Introduction to Subroutines

Benjamin Kong — 1573684
Lora Ma ———— 1570935

ECE 212 Lab Section H11

March 30, 2020

Contents

1 Introduction **1**

2 Design **1**

2.1 Part A 1

2.2 Part B 1

2.3 Part C 1

3 Testing **1**

3.1 Part A 1

3.2 Part B 2

3.3 Part C 2

4 Questions **2**

5 Conclusion **2**

6 Appendix **3**

6.1 Part A Assembler Code 3

6.2 Part B Assembler Code 6

6.3 Part C Assembler Code 8

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

Due to the shutdown of the university, we were unable to test our code in-person. However, we have outlined how we would have tested our code below for each section.

3.1 Part A

For part A, we

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?*

   It is always necessary. A subroutine may be running in several different parts of the program and the data and address registers being used by the subroutine could be overwritten. If the program was using these data or address registers to store information and this information gets overwritten by the subroutine, it could cause errors or cause unexpected behavior.

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

   It is always necessary. Let's say the subroutine only restores old register values, but doesn't clean up the stack. If the program that called the subroutine relies on information from the stack, the offset of the stack pointer may not be correct and the program would not be accessing the correct information.

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.*

   In order to check if a valid # was input, our program would iterate through each element in the buffer. For each element, we would check to see if the character was between '0' and '9'; this could be accomplished using branches. If the input was not valid, a message could be output signifying an invalid entry and a prompt could be displayed to have the user re-enter a valid input.

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

```
WelcomePrompt:
/*Write your program here*****************************************/

/* Backup registers */
lea -40(%sp), %sp
movem.l %d2-%d7/%a2-%a5, (%sp)

/* Print Welcome message */
Welcomemsg:
  pea Welcome /*Push string to stack*/
  jsr iprintf /*Display string */
  adda.l #4, %sp /*Pop string location from stack*/
  jsr cr /*New Line*/

Entries:
  pea EntriesPrompt /*Push string to stack*/
  jsr iprintf /*Display string */
  adda.l #4, %sp /*Pop string location from stack*/
  jsr cr /*New Line*/
  jsr getstring /*Get user input*/
  bra CheckEntry /*Check entry*/

CheckEntry:
  cmpi.l #15, %d0 /* Compare d0 with 15 */
  bgt InvalidEntry /* If greater than 15, go to InvalidEntry */
  cmpi.l #3, %d0 /* Compare d0 with 3 */
  blt InvalidEntry /* if less than 3, go to InvalidEntry */
  cmpi.l #0,  %d0 /* compare d0 with 0 */
  beq exit /*If equal to zero, exit */
  move.l %d0, 48(%sp) /* Move number of entries onto stack */
  move.l %d0, %d2 /* Create counter from input */

InvalidEntry:
  pea Invalid /*Push string to stack*/
  jsr iprintf /*Display string */
  adda.l #4, %sp /*Pop string location from stack*/
  jsr cr /*New Line*/
  jsr getstring /*Get user input*/

/* Ask for divisor */
```

```
41  Divisormsg:
42    pea  DivisorPrompt /*Push string to stack*/
43    jsr iprintf /*Display string */
44    adda.l #4, %sp /*Pop string location from stack*/
45    jsr cr /*New Line*/
46    jsr getstring /*Get user input*/
47    bra CheckDiv
48
49  InvalidDiv:
50    pea Invalid /*Push string to stack*/
51    jsr iprintf /*Display string */
52    adda.l #4, %sp /*Pop string location from stack*/
53    jsr cr /*New Line*/
54    jsr getstring /*Get user input*/
55
56  CheckDiv:
57    cmpi.l #2, %d0 /*Compare d0 with 2*/
58    blt InvalidDiv /*if less than, go to InvalidDiv*/
59    cmpi.l #5, %d0 /*Compare d0 with 5*/
60    bgt InvalidDiv /*If greater than 5, go to InvalidDiv*/
61    move.l %d0, 44(%sp) /* Move divisor number onto stack */
62
63  movea.l     #0x43000000, %a2 /* Pointer to array storing data */
64  Numbermsg:
65    pea NumberPrompt /*Push string to stack*/
66    jsr iprintf /*Display string */
67    adda.l #4, %sp /*Pop string location from stack*/
68    jsr cr /*New Line*/
69    jsr getstring /*Get user input*/
70    bra CheckNum /* go to CheckNum */
71
72  InvalidNum:
73    pea Invalid /*Push string to stack*/
74    jsr iprintf /*Display string */
75    adda.l #4, %sp /*Pop string location from stack*/
76    jsr cr /*New Line*/
77    jsr getstring /*Get user input*/
78
79  CheckNum:
80    cmpi.l #0, %d0 /* Check if negative number entered */
81    blt InvalidNum /*Invalid if negative*/
82    move.l %d0, (%a2)+ /*If positive, move to array, increment pointer*/
83    sub.l #1, %d2 /*Decrement counter*/
84    cmpi.l #1, %d2
85    beq LastNummsg /*If last number, go to LastNummsg*/
```

```
86    bra Numbermsg /*Otherwise, check next number*/
87
88  LastNummsg:
89    pea LastNumPrompt /*Push string to stack*/
90    jsr iprintf /*Display string */
91    adda.l #4, %sp /*Pop string location from stack*/
92    jsr cr /*New Line*/
93    jsr getstring /*Get user input*/
94    bra LastCheck /*Go to LastCheck*/
95
96  InvalidLast:
97    pea Invalid /*Push string to stack*/
98    jsr iprintf /*Display string */
99    adda.l #4, %sp /*Pop string location from stack*/
100   jsr cr /*New Line*/
101   jsr getstring /*Get user input*/
102
103 LastCheck:
104   cmpi.l #0, %d0
105   blt InvalidLast
106   move.l %d0, (%a2)
107
108 exit:
109   /* Restore registers */
110   movem.l %d2-%d7/%a2-%a5, (%sp)
111   lea 40(%sp),%sp
112
113 rts
114
115 /*End of Subroutine ***********************************************s***/
116 .data
117 /*All Strings placed here ***************************************************/
118
119 Welcome:
120 .string "Welcome to Wing's Stats Program"
121
122 EntriesPrompt:
123 .string "Please enter the number (3min-15max) of entries followed by 'enter'"
124
125 DivisorPrompt:
126 .string "Please enter the divisor (2min-5max) followed by 'enter'"
127
128 NumberPrompt:
129 .string "Please enter a number (positive only)"
130
```

```
131 LastNumPrompt:
132 .string "Please enter the last number (positive only)"
133
134 Invalid:
135 .string "Invalid entry, please enter a proper value."
136
137 /*End of Strings ***********************************************/
```

