ECE 441
Microcomputers and Embedded Computing Systems

Instructor: Dr. Jafar Saniie
Teaching Assistant: Xinrui Yu

Final Project Report:
Monitor Project

04/15/2020

By: Alexander Lukens (A20417036)

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

Signature: ___________Alexander Lukens____________

Table of Contents:

Contents

Abstract

This monitor project will entail the creation of a robust debugging environment for the Motorola MC68000 Microprocessor Family. Code will be included to receive input from the user at the terminal, interpret this input, and run debugging commands accordingly. These debugger commands will assist the user in displaying register content from the MC68000 microprocessor, read and alter system memory, store ASCII and HEX data, test blocks of memory, and execute programs starting at a certain memory location.

The monitor program will also include exception handlers to assist the user in recovering from various errors that may occur during program operation, such as bus errors, address error, and privilege violation errors.

This program will be created entirely in the EASY68K simulator environment, meaning that all I/O Trap functions will correspond to the values required in EASY68K. If this program is to be executed on other MC68000 systems, the Trap I/O functions used throughout the Monitor program will have to be altered.

This project should be considered a success if:

1. A command interpreter for user input is successfully implemented
2. All debugger commands are implemented with minimal coding
3. Exception handlers are created and function correctly
4. The monitor program operates successfully in a variety of testing scenarios

1. Introduction

This design project will focus primarily on creating a functional debugging environment for the Motorola MC68000. The code will be compiled and executed inside the Easy68K program, however it should be able to be adapted to function on any MC68000 system by changing the output trap functions accordingly. The debugger will allow the user to input strings through the terminal to alter memory, display system register contents, and execute programs stored in memory

The debugging environment will have 12 commands to assist the user with debugging programs:

- HELP (Help)
- MDSP (Memory Display)
- MM (Memory Modify)
- MS (Memory Set)
- BF (Block Fill)
- BMOV (Block Move)
- BTST (Block Test)
- BSCH (Block Search)
- GO (Execute Program)
- DF (Display Formatted Registers)
- EXIT (Exit Monitor Program)
- AND (Logical AND)
- ADD (Add Memory)

Additionally, the debugging environment will be able to handle all 8 different system exceptions of the MC68000:

- Bus Error Exception
- Address Error Exception
- Illegal Instruction Exception
- Privilege Violation Exception
- Divide by Zero Exception
- CHK Instruction Exception
- Line A Emulator Exception
- Line F Emulator Exception

2. Monitor Program

The main function of the Monitor program will be to provide the user with a robust environment for creating, executing, and debugging MC68000 programs. The code shall be written so that it can be saved in an S-Record file and uploaded to any other computer using a processor from the Motorola MC68000 Family. The Monitor program will consist of several distinct parts: initialization tasks, the command interpreter, debugger commands, and exception handlers.

The initialization tasks performed by the monitor program will ensure that exception handlers function properly by setting the specific exception vectors in the MC68000 vector table to the correct values. Upon initialization, the monitor program will also display a welcome message to acknowledge that the system has successfully started.

The command interpreter will allow for the user to input a string from the terminal and execute one of the debugging commands included in the monitor program. The command interpreter itself will be responsible for interpreting the string input from the terminal and passing control to one of the debugger programs if a correct command string is received. If an incorrect string is input, an error message should be shown assisting the user in proper operation of the Monitor program.

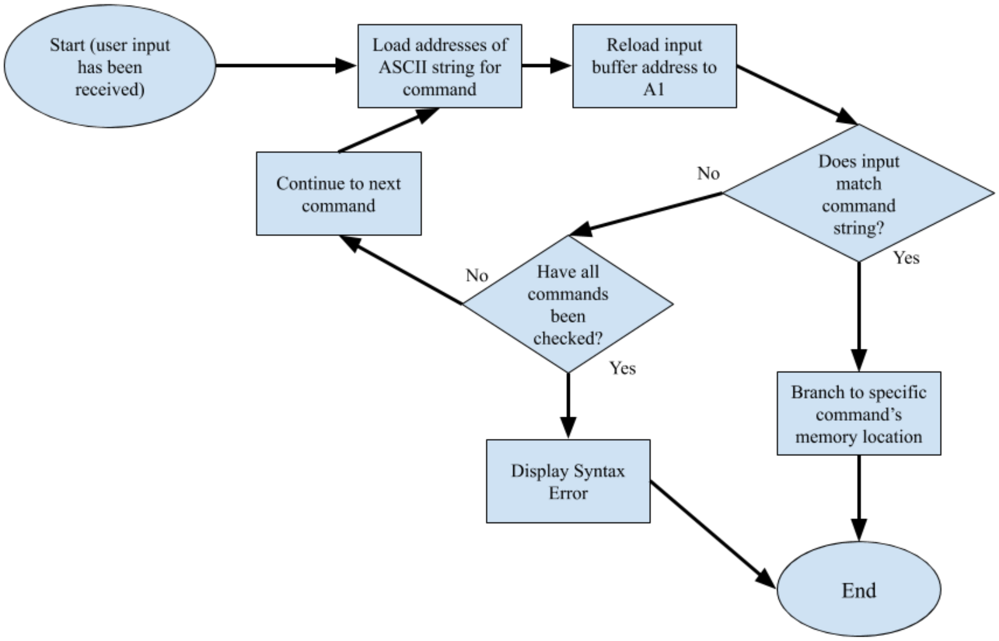
The Debugger commands will include a variety of commands that allow the user to perform different operations on the memory of the system. This will include viewing and editing memory contents, storing ASCII and HEX data, verifying the functionality of a block of memory locations, and executing a program from a specified memory location. Additionally, diagnostic commands will be included to display the contents of system registers and search a memory range for an ASCII string.

The exception handling functionality of the Monitor program will allow the system to recover from unexpected errors in their programs and will assist in debugging programs. The exception handlers will display helpful information about the error that has occurred, including the type of exception that has occurred and the contents of various system registers.

2.1. Command Interpreter

Used to initialize exception vectors, display command cursor, and interpret user input from the terminal. If user input matches a debugger command, the interpreter will execute that command, otherwise a syntax error will be displayed

2.1.1. Algorithm and Flowchart


Start (user input has been received)
Load addresses of ASCII string for command
Reload input buffer address to A1
Does input match command string?
No
Yes
Branch to specific command's memory location
Have all commands been checked?
No
Yes
Continue to next command
Display Syntax Error
End

```
Start                           ;user input (X) received
        While all commands not checked
                Load next command string (Y)
                Reload buffer (X) address
                If X=Y
                        Branch to command address
                Else, repeat loop
        Display Syntax Error            ;if all commands checked, display syntax error
End
```

2.1.2. Assembly Code

```
INIT
    LEA.L   WELCOME,A5
    LEA.L   E_WELCOME,A6
    BSR PRINT   ;print welcome msg

    ;initialize exception vectors

    MOVE.L #ILL_INSTR_EXC,$10
    MOVE.L #CHK_INSTR_EXC,$18
    MOVE.L #PRIV_VIOL_EXC,$20
    MOVE.L #LINE_A_EXC,$28
    MOVE.L #LINE_F_EXC,$2C
    MOVE.L #DIV_ZERO_EXC,$14
```

```
    MOVE.L #BUS_EXC,$8
    MOVE.L #ADDR_EXC,$C
    MOVE.L #$8000,A7

MAINP
    LEA.L   CURSOR,A5 ;load cursor address
to A1
    LEA.L   E_CURSOR,A6
    BSR PRINT_NC      ;print cursor string
    BSR GET_IN
    CMP.B   #NULL,D1     ;check if length
of input string is 0
```

```
    BEQ     MAINP       ;if so, branch to
MAINP
    BSR INTERPRET
    BRA MAINP
INTERPRET   ;DETERMINE WHICH COMMAND WAS
ENTERED
    LEA.L   CMD_HELP,A5
    LEA.L   E_CMD_HELP,A6
    LEA.L   BUFFER,A1
    BSR COMPARE
    BEQ HELP        ;IF HELP COMMAND
INPUT, GO TO HELP CMD

    LEA.L   CMD_MDSP,A5
    LEA.L   E_CMD_MDSP,A6
    LEA.L   BUFFER,A1       ;load address
of CMD string
    BSR COMPARE         ;compare input
buffer to CMD string
    BEQ MDSP    ;if string is correct,
branch to CMD

    LEA.L   CMD_MM,A5
    LEA.L   E_CMD_MM,A6 ;load address of
CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ MM      ;if string is correct,
branch to CMD

    LEA.L   CMD_BF,A5
    LEA.L   E_CMD_BF,A6 ;load address of
CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ BF      ;if string is correct,
branch to CMD

    LEA.L   CMD_BTST,A5
    LEA.L   E_CMD_BTST,A6   ;load address
of CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ BTST    ;if string is correct,
branch to CMD

    LEA.L   CMD_EXIT,A5
    LEA.L   E_CMD_EXIT,A6   ;load address
of CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ EXIT    ;if string is correct,
branch to CMD

    LEA.L   CMD_BSCH,A5
    LEA.L   E_CMD_BSCH,A6   ;load address
of CMD string
    LEA.L   BUFFER,A1
```

```
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ BSCH    ;if string is correct,
branch to CMD

    LEA.L   CMD_GO,A5
    LEA.L   E_CMD_GO,A6 ;load address of
CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ GO      ;if string is correct,
branch to CMD

    LEA.L   CMD_MS,A5
    LEA.L   E_CMD_MS,A6 ;load address of
CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ MS      ;if string is correct,
branch to CMD

    LEA.L   CMD_BMOV,A5
    LEA.L   E_CMD_BMOV,A6   ;load address
of CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ BMOV    ;if string is correct,
branch to CMD

    LEA.L   CMD_DF,A5
    LEA.L   E_CMD_DF,A6 ;load address of
CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ DF      ;if string is correct,
branch to CMD

    LEA.L   CMD_AND,A5
    LEA.L   E_CMD_AND,A6    ;load address
of CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ AND     ;if string is correct,
branch to CMD

    LEA.L   CMD_ADD,A5
    LEA.L   E_CMD_ADD,A6    ;load address
of CMD string
    LEA.L   BUFFER,A1
    BSR COMPARE ;compare input buffer to
CMD string
    BEQ ADD     ;if string is correct,
branch to CMD


    BSR SYNTAX_CMD ;if incorrect syntax,
display error
    BRA MAINP       ;go to main program
```

2.2. Debugger Commands

These debugger commands will be utilized by users to assist with program execution and debugging. Commands will allow users to modify memory contents, perform operations on memory locations, execute a program, and display system register contents.

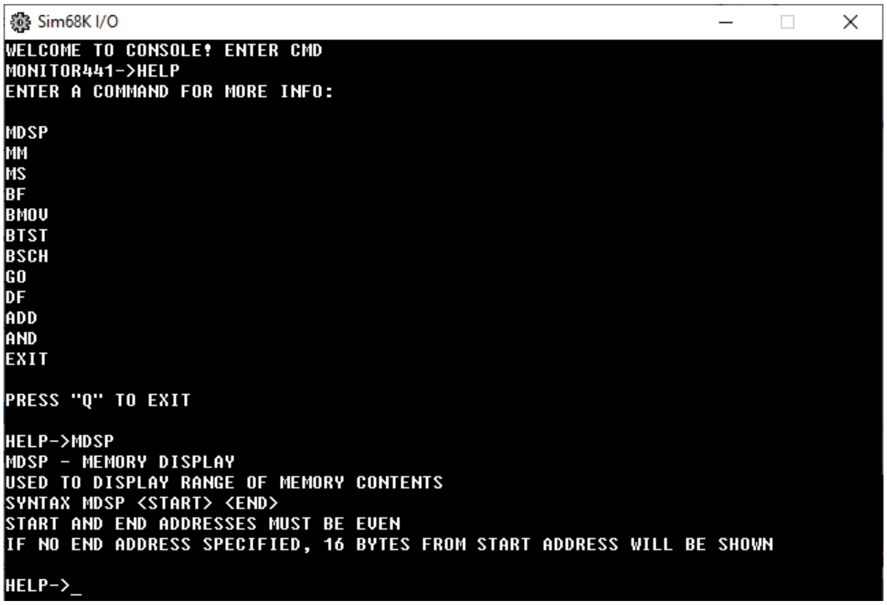
2.2.1. HELP (Help)

The HELP command is used to show a list of commands available in the debugger and the corresponding syntax and information about each command. The HELP command enters a new interpreter environment, allowing the user to enter a specific command and receive detailed information about its functionality and syntax.

Syntax: HELP

When in the environment, enter a specific command to view information about the command.

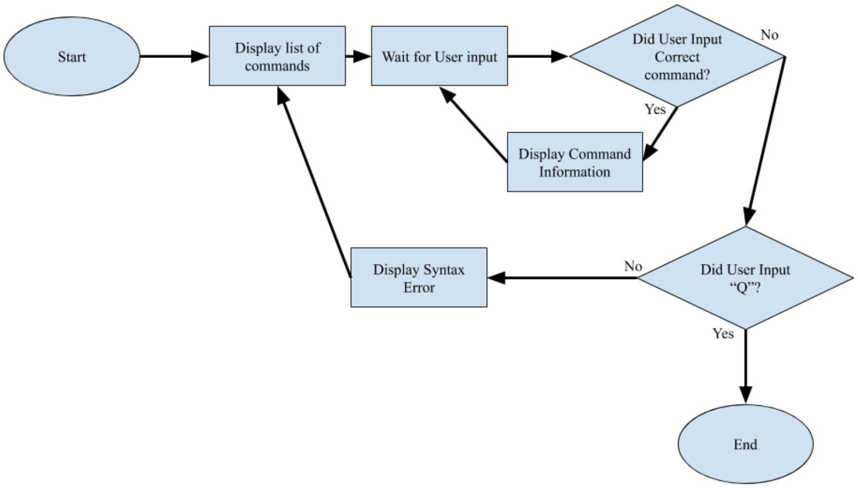
Enter "Q" to exit the HELP environment.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->HELP
ENTER A COMMAND FOR MORE INFO:

MDSP
MM
MS
BF
BMOV
BTST
BSCH
GO
DF
ADD
AND
EXIT

PRESS "Q" TO EXIT

HELP->MDSP
MDSP - MEMORY DISPLAY
USED TO DISPLAY RANGE OF MEMORY CONTENTS
SYNTAX MDSP <START> <END>
START AND END ADDRESSES MUST BE EVEN
IF NO END ADDRESS SPECIFIED, 16 BYTES FROM START ADDRESS WILL BE SHOWN

HELP->_
```

Figure 1: HELP Sample Output

2.2.1.1. Algorithm and Flowchart



```
Start                ;user enters help command
Display list of commands
Get user input (X)
          While all commands not checked
                    Reset buffer address (X)
                    Get next command string (Y)
                    If X=Y
                              Branch to help message for Y
                    Else Continue loop

          If X= "Q"
                    Exit help command
          Else Display syntax error
          Repeat loop
End
```

2.2.1.2. Assembly Code

```
*-----------HELP------------*
HELP
   LEA.L HELP_MSG,A5
   LEA.L E_HELP_MSG,A6       ;print list of commands
   BSR PRINT
HELP_LOOP
   LEA.L CURSOR_HELP,A5
   LEA.L E_CURSOR_HELP,A6 ;display HELP-> cursor
   BSR PRINT_NC
   BSR GET_IN                ;get user input
   LEA.L BUFFER,A1
   CMPI.B #NULL,(A1)
   BEQ HELP_LOOP

   LEA.L CMD_MDSP,A5
   LEA.L E_CMD_MDSP,A6
   LEA.L BUFFER,A1 ;CHECK TO SEE IF MDSP WAS
ENTERED
   BSR COMPARE
   BEQ HELP_MDSP

   LEA.L CMD_MM,A5
   LEA.L E_CMD_MM,A6
   LEA.L BUFFER,A1 ;CHECK TO SEE IF MM WAS
ENTERED
   BSR COMPARE
   BEQ HELP_MM

   LEA.L CMD_MS,A5
   LEA.L E_CMD_MS,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_MS

   LEA.L QUIT,A5
   LEA.L E_QUIT,A6 ;CHECK TO SEE IF Q WAS ENTERED
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ MAINP       ;if so, leave HELP

   LEA.L CMD_BSCH,A5
   LEA.L E_CMD_BSCH,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BSCH
   *insert other help cmds here + usage info*

   LEA.L CMD_BF,A5
   LEA.L E_CMD_BF,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BF

   LEA.L CMD_BMOV,A5
   LEA.L E_CMD_BMOV,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BMOV

   LEA.L CMD_BTST,A5
   LEA.L E_CMD_BTST,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BTST

   LEA.L CMD_GO,A5
   LEA.L E_CMD_GO,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_GO

   LEA.L CMD_DF,A5
   LEA.L E_CMD_DF,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_DF

