1. Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data (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 **0.**`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?

Old = 250 * (n + .4n) = 350n; New = 300 * (n + 0.05n) = 315n; Speedup = 350 / 315 = 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?

Old = 250 * (n + 0.4n) = 350n; New = 300*(n+kn) = 300(1+k)n

350 = 300(1+k). ——> k = 0.16

1.3 Repeat 1.2; however, this time let x represent the number of `NOP` instructions relative to `n`. (In 1.2, x was equal to 0.4) Your answer will be with respect to x.

Old = 250 * (n + xn) = 250(1+k) n; New = 300*(n+kn) = 300(1+k)n

250(1+x) = 300(1+k). ————>. k = 5/6x - 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.No, When x = 0.075, old time = 250 * 1.075 = 268.75 < 300

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?

5/6x-1/6 = 0 —-> 0.2n

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.

sd x29, 12(x16) IF ID EX **MEM** WB

ld x29, 8(x16) IF ID EX **MEM** WB

sub x17, x15, x14 IF ID EX WB

beqz x17, label **IF** ID EX

add x15, x11, x14 **IF** ID EX WB

sub x15, x30, x14 IF ID EX WB

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

Yes.

sub x17, x15, x14 IF ID EX WB
add x15, x11, x14 **IF** ID EX WB
sub x15, x30, x14 IF ID EX WB
sd x29, 12(x16) IF ID EX **MEM** WB
ld x29, 8(x16) IF ID EX **MEM** WB
beqz x17, label **IF** ID EX

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.

sd x29, 12(x16) IF ID EX **MEM** WB
ld x29, 8(x16) IF ID EX **MEM** WB
sub x17, x15, x14 IF ID EX WB
NOP
NOP
beqz x17, label **IF** ID EX
add x15, x11, x14 **IF** ID EX WB
sub x15, x30, x14 IF ID EX WB

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

25%+11% = 36%

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 be shorter because of less stages

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

If MEM stage takes much time, this change may let the pipeline faster

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

Because there are additional add instructions, the total time may decrease if MEM stage is not time consuming.

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 is better.
Before executing the 2nd instruction, the detecting units detects if one of the register of the 2nd instruction comes from previous EX rd register.

5. Consider the following loop.

```
LOOP: ld
x10, 0(x13)
ld
x11, 8(x13)
add x12, x10, x11
subi x13, x13, 16
bnez x12, 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.

Loop: ld x10, 0(x13) IF ID EX MEM WB
ld x11, 8(x13); IF ID EX MEM WB
Bubble
add x12, x10, x11 IF ID EX MEM WB
subi x13, x13, 16 IF ID EX MEM WB
bnez x12, LOOP IF ID EX MEM WB
LOOP: ld x10, 0(x13); IF ID EX MEM WB
ld x11, 8(x13); IF ID EX MEM WB
Bubble
add x12, x10, x11 IF ID EX MEM WB
subi x13, x13, 16 IF ID EX MEM WB
bnez x12, LOOP IF ID EX MEM
Please see below in response to 5.2

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.)

Loop: ld x10, 0(x13) IF ID EX MEM WB
ld x11, 8(x13); IF ID EX MEM WB
IF ID EX MEM WB
add x12, x10, x11 IF ID EX MEM WB
subi x13, x13, 16 IF ID EX MEM WB

