Name: Solution

Part A:
Problem 1 (10 points): 5/5
Problem 2 (10 points): 10
Problem 3 (10 points): 10
Problem 4 (10 points): 10
Problem 5 (10 points): 10

Part A (50 points): 45

Part B:
Problem 6 (20 points): 20
Problem 7 (20 points): 20
Problem 8 (20 points): 20
Problem 9 (20 points): 20

Part B (130 points): 130

Total (130 points): 130

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!
(HAVE A GREAT SEMESTER BREAK)
Problem 1. (10 points):

Part a. (5 points): Construct the output of the truth table for the PLA shown.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Part b. (5 points): In the transistor circuit below, all transistors in the path to the power supply are shown. None of the transistors in the path to ground are shown.

Your job:
1. Draw the missing transistor circuit in the box.

*Note: this question was moved to part B of the exam.

*The following additions to the problem were made during the exam:

1. There are exactly 4 transistors in the box.
2. Every input combination results in a zero or one (the output is never a floating or a short circuit).
**Name:** Solution

**Problem 2.** (10 points): The following program is assembled and stored in the LC-3’s memory. The PC is initially set to x3000. The program is run until the computer halts.

Your job: What is contained in location B after the computer stops?

```
.ORIG x3000
AND R0, R0, #0
NOT R1, R0
ADD R5, R0, #3
ADD R0, R0, #1
ADD R0, R0, R0
ADD R0, R0, R0
ADD R0, R0, R0
NOT R3, R0
AND R1, R3, R1
ADD R0, R0, R0
ADD R0, R0, R0
ADD R0, R0, R0
ADD R0, R0, R0
NOT R3, R0
ADD R1, R3, R1
ADD R5, R5, #1
BRp A
ST R1, B
TRAP x25
B .BLKW 1
```

What is the value in location B? [Box containing the value 7777]
Problem 3. (10 points): This problem involves a new 16-bit floating point data type, specified as follows:

\[
\begin{array}{ccc}
\text{Sign} & \text{Exponent} & \text{Fraction} \\
\hline
\end{array}
\]

To add two floating point values, we first make sure their binary points line up (they have the same exponents).

The assembly program shown below, after the missing instructions have been filled in, compares the exponents of two floating point numbers that have been previously loaded into locations A and B. If the exponents are the same, R5 is set to 0 before the RET is taken. If the exponents are different, R5 is set to 1 before the RET is taken.

Your job: Fill in the missing instructions.

```
.ORIG x3000
ST R0, SaveR0
ST R1, SaveR1

ST R2, C

LD R2, MASK
AND R5, R5, #0
LD R0, A
LD R1, B

AND R0, R0, R2
AND R1, R1, R2
NOT R1, R1
ADD R1, R1, #1

ADD R0, R0, R1

BRZ DONE
ADD R5, R5, #1

DONE
LD R0, SaveR0
LD R1, SaveR1

LD R2, C

RET

MASK .FILL #7F00

A .BLKW #1
B .BLKW #1
SaveR0 .BLKW #1
SaveR1 .BLKW #1
C .BLKW #1

.END
```
Problem 4. (10 points):

1. What opcodes use IR[11:9] as inputs to SR1? 
   **ST, STR, STI**

2. Where does the control signal of this mux come from? Be specific!
   **From IR[5]**

3. What opcodes use this input to the MARMUX? 
   **TRAP**

*Note: "1" in the diagram above corresponds to question 1, "2" corresponds to question 2, and "3" corresponds to question 3.*
Problem 5. (10 points): The modulo operator (A mod B) is the remainder one gets when dividing A by B. For example, 10 mod 5 is 0, 12 mod 7 is 5.

The program below is supposed to perform A mod B, where A is in x3100 and B is in x3101. The result should be stored at location x3200. However, the programmer made a serious mistake, so the program does not work. You can assume that A and B are both positive integers.

```
.ORIG x3000 ; Line 1
LD R3, L2 ; 2
LDR R0, R3, #0 ; 3
LDR R1, R3, #1 ; 4
NOT R2, R1 ; 5
ADD R2, R2, #1 ; 6
L1
ADD R0, R0, R2 ; 7
BRzp L1 ; 8
ADD R0, R0, R1 ; 9
ST R0, L3 ; 10
HALT ; 11
L2 .FILL x3100 ; 12
L3 .FILL x3200 ; 13
.END ; 14
```

Part A. After the instruction at line 6 has executed, what are the contents of R0,R1,and R2? NOTE: the correct answer in each case is one of the following: A, -A, B, -B, 0, 1, -1.

