CSC 3210 Computer organization and programming

Chapter 4

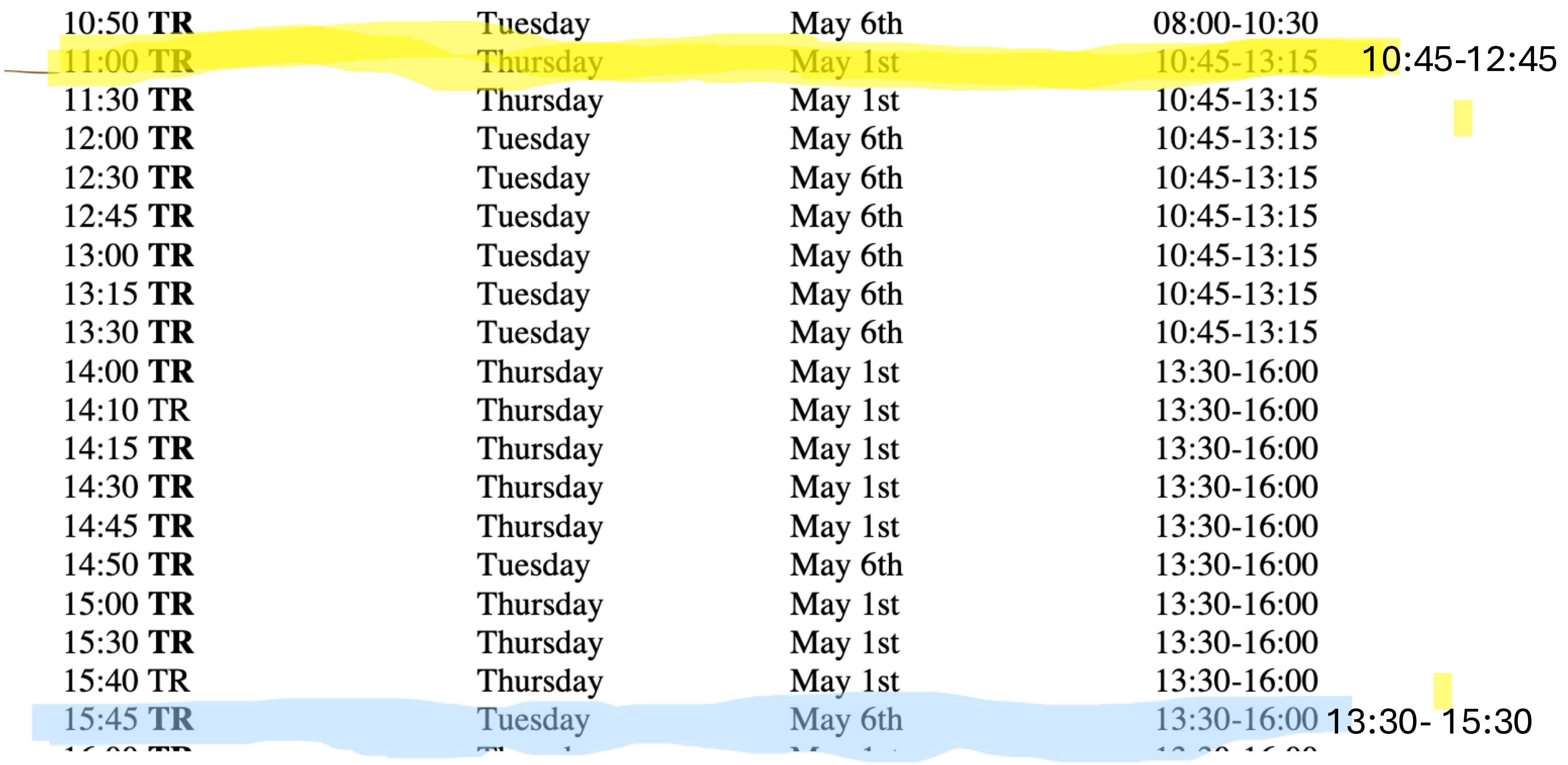
Chunlan Gao

GeorgiaState University

Final

- **Cheat sheet**: you can have one page (two-sides), handwritten(no copy), submit after the test with your name on it.
- Lectures after 9(included): **slides, quizzes, assignments.**
- Part A: Multiple Choice (30 points)
- Part B: True / False (30 points)
- Part C: Short Answer (40 points))[write response]
- Part D: Bonus (20 points)[write response]
- Appendix: Images and Figures(You are going to use in the questions)

Test time (two hours test) :

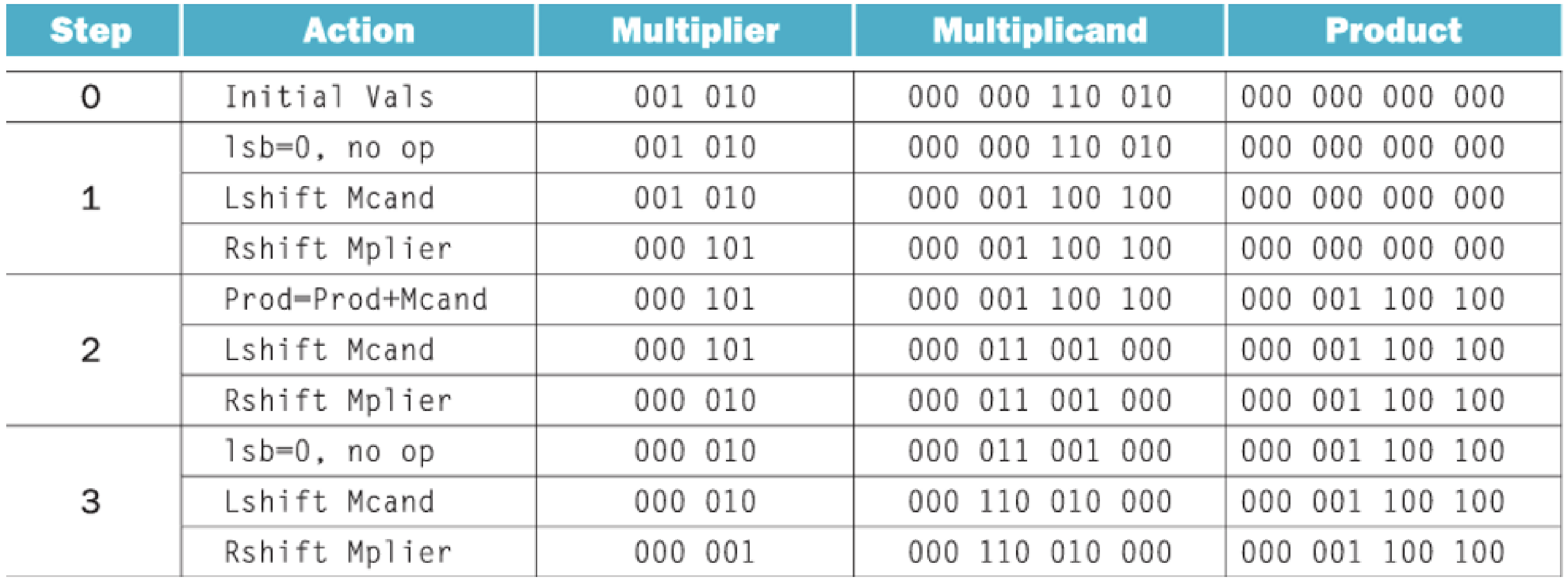


10:50 **TR**	Tuesday	May 6th	08:00-10:30	
11:00 **TR**	Thursday	May 1st	10:45-13:15	10:45-12:45
11:30 **TR**	Thursday	May 1st	10:45-13:15	
12:00 **TR**	Tuesday	May 6th	10:45-13:15	
12:30 **TR**	Tuesday	May 6th	10:45-13:15	
12:45 **TR**	Tuesday	May 6th	10:45-13:15	
13:00 **TR**	Tuesday	May 6th	10:45-13:15	
13:15 **TR**	Tuesday	May 6th	10:45-13:15	
13:30 **TR**	Tuesday	May 6th	10:45-13:15	
14:00 **TR**	Thursday	May 1st	13:30-16:00	
14:10 TR	Thursday	May 1st	13:30-16:00	
14:15 **TR**	Thursday	May 1st	13:30-16:00	
14:30 **TR**	Thursday	May 1st	13:30-16:00	
14:45 **TR**	Thursday	May 1st	13:30-16:00	
14:50 **TR**	Tuesday	May 6th	13:30-16:00	
15:00 **TR**	Thursday	May 1st	13:30-16:00	
15:30 **TR**	Thursday	May 1st	13:30-16:00	
15:40 TR	Thursday	May 1st	13:30-16:00	
15:45 **TR**	Tuesday	May 6th	13:30-16:00	13:30- 15:30

