CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Gurindar Sohi

TAs: Minsub Shin, Lisa Ossian, Sujith Surendran

Midterm Examination 3

In Class (50 minutes)

Friday, November 14, 2014

Weight: 17.5%

NO: BOOK(S), NOTE(S), OR CALCULATORS OF ANY SORT.

The exam has **nine** pages. **Circle your final answers**. Plan your time carefully since some problems are longer than others. You **must turn in the pages 1-8**. Use the blank sides of the exam for scratch work.

The LC-3 instruction set is provided on Page 9

LAST NAME: ____

FIRST NAME: ____

ID#: ____

Problem	**Maximum Points**	**Points Earned**
1	3	
2	2	
3	3	
4	3	
5	5	
6	7	
7	3	
8	4	
Total	30	

Problem 1 **(3 points)**

a) Which of the following LC-3 instructions copies the value of R6 into R1?

```
1) 0001 001 110 1 11111
2) 0101 001 110 1 11111
3) 0101 110 001 1 11111
4) 0001 110 001 1 00000
```

b) Excluding the memory access to fetch the instruction, how many memory accesses are made to fetch and execute the LDI instruction?

1) 4
2) 3
3) 2
4) 1

c) The LC-3 branch instruction `0000 101 000001111` is located at memory address 0x3000. If the branch is taken, what does that imply about the values of the condition codes before the instruction executed?

1) Either N = 1 or P = 1, and Z = 0
2) Either P = 1 or Z = 1, and N = 0
3) Both N = 1 and P = 1, and Z = 0
4) Both N = 1 and Z = 1, and P = 0

Problem 2 **(2 points)**

a) **(1 point)** Write a ***single*** LC-3 instruction to load the number `0x4022` into R2. Assume that your instruction will be located at `0x4000`.

1110 010 000100001

b) **(1 point)** Write a ***single*** LC-3 instruction to load the data stored at memory address `0x4022` into R3. Assume that your instruction will be located at `0x4001`.

0010 011 000100000

Problem 3 **(3 points)**

Assume that the following two LC-3 instructions are a part of a large program:

```
0001 000 000 1 11110
0000 010 000000001
```

a) **(2 points)** If the second instruction (which is a branch) is taken, what can you tell about the value of R0 just before executing these two instructions?

R0 = 2

b) **(1 point)** If the branch instruction is located at address `0x4000`, specify the range of addresses to which you can branch using this instruction.

X3F01 to x4100

Problem 4 **(3 points)**

The table below shows LC-3 instructions starting at `0x3000`, which are executed in sequence. Specify the values at memory locations `0x300F` to `0x3012` after executing each instruction.

Assume that the initial contents of R0 = `0x3010` and R1 = `0x3011`. Also, assume that the initial values of the memory locations `0x300F` to `0x3012` are all zeros.

Address	LC-3 Instruction	Values at memory locations after executing the instruction
`0x3000`	`0011 000 000001111`	Value at `0x300F`: 0x0000 Value at `0x3010`: 0x3010 Value at `0x3011`: 0x0000 Value at `0x3012`: 0x3011
`0x3001`	`0111 001 000 000010`	Value at `0x300F`: 0x0000 Value at `0x3010`: 0x3010 Value at `0x3011`: 0x0000 Value at `0x3012`: 0x3011
`0x3002`	`1011 001 00001110`	Value at `0x300F`: 0x0000 Value at `0x3010`: 0x3010 Value at `0x3011`: 0x0000 Value at `0x3012`: 0x3011

Problem 5 **(5 points)**

Consider the LC-3 program below.

Address	Instruction	Comment
`0x4000`	`0101 100 100 1 00000`	R4 <- 0
`0x4001`	`0001 011 011 1 11011`	R3 <- R3 -5
`0x4002`	`0101 011 011 1 11111`	Just sets the condition flags
`0x4003`	`0000 1 0 0 000000010`	Branch if n to HALT
`0x4004`	`0001 100 100 1 11110`	R4 <- R4 - 2
`0x4005`	`0000 1 1 1 111111000`	Branch if N, Z, or P is set to address 0x4001
`0x4006`	`1111 0000 00000000`	HALT

a) **(2 points)** Fill in the four missing comments in the program above.

b) **(1 point)** If the initial value of R3 is `0x0031`, what is the value of R4 when the HALT instruction is reached?

