CS/ECE 252: INTRODUCTION TO COMPUTER ENGINEERING

UNIVERSITY OF WISCONSIN—MADISON

Prof. Mark D. Hill

TAs: Sujith Surendran, Lisa Ossian

Midterm Examination 4
In Class (50 minutes)
Wednesday, May 6, 2015
Weight: 17.5%

NO: BOOK(S), NOTE(S), OR CALCULATORS OF ANY SORT.

The exam has ten pages. Circle your final answers. Plan your time carefully since some problems are longer than others. You must turn in the pages 1-8. Use the blank sides of the exam for scratch work.

Note: LC-3 instruction set is provided on Page 9. Trap Codes and Assembler directives are provided on page 10

LAST NAME: ___________________________________________________________

FIRST NAME: ___________________________________________________________

ID#: _________________________________________________________________
<table>
<thead>
<tr>
<th>Problem</th>
<th>Maximum Points</th>
<th>Points Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1: Short answer questions (10 points)

a) (1 point) How many accesses to memory are made after the instruction fetch phase of a LDI instruction? Show your work.

b) (1 point) For rare events, would you prefer interrupt-driven I/O or polling I/O? Justify your answer.

c) (1 point) Briefly explain the difference between asynchronous and synchronous I/O events.

d) (2 points) Briefly describe what happens during the linking and loading phases of an assembly program?
e) (2 points) An LC-3 assembly program contains the following instruction:

\[
\text{MAIN} \quad \text{LD R5, MAIN}
\]

The symbol table entry for MAIN is \(x5000\). What will be the value of R5 after the execution of the above instruction? Show your work.

f. (3 points) Identify three assembly errors in the following code:

\[
.\text{ORIG \ x3000}
\]

.\text{LEA \ R1, NUMBER}
.\text{LD \ R1, NUMBER}
.\text{LOOP NOT R5, \#2}
.\text{TRAP \ x29}
.\text{BRzp \ LOOP2}
.\text{AND \ R1, R1, FIVE}
.\text{LD \ R1, FIVE}
.\text{BRp \ LOOP}

.\text{LOOP \ HALT}
.\text{FIVE \ .FILL \ \#5}
.\text{NUMBER \ .FILL \ x60}
.\text{END}
Problem 2: Two-pass assembly process (8 points)

a) (3 points) Consider the following LC-3 assembly program.

```assembly
.ORIG x3000
LEA R2, STRING
LD R3, NUMBER
HERE ADD R1, R2, R3
ADD R2, R1, #0
LDR R0, R1, #0
BRz DONE
OUT
BR HERE

THIS .BLKW 5
STRING .STRINGZ "2down_3to_go"
NUMBER .FILL x4

DONE HALT

.END
```

What would be the output on the console if you run the above code in Pennsim?

b) (3 points) In the first pass, the assembler creates the symbol table. Fill in the symbol table created by the assembler for this program

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
c) (2 points) In the second pass, the assembler creates a binary version (.obj) of the program, using the entries from the symbol table shown below. Given that the following symbol table entries were generated in the first pass of assembly (for another program), fill in the binary code generated by the assembler for the two instructions located at x3000 and x3001.

Symbol Table:

<table>
<thead>
<tr>
<th>Label</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>x3015</td>
</tr>
<tr>
<td>NEXT</td>
<td>x3016</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly code</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3000</td>
<td>LD R0, ADDRESS</td>
<td></td>
</tr>
<tr>
<td>x3001</td>
<td>BRnp NEXT</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3

Consider the program below, the goal of which is to multiply the value in memory location corresponding to label Input1 with the value in memory location corresponding to label Input2 and store the result in the memory location corresponding to label RESULT.

```
.ORIG x3000
LD R2, ZERO
LD R0, Input1
LD R1, Input2

LOOP  BRn DONE
       ADD R2, R2, R0
       ADD R1, R1, -1
       BR LOOP

DONE  ST R2, RESULT
       HALT

RESULT .FILL x0000
ZERO    .FILL x0000
Input1  .FILL x0007
Input2  .FILL x0002
.END
```

a. **(2 points)** What is the value at RESULT after executing the HALT instruction? Write the answer in hexadecimal. Show your work.