Assignment 3_Question1

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

62 octal= 110 010
12 octal = 001 010



Step	Action	Multiplier	Multiplicand	Product
0	Initial Vals	001 010	000 000 110 010	000 000 000 000
1	lsb=0, no op	001 010	000 000 110 010	000 000 000 000
	Lshift Mcand	001 010	000 001 100 100	000 000 000 000
	Rshift Mplier	000 101	000 001 100 100	000 000 000 000
2	Prod=Prod+Mcand	000 101	000 001 100 100	000 001 100 100
	Lshift Mcand	000 101	000 011 001 000	000 001 100 100
	Rshift Mplier	000 010	000 011 001 000	000 001 100 100
3	lsb=0, no op	000 010	000 011 001 000	000 001 100 100
	Lshift Mcand	000 010	000 110 010 000	000 001 100 100
	Rshift Mplier	000 001	000 110 010 000	000 001 100 100

Assignment3_Question2

- As discussed in the text, one possible performance enhancement is to do a shift and add instead of an actual multiplication. Since 9 × 6, for example, can be written (2 × 2 × 2 + 1) × 6, we can calculate 9 × 6 by shifting 6 to the left three times and then adding 6 to that result. Show the best way to calculate 0x33 × 0x55 using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

- use same algorithm:

- x33= 0011 0011two = 2^5 + 2^4 +2^1 + 1 = 51ten
- x55 = 0101 0101two = 2^6+ 2^4 + 2^2+1

- x33 * x55 = 51ten*(2^6+ 2^4 + 2^2+1)
- 1. shift 0011 0011two left by 6 = 0011 0011 0000 00two
- 2. shift 0011 0011two left by 4 =. 00 1100 1100 00 two
- 3. shift 0011 0011two left by 2 = 0011 0011 00 two
- 4. 00 1100 11two
- 1+2+3+4 = xxxxxxxxxx
- Find the final hexadecimal value

Review

- In last class, we talked about the pipelined control, single-cycle pipeline diagram, multiple-cycle pipeline and and Detecting the Need to Forward.

Single-Cycle Pipeline Diagram

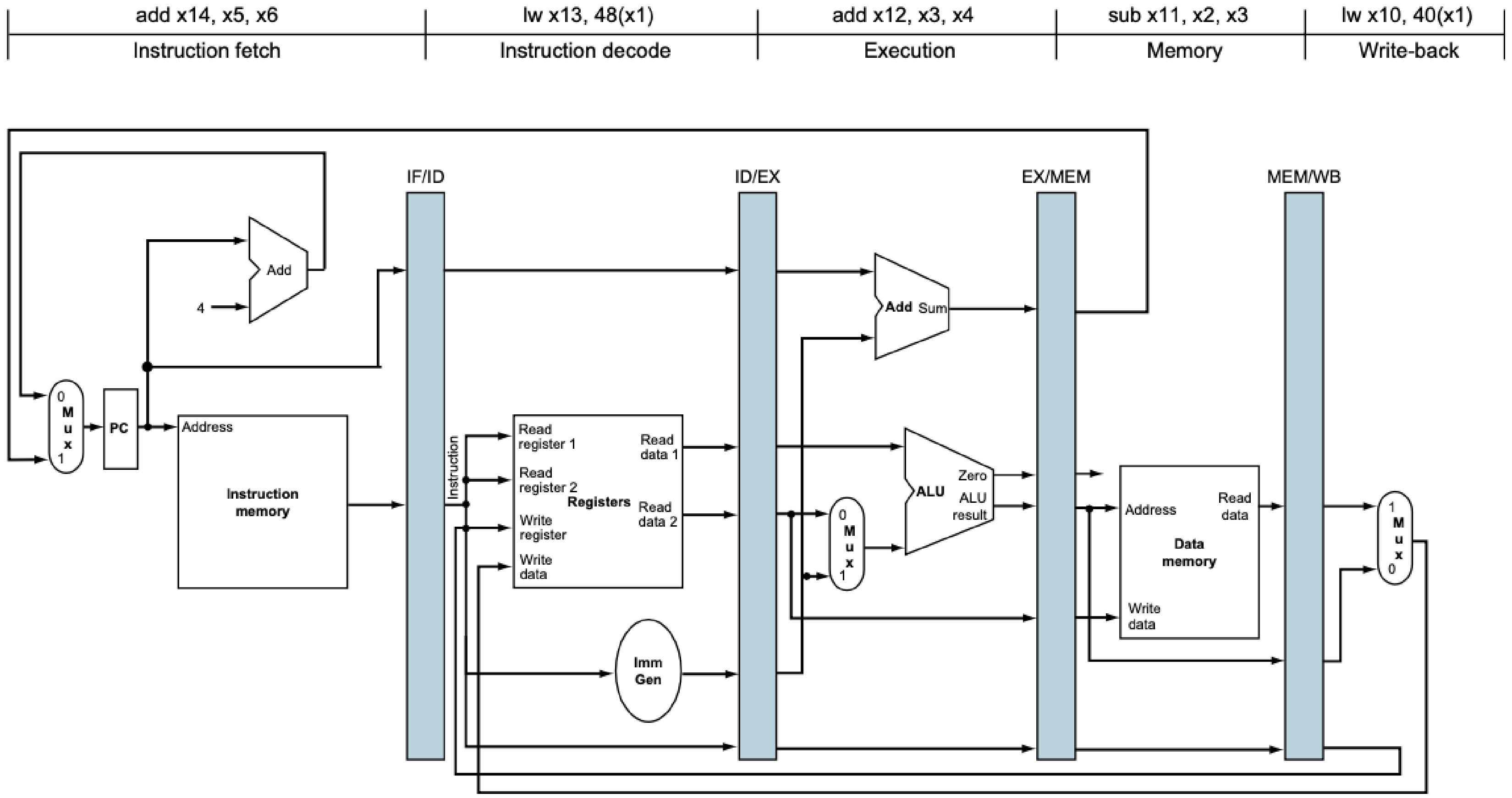


FIGURE 4.47 The single-clock-cycle diagram corresponding to clock cycle 5 of the pipeline in Figures 4.45 and 4.46. As you can see, a single-clock-cycle figure is a vertical slice through a multiple-clock-cycle diagram.

