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

EE 306, Fall 2013
Yale Patt, Instructor
Ben Lin, Mochamad Asri, Ameya Chaudhari, Nikhil Garg, Lauren Guckert,
Jack Koenig, Saijel Mokashi, Sruti Nuthalapathi, Sparsh Singhai, Jiajun Wang
Final Exam, December 13, 2013

Name: Solution

Part A:

Problem 1 (10 points): 10

Problem 2 (10 points): 10

Problem 3 (10 points): 10

Problem 4 (10 points): 10

Problem 5 (10 points): 10 **Part A (50 points):** 50

Part B:

Problem 6 (20 points): 20

Problem 7 (20 points): 20

Problem 8 (20 points): 20

Problem 9 (20 points): 20 **Total (130 points):** 130

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.

I will not cheat on this exam.

_____
Signature

GOOD LUCK!
(HAVE A GREAT SEMESTER BREAK)

Name: Solution

Problem 1. (10 points):

Every LC-3 instruction takes 8 cycles to be fetched and decoded, if we assume every memory access takes 5 cycles. The total number of cycles an LC-3 instruction takes to be completely processed, however, depends on what has to be done for that instruction.

Assuming every memory access takes 5 cycles, and assuming the LC-3 processes one instruction at a time, from beginning to end, how many clock cycles does each instruction take? Your job in this problem is to complete the table below. For each instruction, fill in the number of cycles required to process it.

Incidentally, the information in this table will be needed in Problem 7.

Instruction	Number of cycles
ADD	9
AND	9
LD	15
LEA	9
LDI	21
NOT	9
BRnzp	10
TRAP	15

Name: Solution

Problem 2. (10 points):
Part a. (2 points): The following program needs to be assembled and stored in LC-3 Memory. How many LC-3 memory locations are required to store the assembled program?

```
        .ORIG x4000
        AND  R0,R0,#0
        ADD  R1,R0,#0
        ADD  R0,R0,#4
        LD   R2,B
A       LDR  R3,R2,#0
        ADD  R1,R1,R3
        ADD  R2,R2,#1
        ADD  R0,R0,#-1
        BRnp A
        JSR  SHIFTR
        ADD  R1,R4,#0
        JSR  SHIFTR
        ST   R4;C
        TRAP x25
B       .BLKW 1
C       .BLKW 1
        .END
```

Number of memory locations required to store the binary code

16

What is the address of the location labeled C?