   LEA.L CMD_AND,A5
   LEA.L E_CMD_AND,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_AND
```

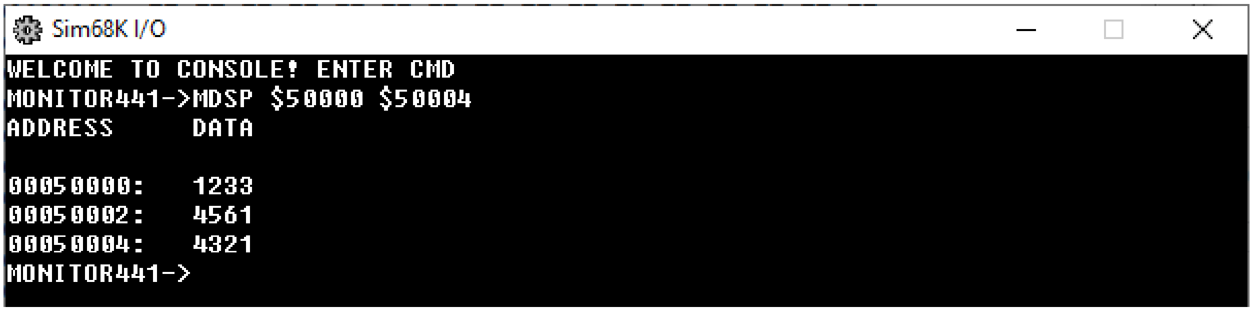
```
    LEA.L CMD_ADD,A5
    LEA.L E_CMD_ADD,A6
    LEA.L BUFFER,A1
    BSR COMPARE
    BEQ HELP_ADD

    LEA.L CMD_EXIT,A5
    LEA.L E_CMD_EXIT,A6
    LEA.L BUFFER,A1
    BSR COMPARE
    BEQ HELP_EXIT
    BRA HELP_SYNTAX
    BRA HELP        ;repeat until a correct option selected
HELP_SYNTAX
    LEA.L HELP_SYNTAX_MSG,A5
    LEA.L E_HELP_SYNTAX_MSG,A6
    BSR PRINT
    BRA HELP
HELP_MDSP
    LEA.L MDSP_HELP_MSG,A5
    LEA.L E_MDSP_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_MM
    LEA.L MM_HELP_MSG,A5
    LEA.L E_MM_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_MS
    LEA.L MS_HELP_MSG,A5
    LEA.L E_MS_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_BSCH
    LEA.L BSCH_HELP_MSG,A5
    LEA.L E_BSCH_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_BF
    LEA.L BF_HELP_MSG,A5
    LEA.L E_BF_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_BMOV
    LEA.L BMOV_HELP_MSG,A5
    LEA.L E_BMOV_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_BTST
    LEA.L BTST_HELP_MSG,A5
    LEA.L E_BMOV_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_GO
    LEA.L GO_HELP_MSG,A5
    LEA.L E_GO_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_DF
    LEA.L  DF_HELP_MSG,A5
    LEA.L  E_DF_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_AND
    LEA.L AND_HELP_MSG,A5
    LEA.L E_AND_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_ADD
    LEA.L ADD_HELP_MSG,A5
    LEA.L E_ADD_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
HELP_EXIT
    LEA.L   EXIT_HELP_MSG,A5
    LEA.L   E_EXIT_HELP_MSG,A6
    BSR PRINT
    BRA HELP_LOOP
```

2.2.2. MDSP (Memory Display)

The MDSP command is used to display a range of memory contents in word size.

Syntax: MDSP <START> <END>

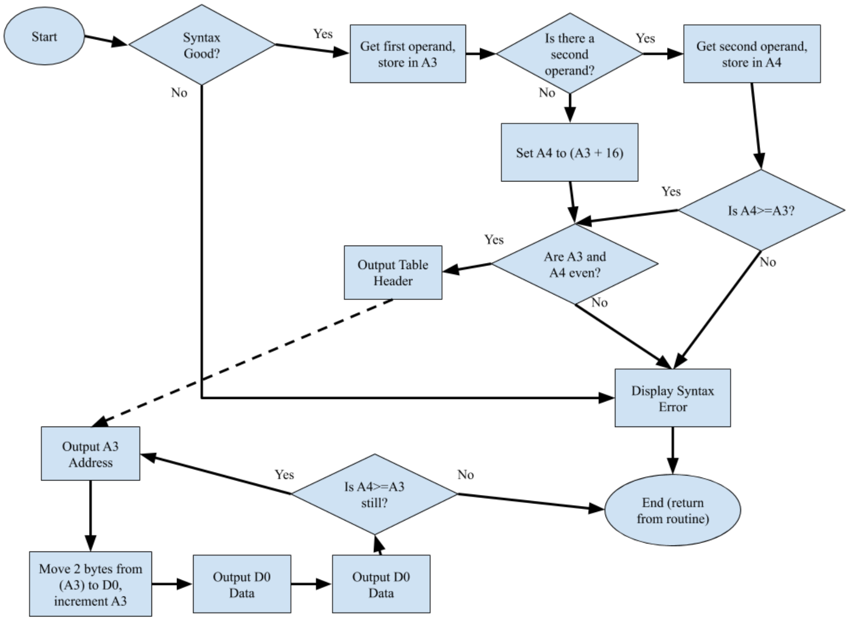
The starting and ending addresses must be even. If no ending address is specified, the command will print the next 16 bytes from the starting address.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->MDSP $50000 $50004
ADDRESS     DATA

00050000:   1233
00050002:   4561
00050004:   4321
MONITOR441->
```

Figure 2: MDSP Sample Output

2.2.2.1. Algorithm and Flowchart



```
Start
    If syntax NOT good
        Display Syntax error
    Else
        Get first operand (X)
        If second operand available
            Get 2nd operand (Y)
        Else
            Y=X+16
        While Y>X
            Output X address, display contents
            Increment X by 2
            Repeat Loop
        If Y<=X
            Exit program
End
```

2.2.2.2. Assembly Code

```
*----------MDSP-----------*
MDSP    ; Lower memory address stored in A3, upper stored in A4
        ; will output values from A3 to A4
        ; MEMORY RANGE MUST START AND END AT EVEN VALUES
   MOVEM.L D0-D2/A3-A6,-(SP)

   CMPI.B #SPACE,(A1)+
   BNE SYNTAX
   BSR ASC2HEX
   MOVE.L D0,A3        ;copy starting address to A3
   LEA MDSP_OUTPUT,A5
   LEA E_MDSP_OUTPUT,A6 ;output memory display header
   BSR PRINT
   CMPI.B #SPACE,(A1)+
   BNE MDSP_ONE       ;if no ending address, specified
output 16 bytes
   BSR ASC2HEX
   MOVE.L D0,A4       ;if ending address specified, copy
to A4

MDSP_TEST
   CMP.L A3,A4        ;make sure A4>A3
   BLT SYNTAX
   MOVE.L A3,D2



   BTST #0,D2         ;ensure starting address is even
   BNE SYNTAX
   MOVE.L A4,D2
   BTST #0,D2         ;ensure ending address is even
   BNE SYNTAX

MDSP_LOOP
```

```
   BSR A3OUT      ;branch to A3OUT Subroutine
   MOVE.W (A3)+,D0 ;move 2 bytes of memory values to
D0, increment A3
   MOVE.B #2,D1
   BSR HEX2ASC
   BSR PRINT
   CMP.L A3,A4    ;check if at end of loop
   BGE MDSP_LOOP
   BRA MDSP_END
MDSP_ONE    ;if only one operand specified, set
A4(ending address) to A3(Start)+ 14
   MOVEA.L A3,A4
   ADDA.L #14,A4   ;set A4 to A3+14 (will display 16
bytes)
   BSR MDSP_TEST   ;branch to test

MDSP_END
   MOVEM.L (SP)+,D0-D2/A3-A6
   BRA MAINP      ;end command
```

2.2.3. MM (Memory Modify)

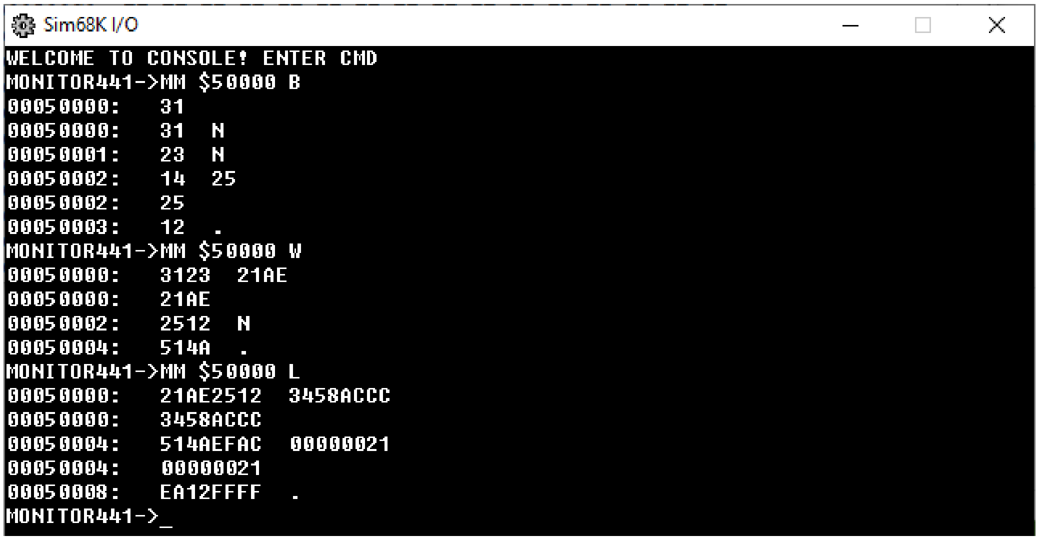
The MM command is used to display and modify memory contents.

Syntax: MM <ADDRESS> <SIZE>

The size parameter controls the size of data to be displayed and modified by the command.

Acceptable values: ‘B’ for Byte size, ‘W’ for Word size, ‘L’ for Long size.

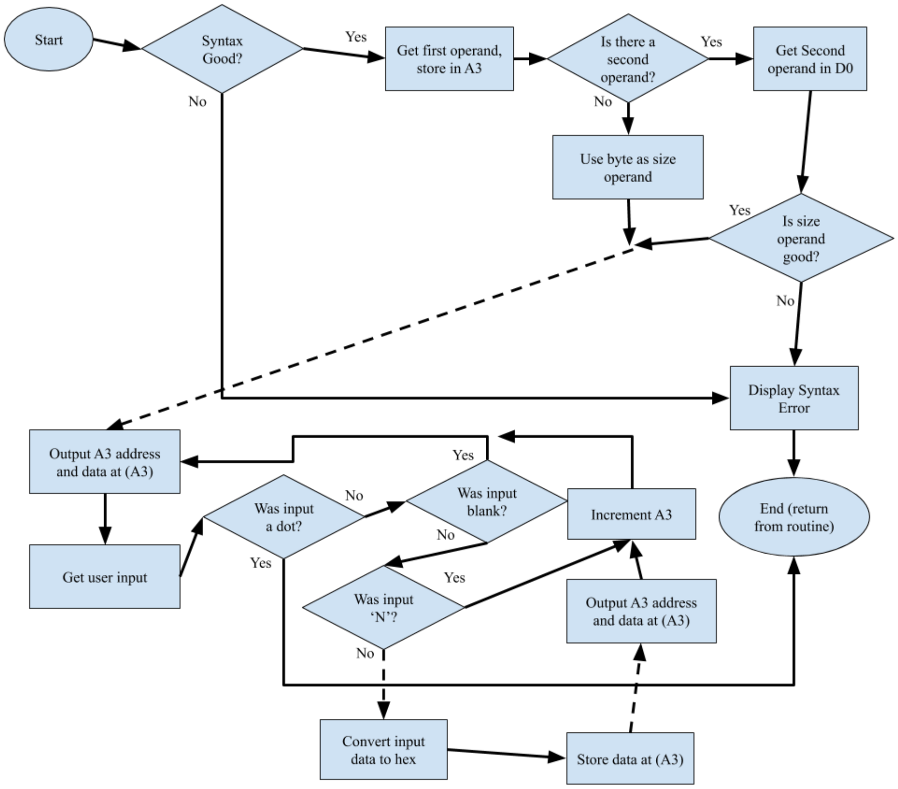
To skip to the next memory location, ener ‘N’. To exit the command, enter ‘.’ (period). To modify a data value, type in the corresponding hex data to be updated at the location. If invalid data is entered a syntax error will occur. If too much data is entered (ex, 2 bytes of data during byte sized operation), the command will repeat



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->MM $50000 B
00050000:   31
00050000:   31  N
00050001:   23  N
00050002:   14  25
00050002:   25
00050003:   12  .
MONITOR441->MM $50000 W
00050000:   3123  21AE
00050000:   21AE
00050002:   2512  N
00050004:   514A  .
MONITOR441->MM $50000 L
00050000:   21AE2512  3458ACCC
00050000:   3458ACCC
00050004:   514AEFAC  00000021
00050004:   00000021
00050008:   EA12FFFF  .
MONITOR441->_
```

Figure 3: MM Sample Output

2.2.3.1. Algorithm and Flowchart



```
Start
    If syntax NOT good
        Display Syntax error
    Else
        Get first operand (X)  ;this is the starting address
        If no second operand
            Use byte size addressing
        Else
            Get 2nd operand (Y)
                If Y=”B”
                    Use byte size addressing
                Else if Y=”W”
                    Use word size addressing
                Else if Y=”L”
                    Use long size addressing
                Else display syntax error
        While (Z = “.” period)                    ;Z is user input
            Display X and data at X       ; X is the current address
            Get user input (Z)
            If user input = “.” period
                Exit program
        Else if user input= "N"
                Jump to next address (increment X)
        Else
                convert Z to HEX data
                Save Z at X address             ;store user data
                Display X and data at X
                Increment X                     ;go to next address
End
```

2.2.3.2. Assembly Code

```
*--------------MM----------------*
MM ; if no size operand, will default to byte size
   MOVEM.L D6,-(SP)
   CMP.B #SPACE,(A1)+   ;check cmd syntax
   BNE SYNTAX
   BSR ASC2HEX
   MOVE.L D0,A3        ;move starting address to A3
   CMP.B #SPACE,(A1)
   BNE MM_B
   ADDA.L #1,A1
   CMP.B #B_AS,(A1)    ;choose correct modify size
   BEQ MM_B
   CMP.B #W_AS,(A1)
   BEQ MM_W
   CMP.B #L_AS,(A1)
   BEQ MM_L
   BRA SYNTAX
MM_B    ;will alter memory in byte steps
       ;assumes correct input size
   BSR A3OUT           ;output address in A3
   MOVE.B (A3),D0
   MOVE.L #1,D1
   BSR D0OUT           ;outputs D0, byte size
   BSR GET_IN
   LEA.L BUFFER,A1
   CMP.B #DOT,(A1)     ; if DOT entered, exit routine
   BEQ MM_END
   CMP.B #NULL,(A1)    ; if NULL entered, repeat current
memory value
   BEQ MM_B
   CMP.B #N_AS,(A1)    ; if 'N' entered, move to next
location without updating memory
   BEQ MM_B_OUT
   BSR ASC2HEX
   CMP.L #2,D2         ;if input too long, repeat prompt
   BGT MM_B
   MOVE.B D0,(A3)
   BSR A3OUT           ;output address in A3

   MOVE.B #1,D1
   BSR HEX2ASC
   BSR PRINT           ;Output updated data value
MM_B_OUT
   ADDA.L #1,A3        Add 1 byte to A3 address
   BRA MM_B
MM_W    ;will alter memory in word sized steps, errors if
starting address is odd
```

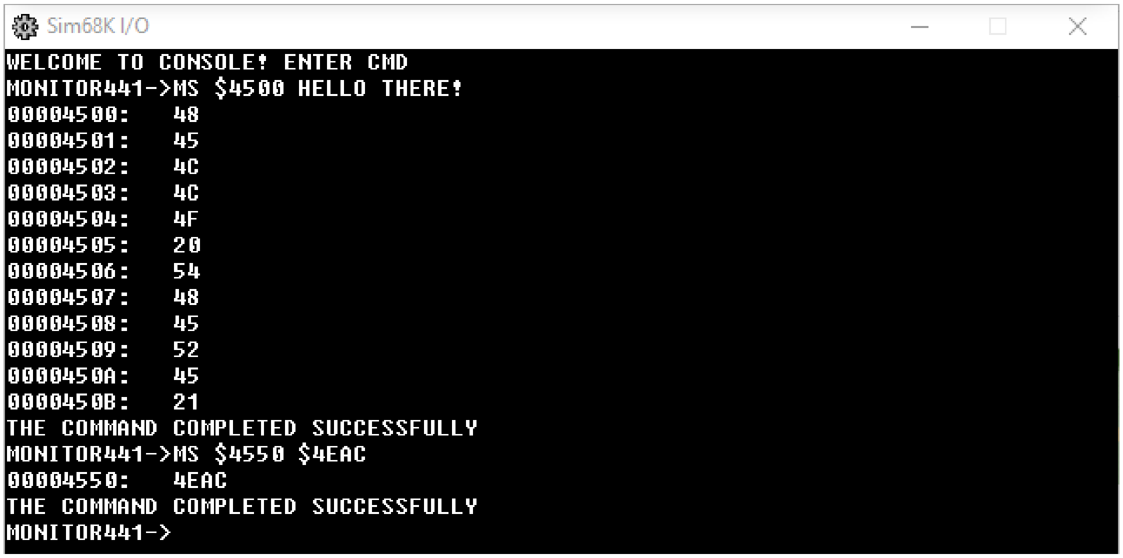
```
       ;Assumes correct input size. If smaller input received,
uses entire space to store value

