ECE190 Exam 1, Fall 2008

Thursday 25 September

Name:

- **Be sure that your exam booklet has 9 pages.**
- **The exam is <u>meant</u> to be taken apart!**
- **Write your name at the top of each page.**
- **This is a closed book exam.**
- **You may not use a calculator.**
- **You are allowed one 8.5×11" sheet of handwritten notes.**
- **Absolutely no interaction between students is allowed.**
- **Challenging problems are marked with ***.**
- **Show all of your work.**
- **Don't panic, and good luck!**

"...there would be no more wars, the nations were so economically interdependent."
—S. Foster Damon, remembrance ca. 1912

Problem 1	20 points	_____
Problem 2	20 points	_____
Problem 3	10 points	_____
Problem 4	25 points	_____
Problem 5	25 points	_____
Total	100 points	_____

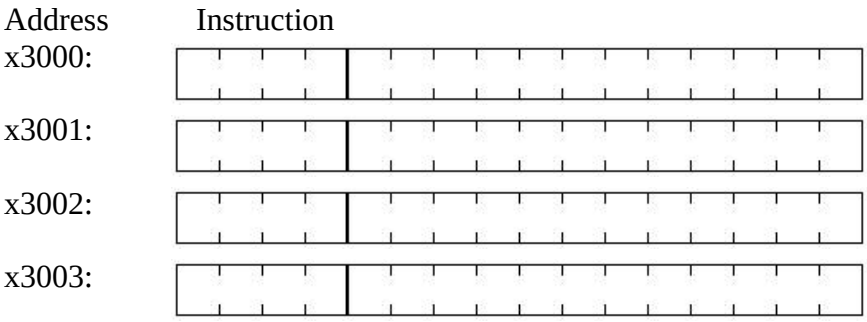
Problem 1 (20 points): Short Answers

Please answer concisely. If you find yourself writing more than a few words or a simple drawing, your answer is probably wrong.

Part A (5 points): Using an 8-bit 2's complement data type, what is the smallest positive number that generates an overflow when added to itself?

Part B (5 points): How many positive integers are represented in a 13-bit 2's complement data type?

Part C (5 points): Using one or more LC-3 instructions, implement a branch if positive (BRp) to an address outside the range of the branch instruction. Assume that the address to which you want to branch is stored in R5. Write your instruction(s) in binary. *Note: you may not need all the lines provided below.*



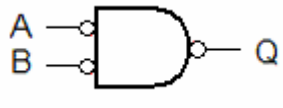
Part D (5 points): The current LC-3 memory contains a total of 1 Megabit (2^{20} bits). If we were to make the memory byte-addressable (a byte is 8 bits) without changing its size, and without changing the size of the instructions (16 bits), how many bits would be necessary in the following registers?

MAR [ ] IR [ ]

Problem 2 (20 points): Logic Circuits

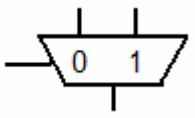
Part A (3 points): Using exactly one logic gate whose inputs are not inverted, draw a logically equivalent circuit to the one pictured below.

Part B (3 points): Using exactly one logic gate whose inputs are not inverted, draw a logically equivalent circuit to the one pictured below.

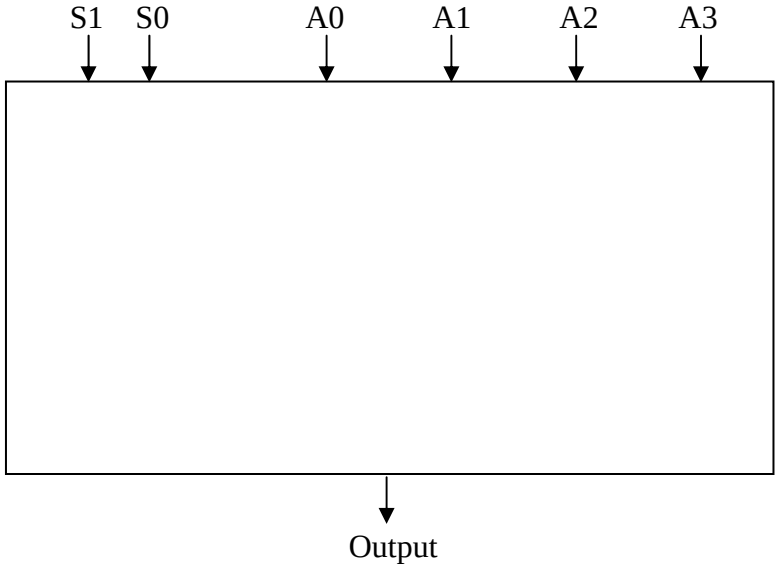


Part C (6 points): Draw the necessary connections to implement a half-adder (an adder without a carry input) using the 2-to-1 decoder and the two NOR gates pictured below.

