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 1-8.** Use the blank sides of the exam for scratch work.

**The LC-3 instruction set is provided on Page 9**
<table>
<thead>
<tr>
<th>Problem</th>
<th>Maximum Points</th>
<th>Points Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1  

(3 points)

a) Which of the following LC-3 instructions copies the value of R7 into R2?

1) 0001 010 111 1 11111
2) 0101 010 111 1 11111
3) 0101 111 010 1 11111
4) 0001 111 010 1 00000

b) Excluding the memory access to fetch the instruction, how many memory accesses are made to fetch and execute the STI instruction?

1) 1
2) 2
3) 3
4) 4

c) The LC-3 branch instruction 0000 110 000001111 is located at memory address 0x3000. If the branch is taken, what does that imply about the values of the condition codes before the instruction executed?

1) Either N = 1 or P = 1, and Z = 0
2) Either N = 1 or Z = 1, and P = 0
3) Both P = 1 and Z = 1, and N = 0
4) Both N = 1 and P = 1, and Z = 0

Problem 2  

(2 points)

a)  (1 point) Write a single LC-3 instruction to load the number 0x4025 into R2. Assume that your instruction will be located at 0x4000.

1110 010 000100100

b)  (1 point) Write a single LC-3 instruction to load the data stored at memory address 0x4025 into R3. Assume that your instruction will be located at 0x4001.

0010 011 000100011
Problem 3 (3 points)

The table below shows LC-3 instructions starting at 0x3000, which are executed in sequence. Specify the values at memory locations 0x300F to 0x312 after executing each instruction.

Assume that the initial contents of R0 = 0x3010 and R1 = 0x3011. Also, assume that the initial values of the memory locations 0x300F to 0x312 are all zeros.

<table>
<thead>
<tr>
<th>Address</th>
<th>LC-3 Instruction</th>
<th>Values at memory locations after executing the instruction</th>
</tr>
</thead>
</table>
| 0x3000  | 0011 000 00001111| Value at 0x300F: 0x0000  
Value at 0x3010: 0x3010  
Value at 0x3011: 0x0000  
Value at 0x3012: 0x3011 |
| 0x3001  | 0111 001 000 000010| Value at 0x300F: 0x0000  
Value at 0x3010: 0x3010  
Value at 0x3011: 0x0000  
Value at 0x3012: 0x3011 |
| 0x3002  | 1011 001 00001110| Value at 0x300F: 0x0000  
Value at 0x3010: 0x3010  
Value at 0x3011: 0x0000  
Value at 0x3012: 0x3011 |

Problem 4 (3 points)

Assume that the following two LC-3 instructions are a part of a large program:

0001 000 000 1 111111 0000 010 00000001

a) **(2 points)** If the second instruction (which is a branch) is taken, what can you tell about the value of R0 just before executing these two instructions?

R0 – 1 = 0  =>  R0 was 1 before execution

b) **(1 point)** If the branch instruction is located at address 0x5000, specify the range of addresses to which you can branch using this instruction.

0x4F01 to 0x5100
Problem 5 (5 points)

Consider the LC-3 program below.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4000</td>
<td>0101 100 100 1 00000</td>
<td>R4 &lt;- 0</td>
</tr>
<tr>
<td>0x4001</td>
<td>0001 011 011 1 11011</td>
<td>R3 &lt;- R3 -5</td>
</tr>
<tr>
<td>0x4002</td>
<td>0101 011 011 1 11111</td>
<td>Just sets the condition flags</td>
</tr>
<tr>
<td>0x4003</td>
<td>0000 1 0 0 0000000010</td>
<td>Branch if n to HALT</td>
</tr>
<tr>
<td>0x4004</td>
<td>0001 100 100 1 11110</td>
<td>R4 &lt;- R4 - 2</td>
</tr>
<tr>
<td>0x4005</td>
<td>0000 1 1 1 111111000</td>
<td>Branch if N, Z, or P is set to address 0x4001</td>
</tr>
<tr>
<td>0x4006</td>
<td>1111 0000 00000000</td>
<td>HALT</td>
</tr>
</tbody>
</table>

a) (2 points) Fill in the four missing comments in the program above.

b) (1 point) If the initial value of R3 is 0x0032, what is the value of R4 when the HALT instruction is reached?

Answer: -20

c) (1 point) If the initial value of R3 is 0x0002, what is the value of R4 when the HALT instruction is reached?

