Lab 1 Assembly Programming

Deadlines & Grading

- **Code and report submission**: Friday, February ~~16~~ 15 at ~~5:00~~ 9:00 PM on CMS, **groups of 2**
- **Grading**: 5% of total grade

Meeting with a TA Mentor

In order to successfully complete a lab assignment in time, we encourage you to start working on each lab early. Learning how to work effectively as a team is also an important part of the labs, and working together when team members have different backgrounds, schedule constraints, and expectations often requires communicating and planning early.

We will assign a TA mentor for each group. The TA mentor will contact you to schedule a 15-min meeting in the coming week. Both of your group members should attend the meeting. At the meeting, you need to be prepared to explain your plan on how your group will work on the lab assignment. You may lose up to 5 points out of 100 if you do not meet your mentor or if you do not have a concrete plan at the meeting (grading: 0-no show, 2-unprepared, 5-prepared). Be prepared to answer the following questions.

- Explain the lab assignment in your words. What are the tasks that you need to do?
- How do you plan to approach the lab conceptually?
- How do you plan to divide the work between group members?
- When and where do you plan to meet and work on the lab? Provide specific dates and time. Set aside at least 4-5 hours to work together as a group.
- How are you and your partner communicating and sharing code?
- Do you have any logistical or technical problems regarding the lab?

The TA mentor can also give you advice and help you understand the lab assignment. However, we do not expect to be able to provide detailed technical help during the 15-min meeting. You can use Piazza and TA office hours (and go to any TA) for additional help on the lab.

We expect both members to contribute equally to the entire lab rather than simply partitioning the work in a way that each member only contributes to a subset of tasks. One way to achieve this goal is to have group members work on different aspects such as coding, reviewing, and testing while working together, alternating the roles. Everyone on the team should have an opportunity to play an active role in major tasks. You can read about pair programming in the links below, which is one such technique where two programmers work together.

Code Review (a.k.a. Peer Code Review)
https://en.wikipedia.org/wiki/Code_review

Pair Programming
https://en.wikipedia.org/wiki/Pair_programming

You can read about Google's research on effective teams in the following article.
https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html

Section I: Overview

Goal. The purpose of this lab is to give you experience writing assembly language programs for the FRDM-K64F 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 uVision. 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 ($35).

Section II: Assignment

Part 1: Morse Code (20 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. 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. *The delay should be chosen so that it is obvious to your grader that your lab is working*!

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 `Framework.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).

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

Part 2: Procedure (25 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.

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. You can start Part 1 immediately without knowledge of calling conventions.

Part 3: Fibonacci (45 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 function using assembly language. The result of the call to fib() should be displayed using the MorseDigit() function. Your program should be able to display up to fib(6).

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.

Write-up Task Describe the usage of the stack for fib(), including a detailed description of the stack frame. We expect to see a picture of the stack, showing stack frames for nested calls to fib(), along with annotations that describe the contents of the stack.

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. You also need to describe the work distribution, in a separate section titled "Work Distribution":
  a) How did you carry out the project? In your own words, identify the major components that constituted the project and how you conducted the a) design, b) coding, c) code review, d) testing, and e) documenting/writing. If you followed well-documented techniques (e.g., peer programming), this is the place to describe it.
  b) How did you collaborate? In your own words, explain how you divided the work, how you communicated with each other, and whether/how everyone on the team had an opportunity to play an active role in all the major tasks. If you used any sort of tools to facilitate collaboration, describe them here.

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

The assembly files should be commented in a way that makes it clear why your program works.

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 (SIMD) 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

The Apple's iOS guide on ARMv6 function calling conventions provides a simpler description of the basics of the ARM calling convention. In the Apple document, however, the discussion on the frame pointer (R7) is more specific to iOS and does not apply to the general ARM calling converntion.

https://developer.apple.com/library/content/documentation/Xcode/Conceptual/iPhoneOSABIReference/Articles/ARMv6FunctionCallingConventions.html

FRDM-K64F also supports Thumb-2 (ARMv7-M), which makes saving and restoring registers simpler.

https://developer.apple.com/library/content/documentation/Xcode/Conceptual/iPhoneOSABIReference/Articles/ARMv7FunctionCallingConventions.html