CS 2110 Homework 07: Assembly Subroutines and Recursion

Yuuna Hoshi, Preston Olds, and Michael Xu

Summer 2018

Contents

1 Overview

The goal of this assignment is to familiarize you with the LC-3 calling convention using assembly recursion. This will involve use of the stack, as seen in lecture, to save the return address (RA) and old frame pointer (OFP).

In each section, you'll act as a compiler, converting the provided pseudocode into assembly code in the corresponding .asm file which follows the LC-3 calling convention. (For more information on the calling convention, see the Calling Convention Guide later in this PDF.) To check your work, we will provide tests which you can run as follows:

```
$ lc3test testfile.xml asmfile.asm
```

To debug failing tests, use Debug → Setup Test in Complx. This way, you can step through step by step and see what goes wrong instead of running the test repeatedly hoping for a new result.

2 Functions

2.1 Mod

The first function will calculate x % y. Your function will take in two arguments, x and y. It is guaranteed that x and y will be positive.

```
""" Function to calculate x % y """
mod(x, y):
    if x < y:
        return x
    else:
        return mod(x - y, y)
```

2.2 Find Max of Array

For the next function, you will write a function to find the maximum value in an array. Your function will take in 2 parameters: the address of the array, and the length of the array.

It is guaranteed that the array will always have at least one element (i.e. the length of the array will be at least 1).

Pseudocode:

```
findmax(array, n):
    if n == 1:
        return array[0]
    else:
        currentMax = findmax(array, n-1)
        return max(currentMax, array[n-1])
```

2.3 Fibonacci

For the last problem, you will write a function to compute the n-th Fibonacci number. It is guaranteed that n will be a non-negative number.

```
fib(n):
    if n <= 1:
        return n
    else:
        return fib(n - 2) + fib (n - 1)
```

3 Rubric

1. mod – 20%
2. findmax – 40%
3. fibonacci – 40%

4 Deliverables

Please upload the following files to **Gradescope**:

1. `mod.asm`
2. `findmax.asm`
3. `fib.asm`

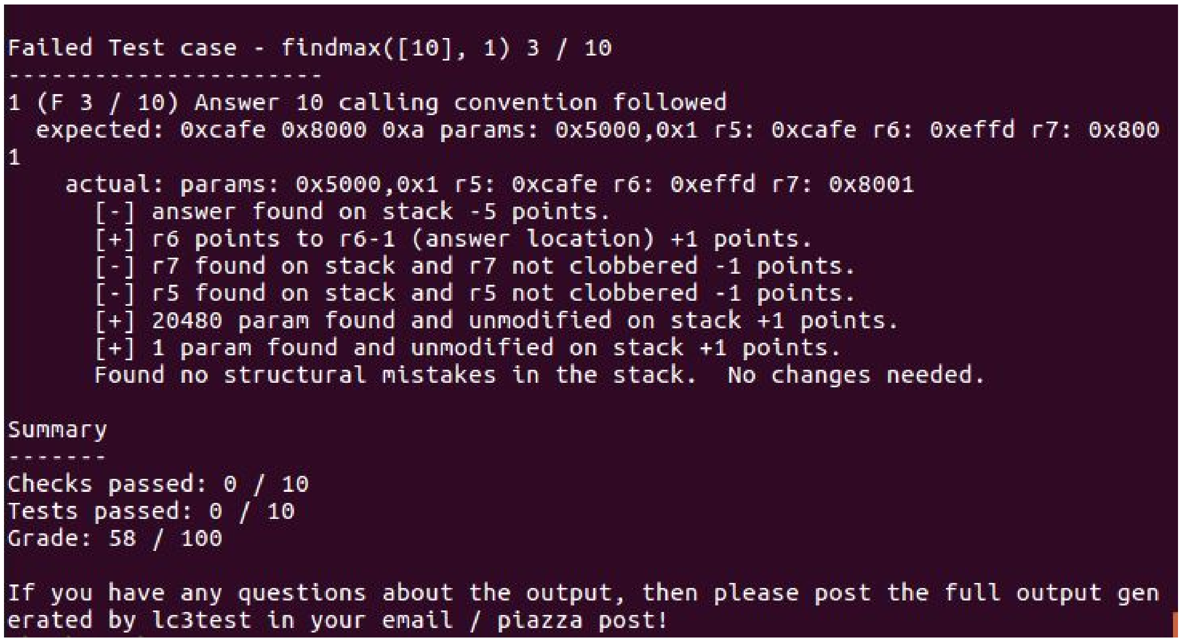
Download and test your submission to make sure you submitted the right files.

