

# ECE/CS 252: INTRODUCTION TO COMPUTER ENGINEERING

## UNIVERSITY OF WISCONSIN—MADISON

Prof. Mikko Lipasti & Prof. Gurindar S. Sohi

TAs: Felix Loh, Daniel Chang, Philip Garcia, Sean Franey, Vignyan Kothinti Naresh, Raghu Raman  
and Newsha Ardalani

### Midterm Examination 3

*In Class (50 minutes)*

*Friday, November 19, 2010*

*Weight: 12.5%*

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

This exam has 7 pages, including one page for the LC3 Instruction Set and two blank pages at the end.  
Plan your time carefully, since some problems are longer than others. You must turn in pages 1 to 7.

LAST NAME: \_\_\_\_\_ **Solution Key** \_\_\_\_\_

FIRST NAME: \_\_\_\_\_

SECTION: \_\_\_\_\_

ID# \_\_\_\_\_

<b>Problem</b>	<b>Maximum Points</b>	<b>Actual Points</b>
<b>1</b>	<b>3</b>	
<b>2</b>	<b>4</b>	
<b>3</b>	<b>2</b>	
<b>4</b>	<b>5</b>	
<b>5</b>	<b>2</b>	
<b>6</b>	<b>5</b>	
<b>7</b>	<b>4</b>	
<b>Total</b>	<b>25</b>	

### Problem 1 (3 Points)

How would you implement the following operation in LC3?

R4 = R1 NOR R2

Write the machine code (binary 16 bit instructions) in the space below.

**NOT R1, R1                      1001 001 001 111 111**

**NOT R2, R2                      1001 010 010 111 111**

**AND R4, R1, R2      0101 100 001 000 010**

### Problem 2 (4 points)

Explain by providing brief definitions of both, the difference between:

I. Data errors and logic errors

**Must mention:**

- a. Data errors: data is incorrect/unexpected**
- b. Logic errors: program is logically wrong -AND/OR- results/program don't match problem statement**

II. Breakpoints and Watchpoints

**Must provide differences that mention (1 point each):**

- a. Breakpoint stops execution at a specified instruction**
- b. Watchpoint stops when specified register or mem location changes**

### Problem 3 (2 points)

If the number of registers in LC3 is doubled, while leaving the instruction size unchanged at 16 bits, what would be the effect, if any, on:

1. The range of values for the ADD immediate instruction:

**They can say:**

**range decreased to -4 to 3**

**-OR-**

**range is 8**

2. The range of addresses a JUMP instruction can have

**No Change**

#### Problem 4 (5 points)

The program below performs multiplication via repeated addition on registers R1 and R2 and stores the result in R0 (i.e.  $R0 \leftarrow R1 * R2$ ). Enter the missing machine language instructions and comments to complete the code (all lines should be commented).

Address	ISA Instruction
x3000	0101 0000 0010 0000 ; Clear R0
x3001	0001 0010 0110 0000 ; $R1 \leftarrow R1 + 0$
x3002	0000 0100 0000 0011 ; BRz x3006
x3003	0001 0000 0000 0010 ; $R0 \leftarrow R0 + R2$
x3004	0001 0010 0111 1111 ; Decrement R1 ( $R1 \leftarrow R1 + -1$ )
x3005	0000 0011 1111 1101 ; BRp x3003
x3005 (alt)	0000 1111 1111 1100 ; BRnzp x3002
x3006	1111 0000 0010 0101 ; TRAP

*Note: 2 possible correct answers for x3005*

#### Problem 5 (2 points)

Consider the following two snippets of LC3 code which achieve the same function:

1.

Address	ISA Instruction
x3000	1010 1010 0000 0001 ; LDI R5, #1

2.

Address	ISA Instruction
x3000	0010 1000 0000 0001 ; LD R4, #1
x3001	0110 1011 0000 0000 ; LDR R5, R4, #0

With the following memory contents:

Address	Data
x3002	x3003
x3003	x007F

Give at least one

advantage of using (1) over (2)?

**Acceptable Answers (only 1 required): More compact, Fewer Registers used**

Give at least one advantage of using (2) over (1)?

