ARM Checkpoint Group 19

Zachary Lazar, Serhii Popov, Adam Marshall, Arya Narang

June 6, 2024

1 Group Organisation

We began work on the project by completing sections 1.1-1.3 together, and then split the remaining sections of part 1 among ourselves so that we were able to work remotely. Serhii handled the decoding step for each section, Adam handled the execution of data processing instruction with immediate addressing (1.4) and the halting instruction (1.9), Zakk the execution of data processing instructions with registers (1.5) and branch instructions (1.8), and Arya the execution of shifts to operands (1.6) and single data transfer instructions (1.7). This approach enabled us to work on the project at our own pace when necessary.

We used the name feature convention for git branches, where each member of the group created a branch for each task they had been assigned to complete. We began our commits with either "feat" or "fix" to be clear what each commit was achieving. We created an instagram group chat to coordinate work between one another and be able to discuss our implementations of certain features remotely.

We trialled pair programming, with different configurations of group members as drivers/navigators, to gauge how best to pair program going forward. Our group was most productive whenever we were physically in the same room, allowing us to bounce ideas off of one another and instantly answer any questions we had for each other. However, most of the work was completed remotely, so for the later tasks we should organise more time where all four of us can collaborate in-person. On the whole, the group is collaborating smoothly and completing the tasks effectively.

2 Implementation Strategies

The state of the emulator is stored as a global struct with a single instance. It stores basal pointers to main memory and the registers, as well as the contents of the program counter and zero register. The PSTATE register is represented as a struct stored within state. Each of the flags within PSTATE are stored as a field of the struct. Instructions are represented as a union of structs, where each struct within the union represents a different type of instruction. This enabled us to concisely simulate the decode phase. The execute phase was largely implemented as helper functions, which were relevantly called from the decode phase after the instruction input had been parsed.

The representation of instructions as a union of structs was key to our implementation of the emulator, and therefore we should attempt to reuse as much of this structure as possible in the assembler (since the assembler will also require an instruction set). The decode logic of the emulator is also liekly be applicable to the assembler, although perhaps not as directly.

Implementing part three is likely to be challenging, as it requires use of the Raspberry Pi device, which we will not have tackled in the first two parts. To mitigate this, it is likely to be worth assigning one member of the group to begin this task while the other three work on the previous task.