Homework 4 CSE120

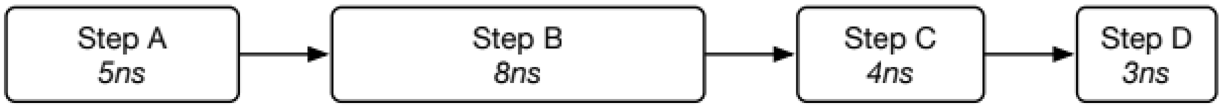
Due by May 8, 2022, 11:59 pm

Please follow the Homework guidelines posted on Canvas. All HW grading queries are to be addressed to the HW4 Regrade Request Form (you must sign in with your UCSC email id), ***Regrade requests are valid only till 1 week after the hw4 grade has been published***

You are NOT allowed to cheat for HW! You may *certainly* discuss amongst yourselves as to how you may proceed with a given problem but verbatim identical answers will not be accepted! Our readers/graders have been instructed to alert us if there is any uncanny similarity between the work turned in by different students.
If an instance of cheating is found, you will be cited for academic misconduct to the extent of receiving a failing grade in this course as a penalty

Q1(10 pts)

You are tasked with improving the performance of a functional unit. The computation for the functional unit has 4 steps (A-D), and each step is indivisible. Assume there is no dependency between successive computations.



A) What is the greatest possible clock rate speedup possible with pipelining? You do not need to worry about the register timing constraints (e.g. delay, setup, hold). **Explain your reasoning**.

Clock period is 5 + 8 + 4 + 3 = **20ns** without pipelining because each instruction has to undergo all 4 steps before the next instruction can begin.

With pipelining, the clock period can be as low as the longest step, which is **8s**. Therefore the greatest possible clock rate speedup is 20ns / 8ns = **2.5x**.

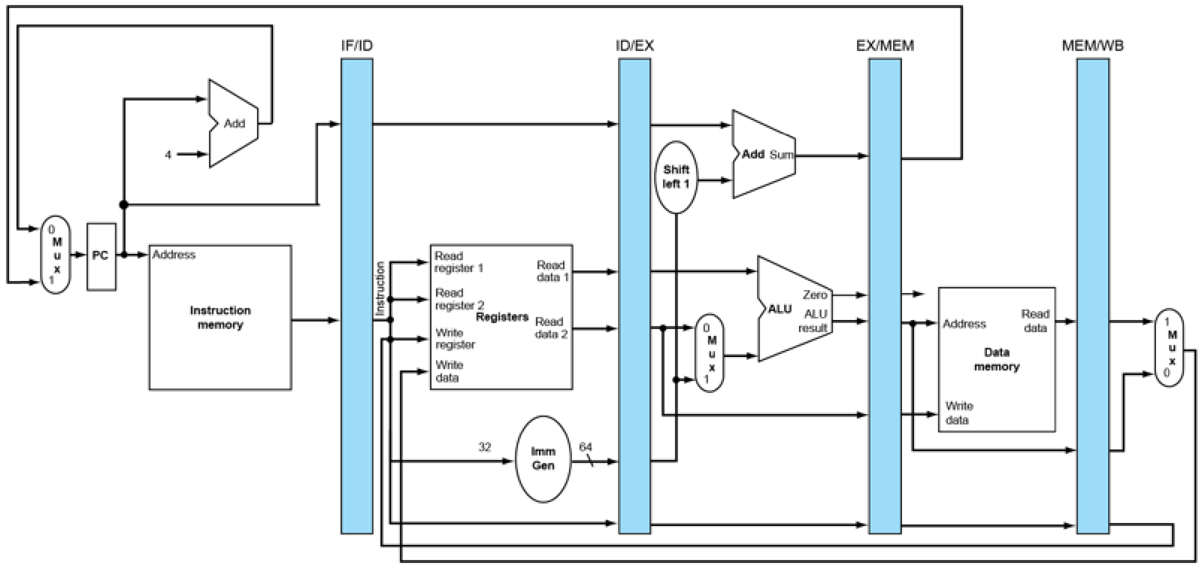
B) For maximizing the clock rate, what is the minimum number of pipeline registers you would use? Where would you insert the registers (draw or describe) into the datapath provided for this functional unit? Why not use fewer or more pipeline stages? **Explain your reasoning**

The number of pipeline registers can be reduced to at least 3 by placing a single register in between each of the steps (by placing a register where each of the arrows are in the diagram). However, the clock period is 8 ns, and steps C and D combined only take 7ns, so they can be done in a single clock cycle. Therefore, we can remove the 3rd register between steps C and D and be left with **a minimum of 2 pipeline registers**, between steps A & B, and between steps B & C. Neither of the two remaining pipeline registers can be removed without increasing the clock period beyond 8ns.

