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 2

2-) Monitor Program 3

2.1-) Command Interpreter 4

2.1.1-) Algorithm and Flowchart 5

2.1.2-) 68000 Assembly Code 5

2.2-) Debugger Commands 6

2.2.1-) Help 6

2.2.2-) Memory Display 7

2.2.3-) Sort 7

2.2.4-) Memory Modify 8

2.2.5-) Memory Set 8

2.2.6-) Block Fill 9

2.2.7-) Block Move 10

2.2.8-) Block Test 10

2.2.9-) Block Search 11

2.2.10-) Execute Program 11

2.2.11-) Display Formatted Registers 12

2.2.12-) Exit Monitor Program 12

2.2.13-) Ascii to Hext and Hex to Ascii Convertor 12

2.2.14-) Clear Registers 12

2.3-)Exception Handlers 13

2.3.1-) Bus Error Exception 13

2.3.2-) Address Error Exception 14

2.3.3-) Illegal Instruction Exception 14

2.3.4-) Privilege Violation Exception 14

2.3.5-) Divide by Zero Exception 15

2.3.6-)Check Instruction Exception 15

2.3.7-) Line A and Line F Emulators 15

2.4-)User Instructional Manual Exception Handlers 16

2.4.1-) Help Menu 16

3-) Discussion 17

4-) Feature Suggestions 18

6-) Conclusions 19

7-) References 20

***Abstract***

The summary of your design should go here. Someone who reads this abstract should have a clear understanding of your design and the overall flow of the report.

# *1-) Introduction*

This will be an introduction to your design. You can give design objectives, a clear description of the problem and design methodology and technology used. Any figures and tables should have clear descriptions.

\*recite the intro page from the assignment with stuff

*Figure 1.1. Monitor command interpreter block diagram*

***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.

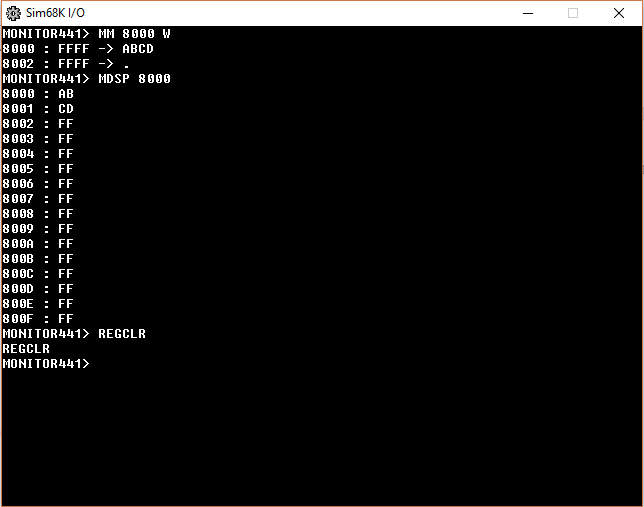


Figure 1: Monitor Program

In addition to those functions 2 helper functions that Convert Hexadecimal number to ASCII and ASCIII to Hexadecimal were implemented. These functions were implemented to assist with interpreting commands with various other subparts 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) than 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 than 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.

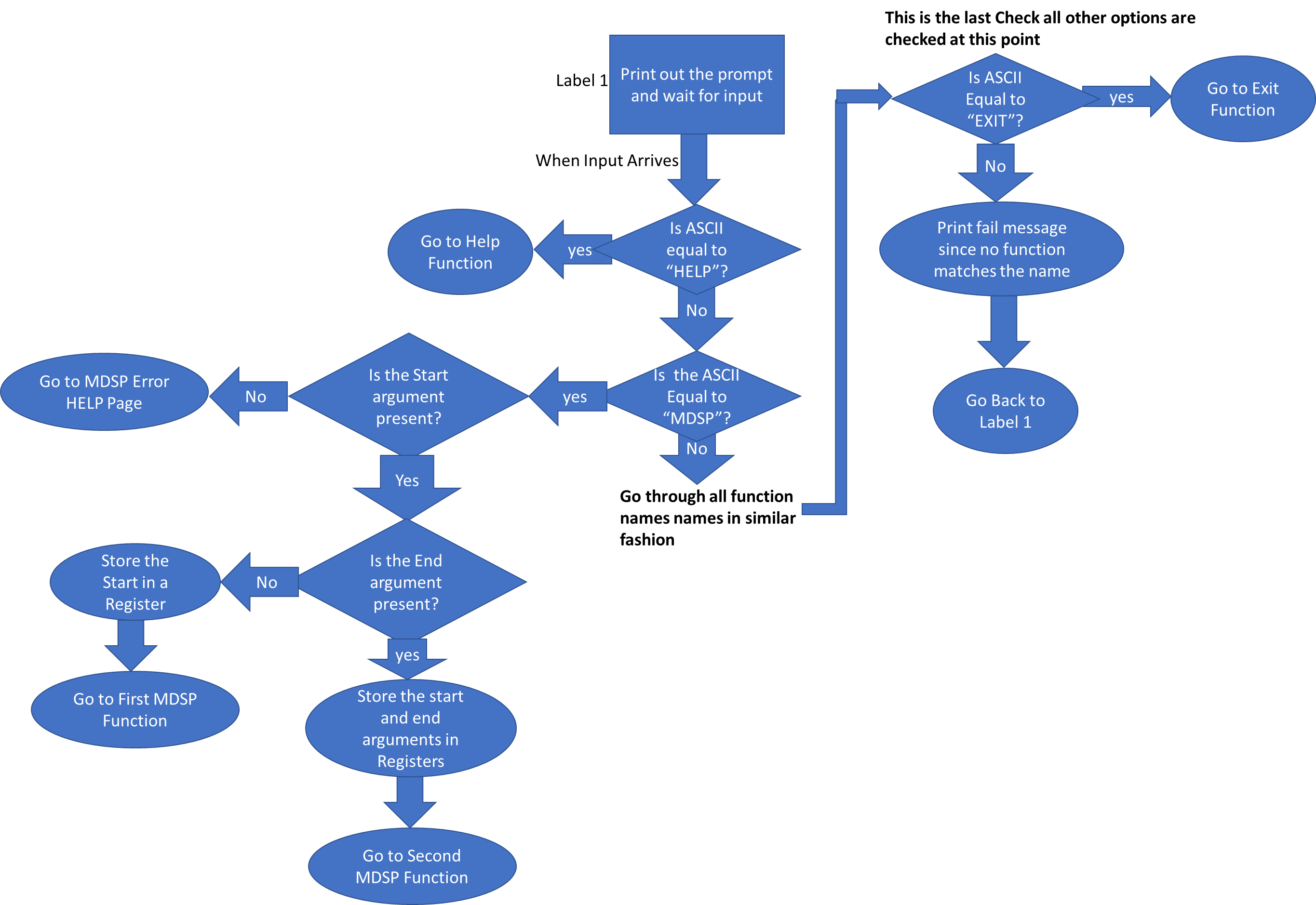


Figure 2: Command Interpreter Architecture

yes

No

yes

No

Is ASCII equal to “HELP”?

Print out the prompt and wait for input

When Input Arrives

Is the ASCII Equal to “MDSP”?

yes

Yes

Is the End argument present?

No

yes

**Go through all function names names in similar fashion**

Is ASCII Equal to “EXIT”?

No

**This is the last Check all other options are checked at this point**

Label 1

***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

COMPAREBMOV ;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 #BMOVPROMPT,A0

COMPAREBMOV2

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 COMPAREBMOV2

;CHECK FOR THE ARGUMENTS

CLR.L D4 ;FIRST ARG

CLR.L D5 ;SECOND ARG

CLR.L D6 ;THIRD ARG

CMPBMOVL1

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 CMPBMOVL2

LSL.L #8,D4

BRA CMPBMOVL1

CMPBMOVL2

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 CMPBMOVL3

LSL.L #8,D5

BRA CMPBMOVL2

CMPBMOVL3

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 CMPBMOVL3

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 COMPAREREGCLR

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 #REGCLRPROMPT,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 #EXITPROMPT,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 a 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***