Multiple-clock-cycle pipeline diagram of five instructions

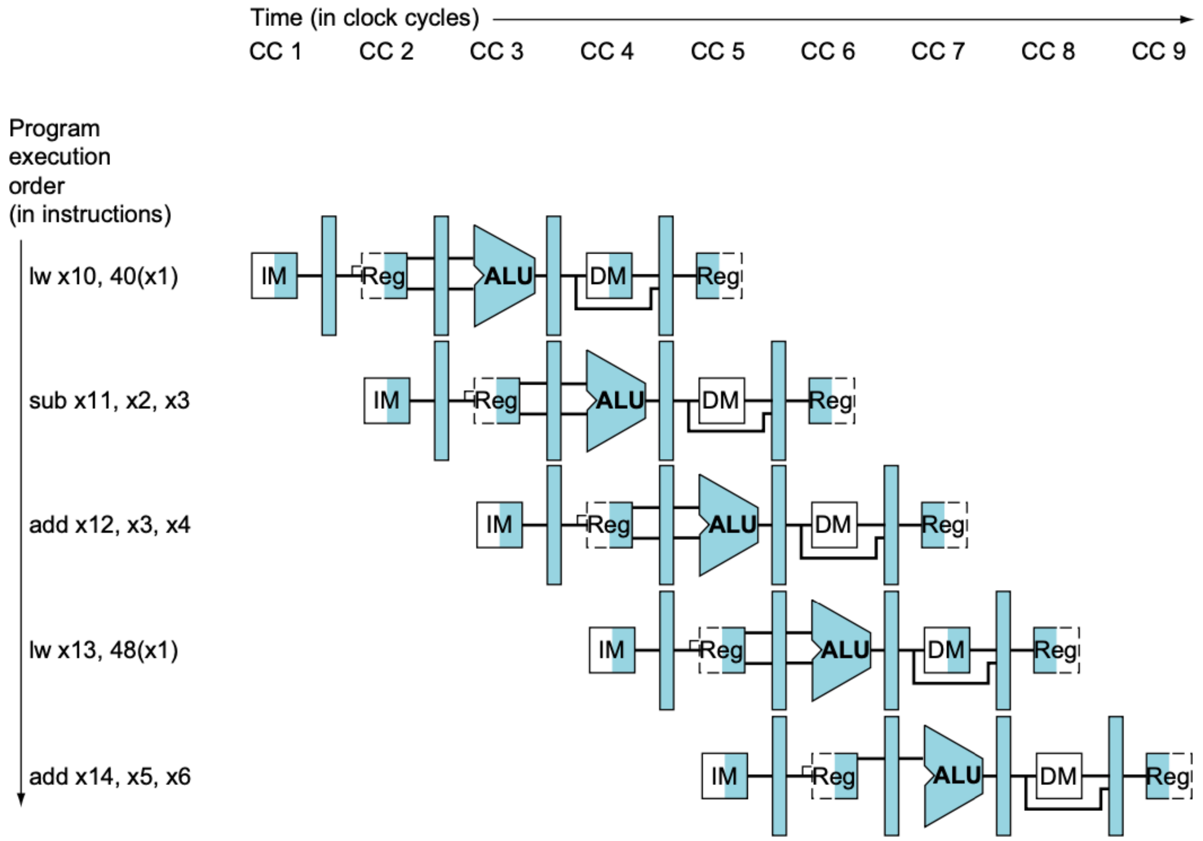
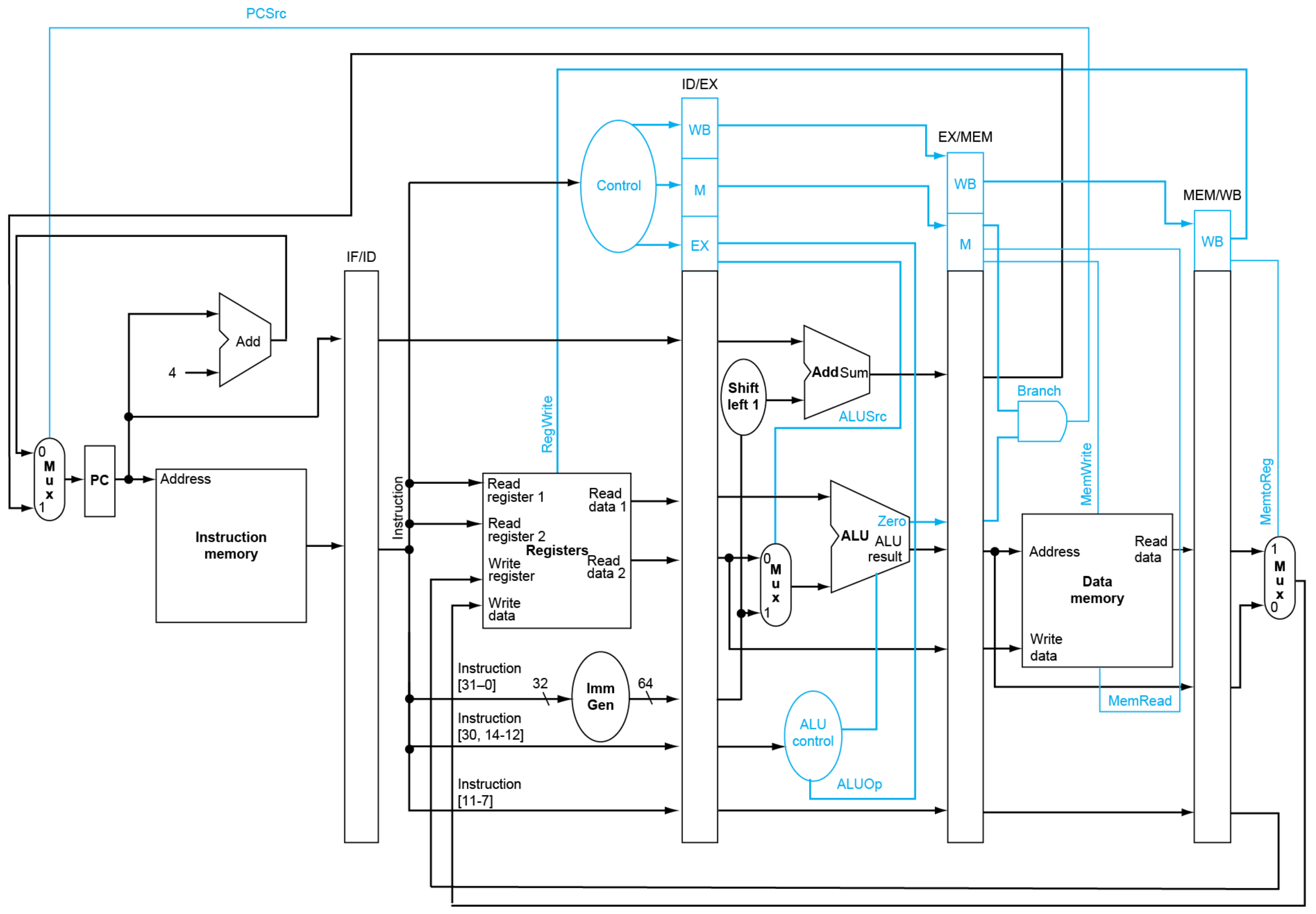


FIGURE 4.45 Multiple-clock-cycle pipeline diagram of five instructions. This style of pipeline representation shows the complete execution of instructions in a single figure. Instructions are listed in instruction execution order from top to bottom, and clock cycles move from left to right. Unlike Figure 4.26, here we show the pipeline registers between each stage. Figure 4.59 shows the traditional way to draw this diagram.

