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): Construct the symbol table for the following LC-3 assembly language program:

Symbol Table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ORIG x4500</td>
<td></td>
</tr>
<tr>
<td>LD R2, BOBO</td>
<td></td>
</tr>
<tr>
<td>LD R3, SAM</td>
<td></td>
</tr>
<tr>
<td>AGAIN ADD R3, R3, R2</td>
<td></td>
</tr>
<tr>
<td>ADD R2, R2, #-1</td>
<td></td>
</tr>
<tr>
<td>BRnzp SAM</td>
<td></td>
</tr>
<tr>
<td>BOBO .STRINGZ &quot;Why are you asking me this?&quot;</td>
<td></td>
</tr>
<tr>
<td>SAM BRnp AGAIN</td>
<td></td>
</tr>
<tr>
<td>TRAP x25</td>
<td></td>
</tr>
<tr>
<td>.BLKW 5</td>
<td></td>
</tr>
<tr>
<td>JOE .FILL x7777</td>
<td></td>
</tr>
<tr>
<td>.END</td>
<td></td>
</tr>
</tbody>
</table>

Part b. (5 points): A stack machine executes the following 6 instructions:

Push 5
Push 4
ADD
Push 6
MUL
POP

What value is popped by the last instruction?

Assume the stack is empty, with R6 = xFE00. Before the stack machine executes "Push 5", the contents of memory locations xFDFA to xFDFF are shown. Show the contents of memory and R6 after the six operations above are executed.

<table>
<thead>
<tr>
<th>Address</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
<tr>
<td>xFDFA</td>
<td>x0000</td>
<td></td>
</tr>
</tbody>
</table>

R6:
**Part c.** (5 points): We want to move a number from A to B. List all LC-3 opcodes that can be used to accomplish this in one instruction when A,B are as specified at the top of each column. We have provided four slots for each column. Use as many as you need.

<table>
<thead>
<tr>
<th>A is memory location</th>
<th>A is a register R0-R7</th>
<th>A is a register R0-R7</th>
<th>A is memory location</th>
</tr>
</thead>
<tbody>
<tr>
<td>B is a register R0-R7</td>
<td>B is a register R0-R7</td>
<td>B is memory location</td>
<td>B is memory location</td>
</tr>
</tbody>
</table>

| | | | |
| | | | |
| | | | |

**Part d.** (5 points): What is wrong with the following program fragment?

```plaintext
... ... 
... 
LD R0, A
SPIN LDI R1,KBSR
BRzp SPIN
STI R0,KBDR
... ... 
RET
KBSR .FILL xFE00
KBDR .FILL xFE02
A .FILL x0041
... ... 
```
Problem 2. (20 points): Since ASCII codes consist of 8 bits each, we can store two ASCII codes in one word of LC-3 memory. If a user types $2n$ characters on the keyboard, followed by the <ENTER> key, the subroutine PACK on the next page will store the corresponding ASCII codes into $n$ sequential memory locations, two per memory location, starting at location A.

You may assume that a user never enters an odd number of characters.

Part a. (15 points): Your job: Fill in the blanks in the program.

Part b. (5 points): If a user types the string Please help! followed by the <ENTER> key, what does the program do?
.ORIG x7020

PACK
ST R7, SAVER7
ST R6, SAVER6
ST R4, SAVER4
ST R3, SAVER3
LEA R6, A ; R6 is the pointer
AND R4, R4, #0
ADD R4, R4, #8 ; R4 is our counter
AND R3, R3, #0
LEA R0, PROMPT
TRAP x22

POLL
BRzp POLL

LD R0, NEG_LF
ADD R0, R7, R0

ADD R4, R4, #0
BRz NOSHIFT

SHIFT
ADD R7, R7, R7
ADD R4, R4, #-1
BRp SHIFT
ADD R3, R7, #0
BRnzp POLL

NOSHIFT
ADD R3, R3, R7

ADD R6, R6, #1
ADD R4, R4, #8
BRnzp POLL

DONE
LD R7, SAVER7
LD R6, SAVER6
LD R4, SAVER4
LD R3, SAVER3
LEA R0, A ; Returns a pointer to the characters
RET

KBSR .FILL xFE00
KBDR .FILL xFE02
NEG_LF .FILL xFFF6
PROMPT .STRINGZ "Please enter a string: "
A .BLKW #5
SAVER7 .BLKW #1
SAVER6 .BLKW #1
SAVER4 .BLKW #1
SAVER3 .BLKW #1
.END
Problem 3. (20 points): Many cities, like New York City, Stockholm, Konigsberg, etc. consist of several areas, connected by bridges. The figure below shows a map of FiveParts, a city made up of five areas A, B, C, D, E, with the areas connected by 9 bridges as shown.

