ECE 441
Microprocessors

Instructor: Dr. Jafar Saniie
Teaching Assistant: Guojun Yang

Final Project Report:
MONITOR PROJECT
10/31/2019

By: Harshitha Panduranga

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

Signature: _____

Table of Contents

Abstract ____ ***2***

1-) Introduction ____ ***2***

2-) Monitor Program ____ *Error! Bookmark not defined.*

2.1-) Command Interpreter ____ **4**

2.1.1-) Block Diagram ____ 4

2.1.2-) 68000 Assembly Code ____ 4

2.2-) Debugger Commands ____ **6**

2.2.1-) HELP ____ 6

2.2.2-) MDSP ____ 11

2.2.3-) MM ____ 14

2.2.4-) MS ____ 16

2.2.5-) BF ____ 18

2.2.6-) BMOV ____ 20

2.2.7-) BTST ____ 21

2.2.8-) BSCH ____ 23

2.2.9-) GO ____ 25

2.2.10-) DF ____ 26

2.2.11-) EXIT ____ 36

2.2.12-) HEXSQR ____ 38

2.2.13-) EVENODD ____ 38

2.3-)Exception Handlers ____ **40**

2.3.1-) Bus Error Exception ____ 40

2.3.2-) Address Error Exception ____ 42

2.3.3-) Illegal Instruction Exception ____ 43

2.3.4-) Privilege Violation Exception ____ 44

2.3.5-) Divide by Zero Exception ____ 45

2.3.6-) Line A and Line F Emulators ____ 46

2.3.7-) Check Error Exception ____ 47

2.4-)User Instructional Manual Exception Handlers ____ **47**

2.4.1-) Help Menu ____ 48

3-) Discussion ____ ***49***

4-) Feature Suggestions ____ ***49***

6-) Conclusions ____ ***50***

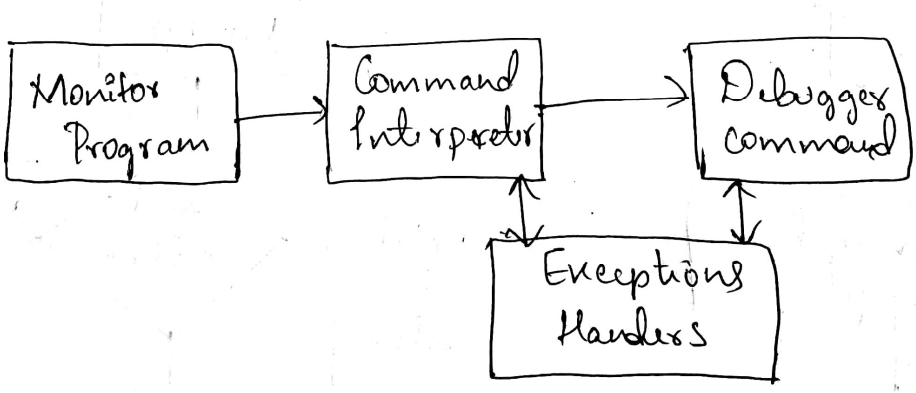
7-) References ____ ***51***

Abstract

The monitor program allows the user to interact with MC68000 processor and its memory. The report consists of 14 debugger commands and 8 exception handling routines with their implementation in the monitor program. Flowcharts and algorithms are provided for all the debugger commands and exception handling routines. In addition to this, instruction manual is provided to the user at the end of the report.

1-) Introduction

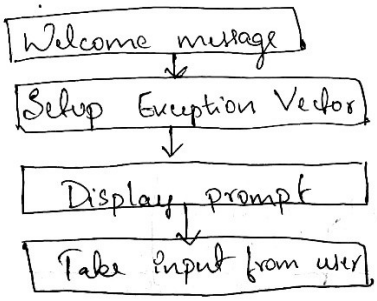
The monitor project has been implemented using EASY68K software. The main goal of the project was to build a system that acts like TUTOR software. The project implemented 13 debugger commands and 8 exception handling routines running similar to the TUTOR software. Based on the input taken from the user, the program branches to the corresponding subroutine and upon execution of the subroutine, the control is given back to the main program where the user can enter another command. The program is designed in such a way that, the user can terminate the program anytime.



2-) Monitor Program

The Monitor program begins at the memory location $1000. Based on the input provided by the user, the program branches to corresponding subroutines. The program starts by printing welcome message on the command prompt. Stack is initialized at memory location $3000. The vector table of exception handlers will be modified as per the address of the subroutines written

by the user for each exception handlers. After initializing the system this way, the control is given to command interpreter. The command prompt displays MONITOR441:> and waits for the user to enter a command. When the user inputs a command, the command will be interpreted as to execute the respective subroutine.

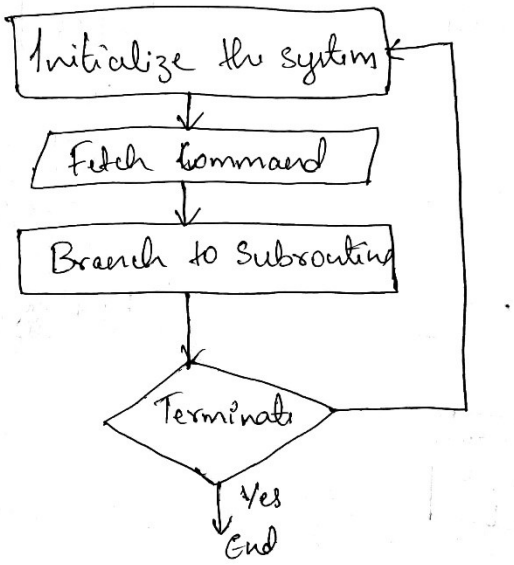


2.1-) Command Interpreter

The Command Interpreter is used to recognize the command given as input by the user. The command interpreter compares the command with the look up table and branches to subroutine to execute this command. When the input given by the user does not match any of the commands stored at the look up table, the monitor throws an error message.

2.1.1-) Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user and store it in A1 // the input entered in the previous step is stored here
If A1 = Null
Return to Subroutine
Else
Compare input with command lookup table // predefined to compare commands
If match found
Branch to Subroutine // branch to execute command
Else
Print error message // print invalid input
Return to Subroutine



2.1.2-) Command Interpreter Assembly Code

```
INTERPRETER:
    ;Check if input buffer is empty
    LEA INPUT_BUFF,A1
    CMPI.B #NULL,(A1)
    BEQ INTERPRETER_END

    ;Check if it's HELP command
    LEA MSG_CMD_HELP,A5
    LEA MSG_CMD_HELP_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HELP

    ;Block search
    LEA MSG_CMD_BSCH,A5
    LEA MSG_CMD_BSCH_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ BSCH

    ;TO display memory
    LEA MSG_CMD_MDSP,A5
    LEA MSG_CMD_MDSP_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ MDSP

    ;To modify memory
    LEA MSG_CMD_MM,A5
    LEA MSG_CMD_MM_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
```

```
    BEQ MM

    ;To set memory
    LEA MSG_CMD_MS,A5
    LEA MSG_CMD_MS_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ MS

    ;To fill block of memory
    LEA MSG_CMD_BF,A5
    LEA MSG_CMD_BF_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ BF

    ;TO move block of memory
    LEA MSG_CMD_BMOV,A5
    LEA MSG_CMD_BMOV_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ BMOV

   ;To perform destructive test on memory
    LEA MSG_CMD_BTST,A5
    LEA MSG_CMD_BTST_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ BTST


    ;To start execution from given address
    LEA MSG_CMD_GO,A5
    LEA MSG_CMD_GO_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ GO

    ;To display formatted register
    LEA MSG_CMD_DF,A5
    LEA MSG_CMD_DF_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ DF

    LEA MSG_CMD_HEXSQR,A5
    LEA MSG_CMD_HEXSQR_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HEXSQR

    LEA MSG_CMD_EVENODD,A5
    LEA MSG_CMD_EVENODD_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ EVENODD
```

```
    ;To terminate the monitor program
    LEA MSG_CMD_EXIT,A5
    LEA MSG_CMD_EXIT_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ EXIT

    ;Invalid input
    BSR SYNTAX_ER
INTERPRETER_END:
    RTS
```

2.2-) Debugger Commands

The monitor project has implemented a total of 13 debugger commands out of which 11 commands are predefined according to the system requirement and 2 other commands are developer custom commands. The detailed explanation of each command along with algorithm and flowchart is shown below.

