ECE 198JL Final Exam
Spring 2013

May 9th, 2013

Name: _____ NetID: _____

Discussion Section:

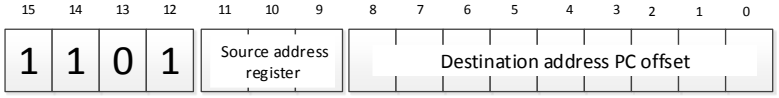
10:00 AM	[ ] JD1	
11:00 AM	[ ] JD2	
2:00 PM	[ ] JD3	
4:00 PM	[ ] JD4	

- **Be sure your exam booklet has 10 pages.**
- **Be sure to write your name and lab section on the first page.**
- **Do not tear the exam booklet apart.**
- **We have provided LC-3 instructions set and other reference materials on separate pages.**
- **Use backs of pages for scratch work if needed.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed two handwritten 8.5 x 11" sheets of notes.**
- **Absolutely no interaction between students is allowed.**
- **Be sure to clearly indicate any assumptions that you make.**
- **The questions are not weighted equally. Budget your time accordingly.**
- **Don't panic, and good luck!**

Problem 1	18 points:	_____
Problem 2	12 points:	_____
Problem 3	10 points:	_____
Problem 4	10 points:	_____
Problem 5	8 points:	_____
Problem 6	10 points:	_____
Problem 7	10 points:	_____
Problem 8	22 points:	_____
Total	100 points:	_____

Problem 1 (18 pts): LC-3 microinstructions

You are adding a new instruction, called **CP**, to the LC-3 instruction set. This instruction copies a value from one memory location whose address is stored in the register specified by IR[11:9] to another memory location whose address is specified as a PC-relative offset by IR[8:0]. The binary encoding of this new instruction is as follows:



a) In RTL form, give a sequence of (at most 4) *microinstructions* that implement the execute phase of the **CP** instruction. Make sure your implementation does not modify any values in the general-purpose register file.

_____

_____

_____

_____

b) The execute phase of **CP** instruction starts at control ROM address _____ (answer in binary with the correct number of bits).

c) Determine control ROM microinstructions that implement the RTL statements from part (a). Complete the table below by filling in 0, 1, or x as appropriate. Use don't cares wherever possible. Specify ROM addresses in decimal. When you need *additional* states, state numbers **55**, **56**, **57**, and **58** are available for your use.

ROM address	IRD	COND(3)	J(6)	LD.BEN LD.MAR LD.MDR LD.IR LD.PC LD.REG LC.CC	GateMARMUX GateMDR GateALU GatePC	MARMUX PCMUX(2) ADDR1MUX ADDR2MUX(2) DRMUX(2) SR1MUX(2) ALUK(2)	MIO.EN R.W
				Do not fill in this space. **Only fill in control word bits for the first two microinstructions.**			

d) We could have accomplished the task of copying a value from one memory location to another memory location by simply writing a short program using existing instructions. Write such a program in LC-3 assembly language assuming that the memory source address is stored in R6 and the memory destination address is labeled as DEST.

