SORCERY BREWS

SORCERY BREWS

CONCOCTED BY HOWARD ARRINGTON

A PRODUCT OF THE GLOBAL SOFTWARE NETWORK

This manual is a treasury of programming tricks that are specific to the Exidy Sorcerer computer, although much is applicable to other microcomputers that employ either Microsoft Basic or a Z80 microprocessor. With this ready reference of valuable examples at your fingertips, your programming efforts will be greatly simplified, and your programs will be more professional in both appearance and performance. Using this manual will unleash the hidden powers of your Sorcerer. You will graduate from being an apprentice to being a full wizard as you study and use the brews concocted by masters of the Sorcerer.

TABLE OF CONTENTS

CHAPTER 1 --- BASIC PROGRAM TIPS

CHAPTER 2 --- BASIC COMMANDS

CHAPTER 3 --- FUNCTIONS

CHAPTER 4 --- KEYBOARD

CHAPTER 5 --- VIDEO

CHAPTER 6 --- JOYSTICKS

CHAPTER 7 --- SOUND

CHAPTER 8 --- BASIC ROMPAC

CHAPTER 9 --- MONITOR

CHAPTER 10 --- MACHINE LANGUAGE INTERFACE

CHAPTER 11 --- MACHINE CODE ROUTINES

CHAPTER 12 --- I/O DRIVERS

CHAPTER 13 --- CASSETTE TAPE

CHAPTER 14 --- EDITOR FOR BASIC

CHAPTER 15 --- CP/M

CHAPTER 16 --- WORD PROCESSOR

CHAPTER 17 --- DEVELOPMENT PAC

CHAPTER 18 --- PLOTTING

CHAPTER 19 --- TABLES AND FORMS

FORWARD

"In promulgating your esoteric cognitations, or articulating your superficial sentimentalities and amicable, philosophical or psychological observations, beware of platitudinous ponderosity. Let your conversational communications possess a clarified conciseness, a compact comprehensibleness, coalescent consistency, and a concantenated cogency. Eschew all conglomerations of flatulent garrulity, jejune babblement and asinine affectation. Let your extemporaneous and unpremeditated expatiations have intelligibility and vivacious vivacity, without rodomontade and thrasonical bombast. Sedulously avoid all polysyllabic profundity, pompous prolixity, osittaceous vacuity, ventriloquial verbosity, and veniloquent vapidity. Shun double-entendres, pruvient jocosity, and pestiferous profanity, obscurant or apparent. In other words, write plainly, briefly, naturally, sensibly, truthfully, purely. Keep from complexities; don't put on airs; write what you mean; and don't forget that others are trying to understand you. Simply stated -- Reread this paragraph until you understand the veniloquent vapidity."

I've tried to heed this good counsel throughout this manual.

To my patient wife

MARILYN

CHAPTER 1 --- PROGRAM TIPS

1.01 RELOCATE PROGRAMS TO A DIFFERENT RUN ADDRESS.

```
BYE                     EXIT TO THE MONITOR.
>EN 149                 CHANGE POINTER IN 149 HEX.
0149: 00 10 /           NEW BASIC START ADDRESS OF 1000 HEX.
>EN 0FFF
0FFF: 0 /               ZERO BYTE BEFORE START ADDRESS.
>PP                     GO ENTER BASIC PROGRAM FROM KEYBOARD.
```

PROGRAM CAN BE RUN AND CSAVED IN NORMAL FASHION.
TO CLOAD, THOUGH, 0149 HEX MUST BE ALTERED AS ABOVE.
THIS ALLOWS A BASIC PROGRAM TO HAVE A PROTECTED MEMORY SPACE FOR MACHINE LANGUAGE ROUTINES FROM 01D5H TO ONE BYTE BEFORE THE START OF YOUR RELOCATED BASIC PROGRAM.

==========

1.02 END-OF-PROGRAM ADDRESS FOR ROMPAC BASIC.

```
BYE                     EXIT TO MONITOR
>DU 1B7 1B8             ADDRESS STORED IN THESE TWO BYTES.

ADDR  0  1  2  3   4  5  6  7   8
01B0: .. .. .. ..  .. .. .. yy  xx
```

HEX ADDRESS IS STORED IN REVERSE BYTE ORDER IN ADDRESS 01B7 AND 01B8 HEX. ADDRESS IS READ AS xxyy.

==========

1.03 CSAVE BASIC AND MACHINE LANGUAGE ROUTINES.

PLACE MACHINE LANGUAGE ROUTINES IN MEMORY JUST BEYOND THE END OF A BASIC PROGRAM.

PUT THE ADDRESS TO SAVE THROUGH IN BYTES 1B7 AND 1B8.

EXAMPLE: IF MACHINE CODE OCCUPIES TO ADDRESS 264F HEX.

```
BYE
>EN 1B7
01B7: 4F 26 /           ENTER NEW ENDING ADDRESS.
>PP
CSAVE XMPLE             SAVE BASIC AND ROUTINES.
```

WHEN THE BASIC PROGRAM AND THE MACHINE CODE RELOAD, THE MACHINE CODE NEEDS TO BE MOVED BACK TO ITS ORIGINAL ADDRESS IN MEMORY. BE CAREFUL THAT VARIABLE STORAGE DOES NOT OVERWRITE IT WHEN THE BASIC PROGRAM RUNS.

1.04 SAVE A CHARACTER SET AND A BASIC PROGRAM TOGETHER.

```
BYE
>DU 1B7 1B8                FIND END OF PROGRAM ADDRESS.
>SA XMPLE FC00 xxyy        SAVE THROUGH xxyy FROM 1B7 1B8.

>LO XMPLE                  TO RERUN YOUR PROGRAM.
>PP
RUN
```

==========

1.05 AUTO-EXECUTE BASIC PROGRAMS

PROGRAM MUST HAVE A LINE 0. EXAMPLE: 0 REM
FIND PROGRAM END IN 1B7-1B8 HEX FOR xxyy ADDRESS.

```
>SE X=C858                 (AUTO EXECUTE ADDRESS)
>SE F=4D                   (FILE TYPE)
>SA PROG 1B7 xxyy
```

LOAD AND AUTO EXECUTE WITH >LOG

==========

1.06 FIND TOP-OF-MEMORY IN ADDRESSES F000-F001 HEX.

```
100 RAMSIZE = PEEK(-4095) * 256 + PEEK(-4096)
```

==========

1.07 ARRAY SPACE REQUIREMENT

THE NUMBER OF BYTES REQUIRED TO STORE AN ARRAY IS:

(# OF ELEMENTS) * 4 + (# OF DIMENSIONS) * 2 + 6

```
EXAMPLE:  100 DIM A(10,10) : REM USES 494 BYTES
                           : REM 494 = 11*11*4+2*2+6
```

==========

1.08 CONCEAL 'REM' AND LINE NUMBER IN SCREEN LISTINGS.

ONE CAN CONCEAL THE LINE NUMBER OF REMARK STATEMENTS BY USING THE FOLLOWING TRICK. THE APPEARANCE OF REMARK STATEMENTS IMPROVES BECAUSE THEY READILY STAND OUT.

SAMPLE OUTPUT:

```
100 PRINT "THIS IS A SAMPLE SCREEN LISTING"

       --- A REMARK, BUT NO 'REM' OR LINE # SHOWS ---

120 PRINT "LINE NUMBER 110 IS PRESENT, YET HIDDEN"
```

TO DO THE ABOVE TO A REMARK STATEMENT, DO THE FOLLOWING:

1. TYPE THE LINE NUMBER AND 'REM', IE. 110 REM
2. NOW, PRESS THE CURSOR LEFT KEY SEVEN OR MORE TIMES TO MOVE THE CURSOR ALL THE WAY BACK TO THE LEFT MARGIN.
3. USE THE SPACE BAR TO ERASE THE VISIBLE LINE NUMBER AND THE 'REM' LETTERS.
4. NOW, PRESS 'LINE FEED' TO INSERT A BLANK LINE.
5. TYPE THE REMARK STATEMENT'S TEXT.
6. TYPE ANOTHER 'LINE FEED' TO INSERT A BLANK LINE BELOW.
7. FINALLY, TERMINATE THE LINE WITH CARRIAGE RETURN.

==========

1.09 MACHINE CODE ROUTINES HIDDEN IN BASIC REMARKS

ONE CAN STORE A SHORT MACHINE LANGUAGE ROUTINE AS PART OF BASIC STATEMENT BY PUTTING IT IN A REM STATEMENT. IT WILL AUTOMATICALLY BE SAVED AND LOADED WITH THE BASIC PROGRAM.

EXAMPLE:

```
100 REM < A LINE FULL OF SPACES >
110 REM THE REST OF THE PROGRAM FOLLOWS
```

IN MEMORY THE FIRST LINE LOOKS LIKE:

```
01D5: yy xx 64 00 8F
01DA: 20 20 20 20 20 20 20 20 20 20 20 20  etc.
```

THE xxyy IS THE LINK POINTER ADDRESS OF WHERE THE NEXT LINE STARTS. THE LINE NUMBER OF 100 IS STORED AS 64 00. THE 'REM' COMMAND IS STORED AS 8F. STARTING IN ADDRESS 01DA ARE ALL THE SPACES OF THE REM TEXT.

YOU CAN INSERT A SHORT MACHINE LANGUAGE ROUTINE IN THE ADDRESS SPACE BETWEEN 01DAH AND xxyy, FOR EXAMPLE. ONCE THE CODE IS INSERTED, DON'T ALTER THE REM STATEMENT OR ANY LINE WHICH MIGHT PRECEED THE REM STATEMENT.

ALSO, THE CODE INSERTED CANNOT CONTAIN A 00 BYTE AS BASIC WILL TREAT IT AS AN END-OF-LINE MARKER.

1.10 THREE MEMORY SIZE DEPENDENT BYTES IN THE BWA

0145-6H - POINTER TO TOP OF BASIC STACK.
0192-3H - POINTER TO HIGHEST RAM LOCATION.
01A6-7H - POINTER TO TOP OF FREE STRING SPACE.

IF THE BASIC WORK AREA FROM 0100H THROUGH 01D4H IS SAVED WITH A BASIC PROGRAM IN THIS FASHION: >SA PROG 100 xxyy, THE PROGRAM WILL NOT RUN ON A MACHINE WHICH HAS LESS MEMORY THAN THE MACHINE ON WHICH THE PROGRAM WAS RECORDED.

THE CONFLICT CAN BE OVERCOME BE LOADING BYTES 0146H, 0193H, AND 01A7H WITH THE CORRECT VALUE, IE. THE DEFAULT VALUE LOADED BY BASIC ON POWER UP. THE VALUE IS 03EH FOR A 16K MACHINE, 07EH FOR A 32K MACHINE, AND 0BEH FOR A 48K MACHINE.

==========

1.11 AN IDEAL PROGRAM START

```
  0 REM ---   PROGRAM NAME   ---
100 :
110 REM AUTHOR'S COPYRIGHT NOTICE
120 REM PROGRAM DATE AND VERSION NUMBER
130 :
140 REM DEFINE CURSOR CONTROL VARIABLES
150 CL$=CHR$(1) : CP$=CHR$(12) : CH$=CHR$(17)
160 CR$=CHR$(19): CU$=CHR$(23) : CD$=CHR$(26)
170 :
180 REM COMPUTE MEMORY SIZE TO LOCATE MONITOR WORK AREA
190 MS=256*PEEK(-4095)+PEEK(-4096)
200 IF MS>32767 THEN MS=MS-65536
210 :
220 REM LOCATE MONITOR WORK AREA CONTROL BYTES
230 PC=MS-47 :REM  PC is printer control poke address.
240 SS=MS-48 :REM  SS is screen speed poke address.
250 BR=MS-49 :REM  BR is baud rate poke address.
```

==========

1.12 FASTER BASIC EXECUTION

BASIC STARTS AT THE BEGINNING OF A PROGRAM AND SCANS THROUGH LOOKING FOR THE LINE NUMBER REFERENCED BY A GOSUB, IF-THEN, OR GOTO.

OPTIMIZE BY HAVING OFTEN CALLED LINE NUMBERS NEAR THE BEGINNING OF A PROGRAM. HAVE SELDOM USED LINES, SUCH AS PROGRAM INSTRUCTIONS AND INITIALIZATION, AT THE BOTTOM OF A PROGRAM.

1.13 MERGE TWO BASIC PROGRAMS TOGETHER.

1. THE LINE NUMBERS IN THE PROGRAM TO BE MERGED MUST BE GREATER THAN THE LAST LINE NUMBER IN THE FIRST PROGRAM. USE A LINE RENUMBERING PROGRAM IF NECESSARY TO ACCOMPLISH THIS.

2. LOAD THE FIRST PROGRAM INTO MEMORY. FIND ITS END LOACATION IN ADRESS 01B7 AND 01B8 HEX.

3. USE THE END ADDRESS TO FIND THE THREE ZEROS IN MEMORY MARKING THE END OF THE PROGRAM. WRITE DOWN THE ADDRESS OF THE SECOND ZERO.

4. USE THE MONITOR TO LOAD THE PROGRAM TO BE MERGED. USE >LO FILENAME 1 xxyy WHERE xxyy IS THE ADDRESS OF THE SECOND ZERO FROM STEP #3.

5. ADD THE MERGING PROGRAM'S SIZE TO xxyy TO HELP LOCATE THE THREE ZEROS AT THE END OF THE TWO COMBINED PROGRAMS.

6. STORE THE ADDRESS OF THE THIRD ZERO IN BYTES 01B7 AND 01B8 HEX. THE ADDRESS IS STORED IN REVERSE BYTE ORDER.

7. IN BASIC, ADD AND DELETE A LINE NUMBER 0, IE. 0 REM THIS WILL RECOMPUTE ALL OF THE LINK POINTERS. THE TWO PROGRAMS ARE NOW MERGED AND CAN BE SAVED ON TAPE.

CHAPTER 2 --- BASIC COMMANDS

2.01 CLEAR COMMAND

```
100 CLEAR xx,yy

xx IS THE NUMBER OF BYTES TO RESERVE FOR STRING SPACE.
yy IS THE TOP-OF-MEMORY ADDRESS FOR THIS STRING SPACE.

EXAMPLE:  100 CLEAR 500,24576

500 BYTES ARE RESERVED FOR STRING MANIPULATIONS.
THE STRING WORKSPACE WILL START AT 6000 HEX AND GROW
DOWNWARD IN MEMORY TOWARD THE VARIABLE STORAGE SPACE.

MEMORY FROM 6000 HEX TO THE BOTTOM OF THE MONITOR STACK
WORKSPACE IS NOW 'PROTECTED' MEMORY AVAILABLE FOR
MACHINE LANGUAGE ROUTINES.
```

==========

2.02 RESTORE nnnn WILL RESTORE TO A SPECIFIC LINE #.

==========

2.03 SWAP INTEGER VARIABLES

```
100 A=A XOR B B: B=B XOR A : A=A XOR B
```

==========

2.04 LEN(STR$(1))=2 SINCE A SPACE PRECEDES THE DIGIT.

```
USE A$ = MID$(STR$(J),2) TO EXTRACT JUST THE DIGITS.
```

==========

2.05 MID$ FUNCTION EXAMPLES.

```
100 A$="1234567890"
110 PRINT MID$(A$,3,2)        : REM OUTPUT "34"
120 PRINT MID$(A$,3,4)        : REM OUTPUT "3456"
130 PRINT MID$(A$,3)          : REM OUTPUT "34567890"
140 PRINT MID$(A$,8,5)        : REM OUTPUT "890"

MID$(A$,1,J) = LEFT$(A$,J)
```

2.06 USE CHR$(64) TO PRINT THE '@' SIGN.

```
EXAMPLE:  100 PRINT "5 APPLES ";CHR$(64);" 15 CENTS EACH"
```

==========

2.07 ORDER OF PRECEDENCE FOR NUMERICAL OPERATIONS

1. () EXPRESSIONS IN PARENTHESES.
2. ^ EXPONENTIATION.
3. - NEGATION
4. * / MULTIPLICATION AND DIVISION
5. + - ADDITION AND SUBTRATION
6. NUMERICAL RELATIONS
   - = EQUAL
   - <> NOT EQUAL
   - < LESS THAN
   - > GREATER THAN
   - <= LESS THAN OR EQUAL
   - >= GREATER THAN OR EQUAL
7. NOT
8. AND
9. OR

==========

2.08 FLOATING POINT INACCURACIES

```
100 J= 30*30 : X= 30^2 : PRINT J,X
```

BOTH SHOULD PRINT 900, HOWEVER THE SECOND NUMBER IS PRINTED AS 900.001 BECAUSE ALL OPERATIONS EMPLOY ONLY SIX DIGITS OF ACCURACY. THUS, ONE MAY HAVE TO TEST FOR A BOUNDED CONDITION RATHER THAN FOR EQUALITY. FOR EXAMPLE:

```
110 IF X>899.997 AND X<900.003 THEN PRINT " X = 900 "
```

==========

2.09 NUMERICAL RELATIONS

IF A RELATION IS 'TRUE', A VALUE OF -1 IS RETURNED.
IF A RELATION IS 'FALSE', A VALUE OF 0 IS RETURNED.

```
EXAMPLE:  100 PRINT 1=2, 1<2, 2=2, "A"="A", "AB"="AC"

OUTPUT:              0    -1   -1     -1      0
```

2.10 LOGICAL OPERATOR TRUTH TABLES

```
AND        I        J      I AND J
          ------------------------
           0        0         0
           0        1         0
           1        0         0
           1        1         1

OR         I        J       I OR J
          ------------------------
           0        0         0
           0        1         1
           1        0         1
           1        1         1
```

EXAMPLE: J AND 255 = JUST THE LOWER BYTE
THIS IS A QUICK WAY TO GET THE
LOWER BYTE OF AN ADDRESS TO POKE.

```
15 AND 13 = 13   IN BINARY  1111 AND 1101 = 1101

-1 AND J  = J    BECAUSE -1 IS ALL ONES.

5  OR  2  = 7    IN BINARY  0101 OR 0010 = 0111

5  AND 2  = 0

NOT 0 = -1       FALSE = 0    TRUE = -1

NOT J = -(J+1)   TWO'S COMPLEMENT.
```

==========

2.11 ONE SECOND DELAY LOOP IN BASIC

```
100 FOR J=1 TO 500:NEXT J
```

==========

2.12 A 0.81 SECOND DELAY LOOP IN THE MONITOR

```
100 POKE 260,39 : POKE 261,224
110 X=USR(0)    : REM 0.81 SEC DELAY WITH EACH CALL
```

THIS 'CASSETTE OFF' ROUTINE GIVES 0.81 SECONDS OF DELAY.

==========

2.13 A 3.24 SECOND DELAY LOOP IN THE MONITOR

```
100 POKE 260,160 : POKE 261,226
110 X=USR(0)     : REM 3.24 SEC DELAY WITH EACH CALL
```

2.14 MULTIPLE LINE CHARACTER STRINGS.

GRAPHICAL FIGURES WHICH OCCUPY THREE OR FOUR PRINT LINES AND ARE A FEW CHARACTERS WIDE MAY BE CREATED IN A SINGLE STRING AND PRINTED WITH ONE PRINT STATEMENT. A SINGLE PRINT STATEMENT USED TO PLACE THE FIGURE ON THE SCREEN IS MUCH FASTER THAN PRINTING THE FIGURE WITH MULTIPLE PRINT STATEMENTS.

EXAMPLE:

```
100 A$ = "*****
          *   *
          *   *
          *****"
110 PRINT A$
```

THE ABOVE PROGRAM WILL PRINT A BOX MADE FROM ASTERISKS. THERE ARE HIDDEN CHARACTERS IN LINE 100 THAT MAKE THE STRING A$ PRINT THE BOX ON FOUR LINES.

A STRING BEGINS AND ENDS WITH QUOTE MARKS, AND MAY CONTAIN ANY CHARACTERS. OUR STRING CONTAINS THE ASTERISKS THAT ARE SEEN, PLUS 'LINE FEED', 'CURSOR LEFT' AND 'SPACE' CHARACTERS.

THE STRING CONSISTS OF THE FOLLOWING SEQUENCE OF CHARACTERS IN THE ORDER PRESSED FROM THE KEYBOARD:

```
A$ = "*****<LF><CL><CL><CL><CL><CL>
      *   *<LF><CL><CL><CL><CL><CL>
      *   *<LF><CL><CL><CL><CL><CL>
      *****"
```

THE STRING A$ IS 38 CHARACTERS IN LENGTH.
<LF> IS A LINE FEED CHARACTER.
<CL> IS A CURSOR LEFT CHARACTER.

==========

2.15 PRINT DECIMAL NUMBER IN HEXADECIMAL.

```
0000: ED 5B 07 01 ENTRY LD   DE,(0107H)   ;GET I # IN E
0004: 57                LD   D,A          ;GET J # IN D
0005: CD E8 E1          CALL 0E1E8H       ;PRINT HEX #
0008: C1                POP  BC           ;RELIEVE STACK
0009: C9                RET
```

ROUTINE MUST RESIDE AT ADDRESS 0 FOR RST 0H COMMAND.

```
100 POKE 262,199 : REM 'OUT I,J' BECOMES RST 0H COMMAND.
110 INPUT X      : REM X IS DECIMAL # TO PRINT IN HEX.
120 I=X : IF X>32767 THEN I=X-65536
130 OUT I AND 255, X/256 : REM HEX # IS PRINTED.
```

2.16 PRINT DECIMAL NUMBER IN HEXADECIMAL

```
0000: CD D0 C7    ENTRY CALL 0C7D0H       ;GET USR() #
0003: C3 E8 E1          JP   0E1E8H       ;PRINT HEX #

100 POKE 260,0 : POKE 261,0 : REM USR() ENTRY ADDRESS
110 X = USR(J)              : REM PRINT HEX VALUE OF J
```

==========

2.17 PRINT DECIMAL NUMBER IN HEXADECIMAL.

```
0000  11 yy xx    ENTRY LD   DE,nn        nn IN DECIMAL
0003  C3 E8 E1          JP   0E1E8H       PRINT HEX #

TO USE THE ABOVE MACHINE LANGUAGE ROUTINE, FIRST PLACE
DECIMAL NUMBER TO CONVERT IN BYTES 1 AND 2.

EXAMPLE:  100 POKE 260,0 : POKE 261,0 : REM USR() ADDR
          110 POKE 1,J AND 255 : POKE 2,J/256
          120 X=USR(0) : REM PRINT HEX EQUIVALENT OF J
```

==========

2.18 CONVERT HEXADECIMAL TO DECIMAL

```
100 DEF FNA(X) = (X AND 15) - 9 * (X > 64)
110 :
120 A$ = "FC07" : REM TYPICAL HEXADECIMAL # TO CONVERT
130 :
140 N=0 : FOR J=1 TO LEN(A$)
150 N=N*16 + FNA(ASC(MID$(A$,J))) : NEXT J
160 :
170 PRINT A$;" HEXADECIMAL EQUALS ";N;" DECIMAL"
```

==========

2.19 CONVERT DECIMAL TO HEXADECIMAL

```
100 N = 4000  : REM TYPICAL NUMBER TO CONVERT TO HEX
110 :
120 J = N : A$="" :REM  A$ TO RECEIVE HEXADECIMAL DIGITS
130 I = J AND 15 : J = INT(J/16)
140 A$= CHR$(I+48 - 7*(I > 9)) + A$ : IF J>0 THEN 130
150 :
160 PRINT N;" DECIMAL EQUALS ";A$;" HEXADECIMAL"
```

2.20 MENU SELECTION USING 'IF X THEN RUN' STATEMENT

```
100 PRINT "MENU SELECTION"
110 PRINT " 1 - FIRST  PROGRAM"
120 PRINT " 2 - SECOND PROGRAM"
130 PRINT " 3 - THIRD  PROGRAM"
140 :
150 INPUT "ENTER SELECTION NUMBER ";X
160 IF X=1 THEN RUN 1000
170 IF X=2 THEN RUN 2000
180 IF X=3 THEN RUN 3000
190 PRINT "ENTER NUMBER FROM 1 TO 3, PLEASE" : GOTO 150
200 :
1000 REM --- FIRST  PROGRAM ---
2000 REM --- SECOND PROGRAM ---
3000 REM --- THIRD  PROGRAM ---
```

ALL VARIABLES ARE RESET BY THE 'RUN' COMMAND.

==========

2.21 CARTESIAN COORDINATES FROM POLAR EQUATIONS.

```
100 FOR TH = 0 TO 180          : REM PLOT MANY POINTS
110 R = COS(TH / 57.2958)      : REM SAMPLE POLAR FUNCTION
120 GOSUB 200                  : REM CONVERT TO (X,Y)
130 NEXT TH : END
140 :
200 REM --- CONVERT RADIUS R AND ANGLE TH TO (X,Y) ---
210 :
220 X = R * SIN(TH / 57.2958)
230 Y = R * COS(TH / 57.2958)
240 :
250 REM --- NOW SCALE AND PLOT COORDINATE (X,Y) ---
260 :
270 RETURN
```

==========

2.22 SHELL SORT TO PUT ARRAY Z() IN ASCENDING ORDER

```
100 D = N                    :REM N=# OF ELEMENTS IN ARRAY Z()
110 D = INT(D/2)
120 FOR I=1 TO N-D
130 IF Z(I) <= Z(I+D) THEN 160
140 T=Z(I) : Z(I)=Z(I+D) : Z(I+D)=T
150 IF I>D THEN IF Z(I-D) > Z(I) THEN I=I-D : GOTO 140
160 NEXT I
170 IF D > 1 THEN 110
180 RETURN
```

CHAPTER 3 --- FUNCTIONS

3.01 RANDOM NUMBER FUNCTION

```
J=RND(X) PRODUCES THE FOLLOWING DEPENDING ON X:

X>0   J = A RANDOM NUMBER BETWEEN 0 AND 1.
X=0   J = THE NUMBER PRODUCED FROM THE LAST RND() AGAIN.
X<0   J = A RANDOM NUMBER BETWEEN 0 AND 1 WHICH IS
          ALWAYS THE SAME FOR A GIVEN X.  BY SEEDING
          RND() WITH THE SAME X A SEQUENCE IS REPEATED.
```

==========

3.02 RANDOMIZE FUNCTION

```
100 POKE 318,237
110 X=INP(95)
120 J=RND(-X*2-1)

X = A RANDOM INTEGER BETWEEN 0 AND 127, INCLUSIVE.
USE ONLY ODD NEGATIVE NUMBERS TO SEED THE RND() FUNCTION.
THE USER HAS NO CONTROL OVER WHICH SEQUENCE IS SELECTED.
THROUGHOUT THE REST OF THE PROGRAM USE  J=RND(1).
```

==========

3.03 INT FUNCTION EXAMPLES.

```
xx.yy > 0     INT(xx.yy) = xx          INT(7.36) = 7

xx.yy < 0     INT(xx.yy) = xx - 1      INT(-7.3) = -8
```

==========

3.04 SGN() FUNCTION

```
SGN(X) = 1  WHEN  X > 0.
SGN(X) = 0  WHEN  X = 0.
SGN(X) = -1 WHEN  X < 0.
```

3.05 DEF FUNCTION RULES

THE FIRST 2 LETTERS MUST BE 'FN'.
THE THIRD CHARACTER MUST BE A LETTER.
ADDITIONAL CHARACTERS ARE OPTIONAL.
THE NAME CANNOT CONTAIN ANY RESERVED WORDS.
ONLY THE 2 CHARACTERS AFTER THE FN MAKE THE NAME UNIQUE.

EXAMPLES: FNA(FNX1(FNAA(FNSQ(FNUPPER(

THE ARGUMENT VARIABLE IS A DUMMY USED TO 'MAP' VALUES
INTO THE FUNCTION IF DESIRED.
OTHER VARIABLES IN THE FUNCTION USE THEIR CURRENT VALUES.

```
EXAMPLE:  100 DEF FNA(J)= J * J + X
          110 J=5 : X=3 : PRINT FNA(2)
```

OUTPUT : 7

SINCE J WAS THE DUMMY ARGUMENT, IT MAPPED A 2 INTO THE
FUNCTION RATHER THAN USE ITS CURRENT VALUE OF 5.
THE CURRENT VALUE OF X WAS USED. THUS, 7 = 2 * 2 + 3.

==========

3.06 DECIMAL POINT ALIGNMENT

```
100 DEF FNA(J)= (J=0) - (ABS(J)<1) - LEN(STR$(INT(J)))
110 PRINT TAB(T+FNA(X));X
```

ALL NUMBERS X WILL BE PRINTED WITH THEIR DECIMAL POINTS
ALIGNED IN COLUMN T.