   MOVE.L A3,D0
   BTST #0,D0          ;ensure that A3 address is even
   BNE SYNTAX
   BSR A3OUT           ;output address in A3
   MOVE.W (A3),D0      ;copy current word sized value at
(A3) to D0
   MOVE.L #2,D1
   BSR D0OUT           ;output current value at (A3)
   BSR GET_IN          ; Get user input
   LEA.L BUFFER,A1
   CMP.B #DOT,(A1)     ; if DOT entered, exit routine
   BEQ MM_END
   CMP.B #NULL,(A1)    ; if NULL entered, repeat current
memory value
   BEQ MM_W
   CMP.B #N_AS,(A1)    ; if 'N' entered, move to next
location without updating memory
   BEQ MM_W_OUT

   BSR ASC2HEX         ;convert input to Hex
   CMP.L #4,D2
   BGT MM_W            ;if input too long, repeat prompt
   MOVE.W D0,(A3)      ;update value at (A3)
   BSR A3OUT           ;output address in A3
   MOVE.B #2,D1
   BSR HEX2ASC
   BSR PRINT           ;Output updated data value
MM_W_OUT
   ADDA.L #2,A3        ;Add 2 bytes to A3 address
   BRA MM_W            ;repeat

MM_L    ;will alter memory in long sized steps, errors if
starting address is odd
   MOVE.L A3,D0
   BTST #0,D0          ;ensure that A3 address is even
   BNE SYNTAX
   BSR A3OUT           ;output address in A3
   MOVE.L (A3),D0
   MOVE.L #4,D1
   BSR D0OUT           ;output current value at (A3)
   BSR GET_IN
   LEA.L BUFFER,A1
   CMP.B #DOT,(A1)     ; if DOT entered, exit routine
   BEQ MM_END
   CMP.B #NULL,(A1)    ; if NULL entered, repeat current memory value
   BEQ MM_L
   CMP.B #N_AS,(A1)    ; if 'N' entered, move to next location without updating memory
   BEQ MM_L_OUT

   BSR ASC2HEX         ;convert input to Hex
   CMP.L #8,D2
   BGT MM_L            ;if input too long, repeat prompt
   MOVE.L D0,(A3)      ;update value at (A3)
   BSR A3OUT           ;output address in A3
   MOVE.B #4,D1
   BSR HEX2ASC
   BSR PRINT           ;Output updated data value
MM_L_OUT
   ADDA.L #4,A3        ;Add 4 bytes to A3 address
   BRA MM_L            ;repeat

MM_END
   MOVEM.L (SP)+,D6
   RTS
```

2.2.4. MS (Memory Set)

The MS command is used to store hex data or an ASCII string in memory.

Syntax: MS <ADDRESS> <DATA>

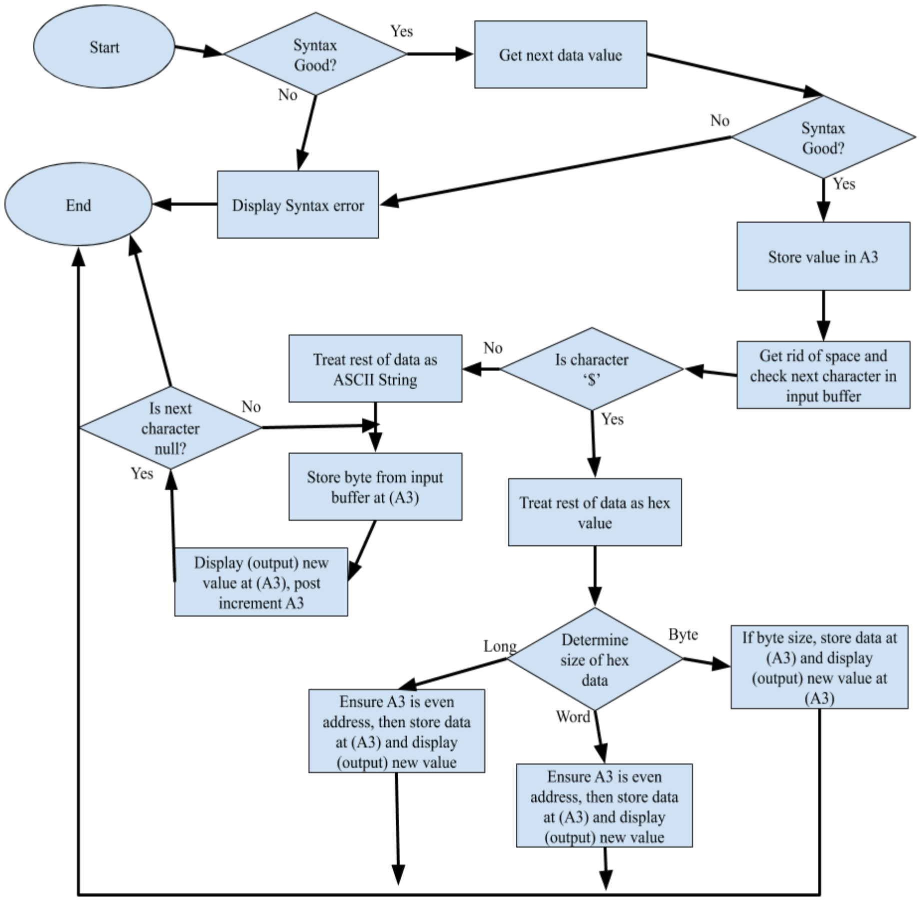
If entering hex data into memory, a dollar sign ($) must be used, otherwise the data will be considered to be an ASCII string. The user may store up to Longword sized data (4 bytes) and the command will automatically select the correct storage size. If a longer storage size is desired, leading zeroes must be added.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->MS $4500 HELLO THERE!
00004500:   48
00004501:   45
00004502:   4C
00004503:   4C
00004504:   4F
00004505:   20
00004506:   54
00004507:   48
00004508:   45
00004509:   52
0000450A:   45
0000450B:   21
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->MS $4550 $4EAC
00004550:   4EAC
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->
```

Figure 4: MS Sample Output

2.2.4.1. Algorithm and Flowchart


Start
Syntax Good?
Yes
No
Get next data value
Syntax Good?
No
Yes
End
Display Syntax error
Store value in A3
Get rid of space and check next character in input buffer
Is character '$'
No
Yes
Treat rest of data as ASCII String
Is next character null?
No
Yes
Store byte from input buffer at (A3)
Display (output) new value at (A3), post increment A3
Treat rest of data as hex value
Determine size of hex data
Long
Byte
Word
If byte size, store data at (A3) and display (output) new value at (A3)
Ensure A3 is even address, then store data at (A3) and display (output) new value
Ensure A3 is even address, then store data at (A3) and display (output) new value

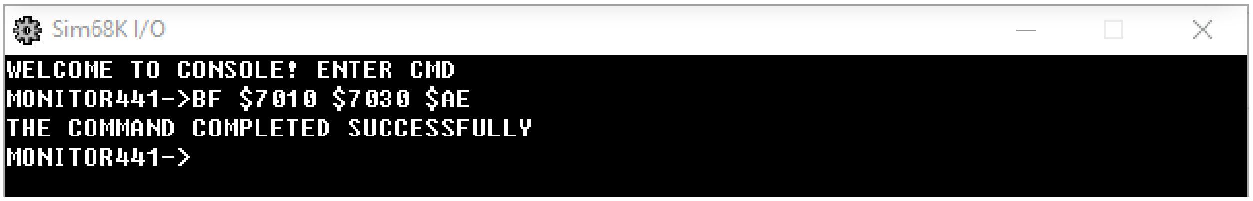
```
Start
    If syntax NOT good
        Display Syntax error
    Else
        Get first operand (X)  ;this is the starting address
        Get 2nd operand (Y)
        If first value (Y)='$'
            Store HEX value (Y) at (X)
        Else
            While (Y) not equal 'NULL'
                Copy byte from (Y), store at (X)
                Increment (Y) and (X)

End
```

2.2.4.2. Assembly Code

```
*----------MS-----------*
MS  ;if setting to hex, must have dollar sign in front of value. Otherwise will be
    ;assumed to be an ascii string
    CMP.B #SPACE,(A1)+
    BNE SYNTAX
    BSR ASC2HEX     ;receive starting address
    MOVE.L D0,A3    ;store address in A3
    CMP.B #SPACE,(A1)+
    BNE SYNTAX
    CMP.B #DOLLAR,(A1)  ;check if next character is '$'
    BEQ MS_HEX          ;if '$', store hex value
                    ;otherwise treat as ASCII string

MS_ASCII
    CMP.B #NULL,(A1)
    BEQ MS_END          ;end command if at end of ASCII string
    BSR A3OUT           ;output current address
    MOVE.B (A1)+,D0     ;move byte from ASCII string to D0
    MOVE.B #1,D1
    MOVE.B D0,(A3)+     ;store D0 at (A3), postincrement A3
    BSR D0OUT           ;output value in D0
    BSR PRINT           ;skip to next line
    BRA MS_ASCII        ;repeat

MS_HEX
    BSR ASC2HEX
    CMP.B #2,D2
    BLE MS_HEX_BYTE     ;if data is byte sized, branch to byte
    CMP.B #4,D2
    BLE MS_HEX_WORD     ;if data is word sized, branch to word
    CMP.B #8,D2
    BLE MS_HEX_LONG     ;if data is long sized, branch to long
    BRA SYNTAX
MS_HEX_BYTE
    MOVE.B D0,(A3)      ;copy byte from D0 to A3
    BSR A3OUT
    MOVE.L #1,D1        ;output current memory address
    BSR D0OUT
    BSR PRINT           ;output new data value
    BRA MS_END          ;end command
MS_HEX_WORD
    MOVE.L A3,D4
    BTST #0,D4          ;ensure that A3 is even
    BNE SYNTAX          ;if not even, invoke syntax error
    MOVE.W D0,(A3)      ;copy word from D0 to A3
    BSR A3OUT           ;output current memory address
    MOVE.L #2,D1
    BSR D0OUT           ;output new data value
    BSR PRINT
    BRA MS_END          ;end command
MS_HEX_LONG
    MOVE.L A3,D4
    BTST #0,D4          ;ensure that A3 is even
    BNE SYNTAX          ;if not even, invoke syntax error
    MOVE.L D0,(A3)      ;copy longword from D0 to A3
    BSR A3OUT           ;output current memory address
    MOVE.L #4,D1
    BSR D0OUT           ;output new data value
    BSR PRINT
    BRA MS_END          ;end command

MS_END
    BSR SUCCESS         ;print success message
    RTS             ;end command
```

2.2.5. BF (Block Fill)

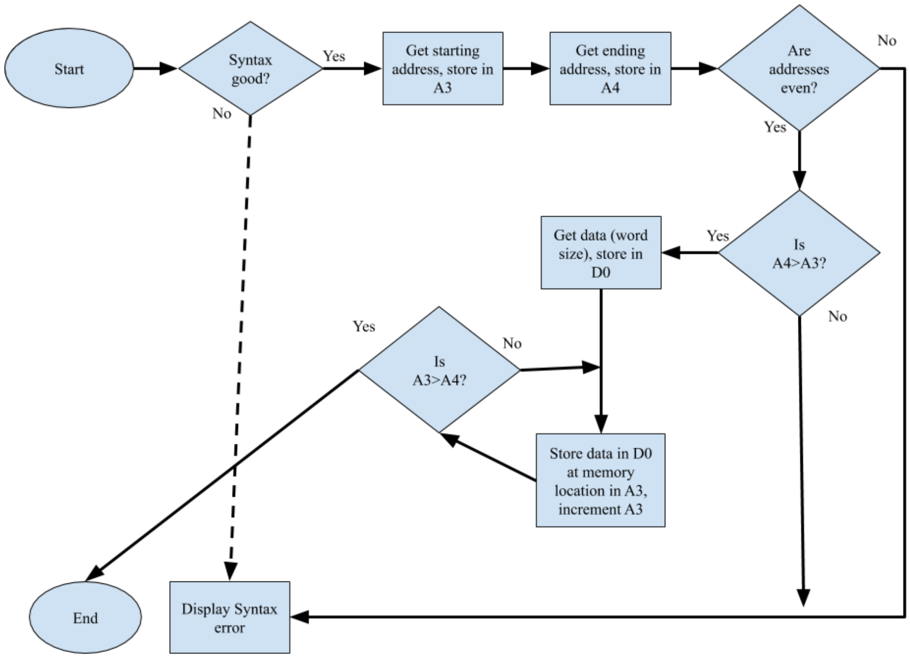
The BF command is used to fill memory with a designated word sized value.
Syntax: BF <START> <END> <DATA>
This command will overwrite all memory locations between the starting and ending addresses with the specified data. The starting and ending addresses must be even. If the data is less than word size, the data will be extended to word size.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->BF $7010 $7030 $AE
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->
```

Figure 5: BF Sample Output

2.2.5.1. Algorithm and Flowchart


Start
Syntax good?
Yes
No
Get starting address, store in A3
Get ending address, store in A4
Are addresses even?
No
Yes
Get data (word size), store in D0
Yes
Is A4>A3?
No
Yes
Is A3>A4?
No
Store data in D0 at memory location in A3, increment A3
End
Display Syntax error

```
Start
    If syntax not good
        Display syntax error message
    Else
        Get next 3 data values (X,Y,Z)
    If (X,Y,Z) not even
        Display syntax error
    Else
        While (X<=Y)
            Store data Z at memory location X
            Increment X
    Display success message
End
```

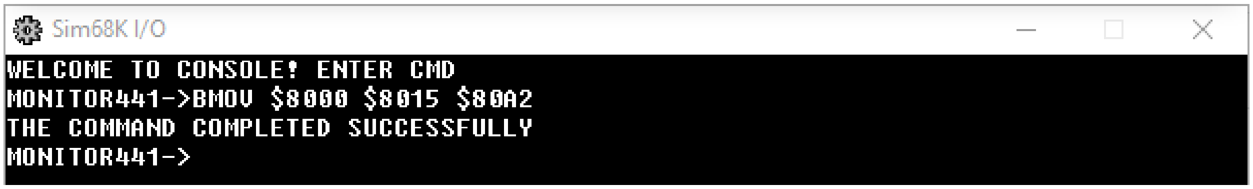
2.2.5.2. Assembly Code

```
*----------BF-----------*
BF  ; BF *START* *END* *DATA TO FILL*
   MOVEM.L A3-A4/D0-D1,-(SP)
   CMP.B #SPACE,(A1)+ ;check correct syntax
   BNE SYNTAX
   BSR ASC2HEX ;get starting address
   BTST #0,D0  ;make sure starting address is even
   BNE SYNTAX
   MOVE.L D0,A3        ;store starting address at A3
   CMP.B #SPACE,(A1)+
   BNE SYNTAX
   BSR ASC2HEX         ;get ending address
   BTST #0,D0          ;Make sure ending address is even
```

```
    BNE SYNTAX
    MOVE.L D0,A4        ;store ending address at A4
    CMPA.L A3,A4        ; make sure ending address is larger than first
    BLE SYNTAX
    CMP.B #SPACE,(A1)+    ;check syntax
    BNE SYNTAX
    BSR ASC2HEX ;get word data to fill

BF_LOOP
    MOVE.W D0,(A3)+ ; store data in D0 at (A3), increment A3
    CMP.L A3,A4
    BGE BF_LOOP      ;continue loop while A3<=A4
BF_END
    BSR SUCCESS      ;print success message
    MOVEM.L (SP)+,A3-A4/D0-D1
    RTS
```

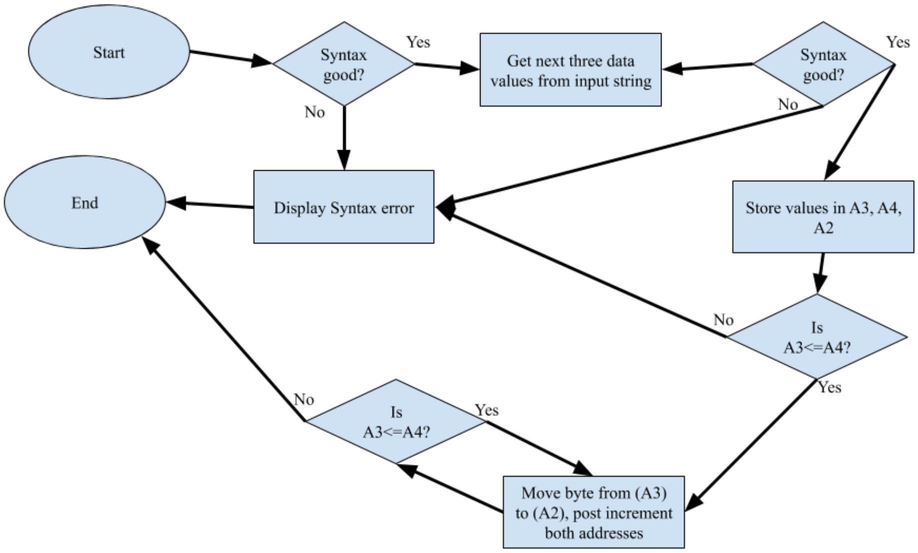
2.2.6. BMOV (Block Move)

This command is used to copy a block of data from one memory range to another location.
Syntax: BMOV <ADDRESS1> <ADDRESS2> <ADDRESS3>
The block of memory to be moved is from ADDRESS1 to ADDRESS2, and the data will be copied to the memory range starting at ADDRESS3



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->BMOV $8000 $8015 $80A2
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->
```

