Illinois Institute of Technology

ECE 441 Monitor Project

Author:
Adam Sumner

Teaching Assistant:
Boyang Wang

April 28th, 2015

Acknowledgment

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

Contents

List of Figures

Abstract

This project involved designing and implementing a Monitor program using the MC68000 assembly language. The program implements twelve basic debugger functions as well as two author defined functions. It is designed to handle exceptions, and is meant to be an educational piece of software for students taking ECE 441 at the Illinois Institute of Technology.

1 Introduction

The SANPER-1 ELU is a Motorola MC68000 based microcomputer designed by Dr. Jafar Saniie and Mr. Stephen Perich for use in college level computer engineering courses[2]. For user interaction, it utilizes a monitor program called TUTOR that enables users to actively interact with the microcomputer. The design objective of this project is to re-implement the functionality of TUTOR into a student written monitor program titled MONITOR441. The program should be able to perform basic debugger functions such as memory display, memory sort, memory change, etc., and must have the ability to handle exceptions. The design constraints are:

- Code must be smaller that 3K starting from address $1000
- Stack size must be 1K starting at memory location $3000
- Macros may not be used
- Erroneous inputs should not kill the program

Twelve debugger functions must be implemented, along with two user defined debugger commands.

2 Monitor Program

The monitor program operates in a command driven environment. It acts as a typical shell, providing a user interface to access the microcomputer's services. The main program being run is a command line interpreter. Based on the input that the user enters, the interpreter determines if the input entered is valid and subsequently executes the specified command. It was

developed using the Easy68K Simulator, thus the TRAP #15 handler is used instead of the MC68000's TRAP #14 handler. The structure of how this program operates is shown in Figure 1.

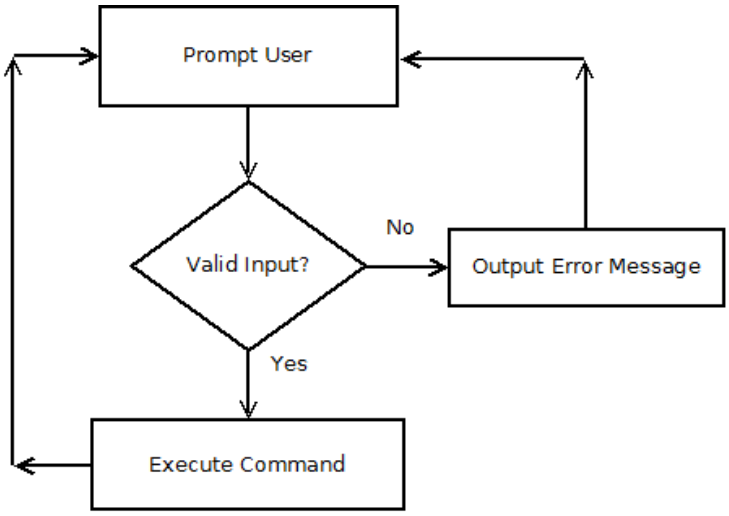

Figure 1: Structure of Monitor Program

2.1 Command Interpreter

2.1.1 Algorithm and Flowchart

The algorithm for the command interpreter uses simple string matching to determine if input is correct. The algorithm begins by outputting the message `MONITOR441>` and accepting input from the user. It then checks for the ASCII value $48 which corresponds to the letter H. This is to check for either the `HELP` command or `HXDC` command. If an H was not entered, it then checks for the ASCII value $4D which corresponds to a memory command. If this fails, then it checks for ASCII value $47, corresponding to the `GO` command. If this fails, the ASCII value $44 is tested, corresponding to the `DF` command. If this fails, it checks for $42, which signifies a `BLCK` command. If this fails, $53 is tested for the `SORTW` command. If this fails, $45 is tested for the `ECHO` command. If this fails $2E is checked for the modify register command. If all of these checks fail, the user has entered incorrect input and an error message is displayed. If any of these checks succeed, the command line interpreter jumps to the respective command's helper interpreter function.

These subroutines check for each character of the user input in order to verify the command the user entered was correct. These helper functions also serve to differentiate commands that start with the same character. The flowchart for this process is shown in Figure 2.

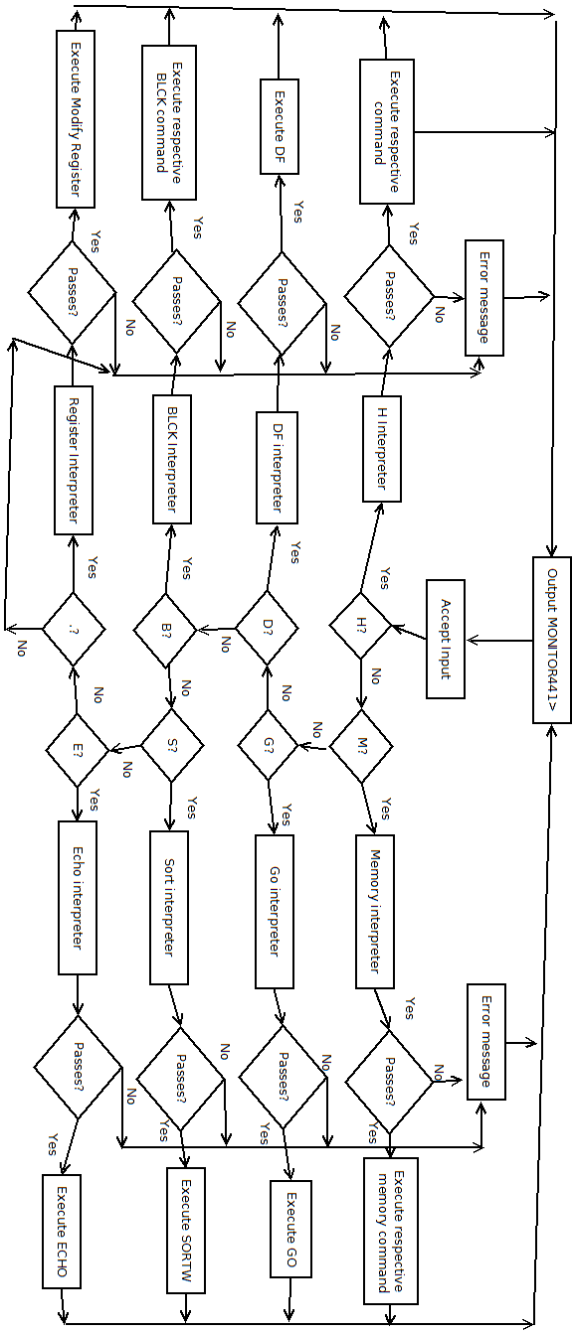

Figure 2: Flowchart for Command Line Interpreter

2.1.2 Assembly Code

```
154 SHELL:
155             PEA     *               ;save PC on Stack for DF
156             ADD.L   #4,SP           ;get original value of stack
        pointer
157             MOVE.L  SP,-8(SP)       ;save it
158             ADD.L   #-8,SP          ;update Stack position
159             MOVE    SR,-(SP)        ;save Status register for use
        with DF
160             MOVE.L  A6,-(SP)        ;temp save
161
162             MOVE    USP,A6          ;for use with DF command
163             MOVE.L  A6,-(SP)        ;store USP to STACK
164             ADD.L   #4,SP
165             MOVE.L  (SP),A6      ;restore original value
166             MOVE.L  -(SP),4(SP)   ;move correct value to correct
        stack position
167             ADD.L   #4,SP           ;point stack to CORRECT PLACE
168
169
170             MOVEM.L D0-D7/A0-A6,-(SP)   ;save initial values of
        registers
171             MOVEM.L D0-D7/A0-A6,-(SP)   ;unorthodox
        implementation to save registers when using DF command
172
173
174             LEA   PROMPT,A1         ;Load message
175             MOVE.W  #11,D1          ;load n bytes
176             MOVE.B  #1,D0           ;set up trap call
177             TRAP    #15
178             LEA     BUFFER,A1       ;set up storage for command
179             MOVE.B  #2,D0           ;load input trap call
180             TRAP    #15
181             CMP.B   #$48,(A1)       ;check for help/hxdc
182             BEQ     HELPORHXDC
183             CMP.B   #$4D,(A1)       ;check for memory command
184             BEQ     MEMTEST
185             CMP.B   #$47,(A1)       ;check for go
186             BEQ     GOTST
187             CMP.B   #$44,(A1)       ;check for df
188             BEQ     DFTST
189             CMP.B   #$42,(A1)       ;check for blck command
190             BEQ     BLCKTEST
191             CMP.B   #$53,(A1)       ;check for sort command
```

```
192                 BEQ       SORTTEST
193                 CMP.B     #$45,(A1)     ;check for echo command
194                 BEQ       ECHOTEST
195                 CMP.B     #$2E,(A1)     ;check for modify register
        command
196                 BEQ       MODIFYREGTEST
197                 BRA       UNKNOWNCMD
198 RESTORE:        MOVEM.L   (SP)+,D0-D7/A0-A6
199                 MOVEM.L   (SP)+,D0-D7/A0-A6 ;double restore because of
         DF hack workaround
200                 ADD.L     #4,SP         ;account for USP, it'll fix
        itself (it shouldn't be used)
201                                         ;EASY68k simulator starts in
        supervisor mode
202                 MOVE      (SP)+,SR
203                 MOVE.L    (SP)+,D0      ;save stack cuz it'll get
        destroyed
204                 ADD.L     #4,SP         ;get rid of PC, itll fix itself
205                 MOVE.L    D0,SP
206                 CLR.L     D0            ;no longer needed
207
208                 BRA       SHELL
209 *-------------------------------------------------------------------------------

210
211 ECHOTEST:       ADD.L     #1,A1
212                 CMP.B     #$43,(A1)+    ;C?
213                 BNE       UNKNOWNCMD
214                 CMP.B     #$48,(A1)+    ;H?
215                 BNE       UNKNOWNCMD
216                 CMP.B     #$4F,(A1)+    ;O?
217                 BNE       UNKNOWNCMD
218                 CMP.B     #$20,(A1)+    ;SPACE?
219                 BEQ       ECHO
220                 BRA       ERRORSR
221 *-------------------------------------------------------------------------------

222
223
224 *-------------------------------------------------------------------------------

225
```

```
226 BLCKTEST:   ADD.L   #1,A1
227             CMP.B   #$46,(A1)       ;BF?
228             BEQ     BFTEST
229             CMP.B   #$4D,(A1)       ;BMOV?
230             BEQ     BMOVTEST
231             CMP.B   #$54,(A1)       ;BTST?
232             BEQ     BTSTTEST
233             CMP.B   #$53,(A1)       ;BSCH?
234             BEQ     BSCHTEST
235             BRA     UNKNOWNCMD
236 *-----------------------------------------------------------------------------

237
238 BSCHTEST:   ADD.L   #1,A1
239             CMP.B   #$43,(A1)
240             BNE     UNKNOWNCMD
241             ADD.L   #1,A1
242             CMP.B   #$48,(A1)
243             BNE     UNKNOWNCMD
244             ADD.L   #1,A1
245             CMP.B   #$20,(A1)
246             BNE     ERRORSR
247             BRA     BSCH
248
249 *-----------------------------------------------------------------------------

250
251 BTSTTEST:
252             ADD.L   #1,A1
253             CMP.B   #$53,(A1)
254             BNE     UNKNOWNCMD
255             ADD.L   #1,A1
256             CMP.B   #$54,(A1)
257             BNE     UNKNOWNCMD
258             ADD.L   #1,A1
259             CMP.B   #$20,(A1)
260             BNE     ERRORSR
261             BRA     BTST
262
263 *-----------------------------------------------------------------------------

264
```

```
265 BMOVTEST:   ADD.L   #1,A1
266             CMP.B   #$4F,(A1)
267             BNE     UNKNOWNCMD
268             ADD.L   #1,A1
269             CMP.B   #$56,(A1)
270             BNE     UNKNOWNCMD
271             ADD.L   #1,A1
272             CMP.B   #$20,(A1)
273             BNE     ERRORSR
274             BRA     BMOV
275 *           ------------------------------------------------------------------------------

276 BFTEST:     ADD.L   #1,A1
277             CMP.B   #$20,(A1)
278             BNE     ERRORSR
279             BRA     BF
280 *           ------------------------------------------------------------------------------

281
282 DFTST:      ADD.L   #1,A1
283             CMP.B   #$46,(A1)
284             BNE     UNKNOWNCMD
285             ADD.L   #1,A1
286             CMP.B   #$00,(A1)
287             BNE     ERRORSR
288             BRA     DF
289 *           ------------------------------------------------------------------------------

290
291 SORTTEST:    ADD.L   #1,A1
292             CMP.B   #$4F,(A1)     ;O?
293             BNE     UNKNOWNCMD
294              ADD.L   #1,A1
295             CMP.B   #$52,(A1)     ;R?
296             BNE     UNKNOWNCMD
297             ADD.L   #1,A1
298             CMP.B   #$54,(A1)     ;T?
299             BNE     UNKNOWNCMD
300             ADD.L   #1,A1
301             CMP.B   #$57,(A1)     ;W?
302             BNE     UNKNOWNCMD
303             ADD.L   #1,A1
```

```
304                 CMP.B   #$20,(A1)
305                 BNE     ERRORSR
306
307                 BRA     SORTW
308 *---------------------------------------------------------------------------

309
310 GOTST:         ADD.L   #1,A1
311                 CMP.B   #$4F,(A1)
312                 BNE     UNKNOWNCMD
313                 ADD.L   #1,A1
314                 CMP.B   #$20,(A1)+
315                 BNE     ERRORSR
316                 BRA     GO
317 *---------------------------------------------------------------------------

318
319 HELPORHXDC:    ADD.L   #1,A1
320                 CMP.B   #$45,(A1)    ;is it help?
321                 BEQ     HELPTST
322                 CMP.B   #$58,(A1)    ;or is it hxdc
323                 BEQ     HXDCTEST
324                 BRA     UNKNOWNCMD
325 *---------------------------------------------------------------------------

326
327 HELPTST:
328                 ADD.L   #1,A1    ; check next char
329                 CMP.B   #$4C,(A1) ;check for L
330                 BNE     UNKNOWNCMD
331                 ADD.L   #1,A1
332                 CMP.B   #$50,(A1)    ;check for P
333                 BNE     UNKNOWNCMD
334                 ADD.L   #1,A1    ;check for anything else
335                 CMP.B   #$00,(A1)
336                 BNE     ERRORSR
337                 BRA     HELP
338
339
340
341 *---------------------------------------------------------------------------
```

```
342
343 MEMTEST:    ADD.L    #1,A1
344             CMP.B    #$53,(A1)
345             BEQ      MSSPCTEST
346             CMP.B    #$44,(A1)
347             BEQ      MDSPCTEST
348             CMP.B    #$4D,(A1)
349             BEQ      MMSPCTEST
350             BRA      UNKNOWNCMD
351
352 MSSPCTEST   ADD.L    #1,A1
353             CMP.B    #$20,(A1)
354             BEQ      MEMSET
355             BRA      ERRORSR
356
357 MDSPCTEST:
358             ADD.L    #1,A1
359             CMP.B    #$53,(A1)
360             BNE      ERRORSR
361             ADD.L    #1,A1
362             CMP.B    #$50,(A1)
363             BNE      UNKNOWNCMD
364             ADD.L    #1,A1
365             CMP.B    #$20,(A1)
366             BEQ      MEMDISP
367             BRA      ERRORSR
368
369 MMSPCTEST:  ADD.L    #1,A1
370             CMP.B    #$20,(A1)
371             BEQ      MM
372             BRA      ERRORSR
373 *           ------------------------------------------------------------------------------

374 HXDCTEST:
375             ADD.L    #1,A1
376             CMP.B    #$44,(A1)
377             BNE      UNKNOWNCMD
378             ADD.L    #1,A1
379             CMP.B    #$45,(A1)
380             BNE      UNKNOWNCMD
381             ADD.L    #1,A1
382             CMP.B    #$43,(A1)
383             BNE      UNKNOWNCMD
```

```
384                 ADD.L    #1,A1
385                 CMP.B    #$20,(A1)
386                 BNE      ERRORSR
387                 BRA      HXDC
388 *-------------------------------------------------------------------------------

389 MODIFYREGTEST:
390                 ADD.L    #1,A1
391                 CMP.B    #$44,(A1)
392                 BEQ      MRD
393                 CMP.B    #$41,(A1)
394                 BEQ      MRA
395                 BRA      UNKNOWNCMD
396
397 *-----------------------------------USER DEFINED COMMANDS
    ------------------------------------*
398 *-------------------------------------------------------------------------------

399 ECHO: *What terminal DOESN'T have echo?*
400
401           MOVE.L   A1,A2    ;setup to find end of string
402 EEND:     CMP.B    #$00,(A2)+
403           BEQ      EFOUND
404           BRA      EEND
405 EFOUND:
406           SUB.L    #1,A2    ;off by one
407           SUB.L    A1,A2    ;find out how many bytes
408           MOVE.L   A2,D1    ;place it for trap function
409           MOVE.L   #0,D0
410           TRAP     #15
411
412           BRA RESTORE
```

2.2 Debugger Commands

2.2.1 Help

2.2.1.1 Algorithm and Flowchart

Help is a simple command that prints out a series of strings that display the available commands, their syntax, and a short description of each command. The syntax to invoke this command is `HELP`. The flowchart for this command is shown in Figure 3.

