

ECE 441

Microprocessors

Instructor: Dr. Jafar Saniie
Teaching Assistant: Guojun Yang

Final Project Report:
MONITOR PROJECT
04/26/17

By: Mete Morris

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

Signature: _____

Table of Contents

Abstract	2
1-) Introduction	3
2-) Monitor Program	4
2.1-) Command Interpreter	6
2.1.1-) Algorithm and Flowchart	7
2.1.2-) 68000 Assembly Code	8
2.2-) Debugger Commands	16
2.2.1-) Help	16
2.2.2-) Memory Display	18
2.2.3-) Sort	24
2.2.4-) Memory Modify	26
2.2.5-) Memory Set	33
2.2.6-) Block Fill	35
2.2.7-) Block Move	37
2.2.8-) Block Test	39
2.2.9-) Block Search	43
2.2.10-) Execute Program	46
2.2.11-) Display Formatted Registers	47
2.2.12-) Exit Monitor Program	54
2.2.13-) Ascii to Hex and Hex to Ascii Convertor	55
2.2.14-) Clear Registers	57
2.3-)Exception Handlers	59
2.3.1-) Bus Error Exception	59
2.3.2-) Address Error Exception	61
2.3.3-) Illegal Instruction Exception	63
2.3.4-) Privilege Violation Exception	65
2.3.5-) Divide by Zero Exception	67
2.3.6-)Check Instruction Exception	69
2.3.7-) Line A and Line F Emulators	71
2.4-)Extra Helper Functions	73
2.4.1-) ASCII to HEX Convertor	73
2.4.1-) Hex to ASCII Convertor	75
3-) Quick User Manual	77
4-) Discussion	78
5-) Feature Suggestions	79
6-) Conclusions	79
7-) References	80

Abstract

This paper includes the manual and report of a Command Prompt like application for 68k processors that works on easy 68k program. The paper includes description of implementations of 14 debugger commands and 8 exception handling routines, a quick user manual and a discussion on the implementation and engineering challenges.

1-) Introduction

The main goal of this project was to build a software that will act in a similar way to TUTOR software installed in the lab computers. Twelve predefined commands and two custom developer designed commands were implemented for this project. ECE 441 Lab 2 and Lab 3 material was heavily utilized for development.

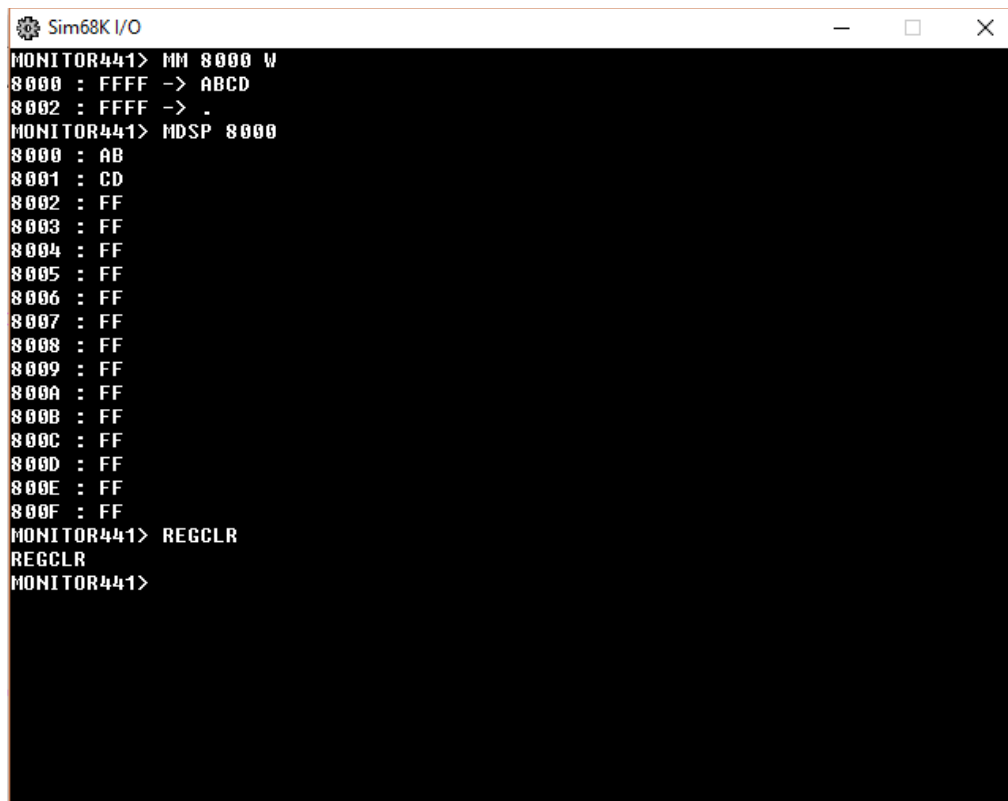
In addition to these fourteen debugger commands, exception handling for 8 different exception routines were implemented. As per the requirements, no Macros were used in this project.

2-) Monitor Program

A clear description of your design should be given here - what this program will do, requirements, etc. You may include a block diagram or table.

The main goal of this program is to behave like the Tutor software installed in the LAB's SANPER units. The program starts with initializing the error vectors to the custom error vectors required by the assignment. After that the Monitor Program stores all the A and D registers into stack and goes into Command Interpreter Part of the code where a command and arguments is expected from the user. Upon recognizing a command and right set of attributes (or sometimes lack of attributes) the monitor branches into subroutine, performs the required actions and restores the A and D registers. After restoration of the registers is done the program is sent to the beginning of command interpreter again. The only exception of the register restoration rule is REGCLR function and this will be elaborated further on the functions upcoming discussion section. The register restoration part is often ignored for flowcharts below as it is a procedure repeated many times. Unless otherwise stated, reader should assume that registers are always restored after the Debugger Commands are executed.

This monitor contains 14 debugger commands and handles 8 different exceptions.



```
Sim68K I/O
MONITOR441> MM 8000 W
8000 : FFFF -> ABCD
8002 : FFFF -> .
MONITOR441> MDSP 8000
8000 : AB
8001 : CD
8002 : FF
8003 : FF
8004 : FF
8005 : FF
8006 : FF
8007 : FF
8008 : FF
8009 : FF
800A : FF
800B : FF
800C : FF
800D : FF
800E : FF
800F : FF
MONITOR441> REGCLR
REGCLR
MONITOR441>
```

Figure 1: Monitor Program

In addition to those functions 2 helper functions that Convert Hexadecimal number to ASCII and ASCII to Hexadecimal were implemented. These functions were implemented to assist with interpreting commands with various other sub parts of the code. These two helper functions should not be confused with debugger command that does ASCII to HEX conversion.

Another additional feature is that, once the Command Interpreter detects that a right input function is implement but there is a mismatch in the algorithm it prints out a specific help message to that Debugger command.

The only other limitation this program has other that operational constraints mentioned in the Introduction is that the longest argument size with the exception of MM L command is four characters. Any argument that is longer than four characters will be cut off.

2.1-) Command Interpreter

Command Interpreter works like a linear structure. All commands have their argument interpreters and branching to functions in this part of the program. Command interpreter starts with initializing the error vector, this is so that the custom error handling routines included in this program is utilized rather than the default ones. The CI (command interpreter) then prints out the prompt 'MONITOR441->' and once ENTER key is pressed on the keyboard it starts comparing the entered set of characters to various commands. All commands are written in a table prior to initialization and this table is referred to compare the given input to the list of commands.

Once input is entered the first characters are compared to the first command in the Checker in the CI which is HELP. If each character entered matches the HELP phrase and is terminated with a null character and carriage return HELP function is executed.

If the command is not HELP a counter that points to the position of the character we are reading resets to the beginning of the entered command and the next command is compared to the input. This goes on until either there is a match or no match. In case that there is no match an error, message is displayed telling the user to check the manual.

A slightly more complicated case is when the Debugger Command has arguments. In this case once the right command is entered and there is a space before the arguments the interpreter goes into the parsing of these arguments. To make explanation easier MDSP command will be utilized. Once MDSP is entered with its arguments the system looks if there is an argument for start address. If there is no start argument but there is a space, a help page is displayed telling the user the correct way to use MDSP. If there is a start argument than the CI then checks if there is another argument for the end part of the memory. In this case if there is no ending address this is still a valid entry since MDSP is designed to handle both cases. If there is no second argument MDSP function that handles this case is called. If there is a second argument the MDSP function that handles this case is called.

The following flowchart explains this functionality.

2.1.1-) Flowchart

Overall Command Interpreter Flowchart.

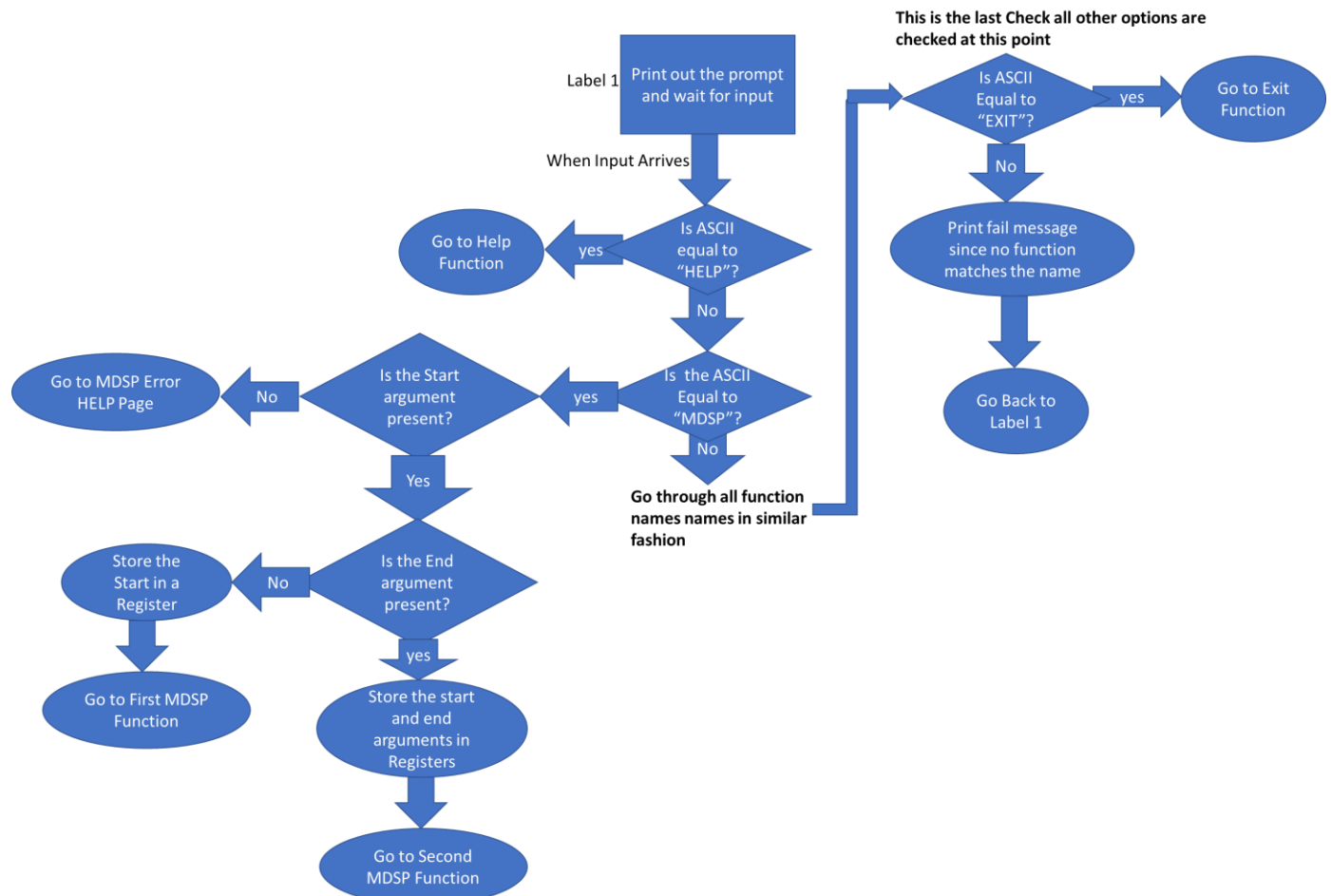


Figure 2: Command Interpreter Architecture

2.1.2-) Command Interpreter Assembly Code

```

START:                                ; first instruction of program

;INITIALIZE ERROR VECTORS
MOVE.L #BUSERRORFUNCTION,$08
MOVE.L #ADDRESSERRORFUNCTION,$0C
MOVE.L #ILLEGALINSTRUCTIONFUNCTION,$10
MOVE.L #PRIVELEGEVIOLATIONFUNCTION,$20
MOVE.L #DIVIDEBYZEROFUNCTION,$14
MOVE.L #CHECKINSTRUCTIONFUNCTION,$18
MOVE.L #LINEAEMULATORFUNCTION,$28
MOVE.L #LINEFEMULATORFUNCTION,$2C

pSTART ; Prints the Monitor prompt and saves the registers
MOVEM.L A0-A6/D0-D7,-(SP)
MOVEA.L #PROMPT,A1
MOVE.B #14,D0
TRAP #15
LEA $4000,A1
MOVE.B #2,D0
TRAP #15

*COMPARING INPUT TO THE MENU ITEMS*
MOVE.L #HELPPROMPT,A0
CLR.L D3 ;COUNTER FOR REVERSING A1 TO USER INPUT
COMPAREHELP
ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPAREMDSP
CMPI.B #00,-1(A0) ;CHECK IF THE BYTE JUST COMPARED WAS NULL TERMINATOR
BNE COMPAREHELP
BSR HELPFUNCTION

COMPAREMDSP
SUB.L D3,A1
CLR.L D3
MOVE.L #MDSPPROMPT,A0
COMPAREMDSP2
ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPARESORTW
CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
BNE COMPAREMDSP2
;CHECK FOR THE ARGUMENTS
CLR.L D4 ;FIRST ARG
CLR.L D5 ;SECOND ARG
CMPMDSPL1
ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ MDSPFUNCTION1 ;FUNCTION FOR 1 ARG
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPMDSPL2
LSL.L #8,D4