2.2.6.1. Algorithm and Flowchart


Start
Syntax good?
Yes
No
Get next three data values from input string
Syntax good?
Yes
No
End
Display Syntax error
Store values in A3, A4, A2
Is A3<=A4?
No
Yes
Is A3<=A4?
No
Yes
Move byte from (A3) to (A2), post increment both addresses

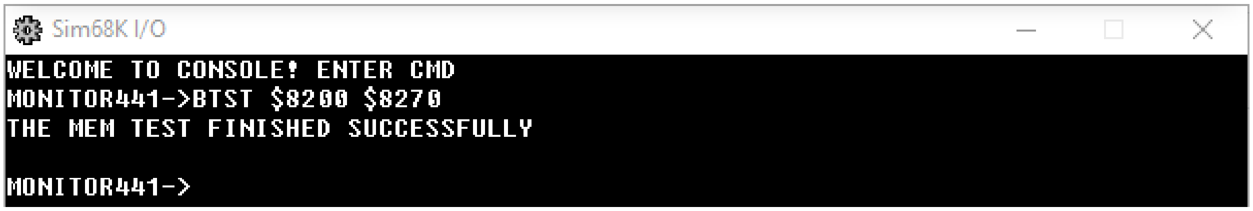
```
Start
    If syntax NOT good
        Display Syntax error
    Else
        Get next 3 data values (X,Y,Z)
    While (X<=Y)
        Copy contents of (X) to (Z)
        Increment (X)
End
```

2.2.6.2. Assembly Code

```
*------------BMOV-----------*
BMOV ; BMOV *START* *END* *NEW_START*

   BSR GET_VALUE
   MOVE.L D0,A3    ;store starting address in A3
   BSR GET_VALUE
   MOVE.L D0,A4    ;store ending address in A4
   BSR GET_VALUE
   MOVE.L D0,A2    ;store NEW starting address in A2

   CMPA.L A3,A4    ;make sure A4>A3
   BLE SYNTAX
BMOV_LOOP
   MOVE.B (A3)+,(A2)+  ;copy byte from (A3) to (A2) increment both values
   CMPA.L A3,A4
   BGE BMOV_LOOP       ;repeat loop while A4>A3
BMOV_END
   BSR SUCCESS         ;print success message
   RTS
```

2.2.7. BTST (Block Test)

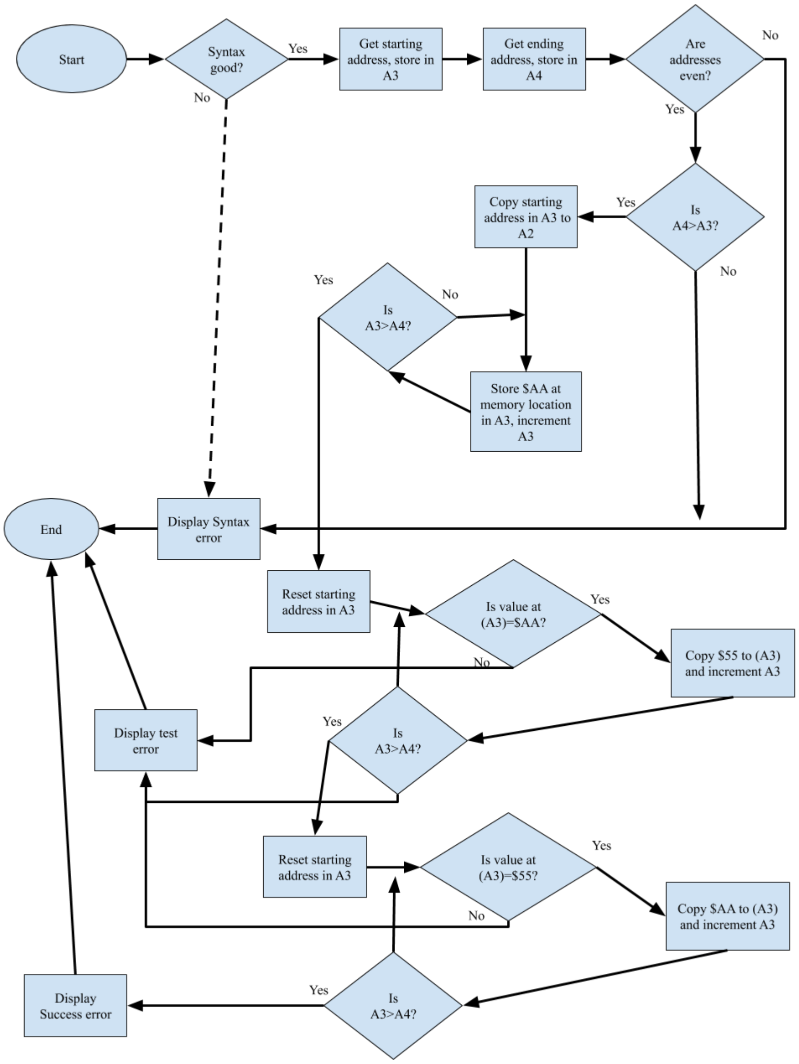
The BTST command is used to test a memory area by writing and reading values from it.
Syntax: BTST <START> <END>
This command will overwrite all memory locations being tested, including the starting and ending addresses.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->BTST $8200 $8270
THE MEM TEST FINISHED SUCCESSFULLY

MONITOR441->
```

Figure 6: BTST Sample Output

2.2.7.1. Algorithm and Flowchart


Start
Syntax good?
Yes
No
Get starting address, store in A3
Get ending address, store in A4
Are addresses even?
No
Yes
Is A4>A3?
Yes
No
Copy starting address in A3 to A2
Is A3>A4?
Yes
No
Store $AA at memory location in A3, increment A3
End
Display Syntax error
Reset starting address in A3
Is value at (A3)=$AA?
Yes
No
Copy $55 to (A3) and increment A3
Display test error
Is A3>A4?
Yes
Reset starting address in A3
Is value at (A3)=$55?
Yes
No
Copy $AA to (A3) and increment A3
Display Success error
Is A3>A4?
Yes

```
Start
    If syntax NOT good
        Display syntax error
    Else
        Get next 2 data values (X,Y)
        Copy starting address X to other register, Z
    While (X<=Y)
        Copy data A to (X)
        Increment X
    Copy Z to X
    While (X<=Y)
        If (X) not equal A
            Display test failed message
        Else
```

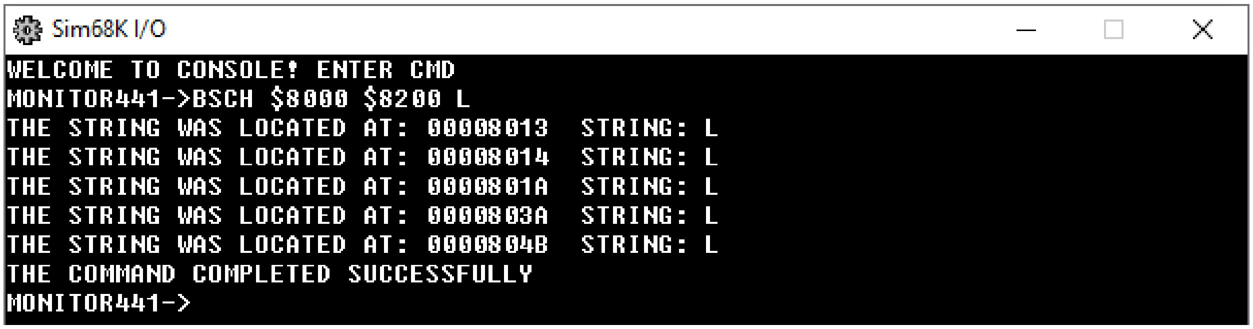
```
            Copy data B to (X)
            Increment X
    Copy Z to X
    While (X<=Y)
        If (X) not equal B
            Display test failed message
        Else
            Copy data A to (X)
            Increment X
    Display test passed message
End
```

2.2.7.2. Assembly Code

```
*-----------BTST-----------*
BTST ;A3 is lower address, A4 is upper address
   CLR.L D0
   CMP.B #SPACE,(A1)
   BNE SYNTAX
   ADDA.L #1,A1
   BSR ASC2HEX
   MOVE.L D0,A3
   CMP.B #SPACE,(A1)
   BNE SYNTAX
   ADDA.L #1,A1    ;get values
   BSR ASC2HEX
   MOVE.L D0,A4

   CMP.L A3,A4
   BLE SYNTAX
   MOVE.L A3,A2    ;copy starting address
BTST_STORE
   MOVE.B #$AA,(A3)+   ;move $AA to (A3), increment
A3
   CMP.L A3,A4     ;check if at end of range
   BGE BTST_STORE  ;continue until A3>A4
   MOVE.L A2,A3    ;reload starting address
   MOVE.B #$AA,D6  ;load EXPECTED value to D6
BTST_CHECK
   MOVE.B (A3),D5  ;load READ value to D5
   CMPI.B #$AA,D5
   BNE BTST_ERR
   MOVE.B #$55,(A3)+   ;move $55 to (A3), increment A3
   CMP.L A3,A4     ;check if at end of range
   BGE BTST_CHECK  ;continue checking until A3>A4
   MOVE.L A2,A3    ;reload starting address
   MOVE.B #$55,D6  ;load EXPECTED value to D6
BTST_CHECK2
   MOVE.B (A3),D5  ;load READ value to D5
   CMPI.B #$55,D5  ;compare READ value to expected
   BNE BTST_ERR
   MOVE.B #$AA,(A3)+  ;move $AA to (A3), increment
A3
   CMP.L A3,A4     ;check if at end of range
   BGE BTST_CHECK2
   BRA BTST_SUCC   ;if done, print success message
BTST_ERR
   LEA.L BTST_ERR_MSG1,A5
   LEA.L E_BTST_ERR_MSG1,A6
   MOVE.L A3,D0        ;get "Failed at" address
   MOVE.L #4,D1        ;add to end of message
   BSR HEX2ASC
   BSR PRINT           ;output 'failed at' message

   LEA.L BTST_ERR_MSG2,A5
   LEA.L E_BTST_ERR_MSG2,A6
   MOVE.B D6,D0        ;get value written
   MOVE.B #1,D1
   BSR HEX2ASC         ;output 'value written' string
   BSR PRINT

   LEA.L BTST_ERR_MSG3,A5
   LEA.L E_BTST_ERR_MSG3,A6
   MOVE.B D5,D0        ;get value read
   MOVE.B #1,D1
   BSR HEX2ASC         ;output 'value read' string
   BSR PRINT

   BRA BTST_END


BTST_SUCC
   LEA.L BTST_SUCC_MSG,A5
   LEA.L E_BTST_SUCC_MSG,A6
   BSR PRINT           ;print success message
BTST_END
   RTS         ;end command
```

2.2.8. BSCH (Block Search)

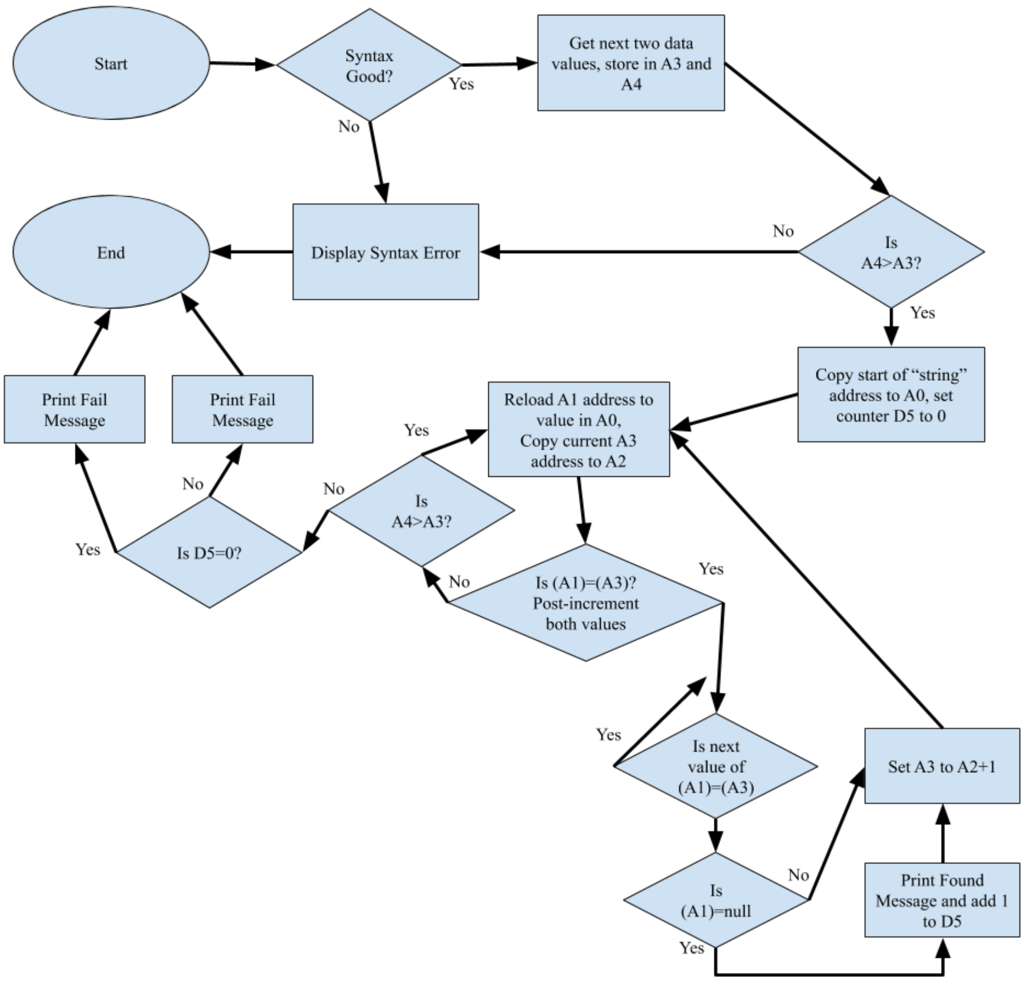
The BSCH command is used to search an area in memory for an ASCII string.
Syntax: BSCH <START> <END> <STRING>
This program will display the location of all instances of the string in the range. Only ASCII strings are allowed



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->BSCH $8000 $8200 L
THE STRING WAS LOCATED AT: 00008013  STRING: L
THE STRING WAS LOCATED AT: 00008014  STRING: L
THE STRING WAS LOCATED AT: 0000801A  STRING: L
THE STRING WAS LOCATED AT: 0000803A  STRING: L
THE STRING WAS LOCATED AT: 0000804B  STRING: L
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->
```

Figure 7: BSCH Sample Output

2.2.8.1. Algorithm and Flowchart


Start
Syntax Good?
Yes
No
Get next two data values, store in A3 and A4
End
Display Syntax Error
No
Is A4>A3?
Yes
Copy start of "string" address to A0, set counter D5 to 0
Print Fail Message
Print Fail Message
Reload A1 address to value in A0, Copy current A3 address to A2
Yes
No
No
Is A4>A3?
Yes
Is D5=0?
No
Is (A1)=(A3)? Post-increment both values
Yes
Yes
Is next value of (A1)=(A3)
Set A3 to A2+1
No
Is (A1)=null
Yes
Print Found Message and add 1 to D5

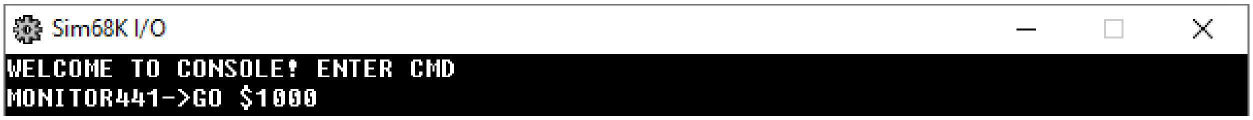
```
Start
    If syntax not good
        Display syntax error message
    Else
        Get next 3 data values (X,Y,Z)
    While (X<=Y)
        If (X)=Z
            Display string found message
            Increment found counter A
        Increment X
    If (counter A=0)
        Display failure message
End
```

2.2.8.2. Assembly Code

```
*-------------BSCH----------*

BSCH
   BSR GET_VALUE
   MOVE.L D0,A3    ;lower bound in A3
   BSR GET_VALUE
   MOVE.L D0,A4    ;upper bound in A4
   CMP.B #SPACE,(A1)+
   BNE SYNTAX
   MOVE.L A1,A0    ;copy start of string address to A0
   MOVE.L #0,D5
BSCH_LOOP
   MOVE.L A0,A1    ;reload string address
   MOVE.L A3,A2    ;copy current address to A2
   CMP.B (A1)+,(A3)+ ;check next memory location
   BEQ BSCH_FIND
   CMP.L A3,A4     ;check if at end of range
   BGE BSCH_LOOP   ;if not at end, repeat loop
   BRA BSCH_DONE   ;if at end, branch to BSCH_DONE

BSCH_FIND
   CMP.B (A1)+,(A3)+   ;check if next value is correct
   BEQ BSCH_FIND       ;repeat if same
   CMP.B #NULL,-(A1)    ;if not same, check if at end of
string
   BNE BSCH_CONTINUE   ;if not, continue searchign

   ADDI.L #1,D5        ;add to strings found counter
   LEA.L BSCH_SUCC_MSG,A5
   LEA.L E_BSCH_SUCC_MSG,A6
   BSR PRINT_NC
   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.L A2,D0
   MOVE.L #4,D1        ;output found msg
   BSR D0OUT
   LEA.L BSCH_STRING,A5
   LEA.L E_BSCH_STRING,A6  ; output 'STRING:'
   BSR PRINT_NC

   MOVE.L #13,D0
   MOVE.L A0,A1        ;output string that was found
   TRAP #15

BSCH_CONTINUE
   MOVE.L A2,A3
   ADDA.L #1,A3        ;reload next memory address to
search
   BRA BSCH_LOOP       ;branch to loop

BSCH_DONE
   CMP.L #0,D5         ;if no strings found, branch to fail
   BEQ BSCH_FAIL
   BSR SUCCESS         ;print success msg
   RTS
BSCH_FAIL
   LEA.L BSCH_FAIL_MSG,A5
   LEA.L E_BSCH_FAIL_MSG,A6
   BSR PRINT           ;display fail msg
   RTS
```

