ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 2

*Trademark of Tandy Corp.

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 2

wayne GREEN inc

PETERBOROUGH NH 03458

FIRST EDITION
FIRST PRINTING SEPTEMBER 1981

Edited by Kate Comiskey, Chris Crocker,
Nan McCarthy, Katherine Putnam
Technical Assistance by Jake Commander,
Dennis Thurlow
Illustrations by Howard Happ
Typeset by Karen Stewart

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia—a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

CONTENTS

CONTENTS

BUSINESS

The Name and Address File

Expense Report

The Name and Address File

by Hubert C. Borrmann K0MVB

This is a program written for Level II, 16K and consists of the main program, written in BASIC, and a machine-language routine with two entry points. The machine-language routine (assembled in Program Listing 2) is POKEd into upper memory which must be reserved. Answer MEMORY SIZE? with 32607. The name and address file is saved on cassette tape and loaded into memory whenever you want to work with it. The 10 fields of my record are:

1) last name
2) first name and initial
3) street address
4) city or town
5) state
6) zip code
7) telephone
8) note 1
9) note 2
10) remarks

Why do I use machine language at all? It is because the TRS-80 does not really understand BASIC, only machine language, and therefore an interpreter is needed to interpret the BASIC statements and issue the proper instructions in machine language. This takes time, since each BASIC line has to be reinterpreted again and again, as long as the program is running. By using the machine language routine, the program executes a lot faster.

The operation of the program is controlled by a menu and has the following selections:

L—load file from tape
W—write file on tape
A—add record to file (in memory)
P—print (and change) records
S—sort the file in memory
E—end the program

The first thing to do before you can work with your file is to load it. Once the data is in memory, you can perform the other operations. If you don't have an existing file, start one with the A selection (add records). If you want to print a record on the screen, type P and, when prompted, type in the last name of the record you want to see. (The search will be made by comparing as many characters as you typed in. If you typed in, for instance, SM, you will get all the Smalls, Smiths, Smythes, etc.) You do all this in BASIC. Once you have found the proper record (all records are in the subscripted variable C$), the memory location is obtained with the VARPTR instruction. (See line 920 of Program Listing 1.) This address is passed to the machine language entry point #1 in variable L1 via the USR (User Service Routine). See line 940.

Let's now go to the machine-language listing (Program Listing 2), entry point #1 (line 20). The first thing to do is CALL 0A7FH. This puts the memory address out of L1 into the register pair HL. The program then proceeds to move the selected record, one character at a time, onto 10 lines of the video screen. The individual fields of each record have a delimiter (!), and this tells the program when to start the next line. This action, being in machine language, is extremely fast. When the record is built, a return in line 210 transfers control back to the BASIC program.

You now have the option either to return to the menu, change the record you have displayed, or look for another record with the same selection criteria. To find the next record, type N. The program now looks for the next record and, if found, gets its memory address and again has machine language display it. If, however, you want to change the record and type C, the program enters an edit mode, and a block cursor may be moved with the horizontal or vertical arrow keys. This cursor is non-destructive, and will not delete characters unless directed to do so. The necessary corrections may now be typed in. To move the cursor to the beginning of the following line, press both the shift and the left arrow keys. When all corrections are made and you press ENTER, the program will now link to the machine-language routine entry point #2 (see line 720 in Program Listing 1). The memory location of the large dummy variable L$ is passed to this entry point (see line 700) because we want the machine-language program to take the record, as shown on the screen, and build a new record in this variable L$.

Entry point #2 is on line 220 of Program Listing 2, and we again CALL 0A7FH and this way get the address of L$ into HL. This register pair is saved in the stack (line 230). Next, this program determines the end of the significant information in each of the 10 lines by looking for the first non-blank in each line, from right to left.

The last blank preceding the first significant character is replaced by !, and when all 10 lines have been delimited, the record can then be packed into variable L$ whose address has been saved in the stack. Packing starts in line 450 of Program Listing 2, where we first retrieve the saved address of L$. Then the record is built, one character at a time, starting with the first screen line (last name) until an ! is found. Next we go to the next screen line (first name) and move those characters until an ! is reached again. This continues until all 10 lines have been packed.

Before returning to the BASIC program, we want to pass the number of characters in our record to it, so that the BASIC program can extract the right number of characters out of the dummy L$, whose length remains unchanged. This is done by transferring the count from IX to HL in lines 670 and 680. We then jump to 0A9AH. (See *Level II BASIC Reference Manual*, page 8/9.) This returns you to the BASIC program, and the record count has been moved out of register pair HL into the variable X. In line 740 of Program Listing 1 we extract the new record and replace the old record in C$(N) with it.

All of those single-character moves are very fast, but would be very slow in BASIC. The above procedure happens if you change a record. You also make use of the same subroutine, entry point #2, whenever you add a record to the file. The add routine also displays the string space remaining. This method of communicating between BASIC and machine language can, of course, be used in many other applications, and I hope that this example will help to bridge the communications gap.

Whenever you are updating data files, it is a good idea to keep at least the two most current versions. This is so that in case of a catastrophe you may go back to the second most current version. This will work all the better if you also keep notes of the changes made during the last updates. If you want to display all records of the file, type P, and when prompted for the last name, type one space and ENTER. The program will display the first record on the file and proceed to the next record as you type N until the end is reached. The sort option is not really necessary but is included anyway since a file in sequence looks better and neater.

You may wonder why I didn't provide for deletion of a record. I did this to conserve program space. A record may easily be deleted (logically) by changing its last name to DELETED, and the next time you want to add a record, first check for any DELETEDs. You may now enter the new information over the old record.

Both programs, BASIC and machine language, are not hard to follow, and for the sake of convenience, I have included a list of the variables used and the lines on which they occur (Table 1), and a list of all branch instructions (Table 2).

Table 1. *Variables list*

A	20									
A$	240	320	400	440	460	480	500	520	780	
	960	1040	1300	1440	1460					
A1$	840	880								
B$	120	140	160	800	980	1000	1060	1420	1500	
C$	20	80	220	300	740	760	860	880	920	1240
L	420	520	540	560	580	600	620	640	660	680
	920									
L$	100	700	740							
L1	920	940								
L2	700	720								
LL	700									
N	20	80	220	240	280	320	400	740	760	1020
	1040	1120	1180	1280						
N1	280	300	840	860	880	900	920	980	1020	1060
N2	1020	1040								
P	420	520	540	560	580	600	620	640	660	680
P1	20	420	520	540	560	580	600	620	640	660
	680									
Q1	1480	1500								
Q3$	1440									
Q4$	1480									
Q5	1500									
Q7	1460									
SA	420	540	560	580	600	620	640	660	680	
SD	1100	1120	1140	1160	1180	1220	1240			
SI	1200	1260								
SJ	1200	1220	1240							
SL	1220	1240								
SP	1100	1180	1280							
SS$	1240									
ST	1180	1200								
SW	20	760	1020	1040						
T1$	40	200	280	1080	1320	1540				
T2$	40	380								
T3$	60	820	980							
T4$	60	840								
T5$	80	200								
T6$	80	280								
X	102	115	720	740	940					
XX	102									
Z	20									

Table 2. *Branch instructions*

```
120   ...GOTO..120 FROM LINE   ..180
      ...GOTO..120 FROM LINE   ..260
      ...GOTO..120 FROM LINE   ..340
      ...THEN..120 FROM LINE   ..800
      ...THEN..120 FROM LINE   ..840
      ...THEN..120 FROM LINE   .1000
      ...THEN..120 FROM LINE   .1060
      ...GOTO..120 FROM LINE   .1300
200   ...THEN..200 FROM LINE   ..120
220   ...GOTO..220 FROM LINE   ..220
280   ...THEN..280 FROM LINE   ..120
360   ...THEN..360 FROM LINE   ..140
380   ...THEN..380 FROM LINE   ..140
400   ...THEN..400 FROM LINE   ..800
420   ...GOTO..420 FROM LINE   .1020
440   ...THEN..440 FROM LINE   ..440
      ...GOTO..440 FROM LINE   ..540
520   ...GOTO..520 FROM LINE   ..440
540   ...ELSE..540 FROM LINE   ..560
      ...GOTO..540 FROM LINE   ..560
      ...GOTO..540 FROM LINE   ..580
      ...ELSE..540 FROM LINE   ..580
      ...GOTO..540 FROM LINE   ..600
      ...ELSE..540 FROM LINE   ..600
      ...GOTO..540 FROM LINE   ..620
      ...GOTO..540 FROM LINE   ..640
      ...ELSE..540 FROM LINE   ..640
      ...GOTO..540 FROM LINE   ..660
      ...ELSE..540 FROM LINE   ..660
560   ...THEN..560 FROM LINE   ..460
580   ...THEN..580 FROM LINE   ..460
600   ...THEN..600 FROM LINE   ..480
620   ...THEN..620 FROM LINE   ..480
640   ...THEN..640 FROM LINE   ..460
660   ...THEN..660 FROM LINE   ..480
680   ...THEN..680 FROM LINE   ..500
780   ...ELSE..780 FROM LINE   ..800
820   ...THEN..820 FROM LINE   ..140
860   ...GOTO..860 FROM LINE   ..880
      ...GOTO..860 FROM LINE   ..980
      ...GOTO..860 FROM LINE   .1060
900   ...THEN..900 FROM LINE   ..880
980   ...THEN..980 FROM LINE   .1000
1040  ...THEN.1040 FROM LINE   ..760
1060  ...ELSE.1060 FROM LINE   .1060
1080  ...THEN.1080 FROM LINE   ..160
1120  ...THEN.1120 FROM LINE   .1120
```

continued

```
1140   ...GOTO.1140 FROM LINE    1260
1220   ...THEN.1220 FROM LINE   .1240
1280   ...THEN.1280 FROM LINE    1160
1320   ..GOSUB.1320 FROM LINE  ...120
1420   ...THEN.1420 FROM LINE   .1420
1440   ...GOSUB.1440 FROM LINE  ..260
       ...GOSUB.1440 FROM LINE  ..340
       ...GOSUB.1440 FROM LINE  ..780
       ...GOSUB.1440 FROM LINE  ..980
       ...GOSUB.1440 FROM LINE   1060
       ...GOSUB.1440 FROM LINE  .1300
1480   ...THEN.1480 FROM LINE    1500
1540   ...GOSUB.1540 FROM LINE  ..380
       ...GOSUB.1540 FROM LINE  ..820
       ...GOSUB.1540 FROM LINE  ..980
       ...GOSUB.1540 FROM LINE  .1060
```

Program Listing 1. *BASIC listing*

```
 1 REM  ******* BY: HUBERT C. BORRMANN, KØMVB *******
 3 REM          MEMORY SIZE : 326Ø7 !
 5 CLS :
   PRINT CHR$(23):
   PRINT @39Ø,"NAME & ADDRESS FILE"
 2Ø CLEAR 9ØØØ:
   DEFINT A - Z:
   DIM C$(1Ø1):
   N = 1:
   P1 = 15572:
   SW = Ø
 4Ø T1$ = "NAME AND ADDRESS LIST ":
   T2$ = "ADDING RECORDS.   RECORD #"
 6Ø T3$ = "PRINTING RECORDS. RECORD #":
   T4$ = "TYPE IN LAST NAME OR 'END'  "
 8Ø C$(N) = "#END#":
   T5$ = "LOADING RECORDS.  RECORD #":
   T6$ = "WRITING RECORDS.  RECORD #"
1ØØ L$ = "1.........2.........3.........4.........5.........6.......
   ..7.........8.........9.........1Ø........11........12........13
   ........14........15........16........17........18........19....
   ....2Ø........21........22........23........ "
1Ø2 RESTORE :
   FOR X = 326Ø8 TO 32762:
    READ XX:
    POKE X,XX:
    NEXT X
1Ø4 DATA 2Ø5,127,1Ø,221,42,243,127,221,34,245,127,1,64,Ø,22,1Ø,126,2
   54,33,4Ø,8,221,119,Ø,35,221,35,24,243,21
1Ø6 DATA 4Ø,13,35,221,42,245,127,221,9,221,34,245,127,24,227,2Ø1,2Ø5
   ,127,1Ø,229,42,247,127,34,249,127,22,1Ø,3Ø,4Ø
1Ø8 DATA 1,64,Ø,126,254,32,32,4,43,29,32,247,35,62,33,119,21,4Ø,11,3
   Ø,4Ø,42,249,127,9,34,249,127,24,229,225
11Ø DATA 253,42,243,127,253,34,245,127,3Ø,32,221,33,Ø,Ø,22,1Ø,253,12
   6,Ø,119,253,115,Ø,221,35,254,33,4Ø,5,253,35
112 DATA 35,24,238,21,4Ø,13,35,253,42,245,127,253,9,253,34,245,127,2
   4,222,221,229,225,195,154,1Ø,21,61,Ø,Ø,6Ø,61,Ø,Ø,Ø
115 FOR X = 1 TO 1ØØØ:
    NEXT X
12Ø GOSUB 132Ø:
   IF B$ = "L"
    THEN
     2ØØ:
    ELSE
     IF B$ = "W"
      THEN
       28Ø
14Ø IF B$ = "A"
    THEN
     38Ø:
    ELSE
     IF B$ = "P"
      THEN
       82Ø:
      ELSE
       IF B$ = "E"
        THEN
         36Ø
16Ø IF B$ = "S"
    THEN
     1Ø8Ø
18Ø GOTO 12Ø
2ØØ CLS :
   PRINT @2Ø,T1$:
   PRINT @64,T5$
```