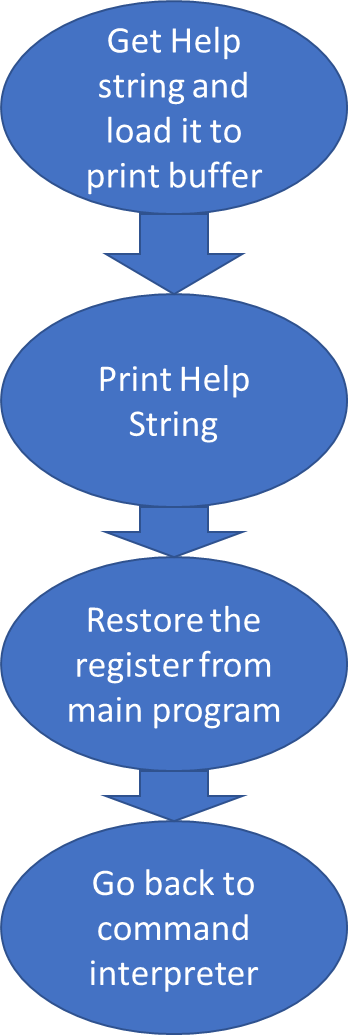


Figure 3: Help flowchart

***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

*Figure 2.7. Help Assembly Code*

***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

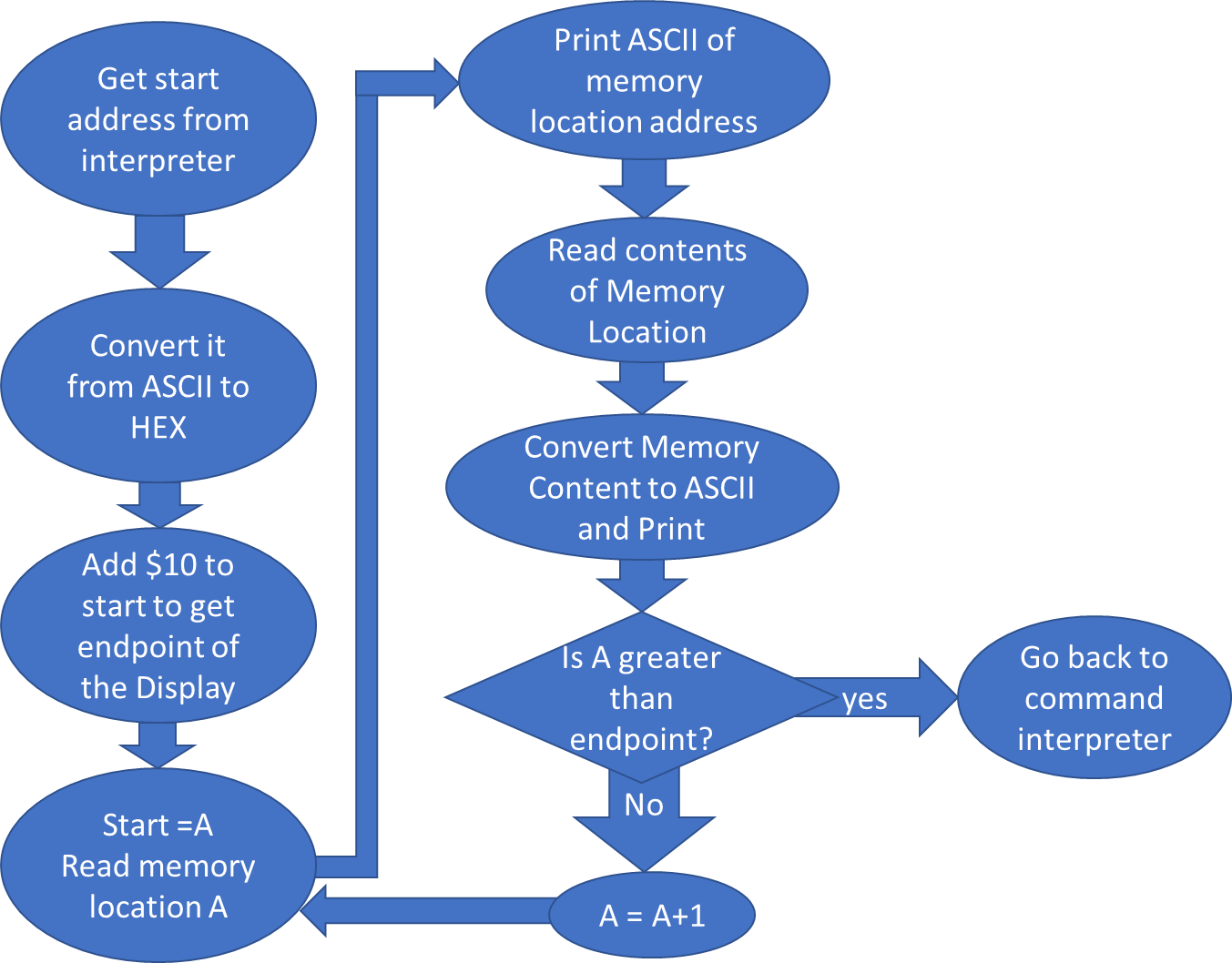


Figure 4: MDSP where only one argument is given

Case II

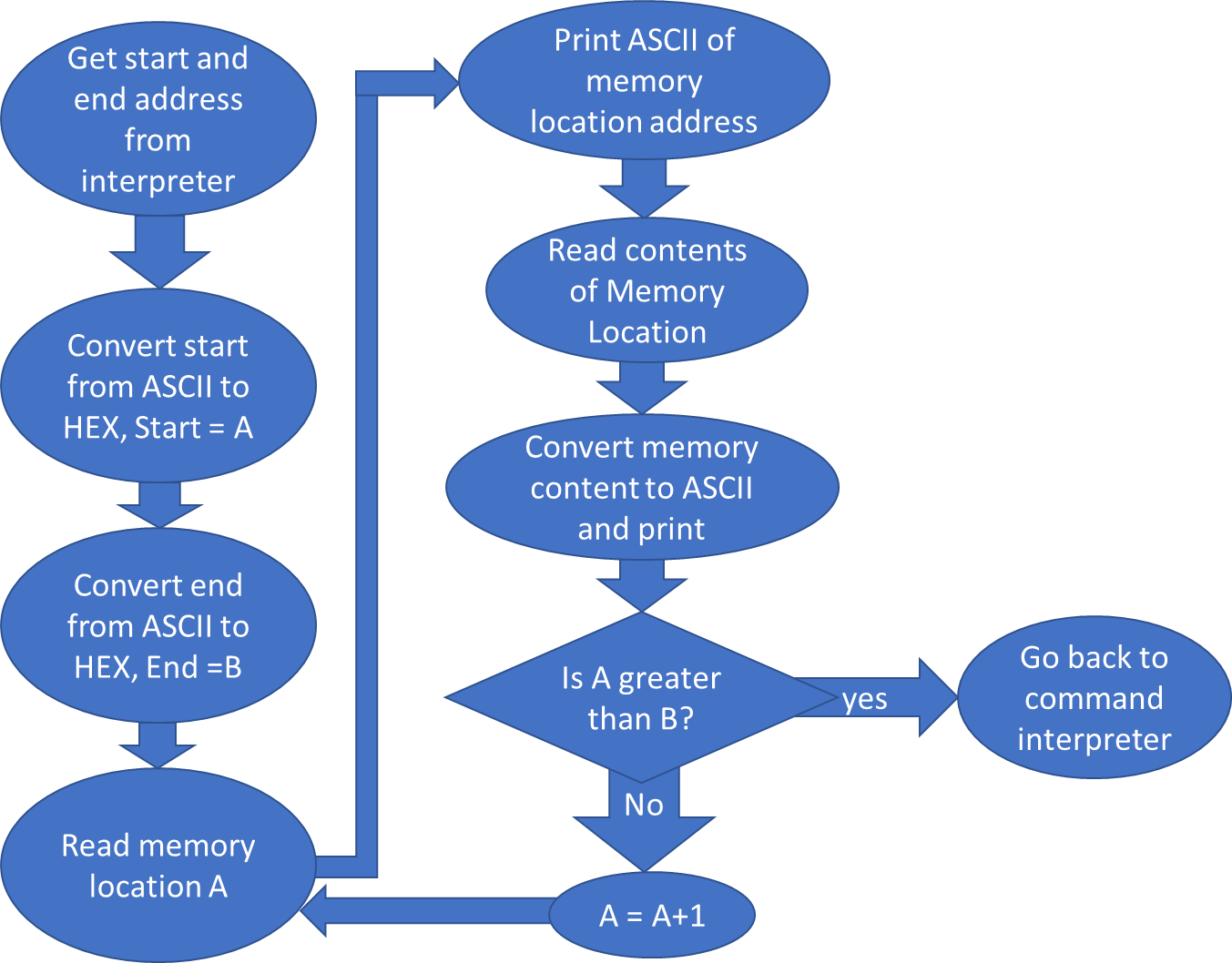


Figure 5: MDSP where both end and start is given

***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 ASCIITOHEX

;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 ASCIITOHEX

;DISPLAY EVERYTHIGN FROM D4 TO D5\

MOVE.L D4,A2 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

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 those 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 3 of ECE 441 course.

***2.2.3.1-) Sort Flowchart***

Sort Ascending

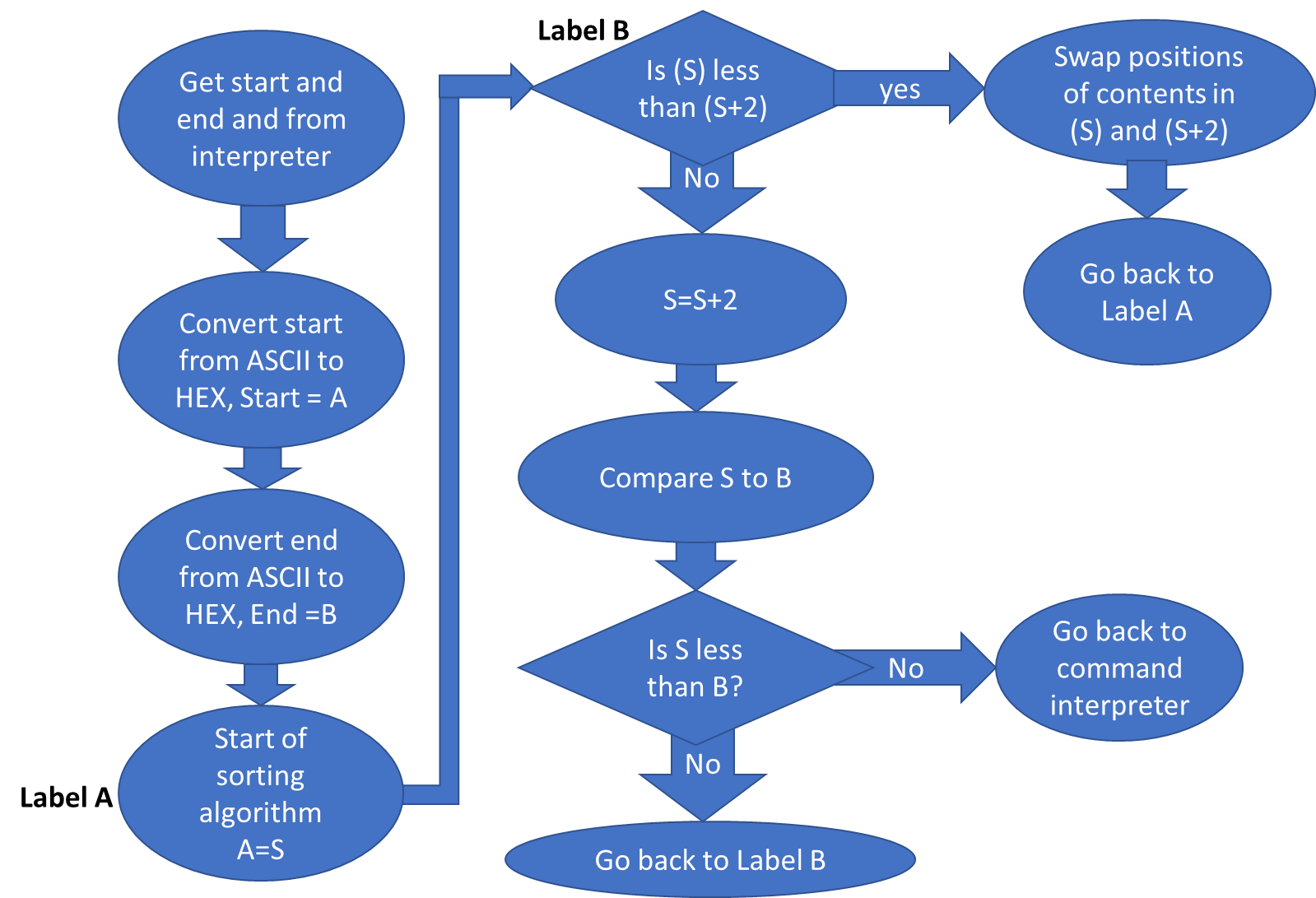


Figure 6: Flowchart for sorting ascending, very similar to descending sort

***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 ASCIITOHEX

MOVE.L D4,A0 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

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 and 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