==========

3.07 TABULATED PRINTING OF DOLLARS AND CENTS.

```
100 DEF FNA(J) = (J=0) - (ABS(J)<1) - LEN(STR$(INT(J)))
110 FOR I=1 TO 4 : READ X : GOSUB 200 : NEXT I : END
120 :
130 DATA 1090, -1090.1, 98.51, 96.3372 : REM 4 EXAMPLES
140 :
200 A$ = MID$(STR$(ABS(X) - INT(ABS(X)) + 1.005),4,2)
210 PRINT TAB(15 + FNA(X));STR$(INT(X));".";A$
220 RETURN
```

```
OUTPUT:   1090.00        DECIMAL POINTS ARE ALIGNED.
         -1090.10        ONLY TWO DECIMAL DIGITS PRINTED.
            98.51        FRACTIONAL PENNIES ROUNDED OFF.
            96.34
```

3.08 ROUND OFF FUNCTION

```
100 DEF FNA(X) = INT( X * 10 ^ J + 0.5 ) / 10 ^ J
```

THE FUNCTION FNA ROUNDS OFF A NUMBER TO J DECIMAL PLACES.

```
EXAMPLE:  110 J=3 : PRINT FNA(2/3)

OUTPUT :  0.667
```

==========

3.09 ASCII CODE FOR A HEX DIGIT

```
100 DEF FNA(J) = J + 48 - 7*(J>9)

EXAMPLE:  PRINT CHR$(FNA(14))
OUTPUT:   D
```

==========

3.10 FIX COMMAND

```
100 DEF FNA(J) = SGN(J) * INT(ABS(J))

EXAMPLE:  PRINT FNA(-5.2)
OUTPUT:   -5
```

==========

3.11 BASE 10 LOGORITHM

```
100 DEF FNLOG10(J) = LOG(J) * 0.4342945

EXAMPLE:  PRINT FNLOG10(1000)
OUTPUT:   3
```

==========

3.12 PI = 3.14159 = 355/113

```
57.2958 = 180 / PI

100 DEF FND(J) = J * PI  / 180 : REM DEGREES TO RADIANS
110 DEF FNR(J) = J * 180 / PI  : REM RADIANS TO DEGREES
```

CHAPTER 4 --- KEYBOARD

4.01 ACCEPT 'YES' ANSWER WITH INPUT OF 'Y', 'YE' OR 'YES'

```
100 INPUT "ENTER 'YES' OR 'NO'";A$
110 IF LEFT$("YES",LEN(A$))=A$ THEN PRINT "YES"
```

==========

4.02 KEYBOARD SCAN OR 'GET' COMMAND

```
100 POKE 318,195 : POKE 320,224
110 X=INP(9) : A$=CHR$(X)

X = ASCII VALUE OF KEY DEPRESSED.  'RETURN' NOT REQUIRED.
X = 0 IF NO KEY IS DEPRESSED.
A$= CHARACTER KEYED.
```

==========

4.03 PRINT ON LINE AFTER INPUT STATEMENT.

```
100 B$="ENTER EXAMPLE TEXT"
110 PRINT B$;:INPUT A$
120 PRINT TAB(LEN(B$)+LEN(A$)+5);CHR$(23);"THANK YOU"

THE ABOVE EXAMPLE TABS BEYOND THE LENGTH OF THE USER'S
INPUT TO PRINT THE COMPUTER'S RETURN RESPONSE ON THE
SAME LINE AS THE INPUT.
```

==========

4.04 INPUT STRINGS CONTAINING COMMAS AND '@' SYMBOLS

```
100 POKE 318,195 : POKE 320,224
110 GOSUB 200 : PRINT A$ : END
120 :
200 A$="" : REM INPUT A$ UNTIL <CR> RECEIVED
210 J = INP(9) : IF J=13 THEN RETURN
220 A$ = A$ + CHR$(J) : GOTO 210
```

4.05 KEYBOARD SCAN OF 'GRAPHIC', 'CONTROL', AND 'SHIFT'.

```
100 J=INP(254) AND 31

      J = 31  IF 'CONTROL' KEY IS DEPRESSED.
      J = 21  IF 'GRAPHIIC' KEY IS DEPRESSED.
      J =  7  IF 'SHIFT' KEY IS DEPRESSED.
      J =  5  IF 'GRAPHIC - SHIFT' KEYS ARE DEPRESSED.
      J = 23  IF 'NO KEY IS DEPRESSED.
```

'SHIFT LOCK' KEY MUST BE DOWN.

==========

4.06 WAIT FOR USER TO PRESS 'SHIFT' KEY TO CONTINUE.

```
100 PRINT "PRESS 'SHIFT' TO CONTINUE"
110 WAIT 254,31,23
120 PRINT "I WAS WAITING FOR YOU.  THANKS."
```

CHAPTER 5 --- VIDEO

5.01 DOUBLE SPACE LISTINGS.

```
POKE 322,0 : LIST
```

==========

5.02 SCREEN POKE ADDRESS FOR ANY ROW AND COLUMN.

```
100 DEF FNA(J) = R * 64 + C - 3968

R = ROW    NUMBER FROM 0 TO 29.
C = COLUMN NUMBER FROM 0 TO 63.
```

==========

5.03 MAKE THE CURSOR DISAPPEAR AFTER A PRINT.

```
100 PRINT CHR$(17); : POKE -3968,32
```

OR, JUST REPLACE THE CHARACTER UNDER THE CURSOR WITH:

```
100 POKE 260,232 : POKE 261,233 : J=USR(0)
```

==========

5.04 REMOVE CURSOR FROM SCREEN.

```
100 PRINT CHR$(12) : POKE -3904,32
```

==========

5.05 SCREEN ADDRESSING

ADDRESS OF THE FIRST COLUMN IN EACH VIDEO LINE.

LINE#	HEX	POKE DECIMAL	LINE#	HEX	POKE DECIMAL
1	F080	-3968	16	F440	-3008
2	F0C0	-3904	17	F480	-2944
3	F100	-3840	18	F4C0	-2880
4	F140	-3776	19	F500	-2816
5	F180	-3712	20	F540	-2752
6	F1C0	-3648	21	F580	-2688
7	F200	-3584	22	F5C0	-2624
8	F240	-3520	23	F600	-2560
9	F280	-3456	24	F640	-2496
10	F2C0	-3392	25	F680	-2432
11	F300	-3328	26	F6C0	-2368
12	F340	-3264	27	F700	-2304
13	F380	-3200	28	F740	-2240
14	F3C0	-3136	29	F780	-2176
15	F400	-3072	30	F7C0	-2112

5.06 SYNC SCREEN MOTION WITH VIDEO HORIZONTAL SYNC.

```
100 WAIT 254,32
```

THIS WILL REMOVE SCREEN FLICKER AND DISAPPEARANCE OF CHARACTERS DURING SCREEN MOTION DUE TO BEING OUT OF SYNC WITH THE SCREEN REFRESH CIRCUITRY.

==========

5.07 PRINT AT SUBROUTINE

```
100 CLEAR 200
110 R$=CHR$(26)      : REM CURSOR DOWN
120 C$=CHR$(19)      : REM CURSOR RIGHT
130 FOR J=1 TO 6 : R$=R$+R$ : C$=C$+C$ : NEXT J
140 R$=CHR$(17)+R$   : REM ADD CURSOR HOME
150 :
160 REM --- PRINT TEXT AT ROW R, COLUMN C ---
170 :
180 R=5 : C=10 : A$="THIS IS AN EXAMPLE" : GOSUB 1000
190 END
200 :
1000 REM --- PRINT AT SUBROUTINE ---
1010 :
1020 PRINT LEFT$(R$,R);LEFT$(C$,C);A$ : RETURN
```

==========

5.08 PLACE CURSOR AT ROW AND COLUMN.

THIS ROUTINE MUST BE LOCATED STARTING AT ADDRESS 0, BECAUSE THE 'OUT' INSTRUCTION IS MADE INTO A RST 0H COMMAND.

```
0000: E5          ENTRY PUSH HL
0001: CD A2 E1          CALL 0E1A2H       ;GET IY
0004: 2A 07 01          LD   HL,(0107H)   ;GET ROW #
0007: 26 00             LD   H,0
0009: 29                ADD  HL,HL        ;*2
000A: 29                ADD  HL,HL        ;*4
000B: 29                ADD  HL,HL        ;*8
000C: 29                ADD  HL,HL        ;*16
000D: 29                ADD  HL,HL        ;*32
000E: 29                ADD  HL,HL        ;*64
000F: FD 75 68          LD   (IY+68H),L   ;ROW ADDRESS
0012: FD 74 69          LD   (IY+69H),H
0015: FD 77 6A          LD   (IY+6AH),A   ;COL ADDRESS
0018: CD CC E9          CALL 0E9CCH       ;MOVE CURSOR
001B: 36 20             LD   (HL),20H     ;STORE SPACE
001D: E1                POP  HL
001E: C1                POP  BC           ;RELIEVE STACK
001F: C9                RET
```

CALLING THE ROUTINE WILL MOVE THE CURSOR TO THE
REFERENCED SCREEN POSITION AND PLACE A SPACE THERE.

```
EXAMPLE:  100 POKE 262,199 : REM INSERT RST 0H COMMAND
          110 OUT 7,15     : REM CURSOR TO ROW 7, COL 15
          120 PRINT "TEXT STARTS ON ROW 7, COLUMN 15"
```

==========

5.09 PLACE CURSOR AT ROW AND COLUMN.

```
0000:             ROW    DEFB 0
0001:             COL    DEFB 0
                  ;
0002: CD A2 E1    ENTRY  CALL 0E1A2H        ;GET IY
0005: 2A 00 00           LD   HL,(ROW)
0008: 7C                 LD   A,H
0009: 26 00              LD   H,0
000B: 29                 ADD  HL,HL         ;*2
000C: 29                 ADD  HL,HL         ;*4
000D: 29                 ADD  HL,HL         ;*8
000E: 29                 ADD  HL,HL         ;*16
000F: 29                 ADD  HL,HL         ;*32
0010: 29                 ADD  HL,HL         ;*64
0011: FD 75 68           LD   (IY+68H),L    ;ROW ADDRESS
0014: FD 74 69           LD   (IY+69H),H
0017: FD 77 6A           LD   (IY+6AH),A    ;COL ADDRESS
001A: CD CC E9           CALL 0E9CCH        ;MOVE CURSOR
001D: 36 20              LD   (HL),20H      ;STORE SPACE
001F: C9                 RET
```

```
POKE ROW    NUMBER IN ADDRESS 0.  RANGE 0 - 29.
POKE COLUMN NUMBER IN ADDRESS 1.  RANGE 0 - 63.
```

CALLING THE ROUTINE WILL MOVE THE CURSOR TO THE
REFERENCED SCREEN POSITION AND PLACE A SPACE THERE.

```
EXAMPLE:  100 POKE 260,2 : POKE 261,0 : REM USR() ADDR
          110 R=7 : C=15 : GOSUB 200
          120 PRINT "TEXT STARTS ON ROW 7, COLUMN 15"
          130 STOP
          140 :
          200 POKE 0,R : POKE 1,C : J=USR(0) : RETURN
```

5.10 DRAW BOX ROUTINE

```
;
;  THIS ROUTINE WILL DRAW A BOX AROUND A TEXT STRING.
;  PUT STARTING LOCATING OF STRING IN ROW AND COL, AND ITS
;  SIZE IN LENGTH.
;
;  THIS ROUTINE USES THE CURSOR PLACEMENT ROUTINE OF 5.09
;
                    CURSOR  EQU  02H
;
                            ORG  20H
0020: 00            LENGTH  DEFB 0
;
                            ORG  2430H
2430: 21 00 00      BOX     LD   HL,ROW       ;STRING START
2433: 35                    DEC  (HL)
2434: 23                    INC  HL
2435: 35                    DEC  (HL)         ;BOX CORNER
2436: 3A 20 00              LD   A,(LENGTH)
2439: 47                    LD   B,A
243A: CD 00 02              CALL CURSOR       ;CURSOR TO CORNER
243D: 36 BC                 LD   (HL),0BCH    ;TOP LEFT CORNER
243F: 11 40 00              LD   DE,040H      ;DOWN ONE ROW
2442: 19                    ADD  HL,DE
2443: 36 A2                 LD   (HL),0A2H    ;LEFT SIDE
2445: 19                    ADD  HL,DE        ;DOWN ONE ROW
2446: 36 BE                 LD   (HL),0BEH    ;BOTTOM CORNER
2448: E5                    PUSH HL           ;SAVE CORNER ADDR
2449: 11 80 FF              LD   DE,0FF80H    ;UP THREE ROWS
244C: 19                    ADD  HL,DE
244D: D1                    POP  DE
244E: 23            HORZ    INC  HL           ;HL = TOP ADDR
244F: 13                    INC  DE           ;DE = BOTTOM ADDR
2450: 36 97                 LD   (HL),097H    ;DRAW TOP LINE
2452: EB                    EX   DE,HL
2453: 36 97                 LD   (HL),097H    ;DRAW BOTTOM LINE
2455: EB                    EX   DE,HL
2456: 10 F6                 DJNZ HORZ-$       ;LOOP ON LENGTH
2458: 11 40 00              LD   DE,040H
245B: 23                    INC  HL
245C: 36 BD                 LD   (HL),0BDH    ;TOP RIGHT CORNER
245E: 19                    ADD  HL,DE
245F: 36 A2                 LD   (HL),0A2H    ;RIGHT SIDE
2461: 19                    ADD  HL,DE
2462: 36 BF                 LD   (HL),0BFH    ;BOTTOM CORNER
2464: 3E 00                 LD   A,000H
2466: 2E 00                 LD   L,000H
2468: C3 09 00              JP   CURSOR+7     ;CURSOR TO HOME
```

5.11 CREATE INVERSE VIDEO CHARACTER SET

```
100 FOR J=-1024 TO -1
110 POKE J,256 + NOT PEEK(J-1024)
120 NEXT J

USE THE 'GRAPHIC' AND THE 'SHIFT GRAPHIC' KEYS TO DISPLAY
THE INVERSE VIDEO CHARACTER SET.

THE SAME THING IS DONE BELOW IN MACHINE LANGUAGE:

0000: E5          ENTRY PUSH HL         ;SAVE REGS
0001: D5                PUSH DE
0002: C5                PUSH BC
0003: F5                PUSH AF
0004: 21 00 F8          LD   HL,0F800H ;CHARACTER SOURCE
0007: 11 00 FC          LD   DE,0FC00  ;DESTINATION
000A: 01 00 04          LD   BE,0400H  ;1024 COUNTER
000D: 7E          LOOP  LD   A,(HL)    ;GET A ROW
000E: 2F                CPL            ;INVERT IT
000F: 12                LD   (DE),A    ;MOVE IT DOWN
0010: 23                INC  HL        ;HL = HL + 1
0011: 13                INC  DE        ;DE = DE + 1
0012: 0B                DEC  BC        ;COUNT IT
0013: 20 F8             JR   NZ,LOOP-$ ;REPEAT TIL DONE
0015: F1                POP  AF
0016: C1                POP  BC
0017: D1                POP  DE
0018: E1                POP  HL        ;RESTORE REGS
0019: C9                RET


NOW THAT THE INVERSE CHARACTER SET HAS BEEN CREATED,
LET'S PRINT INVERSE TEXT FROM THE STRING A$.

200 INPUT A$ : GOSUB 300 : GOTO 200
300 :
310 REM  THIS ROUTINE PRINTS A$ IN INVERSE VIDEO
320 :
330 FOR J=1 TO LEN(A$)
340 PRINT CHR$(ASC(MID$(A$,J)) OR 128); : NEXT J : RETURN
```

5.12 CREATE DOUBLE WIDE CHARACTER SET

64 ASCII CHARACTERS FROM ASCII 20H (SPACE) THROUGH ASCII 5FH (UNDERSCORE) WILL BE MADE INTO DOUBLE WIDE CHARACTERS USING THE 128 AVAILABLE GRAPHIC CHARACTERS. THIS DOUBLE WIDE CHARACTER SET INCLUDES THE DIGITS, SYMBOLS, AND THE UPPER CASE LETTERS.

THIS ROUTINE WILL CREATE THE DOUBLE WIDE CHARACTER SET.

```
100 FOR J= 32 TO 95 : I=(J-256)*8 : K=(2*J-192)*8
110 FOR L= 0 TO 7 : M=PEEK(I+L) : N=INT(M/16): P=M AND 15
120 R=0 : S=0 : FOR T= 0 TO 3 : V=2^T
130 R=R+(N AND V)*V*3 : S=S+(P AND V)*V*3 : NEXT T
140 POKE K+L,R : POKE K+L+8,S : NEXT L,J
```

A CHARACTER WITH ASCII VALUE J CAN BE PRINTED IN DOUBLE WIDTH USING THE TWO GRAPHIC CHARACTERS WITH ASCII VALUE OF 2*J + 64 AND 2*J + 65. SEE THE FOLLOWING EXAMPLE.

```
150 A$="SAMPLE TEXT STRING TO PRINT"
160 FOR I=1 TO LEN(A$) : J=ASC(MID$(A$,I))*2+64
170 PRINT CHR$(J);CHR$(J+1); : NEXT I
```

OR, YOU MAY WISH TO HAVE THE CHARACTERS SENT TO THE VIDEO DISPLAYED IN DOUBLE SIZE AUTOMATICALLY. THIS CAN BE ACCOMPLISHED USING THE FOLLOWING VIDEO DRIVER. CHANGE THE >SE O=xxyy VECTOR ADDRESS TO POINT TO THIS ROUTINE, IE. >SE O=0000, OR POKE 32720,0 : POKE 32721,0 FOR 32K.

```
0000: FE 20         CP   32        ;CHAR BEFORE 'SPACE'
0002: FA 1B E0      JP   M,VIDEO   ;YES, EXIT TO VIDEO
0005: 87            ADD  A,A       ;J*2
0006: C6 40         ADD  A,64      ;J*2+64
0008: CD 1B E0      CALL VIDEO     ;PRINT LEFT HALF
000B: 3C            INC  A         ;J*2+65
000C: C3 1B E0      JP   VIDEO     ;PRINT RIGHT HALF
```

CHAPTER 6 --- JOYSTICKS

6.01 JOYSTICK / KEYBOARD STANDARD FOR THE SORCERER

This standard has been adopted by several software houses in the USA and in Australia. Software offered by these vendors which employs joystick/keyboard control will conform to this standard. It is suggested that all SORCERER owners use this standard for international compatibility of software and hardware.

Two joysticks may be attached to the INPUT of the parallel port. UNIT #1 uses the LOW-order 4 bits, and UNIT #2 uses the HIGH-order 4 bits. Each unit may steer in the four basic directions, LEFT, RIGHT, UP, DOWN, as well as in the four diagonal directions. Both units operate independently, and simultaneous operation is permitted.

FIRE BUTTON control may be included, and has priority over directional control of the joystick unit it is attached to. FIRE BUTTON is activated by grounding both BIT 0 and BIT 1 for unit #1, and BIT 4 and BIT 5 for unit #2 .

KEYBOARD has priority over JOYSTICK, and overrides both joystick units if used. KEYBOARD INPUT RESULT is returned as the RESULT CODE of joystick unit #1, with joystick unit #2 disabled.

Keyboard directional control is via the "arrow" (normally cursor control) keys in the NUMERIC KEYPAD only. The SHIFT key need not be depressed when using these keys. FIRE BUTTON on the keyboard is the NUMERIC-PAD "5" key (HOME). Optional FIRE BUTTONS may be SKIP/TAB or SPACE BAR. FIRE BUTTON overrides directional keys on the keyboard.

In the event that both the LEFT and the RIGHT keys are pressed together, it is treated as NO INPUT. The same rule applies to depressing both the UP and the DOWN keys together. The UP/LEFT ("7"), UP/RIGHT ("9"), DOWN/LEFT ("1") and DOWN/RIGHT ("3") keys on the numeric-pad are optional.

For programming in Z80 machine code, the 8-bit INPUT RESULT CODE is returned in the A-register. No other registers are affected. If there is no input, then return with a zero in the A-register and with the Z-flag set.

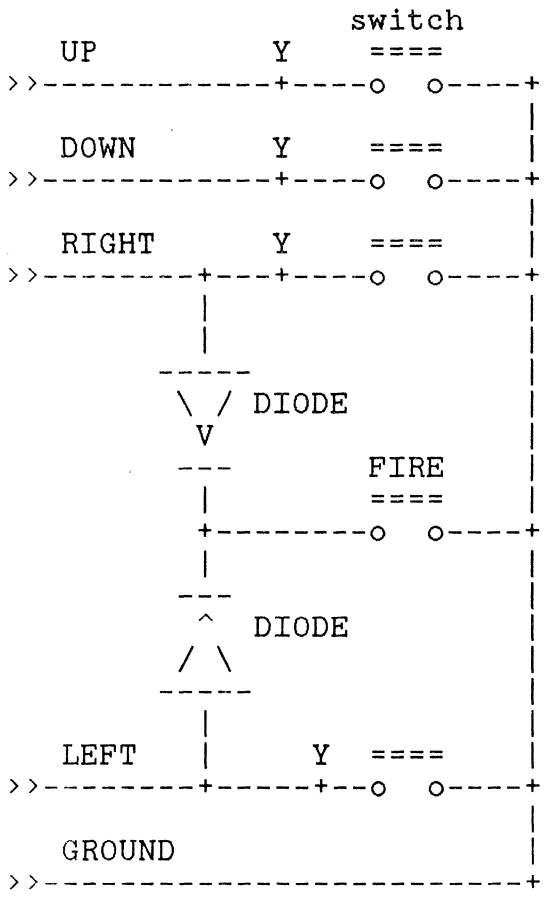
6.02 JOYSTICK INTERFACE STANDARD TO PARALLEL PORT

BIT	PIN	FUNCTION	BIT	PIN	FUNCTION
0	10	UNIT #1 LEFT	4	12	UNIT #2 LEFT
1	22	UNIT #1 RIGHT	5	24	UNIT #2 RIGHT
2	11	UNIT #1 UP	6	13	UNIT #2 UP
3	23	UNIT #1 DOWN	7	25	UNIT #2 DOWN
0-1	10/22	UNIT #1 FIRE	4-5	12/24	UNIT #2 FIRE
	8	Ground		20	+5 VOLT SUPPLY

==========

6.03 JOYSTICK CIRCUIT DIAGRAM

ATARI joysticks can be easily modified to connect directly to the parallel port. A 4.7K 1/4 watt resistor pulls-up each direction input to +5 volts at the nodes marked with 'Y' in the diagram. The FIRE BUTTON employs the two diodes between the LEFT and the RIGHT direction inputs, and GROUND connects to the common line. When the joystick selects a direction, or the fire button is pressed, a switch closes which changes the input bit from +5 volts to ground.



6.04 JOYSTICK EXAMPLE USING BASIC STATEMENTS

```
100 A = 255 - INP(255) : REM  READ PARALLEL PORT
110 IF (A AND  3)= 3 THEN "FIRE BUTTON UNIT #1"
120 IF (A AND 48)=48 THEN "FIRE BUTTON UNIT #2"

130 IF A AND   1 THEN "UNIT #1 LEFT"
140 IF A AND   2 THEN "UNIT #1 RIGHT"
150 IF A AND   4 THEN "UNIT #1 UP"
160 IF A AND   8 THEN "UNIT #1 DOWN"

170 IF A AND  16 THEN "UNIT #2 LEFT"
180 IF A AND  32 THEN "UNIT #2 RIGHT"
190 IF A AND  64 THEN "UNIT #2 UP"
200 IF A AND 128 THEN "UNIT #2 DOWN"

210 GOTO 100 : REM  SCAN AGAIN
```

Use the Basic statements to observe the state of the joysticks. The text strings tell what to do after determining the condition of the joysticks. If you branch to a routine to service Unit #1, be sure to return to the testing of Unit #2 so that it can operate simultaneous with Unit #1.

==========

6.05 JOYSTICK EXAMPLE USING MACHINE LANGUAGE CODE

The joystick source lising of section 6.06 is a useful routine that loads into memory from address 0 through A4 hex. To use the routine from a Basic program, set the USR() jump address to 0 as shown on line 100 below.

Now initialize the location of each joystick cursor by placing a screen row number and a column number in the following addresses using POKE statements: (See example on line 110 below.)

```
ADDRESS 2 - ROW #    UNIT #1, RANGE (1...30).
        3 - COLUMN # UNIT #1, RANGE (1...64).
        4 - ROW #    UNIT #2, RANGE (1...30).
        5 - COLUMN # UNIT #2, RANGE (1...64).
        6 - FIRE FOR BOTH UNITS:  = 1 UNIT #1 ONLY
                                  = 2 UNIT #2 ONLY
                                  = 3 BOTH UNITS FIRING.
        7,8  - SCREEN ADDRESS OF UNIT #1
        9,10 - SCREEN ADDRESS OF UNIT #2
```

```
100 POKE 260,0:POKE 261,0:PRINT CHR$(12):REM USR() ADDR
110 POKE 2,1:POKE 3,1:POKE 4,1:POKE 5,1 :REM INITIALIZE
120 Z=USR(0)                            :REM GET JOYSTICK
130 L=PEEK(2)*64+PEEK(3)-4033           :REM SCREEN ADDR
140 POKE L,ASC("1")                     :REM CURSOR #1
150 M=PEEK(9)+PEEK(10)*256-65536        :REM ANOTHER WAY
160 POKE M,50                           :REM CURSOR #2
170 GOTO 120                            :REM DO AGAIN
```

6.06 MACHINE LANGUAGE JOYSTICK ROUTINE

This routine reads the status of the joysticks connected to the parallel port and updates the ROW, COLUMN and FIRE variables for each joystick. The routine will keep the row variables in the range of (1...30), and the column variables in the range of (1...64). The user's Basic program can access these variables to compute the screen position of each joystick, or read directly the contents of ADDR1 and ADDR2. Section 6.05 gives an example of how Basic might access this routine.

```
0000: 18 09        ENTRY   JR    START-$
                   ;
0002: 01           ROW1    DEFB  1
0003: 01           COL1    DEFB  1
0004: 01           ROW2    DEFB  1
0005: 01           COL2    DEFB  1
0006: 00           FIRE    DEFB  0
0007: 00 00        ADDR1   DEFW  0
0009: 00 00        ADDR2   DEFW  0
                   ;
                   LEFT1   EQU   0
                   RIGHT1  EQU   1
                   UP1     EQU   2
                   DOWN1   EQU   3
                   FIRE1   EQU   3
                   LEFT2   EQU   4
                   RIGHT2  EQU   5
                   UP2     EQU   6
                   DOWN2   EQU   7
                   FIRE2   EQU   48
                   ;
000B: DB FF        START   IN    A,(0FFH)      ;GET JOYSTICK
000D: 2F                   CPL
000E: B7                   OR    A             ;ANY ACTIVITY
000F: 21 06 00             LD    HL,FIRE
0012: 36 00                LD    (HL),000H     ;REMOVE FIRES
0014: C8                   RET   Z             ;RET IF IDLE
                   ;
0015: F5                   PUSH  AF
0016: E6 03                AND   FIRE1
0018: FE 03                CP    FIRE1
001A: 20 01                JR    NZ,L001D-$
001C: 34                   INC   (HL)          ;FIRE1 ACTIVE
001D: F1           L001D   POP   AF
001E: F5                   PUSH  AF
001F: E6 30                AND   FIRE2
0021: FE 30                CP    FIRE2
0023: 20 02                JR    NZ,L0027-$
0025: 34                   INC   (HL)
0026: 34                   INC   (HL)          ;FIRE2 ACTIVE
```

```
                        ;
                        ;  H = COLUMN 1    L = ROW 1
                        ;  D = COLUMN 2    E = ROW 2
                        ;
                        ;  INCREMENT COL IF RIGHT
                        ;  INCREMENT ROW IF DOWN
                        ;
                        ;  DECREMENT COL IF LEFT
                        ;  DECREMENT ROW IF UP
                        ;
0027: F1                L0027   POP   AF
0028: 2A 02 00                  LD    HL,(ROW1)
002B: ED 5B 04 00               LD    DE,(ROW2)
002F: CB 47                     BIT   LEFT1,A
0031: 28 01                     JR    Z,L0034-$
0033: 25                        DEC   H
0034: CB 67             L0034   BIT   LEFT2,A
0036: 28 01                     JR    Z,L0039-$
0038: 15                        DEC   D
0039: CB 4F             L0039   BIT   RIGHT1,A
003B: 28 01                     JR    Z,L003E-$
003D: 24                        INC   H
003E: CB 6F             L003E   BIT   RIGHT2,A
0040: 28 01                     JR    Z,L0043-$
0042: 14                        INC   D
0043: CB 57             L0043   BIT   UP1,A
0045: 28 01                     JR    Z,L0048-$
0047: 2D                        DEC   L
0048: CB 77             L0048   BIT   UP2,A
004A: 28 01                     JR    Z,L004D-$
004C: 1D                        DEC   E
004D: CB 5F             L004D   BIT   DOWN1,A
004F: 28 01                     JR    Z,L0052-$
0051: 2C                        INC   L
0052: CB 7F             L0052   BIT   DOWN2,A
0054: 28 01                     JR    Z,L0057-$
0056: 1C                        INC   E
                        ;
                        ;  COLUMN RANGE:  1 - 64
                        ;  ROW    RANGE:  1 - 30
                        ;
0057: 3E 1F             L0057   LD    A,31          ;ROW LIMIT
0059: BD                        CP    L
005A: 20 01                     JR    NZ,L005D-$
005C: 2D                        DEC   L
005D: BB                L005D   CP    E
005E: 20 01                     JR    NZ,L0061-$
0060: 1D                        DEC   E
0061: 3E 41             L0061   LD    A,65          ;COL LIMIT
0063: BC                        CP    H
0064: 20 01                     JR    NZ,L0067-$
0066: 25                        DEC   H
0067: BA                L0067   CP    D
0068: 20 01                     JR    NZ,L006B-$
006A: 15                        DEC   D
```

```
                        ;
006B: 3E 00             L006B   LD    A,000H        ;KEEP > 0
006D: BD                        CP    L
006E: 20 01                     JR    NZ,L0071-$
0070: 2C                        INC   L
0071: BB                L0071   CP    E
0072: 20 01                     JR    NZ,L0075-$
0074: 1C                        INC   E
0075: BC                L0075   CP    H
0076: 20 01                     JR    NZ,L0079-$
0078: 24                        INC   H
0079: BA                L0079   CP    D
007A: 20 01                     JR    NZ,L007D-$
007C: 14                        INC   D
                        ;
007D: 22 02 00          L007D   LD    (ROW1),HL     ;ROW1 - COL1
0080: ED 53 04 00               LD    (ROW2),DE     ;ROW2 - COL2
0084: CD 94 00                  CALL  ADDRESS
0087: 22 07 00                  LD    (ADDR1),HL    ;SCREEN ADDR1
008A: 2A 04 00                  LD    HL,(ROW2)
008D: CD 94 00                  CALL  ADDRESS
0090: 22 09 00                  LD    (ADDR2),HL    ;SCREEN ADDR2
0093: C9                        RET
                        ;
                        ;  CONVERT ROW AND COL TO A SCREEN
                        ;  ADDRESS STORED IN ADDR1 AND ADDR2
                        ;
                        ;  ADDR = SCREEN BASE + ROW * 64 + COL
                        ;
0094: 5C                ADDRESS LD    E,H           ;DE = COL
0095: 16 00                     LD    D,000H
0097: 26 00                     LD    H,000H        ;HL = ROW
0099: 29                        ADD   HL,HL         ; *2
009A: 29                        ADD   HL,HL         ; *4
009B: 29                        ADD   HL,HL         ; *8
009C: 29                        ADD   HL,HL         ; *16
009D: 29                        ADD   HL,HL         ; *32
009E: 29                        ADD   HL,HL         ;HL = ROW*64
009F: 19                        ADD   HL,DE         ;ROW + COL
00A0: 11 3F F0                  LD    DE,0F03FH     ;SCREEN BASE
00A3: 19                        ADD   HL,DE
00A4: C9                        RET
```

CHAPTER 7 --- SOUND

7.01 PRINCIPLE OF ONE VOICE SOUND

Generating sound from your Sorcerer is not that difficult, and this discussion will help you get started by detailing a machine language routine to generate music.