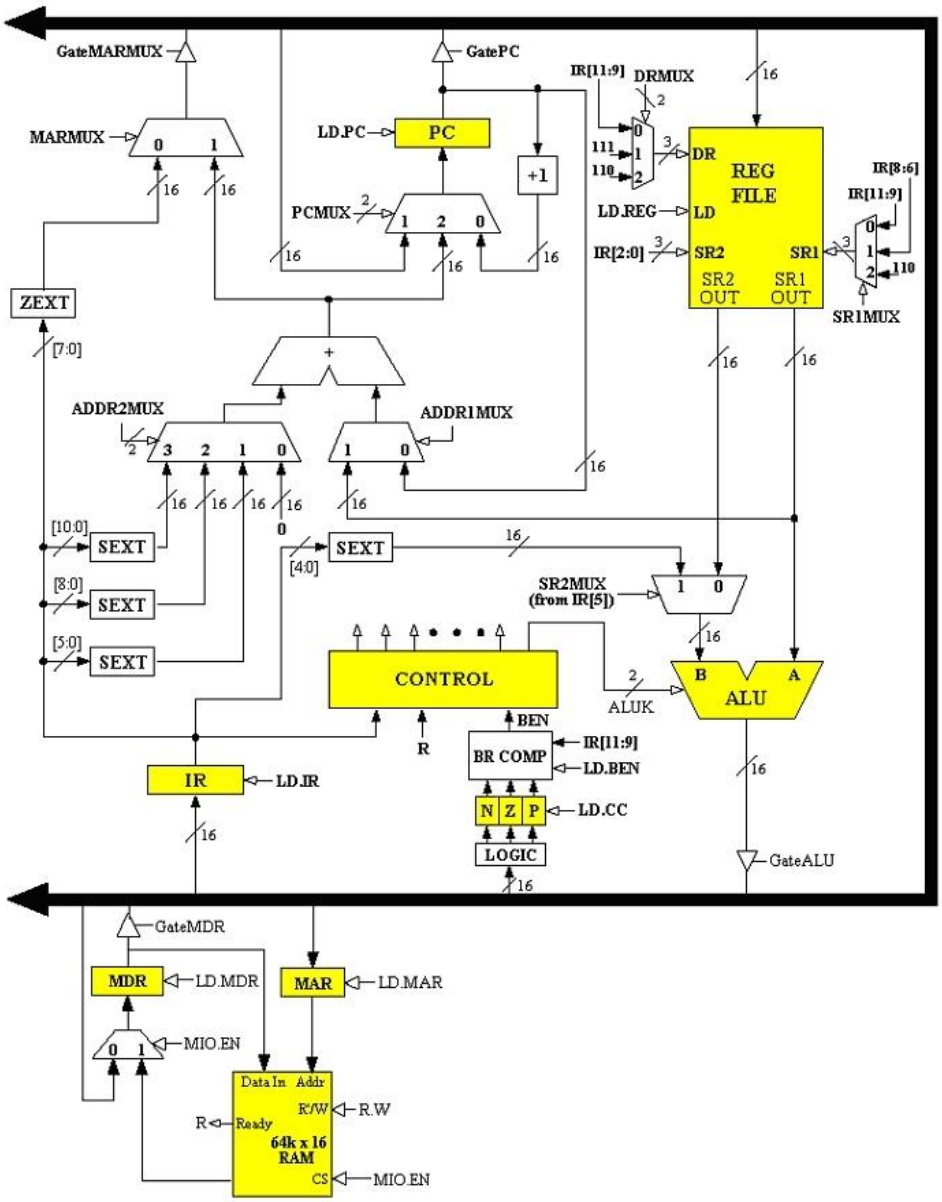
_____

_____

_____

Problem 2 (12 pts): LC-3 datapath and microinstructions

a) On the LC-3 datapath shown on the right, highlight two different paths that can be used to implement the following RTL statement: `PC ← SR1` where `SR1` is specified by `IR[8:6]`. For each path, write down GATE and MUX control signals below. Use don't cares when possible.



Path 1

GateMARMUX: _____
GateMDR: _____
GateALU: _____
GatePC : _____
MARMUX: _____
PCMUX(2): _____
ADDR1MUX: _____
ADDR2MUX(2): _____
DRMUX(2): _____
SR1MUX(2): _____
ALUK(2): _____

Path 2

GateMARMUX: _____
GateMDR: _____
GateALU: _____
GatePC : _____
MARMUX: _____
PCMUX(2): _____
ADDR1MUX: _____
ADDR2MUX(2): _____
DRMUX(2): _____
SR1MUX(2): _____
ALUK(2): _____

b) Give the RTL for the following control word, based on the LC-3 architecture. Do not use terms like 'SR' or 'DR'; use the specific IR bits by name (e.g. IR[11:9] instead of DR).

$0CA2A22_{16}$

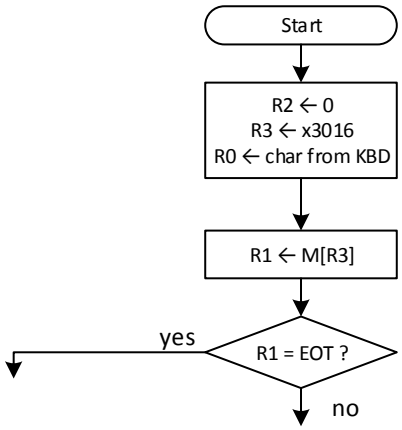
Problem 3 (10 pts): LC-3 assembly program analysis

The following program counts occurrence of a character in a *file*. Character to count is input from the keyboard. The count (which must be < 9) is shown on the display. *File* starts in memory from address x3016 and is terminated by the EOT ASCII control character (EOT's ASCII value is x4).

a) Fill in missing instructions

```
          .ORIG   x3000
          AND     R2, R2, #0
          LD      R3, PTR
          IN
NEXT      LDR     R1, R3, #0
          _____________________________
          BRz     OUTPUT
          NOT     R1, R1
          ADD     R1, R1, #1
          ADD     R4, R1, R0
          _____________________________
          ADD     R2, R2, #1
SKIP      ADD     R3, R3, #1
          BRnzp   NEXT
OUTPUT    LD      R0, ASCII
          _____________________________
          OUT
          HALT
ASCII     .FILL   x0030
PTR       .FILL   x3016
          .END
```

b) Draw flowchart for the above program. Be specific, use standard symbols only (ovals, rhombs, rectangles, etc.) The flowchart is already partially built to give you an example of what's expected.



c) Complete the symbol table for the above program.

