CS 2110 Timed Lab 4: Assembly

Henry, Vivian, Youna, Lauren

Spring 2019

Contents

1 Timed Lab Rules - Please Read **2**
1.1 General Rules 2
1.2 Submission Rules 3
1.3 Is collaboration allowed? 3

2 Overview **3**
2.1 Description 3

3 Instructions **3**
3.1 countMult7 3
3.1.1 Example 1 4
3.1.2 Example 2 4
3.2 Notes 5
3.2.1 Checkpoint 1 5
3.2.2 Checkpoint 2 5
3.2.3 Result 5
3.3 Pseudocode 5
3.4 Restrictions 6

4 Rubric **6**
4.1 Breakdown 6
4.2 Local Autograder 6

5 Deliverables **7**

6 LC-3 Assembly Programming Requirements **7**
6.1 Overview 7

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

Note: You are not allowed to use a compiler on any assembly timed lab

1 Timed Lab Rules - Please Read

1.1 General Rules

1. You are allowed to submit this timed lab starting at the moment the assignment is released, until you are checked off by your TA as you leave the recitation classroom. Gradescope submissions will remain open until 7:15 pm - but you are not allowed to submit after you leave the recitation classroom under any circumstances. **Submitting or resubmitting the assignment after you leave the classroom is a violation of the honor code - doing so will automatically incur a zero on the assignment and might be referred to the Office of Student Integrity.**
2. Make sure to give your TA your Buzzcard before beginning the Timed Lab, and to pick it up and get checked off before you leave. **Students who leave the recitation classroom without getting checked off will receive a zero.**
3. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. **The information provided in this Timed Lab document takes precedence.** If in doubt, please make sure to indicate any conflicting information to your TAs.
4. Resources you are allowed to use during the timed lab:
    - Assignment files
    - Previous homework and lab submissions
    - Your mind
    - Blank paper for scratch work (please ask for permission from your TAs if you want to take paper from your bag during the Timed Lab)
5. Resources you are **NOT** allowed to use:
    - The Internet (except for submissions)
    - Any resources that are not given in the assignment
    - Textbook or notes on paper or saved on your computer
    - Email/messaging
    - Contact in any form with any other person besides TAs
    - Any compiler that outputs LC3 code
6. **Before you start, make sure to close every application on your computer.** Banned resources, if found to be open during the Timed Lab period, will be considered a violation of the Timed Lab rules.
7. We reserve the right to monitor the classroom during the Timed Lab period using cameras, packet capture software, and other means.

1.2 Submission Rules

1. Follow the guidelines under the Deliverables section.
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 Gradescope.
3. 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.

1.3 Is collaboration allowed?

Absolutely NOT. No collaboration is allowed for timed labs.

2 Overview

2.1 Description

In many high-level languages, including Java, C, and Python, you can have functions with a variable number of arguments. For these high level languages, the first argument is often the number of the remaining number of arguments. The remaining arguments can be visualized as an array of size `n` that is stored on the stack, instead of another region of memory. For this timed lab, you will be asked to implement a function, `countMult7`, that takes in a variable of number of arguments and returns a count of the arguments that are divisible by 7.

3 Instructions

3.1 countMult7

You should complete the `countMult7` subroutine. As in the description, `countMult7` is a bit different than normal subroutines as it takes in a variable number of arguments. The first argument, `n`, has a range of `[0, MAX_NUM_OF_ARGS]` and is the count of remaining arguments (`MAX_NUM_OF_ARGS` is just a very large constant that states how many arguments the subroutine could potentially accept). Again, as in the description, the remaining arguments can be visualized as an array that is stored on the stack instead of in another region of memory. This function should return a count of the arguments in our variable array divisible by the number 7. Below is a diagram of the stack to help you visualize how these arguments have been stored by the caller and a few examples to make sure you understand the concept.

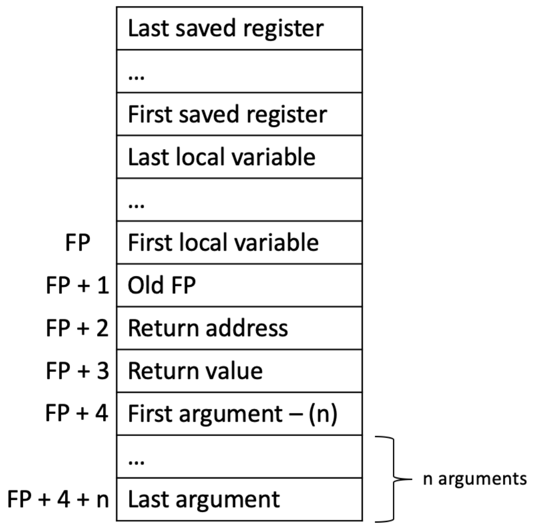

Figure 1: Example Stack – Very cool and useful!

