EE 306 Fall 2002
Yale Patt, Instructor
TAs: Asad Bawa, Linda Bigelow, Mustafa Erwa, Lester Guillory, Kevin Major,
Moinuddin Qureshi, Paroma Sen, Santhosh Srinath, Matt Starolis, David Thompson,
Vikrant Venkateshwar

Exam 2, November 20, 2002

Name (1 point)_________________________________________________

TA Name (1 point)______________________________________________

Problem 1 (18 points) :_________________
Problem 2 (15 points) :_________________
Problem 3 (10 points) :_________________
Problem 4 (10 points) :_________________
Problem 5 (10 points) :_________________
Problem 6 (15 points) :_________________
Problem 7 (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. 5 points will be
deducted from the final grade for each page on which your name does not appear.

GOOD LUCK!
Problem 1 (18 points):

Part 1 (6 points):
We have discussed in class two common ways to terminate a loop. One way uses a counter to keep track of the number of iterations.

The other way uses an element called a __________________. The distinguishing characteristic of this element is (in ten words max):

Part 2 (6 points):
Recall that in class two weeks ago, a student noticed that the RET instruction is simply a special case of the JSRR instruction with the base register R7 and the offset #0. Thus, we can throw out the RET opcode as unnecessary. Several opcodes have been suggested as useful replacements:

a. MOVE Ri,Rj ; The contents of Rj are copied into Ri.
b. NAND Ri,Rj,Rk ; Ri is the bit-wise NAND of Rj,Rk
c. SHFL Ri,Rj,#2 ; The contents of Rj are shifted left 2 bits and stored into Ri.
d. MUL Ri,Rj,Rk ; Ri is the product of 2's complement integers in Rj,Rk.

Of the four instructions, which does it make the most sense to add to the LC-2 ISA if we remove RET? Justify your answer.
**Part 3 (6 points):**
It is also the case that we REALLY don't need to have LDI and STI instructions. We can accomplish the same results using other instruction sequences instead of the LDI or STI. Replace the STI instruction in the code on the left with whatever instructions are necessary to perform the same function in the code on the right.

<table>
<thead>
<tr>
<th>With STI</th>
<th>Without STI</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ORIG x3000</td>
<td>.ORIG x3000</td>
</tr>
<tr>
<td>LD R0, CONST</td>
<td>LD R0, CONST</td>
</tr>
<tr>
<td>STI R0, B</td>
<td></td>
</tr>
<tr>
<td>TRAP x25</td>
<td>TRAP x25</td>
</tr>
<tr>
<td>CONST .FILL x0048</td>
<td>CONST .FILL x0048</td>
</tr>
<tr>
<td>B .FILL xF3FF</td>
<td>B .FILL xF3FF</td>
</tr>
<tr>
<td>.END</td>
<td>.END</td>
</tr>
</tbody>
</table>
Problem 2 (15 points):

Our assembler has crashed and we need your help! Complete the symbol table and assemble the instructions at labels D, E, and F in the space provided. You may assume another module deposits a positive value into A before this module executes.

```
.ORIG     x3000
AND      R0, R0, #0
D
LD       R1, A
AND      R2, R1, #1
BRp      B
E
ADD      R1, R1, #-1
B
ADD      R0, R0, R1
ADD      R1, R1, #-2
F
BRp      B
ST       R0, C
TRAP     x25
A  .BLKW 1
C  .BLKW 1
.END
```

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>MACHINE CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

In fifteen words or less, what does the above program do?
Problem 3 (10 points):

The following program is assembled and executed. There are no assemble time nor run
time errors. What is written to the screen? Assume all registers are initialized to 0 before
the program executes. Recall TRAP x22 prints a character string to the screen.

```
.ORIG x3000
ST R0, x3007
LEA R0, LABEL
TRAP x22
TRAP x25
LABEL .STRINGZ "FUNKY"
LABEL2 .STRINGZ "HELLO WORLD"
.END
```
Problem 4 (10 points):

An engineer is in the process of debugging a program she has written. She is looking at the following segment of the program, and decides to place a breakpoint in memory at location 0xA404. Starting with the PC = 0xA400, she initializes all the registers to zero and runs the program until the breakpoint is encountered.

