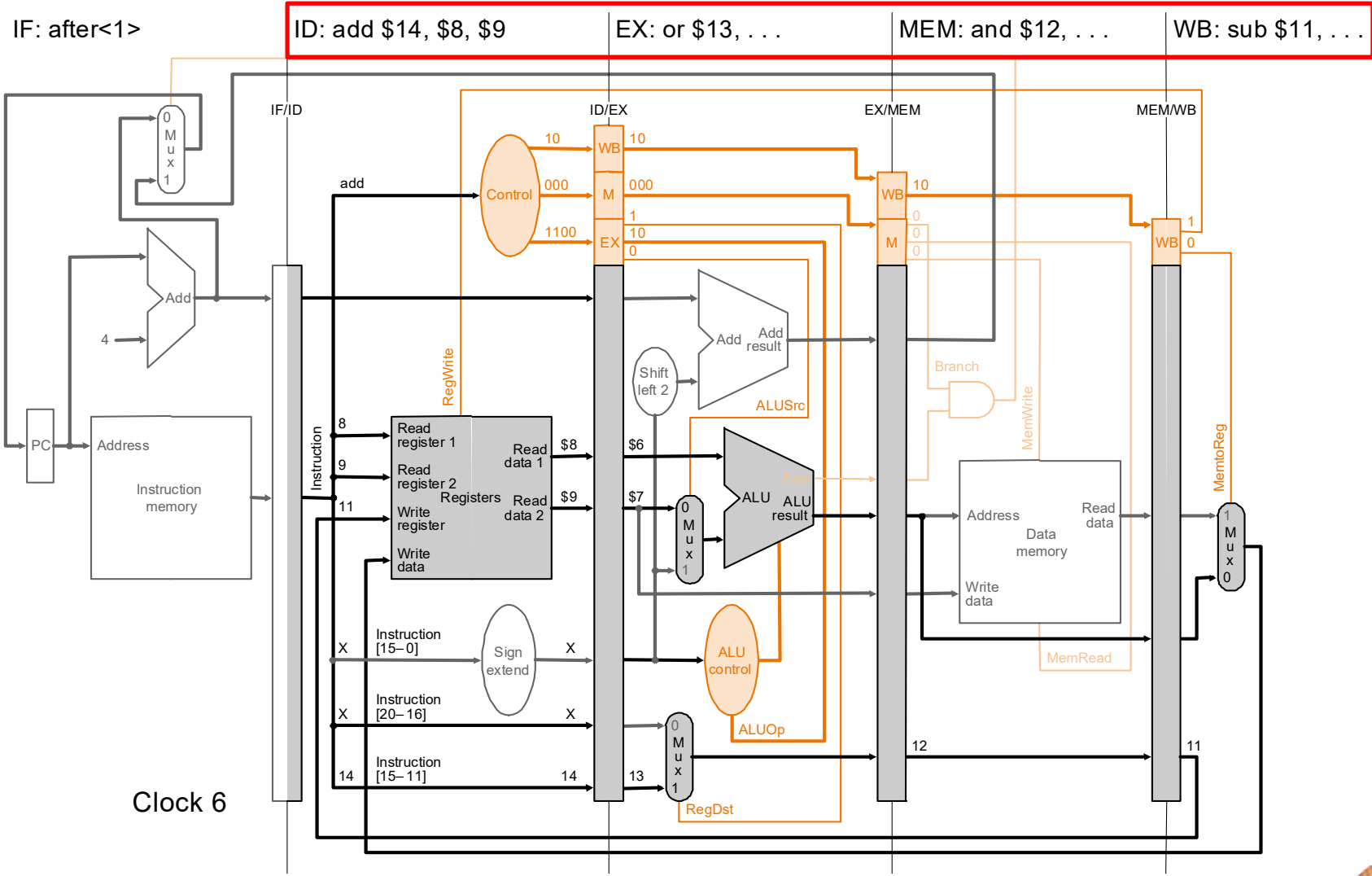
IF: add \$14, \$8, \$9      ID: or \$13, \$6, \$7      EX: and \$12, ...      MEM: sub \$11, ...      WB: lw \$10, ...



# Cycle 6

```

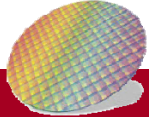
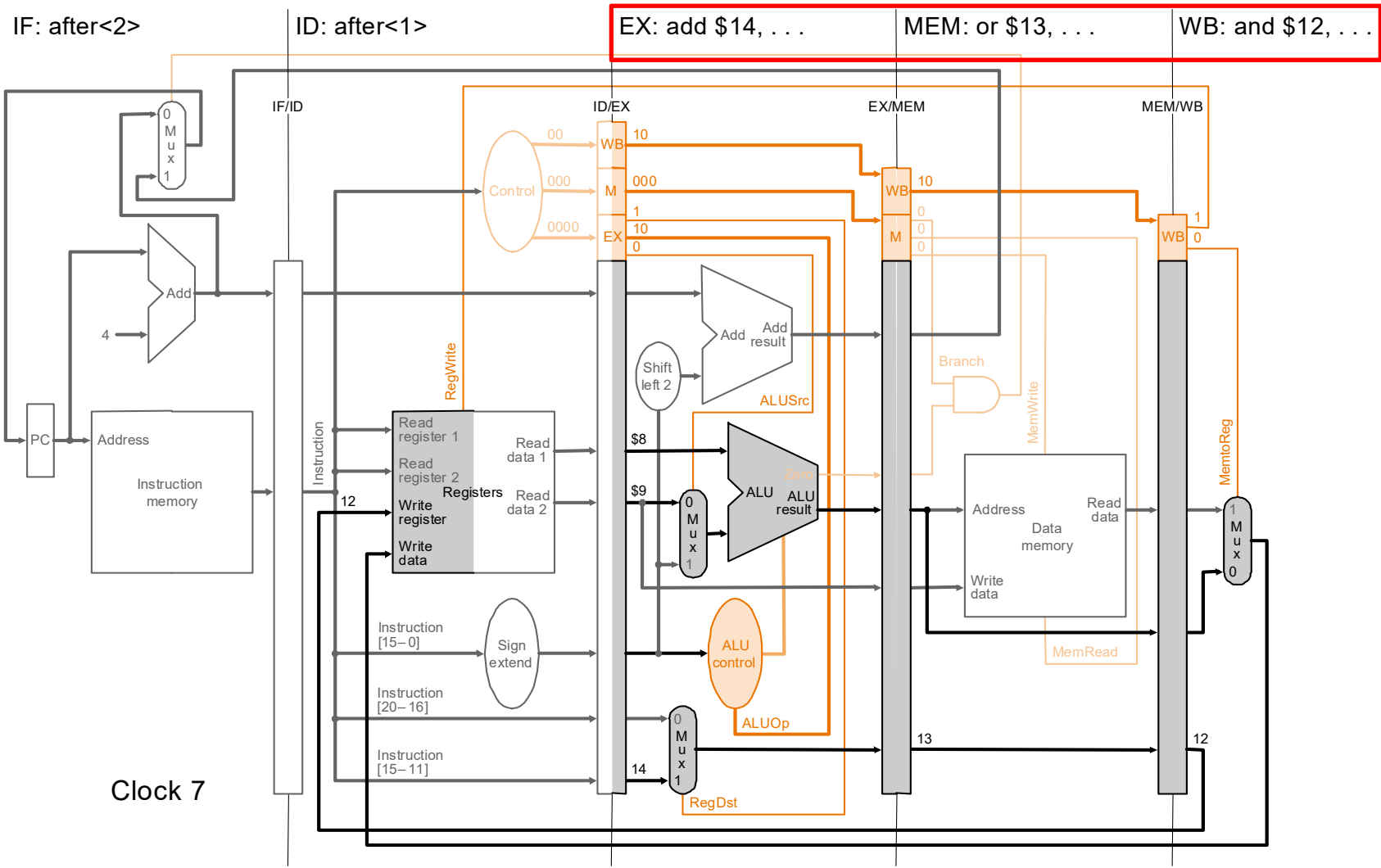
lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```



# Cycle 7

