CLOSED BOOK, NOTE, CALCULATOR, PHONE, & COMPUTER.

The exam is two-sided and has TWELVE pages, including two blank pages and a copy of the Standard ASCII Table, some Trap Service Routines description and the LC-3 Instruction Set handout on the final page (please feel free to detach this final page, but insert it into your exam when you turn it in).

You are required to present a valid UW-Madison student ID card or other government-issued photo ID to one of the teaching assistants who are proctoring this exam before leaving the room. If you fail to do so, we cannot grade your exam.

Plan your time carefully, since some problems are longer than others.

NAME: ____________________KEY ________________________________

SECTION: _____________________________________________________

ID# ___________________________________________________________

“Green”
<table>
<thead>
<tr>
<th>Problem Number</th>
<th>Maximum Points</th>
<th>Graded By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>SW</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>SB</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>SB</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>PS</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>SW</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>PS</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 (12 points): Short Answers

a. The LC-3 assembly process is done in two complete passes through the entire assembly language program. What is the objective of the second pass?

Generates machine code for each instruction

b. What single instruction is equivalent to the following two LC-3 instructions?

LD R0, FooBar
LDR R0, R0, #0

LDI R0, FooBar

c. What single instruction is equivalent to the following one LC-3 instruction?

RET

JMP R7

d. What is the purpose of .BLKW pseudo-op?

Allocates a block of memory

Problem 2 (8 points): Memory-Mapped I/O

a) An LC-3 instruction loads from the address xFE02. How does the LC-3 know whether to load from KBDR or from memory location xFE02?

All addresses in the range xFE00-FFFF are reserved for I/O. The Address Control Logic knows that the location xFE02 maps to the KBDR.

b) How are the bits in the KBSR defined?

KBSR[15] = is there a new character pressed.
KBSR[14-1] = 0
Problem 3 (20 points): Two-Pass Assembly Process

An assembly language LC-3 program is given below:

```
.ORIG x3000

MAIN
  LEA R0, MSG
  PUTS
  JSR RL
  HALT

RL
  ST R7, RL_RETURN
  LD R3, ENTER ; initialize R3 to 'enter char'
  AND R1, R1, #0
  ADD R1, R1, BUFFER ; initialize R1 to point to the
  ; start of buffer

  LD R0, PROMPT
  OUT ; show prompt

  RL_START
    GETC
    OUT ; read input and echo it back

    NOT R4, R3
    ADD R4, R4, #1
    ADD R4, R0, R4
    BRZ RL_END ; leave if user hits enter

    STR R0, R1, #0
    ADD R1, R1, #1
    BR RL_START ; write char, increment pointer, 
    ; read next char

RL_END
  RET

BUFFER .BLKW x000F
RL_RETURN .FILL x0000
PROMPT .FILL x003E ; '>' character
ENTER .FILL x000A ; 'enter' character
MSG .STRINGZ "Enter input:"

.END
```
a. Fill in the symbol table for the program:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>x3000</td>
</tr>
<tr>
<td>RL</td>
<td>x3004</td>
</tr>
<tr>
<td>RL_START</td>
<td>x300A</td>
</tr>
<tr>
<td>RL_END</td>
<td>x3013</td>
</tr>
<tr>
<td>BUFFER</td>
<td>x3014</td>
</tr>
<tr>
<td>RL_RETURN</td>
<td>x3023</td>
</tr>
<tr>
<td>PROMPT</td>
<td>x3024</td>
</tr>
<tr>
<td>ENTER</td>
<td>x3025</td>
</tr>
<tr>
<td>MSG</td>
<td>x3026</td>
</tr>
</tbody>
</table>

b. Assuming that both passes of the assembler were to execute, write the binary word (machine language instruction) that would be generated by the assembler for the instruction at line 11 of the program.

0010 000 0 0001 1111 = x201F

c. The programmer intended that the RL subroutine reads user input, writes it in BUFFER and returns when user types enter. There are two errors in this subroutine. For each, describe the error and indicate whether it will be detected at assembly time or at run time.

