Illinois Institute of Technology

ECE 441 Monitor Project

Author:
Adam Sumner

Teaching Assistant:
Boyang Wang

April 28th, 2015

Acknowledgment

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

Contents

List of Figures

Abstract

This project involved designing and implementing a Monitor program using the MC68000 assembly language. The program implements twelve basic debugger functions as well as two author defined functions. It is designed to handle exceptions, and is meant to be an educational piece of software for students taking ECE 441 at the Illinois Institute of Technology.

1 Introduction

The SANPER-1 ELU is a Motorola MC68000 based microcomputer designed by Dr. Jafar Saniie and Mr. Stephen Perich for use in college level computer engineering courses. For user interaction, it utilizes a monitor program called TUTOR that enables users to actively interact with the microcomputer. The design objective of this project is to re-implement the functionality of TUTOR into a student written monitor program titled MONITOR441. The program should be able to perform basic debugger functions such as memory display, memory sort, memory change, etc., and must have the ability to handle exceptions. The design constraints are:

- Code must be smaller that 3K starting from address \$1000
- Stack size must be 1K starting at memory location \$3000
- Macros may not be used
- Erroneous inputs should not kill the program

Twelve debugger functions must be implemented, along with two user defined debugger commands.

2 Monitor Program

The monitor program operates in a command driven environment. It acts as a typical shell, providing a user interface to access the microcomputer's services. The main program being run is a command line interpreter. Based on the input that the user enters, the interpreter determines if the input entered is valid and subsequently executes the specified command. It was developed using the Easy68K Simulator, thus the TRAP #15 handler is used instead of the MC68000's TRAP #14 handler. The structure of how this program operates is shown in Figure 1.

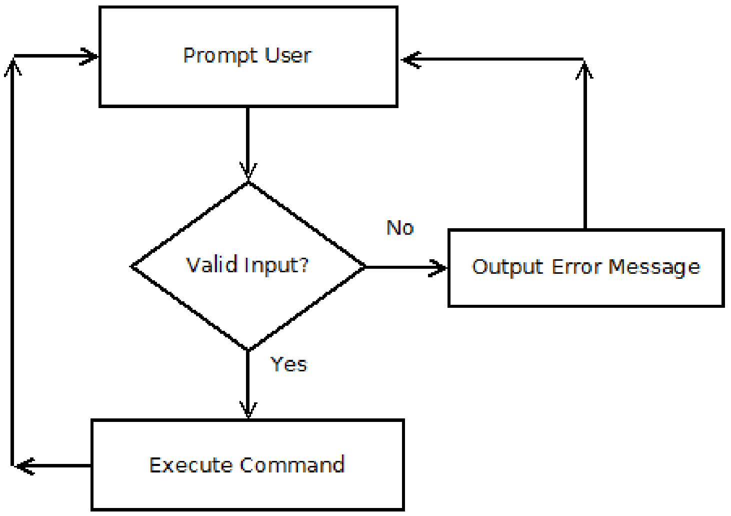

Figure 1: Structure of Monitor Program

2.1 Command Interpreter

2.1.1 Algorithm and Flowchart

The algorithm for the command interpreter uses simple string matching to determine if input is correct. The algorithm begins by outputting the message `MONITOR441>` and accepting input from the user. It then checks for the ASCII value $48 which corresponds to the letter H. This is to check for either the `HELP` command or `HXDC` command. If an H was not entered, it then checks for the ASCII value $4D which corresponds to a memory command. If this fails, then it checks for ASCII value $47, corresponding to the `GO` command. If this fails, the ASCII value $44 is tested, corresponding to the `DF` command. If this fails, it checks for $42, which signifies a `BLCK` command. If this fails, $53 is tested for the `SORTW` command. If this fails, $45 is tested for the `ECHO` command. If this fails $2E is checked for the modify register command. If all of these checks fail, the user has entered incorrect input and an error message is displayed. If any of these checks succeed, the command line interpreter jumps to the respective command's helper interpreter function.

These subroutines check for each character of the user input in order to verify the command the user entered was correct. These helper functions also serve to differentiate commands that start with the same character. The flowchart for this process is shown in Figure 2.

