Instructor: Azeez Bhavnagarwala, email: ajb20@nyu.edu

<u>*Course Assistants*</u>

Varadraj Kakodkar (vns2008), Kartikay Kaushik (kk4332), Siddhanth Iyer (si2152), Swarnashri Chandrashekar (sc8781), Karan Sheth (kk4332), Haotian Zheng (hz2687), Haoren Zhang (kk4332), Varun Kumar (vs2411)

Homework Assignment 6 [released Friday November 4th 2022] [due Friday November 11th by 11:59PM]

You *are allowed* to discuss HW assignments with anyone. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Course Assistants or the Instructor during office hours or by appointment if you need any help with the HW.
Please enter your responses in this Word document after you download it from NYU Classes.
Please use the Brightspace portal to upload your completed HW.

---

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 **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?

The original running time is:

1.4n * 250ps

The running time with forwarding is:

1.05n * 300ps

The speed up is:

1.4n * 250ps / 1.05n * 300ps = 1.111

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?

According to the problem, if the program with forwarding want to run faster than the program without forwarding:

$(1 + k)n * 300ps \leqslant 1.4n * 250ps$

k ≤ 0.1667

Thus, maximum 16.67% of NOPs can be remained

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.

According to 1.2, we can have

$(1 + k)n * 300ps \leq (1 + x)n * 250ps$

$k \leq (5x - 1) / 6$

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

No.

Now the running time without forwarding is:

1.075n * 250ps = 268.75n ps

The best case is after applying the forwarding method, there is no NOPs. Thus the running time is 300n ps, which is still slower than the running time without forwarding.

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?

According to 1.3, if we solve the k < 0, then the speedup cannot be realized. Thus, we can also know that:

$(5x - 1) / 6 >= 0$

$x >= 0.2$

Thus, 20% of NOPs must a program have before it can possibly run faster on the pipeline with forwarding.

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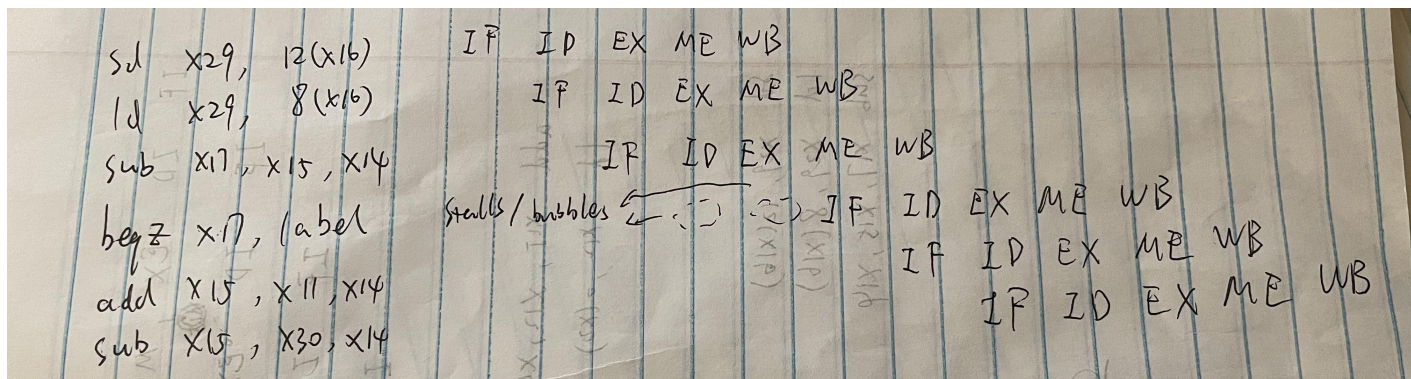
2.2 In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

No.

Because every instruction has the "Instruction fetch" stage, which will make every data access (MEM stage) cause a stall.

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.

No. Because every NOP has to be fetched from memory too.

Separating the data and instruction memory will help.

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%
Because every "ld" and "sd" will cause a stall. Totally 36% of instructions will have structural hazard.

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 be affeted by the reduction in pipeline depth. Cycle time is determined by the slowest stage.

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

It can potentially reduce the number of stalls in the program.

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

The total number of instructions could be larger. Because the "ld" and "sd" instruction will be replaced by the "ld/addi" and "sd/addi" instruction pair.