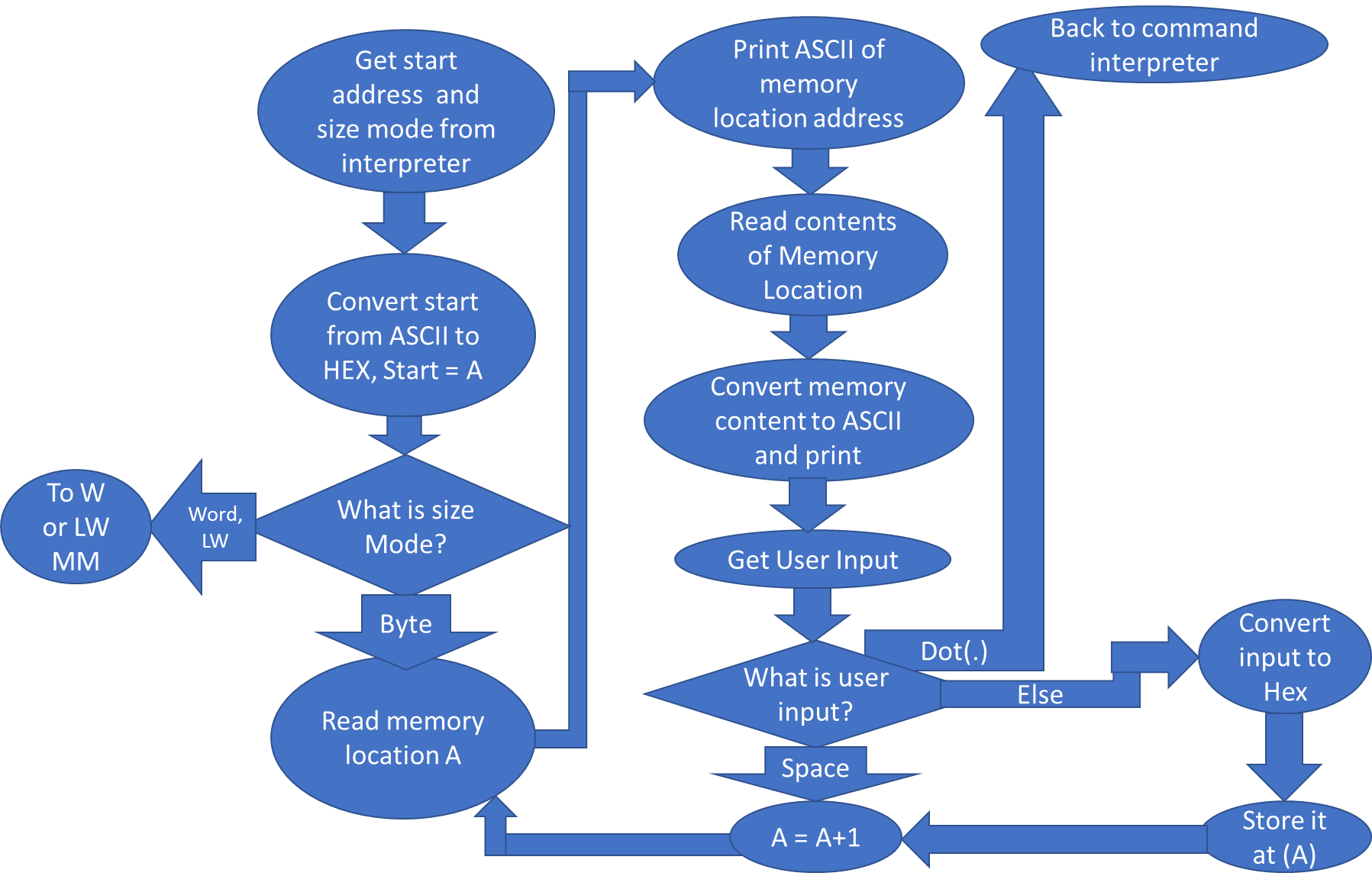


Figure 7: MM operation in Byte mode

***2.2.4.2-) Memory Modify Assembly Code***

MMFUNCTION ;argument passed as D4 AS MEMLOC AND d5 AS THE MODE

BSR ASCIITOHEX

;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 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 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 ASCIITOHEX

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 ASCIITOHEX

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 ASCIITOHEX

MOVE.W D4,-4(A2)

;PRINT SECOND WORD

MOVE.L (A1),D4

BSR ASCIITOHEX

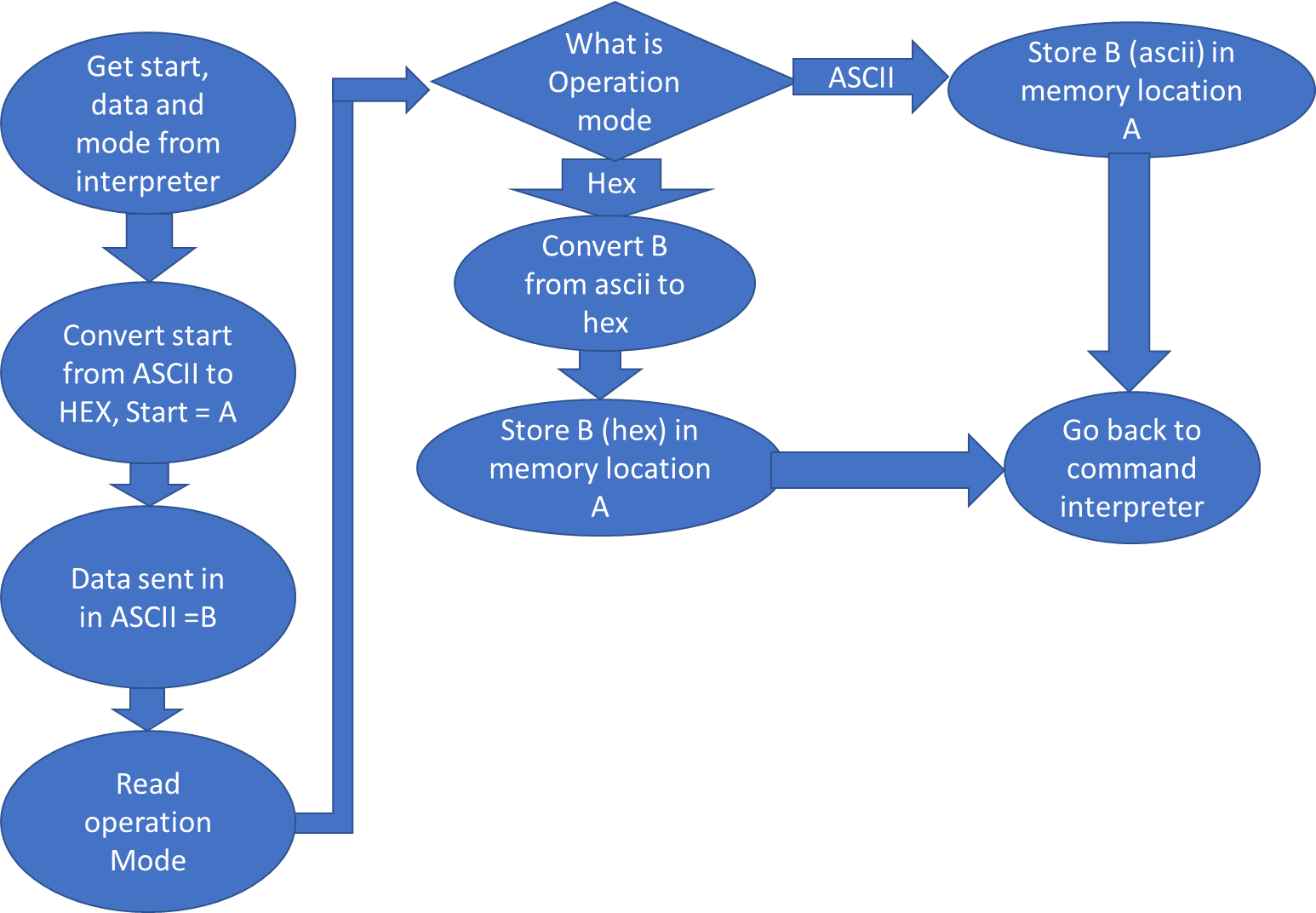
MOVE.W D4,-2(A2)

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***



***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 ASCIITOHEX

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 ASCIITOHEX

MOVE.W D4,(A2)

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

BRA pSTART

MSFUNCTIONL2

MOVE.L D5,(A2)

