Computer Science

COMPSCI 210 S1, 2023

Assignment ONE

Due: **09:30 pm Tuesday 2nd May 2023**
Worth: **5% of the final mark**

Introduction

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

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

Section 0: Getting Started (Running the Simulator)

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') from Canvas (or from the given link in references) 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 Command Line 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 (or label) information so the simulator can display your symbols. **Recall that symbols/labels 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 (Q0.asm) 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 assembly instructions of your program.

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, run the code by hitting the continue button. You can find more details of operations from [1,2].

In section 0, execute the program file (Q0.asm). This program will take two input operands and output the "addition" results of those inputs. You first assemble all the files: 'lc3os.asm', 'data0.asm' and 'Q0.asm'. Then, you execute the following commands: ***load lc3os.obj***, ***load data0.obj*** and ***load Q0.obj*** to load the corresponding machine code files. Click 'Continue' to run the program. You can see the results from the display at the bottom-left.

```
.....
;
; A subroutine to add the values from R2 and R3 (R2 + R3). The result is saved at R3.
;
MyADD               ADD       R3, R2, R3
                    RET

;
; A subroutine to subtract the value of R3 from R2 (R2 - R3). The result is saved at R3.
;
MySUB               ; to be completed
                    RET

;
; A subroutine to OR the values from R2 and R3 (R2 OR R3). The result is saved at R3.
;
MyOR                ; to be completed
                    RET

;
; A subroutine to calculate the value stored at R2 left-shift by the value stored at R3 (R2 << R3). The result is saved
at R3.
;
MySHIFT             ; to be completed
                    RET

;
; A subroutine to XOR the values from R2 and R3 (R2 XOR R3). The result is saved at R3.
;
MyXOR               ; to be completed
                    RET

;
; A subroutine to multiply the value from R3 and R2 (R2 * R3). The result is saved at R3.
;
MyMULT              ; to be completed
                    RET

;
; A subroutine to divide the value stored at R2 (dividend) with the value stored at R3 (divisor) (R2 % R3). The result
(remainder) is saved at R3.
;
MyMOD               ; to be completed
                    RET
.....
```

Go through the program of the sample file (Q0.asm), the output will display the result of ADD operations of every two input values (after the '+' character) from the "data0.asm". The first character of the data file is used to identify which operation is going to be executed. You can save the program as the file Q1.asm. Next, you are going to complete the highlighted area of the program to finish the whole assignment. The output of executing Q0 with data file data0 is shown below.

001+009=010
007+004=011
006+004=010
000+005=005
006+005=011
009+004=013
006+006=012
001+001=002
008+000=008
009+009=018

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 make 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**. The markers may use different data values than what is provided for testing your code.

Section 1: Subtraction (0.5 marks)

You now complete the subroutine “MySUB”. It is a subroutine to subtract the value of R3 from R2 (R2 - R3). The result is saved at R3.

For example (data from ‘data1.asm’):
009-001=008
004-004=000
008-002=006
003-002=001
001-000=001
005-001=004
009-003=006
008-004=004
000-000=000
002-001=001

Section 2: OR Operation (0.5 marks)

You now complete the subroutine “MyOR”. It is a subroutine to “OR” the values from R2 and R3 (R2 OR R3). The result is saved at R3.

For example (data from ‘data2.asm’):
009|001=009
008|002=010
007|003=007
001|009=009
008|007=015
007|006=007
006|005=007
002|000=002
000|000=000
005|004=005

Section 3: Left Shift Operation (1 mark)

You now complete the subroutine “MySHIFT”. It is a subroutine to calculate the value stored at R2 left-shift by the value stored at R3 (R2 << R3). The result is saved at R3.

For example (data from ‘data3.asm’):
009<001=018
004<004=064
008<002=032
007<003=056
006<005=192
005<004=080
004<003=032
003<002=012
001<000=001
002<004=032

Section 4: XOR Operation (1 mark)

You now complete the subroutine "MyXOR". It is a subroutine to XOR the values from R2 and R3 (R2 XOR R3). The result is saved at R3.

For example (data from 'data4.asm'):
002_007=005
003_008=011
001_001=000
005_004=001
001_009=008
003_000=003
009_003=010
008_001=009
007_004=003
000_000=000

Section 5: Multiply Function (1 mark)

You now complete the subroutine "MyMULT". It is a subroutine to multiply the value from R3 and R2 (R2 * R3). The result is saved at R3.

For example (data from 'data5.asm'):
008*002=016
007*003=021
009*003=027
001*005=005
005*001=005
002*002=004
009*000=000
004*004=016
006*001=006
001*009=009

Section 6: Modulus Function (1 mark)

You now complete the subroutine "MyMOD". It is a subroutine to divide the value stored at R2 (dividend) with the value stored at R3 (divisor) (R2 % R3). The result (remainder) is saved at R3.

For example (data from 'data6.asm'):
009%006=003
004%004=000
004%007=004
007%004=003
006%004=002
005%004=001
004%003=001
003%002=001
004%009=004
009%007=002

You can make the following assumptions:

1. All input value should be between 000_{10} – 009_{10}.
2. The results of all output should be between 000_{10} – 999_{10}.
3. All inputs and outputs should be positive.
4. There should not be any invalid inputs from the input data file.

Submission

You may electronically submit your assignment through **Canvas submission system** 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. Only your very latest submission will be marked. Please double check that your submitted file can compile and execute properly with the given data files on the LC3 simulator (java version).

Please include your NAME and UPI in the file you submit.

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

1. **Q1.asm**

Late Submission

- 5% penalty if submitted on 3rd May 2023
- 10% penalty if submitted on 4th May 2023
- 15% penalty if submitted on 5th May 2023

No more submission will be allowed after that period.

Integrity

Any work you submit must be your work and your work alone. To share assignment solutions and source code is not permitted under the 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] https://acg.cis.upenn.edu/milom/cse240-Fall06/pennsim/pennsim-guide.html