The following program prompts the user to enter two areas, and then stores the number of bridges from the first area to the second in location x4500. Your job: On the next page, design the data structure for the city of FiveParts that the program below will use to count the number of bridges between two areas.

```
.ORIG x3000
LEA R0, FROM
TRAP x22
TRAP x20        ; Inputs a char without banner
   NOT R1, R0
   ADD R1, R1, #1
   LEA R0, TO
   TRAP x22
   TRAP x20
   NOT R0, R0
   ADD R0, R0, #1
   AND R5, R5, #0
   LDI R2, HEAD
SEARCH   BRz DONE
   LDR R3, R2, #0
   ADD R7, R1, R3
   BRz FOUND_FROM
   LDR R2, R2, #1
   BRnp SKIP
FOUND_FROM ADD R2, R2, #2
NEXT_BRIDGE LDR R3, R2, #0
   BRz DONE
   LDR R4, R3, #0
   ADD R7, R0, R4
   BRnp SKIP
   ADD R5, R5, #1 ; Increment Counter
SKIP     ADD R2, R2, #1
DONE     STI R5, ANSWER
          HALT
HEAD     .FILL x3050
ANSWER    .FILL x4500
FROM      .STRINGZ "FROM: 
TO        .STRINGZ "TO: 
.END
```
Your job is to provide the contents of the memory locations that are needed to specify the data structure for the city of FiveParts, which is needed by the program on the previous page. We have given you the HEAD pointer for the data structure and in addition, five memory locations and the contents of those five locations. We have also supplied more than enough sequential memory locations after each of the five to enable you to finish the job. Use as many of these memory locations as you need.
Problem 4. (20 points): We wish to use the unused opcode 1101 to add a new instruction to the LC-3 ISA. This requires four new states in the state machine (shown below) and additions to the data path (shown on the next page).

**Part a.** (5 points): Fill in the missing information in the four states. You can assume all control signals not shown are 0. A table of relevant control signals is included in your exam packet.

**Part b.** (10 points): What does the new instruction do (in 15 words or fewer)?

**Part c.** (5 points): Identify the fields of the new instruction. Be sure you indicate clearly the correct bits for each field.
Problem 5. (20 points): Information about members of an extended family is stored in a tree. The first two words in each node are pointers to the oldest child and the next younger sibling. That is, the children of a parent are ordered according to age. We use the 3rd word in each node to represent the year the person was born.
Part a. (15 points): The following recursive subroutine counts the number of family members who are born before 1960. R0 is a pointer to the root of a tree. R1 is the output count. Assume the main program initializes R1 to 0 and R6 to the stack pointer before calling the subroutine. The stack does not overflow during the execution of the subroutine. Fill in the missing instructions.

```
.COUNT .ORG x4000
ADD R6, R6, #1
ADD R6, R6, #1
ADD R0, R0, #0
LD R7, NEG_VALUE
ADD R7, R2, R7
BRzp SKIP
ADD R1, R1, #1
ADD R2, R0, #0
LDR R0, R2, #0
ADD R0, R2, #0
ADD R0, R2, #0
ADD R2, R6, #0
ADD R6, R6, #1
ADD R6, R6, #1
RESTORE
ADD R6, R6, #1
RET
NEG_VALUE .FILL #1960
```

