ECE 441
Microprocessors

Instructor: Dr. Jafar Saniie
Teaching Assistant: Guojun Yang

Final Project Report:
MONITOR PROJECT
04/27/2018

By: Rukang Zhao

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

The project is a monitor program that able to perform basic debugger functions such as memory display, memory set memory modifies, and it can also deal with exceptions. The program is similar version of TUTOR that we used in ECE441 lab. In this report, I'll present a basic idea about how I implement the debugger command and Exception handler. I'll also mention the difficult I met and my future suggestions about the project.

1-) Introduction

The program is a similar program as TUTOR program we used in ECE441 lab. Generally, the program first read a command from user and search the command through the command table, one the program pair the command, it will go to the relevant subroutine and execute the command. The program will report error if the command is not valid. Eight exceptions are also supported in the program, so, the program will display formatted registers when there is exception occurred. For bus error and address error, the program will also print out BA, IR, SSW.

2-) Monitor Program

There are three parts in the program. The first part is command interpreter, which let user input a command and search the command to go to the right sub routine. The second part is debugger commands, which included 12 commands and two user identified command. The third part is exception handler, that could print exception information when it occurred.

2.1-) Command Interpreter

Command Interpreter first show the prompt "MONITOR441>", then it let user input a command and store it in stack. After that, the program will compare each character in the command to the input. If the first part of the command matches with the compare table, the program will go to the command address to call the command subroutine.

2.1.1-) Algorithm and Flowchart

The algorithm already explained before and below is the flow chart of the command interpreter.

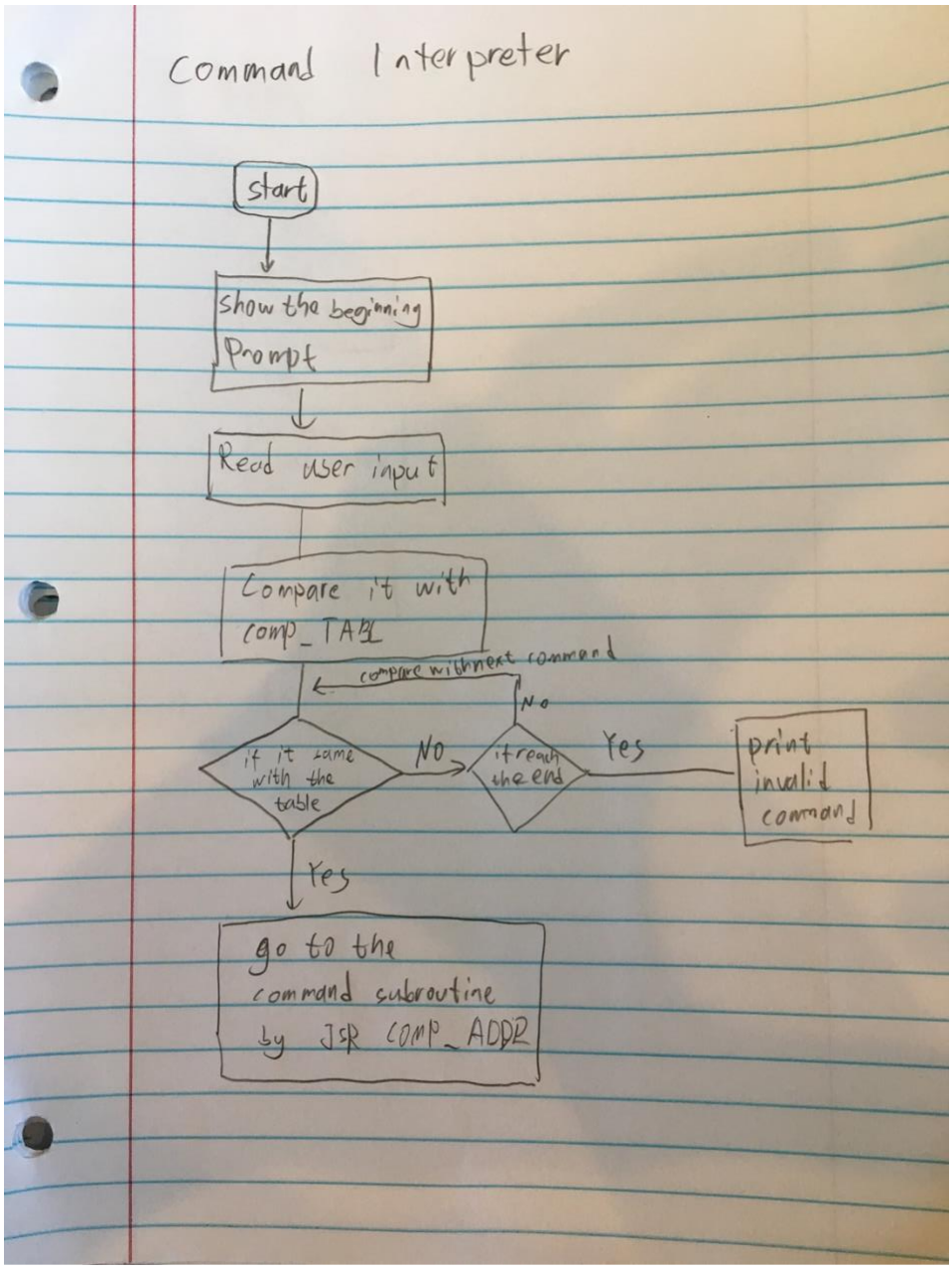


Figure 2.1.1. command interpret flowchart

2.1.2-) Command Interpreter Assembly Code

The assembly code should be written using the algorithm above.

```
*-----------------------------------------------------------
*PROGRAM
*-----------------------------------------------------------
*INITIALIZATION
START:
STACK       EQU $2FFC

*INITIALIZE THE STACK

        LEA     $2FFC,A7    ;INITIALIZE THE STACK
        MOVEM.L D0-D7/A0-A6,-(A7) ;STORE REGISTERS
*INITIALIZE EXCEPTION HANDLER
        MOVE.L #BERR,$8
        MOVE.L #AERR,$C
        MOVE.L #ILL_INSTR,$10
        MOVE.L #ZERO_DEV,$14
        MOVE.L #CHK_INSTR,$18
        MOVE.L #PRIV_VIOL,$20
        MOVE.L #LINEA_EMU,$28
        MOVE.L #LINEF_EMU,$2C
*SET UP INPUT BUFFER
        SUBA.L #80,A7       ;SET INPUT BUFFER FOR A7
MAIN_INTERFACE:
        LEA     MONITOR_PROMPT,A1  ;DISPLAY PROMPT
        MOVE.W  #11,D1
        MOVE.L  #1,D0
        TRAP    #15
        MOVEA.L A7,A1   ;INPUT GOES TO STACK
        MOVE.B  #2,D0
        TRAP #15

        LEA     COMP_TABL,A2 ;A2-LOAD COMMAND TABLE
        LEA     COMP_ADDR,A3 ;A3-COMMAND ADDRESS
        CLR.L   D3      ;D3 RECORD DISPLACEMENT TO VECTOR TABLE
SEARCH_TABLE
        MOVEA.L A1,A4   ;A4 STORE THE COPY OF START INPUT ADDR
        CLR.L   D2      ;SETUP D2
        MOVE.B  (A2)+,D2    ;D2 STORE THE LENGTH OF ONE COMMAND
        SUBI.B  #$30,D2  ;CONNVERT THE LENGTH TO HEX
COMP
        CMPM.B  (A4)+,(A2)+ ;compare each character
        DBNE    D2,COMP
        TST.W   D2 ; if match the command, go to  ADDR_SEARCH
        BLT     ADDR_SEARCH
        ADDA.L  D2,A2   ;ADD THE REST OF D2 TO THE END OF COMMAND
        CMPA.L  A2,A3   ;COMPARE IF THE COMMAND TABLE REACH THE END
        BLE ERROR_MESG  ;IF REACH THE END
        ADD.L   #2,D3   ;IF NOT, ADD DISPLACEMENT TO D3
        ;BRA  TEST_SUCCESS
        BRA     SEARCH_TABLE


ERROR_MESG
        LEA     INVALID_PROMPT,A1   ;DISPLAY INVALID COMMAND
        MOVE.L  #17,D1
        MOVE.L  #1,D0
        TRAP    #15
        BRA     MAIN_INTERFACE
ADDR_SEARCH
        ADDA.L  D3,A3   ;ADD DISPLACEMENT TO A4 TO GET CMD ADDR
        MOVEA.W (A3),A5   ;MOVE THE ADDRESS IN A5
        JSR     (A5)
        BRA     MAIN_INTERFACE
```

Figure 2.4. 68000 Assembly Code

2.2-) Debugger Commands

2.2.1-) HELP (HELP)

The Help command shows all the available prompt.

2.2.1.1-) HELP Algorithm and Flowchart

The command loads the address of the prompt and output. Since the terminal of the MC68K is too small, the command will output the prompt one part first and let user to input enter to output another part.

