CS 2110 Timed Lab 3 Make-up: Subroutines and Calling Conventions

John Ever, Justin Hinckley, Kevin Mankowski, Brian Yuan, Nick Ade, Dayne Bergman

Spring 2023

Contents

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 using Gradescope starting from 12:00 AM on the day this assignment is released until 11:59 PM on the same day. Submitting or resubmitting after this due date may constitute an honor code violation. The grade your receive for this assignment will replace the grade you received on the original Timed Lab if it is higher. Therefore this makeup assignment is optional; you do not need to submit anything if you are happy with the grade you received on the original Timed Lab.

The timed lab is open-resource. You may reference your previous homeworks, class notes, the textbook, etc., but your work must be your own. You may not submit anything written with the use of AI-based code completion software such as GitHub Copilot or ChatGPT. **No collaboration is allowed for timed labs.** This means you may not discuss this assignment with other students, nor publicly post any relevant information on Ed Discussion before the due date. Any Ed Discussion posts related to the Timed Lab must relate only to the logistics and instructions of the assignment, and not the content it tests.

If you have questions regarding this assignment, you may make an Ed Discussion post in the Timed Labs category. The TAs will attempt to answer all such questions in a timely manner, but in the event that your question is not answered, you are ultimately responsible for what you submit.

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

For this Timed Lab, you will be implementing three subroutines: MAX, DIV and MAP. MAX is a non-recursive subroutine that will return 0 if the first argument is larger than the second, and 1 if the second argument is at least as large as the first. DIV is a recursive subroutine that will divide the first argument by the second. MAP will iterate through an array, one pair of elements at a time, and replace the larger of the two elements with the first element divided by the second element.

3.1 `MAX` subroutine

MAX will take two arguments a and b and return 0 if a is greater than b, and 1 if b is greater than or equal to a.

Note: If a and b are equal, you should return 1 as demonstrated in the pseudocode.

You should implement the subroutine as shown in the pseudocode below:

```
// checkpoint 1
int MAX(int a, int b) {
    if (a > b) {
        return 0;
    } else {
        return 1;
    }
}
```

Examples:

- `MAX(2, 5)` returns 1

- `MAX(1, -4)` returns 0

- `MAX(3, 3)` returns 1

Implementing MAX will be the first checkpoint. Please refer to Checkpoints for details on how MAX: will be graded.

3.2 `DIV` subroutine

DIV will take two **non-negative** integer arguments x and y (y will also be non-zero) and compute x/y: that is, x divided by y.

Note: This is integer division, the resulting quotient is truncated. You should implement the subroutine as shown in the pseudocode below:

```
DIV(x, y) {
    // (checkpoint 2) - Base Case
    if (y > x) {
        return 0;
    // (checkpoint 3) - Recursion
    } else {
        return 1 + DIV(x - y, y);
    }
}
```

Examples:

- `DIV(9, 3)` returns 3

- `DIV(3, 5)` returns 0

- `DIV(26, 5)` returns 5

Note: All of the arguments for this function are guaranteed to be non-negative. `y` is also guaranteed to be non-zero.

`DIV` contains two checkpoints. Checkpoint 2 is the base case: returning 0 when `y > x`. Checkpoint 3 is the recursive case: `DIV` should return the correct value for all non-negative integers where `y > 0`. Please refer to Checkpoints for details on how `DIV` will be graded.

3.3 **MAP** subroutine

`MAP` takes two arguments:

- `array`: The address of the array to be modified

- `length`: The length of the array to be modified

Note: The argument `array` is guaranteed to have an even length. All the elements of `MAP` are guaranteed to be non-negative.

`MAP` will iterate through the array one pair of consecutive elements at a time. For each pair of elements, `MAP` will apply `DIV` to divide the first element by the second element and replace the larger of the two elements with the result. `MAP` will determine the index of the larger element by applying `MAX` to the pair and adding the result to the first element's index.

Note: The pairs that `MAP` iterates through should not overlap, meaning the first pair would be the elements at indices 0 and 1, the second pair would be the elements at indices 2 and 3, etc.

You should implement the subroutine as shown in the pseudocode below:

```
// (checkpoint 4)
void MAP(array, length) {
    int i = 0;
    while (i < length) {
        int firstElem = arr[i];
        int secondElem = arr[i + 1];
        int div = DIV(firstElem, secondElem);
        int offset = MAX(firstElem, secondElem);
        arr[i + offset] = div;
        i += 2;
    }
}
```

Examples:

- Before: `array = [12, 5];` `length = 2`

  - After: `array = [2, 5]`

- Before: `array = [8, 2, 0, 7, 12, 1, 5, 5];` `length = 8`

– After: `array = [4, 2, 0, 0, 12, 1, 5, 1]`

You do not need to return a value from `MAP`, you should instead modify `array` directly. While you will still leave a spot for a return value on the stack, it will be ignored.

