CS2208a Assignment 5

Issued on: Monday, November 25, 2013

Due by: 11:55 pm on Monday, December 2, 2013

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

- Attachments must include:
  - ***ONE pdf*** file that has the two flowcharts, program documentations, and any related communications.
  - ***Text*** soft copy of the assembly programs that you wrote for each question (*one program attachment per question*), i.e., ***TWO assembly*** program files in total.
- So, in total, you will submit 1 + 2 = 3 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 MC-342 and MC-08 labs.

The simulator may also be installed on your PC. If you have not installed *Keil micro Vision 4 simulator* on your PC yet, you will need to download and install it from https://www.keil.com/download/product/

Note that during this month (October 2013), *Keil* has released a new version of its *MDK-ARM simulator* (*Keil micro Vision 5*). The *MDK-5* installation process has changed from a single monolithic installer to a set of installers.

So, you need to download *MDK-Core Version 5* from https://www.keil.com/download/product/ as well as the *Legacy support for ARM7 & ARM9 devices* from http://www2.keil.com/mdk5/legacy

The author of our text book has provided a *Quick Guide to Using the Keil ARM Simulator*. If you wish, you can access it at http://www.alanclements.org/usingkeilsimulator.html

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:
  - 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

QUESTION 1 (40 marks)

A linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data element and a *link* to the next node in the sequence Linked lists allow for efficient insertion or removal of elements from any position in the sequence.

Assume that we have a linked list, whose first node is pointed at by register `r0`. Each node in this list composed of a single 32-bit data element and a 32-bit address (*link*) pointing to the next node in the list. The last node in the list points to the null address 0.

Draw **a *detailed flowchart*** and write an ARM assembly ***program*** to search a linked list for the first node in the list whose data is equal to the data in register `r1`. On success, set register `r2` to `0xFFFFFFFF`, and register `r3` should contain the address of the desired node. On failure, set register `r2` to `0xF0F0F0F0`.
You code should be highly optimized, i.e., use as few number of instructions as possible.

Hint, for testing purpose, you can represent a linked list as follow:

```
List  DCD Item5, 0x12341111
Item2 DCD Item3, 0x12342222
Item3 DCD Item4, 0x12343333
Item4 DCD Item6, 0x12344444
Item5 DCD Item2, 0x12345555
Item6 DCD Item7, 0x12346666
Item7 DCD 0x00,  0x12347777  ;terminator
```

You should change the number of nodes and the value of the data elements in the list to test various cases, including empty linked list, one element linked list, and long linked list. You should also test your program by searching for values in various locations the list, as well as values not in the list.

QUESTION 2 (60 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 `n!` recursively, using the formula `n! = n × (n - 1)!`, where `1! = 1`.

```
int fact(unsigned int n)
{
  if (n <= 1)
    return 1;
  else
    return n * fact(n - 1);
}
```

Draw **a *detailed flowchart*** and write an ARM assembly ***program*** to calculate `n!` ***using the above recursive function***, where `n` is passed-by-value through the stack to the function and the returned value is stored in register `r0`. No other registers may be modified by the `fact` function.
You code should be highly optimized, i.e., use as few number of instructions as possible.

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

Based on the size of the stack that you selected, what is the largest `n` that you can successfully calculate its `n!`.