

CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Gurindar Sohi

TAs: Junaid Khalid and Pradip Vallathol

Midterm Examination 3

In Class (50 minutes)

Friday, November 9, 2012

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 1-7.**

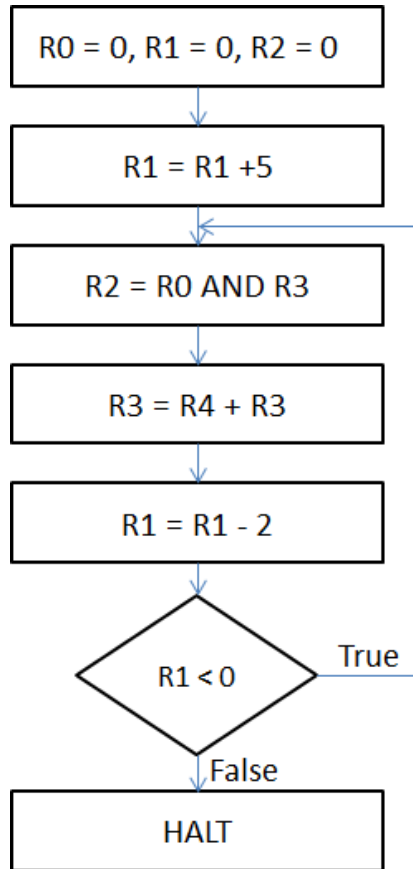
LAST NAME: _____

FIRST NAME: _____

ID# _____

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

Problem 1: The following flowchart is being converted into a sequence of LC-3 instructions as represented in the table below. Fill in the missing instructions and comments. Comments represent a summary of what the instruction does. **(4 Points)**



Address	Instructions	Comments
0x3000	0101 0000 0010 0000	Clear the contents of R0
0x3001	0101 0010 0110 0000	Clear the contents of R1
0x3002	0101 0100 1010 0000	Clear the contents of R2
0x3003	0001 0010 0110 0101	R1 = R1 + 5
0x3004	0101 0100 0000 0011	R2 = R0 AND R3
0x3005	0001 0110 1100 0100	R3 = R3 + R4
0x3006	0001 0010 0111 1110	R1 = R1 - 2
0x3007	0000 1001 1111 1100	If N, branch to x3004
0x3008	1111 0000 0010 0101	HALT

Problem 2: Suppose you are not allowed to use the LC-3 LDI instruction. Write a sequence of LC-3 instructions (in hex) that would achieve the same result as the LC-3 LDI instruction 0xA80C. **(3 Points)**

0x280C ; LD R4, 12

0x6900 ; LDR R4, R4, 0

Problem 3: List and briefly explain the three ways to partially run a program while debugging it. **(3 Points)**

Single Stepping: Execute one instruction at a time.

Breakpoints: Tell the simulator to stop executing at a specific instruction.

Watchpoints: Tell the simulator to stop when the value of a register or memory location changes.

Problem 4: Below is a snapshot of the contents of the 8 registers in LC-3 before and after the instruction at location x3000 is executed. Fill in the bits of the instruction at location x3000 and the values of the P, N and Z flags after the execution of the instruction. **(3 Points)**

Register	Before	After
R0	0xB BBBB	0xB BBBB
R1	0xD DDDD	0xD DDDD
R2	0x2 2222	0x2 2222
R3	0x3 3333	0x3 3333
R4	0x4 4444	0x4 4444
R5	0x5 5555	0x5 5555
R6	0x6 6666	0x6 6666
R7	0x7 7777	0x0 0000

P	0	N	0	Z	1
---	---	---	---	---	---

0x3000:	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
---------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Problem 5: Answer the following questions briefly.

- a) What is the largest positive number that can be represented as an immediate operand in an LC-3 ADD instruction (OPCODE: 0001)? **(1 Point)**

15

- b) Is there a sequence of LC-3 instructions that will cause the condition codes at the end of the sequence to be N=1, Z=1 and P=0? Explain. **(2 Points)**

No, the result of an instruction can only be either positive, negative or zero.

- c) What is the largest address that an LC-3 Load PC-Relative (LD) instruction (OPCODE: 0010), located at 0x4000, can load from? **(1 Point)**

0x4100

- d) What is the difference between logical errors and syntax errors? **(1 Point)**

Syntax error: typing error resulting in illegal operation

Logical error: legal program, but results not matching problem statement

- e) Name the three basic constructs that are used to decompose a task. **(1 Point)**

Sequential, Conditional, Iterative

- f) List any two constituents of the *trace* of a program's execution. **(2 Points)**

Sequence of instructions executed, Results being generated

Problem 6: Consider the following LC-3 program:

Address	Instruction
x3000	0010 0010 1001 1111
x3001	1001 0100 0111 1111
x3002	0001 0100 1010 0001
x3003	0001 0110 1000 0000
x3004	0011 0110 1001 1110

Suppose the contents of registers and memory locations represent the “State” of the system at any time. The table below shows the state of the system at various stages of execution of the above program.

State 0: State before executing the program

State 1: State after executing instruction at location x3002

State 2: State after executing instruction at location x3004.

Fill in the values for State 1 and State 2 in the table below. **(4 Points)**

	State 0	State 1	State 2
R0:	x1208	x1208	x1208
R1:	x2D7C	x3001	x3001
R2:	xE373	xCFFF	xCFFF
R3:	x2053	x2053	xE207
R4:	x33FF	x33FF	x33FF
PC:	x3000	x3003	x3005
...			
x30A0:	x3001	x3001	x3001
x30A1:	x7A00	x7A00	x7A00
x30A2:	x7A2B	x7A2B	x7A2B
x30A3:	xA700	xA700	xE207
..			

Problem 7: The following table shows a program in a part of the LC-3's memory:

Address	Instruction	Comments
0x3000	0001 011 011 0 00 010	$R3 = R3 + R2$
0x3001	0000 100 000000010	If N, branch to x3004
0x3002	0001 010 010 1 00001	$R2 = R2 + 1$
0x3003	0101 011 011 000 010	$R3 = R3 \text{ AND } R2$
0x3004	1001 011 011 111111	$R3 = \text{NOT}(R3)$
0x3005	1001 010 010 111111	$R2 = \text{NOT}(R2)$

If the value of $R3=0x0009$ and $R2=0x00B3$ *after* the execution of above program, what is known about $R2$ and $R3$ *before* the execution of the program? Fill in the comments column with the summary of what each instruction does as you work through the problem. **(5 Points)**

$R2 = 0xFF4C$

$R3 = 0x00AA$

Extra page for hand written work, if needed. This page is not required and will NOT affect your grade. You don't even need to hand this page in.

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		
-----																ADD DR, SR1, SR2 ; Addition	
0	0	0	0	1	DR		SR1	0	0	0	0	SR2				DR ← SR1 + SR2 also setcc()	

-----																ADD DR, SR1, imm5 ; Addition with Immediate	
0	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	0	0	PC ← BaseR	

-----																JSR label ; Jump to Subroutine	
0	1	0	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	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	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 (2 nd 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	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		