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

Prof. Mark D. Hill
TAs Marc de Kruijf & Sanghamitra Roy

Midterm Examination 3
In Class (50 minutes)
Wednesday, April 11, 2007
Weight: 15%

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

The exam in two-sided and has **SEVEN** 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: ____

ID# ____

Problem Number	Maximum Points	Grader
1	4	SR
2	4	SR
3	3	SR
4	6	MK
5	5	SR
6	4	MK
7	4	MK
Total	30	

Problem 1 (4 points)

a) Register R1 contains x7531 and register R2 contains x8642. What will be the values of the condition codes N, Z, and P after executing the instruction AND R3, R1, R2? Please explain.

R1: x7531 = 0111 0101 0011 0001
AND R2: x8642 = 1000 0110 0100 0010
--
R3: x0400 = 0000 0100 0000 0000

As R3 contains a positive two's complement value after executing the AND instruction, the condition codes will be N = 0, Z = 0, P = 1.

b) Will a branch instruction predicated on all three Z, N, and P condition codes (i.e. the instruction is BRnzp) ever cause the PC not to jump to the target address/label? Please explain. Note that when the machine is first initialized, the condition code is set to Z.

No. One of N, Z or P condition code is always set and so a branch predicated on all three condition codes must always jump to the target address/label.

Problem 2 (4 points)

Imagine that bit 9 (the 10th bit) of the LD instruction is part of the PCoffset field rather than the destination register field.

a) What would be the range of addresses that we could load from relative to the current PC?

There are 10 bits for specifying the PCoffset.
Range is (-2^9 + 1) to 2^9 with respect to the current PC
(-2^9 to (2^9 - 1) with respect to the incremented PC)

b) How many choices of register would we have to load the data into?

There are 2 bits to specify the destination register => 4 registers

Problem 3 (3 points)

There is an LC-3 instruction that can be used to clear bits (i.e. set them to '0'). Write the instruction that will clear the three least significant (rightmost) bits of register R1 and store the result in register R1. Note that the thirteen most significant bits of register R1 must retain their original values. Write the answer in both machine language and in symbolic form.

LC3 Machine Instruction:

0	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0

The instruction in either symbolic or assembly notation: AND R1, R1, # -8

Problem 4 (6 points)

The program below checks to see if bit 3 (the 4th bit) of R0 is a one. If the bit is set (if it is a one), the program adds the value 4 to whatever is in R1, otherwise it does nothing. Insert the missing ISA machine language instructions. Adding comments to each machine language instruction will assist in awarding partial credit.

Address	ISA Instruction
x3000	0101 0100 1010 0000 ; Clear R2
x3001	0001 0100 1010 1000 ; R2 <- 8
x3002	**0101 0110 1000 0000 ; R3 <- R0 AND R2**
x3003	0000 0100 0000 0001 ; BRz x3005
x3004	**0001 0010 0110 0100 ; R1 <- R1 + 4**
x3005	1111 0000 0010 0101 ; TRAP

Note: For the instruction in x3002, neither the destination register nor the order of operands matters. It is also correct to AND with the immediate value 8 rather than use R2.

Problem 5 (5 points)

There is something wrong with the following code sequence. Explain what happens when we try to execute it. Comments are provided to save you the effort of decoding the machine language.

Address	ISA Instruction
x3000	0101 0000 0010 0000 ; R0 <- R0 AND 0
x3001	0000 0010 0000 0001 ; BRp x3003
x3002	0001 0000 0010 0001 ; R0 <- R0 ADD 1
x3003	0000 0011 1111 1101 ; BRp x3001

Explanation of what is wrong:

This code sequence contains an infinite loop. The program will keep jumping from x3001 to x3003 and then again to x3001, repeatedly, and hence will not terminate.

Problem 6 (4 points)

a) Give a name and short phrase describing each of *three* ways to refine a programming task into smaller sub-tasks.

Sequential: **Do subtask 1 to completion followed by subtask 2 to completion**
Conditional: **If condition is true, do subtask 1, otherwise do subtask 2**
Iterative: **Do subtask until condition becomes true/false.**

b) What is the difference between *syntax* errors and the other types of programming errors discussed in lecture?

Syntax errors result from illegal or improperly formatted instructions. For example, a non-existent opcode (machine language) or instruction specifier (assembly code) will cause a syntax error. All syntax errors are caught by the assembler. The other types of errors discussed in lecture are logic and data errors. In the case of these errors, the program does not meet the problem specification. These types of errors are not detectable by the assembler because the program itself is legal and well-formed.

Problem 7 (4 points)

We are about to execute the following program:

Address	ISA Instruction
`x3000`	`1110 0000 0000 1110 ; LEA R0, x00E`
`x3001`	`0010 0010 0000 1110 ; LD  R1, x00E`
`x3002`	`0110 0100 1100 1110 ; LDR R2, R3, x0E`
`x3003`	`1111 0000 0010 0101 ; HALT`

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

Registers R0 and R1 contain x200E
Memory location x200E contains x3258
Memory location x300E contains x92FE
Memory location x3010 contains x2257

Registers R2 and R3 contain x3001
Memory location x2257 contains x0000
Memory location x300F contains x3010
Memory location x3258 contains x0001

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	**x300F**
R1	x200E	**x2257**
R2	x3001	**x3010**
R3	x3001	**x3001**

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
```