Assignment #1

Simple Calculator

Name - Krishanu Dey

Student Number - 17306518

Module - CS1021

Lecturer - Rebekah Clarke

- In this assignment, I developed a Simple Arithmetic calculator
- The calculator is able to take positive, zero and negative numbers i.e. signed values.
- The calculator can perform addition, subtraction, multiplication and division.
- The program can take more than two operands(as many as the user wants in fact) at once.
- The numbers are correctly displayed without the leading zeros.
- The negative numbers have been showed with a minus sign before them.
- Once the program has done the calculations and has displayed the result, the program goes back to the start(i.e. restarts) so that the user input a new expression.
- The program does perform division, but not for negative numbers.
- The program has been divided into 3 stages-

Stage 1 - Console Input

In this stage, the values entered by the user are stored in R4. The user can input either numbers(positive, zero or negative numbers) or operators(+, -,*, /).

Stage 2 - Expression Evaluation

The expression is evaluated in this part of the program. The program could solve expression with any number of operands.

Stage 3 - Displaying the Result

In this stage, the evaluated result is displayed in decimal form. The program doesn't display leading zeroes. The negative numbers are displayed with a negative sign before them,

A detailed description of how each of the 3 Stages work has been provided in the following pages.

Stage 1

20	Moving the previous input to next input <- this is identify if 2 signs were inputted one after the other, which could mean the next number is negative. Eg- 12*-, 1+5*-8
22	Read key from console
23	If the user presses the Enter key, these statements would end the input part, and move on to the next section.
27-37	If user inputs +, - or *, the program control will go to the sign() function
39-42	If the user's input is not either an Enter key, a sign(either +, - or *) or a number, it could just be an invalid input so the program will stop.
44-45	If the program control is in this part, the input is surely a number. Using R6 to convert the input in R0 to integer from its ASCII equivalent.
47	Increasing the placeValue(i.e. R7) by 1 each time a new integer is inputted for the same number.
48-52	With this the new integer is put in the correct place of the number.
	Going back to read to take the next input.

Stage 2

56-80 **Sign Function**

R11 contains the previous evaluated result. Checking if there is any previous number input before the new number input. If yes, the program control goes to the eval() function to evaluate them and store them in R4.

If the previous input was a sign as well, it may mean the user is going to input a negative number.

If If the previous input was a sign and the new input is a minus sign, then the next number the user is going to input is a negative number. So, the value in R12 is changed to 1, which I used to mark that the next number is going to be a negative number. Otherwise, if it is *+, +*or … this would be an invalid input and the program would stop.

If the input wasn't a second sign(as in two signs one after the other), we check if there is any value already evaluated in R11, if yes we go to eval to evaluate the result till that point. After the R11 check(and possibly eval), the control of the program goes to evalBack

82-97 **evalBack Function**

If the new input was a plus sign('+'), the control goes to the plus() function. If the new input was a minus sign('-'), the control goes to the minus() function. If the new input was a star sign('*'), the control goes to the star() function. If the new input was a slash('/') sign, the control goes to the divide() function.
If the new input was an Enter key, the control goes to the Stage3 function to display the evaluated result.

99-125 **plus(), minus(), star() and divide() Function**

Each each of these save the respective sign ASCII value in R10, moves the previous result to R11,and refreshes the number input(R4) and the place value(R7) to 0. The control goes back to take a new input.

129-168 **eval Function**

If the boolean R12(which is used to mark negative numbers) is true, i.e. 1 then these statements will convert the new input to negative and make the boolean R12 equal to 0 so that the next number is not automatically considered negative.

Makes the necessary evaluations , that is either adds, subtracts or multiplies the number, and stores the result value in R4 and R5. After that the control goes to the function evalback.

Stage 3

177 First displaying the equal sign('='), so that the operands are separated from the result.

177-187 Checking if the number is negative or not by checking if the number is smaller than 0 or not. If it is negative, printing the minus sign ('-') to show that the result is negative. Then I converted the number to positive so that the program could display the number.

189-198 With this code, we get the appropriate power of 10 to start with. The required value is stored in R8. It works by comparing the respective power of 10 (R12) with the number. When temp i.e. R12 is greater than 1, it means the appropriate power of 10 that we need is stored in R8, which contains a power of 10 one less than that in R12, i.e R8=R12/10

204-218 **Number function**
Here, I divided the evaluated number by R8(which stores the closest smallest power of 10). The quotient is the Most Significant Digit, which we display on the console by using the function display(). The remainder is the rest of the number, which is also needed to be printed. So, the remainder is transferred to R5. Thus, making the remainder as quotient for the next division to remove the MSD.

220-235 These statements reduce the power of 10 by one, so that the value in R8 is the **closest smallest** power of 10.

237 Control going back to number function to retrieve the new MSD.

221-222 The displaying/loop ends when R8 is equal to as then the Least Significant Digit has already been displayed.

241-243 Printing tab space before the program restarts.

Account of Trial Input

Test :

54+16*-2

To test a normal expression with multiple operands with one input being negative; also the result is a negative number

Part 1

Input '5'- Input is stored in R0; Converted to int equivalent from ASCII by subtracting 48(hex value = 0x30); placeValue increased by 1 and stored in r4

Input '4'- same procedure as above but this time, control goes into the loop multiplies number stored in R4(here 5) by 10 and adds the new input. Basically, (5*10)+4. Hence value in R4 now is 54.

```
if(placeValue>1)
        numberInput = numberInput * 10
numberInput = numberInput + input
```

Part 2

Input '+' - As R0 is equal to 0x2B, control goes to sign(); then the control goes to star(); 0x2B is stored in R10; the 54 in R4 is moved to R11, and R4 and R7 are resetted to 0.

Part 3

Input '16' - the number 16 is inputted in the same manner as 54

Part 4

Input '*' - As R0 is equal to 0x2A, control goes to sign(); As there is a previous value in R11, the control goes to eval to evaluate the expression before this '*' input and the evaluated result is stored in R5 and for calculations in R4 as well. Eval evaluates and the control goes to evalback. 0x2A is stored in R10; the 54 in R4 is moved to R11, and R4 and R7 are resetted to 0.

Part 5

Input ‘-’ - As R0 is equal to 0x2D, control goes to sign(); as there are 2 signs inputted together, control will go to secondsign(); the program checks if it is a minus sign and then makes the boolean R12 true to mark that the next number is going to be negative.

Part 6

Input ‘2’- the number 2 is inputted in the same manner as 54 and 16;

Input ‘Enter Key’ - Control goes to label endRead; then to eval(), the programs checks if R12 is true or not; as it is true, the new number is made negative; the expression is then evaluated and stored in R5

Part 7

The control now goes to label Stage3 to display the result. The working of the stage 3 is present on page 4.

Output

54+16*-2=-140