Assignment #1 – Simple Statistics Calculator

Section 1 – Console Input

At the beginning, I chose a register to store the result in and initialised it by setting it to zero. In addition to R13-R15, I did not use R0-R3 as they were affected by the use of the console.

```
result = 0;
```

I then got the first digit from the input console. I then compared this value, which is stored in R0, to the ASCII value for CR, which is 0x0D. If the key entered was not CR, then the while loop would begin.

```
Char = System.in();
While (char != CR)
{
```

In the loop, the character would first be echoed back to the console as output as this is not done automatically in ARM assembly language. I then needed to convert the input to a decimal value as the value stored in R0, is the ASCII value for the number symbol instead of the number. The difference between the two is 0x30 so I needed to subtract this from the value.

```
System.out.print(char);
char = char - 0x30
```

Then I multiplied the result by ten, storing it in the same register, in order to make sure that if multi-digit numbers were entered, each value would be multiplied by an increasing power if ten. The new value was then added to the result. This cannot be done before the multiplication as the final result would be ten times more than the entered value.

```
result *= 10;
result += char;
```

The program then gets another input from the console and begins the loop once more, again checking to see if the value is equivalent to the ASCII value for CR.

```
char = System.in();
}
```

Input	Reason	Output	Stored Value	Conclusion
1	It is a basic, low value.	1	0x00000001	The program works for basic, one-digit values.
0		0	0x00000000	It displays and stores the correct value.
null	I wanted to see what happened when no value is entered.	N/A	0x00000000	It does not display any value and there is no result which is correct.
99999	It is a multi-digit value.	99999	0x0001869F	The program works for high multi-digit values, displaying and storing the correct result.
4294967295	It is the highest value possible.	4294967295	0xFFFFFFFF	Since this is the highest possible value, the program works for all positive values and zero.
-1	It is a negative number.	-1	0xFFFFFFE3	The program will display the correct number but will not store the correct value.
-99	It is a larger negative number (it has two digits).	-99	0xFFFFFF37	For multi-digit negative values, it again displays the right number but does not store the correct result.

Section 2 – Statistics Evaluation

Firstly, I assigned a register to each of the statistics that I was going to be evaluated. I then initialized the registers for each of the statistics that had to be updated after every number in the sequence. The sum and the count were both set to zero. The maximum value was set to the lowest possible value, 0, and the minimum value was set to the highest possible value, 0xFFFFFFFF.

```
Count = 0;
Sum = 0;
maxValue = 0;
minValue = 0xFFFFFFFF;
```

Like the first section of the assignment, a number is gotten with a loop but instead of the end of the number being signalled by CR, it is signalled by a space. The end of the sequence is signalled by a full stop. After every space or full stop, the count, sum, maximum value and minimum value are updated. At first, the full stop in my program would stop the entire loop and therefore not update

the statistic values for the final number entered. To remedy this, I instead branched to the start of these value updates. As a result, I needed to check at the end if it was or space or a full stop and branch back up to the top if it was a space.

```
do
{
        result = 0;
        char = System.in();
        System.out.print();
        if (char != '.')
        {
                number = char;
                number -= 0x30;
                result *= 10;
                result += number;
                char = System.in;
        }
        count++;
        sum += result;
        if (result > maxValue)
                maxValue = result;
        if (result < minValue)
                minValue = result;
} while (char != '.');
```

After this loop is concluded, you need to calculate the mean and I also calculated the range. I firstly initialised the mean by setting it to zero. The mean is the sum divided by the count. Since there is no divide operation in the ARM assembly language, we need to use another method. We firstly copy the sum into a temporary register so that we do not alter the value for the sum. You need to see how many times the count can go into the temporary sum in a while loop adding one to the mean every time. This will not calculate decimal places but the nearest whole number below. The range is gotten simply by subtracting the minimum value from the maximum value.

```
mean = 0;
tempSum = sum;
while (tempSum >= count)
{
        mean++;
        tempSum -= count;
}
range = maxValue – minValue;
```

Input	Reason	Count	Sum	Max Value	Min Value	Mean	Conclusion
1 2 3.	These are basic, low values.	0x000 00003	0x000 00006	0x000 00003	0x000 00001	0x000 00002	The program works for a simple series of low, single-digit values.
.	I wanted to see what happened when no value is entered.	0x000 00001	0x000 00000	0x000 00000	0x000 00000	0x000 00000	When no value is entered, All the values are correct except for the count because the results are updated every time space or full stop is pressed even if there were no numbers entered.
(space) .	In order to see what happened when only a space was entered.	0x000 00002	0x000 00000	0x000 00000	0x000 00000	0x000 00000	Again, the count is wrong for the same reason as before.
99999 67543 9876 4532 876.	These are high multi-digit values.	0x000 00005	0x000 2CA2A	0x000 1869F	0x000 0036C	0x000 0ED5	The program stores the correct results for a sequence of very high multi-digit values.
4294967290 99.	I tested these values to see what happens when the results are greater than the maximum value.	0x000 00002	0x000 0005D	0xFFFF FFFA	0x000 00063	0x000 0002E	For these values, the sum is incorrect as it is greater than the highest value possible, so therefore it wraps around and starts from zero again. As a result of the sum being incorrect, the mean is also incorrect.
-1 1 32 -99	Negative numbers are included in this sequence of numbers.	0x000 00004	0xFFFF FF3B	0xFFFF FFE3	0x000 00001	0x009 5FC52	The count is correct for negative values as it only considers the amount of values rather than the value itself. Since the values for negative numbers is wrong, all of the other numbers are incorrect.