2.2.1-) HELP

HELP command lists all the available debugger commands. Based on the requirement, the user can choose one of those commands. Upon choosing one of the debugger commands under HELP, the information regarding the function and usage of the command will be displayed with syntax.

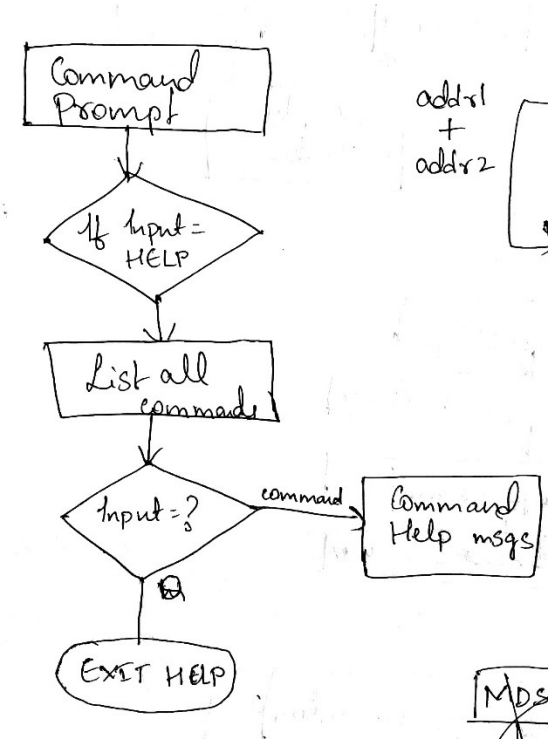
2.2.1.1-) HELP Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = HELP
List all debugger commands
Accept input again from the user
If
Input = debugger command
Branch to respective help subroutine
Else
Print error message

Go back to command interpreter

Else

Compare input with other commands and proceed



2.2.1.2-) Debugger HELP Assembly Code

```
HELP:
    BSR STORE_HIS
    LEA MSG_HELP,A5
    LEA MSG_HELP_ED,A6
    BSR PRINT
    LEA MSG_CMD_FST,A5
    LEA MSG_CMD_LST,A6
    BSR PRINT
HELP_LOOP:
    ;Print help console prompt
    LEA MSG_CMD_HELP,A5
    LEA MSG_CMD_HELP_ED,A6
    MOVE.B #LARGER,(A6)+
    BSR PRINT_C

    ;User input command to be displayed
    BSR INPUT
    LEA INPUT_BUFF,A1

    ;Check if buffer is empty
```

```
CMPI.B #NULL,(A1)
BEQ HELP_LOOP

;To display memory
LEA MSG_CMD_MDSP,A5
LEA MSG_CMD_MDSP_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_MDSP

;To modify memory
LEA MSG_CMD_MM,A5
LEA MSG_CMD_MM_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_MM

;To set memory
LEA MSG_CMD_MS,A5
LEA MSG_CMD_MS_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_MS

;To fill block of memory
LEA MSG_CMD_BF,A5
LEA MSG_CMD_BF_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_BF

;To move block of memory
LEA MSG_CMD_BMOV,A5
LEA MSG_CMD_BMOV_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_BMOV

;To test block of memory
LEA MSG_CMD_BTST,A5
LEA MSG_CMD_BTST_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_BTST

;To do block search
LEA MSG_CMD_BSCH,A5
LEA MSG_CMD_BSCH_ED,A6
LEA INPUT_BUFF,A1
BSR COMPARE
BEQ HELP_BSCH

;To start execution from given address
LEA MSG_CMD_GO,A5
LEA MSG_CMD_GO_ED,A6
LEA INPUT_BUFF,A1
```

```
    BSR COMPARE
    BEQ HELP_GO

    ;To display formatted registers
    LEA MSG_CMD_DF,A5
    LEA MSG_CMD_DF_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HELP_DF



    ;To exit monitor program
    LEA MSG_CMD_EXIT,A5
    LEA MSG_CMD_EXIT_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HELP_EXIT

    ;To calculate square of a hex number
    LEA MSG_CMD_HEXSQR,A5
    LEA MSG_CMD_HEXSQR_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HELP_HEXSQR

;To check whether the number is even or odd
    LEA MSG_CMD_EVENODD,A5
    LEA MSG_CMD_EVENODD_ED,A6
    LEA INPUT_BUFF,A1
    BSR COMPARE
    BEQ HELP_EVENODD
    ;Exit help console
    LEA INPUT_BUFF,A1
    CMPI.B #Q_ASC,(A1)
    BEQ HELP_EXIT
    BRA HELP

HELP_EXIT:
          RTS


HELP_MDSP:
    LEA MSG_HELP_MDSP,A5
    LEA MSG_HELP_MDSP_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_MM:
    LEA MSG_HELP_MM,A5
    LEA MSG_HELP_MM_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_MS:
    LEA MSG_HELP_MS,A5
```

```
    LEA MSG_HELP_MS_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_BF:
    LEA MSG_HELP_BF,A5
    LEA MSG_HELP_BF_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_BMOV:
    LEA MSG_HELP_BMOV,A5
    LEA MSG_HELP_BMOV_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_BTST:
    LEA MSG_HELP_BTST,A5
    LEA MSG_HELP_BTST_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_BSCH:
    LEA MSG_HELP_BSCH,A5
    LEA MSG_HELP_BSCH_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_GO:
    LEA MSG_HELP_GO,A5
    LEA MSG_HELP_GO_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_DF:
    LEA MSG_HELP_DF,A5
    LEA MSG_HELP_DF_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_HEXSQR
    LEA MSG_HELP_HEXSQR,A5
    LEA MSG_HELP_HEXSQR_ED,A6
    BSR PRINT
    BRA HELP_LOOP

HELP_EVENODD
    LEA MSG_HELP_EVENODD,A5
    LEA MSG_HELP_EVENODD_ED,A6
    BSR PRINT
    BRA HELP_LOOP
```

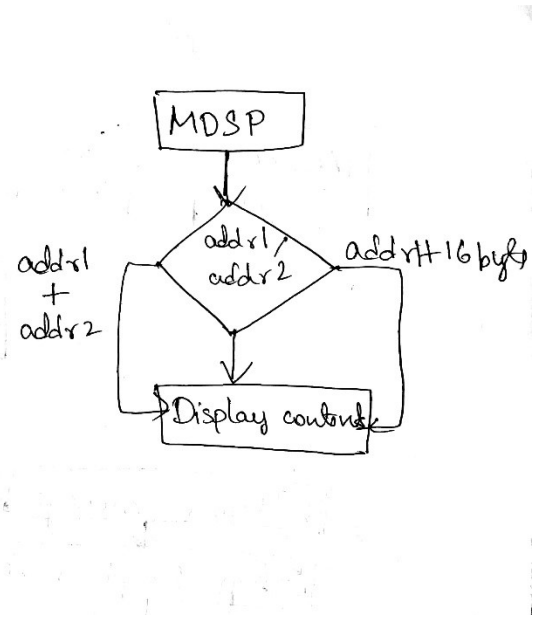
2.2.2-) MDSP (Memory Display)

MDSP command displays the contents of memory. There are two ways of using this command.

