Department of Electrical and Computer Engineering
University of Texas at Austin

EE460N Fall 2020
Y. N. Patt, Instructor
Chester Cai, Sean Stephens, Arjun Ramesh, TAs
Exam 1
October 7th, 2020

Name:	Solution
EID:	

Problem 1: 20 points
Problem 2: 10 points
Problem 3: 20 points
Problem 4: 20 points
Problem 5: 30 points
Total: 100 points

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Please read the following sentence, and if you agree, sign/print your name where requested: I have not given or received any unauthorized help on this exam.

Signature:	Solution

Good Luck!

General Instructions:

1. You are free to use anything in the Handouts section of the course website that is listed under "Course Related Handouts" or "LC-3b Handouts." In particular, Appendix A and Appendix C may be of use. Anything other than that from the course website, textbooks, or the Internet is not allowed and considered unauthorized access.
2. Use of a calculator is not required but is permitted.
3. If you have any questions, join the class Zoom link and ask a TA. You do not need to stay on the Zoom call during the exam unless you have questions.
4. Announcements will be posted here. Check this page periodically throughout the exam.
5. You may take the exam by printing it, editing a PDF, or editing a Google Doc. Read the instructions for your preferred method below.
6. **You are required to stop working on the exam promptly at 6:30 PM.**

Printing or editing a PDF:

1. Download and save the PDF.
2. Edit the PDF to fill in answers with a software of your choice. Feel free to show your work in the available space. You may also choose to print the exam and solve it on paper.
3. When you are ready to submit your exam, save the edited PDF as "Exam 1 <your name>"; if you printed your exam, scan in your written answers as a PDF with the same name. You may use a scanner or an app such as CamScanner.
4. Upload the PDF to Gradescope by 6:40 PM. The entry code for Gradescope is **9RPGX3**.

Editing a Google Doc:

1. Save a copy of the document to your Google Drive.
2. While working on the exam, **DO NOT expand any boxes that are given to you.** Feel free to show your work in the available space. If you need more space, you are writing too much.
3. When you are ready to submit your exam, click "File"-> "Print" and select "Save as PDF". Save the edited PDF as "Exam 1 <your name>".
4. Upload the PDF to Gradescope by 6:40 PM. The entry code for Gradescope is **9RPGX3**.

Problem 1 (20 points): Answer each question in 20 words or fewer. Note: For each of the four answers below, if you leave the box empty, you will receive one point of the five.

Part a (5 points): Must the condition codes be saved on a context switch? Why?

Yes. We may switch before a branch which needs its condition codes when it resumes.

Part b (5 points): Not all ISAs have a LDCTX instruction, but most operating systems today perform the function of the LDCTX instruction. When and why is it necessary to perform that function.

Load Context - It loads all the architectural state of another process (regs, PSR, ...) on a context switch.

Part c (5 points): A company called MIPS was initially adamant in refusing to provide any hardware interlock if the function it performed could be performed in software. To get a value from memory and multiply it by two, most compilers would produce the two instruction sequence

LD R1,A
ADD R1,R1,R1.

This two instruction sequence would not work in a pipelined microarchitecture without stalling the ADD instruction until the LD got data back from memory. MIPS preferred to maintain correct execution of the program in a pipelined processor with software. How did they do it?

They let the compiler insert independent instructions or no-ops between the LD and ADD.

Part d (5 points): The AOBLEQ instruction, when included in the ISA provides benefit to the use of on-chip storage for instructions. How so?

It saves space in the instruction cache because one instruction is fetched for a lot of work instead of several instructions.

Problem 2 (10 points): XYZ computer company has been selling Processor A for several years. Processor A uses instruction types 1, 2, 3, and 4 to execute the only program that matters. The company decided to design a new processor (Processor B) and recompiled that program using instruction types 1, 2, and 5. Neither processor is pipelined. The table below shows the CPI of each instruction type, and the fraction of instructions of each type executed by the program in each processor.