Program continued

```
220 PRINT @91,N;:
    INPUT # - 1,C$(N):
    IF C$(N) < > "#END#"
     THEN
      N = N + 1:
      GOTO 220
240 PRINT @64,N - 1;" RECORDS LOADED FROM CASSETTE ";:
    A$ = "PRESS ANY KEY WHEN READY."
260 GOSUB 1440:
    GOTO 120
280 CLS :
    PRINT @20,T1$:
    PRINT @64,T6$:
    FOR N1 = 1 TO N
300  PRINT @91,N1;:
     PRINT # - 1,C$(N1):
     NEXT N1
320 PRINT @64,N - 1;" RECORDS WRITTEN OUT ON TAPE":
    A$ = "PRESS ANY KEY WHEN READY"
340 GOSUB 1440:
    GOTO 120
360 CLS :
    PRINT CHR$(23);:
    PRINT @460,"THE  END":
    END
380 GOSUB 1540:
    PRINT @64,T2$
400 PRINT @91,N;:
    PRINT @120, FRE(A$);
420 P = 1:
    L = 1:
    SA = PEEK(P1 + P + (L * 64)):
    POKE P1 + P + (L * 64),138
440 A$ = INKEY$:
    IF A$ = ""
     THEN
      440:
     ELSE
      IF A$ = "," OR A$ = ":" OR A$ = "!"
       THEN
        A$ = ".":
        GOTO 520
460 IF A$ = CHR$(8)
     THEN
      560:
     ELSE
      IF A$ = CHR$(9)
       THEN
        580:
       ELSE
        IF A$ = CHR$(91)
         THEN
          640
480 IF A$ = CHR$(10)
     THEN
      660:
     ELSE
      IF A$ = CHR$(24)
       THEN
        600:
       ELSE
        IF A$ = CHR$(25)
         THEN
          620
500 IF A$ = CHR$(13)
     THEN
      680
520 POKE P1 + P + (L * 64), ASC(A$):
    P = P + 1:
    IF P > 40
     THEN
     P = 4Ø
54Ø SA = PEEK(Pl + P + (L * 64)):
    POKE Pl + P + (L * 64),138:
    GOTO 44Ø
56Ø POKE Pl + P + (L * 64),SA:
    P = P - 1:
    IF P < 1
     THEN
      P = 1:
      GOTO 54Ø:
     ELSE
      54Ø
58Ø POKE Pl + P + (L * 64),SA:
    P = P + 1:
    IF P > 4Ø
     THEN
      P = 4Ø:
      GOTO 54Ø:
     ELSE
      54Ø
6ØØ POKE Pl + P + (L * 64),SA:
    P = 1:
    L = L + 1:
    IF L > 1Ø
     THEN
      L = 1:
      GOTO 54Ø:
     ELSE
      54Ø
62Ø POKE Pl + P + (L * 64),SA:
    P = 4Ø:
    GOTO 54Ø
64Ø POKE Pl + P + (L * 64),SA:
    L = L - 1:
    IF L < 1
     THEN
      L = 1Ø:
      GOTO 54Ø:
     ELSE
      54Ø
66Ø POKE Pl + P + (L * 64),SA:
    L = L + 1:
    IF L > 1Ø
     THEN
      L = 1:
      GOTO 54Ø:
     ELSE
      54Ø
68Ø POKE Pl + P + (L * 64),SA
7ØØ LL = VARPTR(L$):
    L2 = PEEK(LL + 1) + 256 * ( PEEK(LL + 2))
72Ø POKE 16526,142:
    POKE 16527,127:
    X = USR(L2)
74Ø C$(N) = LEFT$(L$,X)
76Ø IF SW < > Ø
     THEN
      1Ø4Ø:
     ELSE
      N = N + 1:
      C$(N) = "#END#"
78Ø A$ = "TYPE 'N' FOR NEXT RECORD, 'M' FOR MENU.":
    GOSUB 144Ø
8ØØ IF B$ = "M"
     THEN
      12Ø:
     ELSE
      IF B$ = "N"
       THEN
        4ØØ:
       ELSE
        78Ø
```

Program continued

```
 820 GOSUB 1540:
     PRINT @64,T3$
 840 PRINT @128,T4$;:
     INPUT A1$:
     IF A1$ = "END"
      THEN
       120:
      ELSE
       N1 = 1
 860 IF LEFT$(C$(N1),5) = "#END#"
      THEN
       GOTO 820
 880 IF A1$ = LEFT$(C$(N1), LEN(A1$))
      THEN
       900:
      ELSE
       N1 = N1 + 1:
       GOTO 860
 900 PRINT @91,N1;
 920 L1 = 1:
     L = VARPTR(C$(N1)):
     L1 = PEEK(L + 1) + 256 * ( PEEK(L + 2))
 940 POKE 16526,96:
     POKE 16527,127:
     X = USR(L1)
 960 A$ = "TYPE 'C' TO CHANGE, 'N' FOR NEXT RECORD, 'M' FOR MENU.."
 980 GOSUB 1440:
     IF B$ = "N"
      THEN
       N1 = N1 + 1:
       CLS :
       GOSUB 1540:
       PRINT @64,T3$;:
       GOTO 860
1000 IF B$ = "M"
      THEN
       120:
      ELSE
       IF B$ < > "C"
        THEN
         980
1020 SW = 1:
     N2 = N:
     N = N1:
     GOTO 420
1040 SW = 0:
     N = N2:
     A$ = "TYPE 'N' FOR NEXT RECORD, 'M' FOR MENU"
1060 GOSUB 1440:
     IF B$ = "M"
      THEN
       120:
      ELSE
       IF B$ = "N"
        THEN
         N1 = N1 + 1:
         CLS :
         GOSUB 1540:
         GOTO 860:
        ELSE
         1060
1080 CLS :
     PRINT @20,T1$:
     PRINT @64,"SORTING THE FILE"
1100 SD = 1:
     SP = 0
1120 SD = 2 * SD:
     IF SD < N - 1
      THEN
       1120
1140 SD = INT((SD - 1) / 2)
```

```
1160 IF SD = 0
      THEN
       1280
1180 ST = N - 1 - SD:
     SP = SP + 1:
     PRINT @140,"PASS # ";SP;
1200 FOR SI = 1 TO ST:
      SJ = SI
1220  SL = SJ + SD
1240  IF LEFT$(C$(SL),10) < LEFT$(C$(SJ),10)
       THEN
        SS$ = C$(SJ):
        C$(SJ) = C$(SL):
        C$(SL) = SS$:
        SJ = SJ - SD:
        IF SJ > 0
         THEN
          1220
1260  NEXT SI:
     GOTO 1140
1280 PRINT @204,N - 1;" RECORDS SORTED IN ";SP;" PASSES";
1300 A$ = "PRESS ANY KEY FOR MENU...":
     GOSUB 1440:
     GOTO 120
1320 CLS :
     PRINT CHR$(23):
     PRINT @8,T1$:
     PRINT @196,"L = LOAD FILE FROM TAPE"
1340 PRINT @260,"W = WRITE FILE ON TAPE":
     PRINT @324,"A = ADD RECORDS TO FILE"
1360 PRINT @388,"P = PRINT (AND CHANGE) RECORDS":
     PRINT @516,"E = END THE PROGRAM"
1380 PRINT @452,"S = SORTING THE FILE";
1400 PRINT @970,"TYPE IN YOUR CHOICE";
1420 B$ = INKEY$:
     IF B$ = ""
      THEN
       1420:
      ELSE
       RETURN
1440 A$ = " " + A$ + " ":
     Q3$ = STRING$( LEN(A$), CHR$(8)):
     Q4$ = A$ + Q3$
1460 PRINT @960, STRING$(63, CHR$(140));:
     Q7 = ((64 - LEN(A$)) / 2):
     PRINT @960 + Q7,"";
1480 FOR Q1 = 1 TO LEN(Q4$):
      PRINT MID$(Q4$,Q1,1);
1500  FOR Q5 = 1 TO 7:
       NEXT Q5,Q1:
      B$ = INKEY$:
      IF B$ = ""
       THEN
        1480
1520  PRINT @960, STRING$(63," ");:
      RETURN
1540  CLS :
      PRINT @20,T1$:
      PRINT @257,"LAST NAME":
      PRINT @321,"FIRST NAME & INIT."
1560  PRINT @385,"STREET ADDRESS":
      PRINT @449,"CITY OR TOWN"
1580  PRINT @513,"STATE":
      PRINT @577,"ZIP-CODE":
      PRINT @641,"TELEPHONE #"
1600  PRINT @705,"NOTE 1":
      PRINT @769,"NOTE 2":
      PRINT @833,"REMARKS";:
      RETURN
```

Program Listing 2. *Assembled listing*

```
7F6Ø                      ORG     7F6ØH
7F6Ø CD7FØA         ENTR1 CALL    ØA7FH
7F63 DD2AF37F             LD      IX,(ADDR1)
7F67 DD22F57F             LD      (ADDR2),IX
7F6B Ø14ØØØ               LD      BC,ØØ4ØH
7F6E 16ØA                 LD      D,ØAH
7F7Ø 7E             CHCK1 LD      A,(HL)
7F71 FE21                 CP      21H      ;="!"
7F73 28Ø8                 JR      Z,FND1
7F75 DD77ØØ               LD      (IX+ØØH),A
7F78 23                   INC     HL
7F79 DD23                 INC     IX
7F7B 18F3                 JR      CHCK1
7F7D 15             FND1  DEC     D
7F7E 28ØD                 JR      Z,DONE1
7F8Ø 23                   INC     HL
7F81 DD2AF57F             LD      IX,(ADDR2)
7F85 DDØ9                 ADD     IX,BC
7F87 DD22F57F             LD      (ADDR2),IX
7F8B 18E3                 JR      CHCK1
7F8D C9             DONE1 RET
7F8E CD7FØA         ENTR2 CALL    ØA7FH
7F91 E5                   PUSH    HL
7F92 2AF77F               LD      HL,(ADDR3)
7F95 22F97F               LD      (ADDR4),HL
7F98 16ØA                 LD      D,ØAH
7F9A 1E28                 LD      E,28H    ;="("
7F9C Ø14ØØØ               LD      BC,ØØ4ØH
7F9F 7E             CHCK2 LD      A,(HL)
7FAØ FE2Ø                 CP      2ØH      ;=" "
7FA2 2ØØ4                 JR      NZ,EXCL
7FA4 2B                   DEC     HL
7FA5 1D                   DEC     E
7FA6 2ØF7                 JR      NZ,CHCK2
7FA8 23             EXCL  INC     HL
7FA9 3E21                 LD      A,21H    ;="!"
7FAB 77                   LD      (HL),A
7FAC 15                   DEC     D
7FAD 28ØB                 JR      Z,PACK
7FAF 1E28                 LD      E,28H    ;="("
7FB1 2AF97F               LD      HL,(ADDR4)
7FB4 Ø9                   ADD     HL,BC
7FB5 22F97F               LD      (ADDR4),HL
7FB8 18E5                 JR      CHCK2
7FBA E1             PACK  POP     HL
7FBB FD2AF37F             LD      IY,(ADDR1)
7FBF FD22F57F             LD      (ADDR2),IY
7FC3 1E2Ø                 LD      E,2ØH    ;=" "
7FC5 DD21ØØØØ             LD      IX,ØØØØH
7FC9 16ØA                 LD      D,ØAH
7FCB FD7EØØ         CHCK3 LD      A,(IY+ØØH)
7FCE 77                   LD      (HL),A
7FCF FD73ØØ               LD      (IY+ØØH),E
7FD2 DD23                 INC     IX
7FD4 FE21                 CP      21H      ;="!"
7FD6 28Ø5                 JR      Z,FND2
7FD8 FD23                 INC     IY
7FDA 23                   INC     HL
7FDB 18EE                 JR      CHCK3
7FDD 15             FND2  DEC     D
7FDE 28ØD                 JR      Z,DONE2
7FEØ 23                   INC     HL
7FE1 FD2AF57F             LD      IY,(ADDR2)
7FE5 FDØ9                 ADD     IY,BC
7FE7 FD22F57F             LD      (ADDR2),IY
```

```
7FEB 18DE      00660          JR      CHCK3
7FED DDE5      00670 DONE2    PUSH    IX
7FEF E1        00680          POP     HL
7FF0 C39A0A    00690          JP      0A9AH
7FF3 153D      00700 ADDR1    DEFW    15637
7FF5 0000      00710 ADDR2    DEFW    0
7FF7 3C3D      00720 ADDR3    DEFW    15676
7FF9 0000      00730 ADDR4    DEFW    0
0000           00740          END
```

Expense Report

by William Klungle

This program keeps tabs on your expenses quickly and easily every month. The program provides a monthly printout of expenses for the current month, year to date, and the percentage each represents of your total income. Expense Report requires a TRS-80 with an expansion interface, a minimum of 32K memory, one disk drive, and an 80-column line printer.

Soft Keys

The program uses what I call a "soft key" method of obtaining command decisions from the operator. When the soft keys are activated, only a keyboard input of one of the requested keys allows the program to continue. All major program decisions use the soft keys in place of the standard menu method.

Two subroutines control soft key operation. Subroutine 3000 enables the keys, but before this subroutine is called, the elements of array L$ are defined to properly label the keys. The routine displays the keys on the bottom of the screen, then the program remains in a closed loop until the operator presses one of the requested keys. When a proper key is pressed, the value of the key is returned in variable X. The soft keys are removed from the screen by subroutine 4000, which also nulls the elements of array L$.

Simple Entry

Original entry, account correction, and expense entries are all accomplished through the use of forms. The original entry and correction form is displayed on the screen by subroutine 2000. The monthly expense entry form is displayed by subroutine 165.

The heart of this program is subroutine 8000. This routine allows normal keyboard entry within the form boundaries, yet guards against illegal keystrokes as well as prevents the operator from typing more than the allowable number of characters for any given input request. Before this routine is called, the length variable (LG) is set to the maximum number of characters to be allowed and the location variable (PA) is set to position the pointer in the form. The pointer (>) is displayed in the form position which is currently accepting input.

During original entry, each input requires data. The program does not allow the operator to bypass an entry unless something (in the case of numerics, the proper data) is entered. The back-space key still functions normally to allow corrections before the ENTER key is pressed; the up-arrow key allows the operator to back through the form to a previous entry. If the up arrow is pressed when the pointer is in the first position (ACCT #), the program returns to the main menu keys.

The account numbers must be between 101 and 180. The account number minus 100 is the actual array location of the account data as it is stored in arrays AC$(#) and M(#,#).

Data Storage

When data is to be stored, this program stores one data set per record on the disk. During program operation all account data is held in memory, which allows fast access for modification or entries. If the data currently

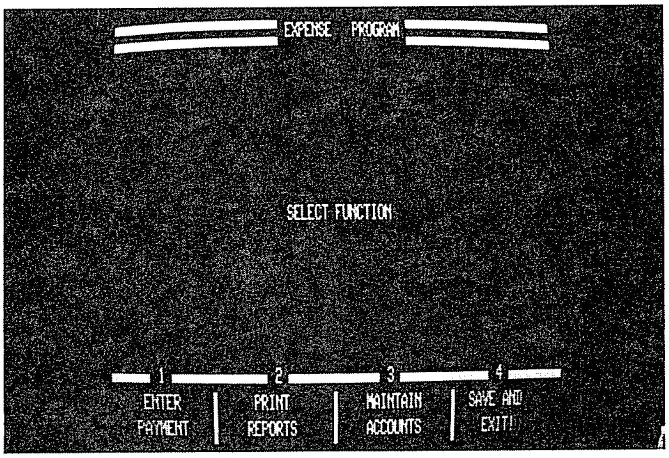


Photo 1. *Expense program menu*

held in memory is modified or updated, file flag F is set equal to one. Before the program is terminated, flag F is checked and if it is equal to one, the operator is required to store the current data on the disk to update the file. Subroutine 5000 writes data from the arrays to the disk.

Data Read

One of the first operations the program performs after initialization is to load the previously stored account data into memory. Subroutine 6000 reads the data from the disk. The program requires from 30 to 45 seconds to completely load all 80 accounts. During the read, the program incorporates an ON ERROR statement to prevent program termination in the event of a data read error. If a read error does occur, the program branches to line

6020 and displays a POSSIBLE DATA LOSS message before continuing the read. If this message appears, check the current data carefully before continuing normal operation. When typing this program, leave the ON ERROR statement out until the program has been fully tested and all typing errors have been corrected.

