Name: ________________________________

Problem 1 (15 points): ________
Problem 2 (10 points): ________
Problem 3 (10 points): ________
Problem 4 (10 points): ________
Problem 5 (10 points): ________
Problem 6 (10 points): ________
Problem 7 (10 points): ________
Problem 8 (10 points): ________
Problem 9 (15 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 written legibly on each sheet of the exam.

I will not cheat on this exam.

______________________________
Signature

GOOD LUCK!
Problem 1 (15 points)

Part a (5 points): Here is a list of the 16 opcodes. Circle the ones that write to a general purpose register (R0 to R7) at some point during the instruction cycle.

<table>
<thead>
<tr>
<th>ADD</th>
<th>AND</th>
<th>BR</th>
<th>JMP</th>
<th>JSR</th>
<th>LD</th>
<th>LEA</th>
<th>LDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>NOT</td>
<td>RTI</td>
<td>ST</td>
<td>STI</td>
<td>STR</td>
<td>TRAP</td>
<td>reserved</td>
</tr>
</tbody>
</table>

Part b (5 points): Program A running at priority level 0 starts running on the LC-3 at the start of time unit 1. Program A requires 8 time units to complete. The table below shows information for three interrupting events (x, y, z): when each occurs, the priority level of each, and the number of time units needed by the corresponding interrupt service routine (X, Y, Z) to complete its task.

x: interrupt occurs at end of time unit 6. Priority 4. X needs 3 time units.
y: interrupt occurs at end of time unit 8. Priority 8. Y needs 5 time units.
z: interrupt occurs at end of time unit 9. Priority 5. Z needs 2 time units.

In the boxes below, write the letter (A, X, Y, or Z) of the program or service routine running during that time unit.

Note: We have inserted the answer for time units 1, 2, and 3.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Part c (5 points): Label the gate inputs to each of the 8 transistors in the figure below so that the operation of the transistor circuit behaves as specified in the truth table below. You can label each gate input as A, \( \bar{A} \), B, or \( \bar{B} \).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{Diagram of transistor circuit} \]
Problem 2 (10 points)

A subroutine \textsc{test\_for\_neg} can be used to examine a value and return a 1 in R5 if it is negative, and a 2 if it is positive or zero. The routine is shown below:

\begin{verbatim}
TEST_FOR_NEG   BRn   SKIP
    AND   R5, R5, #0
    ADD   R5, R5, #2
    BRnzp DONE
SKIP           AND   R5, R5, #0
DONE           ADD   R5, R5, #1
               RET
\end{verbatim}

Suppose we are writing a program that needs this subroutine. For example, we need to test the value contained at location A. We did the following and everything worked fine.

\begin{verbatim}
LD   R0, A
JSR  TEST_FOR_NEG
\end{verbatim}

An A&M student decided that the \texttt{JSR} opcode is unnecessary, and replaced it in our main program with two instructions, as follows:

\begin{verbatim}
LD   R0, A
LEA  R7, #1
BRnzp TEST_FOR_NEG
\end{verbatim}

Part a (5 points): After making this change, the program would not assemble. What was the assemble-time error? Answer in the box provided below in fewer than 20 words. Note: More than 20 words will get a 0.

Answer:

Part b (5 points): With the help of an OU student, they fixed the assembly-time error. When they tried to execute the program, they got a run-time error. What was the run-time error? Answer in the box provided below in fewer than 20 words. Note: More than 20 words will get a 0.

Answer:
Problem 3 (10 points)

The 8 general purpose registers of the LC-3 (R0 to R7) make up the Register File. To write a value to a register, the LC-3 control unit must supply 16 bits of data (BUS[15:0]), a destination register (DR[2:0]), and a write enable signal (LD.REG) to load a register. The Combinational Logic Block below shows inputs BUS[15:0], DR[2:0], and LD.REG and outputs DinR0[15:0], DinR1[15:0], DinR2[15:0], … DinR7[15:0], LD.R0, LD.R1, LD.R2, … LD.R7.

Your job: Add wires, logic gates, and standard logic blocks as necessary to complete the Combinational Logic Block. Note: If you use a standard logic block, it is not necessary to show the individual gates. However, it is necessary to identify the logic block specifically (e.g., “16-to-1 mux”), along with labels for each relevant input or output, according to its function.
Problem 4 (10 points)

The table below shows part of the state of the LC-3 (the contents of the eight registers, the PC, the condition codes, and memory locations x2000 through x2007) at a particular instant in time (the column labeled Before). Then the LC-3 was single-stepped twice (i.e., two LC-3 instructions were executed), producing the state of the machine indicated by the column labeled After.