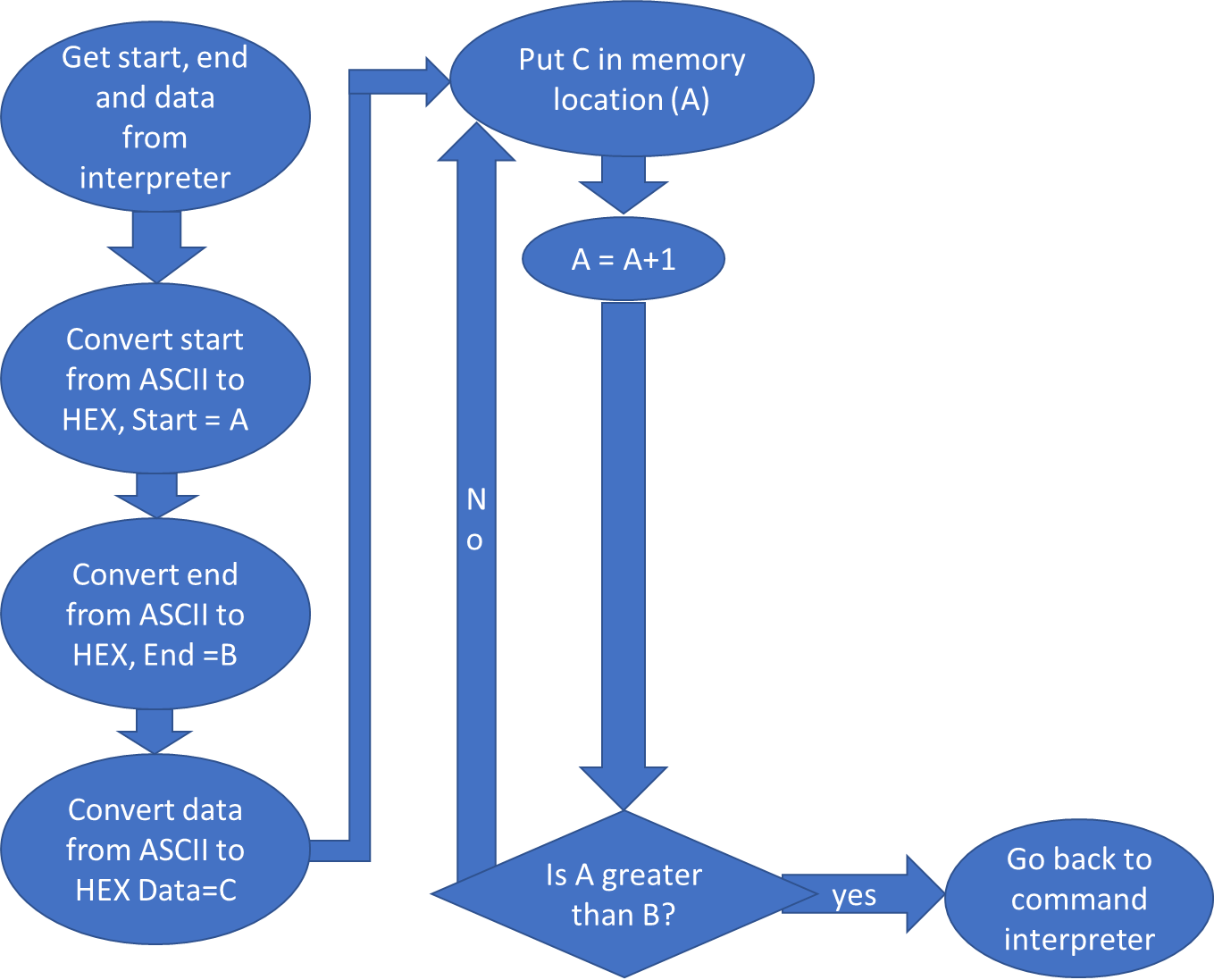
MOVEM.L (SP)+, A0-A6/D0-D7

BRA pSTART

***2.2.6-) Block Fill***

It is similar to 2.2.1

***2.2.6.1-) 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 ASCIITOHEX

MOVE.L D4,A2 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

MOVE.L D4,A3 ;ENDING

ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED

MOVE.L D6,D4

BSR ASCIITOHEX

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

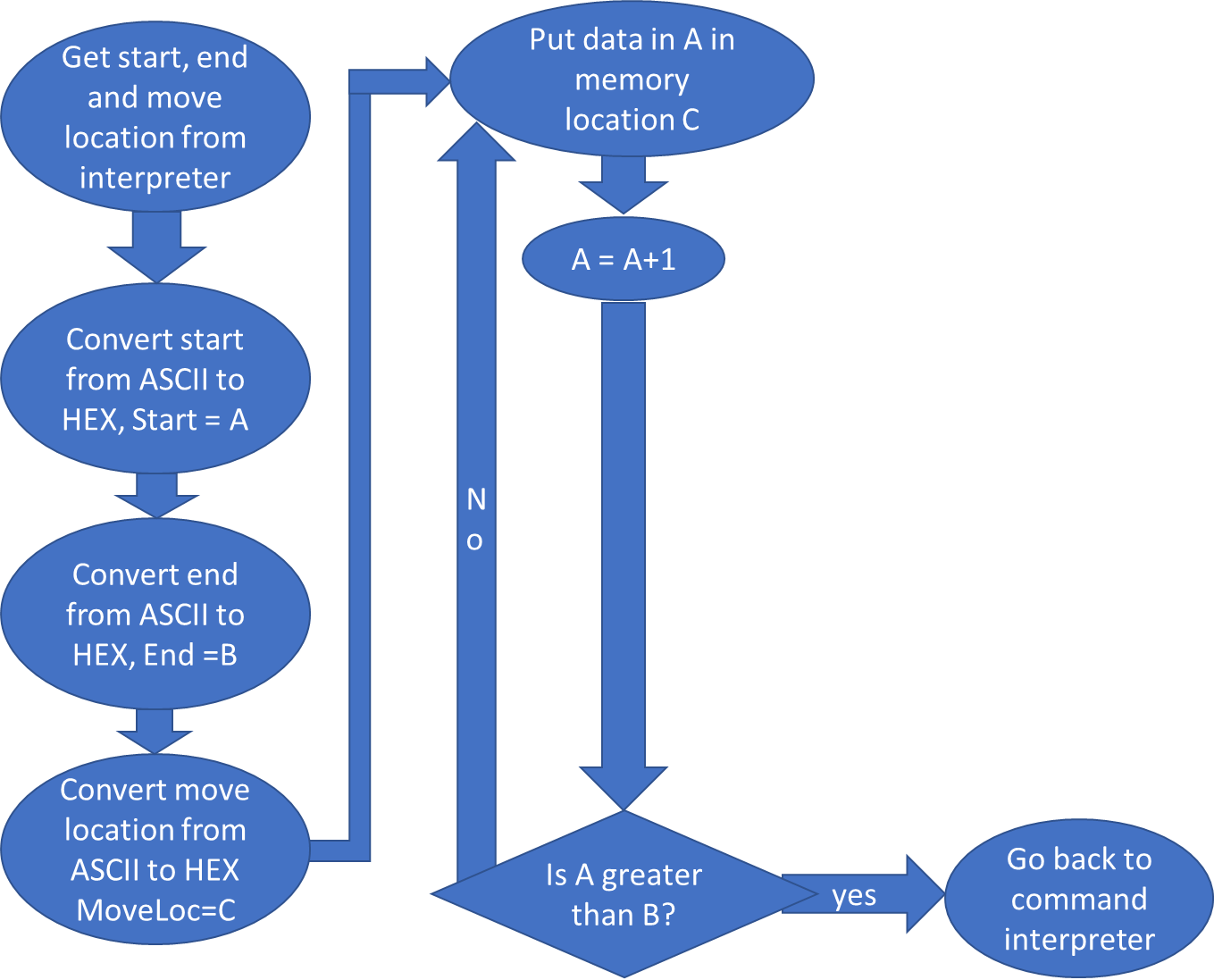
MOVEM.L (SP)+,A0-A6/D0-D7

BRA pSTART

***2.2.7-) Block Move***

It is similar to 2.2.1

***2.2.7.1-) 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

BSR ASCIITOHEX

MOVE.L D4,A2 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

MOVE.L D4,A3 ;ENDING

ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED

MOVE.L D6,D4

BSR ASCIITOHEX

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

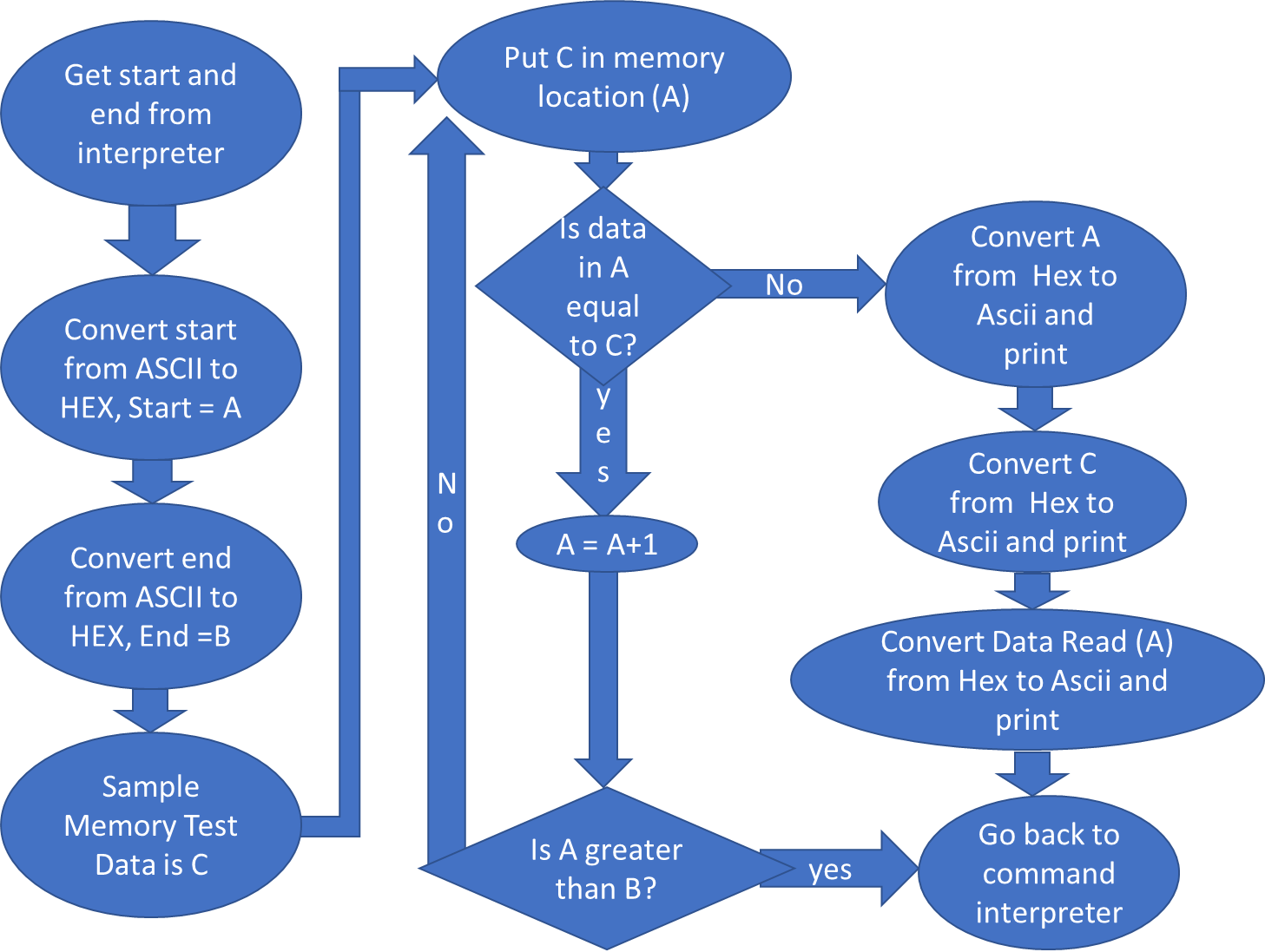
MOVEM.L (SP)+, A0-A6/D0-D7

BRA pSTART

***2.2.8-) Block Test***

It is similar to 2.2.1

***2.2.8.1-) Block Test Flowchart***



***2.2.8.2-) Block Test Assembly Code***

BTSTFUNCTION

BSR ASCIITOHEX