```

```

        BSR CMPMDSPL1
CMPMDSPL2
        ADD.B (A1)+,D5      ; MOVE THE SECOND MEMORY LOCATION NUMBER TO D5
        CMPI.B #0,(A1)     ;COMPARING LAST PART TO EMPTY STRNG
        BEQ MDSPFUNCTION2  ;FUNCTION FOR 2 ARG
        LSL.L #8,D5
        BSR CMPMDSPL2

COMPARESORTW      ;3 arguments, d4 is start,d5 is end, d6 is the ascending or
descending
        SUB.L D3,A1
        CLR.L D3
        MOVE.L #SORTWPROMPT,A0
COMPARESORTW2
        ADD.L #1,D3
        CMPM.B (A0)+,(A1)+
        BNE COMPAREMM
        CMPI.B #32,-1(A0)  ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
        BNE COMPARESORTW2
        ;CHECK FOR THE ARGUMENTS
        CLR.L D4 ;FIRST ARG
        CLR.L D5 ;SECOND ARG
        CLR.L D6 ;THIRD ARG
CMPSORTWL1
        ADD.B (A1)+,D4      ; MOVE THE MEMORY LOCATION NUMBER TO D4
        CMPI.B #0,(A1)     ;COMPARING LAST PART TO EMPTY STRNG
        BEQ SORTWHELP      ;FAILS IF NO MEM IS GIVEN
        CMPI.B #32,(A1)    ;COMPARING LAST PART TO SPACE
        BEQ CMPSORTWL2
        LSL.L #8,D4
        BRA CMPSORTWL1
CMPSORTWL2
        ADD.B (A1)+,D5      ; MOVE THE MEMORY LOCATION NUMBER TO D5
        CMPI.B #0,(A1)     ;COMPARING LAST PART TO EMPTY STRNG
        BEQ SORTWHELP      ;FAILS IF NO MEM IS GIVEN
        CMPI.B #32,(A1)    ;COMPARING LAST PART TO SPACE
        BEQ CMPSORTWL3
        LSL.L #8,D5
        BRA CMPSORTWL2
CMPSORTWL3
        ADD.B 1(A1),D6      ; MOVE THE SECOND MEMORY LOCATION AFTER SPACE TO D5
        CMP.B #'A',D6      ;COMPARING LAST PART TO a or l
        BEQ SORTWFUNCTION
        ;FUNCTION
        CMP.B #'D',D6
        BEQ SORTWFUNCTION ;FUNCTION
        ;D4 GIVES THE ADDRESS D5 GIVES THE MODE
        BRA SORTWHELP ;IF END IS NOT EQUAL FAIL AND EXIT

COMPAREMM
        SUB.L D3,A1
        CLR.L D3
        MOVE.L #MMPROMPT,A0

```

COMPAREMM2

```

ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPAREMS
CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING

```

ENDS WITH A SPACE

```

BNE COMPAREMM2
;CHECK FOR THE ARGUMENTS
CLR.L D4 ;FIRST ARG
CLR.L D5 ;SECOND ARG

```

CMPMML1

```

ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ MMHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPMML2
LSL.L #8,D4
BSR CMPMML1

```

CMPMML2

```

ADD.B 1(A1),D5 ; MOVE THE SECOND MEMORY LOCATION AFTER SPACE TO D5
CMP.B #'B',D5 ;COMPARING LAST PART TO B,W OR L
BEQ MMFUNCTION ;FUNCTION
CMP.B #'W',D5
BEQ MMFUNCTION ;FUNCTION
CMP.B #'L',D5
BEQ MMFUNCTION ;FUNCTION
;D4 GIVES THE ADDRESS D5 GIVES THE MODE
BRA MMHELP ;IF END IS NOT EQUAL FAIL AND EXIT

```

COMPAREMS ;SEND 3 ARGUMENTS, D4 IS MEMLOC, D5 IS DATA, D6 IS ASCII OR HEX

```

SUB.L D3,A1
CLR.L D3
MOVE.L #MSPROMPT,A0

```

COMPAREMS2

```

ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPAREBF
CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING

```

ENDS WITH A SPACE

```

BNE COMPAREMS2
;CHECK FOR THE ARGUMENTS
CLR.L D4 ;FIRST ARG
CLR.L D5 ;SECOND ARG
CLR.L D6 ;THIRD ARG

```

CMPMSL1

```

ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ MSHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPMSL2
LSL.L #8,D4
BRA CMPMSL1

```

CMPMSL2

```

ADD.B (A1)+,D5 ; MOVE THE MEMORY LOCATION NUMBER TO D5
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG

```

```

    BEQ MSHELP ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ CMPMSL3
    LSL.L #8,D5
    BRA CMPMSL2
CMPMSL3
    ADD.B 1(A1),D6 ; MOVE THE SECOND MEMORY LOCATION AFTER SPACE TO D5
    CMP.B #'H',D6 ;COMPARING LAST PART TO B,W OR L
    BEQ MSFUNCTION ;FUNCTION
    CMP.B #'A',D6
    BEQ MSFUNCTION ;FUNCTION
    ;D4 GIVES THE ADDRESS D5 GIVES THE MODE
    BRA MSHELP ;IF END IS NOT EQUAL FAIL AND EXIT

COMPAREBF ;D4 IS BEGGINING ADDRESS, D5 IS END ADDRESS, D6 IS THE WORD DATA TO
BE STORED
    SUB.L D3,A1
    CLR.L D3
    MOVE.L #BFPROMPT,A0
COMPAREBF2
    ADD.L #1,D3
    CMPM.B (A0)+,(A1)+
    BNE COMPAREBMOV
    CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
    BNE COMPAREBF2
    ;CHECK FOR THE ARGUMENTS
    CLR.L D4 ;FIRST ARG
    CLR.L D5 ;SECOND ARG
    CLR.L D6 ;THIRD ARG
CMPBFL1
    ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BFHELP ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ CMPBFL2
    LSL.L #8,D4
    BRA CMPBFL1
CMPBFL2
    ADD.B (A1)+,D5 ; MOVE THE MEMORY LOCATION NUMBER TO D5
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BFHELP ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ CMPBFL3
    LSL.L #8,D5
    BRA CMPBFL2
CMPBFL3
    ADD.B (A1)+,D6 ; MOVE THE MEMORY LOCATION NUMBER TO D5
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BFFUNCTION ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ BFHELP
    LSL.L #8,D6
    BRA CMPBFL3

```

```

COMPAREBMV ;D4 IS THE START OF MOV, D5 IS THE END OF MOV, D6 IS THE
DESTINATION OF THE MOVE
    SUB.L D3,A1
    CLR.L D3
    MOVE.L #BMOVTPROMPT,A0
COMPAREBMV2
    ADD.L #1,D3
    CMPM.B (A0)+,(A1)+
    BNE COMPAREBTST
    CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
    BNE COMPAREBMV2
    ;CHECK FOR THE ARGUMENTS
    CLR.L D4 ;FIRST ARG
    CLR.L D5 ;SECOND ARG
    CLR.L D6 ;THIRD ARG
CMPBMV1
    ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BMOVHELP ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ CMPBMV2
    LSL.L #8,D4
    BRA CMPBMV1
CMPBMV2
    ADD.B (A1)+,D5 ; MOVE THE MEMORY LOCATION NUMBER TO D5
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BMOVHELP ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ CMPBMV3
    LSL.L #8,D5
    BRA CMPBMV2
CMPBMV3
    ADD.B (A1)+,D6 ; MOVE THE MEMORY LOCATION NUMBER TO D5
    CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
    BEQ BMOVFUNCTION
    ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
    BEQ BMOVHELP
    LSL.L #8,D6
    BRA CMPBMV3

COMPAREBTST ;D4 IS START D5 IS END
    SUB.L D3,A1
    CLR.L D3
    MOVE.L #BTSTPROMPT,A0
COMPAREBTST2
    ADD.L #1,D3
    CMPM.B (A0)+,(A1)+
    BNE COMPAREBSCH
    CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
    BNE COMPAREBTST2
    ;CHECK FOR THE ARGUMENTS
    CLR.L D4 ;FIRST ARG

```

```

CLR.L D5 ;SECOND ARG

CMPBTSTL1
ADD.B (A1)+,D4 ; MOVE THE START MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ BTSTHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPBTSTL2
LSL.L #8,D4
BRA CMPBTSTL1

CMPBTSTL2
ADD.B (A1)+,D5 ; MOVE THE MEMORY LOCATION NUMBER TO D5
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ BTSTFUNCTION ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ BTSTHELP
LSL.L #8,D5
BRA CMPBTSTL2

COMPAREBSCH ;D4 GIVES START ADDRESS, D5 GIVES END ADDRESS, D6 GIVES THE WORD
BEING SEARCHED
SUB.L D3,A1
CLR.L D3
MOVE.L #BSCHPROMPT,A0

COMPAREBSCH2
ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPAREGO
CMPI.B #32,-1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
BNE COMPAREBSCH2
;CHECK FOR THE ARGUMENTS
CLR.L D4 ;FIRST ARG
CLR.L D5 ;SECOND ARG
CLR.L D6 ;THIRD ARG

CMPBSCHL1
ADD.B (A1)+,D4 ; MOVE THE START MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ BSCHHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPBSCHL2
LSL.L #8,D4
BRA CMPBSCHL1

CMPBSCHL2
ADD.B (A1)+,D5 ; MOVE THE MEMORY LOCATION NUMBER TO D5
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ BSCHHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPBSCHL3
LSL.L #8,D5
BRA CMPBSCHL2

CMPBSCHL3
ADD.B (A1)+,D6 ; MOVE THE MEMORY LOCATION NUMBER TO D5
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ BSCHFUNCTION ;FAILS IF NO MEM IS GIVEN

```

```

    CMPI.B #32, (A1)    ;COMPARING LAST PART TO SPACE
    BEQ BSCHHELP
    LSL.L #8, D6
    BRA CMPBSCHL3

COMPAREGO ;D4 IS THE ARGUMENT TO GET TO THE MEMLOC
    SUB.L D3, A1
    CLR.L D3
    MOVE.L #GOPROMPT, A0
COMPAREGO2
    ADD.L #1, D3
    CMPM.B (A0)+, (A1)+
    BNE COMPAREDF
    CMPI.B #32, -1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
    BNE COMPAREGO2
;CHECK FOR THE ARGUMENTS
    CLR.L D4 ;FIRST ARG
    CLR.L D5 ;SECOND ARG
    CLR.L D6 ;THIRD ARG
CMPGOL1
    ADD.B (A1)+, D4    ; MOVE THE START MEMORY LOCATION NUMBER TO D4
    CMPI.B #0, (A1)    ;COMPARING LAST PART TO EMPTY STRNG
    BEQ GOFUNCTION    ;FAILS IF NO MEM IS GIVEN
    CMPI.B #32, (A1)    ;COMPARING LAST PART TO SPACE
    BEQ GOHELP
    LSL.L #8, D4
    BRA CMPGOL1

COMPAREDF
    SUB.L D3, A1
    CLR.L D3
    MOVE.L #DFPROMPT, A0
COMPAREDF2
    ADD.L #1, D3
    CMPM.B (A0)+, (A1)+
    BNE COMPARECONVERT
    CMPI.B #0, -1(A0)
    BNE COMPAREDF2
    BSR DFFUNCTION

COMPARECONVERT
    SUB.L D3, A1
    CLR.L D3
    MOVE.L #CONVERT, A0
COMPARECONVERT2
    ADD.L #1, D3
    CMPM.B (A0)+, (A1)+
    BNE COMPARECONVERT2
    CMPI.B #32, -1(A0) ;COMPARING LAST PART TO SPACE TO SEE IF THE STRING
ENDS WITH A SPACE
    BNE COMPARECONVERT2
;CHECK FOR THE ARGUMENTS
    CLR.L D4 ;FIRST ARG

```

```
CLR.L D5 ;SECOND ARG
CMPCONVERTL1
ADD.B (A1)+,D4 ; MOVE THE MEMORY LOCATION NUMBER TO D4
CMPI.B #0,(A1) ;COMPARING LAST PART TO EMPTY STRNG
BEQ CONVERTHELP ;FAILS IF NO MEM IS GIVEN
CMPI.B #32,(A1) ;COMPARING LAST PART TO SPACE
BEQ CMPCONVERTL2
LSL.L #8,D4
BSR CMPCONVERTL1
CMPCONVERTL2
ADD.B 1(A1),D5 ; MOVE THE SECOND MEMORY LOCATION AFTER SPACE TO D5
CMP.B #'A',D5 ;COMPARING LAST PART TO B,W OR L
BEQ CONVERTFUNCTION ;FUNCTION
CMP.B #'H',D5
BEQ CONVERTFUNCTION ;FUNCTION
;D4 GIVES THE ADDRESS D5 GIVES THE MODE
BRA CONVERTHELP ;IF END IS NOT EQUAL FAIL AND EXIT

COMPAREREGCLR
SUB.L D3,A1
CLR.L D3
MOVE.L #REGCLRPRMPT,A0
COMPAREREGCLR2
ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE COMPAREEXIT
CMPI.B #0,-1(A0)
BNE COMPAREREGCLR2
BSR REGCLRFUNCTION

COMPAREEXIT
SUB.L D3,A1
CLR.L D3
MOVE.L #EXITPRMPT,A0
COMPAREEXIT2
ADD.L #1,D3
CMPM.B (A0)+,(A1)+
BNE FAIL
CMPI.B #0,-1(A0)
BNE COMPAREEXIT2
BSR EXITFUNCTION
```


2.2-) *Debugger Commands*

Each debugger command is invoked by the Command Interpreter. The only exception to this rule is the DF instruction which is also invoked after an exception occurs.

Whether it is shown on the flowcharts or not all of the function except REGCLR for reasons discussed under REGCLR implementation section restores all A and D registers from stack when done. These registers are stored in the first line of the Command Interpreter thus whenever a function is finished these registers are restored to make sure no registers are altered.

All function that use arguments get their arguments through the D registers. The specific of these registers will be discussed further in the individual function description. At the end of all function the control is given back to the first line of the Command Interpreter.

2.2.1-) *Help*

Help function invoked by writing HELP in all caps, prints out a quick description of all available commands and functional descriptions. The information for the HELP command is stored in the beginning of the command as a long passage of formatted text for command line optimization terminated by a null string. What HELP function does is print this text out and stop when the null terminator is reached. The TRAP commands are used to achieve this.

2.2.1.1-) *Help Flowchart*

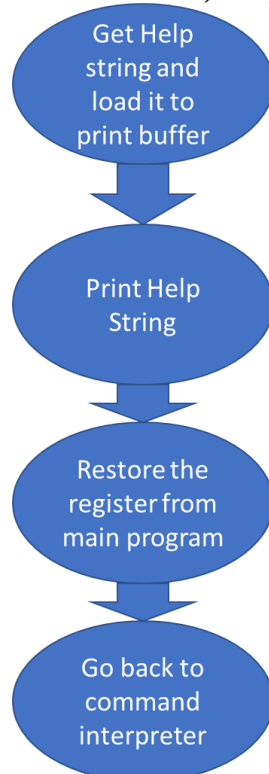


Figure 3: Help flowchart

2.2.1.2-) Help Assembly Code

The assembly code should be written using the algorithm above.

```
HELPFUNCTION ;prints out the help statement stored in memory
MOVEA.L #HELP,A1
MOVE.B #13,D0
TRAP #15
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
```

2.2.2-) Memory Display

Memory Display function is invoked by MDSP command. It has two different ways of reacting to user input depending on the number of arguments it is provided with. The distinction between which version to call is handled by the command interpreter.

