**Disassembler Documentation**

*By: Inverse Assembler*

Zealous Zhu and Eric Feldman

Created: 11/25/18

Updated: 12/07/18

**INTRODUCTION**

Welcome! We are the Inverse Assembler team. This disassembler document outlines all the specification and documentation of our project for CSS 422: Hardware class. This will also highlight all the design and procedures to create this project, including status reports, constraints, limitation, IO, testing, and team member’s objective and their respective role.

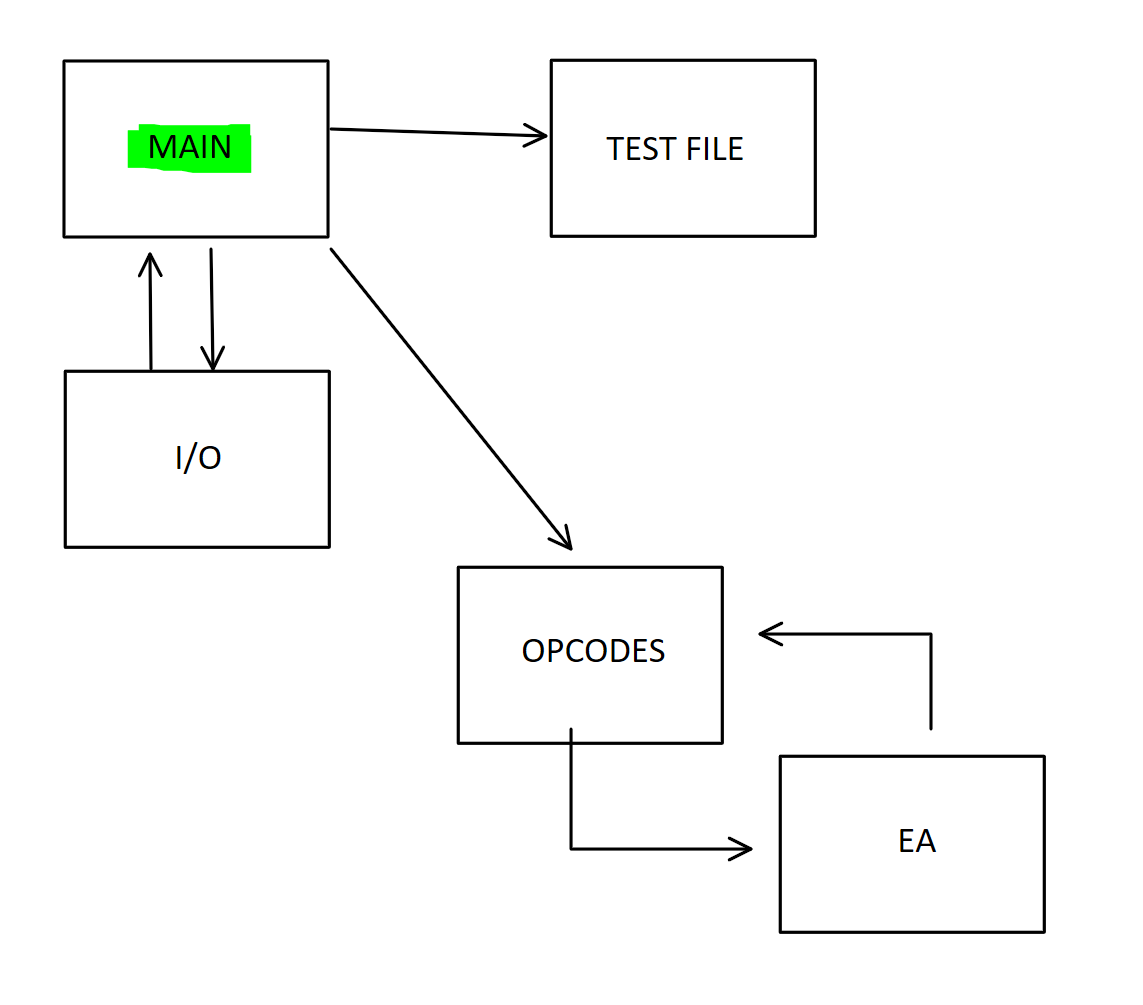
The Disassembler project is a low-level program written in 68k. The purpose of the project is to disassemble all the 68k code and convert them into a complete set of instructions within the memory. The user can input a starting and ending address (in hexadecimal) to scan that specific section of the memory to print them in console.

The requirements and specification of the disassembler is provided to us in class and through the project resources on UW canvas. We then took those requirements and break it down further to finally design and implemented it. Throughout the project, we also created test cases that will be used and added for testing to determine what is working/not working.

