Name:________

Problem 1 (20 points):_______
Problem 2 (20 points):_______
Problem 3 (20 points):_______
Problem 4 (20 points):_______
Problem 5 (20 points):_______
Total (100 points):_________

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

I will not cheat on this exam.

_________________________
Signature

GOOD LUCK!
Problem 1. (20 points):
Part a. (5 points):
A student is debugging his program. His program does not have access to memory locations x0000 to x2FFF. Why that is the case we will discuss before the end of the semester. The term is "privileged memory" but not something for you to worry about today.

He sets a breakpoint at x3050, and then starts executing the program. When the program stops, he examines the contents of several memory locations and registers, then hits single step. The simulator executes one instruction and then stops. He again examines the contents of the memory locations and registers. They are as follows:

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>x3050</td>
<td>x3051</td>
</tr>
<tr>
<td>R0</td>
<td>x2F5F</td>
<td>xFFFF</td>
</tr>
<tr>
<td>R1</td>
<td>x4200</td>
<td>x4200</td>
</tr>
<tr>
<td>R2</td>
<td>x0123</td>
<td>x0123</td>
</tr>
<tr>
<td>R3</td>
<td>x2323</td>
<td>x2323</td>
</tr>
<tr>
<td>R4</td>
<td>x0010</td>
<td>x0010</td>
</tr>
<tr>
<td>R5</td>
<td>x0000</td>
<td>x0000</td>
</tr>
<tr>
<td>R6</td>
<td>x1000</td>
<td>x1000</td>
</tr>
<tr>
<td>R7</td>
<td>x0522</td>
<td>x0522</td>
</tr>
<tr>
<td>M[x3050]</td>
<td>x6??</td>
<td>x6??</td>
</tr>
<tr>
<td>M[x4200]</td>
<td>x5555</td>
<td>x5555</td>
</tr>
<tr>
<td>M[x201]</td>
<td>xFFFF</td>
<td>xFFFF</td>
</tr>
</tbody>
</table>

Complete the contents of location x3050

x3050 01100000001010001

Part b. (5 points):
A student is writing a program and needs to subtract the contents of R1 from the contents of R2, and put the result in R3. Instead of writing:

NOT R3, R1
ADD R3, R3, #1
ADD R3, R3, R2

He writes:

NOT R3, R1
.FILL x16E1
ADD R3, R3, R2

He assembles the program and attempts to execute it. Does the subtract execute correctly? Why or why not?

Circle one: YES/NO.

Explain in not more than fifteen words.

*FILL x16E1 is assembled anagual to the assembled instruction ADD R3, R3, #1*
Part c. (5 points):
An assembly language program contains the following subroutine, which the LC-3 assembler stores in memory, starting at location x3070.

Construct the symbol table entries for the subroutine. **Hint: I am asking you to ONLY construct the symbol table entries for this subroutine, and nothing more.**

```
INPUT   ST R7, SAVER7
        LO R1, MINUS
        LDA R5, BUFFER
        LO R0, LF
        TRAP x21
        LDA R0, PROMPT
        TRAP x22
        LO R0, LF
        TRAP x21
AGAIN   TRAP x20
        STR R0, R5, #0
        ADD R0, R0, R1
        BRz NEXT
        ADD R5, R5, #1
        BRnzp AGAIN
NEXT    LD R7, SAVER7
        RET
SAVER7 .BLKW 1
MINUS   .FILL xFFDD
BUFFER  .BLKW x21
PROMPT  .STRINGZ "Type a word, then type #"
LF      .FILL x0A
```