Let's use the parallel port as a means to output a signal to an external speaker since the Sorcerer does not have an internal speaker like some other computers do. I DO NOT recommend connecting a small 8 ohm speaker directly between one of the parallel port's output bits and ground. It is safer to have an output bit drive a transistor's base via a 1K resistor, and let the transistor switch current through a small speaker. Or, you may wish to use the music interface board that comes with our four voice Music System.

The statement: 10 FOR I=1 TO 100:OUT 255,255:OUT 255,0:NEXT I generates a low pitched note by toggling the output bits on the parallel port from high to low, back to high. It may be sufficient for your needs, but provides no control over the pitch of the sound. However, it illustrates the principle of how sound will be generated in machine language.

The following code generates notes.

```
0000: F5        ENTER PUSH AF             ;SAVE REGISTERS USED
0001: C5              PUSH BC
0002: E5              PUSH HL
0003: 21 60 00        LD   HL,DURATION    ;LENGTH OF SOUND
0006: 79        TOP   LD   A,C
0007: 2F              CPL                 ;TOGGLE OUTPUT BITS
0008: 4F              LD   C,A
0009: 06 40           LD   B,PITCH        ;FREQUENCY CONTROL
000B: D3 FF           OUT  (0FFH),A       ;TO PARALLEL PORT
000D: 10 FE     LOOP  DJNZ LOOP-$         ;DELAY
000F: 2B              DEC  HL             ;DOWN COUNT DURATION
0010: 7D              LD   A,L
0011: B4              OR   H              ;IS COUNT = 0 ?
0012: 20 F2           JR   NZ,TOP-$       ;LOOP WHILE HL<>0
0014: E1              POP  HL
0015: C1              POP  BC
0016: F1              POP  AF             ;RESTORE REGISTERS
0017: C9              RET                 ;RETURN FROM SUBROUTINE
```

You can vary the pitch by changing the values loaded into register B (ie. contents of address 0AH.) How long the note is played is controlled by the duration value loaded into HL (ie. the contents of address 04H and 05H.)

The routine above suffers from the effect of having the delay loop nested inside of the duration loop. If the delay loop is tight, then the duration is accordingly shortened, since the total time in the routine is the product of the two loop parameters. A routine which has a duration independent of the frequency is given in 7.02

7.02 ONE VOICE MUSIC ROUTINE

```
           FREQ          EQU  0FCH      ;STORE PITCH # IN 253
           DURATION      EQU  0FDH      ;DURATION # IN 254 - 255.
           SPEED         EQU  14FFH     ;TEMPO FOR ENTIRE SONG.
                         ;
                         ORG  0D0H      ;LOCATE ON ZERO PAGE
00D0: F5                 PUSH AF        ;SAVE REGISTERS
00D1: C5                 PUSH BC
00D2: D5                 PUSH DE
00D3: E5                 PUSH HL
00D4: FD E5              PUSH IY
00D6: 21 FC 00    ENTER  LD   HL,FREQ
00D9: 4E                 LD   C,(HL)    ;GET FREQUENCY #
00DA: 11 FF 14           LD   DE,SPEED
00DD: FD 2A FD 00        LD   IY,(DUR)  ;GET DURATION
00E1: D3 FF       LOOP   OUT  (0FFH),A  ;TO PARALLEL PORT
00E3: 0D                 DEC  C         ;IS IT TIME TO TOGGLE
00E4: C2 E9 00           JP   NZ,SKIP
00E7: 2F                 CPL            ;TOGGLE OUTPUT
00E8: 4E                 LD   C,(HL)    ;RELOAD FREQUENCY COUNT
00E9: FD 19       SKIP   ADD  IY,DE
00EB: DA E1 00           JP   C,LOOP    ;LOOP UNTIL DURATION UP
00EE: FD E1              POP  IY
00F0: E1                 POP  HL
00F1: D1                 POP  DE
00F2: C1                 POP  BC
00F3: F1                 POP  AF         ;RESTORE REGISTERS
00F4: C9                 RET             ;RETURN
```

To use the above routine from Basic, store the frequency number in byte 253 with POKE 253,FREQ. Store the duration in bytes 254, and 255 with POKE 254,DURATION:POKE 255,0. Usually control from (254) is sufficient, and keep (255) at zero.

Call the routine through the USR() function by poking the entry address into 260, and 261: POKE 260,214:POKE 261,0:X=USR(0).

Here are frequency numbers to generate notes for one octave. Higher octaves can be obtained by dividing these numbers by 2, 4, and 8, etc.

```
C  - 268    C# - 253    D  - 239    D# - 226
E  - 213    F  - 201    F# - 190    G  - 179
G# - 169    A  - 160    A# - 151    B  - 142
```

Although the above music routine is fun and easy, its usefulness is no comparison to the enjoyment you'll get from the four voice Music System from Arrington Software Service.

7.02 ONE VOICE MUSIC ROUTINE

```
          FREQ          EQU  0FCH      ;STORE PITCH # IN 252
          DURATION      EQU  0FDH      ;DURATION # IN 254 - 253.
          SPEED         EQU  FFFFH     ;TEMPO FOR ENTIRE SONG.
                        ;
                        ORG  0D0H      ;LOCATE ON ZERO PAGE
00D0: F5                PUSH AF        ;SAVE REGISTERS
00D1: C5                PUSH BC
00D2: D5                PUSH DE
00D3: E5                PUSH HL
00D4: FD E5             PUSH IY
00D6: 21 FC 00    ENTER LD   HL,FREQ
00D9: 4E                LD   C,(HL)    ;GET FREQUENCY #
00DA: 11 FF FF          LD   DE,SPEED
00DD: FD 2A FD 00       LD   IY,(DUR)  ;GET DURATION
00E1: D3 FF       LOOP  OUT  (0FFH),A ;TO PARALLEL PORT
00E3: 0D                DEC  C         ;IS IT TIME TO TOGGLE
00E4: C2 E9 00          JP   NZ,SKIP
00E7: 2F                CPL            ;TOGGLE OUTPUT
00E8: 4E                LD   C,(HL)    ;RELOAD FREQUENCY COUNT
00E9: FD 19       SKIP  ADD  IY,DE
00EB: DA E1 00          JP   C,LOOP    ;LOOP UNTIL DURATION UP
00EE: FD E1             POP  IY
00F0: E1                POP  HL
00F1: D1                POP  DE
00F2: C1                POP  BC
00F3: F1                POP  AF         ;RESTORE REGISTERS
00F4: C9                RET             ;RETURN
```

To use the above routine from Basic, store the frequency number in byte 252 with POKE 252,FREQ. Store the duration in bytes 254, and 253 with POKE 254,DURATION:POKE 253,0. Usually control from (254) is sufficient, and keep (253) at zero.

Call the routine through the USR() function by poking the entry address into 260, and 261: POKE 260,208:POKE 261,0:X=USR(0).

Here are frequency numbers to generate notes for one octave. Higher octaves can be obtained by dividing these numbers by 2, 4, and 8, etc.

C - 268	C# - 253	D - 239	D# - 226
E - 213	F - 201	F# - 190	G - 179
G# - 169	A - 160	A# - 151	B - 142

Although the above music routine is fun and easy, its usefulness is no comparison to the enjoyment you'll get from the four voice Music System from Arrington Software Service.

7.03 PIEZO SPEAKER AUDIO PROMPTER

A miniature 4.7 KHz piezo speaker can be purchased at Radio Shack for three dollars and connected directly to the Sorcerer's parallel port. Connect the red wire to pin 4, and the black wire to pin 8.

The piezo speaker is a self contained chamber which resonates at a fixed frequency when activated with a voltage between 3 and 9 volts. Although I chose to connect the red wire to bit 7 of the port, any of the eight bit outputs will work equally well. The statement of OUT 255,128 causes bit 7 of the port to go to a logical high, thus sourcing the speaker with 5 volts. The 5 volts causes the speaker to continually sound until bit 7 is returned to a logical low with an OUT 255,0. The 255 is the address of the parallel port.

Sending the command sequence of OUT 255,128:OUT 255,0 will cause the speaker to chirp since it sees a brief 5 volt pulse. This audio addition to the Sorcerer is inexpensive and has found frequent use in programs to signal an event in a program's execution, or to prompt the user that an input is required at the keyboard.

This speaker cannot be used to reproduce the music spoken of in 7.01, 7.02, and 7.04. Also, it will not produce the sound that accompanies much of the game software on the market.

==========

7.04 CONTROL 'G' BEEP

THIS ROUTINE INTERCEPTS CONTROL 'G' CHARACTERS SENT TO THE VIDEO DRIVER AND OUTPUTS A BEEP. USE THE MUSIC INTERFACE BOARD FROM OUR MUSIC SYSTEM.

ACTIVATE WITH >SE 0=0

```
0000: FE 07      ENTER  CP   7              ;CONTROL G ?
0002: C2 F0 E9          JP   NZ,VIDEO       ;EXIT IF NOT
0005: C5                PUSH BC
0006: E5                PUSH HL
0007: 21 60 00          LD   HL,DURATION    ;LENGTH OF
000A: 79           TOP  LD   A,C
000B: 2F                CPL                 ;TOGGLE OUTPUT
000C: 4F                LD   C,A
000D: 06 40             LD   B,PITCH        ;FREQUENCY
000F: D3 FF             OUT  (0FFH),A       ;PARALLEL PORT
0011: 10 FE        LOOP DJNZ LOOP-$         ;DELAY
0013: 2B                DEC  HL             ;DOWN COUNT
0014: 7D                LD   A,L
0015: B4                OR   H              ;IS COUNT=0 ?
0016: 20 F2             JR   NZ,TOP-$       ;LOOP IF HL<>0
0018: E1                POP  HL
0019: C1                POP  BC
001A: C9                RET                 ;RETURN
```

CHAPTER 8 --- BASIC ROMPAC

8.01 BASIC ROMPAC MAP by P. HOLMICK

NOTES: (1) Floating Point variables occupy 4 bytes.
(2) ACC is the floating point accumulator 01BF to 01C2.
(01BF) = Least significant byte.
(01C0) = Next most significant byte.
(01C1) = Most significant byte.
(01C2) = Exponent.
(3) NTF is the Number Type Flag (0190).
(0190) = 1 for Strings
(0190) = 0 for Numbers
(4) For arithmetic operations requiring 2 operands, one is usually the ACC, and the other is either (HL) or BCDE, where BCDE contains:
B = Exponent.
C = Most significant byte.
D = Next most significant byte.
E = Least significant byte.

(5) HL is used as the text pointer for most of the comparison or conversion routines.

ADDRESS FUNCTIONAL DESCRIPTION

C000: Moves a block of memory from C258-C2A6 to 0100-014E.
C06B: BASIC Warm Start
C075: "BYTES FREE" string.
C080: String address for double CR-LF.
C082: String address for single CR-LF.
C085: "EXIDY STANDARD BASIC VER 1.0 .....etc." message string.
C0C6: Command Jump Table.

D606,D6CA,D61C,0103,CEE9,D24A,CF17,D8BA,D999,D4AB,D908,
DA0E,DA14,DA75,DA8A,D6FE,D18B,CF9F,D225,D19A,D1AB,D1BB,
D1EB,D1F5

C0F6: BASIC Token Table.

END,FOR,NEXT,DATA,BYE,INPUT,DIM,READ,LET,GOTO,RUN,IF,
RESTORE GOSUB,RETURN,REM,STOP,OUT,ON,NULL,WAIT,DEF,POKE,
PRINT,CONT,LIST CLEAR,CLOAD,CSAVE,NEW,TAB(,TO,FN,SPC(,
THEN,NOT,STEP,+,-,*,/,^, AND,OR,>,=,<,SGN,INT,ABS,USR,
FRE,INP,POS,SQR,RND,LOG,EXP,COS,SIN,TAN,ATN,PEEK,LEN,
STR$,VAL,ASC,CHR$,LEFT$,RIGHT$,MID$

C1E1: Command Jump Table continued.

C709,C62E,CB34,C8B5,E003,CA43,CD4F,CA72,C8CC,C872,C855,
C944,C6DD,C861,C890,C8B7,C707,D256,C926,C748,D25C,CF1F,
D705,C968,C735,C5C6,C80F,D341,D2C9,C41A

C21D: Jump Table for Arithmetic Operators (+,-,*,/,^,AND,OR)
D7A2,D3AA,D4EA,D54B,D8C3,CCA8,CCA7
Includes a one-byte "precedence" value for each operator which tells the Interpreter the order of arithmetic computations.

C232: Start of Error Codes Table (2 Bytes each)
C258: Start of 4EH bytes to be copied into BASIC CONTROL AREA from 0100H
C2A3: " ERROR" message
C2AA: " IN " message.
C2AF: "READY<CR><LF>" message.
C2B7: "BREAK" message.
C2BD: Used by <FOR>...<NEXT> loops to manipulate SP so that <NEXT> knows where the last <FOR> statement ends.
C2E0: Gets variable name from a line of BASIC - loads it into the variable area.
C2E6: Move bottom of program further down in memory.
BC = Destination address.
DE = Stopping address, ie. stop when HL = DE.
HL = End of program address, ie. source address.

C2F1: Check to see that there's enough free memory for next operation.
C309: OM ERROR
C30E: Puts last-used <DATA> line number into (0147) so that BASIC will print "?SN ERROR IN xx". Where xx=Data Line Number
C314: SN ERROR
C317: /0 ERROR
C31A: NF ERROR
C320: UF ERROR
C322: Prints error message. To use this routine, load the E register with 0 to 24H before jumping here.

Error	Message	E = nn
NF	Next Without For	00
SN	Syntax	02
RG	Return Without Gosub	04
OD	Out of Data	06
FC	Illegal Function Call	08
OV	Calculation Overflow	0A
OM	Out of Memory	0C
UL	Undefined Line Number	0E
BS	Bad Subscript	10
DD	Redimensioned Array	12
/0	Division by Zero	14
ID	Illegal Direct Mode	16
TM	Type Mismatch	18
OS	Out of String Space	1A
LS	String too Long	1C
ST	String Formula too complex	1E
CN	Can't Continue	20
UF	Undefined User Function	22
MO	Missing Operand	24

C359: Print "READY" - back to DIRECT COMMAND mode.
C366: Same as C359 except doesn't print "READY".
C3DD: Reset Program pointers - rejustify Link Pointers.
C3E0: Rejustify Link Pointers (see also D394). This can be used to RECOVER a BASIC program after <NEW> or RESET or even <CLOAD> if the original program hasn't been over-written.
C3FA: Searches for a particular Line Number in a BASIC program.
To search : DE = required line number
If found : BC = Start of Line
HL = Start of next line.
Carry Flag is set (=1).
C41A: <NEW>
C426: Part of <NEW> routine. Entering here resets HIMEM, STRING Space, and variable area pointers BUT doesn't destroy the first Link bytes thus not "erasing" the program.
C45A: Keyboard Input. Prints "?" then jumps to C53A.
C467: This routine scans the input Buffer and converts lower case to upper case (unless the text is enclosed in quotes) - converts BASIC reserved words to TOKENs.
C521: Resets HL = 014B and puts 3 zeros at end of program.
C53A: Inputs a string of characters from the keyboard until RETURN is pressed. Uses the buffer from 014C to 018C. On exit, HL=014B and B=number of characters entered.
C565: This is where BASIC tests for buffer overflow but DOESN'T do anything about it !!! (Perhaps JR Z C550 would be a suitable patch.)
C574: Compares HL with DE and sets the flags accordingly.
Z flag set when DE = HL.
C flag set when DE < HL.
C57A: Tests for "expected" characters such as commas, semi-colons, and left and right brackets.
To use : HL should point to the BASIC text to test.
CALL C57A
" , " Test for comma.
NOTE: If a match is NOT made, then an ?SN error results !

C585: Prints the character in the A register. This calls the Monitor Send Vector, E00C.
C5B4: This could be used to implement a <GET> or <INKEY$> function. Inputs one character to the A register. Calls the Monitor Receive Vector, E009, but will not return until a key is pressed !
C5C6: <LIST>
C62E: <FOR>
C66E: <STEP>
C689: Re-entry point to BASIC interpreter. HL should be pointing to the zero byte at the end of the line, or the colon in a multi- statement line.
C6AD: BASIC Interpreter starts here. HL should point to the byte before the actual text to be interpreted.
C6CD: Skips over blanks in the BASIC text by incrementing HL. The Carry Flag is set if the next character is a number.
C6DD: <RESTORE>
C6F8: This is the routine that's supposed to pause LISTings if ESC or RUN/STOP was pressed, however the byte at C701 is wrong. It should be 1BH instead of 13H.

C707: <STOP>
C709: <END>
C70E: Prints "BREAK" if CONTROL-C pressed in DIRECT MODE,
      else "BREAK IN ....line number" if in RUN MODE.
C735: <CONT>
C748: <NULL>
C75A: CLOAD* jumps here. (01AF) = FF and (01AC) = 01
C75F: CSAVE* jumps here. (01AF) = 01 and (01AC) = 01
C789: Write 4 bytes to tape. Used by CSAVE*
C792: Input 4 bytes from tape. Used by CLOAD*
C7B6: Turn off cassette motors - finish up.
C7BC: Tests for alpha-character pointed to by HL, and resets
      Carry Flag if found.
C7C4: As for C7D0 below, but first evaluates a BASIC expression
      pointed to by HL, then puts result into ACC.
      ?FC ERROR if negative.
C7D0: DE = INTeger value of floating point value in ACCumulator
C7E5: FC ERROR
C7EA: DE = Hex of numeric string pointed to by HL.
      On exit, HL points to first non digit character.
C80F: <CLEAR n,N> where n=string space, N=top of BASIC RAM.
C855: <RUN>
C858: Go address for >LOG BASIC programs. The program must
      have Line Number 0 and FILE Type must be <80H).
C861: <GOSUB>
C872: <GOTO>
C88B: UL ERROR
C890: <RETURN>
C899: RG ERROR
C8B5: <DATA>
C8B7: <REM>
C8CC: <LET>
C91F: Moves a variable (4 bytes) from ACC to (HL). On exit, HL
      points to the delimiter and DE points to the first of the
      4 bytes in the variable area.
C926: <ON>
C944: <IF>
C968: <PRINT>
C9B2: Do a CR-LF unless cursor is in the first column.
C9BF: Do a CR-LF.
C9C9: Delay by sending number of NULL characters in (0141).
C9F1: Here for <TAB(> or <SPC(> initially.
CA01: <TAB(>
CA04: <SPC(>
CA1B: "?REDO FROM START<CR><LF>" message string.
CA43: <INPUT>
CA72: <READ>
CAFF: "?EXTRA IGNORED<CR><LF>" message string.
CB10: Used by <READ>.
CB34: <NEXT>
CB7F: Calls CB93 to evaluate BASIC expression - check for TM
      Error.
CB82: Checks for NTF=0   ==> Numeric Operation.
CB83: Checks for NTF=1   ==> String Operation.
CB8A: TM ERROR

CB8F: Tests for left bracket and evaluates the expression.
Jumps to ?SN ERROR if left bracket not found.
CB93: Evaluates BASIC expression pointed to by HL.
CC0A: Used by several routines (esp.CB93) to test for +,.,-,",
<NOT>,<FN> and evaluates them if found. Also indexes into
Jump Table at C0C6 to get addresses for <SGN> thru <MID$>
CC14: MO ERROR
CC45: Evaluate expression then check for right bracket ")",
else ?SN ERROR.
CC5E: Loads ACC with variable - sets NTF. HL should point to
variable name.
CC6F: All commands <SGN> to <PEEK> and <LEN> to <MID$> pass
through here.
Numeric routines :<SGN> to <PEEK> jump to CC96.
"String" routines :<LEN> to <MID$> continue on to CC7D.
CC96: <USR> and others, jump from here.
CCA7: <OR> ACC = (SP) OR ACC.
CCA8: <AND> ACC = (SP) AND ACC.
CD2F: <NOT>
CD4F: <DIM>
CD54: This routine searches for a variable (name pointed to by
HL) and either (1) Returns its address in DE
or (2) Creates the variable if not found.
CDF8: Make ACC point to "READY".
CE4D: DD ERROR
CE5E: BS ERROR
CEE9: <FRE>
CF17: <POS>
CF1A: ACC = Floating point of value in A.
CF1F: <DEF>
CF42: <FN>
CF80: Checks to see if Sorcerer is in RUN mode or DIRECT mode.
CF89: ID ERROR
CF9F: <STR$>
CFC4: Loads (01A2)=String Length in A.
(01A4)=Start address of string in DE.
CFD3: Counts number of characters in a string.
On entry: HL should point to start of ASCII string.
On exit : BC = string length
HL = address of delimiter.
D00F: ST ERROR
D015: Prints a string using C585. Again, HL must point to start
of string.
D02E: First checks if there's still free string space.
?OS ERROR if none.
After a CALL D02E, DE points to start of sub-string.
A = sub-string length.
D093: String Space Garbage Collection routine.
D10F:
D146: Sets up registers before continuing to D14F.
On entry : Return address - pointer to 4 bytes of string
information are on the Stack.

```
D14F: Used by string commands to remove sub-strings.
      On entry : L=sub-string length.
                 BC=start address of sub-string within main
                 DE=start address of new string.

D159:
D17A:
D18B: <LEN>
D19A: <ASC>
D1AB: <CHR$>
D1BB: <LEFT$>
D1EB: <RIGHT$>
D1F5: <MID$>
D225: <VAL>
D240: Tests for a right bracket ")" after RIGHT$, LEFT$ or MID$
      On return, B=substring length.
D24A: <INP>
D256: <OUT>
D25C: <WAIT>
D27A: Used by <OUT> to evaluate - load the port value.
D28D: A = Hex of numeric string pointed to by HL.
      FC ERROR if > 255.
D2A1: Writes 2 bytes to tape. (Byte is in A).
D2A4: Writes 1 byte  to tape.
D2AD: Reads  1 byte from tape.
D2BD: Writes 1 byte  to tape.
D2C9: <CSAVE>
D2CE: <CSAVE*>
D304: Finish off CSAVE.
D310: Put program FILENAME (max.5 chars) into 16 byte tape
      output file header which starts at (IY+47H).
D336: If length of FILENAME < 5 then pad to right with blanks.
D341: <CLOAD>
D346: <CLOAD*>
D384: This is part of <CLOAD>. Loads (01B7) with the end of the
      CLOADed program, prints "READY" and re-creates the link
      pointers.
D394: Repair the link pointers in a BASIC program.
D39D: ACC = ACC  + 0.5
D3A0: ACC = (HL) + ACC    : ADDITION
D3A6: ACC = (HL) - ACC    : SUBTRACTION
D3AC: ACC = BCDE - ACC
D3AF: ACC = BCDE + ACC
D415: Sets  ACC = Zero  by making the floating point exponent
      (01C2)=0.
D451: OV ERROR
D462: BCDE = - BCDE
D4AB: <LOG>
D4E3: ACC = ACC  * LOG(2)
D4EA: ACC = (SP) * ACC
D4EC: ACC = BCDE * ACC
D54B: ACC = (SP) * ACC
D54D: ACC = BCDE / ACC
D5E0: ACC = ACC  * 10
D5F7: Checks if ACC = 0 ?  and sets Z flag if it is.
D606: <SGN>
```

D60E: ACC = Floating Point of BADE. This routine can be used
to return values to the USR(x) function.
B = Exponent
A = MSB including Sign Bit
D = NSB
E = LSB.
For -65536 < x < +65536 :

```
XOR  A          (A=0  for +ve)
LD   B, 98H     (A=FF for -ve)
LD  DE, NNnn
JP   D60E
```

D61C: <ABS>
D620: Part of ABS routine, the only difference is that it
doesn't test the ACC for zero.
D628: (SP) = ACC. Loads the floating point value from the ACC
to the stack To get it back, POP BC then POP DE.
D635: ACC = (HL). Loads floating point value (4 bytes) from HL
to ACC.
D638: ACC = BCDE.
D643: BCDE = ACC
D646: BCDE = (HL)
D64F: (HL) = ACC
D652: (HL) = (DE). Move 4 bytes from location pointed to by DE
to HL.
D654: (HL) = (DE). Move # bytes in B register.
D65D: ACC = - ACC
D672: Compares ACC with BCDE.
If ACC < BCDE then A = FFH (-1)
ACC = BCDE then A = 00
ACC > BCDE then A = 01
D6CA: <INT>
D6FE: <PEEK>
D705: <POKE>
D716: ACC = Floating Point of Numeric String pointed to by HL.
D7C6: Numeric String = ACC. (Opposite of D716).
D890: Compares ACC with 999,999.
D89F: Floating point 0.5 (00/00/00/80).
D8BA: <SQR>
D8C3: ACC = (SP) ^ ACC (POWER)
D908: <EXP>
D948: Table of floating point constants used by EXP - SQR.
D95D: FF/FF/7F/7F = 0.5
00/00/80/81 = -1
00/00/00/81 = +1
D999: <RND>
DA02: Tables of data used by RND.
DA0E: <COS> = SIN(x + PI/2)
DA14: <SIN> = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!
However, the actual expansion used is:
SIN(2PIx) = 2PIx - (2PIx)^3/3! + (2PIx)^5/5! -
(2PIx)^7/7! + (2PIx)^9/9!
Only the first 5 terms of the series are required for 6
digit precision. The above coefficients are located from
DA61 to DA74.

```
DA58:Table of floating point constants (4 bytes each) used by
     SIN - COS.
     DB/0F/49/81  =  1.5708 radians  =  90 degrees.
     00/00/00/7F  =  0.25
DA60: 05 = Number of terms required to calc. SIN - COS
DA61: BA/D7/1E/86  = +39.7107  = (2PI)^9/9! approx. allows for
     round-off.
     64/26/99/87  = -76.575   = (2PI)^7/7! exactly.
     58/34/23/87  = +81.6022  = (2PI)^5/5! approx.
     E0/5D/A5/86  = -41.3417  = (2PI)^3/3! exactly.
     DA/0F/49/83  = + 6.28319 = (2PI) = 360 degrees.
DA75: <TAN> = <SIN> / <COS>
DA8A: <ATN>
DAB1: Table of floating point constants used by ATN.
DAD6: [ EMPTY  ROM  from here till DFFA !! ]
DFFA: BASIC Warm Start, jumps to C06B.
DFFD: BASIC Cold Start, jumps to C000.
```

8.02 BASIC ROMPAC WORK AREA

ADDR	SIZE	DESCRIPTION
100:	3	Z80 jump instruction to Basic warm start: C3 6B C0 Used if work area and program are saved as a CP/M .COM file.
103:	3	Z80 jump instruction to USR() function routine. Default address is to the FC ERROR routine. The user can insert the address of his routine with: POKE 260,AD AND 255 : POKE 261,AD / 256
106:	3	OUT (nn),A instruction. Used by the OUT I,J command.
109:	14	Fast 4 byte subtract routine used by floating point divide for speed.
117:	35	Pseudo-random number data used as tables and counters by Basic's RND function.
13A:	4	Last psuedo-random number generated by RND, in floating point, for RND(0).
13E:	3	IN A,(nn) instruction. Used by INP() function.
141:	1	Number of ASCII nulls to print after a carriage return. Defaults to 1.
142:	1	Terminal line length. Defaults to 64.
143:	1	Column number of last PRINT with comma field, ie. the start of the last 14 character field.
144:	1	CTRL O output suppression flag. 0H = output. 0FFH = no output.
145:	2	Pointer to top of Basic stack.
147:	2	Current line number. 0FFFEH during initialization. 0FFFFH in direct mode.
149:	2	Pointer to start of Basic program text, ie. 01D5H.
14B:	67	Terminal input buffer. Starts with a comma. Also direct command line.
18E:	1	Current terminal column position.
18F:	1	0 when used by CD54 to locate - create variables. <> 0 when used by <DIM>.
190:	1	Number Type Flag. (See Note 3 above).