```
bnez x12, LOOP                    IF ID EX MEM WB
LOOP: ld x10, 0(x13);               IF ID EX MEM WB
ld x11, 8(x13);                       IF ID EX MEM WB
                                        IF ID EX MEM WB
add x12, x10, x11                       IF ID EX MEM WB
subi x13, x13, 16                         IF ID EX MEM WB
bnez x12, LOOP                              IF ID EX MEM
```

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.

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.

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.

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

that exhibits that dependency.

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

add x10, 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:
sub x2, x7, x8
NOP
NOP
add x10, x2, x9
add x27, x28, x29

MEM to 1st:
ld x2, 0(x7)
NOP
NOP
add x8, x2, x9

EX to 2nd:
add x2, x7, x8
sub x15, x22, x23
NOP
add x22, x2, x23

MEM to 2nd:
ld x2, 0(x7)
add x5, x6, x7
NOP
add x8, x2, x9

EX to 1st and EX to 2nd :
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:
sub x2, x7, x8
NOP
NOP
add x10, x2, x9

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

6.4 Assuming no other hazards, what isthe 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.)

NOPs = 0.05*2 + 0.2*2 + 0.05*1 + 0.1*1 + 0.1*2 = 0.85
CPI = 1.85
0.85/0.185 = 46% NOPs

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%

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

MEM accounts for 20% of instructions that will generate 1 NOP. CPI will increase from 1 to 1.2.
Thus 0.2/1.2 = 0.17, which is 17% NOPs

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?

MEM/WB register unavailable, NOPs average is (0.05 * 0) + (0.2 * 2) + (0.05 * 1) + (0.10 * 1)+ (0.10 * 1)

= 0.65 stalls/instruction. Thus, the CPI is 1.65
EX/MEM register unavailable, NOPs average is (0.05 * 1) + (0.2 * 1) + (0.1 * 1)
= 0.35 stalls/instruction. Thus, the CPI is 1.35

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?

For EX/MEM, speedup = 1.12
For MEM/WEB, speedup = 1.37
For full, speedup = 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.

Time travel CPI will be 1, while CPI is 1.2 for full forwarding. Clock period with zero hazard forwarding is 230, while clock period for full forwarding is 130.
Thus,
Speedup = (1.2 * 130) / (1 * 230) = 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 //EX to 1st RAW Hazard resolution with 2 NOPs
ld x13, 4(x15)
ld x12, 0(x2)
nop //MEM to 2nd RAW Hazard resolution with 1 NOP
or x13, x15, x13
nop
nop //EX to 1st RAW Hazard resolution with 1 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.

not possible to reduce the number of NOPs.

7.3 If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

we required a hazard detection unit even with forwarding because it's accountable for

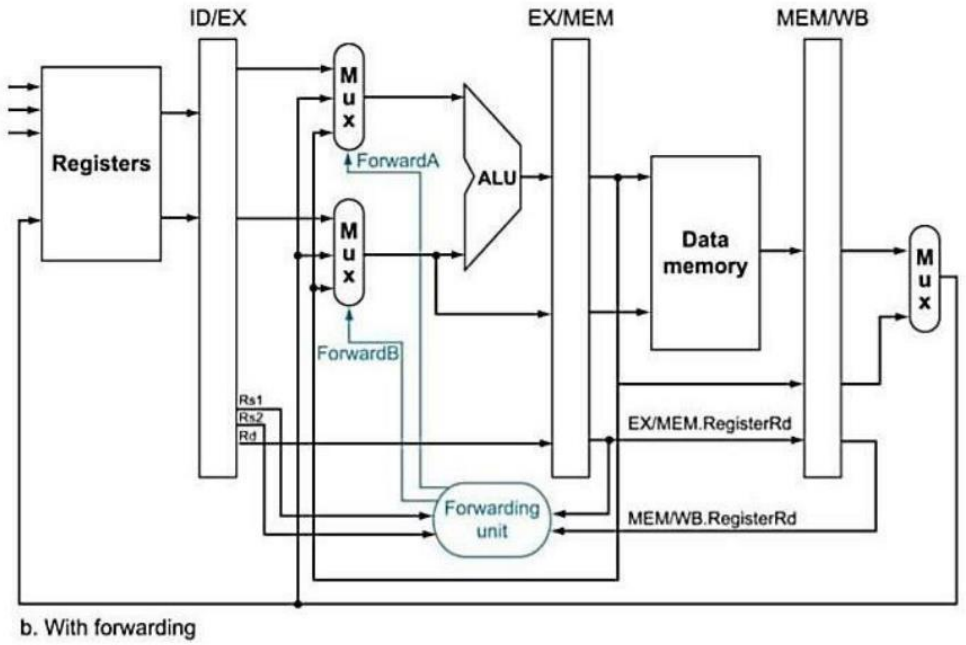
inserting a one-cycle stall whenever the load gives a value to the instruction that immediately pursue by load without hazard detection unit in place, instruction which depends on instruction previous load gets stable data or value the register had by the load instruction.

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 4.5**3** of the RISC V text (reproduced below).

add x15, x12, x11 // no signals are asserted before this
except pcwrite=1, ALUin1=x, ALUin2=x
ld x13, 4(x15)// Forwarding unit inturn asserts. Mux which can
select the inputs from the register of inst. Data coming from
ALU. PCwrite=1, ALUin1=x,ALUin2=xld x12, 0(x2) //no signals are asserted before this except
pcwrite=1, ALUin1=0,ALUin2=0
or x13, x15, x13// Forwarding unit inturn asserts. Mux which can
select the data from the MEM write back of inst. PCwrite=1,
ALUin1=1,ALUin2=0
sd x13, 0(x15) // Forwarding unit inturn asserts. Mux which can select the inputs from the register of inst. Data coming from
ALU. PCwrite=1, ALUin1=0,ALUin2=0

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.

Clock Cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
ld		IF	ID	EX	MEM	WB			
ld			IF	ID	EX	MEM	WB		
or				IF	ID	EX	MEM	WB	
sd					IF	ID	EX	MEM	WB



b. With forwarding

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.

EX/MEM rd is required for the detection of hazards between add and the consequent load.

MEM/WB rd is required for the detection of load-use-data hazard between the first load and the '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.

1. PCWrite = 1; IF / IDWrite = 1; control-mux = 0
2. PCWrite = 1; IF / IDWrite = 1; control-mux = 0

3. PCWrite = 1; IF / IDWrite = 1; control-mux = 0
4. PCWrite = 0; IF / IDWrite = 0; control-mux = 1
5. PCWrite = 0; IF / IDWrite = 0; control-mux = 1

Clock Cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
ld		IF	ID	-	-	EX	MEM	WB	
ld			IF	-	-	ID	EX	MEM	WB