The main functionality of this function does not change. MDSP displays memory address and contents of those addresses in a given range.

Case I:

Case I is when no end address is given. Start address is passed through the Command Interpreter by D4. In this case, the function prints the memory address starting from D4 ending at D4+\$F. MDSP first converts the ASCII inputted start address from ASCII to hex using helper ASCII to hex function. Using this starting address MDSP calculates the end address and stores it in D5. MDSP then loops between these two addresses printing each memory location until the endpoint, D5 is reached. A semicolon is printed between the address and input to distinguish these two fields from each other. When HEX memory addresses are read, these are converted to ASCII using helper function and printed one by one using traps.

Case II:

Case II work in the exact same way as Case I. The only difference is that endpoint is not calculated and stored in D5, rather D5 is passed down to the function as an argument that signals where the function should stop. Every other aspect of Case II is identical to Case I.

Once the function is done it restores the registers and returns control back to the Command Interpreter.

2.2.2.1-) Memory Display and flowchart
Case I

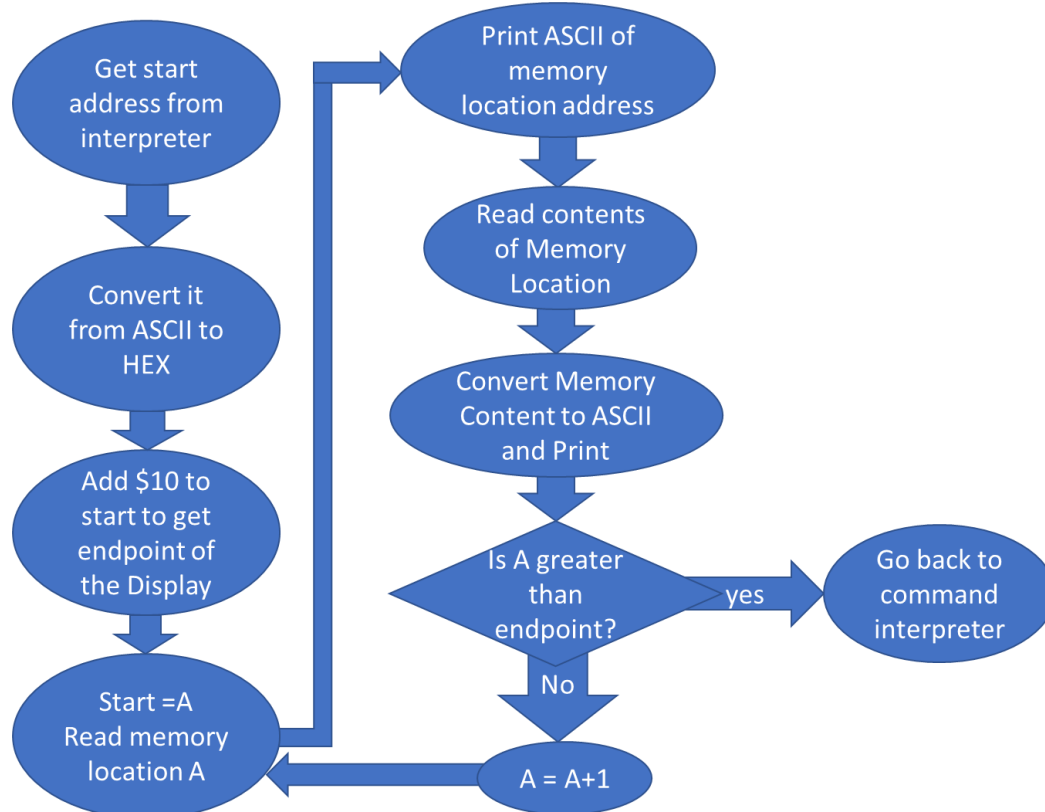


Figure 4: MDSP where only one argument is given

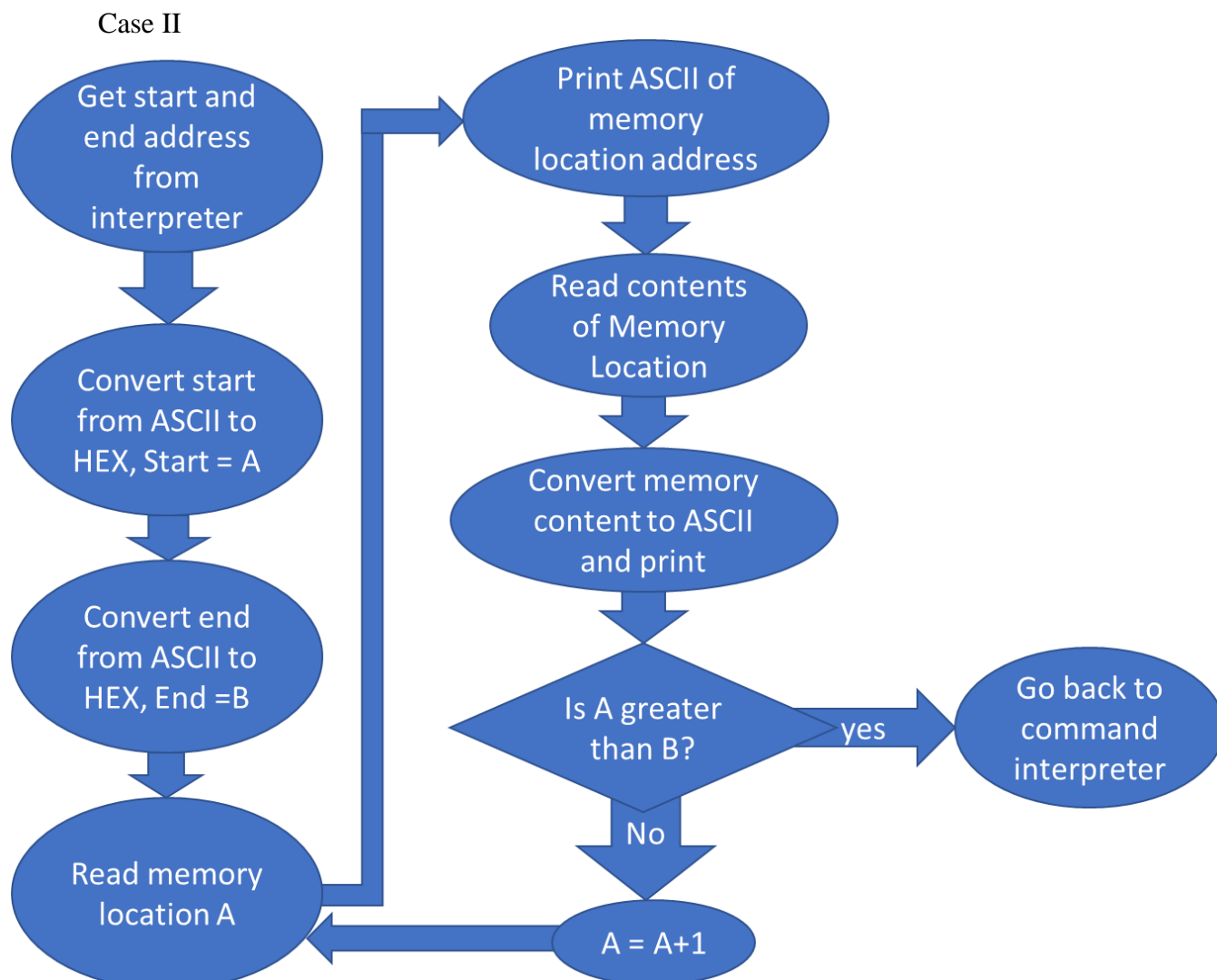


Figure 5: MDSP where both end and start is given

2.2.2.2-) Memory Display Assembly Code For Case I and Case II

```

MDSPFUNCTION1 ;argument passed as D4, PRINT FROM D4 TO D5, first case where
no end is given, display location from D4 to D4+$10 with memory contents
;MOVE.L #$2000,D4
BSR ASCIITOHX
;DISPLAY EVERYTHIGN FROM D4 FOR NOW 2000 TO 2016\
MOVE.L D4,A3
ADD.L #$10,A3 ;ENDING
MOVE.L D4,A2 ;BEGGINING
MDSPFUNCTION1L1
;PRINT MEMLOC
MOVE.L A2,D1
BSR HEXTOASCII
;bit manipulation that helps with printing memory location by byte
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 #SEMICOLONSEP,A1
MOVE.B #14,D0
TRAP #15
```

```
;PRINT CONTENT
CLR.L D1
MOVE.B (A2)+,D1
BSR HEXTOASCII
```

```
;bit manipulation that helps with printing memory location by byte
ROR #8,D1
MOVE.B #6,D0
TRAP #15
```

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

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

```
CMPA.L A2,A3
BGT MDSPFUNCTION1L1
```

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

Case 2:

MDSPFUNCTION2 ;argument passed as D4 and D5, PRINT FROM D4 TO D5

```
BSR ASCIIIOHEX
;DISPLAY EVERYTHIGN FROM D4 TO D5\
MOVE.L D4,A2 ;BEGGINING
MOVE.L D5,D4
BSR ASCIIIOHEX
MOVE.L D4,A3 ;ENDING
ADD.L #1,A3
MDSPFUNCTION2L1
;PRINT MEMLOC
MOVE.L A2,D1
BSR HEXTOASCII

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 #SEMICOLONSEP,A1
MOVE.B #14,D0
TRAP #15

;PRINT CONTENT
CLR.L D1
MOVE.B (A2)+,D1
BSR HEXTOASCII

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

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

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

```
CMPA.L A2,A3  
BGT MDSPFUNCTION2L1
```

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


2.2.3-) Sort

Sort is invoked by the SORTW command. It takes three arguments. Start address, end address and the operation mode. SORTW sorts the words in this memory range depending on the order. An operation mode signaled with 'A' sorts the words ascending and an operation mode signaled with 'D' sorts the items descending. As with MDSP, sort initially gets the start and end arguments and converts these to HEX. After the conversion, it uses the same logic as the MDSP and loops through the memory location until the end is reached. The major difference to MDSP is that each item is compared to the next item and swapped if the items are out of order. While swapping the current progression is nullified and the memory locations are checked again to make sure that everything is in order.

The only difference with Ascending and Descending implementation of sort is the trigger for the swap routine is reversed.

The algorithm for sorting used here is inspired taken from the algorithm in Lab 2 of ECE 441 course.

2.2.3.1-) Sort Flowchart

Sort Ascending

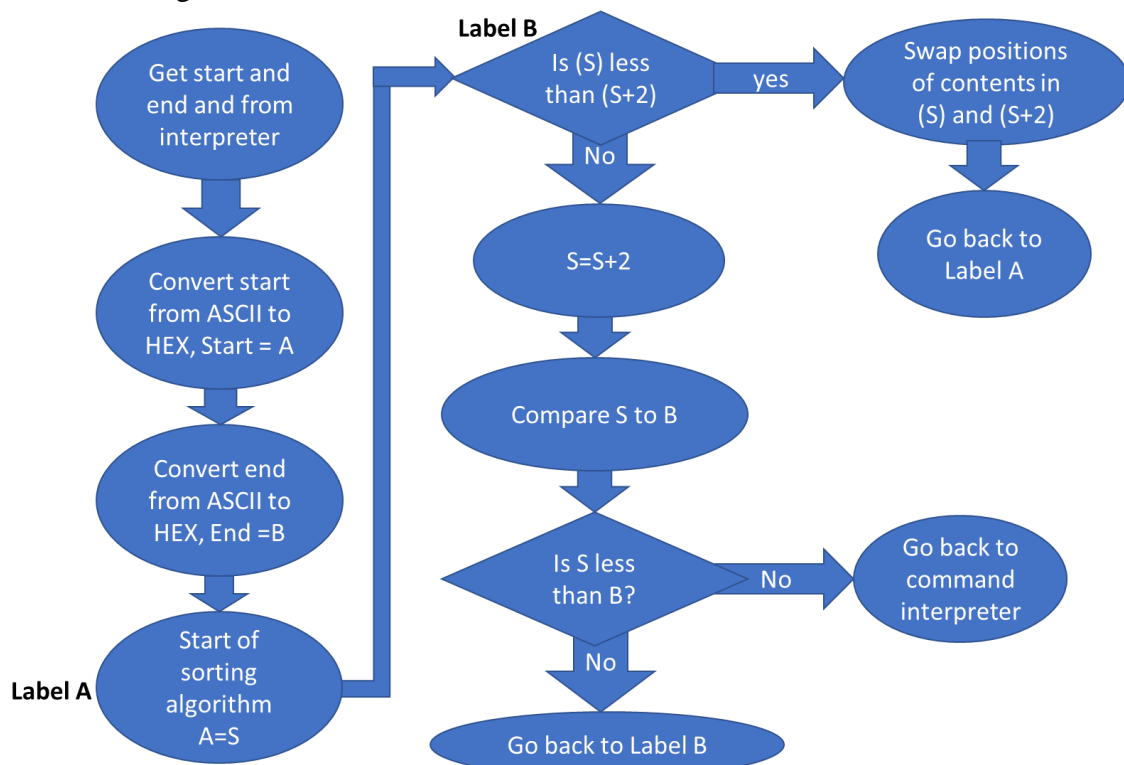


Figure 6: Flowchart for sorting ascending, very similar to descending sort

2.2.3.2-) Sort Assembly Code

SortWFunction ;D4 IS BEGGINNG D5 IS END AND D6 IS A OR D sort this range
either ascending or descending using bubble sort

```
BSR ASCIITOHX
MOVE.L D4,A0 ;BEGGINING
MOVE.L D5,D4
BSR ASCIITOHX
MOVE.L D4,A1 ;END
```

```
SUB.L #2,A1
```

```
MOVE.L A0,A2
```

```
CMP.B #'A',D6 ;If flag says ascending go to sort ascending
BEQ SORTWLA3
```

SortWl3 ;Sort Descending

```
MOVE.L A2,A0
```

SortWl2

```
CMP.W (A0)+,(A0)+
BHI.S SORTWL1 ;RECHECK
SUBQ.L #2,A0
CMP.L A0,A1
BGE SORTWL2 ;RECHECK
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
```

SortWl1

```
MOVE.L -(A0),D0
SWAP.W D0
MOVE.L D0,(A0)
BRA SORTWL3
```

SortWLA3

```
MOVE.L A2,A0
```

SortWLA2

```
CMP.W (A0)+,(A0)+
BHI.S TEMP2 ;RECHECK
BRA SORTWLA1
```

TEMP2

```
SUBQ.L #2,A0
CMP.L A0,A1
BGE SORTWLA2 ;RECHECK
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
```

SortWLA1