Pipelined Control



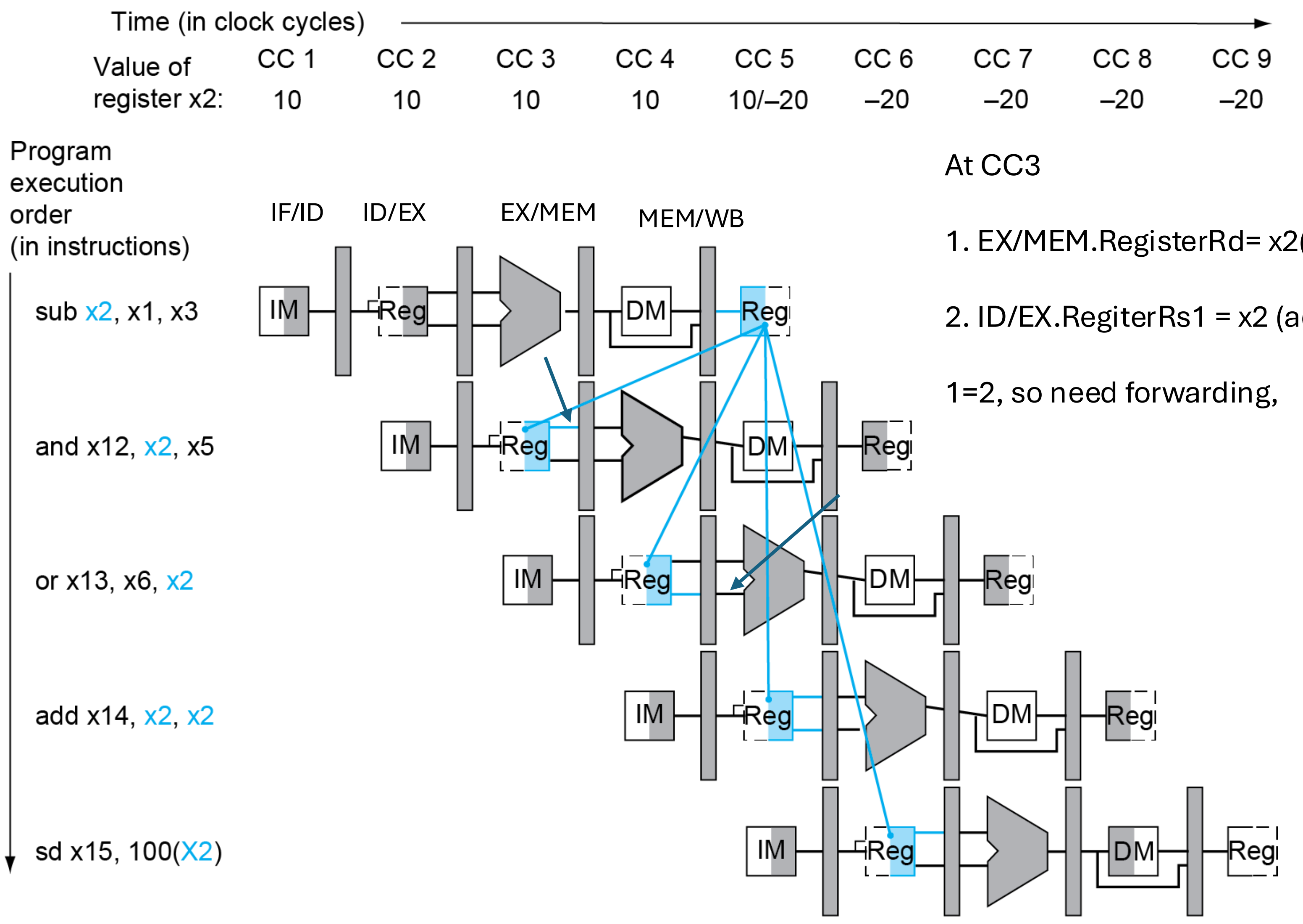
Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs1 = register number for Rs1 sitting in ID/EX pipeline register
- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs1, ID/EX.RegisterRs2
- Data hazards when

  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRs2
  } Fwd from EX/MEM pipeline reg

  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs1
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRs2
  } Fwd from MEM/WB pipeline reg