Symbol Name	Address

d) Briefly explain why the count must be < 9.

Problem 4 (10 pts): LC-3 assembly language programming

Write a program that computes max of 2 integer numbers. The numbers are stored in memory locations labeled as NUM1 and NUM2. The result should be stored in memory location labeled as RES.

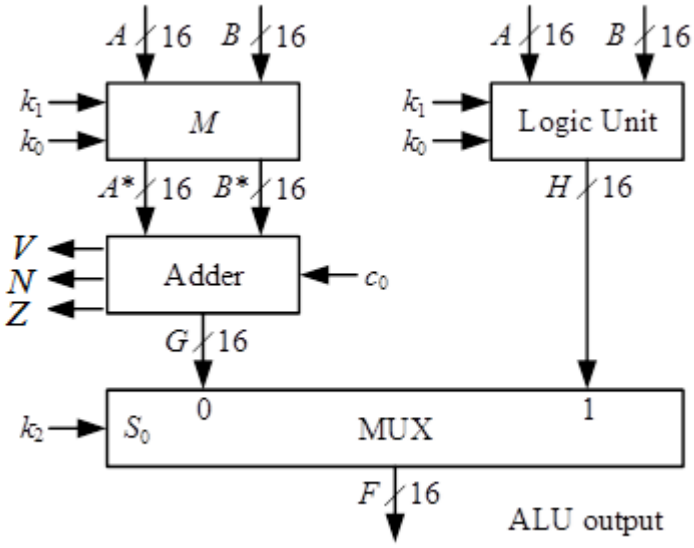
a) Draw flowchart for the solution to the above problem.

b) Write a program in LC-3 assembly language that corresponds to the above flowchart. The program must start at memory address x4000. The two values must be initialized to 10 and -20. The program must terminate and it must be well-documented such that it can be graded by a human.

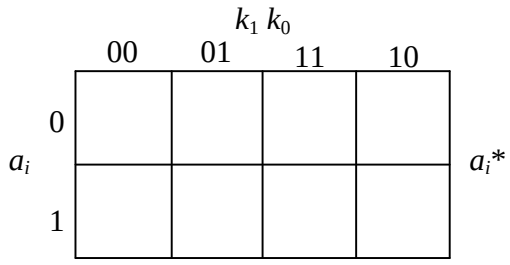
Problem 5 (8 pts): ALU

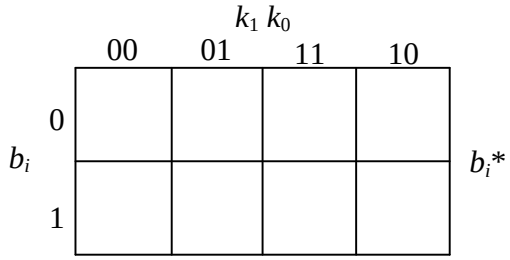
The ALU below has control signals $k_2\,k_1\,k_0$. It takes two 16-bit input words $A = a_{15} \ldots a_0$ and $B = b_{15} \ldots b_0$ and produces a 16-bit output word $F = f_{15} \ldots f_0$. The combinational circuit M produces modified adder inputs $A^* = a_{15}^* \ldots a_0^*$ and $B^* = b_{15}^* \ldots b_0^*$. The Logic Unit produces $H = h_{15} \ldots h_0$.