```
MOVE.L -(A0),D0
SWAP.W D0
MOVE.L D0,(A0)
BRA SORTWLA3
```

2.2.4-) Memory Modify

Memory modify takes in two arguments, beginning address and an operation mode. Possible operation modes are Word, Long Word and Byte. Memory modify displays the content of the memory location using the same logic from MDSP but it doesn't loop through. After displaying 1 address and its content, MM stops and asks user to give an input. User at this point has three choices. Either press enter leave current memory location unchanged, press dot and exit the MM loop or enter a value and proceed to the next memory location.

The value entered by user is converted from ASCII to hex and stored in the memory location signaled on the prompt. This is done through simple move operations.

The difference between different operation modes of memory modify is the amount of data displayed and expected. Long Word displays a long word data and expects a long word input. The implementation of the different version is exactly the same, only difference being the amount of information on the screen and increment in memory address by each carriage return. The only major difference is that the routine for converting the ASCII to hex is called twice for the long word operation since that operation can only take word inputs.

2.2.4.1-) Memory Modify Flowchart

Memory Modify Byte

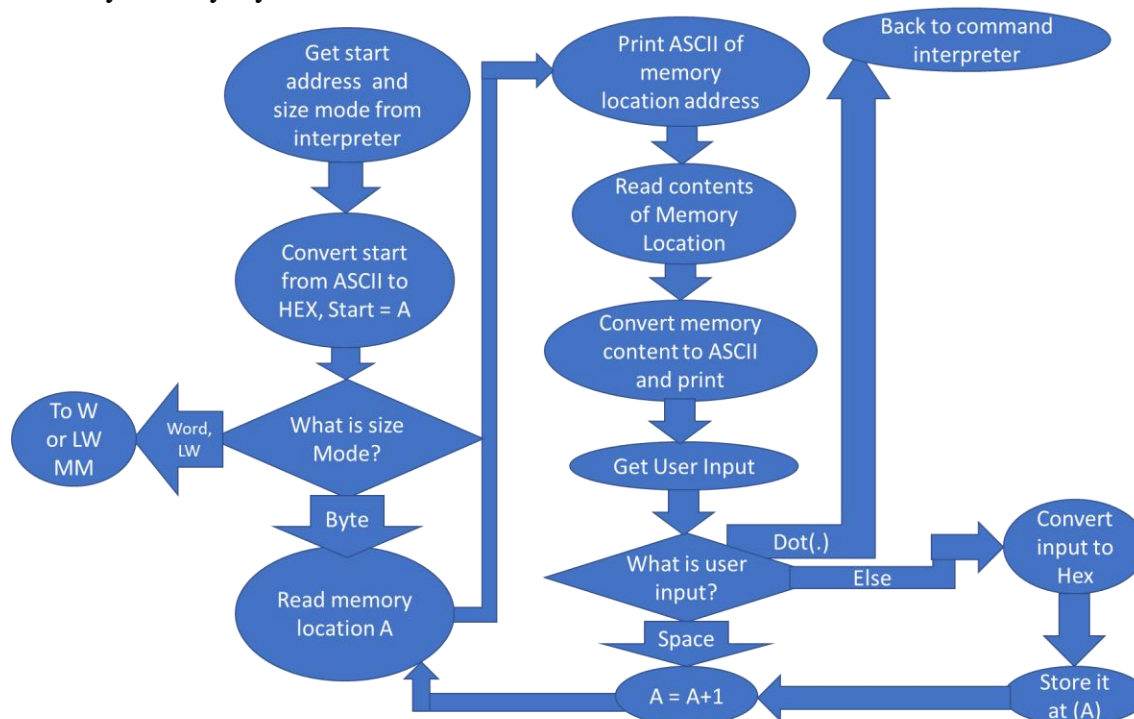


Figure 7: MM operation in Byte mode

2.2.4.2-) Memory Modify Assembly Code

```

MMFUNCTION ;argument passed as D4 AS MEMLOC AND d5 AS THE MODE
BSR ASCII2HEX
;DISPLAY EVERYTHIGN FROM D4 TO D5\
MOVE.L D4,A2 ;BEGGINING
;IGNORING B,M,L BUSSINESS FOR NOW AND ASSUMING B
CMP.B #'B',D5
BEQ MMFUNCTIONL1 ;FUNCTION
CMP.B #'W',D5
BEQ MMFUNCTIONL2 ;FUNCTION
CMP.B #'L',D5
BEQ MMFUNCTIONL3 ;FUNCTION
MMFUNCTIONL1
;PRINT MEMLOC
MOVE.L A2,D1
BSR HEX2ASCII

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 #SEMICOLONSEP,A1
MOVE.B #14,D0
TRAP #15

;PRINT CONTENT
CLR.L D1
MOVE.B (A2)+,D1
BSR HEXTOASCII

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

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

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

;PROMPT FOR INPUT,CONTINUE UNLESS IT IS DOT
LEA $5000,A1
MOVE.B #2,D0
TRAP #15

*COMPARING INPUT TO THE MENU ITEMS*
;COMPARING TO DOT
CMP.B #$2E,(A1) ;2E IS THE DOT
BEQ pSTART ;if input is dot exit
;IF INPUT IS EMPTY MOVE ON TO THE NEXT BIT
CMP.B #00,(A1)
BEQ MMFUNCTIONL1

;IF INPUT ARE NEITHER READ THE ENTERED LINE CONVERT TO HEXT AND STORE
MOVE.W (A1),D4
ADD.L #$30300000,D4
BSR ASCIITOHX

```

```
MOVE.B D4,-1(A2)
BRA MMFUNCTIONL1
```

```
MMFUNCTIONL2 ;FOR WORD OPERATION MM
```

```
;PRINT MEMLOC
MOVE.L A2,D1
BSR HEXTOASCII
```

```
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 #SEMICOLONSEP,A1
MOVE.B #14,D0
TRAP #15
```

```
;PRINT CONTENT
CLR.L D1
MOVE.W (A2)+,D1
BSR HEXTOASCII
```

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

;PROMPT FOR INPUT,CONTINUE UNLESS IT IS DOT
LEA $5000,A1
MOVE.B #2,D0
TRAP #15

*COMPARING INPUT TO THE MENU ITEMS*
;COMPARING TO DOT
CMP.B #$2E,(A1) ;2E IS THE DOT
BEQ pSTART ;if input is dot exit
;IF INPUT IS EMPTY MOVE ON TO THE NEXT BIT
CMP.B #00,(A1)
BEQ MMFUNCTIONL2

;IF INPUT ARE NEITHER READ THE ENTERED LINE CONVERT TO HEXT AND STORE
MOVE.L (A1),D4
BSR ASCIITOHX
MOVE.W D4,-2(A2)
BRA MMFUNCTIONL2
```

MMFUNCTIONL3 ;FOR LONG WORD MM OPERATION

```
;PRINT MEMLOC
MOVE.L A2,D1
BSR HEXTOASCII
```

```
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 #SEMICOLONSEP,A1
MOVE.B #14,D0
TRAP #15
```

```
;PRINT CONTENT
```

```
CLR.L D1
MOVE.W (A2)+,D1
BSR HEXTOASCII
```

```
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 SECOND WORD
;PRINT CONTENT
CLR.L D1
MOVE.W (A2)+,D1
BSR HEXTOASCII
```

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

```
;PROMPT FOR INPUT,CONTINUE UNLESS IT IS DOT
```



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

```
*COMPARING INPUT TO THE MENU ITEMS*
;COMPARING TO DOT
CMP.B #$2E,(A1) ;2E IS THE DOT
BEQ pSTART ;if input is dot exit
;IF INPUT IS EMPTY MOVE ON TO THE NEXT BIT
CMP.B #00,(A1)
BEQ MMFUNCTIONL3
```

```
;IF INPUT ARE NEITHER READ THE ENTERED LINE CONVERT TO HEXT AND STORE
MOVE.L (A1)+,D4
BSR ASCIITOHX
MOVE.W D4,-4(A2)
;PRINT SECOND WORD
MOVE.L (A1),D4
BSR ASCIITOHX
MOVE.W D4,-2(A2)

BRA MMFUNCTIONL3
```

2.2.5-) Memory Set

Memory set takes in three arguments, address, data and operation mode. Invoked by the MS command, Memory Set instruction stores the data given in the argument in the address mentioned. The different operation modes determines whether that data is converted to a hexadecimal number before being stored. Possible operation modes for MS are either 'A' for ASCII or 'H' for hexadecimal. If ASCII is selected, the input that is read as an ASCII is directly taken and stored in the memory location given by the argument. If the mode is Hex, the input is converted from ASCII to HEX using a helper function and then stored in the memory location. As always, the address provided by the Command Interpreter is converted to Hex for utilization by the helper function.

2.2.5.1-) Memory Set Flowchart

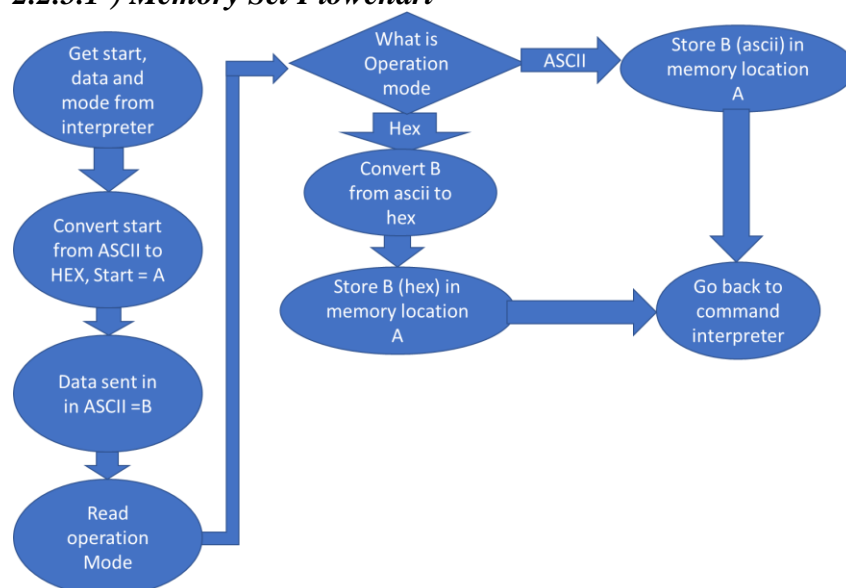


Figure 8: MS function flowchart

2.2.5.2-) Memory Set Assembly Code

MSFUNCTION ;D4 IS ADDRESS, D5 IS DATA, D6 IS ASCII OR HEX, WORD OPERATIONS ONLY, Store data in D4 depending on whether its ascii or hex

```

BSR ASCIITOHX
MOVE.L D4,A2 ;MEMLOC FOR DATA TO BE STORED
;IGNORING B,M,L BUSSINESS FOR NOW AND ASSUMING B
CMP.B #'H',D6
BEQ MSFUNCTIONL1 ;FUNCTION FOR WHEN THE NUM GIVEN IS HEX
CMP.B #'A',D6
BEQ MSFUNCTIONL2 ;FUNCTION FOR WHE THE NUM GIVEN IS ASCII

```

```

MSFUNCTIONL1
MOVE.L D5,D4
BSR ASCIITOHX

```

```

    MOVE.W D4, (A2)
    MOVEM.L (SP)+, A0-A6/D0-D7
    BRA pSTART
MSFUNCTIONL2
    MOVE.L D5, (A2)
    MOVEM.L (SP)+, A0-A6/D0-D7
    BRA pSTART

```

2.2.6-) Block Fill

Block Fill fills a range of memory with the data provided by the user. The command takes three arguments. Start address, end address and data to be used to fill this range. It uses a similar architecture to MDSP command but instead of displaying the memory location and its contents it moves the data provided by the user to that byte. Starting and end addresses must be even. The flowchart below explains the algorithm in detail.

2.2.6.1-) Block Fill Flowchart

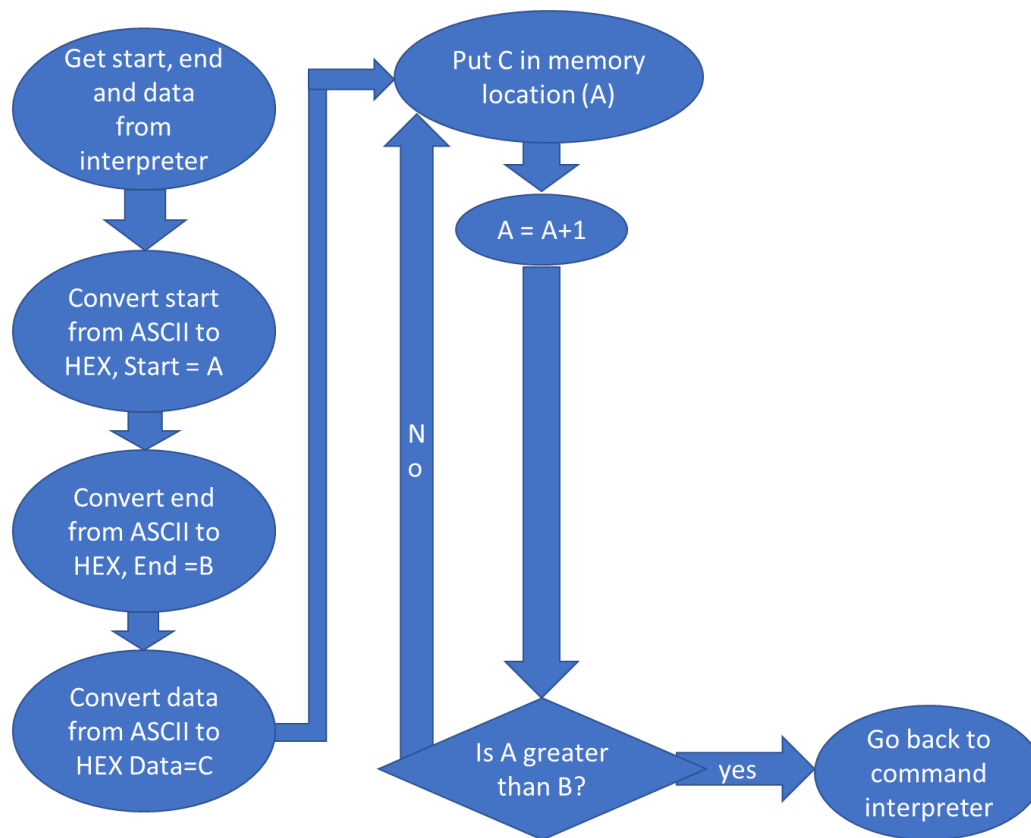


Figure 9: Block Fill flowchart

2.2.6.2-) Block Fill Assembly Code

```

BFFUNCTION ;D4 IS THE START, D5 IS THE END, D6 IS THE NUMBER TO BE WRITTEN
;FILL EVERYTHIGN FROM D4 TO D5 WITH D6
BSR ASCIITOHX
MOVE.L D4,A2 ;BEGGINING
MOVE.L D5,D4
BSR ASCIITOHX
MOVE.L D4,A3 ;ENDING
ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED
MOVE.L D6,D4
BSR ASCIITOHX
MOVE.L D4,D6 ;THE VALUE TO BE ENTERED IN THE MEMORY BLOCK
BFFUNCTIONL1
;MOVE THE WORD INTO THE ADDRESS
MOVE.W D6,(A2)+
CMPA.L A2,A3
BGT BFFUNCTIONL1
;WHEN DONE
MOVEM.L (SP)+,A0-A6/D0-D7
BRA pSTART
    
```

2.2.7-) *Block Move*

Block move is identical in architecture to block fill but instead of filling the memory location with the data given, the range is moved to another memory location. The function is given three arguments. The start address of block to be moved, the end address and the beginning of the destination. Everything from the starting address is moved to the destination until end of start address is reached.