```

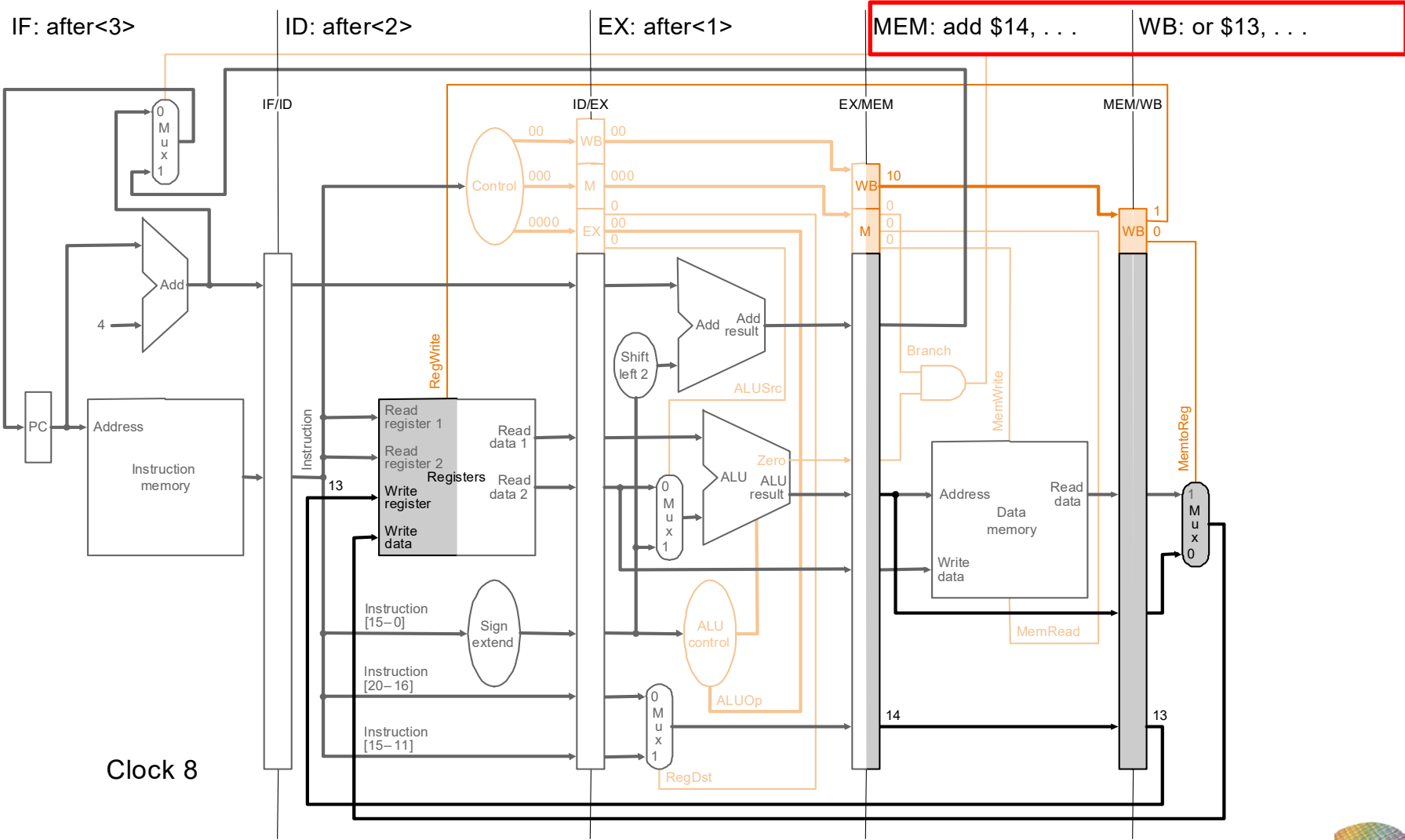
lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```



# Cycle 8

```

lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```

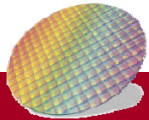
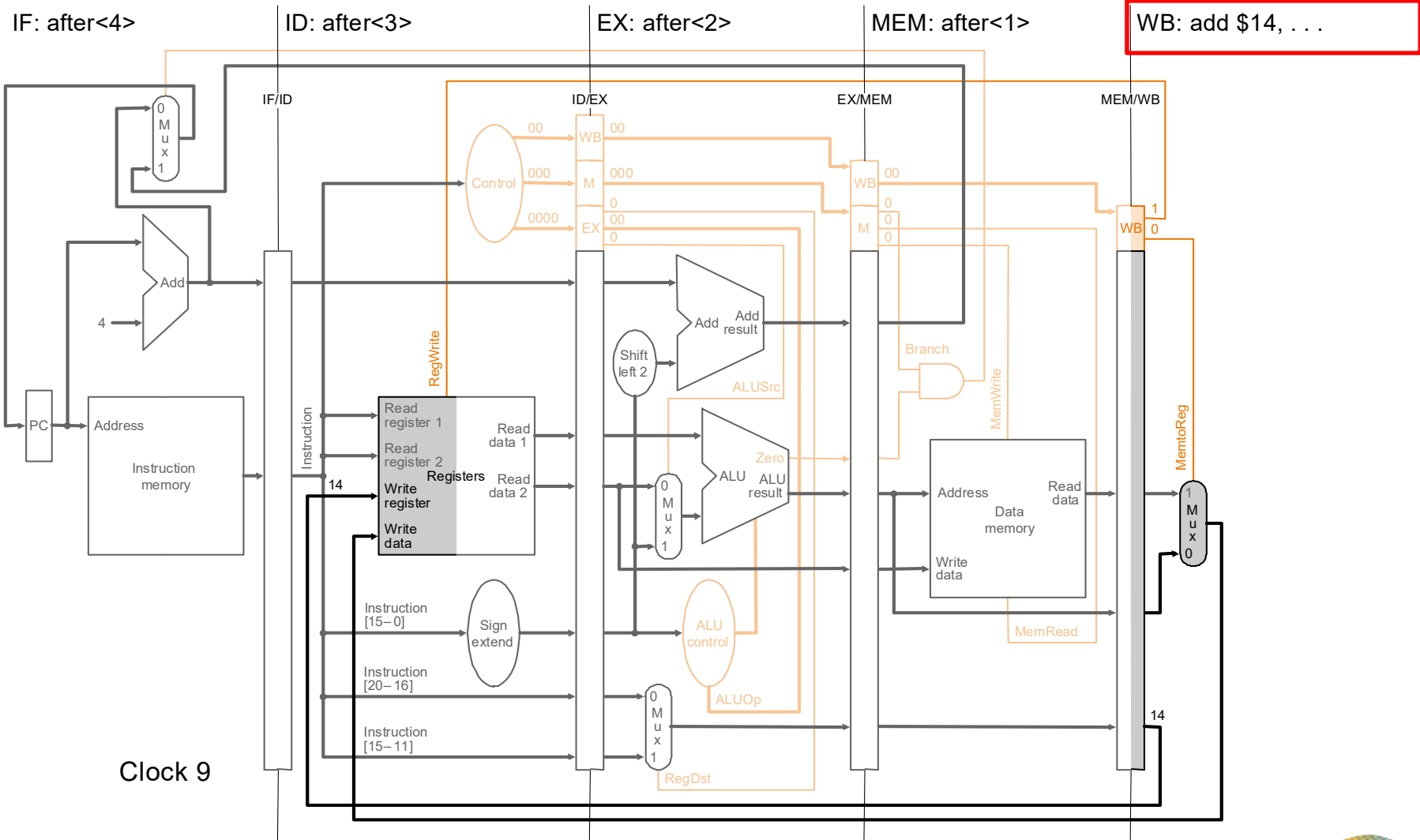




# Cycle 9

```

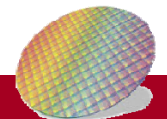
lw $10, 20($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9
    
```





# Outline

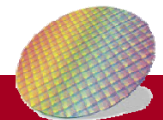
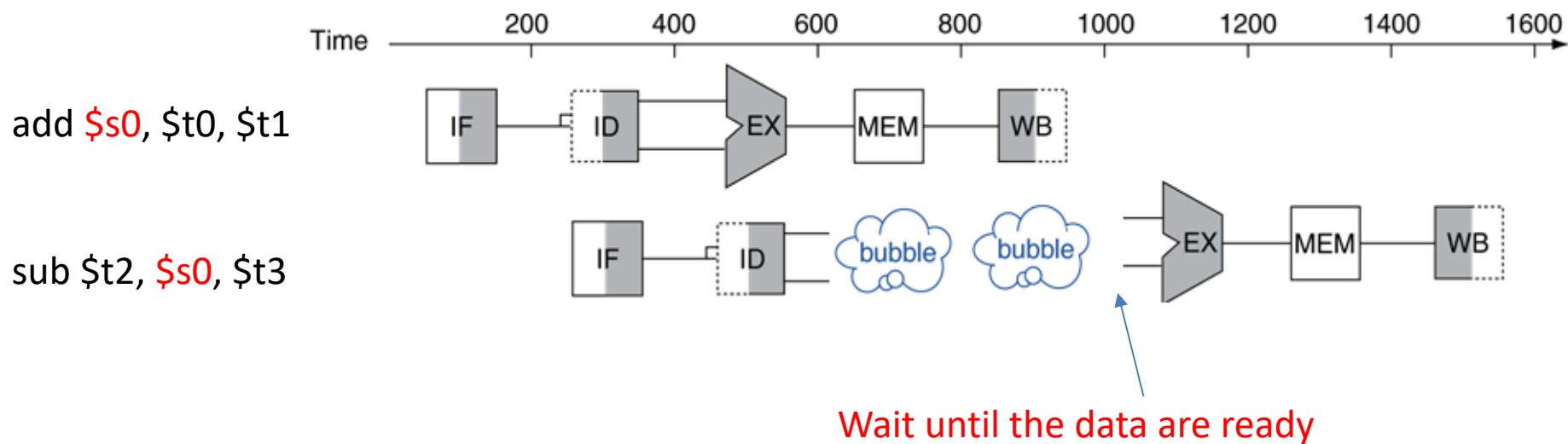
- A pipelined datapath
- Pipelined control
- **Data hazards and forwarding**
- Data hazards and stalls
- Branch hazards





# Pipeline Hazards

- Pipeline Hazards:
  - Structural hazards, Data hazards, Control hazards
- Hazards can be resolved by **waiting (add stalls/bubble)**
  - pipeline control must detect the hazard
  - take action (or delay action) to resolve hazards



# Data Hazards in ALU Instructions

- Consider this sequence:

sub \$2, \$1, \$3

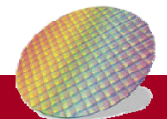
and \$12, \$2, \$5

or \$13, \$6, \$2

add \$14, \$2, \$2

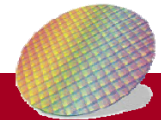
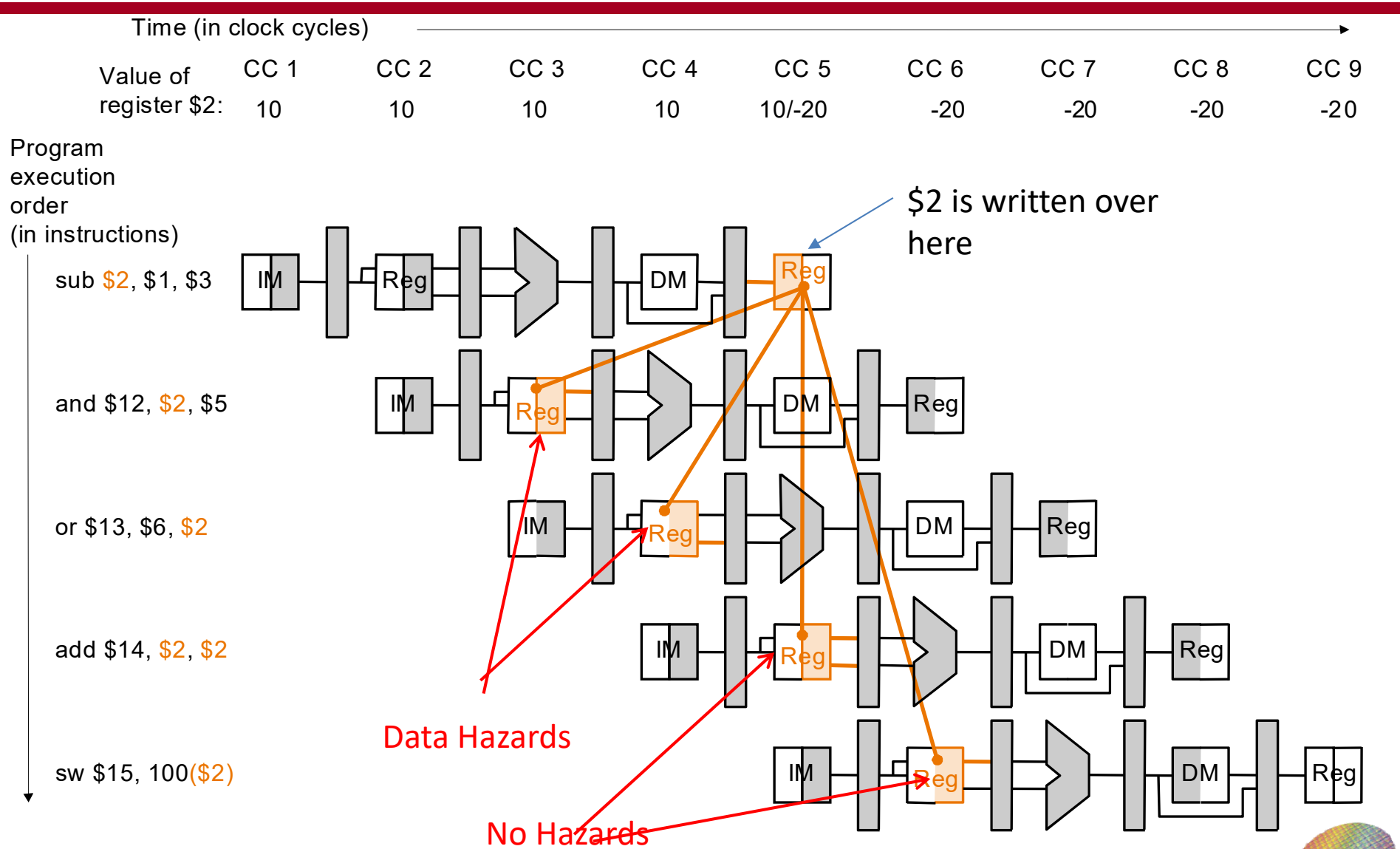
sw \$15, 100(\$2)

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





# Data Hazards





# Three Types of Data Dependency (RAW, WAR, WAW)

- RAW (read after write):

i2 tries to read operand before i1 writes it



*RAW is Data  
Dependence  
also cause data  
hazard*