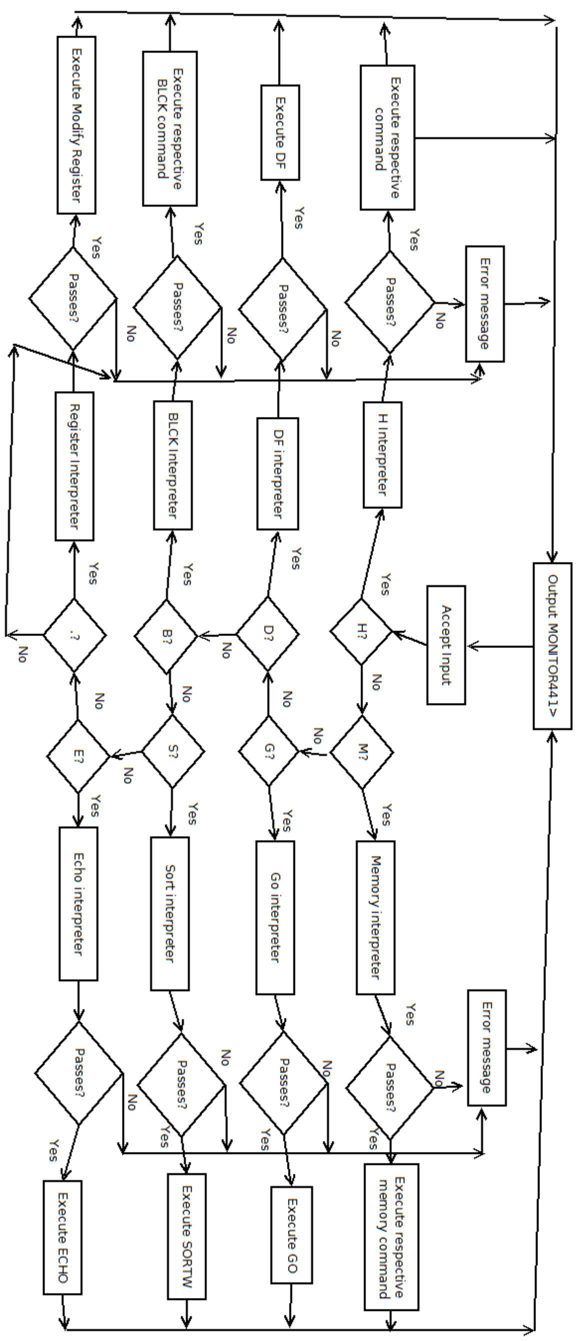

Figure 2: Flowchart for Command Line Interpreter

2.1.2 Assembly Code

```
SHELL:
            PEA      *              ;save PC on Stack for DF
            ADD.L    #4,SP          ;get original value of stack
    pointer
            MOVE.L   SP,-8(SP)      ;save it
            ADD.L    #-8,SP         ;update Stack position
            MOVE     SR,-(SP)       ;save Status register for use
    with DF
            MOVE.L   A6,-(SP)       ;temp save
            MOVE     USP,A6         ;for use with DF command
            MOVE.L   A6,-(SP)       ;store USP to STACK
            ADD.L    #4,SP
            MOVE.L   (SP),A6       ;restore original value
            MOVE.L   -(SP),4(SP)    ;move correct value to correct
    stack position
            ADD.L    #4,SP          ;point stack to CORRECT PLACE


            MOVEM.L D0-D7/A0-A6,-(SP)   ;save initial values of
    registers
            MOVEM.L D0-D7/A0-A6,-(SP)   ;unorthodox
    implementation to save registers when using DF command


            LEA   PROMPT,A1         ;Load message
            MOVE.W   #11,D1         ;load n bytes
            MOVE.B   #1,D0          ;set up trap call
            TRAP     #15
            LEA      BUFFER,A1      ;set up storage for command
            MOVE.B   #2,D0          ;load input trap call
            TRAP     #15
            CMP.B    #$48,(A1)      ;check for help/hxdc
            BEQ      HELPORHXDC
            CMP.B    #$4D,(A1)      ;check for memory command
            BEQ      MEMTEST
            CMP.B    #$47,(A1)      ;check for go
            BEQ      GOTST
            CMP.B    #$44,(A1)      ;check for df
            BEQ      DFTST
            CMP.B    #$42,(A1)      ;check for blck command
            BEQ      BLCKTEST
            CMP.B    #$53,(A1)      ;check for sort command
            BEQ      SORTTEST
            CMP.B   #$45,(A1)     ;check for echo command
            BEQ     ECHOTEST
            CMP.B   #$2E,(A1)     ;check for modify register command
            BEQ     MODIFYREGTEST
            BRA     UNKNOWNCMD
RESTORE:    MOVEM.L (SP)+,D0-D7/A0-A6
            MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF hack workaround
            ADD.L   #4,SP         ;account for USP, it'll fix itself (it shouldn't be used)
                                  ;EASY68k simulator starts in supervisor mode
            MOVE    (SP)+,SR
            MOVE.L  (SP)+,D0      ;save stack cuz it'll get destroyed
            ADD.L   #4,SP         ;get rid of PC, itll fix itself
            MOVE.L  D0,SP
            CLR.L   D0            ;no longer needed

            BRA     SHELL
* ----------------------------------------------------------------------------------------

ECHOTEST:   ADD.L   #1,A1
            CMP.B   #$43,(A1)+    ;C?
            BNE     UNKNOWNCMD
            CMP.B   #$48,(A1)+    ;H?
            BNE     UNKNOWNCMD
            CMP.B   #$4F,(A1)+    ;O?
            BNE     UNKNOWNCMD
            CMP.B   #$20,(A1)+    ;SPACE?
            BEQ     ECHO
            BRA     ERRORSR
* ----------------------------------------------------------------------------------------


* ----------------------------------------------------------------------------------------

BLCKTEST:   ADD.L   #1,A1
```

```
            CMP.B   #$46,(A1)     ;BF?
            BEQ     BFTEST
            CMP.B   #$4D,(A1)     ;BMOV?
            BEQ     BMOVTEST
            CMP.B   #$54,(A1)     ;BTST?
            BEQ     BTSTTEST
            CMP.B   #$53,(A1)     ;BSCH?
            BEQ     BSCHTEST
            BRA     UNKNOWNCMD
*
    ------------------------------------------------------------------------

BSCHTEST:   ADD.L   #1,A1
            CMP.B   #$43,(A1)
            BNE     UNKNOWNCMD
            ADD.L   #1,A1
            CMP.B   #$48,(A1)
            BNE     UNKNOWNCMD
            ADD.L   #1,A1
            CMP.B   #$20,(A1)
            BNE     ERRORSR
            BRA     BSCH

*
    ------------------------------------------------------------------------

BTSTTEST:
            ADD.L   #1,A1
            CMP.B   #$53,(A1)
            BNE     UNKNOWNCMD
            ADD.L   #1,A1
            CMP.B   #$54,(A1)
            BNE     UNKNOWNCMD
            ADD.L   #1,A1
            CMP.B   #$20,(A1)
            BNE     ERRORSR
            BRA     BTST

*
    ------------------------------------------------------------------------

BMOVTEST:   ADD.L   #1,A1
```

```
                CMP.B   #$4F,(A1)
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$56,(A1)
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$20,(A1)
                BNE     ERRORSR
                BRA     BMOV
*
   ------------------------------------------------------------------------------------------------
BFTEST:         ADD.L    #1,A1
                CMP.B   #$20,(A1)
                BNE     ERRORSR
                BRA     BF
*
   ------------------------------------------------------------------------------------------------

DFTST:          ADD.L    #1,A1
                CMP.B   #$46,(A1)
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$00,(A1)
                BNE     ERRORSR
                BRA     DF
*
   ------------------------------------------------------------------------------------------------

SORTTEST:        ADD.L    #1,A1
                CMP.B   #$4F,(A1)      ;O?
                BNE     UNKNOWNCMD
                 ADD.L    #1,A1
                CMP.B   #$52,(A1)      ;R?
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$54,(A1)      ;T?
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$57,(A1)      ;W?
                BNE     UNKNOWNCMD
                ADD.L    #1,A1
                CMP.B   #$20,(A1)
```

```
            BNE     ERRORSR

            BRA     SORTW
*  ------------------------------------------------------------------------------------------

GOTST:      ADD.L    #1,A1
            CMP.B   #$4F,(A1)
            BNE     UNKNOWNCMD
            ADD.L    #1,A1
            CMP.B   #$20,(A1)+
            BNE     ERRORSR
            BRA     GO
*  ------------------------------------------------------------------------------------------

HELPORHXDC: ADD.L    #1,A1
            CMP.B   #$45,(A1)     ;is it help?
            BEQ     HELPTST
            CMP.B   #$58,(A1)     ;or is it hxdc
            BEQ     HXDCTEST
            BRA     UNKNOWNCMD
*  ------------------------------------------------------------------------------------------

HELPTST:
           ADD.L    #1,A1     ; check next char
           CMP.B     #$4C,(A1) ;check for L
           BNE       UNKNOWNCMD
           ADD.L    #1,A1
           CMP.B     #$50,(A1)     ;check for P
           BNE       UNKNOWNCMD
           ADD.L    #1,A1     ;check for anything else
           CMP.B     #$00,(A1)
           BNE       ERRORSR
           BRA       HELP



*  ------------------------------------------------------------------------------------------
```

```
MEMTEST:        ADD.L     #1,A1
                CMP.B     #$53,(A1)
                BEQ       MSSPCTEST
                CMP.B     #$44,(A1)
                BEQ       MDSPCTEST
                CMP.B     #$4D,(A1)
                BEQ       MMSPCTEST
                BRA       UNKNOWNCMD

MSSPCTEST       ADD.L     #1,A1
                CMP.B     #$20,(A1)
                BEQ       MEMSET
                BRA       ERRORSR

MDSPCTEST:
                ADD.L     #1,A1
                CMP.B     #$53,(A1)
                BNE       ERRORSR
                ADD.L     #1,A1
                CMP.B     #$50,(A1)
                BNE       UNKNOWNCMD
                ADD.L     #1,A1
                CMP.B     #$20,(A1)
                BEQ       MEMDISP
                BRA       ERRORSR

MMSPCTEST:      ADD.L     #1,A1
                CMP.B     #$20,(A1)
                BEQ       MM
                BRA       ERRORSR
*
---------------------------------------------------------------------

HXDCTEST:
                ADD.L     #1,A1
                CMP.B     #$44,(A1)
                BNE       UNKNOWNCMD
                ADD.L     #1,A1
                CMP.B     #$45,(A1)
                BNE       UNKNOWNCMD
                ADD.L     #1,A1
                CMP.B     #$43,(A1)
                BNE       UNKNOWNCMD
                ADD.L     #1,A1
                CMP.B   #$20,(A1)
                BNE     ERRORSR
                BRA     HXDC
*
    -----------------------------------------------------------------------------------------------
MODIFYREGTEST:
                ADD.L    #1,A1
                CMP.B   #$44,(A1)
                BEQ     MRD
                CMP.B   #$41,(A1)
                BEQ     MRA
                BRA     UNKNOWNCMD

*---------------------------------USER DEFINED COMMANDS
    --------------------------------------*
*
    -----------------------------------------------------------------------------------------------
ECHO: *What terminal DOESN'T have echo?*

           MOVE.L   A1,A2    ;setup to find end of string
EEND:      CMP.B    #$00,(A2)+
           BEQ      EFOUND
           BRA      EEND
EFOUND:
           SUB.L    #1,A2    ;off by one
           SUB.L    A1,A2    ;find out how many bytes
           MOVE.L   A2,D1    ;place it for trap function
           MOVE.L   #0,D0
           TRAP     #15

           BRA RESTORE
```

2.2 Debugger Commands

2.2.1 Help

2.2.1.1 Algorithm and Flowchart

Help is a simple command that prints out a series of strings that display the available commands, their syntax, and a short description of each command. The syntax to invoke this command is `HELP`. The flowchart for this command is shown in Figure 3.

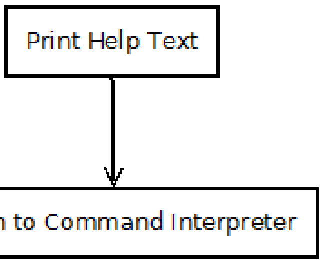

Figure 3: Flowchart for Help

2.2.1.2 Assembly Code

```
HELP:           LEA       HTXT, A1       ; list of commands test
                MOVE.W    #17,D1
                MOVE.B    #0,D0
                TRAP      #15
                MOVE.W    #0,D1          ; newline
                TRAP      #15

                LEA       HTXT1, A1      ;mem display command
                MOVE.W    #75,D1
                MOVE.B    #0,D0
                TRAP      #15
                LEA       HTXT1A, A1     ;mem display
                MOVE.W    #61,D1
                MOVE.B    #0,D0
                TRAP      #15
                LEA       HTXT1B, A1     ;mem display
                MOVE.W    #20,D1
                MOVE.B    #0,D0
                TRAP      #15
                MOVE.W    #0,D1          ; newline
                TRAP      #15

                LEA       HTXT2, A1      ; hxdec command text
                MOVE.W    #75,D1
                MOVE.B    #0,D0
                TRAP      #15
                MOVE.B    #0,D1          ; newline
                TRAP      #15

                LEA       HTXT3, A1      ; sort command text
                MOVE.W    #69,D1
```

```
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT3A,A1       ;sort command text continued
            MOVE.W  #57,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT3B,A1       ;sort command text continued
            MOVE.W  #20,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT3C,A1       ;sort command text continued
            MOVE.W  #21,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT3D,A1       ;sort command text continued
            MOVE.W  #29,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT3E,A1       ;sort command text continued
            MOVE.W  #51,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1          ;newline
            TRAP    #15

            LEA     HTXT4,A1       ;memory modify command text
            MOVE.W  #71,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT4A,A1       ;mem modify command text
    continued
            MOVE.W  #69,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT4B,A1       ;mem modify command text
    continued
            MOVE.W  #27,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT4C,A1       ;mem modify command text
    continued
            MOVE.W  #30,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA      HTXT4D,A1      ;mem modify command text continued
            MOVE.W   #31,D1
            MOVE.B   #0,D0
            TRAP     #15
            LEA      HTXT4E,A1      ;mem modify command text continued
            MOVE.W   #36,D1
            MOVE.B   #0,D0
            TRAP     #15
            MOVE.B   #0,D1
            TRAP     #15           ;newline

            LEA      HTXT5,A1      ;memory set command text
            MOVE.W   #70,D1
            MOVE.B   #0,D0
            TRAP     #15
            LEA      HTXT5A,A1      ;memory set command text continued
            MOVE.W   #9,D1
            MOVE.B   #0,D0
            TRAP     #15
            MOVE.B   #0,D1         ;newline
            TRAP     #15

            LEA      HTXT6,A1      ;block fill command text
            MOVE.W   #69,D1
            MOVE.B   #0,D0
            TRAP     #15
            LEA      HTXT6A,A1      ;block fill command text
            MOVE.W   #72,D1
            MOVE.B   #0,D0
            TRAP     #15
            LEA      HTXT6B,A1      ;block fill command text
            MOVE.W   #38,D1
            MOVE.B   #0,D0
            TRAP     #15
            MOVE.B   #0,D1
            TRAP     #15           ;newline


            LEA      HTXT7,A1      ;block move command text
            MOVE.W   #68,D1
            MOVE.B   #0,D0
            TRAP     #15
            LEA     HTXT7A,A1     ;block move command text
            MOVE.W  #72,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT7B,A1     ;block move command text
            MOVE.W  #24,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1        ;newline
            TRAP    #15

            LEA     HTXT8,A1     ;block test command text
            MOVE.W  #71,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT8A,A1     ;block test command text
            MOVE.W  #40,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1        ;newline
            TRAP    #15

            LEA     HTXT9,A1     ;block search command text
            MOVE.W  #70,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT9A,A1     ;block search command text
            MOVE.W  #45,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1        ;newline
            TRAP    #15

            LEA     HTXT10,A1     ;go command text
            MOVE.W  #61,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1       ;newline
            TRAP    #15

            LEA     HTXT11,A1    ;df command text
            MOVE.W  #56,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1
            TRAP    #15

            LEA     HTXT12,A1       ;help command text
            MOVE.W  #66,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1           ;newline
            TRAP    #15

            LEA     HTXT13,A1       ;echo command text
            MOVE.W  #52,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1           ;newline
            TRAP    #15

            LEA     HTXT14,A1       ;modify register command text
            MOVE.W  #71,D1
            MOVE.B  #0,D0
            TRAP    #15
            LEA     HTXT15,A1       ;modify register command text
            MOVE.W  #63,D1
            MOVE.B  #0,D0
            TRAP    #15
            MOVE.B  #0,D1           ;newline
            TRAP    #15

            BRA     RESTORE
```

2.2.2 Memory Display

2.2.2.1 Algorithm and Flowchart

Memory display is an extremely useful tool to look at blocks of memory. The syntax to call this function is `MDSP <address1> <address2`, where `<address1>` is the starting address and `<address2>` is the ending address of the memory contents to be shown. This command also displays the block of memory from `<address1>` to `<address2 +16bytes>`. The flowchart for this command is shown in Figure 4.

