Topics

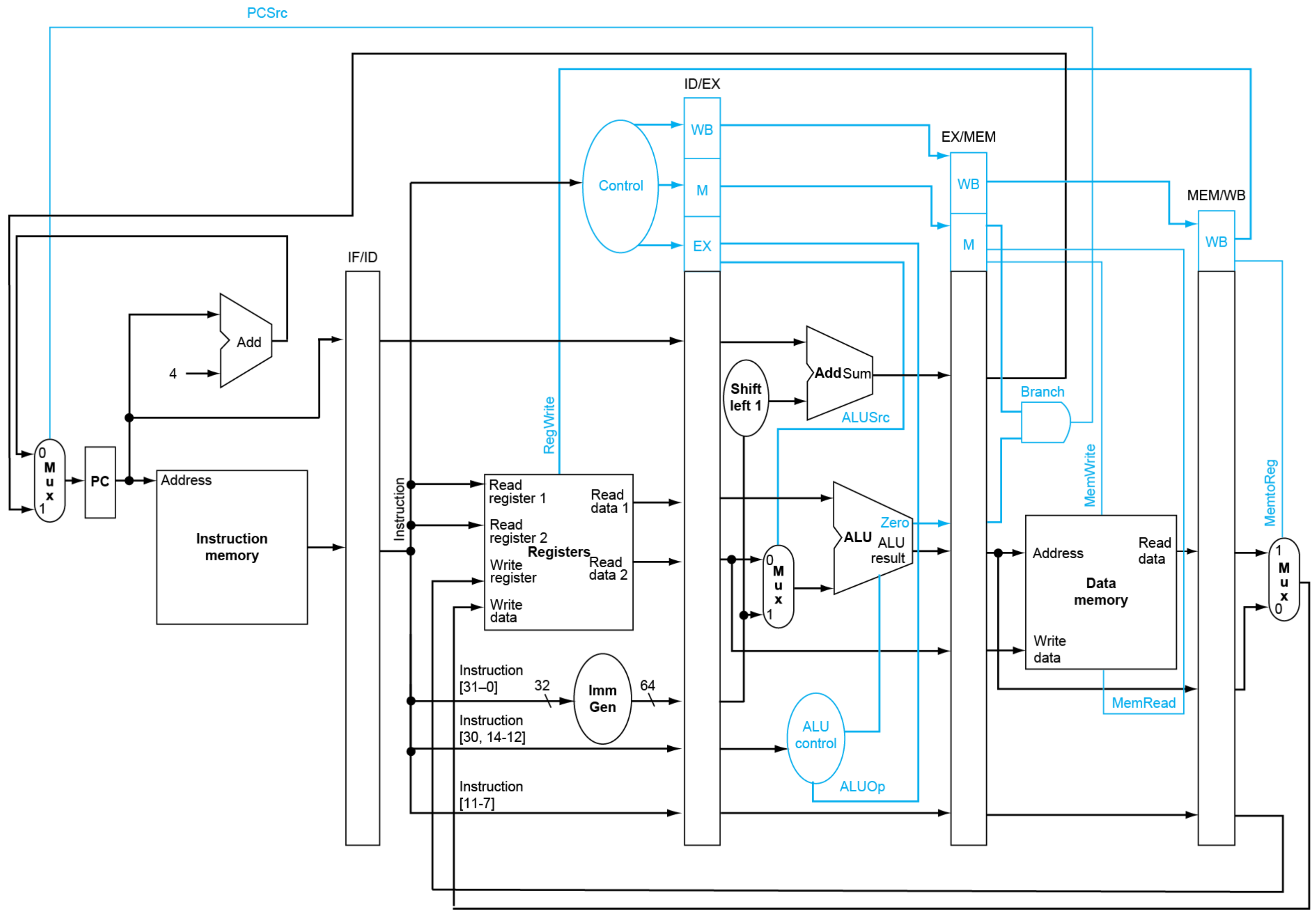
- Single Cycle CPU Design (sections 4.1-4.4)

- Pipelining
  - Overview (section 4.6)
  - Pipelined Datapath and control (section 4.7)
  - ➔ Data hazard (section 4.8)
  - Control hazard (section 4.9)

Thus times do shift,
each thing his turn does hold;
New things succeed,
as former things grow old.

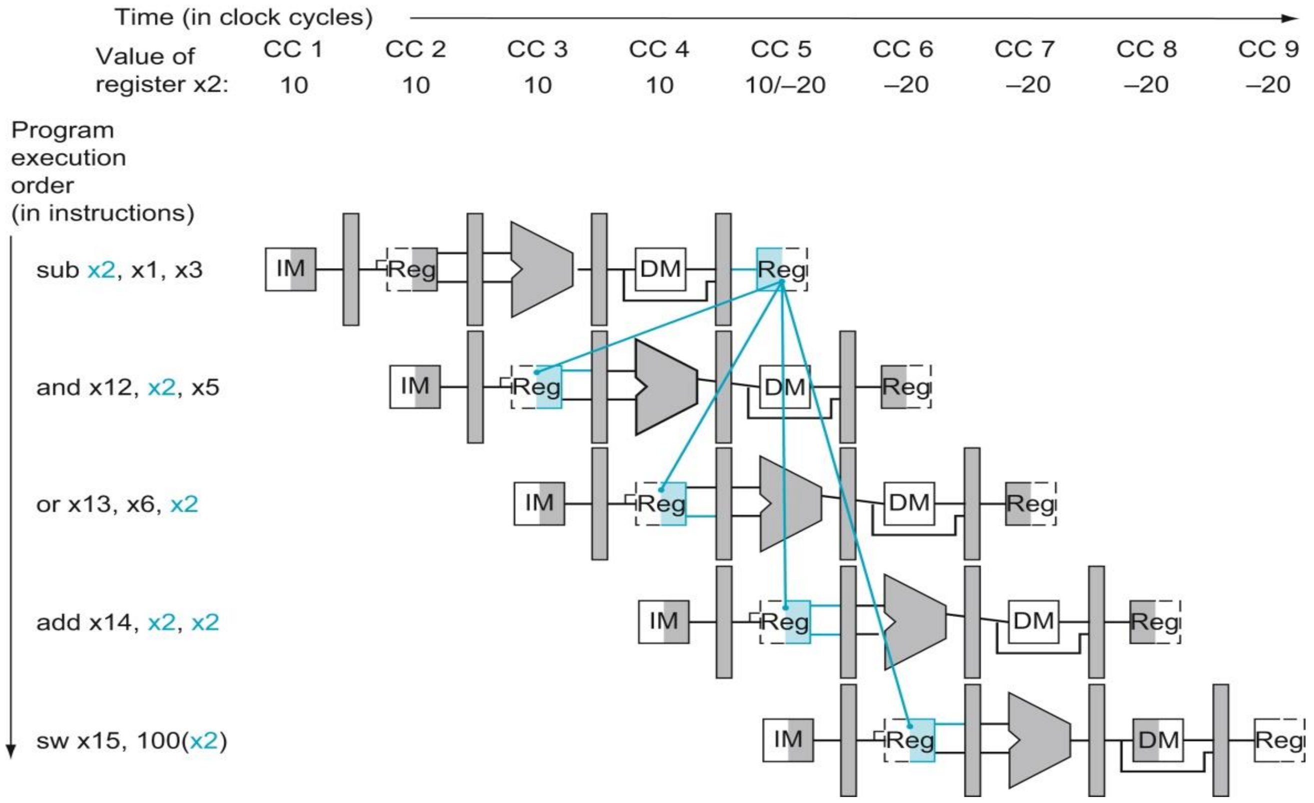
Robert Herrick

Recap: The Complete Pipelined Datapath



Data Hazards

- The input of some instruction depends on the output of another instruction which is still in the pipeline
- An example: initially x2=10, x1=10, x3=30



