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

ECE 306 Fall 2025
Instructor: Yale N. Patt
TAs: Luke Mason, Evan Lai, Madeleine Dreher
Exam 1
October 8, 2025

Name: ______________________SOLUTION____________________________________

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 pts):

Answer the following questions. If you leave a question unanswered, you will receive 1 point out of the 5.

Part A (5 pts): Before the code below executes, memory location x3050 contains the 2's complement integer x. What does memory location x3050 contain after the following code executes?

```
x3000: 0010 011 001001111
x3001: 0000 011 000000011
x3002: 1001 011 011 111111
x3003: 0001 011 011 1 00001
x3004: 0011 011 001001011
x3005: 1111 0000 0010 0101
```

The absolute value of x.

Part B (5 pts): The LC-3 has 3 load instructions (opcodes 0010, 0110, and 1010), each with a different addressing mode for determining the location of the data to be loaded into the register. What benefit does the instruction with opcode 0110 provide over the instruction with opcode 0010.

LDR can load from memory locations beyond the PC-relative range.

Part C (5 pts): The LC-3 has an instruction (opcode 1111) that allows the user program to invoke the operating system to carry out some work that is beyond the knowledge of the user programmer. Most ISAs have that feature. It is usually called **System Call** because the operating system is being called to execute code on behalf of the user program. What is the maximum number of system calls we could provide if we wanted to for the LC-3?

2^8

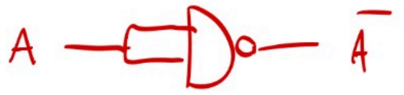
THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: ___________________________________

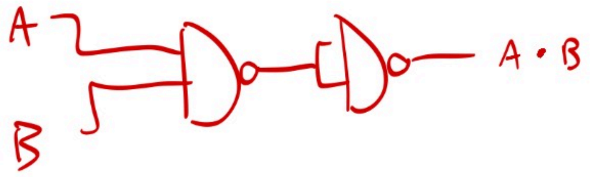
Part D (5 pts): We know that a function f is logically complete if we can implement any truth table, regardless how many input variables it has, if we have a sufficient number of f gates. We know that if we have a sufficient number of AND gates, OR gates, and NOT gates, we can implement any function represented by its truth table. Prove that a NAND gate is by itself logically complete.

NOTE: Your proof must be explicit, that is, each step in your proof must be specific and not rely on the reader to figure out how to do anything. No points if your proof contains statements that rely on the reader to know how to do something.

NOT



AND



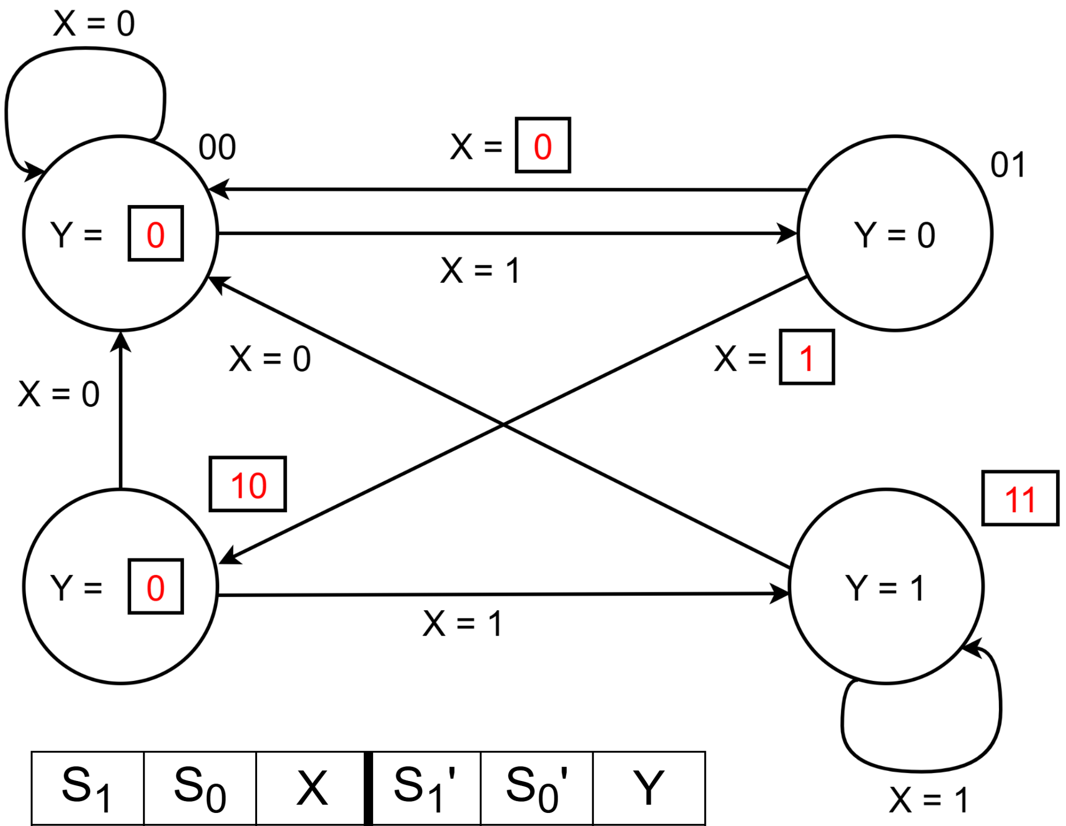
Since we can make AND and NOT gates using only NAND gates, DeMorgan's Law proves that we can also make an OR gate with only NAND gates. Thus, NAND is logically complete.