2.2.9. GO (Execute Program)

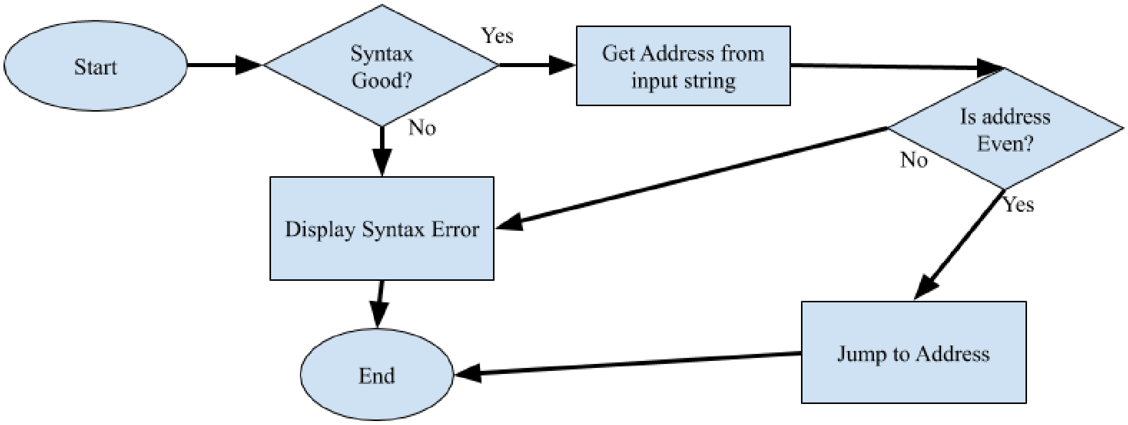
The GO Command is used to execute a program stored in memory at a specified location.
Syntax: GO <ADDRESS>
The starting address specified in the command must be even. Otherwise a syntax error will occur.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->GO $1000
```

Figure 8: GO Sample Output

2.2.9.1. Algorithm and Flowchart



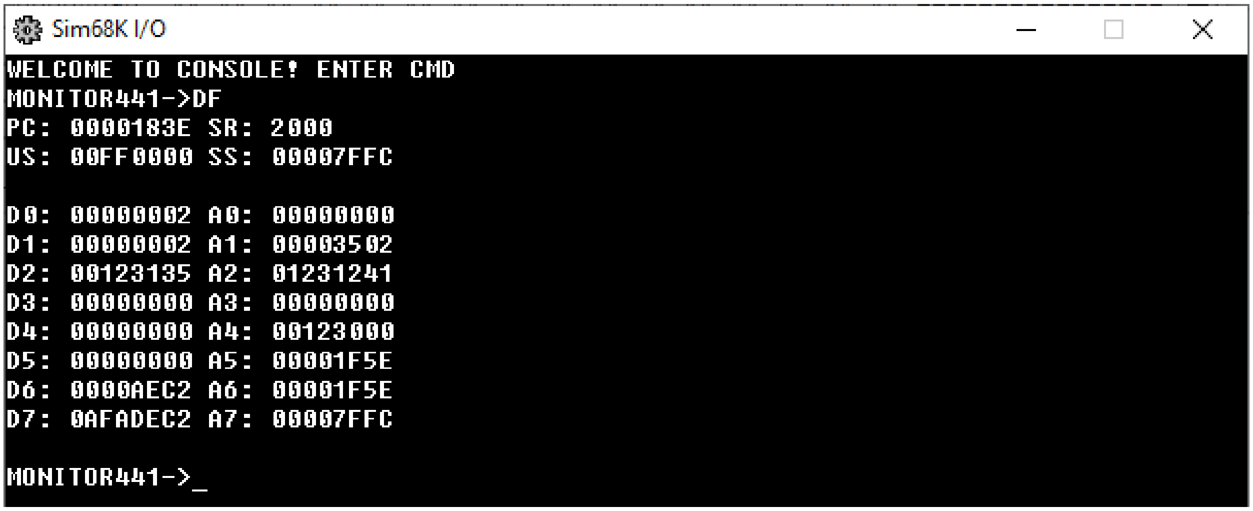
```
Start
    If syntax not good
        Display syntax error message
    Else
        Get input X
    If X is even
        Branch to X address
    Else
        Display syntax error
End
```

2.2.9.2. Assembly Code

```
GO
   CMP.B #SPACE, (A1)+  ; check syntax
   BNE SYNTAX
   BSR ASC2HEX          ;get memory address
   MOVE.L D0,A3         ;copy address to A3
   JMP (0,A3)           ;jump to A3 address
```

2.2.10. DF (Display Formatted Registers)

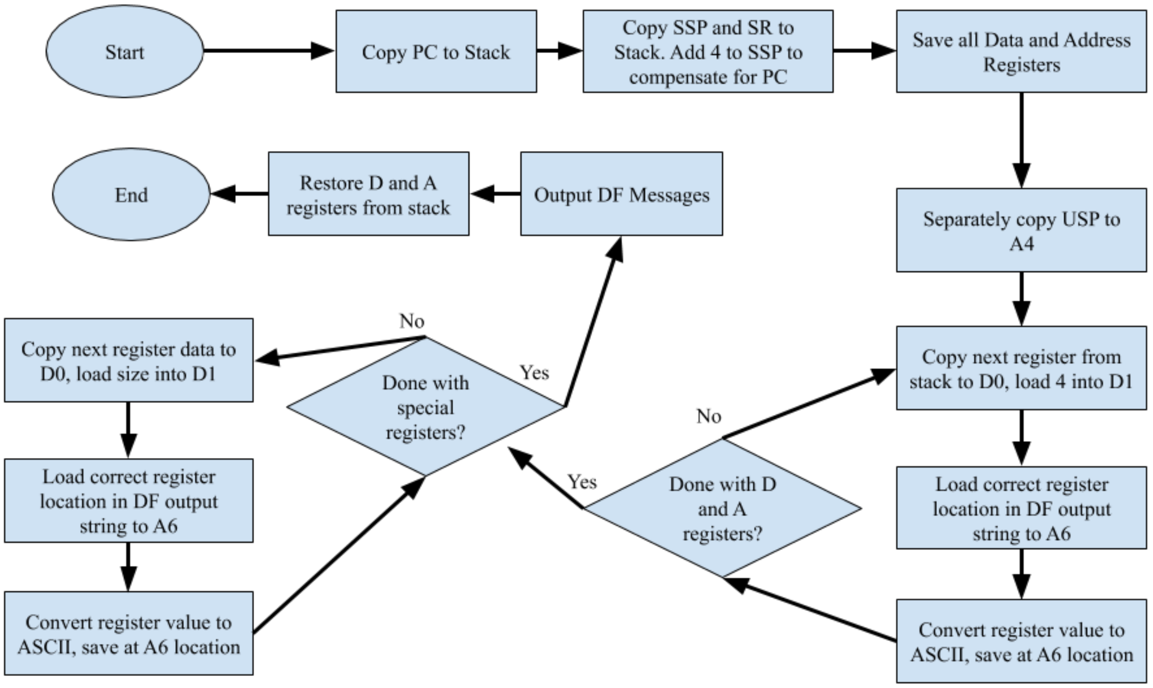
This command is used to display the current contents of system registers in a formatted manner.
Syntax: DF
This command will output all Data and Address registers, along with the current PC value, Status Register, User stack pointer, and Supervisor stack pointer.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->DF
PC: 0000183E SR: 2000
US: 00FF0000 SS: 00007FFC

D0: 00000002 A0: 00000000
D1: 00000002 A1: 00003502
D2: 00123135 A2: 01231241
D3: 00000000 A3: 00000000
D4: 00000000 A4: 00123000
D5: 00000000 A5: 00001F5E
D6: 0000AEC2 A6: 00001F5E
D7: 0AFADEC2 A7: 00007FFC

MONITOR441->_
```

Figure 9: DF Sample Output

2.2.10.1. Algorithm and Flowchart


Start
Copy PC to Stack
Copy SSP and SR to Stack. Add 4 to SSP to compensate for PC
Save all Data and Address Registers
Separately copy USP to A4
Copy next register from stack to D0, load 4 into D1
Load correct register location in DF output string to A6
Convert register value to ASCII, save at A6 location
Done with D and A registers?
No
Yes
Done with special registers?
No
Yes
Copy next register data to D0, load size into D1
Load correct register location in DF output string to A6
Convert register value to ASCII, save at A6 location
Output DF Messages
Restore D and A registers from stack
End

```
Start
        Copy sensitive values to Stack
        Copy All data and Memory registers
        Compensate for SP values that was changed during store
        Convert stored values from stack into ASCII
        Store ASCII values in corresponding output string locations
        Output strings
End
```

2.2.10.2. Assembly Code

```
DF
   PEA.L *(PC)          ;save PC
   MOVE.L SP, -(SP)    ;Save SSP
   ADD.L #4,(SP)  ;correct SSP value
   MOVE.W SR, -(SP)    ;save SR


   MOVEM.L A0-A7/D0-D7,-(SP)   ;save all D and A
address registers
   MOVE USP,A4 ;save USP

   LEA.L DF_D0,A6 ;load Data reg data
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D1,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D2,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D3,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D4,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D5,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D6,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_D7,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
               ;load Address reg data
   LEA.L DF_A0,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A1,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A2,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A3,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A4,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A5,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A6,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC
   LEA.L DF_A7,A6
   MOVE.L (SP)+,D0
   MOVE.B #4,D1
   BSR HEX2ASC


   MOVE.W (SP)+,D0
   MOVE.B #2,D1    ;LOAD SR DATA
   LEA.L DF_SR,A6
   BSR HEX2ASC

   MOVE.L A4,D0
   MOVE.B #4,D1
   LEA.L DF_US,A6  ;LOAD USP DATA
   BSR HEX2ASC

   MOVE.L (SP)+,D0
   MOVE.L D0,D3
   MOVE.B #4,D1
   LEA.L DF_SS,A6  ;LOAD SSP DATA
   BSR HEX2ASC
```

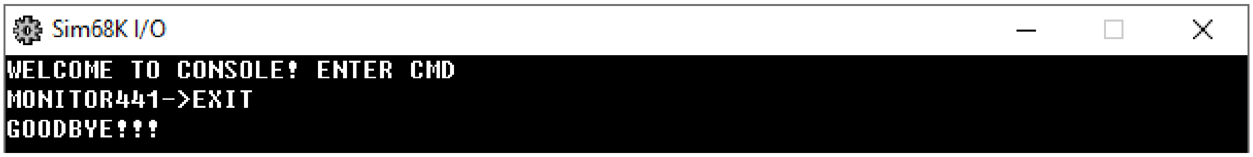
```
 LEA.L DF_A7,A6
 MOVE.L #4,D1
 MOVE.L D3,D0
 BSR HEX2ASC     ;FIX A7 DATA (=to SSP)

 MOVE.L (SP)+,D0
 MOVE.B #4,D1
 LEA.L DF_PC,A6  ;LOAD PC DATA
 BSR HEX2ASC

 LEA.L DF_OUT2,A5
 LEA.L E_DF_OUT2,A6  ;output message 1
 BSR PRINT

 LEA.L DF_OUTPUT,A5
 LEA.L E_DF_OUTPUT,A6    ;output message 2
 BSR PRINT
 SUB.L #74,SP
 MOVEM.L (SP)+,D0-D7/A0-A7
 ADD.L #10,SP
DF_END
 RTS
```

2.2.11. EXIT (Exit Monitor Program)

The EXIT command is used to terminate the monitor program.

Syntax: EXIT

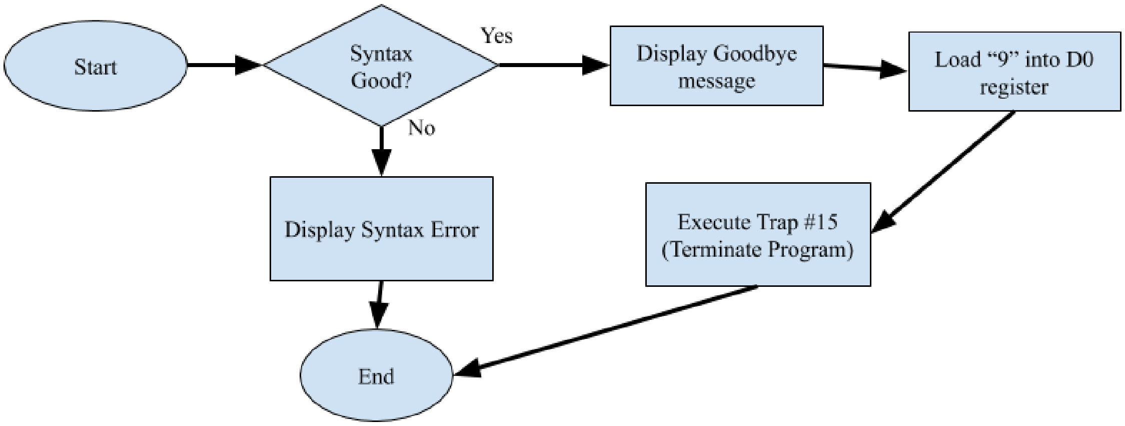
If any data is added after the initial exit command, a syntax error will occur.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->EXIT
GOODBYE!!!
```

Figure 10: EXIT Sample Output

2.2.11.1. Algorithm and Flowchart



```
Start
    If syntax not good
        Display syntax error message
    Else
        Display goodbye message
        Load data A into D0 register
        Execute trap #15

End
```

2.2.11.2. Assembly Code

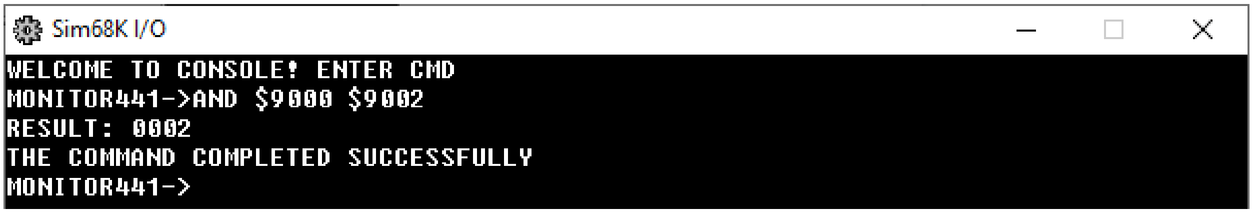
```
*-----------EXIT-----------*
EXIT ; used to stop monitor program
    CMP.B #NULL,(A1)    ; if anything entered after 'EXIT', invoke syntax error
    BNE SYNTAX
    LEA.L GOODBYE,A5
    LEA.L E_GOODBYE,A6  ;output goodbye message
    BSR PRINT
    MOVE.L #9,D0        ;execute trap 15 #9,(terminate program)
    TRAP #15
```

2.2.12. AND (Logical AND)

This command performs the logical AND operation on two word sized values stored in memory.

Syntax: AND <ADDRESS1> <ADDRESS2>

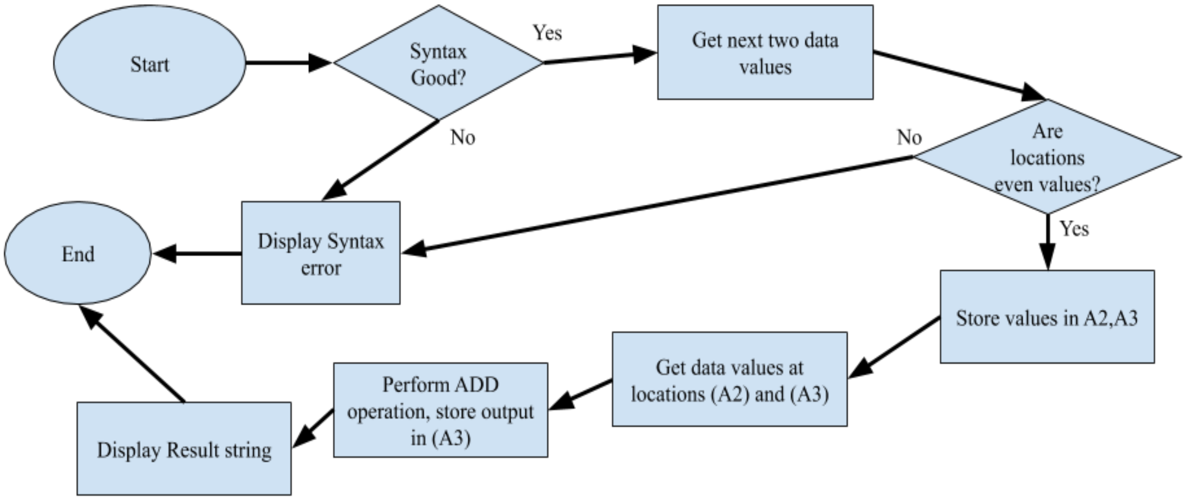
The result will be stored at the second address location.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->AND $9000 $9002
RESULT: 0002
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->
```

Figure 11: AND Sample Output

2.2.12.1. Algorithm and Flowchart



```
Start
        If syntax not good
                Display syntax error message
        Else
                Get next 2 data values (X,Y)
        If (X,Y) not even
                Display syntax error
        Else
                Get data A from memory location X
                Get data B from memory location Y
        AND data A and B to get data C
        Display data C to terminal
        Save data C at memory location Y
End
```

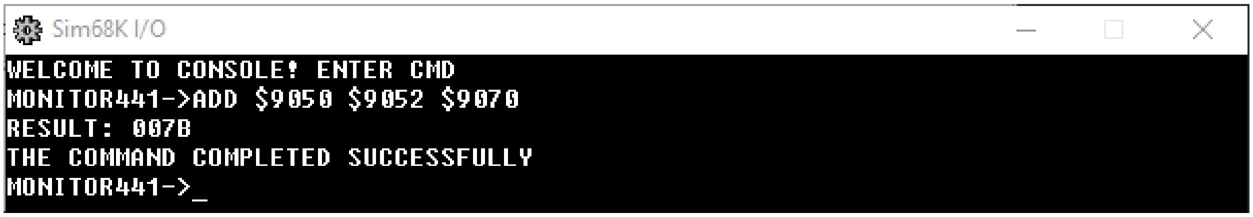
2.2.12.2. Assembly Code

```
*------------AND-------------*
;conduct logical memory AND operation on two memory locations of word size
AND ; AND *ADDRESS1* ADDRESS2* store result in address 2

    BSR GET_VALUE
   BTST #0,D0
   BNE SYNTAX
   MOVE.L D0,A2    ;store 1st value in A2
   BSR GET_VALUE
   BTST #0,D0
   BNE SYNTAX
   MOVE.L D0,A3    ;store 2nd value in A3

   MOVE.W (A2),D2
   MOVE.W (A3),D3
   AND.W   D2,D3   ;AND the values
   MOVE.W D3,(A3)  ; Store at 2nd memory location
   LEA.L RESULT_MSG,A5
   LEA.L E_RESULT_MSG,A6
   BSR PRINT_NC

   MOVE.W D3,D0
   MOVE.L #2,D1
   BSR D0OUT   ;output result
   BSR PRINT

   BSR SUCCESS
   RTS
```

2.2.13. ADD (Add Memory)

This command will perform Addition on two word sized values stored in memory. The result will be stored at the third designated address.

