CS 270 Fall 2011 Final

16 Dec 2011, 9:40–11:40 am

Name _____

Please read these instructions completely before proceeding, and sign below. Your exam will not be graded without your signature.

- This is a closed book exam, but you are allowed to bring in one page (one single side) of handwritten notes. It is designed so that the average score is about 80 (67%). Do not be discouraged if you cannot answer all the questions.
- The next page of this exam has the question-by question table of your final score with six extra "Plan of Attack" columns that list the order in which you plan to solve these problems and the time at which you will start and stop each one. When you start the exam, you should spend 10 minutes to do your plan of attack, and fill these columns. Then refer back to it and manage your time.
- The last pages of this exam have the LC-3 opcodes and the datapath of the LC3 (page 142). You are free to tear them out and use as a reference.
- This exam will last two hours, the total is 120, the weight of each problem corresponds to the time you should spend on it.
- You are allowed to use only paper/pen/pencil and your brain—no calculator, laptop, phone, ipod, or any electronic device. Please turn off cellphones, and please refrain from using any listening device (music, etc.)
- Do not turn this page until you are asked to.

I have read the above instructions. I will do the exam honestly and fairly.

Signature _____

Problem 0: Plan of Attack [10 pts]

Quickly read through the exam and make a plan of attack. For each question, think about what skills it's testing for, how comfortable you feel, and rate its difficulty level (not how long it's going to take—some are long and some are short—but how hard it is). Based on this, fill up the PoA columns on in the table below. Don't fill up the last three columns as yet. Problem 4 is long and counts for 55 points [10+15+10+5+5+10]. You may even want to make a separate PoA for it if you like.

Don't write in these columns				Plan of Attack			Revised PoA		
Prob.	Topic	Max	Score	PoA	Start	End	PoA	Start	End
0	PoA	10	10	0	9:40	9:50			
1	Numbers & Data	10							
2	Gates/Comb Ckts	15							
3	LC3 Architecture	10							
0.b	Revised PoA	5							
4	Programming	45							
5	Pot pourri	15							
6	Activation Records	10							
	Total	120							

Problem 1: Numbers and Data [10 pts]

a. This question is about adding two 8-bit sign-magnitude numbers, $A = -50$, and $B = +28$ (not 2's complement, not 1's complement, but sign-magnitude). First, figure out the binary representation of the two numbers and write it down in the table below (in the order that is most convenient to compute $A + B$). Next, compute $A + B$ in the bottom row (if you want to be considered for partial credit you must show all your work). [5 pts]

A or B								
B or A								
A + B								

b. Suppose you are adding two 16-bit floating point numbers, and suppose that they have the same sign bit, and that their exponents are equal. For this special case, does the result have to be normalized [2 pts]? If not, explain why, otherwise, explain how to compute the number and direction by which to shift the answer [3 pts]? [5 pts]

Problem 2: Gates and Combinational Circuits [15 pts]

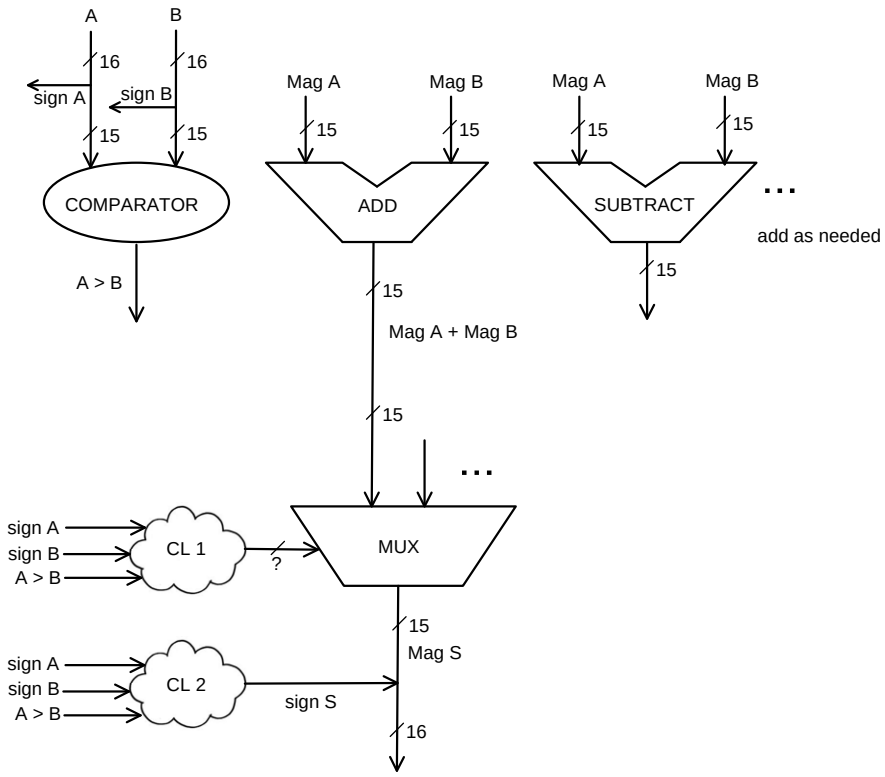
In this problem, we will design a circuit that adds two 16-bit sign-magnitude integers, $A[15:0]$ and $B[15:0]$. We will "peel off" the respective sign bits and call them $SignA$ and $SignB$. The remaining 15 bits we will call, them $MagA$ and $MagB$. We are not going to ask you for the entire design just specific parts. Answer only what is asked for.