<table>
<thead>
<tr>
<th>Label</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>x3070</td>
</tr>
<tr>
<td>AGAIN</td>
<td>x3079</td>
</tr>
<tr>
<td>NEXT</td>
<td>x307F</td>
</tr>
<tr>
<td>SAVER7</td>
<td>x3081</td>
</tr>
<tr>
<td>MINUS</td>
<td>x3082</td>
</tr>
<tr>
<td>BUFFER</td>
<td>x3083</td>
</tr>
<tr>
<td>PROMPT</td>
<td>x30A4</td>
</tr>
<tr>
<td>LF</td>
<td>x30BD</td>
</tr>
</tbody>
</table>
Part d. (5 points):
An Aggie (always an Aggie!) modified the service routine that the operating system executes as a result of a user program executing the TRAP x20 instruction. The modification consists of inserting three lines of code into the trap service routine:

1. Adding the following instruction to the beginning of the service routine:

   \[
   \text{LD R2, MASK}
   \]

2. Inserting the instruction \text{AND R0, R1, R2} in the place shown in the original service routine:

   \begin{verbatim}
   AGAIN     LDI R1, KBSR
             AND R0, R1, R2
             BRzep AGAIN
             LDI R0, KBDR
   \end{verbatim}

3. Inserting the following pseudo-op immediately after \text{RET}:

   \[
   \text{MASK .FILL x7FFF}
   \]

The complete TRAP service routine after adding the three changes:

\begin{verbatim}
ST R1, SaveR1
ST R2, SaveR2
LD R2, MASK
;
AGAIN LDI R1, KBSR
       AND R0, R1, R2
       BRzep AGAIN
       LDI R0, KBDR
;
LD R1, SaveR1
LD R2, SaveR2
;
RET
Mask .FILL x7FFF
KBSR .FILL xFEO0
KBDR .FILL xFEO2
SaveR1 .BLKW 1
SaveR2 .BLKW 1
\end{verbatim}

Your job: Answer the question: Will the trap service routine still work, and explain why or why not in fifteen words or fewer.

\[
\text{No. The inserted AND instruction prevents the ready bit from ever being detected, creating an infinite loop.}
\]
Problem 2. (20 points):
The following program pushes elements onto a stack with JSR PUSH and pops elements off of the stack with JSR POP.

```
.ORIG X3000
lea r6, stack_base

x     trap x20 ; getc
      trap x21 ; out
      add r1, r0, x-0a ; x-0a is ASCII code for line feed,
                          ; x-0a is the negative of x0a
      brz y
      jsr push
      brnzep x

y     lea r2, stack_base
      not r2, r2
      add r2, r2, #1
      add r3, r2, r5
      brz done
      jsr pop
      trap x21 ; out
      brnzep y

done  trap x25 ; halt
stack .blkw 5
stack_base .fill x0fff

push  add r6, r6, #-1
      str r0, r6, #0
      ret

pop   ldr r0, r6, #0
      add r6, r6, #1
      ret

.end
```

What will appear on the screen if a user, sitting at a keyboard, typed the three keys a, b, c, followed by the <Enter> key?

```
abc
```

What will happen if a user, sitting at a keyboard, typed the eight keys a, b, c, d, e, f, g, h, followed by the <Enter> key? (Please, no more than fifteen words in the box.)

```
the stack will overflow and overwrite the TRAP x25 instruction, causing an infinite loop
```
Problem 3. (20 points):

An aggressive young engineer decides to build and sell the LC-3, but is told that if he wants to succeed, he really needs a SUBTRACT instruction. Given the unused opcode 1101, he decides to specify the SUBTRACT instruction as follows:

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SR2</td>
</tr>
</tbody>
</table>

The instruction is defined as: \( DR \leftarrow SR2 - SR1 \), and the condition codes are set. Assume \( DR, SR1, \) and \( SR2 \) are all different registers.

To accomplish this, the engineer needs to add three states to the state machine and a mux and register \( A \) to the data path. The modified state machine is shown below and the modified data path is shown on the next page. The mux is controlled by a new control signal \( SR2SEL \) which selects one of its two sources.

**SR2SEL/1**: \( SR2OUT, \text{REGISTER}_{A} \)

Your job:

For the state machine shown below, fill in the empty boxes with the control signals that are needed in order to implement the SUBTRACT instruction.