2.2.7.1-) *Block Move Flowchart*

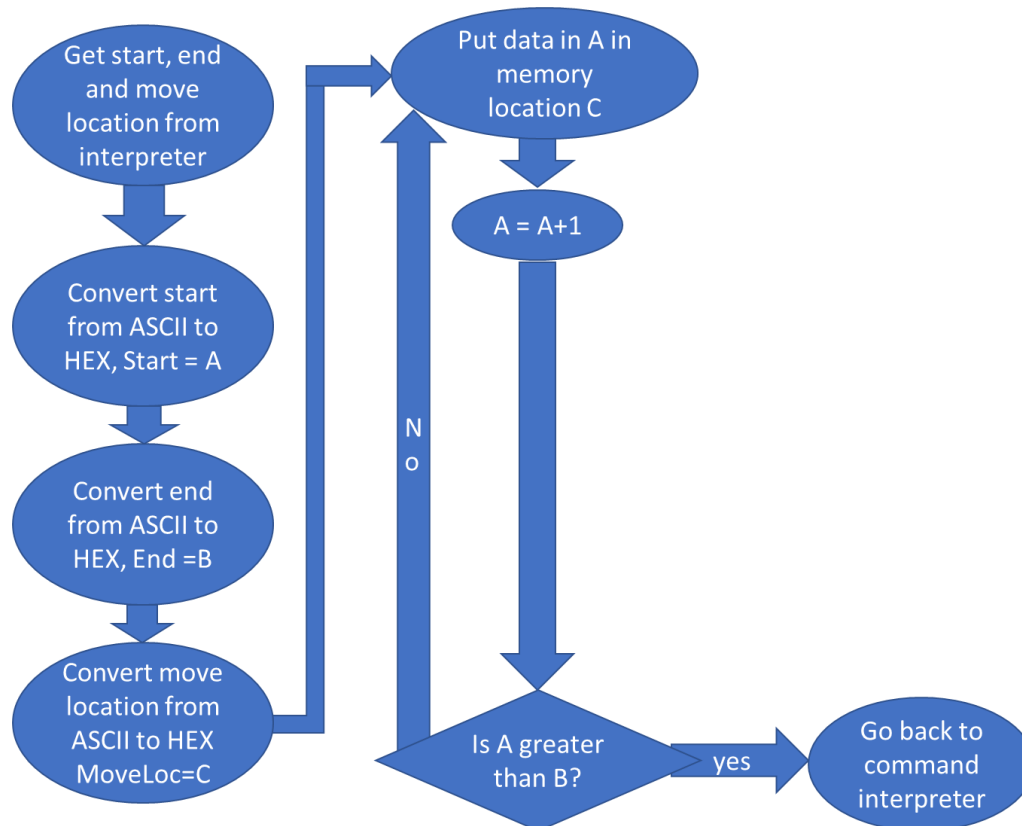


Figure 10: Block Move Flowchart

2.2.7.2-) *Block Move Assembly Code*

```

BMOVFUNCTION ;D4 IS THE START,D5 IS THE END, D6 IS THE LOCATION WE ARE
WRITING TO Moves the range from D4 to D5 to place starting in D6
    BSR ASCIITOHX
    MOVE.L D4,A2 ;BEGGINING
    MOVE.L D5,D4
    BSR ASCIITOHX
    MOVE.L D4,A3 ;ENDING
    ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED
    MOVE.L D6,D4
    BSR ASCIITOHX
    MOVE.L D4,A4 ;THE DESTINATION
  
```

```

BMOVFUNCTIONL1
;MOVE THE WORD INTO THE ADDRESS
MOVE.B (A2)+, (A4)+
CMPA.L A2,A3
BGT BMOVFUNCTIONL1
;WHEN DONE
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
    
```

2.2.8-) Block Test

Block Test is very similar to Block Fill. It fills the memory range with a hex number that alternates every other digit and after filling the block with that number, checks to see if the data written there is equal to data intended to be written there. The command is invoked with BTST text and two arguments that gives beginning and end addresses of the region to be tested. As it can be observed from the flowchart below, the architecture is very similar to other Block commands. The only difference is when there is a mismatch, an error message is displayed that includes where the error occurred, the data written and the data read.

2.2.8.1-) Block Test Flowchart

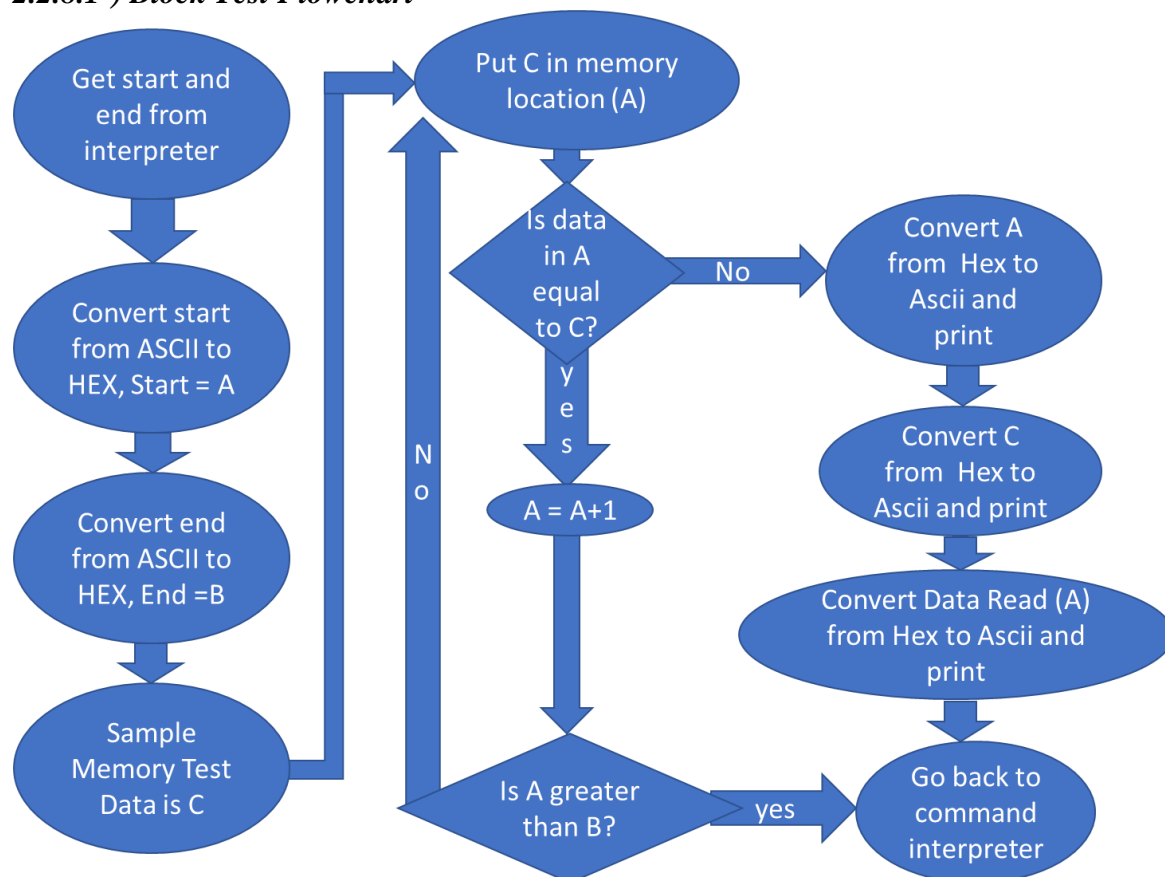


Figure 11: Block Test Flowchart

2.2.8.2-) Block Test Assembly Code

BTSTFUNCTION ;Performs destructive test on memory range from D4 to D5.

```

BSR ASCIIIOHEX
MOVE.L D4,A2 ;BEGGINING
MOVE.L D5,D4
BSR ASCIIIOHEX
MOVE.L D4,A3 ;ENDING
ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED
MOVE.L #$A5A5,D4
;DESTRUCTIVE PART SEARCH IS AT D4
    
```



```
BTSTFUNCTIONL1
    MOVE.W D4, (A2)    ;MOVE THE WORD TO START
    MOVE.W (A2)+, D5    ;READ THE WORD
    CMP.W D4, D5
    BNE BTSTERROR      ;IF NOT EQUAL GO TO SUBROUTINE
    CMPA.L A2, A3
    BGT BTSTFUNCTIONL1
    MOVEA.L #BTSTSUCCESS, A1
    MOVE.B #13, D0
    TRAP #15
    MOVEM.L (SP)+, A0-A6/D0-D7
    BRA pSTART
BTSTERROR ;Print Error Message
    MOVEA.L #BTSTFAIL, A1
    MOVE.B #13, D0
    TRAP #15

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

    SUB.L #2, A2

    MOVE.W A2, D1

    BSR HEXTOASCII

    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

    MOVEA.L #SPACE, A1
    MOVE.B #13, D0
    TRAP #15

    ;PRINT WRITTEN DATA
    MOVEA.L #DATASTORED, A1
    MOVE.B #14, D0
    TRAP #15
```

```

MOVE.W D4,D1

BSR HEXTOASCII

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

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;PRINT DATA READ
MOVEA.L #DATAREAD,A1
MOVE.B #14,D0
TRAP #15

MOVE.W D5,D1

BSR HEXTOASCII

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

```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

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

2.2.9-) Block Search

Block Search searches a range of memory for the data given. It is invoked by BSCH command and takes in three arguments, the start address, end address and the data that is being searched. It has a very similar architecture to Block Test command but instead of filling the memory range with values, it searches the region for a specific data provided by the user. If the data is found it shows where the data was found. If not, it displays an error message.

2.2.9.1-) Block Search Flowchart

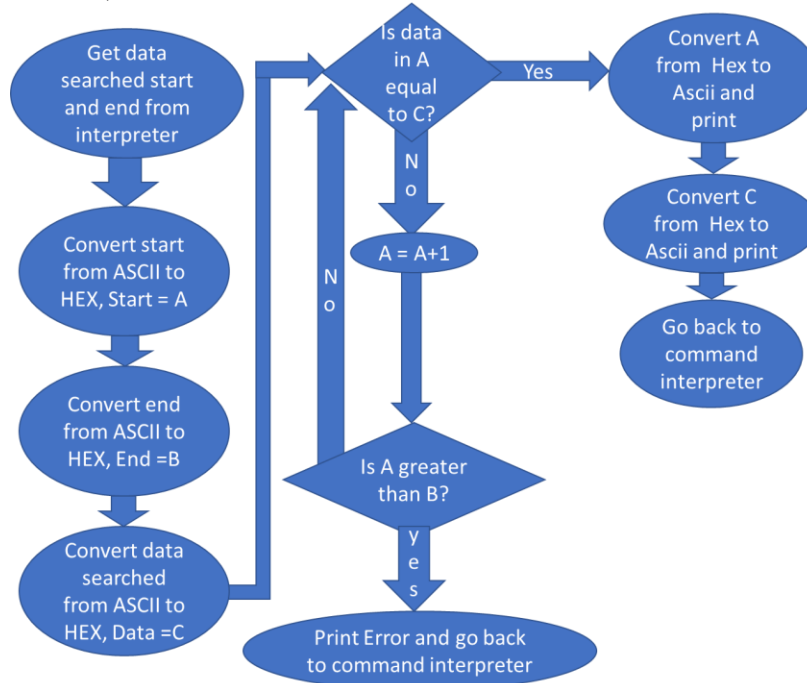


Figure 12: Block Search Flowchart

2.2.9.2-) Block Search Assembly Code

BSCHFUNCTION ;D4 IS START ADDRESS, D5 IS END ADDRESS AND D6 IS THE ITEM WE ARE LOOKING FOR, Find the Item and print Success message if found else Error Message

```

BSR ASCIITOHX
MOVE.L D4,A2 ;BEGGINING
MOVE.L D5,D4
BSR ASCIITOHX
MOVE.L D4,A3 ;ENDING
ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED
MOVE.L D6,D4
BSR ASCIITOHX
;SEARCH IS AT D4
BSCHFUNCTIONL1
;LOOK FOR WORD IN THE ADDRESS RANGE
MOVE.W (A2)+,D5
  
```

```
CMP.W D4,D5
BEQ BSCHEQUAL
CMPA.L A2,A3
BGT BSCHFUNCTIONL1
MOVEA.L #SEARCHFAIL,A1
MOVE.B #13,D0
TRAP #15
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
BSCHEQUAL
MOVEA.L #SEARCHSUCCESS,A1
MOVE.B #13,D0
TRAP #15
;PRINT ADDRESS
MOVEA.L #ADDRESS,A1
MOVE.B #14,D0
TRAP #15

SUB.L #2,A2

MOVE.W A2,D1

BSR HEXTOASCII

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

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

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

MOVE.W D4,D1

BSR HEXTOASCII
```

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

MOVEA.L #SPACE,A1
 MOVE.B #13,D0
 TRAP #15

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

2.2.10-) Execute Program

Go jumps to memory address provided by the user and starts execution there. It takes in one argument and that is the memory address that is to be jumped. The program can say RTS to return to the main program.

2.2.10.1-) Execute Program Flowchart

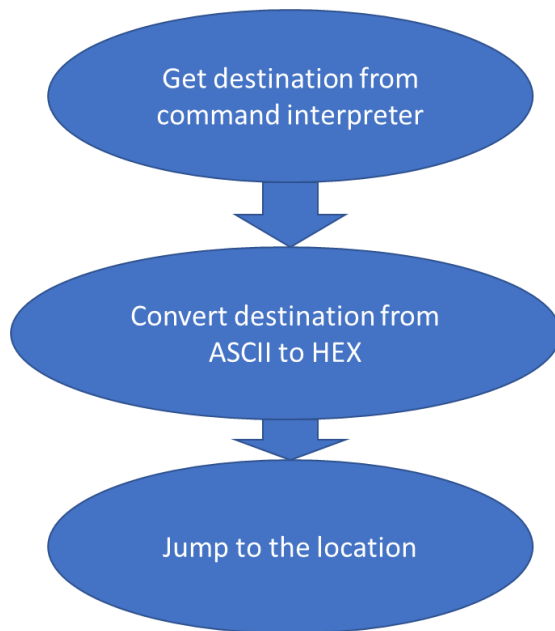


Figure 13: Go flowchart

2.2.10.2-) Execute Program Assembly Code

GOFUNCTION ;D4 IS THE MEMLOC TO EXECUTE, jumps to memory location and starts executing there

```

BSR ASCII2HEX
MOVE.L D4,A1
BSR (A1)
MOVEM.L (SP)+, A0-A6/D0-D7
BRA pSTART
    
```

2.2.11-) Display Formatted Registers

The command displays all data and address registers and Program Counter, Status Register, Supervisor Stack Pointer and User Stack Pointer. The algorithm is straight forward where all the items are displayed sequentially with the attached labels. The command is called by typing DF.

