ECE 441

Microprocessors

Instructor: Dr. Jafar Saniie

Teaching Assistant: Guojun Yang

Final Project Report:

**MONITOR PROJECT**

04/24/17

By: Javier Sorribes

Acknowledgment: I acknowledge all of the work including figures and codes are belongs to me and/or persons who are referenced.

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

Table of Contents

Abstract 2

1-) Introduction 2

2-) Monitor Program 3

2.1-) Command Interpreter 4

2.1.1-) Algorithm and Flowchart 5

2.1.2-) 68000 Assembly Code 5

2.2-) Debugger Commands 6

2.2.1-) Debugger Command #1: HELP 6

2.2.2-) Debugger Command #2: MDSP 7

2.2.3-) Debugger Command #3: SORTW 7

2.2.4-) Debugger Command #4: MM 8

2.2.5-) Debugger Command #5: MS 8

2.2.6-) Debugger Command #6: BF 9

2.2.7-) Debugger Command #7: BMOV 10

2.2.8-) Debugger Command #8: BTST 10

2.2.9-) Debugger Command #9: BSCH 11

2.2.10-) Debugger Command #10: GO 11

2.2.11-) Debugger Command #11: DF 12

2.2.12-) Debugger Command #12: EXIT 12

2.2.12-) Debugger Command #13: BPRINT 12

2.2.12-) Debugger Command #14: CONV 12

2.3-)Exception Handlers 13

2.3.1-) Bus Error Exception 13

2.3.2-) Address Error Exception 14

2.3.3-) Illegal Instruction Exception 14

2.3.4-) Privilege Violation Exception 14

2.3.5-) Divide by Zero Exception 15

2.3.6-) Line A and Line F Emulators 15

2.4-)Quick User Instruction Manual 16

3-) Discussion 17

4-) Feature Suggestions 18

6-) Conclusion 19

7-) References 20

8-) Appendix 21

Abstract

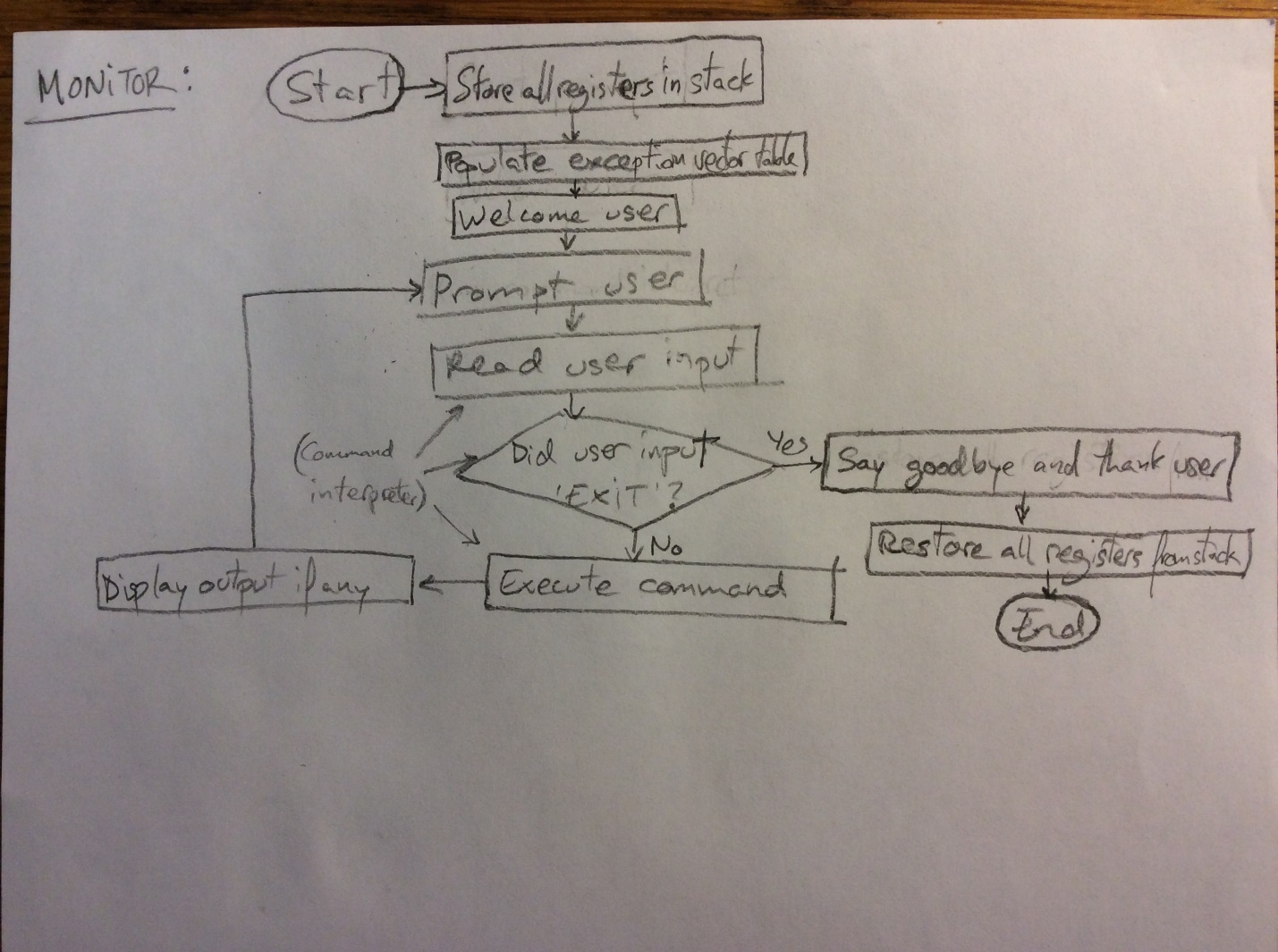
The summary of your design should go here. Someone who reads this abstract should have a clear understanding of your design and the overall flow of the report.

# *1-) Introduction*

This will be an introduction to your design. You can give design objectives, a clear description of the problem and design methodology and technology used. Any figures and tables should have clear descriptions.

***2-) Monitor Program***

This program allows the user to enter an executable command into the console, sometimes providing the appropriate arguments. Then, the command is run, the output (if any) displayed. Finally, the prompt will be redisplayed and the process will start over. The user may run the ‘EXIT’ command to terminate the program. The following flowchart represents this process:



*Figure 2.1. Monitor Program Flowchart*

Descriptions of all the available commands, also named debugger commands for their hardware debugging capabilities, are outlined in the following sections.

In addition, this program accounts for asynchronous exceptions, providing exception handling routines. Their descriptions can also be found in the sections after those dedicated to the debugger commands.

Finally, note that many of the subroutines outlined in this report use other helper subroutines. These are not explicitly explained in this report, but refer to the Appendix for the code in these subroutines, which are simple enough to be understood from the assembly code and comments.

***2.1-) Command Interpreter***

The command interpreter compares the first word of the input against a table with all command names. These command names are preceded with a digit determining their length, which is used by the algorithm to know how to advance to the next row, or name. Each command name is also followed by either a null or space character, depending on whether the command takes arguments or not.

Once the command interpreter finds the command name in the table, it uses the offset within that table to access the correct memory location of the executable in a command location table. If the name is not found, then an invalid message is displayed and the program prompts again.

Note that the command interpreter does not parse the arguments of the command, but rather leaves that task to each command. This design decision was taken because each command may require a variable number of commands in different formats.