**Acceptable Answers: More flexible because of the offset**

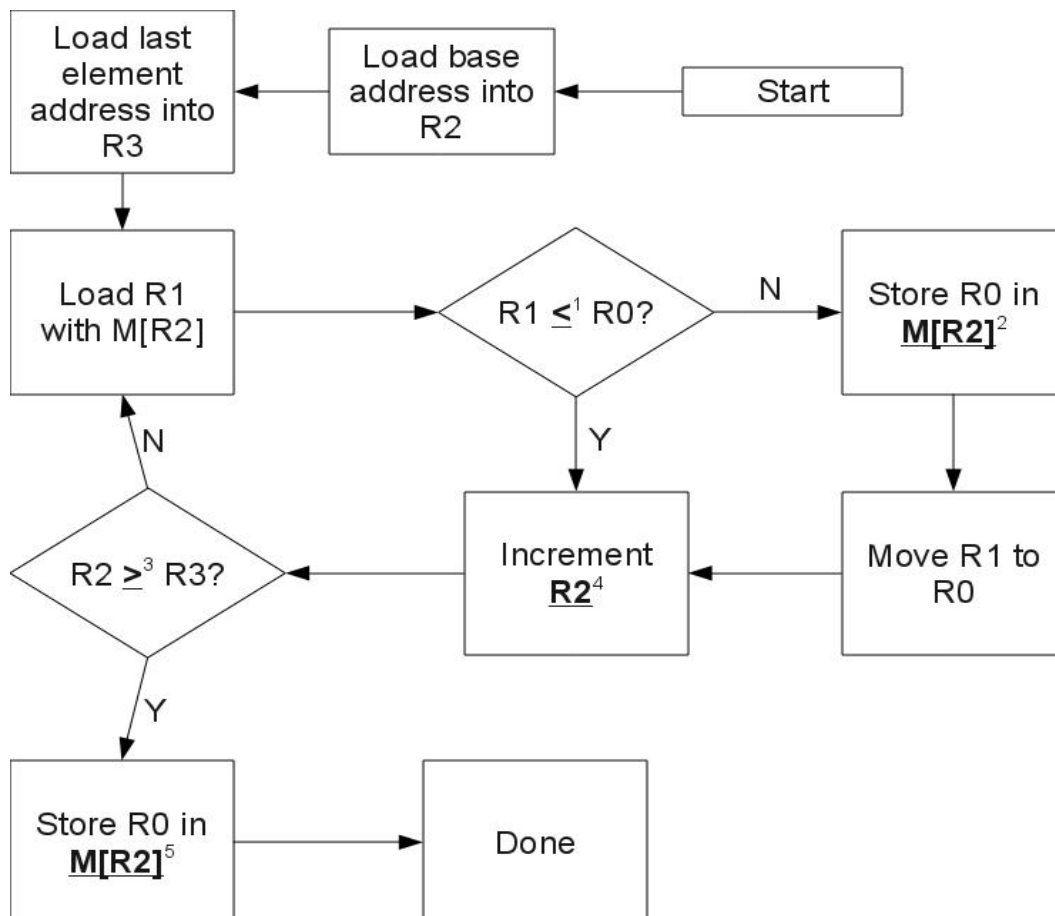
### Problem 6 (5 points)

The flow chart below is for a program that performs an insertion of one element into a list of elements that are sorted in ascending order (i.e. smallest element is at the base address); the element to be inserted is stored in register R0. Briefly, the program works as follows:

With the element to be inserted in R0, the first element of the list is loaded into R1 to be inspected. If R0 is greater than this element, nothing is done and the next element is brought into R1 and inspected. Once R0 is not greater than the element being inspected, it is inserted in that element's memory location. The program then moves the current element to R0. This makes that element the element to be inserted in the remainder of the list and the algorithm continues until the end of the list is reached. The effect is that each subsequent element (after the first insertion) is shifted down by one location.

Address	Initial Value	Final Value
x3100	x001	x001
x3101	x003	x003
x3102	x005	x004
x3103	x008	x005
x3104	x009	x008
x3105	<i>unknown</i>	x009

Fill in the five missing pieces to complete the chart. Remember, R0 contains the element to be inserted.