b. **(2 points)** From your answer from 3a, you would have noticed that value at RESULT is not equal to value at Input1 * value at Input2. Identify what caused this error, and how do you fix it?
Problem 4: Traps and Subroutines

Suppose we want to write a new TRAP subroutine, TRAP x02. This subroutine takes an input from the caller of the subroutine through register R2. R2 has the memory address of the first character of a string. The subroutine then prints all characters that are not ‘a’. Fill in the missing blanks to complete this subroutine code. Assume that we are implementing a callee-save subroutine. Save only those registers that are necessary.

Assume that the trap vector table (also known as the system control block) is shown below:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0001</td>
<td>x2400</td>
</tr>
<tr>
<td>x0002</td>
<td>x2500</td>
</tr>
<tr>
<td>x0003</td>
<td>x2600</td>
</tr>
</tbody>
</table>

```
.ORIG _____

STORE ST __, SAVEREGR1
      ST __, SAVEREGR2
      ST __, SAVEREGR3
      ST __, SAVEREGR4

LOOP   LDR R0, R2, #0 ;Load a character from the string.
       _______________;If there are no more characters, goto RESTORE
       LD R5, neg_a   ;Load negative of ASCII of ‘a’ into R5.
       ADD R2, R2, #1 ;Increment pointer to get next character.
       _______________;Determine if current character equals ‘a’.
       BR z LOOP      ;If character is ‘a’, go load next character.
       _______________;Print the extracted character.
       BR LOOP        ;Branch to LOOP.

RESTORE LD __, SAVEREGR1
        LD __, SAVEREGR2
        LD __, SAVEREGR3
        LD __, SAVEREGR4
        RET

SAVEREGR1 .BLKW 1
SAVEREGR2 .BLKW 1
SAVEREGR3 .BLKW 1
SAVEREGR4 .BLKW 1
neg_a     .FILL 0xFF9D ;This is the negative of ASCII of ‘a’
.END
```
The following code segment should display the string specified at the “STRING” label on to the console. Write the missing assembly instructions of the program (without using PUTS/PUTC/OUT/TRAP instructions).

Hint: Make use of the DSR and DDR, as shown in the figure below.

```
.ORIG x3000

LEA R3, STRING
NEXT  LDR R0, R3, #0

ADD R3, R3, #1 ; Point to the next character
BR NEXT

END  HALT

STRING .STRINGZ "Enjoy your holidays!" ; String to display
DSR .FILL xFE04 ; Display status register location
DDR .FILL xFE06 ; Display data register location

.END
```
LC 3 Instruction Set to be provided here
TRAP CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALT</td>
<td>TRAP x25</td>
<td>Halt execution and print message to console.</td>
</tr>
<tr>
<td>IN</td>
<td>TRAP x23</td>
<td>Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0].</td>
</tr>
<tr>
<td>OUT</td>
<td>TRAP x21</td>
<td>Write one character (in R0[7:0]) to console.</td>
</tr>
<tr>
<td>GETC</td>
<td>TRAP x20</td>
<td>Read one character from keyboard. Character stored in R0[7:0].</td>
</tr>
<tr>
<td>PUTS</td>
<td>TRAP x22</td>
<td>Write null-terminated string to console. Address of string is in R0.</td>
</tr>
</tbody>
</table>

ASSEMBLER DIRECTIVES

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ORIG</td>
<td>address</td>
<td>starting address of program</td>
</tr>
<tr>
<td>.END</td>
<td></td>
<td>end of program</td>
</tr>
<tr>
<td>.BLKW</td>
<td>n</td>
<td>allocate n words of storage</td>
</tr>
<tr>
<td>.FILL</td>
<td>n</td>
<td>allocate one word, initialize with value n</td>
</tr>
<tr>
<td>.STRINGZ</td>
<td>n-character string</td>
<td>allocate n+1 locations, initialize w/characters and null terminator</td>
</tr>
</tbody>
</table>