```

      W R R
sub $2 $1 $3
add $4 $3 $2
      W R R

```

- WAR (write after read):

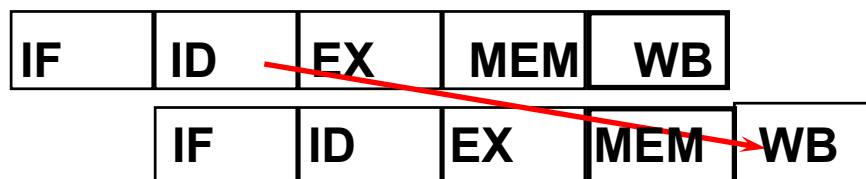
i2 tries to write operand before i1 reads it

```

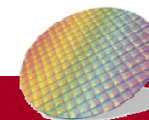
add $4 $2 $3
sub $2 $1 $3

```

- WAR is **not** a issue in MIPS 5-stage pipeline because all instructions take 5 stages, and reads are always in stage 2, and writes are always in stage 5, the following instruction never corrupt the previous instruction



*Do not cause stall !!!  
WAR is not data hazard*





# Types of Data Dependency (RAW, WAR, WAW)

Three types: (inst. **i1** followed by inst. **i2**)

- **WAW (write after write):**

**i2** tries to write operand before **i1** writes it

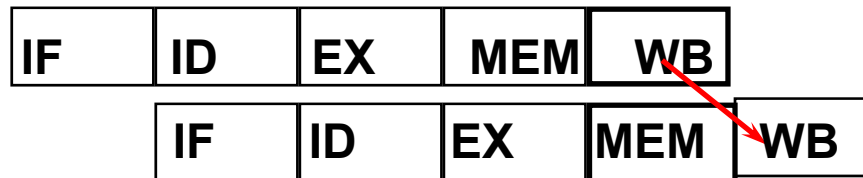
- WAW is not an issue in MIPS 5-stage pipeline because all instructions take 5 stages, and writes are always in stage 5

W R R

sub \$2 \$1 \$3

sub \$2 \$1 \$3

W R R

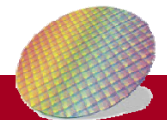


**Do not cause stall !!!**  
**WAW is not data hazard**

Quick summary:

Three data dependency: **RAW**, **WAR**, **WAW**

only **RAW** may cause data hazard in MIPS





## Exercise

- Identify the data dependency (RAW, WAW, WAR) in the following instruction

lw \$1,40(\$6)  
add \$6, \$2, \$2  
sw \$6, 50(\$1)

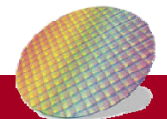
Ans:

lw \$1,40(\$6)  
W R  
add \$6, \$2, \$2  
W R R  
sw \$6, 50(\$1)  
R R

I1 to I2: WAR on \$6

I1 to I3: RAW on \$1

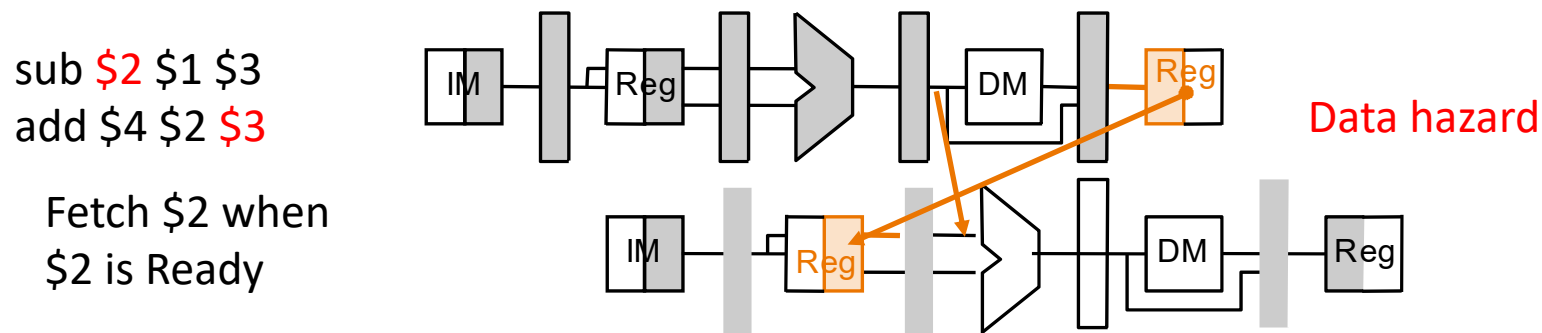
I2 to I3: RAW on \$6





# Hardware Solution: Forwarding

- Idea: fetch “fresh” data as early as possible



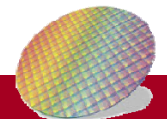
- Two steps:

Step 1: **Detect** data hazard:

Is the datum just produced required by the following inst.?

Step 2: **Forward** intermediate data to resolve hazard

If yes, then forward the requested datum to the requesting inst. immediately.





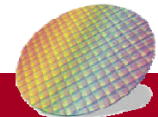
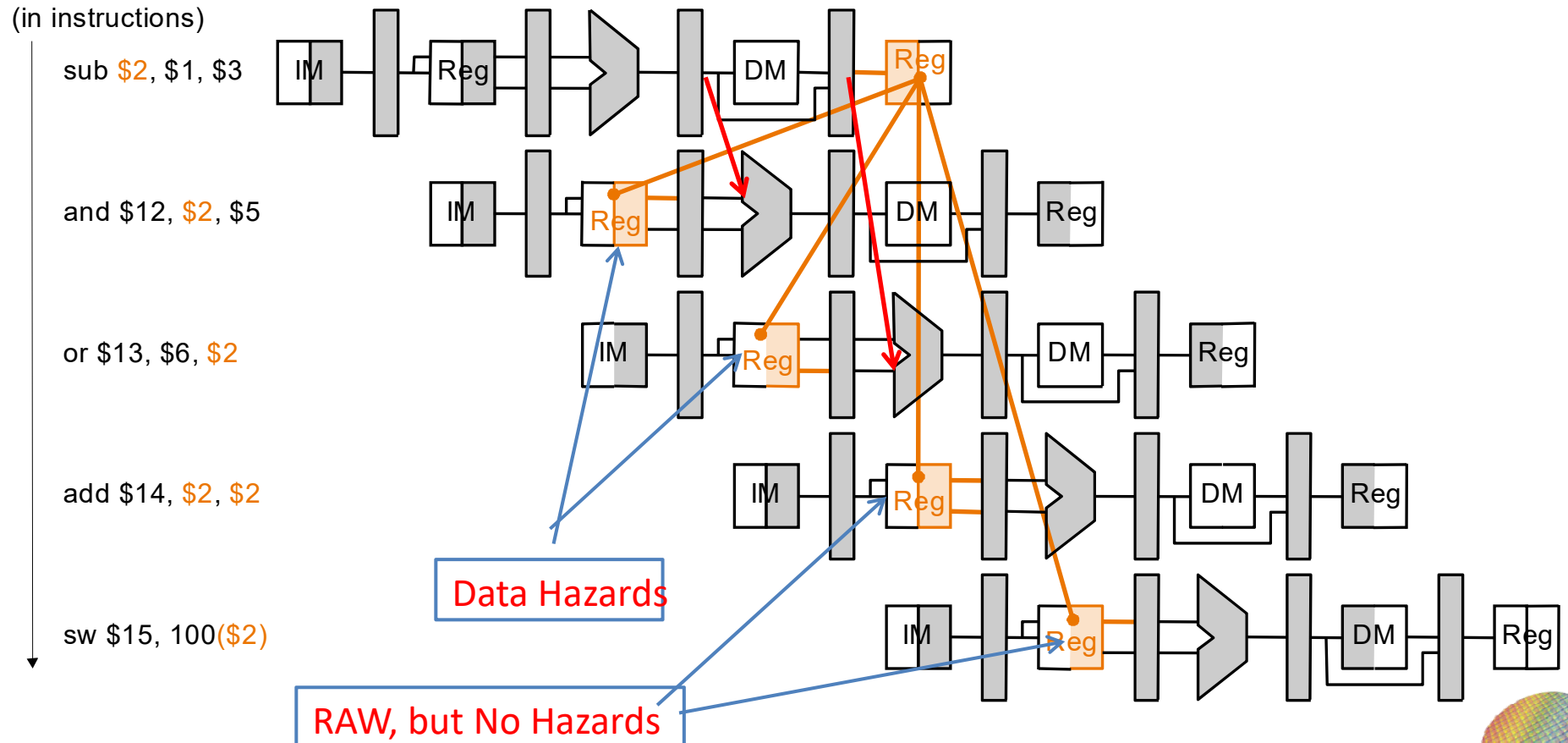
# Data Hazards

Time (in clock cycles)

Value of register \$2:	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
