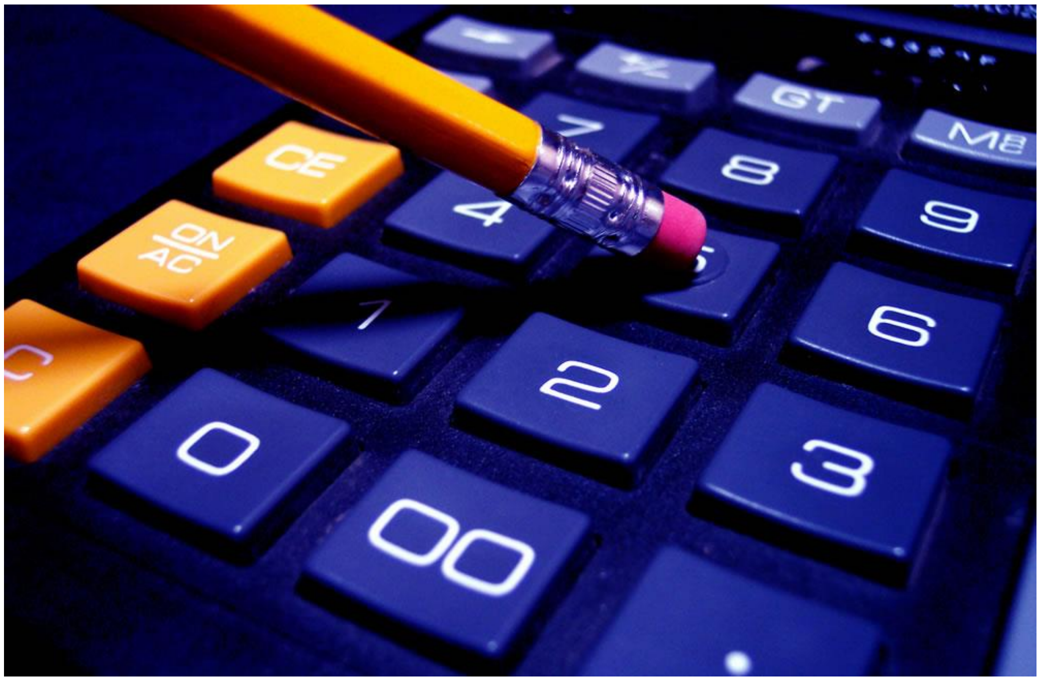
CS1021 - ASSIGNMENT 1



CS1021

‘Simple Statistics Calculator’ by Brandon Dooley

Throughout the following report I will document the various stages of the development of my ARM Assembly Language ‘Simple Statistics Calculator’ program. I will explore various tests that were made throughout the stages of development and difficulties that I encountered respectively.

CS1021 - Assignment 1

'SIMPLE STATISTICS CALCULATOR' BY BRANDON DOOLEY

STAGE ONE | Console Input

"The aim of this stage of the assignment is to design, write and test an ARM Assembly Language program that will read an unsigned value entered by a user in decimal form and store the value in register R4."

This stage of the program fetches a user defined input (decimal number) via the console using the command '*BL getkey*'. After reading and storing the value in the register R0 the program continues on without displaying the key back to the user. This is resolved using the '*BL sendchar*' command which displays the key back to the console.

It reads the key pressed by the user in terms of an ASCII value e.g if the user presses the key '4' the console will read and store the value 0x34 in the register R0. To solve this issue the code subtracts 0x30 from the inputted key leaving the decimal value stored within the register.

To allow for the user to input numbers greater than 10 the program uses a method of a running total and a constant multiplier of 10. The program first converts the number from ASCII to decimal, then it multiplies the initial running total (0) by 10 and adds the converted ASCII key to this running total. This is repeated until the user presses the SPACE/RETURN key whereby the program is then terminated.

