Assignment #1
Simple Calculator

23 October 2017

This is an individual assignment and will contribute 10% of your final mark for CS1021.

You will be asked to demonstrate your solution during the labs on Friday 17th November 2017. Marks will be awarded for this demonstration.

You must submit your solutions using Blackboard no later than 23:59 on Monday 13th November 2017. Late submissions without a satisfactory explanation will receive zero marks.

Submit your `.s` assembly language source files and your report **in PDF format** as attachments to "Assignment #1" in Blackboard.

Your `.s` assembly language source files must be suitably commented.

Solutions will be checked for plagiarism.

Introduction

In this assignment, you will develop a program capable of evaluating simple arithmetic expressions entered by a user using the keyboard. You will develop the program in three stages. In Stage 1, you will write a program that can read unsigned values entered by a user using the keyboard. In Stage 2, you will extend your program to read and evaluate a simple arithmetic expression and store the result in a register. In Stage 3 you will extend your program further to display the result of the expression on the screen.

Important Note: You must submit your solution to each stage of the assignment as a separate .s ARM Assembly Language source file. Stages 2 and 3 build on Stages 1 and 2 respectively, so you should begin these stages by copying and pasting your solution from the preceding stage.

1 Console Input

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 213_{10} (or 11010101_2 or $D5_{16}$) in `R4`.

The **ConsoleInput** µVision project contains a simple program that demonstrates how to read ASCII characters from the console one character at a time, as they are entered by a user. A single character is read by executing the `BL getkey` instruction. This instruction only completes

execution when the user presses a key. When the instruction completes, the ASCII character code corresponding to the key pressed will be stored in `R0`.

The key pressed by the user is not automatically displayed in the console window so we need to do this ourselves with the `BL sendchar` instruction. This instruction displays the character corresponding to the ASCII code contained in `R0`.

Example: If we execute `BL getkey` and the user types 'b' on the keyboard, then 0x62 will be stored in `R0`. If we then execute `BL sendchar` (with 0x62 in `R0`) then 'b' will be displayed on the console.)

Important Note: Executing the `BL getkey` or `BL sendchar` instructions may overwrite the contents of `R1`–`R3` so you should avoid using these registers to store any values that you will need after executing either of these instructions. You must also avoid using `R13`–`R15`, as usual.

The program should stop reading ASCII characters when the ASCII `Carriage Return` character is read. This should occur when the user presses the **RETURN** key. The `Carriage Return` character has ASCII code **0x0D**.

Verify that your program stores the value entered in `R4`.

Using the µVision Console

The program above requires you to enter text using a "console". To simplify this, you should only test your program in Simulation mode, which simulates the ARM hardware. (Using a "console" with the real ARM hardware is more complicated.)

The template project is already configured to simulate the hardware and also open a "console" window called "UART #1" when you start debugging your program. (If for any reason the "UART #1" window does not appear, you can re-open it by selecting the **View – Serial Windows – UART #1** menu option.)

When testing your program, you can either step through the program one instruction at a time or use the run icon to run the program without stopping. If you are stepping through one instruction at a time, after executing `BL getkey` you will need to mouse-click in the "UART #1" console window and press a single key. Your program will then continue to the next instruction.

Alternatively, if you choose to run the whole program, you can mouse-click inside the console window and enter the full expression. To see the result of your program in `R4`, you will need to halt your program with the **Halt** icon (beside **Run**).

2 Expression Evaluation

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 solution should be an extension of the program that you developed for Stage 1. You should use the **ExpEval** µVision project to develop your solution.

The following are examples of valid expressions that your program should be capable of han-

dling:

```
100+250   (result should be 350)
90*3      (result should be 270)
25-5      (result should be 20)
```

Your program should support the addition (+), subtraction (-) and multiplication (*) operators. Each expression will contain two operands (values) and one operator. You may assume when writing your program that the user will be "well behaved" and will only enter valid expressions with operands and results that fit in 32-bit registers. You may also assume that the expressions will always produce results that are positive or zero.

3 Displaying the Result

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.

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`.

Use the **DisplayResult** µVision project to develop your solution, which should be an extension of your solution to Stage 2.

Suggested Approach

You can use the **Divide** program developed in a recent lab exercise to convert a value stored in a register to the ASCII characters representing the value. One approach is to divide the value in `R5` by decreasing powers of 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 next lower power of 10.

For example, given the initial value 4128,

- Dividing 4128 by 1000 will give you a quotient of 4 and a remainder of 128. Therefore $0x04 + 0x30 = 0x34$ (character '4') will be the first ASCII character to be displayed.
- Dividing 128 by 100 will give you a quotient of 1 and a remainder of 28. Therefore $0x01 + 0x30 = 0x31$ (character '1') will be the second ASCII character to be displayed.
- etc.

You will need to choose an appropriate power of 10 to begin with. You can use the **Power** program from the lecture slides to compute powers of 10.

Documentation

You are required to document your approach to the development of the Simple Calculator program. Your documentation should be in the form of a typed document **in PDF format**. This document must be submitted with your three assembly language `.s` source files.

Your report must contain the following:

A description of your solution to each of the three stages in the assignment. Your description should make use of examples, diagrams and pseudo-code where appropriate. **Your description should not take the form of a "narrative" for your assembly language program ("I moved the value from R5 into R4. Then I added R4 to R3 and stored the result in R0"). Such narratives will receive very low marks.** Instead, try to describe how your program works at a conceptual level.

A detailed description of your methodology for testing each of the three stages from above. This must include an account of the inputs used to test your program. Your report should demonstrate that you have thoroughly tested each part of your program. You should give careful consideration to choosing inputs that test different parts of your program (e.g. both positive and zero inputs, inputs that produce positive results, inputs that produce zero results).

Explain why you are using each test input, document the output produced by your program. State whether the output is correct and, if not, state why you think it is incorrect. Consider presenting your tests in the form of a table.

Evaluation

The following broad marking scheme will be used for the assignment:

- Stage 1 – Console Input – 30%
- Stage 2 – Expression Evaluation – 15%
- Stage 3 – Displaying the Result – 30%
- Documentation – 25%

Note that marks will be awarded for both the content and presentation of your document. Solutions that are merely working will not automatically attract 100% of the marks available. Marks will also be awarded for the quality of the solution.

Extra Mile

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