ECE 441
Microprocessors

Instructor: Dr. Jafar Saniie
Teaching Assistant: Guojun Yang

Final Project Report:
MONITOR PROJECT
04/27/2018

By: Hugo Trivino

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

Signature : _____

Table of Contents

Abstract

This report describes the implementation of a monitor program that will be used as a tool for programming and troubleshooting applications written for the MC68000 family of microprocessors. This implementation consist of a shell that supports fourteen different debugging commands and it has the ability to handle the most common types of exceptions.

1-) Introduction

The following document will explain the flow of the overall program in chapter one. In chapter 2.1, there is a description of the command interpreter, which includes the parser and string comparator. In chapter 2.2 there is a more detail description of each of the debugging commands and their usage. In chapter 2.3 there is a description of the exception handlers. Chapters 3 through 6 consists of a discussion, feature suggestions, and conclusions respectively.

2-) Monitor Program

The monitor program consist of one command interpreter for a series of 12 commands that can be used by the user in order to make better and simpler use of the MC68k microprocessor. In this case, this monitor program solution was developed using the EASy68K simulator and therefore it was developed using the trap 15 system calls for reading and for writing to the terminal. There are also 7 exception handlers that are the most common exceptions. In this way, if the user makes a mistake while inputting its own programs, the microprocessor will recover from the error indicating the address at which it took place and the type of error. After recovery, the user will have the shell running and waiting for input.

2.1-) Command Interpreter

The command interpreter is mainly made of three pieces. Firstly, there is a command shell that takes input from the user and it only stops when the user type “EXIT” as a command. Once this command is taken as an input (trap #15 number 2), the second step is going through a function called parser that takes the input and iterates through the string representation of all the commands that there are. Since the commands are organized alphabetically, if the input happens to be smaller (as ASCII comparison), there is not a chance that the subsequent commands will match. This string comparison is performed within the a string comparator function that returns (in D0) the value -1 if the input is smaller than a command, +1 if the input is greater, and 0 if the input is equal. If the input is smaller than the command, there will be an error message and the user can try inputting another command. If the input is greater, the parser will compare the input to the next string until there are not more commands. In the case in which there are not more commands, the same error will be displayed. Finally, only in the outcome in which the output of the string comparator is zero, the parser function performs a jump to the debugger command correctly selected by the user.

2.1.1-) Algorithm and Flowchart

The pseudo code of the algorithm previously described is included as follows:

```
BASH PROGRAM                    //this where bash starts
While (true and not exit)       // only exits when user types EXIT
   ask->user                    // ask input from user
   input <-user                 // takes the input
   parser(input)                // passes to parses
finish                          // finish


PARSER PROGRAM
count=0
For each command
   if(input < cmdStr)
      print 'what?'
     exit parser
   if(input = cmdStr)
      branch -> cmdAddr[count]
     exit parser
   if(input > cmdStr)
      count++
  if(count>#commands)
      print 'what?'
     exit parser
next command
```

Figure 2.2. Command Interpreter Algorithm

2.1.2-) Command Interpreter Assembly Code

The assembly code should be written using the algorithm above.

```
    ORG     $1000
READSTR     EQU     2
WRITESTRLF  EQU     13
WRITESTR    EQU     14
START:                          ; INITIALIZATION OF THE SYSTEM
    MOVEA.L #0,  A1
    MOVEA   #$3400,SP           ;THE STACK POINTER $3000+1K
    MOVE.L  #BUSERR, (8,A1)     ;INITIALIZATION OF EXCEPTION
    MOVE.L  #ADDRERR, (12,A1)   ;VECTORS
    MOVE.L  #ILLADRERR,(16,A1)
    MOVE.L  #DIV0ERR, (20,A1)
    MOVE.L  #CHKINSTERR,(24,A1)
    MOVE.L  #PRIVERR,(32,A1)
    MOVE.L  #LINEA,(40,A1)
    MOVE.L  #LINEF,(44,A1)
```

```
BASH:
    CLR     D0
    LEA     PROMPT,A1
    MOVE.B  #WRITESTR ,D0
    TRAP    #15                 ;PRINTS MONITOR441>
    LEA     BUFFINPUT,A1
    MOVE.B  #READSTR,D0
    TRAP    #15                 ;TAKES INPUT FROM USER
    JSR     PARSER              ;SENDS INPUT TO PARSER
    BRA     BASH                ;RETURNS TO LOOP
PARSER:
    LEA     COMP_TABL,A2        ;ARRAY OF COMMANDS NAMES -> A2
    MOVE.L  #11,D1              ;NUMBER OF FUNCTIONS INDEX ZERO
    MOVE.L  #11,D2
LOOPARSE
    JSR     CMPSTR              ;COMPARE THE INPUT WITH FIRST
    TST.L   D0                  ;IF IS SMALLER THEN ERROR
    BLT     PARSERERR           ;BECAUSE THEY ARE ORDERED
    BGT     NXFN                ;IF GREATER CAN STILL BE NEXT
    MOVE.L  D2,D0               ;IF EQUAL,
    SUB     D1,D0               ;   CALCULATE OFFSET
    LEA     COMP_ADDR,A2        ;BRING ARRAY OF ADDRESSES
    LSL     #1,D0               ;(OFFSET)*2 BECAUSE WORD ADDRESSES
    ADD     D0,A2               ;ADD OFFSET
    MOVEA   (A2),A2             ;BRING ADDRESS TO A2
    JSR     (A2)                JUMP TO COMMAND
    BRA     ENDPARSER
NXFN:
    DBEQ    D1,LOOPARSE         ;LOOP UNTIL THERE ARE NOT MORE CMDS
    JSR PARSERERR               ;IF PASS THE ARRAY THEN PARSE ERROR
    BRA ENDPARSER               ;RETURN TO BASH
PARSERERR:
    LEA     STRWHAT,A1
    MOVE.B  #WRITESTR ,D0
    TRAP #15                    ;PRINT 'WHAT?'
ENDPARSER:
    RTS
CMPSTR:
    MOVEM   A3,-(SP)
    MOVEA   A1, A3              ;SAVE START OF USER INPUT
CMPSTRNX:
    CMPM.B  (A2)+,(A3)+         ;CMP INPUT BUFFER WITH CMD STRING
    BLT     CMPSTRLESS          ;IF IS LESS RETURN TO PARSER D0 =-1
    BGT     CMPSTRGREATER       ;IF IS LESS RETURN TO PARSER D0 = 1
    TST.B   (A2)
    BNE     CMPSTRNX            ;ITERATE UNTIL END OF CMD STRING
    CLR.L   D0                  ;IF EQUAL TEXT RETURN WITH   D0 = 0
    BRA     ENDCMPSTR
```

```
CMPSTRLESS:
    MOVE.L  #-1,D0
    BRA     ENDCMPSTR
CMPSTRGREATER:
    MOVE.L  #$1,D0
    BRA     ENDCMPSTR
ENDCMPSTR:
    MOVEM   (SP)+,A3
NXCMPSTR:
    TST.B   (A2)+
    BNE     NXCMPSTR
    RTS
```

Figure 2.4. 68000 Assembly Code

2.2-) Debugger Commands

Once the commands have been selected by using the command interpreter, there can still be errors produced by either a typo or by not understanding the formats that each of the commands use for their execution. In this chapter I plan to present the implementation and the format of the instruction that the program is expecting.

