This is NOT the complete final exam from the Fall 2007 semesters. Three questions covering topics that were not covered in the Spring 2008 semester have been removed! Also, vertical spacing has been removed to save trees.

ECE 109 Sections 602 to 605
Final exam Fall 2007
13 December, 2007

Problem 1 (6 points) Memories

a) How many **bits** are in a memory with 2K words and a word size of 32?

b) How many **bits** are required to address a memory with 64K words?

c) How many 16-bit words can be stored in a 2 KB memory?

Problem 2 (2 points) Overflow
Add the following pairs of six-bit two's complement numbers **and indicate which additions result in an overflow**.

110111 + 010010	010101 + 010101

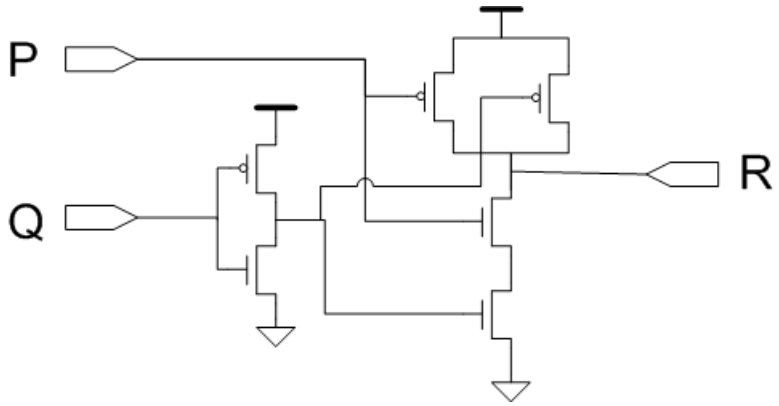
Problem 3 (2 points) Twos complement representation
Translate the following two decimal numbers into five-bit two's complement numbers. Show your work in the space below!

12	-11

Problem 4 (4 points) Fixed point numbers
Express 21.75 as a fixed point binary number. (Remember, 11.1 binary represents 3.5 decimal.)

Problem 5 (4 points) Logical foundations
In Chapter 2, we learned about deMorgan's law and twos-complement representation. How are these fundamental concepts used to perform the bitwise OR and arithmetic subtract operations in the LC/3?

Problem 6 (2 points) MOS to truth
Fill in the truth table on the right to represent the MOS circuit that is on the left. (This should be familiar.)



P	Q	R
0	0	
0	1	
1	0	
1	1	

Problem 7 (4 points) Truth to gates
Draw a circuit, at the gate level, that will implement the truth table, where Z in the single output variable, shown on the right below.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Problem 8 (4 points) Gates to MOS
Draw the MOS circuit that implements a 2-input OR gate.

Problem 10 (12 points) Vocabulary
Compare and contrast the following related pairs of terms. You may cross out one to indicate you don't want it graded. Otherwise I'll grade the first six you answer.

Assembly *versus* machine language
Combinational logic *versus* sequential logic
Conditional *versus* iterative execution

Interrupt-driven I/O *versus* pooling
Interrupt service routine *versus* system call
MAR *versus* MDR
Sign-extension *versus* sign-magnitude

Problem 10 (6 points) Assembling
Using the symbol table shown below

CASH	x4023
LYNN	x4057
NELSON	x40A3
PARTON	x4123

write the appropriate 16-bit LC-3 machine language word, in binary or hex, for each assembly language statement shown in the left column. Assume that the instruction is located at address x3400 in all cases. If the assembly language statement is illegal, state the reason why this is so.

AND R0,R5,#3	
AND R7,NELSON	
BRnzp LYNN	
BRp PARTON	
JSR R3	
JSR PARTON	
JSRR R5	
LEA R2,CASH	
RET	
SUB R3,R3,R3	
STR R7,R3	
STR R3,R6,#-3	
TRAP x15	

Problem 11 (6 points) Reverse engineering

The binary program shown in the left column below is loaded into memory at location x3000. In the right column, write the LC/3 assembly instructions or appropriate psuedo-ops corresponding to this program. Be sure to include appropriate labels and `.ORIG` and `.END` statements.

Binary	Assembly
0101100000100000	
0001100100100001	
0010011000000100	
1000000011000000	
0101101000111111	
0000011111111011	
1111000000100101	
0101000000000000	

Problem 13 (12 points) Being JSR'ed around

The following program ***really*** has been successfully assembled and simulated on the LC/3 software.

```
        .ORIG   x3000           ; instruction location
START   AND     R1,R1,#0        ; x3000
        LEA     R5,START        ; x3001
        LDI     R3,ADD1         ; x3002
        JSRR    R3              ; x3003
        ADD     R1,R1,x9        ; x3004
        JSR     LABZ            ; x3005
LABX    ADD     R1,R1,x3        ; x3006
LABZ    AND     R3,R1,x4        ; x3007
        BRp     STOP            ; x3008
        RET                     ; x3009
STOP    STR     R1,R5,#12       ; x300A
        HALT                    ; x300B
ADD1    .FILL   ADD2            ; x300C
ADD2    .FILL   LABX            ; x300D
        .END
```

The last two `.FILL`'s really are legal LC/3 pseudo-ops. If it makes you more comfortable, you can replace them with the following more obscure, though equivalent, pseudo-ops.

```
ADD1    .FILL   x300D
ADD2    .FILL   x3006
```

Your jobs is to describe what the LC/3 will do while executing this code until either (1) you have filled the following table or (2) the LC/3 executes the `HALT` trap. Put the address of each instruction in the left column and its action in the right column. I've filled in one for you.

x3000	Sets R1 to 0
x3001	
	and many deleted rows follow

Problem 13 (16 points) LC/3 programming
In this long question of many parts, write little (many only two or three instructions long) LC/3 programs to solve the following small problems stated in a C-like syntax. Answers that are unnecessary long or complicated may not receive full credit.
Note: This C-like syntax was covered in class last semester.

```
R3 = R5 + 18 ;
```

```
R5 = R7 - R3 ;
```

```
R4 = R3 ;
```

```
while (R1<93) {
     R1 = 3*R1 ;
}
```

```
if (R4 > 0 && R3>0) {
     R5 = 15 ;
}
```

```
if (R5 == 'n') {
    R5 = 'N' ;
}
```

Problem 14 (2 point) POP and PUSH
Draw the stack the results from the following sequence of pop and push operations and write the value retrieved on the two `POP` operations.

```
PUSH 15 ;
PUSH 109 ;
POP ;
PUSH 209 ;
PUSH 15 ;
POP ;
```

Problem 15 (4 points) Device registers
Using the keyboard status and data registers (`KBSR` and `KBDR`) and none of the LC/3 trap routines, write a silly LC/3 trap routine that reads one character from the keyboard and returns in `R0`, 1 if the character was 'y' or 'Y', and 0, otherwise. Except for `R0`, your routine should restore all registers that it modifies.
P.S. Your code would look the same if it was a subroutine rather than a trap routine.

Problem 16 (4 points) LC/3 subroutine
Write a LC/3 subroutine that receives in register R0 an LC/3 address. Your subroutine should then store zeros in the 15 LC/3 memory locations starting at the address stored in R0. To get full credit for this problem, you must use a loop.

Problem 17 (2 points) Invoking the subroutine
Show how to call the LC/3 subroutine you wrote in Problem 16 to set the 15 memory locations starting with `x5555` to 0.