Your job: Fill in the five missing entries in the table indicated by question marks (?). Note: We have supplied the high order hex digit for the contents of memory location x2006.

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>x0505</td>
<td>x0505</td>
</tr>
<tr>
<td>R1</td>
<td>x1FFF</td>
<td>x1FFF</td>
</tr>
<tr>
<td>R2</td>
<td>x3006</td>
<td>x3006</td>
</tr>
<tr>
<td>R3</td>
<td>x2100</td>
<td>x2101</td>
</tr>
<tr>
<td>R4</td>
<td>x4567</td>
<td>x4567</td>
</tr>
<tr>
<td>R5</td>
<td>xFFFF</td>
<td>xFFFF</td>
</tr>
<tr>
<td>R6</td>
<td>x2002</td>
<td>x2004</td>
</tr>
<tr>
<td>R7</td>
<td>x3010</td>
<td>x3010</td>
</tr>
<tr>
<td>PC</td>
<td>x2006</td>
<td>x3007</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>M[x2000]</td>
<td>x20A0</td>
<td></td>
</tr>
<tr>
<td>M[x2001]</td>
<td>x0702</td>
<td></td>
</tr>
<tr>
<td>M[x2002]</td>
<td>x3007</td>
<td></td>
</tr>
<tr>
<td>M[x2003]</td>
<td>x8004</td>
<td></td>
</tr>
<tr>
<td>M[x2004]</td>
<td>x601A</td>
<td></td>
</tr>
<tr>
<td>M[x2005]</td>
<td>x0501</td>
<td></td>
</tr>
<tr>
<td>M[x2006]</td>
<td>? x1</td>
<td></td>
</tr>
<tr>
<td>M[x2007]</td>
<td>? x</td>
<td></td>
</tr>
</tbody>
</table>
Problem 5 (10 points)

Part a (3 points): Generate the symbol table for the program below. You may not need all of the spaces provided. Note: One of the symbols, PUSH, is global and has already been done for you.

```
.EXTERNAL PUSH

.ORIG x3000
LEA R6, BOTTOM
LEA R1, MESSAGE
AND R0, R0, #0
JSR PUSH
AGAIN LDR R0, R1, #0
BRz DONE
JSR PUSH
ADD R1, R1, #1
BRnzp AGAIN
DONE ADD R0, R6, #0
PUTS
HALT
MESSAGE .STRINGZ "STACKS ARE COOL"
.BLKW #20
BOTTOM .BLKW #1
.END
```

Part b (7 points): What does this program output to the console? Assume that the PUSH and POP subroutines are already written for you and work EXACTLY as described in class. Answer in the box provided, please.
Problem 6 (10 points)

A program (part of which is shown below) uses a stack to store values that the program operates on. The stack is implemented in memory locations x5FF8 (MAX) to x5FFF (BASE). At the time the instruction stored at the label AGAIN is executed, the Stack Pointer (R6) contains x5FF8.

Your job is to show the contents of the stack after the program has executed. Also, show the final contents of the Stack Pointer.

You can assume that the PUSH and POP routines have been written for you and that they behave EXACTLY as described in class. Recall that success (0) and failure (1) information is passed back to the calling program through R5, and if the stack operation failed, R0 and R6 remained unchanged.

```
AGAIN AND RO, RO, #0
JSR POP
ADD R1, RO, #0
AND RO, RO, #0
JSR POP
ADD R2, RO, #0
AND RO, RO, #0
JSR POP
ADD RO, RO, R1
ADD RO, RO, R2
ADD R5, R5, #0
BRp DONE
JSR PUSH
BRnzp AGAIN
DONE JSR PUSH
HALT
```

<table>
<thead>
<tr>
<th>R6</th>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x5FF8</td>
<td>x5FF8</td>
</tr>
<tr>
<td>2</td>
<td>x5FF9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x5FA</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x5FB</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x5FC</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x5FD</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x5FE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x5FF</td>
<td></td>
</tr>
</tbody>
</table>

R6: x5FF8
R6: ____
Problem 7 (10 points)

The statement of this problem is identical to problem 9 on Problem Set 6; only the values and the instruction you have to work out are different.

The figure below shows the part of the LC-3 data path that deals with memory and I/O. Note the signals labeled A through F. A is the memory enable signal; if it is 1, memory is enabled, if it is 0, memory is disabled. B, C, and D are the load enable signals for the Device Registers that can be written to. If the load enable signal is 1, the corresponding register is loaded with a value, otherwise it is not. E[15:0] is the 16-bit output of INMUX. F[1:0] are the two select lines for INMUX.

