Computer Science

COMPSCI 210 S2

Assignment TWO

Due:	**9:00 pm Monday 25th September 2017**
Worth:	**5% of the final mark**
Late Submission	**30% penalty**

Introduction

This assignment is to be done using LC-3 simulator. You can download the JAVA version from Canvas.

You can use the simulator to compile and test the program.

Section 1: Running the Simulator [1] (1 mark)

You can execute the simulator ('LC3sim.jar'). We need to first load some software. The first piece of software we should load is, naturally, an operating system. The LC-3 operating system is very basic: it handles simple I/O operations and is responsible for starting other programs. Download the LC-3 OS ('LC3os.asm') and you can understand what the operating system does.

The LC-3 machine doesn't understand assembly directly; we first have to 'assemble' the assembly code into machine language (it is an '.obj' file containing binary data). The LC-3 simulator has a built-in assembler, accessible (as is the case for most of its functionality) via the Command Line text box. To assemble the operating system, type ***as lc3os.asm*** at the command line and hit enter. Make sure that the OS file is in the same directory as the '.jar' file; the as command also understands relative and absolute paths if the OS is in a different directory. Output from the assembly process is displayed in the CommandLine Output Pane. After assembling the OS, you should notice that 2 new files, 'lc3os.obj' and 'lc3os.sym', have been created. The '.obj' file is the machine language encoding of the assembly language file, and the '.sym' file is a text file that holds symbol information so the simulator can display your symbols. Recall that symbols are really just a convenience for silly humans; the machine language encoding knows only about offsets.

Now we can load the 'lc3os.obj' file into the simulator, either via the command ***load lc3os.obj*** or by going to the File menu and selecting Open '.obj' file. Notice that the contents of the memory change when the OS is loaded. Now assemble and load the solution file for Problem 0 into the simulator. The memory has changed again, but you may not notice since the relevant memory addresses (starting at x3000) aren't visible unless you've scrolled the screen. User-level programs (i.e., non-OS code) start, by convention, at x3000. If you type the command ***list x3000*** the memory view will jump to x3000 and you can see the 1-instruction solution to this problem.

To actually run code, you can use the 4 control buttons at the top of the simulator, or type commands into the command line interface (the command names are the same as the buttons). Note that the PC register is set to x0200, which is the entry point to the operating system by convention. You can set the value in the registers. Example: You can set the value of R2, either by double-clicking it in the Registers section, or via the command ***set R2 (value)***. Now, actually run the code by hitting the continue button. You can find more details of operations from [1].

In section 1, you are going to revise the program below. This program will count the occurrence of a character. You first assemble all the files: 'lc3os.asm', 'sample.asm' and 'sample_data.asm'. Hence, you execute the following commands: ***load lc3os.obj***, ***load sample.obj*** and ***load sameple_data.obj***. Click 'continue' to run the program. You can click on the input panel (bottom left of the simulator). Input a key and the occurrence will be returned. If you input 'H', '1' is returned. It is case sensitive.

```
; Program to count occurrences of a character in a file.
; Character to be input from the keyboard.
; Result to be displayed on the monitor.
; Program only works if no more than 9 occurrences are found.
;
;
; Initialization
;
                .ORIG   x3000
                AND     R2, R2, #0      ; R2 is counter, initially 0
                LD      R3, PTR         ; R3 is pointer to characters
                GETC                    ; R0 gets character input
                LDR     R1, R3, #0      ; R1 gets first character
;
; Test character for end of file
;
TEST    ADD     R4, R1, #-10            ; Test for end of line (ASCII xA)
                BRz     OUTPUT          ; If done, prepare the output
;
; Test character for match.  If a match, increment count.
;
                NOT     R1, R1
                ADD     R1, R1, R0      ; If match, R1 = xFFFF
                NOT     R1, R1          ; If match, R1 = x0000
                BRnp    GETCHAR         ; If no match, test for capital ones
                ADD     R2, R2, #1

;
; Get next character from file.
;
GETCHAR         ADD     R3, R3, #1      ; Point to next character.
                LDR     R1, R3, #0      ; R1 gets next char to test
                BRnzp   TEST
;
; Output the count.
;
OUTPUT          LD      R0, ASCII       ; Load the ASCII template
                ADD     R0, R0, R2      ; Convert binary count to ASCII
                OUT                     ; ASCII code in R0 is displayed.
                HALT                    ; Halt machine
;
; Storage for pointer and ASCII template
;
ASCII   .FILL   x0030
PTR             .FILL   x4000
                .END
```

You now revise the program so the output will also display the inputted character. For example: If you input ‘H’, ‘H1’ is returned.

WARNING: We will use the JAVA simulator for marking. In particular, you should make sure that your answer will produce ONLY the exact output expected. The markers simply makes an exact comparison with the expected output. If you have any debug printouts or other code which produces some **unexpected output**, the markers will give you **zero marks**. If your files **cannot be compiled** successfully or they **cannot be executed** after compilation, the markers will also give you **zero marks**.

Section 2: Supporting Case Insensitive (2 marks)

You are going to revise the program such that the user can input small letters only but the occurrence of both corresponding small and capital letters are counted.

For example (data from 'sample_data.asm'):

Input h: h1

Input w: w1

For example (data from 'sample_data1.asm'):

Input b: b1

Input c: c3

Input d: d4

Section 3: Supporting Double Digit (2 marks)

The previous program can count the occurrence from 0 to 9 only. If it is greater than 9, an unexpected output will be returned. You are going to extend the program to support counting the occurrence from 00 to 99.

For example (data from 'sample_data1.asm'):

Input a: a40

Input b: b01

Input c: c03

Submission

You may electronically submit your assignment through the Web Dropbox (https://adb.auckland.ac.nz/) at any time from the first submission date up until the final date. You can make more than one submission. However, every submission that you make replaces your previous submission. Submit ALL your files in every submission. Only your very latest submission will be marked. Please double check that you have included all the files required to run your program.

No marks will be awarded if your program does not compile and run. You are to electronically submit all the following files:

1. **Q1.asm** for section 1
2. **Q2.asm** for section 2
3. **Q3.asm** for section 3

Integrity

Any work you submit must be your work and your work alone. To share assignment solutions and source code is not permitted under our academic integrity policy. Violation of this will result in your assignment submission attracting no marks, and you will face disciplinary actions in addition.

Reference

[1] http://www.cis.upenn.edu/~milom/cse240-Fall05/handouts/lc3guide.html