2.2.1-) Debugger Command #1: BF

BF stands for bock fill, which means that this commands fills a block of memory. The data that is used to fill the memory has to be of a word size.

This command has the following format:

```
BF <Addr1> <Addr2> <W>
Example: BF $4000 $4500
```

Where Addr1 is the initial address of the block and Addr2 is the final address. Since the data given is of a word size, it is recommended that the user input even addresses. If the user does not input even addresses, the program will ignore the last bit of the address. Another important point is that the first address must be lower than the second address for obvious reasons.

2.2.1.2-) Debugger Command #1: BF Assembly Code

The assembly code should be written using the algorithm above.

```
CMDBF:
    ADD     #3,A1           ;GO TO THE START OF THE ADDRESSES
    CLR.L   D0              ;
    JSR     CAPTURE2ADDR    ;CAPTURE 2 EVEN ADDRESSES ->A2 & ->A3
    TST.B   D0
    BLT     SYNTAXERR       ;IF NOT TWO PROPER ADDRESSES ->ERROR
    LEA     BUFFADDR,A4     ;SAVE ADDRESSES TO PRINT THEM LATER
    MOVE.L  A2,(A4)+        ;SAVE STARTING ADDRESS
    MOVE.L  A3,(A4)+        ;SAVE END ADDRESS
    SUB     #8,A4           ;
    JSR     ASCI2HX         ;CONVERT THE WORD DATA TO HEX
    TST.B   D0              ;TEST IF IS A PROPER HEX
    BLT     SYNTAXERR       ;RETURN ERROR IF IS NOT
BFLOOP
    MOVE.W  D1,(A2)+        ;FILL MEMORY WITH THE WORD DATA
    CMP     A2,A3           ;IF UPPER BOUND STILL GREATER
    BGT     BFLOOP          ;THAN LOWER BOUND, LOOP
    MOVEA   A4,A1           ;MEMORY IS FILLED, NOW PRINT ADDRESS
    MOVE    #2,D0           ;TWO ADDRESSES
    JSR     PHYSICALADDR    ;PHYSICAL ADDRESS: XXXXXXXX XXXXXXXX
    RTS
```

Figure 2.7. Debugger Command # 1 Assembly Code

2.2.2-) Debugger Command # 2: BMOV

BMOV stands for Block Move, which means that this commands takes a block of memory and copy it to a destination address. In this case, the contents can be byte addressable and therefore, the addresses can be even or odd. However, the address 1 must be lower than address 2.

This command has the following format:

```
BMOV <Addr1> <Addr2> <Destination>
Example: BMOV $4000 $4500 $4700
```

Where Addr1 is the initial address of the block, Addr2 is the final address, and the destination is the address to which the block will be moved.

2.2.2.2-) Debugger Command #2: BMOV Assembly Code

```
CMDBMOV:
    ADD     #5,A1               ;GO TO THE START OF THE ADDRESSES
    MOVE    #1,D0               ;
    JSR     CAPTURE2ADDR        ;CAPTURE 2 ADDRESSES ->A2 & ->A3
    TST     D0
    BLT     SYNTAXERR           ;IF NOT TWO PROPER ADDRESSES ->ERROR
    JSR     ASCI2HX             ;CONVERT TO HEX THE DESTINATION ADDR
    TST     D0
    BLT     SYNTAXERR           ;IF DEST IS NOT A PROPER HEX ->ERROR
    LEA     BUFFADDR,A4         ;SAVE THE ADDRESSES TO PRINT AT END
    MOVE.L  A2,(A4)+            ;SAVE STARTING ADDRESS
    MOVE.L  A3,(A4)+            ;SAVE END ADDRESS
    MOVE.L  D1,(A4)+            ;SAVE DESTINATION ADDRESS
    SUB     #12,A4
    MOVEA   D1,A1               ;IN CASE DESTINATION IS IN BETWEEN
    MOVE    A3,D0               ;THE BOUNDARIES, WE HAVE TO TRANSFER
    SUB     A2,D0               ;IN DESCENDING ORDER TO NOT LOSS DATA
    SUB     A2,D1               ;OTHERWISE IS OK TO DO A ASCENDING
    BLT     BMOVASC
    CMP     D1,D0
    BLT     BMOVASC
BMOVDES:
    ADD     D0,A1               ;IF DESC, ADD OFFSET DESTINATION
BMOVDESLP:
    MOVE.B  -(A3),-(A1)         ;START FROM THE END OF THE BLOCK
    SUBQ    #1,D0
    BNE     BMOVDESLP           ;DO IT UNTIL THERE IS NOT MORE BYTES
    BRA     ENDBMOV             ;FINISH IF DONE
BMOVASC:
    MOVE.B  (A2)+,(A1)+         ;THIS IS ASCENDING ORDER
    SUBQ    #1,D0               ;DO IT UNTIL THERE IS NOT MORE BYTES
    BNE     BMOVASC             ;FINISH IF DONE
```

```
ENDBMOV:
    MOVEA    A4,A1              ;MOVE THE BEGGINING OF ADDRESSES
    MOVE     #2,D0              ;PRINT TWO BOUNDARY ADDRESSES
    JSR      PHYSICALADDR       ;PHYSICAL ADDRESS: XXXXXXXX XXXXXXXX
    MOVE     #1,D0              ;PRINT DESTINATION ADDRESS
    JSR      PHYSICALADDR       ;PHYSYCAL ADDRESS: XXXXXXXX
    RTS
```

It is similar to 2.2.1.2

2.2.3-) Debugger Command # 3: BSCH

BSCH stands for Block Search, this commands search in memory for ASCII values and returns the address at which the value was found. If the value was not found within the block, the program will just display the initial and final addresses of the block.

This command has the following format:

```
BSCH <Addr1> <addr2> <Str>
Example: BSCH $4000 $4500 'Hello world'
```

Where Addr1 is the initial address of the block, Addr2 is the final address, and the <Str> is the array of characters that the program have to find.

