ARM11 Checkpoint Report

Miroslav LAMBREV, Aahil MEHTA, Radostin PETROV, Ruari PHIPPS

June 14, 2019

1 Group Organisation

Our group work started by creating a shared Google Document, in which we drafted the project structure and interpreted the spec, so that the information is more condensed and accessible. The first thing we discussed was what data structures we would use for implementing the memory and the registers. After that we decided on the global variables that we would define and created a simple skeleton file of our initial implementation of the emulator, which we pushed to git.

Using this as a base, we split the work into smaller chunks and started building the original implementation with the necessary functions. Although we assigned different jobs to group members, we would often collaborate or switch roles, if the person was having difficulties with their part of the work. As we only worked with a single `.c` file and we were inexperienced with git, we initially used a private Dropbox folder for our emulator. After gaining some confidence with git, we started pushing our code there, because it was a lot more convenient. We used the shared document to create milestones and plan ahead.

Our teamwork was slightly unstable at first, given that some team members were absent for the first few days, however, it greatly improved after we all started meeting and working together on the project. It seems that our team has a good mix of skills with some people more experienced on the logical and implementation side of the project, while others have an understanding of the technical side. Whenever someone encountered a problem, there would always be a team member able to help.

A misunderstanding both on our part and the mentor caused us to reschedule our first meeting with him for Tuesday May 28. The meeting went well, however, our mentor was not able to answer some of our questions as they were too closely connected with implementation of the code. By then we had already created our first version of the emulator, although we had formatting errors and we were not passing the tests.

2 Implementation of `emulate.c`

Our emulator has the following structure:

- The `emulate.c` file contains the `main` function, which takes a filename containing the binary object code. We first initialise the state of the ARM machine (*all memory locations and registers become 0*) and then load the binary file into memory. Then, we start our three cycle (**execute-decode-fetch**) pipeline and finally, after executing all instructions or executing the halt instruction, the `main` function prints the register states to standard output.

- `pipeline.c` contains our three functions for our three cycle pipeline. The `execute` function uses a switch statement on the type of instruction and then calls to a helper function to execute each decoded instruction based on its type.
- The `execute.c` source file has the definitions for our execute helper functions.
- `utils.c` file contains helpful utility functions that we use throughout the emulator, like bit operations, shift operations, as well as functions to check for overflow or whether the Cond field is satisfied by the CPSR register and an output function to display all the registers and memory changes to standard output.

For each of the `.c` files we have their respective header file, which contains the dependant libraries, as well as a header file, used to define constants and type definitions.

- `definitions.h` contains the definitions of all constants and type definitions that we have used.

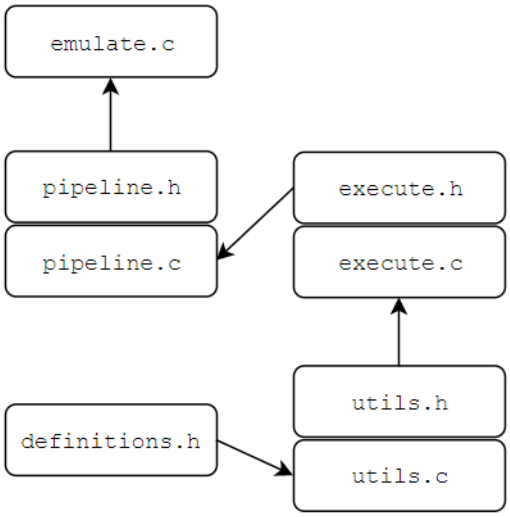

Figure 1: Dependency Diagram for `emulate.c`

3 Implementation of the other parts and the extension

Bearing in mind our late start with the emulator and our current progress, we believe that we will have enough time to finish the rest of the project. We have already started implementing the assembler and our work management is greatly improved.

Our team is now more experienced with git and we therefore have decided to split the work by creating a list of issues. Each team member would flag the issue and start working on it individually on their own branch, and merge with master after running basic tests on the code.

We plan to finish Parts II and III by the beginning of next week, which should allow us enough time for the extension. We have still not thought about our extension, but we want to give it roughly 50% of time spent on the group project.