Pipeline Data Hazards

Z. Jerry Shi
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Troubles from Pipelining

- Pipeline hazards
  - Structural hazards: attempt to use the same resource by two different instructions at the same time
  - Data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - Control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch and jump instructions, exceptions

- Pipeline control must detect and take action to resolve hazards

Data Hazards

- Data dependency may cause data hazard
  - Data dependency: an instruction depends on data produced by a previous instruction
  - Data hazard: an instruction cannot get its data in time

Example: Read after write (RAW) dependency/hazard

```
# SUB and AND depend on ADD
# read is after write
add  x1, x1, x1      # write x1
sub  x4, x1, x5      # read x1
and  x6, x1, x7
```

Data Dependency May Cause Data Hazards

- Later instructions are dependent on ADD

ava vs. needed

Time (clock cycles): 1 2 3 4 5 6 7 8 9

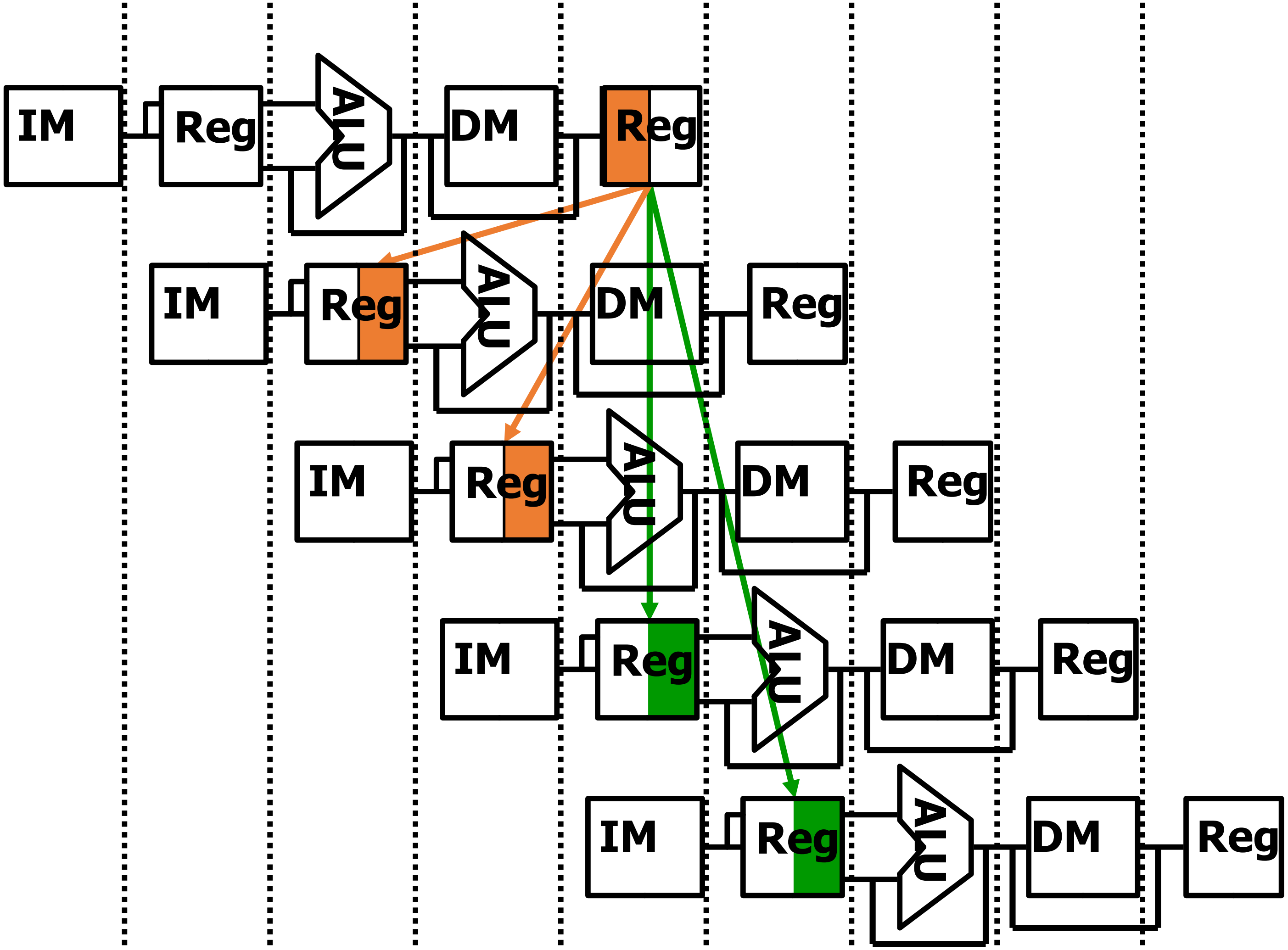
add x1,x1,x1

sub x4,x1,x5

and x6,x1,x7

or x8,x1,x9

xor x4,x1,x5



Dealing with Data Hazards

- We can usually resolve hazards by waiting, but we will try to find ways to reduce time on waiting

- Data hazards
  - Stalling (waiting)
  - Forwarding

Reading: Section 4.6 (on data hazards) and Section 4.8

Software approach

→ no operations

- Wait by inserting NOPs. Increase instruction count

```
add   x1, x2, x3
NOP
NOP
sub   x4, x1, x5
```

	1	2	3	4	5	6	7	8
add x1,x2,x3	IF	ID	EX	MEM	WB			
NOP		IF	ID	EX	MEM	WB		
NOP			IF	ID	EX	MEM	WB	
sub x4,x1,x5				IF	ID	EX	MEM	WB
and x6,x1,x7					IF	ID	EX	MEM

Hardware can detect data hazards and wait

Time (clock cycles): 1 2 3 4 5 6 7 8

load ~~add~~ x1,

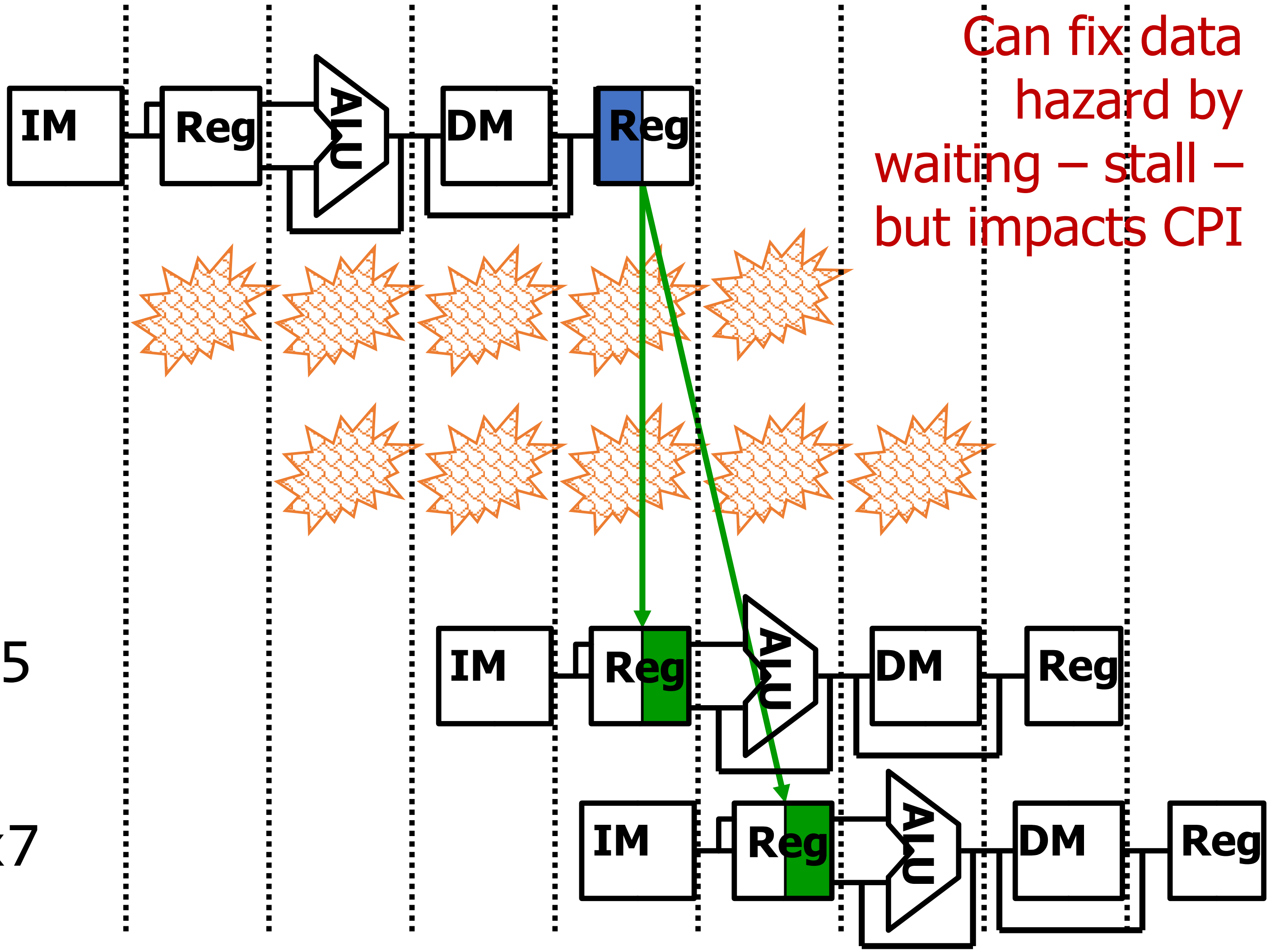
lw x1,

stall

stall

use sub x4,x1,x5

and x6,x1,x7



Can fix data hazard by waiting – stall – but impacts CPI

Stalls Shown in Pipeline Diagram

Register 1 is not updated by ADD until the first half of cycle 5

The SUB instruction cannot get the correct data until the second half of cycle 5

It waits in the ID stage in cycles 4 and 5

The AND instruction cannot get into ID stage. It stays in IF in cycles 4 and 5.