	10	10	10	10	10/ -20	-20	-20	-20	-20

Program execution order (in instructions)

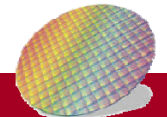
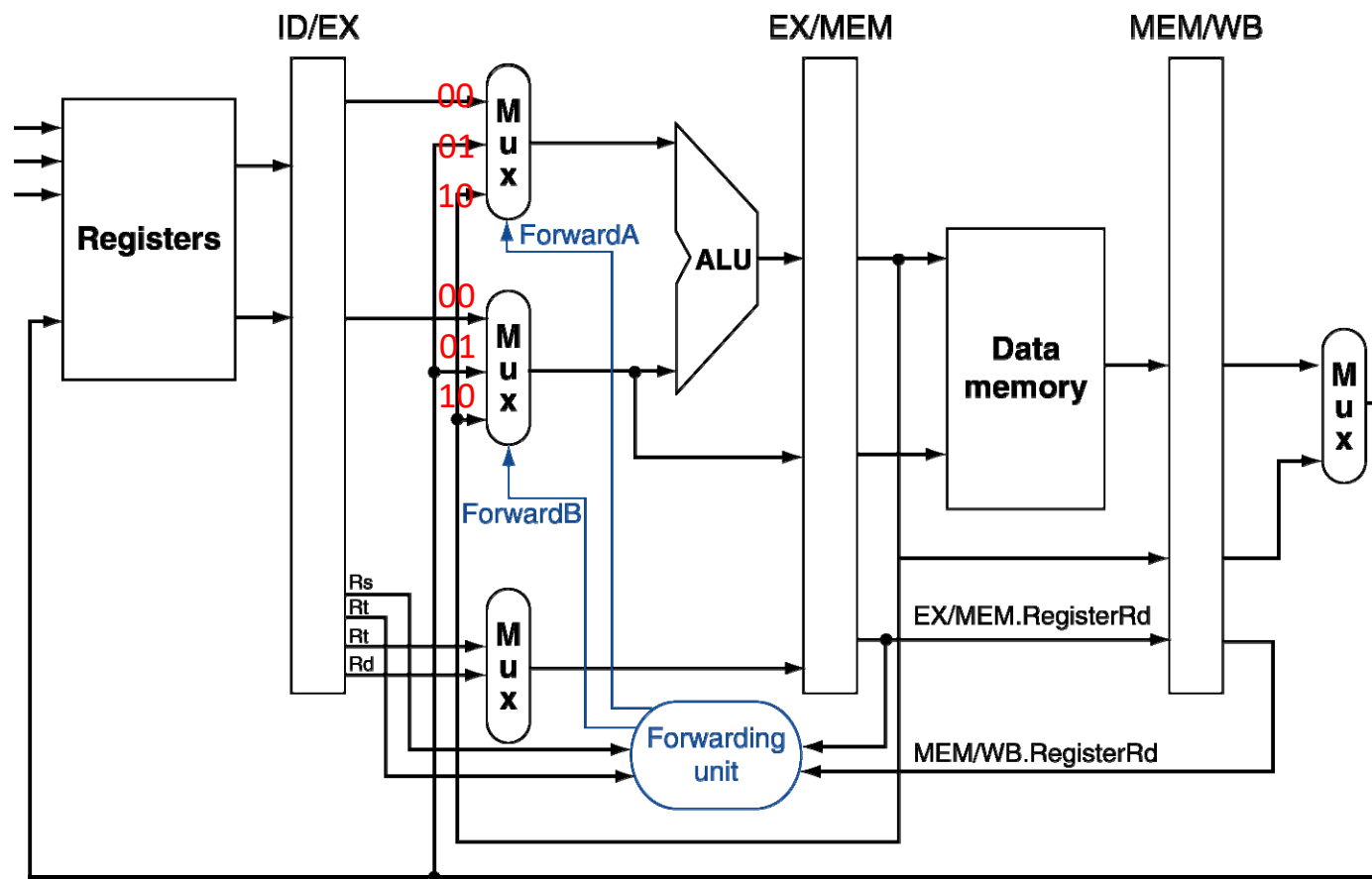
Forwarding to solve Data Hazards





# Forwarding Hardware: Multiplexor Control

- Add **Forwarding unit** and **ForwardA** and **ForwardB** control signal to control mux (See next slides)



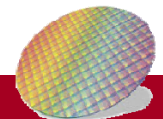


# Forwarding Hardware: Multiplexor Control

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The <b>first</b> ALU operand ( <b>Rs</b> ) comes from the register file
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from <b>prior ALU result</b>
ForwardA = 01	MEM/WB	* The first ALU operand is forwarded from <b>data memory</b> or an earlier <b>ALU result</b>

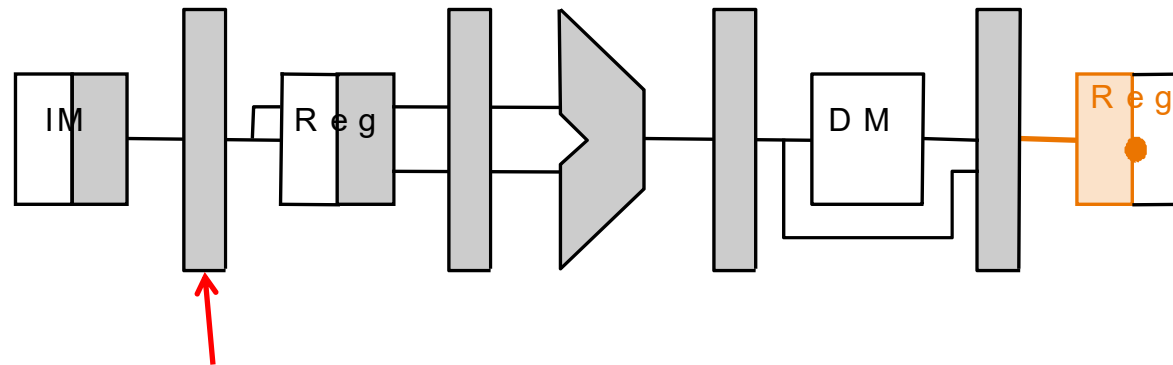
ForwardB = 00	ID/EX	The <b>second</b> ALU operand ( <b>Rt</b> ) comes from the <b>register</b> file
ForwardB = 10	EX/MEM	The <b>second</b> ALU operand is forwarded <b>from prior ALU</b> result
ForwardB = 01	MEM/WB	* The second ALU operand is forwarded from <b>data memory</b> or an earlier ALU result

\* Depending on the selection in the rightmost multiplexor (see datapath with control diagram)



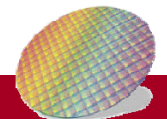
# Detecting the Need to Forward

- **Register numbers** are passed along pipeline
  - e.g., `ID/EX.RegisterRs` = register number for `Rs` sitting in `ID/EX` pipeline register
- E.g.: ALU operand **register numbers** in EX stage are given by
  - `ID/EX.RegisterRs`, `ID/EX.RegisterRt`



sub \$2 \$1 \$3

At the first cycle,  
`IF/ID.RegisterRs`=\$1  
`IF/ID.RegisterRt`=\$3  
`IF/ID.RegisterRd`=\$2





# Detecting hazard- EX and MEM hazard

- Data hazards when

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

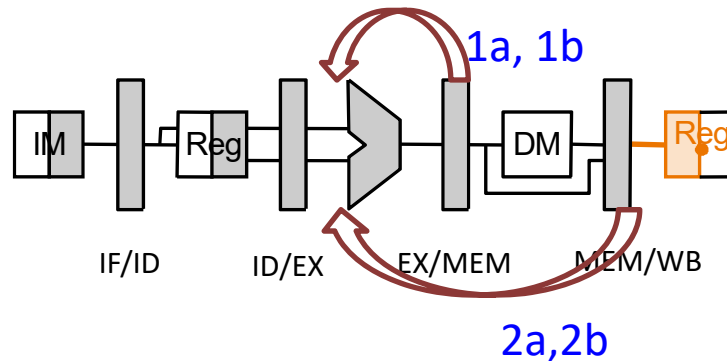
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

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

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

Fwd from  
EX/MEM  
pipeline reg

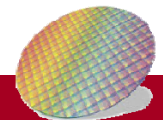
Fwd from  
MEM/WB  
pipeline reg