1. If the syntax given by the user includes address1 and address2, the command will display the address and contents of memory between <address1> and <address2>
2. If the syntax given by the user includes only one address1 after the command, the system outputs address and contents of memory between <address1> and <address1 + 16 bytes>.

2.2.2.1-) MDSP Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = MDSP
Capture <address1>
If
<address2> = Null
<address2> = <address1 + 16 bytes>
Else
Capture <address2>
For <address1> <= <address2>
Display contents of memory from <address1> to <address2>
Go to command interpreter
Else
Compare input with other commands and proceed



2.2.2.2-) MDSP Assembly Code

```
MDSP:
     BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    CMPI.B #NULL,(A1)+
    BEQ TYPE2
    BSR ASCII2HEX
    MOVE.L D0,A4
    MOVE.L A3,A2
    MOVE.L D2,D4
    MOVE.L #4,D5
    BRA DISPLAY

TYPE2:
    MOVE.L A3,A4
    ADD.L #8,A4

    MOVE.L A3,A2

    MOVE.L D2,D4
    MOVE.L #4,D5



DISPLAY:
    MOVE.L A2,D0
    MOVE.L D4,D2
    BSR HEX2ASCII
    ADD.L #4, A6
    MOVE.L -(A6),D1
    SWAP D1
```

```
ROL #8,D1
MOVE.B #6,D0
TRAP #15

ROL #8,D1
MOVE.B #6,D0
TRAP #15

SWAP D1
ROL #8,D1
MOVE.B #6,D0
TRAP #15

ROL #8,D1
MOVE.B #6,D0
TRAP #15

 ;PRINT SEMICOLON
MOVEA.L #SEMI,A1
MOVE.B #14,D0
TRAP #15

;PRINT CONTENT
CLR.L D0
MOVE.L D5,D2
MOVE.B (A2),D0
BSR HEX2ASCII
ADD.L #4,A6
MOVE.L (A6),D1


ROR #8,D1
MOVE.B #6,D0
TRAP #15

ROR #8,D1
MOVE.B #6,D0
TRAP #15

;PRINT empty space
MOVEA.L #SPACE1,A1
MOVE.B #13,D0
TRAP #15

ADD.L #1,A2
CMPA.L A2,A4
BGE DISPLAY
BSR MAIN
```

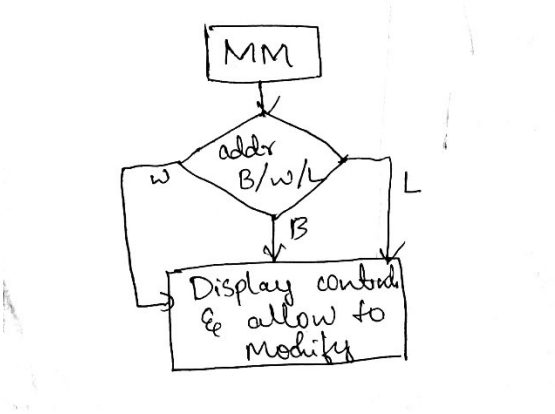
2.2.3-) MM (Memory Modify)

MM (Memory Modify) command is used to display memory and, as required, modify data or enter new data. The size (byte, word, long word) controls the number of bytes displayed

for each address.

2.2.3.1-) MM Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = MM
Capture <address1>, <address2> and size
If size = B
Display and modify data in bytes
Elseif size = W
Display and modify data in word length
Elseif size = L
Display and modify data in longword
Accept the input from user and modify data in the memory
Do this from <address1> to <address2>
After executing return to main program
Else
Compare input with other commands and proceed



2.2.3.2-) MM Assembly Code

```
MM:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
*    BSR ASCII2HEX
```

```
*     MOVE.L D0,A4
      MOVE.L A3,A2
      MOVE.L D2,D4
       MOVE.L #4,D5        ;Parse the second address
       MOVE.B (A1),D6
      CMP.B #$42,D6
      BEQ MM1
MM1:  MOVE.L A3,D0
       BSR HEX2ASCII
       ADD.L #4, A6
       MOVE.L -(A6),D1
       SWAP D1

      ROL #8,D1
      MOVE.B #6,D0
      TRAP #15

      ROL #8,D1
      MOVE.B #6,D0
      TRAP #15

      SWAP D1
      ROL #8,D1
      MOVE.B #6,D0
      TRAP #15

      ROL #8,D1
      MOVE.B #6,D0
      TRAP #15

       ;PRINT SEMICOLON
      MOVEA.L #SEMI,A1
      MOVE.B #14,D0
      TRAP #15

      ;PRINT CONTENT
      CLR.L D0
      MOVE.L D5,D2
      MOVE.B (A2),D0
      BSR HEX2ASCII
      ADD.L #4,A6
      MOVE.L (A6),D1


      ROR #8,D1
      MOVE.B #6,D0
      TRAP #15

      ROR #8,D1
      MOVE.B #6,D0
      TRAP #15

      MOVEA.L #LINEPROMPT,A1
      MOVE.B #14,D0
      TRAP #15
```

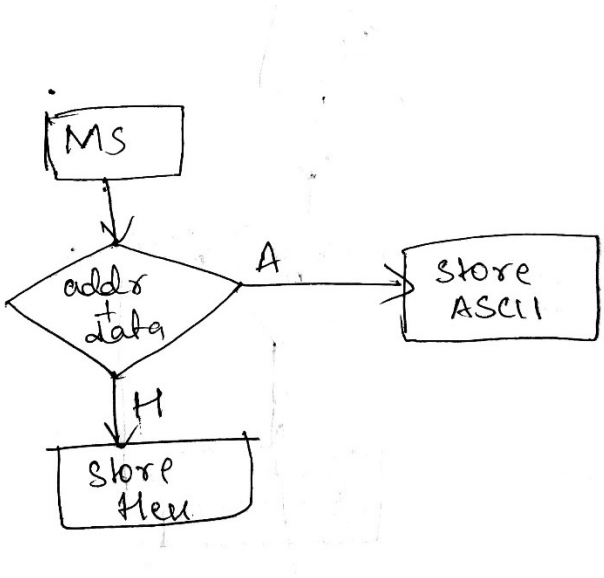
```
LEA $5000,A1
MOVE.B #2,D0
TRAP #15

CMP.B #$2E,(A1)
BEQ MAIN
CMP.B #00,(A1)
BEQ MM1

MOVE (A1)+,(A3)
MOVE.L (A3),(A1)
MOVE (A3),A1
BSR ASCII2HEX
MOVE D0,A3
BRA MM1

BSR MAIN
RTS
```

2.2.4-) MS

The Memory Set (MS) command alters memory by setting data into the address specified. The data can take the form of ASCII string or hexadecimal data.