How many stall cycles do you see?

Stall for two cycles

- means "same stage"

		1	2	3	4	5	6	7	8
add	x1,x2,x3	IF	ID	EX	MEM	WB			
sub	x4,x1,x5		IF	ID	- ID	- ID	EX	MEM	WB
and	x6,x1,x7			IF	- IF	- IF	ID	EX	MEM

Ava → needed
(Produced)

Another (better) Way to “Fix” a Data Hazard

Time (clock cycles):

R-type add x1,

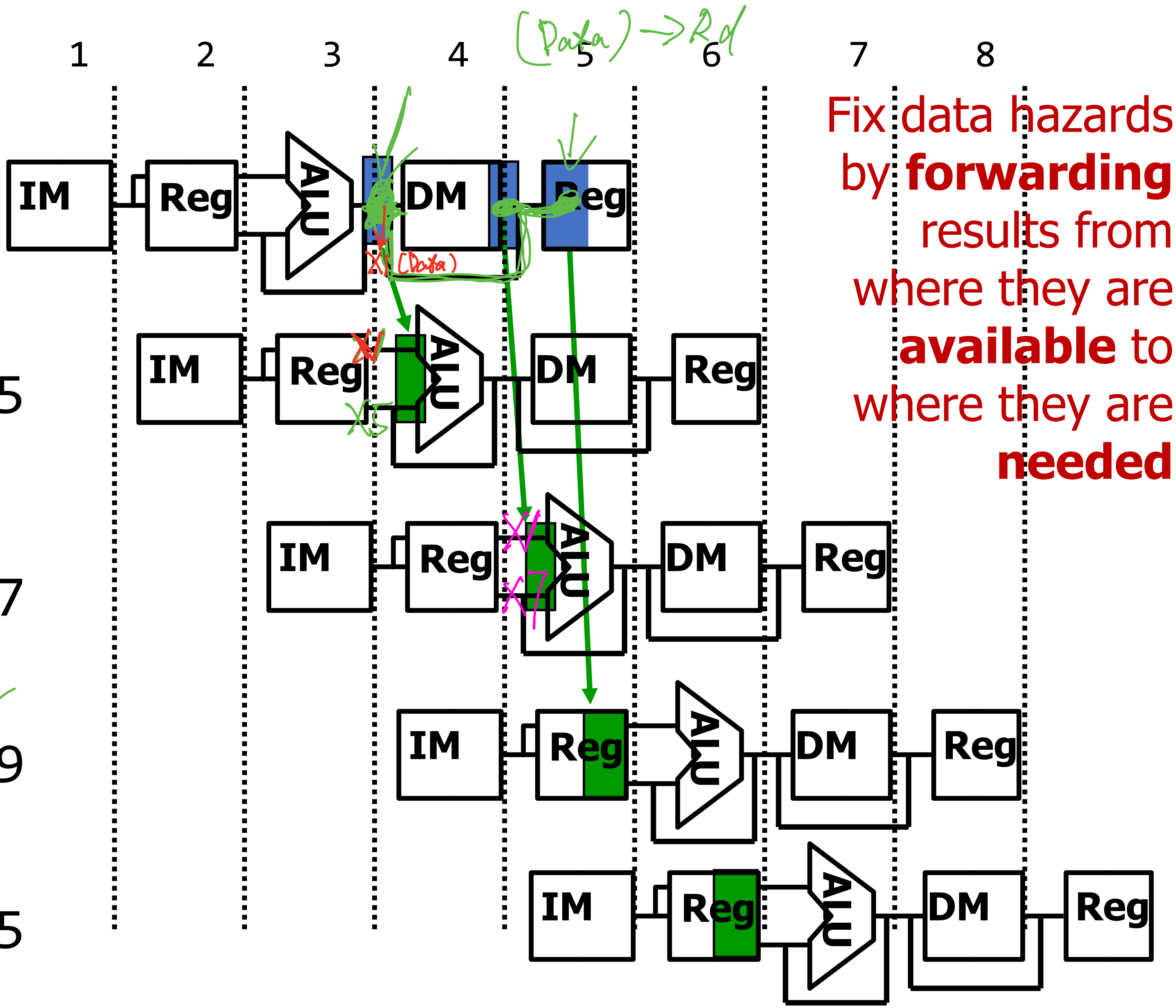
(Data)

sub x4,x1,x5

→ and x6,x1,x7

or x8,x1,x9

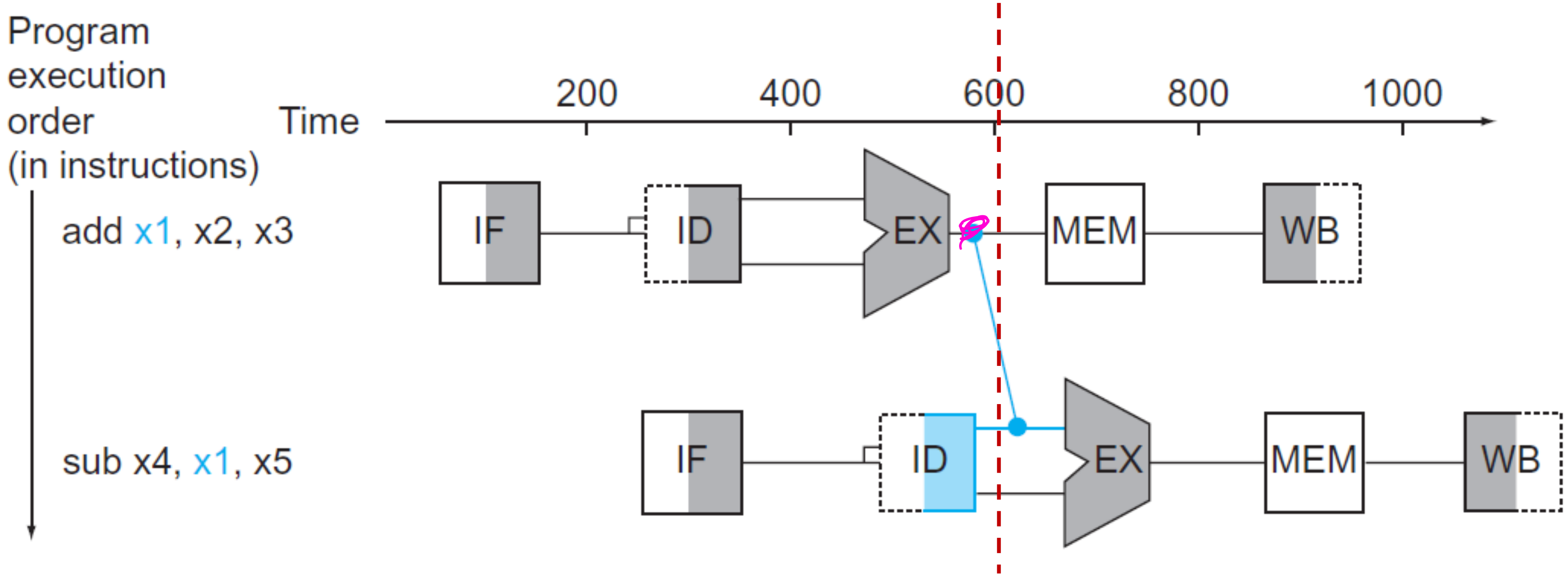
xor x4,x1,x5



Fix data hazards by **forwarding** results from where they are **available** to where they are **needed**

Forwarding (aka Bypassing)

- Use result if it is already computed and available
  - Don't wait for it to be stored in a register
  - Require extra connections in the datapath



Stale data in ID/EX
However, EX/MEM has the correct one

Pipeline Diagram Showing Forwarding

x1 is not updated in register file until the first half of cycle 5,
but it is available at the end of cycle 3

The SUB instruction can get the correct value through forwarding in cycle 4

Can AND get the correct value in ID?
Can AND be executed without stall?