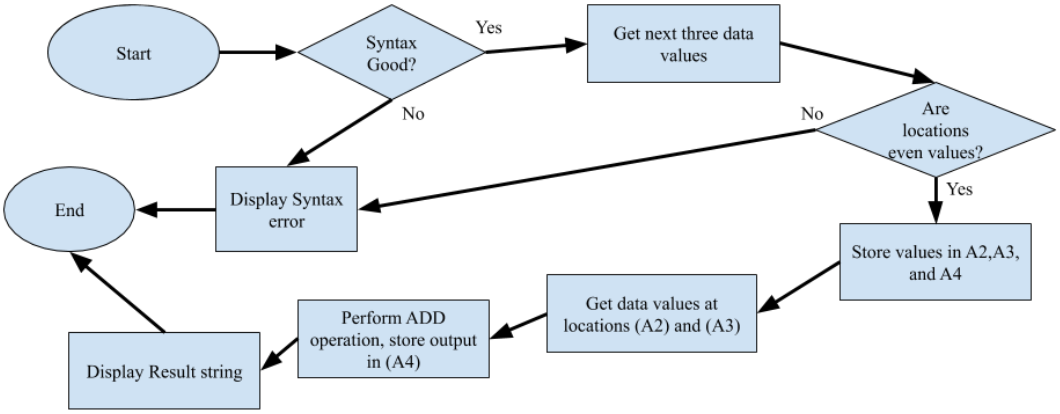
Syntax: ADD <ADDRESS1> <ADDRESS2> <ADDRESS3>



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->ADD $9050 $9052 $9070
RESULT: 007B
THE COMMAND COMPLETED SUCCESSFULLY
MONITOR441->_
```

Figure 12: ADD Sample Output

2.2.13.1. Algorithm and Flowchart


Start
Syntax Good?
Yes
Get next three data values
No
No
Are locations even values?
Yes
End
Display Syntax error
Store values in A2,A3, and A4
Get data values at locations (A2) and (A3)
Perform ADD operation, store output in (A4)
Display Result string

```
Start
    If syntax not good
        Display syntax error message
    Else
        Get next three data values (X,Y,Z)
    If (X,Y,Z) not even
        Display syntax error message
    Else
        Get data A from memory location X
        Get data B from memory location Y
    Add data A and B to get data C
    Output data C to terminal
    Save data C at memory location Z
End
```

2.2.13.2. Assembly Code

```
*-----------ADD------------*
; ADD WORD SIZED VALUES, STORE AT 3rd LOCATION
;   ADD <ADDR1> <ADDR2> <ADDR3>
ADD
```

```
BSR GET_VALUE
BTST #0,D0
BNE SYNTAX
MOVE.L D0,A2    ;store 1st value in A2
BSR GET_VALUE
BTST #0,D0
BNE SYNTAX
MOVE.L D0,A3    ;store 2nd value in A3
BSR GET_VALUE
BTST #0,D0
BNE SYNTAX
MOVE.L D0,A4    ;store 3rd value in A4

MOVE.W (A2),D2
MOVE.W (A3),D3
ADD.W   D2,D3   ;Add values together
MOVE.W D3,(A4)  ;store at 3rd address
LEA.L RESULT_MSG,A5
LEA.L E_RESULT_MSG,A6
BSR PRINT_NC

MOVE.W D3,D0
MOVE.L #2,D1
BSR D0OUT   ;output result
BSR PRINT

BSR SUCCESS
RTS
```

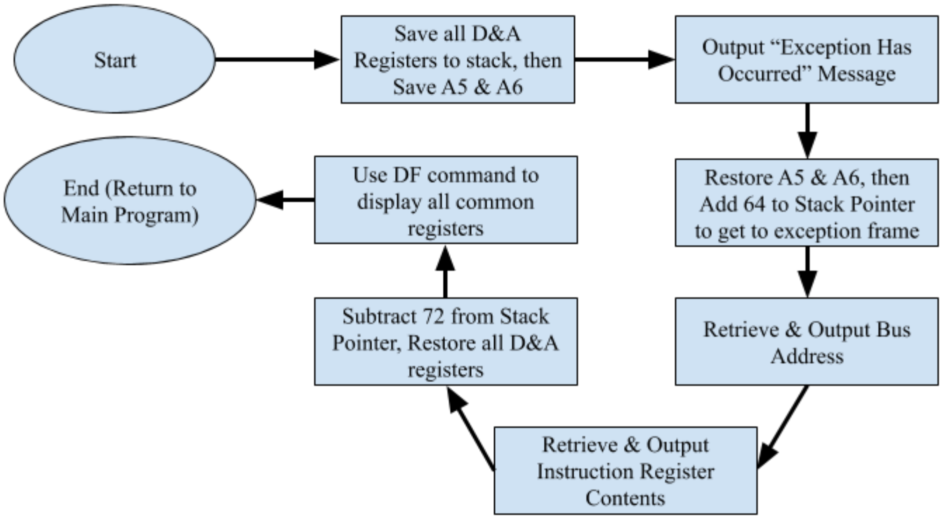
2.3. Exception Handlers

The exception handlers included in the monitor program will allow the system to recover from unanticipated errors during program execution. They will output useful debugging information and return the user to the command interpreter

2.3.1. Bus Error Exception

This exception handler will help the monitor program recover from bus errors. If a bus error is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.1.1. Algorithm and Flowchart


Start
Save all D&A Registers to stack, then Save A5 & A6
Output "Exception Has Occurred" Message
Restore A5 & A6, then Add 64 to Stack Pointer to get to exception frame
Retrieve & Output Bus Address
Retrieve & Output Instruction Register Contents
Subtract 72 from Stack Pointer, Restore all D&A registers
Use DF command to display all common registers
End (Return to Main Program)

```
Start
If Exception has occurred
        Save All D&A Registers
                Save A5/A6 Registers
                        Output "Exception Has Occurred" message
                Restore A5/A6 Registers
                Output Bus Address and Instruction Register contents
        Restore All D&A Registers
        Display Formatted Registers
        Return to Main Program
End
```

2.3.1.2. Assembly Code

```
BUS_EXC
   MOVEM.L D0-D7/A0-A7,-(SP)   ;save all registers
   MOVEM.L A5/A6,-(SP)     ;save A5/A6 reg
   MOVE SR,D4
   LEA.L BUS_EXC_MSG,A5
   LEA.L E_BUS_EXC_MSG,A6  ;print out error msg
   BSR PRINT_NC
BUS_ADDR_COMMON
```

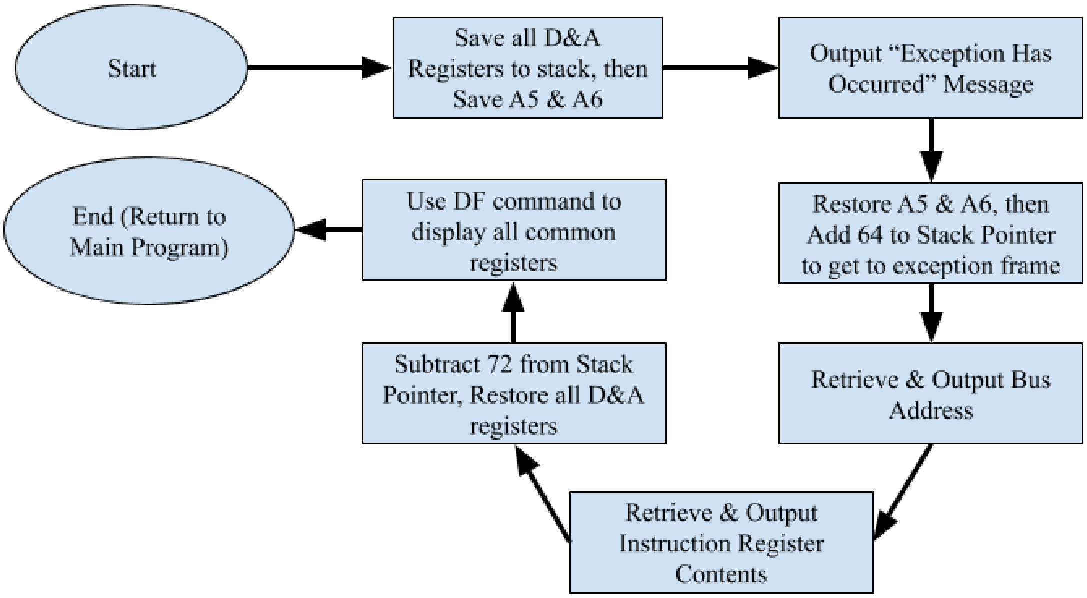
```
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6
   BSR PRINT            ;print out "has occurred"
   MOVEM.L (SP)+,A5/A6     ;restore A5/A6 reg

   ADDA.L #64,SP         ;add 64 to stack pointer to get
past saved reg
   MOVE.W (SP)+,D4        ;Save next word to D4 (SSW)
   MOVE.L (SP)+,D5        ;save next long to D5 (bus
address)
   MOVE.W (SP)+,D3        ;save next word to D3 (instr.
reg)
   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$42,(A6)+      ; ASCII "B"
   MOVE.B #$41,(A6)+      ; ASCII "A"
   MOVE.B #$3D,(A6)+      ; ASCII "="     bus address=
   MOVE.L D5,D0           ;move bus address to D0
   MOVE.L #4,D1
   BSR HEX2ASC
   BSR PRINT_NC           ;output Bus address

   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$20,(A6)+      ; ASCII *space*
   MOVE.B #$49,(A6)+      ; ASCII "I"
   MOVE.B #$52,(A6)+      ; ASCII "R"
   MOVE.B #$3D,(A6)+      ; ASCII "=" Instruction Reg=
   MOVE.W D3,D0           ; move IR value to D0
```

```
   MOVE.L #2,D1
   BSR HEX2ASC
   BSR PRINT_NC           ;output instruction reg

   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$53,(A6)+      ;ASCII "S"
   MOVE.B #$53,(A6)+      ;ASCII "S"
   MOVE.B #$57,(A6)+      ;ASCII "W"
   MOVE.B #$3D,(A6)+      ;ASCII "="
   MOVE.W D4,D0
   MOVE.L #2,D1           ;output SSW value
   BSR HEX2ASC
   BSR PRINT

   SUBA.L #72,SP          ;subtract 72 from stack pointer
   MOVEM.L D0-D7/A0-A7,-(SP)   ;restore all data/addr
reg
   BSR DF                 ;print out all registers
   BRA MAINP
```

2.3.2. Address Error Exception

This exception handler will help the monitor program recover from address errors. If an address error is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.2.1. Algorithm and Flowchart


Start
Save all D&A Registers to stack, then Save A5 & A6
Output "Exception Has Occurred" Message
Restore A5 & A6, then Add 64 to Stack Pointer to get to exception frame
Retrieve & Output Bus Address
Retrieve & Output Instruction Register Contents
Subtract 72 from Stack Pointer, Restore all D&A registers
Use DF command to display all common registers
End (Return to Main Program)

```
Start
If Exception has occurred
    Save All D&A Registers
        Save A5/A6 Registers
            Output "Exception Has Occurred" message
        Restore A5/A6 Registers
        Output Bus Address and Instruction Register contents
    Restore All D&A Registers
    Display Formatted Registers
    Return to Main Program
End
```

2.3.2.2. Assembly Code

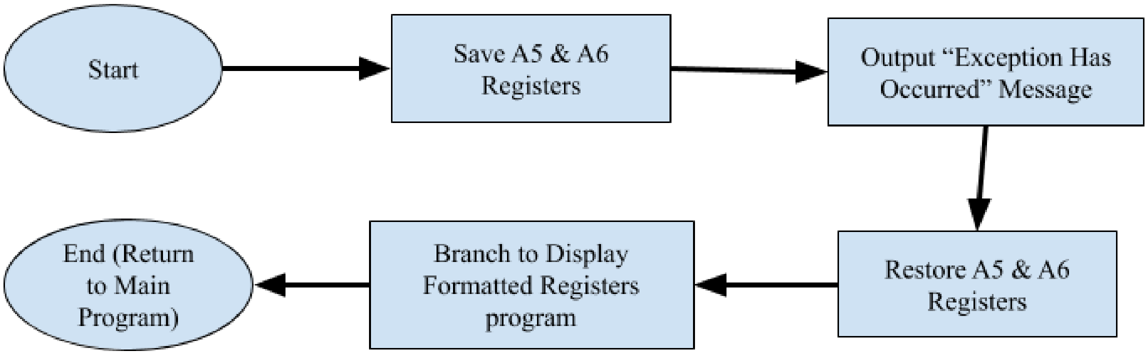
```
BUS_EXC
   MOVEM.L D0-D7/A0-A7,-(SP)   ;save all registers
   MOVEM.L A5/A6,-(SP)     ;save A5/A6 reg
   MOVE SR,D4
   LEA.L BUS_EXC_MSG,A5
   LEA.L E_BUS_EXC_MSG,A6  ;print out error msg
   BSR PRINT_NC
BUS_ADDR_COMMON
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6
   BSR PRINT              ;print out "has occurred"
   MOVEM.L (SP)+,A5/A6     ;restore A5/A6 reg

   ADDA.L #64,SP          ;add 64 to stack pointer to get
past saved reg
   MOVE.W (SP)+,D4        ;Save next word to D4 (SSW)
   MOVE.L (SP)+,D5        ;save next long to D5 (bus
address)
   MOVE.W (SP)+,D3        ;save next word to D3 (instr.
reg)
   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$42,(A6)+       ; ASCII "B"
   MOVE.B #$41,(A6)+       ; ASCII "A"
   MOVE.B #$3D,(A6)+       ; ASCII "="    bus address=
   MOVE.L D5,D0           ;move bus address to D0
   MOVE.L #4,D1
   BSR HEX2ASC
   BSR PRINT_NC           ;output Bus address
```

```
   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$20,(A6)+       ; ASCII *space*
   MOVE.B #$49,(A6)+       ; ASCII "I"
   MOVE.B #$52,(A6)+       ; ASCII "R"
   MOVE.B #$3D,(A6)+       ; ASCII "=" Instruction Reg=
   MOVE.W D3,D0           ; move IR value to D0
   MOVE.L #2,D1
   BSR HEX2ASC
   BSR PRINT_NC            ;output instruction reg

   LEA.L OUTPUT_SPC,A5
   LEA.L OUTPUT_SPC,A6
   MOVE.B #$53,(A6)+       ;ASCII "S"
   MOVE.B #$53,(A6)+       ;ASCII "S"
   MOVE.B #$57,(A6)+       ;ASCII "W"
   MOVE.B #$3D,(A6)+       ;ASCII "="
   MOVE.W D4,D0
   MOVE.L #2,D1           ;output SSW value
   BSR HEX2ASC
   BSR PRINT