191: 1 Multi-Statement Line Flag.
= 0 if multi-statement line
<> 0 if only one statement on line.

192: 2 Pointer to highest RAM location, ie. HIMEM.

194: 18 Internal pointers used for string constant, variable and string space maintenance.

1A2: 2 Length of string that was just printed.

1A4: 2 Start address of string that was just printed.

1A6: 2 Pointer to top of free string space. Set by CLEAR n command.

1A8: 2 Internal pointer used in string space garbage collection.

1AA: 2 Current DATA line number.

1AC: 1 Used by <FOR> and <FN>.
= 64H when used by <FOR>
= 80H when used by <FN>

1AD: 1 Used by <RETURN>. Should = 0 for return. Also, last character entered from keyboard during input.

1AE: 1 Used by <INPUT> - <READ> to distinguish between them.
= 0 for <INPUT>.
<>0 for <READ>.

1AF: 1 Temporary storage for HL. Sometimes used to store current or last position reached in a line of BASIC before extra processing was required. Also used to store pointer to value of variable in variable area.

1B1: 2 Pointer to end of current line being processed by BASIC. This can be useful for passing names instead of numbers as arguments of the USR() function. Points to instruction in the Basic program about to be executed when CTRL-C is used to stop execution.

1B3: 2 Line number in program. Set by CTRL-C or STOP.

1B5: 2 Pointer to current statement in program to be executed next.

1B7: 2 Pointer to start of variable space at end of program.

1B9: 2 Pointer to start of array space.

1BB: 2 Pointer to end of RAM memory in use, ie. end of variable storage space.

1BD: 2 Pointer to current item in DATA list.

1BF: 4 Floating point numeric accumulator.

1C3: 17 Internal storage used for floating point printout and multiplication.

1D4: 1 Zero to signify end of imaginary first program line.

1D5: x Start of Basic program storage.

CHAPTER 9 --- MONITOR

9.01 MONITOR POKE ADDRESSES

		8K	16K	32K	48K
>SE S=xx	SCREEN SPEED	8143	16335	32719	-16433

S=0 FASTEST. POWER-ON DEFAULT.
S=10 SLOWER.

		8K	16K	32K	48K
>SE T=xx	BAUD RATE	8142	16334	32718	-16434

T=0 1200 BAUD FOR SERIAL PORT AND CASSETTE TAPE.
EXAMPLE: POKE 32718,64 :REM 1200 BAUD.

T=1 300 BAUD FOR SERIAL PORT AND CASSETTE TAPE.
EXAMPLE: POKE 32718,0 :REM 300 BAUD.

		8K	16K	32K	48K
>SE O=yy	OUTPUT PORT	8144-5	16336-7	32720-1	-16432-1

V - VIDEO DRIVER	0E01BH	POKE 32720,27	:POKE 32721,224
P - PARALLEL OUT	0E021H	POKE 32720,33	:POKE 32721,224
L - CENTRONICS	0E993H	POKE 32720,147	:POKE 32721,233
S - CASSETTE OUT	0E012H	POKE 32720,18	:POKE 32721,224
xxyy - USER DEFINED	xxyy	POKE 32720,yy	:POKE 32721,xx

		8K	16K	32K	48K
>SE I=xx	INPUT PORT	8146-7	16338-9	32722-3	-16430-29

K - KEYBOARD	0E018H	POKE 32722,24	:POKE 32723,224
P - PARALLEL IN	0E01EH	POKE 32722,30	:POKE 32723,224
S - CASSETTE IN	0E00FH	POKE 32722,15	:POKE 32723,224
xxyy - USER DEFINED	xxyy	POKE 32722,yy	:POKE 32723,xx

9.02 MONITOR WORKAREA

(IY+nn)	RAM	DESCRIPTION
+00	91	60 BYTE MONITOR COMMAND BUFFER. ASCII TEXT IS TERMINATED BY A CARRIAGE RETURN, 0DH.
+3C	CD	PORT 0FEH STATUS.
+3D	CE	BAUD RATE. 1200 BAUD= 040H. 300 BAUD= 00 HEX.
+3E	CF	SEND DELAY TIME FROM >SE S=nn COMMAND.
+3F	D0	CURRENT SEND ROUTINE ADDRESS FROM >SE O=xxyy.
+41	D2	CURRENT RECEIVE ROUTINE ADDRESS FROM >SE I=xxyy.
+43	D4	BATCH MODE STATUS. 0=NORMAL 1=BATCH MODE.
+44	D5	MONITOR OUTPUT PROMPT OF '>'.
+45	D6	BAUD RATE - MOTOR CONTROL 10H=MOTOR1 20H=MOTOR2
+46	D7	TAPE INPUT AND OUTPUT CRC CHECK DATA BYTE.
+47	D8	FIVE LETTER OUTPUT FILE NAME FROM >SA AND CSAVE
+4C	DD	OUTPUT FILE HEADER ID. USUALLY 55H.
+4D	DE	OUTPUT FILE TYPE FROM >SE F=nn COMMAND.
+4E	DF	2-BYTE LENGTH OF FILE IN BYTES.
+50	E1	2-BYTE PROGRAM LOAD ADDRESS. BASIC=01D5H.
+52	E3	2-BYTE PROGRAM >GO ADDRESS. >SE X=xxyy COMMAND.
+54	E5	3 RESERVED BYTES.
+57	E8	FIVE LETTER INPUT FILE NAME FROM >LO AND CLOAD.
+5C	ED	INPUT FILE HEADER ID.
+5D	EE	INPUT FILE TYPE.
+5E	EF	2-BYTE LENGTH OF INPUT FILE IN BYTES.
+60	F1	2-BYTE PROGRAM LOAD ADDRESS FROM TAPE.
+62	F3	2-BYTE PROGRAM >GO ADDRESS FROM TAPE.
+64	F5	3 RESERVED BYTES.
+67	F8	CHARACTER UNDER THE CURSOR.
+68	F9	2-BYTE SCREEN OFFSET FOR ROW CURSOR IS ON.
+6A	FB	2-BYTE COLUMN NUMBER CURSOR IS IN [0-63].
+6C	FD	LAST CHARACTER FROM KEYBOARD FOR REPEAT FUNCTION.
+6D	FE	2 BYTES OF RESERVED SPACE.

9.03 USEFUL MONITOR ROUTINES AND THEIR FUNCTIONS.

ADDR	DESCRIPTION
E000	MONITOR COLD START.
E003	MONITOR WARM START. JUMP TO HERE FROM YOUR PROGRAMS.
E006	RELOCATE MONITOR STACK TO ADDRESS PASSED IN REGISTER HL.
E009	RECEIVE: RETURN CHARACTER IN 'A' AND 'NZ' FLAG.
E00C	SEND: SEND CHARACTER IN 'A' TO CURRENT OUTPUT DEVICE.
E00F	SERIAL IN: READ CHARACTER FROM UART.
E012	SERIAL OUT: SEND CHARACTER TO UART.
E015	QUICK CHECK: RETURN 'NZ' IF 'CTRL-C', 'RUN/STOP' OR 'ESC'
E018	KEYBOARD: ROUTINE FOR RECEIVE IF >SE I=K. KEYBOARD SCAN
E01B	VIDEO: ROUTINE FOR SEND IF >SE O=V. VIDEO DISPLAY
E021	PARALLEL OUT: SEND CHARACTER TO PARALLEL PORT.
E993	CENTRONICS OUT: HANDSHAKE CHARACTER TO CENTRONICS PRINTER
E024	TURN CASSETTE MOTOR ON. REGISTER B CONTAINS 1 OR 2.
E027	TURN CASSETTE MOTOR OFF.
E02A	TAPE SAVE ROUTINE.
E02D	TAPE LOAD ROUTINE.
E13A	MONITOR INPUT ROUTINE FILLS 60 BYTE INPUT BUFFER.
E1A2	FIND MONITOR WORKAREA AND PUT BASE ADDRESS IN REGISTER IY
E1BA	MESSAGE: OUTPUT TEXT STRING THAT ENDS WITH A ZERO BYTE.
E1C9	ERROR: PRINT ERROR MESSAGE AND DIAGNOSTIC MESSAGE.
E1D4	OVER: PROCESS THE BATCH MODE OVER COMMAND.
E1E8	PRINT HEXADECIMAL NUMBER IN REGISTER 'DE' IN HEX.
E1ED	PRINT HEXADECIMAL NUMBER IN REGISTER 'A' IN HEX.
E205	PRINT CARRIAGE RETURN AND LINE FEED.
E22F	PARSE INPUT COMMAND STRING.
E23D	PUT ASCII HEX NUMBER IN REGISTER 'DE'. REVERSE OF E1E8H.
E2D2	SEND THE NUMBER OF BLANKS IN REGISTER 'B'.
E4D3	DUMP COMMAND PROCESSOR.
E538	ENTER COMMAND PROCESSOR.
E562	MOVE COMMAND PROCESSOR.
E597	GO COMMAND PROCESSOR.
E5A2	SET COMMAND PROCESSOR.
E638	SAVE COMMAND PROCESSOR.
E6B9	FILES COMMAND PROCESSOR.
E78A	LOAD COMMAND PROCESSOR.
E858	BATCH COMMAND PROCESSOR.
E85C	CREATE COMMAND PROCESSOR.
E884	LIST COMMAND PROCESSOR.
E8A1	TEST COMMAND PROCESSOR.
E98A	PP COMMAND PROCESSOR.
E9B1	CLEAR THE DISPLAY AND REWRITE THE GRAPHIC CHARACTER SET.
E9CC	MOVE CURSOR TO ROW AND COLUMN NUMBER IN MWA+68 AND MWA+6A
E9D6	FIND THE CURSOR ROW AND COLUMN NUMBERS.
EB10	REPLACE CHARACTER UNDER THE CURSOR.
EC1E	KEYBOARD INPUT TABLES TO DECODE THE ASCII VALUE TO RETURN
EDFE	CHARACTER SET FOR THE FIRST 64 GRAPHIC CHARACTERS.

9.04 RELOCATE THE MONITOR STACK.

```
>EN 0
0000: 21 yy xx C3 06 E0 /
>GO 0
```

THE MONITOR STACK ON POWER UP BUILDS FROM THE LAST MEMORY LOCATION DOWNWARD. IN A 32K MACHINE THIS IS AT 7FFFH. THE STACK CAN BE RELOCATED ELSEWHERE IN MEMORY BY PROVIDING A DIFFERENT LAST ADDRESS IN yy xx OF THE ABOVE CODE.

EXAMPLE: TO RELOCATE TO 6FFFH, USE

```
21 FF 6F C3 06 E0 /
```

==========

9.05 ERASE MEMORY BY FILLING IT WITH ZEROES.

```
BYE                 EXIT TO THE MONITOR.
>EN 0               BYTE 0 CONTAINS THE FILL CHARACTER.
0000: 0 /
>MO 0 7F00 1        ZERO MEMORY THROUGH ADDRESS 7F00 HEX.
```

==========

9.06 EXECUTE MONITOR COMMANDS FROM A BASIC PROGRAM

```
100 C$ = "LO 1 3800"+CHR$(13) : REM TYPICAL COMMAND
110 M = PEEK(-4096)+PEEK(-4095)*256-111
120 IF M>32767 THEN M=M-65536
130 FOR I=1 TO LEN(CM$)
140 POKE M+I,ASC(MID$(C$,I,1)): NEXT I :REM PUT IN BUFFER
150 POKE 260,138:POKE 261,231:X=USR(0) :REM DO COMMAND
```

CHAPTER 10 --- M. CODE INTERFACE

10.01 PROTECT MEMORY FOR MACHINE LANGUAGE ROUTINES

1. 0000 - 00FF HEX IS NEVER USED BY THE ROMPAC BASIC.
2. RELOCATE THE STACK. USE THE MEMORY ABOVE THE STACK.
3. CLEAR xx,yy CREATES A WINDOW ABOVE THE STRING SPACE.

==========

10.02 CALL A MACHINE LANGUAGE ROUTINE

```
100 POKE 260,ADDR AND 255 : POKE 261,ADDR/256
110 X=USR(0)
```

ADDR CONTAINS THE ADDRESS OF THE ROUTINE.
PLACE THE ADDRESS OF THE ROUTINE IN BYTES 260 AND 261.
INVOKE THE CALL TO THE ROUTINE WITH THE USR() FUNCTION.
A 'RET' INSTRUCTION IN THE ROUTINE WILL RETURN TO BASIC.

==========

10.03 FASTER USR() PARAMETER PASSING.

```
100 J=USR(0) : J=PEEK(X)
110 J=PEEK(USR(X))
```

LINE 110 IS EQUAVALENT TO LINE 100, BUT FASTER.

==========

10.04 POKE MACHINE HEX CODE INTO MEMORY.

```
100 DEF FNA(X) = (X AND 15) - 9 * (X > 64)
110 READ J : REM FIRST DATA ITEM IS # OF BYTE TO POKE
120 :
130 FOR I=0 TO J-1 : READ A$
140 K = FNA(ASC(A$)) * 16 + FNA(ASC(RIGHT$(A$,1)))
150 POKE OFFSET + I,K : NEXT I
160 :
200 DATA 6,11,00,00,C3,E8,E1  : REM EXAMPLE HEX CODE
```

THIS ROUTINE CONVERTS HEX CODE INTO DECIMAL TO POKE.
THE CODE LOADS BEGINNING AT THE ADDRESS OF 'OFFSET'.

==========

10.05 PASS ARGUMENT IN 'A' REGISTER TO MACHINE CODE

```
100 POKE 262,195 : POKE 264,ADDR/256
110 OUT ADDR AND 255, X
```

ADDR CONTAINS THE ADDRESS OF THE ROUTINE.
THE 'OUT' COMMAND INVOKES THE CALL TO THE ROUTINE.
ARGUMENT X IS PASSED TO THE ROUTINE IN THE ACCUMULATOR.

10.06 PASS TWO PARAMETERS USING THE OUT I,J INSTRUCTION.

CHANGE THE 'OUT I,J' INSTRUCTION TO A RST 0H COMMAND.
THE RST 0H IS A MACHINE LANGUAGE RESTART TO ADDRESS 0.
IT IS JUST LIKE A CALL INSTRUCTION, EXCEPT IT OCCUPIES
ONLY ONE BYTE OF MEMORY.

```
100 POKE 262,199 : REM 'OUT' IS NOT A 'RST 0H'
110 OUT I,J      : REM CALL YOUR ROUTINE AT ADDRESS 0.
```

YOUR MACHINE LANGUAGE ROUTINE MUST RESIDE AT ADDRESS 0.
THE I PARAMETER FROM THE OUT COMMAND IS PASSED IN
ADDRESS 0107H, AND THE J PARAMETER IS PASSED IN THE Z80
ACCUMULATOR. TO RETURN TO BASIC, YOU MUST POP THE RST
RETURN ADDRESS OFF OF THE STACK SO THAT THE RETURN
ADDRESS FOR THE 'OUT' COMMAND IS USED INSTEAD.

==========

10.07 PASS ROUTINE ADDRESS IN USR() FUNCTION CALL.

```
0000: CD D0 C7    ENTRY CALL 0C7D0H       ;GET USR() #
0003: D5                PUSH DE           ;CALL ADDRESS
0004: C9                RET               ;GOTO ROUTINE
```

```
100 POKE 260,0 : POKE 261,0 : REM USR() ENTRY ADDRESS
110 X = USR(AD) : REM JUMP TO MEMORY ADDRESS IN AD
```

THE USR() FUNCTION WILL ENTER THE ROUTINE AT ADDRESS 0,
AND THEN JUMP THE ADDRESS PASSED AS A PARAMETER IN THE
VARIABLE AD. YOUR CODE AT ADDRESS AD WOULD END WITH
A NORMAL 'RET' TO RETURN TO BASIC.

IF YOUR ROUTINE USES REGISTER HL, YOU SHOULD PUSH HL
UPON ENTRY, AND THEN POP HL JUST BEFORE RETURNING.

==========

10.08 MULTIPLE USR() ROUTINES SELECTED BY LETTER IN ().

```
100 POKE 260,0 : POKE 261,0 : REM ONLY ONE ENTRY ADDRESS.
110 :
120 X = USR(A)     : REM EXECUTE ROUTINE "A"
130 X = USR(B)     : REM EXECUTE ROUTINE "B"
```

THE ADDRESSES FOR THE ABOVE ROUTINES ARE FOUND IN A
TABLE. THE TABLE CONTAINS THE MATCH LETTER, SUCH AS "A",
FOLLOWED BY THE ADDRESS OF THE ROUTINE. A ZERO BYTE IN
THE MATCH LETTER POSITION WILL TERMINATE THE TABLE.

```
0000: CD A2 E1    ENTRY CALL 0E1A2H       ;GET IY
0003: 2A B1 01          LD   HL,(01B1H)   ;BASIC POINTER
0006: 3E B2             LD   A,0B2H       ;USR( TOKEN
0008: 2B          LP1   DEC  HL           ;POINTER BACK
0009: BE                CP   (HL)         ;FOUND USR( YET
000A: 20 FC             JR   NZ,LP1-$     ;KEEP LOOKING
000C: 23                INC  HL           ;SKIP OVER
000D: 23                INC  HL           ;POINT LETTER
000E: 7E                LD   A,(HL)       ;GET LETTER
000F: 21 27 00          LD   HL,TABLE     ;TABLE BASE
0012: BE          LP2   CP   (HL)         ;MATCH ??
0013: 28 0C             JR   Z,FOUND-$    ;JUMP IF MATCH
0015: 47                LD   B,A          ;SAVE LETTER
0016: AF                XOR  A            ;LOAD ZERO
0017: BE                CP   (HL)         ;TABLE END ??
0018: CA E5 C7          JP   Z,0C7E5H     ;?FC ERROR
001B: 78                LD   A,B          ;RESTORE LETTER
001C: 23                INC  HL           ;SKIP LETTER
001D: 23                INC  HL           ;SKIP yy ADDR
001E: 23                INC  HL           ;SKIP xx ADDR
001F: 18 F1             JR   LP2-$        ;TRY NEXT MATCH
0021: 23          FOUND INC  HL           ;SKIP LETTER
0022: 7E                LD   A,(HL)       ;GET yy ADDRESS
0023: 23                INC  HL           ;SKIP yy ADDR
0024: 66                LD   H,(HL)       ;GET xx ADDRESS
0025: 6F                LD   L,A          ;HL = xxyy ADDR
0026: E9                JP   (HL)         ;GOTO ROUTINE
0027: 41          TABLE DEFB 'A'          ;MATCH LETTER
0028: yy xx             DEFW A-LABEL      ;ROUTINE ADDR
002A: 42                DEFB 'B'          ;MATCH LETTER
002B: yy xx             DEFW B-LABEL      ;ROUTINE ADDR
002D: 00                DEFB 00           ;TABLE END FLAG
```

==========

10.09 UP-LOADER FOR MACHINE LANGUAGE ROUTINES

THIS SHORT ROUTINE IS USEFUL FOR MOVING MACHINE CODE TO ITS PROPER ORG ADDRESS AFTER BEING LOADED AT A DIFFERENT ADDRESS.

```
0100: 01 yy xx  ENTER  LD   BC,LENGTH   ;# OF BYTES TO MOVE
0103: 11 yy xx         LD   DE,TO       ;DESTINATION ADDRESS
0106: 21 yy xx         LD   HL,FROM     ;SOURCE ADDRESS
0109: ED B0            LDIR            ;BLOCK MOVE
010B: C3 yy xx         JP   START       ;PROGRAM START
```

THIS ROUTINE IS PARTICULARLY USEFUL IN CP/M ENVIRONMENTS WHERE THE CODE HAS BEEN MOVED DOWN TO 0110H WITH THE MONITOR AND THEN SAVED ON DISK. WHEN THE PROGRAM IS LOADED FROM DISK, CP/M EXECUTES THIS UP-LOADER WHICH LOADED JUST AHEAD OF THE PROGRAM. THE PROGRAM IS MOVED TO ITS PROPER ADDRESS AND EXECUTED.

CHAPTER 11 --- M. CODE ROUTINES

11.01 ROW - COLUMN ROTATION OF A CHARACTER CELL

ENTER WITH AN ASCII CHARACTER IN REGISTER A.

THE BIT PATTERN FOR THE CHARACTER IS CONVERTED FROM ITS ROW BY ROW FORMAT INTO COLUMN BY COLUMN DOT DATA THAT CAN BE USED TO FIRE EIGHT HAMMERS OF A GRAPHICS PRINTER.

TO ACTIVATE, SET THE OUTPUT VECTOR TO THIS DRIVER. >SE O=6500

```
                        ORG   6500H
6500: C5                PUSH  BC
6501: D5                PUCH  DE
6502: E5                PUSH  HL
6503: 6F                LD    L,A
6504: 26 00             LD    H,0           ;HL = ASCII CHAR
6506: 29                ADD   HL,HL         ;*2
6507: 29                ADD   HL,HL         ;*4
6508: 29                ADD   HL,HL         ;*8
6509: 11 00 F8          LD    DE,0F800H     ;ASCII TABLE BASE
650C: 19                ADD   HL,DE         ;TABLE POINTER
650D: 1E 08             LD    E,8           ;COL LOOP COUNTER
650F: 0E 80             LD    C,80H         ;AND MASK BIT 7
;
6511: 16 00     LOOP1   LD    D,0           ;CLEAR D
6513: E5                PUSH  HL            ;SAVE HL NEXT LOOP
6514: 06 08             LD    B,8           ;ROW LOOP COUNTER
;
6516: 7E        LOOP2   LD    A,(HL)        ;GET PATTERN
6517: A1                AND   C             ;MASK BIT OFF
6518: B2                OR    D             ;COMBINE
6519: 07                RLCA                ;ROTATE A LEFT
651A: 57                LD    D,A           ;RESAVE IN D
651B: 23                INC   HL            ;NEXT ROW PATTERN
651C: 10 F8             DJNZ  LOOP2-$       ;GET 8 ROW BITS
;
;  COMPENSATE FOR MASK SHIFT, UNEVEN SHIFTING
;
651E: 43                LD    B,E
651F: 0F        LOOP3   RRCA
6520: 10 FD             DJNZ  LOOP3-$       ;ROTATE BACK
;
6522: D3 FF             OUT   (255),A       ;SEND COL TO PRINTER
6524: CB 09             RRC   C             ;ROTATE MASK
6526: E1                POP   HL            ;ADDRESS BASE
6527: 1D                DEC   E
6528: 20 E7             JR    NZ,LOOP1-$    ;DO 8 COLUMNS
;
652A: E1                POP   HL
652B: D1                POP   DE
652C: C1                POP   BC
652D: C9                RET                 ;BACK TO MAIN PROG
;
```

11.02 IMAGE DRIVER TO CONVERT TO >SA FILES.

This is a simple, yet very useful, driver that converts any video output to a memory image. Two frequent uses of this driver are: convert a Basic program into its ASCII file, and convert a disassembler listing into a word processor file.

```
0000: E5          ENTRY PUSH HL
0001: 2A 0D 00          LD   HL,(POINT)    ;GET POINTER
0004: 77                LD   (HL),A        ;CREATE IMAGE
0005: 23                INC  HL            ;NEXT MEM CELL
0006: 22 0D 00          LD   (POINT),HL    ;PUT POINTER
0009: E1                POP  HL
000A: C3 1B E0          JP   0E01BH        ;TO VIDEO TOO
000D: yy xx       POINT DEFW xxyy          ;START ADDRESS
```

EXAMPLE: CREATE A WORD PROCESSOR FILE FROM DISASSEMBLER.

1. INSERT WORD PROCESSOR ROMPAC.

2. USE THE MONITOR TO LOAD A MACHINE LANGUAGE DISASSEMBLER SUCH AS: >LO DIS32

3. LOAD THE ABOVE DRIVER.

4. START ADDRESS IS 80FH: 000D: 0F 08 /

5. CHANGE OUTPUT VECTOR: >SE O=0000

6. EXECUTE DISASSEMBLER. >GO 6A00

7. DISASSEMBLE ROUTINE OF INTEREST.

8. EXIT DISASSEMBLER BACK TO MONITOR.

9. RESTORE VIDEO VECTOR. >SE O=V

10. FIND FINAL ADDRESS IN (POINT). >DU D E
    ONE MIGHT SAVE THE FILE AT THIS POINT WITH
    >SA NAME 080F xxyy

11. ENTER 03 INTO THIS FINAL ADDRESS. THIS PROVIDES THE END-OF-FILE MARKER FOR THE WORD PROCESSOR.

12. EXECUTE WORD PROCESSOR. >PP
    YOUR FILE SHOULD BE PRESENT. IT MAY BE DOUBLE SPACED SINCE THE WORD PROCESSOR CONVERTS BOTH <CR> AND <LF> INTO <CR>s.