Current Accounts

The current account descriptions and data are stored in two arrays, AC$(#) and M(#,#). AC$(#) contains the account number (3 numeric), the account description (1-30 characters), and the break flag (1-2 characters). The first position of AC$(#) for each active element is a period (.). This character is a program test to determine quickly if a particular array element contains valid data. The data positions in an active element of AC$(#) are:

Position		Description
1	=	(.) if active
2-3	=	break flag
4-33	=	acct. description

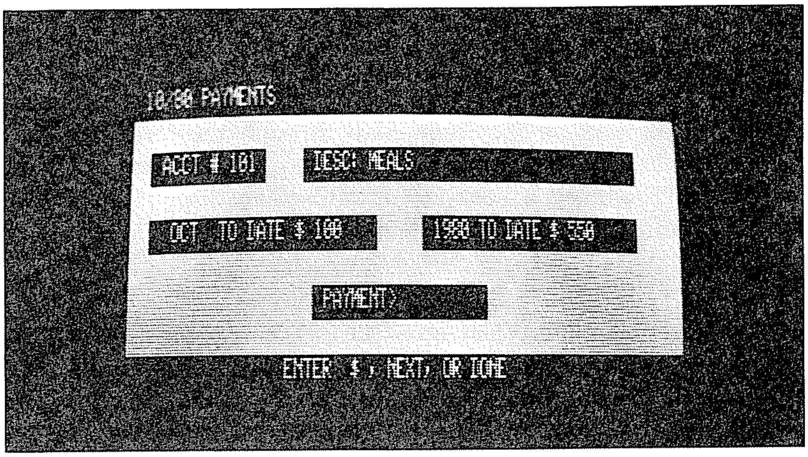
The numeric data for each account is stored by monthly division in the element of array M(#,#) which corresponds to its active element in array AC$(#). The account monthly total is stored in its respective array subelement. Example: M(#,6) = June dollars.

Report Printing

When you select the report printing function, the program reads through data array AC$(#), searching for at least one element with a period in the first position. This insures that the data for at least one account is available for printing. Before printing can begin, you must enter a total income amount for the current month and for the current year-to-date. The report uses these figures in calculating percentage in the various expense accounts. After these figures have been entered, a second group of soft keys is displayed, allowing printing of the report, selection of the number of copies needed, or return to the main menu.

Date Control

The current month entered when the program first initializes controls access to the monthly data. As an example, if the date were entered as 06/30/80, the data entries made to the accounts would be stored in the sixth subelement of array M(#,#). A report printed at this time would provide information for June as the current month; the year-to-date amounts would be calculated on all data from January through June. In order to alter the current month, you must terminate and restart the program.



```
10/80 PAYMENTS

ACCT # 101     DESC: MEALS

OCT  TO DATE $ 100     1980 TO DATE $ 550

          PAYMENT?

     ENTER $, NEXT, OR DONE
```

Photo 2. *Account entry format*

Entry and Correction

After the ENTRY & CORRECT soft key is selected, the proper form appears on the screen. The first entry requested is the account number. This must be a numeric entry from 101 to 180. If the number entered is already assigned to a current account, that account data automatically displays on the form and the program enters the correction mode. In this mode, an entry may be left unchanged by pressing only the ENTER key. Data typed for any entry replaces the current data being displayed. The up-arrow key allows you to step back one entry for each time the key is pressed. If you press the up arrow when the entry pointer is in the account number position, the form is removed and the program returns to the main menu soft keys. After the last entry on the form, soft keys are displayed which allow you to save the data, delete the data in the case of a correction, or return to the menu for a new entry without saving the data.

Break Flag

Since this program does not provide an account sorting feature, original data entry requires planning. The subtotal breaks are keyed by a change in the account group break flag. When making the original entries the operator should plan to enter accounts of the same group in numeric sequence. The account group break flag may be any combination of two letters or numbers to represent a particular account group. As the report prints, a subtotal occurs whenever the flag changes.

Delete

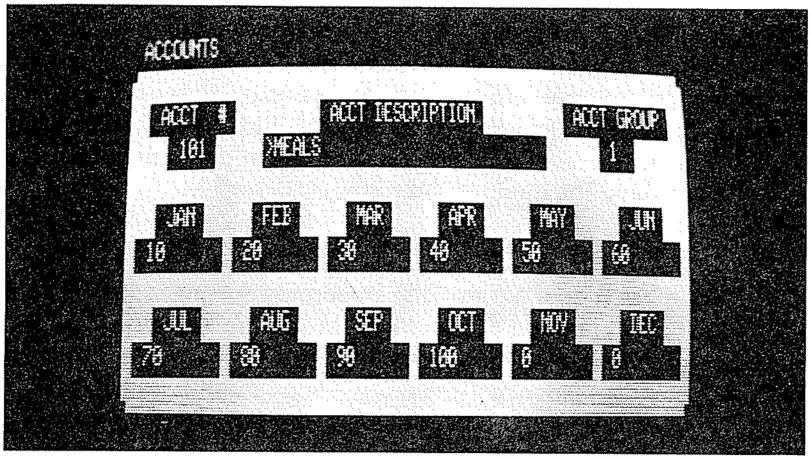
The program provides for account removal through a delete soft key. In the entry and correction mode, if an existing account number is entered, the account data displays on the form. Skip through the form by pressing ENTER, and when the soft keys come up, a delete key is present. After pressing the DELETE soft key, a second set of soft keys are displayed to allow the operator to confirm or cancel the delete command.

Enter Expenses

Enter expenses monthly. When the ENTER PAYMENT soft key is pressed, the expense entry form displays. Entering a current account number brings that account to the form. The account description, current monthly total of expenses already entered for that month, and the year-to-date total of expenses for the account display. When an amount is entered and the ENTER key is pressed, the current month and year-to-date totals update to reflect the added (or subtracted) amount. When all entries are made to that account, type N or NEXT, then press ENTER to clear the form for the next account. When all expense entries are made, type D or DONE, then press ENTER to return to the MENU. Pressing the up-arrow key when the pointer is in the account number position also returns the program to the menu.

Printer Control

If this program is to run on a TRS-80 with a printer connected to the par-



```
ACCOUNTS

  ACCT #          ACCT DESCRIPTION          ACCT GROUP
   101      >MEALS                              1

   JAN      FEB      MAR      APR      MAY      JUN
 10       20       30       40       50       60

   JUL      AUG      SEP      OCT      NOV      DEC
 70       80       90       100      0        0
```

Photo 3. *Year-to-date totals*

allel port of the expansion interface, add the following program line to prevent hangups in the event the printer is off-line when a report is requested.

```
320 CC = 0: IF PEEK(14312) > 127 THEN FOR T = 1 TO 5:
  PRINT@729,"CHECK PRINTER";: FOR I = 1 TO 50: NEXT I:
  PRINT@729,"                     ";: FOR I = 1 TO 50: NEXT I:
  NEXT T: GOTO 310
```

Although this program was written to provide accurate monthly expense reports, you can adapt it to any personal or business application which requires account-type reporting. Type the program into the computer by sections. Enter the subroutines first, then enter and test each section to eliminate typing errors. Remember to leave the ON ERROR statements out until the program has been completely debugged.

```
                                 EXPENSE REPORT
PAGE 1                         PRINTED ON 10/30/80
ACCT #         DESCRIPTION      OCT AMT    %    YEAR TO DATE    %
         INCOME AMOUNT          1324.56  1.00       14235.80  1.00
101      MEALS                   100.00  0.08         550.00  0.04
102      LODGING                 459.67  0.35        2474.67  0.17
103      TRANSPORTATION          345.00  0.26        1681.00  0.12
                               --------         ------------
                               --------         ------------
  SUB-TOTAL                      904.67  0.68        4705.67  0.33
110      UNIFORMS                 23.00  0.02         261.00  0.02
111      TOOLS                    55.00  0.04         420.00  0.03
                               --------         ------------
                               --------         ------------
  SUB-TOTAL                       78.00  0.06         681.00  0.05
120      MILEAGE @.18 PER
         MILE                     54.66  0.04         391.44  0.03
121      TOLLS AND FEES           16.00  0.01         250.00  0.02
                               --------         ------------
                               --------         ------------
  SUB-TOTAL                       70.66  0.05         641.44  0.05
130      TRAINING MATERIALS       89.00  0.07         332.00  0.02
131      TRAINING FEES           125.00  0.09        1039.00  0.07
                               --------         ------------
                               --------         ------------
  SUB-TOTAL                      214.00  0.16        1371.00  0.10
  GRAND TOTALS                  1267.33  0.96        7399.11  0.52
```

Example 1. *Sample expense report*

Program Listing

```
10 :
   '  EXPENSE REPORT PROGRAM          W. KLUNGLE
15 :
   '  TRS-8Ø, 32K MEM, 1 DISC, 8Ø COL PRINTER
2Ø :
   '
29 :
   '  INITIALIZE
3Ø CLEAR 4ØØØ:
   DIM H$(12),L$(8),AC$(8Ø),M(8Ø,12):
   DEFINT I,J,K,T
35 FOR I = 1 TO 12:
    READ H$(I):
    NEXT I:
   DATA " JAN "," FEB "," MAR "," APR "," MAY "," JUN "," JUL "," A
   UG "," SEP "," OCT "," NOV "," DEC "
36 LF$ = "%     %%                           %######.##% %#.##%
       %#######.##% %#.##":
   ST$ = "%                                 %#######.##% %#.##%
       %#######.##% %#.##"
37 US$ = "%                                  %%         %%
      %%         %"
38 :
   '
39 :
   '  MENU
4Ø F = Ø:
   SK = Ø:
   GOSUB 1ØØØ:
   PRINT @467,"ENTER TODAYS DATE AS MM/DD/YY":
   PRINT @537,"DATE=";:
   LINE INPUT "";D$:
   IF LEN(D$) < > 8
    THEN
     4Ø:
    ELSE
     CM = VAL( LEFT$(D$,2)):
     IF CM < 1 OR CM > 12
      THEN
       4Ø
45 ON ERROR GOTO 5ØØ:
   PRINT @467, STRING$(4Ø," ");:
   PRINT @531, STRING$(4Ø," ");:
   GOSUB 6ØØØ:
   ON ERROR GOTO 5ØØ
5Ø GOSUB 1ØØØ:
   PRINT @472,"SELECT FUNCTION":
   L$(1) = "  ENTER":
   L$(5) = " PAYMENT":
   L$(2) = " PRINT":
   L$(6) = "REPORTS":
   L$(3) = " MAINTAIN":
   L$(7) = " ACCOUNTS":
   L$(4) = "SAVE AND":
   L$(8) = "  EXIT!"
55 GOSUB 3ØØØ:
   GOSUB 4ØØØ:
   ON X GOTO 1ØØ,3ØØ,2ØØ,4ØØ
99 :
   '
1ØØ :
   '  ENTER EXPENSES
1Ø5 CLS :
   CM = VAL( LEFT$(D$,2)):
   GOSUB 17Ø
```

```
110 PRINT @530,"  ENTER  ACCOUNT  NUMBER    ";:
    PRINT @139,"    ";:
    PRINT @154, STRING$(30," ");:
    PRINT @276,"        ";:
    PRINT @304,"         ";
115 PA = 139:
    LG = 3:
    GOSUB 8000:
    IF T$ = ""
     THEN
      50:
     ELSE
      IF LEN(T$) < > 3 OR VAL(T$) < 101 OR VAL(T$) > 180
       THEN
        GOSUB 8100:
        GOTO 115
120 A = VAL(T$) - 100:
    IF LEFT$(AC$(A),1) < > "."
     THEN
      GOSUB 8100:
      GOTO 115
125 PRINT @155, MID$(AC$(A),7,30);:
    YT = 0:
    MT = 0:
    FOR I = 1 TO CM:
     YT = YT + M(A,I):
     NEXT I:
    MT = M(A,CM)
135 PRINT @276,MT;:
    PRINT @304,YT;
140 PRINT @530,"ENTER  $ , NEXT, OR DONE ";:
    PA = 413:
    LG = 10:
    PRINT @PA,"          ";:
    GOSUB 8000:
    IF LEFT$(T$,1) = "N"
     THEN
      110:
     ELSE
      IF LEFT$(T$,1) = "D"
       THEN
        50:
       ELSE
        AP = VAL(T$):
        MT = MT + AP:
        YT = YT + AP:
        PRINT @276,MT;:
        PRINT @304,YT;
145 F = 1:
    PRINT @530,"     * DATA FILED *      ";:
    M(A,CM) = MT:
    FOR T = 1 TO 250:
     NEXT T:
    GOTO 140
164 :
    '
165 :
    '  EXPENSE ENTRY FORM
170 CLS :
    FOR I = 66 TO 450 STEP 64:
     PRINT @I, STRING$(60, CHR$(191));:
     NEXT I:
    PRINT @132," ACCT #     ";:
    PRINT @148," DESC:                                 ";:
    PRINT @260," ";H$(CM);" TO DATE $        ";:
    PRINT @289," 19"; RIGHT$(D$,2);" TO DATE $         ";
175 PRINT @405," PAYMENT             ";:
    PRINT @3, LEFT$(D$,3); RIGHT$(D$,2);" PAYMENTS";:
    RETURN
199 :
    '
```

Program continued

```
200 :
    '  ENTER AND MAINTAIN ACCOUNTS
205 CLS :
    GOSUB 2000
210 PA = 198:
    LG = 3:
    RT = 0:
    CR = 0:
    GOSUB 8000:
    IF RT = 1 AND T$ = ""
     THEN
      50:
     ELSE
      IF LEN(T$) < 3
       THEN
        215:
       ELSE
        A = VAL(T$) - 100:
        IF A < 1 OR A > 80
         THEN
          215:
         ELSE
          N$ = T$:
          GOTO 220
215 GOSUB 8100:
    PRINT @PA,"      ";:
    GOTO 210
220 GOSUB 2100:
    IF LEFT$(AC$(A),1) = "."
     THEN
      CR = 1:
      GOSUB 2200
225 PA = 208:
    LG = 30:
    RT = 0:
    GOSUB 8000:
    IF RT = 1 AND T$ = ""
     THEN
      210:
     ELSE
      IF T$ < > ""
       THEN
        DS$ = T$:
        IF RT = 1
         THEN
          210
230 IF T$ = "" AND CR = 1
     THEN
      DS$ = MID$(AC$(A),7,30):
     ELSE
      IF T$ = ""
       THEN
        GOSUB 8100:
        PRINT @PA, STRING$(30," ");:
        GOTO 225
235 PA = 245:
    LG = 2:
    RT = 0:
    GOSUB 8000:
    IF RT = 1 AND T$ = ""
     THEN
      225:
     ELSE
      IF T$ < > ""
       THEN
        B$ = T$:
        IF RT = 1
         THEN
          225
240 IF T$ = "" AND CR = 1
     THEN
```

```
      B$ = MID$(AC$(A),2,2):
     ELSE
      IF T$ = ""
       THEN
        GOSUB 8100:
        GOTO 235
245 PA = 387:
    LG = 7:
    RT = 0:
    I = 1:
    GOTO 255
250 PA = PA - 10:
    I = I - 1:
    RT = 0:
    IF PA = 569
     THEN
      PA = 437:
     ELSE
      IF PA < 384
       THEN
        235
255 GOSUB 8000:
    IF RT = 1 AND T$ = ""
     THEN
      250:
     ELSE
      IF T$ < > ""
       THEN
        M(A,I) = VAL(T$):
        IF RT = 1
         THEN
          250
256 IF T$ = "" AND CR = 1
     THEN
      PRINT @PA, LEFT$( STR$(M(A,I)),7);:
     ELSE
      IF T$ = ""
       THEN
        GOSUB 8100:
        PRINT @PA,"          ";:
        GOTO 255
260 PA = PA + 10:
    I = I + 1:
    IF PA = 447
     THEN
      PA = 579
265 IF PA < 630
     THEN
      255
266 IF LEN(B$) = 1
     THEN
      B$ = B$ + " "
267 IF LEN(DS$) < 30
     THEN
      DS$ = DS$ + " ":
      GOTO 267
270 L$(1) = "   FILE":
    L$(5) = "   DATA":
    L$(3) = "  DELETE":
    L$(7) = " ACCOUNT":
    IF LEFT$(AC$(A),1) < > "."
     THEN
      L$(4) = "  RETURN":
      L$(8) = " TO  MENU"
271 GOSUB 3000:
    GOSUB 4000:
    ON X GOTO 275,270,280:
    IF LEFT$(AC$(A),1) < > "."
     THEN
      50:
     ELSE
      270
```

