Department of Electrical and Computer Engineering
The University of Texas at Austin

ECE 306 Fall 2023
Instructor: Yale N. Patt
TAs: Chester Cai, Jaeyoung Park, Sophia Jiang, Anna Guo, Asher Nederveld, Edgar Turcotte, Nadia Houston, Varun Arumugam, Ali Mansoorshahi
Exam 1
October 11, 2023

Name: ____

Problem 1 (20 points): _______

Problem 2 (15 points): _______

Problem 3 (15 points): _______

Problem 4 (25 points): _______

Problem 5 (25 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.

Note: Please be sure your name is recorded on each sheet of the exam.

Please read the following sentence, and if you agree, sign where requested:
I have not given nor received any unauthorized help on this exam.

Signature: ____

GOOD LUCK!

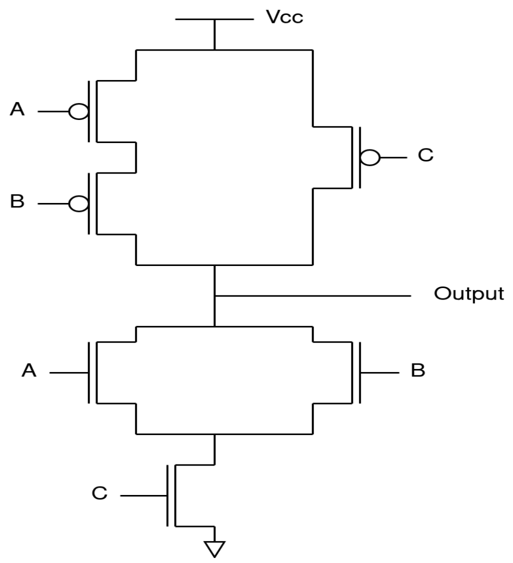
Name: ___________________________________

Question 1 (20 points): Answer the following questions.

Note: For each of the four answers below, if you leave the box empty, you will receive one point.

Part a (5 points): The TRAP instruction provides a mechanism for a user program to ask the Operating System to do something on behalf of the user program. How does the Operating System know what the user program wants it to do (in 15 words or fewer)?

Part b (5 points): What logic function is performed by the following transistor diagram? Complete the truth table for that logic function.



A	B	C	Output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Part c (5 points): You know from constructing programming logic arrays that with a sufficient number of AND, OR, and NOT gates, one can implement every logic function, regardless how many input variables there are. If you only have AND and NOT gates, can you still implement every logic function, regardless how many input variables there are. (**Circle one:** YES/NO) Explain in 15 words or fewer:

Part d (5 points): The computer has just executed the instruction A:

0101 010 010 1 00000

and the computer is now at state 18, about to start executing the next instruction B, which is:

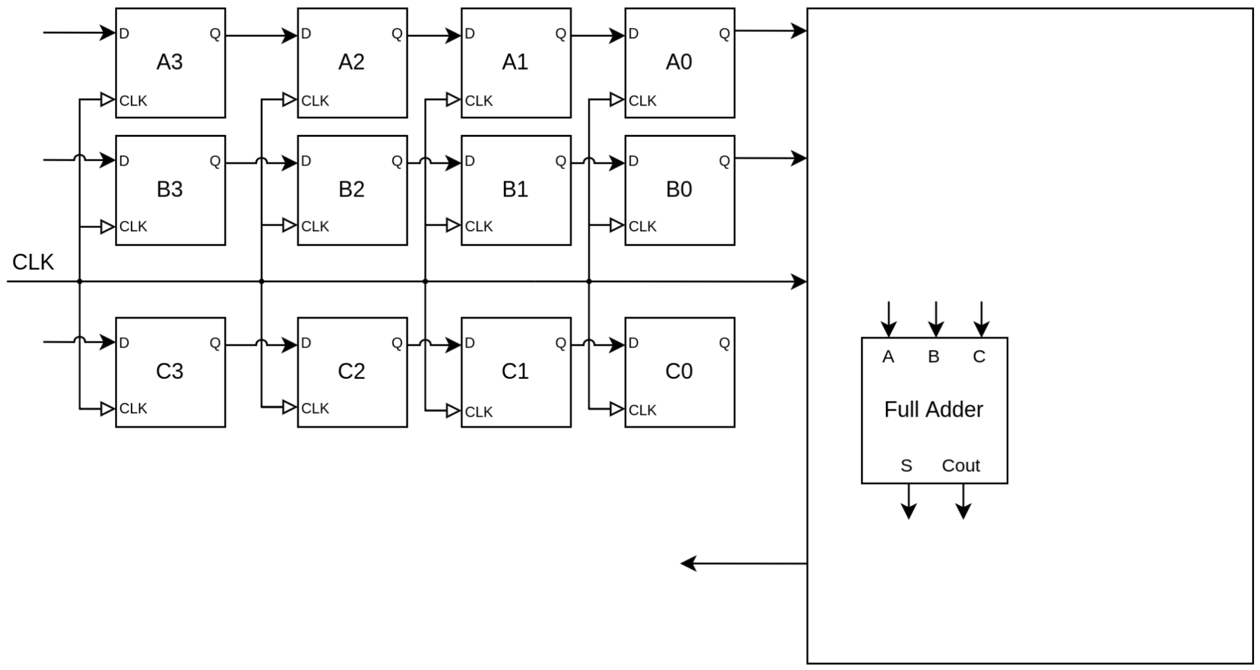
0000 011 000011111

List the states that the microarchitecture will execute, starting with state 18 when processing instruction B.

Name: ___________________________________

Question 2 (15 points): We wish to add two unsigned integers with the logic blocks shown below. You can assume that A and B are each stored in four flip flops A3-A0 and B3-B0 (least significant bit in A0/B0) and that the sum is to be stored in C3-C0 after four clock cycles.

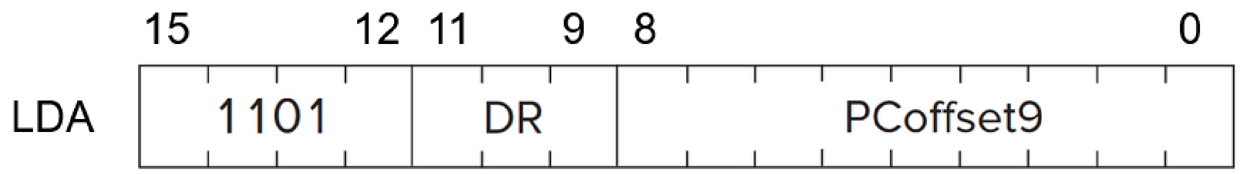
Your job: Add the additional logic and connections to the figure to accomplish this. Some inputs and outputs have already been drawn for you. You may add more, or leave some unused if you wish. Note that you only have one Full Adder, and can not add any more. You are allowed to add Constant Values, NANDs, NORs, D-Latches, and FlipFlops. You are not allowed to construct additional Full Adders with NANDs and NORs. You can assume that any added flip flops or latches start with value "0".