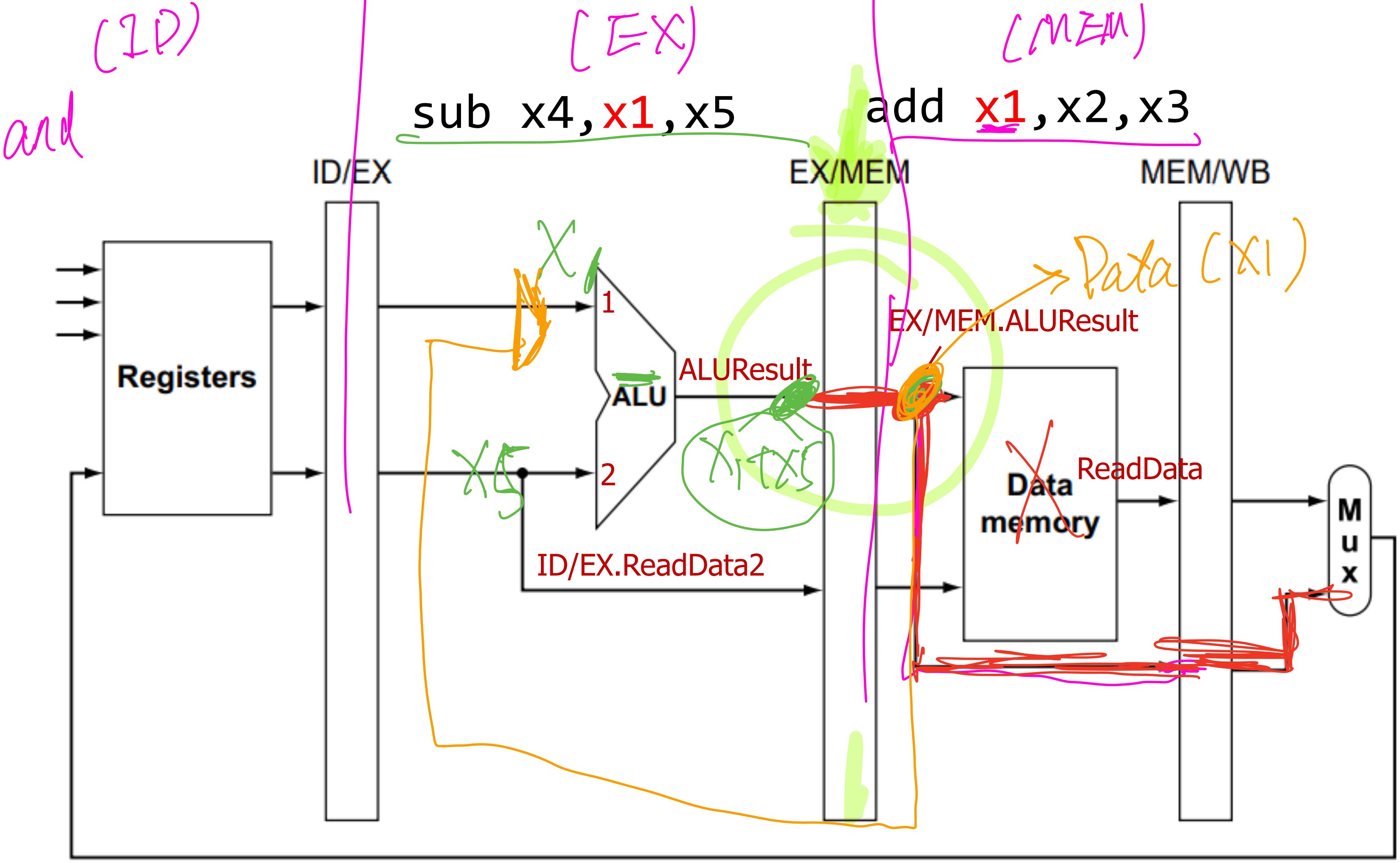
EXE

		1	2	3	4	5	6	7	8
add	x1,x2,x3	IF	ID	EX	MEM	WB			
sub	x4,x1,x5		IF	ID	EX	MEM	WB		
and	x6,x7,x1			IF	ID	EX	MEM	WB	

Adding forwarding paths

- Where would you add the path for SUB?

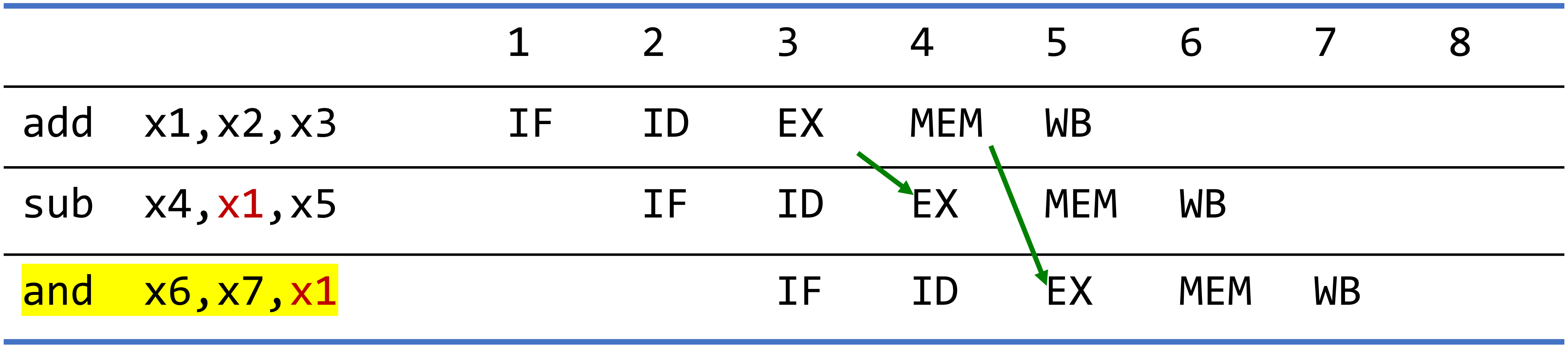
In previous cycle:
ADD was in EX, computing x2 + x3. SUB was in ID, reading x1 from register file



What about AND?

x1 is not updated in register file until the first half of cycle 5,
but it is available at the end of cycle 3

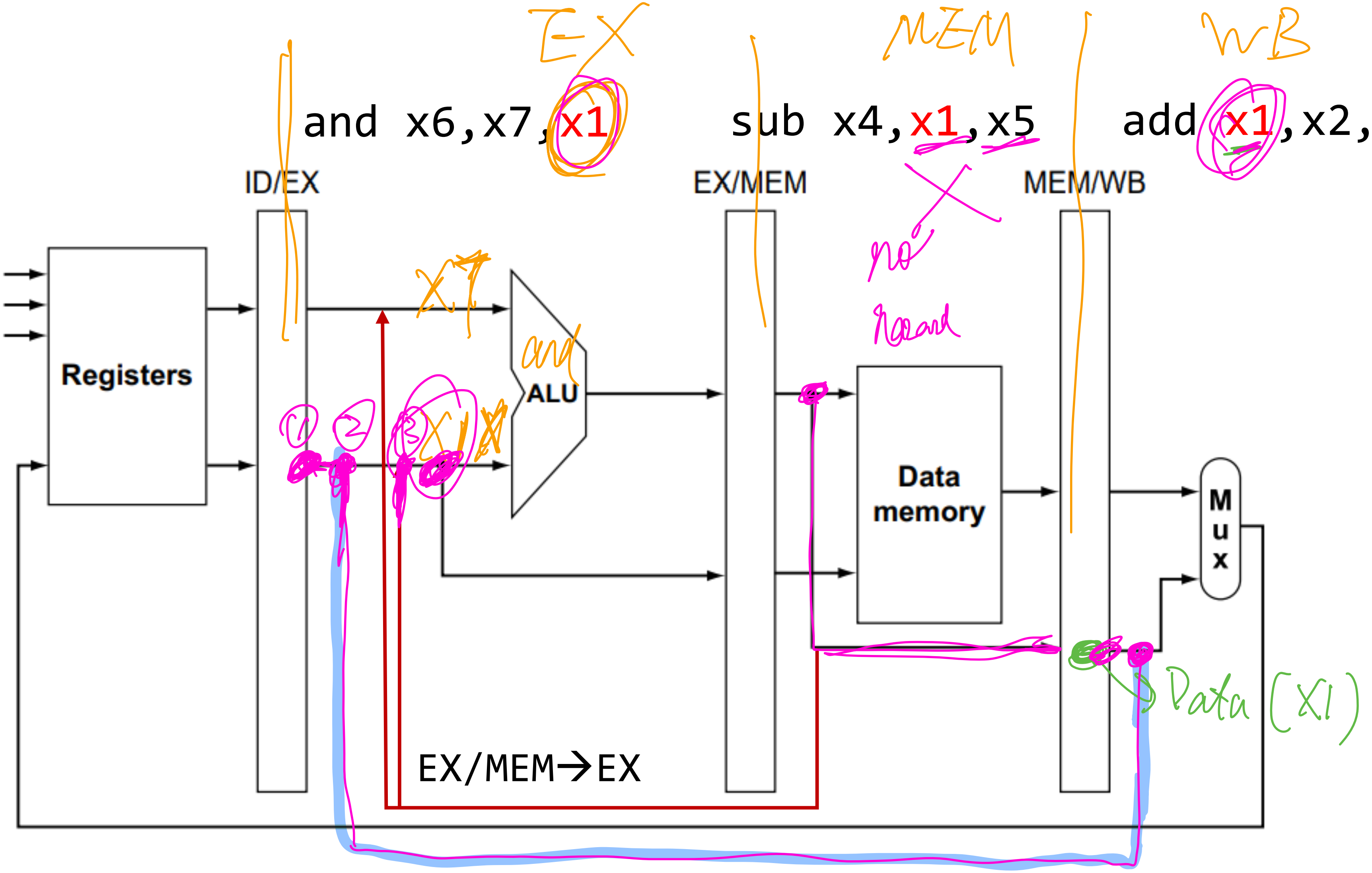
The AND instruction cannot the correct value in ID (cycle 4)
It can get the correct value through forwarding in cycle 5



	1	2	3	4	5	6	7	8
add x1,x2,x3	IF	ID	EX	MEM	WB			
sub x4,x1,x5		IF	ID	EX	MEM	WB		
and x6,x7,x1			IF	ID	EX	MEM	WB	

Adding forwarding paths - 2

- Where would you add the path for AND?