Program continued

```
275 F = 1:
    PRINT @793,"*  DATA  FILED  *";:
    AC$(A) = "." + B$ + N$ + DS$:
    FOR T = 1 TO 200:
     NEXT T:
    PRINT @793,"                        ";:
    PRINT @198,"     ";:
    GOSUB 2100:
    GOTO 210
280 L$(1) = " CONFIRM":
    L$(5) = " DELETE!":
    L$(4) = " CANCEL":
    L$(8) = " DELETE":
    GOSUB 3000:
    GOSUB 4000:
    ON X GOTO 285,280,280,270
285 AC$(A) = " ":
    PRINT @793,"ACCOUNT DELETED":
    FOR T = 1 TO 1000:
     NEXT T:
    PRINT @793,"                   ":
    PRINT @198,"     ";:
    GOSUB 2100:
    GOTO 210
299 :
    '
300 :
    '  REPORT PRINTING SECTION
305 GOSUB 1000:
    PRINT @23,"  REPORT PRINTING  ";:
    PRINT @464,"PRINTER ON-LINE, PAPER IN PLACE!";:
    NC = 1
306 PRINT @590,"ENTER NET INCOME FOR ";H$(CM);" $";:
    LINE INPUT "";NS$:
    NS = VAL(NS$):
    IF NS < 1
     THEN
      306
307 PRINT @654,"ENTER NET INCOME FOR 19"; RIGHT$(D$,2);" $";:
    LINE INPUT "";YS$:
    YS = VAL(YS$):
    IF YS < 1
     THEN
      307
310 PRINT @588, STRING$(40," ");:
    PRINT @652, STRING$(40," ");:
    PRINT @603,"COPIES= ";NC;:
    L$(1) = "  PRINT":
    L$(5) = "  REPORT":
    L$(3) = "  SELECT":
    L$(7) = "  COPIES":
    L$(4) = "  RETURN":
    L$(8) = " TO  MENU":
    GOSUB 3000:
    GOSUB 4000:
    ON X GOTO 320,310,315,50
315 NC = NC + 1:
    IF NC > 5
     THEN
      NC = 1
316 GOTO 310
320 CC = 0:
    REM -TYPE PRINTER CHECK HERE FOR PARALLEL PRINTER
323 T = 1:
    S1 = 0:
    S2 = 0:
    G1 = 0:
    G2 = 0:
    SK = 1:
    TP$ = LEFT$(AC$(1),3):
```

```
    LC = 0:
    PC = 1:
    PRINT @594,"    PRINTING   COPY    ";CC + 1;"          ":
    L$(3) = "  STOP":
    L$(7) = " PRINT!":
    GOSUB 3000:
    SK = 0
324 IF LEFT$(AC$(T),1) < > "."
     THEN
      350
325 IF LC = 0
     THEN
      LPRINT TAB(33);"   EXPENSE   REPORT   ":
      LPRINT " ":
      LPRINT "PAGE ";PC; TAB(34);"PRINTED ON ";D$:
      PC = PC + 1
330 IF LC = 0
     THEN
      LPRINT " ":
      LPRINT "ACCT #          DESCRIPTION           ";H$(CM);" AMT
       %         YEAR TO DATE      %":
      LPRINT " ":
      LPRINT USING LF$;" ","INCOME AMOUNT",NS," ",1," ",YS," ",1:
      LPRINT " ":
      LC = 8
335 IF TP$ < > LEFT$(AC$(T),3)
     THEN
      TP$ = LEFT$(AC$(T),3):
      P1 = S1 / NS:
      P2 = S2 / YS:
      GOSUB 360:
      LPRINT USING ST$;" SUB-TOTAL",S1," ",P1," ",S2," ",P2:
      LPRINT " ":
      G1 = G1 + S1:
      G2 = G2 + S2:
      S1 = 0:
      S2 = 0:
      LC = LC + 2
340 AD = M(T,CM):
    YD = 0:
    :
    FOR I = 1 TO CM:
     YD = YD + M(T,I):
     NEXT I:
    S1 = S1 + AD:
    S2 = S2 + YD:
    P1 = AD / NS:
    P2 = YD / YS:
    X$ = INKEY$:
    IF X$ = "3"
     THEN
      GOSUB 4000:
      LPRINT "*INCOMPLETE REPORT!":
      FOR TP = 1 TO 65 - LC:
       LPRINT " ":
       NEXT :
      GOTO 310
345 LPRINT USING LF$; MID$(AC$(T),4,3), MID$(AC$(T),7,30),AD," ",P1,
    " ",YD," ",P2:
    LC = LC + 1:
    IF LC > 55
     THEN
      FOR TP = 1 TO 66 - LC:
       LPRINT " ":
       NEXT :
      LC = 0
350 T = T + 1:
    IF T < 81
     THEN 324:
     ELSE
```

Program continued

```
        P1 = S1 / NS:
        P2 = S2 / YS:
        GOSUB 360:
        LPRINT USING ST$;" SUB-TOTAL",S1," ",P1," ",S2," ",P2:
        G1 = G1 + S1:
        G2 = G2 + S2:
        P1 = G1 / NS:
        P2 = G2 / YS:
        LPRINT " ":
        LPRINT USING ST$;" GRAND TOTALS",G1," ",P1," ",G2," ",P2:
        LC = LC + 3
 355 FOR TP = 1 TO 66 - LC:
      LPRINT " ":
      NEXT :
     CC = CC + 1:
     IF CC < NC
      THEN
       323:
      ELSE
       310
 360 LPRINT USING US$;" ","=========="," ","============":
     LC = LC + 1:
     RETURN
 399 :
     '
 400 :
     '  SAVE DATA AND EXIT PROGRAM
 405 GOSUB 1000:
     IF F = 1
      THEN
       GOSUB 5000:
       PRINT @472,"SELECT  FUNCTION";:
       L$(1) = " RETURN":
       L$(5) = " TO MENU":
       L$(3) = "  STORE":
       L$(7) = "  AGAIN":
       L$(4) = "  EXIT":
       GOSUB 3000:
       GOSUB 4000:
       ON X GOTO 415,405,400,410
 410 CLS :
     PRINT :
     PRINT "EXPENSE PROGRAM TERMINATED.":
     PRINT @890,"":
     END
 415 F = 0:
     GOTO 50
 500 RESUME 50
 999 :
     '
1000 :
     '   SCREEN HEADER ROUTINE
1005 CLS :
     PRINT STRING$(64, CHR$(179));:
     PRINT @23," EXPENSE   PROGRAM ";:
     RETURN
1099 :
     '
2000 :
     '   ENTRY FORM ROUTINE
2005 FOR I = 66 TO 665 STEP 64:
      PRINT @I, STRING$(61, CHR$(191));:
      NEXT I:
     POKE 15426,190:
     POKE 15486,189:
     POKE 16002,175:
     POKE 16062,159
2010 PRINT @3,"ACCOUNTS";:
     PRINT @132," ACCT  # ";:
     PRINT @150," ACCT DESCRIPTION ";:
     PRINT @177," ACCT GROUP ";:
```

```
     PRINT @198,"     ";:
     PRINT @2Ø8, STRING$(3Ø," ");:
     PRINT @245,"    ";
2Ø15 C = 1:
     FOR I = 325 TO 382 STEP 1Ø:
      PRINT @I,H$(C);:
      PRINT @I + 62,"          ";:
      C = C + 1:
      NEXT I:
     FOR I = 517 TO 575 STEP 1Ø:
      PRINT @I,H$(C);:
      PRINT @I + 62,"          ";:
      C = C + 1:
      NEXT I
2Ø2Ø RETURN
2Ø99 :
     '
21ØØ :
     '  CLEAR FORM ROUTINE
21Ø5 PRINT @2Ø8, STRING$(3Ø," ");:
     PRINT @245,"    ";:
     FOR I = 387 TO 437 STEP 1Ø:
      PRINT @I,"         ";:
      NEXT I:
     FOR I = 579 TO 629 STEP 1Ø:
      PRINT @I,"         ";:
      NEXT I:
     RETURN
2199 :
     '
22ØØ :
     '  PRINT IN FORM ROUTINE
22Ø5 PRINT @2Ø9, MID$(AC$(A),7,3Ø);:
     PRINT @246, MID$(AC$(A),2,2);:
     C = 1:
     FOR I = 387 TO 437 STEP 1Ø:
      PRINT @I, LEFT$( STR$(M(A,C)),7);:
      C = C + 1:
      NEXT I:
     FOR I = 579 TO 629 STEP 1Ø:
      PRINT @I, LEFT$( STR$(M(A,C)),7);:
      C = C + 1:
      NEXT I:
     RETURN
2999 :
     '
3ØØØ :
     '   SOFT KEY ENABLE ROUTINE
3ØØ5 PRINT @832, STRING$(64, CHR$(14Ø));:
     FOR K = 16269 TO 16319 STEP 17:
      POKE K,17Ø:
      POKE K + 64,17Ø:
      NEXT :
     X = 1:
     FOR K = 837 TO 894 STEP 16:
      PRINT @K,X;:
      PRINT @K + 61,L$(X);:
      PRINT @K + 125,L$(X + 4);:
      X = X + 1:
      NEXT :
     IF SK = 1
      THEN
       RETURN
3Ø1Ø X$ = INKEY$:
     IF X$ < "1" OR X$ > "4"
      THEN
       3Ø1Ø:
      ELSE
       X = VAL(X$):
       RETURN
3Ø99 :
     '
```

Program continued

```
4000 :
     '   SOFT KEY DISABLE ROUTINE
4005 X$ = STRING$(64," "):
     PRINT @832,X$;:
     PRINT @896,X$;:
     PRINT @960, LEFT$(X$,63);:
     FOR K = 1 TO 8:
      L$(K) = "":
      NEXT :
     RETURN
4090 :
     '
5000 :
     '   PRINT TO FILE
5005 OPEN "O",1,"DATAFILE":
     PRINT @473,"UPDATING FILES";:
     FOR T = 1 TO 80:
      PRINT #1, CHR$(34);AC$(T); CHR$(34);M(T,1);M(T,2);M(T,3);M(T,4)
      ;M(T,5);M(T,6);M(T,7);M(T,8);M(T,9);M(T,10);M(T,11);M(T,12):
      PRINT @542,T;:
      NEXT T:
     PRINT @542,"     ";:
     CLOSE :
     RETURN
5090 :
     '
6000 :
     '  READ FROM FILE
6005 OPEN "I",1,"DATAFILE":
     ON ERROR GOTO 6020:
     PRINT @473,"READING FILES";:
     FOR T = 1 TO 80:
      INPUT #1,AC$(T),M(T,1),M(T,2),M(T,3),M(T,4),M(T,5),M(T,6),M(T,7
      ),M(T,8),M(T,9),M(T,10),M(T,11),M(T,12):
      PRINT @542,T;:
      NEXT T:
     CLOSE :
     RETURN
6010 PRINT @473,"              ";:
     S = 37:
     FOR I = 1 TO 12:
      M(T,I) = VAL( MID$(W$,S,7)):
      M(T + 1,I) = VAL( MID$(W$,S + 120,7)):
      NEXT I:
     NEXT T:
     CLOSE 1:
     RETURN
6020 PRINT @600,"POSSIBLE DATA LOSS!";:
     RESUME NEXT
7999 :
     '
8000 :
     '   KEY INPUT ROUTINE
8010 PRINT @PA,">";:
     T$ = ""
8020 X$ = INKEY$:
     IF X$ > CHR$(31) AND X$ < CHR$(91)
      THEN
       8030:
      ELSE
       IF X$ = CHR$(91)
        THEN
         8070:
        ELSE
         IF X$ = CHR$(8)
          THEN
           8050:
          ELSE
           IF X$ = CHR$(13)
            THEN
             8080:
```

```
           ELSE
            8020
8030 IF LEN(T$) < LG
      THEN
       PRINT X$;:
       T$ = T$ + X$:
       IF LEN(T$) < LG - 1
        THEN
         PRINT CHR$(95); CHR$(24);
8040 GOTO 8020
8050 IF LEN(T$) < LG
      THEN
       PRINT " "; CHR$(24);
8060 IF LEN(T$) > 0
      THEN
       T$ = LEFT$(T$, LEN(T$) - 1):
       PRINT X$; CHR$(95); CHR$(24);
8065 GOTO 8020
8070 RT = 1
8080 IF LEN(T$) > 0 AND LEN(T$) < LG
      THEN
       PRINT " ";
8090 PRINT @PA," ";:
     RETURN
8100 :
     '
8110 :
     '    ENTRY ERROR ROUTINE
8120 PRINT @23,"*  ENTRY  ERROR  *";:
     FOR K = 1 TO 5:
      PRINT @PA,"*";:
      FOR J = 1 TO 30:
       NEXT J:
      PRINT @PA," ";:
      FOR J = 1 TO 30:
       NEXT J:
      NEXT K:
     PRINT @23, STRING$(30," ");:
     RETURN
```

EDUCATION

Story Math

Smile—TRS-80 Loves You

Story Math

by William Klungle

I don't understand these questions! They're dumb and I can't do them anyway! I quit!" With that, my fifth grade daughter closed the book on her math assignment of story problems. Later that evening she was happily sharing the computer with her sister, enjoying a program called Story. Story generates random short stories by using the names of the children running the program. My daughters and their friends have run and enjoyed this program for the last year.