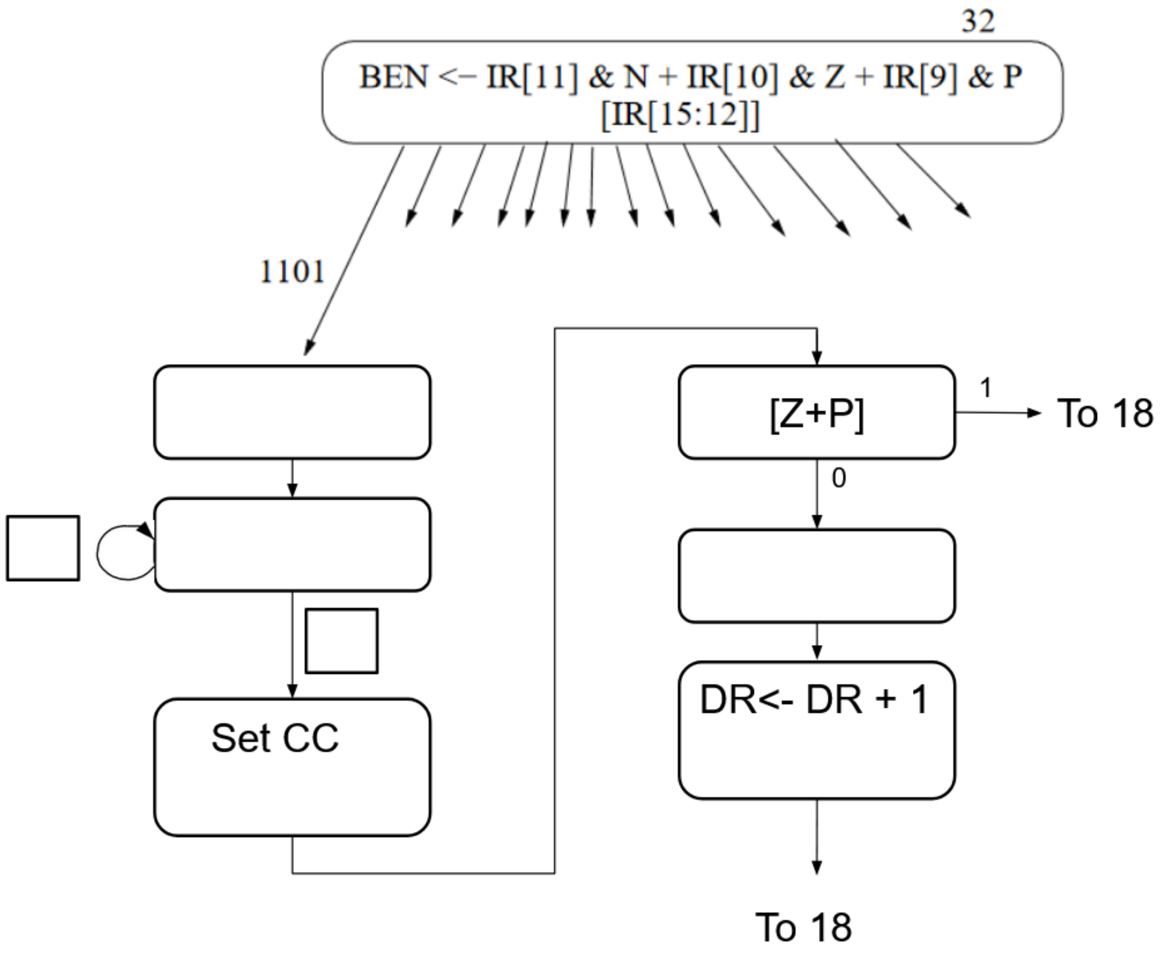
Name: ___________________________________

Problem 3. (15 points) We wish to use the unused opcode to add a new instruction Load absolute (LDA) to the LC-3 instruction set. LDA will take a 16-bit 2's complement signed integer stored in memory, form its absolute value, load the absolute value into the register specified by bits [11:9] of the instruction, and set the condition codes based on the absolute value. The addressing mode for determining the address of the memory location where the 2's complement integer is stored is PC + offset9. This is the same addressing mode as used for LD.

The new instruction has the following format:



Your Job: Fill out the state machine below to properly implement the LDA instruction. Please note that some states have already been completely or partially filled out.