x400F

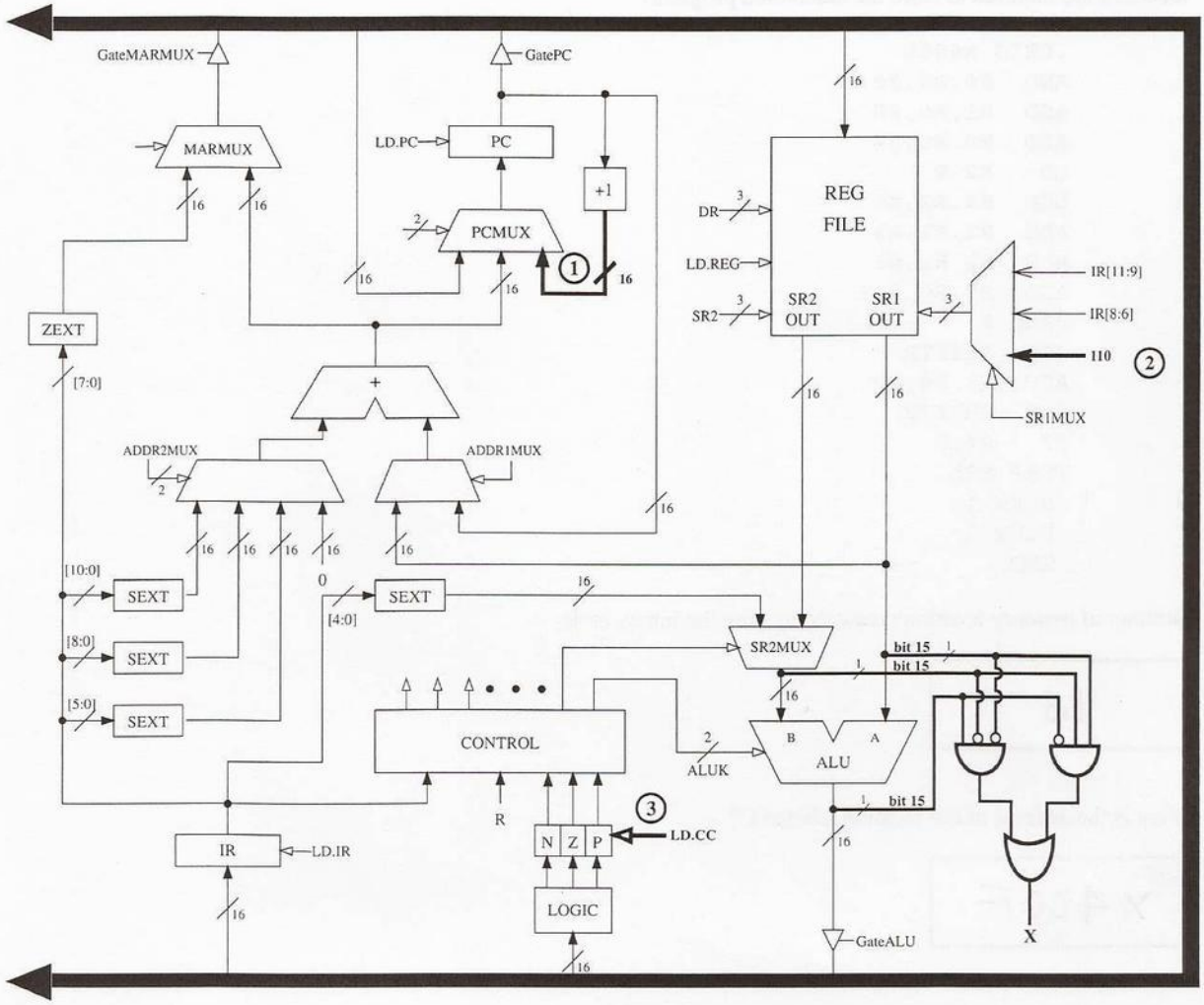
Part b. (8 points):
Before the program of part a) can execute, the location labeled B must be loaded by some external means. You can assume that happens before this program starts executing. The program also contains a subroutine whose starting address is SHIFTR. Recall in class, we talked about subroutine libraries (collections of subroutines) that are linked with your program to provide one complete executable image. You can assume that the subroutine starting at location SHIFTR is available for the above program to use. SHIFTR takes the value in R1, shifts it right one bit, and stores the result in R4.

After the program segment executes, what is in location C? Please answer in fewer than 25 words. Actually, ten should be more than enough.

The average of 4 values from memory

Name: Solution

Problem 3. (10 points):



Part a. (6 points): Note that several of the lines (i.e., signals) in the LC-3 data path have been drawn in boldface. The line is designated by a boldface number 1,2,or 3. Not all instructions use all three lines. That is, some instructions would not function correctly if the line (i.e., signal) were removed.

List the opcodes that utilize line 1 during their processing of an instruction.
ALL
List the opcodes that utilize line 2 during their processing of an instruction.
RTI
List the opcodes that require LD.CC=1 on line 3 during their processing of an instruction.
AND, ADD, NOT, LD, LDR, LDI, (LEA)

Part b. (4 points): Note the logic (in boldface) added to the data path. The output of that logic is labeled X. What does X=1 indicate (in 15 words or fewer, please) if ALUK is ADD?

overflow

Name: Solution

Problem 4. (10 points):
The following program is supposed to print HELP! to the monitor.

```
        .ORIG x3000
        LEA  R0,OUTPUT
        LDR  R1,R0,#0
AGAIN   LDI  R2,DSR
        BRzp AGAIN
        STI  R1, DDR
        ADD  R0,R0,#1
        LDR  R1,R0,#0
        BRz  AGAIN
        TRAP x25
OUTPUT  .STRINGZ "HELP!"
DSR     .FILL xFE04
DDR     .FILL xFE06
        .END
```

However, there is a small bug in the program. What is actually printed on the monitor?

H

What is the bug?

BRz Again goes the wrong way

How would you fix it so the program would work properly?

Should be BRnp or BRp

Name: Solution

Problem 5. (10 points):
It is often useful to find the midpoint between two values. For this problem, assume A and B are both even numbers, and that A is less than B. For example, if A=2 and B=8, the midpoint is 5. The following program finds the midpoint of two even numbers A and B by continually incrementing the smaller number and decrementing the larger number. You can assume that A and B have been loaded with values before this program starts execution.