2.2.4.1-) MS Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = MS
Capture <address1>, data and mode (Hex or ASCII)
If mode = H
The data from the user is captured as Hex value and is stored in the memory
Elseif mode = A
The data from the user is captured as ASCII value and is stored in the memory
The address will contain the data entered
After executing return to main program
Else
Compare input with other commands and proceed



2.2.4.2-) MS Assembly Code

```
MS: BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address IN A3
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,D4        ;Parse the string IN D4
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    CMPI.B #'H',(A1)
    BEQ MS_HEX
    CMPI.B #'A',(A1)
    BEQ MS_ASCII
    LEA ERR_MS,A1
    MOVE.B #14,D0
    TRAP #15
    LEA NEXTLINE,A1
    MOVE.B #14,D0
    TRAP #15
    BRA MS_EXIT

MS_HEX:
    MOVE.L D4,(A3)
    LEA SUCCESS_MS,A1
    MOVE.B #14,D0
    TRAP #15
    LEA NEXTLINE,A1
    MOVE.B #14,D0
    TRAP #15
    BRA MS_EXIT
```

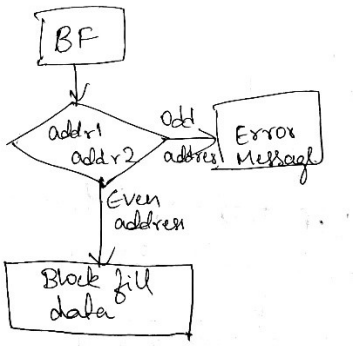
```
MS_ASCII: MOVE.L D4,D0
        BSR HEX2ASCII
        MOVE.L (A6),(A3)
        LEA SUCCESS_MS,A1
        MOVE.B #14,D0
        TRAP #15
        LEA NEXTLINE,A1
        MOVE.B #14,D0
        TRAP #15
        BRA MS_EXIT
MS_EXIT: BRA MAIN
```

2.2.5-) BF

The Block Fill (BF) command fills memory starting with the word boundary <address1> through <address2>. Both <address1> and <address2> must be even addresses. This command only fills with a word-size (2 bytes) data pattern. If an entire word-size data pattern is not entered, the pattern is right justified and leading zeros are inserted.

2.2.5.1-) BF Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = BF
Capture <address1>, <address2> and data
If
<address1> or <address2> is Odd
Prompt the user to provide even address through error message
Else
Fill the data provided at the input from <address1> to <address2>
After executing, return to main program
Else
Compare input with other commands and proceed



2.2.5.2-) BF Assembly Code

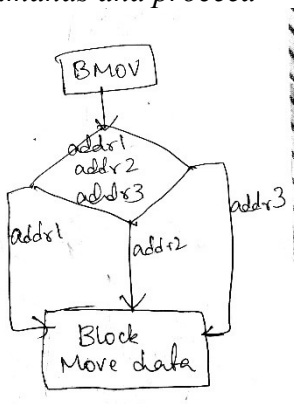
```
BF:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    MOVE.L A3,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A4        ;Parse the second address
    MOVE.L A4,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD
    CMPA.L A4,A3        ;Check if the first address is smalller
    BGE SYNTAX_ER
    ADD.L #1,A4
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A5

BF1:
    MOVE.W A5,(A3)+
    CMPA.L A3,A4
    BGT BF1
    BRA MAIN
ODD:
    LEA EVEN,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    BSR MAIN
```

2.2.6-) BMOV

The Block Move (BMOV) command is used to move (duplicate) blocks of memory from one area to another.

2.2.6.1-) BMOV Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = BMOV
Capture <address1>, <address2> and <address3>
For <address1> < = <address2>
Copy contents of memory and transfer it to <address3> and increment increment <address1> and transfer this to incremented <address3>
Continue till <address1> = <address2>
After executing, return to main program
Else
Compare input with other commands and proceed



2.2.6.2-) BMOV Assembly Code

```
BMOV:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    MOVE.L A3,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD
    CMPI.B #SPACE,(A1)+
```

```
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A4        ;Parse the second address
    MOVE.L A4,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD

    CMPA.L A4,A3        ;Check if the first address is smalller
    BGE SYNTAX_ER
    ADD.L #1,A4
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A5
    MOVE.L A5,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD

BMOV1:
    MOVE.L (A3)+,(A5)+
    CMPA.L A3,A4
    BGT BMOV1
    BRA MAIN
```

2.2.7-) BTST

1. It is similar to 2.2.1 The Block Test (BT) command is a destructive test of a block of memory beginning at <address1> through <address2>. If this test runs to completion without detecting errors and display a message that no error was detected.
2. If memory problems are found, a message is displayed indicating the address, the data stored, and the data read of the failing memory.

2.2.7.1-) BTST Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = BTST
Capture <address1> and <address2>
Consider <address1> and store predefined data in it
Now read the memory and compare with the stored value
If
Data stored and read are same, go to next location

Else

Print the address, data stored and data read from the memory

Continue storing and reading data till <address2>

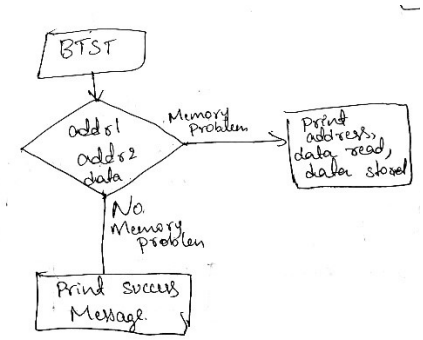
If

No memory problems found, print success message

Return to main program

Else

Compare input with other commands and proceed



2.2.7.2-) BTST Assembly Code

```
BTST:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    MOVE.L A3,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A4         ;Parse the second address
    MOVE.L A4,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODD
    CMPA.L A4,A3         ;Check if the first address is smalller
    BGE SYNTAX_ER
    ADD.L #1,A4
    MOVE.L #$0101,A5

BTST1:
    MOVE.W A5,(A3)   ;MOVE THE WORD TO START
    MOVE.W (A3)+,D5  ;READ THE WORD
```

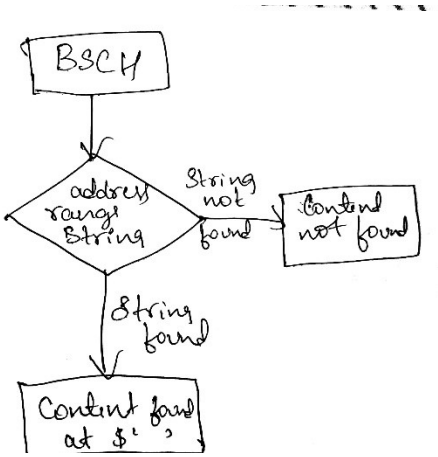
```
    CMP.W A5,D5
    BNE BTSTERROR   ;IF NOT EQUAL GO TO SUBROUTINE
    CMPA.L A3,A4
    BGT BTST1
    LEA MSG_BTST_SUCCESS,A5
    LEA MSG_BTST_SUCCESS_ED,A6
    BSR PRINT
    BRA MAIN

BTSTERROR:
    LEA MSG_BTST_FAILD,A5
    LEA MSG_BTST_FAILD_ED,A6
    BSR PRINT
    BRA MAIN
```

2.2.8-) BSCH

The BSCH (Block Search) command is used to search a literal string in a memory block starting at <address1> through <address2> both inclusive. In BSCH command, if search finds matching data, the data and address(es) must be displayed.

