1. Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting `NOP` instructions where necessary). Suppose that (after optimization) a typical `n`- instruction program requires an additional `4*n NOP` instructions to correctly handle data hazards.

1.1 Suppose that the cycle time of this pipeline without forwarding is 250 ps. Suppose also that adding forwarding hardware will reduce the number of `NOPs` from `.4*n to .05*n`, but increase the cycle time to 300 ps. What is the speedup of this new pipeline compared to the one without forwarding?

$T_0 = n \times 250\,ps$

$T_1 = 1.4n \times 250\,ps$

$T_2 = 1.05n \times 300\,ps$

Thus, $Speedup = \frac{T_1}{T_2} = \frac{1.4n \times 250\,ps}{1.05n \times 300\,ps} = 1.11$

1.2 Different programs will require different amounts of `NOPs`. How many `NOPs` (as a percentage of code instructions) can remain in the typical program before that program runs slower on the pipeline with forwarding?

$T_0 = 1.4n \times 250\,ps$

assuming γ = the number of stalls per instruction, then

$T_F = (1+\gamma)n \times 300\,ps$

When $T_0 \geq T_F$,

$1.4n \times 250\,ps \geq (1+\gamma)n \times 300\,ps$

Thus, $\gamma \leq 0.167$

1.3 Repeat 1.2; however, this time let x represent the number of `NOP` instructions relative to `n`. (In 9.2, x was equal to .4.) Your answer will be with respect to x.

$(1+x)n \times 250\,ps \geq (1+\gamma)n \times 300\,ps$

Thus, $\gamma \leq \frac{250x-50}{300} = \frac{5x-1}{6}$

1.4 Can a program with only `.075*n NOPs` possibly run faster on the pipeline with forwarding? Explain why or why not.

When x=0.075,

$(1 + 0.075)n \times 250\ ps = 268.75\ ps, which\ is\ smaller\ than\ 300\ ps$

No.

1.5 At minimum, how many `NOPs` (as a percentage of code instructions) must a program have before it can possibly run faster on the pipeline with forwarding?

$\frac{300}{250} = 1.2$

Thus, at minimum, 0.2×n NOPs a program must have.

