Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 306, Fall 2014
Aater Suleman, Instructor
Owais Khan, Cagri Eryilmaz, Chirag Sakhuja, TAs
Exam 2, November 12, 2014

Solution

Name:____________________________

Problem 1 (20 points):_______

Problem 2 (20 points):_______

Problem 3 (20 points):_______

Problem 4 (20 points):_______

Problem 5 (10 points):_______

Total (90 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!**

Name:______________________________

**Problem 1.** (20 points):

**Part a.** (5 points): The following subroutine converts the 1's complement number in R0 to its 2's complement equivalent and stores the result back in R0. Complete this subroutine by filling in the two missing instructions.

Note: Your subroutine cannot clobber any registers other than R0. The input can be positive, negative, or zero. Recall that in 1's complement, a number can be negated by taking its NOT.

```
ADD R0,R0,#0
BRzp #1
ADD RΦ,RΦ,#1
RET
```

**Part b.** (5 points): When using memory-mapped I/O, I/O registers can be accessed using regular load and store instructions rather than special I/O instructions.

**Part c.** (5 points): Our LC-3 has become the CPU of choice for the upcoming iPhone. An Aggie is asked to implement a feature in which the volume button can be used to take a picture when the camera application is open. The Aggie ironically gets a working solution. She writes a subroutine that checks the button's status in a loop and takes a picture as soon as it detects that the button is pressed. She recommends that this subroutine be called when the camera app starts. What feature of LC-3 is the Aggie not using? Why will it be better to use this feature (answer in less than 10 words)?

Feature: Interrupts

Explanation:

Other work can be done ~~whike~~ instead of polling.

**Part d.** (5 points): An array has fast ______(1)______, while a linked list has fast ______(2)______. Circle all that apply.

(1) Inserts, Lookups
(2) Inserts, Lookups

Name:______________________________

**Problem 2.** (20 points):

Your job is to identify the bugs in an LC-3 assember written by a Sooner. You have already verified that the assembler **can correctly** handle opcode and register operands. You now try to assemble and run test programs to see if the assembler can handle offsets and assembler directives correctly.

Note 1: Assume that all unused memory locations and registers have x0000 when the program is run.
Note 2: The trap vector routines are all written correctly.
Note 3: The simulator has no bugs.

**Part a.** (10 points): When you assemble and run the following program, it prints "%A" on the console.

2 pts for showing understanding of program
2 pts for explanation
6 pts for correct solution

```
    .ORIG x3000
    LD R0, A
    OUT
    LD R0, B
    OUT
    HALT
A   .FILL x0041
B   .FILL x0042
    .END
```

x F025

Expected : AB
Reality : %A

'%' is x25

What is the bug? Explain in less than 15 words.

The assembler did not calculate the offset using the incremented PC.

**Part b.** (10 points): You fix the bug found in the previous test case. When you assemble and run the following program, it prints "HelloH" on the console.

```
    .ORIG x3000
    LEA R0, A
    PUTS
    AND R1, R1, #0
    ST R1, B
    PUTS
    HALT
A   .STRINGZ "Hello"
B   .BLKW 1
    .END
```

stores 0

Expected : Hello Hello
Reality : Hello H

0 ends up here

What is the bug? Explain in less than 15 words.

The assembler did not incorporate the string length when generating symbol addresses.

Name:________________________________

**Problem 3.** (20 points):

In genomics DNA computation, cancer cells are identified by comparing the DNA sequence of a normal cell to the DNA sequence of a cancer cell. We want to write a program that will perform this comparison for us.

For our program, we will represent DNA using a string of ASCII characters in memory. There will be two DNA sequences stored in memory. For each difference in the two sequences, we will insert a new node into a linked list. Each linked list node occupies 4 memory locations and contains the following elements in this order:

1. The original DNA character.
2. The position at which the character occurred. Position will start from 0, e.g., if the first character in the sequences did not match, the position field will be set to 0.
3. The mutated DNA character.
4. The pointer to the next node in the linked list. For simplicity, we will assume that the starting address of the next node will always be exactly 8 memory locations after the starting address of the current node. For example, if the current node is at location x4000, the next node will start at location x4008.

The result of our program will be a linked list that contains all the differences between the two sequences. The last node in the linked list will have x0000 stored in the pointer field. You may assume that the normal DNA sequence and the mutated DNA sequence will always be the same length, and this length will be provided in location x3100. You may also assume that the mutated DNA sequence will differ from the normal DNA sequence by at least one character.

**Your job:** Complete the code given on the next page. There are 6 boxes to be filled, and each box corresponds to exactly one line.