2.2.11.1-) Display Formatted Registers Algorithm and Flowchart

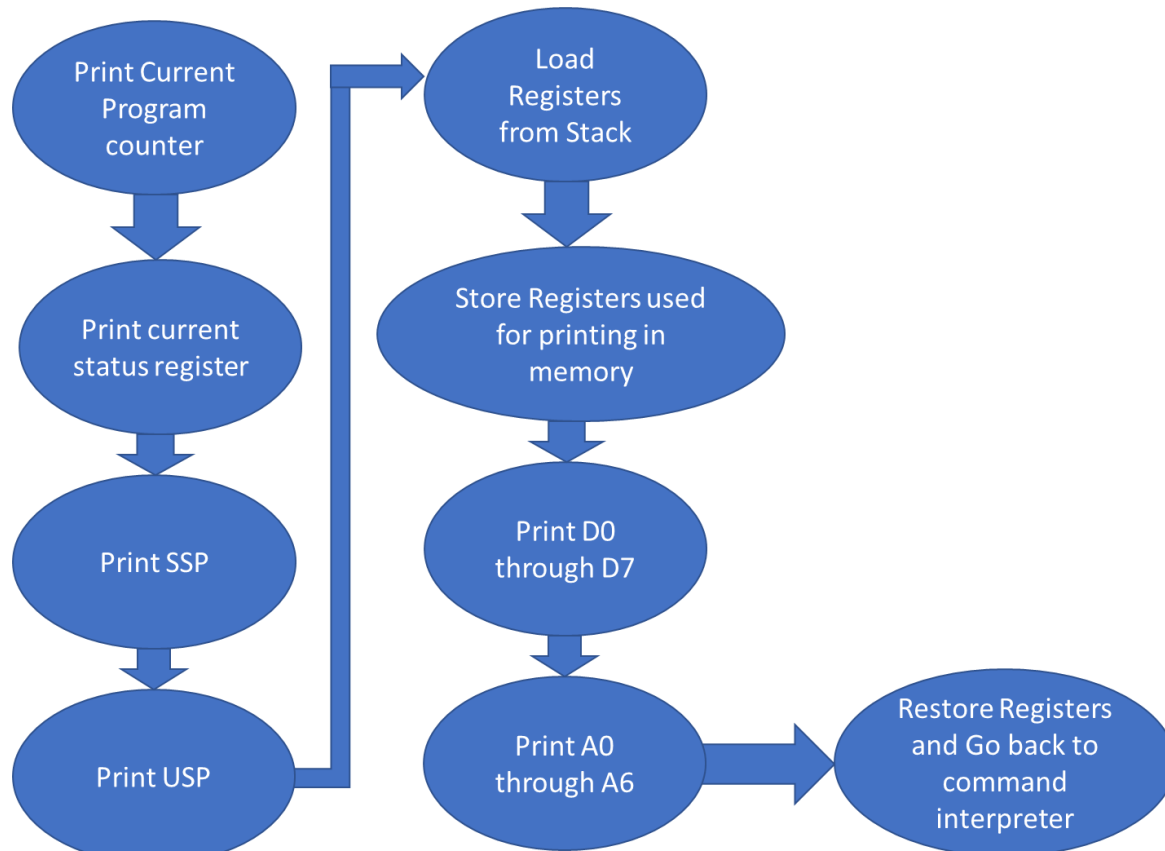


Figure 14: DF flowchart

2.2.11.2-) Display Formatted Registers Assembly Code

```

DFFUNCTION ;Displays PC, SR, SSP,USP and all data and address registers
;PRINT PC
MOVEA.L #PCDF,A1
MOVE.B #14,D0
TRAP #15
PCHERE
MOVE #PCHERE,D1
MOVE.B #16,D2
;MOVEA.L PC,A1
MOVE.B #15,D0
TRAP #15
  
```



```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;PRINT SR
MOVEA.L #SRDF,A1
MOVE.B #14,D0
TRAP #15

MOVE SR,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;PRINT SSP
MOVEA.L #SSPDF,A1
MOVE.B #14,D0
TRAP #15

MOVE A7,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;PRINT USP
MOVEA.L #USPDF,A1
MOVE.B #14,D0
TRAP #15

MOVE.L USP,A1
MOVE.L A1,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

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

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

```

;D0
MOVEA.L #D0DF,A1
MOVE.B #14,D0
TRAP #15

MOVE $8000,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;D1
MOVEA.L #D1DF,A1
MOVE.B #14,D0
TRAP #15

MOVE $8004,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;D2
MOVEA.L #D2DF,A1
MOVE.B #14,D0
TRAP #15

MOVE $8008,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15

;D3
MOVEA.L #D3DF,A1
MOVE.B #14,D0
TRAP #15

MOVE D3,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15

MOVEA.L #SPACE,A1

```

```
MOVE.B #13,D0
TRAP #15
```

```
;D4
MOVEA.L #D4DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE D4,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;D5
MOVEA.L #D5DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE D5,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;D6
MOVEA.L #D6DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE D6,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;D7
MOVEA.L #D7DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE D7,D1
MOVE.B #16,D2
MOVE.B #15,D0
```

```
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A0
```

```
MOVEA.L #A0DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE A0,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A1
```

```
MOVEA.L #A1DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE $8012,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A2
```

```
MOVEA.L #A2DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE A2,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A3
```

```
MOVEA.L #A4DF,A1
MOVE.B #14,D0
```

```
TRAP #15
```

```
MOVE A3,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A4
```

```
MOVEA.L #A4DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE A4,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A5
```

```
MOVEA.L #A5DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE A5,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```
;A6
```

```
MOVEA.L #A6DF,A1
MOVE.B #14,D0
TRAP #15
```

```
MOVE A6,D1
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
```

```
MOVEA.L #SPACE,A1
MOVE.B #13,D0
TRAP #15
```

```

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

```

```

BRA pSTART

```

2.2.12-) Exit Monitor Program

This command is called by typing EXIT. It stops the execution of the monitor.

2.2.12.1-) Exit Monitor Program Flowchart



Figure 15: Exits the Monitor program

2.2.12.2-) Exit Monitor Program Assembly Code

```

EXITFUNCTION ;exits the program
    MOVE.B #9,D0
    TRAP #15
  
```

2.2.13-) Hex to ASCII and ASCII to Hex Convertor

This function has two operating modes. Overall it takes in 2 arguments. The first one is the data being sent and second one is the operation mode.

Hex Operation Mode:

This mode inputs two ASCII characters and converts them to hex. The ASCII characters must correspond to a valid hex number. The easiest way to explain this algorithm is by providing an example. For instance, user enters words 3132. This two ASCII characters correspond to 12. 3132 is sent to the program as an ASCII inputted 33 31 33 32. This is done by the command interpreter that inputs any character that is written by the user as ASCII. 33 31 33 32 that is now stored in the register is then converted from ASCII to HEX using the custom subroutines. Now the result that is returned is 3132. These two characters are printed one by one writing 1 and 2. A trap function that prints ASCII is used for this

ASCII operation Mode:

This mode is more straightforward than the previous one. Say input is 3132 and we want to convert. This is passed down to the function as 33 31 33 32. This is then displayed as hex numbers using a Trap function so 33 31 33 32 is displayed.

2.2.13.1-) Hex to ASCII and ASCII to Hex Convertor Flowchart

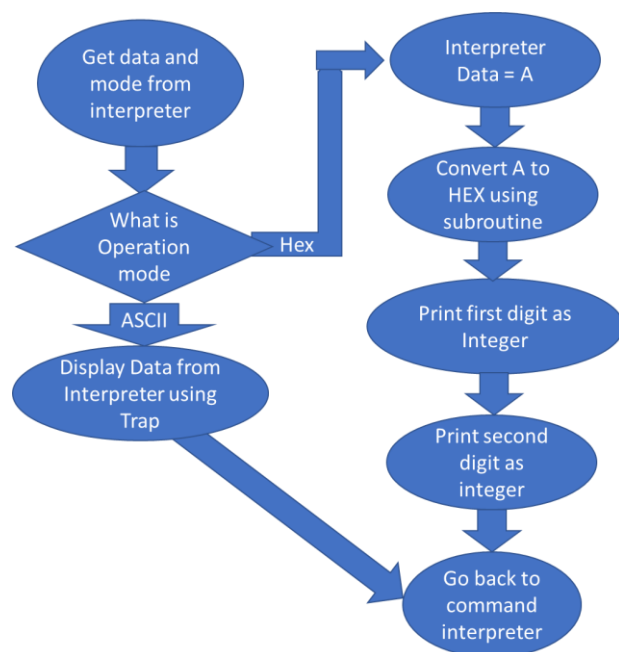


Figure 16: Converter Flowchart

2.2.13.2-) Hex to ASCII and ASCII to Hex Convertor Assembly Code

CONVERTFUNCTION: ;D4 IS THE DATA IN ASCII, D5 IS THE MODE, IF A, CONVERT IT TO ASCII AND STORE IN 8000, IF H CONVERT THE NUMBER TO HEX AND STORE IT

CMP.B #'H',D5


```
BEQ CONVERTFUNCTIONL1 ;CONVERT TO HEX AND DISPLAY
CMP.B #'A',D5
BEQ CONVERTFUNCTIONL2 ;CONVERT TO ASCII AND DISPLAY

CONVERTFUNCTIONL1 ;CONVERTS ASCII TO HEX AND DISPLAYS
BSR ASCIIIOHEX

MOVE.L D4,D1 ; ALREADY HAVE IT IN ASCII

;PRINT NUMBER 1 BY 1
ROL #8,D1
MOVE.B #6,D0
TRAP #15

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

MOVEA.L #SPACE ,A1
MOVE.B #13,D0
TRAP #15
;Restore register and go back
MOVEM.L (SP)+,A0-A6/D0-D7
BRA pSTART

CONVERTFUNCTIONL2 ;CONVERTS HEX TO ASCII AND DISPLAYS
;STORE IT IN A REGISTER
MOVE.L D4,D1
;DISPLAY AS IS
MOVE.B #16,D2
MOVE.B #15,D0
TRAP #15
;PRINT SPACE FOR NEXT LINE
MOVEA.L #SPACE ,A1
MOVE.B #13,D0
TRAP #15

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

2.2.14-) Clear Registers

This function when called clears all the data and address registers. This is the only function that does not restore the registers used after finishing execution since the aim is to have the changes applied to the registers permanent.

2.2.14.1-) Clear Registers Flowchart

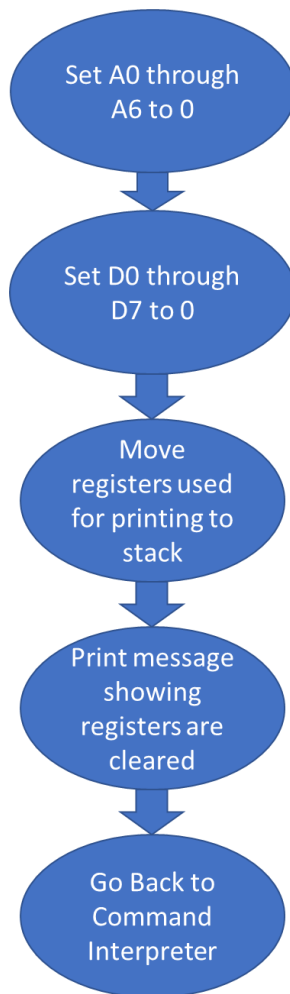


Figure 17: Clear Registers flowchart

2.2.14.2-) Clear Registers Assembly Code

```
REGCLRFUNCTION: ;CLEARS A and D REGISTERS
;CLEAR A REGISTERS
MOVE.L #0,A0
MOVE.L #0,A1
MOVE.L #0,A2
MOVE.L #0,A3
MOVE.L #0,A4
MOVE.L #0,A5
MOVE.L #0,A6
```

```

;CLEAR D REGISTERS
MOVE.L #0,D1
MOVE.L #0,D2
MOVE.L #0,D3
MOVE.L #0,D4
MOVE.L #0,D5
MOVE.L #0,D6
MOVE.L #0,D7
;DISPLAY REGISTERS CLEARED MESSAGE
MOVEM.L A1/D0,-(SP)
MOVEA.L #REGCLRPRMPT,A1
MOVE.B #13,D0
TRAP #15
MOVEM.L (SP)+,A1/D0
BRA pSTART
    
```

2.3-) *Exception Handlers*

Default exception handles are replaced in the beginning of the monitor program by changing the go to values on the exception vector table. Descriptions for each exception handling routine is described below.

2.3.1-) *Bus Error Exception*

Initially Bus Error prints out a text saying that the current error is a bus error. Bus error simply prints out the Bus Address, Instruction Register, Supervisor Status Word collecting them from stack. The function then calls DF, where values of all registers are displayed and eventually control is given back to the monitor program.

