ECE 441 Final Project Report

Spring 2016

By: Daniel H. Meves

Instructor: Dr. Jafar Saniie

TA: Boyang Wang

ECE 441-001

Due Date: 04-27-2016

Acknowledgment: I acknowledge all of the work (including figures, codes and writings) belongs to me.

Signature : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table of Contents**

**Abstract**

Resident monitor programs have been a part of computers since the early days of punch card computing and simple batch systems. They were simple versions of the modern operating system. The goal of this Design Project is to build up a Monitor program using the MC68000 assembly language. The monitor we designed is capable of performing basic debugging functions like memory display, memory sort, memory modify, block fill, block search, and block move. The monitor also can process exceptions through its exception handling routines.

**Implementation of Commands**

1. **Help**
2. **Memory Display (MDSP)**
3. **Sort Word (SORTW)**
4. **Memory Modify (MM)**
5. **Memory Set (MS)**
6. **Block Fill (BF)**
7. **Block Move (BMOV)**
8. **Block Test (BTST)**
9. **Block Search (BSCH)**
10. **Go (GO)**
11. **Display Formatted Register (DF)**
12. **Exit**
13. **Change/Modify Registers**
14. **Clear**
15. **Command History (HISTORY)**

**Quick Manual**

HELP: Displays this message

MDSP: Outputs Address and Memory Contents

MDSP <address1> <address2> eg: MDSP $908 $90A<CR>

SORTW: Sorts A Block of Memory

Default: Descending Order

A: Ascending Order'

D: Descending Order

SORTW <address1> <address2> [order] eg: SORTW $904 $90E A<CR>

MM: Modifies Data in Memory

Default: Displays One Byte

W: Displays One Word

L: Displays One Long Word

MM <address> [size] eg: MM $904 W<CR>

Exit MM command by typing "."<CR>

MS: Memory Set

MS <address> [data] eg: MS $904 $FFFF<CR>

BF: Fills A Block of Memory

BF <address1> <address2> <word> eg: BF $904 $908 475A<CR>

BMOV: Moves A Block of Memory to Another Area

BMOV <address1> <address2> <address3> eg: BMOV $908 $90B $909<CR>

BTST: Block Test

BTST <address1> <address2> eg: BTST $900 $90A<CR>

BSCH: Searches A Literal String In The Memory

BSCH <address1> <address2> "literal string" eg: BSCH $900 $910 "MATCH"<CR>

GO: Starts Execution from Given Address

GO <address> eg: GO $900<CR>

DF: Display Formatted Registers eg: DF<CR>

EXIT: Exit the monitor program eg: EXIT<CR>

Modify Registers: Modify the contents of each register individually

.D[0-7] [data] / .A[0-7] [data] / .SR [data] eg: .D4 $ABCD<CR>

HISTORY: Display Command History

N: Number of commands to display

HISTORY with no arguments prints all history

HISTORY [N] eg: HISTORY 10<CR>

CLEAR: Clear text from terminal eg: CLEAR<CR>

**Engineering and Design Challenges**

Writing an assembly program of this magnitude was quite challenging for a novice programmer such as myself. One major challenge was code modularization. In an effort to reuse as much code as possible, I tried to encapsulate frequently used blocks of code into functions/subroutines. This was easy for obvious functions such as ASCII to Hexadecimal and Hexadecimal to ASCII conversion. However, I failed to find an efficient way to print the help messages for each command when only the command name is typed. I also had issues in developing a uniform way of pointing address registers to command arguments. These two examples are cases in which I needed to custom tailor the code for each situation/command subroutine.

Another challenge was the length of the entire program and its effect on the behavior of the Easy68k simulator/assembler software. As my monitor program grew in length, I noticed slow down and lag in both the text editor/assembler and the simulator program. One way in which I counteracted this negative effect was to divide my code into separate files based on subroutine. I then made a single header file with all the INCLUDE statements for the individual files. It is from this file that you must execute the program.

**Expansion of the Existing Monitor Program**

If given more time to work on the project, I would like to add additional commands to the Monitor program. I originally wanted to incorporate the up and down arrow keys into the command history command. This would have been implemented using trap task 19 of the trap 15 I/O commands. The problem I ran into when implementing this feature was the inability to accept input on the command line in the main program while also checking for up and down arrow key presses. Given the single threaded nature of the 68k simulator, only one instruction can be executed at a time. Therefore, it may not be feasible to allow scrolling through the command history with the arrow keys unless we were developing for a multi core microprocessor (the M68000 is not one unfortunately).  
 It would also be interesting to rewrite the monitor program to use TRAP #14 on the SANPER-1 ELU hardware. This would allow us to implement commands that communicate with external devices via the PIA and ACIA interfaces.

Because resident monitor programs are primitive versions of modern operating systems, developing a full-fledged OS for an embedded system using the MC68000 would mean building up a program similar to the one we built in this design project. An example of a more advanced implementation of this monitor program that would help in building an OS is the addition of networking. We may be able to achieve a primitive form of networking using the ACIA interface. We could also design a new board to insert into the SANPER unit that features a RJ45 port. This would also require that we write a driver/handler for this networking card.

In order to save time in the development of an embedded system operating system, it might be beneficial to rewrite this monitor program and associated drivers in a higher level language such as C. This comes with added benefit of code portability. However, some features such as our custom exception vector handlers may still have to be written in assembly. It may not be possible to modify the exception vector table in a higher level language such as C.