Cian Higgins
Rebekah Clarke
CS1021
Introduction to Computing I
13 November 2017

Assignment 1 - Simple Calculator

1 Console Input

Given Task

'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. The value will be entered by the user as a string of ASCII characters. For example, if a user enters "2", followed by "1", followed by "3", your program should store 21310 (or 110101012 or D5 16) in R4.'

Pseudocode

```
int result = 0;

readkey();
while (key != CR)
{
    printkey();
    result = result * 10;
    value = key - #0x30;
    result = result + value;
    readkey();
}
```

Solution

The chosen solution utilises a while loop using flow control methods available in ARM Assembly Language. The functions BL getkey and BL sendchar are also used to retrieve user input from the console and display the input to the user. The BL getkey function gets the user input in the form of an ASCII character value from the keyboard. This means the ASCII value must be converted to its actual numeric value for it to be of any use in the program. This is done by simply subtracting 0x30 or 48_{10} from the ASCII value. This is because the keys 0 - 9 on the keyboard are represented as 0x30 - 0x39 in ASCII. After this conversion has been made the program adds the proper numeric value to the result variable. On a second iteration

the result variable is first multiplied by 10_{10} before the ASCII input is converted and added to the result.

Example

For instance, if the user wanted to input the number 125_{10} the program would behave in the following way:

- The first key on the keyboard pressed is '1'. The BL get key function receives this input as 0x31, the keys ASCII character code. The program checks that the key is not the carriage return key which is represented as 0x0D in ASCII. Since this is not the case, the while loop is not broken. The key is echoed to the console with BL sendchar. Then the result variable, which is currently equal to zero, is multiplied by 10, leaving the result unchanged. The ASCII character code 0x31 is converted to the numeric value 1 and added to the result. The unconditional branch brings the loop back to the start.
- The second key input by the user is '2' or 0x32. Again the program checks for the carriage return key and continues the loop when it is not found. The '2' key is echoed to the console. The result (currently 1) is multiplied by 10, making the result 10. The ASCII code for '2' is converted to a numeric 2 and added to the result, making the result 12. The loop continues.
- The third key pressed is '5' or 0x35, and it is echoed to the console window. The result is multiplied by 10 giving 120. Then the numeric value of 5 is added to 120 giving the desired result of 125.
- Finally, the carriage return key (0x0D) is pressed, breaking the while-loop. The saved result of 125 (or 0x7D in hexadecimal) stays unchanged in its register.

2 Expression Evaluation

Given Task

*'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. Each expression will contain two operands (values) and one operator.'*

Along with the following assumptions:

- Users will only input sums in a valid format
- Operands and/or the result will not exceed the maximum that will fit into a 32-bit register
- Expressions will only produce results which are zero or positive.

Pseudocode

```
int result = 0;
int operand1 = 0;
int operator;
//The operator is stored as one of the following values:
//1 = addition, 2 = subtraction and 3 = multiplication.

readkey();
while (key != CR)
{
    printkey();

    if (key == ‘*’) {
        operator = 1;
    }
    else if (key == ‘+’) {
        operator = 2;
    }
    else if (key == ‘-’) {
        operator = 3;
    }
    else {
        result = result * 10;
        value = key - #0x30;
        result = result + value;
        readkey();
    }
}
int total;

//Calculations
if(opStorage == 1) {
    total = op1 + op2;
}
else if (opStorage == 2) {
    total = op1 - op2;
}
else if (opStorage == 3) {
    total = op1 * op2;
}
```

Solution

The program checks if the key input is equal to ‘+’, ‘-’ or ‘*’ (0x2B, 0x2D or 0x2A, respectively). If one of the if statements return true, the operator variable will be set

to either 1, 2 or 3 which corresponds to addition, subtraction or multiplication, respectively. When one of these characters is found the program moves the result into a new register and resets the result register. The operator is also stored in a register and will be used later in the calculations. It then repeats the console input part to get the next operand of the expression, storing the result in the result register. The while loop is then terminated by the carriage return key and the program moves on to the calculations (*N.B.:* the ExpEval.s file works with two operands, whereas the final version of the DisplayResult.s file was edited to include some of the 'extra mile' tasks. This is discussed further under the Extra Mile heading). The calculations are executed as follows:

- The program checks if the operator storage (opStorage) is equal to 1. If true, the code branches to the addition section and adds the two operands, storing the result in total (R5).
- An else if statement is used to then check if the operator is equal to 2. If this is true, the code branches to the subtraction section and performs the subtraction, saving the result in the total register (R5).
- Another else if statement is used to check the operator storage for 3. If this is true, the code branches to the multiplication section and performs the multiplication, storing the result in R5.

Testing

To test this a selection of numbers were input to the console R5, R6, and R7 for the numbers ensuring that the registers had stored correctly what was input to the console. Below is a summary of the test:

R6 (OPERAND 1)	R10 (OPERATOR STORAGE)	R7 (OPERAND 2)	R5 (TOTAL)
1_{10} / 0x1	+	1_{10} / 0x1	2_{10} / 0x2
75_{10} / 0x4B	*	32_{10} / 0x20	2400_{10} / 0x960
100_{10} / 0x64	-	50_{10} / 0x32	50_{10} / 0x32
6793_{10} / 0x1A89	-	4567_{10} / 0x11D7	2226_{10} / 0x8B2
753_{10} / 0x2F1	*	23_{10} / 0x17	17319_{10} / 0x43A7
893_{10} / 0x37D	+	217_{10} / 0xD9	1110_{10} / 0x456
9999_{10} / 0x270F	+	1_{10} / 0x1	10000_{10} / 0x2710
500_{10} / 0x1F4	-	500_{10} / 0x1F4	0_{10} / 0x0
25_{10} / 0x19	*	25_{10} / 0x19	625_{10} / 0x271

3 Displaying the Result

Given Task

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. You can use the BL sendchar instruction to display an ASCII symbol on the console. This instruction displays the symbol corresponding to the ASCII code contained in R0.

Pseudocode

N.B.: *Continuation of expression evaluation pseudocode above.*

```
int quotient = 0;
int remainder = 0;
int powerOfTen = 10;
printkey('=');

while (remainder > 9) {
    remainder = total; //total calculated at expression evaluation
    if (remainder >= 10)
    {
        while (remainder > powerOfTen)
        {
            quotient = quotient + 1;
            remainder = remainder - powerOfTen;
        }
    if (quotient > 9)
    {
        powerOfTen = powerOfTen * 10;
        quotient = 0;
    }
    else {
        output = quotient + 0x30;
        printkey(output);
        powerOfTen = 10;
        total = remainder;
        if (remainder >= 10)
        {
            quotient = 0;
        }
    }
}
remainder += 0x30;
output = remainder;
printkey(output);
```

Solution

Registers for the quotient and the remainder are set to 0, another register is set to 10 and will be used to calculate the required powers of ten for the total. For example: if the total were 8562, the powers of ten required to print the number would be 10^3, 10^2, 10^1 and 10^0 or 1000, 100, 10 and 1. The total would be divided by 1000, leaving a quotient of 8 which would be sent to the console. The remainder of 562 would then be divided by 100, leaving a quotient of 5, which would be sent to the console, the new remainder of 62 would be divided by 10, leaving 6, which in turn would be sent to the console. The final remainder of 2 would also be sent to the console and the program would terminate as the result has been displayed fully.

The above code works on this principle but has there are a few differences. Firstly the total is checked and if it is less than or equal to nine, it will skip the while loop and print the single ASCII character. If it is more than nine, as in the example of 8562, the while-loop will start. Another while-loop divides the total by the current power of ten, getting the quotient. This quotient is checked, if it is more than nine, the power of ten is increased, and the quotient is reset to zero. The total is divided by 100 and gets the quotient. The quotient is checked again to see if it is more than nine. If true, the power of ten is increased once again. However, if this returns false, the quotient must be less than or equal to nine so it converts the numeric value to an ASCII character code by adding 0x30 and prints it to the console using BL sendchar. The power of 10 is reset to 10 and the remainder is moved into the total variable.