The initial values of some of the processor registers and the I/O registers, and some memory locations are as follows:

\[
\begin{align*}
R0 &= \text{xFE00} \\
R1 &= \text{x0039} \\
PC &= \text{x3000} \\
KBSR &= \text{x8000} \\
KBDR &= \text{x0061} \\
DSR &= \text{x8000} \\
DDR &= \text{x0031} \\
M[x3009] &= \text{xFE00} \\
M[x300A] &= \text{xFE02} \\
M[x300B] &= \text{xFE04} \\
M[x300C] &= \text{xFE06}
\end{align*}
\]

As you know, during one complete instruction cycle, memory-I/O is accessed either one, two, or three times. Your job in this problem is to specify all the memory-I/O accesses associated with processing the instruction STI R1,#11.

Complete the table with the values of each control signal and register at the end of each cycle in which a memory-I/O access occurs. If the value of a signal does not matter in a cycle, put an X in the corresponding entry in that row. If the instruction does not require three memory-I/O accesses, draw a line through each unnecessary row in the table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x3000</td>
<td>STI R1,#11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 8 (10 points)

The program below inputs a single digit from the keyboard (1-9) and then echoes it to the console. Let’s call the number the user typed N. The program displays to the console every Nth letter of the alphabet, starting with the letter A. Some of the instructions are missing. Fill in the blanks to make the program work as specified. Note: Each blank corresponds to one instruction. Some example output is provided below:

Example 1:
1ABCDEFHIJKLMNOPQRSTUVWXYZ

Example 2:
2ACEGIKMOQSUWY

Example 3:
5AFKPUZ

.ORIG x3000
TRAP x20
TRAP x21

; R1 = N, converted from ascii

LEA R2, ALPHA

LOOP

BRzz END
TRAP x21

BRnzp LOOP

END HALT
ALPHA .STRINGZ "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.FILL #0
.END
Problem 9 (15 points)

A program encounters a breakpoint and halts. The computer operator does not change the state of the computer in any way, but immediately presses the run button to resume execution.

The table below shows the contents of MAR and MDR for the first nine memory accesses that the LC-3 performs after resuming execution.

Your job: Fill in the missing entries.

<table>
<thead>
<tr>
<th>MAR</th>
<th>MDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st:</td>
<td>x5020</td>
</tr>
<tr>
<td>2nd:</td>
<td>xF0F0</td>
</tr>
<tr>
<td>3rd:</td>
<td></td>
</tr>
<tr>
<td>4th:</td>
<td>x2000</td>
</tr>
<tr>
<td>5th:</td>
<td>x040A</td>
</tr>
<tr>
<td>6th:</td>
<td>x61FE</td>
</tr>
<tr>
<td>7th:</td>
<td></td>
</tr>
<tr>
<td>8th:</td>
<td>xC1C0</td>
</tr>
<tr>
<td>9th:</td>
<td>x4002</td>
</tr>
</tbody>
</table>
The Stack Protocol

01 ; Subroutine for carrying out the PUSH and POP functions. This
02 ; program works with a stack consisting of memory locations BASE
03 ; through MAX. R6 is the stack pointer. R0 contains the data.
04 ;
05 PUSH   ST   R1,SaveR1 ; R1 is needed by PUSH routine
06 ADD    R1,FULL ; FULL contains -(MAX)
07 ADD    R1,R1,R6
08 BRz    Fail_exit ; SP=MAX (no room on the stack)
09 ;
0A ADD    R6,R6,#-1
0B STR    R0,R6,#0 ; R0 gets PUSHed onto the stack
0C BR     Success_exit
0D ;
0E POP    ST   R1,SaveR1 ; R1 is needed by POP routine
0F ADD    R1,EMPTY ; EMPTY contains -(BASE+1)
10 ADD    R1,R1,R6
11 BRz    Fail_exit ; SP=(BASE+1) (nothing on the stack)
12 ;
13 LDR    R0,R6,#0 ; Top of stack gets POPped to R0
14 ADD    R6,R6,#1
15 ;
16 Success_exit AND R5,R5,#0 ; Success code: R5=0
17 ADD    R5,R5,#0
18 RET
19 Fail_exit AND R5,R5,#0 ; Fail code: R5=1
1A ADD    R5,R5,#1
1B LD      R1,SaveR1
1C RET
1D FULL   .FILL -MAX
1E EMPTY  .FILL -(BASE+1)
1F SaveR1 .BLKW 1

The PSR register

![Diagram of the PSR register with Privilege (0: Supervisor, 1: User) and Priority (0 - 7)]