

# CS 2110 Timed Lab 3: Subroutines and Calling Conventions

Max Everest, Sebastian Jankowski, Jonathan Marto, Annelise Lloyd, Zaid Akkawi

Summer 2023

## Contents

<b>1</b>	<b>Timed Lab Rules - Please Read</b>	<b>2</b>
<b>2</b>	<b>Overview</b>	<b>2</b>
2.1	Purpose . . . . .	2
2.2	Task . . . . .	2
2.3	Criteria . . . . .	2
<b>3</b>	<b>Detailed Instructions</b>	<b>3</b>
3.1	FIBONACCI . . . . .	3
3.2	ADDALL . . . . .	3
3.3	CHANGETREE . . . . .	4
<b>4</b>	<b>Checkpoints</b>	<b>5</b>
4.1	Checkpoints (67 points) . . . . .	5
4.2	Other Requirements (33 points) . . . . .	5
<b>5</b>	<b>Deliverables</b>	<b>5</b>
<b>6</b>	<b>Local Autograder</b>	<b>6</b>
<b>7</b>	<b>LC-3 Assembly Programming Requirements</b>	<b>7</b>
7.1	Overview . . . . .	7
<b>8</b>	<b>Appendix</b>	<b>8</b>
8.1	Appendix A: LC-3 Instruction Set Architecture . . . . .	8

**Please take the time to read the entire document before starting the assignment.** It is your responsibility to follow the instructions and rules.

## 1 Timed Lab Rules - Please Read

You are allowed to submit this timed lab starting from the moment your assignment is released until your individual period is over. You have 75 minutes to complete the lab, unless you have accommodations that have already been discussed with your professor. Gradescope submissions will remain open for several days, but you are not allowed to submit after the lab period is over. **You are responsible for watching your own time. Submitting or resubmitting after your due date may constitute an honor code violation.**

If you have questions during the timed lab, you may ask the TAs for clarification in lab, though you are ultimately responsible for what you submit. The information provided in this Timed Lab document takes precedence. If you notice any conflicting information, please indicate it to your TAs.

The timed lab is open-resource. You may reference your previous homeworks, class notes, etc., but your work must be your own. Contact in any form with any other person besides a TA is absolutely forbidden. **No collaboration is allowed for timed labs.**

## 2 Overview

### 2.1 Purpose

The purpose of this timed lab is to test your understanding of implementing subroutines in the LC-3 assembly language using the calling convention, from both the callee and caller side.

### 2.2 Task

You will implement the subroutines listed below in LC-3 assembly language. Please see the detailed instructions for the subroutines on the following pages. We have provided pseudocode for the subroutines—you should follow these algorithms when writing your assembly code. Your subroutines must adhere to the LC-3 calling conventions.

### 2.3 Criteria

Your assignment will be graded based on your ability to correctly translate the given pseudocode for a subroutine (function) into LC-3 assembly code, following the LC-3 calling convention. Please use the LC-3 instruction set when writing these programs. Check the Deliverables section for what you must submit to Gradescope.

You must produce the correct return values for each function. In addition, registers R0-R5 and R7 must be restored from the perspective of the caller, so they contain the same values after the caller's JSR subroutine call. Your subroutine must return to the correct point in the caller's code, and the caller must find the return value on the stack where it is expected to be. If you follow the LC-3 calling conventions correctly, all of these things will happen automatically. Additionally, we will check that you made the correct subroutine calls, so you should not try to implement a recursively subroutine iteratively.

Your code must assemble with no warnings or errors (Complx and the autograder will tell you if there are any). If your code does not assemble, we will not be able to grade that file and you will not receive any points.

## 3 Detailed Instructions

### 3.1 FIBONACCI

Calculates and returns a number from the fibonacci sequence.

**Parameter num:** Number of the fibonacci sequence to return

**Returns:** The numth fibonacci number

**Pseudocode:**

```
FIBONACCI(int num) {  
    if (num <= 1) {  
        return num;  
    } else {  
        int a = FIBONACCI(num - 1);  
        int b = FIBONACCI(num - 2);  
        return a + b;  
    }  
}
```

### 3.2 ADDALL

This function takes in a number num and returns the sum of all numbers between 1 and num inclusive.

**Parameter num:** The number that should be added with every positive number below it.

**Returns:** Sum of all numbers between 1 and num inclusive.

**Pseudocode:**

```
ADDALL(int num) {  
    int sum = 0;  
  
    for (int i = 1; i <= num; i++) {  
        sum = sum + i;  
    }  
  
    return sum;  
}
```

### 3.3 CHANGETREE

For this part of the Timed Lab, you will be given a binary tree. Each node in the tree will have three attributes: its value, its left node, and its right node. Each node is treated as an address in memory.