Part a. The sign of the result (SignR) is a function of SignA, SignB and $A > B$ the output of a "comparator" circuit. Fill up the truth table of this function [5 pts], and draw a circuit that implements it using only NOR gates [5 pts] (a correct circuit with other gates will get you some partial credit). [10 pts]

SignA	SignB	$A > B$	SignR
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 b. The circuit diagram of this adder, is partially shown on the next page. Complete it. You'll have to add new elements, but use as few as possible. [5 pts]

You may use shorthand—rather than drawing out parts, you may just use names (e.g., we used names to show the inputs to the combinational logic "clouds" and MagA and MagB to show the adder inputs).



Problem 3: LC-3 Architecture [10 pts]

This question is about life as an LC-3 architecture detective. Your spy is snooping on the activities going on in the LC-3 data-path and reports which signals were active in successive cycles. Your job is to deduce what happened. The signals are: LD_PC, GatePC, GateMARMUX, ADDR1MUX, ADDR2MUX, DR, SR1, SR2, LD_REG, LD_IR, LD_MDR, LD_MAR, MEM_EN, and R_W.

Part a. Here is your spy's first report:

Cycle	Signals that are active
1	GatePC, LD_MAR, MEM_EN, R_W
2	LD_MDR
3	LD_IR

What did the LC-3 just do? [2 pts]

Part b. The next report is a few hundred clock cycles later, and is somewhat sketchy, but your spy is sure that just before Cycle 1 was a decode step.

Cycle	Signals that are active
1	GatePC, DR=111, LD_REG
2	Smudged

With just this much information, what can you deduce about the op-code of the instruction in IR. [2 pts]

Part c. Because the last line is smudged, you call your spy who says, "I can tell you that GateMARMUX is 0, SR1 is 010, and also, ADDR1MUX = 0, i.e., it is selecting left input, SR1OUT." You, the master detective thank your spy, turn to the assembled audience and say, "I now know all!" Explain: the complete instruction in IR, as well as *all other signals* that are on in Cycle 2. [8 pts]

Problem 0.b: Revised PoA [5 pts]

This is a 5 minute, strategy break. First, take a quick 1-minute break. Close you eyes, calm down, breathe deeply and relax. Get up and physically stretch and try to ease tension. Make eye contact with your friends and smile. Look at Sanjay and scowl.

Now, revisit your plan. Draw a line through all problems that you have finished, and revise the plan as needed. Budget the remaining time appropriately. Maybe now, you want to make a plan for the parts of Problem 4.

Problem 4: Programming: [55 pts]

This problem deals with LC-3 as well as C. It is broken up into pieces and each piece could be solved independently. However, you should at least *read through the problem sequentially* during your PoA.

Part a. [**10 pts**] Write an LC-3 program that implements the following single line of C, assuming that i and k are ints, have non-negative values and the loop is guaranteed to terminate.

```
for (i=k; i<16; i++){ LOOP_BODY }
```

Don't worry about stacks, activation records, etc. Assume that variables are accessed directly from memory (labels in the LC-3 code), Your code must use *a single register*, R0. *No other registers must be touched, even to save them.* [10 pts]

```
.ORIG 3200
;;  Code to be executed bfore the loop will come here
BEFORE:  ;;  blah ... blah
         ;;  more blah
;;The logic to execute the loop starts from the next line
START:




BODY:
;;  The LOOP_BODY starts here, whatever it is.  It does not touch R0
;; end of LOOP_BODY




;; The code after the loop starts on the next line
EXIT:
```

Part b. [**25 pts**] Now we implement the loop body, which is

```
if (input & pow(2,i)){ COND_BODY }
```

This uses C's bitwise-and operator, and the math library function that computes the power, but your LC-3 code is not allowed to do this. So, we first modify the C loop as follows:

```
mask = /* some expression E1 to initialize mask */ ;
for (i=k; i<16; i++){
    if (input & mask) {
        COND_BODY;
    }
    mask = /* expression E2 to update mask */
}
```