Name: ______________________________

Question 4 (25 points): An Aggie, browsing the bookstore, stumbled across Patt's 306 textbook, and said, "Hey, I guess anyone can program with this!," and proceeded to generate the following program that does nothing useful:

Assembly	**Machine Code**
.ORIG x_____	x______
AND R1, R1, #0	x5260
LD R2,A	x2402
BRz B	x0402
ADD R2,R2,R2	x1482
A ST R2,C	x3401
B TRAP x25	xF025
C ADD R3,R1.R2	x1642
.END	- -

We decided to analyze his program. The table in part a shows a number of clock cycles, **and for each clock cycle, the value(s) written at the END of that clock cycle.** Some of the values are already in the table. A dash in an entry means the value is not being updated in that particular cycle. The first instruction in the program starts in state 18 in cycle 1.

Also note that the LC-3 we are running this program on was built by an Aggie, so the number of clock cycles required to perform a memory access may not be five.

Name: ___________________________________

Part a (15 points): Fill out all missing entries in the table below, including the cycle # on the last row. Provide the value **ONLY** if it was updated in **THAT CYCLE**, use a "-" to indicate the value has not been modified in **THAT CYCLE**.

Cycle #	PC	MAR	MDR	IR	BEN	NZP	R0	R1	R2	R3
7	-	-	-	-	-		-	x0000	-	-
8	x4429									
14										
19	x442A									
24										
38										
40										
	-	-	-	xF025	-	-	-	-	-	-

Part b (4 points): Recall that in our lectures this semester, Dr. Patt has said that memory accesses take 5 clock cycles each. In our measurements, we discovered that the LC-3 we are using took a different number of clock cycles to access memory. How many?

Part c (4 points): You will note that .ORIG does not have a value after it. What value should go there?

Part d (2 points): In executing the Aggie's program, what value will the LC-3 store in **R3**?

Name: ___________________________________

Question 5 (25 points): R0 and R1 contain 16-bit bit vectors. The program in the next page determines if rotating R1 by *n* bits to the left produces the same bit vector that is in R0. If yes, the program stores the value *n* in M[x3070]. If not, the program stores -1 to M[x3070].

Rotating left a bit vector one bit consists of left shifting the bit vector one bit, and then loading into bit[0] the bit that was shifted out of bit[15].

For example, rotating left **1**111000011110000 one bit produces 111000011110000**1**.

Rotating left *n* bits is simply rotating left one bit, but doing it *n* times.

For example, rotating left **111**1000011110000 3 bits produces 1000011110000**111**.

An example of the program's execution is shown:

R0 = 1000 0000 0000 0110
R1 = 1101 0000 0000 0000
→ The program will store 3 to M[x3070]

Part a (5 points)

For each iteration, the program compares R0 with R1 and rotates R1 one bit to the left if those two registers don't match. What is the minimum number of 1-bit left rotations the program has to make to guarantee that the contents of R0 and R1 will never be the same? Add a brief explanation why.

Name: ___________________________________

Part b (20 points)

Your job: Complete the program below by supplying the missing instructions so it stores *n* in location M[x3070] if rotating left R1 *n* bits produces the bit vector in R0, and store -1 if it is not possible to produce the bit vector of R0 by rotating R1.

Note: The Comments section of the table is strictly for your use. It will not be considered in the grading.

Address	Value	Comments
x3000	0101 010 010 1 00000	
x3001	1001 000 000 1 11111	
x3002	0001 000 000 1 00001	
x3003		
x3004	0000 001 000001010	
x3005	0001 011 001 0 00000	
x3006		
x3007	0001 010 010 1 00001	
x3008	0001 001 001 1 00000	
x3009	0000 100 000000010	
x300A	0001 001 001 0 00001	
x300B	0000 111 111110111	
x300C	0001 001 001 0 00001	
x300D		
x300E		
x300F	0101 010 010 1 00000	
x3010	0001 010 010 1 11111	
x3011		
x3012	1111 0000 00100101	

Name: ___________________________________

This page is left blank intentionally. Feel free to use it for scratch work.
You may tear the page off if you wish.
Nothing on this page will be considered for grading.