MOVE.L D4,A2 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

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 ADDRESSS

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

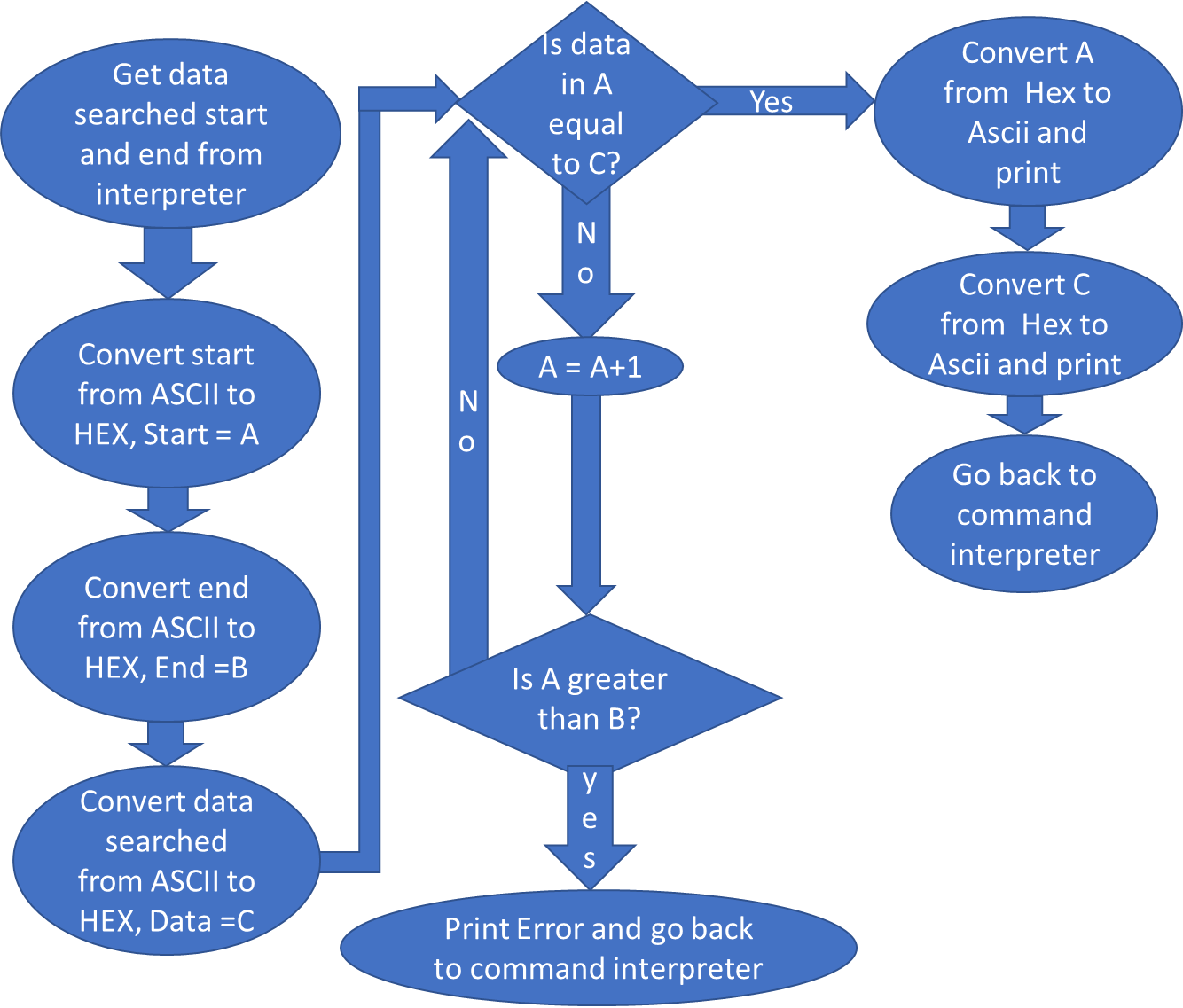
MOVEM.L (SP)+, A0-A6/D0-D7

BRA pSTART

***2.2.9-) Block Search***

It is similar to 2.2.1

***2.2.9.1-) Block Search Flowchart***



***2.2.9.2-) Block Search Assembly Code***

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

BSR ASCIITOHEX

MOVE.L D4,A2 ;BEGGINING

MOVE.L D5,D4

BSR ASCIITOHEX

MOVE.L D4,A3 ;ENDING

ADD.L #1,A3 ;MAKE SURE LAST ONE IS ENTERED

MOVE.L D6,D4

BSR ASCIITOHEX

;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 ADDRESSS

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

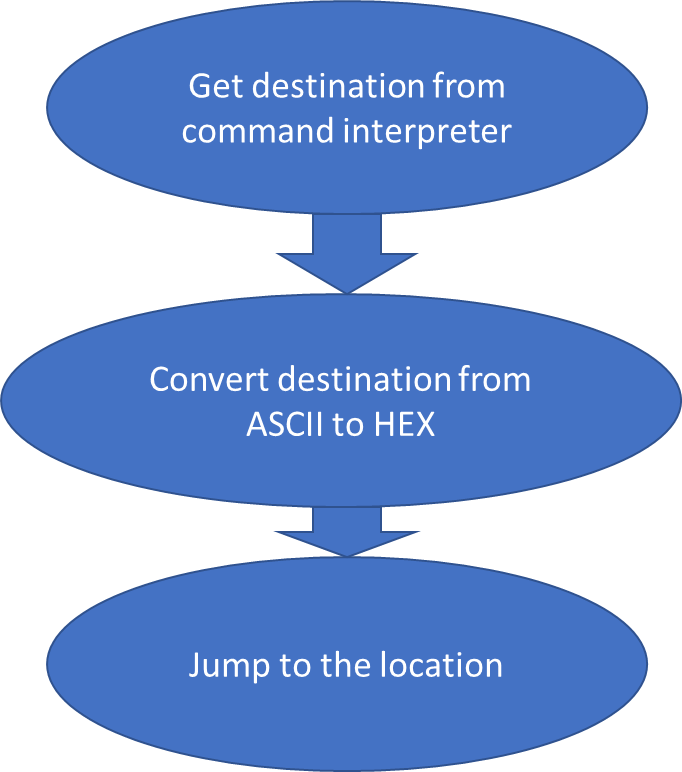
MOVEM.L (SP)+, A0-A6/D0-D7

BRA pSTART

***2.2.10-) Execute Program***

\*Explain how it can act as a jump and user would have a trap that would return the execution

***2.2.10.1-) Execute Program Flowchart***



***2.2.10.2-) Execute Program Assembly Code***

GOFUNCTION ;D4 IS THE MEMLOC TO EXECUTE

BSR ASCIITOHEX

MOVE.L D4,A1

BSR (A1)