13. USE THIS MACRO TO REMOVE DOUBLE SPACING.

```
D1
F1

COMMAND> A          (SAVE MACRO)
COMMAND> B          (START ON LINE TO DELETE)
COMMAND> A200       (EXECUTE MACRO)
```

11.03 PINE WOOD DERBY CONTROLLER

The following machine language and basic programs are included as illustrations of how to use the Sorcerer to monitor external events. In this case, photo detectors were mounted in each of three racing lanes for a Cub Scout Pine Wood Derby. As the hand made cars crossed the finish line, the photo detector light source would be interrupted. The Sorcerer would observe the changed state of the photo detectors and stop that car's timer. After all three cars had crossed the finish line, the three timers would be displayed in large block lettering for the anxious audience to read. The resolution of the timing was around 1 milisecond which kept parents and their sons from arguing over whose car was the fastest.

```
;
;
;       PINE WOOD DERBY CONTROLLER
;
                                ORG   4000H
4000: 18 05           ENTRY     JR    SKIP-$
;
4002: 00              CAR1      DEFB  0
4003: 00              CAR2      DEFB  0
4004: 00              CAR3      DEFB  0
4005: 00              COUNT     DEFW  0
;
                      BUFF      EQU   0
;
4007: DB FF           SKIP      IN    A,(0FFH)      ;WAIT FOR START
4009: CB 47                     BIT   0,A           ;GATE TO OPEN
400B: 20 FA                     JR    NZ,L4007-$
;
; THE GATE IS OPEN.  THE CARS ARE ON THEIR WAY DOWN THE TRACK
;
400D: 3E 01                     LD    A,001H
400F: 32 02 40                  LD    (CAR1),A
4012: 32 03 40                  LD    (CAR2),A
4015: 32 04 40                  LD    (CAR3),A      ;START ALL TIMERS
;
4018: DB FF           FINISH    IN    A,(0FFH)      ;WATCH FINISH LINE
401A: CB 4F                     BIT   1,A           ;IS CAR1 ACROSS?
401C: 20 07                     JR    NZ,L4025-$    ;JUMP IF NOT
401E: F5                        PUSH  AF
401F: 3E 00                     LD    A,000H
4021: 32 02 40                  LD    (CAR1),A      ;STOP TIMER #1
4024: F1                        POP   AF
;
4025: CB 57           L4025     BIT   2,A           ;IS CAR2 ACROSS?
4027: 20 07                     JR    NZ,L4030-$    ;JUMP IF NOT
4029: F5                        PUSH  AF
402A: 3E 00                     LD    A,000H
402C: 32 03 40                  LD    (CAR2),A      ;STOP TIMER #2
402F: F1                        POP   AF
```

```
;
4030: CB 5F         L4030   BIT  3,A          ;IS CAR3 ACROSS?
4032: 20 07                 JR   NZ,L403B-$   ;JUMP IF NOT
4034: F5                    PUSH AF
4035: 3E 00                 LD   A,000H
4037: 32 04 40              LD   (CAR3),A     ;STOP TIMER #3
403A: F1                    POP  AF
;
; UPDATE TIMER CLOCK, WHICH IS A SIMPLE INCREMENTING COUNTER.
;
403B: 2A 05 40      L403B   LD   HL,(COUNT)   ;GET TIMER.
403E: 7D                    LD   A,L          ;HL = 4 DECIMAL
403F: 3C                    INC  A            ; DIGITS RATHER
4040: 27                    DAA               ; A BINARY #.
4041: 6F                    LD   L,A
4042: 3E 00                 LD   A,000H
4044: 8C                    ADC  A,H          ;HL = HL + 1
4045: 27                    DAA
4046: 67                    LD   H,A
4047: 22 05 40              LD   (COUNT),HL   ;REPLACE TIMER
;
; UPDATE EACH CAR'S DISPLAY ON CRT WITH NEW COUNT VALUE UNTIL
; CAR'S FINISH FLAG = 0.  THE RACE IS OVER WHEN ALL FLAGS = 0.
;
404A: 3A 02 40              LD   A,(CAR1)     ;IS CAR1 ACROSS?
404D: B7                    OR   A
404E: 28 06                 JR   Z,L4056-$
4050: 11 04 00              LD   DE,BUFF+4H   ;CAR'S BUFFER
4053: CD 7D 40              CALL MOVE         ;LOAD NEW TIME
;
4056: 3A 03 40      L4056   LD   A,(CAR2)     ;IS CAR2 ACROSS?
4059: B7                    OR   A
405A: 28 06                 JR   Z,L4062-$
405C: 11 0C 00              LD   DE,BUFF+0CH  ;CAR'S BUFFER
405F: CD 7D 40              CALL MOVE         ;LOAD NEW TIME
;
4062: 3A 04 40      L4062   LD   A,(CAR3)     ;IS CAR3 ACROSS?
4065: B7                    OR   A
4066: 28 06                 JR   Z,L406E-$
4068: 11 14 00              LD   DE,BUFF+14H  ;CAR'S BUFFER
406B: CD 7D 40              CALL MOVE         ;LOAD NEW TIME
;
406E: CD A9 40      L406E   CALL MESSAGE      ;NEW CRT DISPLAY
;
; NOW CHECK TO SEE IF ALL CARS ARE ACROSS
;
4071: 21 02 40              LD   HL,CAR1      ;CHECK FIRST FLAG
4074: 7E                    LD   A,(HL)
4075: 23                    INC  HL
4076: B6                    OR   (HL)         ;CAR2 FINISH FLAG
4077: 23                    INC  HL
4078: B6                    OR   (HL)         ;CAR3 FINISH FLAG
4079: C2 18 40              JP   NZ,FINISH    ;DONE IF ALL = 0
407C: C9                    RET               ;BACK TO BASIC
```

```
;
; THIS ROUTINE LOADS EACH CAR'S BUFFER WITH THE ASCII
; CHARACTERS THAT MAKE UP THE FOUR DECIMAL DIGIT COUNT IN HL.
;
; ENTER WITH A CAR'S UNIQUE BUFFER ADDRESS IN DE.
;
407D: 2A 05 40      MOVE    LD   HL,(COUNT)  ;GET TIMER VALUE
4080: EB                    EX   DE,HL       ;HL = BUFFER ADDR
4081: 7A                    LD   A,D         ;DE = 4 DIGITS
4082: CB 3F                 SRL  A
4084: CB 3F                 SRL  A
4086: CB 3F                 SRL  A
4088: CB 3F                 SRL  A
408A: F6 30                 OR   030H        ;FIRST ASCII DIGIT
408C: 77                    LD   (HL),A      ;PUT IT IN BUFFER
408D: 23                    INC  HL
408E: 7A                    LD   A,D
408F: E6 0F                 AND  00FH
4091: F6 30                 OR   030H        ;2ND ASCII DIGIT
4093: 77                    LD   (HL),A      ;PUT IT IN BUFFER
4094: 23                    INC  HL
4095: 7B                    LD   A,E
4096: CB 3F                 SRL  A
4098: CB 3F                 SRL  A
409A: CB 3F                 SRL  A
409C: CB 3F                 SRL  A
409E: F6 30                 OR   030H        ;3RD ASCII DIGIT
40A0: 77                    LD   (HL),A      ;PUT IT IN BUFFER
40A1: 23                    INC  HL
40A2: 7B                    LD   A,E
40A3: E6 0F                 AND  00FH
40A5: F6 30                 OR   030H        ;4TH ASCII DIGIT
40A7: 77                    LD   (HL),A      ;PUT IT IN BUFFER
40A8: C9                    RET
;
; THIS ROUTINE DISPLAYS A 24 CHARACTER BUFFER IN THE CRT IN
; THREE ROWS OF EIGHT CHARACTERS EACH.  THE CHARACTERS ARE
; DISPLAYED IN LARGE BLOCK SIZE WHERE EACH DOT OCCUPIES A FULL
; CHARACTER CELL.  THE START ADDRESS OF THE BUFFER IS BUFF.
;
40A9: FD E5         MESSAGE PUSH IY
40AB: FD 21 00 F1           LD   IY,0F100H   ;TOP OF SCREEN
40AF: 21 00 00              LD   HL,BUFF
40B2: 0E 03                 LD   C,003H      ;C = # OF ROWS
40B4: 06 08         NEXTROW LD   B,008H      ;B = # OF CHARS
40B6: 7E            NEXTCOL LD   A,(HL)      ;GET ASCII CHAR
40B7: C5                    PUSH BC
40B8: E5                    PUSH HL
40B9: FD E5                 PUSH IY
40BB: CD D7 40              CALL DISPLAY     ;SHOW IN BIG SIZE
40BE: FD E1                 POP  IY
40C0: E1                    POP  HL
40C1: C1                    POP  BC
40C2: 23                    INC  HL
40C3: 11 08 00              LD   DE,8        ;MOVE SCRN POINTER
```

```
40C6: FD 19                    ADD  IY,DE
40C8: 10 EC                    DJNZ NEXTCOL-$     ;NEXT CHARACTER
40CA: 0D                       DEC  C             ;THIS ROW DONE
40CB: 28 07                    JR   Z,L40D4-$     ;ANY MORE TO DO?
40CD: 11 40 02                 LD   DE,0240H      ;MOVE SCRN POINTER
40D0: FD 19                    ADD  IY,DE         ; TO NEXT ROW.
40D2: 18 E0                    JR   NEXTROW-$
40D4: FD E1          L40D4     POP  IY            ;SCREEN COMPLETE
40D6: C9                       RET
;
; THIS ROUTINE CREATES A LARGE SIZE CHARACTER FROM THE BIT
; PATTERN FOR THE ASCII CHARACTER.  THE CHARACTER IS LOCATED
; BY HAVING IY POINT TO THE TOP LEFT CORNER OF THE CHARACTER.
;
; ENTER WITH ASCII CHARACTER IN A, AND SCREEN LOCATION IN IY.
;
40D7: 0E 08          DISPLAY   LD   C,008H        ;ROW COUNTER
40D9: 6F                       LD   L,A
40DA: 26 00                    LD   H,000H
40DC: 29                       ADD  HL,HL
40DD: 29                       ADD  HL,HL
40DE: 29                       ADD  HL,HL         ;OFFSET = ASCII*8
40DF: 11 00 F8                 LD   DE,0F800H     ;START OF BIT MAP
40E2: 19                       ADD  HL,DE         ;HL = PATTERN ADDR
40E3: 7F             CHAR      LD   A,(HL)        ;A = FIRST ROW
40E4: 06 08                    LD   B,008H        ;CELL WIDTH
40E6: 07             COL       RLCA               ;DETERMINE WHITE
40E7: 38 06                    JR   C,WHITE-$     ; OR BLACK DOT.
40E9: FD 36 00 20              LD   (IY+000H),020H    ;BLACK
40ED: 18 04                    JR   NEXT-$
40EF: FD 36 00 C0    WHITE     LD   (IY+000H),0C0H    ;WHITE
40F3: FD 23          NEXT      INC  IY
40F5: 10 EF                    DJNZ COL-$         ;LOOP CELL WIDTH
40F7: 0D                       DEC  C             ;COUNT THIS ROW
40F8: C8                       RET  Z             :EXIT AFTER 8TH
40F9: 11 38 00                 LD   DE,038H       ;POINT START NEXT
40FC: FD 19                    ADD  IY,DE         ; ROW ON SCREEN.
40FE: 23                       INC  HL            ;NEXT ROW PATTERN
40FF: 18 E2                    JR   CHAR-$
```

11.04 PINE WOOD DERBY'S BASIC PROGRAM

```
100 REM              PINE WOOD DERBY
110 :
120 REM  CREATE SOLID WHITE GRAPHIC CHARACTER IN ASCII #192
130 :
140 FOR I=1 TO 8:POKE -513+I,255:NEXT I
150 :
160 REM  ESTABLISH USR() ADDRESS FOR 4000H.
170 :
180 POKE 261,64
190 :
200 REM  CLEAR SCREEN, PUT MESSAGE IN 24 BYTE BUFFER
210 :
220 PRINT CHR$(12):A$="READY   GET SET GO . . ."
230 FOR I=1TO24:POKE I-1,ASC(MID$(A$,I,1)):NEXT I
240 :
250 REM  NOW SEND BUFFER TO CRT IN LARGE LETTERS
260 :
270 POKE 260,169:X=USR(X)
280 :
290 REM  SCREEN LOOKS LIKE THIS WITH 3 ROWS OF 8 CHARACTERS:
300 REM
310 REM  READY
320 REM  GET SET
330 REM  GO . . .
340 REM
350 :
360 REM  NOW WAIT FOR THE GATE TO OPEN AND START TIMERS
370 REM  CALL MACHINE CODE ROUTINE AT 4000H.
380 :
390 POKE 260,0:X=USR(X)
400 :
410 REM  RACE IS NOW OVER.  SORT TIMES FOR 1ST, 2ND, 3RD PLACE
420 REM  TIMES FOUND IN BUFFER AS ASCII DIGITS.
430 REM  CONVERT TO A DECIMAL VALUE.
440 :
450 K=4 : GOSUB 900 : A=X : REM  K = BUFFER OFFSET
460 K=12: GOSUB 900 : B=X
470 K=20: GOSUB 900 : C=X
480 F=49: G=F : H=F : REM  F=G=H= ASCII ZERO
490 :
500 REM  SORT WINNER, 2ND AND 3RD PLACE
510 :
520 IF A>B THEN F=F+1
530 IF A>C THEN F=F+1
540 IF B>A THEN G=G+1
550 IF B>C THEN G=G+1
560 IF C>A THEN H=H+1
570 IF C>B THEN H=H+1
```

```
580 :
590 REM  PUT 1ST, 2ND, 3RD PLACEMENT IN BUFFER
600 :
610 POKE 0,65 : POKE 1,45 : POKE 3,32
620 POKE 8,66 : POKE 9,45 : POKE11,32
630 POKE16,67 : POKE17,45 : POKE19,32
640 POKE 2,F  : POKE10,G  : POKE18,H
650 :
660 REM  NOW DISPLAY WINNERS IN LARGE LETTERS
670 :
680 POKE 260,169 : X=USR(X)
690 PRINT CHR$(17);"  LANE           PLACE "; : REM HEADINGS
700 :
710 REM  SCREEN LOOKS LIKE THIS:  LANE - PLACE - TIMER
720 REM
730 REM  A-2 8034
740 REM  B-1 7892
750 REM  C-3 8047
760 REM
770 :
780 REM  WAIT FOR OPERATOR TO START NEXT RACE.
790 REM  USER HITS RETURN TO CONTINUE.
800 :
810 INPUT "          press RETURN to race again ";A$:GOTO200
900 :
910 REM  CONVERT 4 ASCII CHARACTERS FROM BUFFER INTO NUMBER
920 :
930 X=0:FOR I=K TO K+4:J=PEEK(I)-30:X=X*10+J:NEXT I:RETURN
```

11.05 IDEAL MACHINE LANGUAGE PROGRAM START

```
                        ; PROGRAM NAME
                        ;
                        ; AUTHOR'S COPYRIGHT NOTICE
                        ; PROGRAM DATE AND VERSION NUMBER
                        ;
                        ; PROGRAM DESCRIPTION
                        ;
                        MONITOR EQU  0E003H    ;EQUATE LABELS
                        ;
                                ORG  0500H     ;ORG VARIABLES
                        COUNT   DEFB 0
                        ;
                                ORG  0100H     ;ORG PROGRAM
0100: C3 yy xx          ENTER   JP   COLDST    ;JUMP COLD START
0103: C3 yy xx          WARM    JP   WARMST    ;JUMP WARM START
0106: C3 yy xx          SEND    JP   VIDEO     ;OUTPUT DRIVER
0109: C3 yy xx          RECEIVE JP   KEYBRD    ;INPUT DRIVER
010C: C3 yy xx          PRINTER JP   OUTPUT    ;PRINTER DRIVER
010F: C3 yy xx          SOUND   JP   NOISE     ;SOUND ROUTINES
                        ;
                        ;       ETC
                        ;
                        ; HAVING THE PROGRAM START AT 100H MAKES
                        ; IT EASY TO PLACE PROGRAM ON CP/M DISK.
                        ;
                        ; START PROGRAM WITH VECTOR JUMPS TO THE
                        ; VARIOUS MAJOR ROUTINES USED BY THE
                        ; PROGRAM.  THUS, IF A USER MUST MODIFY
                        ; THE PROGRAM TO SUIT HIS SYSTEM, HE
                        ; ONLY HAS TO CHANGE THE JUMP ADDRESS TO
                        ; ACCESS HIS NEW SUBSTITUTE ROUTINE.
                        ;
0112: C0                BAUD    DEFB 0C0H      ;SYSTEM PARAMETERS
0113: FE                PORT    DEFB 0FEH      ;STORED HERE
                        ;
                        ; NOW PLACE A TABLE OF SYSTEM PARAMETERS
                        ; USED SUCH AS BAUD RATE.  A USER WHO
                        ; USES A DIFFERENT PARAMETER NOW ONLY
                        ; HAS TO STORE THE DIFFERENT VALUE HERE.
                        ;
0114:                   MAIN    ???? ??        ;BEGIN MAIN PROGRAM

                                ETC.
```

CHAPTER 12 --- I/O DRIVERS

12.01 RS232 OUTPUT DRIVER ROUTINE

```
0000:  F5         ENTRY PUSH AF            ;SAVE CHARACTER
0001:  F5               PUSH AF
0002:  FD 7E 3D         LD   A,(IY+03DH)   ;UART CONTROL
0005:  F6 80            OR   80H           ;SET BIT 7
0007:  D3 FE            OUT  (0FEH),A      ;TURN RS232 ON
0009:  F1               POP  AF            ;GET CHARACTER
000A:  CD 12 E0         CALL 0E012H        ;RS232 CHAR OUT
000D:  CD 1B E0         CALL 0E01BH        ;SEND TO VIDEO
0010:  F1               POP  AF            ;GET CHARACTER
0011:  C9               RET                ;RETURN
```

TO DIRECT ALL OUTPUT TO BOTH THE RS232 PORT AND TO THE VIDEO, USE THE MONITOR COMMAND:

```
>SE O=xxyy
```

WHERE xxyy IS THE ENTRY ADDRESS FOR WHERE EVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE >SE O=0000.
TO RESTORE THE OUTPUT TO JUST THE VIDEO DRIVER USE:

```
>SE O=V
```

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.

==========

12.02 RS232 DRIVER WITH PERFERATION SKIP.

THE ROUTINE WILL COUNT THE NUMBER OF LINES PRINTED AND AUTOMATICALLY ISSUE A FORM FEED AFTER 54 LINES OF PRINT.

```
0000:  F5         ENTRY PUSH AF            ;SAVE CHARACTER
0001:  F5               PUSH AF
0002:  FD 7E 3D         LD   A,(IY+03DH)   ;UART CONTROL
0005:  F6 80            OR   80H           ;SET BIT 7
0007:  D3 FE            OUT  (0FEH),A      ;TURN RS232 ON
0009:  F1               POP  AF            ;GET CHARACTER
000A:  CD 12 E0         CALL 0E012H        ;RS232 CHAR OUT
000D:  CD 1B E0         CALL 0E01BH        ;SEND TO VIDEO
0010:  F1               POP  AF            ;GET CHARACTER
0011:  FE 0A            CP   0AH           ;TEST FOR LF
0013:  C0               RET  NZ            ;RETURN IF NOT
0014:  F5               PUSH AF            ;SAVE CHARACTER
0015:  3A FF FF         LD   A,(0FFFFH)    ;LINE COUNTER
0018:  3C               INC  A             ;# = # + 1
0019:  FE 36            CP   54            ;54 LINES YET?
001B:  20 06            JR   NZ,EXIT-$     ;EXIT IF NOT 54
001D:  3E 0C            LD   A,0CH         ;GET FORM FEED
001F:  CD 12 E0         CALL 0E012H        ;RS232 CHAR OUT
0022:  AF               XOR  A             ;COUNTER = 0
0023:  32 FF FF  EXIT   LD   (0FFFFH),A    ;RESAVE COUNTER
0026:  F1               POP  AF            ;GET CHARACTER
0027:  C9               RET                ;RETURN
```

SYNCHRONIZE THE LINE COUNTER WITH PAPER TOP-OF-FORM BY ENTERING A 0 INTO BYTE FFFF HEX.

```
EXAMPLE:   >EN FFFF            FROM THE MONITOR.
           FFFF: 0 /
      OR
           POKE -1,0           FROM BASIC.
```

TO LIST BASIC PROGRAM USE: POKE -1,0:LIST

TO DIRECT ALL OUTPUT TO BOTH THE RS232 PORT AND TO THE VIDEO, USE THE MONITOR COMMAND:

```
>SE O=xxyy
```

WHERE xxyy IS THE ENTRY ADDRESS FOR WHERE EVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE >SE O=0000.

TO RESTORE THE OUTPUT TO JUST THE VIDEO DRIVER USE:

```
>SE O=V
```

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.

==========

12.03 RS232 INPUT DRIVER ROUTINE.

```
0000:  FD 7E 3D  ENTRY LD   A,(IY+03DH)  ;UART CONTROL
0003:  F6 80           OR   80H          ;SET BIT 7
0005:  D3 FE           OUT  (0FEH),A     ;TURN ON RS232
0007:  C3 0F E0        JP   0E00FH       ;GET RS232 INPUT
```

TO RECEIVE ALL INPUT FROM THE RS232 PORT, USE THE MONITOR COMMAND:

```
>SE I=xxyy
```

WHERE xxyy IS THE ENTRY ADDRESS FOR WHERE EVER THE ROUTINE IS LOCATED IN MEMORY. FOR THE ABOVE ADDRESSING THE COMMAND WOULD BE >SE I=0000.

ONE LOOSES KEYBOARD CONTROL OF THE SORCERER WHEN THE INPUT IS SET TO THE ABOVE RS232 INPUT DRIVER.

PERHAPS A BETTER USE OF THE ROUTINE WOULD BE TO CALL IT FROM AN APPLICATION PROGRAM WITH THE INP() FUNCTION.

```
EXAMPLE:  100 POKE 318,195:POKE 320,0
          110 X=INP(0) : A$=CHR$(X)
```

THE INP() EXAMPLE USES AN ENTRY ADDRESS OF 0000 HEX. A$ IS THE CHARACTER RECEIVED FROM THE RS232 ROUTINE.

THE ABOVE RS232 DRIVER IS FULLY RELOCATABLE.

12.04 DUMB TERMINAL ROUTINE

```
;
;  GET CHARACTERS FROM KEYBOARD AND SEND TO RS232.
;  CHARACTERS RECEIVED FROM RS232 SENT TO VIDEO SCREEN.
;  RETURN TO CALLING PROGRAM IF 'RUN/STOP' KEY IS PRESSED.
;
                              ORG  2470H
;
2470: 3E C0          DUMB     LD   A,0C0H       ;BAUD - RS232 ON
2472: D3 FE                   OUT  (0FEH),A     ;TURN RS232 ON
2474: DB FE          LOOP     IN   A,(0FEH)     ;LOOK FOR 'R/S'
2476: CB 47                   BIT  0,A
2478: C8                      RET  Z            ;ABORT IF 'R/S'
2479: CD 1C EB                CALL 0EB1CH       ;SCAN KEYBOARD
247C: 28 0A                   JR   Z,SCAN-$     ;SKIP IF NO INPUT
247E: F5                      PUSH AF
247F: DB FD          UART     IN   A,(0FDH)     ;WAIT UART DONE
2481: CB 47                   BIT  0,A
2483: 28 FA                   JR   Z,UART-$
2485: F1                      POP  AF
2486: D3 FC                   OUT  (0FCH),A     ;SEND CHARACTER
2488: DB FD          SCAN     IN   A,(0FDH)     ;CHAR RECEIVED
248A: CB 4F                   BIT  1,A
248C: 28 E6                   JR   Z,LOOP -$    ;LOOP IF NOTHING
248E: DB FC                   IN   A,(0FCH)     ;GET INCOMING CHAR
2490: CD 1B E0                CALL 0E01BH       ;SEND TO VIDEO
2493: 18 DF                   JR   LOOP-$       ;LOOP
```

==========

12.05 VARIABLE LINE LENGTHS FOR PRINTERS.

```
100 POKE 322,J
```

J = LINE LENGTH FROM 0 TO 255.
BASIC WILL ISSUE A <CR> WHEN THE LINE LENGTH IS EXCEEDED.

```
EXAMPLE:  100 POKE 322,132 : REM 132 CHARACTER LINE
```

==========

12.06 CENTRONICS SCREEN PRINT ROUTINE

```
100 POKE 262,195 : POKE 264,233
110 AD = -3968
120 FOR R = 0 TO 29 : FOR C = 0 TO 63
130 OUT 147,PEEK(AD) : AD = AD + 1
140 NEXT C
150 OUT 147,13 : REM SEND <CR> AT END-OF-LINE
160 NEXT R
```

THE POKE STATEMENTS ON LINE 100 INITIALIZE THE ROUTINE.
EACH 'OUT 147,##' SENDS ONE ASCII CHARACTER TO THE
MONITOR'S CENTRONICS DRIVER ROUTINE.

12.07 CENTRONICS PRINTER DRIVER

```
100 OUT 255,J OR 128
110 OUT 255,J
120 OUT 255,J OR 128
```

THESE THREE OUT STATEMENTS SEND THE ASCII CHARACTER VALUE IN J, AND STROBE THE HANDSHAKE ON BIT 7. THIS METHOD OF SENDING CHARACTERS TO A CENTRONICS PRINTER IS MUCH SLOWER THAN THE METHOD OF 12.06 WHERE THE 'OUT' COMMAND PASSES THE ASCII DIRECTLY TO THE MONITOR'S CENTRONICS ROUTINE.

==========

12.08 ACCESS CENTRONICS PRINTER DRIVER FROM BASIC.

```
100 RAMSIZE = PEEK(-4095) * 256 + PEEK(-4096)
110 IF RAMSIZE > 32767 THEN RAMSIZE = RAMSIZE - 65536
120 :
130 POKE RAMSIZE - 47,147 : REM TURN PRINTER ON
140 :
200 POKE RAMSIZE - 47,240 : REM TURN PRINTER OFF
```

THE ABOVE IS EQUIVALENT TO >SE O=L AND >SE O=V.

==========

12.09 PROGRAMMING THE UART FOR PARITY OPTIONS

THE UART USED FOR SERIAL TRANSMISSION CAN BE PROGRAMMED FOR VARIOUS PARITY AND STOP BIT OPTIONS.

```
0000: 3E xx       START   LD    A,VALUE  ;CONTROL PARAMETER
0002: D3 FD               OUT   (0FDH),A ;PROGRAM UART
0004: C9                  RET            ;BACK TO MAIN PROGRAM
```

```
          7  6  5  4  3  2  1  0   (8 BIT POSITIONS)
FORMAT=   X  X  X  P  PS S NB2 NB1
```

```
X   = DON'T CARE BIT POSITION
P   = PARITY ENABLE :  0=ENABLED   1=NONE
PS  = PARITY SELECT :  0=ODD       1=EVEN
S   = # OF STOP BITS:  0=ONE       1=TWO BITS
NB2 - NB1 = NUMBER OF BITS PER CHARACTER
```

NB2	NB1	# OF BITS
0	0	5
0	1	6
1	0	7
1	1	8

```
EXAMPLE:   VALUE = 0EH = 00001110 BINARY
                         7 BITS PER CHARACTER
                         2 STOP BITS
                         PARITY ENABLED
                         PARITY EVEN
```

CHAPTER 13 --- CASSETTE TAPE

13.01 WRITE DATA TO CASSETTE TAPE

```
100 MS=256*PEEK(-4095)+PEEK(-4096) : REM MEMORY SIZE
110 IF MS>32767 THEN MS=MS-65536
120 POKE MS-45,8 : POKE MS-44,1 : REM DISABLE KEYBOARD
130 POKE MS-41,16               : REM MOTOR #1 CONTROL
140 OUT 254,16                  : REM TURN ON MOTOR #1.
150 POKE MS-47,18:POKE MS-46,244: REM OUTPUT TO TAPE
160 FOR J=1 TO 10
170 FOR K=1 TO 100:NEXT K       : REM DELAY BETWEEN DATA
180 PRINT A$(J);",";A(J)        : REM PRINT DATA ON TAPE
190 NEXT J
200 POKE MS-47,27               : REM RESTORE VIDEO
210 OUT 254,0                   : REM OFF MOTOR #1
220 POKE MS-45,24:POKE MS-44,224: REM RESTORE KEYBOARD
```

BOTH STINGS AND NUMBERS MAY BE PRINTED ON THE TAPE.
"," SEPARATES THE DATA TO MATCH THE INPUT STATEMENT.
THE FILE CREATED HAS NO NAME AND NO CRC ERROR CHECKING.
KEYBOARD IS DISABLED SO THAT ITS USUAL SCAN DOES NOT
TURN THE CASSETTE MOTORS OFF.

==========

13.02 READ DATA FROM CASSETTE TAPE

```
400 MS=256*PEEK(-4095)+PEEK(-4096) : REM MEMORY SIZE
410 IF MS>32767 THEN MS=MS-65536
420 POKE MS-45,15:POKE MS-44,224: REM TAPE TO INPUT
430 POKE MS-41,16               : REM MOTOR #1 CONTROL
440 OUT 254,16                  : REM TURN ON MOTOR #1.
450 FOR J=1 TO 10
460 FOR K=1 TO 30:NEXT K : REM DELAY < THAN WHEN WRITTEN
470 INPUT A$(J),A(J)     : REM INPUT DATA FROM TAPE
480 NEXT J
490 POKE MS-45,24        : REM RESTORE INPUT TO KEYBOARD
500 OUT 254,0            : REM TURN OFF MOTOR #1
```

THIS ROUTINE READS THE DATA TAPES CREATED BY 13.01. THE
INPUT STATEMENT ON LINE 470 IS A MATCHED STATEMENT TO
THE PRINT STATEMENT ON LINE 180. THE INPUT VECTOR POINTS
TO THE TAPE INPUT ROUTINE RATHER THAN TO THE KEYBOARD.