2.2.3.2-) Debugger Command #3 : BSCH Assembly Code

```
CMDBSCH:
    MOVE     #1,D2
    ADD      #5,A1              ;GO TO THE START OF THE ADDRESSES
    MOVE     #1,D0
    JSR      CAPTURE2ADDR       ;CAPTURE 2 ADDRESSES ->A2 & ->A3
    TST      D0
    BLT      SYNTAXERR          ;IF NOT TWO PROPER ADDRESSES ->ERROR
    LEA      BUFFADDR,A4        ;SAVE ADDRESSES TO PRINT LATER
    MOVE.L   A2,(A4)+
    MOVE.L   A3,(A4)+
    MOVE.B   (A1)+,D0           ;ERROR IF NOT STRING TO SEARCH
    BEQ      SYNTAXERR          ;
    MOVE.L   #$27,D1            ;CHECK THAT STRING START WITH QUOTES
    CMP.B    D1,D0
    BNE      SYNTAXERR          ;ERROR IF IT DOESN'T HAVE QUOTES
    MOVE.L   A1,A5              ;SAVE STARTING ADDRESS OF INPUT
    MOVE.L   A3,D0              ;SAVE ENDING ADDRESS OF SEARCH TO CMP
BSCHLOOP:
    TST      D2                 ;THE FIRST TIME IS NOT EQUAL TO ZERO
    BEQ      BSCHLOOP2          ;THIS IS JUST TO INITIALIZE
    SUB      #1,A2
BSCHLOOP2:
    CMPA     D0,A2              ;IS LOWER GREATER THAN UPPER BOUND?
    BGT      BSCHNOTFOUND       ;WE FINISH WITH NO SUCCESS
    CLR      D2                 ;ELSE MAKE D2=0
    MOVE.L   A2 ,(A4)           ;
    MOVEA.L A5,A1
```

```
BSCHLP:
    CMPM.B    (A1)+,(A2)+       ;COMP ASCI FROM INPUT TO MEMORY RANGE
    BNE       BSCHLOOP          ;IF NOT EQUAL BEGIN FROM START +1
    ADDQ      #1,D2             ;
    CMPA      D0,A2             ;IS LOWER GREATER THAN UPPER BOUND?
    BGT       BSCHNOTFOUND      ;WE FINISH WITH NO SUCCESS
    CMP.B     (A1),D1           ;IS THE NEXT CHAR IN INPUT A QUOTE?
    BEQ       BSCHFOUND         ;WE FINISH IF THIS IS THAT CASE
    TST.B     (A1)              ;IS THE NEXT CHAR IN INPUT A NULL?
    BEQ       BSCHFOUND         ;WE FINISH IF THIS IS THAT CASE
    BRA       BSCHLP            ;ELSE, REVIEW NEXT CHAR
BSCHNOTFOUND:
    LEA       BUFFADDR,A1       ;IF NOT FOUND
    MOVE      #2,D0             ;JUST PRINT THE RANGE
    JSR       PHYSICALADDR      ;PHYSICAL ADDRESS: XXXXXXXX XXXXXXXX
    RTS
BSCHFOUND:
    LEA       BUFFADDR,A1       ;IF FOUND, PRINT RANGE AND MATCH
    MOVE.L    #2,D0
    JSR       PHYSICALADDR      ;PHYSICAL ADDRESS: XXXXXXXX XXXXXXXX
    LEA       BUFFINPUT,A1
    MOVE.L    (A4),D0           ;THIS IS THE ADDRESS WITH A MATCH
    MOVE.L    #8,D1
    JSR       HX2ASCI           ;CONVERT IT IN ASCII TO PRINT
    MOVE.B    #$20,(A1)+        ;PRINT A SPACE IN FRONT
    MOVE.L    A2,D1
    MOVEA.L   (A4),A4           ;START OF THE MATCHING STRING
    MOVE.B    #$27,(A1)+
LOOPBSCH:
    MOVE.B    (A4)+,(A1)+       ;MOVE CHARS MATCHING STRING TO BUFFER
    CMPA      D1,A4             ;SEE IF THE END OF STRING
    BEQ       ENDBSCH           ;IF IT IS PRINT AND EXIT
    BRA       LOOPBSCH          ;ELSE KEEP LOOPING
    MOVE.L    (A4),D0           ;
    MOVE.L    #8,D1
    JSR       HX2ASCI           ;CONVERT THE ADDRESS TO STRING
    MOVE.B    #$20,(A1)
ENDBSCH:
    MOVE.B    #$27,(A1)+        ;ADD A QUOTE AT THE END
    MOVE.B    #$0,(A1)+
    LEA       BUFFINPUT,A1      ;PRINT IN THE FOLLOWING FORMAT
    MOVE.B    #WRITESTRLF,D0    ;XXXXXXXX 'STR'
    TRAP      #15
    RTS
```

2.2.4-) Debugger Command # 4: BTST

BTST stands for Block Test, this command is a destructive write and read test to check if the memory of a device works properly.

This command has the following format:

```
BTST <Addr1> <Addr2>
Example: BTST $7000 $9500
```

Where Addr1 is the initial address of the block, Addr2 is the end of the block. If all the blocks are properly written and read from with the expected values, the command will just print the lower and upper addresses. On the other hand, if there is a memory error, there will be an indication such as : `FAILED AT $7500 WROTE=A5A5 READ=A5B5`

