Lab 6: Programming in LC-3 machine language

This Lab assignment is due on next Friday in MySTU.

> **Lab 6 is to be done individually. This is not a team project!**
>
> **You cannot look at anybody else's solution! You cannot get help from other students! If we find your code to be substantially similar to the code of another student, you will receive 0 for the lab and the incident will be reported to the College.**

In this lab, you will write a program in binary LC-3 machine language. Your program will compute the sum of positive numbers among a set of 10 numbers represented in 2's complement format and will store the result in register R5. Your program will begin at memory location x3100. The ten 16-bit binary numbers to analyze will be stored in memory starting from address x3132.

Please download **lab06.tar.gz** from MySTU and extract to your "lab6" folder.

- The file **lab6.bin** is where you will write your binary program.
- The file **numbers.bin** will contain the 10 16-bit binary numbers that your program will analyze.

Example

Given the following 10 numbers:

Address	Binary Number	(in Hex)	(in Decimal)
x3132	0000 0000 0000 0001	x0001	1
x3133	1111 1111 1000 0000	xFF80	-128
x3134	0000 0000 0000 0100	x0004	4
x3135	1111 1111 1111 1000	xFFF8	-8
x3136	0000 0000 0010 0000	x0020	32
x3137	0000 0010 0000 0000	x0200	512
x3138	1111 1111 1100 0000	xFFC0	-64
x3139	1111 1111 1111 1110	xFFFE	-2
x313A	1111 1111 0000 0000	xFF00	-256
x313B	0000 0000 0001 0000	x0010	16

Your program should add up all positive numbers (1, 4, 32, 512, and 16) and the following value should be stored in Register 5 after the completion of your program:

Value stored in R5	(in Hex)	(in Decimal)
0000 0010 0001 0111	x0235	565

Suggested Algorithm/Decomposition

If you're not sure how to get started with this lab assignment, we've provided a simple systematic decomposition that you can build on to develop your algorithm and your program. You are not required to use this decomposition; we provide it only as a reference to get you started. Note that we use generic action terms here like "Load" and "Compare" – these are an important part of the algorithm, but it is up to you to figure out how to accomplish these actions.

Basic Algorithm

1. Initialize Registers
    a. Initialize a register, say R3, to be used as a pointer to point to the location of the numbers in memory that are being analyzed, starting from address x3132
    b. Initialize a register, say R4, to the value 10. This will be used as a counter
    c. Initialize Register 5 to 0, this will be your final sum
2. Set up your loop
    a. Load in from memory (using R3 to tell you where) one of the ten binary numbers
    b. Check if the loaded number is positive
        i. If the number is positive, add it to the value stored in R5
        ii. if the new number is negative, ignore it
    c. Increment pointer and decrement counter
    d. repeat steps 2.a - 2.c until all 10 numbers have been examined

3. Halt the program

Specific Requirements

- Your code must be written in the **binary** LC-3 machine language, and be named **lab6.bin**.
- Your code must begin at memory location x3100, whereas the 10 positive numbers begin at memory location x3132.
- Your program must use a loop.
- You program may only consist of at most 50 16-bit binary words (not including the 10 positive numbers).
- Your code must be commented properly (see Style and Comments section below).

Style and Comments

For readability reasons, you must not have more than 120 characters on a single line. If you are writing a long comment that would otherwise exceed 120 characters, you should continue your comment on the next line.

Comment/Style Requirements:

- At the very top of your code, you must include an introductory comment section with your name, date, and a brief explanation of the purpose and general function of your program.
- Below that paragraph, include a register table explaining the role of each register used by your program.
- For each binary word, you may use spaces to separate instruction fields (e.g. 0010 001 000000100 for the LD instruction instead of 0010001000000100).
- Your code must be well-commented on every line (every line, as it is in binary. Programs not coded in binary do not need comments for every single line).
    - Comments must begin with a semicolon
    - You may use RTL or Assembly as your comment (i.e. "R7 <-- R7 + 1"), but you MUST also include some more description about what this particular line of code is doing
        - for example:
          0001 111 111 1 00001 ; R7<--R7+1, incrementing the counter register (R7)

In case you still don't understand the coding style of machine code, below is a sample code you can refer to. Please follow this format! (Attention: this code has nothing to do with your lab assignment, it's just a sample.)

```
; read a decimal number from the keyboard,
; convert it from ASCII to 2's complement, and
; store it in a predefined memory location. &amp;nbsp;If
; any non-numeric character is pressed, or the
; number overflows, store a 0 and print an error
; message.

; R0 holds the value of the last key pressed
; R1 holds the current value of the number being input
; R2 holds the additive inverse of ASCII '0' (0xFFD0)
; R3 is used as a temporary register

00110000 00000000          ; starting address is x3000

0010 010 000010100         ; R2 <- M[PC+x14]          put the value \-x30 in R2 0101 001
001 1 00000                ; R1 <- 0                  clear the current value 1111 0000
00100000                   ; R0 <- keyboard           read a character
1111 0000 00100001         ; display <- R0            echo it back to monitor 0001 011
000 1 10110                ; R3 <- R0-10              compare with ENTER
0000 010 000010001         ; if z, PC <- PC+x11       ENTER pressed, so done 0001
000 000 0 00 010 ; R0 <- R0+R2  subtract x30 from R0
0000 100 000010001         ; if n, PC <- PC+x11       smaller than '0' means error 0001 011
000 1 10110                ; R3 <- R0-10              check if > '9'
0000 011 000001111         ; if zp, PC <-PC+xF        greater than '9' means error
0001 011 001 0 00 001 ; R3 <- R1+R1                   sequence of adds multiplies R1 by 10 0000 100
```

000010101 ; if n, PC <- PC+x15 overflow, but not really necessary here

...

