CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Gurindar Sohi, Kai Zhao
TAs: Daniel Griffin, Neha Mittal, Annie Lin, Mohit Verma, Yuzhe Ma

Examination 3
In Class (50 minutes)
Wednesday, November 16, 2016
Weight: 17.5%

NO: BOOK(S), NOTE(S), CALCULATORS OR ELECTRONIC DEVICES OF ANY SORT.
The exam has **eleven pages**. You **must turn in the pages 1-10. Circle your final answers**. Plan your time carefully since some problems are longer than others. Use the blank sides of the exam for scratch work.

LAST NAME: _____
FIRST NAME: _____
Section: _____
ID#: _____

Problem	Maximum Points	Points Earned
1	3	
2	4	
3	3	
4	3	
5	3	
6	8	
7	7	
Total	31	

Problem 1 **(3 points)**

Fill in the six missing comments in the program below:

Address	Instruction	Comment
0x4000	0101 011 011 1 00000	R3 <- 0
0x4001	0001 110 011 1 00001	R6 <- R3 + 1
0x4002	0101 100 101 0 00 110	R4 <- R5&R6
0x4003	0000 010 000000001	BRz 0x4005
0x4004	0001 000 000 1 00001	R0 <- R0 + 1
0x4005	0001 110 110 0 00 110	R6 <- R6 + R6 = 2*R6
0x4006	0001 011 011 1 00001	R3 ← R3 + 1
0x4007	0001 001 011 1 11100	R1 ← R3 – 4
0x4008	0000 100 111111001	BRn 0x4002
0x4009	1111 0000 00100101	HALT

Problem 2 **(4 points)**

Address	Instruction	Comment
0x3000	0101 010 010 1 00000	R2 ← 0
0x3001	0001 001 001 1 11111	R1 ← R1 – 1
0x3002	0001 001 001 1 11111	R1 ← R1 – 1
0x3003	0000 100 000000010	BRn x3006
0x3004	0001 010 010 1 00001	R2 ← R2 + 1
0x3005	0000 111 111111011	BRnzp x3001
0x3006	1111 0000 00100101	HALT

a) The above program processes a value initially stored in register R1 according to an algorithm, and stores the result in register R2. Assuming the initial value in R1 is greater than 0, describe how the value in R2 is related to the value that was initially in R1 when the program reaches the HALT instruction at address x3006?

R2 is the rounded down value of R1 / 2. Aka, it's floor(R2/2)

b) What is the final value of R2 if R1 is initially the decimal value 12? Give your answer in decimal.

R2 = 6

Problem 3 **(3 points)**

Shown below are the contents of memory and registers before and after the LC-3 instruction at location 0x4080 is executed. Identify the instruction stored in x4080 and give your answer in hexadecimal form. (There is enough information below to uniquely specify the instruction). **Explain your reasoning to receive credit**; **no explanation means no credit even if your final answer is correct.**

	Before	**After**
R0	x1000	x1000
R1	x10A1	x10A1
R2	x2300	x2300
R3	x1234	x1234
R4	x11AA	x11AA
R5	x2BEF	x2BEF
R6	x1254	x1254
R7	x1421	x1421
mem[x4050]	x3001	x3001
mem[x4051]	xADD1	xADD1
mem[x4052]	x2412	x2412
mem[x4053]	x3213	x2300
mem[x4054]	xFFFF	xFFFF

x35D2

Problem 4 **(3 points)**

The following (incomplete) binary code snippet accepts an input value in register R3, increments it by 2 if the value is even, and then halts. Odd values are left untouched. This can be represented in pseudo code as:

if R3 is divisible by 2 then:
    R3 ← R3 + 2
end if
halt

Complete the code below, by filling in the LC-3 instructions (in binary format) at memory locations x3001 and x3003; the instructions at memory locations x3002 and x3004 have already been filled in for you.

The PC register is set to x3001 before this code executes.