Part b. (5 points): Can we speed up the subroutine by eliminating visits to unnecessary nodes in the tree? How (in 20 words or fewer)?
## Data Path Control Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD.MAR/1</td>
<td>NO(0), LOAD(1)</td>
</tr>
<tr>
<td>LD.MDR/1</td>
<td>NO(0), LOAD(1)</td>
</tr>
<tr>
<td>LD.IR/1</td>
<td>NO(0), LOAD(1)</td>
</tr>
<tr>
<td>LD.REG/1</td>
<td>NO(0), LOAD(1)</td>
</tr>
<tr>
<td>LD.CC/1</td>
<td>NO(0), LOAD(1)</td>
</tr>
<tr>
<td><strong>LD.TEMP/1</strong></td>
<td><strong>NO(0), LOAD(1)</strong></td>
</tr>
<tr>
<td>Gate.MARMUX/1</td>
<td>NO(0), YES(1)</td>
</tr>
<tr>
<td>Gate.MDR/1</td>
<td>NO(0), YES(1)</td>
</tr>
<tr>
<td>Gate.PC/1</td>
<td>NO(0), YES(1)</td>
</tr>
<tr>
<td>Gate.ALU/1</td>
<td>NO(0), YES(1)</td>
</tr>
<tr>
<td>ADDR1MUX/1</td>
<td>PC(0), BaseR(1)</td>
</tr>
<tr>
<td>ADDR2MUX/2</td>
<td>ZERO(00), offset6(01), PCoffset9(10), PCoffset11(11)</td>
</tr>
<tr>
<td>MARMUX/1</td>
<td>IR7.0(0), ADDER(1)</td>
</tr>
<tr>
<td>SR1MUX/2</td>
<td>11.9(00), 8.6(01), SP(10)</td>
</tr>
<tr>
<td><strong>TEMPMUX/1</strong></td>
<td><strong>OP2(0), TEMP(1)</strong></td>
</tr>
<tr>
<td>ALUK/2</td>
<td>ADD(00), AND(01), NOT(10), PASSA(11)</td>
</tr>
<tr>
<td>MIO.EN/1</td>
<td>NO(0), YES(1)</td>
</tr>
<tr>
<td>R.W/1</td>
<td>RD(0), WR(1)</td>
</tr>
<tr>
<td>Instruction</td>
<td>Opcode</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>ADD+</td>
<td>0001</td>
</tr>
<tr>
<td>ADD+</td>
<td>0001</td>
</tr>
<tr>
<td>AND+</td>
<td>0101</td>
</tr>
<tr>
<td>AND+</td>
<td>0101</td>
</tr>
<tr>
<td>BR</td>
<td>0000</td>
</tr>
<tr>
<td>JMP</td>
<td>1100</td>
</tr>
<tr>
<td>JSR</td>
<td>0100</td>
</tr>
<tr>
<td>JSRR</td>
<td>0100</td>
</tr>
<tr>
<td>LD+</td>
<td>0010</td>
</tr>
<tr>
<td>LDI+</td>
<td>1010</td>
</tr>
<tr>
<td>LDR+</td>
<td>0110</td>
</tr>
<tr>
<td>LEA</td>
<td>1110</td>
</tr>
<tr>
<td>NOT+</td>
<td>1001</td>
</tr>
<tr>
<td>RET</td>
<td>1100</td>
</tr>
<tr>
<td>RTI</td>
<td>1000</td>
</tr>
<tr>
<td>ST</td>
<td>0011</td>
</tr>
<tr>
<td>STI</td>
<td>1011</td>
</tr>
<tr>
<td>STR</td>
<td>0111</td>
</tr>
<tr>
<td>TRAP</td>
<td>1111</td>
</tr>
</tbody>
</table>