Answer : -18

c) **(1 point)** If the initial value of R3 is `0x0003`, what is the value of R4 when the HALT instruction is reached?

Answer : 0

d) **(1 point)** What is the minimum value of R3 that causes the value of R4 to be -8 upon reaching the HALT instruction?

Answer : 20

Problem 6 **(7 points)**

We are about to execute the program below. Assume the condition codes before execution of the program are N=1, Z=0, P=0.

Address	Instruction	Comments
0x3000	0011 000 000001100	Store R0 into memory location 0x300D
0x3001	0000 100 000000011	If n flag is set, branch to 0x3005
0x3002	0001 000 000 11011	Subtract 5 from R0 and store the result in R0
0x3003	0101 010 010 0 00 000	R2 <- R2 AND R0
0x3004	1111 0000 00000000	HALT
0x3005	1010 010 000000100	LDI: Load the value from a memory location, whose address is stored in location 0x300A, into R2
0x3006	1111 0000 00000000	HALT

a) **(3 points)** Fill in the three missing instructions in the program above.

b) **4 points)** Suppose a section in memory before execution of the program is as follows:

Address	Value
0x300A	0x300D
0x300B	0x300F
0x300C	0xFACE
0x300D	0x300B

Given the initial values of the below registers, fill in the values after the program has completed execution (i.e., reached a HALT). Give your answers in **hex**.

Register	Initial Value	Final Value
MAR	0X300B	0x300D (Also accepted 0x300E assuming you would have fetched HALT instruction)
MDR	0xDCBA	0x4444 (Also accepted F025 for the same reason above)
R0	0x4444	0x4444
R1	0x300D	0x300D
R2	0x300A	0x4444

Problem 7 **(3 points)**

Assume that you wrote a program which asks the user to enter a number and then identifies whether it is a prime number or a composite number. When you run the program, you see that it created an illegal operation.

(a) **(1 point)** What kind of error have you committed? Explain.

Syntax error (accepted other errors as long as the explanation was valid)

(b) **(1 point)** What are the different options available to you to trace this program and identify the wrong instruction? Explain the options.

Breakpoints, Watchpoints, etc

(c) **(1 point)** Now assume that you were able to trace the bug in the program and after you modified it, assume that it ran successfully (without creating any illegal instructions) and gave the correct output when the user input was 5. However, when the user then gave an input of 50000, it did not give the correct answer. What kind of error do you think you have committed now? Explain the error.

Data error (accepted other errors as long as the explanation was valid)

Problem 8 **(4 points)**

Assume that your friend John has written a large LC-3 program which is working correctly. Column A in the table below shows 4 sets of instructions which are part of his working program. Now, suppose you replaced one set of instructions in Column A with the corresponding set of instructions in Column B. Without making assumptions about any register or memory location, specify if the program is still guaranteed to work correctly. Justify your answer.

Set #	Column A	Column B
1	`0101 001 001 1 00000` (R1 <- R1 AND 0) `0101 010 010 1 00000` (R2 <- R2 AND 0) `0000 010 000000010` (Branch if Z to PC'+2)	`0101 010 010 1 00000` (R2 <- R2 AND 0) `0101 001 001 1 00000` (R1 <- R1 AND 0) `0000 010 000000010` (Branch if Z to PC'+2)
	Answer/Justification: Yes, they are identical	
2	`0001 001 001 1 11111` (R1 <- R1 - 1) `0001 010 010 1 11111` (R2 <- R2 -1) `0000 010 000000010` (Branch if Z to PC'+2)	`0001 010 010 1 11111` (R2 <- R2 -1) `0001 001 001 1 11111` (R1 <- R1 - 1) `0000 010 000000010` (Branch if Z to PC'+2)
	Answer/Justification: No because condition flags on the left column are set based on value of R1. But condition flags on right are set based on the value of R0. So branch could happen on one and not happen on the other. So there is no guarantee that the code will still work.	
3	`0001 001 001 1 11111` (R1 <- R1 - 1) `0011 010 000000011` (R2 <- Mem[PC'+3])	`0011 010 000000011` (R2 <- Mem[PC'+3]) `0001 001 001 1 11111` (R1 <- R1 - 1)
	Answer/Justification: No because PC' is different in both the cases. So, the data loaded into R1 is different.	
4	`0001 001 010 1 11111` (R1 <- R2 - 1) `0111 010 010 000011` (R2 <- Mem[R2]+3)	`0111 010 010 000011` (R2 <- Mem[R2]+3) `0001 001 010 1 11111` (R1 <- R2 - 1)
	Answer/Justification: No, because value at R0 at the end of execution is different.	