When the final digit is reached (the remainder is less than or equal to nine) the while-loop is not used and a different part of the program prints the final digit to the console and the program terminates.

Example

Below is a step-by-step walkthrough of the display result assembly code using 8562 as an example:

- Firstly the equals character is sent to the console and the register R11 is set to 0. It will be used to store the quotient. R9 is set to 10, this register will be used to store the current power of ten.
- The total (8562) is moved into a new register (R6), which will be from now on called remainder as will be used to store the remainder.
- Since 8562 is more than 9, the program enters the while-loop. The remainder is then compared with the current power of ten. Since it is more than the current

power (which is 10) the next while-loop. This loop divides the remainder by the power of 10 (currently 10). On each iteration the quotient is incremented and the remainder has the power of ten taken away from it. This loop continues until the remainder is less than or equal to the current power of ten. After a number of iterations this will be true, at this point the quotient will be 856.

- Moving through the code, 856 is more than 9, so the power of ten is increased, making it 100, and the quotient is reset. The while-loop runs for its second iteration.
- On this iteration many of the same things happen as on the first. However, this time 8562 is divided by 100, leaving 85 in the quotient.
- 85 is still more than 9, so the power of ten is increased once again, and the quotient is also reset. The current power of ten is now 10^3 or 1000.
- On the third iteration, 8562 is divided by 1000, leaving a quotient of 8 which is not more than 9. So the else statement is executed. The quotient of 8 is converted to its corresponding ASCII character code by adding 0x30. The ASCII value 0x38 is printed to the console.
- The power of ten variable is reset to 10^1 (10) and the remainder of 562 is moved from R6 to R5 (from remainder to total).
- The while-loop runs another iteration, once again moving the total into the remainder and dividing 562 by 10 firstly. This returns 56, more than 9 and the power of 10 is increased. 562 is divided by 100 leaving a quotient of 5. This is not more than 9 so the ASCII character ‘5’ is printed to the console in the same way as above.
- The remainder of 62 is then divided by the current power of ten which has been reset to 10, which leaves a quotient of 6. This is then printed to the console in the same way as ‘8’ and ‘5’.
- The while-loop attempts to run another iteration but the condition is not returned as true because the remainder is not more than 9. The code jumps to another part of code which converted the remainder of 2 to an ASCII character and prints it to the screen.
- The result of 8562 has now been displayed in the console and the program terminates.

4 Extra Mile

Given Task

For each of the three stages in this assignment, 20% of the marks available will be awarded for going the "extra mile". How you do this is up to you. The following are some examples of features that may be worthy of these marks:

- Allowing the user to enter either positive or negative values
- Evaluating expressions with three or more operands
- Formatting the result in some way (e.g. preventing leading zeros from being displayed)
- Correctly displaying negative results

Solution

In the program the following were implemented to go the 'extra mile':

- Evaluation of two or three operands
- Formatting the result correctly by preventing leading zeros from being displayed

These were implemented into the final DisplayResult.s source file. There is pseudocode available in the source file which shows how both of these extras function.

The evaluation of three operands ensures that the correct order of operations, frequently referred to as BIMDAS, is followed. This means the program will evaluate both of the operators input by the user and will give multiplication precedence over addition or subtraction. For example the expression 3 + 9 x 3 should give a result of 30, not 36, which would be the answer if the expression was evaluated from left to right. The user would enter the expression as 3+9*3 in the program and the 9*3 would be evaluated first, then its result (27) would be added to 3, giving the final result of 30.

The prevention of leading zeros is done by evaluating the result by increasing the power of ten on each iteration and then printing the result when the correct one is found and repeating this for each digit in the result. Another way of doing this is starting with a large power of ten which would start at the left-most digit of the 32-bit register and work its way to the right. However using this method means that leading zeros will be printed in most cases. The alternative method is what this program uses and is described in detail above under the 'displaying the result' heading.