Peter Chinetti

ECE 441 Monitor Project

December 3, 2013

Contents

1

Abstract

1.1 Background

The SANPER Educational Lab Unit is a Motorola 68k based microcomputer designed for use in college level computer engineering classes. It is equipped with some useful peripherals, such as ROM, SRAM, EEPROM, Serial and Parallel I/O, and an expansion board.

1.2 The Problem

To educate students about programming for the 68k processor, this monitor project was assigned. The project asks for students to re-implement a subset of the commands found in the TUTOR monitor that ships with the SANPER. The selection of commands is somewhat up to the student implementing the software, which for this project was:

- Modify Registers (.A*, .D*)
- Block Fill (BF)
- Block Move (BM)
- Block Search (BS)
- Block Test (BT)
- Data Conversion (DC)
- Display Formatted Registers (DF)

- Run Program [GO] (G)
- Help (H)
- Memory Display (MD)
- Memory Modify (MM)
- Memory Sort (MS)

2

Command Implementation

2.1 Modify Registers (.A*, .D*)

The Modify Register command was implemented by recognizing a '.' character in the zeroth spot of the input string, then switching on the second character, then further switching on the third character. If the second character is not an 'A' or a 'D', the program returns to the main prompt. If the third character is not in the range 0-7, the program returns to the main prompt.

2.2 Block Fill (BF)

The Block Fill command was implemented by recognizing a 'B' character in the zeroth spot of the input string, then a 'F' character in the first spot. The program then tried to decode three blocks of four characters into a start and stop address, as well as a fill word. After decoding, the program stepped word by word and copied the fill word to the destination. After completion of the transfer, the program returned to the main prompt.

2.3 Block Move (BM)

The Block Move command was implemented by recognizing a 'B' character in the zeroth spot of the input string, then a 'M' character in the first spot. The program then tried to decode three blocks of four characters into a start, stop and destination address. After decoding, the program stepped byte by byte and copied the range to the destination. After completion of the transfer, the program returned to the main prompt.

2.4 Block Search (BS)

The Block Search command was implemented by recognizing a 'B' character in the zeroth spot of the input string, then a 'S' character in the first spot. The program then tried to decode two blocks of four characters into a start and stop address. After decoding the address, the program decoded one or more groups of two characters into bytes to search for. After all decoding was finished, the program searched every byte in the range to see if it was the start of a matching sequence. If a match was found, a message was printed with the starting address of the match included. When done, the program returned to the main prompt.

2.5 Block Test (BT)

The Block Test command was implemented by recognizing a 'B' character in the zeroth spot of the input string, then a 'T' character in the first spot. The program then tried to decode two blocks of four characters into a start and stop address. After decoding, a destructive test occurred as the program wrote a word (A0A0) into memory, then checked to see that it was actually written by reading the memory. If there was a failure a message was printed. When the test was done, the program returned to the main prompt.

2.6 Data Conversion (DC)

The Data Conversion program switched on first 'D', then 'C'. Then, the next characters were parsed using a homemade ASCII-HEX to Binary conversion routine. Then the result was printed on the screen using a Trap #15 command. When done, the program returned to the main prompt.

2.7 Display Formatted Registers (DF)

The Display Formatted Registers command switched on first 'D', then 'F'. Then, the program stepped through a sequence of save registers to the stack, move a register to D1 to be printed, run a Trap #15 command, and restore

from the stack for every register in the processor. When done, the program returned to the main prompt.

2.8 Run Program [GO] (G)

The Go command switched on 'G', then decoded a jump address from the next four characters, then set the PC to that address.

2.9 Help (H)

The Help command switched on 'H', then printed a series of help messages. When done, the program returned to the main prompt.

2.10 Memory Display (MD)

The Memory Display command switched first on 'M', then on 'D'. Then, it decoded a four character address. Starting from that address, the program printed 0x10 bytes on one line, then accepted input. If the input was anything but '.', the program printed another 0x10 bytes on another line, and again waited for input. If the input was '.', the program was done and returned to the main prompt.

2.11 Memory Modify (MM)

The Memory Modify command switched first on 'M', then again on 'M'. Then, it decoded a four character address. Starting at that address, it printed the byte located there, then waited for two characters of input. If either of the characters were '.', it exited to the main prompt, otherwise it decoded the two characters from hex to binary and stored them at the address, then repeated for the next byte.

2.12 Memory Sort (MS)

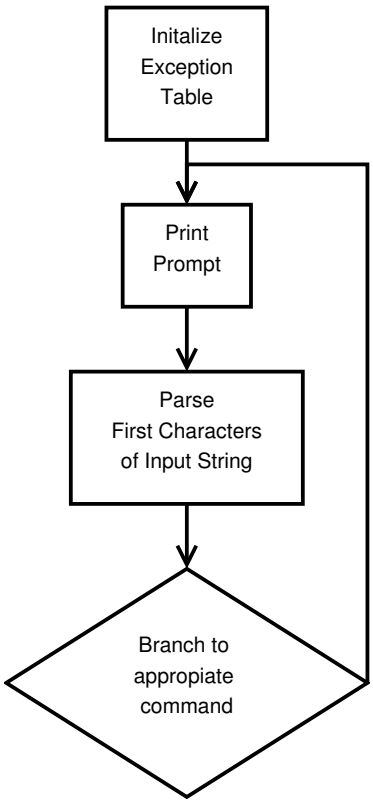
The Memory Sort command switched first on 'M', then on 'S'. Then, it decoded two groups of four character addresses, the start and end addresses.

Then it ran selection sort that range, iterating through the range byte by byte and replacing the current byte in the iteration with the smallest byte remaining in the search space. When done, the program returned to the main prompt.