Forwarding



At CC3

1. EX/MEM.RegisterRd= x2(sub)

2. ID/EX.RegiterRs1 = x2 (add)

1=2, so need forwarding,

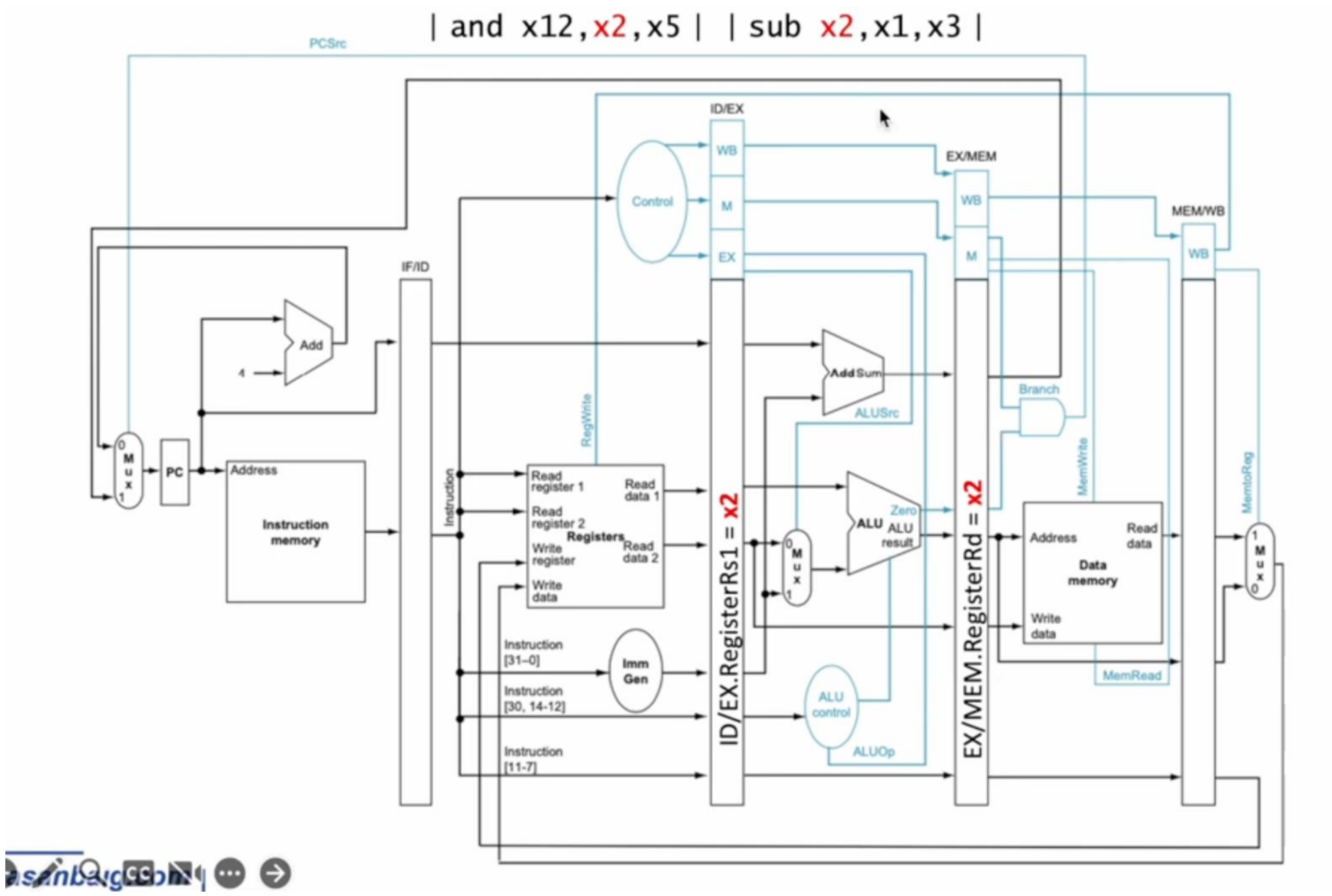
At CC4

1. MEM/WB.RegisterRd= x2(sub)

2. ID/EX.RegiterRs2 = x2 (add)

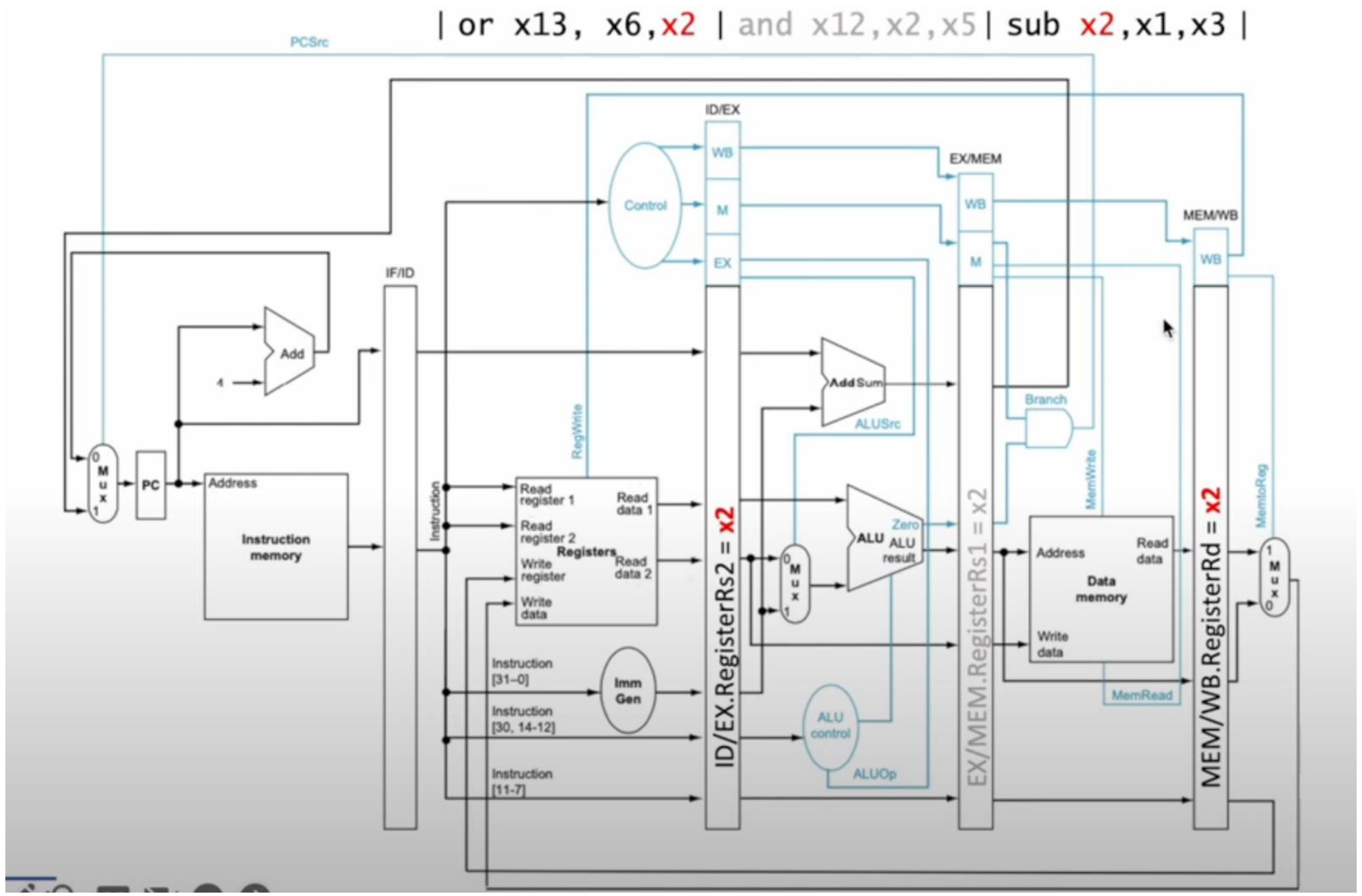
1=2, so need forwarding,

Detecting the Need to Forward



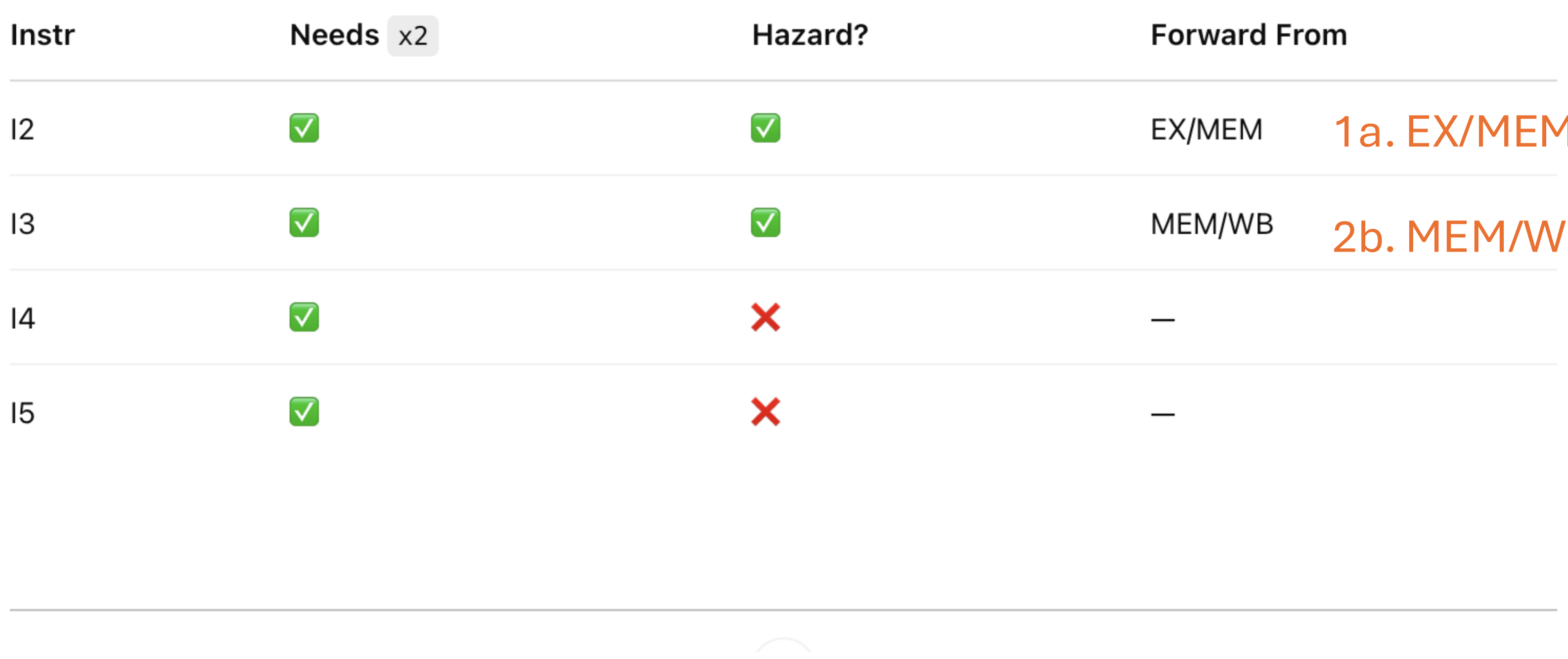
1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1
1b. EX/MEM.RegisterRd = ID/EX.RegisterRs2
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs1
2b. MEM/WB.RegisterRd = ID/EX.RegisterRs2

Detecting the Need to Forward



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

Conclusion



Instr	Needs x2	Hazard?	Forward From
I2	✅	✅	EX/MEM
I3	✅	✅	MEM/WB
I4	✅	❌	—
I5	✅	❌	—

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1

2b. MEM/WB.RegisterRd = ID/EX.RegisterRs2

Detecting the Need to Forward

Rs1
Rs2

check if any data dependency here?

1. sw x1,40(x3)

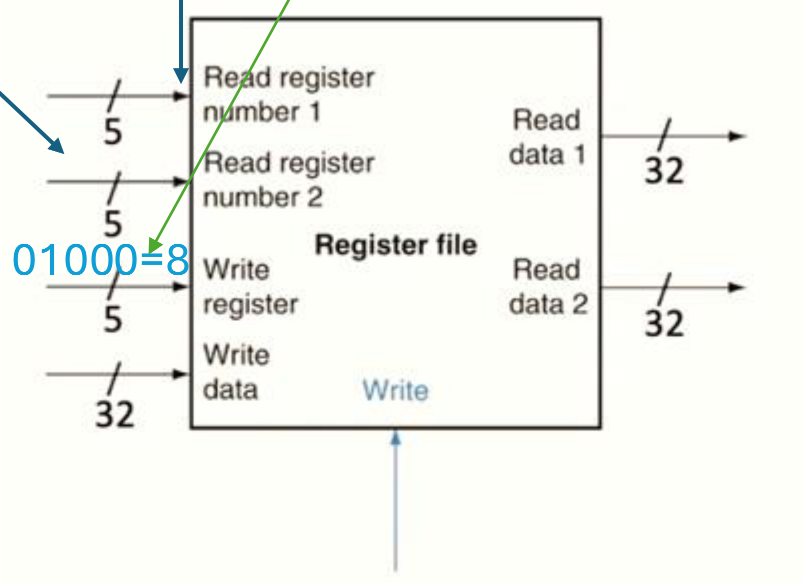
   add x2, x4,x8

Rd rs2 rs1

IM[11,5]	rs2	rs1	f3	IM[4,0]	op

fun7	rs2	rs1	fun3	rd	op

40 = 000000 101000



- if instruction will write to a register by checking:
- EX/MEM.RegWrite, MEM/WE.RegWrite