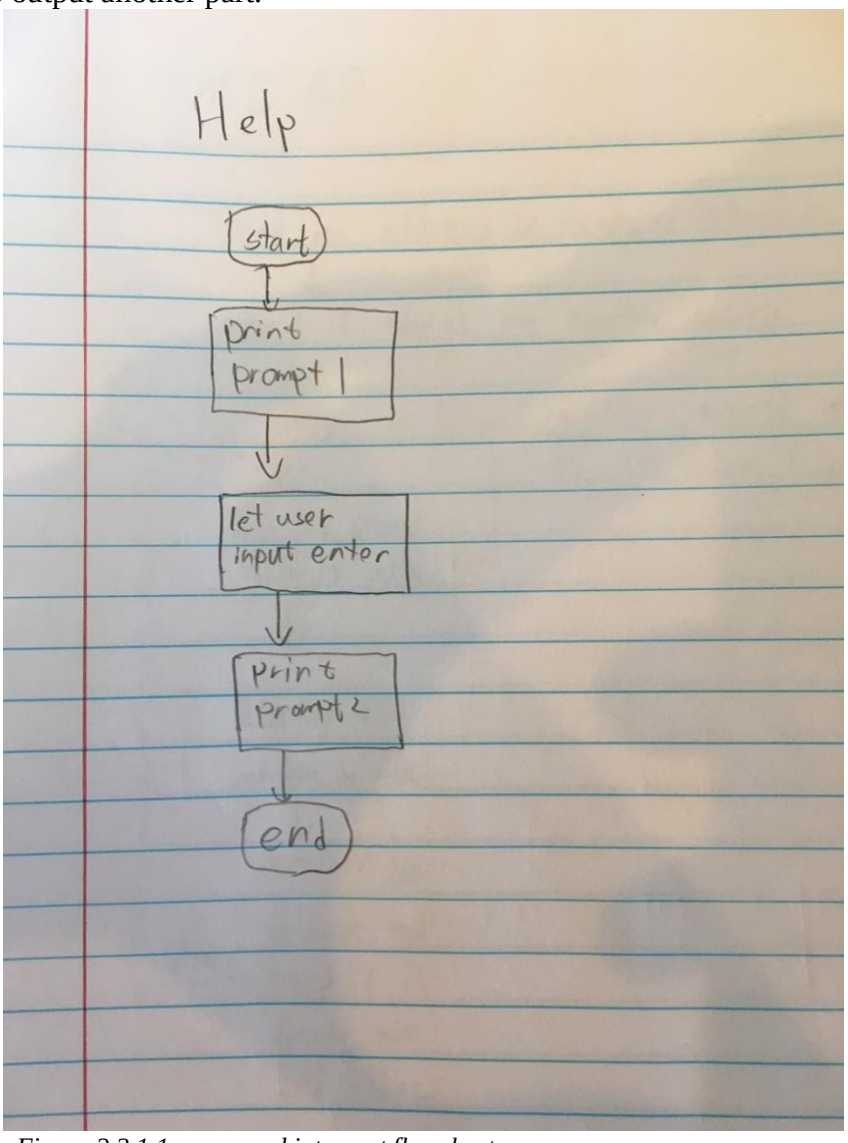


Figure 2.2.1.1. command interpret flowchart

2.2.1.2-) HELP Assembly Code

The assembly code should be written using the algorithm above.

```
HELP_PROMPT1
            DC.B    'Help: HELP',$A,$D
            DC.B    'Display all available commands & description',$A,$A,$D

            DC.B    'Memory Display: MDSP <addr1> <addr2>',$A,$D
            DC.B    'Outputs the address and memory contents betwee two address',$A,$D
            DC.B    'If no address 2 specified, display up to addres1 + 16bytes',$A,$A,$D

            DC.B    'Sort: SORTW <A or D> <addr1> <addr2>',$A,$D
            DC.B    'Sorts a block of memory word specified between two address',$A,$D
            DC.B    'A or D specifies whether is Ascending or Descending order',$A,$A,$D

            DC.B    'Memory Modify: MM <size> <addr>',$A,$D
            DC.B    'Used to display memory and modify data or enter new data',$A,$D
            DC.B    'B for byte, W for world, L for long word' ,$A,$A,$D

            DC.B    'Memory Set: MS <type> <addr> <data>',$A,$D
            DC.B    'Alters memory by setting data address into address specified',$A,$D
            DC.B    'S for ASCII string and H for hexadecimal data',$A,$D
            DC.B    'Hex must be even number to fit the memory in address',$A,$A,$D


            DC.B    'Block Fill: <addr1> <addr2> <WORD PATTERN>',$A,$D
            DC.B    'Fill the memory between two addresses',$A,$D
            DC.B    'Address should be even and be filled with word size parrern',$A,$D
            DC.B    '0 will be filled if the pattern is less than word size',$A,$A,$D,0
HELP_PROMPT2
            DC.B    'Block Move: BMOV <addr1_A> <addr2_A> <addr_B>',$A,$D
            DC.B    'Move (duplicate)blocks of memory from one area(A) to another(B)',$A,
$A,$D

            DC.B    'Block Test: BTST <addr1> <addr2>',$A,$D
            DC.B    'Detect memory block between two address and report error',$A,$A,$D

            DC.B    'Block Search: BSCH <addr1> <addr2> <string>',$A,$D
            DC.B    'Search a literal string between two address(inclusive)',$A,$D
            DC.B    'If match the content the data and address will be displayed',$A,$A,$D

            DC.B    'Execute Program: Go <Addr>',$A,$D
            DC.B    'Start execution from a given address.',$A,$A,$D

            DC.B    'Display Formatted Registers: DF',$A,$D
            DC.B    'Display the MC68000 formated registers',$A,$A,$D

            DC.B    'COWASY: COWSAY <MESSAGE>',$A,$D
            DC.B    'Generate ASCII pictures of a cow with given messages',$A,$A,$D

            DC.B    'Generate Random Number: ROLL',$A,$D
            DC.B    'Generate a random number between 0 to 9',$A,$A,$D

            DC.B    'Exit Monitor Program: EXIT',$A,$A,$D

            DC.B    'The program will only read front valid part of command',$A,$D,0

HELP_PROMPT3
            DC.B    'Press Enter To Continue',$A,$D,0


.
```

```
*-----------------------------------------------------------
HELP
        MOVEM.L D0-D7/A0-A6,-(SP)
        LEA     HELP_PROMPT1,A1  ;LOAD PROMPT1 ADDRESS
        MOVE.L  #13,D0
        TRAP    #15

        LEA     HELP_PROMPT3,A1  ;LET USER PRESS ENTER TO CONTINUE
        MOVE.L  #13,D0
        TRAP    #15

        MOVE.B  #2,D0           ;READ A CHARACTER
        TRAP    #15

        LEA     HELP_PROMPT2,A1  ;LOAD PROMPT2 ADDRESS
        MOVE.L  #13,D0
        TRAP    #15

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

Figure 2.2.1.2 HELP Assembly Code

2.2.2-) Memory Display(MDSP)

The MDSP (Memory Display) command outputs the address and memory contents from <address1> to <address2>.

The MDSP (Memory Display) command ALSO outputs the address and memory contents from <address1> to <address1 + 16bytes>.

2.2.2.1-) Debugger Command #2 Algorithm and Flowchart

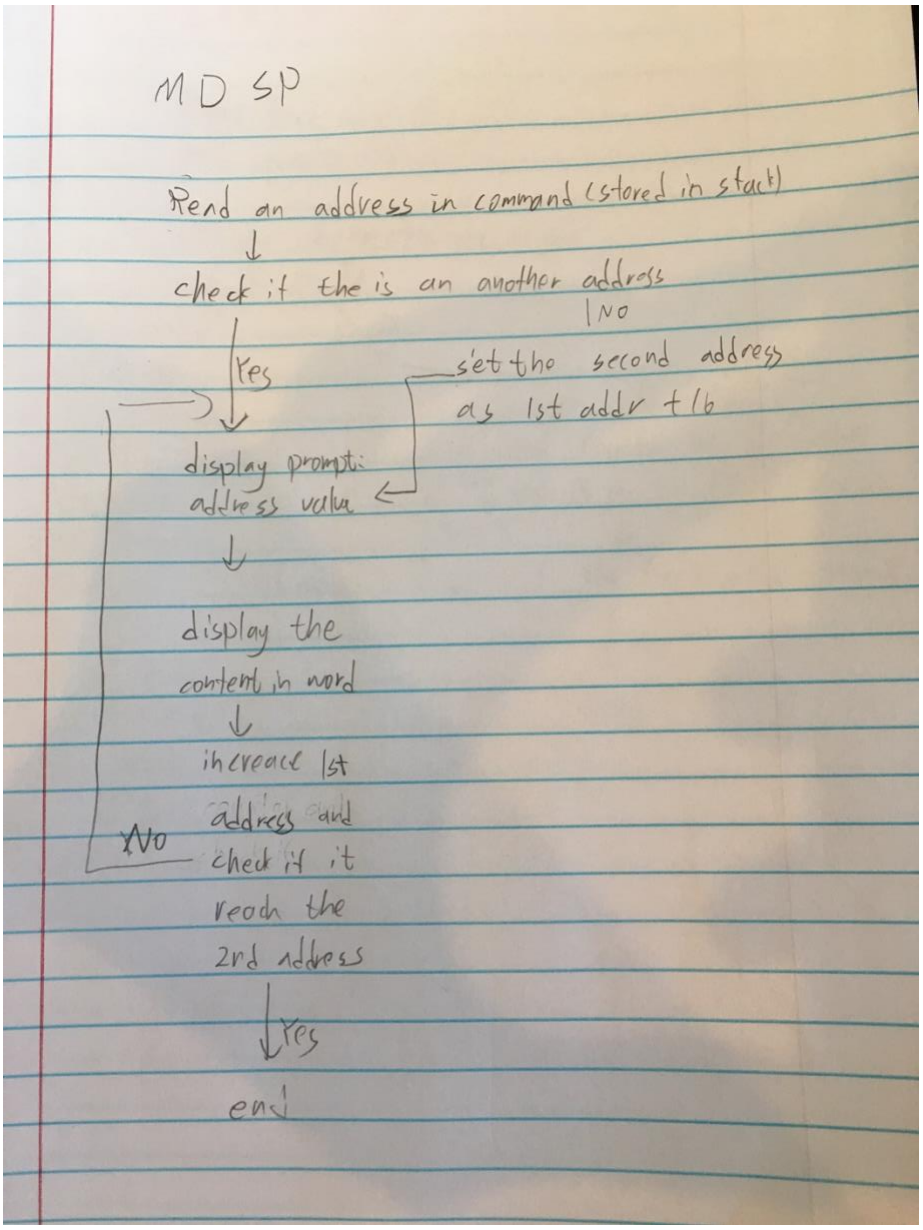


Figure 2.2.2.1. command interpret flowchart

2.2.2.2-) Debugger Command #2 Assembly Code

```
        BSR
MDSP
*DISPLAY MEMORY BETWEEN ADDRESS A2 AND A3
        MOVEM.L D0-D6/A0-A3,-(SP)
        BSR     LOAD_ONE_ADDR
        MOVEA.L  D3, A2
        BSR     IS_NEXT         ;CHECK IF IT'S SINGLE ADDR OR DOUBLE
        CMP.B    #0,D6
        BNE     MDSP_DOUBLE_ADDR    ; IF D6 NOT 0, GO TO DOUBLE ADDRESS

MDSP_SINGLE_ADDR            ;A3 = A2+16
        MOVE.L  A2,A3
        ADDA.L  #16,A3
        BRA     MDSP_OUTPUT
MDSP_DOUBLE_ADDR            ;A3 = READ NEXT MEMORY
        BSR     LOAD_ONE_ADDR
        MOVEA.L D3,A3
MDSP_OUTPUT
*CHECKING ERROR FIRST A3 SHOULD GREATER THAN A2
        CMPA.L  A2,A3   ;CHECK WHICH ON IS BIGGER
        BGE     MDSP_RIGHT_ADDRESS  ;IF A3 GREATER OR EQUAL THAN A2
        BRA     ERROR_MESG
MDSP_RIGHT_ADDRESS
        MOVE.L  #$24,D1 ;DISPLAY '$' FIRRST
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #16,D2  ;D2 AS OUTPUT BASE
        MOVE.L  A2,D1   ;PUT A2 IN D1 TO DISPLAY
        MOVE.L  #15,D0  ;OUTPUT ADDRESS
        TRAP    #15

        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$3A,D1 ;DISPLAY ':'
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #16,D2  ;D2 AS ADDRESS CONTENT BASE
        MOVE.L  (A2)+,D1    ;OUTPUT THE CONTENT IN A2
        MOVE.L  #15,D0
        TRAP #15

        MOVE.L  #$0A,D1 ;DISPLAY 'LF'
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$0D,D1 ;DISPLAY 'CR'
        MOVE.L  #6,D0
        TRAP    #15

        CMPA.L  A2,A3
        BGE     MDSP_RIGHT_ADDRESS


        MOVEM.L (SP)+, D0-D6/A0-A3
        RTS
```

Figure 2.2.2.2 MDSP Assembly Code

It is similar to 2.2.1.2

2.2.3-) Sort(SORTW)

The command sort a block of memory with given key word (ether ascending order or descending order)

2.2.3.1-) Debugger Command #3 Algorithm and Flowchart

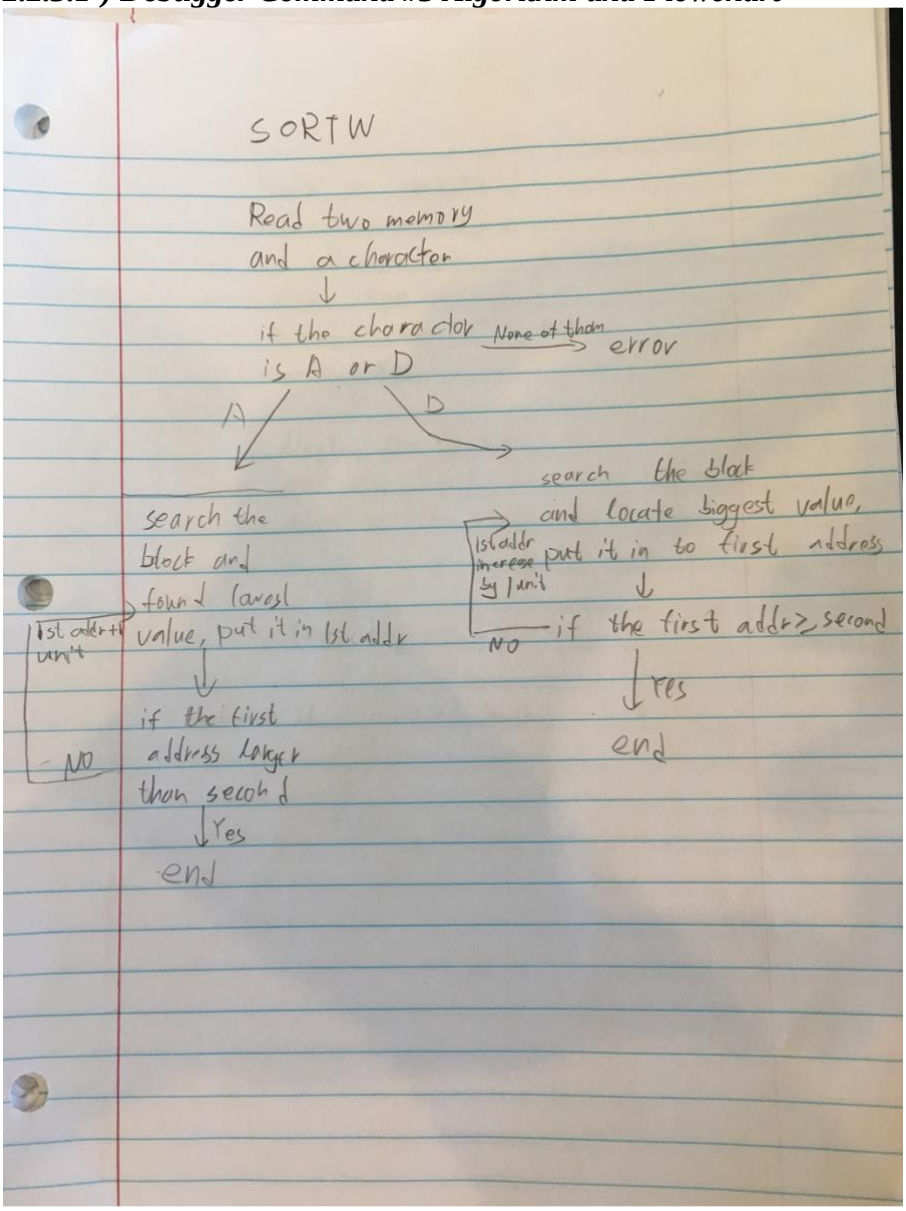


Figure 2.2.3.1. command interpret flowchart

2.2.3.2-) Debugger Command #3 Assembly Code