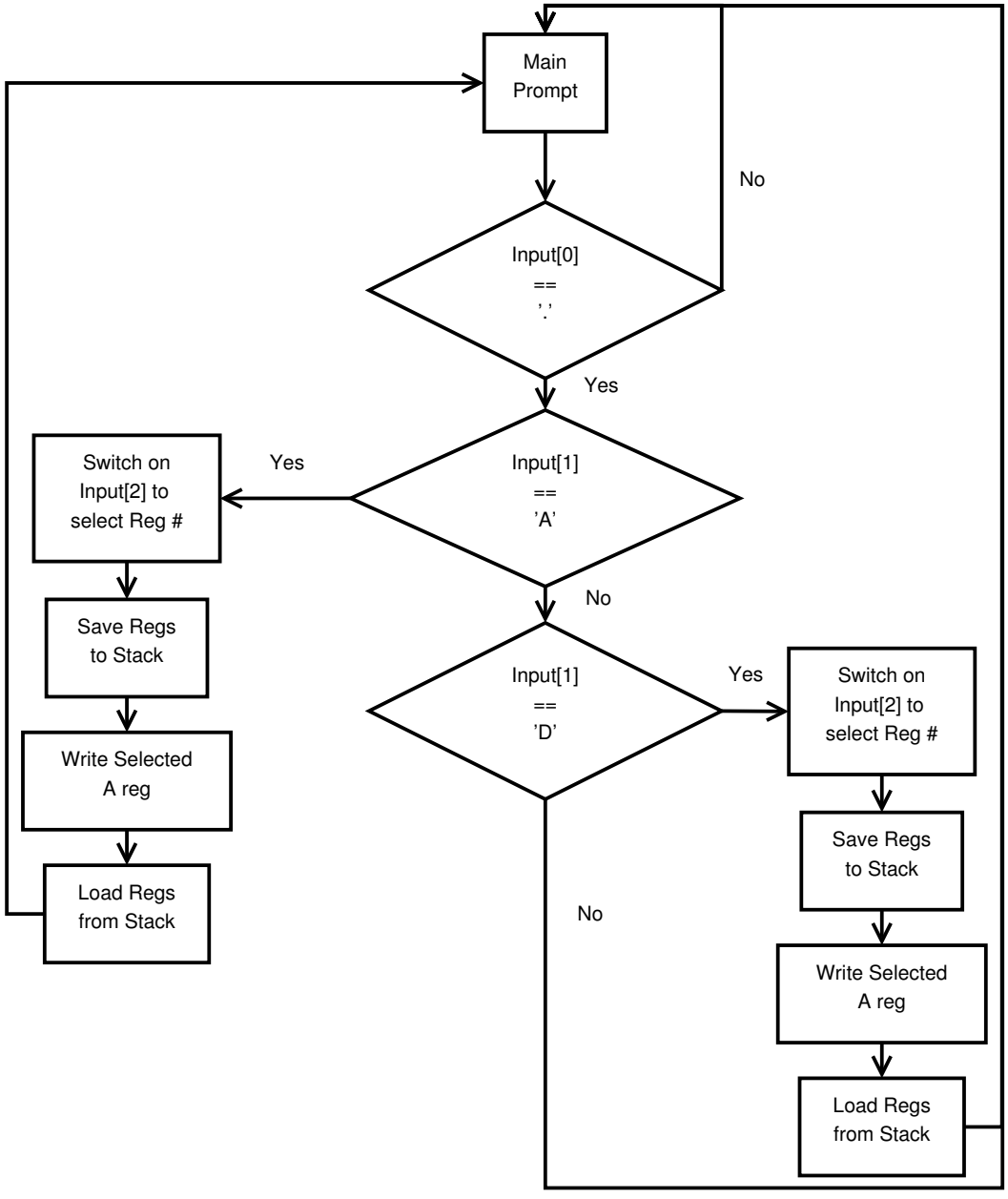
3

Flowcharts

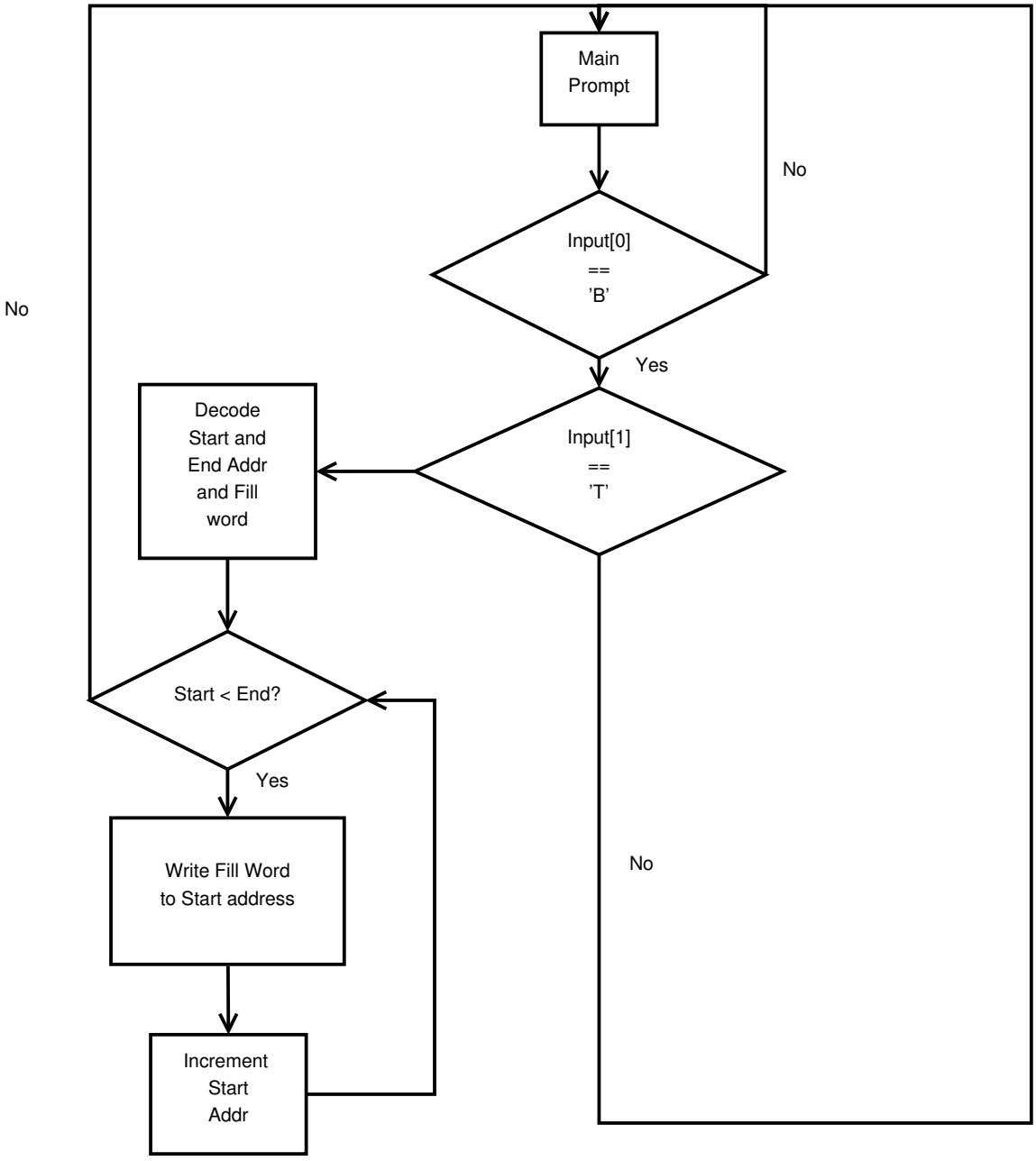
3.1 Command Interpreter



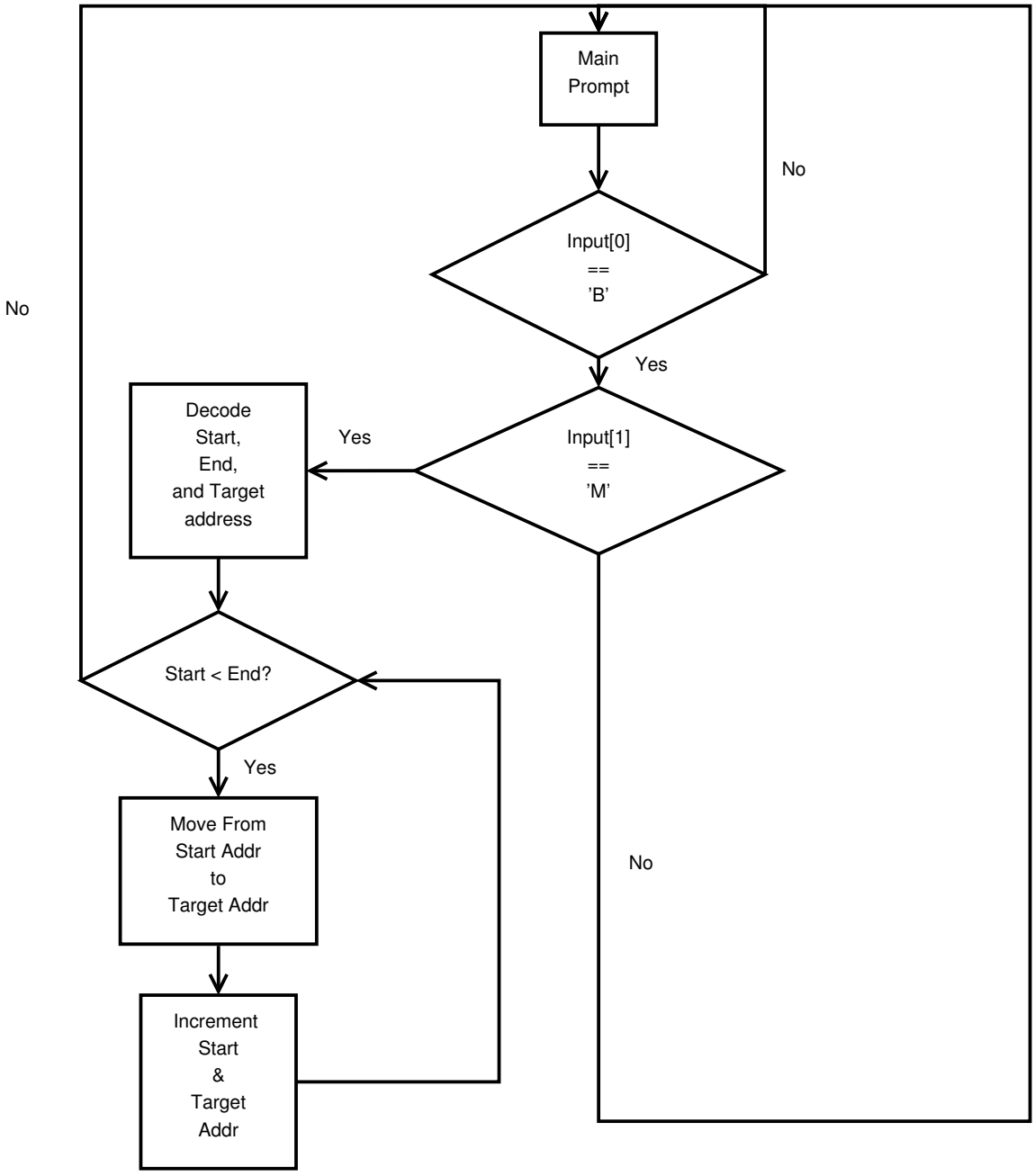
3.2 Modify Registers (.A*, .D*)



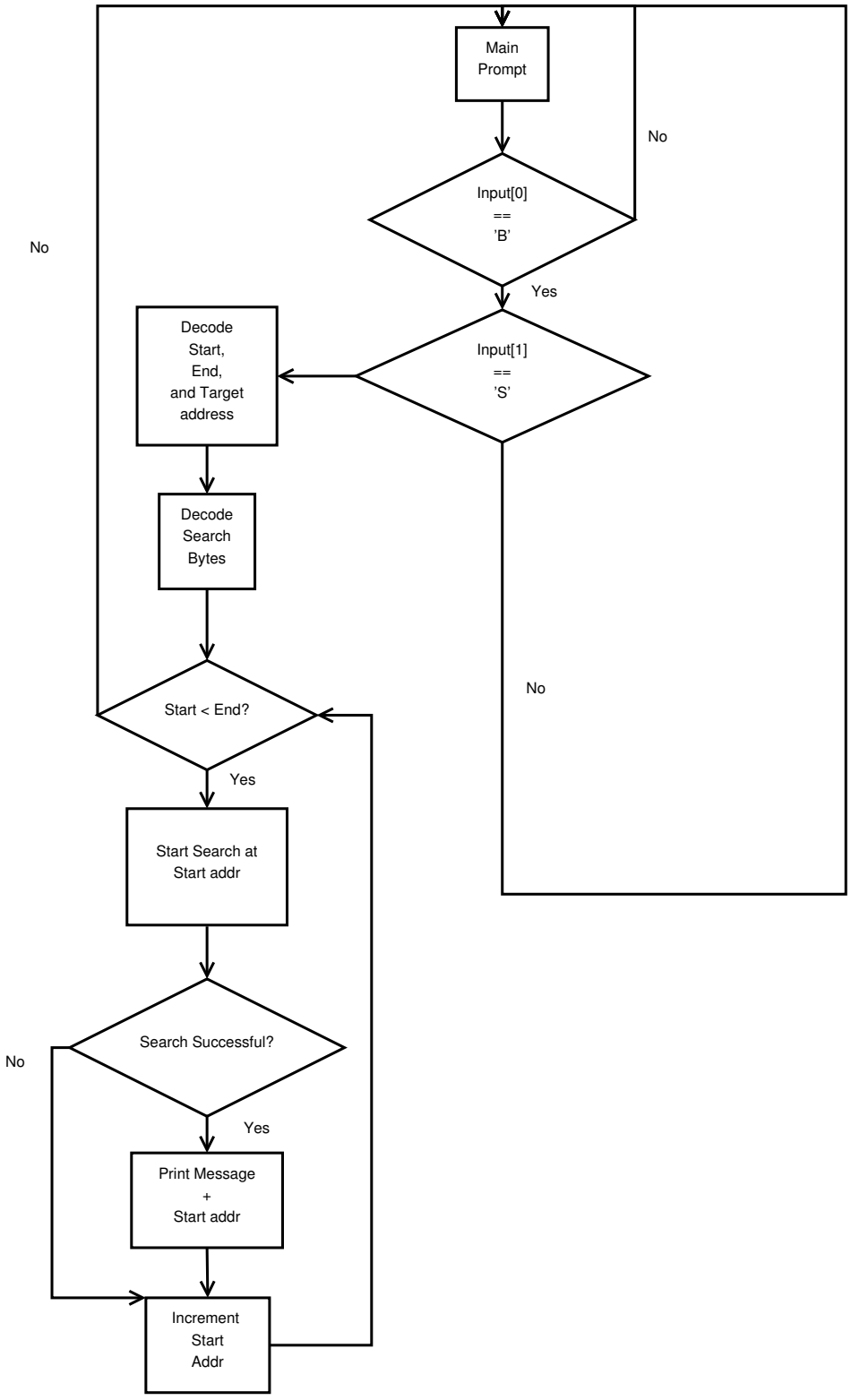
3.3 Block Fill (BF)