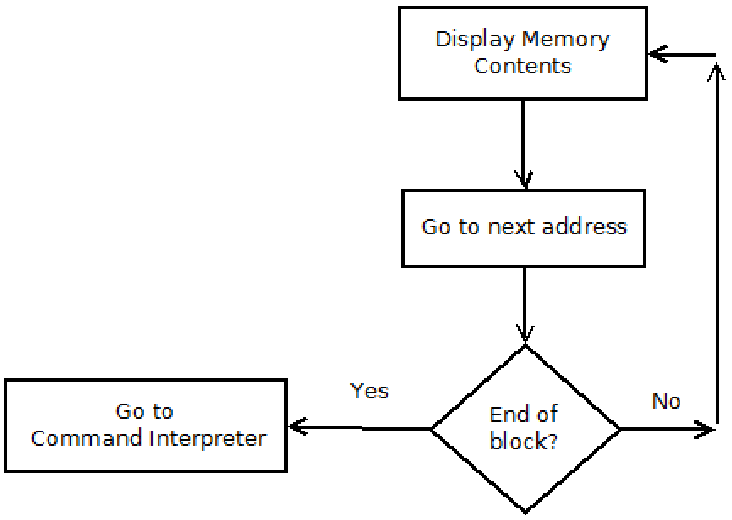

Figure 4: Flowchart for Memory Display

2.2.2.2 Assembly Code

```
MEMDISP:    LEA      BUFFER, A2
            MOVE.L   #1,D6        ;counter for how many times to
    loop
            ADD.L    #5,A2        ;get first address
            MOVE.L   A2,A3
FINDEND1:   CMP.B    #$20,(A3)+
            BEQ      FINDNEXT
            BRA      FINDEND1
FINDNEXT:   MOVE.L   A3,A4
            MOVE.L   A3,A5
            SUB.L    #1,A3    ;get rid of off by one error
FINDEND2:   CMP.B    #$00,(A5)+
            BEQ      MEMNEXT
            BRA      FINDEND2
MEMNEXT:    SUB.L    #1,A5    ;off by one error
            JSR      ASCII_ADDRESS
            MOVE.L   D5,A6    ;put 1st address in A6
            MOVE.L   A4,A2
            MOVE.L   A5,A3
            JSR  ASCII_ADDRESS
            MOVE.L   D5,A5    ;second address in A5
            MOVE.L   A6,A0    ;for second run through
            MOVE.L   A5,A1    ;see above comment
```

```
                ADD.L    #16,A1 ;16 byte offset
                MOVEM.L  A1,-(SP)
DISPLOOP:       CMP.L    A6,A5
                BLT      SECONDLOOP
                MOVE.L   A6,D3
                JSR      HEXTOASCII
                SUB.L    A2,A3
                MOVE.L   A3,D1    ;number of ascii values to display
                MOVE.L   A2,A1
                MOVE.L   #1,D0
                TRAP     #15
                LEA      SPACE,A1
                MOVE.L #1,D1
                TRAP     #15
                CLR.L    D3
                MOVE.B   (A6),D3
                JSR      HEXTOASCII
                SUB.L    A2,A3
                MOVE.L   A3,D1
                MOVE.L   A2,A1
                MOVE.L   #0,D0
                TRAP     #15
                ADD.L    #1,A6
                BRA      DISPLOOP

SECONDLOOP:
                MOVE.B   #0,D0
                MOVE.B   #0,D1
                TRAP     #15
                MOVEM.L  (SP)+,A1
                MOVE.L   A0,A6    ;reinit
                MOVE.L   A1,A5
                SUBI.L   #1,D6
                CMP.L    #$0,D6
                BEQ      DISPLOOP
                SUB.L    #4,SP    ;off by long error on stack
                BRA      RESTORE
```

2.2.3 HXDEC

2.2.3.1 Algorithm and Flowchart

This command will allow the user to enter a hexadecimal value (up to FFFF), and the program will return the equivalent value in decimal format. The syntax to call this function is `HXDEC <data>`. It works by extracting the

ASCII values byte by byte and determining the 16's place of each byte. The value extracted is then multiplied by its respective 16's place and added to a register that stores the total. This total must then be converted into BCD for output and then into ASCII to display it on the terminal.

2.2.3.2 Assembly Code

```
HXDC:     LEA BUFFER,A2     ;load buffer
          ADD.L    #6,A2     ; start of number
          MOVE.L   A2,A3     ;set up end pointer
          MOVE.L   #1,D1     ;set up 16's place
          CLR.L    D2        ;clear total
          CLR.L    D3        ;temp holder for number
          CLR.L    D6        ;Final Value in BCD
          MOVE.L   #10000,D4      ;maximum 10's place of converted
    number
          MOVE.L   #16,D5       ;Max number of rotates needed
          LEA $3A00,A5
          LEA $3A00,A4    ;set up start pointer
FINDLASTNUM:
          CMP.B #$00,(A3)+
          BEQ      CONVERTMINUS1
          BRA      FINDLASTNUM
CONVERTMINUS1:
                SUB.L    #1,A3 ; cure off by 1 error
CONVERT:
               SUB.L    #1,A3
               CMP    A3,A2
               BGT    ENDCONVERT
               CMP.B    #$40,(A3)
               BGT      HIGHHEX
               SUBI.B   #$30,(A3)    ;get hex value
               BRA      COMPUTATION
HIGHHEX:        SUBI.B   #$37,(A3)    ;get hex value
COMPUTATION:
               MOVE.B   (A3),D3
               MULU     D1,D3    ;get 16's place
              ; DIVU     #16,D3   ;get rid of off by 1 exponent error
               MULU     #16,D1   ;inc 16's place counter
               MOVE.B   D3,(A4)
               SUB.L    #1,A4
               ADD.L    D3,D2    ;store it in total for debugging
               CLR.L    D3       ;get rid of any numbers in there
               BRA      CONVERT
```

```
ENDCONVERT:                       ;must convert back to ascii for
    display
                CLR.L     D3      ;Cleared for workability
                DIVU      D4,D2   ;get 10's place digit
                MOVE.W    D2,D3   ;extract 10's place digit to D3
                ROL.L     D5,D3   ;put it in its place
                CLR.W     D2      ;get rid of whole number
                SWAP      D2      ;keep remainder
                SUBI.L    #4,D5   ;dec rotate counter
                ADD.L     D3,D6   ;put it into it's place
                DIVU      #10,D4  ;go down a 10's place
                CMP.W     #0,D4   ;are we done
                BEQ       OUTPUTNUM
                BRA       ENDCONVERT

OUTPUTNUM:
               MOVE.L     D6,D3   ;put into register for conversion to
     ASCII
               JSR        HEXTOASCII
               MOVEA.L    A2,A1   ;get start of number
               SUBA       A2,A3   ;get how many bytes to output
               MOVE.L     A3,D1   ;for Trap call
               MOVE.L     #0,D0
               TRAP       #15

               BRA RESTORE
```

2.2.4 SORTW

This command implements the most common sort algorithm for a set of data, the bubble sort. Because the user has the choice to choose between sorting the data in ascending or descending order, it also implements a “rock” sort. It works by first determining which option, ascending or descending, the user has selected. Once determined, the first data in the set is analyzed to the next immediate adjacent value in memory. If the current data is larger than the next data (assuming ascending order for example), the two words of data are swapped. This value is continuously checked against its immediate adjacent memory until it “fits” in the current state of the list. This process is repeated for n elements in a list of n words. The runtime is $\mathcal{O}(n^2)$, and the syntax for this command is `SORTW <option> <address1> <address2>`, where both `<address1>` and `<address2>` are even addresses.

2.2.4.1 Algorithm and Flowchart

dfg

2.2.4.2 Assembly Code

```
SORTW:  ADD.L   #1,A1          ;increment to check for semicolon/
    dash
         CMP.B   #$2D,(A1)      ;check for default
         BEQ     DESCEND
         CMP.B   #$3B,(A1)+
         BNE     ERRORSR
         CMP.B   #$41,(A1)      ;is it ascending?
         BEQ     ASCEND
         CMP.B   #$44,(A1)      ;or descending?
         BNE     ERRORSR
         BRA     DESCEND

ASCEND:
        ADD.L    #1,A1     ;inc
        CMP.B    #$20,(A1)      ;check space
        BNE      ERRORSR
        ADD.L    #1,A1     ;start of 1st address
        MOVE.L   A1,A2
        MOVE.L   A2,A3
AGETFIRSTADDRESS:
        CMP.B    #$00,(A3)
        BEQ      ERRORSR        ;incorrect syntax
        CMP.B    #$20,(A3)+     ;trying to find the end
        BEQ      AFADDCONV
        BRA      AGETFIRSTADDRESS
AFADDCONV:
        SUB.L    #1,A3     ;off by one error
        JSR ASCII_ADDRESS      ;D5 now has that address
        MOVE.L D5,A4
        ADD.L    #1,A3     ;start of second address
        MOVE.L   A3,A2     ;setup for second address
AGETSECADDRESS:
        CMP.B    #$00,(A3)+     ;trying to find the end
        BEQ      ASADDCONV
        BRA      AGETSECADDRESS
ASADDCONV:
         SUB.L   #1,A3     ;off by one
         JSR     ASCII_ADDRESS
         MOVE.L  D5,A5
```

```
            MOVEA.L   A4,A6   ;CLR A6

ARESETLOOP: MOVE.L   A6,A4     ;reset to top of loop
ACMP:        CMP.W    (A4)+,(A4)+  ;check adjacent mem
             BLS.S    ASWAP
             SUBQ.L   #2,A4
             CMP.L    A4,A5    ;done?
             BNE      ACMP      ;nope
             BRA      DONEASCEND  ;yep
ASWAP:        MOVE.L   -(A4),D0     ;start bubbling
             SWAP.W   D0
             MOVE.L   D0,(A4)
             BRA      ARESETLOOP


DESCEND:
        ADD.L     #1,A1    ;inc
        CMP.B     #$20,(A1)    ;check space
        BNE       ERRORSR
        ADD.L     #1,A1    ;start of 1st address
        MOVE.L    A1,A2
        MOVE.L    A2,A3
DGETFIRSTADDRESS:
        CMP.B     #$00,(A3)
        BEQ       ERRORSR      ;incorrect syntax
        CMP.B     #$20,(A3)+  ;trying to find the end
        BEQ       DFADDCONV
        BRA       DGETFIRSTADDRESS
DFADDCONV:
        SUB.L     #1,A3    ;off by one error
        JSR  ASCII_ADDRESS     ;D5 now has that address
        MOVE.L D5,A4
        ADD.L     #1,A3    ;start of second address
        MOVE.L    A3,A2    ;setup for second address
DGETSECADDRESS:
        CMP.B     #$00,(A3)+  ;trying to find the end
        BEQ       DSADDCONV
        BRA       DGETSECADDRESS
DSADDCONV:
         SUB.L    #1,A3    ;off by one
         JSR      ASCII_ADDRESS
         MOVE.L   D5,A5
         MOVEA.L   A4,A6   ;CLR A6

DRESETLOOP: MOVE.L   A6,A4     ;reset to top of loop
```

```
DCMP:           CMP.W    (A4)+,(A4)+   ;check adjacent mem
                BHI.S    DSWAP
                SUBQ.L   #2,A4
                CMP.L    A4,A5     ;done?
                BNE      DCMP      ;nope
                BRA      DONEDESCEND   ;yep
DSWAP:           MOVE.L   -(A4),D0      ;start bubbling
                SWAP.W   D0
                MOVE.L   D0,(A4)
                BRA      DRESETLOOP
x
DONEASCEND:
DONEDESCEND:
                BRA RESTORE
```

2.2.5 Memory Modify

This command first determines which option the user has selected. Depending on this option, it reads the address entered by the user and displays the specified amount of data currently stored in memory. The user is then prompted to enter data to store into memory. The command increments the memory location and asks for input until the user enters the '.' character. The syntax for this command is `MM <option> <address>`.