***2.1.1-) Algorithm and Flowchart***

An algorithm of the design and its flowchart are displayed below:

*COMMAND INTERPRETER*

*While input != ‘EXIT’*

*Print prompt*

*Read input into the stack*

*counter = 0*

*row 🡨 first row in command names table // row is name with length preceding*

*While row < last row in table*

*counter2 🡨 length of name from row*

*While counter2 > 0*

*If next byte of input == next byte in row // keep comparing*

*counter2 = counter2 – 1*

*Else // name is different from input*

*counter = counter + 1*

*row 🡨 next row*

*Break while loop*

*End while*

*If counter > # command names // name not in table*

*Print invalid message*

*Break while loop*

*Else if counter = 0 // name was found*

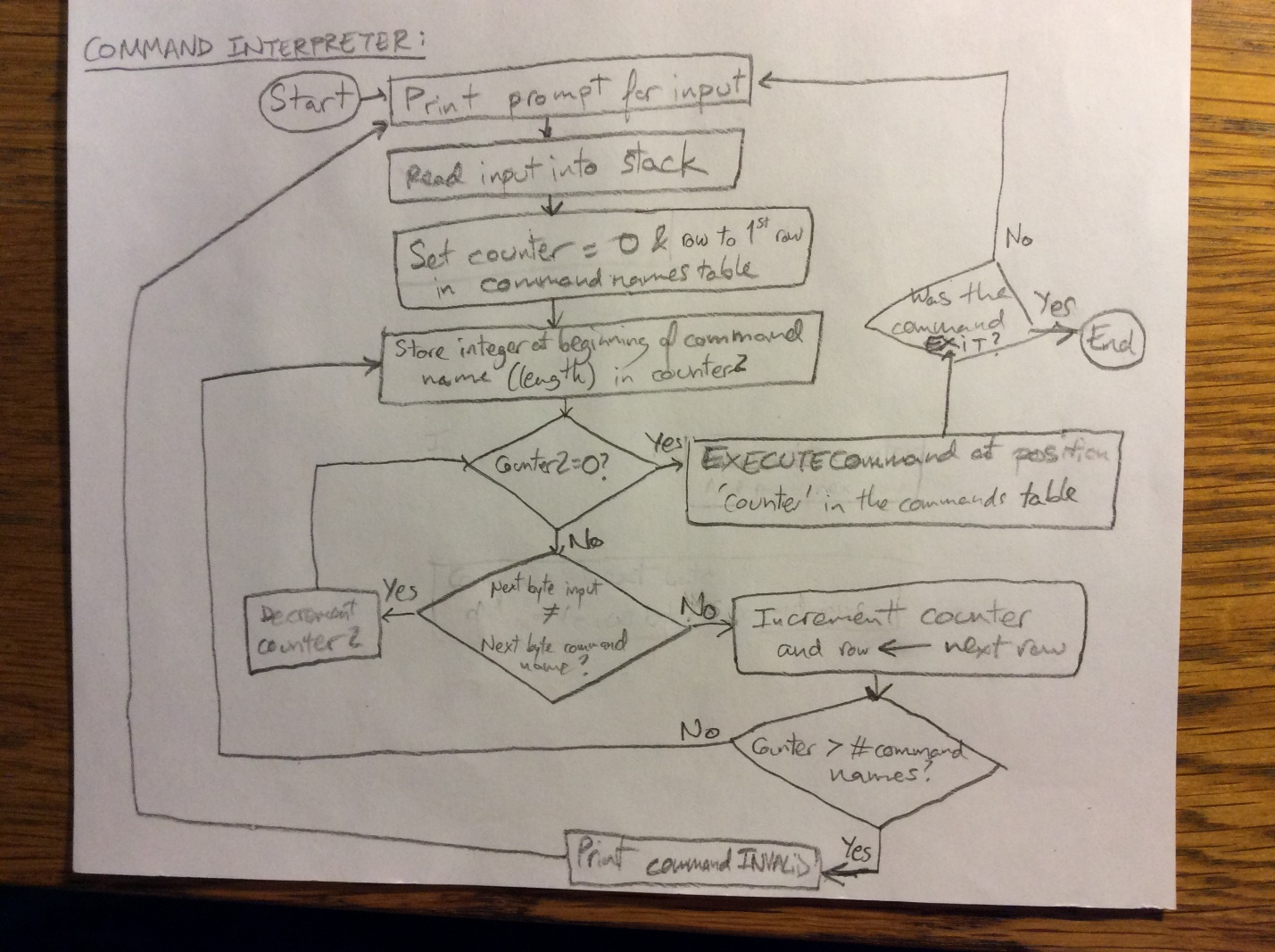
*Execute command at offset counter from command addresses table*

*End while*

*End while*

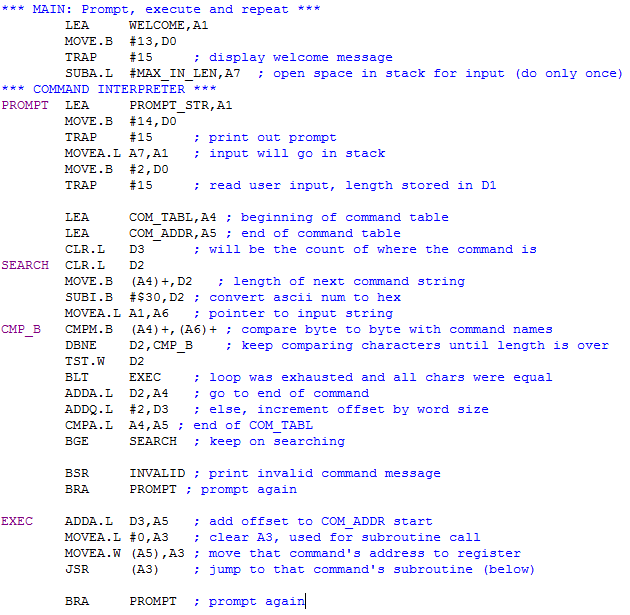
*Finish // finish*

*Figure 2.2. Command Interpreter Algorithm*



*Figure 2.3. Command Interpreter Flowchart*

***2.1.2-) Command Interpreter Assembly Code***



*Figure 2.4. Main & Command Interpreter 68000 Assembly Code*

***2.2-) Debugger Commands***

A clear description of your design should be given here.

***2.2.1-) Debugger Command #1: HELP***

A clear description of this debugger command should be given here.

***2.2.1.1-) Debugger Command #1 Algorithm and Flowchart***

An algorithm of the design and its flowchart will be explained here. You may need to include comments for your algorithm.

*Clear //this where things starts*

*Do this m=0 // assign m*

*While m > n // while m > n*

*If m > n //*

*Do this //*

*Else //*

*Do these more //*

*End if //*

*m = m + 1 // increment m by 1*

*finish // finish*

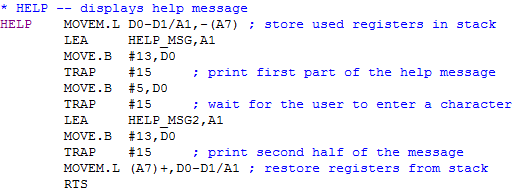
*Figure 2.5. Debugger Command # 1 Algorithm*

It may be necessary to explain more about your flowchart and your design ideas.