Answer: 0

d) (1 point) What is the minimum value of R3 that causes the value of R4 to be -8 upon reaching the HALT instruction?

Answer: 20
Problem 6  

We are about to execute the program below. Assume the condition codes before execution of the program are \(N=1, Z=0, P=0\).

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3000</td>
<td>0011 000 000001100</td>
<td>Store R0 into memory location 0x300D</td>
</tr>
<tr>
<td>0x3001</td>
<td>0000 100 00000011</td>
<td>If n flag is set, branch to 0x3005</td>
</tr>
<tr>
<td>0x3002</td>
<td>0001 000 000 11011</td>
<td>Subtract 5 from R0 and store the result in R0</td>
</tr>
<tr>
<td>0x3003</td>
<td>0101 010 010 0 00 000</td>
<td>R2 ← R2 AND R0</td>
</tr>
<tr>
<td>0x3004</td>
<td>1111 0000 00000000</td>
<td>HALT</td>
</tr>
<tr>
<td>0x3005</td>
<td>1010 010 000000100</td>
<td>LDI: Load the value from a memory location, whose address is stored in location 0x300A, into R2</td>
</tr>
<tr>
<td>0x3006</td>
<td>1111 0000 00000000</td>
<td>HALT</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x300A</td>
<td>0x300D</td>
</tr>
<tr>
<td>0x300B</td>
<td>0x300F</td>
</tr>
<tr>
<td>0x300C</td>
<td>0xFEED</td>
</tr>
<tr>
<td>0x300D</td>
<td>0x300B</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Register</th>
<th>Initial Value</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR</td>
<td>0x300B</td>
<td>0x300D (Also accepted 0x300E assuming you would have fetched HALT instruction)</td>
</tr>
<tr>
<td>MDR</td>
<td>0xDFEA</td>
<td>0x3333 (Also accepted F025 for the same reason above)</td>
</tr>
<tr>
<td>R0</td>
<td>0x3333</td>
<td>0x3333</td>
</tr>
<tr>
<td>R1</td>
<td>0x300D</td>
<td>0x300D</td>
</tr>
<tr>
<td>R2</td>
<td>0x300A</td>
<td>0x3333</td>
</tr>
</tbody>
</table>
Problem 7 \hspace{1cm} (3 points)

Assume that you wrote a program which asks the user to enter a number and then identifies whether it is a prime number or a composite number. When you run the program, you see that it created an illegal operation.

(a) \textbf{(1 point)} What kind of error have you committed? Explain.

\textbf{Syntax error} (accepted other errors as long as the explanation was valid)

(b) \textbf{(1 point)} What are the different options available to you to trace this program and identify the wrong instruction? Explain the options.

\textbf{Breakpoints, Watchpoints, etc}

(c) \textbf{(1 point)} Now assume that you were able to trace the bug in the program and after you modified it, assume that it ran successfully (without creating any illegal instructions) and gave the correct output when the user input was 7. However, when the user then gave an input of 50000, it did not give the correct answer. What kind of error do you think you have committed now? Explain the error.

\textbf{Data error} (accepted other errors as long as the explanation was valid)
Problem 8  

(4 points)

Assume that your friend John has written a large LC-3 program which is working correctly. Column A in the table below shows 4 sets of instructions which are part of his working program. Now, suppose you replaced one set of instructions in Column A with the corresponding set of instructions in Column B. Without making assumptions about any register or memory location, specify if the program is still guaranteed to work correctly. Justify your answer.