2.2.5.1 Algorithm and Flowchart

2.2.5.2 Assembly Code

```
MM:       CLR.L     D2         ;used for storing values
          CLR.L     D6
SIZECHECK:
           MOVE.L    A1,A3     ;set up to find end ptr
ENDPTRMM:
           CMP.B    #$00,(A3)+
           BNE      ENDPTRMM
           SUB.L     #1,A3     ;off by one error
           ADD.L     #1,A1     ;inc pointer to start of specifier
           CMP.B    #$2D,(A1)     ;check for default
           BEQ      DEFAULT
           CMP.B    #$3B,(A1)
```

```
        BNE      ERRORSR
        ADD.L    #1,A1      ;find out which size
        CMP.B    #$57,(A1)  ; word
        BEQ      WORD
        CMP.B    #$4C,(A1)      ;long
        BEQ      LONG
        BRA      ERRORSR

*************************************************

DEFAULT:

        ADD.L    #2,A1          ;set up for subroutine
        MOVE     A1,A2          ;set up for subroutine
        MOVEM.L  D1/D6/A1-A3,-(SP)
        JSR      ASCII_ADDRESS
        MOVEM.L  (SP)+,D1/D6/A1-A3
        MOVE.L   D5,A4          ;set up address to modify

MODIFYLOOP:
        *--------Display Memory First-----*
        MOVE.L   A4,D3          ;set up for subroutine
        JSR      HEXTOASCII     ;convert new address to ascii for
    output
        SUBA     A2,A3          ;get num of bytes to produce
        MOVE.L   #1,D0
        MOVE.L   A3,D1
        MOVE.L   A2,A1
        TRAP     #15

        *add colon to denote containing data*
        MOVE.B   #$3A,(A1)
        MOVE.L   #1,D1      ;display only the colon
        MOVE.L   #1,D0
        TRAP     #15

        MOVE.B   (A4),D3
        JSR      HEXTOASCII
        MOVE.L   #2,D1
        SUB.L    A2,A3
        CMP      #2,A3
        BEQ      FORMATGOOD
        SUB.L    #1,A2
FORMATGOOD:
        MOVE.L   A2,A1
```

```
         MOVE.B   #1,D0
         TRAP     #15

         MOVE.B   #$20,(A1)
         MOVE.L   #1,D1    ;space between held data and input
         MOVE.L   #1,D0
         TRAP     #15


         *------Enter Input--------*
         CLR.L    D3
         MOVE.L   #4,D6
         LEA      BUFFER,A1    ;set up storage for command
         MOVE.B   #2,D0        ;load input trap call
         TRAP     #15
         CMP.B    #$2E,(A1)
         BEQ      ENDLP
         CMP.B    #$00,(A1)
         BEQ      ENTER

PARSELOOP:
         CMP.B    #$00,(A1)
         BEQ      ENDPARSE
         CMP.B    #$40,(A1)
         BGT      HIGHHEXMM
         SUBI.B   #$30,(A1)    ;get hex value
         BRA      NEXTMMSTEP
HIGHHEXMM: SUBI.B   #$37,(A1)   ;get hex value
NEXTMMSTEP:
         MOVE.B   (A1),D2
         ROL.L    D6,D2
         SUBI.L   #4,D6
         ADD.L    #1,A1
         ADD.B    D2,D3    ;total byte stored in D3
         BRA      PARSELOOP
ENDPARSE:
         MOVE.B   D3,(A4)    ;commit memory change
ENTER:   ADD.L    #1,A4    ;increment address
         BRA      MODIFYLOOP

**************************************************

WORD:

         ADD.L    #2,A1        ;set up for subroutine
```

```
        MOVE     A1,A2          ;set up for subroutine
        MOVEM.L  D1/D6/A1-A3,-(SP)
        JSR      ASCII_ADDRESS
        MOVEM.L  (SP)+,D1/D6/A1-A3
        MOVE.L   D5,A4          ;set up address to modify

MODIFYLOOPW:
        *--------Display Memory First-----*
        MOVE.L   A4,D0
        DIVU     #2,D0
        SWAP     D0       ;check if it's an odd address
        CMP.W    #$00,D0
        BNE      ERRORSR
        MOVE.L   A4,D3          ;set up for subroutine
        MOVE.L   A4,A5          ;next byte of memory may not be
    needed
        ADD.L    #1,A5
        JSR      HEXTOASCII     ;convert new address to ascii for
    output
        SUBA     A2,A3          ;get num of bytes to produce
        MOVE.L   #1,D0
        MOVE.L   A3,D1
        MOVE.L   A2,A1
        TRAP     #15

        *add colon to denote containing data*
        MOVE.B   #$3A,(A1)
        MOVE.L   #1,D1    ;display only the colon
        MOVE.L   #1,D0
        TRAP     #15

        MOVE.B   (A4),D3
        JSR      HEXTOASCII
        MOVE.L   #2,D1
        SUB.L    A2,A3
        CMP      #2,A3
        BEQ      FORMATGOOD1
        SUB.L    #1,A2
FORMATGOOD1:

        MOVE.L   A2,A1
        MOVE.B   #1,D0
        TRAP     #15

        MOVE.B   (A5),D3
```

```
            JSR         HEXTOASCII
            MOVE.L      #2,D1
            SUB.L       A2,A3
            CMP         #2,A3
            BEQ         FORMATGOOD2
            SUB.L       #1,A2
FORMATGOOD2:

            MOVE.L      A2,A1
            MOVE.B      #1,D0
            TRAP        #15


            MOVE.B      #$20,(A1)
            MOVE.L      #1,D1       ;space between held data and input
            MOVE.L      #1,D0
            TRAP        #15


            *-----Enter Input-------*
            CLR.L       D3
            MOVE.L      #12,D6
            LEA         BUFFER,A1       ;set up storage for command
            MOVE.B      #2,D0           ;load input trap call
            TRAP        #15
            CMP.B       #$2E,(A1)
            BEQ         ENDLP
            CMP.B       #$00,(A1)
            BEQ         ENTERW

PARSELOOPW:
            CMP.B       #$00,(A1)
            BEQ         ENDPARSEW
            CMP.B       #$40,(A1)
            BGT         HIGHHEXMMW
            SUBI.B      #$30,(A1)       ;get hex value
            BRA         NEXTMMSTEPW
HIGHHEXMMW: SUBI.B      #$37,(A1)       ;get hex value
NEXTMMSTEPW:
            MOVE.B      (A1),D2
            ROL.L       D6,D2
            SUBI.L      #4,D6
            ADD.L       #1,A1
            ADD.L       D2,D3       ;total byte stored in D3
            CLR.L       D2          ;clear for next rotate
```

```
        BRA      PARSELOOPW
ENDPARSEW:

        MOVE.W   D3,(A4)    ;commit memory change
ENTERW:   ADD.L    #2,A4    ;increment address
        BRA      MODIFYLOOPW

************************************************

LONG:
        ADD.L    #2,A1          ;set up for subroutine
        MOVE     A1,A2          ;set up for subroutine
        MOVEM.L  D1/D6/A1-A3,-(SP)
        JSR      ASCII_ADDRESS
        MOVEM.L  (SP)+,D1/D6/A1-A3
        MOVE.L   D5,A4          ;set up address to modify

MODIFYLOOPL:
        *--------Display Memory First------*
        MOVE.L   A4,D0
        DIVU     #2,D0
        SWAP     D0       ;check if it's an odd address
        CMP.W    #$00,D0
        BNE      ERRORSR
        MOVE.L   A4,D3          ;set up for subroutine
        MOVE.L   A4,A5          ;next byte of memory may not be
    needed
        ADD.L    #1,A5
        JSR      HEXTOASCII  ;convert new address to ascii for
    output
        SUBA     A2,A3          ;get num of bytes to produce
        MOVE.L   #1,D0
        MOVE.L   A3,D1
        MOVE.L   A2,A1
        TRAP     #15

        *add colon to denote containing data*
        MOVE.B   #$3A,(A1)
        MOVE.L   #1,D1    ;display only the colon
        MOVE.L   #1,D0
        TRAP     #15

        MOVE.B   (A4),D3
        JSR      HEXTOASCII
        MOVE.L   #2,D1
```

```
            SUB.L   A2,A3
            CMP     #2,A3
            BEQ     FORMATGOOD3
            SUB.L   #1,A2
FORMATGOOD3:

            MOVE.L  A2,A1
            MOVE.B  #1,D0
            TRAP    #15

            MOVE.B  (A5)+,D3
            JSR     HEXTOASCII
            MOVE.L  #2,D1
            SUB.L   A2,A3
            CMP     #2,A3
            BEQ     FORMATGOOD4
            SUB.L   #1,A2
FORMATGOOD4:

            MOVE.L  A2,A1
            MOVE.B  #1,D0
            TRAP    #15

            MOVE.B  (A5)+,D3
            JSR     HEXTOASCII
            MOVE.L  #2,D1
            SUB.L   A2,A3
            CMP     #2,A3
            BEQ     FORMATGOOD5
            SUB.L   #1,A2
FORMATGOOD5:

            MOVE.L  A2,A1
            MOVE.B  #1,D0
            TRAP    #15
            MOVE.B  (A5)+,D3
            JSR     HEXTOASCII
            MOVE.L  #2,D1
            SUB.L   A2,A3
            CMP     #2,A3
            BEQ     FORMATGOOD6
            SUB.L   #1,A2
FORMATGOOD6:

            MOVE.L  A2,A1
```

```
        MOVE.B  #1,D0
        TRAP    #15
```

2.2.6 Memory Set

2.2.6.1 Algorithm and Flowchart

This command is a simpler version of Memory Modify. It parses the data the user entered and stores it at one specified address. It has the syntax `MS <data> <address>`.