Resolving Data Hazard

- **Write reg in the first half of CC and read it in the second half of that CC.**

- **Insert NOP instructions, or independent instructions by compiler**

- **Detect the hazard, then forward the proper value**

Data Hazard Detection

- From the example,

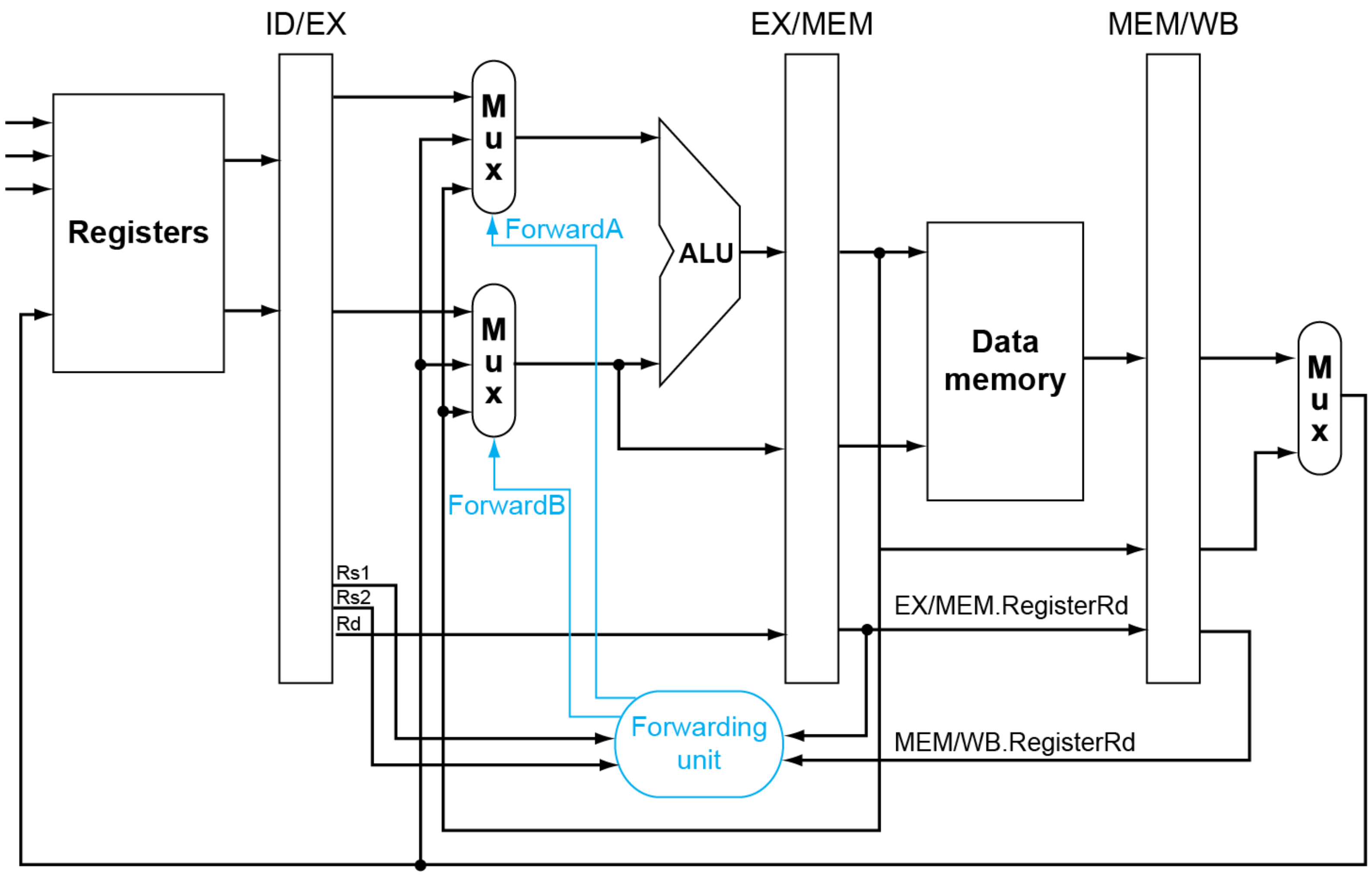
```
1.    sub    s2, s1, s3
2.    and    t1, s2, s5
3.    or     t2, s6, s2
```

  - and and or needs the value of s2 at ALU stage
  - For first two instructions, hazard happens when sub is in MEM stage, while and is in ALU (EX) stage
  - For the first and third instructions, hazard happens when sub is in WB stage while or is in ALU (EX) stage

- Hazard detection conditions: EX hazard and MEM hazard

1a.	EX/MEM.RegisterRd	=	ID/EX.RegisterRs1
1b.	EX/MEM.RegisterRd	=	ID/EX.RegisterRs2
2a.	MEM/WB.RegisterRd	=	ID/EX.RegisterRs1
2b.	MEM/WB.RegisterRd	=	ID/EX.RegisterRs2

Add Forwarding Paths



Refine the Hazard Detection Condition

- **If the instruction in MEM stage and WB stage does not write**
  - No hazard
  - Check RegWrite signal in the WB field of the EX/MEM and MEM/WB pipeline register

- **If RegisterRd is x0.**
  - No hazard

New Hazard Detection Conditions

- EX hazard

```
if (      EX/MEM.RegWrite
    and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))
            ForwardA = 10

if (      EX/MEM.RegWrite
    and (EX/MEM.RegisterRd != 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
            ForwardB = 10
```

- One instruction ahead

New Hazard Detection Conditions

- MEM Hazard

```
    if (     MEM/WB.RegWrite
        and (MEM/wB.RegisterRd !=0)
        and (MEM/wB.RegisterRd = ID/EX.RegisterRs1))
            ForwardA = 01

    if (     MEM/WB.RegWrite
        and (MEM/wB.RegisterRd !=0)
        and (MEM/wB.RegisterRd = ID/EX.RegisterRs2))
            ForwardB = 01
```