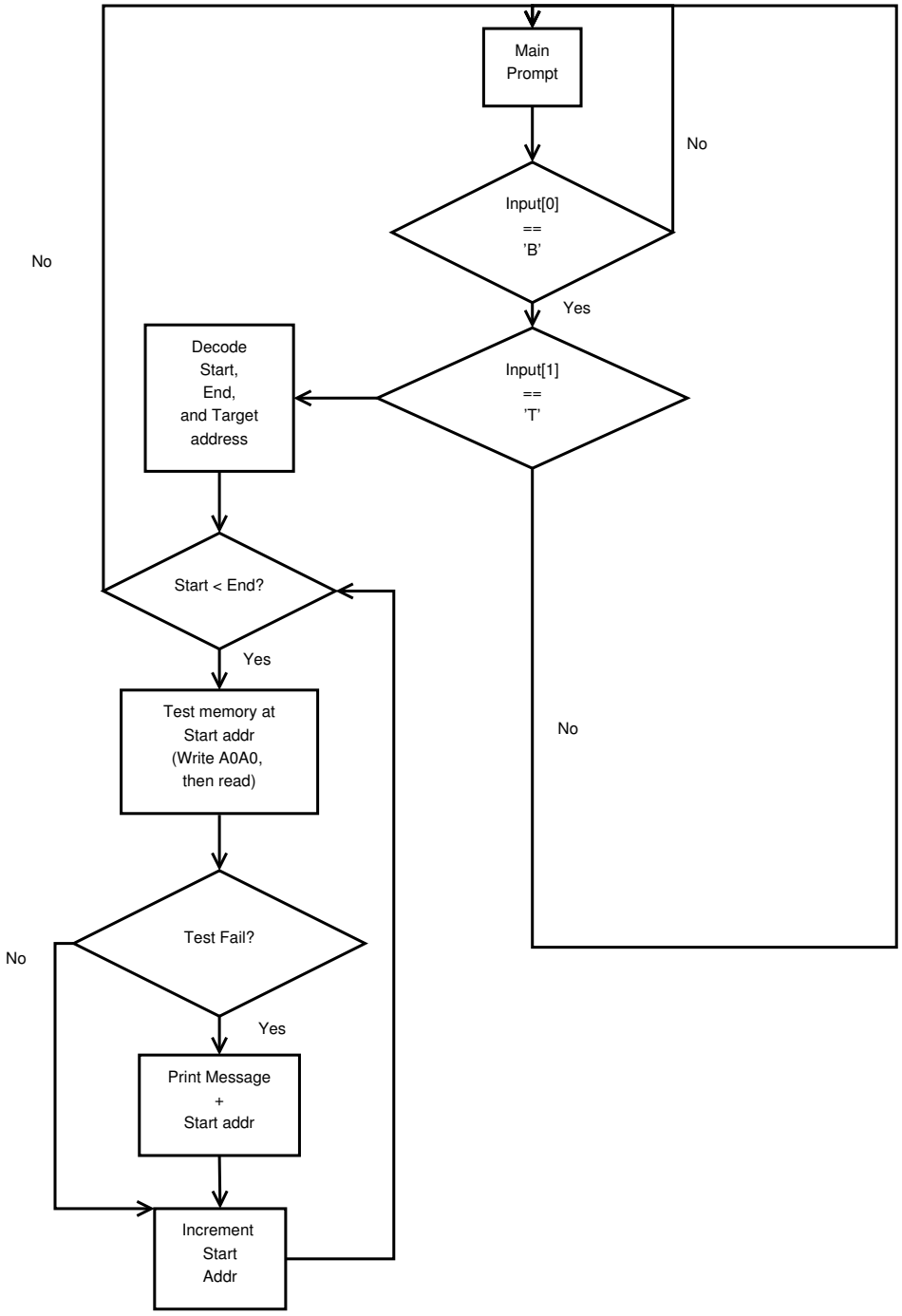
3.4 Block Move (BM)



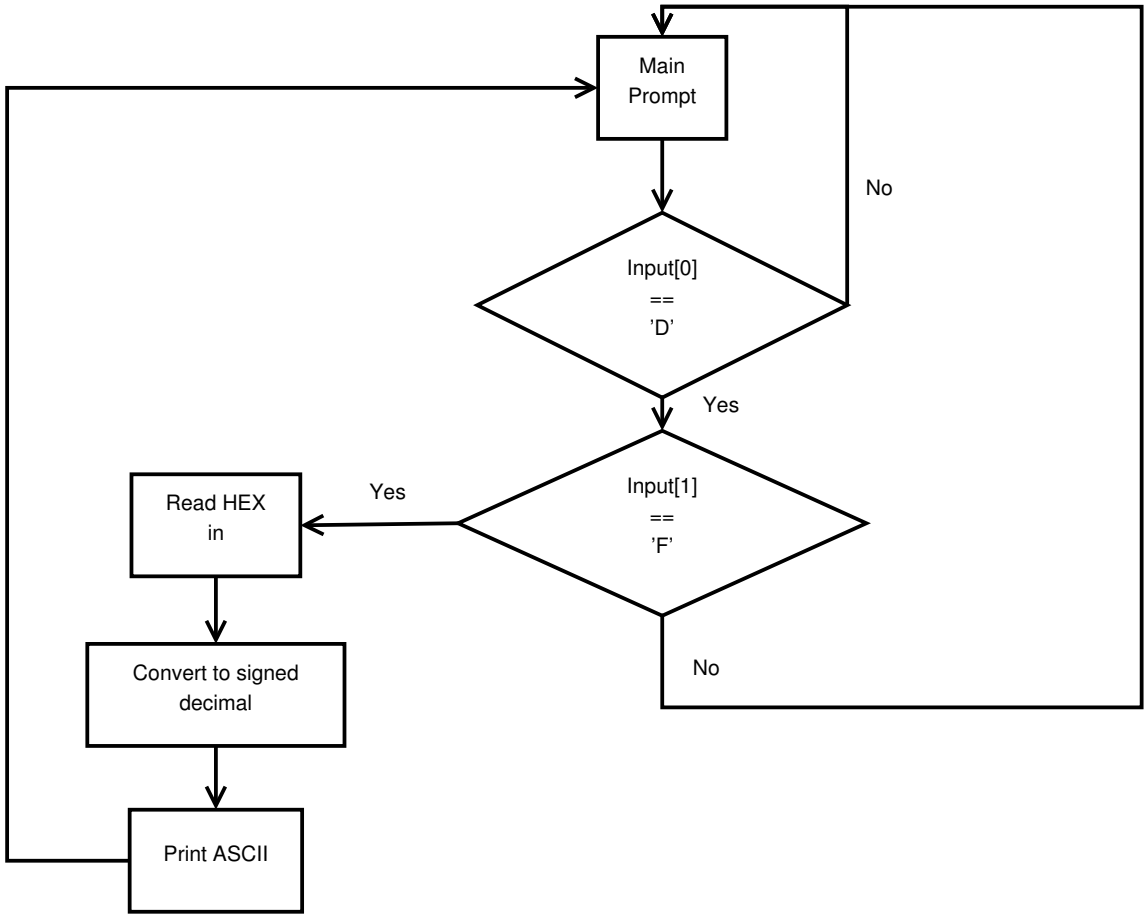
3.5 Block Search (BS)



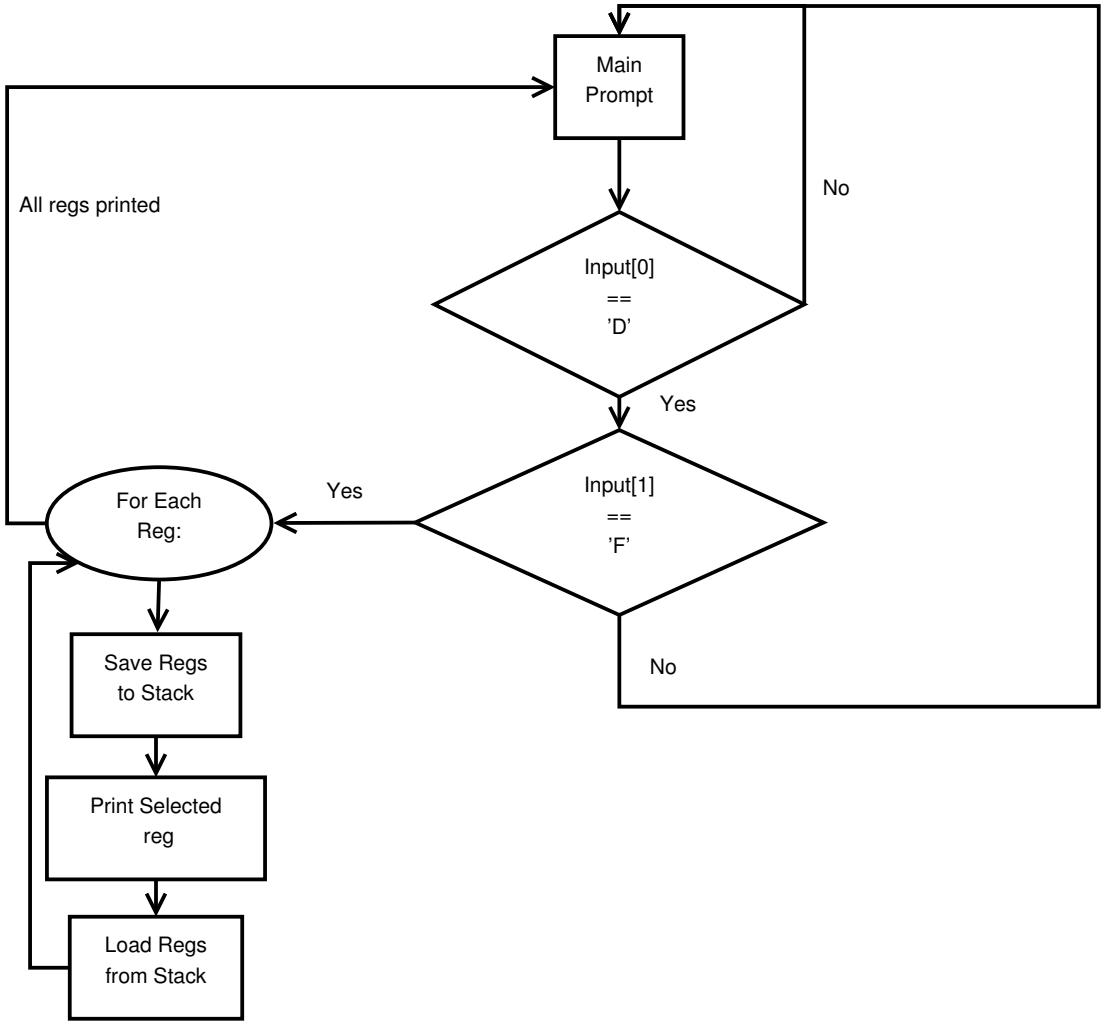
3.6 Block Test (BT)