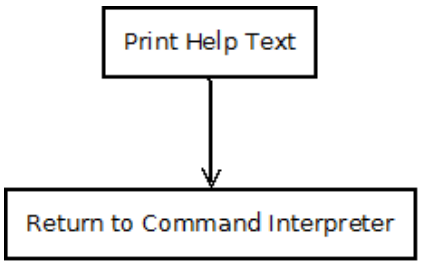

Figure 3: Flowchart for Help

2.2.1.2 Assembly Code

```
797 HELP:       LEA       HTXT,A1       ;list of commands test
798             MOVE.W    #17,D1
799             MOVE.B    #0,D0
800             TRAP      #15
801             MOVE.W    #0,D1         ;newline
802             TRAP      #15
803
804             LEA       HTXT1,A1      ;mem display command
805             MOVE.W    #75,D1
806             MOVE.B    #0,D0
807             TRAP      #15
808             LEA       HTXT1A,A1     ;mem display
809             MOVE.W    #61,D1
810             MOVE.B    #0,D0
811             TRAP      #15
812             LEA       HTXT1B,A1     ;mem display
813             MOVE.W    #20,D1
814             MOVE.B    #0,D0
815             TRAP      #15
816             MOVE.W    #0,D1         ;newline
817             TRAP      #15
818
819             LEA       HTXT2,A1      ;hxdec command text
820             MOVE.W    #75,D1
821             MOVE.B    #0,D0
822             TRAP      #15
823             MOVE.B    #0,D1         ;newline
824             TRAP      #15
825
826             LEA       HTXT3,A1      ;sort command text
827             MOVE.W    #69,D1
```

```
828             MOVE.B  #0,D0
829             TRAP    #15
830             LEA     HTXT3A,A1       ;sort command text continued
831             MOVE.W  #57,D1
832             MOVE.B  #0,D0
833             TRAP    #15
834             LEA     HTXT3B,A1       ;sort command text continued
835             MOVE.W  #20,D1
836             MOVE.B  #0,D0
837             TRAP    #15
838             LEA     HTXT3C,A1       ;sort command text continued
839             MOVE.W  #21,D1
840             MOVE.B  #0,D0
841             TRAP    #15
842             LEA     HTXT3D,A1       ;sort command text continued
843             MOVE.W  #29,D1
844             MOVE.B  #0,D0
845             TRAP    #15
846             LEA     HTXT3E,A1       ;sort command text continued
847             MOVE.W  #51,D1
848             MOVE.B  #0,D0
849             TRAP    #15
850             MOVE.B  #0,D1           ;newline
851             TRAP    #15
852
853             LEA     HTXT4,A1        ;memory modify command text
854             MOVE.W  #71,D1
855             MOVE.B  #0,D0
856             TRAP    #15
857             LEA     HTXT4A,A1       ;mem modify command text
    continued
858             MOVE.W  #69,D1
859             MOVE.B  #0,D0
860             TRAP    #15
861             LEA     HTXT4B,A1       ;mem modify command text
    continued
862             MOVE.W  #27,D1
863             MOVE.B  #0,D0
864             TRAP    #15
865             LEA     HTXT4C,A1       ;mem modify command text
    continued
866             MOVE.W  #30,D1
867             MOVE.B  #0,D0
868             TRAP    #15
```

```
869             LEA     HTXT4D,A1       ;mem modify command text
    continued
870             MOVE.W  #31,D1
871             MOVE.B  #0,D0
872             TRAP    #15
873             LEA     HTXT4E,A1       ;mem modify command text
    continued
874             MOVE.W  #36,D1
875             MOVE.B  #0,D0
876             TRAP    #15
877             MOVE.B  #0,D1
878             TRAP    #15             ;newline
879
880             LEA     HTXT5,A1        ;memory set command text
881             MOVE.W  #70,D1
882             MOVE.B  #0,D0
883             TRAP    #15
884             LEA     HTXT5A,A1       ;memory set command text
    continued
885             MOVE.W  #28,D1
886             MOVE.B  #0,D0
887             TRAP    #15
888             MOVE.B  #0,D1           ;newline
889             TRAP    #15
890
891             LEA     HTXT6,A1        ;block fill command text
892             MOVE.W  #70,D1
893             MOVE.B  #0,D0
894             TRAP    #15
895             LEA     HTXT6A,A1       ;block fill command text
896             MOVE.W  #72,D1
897             MOVE.B  #0,D0
898             TRAP    #15
899             LEA     HTXT6B,A1       ;block fill command text
900             MOVE.W  #38,D1
901             MOVE.B  #0,D0
902             TRAP    #15
903             MOVE.B  #0,D1
904             TRAP    #15             ;newline
905
906
907             LEA     HTXT7,A1        ;block move command text
908             MOVE.W  #68,D1
909             MOVE.B  #0,D0
910             TRAP    #15
```

```
911             LEA     HTXT7A,A1       ;block move command text
912             MOVE.W  #72,D1
913             MOVE.B  #0,D0
914             TRAP    #15
915             LEA     HTXT7B,A1       ;block move command text
916             MOVE.W  #24,D1
917             MOVE.B  #0,D0
918             TRAP    #15
919             MOVE.B  #0,D1           ;newline
920             TRAP    #15
921
922             LEA     HTXT8,A1        ;block test command text
923             MOVE.W  #71,D1
924             MOVE.B  #0,D0
925             TRAP    #15
926             LEA     HTXT8A,A1       ;block test command text
927             MOVE.W  #59,D1
928             MOVE.B  #0,D0
929             TRAP    #15
930             MOVE.B  #0,D1           ;newline
931             TRAP    #15
932
933             LEA     HTXT9,A1        ;block search command text
934             MOVE.W  #70,D1
935             MOVE.B  #0,D0
936             TRAP    #15
937             LEA     HTXT9A,A1       ;block search command text
938             MOVE.W  #45,D1
939             MOVE.B  #0,D0
940             TRAP    #15
941             MOVE.B  #0,D1           ;newline
942             TRAP    #15
943
944             LEA     HTXT10,A1       ;go command text
945             MOVE.W  #61,D1
946             MOVE.B  #0,D0
947             TRAP    #15
948             MOVE.B  #0,D1          ;newline
949             TRAP    #15
950
951             LEA     HTXT11,A1       ;df command text
952             MOVE.W  #56,D1
953             MOVE.B  #0,D0
954             TRAP    #15
955             MOVE.B  #0,D1
```

```
956             TRAP    #15
957
958             LEA     HTXT12,A1   ;help command text
959             MOVE.W  #66,D1
960             MOVE.B  #0,D0
961             TRAP    #15
962             MOVE.B  #0,D1       ;newline
963             TRAP    #15
964
965             LEA     HTXT13,A1   ;echo command text
966             MOVE.W  #52,D1
967             MOVE.B  #0,D0
968             TRAP    #15
969             MOVE.B  #0,D1       ;newline
970             TRAP    #15
971
972             LEA     HTXT14,A1   ;modify register command text
973             MOVE.W  #71,D1
974             MOVE.B  #0,D0
975             TRAP    #15
976             LEA     HTXT15,A1   ;modify register command text
977             MOVE.W  #63,D1
978             MOVE.B  #0,D0
979             TRAP    #15
980             MOVE.B  #0,D1       ;newline
981             TRAP    #15
982
983             BRA     RESTORE
```

2.2.2 Memory Display

2.2.2.1 Algorithm and Flowchart

Memory display is an extremely useful tool to look at blocks of memory. The syntax to call this function is `MDSP <address1> <address2`, where `<address1>` is the starting address and `<address2>` is the ending address of the memory contents to be shown. This command also displays the block of memory from `<address1>` to `<address2 +16bytes>`. The flowchart for this command is shown in Figure 4.

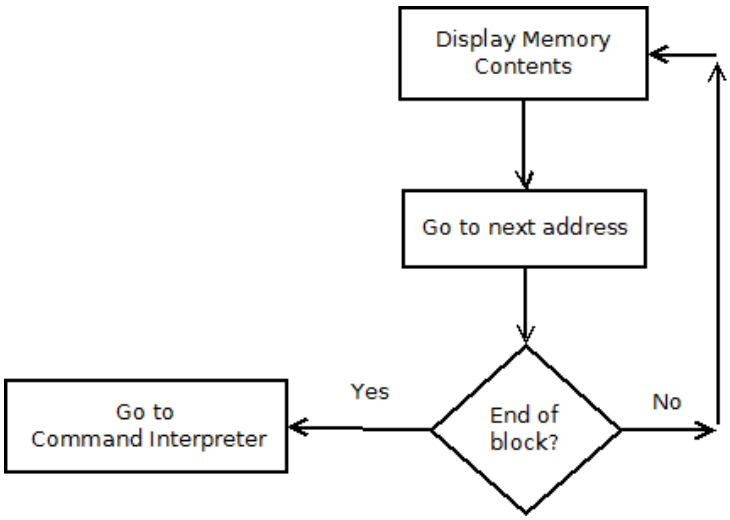

Figure 4: Flowchart for Memory Display

2.2.2.2 Assembly Code

```
1034 MEMDISP:    LEA       BUFFER,A2
1035             MOVE.L    #1,D6          ;counter for how many times to
       loop
1036             ADD.L     #5,A2          ;get first address
1037             MOVE.L    A2,A3
1038 FINDEND1:   CMP.B     #$20,(A3)+
1039             BEQ       FINDNEXT
1040             BRA       FINDEND1
1041 FINDNEXT:   MOVE.L    A3,A4
1042             MOVE.L    A3,A5
1043             SUB.L     #1,A3     ;get rid of off by one error
1044 FINDEND2:   CMP.B     #$00,(A5)+
1045             BEQ       MEMNEXT
1046             BRA       FINDEND2
1047 MEMNEXT:    SUB.L     #1,A5     ;off by one error
1048             JSR       ASCII_ADDRESS
1049             MOVE.L    D5,A6     ;put 1st address in A6
1050             MOVE.L    A4,A2
1051             MOVE.L    A5,A3
1052             JSR ASCII_ADDRESS
1053             MOVE.L    D5,A5     ;second address in A5
1054             MOVE.L    A6,A0     ;for second run through
1055             MOVE.L    A5,A1     ;see above comment
```

```
1056                ADD.L    #16,A1 ;16 byte offset
1057                MOVEM.L  A1,-(SP)
1058 DISPLOOP:     CMP.L    A6,A5
1059                BLT      SECONDLOOP
1060                MOVE.L   A6,D3
1061                JSR      HEXTOASCII
1062                SUB.L    A2,A3
1063                MOVE.L   A3,D1    ;number of ascii values to display
1064                MOVE.L   A2,A1
1065                MOVE.L   #1,D0
1066                TRAP     #15
1067                LEA      SPACE,A1
1068                MOVE.L   #1,D1
1069                TRAP     #15
1070                CLR.L    D3
1071                MOVE.B   (A6),D3
1072                JSR      HEXTOASCII
1073                SUB.L    A2,A3
1074                MOVE.L   A3,D1
1075                MOVE.L   A2,A1
1076                MOVE.L   #0,D0
1077                TRAP     #15
1078                ADD.L    #1,A6
1079                BRA      DISPLOOP
1080
1081 SECONDLOOP:
1082                MOVE.B   #0,D0
1083                MOVE.B   #0,D1
1084                TRAP     #15
1085                MOVEM.L  (SP)+,A1
1086                MOVE.L   A0,A6    ;reinit
1087                MOVE.L   A1,A5
1088                SUBI.L   #1,D6
1089                CMP.L    #$0,D6
1090                BEQ      DISPLOOP
1091                SUB.L    #4,SP    ;off by long error on stack
1092                BRA      RESTORE
```

2.2.3 HXDEC

2.2.3.1 Algorithm and Flowchart

This command allows the user to enter a hexadecimal value (up to FFFF), and the program will return the equivalent value in decimal format. The syntax to call this function is `HXDEC <data>`. It works by extracting the

ASCII values byte by byte and determining the 16's place of each byte. The value extracted is then multiplied by its respective 16's place and added to a register that stores the total. This total must then be converted into BCD for output and then into ASCII to display it on the terminal. The flowchart for this command is shown in Figure 5.

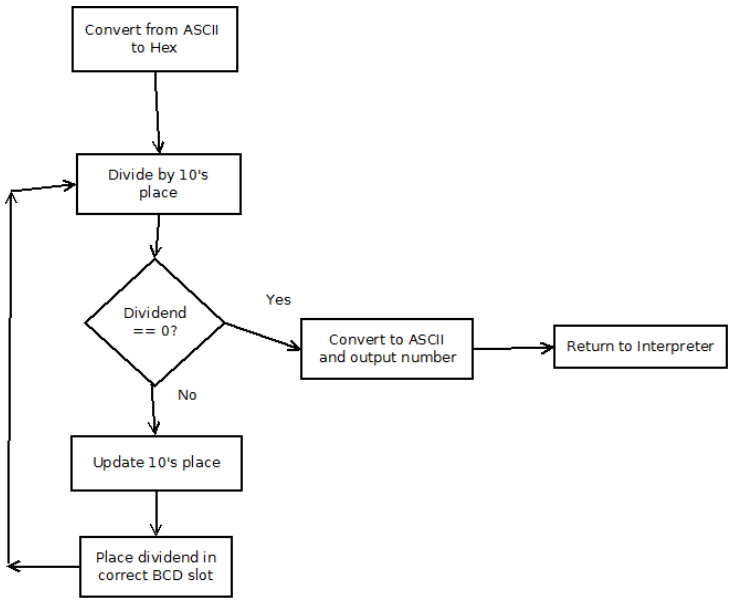

Figure 5: Flowchart for HXDEC

2.2.3.2 Assembly Code

```
1096 HXDC:     LEA BUFFER,A2     ;load buffer
1097           ADD.L    #6,A2    ; start of number
1098           MOVE.L   A2,A3    ;set up end pointer
1099           MOVE.L   #1,D1    ;set up 16's place
1100           CLR.L    D2       ;clear total
1101           CLR.L    D3       ;temp holder for number
1102           CLR.L    D6       ;Final Value in BCD
1103           MOVE.L   #10000,D4     ;maximum 10's place of converted
       number
1104           MOVE.L   #16,D5       ;Max number of rotates needed
1105           LEA $3A00,A5
1106           LEA $3A00,A4     ;set up start pointer
1107 FINDLASTNUM:
```

```
1108            CMP.B #$00,(A3)+
1109            BEQ       CONVERTMINUS1
1110            BRA       FINDLASTNUM
1111 CONVERTMINUS1:
1112                  SUB.L    #1,A3 ; cure off by 1 error
1113 CONVERT:
1114                  SUB.L    #1,A3
1115                  CMP      A3,A2
1116                  BGT      ENDCONVERT
1117                  CMP.B    #$40,(A3)
1118                  BGT      HIGHHEX
1119                  SUBI.B   #$30,(A3)    ;get hex value
1120                  BRA      COMPUTATION
1121 HIGHHEX:        SUBI.B   #$37,(A3)    ;get hex value
1122 COMPUTATION:
1123                  MOVE.B   (A3),D3
1124                  MULU     D1,D3    ;get 16's place
1125                 ; DIVU     #16,D3   ;get rid of off by 1 exponent error
1126                  MULU     #16,D1   ;inc 16's place counter
1127                  MOVE.B   D3,(A4)
1128                  SUB.L    #1,A4
1129                  ADD.L    D3,D2    ;store it in total for debugging
1130                  CLR.L    D3       ;get rid of any numbers in there
1131                  BRA      CONVERT
1132 ENDCONVERT:                         ;must convert back to ascii for
        display
1133                  CLR.L    D3       ;Cleared for workability
1134                  DIVU     D4,D2    ;get 10's place digit
1135                  MOVE.W   D2,D3    ;extract 10's place digit to D3
1136                  ROL.L    D5,D3    ;put it in its place
1137                  CLR.W    D2       ;get rid of whole number
1138                  SWAP     D2       ;keep remainder
1139                  SUBI.L   #4,D5    ;dec rotate counter
1140                  ADD.L    D3,D6    ;put it into it's place
1141                  DIVU     #10,D4   ;go down a 10's place
1142                  CMP.W    #0,D4    ;are we done
1143                  BEQ      OUTPUTNUM
1144                  BRA      ENDCONVERT
1145
1146 OUTPUTNUM:
1147                MOVE.L   D6,D3    ;put into register for conversion to
         ASCII
1148                  JSR      HEXTOASCII
1149                  MOVEA.L  A2,A1    ;get start of number
1150                  SUBA     A2,A3    ;get how many bytes to output
```

```
1151            MOVE.L    A3,D1    ;for Trap call
1152            MOVE.L    #0,D0
1153            TRAP      #15
1154
1155            BRA RESTORE
```

2.2.4 SORTW

2.2.4.1 Algorithm and Flowchart

This command implements the most common sort algorithm for a set of data, the bubble sort. Because the user has the choice to choose between sorting the data in ascending or descending order, it also implements a "rock" sort. It works by first determining which option, ascending or descending, the user has selected. Once determined, the first data in the set is analyzed to the next immediate adjacent value in memory. If the current data is larger than the next data (assuming ascending order for example), the two words of data are swapped. This value is continuously checked against its immediate adjacent memory until it "fits" in the current state of the list. This process is repeated for n elements in a list of n words. The runtime is $\mathcal{O}(n^2)$, and the syntax for this command is `SORTW <option> <address1> <address2>`, where both `<address1>` and `<address2>` are even addresses. The flowchart is shown in Figure 6.

