CS2208b Assignment 4

Issued on: Thursday, March 15, 2018

Due by: 11:55 pm on Thursday, March 22, 2018

For this assignment, only an electronic submission (***attachments***) at owl.uwo.ca is required.

- Attachments must include:
  - ***ONE pdf*** file (named **`report2.pdf`**) that has the one flowchart, program documentations, and any related communications, if any.
  - ***TWO Text*** files (named **`question1.s`** and **`question2.s`**) that have softcopy of the assembly source programs you wrote for each question (*one program per file*), i.e., ***TWO assembly*** source files in total.
- So, in total, you will submit 1 + 2 = 3 files (**`report2.pdf, question1.s`** and **`question2.s)`**
- **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

Programming style is very important in assembly language. It is expected to do the following in your programs:

- Using EQU directive to give a symbolic name to a numeric constant to make it more readable.
- Applying neat spacing and code organization:
  - 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
- Commenting *each* logical part of your code.

Great Ways to Lose Marks

- Not grouping your lines into logical ideas
- Not appropriately using whitespace
- Not bothering to comment your code
- 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)
- Not optimizing your code by using unnecessary assembly instructions. The more instructions in your program the less your mark will be.
- Handing in your code as soon as it assembles, without testing and validating your code
- Not using proper flowchart symbols
- Not following the flowchart rules

QUESTION 1 (50 marks)

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 allocated 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.

Draw a *detailed flowchart* and write an ARM assembly language *program* to copy a *null* terminated **`STRING1`** to a *null* terminated **`STRING2`**, after 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 they took breathe"`** then **`STRING2`** would become **`"and they took breathe"`** without any change. You can assume that **`STRING2`** will be *less than* 255 characters.

Your code should be highly optimized. Use as few instructions as possible (as little as 30 assembly instructions only, NOT including assembly directives or data definitions)!!.

Define the data of this program in a separate DATA area.

Define the strings as follow:

```
STRING1 DCB "and the man said they must go"   ;String1
EoS     DCB 0x00                              ;end of string1
STRING2 space 0xFF                            ;just allocating 255 bytes
```

More test cases:

```
"the the   the 123    the" ➔ "     123    "
"" ➔ ""
"the" ➔ ""
"The" ➔ "The"
"them   the   the1" ➔ "them      the1"
```

QUESTION 2 (50 marks)

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 calculates $y = a \times x^2 + b \times x + c$ 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`** assembly directives. If the value of *y* is greater than a value *d* (where *d* is another parameter defined in the memory using a **`DCD`** assembly directive), the function will return the value of *d*. Otherwise, it will return the value of *y*. The input value (i.e., the value of **`r0`**) is a *signed* integer value.

Apart from **`r0`**, 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 loads the value of *x* from the memory to **`r0`** and calls your function. The value of *x* is defined in the memory using a **`DCD`** assembly directive. 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. Use as few instructions as possible (as little as 14 assembly instructions only, for both the program and the function, NOT including assembly directives or data definitions)!!.

Define the data of this program in a separate DATA area.

Example1: if $a = 5$, $b = 6$, $c = 7$, $d = 90$, and $x = 3$,
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 $x = 3$,
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 $x = 3$,
then the returned value in **`r0`** should be -20 and the value in **`r1`** will be -40.