Name: ______________________________________

Question 2 (15 pts):

You have been given an incomplete Moore state machine. Remember, the output(s) of a Moore machine are determined by only the current state. The state numbers are encoded as 2-bit values represented by S_1S_0. For instance, S_1=0 and S_0=1 correspond to the top right state, state 01.

Your job: Complete the partial state machine and truth table below by filling the blanks.

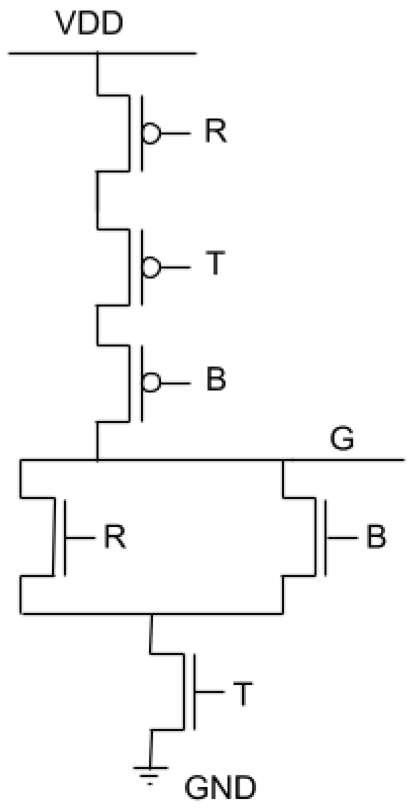


S_1	S_0	X	S_1'	S_0'	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Name: ______________________________

Question 3 (15 pts):

Aggie has attempted to construct a CMOS transistor circuit. His goal was to determine if the garden sprinkler should turn on (G = 1). If it rained today (R = 1) or the water bill was high last month (B = 1), the sprinkler should **not** turn on. Otherwise, it should turn on if it's 9pm (T = 1). Unfortunately, Aggie doesn't know how to design a circuit and created what you see below:



Part A (3 pts): Write the **intended** logic equation for the output G in terms of R, B, and T

G = R'B'T (Or an equivalent equation)

Part B (6 pts): Write down two specific reasons the design does not work. Explain briefly.

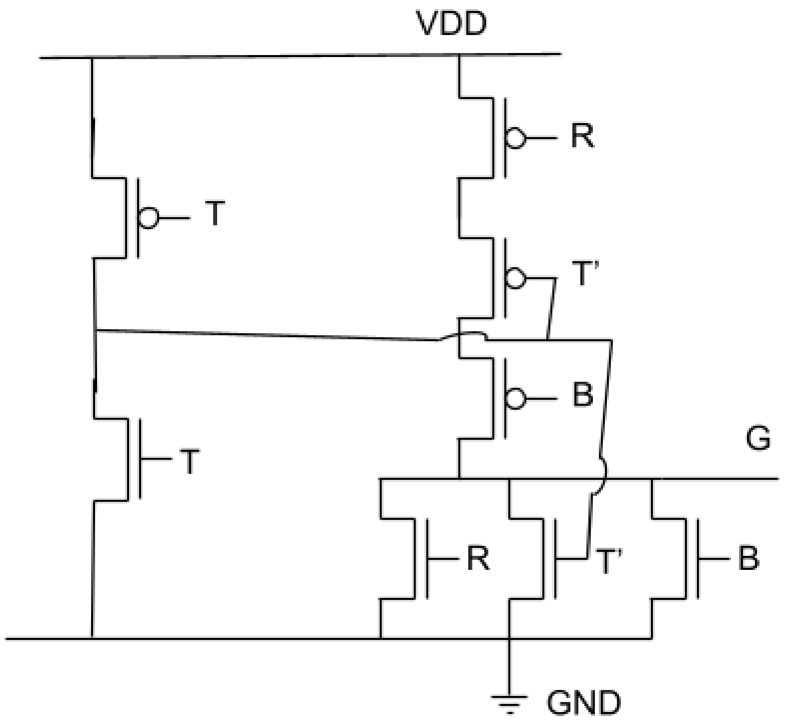
Reason 1: **We want to use T' for the gate of the transistors.**
Reason 2: **The T transistor on NMOS side should be in parallel with R and B transistors.**

THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: ___________________________________

Part C (6 pts): Help Aggie and sketch the correct transistor level circuit:
This circuit inverts T for use as the original T gate inputs. And, we fix the parallel issue on NMOS side.

It's possible to do this circuit differently, but this is simplest.



Name: ___________________________________

Problem 4 (25 points):

Aggie has returned to you for help debugging a new program. The program is trying to perform an integer divide, with the value at x300A being the dividend, the value at x300B being the divisor, and R2 being the quotient. In other words, $R2 = \frac{M[x300A]}{M[x300B]}$.
Note that in an integer divide, the remainder is thrown away (i.e 10/3 = 3).

A snapshot of the PC and some memory locations before the program is executed is shown below. **There is exactly one bug.** The Comments column will not be graded.

PC	x3000

Address	**Contents**	**Comments**
x3000	0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1	R0 <- M[x300A] = x3025
x3001	0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1	R1 <- M[x300B] = x1000
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 <- R2 & 0 = 0
x3003	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	
x3004	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	R1 <- (-R1) = xF000
x3005	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1	R0 <- R0 + R1
x3006	0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1	BRn #3 (BUG!)
x3007	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	Increment R2
x3008	0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0	Branch back to x3005
x3009	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	TRAP x25 or HALT
x300A	0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1	Contains dividend x3025
x300B	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	Contains divisor x1000
x300C …	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …	The rest of the locations are x0000

Part A (5 points): After 1 instruction is executed, the PC is x3001 and R0 has been loaded with M[x300A]. What is the PC after 10 instructions are executed?

PC: x3006

THE PROBLEM CONTINUES ON THE NEXT PAGE

Name: ___________________________________

Part B (10 points): As stated above, there is exactly one bug in the program. Describe the bug (in 15 words or fewer).

The branch offset at x3006 goes past the TRAP x25

Part C (10 points): Despite the bug, the program shown still completes execution correctly. Explain why.

There are two parts to this: the program completes execution (i.e it executes a HALT instruction), and when it does the value in R2 is the correct quotient.

In this case, as you are looping R0 goes from x3025 to x2025 to x1025 to x0025 to xF025, which is negative, causing the program to take the branch at x3006, which ends up executing the instructions at x300A onwards.

x300A is a store instruction, and it stores R0 (which has xF025!) into x3030.
x300B is an add instruction, and it adds R0 to itself. This doesn't affect anything.
x300C and onwards are branches that never branch, as none of the NZP bits in the instruction are set.

Thus the program will execute x300C, x300D, etc… until it reaches x3030. It will execute xF025, which is TRAP x25, which is a HALT, meaning it halts.

R2 never gets touched, meaning it still contains the correct quotient of 3.

Name: ______________________________

Question 5 (25 pts):

Write a program to implement f(x) = 7 * x. The location x3050 contains x, and you will store the result, (7 * x), into location x3051. To accomplish this, you are given a list of allowed instructions. **Not every instruction will be used, and each instruction can only be used once.**

Part A (20 pts): Fill in the program on the right, using instructions from the instruction list on the left. PC = x3000 to start.

*Hint: 7 * x can be accomplished by calculating 1 * x + 2 * x + 4 * x.*

Allowed Instruction List
0010 000 001001110
1111 0000 00100101
0001 001 000 0 00 000
0001 001 001 0 00 000
0001 001 001 0 00 001
0001 010 010 1 11111
0000 100 111111100
0000 101 111111100
0011 001 001001000
0101 001 001 1 00000
0101 010 010 1 00000

Program	
x3000	0101 010 010 1 00000
x3001	0010 000 001001110
x3002	0001 010 010 1 00011
x3003	0101 001 001 1 00000
x3004	0001 001 001 0 00 000
x3005	0001 000 000 0 00 000
x3006	0001 010 010 1 11111
x3007	0000 101 111111100
x3008	0011 001 001001000
x3009	1111 0000 00100101

Part B (5 pts): Assume any state in the LC-3 FSM that accesses memory takes 5 clock cycles. After 30 clock cycles of this program's execution, what are the contents of the PC? (Refer to the LC-3 FSM at the end of this exam)

x3003

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.