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
```

Solution:

Choice 2 is more proper to describe the operation of the pipeline's hazard detection.

That is because Choice 1 will have the "load-use-data-hazard". When the second instruction want to read the number at "x11", the "ME" stage of last instruction is not completed. The data will be loaded to "x11" at the end of "ME" stage.

For Choice 2, the "add" instruction will wait after the data has been loaded to "x11".

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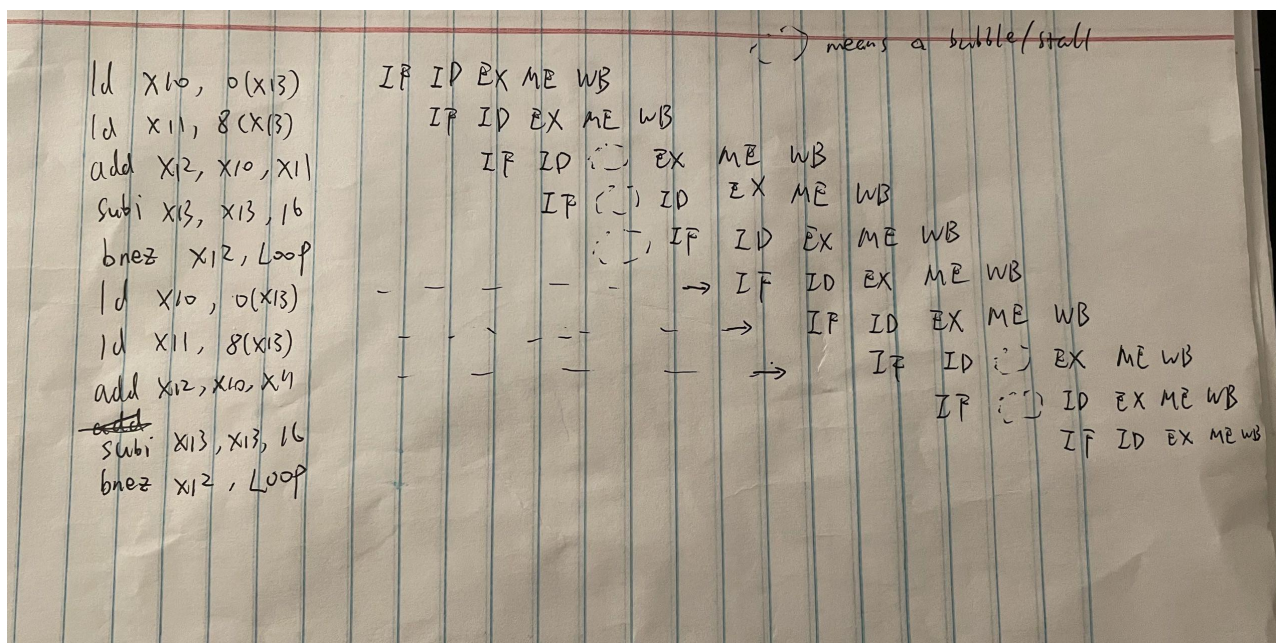
5.2 Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the subi is in the IF stage. End with the cycle during which the bnez is in the IF stage.)

For this program, there does not exist any cycles in which all five pipeline stages are doing useful work.

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:

add x5, x6, x7

add x8, x5, x9

add x28, x29, x30

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.

MEM to 1st Only:

ld x5, 0(x6)

nop

nop

add x7, x5, x8

add x28, x29, x30

EX to 2nd Only:

add x5, x6, x7

add x28, x29, x30

nop

add x8, x5, x6

MEM to 2nd Only:

ld x5, 0(x6)

add x28, x29, x30

nop

add x7, x5, x8

EX to 1st and EX to 2nd :

add x5, x6, x7

nop

nop

add x8, x5, x9

add x28, x5, x6

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.

a

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.)

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%

According to 6.2, we can have

5% * 2 + 20% * 2 + 5% * 1 + 10% * 1 + 10% * 2 = 0.85

Thus, there is on average 0.85 stalls per instruction and the CPI is 1.85.

0.85 / 1.85 = 0.4595

Thus, 45.95% 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?

If we use full forwarding, the dependencies from MEM stage to next instruction (MEM to 1st only) cannot be eliminated.

Thus 20% of instructions will generate 1 stall per instruction. The CPI is 1.2.

0.2 / 1.2 = 0.1667

Thus, 16.67% of instructions 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?

If we only forward EX/MEM register, the NOPs we need are:

EX to 1st Only: 0

MEM to 1st Only: 2

EX to 2nd Only: 1

MEM to 2nd Only: 1

EX to 1st and EX to 2nd : 1