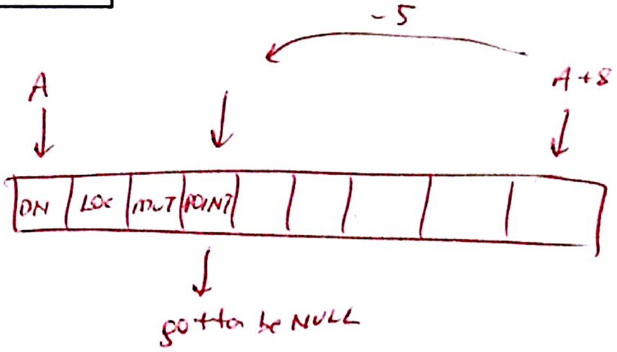
Name:______________________________

Fill in the empty boxes below.

```
; Assume all the registers are initially zero
.ORIG x3000

LD R0, DNASTART
LD R1, MUTATESTART
LDI R2, LENGTH
LD R5, LNKDLSTSTART

JSR CALCMUT
HALT

DNASTART        .FILL x3300
LENGTH          .FILL x3100
MUTATESTART     .FILL x3400
LNKDLSTSTART    .FILL x3500

CALCMUT  LDR R3,R0,#0     ; DNA CHAR
         LDR R4,R1,#0     ; MUT CHAR
         NOT R3,R3
         ADD R3,R3,#1
         ADD R3,R4,R3
         BRnp [ INSERT ]
CONT     ADD R0,R0,#1
         ADD R1,R1,#1
         [ ADD R6,R6,#1 ]
         ADD R2,R2,#-1
         BRp [ CALCMUT ]
         AND R2,R2,#0
         [ STR R2,R5,#-5 ]
         RET
INSERT   [ LDR R3,R0,#0 ]
         STR R3,R5,#0
         STR R6,R5,#1
         STR R4,R5,#2
         ADD R4,R5,#8
         STR R4,R5,#3
         [ ADD R5,R5,#8 ]
         BRnzp CONT

.END
```

positive and zero number -3
negative number -2
(e.g -3,-4,-2...)

OR ADD R5,R4,#0



Name:________________________________

**Problem 4.** (20 points):
Consider an encryption scheme that takes a string and a password as inputs. To encrypt the string, you simply add the ASCII values of the string and the password, character by character. For instance, if the input string is "Input" and the password is "Password", the encrypted result can be calculated as follows.

| Password | P | a | s | s | w | o | r | d |
|---|---|---|---|---|---|---|---|---|
| Input | I | n | p | u | t | | | |
| ASII Password | x50 | x61 | x73 | x73 | x77 | x6F | x72 | x64 |
| ASCII Input | x49 | x6E | x70 | x75 | x74 | x00 | x00 | x00 |
| Encrypted | x99 | xCF | xE3 | xE8 | xEB | x6F | x72 | x64 |

The following program takes a single input string and implements the encryption scheme defined above using a password labeled as Pass. At the end of execution, the encrypted string will overwrite the existing password. Answer the questions on the next page regarding the program.

```
            .ORIG x3000
            LEA R0, Msg
            PUTS
            JSR Encrypt
            HALT

Encrypt     ST R7, SaveR7
            LD R1, NegEnt
            LEA R2, Pass
Loop        LDR R3, R2, #0
            GETC
            ADD R7, R0, R1
            BRz Exit
            OUT
            ADD R7, R0, R3
            STR R7, R2, #0
            ADD R2, R2, #1
            BRnzp Loop
Exit        AND R0, R0, #0
            STR R0, R2, #0
            LD R7, SaveR7
            RET

NegEnt      .FILL x-A
Msg         .STRINGZ "Enter a string: "
Pass        .STRINGZ "PASSWORD"
SaveR7      .BLKW 1
            .END
```

x3000+47 =
x302F

Name:______________________________

**Part a.** (10 points): The Encrypt subroutine makes an assumption about the length of the input string. Explain how this can be problematic in less than 15 words.

You can overflow memory because the program doesn't check length.

**Part b.** (10 points): A hacker places malicious code at location x3050 and then runs the encryption program above. Give an example input string that the hacker can type to make his code execute.

Want to overflow SaveR7.

SaveR7 contains x3003 at beginning of subroutine.

To change it to x3050, need to add x4D = 'M'.

Technically, memory between encryption program and the malicious code is unknown. If you assume it to be x0000 (i.e. branch never), you can jump there and expect PC to eventually reach x3050.

Anything between x3030 & x3050 is valid.