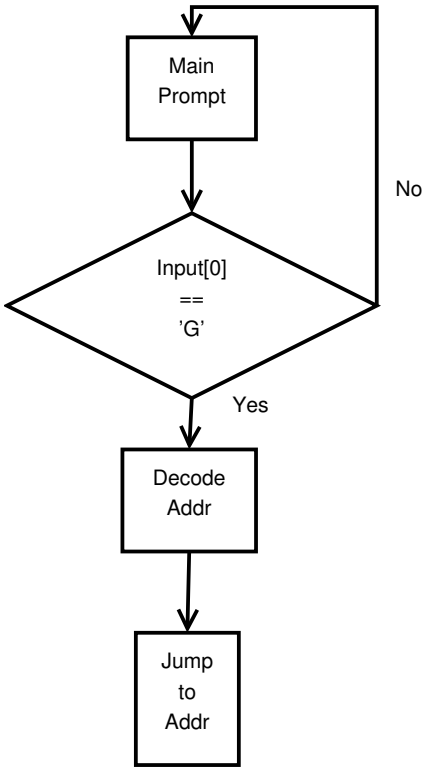
3.7 Data Conversion (DC)



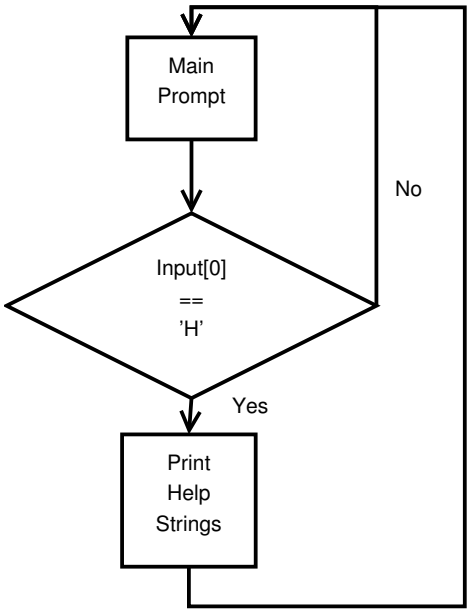
3.8 Display Formatted Registers (DF)



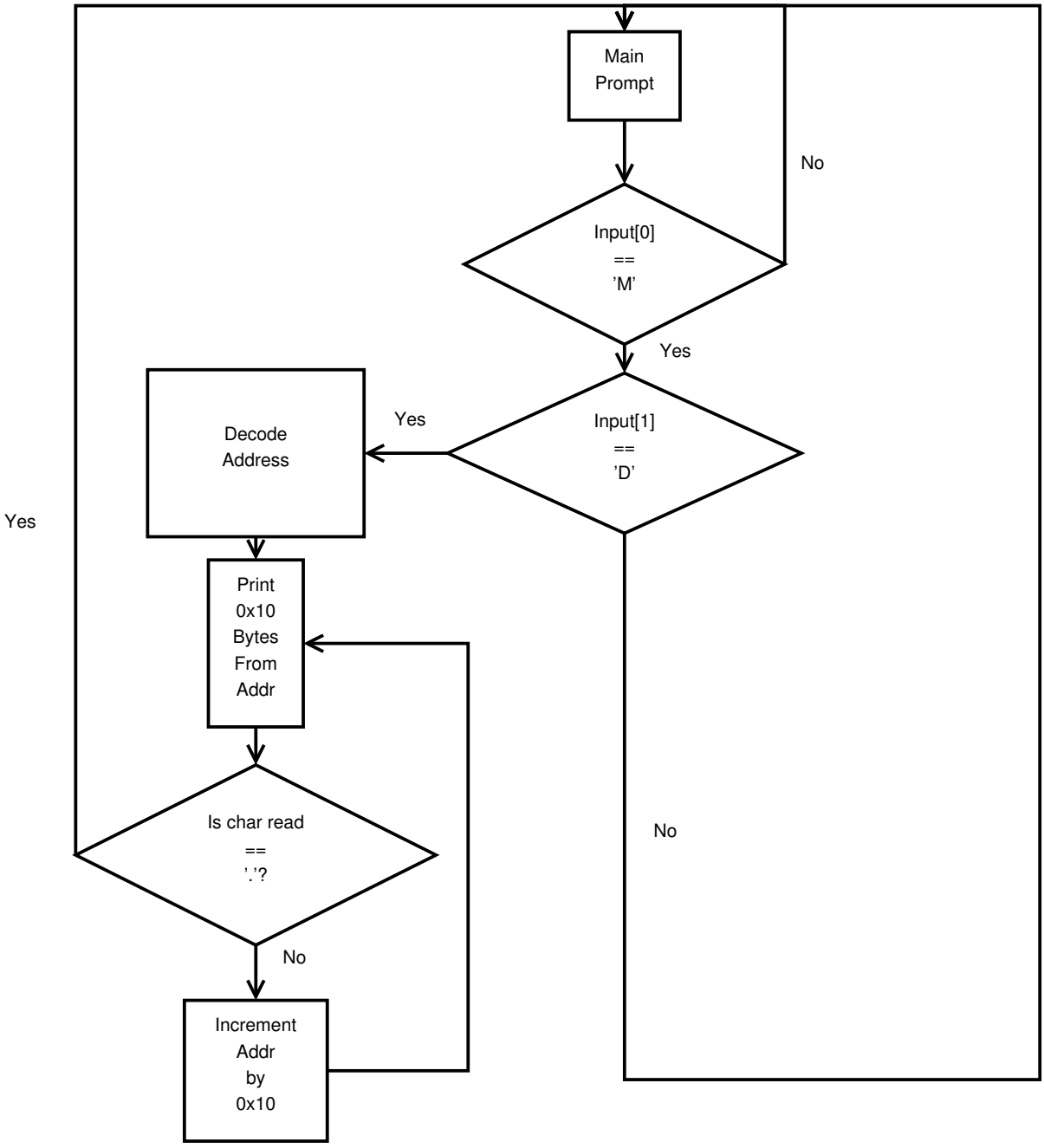
3.9 Run Program [GO] (G)



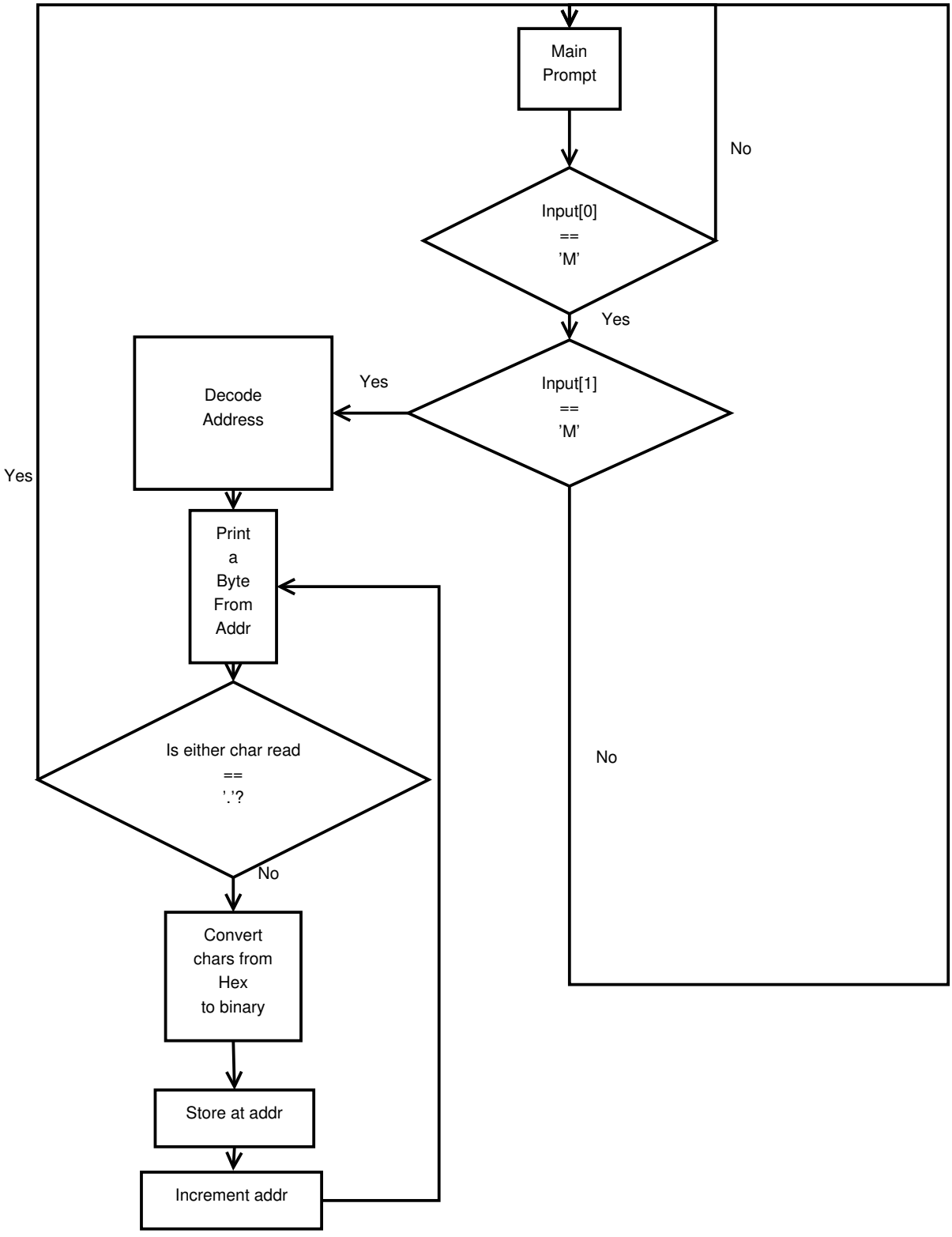
3.10 Help (H)