Code Segment:

```
0xA400    THIS1    LEA    R0, THIS1
0xA401    THIS2    LD     R1, THIS2
0xA402    THIS3    LDI    R2, THIS5
0xA403    THIS4    LDR    R3, R0, #2
0xA404    THIS5    .FILL  xA400
```

Show the contents of the register file (in hexadecimal) when the breakpoint is encountered.

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
</tr>
</tbody>
</table>
Problem 5 (10 points):

The following program adds the values stored in memory locations A, B, and C, and stores the result into memory. The code was written by a student who decided not to take EE 306! There are two errors in the code. For each, describe the error and indicate whether it will be detected at assembly time or at run time.

```
.ORG x3000
ONE LD R0, A
ADD R1, R1, R0
TWO LD R0, B
ADD R1, R1, R0
THREE LD R0, C
ADD R1, R1, R0
ST R1, SUM
TRAP x25
A .FILL x0001
B .FILL x0002
C .FILL x0003
D .FILL x0004
.END
```

<table>
<thead>
<tr>
<th>Line No.</th>
<th>Assemble Time</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>___</td>
<td>_____________</td>
<td>___</td>
</tr>
</tbody>
</table>

Error: ____________________________________________

Line No._____  □  Assemble Time  □  Run Time
Error: ____________________________________________
Problem 6 (15 points):

As you know, Push and Pop are two stack operations. Push Rn pushes the value in Register n onto the stack. Pop Rn removes a value from the stack and loads it into Rn. Below is a snapshot of the eight registers of the LC-2 BEFORE and AFTER the following six stack operations are performed. Note that four of the six operations are not completely specified. Fill in the four blanks with the proper register numbers.

PUSH R4
PUSH ______
POP ______
PUSH ______
POP R2
POP ______

BEFORE

<table>
<thead>
<tr>
<th>R0</th>
<th>x0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>x1111</td>
</tr>
<tr>
<td>R2</td>
<td>x2222</td>
</tr>
<tr>
<td>R3</td>
<td>x3333</td>
</tr>
<tr>
<td>R4</td>
<td>x4444</td>
</tr>
<tr>
<td>R5</td>
<td>x5555</td>
</tr>
<tr>
<td>R6</td>
<td>x6666</td>
</tr>
<tr>
<td>R7</td>
<td>x7777</td>
</tr>
</tbody>
</table>

AFTER

<table>
<thead>
<tr>
<th>R0</th>
<th>x1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>x1111</td>
</tr>
<tr>
<td>R2</td>
<td>x3333</td>
</tr>
<tr>
<td>R3</td>
<td>x3333</td>
</tr>
<tr>
<td>R4</td>
<td>x4444</td>
</tr>
<tr>
<td>R5</td>
<td>x5555</td>
</tr>
<tr>
<td>R6</td>
<td>x6666</td>
</tr>
<tr>
<td>R7</td>
<td>x4444</td>
</tr>
</tbody>
</table>
Problem 7 (20 points):

Yikes! The code below is missing some important instructions! When completed correctly, the program should print the following to the monitor:

ABCFGH

Fill in the missing instructions so the program may once again work as originally intended. Each blank box is provided for one missing instruction. Note: those instructions which are present are all correct and do not contain any errors.

.ORIG x3000
LEA R1, TESTOUT
BACK_1
LDR R0, R1, #0
BRz NEXT_1
TRAP x21
BRnzp BACK_1
;
NEXT_1 LEA R1, TESTOUT
BACK_2 LDR R0, R1, #0
BRz NEXT_2
JSR SUB_1
ADD R1, R1, #1
BRnzp BACK_2
;
NEXT_2
SUB_1
LDI R2, CRTSR
STI R0, CRTDR
RET
CRTSR .FILL xF3FC
CRTDR .FILL xF3FF
TESTOUT .STRINGZ "ABC"
.END