UNIVERSITY *of* HOUSTON

Computer Organization & Architecture
College of Natural Sciences and Mathematics
Fall 2020

Exam 2.

This exam is given under the University of Houston Honor Code System. You must observe and sign the Honor Pledge: "I have neither given nor received aid on this exam." Your print name and signature below signifies your compliance with this honor code. Note that we will not grade your exam unless it is acknowledged.

Student ID (last four digits)	0918
Name:	Samita Damani

By clicking the checkbox below, I acknowledge my responsibility and commitment to the Academic Honor Code.

☒ Acknowledgment

1. ______________ (28 pts)
2. ______________ (32 pts)
3. ______________ (40 pts)

Total (100 pts) _____________

Make sure you use a software that allows filling PDF forms.

- *Foxit (Web, Android, iOS, Windows, Mac) to edit PDFs everywhere*
- *PDF Expert (iOS, Mac) to quickly edit PDF text and images*
- *PDFelement (Android, iOS,Windows, Mac) to edit PDFs*
- *Adobe Acrobat (Windows, Mac) to create detailed PDFs and forms*

Make sure you save your answers (doublecheck before submit). In addition, doublecheck that you have submitted your filled pdf.

If you see dots, it means that you are using a pdf software or a setting that has missing font. You may resolve this by switching to a different pdf software (such what mentioned above) or write down your solution in a separate file.

```
cd lab0-your-github-account-name/exams/
git add "*"
git commit -m "Exam2"
git push origin master
```

1. **Pipelining (28 pts).** You are given a non-pipelined processor design which has a cycle time of 10ns and average CPI of **1.4**. Calculate the latency speedup in the following questions.(**speedup over the non-pipelined version)**

(a). **8pt** What is the best speedup you can get by pipelining it into **8 stages (assume no pipeline hazard and equal time of pipeline stages)**? What is the cycle time? Your calculation

LATENCY = 10/8 = 1.25   SPEEDUP = CPI (CYCLETIME)/LATENCY = 14/1.25 = 11.2

Cycle time	=	1.25
Speedup	=	11.2

S1	S2	S3	S4	S5	S6	S7	S8
1.1ns	0.9ns	2.1ns	1.4ns	1.4ns	1.1ns	0.9ns	1.1ns

(b). **8pt.** If the 8 stages have latency above, what is the speedup you can get compared to the original processor (Assume that the time also include extra delay for the registers between pipeline stages). Your calculation

MAX = 2.1 ns - cycle time (slowest stage) speedup = 1.4(10)/2.1 = 6.67

Cycle time	=	2.1
Speedup	=	6.67

2. Data Hazard. (32 pts). Use the following code fragment:

I1	lw \$s2, 0(\$s1)
I2	lw \$s1, 40(\$s3)
I3	sub \$s3, \$s1, \$s2
I4	add \$s3, \$s3, \$s3
I5	xor \$s4, \$s3, \$zero

(a). Identify all **read after write** data dependencies (by instruction pair and register involved) (**12 pts**):

(I2,I3) on \$s1 (I3,I4) on \$s3 (I4,I5) on \$s3

(b) **20 pts.** Using a pipeline timing char below, show the timing of this instruction sequence for a **7 stage pipeline** (**IF, ID1, ID2, EX1, EX2, MEM, WB**) **with full data forwarding**. Assume registers can be written and read in the same cycle, during writeback. Indicate any necessary stalls on the pipeline diagram. At any time, a pipeline stage can be only occupied by at most one instruction.

I1	*lw $s2, 0($s1)*
I2	*lw $s1, 40($s3)*
I3	*sub $s3, $s1, $s2*
I4	*add $s3, $s3, $s2*
I5	*xor $s4, $s3, $zero*

	Clock Cycles																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
lw	IF	ID1	ID2	EX1	EX2	MEM	WB																		
lw		IF	ID1	ID2	S	S	EX1	EX2	MEM	WB															
sub			IF	ID1	S	S	ID2	S	S	EX1	EX2	MEM	WB												
add				IF	S	S	ID1	S	S	ID2	S	E1	E2	MEM	WB										
or							IF	S	S	ID1	S	ID2	S	EX1	EX2	MEM	WB								

When filling the table, please use IF for IF; ID1 for ID1; ID2 for ID2; E1 for EX1; E2 for EX2; M for MEM; WB for WB; S for pipeline stall. Note that the table above will not be graded. **Your grade will be based on the table below. Indicate clock time that an instruction starts a pipeline stage.**