3.1.1 Example 1

```
countMult7(4, 7, 3, 1, 14)
```

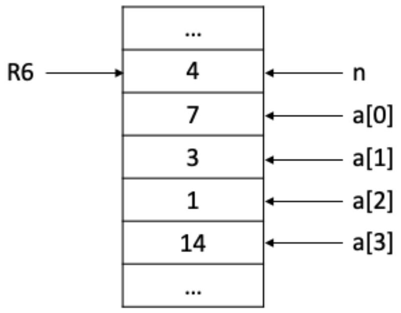

Figure 2: The argument portion of the stack frame for this example.

3.1.2 Example 2

```
countMult7(2, 7, 3)
```

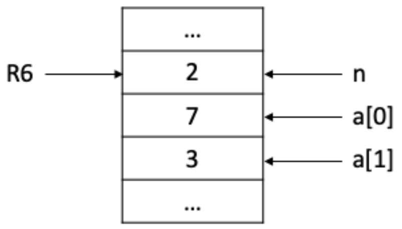

Figure 3: The argument portion of the stack frame for this example.

3.2 Notes

Like the last timed lab, we have a few checkpoints to help you along the way. You **will** be graded on these checkpoints, so be sure to do them!

3.2.1 Checkpoint 1

For the first checkpoint, grab the first argument off of the stack, `n`, and store it in `CHECKPOINT1`. This argument states how many remaining arguments there are.

3.2.2 Checkpoint 2

For the second checkpoint, grab the **address** of the first variable argument (i.e. the address of a[0]) and store it in `CHECKPOINT2`. In the case where `n` is 0, meaning there are no variable arguments, do not change the value at the `CHECKPOINT2` label.

3.2.3 Result

Now that you have the length of the variable arguments and the address of the first variable argument, you should sum up the number of variable arguments that are divisible by 7 and store this into the return value on the stack. We have provided you with a subroutine, `mod(a, b)`, that **returns 1 if** `a` **is divisible by** `b` **and 0 otherwise**. You should use this subroutine to determine if a number is divisible by 7 or points will be deducted. When calling this subroutine, be sure to think about what order the arguments should be pushed onto the stack!

3.3 Pseudocode

```
int mod(int a, int b); // provided subroutine header

int countMult7(n, a[0], a[1], ... a[n-1]) {
    int count = 0;
    int i = 0;

    CHECKPOINT1 = n;
    if (n > 0)
        CHECKPOINT2 = a; // store the *address* (not the value) of a[0] here
                         // note that a[0] is the second parameter

    while (n > 0) {
        int num = a[i];
        if (mod(num, 7) == 1) {
            count++;
        }
        n--;
        i++;
    }

    return count;
}
```

3.4 Restrictions

Use the calling convention taught in this course otherwise you will not receive full credit for your code. Do not remove or modify anything outside of the function you are being asked to implement. Doing so may break the autograder and result in you receiving a 0. If you feel you need to store something in memory, do not use labels, use local variables stored on the stack (except for the checkpoints).

4 Rubric

4.1 Breakdown

- Stack and Registers (50 points) – be sure to follow the calling convention taught in this course by implementing the proper "setup" and "teardown" for this subroutine.
- Checkpoint 1 (10 points)
- Checkpoint 2 (10 points)
- mod calls (15 points) – you must make the proper calls to the mod subroutine, otherwise points will be deducted.
- Return Value (15 points)

4.2 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, not in your browser**
  2. Run the command `sudo chmod +x grade.sh`
  3. Now run `./grade.sh`
- Windows Users:
  1. On **docker quickstart**, navigate to the directory your timed lab is in
  2. Run `./grade.sh`

The output of the autograder is an approximation of your score on this timed lab. It is a tool provided to students so that you can evaluate how much of the assignment expectations your submission fulfills. However, **we reserve the right to run additional tests, fewer tests, different tests, or change individual tests -** your final score will be determined by your instructors and no guarantee of tester output correlation is given.

5 Deliverables

Please upload the following files to Gradescope:

1. tl4.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.