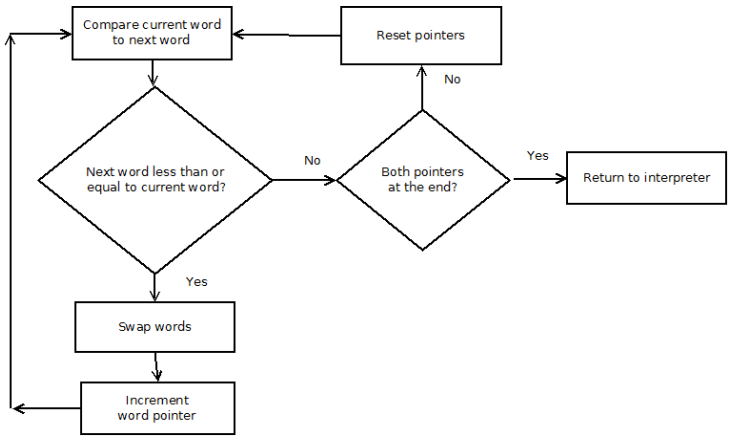

Figure 6: Flowchart for SORTW

2.2.4.2 Assembly Code

```
1159 SORTW:   ADD.L    #1,A1          ;increment to check for semicolon/
         dash
1160          CMP.B    #$2D,(A1)      ;check for default
1161          BEQ      DEFAULTTEST
1162          CMP.B    #$3B,(A1)+
1163          BNE      ERRORSR
1164          CMP.B    #$41,(A1)      ;is it ascending?
1165          BEQ      ASCEND
1166          CMP.B    #$44,(A1)      ;or descending?
1167          BNE      ERRORSR
1168          BRA      DESCEND
1169
1170 DEFAULTTEST:
1171                   ADD.L  #1,A1    ;check for paren
1172          CMP.B    #$28,(A1)+     ;(
1173          BNE      ERRORSR
1174          CMP.B    #$64,(A1)+     ;d
1175          BNE      ERRORSR
1176          CMP.B    #$65,(A1)+     ;e
1177          BNE      ERRORSR
1178          CMP.B    #$66,(A1)+     ;f
1179          BNE      ERRORSR
1180          CMP.B    #$61,(A1)+     ;a
1181          BNE      ERRORSR
1182          CMP.B    #$75,(A1)+     ;u
1183          BNE      ERRORSR
1184          CMP.B    #$6C,(A1)+     ;l
1185          BNE      ERRORSR
1186          CMP.B    #$74,(A1)+     ;t
1187          BNE      ERRORSR
1188          CMP.B    #$29,(A1)    ;)
1189          BNE      ERRORSR
1190          BRA      DESCEND
1191
1192
1193 ASCEND:
1194          ADD.L    #1,A1     ;inc
1195          CMP.B    #$20,(A1)     ;check space
1196          BNE      ERRORSR
1197          ADD.L    #1,A1     ;start of 1st address
1198          MOVE.L   A1,A2
1199          MOVE.L   A2,A3
1200 AGETFIRSTADDRESS:
```

```
1201          CMP.B     #$00,(A3)
1202          BEQ       ERRORSR       ;incorrect syntax
1203          CMP.B     #$20,(A3)+    ;trying to find the end
1204          BEQ       AFADDCONV
1205          BRA       AGETFIRSTADDRESS
1206 AFADDCONV:
1207          SUB.L     #1,A3     ;off by one error
1208          JSR ASCII_ADDRESS     ;D5 now has that address
1209          MOVE.L D5,A4
1210          ADD.L     #1,A3     ;start of second address
1211          MOVE.L    A3,A2     ;setup for second address
1212 AGETSECADDRESS:
1213          CMP.B     #$00,(A3)+    ;trying to find the end
1214          BEQ       ASADDCONV
1215          BRA       AGETSECADDRESS
1216 ASADDCONV:
1217           SUB.L    #1,A3     ;off by one
1218           JSR      ASCII_ADDRESS
1219           MOVE.L   D5,A5
1220           MOVEA.L  A4,A6     ;CLR A6
1221
1222 ARESETLOOP: MOVE.L   A6,A4     ;reset to top of loop
1223 ACMP:       CMP.W    (A4)+,(A4)+   ;check adjacent mem
1224             BLS.S    ASWAP
1225             SUBQ.L   #2,A4
1226             CMP.L    A4,A5     ;done?
1227             BNE      ACMP       ;nope
1228             BRA      DONEASCEND   ;yep
1229 ASWAP:       MOVE.L   -(A4),D0     ;start bubbling
1230             SWAP.W   D0
1231             MOVE.L   D0,(A4)
1232             BRA      ARESETLOOP
1233
1234
1235 DESCEND:
1236          ADD.L     #1,A1     ;inc
1237          CMP.B     #$20,(A1)     ;check space
1238          BNE       ERRORSR
1239          ADD.L     #1,A1     ;start of 1st address
1240          MOVE.L    A1,A2
1241          MOVE.L    A2,A3
1242 DGETFIRSTADDRESS:
1243          CMP.B     #$00,(A3)
1244          BEQ       ERRORSR       ;incorrect syntax
1245          CMP.B     #$20,(A3)+    ;trying to find the end
```

```
1246            BEQ         DFADDCONV
1247            BRA         DGETFIRSTADDRESS
1248 DFADDCONV:
1249            SUB.L       #1,A3     ;off by one error
1250            JSR ASCII_ADDRESS     ;D5 now has that address
1251            MOVE.L D5,A4
1252            ADD.L       #1,A3     ;start of second address
1253            MOVE.L      A3,A2     ;setup for second address
1254 DGETSECADDRESS:
1255            CMP.B       #$00,(A3)+  ;trying to find the end
1256            BEQ         DSADDCONV
1257            BRA         DGETSECADDRESS
1258 DSADDCONV:
1259             SUB.L      #1,A3     ;off by one
1260             JSR        ASCII_ADDRESS
1261             MOVE.L     D5,A5
1262             MOVEA.L    A4,A6   ;CLR A6
1263
1264 DRESETLOOP: MOVE.L     A6,A4     ;reset to top of loop
1265 DCMP:       CMP.W      (A4)+,(A4)+  ;check adjacent mem
1266              BHI.S      DSWAP
1267              SUBQ.L     #2,A4
1268              CMP.L      A4,A5     ;done?
1269              BNE        DCMP      ;nope
1270              BRA        DONEDESCEND  ;yep
1271 DSWAP:       MOVE.L    -(A4),D0     ;start bubbling
1272              SWAP.W     D0
1273              MOVE.L     D0,(A4)
1274              BRA        DRESETLOOP
1275
1276 DONEASCEND:
1277 DONEDESCEND:
1278              BRA RESTORE
```

2.2.5 Memory Modify

2.2.5.1 Algorithm and Flowchart

This command first determines which option the user has selected. Depending on this option, it reads the address entered by the user and displays the specified amount of data currently stored in memory. The user is then prompted to enter data to store into memory. The command increments the memory location and asks for input until the user enters the '.' character. The syntax for this command is `MM <option> <address>`. The flowchart is

shown in Figure 7.

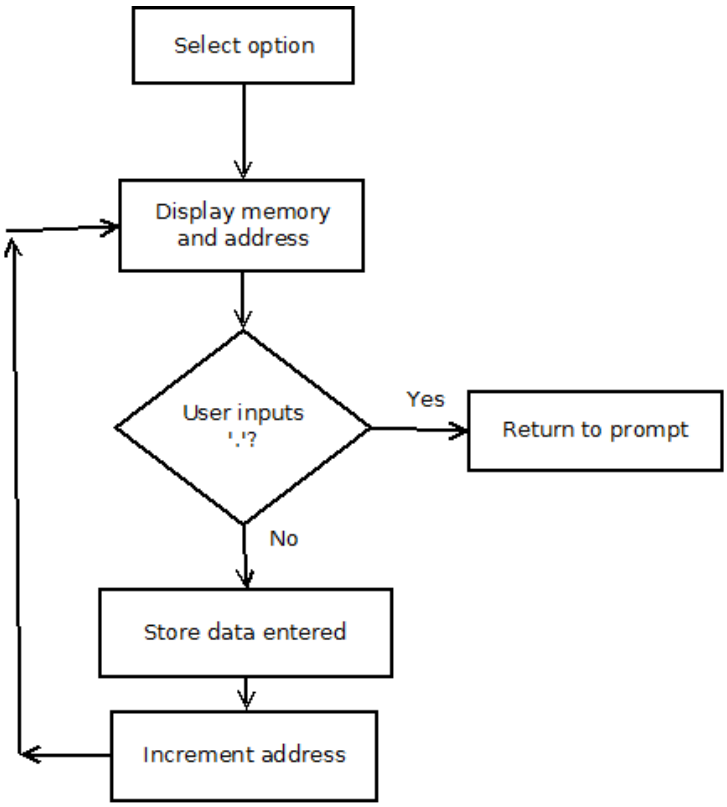

Figure 7: Flowchart for Memory Modify

2.2.5.2 Assembly Code

```
1282 MM:      CLR.L     D2        ;used for storing values
1283          CLR.L     D6
1284 SIZECHECK:
1285          MOVE.L    A1,A3     ;set up to find end ptr
1286 ENDPTRMM:
1287          CMP.B     #$00,(A3)+
1288          BNE       ENDPTRMM
1289          SUB.L     #1,A3     ;off by one error
1290          ADD.L     #1,A1     ;inc pointer to start of specifier
1291          CMP.B     #$2D,(A1)    ;check for default
1292          BEQ       DEFAULT
1293          CMP.B     #$3B,(A1)
```

```
1294            BNE       ERRORSR
1295            ADD.L     #1,A1     ;find out which size
1296            CMP.B     #$57,(A1) ; word
1297            BEQ       WORD
1298            CMP.B     #$4C,(A1)     ;long
1299            BEQ       LONG
1300            BRA       ERRORSR
1301
1302 ************************************************
1303
1304 DEFAULT:
1305            ADD.L   #1,A1     ;check for paren
1306            CMP.B     #$28,(A1)+   ;(
1307            BNE       ERRORSR
1308            CMP.B     #$64,(A1)+   ;d
1309            BNE       ERRORSR
1310            CMP.B     #$65,(A1)+   ;e
1311            BNE       ERRORSR
1312            CMP.B     #$66,(A1)+   ;f
1313            BNE       ERRORSR
1314            CMP.B     #$61,(A1)+   ;a
1315            BNE       ERRORSR
1316            CMP.B     #$75,(A1)+   ;u
1317            BNE       ERRORSR
1318            CMP.B     #$6C,(A1)+   ;l
1319            BNE       ERRORSR
1320            CMP.B     #$74,(A1)+   ;t
1321            BNE       ERRORSR
1322            CMP.B     #$29,(A1)+   ;)
1323            BNE       ERRORSR
1324
1325
1326            ADD.L     #1,A1         ;set up for subroutine
1327            MOVE      A1,A2         ;set up for subroutine
1328            MOVEM.L   D1/D6/A1-A3,-(SP)
1329            JSR       ASCII_ADDRESS
1330            MOVEM.L   (SP)+,D1/D6/A1-A3
1331            MOVE.L    D5,A4         ;set up address to modify
1332
1333 MODIFYLOOP:
1334            *--------Display Memory First------*
1335            MOVE.L    A4,D3         ;set up for subroutine
1336            JSR       HEXTOASCII    ;convert new address to ascii for
       output
1337            SUBA      A2,A3         ;get num of bytes to produce
```

```
1338            MOVE.L   #1,D0
1339            MOVE.L   A3,D1
1340            MOVE.L   A2,A1
1341            TRAP     #15
1342
1343            *add colon to denote containing data*
1344            MOVE.B   #$3A,(A1)
1345            MOVE.L   #1,D1    ;display only the colon
1346            MOVE.L   #1,D0
1347            TRAP     #15
1348
1349            MOVE.B   (A4),D3
1350            JSR      HEXTOASCII
1351            MOVE.L   #2,D1
1352            SUB.L    A2,A3
1353            CMP      #2,A3
1354            BEQ      FORMATGOOD
1355            SUB.L    #1,A2
1356 FORMATGOOD:
1357            MOVE.L   A2,A1
1358            MOVE.B   #1,D0
1359            TRAP     #15
1360
1361            MOVE.B   #$20,(A1)
1362            MOVE.L   #1,D1    ;space between held data and input
1363            MOVE.L   #1,D0
1364            TRAP     #15
1365
1366
1367            *------Enter Input--------*
1368            CLR.L    D3
1369            MOVE.L   #4,D6
1370            LEA      BUFFER,A1    ;set up storage for command
1371            MOVE.B   #2,D0        ;load input trap call
1372            TRAP     #15
1373            CMP.B    #$2E,(A1)
1374            BEQ      ENDLP
1375            CMP.B    #$00,(A1)
1376            BEQ      ENTER
1377
1378 PARSELOOP:
1379            CMP.B    #$00,(A1)
1380            BEQ      ENDPARSE
1381            CMP.B    #$40,(A1)
1382            BGT      HIGHHEXMM
```

```
1383            SUBI.B  #$30,(A1)    ;get hex value
1384            BRA     NEXTMMSTEP
1385 HIGHHEXMM: SUBI.B  #$37,(A1)    ;get hex value
1386 NEXTMMSTEP:
1387            MOVE.B  (A1),D2
1388            ROL.L   D6,D2
1389            SUBI.L  #4,D6
1390            ADD.L   #1,A1
1391            ADD.B   D2,D3    ;total byte stored in D3
1392            BRA     PARSELOOP
1393 ENDPARSE:
1394            MOVE.B  D3,(A4)    ;commit memory change
1395 ENTER:     ADD.L   #1,A4    ;increment address
1396            BRA     MODIFYLOOP
1397
1398 ************************************************
1399
1400 WORD:
1401
1402            ADD.L   #2,A1        ;set up for subroutine
1403            MOVE    A1,A2        ;set up for subroutine
1404            MOVEM.L D1/D6/A1-A3,-(SP)
1405            JSR     ASCII_ADDRESS
1406            MOVEM.L (SP)+,D1/D6/A1-A3
1407            MOVE.L  D5,A4        ;set up address to modify
1408
1409 MODIFYLOOPW:
1410            *--------Display Memory First-----*
1411          ; MOVE.L  A4,D0
1412          ; DIVU    #2,D0
1413          ; SWAP    D0       ;check if it's an odd address
1414          ; CMP.W   #$00,D0
1415          ; BNE     ERRORSR
1416            MOVE.L  A4,D3        ;set up for subroutine
1417            MOVE.L  A4,A5        ;next byte of memory may not be
      needed
1418            ADD.L   #1,A5
1419            JSR     HEXTOASCII   ;convert new address to ascii for
      output
1420            SUBA    A2,A3        ;get num of bytes to produce
1421            MOVE.L  #1,D0
1422            MOVE.L  A3,D1
1423            MOVE.L  A2,A1
1424            TRAP    #15
1425
```

```
1426            *add colon to denote containing data*
1427            MOVE.B  #$3A,(A1)
1428            MOVE.L  #1,D1    ;display only the colon
1429            MOVE.L  #1,D0
1430            TRAP    #15
1431
1432            MOVE.B  (A4),D3
1433            JSR     HEXTOASCII
1434            MOVE.L  #2,D1
1435            SUB.L   A2,A3
1436            CMP     #2,A3
1437            BEQ     FORMATGOOD1
1438            SUB.L   #1,A2
1439 FORMATGOOD1:
1440
1441            MOVE.L  A2,A1
1442            MOVE.B  #1,D0
1443            TRAP    #15
1444
1445            MOVE.B  (A5),D3
1446            JSR     HEXTOASCII
1447            MOVE.L  #2,D1
1448            SUB.L   A2,A3
1449            CMP     #2,A3
1450            BEQ     FORMATGOOD2
1451            SUB.L   #1,A2
1452 FORMATGOOD2:
1453
1454            MOVE.L  A2,A1
1455            MOVE.B  #1,D0
1456            TRAP    #15
1457
1458
1459            MOVE.B  #$20,(A1)
1460            MOVE.L  #1,D1    ;space between held data and input
1461            MOVE.L  #1,D0
1462            TRAP    #15
1463
1464
1465            *------Enter Input--------*
1466            CLR.L   D3
1467            MOVE.L  #12,D6
1468            LEA     BUFFER,A1    ;set up storage for command
1469            MOVE.B  #2,D0        ;load input trap call
1470            TRAP    #15
```

```
1471            CMP.B    #$2E,(A1)
1472            BEQ      ENDLP
1473            CMP.B    #$00,(A1)
1474            BEQ      ENTERW
1475
1476 PARSELOOPW:
1477            CMP.B    #$00,(A1)
1478            BEQ      ENDPARSEW
1479            CMP.B    #$40,(A1)
1480            BGT      HIGHHEXMMW
1481            SUBI.B   #$30,(A1)    ;get hex value
1482            BRA      NEXTMMSTEPW
1483 HIGHHEXMMW: SUBI.B  #$37,(A1)   ;get hex value
1484 NEXTMMSTEPW:
1485            MOVE.B   (A1),D2
1486            ROL.L    D6,D2
1487            SUBI.L   #4,D6
1488            ADD.L    #1,A1
1489            ADD.L    D2,D3    ;total byte stored in D3
1490            CLR.L    D2       ;clear for next rotate
1491            BRA      PARSELOOPW
1492 ENDPARSEW:
1493
1494            MOVE.W   D3,(A4)    ;commit memory change
1495 ENTERW:    ADD.L    #2,A4    ;increment address
1496            BRA      MODIFYLOOPW
1497
1498 ************************************************
1499
1500 LONG:
1501            ADD.L    #2,A1         ;set up for subroutine
1502            MOVE     A1,A2         ;set up for subroutine
1503            MOVEM.L  D1/D6/A1-A3,-(SP)
1504            JSR      ASCII_ADDRESS
1505            MOVE.L   D5,A4         ;set up address to modify
1506            MOVEM.L  (SP)+,D1/D6/A1-A3
1507
1508
1509 MODIFYLOOPL:
1510            *--------Display Memory First------*
1511           ; MOVE.L   A4,D0
1512           ; DIVU     #2,D0
1513           ; SWAP     D0       ;check if it's an odd address
1514           ; CMP.W    #$00,D0
1515           ; BNE      ERRORSR
```

```
1516            MOVE.L   A4,D3          ;set up for subroutine
1517            MOVE.L   A4,A5          ;next byte of memory may not be
        needed
1518            ADD.L    #1,A5
1519            JSR      HEXTOASCII     ;convert new address to ascii for
        output
1520            SUBA     A2,A3          ;get num of bytes to produce
1521            MOVE.L   #1,D0
1522            MOVE.L   A3,D1
1523            MOVE.L   A2,A1
1524            TRAP     #15
1525
1526            *add colon to denote containing data*
1527            MOVE.B   #$3A,(A1)
1528            MOVE.L   #1,D1    ;display only the colon
1529            MOVE.L   #1,D0
1530            TRAP     #15
1531
1532            MOVE.B   (A4),D3
1533            JSR      HEXTOASCII
1534            MOVE.L   #2,D1
1535            SUB.L    A2,A3
1536            CMP      #2,A3
1537            BEQ      FORMATGOOD3
1538            SUB.L    #1,A2
1539 FORMATGOOD3:
1540
1541            MOVE.L   A2,A1
1542            MOVE.B   #1,D0
1543            TRAP     #15
1544
1545            MOVE.B   (A5)+,D3
1546            JSR      HEXTOASCII
1547            MOVE.L   #2,D1
1548            SUB.L    A2,A3
1549            CMP      #2,A3
1550            BEQ      FORMATGOOD4
1551            SUB.L    #1,A2
1552 FORMATGOOD4:
1553
1554            MOVE.L   A2,A1
1555            MOVE.B   #1,D0
1556            TRAP     #15
1557
1558            MOVE.B   (A5)+,D3
```

```
1559            JSR         HEXTOASCII
1560            MOVE.L      #2,D1
1561            SUB.L       A2,A3
1562            CMP         #2,A3
1563            BEQ         FORMATGOOD5
1564            SUB.L       #1,A2
1565 FORMATGOOD5:
1566
1567            MOVE.L      A2,A1
1568            MOVE.B      #1,D0
1569            TRAP        #15
1570            MOVE.B      (A5)+,D3
1571            JSR         HEXTOASCII
1572            MOVE.L      #2,D1
1573            SUB.L       A2,A3
1574            CMP         #2,A3
1575            BEQ         FORMATGOOD6
1576            SUB.L       #1,A2
1577 FORMATGOOD6:
1578
1579            MOVE.L      A2,A1
1580            MOVE.B      #1,D0
1581            TRAP        #15
1582
1583            MOVE.B      #$20,(A1)
1584            MOVE.L      #1,D1       ;space between held data and input
1585            MOVE.L      #1,D0
1586            TRAP        #15
1587
1588
1589            *------Enter Input-------*
1590            CLR.L       D3
1591            MOVE.L      #28,D6
1592            LEA         BUFFER,A1   ;set up storage for command
1593            MOVE.B      #2,D0       ;load input trap call
1594            TRAP        #15
1595            CMP.B       #$2E,(A1)
1596            BEQ         ENDLP
1597            CMP.B       #$00,(A1)
1598            BEQ         ENTERL
1599
1600 PARSELOOPL:
1601            CMP.B       #$00,(A1)
1602            BEQ         ENDPARSEL
1603            CMP.B       #$40,(A1)
```

```
1604            BGT       HIGHHEXMML
1605            SUBI.B    #$30,(A1)     ;get hex value
1606            BRA       NEXTMMSTEPL
1607 HIGHHEXMML: SUBI.B   #$37,(A1)    ;get hex value
1608 NEXTMMSTEPL:
1609            MOVE.B    (A1),D2
1610            ROL.L     D6,D2
1611            SUBI.L    #4,D6
1612            ADD.L     #1,A1
1613            ADD.L     D2,D3     ;total byte stored in D3
1614            CLR.L     D2        ;clear for next rotate
1615            BRA       PARSELOOPL
1616 ENDPARSEL:
1617            MOVE.L    D3,(A4)     ;commit memory change
1618 ENTERL:    ADD.L     #4,A4     ;increment address
1619            BRA       MODIFYLOOPL
1620
1621
1622 ENDLP:    BRA RESTORE
```