Figure A.2  Format of the entire LC-3 instruction set. Note: + indicates instructions that modify condition codes
<table>
<thead>
<tr>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>nul</td>
<td>0</td>
<td>00</td>
<td>sp</td>
<td>32</td>
<td>20</td>
<td>@</td>
<td>64</td>
<td>40</td>
<td>'</td>
<td>96</td>
<td>60</td>
</tr>
<tr>
<td>soh</td>
<td>1</td>
<td>01</td>
<td>!</td>
<td>33</td>
<td>21</td>
<td>A</td>
<td>65</td>
<td>41</td>
<td>a</td>
<td>97</td>
<td>61</td>
</tr>
<tr>
<td>stx</td>
<td>2</td>
<td>02</td>
<td>&quot;</td>
<td>34</td>
<td>22</td>
<td>B</td>
<td>66</td>
<td>42</td>
<td>b</td>
<td>98</td>
<td>62</td>
</tr>
<tr>
<td>etx</td>
<td>3</td>
<td>03</td>
<td>#</td>
<td>35</td>
<td>23</td>
<td>C</td>
<td>67</td>
<td>43</td>
<td>c</td>
<td>99</td>
<td>63</td>
</tr>
<tr>
<td>eot</td>
<td>4</td>
<td>04</td>
<td>$</td>
<td>36</td>
<td>24</td>
<td>D</td>
<td>68</td>
<td>44</td>
<td>d</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>enq</td>
<td>5</td>
<td>05</td>
<td>%</td>
<td>37</td>
<td>25</td>
<td>E</td>
<td>69</td>
<td>45</td>
<td>e</td>
<td>101</td>
<td>65</td>
</tr>
<tr>
<td>ack</td>
<td>6</td>
<td>06</td>
<td>&amp;</td>
<td>38</td>
<td>26</td>
<td>F</td>
<td>70</td>
<td>46</td>
<td>f</td>
<td>102</td>
<td>66</td>
</tr>
<tr>
<td>bel</td>
<td>7</td>
<td>07</td>
<td>'</td>
<td>39</td>
<td>27</td>
<td>G</td>
<td>71</td>
<td>47</td>
<td>g</td>
<td>103</td>
<td>67</td>
</tr>
<tr>
<td>bs</td>
<td>8</td>
<td>08</td>
<td>(</td>
<td>40</td>
<td>28</td>
<td>H</td>
<td>72</td>
<td>48</td>
<td>h</td>
<td>104</td>
<td>68</td>
</tr>
<tr>
<td>ht</td>
<td>9</td>
<td>09</td>
<td>)</td>
<td>41</td>
<td>29</td>
<td>I</td>
<td>73</td>
<td>49</td>
<td>i</td>
<td>105</td>
<td>69</td>
</tr>
<tr>
<td>lf</td>
<td>10</td>
<td>0A</td>
<td>+</td>
<td>42</td>
<td>2A</td>
<td>J</td>
<td>74</td>
<td>4A</td>
<td>j</td>
<td>106</td>
<td>6A</td>
</tr>
<tr>
<td>vt</td>
<td>11</td>
<td>0B</td>
<td>*</td>
<td>43</td>
<td>2B</td>
<td>K</td>
<td>75</td>
<td>4B</td>
<td>k</td>
<td>107</td>
<td>6B</td>
</tr>
<tr>
<td>ff</td>
<td>12</td>
<td>0C</td>
<td>-</td>
<td>44</td>
<td>2C</td>
<td>L</td>
<td>76</td>
<td>4C</td>
<td>l</td>
<td>108</td>
<td>6C</td>
</tr>
<tr>
<td>cr</td>
<td>13</td>
<td>0D</td>
<td>.</td>
<td>45</td>
<td>2D</td>
<td>M</td>
<td>77</td>
<td>4D</td>
<td>m</td>
<td>109</td>
<td>6D</td>
</tr>
<tr>
<td>so</td>
<td>14</td>
<td>0E</td>
<td>/</td>
<td>46</td>
<td>2E</td>
<td>N</td>
<td>78</td>
<td>4E</td>
<td>n</td>
<td>110</td>
<td>6E</td>
</tr>
<tr>
<td>si</td>
<td>15</td>
<td>0F</td>
<td>0</td>
<td>47</td>
<td>2F</td>
<td>O</td>
<td>79</td>
<td>4F</td>
<td>o</td>
<td>111</td>
<td>6F</td>
</tr>
<tr>
<td>dle</td>
<td>16</td>
<td>10</td>
<td>\</td>
<td>48</td>
<td>30</td>
<td>P</td>
<td>80</td>
<td>50</td>
<td>p</td>
<td>112</td>
<td>70</td>
</tr>
<tr>
<td>dc1</td>
<td>17</td>
<td>11</td>
<td>1</td>
<td>49</td>
<td>31</td>
<td>Q</td>
<td>81</td>
<td>51</td>
<td>q</td>
<td>113</td>
<td>71</td>
</tr>
<tr>
<td>dc2</td>
<td>18</td>
<td>12</td>
<td>2</td>
<td>50</td>
<td>32</td>
<td>R</td>
<td>82</td>
<td>52</td>
<td>r</td>
<td>114</td>
<td>72</td>
</tr>
<tr>
<td>dc3</td>
<td>19</td>
<td>13</td>
<td>3</td>
<td>51</td>
<td>33</td>
<td>S</td>
<td>83</td>
<td>53</td>
<td>s</td>
<td>115</td>
<td>73</td>
</tr>
<tr>
<td>dc4</td>
<td>20</td>
<td>14</td>
<td>4</td>
<td>52</td>
<td>34</td>
<td>T</td>
<td>84</td>
<td>54</td>
<td>t</td>
<td>116</td>
<td>74</td>
</tr>
<tr>
<td>nak</td>
<td>21</td>
<td>15</td>
<td>5</td>
<td>53</td>
<td>35</td>
<td>U</td>
<td>85</td>
<td>55</td>
<td>u</td>
<td>117</td>
<td>75</td>
</tr>
<tr>
<td>syn</td>
<td>22</td>
<td>16</td>
<td>6</td>
<td>54</td>
<td>36</td>
<td>V</td>
<td>86</td>
<td>56</td>