```
sub $2, $1, $3
and $12, $2, $5
```

Hazard is detected when the **and** is in EX stage and the **sub** is in MEM stage because

$EX/MEM.RegisterRd = ID/EX.RegisterRs = \$2$  (1a)





# Detecting EX hazard and forward

- Forwarding is needed if **earlier** instruction **write** to a register!
  - Check if EX/MEM.RegWrite, MEM/WB.RegWrite is 1

```
Inst1 sub $2, $1, $3
Inst2 and $12, $2, $5
```

Forwarding is needed if **Rd** is not **\$zero**

If **Rd=\$zero**, result is always 0

Check if EX/MEM.RegisterRd  $\neq$  0, MEM/WB.RegisterRd  $\neq$  0

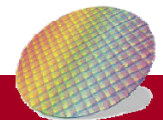
```
Inst1 sub $2, $1, $3
Inst2 and $0, $2, $5
```

**Inst 1 does not need to forward to inst2 because inst2 always produce 0 (\$zero)**

Summary for 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

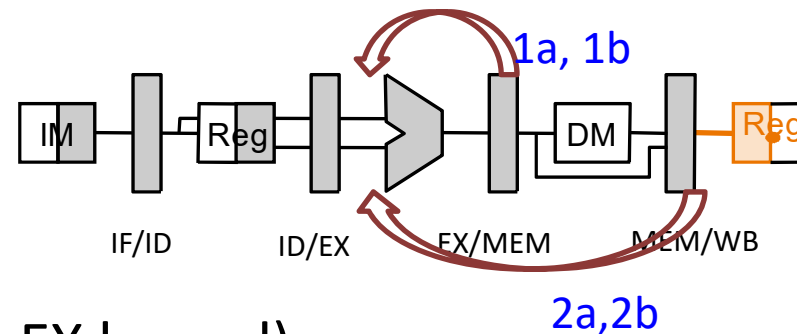


# Detecting hazard – MEM hazard (1)

- MEM hazards when

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

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



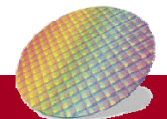
Fwd from  
EX/MEM  
pipeline reg

Fwd from  
MEM/WB  
pipeline reg

Version 1 (similar to EX 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





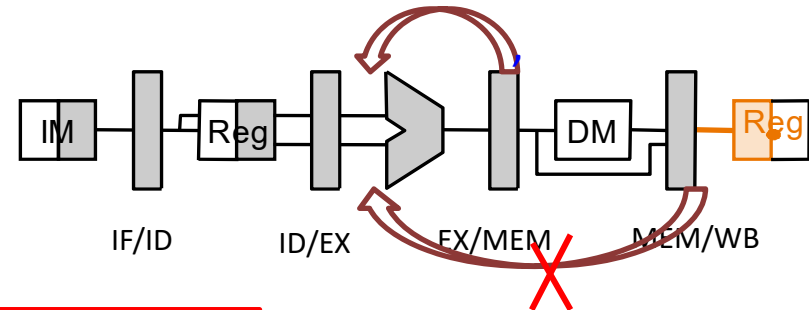


# Additional rule for MEM hazard

- Forwarding is not needed if the later instruction is going to write the same register, even if there is register number match as in conditions above

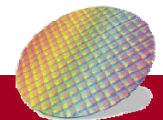
Inst1 **add \$7, \$7, \$9**  
 Inst2 **add \$7, \$7, \$10**  
 Inst3 **add \$7, \$7, \$11**

**Inst 1 does not need to forward to inst 3 because inst2 has more recent data**



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**

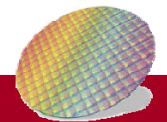
If(MEM/WB.RegWrite  
 and (MEM/WB.RegisterRd !=0)  
 and **not (EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0))**  
 and **(EX/MEM.RegisterRd != ID/EX.RegisterRt)**  
 and (MEM/WB.RegisterRd=ID/EX.RegisterRt) **ForwardB=01**



# Outline

---

- A pipelined datapath
- Pipelined control
- Data hazards and forwarding
- **Data hazards and stalls**
- Branch hazards





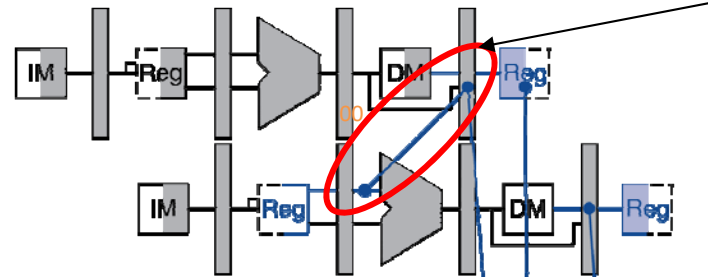
# Load-Use Data Hazard

- One **stall** is needed for load-use data hazard  
=> similar to add a nop (no operation) instruction

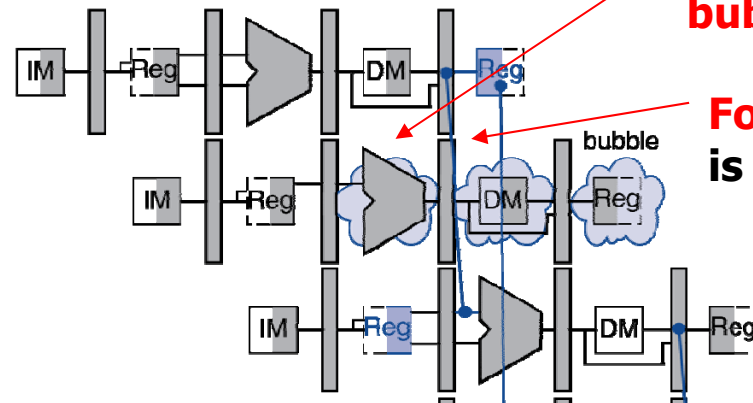
lw \$2, 20(\$1)  
and \$4, \$2, \$5



lw \$2, 20(\$1)  
nop  
and \$4, \$2, \$5

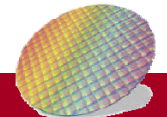


Need to stall one cycle



Hazard detection unit inserts a 1-cycle bubble in the pipeline

Forward data when it is ready





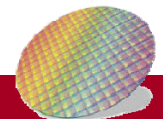
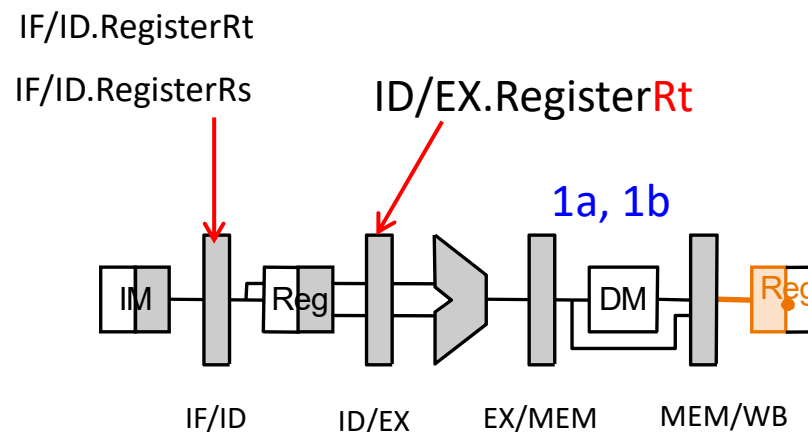
# Hazard Detection Logic to Stall

- Recall: lw instruction format: `lw Rt, offset(Rs)`
- Hardware to check if **stall** is needed

Destination Register in lw instruction.

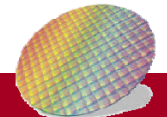
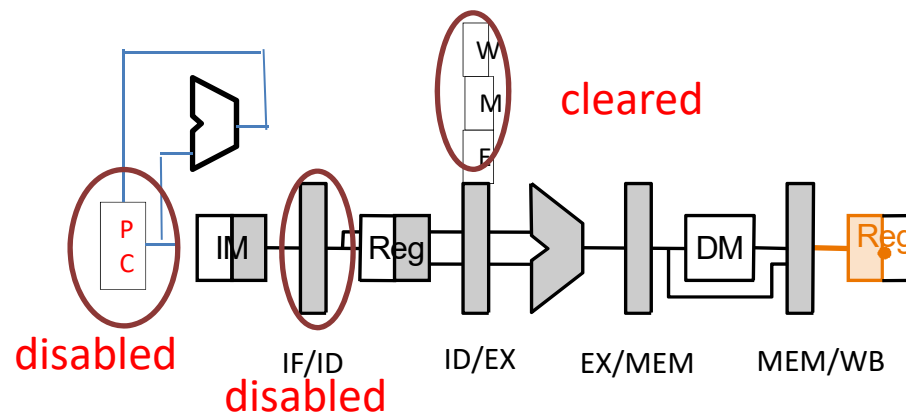
```