Q2 (12 pts)

For the following questions, it will be helpful for you to draw the Pipeline progression such as **F D X M W** (which stand for the different pipeline stages) for each instruction and trace in which clock cycle the computation for an instruction is actually taking place. That way, you will know the correct value of the operands in the computation.

A) **(4 pts)** Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline that **does not** handle data hazards (i.e., the compiler is responsible for addressing data hazards by inserting NOP instructions where necessary).



Also, the register file does not have the feature of reading and writing in the same clock cycle(i.e. no fast Register File). That is, in the conflict of reading and writing to the same data location, the data read would be the old value. E.g., if the old value of x1=10 and we are both writing the new value 20 to x1 and also reading x1 in the same clock cycle, the output value from read would be 10, NOT 20.

In this context, what would the final values of registers x13 and x14 be? (assuming compiler did not add any NOPs)

You must show some amount of reasoning as to why you arrived at a specific answer (e.g. x13=-18,x14=5) instead of just writing out the register values.

Your explanation does not have to be an essay or even a full sentence but it must show the grader your rationale for choosing your answers

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
```

B) **(4 pts)** Assume that x11 is initialized to 11 and x12 is initialized to22. Suppose you executed the code below on a version of the pipeline that does not handle data hazards (i.e., the compiler is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register x15 be? **Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle.**

You must show some amount of reasoning as to why you arrived at a specific answer (eg. x15=5) instead of just writing out the register values.

Your explanation does not have to be an essay or even a full sentence but it must show the grader your rationale for choosing your answers

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
add x15, x11, x11
```

C) **(4 pts)** Add the **minimum number** of NOP instructions to the code below so that it will run correctly on a pipeline that **does not** handle data hazards. You may assume that the register file operates as described in 2.B above (i.e. as a fast register file).
*Make sure it is the minimum number of NOPs (**might be** 0 in between some instructions)!* For example, without even thinking twice and not even looking at the code content below, I can state with confidence that adding 10 NOPs in between each instruction will solve all data hazards! The problem is then you are heavily penalizing your energy expenditure and throughput in catering to these excess NOPs in your code!
You must show some amount of reasoning as to why you arrived at a specific answer (eg. 4 NOPS between instructions I1 and I2) instead of just showing off the number of NOPs.
Your explanation does not have to be an essay or even a full sentence but it must show the grader your rationale for choosing your answers

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
add x15, x13, x12
```

Q3 (10 pts)

We will be working with the code snippet below for this problem as it passes through a 5 stage (F D X M W) processor. It resolves branches in the decode stage.