5 FAQ

1. **What does the output of the tester mean?**

   The tester will run through a all of the test cases provided, and give a breakdown of what score you received on each test case, along with a total score.

   For example, let's say your output looked like this:



```
Failed Test case - findmax([10], 1) 3 / 10
----------------------
1 (F 3 / 10) Answer 10 calling convention followed
  expected: 0xcafe 0x8000 0xa params: 0x5000,0x1 r5: 0xcafe r6: 0xeffd r7: 0x800
1
    actual: params: 0x5000,0x1 r5: 0xcafe r6: 0xeffd r7: 0x8001
      [-] answer found on stack -5 points.
      [+] r6 points to r6-1 (answer location) +1 points.
      [-] r7 found on stack and r7 not clobbered -1 points.
      [-] r5 found on stack and r5 not clobbered -1 points.
      [+] 20480 param found and unmodified on stack +1 points.
      [+] 1 param found and unmodified on stack +1 points.
      Found no structural mistakes in the stack.  No changes needed.

Summary
-------
Checks passed: 0 / 10
Tests passed: 0 / 10
Grade: 58 / 100

If you have any questions about the output, then please post the full output gen
erated by lc3test in your email / piazza post!
```

   `[-] answer found on stack -5 points.`: This means that the answer that should have been returned by your function was not placed on the stack, when it should have been.

   `[+] r6 points to r6-1 (answer location) +1 points.`: This means that your R6 (stack pointer) is pointing to the correct location.

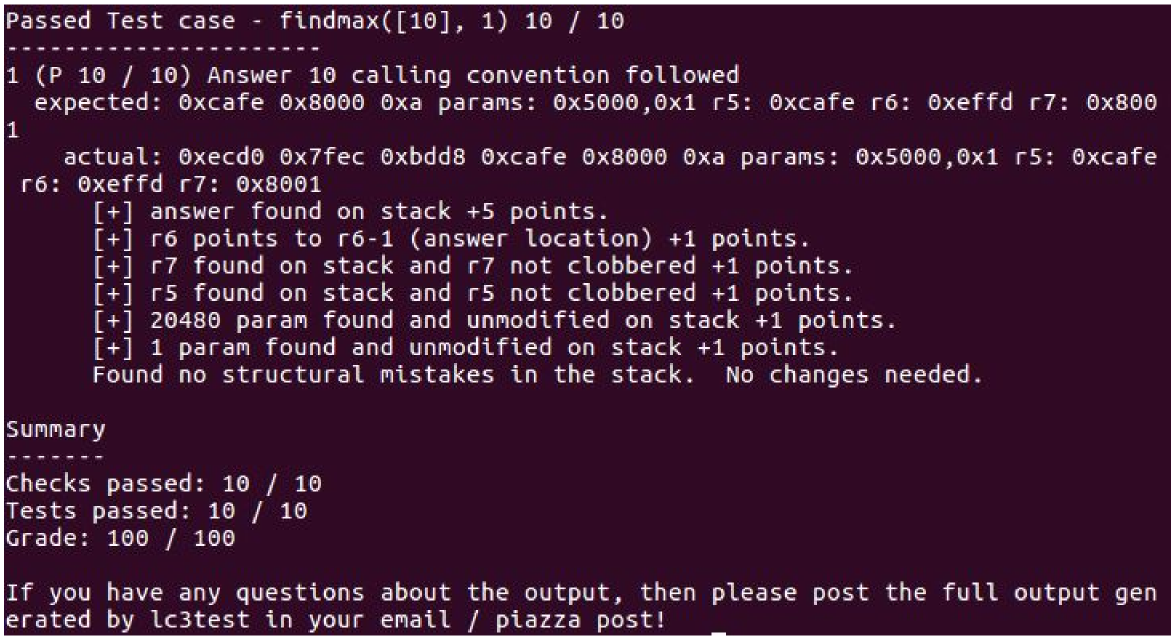
`[-] r7 found on stack and r7 not clobbered.`: This means that your R7 was either not found on the stack or did not hold the correct value.