Assembly time error: ADD R1, R1, BUFFER is not valid. It should be LEA R1, BUFFER
Runtime Error: The trap GETC overwrites R7 so subroutine RL doesn't return properly.
Problems 4, 5, 6 make use of the following program

```
.ORIG x3000

0   ST   R0, SAVER0
1   ST   R1, SAVER1
2   JSR  SUBROUTINE1
3   LD   R0, SAVER0
4   LD   R1, SAVER1
5   HALT

6   SUBROUTINE1 ST   R7, SAVER7
7   ST   R2, SAVER2
8   ST   R3, SAVER3
9   CHECKPOINT1 LEA R4, BUFFER
10  LD   R3, DELIM
11  NOT R3, R3
12  ADD R3, R3, #1
13  LOOP_START JSR  SUBROUTINE2
14  ADD R2, R0, R3
15  BRz  LOOP_END
16  STR R0, R4, #0
17  ADD R4, R4, #1
18  BR   LOOP_START
19  LOOP_END AND R0, R0, #0
20  STR R0, R4, #0
21  LD   R2, SAVER2
22  LD   R7, SAVER7
23  LD   R3, SAVER3
24  RET

25  SUBROUTINE2 LDI R1, KBSR
26  BRzp SUBROUTINE2
27  LDI R0, KBDR
28  CHECKPOINT2 RET

29  SAVER1 .FILL x0000
30  SAVER2 .FILL x0000
31  SAVER7 .FILL x0000
32  SAVER0 .FILL x0000
33  SAVER3 .FILL x0000
34  DELIM .FILL x003B
35  KBSR  .FILL xFE00
36  KBDR  .FILL xFE02
37  BUFFER .BLKW x0030

.END
```
Problem 4 (20 points): Traps and Subroutines

a) In the program in page 6, what registers are callee-saved, and what registers are caller-saved?

Caller Saved: R0,R1
Callee Saved: R7,R2,R3

b) Is there a register which cannot be callee-saved? If yes, why not?

R7. There is no point in saving R7 in the callee, since R7 gets overwritten by the JSR instruction.

c) What will be the value in R7:
   1. If you put a breakpoint at Checkpoint1?

      x3003

   2. If you put a breakpoint at Checkpoint2?

      x300E

d) Can interrupts use R7 to hold the return address? If no, why not?

   No. Interrupts can occur at any time, so the programmer cannot save-restore values as could be done in the case of subroutines.
Problem 5 (26 points): Input/Output

a) In the program in page 6, what does the subroutine SUBROUTINE2 do?

Polls the keyboard until it gets a character

b) When does the loop in SUBROUTINE1 terminate?

When the key pressed is ';'

c) What does the subroutine SUBROUTINE1 do?

Reads characters from keyboard and copies it into a buffer, terminates when a ';' is pressed

d) What does this program do?

Reads characters from keyboard and copies it into a buffer

e) Assume that the label BUFFER points to address x3037. If the user types the following sequence:

A B C ; K M \ +

What would be the contents of the following memory locations

<table>
<thead>
<tr>
<th>Address</th>
<th>ASCII value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3037</td>
<td>‘A’</td>
</tr>
<tr>
<td>x3038</td>
<td>‘B’</td>
</tr>
<tr>
<td>x3039</td>
<td>‘C’</td>
</tr>
<tr>
<td>x303A</td>
<td>0</td>
</tr>
</tbody>
</table>
Problem 6 (9 points): Input/Output

a) What is the purpose of the Keyboard Status Register?

The keyboard status registers maintains a flag indicating “has the character in KBDR been read?”. If it's 0, that means the character has already been read, if it's 1 it means the character is new and has not been read.

b) What problem could occur if the keyboard hardware doesn't check the KBSR before writing to the KBDR?

The previously typed value in KBDR will be lost.

c) Circle the correct combination that describes the program on page 6.
   1. Special Opcode for I/O and interrupt driven
   2. Special Opcode for I/O and polling
   3. Memory mapped and interrupt driven
   4. Memory mapped and polling
Scratch Sheet 1 (in case you need additional space for some of your answers)
# ASCII Table