Section 3 – Display Results

I wanted to display all six results to the console. In order to do this, I created a for loop, initialising the count as one and incrementing it until it reached six as there were six results to be displayed. Within this loop I had what was effectively a switch statement with six different cases. For each one, I would output the name of the statistic and copy the result of the statistic into another register so as not to destroy the original value.

```
for (count = 1; count <= 6; count++)
{
switch (count)
{
case 1:
        resultToBeDispalyed = count;
        System.out.print(" Count: ");
        break;
case 2:
        resultToBeDisplayed = sum;
        System.out.print(" Sum: ");
        break;
.
.
.
```

Once the result to be displayed was in the correct register, I needed to output that particular value. You can only output one number at a time so you need to split up multi-digit values. You could keep dividing the result by decreasing powers of ten starting with the highest possible in this assembly language but this would most likely leave a lot of unnecessary leading zeros. Instead I kept dividing the result to be displayed by ten until it was not possible to do so. Every time I did this I added one onto a register assigned to the power which was initialised to zero.

```
highestPower = 0;
divisionResult = 0;
while ( resultToBeDisplayed >= 10)
{
        while (resultToBeDisplayed >= 10)
        {
                resultToBeDisplayed -= 10;
                divisionResult++;
        }
        highestPower++;
        resultToBeDisplayed = divisionResult;
}
```

Now that I had the highest power of ten for the particular result to be displayed I needed to find this power of ten. I did this with a loop which initialised the result as one and multiplied the result by for as many times as the power was. I then needed to copy the result to be displayed again into another register again as it had been previously altered in the program.

```
highestPowerOfTen = 1;
mulCount = 0;
while (mulCount < highestPower)
{
        highestPowerOfTen *= 10;
        mulCount++;
}
```

Once you had the result to be displayed and the highest power of ten, you could then output the value. You needed to divide the result by the highest power of ten using the loop previously shown. The remainder from this division is then set as the new result to be displayed. The highest power of ten is also divided by ten using another division loop. The result to be displayed is divided by the next power of ten again and again until the power of ten is one. Once this is done the program loops back up to display the next result until all of the results are displayed on the console.

```
switch (count)
{
case 1:
        resultToBeDisplayed = count
        break;
case 2:
.
.
.
displayResult = 0;
while (highestPowerOfTen > 1)
{
        while (resultToBeDispalyed > highestPowerOfTen)
        {
                resultToBeDisplayed -= highestPowerOfTen;
                dispalyResult++;
        }
        char = dispalyResult;
        char = (hexadecimal) char;
        System.out.print(char);
        tempHighestPowerOfTen = highestPowerOfTen;
        highestPowerOfTen = 0;
        while (tempHighestPowerOfTen >= 10)
        {
                tempHighestPowerOfTen -= 10;
                highestPowerOfTen++;
        }
        displayResult = 0;
}
```

Input	Reason	Count	Sum	Max Value	Min Value	Mean	Range	Conclusion
1 2 3 4 5.	These are basic, low values.	5	15	5	1	3	4	The program displays the correct results for low values without the leading zeros.
.	I wanted to see what happened when no value is entered.	1	0	0	0	0	0	When no value was entered, the count displayed was incorrect because the value stored was wrong from the previous part of the program.
1 2 3 4 5 6 7 8.	In order to test when the mean would not be a whole number.	8	36	8	1	4	7	The mean displayed was incorrect because the program does not deal with non-integer values and therefor rounds down to the next integer.
99999 67543 9876 4532 876.	These are high multi-digit values.	5	182826	99999	876	36565	99123	The program still works for higher, multi-digit values but the mean again rounds down to the nearest integer.
4294967290 99.	I tested these values to see what happens when the results are greater than the maximum value.	2	93	N/A	N/A	N/A	N/A	The program displays the count correctly but not the sum because it is too large. As a result, the program cannot calculate the maximum value and breaks down.
101.	In order to test when only one	1	101	101	101	101	0	Even for single value inputs, the program still works correctly.

	value was entered.							