if ( ID/EX.MemRead // if the instruction in the EX stage is a load...
    and ( ( ID/EX.RegisterRt = IF/ID.RegisterRs // and the destination register
        or ( ID/EX.RegisterRt = IF/ID.RegisterRt ) ) // matches either source register
        stall the pipeline // of the instruction in the ID stage
  
```



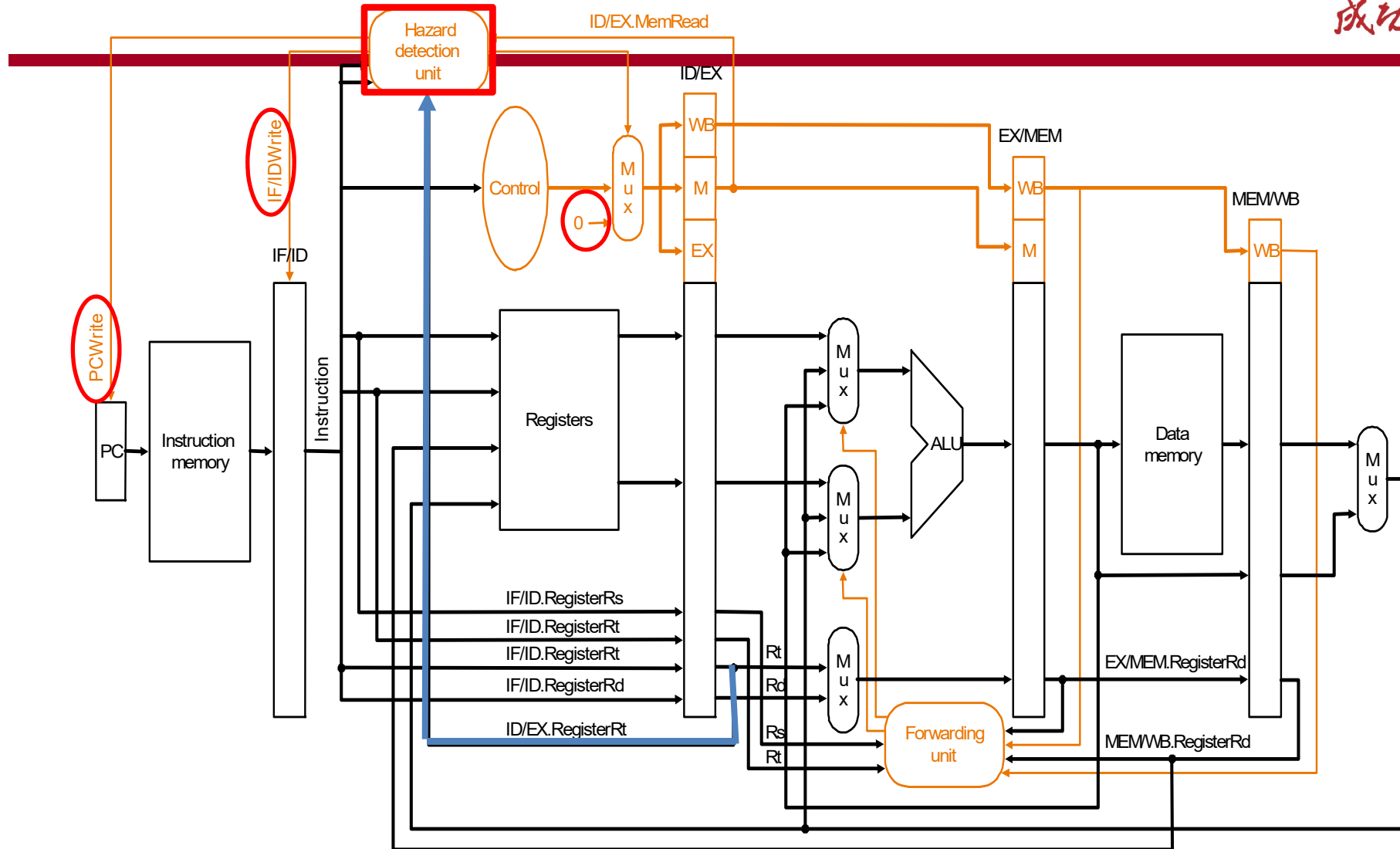
# How to stall the Pipeline

- Only **Stall 1 clock** cycle after the load
  - the forwarding unit can resolve the dependency
- How hardware stalls the pipeline 1 cycle:
  - **Disable write on PC** => this will cause the instruction in the **IF stage** to repeat, i.e., *stall*
  - **Disable write on IF/ID register** => this will cause the instruction in the **ID stage** to repeat, i.e., *stall*
  - **Changes all the EX, MEM and WB control fields in the ID/EX pipeline register to 0** => , so the instruction just behind the **load** becomes a **nop** – a **bubble** is said to have been inserted into the pipeline

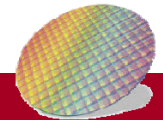




# Adding Hazard Detection Unit

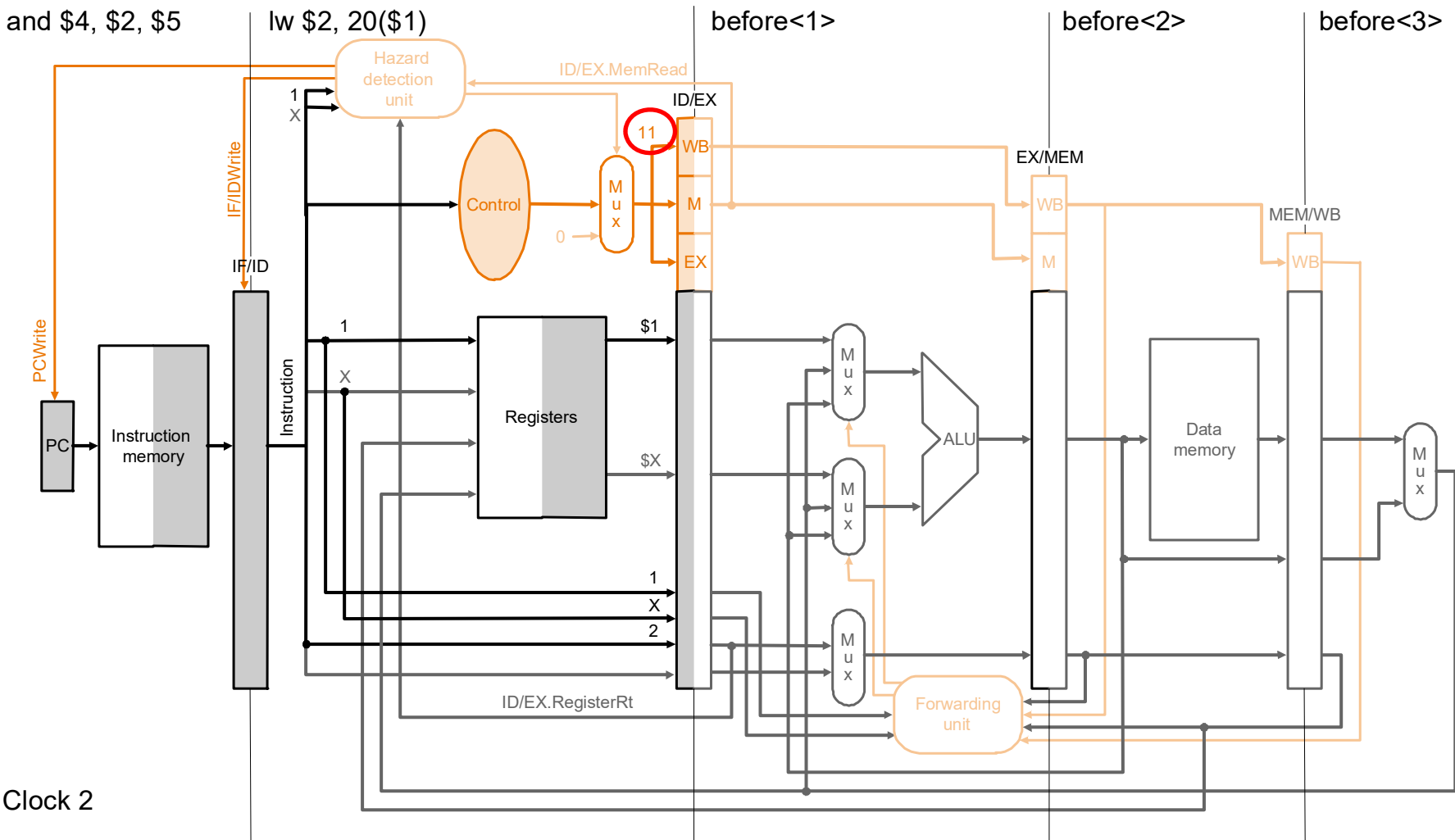


Datapath with forwarding hardware, the hazard detection unit and controls wires – certain details, e.g., branching hardware are omitted to simplify the drawing



# Cycle 2