Part D (8 points): Using three 2-to-1 multiplexers (shown to the left below), draw a circuit which performs the function of a 4-to-1 multiplexer with inputs: A0, A1, A2, A3, and select bits: S1, S0.



2-to-1 Mux



Problem 3 (10 points): Memory

In a typical memory, setting the write enable input (WE) to 1 causes all bits of the selected memory location to accept new values. For a certain ECE445 (senior design) project, we need a 4-address, 3-bit addressable memory that allows us to write to each bit at a location separately. To do so, we hired a computer engineer from the University of Michigan.

The design is done by modifying the 4-address, 3-bit, memory that we discussed in class. Except for the WE input, all inputs and outputs remain the same. More specifically, ADDR is a 2-bit input for address selection, Din is a 3-bit input for providing bits when writing to a memory location.

As for the WE input, the modified memory structure no longer has a single WE input to all bits. Rather, when a location is selected, 3 WEout signals will independently determine if each bit in that location will be written. This modification has already been done by an Illinois student.

The Michigan engineer proposes to construct a combinational logic that provides a nice external interface for using the memory. The logic translates a 4-bit WE external input signal into the internal 3-bit WEout signals as follows:

WE is 4 bits wide and behaves as follows:

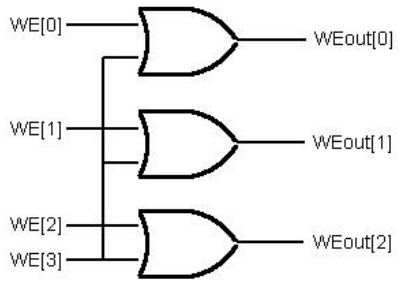
WE [3:0] = 0001	=>	write to bit 0 only
WE [3:0] = 0010	=>	write to bit 1 only
WE [3:0] = 0100	=>	write to bit 2 only
WE [3:0] = 1000	=>	write to all the bits
WE [3:0] = 0000	=>	do not write to any of the bits
WE [3:0] = other	=>	treat as a read cycle

Part A (6 points): For a first round of testing, we executed the six cycles shown on the left below. Assuming correct implementation and assuming that all the bits in the memory are initialized to 1, fill in the table on the right to reflect the final state of the memory (after the six cycles).

Cycle #	ADDR[1:0]	WE[3:0]	Din[2:0]
1	00	0100	000
2	10	1000	010
3	10	0000	110
4	01	0010	001
5	11	0001	011
6	00	1000	100

address	bit 2	bit 1	bit 0
00			
01			
10			
11			

Part B (4 points): The memory designed by the Michigan alumnus passed the first test. However, the next round of tests revealed some undesired behavior. Examining his design, we found the following logic for generating the signals that specify whether or not each individual bit is written (WEout[2:0]).



In 25 words or less, explain the undesired behavior.

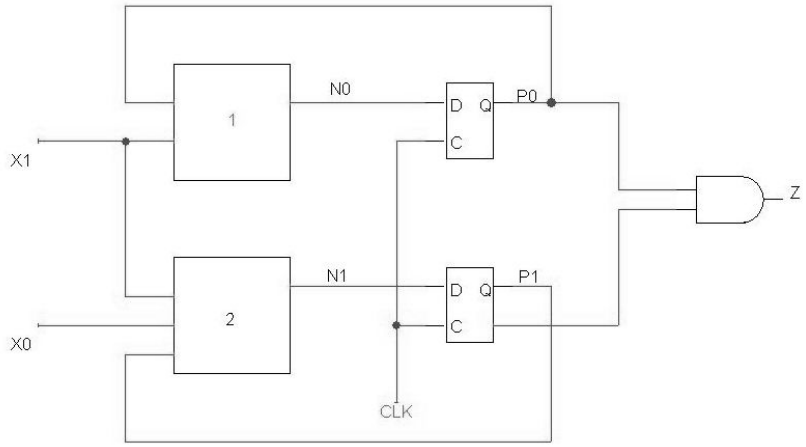
Problem 4 (25 points): Finite State Machines

