# ECE9047: Laboratory 1

John McLeod

Due Date: 2021 02 28

## *Objective*

This lab is about programming an ARM®Cortex-A9 on a DE1 SoC board in the ARMv7 assembly language. For this lab, you can use an online simulator to test your code.

- The simulator is available at https://cpulator.01xz.net/?sys=arm-de1soc.
  - This simulator runs in your browser and should work with any computer system.
  - This simulator is *not* recommended for tablets, but might still work.
  - This simulator is reported to run best in Firefox. If the emulator doesn't seem to work well in your browser of choice (Safari, Edge, Chrome, etc.), try Firefox first before complaining to me.
- Alternatively, if you have (or are willing to purchase) an actual ARM®Cortex-A9 and a suitable development board, then by all means do so! Please take some pictures to include with your report.

The goal of this lab is to implement the following system:

> **Write the first 10 *Fibonacci numbers* to a seven segment display. Show each number on the display for one second** $(1\,\mathrm{s})$ **before showing the next. Your program should endlessly loop, cycling through displaying these 10 numbers.**

The Fibonacci numbers are defined as $F_n$:

$$F_0 = 0, \quad F_1 = 1, \quad \text{and} \quad F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n > 1.$$

## *Deliverables*

You must submit your lab report online to the course website on OWL. Your lab report must include:

1. A statement of the problem. (You can copy this from the *objective* given above.)
2. An explanation of your solution, outlining how you implemented the solution in your code.
   - A flowchart and/or a description using pseudocode may be appropriate, especially if your actual code involves lots of branches and subroutines, and/or lots of hard-coded numbers.

   The goal here is to make sure someone else (the TA, in particular) can understand how your code solves the problem.
3. A description of how you approached and implemented the problem: what did you implement first, how and when did you test whether your code worked, etc.
   - There are at least three separate parts to this project (obtaining the Fibonacci number, activating the seven segment display, and pausing for $1\,\mathrm{s}$), unless you are very skilled at assembly you probably didn't write all of this code at once.
   - Reread section 2.6 of Valvano's textbook (available on OWL) on debugging theory if you aren't sure what to do or why I am asking you to do this.

The goal here is to make sure someone else understands how, starting from a blank page, you ended up with some working code. You do not need to describe every little step and mistake that was made, just enough to give an overview of your design process.

4. A qualitative cost/benefit analysis of your solution, to help justify your approach. This discussion should be in terms of time spent designing your solution, time spent actually implementing this solution in code, and time that might be spent in the future maintaining or extending this code. For example:

   - At one extreme, you might write a subroutine to calculate the $n^{\text{th}}$ Fibonnaci number, a subroutine to pause the program for an arbitrary amount of time, and a driver to display an arbitrary number on a seven segment display. This code would be very portable (drivers, timers, etc. could be reused for other projects) and easy to extend (trivial to modify code if we need the first 100 Fibonacci numbers), but might take a long time to implement (this is just an ECE9047 lab, after all, is it worth spending that much time on?).
   - At the other extreme, you might hard-code how to display the first 10 Fibonacci numbers and a $1\,\mathrm{s}$ timer. This code is very simple and readable (mostly linear, almost no branches) and probably quick to write (why waste your time on this silly course?), but isn't portable and would be difficult to extend (what if the client for this million-dollar project decides they want the first 100 Fibonacci numbers instead?).

   The goal here is to make sure someone else understands your decision making process that lead to the solution you are providing.

5. A working copy of your assembly code. The TA is going to copy+paste this into the simulator to test that it works. To make this as simple as possible, *please submit your code as a separate plain-text file (i.e. in TXT format, from Notepad or something equivalent)*.

Your report should be in a common document format (probably PDF, DOC, or DOCX) and should be reasonably professional: written in complete sentences, appropriate spelling and grammar, and appropriate use of graphics. "Appropriate use of graphics" is deliberately vague and subjective.

- It is possible to produce an excellent lab report with no graphics at all, but often block diagrams, flow charts, and/or screenshots of the simulator may help clarify your written text.
- You should avoid placing full screenshots of the simulator if you are only calling attention to particular parts (i.e. sample seven segment display output, or sample register content, etc.). In that case, make sure you crop down the screenshot so only the relevant portions are shown in your report.

Your report should not be excessively long. Probably between two and five pages is a good guideline.

## *Grading Scheme*

Your report will be graded by the following scheme.

*5%* Statement of the problem. Marks awarded based on how clearly the problem is stated.

*15%* Explanation of your solution. Marks awarded based on how clearly you explain how your approach actually solves the problem, and on how easily the TA can tell that the explanation here matches the source code provided.

*15%* Description of your approach and how you implemented that solution. Marks awarded based on how clearly you explain, step-by-step, how you ended up with working code. Marks will be deducted if this section is too detailed (i.e. don't explain how/why you wrote each *line of code*, just each *section of the program*).

*15%* Justification of your solution. Marks awarded based on how clearly you explain all the factors that went into your decisions on how to solve the problem.

*40%* Working code. Full marks awarded if the TA can copy+paste your code into the simulator and it correctly implements the solution. *Marks may be deducted if the TA needs to do excessive formatting to get your code into the simulator! Please submit your code as a separate plain-text file (i.e. in TXT format, from Notepad or something equivalent)*.

*10%* Writing quality. Marks awarded based on the extent to which your report is written coherently, in paragraph form, with complete sentences. Correct spelling and grammar are also included. Marks may also be deducted if the report is exhaustively long.

### *Resources & Lab Session*

A reference document for the necessary code in the ARMv7 language is available on OWL. The lesson material will also help explain some of these concepts.

- At some point (to be scheduled) there will be a live tutorial session over *Zoom* to demonstrate some of the functionality of the simulator and some basic assembly language.
- Attendance is optional, and if you are familiar with low-level programming you probably do not need to attend this session — it will be very beginner-friendly.