CISC 181 Lab 3 Programming the Toy

**Submission guidelines**

The problems on this lab require a mix of solution types. Problem 1 requires you to type a solution into a text box on this document. Problem 3d has a table for you to fill in and another text box. Problems 2, 3 and 4 require you to create and submit program files called Lab03\_2.toy, Lab03\_3.toy, and Lab03\_4.toy respectively. These are to be plain text files, as explained below. **Place the following into a zip file called Lab03.zip**

* **Your completed Lab 03.docx file**
* **Your Lab03\_2.toy program file**
* **Your Lab03\_3.toy program file**
* **Your Lab03\_4.toy program file**

**Upload that zip file to onQ by the lab's due date and time.**

**Introduction**

**Don't attempt this lab before you have read through this week's lecture slides!**

In this lab, you will try out a few programs written in the assembly language for the Toy computer described in the Week 3 slides. I have put a version of the Toy simulator on one of our own School of Computing web servers, and you will find it by starting up a web browser and going to [**http://sites.cs.queensu.ca/courses/cisc181/toy.html**](http://sites.cs.queensu.ca/courses/cisc181/toy.html). You may also download a copy for use on your own machine with or without a network connection from this week's resources on onQ. If you choose the latter route, you can open the file in a browser by dragging it from a file listing onto your open browser or by opening it from your browser by using Ctrl-O (Windows) or Cmd-O (Mac).

**Note that all mnemonic opcodes (get, print, load, store, add, sub, goto, ifpos, ifzero, stop) MUST be indented by at least one space from the left edge, and all memory location labels (e.g., top and bottom in the first exercise) and comments (lines starting with #) MUST NOT BE indented.**

Though you may type or copy and paste the code for some of these problems directly into the box on the left of the Toy interface I strongly recommend using a text editor to create and save them instead. A text editor is a program that works something like a *word processor* such as Microsoft Word, but without the ability to do special formatting, bolding, underlining, etc. It just stores plain, ordinary text. The free program Visual Studio Code (<https://code.visualstudio.com/download>) is an excellent choices for this and will come in handy for some future labs. The editor will let you save your programs to disk or (even better) cloud storage like OneDrive regularly. That is likely to save you time and frustration over lost work. The saved files can then be loaded into the simulator using its "Choose File" button.

Windows users may have trouble if their systems hide the extensions (last few characters) of file names. Unfortunately, this is the default behaviour of Windows Explorer, and it may lead to situations in which, for example, a file that looks like it is called "Lab03\_2.toy" is actually called "Lab03\_2.toy**.txt**". To make sure this is not a problem for you, follow the instructions given on this page, <https://fileinfo.com/help/windows_10_show_file_extensions>. Also, when saving a file from a text editor in windows, put the full file name in quotation marks, like this:

Graphical user interface, application, Word

Description automatically generated

Unfortunately, the programming language that the simulator was written in (JavaScript) specifically forbids file output for security reasons, so there is no automatic way to save your lovely programs if you have edited them in the Toy. Make your changes in your text editor, save them, and then reload your programs into the Toy.

1. (1 mark) Just to make sure you have got a handle on how to make the simulator work, type this short program into a text file using the editor of your choice, save it, then load it into the Toy simulator using the "Open File" button. (I'm suggesting typing rather than copying and pasting as the simulator will show you if you are prone to making typing errors.)

top get

ifzero bottom

print

goto top

bottom stop

Try to predict the behaviour of this program. Will it print out a 0 or some other number as its last operation? Execute the program by clicking on the simulator's RUN button. Here is a sequence of input values to try:

6

-2

5

1

0

Of these, which was the last value printed in the output window (on the right, above "stopped")?

1

Note that you don't need to include your saved file for this problem in your submitted .zip file.

