C-Programming Project
ARM Checkpoint Report

Louis Aldous, Tao Wang,
Mohammed Abdus Samad Hannan, Armaan Sabharwal

June 2021

The implementation of the emulator gave our group an effective working process that we will use for later stages of this project and was a helpful introduction to working collaboratively using Git.

To spread the work across the group, the specification for the emulator was partitioned into sections and group members were assigned a section to work on.
The work was split across the group as follows:

- One team member was responsible for the control flow of the program, e.g. loading the instructions from the binary file, the fetch-decode-execute cycle etc. Branching instructions were also included in this category as it required the pipeline to reset its next decoded instruction after a successful branch so its implementation was simplified by having the same member do this.
- Two members were assigned to implementing the data processing instructions as it was the largest section. It required the implementation of many different instructions which was also in addition to operations such as number shifts and rotations. Assigning more than one member of the group to this task had the fortunate impact of allowing more efficient debugging after the emulator was done. As this section is the largest, there is a greater chance for bugs to occur. Therefore, having two members assigned to this section allowed for quicker error spotting.
- The final member implemented multiply and single data transfer instruction handling. These sections together were a good size for a single team member to do as it didn't require the tedious programming of individual instructions like the data processing section did. The implementation of single data transfer instructions also required the implementation of the memory of the emulated ARM machine, hence emulating the memory was also included in this section of work.

Our group has been very effective in working together to complete the emulator. Every change we make to our code goes to a new branch which allows the code to be peer-reviewed before it is merged with master. Numerous times this has led to cleaner and more efficient code so everyone is satisfied with this approach. Any deadline made by the group to get a section of work done is always met and, in many cases, work is completed earlier than expected. All communication is done through a group chat which allows for quick communication between members of the group. This has been especially helpful in cases where someone has a question about another member's code or when a group member asks another member to review their code for errors or any possible optimisations. In future stages of this project, we are planning to make our commit messages more descriptive of what they change. The group plans to do this through a technique called *semantic commit messages.*

As a group we decided to keep **emulate.c** as small as possible in terms of code size, instead relying on separate C source and header files to implement specific parts of the emulator. For example, we had separate files for the pipeline (pipeline.c) and each instruction type also had its own source file (e.g. *singledatatransfer.c*). This allowed us to treat each part as its own module and especially helped to prevent conflicts between multiple group members editing different files at the same time. In addition, this will allow us to reuse code in later parts of the project without difficulty as it helps with organisation, therefore it won't take long to find a specific utility function that we wish to use. For example, we could reuse our little endian converter either in the assembler or in the extension to convert a binary number from big endian to little endian representation.
The entry function in **emulate.c** only consisted of a few calls to functions in other modules, the flow was as follows:

1. Create an instance of an ARM machine to be emulated and initialise its memory and registers to their correct starting values.
2. Load instructions from a binary application file to the machine's memory.
3. Execute the pipeline and fetch-decode-execute until an all-zero instruction is executed.
4. Output the state of the emulated ARM machine (i.e. its register values and any non-zero memory) and terminate the emulator.

All compilation was done through makefiles to prevent slowdown as the number of files in the emulator grew.

We are confident that the future stages of the project won't have any major issues in terms of implementation. However, thinking of ideas for a unique and useful extension to our project has proven to be difficult. To mitigate this we will continue to brainstorm as many ideas as possible throughout the implementation of the assembler to ensure we have a good understanding of what we want to create for an extension. Creating the assembler will give us more experience with the project and will therefore give us more potential ideas for what we can do as an extension and so by the time the assembler is implemented, we plan to know what we want to produce for the final part of the project.