For the data path, fill in the value in register \( A \).
Problem 4. (20 points):
In class we talked about stacks, which are LIFO (Last In, First Out) mechanisms that allow us to push (insert) and pop (remove) elements from the top. Another data structure is a queue. It operates as a FIFO (First In, First Out) mechanism. Elements are removed from the front and inserted in the back, much like the way a queue works in our daily lives.

Our queue is implemented as a linked list. Each element consists of two words: a pointer to the next element that entered the queue and a value. We also need two pointers, one to the front of the queue which we use to remove elements, and one to the last element of the queue which we use to add another element. The last element points to NULL (x0000).

The figure below shows a queue with three elements, the first is A, the second is B, the last is C. M[x3100] contains the address of the front of the queue. M[x3101] contains the address of the last element of the queue.

If we add an element D and we remove the elements A and B, the queue looks like this:
Your job: Complete the subroutines to dequeue (remove) the front element of the queue and enqueue (insert) a new element to the back of the queue. After the DEQUEUE subroutine is executed, R3 should contain the address of the element that was just dequeued; before the ENQUEUE subroutine is executed, R3 will contain the address of the new element to be enqueued. You do NOT have to worry about the case when the queue is (or becomes) empty; that is, you can assume the queue will always have at least one element before and after any operation.

**DEQUEUE**

```
ST R0, A
LDI R3, B
LDR R0, R3, #0
STI R0, B
```

```
LD R0, A
RET
```

```
.A: .BLKW 1
```

```
B: .FILL x3100
```

**ENQUEUE**

```
ST R0, C
LDI R0, D
STR R3, R0, #0
STI R3, D
```

```
LD R0, C
RET
```

```
C: .BLKW 1
```

```
D: .FILL x3101
```
Problem 5. (20 points):

During the execution of an LC-3 program, an instruction in the program starts executing at clock cycle $T$ and requires 15 cycles to complete.

The table below lists ALL five clock cycles during the processing of this instruction which require use of the bus. The table shows for each of those clock cycles: which clock cycle, the state of the state machine, the value on the bus, and the important control signals that are active during that clock cycle.

Note: In class on Monday, I gave an example where it took 18 clock cycles for memory to read or write. Part (d) of this problem asks you how many clock cycles it takes for memory to read or write in this example.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>State</th>
<th>Bus</th>
<th>Important Control Signals For This Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>18</td>
<td>x3010</td>
<td>LD.MAR = 1, LD.PC = 1, PCMux = PC + 1, GatePC = 1</td>
</tr>
<tr>
<td>$T + 4$</td>
<td>35</td>
<td>xA202</td>
<td>GateMDR = 1, LD.FR = 1</td>
</tr>
<tr>
<td>$T + 6$</td>
<td>10</td>
<td>x3013</td>
<td>ADDRMUX = PC, ADDRMUX = PC + 1, MARMUX = ADDER, GateMARMUX = 1, LD.MAR = 1</td>
</tr>
<tr>
<td>$T + 10$</td>
<td>26</td>
<td>x4567</td>
<td>GateMDR = 1, LD.MAR = 1</td>
</tr>
<tr>
<td>$T + 14$</td>
<td>27</td>
<td>x0000</td>
<td>LD.REG = 1, LD.CC = 1, GateMDR = 1, DR = 001</td>
</tr>
</tbody>
</table>

a. Fill in the missing entries in the table.

b. What is the instruction being processed?

   **LDI R1, #2**

c. Where in memory is that instruction?

   **x3010**

d. How many clock cycles does it take memory to read or write?

   **3**

e. There is enough information above for you to know the contents of three memory locations. What are they and what are their contents?

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3010</td>
<td>xA202</td>
</tr>
<tr>
<td>x3013</td>
<td>x4567</td>
</tr>
<tr>
<td>x4567</td>
<td>x0000</td>
</tr>
</tbody>
</table>