CS2208A Assignment 4

Due by: 11:55 pm on Tuesday, November 8, 2022

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

- Attachment is:
  - ***ONE Text*** file (named `question1.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)

Most goods sold in U.S. and Canadian stores are marked with a Universal Product Code (UPC). The meanings of the digits underneath the bar code (from left to right) are:

- First digit: type of item,
- The first group of five digits: manufacturer,
- The second group of five digits: product, and
- Final digit: check digit, used to help identify an error in the preceding digits.

To ***compute*** the check digit,

a) Add the first, third, fifth, seventh, ninth, and eleventh digits
b) Add the second, fourth, sixth, eighth, and tenth digits
c) Multiply the first sum by 3 and add it to the second sum
d) The check digit is the digit which, when added to the above sum, produces a sum that is multiple of 10

Example for UPC 0 13800 15073 8:

- First sum: 0 + 3 + 0 + 1 + 0 + 3 = 7
- Second sum: 1 + 8 + 0 + 5 + 7 = 21
- Multiplying the first sum by 3 and adding the second yields 42
- The check digit is 8, as 42 + 8 = 50 (multiple of 10).

To ***verify*** the check digit, you do the first 3 steps above from a) to c) and then add the sum to the check digit. Finally, you check if the result is multiple of 10 or not.

Write an ARM assembly program to ***verify*** whether a string of 12 ASCII encoded digits stored in memory is ***a valid UPC or not***. If valid, you should store 1 in `r0`, if not, you should store 2 in `r0`. **Your code should be highly optimized. Use as few instructions as possible (as little as 19 assembly instructions only, NOT including any assembly directives or data definitions)!!.**

R10: 1st
R9: 2nd.

Define in your assembly program the 12 digits UPC string as follow:

```
UPC     DCB "013800150738"    ;correct UPC string
```

To test your program, you can use the following UPCs:

```
UPC2    DCB "060383755577"   ;correct UPC string
UPC3    DCB "065633454712"   ;correct UPC string
```

You can also get more UPC codes from your kitchen items.

For an incorrect UPC, you can change any digit in a correct UPC to generate an incorrect UPC.

HINT 1: To load a byte to a register, use **`LDRB,`** not **`LDR.`**
HINT 2: To calculate 3 × Z, you can do so using only one ADD instruction with LSL#1 shift.
HINT 3: You can implement the division operation using repeated subtraction.

Digits' ASCII Code
'0' → 0x30
'1' → 0x31
'2' → 0x32
'3' → 0x33
'4' → 0x34
'5' → 0x35
'6' → 0x36
'7' → 0x37
'8' → 0x38
'9' → 0x39