```
I1 ld x3, 8(x1)
I2 add x4, x3, x2
I3 add x5, x5, x4
I4 ld x1, 0(x1)
I5 bne x5, x1, exit
```

A) **(5 pts)** For the code snippet, list all possible types of data hazards (RAW, WAR, WAW). E.g. "WAR on x2 between I1-I3".

WAR on x1 between I1–I4

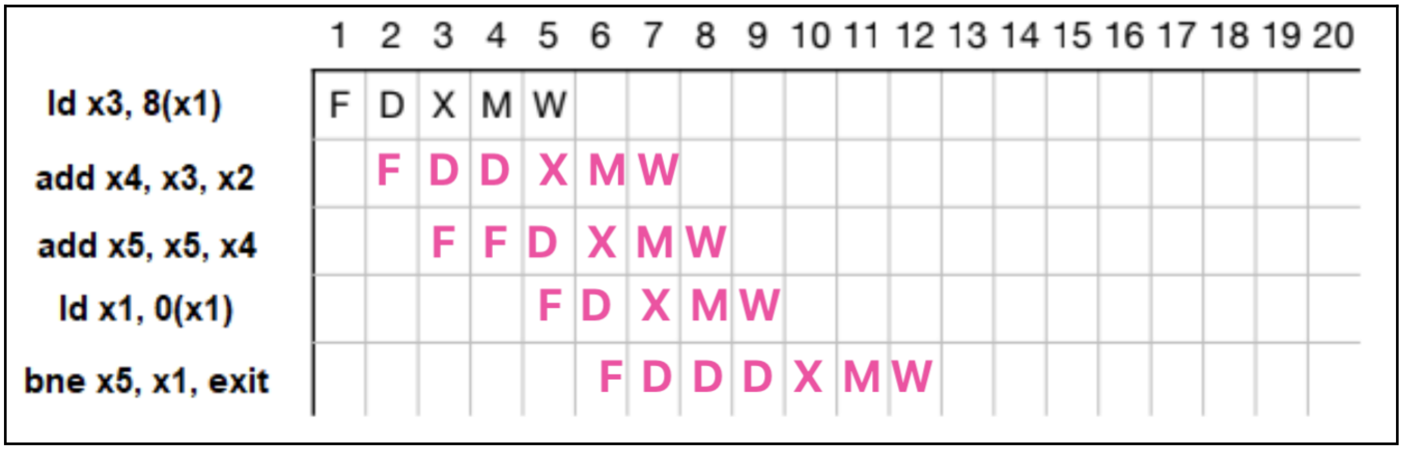
RAW on x1 between I4–I5

RAW on ×3 between I1–I2

RAW on ×4 between I2–I3

RAW on x5 between I3–I5

B) **(5 pts)** Assume the processor has full forwarding paths (including half-cycle write back/fast register file) **as well as stall capability in the event of general data hazard detection not resolved by forwarding.** Fill in the diagram to show how the code progresses through the pipeline:

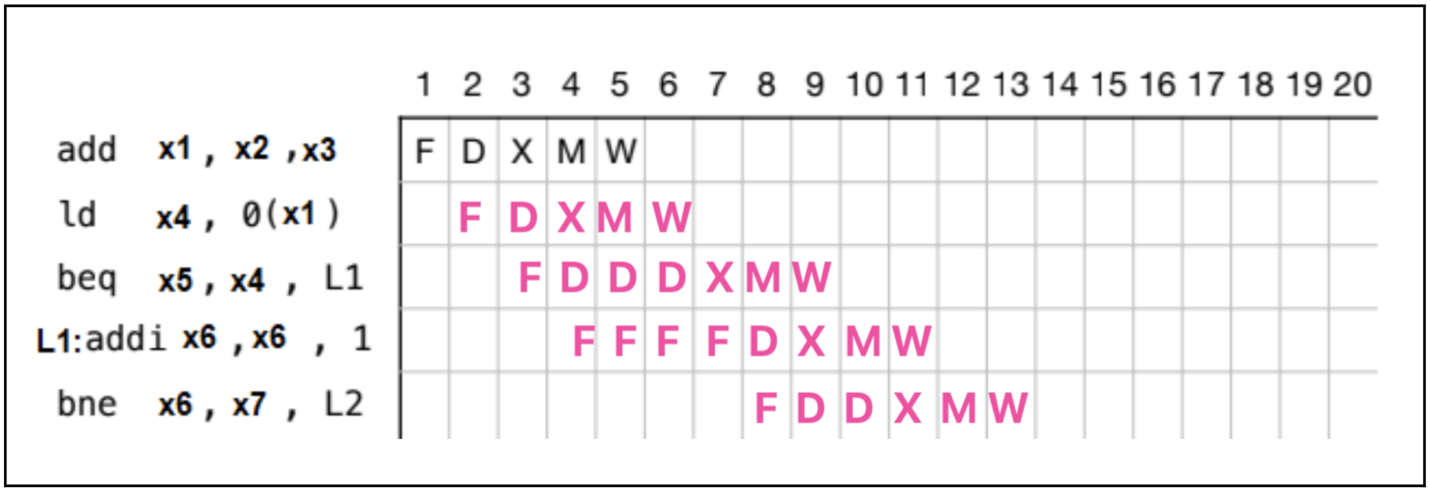


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ld x3, 8(x1)	F	D	X	M	W															
add x4, x3, x2		F	D	D	X	M	W													
add x5, x5, x4			F	F	D	X	M	W												
ld x1, 0(x1)					F	D	X	M	W											
bne x5, x1, exit						F	D	D	D	X	M	W								

Q4 (10 pts)

A) **(7 pts)** Assume the processor has full forwarding paths (including half-cycle write back), has stall capability based on non-forwardable data hazard detection, and resolves branches in the **decode stage**.

Also assume that if the decode stage does not have the data available yet to determine branch taken/not taken, it will stall the branch instruction at that stage until the data becomes available from previous instruction. ***Assume the branch is taken***. The table correctly shows next instruction (addi) being taken after the beq instruction. Fill in the table to show how the pipeline progresses:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
add x1 , x2 ,x3	F	D	X	M	W															
ld x4 , 0(x1)		F	D	X	M	W														
beq x5 , x4 , L1			F	D	D	D	X	M	W											
L1:addi x6 ,x6 , 1				F	F	F	F	D	X	M	W									
bne x6 , x7 , L2								F	D	D	X	M	W							

B) **(3 pts)** You are considering removing the forwarding paths from your processor because they are too expensive. How will removing the forwarding paths qualitatively affect the 3 terms in the CPU Performance Iron Law equation? **For each term shown below**, explain why it will increase, stay the same, or decrease. If it depends on other factors, explain those too. You can assume your processor is pipelined with full forwarding (initially) and it is executing a generic workload.

(i) Instruction count

The original program and it's instruction count will not be influenced by forwarding paths. The instruction count will **stay the same**.

(ii) CPI

Removing forwarding paths will increase data hazards, which will require NOP & stalling. Therefore the CPI **will increase**.

(iii) Clock Period

Removing forwarding paths will allow the clock period to **decrease** (unless the forwarding paths are **not** on the critical path. In that case, clock period would **stay the same**.)