***2.1-)***

*Figure 2.6. Debugger Command # 1 Flowchart*

***2.2.1.2-) Debugger Command #1 Assembly Code***



*Figure 2.7. Debugger Command #1 Assembly Code*

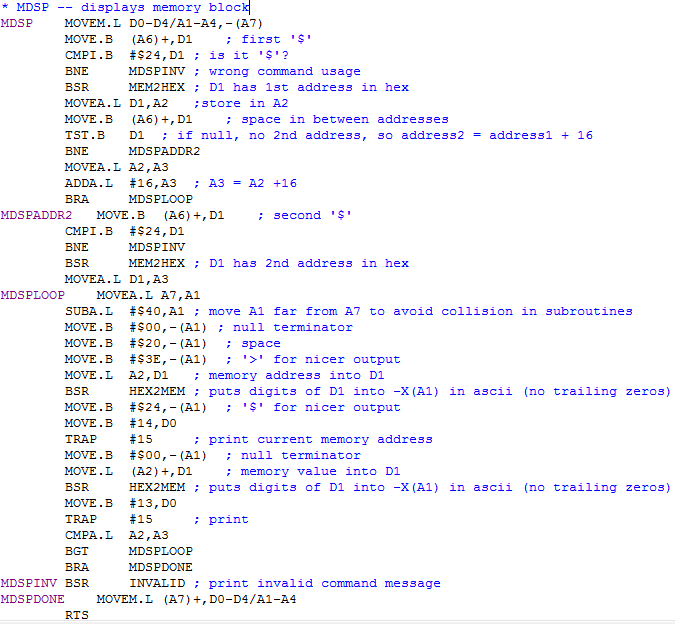
***2.2.2-) Debugger Command #2: MDSP***

It is similar to 2.2.1

***2.2.2.1-) Debugger Command #2 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.2.2-) Debugger Command #2 Assembly Code***



*Figure 2.10. Debugger Command #2 Assembly Code*

***2.2.3-) Debugger Command #3: SORTW***

Sorts a block of memory in between addresses 1 and 2 (inclusive) in either ascending or descending order. The command should be called in the form “SORTW <address1> <address2> A|D”, where A refers to ascending and D to descending (default).

The size of each number within the memory specified is expected to be word, and the type unsigned.

***2.2.3.1-) Debugger Command #3 Algorithm and Flowchart***

The algorithm for sorting is based on Bubble Sort, a method to “bubble up” items to their correct locations. By comparing numbers to the adjacent ones, we can decide whether to swap these or continue. Please refer to Lab Manual 2, Procedure 2.5 for more details.

In addition, a small check was implemented to be able to do either ascending or descending order as requested by the user.

*SORTW // first line*

*Parse input to get ‘start’, ‘end’ and ‘type’ (A or D)*

*While start < end // start will serve as an incrementing pointer*

*If start < start+1 and type = A // using start as an address pointer*

*Swap start with start+1 // so start+1 is the item after start*

*Reset start to original value (start over)*

*Else if start > start+1 and type = D*

*Swap start with start+1*

*Reset start to original value*

*Else // order is fine, move on to next*

*start = start + 1*

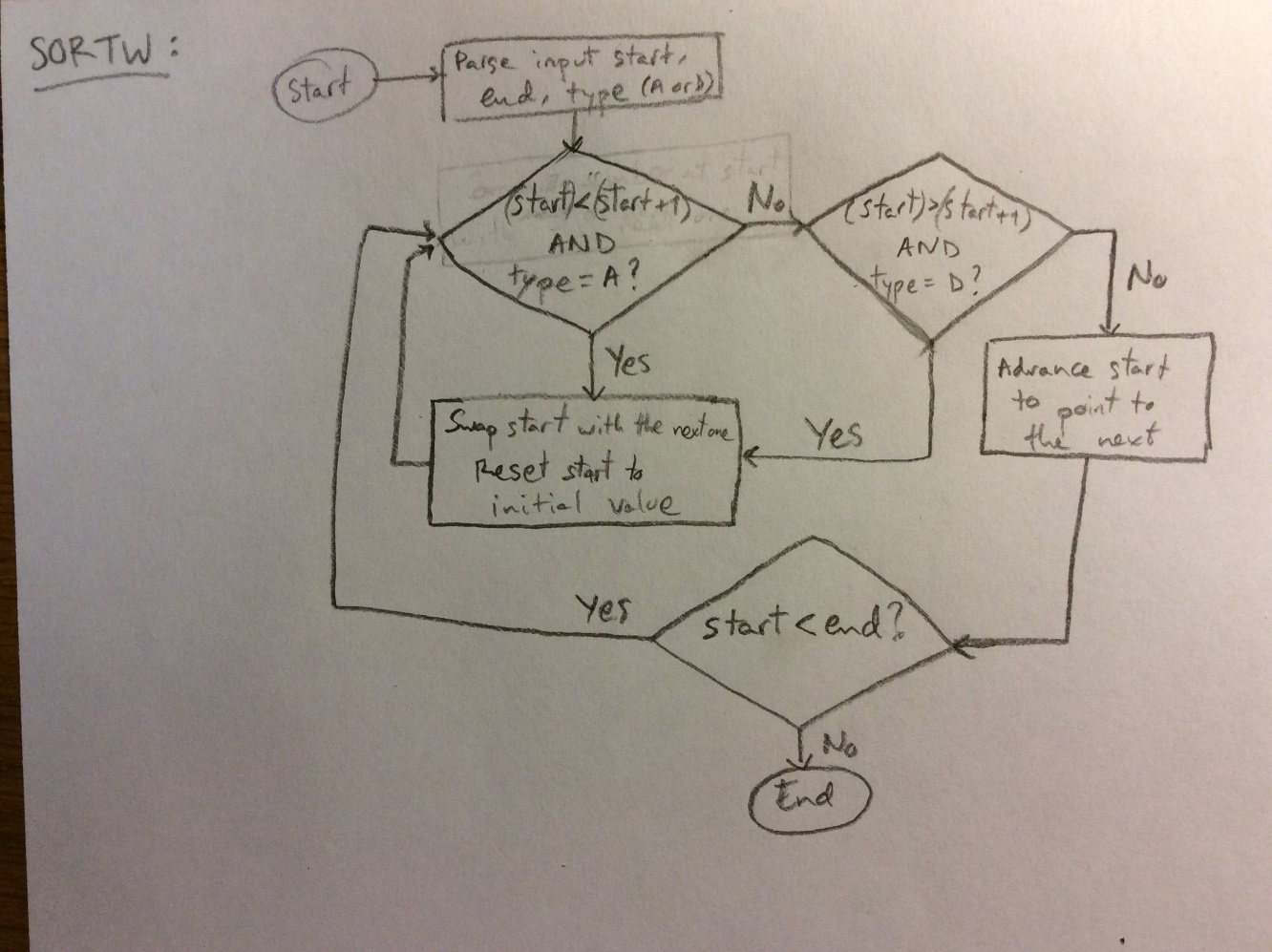
*End if*

*End while*

*Finish // finish*

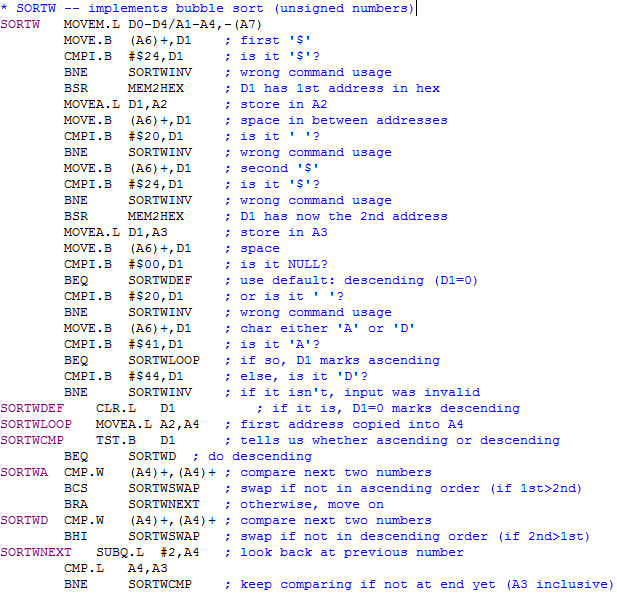
*Figure 2.X. Debugger Command #3 Algorithm*