On average the stalls per instruction is:

5% * 0 + 20% * 2 + 5% * 1 + 10% * 1 + 10% * 1 = 0.65

The CPI is 1.65

If we only forward MEM/WB register, the NOPs we need are:

EX to 1st Only: 1

MEM to 1st Only: 1

EX to 2nd Only: 0

MEM to 2nd Only: 0

EX to 1st and EX to 2nd : 1

On average the stalls per instruction is:

5% * 1 + 20% * 1 + 5% * 0 + 10% * 0 + 10% * 1 = 0.35

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?

Assume the number of total instuctions are n:

	Without Forwarding	Forwarding EX/MEM	Forwarding MEM/WB	Full Forwarding
CPI	1.85	1.65	1.35	1.2
Period	120	120	120	130
Total Time	222n	198n	162n	156n
Speedup	1	1.12	1.37	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.

If the "timetravel" can eliminate all data hazard, the CPI will be 1.

The clock period then becomes 230ps.

So the speed up is $(1.2*130) / (230*1) = 0.6782 = 67.82\%$

Thus, the "timetravel" is actually slower than the full forwarding.

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

For this piece of code. we cannot reduce the number of NOPs by chaging or rearranging the code.

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

For this program, it will still give the right result.

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).

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
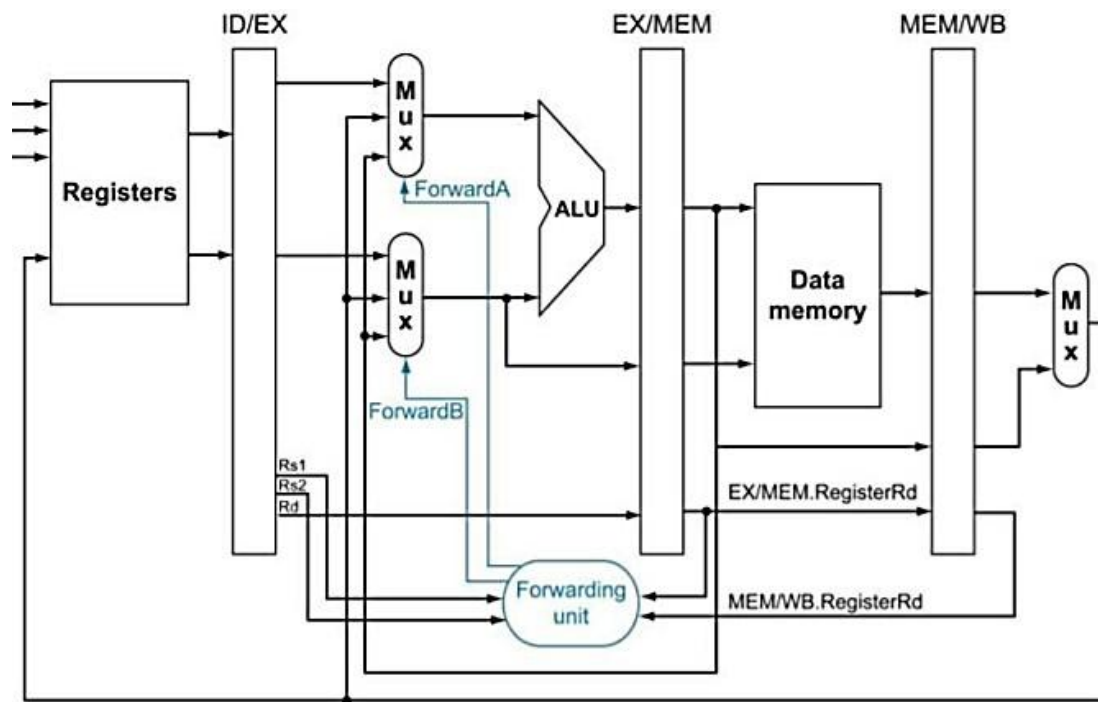
b. With forwarding

For this piece of code, the PCWrite and IF/IDWrite signal are always 1 because there is no stalls in this code.

(1) ForwardA = X; ForwardB = X

(2) ForwardA = X; ForwardB = X

(3) ForwardA = 00; ForwardB = 00

(4) ForwardA = 10; ForwardB = 00

(5) ForwardA = 01; ForwardB = 01

(6) ForwardA = 00; ForwardB = 10

(7) ForwardA = 00; ForwardB = 10

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.

We need to add the MEM/WB register rd value to the hazard detection unit, which already has the value from the EX/MEM register.

To detect the data hazard, we need the EX/MEM register rd value and the MEM/WB register rd value.

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.

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

(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