DAVY NOLAN
CS1021 ASSIGNMENT #1

Simple Calculator

Date: 9th of November 2017

Davy Nolan
Assignment #1

Stage 1: Console Input

Aim:

"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."

Solution:

The pseudocode below explains how to display a number that has been entered in the console following an enter key.

Pseudocode:

```
Read key; (read key and store in result)
Input: 2, 3, 1, Carriage Return (enter key)
Result = key – value                          [2]
Result = Result x 10                          [20]
Result = Result + key - value                 [23]
Result = Result x 10                          [230]
Result = Result + key - value                 [231]
Printkey; (display result in console)
```

This pseudocode then developed into a more suitable pseudocode.

Pseudocode:

```
Result = 0;
Readkey;
While( key != enter key)
{
        Printkey;
        Result = Result x 10
        Value = key - #0x30     ;convert from Ascii to numerics
        Result = Result + value
        Readkey;
}
```

This was then used to develop the program even further but in ARM assembly language.

```
 8  start
 9      LDR R6, =10
10      LDR R4, =0
11  read
12      BL  getkey      ; read key from console
13      CMP R0, #0x0D   ; while (key != CR)
14      BEQ endRead     ; {
15      BL  sendchar    ;   echo key back to console
16
17
18      MUL R4, R6, R4      ;result = result x 10
19      SUB R5, R0, #0x30   ;convert ascii to numeric
20      ADD R4, R4, R5      ;result = result + value
21      B   read        ; }
22
23  endRead
24
25  stop    B   stop
26
27      END
```

This program shows a while-loop. The program continues to carry out the expressions from lines 15-20 until the “Carriage Return” key is entered, then it branches to “endRead” and the program stops. The user must enter all digits followed by the enter key and then the entered number will be stored in register R4.

Methodology:

The following inputs were tested with this program and they gave the following values in R4.

Input key	R4 value
1234	0x000004D2
438	0X000001B6
27	0x0000001B
5	0x00000005
0	0x00000000
-1	0xFFFFFFE3

I tested the program using these values of input as they test if the program runs well with positive large numbers, positive small numbers, zero, and negative numbers. All positive inputs and zero gave correct answers in the register R4. However, negative inputs gave incorrect answers in the register R4. This is because the program is only fit to read positive numbers.

Stage 2: Expression Evaluation

Aim:

- "In this stage of the assignment, you will extend your program from Stage 1 to read and evaluate expressions such as "100+250", "90*3" and "25-5". Your program should store the result of the operation in register R5."
- "Your program should support the addition (+), subtraction (-) and multiplication (*) operators."

Solution:

To make this program work, the first program had to be extended upon. In stage 1, the program read a key input from the console and stopped once the enter key was input. Therefore, the program was able to detect the enter key with the use of the Ascii table. In stage 2, the program had to be able to read and detect the '+' symbol for addition, the '-' symbol for subtraction and the '*' symbol for multiplication. This was done by comparing R0 to their corresponding Ascii values and branching the program if they are entered.

```
15  read
16      BL  getkey          ;read key from console
17      CMP R0, #0x0D       ;while (key != CR)
18      BEQ endRead         ;{
19      CMP R0, #0x2B       ;compare read key to '+'
20      BEQ addition        ;branch to addition if '+' is entered
21      CMP R0, #0x2D       ;compare read key to '-'
22      BEQ subtraction     ;branch to subtraction if '-' is entered
23      CMP R0, #0x2A       ;compare read key to '*'
24      BEQ multiplication ;branch to multiplication if '*' is entered
25      BL  sendchar        ;echo key back to console
```

However, I needed the program to be able to tell if the first and second operands had been entered. I did this by loading R10 with 0 to act as the operator codes.

```
32      CMP R10, #0   ; while ( R10 < 0)
33      BHI operand2  ; {
34      MOV R9, R4    ; make R4 the first operand
35      B   read      ; }
36
37  operand2
38      MOV R8, R4   ; make R4 the second operand
39      MOV R3, #1   ; give R3 the value of 1
40      B read
```

Lines 19-24 determine when and where the code branches. This is split into 3 branches; “addition”, “subtraction” and “multiplication”.

Addition branch

```
52  addition
53      BL sendchar     ;echo key back to console
54      LDR R4, =0      ;update R4 to 0
55      LDR R10, =1     ;update R10 to 1
56      CMP R3, #0      ;while (R3 != 0)
57      BEQ read        ;{
58      ADD R5, R9, R8 ;add 1st operand to 2nd operand and store result in R5
59      B read          ;}
```

Subtraction branch

```
61  subtraction
62      BL sendchar     ;echo key back to console
63      LDR R4, =0      ;update R4 to 0
64      LDR R10, =2     ;update R10 to 2
65      CMP R3, #0      ;while (R3 != 0)
66      BEQ read        ;{
67      SUB R5, R9, R8 ;subtract 1st operand from 2nd operand and store result in R5
68      B read          ;}
```

Multiplication branch

```
70  multiplication
71      BL sendchar     ;echo key back to console
72      LDR R4, =0      ;update R4 to 0
73      LDR R10, =3     ;update R10 to 3
74      CMP R3, #0      ;while (R3 != 0)
75      BEQ read        ;{
76      MUL R5, R9, R8 ;multiply 1st operand by 2nd operand and store result in R5
77      B read          ;}
```

Methodology:

The following inputs were tested, and the following answers were stored in R4.

Addition

Input	R4 Value
456+297	0x000002F1
2+3	0x00000005
6+0	0x00000006
-2+5	0x00000005

All inputs give the correct answer except “-2+5”. This input contains a negative value which the program cannot deal with. The program clearly succeeds with high positive values, low positive values and zero.