**DESIGN**

For our project design, we followed the principles of Object-Oriented Programming. We treat each of the individual files as classes and we have a main that utilizes all of those files. This provides reusability and a more organized structure. This is important because the disassembler is a very long program in an old IDE that has no auto-format or intelligent debugger like the high-level programming languages. So, this can create issues of mixing up between different functions and it is very difficult to identify which part of the code is causing the problem (IO, Op-codes, EA, etc.).

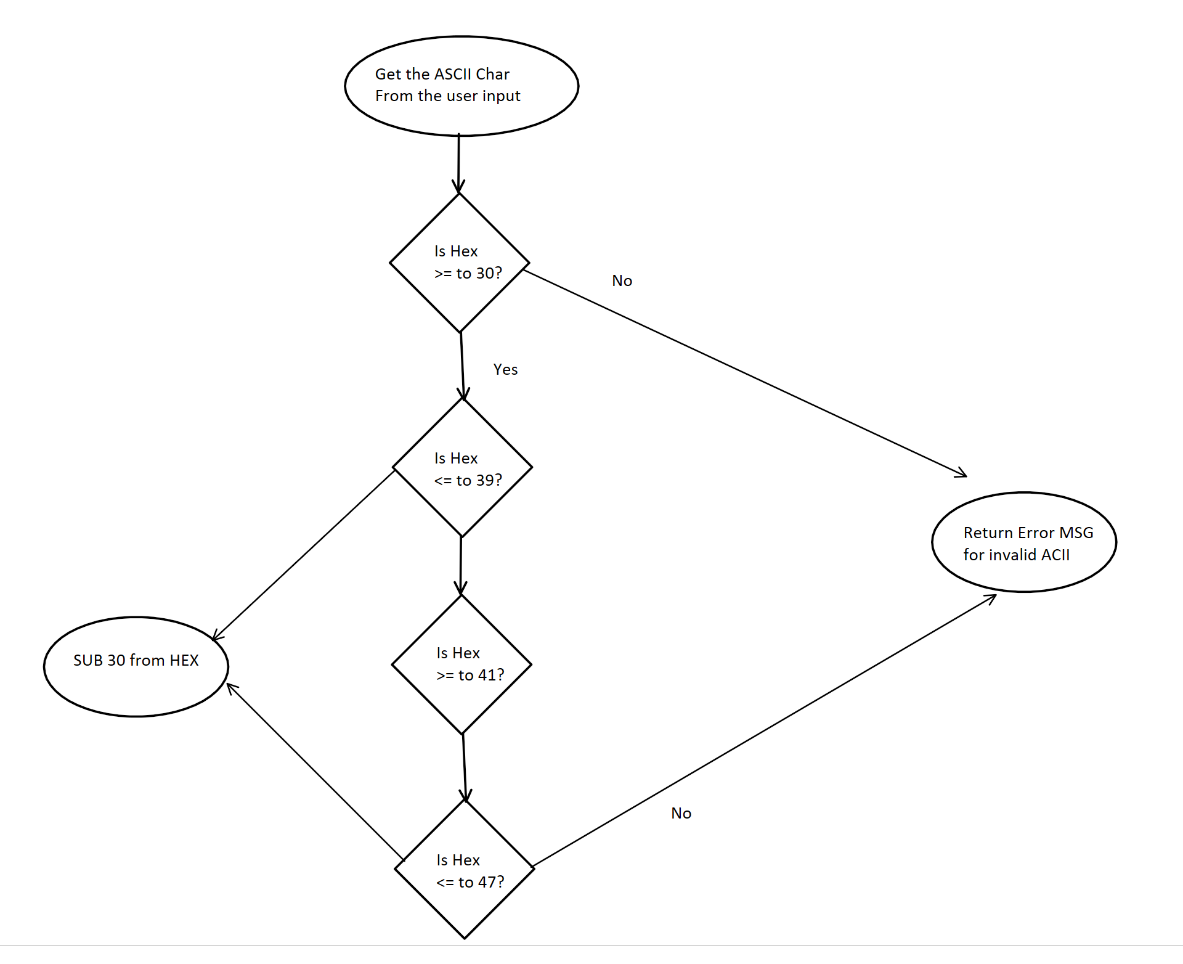
As a result, our file structure is organized in the follow below:



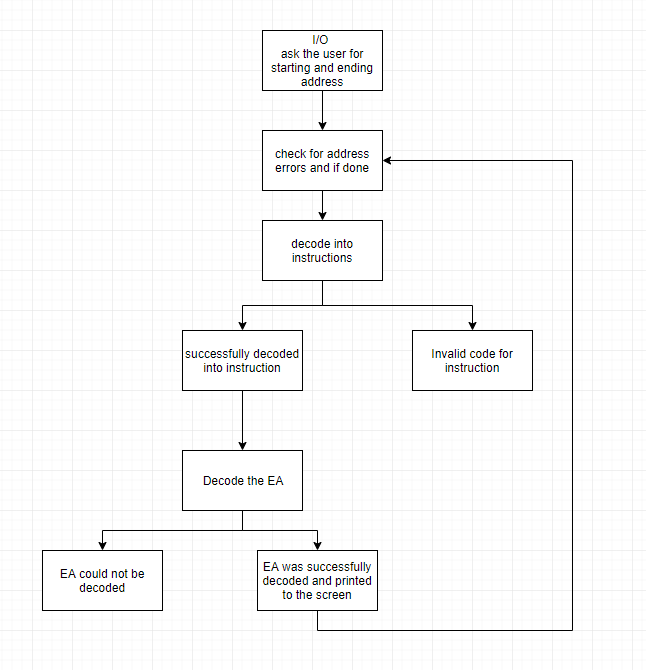
* The MAIN file is basically the control center that defines all the variables and data. Then it can call each of those individual files in any orders we want (I/O is first). It also includes all the files so that they are inter-connected.
* The next file is the INPUT/OUTPUT (IO) file that interacts with the user by asking them for starting and ending addresses to place in memory to decode. Here in the IO, it will also display the instructions to the user and check for any wrong inputs. If the user puts in the correct addresses, it will store the starting address in A4 and ending address in A5.
* The OP-CODES file is where the decoding for all the assembly instructions lives. It will also call the EA file for any instructions that requires EA to function properly. We have all the required opcodes that are supported, but also a list of unsupported opcodes (not required). This will be discussed further down the document.
* The Effective Address (EA) file is essentially the determining factors for the source and destination of an instruction move set. The EA can determine if the source/destination is a direct, indirect, absolute, or immediate values. It will also handle all the different cases here.
* Finally, we have the testing files. The testing files is a long list of instructions that we use to test if it will decode those instructions correctly onto the console. This file is helpful to determining what is working and what is not.

Overall, we have 5 files in total that are independent of each other with their own functions, but they also work together perfectly through the correct orders.

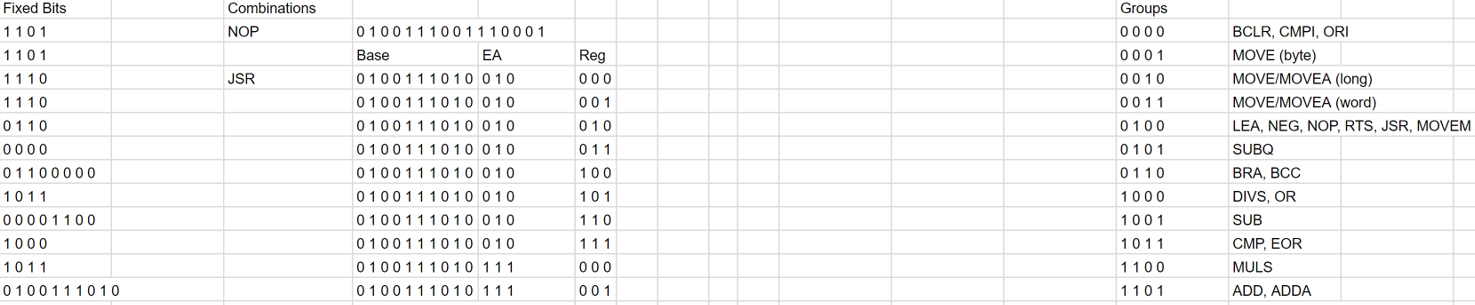
The followings are flow charts that we created during our early planning and design:

* This is a simple ASCII to HEX conversion flow chart.

This is the early design flowchart for the disassembler program.



We also started a google sheets that organizes the opcodes and group the ones that have the same codes, mode, or register.



In order to facilitate collaboration and work, we used Github as our version control and file management.



* Note: We included the README.md file which contains our project overview for the disassembler program

**DIFFICULTIES**

Throughout the project, we stumble upon some difficulties in terms of the program itself and ourselves as a team. First, we will outline the team difficulties and then the program.