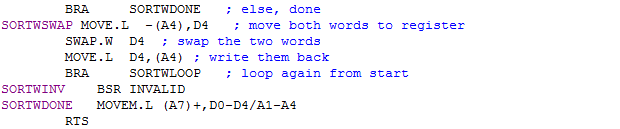
The following flowchart is an abstraction of the algorithm described above:



*Figure 2.X. Debugger Command #3 Flowchart*

***2.2.3.2-) Debugger Command #3 Assembly Code***



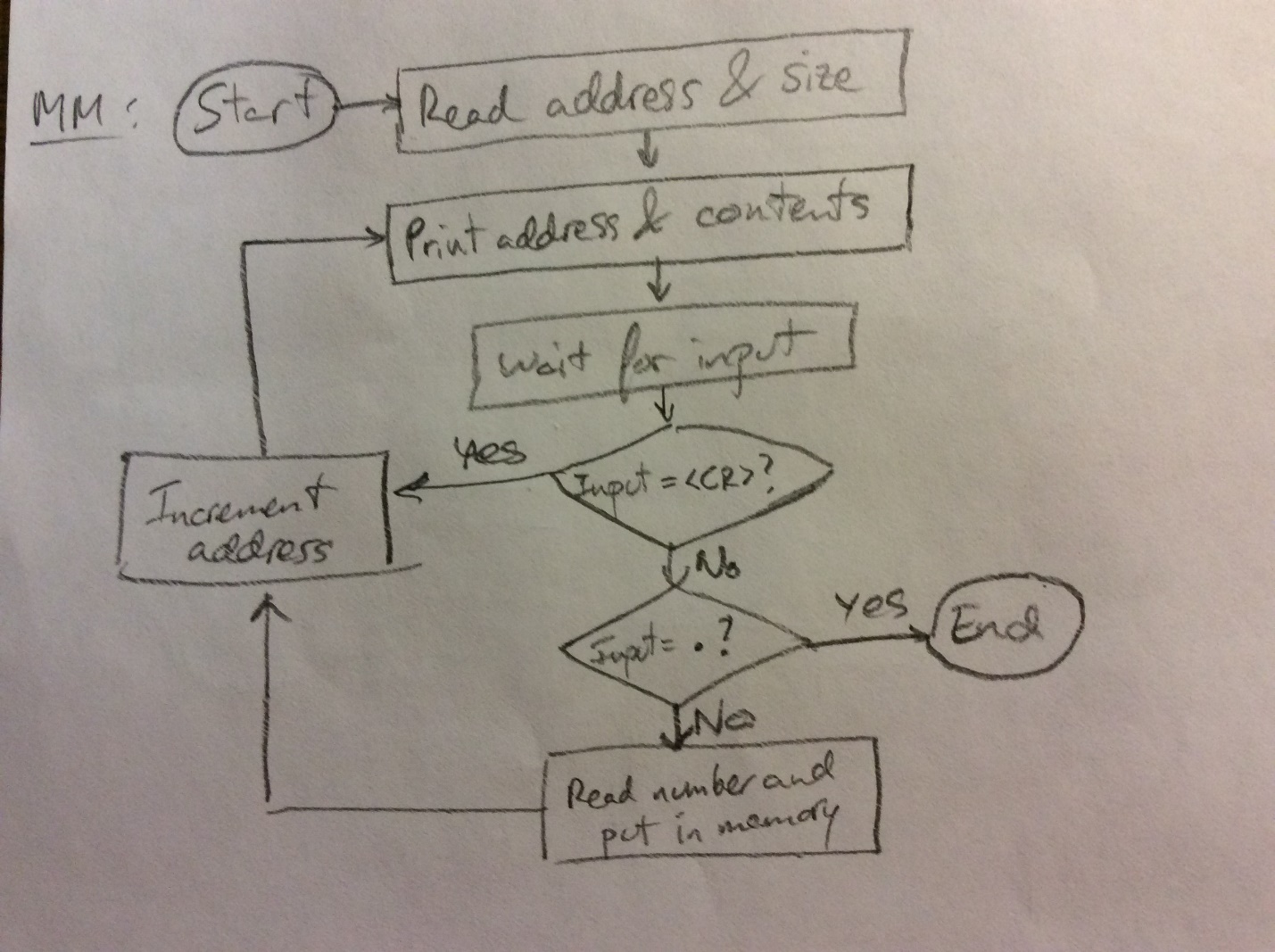


*Figure 2.13. Debugger Command #3 Assembly Code*

***2.2.4-) Debugger Command #4: MM***

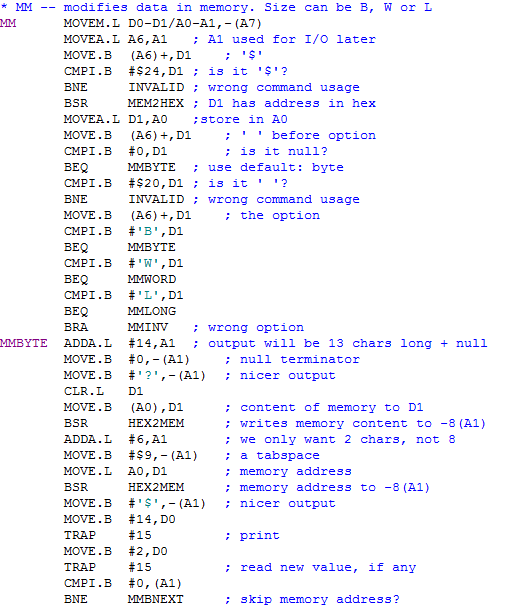
It is similar to 2.2.1

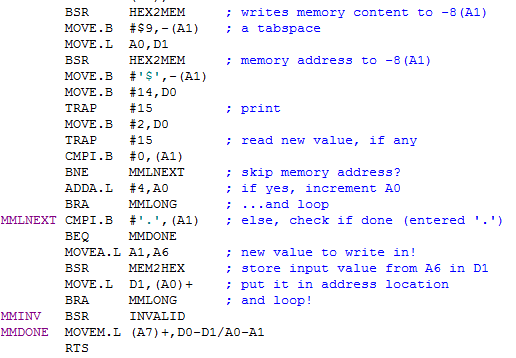
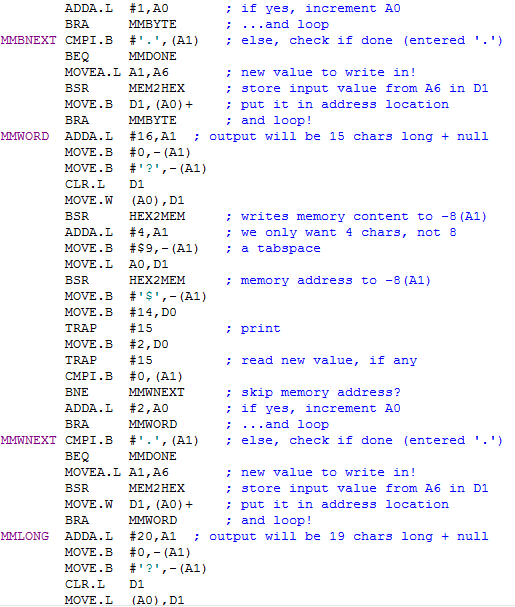
***2.2.4.1-) Debugger Command #4 Algorithm and Flowchart***