Processor A			Processor B		
Inst Type	Ratio	CPI	Inst Type	Ratio	CPI
1	40%	4	1	40%	4
2	30%	5	2	30%	5
3	20%	6	5	30%	7
4	10%	5			

- Processor B needs to execute 10% fewer instructions than Processor A on this program.
- The clock frequency for Processor B will be 10% faster than Processor A

Part a (4 pts): What is the CPI for each processor on this program?

Processor A: 4.8

Processor B: 5.2

Part b (6 pts): Which processor will execute the program faster? Show your work.

Time = (Cycles/Instr) * (Time/Cycle) * Instructions
Freq = Cycle / Time
Let A take n instructions. Thus B takes .9n instructions.
Let A have f frequency. Thus B has 1.1f frequency.
Time A = 4.8 * [1/(f)] * n = 4.8n/f
Time B = 5.2 * [1/(1.1f)] * .9n = 4.25n/f
n and f are both positive, so Time B < Time A
Processor B is faster.

Note that cycle frequency is 10% faster is not the same as cycle time being 10% faster.

Problem 3 (20 points): An LC-3b computer has an in-order pipeline with a gshare predictor. The PHT is indexed using the BHR along with 4 bits of the address (PC[10], PC[4], PC[3], and PC[1]) of the branch instruction being predicted. The rightmost bit of the BHR contains the **predicted** direction of the most recent (previous) branch, and is updated at the time a branch is predicted. The BHR is updated by shifting the **prediction** into its right-most bit. The predicted direction is used to fetch the next instruction in the next clock cycle. The PHT is updated when the **actual** direction of the branch is resolved. Unlike some commercial products, the BHR is not corrected when the actual direction of the branch is resolved.

The snapshot of the BHR and PHT shown below was taken at the end of the cycle in which the PHT was updated as a result of conditional branch A being resolved. No conditional branches entered the pipeline after conditional branch A was fetched.

BHR
1001

	PHT
0	10
1	01
2	01
3	01
4	00
5	00
6	11
7	10
8	11
9	00
10	01
11	11
12	11
13	01
14	01
15	10

Part a (5 points): List all possible values in the BHR at the time conditional branch A was fetched. Use as many entries as needed.

0100	1100		

Part b (15 points): Each case below is an independent situation, resulting in the snapshot of the BHR and PHT above. For each case, the address of conditional branch A, and both its predicted direction and actual resolved direction are shown. Given the information about each branch shown below, **is the snapshot of BHR and PHT possible?** Explain why you answer yes or no.

Case 1: **PC = x2D54 (0010 1101 0101 0100)**. Predicted **taken**, actually **not taken**.

PC bits = 1100.
BHR = 0100: 0100 ^ 1100 = 1000 = 8. PHT entry = 11. Not possible since actually not taken would have decremented this entry.
BHR = 1100: 1100 ^ 1100 = 0000 = 0. PHT entry = 10. Possible, was 11 before and got decremented.
BHR is valid since lowest bit is a 1, the predicted result.

Overall, Case 1 is Possible. The previous BHR must've been 1100 and PHT entry 10 was binary 11.

Case 2: **PC = x7278 (0111 0010 0111 1000)**. Predicted **not taken**, actually **taken**.

BHR is NOT valid since its lowest bit is 1, contradicting the **not taken** prediction.

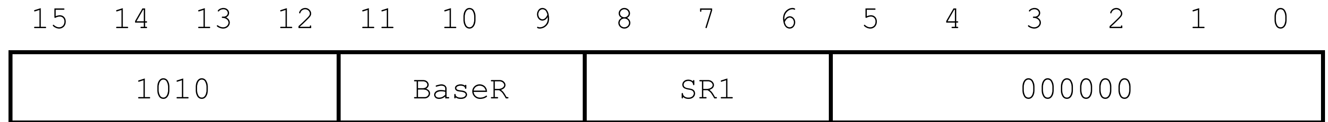
Overall, Case 2 is Not Possible since the BHR is invalid.

Case 3: **PC = xF940 (1111 1001 0100 0000)**. Predicted to be **taken**, actually **not taken**.

PC bits = 0000
BHR = 0100: 0100 ^ 0000 = 0100 = 4. PHT entry = 00. Not possible, couldn't have predicted taken.
BHR = 1100: 1100 ^ 0000 = 1100 = 12. PHT entry = 11. Not possible, couldn't have been not taken because would have decremented counter.
BHR is valid but doesn't matter since neither PHT entry is possible.