A certain finite state machine implements a pattern recognition system that recognizes specific sequences of inputs. Each input represents a letter, such as A or B. An input sequence can be written as a sequence of letters such as "ABAAABBBABA…". For this finite state machine, the output Z is 1 when an input sequence has an even number of A's and an odd number of B's. Otherwise, the output Z is 0. The states needed are named as follows:

S_{EE}	even number of A's and even number of B's
S_{EO}	even number of A's and odd number of B's
S_{OO}	odd number of A's and odd number of B's
S_{OE}	odd number of A's and even number of B's

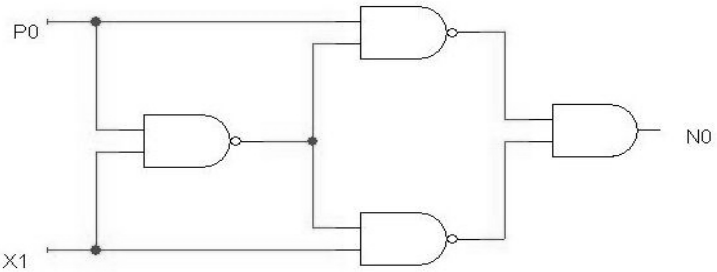
Part A (8 points): Draw a high-level finite state machine transition diagram for the system. States and state names have been drawn for you. Add appropriate output values and all transition arcs. Label arcs as either A or B.

The circuit below shows an implementation of the finite state machine using two flip-flops and state representation (bits P_1P_0) given by S_{EE} = 00, S_{EO} = 01, S_{OO} = 10, and S_{OE} = 11. The blocks labeled 1 and 2 use the current state P_1P_0 and the inputs X_1X_0 to calculate the next state, N_1N_0. The letter A is represented as X_1X_0=00, and the letter B is represented as X_1X_0=01.



Problem 4, continued:

Part B (4 points): Based on the circuit below for **Block 1**, fill in the table for **N0**.



P0	X1	N0
0	0	
0	1	
1	0	
1	1	

Part C (8 points): Based on your answer to **Part B** and on the truth table for **Block 2** (shown to the left below) fill in the next state table on the right for the FSM implementation. Note that input X_1X_0=11 is not included in the table.

P1	X1	X0	N1
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

P1	P0	X1	X0	N1	N0
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	1	0	0		
0	1	0	1		
0	1	1	0		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	1	0	0		
1	1	0	1		
1	1	1	0		

***Part D** (5 points): The input bits X_1X_0=10 represent the letter C. Explain in one sentence how you can use the next state table from **Part C** to verify that the letter C is handled correctly by the FSM implementation.

Problem 5 (25 points): The von Neumann Model

An LC-3 is about to perform an instruction **FETCH**. The contents of the register file, PC, MAR, MDR, IR, and part of memory are shown to the right.

You may wish to consult the last page of this exam, which gives the LC-3 instruction set encoding and RTL.

R0	xA5A5
R1	x1000
R2	x1298
R3	x1981
R4	x4345
R5	x0BEE
R6	x3A0F
R7	xF025

PC	x3A0A
MAR	x39A3
MDR	x0E66
IR	x0E66

...	
x3A06	x6CBC
x3A07	x1FF2
x3A08	x0000
x3A09	x0589
x3A0A	x2004
x3A0B	x1221
x3A0C	x1025
x3A0D	x09FD
x3A0E	xF025
x3A0F	xFFF0
x3A10	x0000
x3A11	x2005
x3A12	x0589
x3A13	x6CBC
...	

Part A (8 points): On the lines below, write RTL for the **next four instructions** to be executed by the LC-3. For PC-relative addressing modes, write addresses relative to PC rather than calculating the actual address to be used, e.g., write "PC+x12" rather than adding the two values for that instruction.

Address	**RTL**

Part B (6 points): Using hexadecimal notation, write the contents of the registers R0 through R7, PC, MAR, MDR and IR **after the LC-3 processes the first three instructions**.

R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	

PC	
MAR	
MDR	
IR	

Part C (4 points): Using hexadecimal notation, write the contents of the PC, MAR and MDR after the LC-3 processes one additional instruction after **Part B**, *i.e.*, **four instructions in all**.