Check

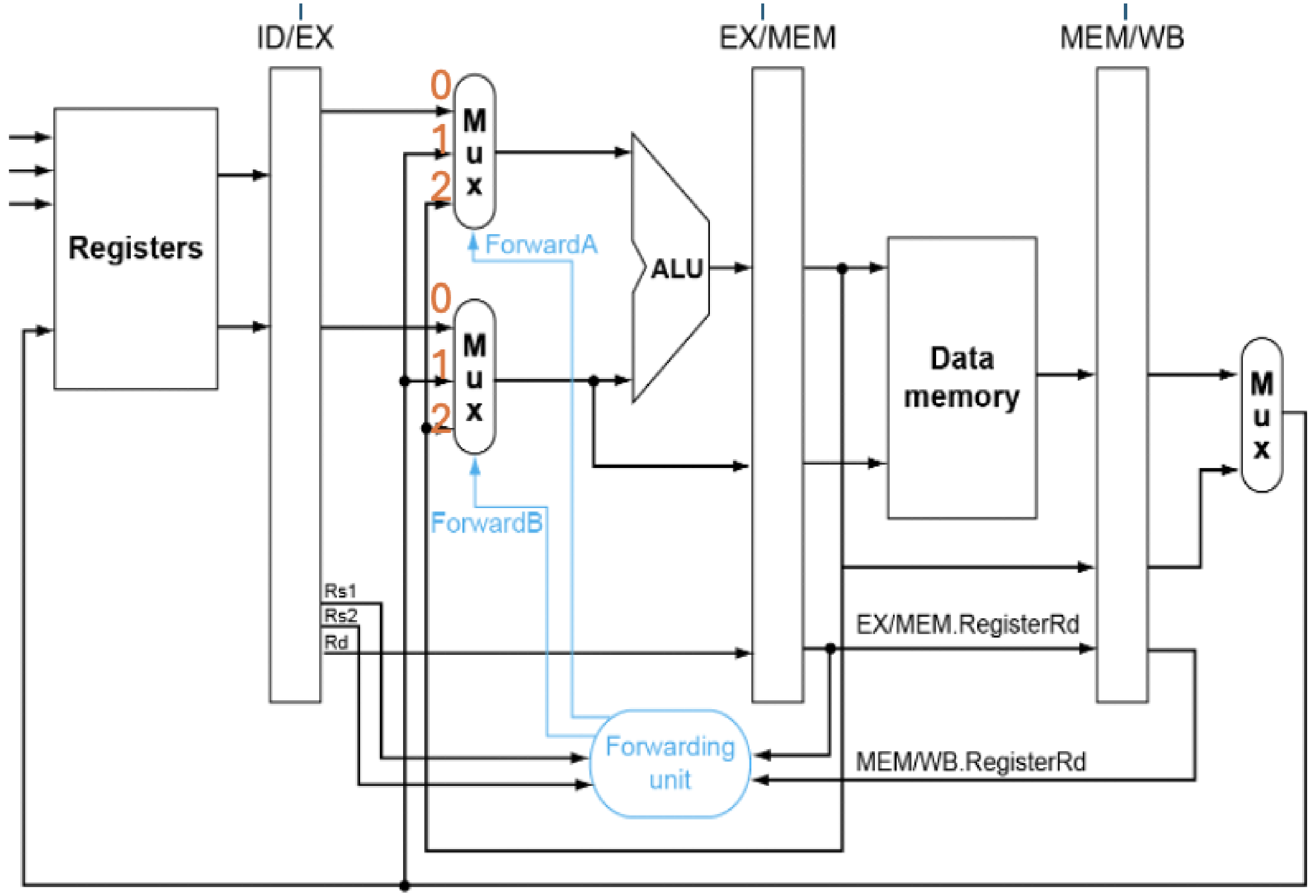
2. And only if Rd for that instruction is not x0

```
addi x0,x1,3
or   x10,x0,x2
```

- EX/MEM.RegisterRd ≠ 0,
  MEM/WB.RegisterRd ≠ 0

- 1 and 2 must be checked before decide the forwarding

Forwarding Paths



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.

Double Data Hazard

- Consider the sequence:

```
add x1, x1, x2
add x1, x1, x3      hazard in EX /MEM
add x2, x1, x4      hazard in EX/ MEM , and hazard in MEM/WB
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn’t true

Revised Forwarding Condition

- MEM hazard

```
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.RegisterRs1))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
```

used data from EX/MEM Register