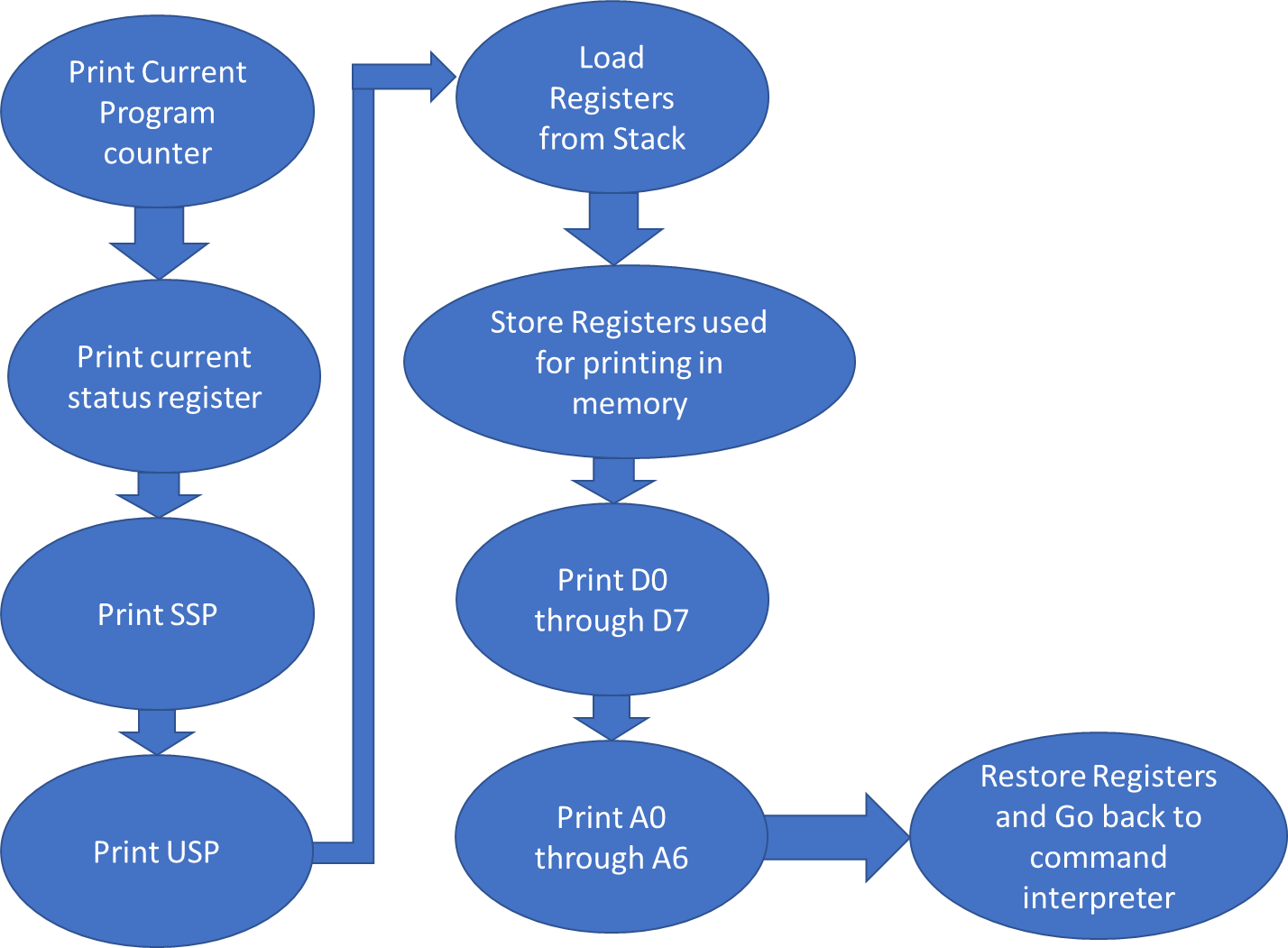
MOVEM.L (SP)+, A0-A6/D0-D7

BRA pSTART

***2.2.11-) Display Formatted Registers***

It is similar to 2.2.1

***2.2.11.1-) Display Formatted Registers Algorithm and Flowchart***



***2.2.11.2-) Display Formatted Registers Assembly Code***

DFFUNCTION

;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***

It is similar to 2.2.1

***2.2.12.1-) Exit Monitor Program Flowchart***



***2.2.12.2-) Exit Monitor Program Assembly Code***

EXITFUNCTION

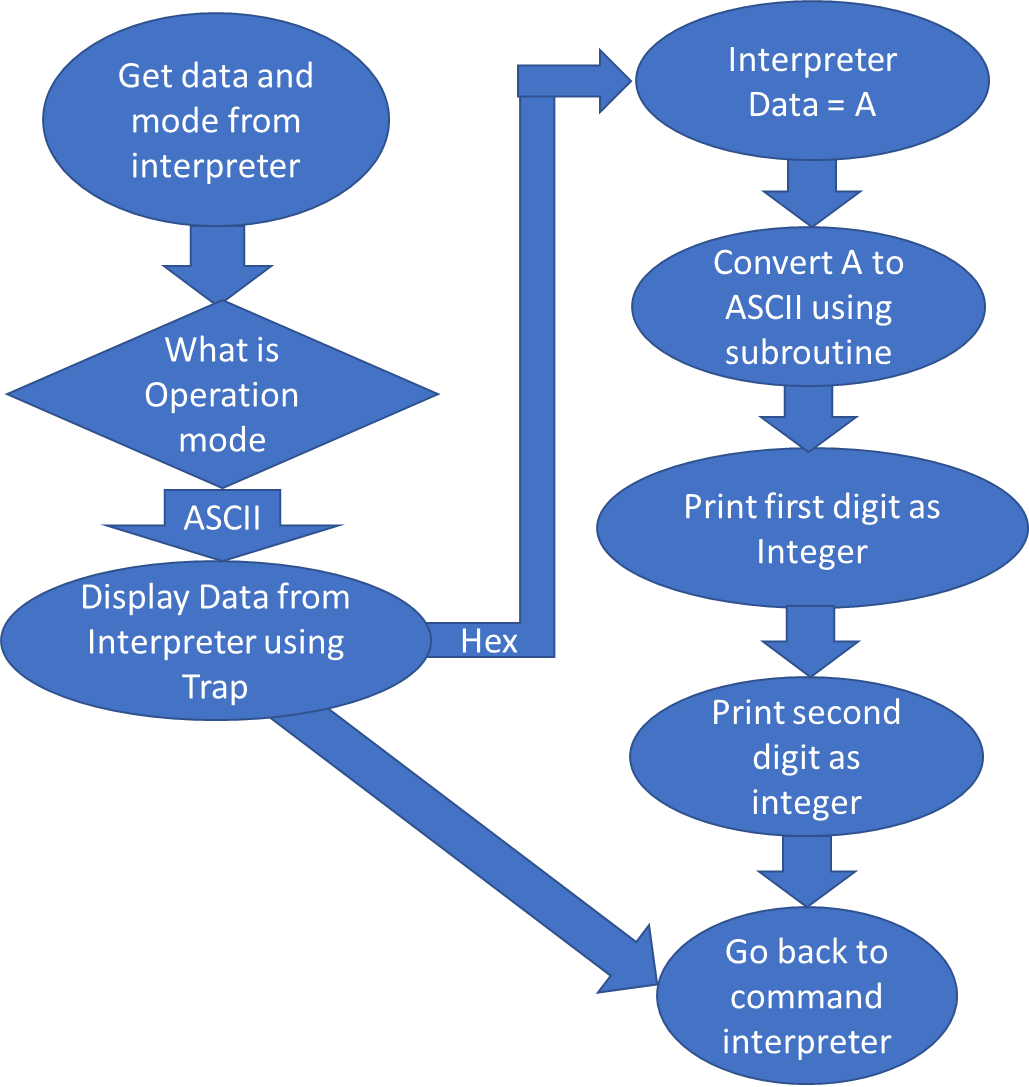
MOVE.B #9,D0

TRAP #15

***2.2.13-) Hex to ASCII and ASCII to Hex Convertor***

It is similar to 2.2.1

***2.2.13.1-) Hex to ASCII and ASCII to Hex Convertor 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 ASCIITOHEX

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

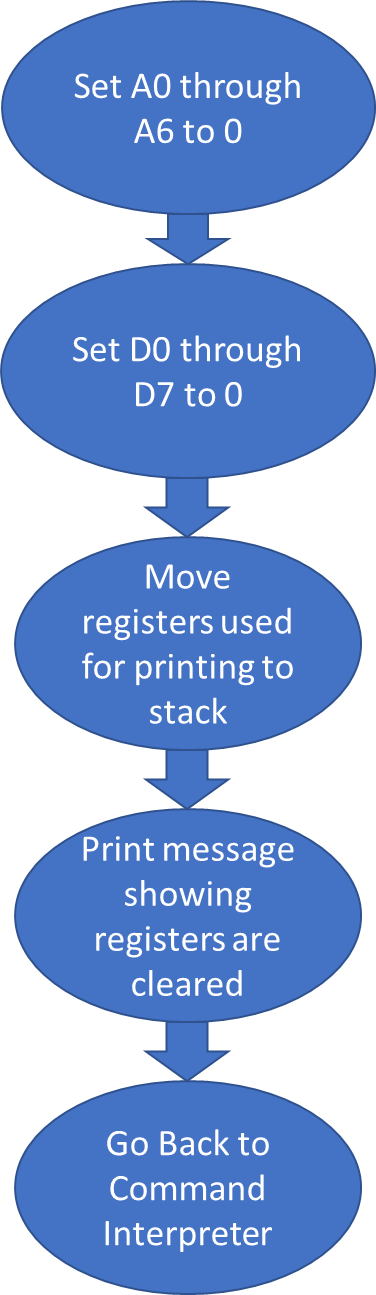
MOVEM.L (SP)+,A0-A6/D0-D7

BRA pSTART

***2.2.14-) Clear Registers***

It is similar to 2.2.1

***2.2.14.1-) Clear Registers Flowchart***



***2.2.14.2-) Clear Registers Assembly Code***

REGCLRFUNCTION:

;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 #REGCLRPROMPT,A1

MOVE.B #13,D0

TRAP #15

MOVEM.L (SP)+,A1/D0

BRA pSTART

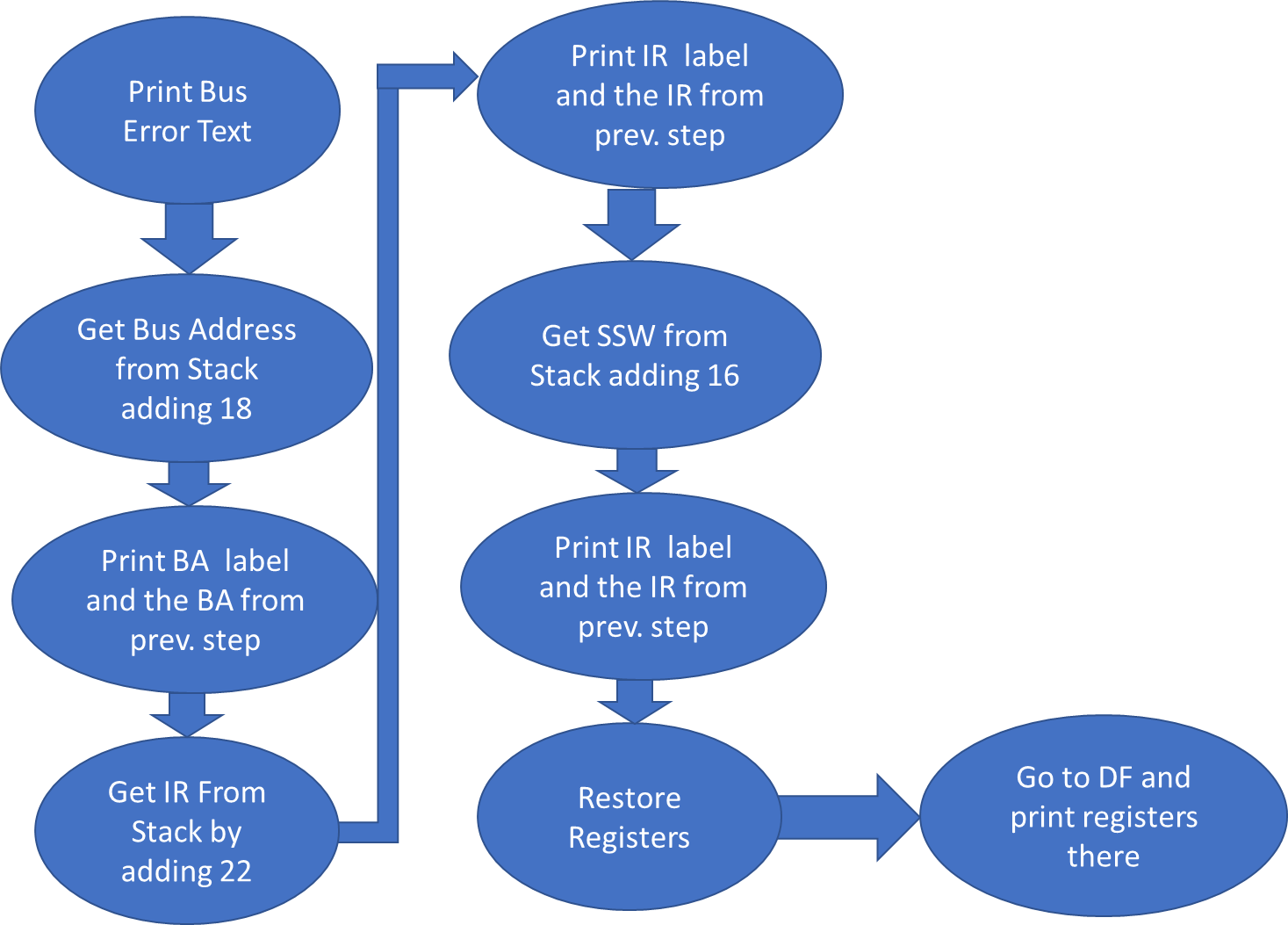
***2.3-) Exception Handlers***

Brief information about Exception Handlers should be given here.

***2.3.1-) Bus Error Exception***

A clear description of this debugger command should be given here.