2.2.6 Memory Set

2.2.6.1 Algorithm and Flowchart

This command is a simpler version of Memory Modify. It parses the data the user entered and stores it at one specified address. It has the syntax `MS <data> <address>`. The data entered must be byte sized. The flowchart is shown is Figure 8.

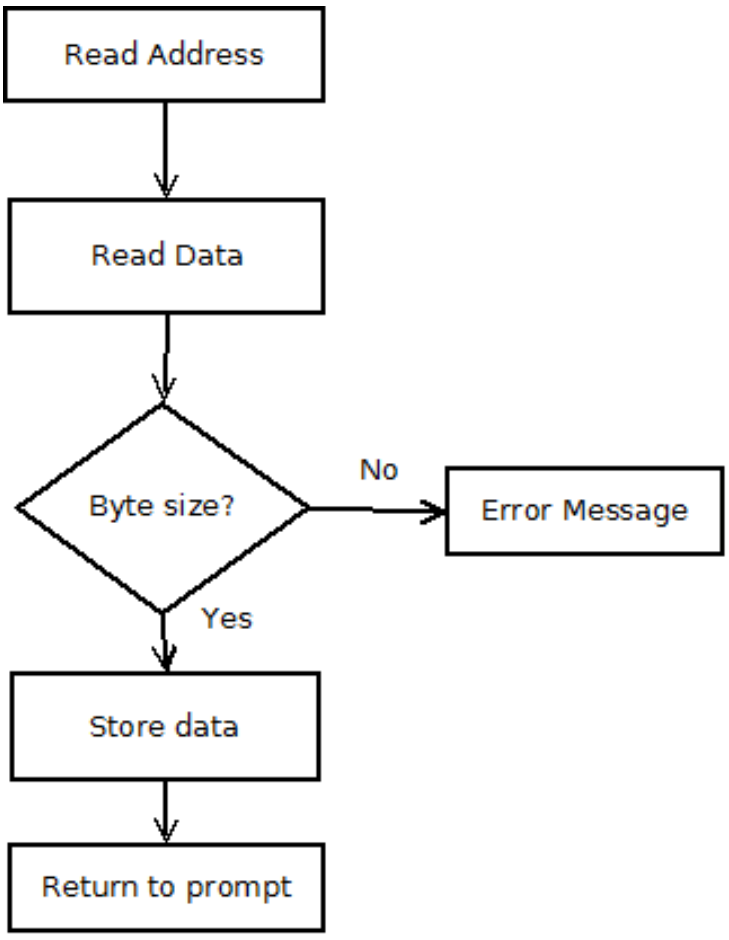

Figure 8: Flowchart for Memory Set

2.2.6.2 Assembly Code

```
986 MEMSET:     LEA       BUFFER,A2
987             ADD.L     #3,A2
988             MOVE.L    A2,A3     ;set up to find end
989 FINDEND:    CMP.B     #$20,(A3)+
990             BEQ       MEMCONT
991             BRA       FINDEND
992 MEMCONT:    SUB.L     #1,A3     ;get rid of off by one erro
993             MOVE.L    A3,A4     ;used for data length calculator
994
995             JSR  ASCII_ADDRESS
```

```
996                 MOVE.L   D5,D7    ;store value to be put in mem into
        D7
997                 ADD.L    #1,A3    ;increment to address to store it
998                 MOVE.L   A3,A2
999 MSFINDADDRESS:
1000                CMP.B    #$00,(A3)+
1001                BEQ      MOVEDATA
1002                BRA      MSFINDADDRESS
1003
1004 MOVEDATA:
1005                SUB.L    #1,A3    ;off by one error
1006                JSR      ASCII_ADDRESS
1007                MOVE.L   D5,A3    ;setup for storage
1008                MOVE.B   D7,(A3)    ;store data
1009                BRA      RESTORE
```

2.2.7 Block Fill

2.2.7.1 Algorithm and Flowchart

This command requires two even addresses to be entered. It then parses the word sized data entered by the user and fills the block of memory from the first address to the second address. The syntax for this command is `BF <data> <address1> <address2>`. The flowchart is shown in Figure 9.

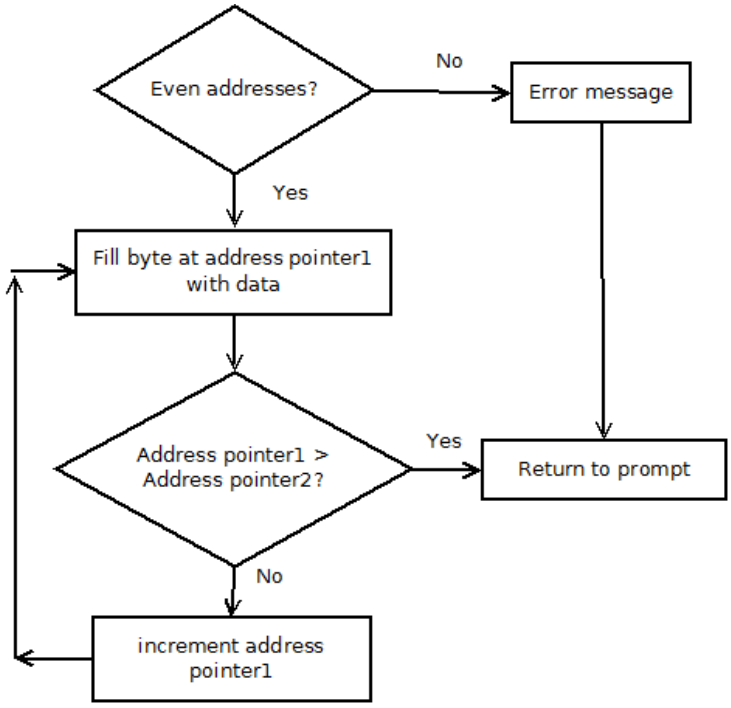

Figure 9: Flowchart for Block Fill

2.2.7.2 Assembly Code

```
1627 BF:
1628         ADD.L     #1,A1     ;first byte of data
1629         MOVE.L    A1,A3     ;for end ptr
1630 BFGETENDDATA:
1631          CMP.B    #$20,(A3)+
1632          BEQ      BFNEXTADDR
1633          BRA      BFGETENDDATA
1634 BFNEXTADDR:
1635          MOVE.L   A1,A2     ;for subroutine
1636          SUB.L    #1,A3     ;off by one error
1637          JSR      ASCII_ADDRESS
1638          MOVE.L   D5,-(SP)      ;save data on the stack
1639
1640          ADD.L    #1,A3     ;inc end ptr to first byte of address
1641          MOVE.L   A3,A2     ;set start ptr
1642 BFGETENDADDRONE:
1643          CMP.B    #$20,(A3)+
```

```
1644            BEQ       BFNEXTADDRTWO
1645            BRA       BFGETENDADDRONE
1646
1647 BFNEXTADDRTWO:
1648            SUB.L     #1,A3     ;off by one error
1649            JSR       ASCII_ADDRESS    ;convert address to hex
1650            MOVE.L    D5,A5          ;store address 1 in A5
1651            DIVU      #2,D5
1652            SWAP      D5
1653            CMP.W     #$00,D5
1654            BNE       ERRORSR
1655
1656            ADD.L     #1,A3     ;inc end ptr to first byte of address
1657            MOVE.L    A3,A2     ;set start ptr
1658 BFGETLASTEND:
1659            CMP.B     #$00,(A3)+
1660            BEQ       STOREDATA
1661            BRA       BFGETLASTEND
1662
1663 STOREDATA:
1664            SUB.L     #1,A3     ;off by one error
1665            JSR       ASCII_ADDRESS
1666            MOVE.L    D5,A6     ;end address in A6
1667            DIVU      #2,D5
1668            SWAP      D5
1669            CMP.W     #$00,D5
1670            BNE       ERRORSR
1671            MOVE.L    (SP)+,D5
1672
1673 DATALOOP:
1674            CMP.L     A5,A6
1675            BLT       ENDBF
1676            MOVE.W    D5,(A5)+
1677            BRA       DATALOOP
1678
1679 ENDBF:    BRA RESTORE
```

2.2.8 Block Move

2.2.8.1 Algorithm and Flowchart

This command move a block of memory from one section to another. Both block sizes must be equal. Starting from the first address of the first block and the first address of the second block, it moves data byte by byte to the respective memory locations until all data has been copied. Its syntax is

`BMOV <address1> <address2> <address3> <address4>`. The flowchart is shown in Figure 10.

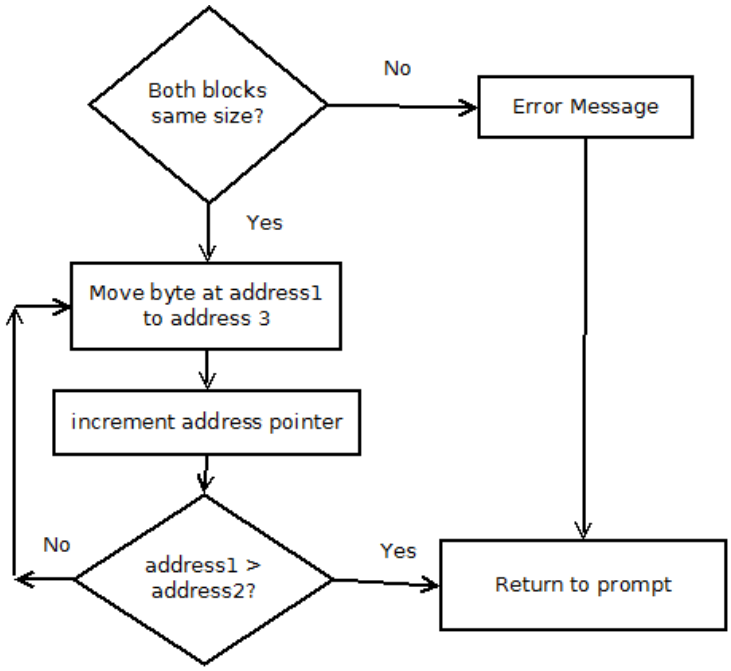

Figure 10: Flowchart for Block Move

2.2.8.2 Assembly Code

```
1682 BMOV:    ADD.L   #1,A1    ;get to start of first address
1683          MOVE.L  A1,A2    ;set up start ptr
1684          MOVE.L  A2,A3    ;set up end ptr
1685
1686 FIRSTADDRESS:
1687          CMP.B #$20,(A3)+
1688          BEQ     COMPUTEFIRSTADD
1689          BRA     FIRSTADDRESS
1690
1691 COMPUTEFIRSTADD:
1692          SUB.L   #1,A3    ;off by one error
1693          JSR     ASCII_ADDRESS
1694          MOVE.L  D5,A0    ; save 1st address
1695
1696          ADD.L   #1,A3
```

```
1697            MOVE.L   A3,A2
1698 SECONDADDRESS:
1699            CMP.B    #$20,(A3)+
1700            BEQ      COMPUTESECONDADDRESS
1701            BRA      SECONDADDRESS
1702
1703 COMPUTESECONDADDRESS:
1704            SUB.L    #1,A3    ;off by one error
1705            JSR      ASCII_ADDRESS
1706            MOVE.L   D5,A4    ;save 2nd address
1707
1708            ADD.L    #1,A3
1709            MOVE.L   A3,A2
1710 THIRDADDRESS:
1711            CMP.B    #$20,(A3)+
1712            BEQ      COMPUTETHIRDADDRESS
1713            BRA      THIRDADDRESS
1714
1715 COMPUTETHIRDADDRESS:
1716            SUB.L    #1,A3
1717            JSR      ASCII_ADDRESS
1718            MOVE.L   D5,A5    ;save 3rd address
1719
1720            ADD.L    #1,A3
1721            MOVE.L   A3,A2
1722 FOURTHADDRESS:
1723            CMP.B    #$00,(A3)+
1724            BEQ      COMPUTEFOURTHADDRESS
1725            BRA      FOURTHADDRESS
1726
1727 COMPUTEFOURTHADDRESS:
1728            SUB.L    #1,A3
1729            JSR      ASCII_ADDRESS
1730            MOVE.L   D5,A6    ;save 3rd address
1731
1732
1733
1734            *Check for matching dimensions*
1735            MOVE.L   A0,D0
1736            MOVE.L   A4,D1
1737            MOVE.L   A5,D5
1738            MOVE.L   A6,D6
1739            SUB.L    D0,D1
1740            SUB.L    D5,D6
1741            CMP.L    D1,D6
```

```
1742            BNE         ERRORSR
1743            CMP.L       A0,A4
1744            BLT         ERRORSR
1745            CMP.L       A5,A6
1746            BLT         ERRORSR
1747            ADD.L       #1,A4
1748
1749 DATATRANSFER:
1750            CMP.L       A0,A4
1751            BLT         BMOVDONE
1752            MOVE.B      (A0)+,(A5)+
1753            BRA         DATATRANSFER
1754
1755
1756
1757 BMOVDONE:
1758            BRA RESTORE
```

2.2.9 Block Test

2.2.9.1 Algorithm and Flowchart

This command fills a block of memory with byte sized data, then checks each byte of the block. If any byte is not equal to the data originally written, the program outputs the data read and the address where the test failed. If no error is detected, the program outputs a message declaring the test passed. The syntax for this command is `BTST <data> <address1> <address2>`. The flowchart is shown in Figure 11.