Given some node that lives at address x:

- $\text{mem}[x] = \text{node.data}$
- $\text{mem}[x + 1] = \text{node.left}$
- $\text{mem}[x + 2] = \text{node.right}$

If a node does not have a child, the address of that child in memory will be 0.

Takes an address to a node which contains a number and a maximum of two children. If the node contains an even number, the number should be replaced by the result of FIBONACCI with that number as an argument. If the node contains an odd number, the number should be replaced by the result of ADDALL with that number as an argument.

**Parameter address:** The address of the node

**Returns:** If the node is 0, return -1. Otherwise, return 0.

**Pseudocode:**

```
CHANGETREE(Node node (address)) {
    if (node == 0) {
        return -1;
    }

    int data = mem[node];

    if (data % 2 == 0) {
        mem[node] = FIBONACCI(data);
    } else {
        mem[node] = ADDALL(data);
    }

    Node left = mem[node + 1];
    CHANGETREE(left);

    Node right = mem[node + 2];
    CHANGETREE(right);

    return 0;
}
```

## 4 Checkpoints

### 4.1 Checkpoints (67 points)

In order to get all of the points for this timed lab, your code must meet these checkpoints:

- Checkpoint 1 (19 points): Implement subroutine `FIBONACCI` to return the appropriate number in the fibonacci sequence.
- Checkpoint 2 (19 points): Implement subroutine `ADDALL` to return the sum of all numbers in a range.
- Checkpoint 3 (10 points): Implement the base case of `CHANGETREE` to return -1 if node is 0.
- Checkpoint 4 (20 points): Implement the recursive case of `CHANGETREE` to successfully change the data in the tree and return 0.

### 4.2 Other Requirements (33 points)

Your subroutine must follow the LC-3 calling convention. Specifically, it must fulfill the following conditions:

- Your `FIBONACCI` and `CHANGETREE` subroutines must be recursive and call themselves according to the pseudocode's description.
- When your subroutine returns, every register must have its original value preserved (except R6).
- When your subroutine returns, the stack pointer (R6) must be decreased by 1 from its original value so that it now points to the return value.
  - If the autograder claims that you are making an unknown subroutine call to some label in your code, it may be that your code has two labels without an instruction between them. Removing one of the labels should appease the autograder.

## 5 Deliverables

Turn in the following files on Gradescope during your assigned timed lab slot:

1. `t103.asm`

## 6 Local Autograder

To run the autograder locally, follow the steps below depending upon your operating system:

- Mac/Linux Users:
  1. Navigate to the directory your timed lab is in (**in your terminal on your host machine, not in the Docker container via your browser**)
  2. Run the command `sudo chmod +x grade.sh`
  3. Now run `./grade.sh`
- Windows Users:
  1. In Git Bash (or Docker Quickstart Terminal for legacy Docker installations), navigate to the directory your timed lab is in
  2. Run `chmod +x grade.sh`
  3. Run `./grade.sh`

## 7 LC-3 Assembly Programming Requirements

### 7.1 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 4., you can randomize the memory and load your program by going to File ↩ Advanced Load and selecting RANDOMIZE for registers and memory.
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. The stack will start 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 HALT or RET instructions before any .fills).
9. Do not add any comments beginning with @plugin or change any comments of this kind.
10. You should not use a compiler that outputs LC3 to do this assignment.
11. **Test your assembly.** Don't just assume it works and turn it in.

## 8 Appendix

### 8.1 Appendix A: LC-3 Instruction Set Architecture

ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
BR	0000	n	z	p	PCOffset9	
JMP	1100	000	BaseR	000000		
JSR	0100	1	PCOffset11			
JSRR	0100	0	00	BaseR	000000	
LD	0010	DR	PCOffset9			
LDI	1010	DR	PCOffset9			
LDR	0110	DR	BaseR	offset6		
LEA	1110	DR	PCOffset9			
NOT	1001	DR	SR	111111		
ST	0011	SR	PCOffset9			
STI	1011	SR	PCOffset9			
STR	0111	SR	BaseR	offset6		
TRAP	1111	0000	trapvect8			

Trap Vector	Assembler Name
x20	GETC
x21	OUT
x22	PUTS
x23	IN
x25	HALT

Device Register	Address
Keybd Status Reg	xFE00
Keybd Data Reg	xFE02
Display Status Reg	xFE04
Display Data Reg	xFE06