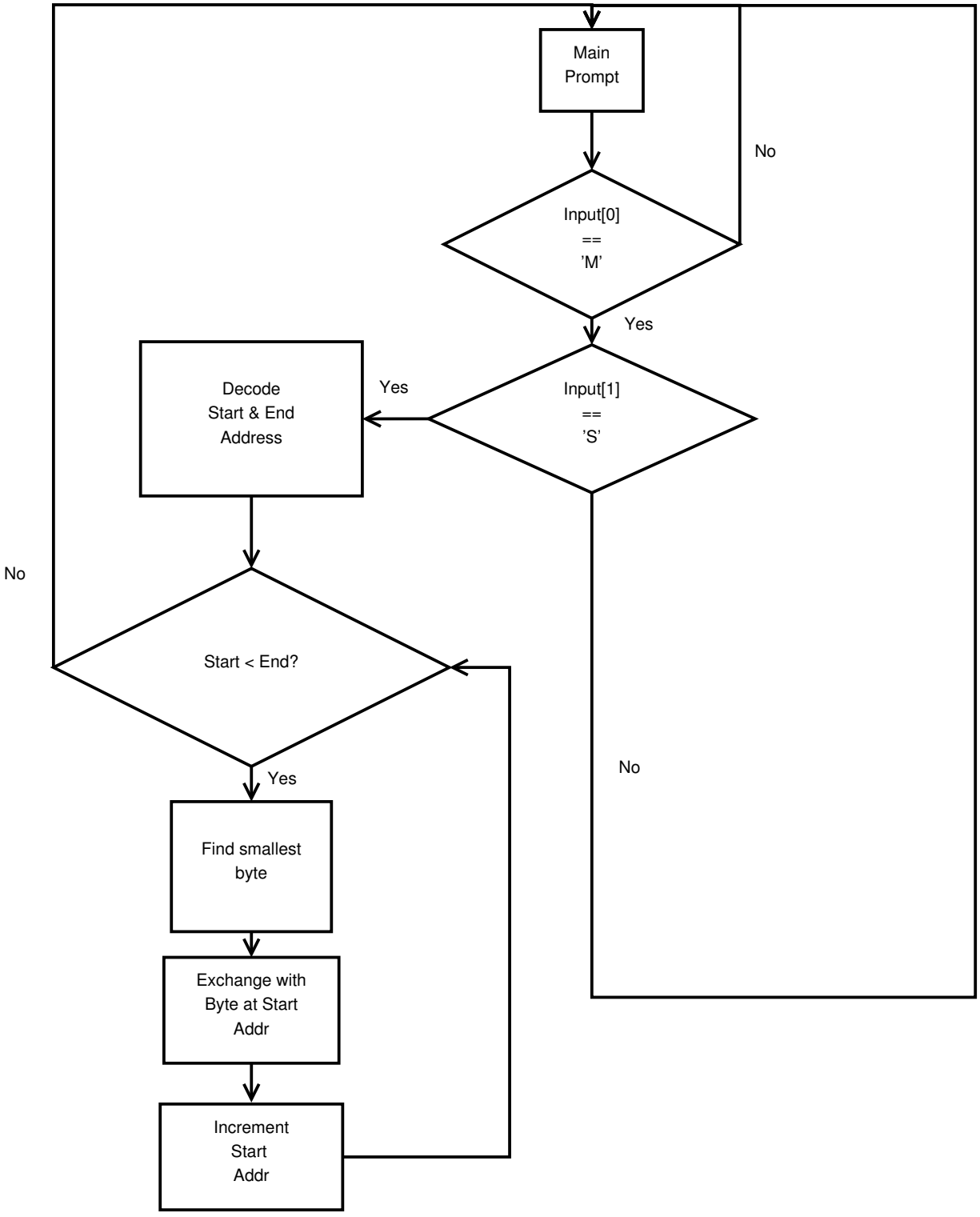
3.11 Memory Display (MD)