2.2.6.2 Assembly Code

```
MEMSET:         LEA     BUFFER,A2
                ADD.L   #3,A2
                MOVE.L  A2,A3    ;set up to find end
FINDEND:        CMP.B   #$00,(A3)+
                BEQ     MEMCONT
                BRA     FINDEND
MEMCONT:        SUB.L   #1,A3    ;get rid of off by one erro
                MOVE.B  (A2)+,D1
                MOVE.B  (A2),D2
                MOVE.B  D1,D3           ;pass value to subroutine
                JSR     ASCII_TO_BCD
                MOVE.B  D3,D1           ;get converted value
                MOVE.B  D2,D3           ;pass value
                JSR     ASCII_TO_BCD
                MOVE.B  D3,D2           ;get returned value
                MOVE.B  D1,D3
                JSR     BCD_TO_HEX
                MOVE.B  D3,D1
                MOVE.B  D2,D3
                JSR     BCD_TO_HEX
                MOVE.B  D3,D2
                ROL.L   #4,D1           ;put data in correct place
                ADD     D1,D2           ;get combined data input
                ADD.L   #2,A2           ;go to start of address
                JSR     ASCII_ADDRESS   ;get address in workable form
                MOVE.L  D5,A4           ;load target address
                MOVE.B  D2,(A4)           ;put data in target address
                BRA     RESTORE                 ;return to shell
```

2.2.7 Block Fill

2.2.7.1 Algorithm and Flowchart

This command requires two even addresses to be entered. It then parses the word sized data entered by the user and fills the block of memory from the first address to the second address. The syntax for this command is `BF <data> <address1> <address2>`.

2.2.7.2 Assembly Code

```
            TRAP     #15


            *———Enter Input————*
            CLR.L    D3
            MOVE.L   #28,D6
            LEA      BUFFER,A1     ;set up storage for command
            MOVE.B   #2,D0         ;load input trap call
            TRAP     #15
            CMP.B    #$2E,(A1)
            BEQ      ENDLP
            CMP.B    #$00,(A1)
            BEQ      ENTERL

PARSELOOPL:
            CMP.B    #$00,(A1)
            BEQ      ENDPARSEL
            CMP.B    #$40,(A1)
            BGT      HIGHHEXMML
            SUBI.B   #$30,(A1)     ;get hex value
            BRA      NEXTMMSTEPL
HIGHHEXMML: SUBI.B   #$37,(A1)     ;get hex value
NEXTMMSTEPL:
            MOVE.B   (A1),D2
            ROL.L    D6,D2
            SUBI.L   #4,D6
            ADD.L    #1,A1
            ADD.L    D2,D3      ;total byte stored in D3
            CLR.L    D2         ;clear for next rotate
            BRA      PARSELOOPL
ENDPARSEL:
            MOVE.L   D3,(A4)     ;commit memory change
ENTERL:     ADD.L    #4,A4      ;increment address
```

```
            BRA       MODIFYLOOPL


ENDLP:      BRA RESTORE


*   ----------------------------------------------------------------------------

BF:
           ADD.L      #1,A1     ;first byte of data
           MOVE.L     A1,A3     ;for end ptr
BFGETENDDATA:
            CMP.B     #$20,(A3)+
            BEQ       BFNEXTADDR
            BRA       BFGETENDDATA
BFNEXTADDR:
            MOVE.L    A1,A2     ;for subroutine
            SUB.L     #1,A3     ;off by one error
            JSR       ASCII_ADDRESS
            MOVE.L    D5,-(SP)      ;save data on the stack
```

2.2.8 Block Move

This command move a block of memory from one section to another. Both block sizes must be equal. Starting from the first address of the first block, it moves data byte by byte starting from the first address of the second block until all data has been copied. Its syntax is `BMOV <address1> <address2> <address3> <address4>`.

2.2.8.1 Algorithm and Flowchart

2.2.8.2 Assembly Code

```
            MOVE.L    A3,A2     ;set start ptr
BFGETENDADDRONE:
            CMP.B     #$20,(A3)+
            BEQ       BFNEXTADDRTWO
            BRA       BFGETENDADDRONE

```

```
BFNEXTADDRTWO:
          SUB.L    #1,A3    ;off by one error
          JSR      ASCII_ADDRESS    ;convert address to hex
          MOVE.L   D5,A5        ;store address 1 in A5
          DIVU     #2,D5
          SWAP     D5
          CMP.W    #$00,D5
          BNE      ERRORSR

          ADD.L    #1,A3    ;inc end ptr to first byte of address
          MOVE.L   A3,A2    ;set start ptr
BFGETLASTEND:
          CMP.B    #$00,(A3)+
          BEQ      STOREDATA
          BRA      BFGETLASTEND

STOREDATA:
          SUB.L    #1,A3    ;off by one error
          JSR      ASCII_ADDRESS
          MOVE.L   D5,A6    ;end address in A6
          DIVU     #2,D5
          SWAP     D5
          CMP.W    #$00,D5
          BNE      ERRORSR
          MOVE.L   (SP)+,D5

DATALOOP:
          CMP.L    A5,A6
          BLT      ENDBF
          MOVE.W   D5,(A5)+
          BRA      DATALOOP

ENDBF:    BRA RESTORE
*   ------------------------------------------------------------------------------------------------

BMOV:     ADD.L    #1,A1    ;get to start of first address
          MOVE.L   A1,A2    ;set up start ptr
          MOVE.L   A2,A3    ;set up end ptr

FIRSTADDRESS:
          CMP.B #$20,(A3)+
          BEQ      COMPUTEFIRSTADD
          BRA      FIRSTADDRESS
```

```

COMPUTEFIRSTADD:
        SUB.L   #1,A3   ;off by one error
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A0   ; save 1st address

        ADD.L   #1,A3
        MOVE.L  A3,A2
SECONDADDRESS:
        CMP.B   #$20,(A3)+
        BEQ     COMPUTESECONDADDRESS
        BRA     SECONDADDRESS

COMPUTESECONDADDRESS:
        SUB.L   #1,A3   ;off by one error
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A4   ;save 2nd address

        ADD.L   #1,A3
        MOVE.L  A3,A2
THIRDADDRESS:
        CMP.B   #$20,(A3)+
        BEQ     COMPUTETHIRDADDRESS
        BRA     THIRDADDRESS

COMPUTETHIRDADDRESS:
        SUB.L   #1,A3
```

2.2.9 Block Test

2.2.9.1 Algorithm and Flowchart

2.2.9.2 Assembly Code

```
        ADD.L   #1,A3
        MOVE.L  A3,A2
FOURTHADDRESS:
        CMP.B   #$00,(A3)+
        BEQ     COMPUTEFOURTHADDRESS
        BRA     FOURTHADDRESS

COMPUTEFOURTHADDRESS:
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A6   ;save 3rd address



        *Check for matching dimensions*
        MOVE.L  A0,D0
        MOVE.L  A4,D1
        MOVE.L  A5,D5
        MOVE.L  A6,D6
        SUB.L   D0,D1
        SUB.L   D5,D6
        CMP.L   D1,D6
        BNE     ERRORSR
        CMP.L   A0,A4
        BLT     ERRORSR
        CMP.L   A5,A6
        BLT     ERRORSR

DATATRANSFER:
        CMP.L   A0,A4
        BLT     BMOVDONE
        MOVE.B  (A0)+,(A5)+
        BRA     DATATRANSFER



BMOVDONE:
        BRA RESTORE

*
    ------------------------------------------------------------------------

BTST:
       ADD.L    #1,A1   ;first byte of data
       MOVE.L   A1,A3   ;for end ptr
BTSTGETENDDATA:
        CMP.B   #$20,(A3)+
        BEQ     BTSTNEXTADDR
        BRA     BTSTGETENDDATA
BTSTNEXTADDR:
        MOVE.L  A1,A2   ;for subroutine
        SUB.L   #1,A3   ;off by one error
        JSR     ASCII_ADDRESS
        MOVE.L  D5,-(SP)        ;save data on the stack

        ADD.L   #1,A3   ;inc end ptr to first byte of address
        MOVE.L  A3,A2   ;set start ptr
BTSTGETENDADDRONE:
        CMP.B   #$20,(A3)+
        BEQ     BTSTNEXTADDRTWO
        BRA     BTSTGETENDADDRONE

BTSTNEXTADDRTWO:
        SUB.L   #1,A3   ;off by one error
        JSR     ASCII_ADDRESS   ;convert address to hex
        MOVE.L  D5,A5           ;store address 1 in A5
        MOVE.L  D5,A4           ;for second run through

        ADD.L   #1,A3   ;inc end ptr to first byte of address
        MOVE.L  A3,A2   ;set start ptr
BTSTGETLASTEND:
        CMP.B   #$00,(A3)+
        BEQ     STOREDATABTST
        BRA     BTSTGETLASTEND


STOREDATABTST:
        SUB.L   #1,A3   ;off by one error
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A6   ;end address in A6
        MOVE.L  (SP)+,D5

BTSTDATALOOP:
        CMP.L   A5,A6
        BLT     READ
        MOVE.B  D5,(A5)+
        BRA     BTSTDATALOOP


READ:
        CMP.L   A4,A6
        BLT     COMPLETE
        CMP.B   (A4)+,D5
        BNE     FAIL
        BRA     READ

FAIL:
        LEA     BTST4,A1
        MOVE.L  #11,D1
        MOVE.L  #0,D0
        TRAP    #15

        LEA     BTST1,A1
        MOVE.L  #1,D0
        MOVE.L  #20,D1
        TRAP    #15

        MOVE.B  D5,D3       ;for subroutine
        JSR     HEXTOASCII
        MOVE.L      A2,A1
        MOVE.L  #0,D0
```