Overall, Case 3 is Not Possible. Neither PHT entry could correspond to this branch.

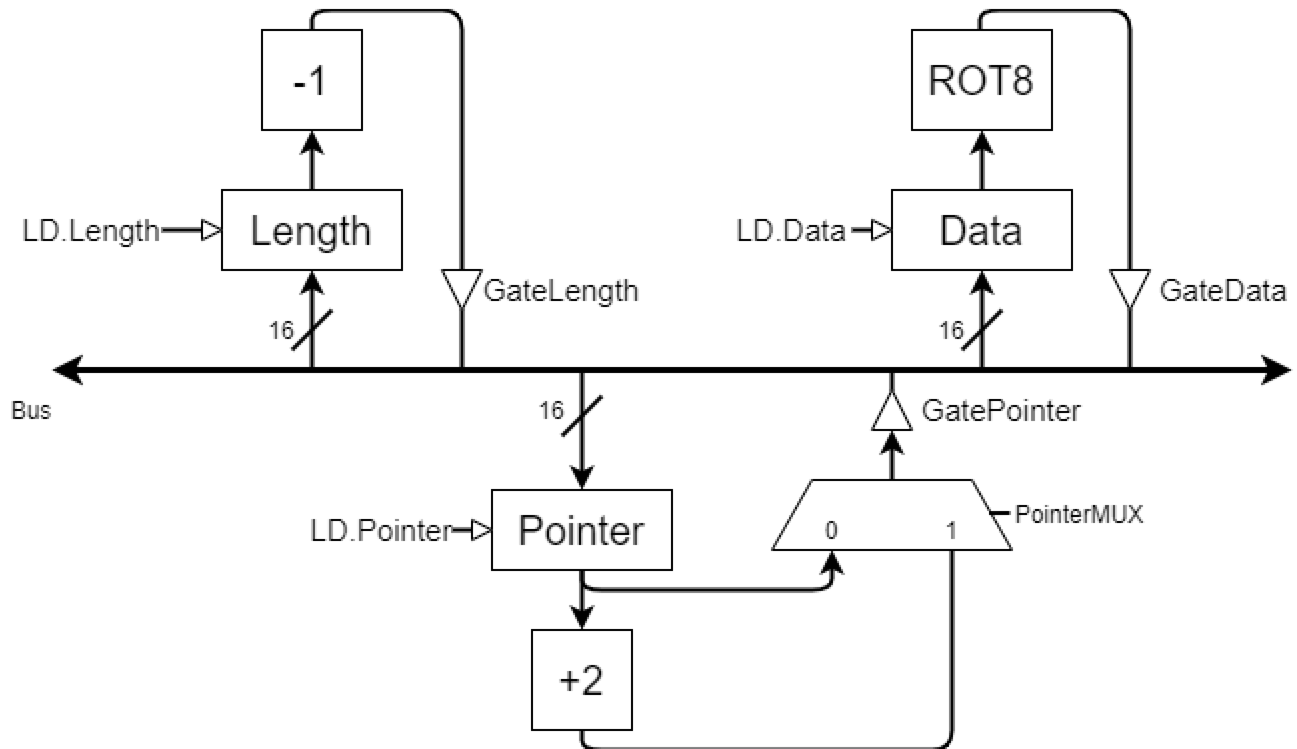
Problem 4 (20 Points): In your first lab assignment you were asked to write a shuffle program that exchanges the high and low bytes of 16 bit words in an array. Your job here is to implement a SHUFFLE instruction to do the job.



