CS2208A Assignment 5

Due by: 11:55 pm on Tuesday, November 29, 2022

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

- Attachment is:
  - ***ONE Text*** file (named `question.s`) that has the softcopy of your assembly source program.
- **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 Keil's micro Vision ARM simulator to develop the required programs.

Programming Style

The programming style is essential in assembly language.
It is expected to do the following in your programs:

- Using the EQU directive to give a symbolic name to a numeric constant to make it more readable.
- Applying neat spacing and code organization (*do not use TABS, instead use SPACES*).:
  - 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 on *each* assembly line
- Commenting on *each* logical part of your code.

Great Ways to Lose Marks

- Not appropriately using spaces to lineup your program
- Not appropriately using EQU to make your code more readable
- Not appropriately using labels to make your code more readable
- Not bothering to comment on 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 grouping your lines into logical ideas
- 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

QUESTION 1 (20 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 string's characters plus one extra unique character—the *null* character—to mark the string's end. 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.

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***" (case sensitive) 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* 127 characters.

Your code should be highly optimized. Use as few instructions as possible (as little as 30 assembly instructions only, NOT including any 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 0x7F                              ;just allocating 127 bytes
```

More test cases:

"the the the 123 the" ➔" 123 "

"the, the the 123 the." ➔"the, 123 the."

"" ➔ ""

"the" ➔ ""

"The" ➔ "The"

"them the the1" ➔ "them the1"

"4the the 4the The the the1" ➔"4the 4the The the1"

ASCII Table

'0' ➔ 0x30
'1' ➔ 0x31
'2' ➔ 0x32
...
'8' ➔ 0x38
'9' ➔ 0x39
...
'A' ➔ 0x41
'B' ➔ 0x42
'C' ➔ 0x43
'D' ➔ 0x44
'E' ➔ 0x45
'F' ➔ 0x46
...
'X' ➔ 0x58
'Y' ➔ 0x59
'Z' ➔ 0x5A
...
'a' ➔ 0x61
'b' ➔ 0x62
'c' ➔ 0x63
'd' ➔ 0x64
'e' ➔ 0x65
'f' ➔ 0x66
...
'x' ➔ 0x78
'y' ➔ 0x79
'z' ➔ 0x7A