2.2.8.1-) BSCH Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = BSCH
Capture <address1>, <address2> and string
Consider <address1> and compare its contents with the string
For <address1> <= <address2>
Compare the data in memory with given string
If
Match found in the memory
Print success message with the address where the match was found
Else
Continue searching till <address2>
Print failure message if no match was found
Return to main program
Else
Compare input with other commands and proceed



2.2.8.2-) BSCH Assembly Code

```
BSCH:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A4        ;Parse the second address
    CMPA.L A4,A3        ;Check if the first address is smaller
    BGE SYNTAX_ER
    CMPI.B #SPACE,(A1)+
    BNE SYNTAX_ER
    MOVE.L A1,-(A7)
BSCH_LOOP_1:
    MOVE.L (A7),A1
    CMPA.L A3,A4
    BEQ BSCH_FAILD
    CMPM.B (A3)+,(A1)+
    BNE BSCH_LOOP_1
    MOVE.L A3,D0
    SUBQ #1,D0
BSCH_LOOP_2:
    CMPA.L A3,A4
    BLT BSCH_FAILD
    CMPM.B (A3)+,(A1)+
    BEQ BSCH_LOOP_2
    CMPI.B #0,-1(A1)
    BEQ BSCH_SUCCESS
    CMPA.L A3,A4
    BEQ BSCH_FAILD
    MOVE.L (A7),A1
    BRA BSCH_LOOP_1
BSCH_FAILD:
    LEA MSG_BSCH_FAILD,A5
    LEA MSG_BSCH_FAILD_ED,A6
    BSR PRINT
```

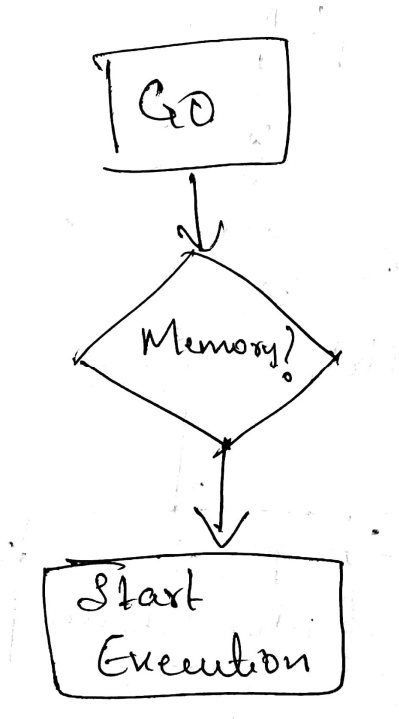
```
    BRA BSCH_END
BSCH_SUCCESS:
    LEA MSG_BSCH_SUCCESS,A5
    LEA MSG_BSCH_SUCCESS_ED,A6
    BSR PRINT_C
    LEA OUTPUT_BUFF,A6
    BSR HEX2ASCII
    LEA OUTPUT_BUFF,A5
    BSR PRINT
    BRA BSCH_END
BSCH_END:
    ADDQ #4,A7
    RTS
```

2.2.9-) GO

The GO command is used to start execution from a given address.

2.2.9.1-) GO Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = GO
Capture <address>
Branch to Subroutine which starts with the given address
After the completion of the Subroutine, Return to main program
Else
Compare input with other commands and proceed



2.2.9.2-) GO Assembly Code

```
GO:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    JSR (A3)
    BRA MAIN
```

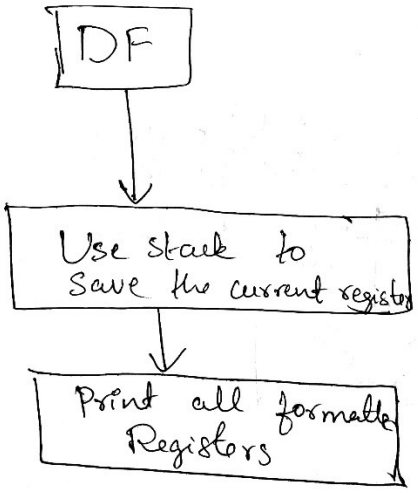
2.2.10-) DF

The Display Formatted Registers (DF) command is used to display the MC68000 processor registers. This command should display current PC, SR, US, SS and D, A registers.

2.2.10.1-) DF Algorithm and Flowchart

Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> *//at this step, the user is prompted to give input*
Accept input from user
If
Input = DF
Store contens on STACK
Print contents of PC, SR, US, SS, D and A registers
Upon completion, return to main program
Else

Compare input with other commands and proceed



2.2.10.2-) DF Assembly Code

```
DF:
    ;PRINT          ;PRINT PC
PC
    MOVEA.L         MOVEA.L #PCDF,A1
#PCDF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
PCHERE          PCHERE
    MOVE            MOVE #PCHERE,D1
#PCHERE,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    ;MOVEA.L        ;MOVEA.L PC,A1
PC,A1
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15

    ;PRINT          ;PRINT SR
SR
```

```
    MOVEA.L         MOVEA.L #SRDF,A1
#SRDF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE SR,D1
SR,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15

    ;PRINT          ;PRINT SSP
SSP
    MOVEA.L         MOVEA.L #SSPDF,A1
#SSPDF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A7,D1
A7,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15

    ;PRINT          ;PRINT USP
USP
    MOVEA.L         MOVEA.L #USPDF,A1
#USPDF,A1
```

```
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE.L          MOVE.L USP,A1
USP,A1
    MOVE.L          MOVE.L A1,D1
A1,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15

    ;load           ;load values from memory
values from
memory
    MOVEM.L         MOVEM.L (SP)+, A0-A6/D0-D7
(SP)+, A0-
A6/D0-D7

    ;save it        ;save it to a mem loc
to a mem loc
    MOVE.L          MOVE.L D0,$8000
D0,$8000
    MOVE.L          MOVE.L D1,$8004
D1,$8004
    MOVE.L          MOVE.L D2,$8008
D2,$8008
    MOVE.L          MOVE.L A1,$8012
A1,$8012
    ;read           ;read memlocs and print a and d regs
memlocs and
print a and
d regs
    ;D0             ;D0
    MOVEA.L         MOVEA.L #D0DF,A1
#D0DF,A1
```

```
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE $8000,D1
$8000,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15

     ;D1             ;D1
    MOVEA.L         MOVEA.L #D1DF,A1
#D1DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE $8004,D1
$8004,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D2             ;D2
    MOVEA.L         MOVEA.L #D2DF,A1
#D2DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE $8008,D1
$8008,D1
```

```
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D3             ;D3
    MOVEA.L         MOVEA.L #D3DF,A1
#D3DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE D3,D1
D3,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D4             ;D4
    MOVEA.L         MOVEA.L #D4DF,A1
#D4DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE D4,D1
D4,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
```

```
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D5             ;D5
    MOVEA.L         MOVEA.L #D5DF,A1
#D5DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE D5,D1
D5,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D6             ;D6
    MOVEA.L         MOVEA.L #D6DF,A1
#D6DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE D6,D1
D6,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;D7             ;D7
    MOVEA.L         MOVEA.L #D7DF,A1
```

```
#D7DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE D7,D1
D7,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15


    ;A0             ;A0
    MOVEA.L         MOVEA.L #A0DF,A1
#A0DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A0,D1
A0,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;A1             ;A1
    MOVEA.L         MOVEA.L #A1DF,A1
#A1DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
```

```
    MOVE            MOVE $8012,D1
$8012,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;A2             ;A2
    MOVEA.L         MOVEA.L #A2DF,A1
#A2DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A2,D1
A2,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;A3             ;A3
    MOVEA.L         MOVEA.L #A4DF,A1
#A4DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A3,D1
A3,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
```

```
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;A4             ;A4
    MOVEA.L         MOVEA.L #A4DF,A1
#A4DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A4,D1
A4,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
    ;A5             ;A5
    MOVEA.L         MOVEA.L #A5DF,A1
#A5DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A5,D1
A5,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15
```