2.2.4.2-) Debugger Command #4: BTST Assembly Code

```
CMDBTST:
    ADD     #5,A1               ;START FROM BEGGINNING OF ADDRESSES
    MOVE    #0,D0
    JSR     CAPTURE2ADDR        ;CAPTURE 2 EVEN ADDRESSES ->A2 &->A3
    TST.B   D0
    BLT     SYNTAXERR           ;ERROR IF NOT PROPER ADDRESSES
    MOVE.L  A3,D1               ;MOVE UPPER BOUND TO D1
    LEA     BUFFADDR,A4         ;SAVE THE ADDRESSES TO PRINT AT END
    MOVE.L  A2,(A4)+            ;SAVE LOWER BOUND
    MOVE.L  A3,(A4)+            ;SAVE UPPER BOUND
    SUB     #8,A4
    LEA     TESTBTST,A3         ;BRING THE ARRAY OF TEST WORDS
NEXTBTST:
    MOVE.W  (A3)+,D0            ;TEST WORD->D0 (E.G AAAA OR 5555)
    MOVE.W  D0,(A2)             ;TST WORD ->MEM[A2] (WRITE)
    MOVE.W  (A2),D2             ; MEM[A2] -> D2     (READ)
    CMP.W   D0,D2               ;COMPARE WRITTEN WITH READ
    BNE     BTSTERR             ;PRINT ERROR IF THEY ARE NOT SAME
    TST.W   (A2)                ;LAST WORD IS 0000, IS THIS LAST?
    BNE     NEXTBTST            ;IF NOT TEST THE OTHER WORDS
    CMP.W   A2,D1               ;IS THIS THE UPPER BOUND IN MEMORY?
    BLT     ENDBTST             ;THEN FINISH TEST
    ADD     #2,A2               ;ELSE MOVE A WORD IN MEMORY
    LEA     TESTBTST,A3         ;LOAD LIST OF WORDS IN A3
    BRA     NEXTBTST            ;TEST ALL WORDS IN NEXT LOCATION
```

```
BTSTERR:
    LEA     WROTEBTST,A1        ;SAVE IN ASCII WHAT WE WROTE IN ERR
    MOVE    #4,D1
    JSR     HX2ASCI
    LEA     READBTST ,A1        ;SAVE IN ASCII WHAT WE READ IN ERR
    MOVE    D2,D0
    MOVE    #4,D1
    JSR     HX2ASCI
    LEA     BTSTERRADDR ,A1     ;SAVE IN ASCII ADDRESS OF ERR
    MOVE.L  A2,D0
    MOVE    #8,D1
    JSR     HX2ASCI
    LEA     BTSTERRSTR ,A1      ;FAILED AT XXXXXXXX
    MOVE    #WRITESTRLF,D0      ;WROTE=XXXX READ=XXXX
    TRAP    #15                 ;(WITHOUT <CR>)
    RTS
ENDBTST:
    MOVEA   A4,A1               ;IF TEST IS PASSED
    MOVE    #2,D0               ;WE PRINT THE FOLLOWING
    JSR     PHYSICALADDR        ;PHYSICAL ADDRESS: XXXXXXXX XXXXXXXX
    RTS
```

It is similar to 2.2.1.2

2.2.5-) Debugger Command # 5: DF

This command prints the contents in the PC, SR, US, SP, D, and A registers

Example: DF

```
MONITOR441> DF
 PC=000015A6  SR=2000  US=00FF0000  SS=00003378
 D0=00000008  D1=00000007  D2=0000000B  D3=00000000
 D4=00000000  D5=00000000  D6=00000000  D7=00000000
 A0=00000000  A1=00001A10  A2=000015A6  A3=0000200F
 A4=00000000  A5=00000000  A6=00000000  A7=000033F8
```

2.2.5.2-) Debugger Command #5: DF Assembly Code

```
CMDDF:
    MOVEM.L D0-D7/A0-A7,-(SP)   ;SAVE ALL ORIGINAL REGISTERS
    MOVE.L  A7,D5               ;SAVE THE STACK POINTER IN D5
    SUBI    #64,D5              ;SUB THE REGISTERS THAT WE STORED
    MOVE.L  USP,A1              ;MOVE THE USER STACK POINTER A1
    MOVE.L  A1,D6               ;MAKE A COPY OF THE USP ->D6
    MOVE    SR,D3               ;COPY SR IN D3
    MOVE.L  #CMDDF,D4           ;PRINT
    LEA     DFSTRING,A1         ;PRINT :' PC='
    MOVE.B  #WRITESTR ,D0
    TRAP    #15
```

```
    LEA     BUFFINPUT,A1        ;CONVERT HEX TO ASCII INTO BUFFER
    MOVE    #8,D1               ;LONG REGISTER PC
    MOVE.L  D4,D0               ;CONVERTING TO ASCII
    JSR     HX2ASCI             ;
    MOVE.B  #$00,(A1)           ;PRINT PC IN NULL TERMINATED ASCII
    LEA     BUFFINPUT,A1        ;
    MOVE.B  #WRITESTR ,D0       ; BY HERE IS PC=XXXXXXXX
    TRAP    #15
    MOVE.L  #$2053523D,D2       ;PRINT :' SR='
    MOVE    #4,D1               ;WORD REGISTER SR
    MOVE.L  D3,D0               ;
    JSR     DFHELPER            ;CONV HX2ASCII AND ADD TO BUFFER
    MOVE.L  #$2055533D,D2       ;PRINT :' US='
    MOVE    #8,D1               ;LONG REGISTER US
    MOVE.L  D6,D0
    JSR     DFHELPER            ;CONV HX2ASCII AND ADD TO BUFFER
    MOVE.L  #$2053533D,D2       ;PRINT :' SS='
    MOVE.L  D5,D0               ;LONG REGISTER SS
    JSR     DFHELPER            ;CONV HX2ASCII AND ADD TO BUFFER
    LEA     BUFFINPUT,A1        ;
    MOVE.B  #$00,(A1)
    MOVE.B  #WRITESTRLF ,D0     ;PC=XXXXXXXXX SR=XXXX US=XXXXXX...
    TRAP    #15
    MOVE.L  SP,A4               ;SP ->A4
    MOVE    #4,D3               ;ITERATE FOUR TIMES
    MOVE.L  #$2044303D,D2       ;START FROM ' D0='
    BSR     DFLOOP              ;THIS PRINTS ' D0=XXXX... D3=XXXX...
    MOVE.L  #$2044343D,D2       ;START FROM ' D4='
    MOVE    #4,D3               ;ITERATE FOUR TIMES
    BSR     DFLOOP              ;THIS PRINTS ' D4=XXXX... D7=XXXX...
    MOVE.L  #$2041303D,D2
    MOVE    #4,D3               ;ITERATE FOUR TIMES
    BSR     DFLOOP              ;THIS PRINTS ' A0=XXXX... A3=XXXX...
    MOVE.L  #$2041343D,D2
    MOVE    #4,D3               ;ITERATE FOUR TIMES
    BSR     DFLOOP              ;THIS PRINTS ' A4=XXXX... A7=XXXX...
    BRA     ENDDF
DFLOOP:
    LEA     BUFFINPUT,A1
    MOVE.L  (A4)+,D0
    JSR     DFHELPER            ;WRITES IN BUFFER CONSECUTIVE REG
    ADD     #$100,D2            ;ADD 1 TO SECND CHAR (E.G 'D0'->'D1')
    SUBI    #1,D3               ;COUNT ITERATIONS
    TST     D3
    BNE     DFLOOP
    JSR     PRINTENTER          ;PRINT <CR> AT THE END OF FOUR REGS
    RTS
```

```
DFHELPER:
    LEA       BUFFINPUT,A1          ;START FROM BUFFER
    ROL.L     #8,D2                 ;MOVES THE UPPER BYTE TO LOWEST
    MOVE.B    D2,(A1)+              ;MOVE LOWEST BYTE TO BUFFER
    ROL.L     #8,D2                 ;MOVES THE UPPER BYTE TO LOWEST
    MOVE.B    D2,(A1)+              ;MOVE LOWEST BYTE TO BUFFER
    ROL.L     #8,D2                 ;MOVES THE UPPER BYTE TO LOWEST
    MOVE.B    D2,(A1)+              ;MOVE LOWEST BYTE TO BUFFER
    ROL.L     #8,D2                 ;MOVES THE UPPER BYTE TO LOWEST
    MOVE.B    D2,(A1)+              ;MOVE LOWEST BYTE TO BUFFER
    JSR       HX2ASCI               ;CONVERT HX2ASCI THE REGISTER 0
    MOVE.B    #$00,(A1)             ;IT ADDS IT TO BUFF AND NULL TERMS
    LEA       BUFFINPUT,A1
    MOVE.B    #WRITESTR ,D0         ;PRINT TOGETHER THE REG WITH NAME
    TRAP      #15
    RTS
ENDDF:
    MOVEM.L   (SP)+,D0-D7/A0-A7     ;RETURN ORIGINAL REGISTERS
    RTS
```