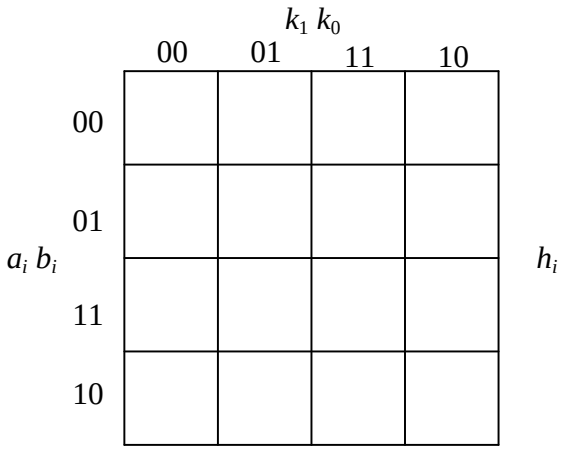
$k_2\,k_1\,k_0$	F
0 0 0	0
0 0 1	A plus B
0 1 0	B plus 1
0 1 1	B minus 1
1 0 0	A
1 0 1	B
1 1 0	not B
1 1 1	A and B



a) Using these Karnaugh maps, define the modified adder inputs a_i^* and b_i^* and the logic unit outputs h_i. Also write the simplest possible Boolean expression for c_0, the carry-in to the adder.







c_0 = _____

b) The adder also produces one-bit status signals V, N, and Z. Write Boolean expressions for these signals as functions of g_i and c_i.

$V = 1$ in case of overflow: V = _____

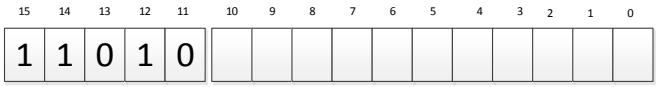
$N = 1$ if the result is negative: N = _____

$Z = 1$ if the result is 0: Z = _____

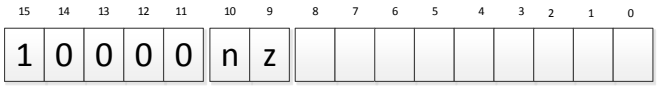
Problem 6 (10 pts): Next-instruction unit design

You are designing a next-generation computer called NextGen198jl. Among other things, this computer has the *next-instruction* unit that decides what the PC's next value should be. For *jumps*, the PC should be loaded with the zero-extended target address specified in the instruction (PC ← ZEXT(IR[11:0])). For *branch* instructions, the PC should be loaded with the updated target address *only* if the corresponding status bit (N for negative or Z for zero) is true (if N & IR[10] or Z & IR[9], then PC ← PC +SEXT(IR[8:0]), otherwise PC ← PC + 1). For *all other* instructions, the PC should just be incremented by 1 (PC ← PC + 1). Formats of JUMP and BRANCH instructions are shown below. Note that the opcodes in NextGen198jl ISA are 5-bit long.

JUMP



BRANCH



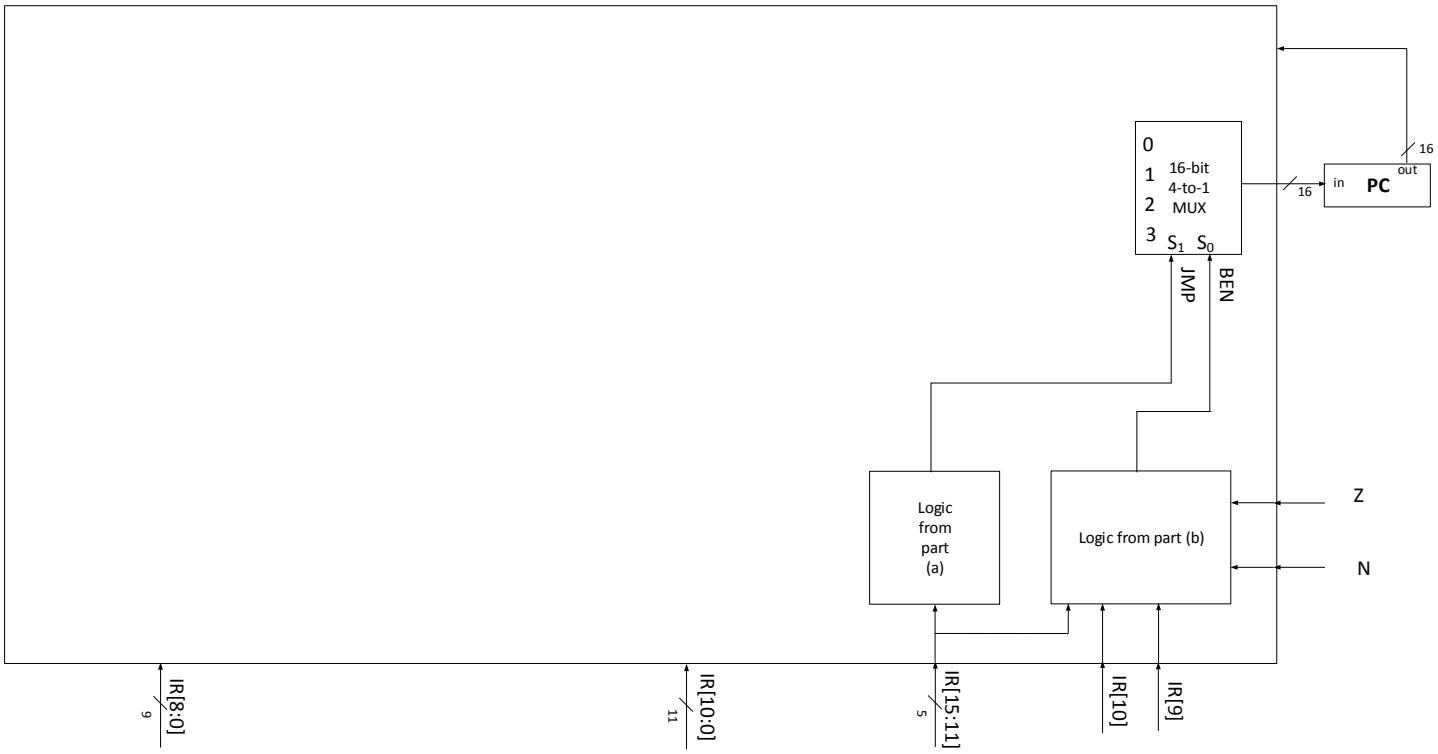
a) Write Boolean expression for JMP as a function of OPCODE. JMP = 1 when IR[15:11]=11010.