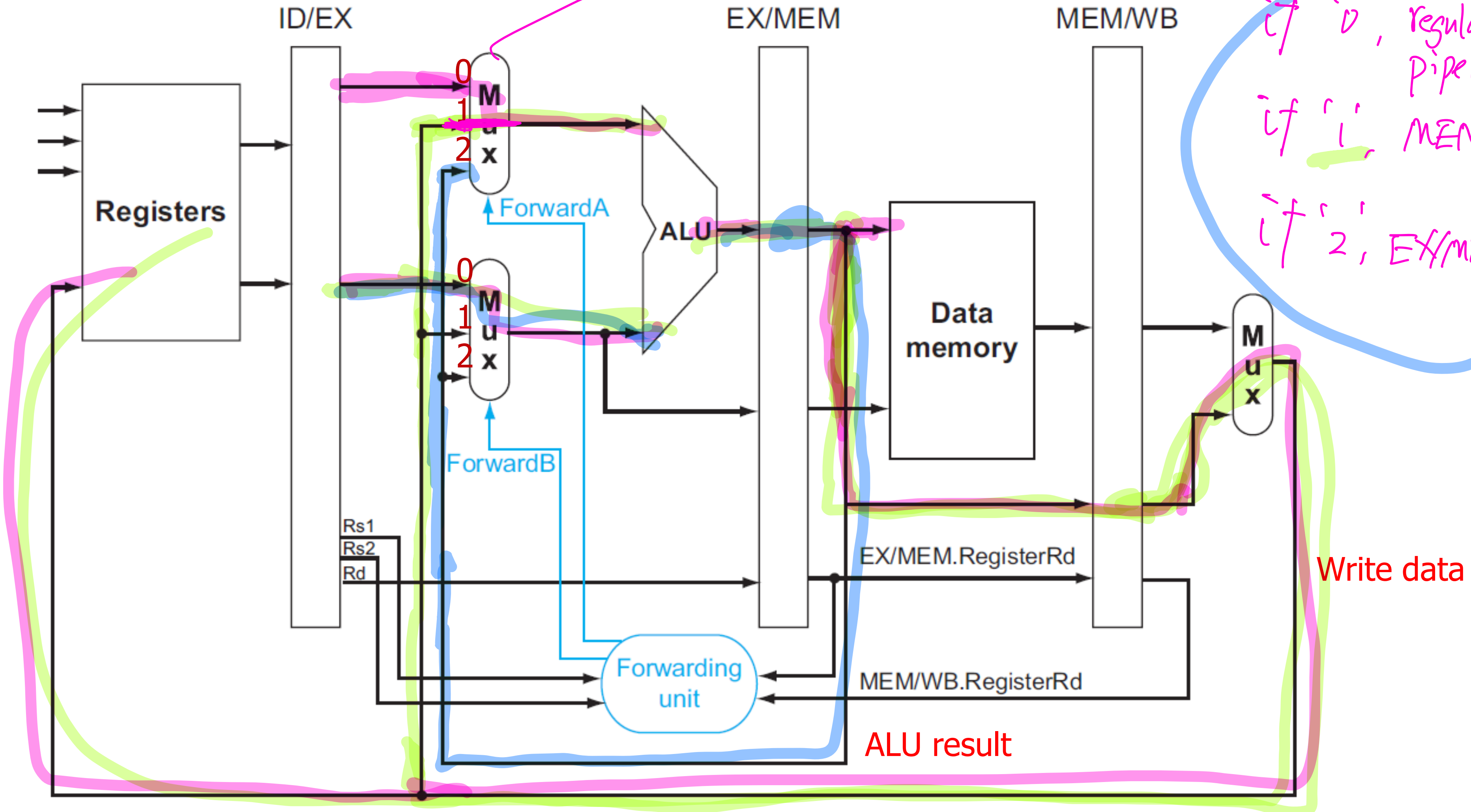
```
and x6,x7,x1      sub x4,x1,x5      add x1,x2,x3
```

EX/MEM→EX

Forwarding Paths to ALU

ALU result is stored in EX/MEM and MEM/WB,

Data loaded from data memory can also be forwarded via Write Data.



if '0', regular pipeline

if '1', MEM/WB → EX

if '2', EX/MEM → EX

b. With forwarding

MUX Added before ALU input

- A MUX is added before each input to ALU

  00: Take the value from the register file, stored in ID/EX

  10: Take the value from EX/MEM → ALU/EX

  01: Take the value from MEM/WB → ALU/EX

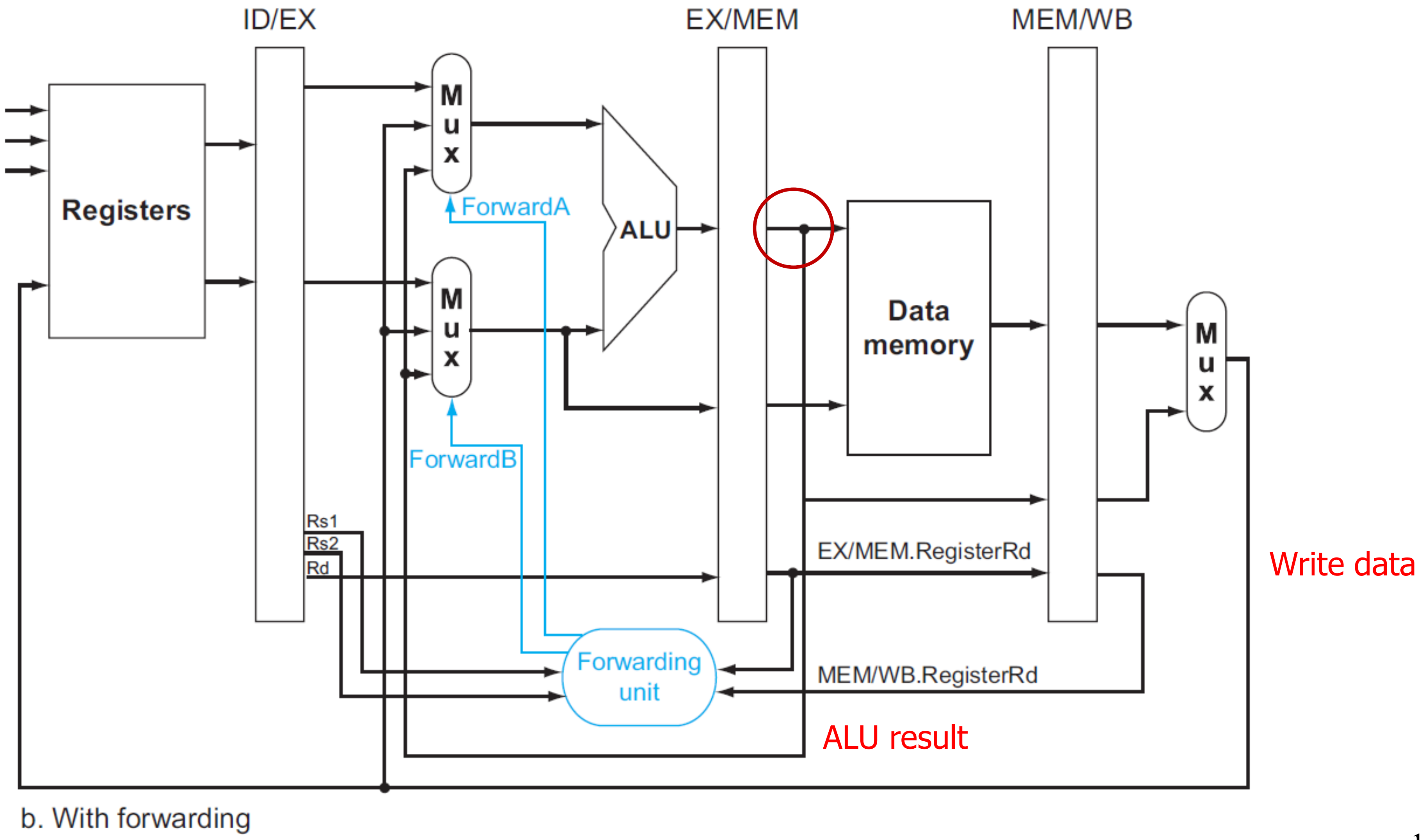
- A forwarding unit needs to generate the control signals
  - ForwardA and FowardB, each having 2 bits

When should we forward from EX/MEM to EX?

When should we forward from MEM/WB to EX?

Detecting the Need to Forward from EX/MEM to EX

When should we forward from EX/MEM to EX? Describe the logic in English.