2.2.6-) Debugger Command # 6: EXIT

The exit command terminates the monitor program. As shown in the command interpreter, the EXIT command is the only way to legally exit the program.

2.2.6.2-) Debugger Command #6: EXIT Assembly Code

```
CMDEXIT:
    LEA       EXITSTR,A1            ;BRINGS WORD 'EXIT'
    ADD       #2,A1
    MOVE.B    #WRITESTRLF ,D0
    TRAP      #15                   ;PRINTS 'EXIT'<CR> TO TERMINAL
    BRA       END                   ;BRANCH TO END OF THE PROGRAM
    RTS
```

2.2.7-) Debugger Command # 7: GO

This command start the execution at the given address:

```
GO <Addr>
Example: GO $5100
```

2.2.7.2-) Debugger Command #7: GO Assembly Code

```
CMDGO:
    ADD       #3,A1                 ;BRINGS STARTING ADDRESS
    JSR       ASCI2HX               ;CONVERT ADDRESS TO HEX
    TST       D0                    ;ERROR IF NOT PROPER ADDRESS
    BLT       SYNTAXERR             ;
    MOVE.L    D1,A1                 ;BRING ADDRESS TO A1
    JMP       (A1)                  ;JUMP TO ADDRESS
    RTS
```

2.2.8-) Debugger Command # 8: HELP

The help command prints a summary about the manuals for each command in the monitor program.

```
MONITOR441> HELP
HELP: Display this msg
BF: Fill memory range with word data
BF <Addr1> <Addr2> <W> e.g BF 800 900 4848<CR>
BMOV: Move range of memory to location
BMOV <Addr1> <Addr2> <Dest> e.g BMOV 700 800 850<CR>
BSCH: Return first match for string in memory range
BSCH <Addr1> <addr2> <Str> e.g BSCH 800 900 'hi'<CR>
BTST: Destructive R/W test in memory range
BTST <Addr1> <Addr2> e.g BTST 800 950<CR>
DF: Display values in reg: PC,SR,US,SP,D,A
GO: Start Execution at given address
GO <target addr> e.g GO 900<CR>
MDSP: Output Address and Memory Contents
MDSP <Addr1> <Addr2> e.g MDSP 900 9D2<CR>
MM: Modify memory manually ,default size byte
MM <addr>;<sz> e.g MM 700<CR> OR MM 700;W<CR>
MS: Write bytes in memory in Hex or ASCII
MS <addr> <data> e.g MS 600 'hi'<CR> OR MS 600 35C<CR>
SORTW: Sort words in asc or desc order (A or D) in mem range
SORTW <addr1> <addr2> <ord> e.g SORTW 600 600 D<CR>
EXIT: Terminates Monitor Program
```

2.2.8.2-) Debugger Command #8: HELP Assembly Code

```
CMDHELP:
    LEA HELPDISPLAY,A1          ;BRINGS THE HELP MESSAGE
    MOVE.B  #WRITESTRLF ,D0
    TRAP #15                    ;PRINTS THE HELP MESSAGE
    RTS
```

It is similar to 2.2.1.2

2.2.9-) Debugger Command # 9: MDSP

MDSP stands for Memory Display. Using this command we can see the contents in memory from a range in the format:

```
MDSP <Addr1> <Addr2> (e.g MDSP $2000 $2020)
```

We can also display memory in the format:

```
MDSP <Addr1> (e.g MDSP $2000)
```

In this second format, the program will display by default 16 bytes of memory.

2.2.9.2-) Debugger Command #9: MDSP Assembly Code

```
CMDMDSP:
    ADD     #5,A1               ;STARTING OF THE ADDRESS(ES)
    JSR     ASCI2HX             ;CONVERT FIRST ADDRESS TO HEX
    TST     D0
    BLT     SYNTAXERR           ;ERROR IF NOT PROPER HEX
    MOVEA   D1,A2               ;MOVE LOWER BOUND TO A2
    TST.B   (A1)                ;IS THERE ANOTHER ADDRESS?
    BEQ     MDSP1ND             ;IF THERE IS NOT, DO DEFAULT
    JSR     ASCI2HX             ;CONVERT SECOND ADDRESS TO HEX
    TST     D0
    BLT     SYNTAXERR           ;ERROR IF NOT PROPER HEX
    MOVEA   D1,A3               ;MOVE UPPER BOUND TO A2
    CMP     A3,A2
    BGT     SYNTAXERR           ;ERROR IF LOWER>UPPER ADDRESSES
    BRA     MDSPLOOP
MDSP1ND
    MOVEA   A2,A3               ;DEFAULT WILL SHOW $F ADDRESSES
    ADD     #15,A3              ;MAKE UPPER BOUND A2+$F ->A3
MDSPLOOP
    LEA     BUFFINPUT ,A1       ;BUFFER TO PRINT
    MOVE    A2, D0              ;BRING ADDRESS TO D0 TO PRINT
    MOVE    #8,D1
    JSR     HX2ASCI             ;CONVERT ADDRESS TO ASCII
    MOVE.B  #$20,(A1)+          ;PLACE TWO SPACES BEFORE CONTENT
    MOVE.B  #$20,(A1)+
    MOVE.B  (A2)+,D0            ;BRING CONTENT TO D0
    MOVE    #2,D1
    JSR     HX2ASCI             ;CONVERT TO ASCII BYTE SIZE
    MOVE.B  #0,(A1)
    LEA     BUFFINPUT,A1
    MOVE.B  #WRITESTRLF ,D0     ;PRINT XXXXXXXXX  XX
    TRAP    #15
    CMP     A2,A3               ;LOWER <UPPER? KEEP GOING
    BGT     MDSPLOOP            ;ELSE, END
    RTS
```

2.2.10-) Debugger Command # 10: MM

MM stands for Memory Modify, this command displays the memory and receives an input to modify the memory that is being display.