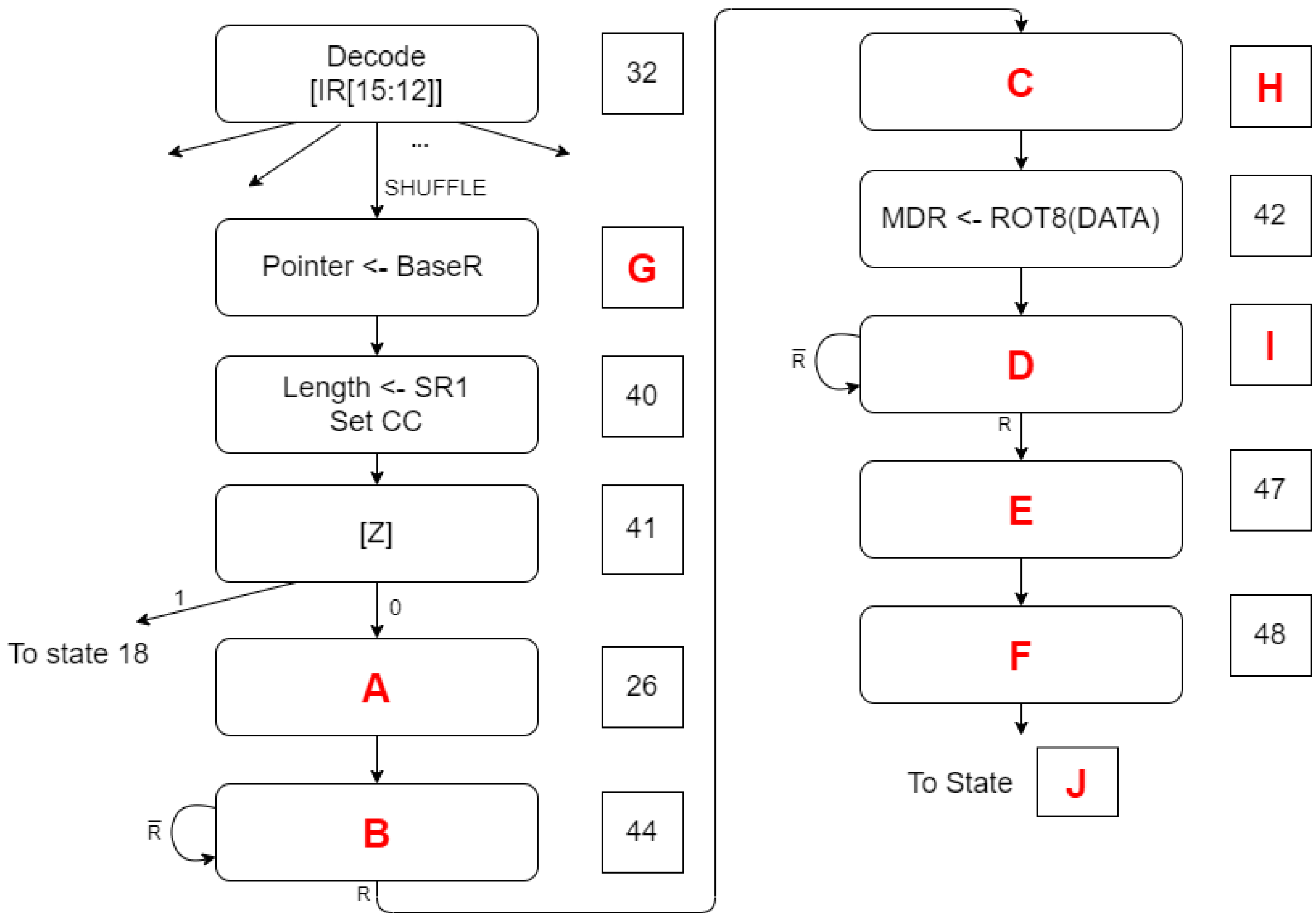
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1010				BaseR			SR1			000000					

BaseR contains a pointer to the start of the array. SR1 contains the number of words in the array.

To accomplish this, we need to add three new registers to the data path, along with some additional logic, as shown below. ROT8 rotates the 16-bit input by 8 bit. We will also need to add 7 control signals, also shown.