- Two instructions ahead

New Complication

- **For code sequence:**

```
add x1, x1, x2
add x1, x1, x3
add x1, x1, x4
```

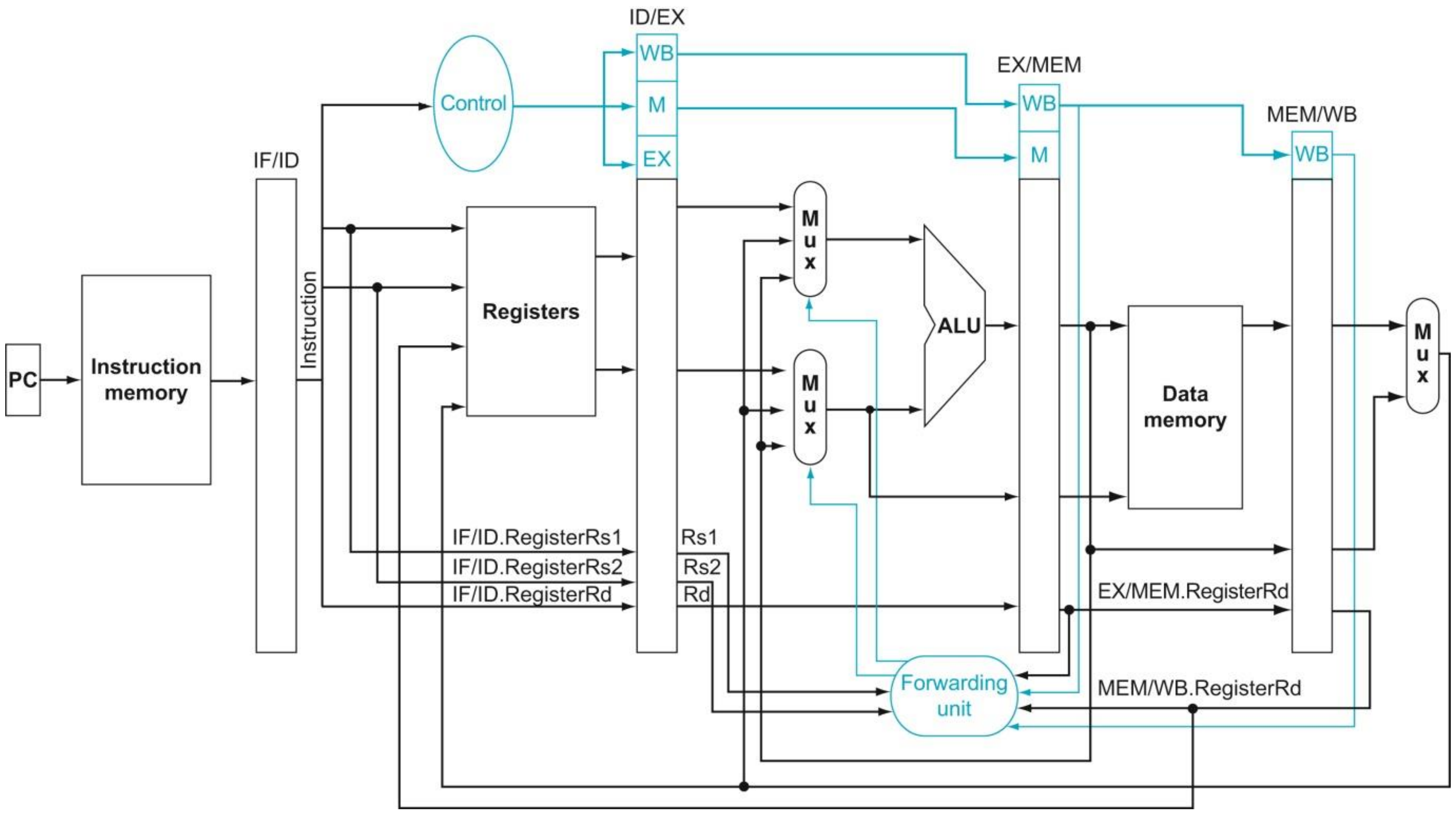
- **3rd instruction depends on the 2nd, not the 1st**
  - Should forward the ALU result from the second instruction (MEM stage)
- **For MEM hazard**
  - EX/MEM.RegisterRd != ID/EX.RegisterRs1
  - EX/MEM.RegisterRd != ID/EX.RegisterRs2

Corrected (MEM) Hazard Detection Conditions

- MEM Hazard (2 instructions ahead)

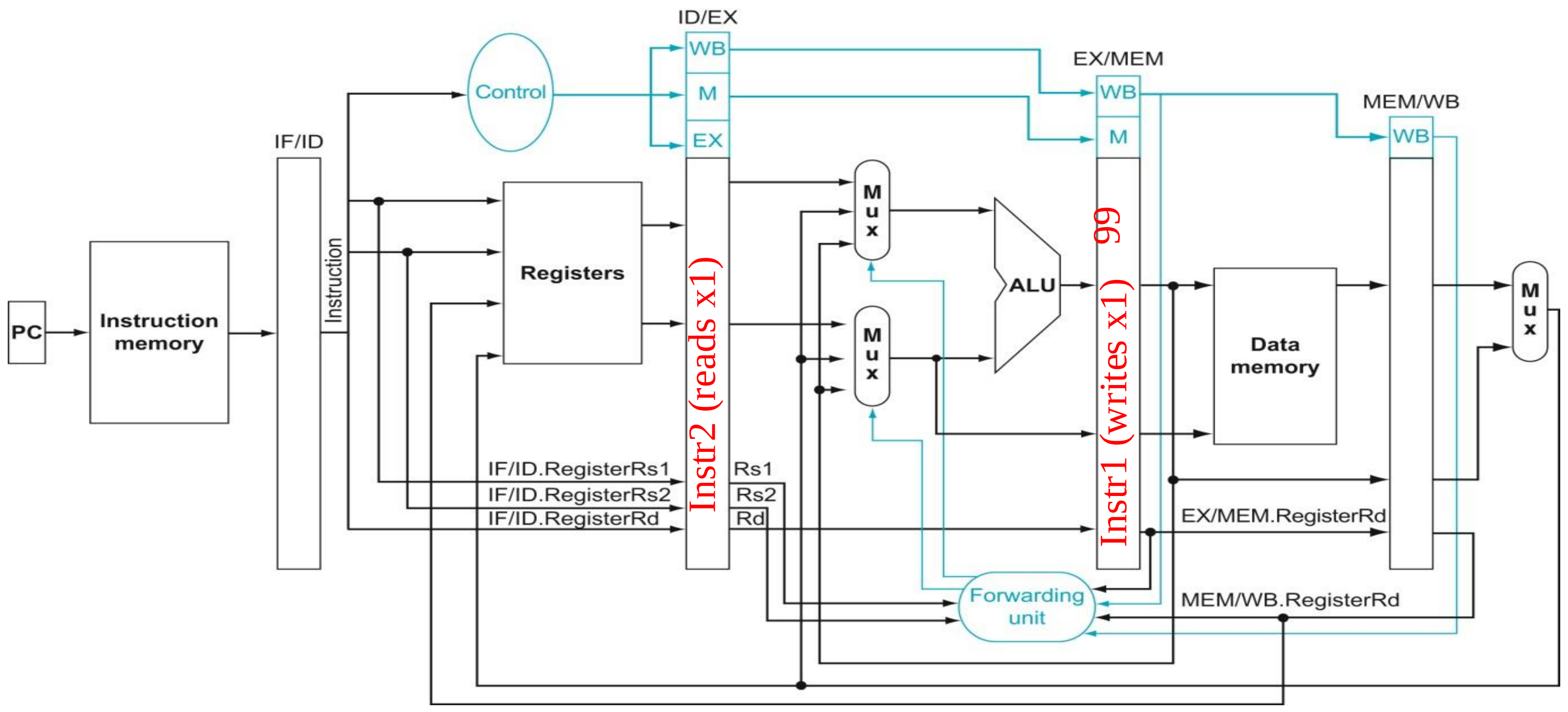
```
if (   MEM/WB.RegWrite
    and (MEM/WB.RegisterRd !=0)
    and (EX/MEM.RegisterRd != ID/EX.RegisterRs1)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs1))
        ForwardA = 01

if (   MEM/WB.RegWrite
    and (MEM/WB.RegisterRd !=0)
    and (EX/MEM.RegisterRd != ID/EX.RegisterRs2)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs2))
        ForwardB = 01
```