Forwarding from EX/MEM to EX

Logic

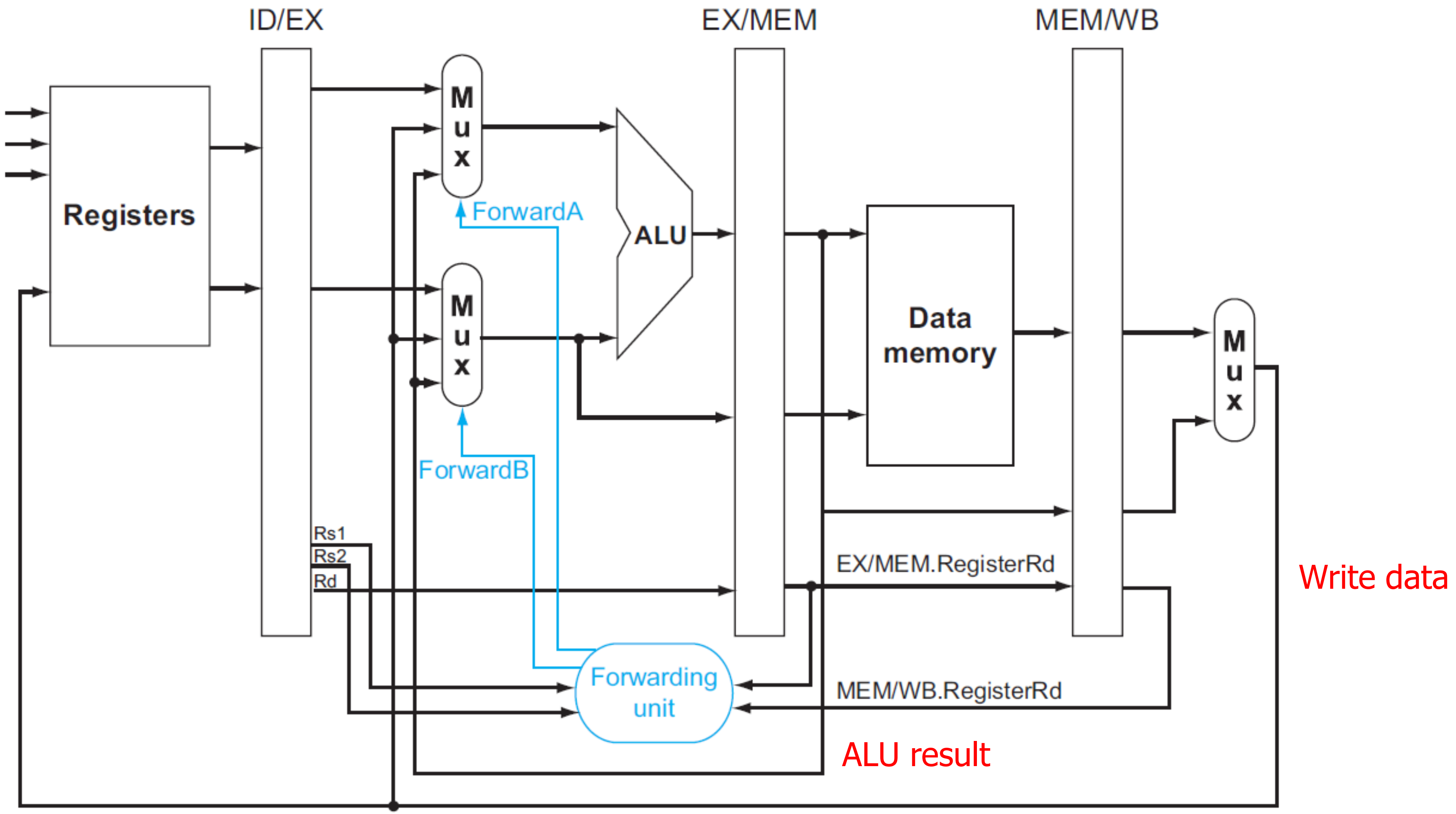
```
if     EX/MEM.RegWrite and
      (not EX/MEM.MemRead) and
       EX/MEM.rd != 0 and
       EX/MEM.rd == ID/EXE.rs1
                 x1            x1
then
       ForwardA = 0b10
```

Only R-type computes new result in EX

Similar for ForwardB

Detecting the Need to Forward from MEM/WB to EX

When should we forward from MEM/WB to EX? Describe the logic in English.



b. With forwarding

Forwarding from MEM/WB to EX

Logic

```
if      MEM/WB.RegWrite and       ←——  R-type or Load
        MEM/WB.rd != 0 and
        MEM/WB.rd == ID/EXE.rs1
                 →x1         →  x1
then
        ForwardA = 0b01
```

Similar for ForwardB

Yet Another Complication!

What if the SUB instruction writes to x1, too

Which x1 should AND use? Which forwarding path?

		1	2	3	4	5	6	7	8
add	x1,x1,x2	IF	ID	EX	MEM	WB			
sub	x1,x1,x3		IF	ID	EX	MEM	WB		
and	x1,x4,x1			IF	ID	EX	MEM	WB	

Forwarding from MEM/WB to EX (corrected)

Logic

```
if      MEM/WB.RegWrite and
        MEM/WB.rd != 0 and
        MEM/WB.rd == IE/EXE.rs1 and
        not (forward from EX/MEM)
then
        ForwardA = 0b01
```

Corrected !

Similar for ForwardB

Data Forwarding (aka Bypassing)

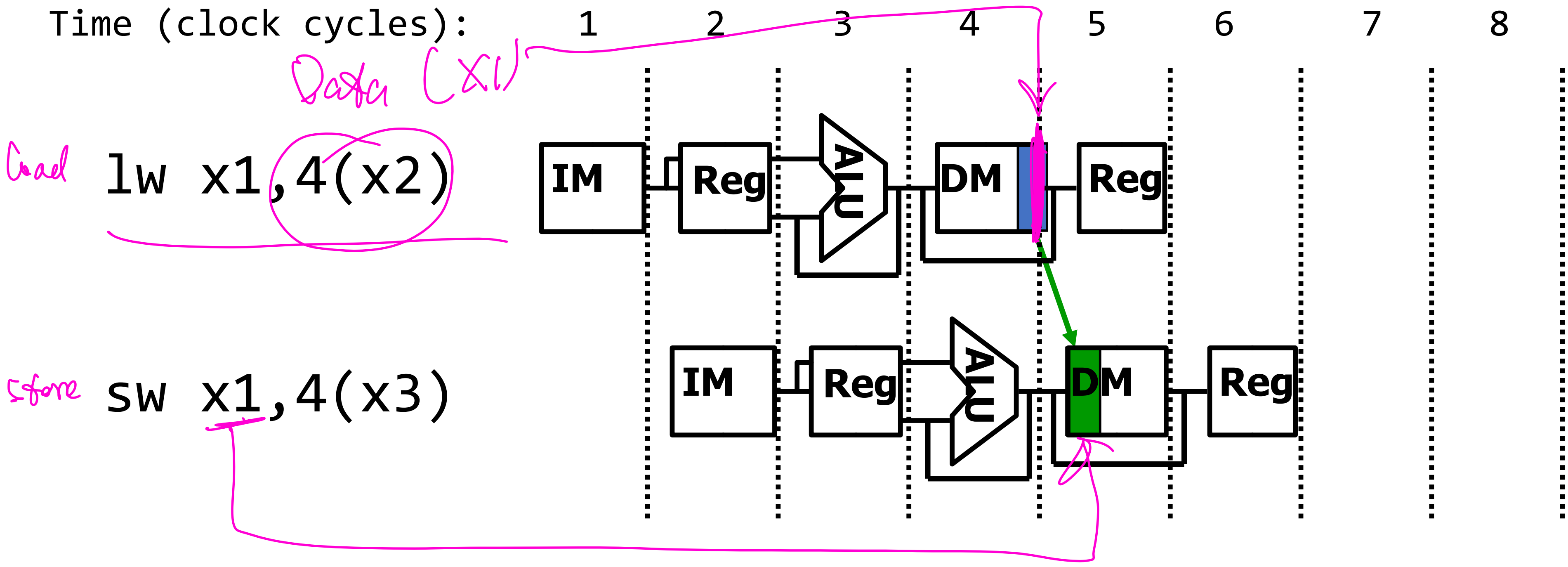
- Take data where it is available in any of the pipeline registers and forward it to the units (e.g., the ALU) that need it

- For ALU: the inputs can come from any pipeline register rather than just from ID/EX
  - Add multiplexors to the both inputs of the ALU
  - Connect ALU result in EX/MEM and Write data in WB to the MUXes
  - Add the proper forwarding logic for the MUXes
  - Pass register numbers along the pipeline and send to forwarding logic

- Other units may need similar forwarding logic (e.g., the D-Mem)

MEM/WB to MEM forwarding

- Loads that immediately followed by stores (memory-to-memory copies) can avoid a stall via forwarding
  - From the MEM/WB register to the data memory input
  - MUX and forwarding logic are added in MEM stage



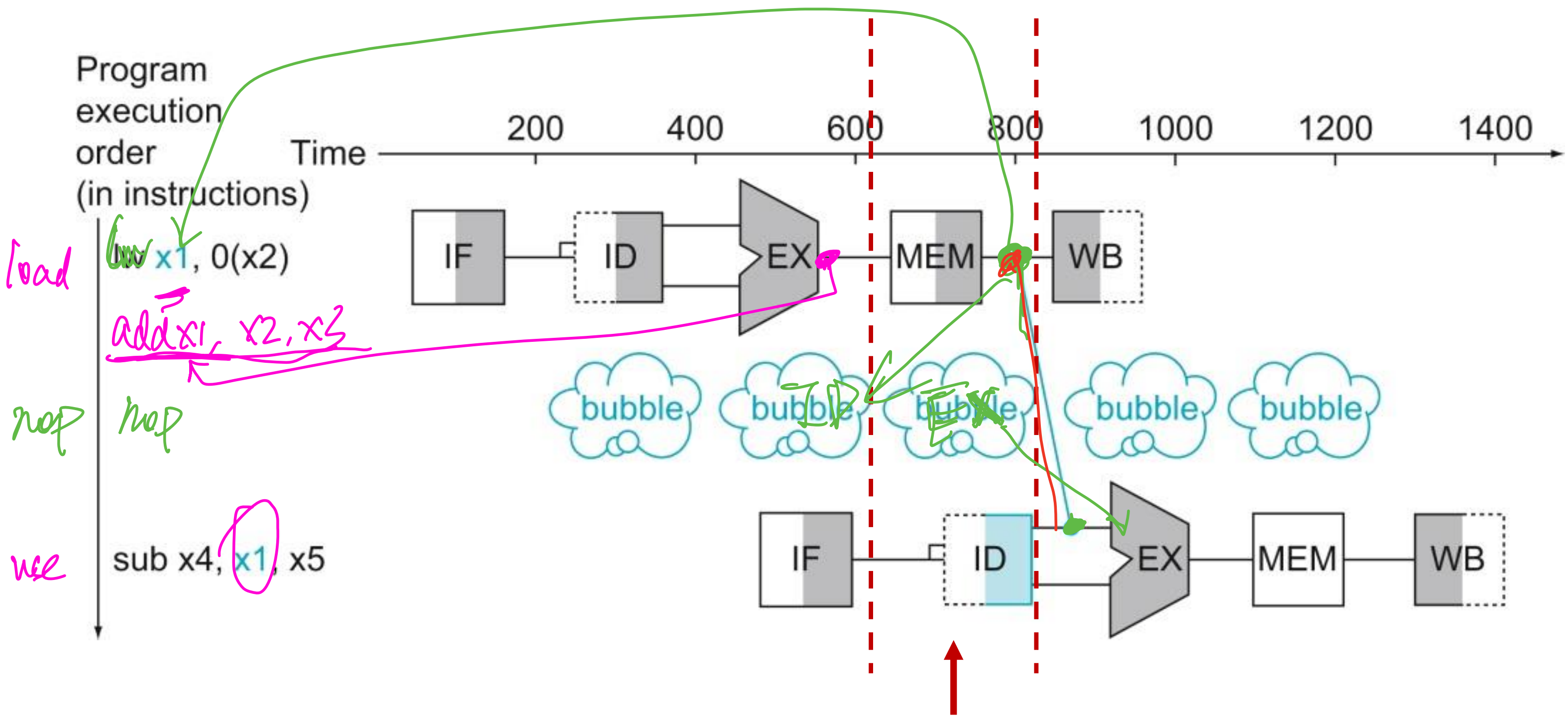
Time (clock cycles): 1 2 3 4 5 6 7 8

lw x1,4(x2)

sw x1,4(x3)