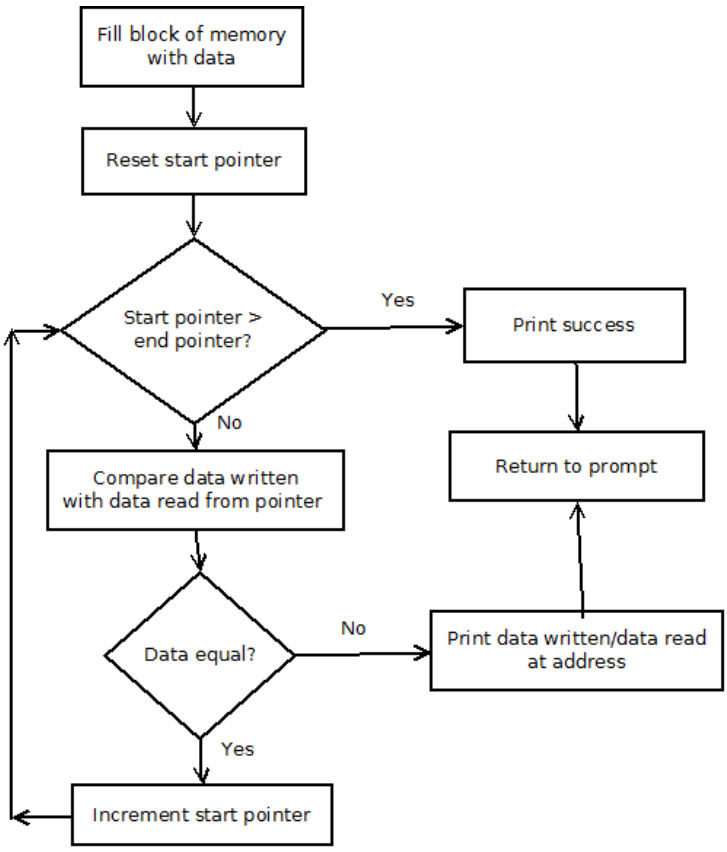

Figure 11: Flowchart for Block Test

2.2.9.2 Assembly Code

```
1762 BTST:
1763         ADD.L     #1,A1     ;first byte of data
1764         MOVE.L    A1,A3     ;for end ptr
1765 BTSTGETENDDATA:
1766          CMP.B    #$20,(A3)+
1767          BEQ      BTSTNEXTADDR
1768          BRA      BTSTGETENDDATA
1769 BTSTNEXTADDR:
1770          MOVE.L   A1,A2     ;for subroutine
1771          SUB.L    #1,A3     ;off by one error
1772          JSR      ASCII_ADDRESS
1773          MOVE.L   D5,-(SP)      ;save data on the stack
1774
```

```
1775            ADD.L   #1,A3    ;inc end ptr to first byte of address
1776            MOVE.L  A3,A2    ;set start ptr
1777 BTSTGETENDADDRONE:
1778            CMP.B   #$20,(A3)+
1779            BEQ     BTSTNEXTADDRTWO
1780            BRA     BTSTGETENDADDRONE
1781
1782 BTSTNEXTADDRTWO:
1783            SUB.L   #1,A3    ;off by one error
1784            JSR     ASCII_ADDRESS    ;convert address to hex
1785            MOVE.L  D5,A5        ;store address 1 in A5
1786            MOVE.L  D5,A4        ;for second run through
1787
1788            ADD.L   #1,A3    ;inc end ptr to first byte of address
1789            MOVE.L  A3,A2    ;set start ptr
1790 BTSTGETLASTEND:
1791            CMP.B   #$00,(A3)+
1792            BEQ     STOREDATABTST
1793            BRA     BTSTGETLASTEND
1794
1795
1796 STOREDATABTST:
1797            SUB.L   #1,A3    ;off by one error
1798            JSR     ASCII_ADDRESS
1799            MOVE.L  D5,A6    ;end address in A6
1800            MOVE.L  (SP)+,D5
1801
1802 BTSTDATALOOP:
1803            CMP.L   A5,A6
1804            BLT     READ
1805            MOVE.B  D5,(A5)+
1806            BRA     BTSTDATALOOP
1807
1808
1809 READ:
1810            CMP.L   A4,A6
1811            BLT     COMPLETE
1812            CMP.B   (A4)+,D5
1813            BNE     FAIL
1814            BRA     READ
1815
1816 FAIL:
1817            LEA     BTST4,A1
1818            MOVE.L  #11,D1
1819            MOVE.L  #0,D0
```

```
1820            TRAP    #15
1821
1822            LEA     BTST1,A1
1823            MOVE.L  #1,D0
1824            MOVE.L  #20,D1
1825            TRAP    #15
1826
1827            MOVE.B  D5,D3    ;for subroutine
1828            JSR     HEXTOASCII
1829            MOVE.L      A2,A1
1830            MOVE.L  #0,D0
1831            SUBA.L  A2,A3    ;number of bytes
1832            MOVE.L  A3,D1
1833            TRAP    #15
1834
1835
1836            LEA     BTST2,A1
1837            MOVE.L  #1,D0
1838            MOVE.L  #17,D1
1839            TRAP    #15
1840
1841
1842            SUB.L   #1,A4    ;go back to address that failed
1843            MOVE.B  (A4),D3
1844            JSR     HEXTOASCII  ;convert for output
1845            MOVE.L      A2,A1
1846            MOVE.L  #0,D0
1847            SUBA.L  A2,A3    ;number of bytes
1848            MOVE.L  A3,D1
1849            TRAP    #15
1850
1851            LEA     BTST5,A1
1852            MOVE.L  #27,D1
1853            MOVE.B  #1,D0
1854            TRAP    #15
1855            MOVE.L  A4,D3
1856            JSR     HEXTOASCII
1857            MOVE.L      A2,A1
1858            MOVE.L  #0,D0
1859            SUBA.L  A2,A3    ;number of bytes
1860            MOVE.L  A3,D1
1861            TRAP    #15
1862
1863
1864
```

```
1865 COMPLETE:
1866
1867            LEA       BTST3,A1
1868            MOVE.L    #18,D1
1869            MOVE.L    #0,D0
1870            TRAP      #15
1871            BRA RESTORE
```

2.2.10 Block Search

2.2.10.1 Algorithm and Flowchart

This command searches through a block of memory for data entered by the user. It does so by checking each value in memory byte by byte. The syntax for this command is `BSCH <data> <address1> <address2>`. The flowchart is shown in Figure 12.

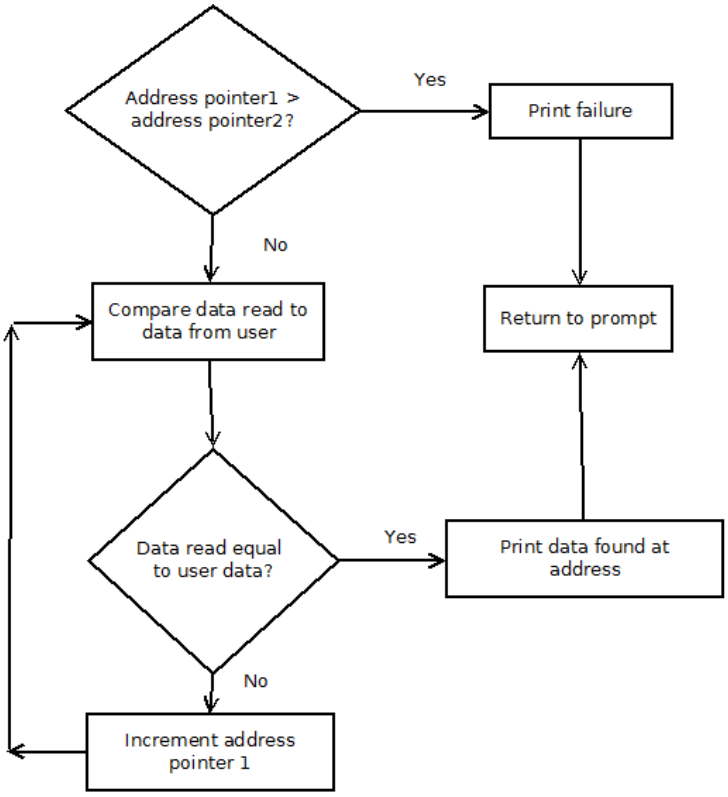

Figure 12: Flowchart for Block Search

2.2.10.2 Assembly Code

```
1875 BSCH:
1876            ADD.L    #1,A1     ;start of data
1877            MOVE.L   A1,A2     ;set up bac ptr
1878
1879 BSCHENDDATA:
1880            CMP.B    #$20,(A2)+
1881            BEQ      BSCHFIRSTADD
1882            BRA      BSCHENDDATA
1883
1884
1885 BSCHFIRSTADD:
1886            SUB.L    #1,A2
1887            MOVE.L   A2,A3
1888            MOVE.L   A1,A2
1889            JSR      ASCII_ADDRESS
1890            SUB.L    A1,A3     ;see how many bytes
1891            MOVE.L   A3,D6     ;store byte/word/long in D6
1892            ADD.L    #1,A2     ;set up for start of next address
1893            MOVE.L   A2,A3     ;set up for end ptr
1894            MOVE.L   D5,-(SP)      ;save data to stack
1895
1896
1897 BSCHFADDEND:
1898            CMP.B    #$20,(A3)+
1899            BEQ      BSCHSECONDADD
1900            BRA      BSCHFADDEND
1901
1902
1903 BSCHSECONDADD:
1904            SUB.L    #1,A3     ;off by one
1905            JSR      ASCII_ADDRESS
1906            MOVE.L   D5,A5     ;first address destination
1907            ADD.L    #1,A3     ;start it at next address
1908            MOVE.L   A3,A2     ; set up for next address
1909
1910
1911 BSCHSECONDFIND:
1912            CMP.B    #$00,(A3)+
1913            BEQ      TESTOP
1914            BRA      BSCHSECONDFIND
1915
1916
1917 TESTOP:
```

```
1918            SUB.L    #1,A3    ;off by one
1919            JSR      ASCII_ADDRESS
1920            MOVE.L   D5,A6    ;end address at A6
1921            MOVE.L   (SP)+,D5     ;restore data
1922            CMP.B    #2,D6
1923            BEQ      BYTEBSCH
1924            CMP.B    #4,D6
1925            BEQ      WORDBSCH
1926            CMP.B    #8,D6
1927            BEQ      LONGBSCH
1928            BRA      ERRORSR
1929
1930 BYTEBSCH:
1931            CMP.L    A5,A6
1932            BLT      ENDBSCH
1933            CMP.B    (A5)+,D5
1934            BEQ      FOUNDB
1935            BRA      BYTEBSCH
1936
1937 WORDBSCH:
1938            CMP.L    A5,A6
1939            BLT      ENDBSCH
1940            CMP.W    (A5)+,D5
1941            BEQ      FOUNDW
1942            BRA      WORDBSCH
1943
1944 LONGBSCH:
1945            CMP.L    A5,A6
1946            BLT      ENDBSCH
1947            CMP.L    (A5)+,D5
1948            BEQ      FOUNDL
1949            BRA      LONGBSCH
1950
1951
1952
1953 FOUNDB:
1954            SUB.L    #1,A5
1955            MOVE.B   (A5),D3
1956            BRA      SUCCESSTEXT
1957 FOUNDW:
1958            SUB.L    #2,A5
1959            MOVE.W   (A5),D3
1960            BRA      SUCCESSTEXT
1961 FOUNDL:
1962            SUB.L    #4,A5
```

```
1963            MOVE.L   (A5),D3
1964
1965 SUCCESSTEXT:
1966            LEA  BSCH1,A1
1967            MOVE.L   #6,D1
1968            MOVE.L   #1,D0
1969            TRAP     #15
1970
1971            JSR      HEXTOASCII
1972            MOVE.L   A2,A1
1973            SUB.L    A2,A3
1974            MOVE.L   A3,D1    ;how many bytes
1975            MOVE.L   #0,D0
1976            TRAP     #15
1977
1978            LEA  BSCH2,A1
1979            MOVE.L   #18,D1
1980            MOVE.L   #1,D0
1981            TRAP     #15
1982
1983            MOVE.L   A5,D3
1984            JSR      HEXTOASCII
1985            MOVE.L   A2,A1
1986            SUB.L    A2,A3
1987            MOVE.L   A3,D1    ;how many bytes
1988            MOVE.L   #0,D0
1989            TRAP     #15
1990
1991
1992 ENDBSCH:
1993            BRA RESTORE
```

2.2.11 Go

2.2.11.1 Algorithm and Flowchart

This command jumps to an address in memory and executes the machine code stored at that address. The syntax is `GO <address>`. The flowchart is shown in Figure 13.

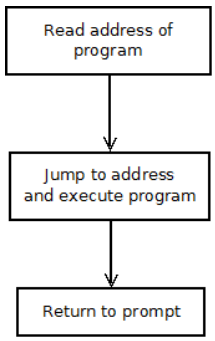

Figure 13: Flowchart for Go

2.2.11.2 Assembly Code

```
1997 GO:
1998            MOVE.L   A1,A2    ;setup for hex conversion
1999            MOVE.L   A2,A3
2000 GGETEND:
2001            CMP.B    #$00,(A3)+
2002            BEQ      EXECUTE
2003            BRA      GGETEND
2004
2005 EXECUTE:
2006            SUB.L    #1,A3    ;off by one error
2007            JSR      ASCII_ADDRESS
2008            MOVE.L   D5,A0
2009            JSR      (A0)     ;go to program
2010            **NOTE: THE PROGRAM MUST HAVE RTS OR CONTROL WILL NOT BE
        RETURNED BACK TO MONITOR441!!!**
2011            BRA RESTORE
```

2.2.12 Display Formatted Registers

2.2.12.1 Algorithm and Flowchart

This command displays the values of the registers as well as the stack pointers and program counter. It does so by first popping these values which were previously stored on stack item by item. They are then converted to ASCII for output and displayed on the terminal. The syntax is `DF`. The flowchart is shown in Figure 14.