*Figure 2.X. Debugger Command #4 Flowchart*

***2.2.4.2-) Debugger Command #4 Assembly Code***





*Figure 2.13. Debugger Command #4 Assembly Code*

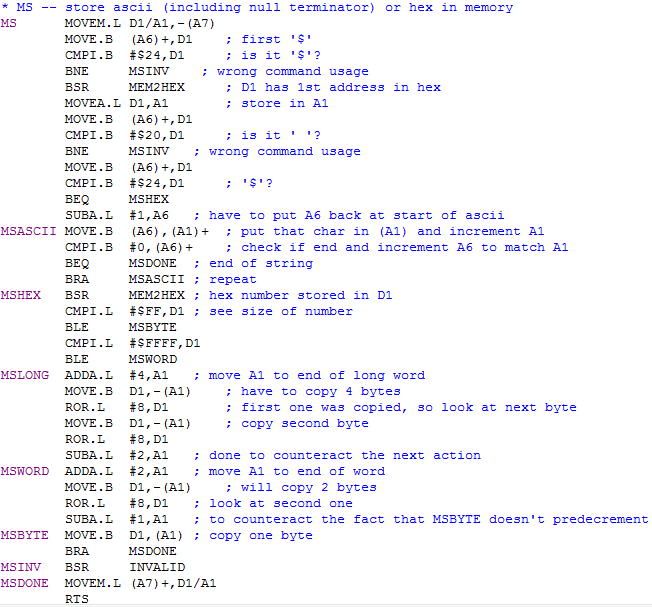
***2.2.5-) Debugger Command #5: MS***

It is similar to 2.2.1

***2.2.5.1-) Debugger Command #5 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.5.2-) Debugger Command #5 Assembly Code***

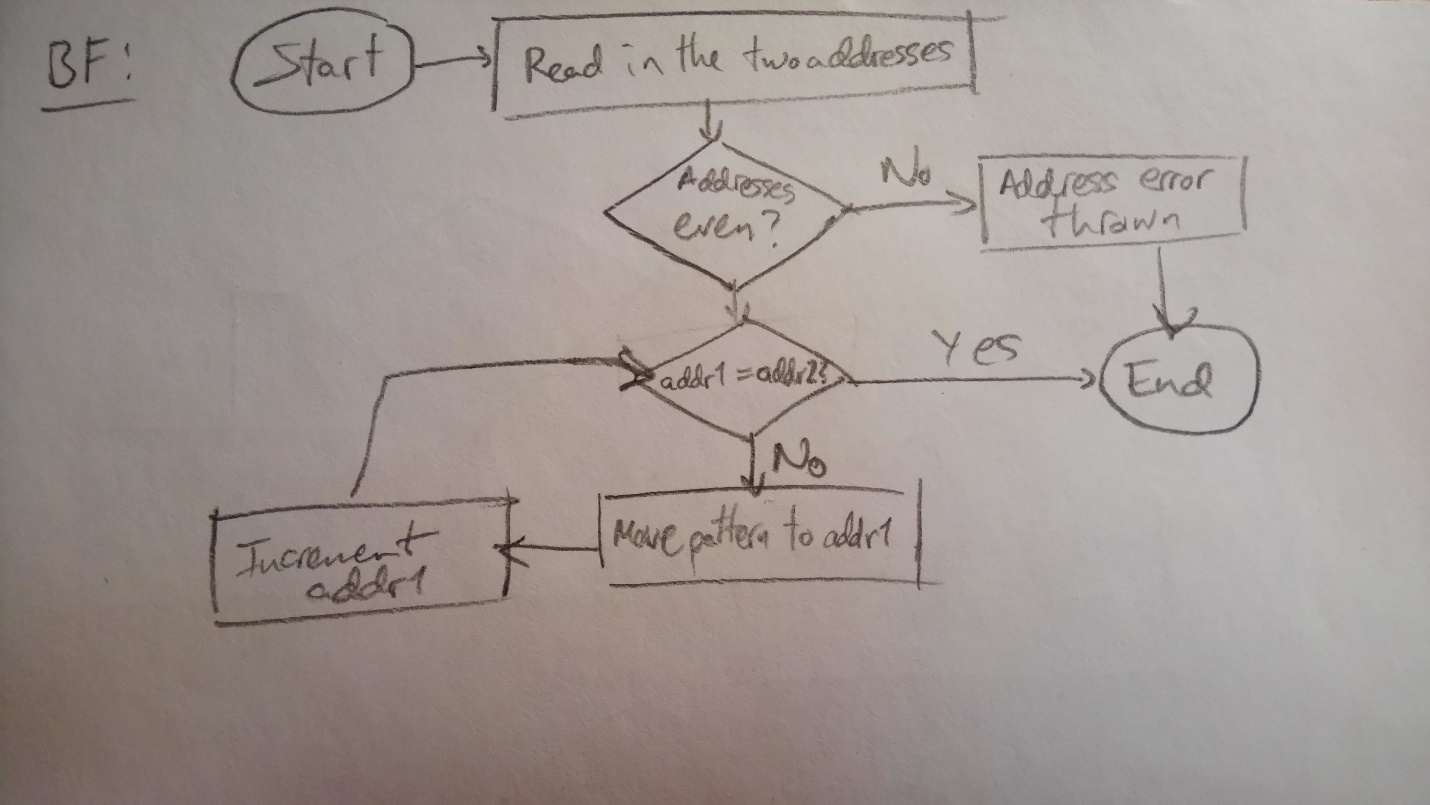


*Figure 2.13. Debugger Command #5 Assembly Code*

***2.2.6-) Debugger Command #6: BF***

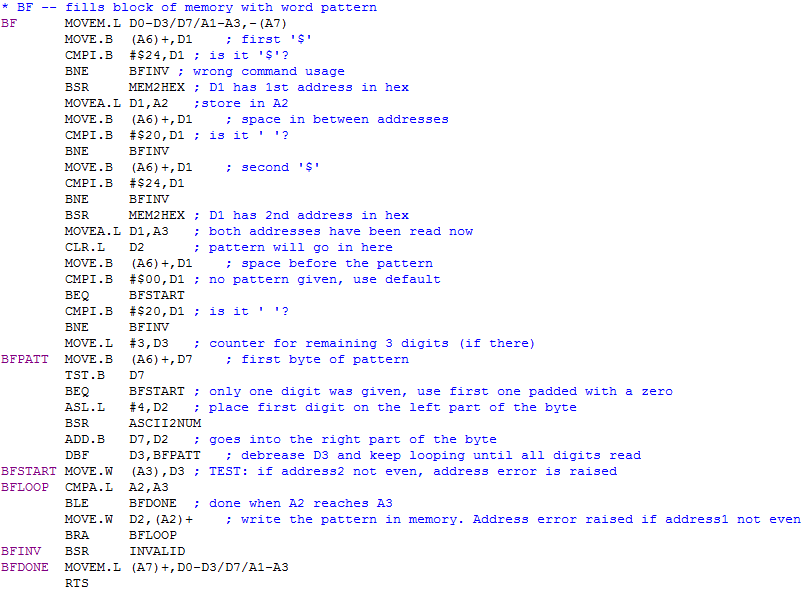
It is similar to 2.2.1

***2.2.6.1-) Debugger Command #6 Algorithm and Flowchart***



*Figure 2.13. Debugger Command #6 Flowchart*

***2.2.6.2-) Debugger Command #6 Assembly Code***



*Figure 2.13. Debugger Command #6 Assembly Code*

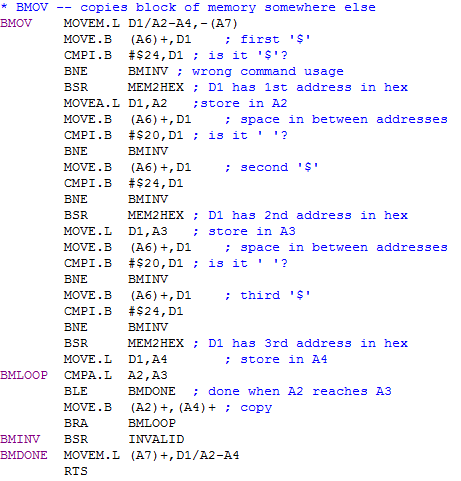
***2.2.7-) Debugger Command #7: BMOV***

It is similar to 2.2.1

***2.2.7.1-) Debugger Command #7 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.7.2-) Debugger Command #7 Assembly Code***