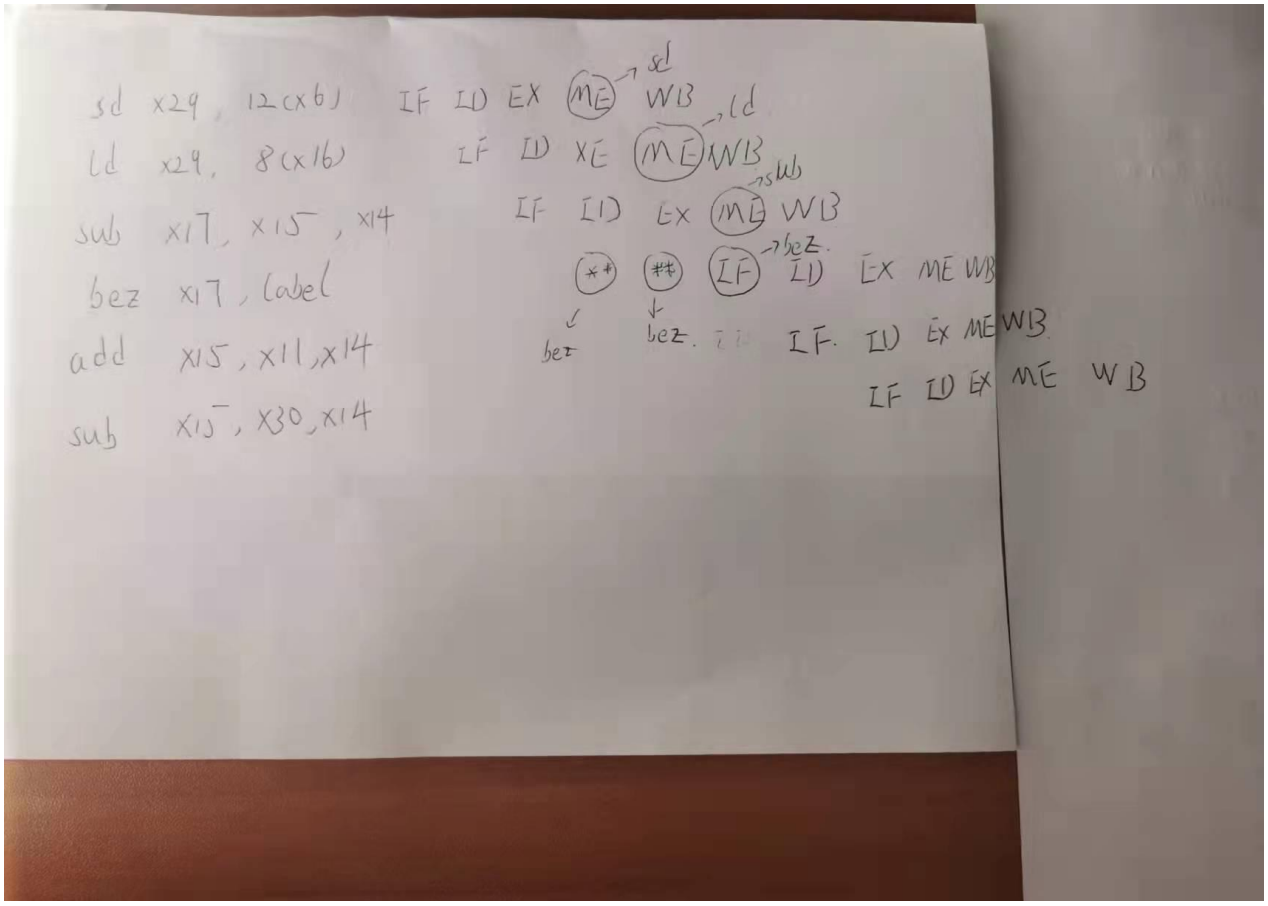
2. Consider the fragment of RISC-V assembly below:

```
sd x29, 12(x16)
ld x29, 8(x16)
sub x17, x15, x14

beqz x17, label
add x15, x11, x14
sub x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (*that handles both instructions and data*). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

2.1 Draw a pipeline diagram to show were the code above will stall.



2.2 In general, is it possible to reduce the number of stalls/`NOPs` resulting from this structural hazard by reordering code?

No. Since every instruction 3 cycles after a load/store instruction have to access memory for IF, it is not possible to lower NOPs by reordering the code.

2.3 Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding `NOPs` to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

Yes. Adding a separate memory array for Data and for Instructions, which is the only way to resolve this structural hazard in hardware.

2.4 Approximately how many stalls would you expect this structural hazard to generate in a typical program? (*Use the instruction mix shown below*)

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

25%+11% = 36%

Thus, 36% of the instructions.

3. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the `ALU`. (See Problem 4 in HW 4) As a result, the `MEM` and `EX` stages can be overlapped and the pipeline has only four stages.

3.1 How will the reduction in pipeline depth affect the cycle time?

The cycle time will not change by reduction.

3.2 How might this change improve the performance of the pipeline?

It might reduce the latency of instructions.

3.3 How might this change degrade the performance of the pipeline?

Since addi might be used with ld/sd, thus, it might increase the number of instructions in the program.

4. Which of the two pipeline diagrams below better describes the operation of the pipeline's hazard detection unit? Why?

Choice 1:

```
ld x11, 0(x12): IF ID EX ME WB

add x13, x11, x14: IF ID EX .. ME WB

or x15, x16, x17:    IF ID .. EX ME WB
```

Choice 2:

```
ld x11, 0(x12): IF ID EX ME WB

add x13, x11, x14: IF ID .. EX ME WB

or x15, x16, x17:     IF .. ID EX ME WB
```

Choice 2.

In Choice 1, it will cause a “load-use-data-hazard”, which is that x11 being used at the start of ME stage in the execute stage of the add instruction will only be available at the end of ME stage.

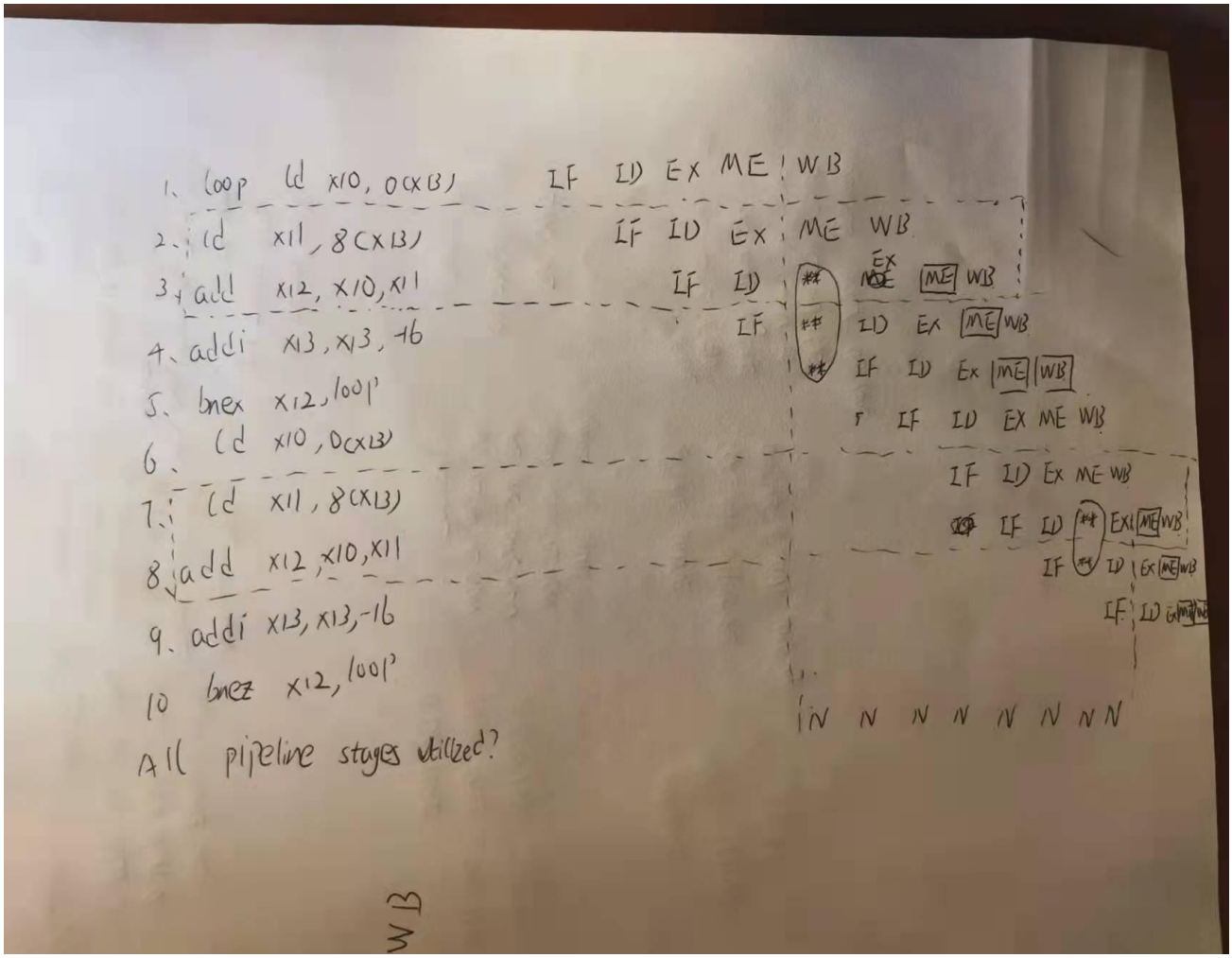
For Choice 2, x11 being available at the end of ME stage is used at the start of WB stage because the EX stage has been delayed in ME stage.

5. Consider the following loop. `LOOP: ldx10, 0(x13)`

```
ldx11, 8(x13)
addx12, x10, x11
subix13, x13, 16
bnezx12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the `EX` (as opposed to the `ID`) stage.