2.2.10 Block Search

2.2.10.1 Algorithm and Flowchart

2.2.10.2 Assembly Code

```


        LEA     BTST2,A1
        MOVE.L  #1,D0
        MOVE.L  #17,D1
        TRAP    #15


        SUB.L   #1,A4       ;go back to address that failed
        MOVE.B  (A4),D3
        JSR     HEXTOASCII  ;convert for output
        MOVE.L      A2,A1
        MOVE.L  #0,D0
        SUBA.L  A2,A3       ;number of bytes
        MOVE.L  A3,D1
        TRAP    #15

        LEA     BTST5,A1
        MOVE.L  #27,D1
        MOVE.B  #1,D0
        TRAP    #15
        MOVE.L  A4,D3
        JSR     HEXTOASCII
        MOVE.L      A2,A1
        MOVE.L  #0,D0
        SUBA.L  A2,A3    ;number of bytes
        MOVE.L  A3,D1
        TRAP    #15



COMPLETE:

        LEA     BTST3,A1
        MOVE.L  #18,D1
        MOVE.L  #0,D0
        TRAP    #15
        BRA RESTORE

*    ---------------------------------------------------------------------------------------------------------


BSCH:
        ADD.L   #1,A1    ;start of data
        MOVE.L  A1,A2    ;set up bac ptr

BSCHENDDATA:
        CMP.B   #$20,(A2)+
        BEQ     BSCHFIRSTADD
        BRA     BSCHENDDATA


BSCHFIRSTADD:
        SUB.L   #1,A2
        MOVE.L  A2,A3
        MOVE.L  A1,A2
        JSR     ASCII_ADDRESS
        SUB.L   A1,A3    ;see how many bytes
        MOVE.L  A3,D6    ;store byte/word/long in D6
        ADD.L   #1,A2    ;set up for start of next address
        MOVE.L  A2,A3    ;set up for end ptr
        MOVE.L  D5,-(SP)     ;save data to stack


BSCHFADDEND:
        CMP.B   #$20,(A3)+
        BEQ     BSCHSECONDADD
        BRA     BSCHFADDEND


BSCHSECONDADD:
        SUB.L   #1,A3    ;off by one
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A5    ;first address destination
        ADD.L   #1,A3    ;start it at next address
        MOVE.L  A3,A2    ; set up for next address


BSCHSECONDFIND:
        CMP.B   #$00,(A3)+
        BEQ     TESTOP
        BRA     BSCHSECONDFIND


TESTOP:
        SUB.L   #1,A3    ;off by one
        JSR     ASCII_ADDRESS
        MOVE.L  D5,A6    ;end address at A6
        MOVE.L  (SP)+,D5     ;restore data
        CMP.B   #2,D6
        BEQ     BYTEBSCH
        CMP.B   #4,D6
        BEQ     WORDBSCH
        CMP.B   #8,D6
        BEQ     LONGBSCH
        BRA     ERRORSR

BYTEBSCH:
        CMP.L   A5,A6
        BLT     ENDBSCH
        CMP.B   (A5)+,D5
        BEQ     FOUNDB
        BRA     BYTEBSCH

WORDBSCH:
        CMP.L   A5,A6
        BLT     ENDBSCH
        CMP.W   (A5)+,D5
        BEQ     FOUNDW
        BRA     WORDBSCH
```

```

LONGBSCH:
            CMP.L    A5,A6
            BLT      ENDBSCH
            CMP.L    (A5)+,D5
            BEQ      FOUNDL
            BRA      LONGBSCH
```

2.2.11 Go

2.2.11.1 Algorithm and Flowchart

2.2.11.2 Assembly Code

```
            BRA      SUCCESSTEXT
FOUNDW:
            SUB.L    #2,A5
            MOVE.W   (A5),D3
            BRA      SUCCESSTEXT
FOUNDL:
            SUB.L    #4,A5
            MOVE.L   (A5),D3

SUCCESSTEXT:
            LEA  BSCH1,A1
            MOVE.L   #6,D1
            MOVE.L   #1,D0
            TRAP     #15
```

2.2.12 Display Formatted Registers

2.2.12.1 Algorithm and Flowchart

2.2.12.2 Assembly Code

```
            MOVE.L   A3,D1     ;how many bytes
            MOVE.L   #0,D0
            TRAP     #15
```

```

        LEA BSCH2,A1
        MOVE.L   #18,D1
        MOVE.L   #1,D0
        TRAP     #15

        MOVE.L   A5,D3
        JSR      HEXTOASCII
        MOVE.L   A2,A1
        SUB.L    A2,A3
        MOVE.L   A3,D1    ;how many bytes
        MOVE.L   #0,D0
        TRAP     #15


ENDBSCH:
        BRA RESTORE

*
    ________________________________________________________________________________


GO:
        MOVE.L   A1,A2    ;setup for hex conversion
        MOVE.L   A2,A3
GGETEND:
        CMP.B    #$00,(A3)+
        BEQ      EXECUTE
        BRA      GGETEND

EXECUTE:
        SUB.L    #1,A3    ;off by one error
        JSR      ASCII_ADDRESS
        MOVE.L   D5,A0
        JSR      (A0)     ;go to program
        **NOTE: THE PROGRAM MUST HAVE RTS OR CONTROL WILL NOT BE
    RETURNED BACK TO MONITOR441!!!**
        BRA RESTORE

*
    ________________________________________________________________________________


DF:     *Registers have already been saved to STACK, just need to
    pop them off first*
```

```
        *Stack looks like this*

        *--------------*
        *|D0-D7/A0-A6|*
        *|     USP    |*
        *|     SR     |*
        *|     SSP    |*
        *|     PC     |*
        *--------------*
        *I should've used loops for efficiency but runtime is not a design constraint*
        *Maybe fix this in the future?*

          *-----------D0------------*
        LEA         RD0,A1
        MOVE.L      #4,D1
        MOVE.L      #1,D0
        TRAP        #15
        MOVE.L      (SP)+,D3
        JSR         HEXTOASCII
        MOVE.L      A2,A1
        SUB.L       A3,A2
        MOVE.L      A2,D2
        CMP.L       #-8,D2
        BEQ         D0DONTWORRY
D0ACCOUNTFORZEROS:
         ADDI.L     #8,D2
         SUB.L      D2,A1
D0DONTWORRY:
        MOVE.L      #0,D0
        MOVE.L      #8,D1
        TRAP        #15

           *-----------D1------------*
        LEA         RD1,A1
        MOVE.L      #4,D1
        MOVE.L      #1,D0
        TRAP        #15
        MOVE.L      (SP)+,D3
        JSR         HEXTOASCII
        MOVE.L      A2,A1
        SUB.L       A3,A2
        MOVE.L      A2,D2
        CMP.L       #-8,D2
        BEQ         D1DONTWORRY
```

```
D1ACCOUNTFORZEROS:
                ADDI.L  #8,D2
                SUB.L   D2,A1
D1DONTWORRY:
              MOVE.L    #0,D0
              MOVE.L    #8,D1
              TRAP      #15

                      *-------------D2--------------*
              LEA       RD2,A1
              MOVE.L    #4,D1
              MOVE.L    #1,D0
              TRAP      #15
              MOVE.L    (SP)+,D3
              JSR       HEXTOASCII
              MOVE.L    A2,A1
              SUB.L     A3,A2
              MOVE.L    A2,D2
              CMP.L     #-8,D2
              BEQ       D2DONTWORRY
D2ACCOUNTFORZEROS:
                ADDI.L  #8,D2
                SUB.L   D2,A1
D2DONTWORRY:
              MOVE.L    #0,D0
              MOVE.L    #8,D1
              TRAP      #15

                        *-------------D3--------------*
              LEA       RD3,A1
              MOVE.L    #4,D1
              MOVE.L    #1,D0
              TRAP      #15
              MOVE.L    (SP)+,D3
              JSR       HEXTOASCII
              MOVE.L    A2,A1
              SUB.L     A3,A2
              MOVE.L    A2,D2
              CMP.L     #-8,D2
              BEQ       D3DONTWORRY
D3ACCOUNTFORZEROS:
                ADDI.L  #8,D2
                SUB.L   D2,A1
D3DONTWORRY:
              MOVE.L    #0,D0
```

```
        MOVE.L  #8,D1
        TRAP    #15

                *------------D4--------------*
        LEA     RD4,A1
        MOVE.L  #4,D1
        MOVE.L  #1,D0
        TRAP    #15
        MOVE.L  (SP)+,D3
        JSR     HEXTOASCII
        MOVE.L  A2,A1
        SUB.L   A3,A2
        MOVE.L  A2,D2
        CMP.L   #-8,D2
        BEQ     D4DONTWORRY
D4ACCOUNTFORZEROS:
          ADDI.L  #8,D2
          SUB.L   D2,A1
D4DONTWORRY:
        MOVE.L  #0,D0
        MOVE.L  #8,D1
        TRAP    #15

        *------------D5--------------*
        LEA     RD5,A1
        MOVE.L  #4,D1
        MOVE.L  #1,D0
        TRAP    #15
        MOVE.L  (SP)+,D3
        JSR     HEXTOASCII
        MOVE.L  A2,A1
        SUB.L   A3,A2
        MOVE.L  A2,D2
        CMP.L   #-8,D2
        BEQ     D5DONTWORRY
D5ACCOUNTFORZEROS:
          ADDI.L  #8,D2
          SUB.L   D2,A1
D5DONTWORRY:
        MOVE.L  #0,D0
        MOVE.L  #8,D1
        TRAP    #15

          *------------D6--------------*
        LEA     RD6,A1
```

```
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L      A2,A1
            SUB.L       A3,A2
            MOVE.L      A2,D2
            CMP.L       #-8,D2
            BEQ         D6DONTWORRY
D6ACCOUNTFORZEROS:
              ADDI.L    #8,D2
              SUB.L     D2,A1
D6DONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #8,D1
            TRAP        #15

              *--------------D7---------------*
            LEA         RD7,A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L      A2,A1
            SUB.L       A3,A2
            MOVE.L      A2,D2
            CMP.L       #-8,D2
            BEQ         D7DONTWORRY
D7ACCOUNTFORZEROS:
              ADDI.L    #8,D2
              SUB.L     D2,A1
D7DONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #8,D1
            TRAP        #15

     *--------------A0---------------*
            LEA         RA0,A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L    A2,A1
            SUB.L     A3,A2
            MOVE.L    A2,D2
            CMP.L     #-8,D2
            BEQ       A0DONTWORRY
A0ACCOUNTFORZEROS:
              ADDI.L  #8,D2
              SUB.L   D2,A1
A0DONTWORRY:
            MOVE.L    #0,D0
            MOVE.L    #8,D1
            TRAP      #15

                 *-------------A1--------------*
            LEA       RA1,A1
            MOVE.L    #4,D1
            MOVE.L    #1,D0
            TRAP      #15
            MOVE.L    (SP)+,D3
            JSR       HEXTOASCII
            MOVE.L    A2,A1
            SUB.L     A3,A2
            MOVE.L    A2,D2
            CMP.L     #-8,D2
            BEQ       A1DONTWORRY
A1ACCOUNTFORZEROS:
              ADDI.L  #8,D2
              SUB.L   D2,A1
A1DONTWORRY:
            MOVE.L    #0,D0
            MOVE.L    #8,D1
            TRAP      #15

              *-------------A2--------------*
            LEA       RA2,A1
            MOVE.L    #4,D1
            MOVE.L    #1,D0
            TRAP      #15
            MOVE.L    (SP)+,D3
            JSR       HEXTOASCII
            MOVE.L    A2,A1
            SUB.L     A3,A2
            MOVE.L    A2,D2
            CMP.L     #-8,D2
            BEQ       A2DONTWORRY
```

```
A2ACCOUNTFORZEROS:
            ADDI.L    #8,D2
            SUB.L     D2,A1
A2DONTWORRY:
          MOVE.L      #0,D0
          MOVE.L      #8,D1
          TRAP        #15

            *------------A3-------------*
          LEA         RA3,A1
          MOVE.L      #4,D1
          MOVE.L      #1,D0
          TRAP        #15
          MOVE.L      (SP)+,D3
          JSR         HEXTOASCII
          MOVE.L      A2,A1
          SUB.L       A3,A2
          MOVE.L      A2,D2
          CMP.L       #-8,D2
          BEQ         A3DONTWORRY
A3ACCOUNTFORZEROS:
            ADDI.L    #8,D2
            SUB.L     D2,A1
A3DONTWORRY:
          MOVE.L      #0,D0
          MOVE.L      #8,D1
          TRAP        #15

           *------------A4-------------*
          LEA         RA3,A1
          MOVE.L      #4,D1
          MOVE.L      #1,D0
          TRAP        #15
          MOVE.L      (SP)+,D3
          JSR         HEXTOASCII
          MOVE.L      A2,A1
          SUB.L       A3,A2
          MOVE.L      A2,D2
          CMP.L       #-8,D2
          BEQ         A4DONTWORRY
A4ACCOUNTFORZEROS:
            ADDI.L    #8,D2
            SUB.L     D2,A1
A4DONTWORRY:
          MOVE.L      #0,D0
```

```
            MOVE.L      #8,D1
            TRAP        #15

            *-----------A5------------*
            LEA         RA3,A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L      A2,A1
            SUB.L       A3,A2
            MOVE.L      A2,D2
            CMP.L       #-8,D2
            BEQ         A5DONTWORRY
A5ACCOUNTFORZEROS:
              ADDI.L    #8,D2
              SUB.L     D2,A1
A5DONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #8,D1
            TRAP        #15

            *-----------A6------------*
            LEA         RA3,A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L      A2,A1
            SUB.L       A3,A2
            MOVE.L      A2,D2
            CMP.L       #-8,D2
            BEQ         A6DONTWORRY
A6ACCOUNTFORZEROS:
              ADDI.L    #8,D2
              SUB.L     D2,A1
A6DONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #8,D1
            TRAP        #15
        *------HACK---*
        ADD.L   #60,SP   ;should put stack in correct place