2.3.1.1-) *Bus Error Flowchart*

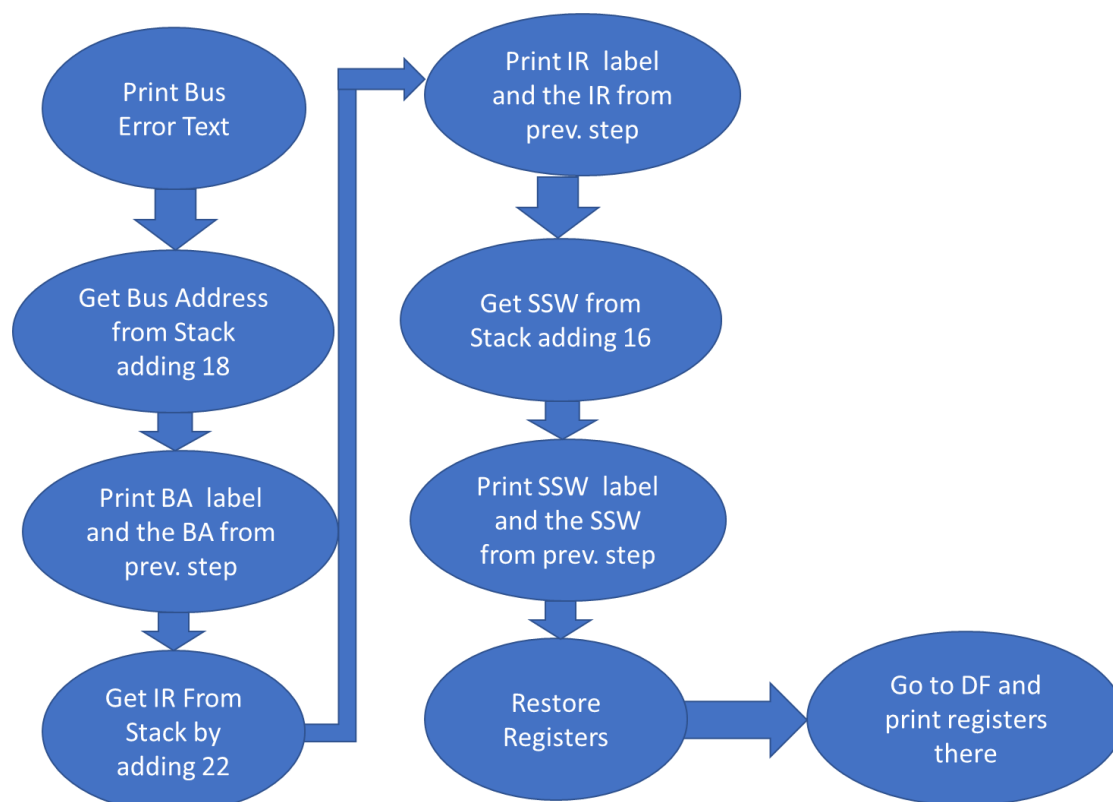


Figure 18: Bus Error Flowchart

2.3.1.2-) *Bus Error Exception Assembly Code*

BUSERORFUNCTION

```

;PRINT OUT BERR STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA BERRTEXT,A1
MOVE.L #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

;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 DFFUNCTION

```

2.3.2-) Address Error Exception

The program is exactly same as the bus error, the only difference is that the text displayed initially says the error is an address error.

2.3.1.1-) Address Error Exception Flowchart

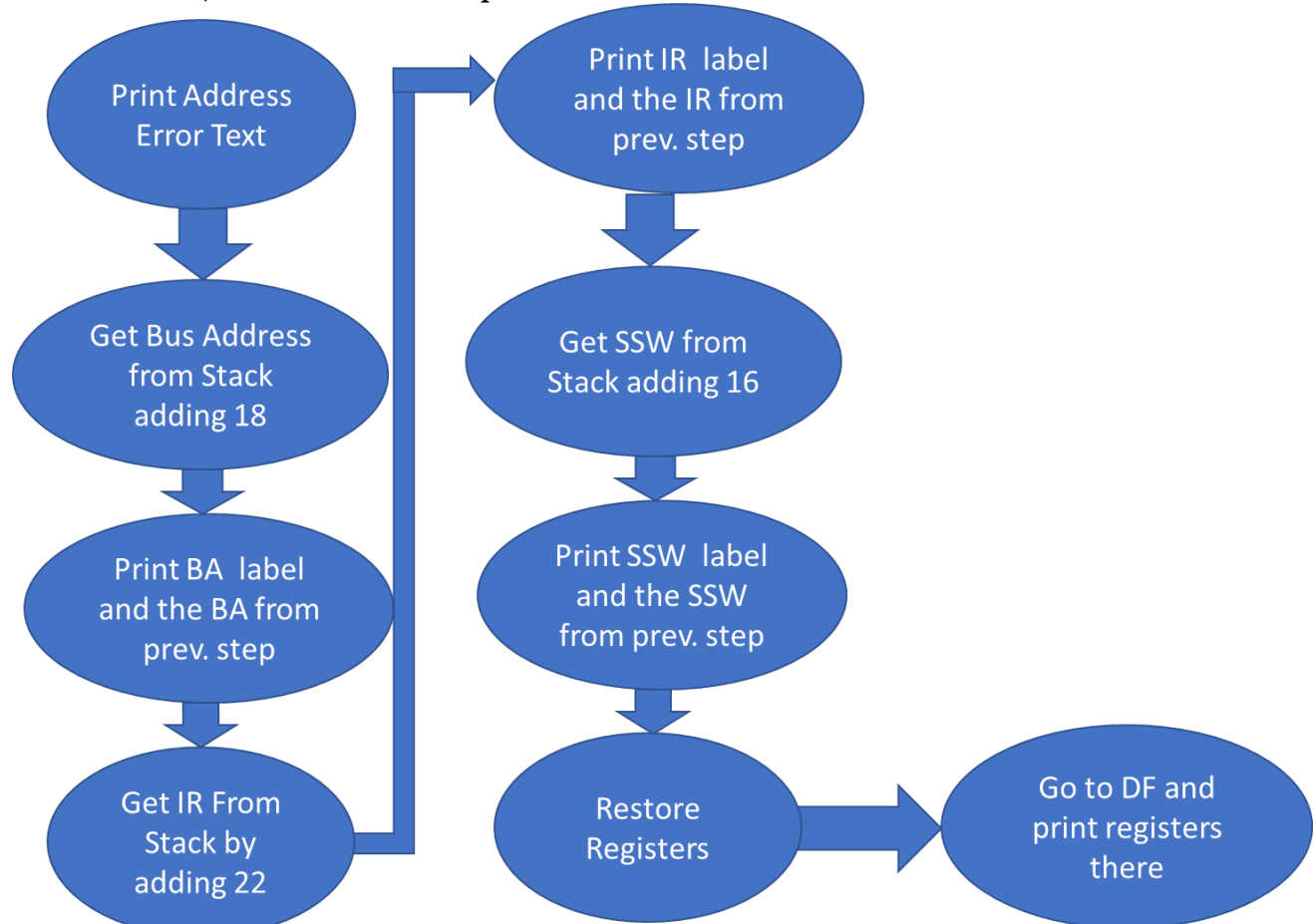


Figure 19: Address Error Flowchart

It is similar to 2.3.1.1

2.3.1.2-) Bus Error Exception Assembly Code

```

ADDRESSERRORFUNCTION
;PRINT OUT BERR STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA ADDRESSERRORTEXT,A1
MOVE.L #13,D0
TRAP #15

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

```

TRAP #15

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

LEA SPACE,A1
MOVE.B #13,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 SPACE,A1
MOVE.B #13,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 SPACE,A1
MOVE.B #13,D0
TRAP #15

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

BRA DFFUNCTION

```

2.3.3-) *Illegal Instruction Exception*

This exception routine prints out the custom error message for the exception and displays the bus address and displays all data and address registers. The control is eventually given back to monitor program.

2.3.3.1-) *Illegal Instruction Exception Algorithm and Flowchart*

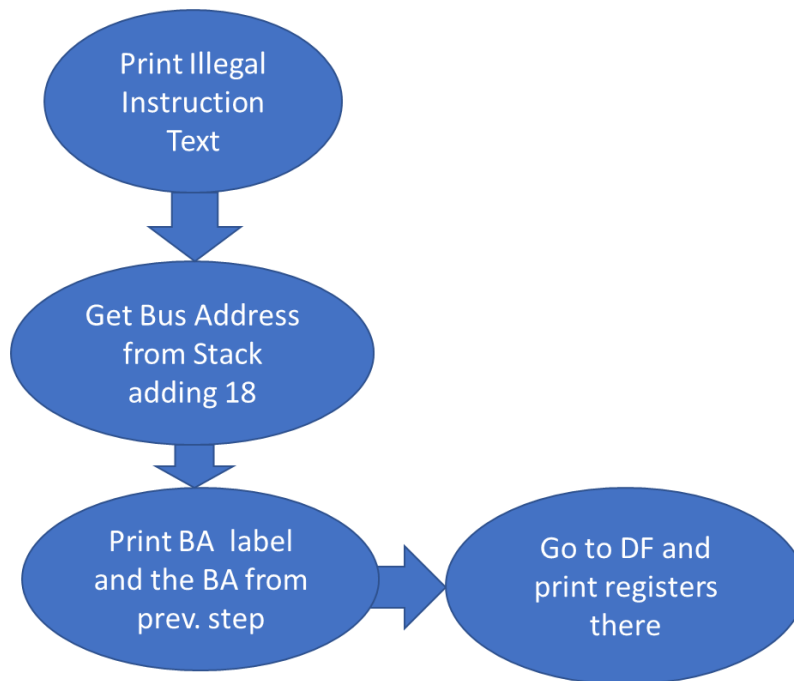


Figure 20: Illegal Instruction Flowchart

2.3.3.2-) *Illegal Instruction Exception Assembly Code*

```

ILLEGALINSTRUCTIONFUNCTION
;PRINT OUT ILLEGAL INSTR STRING
MOVEM.L A0-A6/D0-D7, -(SP)
LEA ILLEGALVECTORTTEXT, A1
MOVE.L #13, D0
TRAP #15
;PRINT BA
LEA BUSADDRESS, A1
MOVE.B #14, D0
TRAP #15

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

LEA SPACE, A1
    
```



```

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

BRA DFFUNCTION
    
```

2.3.4-) *Privilege Violation Exception*

This exception routine prints out the custom error message for the exception and displays the bus address and displays all data and address registers. The control is eventually given back to monitor program.

2.3.4.1-) *Privilege Violation Exception Flowchart*

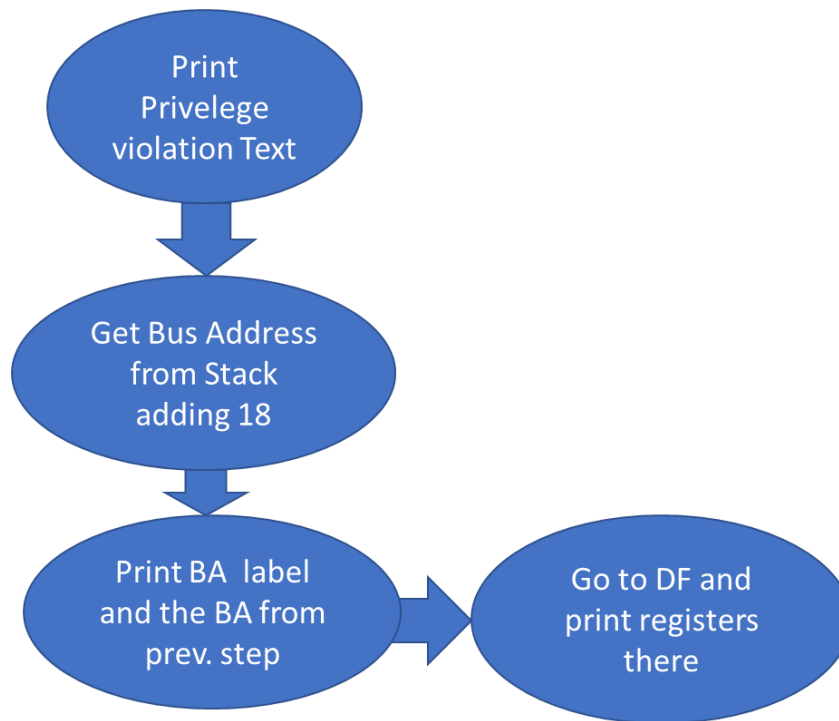


Figure 21: Privilege Violation Flowchart

2.3.4.2-) *Privilege Violation Exception Assembly Code*

```

PRIVELEGEVIOLATIONFUNCTION
;PRINT OUT PRIV VIOL STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA PRIVELEGEVECTORTTEXT,A1
MOVE.L #13,D0
TRAP #15
;PRINT BA
LEA BUSADDRESS,A1
MOVE.B #14,D0
TRAP #15

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

LEA SPACE,A1
MOVE.B #13,D0
    
```

```

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

BRA DFFUNCTION
    
```

2.3.5-) Divide by Zero Exception

This exception routine prints out the custom error message for the exception and displays the bus address and displays all data and address registers. The control is eventually given back to monitor program.

2.3.5.1-) Divide by Zero Exception Flowchart

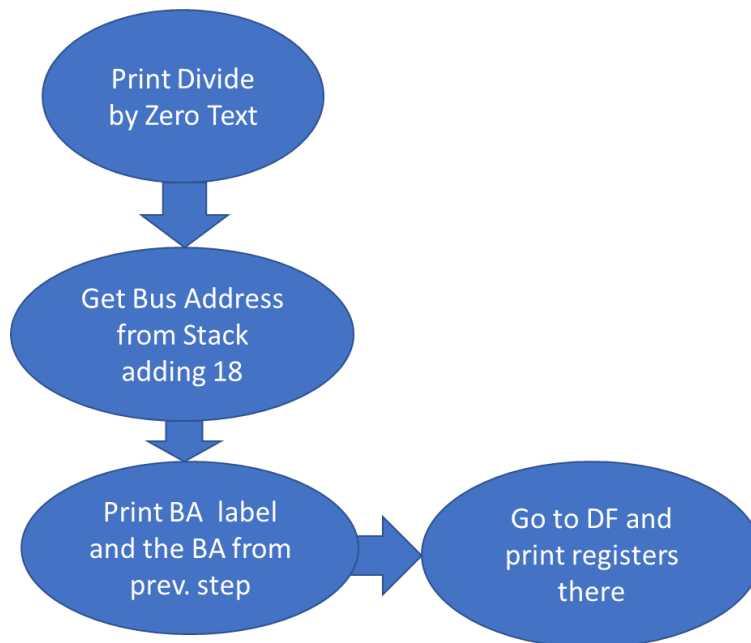


Figure 22: Divide by Zero flowchart

2.3.5.2-) Divide by Zero Exception Assembly Code

```

DIVIDEBYZEROFUNCTION
;PRINT OUT DIV BY ZERO STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA DIVIDEZEROTEXT,A1
MOVE.L #13,D0
TRAP #15
;PRINT BA
LEA BUSADDRESS,A1
MOVE.B #14,D0
TRAP #15

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

LEA SPACE,A1
MOVE.B #13,D0
TRAP #15
  
```

```

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

BRA DFFUNCTION
    
```

2.3.6-) Check Instruction Exception

This exception routine prints out the custom error message for the exception and displays the bus address and displays all data and address registers. The control is eventually given back to monitor program.

2.3.6.1-) Check Instruction Exception Flowchart

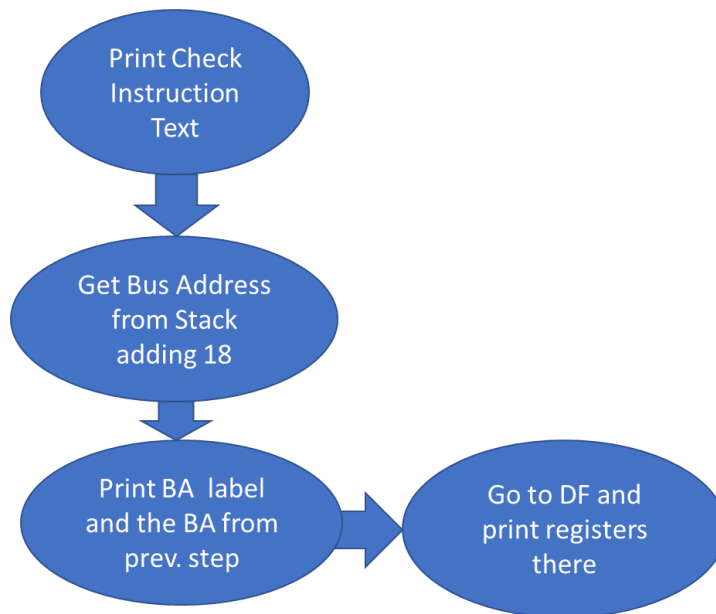


Figure 23: Check Instruction Flowchart

2.3.6.2-) Check Instruction Exception Assembly Code

```

CHECKINSTRUCTIONFUNCTION
;PRINT OUT CHECK STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA CHECKVECTORTTEXT,A1
MOVE.L #13,D0
TRAP #15
;PRINT BA
LEA BUSADDRESS,A1
MOVE.B #14,D0
TRAP #15

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

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

```
LEA SPACE,A1  
MOVE.B #13,D0  
TRAP #15
```

```
BRA DFFUNCTION
```

2.3.7-) *Line A and Line F Emulators*

This exception routine prints out the custom error message for the exception and displays the bus address and displays all data and address registers. The control is eventually given back to monitor program. Line A and Line F emulator exceptions are exactly the same the only difference is the error message printed.

