Final Report

Vince Maranan

CSS 422

**Section 1 Specification.**

The disassembler program can take data and an input range of addresses to disassemble.

To run the program

1. Open easy68k
2. Open disassembler.x68
3. Click project
4. Assemble source
5. Execute
6. One the new windows pop up, click file
7. Open data
8. Open test\_disasm.s68
9. Click run -> Run
10. Enter start address: 400
11. Enter end address: 500
12. Press enter to read more data

Not all required opcodes are implemented but a lot of them are implemented. When the data is processed, it processes 20 at a time. Whenever an instruction is given but is not supported by the system, it displays “DATA $hex” with the $hex being the actual hex value that is calculated. The program is able to disassemble successfully showing the correct instruction and hex address that was provided.

**Section 2 implementation.**

In the program, A2 holds the message buffer that displays to the user.

To find opcodes we use a jump table, jmp\_table and jmp\_table1 and calculate the offset

Whenever we find an opcode, we start putting onto the message buffer like this:

MOVE.B #’M’, (A2)+

MOVE.B #’O’, (A2)+

MOVE.B #’V’, (A2)+

MOVE.B #’E’, (A2)+

And we can calculate the sizes of the instructions using putSizeBuffer and putting them in the buffer as well.

To calculate the address, we first grab the ascii keys and convert it to hex using asciiToHex and using that calculated value to display using A2.

To ask for display for the user, we use the SHOW function which leads into intro.

The fillbuffer subroutine counts the numbers of lines that are printed and allows the program to print 20 outputs at a time.

To calculate effective address mode, we use ea\_mode\_table and just like jmp\_table, we calculate the offset given with the instruction. We can add the calculated effective address modes by doing:

MOVE.B #’(‘ (A2)+

MOVE.B #’P’, (A2)+

MOVE.B #’C’, (A2)+

MOVE.B #’)’, (A2)+

If we find an opcode that we don’t support, we jump to the illegalCode subroutine, append “DATA” to the message buffer and then load the data into the buffer. So once we find the error, we can grab the hex data and then append it to the message buffer A2.

The handleinputs subroutine handles the inputs of the terminal however it does not check the address range very well.

**Section 3 Test Plans**

Throughout the entire development process, the program got tested incrementally. There were not much different types of test done aside from the provided Test\_Disasm.s68

To test the program:

1. Open easy68k
2. Open disassembler.x68
3. Click project
4. Assemble source
5. Execute
6. One the new windows pop up, click file
7. Open data
8. Open test\_disasm.s68
9. Click run -> Run
10. Enter start address: 400
11. Enter end address: 500
12. Press enter to read more data

The program does not have a lot of safes. For example, the program does not check for valid starting and ending address ranges.

**Section 4 Challenges**.

The program cannot calculate valid start and ending addresses. The program is lacking some of the required opcodes no non-required opcodes were implemented. Another challenge I faced was implementing more tests especially for the effective address modes. Currently it is unknown how well those subroutines work.

**Section 5 Team Assignments.**

I (Vince Maranan) did all of the work for this project. It took a lot of long hours and all-nighters to get this project working and running up. I did both the UI and Effective address calculations.