**Checkpoint Report**

When we were given the specification, we all read through it and decided the best way to split it between us, was for each of us to tackle one different type of instruction in the emulator; Branch, Data Processing, Single Data Transfer and Multiply. So Ayman was given Data Processing, Joseph had Single Data Transfer, Pavlos had Multiply and Nikhita had Branch. However, when we proceeded with the project, we realised the different instructions consisted of different amounts of work, for example, Data Processing contained a lot more work than Branch. Instead, Ayman took on the binary file loader, the auxiliary functions (for example, printing the registers and the memory). Pavlos and Joseph completed the conditions, and then Pavlos implemented them in the pipeline, which he also did. Joseph continued with the Single Data Transfer operations, but also did the branch, and Nikhita did the barrel shifter, and the basic processing instructions and put it all together in Data Processing. Finally, we merged it all in git, and then worked on the debugging together.

The group worked very well together, the splitting of the tasks was agreed on by everyone and nobody felt like they were left out of the group discussions. We divided the tasks without any conflicts, and we organised group chats to keep in contact with each other. We each made a git branch, to complete our assigned tasks in, and then merged when we had completed them. We met daily, to discuss how the project was going and any possible problems, which we then resolved with each other's help. This helped us work more effectively, as nobody was left behind in the ongoing project. In the future however, we feel that we could designate more individual tasks for the group members, as sometimes it felt like we were all sat around a computer, as one person did the typing.

We decided to split the emulator into four functions;

void initialise() - This allocates memory to the different structures, and clears the registers, basically setting the processor to its initial state.

Void loadBinary(filename) – Loads the contents of the file into memory

pipeline(ARM11 \*armPtr) – Implements the three stage process of the Pipeline; fetch, decode, execute

printRegisters(), printMemory() - Prints out the contents of the registers, and all non-zero memory locations

We had a basic structure for the ARM processor, which contained all the general purpose registers, the PC, CPSR register and a pointer to the memory. A pointer to the ARM processor would then be passed to the pipeline, which would execute each instruction by passing it to the relevant function.

We can reuse the basic auxiliary functions for the assembler, such as getBits and printBits. However it is difficult for us to know what we will reuse at this point in time.

From our experience completing Part 1 of the project, we realise it is difficult to split up the tasks evenly between the group members, so that each member is doing the same amount of work. Also, we feel it will be challenging to merge our branches later on during the project, as we each create more and more code. It will be hard to maintain the merged code, and make sure nothing is lost. However, as we become more experienced with Git, hopefully we will face these problems less often.