```

```
                    *————————USP————————*
            LEA         RUS, A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.L      (SP)+,D3
            JSR         HEXTOASCII
            MOVE.L      A2, A1
            SUB.L       A3, A2
            MOVE.L      A2, D2
            CMP.L       #-8,D2
            BEQ         USPDONTWORRY
USPACCOUNTFORZEROS:
              ADDI.L    #8,D2
              SUB.L     D2, A1
USPDONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #8,D1
            TRAP        #15

                    *————————SR————————*
            LEA         RSR, A1
            MOVE.L      #4,D1
            MOVE.L      #1,D0
            TRAP        #15
            MOVE.W      (SP)+,D3
            MOVE.W      D3, D7      ;temp storage to restore before return
            JSR         HEXTOASCII
            MOVE.L      A2, A1
            SUB.L       A3, A2
            MOVE.L      A2, D2
            CMP.L       #-4,D2
            BEQ         SRDONTWORRY
SRACCOUNTFORZEROS:
              ADDI.L    #4,D2
              SUB.L     D2, A1
SRDONTWORRY:
            MOVE.L      #0,D0
            MOVE.L      #4,D1
            TRAP        #15

       *————————SS/A7————————*
            LEA         RSS, A1
            MOVE.L      #7,D1
            MOVE.L      #1,D0
```

```
        TRAP    #15
        MOVE.L  (SP)+,D3
        JSR     HEXTOASCII
```

2.2.13 Modify Register

2.2.13.1 Algorithm and Flowchart

2.2.13.2 Assembly Code

```
MODIFYREGS:

MRD:
        ADD.L   #1,A1       ;inc
        CMP.B   #$30,(A1)
        BEQ     MRD0
        CMP.B   #$31,(A1)
        BEQ     MRD1
        CMP.B   #$32,(A1)
        BEQ     MRD2
        CMP.B   #$33,(A1)
        BEQ     MRD3
        CMP.B   #$34,(A1)
        BEQ     MRD4
        CMP.B   #$35,(A1)
        BEQ     MRD5
        CMP.B   #$36,(A1)
        BEQ     MRD6
        CMP.B   #$37,(A1)
        BEQ     MRD7
        BRA     ERRORSR

MRA:
        ADD.L   #1,A1       ;inc
        CMP.B   #$30,(A1)
        BEQ     MRA0
        CMP.B   #$31,(A1)
        BEQ     MRA1
        CMP.B   #$32,(A1)
        BEQ     MRA2
        CMP.B   #$33,(A1)
        BEQ     MRA3
        CMP.B   #$34,(A1)
        BEQ     MRA4
        CMP.B   #$35,(A1)
        BEQ     MRA5
        CMP.B   #$36,(A1)
        BEQ     MRA6
        BRA     ERRORSR





MRD0:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP       ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
                                ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP       ;skip saved stack
        SUB.L   #134,SP     ;find data again
        MOVE.L  (SP),D0
        ADD.L   #138,SP     ;go back to original spot
        BRA     SHELL

MRD1:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),D1
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRD2:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),D2
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRD3:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),D3
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRD4:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),D4
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRD5:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP       ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
                                ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP       ;skip saved stack
        SUB.L   #134,SP     ;find data again
        MOVE.L  (SP),D5
        ADD.L   #138,SP     ;go back to original spot
        BRA     SHELL

MRD6:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP       ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
                                ;EASY68k simulator starts in
    supervisor mode
```

```
        MOVE     (SP)+,SR
        ADD.L    #4,SP          ;skip saved stack
        SUB.L    #134,SP        ;find data again
        MOVE.L   (SP),D6
        ADD.L    #138,SP        ;go back to original spot
        BRA      SHELL

MRD7:
        ADD.L    #1,A1
        CMP.B    #$20,(A1)+
        BNE      ERRORSR
        MOVE.L   A1,A2
        MOVE.L   A2,A3
        JSR      MRDFINDDATA
        SUB.L    #1,A3
        JSR      ASCII_ADDRESS    ;convert data to hex
        MOVE.L   D5,-(SP)         ;store it temporarily
        ADD.L    #4,SP          ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L    #4,SP          ;account for USP, it'll fix itself (
    it shouldn't be used)
                                  ;EASY68k simulator starts in
    supervisor mode
        MOVE     (SP)+,SR
       ADD.L    #4,SP          ;skip saved stack
        SUB.L    #134,SP        ;find data again
        MOVE.L   (SP),D7
        ADD.L    #138,SP        ;go back to original spot
        BRA      SHELL

MRA0:
        ADD.L    #1,A1
        CMP.B    #$20,(A1)+
        BNE      ERRORSR
        MOVE.L   A1,A2
        MOVE.L   A2,A3
        JSR      MRDFINDDATA
        SUB.L    #1,A3
        JSR      ASCII_ADDRESS    ;convert data to hex
        MOVE.L   D5,-(SP)         ;store it temporarily
        ADD.L    #4,SP          ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (it shouldn't be used)
                                  ;EASY68k simulator starts in supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),A0
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL
MRA1:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (it shouldn't be used)
                                  ;EASY68k simulator starts in supervisor mode
        MOVE    (SP)+,SR
       ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),A1
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRA2:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP       ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
                                ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP       ;skip saved stack
        SUB.L   #134,SP     ;find data again
        MOVE.L  (SP),A2
        ADD.L   #138,SP     ;go back to original spot
        BRA     SHELL

MRA3:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS   ;convert data to hex
        MOVE.L  D5,-(SP)        ;store it temporarily
        ADD.L   #4,SP       ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
                                ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP       ;skip saved stack
        SUB.L   #134,SP     ;find data again
        MOVE.L  (SP),A3
        ADD.L   #138,SP     ;go back to original spot
        BRA     SHELL

MRA4:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
        ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),A4
        ADD.L   #138,SP      ;go back to original spot
        BRA     SHELL

MRA5:
        ADD.L   #1,A1
        CMP.B   #$20,(A1)+
        BNE     ERRORSR
        MOVE.L  A1,A2
        MOVE.L  A2,A3
        JSR     MRDFINDDATA
        SUB.L   #1,A3
        JSR     ASCII_ADDRESS    ;convert data to hex
        MOVE.L  D5,-(SP)         ;store it temporarily
        ADD.L   #4,SP        ;dont lose data
        MOVEM.L (SP)+,D0-D7/A0-A6
        MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                 ;EASY68k simulator starts in
    supervisor mode
        MOVE    (SP)+,SR
      ADD.L   #4,SP        ;skip saved stack
        SUB.L   #134,SP      ;find data again
        MOVE.L  (SP),A5
        ADD.L   #138,SP      ;go back to original spot
        BRA      SHELL

MRA6:
        ADD.L    #1,A1
        CMP.B    #$20,(A1)+
        BNE      ERRORSR
        MOVE.L   A1,A2
        MOVE.L   A2,A3
        JSR      MRDFINDDATA
        SUB.L    #1,A3
        JSR      ASCII_ADDRESS    ;convert data to hex
        MOVE.L   D5,-(SP)         ;store it temporarily
        ADD.L    #4,SP        ;dont lose data
        MOVEM.L  (SP)+,D0-D7/A0-A6
        MOVEM.L  (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
        ADD.L    #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
                                  ;EASY68k simulator starts in
    supervisor mode
        MOVE     (SP)+,SR
        ADD.L    #4,SP        ;skip saved stack
        SUB.L    #134,SP      ;find data again
        MOVE.L   (SP),A6
        ADD.L    #138,SP      ;go back to original spot
        BRA      SHELL

MRDFINDDATA:
        CMP.B    #$00,(A3)+
        BEQ      GOBACK
        BRA      MRDFINDDATA
GOBACK: RTS


        BRA RESTORE
```