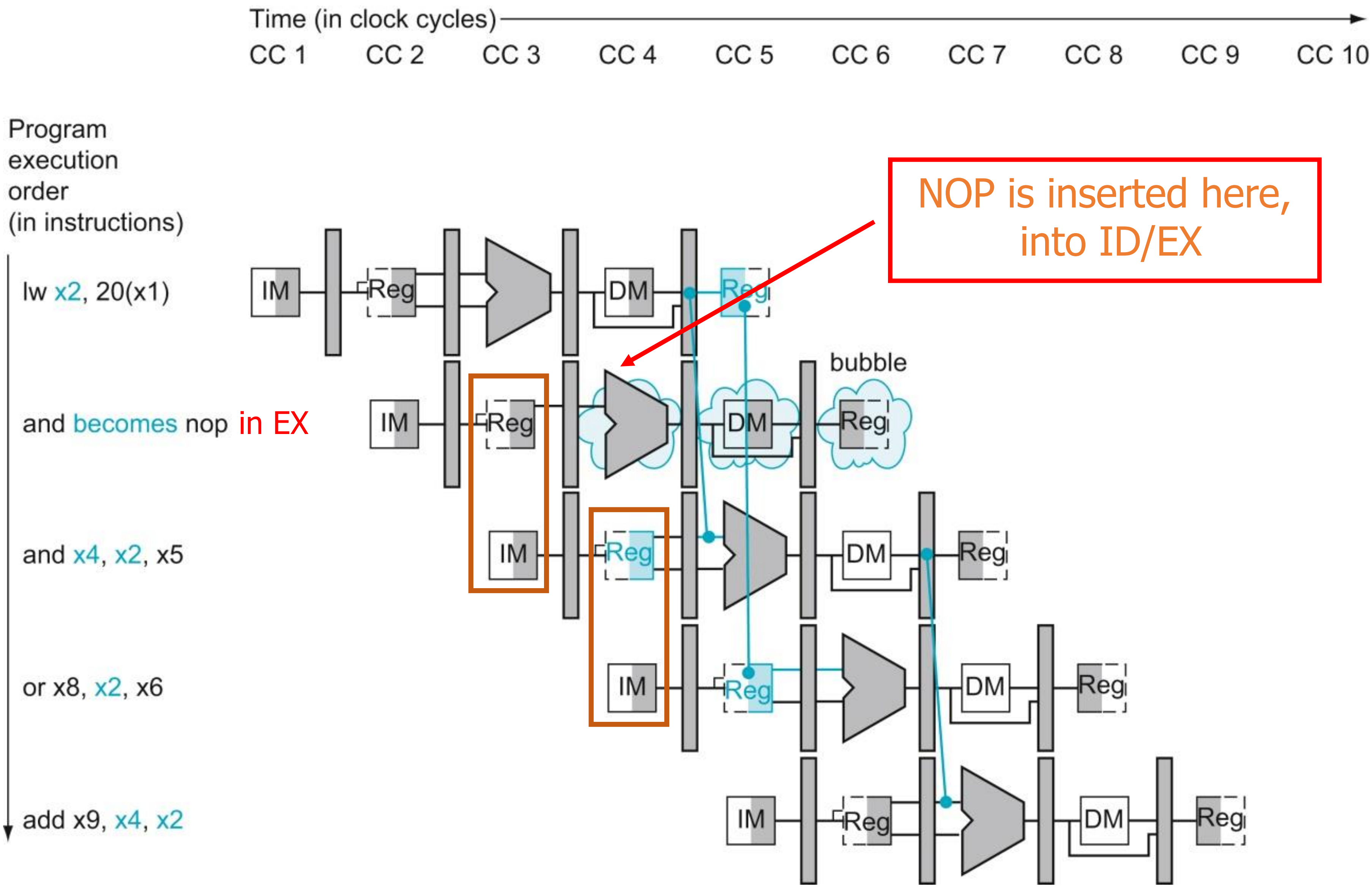
Load-Use Data Hazard

- Cannot always avoid stalls by forwarding
  - The value needed is not computed/loaded yet



SUB cannot get in Ex and use x1 in this cycle
Wait in ID (not in EX!)

Stall/Bubble in the Pipeline



Time (in clock cycles)

CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9 CC 10

Program execution order (in instructions)

lw x2, 20(x1)

and becomes nop in EX

and x4, x2, x5

or x8, x2, x6

add x9, x4, x2

NOP is inserted here, into ID/EX

Another view: Stall/Bubble in the Pipeline

- Cycle 3: hazard is detected (in ID)
- Cycle 4: IF and ID repeat. EX has a NOP. LW continues
- Cycle 5: all instructions move to the next stage

How does AND get x2 in cycle 5?

Cycle	IF	ID	EX	MEM	WB
3	`or x8,x2,x6`	`and x4,x2,x5`	`lw x2,20(x1)`		
4	`or x8,x2,x6`	`and x4,x2,x5`	NOP	`lw x2,20(x1)`	
5	`add x9,x4,x2`	`or x8,x2,x6`	`and x4,x2,x5`	NOP	`lw`

Yet another view, with pipeline diagram

The result of lw is not available until the end of cycle 4

The AND instruction cannot enter the EX stage in cycle 4.

In cycle 3:

- The hazard is detected in ID
- A bubble (NOP) is inserted in the pipeline (stored in ID/EX)

In cycle 4:

- The EX stage does NOP (the inserted bubble)
- The AND instruction stays in the ID stage in cycle 4
- The OR instruction stays in the IF stage

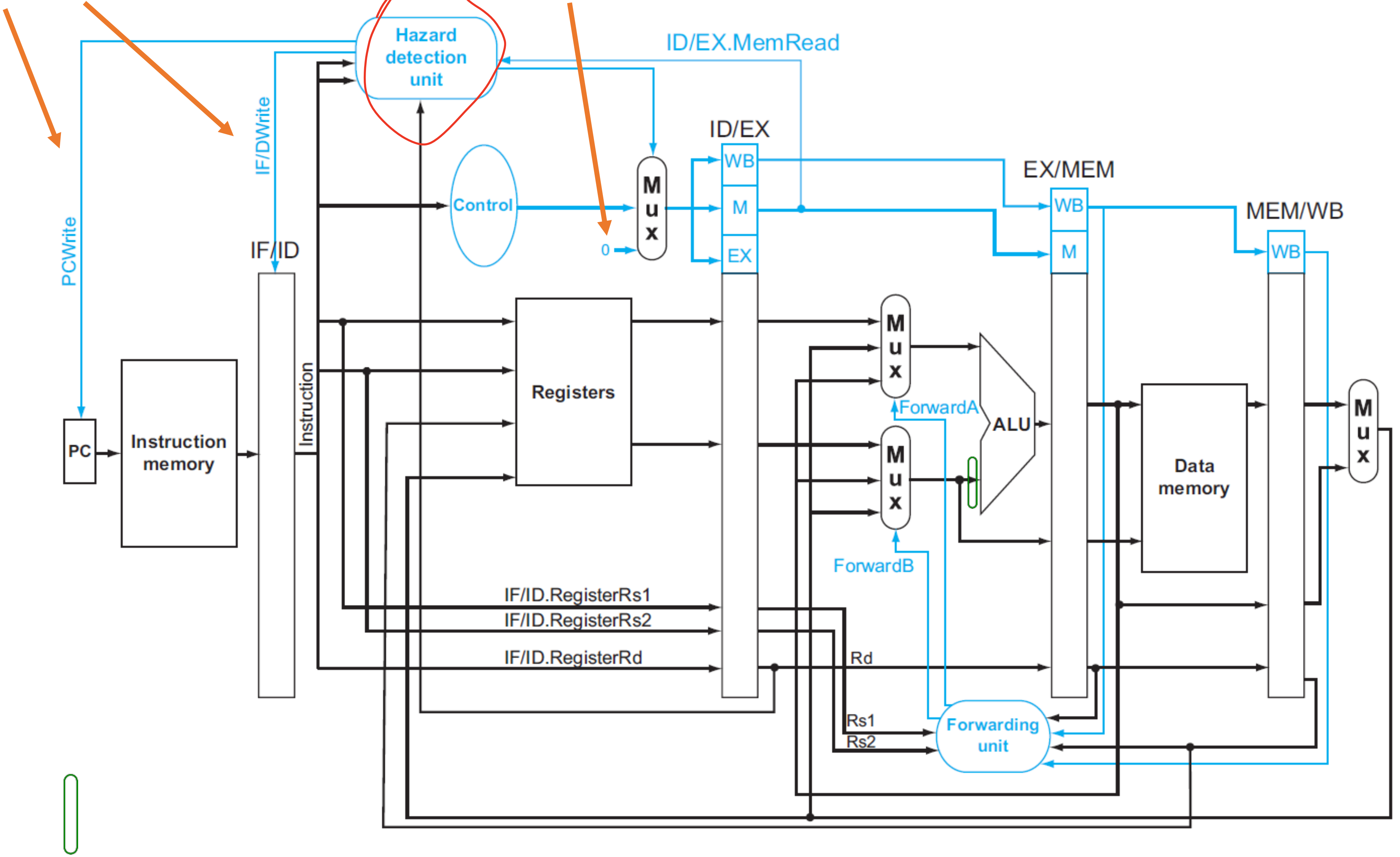
	1	2	3 EX	4	5	6	7	8
lw x2,20(x1)	IF	ID	EXE	MEM	WB			
and x4,x2,x5		IF	ID	- ID	EXE	MEM	WB	
or x8,x2,x6			IF	- ID IF	ID	EXE	MEM	WB

Datapath with Hazard Detection

: Insert NOP

IF and ID repeat on Hazards

All control signals are set to 0



MUX before ALU input 2 for Selecting imm

Performance impact - 1

One cycle is wasted for each load-use hazard. → forwarding Unit

If 50% of the load instructions cause load-use hazard, what is the average number of stall cycles per each load instruction?