Unfortunately, the program does not handle negative user inputs and will give an invalid return value in the case of a negative number being entered. I did not have enough time to compensate for this and as a result the console interprets the – sign as ASCII 0x2D and uses this value as part of the computation, whereas it should check to see if the key is a minus sign and then invert the bits and add 1, giving a two's complement representation of a negative input. The program also only handles whole number integers.

STAGE ONE | Testing

Input	Expected Value	Actual Value
0	0x00000000	0x00000000 ✓
-1	0xFFFFFFFF	0xFFFFFFE3 ✗
397	0x0000018D	0x0000018D ✓
1927293	0x001D687D	0x001D687D ✓
500000023	0x1DCD6517	0x1DCD6517 ✓

STAGE TWO | Sample Sequence and Statistic Computation

"In this Stage of development, you will extend your program to read a sequence of values and compute the following statistical measures: Count, Sum, Min, Max, Mean"

This stage of the program uses the user defined values obtained in stage one to compute the various statistical measures stated above. However, to allow the program to do this it first has to be adjusted to let the user enter a series of numbers rather than just one.

This is done by getting the program to compute the statistical measures every time the SPACE KEY is entered by the user. The statistical measures are stored in individual registers. When the SPACE KEY is pressed by the user the program:

- Increases the value stored in the *Count* register (R5) by 1
- Adds the new value entered by the user to the previous value(s) to compute the *Sum* of the series of numbers and stores the continued result in the *Sum* register (R7)
- Compares the value entered by the user to the previous *Min* value (initialized at 0x7FFFFFFF). If the value is less than the current *Min,* then the value contained in the *Min* register (R8) is replaced with the current user input value.

- Compares the value entered by the user to the previous *Max* value (initialized at 0x00000000). If the value is greater than the current *Max*, then the value contained in the *Max* register (R9) is replaced with the current user input value.

After these are computed the program loops back to Stage One to obtain another value from the user. However, if the user has pressed the RETURN KEY, the program then branches to compute the *Mean* of the series of values entered.

To do this the program divides the *Sum* by the *Count*. This is done by continuously subtracting the *Count* from the *Sum*, each time this occurs the *Mean* value is increased by 1. This is repeated until the *Count* can no longer be subtracted from the *Sum*. The whole number value of the *Mean* is then stored within the register R11.

Similar to Stage One this section also does not handle negative user inputs and only operates correctly when positive integers are entered. It also does not handle significantly large values that cannot be stored in a 32-bit register and result in an overflow. If I had of had more time I would resolve this by checking the Overflow flag and compensating for such an occurance.

STAGE TWO | Testing

Inputs	**Sum (R7)**	**Count (R5)**	**Min (R8)**	**Max (R9)**	**Mean (R11)**
2, 3, 4, 7	0x00000010 (16) ✓	0x00000004 (4) ✓	0x00000002 (2) ✓	0x00000007 (7) ✓	0x00000004 (4) ✓
17293821987 , 3910201, 2010, -12, 81	0x07067967 (117,864,807) ✗	0x00000005 (5) ✓	0x00000051 (81) ✗	0x06CAC823 (113,952,803) ✗	0x0012F1FB (1,241,595) ✓
19273, 129, 6, 2932, 1, 1, 1, 3	0x0000574A (22,346) ✓	0x00000008 (8) ✓	0x00000051 (1) ✓	0x00004B49 (19,273) ✓	0x00000AE9 (2,793) ✓
0, 0, 0, 0, 0, 0, 0, 0, 0	0x00000000 (0) ✓	0x00000009 (9) ✓	0x00000000 (0) ✓	0x00000000 (0) ✓	0x00000000 (0) ✓
Null	0x00000000 (0) ✓	0x00000000 (0) ✓	0x0000000 (0) ✓	0x0000000 (0) ✓	0x0000000 (0) ✓

STAGE THREE | Displaying The Mean

"In this final Stage of development, you will extend your program to display the mean computed in Stage Two. The result must be displayed in decimal form."

This stage of the program uses an algorithm designed to check what power of 10 the leading digit of a value is. It begins by comparing the value (*e.g Mean in R11*) to the number 1 (10^0) and the number 10 (10^1).