```
SORTW
        MOVEM.L     D0-D6/A0-A7,-(SP)
        BSR         CLEAR_SPACE       ;READ A CHARACTER IN D2 FIRST
        MOVE.B      (A4)+,D2
        BSR         LOAD_TWO_ADDR       ;READ AND LOAD TWO ADDRESS TO A2 AND A3
        MOVEA.L     D4,A2
        MOVEA.L     D5,A3
        CMP.B       #$44,D2     ;CHECK IF IT'S D
        BEQ         SORTW_D
        CMP.B       #$41,D2     ;CHECK IF IT'S A
        BEQ         SORTW_A
        BRA         ERROR_MESG
SORTW_D_SWAP  ;SWAP THE VALUE BETWEEN TWO ADDRESS A2,A5
        MOVE.W      (A2),D6
        MOVE.W      (A5),(A2)
        MOVE.W       D6,(A5)
        BRA         SORTW_D_DONE_SWEAP

SORTW_D
        MOVEA.L     A2,A5      ;A5 AS THE BEGINNGING ADDRESS
        MOVE.W      (A2),D3    ;SET THE BEGINNING VALUE IN D3

SORTW_D_LOOP
        CMP.W       (A5),D3    ;COMPARE D3 AND NEXT VALUE
        BLT         SORTW_D_SWAP ;IF (A5) LARGER
SORTW_D_DONE_SWEAP
        ADDA.L      #2,A5
        CMPA.L      A5,A3   ;IF A5 REACH THE END OF THE ADDRESS
        BGE         SORTW_D_LOOP


        ADDA.L      #2,A2
        CMPA.L      A2,A3   ;CHECK IF A2 REACH THE END
        BGT         SORTW_D
        ;BRA         TEST_SUCCESS
        BRA         SORTW_DONE



SORTW_A_SWAP  ;SWAP THE VALUE BETWEEN TWO ADDRESS A2,A5
        MOVE.W      (A2),D6
        MOVE.W      (A5),(A2)
        MOVE.W       D6,(A5)
        BRA         SORTW_A_DONE_SWEAP
```

```
SORTW_A_SWAP  ;SWAP THE VALUE BETWEEN TWO ADDRESS A2,A5
        MOVE.W      (A2),D6
        MOVE.W      (A5),(A2)
        MOVE.W       D6,(A5)
        BRA         SORTW_A_DONE_SWEAP


SORTW_A
        MOVEA.L     A2,A5       ;A5 AS THE BEGINNGING ADDRESS
        MOVE.W      (A2),D3     ;SET THE BEGINNING VALUE IN D3

SORTW_A_LOOP
        CMP.W       (A5),D3     ;COMPARE D3 AND NEXT VALUE
        BGT         SORTW_A_SWAP
SORTW_A_DONE_SWEAP
        ADDA.L      #2,A5
        CMPA.L      A5,A3   ;IF A5 REACH THE END OF THE ADDRESS
        BGE         SORTW_A_LOOP


        ADDA.L      #2,A2
        CMPA.L      A2,A3   ;CHECK IF A2 REACH THE END
        BGT         SORTW_A

        BRA         SORTW_DONE

SORTW_DONE
        MOVEM.L     (SP)+,D0-D6/A0-A7
```

Figure 2.2.3.2 SORT Assembly Code

2.2.4-) Memory Modify(MM)

The command modify a block of memory with the size specified by user

2.2.4.1-) Debugger Command #4 Algorithm and Flowchart

The algorithm shows in the flowchart

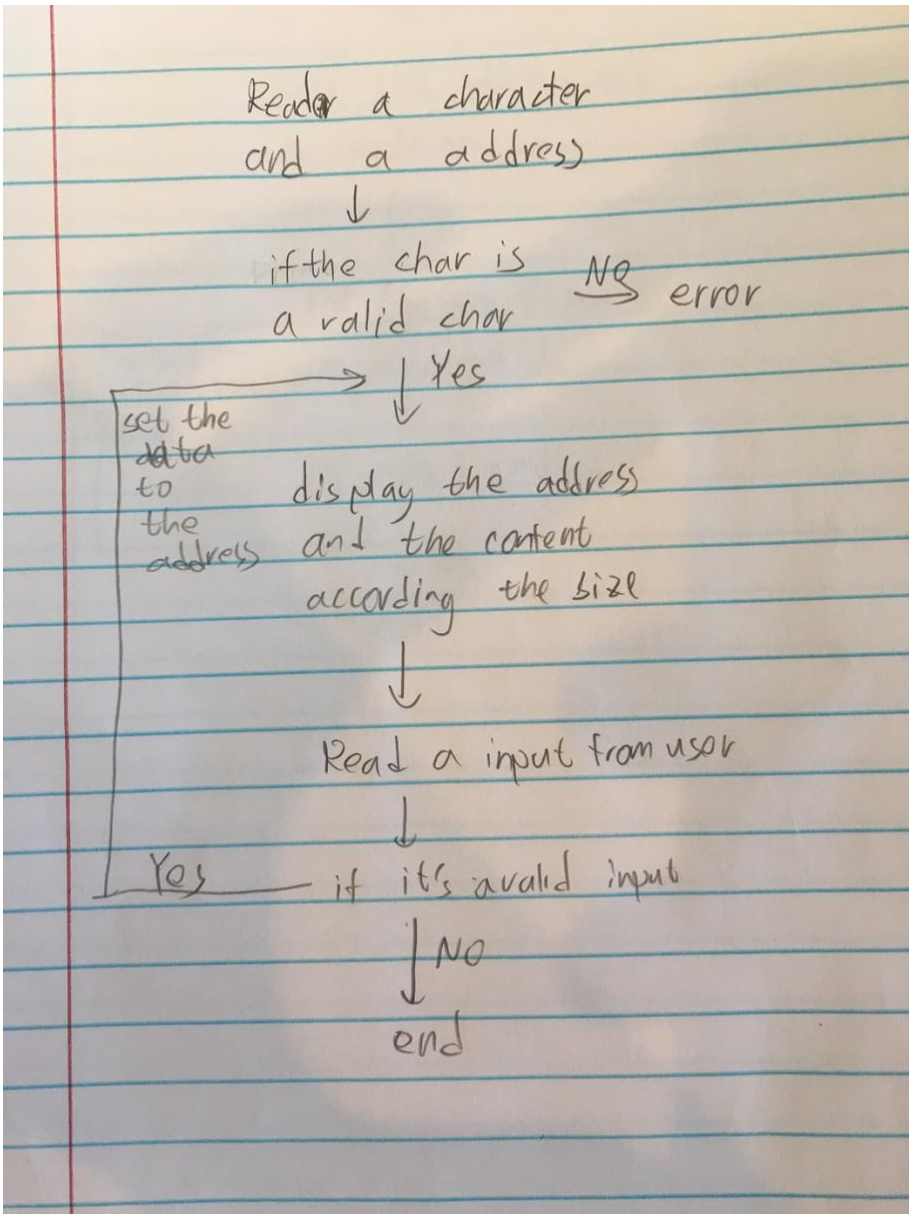


Figure 2.2.4.2. command interpret flowchart

2.2.4.2-) Debugger Command #4 Assembly Code

```
        RTS
*MEMORY MODIFY, IF THE PATTERN LARGER THAN A WORD, IT WILL
*COUNT AS LAST FOUR DIGIT
MM
        MOVEM.L     D0-D7/A0-A6,-(SP)


        BSR         ASCII_READ       ;READ A CHARACTER IN D2
        CLR.L       D6
        MOVE.L      D2,D6            ;TRANSFER IT TO D6

        BSR         LOAD_ONE_ADDR   *LOAD ONE ADDRESS IN D3
        MOVEA.L     D3,A6           ;PUT THE ADDRESS IN A6

        CMP.B       #$42,D6     ;CHECK B
        BEQ         MM_BEFORE_OUTPUT
        CMP.B       #$57,D6     ;CHECK W
        BEQ         MM_BEFORE_OUTPUT
        CMP.B       #$4C,D6     ;CHECK L
        BEQ         MM_BEFORE_OUTPUT
        BRA         ERROR_MESG
MM_BEFORE_OUTPUT
        LEA         MM_PROMPT_1,A1    ;PRINT MM PROMPT FIRST
        MOVE.B      #1,D0
        MOVE.L      #200,D1
        TRAP        #15

MM_OUTPUT
        MOVE.L  #$24,D1 ;DISPLAY '$' FIRRST
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #16,D2  ;D2 AS OUTPUT BASE
        MOVE.L  A6,D1   ;PUT A2 IN D1 TO DISPLAY
        MOVE.L  #15,D0  ;OUTPUT ADDRESS
        TRAP    #15

        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15
        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15
        MOVE.L  #$3F,D1 ;DISPLAY '?'
        MOVE.L  #6,D0
        TRAP    #15
        MOVE.L  #$20,D1 ;DISPLAY SPACE
        MOVE.L  #6,D0
        TRAP    #15
MM_OUTPUT_READ
        MOVEA.L A7,A1
        SUBA.L  #80,A1  ;LEAVE SOME SPACE FOR THE INPUT BUFFER
        MOVE.L  #2,D0
        TRAP    #15
        BSR     MM_READ_ADDR
        BRA     MM_OUTPUT
MM_OUTPUT_DONE
        MOVEM.L     (SP)+,D0-D7/A0-A6
        RTS
*-------------------MM SUBROUTINE-----------------------------------
MM_OUTPUT_ADDR
*OUTPUT THE ADDRESS CONTENT DEPENDSON A6
        CMP.B       #$42,D6     ;CHECK B
        BEQ         MM_OUTPUT_ADDR_B
        CMP.B       #$57,D6     ;CHECK W
        BEQ         MM_OUTPUT_ADDR_W
        CMP.B       #$4C,D6     ;CHECK L
        BEQ         MM_OUTPUT_ADDR_L
MM_OUTPUT_ADDR_B
        MOVE.L      #2,D3   ;D2 STORE THE LENGTH TO OUTPUT
        MOVE.L  #16,D2  ;D2 AS OUTPUT BASE
        CLR.L   D1
        MOVE.B  (A6),D1   ;PUT A2 IN D1 TO DISPLAY
        MOVE.L  #15,D0  ;OUTPUT ADDRESS
        TRAP    #15
        RTS
```

```
MM_OUTPUT_ADDR_W
        MOVE.L      #4,D3
        MOVE.L  #16,D2  ;D2 AS OUTPUT BASE
        CLR.L   D1
        MOVE.W  (A6),D1   ;PUT A2 IN D1 TO DISPLAY
        MOVE.L  #15,D0  ;OUTPUT ADDRESS
        TRAP    #15
        RTS
MM_OUTPUT_ADDR_L
        MOVE.L      #8,D3
        MOVE.L  #16,D2  ;D2 AS OUTPUT BASE
        CLR.L   D1
        MOVE.L  (A6),D1   ;PUT A2 IN D1 TO DISPLAY
        MOVE.L  #15,D0  ;OUTPUT ADDRESS
        TRAP    #15
        RTS

MM_READ_ADDR
        MOVEM.L     D0-D6/A0-A5,-(SP)

*READ STRING IN A1 AND STORE IT IN D5 DEPENDS ON THE SIZE

        MOVEA.L     A4,A3   ;STORE A4 IN A3
        MOVEA.L     A1,A4   ;MOVE A1 TO A4 TO TRIGGER FUNCTION
MM_IS_HEX
        MOVE.B      (A4),D0     ;CHECK IF IT SHOULD END
        CMP.B       #$30,D0     ;END IF IT HAVE OTHER CHARACTERS
        BLT         MM_READ_ADDR_EXIT

        CMP.B       #$46,D0
        BGT         MM_READ_ADDR_EXIT

        CMP.B       #$41,D0
        BGT         MM_READ_ADDR_NEXT   ;CHECK IF IT'S MIDDLE CHARACTERS

MM_IS_MIDDLE
        CMP.B       #$39,D0
        BGT         MM_READ_ADDR_EXIT

        BRA         MM_READ_ADDR_NEXT
```

```
MM_READ_ADDR_NEXT
        CMP.B       #$42,D6     ;CHECK B
        BEQ         MM_READ_ADDR_B
        CMP.B       #$57,D6     ;CHECK W
        BEQ         MM_READ_ADDR_W
        CMP.B       #$4C,D6     ;CHECK L
        BEQ         MM_READ_ADDR_L
        BRA         ERROR_MESG
MM_READ_ADDR_B

        BSR         LOAD_ONE_ADDR
        MOVE.L      D3,D5       ;MODIFY THE ADDRESS
        MOVE.B      D5,(A6)+
        BRA         MM_READ_ADDR_DONE
MM_READ_ADDR_W

        BSR         LOAD_ONE_ADDR
        MOVE.L      D3,D5       ;MODIFY THE ADDRESS
        MOVE.W      D5,(A6)+
        BRA         MM_READ_ADDR_DONE
MM_READ_ADDR_L
        BSR         LOAD_ONE_ADDR
        MOVE.L      D3,D5       ;MODIFY THE ADDRESS
        MOVE.L      D5,(A6)+
        BRA         MM_READ_ADDR_DONE
MM_READ_ADDR_DONE
        MOVEM.L     (SP)+,D0-D6/A0-A5
        RTS
MM_READ_ADDR_EXIT