```
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.RegisterRs2))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01
```

Used data from MEM/WB Register

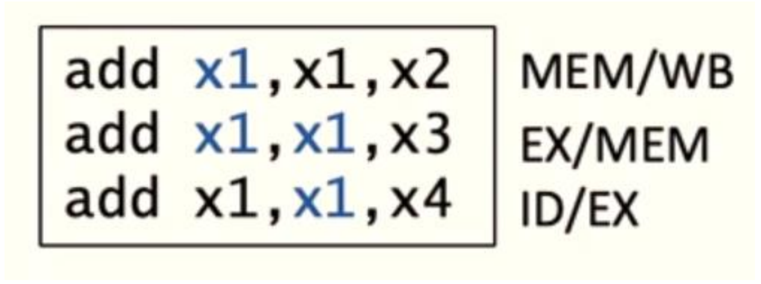
Revised Forwarding Condition

MEM hazard?
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0))

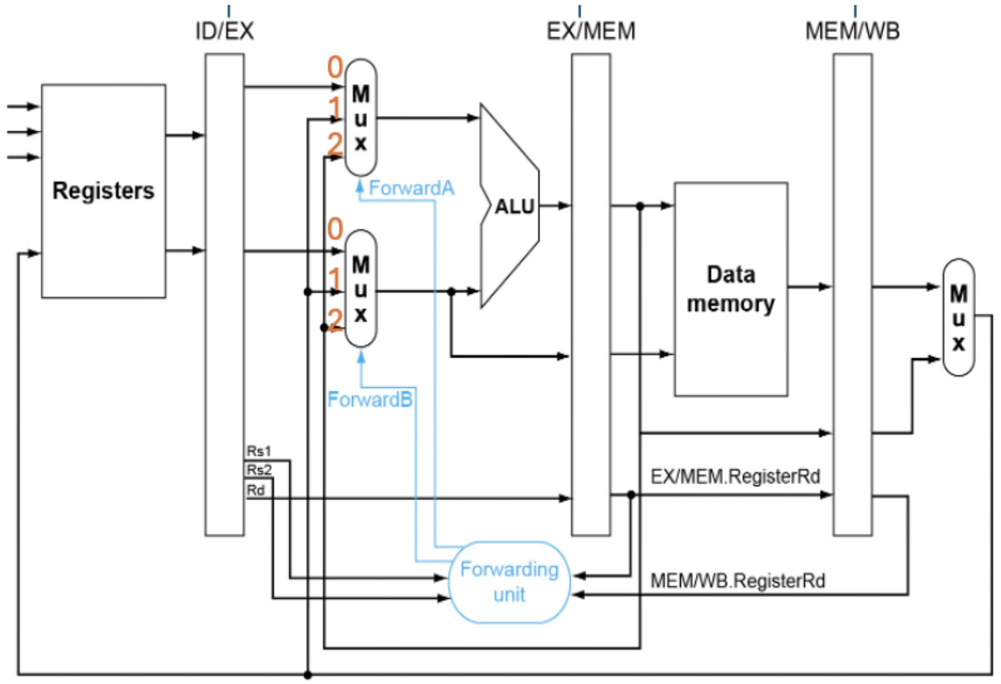
and not((EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0))
and (EX/MEM.RegisterRd = ID/EX.RegisterRs1))

and (MEM/WB.RegisterRD= ID/EX.RgisterRs1))   Forward A = 01

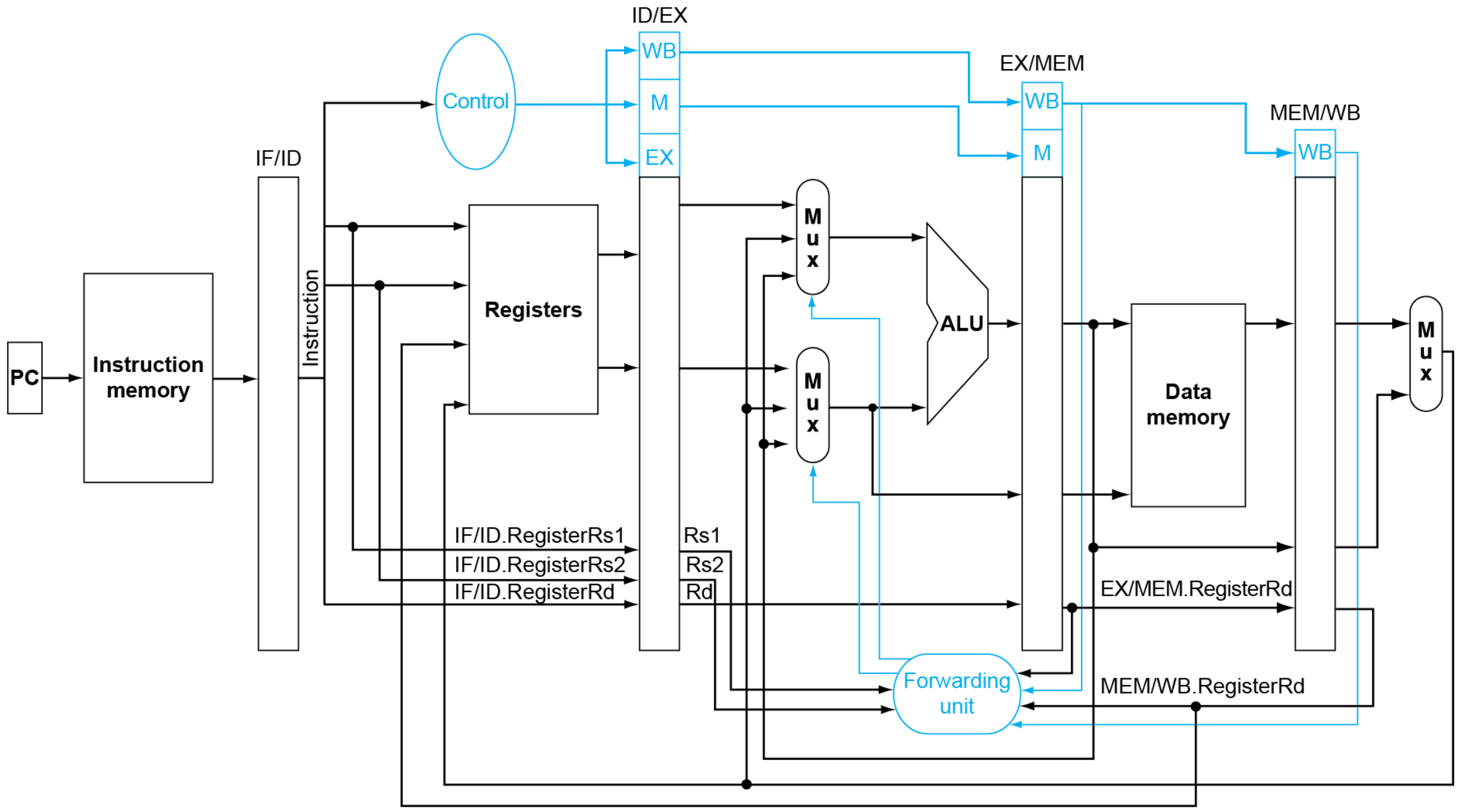
else ForwardA = 10



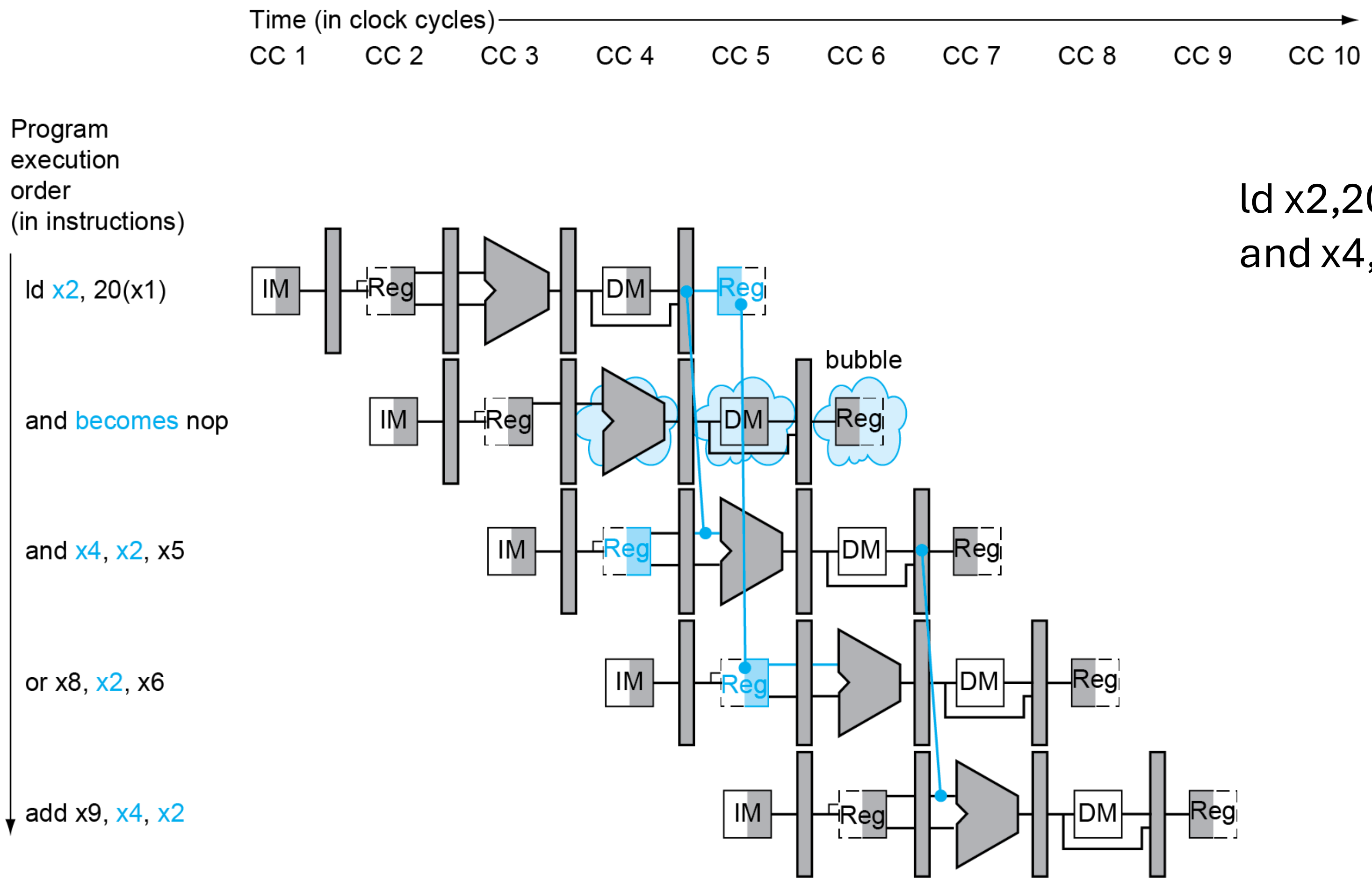
```
add x1,x1,x2    MEM/WB
add x1,x1,x3    EX/MEM
add x1,x1,x4    ID/EX
```



Datapath with Forwarding