5.1 Show a pipeline execution diagram for the first two iterations of this loop.

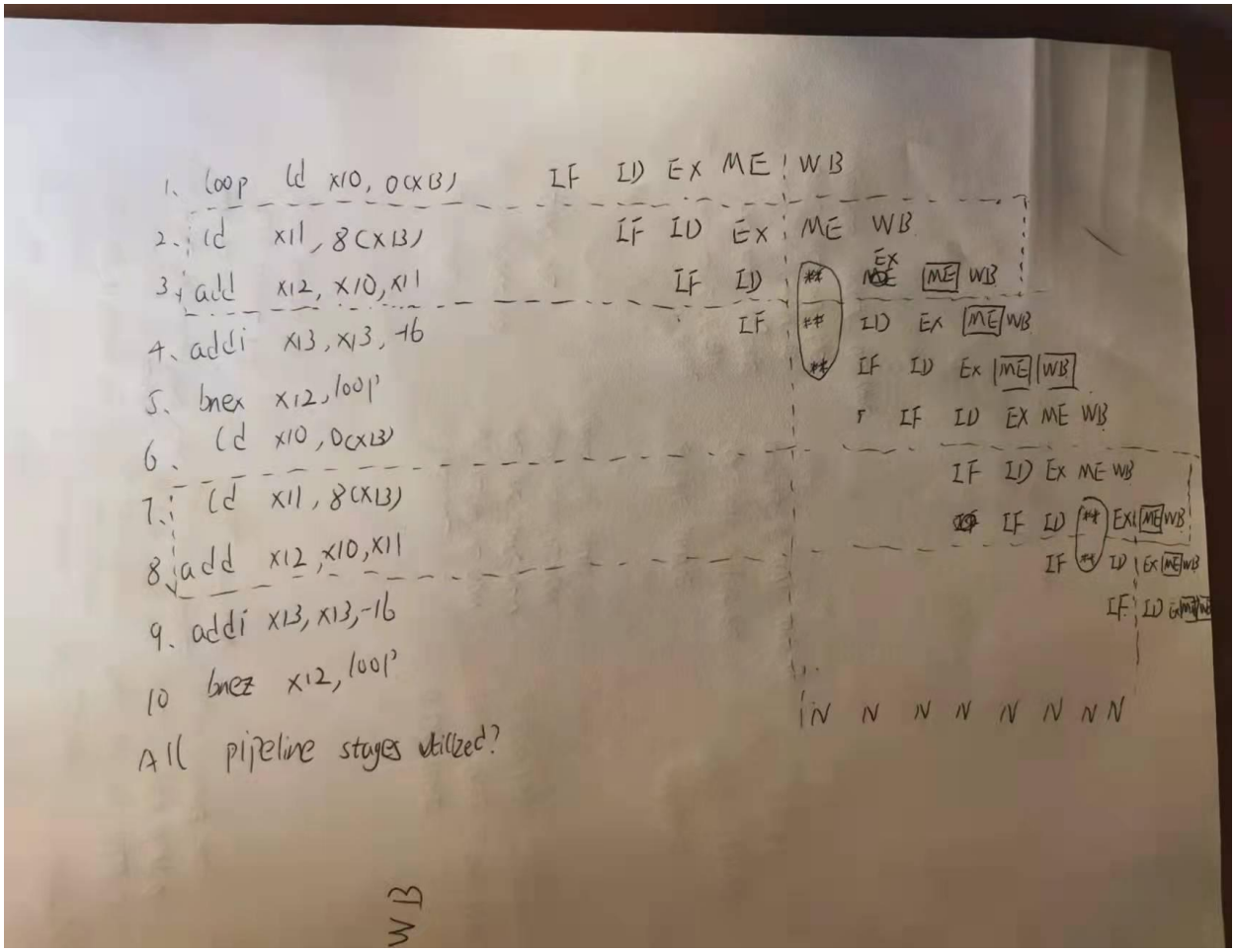


5.2 Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the `IF` stage. End with the cycle during which the `bnez` is in the `IF` stage.)

1. pipeline stages unused by any instruction are in solid box.

2. A clock cycle that does not have all of the pipeline stages utilized is marked with "N".

3. Load-use-hazards are in two dashed boxes shown below.



6. This exercise is intended to help you understand the cost/complexity/performance trade-offs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from *Figure 4.53 in RISC-V text (reproduced below)*. These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions has a particular type of RAW data dependence.

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.

The type of `RAW` data dependence is identified by the stage that produces the result (`EX` or `MEM`) and the next instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We

assume that the register write is done in the first half of the clock cycle and that register reads are done in the second half of the cycle, so "EX to 3rd" and "MEM to 3rd" dependences are not counted because they cannot result in data hazards. We also assume that branches are resolved in the EX stage (as opposed to the ID stage), and that the CPI of the processor is 1 if there are no data hazards.