***2.3.1.1-) Bus Error Flowchart***



***2.3.1.2-) Bus Error Exception Assembly Code***

BUSERRORFUNCTION

;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

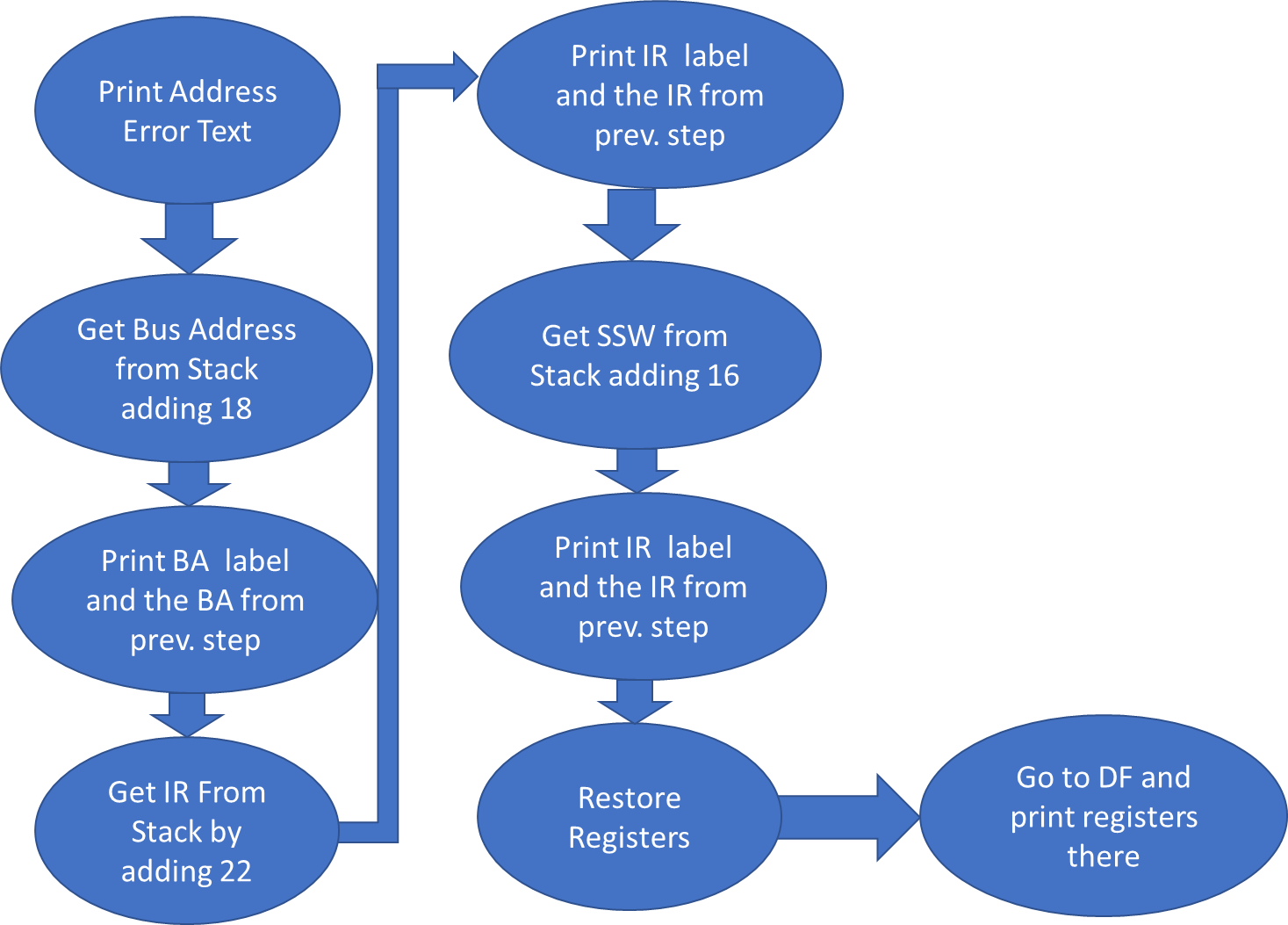
MOVEM.L (SP)+,A0-A6/D0-D7

BRA DFFUNCTION

***2.3.2-) Address Error Exception***

Mention Similarity to Bus Error

***2.3.1.1-) Address Error Exception 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

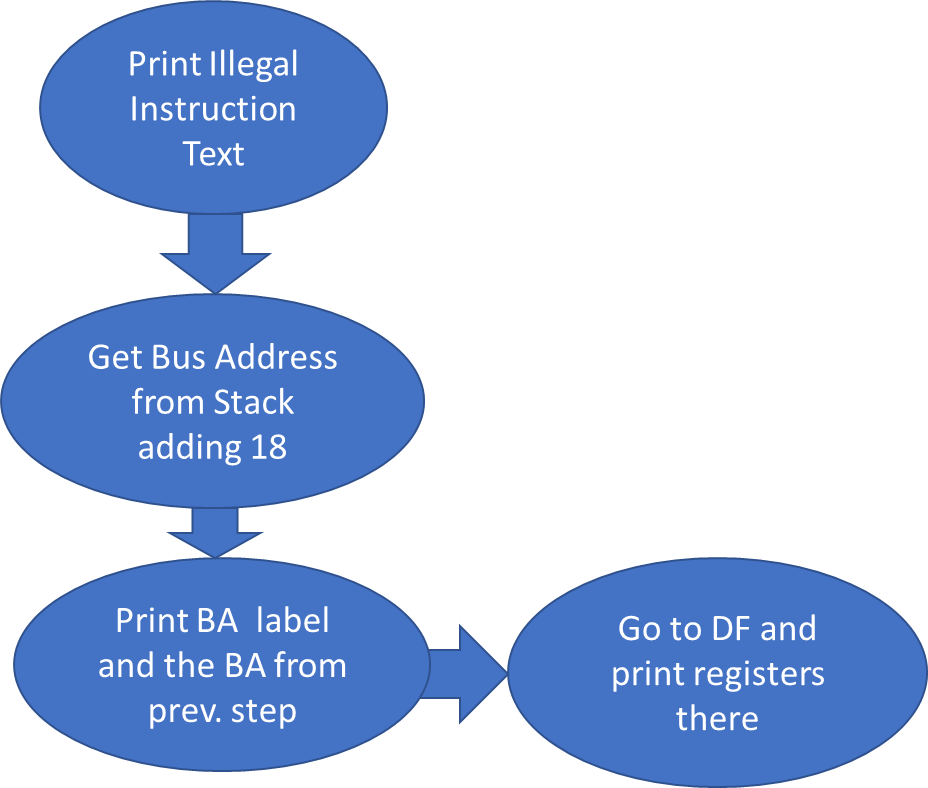
TRAP #15

BRA DFFUNCTION

***2.3.3-) Illegal Instruction Exception***

It is similar to 2.3.1

***2.3.3.1-) Illegal Instruction Exception Algorithm and Flowchart***

It is similar to 2.3.1.1

***2.3.3.2-) Illegal Instruction Exception Assembly Code***

ILLEGALINSTRUCTIONFUNCTION

;PRINT OUT ILLEGAL INSTR STRING

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

LEA ILLEGALVECTORTEXT,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

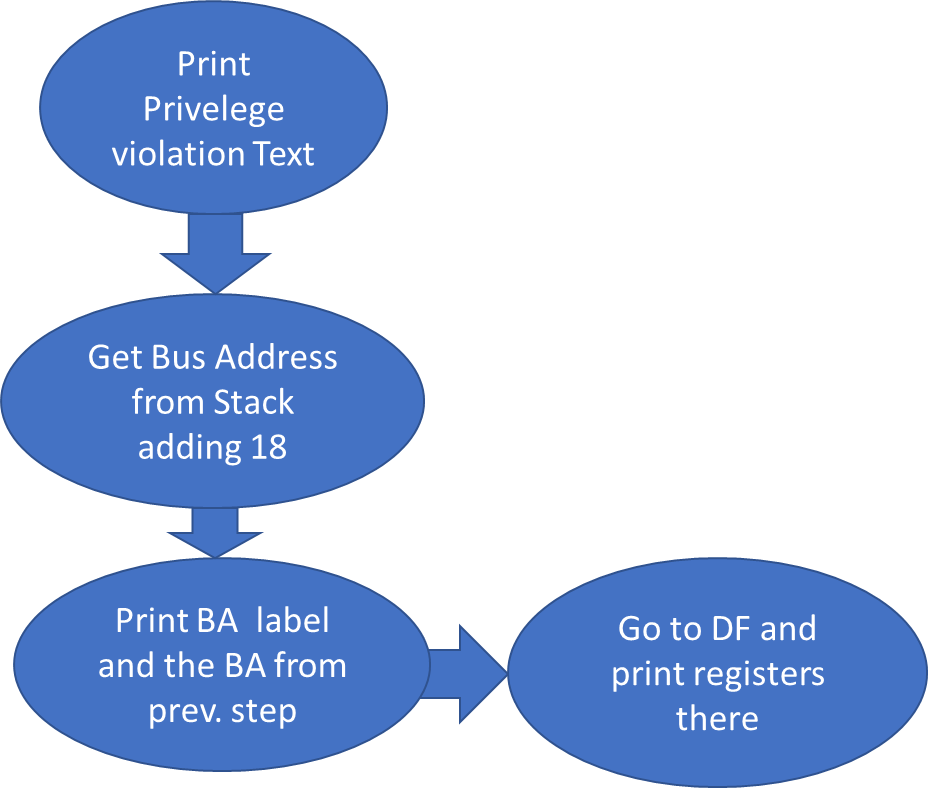
TRAP #15

BRA DFFUNCTION

***2.3.4-) Privilege Violation Exception***

It is similar to 2.3.1

***2.3.4.1-) Privilege Violation Exception Flowchart***

******

***2.3.4.2-) Privilege Violation Exception Assembly Code***

PRIVELEGEVIOLATIONFUNCTION

;PRINT OUT PRIV VIOL STRING

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

LEA PRIVELEGEVECTORTEXT,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

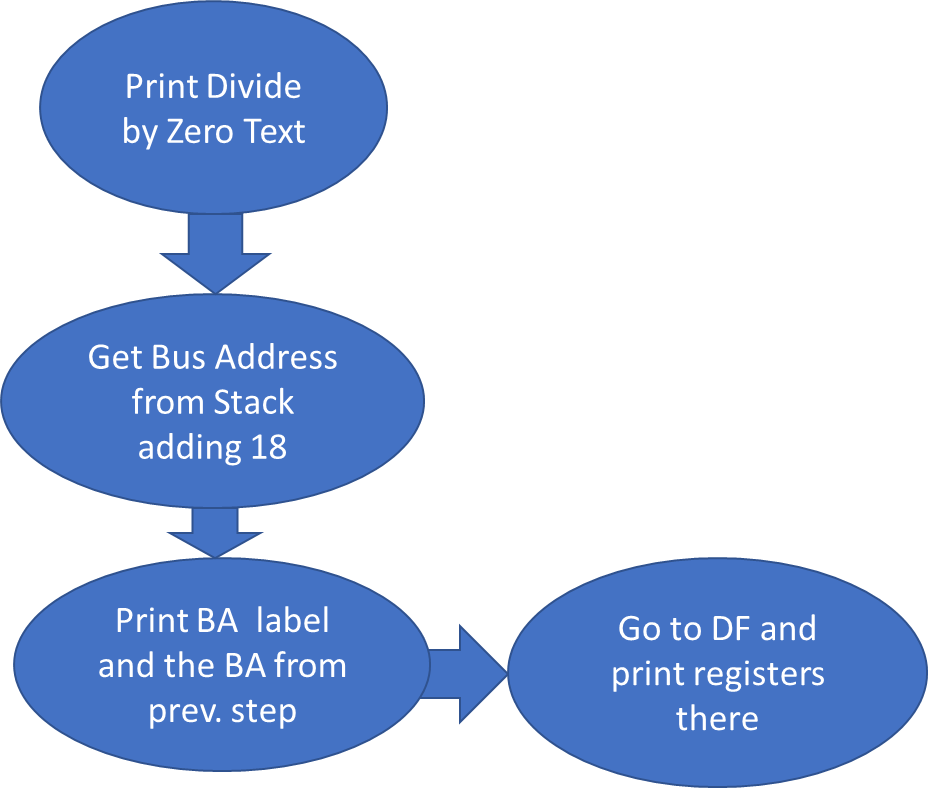
TRAP #15

BRA DFFUNCTION

***2.3.5-) Divide by Zero Exception***

It is similar to 2.3.1

***2.3.5.1-) Divide by Zero Exception Flowchart***

It is similar to 2.3.1.1

***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

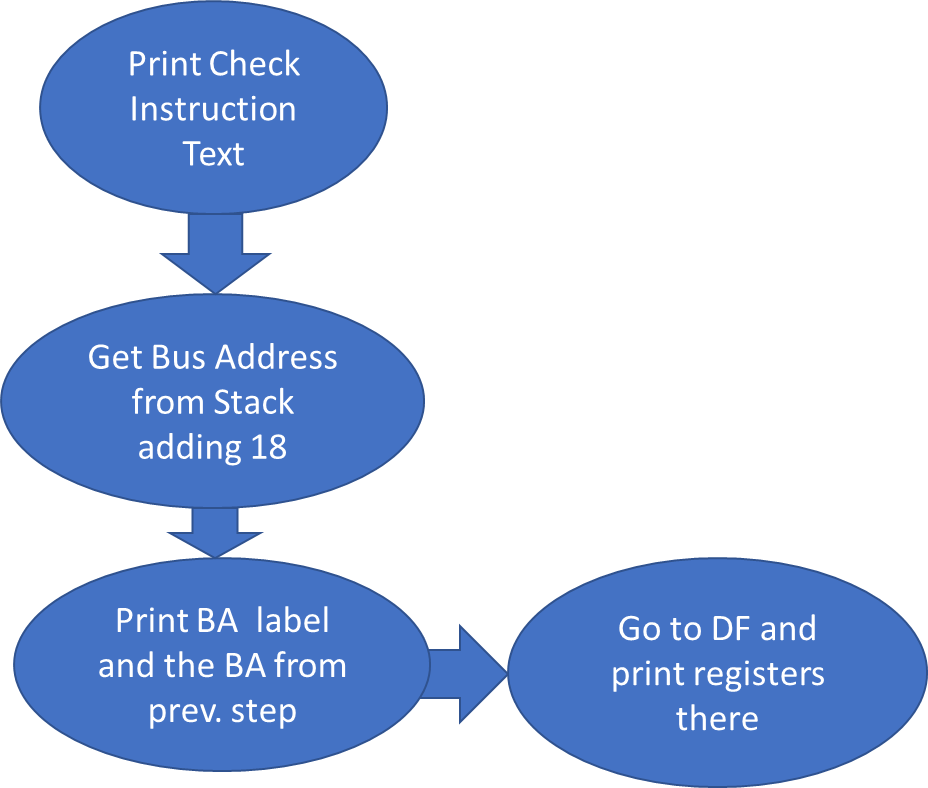
TRAP #15

BRA DFFUNCTION

***2.3.6-)*** ***Check Instruction Exception***

It is similar to 2.3.1

***2.3.6.1-) Check Instruction Exception Flowchart***



***2.3.6.2-) Check Instruction Exception Assembly Code***

CHECKINSTRUCTIONFUNCTION

;PRINT OUT CHECK STRING

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

LEA CHECKVECTORTEXT,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

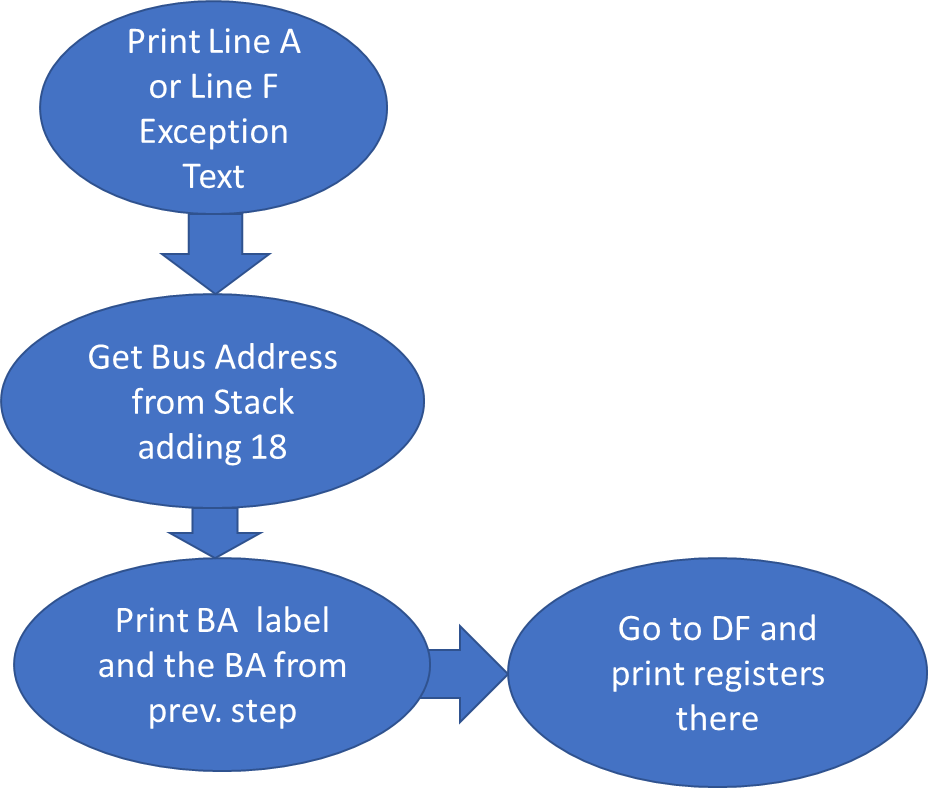
TRAP #15

BRA DFFUNCTION

***2.3.7-) Line A and Line F Emulators***

It is similar to 2.3.1

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

It is similar to 2.3.1.1

***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-) User Instructional Manual Exception Handlers***

Brief information about Instructional Manual Handlers should be given here.

***2.4.1-) Help Menu***

A clear description of this debugger command should be given here

***2.4.1.1-) Algorithm and Flowchart***

An algorithm of the design and its flowchart will be explained here. You may need to include comments for your algorithm.

