Name: Ciaran Coady

Student Number: 17326951

1 Console Input

After assessing the question for this stage, I made a logical procedure for how my code should function:

1. Read the user input e.g. 1, 2 etc
2. Store the decimal value in a register
3. Display what has been inputted to the console

Upon starting my code I soon realised it was slightly more complex, (1) What happens after the first digit is stored? (2) When do I stop reading values? These were questions that I began to ask.

This is the sudo code I then used to help me write the assembly code:

```
Key entered = console input (bl getkey)
Bl sendchar; <- displays number inputted
While (key entered != carriage return)
{
    Key entered = console input (bl getkey)
    Bl sendchar; <- displays number inputted

    If (currentNumber > 0)
    {
        currentNumber = key entered – 0x30; <- ascii to decimal conversion
    }
    Else
    {
        currentNumber = currentNumber *10;
        key entered = key entered – 0x30;
        currentNumber = currentNumber + key entered;
    }
```

Answering my questions:

1. After storing the first number, multiply the existing number by ten and add the new one onto it.
2. Stop reading values (break the loop) when enter is pressed 0x30

Using this sudo code as a guide I developed my arm assemble language code. I then proceeded to test the code using the following inputs:

Key pressed by user	Expected value stored	Actual value stored	Expected output to console	Actual output to console
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5
6	6	6	6	6
7	7	7	7	7
8	8	8	8	8
9	9	9	9	9
Carriage return	0x0D	0x0D	None (as program should stop reading values)	None (as program should stop reading values)

The code functioned as expected and I was very happy with the result. Unfortunately I didn't get time to add additional functionality like taking the input of negative numbers. I used unsigned bits for my program. But I intended if the minus sign was inputted that the next number entered would be stored as normal but then the 2s complement of the number would be calculated and the new value stored. If I had done this it would have changed my code slightly in stage 2 as:

1. I would be using signed operations and comparisons rather than unsigned.
2. I could have used my addition code for both addition operations and subtraction operations. If subtracting a number just get its 2s complement and add it on to the existing total.

2 Expression Evaluation

I had thought of various different ways of approaching this problem but I decided to take the following approach:

- Use the previous code from stage 1, to get the users input and echo it back to the console.
- When any operation is entered e.g. +, - or * store that operation and take the input of the next number.
- When the next number gets entered check is there a pending operation. If so do it.
- When enter gets pressed end of code, carry out any pending operations and store the result.

By using this approach the code will work with as many operations as desired. It will also allow me to work with negative numbers as long as the end result isn't negative.

The following is the sudo code I used to help me approach this problem:

```
While (key pressed != carriage return)
{
    If (key pressed = +)
    {
        Total = total + number being worked on
        Number being worked on = 0
    }
    Else if (key pressed = -)
    {
        Total = total - number being worked on
        Number being worked on = 0
    }
    Else if (key pressed = *)
    {
        Total = total * number being worked on
        Number being worked on = 0
    }
}
```

Using this sudo code I was able to develop my arm assembly code and then began to test some values (for the purpose of display I have converted and hex values to decimal)

Operation	Result	Reason for testing?
1+1	2	Checking if addition works
1-1	0	Checking if subtraction works
1*1	1	Checking if multiplication works
0*1	1	How are zeros handled
1+0+1	2	Multiple operations
1*5-2	3	Multiple different operations

After doing some testing I discovered that I had made an error in my code, I tried to have one function to add in numbers, but I had overlooked how the first number was treated differently to others, therefore it need a separate function. Luckily this didn’t take long to correct.

I felt this stage went well as my code could carry out as many operations as the user desired, but still has the limitation that it can display a result if it is negative (it becomes an extremely positive number as the number system wraps around)

3 Displaying the result

As suggested I went about using previous code we developed to break up the number into single digits. This is required as we can only output a single character at a time to the console.

My logic and approach to this stage was as follows:

- Find the largest multiple of 10 that will divide into the number.
- Subtract that multiple of 10 from the number, keeping track of how many times it does so.
- When the number becomes smaller than the divisor:

1. Output the quotient of division.
2. Reset the divisor and start again.

The following is the sudo code I used to help me write my arm assembly language code:

```
While (number >= testdivisor)
{
    Divisor = testdivisor
    testdivisor = testdivisor * 10
}
```

This code will find out what power of 10 we need to use to break up the number and stores it in the variable divisor.

```
While (number >= divisor)
{
    Number = number – divisor
    Quotient = quotient +1
}
Else while (next divisor > number)
{
    Send a zero to the console
    Go to code to get next divisor
}
```

This gets the next digit of the number that we want to display, if the number is smaller than the next divisor, then we need to display a zero in the console. After this we need to calculate the next divisor and check do we need to display another zero.

```
While (temp divisor != 0)

{

Temp divisor = temp divisor -10

New divisor = new divisor +1

}

Go back up to code to check for zeros
```

This code will calculate what the next divisor is and allow us to check if more zeros should be displayed.

Using a register to hold a test value I tried various numbers to see how my code handled them:

Test Number	Result	Reason for testing?
10	10	See if zeros display properly at the end of a number
2001	2000	See how zeros in the middle of numbers are displayed
4123	4123	Test a random number with no zeros
0003	3	Can it handle leading zeros

After doing this preliminary testing I discovered that the my code would not properly display zeros in some numbers, if the zero occurred in the middle of the number it would display, but the rest of the number would now become zeros also. But my code works for any numbers that don't contain zeros, and gets rid of any leading zeros.

I then added a few more conditions, this got rid of the problem with the zeros and also made sure the correct number was displayed in all cases.

Now happy that my code worked well I done some more testing:

Test Number	Result	Reason for testing?
10	10	See if zeros display properly at the end of a number
01	01	Can it handle leading zeros
0	0	Will output zero on its own
2001	2001	See how zeros in the middle of numbers are displayed
20401	20401	Number with multiple zeros in the middle
4123	4123	Test a random number with no zeros
-30	na	Negative numbers?

From my testing I was happy to conclude that my code worked in most cases, with the exception of negative numbers which returned no result.

I then added the code to previous code from stages 1 and 2 and began final testing.

Operation	Result
1+1-1	1
0*6*7	0
20*20	400
-5+10	5
4798-12	4786
0-1	No result

Overall I was happy with the functionality of my program as a whole. Each stage moved nicely into the next one with much ease. I regret starting with unsigned values as it made it very difficult for me to work with negative numbers and prevented me from displaying negative results.

Another problem with the code which I can't quite figure out is that after pressing enter, you must press enter again for the code to display the result. My suspicion is that due to my methodology of having an operation pending, plus the fact that the program will hang at bl getkey until something is pressed is the cause of this issue.

Other than that I feel the code works well and does what is required, while also adding the extra functionality of multiple operations and also formatting the result correctly.