<table>
<thead>
<tr>
<th>Character</th>
<th>Hex</th>
<th>Character</th>
<th>Hex</th>
<th>Character</th>
<th>Hex</th>
<th>Character</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>nul</td>
<td>00</td>
<td>sp</td>
<td>20</td>
<td>@</td>
<td>40</td>
<td>`</td>
<td>60</td>
</tr>
<tr>
<td>soh</td>
<td>01</td>
<td>!</td>
<td>21</td>
<td>A</td>
<td>41</td>
<td>a</td>
<td>61</td>
</tr>
<tr>
<td>stx</td>
<td>02</td>
<td>&quot;</td>
<td>22</td>
<td>B</td>
<td>42</td>
<td>b</td>
<td>62</td>
</tr>
<tr>
<td>etx</td>
<td>03</td>
<td>#</td>
<td>23</td>
<td>C</td>
<td>43</td>
<td>c</td>
<td>63</td>
</tr>
<tr>
<td>eot</td>
<td>04</td>
<td>$</td>
<td>24</td>
<td>D</td>
<td>44</td>
<td>d</td>
<td>64</td>
</tr>
<tr>
<td>enq</td>
<td>05</td>
<td>%</td>
<td>25</td>
<td>E</td>
<td>45</td>
<td>e</td>
<td>65</td>
</tr>
<tr>
<td>ack</td>
<td>06</td>
<td>&amp;</td>
<td>26</td>
<td>F</td>
<td>46</td>
<td>f</td>
<td>66</td>
</tr>
<tr>
<td>bel</td>
<td>07</td>
<td>'</td>
<td>27</td>
<td>G</td>
<td>47</td>
<td>g</td>
<td>67</td>
</tr>
<tr>
<td>bs</td>
<td>08</td>
<td>(</td>
<td>28</td>
<td>H</td>
<td>48</td>
<td>h</td>
<td>68</td>
</tr>
<tr>
<td>ht</td>
<td>09</td>
<td>)</td>
<td>29</td>
<td>I</td>
<td>49</td>
<td>i</td>
<td>69</td>
</tr>
<tr>
<td>lf</td>
<td>0A</td>
<td>*</td>
<td>2A</td>
<td>J</td>
<td>4A</td>
<td>j</td>
<td>6A</td>
</tr>
<tr>
<td>vt</td>
<td>0B</td>
<td>+</td>
<td>2B</td>
<td>K</td>
<td>4B</td>
<td>k</td>
<td>6B</td>
</tr>
<tr>
<td>ff</td>
<td>0C</td>
<td>,</td>
<td>2C</td>
<td>L</td>
<td>4C</td>
<td>l</td>
<td>6C</td>
</tr>
<tr>
<td>cr</td>
<td>0D</td>
<td>-</td>
<td>2D</td>
<td>M</td>
<td>4D</td>
<td>m</td>
<td>6D</td>
</tr>
<tr>
<td>so</td>
<td>0E</td>
<td>.</td>
<td>2E</td>
<td>N</td>
<td>4E</td>
<td>n</td>
<td>6E</td>
</tr>
<tr>
<td>si</td>
<td>0F</td>
<td>/</td>
<td>2F</td>
<td>O</td>
<td>4F</td>
<td>o</td>
<td>6F</td>
</tr>
<tr>
<td>dle</td>
<td>10</td>
<td>0</td>
<td>30</td>
<td>P</td>
<td>50</td>
<td>p</td>
<td>70</td>
</tr>
<tr>
<td>dc1</td>
<td>11</td>
<td>1</td>
<td>31</td>
<td>Q</td>
<td>51</td>
<td>q</td>
<td>71</td>
</tr>
<tr>
<td>dc2</td>
<td>12</td>
<td>2</td>
<td>32</td>
<td>R</td>
<td>52</td>
<td>r</td>
<td>72</td>
</tr>
<tr>
<td>dc3</td>
<td>13</td>
<td>3</td>
<td>33</td>
<td>S</td>
<td>53</td>
<td>s</td>
<td>73</td>
</tr>
<tr>
<td>dc4</td>
<td>14</td>
<td>4</td>
<td>34</td>
<td>T</td>
<td>54</td>
<td>t</td>
<td>74</td>
</tr>
<tr>
<td>nak</td>
<td>15</td>
<td>5</td>
<td>35</td>
<td>U</td>
<td>55</td>
<td>u</td>
<td>75</td>
</tr>
<tr>
<td>syn</td>
<td>16</td>
<td>6</td>
<td>36</td>
<td>V</td>
<td>56</td>
<td>v</td>
<td>76</td>
</tr>
<tr>
<td>etb</td>
<td>17</td>
<td>7</td>
<td>37</td>
<td>W</td>
<td>57</td>
<td>w</td>
<td>77</td>
</tr>
<tr>
<td>can</td>
<td>18</td>
<td>8</td>
<td>38</td>
<td>X</td>
<td>58</td>
<td>x</td>
<td>78</td>
</tr>
<tr>
<td>em</td>
<td>19</td>
<td>9</td>
<td>39</td>
<td>Y</td>
<td>59</td>
<td>y</td>
<td>79</td>
</tr>
<tr>
<td>sub</td>
<td>1A</td>
<td>:</td>
<td>3A</td>
<td>Z</td>
<td>5A</td>
<td>z</td>
<td>7A</td>
</tr>
<tr>
<td>esc</td>
<td>1B</td>
<td>;</td>
<td>3B</td>
<td>[</td>
<td>5B</td>
<td>{</td>
<td>7B</td>
</tr>
<tr>
<td>fs</td>
<td>1C</td>
<td>&lt;</td>
<td>3C</td>
<td>\</td>
<td>5C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gs</td>
<td>1D</td>
<td>=</td>
<td>3D</td>
<td>]</td>
<td>5D</td>
<td>}</td>
<td>7D</td>
</tr>
<tr>
<td>rs</td>
<td>1E</td>
<td>&gt;</td>
<td>3E</td>
<td>^</td>
<td>5E</td>
<td>~</td>
<td>7E</td>
</tr>
<tr>
<td>us</td>
<td>1F</td>
<td>?</td>
<td>3F</td>
<td>_</td>
<td>5F</td>
<td>del</td>
<td>7F</td>
</tr>
</tbody>
</table>

