**ECE 212 Lab - Introduction to Microprocessors**

**Department of Electrical and Computer Engineering**

**University of Alberta**



**Lab 3: Subroutines**

| **Student Name** | **Student ID** |
| --- | --- |
| Marlene Gong | 1572719 |
| Darius Fang | 1570975 |

Table of Contents

[Introduction 3](#_gjdgxs)

[Design 4](#_30j0zll)

[Part A](#_1fob9te) 5

[Part A Flowchart](#_3znysh7) 5

[Part A Assembler Code](#_2et92p0) 6

[Part B](#_tyjcwt) 9

[Part B Flowchart](#_3dy6vkm) 10

[Part B Assembler Code](#_1t3h5sf) 11

[Part C](#_4d34og8) 13

[Part C Flowchart](#_2s8eyo1) 13

[Part C Assembler Code](#_17dp8vu) 14

[Questions](#_3rdcrjn) 16

[Conclusion](#_26in1rg) 17

# Introduction

This lab deals with stack operation (push and pop) and segmenting a long program/function into several smaller and simpler subroutines/sub-functions. The main objective of this lab was to familiarize students with these techniques along with making sure students were able to call on the subroutines and pass parameters between the main program and the subroutines. Each subroutine should have one entry point and one exit point and all registers used in the subroutines should be preserved and restored.

In Part A, students were asked to code a subroutine called *WelcomePrompt*; the subroutine would prompt the user to enter a series of numbers. The first number would be the number of entries in which only numbers between and including 3 to 15 would be accepted. The next number would be the divisor and only a minimum of 2 and a maximum of 5 can be allowed. Once a divisor is obtained, the user will then be asked to enter all of their entries one by one, all of which must be positive, non-zero numbers. If any of these restrictions are not met, the entered number will be rejected and the user will be asked to try again. All entries will be stored starting from the memory location 0x43000000 and the user will be informed that they are at their last entry when there is only one entry left to be entered.

In Part B, students had to code a subroutine called *Stats* in which the previously entered entries by the user would be analyzed. From the set of entries, the subroutine would pick out the min, max, mean, and the number of entries that were divisible by the divisor and what they were. The number of entries and the divisor were the input parameters. The number of entries that were divisible by the divisor would be the output parameter while the rest of the results were stored starting from the memory location 0x43100000.