1. (6 marks) Let us try working with a memory location for storing data. Type this into a new text file and save it with the name Lab03\_2.toy:

top get

ifzero show

print

add sum

store sum

goto top

show load sum

print

stop

sum 0

By the name I have given the memory location sum, you might suspect that this program adds up a list of number values, and you would be right. To see that this is the case, save the program, load it into the Toy simulator, run it, and enter the same sequence of input values that you used for part 1, above. When the value of sum is loaded into the accumulator, and then printed, its value should be 10. Now modify the program in your editor so that it not only prints out the sum, but the number (count) of non-zero values that were entered, too. Here is how:

* 1. Start by adding a memory address label called *count* to the bottom of the program code (i.e., under ‘sum 0’) and give it an initial value of 0.
  2. Insert a new instruction (on a new line) after the ‘store sum’ instruction to load the value of count into the accumulator. (Hint: the instruction is ‘load count’. Remember that instructions must be indented!)
  3. Insert an instruction after that to add 1 to the value in the accumulator.
  4. Insert an instruction after that to store the value in the accumulator at the memory location you have labelled *count*. (This will automatically overwrite whatever happens to be stored at location *count*.)
  5. Insert an instruction after the final ‘print’ that loads the value stored at memory location *count* into the accumulator.
  6. Insert a print instruction after that. (It should be immediately above the ‘stop’ instruction.)

If you have done your work properly, you should end up with a program that is 16 lines long (more if you have added any comments – that is entirely optional), and that, when saved, loaded into the simulator, and run with the same input data as that used the program in part 1, gives the following output:

6

-2

5

1

10

4

stopped

(where the 10 represents the sum, and the 4 represents the number of input values.)

Test your program on some other input, until you are satisfied that it is working correctly. Make sure you include your completed Lab03\_2.toy file in the .zip file you submit.

1. (6 marks) There is no division instruction in the Toy language, but maybe we can crudely approximate one by doing repeated subtractions to let the Toy calculate an integer **average** of a list of numbers. To keep things as simple as possible, we will arbitrarily specify that this program need only work for a list of non-zero positive integers, and that (as with the previous two programs) input will stop when a 0 is entered. NOTE: Sometimes this program will work with negative integers in the input, but if the overall sum of the numbers is negative, it will give incorrect results. I'm sure there's a way to program around this problem, but I don't want to make our code overlong, so let's just stick with non-negative input. Start with your Lab03\_2.toy code and save it as a new file called Lab03\_3.toy. (Be careful not to overwrite Lab03\_2.toy.) Make the following additions to the code:
   1. Add two new memory address labels at the bottom of the code, *avg* and *rem*. Give both initial values of 0.
   2. **Insert** the following code **between** the final *print* instruction and the *stop* instruction. **Note that I said ‘between the final *print* and the *stop* instructon;' *stop* should still be the last instruction executed.** If you are typing the code in, type carefully to catch any errors.

calcavg load sum

ifpos cont

goto showavg

cont store rem

sub count

store sum

ifpos incavg

goto calcavg

incavg load avg

add 1

store avg

goto calcavg

showavg load avg

print

load rem

print

* 1. Save your work in the text editor frequently, so that if something needs fixing you will not lose what you've done. **Include your completed Lab03\_3.toy program in your .zip file submission.**
  2. Run the program (at least) three times, using the following input data, noting what values print out for *avg* and *rem* in each case. Note that the 0 value at the end of each of these lists signals an end to input and is NOT used in the average calculations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Input sequence | Value printed for avg | Value printed for rem |
| i. | 1, 2, 3, 4, 5, 0 | 3 | 0 |
| ii. | 6, 8, 4, 2, 1, 7, 0 | 4 | 4 |
| iii. | 12, 13, 14, 3, 0 | 10 | 2 |

Hint: The last two numbers printed out are the values of *avg* and *rem*. *avg* is my abbreviation for *average* and *rem* is short for *remainder*. *rem* will, of course, always be 0 if the *sum* divided by the *count* is an integer. Check using a calculator app or by doing the arithmetic in your head that you are getting the correct results.

**To get any marks for this problem, your code must execute correctly.**

1. (3 marks) You are on your own for this problem. **Save a copy of Lab03\_3.toy as Lab03\_4.toy (again taking care not to overwrite Lab03\_3.toy).** Alter the program **so that it can include 0 values** in the sum and ends input, instead, after the user enters a **negative** number. For example, if the user enters 1, 2, 4, 0, 3, 8, -1, the output should be:

Text

Description automatically generated with medium confidence

**Note that if your revised program is much longer than the original, you are probably trying too hard.** To make sure your finished program is working correctly for more than just the input suggested above, test it using other lists of numbers to make sure it gives consistent results.

**Include your completed Lab03\_4.toy program in your .zip file submission.**

**To get any marks for this problem, your code must execute correctly.**