Lab 1 Assembly Programming

Deadlines & Grading

- **Prelab Due:** Friday, Feb 11 at 11:59 PM EST on Canvas.
- **Code and report submission Due**: Friday, Feb 18 at 11:59 PM EST on CMS.
- **Grading**: 100 points, 8% of final grade.

Prelab Questions (5pts)

In order to complete lab assignments successfully and on time, we encourage you to start working on each lab early. To encourage this and to help you clarify your tasks and clear up any lingering questions about the material, this lab has a pre-lab that is due a week before the actual submission. To complete the pre-lab you do not need to write any code, but you will need to understand what the lab is about and what approach you will be taking to structuring your code.

Step 1: Please read the entire lab! This is the time to clarify anything with the course staff.

Step 2: Please answer the following questions in 1-3 sentences each. Use your own words. You may want to type out the answers in a separate document and then cut/paste them into the submission form.

Question 1: For implementing Morse code, you need to keep the LED on/off of specific time intervals. How will you implement this timing?

Question 2: You will probably want to implement some other functions in addition to the ones required by Parts 2 and 3. Will they all need to push LR onto the stack? If not, which type of function can skip this step?

Question 3: You will need to do some math in registers when computing Fibonacci numbers. What do your choices imply for when/where they need to be saved and restored in terms of the caller/callee convention? How many registers do you think you will need and which ones do you plan to use?

Section I: Overview

Goal. The purpose of this lab is to give you experience writing assembly language programs for the FRDM-KL46Z microcontroller. To successfully complete this lab, you will need to understand the following:

- The ARM instruction set.
- The calling conventions for ARM.

NOTE 1 For each part of the lab, we recommend that you create a separate project in MCUXpresso. To do this, refer to lab 0.

Precautions.

- The micro-controller boards should be handled with care. Misuse such as incorrectly connecting the board is likely to damage the device.
- These devices are *static sensitive*. This means that you can "zap" them with static electricity (a bigger problem in the winter months). Be very careful of handling boards that are not in their package. Your body should be at the same potential as the boards to avoid damaging them. For more information, check wikipedia.
- It is your responsibility to ensure that the boards are returned in the same condition as you received them. If you damage the boards, it is your responsibility to get a replacement.

Section II: Assignment

Part 1: Morse Code (15 points)

The first assembly language program you write will read the value stored in register `R0`, and use the LED to signal its value. The signaling convention we will be using is Morse code. You may assume that the value stored in **`R0`** is between zero and nine.

The Morse code uses a combination of *dash*es and *dot*s to represent letters and numbers. A dot will be signaled by turning the red LED on for a brief duration (aim for 0.3—0.8 Seconds). A dash is signaled by turning the red LED on for three times the duration as a dot. The code for digits is:

Digit	Code
1	dot dash dash dash dash
2	dot dot dash dash dash
3	dot dot dot dash dash
4	dot dot dot dot dash
5	dot dot dot dot dot
6	dash dot dot dot dot

7	dash dash dot dot dot
8	dash dash dash dot dot
9	dash dash dash dash dot
0	dash dash dash dash dash

The delay between parts of the same digit is the same duration as one dot.

Write an assembly language program that uses the red LED to signal the value of `R0` using Morse code. Start from the template assembly program available in `lab1.s`. The template contains the assembly language instructions that turn the LED on and off.

To test your program, assemble it multiple times with different initial values of `R0` (modify the statement specified in the template program). We expect each run of your submitted program to display exactly one digit. It should be value stored in `R0` at the beginning of the program.

Hint: Use the structure of the Morse code to simplify your assembly language program.

Part 2: Procedure (20 points)

The functionality of this part of the assignment is identical to the previous part, except this time your assembly language program must implement the function and respect all the ARM calling conventions:

```
void MorseDigit(int n)
```

where `n` is an integer between zero and nine. This function should not have any side effects beyond changing the caller saved registers. We expect to be able to link this function into a C-program and call it.

To test your program, you must call your function from assembly language as well. Include your test cases in your submission. Your submission must include at least one call to your function. All functions and function calls should respect the ARM calling conventions. Use the template from part 1 for this part as well.

NOTE 2 A valid solution to Part 2 (one that conforms to all the calling conventions and implements the functionality correctly) is also a valid solution to Part 1. Hence, assuming you have a functional Part 1, the grade for this part will be based on the correct use of calling conventions. If there is an error in Part 1, but you implement the calling convention correctly, you can still get full credit for this part. Also, you can start Part 1 immediately without knowledge of calling conventions.

Part 3: Fibonacci (40 points)

The Fibonacci sequence is defined by the recursion $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for n>1. The following C program shows a simple way to compute the n^{th} Fibonacci number (which also returns zero for negative arguments):

```
int fib(int n){
     if (n <= 0) return 0;
     if (n == 1) return 1;
     return fib(n-1)+fib(n-2);
}
```

Implement this recursive function using assembly language. Then display the returned value of the call to `fib()` using the `MorseDigit()` function. Your program should be able to display up to `fib(6)`. To receive full credit, your implementation must have the same call structure as the C-code above and not compute the answer in some other way, e.g. a loop.

To test your program, call `fib()` from assembly language, and display the result by calling `MorseDigit()`. Include your test cases in your submission. Your submission must include at least one call to each of your functions (`fib` and `MorseDigit`). All function calls must respect the ARM calling conventions. Use the template from part 1 for this part as well.

Report (10 points + 10 points)

Please prepare a brief report (up to 4 pages) of your approach. Include the following sections.

Design: A high-level overview of your code. Did you make and design choices on the data-structure? Include a description and a diagram of the stack during the recursive call to `fib()`.

Coding: Include any specific details involved in implementing your design. Did you have to make any assumptions, i.e. number of processes, maximum stack use by each process?

Code Review and Testing: We would like to hear both about your high-level approach to test and any particular problems you encountered. Describe your testing strategy and how you ensured correct code functionality. Where there any things that were difficult to test for?

Figures: Drawing and diagrams can be very helpful to explaining what is going on. You are encouraged to use them and they will not count against the page limit. Please include them at the end, with a caption and refer to them in the text.

Style and Clarity: There are 10 discretionary points for style and clarity. Please use good formatting, meaningful variable names, and appropriate comments. By default, you will get all these points, but if reading your code is confusing, your submission violates formatting guidelines you will take off points.

Section III: Extra Credit

Make Part 3 work for fib(n), where n>6. You will need MorseDigit() to work for multi-digit decimal numbers. If you attempt this, include a description in your write-up and submit the code as a separate assembly file. Note that the delay between two digits should be equal to three dots.

All extra credit during the semester will be taken into account when assigning final grades (instructor's discretion).

Section IV: Submission

The lab requires the following files to be submitted:

- **part1.s**: for part 1
- **part2.s**: for part 2
- **part3.s**: for part 3
- **part3ec**.s: for the extra credit (optional)
- **report.pdf** : include the response for the write-up task.
-

All files should be uploaded to CMS before the deadline. Multiple submissions are allowed, but only the latest submission will be graded.

References for ARM Calling Convention

The following ARM document (AAPCS) describes the official Procedure Call Standard for the ARM architecture. The document discusses features such as co-processors and data types such as floating point numbers, which are not used in this course.

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042f/IHI0042F_aapcs.pdf