Datapath with Forwarding Path



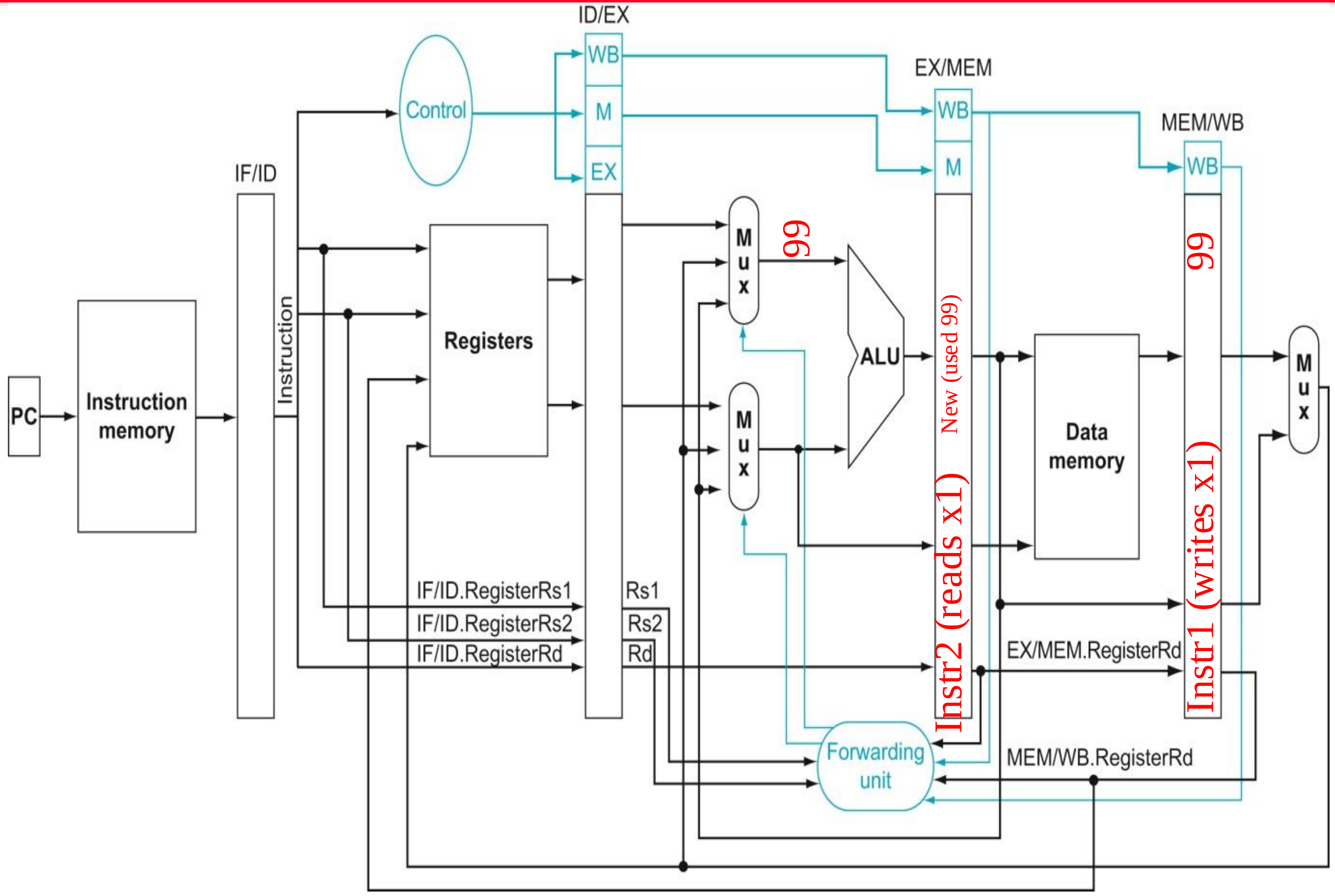
A Forwarding Example - 1

add x1, x2, x3
sub x4, x1, x4



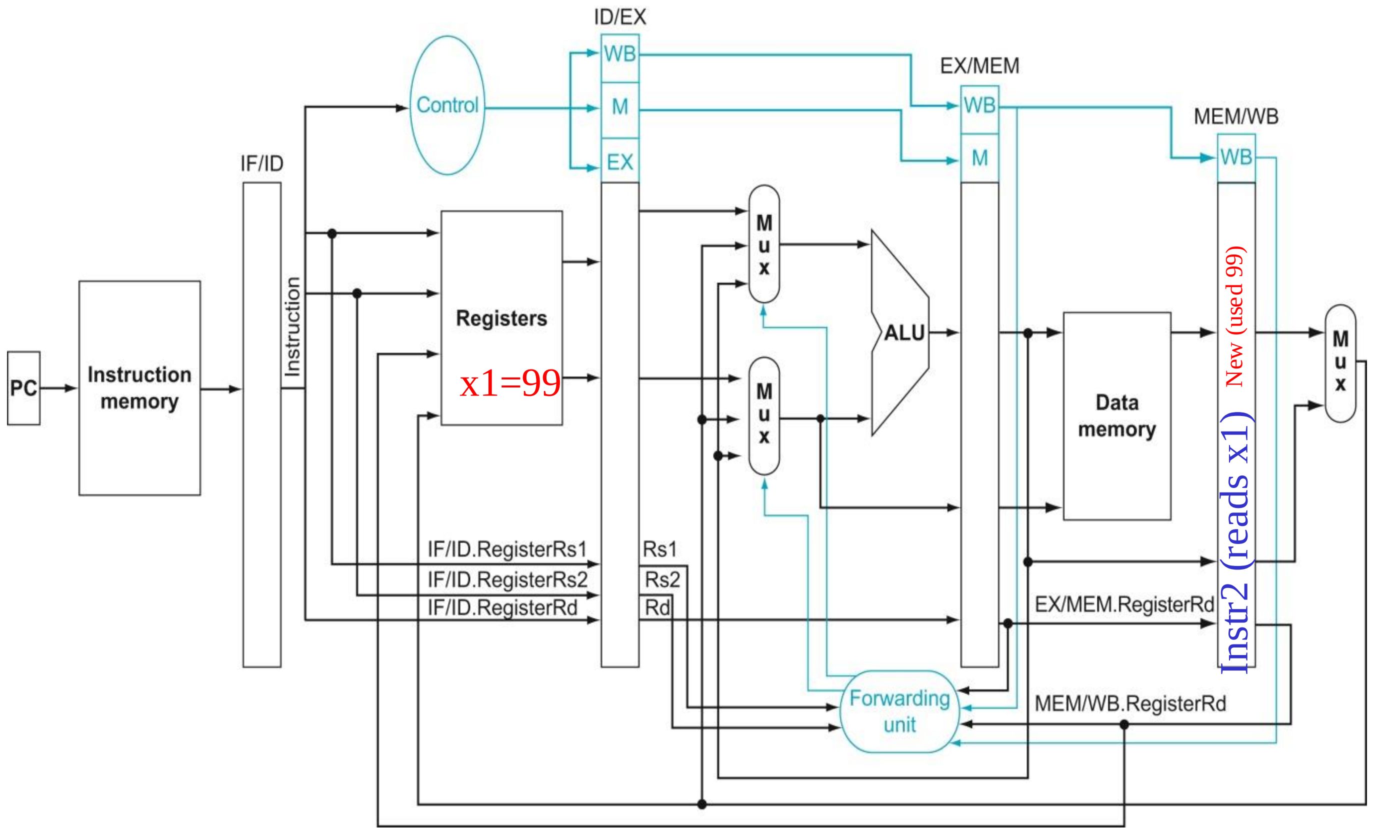
Instr1 computes a value (e.g., 99) and will eventually write that result to x1