        LEA         MM_PROMPT_2,A1    ;PRINT MM PROMPT FIRST
        MOVE.B      #1,D0
        MOVE.L      #200,D1
        TRAP        #15
        MOVEM.L    (SP)+, D0-D6/A0-A5
        ;BRA TEST_SUCCESS
        ADDA.L      #4,SP
        BRA         MM_OUTPUT_DONE
```

Figure 2.2.4.1MM Assembly Code

2.2.5-) Block Fill (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-) Block Fill Algorithm and Flowchart

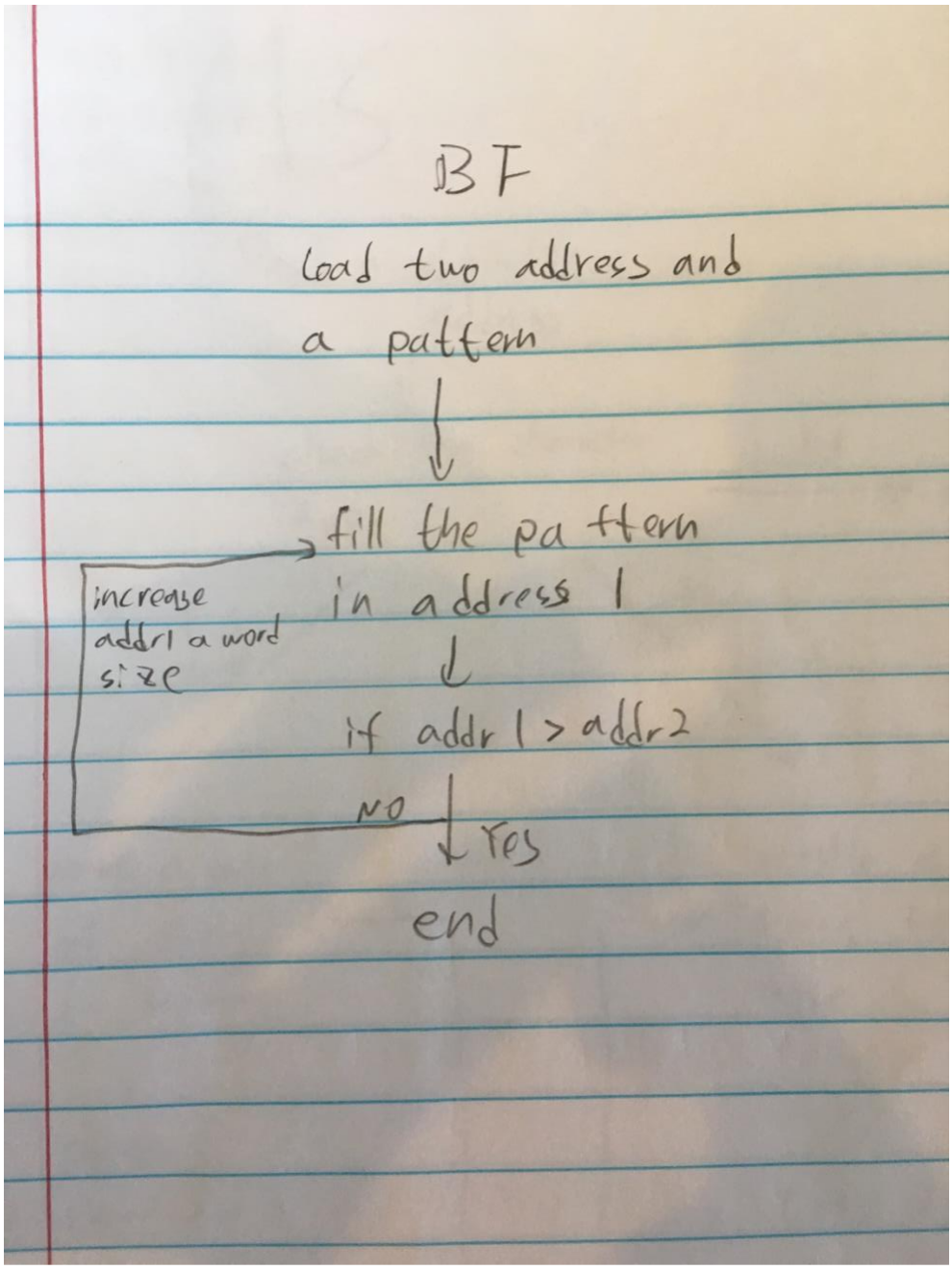


Figure 2.2.5.1. command interpret flowchart

2.2.5.2-) Block Fill 5 Assembly Code

```
BF
          MOVEM.L   D0-D7/A0-A6,-(SP)
          BSR       LOAD_TWO_ADDR       ;LOAD TWO ADDRESS AND STORE IT IN D4 AND D5
          MOVEA.L   D4,A5               ;TWO ADDRESS STORE IN A5 AND A6
          MOVEA.L    D5,A6
          MOVE.W    (A5),D0             ;CHECK IF TWO ADDRESS ARE EVEN, IF NOT, TRIGGER
EXCEPTION
          MOVE.W    (A6),D0
          BSR       LOAD_ONE_ADDR       ;LOAD THE PATTERN AND STORE IN D3
          ;BRA   TEST_SUCCESS
BF_FILL_LOOP
          MOVE.W    D3,(A5)+    FILL A BLOCK
          CMPA.L    A5,A6      ;COMPARE IF REACH THE END
          BGE       BF_FILL_LOOP
          MOVEM.L   (SP)+, D0-D7/A0-A6
          RTS
```

Figure 2.2.5.1 BF Assembly Code

It is similar to 2.2.1.2

2.2.6-) Memory Set(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.

2.2.6.1-) Debugger Command #6 Algorithm and Flowchart

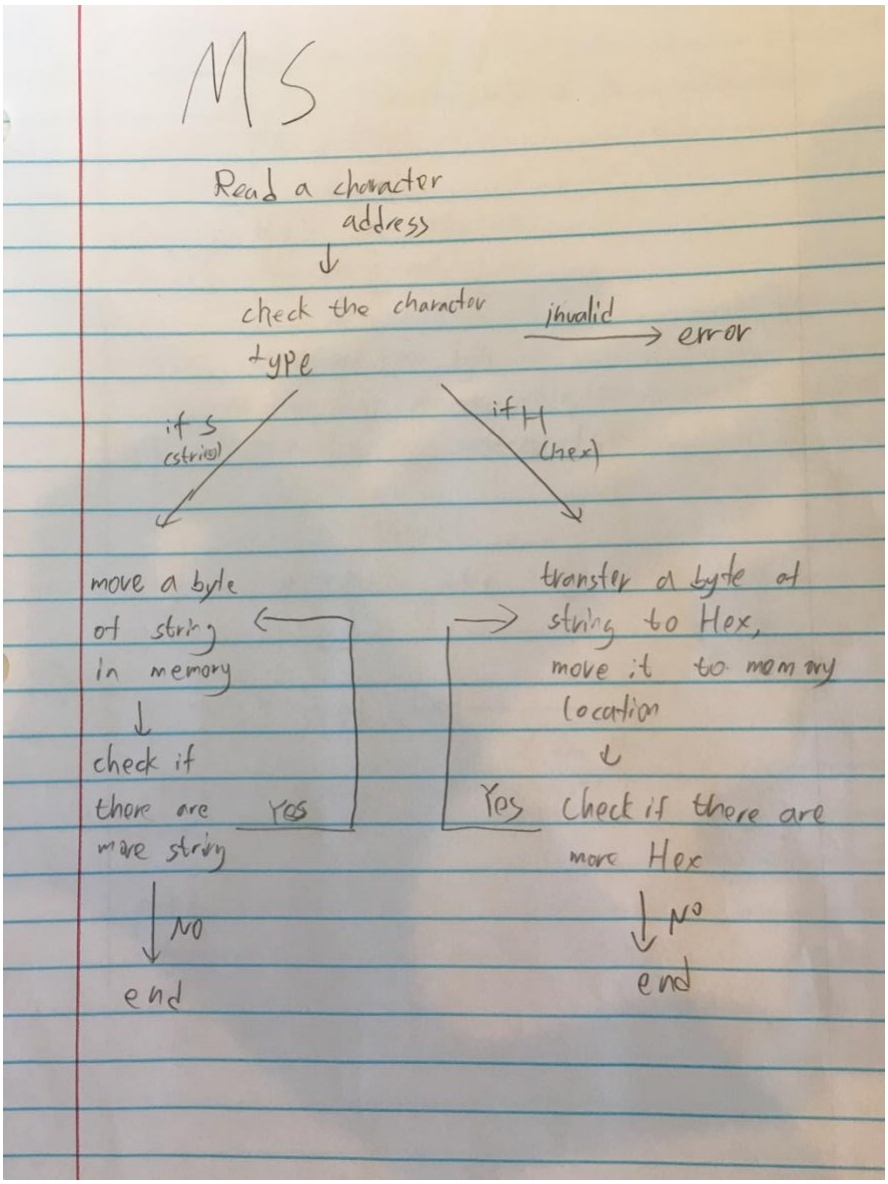


Figure 2.2.6.1. command interpret flowchart

2.2.6.2-) Debugger Command #6 Assembly Code

```
MS
        MOVEM.L   D0-D7/A0-A6,-(SP)
        BSR       ASCII_READ ;READ A CHARACTER IN D2

        BSR       LOAD_ONE_ADDR  ;LOAD ADDRESS AND MOVE IT FROM D3 TO A5
        MOVEA.L   D3,A5
        BSR       IS_NEXT
        BSR       CLEAR_SPACE       ;CHECK IS THERE MODIFIED CONTENT
        CMP.B     #1,D6
        BEQ       MS_LOOP
        BRA       ERROR_MESG        ;ERROR IF NO CONTENT
MS_LOOP
        *CHECK IF IT'S STRING OR HEX
        CMPI.B    #$53,D2
        BEQ       MS_LOOP_STRING
        CMPI.B    #$48,D2
        BEQ       MS_LOOP_HEX
        BRA       ERROR_MESG

MS_LOOP_STRING
        MOVE.B    (A4)+,(A5)+
        BSR       IS_NEXT
        CMP.B     #1,D6
        BEQ        MS_LOOP_STRING
        MOVEM.L   (SP)+, D0-D7/A0-A6
        RTS
MS_LOOP_HEX
        MOVE.B    (A4)+,D2
        MOVEA.L    A4,A6        ;A6 STORE A COPY OF A4
        MOVE.B     #$20,-(A4)   ;WRITE A SPACE TO FIT THE CONVERSION SUBROUTINE
        MOVE.B     D2,-(A4)
        BSR        LOAD_ONE_ADDR    ;LOAD A ASCII AND TRANSFER IT TO HEX IN D3
        MOVE.B     D3,D5            ;D5 STORE THE FIRST HEX
        ASL.B      #4,D5
        MOVEA.L    A6,A4            ;RESTORE A4

        MOVE.B    (A4)+,D2
        MOVEA.L    A4,A6        ;A6 STORE A COPY OF A4
        MOVE.B     #$20,-(A4)   ;WRITE A SPACE TO FIT THE CONVERSION SUBROUTINE
        MOVE.B     D2,-(A4)
        BSR        LOAD_ONE_ADDR    ;LOAD A ASCII AND TRANSFER IT TO HEX IN D3
        ADD.B      D3,D5            ;FINISH TRANSFER TWO ASCII
        MOVEA.L    A6,A4            ;RESTORE A4

        MOVE.B     D5,(A5)+

        BSR       IS_NEXT
        CMP.B     #1,D6
        BEQ       MS_LOOP_HEX
        MOVEM.L   (SP)+, D0-D7/A0-A6
        RTS
```

Figure 2.2.6.1 HELP Assembly Code

2.2.7-) Block Search (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.

2.2.7.1-) Block Search Algorithm and Flowchart

2.2.7.2-) Block Search Assembly Code

```
BSCH
        MOVEM.L   D0-D7/A0-A6,-(SP)
        BSR       LOAD_TWO_ADDR  ;LOAD TWO ADRESS IN D4,AND D5, STORE IT IN A2,A3
        MOVEA.L   D4,A2
        MOVEA.L   D5,A3
        CMPA.L    A2,A3         ;CHECK IF A3 GREATER THAN A2
        BGT       BSCH_NEXT     ;IF GREATER , GO ON
        BRA       ERROR_MESG

BSCH_NEXT
        BSR       IS_NEXT
        BSR       CLEAR_SPACE       ;CHECK IS THERE CONTENT FOR SEARCHINH
        MOVEA.L   A4,A5            ;MAKE A5 A COPY OF A4 TO SEARCH
        MOVEA.L   A2,A6            ;MAKE A COPY OF START STRING ADDRSS IF FIND
        CMP.B     #1,D6
        BEQ       BSCH_SEARCH
        BRA       ERROR_MESG        ;ERROR IF NO CONTENT