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

Assume the following latencies for individual pipeline stages. For the EX stage, latencies are given separately for a processor without forwarding and for a processor with different kinds of forwarding.

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
120 ps	100 ps	110 ps	130 ps	120 ps	120 ps	120 ps	100 ps

6.1 For each RAW dependency listed above, give a sequence of at least three assembly statements that exhibits that dependency.

EX to 1st only	MEM to 1st only	EX to 2nd only	MEM to 2nd only	EX to 1st and EX to 2nd
sub x2, x7, x8	ld x2, 0(x7)	add x2, x7, x8	ld x2, 0(x7)	sub x2, x7, x8
add x10, x2, x9	add x8, x2, x9	sub x15, x22, x23	add x5, x6, x7	add x10, x2, x9
add x27, x28, x29	sub x15, x22, x23	add x22, x2, x23	add x8, x2, x9	add x27, x2, x29

6.2 For each RAW dependency above, how many NOPs would need to be inserted to allow your code from ***6.1*** to run correctly on a pipeline with no forwarding or hazard detection? Show where the NOPs could be inserted.

EX to 1st only	MEM to 1st only	EX to 2nd only	MEM to 2nd only	EX to 1st and EX to 2nd

sub x2, x7, x8
NOP
NOP
add x10, x2, x9
add x27, x28, x29
	ld x2, 0(x7)
	NOP
	NOP
	add x8, x2, x9
		add x2, x7, x8
		sub x15, x22, x23
		NOP
		add x22, x2, x23
			ld x2, 0(x7)
			add x5, x6, x7
			NOP
			add x8, x2, x9
				sub x2, x7, x8
				NOP
				NOP
				add x10, x2, x9
				add x27, x2, x29

6.3 Analyzing each instruction independently will over-count the number of NOPs needed to run a program on a pipeline with no forwarding or hazard detection. Write a sequence of three assembly instructions so that, when you consider each instruction in the sequence independently, the sum of the stalls is larger than the number of stalls the sequence actually needs to avoid data hazards.

EX to 1st and 2nd:

sub x2, x7, x8

add x10, x2, x9

add x27, x2, x29

Since EX to 1st needs 2 NOPs and EX to 2nd needs 1 NOP, then we will get that

sum = 2 + 1 = 3

In real, number of NOPs = 2

6.4 Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.)

$2 \times 5\% + 2 \times 20\% + 1 \times 5\% + 1 \times 10\% + 2 \times 10\% = 0.85$

CPI=1+ 0.85 =1.85

Since 1.85 CPI, then

0.85/1.85 = 46%

Thus, 46% of cycles are stalls.

6.5 What is the CPI if we use full forwarding (forward all results that can be forwarded)? What percent of cycles are stalls?

CPI = 1 + 0.2 = 1.2

Since 1.2 CPI, then

0.2/1.2 = 17%

Thus, 17% of cycles are stalls.

6.6 Let us assume that we cannot afford to have three-input multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the EX/MEM pipeline register (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

When MEM/WB register unavailable, then
2*20%+1*5%+1*10%+1*10% = 0.65
Thus, 1.65 CPI

When EX/MEM register unavailable, then
1*5%+1*20%+1*10% = 0.35
Thus, 1.35 CPI

6.7 For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by each type of forwarding (EX/MEM, MEM/WB, for full) as compared to a pipeline that has no forwarding?

	No forwarding	Forwarding EX/MEM	Forwarding MEM/WB	Full Forwarding
CPI	1.85	1.65	1.35	1.2
Cycle time	120ps	120ps	120ps	130ps
New time	222ps	198ps	162ps	156ps
Speedup		1.12	1.37	1.42

For EX/MEM:

Speedup = (120*1.85)/(120*1.65) = 1.12

For MEM/WB:

Speedup = (120*1.85)/(120*1.35) = 1.37

For full Forwarding:

Speedup = (120*1.85)/(120*1.2) = 1.42

6.8 What would be the additional speedup (relative to the fastest processor from 6.7) be if we added "timetravel" forwarding that eliminates all data hazards? Assume that the yet-to-be- invented time-travel circuitry adds 100 ps to the latency of the full-forwarding EX stage.