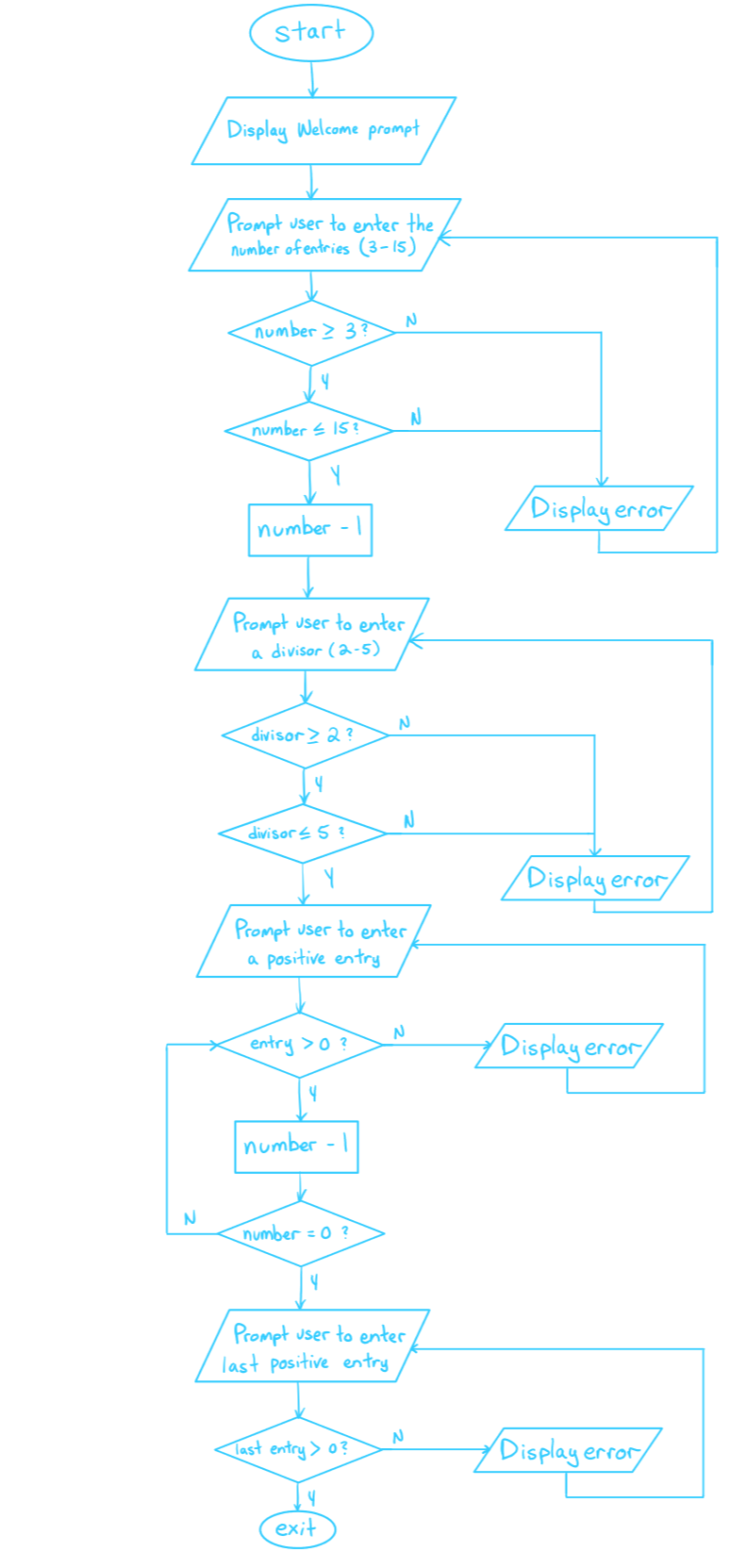
Finally, in Part C, students coded a subroutine called *Display.* It would display the number of entries entered by the user in Part A and what they were along with the analyzed results from Part B on MTTTY. The input parameters of this subroutine were the number of entries, the divisor, and how many entries were divisible by the divisor. Once all the results have been displayed, the program would display a final string informing the user of the end of the program.

# Design

## **Part A**

Part A subroutine mainly utilizes the user to input a bunch of numbers and helps us familiarize with strings, and how the stack is being used. The interface will pop up questions for the user to input. such as the number of numbers to store, a number to be a divisor, and positive numbers after that. If the user fails to give an integer within the range, the error message will display, and the question is repeated. In this design. There will be 3-15 numbers stored, for all being natural numbers, and a divisor with an integer between 2-5. the stack will store the number of entries, as well as the divisor that can be used later for part B. Address of the array of entries will be stored at 0x43000000

## **Part A Flowchart**



## **Part A Assembler Code**

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global WelcomePrompt

.extern iprintf

.extern cr

.extern value

.extern getstring

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab3a.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Darius Fang and Marlene Gong \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* IDs: 1570975 and 1572719 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Date: March 11, 2020 \*\*/

/\* General Description: Subroutine will prompt user to enter numbers \*\*/

/\* from the keyboard \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WelcomePrompt:

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sub.l #40, %a7 /\* preserve registers ... \*/

movem.l %a2-%a5/%d2-%d7, (%a7) /\* ...except for stack pointer \*/

move.l #0x43000000, %a2 /\* set a2 as memory location of values to be analyzed \*/

clr.l %d5 /\* clear d5 \*/

pea Greeting /\* push welcome string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

move.l #0, %d3 /\* clear d3 \*/

get\_entry:

move.l #0, %d5 /\* move 0 to d5 \*/

pea Entries /\* push number of entries string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

jsr getstring /\* jump to getstring subroutine, store user input in d0 \*/

move.l %d0, %d2 /\* copy user input into d2 \*/

move.l %d2, -(%sp) /\* push user input onto stack \*/

jsr value /\* jump to value subroutine, display user input \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

move.l %d2, 48(%sp) /\* replace 0xEEEEEEEE (from main program) with number of entries \*/

cmp.l #3, %d2 /\* compare number of entries to 3 \*/

blt error\_input /\* if less than 3, branch to error\_input \*/

cmp.l #15, %d2 /\* otherwise, compare number of entries to 15 \*/

ble get\_div /\* if less than or equal to 15, branch to get\_div \*/

bra error\_input /\* otherwise, branch to error\_input \*/

get\_div:

move.l #1, %d5 /\* move 1 to d5 \*/

move.l %d2, %d4 /\* copy number of entries to d4 \*/

sub.l #1, %d4 /\* subtract number of entries by 1 \*/

pea Divisor /\* push divisor string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

jsr getstring /\* jump to getstring subroutine, store user input in d0 \*/

move.l %d0, %d2 /\* copy user input into d2 \*/

move.l %d2, -(%sp) /\* push user input onto stack \*/

jsr value /\* jump to value subroutine, display user input \*/

add.l #4, %sp /\* clean up stack \*/

move.l %d2, 44(%sp) /\* replace 0xDDDDDDDD (from main program) with divisor \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

cmp.l #2, %d2 /\* compare divisor to 2 \*/

blt error\_input /\* if less than 2, branch to error\_input \*/

cmp.l #5, %d2 /\* otherwise, compare divisor to 5 \*/

ble get\_posNum /\* if less than or equal to, branch to get\_posNum \*/

bra error\_input /\* otherwise, branch to error\_input \*/

get\_posNum:

move.l #2, %d5 /\* move 2 to d5 \*/

pea posNumber /\* push positive number string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

jsr getstring /\* jump to getstring subroutine, store user input in d0 \*/

move.l %d0, %d2 /\* copy user input into d2 \*/

move.l %d2, -(%sp) /\* push user input onto stack \*/

jsr value /\* jump to value subroutine, display user input \*/

addq.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

cmp.l #0, %d2 /\* compare number to 0 \*/

ble error\_input /\* if less than or equal to 0, branch to error\_input \*/

move.l %d2, (%a2)+ /\* otherwise, copy number to memory location of values to be analyzed, increment a2 \*/

sub.l #1, %d4 /\* subtract number of entries by 1 \*/

bne get\_posNum /\* if number of entries doesn't equal zero, branch to get\_posNum \*/

bra get\_lastNum /\* otherwise, branch to get\_lastNum \*/

get\_lastNum:

move.l #3, %d5 /\* move 3 to d5 \*/

pea lastNumber /\* push last number string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

jsr getstring /\* jump to getstring subroutine, store user input in d0 \*/

move.l %d0, %d2 /\* copy user input into d2 \*/

move.l %d2, -(%sp) /\* push user input onto stack \*/

jsr value /\* jump to value subroutine, display user input \*/

addq.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

cmp.l #0, %d2 /\* compare last number to 0 \*/

ble error\_input /\* if less than or equal to 0, branch to error\_input \*/

move.l %d2, (%a2)+ /\* otherwise, copy number to memory location of values to be analyzed, increment a2 \*/

bra end /\* branch to end \*/

error\_input:

pea Invalid /\* push invalid entry string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