`[-] r5 found on stack and r5 not clobbered.`: This means that your R5 (frame pointer) was either not found on the stack or did not hold the correct value.

`[+] 20480 param found and unmodified on stack.` : This means that the parameter `0x5000` was properly put on the stack.

`[+] 1 param found and unmodified on stack.` : This means that the parameter 1 was properly put on the stack.

If your code passes the test cases, the output should look something like the following:



```
Passed Test case - findmax([10], 1) 10 / 10
----------------------
1 (P 10 / 10) Answer 10 calling convention followed
  expected: 0xcafe 0x8000 0xa params: 0x5000,0x1 r5: 0xcafe r6: 0xeffd r7: 0x800
1
    actual: 0xecd0 0x7fec 0xbdd8 0xcafe 0x8000 0xa params: 0x5000,0x1 r5: 0xcafe
 r6: 0xeffd r7: 0x8001
      [+] answer found on stack +5 points.
      [+] r6 points to r6-1 (answer location) +1 points.
      [+] r7 found on stack and r7 not clobbered +1 points.
      [+] r5 found on stack and r5 not clobbered +1 points.
      [+] 20480 param found and unmodified on stack +1 points.
      [+] 1 param found and unmodified on stack +1 points.
      Found no structural mistakes in the stack.  No changes needed.

Summary
-------
Checks passed: 10 / 10
Tests passed: 10 / 10
Grade: 100 / 100

If you have any questions about the output, then please post the full output gen
erated by lc3test in your email / piazza post!
```

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.

6.2 LC-3 Calling Convention Guide

A handy assembly guide follows on the next page:

Boilerplate: Caller

before foo() returns - (3 arguments, 2 local variables, 2 saved registers)

```
Caller ADD R6, R6, -1 ; push(arg3)
       AND R0, R0, 0
       ADD R0, R0, 3
       STR R0, R6, 0
       ADD R6, R6, -1 ; push(arg2)
       AND R0, R0, 0
       ADD  R0, R0, 1
       STR R0, R6, 0
       ADD R6, R6, -1 ; push(arg1)
       LD  R0, X
       STR R0, R6, 0
       JSR FOO       ; foo()
       ...
```

```
m = foo(x, 1, 3)
  = x + 1 - 3
```

0000 ⇑

		SR2
		SR1
		lv 1
		lv 0
		OldFP
		RA
		RV
SP->	x	arg 1
SP->	1	arg 2
SP->	3	arg 3
SP->		

⇓ FFFF

Boilerplate: Caller

after foo() returns - (3 arguments, 2 local variables, 2 saved registers)

```
...
JSR FOO       ; foo()
LDR R0, R6, 0    ; m = RV
ST  R0, M
ADD R6, R6, 4    ; pop 4 words
```

```
; and the function call is done!
```

```
m = foo(x, 1, 3)
  = x + 1 - 3
```

0000 ⇑

	old R2	SR2
	old R1	SR1
		lv 1
		lv 0
	old R5	OldFP
	old R7	RA
~~SP->~~	x+1-3	RV
	x	arg 1
	1	arg 2
	3	arg 3
SP->		