R0: \[ A \]  
R1: \[ B \]  
R2: \[ -B \]

Part B. There is a bug in the program. The instruction at line \[ 10 \] should be \[ STI R0, L3 \]
Problem 6. (20 points): A free list is a collection of blocks of consecutive memory locations of various sizes that are not being used by currently executing programs. A free list is normally organized as a linked list, where each element in the linked list is associated with a single block of memory. Each element consists of three words: the address of the next element in the linked list, the number of consecutive memory locations in this block, and the starting address of the block. R1 contains the address of a memory location that points to the first node in the free list.

\[
\begin{array}{cccc}
R1: & xC000 & xC000: & x8000 \\
     & x8000: & xA000 & xA000: \\
     & x801: & x0100 & xA001: \\
     & x802: & x6000 & xA002: & x7050
\end{array}
\]

The free list above consists of two nodes, one of size x100 comprising M[x6000] to M[x60FF] and one of size x10 comprising locations M[x7050] to M[x705F].

A procedure MALLOC is used to provide blocks of storage to programs that request them.

If Program A needs n words of memory, it loads n into R2 and does a JSR to MALLOC. MALLOC finds the first block in the free list that can satisfy the request, loads the starting address of the block into R0, updates the free list to reflect the fact that those n words are no longer available, and does a JMP R7. If MALLOC can't find a block that can satisfy the request, x0000 is returned in R0. If the block that supplied the n-words consisted of exactly n-words (a perfect fit), then no words from that block are still available and so the node is removed from the free list.

On the next page is the procedure MALLOC. Your job: Add the missing instructions.
ALLOC
ST R1, SAVE_R1
ST R3, SAVE_R3
ST R4, SAVE_R4
ST R5, SAVE_R5
AND R0, R0, #0
NOT R3, R2
ADD R3, R3, #1

NEXT_NODE
LDR R4, R1, #0
BRz RETURN
LDR R5, R4, #1
ADD R5, R3, R5
BRz PERFECT_FIT
BRp FRAGMENT

LDR R1, R1, #0

BRnzp NEXT_NODE

PERFECT_FIT
LDR R0, R4, #2

LDR R4, R4, #0

STR R4, R1, #0
BRnzp RETURN

FRAGMENT
LDR R0, R4, #2
STR R5, R4, #1

ADD R1, R2, R0

STR R1, R4, #2

RETURN
LD R5, SAVE_R5
LD R4, SAVE_R4
LD R3, SAVE_R3
LD R1, SAVE_R1
RET

SAVE_R1 .BLKW 1
SAVE_R3 .BLKW 1
SAVE_R4 .BLKW 1
SAVE_R5 .BLKW 1
Problem 7. (20 points): During the processing of an LC-3 program by the data path we have been using in class, the computer stops due to a breakpoint set at x3000. The contents of certain registers and memory locations at that time are as follows:

\[ R2 \text{ through } R7: \ x0000 \]
\[ M[x3000]: \ x1263 \]
\[ M[x3003]: \ x0000 \]

The LC-3 is restarted and executes exactly four instructions. To accomplish this, a number of clock cycles are required. In 15 of those clock cycles, the bus must be utilized. The table below lists those 15 clock cycles in sequential order, along with the values that are gated onto the LC-3 bus in each.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st:</td>
<td>x3000</td>
</tr>
<tr>
<td>2nd:</td>
<td>x1263</td>
</tr>
<tr>
<td>3rd:</td>
<td>x009A</td>
</tr>
<tr>
<td>4th:</td>
<td>x3001</td>
</tr>
<tr>
<td>5th:</td>
<td>xA000</td>
</tr>
<tr>
<td>6th:</td>
<td>x3002</td>
</tr>
<tr>
<td>7th:</td>
<td>x3003</td>
</tr>
<tr>
<td>8th:</td>
<td>x1263</td>
</tr>
<tr>
<td>9th:</td>
<td>x3002</td>
</tr>
<tr>
<td>10th:</td>
<td>x3000</td>
</tr>
<tr>
<td>11th:</td>
<td>x3003</td>
</tr>
<tr>
<td>12th:</td>
<td>x1263</td>
</tr>
<tr>
<td>13th:</td>
<td>x0093</td>
</tr>
<tr>
<td>14th:</td>
<td>x1263</td>
</tr>
<tr>
<td>15th:</td>
<td>x009D</td>
</tr>
</tbody>
</table>

Part a: Fill in the missing entries above.

Part b: What are the four instructions that were executed?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R1, #3</td>
<td></td>
</tr>
<tr>
<td>LDI R0, #0</td>
<td></td>
</tr>
<tr>
<td>ST R0, #0</td>
<td></td>
</tr>
<tr>
<td>ADD R1, R1, #3</td>
<td></td>
</tr>
</tbody>
</table>

Part c: What are the contents of R0 and R1 after the four instructions execute?

\[ R0: \ x1263 \]
\[ R1: \ x009D \]
Problem 8. (20 points): Let's use the unused opcode to implement a new instruction, as shown below:

To accomplish this, we will need a small addition to the data path, shown below in boldface:

The following five additional states are needed to control the data path to carry out the work of this instruction.

