Introduction to Subroutines

Benjamin Kong — 1573684

Lora Ma ————— 1570935

ECE 212 Lab Section H11

March 30, 2020

Contents

1 Introduction

In a program, we often need to perform various sub-tasks many times. For example, we may need to sort numbers in an integer array. We could write the block of instructions that performs this sub-task over and over again, but this would be tedious and a waste of memory space. Therefore, we code these blocks of repeated instructions somewhere in memory and any time we need to use this block of instructions, we simply tell the program to branch to the location of the block of instructions. This block of instructions or sub-tasks are called *subroutines*. The instruction that branches to the subroutine is called a *call* instruction. Furthermore, the calling program is called the *caller* and the subroutine itself is called the *callee*. Once we reach the last instruction in a subroutine (called the *return* instruction), the subroutine returns to the program that called it.

In this lab, our objective was to gain experience using subroutines by writing subroutines for a statistics program. In specific, we wanted to gain experience

1. using the STACK (Push and Pop),
2. dividing up existing code into subroutines,
3. calling subroutines/functions, and
4. using basic parameter passing techniques.

The lab was broken up into three parts. For part A of the lab, we created a program that prompts the user to enter numbers using the keyboard. We also checked that the input met certain restrictions, namely

- the number of entries must be between 3 and 15,
- the divisor must be between 2 and 5, and
- the values entered must be positive.

For part B of the lab, we created a subroutine that, based on the input, finds the min, max, mean, and how many numbers were divisible by the divisor and what those numbers were.

For part C of the lab, we created a subroutine that displayed the results from part B on the MTTTY. Furthermore, we made the subroutine re-display all the numbers that the user input at the beginning.

2 Design

2.1 Part A

2.2 Part B

2.3 Part C

3 Testing

3.1 Part A

3.2 Part B

3.3 Part C

4 Questions

1. *Is it always necessary to implement either callee or caller preservation of registers when calling a subroutine? Why?*
2. *Is it always necessary to clean up the stack? Why?*
3. *If a proper check for the getstring function was not provided and you have access to the buffer, how*

would you check to see if a valid # was entered? A detailed description is sufficient. You do not need to implement this in your code.

5 Conclusion

The objective of this lab was to explore the use of subroutines for creating a statistics program.

In part A of the lab, we wrote a program that welcomed the user and promoted the user to enter numbers using the keyboard (using the MTTTY terminal). We checked that the inputs met certain requirements as mentioned in the introduction and design section. This part of the lab was successful as our code performed the required tasks.

In part B of the lab, we wrote a program that, based on the previous input, found the min, max, mean, how many numbers were divisible by the divisor, and what those numbers were. In part C of the lab, we finished our program and made it so that it displayed the values we found in part B. These two parts of the lab were completed successfully, as it was evident that the output values matched with the expected output.

6 Appendix

6.1 Part A Assembler Code

```
```

6.2 Part B Assembler Code

```
```

6.3 Part C Assembler Code

```
```