A Forwarding Example - 2



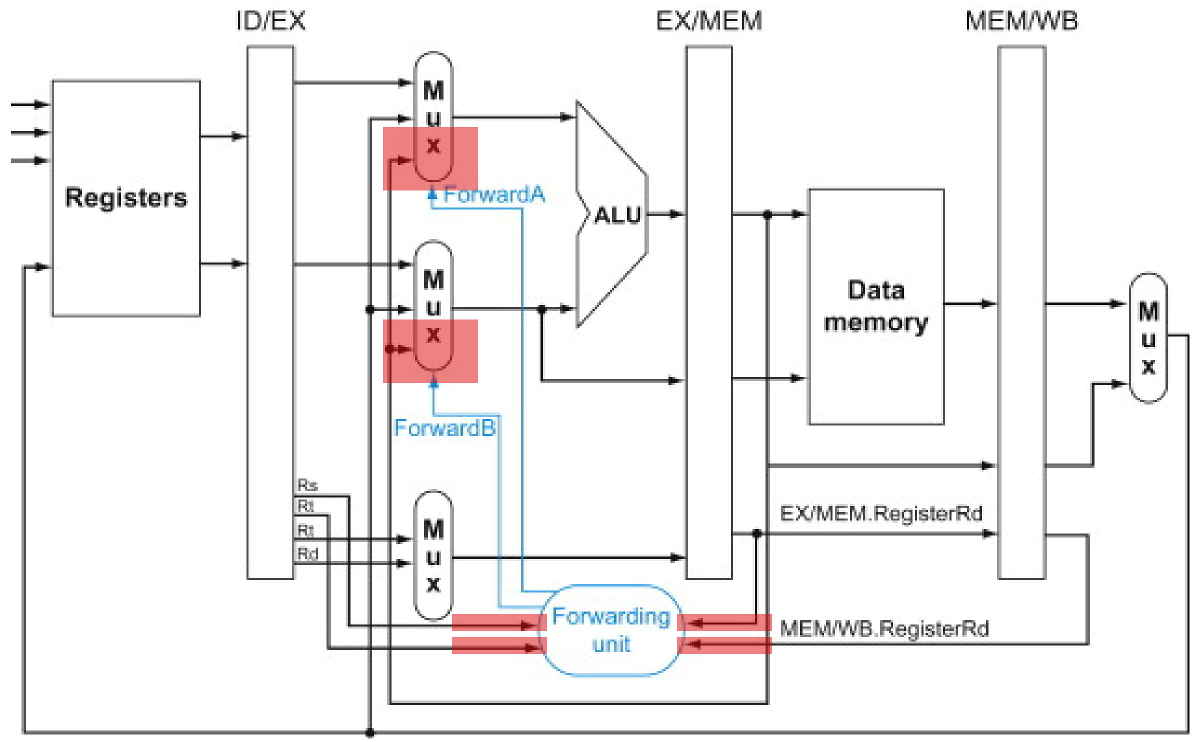
The value (99) isn't in x1 yet, but instr2 needs to read that value. By detecting the hazard, the forwarding unit can pass the value directly from EX/MEM to the ALU

A Forwarding Example - 3



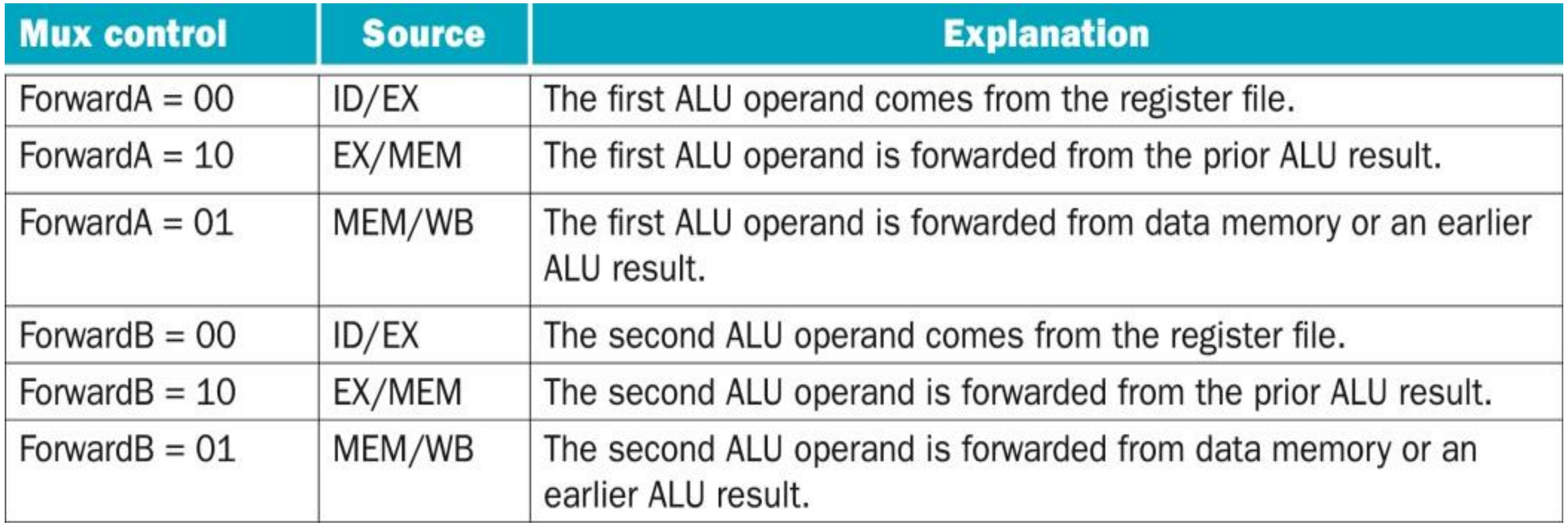
Of course, 99 is still written to x1 during Instr1's write back stage. Forwarding just made 99 available to Instr2 sooner.

A Forwarding Example - 4



The forwarding unit monitors the register source and destination fields in the pipeline registers, detects hazards, and configures the ALU's muxes for forwarding if needed.