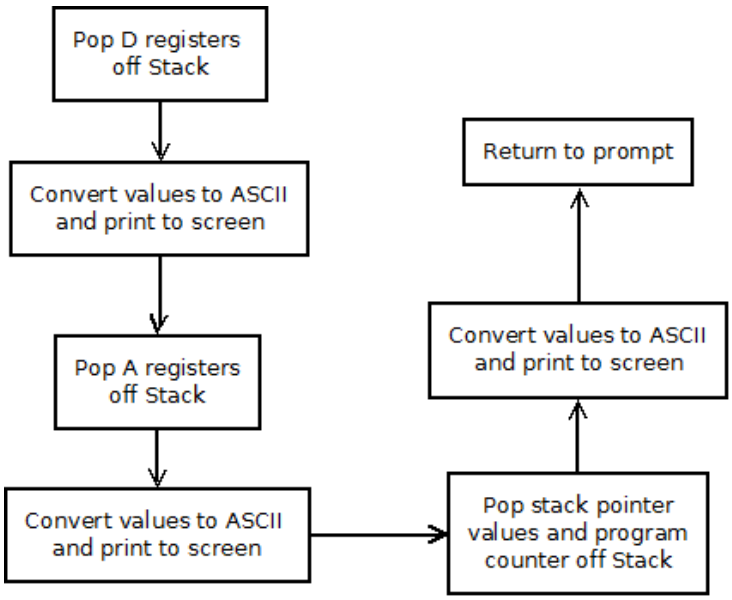

Figure 14: Flowchart for Display Formatted Registers

2.2.12.2 Assembly Code

```
2015 DF:    *Registers have already been saved to STACK, just need to
         pop them off first*
2016        *Stack looks like this*
2017
2018        *----------------*
2019        *|D0-D7/A0-A6|*
2020        *|      USP    |*
2021        *|      SR     |*
2022        *|      SSP    |*
2023        *|      PC     |*
2024        *----------------*
2025        *I should've used loops for efficiency but runtime is
         not a design constraint*
2026        *Maybe fix this in the future?*
2027
2028          *-----------D0-----------*
2029        LEA       RD0,A1
2030        MOVE.L    #4,D1
2031        MOVE.L    #1,D0
2032        TRAP      #15
2033        MOVE.L    (SP)+,D3
```

```
2034            JSR         HEXTOASCII
2035            MOVE.L      A2,A1
2036            SUB.L       A3,A2
2037            MOVE.L      A2,D2
2038            CMP.L       #-8,D2
2039            BEQ         D0DONTWORRY
2040 D0ACCOUNTFORZEROS:
2041              ADDI.L    #8,D2
2042              SUB.L     D2,A1
2043 D0DONTWORRY:
2044            MOVE.L      #0,D0
2045            MOVE.L      #8,D1
2046            TRAP        #15
2047
2048                  *-------------D1-------------*
2049            LEA         RD1,A1
2050            MOVE.L      #4,D1
2051            MOVE.L      #1,D0
2052            TRAP        #15
2053            MOVE.L      (SP)+,D3
2054            JSR         HEXTOASCII
2055            MOVE.L      A2,A1
2056            SUB.L       A3,A2
2057            MOVE.L      A2,D2
2058            CMP.L       #-8,D2
2059            BEQ         D1DONTWORRY
2060 D1ACCOUNTFORZEROS:
2061              ADDI.L    #8,D2
2062              SUB.L     D2,A1
2063 D1DONTWORRY:
2064            MOVE.L      #0,D0
2065            MOVE.L      #8,D1
2066            TRAP        #15
2067
2068                  *-------------D2-------------*
2069            LEA         RD2,A1
2070            MOVE.L      #4,D1
2071            MOVE.L      #1,D0
2072            TRAP        #15
2073            MOVE.L      (SP)+,D3
2074            JSR         HEXTOASCII
2075            MOVE.L      A2,A1
2076            SUB.L       A3,A2
2077            MOVE.L      A2,D2
2078            CMP.L       #-8,D2
```

```
2079            BEQ         D2DONTWORRY
2080 D2ACCOUNTFORZEROS:
2081              ADDI.L    #8,D2
2082              SUB.L     D2,A1
2083 D2DONTWORRY:
2084            MOVE.L      #0,D0
2085            MOVE.L      #8,D1
2086            TRAP        #15
2087
2088                    *-----------D3-----------*
2089            LEA         RD3,A1
2090            MOVE.L      #4,D1
2091            MOVE.L      #1,D0
2092            TRAP        #15
2093            MOVE.L      (SP)+,D3
2094            JSR         HEXTOASCII
2095            MOVE.L      A2,A1
2096            SUB.L       A3,A2
2097            MOVE.L      A2,D2
2098            CMP.L       #-8,D2
2099            BEQ         D3DONTWORRY
2100 D3ACCOUNTFORZEROS:
2101              ADDI.L    #8,D2
2102              SUB.L     D2,A1
2103 D3DONTWORRY:
2104            MOVE.L      #0,D0
2105            MOVE.L      #8,D1
2106            TRAP        #15
2107
2108                    *-----------D4-----------*
2109            LEA         RD4,A1
2110            MOVE.L      #4,D1
2111            MOVE.L      #1,D0
2112            TRAP        #15
2113            MOVE.L      (SP)+,D3
2114            JSR         HEXTOASCII
2115            MOVE.L      A2,A1
2116            SUB.L       A3,A2
2117            MOVE.L      A2,D2
2118            CMP.L       #-8,D2
2119            BEQ         D4DONTWORRY
2120 D4ACCOUNTFORZEROS:
2121              ADDI.L    #8,D2
2122              SUB.L     D2,A1
2123 D4DONTWORRY:
```

```
2124            MOVE.L      #0,D0
2125            MOVE.L      #8,D1
2126            TRAP        #15
2127
2128            *-----------D5-----------*
2129            LEA         RD5,A1
2130            MOVE.L      #4,D1
2131            MOVE.L      #1,D0
2132            TRAP        #15
2133            MOVE.L      (SP)+,D3
2134            JSR         HEXTOASCII
2135            MOVE.L      A2,A1
2136            SUB.L       A3,A2
2137            MOVE.L      A2,D2
2138            CMP.L       #-8,D2
2139            BEQ         D5DONTWORRY
2140 D5ACCOUNTFORZEROS:
2141              ADDI.L    #8,D2
2142              SUB.L     D2,A1
2143 D5DONTWORRY:
2144            MOVE.L      #0,D0
2145            MOVE.L      #8,D1
2146            TRAP        #15
2147
2148              *-----------D6-----------*
2149            LEA         RD6,A1
2150            MOVE.L      #4,D1
2151            MOVE.L      #1,D0
2152            TRAP        #15
2153            MOVE.L      (SP)+,D3
2154            JSR         HEXTOASCII
2155            MOVE.L      A2,A1
2156            SUB.L       A3,A2
2157            MOVE.L      A2,D2
2158            CMP.L       #-8,D2
2159            BEQ         D6DONTWORRY
2160 D6ACCOUNTFORZEROS:
2161              ADDI.L    #8,D2
2162              SUB.L     D2,A1
2163 D6DONTWORRY:
2164            MOVE.L      #0,D0
2165            MOVE.L      #8,D1
2166            TRAP        #15
2167
2168              *-----------D7-----------*
```

```
2169            LEA         RD7,A1
2170            MOVE.L      #4,D1
2171            MOVE.L      #1,D0
2172            TRAP        #15
2173            MOVE.L      (SP)+,D3
2174            JSR         HEXTOASCII
2175            MOVE.L      A2,A1
2176            SUB.L       A3,A2
2177            MOVE.L      A2,D2
2178            CMP.L       #-8,D2
2179            BEQ         D7DONTWORRY
2180 D7ACCOUNTFORZEROS:
2181              ADDI.L    #8,D2
2182              SUB.L     D2,A1
2183 D7DONTWORRY:
2184            MOVE.L      #0,D0
2185            MOVE.L      #8,D1
2186            TRAP        #15
2187
2188      *-----------A0-----------*
2189            LEA         RA0,A1
2190            MOVE.L      #4,D1
2191            MOVE.L      #1,D0
2192            TRAP        #15
2193            MOVE.L      (SP)+,D3
2194            JSR         HEXTOASCII
2195            MOVE.L      A2,A1
2196            SUB.L       A3,A2
2197            MOVE.L      A2,D2
2198            CMP.L       #-8,D2
2199            BEQ         A0DONTWORRY
2200 A0ACCOUNTFORZEROS:
2201              ADDI.L    #8,D2
2202              SUB.L     D2,A1
2203 A0DONTWORRY:
2204            MOVE.L      #0,D0
2205            MOVE.L      #8,D1
2206            TRAP        #15
2207
2208                *-----------A1-----------*
2209            LEA         RA1,A1
2210            MOVE.L      #4,D1
2211            MOVE.L      #1,D0
2212            TRAP        #15
2213            MOVE.L      (SP)+,D3
```

```
2214            JSR         HEXTOASCII
2215            MOVE.L      A2,A1
2216            SUB.L       A3,A2
2217            MOVE.L      A2,D2
2218            CMP.L       #-8,D2
2219            BEQ         A1DONTWORRY
2220 A1ACCOUNTFORZEROS:
2221              ADDI.L    #8,D2
2222              SUB.L     D2,A1
2223 A1DONTWORRY:
2224            MOVE.L      #0,D0
2225            MOVE.L      #8,D1
2226            TRAP        #15
2227
2228              *-----------A2-----------*
2229            LEA         RA2,A1
2230            MOVE.L      #4,D1
2231            MOVE.L      #1,D0
2232            TRAP        #15
2233            MOVE.L      (SP)+,D3
2234            JSR         HEXTOASCII
2235            MOVE.L      A2,A1
2236            SUB.L       A3,A2
2237            MOVE.L      A2,D2
2238            CMP.L       #-8,D2
2239            BEQ         A2DONTWORRY
2240 A2ACCOUNTFORZEROS:
2241              ADDI.L    #8,D2
2242              SUB.L     D2,A1
2243 A2DONTWORRY:
2244            MOVE.L      #0,D0
2245            MOVE.L      #8,D1
2246            TRAP        #15
2247
2248              *-----------A3-----------*
2249            LEA         RA3,A1
2250            MOVE.L      #4,D1
2251            MOVE.L      #1,D0
2252            TRAP        #15
2253            MOVE.L      (SP)+,D3
2254            JSR         HEXTOASCII
2255            MOVE.L      A2,A1
2256            SUB.L       A3,A2
2257            MOVE.L      A2,D2
2258            CMP.L       #-8,D2
```

```
2259            BEQ         A3DONTWORRY
2260 A3ACCOUNTFORZEROS:
2261              ADDI.L    #8,D2
2262              SUB.L     D2,A1
2263 A3DONTWORRY:
2264            MOVE.L      #0,D0
2265            MOVE.L      #8,D1
2266            TRAP        #15
2267
2268              *-----------A4-----------*
2269            LEA         RA3,A1
2270            MOVE.L      #4,D1
2271            MOVE.L      #1,D0
2272            TRAP        #15
2273            MOVE.L      (SP)+,D3
2274            JSR         HEXTOASCII
2275            MOVE.L      A2,A1
2276            SUB.L       A3,A2
2277            MOVE.L      A2,D2
2278            CMP.L       #-8,D2
2279            BEQ         A4DONTWORRY
2280 A4ACCOUNTFORZEROS:
2281              ADDI.L    #8,D2
2282              SUB.L     D2,A1
2283 A4DONTWORRY:
2284            MOVE.L      #0,D0
2285            MOVE.L      #8,D1
2286            TRAP        #15
2287
2288             *-----------A5-----------*
2289            LEA         RA3,A1
2290            MOVE.L      #4,D1
2291            MOVE.L      #1,D0
2292            TRAP        #15
2293            MOVE.L      (SP)+,D3
2294            JSR         HEXTOASCII
2295            MOVE.L      A2,A1
2296            SUB.L       A3,A2
2297            MOVE.L      A2,D2
2298            CMP.L       #-8,D2
2299            BEQ         A5DONTWORRY
2300 A5ACCOUNTFORZEROS:
2301              ADDI.L    #8,D2
2302              SUB.L     D2,A1
2303 A5DONTWORRY:
```

```
2304            MOVE.L      #0,D0
2305            MOVE.L      #8,D1
2306            TRAP        #15
2307
2308            *-----------A6-----------*
2309            LEA         RA3,A1
2310            MOVE.L      #4,D1
2311            MOVE.L      #1,D0
2312            TRAP        #15
2313            MOVE.L      (SP)+,D3
2314            JSR         HEXTOASCII
2315            MOVE.L      A2,A1
2316            SUB.L       A3,A2
2317            MOVE.L      A2,D2
2318            CMP.L       #-8,D2
2319            BEQ         A6DONTWORRY
2320 A6ACCOUNTFORZEROS:
2321              ADDI.L    #8,D2
2322              SUB.L     D2,A1
2323 A6DONTWORRY:
2324            MOVE.L      #0,D0
2325            MOVE.L      #8,D1
2326            TRAP        #15
2327          *-----HACK---*
2328          ADD.L   #60,SP   ;should put stack in correct place
2329
2330                      *-----------USP-----------*
2331            LEA         RUS,A1
2332            MOVE.L      #4,D1
2333            MOVE.L      #1,D0
2334            TRAP        #15
2335            MOVE.L      (SP)+,D3
2336            JSR         HEXTOASCII
2337            MOVE.L      A2,A1
2338            SUB.L       A3,A2
2339            MOVE.L      A2,D2
2340            CMP.L       #-8,D2
2341            BEQ         USPDONTWORRY
2342 USPACCOUNTFORZEROS:
2343              ADDI.L    #8,D2
2344              SUB.L     D2,A1
2345 USPDONTWORRY:
2346            MOVE.L      #0,D0
2347            MOVE.L      #8,D1
2348            TRAP        #15
```

```
2349
2350                    *-------------SR-------------*
2351            LEA         RSR,A1
2352            MOVE.L      #4,D1
2353            MOVE.L      #1,D0
2354            TRAP        #15
2355            MOVE.W      (SP)+,D3
2356            MOVE.W      D3,D7    ;temp storage to restore before return
2357            JSR         HEXTOASCII
2358            MOVE.L      A2,A1
2359            SUB.L       A3,A2
2360            MOVE.L      A2,D2
2361            CMP.L       #-4,D2
2362            BEQ         SRDONTWORRY
2363 SRACCOUNTFORZEROS:
2364              ADDI.L    #4,D2
2365              SUB.L     D2,A1
2366 SRDONTWORRY:
2367            MOVE.L      #0,D0
2368            MOVE.L      #4,D1
2369            TRAP        #15
2370
2371        *-------------SS/A7-------------*
2372            LEA         RSS,A1
2373            MOVE.L      #7,D1
2374            MOVE.L      #1,D0
2375            TRAP        #15
2376            MOVE.L      (SP)+,D3
2377            JSR         HEXTOASCII
2378            MOVE.L      A2,A1
2379            SUB.L       A3,A2
2380            MOVE.L      A2,D2
2381            CMP.L       #-8,D2
2382            BEQ         SSDONTWORRY
2383 SSACCOUNTFORZEROS:
2384              ADDI.L    #8,D2
2385              SUB.L     D2,A1
2386 SSDONTWORRY:
2387            MOVE.L      #0,D0
2388            MOVE.L      #8,D1
2389            TRAP        #15
2390
2391        *-------------PC-------------*
2392            LEA         RPC,A1
2393            MOVE.L      #4,D1
```

```
2394            MOVE.L      #1,D0
2395            TRAP        #15
2396            MOVE.L      (SP)+,D3
2397            JSR         HEXTOASCII
2398            MOVE.L      A2,A1
2399            SUB.L       A3,A2
2400            MOVE.L      A2,D2
2401            CMP.L       #-8,D2
2402            BEQ         PCDONTWORRY
2403 PCACCOUNTFORZEROS:
2404              ADDI.L    #8,D2
2405              SUB.L     D2,A1
2406 PCDONTWORRY:
2407            MOVE.L      #0,D0
2408            MOVE.L      #8,D1
2409            TRAP        #15
2410
2411     *---DF HACK RESTORE---*
2412     MOVE.W     D7,-(SP)
2413     ADD.L      #-72,SP
2414     MOVEM.L    (SP)+,D0-D7/A0-A6
2415     ADD.L      #12,SP   ;go back to original value
2416     ;MOVE.W    (SP)+,SR
2417     ORI.W    #$2000,SR     ;easy68k simulator is always in
          supervisor mode
2418     MOVE.L   #$01000000,SP    ;reset stack
2419          BRA SHELL
```

2.2.13 Modify Register

2.2.13.1 Algorithm and Flowchart

This command is used to change the value of a specific A or D register. This is done by parsing the data entered by the user, then updating the current value of the selected register. The syntax is `.<Register Type> <data>`. The flowchart is shown in Figure 15.

