Project Description

DifficultWare, CSS 422 A

This document describes our project and the approach we took, the design we chose, our implementation, and the external sources we used.

Approach

To understand how we were to disassemble each of the instructions from the list we were given, we created a tree that branched based on the bits that made up the instruction. This way we were able to group the instructions into similar categories based on their first four bits.

From each group, we then analyzed all known bits to determine exactly what the instruction was. When bit masking the opcodes, we made sure that we check the opcodes with more known bits first to avoid branching early into a wrong function.

Once this was known, we created at least one method for each individual instructions that handled all possible representations of its bits. For most of the methods, we used several helper methods that help to determine sizes, print immediate values, or increment addresses by the correct amount.

After this was all implemented in code, we wrote a test file that contains all possible scenarios for each instruction to make sure we were able to disassemble each one.

Design

This section includes the flowcharts we used when designing our disassembler.

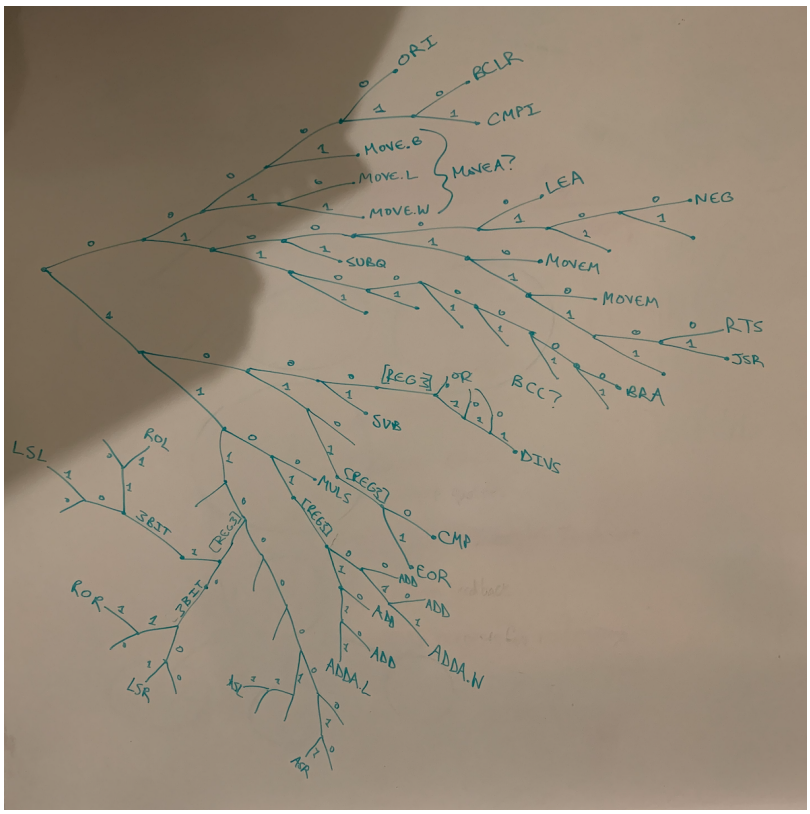

Figure 1: Our Initial Instruction Tree

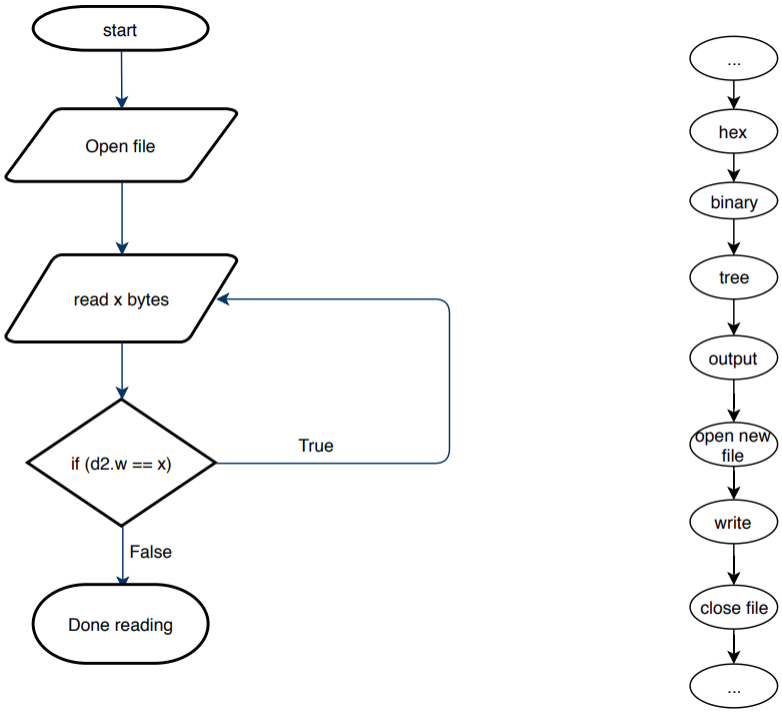

Figure 2: High-Level Flow Chart for Reading Memory and Disassembling

Implementation

Some sections of our project were particularly difficult. This section mentions those difficulties.

We implemented a method for each supported instruction using the memory directly given before we decided to go back and correctly implement the memory address displayed by BRA. Since Bcc uses a displacement instead of an actual address like most other instructions, we did not have a method to handle it easily. To properly obtain the address of the branch that is specified instead of the displacement to it, we obtained the displacement and added it to the current address plus an extra Byte (Bcc displacement gives us the distance between the current address and the branch, excluding a Byte to point to the branch itself).

We had another issue when a negative displacement was given for Bcc. We solved that by using EXT and NEG to obtain positive displacement and using SUB to obtain a branch before the current address.

Our next challenge was figuring out how to handle the opcode MOVEM. Nothing particular about this method was hard, aside from the fact that the EA for it is very unique, which required a lot of code that couldn't be shared with other instructions.

Lastly, we encountered some difficulties when dealing with EA compared to the IO and OP codes. This could be attributed to the fact that each instruction is unique and poses its own issues. Even though we ran into these troubles, we were able to work as a team to discover solutions that worked.