BSCH_SEARCH

        CMP.B     #$0,(A5)       ;CHECK IF (A5) IS NULL
        BEQ       BSCH_SEARCH_MIGHT_FOUND
        CMPA.L    A2,A3            ;CHECK IF REACH THE END
        BLE       BSCH_SEARCH_NO_FOUND
        CMP.B     (A2)+,(A5)+      ;COMPARE THE STRING
        BEQ        BSCH_SEARCH     ;IF SAME, KEEP SEARCHING
        MOVEA.L    A2,A6            ;UPDATE THE COPY
        MOVEA.L    A4,A5           ;RESET A5 IF NOT EQUAL
        BRA        BSCH_SEARCH
BSCH_SEARCH_MIGHT_FOUND
        CMP.L     A2,A3        ;COMPARE IF IT'S  END
        BGE       BSCH_SEARCH_FOUND    ;IF A3 STILL GREATER THAN A3 FOUND
BSCH_SEARCH_NO_FOUND
        MOVE.B   #13,D0 ;PRINT NOT FOUND MESSAGE
        LEA      BSCH_NOT_FOUND_PROMPT,A1
        TRAP     #15
        BRA      BSCH_END
          BRA       BSCH_END

BSCH_SEARCH_FOUND
         MOVE.B   #1,D0  ;PRINT FOUND MESSAGE
         LEA      BSCH_FOUND_PROMPT_1,A1
         TRAP     #15
         MOVE.B   #13,D0     ;DISPLAY THE STRING
         MOVEA.L  A4,A1
         TRAP     #15

         MOVE.B   #1,D0
         LEA      BECH_FOUND_PROMPT_2,A1
         TRAP     #15

         MOVE.L   #$24,D1 ;DISPLAY '$'
         MOVE.L   #6,D0
         TRAP     #15
         MOVE.L   #16,D2    ;SETUP THE BASE
         MOVE.L   A6,D1     ;SETUP THE ADDRESS
         MOVE.L   #15,D0
         TRAP     #15

        MOVE.L     #$A,D1 ;DISPLAY LF
        MOVE.L     #6,D0
        TRAP       #15

        MOVE.L     #$D,D1 ;DISPLAY CR
        MOVE.L     #6,D0
        TRAP       #15

BSCH_END

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

Figure 2.2.7 BSCH Assembly Code

2.2.8-) Block Move(BMOV)

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

2.2.8.1-) Block Move Algorithm and Flowchart

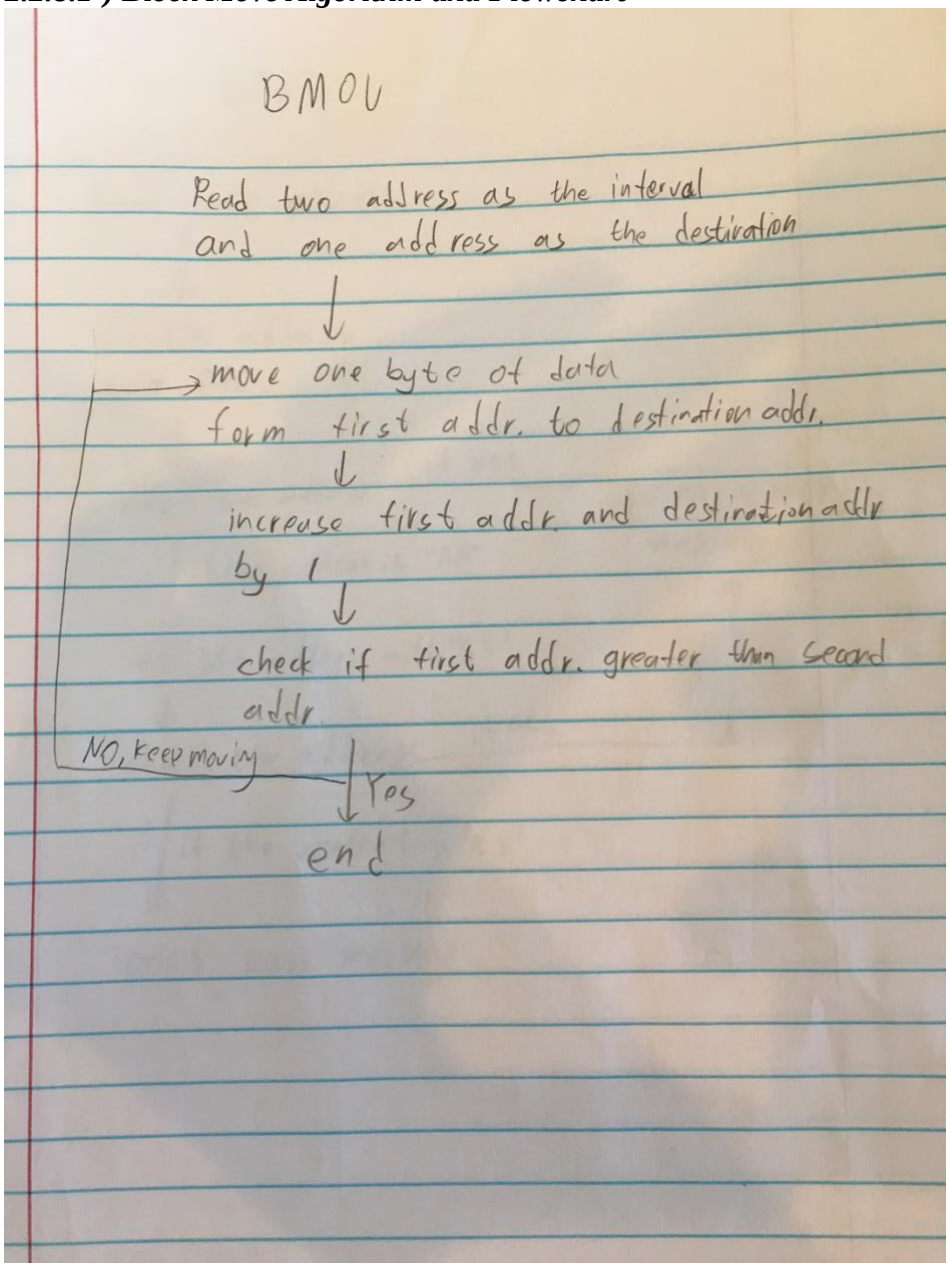


Figure 2.2.8.1. command interpret flowchart

2.2.8.2-) Block Move Assembly Code

```
          RTS
BMOV
*BLOCK MOVE
          MOVEM.L   D3-D6/A2-A5,-(SP)
          BSR       LOAD_TWO_ADDR   ;LOAD TWO ADRESS IN D4,AND D5, STORE IT IN A2,A3
          MOVEA.L   D4,A2
          MOVEA.L   D5,A3
          BSR       LOAD_ONE_ADDR   ;LOAD ONE ADDRESS IN D3,PUTIT IN A5
          MOVEA.L   D3,A5
          CMPA.L    A2,A3         ;CHECK IF A3 GREATER THAN A2
          BGT       BMOV_LOOP     ;IF GREATER , GO ON
          BRA       ERROR_MESG
BMOV_LOOP
          MOVE.B    (A2)+,(A5)+ ;MOVE A BYTE OF BLOCK
          CMP.L     A2,A3       ;CHECK IF IT IS END
          BGE       BMOV_LOOP   ;LOOP IF NOT END
          MOVEM.L   (SP)+, D3-D6/A2-A5
          RTS
```

Figure 2.2.8 HELP Assembly Code

2.2.9-) Block Test (BTST)

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.

If memory problems are found, a message is displayed indicating the address, the data stored, and the data read of the failing memory.