Speedup = $\frac{130\times1.2}{230\times1} = 0.68$

Thus, the speedup is 0.68.

7. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
ld x13, 4(x15)
ld x12, 0(x2)
or x13, x15, x13
sd x13, 0(x15)
```

7.1 If there is no forwarding or hazard detection, insert `NOPs` to ensure correct execution.

add x15, x12, x11

NOP

NOP

ld x13, 4(x15)

ld x12, 0(x2)

NOP

or x13, x15, x13

NOP

NOP

sd x13, 0(x15)

7.2 Now, change and/or rearrange the code to minimize the number of `NOPs` needed. You can

assume register `x17` can be used to hold temporary values in your modified code.

The number of NOPs could not be minimized.

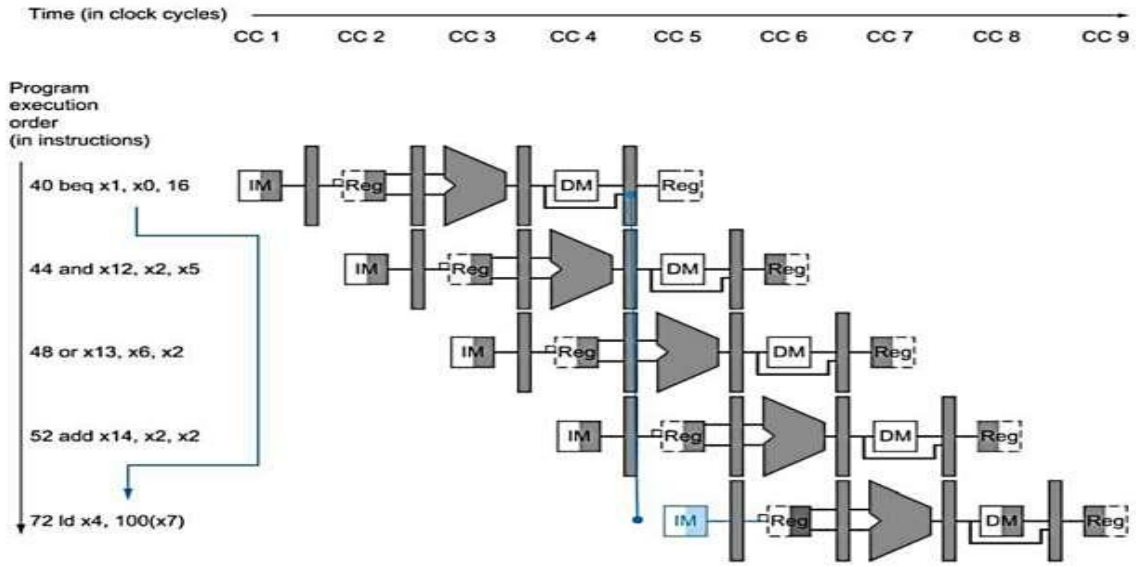
7.3 If the processor has forwarding, but we forgot to implement the hazard detection unit, what

happens when the original code executes?

Nothing will happen. The code will work correctly.

Since there is no ld in given instructions.

7.4 If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59 of the RISC V text (reproduced below).



Answer:

Stage	ForwardA	ForwardB

1	X	X
2	X	X
3	0	0
4	2	0
5	0	0
6	0	1
7	0	2

7.5 If there is no forwarding, what new input and output signals do we need for the hazard detection unit in the Figure above? Using this instruction sequence as an example, explain why each signal is needed.

To check the destination register of these two instructions, we only need to add the value from the MEM/WB register since the Hazard unit already has the value of rd from the EX/MEM register. And there are no additional outputs are needed.

To detect the data hazard between the add and the following ld, we need the value of rd from EX/MEM. For the value of rd form MEM/WB, it is needed as well, which is to detect the data hazard between the first ld instruction and or instruction.

7.6 For the new hazard detection unit from Problem 6.5 of this HW assignment, specify which output signals it asserts in each of the first five cycles during the execution of this code.

Answer:

Stage	PCWrite	IF/IDWrite	Control mux
1	1	1	0
2	1	1	0
3	1	1	0
4	0	0	1
5	0	0	1