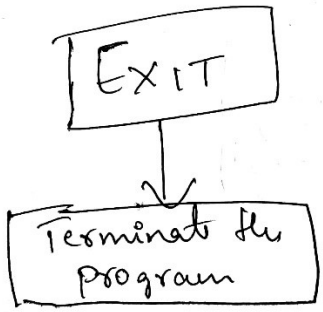
```
    ;A6             ;A6
    MOVEA.L         MOVEA.L #A6DF,A1
#A6DF,A1
    MOVE.B          MOVE.B #14,D0
#14,D0
    TRAP #15        TRAP #15
    MOVE            MOVE A6,D1
A6,D1
    MOVE.B          MOVE.B #16,D2
#16,D2
    MOVE.B          MOVE.B #15,D0
#15,D0
    TRAP #15        TRAP #15
    MOVEA.L         MOVEA.L #SPACE1,A1
#SPACE1,A1
    MOVE.B          MOVE.B #13,D0
#13,D0
    TRAP #15        TRAP #15


    ;MOVING         ;MOVING OLD VALUES BACK TO D0,D1,D2 AND A1
OLD VALUES
BACK TO
D0,D1,D2 AND
A1
    MOVE.L          MOVE.L $8000,D0
$8000,D0
    MOVE.L          MOVE.L $8004,D1
$8004,D1
    MOVE.L          MOVE.L $8008,D2
$8008,D2
    MOVE.L          MOVE.L $8012,A1
$8012,A1


    BRA MAIN        BRA MAIN
```

2.2.11-) EXIT

The EXIT command terminates/exits the Monitor program.

2.2.11.1-) EXIT Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = EXIT
Print termination message
Load 9 to D0 and call TRAP function to terminate
Else
Compare input with other commands and proceed



2.2.11.2-) EXIT Assembly Code

```
EXIT:
     LEA EXIT_MESSAGE,A1
     MOVE #14,D0
     TRAP #15
     MOVE.B #9,D0
     TRAP #15
```

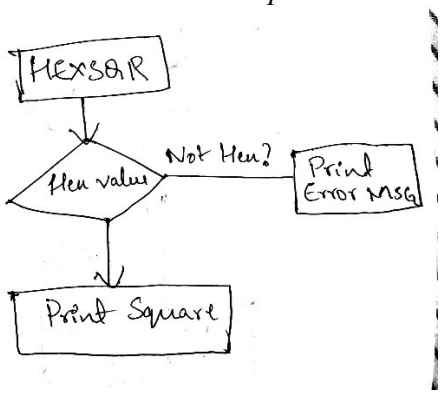
2.2.12-) HEXSQR

The HEXSQR command is used to calculate square of a number given by the user at the input. The command accepts hexadecimal numbers and calculates it's square.

2.2.12.1-) HEXSQR Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system
Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = HEXSQR
Capture the hex-number

Load the captured value in registers D1 and D2
Perform Multiplication operation to obtain square
Print the result and return control
Else
Compare input with other commands and proceed



2.2.12.2-) HEQSQR Assembly Code

```
HEXSQR:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.W D0,D4
    MOVE.W D4,D1        ;Parse the first address
    MOVE D1,D2
    MULU D1,D2
    MOVE.W D2,D0
    MOVE.W #2,D2
    BSR HEX2ASCII
    MOVE.W A6,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    BSR MAIN
```

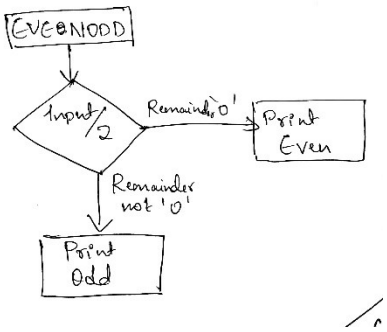
2.2.13-) EVENODD

The command is used to check whether the given number is even or odd. The command accepts only Hexadecimal values.

2.2.12.1-) EVENODD Algorithm and Flowchart

Start
Print welcome message on command prompt
Initialize the system

Print msg prompt Monitor441:> //at this step, the user is prompted to give input
Accept input from user
If
Input = EVENODD
Capture the hex-number
Load the captured value in D0 register and perform division by '2'
The obtained rnesult is swapped to check the remainder
If
Remainder = 0, print that the number is even
Else
Print that the number is odd
Upon completion, return control to main program
Else
Compare input with other commands and proceed



2.2.12.2-) EVENODD Assembly Code

```
EVENODD:
    BSR STORE_HIS
    CMPI.B #SPACE,(A1)+ ;Get rid of the space after command
    BNE SYNTAX_ER
    BSR ASCII2HEX
    MOVE.L D0,A3        ;Parse the first address
    MOVE.L A3,D0
    DIVU #2,D0
    SWAP D0
    CMP.W #0,D0
    BNE ODDN
    LEA EVENNUMBER,A1
    MOVE #14,D0
    TRAP #15
    LEA NEXTLINE,A1
    MOVE #14,D0
    TRAP #15
    BRA MAIN
ODDN    LEA ODDNUMBER,A1
        MOVE #14,D0
        TRAP #15
        LEA NEXTLINE,A1
        MOVE #14,D0
        TRAP #15
        BRA MAIN
```

2.3-) Exception Handlers

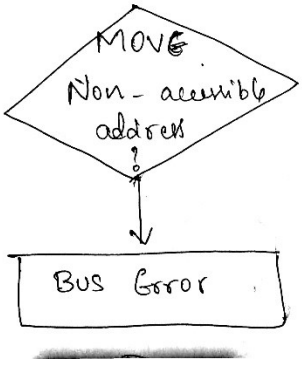
Exception handling is the process of responding to the occurrence, during computation with exceptional conditions requiring special processing , often disrupting the normal flow of program execution. MC68000 uses various exceptions to handle such cases. The exception vector table is modified and 8 exception handlers have been implemented in the program.

2.3.1-) Bus Error Exception

The bus error exception results from anything that causes a failure to complete a bus cycle. It is itself divided in several different groups:

1. Illegal memory access: caused by the processor trying to read memory at a nonexisting address.
2. Faulty memory access: some 68000 systems employ error-detecting memory, which doesn't allow the processor to read memory at an address where an error has occured. If the processor makes an attempt to read memory at such an address, this exception would be generated.
3. Memory privilege violation: A memory privilege violation can occur if an user program attempts to access memory space reserved for another user program, or memory space reserved for the supervisor state.
4. Double bus fault: This is not an exception in itself, but it occurs when two exceptions occur in close proximity and prevent each other from being processed. For example, there can be one bus error generated from reading from a non-existing address, and when the 68000 attempts to process this exception, another bus can occur preventing, say, the SR from being stacked. The processor is blocked, and the only way to recover from a double bus fault is through a reset.