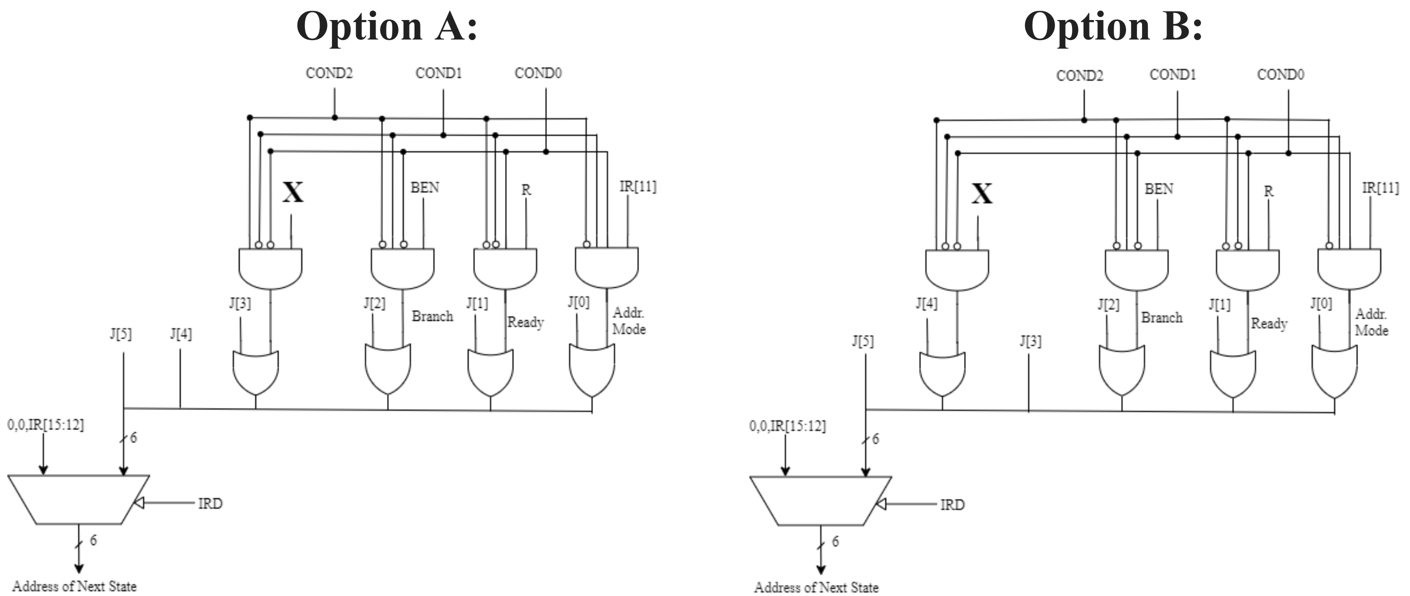
Part a (8 points): Note there are 6 states and 4 state numbers (labeled A, B, C … J) in the state diagram shown below. Put into the boxes the corresponding information about each state and state number.



A	MAR ← Pointer	B	MDR ← M[MAR]	C	DATA ← MDR
D	M[MAR] ← MDR	E	Pointer ← Pointer + 2	F	Length ← Length - 1 Set CC

G	10	H	46	I	45	J	41

Part b (6 Points): Shown below are two possible changes to the microsequencer to implement SHUFFLE.



Which option would you use? Explain.

Option A. The choice that the microsequencer decision depends on is leaving state 41. It chooses between states 18 and 26, which are 8 apart. Thus, we must use option A, since that one allows X to set the 3rd bit of the microsequencer, incrementing the state by 8.

What is the signal X in the microsequencer? NOT Z

Part c (6 points): Fill in the following table of control store signals for states 40, 41 and 44. Please fill in 0 for signals that are "Don't care". If there are multiple paths, you may pick any of them. Refer to page 8 of Appendix C for more information about what each signal does.

State No.	cond2	cond1	cond0	J[5:0]	SR1MUX	ALUK[1:0]	GateMARMUX	MIO.EN	R.W
40	0	0	0	101001	1	11	0	0	0
41	1	0	0	010010	0	00	0	0	0
44	0	0	1	101100	0	00	0	1	0

Note in state 40, there is another path from the reg file to the bus through GateMARMUX

Problem 5 (30 points): An out-of-order processor executes instructions based on the original Tomasulo algorithm without in-order retirement. The processor implements a standard 4-stage pipeline: Fetch, Decode, Execute, Writeback. The Execute stage of the pipeline contains *one non-pipelined adder*, *one non-pipelined multiplier*, and *one branch unit*.