2.2.9.1-) Block Test Algorithm and Flowchart

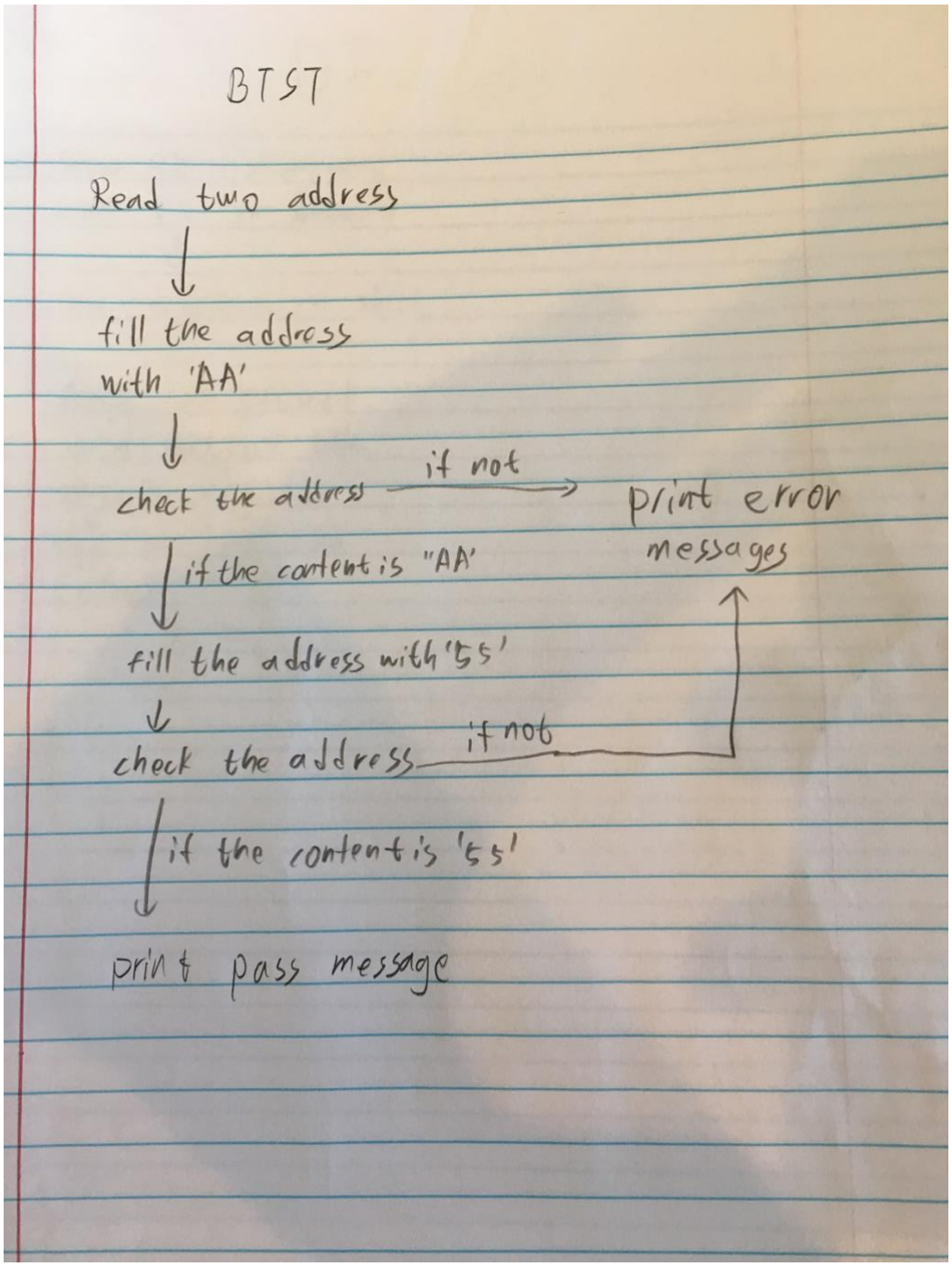


Figure 2.2.9.1. command interpret flowchart

2.2.9.2-) Block Test Assembly Code

```
          RTS
BTST
          MOVEM.L   D0-D7/A0-A6,-(SP)
          BSR       LOAD_TWO_ADDR   ;LOAD TWO ADRESS IN D4,AND D5, STORE IT IN A2,A3
          MOVEA.L   D4,A2
          MOVEA.L   D5,A3
          MOVEA.L   A2,A1       ;MAKE A COPY OF A1

BTST_WRITE_LOOP
           MOVE.B   #$AA,(A1)+       ;fill the memory
           CMPA.L   A1,A3           ;check if it is done
           BLT      BTST_AA_READ
           BRA      BTST_WRITE_LOOP         ;loop to fill the memory
BTST_AA_READ
           MOVEA.L  A2,A1       ;BACK UP TO THE BEGINNING ADDRESS
BTST_AA_READ_LOOP
           CMPI.B   #$AA,(A1)   ;CHECK IF IT'S AA
           BNE      BTST_FAIL_AA   ;PRINT FAIL ADDRESS
           MOVE.B   #$55,(A1)+  ;FILL THE MEMORY WITH 55
           CMPA.L   A1,A3       ;CHECK IF REACH THE END
           BLT      BTST_55_READ     ;IF END CHECK $55
           BRA      BTST_AA_READ_LOOP
BTST_55_READ
           MOVEA.L  A2,A1       ;BACK UP TO THE BEGINNING ADDRESS
BTST_55_READ_LOOP
           CMPI.B   #$55,(A1)   ;CHECK IF IT'S AA
           BNE      BTST_FAIL_55   ;PRINT FAIL ADDRESS
           ADDA.L   #1,A1
           CMPA.L   A1,A3       ;CHECK IF REACH THE END
           BLT      BTST_PASS    ;IF END CHECK $55
           BRA      BTST_55_READ_LOOP
```

```
BTST_FAIL_AA
          MOVEA.L  A1,A6   ;MAKE A1 TO A6 TO USE A1 TO PRINT

          LEA      BTST_FAIL_PROMPT1,A1    ;PRINT FAIL MESSAGE
          MOVE.L   #14,D0
          TRAP #15

          MOVE.L   A6,D1       ;OUTPUT ADDRESS
          MOVE.L   #16,D2       ;SETUP THE BASE
          MOVE.B   #15,D0      ;PRINT THE ERROR MESSAGE
          TRAP     #15         ;PRINT THE ERROR MESSAGE

          MOVE.L  #$A,D1 ;DISPLAY LF
          MOVE.L  #6,D0
          TRAP    #15

          MOVE.L  #$D,D1 ;DISPLAY CR
          MOVE.L  #6,D0
          TRAP    #15

          LEA      BTST_FAIL_PROMPT2,A1    ;PRINT FAIL MESSAGE
          MOVE.L   #14,D0
          TRAP #15

          CLR.L    D1
          MOVE.B  (A6),D1 ;DISPLAY THE DATA
          MOVE.L  #16,D2
          MOVE.W  #15,D0
          TRAP    #15

          MOVE.L  #$A,D1 ;DISPLAY LF
          MOVE.L  #6,D0
          TRAP    #15

          MOVE.L  #$D,D1 ;DISPLAY CR
          MOVE.L  #6,D0
          TRAP    #15

          LEA      BTST_FAIL_PROMPT3,A1    ;PRINT FAIL MESSAGE
          MOVE.L   #14,D0
          TRAP #15

          CLR.L    D1
          MOVE.B  #$AA,D1 ;DISPLAY THE AA
          MOVE.L  #16,D2
          MOVE.W  #15,D0
          TRAP    #15
```

```
            MOVE.L  #$A,D1 ;DISPLAY LF
            MOVE.L  #6,D0
            TRAP    #15

            MOVE.L  #$D,D1 ;DISPLAY CR
            MOVE.L  #6,D0
            TRAP    #15

            BRA      BTST_DONE

BTST_FAIL_55
             MOVEA.L  A1,A6   ;MAKE A1 TO A6 TO USE A1 TO PRINT

            LEA      BTST_FAIL_PROMPT1,A1    ;PRINT FAIL MESSAGE
            MOVE.L   #14,D0
            TRAP #15

            MOVE.L   A6,D1       ;OUTPUT ADDRESS
            MOVE.L   #16,D2       ;SETUP THE BASE
            MOVE.B   #15,D0      ;PRINT THE ERROR MESSAGE
            TRAP     #15         ;PRINT THE ERROR MESSAGE

            MOVE.L  #$A,D1 ;DISPLAY LF
            MOVE.L  #6,D0
            TRAP    #15

            MOVE.L  #$D,D1 ;DISPLAY CR
            MOVE.L  #6,D0
            TRAP    #15

            LEA      BTST_FAIL_PROMPT2,A1    ;PRINT FAIL MESSAGE
            MOVE.L   #14,D0
            TRAP #15

            CLR.L    D1
            MOVE.B  (A6),D1 ;DISPLAY THE DATA
            MOVE.L  #16,D2
            MOVE.W  #15,D0
            TRAP    #15

            MOVE.L  #$A,D1 ;DISPLAY LF
            MOVE.L  #6,D0
            TRAP    #15

            MOVE.L  #$D,D1 ;DISPLAY CR
            MOVE.L  #6,D0
            TRAP    #15
```

```
            LEA      BTST_FAIL_PROMPT3,A1    ;PRINT FAIL MESSAGE
            MOVE.L   #14,D0
            TRAP #15

            CLR.L    D1
            MOVE.B  #$55,D1 ;DISPLAY THE 55
            MOVE.L  #16,D2
            MOVE.W  #15,D0
            TRAP    #15

            MOVE.L  #$A,D1 ;DISPLAY LF
            MOVE.L  #6,D0
            TRAP    #15

            MOVE.L  #$D,D1 ;DISPLAY CR
            MOVE.L  #6,D0
            TRAP    #15

            BRA      BTST_DONE


BTST_PASS
            LEA      BTST_PASS_PROMPT,A1    ;PRINT THE PASS MESSAGE
            MOVE.L   #13,D0
            TRAP #15


BTST_DONE
            MOVEM.L   (SP)+, D0-D7/A0-A6
            RTS
```

Figure 2.2.9.2 BTST Assembly Code

2.2.10-) Execute Program(GO)

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

2.2.10.1-) Execute Program Command Algorithm

There is no need to draw a flow chart for the go command. The command just take the address and branch to subroutine.

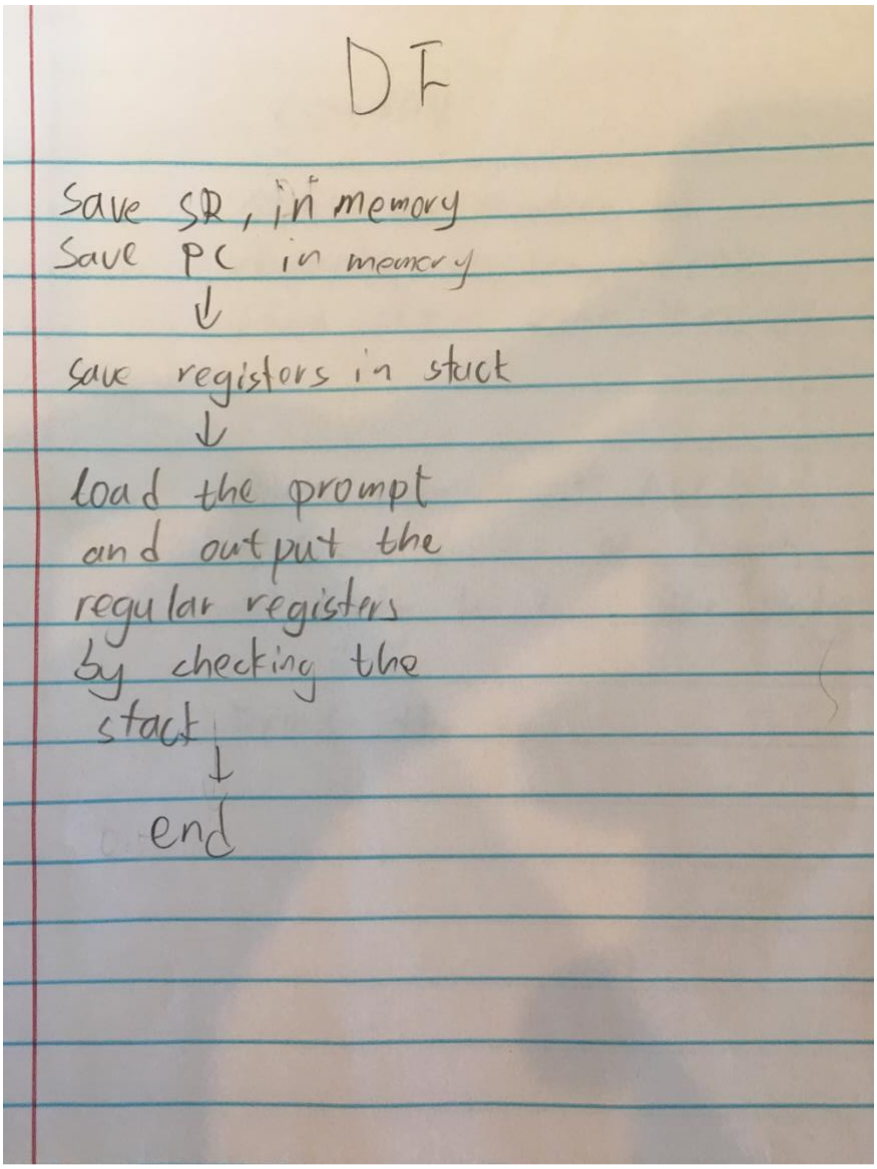
2.2.10.2-) Execute Program Command Assembly Code

```
GO
          MOVEM.L   D0-D7/A0-A6,-(SP)
          BSR       LOAD_ONE_ADDR   ;LOAD ONE ADDRESS IN D3, MOVE IT TO A5
          MOVEA.L   D3,A5
          JSR       (A5)        ;EXECUTE THE PROGRAM
          MOVEM.L   (SP)+,D0-D7/A0-A6
          RTS
```

Figure 2.2.10.1 BTST Assembly Code

2.2.11-) Display Formatted Registers(DF)

2.2.11.1-) Display Formatted Registers Algorithm and Flowchart



2.2.11.2-) Display Formatted Registers Assembly Code

```
            RTS
DF
            MOVE.W     SR,DF_SR        ;SAVE SOME REGISTERS IN THE MEMORY FIRST
            MOVE.L     (A7),DF_PC
            MOVE.L     A7,DF_SS
            MOVEM.L    D0-D7/A0-A6,-(SP) ;SAVE REGISTERS IN STACK

            MOVEA.L    A7,A3    ;MOVE STACK POINTER TO A3 TO OUTPUT REGISTERS
DF_D0_D3
            LEA        DF_PROMPT1,A1    ;LOAD THE PROMPT
            MOVE.L     (A3)+,D2
            BSR        DF_OUTPUT_PROMPT ;PRINT D0 PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D0

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D1

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D2

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D3

            MOVE.L     #$A,D1 ;DISPLAY LF
            MOVE.L     #6,D0
            TRAP       #15

            MOVE.L     #$D,D1 ;DISPLAY CR
            MOVE.L     #6,D0
            TRAP       #15
DF_D4_D7
            LEA        DF_PROMPT2,A1    ;LOAD THE PROMPT
            MOVE.L     (A3)+,D2
            BSR        DF_OUTPUT_PROMPT ;PRINT D4 PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D4

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D5

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D6

            MOVE.L     (A3)+,D2
            ADDA.L     #4,A1
            BSR        DF_OUTPUT_PROMPT
            BSR        DF_OUTPUT_STACK  ;OUTPUT D7

            MOVE.L     #$A,D1 ;DISPLAY LF
            MOVE.L     #6,D0
            TRAP       #15

            MOVE.L     #$D,D1 ;DISPLAY CR
            MOVE.L     #6,D0
            TRAP       #15
```

```
DF_A0_A3
        LEA     DF_PROMPT3,A1    ;LOAD THE PROMPT
        MOVE.L  (A3)+,D2
        BSR     DF_OUTPUT_PROMPT ;PRINT D4 PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A0

        MOVE.L  (A3)+,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A1

        MOVE.L  (A3)+,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A2

        MOVE.L  (A3)+,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A3

        MOVE.L  #$A,D1 ;DISPLAY LF
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$D,D1 ;DISPLAY CR
        MOVE.L  #6,D0
        TRAP    #15
DF_A4_A6
        LEA     DF_PROMPT4,A1    ;LOAD THE PROMPT
        MOVE.L  (A3)+,D2
        BSR     DF_OUTPUT_PROMPT ;PRINT D4 PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A4

        MOVE.L  (A3)+,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A5

        MOVE.L  (A3)+,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A6

        MOVE.L  DF_SS,D2
        ADDA.L  #4,A1
        BSR     DF_OUTPUT_PROMPT ;PRINT A7 PROMPT
        BSR     DF_OUTPUT_STACK  ;OUTPUT A7

        MOVE.L  #$A,D1 ;DISPLAY LF
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$D,D1 ;DISPLAY CR
        MOVE.L  #6,D0
        TRAP    #15
```

```
            TRAP        #15
DF_SSUSPCSR
            LEA         DF_PROMPT5,A1     ;LOAD THE PROMPT

            MOVE.L      DF_SS,D2
            ADDA.L      #4,A1
            BSR         DF_OUTPUT_PROMPT ;PRINT A7 PROMPT
            BSR         DF_OUTPUT_STACK  ;OUTPUT A7



            MOVE.L      USP,A6        ;US CAN ONLY MOVE INTO A REG
            MOVE.L      A6,D2
            BSR         DF_OUTPUT_PROMPT
            BSR         DF_OUTPUT_STACK  ;OUTPUT US

            MOVE.L      DF_PC,D2
            ADDA.L      #4,A1
            BSR         DF_OUTPUT_PROMPT
            BSR         DF_OUTPUT_STACK  ;OUTPUT PC


            ADDA.L      #4,A1
            BSR         DF_OUTPUT_PROMPT
            CLR.L       D1               ;USE ANOTHER WAY TO OUTPUT SR
            MOVE.W      DF_SR,D1
            MOVE.B      #15,D0
            MOVE.B      #16,D2  ;SET THE BASE
            TRAP        #15

        MOVE.L    #$A,D1 ;DISPLAY LF
        MOVE.L    #6,D0
        TRAP      #15

        MOVE.L    #$D,D1 ;DISPLAY CR
        MOVE.L    #6,D0
        TRAP      #15

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

        RTS

*DF SUBROUTINE TO OUTPUT 4 CHARACTER STORED IN A1
DF_OUTPUT_PROMPT
        MOVEM.L   D0-D7/A0-A6,-(SP)
        MOVE.W    #4,D1    ;OUTPUT 4 CHARACTERS
        MOVE.B    #1,D0
        TRAP      #15

        MOVEM.L   (SP)+,D0-D7/A0-A6
        RTS
* THE SUBROUTINE OUTPUT A LONG WORLD IN D2
DF_OUTPUT_STACK
        MOVEM.L   D0-D7/A0/A2,-(SP)
        MOVE.L    D2,D3        ;SAVE D2 TO D3
        ASR.L     #8,D2        ;OUT PUT LEFT WORD FIRST
        ASR.L     #8,D2        ;OUT PUT LEFT WORD FIRST
        BSR       WORD_TO_HEX  ;TAKE HEX IN D2 AND STORE THE HEX IN D5
*DF_OUT PUT THE LEFT WORD
        MOVE.L     D5,D1       ;OUTPUT THE 4 BYTE
        ASR.L      #8,D1
        ASR.L      #8,D1
        ASR.L      #8,D1
        MOVE.B     #6,D0
        TRAP       #15
```

```
            MOVE.L      D5,D1       ;OUTPUT THE 3 BYTE
            ASR.L       #8,D1
            ASR.L       #8,D1
            MOVE.B      #6,D0
            TRAP        #15

            MOVE.L      D5,D1       ;OUTPUT THE 2 BYTE
            ASR.L       #8,D1
            MOVE.B      #6,D0
            TRAP        #15

            MOVE.L      D5,D1       ;OUTPUT THE 1 BYTE
            MOVE.B      #6,D0
            TRAP        #15

DF_OUTPUT_THE_RIGHT_WORD
            MOVE.L      D3,D2   ;RESTOR D2 TO OUTPUT RIGHT WORD
            BSR         WORD_TO_HEX ;GET THE LEFT WORD HEX IN D5
            MOVE.L      D5,D1       ;OUTPUT THE FIRST BYTE
            ASR.L       #8,D1
            ASR.L       #8,D1
            ASR.L       #8,D1
            MOVE.B      #6,D0
            TRAP        #15

            MOVE.L      D5,D1       ;OUTPUT THE SECOND BYTE
            ASR.L       #8,D1
            ASR.L       #8,D1
            MOVE.B      #6,D0
            TRAP        #15

            MOVE.L      D5,D1       ;OUTPUT THE THIRD BYTE
            ASR.L       #8,D1
            MOVE.B      #6,D0
            TRAP        #15

            MOVE.L      D5,D1       ;OUTPUT THE FOURTH BYTE
            MOVE.B      #6,D0
            TRAP        #15


            MOVEM.L   (SP)+,D0-D7/A0/A2
            RTS
```