Your job: Insert the missing instructions.

```
        .ORIG   x3000
        LD      R0,A
        LD      R1,B
X       NOT R2, R0
        ADD R2, R2, #1
        ADD     R2,R2,R1
        BRz DONE        ; Hint: the missing instruction is a branch.
        ADD     R1,R1,#-1
        ADD R0, R0, #1
        BRnzp   X
DONE    ST      R1,C
        TRAP    x25
A       .BLKW   1
B       .BLKW   1
C       .BLKW   1
        .END
```

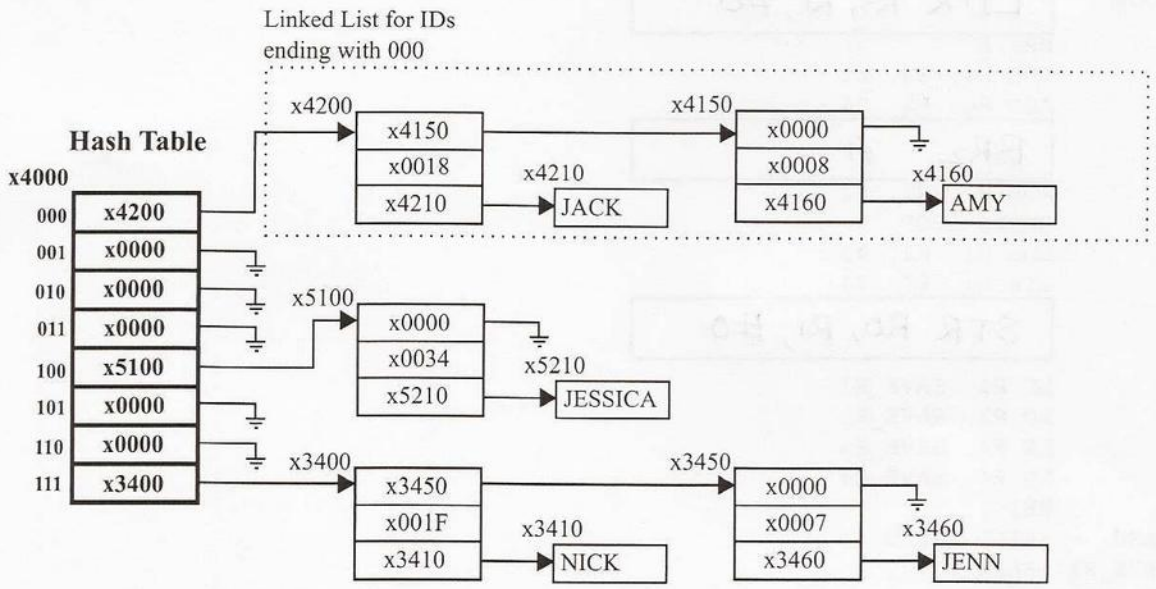
Name: Solution

Problem 6. (20 points):
The primary disadvantage of linked lists is that searching for an element often requires looking at every element in the list. To overcome this issue, we can replace the single linked list with a collection of linked lists, where every element of the original linked list is in exactly one of the *new* linked lists, and we know which linked list to search to find the element. The idea is to compute some function of each element, and to put all elements that produce the same value of that function in the same linked list. That is, in order to know which linked list to search, we compute the function, and that tells us which linked list the element must be in (assuming the element is present in the first place, of course). Thus, if there are n linked lists, we ought to be able to find the element in approximately $\frac{1}{n}$th of the time.

In this problem we will use a collection of linked lists to implement a student directory. Each student has two pieces of data: a 16 bit student ID number stored as an unsigned integer and a name stored as a character string. Each data node consists of 3 words and its configuration is shown below.

word 0	pointer to the next node
word 1	student ID as 16-bit unsigned integer
word 2	pointer to character string of student name

The function that will determine which linked list the student belongs in will simply be bits[2:0] of the student ID. Thus, there must be 8 linked lists. We conveniently store pointers to the first element in each of the 8 linked lists in an 8-entry table, starting at location x4000, as shown below. If there are no elements in a linked list, that table entry would be x0000.



Actually, the data structure world does have names for this mechanism. The function used to determine which linked list is called a "hash function," and the table of pointers to the first element of each linked list is called a "hash table," but you don't need these names to solve the problem.

Problem 6 continues on the next page.

