Program Specification

DifficultWare, CSS 422 A

This document contains the manual for our disassemble as well as a description of our team's code standards.

Disassembler Manual

Our disassembler will read a starting and ending address from a file in the same directory titled 'Config.cfg', then read the data between those addresses, and disassemble it into to the output window and a file titled 'Output.txt'.

To use our disassembler, we first need to open it in Easy68k. This can be done from within Easy68k and then navigating to 'File' > 'Open File …', finding our file titled 'Main.X68', and clicking Open. Now that our disassembler is loaded, we can execute it by pressing Easy68k's 'Assemble Source' button and then clicking 'Execute' in the window that pops up. The next window that appears will display the list file generated from our project's source code. Before we run it, navigate to 'File' > 'Open Data…' and locate the file you want to disassemble. By clicking 'Open' that file will be loaded into memory (note that this file must be an '.s68' file). Now is a good time to make sure that a file titled 'Config.cfg' is in the same directory as 'Main.X68'. This configuration file needs to be in the format:

<Long_StartAddress>\CR\LF
<Long_EndAddress>\CR\LF<EOF>

Finally, clicking 'Run' will start the disassembling process and print the assembly code to the output window as well as the file 'Output.txt'.

Code Standards

As a team, we agreed on some standards to make our implementation process go smoothly.

For commenting, we placed a header with comments that included the method name, description, preconditions, and postconditions above each method. Also, on almost every line we wrote, we wrote a comment describing its action at the end.

To reduce redundancy, we used a few methods that carried-out common operations such as TrapTask13, AsciiToHex, hexToAscii, findModeReg, and checkImmediate. This way our file size was lessened and our time saved.

We also created e a method for saving our place in memory so we agreed that after determining an instruction, we would return back to our main loop with the address register containing the value of the next instruction’s memory.

For this project we had a lot of data to store with only a limited number of registers available. To get around this, we utilized a callee-saved workflow so that each method had full access to every register.