```
lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
```



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

Detect a hazard and insert a stall

# Cycle 3

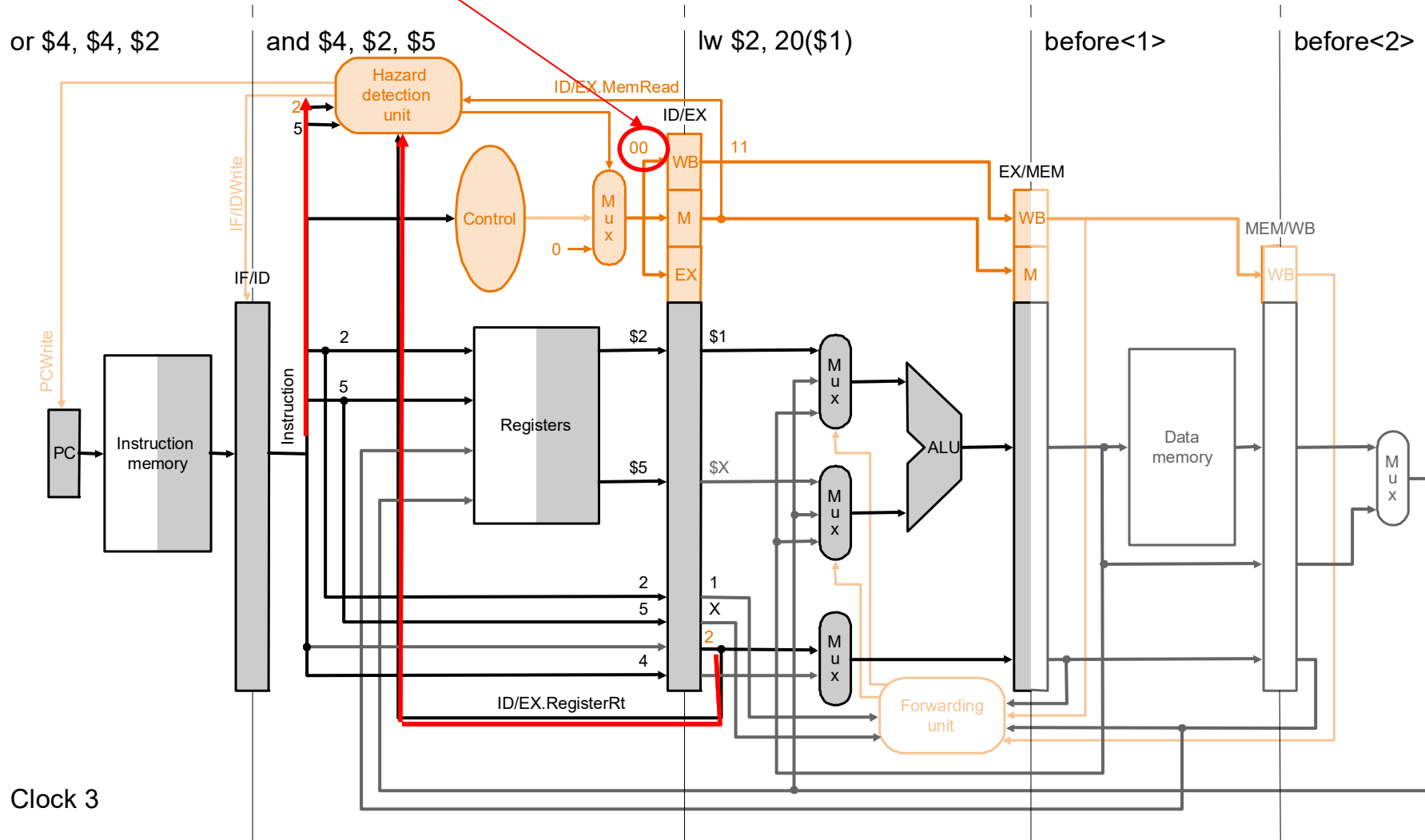
or \$4, \$4, \$2

and \$4, \$2, \$5

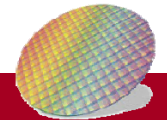
lw \$2, 20(\$1)

before<1>

before<2>



Clock 3



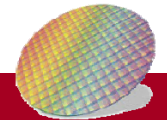
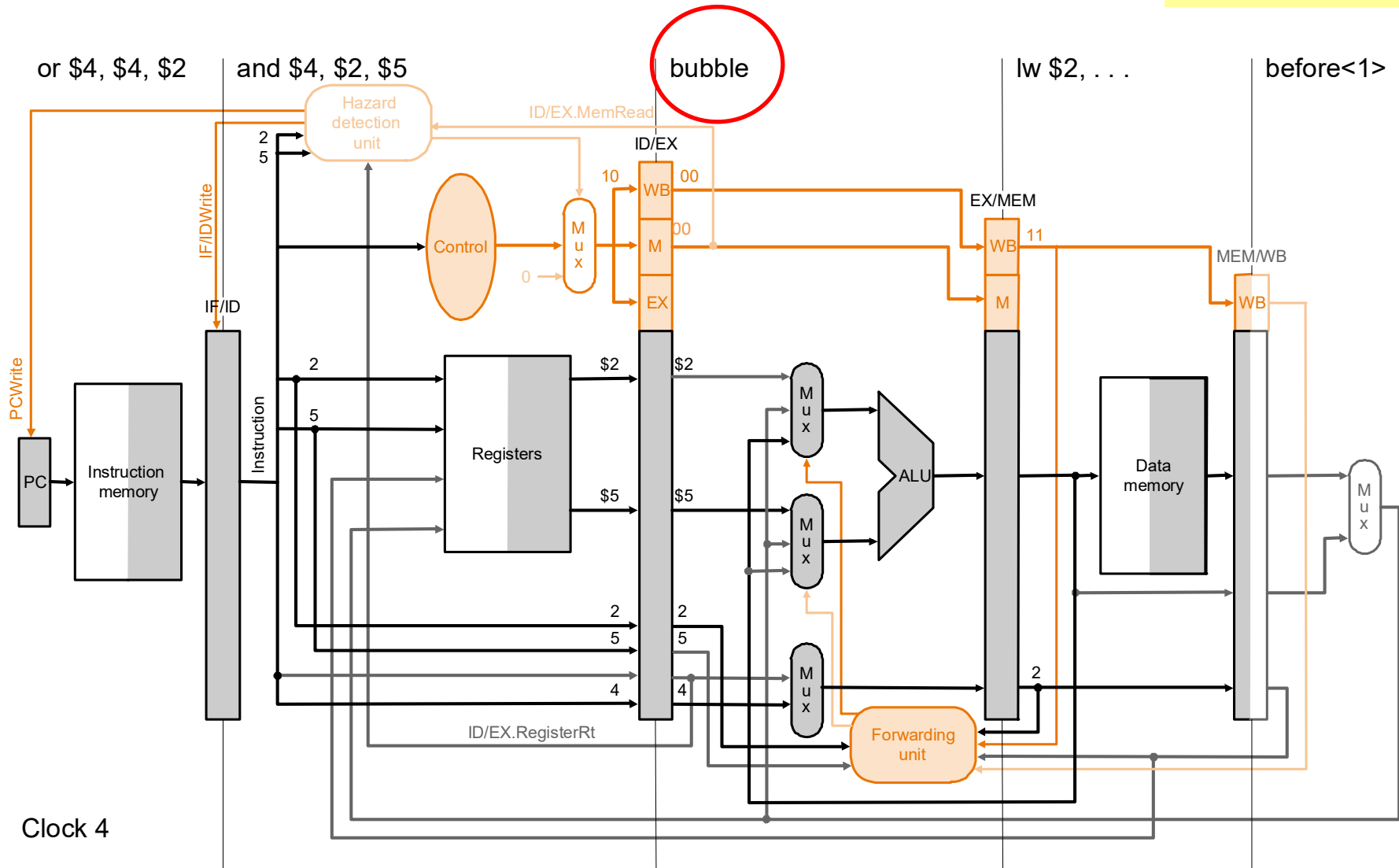