Name: Solution

Your job in this problem is to complete the INSERT subroutine which is used to add a new element (a new student!) to the student directory. This subroutine takes the pointer to the node which is stored in R0 and inserts the node into the appropriate linked list. If there is an element with the same student ID present, the subroutine **replaces** the old element with the new one. That is, the old student is removed from the linked list and the new student is inserted.

```
; INSERT - This subroutine inserts a node into the student directory
; Inputs:  R0 contains a pointer to the node to be inserted
; Outputs;  none
INSERT  ST R1, SAVE_R1
        ST R2, SAVE_R2
        ST R3, SAVE_R3
        ST R4, SAVE_R4

        LDR R2, R0, #1
        AND R3, R2, #7
        LD R1, HASH
        ADD R1, R3, R1
        NOT R2, R2
        ADD R2, R2, #1
LOOP    LDR R3, R1, #0
        BRz B
        LDR R4, R3, #1
        ADD R4, R2, R4
        BR2 A
        ADD R1, R3, #0
        BRnzp LOOP
A       LDR R4, R3, #0
        STR R4, R0, #0
B       STR R0, R1, #0
        LD R1, SAVE_R1
        LD R2, SAVE_R2
        LD R3, SAVE_R3
        LD R4, SAVE_R4
        RET
HASH    .FILL x4000
SAVE_R1 .BLKW 1
SAVE_R2 .BLKW 1
SAVE_R3 .BLKW 1
SAVE_R4 .BLKW 1
```

Name: Solution

Problem 7. (20 points):
Now that the keyboard interrupt is old stuff for you, it is time to introduce two interrupts for the LC-3: INTA and INTB. The necessary hardware has been added to allow them to happen. INTA has priority 2 and an interrupt vector of x50. INTB has priority 4 and an interrupt vector of x60.

Recall that the priority is specified in bits[10:8] of the PSR. In fact, the full PSR specification is:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSR:	Pr	0	0	0	0	Priority			0	0	0	0	0	N	Z	P

where PSR[15] = 0 (supervisor mode), 1 (user mode).
PSR[14:11] = 0000
PSR[10:8] = priority, 0 (lowest) to 7 (highest).
PSR[7:3] = 00000
PSR[2:0] = condition codes for N,Z,P

In this problem you are given the user program and the two interrupt service routines. The user program starts executing at cycle 1 and runs at priority 0.

User program:

```
        .ORIG x3000
        AND R0,R0,#0
        ADD R0,R0,#5
        LD R1,COUNT
        NOT R0,R0
        ADD R0,R0,#1
AGAIN   ADD R2,R0,R1
        BRz DONE
        ADD R1,R1,#-1
        BRnzp AGAIN
DONE    TRAP x25
COUNT   .FILL x000F
        .END
```

INTA Service Routine:

```
        .ORIG x1000
        AND R5,R4,#0
        ADD R5,R5,#-1
        LD R3,VAL
        ADD R3,R3,R5
        ST R3,VAL
        RTI
VAL     .BLKW 1
        .END
```

INTB Service Routine:

```
        .ORIG x2000
        LDI R4,VAL2
        NOT R4,R4
        ADD R4,R4,#1
        STI R4,VAL2
        RTI
VAL2    .FILL xFE08
        .END
```

Assume both interrupts are enabled. Assume 22 cycles are needed to initiate an interrupt when you are in user mode; that is, from the time the test is taken until the interrupt service routine starts executing. Assume it takes 21 cycles if you are in privileged (supervisor) mode. You already know from problem 1 the number of cycles individual instructions take.

Part a: In order to support INTA and INTB, the interrupt vector table must have entries. Show the addresses of these entries and the contents of those memory locations.

Memory address	Content
x0150	x1000
x0160	x2000

Part b: Suppose INTA requests service at cycle 30 and INTB requests service at cycle 68. In which cycle does each **service routine** start executing?

INTA cycle number: 57 — 54-57 accepted.

INTB cycle number: 97 — 92-97 accepted.

Name: Solution

Part c: The table below shows the contents of a section of memory (locations x2FFA to x3002) before the user program starts executing. Show the contents of these locations and the contents of the Stack Pointer in cycle 100.

	Initial	At the end of cycle 100
x2FFA	x0001	x0001
x2FFB	x0010	x0010
x2FFC	x0100	x1002
x2FFD	x1000	x0204
x2FFE	x1100	x3003
x2FFF	x1110	x8001
x3000	x5020	x5020
x3001	x1025	x1025
x3002	x2207	x2207
Stack Pointer	x3000	x2FFC

x0204 also accepted.
x1002
x8001
x3003

Name: Solution

Problem 8. (20 points):
The program stored in memory locations x3000 to x3007 loads a value from memory location x3100, then does some processing, and then stores a result in memory location x3101. Shown below is an incomplete specification of the program. Your job (part 1): complete the specification of the program.