(load-use) (no load-use)

overhead

50% × 1 + 50% × 0 = 0.5

Performance impact - 2

One cycle is wasted for each load-use hazard   Forwarding

The ideal CPI is 1.

Suppose 30% of the instructions are load and 60% of them cause load-use hazard.

What is the average CPI of all instructions?

$$1 + [30\% \times 60\% \times 1 \cancel{+ 70\% \times 0}$$

$$= \text{ideal} + \text{overhead} =$$

Code Scheduling to Avoid Stalls

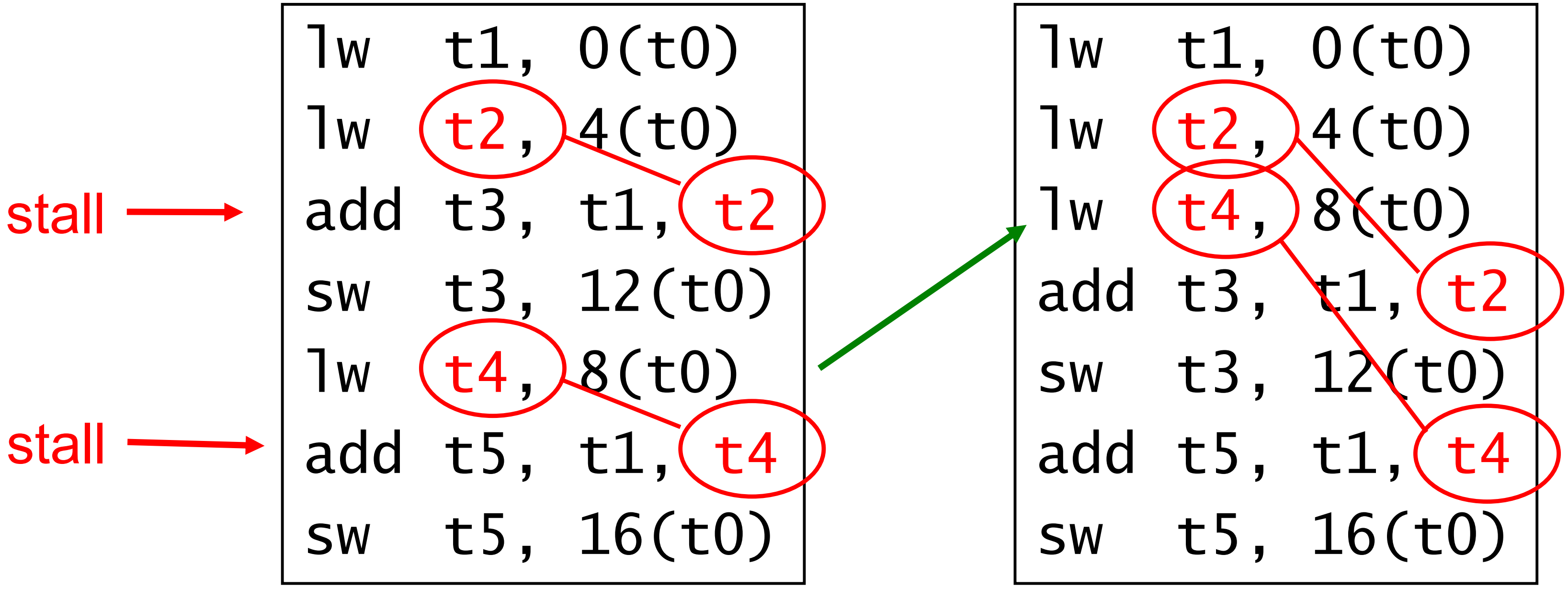
- Reorder code to avoid use of load result in the next instruction

Consider code:

```
A = B + E; C = B + F;
```

Instructions are

Load B, load E, compute A, store A. Load F, compute C, store C.



```
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)
```

9 cycles

```
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)
```

7 cycles

Draw pipeline diagrams yourself.

Summary – Handling Data Hazards

One of the reasons we cannot achieve the ideal speedup is hazards

Handling data hazards (RAW in 5-stage RISC-V ~~MIPS~~ pipeline)

- Forwarding can eliminate many hazards
  - ALU-to-ALU (to both ALU input ports) RAW
  - Memory-to-ALU (to both ALU input ports) RAW, Load-use
  - Memory-to-Memory (to the memory Write Data port) lw sw
    - Load followed by store
- Still, pipeline needs to support stall
  - Not all hazards can be avoided by forwarding (Load-use)
- Code scheduling can avoid some stalls

- Study the remaining slides yourself

Common mistakes

	1	2	3	4	5	6	7	8
I1	IF	ID	EXE	MEM	WB			
I2		IF	ID	-	EX	MEM	WB	
I3			IF	-	ID	EX	-	MEM
I4			-	-	IF			

?

I4 is not fetched until cycle 5
'-' means repeat

Hazard detection is in ID
No stalls in EX, MEM, WB

Writing pipeline diagram

- Recognize and handle data dependence
  - Understand when data are generated and when they are used

- Two instructions cannot be in the same stage in the same cycle

- Detect hazard and wait, if necessary, in ID stage
  - Once an instruction enters EX, it does not stop until it's completed
  - IF only fetches instructions. It does not check instructions

- Do not stall unnecessarily

Summary of forwarding to the next instruction

Where data is generated	Where data is consumed	Forwarding paths
EX	EX	EX/MEM to EX
EX	MEM	EX/MEM to EX
MEM	EX	Stall
MEM	MEM	MEM/WB to MEM

For each case, find an example consisting of two instructions.
Explain the execution of the instructions.

1. On processors without forwarding, how many stall cycles are needed?
2. On processors with forwarding, how does the second instruction get the operand?

Question

- If there is no forwarding, how many stall cycles does the following instructions have?

RAW

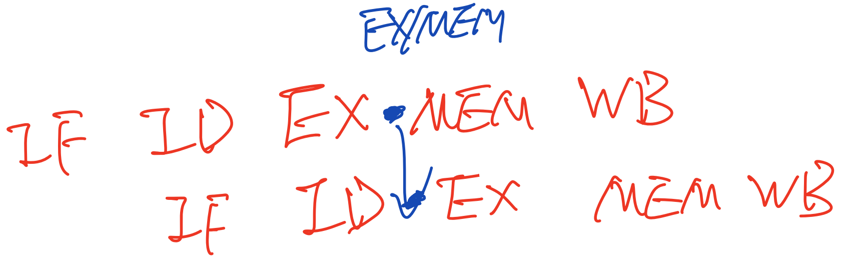
2

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

Question

- With forwarding, which forwarding path does the sub instruction use?

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



EX/MEM

IF ID EX MEM WB

IF ID EX MEM WB

[✓ A. EX/MEM to EX]

B. MEM/WB to EX

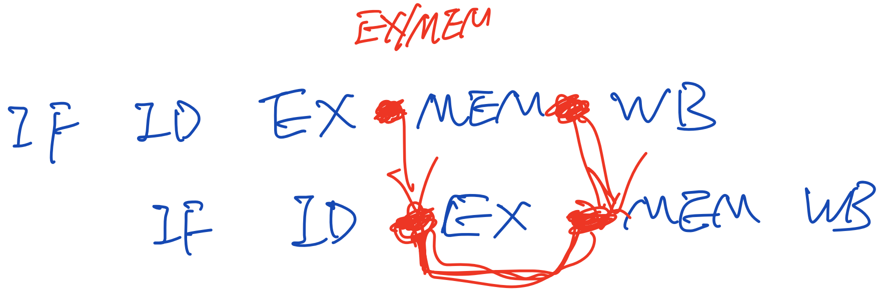
C. MEM/WB to MEM

D. None

Question

- With forwarding, which forwarding path does the sw instruction use?

```
sub   x2, x1, x4
sw    x2, (x10)
```



[✓ A. EX/MEM to EX] A is better

B. MEM/WB to EX

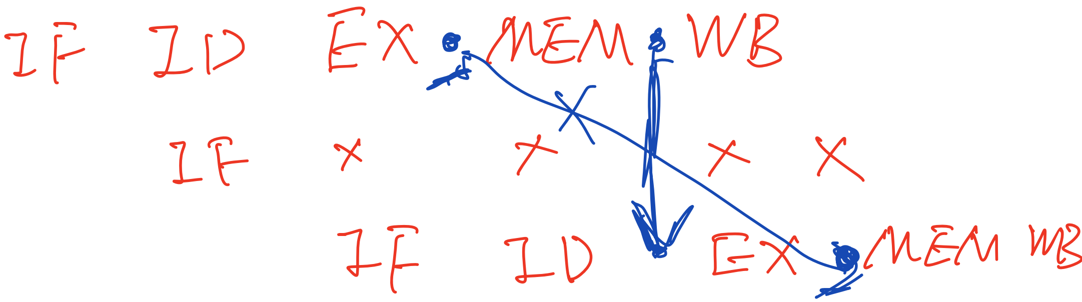
[✓ C. MEM/WB to MEM] ✓

D. None

Question

- With forwarding, which forwarding path does the sw instruction use?

```
sub   x2, x1, x4
nop
sw    x2, (x10)
```



A. EX/MEM to EX

[✓ B.] MEM/WB to EX

C. MEM/WB to MEM

D. None

Question

- With forwarding, which forwarding path does the sub instruction use?

```
lw    x1, (x2)
nop
sub   x2, x1, x4
```

```
A. EX/MEM to EX
B. MEM/WB to EX
C. MEM/WB to MEM
D. None
```

Load-use Hazard Detection Unit in ID

Detect Hazard in ID.