addq.l #4, %sp /\* clean up stack \*/

cmp.l #0, %d5 /\* compare d5 to 0 \*/

beq get\_entry /\* if equal, branch to get\_entry \*/

cmp.l #1, %d5 /\* otherwise, compare d5 to 1 \*/

beq get\_div /\* if equal, branch to get\_div \*/

cmp.l #2, %d5 /\* otherwise, compare d5 to 2 \*/

beq get\_posNum /\* if equal, branch to get\_posNum \*/

cmp.l #3, %d5 /\* otherwise, compare d5 to 3 \*/

beq get\_lastNum /\* if equal, branch to get\_lastNum \*/

end:

movem.l (%a7), %a2-%a5/%d2-%d7 /\* restore registers \*/

add.l #40, %a7 /\* stack pointer points at return address \*/

rts /\* return to main program \*/

/\*End of Subroutine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.data

/\*All Strings placed here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Greeting:

.string "Welcome to Wing's Stats Program"

Entries:

.string "Please enter the number(3min-15max) of entries followed by 'enter'"

Divisor:

.string "Please enter the divisor(2min-5max) followed by 'enter'"

posNumber:

.string "Please enter a number(positive only)"

lastNumber:

.string "Please enter the last number(positive only)"

Invalid:

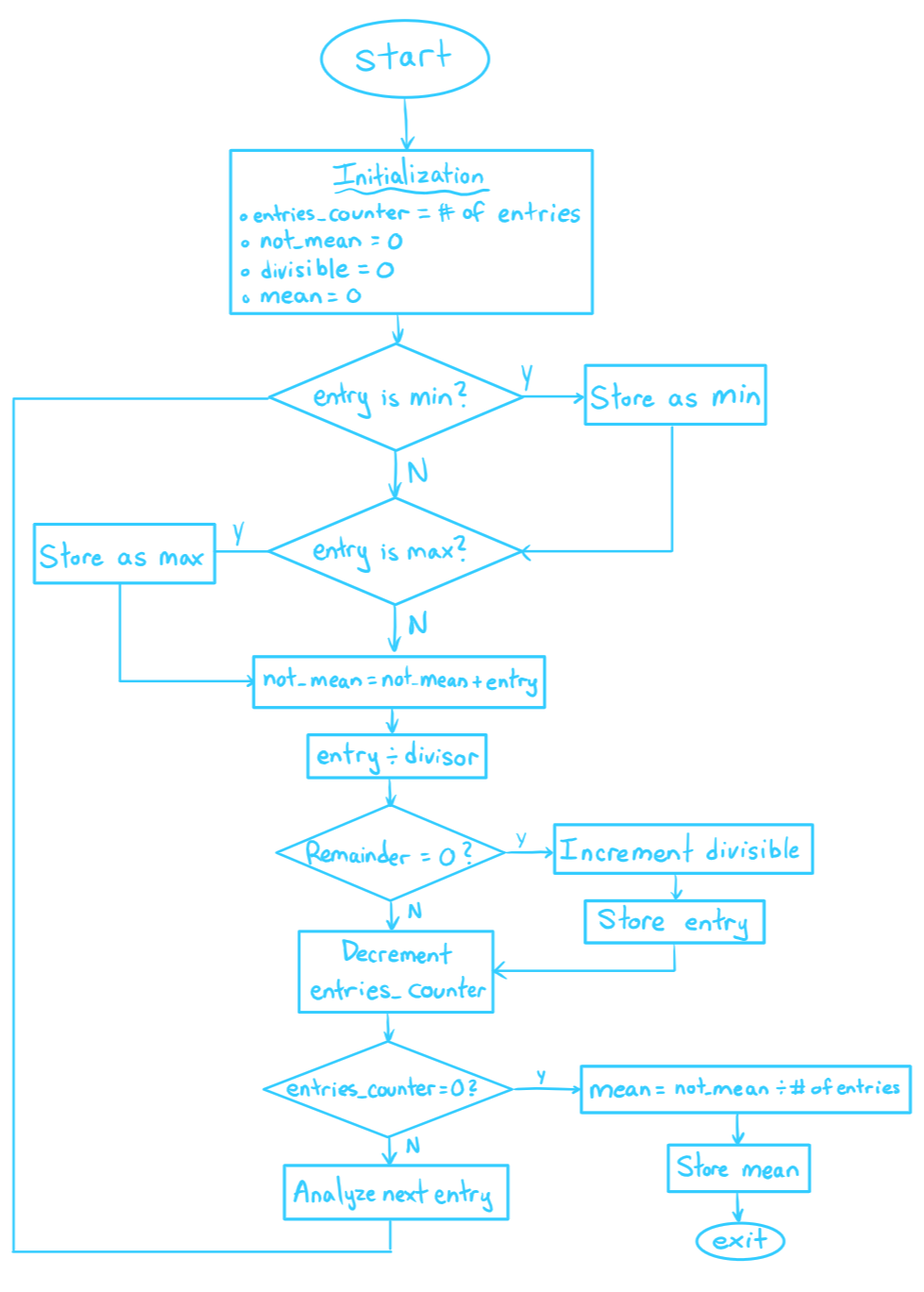
.string "Invalid entry, please enter proper value."