⇓ FFFF

Boilerplate: Callee

(3 arguments, 2 local variables (lv), 2 saved registers (SR))

```
FOO   ADD R6, R6, -4  ; push 4 words
          ; for RV, RA, OldFP, lv0
                      ; set RV later
      STR R7, R6, 2   ; store RA
      STR R5, R6, 1   ; store OldFP
                 ; set lv0 & 1 later
      ADD R5, R6, 0   ; FP = SP
      ADD R6, R6, -3  ; push 3 words
                      ; 2+2-1(lv0)
      STR R1, R5, -2  ; save SR1
      STR R2, R5, -3  ; save SR2
      ... ; foo() implementation
```

0000 ⇑

SP->	old R2	SR2	FP-3
	old R1	SR1	FP-2
		lv 1	FP-1
~~SP->~~ FP->		lv 0	FP
	old R5	OldFP	FP+1
	old R7	RA	FP+2
		RV	FP+3
SP->	x	arg 1	FP+4
	1	arg 2	FP+5
	3	arg 3	FP+6

⇓ FFFF

`int foo(int a, int b, int c)`…

Boilerplate: Callee

(3 arguments, 2 local variables (lv), 2 saved registers (SR))

```
... ;foo() implementation
; And now we're ready to return
LDR R1, R5, -2  ; restore SR1
LDR R2, R5, -3  ; restore SR2
ADD R6, R5, 0   ; pop lv0/1 & SR1/2
LDR R7, R5, 2   ; R7 = RA
LDR R5, R5, 1   ; FP = OldFP
ADD R6, R6, 3   ; pop 3 words
RET             ; foo() is done!
```

0000 ⇑

~~SP->~~	old R2	SR2
	old R1	SR1
		lv 1
~~SP->~~ ~~FP->~~		lv 0
	old R5	OldFP
	old R7	RA
SP->	x+1-3	RV
	x	arg 1
	1	arg 2
	3	arg 3

⇓ FFFF

…
FP->

`int foo(int a, int b, int c)`…

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.

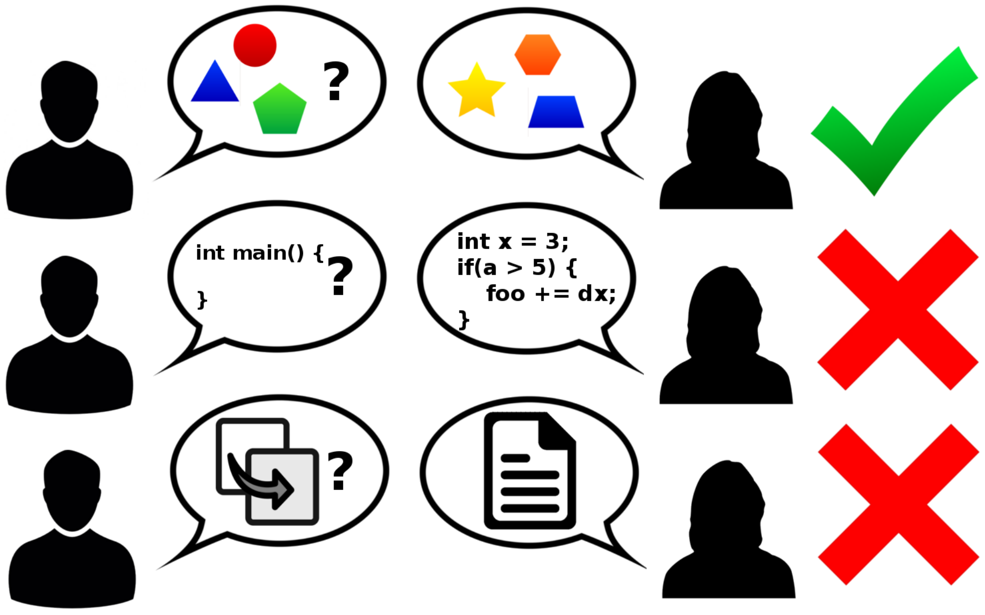


Figure 1: Collaboration rules, explained