6.2 Part B Assembler Code

```
1  Stats:
2  /*Write your program here**************************************/
3
4  /* Backup registers */
5  lea -40(%sp),%sp
6  movem.l %d2-%d7/%a2-%a5, (%sp)
7
8  FindMin:
9    move.l (%a2)+, %d2
10   move.l 48(%sp), %d4 /*Initialize counter*/
11
12 LoopMin:
13   sub.l #1, %d4 /* Decrement counter */
14   move.l (%a2)+, %d3 /* Load Next number to test */
15   cmp.l #0, %d4 /* Check if counter done */
16   ble FindMaxVal /* If counter done, go to FindMaxVal */
17   cmp.l %d3, %d2 /* Compare stored and new number */
18   blt LoopMin /* If stored is smaller, repeat checks */
19   move.l %d3, %d2 /* If stored is larger, save new number */
20   bra LoopMin /* Repeat loop */
21
22 FindMaxVal:
23   move.l %d2, (%a3)+ /* Save min number */
24   movea.l #0x43000000, %a2 /* Reinitialize counter to data array */
25   move.l (%a2)+, %d2 /* Load First number */
26   move.l 48(%sp), %d4 /* Reload Counter */
27
28 LoopMax:
29   sub.l #1, %d4 /* Decrement counter */
30   move.l (%a2)+, %d3 /* Load Next number to test */
31   cmp.l #0, %d4 /* Check if counter done */
32   ble FindMeanVal /* If counter done, go to FindMaxVal */
33   cmp.l %d3, %d2 /* Compare stored and new number */
34   bgt LoopMax /* If stored is larger, repeat checks */
35   move.l %d3, %d2 /* If stored is smaller, save new number */
```

```
36    bra LoopMax /* Repeat loop */
37
38  FindMeanVal:
39    move.l %d2, (%a3)+ /* Save max number */
40    movea.l #0x43000000, %a2 /* Reinitialize counter to data array */
41    move.l 48(%sp), %d3 /* Reload Counter */
42    clr.l %d2 /* Clear D5 to use as cum sum */
43
44  LoopMean:
45    add.l (%a2)+, %d2 /*Add num at address register to d2 and increment a2*/
46    sub.l #1, %d3 /*Subtract o1 from d3*/
47    bne LoopMean /*If not equal to zero, go to LoopMean*/
48    divu.l 48(%sp), %d2 /*Divide total sum by amount of integers*/
49    move.l %d2, (%a3)+ /*Move mean to output array*/
50
51  move.l 48(%sp), %d3 /*Reload Counter*/
52  movea.l #0x43000000, %a2 /*Reinitialize counter to data array*/
53  move.l #4, %d5 /*Move #4 to d5*/
54  clr.l %d6 /*Divisor counter*/
55
56  FindDivisible:
57    move.l (%a2)+, %d2 /*Move data at address a2 to d2 and increment a2*/
58    move.l %d2, %d4 /*Move data at d2 to d4*/
59    move.l 44(%sp), %d7 /*Move stack pointer*/
60    divu.w %d7, %d4 /*lower word contains quotient - higher word has remainder*/
61
62  Remainder:
63    and.l #0xFFFF0000, %d4
64    cmp.l #0, %d4 /*Compare #0 with d4*/
65    bne   NotDivisible /*If not equal, go to NotDivisible*/
66    move.l %d2,(%a3)+ /*Move value at d2 to address a3 and increment*/
67    add.l #1, %d6     /*Increment divisor counter*/
68    sub.l #1,%d3 /*Subtract #1 from d3*/
69    bne FindDivisible /*if not equal to zero, go to FindDivisible*/
70    bra exit /*else, exit*/
71
72  NotDivisible:
73    sub.l #1, %d3 /*Subtract 1 from d3*/
74    bne FindDivisible /*If not equal to zero, go to FindDivisible*/
75
76  exit:
77    move.l %d6, 52(%sp)
78
79
80  movem.l (%sp), %d2-%d7/%a2-%a5
```

```
81  lea 40(%sp), %sp
82  rts
83
84  /*End of Subroutine ************************************************/
85  .data
86  /*All Strings placed here ******************************************/
87
88
89
90  /*End of Strings ***************************************************/
```