2.3.1.1-) Bus Error Exception Algorithm and Flowchart



2.3.1.2-) Bus Error Exception Assembly Code

```
ORG $7520
    MOVE.L #$FF,D0
    MOVE.B $A00000,D0
    BRA MAIN

    ORG $7000
    MOVEM.L A0-A6/D0-D7,-(SP)
    LEA BUS_ERROR,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15

    ;PRINT BA
    LEA BUSADDRESS,A1
    MOVE.B #14,D0
    TRAP #15

    MOVE.L (18,A7),D1
    MOVE.B #16,D2
    MOVE.B #15,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15

    ;PRINT IR
    LEA IRTEXT,A1
    MOVE.B #14,D0
    TRAP #15

    CLR.L D1
    MOVE.W (22,A7),D1
    MOVE.B #16,D2
    MOVE.B #15,D0
    TRAP #15

    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    ;PRINT SSW
    LEA SSWTEXT,A1
    MOVE.B #14,D0
    TRAP #15

    CLR.L D1
    MOVE.W (16,A7),D1
    MOVE.B #16,D2
    MOVE.B #15,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
```

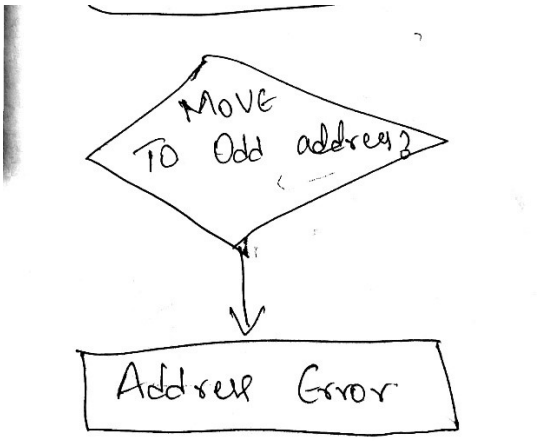
```
    TRAP #15

    ;PRINT EMPTY LINE TO END
    LEA SPACE,A1
    MOVE.B #13,D0
    TRAP #15

    MOVEM.L (SP)+,A0-A6/D0-D7
    BRA  DF
```

Figure 2.10. Debugger Command # 1 Assembly Code

2.3.2-) Address Error Exception

Address error exception occurs when we try to write to odd address location

2.3.1.1-) Address Error Exception Algorithm and Flowchart



2.3.1.2-) Address Error Exception Assembly Code

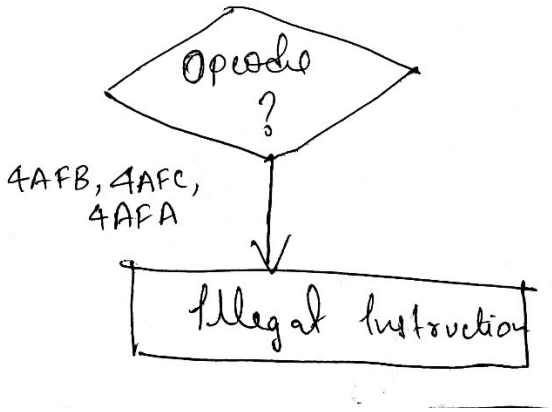
```
* ADDRESS ERROR EXCEPTION

    ORG $9000
    MOVE.W $1234,$5001
    BRA MAIN
    ORG $7100
    MOVEM.L A0-A6/D0-D7,-(SP)
    LEA ADDRESS_ERROR,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
```

```
;PRINT OUT BERR STRING


;PRINT BA
LEA BUSADDRESS,A1
MOVE.B #14,D0
TRAP #15

MOVE.L (18,A7),D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15


;PRINT IR
LEA IRTEXT,A1
MOVE.B #14,D0
TRAP #15

CLR.L D1
MOVE.W (22,A7),D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

;PRINT SSW
LEA SSWTEXT,A1
MOVE.B #14,D0
TRAP #15

CLR.L D1
MOVE.W (16,A7),D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

;PRINT EMPTY LINE TO END
LEA SPACE,A1
MOVE.B #13,D0
TRAP #15

MOVEM.L (SP)+,A0-A6/D0-D7
BRA  DF
```

2.3.3-) Illegal Instruction Exception

Illegal Instruction exception occurs when we attempt to execute an operation code not part of the 68000 instuction set such as $4AFC, $4AFB and $4AFA

2.3.3.1-) Illegal Instruction Exception Algorithm and Flowchart



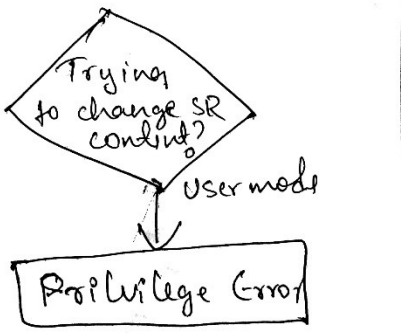
2.3.3.2-) Illegal Instruction Exception Assembly Code

```
ORG $6666
DC.W $4AFC
BRA MAIN
ORG $7200
LEA ILLEGAL_ERROR,A1
MOVE #14,D0

TRAP #15
LEA NEWLINE,A1
MOVE #14,D0
TRAP #15
BSR DF
```

2.3.4-) Privilege Violation Exception

Privilege Violation Exception occurs when a user program attempting to execute a privileged instruction

2.3.4.1-) Privilege Violation Exception Algorithm and Flowchart



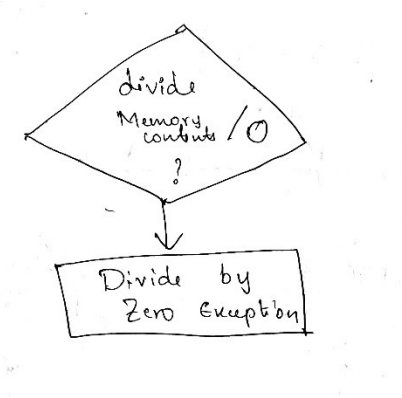
2.3.4.2-) Privilege Violation Exception Assembly Code

```
ORG $5050
Privilege_violation     ANDI.W #$0700,SR
                        BRA Privilege_violation

      ORG $7400
    LEA PRIVILEGE_ERROR,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    BSR DF
```

2.3.5-) Divide by Zero Exception

Divide by zero exception occurs when the user tries to divide a number by 0. This computation is mathematically not feasible.