Subtraction

Input	R4 Value
8967-4367	0x000011F8
14-8	0x00000006
3-0	0x00000003
-6-2	0xFFFFFFFE

All inputs give the correct answer except “-6-2”. This input contains a negative value which the program cannot deal with. The program clearly succeeds with high positive values, low positive values and zero.

Multiplication

Input	R4 Value
342*45	0x00003C1E
12*3	0x00000024
3*0	0x00000000
-2*2	0x00000000

All inputs give the correct answer except “-2*2”. This input contains a negative value which the program cannot deal with. The program clearly succeeds with high positive values, low positive values and zero.

Stage 3: Displaying the Result

Aim:

"In this final stage of the assignment, you will extend your program from Stage 2 to display the result contained in R5 in the console window. The result must be displayed in decimal form."

Solution:

In order to display the values in the register, they must be converted to the Ascii characters representing the value and then displayed in the console window. This was done by dividing the value in R5 by 10, using the whole part of the result as the next Ascii digit and the remainder as the next value to be divided by the lower power of 10.

For example, given the input value 123:

Dividing 123 by 100 gives a quotient of 1 and a remainder of 23. Therefore 0x01 + 0x30 = 0x31 (character '1') will be the first Ascii character to be displayed.

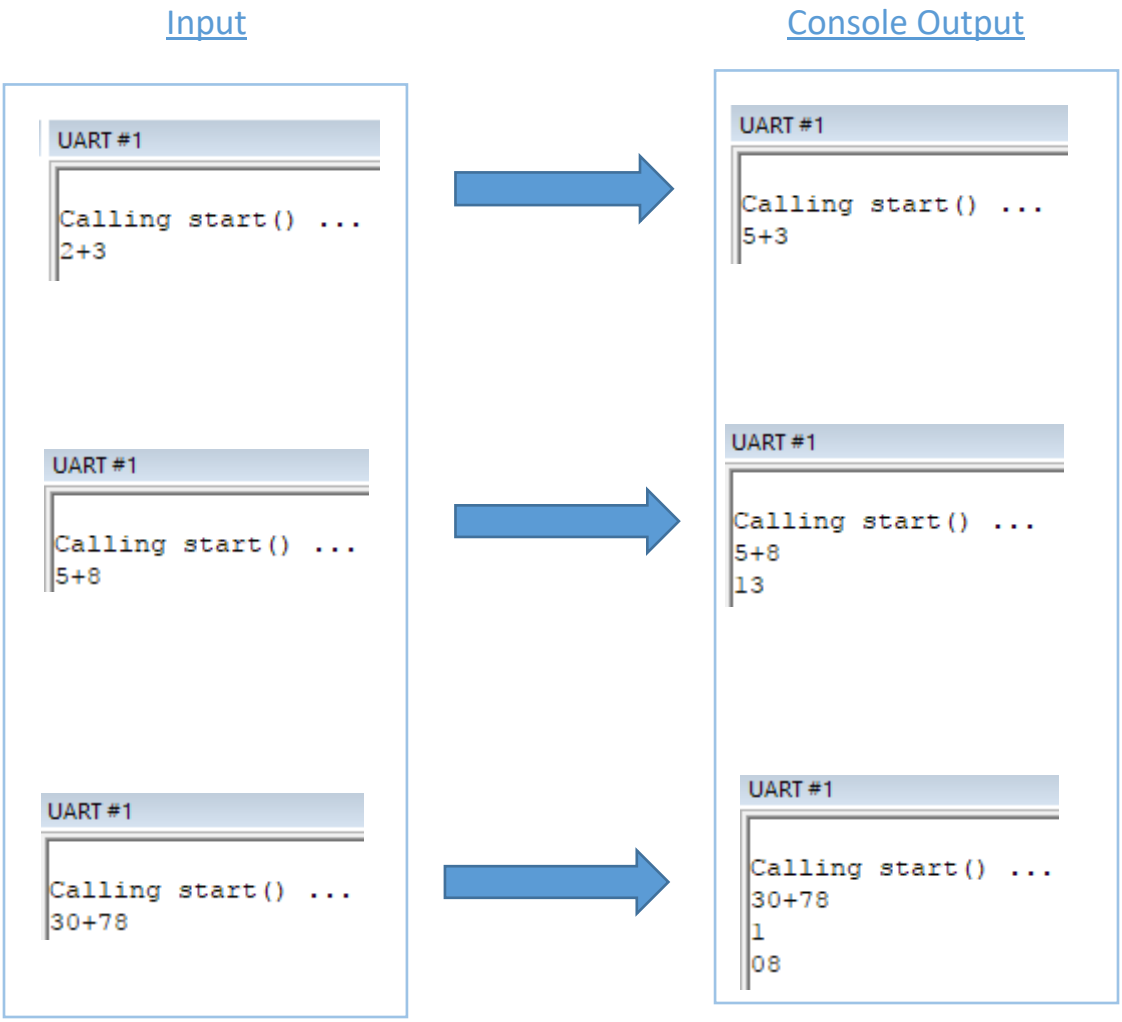
This is done in the part of the program below.

```
127  printDigit
128      ADD R0, #0x30   ; R0 =quotient + character '0'
129      BL sendchar     ; print
130      MOV R5, R12
131      B nextDigit
132
133  lastDigit
134      MOV R0, R5
135      ADD R0, #0x30
136      BL sendchar
137      B endProgram
138
139  endProgram
```

Methodology:

Different inputs were tested with addition, subtraction and multiplication.

Addition



The program works to an extent. For 2- digit answers, the program displays the answer perfectly on the next line. However, for 1 -digit answers, the program displays the answer but also displays the second operand with it. For 3- digit answers, the program displays the answer, but the first digit is on the second line and the rest of the digits are on the third line.

The exact same situation happens for *subtraction* and *multiplication*