Correctly implementing `MAP` is checkpoint 4. Please refer to Checkpoints for details on how `MAP` will be graded.

4 Checkpoints

4.1 Checkpoints (70 points)

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

- Checkpoint 1 (15 points): In MAX, load and compare the parameters a and b to return 0 when a > b and 1 when b >= a.

- Checkpoint 2 (15 points): Implement the base case for DIV to return 0 when y > x.

- Checkpoint 3 (20 points): Implement the recursive case of DIV to successfully compute and return x/y for any non-negative integer x and any positive integer y.

- Checkpoint 4 (20 points): Implement MAP. You should iterate through the array one pair of elements at a time, and use the MAX subroutine to determine the index of the larger element of the pair. You should then apply the DIV subroutine to the pair. The results of each DIV subroutine call must be properly stored back into the array at the index of the larger element.

4.2 Other Requirements (30 points)

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

- Your DIV subroutine must be recursive and call itself 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.

- During the execution of your subroutine, you must make the correct number of calls to MAX and DIV, corresponding to the pseudocode.

  - 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. tl03.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 homework 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 homework 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.

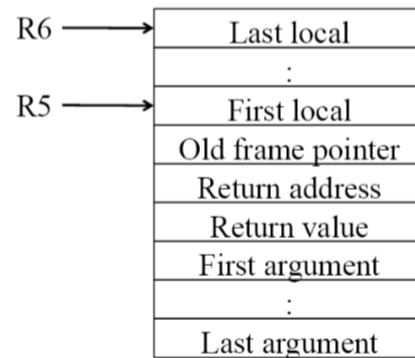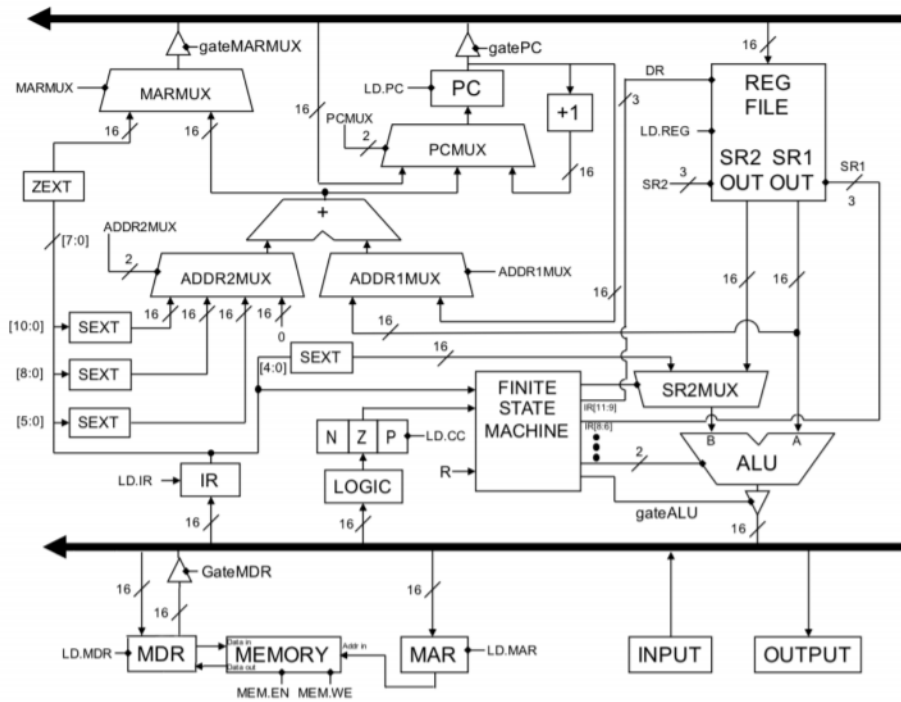11. **Test your assembly.** Don't just assume it works and turn it in.

8 Appendix

8.1 Appendix A: LC-3 Instruction Set Architecture

Instruction						
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

R6 →
| Last local |
| : |
R5 →
| First local |
| Old frame pointer |
| Return address |
| Return value |
| First argument |
| : |
| Last argument |

Boolean Signals	
LD.MAR	GateMARMUX
LD.MDR	GateMDR
LD.REG	GatePC
LD.CC	GateALU
LD.PC	LD.IR
MEM.EN	MEM.WE

MUX Name	Possible Values
ALUK	ADD, AND, NOT, PASSA
ADDR1MUX	PC, BaseR
ADDR2MUX	ZERO, offset6, PCoffset9, PCoffset11
PCMUX	PC+1, ADDER, BUS
MARMUX	ZEXT, ADDER
SR2MUX	SR2, SEXT