- Fetch and Decode takes one cycle each.
- The result of a functional unit is broadcast during the writeback stage and is ready for use the cycle immediately after writeback.
- ***An ADD takes 3 cycles*** to execute, ***MUL takes 5 cycles***, and ***BR takes 1 cycle***.
- Branch prediction is implemented using a *2-bit saturating counter*, with initial state 10.
- A correct prediction allows the next instruction to be fetched in the cycle after the branch is fetched. A misprediction is resolved at the end of the Execute stage, i.e, the next instruction will be fetched immediately AFTER the Execute stage of the branch.
- All three functional units have three-entry reservation stations that are initially empty, and are allocated in a top-to-bottom manner.
- Entries are put into reservation stations at the end of Decode and removed at the end of Writeback.
- Instructions with no dependencies can start executing immediately after Decode.

The following snippet of code is executed until the instruction at I5 has completed.
(Hint: I2 is not a branch instruction.)

Instr #	Label	Instruction
`I1`	`TARGET`	`MUL R2, R0, R1`
`I2`		`ADD R1, R1, R7`
`I3`		`BRp TARGET`
`I4`		`ADD R5, R2, R5`
`I5`		`MUL R6, R1, R3`

To find TARGET: At the end cycle 6, 5 instructions have been decoded. We see a total of 4 instructions in the RS (2 ADD and 2 MUL). This implies the second instruction has to be an ADD. The branch could not have been resolved at that point, so at cycle 6, the two instructions are after the TARGET. TARGET can only be on I1 or I4 to get 4 instructions in RS, We know it's not I4 because R6 is valid at the end of cycle 6.

The state of the register alias table (RAT) is partially shown before the code starts executing, after cycle 6, and after the code completes execution.

Before *Cycle 1*

	V	Tag	Value
R0	1	-	4
R1	1	-	4
R2	1	-	0
R3	1	-	2
R4	1	-	-1
R5	1	-	10
R6	1	-	6
R7	1	-	-2

After *Cycle 6*

	V	Tag	Value
R0	1	-	4
R1	0	β	-
R2	0	ρ	-
R3	1	-	2
R4	1	-	-1
R5	1	-	10
R6	1	-	6
R7	1	-	-2

After *Completion*

	V	Tag	Value
R0	1	-	4
R1	1	-	0
R2	1	-	8
R3	1	-	2
R4	1	-	-1
R5	1	-	18
R6	1	-	0
R7	1	-	-2

The state of the ADD and MUL reservation stations are partially shown at the end of cycle 6.

ADD RS

	Valid	*Tag*	*Value*	*Valid*	*Tag*	*Value*
α	1	-	4	1	-	-2
β	0	α	-	1	-	-2
γ						

MUL RS

	Valid	*Tag*	*Value*	*Valid*	*Tag*	*Value*
π	1	-	4	1	-	4
ρ	1	-	4	0	α	-
σ						

Your job:

1. Fill in the missing entries in the program snippet (including the location of the label TARGET), RATs, and ADD and MUL reservation stations. You might find it helpful to fill in the timing diagram until cycle 6.

2. Complete the dynamic timing diagram below for the execution of the program snippet, as we have done in class. **Only committed instructions need to be shown.**

- Each row corresponds to one dynamic instruction. The leftmost column identifies the static instruction as I1, I2, etc.
- Use F, D, M (for MUL), A (for ADD), B (for Branch) to indicate what is going on with each instruction during each clock cycle.
- If a result is written to a register, write that register number in the cycle the writeback occurs, as we did in class. Branch instructions do not write back.
- Use * to indicate a clock cycle when an instruction is waiting to continue processing.
- Use as many rows/columns as needed.

Inst	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
I1	F	D	M	M	M	M	M	R2																	
I2		F	D	A	A	A	R1																		
I3			F	D	*	*	*	B																	
I1				F	D	*	*	M	M	M	M	M	R2												
I2					F	D	*	A	A	A	R1														
I3						F	D	*	*	*	*	B													
I4													F	D	A	A	A	R5							
I5														F	D	M	M	M	M	M	R6				