PC	
IR	

MAR	
MDR	

Part D (7 points): The instruction xF025 halts LC-3 execution (*i.e.*, no further instructions are processed after xF025). What are the contents of R0 and R1 **when the LC-3 halts**?

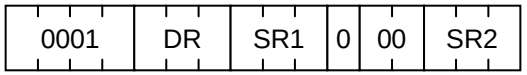
R0	

R1	

Use this page for scratchwork.

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

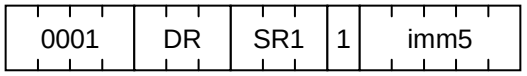
ADD



ADD DR, SR1, SR2

DR ← SR1 + SR2, Setcc

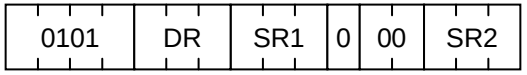
ADD



ADD DR, SR1, *imm5*

DR ← SR1 + SEXT(imm5), Setcc

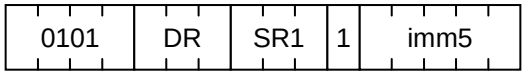
AND



AND DR, SR1, SR2

DR ← SR1 AND SR2, Setcc

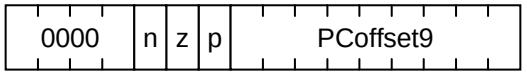
AND



AND DR, SR1, *imm5*

DR ← SR1 AND SEXT(imm5), Setcc

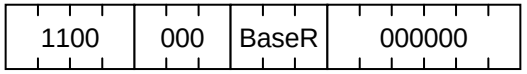
BR



BR{nzp} *PCoffset9*

((n AND N) OR (z AND Z) OR (p AND P)):
PC ← PC + SEXT(PCoffset9)

JMP



JMP BaseR

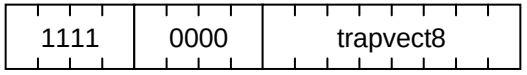
PC ← BaseR

JSR

JSR *PCoffset11*

R7 ← PC, PC ← PC + SEXT(PCoffset11)

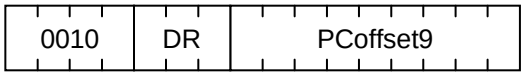
TRAP



TRAP *trapvect8*

R7 ← PC, PC ← M[ZEXT(trapvect8)]

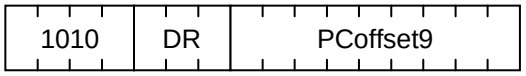
LD



LD DR, *PCoffset9*

DR ← M[PC + SEXT(PCoffset9)], Setcc

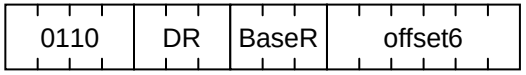
LDI



LDI DR, *PCoffset9*

DR ← M[M[PC + SEXT(PCoffset9)]], Setcc

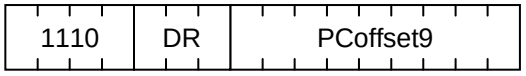
LDR



LDR DR, BaseR, *offset6*

DR ← M[BaseR + SEXT(offset6)], Setcc

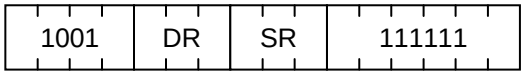
LEA



LEA DR, *PCoffset9*

DR ← PC + SEXT(PCoffset9), Setcc

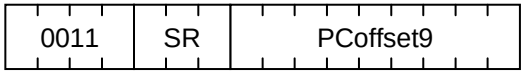
NOT



NOT DR, SR

DR ← NOT SR, Setcc

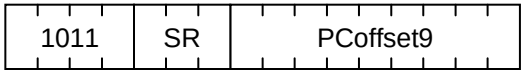
ST



ST SR, *PCoffset9*

M[PC + SEXT(PCoffset9)] ← SR

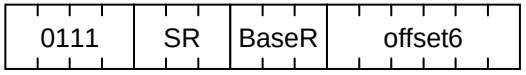
STI



STI SR, *PCoffset9*

M[M[PC + SEXT(PCoffset9)]] ← SR

STR



STR SR, BaseR, *offset6*

M[BaseR + SEXT(offset6)] ← SR