CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

Prof. Mark D. Hill & Prof. Mikko H. Lipasti
TAs Sanghamitra Roy, Eric Hill, Samuel Javner, Natalie Enright Jerger, & Guoliang Jin

Midterm Examination 3
In Class (50 minutes)
Friday, November 16, 2007
Weight: 15%

CLOSED BOOK, NOTE, CALCULATOR, PHONE, & COMPUTER.

The exam in two-sided and has **TEN** pages, including two blank pages and a copy of the ***LC-3 Instruction Set handout*** on the final page (please feel free to detach this final page, but insert it into your exam when you turn it in).

Plan your time carefully, since some problems are longer than others.

NAME: ____

SECTION:____

ID# ____

Problem Number	**Maximum Points**	**Graded By**
1	4	NEJ
2	4	NEJ
3	6	SJ
4	4	EH
5	4	SR
6	4	GJ
7	4	EH
Total	30	

Problem 1 (4 points)

The following LC-3 instruction is located at memory address x7000.

x7000: 0000 101 000000100

R0 contains 4
R1 contains 3
R2 contains 0
R3 contains 5

a. If the preceding instruction is the one shown below, what is the value of the PC after the instruction at 0x7000 is executed?

```
x6fff:    0001 000 001 1 00001
x7000:    0000 101 000000100
```

0x7005

b. If the preceding instruction is the one shown below, what is the value of the PC after the instruction at 0x7000 is executed?

```
x6fff:    0101 010 011 1 00000
x7000:    0000 101 000000100
```

0x7001

Problem 2 (4 points)

Imagine the DR and BaseR fields of the LDR instruction are each 4 bits wide

If the instruction is 0110 0001 0010 xxxx

R0	x0
R1	x0
R2	x0308
R3	xFF
R4	x123

a. What is the maximum and minimum address that the above instruction could load from?

0x0300 to 0x030F

b. What is the maximum number of registers for DR?
16

Problem 3 (6 points)

The program below checks to see if the value stored in R0 is greater than or equal to the value stored in R5. If R0 is smaller than R5, the value of R5 is copied to R0. Otherwise nothing is done. Insert the missing LC-3 machine language instructions. Adding comments to each machine language instruction will assist in awarding partial credit.

Address	ISA Instruction
x3000	**1001 0101 0111 1111 ; NOT R2, R5**
x3001	0001 0100 1010 0001 ; ADD R2, R2, #1
x3002	0001 0110 0000 0010 ; ADD R3, R0, R2
x3003	**0000 0110 0000 0001 ; BRzp x3005**
x3004	0001 0001 0110 0000 ; ADD R0, R5, #0
x3005	1111 0000 0010 0101 ; HLT

Problem 4 (4 points)

There is something wrong with the following code sequence. This code is supposed to continuously decrement the value stored in R5 until it is equal to zero, and then exit. Explain what happens when we try to execute this code. Comments are provided to save you the effort of decoding the machine language.

Address	ISA Instruction
x3000	0001 1011 0111 1111 ; ADD R5, R5, #-1
x3001	0000 0111 1111 1110 ; BRzp x3000
x3002	1111 0000 0010 0101 ; HLT

Explanation of what is wrong:

Because the instruction at location x3001 branches on the zero condition code, the loop will have an extra iteration.

Problem 5 (4 points)

a. Briefly describe 2 ways to partially execute a program while debugging it.

   (**Any 2 of 3)**

   Single Step: execute 1 instruction at a time

   Breakpoint: tell simulator/program to stop executing when it reaches a specific instruction

   Watchpoint: tell simulator/program to stop executing when the value in specific register or memory location changes

b. Briefly describe the 3 ways to decompose a program into subtasks

   Sequential: do subtask 1 followed by subtask 2

   Conditional: if condition is true, do subtask 1. If condition is false, do subtask 2

   Iterative: repeat subtask over and over until test condition is false

Problem 6 (4 points)

We are about to execute the following program:

Address	**ISA Instruction**
`x3000`	`0010 0000 0000 0101 ; LD R0, x005`
`x3001`	`0110 0000 0000 0000 ; LDR R0, R0, x0`
`x3002`	`0010 0010 1111 0000 ; LD  R1, x0F0`
`x3003`	`0110 0100 0000 1110 ; LDR R2, R0, x0E`
`x3004`	`1111 0000 0010 0101 ; HALT`

The state of the machine before the program starts is given below:

Memory Address	Memory Contents
x3006	xABCD
xABCD	x3220
x2FFF	x4567
x322E	x7564
xABDB	x0001
x30F3	x0020
x200E	x3258
x2257	x0000
x300E	x92FE
x3005	x3010

What will be the final contents of registers R0-R3 when we reach the HALT instruction? Write your answers in hexadecimal format.

Register	**Initial contents**	**Final contents**
R0	x200E	**0x3220**
R1	x200E	**0x0020**
R2	x3001	**0x7564**
R3	x3001	**0x3001**

Problem 7 (4 points)

a. If the value stored in R0 is 1 at the end of the execution of the following instructions, what can be inferred about R5?

Address	Instruction
0x3000	0101 000 000 1 00000 ; R0 R0 AND #0
0x3001	0101 100 101 1 000001 ; R4 R5 AND #1
0x3002	0000 010 000000001 ; BRz, #1
0x3003	0001 000 000 1 00001 ; R0 R0 + #1

a. R5 is equal to 1
b. R5 is even
c. R5 is odd
d. R5 is equal to 0

Answer: c

b. Which of the following LC-3 instructions at address 0x0200 will always clear register R5 (i.e. set the contents of R5 to all zeroes) ?

a. 1110 101 000 000000
b. 0010 101 000 000000
c. 0101 101 101 100000
d. 0001 101 101 100000

Answer: c

Scratch Sheet 1 (in case you need additional space for some of your answers)

Scratch Sheet 2 (in case you need additional space for some of your answers)

LC-3 Instruction Set (Entered by Mark D. Hill on 03/14/2007; last update 03/15/2007)

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. Page 2 has an ASCII character table.

```
 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′ + SEXT(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
```