CS2208b Assignment 5

Issued on: Tuesday, March 26, 2019
Due by: 11:55 pm on Tuesday, April 2, 2019

For this assignment, only an electronic submission (***attachments***) at owl.uwo.ca is required.
- Attachments must include:
  - ***ONE pdf*** file (named **report3.pdf**) that has a copy of the source code, the structure of the stack frame that you utilized in your program, and any related communications.
  - ***ONE Text*** file (named **power.s**) that has softcopy of the assembly source program that you wrote (*one program attachment per question*), i.e., ***ONE assembly*** source file in total.
- So, in total, you will submit 1 + 1 = 2 files (**report3.pdf** and **power.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 GEN labs*, ***except*** NCB-105.

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 (100 marks)

Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. A function is considered recursive if it calls itself.

The following function computes x^n recursively, where x is an integer number and n is a non-negative integer number.

```
int power(int x, unsigned int n)
{
  int y;

  if (n == 0)
    return 1;

  if (n & 1)                // if n is odd
    return x * power(x, n - 1);
  else                      // if n is even
  { y = power(x, n >> 1); // n >> 1 means n/2
    return y * y;
  }
}
```

Write an ARM assembly _**program**_ to calculate x^n _**using the above recursive function**_, where x and n are passed-by-value _through the stack_ (not via registers) to the function and the returned value is _stored in the stack_ just above the parameter. No other registers may be modified by the power function. Once the control is _completely_ returned back from the function (i.e., after calculating x^n), the returned value must be stored in a local variable (called _result_) in the main function. _**Your code should be highly optimized. Use as few instructions as possible (as little as 45 assembly instructions only for both the main program and the function, NOT including any assembly directives or data definitions)!!.**_

Sketch the structure of the stack frame that you utilized in your program.

You should utilize a big enough stack so that you can calculate x^n for various n values.

How many stack frames are needed to calculate x^n, when $n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,$ and 12?