*Figure 2.13. Debugger Command #7 Assembly Code*

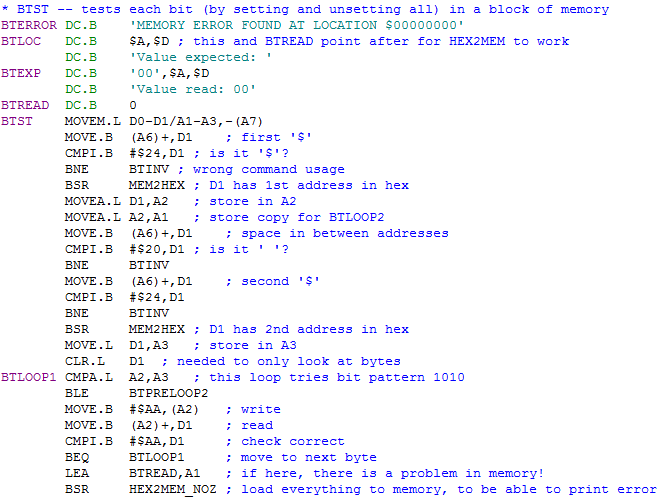
***2.2.8-) Debugger Command #8: BTST***

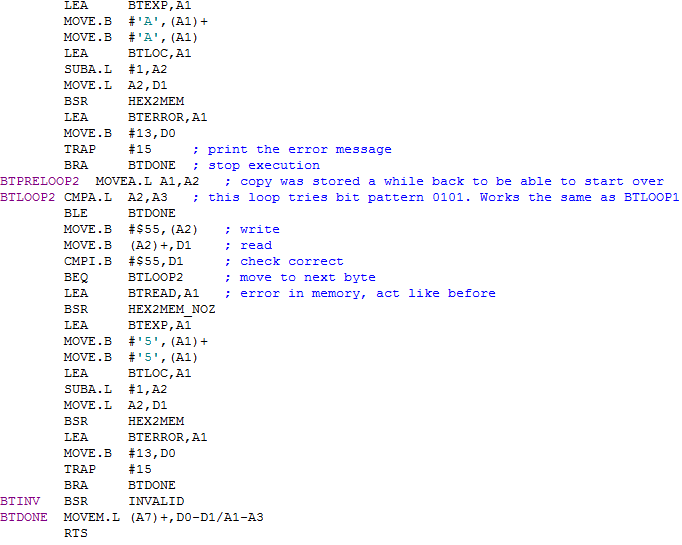
It is similar to 2.2.1

***2.2.8.1-) Debugger Command #8 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.8.2-) Debugger Command #8 Assembly Code***





*Figure 2.13. Debugger Command #8 Assembly Code*

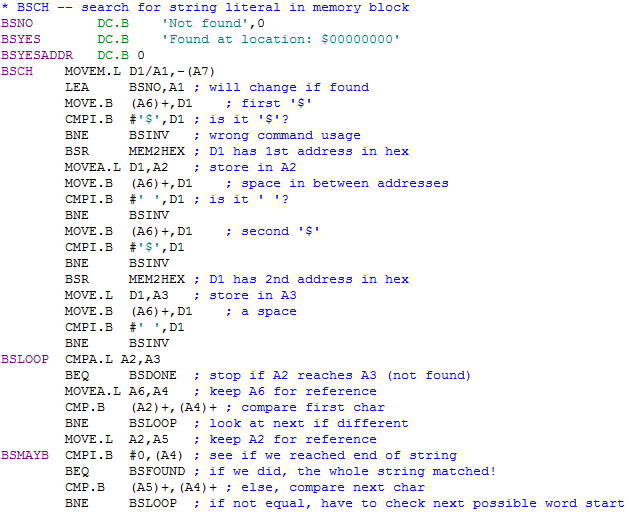
***2.2.9-) Debugger Command #9: BSCH***

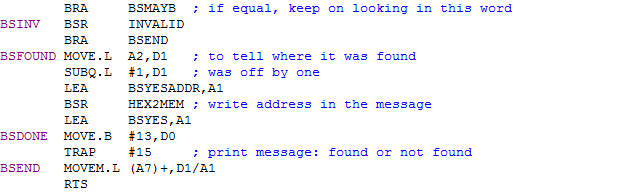
It is similar to 2.2.1

***2.2.9.1-) Debugger Command #9 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.9.2-) Debugger Command #9 Assembly Code***





*Figure 2.13. Debugger Command #9 Assembly Code*

***2.2.10-) Debugger Command #10: GO***

It is similar to 2.2.1

***2.2.10.1-) Debugger Command #10 Algorithm and Flowchart***

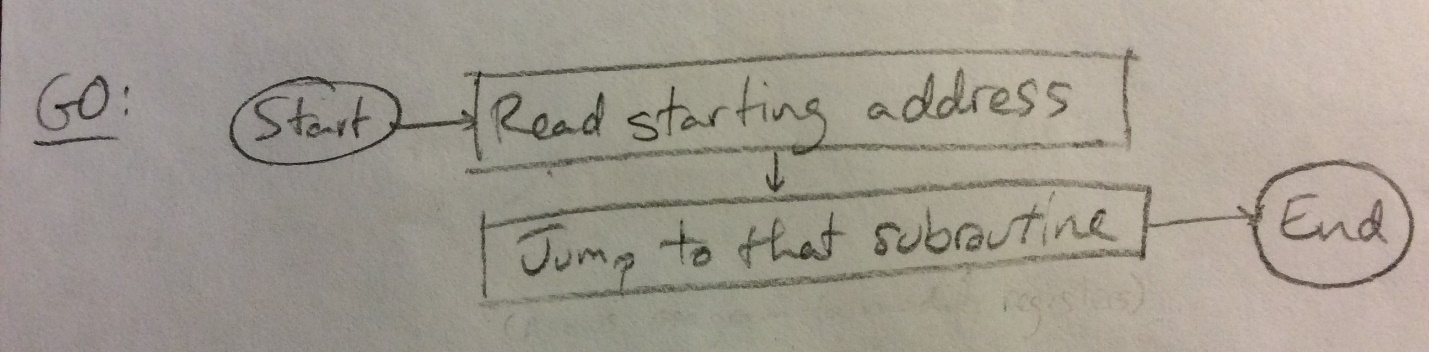
*GO*

*Read starting address from input*

*Jump to that subroutine // execute user’s program*

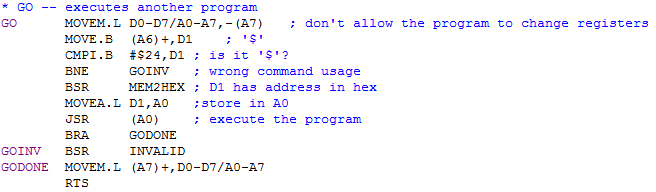
*Finish*

*Figure 2.13. Debugger Command #10 Flowchart*



*Figure 2.13. Debugger Command #10 Flowchart*

***2.2.10.2-) Debugger Command #10 Assembly Code***



*Figure 2.13. Debugger Command #10 Assembly Code*

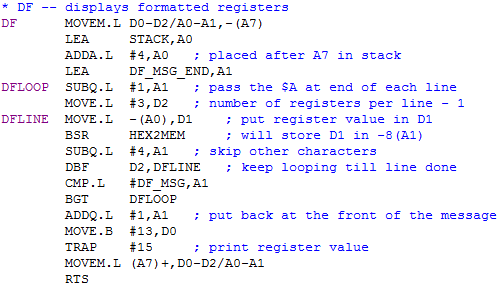
***2.2.11-) Debugger Command #11: DF***

It is similar to 2.2.1

***2.2.11.1-) Debugger Command #11 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.11.2-) Debugger Command #11 Assembly Code***