   SUBA.L #72,SP          ;subtract 72 from stack pointer
   MOVEM.L D0-D7/A0-A7,-(SP)   ;restore all data/addr
reg
   BSR DF                 ;print out all registers
   BRA MAINP
```

2.3.3. Illegal Instruction Exception

This exception handler will help the monitor program recover from illegal instruction errors. If an illegal instruction error is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.3.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output "Exception Has Occurred" Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

```
*Start*
*If Exception has occurred*
    *Save A5/A6 Registers*
    *Output "Exception Has Occurred" message*
    *Restore A5/A6 Registers*
    *Display Formatted Registers*
    *Return to Main Program*
*End*
```

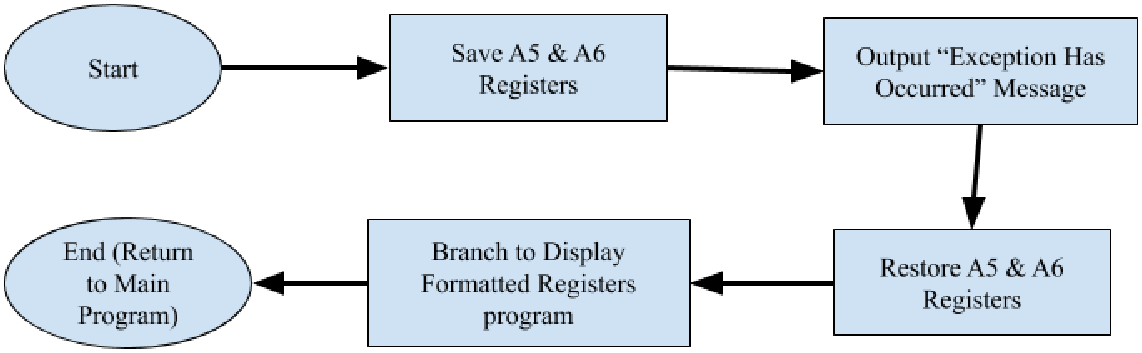
2.3.3.2. Assembly Code

```
ILL_INSTR_EXC
   MOVEM.L A5/A6,-(SP)     ;save A5/A6 reg
   LEA.L ILL_INSTR_EXC_MSG,A5
   LEA.L E_ILL_INSTR_EXC_MSG,A6    ;print out error msg
   BSR PRINT_NC
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6
   BSR PRINT             ;print out "has occurred"
   MOVEM.L (SP)+,A5/A6     ;restore A5/A6 reg
   BSR DF               ;print out all registers
   BRA MAINP
```

2.3.4. Privilege Violation Exception

This exception handler will help the monitor program recover from privilege violation errors. If a privilege violation error is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.4.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output "Exception Has Occurred" Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

```
Start
If Exception has occurred
    Save A5/A6 Registers
    Output "Exception Has Occurred" message
    Restore A5/A6 Registers
    Display Formatted Registers
    Return to Main Program
End
```

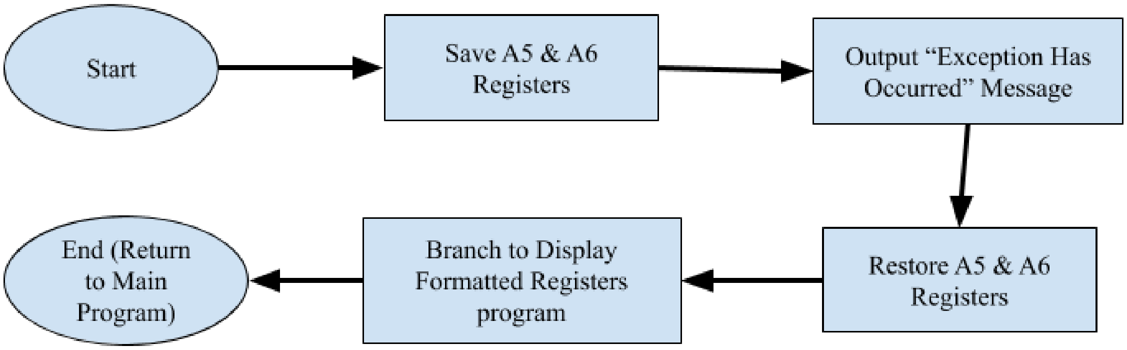
2.3.4.2. Assembly Code

```
PRIV_VIOL_EXC
   MOVEM.L A5/A6,-(SP)        ;save A5/A6 reg
   LEA.L PRIV_VIOL_EXC_MSG,A5
   LEA.L E_PRIV_VIOL_EXC_MSG,A6    ;print out error msg
   BSR PRINT_NC
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6
   BSR PRINT             ;print out "has occurred"
   MOVEM.L (SP)+,A5/A6     ;restore A5/A6 reg
   BSR DF               ;print out all registers
   BRA MAINP
```

2.3.5. Divide by Zero Exception

This exception handler will help the monitor program recover from divide by zero errors. If an attempt to divide by zero is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.5.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output "Exception Has Occurred" Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

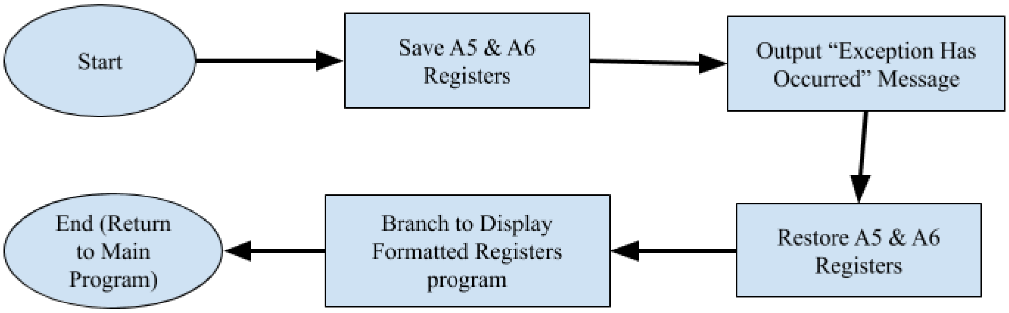
```
Start
If Exception has occurred
        Save A5/A6 Registers
        Output "Exception Has Occurred" message
        Restore A5/A6 Registers
        Display Formatted Registers
        Return to Main Program
End
```

2.3.5.2. Assembly Code

```
DIV_ZERO_EXC
  MOVEM.L A5/A6,-(SP)     ;save A5/A6 reg
  LEA.L DIV_ZERO_EXC_MSG,A5
  LEA.L E_DIV_ZERO_EXC_MSG,A6 ;print out error msg
  BSR PRINT_NC
  LEA.L HAS_OCCURRED,A5
  LEA.L E_HAS_OCCURRED,A6  ;print out "has occurred"
  BSR PRINT
  MOVEM.L (SP)+,A5/A6     ;restore A5/A6 reg
  BSR DF               ;print out all registers
  BRA MAINP
```

2.3.6. CHK Instruction Exception

2.3.6.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output "Exception Has Occurred" Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

```
Start
If Exception has occurred
        Save A5/A6 Registers
        Output "Exception Has Occurred" message
        Restore A5/A6 Registers
        Display Formatted Registers
        Return to Main Program
End
```

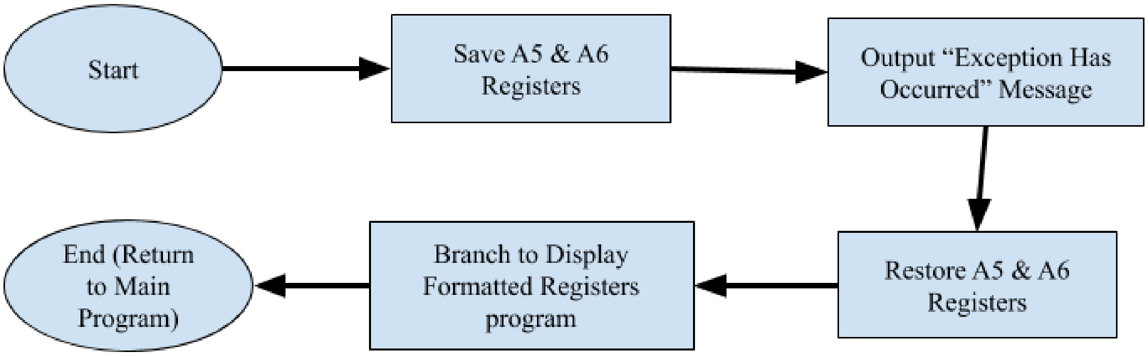
2.3.6.2. Assembly Code

```
CHK_INSTR_EXC
   MOVEM.L A5/A6,-(SP)     ;save A5/A6 reg
   LEA.L CHK_INSTR_EXC_MSG,A5
   LEA.L E_CHK_INSTR_EXC_MSG,A6    ;print out error msg
   BSR PRINT_NC
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6      ;print out "has occurred"

   BSR PRINT
   MOVEM.L (SP)+,A5/A6        ;restore A5/A6 reg
   BSR DF                  ;print out all registers
   BRA MAINP
```

2.3.7. Line A Emulator Exception

This exception handler will help the monitor program recover from Line A Emulator errors. If an attempt to use an instruction starting with the hex value “A” is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.7.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output “Exception Has Occurred” Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

```
Start
If Exception has occurred
    Save A5/A6 Registers
    Output “Exception Has Occurred” message
    Restore A5/A6 Registers
    Display Formatted Registers
    Return to Main Program
End
```

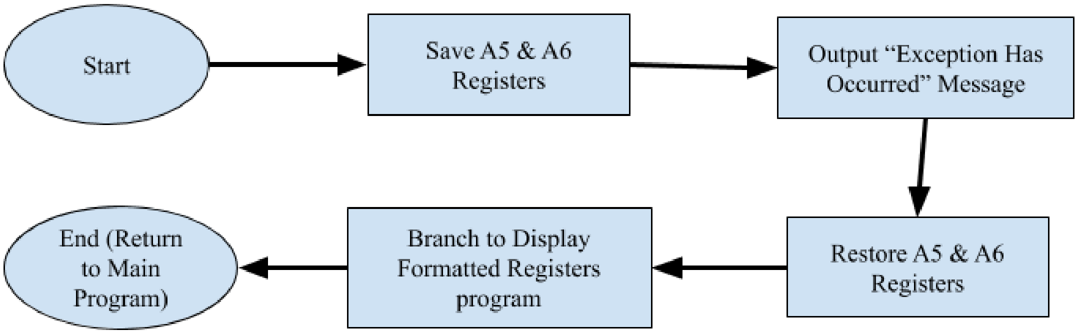
2.3.7.2. Assembly Code

```
LINE_A_EXC
   MOVEM.L A5/A6,-(SP)        ;save A5/A6 reg
   LEA.L LINE_A_EXC_MSG,A5
   LEA.L E_LINE_A_EXC_MSG,A6   ;print out error msg
   BSR PRINT_NC
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6      ;print out "has occurred"
   BSR PRINT
   MOVEM.L (SP)+,A5/A6        ;restore A5/A6 reg
   BSR DF                   ;print out all registers
   BRA MAINP
```

2.3.8. Line F Emulator Exception

This exception handler will help the monitor program recover from Line F Emulator errors. If an attempt to use an instruction starting with the hex value “F” is detected by the MC68000 microprocessor, the processor will branch to this exception routine which displays information helpful in debugging the error. After displaying information, the handler will return control to the command interpreter.

2.3.8.1. Algorithm and Flowchart


Start
Save A5 & A6 Registers
Output “Exception Has Occurred” Message
Restore A5 & A6 Registers
Branch to Display Formatted Registers program
End (Return to Main Program)

```
Start
If Exception has occurred
        Save A5/A6 Registers
        Output “Exception Has Occurred” message
        Restore A5/A6 Registers
        Display Formatted Registers
        Return to Main Program
End
```

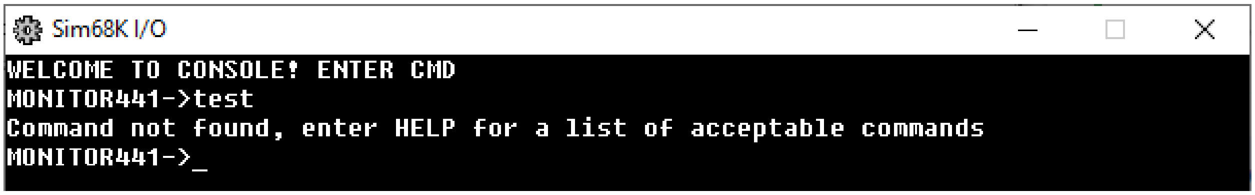
2.3.8.2. Assembly Code

```
LINE_F_EXC
   MOVEM.L A5/A6,-(SP)         ;save A5/A6 reg
   LEA.L LINE_F_EXC_MSG,A5
   LEA.L E_LINE_F_EXC_MSG,A6   ;print out error msg
   BSR PRINT_NC
   LEA.L HAS_OCCURRED,A5
   LEA.L E_HAS_OCCURRED,A6     ;print out "has occurred"
   BSR PRINT
   MOVEM.L (SP)+,A5/A6         ;restore A5/A6 reg
   BSR DF                      ;print out all registers
   BRA MAINP
```

2.4. User Instruction Manual

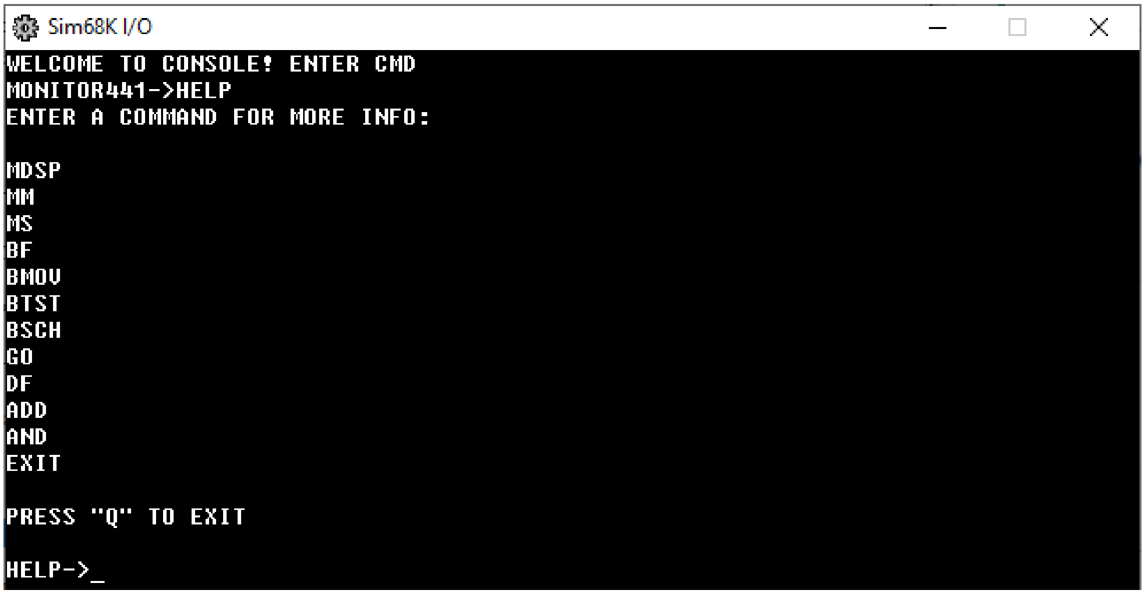
2.4.1. Help Menu

Provided in the monitor program is a comprehensive help environment to improve the user experience and provide the correct syntax for using debugger commands. Any time an incorrect command is entered by the user, a short message will be displayed, explaining to the user the existence of the “HELP” command. This will prevent the user from being stuck due to using incorrect syntax.

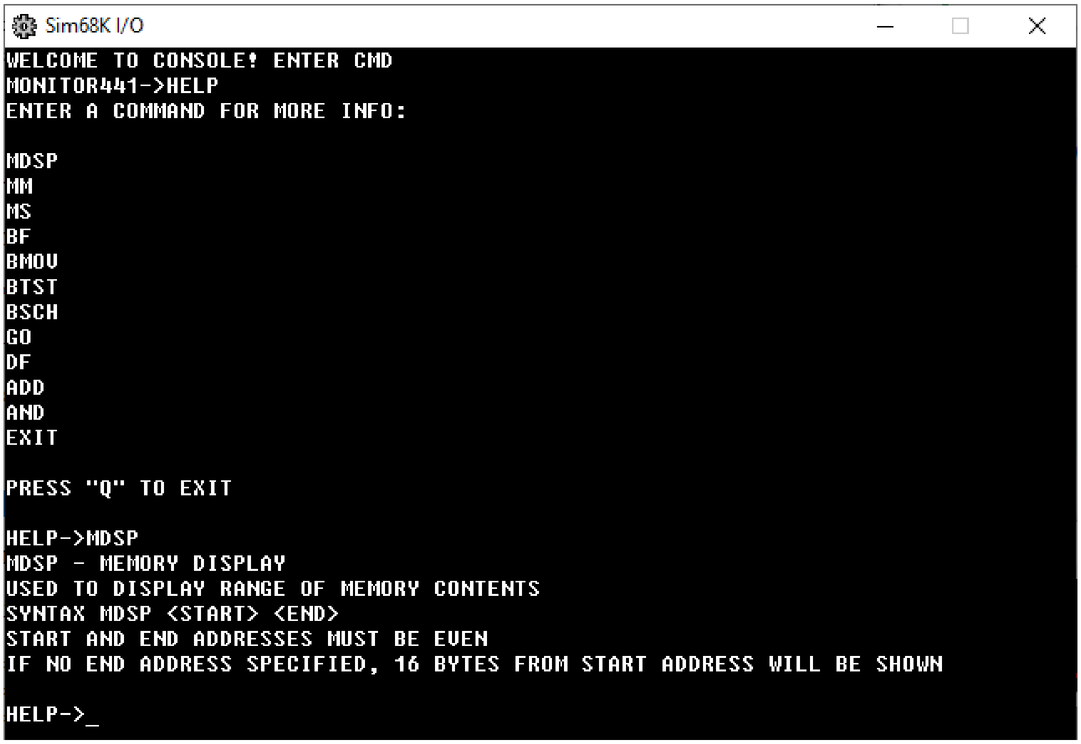