If ((mean >= 1 && mean < 10))

If this comparison fails, both powers of 10 are multiplied by a factor of 10 and the comparison is repeated until the value lies within the limits. When this occurs the program subtracts the lower power of 10 continuously until the remainder is less than that number.

Each time a subtraction occurs the value of *outputDigit is increased*. The program then prints the front digit and repeats the algorithm from the start for the remaining digits.

However, this doesn't suffice for digits containing zero's e.g *Mean = 4100,* as there will be a stage when the remainder is 0 after the subtraction whilst the two end digits have not been displayed. To prevent this from occurring the program checks before each subtraction that the remainder will not be 0, besides in the case when the remainder =1.

It then loops to another section in the program which calculates how many 0's to display for a given number using the same methodology as the above algorithm except that every time a 0 is printed to the console the remainder is divided by 10.

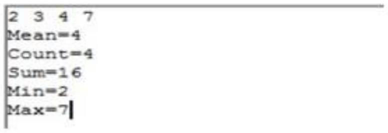
Once the *Mean* is displayed the program continues on to use the above algorithms to print the other statistical measures computed in Stage 2 of the assignment; *Count, Sum, Min, Max*.

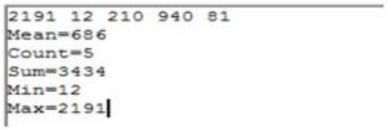
The program also handles invalid user inputs such as the user entering no digits and immediately hitting the RETURN KEY.

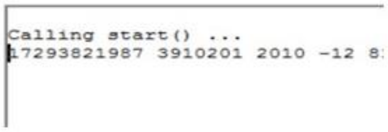
However, similar to the above stages the program does not handle negative values nor does it work with large values which cause an overflow.

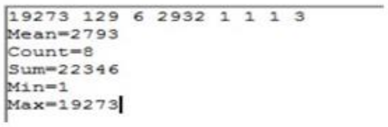
STAGE THREE | Testing

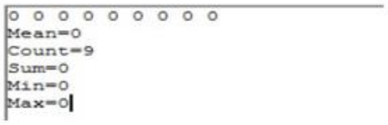
Inputs	**Outputs**	

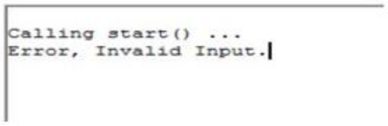
2, 3, 4, 7	 2 3 4 7
	Mean=4
	Count=4
	Sum=16
	Min=2
	Max=7
		✓

2191, 12, 210, 940, 81	 2191 12 210 940 81
	Mean=686
	Count=5
	Sum=3434
	Min=12
	Max=2191
		✓

17293821987, 3910201, 2010, -12, 81	 Calling start() ... 17293821987 3910201 2010 -12 8	✗

1927293, 129, 6, 2932, 1, 1, 1, 3	 19273 129 6 2932 1 1 1 3
	Mean=2793
	Count=8
	Sum=22346
	Min=1
	Max=19273
		✓

0, 0, 0, 0, 0, 0, 0, 0, 0	 0 0 0 0 0 0 0 0 0
	Mean=0
	Count=9
	Sum=0
	Min=0
	Max=0
		✓

Null	 Calling start() ... Error, Invalid Input.	✓

ANALYSIS| Overall Program

I feel that the program on an overall level serves its purpose both efficiently and clearly. Within reason it allows the user to compute various statistical measures at the touch of a button. However, if I had of had more time I would have liked to resolve the issues of negative inputs and overflow errors.

I felt the assignment itself was challenging yet rewarding. I feel I have gained immense knowledge of the assembly language itself and also learned how to solve large problems by using the simplest of utilities e.g division by subtraction. I have also realized the necessity of thoroughly and rigorously testing programs to ensure all possible inputs and scenarios are handled.

Brandon Dooley (#16327446)

CS1021 Assignment 1 – Simple Statistics Calculator

Submitted 21/11/2016