As I watched, a method of changing my daughter's opinion of story problems came to mind. If she enjoyed the Story program so much, why not combine it with a math program? The results were well worth the effort. If you own a 16K Level II TRS-80 and have grade-school-age children that dislike story problems as much as mine, you may find it worth your time to try this program.

Designing the Program

Programming the math function on the computer is not very difficult, but when you wish to have the problems presented in a variety of stories, the task becomes more difficult. Story Math is designed for children in grades one through six, and the difficulty level of the questions asked is determined by the school grade of the child operating the program. The questions for grades one through three involve only addition and subtraction. Multiplication is added for fourth grade, and in fifth grade, division is included. The school grade which the child enters is also used as a multiplier for the random numbers which are generated for the problems. The multiplier increases the difficulty of the problem as the child progresses in school. There are several names used in the story problems, but the name of the child running the program is always added to the list so that his or her name is used in at least some of the questions. A series of ten questions is asked, after which a summary of the number answered right and wrong is given. If your system includes a printer, optional hard copy of the questions asked and the child's answers may be obtained simply by turning on the printer. My children seem to try harder to provide the correct answers to the problems if they know the results are being printed.

The Variables

The program uses several string arrays to provide the variety of text lines used to form the story problems. The string array variables and their uses are shown in Table 1.

B$(*)	the first line of the story
I$(*)	the item used in the story
C$(*)	the container used for the item
A$(*)	the action taken with the item
E$(*)	the question line of the story
NM$(*)	names used in the story
M$(*)	messages used for incorrect actions
AC$(*)	action needed to solve the problem

Table 1

Additionally, string variable N$ stores the child's name and X$, AN$, and SC$ are used as temporary variables to accept answers or provide responses. The numeric variables used are shown in Table 2.

V1	random selection for B$(*)
V2	random selection for E$(*)
V3	random selection for I$(*)
V4	random selection for C$(*)
V5	random selection for A$(*)
V6	random selection for NM$(*)
N1	top number of the problem
N2	bottom number of the problem
F	flags type of problem: 1 = add, 2 = subtract, 3 = multiply, 4 = divide
GR	school grade of child
RT	right answers
WR	wrong answers
QN	current question number
AN	child's answer given
CA	correct answer to the problem
FT	printer flag for the first question

Table 2

How it Works

Program lines 50 through 70 are housekeeping chores which clear memory space for string variables, define the integer variables, and read the data into the various arrays. Line 80 is the actual entry point of the program which sets the right or wrong answer counters, the question number to one, and the first time flag to zero. Program line 100 clears the screen and asks the child to type his or her name. If the name variable is null, the question is asked again. If the child responds with QUIT, the program terminates.

Building the Story

The beginning and ending lines and the name to be used in the story are selected in line 140. The container and item variables are selected in program lines 150 and 160. The highest allowable action as determined by school grade is selected in program lines 170-220. The numbers to be used in the story problem are selected and tested in lines 230 and 240. In order to keep the wording in the stories correct, the numbers used are never less than two. Lines 250 and 260 make sure that a larger number is not to be subtracted from or divided into a smaller one.

Finding the Answer

The story, depending upon the math action necessary in the question, is displayed by lines 270-470. All displays are generated in the 32-character-per-line mode (CHR$(23)) to help the child read the questions. Once the story has been displayed, the child is asked to decide what action must be taken to solve the problem presented. Until the child correctly determines the proper action, he or she cannot continue with the problem. A wrong answer at this stage will provide only a visual prompt, but is not considered as a right or wrong answer. Once the proper action is determined by the child, the answer to the question is requested by line 510. When the child responds, the correct answer is determined by program lines 520-550 and then compared with the child's answer in lines 570 and 580. Please note that for division problems, the correct answer and the child's answer are both rounded to one decimal place to keep the problems simple. The degree of difficulty in division problems may be altered by removing the rounding function on line 550 or by changing the rounding factor (10) to 100 or 1000 or 10000.

Printing the Results

Lines 590-660 are for those of you who wish to use a printer with this program. If you do not have or don't intend to use a printer you can eliminate these lines. If your printer is not connected via the expansion interface parallel port you will have to change line 590 to some other method of determining when the printer is ready to accept data. If the printer is turned on, the question and answer will be printed after the child has answered.

The Score

After 10 questions have been answered, the tally of right and wrong answers is displayed, and the child may choose to run the program again or to stop. If the child elects to continue, the program is restarted at line 80.

Expanding the Program

The data for the program lines, articles used, names, etc. is stored in program lines 700-770. Expanding the program is easy once you have worked out additional sentences and phrases which will fit in well and make sense with the other possible story line combinations. Simply increase the array dimension for the phrase or article you wish to expand and add the words in the data line to which they apply. I hope your children have as much fun running this program as mine do.

Sample Run

```
NIKKI OWNS
  37 CUPBOARDS
CONTAINING
  64 PEANUTS EACH.
HOW MANY PEANUTS DOES
NIKKI HAVE?
TO SOLVE THIS PROBLEM YOU MUST;
    (1) ADD                          (2) SUBTRACT
    (3) MULTIPLY                     (4) DIVIDE
(ENTER NUMBER OF ANSWER)? 3

NIKKI OWNS
  37 CUPBOARDS
CONTAINING
  64 PEANUTS EACH.
HOW MANY PEANUTS DOES
NIKKI HAVE?
AFTER YOU MULTIPLY
  YOUR ANSWER IS? 2368
```

Sample Printout

```
CARLA'S STORY MATH QUESTIONS AND ANSWERS FOR GRADE 5

TAMMY HAS 67 DOLLARS
AND MUST PUT THEM IN 18 BAGS.
HOW MANY DOLLARS ARE IN EACH BAG?
ANSWER GIVEN = 3.7, THE CORRECT ANSWER
 = 3.7                                  CARLA'S ANSWER IS CORRECT

DAVID OWNS 17 MIRRORS
AND MUST PACK THEM IN 6 TRUCKS.
HOW MANY MIRRORS ARE IN EACH TRUCK?
ANSWER GIVEN = 2.7, THE CORRECT ANSWER
 = 2.8                                  CARLA'S ANSWER IS WRONG

JOE OWNS 2 CUPBOARDS
CONTAINING 68 DOLLARS EACH.
HOW MANY DOLLARS DOES JOE HAVE?
ANSWER GIVEN = 136, THE CORRECT
ANSWER = 136                            CARLA'S ANSWER IS CORRECT

GEORGE OWNS 48 RECORDS
AND MOTHER GIVES 31 RECORDS
HOW MANY RECORDS DOES GEORGE HAVE?
ANSWER GIVEN = 79, THE CORRECT ANSWER
 = 79                                   CARLA'S ANSWER IS CORRECT

RALPH HAS 74 COO COO CLOCKS
AND THEN WINS 32 COO COO CLOCKS
HOW MANY COO COO CLOCKS DOES RALPH
HOLD?
ANSWER GIVEN = 106, THE CORRECT
ANSWER = 106                            CARLA'S ANSWER IS CORRECT

CAROL HAS 29 CUPBOARDS
OF 54 RECORDS EACH.
HOW MANY RECORDS DOES CAROL POSSESS?
ANSWER GIVEN = 1566, THE CORRECT
ANSWER = 1566                           CARLA'S ANSWER IS CORRECT

CARLA HAS 43 SPOONS
AND DISTRIBUTES THEM BETWEEN 7 TRUCKS.
HOW MANY SPOONS ARE IN EACH TRUCK?
ANSWER GIVEN = 6, THE CORRECT ANSWER
 = 6.1                                  CARLA'S ANSWER IS WRONG
```

continued

```
JOE OWNS 41 SPOONS
AND MUST PUT THEM IN 11 WASH TUBS.
HOW MANY SPOONS ARE IN EACH WASH TUB?
ANSWER GIVEN 3.7, THE CORRECT ANSWER
 = 3.7                                        CARLA'S ANSWER IS CORRECT

CARLA WILL EARN 45 DOGS
AND SISTER LOSES 21 DOGS
HOW MANY DOGS DOES CARLA HOLD?
ANSWER GIVEN = 24, THE CORRECT ANSWER
 = 24                                         CARLA'S ANSWER IS CORRECT

GEORGE WAS GIVEN 78 DOLLARS
AND IS PAID 7 DOLLARS
HOW MANY DOLLARS DOES GEORGE OWN?
ANSWER GIVEN = 85, THE CORRECT ANSWER
 = 85                                         CARLA'S ANSWER IS CORRECT
```

Program Listing 1. *Story Math*

```
 10 :
    ' STORY MATH
 20 :
    ' TRS-80 LEVEL II
 30 :
    ' W. KLUNGLE   #110780-1
 40 :
    '
 50 CLEAR 5000:
    DEFINT W,R,D,V
 60 DIM B$(4),E$(4),I$(8),C$(6),A$(12),NM$(10),M$(4),AC$(4)
 70 FOR R = 1 TO 4:
     READ B$(R):
     NEXT R:
    FOR R = 1 TO 4:
     READ E$(R):
     NEXT R:
    FOR R = 1 TO 8:
     READ I$(R):
     NEXT R:
    FOR R = 1 TO 6:
     READ C$(R):
     NEXT R:
    FOR R = 1 TO 12:
     READ A$(R):
     NEXT R:
    FOR R = 1 TO 10:
     READ NM$(R):
     NEXT R:
    FOR R = 1 TO 4:
     READ M$(R):
     NEXT R:
    FOR R = 1 TO 4:
     READ AC$(R):
     NEXT R
 80 RT = 0:
    WR = 0:
    QN = 1:
    FT = 0
 90 RANDOM
100 CLS :
    N$ = "":
    PRINT CHR$(23); TAB(11);"HELLO!":
    PRINT :
    PRINT "  WELCOME TO A MATH ADVENTURE.":
    PRINT :
    PRINT :
    PRINT :
    INPUT "WHAT IS YOUR NAME ";N$:
    IF N$ = ""
     THEN
      100:
     ELSE
      IF N$ = "QUIT"
       THEN
        END
110 NM$( RND(10)) = N$
120 PRINT @320,"WHAT SCHOOL GRADE IS ";N$:
    INPUT "IN THIS YEAR ";GR$:
    GR = VAL(GR$):
    IF GR > 6
     THEN
    GR = 6
```

Program continued

```
130 CLS :
    X$ = "":
    PRINT CHR$(23);"OK ";N$;",":
    PRINT "ARE YOU READY FOR":
    PRINT "QUESTION NUMBER ";QN;:
    INPUT "  ";X$:
    IF LEFT$(X$,1) < > "Y"
     THEN
      680
140 V1 = RND(4):
    V2 = RND(4):
    V6 = RND(10)
150 V3 = RND(8):
    IF V3 < 2
     THEN
      150
160 V4 = RND(6):
    IF V4 < 2
     THEN
      160
170 F = 1:
    IF GR < 4
     THEN
      V5 = RND(6)
180 IF GR = 4
     THEN
      V5 = RND(9)
190 IF GR > 4
     THEN
      V5 = RND(12)
200 IF V5 > 3
     THEN
      F = 2
210 IF V5 > 6
     THEN
      F = 3
220 IF V5 > 9
     THEN
      F = 4
230 N1 = RND(GR * 26):
    IF N1 < 2
     THEN
      230
240 N2 = RND(GR * 12):
    IF N2 < 2
     THEN
      240
250 IF N1 < N2 AND F = 2
     THEN
      RANDOM :
      GOTO 230
260 IF N1 < N2 AND F = 4
     THEN
      RANDOM :
      GOTO 230
270 CLS :
    PRINT CHR$(23)
280 PRINT NM$(V6);B$(V1)
290 IF F > 2
     THEN
      360
300 PRINT N1;I$(V3)
310 PRINT A$(V5)
320 PRINT N2;I$(V3)
330 PRINT :
    PRINT "HOW MANY";I$(V3);"DOES"
340 PRINT NM$(V6);E$(V2);" ?"
350 GOTO 480
360 IF F > 3
```

```
     THEN
      43Ø
37Ø PRINT N2;C$(V4)
38Ø PRINT A$(V5)
39Ø PRINT N1;I$(V3);"EACH."
40Ø PRINT :
    PRINT "HOW MANY";I$(V3);"DOES"
41Ø PRINT NM$(V6);E$(V2);" ?"
42Ø GOTO 48Ø
43Ø PRINT N1;I$(V3)
44Ø PRINT A$(V5)
45Ø PRINT N2;C$(V4);"."
46Ø PRINT :
    PRINT "HOW MANY";I$(V3)
47Ø PRINT "ARE IN EACH"; LEFT$(C$(V4), LEN(C$(V4)) - 2);" ?"
48Ø PRINT :
    PRINT "TO SOLVE THIS PROBLEM YOU MUST;":
    PRINT "  (1) ADD        (2) SUBTRACT":
    IF GR > 3
     THEN
      PRINT "  (3) MULTIPLY";:
      IF GR > 4
       THEN
        PRINT "   (4) DIVIDE"
49Ø PRINT @768,"";:
    INPUT "(ENTER NUMBER OF ANSWER)  ";AC$:
    AC = VAL(AC$):
    IF AC < > F
     THEN
      PRINT @832,M$( RND(4)):
      GOTO 49Ø
50Ø FOR R = 576 TO 896 STEP 64:
     PRINT @R, STRING$(32," ");:
     NEXT R
51Ø PRINT @576,"AFTER YOU ";AC$(F):
    INPUT " YOUR ANSWER IS ";AN$:
    AN = VAL(AN$):
    QN = QN + 1
52Ø IF F = 1
     THEN
      CA = N1 + N2
53Ø IF F = 2
     THEN
      CA = N1 - N2
54Ø IF F = 3
     THEN
      CA = N1 * N2
55Ø IF F = 4
     THEN
      CA = N1 / N2:
      CA = INT(CA) + ( INT((CA - INT(CA)) * 1Ø) / 1Ø):
      AN = INT(AN) + ( INT((AN - INT(AN)) * 1Ø) / 1Ø)
56Ø PRINT :
    PRINT
57Ø IF AN = CA
     THEN
      PRINT "YOUR ANSWER IS CORRECT !":
      RT = RT + 1:
      SC$ = "CORRECT"
58Ø IF AN < > CA
     THEN
      PRINT "SORRY, YOUR ANSWER IS WRONG !":
      WR = WR + 1:
      SC$ = "WRONG"
59Ø IF PEEK(14312) > 127
     THEN
      FOR R = 1 TO 9ØØ:
       NEXT R:
      GOTO 67Ø
6ØØ IF FT = Ø
```