Figure 2.2.11.2 DF Assembly Code

2.2.12-) Exit Monitor Program(EXIT)

The EXIT command terminates/exits your Monitor program.

2.2.12.1-) EXIT Algorithm

The EXIT will call number 9 trap #15 functions to terminate the program.

2.2.12.2-) Debugger Command #12 Assembly Code

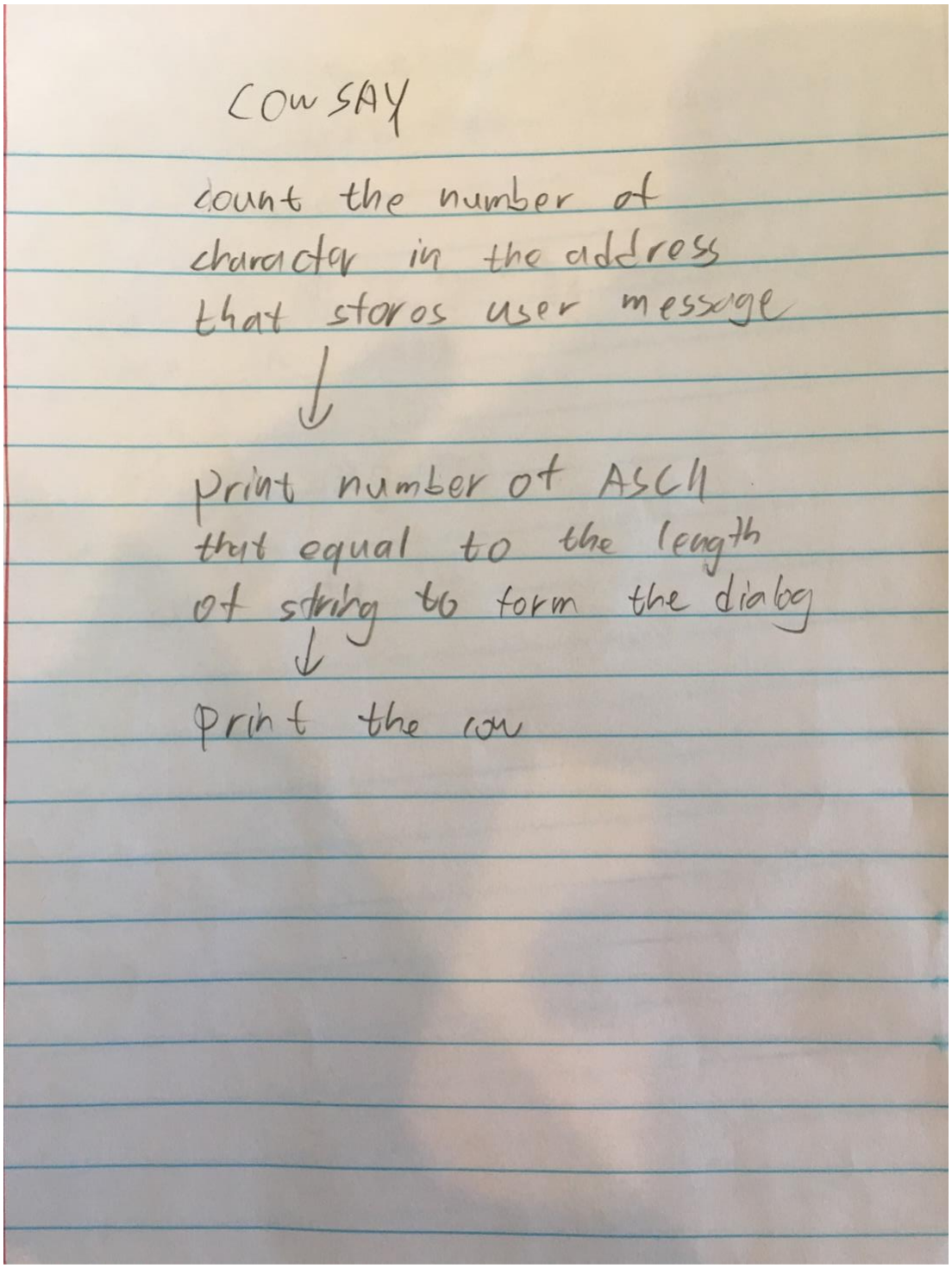
```
RTS
EXIT
        MOVE.L  #9,D0
        TRAP    #15

COWSAY
```

Figure 2.2.12.1EXIT Assembly Code

2.2.13-) COWSAY

The cowsay command will generate a ASCII picture of a cow and a dialog box with the given message from user.

2.2.12.1-) COWSAY Algorithm


COW SAY
count the number of
character in the address
that stores user message
print number of ASCII
that equal to the length
of string to form the dialog
print the cow

2.2.12.2-) COWSAY Assembly Code

```
COWSAY
        MOVEM.L   D0-D7/A0-A6,-(SP)

        BSR       IS_NEXT
        CMP.W     #0,D6
        BEQ       ERROR_MESG
COWSAY_COUNT_WORD
* COUNT THE WORD OF THE STRING
        MOVE.L    #1,D3 ;INITIAL WOLD NUMBER
        MOVEA.L   A4,A3 ;COPY A3 TO COUNT
        ADDA.L    #1,A3
COWSAY_COUNT_WORD_LOOP
        CMP.B     #$0,(A3)+ ;CHECK IF A3 REACH THE END
        BEQ       COWSAY_OUTPUT
        ADD.L     #1,D3
        BRA       COWSAY_COUNT_WORD_LOOP
COWSAY_OUTPUT
        ADD.L    #2,D3     ;LEAVE SOME SPACE
        MOVE.L    D3,D4     ;MAKE A COPY OF THE LENGTH
COWSAY_OUTPUT_LOOP1
        MOVE.L  #$2D,D1 ;DISPLAY '-'
        MOVE.L  #6,D0
        TRAP    #15
        SUBI    #1,D4
        CMP.L   #0,D4
        BGT     COWSAY_OUTPUT_LOOP1

        MOVE.L  #$A,D1 ;DISPLAY LF
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$D,D1 ;DISPLAY CR
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$7C,D1 ;DISPLAY '|'
        MOVE.L  #6,D0
        TRAP    #15
```

```
COWSAY_OUTPUT_MESSAGE
        MOVEA.L A4,A1   ;OUTPUT THE MESSAGE
        MOVE.B  #1,D0
        MOVE.W  #40,D1
        TRAP    #15

        MOVE.L  #$7C,D1 ;DISPLAY '|'
        MOVE.L  #6,D0
        TRAP    #15


        MOVE.L  #$A,D1 ;DISPLAY LF
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$D,D1 ;DISPLAY CR
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L    D3,D4     ;MAKE A COPY OF THE LENGTH
COWSAY_OUTPUT_LOOP2

        MOVE.L  #$2D,D1 ;DISPLAY '-'
        MOVE.L  #6,D0
        TRAP    #15
        SUBI    #1,D4
        CMP.L   #0,D4
        BGT     COWSAY_OUTPUT_LOOP2

        MOVE.L  #$A,D1 ;DISPLAY LF
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.L  #$D,D1 ;DISPLAY CR
        MOVE.L  #6,D0
        TRAP    #15

        MOVE.B    #0,D0         ;OUTPUT THE COW
        LEA       COW,A1
        MOVE.L    #255,D1
        TRAP     #15

COWSAY_END
        MOVEM.L   (SP)+,D0-D7/A0-A6
        RTS
ROLL
```

Figure 2.2.13.2EXIT Assembly Code

2.2.14-) Random(ROLL)

Generate a random number from 0 to 9.

2.2.12.1-) Random Algorithm and Flowchart

The program will first read the time of the system in the base 1/100 seconds. Then, the number of time will divide by base, in the program is 10. The remainder will be printed out as the random number.

2.2.12.2-) Random Assembly Code

```
ROLL
        MOVEM.L   D0-D7/A0-A6,-(SP)

       ; MOVE.L    #50,D1        ;DELAY BY HALF SECOND TO MAKE USER CALM DOWN
        ;MOVE.B    #23,D0        ;AND DONT PUT TOO MUCH TIME IN PLAYING IT
       ; TRAP      #15

        MOVE.B    #8,D0          ;GET THE CURRENT TIME AND STORE IT IN D1.L
        TRAP      #15
        AND.L     #$7FFFF,D1    ;PREVENT ONVERFLOW
        DIVU      #10,D1
        SWAP      D1

        CLR.L     D3             ;MAKE D1 ONLY HAVE D1.B
        MOVE.W    D1,D3
        CLR.L     D1
        MOVE.W    D3,D1
        MOVE.B    #15,D0         ;OUT PUT D1.W
        MOVE.W    #10,D2
        TRAP      #15

        MOVE.L    #$A,D1 ;DISPLAY LF
        MOVE.L    #6,D0
        TRAP      #15

        MOVE.L    #$D,D1 ;DISPLAY CR
        MOVE.L    #6,D0
        TRAP      #15
        MOVEM.L   (SP)+,D0-D7/A0-A6
        RTS
```

Figure 2.2.14.1EXIT Assembly Code

2.3-) Exception Handlers

The exception Handlers is to show user the exceptions when there is exception error occurred and help user locate the error.