There are three different formats:

```
MM <Addr1> (E.G MM $2000)
MM <Addr1> ;W (E.G MM $2000 ;W)
MM <Addr1> ;L (E.G MM $2000 ;L)
```

As shown in the examples, only the Word and Long types have to be explicitly passed as an argument while the byte size is default. In order to finish displaying and entering values into memory, the user can type a dot within the terminal ".". Or in case that the user wants to look at the next memory location without altering the current memory location, the user can press enter l

2.2.10.2-) Debugger Command #10: MM Assembly Code

```
CMDMM:
    ADD     #3,A1               ;START AT THE BEGGINING OF ADDRESS
    JSR     ASCI2HX             ;ADDRESS TO HEX
    TST     D0
    BLT     SYNTAXERR           ;ERROR IF NOT PROPER ADDRESS
    TST.B   (A1)                ;IF NOT OTHER ARG
    BEQ     MMBYTE              ;THEN MUST BE BYTE BY DEFAULT
    AND.B   #$FE,D1             ;ELSE, MAKE SURE IS AN EVEN ADDRESS
    MOVE.L  D1,A2               ;ADDRESS GIVEN ->A2
    CMP.W  #$3B57,(A1)          ;IS NEXT ARGUMENT ':W'??
    BEQ     MMWORD              ;THEN GO TO THE WORD ROUTINE
    CMP.W  #$3B4C,(A1)          ;IS NEXT ARGUMENT ':L'??
    BEQ     MMLONG              ;THEN GO TO THE LONG ROUTINE
    BRA     SYNTAXERR           ;ERROR IF NEITHER OF THOSE ARGS
MMBYTE:
    MOVEA   D1,A2
    MOVE    #2,D2               ;SIZE TO WORK WITH IS 2 CHARS (BYTE)
    BRA     MMLOOP              ;GO TO LOOP
MMWORD
    MOVE    #4,D2               ;SIZE TO WORK WITH IS 4 CHARS (WORD)
    BRA     MMLOOP              ;GO TO LOOP
MMLONG:
    MOVE    #8,D2               ;SIZE TO WORK WITH IS 8 CHARS (LONG)
MMLOOP:
    LEA     BUFFINPUT ,A1       ;INPUT BUFFER
    MOVE.L  A2, D0
    MOVE    #8,D1
    JSR     HX2ASCI             ;MOVE CURRENT ADDRESS IN HEX
    MOVE.B  #$20,(A1)+          ;PLUS SPACE
    CMP.B   #2,D2               ;IS THE BYTE?
    BEQ     MMBYTPR             ;    PRINT A BYTE
    CMP.B   #4,D2               ;    IS THE WORD?
    BEQ     MMWRDPR             ;        PRINT A WORD
    BRA     MMLNGPR             ;ELSE LONG
MMBYTPR:
    MOVE.B  (A2),D0             ;MOVE FOR CONVERSION SIZE DATA BYTE
    BRA     MMCONT
MMWRDPR:
    MOVE.W  (A2),D0             ;MOVE FOR CONVERSION SIZE DATA WORD
    BRA     MMCONT
MMLNGPR:
    MOVE.L  (A2),D0             ;MOVE FOR CONVERSION SIZE DATA LONG
```

```
MMCONT:
    MOVE.L  D2,D1               ;MOVE THE SIZE FOR HEX CONVERSION
    JSR     HX2ASCI             ;CONVERT
    SUB     #1,A1
    MOVE.L  #$203F2000,(A1)+    ;ADD ' ? ' AT THE END OF ADDRESS
    LEA     BUFFINPUT,A1
    MOVE.B  #WRITESTR,D0
    TRAP    #15                 ;PRINT 'XXXXXXXX ? '
    LEA     BUFFINPUT,A1
    MOVE.B  #READSTR,D0
    TRAP    #15                 ;READ ANSWER FROM USER
    LEA     BUFFINPUT,A1        ;
    MOVE.B  (A1),D0             ;MOVE THE FIRST BYTE OF THE ANSWER
    CMP.B   #$2E,D0             ;IF IS A DOT "." END THE COMMAND
    BEQ     ENDMM               ;
    TST.B   D0
    BEQ     MMPASS              ;IF EMPTY MEANS THAT PASS W/O CHANGE
    JSR     ASCI2HX             ;ELSE, CONVERT TO HEX TO SAVE CHANGES
    TST     D0
    BLT     SYNTAXERR           ;IF NOT A PROPER HEX PRINT ERR
    CMP.B   #2,D2
    BEQ     MMMVBYT             ;SAVE BYTE
    CMP.B   #4,D2
    BEQ     MMMVWRD             ;SAVE WORD
    BRA     MMMVLNG             ;SAVE LONG
MMMVBYT
    MOVE.B  D1,(A2)+            ;SAVE BYTE
    BRA     MMLOOP              ;KEEP GOING UNTIL A DOT "."
MMMVWRD
    MOVE.W  D1,(A2)+            ;SAVE WORD
    BRA     MMLOOP              ;KEEP GOING UNTIL A DOT "."
MMMVLNG
    MOVE.L  D1,(A2)+            ;SAVE LONG
    BRA     MMLOOP              ;KEEP GOING UNTIL A DOT "."
MMPASS:
    MOVE    D2,D3               ;IF PASSED, ADD TO THE ADDRESS
    ASR     #1,D3               ;A2+1 OR +2 OR +4 DEPENDING OF D2
    ADD     D3,A2
    BRA     MMLOOP              ;LOOP
ENDMM
    RTS
```

2.2.11-) Debugger Command # 11: MS

MS stands for Memory Set. This command can set up to 4 bytes of an hexadecimal number into a memory location or it can move up to 40 characters into the memory location given as an argument.