CHAPTER 14 --- EDITOR FOR BASIC

14.01 EDITOR FOR BASIC INSTRUCTIONS.

```
TO USE  --  >LO EDIT          (Load the editor from 14.02)
            >SE I=7000        (For 32K version, ie. ORG address)
            >PP               (Exit back to Basic)
```

Any cursor movement key, HOME, or TAB will activate edit mode as indicated by the inverse video cursor.

CTRL E -- Expand the line by moving the rightmost characters one space right for insertion ahead of the cursor.

CTRL R -- Reduce the line by deleting the character under the cursor. Rightmost characters move one space left.

RUBOUT -- Rubout the character under the cursor and replace with a space. Note that RUBOUT is now unshifted.

CTRL N -- Renumber all program statements in increments of 10 starting at 100. Starting line number is stored in bytes 7136 and 7137 hex, and the increment is stored in bytes 71A1 and 71F9 hex if you desire to change them.

```
Example:  >EN 7136          (Enter Monitor by typing  BYE )
          7136: E8 03 /     (Change starting line # to 1000)
          >EN 71A1
          71A1: 64 /        (Changes increment to 100)
          >EN 71F9
          71F9: 64 /
          >PP               (Exit back to Basic)
```

CTRL U -- Up (ie. revive) a program lost due to mistakenly typing NEW or CLOAD. (Hard reset destroys a portion of your program near the start. However, you may be able to revive from hard reset by using CTRL U, listing the program, and deleting the line number that is messed up. Loosing a couple lines at the start of a program from a hard reset is better than loosing the entire program.)

TAB -- Tab to the preset tabs in columns 1, 9, 17, 25, 33, 41, 49, 57, and enter edit mode.

INVERSE CURSOR -- The editor's cursor is the inverse video of the character it is sitting on top of.

LINE NUMBER EDITING -- Editing a statement's line number will COPY the line under the new line number. The original line still exits and may deleted, if you desire, by typing the old line number and RETURN.

WRITING CODE -- Preferably enter edit mode before starting a statement. (If you enter the edit mode DURING the typing of a line, all of the text on the screen will be added behind what you have already typed. Therefore, return to the line in edit mode and hit RETURN a second time. Basic will then have the line as it appears on the screen.)

TRANSPARENCY -- When not in edit mode, the keyboard functions normally wherein you may RUN, CSAVE, LIST, write code, insert or delete lines, etc. The Monitor's >SE I=K will restore the regular Sorcerer keyboard routine.

CLEAR -- Using CLEAR will no longer generate a SYNTAX ERROR message.

CTRL X -- Cross reference the Basic program's variables and statement references such as GOTO, GOSUB, RESTORE, etc. This feature is present ONLY if you bought our CROSS REFERENCE program and added it to your Editor for Basic.

14.02 EDITOR FOR BASIC SOURCE LISTING

```
                                ORG     7000H
7000: CD 1C EB      EDITOR      CALL 0EB1CH       ;SCAN KEYBOARD
7003: C8                        RET  Z            ;RETURN IF NOTHING
7004: C5                        PUSH BC
7005: D5                        PUSH DE
7006: E5                        PUSH HL
7007: FD E5                     PUSH IY
7009: CD A2 E1                  CALL 0E1A2H       ;GET IY
700C: FE 0C                     CP   00CH         ;CLEAR KEY?
700E: 20 05                     JR   NZ,L7015-$
7010: CD 0C E0                  CALL 0E00CH       ;CLEAR VIDEO CRT
7013: 18 71                     JR   CLEAR-$      ;RET WITH NOTHING
7015: FE 01         L7015       CP   001H         ;CURSOR LEFT
7017: 28 46                     JR   Z,VIDEO-$
7019: FE 11                     CP   011H         ;CURSOR HOME
701B: 28 42                     JR   Z,VIDEO-$
701D: FE 13                     CP   013H         ;CURSOR RIGHT
701F: 28 3E                     JR   Z,VIDEO-$
7021: FE 17                     CP   017H         ;CURSOR UP
7023: 28 3A                     JR   Z,VIDEO-$
7025: FE 1A                     CP   01AH         ;CURSOR DOWN
7027: 28 36                     JR   Z,VIDEO-$
7029: FE 0B                     CP   00BH         ;TAB
702B: 28 62                     JR   Z,TAB-$
702D: FE 05                     CP   005H         ;CTRL E - EXPAND
702F: 28 73                     JR   Z,EXPAND-$
7031: FE 12                     CP   012H         ;CTRL R - REDUCE
7033: 28 7C                     JR   Z,REDUCE-$
7035: FE 0E                     CP   00EH         ;CTRL N - RENUMBER
7037: CA 35 71                  JP   Z,RENUM
703A: FE 15                     CP   015H         ;CTRL U - UP PROG
703C: CA 19 71                  JP   Z,CLOAD
703F: FE 7F                     CP   07FH         ;RUBOUT
7041: 20 04                     JR   NZ,L7047-$
7043: 3E 5F                     LD   A,05FH       ;CHANGE TO UNDERSC
7045: 18 06                     JR   L704D-$
7047: FE 5F         L7047       CP   05FH         ;UNDERSCORE
7049: 20 02                     JR   NZ,L704D-$
704B: 3E 7F                     LD   A,07FH       ;CHANGE TO RUBOUT
704D: 47            L704D       LD   B,A
704E: 3A 18 71                  LD   A,(FLAG)
7051: B7                        OR   A
7052: 78                        LD   A,B          ;RESTORE CHAR
7053: 28 32                     JR   Z,END-$      ;EXIT NORMAL MODE
7055: FE 0D                     CP   00DH         ;<CR>
7057: 28 7D                     JR   Z,LINE-$
7059: FE 7F                     CP   07FH         ;RUBOUT
705B: 20 02                     JR   NZ,VIDEO-$
705D: 3E 08                     LD   A,008H       ;BACKSPACE
705F: 32 18 71      VIDEO       LD   (FLAG),A     ;ENTER EDIT MODE
7062: CD 0C E0                  CALL 0E00CH       ;CHAR TO VIDEO
7065: CD D6 E9                  CALL 0E9D6H       ;GET CURSOR ADDR
```

```
;
;  REVERSE VIDEO CHARACTER UNDER CURSOR
;
7068: 36 FE          INVERSE LD   (HL),0FEH    ;INVERSE VIDEO
706A: FD 6E 67               LD   L,(IY+067H) ;CHAR UNDER CURSOR
706D: 26 00                  LD   H,000H
706F: 29                     ADD  HL,HL
7070: 29                     ADD  HL,HL
7071: 29                     ADD  HL,HL       ;ASCII OFFSET
7072: 11 00 F8               LD   DE,0F800H   ;TABLE BASE
7075: 19                     ADD  HL,DE       ;HL = BIT PATTERN
7076: DD 21 F0 FF            LD   IX,0FFF0H   ;GRAPHIC CHAR 254
707A: 06 08                  LD   B,008H      ;ROW COUNTER
707C: 7E             L707C   LD   A,(HL)      ;GET BIT PATTERN
707D: 2F                     CPL              ;INVERSE
707E: DD 77 00               LD   (IX+000H),A ;CREATE GRAPHICS
7081: 23                     INC  HL          ;NEXT ROW
7082: DD 23                  INC  IX
7084: 10 F6                  DJNZ L707C-$     ;LOOP 8 TIMES
;
7086: AF             CLEAR   XOR  A           ;RETURN NOTHING
7087: FE 00          END     CP   000H        ;SET Z FLAG
7089: FD E1                  POP  IY
708B: E1                     POP  HL
708C: D1                     POP  DE
708D: C1                     POP  BC
708E: C9                     RET              ;EXIT
;
;  PERFORM TAB FUNCTION
;
708F: CD E8 E9       TAB     CALL 0E9E8H      :REPLACE CURSOR
7092: FD 7E 6A               LD   A,(IY+06AH) ;COLUMN NUMBER
7095: C6 08                  ADD  A,008H
7097: E6 38                  AND  038H        ;UNIFORM TABS
7099: FD 77 6A               LD   (IY+06AH),A ;NEW COLUMN
709C: 32 18 71       L709C   LD   (FLAG),A    ;ENTER EDIT MODE
709F: CD CC E9               CALL 0E9CCH      ;MOVE CURSOR
70A2: 18 C4                  JR   INVERSE-$   ;INVERSE VIDEO
;
;  EXPAND OR REDUCE LINE ONE CHARACTER AT CURSOR LOCATION
;
70A4: CD C4 70       EXPAND  CALL DELTA       ;GET SCREEN ADDRS
70A7: 28 13                  JR   Z,L70BC-$
70A9: E5                     PUSH HL
70AA: D1                     POP  DE
70AB: 2B                     DEC  HL
70AC: ED B8                  LDDR             ;SHIFT LINE RIGHT
70AE: 23                     INC  HL
70AF: 18 0B                  JR   L70BC-$
```

```
;
70B1: CD C4 70      REDUCE  CALL DELTA        ;GET SCREEN ADDRS
70B4: 28 06                 JR   Z,L70BC-$
70B6: D5                    PUSH DE
70B7: E1                    POP  HL
70B8: 23                    INC  HL
70B9: ED B0                 LDIR              ;SHIFT LINE LEFT
70BB: 2B                    DEC  HL
70BC: 3E 20         L70BC   LD   A,020H       ;INSERT SPACE
70BE: 77                    LD   (HL),A
70BF: CD CC E9              CALL 0E9CCH       ;SAVE CURSOR
70C2: 18 A4                 JR   INVERSE-$
;
;  CALCULATE NUMBER OF CHARACTERS TO SHIFT
;
;  ON EXIT:   DE = PRESENT CURSOR ADDR.   HL = LINE END ADDR.
;
70C4: CD E8 E9      DELTA   CALL 0E9E8H       ;REPLACE CURSOR
70C7: 7D                    LD   A,L
70C8: E6 3F                 AND  03FH
70CA: D6 40                 SUB  040H
70CC: 2F                    CPL
70CD: 4F                    LD   C,A
70CE: 06 00                 LD   B,000H       ;BC = # CHAR
70D0: EB                    EX   DE,HL        ;DE = PRESENT
70D1: C5                    PUSH BC
70D2: E1                    POP  HL
70D3: 19                    ADD  HL,DE        ;HL = LINE END
70D4: B1                    OR   C            ;TEST BC=0
70D5: C9                    RET
;
;  COPY EDITED LINE FROM SCREEN TO REPLACE ONE IN MEMORY
;
70D6: 21 F4 70      LINE    LD   HL,NEW       ;NEW INPUT VECTOR
70D9: FD 75 41              LD   (IY+041H),L
70DC: FD 74 42              LD   (IY+042H),H
70DF: CD C4 70              CALL DELTA        ;GET LINE END ADDR
70E2: 06 41                 LD   B,041H       ;65 CHAR DEFAULT
70E4: 3E 20                 LD   A,020H
70E6: BE            L70E6   CP   (HL)
70E7: 20 04                 JR   NZ,L70ED-$
70E9: 2B                    DEC  HL
70EA: 10 FA                 DJNZ L70E6-$      ;SUBTRACT SPACES
70EC: 04                    INC  B            ;B <> 0
70ED: 78            L70ED   LD   A,B          ;# CHAR TO MOVE
70EE: FD 36 6A 00           LD   (IY+06AH),000H  ;FIRST COLUMN
70F2: 18 A8                 JR   L709C-$      ;GO UPDATE MEMORY
```

```
;
;  I/O INPUT FROM SCREEN TO BASIC.  REPLACES KEYBOARD INPUT.
;
70F4: FD E5          NEW     PUSH IY
70F6: CD A2 E1               CALL 0E1A2H       ;GET IY
70F9: 3A 18 71               LD   A,(FLAG)     ;DEC CHAR COUNTER
70FC: 3D                     DEC  A
70FD: 32 18 71               LD   (FLAG),A
7100: 20 0F                  JR   NZ,L7111-$   ;LOOP TIL DONE
7102: E5                     PUSH HL
7103: 21 00 70               LD   HL,EDITOR    ;RESTORE KEYBOARD
7106: FD 75 41               LD   (IY+041H),L
7109: FD 74 42               LD   (IY+042H),H
710C: E1                     POP  HL
710D: 3E 0D                  LD   A,00DH       ;RETURN WITH <CR>
710F: 18 03                  JR   L7114-$
7111: FD 7E 67       L7111   LD   A,(IY+067H)  ;GET CHARACTER
7114: B7             L7114   OR   A            ;SET NZ FLAG
7115: FD E1                  POP  IY
7117: C9                     RET
;
7118: 00             FLAG    DEFB 0
;
;  RESTORE PROGRAM AFTER CLOAD
;
7119: 21 D9 01       CLOAD   LD   HL,001D9H    ;PROGRAM START
711C: AF                     XOR  A
711D: BE             L711D   CP   (HL)         ;LOOK FOR ZERO
711E: 23                     INC  HL
711F: 20 FC                  JR   NZ,L711D-$   ;LOOP TIL FOUND
7121: 22 D5 01               LD   (L01D5),HL   ;RESTORE POINTER
7124: 5E             L7124   LD   E,(HL)       ;NEXT LINE ADDR
7125: 23                     INC  HL
7126: 56                     LD   D,(HL)
7127: EB                     EX   DE,HL
7128: 7D                     LD   A,L
7129: B4                     OR   H
712A: 20 F8                  JR   NZ,L7124-$   ;FIND PROG END
712C: 13                     INC  DE
712D: 13                     INC  DE
712E: ED 53 B7 01            LD   (L01B7),DE   ;SAVE END ADDRESS
7132: C3 86 70               JP   CLEAR
;
;  PERFORM RENUMBER FUNCTION
;
7135: 21 64 00       RENUM   LD   HL,100       ;DEFAULT START
7138: 22 58 72               LD   (INIT),HL
713B: 21 D5 01               LD   HL,01D5H     ;PROG START ADDR
713E: 22 54 72               LD   (NEXT),HL
```

```
;
;  PUT ADDRESS OF NEXT LINE IN (NEXT)
;
7141: 2A 54 72      NEXTLIN LD   HL,(NEXT)
7144: 22 4E 72              LD   (PRESENT),HL
7147: 5E                    LD   E,(HL)
7148: 23                    INC  HL
7149: 56                    LD   D,(HL)       ;DE = NEXT LINE
714A: 23                    INC  HL
714B: ED 53 54 72           LD   (NEXT),DE
714F: 7B                    LD   A,E
7150: B2                    OR   D
7151: CA E6 71              JP   Z,PASS2      ;TEST END OF PROG
7154: 23                    INC  HL
7155: 23                    INC  HL
;
;  LOOK FOR REM, GOTO, GOSUB, THEN
;
7156: 7E            LOOK    LD   A,(HL)       ;GET CHARACTER
7157: 23                    INC  HL
7158: FE 00         L7158   CP   000H         ;END OF LINE?
715A: 28 E5                 JR   Z,NEXTLIN-$
715C: FE 8F                 CP   08FH         ;REM ?
715E: 28 E1                 JR   Z,NEXTLIN-$
7160: FE 89                 CP   089H         ;GOTO ?
7162: 28 0E                 JR   Z,GOTO-$
7164: FE 8D                 CP   08DH         ;GOSUB ?
7166: 28 0A                 JR   Z,GOTO-$
7168: FE 8C                 CP   08CH         ;RESTORE ?
716A: 28 06                 JR   Z,GOTO-$
716C: FE A2                 CP   0A2H         ;THEN
716E: 28 02                 JR   Z,GOTO-$
7170: 18 E4                 JR   LOOK-$       ;CONTINUE SEARCH
;
;  PROCESS GOTO, GOSUB, THEN STATEMENTS
;
7172: E5            GOTO    PUSH HL           ;SAVE BEGIN ADDR
7173: CD EA C7              CALL 0C7EAH       ;CONVERT TO BINARY
7176: 22 50 72              LD   (POINT),HL   ;END OF # ADDR
7179: AF                    XOR  A
717A: C1                    POP  BC
717B: ED 42                 SBC  HL,BC        ;LENGTH OF LINE #
717D: 22 56 72              LD   (LENGTH),HL
```

```
;
;  SEARCH FOR LINE # MATCH
;
7180: 2A 58 72              LD   HL,(INIT)
7183: 22 52 72              LD   (COUNT),HL  ;INITIALIZE LINE #
7186: 21 D5 01              LD   HL,01D5H
7189: 4E            L7189   LD   C,(HL)
718A: 23                    INC  HL
718B: 46                    LD   B,(HL)      ;NEXT LINE ADDR
718C: 23                    INC  HL
718D: 79                    LD   A,C
718E: B0                    OR   B
718F: 28 48                 JR   Z,L71D9-$   ;END OF PROG?
7191: D5                    PUSH DE          ;SAVE MATCH #
7192: 5E                    LD   E,(HL)
7193: 23                    INC  HL
7194: 56                    LD   D,(HL)      ;CURRENT LINE #
7195: EB                    EX   DE,HL
7196: D1                    POP  DE
7197: CD 74 C5              CALL 0C574H      ;COMPARE HL-DE
719A: 28 10                 JR   Z,L71AC-$
719C: D5                    PUSH DE          ;NO MATCH
719D: 2A 52 72              LD   HL,(COUNT)
71A0: 11 0A 00              LD   DE,10       ;ADD 10 TO COUNT
71A3: 19                    ADD  HL,DE
71A4: 22 52 72              LD   (COUNT),HL
71A7: D1                    POP  DE
71A8: 69                    LD   L,C
71A9: 60                    LD   H,B         ;NEXT LINE ADDR
71AA: 18 DD                 JR   L7189-$
;
;  FOUND MATCH, CONVERT LINE # TO DECIMAL AND REPLACE
;
71AC: CD B2 C9      L71AC   CALL 0C9B2H      ;FIRST COLUMN
71AF: 2A 52 72              LD   HL,(COUNT)  ;# TO CONVERT
71B2: CD BB D7              CALL 0D7BBH      ;PRINT DECIMAL #
71B5: ED 4B 56 72           LD   BC,(LENGTH) ;LENGTH OF #
71B9: 41                    LD   B,C
71BA: CD A2 E1              CALL 0E1A2H      ;GET IY
71BD: CD D6 E9              CALL 0E9D6H      ;GET CURSOR ADDR
71C0: 7D                    LD   A,L
71C1: E6 07                 AND  007H
71C3: 4F                    LD   C,A
71C4: 78                    LD   A,B
71C5: 91                    SUB  C
71C6: FC 0C 72              CALL M,EXPAND    ;MOVE PROG DOWN
71C9: 2B                    DEC  HL
71CA: DD 2A 50 72           LD   IX,(POINT)
71CE: DD 2B                 DEC  IX
;
;  MOVE NUMBER ON SCREEN TO MEMORY IN REVERSE ORDER
;
71D0: 7E            L71D0   LD   A,(HL)
71D1: DD 77 00              LD   (IX+000H),A ;TRANSFER CHAR
71D4: 2B                    DEC  HL
71D5: DD 2B                 DEC  IX
71D7: 10 F7                 DJNZ L71D0-$
```

```
;
;  CHECK FOR COMMA AFTER NUMBER
;
71D9: 2A 50 72      L71D9   LD   HL,(POINT)   ;LAST LOOK ADDR
71DC: 7E                    LD   A,(HL)
71DD: FE 2C                 CP   02CH         ;IS IT A COMMA?
71DF: 23                    INC  HL
71E0: CA 72 71              JP   Z,GOTO       ;YES, CONVERT #
71E3: C3 58 71              JP   L7158        ;NO, CONTINUE SCAN
;
;  PASS TWO - RENUMBER ALL LINES
;
71E6: 21 D5 01      PASS2   LD   HL,01D5H     ;START OF PROGRAM
71E9: ED 5B 58 72           LD   DE,(INIT)    ;NEW LINE NUMBER
71ED: 4E            L71ED   LD   C,(HL)
71EE: 23                    INC  HL
71EF: 46                    LD   B,(HL)       ;GET FORWARD LINK
71F0: 23                    INC  HL
71F1: 79                    LD   A,C
71F2: B0                    OR   B
71F3: 28 0C                 JR   Z,L7201-$    ;END OF PROGRAM?
71F5: 73                    LD   (HL),E
71F6: 23                    INC  HL
71F7: 72                    LD   (HL),D       ;STORE NEW LINE #
71F8: 21 0A 00              LD   HL,10        ;DEFAULT INCREMENT
71FB: 19                    ADD  HL,DE
71FC: EB                    EX   DE,HL
71FD: 69                    LD   L,C
71FE: 60                    LD   H,B
71FF: 18 EC                 JR   L71ED-$      ;LOOP TIL PROG END
;
7201: 21 AF C2      L7201   LD   HL,0C2AFH
7204: CD 15 D0              CALL 0D015H       ;PRINT READY
7207: 3E 0D                 LD   A,00DH       ;RETURN <CR>
7209: C3 87 70              JP   END
;
;  MOVE PROGRAM DOWN
;
720C: C5            EXPAND  PUSH BC
720D: E5                    PUSH HL
720E: 79                    LD   A,C
720F: 90                    SUB  B            ;A = # TO MOVE
7210: 4F                    LD   C,A
7211: 06 00                 LD   B,000H
7213: C5                    PUSH BC
7214: 2A 54 72              LD   HL,(NEXT)
7217: 09                    ADD  HL,BC
7218: 22 54 72              LD   (NEXT),HL    ;ADJUST POINTERS
721B: 2A 50 72              LD   HL,(POINT)
721E: 09                    ADD  HL,BC
721F: 22 50 72              LD   (POINT),HL   ;ADJUST POINTERS
7222: EB                    EX   DE,HL
7223: 2A B7 01              LD   HL,(01B7H)   ;NEW PROG END ADDR
```

```
7226: E5                    PUSH HL
7227: 09                    ADD  HL,BC
7228: 22 B7 01              LD   (01B7H),HL
722B: E5                    PUSH HL
722C: ED 52                 SBC  HL,DE
722E: E5                    PUSH HL
722F: C1                    POP  BC
7230: D1                    POP  DE
7231: E1                    POP  HL
7232: 03                    INC  BC
7233: ED B8                 LDDR              ;MOVE PROG DOWN
7235: 2A 4E 72              LD   HL,(PRESENT)
7238: C1                    POP  BC           ;DISTANCE MOVED
;
7239: 5E            RELINK  LD   E,(HL)       ;RELINK BOTTOM
723A: 23                    INC  HL
723B: 56                    LD   D,(HL)
723C: 7B                    LD   A,E
723D: B2                    OR   D
723E: 28 0A                 JR   Z,L724A-$    ;AT PROG END?
7240: 2B                    DEC  HL
7241: EB                    EX   DE,HL
7242: 09                    ADD  HL,BC
7243: EB                    EX   DE,HL
7244: 73                    LD   (HL),E
7245: 23                    INC  HL
7246: 72                    LD   (HL),D       ;ADJUST LINK
7247: EB                    EX   DE,HL
7248: 18 EF                 JR   RELINK-$
;
724A: E1            L724A   POP  HL
724B: C1                    POP  BC
724C: 41                    LD   B,C          ;RELINK FINISHED
724D: C9                    RET
;
724E: 00            PRESENT DEFW 0
7250: 00            POINT   DEFW 0
7252: 00            COUNT   DEFW 0
7254: 00            NEXT    DEFW 0
7256: 00            LENGTH  DEFW 0
7258: 00            INIT    DEFW 0
```

CHAPTER 15 --- CP/M

15.01 SAVE ROMPAC BASIC PROGRAM ON CP/M DISK.

REQUIREMENT: DISK BOOT ADDRESS MUST NOT CONFLICT WITH THE ROMPAC ADDRESSES OF C000 - DFFF HEX.

```
CLOAD XMPLE                 CLOAD PROGRAM FROM TAPE.
BYE
>DU 1B7 1B8                 FIND END OF BASIC PROGRAM.
01B7: C4 24                 TYPICAL END-OF-PROGRAM ADDRESS.
>GO B900                    TYPICAL DISK BOOT ADDRESS.
A>SAVE 36 XMPLE.COM         STORE xx BLOCKS ON DISK.
A>XMPLE                     LOAD PROGRAM FROM DISK.
READY                       PRINTED BY BASIC ROMPAC.
RUN                         EXECUTE ROMPAC BASIC.
```

JUST CONVERT CONTENTS OF BYTE 1B8 TO DECIMAL FOR xx. WHEN CP/M LOADS THE PROGRAM AND BEGINS EXECUTION, IT FINDS A JP 0C06BH AT 100H. THIS JUMP WAS PLACED AT 100H BY THE BASIC ROMPAC.