3.12 Memory Modify (MM)



3.13 Memory Sort (MS)



4

Code Listings

```
1       ORG     $1000
PROMPT      DC.B  'MONITOR441>',0
3 INPUT_BUFFER DS.B 80
BLOCK_TEST_FAILED DC.B 'BLOCK TEST FAILED AT: ',0
5 BLOCK_SEARCH_FOUND DC.B 'BLOCK SEARCH FOUND A MATCH STARTING AT
      : ',0
NEWLINE_STRING DC.B '  ',0
7 BS_TEST DC.B 'BS 0002 0010 AA AA',0
HELP1   DC.B 'COMMANDS: USAGE',0
9 HELP2   DC.B 'MEMORY DISPLAY: MD ADDR',0
HELP3   DC.B 'MEMORY SORT: MS LOW_ADDR HIGH_ADDR',0
11 HELP4   DC.B 'MEMORY MODIFY: MM ADDR',0
HELP5   DC.B 'BLOCK FILL: BF LOW_ADDR HIGH_ADDR FILL_WORD',0
13 HELP6   DC.B 'BLOCK SEARCH: BS LOW_ADDR HIGH_ADDR BYTE_1 BYTE_2
      ...',0
HELP7   DC.B 'BLOCK MOVE: BM LOW_ADDR HIGH_ADDR MOVE_ADDR',0
15 HELP8   DC.B 'BLOCK TEST: BT LOW_ADDR HIGH_ADDR',0
HELP9   DC.B 'HELP: H',0
17 HELP10 DC.B 'DISPLAY REGS: D',0
HELP11 DC.B 'MODIFY REG: .A[0-7] || .D[0-7]',0
19 HELP12 DC.B 'GO: G ADDR',0
HELP13 DC.B 'DATA CONVERSION: DC DATA',0
21 ASCII_FAIL DC.B 'THERE IS TO BE A SINGLE CHAR OF WHITESPACE
     BETWEEN COMMAND SEGMENTS, ADDR=WORD',0
RA0 DC.B 'A0 ',0
23 RA1 DC.B 'A1 ',0
RA2 DC.B 'A2 ',0
25 RA3 DC.B 'A3 ',0
RA4 DC.B 'A4 ',0
27 RA5 DC.B 'A5 ',0
RA6 DC.B 'A6 ',0
29 RA7 DC.B 'A7 ',0
RD0 DC.B 'D0 ',0
```

```
31 RD1 DC.B 'D1 ',0
   RD2 DC.B 'D2 ',0
33 RD3 DC.B 'D3 ',0
   RD4 DC.B 'D4 ',0
35 RD5 DC.B 'D5 ',0
   RD6 DC.B 'D6 ',0
37 RD7 DC.B 'D7 ',0
   RSR DC.B 'SR ',0
39 MS_TEST DC.B '987645321'
   MS_TEST_END DC.B '1'
41 BERR_TEXT  DC.B 'A BUS ERROR OCCURED: PRINTING REGS',0
   AERR_TEXT  DC.B 'AN ADDRESS ERROR OCCURED: PRINTING REGS',0
43 IERR_TEXT  DC.B 'AN ILLEGAL INSTRUCTION OCCURED: PRINTING REGS
       ',0
   PERR_TEXT  DC.B 'A PRIVILEGE VIOLATION OCCURED: PRINTING REGS
       ',0
45 ZERR_TEXT  DC.B 'A DIVIDE BY ZERO ERROR OCCURED: PRINTING REGS
       ',0
   AFERR_TEXT DC.B 'A A/F LINE ERROR OCCURED: PRINTING REGS',0
47
       ORG     $3000
49 ASCII
   ;this function converts a string (hex represenation) between
       addresses (A5) and (A6) into
51 ;binary
       MOVEM.L A0-A5/D1-D7,-(SP)
53     MOVE.L #1,D1
       CLR.L D0
55 ASCII_LOOP
       CLR.L D2
57     CMP.L A5,A6
       BLE ASCII_DONE
59     MOVE.B -(A6),D2
       CMP.B #'0',D2 ;LESS THAN 0?
61     BLT ASCII_OUT_OF_RANGE
       CMP.B #'F',D2 ;MORE THAN F?
63     BGT ASCII_OUT_OF_RANGE
       CMP.B #'9',D2
65     BLE ASCII_GOOD
       CMP.B #'A',D2
67     BGE ASCII_GOOD_HEX
       BRA ASCII_OUT_OF_RANGE
69 ASCII_GOOD_HEX
       SUB.B #$37,D2
71     BRA ASCII_SHIFTED
```

```
ASCII_GOOD
73      SUB #$30,D2
ASCII_SHIFTED
75      MULU D1,D2
        ADD.L D2,D0
77      MULU #$10,D1
        BRA ASCII_LOOP
79 ASCII_DONE       MOVEM.L (SP)+,A0-A5/D1-D7
        RTS
81 ASCII_OUT_OF_RANGE
        LEA ASCII_FAIL,A1
83      MOVE #13,D0
        TRAP #15
85      BRA ASCII_DONE

87 HEX
;this function outputs D3 characters of hex converted from D1
89      MOVEM.L A0-A6/D0-D7,-(SP)
        MOVE.L D1,D2
91 HEX_ROLL
        ROL.L #4,D2
93      MOVE.L D2,D1
        AND.L #$0000000F,D1
95      CMP.B #$A,D1
        BLT HEX_SKIP_ADD
97      ADD.B #$37,D1
        BRA HEX_WRITE_OUT
99 HEX_SKIP_ADD
        ADD.B #'0',D1
101 HEX_WRITE_OUT
        MOVE.L #6,D0
103     TRAP #15
        DBF D3,HEX_ROLL
105     MOVEM.L (SP)+,A0-A6/D0-D7
        RTS
107

SELECT_REG
109 ;this is the implementation for the .A* and .D* commands
;it works by selecting which reg to write into then writing the
111 ;ASCII coded hex (converted to binary) into that register.
        MOVEA A1,A5
113     ADDA #4,A5 ;DONT NEED THE FIRST 4 CHARS
        MOVEA A1,A6
115     ADDA  D1,A6 ;END OF STRING
        JSR ASCII ; DECODE ASCII(HEX)->BINARY
```

```
117         CMP.B #'A',(1,A1) ; 'A'
            BNE NEXT_REG_D
119         MOVE.B (2,A1),D2
            CMP #'0',D2
121         BNE REG_A1
            MOVEA D0,A0
123         BRA SELECT_REG_DONE
    REG_A1
125         CMP #'1',D2
            BNE REG_A2
127         MOVEA D0,A1
            BRA SELECT_REG_DONE
129 REG_A2
            CMP #'2',D2
131         BNE REG_A3
            MOVEA D0,A2
133         BRA SELECT_REG_DONE
    REG_A3
135         CMP #'3',D2
            BNE REG_A4
137         MOVEA D0,A3
            BRA SELECT_REG_DONE
139 REG_A4
            CMP #'4',D2
141         BNE REG_A5
            MOVEA D0,A4
143         BRA SELECT_REG_DONE
    REG_A5
145         CMP #'5',D2
            BNE REG_A6
147         MOVEA D0,A5
            BRA SELECT_REG_DONE
149 REG_A6
            CMP #'6',D2
151         BNE REG_A7
            MOVEA D0,A6
153         BRA SELECT_REG_DONE
    REG_A7
155         CMP #'7',D2
            BNE REG_A_FAIL
157         MOVEA D0,A7
            BRA SELECT_REG_DONE
159 REG_A_FAIL

161 SELECT_REG_DONE
```

```
        RTS
163
NEXT_REG_D
165     CMP.B #$44,(1,A1) ; 'D'
        BNE NEXT_REG_S
167     MOVE.B (2,A1),D2
        CMP #'0',D2
169     BNE REG_D1
        MOVE D0,D0
171     BRA SELECT_REG_DONE
REG_D1
173     CMP #'1',D2
        BNE REG_D2
175     MOVE D0,D1
        BRA SELECT_REG_DONE
177 REG_D2
        CMP #'2',D2
179     BNE REG_D3
        MOVE D0,D2
181     BRA SELECT_REG_DONE
REG_D3
183     CMP #'3',D2
        BNE REG_D4
185     MOVE D0,D3
        BRA SELECT_REG_DONE
187 REG_D4
        CMP #'4',D2
189     BNE REG_D5
        MOVE D0,D4
191     BRA SELECT_REG_DONE
REG_D5
193     CMP #'5',D2
        BNE REG_D6
195     MOVE D0,D5
        BRA SELECT_REG_DONE
197 REG_D6
        CMP #'6',D2
199     BNE REG_D7
        MOVE D0,D6
201     BRA SELECT_REG_DONE
REG_D7
203     CMP #'7',D2
        BNE REG_D_FAIL
205     MOVE D0,D7
        BRA SELECT_REG_DONE
```

```
207 REG_D_FAIL
    NEXT_REG_S
209
    BLOCK_DECODE
211 ;used to select which 'B' command was requested
        CMP.B #'F',(1,A1)
213     BEQ BLOCK_FILL
        CMP.B #'M',(1,A1)
215     BEQ BLOCK_MOVE
        CMP.B #'S',(1,A1)
217     BEQ BLOCK_SEARCH
        CMP.B #'T',(1,A1)
219     BEQ BLOCK_TEST
        RTS
221
    BLOCK_FILL
223 ;read start and end address, and fill word
        MOVEM.L A0-A6/D0-D7,-(SP)
225     MOVEA A1,A5
        ADDA #3,A5
227     MOVEA A5,A6
        ADDA #4,A6
229     BSR ASCII
        MOVEA D0,A2
231     ADDA #5,A5
        ADDA #9,A6
233     BSR ASCII
        MOVE D0,A3
235     ADDA #5,A5
        ADDA #9,A6
237     BSR ASCII
    ;while in the range, write the fill word, then increment the
        start address
239 BLOCK_FILL_LOOP
        CMP A3,A2
241     BGT BLOCK_FILL_DONE
        MOVE.W D0,(A2)+
243     BRA BLOCK_FILL_LOOP

245 BLOCK_FILL_DONE
        MOVEM.L (SP)+,A0-A6/D0-D7
247     RTS
    BLOCK_MOVE
249 ;read start, end, and target address
        MOVEM.L A0-A6/D0-D7,-(SP)
```

```
251     MOVEA A1,A5
        ADDA #3,A5
253     MOVEA A5,A6
        ADDA #4,A6
255     BSR ASCII
        MOVEA D0,A2
257     ADDA #5,A5
        ADDA #9,A6
259     BSR ASCII
        MOVE D0,A3
261     ADDA #5,A5
        ADDA #9,A6
263     BSR ASCII
        MOVEA D0,A5
265 ;while in the range, read from the start address, write to the
    target address, and increment both.
BLOCK_MOVE_LOOP
267     CMP A3,A2
        BGT BLOCK_MOVE_DONE
269     MOVE.W (A2)+,(A5)+
        BRA BLOCK_MOVE_LOOP
271
BLOCK_MOVE_DONE
273     MOVEM.L (SP)+,A0-A6/D0-D7
        RTS
275 BLOCK_SEARCH
        MOVEM.L A0-A6/D0-D7,-(SP)
277     MOVEA A1,A5 ;MOVE INPUT BUFFER ADDRESS TO A5
        ADDA #3,A5 ;SKIP PAST 'BS '
279     MOVEA A5,A6
        ADDA #4,A6 ;PUT END ADDRESS THE BYTE AFTER THE 4 CHARS FOR
    THE WORD'
281     BSR ASCII
        MOVEA D0,A2 ;LOW END OF SEARCH RANGE GOES IN A2
283     ADDA #5,A5 ;SKIP TO NEXT PART
        ADDA #9,A6
285     BSR ASCII
        MOVE D0,A3 ;HIGH END OF SEARCH RANGE TO A3
287     CLR.L D2 ; COUNTER OF BYTES
        ADDA #5,A5
289     ADDA #7,A6 ;ONLY 2 CHARS
        BRA BLOCK_SEARCH_SKIP
291 BLOCK_SEARCH_INPUT_LOOP
        ADDA #3,A5 ;ONLY 2 CHARS
293     ADDA #5,A6
```

```
BLOCK_SEARCH_SKIP
295     LEA INPUT_BUFFER,A4
        ADDA D1,A4
297     CMP.L A4,A6 ;CHECK IF YOU'VE HIT END OF INPUT
        BGT BLOCK_SEARCH_LOOP
299     BSR ASCII
        MOVE.B D0,(A1)+ ;WRITE OUT INTO BUFFER
301     ADD.L #1,D2 ;ADD TO THE COUNT
        BRA BLOCK_SEARCH_INPUT_LOOP
303 BLOCK_SEARCH_LOOP
        CMP.L A2,A3
305     BLT BLOCK_SEARCH_LOOP_DONE ;check bounds
        BSR BLOCK_SEARCH_FROM_HERE
307     CMP.L D2,D0                ;did the right number of bytes
    match?
        BNE BLOCK_SEARCH_NOT_FOUND
309     MOVEM.L A0-A6/D0-D7,-(SP)
        LEA BLOCK_SEARCH_FOUND,A1
311     MOVE.L A2,D1               ;this block prints a matching
    message
        MOVE.L #17,D0
313     TRAP #15
        LEA NEWLINE_STRING,A1
315     MOVE #13,D0
        TRAP #15
317     MOVEM.L (SP)+,A0-A6/D0-D7
BLOCK_SEARCH_NOT_FOUND         ;otherwise increment up A2, and
    loop
319     ADDA #1,A2
        BRA BLOCK_SEARCH_LOOP
321 BLOCK_SEARCH_LOOP_DONE
        MOVEM.L (SP)+,A0-A6/D0-D7
323     RTS

325 BLOCK_SEARCH_FROM_HERE
;this will search from the address passed in through A2
327 ;and return the number of bytes matching through D0.
        CLR.L D0
329     MOVEM.L A0-A6/D1-D7,-(SP)
        LEA INPUT_BUFFER,A1
331 BLOCK_SEARCH_FROM_HERE_LOOP
        CMP.B (A2)+,(A1)+
333     BNE BLOCK_SEARCH_LOOP_FAIL
        ADD.L #1,D0
335     CMP D2,D0
```

```
        BLT BLOCK_SEARCH_FROM_HERE_LOOP
337 BLOCK_SEARCH_LOOP_FAIL
        MOVEM.L (SP)+,A0-A6/D1-D7
339     RTS


341
    BLOCK_TEST
343 ;read start and end address
        MOVEM.L A0-A6/D0-D7,-(SP)
345     MOVEA A1,A5
        ADDA #3,A5
347     MOVEA A5,A6
        ADDA #4,A6
349     BSR ASCII
        MOVE.L D0,A2
351     ADDA #5,A5
        ADDA #9,A6
353     BSR ASCII
        MOVE.L D0,A3
355 BLOCK_TEST_LOOP
        CMP A3,A2
357     BGT BLOCK_TEST_DONE
        MOVE.W #$A0A0,(A2)    ;write to memory
359     CMP.W #$A0A0,(A2)+    ;then read from it to check
        BNE BLOCK_TEST_ERROR
361     BRA BLOCK_TEST_LOOP
    BLOCK_TEST_ERROR
363     MOVEM.L A0-A6/D0-D7,-(SP)
        LEA BLOCK_TEST_FAILED,A1
365     MOVE #14,D0
        TRAP #15
367     MOVE.L A2,D1
        SUB.L #2,D1
369     MOVE #3,D0
        TRAP #15
371     MOVEM.L (SP)+,A0-A6/D0-D7 ;when you hit a failure, exit
        test

373 BLOCK_TEST_DONE
        MOVEM.L (SP)+,A0-A6/D0-D7
375     RTS

377 GO
    ;read an address, then jmp to that address
379     MOVEA A1,A5
```

```
        ADDA #2,A5
381     MOVEA A5,A6
        ADDA #4,A6
383     BSR ASCII
        MOVEA D0,A0
385     JMP (A0)
        CMP.B D1,D3
387     CMP.B D2,D4

389 MEMORY_DECODE
    ;select which 'M' command was requested
391     CMP.B #'D',(1,A1)
        BEQ MD
393     CMP.B #'M',(1,A1)
        BEQ MM
395     CMP.B #'S',(1,A1)
        BEQ MS
397     RTS

399 MS
    ;read low and high address for sort
401     MOVEM.L A0-A6/D0-D7,-(SP)
        MOVEA A1,A5
403     ADDA #3,A5
        MOVEA A5,A6
405     ADDA #4,A6
        BSR ASCII
407     MOVEA D0,A2
        ADDA #5,A5
409     ADDA #9,A6
        BSR ASCII
411     MOVE D0,A3
    MS_LOOP
413     CMP.L A2,A3 ;check bounds
        BLE MS_DONE
415     BSR MS_SMALLEST ;find the smallest value address
        MOVE.B (A2),D1  ;swap it
417     MOVE.B D0,(A2)+
        MOVE.B D1,(A0)
419     BRA MS_LOOP
    MS_DONE
421     MOVEM.L (SP)+,A0-A6/D0-D7
        RTS
423
```

```
425 MS_SMALLEST
        MOVEM.L D1-D7/A1-A6,-(SP)
427     MOVE.B (A2),D0
        SUBA #1,A2
429 MS_SMALLEST_LOOP    ;iterate through the input, saving the
        smallest value and location
        ADDA #1,A2
431     CMP.L A2,A3
        BLE MS_SMALLEST_DONE
433     CMP.B (A2),D0
        BLT MS_SMALLEST_LOOP
435     MOVEA A2,A0
        MOVE.B (A2),D0
437     BRA MS_SMALLEST_LOOP
    MS_SMALLEST_DONE