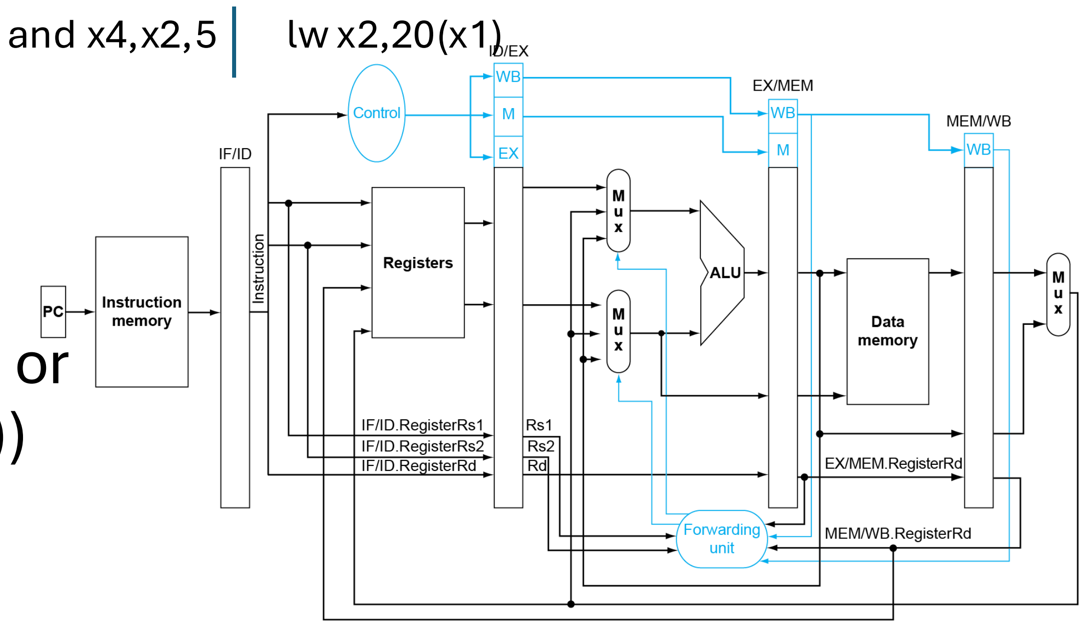
Load-Use Data Hazard



ld x2,20(x1)
and x4,x1,x5

Load-Use Hazard Detection

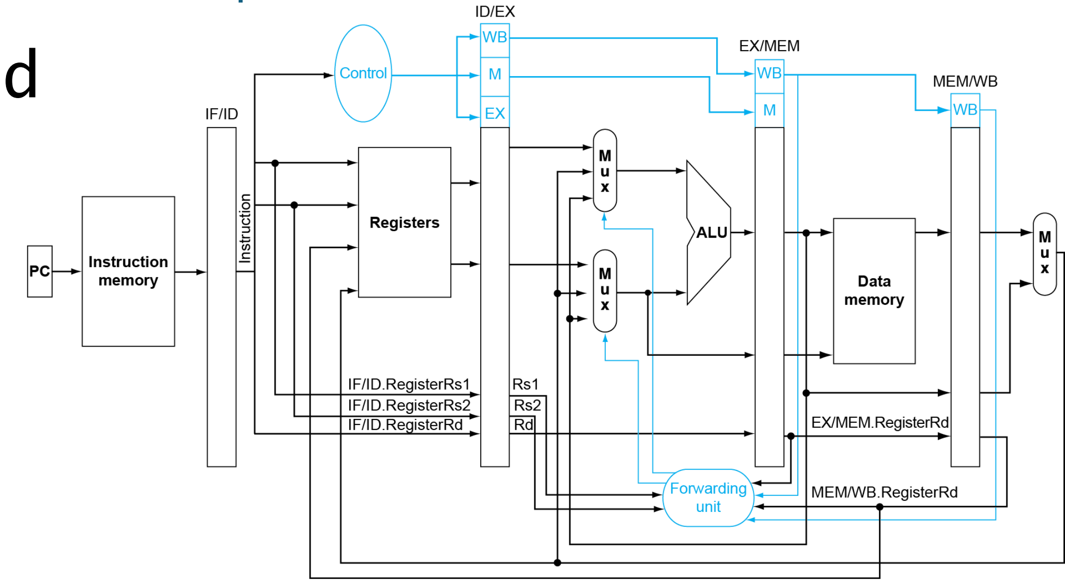
- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2))
- If detected, stall and insert bubble



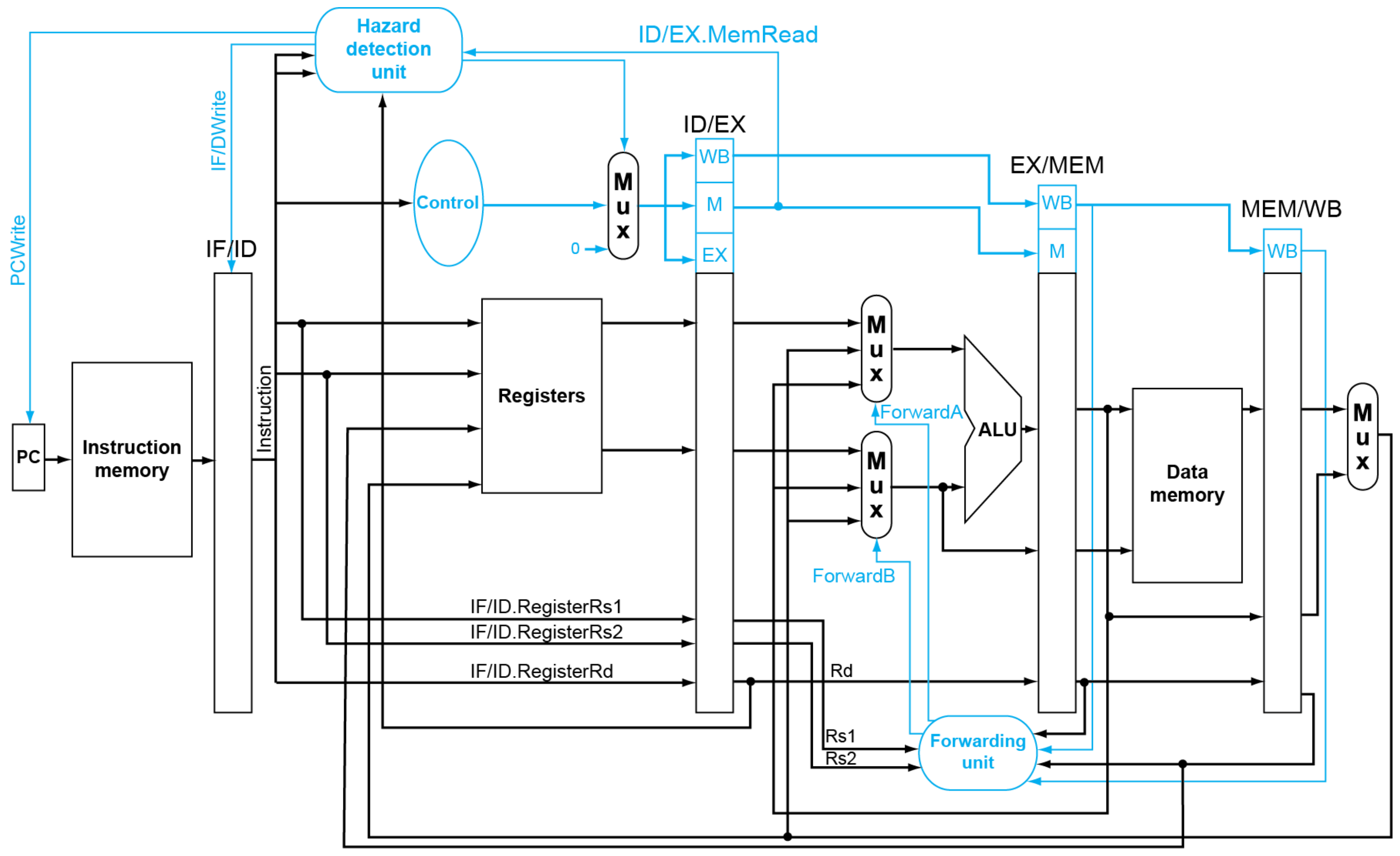
How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do **nop** (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for `ld`
    - Can subsequently forward to EX stage

and x4,x2,5 | lw x2,20(x1)



Datapath with Hazard Detection



Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, syscall, …
- Interrupt
  - From an external I/O controller
- Dealing with them without sacrificing performance is hard