There are two different formats for this instruction:

```
MS <Addr1> $xxxxxxxx (E.G MS $2000 $324EE)
MS <Addr1> <Str> (E.G MS $2000 'HELLO WORLD')
```

2.2.11.2-) Debugger Command #11: MS Assembly Code

```
CMDMS:
    ADD     #3,A1               ;STARTING ADDRESS
    JSR     ASCI2HX             ;CONVERT FIRST ADDRESS TO HEX
    TST     D0
    BLT     SYNTAXERR           ;ERROR IF NOT A PROPER ADDRESS
    SUB     #1,A1
    TST.B   (A1)+
    BEQ     SYNTAXERR           ;IF NOT INPUT HEX OR ASCI IS ERR
    MOVEA   D1, A2
    SUBI.B  #$27,(A1)+          ;IF INPUT START WITH QUOTES IS ASCII
    BEQ     MSASCII             ;BRANCH TO ASCII
    ADDI.B  #$27,-(A1)
    CMP.B   #$24,(A1)           ;IF HEX INPUT W/O '$' TAKE INPUT
    BNE     MSHEXNOTDOL         ;INMEDIATELY
    ADD     #1,A1               ;OTHERWISE ADD 1 BYTE AND TAKE INPUT
MSHEXNOTDOL:
    MOVE    A1,A3               ;
MSHEX:
    JSR     ASCI2HX             ;CONVERT THE INPUT TO HEX
    TST.B   D0
    BLT     SYNTAXERR           ;IF NOT A PROPER HEX PRINT ERR
    ADDQ    #1,D0               ;ADD ONE AND DIVIDE BY TWO TO GET
    LSR     #1,D0               ;ROOF OF # OF PAIR CHARACTERS
    MOVE    D0,D2               ;SINCE 2 NIBBLES PER BYTE
    SUB     #4,D2
MSPOSSITION:
    TST     D2                  ;ROTATE MSB TO THE LSB POSITION
    BEQ     MSLOOPHX            ;IN ORDER TO WORK BYTE PER BYTE
    ROL.L   #8,D1
    ADDQ    #1,D2
    BRA     MSPOSSITION
MSLOOPHX:
    ROL.L   #8,D1               ;WRITE EACH HEX BYTE INTO THE ADDR
    MOVE.B  D1,(A2)+            ;GIVEN BY A2
    SUBI.B  #1,D0               ;UNTIL THE NUMBER OF PAIRS IS ZERO
    BNE     MSLOOPHX            ;LOOP
    TST.B   -(A1)               ;END IF THERE IS A NULL TERMINATION
    BEQ     ENDMS               ;IN THE USER INPUT.
    ADD     #1,A1
    BRA     MSHEX
```

```
MSASCII:
    MOVE.B  #$27,-(A1)          ;ASCII IS EASIER, IS LAST CHAR A "'"?
    ADD     #1,A1               ;
MSASCIILOOP:
    TST.B   (A1)                ;IF CHAR IS NULL, STRING TERMINATED
    BEQ     ENDMS
    MOVE.B  (A1),D0
    CMP.B   #$27, D0            ;IF CHAR IS "'", STRING TERMINATED
    BEQ     ENDMS
    MOVE.B  (A1)+,(A2)+         ;COPY FROM INPUT TO MEMORY
    BRA     MSASCIILOOP
ENDMS
    RTS
```

2.2.12-) Debugger Command #12: SORTW

The SORTW command sorts the words of a memory block in either ascending or descending order. The default is an ascending order, therefore if an argument is omitted, the ascending order will be executed.

There are three different formats for this instruction but only two modes:

```
SORTW <addr1> <addr2> <ord> (e.g SORTW $200 $500 A)
SORTW <addr1> <addr2> <ord> (e.g SORTW $200 $500 D)
 SORTW <addr1> <addr2>       (e.g SORTW $200 $500)
```

2.2.12.2-) Debugger Command #12: SORTW Assembly Code

```
CMDSORTW:
    ADD     #6,A1               ;BEGGINING OF ADDRESSES
    MOVE    #0,D0
    JSR     CAPTURE2ADDR        ;GET TWO EVEN ADDRESSES A2 & A3
    TST     D0
    BLT     SYNTAXERR           ;ERROR IF NOT PROPER ADDRESS
    MOVE.L  #-1,D1              ;IS DESCENDENT D1=-1
    CMP.B   #$44,(A1)           ;IF ARGUMENT IS 'D' DESCENDENT
    BNE     SORTWDESC
    CLR.L   D1                  ;IS ASCENDENT D1=0
SORTWDESC:                      ;THIS IS PRETTY MUCH BUBBLE SORT
    MOVE.L  A2,A0
    MOVE.L  A0,A2               ; A2 IS THE STARTING ADDRESS
SORTBGNN:
    MOVE.L  A2,A0               ; START FROM BEGGINING AGAIN
SORTCMP:
    CMP.W   (A0)+,(A0)+         ; IF (ARRAY[i] < ARRAY[i+1])
    BHI.S   SWAPSORT            ; THEN SWAP THE VALUES
    SUBQ.L  #2,A0               ; ELSE: GO TO THE NEXT ELEMENT
    CMP.L   A0,A3               ; IF CURRENT ELEMENT == LAST?
    BNE     SORTCMP             ; IF NOT, GO TO THE NEXT ELEMENT
    TST     D1                  ; IF THEY ARE EQUAL, YOU HAVE
    BEQ     ENDSORT             ; ALL SORTED IN DESCENDING
    ADD     #2,A3               ; IF WANT ASCENDING SWAP WHOLE ARRAY
```

```
SORTASC:
    MOVE.W   (A2),D1              ;COPY FIRST VALUE IN D1
    MOVE.W   -(A3),(A2)+          ;COPY LAST VALUE IN FIRST POSITION
    MOVE.W   D1,(A3)              ;COPY FIRST VALUE IN LAST POSITION
    CMP      A2,A3                ;DO IT UNTIL LOWER>UPPER BOUND
    BGT      SORTASC
    BRA      ENDSORT              ;WHEN THIS IS DONE YOU FINISH!
SWAPSORT:
    MOVE.L   -(A0),D0             ; PUT 2 WORDS IN D0
    SWAP.W   D0                   ; SWAP WORDS D0
    MOVE.L   D0,(A0)              ; BRING IT BACK TO MEMORY
    BRA      SORTBGNN             ; START FROM BEGGINING COMPARISON
ENDSORT

    RTS
```

2.3-) Exception Handlers

In this implementation, we handle the following exceptions: Bus Error Exception, Address Error exception, Illegal Instruction Exception, Privilege Violation Exception, Divide by Zero Exception, Check Instruction Exception, and LineA and LineF special exceptions.

All exceptions restart the bash program, so further implementation for the special functions LineA and LineF should be implemented when the vectors are created by the developers that want to use them.

2.3.1-) Bus Error Exception

This exception occurs when the program tries to access a device at an inexistent address. In this circumstances, the watchdog timer would produce this exception. The exception shows the SSW,BA,IR as well as all the registers shown in with the DF command.