2.2.14 Echo

2.2.14.1 Algorithm and Flowchart

2.2.14.2 Assembly Code

```
ECHO: *What terminal DOESN'T have echo?*

        MOVE.L  A1,A2   ;setup to find end of string
EEND:   CMP.B   #$00,(A2)+
        BEQ     EFOUND
        BRA     EEND
EFOUND:
        SUB.L   #1,A2   ;off by one
        SUB.L   A1,A2   ;find out how many bytes
        MOVE.L  A2,D1   ;place it for trap function
        MOVE.L  #0,D0
        TRAP    #15

        BRA RESTORE
```

2.3 Subroutines

2.3.1 Hexadecimal to ASCII

2.3.1.1 Algorithm

2.3.1.2 Assembly Code

```
BCD_TO_HEX:     *Number passed via D3 accepts BYTE ONLY*
                MOVE.L  D3,D4
                MOVE.L  D3,D5
                ANDI.L  #240,D4 ;upper byte
                ANDI.L  #15,D5  ;lower byte
                ROR.L   #4,D4   ;get bits into correct place
                MULU    #10,D4  ;multiply by its tens place
                CLR.L   D3
                ADD.L   D4,D3
                ADD.L   D5,D3
                RTS
*
    ------------------------------------------------------------------------------------------

ASCII_ADDRESS:   *Address to be converted from ascii to hex
    passed through A2 and A3*
                 *Returned in D5
                        *
                CLR.L   D3
                CLR.L   D5
                MOVE.L   A2,D1
                MOVE.L   A3,D0
                SUB.L   D1,D0    ;store the difference in D0
                MOVE.L  #0,D4    ;set up 10's place counter
                SUBI.L  #1,D0
PLACECOUNTER:   CMP     #0,D0
                BEQ     CONVERTADDRESS
                ADDI.L  #4,D4
                SUBI.L  #1,D0
                BRA     PLACECOUNTER
CONVERTADDRESS  CMP     A2,A3
                BEQ     ADDRESSDONE
                CLR.L   D3
                MOVE.B  (A2)+,D3
                MOVEM.L A2-A3/D0-D2/D4-D5,-(SP)    ;so regs dont
    get destroyed
                JSR     ASCII_TO_BCD
                JSR     BCD_TO_HEX
                MOVEM.L (SP)+,A2-A3/D0-D2/D4-D5
                ROL.L     D4,D3
                SUBI.L  #4,D4
                ADD.L     D3,D5   ;get total
                BRA     CONVERTADDRESS
ADDRESSDONE     RTS
```

2.3.2 ASCII to Hexadecimal

2.3.2.1 Algorithm

2.3.2.2 Assembly Code

```
    MOVE.L #13,D0
    TRAP #15
    MOVEM.L (SP)+,A1/D0
    MOVE.L #$01000000,SP
    BRA SHELL



***************************COMMON SUBROUTINES NEEDED
    *********************************
```

```


*
 ----------------------------------------------------------------------------------------------------

ERRORSR:          LEA     ERROR, A1      ;load  message
                  MOVE.W   #44,D1
                  MOVE.L   #0,D0
                  TRAP      #15
                  BRA       RESTORE
*
 ----------------------------------------------------------------------------------------------------

ASCII_TO_BCD:     *Number  passed  via  D3  byte  size  only(to  be
    expected)*
                  CMP #$46 ,D3
                  BGT ERRORSR
                  CMP #$40 ,D3
                  BGT UPPER
                  SUBI.L   #$30 ,D3
                  RTS
UPPER:            SUBI.L   #$31 ,D3 ;If  ASCII  number  is  A-F
                  RTS
```

2.3.3 BCD to Hexadecimal

2.3.3.1 Algorithm

2.3.3.2 Assembly Code

```
FLERR:
    MOVEM.L  A1/D0,-(SP)
    LEA  FLERR_TEXT, A1
    MOVE.L  #13,D0
    TRAP  #15
    MOVEM.L  (SP)+,A1/D0
    MOVE.L  #$01000000 ,SP
    BRA  SHELL

CHKERR:
    MOVEM.L  A1/D0,-(SP)
```

2.3.4 ASCII to BCD

2.3.4.1 Algorithm

2.3.4.2 Assembly Code

```
    BRA SHELL

ALERR:
    MOVEM.L  A1/D0,-(SP)
    LEA  ALERR_TEXT,A1
    MOVE.L  #13,D0
    TRAP  #15
    MOVEM.L  (SP)+,A1/D0
    MOVE.L #$01000000 ,SP
```

2.4 Exception Handlers

The Monitor441 program uses custom exception handlers. They are loaded using the source code:

```
        *Load custom exceptions*
        LEA BERR,A1 ;init exception handlers
        MOVE.L  A1,$8
        LEA  AERR,A1
        MOVE.L  A1,$C
        LEA  IERR,A1
        MOVE.L  A1,$10
        LEA  ZERR,A1
        MOVE.L  A1,$14
        LEA  CHKERR,A1
        MOVE.L   A1,$18
        LEA  PERR,A1
        MOVE.L  A1,$20
        LEA  ALERR,A1
        MOVE.L  A1,$28
        LEA  FLERR,A1
        MOVE.L  A1,$2C
        MOVEM.L  (SP)+,D0-D2/A1   ;restore any preset values
```

2.4.1 Bus Error Exception

2.4.1.1 Algorithm and Flowchart

2.4.1.2 Assembly Code

```
          SUB.L     D2,A1
SSDONTWORRY:
         MOVE.L     #0,D0
         MOVE.L     #8,D1
         TRAP       #15

     *--------------PC---------------*
         LEA        RPC,A1
         MOVE.L     #4,D1
         MOVE.L     #1,D0
         TRAP       #15
         MOVE.L     (SP)+,D3
         JSR        HEXTOASCII
         MOVE.L     A2,A1
         SUB.L      A3,A2
         MOVE.L     A2,D2
         CMP.L      #-8,D2
         BEQ        PCDONTWORRY
PCACCOUNTFORZEROS:
          ADDI.L    #8,D2
          SUB.L     D2,A1
PCDONTWORRY:
         MOVE.L     #0,D0
         MOVE.L     #8,D1
         TRAP       #15

    *----DF HACK RESTORE----*
    MOVE.W     D7,-(SP)
    ADD.L      #-72,SP
    MOVEM.L    (SP)+,D0-D7/A0-A6
    ADD.L      #12,SP    ;go back to original value
    MOVE.W     (SP)+,SR

         BRA SHELL

*
    ----------------------------------------------------------------------------------------------------
```

2.4.2 Address Error Exception

2.4.2.1 Algorithm and Flowchart

2.4.2.2 Assembly Code

```
*
    ------------------------------------------------------------------------------------------

BERR:
            MOVEM.L A1–A3/D0–D1,-(SP)
            LEA     BERR_TEXT,A1
            MOVE.L  #13,D0
            TRAP    #15
            LEA     SSW,A1
            MOVE.L  #14,D0
            TRAP    #15
            MOVE.W  (20,SP),D3
            JSR     HEXTOASCII
            SUB.L   #4,A3
            MOVEA.L A3,A1
            MOVE.L  #4,D1
            MOVE.L  #0,D0
            TRAP    #15
            LEA     BA,A1
            MOVE.L  #14,D0
            TRAP    #15
            MOVE.L  (22,SP),D3
            JSR     HEXTOASCII
            SUB.L   #8,A3
            MOVEA.L A3,A1
            MOVE.L  #8,D1
            MOVE.L  #0,D0
            TRAP    #15
            LEA     IR,A1
            MOVE.L  #14,D0
            TRAP    #15
            MOVE.W  (26,SP),D3
            JSR     HEXTOASCII
```

```
        SUB.L     #4,A3
        MOVEA.L   A3,A1
        MOVE.L    #4,D1
        MOVE.L    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1-A3/D0-D1
```

2.4.3 Illegal Instruction Error Exception

2.4.3.1 Algorithm and Flowchart

2.4.3.2 Assembly Code

```
        BRA       SHELL

AERR:
        MOVEM.L   A1-A3/D0-D1,-(SP)
        LEA       AERR_TEXT,A1
        MOVE.L    #13,D0
        TRAP      #15
        LEA       SSW,A1
```

2.4.4 Privilege Violation Error Exception

2.4.4.1 Algorithm and Flowchart

2.4.4.2 Assembly Code

```
        TRAP      #15
        MOVE.W    (20,SP),D3
        JSR       HEXTOASCII
        SUB.L     #4,A3
        MOVEA.L   A3,A1
        MOVE.L    #4,D1
        MOVE.L    #0,D0
        TRAP      #15
```

2.4.5 Divide by Zero Error Exception

2.4.5.1 Algorithm and Flowchart

2.4.5.2 Assembly Code

```
        MOVE.L  #14,D0
        TRAP    #15
        MOVE.L  (22,SP),D3
        JSR     HEXTOASCII
        SUB.L   #8,A3
        MOVEA.L A3,A1
        MOVE.L  #8,D1
        MOVE.L  #0,D0
```

2.4.6 A Line Emulator Error Exception

2.4.6.1 Algorithm and Flowchart

2.4.6.2 Assembly Code

```
        LEA     IR,A1
        MOVE.L  #14,D0
        TRAP    #15
        MOVE.W  (26,SP),D3
        JSR     HEXTOASCII
        SUB.L   #4,A3
        MOVEA.L A3,A1
        MOVE.L  #4,D1
```

2.4.7 F Line Emulator Error Exception

2.4.7.1 Algorithm and Flowchart

2.4.7.2 Assembly Code

```
          TRAP      #15
          MOVEM.L  (SP)+,A1-A3/D0-D1
          MOVE.L   #$01000000,SP     ;reset stack
          BRA       SHELL

IERR:
     MOVEM.L  A1/D0,-(SP)
     LEA  IERR_TEXT,A1
```

2.4.8 Check Instruction Error Exception

2.4.8.1 Algorithm and Flowchart

2.4.8.2 Assembly Code

```
     TRAP  #15
     MOVEM.L  (SP)+,A1/D0
     MOVE.L  #$01000000,SP
     BRA  SHELL

PERR:
     MOVEM.L  A1/D0,-(SP)
     LEA  PERR_TEXT,A1
```

2.5 User Instruction Manual Exception Handlers

2.5.0.3 Algorithm and Flowchart

2.5.0.4 Assembly Code

3 Discussion

4 Feature Suggestions

5 Conclusion

References

[1] test