2.3.1-) Bus Error Exception

When a Bus Error Exception occurred, the program will print BA, IR, SSW and call stack

2.3.1.1-) Bus Error Exception Algorithm and Flowchart

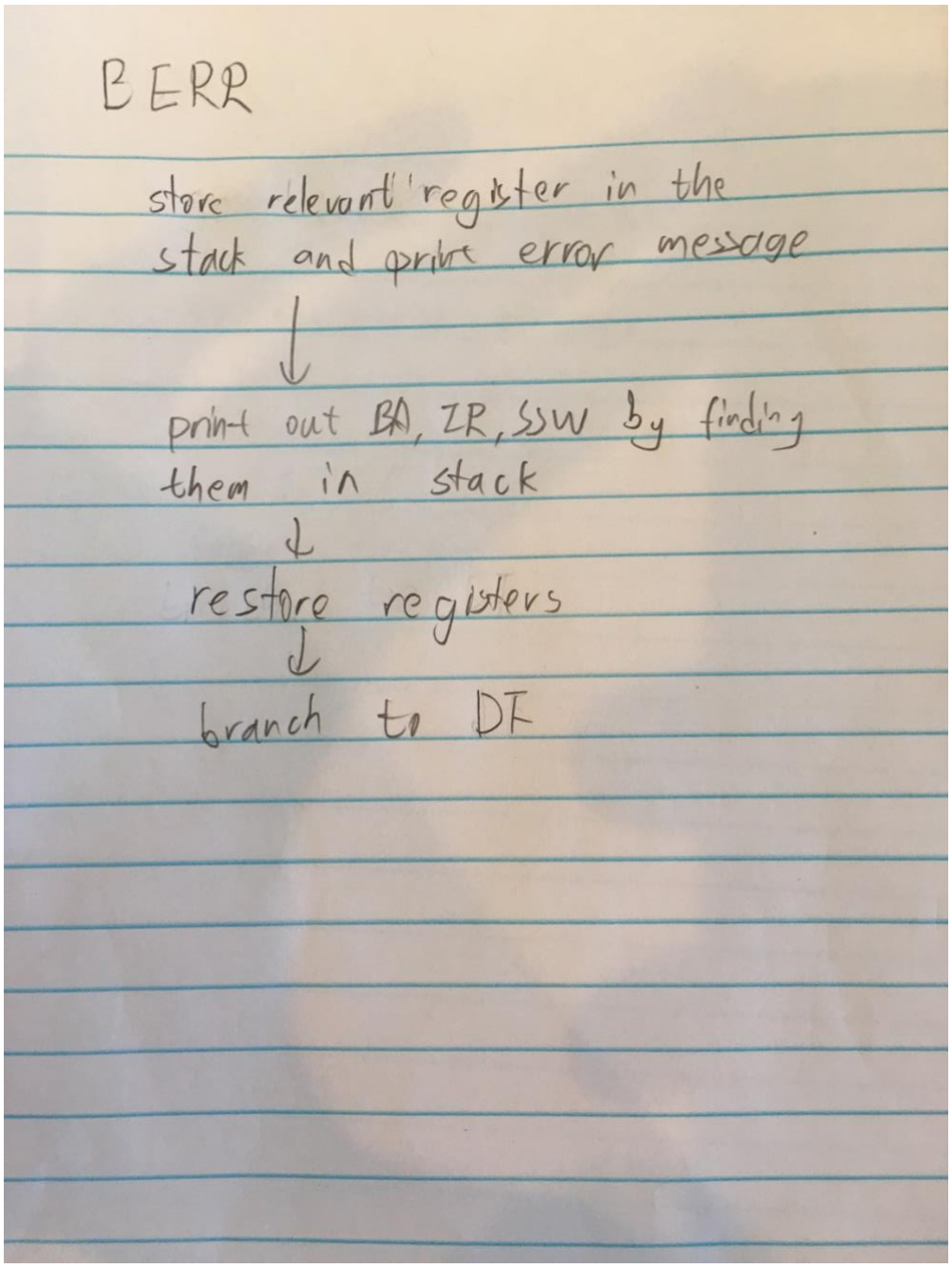


Figure 2.3.1.1. BERR Algorithm

Figure 2.9. Debugger Command # 1 Flowchart

2.3.1.2-) Bus Error Exception Assembly Code

```
BERR
        MOVEM.L A1/D0-D2, -(SP)
        LEA       BERR_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        BSR       OUTPUT_SPECIAL_REGISTERS
        MOVEM.L   (SP)+,A1/D0-D2
        BSR       DF
        BRA       SET_UP_REGISTERS
```

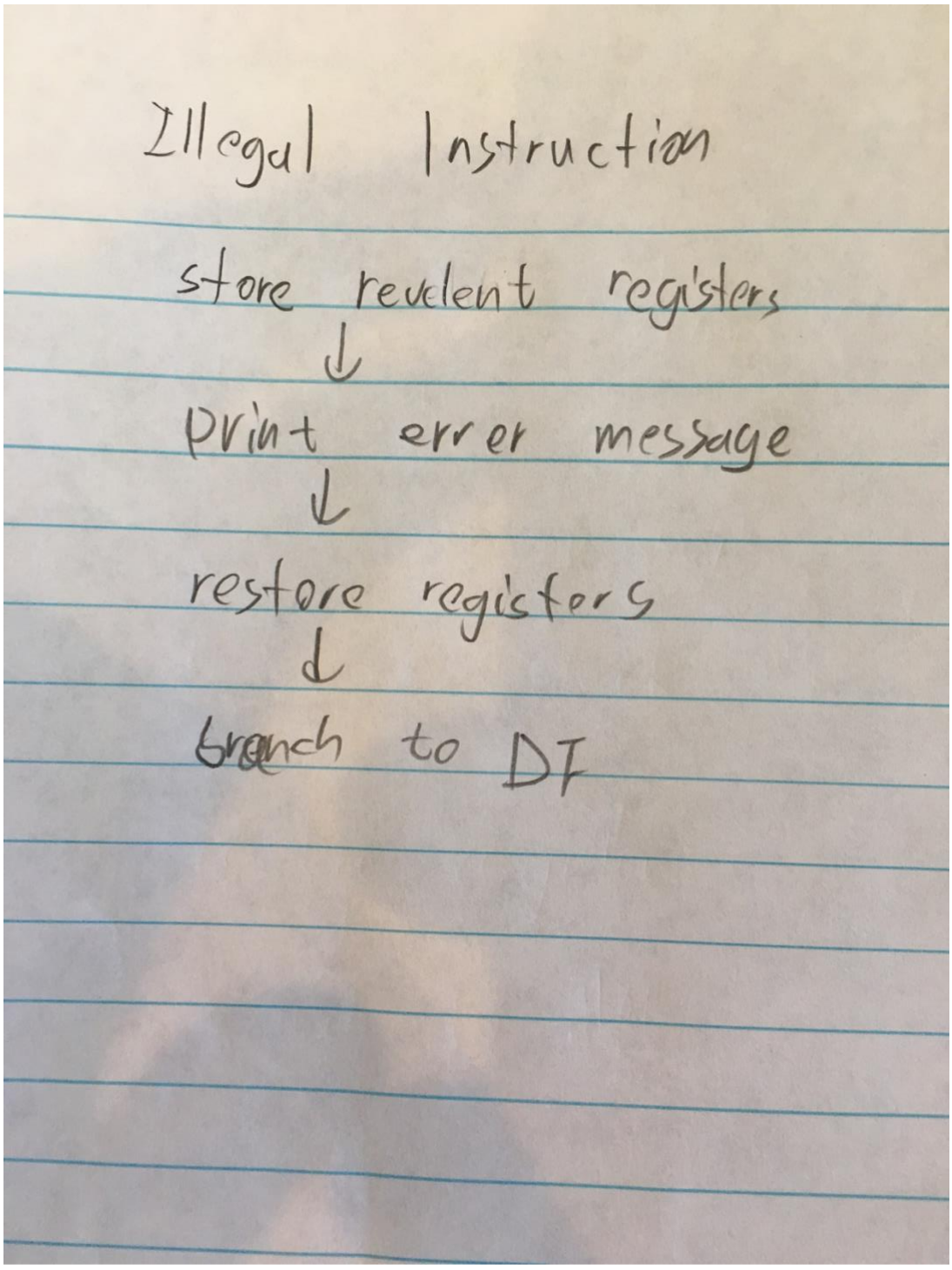
```
OUTPUT_SPECIAL_REGISTERS
* OUTPUT SSW BA IR
        LEA     BA,A1  ;PRINT BA MESG
        MOVE.L  #14,D0
        TRAP    #15

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

        LEA     IR,A1  ;PRINT IR MESG
        MOVE.L  #14,D0
        TRAP    #15

        MOVE.L  (26,A7),D1  ;DISPLAY IR
        MOVE.B  #15,D0
        MOVE.W  #16,D2
        TRAP    #15

        LEA     SSW,A1  ;PRINT SSW MESG
        MOVE.L  #14,D0
        TRAP    #15

        MOVE.L  (20,A7),D1  ;DISPLAY SSW
        MOVE.B  #15,D0
        MOVE.W  #16,D2
        TRAP    #15

        MOVE.L    #$A,D1 ;DISPLAY LF
        MOVE.L    #6,D0
        TRAP      #15

        MOVE.L    #$D,D1 ;DISPLAY CR
        MOVE.L    #6,D0
        TRAP      #15

        RTS
```

Figure 2.3.1.1. Bus Error Assembly Code

2.3.2-) Address Error Exception

2.3.1.1-) Address Error Algorithm and Flowchart

Same as Bus Error exception.



2.3.1.2-) Address Error Assembly Code

```
OUTPUT_SPECIAL_REGISTERS
* OUTPUT SSW BA IR
        LEA     BA,A1  ;PRINT BA MESG
        MOVE.L  #14,D0
        TRAP    #15

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

        LEA     IR,A1  ;PRINT IR MESG
        MOVE.L  #14,D0
        TRAP    #15

        MOVE.L  (26,A7),D1  ;DISPLAY IR
        MOVE.B  #15,D0
        MOVE.W  #16,D2
        TRAP    #15

        LEA     SSW,A1  ;PRINT SSW MESG
        MOVE.L  #14,D0
        TRAP    #15

        MOVE.L  (20,A7),D1  ;DISPLAY SSW
        MOVE.B  #15,D0
        MOVE.W  #16,D2
        TRAP    #15

        MOVE.L    #$A,D1 ;DISPLAY LF
        MOVE.L    #6,D0
        TRAP      #15

        MOVE.L    #$D,D1 ;DISPLAY CR
        MOVE.L    #6,D0
        TRAP      #15

        RTS
```

```
AERR
      MOVEM.L A1/D0-D2, -(SP)
      LEA       AERR_MESG,A1     ;OUTPUT ERRROR MESSAGE
      MOVE.W    #40,D1
      MOVE.B    #0,D0
      TRAP      #15
      BSR       OUTPUT_SPECIAL_REGISTERS
      MOVEM.L   (SP)+,A1/D0-D2
      BSR       DF
      BRA       SET_UP_REGISTERS
```

Figure 2.3.2.1. Bus Error Assembly Code

2.3.3-) Illegal Instruction Exception

2.3.3.1-) Illegal Instruction Exception Algorithm and Flowchart

When exception occurred, the program will save the relevant register first and then output the error message. After that, the program will restore the registers and branch to DF.

2.3.3.2-) Illegal Instruction Exception Assembly Code

```
ILL_INSTR
        MOVEM.L   A1/D0,-(SP)
        LEA       ILL_INSTR_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1/D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

Figure 2.3.3.1. Bus Error Assembly Code

2.3.4-) Privilege Violation Exception

2.3.4.1-) Privilege Violation Exception Algorithm and Flowchart

Same as illegal instruction exception

2.3.4.2-) Privilege Violation Exception Assembly Code

```
PRIV_VIOL
        MOVEM.L   A1/D0,-(SP)
        LEA       PRIV_VIOL_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1/D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

Figure 2.3.4.1. Bus Error Assembly Code

2.3.5-) Divide by Zero Exception

2.3.5.1-) Divide by Zero Exception Algorithm and Flowchart

Same as illegal instruction exception

2.3.5.2-) Divide by Zero Exception Assembly Code

```
ZERO_DEV
        MOVEM.L   A1/D0,-(SP)
        LEA       ZERO_DEV_MESG,A1      ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1/D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

Figure 2.3.5.2. Bus Error Assembly Code

2.3.6-) CHK instruction exceptions

2.3.6.1-) CHK instruction exceptions Algorithm and Flowchart

Same as illegal instruction exception

2.3.6.2-) CHK instruction exceptions Assembly Code

```
CHK_INSTR
        MOVEM.L   A1/D0,-(SP)
        LEA       CHK_INSTR_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1/D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

Figure 2.3.6.1 Bus Error Assembly Code

2.3.7-) Line A and Line F Emulators

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

Same as illegal instruction exception

2.3.7.2-) Line A and Line F Emulators Assembly Code

```
LINEA_EMU
        MOVEM.L   A1/D0,-(SP)
        LEA       LINEA_EMU_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1/D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

```
LINEF_EMU
        MOVEM.L   A1/D0,-(SP)
        LEA       LINEF_EMU_MESG,A1     ;OUTPUT ERRROR MESSAGE
        MOVE.W    #40,D1
        MOVE.B    #0,D0
        TRAP      #15
        MOVEM.L   (SP)+,A1,D0
        BSR       DF
        BRA       SET_UP_REGISTERS
```

Figure 2.3.7.2. Bus Error Assembly Code

3-) Discussion

There are lots of design challenges in the project. First one is to find the common part of all the programs and write the subroutine. Also, debugging is also a hard part in the program. Sometimes to find where the bug is, I need to spend several hours and check the instruction over and over again. The Easy68k is not a good IDE so the program interface is also very good to write a project.

4-) Feature Suggestions

Since I don't have much time, I cannot implement everything I want to in the additional commands. I'll just write done my idea in here. For the cowsay command, in the original Linux command, there are lots of other animals and the command can generate random one. Mine command can only generate same cow each time. For the random command, I can add a new feature that let user input a range so user had more space to use that. For the other commands, I think it might be too much for students, I use two weeks to finish that and found some of commands are similar, if the target of the project it's to let student understand TUTOR and more familiar with 68k, I think the number of commands can be shrinked.

5-) Conclusion

In conclusion, I learned a lot while writing the project. I also have a deeper understanding about TUTOR and how it works with memory. Overall, It's a very good experience of implementing a debugger command tool by my own. The project also let me go over most of things we learn in the 441.

6-) References

[1] M68K_Prog_Ref_Manual, Motorola
[2] ECE441 Exam Handout