Student Name:

Student ID:

COMP 303 - Computer Architecture: HW3

Due: Nov 13, 2019, 4:00 pm

Instructor: Didem Unat

Corresponding TAs: Aditya Sasongko

Notes: This is an individual assignment (no groups). You may discuss the problems with your peers but the submitted work must be your own work. No late assignment will be accepted. Submit a SOFT copy of your assignment to the blackboard *AND* submit a HARD copy of it to the COMP 303 mailboxes or bring it to the class. This assignment is worth 4% of your total grade.

Problem 1

(25+25+10 pts) Consider a six-stage pipeline (IF, ID, EX, M1, M2, WB) processor where memory operations take two pipeline stages, so load result values are not available until after the M2 (memory) stage is completed. For this problem we will use the following assembly language sequence.

```
0  myloop:
1       ADD r1, r3, r1
2       LW r1, 0(r1)
3       SW r2, 0(r1)
4       SUBI r4, r4, 4
5       LW r3, 0(r4)
6       ADD r2, r2, r1
7       BNEZ r4, myloop
8       SUB r1, r3, r1
9       OR r5, r1, r2
10      ...
```

a) Draw the pipeline timing diagram using the table below for the code sequence above.

- Start with the first instruction of the loop (line 1) and draw the timing diagram through one full loop iteration including the first ADD of the second loop iteration.
- Assume branches are resolved in the execution stage (EX) and branches are NOT taken.
- Show all stalls with an X and all the flushes with an F.
- Show the forwardings with an arrow needed to help eliminate stalls and fill the forwarding table with source and destination pipeline registers.

Cycles (time) →

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
ADD																		
LW																		
SW																		
SUBI																		
LW																		
ADD																		
Bnez																		
….																		

Source Instruction (e.g. ADD r1, r3, r1)	**Source Location (e.g. ID/EX)**	**Destination Instruction**	**Destination Location (e.g. ID/EX)**	**Due to Register (e.g. R1)**

b) Now assume that the architecture uses branch delay slots to try to eliminate all the branch penalty and compiler rearranges the code for the delay slots as follows:

```
0  myloop:
1       ADD r1, r3, r1
2       LW r1, 0(r1)
3       SW r2, 0(r1)
4       SUBI r4, r4, 4
5       BNEZ r4, myloop
6       LW r3, 0(r4)
7       ADD r2, r2, r1
8       SUB r1, r3, r1
9       OR r5, r1, r2
10      ...
```

Note that instructions 6 and 7 (lines 6 and 7) are "delay slot" instructions which means that they will be executed even when the branch in line 5 is taken. Draw the pipeline timing diagram and fill the forwarding table for this code. Start with the first instruction of the loop (line 1) and draw the timing diagram through one full loop iteration including the first ADD of the second loop iteration.

Cycles (time) →

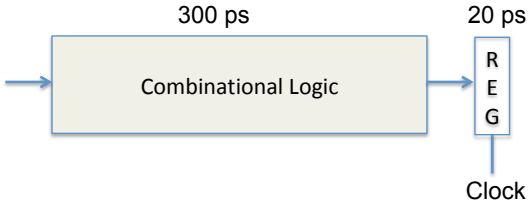
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
ADD																		
LW																		
SW																		
SUBI																		
Bnez																		
LW																		
ADD																		
...																		

Source Instruction (e.g. ADD r1, r3, r1)	**Source Location (e.g. ID/EX)**	**Destination Instruction**	**Destination Location (e.g. ID/EX)**	**Due to Register (e.g. R1)**

c) Fill the table with RAW, WAW, and WAR dependencies for the above code example in (b) where delay slot instructions are used.