Tools / Running Your Code

In your GIT repository, you will be provided with the following 2 template files inside of the **lab6** directory: **lab6.bin** and **numbers.bin**.

In file **lab6.bin**, you will write all of the code for this lab. In file **numbers.bin**, you will place your ten numbers (in 16-bit 2's complement binary notation) that your program will sort through. We have provided a default set of numbers to try, but **DO BE SURE TO TRY OTHER NUMBERS!** When we grade this lab, we will be trying to run your code with several different assortments of numbers. A good strategy is to test (at the very least) the number sets provided in the above example.

To run and test your code, execute the following steps

First, you will have to convert your code:

1. Open up a terminal, and navigate to the folder where your code is located. Be sure your files are saved (both lab6.bin and numbers.bin)
2. Type in: **lc3convert lab6.bin**
3. Type in: **lc3convert numbers.bin**

Now you have 2 options (once you have run the lc3convert command on both of your files): You may run your code in the command line simulator, or the graphical simulator. We suggest using the graphical simulator to begin with, as this may be a more intuitive way to debug your code. However, the graders will be running your code in the command line simulator, so be sure your program works in the command line simulator! (Although if it works in one, most likely it will work in the other)

To run your code in the Graphical simulator:

1. While still in the folder where your code is located, type in **lc3sim-tk**. (NOTE: do not enter any .obj files after the command. we will load them momentarily inside the simulator)
2. On the bottom of the editor, it has a text bar that is labeled "File to Load". Click on the **Browse** button located right next to this text bar
3. Select the file **numbers.obj**, and click **Open**. (NOTE: Loading the files in the right order matters, so be sure to load numbers.obj first)
4. A pop-up box may appear that reads: "WARNING: No symbols are available". This is fine, just click **OK** if this popup box opens
5. Next, click on the **Browse** button one more time.
6. Select the file **lab6.obj**, and click **Open**.
7. Again click **OK** if a warning box appears.
8. Your program is now loaded. You can now click the buttons **Step** to step through your code line by line and see how it changes the registers, or you can click **Continue** to execute your whole program at once (it will stop when it reaches the HALT instruction). Once completed, your value for R5 should contain the sum (in hex) of the positive numbers out of the 10 numbers you entered in the file **numbers.bin**.

To run your code in the Graphical simulator a second time without closing and reopening the simulator:

1. After your program runs, click the "Reset LC-3" button in the bottom left.
2. Follow steps 2 through 8 above.

To run your code in the command line simulator (HOW THE GRADERS WILL DO IT):

1. While still in the folder where your code is located, type in **lc3sim**. (NOTE: do not enter any .obj files after the command. we will load them momentarily inside the simulator)
2. Once in the simulator, type: **file numbers.obj** (this loads the numbers into the simulator)
3. Next, type: **file lab6.obj** (this loads your program into the simulator)
4. Your program is now loaded! You can now type **step** to step through your code line by line and see how it changes the registers (you will have to type step repeatedly), or you can type **continue** to execute your whole program at once (it will stop when it reaches the HALT instruction). Once completed, your value for R5 should contain the sum (in hex) of the positive numbers out of the 10 numbers you entered in the file numbers.bin.

While there are many different ways to test and run your code, **Your code must work in the LC-3 command line simulator on the Linux machines in order to receive credit.** Please make sure your code works before you make your final commit to GIT repository.

What to Submit

- Your final program **lab6.bin** to MySTU.
- Demonstrate your program in front of a Teaching Assistant at Lab, answer questions, and sign your name.

Grading

The functionality grade script will execute your program with several test inputs (different combinations of numbers in the **numbers.bin** file) and will verify the correctness of the answers computed by your program. If the correct answer is stored in register 5 after it has run each time, that will be considered a successful implementation. If your program does not store the correct answer in R5, you will not get any credit for your implementation!

Grading Breakdown

- Functionality (70%)
  - 5%: Program starts at x3100
  - 5%: Result is stored in R5
  - 5%: Program halts
  - 5%: Program reads 10 consecutive numbers starting from address x3132
  - 5%: Registers are properly initialized/cleared
  - 15%: Passes test case with all positive numbers
  - 15%: Passes test case with all negative numbers
  - 15%: Passes test case with mixed positive/negative numbers
- Style (20%)
  - 5%: Uses one loop
  - 5%: Uses one test to compare numbers
  - 5%: No unnecessary data movement
  - 5%: Code is short
- Format (10%)
  - 2%: Lines are kept to 120 characters.
  - 3%: Machine code and comments are formatted properly.
  - 2%: Brief introductory paragraph explaining your overall approach to the solution
  - 3%: Code is well-commented

Late Work Policy

This lab assignment is complicated. Therefore it can be turned in up to 48 hours late, but with a late penalty of 0.5% per hour (rounded up to the nearest hour).