The logic is

```
stall =
ID/EX.MemRead  and    # instructions in EX is a load
ID/EX.rd != 0  and    # it does not load into x0
( (IF/ID.rs1 == ID/EX.rd)  ||
 ((IF/ID.rs2 == ID/EX.rd) && ! MemWrite))
```

If the current instruction in ID
is SW, no need to stall for rs2

Forward from MEM/WB to MEM

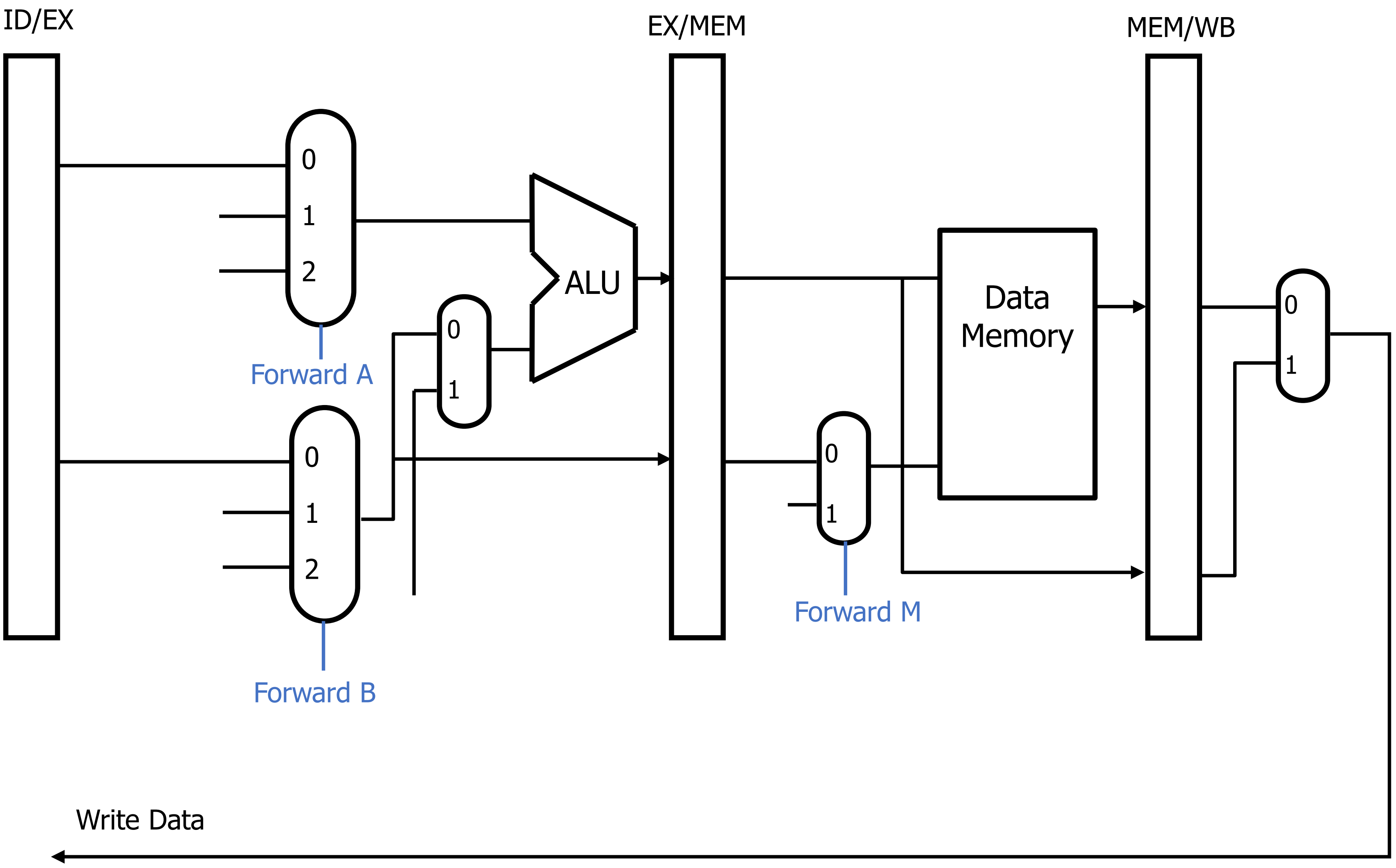
From MEM/WB to MEM

ADD a MUX to select data going to D-MEM.WriteData
EX/MEM.ReadData2 or MEM/WB.WriteData ?

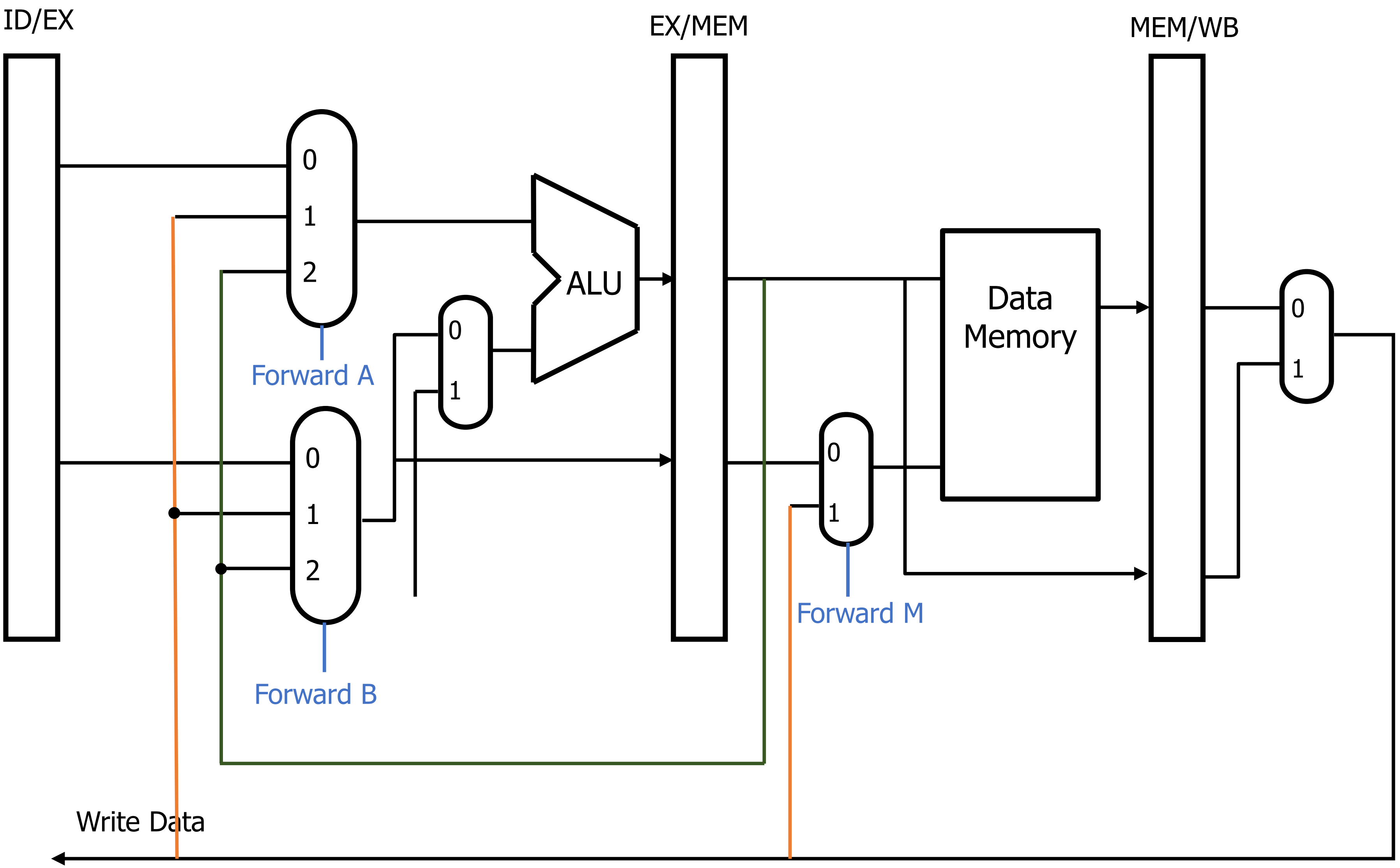
rs2 only

```
if      MEM/WB.rd != 0 and
        MEM/WB.rd == EX/MEM.rs2 and
        MEM/WB.MemRead and
        EX/MEM.MemWrite
then
        ForwardM = 1    # forward from MEM/WB to MEM
```

Complete the forwarding paths



EX and MEM stages with forwarding paths


ID/EX
EX/MEM
MEM/WB
0
1
2
Forward A
ALU
0
1
0
1
2
Forward B
Data Memory
0
1
Forward M
0
1
Write Data