Program continued

```
     THEN
      LPRINT N$;"'S  STORY MATH QUESTIONS AND ANSWERS FOR GRADE";GR:
      LPRINT " ":
      LPRINT " ":
      FT = 1
610 IF F > 2
     THEN
      630
620 LPRINT NM$(V6);B$(V1);N1;I$(V3):
    LPRINT A$(V5);N2;I$(V3):
    LPRINT "HOW MANY";I$(V3);"DOES ";NM$(V6);E$(V2);"?":
    GOTO 660
630 IF F > 3
     THEN
      650
640 LPRINT NM$(V6);B$(V1);N2;C$(V4):
    LPRINT A$(V5);N1;I$(V3);"EACH.":
    LPRINT "HOW MANY";I$(V3);"DOES ";NM$(V6);E$(V2);"?":
    GOTO 660
650 LPRINT NM$(V6);B$(V1);N1;I$(V3):
    LPRINT A$(V5);N2;C$(V4);".":
    LPRINT "HOW MANY";I$(V3);"ARE IN EACH"; LEFT$(C$(V4), LEN(C$(V4)
    ) - 2);" ?"
660 LPRINT "ANSWER GIVEN =";AN;",  THE CORRECT ANSWER =";CA;
    TAB(50);N$;"'S ANSWER IS ";SC$:
    LPRINT " "
670 IF QN < 11
     THEN
      130
680 CLS :
    PRINT CHR$(23);"STORY MATH":
    PRINT :
    PRINT :
    PRINT "YOU HAD ";RT;" CORRECT ":
    PRINT "AND ";WR;" INCORRECT ANSWERS.":
    PRINT :
    PRINT :
    INPUT "WOULD YOU LIKE TO TRY AGAIN ";AN$:
    IF LEFT$(AN$,1) = "Y"
     THEN
      80
690 CLS :
    PRINT CHR$(23):
    PRINT :
    PRINT :
    PRINT :
    PRINT TAB(10);"BYE ";N$:
    END
700 DATA " HAS "," WAS GIVEN "," WILL EARN "," OWNS "
710 DATA " HAVE"," POSSESS"," OWN"," HOLD"
720 DATA " APPLES "," DOLLARS "," PEANUTS "," COO COO CLOCKS "," DOG
    S "," RECORDS "," SPOONS "," MIRRORS "
730 DATA " BOXES "," BAGS "," WASH TUBS "," CUPBOARDS "," SHELVES ",
    " TRUCKS "
740 DATA "AND MOTHER GIVES "," AND IS PAID ","AND THEN WINS ","WHEN
    A MAN TAKES ","THEN DAD REMOVES ","AND SISTER LOSES ","OF ","CON
    TAINING ","WITH ","AND MUST PUT THEM IN ","AND MUST PACK THEM IN
     ","AND DISTRIBUTES THEM BETWEEN "
750 DATA "JOE","CAROL","SALLY","SUE","DAVID","GEORGE","RALPH","POSIE
    ","TOM","TAMMY"
760 DATA "SORRY, THAT'S NOT QUITE RIGHT.","BETTER READ IT AGAIN!","N
    O, LOOK IT OVER AGAIN!","PLEASE BE MORE CAREFUL!"
770 DATA "ADD","SUBTRACT","MULTIPLY","DIVIDE"
```

Smile—TRS-80 Loves You

by J. David McClung

People communicate continuously with facial expressions. A TRS-80 can, too. I have found that one of the greatest problems in writing educational programs for children can be overcome through the use of faces drawn on the computer with graphic blocks. I have purchased several educational programs for my children but have discovered that all of them require the children to read. Although my children enjoy the programs, I get tired of having to read the instructions and responses to them. To overcome the problem, I have written a series of programs which the children can use without adult supervision.

The common feature of the programs is a graphic face drawn in the upper right corner of the screen with graphic blocks. The face has three expressions. A neutral face indicates that some response is expected from the child. A smiling face shows that the child responded correctly. An incorrect response gets a frown. The Faces subroutines are shown in Program Listing 1. These lines are included in each program that uses the faces. To use the program you must merge lines 8000 through 8550 with another program with these features:

- The first program command should be GOTO 8500.
- The active program must begin at line 1000.
- Whenever a neutral face is needed, include GOSUB 8200.
- Whenever a smiling face is needed, include GOSUB 8100.
- Whenever a frowning face is needed, include GOSUB 8300.

Each of the remaining program listings can be merged with Program Listing 1 to make an educational program:

Count (Program Listing 2)—This program is the favorite of my five-year-old daughter, Jenny. The program places up to twenty graphic blocks on the CRT screen. Jenny is expected to press the white ENTER key the same number of times.

Number Series (Program Listing 3)—This game is slightly more advanced than Count, but my five year old plays it without help from me. The program prints three consecutive numbers on the screen. The child must respond by selecting the next number from the keyboard.

Alphabet (Program Listing 4)—similar to Number Series except that three consecutive letters of the alphabet are displayed rather than numbers. The child is required to respond with the next letter of the alphabet.

Odd1Out (Program Listing 5)—displays four graphic blocks. Three of the blocks are identical. The child is expected to respond with the number of the block that is different.

Months (Program Listing 6)—displays the names of eleven months. The child is expected to spell the name of the missing month correctly.

Days (Program Listing 7)—displays the names of six days of the week. The child is expected to correctly spell the name of the seventh day.

Addition (Program Listing 8)—teaches the child to add.

Subtraction (Program Listing 9)—teaches the child to subtract.

Multiply (Program Listing 10)—teaches the child to multiply.

My children have gotten hours of enjoyment from these programs. There is nothing like a smiling face to brighten the day.

Program Listing 1. *Faces subroutines*

```
8000 :
     ' FACE SUBROUTINES
8010 :
     ' THIS SUBROUTINE IS DESIGNED TO PROVIDE THE GRAPHIC FACES FOR S
     EVERAL EDUCATIONAL PROGRAMS
8020 :
     ' USE "GOTO 8500" AS THE FIRST COMMAND IN THE PROGRAM:  USE A DA
     TA STATEMENT FOR LINE 1000 OF THE EDUCATIONAL PROGRAM.
8030 :
     ' (C) 1980 BY DAVE MCCLUNG, RICHARDSON, TEXAS
8100 :
     ' HAPPY FACE
8110 P = 41:
     FOR X = 1 TO 7:
      PRINT @P,H$(X);:
      P = P + 64:
      NEXT X
8120 RETURN
8200 :
     ' NEUTRAL FACE
8210 P = 41:
     FOR X = 1 TO 7:
      PRINT @P,N$(X);:
      P = P + 64:
      NEXT X
8220 RETURN
8300 :
     ' SAD FACE
8310 P = 41:
     FOR X = 1 TO 7:
      PRINT @P,S$(X);:
      P = P + 64:
      NEXT X
8320 RETURN
8400 :
     ' DATA FOR FACES
8402 DATA 32,32,32,176,140,140,131,131,131,131,131,131,131,140,140,17
     6,32,32,32,32,32,184,131,32,32,144,160,32,32,32,32,32,144,160,32
     ,32,131,164,144,32,168,129,32,32,32,130,129,32,32,32,32,32,130,1
     29,32,32,32,32,169,32
8404 DATA 149,32,32,32,32,32,32,32,136,176,132,32,32,32,32,32,32,32,3
     2,149,138,144,32,32,131,164,144,32,32,32,32,32,160,152,131,32,32
     ,32,154,32,32,130,137,164,176,176,130,131,131,131,131,131,129,32
     ,160,176,140,131,32,32
8406 DATA 32,32,32,32,32,131,131,131,140,140,140,134,131,131,131,32,3
     2,32,32,32
8408 DATA 32,32,32,176,140,140,131,131,131,131,131,131,131,140,140,17
     6,32,32,32,32
8410 DATA 32,184,131,32,32,160,144,32,32,32,32,32,160,144,32,32,131,1
     64,144,32
8412 DATA 168,129,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,169
     ,32
8414 DATA 149,32,32,32,32,32,32,32,136,143,132,32,32,32,32,32,32,32,3
     2,149
8416 DATA 138,144,32,32,32,32,32,176,176,176,176,176,152,32,32,32,32,
     32,154,32
8418 DATA 32,130,137,164,176,32,130,32,32,32,32,32,32,32,160,176,140,
     131,32,32
8420 DATA 32,32,32,32,32,131,131,131,140,140,140,134,131,131,131,32,3
     2,32,32,32
8422 DATA 32,32,32,176,140,140,131,131,131,131,131,131,131,140,140,17
     6,32,32,32,32
8424 DATA 32,184,131,32,32,152,164,32,32,32,32,32,152,164,32,32,131,1
     64,144,32
```

Program continued

```
8426 DATA 168,129,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,169
     ,32
8428 DATA 149,32,32,32,32,32,32,32,136,131,132,32,32,32,32,32,32,32,3
     2,149
8430 DATA 138,144,32,32,32,32,176,152,140,140,140,140,164,176,32,32,3
     2,32,154,32
8432 DATA 32,130,137,164,176,131,32,32,32,32,32,32,32,32,163,176,140,
     131,32,32
8434 DATA 32,32,32,32,32,131,131,131,140,140,140,134,131,131,131,32,3
     2,32,32,32
8500 :
     ' LOAD FACES
8510 CLEAR 1000:
     DIM A(20),H$(7),N$(7),S$(7)
8520 FOR X = 1 TO 7:
      FOR Y = 1 TO 20:
       READ A:
       H$(X) = H$(X) + CHR$(A):
       NEXT Y:
      NEXT X
8530 FOR X = 1 TO 7:
      FOR Y = 1 TO 20:
       READ A:
       N$(X) = N$(X) + CHR$(A):
       NEXT Y:
      NEXT X
8540 FOR X = 1 TO 7:
      FOR Y = 1 TO 20:
       READ A:
       S$(X) = S$(X) + CHR$(A):
       NEXT Y:
      NEXT X
8550 GOTO 1000
```

Program Listing 2. *Count*

```
 100 :
     ' COUNT TO TWENTY
 110 :
     ' (C) 1980  DAVE MCCLUNG, RICHARDSON, TEXAS
 120 CLS :
     PRINT @15, "COUNT TO TWENTY":
     PRINT :
     PRINT "AN EDUCATIONAL GAME FOR PRESCHOOL CHILDREN":
     PRINT :
     PRINT "(C) 1980  - DAVE MCCLUNG, RICHARDSON, TEXAS"
 130 GOTO 8500
1000 :
     ' ALWAYS INCLUDE A DATA STATEMENT AT 1000
1010 CLS :
     GOSUB 8100
1020 D = RND(20):
     ON D GOTO 1030,1040,1050,1060,1070,1080,1090,1100,1110,1120,1130
     ,1140,1150,1160,1170,1180,1190,1200,1210,1220
1030 Q$ = "O N E":
     GOTO 1230
1040 Q$ = "T W O":
     GOTO 1230
1050 Q$ = "T H R E E":
     GOTO 1230
1060 Q$ = "F O U R":
     GOTO 1230
1070 Q$ = "F I V E":
     GOTO 1230
1080 Q$ = "S I X":
     GOTO 1230
1090 Q$ = "S E V E N":
     GOTO 1230
1100 Q$ = "E I G H T":
     GOTO 1230
```

```
1110 Q$ = "N I N E":
     GOTO 1230
1120 Q$ = "T E N":
     GOTO 1230
1130 Q$ = "E L E V E N":
     GOTO 1230
1140 Q$ = "T W E L V E":
     GOTO 1230
1150 Q$ = "T H I R T E E N":
     GOTO 1230
1160 Q$ = "F O U R T E E N":
     GOTO 1230
1170 Q$ = "F I F T E E N":
     GOTO 1230
1180 Q$ = "S I X T E E N":
     GOTO 1230
1190 Q$ = "S E V E N T E E N":
     GOTO 1230
1200 Q$ = "E I G H T E E N":
     GOTO 1230
1210 Q$ = "N I N E T E E N":
     GOTO 1230
1220 Q$ = "T W E N T Y":
     GOTO 1230
1230 :
     ' PRINT BLOCKS
1240 FOR S = 1 TO D
1250  B = RND(862)
1260  IF ((B > 514) AND (B < 544)) OR ((B > 642) AND (B < 672))
      OR ((B > 770) AND (B < 800)) GOTO 1270 :
       ELSE
        GOTO 1250
1270  FOR T = 1 TO S:
       IF (B = A(T)) OR (B = A(T) + 1) OR (B = A(T) - 1) GOTO 1250
1280   NEXT T:
      A(S) = B
1290  PRINT @ B, CHR$(191);
1300  NEXT
1310 PRINT @458,Q$;:
     PRINT @396,D;
1320 GOSUB 8200
1330 :
     '  COUNT
1340 C$ = INKEY$:
     IF C$ = "" GOTO 1340
1350 E = 1
1360 FOR Z = 1 TO 150:
      C$ = INKEY$:
      IF C$ < > "" E = E + 1:
      GOTO 1360
1370  NEXT Z
1380 IF E = D GOSUB 8100:
     FOR X = 1 TO 1500:
      NEXT X :
      ELSE
       GOSUB 8300:
       FOR X = 1 TO 1500:
        NEXT X
1390 CLS :
     GOSUB 8100:
     GOTO 1020
1400 E = E + 1:
     GOTO 1360
1410 END
```

Program Listing 3. *Number Series*

```
100 CLS :
    PRINT "NAME THE NEXT NUMBER":
    PRINT
```

Program continued

```
 1Ø2 PRINT "A GAME FOR PRESCHOOLERS":
     PRINT
 1Ø3 PRINT "(C) 198Ø -- DAVE MCCLUNG, RICHARDSON, TEXAS"
 1Ø4 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE
 2ØØ GOTO 85ØØ
1ØØØ :
     ' ALWAYS INCLUDE A REMARK STATEMENT AT 1ØØØ
1Ø1Ø A$ = CHR$(191):
     B$ = CHR$(191) + CHR$(191):
     C$ = "     ## - ## - %%"
1Ø2Ø CLS :
     GOSUB 82ØØ
1Ø3Ø N = RND(2Ø):
     M = N - 2
1Ø4Ø PRINT @7ØØ,"";
1Ø5Ø IF N > 9 PRINT USING C$;M,M + 1,B$;:
      ELSE
       PRINT USING C$;M,M + 1,A$;
1Ø6Ø IF N < 1Ø F$ = "":
     GOTO 11ØØ
1Ø7Ø F$ = INKEY$:
     IF F$ = "" GOTO 1Ø7Ø
1Ø8Ø PRINT @ 7ØØ,"";:
     PRINT USING C$;M,M + 1,F$;
11ØØ G$ = INKEY$:
     IF G$ = "" GOTO 11ØØ
111Ø H$ = F$ + G$:
     Q = VAL(H$)
1115 PRINT @ 7ØØ,"";:
     PRINT USING C$;M,M + 1,H$;
112Ø IF Q = N GOSUB 81ØØ :
      ELSE
       GOSUB 83ØØ
113Ø FOR X = 1 TO 15ØØ:
      NEXT X
114Ø GOTO 1ØØØ
12ØØ END
```