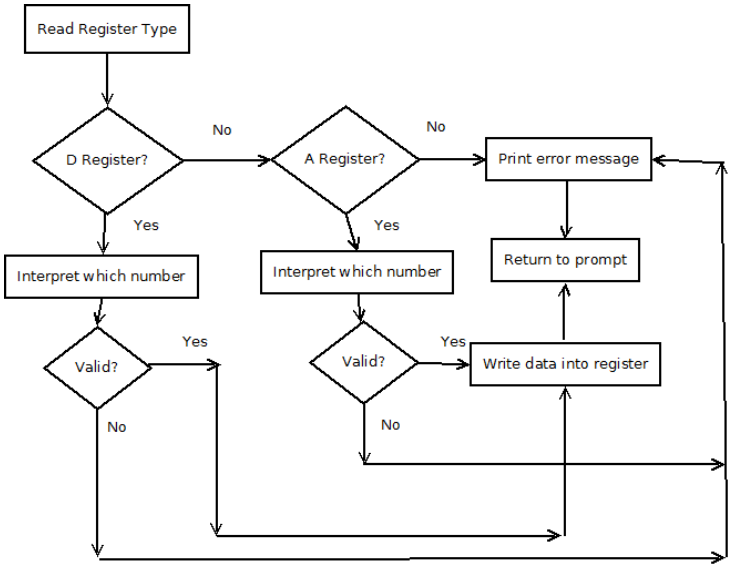

Figure 15: Flowchart for Modify Register

2.2.13.2 Assembly Code

```
414 MODIFYREGS:
415
416 MRD:
417             ADD.L    #1,A1     ;inc
418             CMP.B    #$30,(A1)
419             BEQ      MRD0
420             CMP.B    #$31,(A1)
421             BEQ      MRD1
422             CMP.B    #$32,(A1)
423             BEQ      MRD2
424             CMP.B    #$33,(A1)
425             BEQ      MRD3
426             CMP.B    #$34,(A1)
427             BEQ      MRD4
428             CMP.B    #$35,(A1)
429             BEQ      MRD5
430             CMP.B    #$36,(A1)
431             BEQ      MRD6
432             CMP.B    #$37,(A1)
433             BEQ      MRD7
434             BRA      ERRORSR
```

```
435
436 MRA:
437             ADD.L    #1,A1    ;inc
438             CMP.B    #$30,(A1)
439             BEQ      MRA0
440             CMP.B    #$31,(A1)
441             BEQ      MRA1
442             CMP.B    #$32,(A1)
443             BEQ      MRA2
444             CMP.B    #$33,(A1)
445             BEQ      MRA3
446             CMP.B    #$34,(A1)
447             BEQ      MRA4
448             CMP.B    #$35,(A1)
449             BEQ      MRA5
450             CMP.B    #$36,(A1)
451             BEQ      MRA6
452             BRA      ERRORSR
453
454
455
456
457
458 MRD0:
459             ADD.L    #1,A1
460             CMP.B    #$20,(A1)+
461             BNE      ERRORSR
462             MOVE.L   A1,A2
463             MOVE.L   A2,A3
464             JSR      MRDFINDDATA
465             SUB.L    #1,A3
466             JSR      ASCII_ADDRESS    ;convert data to hex
467             MOVE.L   D5,-(SP)         ;store it temporarily
468             ADD.L    #4,SP        ;dont lose data
469             MOVEM.L  (SP)+,D0-D7/A0-A6
470             MOVEM.L  (SP)+,D0-D7/A0-A6 ;double restore because of DF
        hack workaround
471             ADD.L    #4,SP        ;account for USP, it'll fix itself (
        it shouldn't be used)
472                                       ;EASY68k simulator starts in
        supervisor mode
473             MOVE     (SP)+,SR
474             ADD.L    #4,SP        ;skip saved stack
475             SUB.L    #134,SP      ;find data again
476             MOVE.L   (SP),D0
```

```
477         ADD.L   #138,SP      ;go back to original spot
478         BRA     SHELL
479
480 MRD1:
481         ADD.L   #1,A1
482         CMP.B   #$20,(A1)+
483         BNE     ERRORSR
484         MOVE.L  A1,A2
485         MOVE.L  A2,A3
486         JSR     MRDFINDDATA
487         SUB.L   #1,A3
488         JSR     ASCII_ADDRESS    ;convert data to hex
489         MOVE.L  D5,-(SP)         ;store it temporarily
490         ADD.L   #4,SP        ;dont lose data
491         MOVEM.L (SP)+,D0-D7/A0-A6
492         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
493         ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
494                                  ;EASY68k simulator starts in
    supervisor mode
495         MOVE    (SP)+,SR
496         ADD.L   #4,SP        ;skip saved stack
497         SUB.L   #134,SP      ;find data again
498         MOVE.L  (SP),D1
499         ADD.L   #138,SP      ;go back to original spot
500         BRA     SHELL
501
502 MRD2:
503         ADD.L   #1,A1
504         CMP.B   #$20,(A1)+
505         BNE     ERRORSR
506         MOVE.L  A1,A2
507         MOVE.L  A2,A3
508         JSR     MRDFINDDATA
509         SUB.L   #1,A3
510         JSR     ASCII_ADDRESS    ;convert data to hex
511         MOVE.L  D5,-(SP)         ;store it temporarily
512         ADD.L   #4,SP        ;dont lose data
513         MOVEM.L (SP)+,D0-D7/A0-A6
514         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
515         ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
```

```
516                                     ;EASY68k simulator starts in
    supervisor mode
517         MOVE    (SP)+,SR
518         ADD.L   #4,SP           ;skip saved stack
519         SUB.L   #134,SP         ;find data again
520         MOVE.L  (SP),D2
521         ADD.L   #138,SP         ;go back to original spot
522         BRA     SHELL
523
524 MRD3:
525         ADD.L   #1,A1
526         CMP.B   #$20,(A1)+
527         BNE     ERRORSR
528         MOVE.L  A1,A2
529         MOVE.L  A2,A3
530         JSR     MRDFINDDATA
531         SUB.L   #1,A3
532         JSR     ASCII_ADDRESS   ;convert data to hex
533         MOVE.L  D5,-(SP)        ;store it temporarily
534         ADD.L   #4,SP           ;dont lose data
535         MOVEM.L (SP)+,D0-D7/A0-A6
536         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
537         ADD.L   #4,SP           ;account for USP, it'll fix itself (
    it shouldn't be used)
538                                     ;EASY68k simulator starts in
    supervisor mode
539         MOVE    (SP)+,SR
540         ADD.L   #4,SP           ;skip saved stack
541         SUB.L   #134,SP         ;find data again
542         MOVE.L  (SP),D3
543         ADD.L   #138,SP         ;go back to original spot
544         BRA     SHELL
545
546 MRD4:
547         ADD.L   #1,A1
548         CMP.B   #$20,(A1)+
549         BNE     ERRORSR
550         MOVE.L  A1,A2
551         MOVE.L  A2,A3
552         JSR     MRDFINDDATA
553         SUB.L   #1,A3
554         JSR     ASCII_ADDRESS   ;convert data to hex
555         MOVE.L  D5,-(SP)        ;store it temporarily
556         ADD.L   #4,SP           ;dont lose data
```

```
557         MOVEM.L (SP)+,D0-D7/A0-A6
558         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
559         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
560                                 ;EASY68k simulator starts in
    supervisor mode
561         MOVE    (SP)+,SR
562         ADD.L   #4,SP       ;skip saved stack
563         SUB.L   #134,SP     ;find data again
564         MOVE.L  (SP),D4
565         ADD.L   #138,SP     ;go back to original spot
566         BRA     SHELL
567
568 MRD5:
569         ADD.L   #1,A1
570         CMP.B   #$20,(A1)+
571         BNE     ERRORSR
572         MOVE.L  A1,A2
573         MOVE.L  A2,A3
574         JSR     MRDFINDDATA
575         SUB.L   #1,A3
576         JSR     ASCII_ADDRESS   ;convert data to hex
577         MOVE.L  D5,-(SP)        ;store it temporarily
578         ADD.L   #4,SP       ;dont lose data
579         MOVEM.L (SP)+,D0-D7/A0-A6
580         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
581         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
582                                 ;EASY68k simulator starts in
    supervisor mode
583         MOVE    (SP)+,SR
584         ADD.L   #4,SP       ;skip saved stack
585         SUB.L   #134,SP     ;find data again
586         MOVE.L  (SP),D5
587         ADD.L   #138,SP     ;go back to original spot
588         BRA     SHELL
589
590 MRD6:
591         ADD.L   #1,A1
592         CMP.B   #$20,(A1)+
593         BNE     ERRORSR
594         MOVE.L  A1,A2
595         MOVE.L  A2,A3
```

```
596         JSR     MRDFINDDATA
597         SUB.L   #1,A3
598         JSR     ASCII_ADDRESS   ;convert data to hex
599         MOVE.L  D5,-(SP)        ;store it temporarily
600         ADD.L   #4,SP       ;dont lose data
601         MOVEM.L (SP)+,D0-D7/A0-A6
602         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
603         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
604                                 ;EASY68k simulator starts in
    supervisor mode
605         MOVE    (SP)+,SR
606         ADD.L   #4,SP       ;skip saved stack
607         SUB.L   #134,SP     ;find data again
608         MOVE.L  (SP),D6
609         ADD.L   #138,SP     ;go back to original spot
610         BRA     SHELL
611
612 MRD7:
613         ADD.L   #1,A1
614         CMP.B   #$20,(A1)+
615         BNE     ERRORSR
616         MOVE.L  A1,A2
617         MOVE.L  A2,A3
618         JSR     MRDFINDDATA
619         SUB.L   #1,A3
620         JSR     ASCII_ADDRESS   ;convert data to hex
621         MOVE.L  D5,-(SP)        ;store it temporarily
622         ADD.L   #4,SP       ;dont lose data
623         MOVEM.L (SP)+,D0-D7/A0-A6
624         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
625         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
626                                 ;EASY68k simulator starts in
    supervisor mode
627         MOVE    (SP)+,SR
628        ADD.L   #4,SP       ;skip saved stack
629         SUB.L   #134,SP     ;find data again
630         MOVE.L  (SP),D7
631         ADD.L   #138,SP     ;go back to original spot
632         BRA     SHELL
633
634 MRA0:
```

```
635         ADD.L   #1,A1
636         CMP.B   #$20,(A1)+
637         BNE     ERRORSR
638         MOVE.L  A1,A2
639         MOVE.L  A2,A3
640         JSR     MRDFINDDATA
641         SUB.L   #1,A3
642         JSR     ASCII_ADDRESS   ;convert data to hex
643         MOVE.L  D5,-(SP)        ;store it temporarily
644         ADD.L   #4,SP       ;dont lose data
645         MOVEM.L (SP)+,D0-D7/A0-A6
646         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
647         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
648                                 ;EASY68k simulator starts in
    supervisor mode
649         MOVE    (SP)+,SR
650         ADD.L   #4,SP       ;skip saved stack
651         SUB.L   #134,SP     ;find data again
652         MOVE.L  (SP),A0
653         ADD.L   #138,SP     ;go back to original spot
654         BRA     SHELL
655 MRA1:
656         ADD.L   #1,A1
657         CMP.B   #$20,(A1)+
658         BNE     ERRORSR
659         MOVE.L  A1,A2
660         MOVE.L  A2,A3
661         JSR     MRDFINDDATA
662         SUB.L   #1,A3
663         JSR     ASCII_ADDRESS   ;convert data to hex
664         MOVE.L  D5,-(SP)        ;store it temporarily
665         ADD.L   #4,SP       ;dont lose data
666         MOVEM.L (SP)+,D0-D7/A0-A6
667         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
668         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
669                                 ;EASY68k simulator starts in
    supervisor mode
670         MOVE    (SP)+,SR
671        ADD.L   #4,SP       ;skip saved stack
672         SUB.L   #134,SP     ;find data again
673         MOVE.L  (SP),A1
```

```
674         ADD.L   #138,SP      ;go back to original spot
675         BRA     SHELL
676
677 MRA2:
678         ADD.L   #1,A1
679         CMP.B   #$20,(A1)+
680         BNE     ERRORSR
681         MOVE.L  A1,A2
682         MOVE.L  A2,A3
683         JSR     MRDFINDDATA
684         SUB.L   #1,A3
685         JSR     ASCII_ADDRESS    ;convert data to hex
686         MOVE.L  D5,-(SP)         ;store it temporarily
687         ADD.L   #4,SP        ;dont lose data
688         MOVEM.L (SP)+,D0-D7/A0-A6
689         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
690         ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
691                                  ;EASY68k simulator starts in
    supervisor mode
692         MOVE    (SP)+,SR
693         ADD.L   #4,SP        ;skip saved stack
694         SUB.L   #134,SP      ;find data again
695         MOVE.L  (SP),A2
696         ADD.L   #138,SP      ;go back to original spot
697         BRA     SHELL
698
699 MRA3:
700         ADD.L   #1,A1
701         CMP.B   #$20,(A1)+
702         BNE     ERRORSR
703         MOVE.L  A1,A2
704         MOVE.L  A2,A3
705         JSR     MRDFINDDATA
706         SUB.L   #1,A3
707         JSR     ASCII_ADDRESS    ;convert data to hex
708         MOVE.L  D5,-(SP)         ;store it temporarily
709         ADD.L   #4,SP        ;dont lose data
710         MOVEM.L (SP)+,D0-D7/A0-A6
711         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
712         ADD.L   #4,SP        ;account for USP, it'll fix itself (
    it shouldn't be used)
```

```
713                                     ;EASY68k simulator starts in
    supervisor mode
714         MOVE    (SP)+,SR
715         ADD.L   #4,SP           ;skip saved stack
716         SUB.L   #134,SP         ;find data again
717         MOVE.L  (SP),A3
718         ADD.L   #138,SP         ;go back to original spot
719         BRA     SHELL
720
721 MRA4:
722         ADD.L   #1,A1
723         CMP.B   #$20,(A1)+
724         BNE     ERRORSR
725         MOVE.L  A1,A2
726         MOVE.L  A2,A3
727         JSR     MRDFINDDATA
728         SUB.L   #1,A3
729         JSR     ASCII_ADDRESS   ;convert data to hex
730         MOVE.L  D5,-(SP)        ;store it temporarily
731         ADD.L   #4,SP           ;dont lose data
732         MOVEM.L (SP)+,D0-D7/A0-A6
733         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
734         ADD.L   #4,SP           ;account for USP, it'll fix itself (
    it shouldn't be used)
735                                     ;EASY68k simulator starts in
    supervisor mode
736         MOVE    (SP)+,SR
737         ADD.L   #4,SP           ;skip saved stack
738         SUB.L   #134,SP         ;find data again
739         MOVE.L  (SP),A4
740         ADD.L   #138,SP         ;go back to original spot
741         BRA     SHELL
742
743 MRA5:
744         ADD.L   #1,A1
745         CMP.B   #$20,(A1)+
746         BNE     ERRORSR
747         MOVE.L  A1,A2
748         MOVE.L  A2,A3
749         JSR     MRDFINDDATA
750         SUB.L   #1,A3
751         JSR     ASCII_ADDRESS   ;convert data to hex
752         MOVE.L  D5,-(SP)        ;store it temporarily
753         ADD.L   #4,SP           ;dont lose data
```

```
754         MOVEM.L (SP)+,D0-D7/A0-A6
755         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
756         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
757                                 ;EASY68k simulator starts in
    supervisor mode
758         MOVE    (SP)+,SR
759        ADD.L   #4,SP       ;skip saved stack
760         SUB.L   #134,SP     ;find data again
761         MOVE.L  (SP),A5
762         ADD.L   #138,SP     ;go back to original spot
763         BRA     SHELL
764
765 MRA6:
766         ADD.L   #1,A1
767         CMP.B   #$20,(A1)+
768         BNE     ERRORSR
769         MOVE.L  A1,A2
770         MOVE.L  A2,A3
771         JSR     MRDFINDDATA
772         SUB.L   #1,A3
773         JSR     ASCII_ADDRESS   ;convert data to hex
774         MOVE.L  D5,-(SP)        ;store it temporarily
775         ADD.L   #4,SP       ;dont lose data
776         MOVEM.L (SP)+,D0-D7/A0-A6
777         MOVEM.L (SP)+,D0-D7/A0-A6 ;double restore because of DF
    hack workaround
778         ADD.L   #4,SP       ;account for USP, it'll fix itself (
    it shouldn't be used)
779                                 ;EASY68k simulator starts in
    supervisor mode
780         MOVE    (SP)+,SR
781         ADD.L   #4,SP       ;skip saved stack
782         SUB.L   #134,SP     ;find data again
783         MOVE.L  (SP),A6
784         ADD.L   #138,SP     ;go back to original spot
785         BRA     SHELL
786
787 MRDFINDDATA:
788         CMP.B   #$00,(A3)+
789         BEQ     GOBACK
790         BRA     MRDFINDDATA
791 GOBACK: RTS
792
```

```
793
794             BRA RESTORE
```

2.2.14 Echo

2.2.14.1 Algorithm and Flowchart

This is a simple command that outputs what the user inputs. This is done by parsing the data entered by the user and immediately setting up a trap I/O call that outputs what was just entered. The syntax is `ECHO <data>`. The flowchart is shown in Figure 16.