The expression, `E1` may refer to only `k`, and `E2` may refer to only `mask`. Neither should make any function calls. You are to write the two expressions [5 pts], the corresponding LC-3 code to implement the two statements that write to `mask` [10 pts], and also the code that implements the logic of the `if` statement. [10 pts]

First write the two C expressions, E1 and E2. [5 pts]

Now write the LC-3 code to initialize `mask = E1`. Use R1 for mask. You are only allowed one other register, R2. [10 pts]

Finally, write the LC-3 code to evaluate the conditional (loop body) and at the appropriate place, also the code to do `mask = E2`. [10 pts]

```
LOOP BODY:
;; Code to evaluate the condition expression, and based on that, to either do
;; or skip the COND_BODY.  At the right place, insert code to do mask = E2









;;
;;   Here is COND_BODY
;;









;; The code after the if-statement loop starts on the next line









;; This is the end of the loop body
```

Part c. [**5 pts**] Now for `COND_BODY` which is just

```
output = output | pow(2,(i-k)
```

This uses the power function, and you must write LC-3 code for it. So, we modify the C code to introduce another variable `mask2`, in such away that the assignment statement now becomes `output = output | mask2`. Give the C code for initializing `mask` (outside the loop) and updating it inside the loop. [5 pts]

Part d. [**5 pts**] The initializations and update of `mask2` can be written as LC-3 code (similar to Part b) so we won't ask you for it. Write the snippet of LC-3 code that computes `output = output | mask2`. Assume that `mask2` is in R3, and you are allowed to use only one other register, R4. [5 pts]

Part e. [**10 pts**] Consider the following little snippet of C.

```
//  Some code that declares input, output, i, and k as integers
//  It is assured that k is non-negative and "small enough"
//  pow is the power function: pow(x,y) returns the yth power of x
//  For example, pow(3, 2) returns 9
0   for (i=k; i<16; i++){
1:      if (input & pow(2,i))
2:          output = output | pow(2,(i-k);
3:  }
```

If you did parts a-d, you wrote LC-3 code to execute it, but even if you haven't done that yet, you can still do Part e. This code is intended to do something useful, but is incomplete. It needs one more statement, `output = expr` before the loop, where `expr` is a C expression.

Describe in one simple, short sentence what this code is supposed to do, and fix it so that it does. This takes more thinking, but the answer is short. If you want some partial credit, work out an example, say `input` is 12,300 (in hex that's x300C, and `k` is 4. [10 pts]

Problem 5: Pot Pourri & Jeopardy [15 pts]

Part a. We wrote a new service subroutine called `NewService` which provides some very important security service for the LC-3. We stored this service function at location x4000. Remember that because it is s service routine that executes in privileged mode, it is not possible to link it to the user code, and it canot be called with something like `JSR NewService`. Describe how the user will be able to avail of this routine. There are many choices, you have to make a design decision. [5 pts]

Part b. What is the symbol table of the LC-3 program shown on the next page? [5 pts]

```
;
; Program to count occurrences of a char in a file.
; Character to be input from the keyboard.
; Result to be displayed on the monitor.
; Program only works if <= 9 occurrences are found.
;
; Initialization
;
      .ORIG    x3000
      AND R2, R2, #0  ; R2 is counter
      LD       R3, PTR     ; R3 is pointer to chars
      GETC                 ; R0 gets character input
      LDR      R1, R3, #0  ; R1 gets first character
;
; Test character for end of file
;
TEST  ADD      R4, R1, #-4 ; Test for EOT
      BRz      OUTPUT      ; If done, prepare output
; Test character for match, if so increment count.
;
        NOT   R1, R1
        ADD   R1, R1, R0 ; If match, R1 = xFFFF
        NOT   R1, R1     ; If match, R1 = x0000
        BRnp  GETCHAR    ; No match, no increment
        ADD   R2, R2, #1
;
; Get next character from file.
;
GETCHAR ADD   R3, R3, #1 ; Point to next character.
        LDR   R1, R3, #0 ; R1 gets next char to test
        BRnzp TEST
;
; Output the count.
;
OUTPUT  LD    R0, ASCII  ; Load the ASCII template
        ADD   R0, R0, R2 ; Covert binary to ASCII
        OUT              ; ASCII code is displayed.
        HALT             ; Halt machine
;
; Storage for pointer and ASCII template
ASCII .FILL x0030
PTR .FILL x4000
.END
```

Part c. Given a C program with the following signature

`int foo(input, k)`. The program has two int variable `m1` and `m2`, and needs to push additional temporary values on the stack as it executes. Show the activation record during the execution of this function if it is called with the parameters 12,300 (in hex that's x300C) and 5, during the course of execution, it has pushed two values on the stack. [5 pts]