CS2208b Assignment 4

Issued on: Tuesday, March 15, 2016
Due by: 11:55 pm on Tuesday, March 22, 2016

For this assignment, only an electronic submission (***attachments***) at owl.uwo.ca is required.
- Attachments must include:
    - o ***ONE pdf*** file that has the three flowcharts, program documentations, and any related communications.
    - o ***Text*** soft copy of the assembly programs that you wrote for each question (*one program attachment per question*), i.e., ***THREE assembly*** source files in total.
- So, in total, you will submit 1 + 3 = 4 files.
- **Failure to follow the above format may cost you 10% of the total assignment mark.**

Late assignments are strongly discouraged
- 10% will be deducted from a late assignment (up to 24 hours after the due date/time)
- After 24 hours from the due date/time, late assignments will receive a zero grade.

In this assignment, you will use the *micro Vision ARM simulator* by *Keil*, which is an **MS Windows** based software, to develop the required programs in this assignment. The simulator (version 4) has been installed on all PCs at SSC-1032 and HSB-14 labs.

The *Keil micro Vision* simulator may also be installed on your Windows PC. You just need to download it from OWL and install it.

Programming style is very important in assembly language. It is expected to do the following in your programs:
- Using macros for the constants in your program to make it more readable.
- Applying neat spacing and code organization:
    - o Assembly language source code should be arranged in three columns: *label*, *instruction*, and *comments*:
        - ▪ the *label* field starts at the beginning of the line,
        - ▪ the *instruction* field (opcodes + operands) starts at the next TAB stop, and
        - ▪ the *comments* are aligned in a column on the right.
- Using appropriate label names.
- Commenting each assembly line

Great Ways to Lose Marks
- Not grouping your lines into logical ideas
- Not using any whitespace at all
- Not bothering to comment
- Commenting the code by just stating what you're doing, instead of why, e.g.,
  ```
  MOV r0, #5    ;move 5 into r0
  ```
- Not paying attention to the programming style (see the previous paragraph)
- Handing it in as soon as it assembles without testing and/or trying to break your code

Strings

A string is an array representing a sequence of characters. To store a string of *n* characters in your program, you need to set aside *n+1* bytes of memory. This area of memory will contain the characters in the string, plus one extra special character—the *null* character—to mark the end of the string. The *null* character is a byte whose bits are all zeros (`0x00`). The actual string consists of any group of characters, which none of them can be the *null* character.

QUESTION 1 (25 marks)

Draw a *detailed flowchart* and write an ARM assembly language *program* to concatenate two stings (**STRING1** and **STRING2**) and store the result in a *null* terminated **STRING3.** Assume that the length of **STRING1** + the length of **STRING2** ≤ 255.

Your code should be highly optimized, i.e., use as little number of instructions as possible.

You may want to define the strings as follow:
```
STRING1 DCB "This is a test string1"   ;String1
EoS1    DCB 0x00                        ;end of string1
STRING2 DCB "This is a test string2"   ;String
EoS2    DCB 0x00                        ;end of string2
STRING3 space 0xFF
```

QUESTION 2 (35 marks)

Draw a *detailed flowchart* and write an ARM assembly language *program* to copy a *null* terminated **STRING1** to a *null* terminated **STRING2**, underline removing any occurrences of the word "*the*" in **STRING1**. I.e., if **STRING1** is "**the** woman and **The** man said **the**" then **STRING2** would become " woman and **The** man said ". However, if **STRING1** is "and **the**y took brea**the**" then **STRING2** would become "and **the**y took brea**the**" without any change.

Your code should be highly optimized, i.e., use as little number of instructions as possible.

You may want to define the strings as follow:
```
STRING1 DCB "and the man said they must go"   ;String1
EoS     DCB 0x00                              ;end of string1
STRING2 space 0xFF
```

QUESTION 3 (40 marks)

Draw a *detailed flowchart* and write an ARM assembly language function (subroutine) that takes a data value stored in register **r0** and returns a value in **r0** as well. The function returns $y = a \times x^2 + b \times x + c$ where *a*, *b*, and *c* are signed integer parameters built into the function (i.e., they are not passed to it) and are defined in the memory using three **DCD** instructions. The subroutine also performs clipping, i.e., if the output is greater than a value *d*, it is clipped to *d*, where *d* is another parameter defined in the memory using a **DCD** instruction. The input in **r0** is a signed integer binary value. Apart from **r0**, underline no other registers may be modified by this subroutine, i.e., if you want to use any register as a working register, you have to store its value in a safe place first prior changing it, and to restore this value before returning from the function.

After implementing the function, write an assembly program which stores a value in **r0** and calls your function. Once the control is returned back from the function, the program will double the returned value and store this doubled value in **r1**.

Your code should be highly optimized, i.e., use as little number of instructions as possible.

Example1: if *a = 5, b = 6, c = 7, d = 90*, and **r0** = *3*,
then the returned value in **r0** should be 70 and the value in **r1** will be 140.

Example2: if *a = 5, b = 6, c = 7, d = 50*, and **r0** = *3*,
then the returned value in **r0** should be 50 and the value in **r1** will be 100.

Example3: if *a = -5, b = 6, c = 7, d = 10*, and **r0** = *3*,
then the returned value in **r0** should be -20 and the value in **r1** will be -40.