LC-3 Instruction Set (Entered by Mark D. Hill on 03/14/2007)

PC′: incremented PC. setcc(): set condition codes N, Z, and P. mem[A]:memory contents at address A. SEXT(immediate): sign-extend immediate to 16 bits. ZEXT(immediate): zero-extend immediate to 16 bits.

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, SR2 ; Addition
| 0   0   0   1 |     DR    |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ADD DR, SR1, imm5 ; Addition with Immediate
| 0   0   0   1 |     DR    |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 + SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND  DR, SR1, SR2 ; Bit-wise AND
| 0   1   0   1 |     DR    |    SR1    | 0 | 0   0 |    SR2    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SR2 also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ AND DR, SR1, imm5 ; Bit-wise AND with Immediate
| 0   1   0   1 |     DR    |    SR1    | 1 |       imm5        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← SR1 AND SEXT(imm5) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ BRx, label (where x = {n,z,p,zp,np,nz,nzp}) ; Branch
| 0   0   0   0 | n | z | p |             PCoffset9             | GO ← ((n and N) OR (z AND Z) OR (p AND P))
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ if (GO is true) then PC ← PC′ + EXT(PCoffset9)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JMP BaseR ; Jump
| 1   1   0   0 | 0   0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← BaseR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSR label ; Jump to Subroutine
| 0   1   0   0 | 1 |                 PCoffset11                |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC′, PC ← PC′ + SEXT(PCoffset11)

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ JSRR BaseR ; Jump to Subroutine in Register
| 0   1   0   0 | 0 | 0   0 |   BaseR   | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ temp ← PC′, PC ← BaseR, R7 ← temp

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LD DR, label ; Load PC-Relative
| 0   0   1   0 |     DR    |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[PC′ + SEXT(PCoffset9)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDI DR, label ; Load Indirect
| 1   0   1   0 |     DR    |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[mem[PC′ + SEXT(PCoffset9)]] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LDR DR, BaseR, offset6 ; Load Base+Offset
| 0   1   1   0 |     DR    |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← mem[BaseR + SEXT(offset6)] also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ LEA, DR, label ; Load Effective Address
| 1   1   1   0 |     DR    |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← PC′ + SEXT(PCoffset9) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ NOT DR, SR ; Bit-wise Complement
| 1   0   0   1 |     DR    |     SR    | 1 | 1   1   1   1   1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ DR ← NOT(SR) also setcc()

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RET ; Return from Subroutine
| 1   1   0   0 | 0   0   0 | 1   1   1 | 0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ PC ← R7

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ RTI ; Return from Interrupt
| 1   0   0   0 | 0   0   0   0   0   0   0   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ See textbook (2nd Ed. page 537).

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ST SR, label ; Store PC-Relative
| 0   0   1   1 |     SR    |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[PC′ + SEXT(PCoffset9)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STI, SR, label ; Store Indirect
| 1   0   1   1 |     SR    |             PCoffset9             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[mem[PC′ + SEXT(PCoffset9)]] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ STR SR, BaseR, offset6 ; Store Base+Offset
| 0   1   1   1 |     SR    |   BaseR   |        offset6        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ mem[BaseR + SEXT(offset6)] ← SR

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ TRAP ; System Call
| 1   1   1   1 | 0   0   0   0 |           trapvect8           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ R7 ← PC′, PC ← mem[ZEXT(trapvect8)]

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ ; Unused Opcode
| 1   1   0   1 |                                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+ Initiate illegal opcode exception
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
```