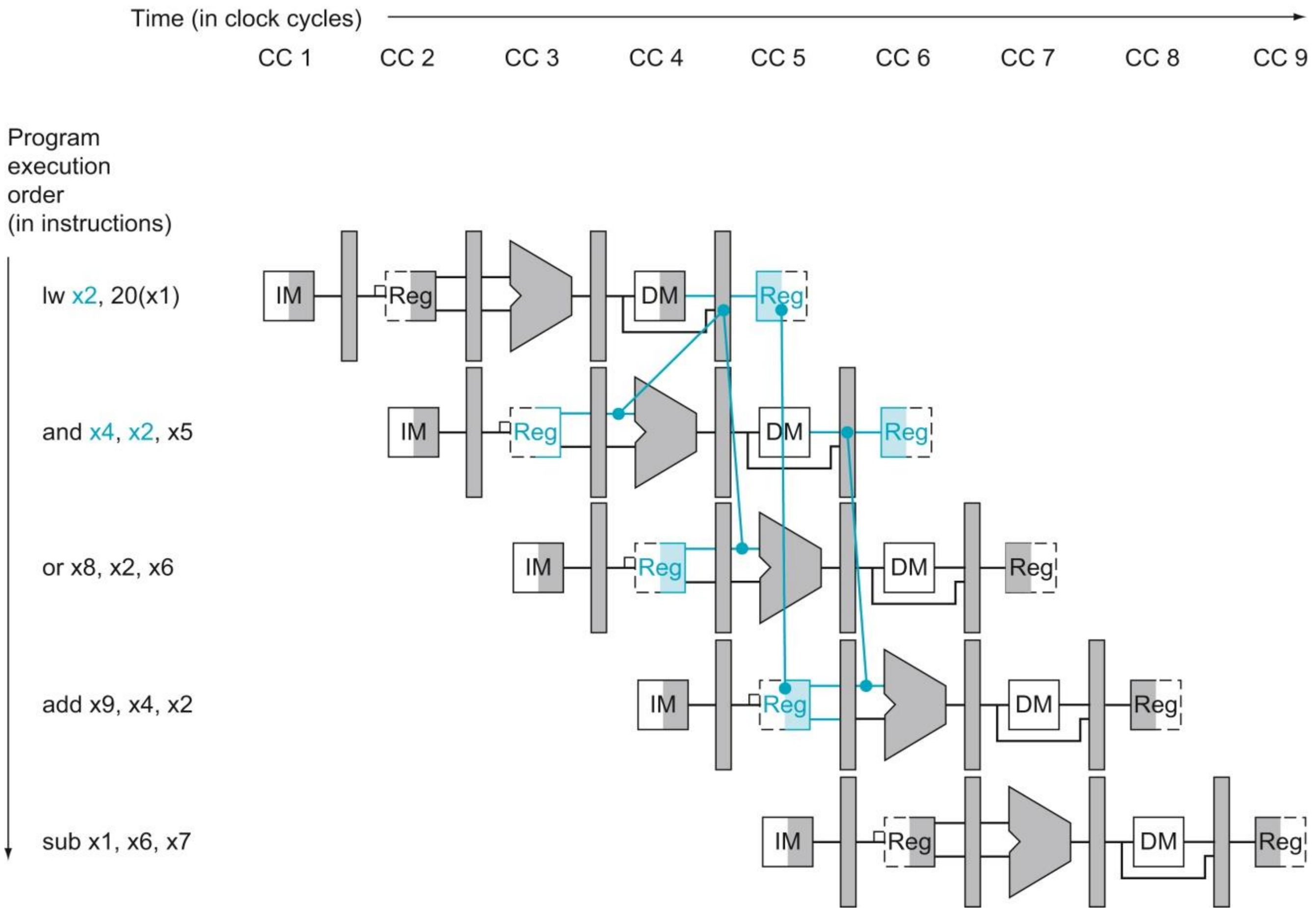
Forwarding Unit – Controls the Multiplexors



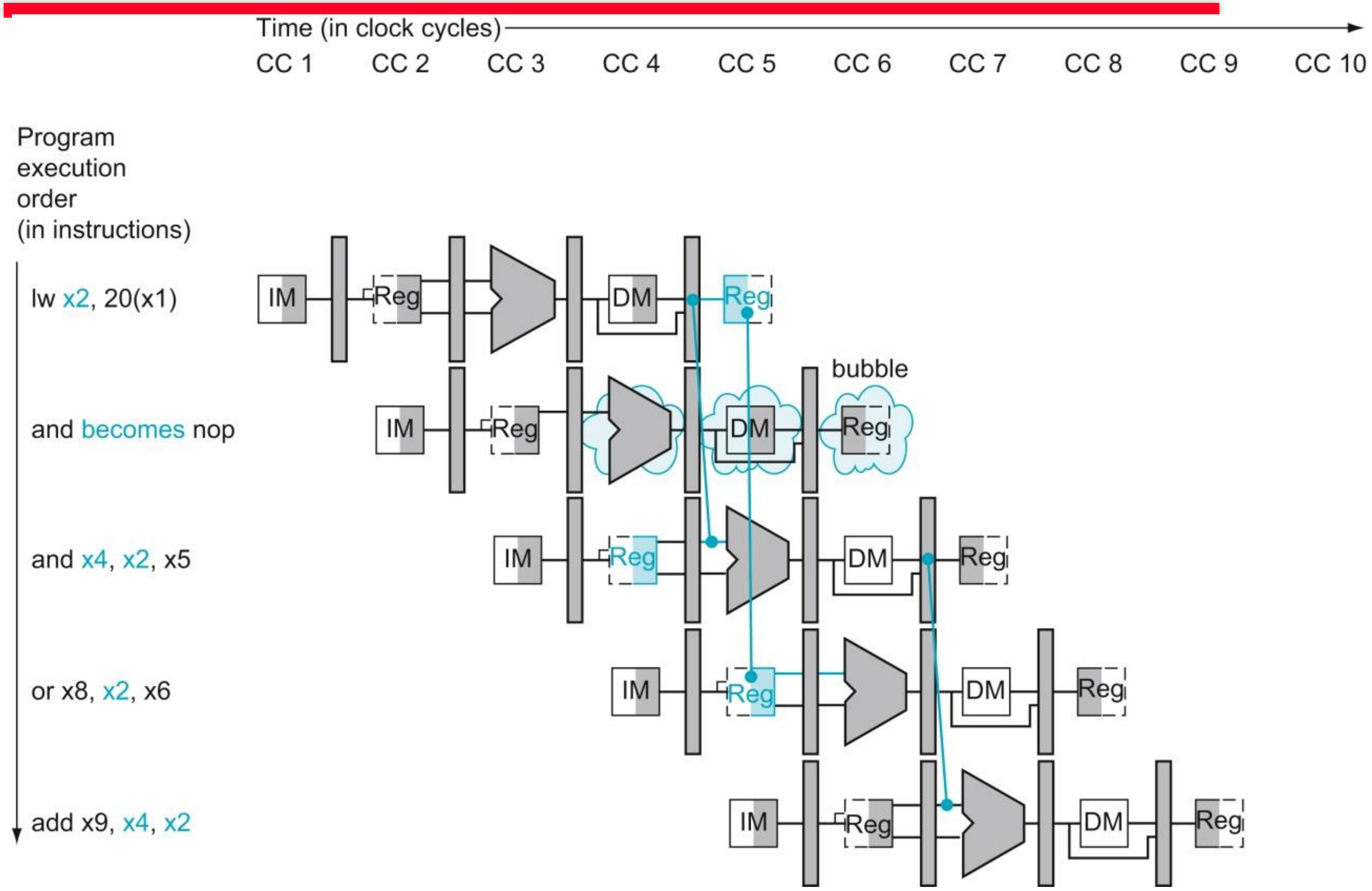
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Forwarding Can't do Everything!

- When an instruction reading a register following a load instruction that writes the same register, forwarding does not solve the data hazard