*Figure 2.13. Debugger Command #11 Assembly Code*

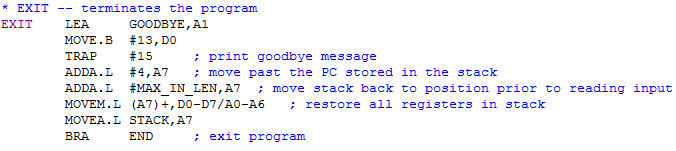
***2.2.12-) Debugger Command #12: EXIT***

It is similar to 2.2.1

***2.2.12.1-) Debugger Command #12 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.12.2-) Debugger Command #12 Assembly Code***



*Figure 2.13. Debugger Command #12 Assembly Code*

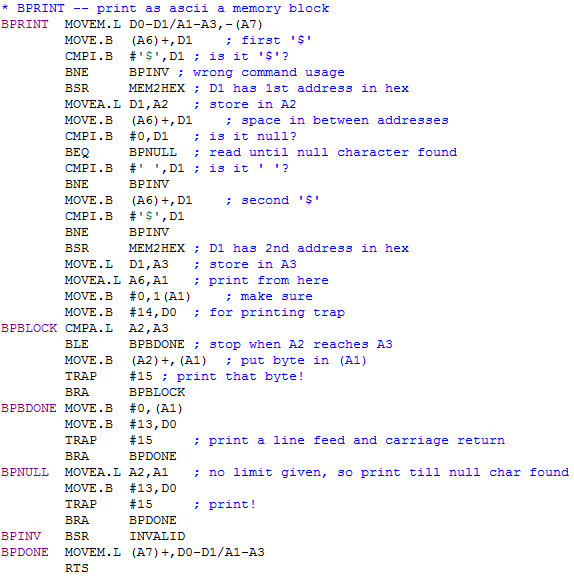
***2.2.13-) Debugger Command #13***

It is similar to 2.2.1

***2.2.13.1-) Debugger Command #13 Algorithm and Flowchart***

It is similar to 2.2.1.1

***2.2.13.2-) Debugger Command #13 Assembly Code***

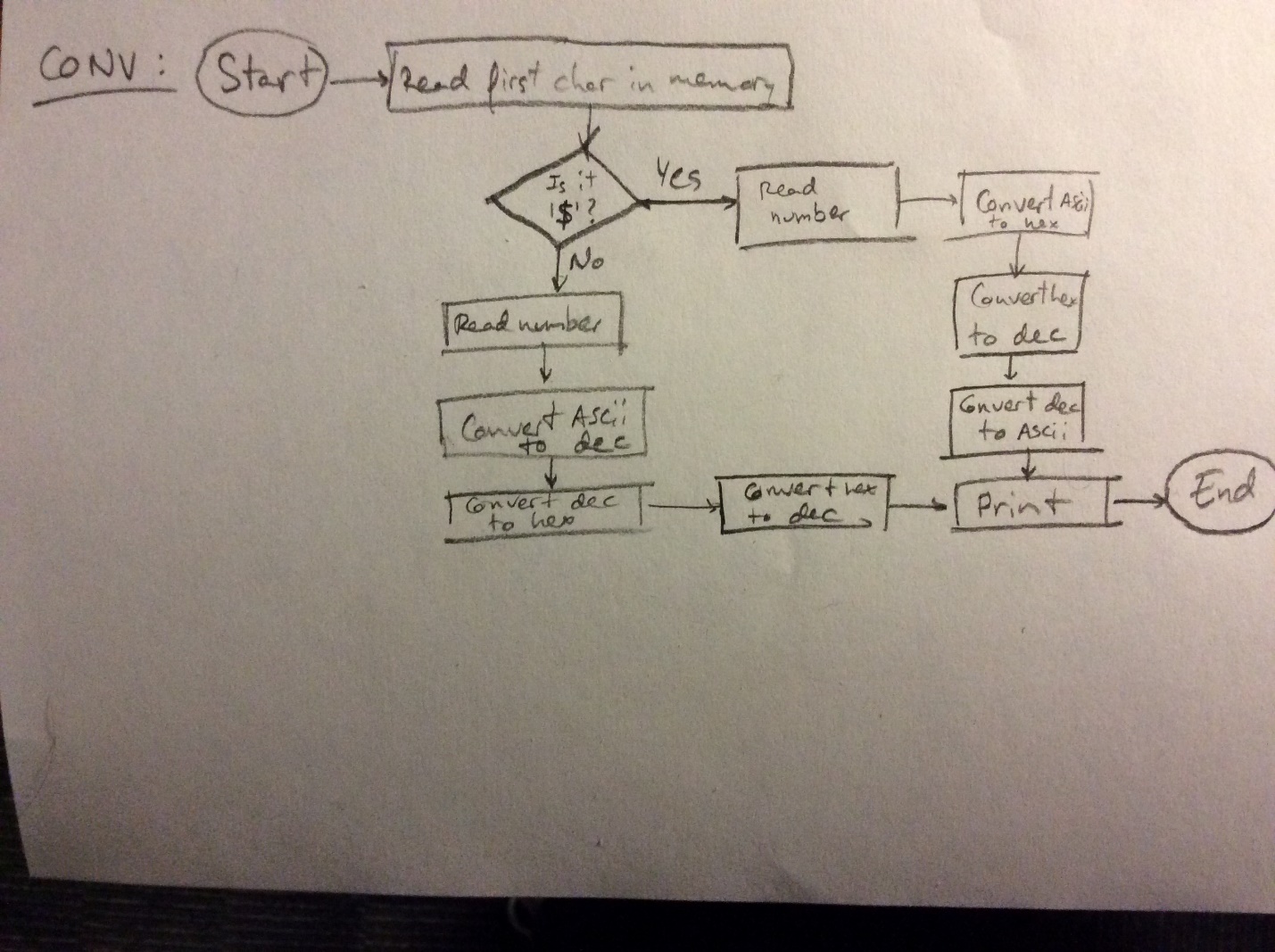


*Figure 2.13. Debugger Command #13 Assembly Code*

***2.2.14-) Debugger Command #14: CONV***

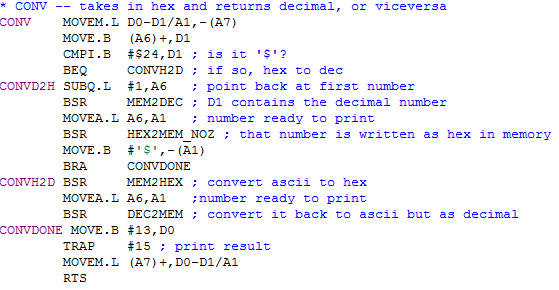
It is similar to 2.2.1

***2.2.14.1-) Debugger Command #14 Algorithm and Flowchart***



*Figure 2.13. Debugger Command #14 Flowchart*

***2.2.14.2-) Debugger Command #14 Assembly Code***



*Figure 2.13. Debugger Command #14 Assembly Code*

***2.3-) Exception Handlers***

Brief information about Exception Handlers should be given here.