==========

15.02 SAVE WORD PROCESSOR ROMPAC FILES ON CP/M DISK.

REQUIREMENT: DISK BOOT ADDRESS MUST NOT CONFLICT WITH THE ROMPAC ADDRESSES OF C000 - DFFF HEX.

```
COMMAND> X                  EXIT TO MONITOR FROM WP ROMPAC.
>DU 74A 74B                 FIND END OF WORD PROCESSOR FILE.
074A: 45 1A                 TYPICAL END-OF-FILE ADDRESS
>EN 100
0100: C3 03 C0              NEED JUMP TO ROMPAC WARM START.
>GO B900                    TYPICAL DISK BOOT ADDRESS.
A>SAVE 26 WPFILE.COM        SAVE xx BLOCKS THROUGH FILE END.
                            26 DECIMAL = 1A HEX FROM 74B.
A>WPFILE                    RELOAD AND JUMP TO WARM START.
```

15.03 CP/M COMMANDS

PIP

```
PIP A:=B:XXX.COM            Copy XXX file from      B to A
PIP A:=B:*.*                Copy all files from     B to A
PIP B:=C:*.COM              Copy all COM files from C to B
PIP B:PROG.BAK=C:PROG.COM   Make backup of PROG from C to B
PIP B:=C:*.COM[V]           Copies all COM files from C to B
                               with verification
PIP B:XXX=A:YYY             Copy YYY from A to B and rename XXX
PIP B:=A:XXX.*              Copy all files with name of XXX on
                                A onto B
```

REN

```
REN NEW.COM=OLD.COM     Rename file name
REN NEW.BAK=OLD.COM     Rename file OLD type COM to NEW type BAK
REN XXX.BAK=XXX.COM     Rename filetype
REN C:XXX.COM=YYY.COM   Rename filename YYY on C to XXX
```

ERA

```
ERA XXX.COM           Erase filename XXX with type COM
ERA *.DAT             Erase all file types of DAT
ERA XXX.*             Erase all files with name of XXX
ERA *.*               Erase all files
ERA C:*.COM           Erase all COM type files on C
```

SAVE

```
A>SAVE xx PROGNAME.COM      Save file on A
A>SAVE xx B:PROGNAME.COM    Save file on B
```

CTRL CHARACTERS

```
CTRL X - delete line typed
CTRL R - retype line
CTRL C - reboot system  (warm)
```

15.04 BIOS MODIFICATIONS TO GIVE BACKSPACE TYPE RUBOUTS

```
;
;                       CONSOLE INPUT
;
;  This routine must get a character from the console,
;  and return the character in the ACCUMULATOR.
;
;  If the character is a RUBOUT, then a new delete flag
;  is set to cause the backspace sequence of characters
;  to be sent to the VIDEO display by the CONSOLE OUTPUT
;  driver.
;
B24D: CD 26 B2   CINP    CALL CSTAT       ;IS A CHARACTER READY?
B250: CA 4D B2           JZ   CINP        ;NO--WAIT FOR IT.
B253: CD 09 E0           CALL KEYBRD      ;GET THE CHARACTER.
B256: CA 4D B2           JZ   CINP        ;TRY AGAIN IF NOT IN TIME
B259: FE 0B              CPI  'K'-40H     ;IS THE CHARACTER A TAB?
B25B: CA 6E B2           JZ   TABIT       ;YES--CONVERT IT.
B25E: FE 09              CPI  'I'-40H     ;IS IT A CONTROL I ?
B260: CA 71 B2           JZ   UNTAB       ;YES--
B263: FE 7F              CPI  5FH         ;IS THIS AN UNDERSCORE?
B265: C0                 RNZ              ;NO, FINISHED WITH CINP
B266: 3E 01              MVI  A,1         ;SET
B268: 32 EA B2           STA  DELF        ;    DELETE FLAG
B26B: 3E 7F              MVI  A,7FH       ;CHANGE TO RUBOUT
B26D: C9                 RET
                         ;
B26E: 3E 09      TABIT   MVI  A,'I'-40H   ;CHANGE TO THE CP/M TAB
B270: C9                 RET
                         ;
B271: 3E 0B      UNTAB   MVI  A,'K'-40H   ;CHANGE TO CONTROL K.
B273: C9                 RET
;
;
;                       CONSOLE OUTPUT
;
;  This routine will be called with the character to be
;  output in the 'C' REGISTER.   If the delete flag is
;  set, then generate the characters to erase the deleted
;  character from the video display rathar than echo it.
;
B274: 3A EA B2   COUT    LDA  DELF        ;CHECK
B277: FE 01              CPI  1           ;   DELETE FLAG
B279: C2 97 B2           JNZ  JMPRBO      ;NO, GO JMPRBO
B27C: E5                 PUSH H
B27D: 21 EB B2           LXI  H,CHAR2
B280: 3E 01              MVI  A,1         ;MOVE CURSOR LEFT
B282: CD 1B E0           CALL VIDEO
B285: 3E 20              MVI  A,020H      ;SPACE ERASE CHAR
B287: CD 1B E0           CALL VIDEO
B28A: 3E 01              MVI  A,1         ;MOVE CURSOR LEFT
B28C: CD 1B E0           CALL VIDEO
```

```
B28F: 34                INR  M          ;ADJUST CHAR COUNT
B290: E1                POP  H
B291: 3E 00             MVI  A,0        ;RESET
B293: 32 EA B2          STA  DELF       ;   DELETE FLAG
B296: C9                RET
                        ;
B297: 79          JMPRBO MOV  A,C        ;GET IT TO THE ACUM.
B298: E5                PUSH H          ;SAVE HL.
B299: 21 EB B2          LXI  H,CHAR2    ;POINT TO CHAR COUNT
B29C: FE 0D             CPI  0DH        ;CARRIAGE RETURN?
B29E: CA B5 B2          JZ   NEW
B2A1: FE 0C             CPI  0CH        ;FORM FEED?
B2A3: CA B5 B2          JZ   NEW
B2A6: 35                DCR  M          ;COUNT OFF CHAR.
B2A7: C2 B7 B2          JNZ  VOUT       ;GO WORK.
B2AA: 3E 0D             MVI  A,13
B2AC: CD 1B E0          CALL VIDEO      ;SEND CR
B2AF: 3E 0A             MVI  A,10
B2B1: CD 1B E0          CALL VIDEO      ;SEND A LINE FEED
B2B4: 79                MOV  A,C        ;CHARACTER TO ACCUM
B2B5: 36 42        NEW  MVI  M,42H      ;RESET CHARACTER COUNT
B2B7: CD 1B E0    VOUT  CALL VIDEO      ;MORE WORK.
B2BA: E1                POP  H          ;JUST LIKE WE CAME IN
B2BB: C9                RET             ;BACK TO CALLER
;
                        ORG  0B2EAH     ;IE. AFTER END OF BIOS
B2EA: 00           DELF DEFB 0          ;DELETE FLAG
;
```

15.05 CP/M LOAD ROUTINE OF WP FILES FOR DEVPAC

This program operates with the Development Pac by bringing a Word Processor File into the DEVPAC file area from CP/M disk. The program then sets the I/O vectors, and jumps to the Development Pac.


```
        ORG   100H
BDOS    EQU   0005H        ;DOS ENTRY POINT
OPENF   EQU   15           ;FILE OPEN
READF   EQU   20           ;READ FUNCTION
DMAF    EQU   26           ;SET DMA ADDRESS FUNCTION
FILE    EQU   05A80H       ;START ADDRESS OF FILE FOR
                           ;46K MEMORY CONFIGURATION
FCB     EQU   5CH          ;FILE CONTROL BLOCK
FCBCR   EQU   FCB+32       ;CURRENT (NEXT) RECORD
BUFINC  EQU   80H          ;READ BLOCK LENGTH
;
OPEN    LD    DE,FCB       ;OPEN NAMED FILE
        LD    C,OPENF
        CALL  BDOS
        CP    255          ;CHECK FOR OPEN ERROR
        RET   Z            ;BAD OPEN
        XOR   A
        LD    (FCBCR),A    ;SET FIRST RECORD TO 0
;
MAIN    PUSH  BC           ;SAVE BC
        LD    BC,FILE
        LD    (FILBUF),BC
        CALL  MOVBUF
        CALL  DISKR
        CP    0            ;CHECK FOR ERROR
        JR    Z,LOOP-$
        POP   BC
        RET                ;BACK TO CPM ON READ ERROR
;
LOOP    PUSH  HL           ;SAVE HL
        LD    HL,(FILBUF)
        LD    BC,BUFINC    ;BUFFER LENGTH
        ADD   HL,BC
        LD    (FILBUF),HL
        POP   HL
        CALL  MOVBUF
        CALL  DISKR
        CP    0            ;CHECK FOR ERROR
        JR    Z,LOOP-$
        POP   BC
        RET                ;BACK TO CPM ON READ ERROR
;
```

```
DISKR   PUSH HL          ;READ ONE BUFFER FROM DISK
        PUSH DE
        PUSH BC
        LD   DE,FCB
        LD   C,READF
        CALL BDOS
        POP  BC
        POP  DE
        POP  HL
        CP   1           ;CHECK FOR EOF
        JR   Z,FINIS-$
        RET
;
MOVBUF  PUSH HL          ;MOVE BUFFER TO RAM
        PUSH DE
        PUSH BC
        LD   DE,(FILBUF)
        LD   C,DMAF
        CALL BDOS
        POP  BC
        POP  DE
        POP  HL
        RET
;
CI      EQU  0F01EH
CO      EQU  0F020H
OI      EQU  0F022H
OO      EQU  0F024H
SI      EQU  0F026H
SO      EQU  0F028H
SK      EQU  0C547H
SV      EQU  0C54EH
AO      EQU  0C60AH
AI      EQU  0C5F5H
BO      EQU  0C624H
;
FINIS   LD   BC,SK       ;SET DEVELOPMENT PAC VECTORS
        LD   (CI),BC
        LD   BC,SV
        LD   (CO),BC
        LD   BC,AO
        LD   (OI),BC
        LD   BC,AI
        LD   (OO),BC
        LD   BC,BO
        LD   (SI),BC
        LD   BC,SV
        LD   (SO),BC
        LD   BC,FILE
;
```

```
ELOOP   INC   BC            ;NOW PUT A '0' AT EOF
        LD    A,(BC)
        CP    01AH          ;IS IT DISK EOF MARKER?
        JR    NZ,ELOOP-$
        XOR   A
        LD    (BC),A
        POP   BC            ;POP DISKR RETURN ADDRESS
        POP   BC            ;RESTORE BC FROM MAIN PUSH
        JP    0E000H        ;TO RESET ADDRESS OF DEVPAC
;
FILBUF  DEFW 0
;       END
```

CHAPTER 16 --- WORD PROCESSOR

16.01 WORD PROCESSOR PRINTER DRIVER

INSTALL YOUR OWN PRINTER DRIVER BY PUTTING THE ADDRESS
OF YOUR ROUTINE IN BYTES 7E7 AND 7E8 HEX.
USES DRIVER #1 IN THE Y TABLE.

```
0000: 21 yy xx  ENTRY  LD  HL,NEWDRIVER
0003: 22 E7 07         LD  (07E7H),HL
```

==========

16.02 SALVAGE WORD PROCESSOR FILE FROM RESET.

IF THE WORD PROCESSOR RESETS, AND YOU ARE CURSING BECAUSE
YOUR FILE WASN'T SAVED ON TAPE OR DISK YET, ALL IS NOT
LOST. YOU MAY BE ABLE TO SALVAGE YOUR FILE (EVERYTHING
EXCEPT FOR THE FIRST 175 CHARACTERS.)

GO TO THE MONITOR AND TRY:
```
>MO 800 8CE 801
>PP
```

YOUR FILE IS STILL IN MEMORY, AND ONLY THE FIRST FEW
LINES OF TEXT HAVE BEEN LOST. THESE LOST LINES WILL
HAVE TO BE REENTERED.

==========

16.03 PRINTER DRIVER TO SEND ESCAPE SEQUENCES

THIS DRIVER PATCHES INTO THE WORD PROCESSOR ROMPAC PRINTER
DRIVER TO ALLOW ONE TO SEND ESCAPE SEQUENCES TO ACCESS A
PRINTER'S SPECIAL FEATURES SUCH AS FONT SIZES AND FORM FEED.

CREATE A TABLE OF ESCAPE SEQUENCES THAT YOUR PRINTER
UNDERSTANDS AT THE END OF THIS ROUTINE. THE TABLE IS
CONSTRUCTED WITH A MATCH LETTER FOLLOWED BY A FIVE BYTE ESCAPE
SEQUENCE. THUS, THE TABLE IS A MULTIPLE OF 6 BYTES IN LENGTH.

THIS DRIVER WATCHES FOR AN @ SIGN IN THE WORD PROCESSOR TEXT
TO SIGNAL THAT THE NEXT LETTER IS TO GENERATE SOME DESIRED
ESCAPE SEQUENCE. NEITHER THE @ SIGN, NOR THE FOLLOWING MATCH
LETTER WILL BE PRINTED. INSTEAD, THE ROUTINE WILL SEARCH THE
TABLE FOR THIS LETTER AND SEND THE 5 BYTE SEQUENCE THAT
FOLLOWS THE MATCH LETTER.

AFTER LOADING THIS ROUTINE, >GO 0 TO INITIALIZE THE ROUTINE.
THE ROUTINE WILL PATCH ITSELF INTO THE WORD PROCESSOR PRINTER
DRIVER. IN THE 'Y' TABLE, USE PRINTER ROUTINE #1.

```
;
0000: AF            INITZ   XOR  A
0001: 32 61 00              LD   (FLAG),A       ;CLEAR FLAG
0004: 21 0D 00              LD   HL,DRIVER
0007: 22 E7 07              LD   (07E7H),HL     ;WP VECTOR
000A: C3 FA DF              JP   0DFFAH         ;GO WARM START
;
```

```
;  THE FOLLOWING DRIVER NOW SUBSTITUTES FOR THE REGULAR WORD
;  PROCESSOR PRINTER ROUTINE.
;  WHEN THE @ SIGN IS SEEN, A FLAG IS SET TO INDICATE THAT
;  THE NEXT CHARACTER SHOULD SELECT A SEQUENCE FROM THE TABLE.
;
000D: F5            DRIVER  PUSH AF             ;SAVE REGISTERS
000E: C5                    PUSH BC
000F: E5                    PUSH HL
0010: 4F                    LD   C,A            ;SAVE CHARACTER
0011: 3A 61 00              LD   A,(FLAG)
0014: B7                    OR   A              ;IS FLAG SET
0015: 20 0A                 JR   NZ,CTRL-$      ;JP IF COMMAND
0017: 79                    LD   A,C            ;GET CHARACTER
0018: FE 40                 CP   040H           ;TEST @ SIGN
001A: 20 1C                 JR   NZ,PRINT-$     ;GO PRINT CHAR
001C: 32 61 00              LD   (FLAG),A       ;REMEMBER @
001F: 18 1A                 JR   EXIT-$         ;EXIT
;
;  AN @ SIGN WAS THE PREVIOUS CHARACTER.  TAKE THE PRESENT
;  CHARACTER AND SEARCH THE TABLE FOR A MATCH.  IF WE ARRIVE
;  AT THE BOTTOM OF THE TABLE WITHOUT A MATCH, THEN GO
;  AHEAD AND PRINT THE CHARACTER.  THIS ALLOWS THE @ SIGN
;  TO BE PRINTED FROM OUR TEXT BY USING TWO @ SIGNS TOGETHER.
;
0021: AF            CTRL    XOR  A              ;CLEAR FLAG
0022: 32 61 00              LD   (FLAG),A
0025: 21 62 00              LD   HL,BASE        ;TABLE BASE
0028: 7E            LOOP    LD   A,(HL)         ;GET LETTER
0029: B7                    OR   A              ;TEST 0 END
002A: 28 0B                 JR   Z,METOO-$      ;NOT IN TABLE
002C: B9                    CP   C              ;MATCH CHAR?
002D: 28 10                 JR   Z,FOUND-$      ;GO SEND SEQ.
002F: 23                    INC  HL             ;SKIP TO NEXT
0030: 23                    INC  HL             ; ENTRY IN
0031: 23                    INC  HL             ; TABLE WHEN
0032: 23                    INC  HL             ; THERE IS NO
0033: 23                    INC  HL             ; MATCH.
0034: 23                    INC  HL             ;SEQ IS 6 LONG
0035: 18 F1                 JR   LOOP-$
;
;  SEND THE CHARACTER TO THE WORD PROCESSOR PRINTER DRIVER.
;
0037: 79            METOO   LD   A,C            ;GET CHARACTER
0038: CD 90 DE      PRINT   CALL 0DE90H         ;WP SERIAL
003B: E1            EXIT    POP  HL
003C: C1                    POP  BC
003D: F1                    POP  AF             ;RESTORE REGS
003E: C9                    RET                 ;BACK TO WP
;
;  A SEQUENCE HAS BEEN LOCATED IN THE TABLE.  NOW SEND OUT THE
;  5 BYTES THAT FOLLOW IN THE TABLE.
;
005A: DB FD         LOOP3   IN   A,(0FDH)       ;WAIT FOR UART
005C: CB 47                 BIT  0,A
005E: 28 FA                 JR   Z,LOOP3-$
0060: C9                    RET
;
```

```
;
                        ESC     EQU  1BH            ;USEFUL EQUATES
                        CR      EQU  0DH
                        LF      EQU  0AH
                        BELL    EQU  07H
                        NULL    EQU  00H
;
0061: 00                FLAG    DEFB 0
;
0062: 45                BASE    DEFB 'E'            ;MATCH LETTER
0063: 1B                        DEFB ESC            ;ESC SEQUENCE
0064: 26                        DEFB ')'            ;  FIVE
0065: 6B                        DEFB 'k'            ;  BYTES
0066: 31                        DEFB '1'            ;  LONG
0067: 00                        DEFB NULL           ;FILLER NULL
;
0068: 00                        DEFB 0              ;0 MATCH ENDS
                                                    ;TABLE.
```

OTHER SEQUENCES FOR YOUR SERIAL PRINTER WOULD CONTINUE AT ADDRESS 0068H. JUST BE SURE TO MARK THE END OF THE TABLE WITH A 0 BYTE IN THE POSITION WHERE A MATCH LETTER WOULD GO. ALSO, PAD A SEQUENCE WITH 'NULL' FILLER CHARACTERS IF IT IS LESS THAN FIVE BYTES IN LENGTH AS IN THE ABOVE EXAMPLE.

```
EXAMPLE TEXT:    @E ENTER EXPANDED MODE.
                 @C ENTER COMPRESSED MODE.
                 @N RETURN TO NORMAL SIZED PRINT.
```

THE @E, @N, AND @C DO NOT PRINT WHEN THE FEATURE IS ACCESSED. THEY ARE SHOWN IN THE EXAMPLE TEXT AS IT WOULD APPEAR ON THE VIDEO DISPLAY.

16.03 CORRECTION

```
;  AN @ SIGN WAS THE PREVIOUS CHARACTER.  TAKE THE PRESENT
;  CHARACTER AND SEARCH THE TABLE FOR A MATCH.  IF WE ARRIVE
;  AT THE BOTTOM OF THE TABLE WITHOUT A MATCH, THEN GO
;  AHEAD AND PRINT THE CHARACTER.  THIS ALLOWS THE @ SIGN
;  TO BE PRINTED FROM OUR TEXT BY USING TWO @ SIGNS TOGETHER.
;
0021: AF              CTRL    XOR   A                 ;CLEAR FLAG
0022: 32 61 00                LD    (FLAG),A
0025: 21 62 00                LD    HL,BASE           ;TABLE BASE
0028: 7E              LOOP    LD    A,(HL)            ;GET LETTER
0029: B7                      OR    A                 ;TEST 0 END
002A: 28 0B                   JR    Z,METOO-$         ;NOT IN TABLE
002C: B9                      CP    C                 ;MATCH CHAR?
002D: 28 10                   JR    Z,FOUND-$         ;GO SEND SEQ.
002F: 23                      INC   HL                ;SKIP TO NEXT
0030: 23                      INC   HL                ; ENTRY IN
0031: 23                      INC   HL                ; TABLE WHEN
0032: 23                      INC   HL                ; THERE IS NO
0033: 23                      INC   HL                ; MATCH.
0034: 23                      INC   HL                ;SEQ IS 6 LONG
0035: 18 F1                   JR    LOOP-$
;
;  SEND THE CHARACTER TO THE WORD PROCESSOR PRINTER DRIVER.
;
0037: 79              METOO   LD    A,C               ;GET CHARACTER
0038: CD 90 DE        PRINT   CALL  0DE90H            ;WP SERIAL
003B: E1              EXIT    POP   HL
003C: C1                      POP   BC
003D: F1                      POP   AF                ;RESTORE REGS
003E: C9                      RET                     ;BACK TO WP
;
;  A SEQUENCE HAS BEEN LOCATED IN THE TABLE.  NOW SEND OUT THE
;  5 BYTES THAT FOLLOW IN THE TABLE.
;
003F: 06 05           FOUND   LD    B,005H            ;SEND 5 BYTES
0041: 23                      INC   HL                ;FIRST BYTE
0042: 7E              LOOP2   LD    A,(HL)
0043: CD 4B 00                CALL  SERIAL            ;SERIAL OUT
0046: 23                      INC   HL                ;NEXT BYTE
0047: 10 F9                   DJNZ  LOOP2-$           ;LOOP THRU 5
0049: 18 F0                   JR    EXIT-$            ;DONE NOW
;
;  A SERIAL PRINTER ROUTINE FOR 1200 BAUD OR 300 BAUD.
;  THIS ROUTINE IS USED TO SEND OUT THE SEQUENCE SO THAT
;  THE WORD PROCESSOR CANNOT HAVE CONTROL UNTIL WE ARE THROUGH.
;
004B: F5              SERIAL  PUSH AF
004C: FD 7E 3D                LD    A,(IY+03DH)       ;BAUD RATE
004F: F6 80                   OR    080H              ;RS232 ON
0051: D3 FE                   OUT   (0FEH),A
0053: FD 77 45                LD    (IY+045H),A       ;TAPE STATUS
0056: F1                      POP   AF
0057: CD 12 E0                CALL  LE012             ;SERIAL OUT
005A: DB FD           LOOP3   IN    A,(0FDH)          ;WAIT FOR UART
005C: CB 47                   BIT   0,A
005E: 28 FA                   JR    Z,LOOP3-$
0060: C9                      RET
;
```

CHAPTER 17 --- DEVELOPMENT PAC

17.01 PAUSE DEVELOPMENT PAC LISTINGS

```
0000: CD 15 C0    ENTRY CALL  QUICKCHECK
0003: 20 FB             JR    NZ,ENTRY-$
0005: 7A                LD    A,D
0006: CD 1B E0          CALL  VIDEO
0009: FE 0A             CP    LINEFEED
000B: C0                RET   NZ
000C: C3 1B E0          JP    VIDEO
```

UNDER DDT80, CHANGE THE :SO AND :CO VECTORS TO 0000 HEX.
NOW, HOLDING DOWN THE RUN/STOP KEY WILL PAUSE A LISTING.

==========

17.02 MEMORY PARTITIONS FOR 32K CONFIGURATION.

#1 - I/O AND STACK	7D00 - 7FFF
#2 - PROGRAM SOURCE, 'B' BUFFER	3E80 - 7CFF
#3 - ASSEMBLED CODE, 'A' BUFFER	1F40 - 3E7F
#4 - USER LOAD AREA	013A - 1F3F
#5 - ASSEMBLER'S RAM SPACE	0100 - 0139
#6 - USER LOAD AREA	0000 - 00FF

==========

17.03 I/O VECTOR ASSIGNMENTS:

VECTOR	EDITOR	ASSEMBLER	CASSETTE	
:CI	:SK	:SK	:SK	CHANNEL
:CO	:SV	:SV	:SV	
:OI	:AO	:AO	:I1	ASSEMBLER
:OO	:AI	:AI	:O1	
:SI	:BI	:BO	:I1	SOURCE
:SO	:SV	:SV	:O1	

17.04 SAMPLE COMMANDS

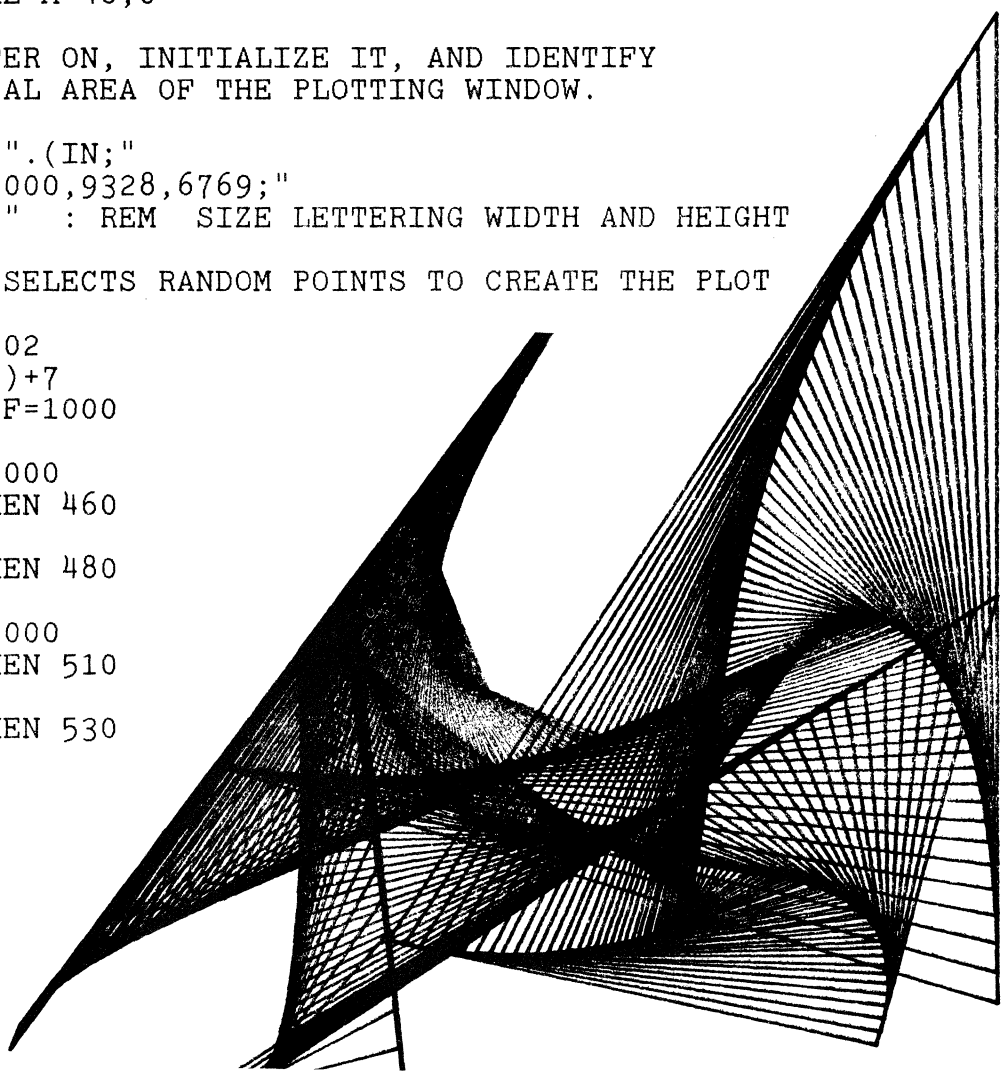
```
.M :SI
:SI  :BI  :BO      ;CHANGE FROM EDITOR TO ASSEMBLER
                   ;CHANGE BACK TO :BI FOR EDITOR

.E E003            ;EXIT TO MONITOR
.E :AS             ;EXIT TO ASSEMBLER

.E :ED             ;ENTER EDITOR TO START NEW FILE
.E :ER             ;REENTER EDITOR TO MODIFY A FILE

.L 0,200           ;CALL LOADER WITH 0 OFFSET TO ORG
                   ;BUILD SYMBOL TABLE AT 200H

LABEL-$            ;RELATIVE ADDRESSING USES -$
                   ;LABELS DO NOT HAVE A ':'
```

CHAPTER 18 --- PLOTTING

18.01 BEAUTIFUL BIRTHDAY PLOTS

THE PROGRAM LISTING USES AN HP7225A PLOTTER WHICH IS VERY INTELLIGENT. HOWEVER, YOU CAN SUBSTITUTE EQUIVALENT PLOTTER CONTROL COMMANDS FOR YOUR PLOTTER

```
100 REM  ---      BIRTHDAY PLOTS      ---
110 :
120 CLEAR 150:DIM R(18,18)
130 :
140 REM  POKE AN RS232 DRIVER IN MEMORY AND USE IT TO SEND
150 REM  PLOTTER CONTROL SEQUENCES VIA PRINT STRINGS.
160 :
170 FOR J=0 TO 14:READ I:POKE J,I:NEXT J
180 DATA 245,245,62,128,211,254,241,205,18
190 DATA 224,205,27,224,241,201
200 :
210 INPUT "NAME";N$
220 INPUT "RANDOM NUMBER";A:J=RND(-A*2-1)
230 :
240 REM  FIND MONITOR WORK AREA AND CHANGE OUTPUT VECTOR
250 REM  TO POINT TO THE RS232 DRIVER ROUTINE.
260 :
270 M=256*PEEK(-4095)+PEEK(-4096)-65536
280 POKE M-47,0:POKE M-46,0
290 :
300 REM  TURN PLOTTER ON, INITIALIZE IT, AND IDENTIFY
310 REM  THE PHYSICAL AREA OF THE PLOTTING WINDOW.
320 :
330 PRINT CHR$(27)+".(IN;"
340 PRINT "IP1328,1000,9328,6769;"
350 PRINT "SI.3,.4;"  : REM  SIZE LETTERING WIDTH AND HEIGHT
360 :
370 REM  THIS LOOP SELECTS RANDOM POINTS TO CREATE THE PLOT
380 :
390 B=RND(1)*.1 + .02
400 C=INT(RND(1)*10)+7
410 E=0:D=0:G=1000:F=1000
420 FOR H=1TOC
430 R(H,1)=RND(1)*1000
440 IF R(H,1)<=D THEN 460
450 D=R(H,1)
460 IF R(H,1)>=F THEN 480
470 F=R(H,1)
480 R(H,2)=RND(1)*1000
490 IF R(H,2)<=E THEN 510
500 E=R(H,2)
510 IF R(H,2)>=G THEN 530
520 G=R(H,2)
530 NEXT H
540 :
```



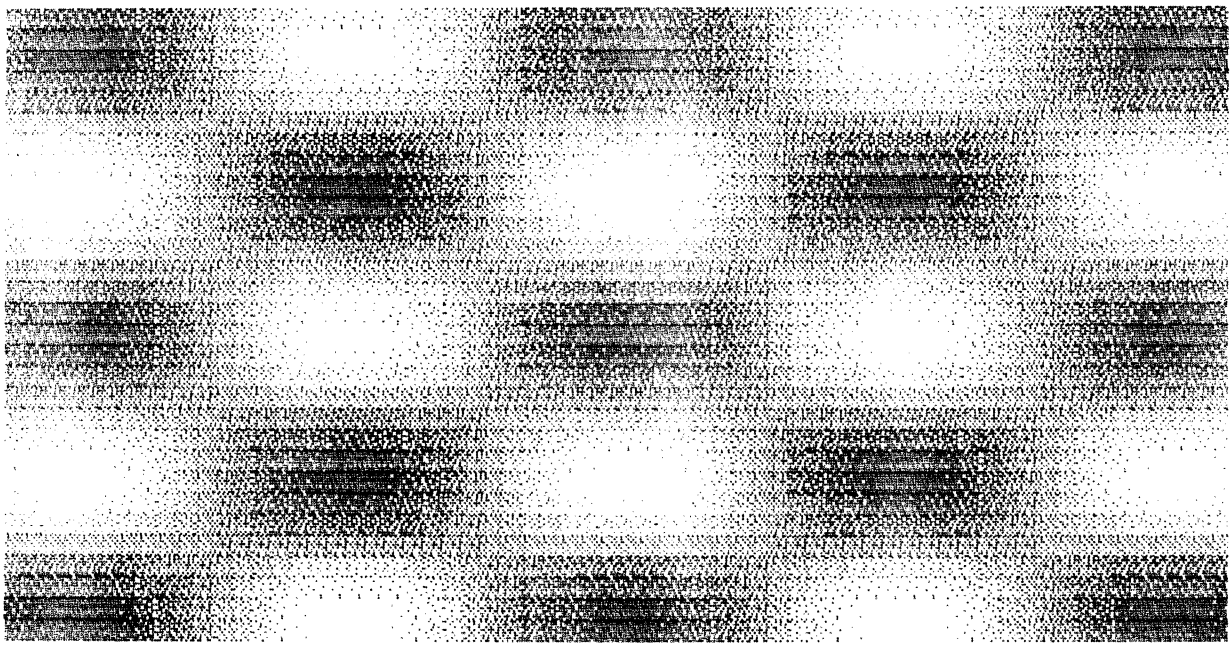
```
550 REM  NOW MAP THE LOGICAL WINDOW TO THE PLOTTING SURFACE
560 :
570 PRINT "SC";F;D;G-.1*(E-G);E;";"
580 :
590 REM  PLOT THE BIRTHDAY PLOT
600 :
610 PRINT "PA";R(1,1);R(1,2);";PD;"
620 FOR H=1TO50
630 R(C+1,1)=R(1,1)
640 R(C+1,2)=R(1,2)
650 FOR I=1TOC+1
660 PRINT "PA";INT(R(I,1));INT(R(I,2));";"
670 IF I>C THEN 700
680 R(I,1)=B*(R(I+1,1)-R(I,1))+R(I,1)
690 R(I,2)=B*(R(I+1,2)-R(I,2))+R(I,2)
700 NEXT I
710 NEXT H
720 :
730 REM  LABEL THE PLOT WITH THE NAME TEXT
740 :
750 PRINT "PU;PA";(D-F)*.5;G-.1*(E-G)";"
760 A$=MID$(STR$(A),2)+"   "+N$
770 R=LEN(A$)/2
780 PRINT "CP";R;-1;";"
790 PRINT "LB#";A$;CHR$(3)
800 :
810 REM  TURN PLOTTER OFF, AND RESTORE VECTOR TO >SE O=V
820 :
830 PRINT "IN;";CHR$(27)+".)"
840 POKE M-47,27:POKE M-46,224
850 END
```