TEAM:

We are a group of 2 individuals working on this project, which means we must put in much more work and efforts as oppose to those with a group of 3. We also had problems deciding which approach to take during the beginning of the project. We each wanted to do our own ways, which soon leads to some conflicts and time wasting when combing our codes. For example, Zealous wants to do this project using a Jump Table structure in a single file. Whereas Eric wants to do this using the opcodes as the main source for calling and separating the different files into IO, OPCODES, and EA.

While we didn’t really “decide” on it, we ultimately went with Eric’s approach as the project progressed when we made our own individual files. This has created many bugs and problems with compatibility issues when trying to combine our codes. Zealous was working on the EA file in a specific way and Eric was working on the opcodes in a specific way. This is the result of a small communication problem.

Resolution:

Despite the different approach and problems, we have in the project, we were able to complete this project for a several reasons:

* Although we miscommunicate in terms of the project approach, we established a good communication channel and always communicating when to work on it.
* Meeting in person more often has helped both of us solve those conflicts and help each other understand the direction of the project better.
* Debugging and tracing through the program together also helped each other understand what we are trying to do.

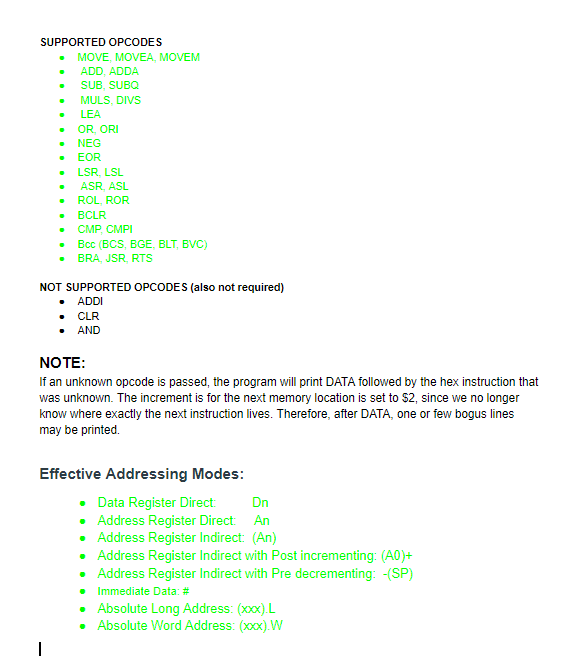
PROJECT:

* We fully support all the opcodes that are required in the canvas requirement page.
  + However, one problem is that sometimes the MOVEM only prints the size itself, and a place holder for list and EA. This might create a small problem that will affect other part of the instructions.
* OPCODES that are not supported are (however, not required):
  + ADDI
  + CLR
  + AND
  + Etc.

**Testing**

Our goal is to make sure the program does everything it should and that it follows the guidelines from the project specification. Throughout the project, we were testing the individual opcodes/EA as we are implementing them. This provides a quick fix if we found a problem with it.

We did not do logging as part of our test; however, we did perform tests in several ways:

* We used the 68k output window to view the output while testing.
* We also place the testing file into different location of the memory to test.
* We were also doing black box testing (I/O) to see if it produces the correct output given our input.
* We also tested memory incrementation loop by testing instructions of various sizes in different orders.
* As part of debugging, we used the Easy68k editor’s trace feature to watch the memory as the compiler executed each line.
* We were also writing down and checking off the individual opcodes/EA on the google doc file:

**Team Responsibilities**

Overall, we worked, tested, and fixed on all the different files despite the specific task.

This was the general responsibility.

|  |  |  |
| --- | --- | --- |
| **Tasks** | **Member Names** | **Description** |
| IO | Zealous | Ask user input, checks for all types of wrong inputs |
| OP-CODES | Eric | Decode instructions and call different EA |
| EFFECTIVE ADDRESS | Zealous | Different EA types depending on the modes/registers |
| MAIN | Eric | The main contains the buffer loop and calls the other files |
| TEST FILE | Eric | Create a test file to test opcodes |
| DOCUMENTATION | Zealous | Work on this documentation and most comments in codes |

The table above highlights the general team responsibility for this overall project. However, due to the nature of the program, Eric has also spent most of the time debugging and trying to combine our files together to fit with the way the opcodes are structured. As a result, he is the team leader in this group since everything is based on his design.

**Overall Individual work**

Despite the different approaches and small conflicts, we ultimately finished the project with all the efforts and hard work we put in this last week. However, I believe Eric has put in more time to try and fix my part of the code so that it can be compatible with his. For this reason, I think he has done a bit more work than I did.

Eric – 55%

Zealous – 45%

Zealous Zhu

12/7/18