```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->test
Command not found, enter HELP for a list of acceptable commands
MONITOR441->_
```

Once in the HELP command, a new monitor is utilized, designated by the “HELP->” cursor being displayed in the terminal window. In this environment, the user may enter any supported debugger command and receive detailed information about the syntax and utility of the command.

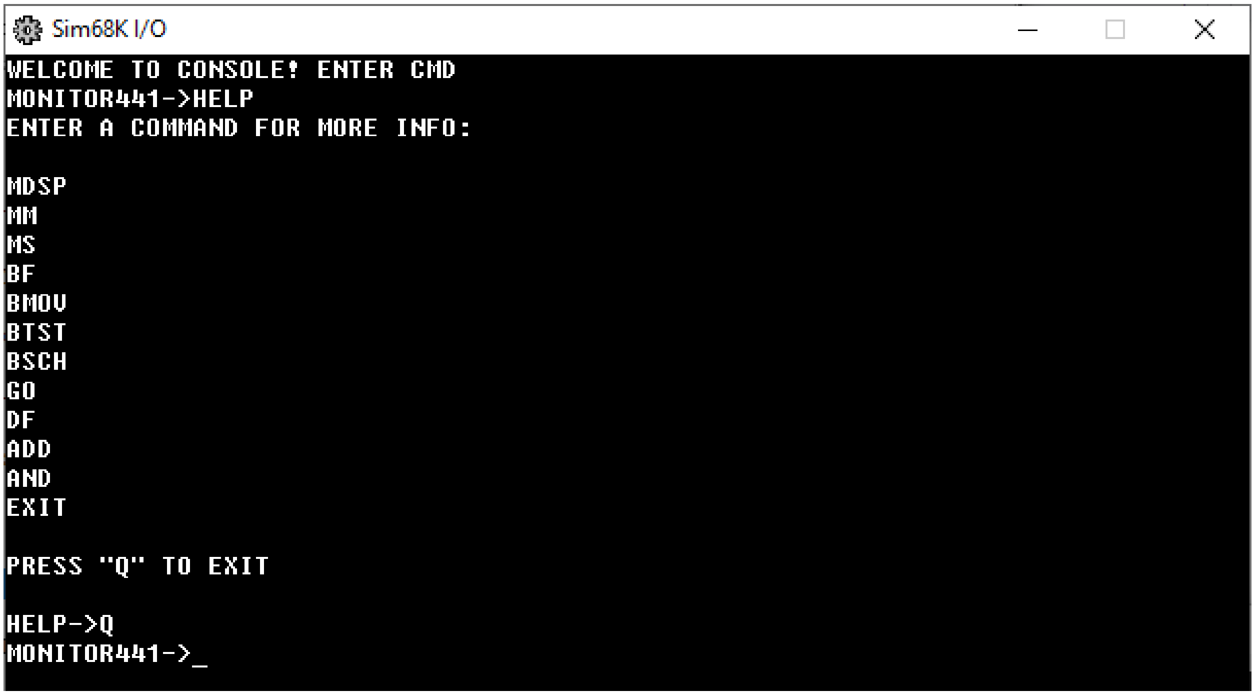


```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->HELP
ENTER A COMMAND FOR MORE INFO:

MDSP
MM
MS
BF
BMOV
BTST
BSCH
GO
DF
ADD
AND
EXIT

PRESS "Q" TO EXIT

HELP->_
```



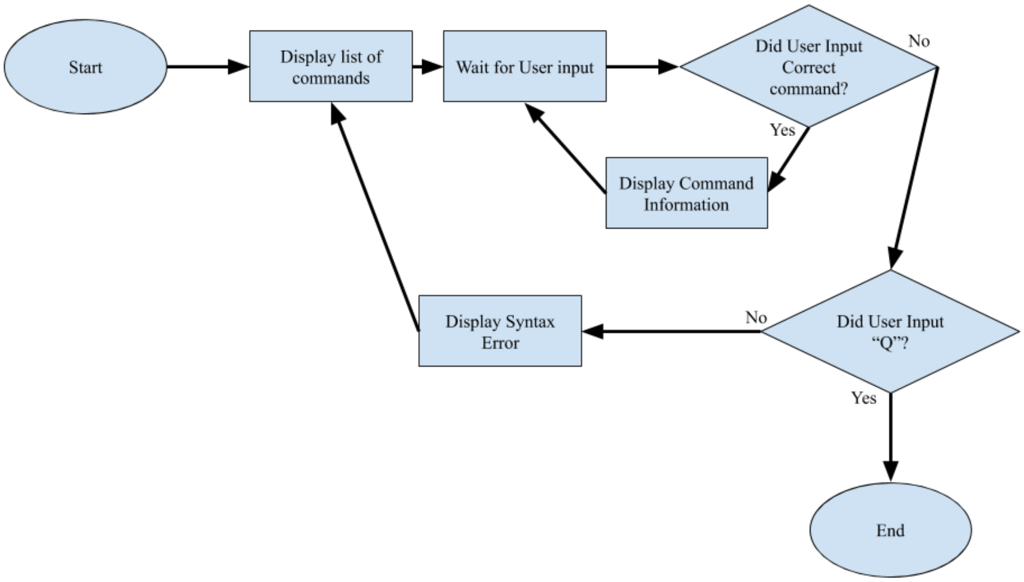
```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->HELP
ENTER A COMMAND FOR MORE INFO:

MDSP
MM
MS
BF
BMOV
BTST
BSCH
GO
DF
ADD
AND
EXIT

PRESS "Q" TO EXIT

HELP->MDSP
MDSP - MEMORY DISPLAY
USED TO DISPLAY RANGE OF MEMORY CONTENTS
SYNTAX MDSP <START> <END>
START AND END ADDRESSES MUST BE EVEN
IF NO END ADDRESS SPECIFIED, 16 BYTES FROM START ADDRESS WILL BE SHOWN

HELP->_
```

To exit the HELP environment, enter “Q” at the prompt. This will result with the help program being terminated, and control will return to the monitor, where the user can input debugger commands as normal.



```
WELCOME TO CONSOLE! ENTER CMD
MONITOR441->HELP
ENTER A COMMAND FOR MORE INFO:

MDSP
MM
MS
BF
BMOV
BTST
BSCH
GO
DF
ADD
AND
EXIT

PRESS "Q" TO EXIT

HELP->Q
MONITOR441->_
```

2.4.1.1. Algorithm and Flowchart



```
Start            ;user enters help command
Display list of commands
Get user input (X)
        While all commands not checked
                Reset buffer address (X)
                Get next command string (Y)
                If X=Y
                        Branch to help message for Y
                Else Continue loop

        If X= "Q"
                Exit help command
        Else Display syntax error
        Repeat loop
End
```

2.4.1.2. Assembly Code

```
*-----------HELP------------*
HELP
   LEA.L HELP_MSG,A5
   LEA.L E_HELP_MSG,A6          ;print list of commands
   BSR PRINT
HELP_LOOP
   LEA.L CURSOR_HELP,A5
   LEA.L E_CURSOR_HELP,A6 ;display HELP-> cursor
   BSR PRINT_NC
   BSR GET_IN                   ;get user input
   LEA.L BUFFER,A1
   CMPI.B #NULL,(A1)
   BEQ HELP_LOOP

   LEA.L CMD_MDSP,A5
   LEA.L E_CMD_MDSP,A6
   LEA.L BUFFER,A1 ;CHECK TO SEE IF MDSP WAS ENTERED
   BSR COMPARE
   BEQ HELP_MDSP

   LEA.L CMD_MM,A5
   LEA.L E_CMD_MM,A6
   LEA.L BUFFER,A1 ;CHECK TO SEE IF MM WAS ENTERED
   BSR COMPARE
   BEQ HELP_MM

   LEA.L CMD_MS,A5
   LEA.L E_CMD_MS,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_MS

   LEA.L QUIT,A5
   LEA.L E_QUIT,A6 ;CHECK TO SEE IF Q WAS ENTERED
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ MAINP       ;if so, leave HELP

   LEA.L CMD_BSCH,A5
   LEA.L E_CMD_BSCH,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BSCH
   *insert other help cmds here + usage info*

   LEA.L CMD_BF,A5
   LEA.L E_CMD_BF,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BF

   LEA.L CMD_BMOV,A5
   LEA.L E_CMD_BMOV,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BMOV

   LEA.L CMD_BTST,A5
   LEA.L E_CMD_BTST,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_BTST

   LEA.L CMD_GO,A5
   LEA.L E_CMD_GO,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_GO

   LEA.L CMD_DF,A5
   LEA.L E_CMD_DF,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_DF

   LEA.L CMD_AND,A5
   LEA.L E_CMD_AND,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_AND

   LEA.L CMD_ADD,A5
   LEA.L E_CMD_ADD,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_ADD

   LEA.L CMD_EXIT,A5
   LEA.L E_CMD_EXIT,A6
   LEA.L BUFFER,A1
   BSR COMPARE
   BEQ HELP_EXIT

   BRA HELP_SYNTAX
   BRA HELP        ;repeat until a correct option selected
HELP_SYNTAX
   LEA.L HELP_SYNTAX_MSG,A5
   LEA.L E_HELP_SYNTAX_MSG,A6
```

```
   BSR PRINT
   BRA HELP
HELP_MDSP
   LEA.L MDSP_HELP_MSG,A5
   LEA.L E_MDSP_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_MM
   LEA.L MM_HELP_MSG,A5
   LEA.L E_MM_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_MS
   LEA.L MS_HELP_MSG,A5
   LEA.L E_MS_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_BSCH
   LEA.L BSCH_HELP_MSG,A5
   LEA.L E_BSCH_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_BF
   LEA.L BF_HELP_MSG,A5
   LEA.L E_BF_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_BMOV
   LEA.L BMOV_HELP_MSG,A5
   LEA.L E_BMOV_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_BTST
   LEA.L BTST_HELP_MSG,A5
   LEA.L E_BMOV_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_GO
   LEA.L GO_HELP_MSG,A5
   LEA.L E_GO_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_DF
   LEA.L  DF_HELP_MSG,A5
   LEA.L  E_DF_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_AND
   LEA.L AND_HELP_MSG,A5
   LEA.L E_AND_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_ADD
   LEA.L ADD_HELP_MSG,A5
   LEA.L E_ADD_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
HELP_EXIT
   LEA.L   EXIT_HELP_MSG,A5
   LEA.L   E_EXIT_HELP_MSG,A6
   BSR PRINT
   BRA HELP_LOOP
```

3. Discussion

3.1. Design Challenges

During the implementation of this monitor program, I ran into issues when attempting to keep program size at a minimum, primarily due to the coding required to output data to the terminal. The best way to resolve this would be to create a single, robust data output subroutine that can be invoked in a variety of situations, however I found this increasingly difficult as I encountered a variety of debugger program types. What resulted was the creation of several, specific purpose output subroutines. For example, there is one subroutine named “D0OUT” that will take data from the D0 register with the data size specified in D1, and output this to the terminal. This is tailor-built, and hard to generalize for use in multiple debugger programs. However, if I were to try to use a single output program, far more coding would have been required in each individual program, so this seemed to be the best solution

Furthermore, when designing the “Help” command for the monitor program, I wanted to create a separate interpreter in order to create a better user experience, but this came at the consequence of size. I did not want to share the code utilized in the main command interpreter, as this would increase complexity drastically, but this meant creating a separate interpreter and routines for each command featured inside the “Help” command environment, and came at the cost of large program size.

The smoothest part of this project was the implementation of the various exception handlers for the MC68000. After reviewing the MC68000’s Programmer’s reference manual, I noted the location of each exception in the MC68000’s vector table and initialized each exception vector at the start of the monitor program. This ensures that if an exception occurs during the command interpreter or other normal operations, the exception vectors will already be correctly initialized and work properly. I took advantage of the “DF” debugger command to display useful register data to the user whenever an exception occurred, reducing the complexity of the exception vectors drastically.

3.2. Advanced Implementation

This monitor program could be enhanced to create a more robust operating system (OS) for a MC68000 family microprocessor by adding static UI elements such as the time and date in the upper corner of the screen. This could be easily accomplished by incorporating the time and date into the main interpreter program that displays the cursor on the screen.

Furthermore, in order to function efficiently in an operating system setting, the monitor program would require more comprehensive routines for interfacing with external peripheral devices. This would be accomplished by creating several trap vectors (one for each peripheral device) in order to efficiently communicate with each device. This would allow for additional functionality with external storage devices and allow the computer to incorporate output devices other than the terminal into programs. This would be very useful for automation uses, where a program could be written to turn on a light after a specific amount of time or adjust a thermostat after a change in the ambient temperature.

Another change that would be important to improving the user experience when utilizing the monitor program would be the inclusion of graphics. The MC68000 has various trap functions available for outputting graphics to the terminal window (see EASY68K documentation, Graphics section) [3]. These graphics could be utilized to display a variety of information to the user, such as a memory diagram, available memory resources (in the form of a pie graph), an analog clock, and other functions as supported by outside programs.

4. Feature Suggestions

4.1. Location Independent Code (Relative Location Coding)

A major improvement to this project would be to translate this project into location independent code. For example, instead of branching to a specific memory address as part of the code, the program could be designed to branch to a location relative to the current instruction. This would be made possible by use of the MC68000's "Memory Indirect" addressing modes. These addressing modes would be utilized to access memory locations relative to the Program counter's (PC) current value. This would allow the monitor program to be placed anywhere in memory and still function correctly.

4.2. Improved Input/Output Subroutines

Additional subroutines could be introduced to allow for a variety of different I/O interactions with the terminal. Currently, many debugger programs utilize unique output mechanisms to display information to the terminal. In the future, programs could be designed as to utilize common output subroutines. This would drastically reduce the size of the monitor program, but would come at the cost of reduced performance due to increased branching to subroutines

4.3. Command History Tracker

Another point where the monitor program is lacking is past command history. Currently there is no way for the monitor program to know what commands were previously input. To improve the robustness of the monitor program, a history buffer could be introduced. This history buffer could allow the user to recall previously input commands and display a formatted history of commands that would be useful when debugging programs. Additionally, this command history could be elaborated to allow the user to see errors caused during previously entered commands and the memory addresses altered by these commands.

4.4. Terminal Appearance Customization

One way to improve the user experience of this monitor program would be to include terminal appearance customization options. EASY68K allows the user to alter the appearance of the terminal using TRAP #15 task number 21. This will allow the user to edit various font properties for the terminal windows, such as font color, size, and actual font type. The monitor program could provide several different presets for the user and could allow for complete control by selecting custom values as directed in the EASY68K Help information. An additional usability feature could include a CLEAR command to clear the screen of any current data.

5. Conclusions

This monitor project should be considered a complete success. The monitor program was successfully created to meet all design requirements. The program completes all expected tasks, including initialization, command interpretation, command execution, and exception processing.

The command interpreter is successfully able to display a cursor to the terminal, receive user input from the terminal, then convert the user input from ASCII data into usable hex data. If an incorrect command is entered, the command interpreter can display errors and direct the user in the correct syntax of the command through use of the HELP command environment.

Furthermore, all debugging programs were successfully implemented, and follow their individual design flowcharts closely. Although the programs take up more space than anticipated, all programs function correctly without unanticipated memory edits or outputs.

Exception processing was successfully implemented to allow the system to recover from unanticipated errors during program execution. The exception handlers also display useful information about the type of error encountered and the status of various system registers. These outputs will allow the user to more efficiently debug programs and determine what went wrong.

This monitor program can be used in a variety of systems utilizing MC68000 processors by altering the Trap output functions used throughout the code. This means that the program has strong potential to be used as the basis for a more comprehensive operating system environment for MC68000 computers.

In the future this project can be expanded upon by adding features such as graphical functionality to the terminal, trap vectors to interface with specific peripheral devices attached by the user, and static UI elements such as a time and date that may be shown at the top of the terminal window. Additionally, other user experience improvements may be included, such as the ability to alter text size, text font, and the ability to clear the terminal screen. These functionalities will greatly improve the user experience.

6. References

[1] Motorola. *Motorola M68000 Family Programmer's Reference Manual*. Motorola Inc., 1992.

[2] Saniie, Dr. Jafar. “ECE 441 Monitor Project.” Illinois Institute of Technology ECE Department.

[3] Kelly, Chuck. “EASY68K Quick Reference.” *EASY68K*, 2009, www.easy68k.com/files/EASy68KQuickRef.pdf.

[4] Saniie, Dr Jafar. “ECE 441 Design Project Spring 2020 Requirement.” Illinois Institute of Technology ECE Dept, Feb. 2020.

[5] Saniie, Dr Jafar. “ECE 441 Experiment 3.” Illinois Institute of Technology ECE Dept, Feb. 2020.