# CS 2110 Homework 06: Introduction to LC-3 Assembly

Madeleine Brickell, Cem Gokmen and Daniel Becker

Spring 2018

## Contents

# 1 Overview

The goal of this first assembly assignment is to help you become comfortable coding in the LC-3 assembly language. This will involve writing small programs, modifying memory, printing to the console, and converting from high-level code to assembly.

## 1.1 Helpful Hints

### Pseudo-Ops

Pseudo-Ops are directions for the assembler that aren't actually instructions in the ISA.

1. `.orig`: Tells the assembler to put this block of code at the desired address.

   Ex: `.orig x3000` will put the block of code at address x3000

2. `.stringz "something"`: Will put a string of characters in memory followed by NULL (which is a single ASCII character with the value 0).

   Ex: `.stringz "sanjay"` will put the ASCII code for the letter 's' in the first memory location, the code for 'a' in the second memory location, and so on until putting 'y' in the next-to-last position and NULL (which again, has the ASCII code 0) in the last memory location as the terminator.

3. `.blkw n`: A pseudo-op that will allocate the next $n$ locations of memory for a specified label.

4. `.fill value`: Pseudo-op that will put the value in that memory location

5. `.end`: A psuedo-op that denotes the end of an address block. Matches with an `.orig`.

### Traps

Traps are subroutines built into the LC-3 to help simplify instructions. Some helpful TRAPS include:

1. HALT: Halt is an alias for a TRAP that will stop the LC-3 from running
2. OUT: Out is an alias for a TRAP that will take whatever character is in R0 and print it to the console
3. PUTS: Puts is an alias for a TRAP that will print a string of characters with the starting address saved in R0 until it reaches a null (0) character
4. GETC: Getc is another TRAP alias that will take in a character input and store it in R0

Being an alias for a TRAP instruction means that the assembler will convert them to TRAP instructions at assembly time. For example, if you write HALT in your code, the assembler will convert it to the instruction, TRAP x25 for you.

### Example

The following small example will demonstrate the use of these Traps and Pseudo-Ops

```
.orig x3000                ;this is where our code will begin
LEA R0, HW                 ;loads the address of our string into R0
PUTS                       ;will print the string whose address is in R0
HALT                       ;stops the program from executing
HW .stringz "Hello. \n"    ;stores the word 'Hello' in memory with 'H' be located at address x3003,
                           ;'e' will be located at address x3004, etc
.end                       ;denotes the end of the address block
```

# 2 Instructions

## 2.1 Odd vs Even

To start you off with assembly, we would like you to write a small assembly program to determine whether each element in the array (located in memory location x6000) is an odd or even number and print that to the console, as ODD or EVEN. If the element is zero, print ZERO to the console instead.

You need to implement the odd/even check for an individual element in the array without using a loop - that means the odd/even/zero check for one given number should be $O(1)$ (its runtime should not depend on the magnitude of the number) and thus the overall code should be $O(n)$. The tester has a maximum instruction count that will enforce this - if you're passing the tests, you're good.

Ex: array: [9, 27, 64, 0] will produce this output:

```
ODD
ODD
EVEN
ZERO
```

## 2.2 Add-Subtract

In this problem, you will have to iterate over items in an array (located at memory location x6000) and either add or subtract them, depending on their index (numbers at even indices will be added, and numbers at odd indices will be subtracted). Therefore, the element at index 0 will be added, the element at index 1 will be subtracted, the element at index 2 will be added, etc until you reach the end of the array. Once you have the result, save it at the memory location given by the label RESULT.

Ex: array: [47, 36, 25, 9, 14, 11, 42] will result in

+ 47 - 36 + 25 - 9 + 14 - 11 + 42 = 72

## 2.3 Reverse String

Your next problem will be to reverse a string. You will be given the memory address of the first character of a string and the length of the string. Reverse the string IN PLACE and make sure to maintain '\n' as the last character (so that it prints to the console via PUTS correctly). The trailing newline character is included in the character count. You can assume that the last character will always be a newline.

Ex: string: "Ramblin Wreck\n" length: 14 will result in "kcerW nilbmaR\n".

## 2.4 Number Triangle

For this problem, you will be given a number in the label TRIANGLE and will build a decreasing number triangle from it. Example:

```
TRIANGLE - 9

999999999
 7777777
  55555
   333
    1
```

It's not the funnest (Yes I know it isn't a word, this is why I am a CS major) thing to derive an assembly file to solve this problem, so we will be giving you the following C code. It is recommended you follow the C code exactly in order to pass all grading tests.

```
int numberTriangle() {
        int counter = 0;
        int triangle = 5;
        int decimal = triangle;
        int forspaces = 0;

        if (decimal != 0) {
                while (decimal > 0) {
                        counter = 0;
                        while(counter < forspaces) {
                                printf(" ");
                                counter++;
                        }

                        counter = 0;
                        while (counter < decimal) {
                                printf("%d", decimal);
                                counter++;
                        }

                        decimal = decimal - 2;
                        forspaces++;
                        printf("\n");
                }
        }
}
```

Note: You will only be given single digit numbers. i.e. 0-9

# 3 Testing Your Work

## 3.1 Debugging

To debug assembly code, we highly recommend using complx, an LC-3 simulator. To use complx:

1. Type complx in the command prompt
2. Go to the 'File' menu, and select 'Randomize and Reload'. This will randomly assign values to every memory location and register, and will load your assembly file over the randomized memory.
3. To debug, click the 'Step' button to execute one instruction at a time.
4. To run, click the 'Run' button to execute the entire file

   Note: If you want to see how your code is modifying memory, press Ctrl + V to open a new view, and then press Ctrl + G to navigate to the desired address in memory.