Note: State B loads the negative of the contents of MDR into TEMP.
Name: Solution

Part a: Complete the table below by identifying the values of the control signals needed to carry out the work of each state.

Note: For a particular state, if the value of a control signal does not matter, fill it with an X.

<table>
<thead>
<tr>
<th>State</th>
<th>LDPC</th>
<th>LD.MAR</th>
<th>LD.MDR</th>
<th>LD.CC</th>
<th>LD.TEMP</th>
<th>GatePC</th>
<th>GateMDR</th>
<th>GateALU</th>
<th>SR1MUX[1:0]</th>
<th>ALUMUX</th>
<th>ALCK[1:0]</th>
<th>MIO.EN</th>
<th>R.W</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 13</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>State A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>State B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>State C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>State D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

LD.PC
0: load not enabled
1: load enabled

LD.MAR
0: load not enabled
1: load enabled

LD.MDR
0: load not enabled
1: load enabled

LD.CC
0: load not enabled
1: load enabled

LD.TEMP
0: load not enabled
1: load enabled

GatePC
0: do not pass signal
1: pass signal

GateMDR
0: do not pass signal
1: pass signal

GateALU
0: do not pass signal
1: pass signal

SR1MUX
00: Source IR[11:9]
01: Source IR[8:6]
10: Source R6

ALUMUX
0: Choose SR1
1: Choose TEMP

ALUK
00: ADD
01: AND
10: NOT
11: Pass input A

MIO.EN
0: MIO not enabled
1: MIO enabled

R.W
0: Read
1: Write

Part b: What does the new instruction do?

If the contents of Reg3 equals M[Reg2], the CC are set to Z and M[Reg2] gets the contents of Reg1. Otherwise, memory is unchanged, and the CC are set to N or D.
Problem 9. (20 points): Consider a two player game where the players must think quickly each time it is their turn to make a move. Each player has a total allotted amount of time to make all his/her moves. Two clocks display the remaining time for each player. While a player is thinking of his/her move, his clock counts down. If time runs out, the other player wins. As soon as a player makes his/her move, he hits a button, which serves to stop counting down his clock and start counting down the other player’s clock.

The program on the next page implements this mechanism. The main program keeps track of the time remaining for each player by decrementing the proper counter once per second while the player is thinking. When a player’s counter reaches zero, a message is printed on the screen declaring the winner. When a player hits the button, an interrupt is taken. The interrupt service routine takes such action as to enable the main program (after returning from the interrupt) to start decrementing the other counter.

The interrupt vector for the button is x35. The priority level of the button is #2. Assume that the operating system has set the Interrupt Enable bit of the button to enable it to interrupt. Assume the main program runs at priority #1 and executes in user mode.

Part a: In order for the interrupt service routine to be executed when the button is pushed, what memory location must contain what value?

Address: \textbf{x0135}  
Value: \textbf{x1550}

Part b: Assume a player hits the button while the instruction at line 16 is being executed. What two values (in hex) will be pushed on the stack?

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{PC} & \textbf{PSR} \\
\hline
\textbf{x3010} & \textbf{x8101} \\
\hline
\end{tabular}
\end{center}

Part c: Fill in the missing instructions in the user program.

Part d: This program has a bug that will only occur if an interrupt is taken at an inappropriate time. Write down the line number of an instruction such that if the button is pressed while that instruction is executing, unintended behavior will result.

Line Number: \textbf{9,10,13,14}  
\textbf{< Any of these received full credit} 

How could we fix this bug?

Use a different register (for example, \textbf{R4}) instead of \textbf{R0} to keep track of the turn.
; Interrupt Service Routine
.ORIG x1550
NOT R0, R0
RTI
.END

; User Program
.ORIG x3000
AND R0, R0, #0 ; Line 1
LD R1, TIME ; Line 2
LD R2, TIME ; Line 3

NEXT

JSR COUNT

ADD R0, R0, #0

BRn P2_DEC ; Line 6

ADD R1, R1, #1 ; Line 7

BRP NEXT

LEA R0, P2WINS ; Line 9
BRnzp END ; Line 10

P2_DEC
ADD R2, R2, #1 ; Line 11

BRP NEXT

LEA R0, P1WINS ; Line 13

END
PUTS ; Line 14
HALT ; Line 15

COUNT
LD R3, SECOND ; Line 16

LOOP
ADD R3, R3, #1 ; Line 17

BRp LOOP ; Line 18

RET

TIME .FILL #300
SECOND .FILL #25000 ; 1 second
P1WINS .STRINGZ "Player 1 Wins."
P2WINS .STRINGZ "Player 2 Wins."
.END
NOTES
B\text{off6} : \text{Base} + \text{SEXT[off6]}
PC\text{off9} : \text{PC} + \text{SEXT[off9]}
PC\text{off11} : \text{PC} + \text{SEXT[off11]}
*OP2 may be SR2 or SEXT[imm5]