Address	Instruction
0x3001	0101 000 011 1 00001
0x3002	0000 001 000000001
0x3003	0001 011 011 1 00010
0x3004	1111 0000 00100101

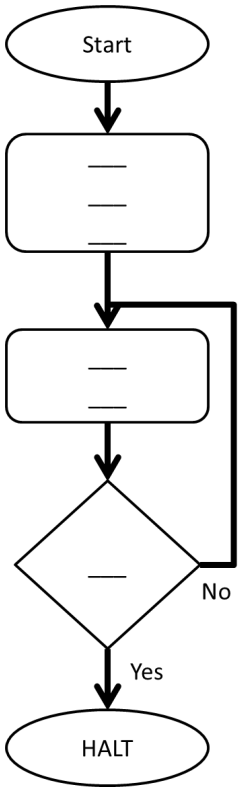
Note that TRAP x25 is used to halt execution.

The DR in 0x3001 can be any register except R3.

Problem 5 **(3 points)**

The diagram shown to the right represents the flow chart of a program that multiplies **the integer numbers 23 and 10 together, and leaves the result in register R3**. The table below gives the register operations that implement this program. Note that each register operation might translate into multiple LC-3 instructions. Fill in the spaces in the program diagram with the letters from the table that correctly implement this program. There are multiple combinations of letter assignments that will work for this program. Choose any single assignment that works. **Each blank should contain a single letter option, and not all of the options below will be used.**

Letter	Operations
A	R0 ← 23
B	R2 = 0?
C	R3 ← 0
D	R2 ← 10
E	R3 ← R5
F	R2 ← R2 – 1
G	R0 = 0?
H	R3 ← R3 + R0



A, C, D
H, F
B

Problem 6 **(8 Points)**

a) (2 points) We wish to execute a single LC-3 instruction that will subtract the decimal number 16 from register R1 and put the result into register R2. Can we do it? If yes, write the LC-3 instruction to do so in its binary format. If not, explain why not.

Yes, 0001 010 001 1 10000

b) (1 point) Consider the following LC-3 instruction located at address 0x4010:

0010 010 101001110

What is the memory address whose contents are loaded into R2? Show your work and give answer in hex. **(No credit without shown work, even if your answer is correct.)**

x4011 + xFF4E = x3F5F = 0011 1111 0101 1111

c) (1 point) Consider the following LC-3 instruction:

0000 101 000001110

Does the execution of the above instruction change any condition codes? Why or why not? **(No credit without an explanation, even if your answer is correct.)**

No, only instructions that write to registers change condition codes.

d) (1 point) Consider the following LC-3 branch instruction located at memory address 0x3000:
0000 101 000001111
If the value of the condition codes before executing this instruction are (N=0, Z=1, P=0), then what is the value of the PC after the above instruction finishes execution?

PC = 0x3001

e) (1 point) Name at least two types of errors that can occur when writing a program?

i) Syntax Errors
ii) Logic Errors
iii) Data Errors
iv) Runtime Errors (not discussed, but it occurs when out of memory)

f) (2 points) Match the following four statements (on the right side) with the letters (A, B, or C) for their corresponding subtask constructs (shown on the left).

A	Sequential
B	Conditional
C	Iterative

C	'For each O, do P'
B	'If G, then do H'
A	'Do E, then do F'
C	'Do M until N'

Problem 7 **(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 000001011	Store R0 into memory location 0x300C
0x3001	0001 000 000 1 11101	Subtract 3 from R0 and store the result in R0
0x3002	0000 001 000000010	If p flag is set, branch to 0x3005
0x3003	0101 010 010 0 00 000	R2 ← R2 AND R0
0x3004	0000 111 000000001	BRnzp 0x3006
0x3005	1010 010 000000111	LDI: Load the value from a memory location, whose address is stored in location 0x300D, into R2
0x3006	1111 0000 00100101	HALT (Trap x25)

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	0x300B
0x300B	0x300F
0x300C	0xACED
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
Memory Address Register (MAR)	0x300B	X300B or x3006
Memory Data Register (MDR)	0xABCD	X300F or xF025
Instruction Register (IR)	0x1000	xA407 or xF025
R0	0x5555	0x5552
R1	0x300D	X300D
R2	0x300A	X300F

Deleted: x

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 | 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
```