```

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

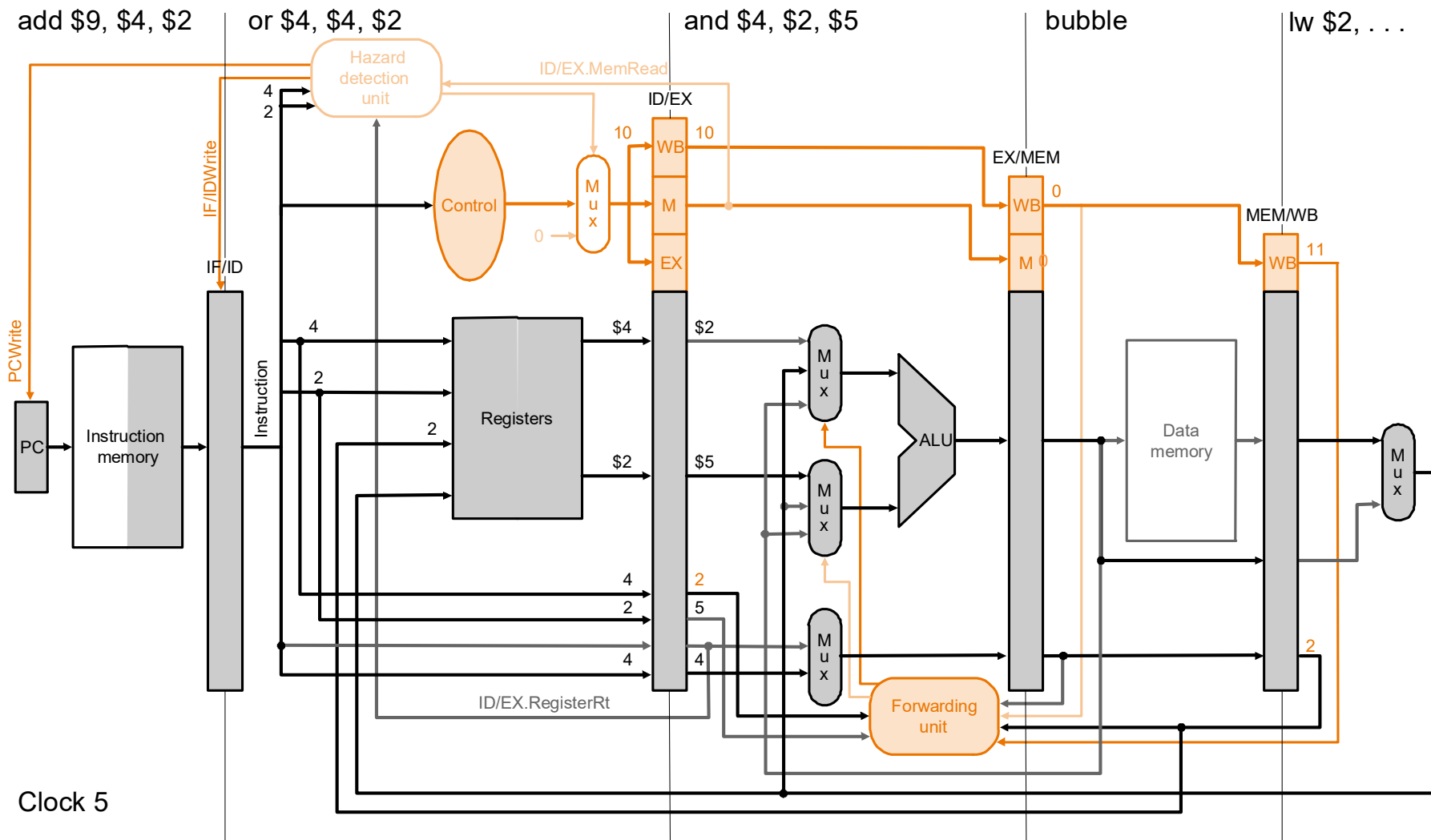
```

# Cycle 4



# Cycle 5

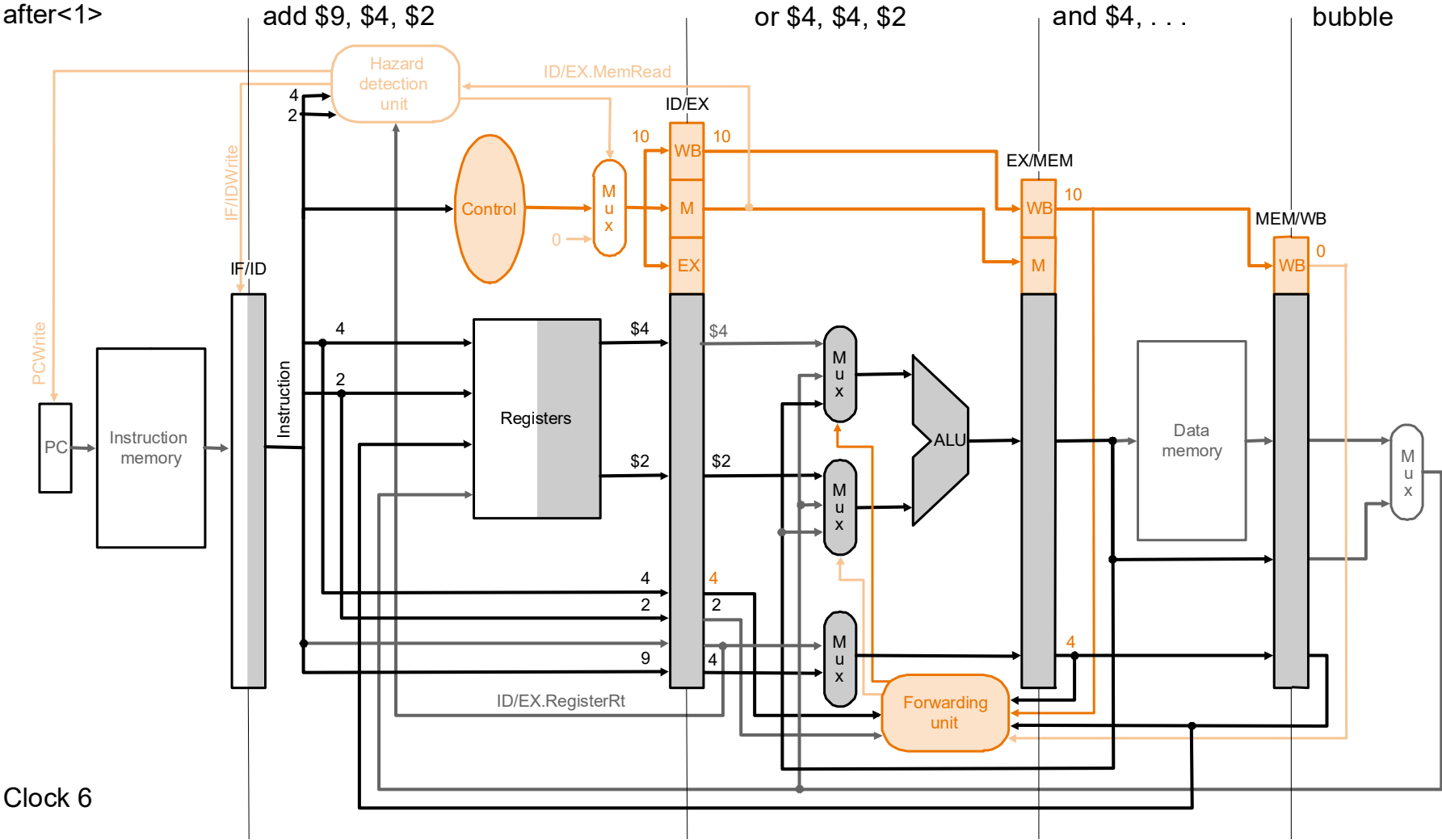
```
lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
```



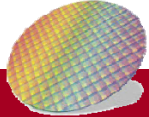
Clock 5

# Cycle 6

```
lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
```

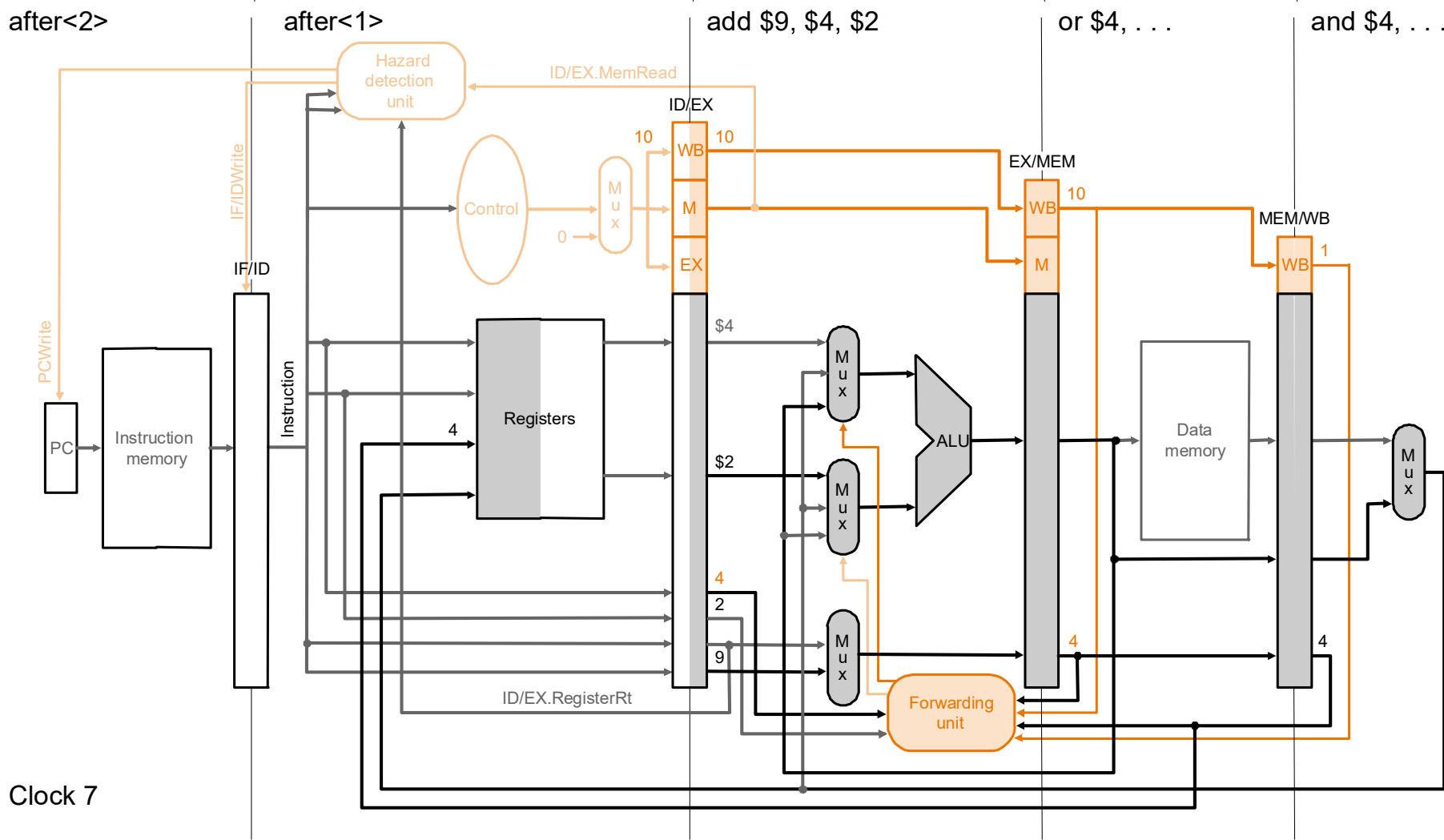


Clock 6

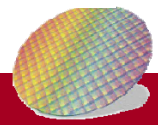


# Cycle 7

```
lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
```



Clock 7



## Special case: load store pair

- Are there any stalls in the following instructions if forwarding is used? If yes, how many?

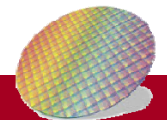
lw \$2, 20(\$1)

and \$4, \$2, \$5

- Are there any stalls in the following instructions if forwarding is used? If yes, how many?

lw \$2, 20(\$1)

sw \$2, 4(\$3)





成功大學

National Cheng Kung University

---

# Backup Slides