*Clear //this where things starts*

*Do this m=0 // assign m*

*While m > n // while m > n*

*If m > n //*

*Do this //*

*Else //*

*Do these more //*

*End if //*

*m = m + 1 // increment m by 1*

*finish // finish*

*Figure 2.11. Debugger Command # 1 Algorithm*

It may be necessary to explain more about your flowchart and your design ideas.



*Figure 2.12. Debugger Command # 1 Flowchart*

***2.4.1.2-) Assembly Code***

The assembly code should be written using the algorithm above.

*ORG $1000*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*\*\*\*\*\*\*\*\*\* // Comment goes here*

*END ;*

*Figure 2.13. Debugger Command # 1 Assembly Code*

***2.5-) Extra Helper Functions***

***2.5.1-) ASCII to Hex Convertor***

***2.5.2-) Hex to ASCII Convertor***

***3-) Quick User Manual***

1. **HELP:** Display Help all command available and usage

FORMAT: ->HELP

1. **MDSP:** Display Memory range given

FORMAT: MDSP <ADDR>- OR -MDSP <SADDR> <EADDR>-

1. **SORTW:** Sort Words in the memory range

FORMAT: ->MM <SADDR> <EADDR> X- WHERE X=A OR D, A=ASCENDING D=DESCENDING

1. **MM:** Display memory and modify or enter data, used with different sizes

FORMAT: ->MM <ADDR> X- WHERE X=B,W,L

1. **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

1. **BF:** Fill the memory block from start to end with the data given data

FORMAT: ->BF <SADDR> <EADDR> <DATA>

1. **BMOV:** Move the specified memory block from start to end to destination

FORMAT: ->BMOV <SADDR> <EADDR> <DESTADDR>

1. **BTST:** Destructively test the memory block, if error detected, display

FORMAT: ->BTST <SADDR> <EADDR>

1. **BSCH:** Search the memory block for a given data

FORMAT: ->BSCH <SADDR> <EADDR> <DATA>

1. **GO:** Go to memory address and start execution there

FORMAT: ->GO <ADDR>-

1. **DF:** Display all the registers  
   FORMAT: ->DF
2. **EXIT:** Exit the program  
   FORMAT: ->EXIT
3. **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

1. **REGISTER CLEAN**: Set all registers to 0  
   FORMAT: ->REGCLR

***4-) Discussion***

Design challenges and discussion about this project should be provided here. You may subdivide this section further and supply figures and table if necessary.

Discussion for engineering and design challenges

***5-) Feature Suggestions***

You may suggest ideas for expanding this project, such as exception handlers, etc. You may subdivide this section further and supply figures and tables if necessary.

***6-) Conclusion***

The conclusion goes here.[1]

***7-) References***

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

[1] Lab 3 Manual, ECE 441 IIT

[2] project description