Program Listing 4. *Alphabet*

```
 1ØØ CLS :
     PRINT "NAME THE NEXT LETTER ":
     PRINT
 1Ø2 PRINT "A GAME FOR PRESCHOOLERS":
     PRINT
 1Ø3 PRINT "(C) 198Ø -- DAVE MCCLUNG, RICHARDSON, TEXAS"
 1Ø4 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE
 2ØØ GOTO 85ØØ
1ØØØ :
     ' ALWAYS INCLUDE A REMARK STATEMENT AT 1ØØØ
1Ø1Ø C$ = "     %%  -  %%  -  %%  -  %%":
     B$ = CHR$(191)
1Ø2Ø CLS :
     GOSUB 82ØØ
1Ø3Ø N = RND(26):
     N = N + 64
1Ø31 M$ = CHR$(N):
     GOSUB 2ØØØ
1Ø32 O$ = CHR$(N):
     GOSUB 2ØØØ
1Ø33 P$ = CHR$(N):
     GOSUB 2ØØØ
1Ø4Ø PRINT @7ØØ,"";
1Ø5Ø PRINT USING C$;M$,O$,P$,B$
1Ø6Ø H$ = INKEY$:
     IF H$ = "" GOTO 1Ø6Ø
1Ø64 PRINT @7ØØ,"";
1Ø65 PRINT USING C$;M$,O$,P$,H$
```

```
1070 IF H$ = CHR$(N) GOSUB 8100:
      ELSE
       GOSUB 8300
1130 FOR X = 1 TO 1500:
      NEXT X
1140 GOTO 1000
1200 END
2000 IF N = 90
      THEN
       N = N - 25:
      ELSE
       N = N + 1
2010 RETURN
```

Program Listing 5. *Odd1Out*

```
 100 CLS :
     PRINT "ODD ONE OUT":
     PRINT
 110 PRINT "A GAME FOR PRESCHOOLERS":
     PRINT
 120 PRINT "(C) 1980 -- DAVE mCCLUNG, RICHARDSON, TEXAS
 130 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE
 200 GOTO 8500
1000 :
     ' ALWAYS INCLUDE A REMARK STATEMENT AT 1000
1010 CLS :
     N = RND(61):
     N = N + 129:
     V = RND(61):
     V = V + 129:
     IF N = V GOTO 1010
1020 B$ = "      1    2    3    4":
     A$ = CHR$(V):
     D$ = CHR$(N)
1030 C$ = "     %%   %%   %%   %%"
1035 GOSUB 8200
1040 PRINT @636,B$;
1050 PRINT @700,"";
1060 R = RND(4):
     ON R GOTO 1061,1062,1063,1064
1061 PRINT USING C$;A$,D$,D$,D$;:
     GOTO 1070
1062 PRINT USING C$;D$,A$,D$,D$;:
     GOTO 1070
1063 PRINT USING C$;D$,D$,A$,D$;:
     GOTO 1070
1064 PRINT USING C$;D$,D$,D$,A$;:
     GOTO 1070
1070 H$ = INKEY$:
     IF H$ = "" GOTO 1070
1080 IF VAL(H$) = R GOSUB 8100 :
      ELSE
       GOSUB 8300
1090 FOR X = 1 TO 1500:
      NEXT X
1100 GOTO 1000
1200 END
```

Program Listing 6. *Months*

```
100 CLS :
    PRINT "MONTHS":
    PRINT
101 PRINT "A GAME FOR GRADE SCHOOL CHILDREN":
    PRINT
```

Program continued

```
 102 PRINT "(C) 1980 -- DAVE MCCLUNG, RICHARDSON, TEXAS
 110 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE.
 200 GOSUB 8500
1000 :
     ' ALWAYS INCULDE A REMARK AT 1000
1005 CLS
1010 GOSUB 8200
1020 R = RND(12):
     GOSUB 3000:
     GOSUB 2000
1030 PRINT @810,"NAME THE MISSING MONTH";:
     PRINT @ 874,"";
1040 INPUT W$
1050 IF W$ = M$ GOSUB 8100 :
      ELSE
       GOSUB 8300
1060 FOR X = 1 TO 1000:
      NEXT X:
     GOTO 1000
1070 STOP
2000 :
     ' MONTHS
2001 IF R < > 1 PRINT @ 64,"JANUARY";
2002 IF R < > 2 PRINT @ 128,"FEBRUARY";
2003 IF R < > 3 PRINT @192, "MARCH";
2004 IF R < > 4 PRINT @256,"APRIL";
2005 IF R < > 5 PRINT @320,"MAY";
2006 IF R < > 6 PRINT @384,"JUNE";
2007 IF R < > 7 PRINT @448,"JULY";
2008 IF R < > 8 PRINT @512,"AUGUST";
2009 IF R < > 9 PRINT @576,"SEPTEMBER";
2010 IF R < > 10 PRINT @640,"OCTOBER";
2011 IF R < > 11 PRINT @704,"NOVEMBER";
2012 IF R < > 12 PRINT @768,"DECEMBER";
2013 RETURN
3000 :
     ' MONTH
3001 IF R = 1 M$ = "JANUARY"
3002 IF R = 2 M$ = "FEBRUARY"
3003 IF R = 3 M$ = "MARCH"
3004 IF R = 4 M$ = "APRIL"
3005 IF R = 5 M$ = "MAY"
3006 IF R = 6 M$ = "JUNE"
3007 IF R = 7 M$ = "JULY"
3008 IF R = 8 M$ = "AUGUST"
3009 IF R = 9 M$ = "SEPTEMPER"
3010 IF R = 10 M$ = "OCTOBER"
3011 IF R = 11 M$ = "NOVEMBER"
3012 IF R = 12 M$ = "DECEMBER"
3013 RETURN
```

Program Listing 7. *Days*

```
 100 CLS :
     PRINT "DAYS OF THE WEEK":
     PRINT
 101 PRINT "A GAME FOR GRADE SCHOOL CHILDREN":
     PRINT
 102 PRINT "(C) 1980 -- DAVE MCCLUNG, RICHARDSON, TEXAS
 110 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE.
 200 GOSUB 8500
1000 :
     ' ALWAYS INCULDE A REMARK AT 1000
1005 CLS
1010 GOSUB 8200
1020 R = RND(7):
     GOSUB 3000:
     GOSUB 2000
```

```
1030 PRINT @810,"NAME THE MISSING DAY";:
     PRINT @ 874,"";
1040 INPUT W$
1050 IF W$ = M$ GOSUB 8100 :
      ELSE
       GOSUB 8300
1060 FOR X = 1 TO 1000:
      NEXT X:
     GOTO 1000
1070 STOP
2000 :
     ' DAYS OF THE WEEK
2001 IF R < > 1 PRINT @ 64,"MONDAY";
2002 IF R < > 2 PRINT @ 128,"TUESDAY";
2003 IF R < > 3 PRINT @192, "WEDNESDAY";
2004 IF R < > 4 PRINT @256,"THURSDAY";
2005 IF R < > 5 PRINT @320,"FRIDAY";
2006 IF R < > 6 PRINT @384,"SATURDAY";
2007 IF R < > 7 PRINT @448,"SUNDAY";
2013 RETURN
3000 :
     ' MONTH
3001 IF R = 1 M$ = "MONDAY"
3002 IF R = 2 M$ = "TUESDAY"
3003 IF R = 3 M$ = "WEDNESDAY"
3004 IF R = 4 M$ = "THURSDAY"
3005 IF R = 5 M$ = "FRIDAY"
3006 IF R = 6 M$ = "SATURDAY"
3007 IF R = 7 M$ = "SUNDAY"
3013 RETURN
```

Program Listing 8. *Addition*

```
 100 CLS :
     PRINT "ADDITION":
     PRINT
 101 PRINT "A GAME FOR GRADE SCHOOL CHILDREN":
     PRINT
 102 PRINT "(C) 1980 -- DAVE MCCLUNG, RICHARDSON, TEXAS
 110 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE.
 200 GOSUB 8500
1000 :
     ' ALWAYS INCULDE A REMARK AT 1000
1005 CLS :
     GOSUB 8200
1010 A = RND(99):
     B = RND(99)
1020 P$ = "##":
     Q$ = "+##"
1030 PRINT @ 586,"";:
     PRINT USING P$;A;
1040 PRINT @ 649,"";:
     PRINT USING Q$;B;
1050 PRINT @ 713,"---";
1060 PRINT @ 777,"";:
     PRINT USING "  %%"; CHR$(191);
1070 H$ = INKEY$:
     IF H$ = "" GOTO 1070
1080 PRINT @ 778,"";:
     PRINT CHR$(191);:
     PRINT H$;
1090 IF (A + B) < 9 GOTO 1200
1100 J$ = INKEY$:
     IF J$ = "" GOTO 1100
1110 H$ = J$ + H$:
     PRINT @ 777,"";:
     PRINT USING "###"; VAL(H$);
1115 IF J$ = "0" PRINT @778,"0";
```

Program continued

```
1120 IF (A + B) < 99 GOTO 1200
1125 PRINT @ 777, CHR$(191);
1130 J$ = INKEY$:
     IF J$ = "" GOTO 1130
1140 H$ = J$ + H$:
     PRINT @ 777,"";:
     PRINT USING "###"; VAL(H$);
1200 IF (A + B) = VAL(H$) GOSUB 8100 :
      ELSE
       GOSUB 8300
1210 FOR X = 1 TO 1500:
      NEXT X
1220 GOTO 1000
1230 END
```

Program Listing 9. *Subtraction*

```
 100 CLS :
     PRINT "SUBTRACTION":
     PRINT
 101 PRINT "A GAME FOR GRADE SCHOOL CHILDREN":
     PRINT
 102 PRINT "(C) 1980 -- DAVE MCCLUNG, RICHARDSON, TEXAS
 110 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE.
 200 GOSUB 8500
1000 :
     ' ALWAYS INCULDE A REMARK AT 1000
1005 CLS :
     GOSUB 8200
1010 A = RND(99):
     B = RND(99):
     IF B > A GOTO 1010
1020 P$ = "##":
     Q$ = "-##"
1030 PRINT @ 586,"";:
     PRINT USING P$;A;
1040 PRINT @ 649,"";:
     PRINT USING Q$;B;
1050 PRINT @ 713,"---";
1060 PRINT @ 777,"";:
     PRINT USING "  %%"; CHR$(191);
1070 H$ = INKEY$:
     IF H$ = "" GOTO 1070
1080 PRINT @ 777,"";:
     PRINT USING "###"; VAL(H$);
1090 IF (A - B) < 9 GOTO 1200
1095 PRINT @778, CHR$(191);
1100 J$ = INKEY$:
     IF J$ = "" GOTO 1100
1110 H$ = J$ + H$:
     PRINT @ 777,"";:
     PRINT USING "###"; VAL(H$);
1120 IF (A - B) < 99 GOTO 1200
1130 J$ = INKEY$:
     IF J$ = "" GOTO 1130
1140 H$ = J$ + H$:
     PRINT @ 777,"";:
     PRINT USING "###"; VAL(H$);
1200 IF (A - B) = VAL(H$) GOSUB 8100 :
      ELSE
       GOSUB 8300
1210 FOR X = 1 TO 1500:
      NEXT X
1220 GOTO 1000
1230 END
```

Program Listing 10. *Multiply*

```
 100 CLS :
     PRINT "MULTIPLY":
     PRINT
 101 PRINT "A GAME FOR GRADE SCHOOL CHILDREN":
     PRINT
 102 PRINT "(C) 1980 -- DAVE MCCLUNG, RICHARDSON, TEXAS
 110 :
     ' MUST BE MERGED WITH "FACES" SUBROUTINE.
 200 GOSUB 8500
1000 :
     ' ALWAYS INCULDE A REMARK AT 1000
1005 CLS :
     GOSUB 8200
1010 A = RND(99):
     B = RND(9):
     C = RND(9):
     D = (B * 10) + C:
     IF A < D GOTO 1010
1020 P$ = "##":
     Q$ = "x##":
     W$ = CHR$(191)
1030 PRINT @ 333,"";:
     PRINT USING P$;A;
1040 PRINT @ 396,"";:
     PRINT USING Q$;D;
1050 PRINT @ 460,"---";
1060 L = 526:
     GOSUB 7000:
     H$ = L$
1080 PRINT @ 526,H$;:
     L = 525:
     GOSUB 7000:
     J$ = L$
1110 H$ = J$ + H$:
     PRINT @ 525,H$;
1120 IF (A * C) < 100 GOTO 1200
1130 L = 524:
     GOSUB 7000:
     J$ = L$
1140 H$ = J$ + H$:
     PRINT @ 524,H$;
1200 L = 589:
     GOSUB 7000:
     G$ = L$
1220 PRINT @ 589,G$;
1240 L = 588:
     GOSUB 7000:
     I$ = L$
1250 G$ = I$ + G$:
     PRINT @588,G$;
1260 IF (A * B) < 100 GOTO 1300
1270 L = 587:
     GOSUB 7000:
     I$ = L$
1280 G$ = I$ + G$:
     PRINT @587,G$;
1300 PRINT @ 650,"-----";
1310 L = 718:
     GOSUB 7000:
     I$ = L$
1320 P = VAL(G$) * 10 + VAL(H$)
1340 PRINT @718,I$;
1360 L = 717:
     GOSUB 7000:
     G$ = L$
1370 I$ = G$ + I$
1380 PRINT @717,I$;
1390 IF P < 100 GOTO 1500
```

Program continued

```
1400 L = 716:
     GOSUB 7000:
     G$ = L$
1410 I$ = G$ + I$
1420 PRINT @716,I$;
1430 IF P < 1000 GOTO 1460
1440 PRINT @715,",";:
     L = 714:
     GOSUB 7000:
     G$ = L$
1450 I$ = G$ + I$
1460 PRINT @714,"";:
     PRINT USING "#,###"; VAL(I$);:
     PRINT "    = YOUR ANSWER";
1470 PRINT @ 778,"";:
     PRINT USING "#,###";(A * D);:
     PRINT "    = CORRECT ANSWER"
1500 IF VAL(I$) = A * D GOSUB 8100 :
      ELSE
       GOSUB 8300
1510 FOR X = 1 TO 2500:
      NEXT X
1520 GOTO 1000
1530 END
7000 :
     ' BLINKING CURSOR
7010 PRINT @ L,W$;
7020 FOR X = 1 TO 15:
      L$ = INKEY$:
      IF L$ < > "" GOTO 7080
7030  NEXT X
7040 PRINT @ L," ";
7050 FOR X = 1 TO 15:
      L$ = INKEY$:
      IF L$ < > "" GOTO 7080
7060  NEXT X
7070 GOTO 7010
7080 RETURN
```

GAMES

Keno

Tie Attack

GAMES

Keno

by C. Brian Honess

The ancient Chinese game that we know today as keno is one of the most popular casino gambling games. The game is played on a card, not unlike a bingo card. The keno card consists of the first 80 integer numbers, arranged in eight rows of 10 numbers each. Figure 1 shows the typical keno card.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80

Figure 1. *Typical keno card*

Each of the numbers is called a spot, and you may play from one to fifteen spots. A spot is played by marking through it with a crayon, although in our

	NUMBER OF SPOTS MARKED														
NUMBER OF CATCHES	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3														
2		12	1	1											
3			42	4	2	1									
4				112	20	4	2								
5					480	88	24	9	4	2	1				
6						1480	360	92	44	20	8	5	1	1	
7							5000	1480	300	132	72	32	16	10	8
8								18000	4000	960	360	240	80	40	28
9									20000	3800	1800	600	720	300	132
10										25000	12000	1480	4000	1000	300
11											25000	8000	8000	3200	2600
12												25000	20000	16000	8000
13													25000	24000	24000
14														25000	25000
15															25000

Table 1. *Dollar payoff per dollar bet*

keno program, we'll just make the spot disappear. After you've chosen the spot or spots you want to play, the house draws exactly 20 numbers at random. Then you compare the spots you chose with the numbers that have been selected at random. Every time one of the numbers that you selected is the same as one of the random numbers drawn, it is called a catch. Depending upon the number of catches you have and the number of spots you marked, you win or lose, based on a payoff table. Table 1 shows a typical payoff table.