**Problem 7 (4 points)**

We are about to execute the following program:

Address	ISA Instruction
x3000	1110 0000 0001 0100 ; LEA R0, x014
x3001	0010 0010 0001 0100 ; LD R1, x014
x3002	0110 0100 0000 0010 ; LDR R2, R0, x02
x3003	1010 0110 0001 0001 ; LDI R3, x011
x3004	1111 0000 0010 0101 ; HALT

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

Memory Address	Memory Contents
x3010	x9876
x3011	x3258
x3012	x0000
x3013	x4567
x3014	x3017
x3015	x3018
x3016	x92FE
x3017	x92FF
x3018	x0020
x3019	x1220
x301A	x0001

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

(“NOTE” for R2 indicates correct answers based on mistakes made assigning R0)

Register	Initial contents	Final contents
R0	x200E	<b>x3015</b>
R1	x200E	<b>x92FE</b>
R2	x3001	<b>x92FF</b>  <b>NOTE: if</b> <b>R0=x3014 =&gt; x92FE</b> <b>R0=x3017 =&gt; x1220</b> <b>R0=x3018 =&gt; x0001</b>
R3	x3001	<b>x0020</b>

# 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.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	1		DR		SR1		0	0	0		SR2		
																ADD DR, SR1, SR2 ; Addition
																DR ← SR1 + SR2 also setcc()
	0	0	0	1		DR		SR1		1		imm5				
																ADD DR, SR1, imm5 ; Addition with Immediate
																DR ← SR1 + SEXT(imm5) also setcc()
	0	1	0	1		DR		SR1		0	0	0		SR2		
																AND DR, SR1, SR2 ; Bit-wise AND
																DR ← SR1 AND SR2 also setcc()
	0	1	0	1		DR		SR1		1		imm5				
																AND DR,SR1,imm5 ; Bit-wise AND with Immediate
																DR ← SR1 AND SEXT(imm5) also setcc()
	0	0	0	0		n		z		p		PCoffset9				
																BRx,label (where x={n,z,p,zp,np,nz,nzp}); Branch
																GO ← ((n and N) OR (z AND Z) OR (p AND P))
																if(GO is true) then PC←PC' + SEXT(PCoffset9)
	1	1	0	0		0	0	0		BaseR		0	0	0	0	0
																JMP BaseR ; Jump
																PC ← BaseR
	0	1	0	0		1		PCoffset11								
																JSR label ; Jump to Subroutine
																R7 ← PC', PC ← PC' + SEXT(PCoffset11)
	0	1	0	0		0	0	0		BaseR		0	0	0	0	0
																JSRR BaseR ; Jump to Subroutine in Register
																temp ← PC', PC ← BaseR, R7 ← temp
	0	0	1	0		DR		PCoffset9								
																LD DR, label ; Load PC-Relative
																DR ← mem[PC' + SEXT(PCoffset9)] also setcc()
	1	0	1	0		DR		PCoffset9								
																LDI DR, label ; Load Indirect
																DR←mem[mem[PC'+SEXT(PCoffset9)]] also setcc()
	0	1	1	0		DR		BaseR		offset6						
																LDR DR, BaseR, offset6 ; Load Base+Offset
																DR ← mem[BaseR + SEXT(offset6)] also setcc()
	1	1	1	0		DR		PCoffset9								
																LEA, DR, label ; Load Effective Address
																DR ← PC' + SEXT(PCoffset9) also setcc()
	1	0	0	1		DR		SR		1	1	1	1	1	1	1
																NOT DR, SR ; Bit-wise Complement
																DR ← NOT(SR) also setcc()
	1	1	0	0		0	0	0		1	1	1	0	0	0	0
																RET ; Return from Subroutine
																PC ← R7
	1	0	0	0		0	0	0	0	0	0	0	0	0	0	0
																RTI ; Return from Interrupt
																See textbook (2 <sup>nd</sup> Ed. page 537).
	0	0	1	1		SR		PCoffset9								
																ST SR, label ; Store PC-Relative
																mem[PC' + SEXT(PCoffset9)] ← SR
	1	0	1	1		SR		PCoffset9								
																STI, SR, label ; Store Indirect
																mem[mem[PC' + SEXT(PCoffset9)]] ← SR
	0	1	1	1		SR		BaseR		offset6						
																STR SR, BaseR, offset6 ; Store Base+Offset
																mem[BaseR + SEXT(offset6)] ← SR
	1	1	1	1		0	0	0	0		trapvect8					
																TRAP ; System Call
																R7 ← PC', PC ← mem[ZEXT(trapvect8)]
	1	1	0	1												
																; Unused Opcode
																Initiate illegal opcode exception
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	