Stalling the pipeline – Bubble



Hazard Detection

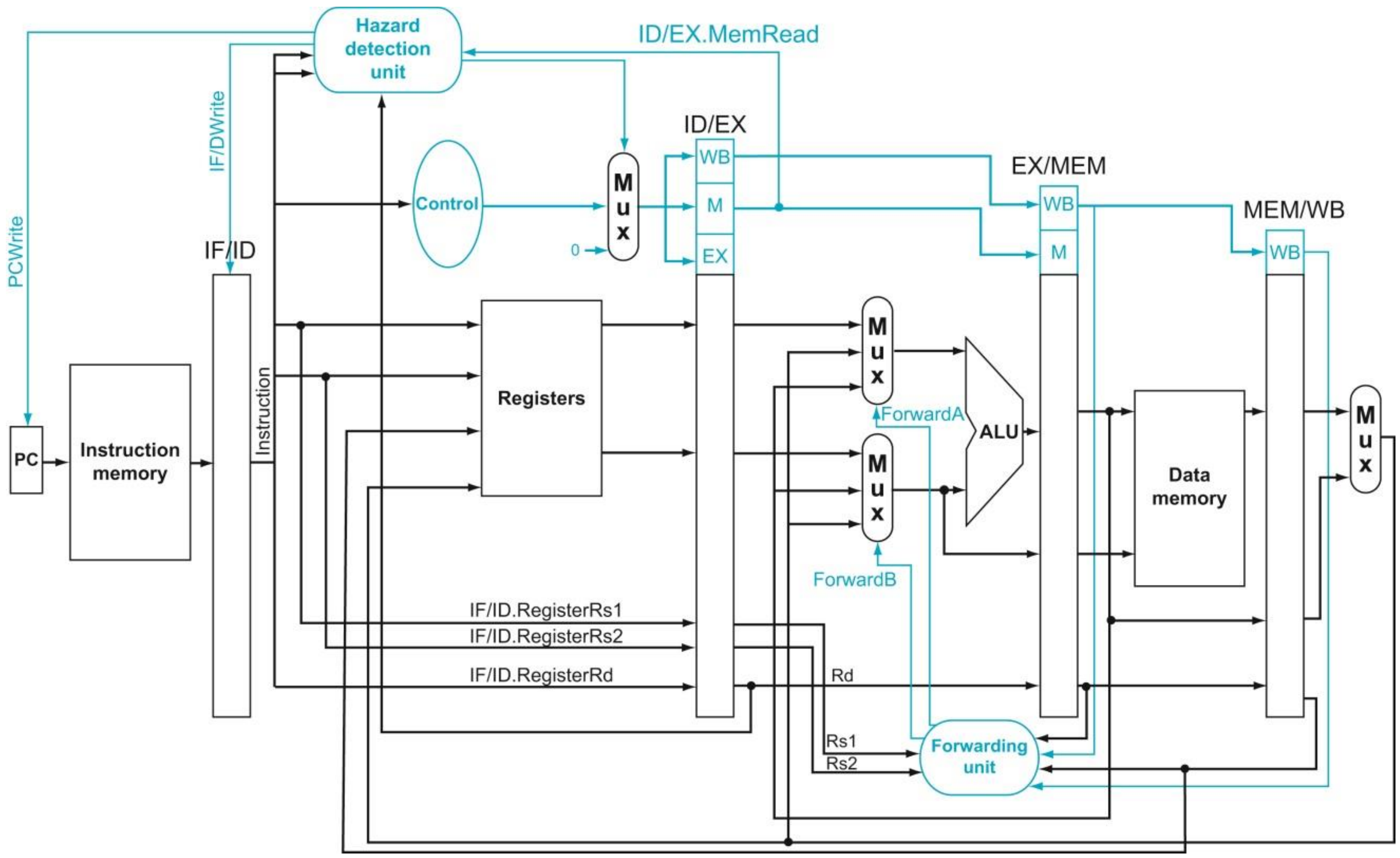
- At the ID stage
- Detection logic

```
        if  (     ID/EX.MemRead
              and (     (ID/EX.RegisterRd = IF/ID.RegisterRs1)
                        or (ID/EX.RegisterRd = IF/ID.RegisterRs2)
    ))
              stall the pipeline
```

How to Stall the Pipeline at ID Stage

- **Deassert all control signals in the EX, MEM, WB stages**
- **Keep IF/ID unchanged – repeat the previous cycle**
  - Add IF/IDWrite control to to data hazard detection unit
- **Keep PC unchanged – refetch the same instruction**
  - Add PCWrite control to to data hazard detection unit

Pipelined Control with Hazard Detection and Data Forwarding Units



Example 1

- **Show how hazard detection unit and forwarding unit work with the following instruction sequence**

```
40: lw  x2, 20 (x1)
44: and x4, x2, x5
48: or  x8, x2, x4
```

Example 1 Solution: Pipelined Execution Diagram

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
lw x2, 20(x1)	IF	ID	EX	MEM	WB			
and x4, x2, x5		IF	ID	***	EX	MEM	WB	
or x8, x2, x4			IF	***	ID	EX	MEM	WB

Example 2

How many cycles are needed to complete the following instruction sequence? Assume that forwarding is implemented.

```
sub    x2, x1, x3
and    x4, x2, x5
or     x4, x4, x2
add    x9, x4, x2
```

Example 2 Solution

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
sub x2, x1, x3	IF	ID	EX	MEM	WB			
and x4, x2, x5		IF	ID	EX	MEM	WB		
or x4, x4, x2			IF	ID	EX	MEM	WB	
add x9, x4, x2				IF	ID	EX	MEM	WB

Example 3

- **How many cycles are needed to complete the following instruction sequence? Assume that forwarding is implemented.**

```
sub x2, x1, x3
 lw x4, 8(x2)
or   x4, x4, x2
add x9, x4, x2
```

Example 3 Solution

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
sub x2, x1, x3	IF	ID	EX	MEM	WB				
lw x4, 8 (x2)		IF	ID	EX	MEM	WB			
or x4, x4, x2			IF	ID	***	EX	MEM	WB	
add x9, x4, x2				IF	***	ID	EX	MEM	WB

Example 4

a) If there is no forwarding or hazard detection, insert nops to ensure correct execution

b) If the processor has forwarding but not hazard detection, insert nops to ensure correct execution

```
1.  add x5, x2, x1
2.  lw x3, 4 (x5)
3.  lw x2, 0(x2)
4.  or x3, x5, x3
5.  sw x3, 0 (x5)
```