2.2.14.2 Assembly Code

```
399 ECHO: *What terminal DOESN'T have echo?*
400
401             MOVE.L  A1,A2    ;setup to find end of string
402 EEND:       CMP.B   #$00,(A2)+
403             BEQ     EFOUND
404             BRA     EEND
405 EFOUND:
406             SUB.L   #1,A2    ;off by one
407             SUB.L   A1,A2    ;find out how many bytes
408             MOVE.L  A2,D1    ;place it for trap function
409             MOVE.L  #0,D0
410             TRAP    #15
411
412             BRA RESTORE
```

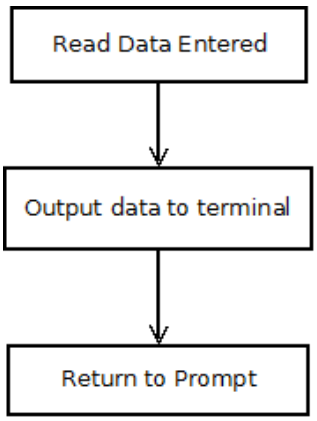

Figure 16: Flowchart for Echo

2.3 Exception Handlers

The Monitor441 program uses custom exception handlers. They are loaded using the source code:

```
134         *Load custom exceptions*
135         LEA BERR,A1 ;init exception handlers
136         MOVE.L A1,$8
137         LEA AERR,A1
138         MOVE.L A1,$C
139         LEA IERR,A1
140         MOVE.L A1,$10
141         LEA ZERR,A1
142         MOVE.L A1,$14
143         LEA CHKERR,A1
144         MOVE.L  A1,$18
145         LEA PERR,A1
146         MOVE.L A1,$20
147         LEA ALERR,A1
148         MOVE.L A1,$28
149         LEA FLERR,A1
150         MOVE.L A1,$2C
151         MOVEM.L (SP)+,D0-D2/A1  ;restore any preset values
```

2.3.1 Bus Error Exception

2.3.1.1 Algorithm and Flowchart

This exception is called whenever a bus error exception occurs. It outputs the SSW, IR, and BA along with a custom string message. The register values are also printed to the screen. The flowchart is shown in Figure 17.

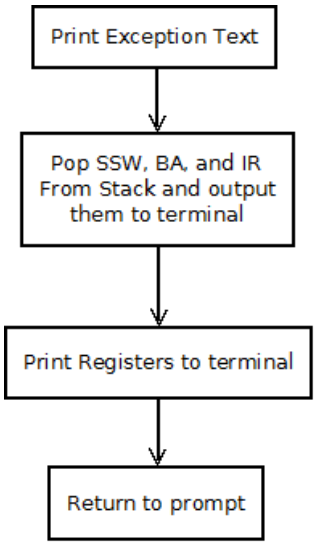

Figure 17: Flowchart for Bus Error Exception

2.3.1.2 Assembly Code

```
2427 BERR:
2428            MOVEM.L  A1-A3/D0-D1,-(SP)
2429            LEA      BERR_TEXT,A1
2430            MOVE.L   #13,D0
2431            TRAP     #15
2432            LEA      SSW,A1
2433            MOVE.L   #14,D0
2434            TRAP     #15
2435            MOVE.W   (28,SP),D3
2436            JSR      HEXTOASCII
2437            SUB.L    #4,A3
2438            MOVEA.L  A3,A1
2439            MOVE.L   #4,D1
2440            MOVE.L   #0,D0
2441            TRAP     #15
2442            LEA      BA,A1
2443            MOVE.L   #14,D0
```

```
2444            TRAP      #15
2445            MOVE.L    (30,SP),D3
2446            JSR       HEXTOASCII
2447            SUB.L     #8,A3
2448            MOVEA.L   A3,A1
2449            MOVE.L    #8,D1
2450            MOVE.L    #0,D0
2451            TRAP      #15
2452            LEA       IR,A1
2453            MOVE.L    #14,D0
2454            TRAP      #15
2455            MOVE.W    (34,SP),D3
2456            JSR       HEXTOASCII
2457            SUB.L     #4,A3
2458            MOVEA.L   A3,A1
2459            MOVE.L    #4,D1
2460            MOVE.L    #0,D0
2461            TRAP      #15
2462            MOVEM.L   (SP)+,A1-A3/D0-D1
2463
2464            JSR       DF
2465            BRA       SHELL
```

2.3.2 Address Error Exception

2.3.2.1 Algorithm and Flowchart

This exception is called whenever an address error exception occurs. It outputs the SSW, IR, and BA along with a custom string message. The register values are also printed to the screen. The flowchart is shown in Figure 18.

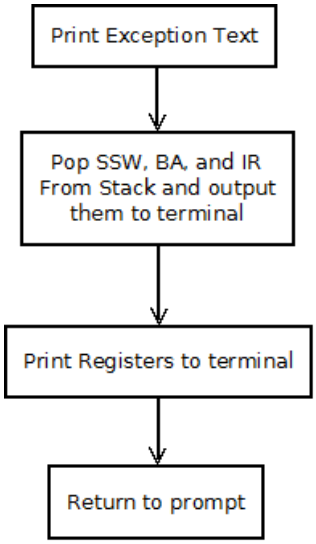

Figure 18: Flowchart for Address Error Exception

2.3.2.2 Assembly Code

```
2467 AERR:
2468            MOVEM.L  A1-A3/D0-D1/D4-D5,-(SP)
2469            LEA      AERR_TEXT,A1
2470            MOVE.L   #13,D0
2471            TRAP     #15
2472            LEA      SSW,A1
2473            MOVE.L   #14,D0
2474            TRAP     #15
2475            MOVE.W   (28,SP),D3
2476            JSR      HEXTOASCII
2477            SUB.L    #4,A3
2478            MOVEA.L  A3,A1
2479            MOVE.L   #4,D1
2480            MOVE.L   #0,D0
2481            TRAP     #15
2482            LEA      BA,A1
2483            MOVE.L   #14,D0
2484            TRAP     #15
2485            MOVE.L   (30,SP),D3
2486            JSR      HEXTOASCII
2487            SUB.L    #8,A3
2488            MOVEA.L  A3,A1
```

```
2489            MOVE.L   #8,D1
2490            MOVE.L   #0,D0
2491            TRAP     #15
2492            LEA      IR,A1
2493            MOVE.L   #14,D0
2494            TRAP     #15
2495            MOVE.W   (34,SP),D3
2496            JSR      HEXTOASCII
2497            SUB.L    #4,A3
2498            MOVEA.L  A3,A1
2499            MOVE.L   #4,D1
2500            MOVE.L   #0,D0
2501            TRAP     #15
2502            MOVEM.L  (SP)+,A1-A3/D0-D1/D4-D5
2503
2504            JSR      DF
2505            BRA      SHELL
```

2.3.3 Illegal Instruction Error Exception

2.3.3.1 Algorithm and Flowchart

This exception is called whenever an illegal instruction error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 19.

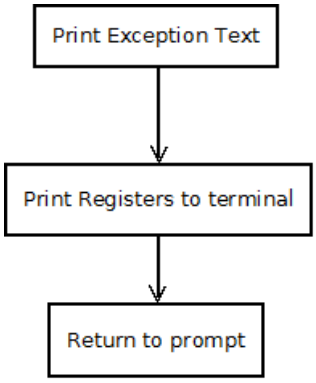

Figure 19: Flowchart for Illegal Instruction Exception

2.3.3.2 Assembly Code

```
2507 IERR:
2508     MOVEM.L  A1/D0,-(SP)
2509     LEA  IERR_TEXT,A1
2510     MOVE.L  #13,D0
2511     TRAP  #15
2512     MOVEM.L  (SP)+,A1/D0
2513     JSR  DF
2514     BRA  SHELL
```

2.3.4 Privilege Violation Error Exception

2.3.4.1 Algorithm and Flowchart

This exception is called whenever an privilege violation error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 20.

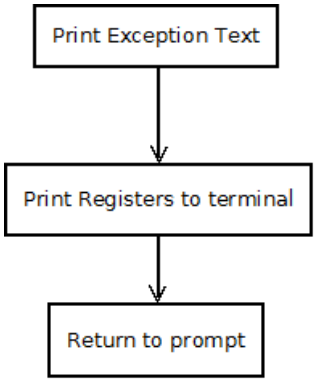

Figure 20: Flowchart for Privilege Violation Exception

2.3.4.2 Assembly Code

```
2516 PERR:
2517     MOVEM.L  A1/D0,-(SP)
2518     LEA  PERR_TEXT,A1
2519     MOVE.L  #13,D0
2520     TRAP  #15
2521     MOVEM.L  (SP)+,A1/D0
2522     JSR  DF
2523     BRA  SHELL
```

2.3.5 Divide by Zero Error Exception

2.3.5.1 Algorithm and Flowchart

This exception is called whenever a divide by zero error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 21.

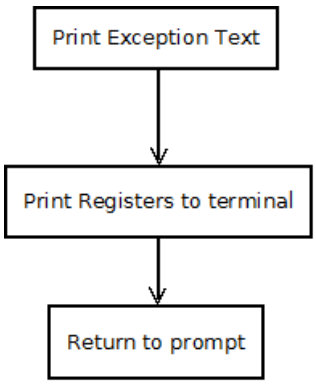

Figure 21: Flowchart for Divide by Zero Exception

2.3.5.2 Assembly Code

```
2525 ZERR:
2526     MOVEM.L  A1/D0,-(SP)
2527     LEA  ZERR_TEXT,A1
2528     MOVE.L  #13,D0
2529     TRAP  #15
2530     MOVEM.L  (SP)+,A1/D0
2531     JSR  DF
2532     BRA  SHELL
```

2.3.6 A Line Emulator Error Exception

2.3.6.1 Algorithm and Flowchart

This exception is called whenever an A line emulator error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 22.

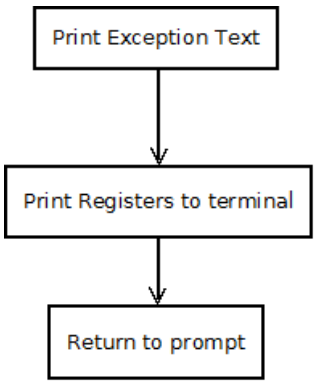

Figure 22: Flowchart for A Line Emulator Error Exception

2.3.6.2 Assembly Code

```
2534 ALERR:
2535        MOVEM.L  A1/D0,-(SP)
2536        LEA  ALERR_TEXT,A1
2537        MOVE.L  #13,D0
2538        TRAP  #15
2539        MOVEM.L  (SP)+,A1/D0
2540        JSR  DF
2541        BRA  SHELL
```

2.3.7 F Line Emulator Error Exception

2.3.7.1 Algorithm and Flowchart

This exception is called whenever an F line emulator error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 23.

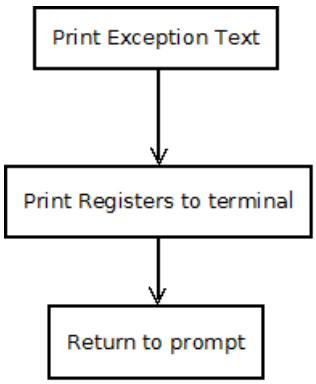

Figure 23: Flowchart for F Line Emulator Error Exception

2.3.7.2 Assembly Code

```
2543 FLERR:
2544        MOVEM.L  A1/D0,-(SP)
2545        LEA  FLERR_TEXT,A1
2546        MOVE.L  #13,D0
2547        TRAP  #15
2548        MOVEM.L  (SP)+,A1/D0
2549
2550        JSR  DF
2551        BRA  SHELL
```

2.3.8 Check Instruction Error Exception

2.3.8.1 Algorithm and Flowchart

This exception is called whenever a check instruction error exception occurs. It outputs a custom string message, and the register values are also printed to the screen. The flowchart is shown in Figure 24.

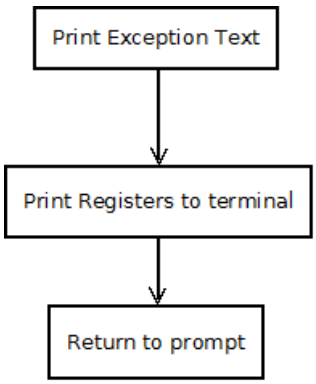

Figure 24: Flowchart for Check Instruction Error Exception

2.3.8.2 Assembly Code

```
2553 CHKERR:
2554        MOVEM.L  A1/D0,-(SP)
2555        LEA  CHKERR_TEXT,A1
2556        MOVE.L  #13,D0
2557        TRAP  #15
2558        MOVEM.L  (SP)+,A1/D0
2559
2560        JSR  DF
2561        BRA  SHELL
```

2.4 User Instruction Manual Exception Handlers

2.4.1 Syntax/Unknown Command Error

2.4.1.1 Algorithm and Flowchart

This error is meant to guide the user to input the correct syntax for a command. It first checks if the command entered is valid. If not, an unknown command message is displayed. If the command entered is valid but the syntax is incorrect, an incorrect syntax message is outputted to the terminal. The flowchart is shown in Figure 25.

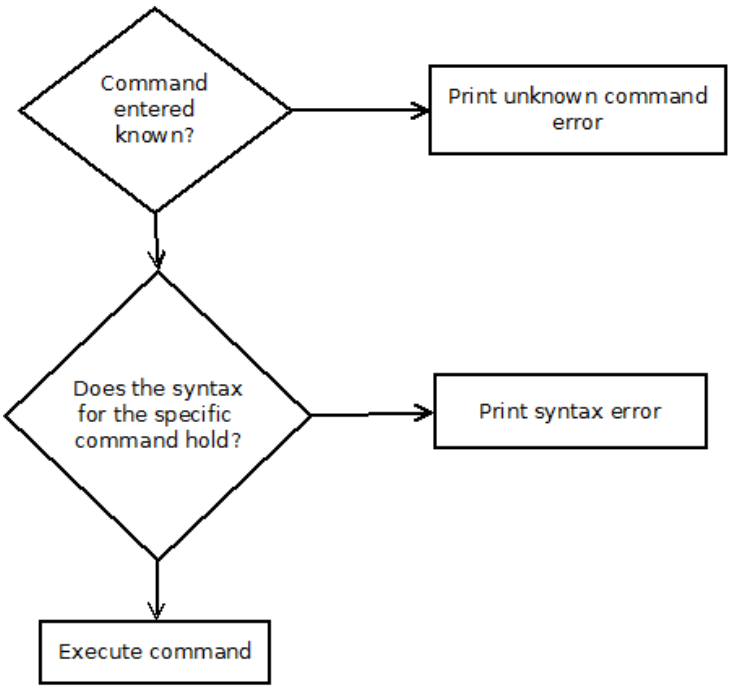

Figure 25: Flowchart for User Instruction Manual Exception Handler

2.4.1.2 Assembly Code

```
2569 ERRORSR:       LEA     ERROR, A1       ;load message
2570                MOVE.W  #44,D1
2571                MOVE.L  #0,D0
2572                TRAP    #15
2573                BRA     RESTORE

2667 UNKNOWNCMD:    LEA     ERROR1, A1      ;load message
2668                MOVE.W  #22,D1
2669                MOVE.L  #0,D0
2670                TRAP    #15
2671                BRA     RESTORE
```

3 Discussion

With a low level programming language such as assembly, the computer engineer is in full control of how each command is built. When building a

monitor type program, assembly can be great because of its simplicity, but it also has its draw backs due to a lack of high level API. This increases the chances that erroneous input will kill the program or produce unpredictable results. This means that in order to have a flawless program, the amount of source code needed to be written per command could easily violate the design constraint of having under 3K code size. Because of this, major errors were checked, but it is still possible to break the program by producing minor errors. The program assumes the user knows how to accurately use the tools provided, and if wrong input is entered and an error message is not displayed, the user will have already known their input was invalid. Furthermore, there are some major limitations to the functionality of this program, for example, certain commands can only take in byte sized data or a hexadecimal to decimal conversion can only accept a maximum value of FFFF. Regardless, this program performs on par with Motorola's Tutor software and, with error checking aside, could replace Tutor entirely.

There were many engineering and design challenges encountered during the journey to construct this program. A majority of these challenges came up during the debugging of each separate command. Taking an algorithm and knowing how to code it is a simple process, but the actual implementation is the hard part. This is when things such as runtime errors must be eliminated. Overall, runtime errors accounted for 95% of the total generated errors, and the code had to be run step by step to pinpoint the exact moment of error in order to be fixed. Furthermore, deciding on how to store and manipulate data was a huge design concern. With only 7 data registers and 7 address registers, storage "containers" had to be carefully picked so that there were no memory leaks and registers needed to be untouched due to subroutines were not accessed accidentally.

4 Feature Suggestions

As discussed earlier in Section 3, complex error checking should be implemented in this program. Furthermore, more commands should be implemented to emulate not just Motorola's Tutor software, but current Operating System distributions as well. This could include commands such as , `ls` (listing files in current directory) or `cd` (change directory). If implementing an embedded system using the MC68000 processor, and the `MONITOR441` program is used as a basis, these commands could help the programmer eas-

ily analyze top layer applications such as installed files as well as low layer applications such as displaying register values.

5 Conclusion

Overall, the monitor program was created, and it has all of the requested functionality implemented. While it is not 100% error free, it provides the user a great MC6000 based piece of software for use with debugging the microprocessor. Further work could be done to improve the functionality to be on the level of Motorola's Tutor software, but this shouldn't be done as no modern day technology runs based on the MC68000 microprocessor.

References

[1] Harman, Thomas L., and Barbara Lawson. *The Motorola MC68000 Microprocessor Family: Assembly Language, Interface Design, and System Design.* Englewood Cliffs, NJ: Prentice-Hall, 1985. Print.

[2] MC68000 Microprocessor Programmer's Reference Manual

[3] SANPER-1 Lab Manuals

[4] MC68000 Educational Computer Board User's Manual