Address	Contents	Assembly code
x3000	0101 001 001 1 00000	AND R1, R1, #0
x3001	0010 000 01111 1110	LD R0, x3100
x3002	0000 110 000000011	BRnz x3006
x3003	0001 0010 0100 0000	ADD R1, R1, R0
x3004	0001 0000 0011 1111	ADD R0, R0, #-1
x3005	0000 1111 1111 1100	BRnzp x3002
x3006	0011 001 0 1111 1010	ST R1, x3101
x3007	1111 0000 0010 0101	HALT

To help you in this process, we have taken a snapshot of part of the state of the machine before the first instruction executes and at several instruction boundaries thereafter, that is, after a number of instructions executed. Part of the snapshot is shown below. Your job (part 2) is to complete the snapshot. Note that the program enters the TRAP x25 service routine after executing 17 instructions. Therefore some instructions must execute more than once.

Note that in the table below: some entries are designated xxxx. You do not have to fill in those entries. Also, you can ignore snapshots for any instructions that are not listed in the table.

Instruction #	PC	MAR	MDR	R0	R1
Initial	x3000	xxxx	xxxx	xxxx	xxxx
1	x3001	xxxx	xxxx	xxxx	0
2	x3002	x3100	3	3	0
3	x3003	xxxx	xxxx	3	0
4	x3004	x3003	x1240	3	3
5	x3005	xxxx	xxxx	x0002	3
9	x3005	xxxx	xxxx	x0001	5
13	x3005	xxxx	xxxx	x0000	6
14	x3002	xxxx	xxxx	0	6
15	x3006	xxxx	xxxx	0	6
16	x3007	x3101	6	0	6
17	xxxx	xxxx	xxxx	0	6

Name: Solution

Problem 9. (20 points):
During the execution of an LC3 program, the processor datapath was monitored for four instructions in the program that were processed consecutively. The table below shows all clock cycles during which the bus was utilized. It shows the clock cycle number, the value on the bus and the state (from the state machine diagram) for some of these clock cycles. Processing of the first instruction starts at clock cycle T. Each memory access in this LC3 machine takes five clock cycles.

Your job: Fill in the missing entries in the table below. You only need to fill in the cells not marked with 'x'.

Note: There are five clock cycles for which you need to provide the control signals. Not all LC3 control signals are shown in the table. However, all control signals that are required for those five clock cycles have been included.

Note: For the DRMUX signal, write '11.9', 'R7', or 'SP'; for the R.W signal, write a 'R' or a 'W'; for the PCMUX signal, write 'PC+1', 'BUS', or 'ADDER'; for all other control signals, write down the actual bit. If a control signal is not relevant in a given cycle, mark it with a dash (i.e., -).

Inst. #	Clock Cycle	Bus	State	Control Signals												
				Gate PC	Gate MDR	Gate ALU	Gate MARMUX	LD. PC	LD. MDR	LD. MAR	LD. CC	LD. Reg	DR MUX	MIO. EN	R.W	PC MUX
Inst. 1	T + 0	x3010	18	1	0	0	0	1	0	1	0	0	—	0	—	PC+1
	T + 6	xF0AB	35	x	x	x	x	x	x	x	x	x	x	x	x	x
	T + 8	x00AB	15	x	x	x	x	x	x	x	x	x	x	x	x	x
	T + 9	x3011	28	1	0	0	0	0	1	0	0	1	R7	1	R	—
	T + 10	x3011	28	1	0	0	0	0	1	0	0	1	R7	1	R	—
	T + 11	x3011	28	1	0	0	0	0	1	0	0	1	R7	1	R	—
	T + 12	x	28	x	x	x	x	x	x	x	x	x	x	x	x	x
	T + 13	x	28	x	x	x	x	x	x	x	x	x	x	x	x	x
	T + 14	x1510	30	x	x	x	x	x	x	x	x	x	x	x	x	x
Inst. 2	T + 15	x1510	18	x	x	x	x	x	x	x	x	x	x	x	x	x
	T+21	x2219	35	x	x	x	x	x	x	x	x	x	x	x	x	x
	T+23	x152A	2	x	x	x	x	x	x	x	x	x	x	x	x	x
	T + 29	x8001	27	0	1	0	0	0	0	0	1	1	11.9	0	—	—
Inst. 3	T + 30	x1511	18	x	x	x	x	x	x	x	x	x	x	x	x	x
	T+ 36	x0804	35	x	x	x	x	x	x	x	x	x	x	x	x	x
Inst. 4	T+40	x1516	18	x	x	x	x	x	x	x	x	x	x	x	x	x
	x	x1200	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	x	x0000	x	x	x	x	x	x	x	x	x	x	x	x	x	x