/\*End of Strings \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## **Part B**

Part B’s subroutine mainly focuses on interpreting the stored data to find a bunch of values, such as mean, min, max. The design we made, is so that it calculates all the values at that point of the array. For example the value in the array is compared to the min, then compared to the max, then is added to the mean value, and finally compares to the divisor. If the value is smaller than the min, that is the new min. If the value is bigger than the max, then the max will be that value. If the value is divided by the divisor to be 0, the value will be stored to another location 0x4310000C, and a counter the amount with divided goes up. It will also add itself to the mean. The array iterated through will be the number of entries obtained from the stack. The mean is calculated after the array has iterated through and will be divided by the number of entries. What will be returned on the stack is the number divisible, and min, max, mean will be stored 0x43100000, 0x43100004, 0x43100008 respectively.

## **Part B Flowchart**



## **Part B Assembler Code**

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global Stats

.extern iprintf

.extern cr

.extern value

.extern getstring

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab3b.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Darius Fang and Marlene Gong \*\*\*\*\*\*\*/

/\* IDs: 1570975 and 1572719 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Date: March 11, 2020 \*\*/

/\* General Description: Subroutine will find the min, max, \*\*/

/\* mean, how many numbers were divisible by the divisor \*\*/

/\* and what are they from the numbers entered \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Stats:

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sub.l #40, %a7 /\* preserve registers ... \*/

movem.l %a2-%a5/%d2-%d7, (%a7) /\* ...except for stack pointer \*/

/\* substitute values for labels \*/

.equ Min, 0

.equ Max, 4

.equ Mean, 8

.equ Divisible, 12

/\*

a2: array index (values to be analyzed), a3: results

d2: divisable

d3: entries

d4: dummy variable

d5: number of div

d6: counter

\*/

move.l %d3, %d6 /\* copy the number of entries to a counter \*/

clr.l %d5 /\* clear d5 \*/

clr.l Mean(%a3) /\* clear contents of 0x43100008 \*/

move.l (%a2), %d4 /\* copy first entry to d4 \*/

move.l %d4, Min(%a3) /\* copy entry into 0x43100000 \*/

move.l %d4, Max(%a3) /\* copy entry into 0x43100004

Loop:

move.l (%a2)+, %d4 /\* copy entry to d4, increment pointer to next entry \*/

cmp.l Min(%a3), %d4 /\* compare entry to contents of 0x43100000 \*/

blt min /\* if less than, branch to min \*/

Return1:

cmp.l Max(%a3), %d4 /\* otherwise, compare entry to contents of 0x43100004 \*/

bgt max /\* if greater than, branch to max \*/

Return2:

add.l %d4, Mean(%a3) /\* otherwise, add entry to contents of 0x43100008 \*/

divu.w %d2, %d4 /\* divide entry by divisor \*/

lsr.l #8, %d4 /\* shift result by 8 bits \*/

lsr.l #8, %d4 /\* shift result by 8 bits \*/

beq div /\* if remainder is equal to 0, branch to div \*/

Return3:

sub.l #1, %d6 /\* decrement counter \*/

bne Loop /\* if counter isn’t 0, branch to Loop \*/

bra end /\* otherwise, branch to end \*/

min:

move.l %d4, Min(%a3) /\* copy d4 into 0x43100000 \*/

bra Return1 /\* branch to Return 1 \*/

max:

move.l %d4, Max(%a3) /\* copy d4 into 0x43100004 \*/

bra Return2

div:

move.l %d5, %d4 /\* copy number of divisible entries to d4 \*/

muls.w #4, %d4 /\* multiply d4 by 4 \*/

move.l -(%a2), Divisible(%a3, %d4) /\* point at previous entry and copy to index in 0x4310000C \*/

add.l #4, %a2 /\* point at next entry \*/

add.l #1, %d5 /\* add 1 to number of divisible entries \*/

bra Return3

end:

move.l Mean(%a3), %d4 /\* copy contents of 0x43100008 to d4 \*/

divu.l %d3, %d4 /\* divide d4 by the number of entries \*/

move.l %d4, Mean(%a3) /\* copy results into 0x43100008 \*/

move.l %d5, 52(%sp) /\* replace 0xFFFFFFFF (from main program) with number of divisible entries\*/

movem.l (%a7), %a2-%a5/%d2-%d7 /\* restore registers \*/

add.l #40, %a7 /\* stack pointer points at return address \*/

rts /\* return to main program \*/

/\*End of Subroutine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.data

/\*All Strings placed here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*End of Strings \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## **Part C**

Part C makes a subroutine to output our results. It will display the number of entries stored from the stack, then iterates through the array to display each value, it will then display the min number, the max number, the mean, and number of entries that were divisible by the divisor. Lastly it will iterate to the array at 0x4310000C and display the numbers that were divisible

## **Part C Flowchart**

## **Part C Assembler Code**

/\* DO NOT MODIFY THIS --------------------------------------------\*/

.text

.global Display

.extern iprintf

.extern cr

.extern value

.extern getstring

/\*----------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* General Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* File Name: Lab3c.s \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Names of Students: Darius Fang and Marlene Gong \*\*\*\*\*\*\*\*\*/

/\* IDs: 1570975 and 1572719 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Date: March 11, 2020 \*\*/

/\* General Description: Subroutine will display the min, max, \*\*/

/\* mean, how many numbers were divisible by the divisor \*\*/

/\* and what are they on MTTTY \*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Display:

/\*Write your program here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sub.l #40, %a7 /\* preserve registers ... \*/

movem.l %a2-%a5/%d2-%d7, (%a7) /\* ...except for stack pointer \*/

pea numEntries /\* push number of entries string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

addq.l #4, %sp /\* clean up stack \*/

move.l %d3, -(%sp) /\* push number of entries onto stack \*/

jsr value /\* jump to value subroutine, display number of entries \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

add.l #4, %sp /\* clean up stack \*/

loop:

move.l (%a2)+, %d5 /\* copy entry into d5, increment to next entry \*/

move.l %d5, -(%sp) /\* push entry onto stack \*/

jsr value /\* jump to value subroutine, display entry \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

sub.l #1, %d3 /\* subtract 1 from the number of entries \*/

bne loop /\* if number of entries isn’t 0, branch to loop \*/

movea.l #0x43100000, %a2 /\* copy 0x43100000 to a2 \*/

pea minNum /\* push min number string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l (%a2)+, %d3 /\* copy min number to d3, increment to max number \*/

move.l %d3, -(%sp) /\* push min number onto stack \*/

jsr value /\* jump to value subroutine, display min number \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

pea maxNum /\* push max number string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l (%a2)+, %d3 /\* copy max number to d3, increment to mean number \*/