Example 4(a) Solution

a)

```
1. add x5, x2, x1
nop
nop
2. lw x3, 4 (x5)
3. lw x2, 0(x2)
```

```
nop
4. or x3, x5, x3
nop
nop
5. sw x3, 0 (x5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
add	IF	ID	EX	MEM	WB						
lw		***	***	IF	ID	EX	MEM	WB			
lw					IF	ID	EX	MEM	WB		
or						***	IF	ID	EX	MEM	WB
sw								***	***	IF	ID

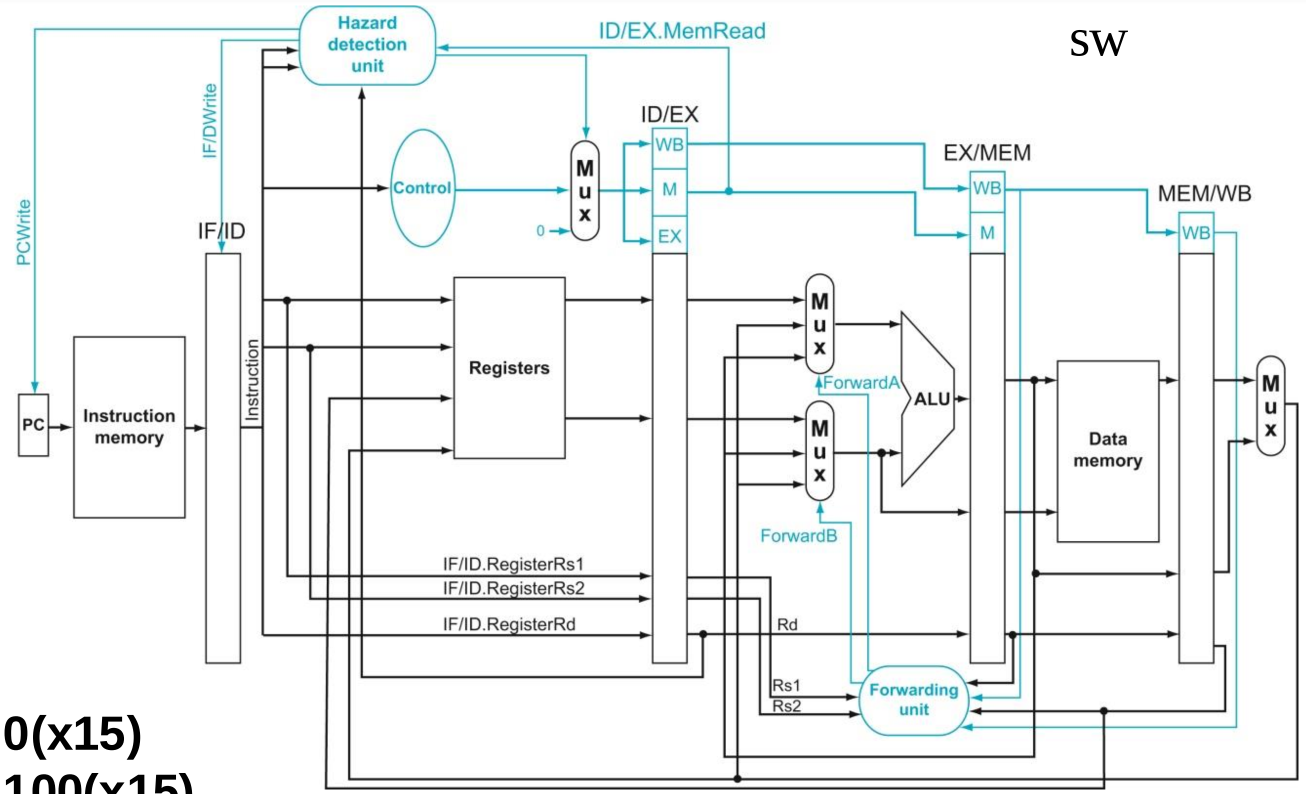
Example 4 (b) Solution

b) If the processor has forwarding but not hazard detection, no need to keep any of the ‘nop’

How about sw?

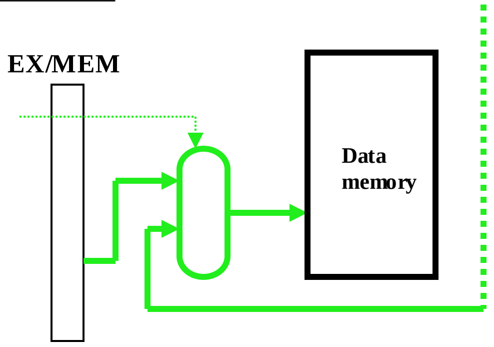
- sw can cause data hazards too
- Does forwarding help?
- Does the existing forwarding hardware help?

Case 1: SW is in MEM Stage



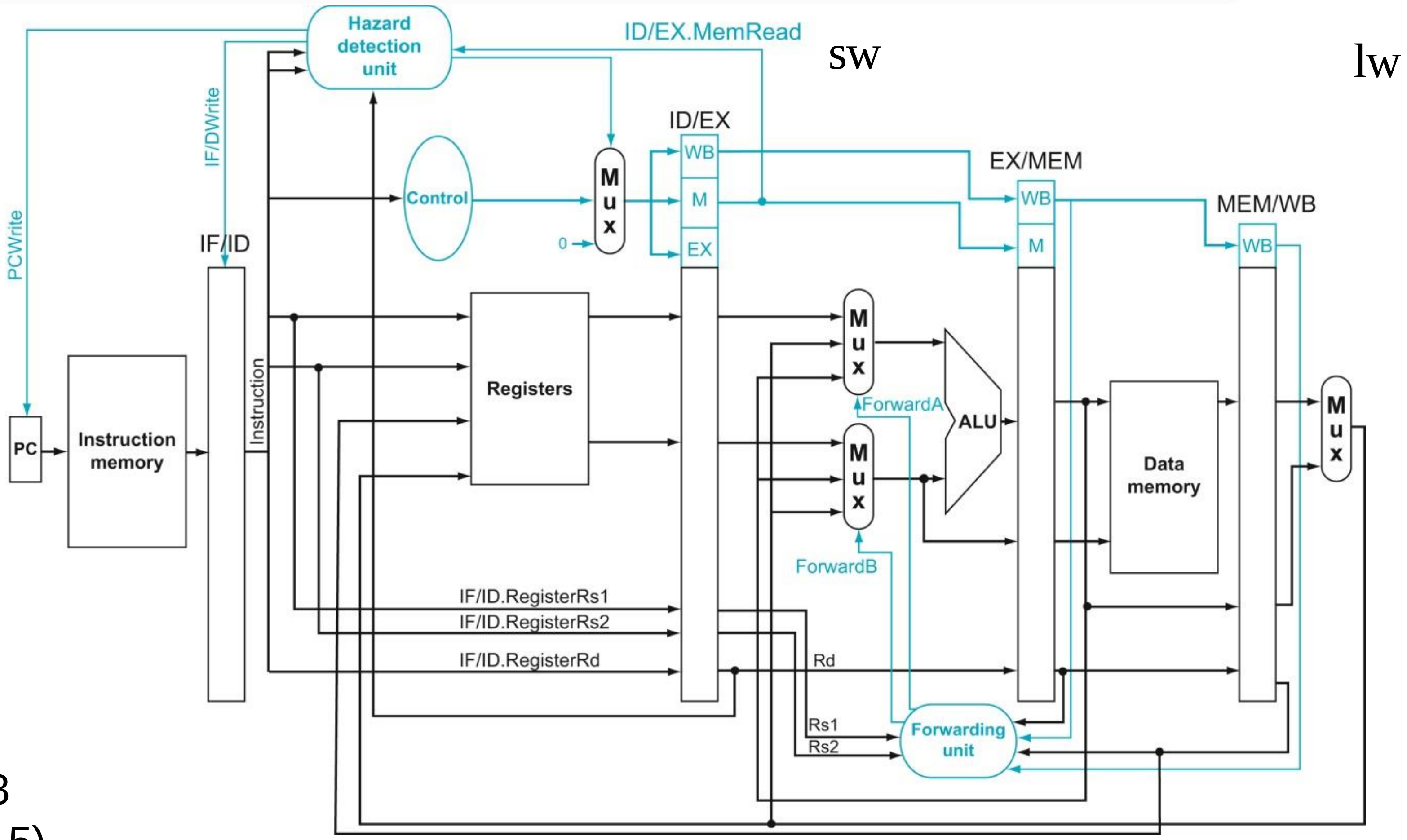
sw lw

lw	x5, 0(x15)
sw	x5, 100(x15)



MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and EX/MEM.MemWrite and MEM/WB.RegisterRd = EX/MEM.RegisterRs2

Case 2: SW is In EX Stage



lw x5, 0(x15)
add x8, x8, x8
sw x5, 100(x15)

MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and
ID/EX.MemWrite and
MEM/WB.RegisterRd = ID/EX.RegisterRs2

Case 3

- lw x15, 0(x8) # load-use,
  sw x5, 100(x15) # stall pipeline

```
lw x15, 0(x8)       IF ID EX MEM WB
sw x5, 100(x15)        IF  ID  Stall  EX    MEM  WB
```

Case 4

- R-Type followed by sw
- The result from R-Type will be saved into memory

```
e.g. add x1, x2, x3
     sw x1, 0(x4)
```

$$\begin{tikzcd}
\text{add x1,x2,x3} & \text{IF} & \text{ID} & \text{EX} \arrow[dr] & \text{MEM} & \text{WB} & \\
\text{sw x1, 0(x4)} & & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB}
\end{tikzcd}$$

Case 5

- **R-Type followed by sw**
- **Case 5 (EX/MEM Hazard): R-Type will overwrite base register for sw**

```
e.g. add x1, x2, x3
     sw x4, 0(x1)
```

```
add x1,x2,x3       IF ID EX MEM WB
sw x4, 0(x1)          IF  ID  EX    MEM  WB
```