2.3.5.1-) Divide by Zero Exception Algorithm and Flowchart



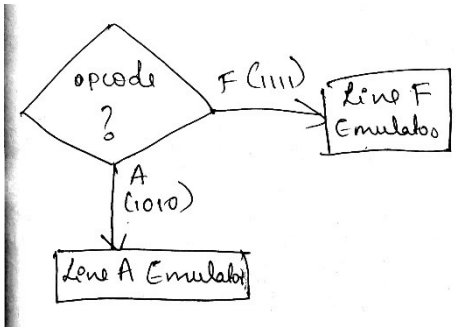
2.3.5.2-) Divide by Zero Exception Assembly Code

```
ORG $5000
    MOVE.B #0,D1        ;DIVIDE BY ZERO ERROR
    MOVE.B #5,D2
    DIVU D1,D2
    BRA MAIN

    ORG $7300
    LEA DIVIDE_ERROR,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    BSR DF
```

2.3.6-) Line A and Line F Emulators

Line A and Line F Emulator exceptions occur if the instruction opcode starts with A(1010) or F(1111) respectively as these opcodes are not defined under the instruction set of MC68000.

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



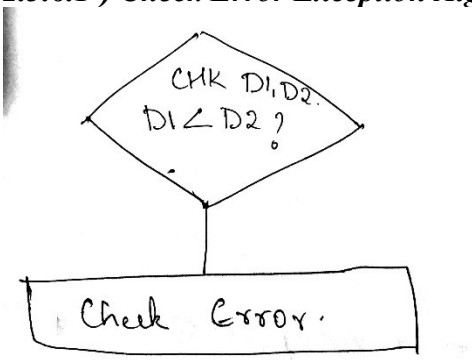
2.3.6.2-) Line A and Line F Emulators Assembly Code

```
** LINE 1010 ERROR EXCEPTION

    ORG $7666
    DC.W $AAAA
    BRA MAIN
    ORG $7500
    LEA LINEA_EXCEPTION,A1
    MOVE #14,D0
    TRAP #15
    LEA NEWLINE,A1
    MOVE #14,D0
    TRAP #15
    BSR DF

* LINE 1111 ERROR EXCEPTION

    ORG $8666
    DC.W $FFFF
    BRA MAIN
     ORG $7600
     MOVE $F000,A0
     LEA LINEF_EXCEPTION,A1
     MOVE #14,D0
     TRAP #15
     LEA NEWLINE,A1
     MOVE #14,D0
     TRAP #15
     BSR DF
```

.3.6-) Check Error Exception

Check Error Exception is mainly used to check registers against boundaries. It compares the values stored in the registers and generates error accordingly.

2.3.6.1-) Check Error Exception Algorithm and Flowchart



2.3.6.2-) Check Error Exception Assembly code

```
* CHECK ERROR EXCEPTION

     ORG $4260

     MOVE.L #$3000,D6
     MOVE.L #$3010,D7
     CHK.W D6,D7
     BRA MAIN

     ORG $7700
     LEA CHECK_ERROR,A1
     MOVE #14,D0
     TRAP #15
     LEA NEWLINE,A1
     MOVE #14,D0
     TRAP #15
     BSR DF
```

2.4-) User Instructional Manual

The following section contains quick user manual with usage descriptions for all commands. All the input values must be given in hex and all address locations should start with a '$' symbol.

```
;BSCH
MSG_HELP_BSCH:
    DC.B    'Search for specific pattern (input as string) within',CR,LF
    DC.B    'a memory range. If found, print the location of such',CR,LF
    DC.B    'string, if not found print failed promotion.',CR,LF
    DC.B    'Syntax - BSCH <addr1> <addr2> string'
MSG_HELP_BSCH_ED
```

```
;MDSP
MSG_HELP_MDSP:
    DC.B    'The command outputs the address and memory contents from.',CR,LF
    DC.B    '<address1> to <address2>',CR,LF
    DC.B    'The command also outputs the address and memory contents
from',CR,LF
    DC.B    '<address1> to <address1 + 16bytes>',CR,LF
    DC.B    'Syntax - MDSP <addr1> <addr2> or MDSP <addr1> '
MSG_HELP_MDSP_ED

;MM
MSG_HELP_MM:
    DC.B    'The memory modify command is used to display and modify
data',CR,LF
    DC.B    'The size (byte, word, long word) controls the number of
bytes',CR,LF
    DC.B    'displayed for each address.',CR,LF
    DC.B    'Syntax - MM <addr1> <addr2> <B/W/L>'
MSG_HELP_MM_ED

;MS
MSG_HELP_MS:
    DC.B    'The command alters memory by setting data into the address
specified.',CR,LF
    DC.B    'The data can take the form of ASCII string or hexadecimal
data.',CR,LF
    DC.B    'Syntax - MS <addr> <data> <A/H>'
MSG_HELP_MS_ED

;BF
MSG_HELP_BF:
    DC.B    'The command fills memory starting with the word boundary
from',CR,LF
    DC.B    '<addr1> to <addr2>. The command fills with only word-size
pattern',CR,LF
    DC.B    'Syntax - BF <addr1> <addr2> Hex-value'
MSG_HELP_BF_ED

;BMOV
MSG_HELP_BMOV:
    DC.B    'The command is used to move blocks of memory from',CR,LF
    DC.B    'one area to another',CR,LF
    DC.B    'Syntax - BMOV <addr1> <addr2> <addr3>'
MSG_HELP_BMOV_ED

;BTST
MSG_HELP_BTST:
    DC.B    'The command performs destructive test for a block of
memory',CR,LF
    DC.B    'If the test runs to completion, success message is
displayed.',CR,LF
    DC.B    'Else,address of the memory, data stored and data read are
displayed ',CR,LF
    DC.B    'Syntax - BTST <addr1> <addr2>',CR,LF
MSG_HELP_BTST_ED
```

```
;DF
MSG_HELP_DF:
    DC.B    'The command displays the MC68K processor registers.',CR,LF
    DC.B    'The command displays current PC, SR, US, SS and D, A
registers.',CR,LF
    DC.B    'Syntax - DF'
MSG_HELP_DF_ED

;GO
MSG_HELP_GO:
    DC.B    'The command is used to start the execution of program',CR,LF
    DC.B    'from given address. Syntax - GO <addr>'
MSG_HELP_GO_ED

;EXIT
MSG_HELP_EXIT:
    DC.B    'The command terminates the monitor program. '
MSG_HELP_EXIT_ED

;HEZSQR
MSG_HELP_HEXSQR:
    DC.B    'The command calculates the square of a hex number.',CR,LF
    DC.B    'Syntax - HEXSQR <hexvalue>'
MSG_HELP_HEXSQR_ED

;EVENODD
MSG_HELP_EVENODD:
    DC.B    'The command checks whether the given number is even or
odd.',CR,LF
    DC.B    'Syntax - EVENODD <hexvalue>'
MSG_HELP_EVENODD_ED
```

3-) Discussion

The design of this monitor program was very challenging. It encouraged students in decision making by providing good number of commands to experiment with. Although it was difficult to implement few commands initially, by the end of the project, thorough knowledge was gained in terms of their usage.

4-) Feature Suggestions

The monitor project could be implemented with many more debugger commands. As this project was completely based on software, methods can be developed to incorporate functionality of peripheral devices.

5-) Conclusion

All the debugger commands and exception handlers were implemented successfully. With the help of instruction manual, any user will be able to use the monitor program. Error

checks and messages have been implemented to help the user to interact with the monitor program comfortably.

6-) References

Supply all references here (books, internet resources, papers, manuals, etc). You need to use square parentheses.

[1] ECE441 lab manual, Illinois Institute of Technology
[2] Educational Computer Board Manual

[3] https://www.cs.mcgill.ca/~cs573/fall2002/notes/lec273/lecture21/21_3.htm
[4]