## 3.2 Testing

To fully test your code:

Run lc3test example.xml example.asm

# 4 Rubric

Breakdown of Grade:

```
oddvseven 20%
addsubtract 20%
reversestring 30%
numbertriangle 30%
```

Note: Your TA's have the right to add more tests in grading. To ensure your assembly code covers all base cases, it is recommended that you test your own code. How? You can change the values of the array and run the code in complx.

# 5 Deliverables

Please upload the following files to Gradescope:

1. oddvseven.asm
2. addsubtract.asm
3. reverse.asm
4. numbertriangle.asm

**Download and test your submission to make sure you submitted the right files.**

# 6 LC-3 Assembly Programming Requirements

## 6.1 Overview

1. Your code must assemble with **NO WARNINGS OR ERRORS**. To assemble your program, open the file with Complx. It will complain if there are any issues. **If your code does not assemble you WILL get a zero for that file.**
2. **Comment your code!** This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad you left yourself notes on what certain instructions are contributing to the code. Comment things like what registers are being used for and what less intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semicolon (;), and the rest of that line will be a comment.
3. Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

   **Good Comment**

   ```
   ADD R3, R3, -1          ; counter--
   BRp LOOP                ; if counter == 0 don't loop again
   ```

   **Bad Comment**

```
ADD R3, R3, -1          ; Decrement R3
BRp LOOP                ; Branch to LOOP if positive
```

4. **DO NOT assume that ANYTHING in the LC-3 is already zero.** Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.
5. Following from 3. You can randomize the memory and load your program by doing File - Randomize and Load.
6. Use the LC-3 calling convention. This means that all local variables, frame pointer, etc... must be pushed onto the stack. Our autograder will be checking for correct stack setup.
7. Start the stack at xF000. **The stack pointer always points to the last used stack location.** This means you will allocate space **first**, then store onto the stack pointer.
8. Do NOT execute any data as if it were an instruction (meaning you should put .fills after **HALT** or RET).
9. Do not add any comments beginning with @plugin or change any comments of this kind.
10. **Test your assembly.** Don't just assume it works and turn it in.

# 7 Rules and Regulations

## 7.1 General Rules

1. Starting with the assembly homeworks, Any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

## 7.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.
2. When preparing your submission you may either submit the files individually to Canvas or you may submit an archive (zip or tar.gz only please) of the files (preferred). You can create an archive by right clicking on files and selecting the appropriate compress option on your system.
3. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want (see Deliverables).

4. Do not submit compiled files that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

5. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

## 7.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

## 7.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

**You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu**

## 7.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, as well as help each other debug code. What you shouldn't be doing,

however, is paired programming where you collaborate with each other on a low level. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, and it is often the case that the recipient will simply modify the code and submit it as their own.

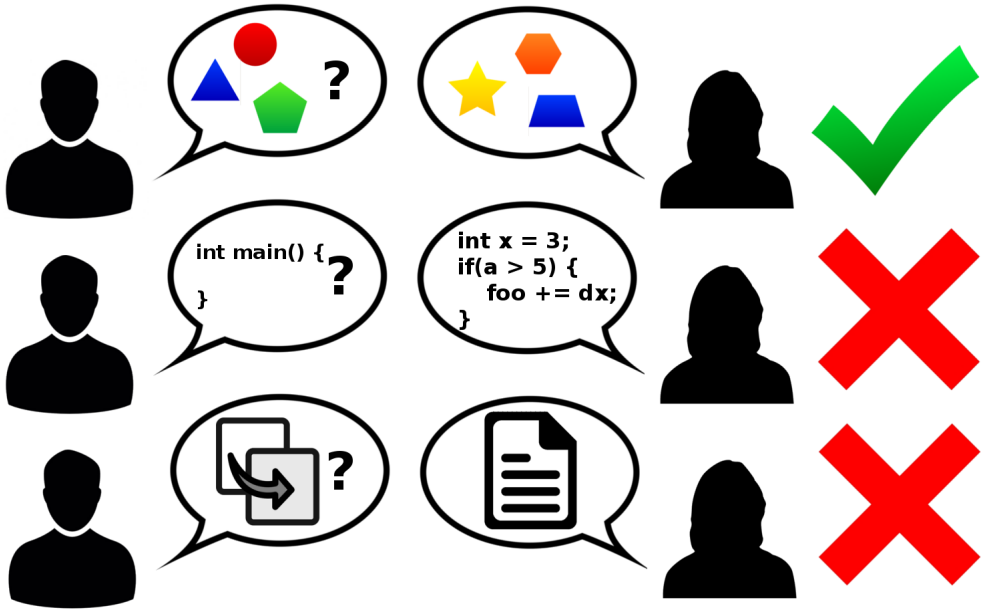


Figure 1: Collaboration rules, explained