SUMMARY OF 7225A PLOTTER CONTROLS USED IN THE EXAMPLE:

```
IPx1,y1,x2,y2  - WINDOW CORNER POINTS ON PLOTTER SURFACE
SCx1,y1,x2,y2  - SCALE LOGICAL CORNER POINTS TO PHYSICAL WINDOW
PAx,y          - MOVE PEN TO ABSOLUTE LOGICAL POINT OF (x,y)
PU             - PEN UP
PD             - PEN DOWN
CPx,y          - MOVE RELATIVE x CHARACTER WIDTHS, y HEIGHTS
LB             - LETTER TEXT STRING WHICH FOLLOWS
SIx,y          - LETTER WIDTH x, HEIGHT y IN CENTIMETERS
```

18.02 3-D FUNCTION PLOTS USING SHADED PRINT DENSITY

```
100 REM  ---      PATTERNS      ---
110 :
120 REM  PUT FUNCTION TO BE GRAPHED IN  FNZ()=  ON LINE 140
130 :
140 DEF FNZ(X)=COS(X)*COS(Y)          :REM  SAMPLE 3-D FUNCTION
150 :
160 L=18:GOSUB 5000
170 INPUT "DOMAIN OF X-AXIS ";X1,X2 :REM TRY -7,7
180 U=(X2-X1)/64
190 INPUT "DOMAIN OF Y-AXIS ";Y1,Y2 :REM TRY -7,7
200 W=-(Y1-Y2)/30
210 INPUT "RANGE OF FUNCTION";R1,R2 :REM TRY -1.01,1.01
220 L=L/(R2-R1):PRINT CHR$(12):Y=Y1
230 FOR J=1 TO 30:X=X1:FOR I=1 TO 64
240 P=INT(L*(FNZ(X)-R1))+192 : POKE -4033+I+J*64,P
250 X=X+U : NEXT I : Y=Y+W : NEXT J
260 GOTO 260 : REM ENDLESS LOOP SO DISPLAY IS NOT RUINED
270 END
5000 REM  ---      DEFINE GRAPHIC SYMBOLS      ---
5010 FOR I=1 TO L*8: READ A: POKE -513+I,A: NEXT I: RETURN
6000 DATA 0,0,0,0,0,0,0,0
6010 DATA 0,0,32,0,0,0,2,0
6020 DATA 0,32,0,4,64,0,8,0
6030 DATA 4,64,16,1,8,128,2,32
6040 DATA 65,16,20,33,132,16,66,8
6050 DATA 146,32,9,128,17,138,16,69
6060 DATA 145,74,17,68,137,18,132,82
6070 DATA 41,138,98,41,146,41,196,41
6080 DATA 85,74,101,146,73,178,74,149
6090 DATA 85,42,83,186,86,170,165,102
6100 DATA 85,229,85,174,85,202,85,171
6110 DATA 93,234,87,186,213,174,117,171
6120 DATA 109,222,245,107,218,183,247,251
6130 DATA 237,119,221,119,237,190,111,219
6140 DATA 190,238,251,222,123,238,190,247
6150 DATA 251,191,238,254,247,127,253,223
6160 DATA 255,223,255,254,255,191,255,251
6170 DATA 255,255,255,255,255,255,255,255
```



CHAPTER 19 -- TABLES AND FORMS

19.01 BASIC'S TOKENS

HEX	-8-	-9-	-A-	-B-	-C-
0	END	STOP	FN	INT	STR$
1	FOR	OUT	SPC	ABS	VAL
2	NEXT	ON	THEN	USR	ASC
3	DATA	NULL	NOT	FRE	CHR$
4	BYE	WAIT	STEP	INP	LEFT$
5	INPUT	DEF	+	POS	RIGHT$
6	DIM	POKE	-	SQR	MID$
7	READ	PRINT	*	RND	
8	LET	CONT	/	LOG	
9	GOTO	LIST	^	EXP	
A	RUN	CLEAR	AND	COS	
B	IF	CLOAD	OR	SIN	
C	RESTORE	CSAVE	>	TAN	
D	GOSUB	NEW	=	ATN	
E	RETURN	TAB	<	PEEK	
F	REM	TO	SGN	LEN	

EXAMPLE: 8F - REM BE - PEEK C6 - MID$

==========

19.02 HEXADECIMAL - BINARY CONVERSION TABLE

0 - 0000	8 - 1000
1 - 0001	9 - 1001
2 - 0010	A - 1010
3 - 0011	B - 1011
4 - 0100	C - 1100
5 - 0101	D - 1101
6 - 0110	E - 1110
7 - 0111	F - 1111

==========

19.03 POWERS OF 2 TABLE: 2 ^ N

0 - 1	9 - 512
1 - 2	10 - 1024
2 - 4	11 - 2048
3 - 8	12 - 4096
4 - 16	13 - 8192
5 - 32	14 - 16384
6 - 64	15 - 32768
7 - 128	16 - 65536
8 - 256	17 -131072

19.04 DECIMAL - HEXADECIMAL CONVERSION TABLE

0 - 00	48 - 30	96 - 60	144 - 90	192 - C0	240 - F0
1 - 01	49 - 31	97 - 61	145 - 91	193 - C1	241 - F1
2 - 02	50 - 32	98 - 62	146 - 92	194 - C2	242 - F2
3 - 03	51 - 33	99 - 63	147 - 93	195 - C3	243 - F3
4 - 04	52 - 34	100 - 64	148 - 94	196 - C4	244 - F4
5 - 05	53 - 35	101 - 65	149 - 95	197 - C5	245 - F5
6 - 06	54 - 36	102 - 66	150 - 96	198 - C6	246 - F6
7 - 07	55 - 37	103 - 67	151 - 97	199 - C7	247 - F7
8 - 08	56 - 38	104 - 68	152 - 98	200 - C8	248 - F8
9 - 09	57 - 39	105 - 69	153 - 99	201 - C9	249 - F9
10 - 0A	58 - 3A	106 - 6A	154 - 9A	202 - CA	250 - FA
11 - 0B	59 - 3B	107 - 6B	155 - 9B	203 - CB	251 - FB
12 - 0C	60 - 3C	108 - 6C	156 - 9C	204 - CC	252 - FC
13 - 0D	61 - 3D	109 - 6D	157 - 9D	205 - CD	253 - FD
14 - 0E	62 - 3E	110 - 6E	158 - 9E	206 - CE	254 - FE
15 - 0F	63 - 3F	111 - 6F	159 - 9F	207 - CF	255 - FF
16 - 10	64 - 40	112 - 70	160 - A0	208 - D0	
17 - 11	65 - 41	113 - 71	161 - A1	209 - D1	
18 - 12	66 - 42	114 - 72	162 - A2	210 - D2	
19 - 13	67 - 43	115 - 73	163 - A3	211 - D3	
20 - 14	68 - 44	116 - 74	164 - A4	212 - D4	
21 - 15	69 - 45	117 - 75	165 - A5	213 - D5	
22 - 16	70 - 46	118 - 76	166 - A6	214 - D6	
23 - 17	71 - 47	119 - 77	167 - A7	215 - D7	
24 - 18	72 - 48	120 - 78	168 - A8	216 - D8	
25 - 19	73 - 49	121 - 79	169 - A9	217 - D9	
26 - 1A	74 - 4A	122 - 7A	170 - AA	218 - DA	
27 - 1B	75 - 4B	123 - 7B	171 - AB	219 - DB	
28 - 1C	76 - 4C	124 - 7C	172 - AC	220 - DC	
29 - 1D	77 - 4D	125 - 7D	173 - AD	221 - DD	
30 - 1E	78 - 4E	126 - 7E	174 - AE	222 - DE	
31 - 1F	79 - 4F	127 - 7F	175 - AF	223 - DF	
32 - 20	80 - 50	128 - 80	176 - B0	224 - E0	
33 - 21	81 - 51	129 - 81	177 - B1	225 - E1	
34 - 22	82 - 52	130 - 82	178 - B2	226 - E2	
35 - 23	83 - 53	131 - 83	179 - B3	227 - E3	
36 - 24	84 - 54	132 - 84	180 - B4	228 - E4	
37 - 25	85 - 55	133 - 85	181 - B5	229 - E5	
38 - 26	86 - 56	134 - 86	182 - B6	230 - E6	
39 - 27	87 - 57	135 - 87	183 - B7	231 - E7	
40 - 28	88 - 58	136 - 88	184 - B8	232 - E8	
41 - 29	89 - 59	137 - 89	185 - B9	233 - E9	
42 - 2A	90 - 5A	138 - 8A	186 - BA	234 - EA	
43 - 2B	91 - 5B	139 - 8B	187 - BB	235 - EB	
44 - 2C	92 - 5C	140 - 8C	188 - BC	236 - EC	
45 - 2D	93 - 5D	141 - 8D	189 - BD	237 - ED	
46 - 2E	94 - 5E	142 - 8E	190 - BE	238 - EE	
47 - 2F	95 - 5F	143 - 8F	191 - BF	239 - EF	

19.05 CONTROL CODES AND THEIR FUNCTIONS

HEX	DEC	CONTROL	SYMBOL	FUNCTION
00	0	@	NULL	
01	1	A	SOH	CURSOR LEFT
02	2	B	STX	
03	3	C	ETX	
04	4	D	EOT	
05	5	E	ENQ	
06	6	F	ACK	
07	7	G	BELL	
08	8	H	BS	RUBOUT
09	9	I	HT	
0A	10	J	LF	LINE FEED
0B	11	K	VT	
0C	12	L	FF	CLEAR SCREEN
0D	13	<CR>	CR	RETURN
0E	14	N	SO	
0F	15	O	SI	STOP INPUT
10	16	P	DLE	
11	17	Q	DC1	HOME
12	18	R	DC2	
13	19	S	DC3	CURSOR RIGHT
14	20	T	DC4	
15	21	U	NAK	
16	22	V	SYN	
17	23	W	ETB	CURSOR UP
18	24	X	CAN	
19	25	Y	EM	
1A	26	Z	SUB	CURSOR DOWN
1B	27	[	ESC	ESCAPE
1C	28	\	FS	
1D	29	]	GS	
1E	30	^	RS	
1F	31	-	US	

FUNCTIONS ACCESSED USING CHR$().

```
EXAMPLE:   100 PRINT CHR$(12);:REM CLEARS SCREEN
```

19.06 GRAPHIC CHARACTER DESIGNER'S FORM

00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
----	----	----	----
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
----	----	----	----
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
----	----	----	----
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
----	----	----	----

19.07 MACHINE LANGUAGE CODING FORM

ROUTINE NAME: PAGE OF

DESCRIPTION: DATE

ADDRESS	OBJECT CODE	LABEL	INSTRUCTION	REMARKS

19.08 SERIAL INTERFACE PINOUTS

PIN#	SIGNAL	PIN#	SIGNAL
1	SHIELD 1	14	SHIELD 2
2	RS232 OUT	15	MIC 1
3	RS232 IN	16	MIC 2
4	GROUND	17	GROUND
5	AUX 1	18	AUX 2
6	GROUND	19	GROUND
7	GROUND	20	EAR 1
8	GROUND	21	EAR 2
9	+12 VOLTS	22	UNUSED
10	UNUSED	23	RS232 OUT
11	RS232 IN	24	MOTOR #1-
12	MOTOR #1+	25	MOTOR #2-
13	MOTOR #2+		

==========

19.09 PARALLEL INTERFACE PINOUTS

PIN#	SIGNAL	PIN#	SIGNAL
1	GROUND	14	UNUSED
2	OUTPUT DATA ACCEPTED	15	+5 VOLTS
3	OUTPUT DATA AVAILABLE	16	OUTPUT BIT 0
4	OUTPUT BIT 7	17	OUTPUT BIT 1
5	OUTPUT BIT 6	18	OUTPUT BIT 2
6	OUTPUT BIT 5	19	OUTPUT BIT 3
7	OUTPUT BIT 4	20	+5 VOLTS
8	GROUND	21	INPUT DATA ACCEPTED
9	INPUT DATA AVAILABLE	22	INPUT BIT 1
10	INPUT BIT 0	23	INPUT BIT 3
11	INPUT BIT 2	24	INPUT BIT 5
12	INPUT BIT 4	25	INPUT BIT 7
13	INPUT BIT 6		

19.10 DATA PORT ASSIGNMENTS

PORT	BIT	INPUT FUNCTION	OUTPUT FUNCTION
0FCH	0	|	|
	1	|	|
252	2	| SERIAL	| SERIAL
	3	| INPUT	| OUTPUT
	4	| DATA	| DATA
	5	| BYTE	| BYTE
	6	|	|
	7	|	|
0FDH	0	SERIAL OUT BUFFER EMPTY	BITS PER CHARACTER NB1
	1	SERIAL DATA AVAILABLE	BITS PER CHARACTER NB2
253	2	OVER-RUN	NUMBER OF STOP BITS
	3	FRAMING ERROR	PARITY SELECT
	4	PARITY ERROR	NO PARITY
	5	UNUSED	UNUSED
	6	UNUSED	UNUSED
	7	UNUSED	UNUSED
0FEH	0	|	|
	1	| KEYBOARD	| KEYBOARD
254	2	| DATA	| SCAN COLUMN
	3	| BYTE (5 BITS)	|
	4	|	MOTOR #1 CONTROL
	5	HORIZONTAL SYNC PULSE	MOTOR #2 CONTROL
	6	PARALLEL OUTPUT READ	BAUD RATE SELECT
	7	PARALLEL INPUT AVAILABLE	0=CASSETTE 1=RS232
0FFH	0	|	|
	1	|	|
255	2	| PARALLEL	| PARALLEL
	3	| INPUT	| OUTPUT
	4	| DATA	| DATA
	5	| BYTE	| BYTE
	6	|	|
	7	|	|

19.11 ASCII CHARACTER SET

MSB / LSB	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENG	NAK	%	5	E	U	e	u
6	ACK	SYN	AMP	6	F	V	f	v
7	BELL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	VS	/	?	O	UNDER	o	RUB

MSB / LSB	8	9	A	B	C	D	E	F
0	GR 1	GR E	GR J	GR /	GS 1	GS E	GS J	GS /
1	GR 2	GR R	GR K	GR K-	GS 2	GS R	GS K	GS K-
2	GR 3	GR T	GR L	GR K7	GS 3	GS T	GS L	GS K7
3	GR 4	GR Y	GR ;	GR K8	GS 4	GS Y	GS ;	GS K8
4	GR 5	GR U	GR @	GR K9	GS 5	GS U	GS @	GS K9
5	GR 6	GR I	GR \	GR K/	GS 6	GS I	GS \	GS K/
6	GR 7	GR O	GR RUB	GR K4	GS 7	GS O	GS RUB	GS K4
7	GR 8	GR P	GR Z	GR K6	GS 8	GS P	GS Z	GS K6
8	GR 9	GR [	GR X	GR K*	GS 9	GS [	GS X	GS K*
9	GR 0	GR]	GR C	GR K1	GS 0	GS]	GS C	GS K1
A	GR :	GR A	GR V	GR K2	GS :	GS A	GS V	GS K2
B	GR -	GR S	GR B	GR K3	GS -	GS S	GS B	GS K3
C	GR ^	GR D	GR N	GR K+	GS ^	GS D	GS N	GS K+
D	GR TAB	GR F	GR M	GR K0	GS TAB	GS F	GS M	GS K0
E	GR Q	GR G	GR ,	GR K.	GS Q	GS G	GS ,	GS K.
F	GR W	GR H	GR .	GR K=	GS W	GS H	GS .	GS K=

KEY: GR = GRAPHIC GS = GRAPHIC AND SHIFT K1 = KEYPAD 1
MSB= MOST SIGNIFICANT BYTE LSB= LEAST SIGNIFICANT BYTE

EXAMPLE: GRAPHIC SHIFT TAB = CD HEX (205 DECIMAL)

19.12 KEYBOARD STRUCTURE: OUT (0FEH),nn AND IN A,(0FEH)

THE KEY IS DEPRESSED WHEN THE BIT IS A LOGICAL LOW.

OUT nn	BIT 0	BIT 1	BIT 2	BIT 3	BIT 4
0	STOP	GRAPHIC	CTRL	SHFTLOCK	SHIFT
1	CLEAR	REPEAT	SPACE	SKIP	ESC
2	X	Z	A	Q	1
3	C	D	S	W	2
4	F	R	E	4	3
5	B	V	G	T	5
6	M	N	H	Y	6
7	K	I	J	U	7
8	,	L	O	9	8
9	/	.	;	P	0
10	\	@	]	[	:
11	UNDERLINE	RETURN	LINEFEED	^	-
12	KEYPAD +	KEYPAD *	KEYPAD /	KEYPAD -	NOT USED
13	KEYPAD 0	KEYPAD 1	KEYPAD 4	KEYPAD 8	KEYPAD 7
14	KEYPAD .	KEYPAD 2	KEYPAD 5	KEYPAD 6	KEYPAD 9
15	NOT USED	NOT USED	NOT USED	KEYPAD =	KEYPAD 3

==========

19.13 GRAPHIC CHARACTER STARTING ADDRESSES

ASC	KEY	HEX	DEC	ASC	KEY	HEX	DEC
128	GR 1	FC00	-1024	192	GS 1	FE00	-512
129	GR 2	FC08	-1016	193	GS 2	FE08	-504
130	GR 3	FC10	-1008	194	GS 3	FE10	-496
131	GR 4	FC18	-1000	195	GS 4	FE18	-488
132	GR 5	FC20	-992	196	GS 5	FE20	-480
133	GR 6	FC28	-984	197	GS 6	FE28	-472
134	GR 7	FC30	-976	198	GS 7	FE30	-464
135	GR 8	FC38	-968	199	GS 8	FE38	-456
136	GR 9	FC40	-960	200	GS 9	FE40	-448
137	GR 0	FC48	-952	201	GS 0	FE48	-440
138	GR :	FC50	-944	202	GS :	FE50	-432
139	GR -	FC58	-936	203	GS -	FE58	-424
140	GR ^	FC60	-928	204	GS ^	FE60	-416
141	GR TAB	FC68	-920	205	GS TAB	FE68	-408
142	GR Q	FC70	-912	206	GS Q	FE70	-400
143	GR W	FC78	-904	207	GS W	FE78	-392

144	GR E	FC80	-896	208	GS E	FE80	-384
145	GR R	FC88	-888	209	GS R	FE88	-376
146	GR T	FC90	-880	210	GS T	FE90	-368
147	GR Y	FC98	-872	211	GS Y	FE98	-360
148	GR U	FCA0	-864	212	GS U	FEA0	-352
149	GR I	FCA8	-856	213	GS I	FEA8	-344
150	GR O	FCB0	-848	214	GS O	FEB0	-336
151	GR P	FCB8	-840	215	GS P	FEB8	-328
152	GR [	FCC0	-832	216	GS [	FEC0	-320
153	GR]	FCC8	-824	217	GS]	FEC8	-312
154	GR A	FCD0	-816	218	GS A	FED0	-304
155	GR S	FCD8	-808	219	GS S	FED8	-296
156	GR D	FCE0	-800	220	GS D	FEE0	-288
157	GR F	FCE8	-792	221	GS F	FEE8	-280
158	GR G	FCF0	-784	222	GS G	FEF0	-272
159	GR H	FCF8	-776	223	GS H	FEF8	-264
160	GR J	FD00	-768	224	GS J	FF00	-252
161	GR K	FD08	-760	225	GS K	FF08	-244
162	GR L	FD10	-752	226	GS L	FF10	-236
163	GR ;	FD18	-744	227	GS ;	FF18	-228
164	GR @	FD20	-736	228	GS @	FF20	-220
165	GR \	FD28	-728	229	GS \	FF28	-212
166	GR RUB	FD30	-720	230	GS RUB	FF30	-204
167	GR Z	FD38	-712	231	GS Z	FF38	-196
168	GR X	FD40	-704	232	GS X	FF40	-188
169	GR C	FD48	-696	233	GS C	FF48	-180
170	GR V	FD50	-688	234	GS V	FF50	-172
171	GR B	FD58	-680	235	GS B	FF58	-168
172	GR N	FD60	-672	236	GS N	FF60	-160
173	GR M	FD68	-664	237	GS M	FF68	-152
174	GR ,	FD70	-656	238	GS ,	FF70	-144
175	GR .	FD78	-648	239	GS .	FF78	-136
176	GR /	FD80	-640	240	GS /	FF80	-128
177	GR K-	FD88	-632	241	GS K-	FF88	-120
178	GR K7	FD90	-624	242	GS K7	FF90	-112
179	GR K8	FD98	-616	243	GS K8	FF98	-104
180	GR K9	FDA0	-608	244	GS K9	FFA0	-96
181	GR K/	FDA8	-600	245	GS K/	FFA8	-88
182	GR K4	FDB0	-592	246	GS K4	FFB0	-80
183	GR K6	FDB8	-584	247	GS K6	FFB8	-72
184	GR K*	FDC0	-576	248	GS K*	FFC0	-64
185	GR K1	FDC8	-568	249	GS K1	FFC8	-56
186	GR K2	FDD0	-560	250	GS K2	FFD0	-48
187	GR K3	FDD8	-552	251	GS K3	FFD8	-40
188	GR K+	FDE0	-544	252	GS K+	FFE0	-32
189	GR K0	FDE8	-536	253	GS K0	FFE8	-24
190	GR K.	FDF0	-528	254	GS K.	FFF0	-16
191	GR K=	FDF8	-520	255	GS K=	FFF8	-8

KEY: GR = GRAPHIC GS = GRAPHIC AND SHIFT K1 = KEYPAD 1

19.14 ONE-STROKE BASIC COMMANDS

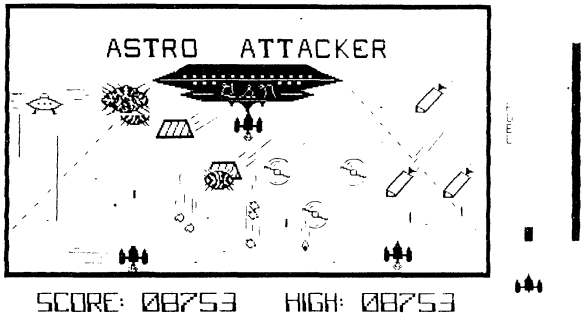
ASC	KEYS	COMMAND	ASC	KEYS	COMMAND
128	GR 1	END	192	GS 1	STR$
129	GR 2	FOR	193	GS 2	VAL
130	GR 3	NEXT	194	GS 3	ASC
131	GR 4	DATA	195	GS 4	CHR$
132	GR 5	BYE 0	196	GS 5	LEFT$
133	GR 6	INPUT	197	GS 6	RIGHT$
134	GR 7	DIM	198	GS 7	MID$
135	GR 8	READ			
136	GR 9	LET			
137	GR 0	GOTO			
138	GR :	RUN			
139	GR -	IF			
140	GR ^	RESTORE			
141	GR TAB	GOSUB			
142	GR Q	RETURN			
143	GR W	REM			
144	GR E	STOP			
145	GR R	OUT			
146	GR T	ON			
147	GR Y	NULL			
148	GR U	WAIT			
149	GR I	DEF			
150	GR O	POKE			
151	GR P	PRINT			
152	GR [	CONT			
153	GR]	LIST			
154	GR A	CLEAR			
155	GR S	CLOAD			
156	GR D	CSAVE			
157	GR F	NEW			
158	GR G	TAB(			
159	GR H	TO			
160	GR J	FN	176	GR /	INT
161	GR K	SPC(	177	GR K-	ABS
162	GR L	THEN	178	GR K7	USR
163	GR ;	NOT	179	GR K8	FRE
164	GR @	STEP	180	GR K9	INP
165	GR \	+	181	GR K/	POS
166	GR RUB	-	182	GR K4	SQR
167	GR Z	.	183	GR K6	RND
168	GR X	/	184	GR K*	LOG
169	GR C	^	185	GR K1	EXP
170	GR V	AND	186	GR K2	COS
171	GR B	OR	187	GR K3	SIN
172	GR N	>	188	GR K+	TAN
173	GR M	=	189	GR K0	ATN
174	GR ,	<	190	GR K.	PEEK
175	GR .	SGN	191	GR K=	LEN

KEY: GR = GRAPHIC KEY GS = GRAPHIC AND SHIFT KEYS

ARRINGTON SOFTWARE SERVICE
9522 Linstock, Boise, Idaho 83704 U.S.A.

ASTRO ATTACKER $21.95

is similar to the arcade game called "ASTRO BLASTER". This action game for the Sorcerer is far superior to all other Sorcerer games because of its high resolution graphics, sound, variety and playability. Astro Attacker graphics are extremely advanced. The display is of the console inside your astro fighter craft. In your console window you see the enemy ships placed against a background of continuously moving stars. Gauges also indicate the amount of fuel remaining and the temperature of your lazer cannons. If you fire too frequently you can overheat the lazers, or if you move recklessly you may run out of fuel. Your challenge is to survive and destroy the Spinners, the Lazer Ships, the Rockets, the Flame Throwers, and the Meteor shower. Docking with the mother ship is crucial to survival as this restores your shield strength and fuel, and cools your lazer cannon. With each succeeding level of play, survival becomes more difficult as the enemy ships attack with greater frequency and quickness. Superb sound too.



Please let us know if you should not be receiving this catalog so we can remove your name from our mailing list.

LIMITED WARRANTY

SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. IN NO EVENT WILL ENSIGN SOFTWARE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE ANY OF OUR PRODUCTS.

YOU ASSUME RESPONSIBILITY FOR THE SELECTION OF THE PRODUCTS TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION, REPAIR, USE AND RESULTS OBTAINED FROM THE PRODUCTS.

HOW TO ORDER:

ALL PRICES ARE IN U.S. DOLLARS

PRODUCT	PRICE
OVERSEAS POSTAGE: ADD $3 NO POSTAGE CHARGE IN USA & CANADA	
VISA MasterCard TOTAL	

ENCLOSE:

(A) A CHECK IN U.S. DOLLARS DRAWN ON A U.S. BANK, MONEY ORDER OR CASH FOR THE ABOVE AMOUNT, OR

(B) CREDIT CARD, EXPIRATION DATE_____

NUMBER:.....

My name and address:

NAME.....

STREET:.....

CITY/STATE.....

POSTCODE.....COUNTRY:.....

PLEASE DESCRIBE YOUR COMPUTER CONFIGURATION

MAIL YOUR ORDER TO:

ENSIGN SOFTWARE
2312 N. COLE RD., SUITE E
BOISE, IDAHO 83704 U.S.A.

PHONE: (208) 378-8086

AUTHORS: We seek excellent programs to market worldwide. Our royalties are generous, and our interest in having satisfied customers is keen. We invite you to join with us and our distributors to bring your product into the marketplace. Submit programs for evaluation along with documentation and your phone number.

ENSIGN SOFTWARE CATALOGUE

Please send the addresses of others who would like to receive this catalogue.

THE NEW ARRINGTON SOFTWARE SERVICE

HIGH QUALITY SOFTWARE AT A LOW PRICE

ENSIGN SOFTWARE

2312 N. Cole Rd., Suite E, Boise, Idaho 83704 U.S.A.