All exceptions with exception of Bus Error and Address Error have the same type of handler that is why I will not repeat the explanation. These handlers show the PC at which an instruction produced an error and print it together with the type of error.

2.3.1.2-) Bus Error Exception Assembly Code

```
BUSERR:

    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     SSWINTSTR,A1        ;BUFFER TO SAVE THE ASCII OF THE SSW
    MOVE.W  (12,SP),D0          ;GET SSW FROM STACK FRAME
    MOVE.W  #4,D1               ;CONVERT 4 DIGITS (WORD) TO ASCII
    JSR     HX2ASCI             ;ASCII CONVERTION
    LEA     BASTR,A1            ;BUFFER TO SAVE THE ASCII OF THE BA
    MOVE.L  (14,SP),D0          ;GET BUS ADDRESS FROM STACK FRAME
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT 8 DIG (LONG) BA TO ASCII
    LEA     IRSTR,A1            ;BUFFER TO SAVE THE ASCII OF THE IR
    MOVE.L  (18,SP),D0          ;GET IR FROM STACK FRAME
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT 8 DIG (LONG) BA TO ASCII
    LEA     BUSERRSTR,A1        ;
    MOVE.B  #WRITESTR,D0
    TRAP    #15                 ;PRINT BUS ERROR STRING
    LEA     SSWSTR,A1
    MOVE.B  #WRITESTRLF,D0      ;PRINT SSW,BA,IR
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.2-) Address Error Exception

This exception occurs when the program tries to access a word or longword address at an odd location. The exception shows the SSW,BA,IR as well as all the registers shown in with the

DF command. In this way, the user can check in the BA the specific faulty instruction that produced the exception and correct it.

2.3.1.2-) Address Error Exception Assembly Code

```
ADDRERR:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     SSWINTSTR,A1        ;BUFFER TO SAVE THE ASCII OF THE SSW
    MOVE.W  (12,SP),D0          ;GET SSW FROM STACK FRAME
    MOVE.W  #4,D1               ;CONVERT 4 DIGITS (WORD) TO ASCII
    JSR     HX2ASCI             ;ASCII CONVERTION
    LEA     BASTR,A1            ;BUFFER TO SAVE THE ASCII OF THE BA
    MOVE.L  (14,SP),D0          ;GET BUS ADDRESS FROM STACK FRAME
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT 8 DIG (LONG) BA TO ASCII
    LEA     IRSTR,A1            ;BUFFER TO SAVE THE ASCII OF THE IR
    MOVE.L  (18,SP),D0          ;GET IR FROM STACK FRAME
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT 8 DIG (LONG) BA TO ASCII
    LEA     ADDRERRSTR,A1        ;
    MOVE.B  #WRITESTR,D0
    TRAP    #15                 ;PRINT ADDRESS ERROR STRING
    LEA     SSWSTR,A1
    MOVE.B  #WRITESTRLF,D0      ;PRINT SSW,BA,IR
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.3)Illegal Instruction Exception Assembly Code

```
ILLADRERR:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     ILLERRSTR ,A1       ;PRINT ILLEGAL INSTRUCTION ERROR STR
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.4) Privilege Violation Exception Assembly Code

```
PRIVERR:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     PRVERRSTR ,A1       ;PRINT PRIVILEGE VIOLATION ERROR STR
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.5) Divide by Zero Exception Assembly Code

```
DIV0ERR:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     DIV0ERRSTR,A1       ;PRINT DIV0  ERROR STR
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.6.2-) Line A and Line F Emulators Assembly Code

```
LINEA:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     LINEASRT ,A1        ;PRINT LINEA EMULATION EXCEP
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

```
LINEF:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     LINEFSRT ,A1        ;PRINT LINEF EMULATION EXCEP
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.3.6.2-) Check Instruction Exception Assembly Code

```
CHKINSTERR:
    MOVEM.L D0-D1/A1,-(SP)      ;SAVE ALL REGISTERS IN THE SSP
    LEA     IRSTR,A1            ;GET BUS ADDRESS FROM STACK FRAM
    MOVE.L  (14,SP),D0
    MOVE.W  #8,D1
    JSR     HX2ASCI             ;CONVERT BA TO ASCII
    LEA     CHKERRSTR ,A1       ;PRINT CHK INSTRUCTION ERROR STR
    MOVE.B  #WRITESTR,D0
    TRAP    #15
    LEA     IRSTR,A1            ;PRINT PC LOCATION OF THE ERROR
    MOVE.B  #WRITESTRLF,D0
    TRAP    #15
    MOVEM.L (SP)+,D0-D1/A1      ;RECOVER STACK
    JSR     CMDDF               ;GET DF OUTPUT
    BRA     START               ;RESTART PROGRAM
```

2.4.1-) Help Menu

```
MONITOR441> HELP
HELP: Display this msg
BF: Fill memory range with word data
BF <Addr1> <Addr2> <W> e.g BF 800 900 4848<CR>
BMOV: Move range of memory to location
BMOV <Addr1> <Addr2> <Dest> e.g BMOV 700 800 850<CR>
BSCH: Return first match for string in memory range
BSCH <Addr1> <addr2> <Str> e.g BSCH 800 900 'hi'<CR>
BTST: Destructive R/W test in memory range
BTST <Addr1> <Addr2> e.g BTST 800 950<CR>
DF: Display values in reg: PC,SR,US,SP,D,A
GO: Start Execution at given address
GO <target addr> e.g GO 900<CR>
MDSP: Output Address and Memory Contents
MDSP <Addr1> <Addr2> e.g MDSP 900 9D2<CR>
MM: Modify memory manually ,default size byte
MM <addr>;<sz> e.g MM 700<CR> OR MM 700;W<CR>
MS: Write bytes in memory in Hex or ASCII
MS <addr> <data> e.g MS 600 'hi'<CR> OR MS 600 35C<CR>
SORTW: Sort words in asc or desc order (A or D) in mem range
SORTW <addr1> <addr2> <ord> e.g SORTW 600 600 D<CR>
EXIT: Terminates Monitor Program
```

3-) Discussion

I have learn many things from this project, but I could have learn far more if the organization of the class (and labs) allowed students to have more time to be creative rather than getting the program done to survive in the class. I see that in this types of projects, there is a really interesting kind of ideas, ideas about the big picture in system design that we rarely have the time to deal with.

4-) Feature Suggestions

A good idea to explore would be to use the LineA and LineF emulators exceptions to perform the commands that we are using in this monitor project in order to make the user work in user mode. When the user wants to call a system call through the exception, the supervisor mode will be activated and it would be encapsulated from the user. It would also be interesting the implementation of this code using different data and programming spaces using the Check exception to create a robust program and maybe a simple OS.