***2.3.1-) Bus Error Exception***

A clear description of this debugger command should be given here.

***2.3.1.1-) Bus Error Exception Algorithm and Flowchart***

An algorithm of the design and its flowchart will be explained here. You may need to add comments for your algorithm.

*Clear //this where things starts*

*Do this m=0 // assign m*

*While m > n // while m > n*

*If m > n //*

*Do this //*

*Else //*

*Do these more //*

*End if //*

*m = m + 1 // increment m by 1*

*finish // finish*

*Figure 2.8. Debugger Command # 1 Algorithm*

It may be necessary to explain more about your flowchart and your design ideas.



*Figure 2.9. Debugger Command # 1 Flowchart*

***2.3.1.2-) Bus Error Exception Assembly Code***

The assembly code should be written using the algorithm above.

*ORG $1000*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*END ;*

*Figure 2.10. Debugger Command # 1 Assembly Code*

***2.3.2-) Address Error Exception***

It is similar to 2.3.1

***2.3.1.1-) Address Error Exception Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.1.2-) Address Error Exception Assembly Code***

It is similar to 2.3.1.2

***2.3.3-) Illegal Instruction Exception***

It is similar to 2.3.1

***2.3.3.1-) Illegal Instruction Exception Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.3.2-) Illegal Instruction Exception Assembly Code***

It is similar to 2.3.1.2

***2.3.4-) Privilege Violation Exception***

It is similar to 2.3.1

***2.3.4.1-) Privilege Violation Exception Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.4.2-) Privilege Violation Exception Assembly Code***

It is similar to 2.3.1.2

***2.3.5-) Divide by Zero Exception***

It is similar to 2.3.1

***2.3.5.1-) Divide by Zero Exception Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.5.2-) Divide by Zero Exception Assembly Code***

It is similar to 2.3.1.2

***2.3.6-) Line A and Line F Emulators***

It is similar to 2.3.1

***2.3.6.1-) Line A and Line F Emulators Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.6.2-) Line A and Line F Emulators Assembly Code***

It is similar to 2.3.1.2

***2.4-) Quick User Instruction Manual***

The following text contains a quick user manual with usage for all commands. All addresses must be given in hex, and all hex values must be given with a preceding dollar sign ‘$’. It can be accessed from the running program by executing the ‘HELP’ command:

*HELP: Displays This Message*

*MDSP: Outputs Address And Memory Contents*

*Default address2: address1 + 16*

*MDSP <address1>[ <address2>] eg: MDSP $908 $904<CR>*

*SORTW: Sorts Unsigned Words In A Memory Block*

*Both address1 and address2 are inclusive*

*Default order: descending*

*SORTW <address1> <address2>[ A|D] eg: SORTW $2000 $201E A<CR>*

*MM: Modifies Data In Memory*

*Default: Displays one byte*

*W: Displays one word*

*L: Displays one long word*

*MM <address>[ size]*

*MS: Set Memory To Given ASCII Or Hex*

*Default: ASCII. Prepend $ for hex (byte, word or long)*

*MS <address1> [$]<string|hex> eg: MS $4000 Hello!*

*BF: Fills Block Of Memory With Word Pattern*

*Both addresses must be even*

*Default pattern: 0000*

*If less than 4 digits given, right justified and zero padded*

*BF <address1> <address2>[ pattern] eg: BF $2000 $2200 4325<CR>*

*BMOV: Duplicate A Memory Block At Another Address*

*Must provide two addresses (inclusive, exclusive) for first block*

*Only one address (inclusive start) for second block*

*BMOV <address1.1> <address1.2> <address2>*

*BTST: Test Memory Block*

*BTST <address1> <address2>*

*BSCH: Search In Memory Block*

*BSCH <address1> <address2> <string>*

*GO: Execute Another Program*

*GO <address1>*

*DF: Displays All Formatted Registers eg: DF<CR>*

*EXIT: Exit The Monitor Program eg: EXIT<CR>*

*The two extra commands:*

*BPRINT: Print Block Of Memory*

*Default end: wherever a null char is found*

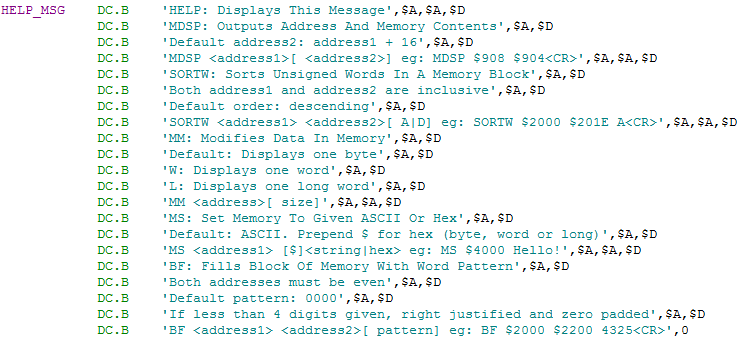
*BPRINT <address1>[ <address2>]*

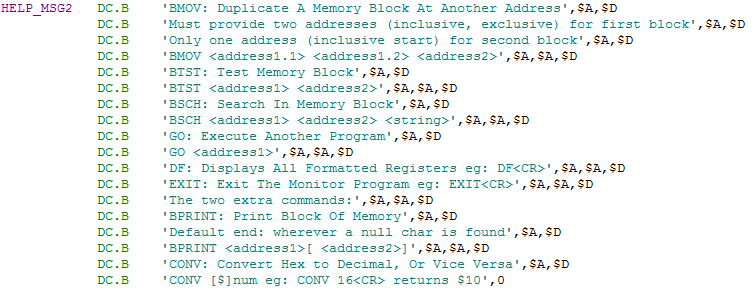
*CONV: Convert Hex to Decimal, Or Vice Versa*

*CONV [$]num eg: CONV 16<CR> returns $10*

***2.4.1-) Assembly Code***

The above message is stored in memory between locations $10FE and $16FB (~1.5KB). The assembly code for it is shown below:





*Figure 2.13. Help Message Assembly Code*

***3-) Discussion***

Design challenges and discussion about this project should be provided here. You may subdivide this section further and supply figures and table if necessary.

***4-) Feature Suggestions***

You may suggest ideas for expanding this project, such as exception handlers, etc. You may subdivide this section further and supply figures and tables if necessary.

***5-) Conclusion***

All commands and exception handling routines were implemented successfully. Any user will be capable of executing the previously described functionality. With the help of the Quick User Instruction Manual, it is not hard to get started quickly.

In addition, error handling has been implemented, so that it is difficult and unexpected for the common user to break the code or come across unforeseen hindrances. As long as the commands are used in a logical manner, following the given descriptions, the program will run appropriately.

***6-) References***

[1] T. Harman and D. Hein, “The Motorola MC 68000 Microprocessor Family”, Prentice-Hall Inc., Englewood Cliffs, NJ, 1996.

[2] A. Clements, “Microprocessor Systems Design”, PWS Publishing Company, Boston, MA, 1997.

[3] Educational Computer Board Manual

[4] Experiment 2 Lab Manual

[5] Experiment 3 Lab Manual