439     MOVEM.L (SP)+,D1-D7/A1-A6 ;return the addr of the smallest
        in A0, its value in D0
        RTS
441


443
    MM
445 ;read address to start
        MOVEM.L A1-A6/D0-D7,-(SP)
447     MOVEA A1,A5
        ADDA #3,A5
449     MOVEA A5,A6
        ADDA #4,A6
451     BSR ASCII
        MOVEA D0,A2
453 MM_LOOP
    ;print addr
455     MOVE.L A2,D1
        MOVE.L #7,D3 ; FOR ADDR
457     BSR HEX
        LEA NEWLINE_STRING,A1 ; PUT A SPACE
459     MOVE.L #14,D0
        TRAP #15
461     MOVE.B (A2),D1 ; PULL DOWN CHAR
        SWAP D1 ;FIX IT FOR OUTPUT
463     LSL.L #8,D1
        MOVE.L #1,D3 ; ONLY 2 HEX DIGITS
465     BSR HEX
        TRAP #15 ;SPACE
467     MOVE #5, D0 ;INPUT CHAR
```

```
        TRAP #15
469     CMP.B #'.',D1 ;QUIT ON .
        BEQ MM_QUIT
471     LEA INPUT_BUFFER,A6 ;SAVING ON THAT SAME BUFFER
        MOVE.B D1,(A6)+
473     MOVE #5, D0
        TRAP #15
475     CMP.B #'.',D1
        BEQ MM_QUIT
477     MOVE.B D1,(A6)+
        LEA INPUT_BUFFER,A5
479     BSR ASCII
        MOVE.B D0,(A2)+ ;CYCLING UP TO NEXT BYTE
481     MOVE.L #13,D0
        TRAP #15
483     BRA MM_LOOP
MM_QUIT
485     MOVEM.L (SP)+,A1-A6/D0-D7
        RTS
487



489



491 MD
;read start address
493     MOVEM.L A0-A6/D0-D7,-(SP)
        MOVEA A1,A5
495     ADDA #3,A5
        MOVEA A5,A6
497     ADDA #4,A6
        BSR ASCII
499     MOVEA D0,A2
MD_BIGLOOP
501 ;print addr
        MOVE.L #$F, D5
503     MOVE.L A2,D1
        MOVE.L #7,D3
505     BSR HEX
        LEA NEWLINE_STRING,A1
507     MOVE.L #14,D0
        TRAP #15
509 MD_LOOP
;print 0x10 bytes
511     MOVE.B (A2)+,D1
        SWAP D1
```

```
513     LSL.L #8,D1
        MOVE.L #1,D3
515     BSR HEX
        TRAP #15
517     DBF D5,MD_LOOP
        MOVE #5, D0
519     TRAP #15
        MOVE.L #13,D0
521     TRAP #15
        CMP.B #'.',D1 ;check for exit
523     BNE MD_BIGLOOP
        MOVEM.L (SP)+,A0-A6/D0-D7
525     RTS

527 D_DECODE
    ;select 'D' command
529     CMP.B #0,(1,A1)
        BEQ DISPLAY_COMMAND
531     CMP.B #'C',(1,A1)
        BEQ DC
533     RTS
    DC
535 ;read HEX in, then print signed value in DEC.
        MOVEM.L D0-D7/A0-A6,-(SP)
537     MOVEA A1,A5
        ADDA #3,A5
539     MOVEA A1,A6
        ADDA D1,A6
541     BSR ASCII
        MOVE.L D0,D1
543     MOVE.L #3,D0
        TRAP #15
545     LEA NEWLINE_STRING,A1
        MOVE.L #13,D0
547     TRAP #15
        MOVEM.L (SP)+,D0-D7/A0-A6
549     RTS

551 DISPLAY_COMMAND
    ;iterate through all registers, saving stack, moving the
        appropriate
553 ;values to D0 and D1, printing identifiers and register values.
    ;and restoring the values from the stack
555     MOVEM.L D0/D1,-(SP) ;D0
        LEA RD0,A1
```

```
557     MOVE.L  D0,D1
        MOVE.L  #14,D0
559     TRAP    #15
        MOVE.L  #7,D3
561     BSR     HEX
        LEA     NEWLINE_STRING,A1
563     MOVE.L  #13,D0
        TRAP    #15
565     MOVEM.L (SP)+,D0/D1

567     MOVEM.L D0/D1,-(SP)  ;D1
        LEA     RD1,A1
569     MOVE.L  D1,D1
        MOVE.L  #14,D0
571     TRAP    #15
        MOVE.L  #7,D3
573     BSR     HEX
        LEA     NEWLINE_STRING,A1
575     MOVE.L  #13,D0
        TRAP    #15
577     MOVEM.L (SP)+,D0/D1

579     MOVEM.L D0/D1,-(SP)  ;D2
        LEA     RD2,A1
581     MOVE.L  D2,D1
        MOVE.L  #14,D0
583     TRAP    #15
        MOVE.L  #7,D3
585     BSR     HEX
        LEA     NEWLINE_STRING,A1
587     MOVE.L  #13,D0
        TRAP    #15
589     MOVEM.L (SP)+,D0/D1

591     MOVEM.L D0/D1,-(SP)  ;D3
        LEA     RD3,A1
593     MOVE.L  D3,D1
        MOVE.L  #14,D0
595     TRAP    #15
        MOVE.L  #7,D3
597     BSR     HEX
        LEA     NEWLINE_STRING,A1
599     MOVE.L  #13,D0
        TRAP    #15
601     MOVEM.L (SP)+,D0/D1
```

```
603     MOVEM.L D0/D1,-(SP) ;D4
        LEA RD4,A1
605     MOVE.L D4,D1
        MOVE.L #14,D0
607     TRAP #15
        MOVE.L #7,D3
609     BSR HEX
        LEA NEWLINE_STRING,A1
611     MOVE.L #13,D0
        TRAP #15
613     MOVEM.L (SP)+,D0/D1

615     MOVEM.L D0/D1,-(SP)
        LEA RD5,A1
617     MOVE.L D5,D1
        MOVE.L #14,D0
619     TRAP #15
        MOVE.L #7,D3
621     BSR HEX
        LEA NEWLINE_STRING,A1
623     MOVE.L #13,D0
        TRAP #15
625     MOVEM.L (SP)+,D0/D1

627     MOVEM.L D0/D1,-(SP) ;D4
        LEA RD6,A1
629     MOVE.L D6,D1
        MOVE.L #14,D0
631     TRAP #15
        MOVE.L #7,D3
633     BSR HEX
        LEA NEWLINE_STRING,A1
635     MOVE.L #13,D0
        TRAP #15
637     MOVEM.L (SP)+,D0/D1

639     MOVEM.L D0/D1,-(SP) ;D4
        LEA RD7,A1
641     MOVE.L D7,D1
        MOVE.L #14,D0
643     TRAP #15
        MOVE.L #7,D3
645     BSR HEX
        LEA NEWLINE_STRING,A1
```

```
647     MOVE.L  #13,D0
        TRAP    #15
649     MOVEM.L (SP)+,D0/D1

651     MOVEM.L D0/D1,-(SP)
        LEA     RA0,A1
653     MOVE.L  A0,D1
        MOVE.L  #14,D0
655     TRAP    #15
        MOVE.L  #7,D3
657     BSR     HEX
        LEA     NEWLINE_STRING,A1
659     MOVE.L  #13,D0
        TRAP    #15
661     MOVEM.L (SP)+,D0/D1

663     MOVEM.L D0/D1,-(SP)
        LEA     RA1,A1
665     MOVE.L  A1,D1
        MOVE.L  #14,D0
667     TRAP    #15
        MOVE.L  #7,D3
669     BSR     HEX
        LEA     NEWLINE_STRING,A1
671     MOVE.L  #13,D0
        TRAP    #15
673     MOVEM.L (SP)+,D0/D1

675     MOVEM.L D0/D1,-(SP)
        LEA     RA2,A1
677     MOVE.L  A2,D1
        MOVE.L  #14,D0
679     TRAP    #15
        MOVE.L  #7,D3
681     BSR     HEX
        LEA     NEWLINE_STRING,A1
683     MOVE.L  #13,D0
        TRAP    #15
685     MOVEM.L (SP)+,D0/D1

687     MOVEM.L D0/D1,-(SP)
        LEA     RA3,A1
689     MOVE.L  A3,D1
        MOVE.L  #14,D0
691     TRAP    #15
```

```
        MOVE.L  #7,D3
693     BSR  HEX
        LEA  NEWLINE_STRING,A1
695     MOVE.L  #13,D0
        TRAP  #15
697     MOVEM.L  (SP)+,D0/D1

699     MOVEM.L  D0/D1,-(SP)
        LEA  RA4,A1
701     MOVE.L  A4,D1
        MOVE.L  #14,D0
703     TRAP  #15
        MOVE.L  #7,D3
705     BSR  HEX
        LEA  NEWLINE_STRING,A1
707     MOVE.L  #13,D0
        TRAP  #15
709     MOVEM.L  (SP)+,D0/D1

711     MOVEM.L  D0/D1,-(SP)
        LEA  RA5,A1
713     MOVE.L  A5,D1
        MOVE.L  #14,D0
715     TRAP  #15
        MOVE.L  #7,D3
717     BSR  HEX
        LEA  NEWLINE_STRING,A1
719     MOVE.L  #13,D0
        TRAP  #15
721     MOVEM.L  (SP)+,D0/D1

723     MOVEM.L  D0/D1,-(SP)
        LEA  RA6,A1
725     MOVE.L  A6,D1
        MOVE.L  #14,D0
727     TRAP  #15
        MOVE.L  #7,D3
729     BSR  HEX
        LEA  NEWLINE_STRING,A1
731     MOVE.L  #13,D0
        TRAP  #15
733     MOVEM.L  (SP)+,D0/D1

735     MOVEM.L  D0/D1,-(SP)
        LEA  RA7,A1
```

```
737     MOVE.L  A7,D1
        MOVE.L  #14,D0
739     TRAP  #15
        MOVE.L  #7,D3
741     BSR  HEX
        LEA  NEWLINE_STRING,A1
743     MOVE.L  #13,D0
        TRAP  #15
745     MOVEM.L  (SP)+,D0/D1

747     MOVEM.L  D0/D1,-(SP)
        LEA  RSR,A1
749     MOVE  SR,D1
        MOVE.L  #14,D0
751     TRAP  #15
        MOVE.L  #7,D3
753     BSR  HEX
        LEA  NEWLINE_STRING,A1
755     MOVE.L  #13,D0
        TRAP  #15
757     MOVEM.L  (SP)+,D0/D1

759     RTS
HELP
761 ;print appropiate strings, return
        MOVEM.L  A1/D0,-(SP)
763     MOVE.L  #13,D0
        LEA  HELP1,A1
765     TRAP  #15
        LEA  HELP2,A1
767     TRAP  #15
        LEA  HELP3,A1
769     TRAP  #15
        LEA  HELP4,A1
771     TRAP  #15
        LEA  HELP5,A1
773     TRAP  #15
        LEA  HELP6,A1
775     TRAP  #15
        LEA  HELP7,A1
777     TRAP  #15
        LEA  HELP8,A1
779     TRAP  #15
        LEA  HELP9,A1
781     TRAP  #15
```

```
    LEA HELP10,A1
783 TRAP #15
    LEA HELP11,A1
785 TRAP #15
    LEA HELP12,A1
787 TRAP #15
    LEA HELP13,A1
789 TRAP #15
    MOVEM.L (SP)+,A1/D0
791 RTS

793 BERR
;print that an error occured, (possibly) print some information
    from the stack,
795 ;and print the registers
    MOVEM.L A1/D0,-(SP) ;save values to the stack
797 LEA BERR_TEXT,A1    ;print error message
    MOVE.L #13,D0
799 TRAP #15
    MOVE.W (12,SP),D1   ;print Status Word
801 SWAP D1
    MOVE.L #3,D3
803 BSR HEX
    LEA NEWLINE_STRING,A1
805 MOVE.L #14,D0
    TRAP #15
807 MOVE.L (14,SP),D1   ;print Addr
    MOVE.L #7,D3
809 BSR HEX
    MOVE.L #14,D0
811 TRAP #15
    MOVE.W (18,SP),D1   ;print IR
813 SWAP D1
    MOVE.L #3,D3
815 BSR HEX
    MOVE.L #13,D0
817 TRAP #15
    MOVEM.L (SP)+,A1/D0 ;restore values before printing
819 BSR DISPLAY_COMMAND ;print registers
    MOVE.L #$01000000,SP ;reset the stack
821 BRA MAIN
    AERR
823 ;This and the rest of the error handlers are the same as BERR
    MOVEM.L A1/D0/D1,-(SP)
825 LEA AERR_TEXT,A1
```

```
        MOVE.L  #13,D0
827     TRAP    #15
        MOVE.W  (12,SP),D1
829     SWAP    D1
        MOVE.L  #3,D3
831     BSR     HEX
        LEA     NEWLINE_STRING,A1
833     MOVE.L  #14,D0
        TRAP    #15
835     MOVE.L  (14,SP),D1
        MOVE.L  #7,D3
837     BSR     HEX
        MOVE.L  #14,D0
839     TRAP    #15
        MOVE.W  (18,SP),D1
841     SWAP    D1
        MOVE.L  #3,D3
843     BSR     HEX
        MOVE.L  #13,D0
845     TRAP    #15
        MOVEM.L (SP)+,A1/D0/D1
847     BSR     DISPLAY_COMMAND
        MOVE.L  #$01000000,SP
849     BRA     MAIN
    IERR
851     MOVEM.L A1/D0,-(SP)
        LEA     IERR_TEXT,A1
853     MOVE.L  #13,D0
        TRAP    #15
855     MOVEM.L (SP)+,A1/D0
        BSR     DISPLAY_COMMAND
857     MOVE.L  #$01000000,SP
        BRA     MAIN
859 PERR
        MOVEM.L A1/D0,-(SP)
861     LEA     PERR_TEXT,A1
        MOVE.L  #13,D0
863     TRAP    #15
        MOVEM.L (SP)+,A1/D0
865     BSR     DISPLAY_COMMAND
        MOVE.L  #$01000000,SP
867     BRA     MAIN
    ZERR
869     MOVEM.L A1/D0,-(SP)
        LEA     ZERR_TEXT,A1
```

```
871     MOVE.L  #13,D0
        TRAP    #15
873     MOVEM.L (SP)+,A1/D0
        BSR     DISPLAY_COMMAND
875     MOVE.L  #$01000000,SP
        BRA     MAIN
877 AFERR
        MOVEM.L A1/D0,-(SP)
879     LEA     AFERR_TEXT,A1
        MOVE.L  #13,D0
881     TRAP    #15
        MOVEM.L (SP)+,A1/D0
883     BSR     DISPLAY_COMMAND
        MOVE.L  #$01000000,SP
885     BRA     MAIN


887
    START:                  ; FIRST INSTRUCTION OF PROGRAM
889     MOVEM.L A1,-(SP)
        LEA     BERR,A1 ;init exception handlers
891     MOVE.L  A1,$8
        LEA     AERR,A1
893     MOVE.L  A1,$C
        LEA     IERR,A1
895     MOVE.L  A1,$10
        LEA     ZERR,A1
897     MOVE.L  A1,$14
        LEA     PERR,A1
899     MOVE.L  A1,$20
        LEA     AFERR,A1
901     MOVE.L  A1,$28
        MOVE.L  A1,$2C
903     MOVEM.L (SP)+,A1
    MAIN
905     LEA     NEWLINE_STRING,A1 ;print prompt, read string in, switch
         on first character.
        MOVE    #13,D0
907     TRAP    #15
        LEA     PROMPT,A1
909     MOVE    #14,D0
        TRAP    #15
911     LEA     INPUT_BUFFER,A1
        MOVE.L  #2,D0
913     TRAP    #15
        CMP.B   #'.',(A1)
```

```
915         BNE B_COMMAND
            JSR SELECT_REG
917         BRA MAIN
    B_COMMAND
919         CMP.B #$42,(A1)
            BNE G_COMMAND
921         BSR BLOCK_DECODE
            BRA MAIN
923 G_COMMAND
            CMP.B #'G',(A1)
925         BNE M_COMMAND
            BSR GO
927         BRA MAIN
    M_COMMAND
929         CMP.B #'M',(A1)
            BNE D_COMMAND
931         BSR MEMORY_DECODE
            BRA MAIN
933 D_COMMAND
            CMP.B #'D',(A1)
935         BNE H_COMMAND
            BSR D_DECODE
937         BRA MAIN
    H_COMMAND
939         CMP.B #'H',(A1)
            BNE MAIN
941         BSR HELP
            BRA MAIN
943 * PUT PROGRAM CODE HERE

945         SIMHALT             ; HALT SIMULATOR

947         END     START       ; LAST LINE OF SOURCE
```