<table>
<thead>
<tr>
<th>Set #</th>
<th>Column A</th>
<th>Column B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0101 000 000 1 00000 (R0 &lt;- R0 AND 0)</td>
<td>0101 001 001 1 00000 (R1 &lt;- R1 AND 0)</td>
</tr>
<tr>
<td></td>
<td>0101 001 001 1 00000 (R1 &lt;- R1 AND 0)</td>
<td>0101 001 001 1 00000 (R0 &lt;- R0 AND 0)</td>
</tr>
<tr>
<td></td>
<td>0000 010 00000010 (Branch if Z to PC'+2)</td>
<td>0000 010 00000010 (Branch if Z to PC'+2)</td>
</tr>
</tbody>
</table>
|       | **Answer/Justification:**  
|       | Yes, they are identical | **Answer/Justification:**  
| 2     | 0001 000 000 1 11111 (R0 <- R0 - 1) | 0001 001 001 1 11111 (R1 <- R1 -1) |
|       | 0001 001 001 1 11111 (R1 <- R1 -1) | 0001 000 000 1 11111 (R0 <- R0 - 1) |
|       | 0000 010 00000010 (Branch if Z to PC'+2) | 0000 010 00000010 (Branch if Z to PC'+2) |
|       | **Answer/Justification:**  
|       | No because condition flags on the left column are set based on value of R1. But condition flags on right are set based on the value of R0. So branch could happen on one and not happen on the other. So there is no guarantee that the code will still work. | **Answer/Justification:**  
| 3     | 0001 000 000 1 11111 (R0 <- R0 - 1) | 0011 001 00000011 (R1 <- Mem[PC'+3]) |
|       | 0011 001 00000011 (R1 <- Mem[PC'+3]) | 0001 000 000 1 11111 (R0 <- R0 - 1) |
|       | **Answer/Justification:**  
|       | No because PC’ is different in both the cases. So, the data loaded into R1 is different. | **Answer/Justification:**  
| 4     | 0001 000 001 1 11111 (R0 <- R1 - 1) | 0111 001 001 000011 (R1 <- Mem[R1]+3) |
|       | 0111 001 001 000011 (R1 <- Mem[R1]+3) | 0001 000 001 1 11111 (R0 <- R1 - 1) |
|       | **Answer/Justification:**  
|       | No, because value at R0 at the end of execution is different. | **Answer/Justification:**  

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

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
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | DR | SR1 | 0 | 0 | 0 | 0 | SR2 | ADD DR, SR1, SR2 ; Addition
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | SR1 | 0 | 0 | 0 | 0 | SR2 | DR ← SR1 + SR2 also setcc()
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 0 | 1 | DR | SR1 | 1 | imm5 | ADD DR, SR1, imm5 ; Addition with Immediate
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 1 | DR | SR1 | 1 | imm5 | DR ← SR1 + SEXT(imm5) also setcc()
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 0 | 1 | DR | SR1 | 1 | imm5 | AND DR, SR1, SR2 ; Bit-wise AND
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 1 | DR | SR1 | 1 | imm5 | DR ← SR1 AND SR2 also setcc()
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 1 | 0 | 0 | BaseR | 0 | 0 | 0 | 0 | 0 | JMP BaseR ; Jump
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 0 | PCoffset11 | PC ← BaseR
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | BaseR | JSR label ; Jump to Subroutine
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 0 | BaseR | 0 | 0 | 0 | 0 | 0 | JSR label ; Jump to Subroutine in Register
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 1 | 0 | DR | PCoffset9 | LD DR, label ; Load PC-Relative
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | 0 | mem[PC' + SEXT(PCoffset9)] also setcc() | R7 ← PC', PC ← PC' + SEXT(PCoffset9)
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 0 | DR | PCoffset9 | LDI DR, label ; Load Indirect
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | mem[mem[PC' + SEXT(PCoffset9)]] also setcc() | DR ← mem[mem[PC' + SEXT(PCoffset9)]]
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 1 | 0 | DR | BaseR | offset6 | LDR DR, BaseR, offset6 ; Load Base+Offset
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 1 | 1 | 0 | DR | BaseR | offset6 | DR ← mem[BaseR + SEXT(offset6)] also setcc()
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | DR | PCoffset9 | LEA, DR, label ; Load Effective Address
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | mem[PC' + SEXT(PCoffset9)] also setcc() | DR ← PC' + SEXT(PCoffset9)
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 1 | 1 | DR | SR | NOT DR, SR ; Bit-wise Complement
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | mem[mem[PC' + SEXT(PCoffset9)]] also setcc() | DR ← NOT(SR) also setcc()
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | RET ; Return from Subroutine
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | PCoffset9 | PC ← R7
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | RFI ; Return from Interrupt
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | See textbook (2nd Ed. page 537).
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 0 | 1 | 1 | ST SR, label ; Store PC-Relative
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 0 | 1 | 1 | STI, SR, label ; Store Indirect
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 1 | 1 | STR SR, BaseR, offset6 ; Store Base+Offset
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 1 | 1 | 1 | TRAP ; System Call
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 0 | 1 | 0 | 1 | BaseR | offset6 | mem[BaseR + SEXT(offset6)]
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Initiate illegal opcode exception
+----------------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+