## Trap Service Routines

<table>
<thead>
<tr>
<th>Trap Vector</th>
<th>Assembler Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x20</td>
<td>GETC</td>
<td>Read a single character from the keyboard. The Character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.</td>
</tr>
<tr>
<td>x21</td>
<td>OUT</td>
<td>Write a character in R0[7:0] to the console display.</td>
</tr>
<tr>
<td>x25</td>
<td>HALT</td>
<td>Halt execution and print a message on the console.</td>
</tr>
</tbody>
</table>
LC-3 Instruction Set

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

ADD DR, SR1, SR2; Addition
ADD DR, SR1, imm5; Addition with Immediate
AND DR, SR1, SR2; Bit-wise AND
AND DR, SR1, imm5; Bit-wise AND with Immediate
BRx, label (where x = {n, z, p, np, nz, npz}) ; Branch
if (GO is true) then PC <- PC' + SEXT(PCoffset9)

JMP BaseR ; Jump
JSR label ; Jump to Subroutine
JSRR BaseR ; Jump to Subroutine in Register
LD DR, label ; Load PC-Relative
LDI DR, label ; Load Indirect
LEA, DR, label ; Load Effective Address
NOT DR, SR; Bit-wise Complement
NOT(DR) also setcc()
PC' + SEXT(PCoffset9) also setcc()
mem[BaseR + SEXT(offset6)] also setcc()
mem[mem[PC' + SEXT(PCoffset9)]] also setcc()
mem[BaseR + SEXT((offset6))] also setcc()
mem[BaseR + SEXT(offset6)] also setcc()
mem[BaseR + SEXT(offset6)] also setcc()
NOT DR, SR; Bit-wise Complement
NOT(SR) also setcc()

RET; Return from Subroutine
PC <- R7

RTI; Return from Interrupt
See textbook (2nd Ed. page 537).

ST SR, label ; Store PC-Relative
mem[PC' + SEXT(PCoffset9)] <- SR

STI, SR, label ; Store Indirect
mem[mem[PC' + SEXT(PCoffset9)]] <- SR

STR SR, BaseR, offset6 ; Store Base+Offset
mem[BaseR + SEXT(offset6)] <- SR

TRAP ; System Call
R7 <- PC', PC <- mem[ZEXT(trapvect8)]

; Unused Opcode

SEXT(immediate): sign-extend immediate to 16 bits.
ZEXT(immediate): zero-extend immediate to 16 bits.

Page 2 has an ASCII character table.