<td>v</td>
<td>118</td>
<td>76</td>
</tr>
<tr>
<td>etb</td>
<td>23</td>
<td>17</td>
<td>7</td>
<td>55</td>
<td>37</td>
<td>W</td>
<td>87</td>
<td>57</td>
<td>w</td>
<td>119</td>
<td>77</td>
</tr>
<tr>
<td>can</td>
<td>24</td>
<td>18</td>
<td>8</td>
<td>56</td>
<td>38</td>
<td>X</td>
<td>88</td>
<td>58</td>
<td>x</td>
<td>120</td>
<td>78</td>
</tr>
<tr>
<td>em</td>
<td>25</td>
<td>19</td>
<td>9</td>
<td>57</td>
<td>39</td>
<td>Y</td>
<td>89</td>
<td>59</td>
<td>y</td>
<td>121</td>
<td>79</td>
</tr>
<tr>
<td>sub</td>
<td>26</td>
<td>1A</td>
<td>:</td>
<td>58</td>
<td>3A</td>
<td>Z</td>
<td>90</td>
<td>5A</td>
<td>z</td>
<td>122</td>
<td>7A</td>
</tr>
<tr>
<td>esc</td>
<td>27</td>
<td>1B</td>
<td>;</td>
<td>59</td>
<td>3B</td>
<td>[</td>
<td>91</td>
<td>5B</td>
<td>{</td>
<td>123</td>
<td>7B</td>
</tr>
<tr>
<td>fs</td>
<td>28</td>
<td>1C</td>
<td>&lt;</td>
<td>60</td>
<td>3C</td>
<td>\</td>
<td>92</td>
<td>5C</td>
<td></td>
<td></td>
<td>124</td>
</tr>
<tr>
<td>gs</td>
<td>29</td>
<td>1D</td>
<td>=</td>
<td>61</td>
<td>3D</td>
<td>]</td>
<td>93</td>
<td>5D</td>
<td>}</td>
<td>125</td>
<td>7D</td>
</tr>
<tr>
<td>rs</td>
<td>30</td>
<td>1E</td>
<td>&gt;</td>
<td>62</td>
<td>3E</td>
<td>^</td>
<td>94</td>
<td>5E</td>
<td>~</td>
<td>126</td>
<td>7E</td>
</tr>
<tr>
<td>us</td>
<td>31</td>
<td>1F</td>
<td>?</td>
<td>63</td>
<td>3F</td>
<td>del</td>
<td>95</td>
<td>5F</td>
<td>del</td>
<td>127</td>
<td>7F</td>
</tr>
</tbody>
</table>
### Table A.2  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>x22</td>
<td>PUTS</td>
<td>Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. Writing terminates with the occurrence of x0000 in a memory location.</td>
</tr>
<tr>
<td>x23</td>
<td>IN</td>
<td>Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console monitor, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.</td>
</tr>
<tr>
<td>x24</td>
<td>PUTSP</td>
<td>Write a string of ASCII characters to the console. The characters are contained in consecutive memory locations, two characters per memory location, starting with the address specified in R0. The ASCII code contained in bits [7:0] of a memory location is written to the console first. Then the ASCII code contained in bits [15:8] of that memory location is written to the console. (A character string consisting of an odd number of characters to be written will have x00 in bits [15:8] of the memory location containing the last character to be written.) Writing terminates with the occurrence of x0000 in a memory location.</td>
</tr>
<tr>
<td>x25</td>
<td>HALT</td>
<td>Halt execution and print a message on the console.</td>
</tr>
</tbody>
</table>

### Table A.3  Device Register Assignments

<table>
<thead>
<tr>
<th>Address</th>
<th>I/O Register Name</th>
<th>I/O Register Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xFE00</td>
<td>Keyboard status register</td>
<td>Also known as KBSR. The ready bit (bit [15]) indicates if the keyboard has received a new character.</td>
</tr>
<tr>
<td>xFE02</td>
<td>Keyboard data register</td>
<td>Also known as KBDR. Bits [7:0] contain the last character typed on the keyboard.</td>
</tr>
<tr>
<td>xFE04</td>
<td>Display status register</td>
<td>Also known as DSR. The ready bit (bit [15]) indicates if the display device is ready to receive another character to print on the screen.</td>
</tr>
<tr>
<td>xFE06</td>
<td>Display data register</td>
<td>Also known as DDR. A character written in the low byte of this register will be displayed on the screen.</td>
</tr>
<tr>
<td>xFFE</td>
<td>Machine control register</td>
<td>Also known as MCR. Bit [15] is the clock enable bit. When cleared, instruction processing stops.</td>
</tr>
</tbody>
</table>