	IF	ID1	ID2	EX1	EX2	MEM	WB
I1	1	2	3	4	5	6	7
I2	2	3	4	7	8	9	10
I3	3	4	7	10	11	12	13
I4	4	7	10	12	13	14	15
I5	7	10	12	14	15	16	17

3. Out-of-order Execution. 40 pts. Consider the following code sequence (Rx are integer register and Fx are floating point registers). Instructions are in {dest, src, src} format except for branches and stores which are {src, src}; consider MUL.D, ADD.D as FP instructions. ADDI is integer ADD.

```
I1. L.D  F8, 32(R1)
I2. MUL.D F4, F8, F2
I3. ADD.D F5, F8, F12
I4. ADD.D F5, F5, F5
I5. ADDI R1, R1, 4
```

(a). Identify all of the dependencies (**15 pts**):

RAW	WAW	WAR
(I2,I1) on F8 (I3,I1) on F8 (I4,I3) on F5	(I3,I4) on F5	(I1,I5) on R1

(b). (25 pts) Using Tomasulo's algorithm, fill in the table below to show the clock cycle on which each instruction progresses through the corresponding functional unit. Start at clock cycle 0. For each of the instructions, write the RS identifier that the instruction issues to, write the sources (e.g., R1 if read from the register file, ADDI1 if waiting on ADDI1's result), and the cycle in which the instruction issues, executes, and writes-back its result. **Put x for operation that doesn't use memory. For execute, specify both starting time and finishing time using format a,b where a is the starting cycle and b is the finishing cycle. Assume:**

1. **A1: One instruction can issue (allocate) per cycle**
2. **A2: Reservation stations**: Load/store unit (5), FP ADD(3), FP MD(3), Integer ADD (3); call these LD1-LD5, ADF1, ADF2, ADF3, MDF1, MDF2, ADDI1, ADDI2, ADDI3
3. **A3:** There is **one FP adder, one FP multiplier, one Integer adder**
4. **A4:** Functional units **are not pipelined**
5. **A5: Latency: FP ADD(2 cycles), FP MUL (8 cycles), Integer ADD (1 cycle)**
6. **A6: Write Result and Exec occur in separate cycles** (i.e., **an instruction writes to the CDB in cycle n, and its dependent instruction starts execution in cycle n+1**)
7. **A7: There is only one CDB (means that two functional units cannot broadcast result in the same cycle – if two instructions finish execution in cycle n, one of them can write back results in cycle n+1 – priority is Integer ADD, FP ADD, FP MUL, and Load/Store – means that if Integer ADD and Load/Store compete for CDB, Integer ADD writes back result first)**
8. **A8:** If a functional unit completes execution at cycle **n**, it can be used for another instruction at cycle **n+1** (* dependent instruction uses the same functional unit starts at **n+2** – **see A6**)
9. **A9:** Following A8, if there is more than one ready instruction for a functional unit, pick the one that has the earliest issue time
10. **A10: Loads and stores take 6 cycles** in Memory stage and 1 cycle in Execute stage
11. **A11: An instruction stalls at issue if an appropriate reservation station (RS) is not available**. It may issue in the cycle *after* the previous instruction starts execution (i.e., if ADD.D X is stalled because all add RS's are occupied, and then one of the ADD.D starts execution on cycle n, then the ADD.D X can issue on cycle n+1)
12. **A12:** If more than one reservation station is available (e.g., ADF1 and ADF2), issue the instruction to the RS with the lower number (e.g., ADF1)
13. **A13:** Re-order buffer size is 64
14. **A14: If one instruction stalls at issue, the succeeding instructions also stall.**

Instruction	RS	Issue	Execute	Memory	Writeback CDB	Commit	Comments
I1. L.D F8, 32(R1)	LD1	1	2,2	3,6	8	10	
I2. MUL.D F4, F8, F2	MDF1	2	9,16	x	17	18	wait for LD1
I3. ADD.D F5, F8, F12	ADF1	3	9,10	x	11	19	wait for LD1
I4. ADD.D F5, F5, F5	ADF2	4	12,13	x	14	20	wait for ADF1
I5. ADDI R1, R1, 4	ADDI1	5	6,6	x	7	21	

How many total clock cycles required to complete? 21

Order of execution (count I1)	I1	I5	(I3,I2)	I4	-
Order of write back (count I1)	I1	I5	I3	I4	I2
Order of commit (count I1)	I1	I2	I3	I4	I5