move.l %d3, -(%sp) /\* push max number onto stack \*/

jsr value /\*jump to value subroutine, display max number \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

pea meanNum /\* push mean number string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l (%a2)+, %d3 /\* copy mean number to d3, increment to first divisible number \*/

move.l %d3, -(%sp) /\* push mean number onto stack \*/

jsr value /\* jump to value subroutine, display mean number \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

pea ThereAre /\* push “There are” string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l %d4, -(%sp) /\* push number of divisible entries onto stack \*/

jsr value /\* jump to value subroutine, display number of divisible entries \*/

add.l #4, %sp /\* clean up stack \*/

pea divisible /\* push “ number(s) divisible by “ string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l %d2, -(%sp) /\* push divisor onto stack \*/

jsr value /\* jump to value subroutine, display divisor \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

pea TheyAre /\* push “They are” string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

divLoop:

pea space /\* push “ “ string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

move.l (%a2)+, %d3 /\* copy divisible number to d3, increment to next divisible number \*/

move.l %d3, -(%sp) /\* push divisible number onto stack \*/

jsr value /\* jump to value subroutine, display divisible number \*/

add.l #4, %sp /\* clean up stack \*/

sub.l #1, %d4 /\* subtract 1 from number of divisible entries \*/

bne divLoop /\* if number of divisible entries isn’t 0, branch to divLoop \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

pea end /\* push program end string onto stack \*/

jsr iprintf /\* jump to iprintf subroutine, print string \*/

add.l #4, %sp /\* clean up stack \*/

jsr cr /\* jump to cr subroutine, generate carriage return and linefeed \*/

movem.l (%a7), %a2-%a5/%d2-%d7 /\* restore registers \*/

add.l #40, %a7 /\* stack pointer points at return address \*/

rts /\* return to main program \*/

/\*End of Subroutine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.data

/\*All Strings placed here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

numEntries:

.string "The number of entries was "

minNum:

.string "Min number = "

maxNum:

.string "Max number = "

meanNum:

.string "Mean number = "

ThereAre:

.string "There are "

divisible:

.string " number(s) divisible by "

TheyAre:

.string “They are”

space:

.string “ “

end:

.string "Program ended"

/\*End of Strings \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# Questions

1. *Is it always necessary to implement either callee or caller preservation of registers when calling a subroutine. Why?*

**Answer:**

Calling a subroutine helps recall doing a specific task in mind with different parameters, but once finishing the task is over, returning with the original data registers is important to use after the subroutine is finished since the main code, or the current code that the program is working on is not finished.

1. *Is it always necessary to clean up the stack. Why?*

**Answer:**

When the stack is cleaned up, you cannot access the values below the stack since it is not guaranteed, but it is necessary to clean up the stack since you run into trap errors and that prevents the program from continuing. Avoid trap errors since it is memory the user is not supposed to access at all. Is also to know that values that are supposed to be outside of the scope when can have a random value that will affect the program.

1. *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*.*

**Answer:**

A way to check would be to first retrieve the data from the buffer and convert it to its ASCII value. Then, compare the value with the ASCII table to see if the value falls within the acceptable range. In hex, valid numbers would be from 0x30 to 0x39; any other numbers would be invalid.

# Conclusion

The point of this lab was to ensure that students knew how to code subroutines using stack operations. Each part of the lab had their own subroutine associated with it. In Part A, we had to make the program prompt the user to enter a number for the amount of entries, the said entries, and a divisor. For Part B, the numbers obtained from Part A would be analyzed and the program would identify and store the numbers that were the min, max, mean, and the numbers that were divisible by the divisor. Part C would display the analyzed results to the user and that was the end of the program. In order to carry out this lab, we used instructions like JSR and PEA to jump to subroutines and to push effective addresses onto the stack, respectively. We were able to run our program on MTTTY and ended up obtaining the results that were expected.