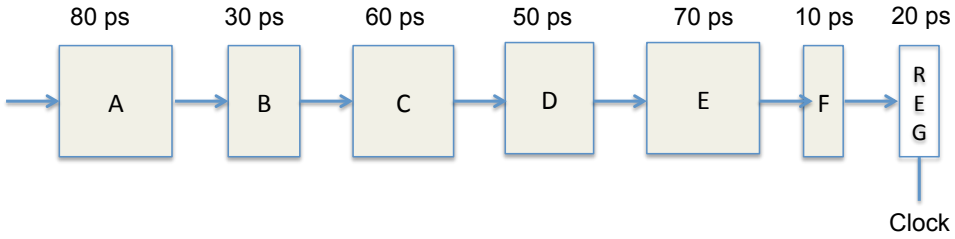
RAW			WAR			WAW		
From Instruction (Read)	To Instruction (Write)	Due to Register	From Instruction (Write)	To instruction (Read)	Due to Register	From Instruction (Write)	To Instruction (Write)	Due to Register

Problem 2



(24 pts) In figure above, unpipelined computation hardware is shown. On each 320 ps cycle, the system spends 300 ps evaluation of a combinational logic function and 20 ps storing the results in an output register. As a result, the latency of the hardware is 320 ps and throughput is 3.125 GIPS (giga instructions per second).

We would like to divide this combinational logic into separate logics; sequence of six blocks, named A to F, having delays of 80, 30, 60, 50, 70, and 10 ps, respectively as illustrated below:



Then we can create pipeline versions of this design by inserting pipeline registers between pairs of these blocks. Different combinations of pipeline depth (how many stages) and maximum throughput arise, depending on where we insert the pipeline registers. Assume that each pipeline register has a delay of 20 ps.

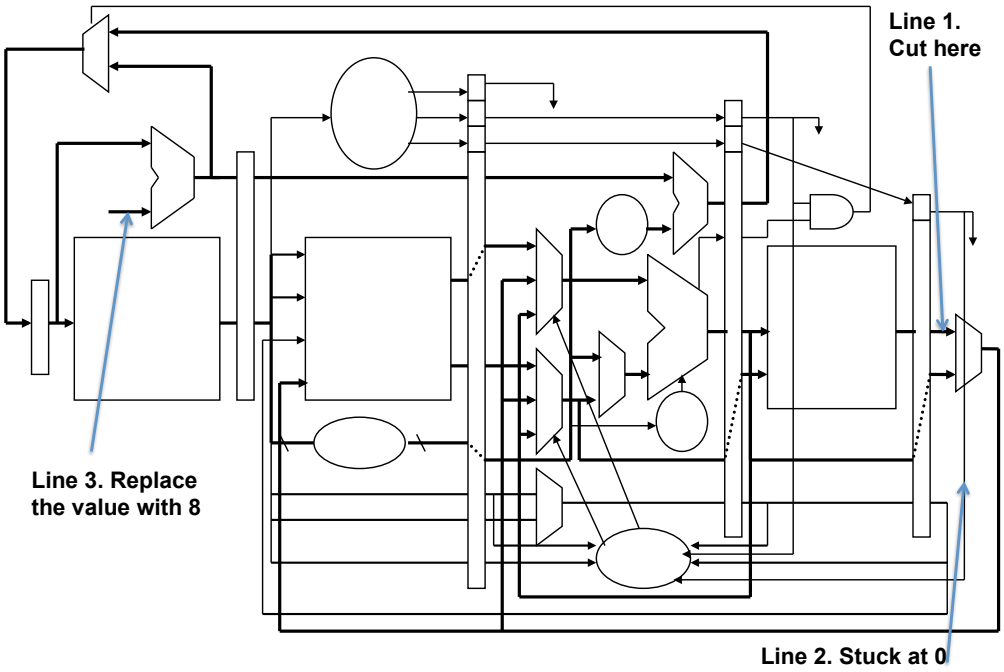
a) Inserting a single register gives a 2-stage pipeline. Where should the register be inserted to maximize throughput? What would be the clock cycle time, throughput and latency?

b) Where should two registers be inserted to maximize the throughput of a 3-stage pipeline? What would be the clock cycle time, throughput and latency?

c) Where should three registers be inserted to maximize the throughput of a 4-stage pipeline? What would be the clock cycle time, throughput and latency?

d) What is the minimum number of stages that would yield a design with the maximum achievable throughput? Describe this design, its clock cycle time, throughput, and its latency.

Problem 3



(16 pts) For the MIPS datapath shown above there parts are malfunctioning. For each one: Provide a snippet of code that will fail and that will still work. Explain in words the negative consequence of cutting the line relative to the working, unmodified processor.

a) Line 1:

Failed Code	Working Code	Explanation (be concise)

b) Line 2:

Failed Code	Working Code	Explanation (be concise)

c) Line 3: Explain the consequences of replacing the value with 8.