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

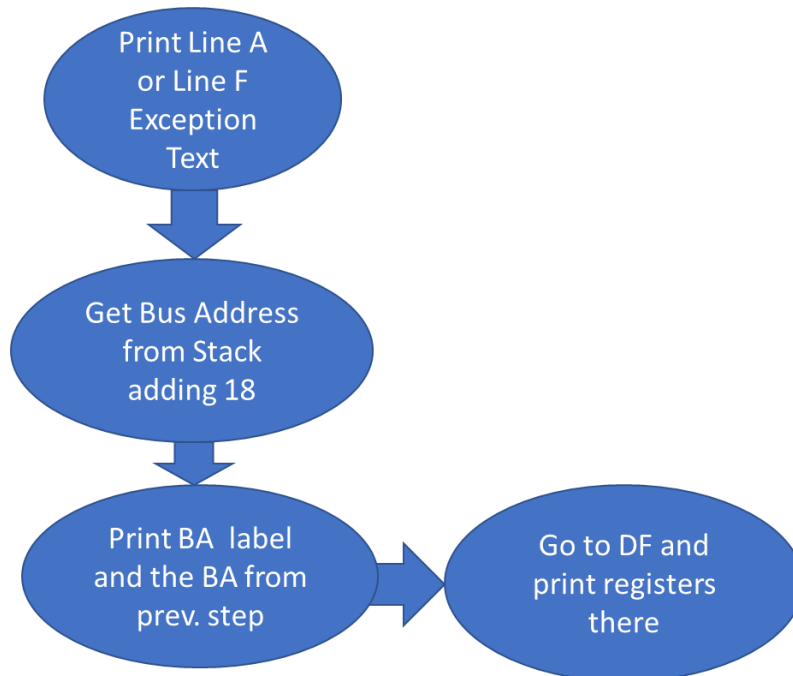


Figure 24: Line A or Line F Emulator Exception

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

```

LINEAEMULATORFUNCTION
;PRINT OUT LINEA STRING
MOVEM.L A0-A6/D0-D7,-(SP)
LEA LINEAEMULATORTEXT,A1
MOVE.L #13,D0
TRAP #15
;PRINT BA
LEA BUSADDRESS,A1
MOVE.B #14,D0
TRAP #15

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



```

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

    BRA DFFUNCTION

LINEFEMULATORFUNCTION
    ;PRINT OUT LINEF STRING
    MOVEM.L A0-A6/D0-D7,-(SP)
    LEA LINEFEMULATORTEXT,A1
    MOVE.L #13,D0
    TRAP #15
    ;PRINT BA
    LEA BUSADDRESS,A1
    MOVE.B #14,D0
    TRAP #15

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

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

    BRA DFFUNCTION

```

2.4-) Extra Helper Functions

2.4.1-) ASCII to Hex Convertor

Description of function: Converts the hex numbers to ascii. Restores registers when done. Mainly utilizes bit manipulation to achieve this.

Assembly Code:

HEXTOASCII

```
*-----
* Title       : HEX to ASCII CONVERTER
* Written by  : METE MORRIS
* Date       :
* Description: CONVERTS HEX WORD READ TO ASCII, REGISTERS MODIFIED ARE
D2,D3,D4,D5 RETURNS D1
*-----
```

```
MOVEM.L D2-D5, -(SP)
```

```
CLR.L D2
CLR.L D3
CLR.L D4
CLR.L D5
```

```
;FIRST NO
MOVE.B D1,D2
BCLR #4,D2
BCLR #5,D2
BCLR #6,D2
BCLR #7,D2
```

```
;SECOND NO
LSR #4,D1
MOVE.B D1,D3
BCLR #4,D3
BCLR #5,D3
BCLR #6,D3
BCLR #7,D3
```

```
;THIRD NO
LSR #4,D1
MOVE.B D1,D4
BCLR #4,D4
BCLR #5,D4
BCLR #6,D4
BCLR #7,D4
```

```
;FOURTH NO
LSR #4,D1
MOVE.B D1,D5
BCLR #4,D5
BCLR #5,D5
BCLR #6,D5
BCLR #7,D5
```

```

;CONVERT FIRST NO TO ASCII
CMP.B #9,D2
BGT LARGER1
ADD.L #$30,D2
BRA SECOND
LARGER1
ADD.L #$37,D2

SECOND
;CONVERT SECOND NO TO ASCII
CMP.B #9,D3
BGT LARGER2
ADD.L #$30,D3
BRA THIRD
LARGER2
ADD.L #$37,D3

THIRD
;CONVERT THIRD NO TO ASCII
CMP.B #9,D4
BGT LARGER3
ADD.L #$30,D4
BRA FOURTH
LARGER3
ADD.L #$37,D4

FOURTH
;CONVERT FOURTH NO TO ASCII
CMP.B #9,D5
BGT LARGER4
ADD.L #$30,D5
BRA ADDFINISH
LARGER4
ADD.L #$37,D5

;ROTATE AND ADD THE NUMBERS
ADDFINISH
CLR.L D1
ADD.L D2,D1

LSL #8,D3
ADD.L D3,D1

SWAP.W D4
ADD.L D4,D1

LSL #8,D5
SWAP.W D5
ADD.L D5,D1

MOVEM.L (SP)+,D2-D5

RTS

```

2.4.2-) Hex to ASCII Convertor

Description of function: Converts ASCII number to HEX. The ASCII has to be a valid hex digit otherwise error message is displayed.

Assembly Code:

ASCIITOHX

```
*-----
* Title       : ASCII to HEX CONVERTER
* Written by  : METE MORRIS
* Date       :
* Description: CONVERTS ASCII LWORD READ TO HEX, REGISTERS MODIFIED ARE
D3,D5,D6,D7 RETURNS RESUT IN D4
*-----
```

```
MOVEM.L D0/D3/D5-D7/A1,-(SP)
CLR.L D3
CLR.L D5
CLR.L D6
CLR.L D7
```

FIRSTBYTE

```
MOVE.B D4,D3
CMP #$41,D3
BGE ALP1
CMP #$2F,D3
BLE ASCIIFAIL
CMP #$3A,D3
BGE ASCIIFAIL
SUB.B #$30,D3
BRA SECONDBYTE
```

ALP1

```
CMP #$47,D3
BGE ASCIIFAIL
SUB.B #$37,D3
```

SECONDBYTE

```
LSR #8,D4
MOVE.B D4,D5
CMP #$41,D5
BGE ALP2
CMP #$2F,D5
BLE ASCIIFAIL
CMP #$3A,D5
BGE ASCIIFAIL
SUB.B #$30,D5
BRA THIRDBYTE
```

ALP2

```
CMP #$47,D5
BGE ASCIIFAIL
SUB.B #$37,D5
```

THIRDBYTE

```
SWAP D4
MOVE.B D4,D6
CMP #$41,D6
BGE ALP3
```

```

CMP #$2F,D6
BLE ASCIIIFAIL
CMP #$3A,D6
BGE ASCIIIFAIL
SUB.B #$30,D6
BRA FOURTHBYTE

```

ALP3

```

CMP #$47,D6
BGE ASCIIIFAIL
SUB.B #$37,D6

```

FOURTHBYTE

```

LSR #8,D4
MOVE.B D4,D7
CMP #$41,D7
BGE ALP4
CMP #$2F,D7
BLE ASCIIIFAIL
CMP #$3A,D7
BGE ASCIIIFAIL
SUB.B #$30,D7
BRA FINISH2

```

ALP4

```

CMP #$47,D7
BGE ASCIIIFAIL
SUB.B #$37,D7

```

;ROTATE AND ADD THE NUMBERS

FINISH2

```

CLR.L D4
ADD.L D3,D4

LSL #4,D5
ADD.L D5,D4

LSL #8,D6
ADD.L D6,D4

LSL #8,D7
LSL #4,D7
ADD.L D7,D4

MOVEM.L (SP)+,D0/D3/D5-D7/A1

```

RTS

ASCIIIFAIL

```

MOVE.L #ASCIIICHECK,A1
MOVE.L #13,D0
TRAP #15
MOVEM.L (SP)+,D0/D3/D5-D7/A1
BRA pSTART

```

3-) *Quick User Manual*

- 1) **HELP:** Display Help all command available and usage
FORMAT: ->HELP
- 2) **MDSP:** Display Memory range given
FORMAT: MDSP <ADDR>- OR -MDSP <SADDR> <EADDR>-
- 3) **SORTW:** Sort Words in the memory range
FORMAT: ->MM <SADDR> <EADDR> X- WHERE X=A OR D, A=ASCENDING
D=DESCENDING
- 4) **MM:** Display memory and modify or enter data, used with different sizes
FORMAT: ->MM <ADDR> X- WHERE X=B,W,L
- 5) **MS:** Alters the content of the memory location according to given data
FORMAT: ->MS <ADDR> <DATA> X- WHERE X=A OR H, A=ASCII H=HEX
- 6) **BF:** Fill the memory block from start to end with the data given data
FORMAT: ->BF <SADDR> <EADDR> <DATA>
- 7) **BMOV:** Move the specified memory block from start to end to destination
FORMAT: ->BMOV <SADDR> <EADDR> <DESTADDR>
- 8) **BTST:** Destructively test the memory block, if error detected, display
FORMAT: ->BTST <SADDR> <EADDR>
- 9) **BSCH:** Search the memory block for a given data
FORMAT: ->BSCH <SADDR> <EADDR> <DATA>
- 10) **GO:** Go to memory address and start execution there
FORMAT: ->GO <ADDR>-
- 11) **DF:** Display all the registers
FORMAT: ->DF
- 12) **EXIT:** Exit the program
FORMAT: ->EXIT
- 13) **CONVERT:** Convert 2 ASCII characters to HEX or 1 Hex Word to ASCII
FORMAT: ->CV <DATA> X- WHERE X=A,H, A=ASCII H=HEX
- 14) **REGISTER CLEAN:** Set all registers to 0
FORMAT: ->REGCLR

4-) *Discussion*

Overall there were many design challenges that needed to be overcome and this project was extremely time exhausting. Initially the main skeleton which is recognizing different commands and branching to those functions were implemented. This was just recognizing different inputs and branching to a function and printing that inputs name. No arguments were taken into consideration at this stage. Developing this overall skeleton that provided branching to different functions and recognizing different input was proven to be very useful as in the later stages of the development the functions were easy to develop.

Next the work was on argument parsing. Initially parsing the arguments were really challenging. To reduce to length of code and for use in different parts of the code, the helper functions that convert hex to ASCII and ASCII to hex was implemented. There was a design choice made so that the arguments taken in from the command interpreter would be passed as ASCII and it would be the individual function responsibility to convert this to the desired form. This proved to be very useful and allowed the argument parsing part of the command interpreter to be very straightforward. The first argument parsing was implemented for MDSP command and this took quite a lot of time. After the implementation for this one was done argument parsing became very easy as all other commands used a similar structure.

The functions themselves were initially developed independently from the argument parsing part with default input and tested there. Again, MDSP was the first function to be implemented and this was quite challenging. Once getting input and printing and different bit manipulation to do those actions were performed the implementation of all the other functions became easy as bits and pieces of code was reused very frequently. All memory commands were very similar to each other and all block command were also very alike. These similarities were used and development was made fairly easy. The only functions that were much different was DF, Exit, REGCLR and Convert but some logic from memory and logic commands were still used for these functions where possible.

Penultimate part of this program was the exception handling. The Appendix A was utilized for this part to get the Bus Address and other information from stack. After this was achieved the remaining part was straightforward.

After all the functions and error handling was implemented there was an exhaustive check to make sure all functions were working as expected.

Overall this was a great design experience. Using separate subroutines for Hex and ASCII conversion was very useful. The overall strategy of developing the skeleton first and then implementing functions were very useful.

However, rather than developing the functions under the main program, it would have been better to develop the functions in separate EASY 68k forms and copy paste them into the main program when ready.

Lastly the different debugging tools in EASY 68k such as auto-trace and breakpoints were heavily utilized and very useful.

5-) Feature Suggestions

The project was successfully implemented and everything worked as expected but there were some features that might have been would be implemented in the future

First would be a series of TRAP instructions for the user to use in their programs. User can use these traps to print data, get input from the user and return control back to the user. This would allow the user to utilize this program as a fully functional 68k monitor.

Another instruction would be the disassembler option for Memory Modify. Using this, the user can enter function directly into memory and run them using the GO command. This function is already present in TUTOR.

Lastly the biggest improvement would be granting access to memory locations higher than 10 000. The user won't be able to access these locations due to the way argument parser was implemented. A great improvement would be to copy logic similar to that used in Memory Modify Long to allow more access address to user.

Some extra improvements would be to divide the code to further subroutines and calling these when required rather than repeating these actions in the commands. This would take up much less space and make the code more legible. Some helper subroutines might be the printing and bit manipulation ones.

6-) Conclusion

This project was a great way to polish knowledge in 68k and the assembly language for this processor. This project also contributed to the knowledge of debugging tools of EASY 68k program. ASCII to HEX and HEX to ASCII conversion was a big part of this project. The project was extremely time consuming but overall result was very satisfactory. This monitor would probably not be very useful in the real world but it was a great insight to how TUTOR and TUTOR like applications for modern computers like Command Prompt and Linux Shell. Although the program itself won't be useful in the real world, the lessons taken from the development of this project would be very useful in any other software project.

7-) References

- [1] Experiment 2 Lab Manual
- [2] ECE 441 Monitor Design Project Spring 2017
- [3] Experiment 3 Lab Manual
- [4] Educational Computer Board Manual Appendix
- [5] ECE 441 Course Instructions Appendix A