JMP = _____

b) Write Boolean expression for BEN. BEN = 1 when IR holds BRANCH instruction and the corresponding status bit (N or Z) is true.

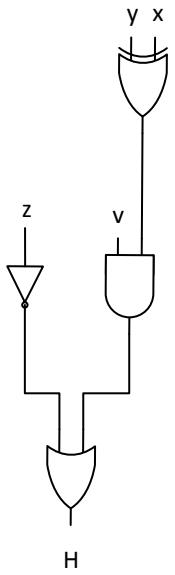
BEN = _____

c) Complete the implementation of the next-instruction unit below. You can only use **one adder** (ADD), **one zero extension unit** (ZEXT), **one sign extension unit** (SEXT), **one increment-by-one unit** (+1), **one MUX** (already drawn), and *as few additional gates as possible*. You can size all the units/gates as needed. Use signals JMP and BEN from parts (a) and (b) to control the MUX. Label all components and signals properly. E.g., IR[15:11] can be used to label wires that carry opcode of the instruction. Excessive complexity will be penalized.



Problem 7 (10 pts): Boolean algebra and combinational circuits

a) From the circuit shown on the right, write Boolean expression for H.



H = _____

b) Express H in **minimal SOP** form.

H = _____

c) Express H in **minimal POS** form.

H = _____

d) Draw two-level **AND-to-OR** network and two-level **NOR-NOR** network for H. Assume that inverted inputs are available.

AND-OR	NOR-NOR

e) Explain advantages of implementing circuits as two-level AND-to-OR or OR-to-AND networks.

f) Explain advantages of implementing circuits as two-level NAND-to-NAND or NOR-to-NOR networks when using CMOS technology.

Problem 8 (22 pts): Sequential circuits and FSM

You are designing a vending machine. The machine sells jawbreaker candy for 25¢. The machine accepts *N* (nickels = 5¢), *D* (dimes = 10¢), and *Q* (quarters = 25¢) coins only. When the sum of the coins inserted in sequence is 25¢ or more, the machine dispenses one jawbreaker by asserting *DJ* output signal and returns to its *initial* state. No change is returned. *DJ* = 0 in all other states. If anything less than 25¢ is inserted and the Coin Return (*CR*) pushbutton is pushed (*CR*=1), the coins deposited so far are returned by asserting *RC* output signal, after which the machine returns back to its *initial* state. *CR* = 0 in all other states.

a) Draw Moore state diagram for this FSM below. Assign states (you will need this for part (b)). Make sure to label each state with name and output and each edge with input. Make sure to label all parts, inputs, outputs, edges, etc. Points will be deducted for missing labels and messy drawing.

b) Based on the above FSM and your state assignments, show the next-state table for your ***initial* state only**. (Remember, only one coin can be inserted at a time, therefore your next-state table should be fairly short.)

c) Implement the above FSM using *negative-edge* triggered D flip-flops. Do not implement the actual next-state and output logic; instead just draw boxes representing the circuits that compute *next-state bits* as functions of current state and external inputs, and *outputs* as functions of current state only. Make sure to label all parts, inputs, and outputs. Points will be deducted for missing labels and messy drawing.