chinetti_proj.X68

5

Manual

5.1 Command Interpreter

Spaces matter in commands. The correct format uses one space to delimit groups of characters.
The following sections of the manual outline how to use the commands, in the format:
COMMANDS: USAGE

5.2 Modify Registers (.A*, .D*)

MODIFY REG: .A[0-7] or .D[0-7]

5.3 Block Fill (BF)

BLOCK FILL: BF LOW_ADDR HIGH_ADDR FILL_WORD

5.4 Block Move (BM)

BLOCK MOVE: BM LOW_ADDR HIGH_ADDR MOVE_ADDR

5.5 Block Search (BS)

BLOCK SEARCH: BS LOW_ADDR HIGH_ADDR BYTE_1 BYTE_2 ...

5.6 Block Test (BT)

BLOCK TEST: BT LOW_ADDR HIGH_ADDR

5.7 Data Conversion (DC)

DATA CONVERSION: DC DATA

5.8 Display Formatted Registers (DF)

DISPLAY REGS: DF

5.9 Run Program [GO] (G)

GO: G ADDR

5.10 Help (H)

HELP: H

5.11 Memory Display (MD)

MEMORY DISPLAY: MD ADDR

5.12 Memory Modify (MM)

MEMORY MODIFY: MM ADDR

5.13 Memory Sort (MS)

MEMORY SORT: MS LOW_ADDR HIGH_ADDR

6

Engineering and Design Challenges

This was a fairly simply project from a design and engineering perspective. Most of the effort expended on the project was debugging the assembly code to verify functionality. However, there were a few wrinkles.

The first was deciding which commands to implement. Commands were selected to be the simplest set that provided relatively complete functionality. Then, when implementing the commands, they had to be pared down to the easiest to implement case that still allowed for full functionality. For example, there is only functionality to modify memory a byte at a time, not a word or a longword. This is because with more entries, you can use byte size modifications to simulate word or longword modifications. Another example is block search. In that command, bytes must be specified as two character sets, not directly as ASCII characters. Strings can still be searched for, but they must be manually translated to the hex values of each character.

7

Conclusion

The Monitor Program was written, and has the functionality requested. Further work could be done to implement the full functionality of the TUTOR program, but if that fuctionality was required, it would be better to begin again in a higher level language.

Bibliography

[1] *MC6800 Educational Computer Board*. Motorola, Arizona, Second Edition, 1982.