6.3 Part C Assembler Code

```
1   Display:
2   /*Write your program here*************************************/
3   lea -40(%sp), %sp
4   movem.l %d2-%d7/%a2-%a5, (%sp)
5
6   pea NumPrompt /* Push location of string to stack */
7   jsr iprintf /* Print out string */
8   adda.l #4, %sp /* Clean up stack */
9
10  move.l 48(%sp), %d2 /* Load number */
11  move.l %d2, -(%sp) /* Store to stack */
12  jsr value /* Print number */
13  adda.l #4, %sp /* Clean up stack */
14  jsr cr /* Print newline */
15  clr.l %d3 /* Prepare counter */
16
17  Loop: /* Counter print loop *?
18    move.l (%a2, %d3*4), -(%sp) /* Load number to stack, using counter as index */
19    jsr value /* Print number */
20    jsr cr /* Print newline */
21    adda.l #4, %sp /* Remove number from stack */
22    add.l #1, %d3 /* Increment counter */
23    cmp.l %d2, %d3 /* Check if done */
24    blt Loop /* repeat if not */
25
26  pea MinPrompt /* Push string to stack */
27  jsr iprintf /* Print out string */
28  adda.l #4, %sp /* pop stack */
29  move.l (%a3)+, -(%sp) /* Load smallest number to stack */
30  jsr value /* Print out number to stack */
31  adda.l #4, %sp /* Pop stack */
32  jsr cr /* Print newline */
```

```
33
34  pea MaxPrompt /* Push string to stack */
35  jsr iprintf /* Print out string */
36  adda.l #4, %sp /* pop stack */
37  move.l (%a3)+, -(%sp) /* Load largest number to stack */
38  jsr value /* Print out number to stack */
39  adda.l #4, %sp /* Pop stack */
40  jsr cr /* Print newline */
41
42  pea MeanPrompt /* Push string to stack */
43  jsr iprintf /* Print out string */
44  adda.l #4, %sp /* pop stack */
45  move.l (%a3), -(%sp) /* Load average number to stack */
46  jsr value /* Print out number to stack */
47  adda.l #4, %sp /* Pop stack */
48  jsr cr /* Print newline */
49
50  pea Divisible1 /* Push string to stack */
51  jsr iprintf /* Print out string */
52  adda.l #4, %sp /* pop stack */
53  move.l 52(%sp), %d2 /* Load number to register */
54  move.l %d2, -(%sp) /* Push number to stack */
55  jsr value /* Print out number to stack */
56  adda.l #4, %sp /* pop stack */
57
58  pea Divisible2 /* Push string to stack */
59  jsr iprintf /* Print out string */
60  adda.l #4, %sp /* pop stack */
61  move.l 44(%sp), %d2 /* Load number to register */
62  move.l %d2, -(%sp) /* Push number to stack */
63  jsr value /* Print out number to stack */
64  adda.l #4, %sp /* pop stack */
65
66  jsr cr /* Print newline */
67
68  pea EndText /* Push end text to stack */
69  jsr iprintf /* Print text */
70  adda.l #4, %sp /* Pop stack */
71
72  jsr cr /* Print newline */
73
74  movem.l (%sp), %d2-%d7/%a2-%a5 /* Restore registers */
75  lea 40(%sp), %sp /* Reset stack pointer */
76
77  rts
```

```
78
79  /*End of Subroutine ************************************************/
80  .data
81  /*All Strings placed here ******************************************/
82  NumEntriesText:
83  .string "The number of entries was "
84
85  MinPrompt:
86  .string "Min Number = "
87
88  MaxPrompt:
89  .string "Max Number = "
90
91  MeanPrompt:
92  .string "Mean Number = "
93
94  Divisible1:
95  .string "There are "
96
97  Divisible2:
98  .string " number(s) divisble by "
99
100 EndText:
101 .string "Program ended"
102
103 /*End of Strings ****************************************************/
```