Notice in Table 1 that you don't have to get exactly the same number of catches as the number of spots marked in order to be a winner. For example, suppose you had decided to mark five spots. If those five spots were all in the 20 random numbers selected by the house, you'd win $480.00 for each $1.00 bet. You'd also win if only three or four of your selections were in the group of 20 numbers called. For three catches you'd win $2.00, and for four you'd win $20.00. Notice too, that there is an upper limit on the amount you can win, in this case, $25,000.00. This program is coded so that you can win $25,000.00 maximum for *each* $1.00 bet. If you'd like to reduce your chances of winning big and have your program conform more closely to the big casino games, insert the following line:

```
5675 IF WN > 25000 THEN WN = 25000
```

You're going to find out quickly that it is much more difficult to win than you would first suppose. When you play keno in the casinos there are usually special combination tickets that give you more ways to win. These specials are not included in this program, since they vary widely and are often extremely complex. Figure 2 gives an example of the screen display midway into a typical game.

```
             1     2     3     4     5     6     7     8     9    10
            11    12    13    14    15    16    17    18    19    20
            21    22    23    24    25    26    27    28    29    30
            31    32          34    35          37    38    39    40
            41    42    43    44    45    46    47    48    49    50
            51    52    53    54    55    56    57    58    59    60
            61    62    63    64    65    66    67    68          70
            71    72    73    74    75    76    77    78    79    80

BANKROLL = $100                                              BET = $50

CALLED NUMBERS: 63 18 33 36 19 68 78 80 26 11
                 4 23 77  9 10  2 27 15 47 69

NUMBER OF HITS: 3                                      YOU WON $2100
```

Figure 2. *Example of screen display for keno*

About the Keno Program

At the beginning of the Program Listing, 400 string locations are reserved, and the arrays are dimensioned. P holds the payoff table, S holds the spots you want to play, and RN holds the 20 random numbers chosen by the house. The payoff table is read, beginning in line 120. If the player wants instructions, the program executes the subroutine beginning at line 9000, and upon returning to the main part of the program, the player is bankrolled with $100.00 in line 140.

The subroutine at line 8000 draws the playing screen. Since we want an 8-by-10 playing card, the row numbers in line 8030 go from one to eight. The column number is adjusted in lines 8020, 8040, and 8080 to center the 10 columns of the keno card. After printing the amount in the player's bankroll, the program goes to the subroutine starting at line 5000, where the number of spots to be played, NS, and the bet, B, are determined. Starting in line 5180, the NS spots are selected, and there are appropriate traps to make sure you don't enter the same number twice or don't choose a number greater than 80.

The routine starting in line 5270 blanks out the spot that was chosen on the playing card. This is done by subtracting one from the number, dividing by 10, and looking at the integer part of the resulting number. This calculation gives the line number of the chosen spot on the playing card. For example, suppose that the number 45 is selected. (45 − 1) / 10 = 4.4, and if we take INT(4.4), this results in the row being identified as row 4. To find out why I had to subtract one from the spot number, I suggest you try a few numbers: 19, 20, and 21 will do nicely.

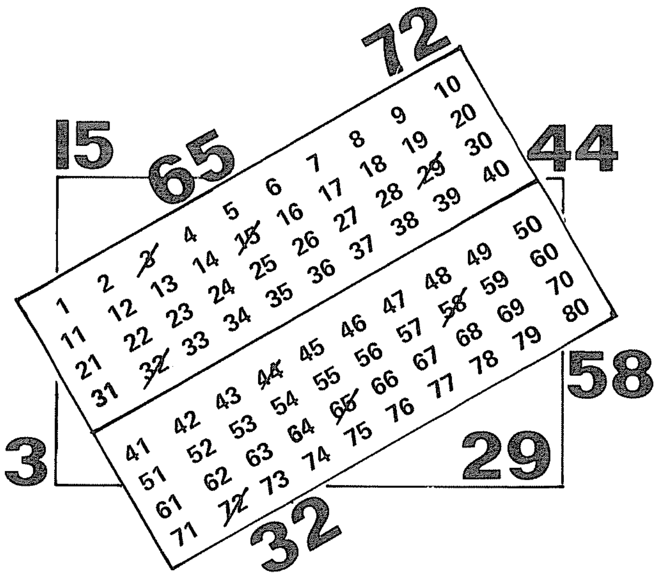
The remainder of this section of the program, through line 5330, calculates the column number in a similar fashion and blanks out that location on the screen. The routines at line 5340 and 5410 simply produce and print the 20 random numbers. The PRINT USING statement is used to nicely format the output. At this point, the program diverts control to the subroutine at line 6000, which simply holds the display on the screen until the space bar is pressed.

Returning to the routine starting at line 5530, the program now has to determine the number of catches, or hits, between the numbers selected by the player and the 20 numbers selected by the computer. A counter, NH (number of hits), is initialized to zero and then incremented by one each time a number shows up in both categories. This is accomplished in the nested loops in lines 5560 through 5600, after which the contents of NH are printed. Starting in line 5620, the payoff table is checked to see if there is a payoff for the number of hits versus the number of spots played. If there is, this amount is added to the bankroll in line 5770, and if not, it is subtracted in line 5710. Next, the program returns to play again, unless the bankroll has fallen to zero, in which case the "you lost it all" subroutine at line 4000 is executed.

A Final Word

Sections of this program are of special interest, such as those in which just a portion of the screen is altered, while some lines remain intact. This technique is especially obvious where the payoff table is printed, starting in line 9120. By carefully using PRINT@, PRINT TAB, and PRINT USING statements, the headings remain on the screen while the numbers below them change.

An excellent way to learn screen formatting is to get a blank screen video worksheet and go through the program by hand, filling in the worksheet at the appropriate locations as you encounter the various types of PRINT statements. You wouldn't have to do this with the paragraph of instructions from lines 9000 through 9100, but it would be instructive to do it from line 9110 through line 9420, and also for the main subroutine, from line 5000 through about line 5500.



Program Listing

```
  1 REM    ************************************************
  2 REM    *                                              *
  3 REM    *              <<<   KENO   >>>                *
  4 REM    *   BY:                                        *
  5 REM    *           C. BRIAN HONESS                    *
  6 REM    *           COL. OF BUS. ADM.                  *
  7 REM    *           UNIV. OF SO. CAROLINA              *
  8 REM    *           COLUMBIA, SC    29208              *
  9 REM    *                                              *
 10 REM   ************************************************
100 CLEAR 400
110 DIM P(73,3), S(15), RN(20)
120 REM  * READ PAYOFF TABLE *
121 FOR I = 1 TO 73
122  READ P(I,1),P(I,2),P(I,3)
123  NEXT I
125 CLS
126 PRINT @ 520, "WANT TO SEE INSTRUCTIONS (YES/NO)";:
    INPUT Y$
130 IF Y$ = "YES" GOSUB 9000
140 BR = 100.00
150 GOSUB 8000
160 GOSUB 5000
999 END
4000 REM  ***   YOU LOST IT ALL ROUTINE   ***
4010 CLS :
     PRINT @ 533, "YOU LOST YOUR BANKROLL"
4020 PRINT @ 593, "WANT TO PLAY AGAIN ( YES / NO )";:
     INPUT Y$
4030 IF Y$ = "YES"
      THEN
       140
4040 CLS :
     GOTO 999
5000 REM  ***   PLAY A GAME   ***
5010 REM  * CHOOSE SPOTS *
5020 PRINT @ 778,"HOW MANY SPOTS DO YOU WANT TO PLAY ( 1-15 )";
5030 INPUT NS
5040 IF NS < 16 AND NS > 0
      THEN
       5070
5050 PRINT @ 778, STRING$(48," ")
5060 GOTO 5020
5070 REM  * MAKE BET *
5080 PRINT @ 848, "HOW MUCH WOULD YOU LIKE TO BET";:
     INPUT B
5090 IF B < = BR
      THEN
       5150
5100 PRINT @ 973, "THAT'S MORE THAN YOU HAVE! -- RE-ENTER BET";
5110 FOR I = 1 TO 999 :
      NEXT I
5120 PRINT @ 968, STRING$(48," ");
5130 PRINT @ 848, STRING$(48," ");
5140 GOTO 5080
5150 PRINT @ 688, "BET = $"; B ;
5160 PRINT @ 848, STRING$(48," ");
5170 PRINT @ 778, STRING$(54," ");
5180 REM  * SELECT SPOTS TO PLAY *
5190 FOR I = 1 TO NS
5200  PRINT @ 772, STRING$(24," ");
5210  PRINT @ 772, "ENTER SPOT NO. "; I;
5220  INPUT S(I)
5230  IF S(I) > 80 OR S(I) < 1
       THEN
        5200
```

Program continued

```
5240  FOR J = I - 1 TO 1 STEP - 1
5250   IF S(I) = S(J)
        THEN
         5200
5260   NEXT J
5270  REM  * PUT 'BLANKS' IN SPOT'S NUMBER IN BOARD *
5280  Z1 = INT((S(I) - 1) / 10)
5290  Z2 = 77 + (Z1 * 64)
5300  Z3 = Z2 + (4 * ((S(I) - 1) - (Z1 * 10)))
5310  PRINT @ Z3, "    ";
5320  NEXT I
5330 PRINT @ 772, STRING$(24," ");
5340 REM  * PRODUCE 20 RANDOM NUMBERS *
5350 FOR I = 1 TO 20
5360  RN(I) = RND(80)
5370  FOR J = I - 1 TO 0 STEP - 1
5380   IF RN(I) = RN(J)
        THEN
         5360
5390   NEXT J
5400  NEXT I
5410 REM  * PRINT THE 20 RANDOM NUMBERS *
5420 PRINT @ 772, "CALLED NUMBERS: ";
5430 A$ = " ##"
5440 FOR I = 1 TO 10
5450  PRINT USING A$; RN(I);
5460  NEXT I
5470 PRINT @ 852, "";
5480 FOR I = 11 TO 20
5490  PRINT USING A$; RN(I);
5500  NEXT I
5510 GOSUB 6000
5520 PRINT @ 960, STRING$(63," ");
5530 REM  * DETERMINE NUMBER OF 'HITS' *
5540 PRINT @ 964, "NUMBER OF HITS: ";
5550 NH = 0
5560 FOR I = 1 TO 20
5570  FOR J = 1 TO NS
5580   IF S(J) = RN(I)
        THEN
         NH = NH + 1
5590   NEXT J
5600  NEXT I
5610 PRINT @ 980, NH;
5620 REM  * CHECK PAYOFF MATRIX TO SEE IF WIN *
5630 WN = 0
5640 FOR I = 1 TO 73
5650  IF NS < > P(I,1)
       THEN
        5690
5660  IF NH < > P(I,2)
       THEN
        5690
5670  WN = P(I,3) * B
5680  GOTO 5760
5690  NEXT I
5700 PRINT @ 1000, "SORRY .... YOU LOSE";
5710 BR = BR - B
5720 PRINT @ 644, STRING$(24," ");
5730 PRINT @ 644, "BANKROLL = $"; BR;
5740 FOR I = 1 TO 1500 :
      NEXT I
5745 IF BR < = 0
      THEN
       4000
5750 GOTO 150
5760 PRINT @ 1000, "YOU WON $"; WN;
5770 BR = BR + WN
5775 FOR I = 1 TO 1500 :
      NEXT I
```

```
5780 GOTO 150
6000 REM  ***   PRESS 'SPACE' WHEN READY   ***
6010 PRINT @ 960, "..................PRESS.........WHEN READY.......
     ............";
6020 FOR K = 1 TO 25:
      K$ = INKEY$:
      IF K$ < > ""
       THEN
        RETURN :
       ELSE
        NEXT K
6030 PRINT @ 985, "'SPACE'";
6040 FOR K = 1 TO 25:
      K$ = INKEY$:
      IF K$ < > ""
       THEN
        RETURN :
       ELSE
        NEXT K
6050 GOTO 6010
7000 REM  ***   PAYOFF TABLE   ***
7001 DATA 1,1,3
7002 DATA 2,2,12
7003 DATA 3,2,1,3,3,42
7004 DATA 4,2,1,4,3,4,4,4,112
7005 DATA 5,3,2,5,4,20,5,5,480
7006 DATA 6,3,1,6,4,4,6,5,88,6,6,1480
7007 DATA 7,4,2,7,5,24,7,6,360,7,7,5000
7008 DATA 8,5,9,8,6,92,8,7,1480,8,8,18000
7009 DATA 9,5,4,9,6,44,9,7,300,9,8,4000,9,9,20000
7010 DATA 10,5,2,10,6,20,10,7,132,10,8,960,10,9,3800,10,10,25000
7011 DATA 11,5,1,11,6,8,11,7,72,11,8,360,11,9,1800,11,10,12000,11,11,
     25000
7012 DATA 12,6,5,12,7,32,12,8,240,12,9,600,12,10,1480,12,11,8000,12,1
     2,25000
7013 DATA 13,6,1,13,7,16,13,8,80,13,9,720,13,10,4000,13,11,8000,13,12
     ,20000,13,13,25000
7014 DATA 14,6,1,14,7,10,14,8,40,14,9,300,14,10,1000,14,11,3200,14,12
     ,16000,14,13,24000,14,14,25000
7015 DATA 15,7,8,15,8,28,15,9,132,15,10,300,15,11,2600,15,12,8000,15,
     13,24000,15,14,25000,15,15,25000
8000 REM  ***   PLAYING SCREEN   ***
8010 CLS :
     PRINT
8020 X1 = 77 :
     X2 = 114 :
     N = 1
8030 FOR R = 1 TO 8
8040  FOR X = X1 TO X2 STEP 4
8050   PRINT @ X, N;
8060   N = N + 1
8070   NEXT X
8080  X1 = X1 + 64 :
      X2 = X2 + 64
8090  NEXT R
8100 PRINT @ 644, "BANKROLL = $"; BR
8999 RETURN
9000 REM  ***   INSTRUCTIONS   ***
9010 CLS :
     PRINT @ 153, "*** KENO ***" :
     PRINT :
     PRINT
9020 PRINT "    THE ANCIENT CHINESE GAME WE KNOW TODAY AS KENO, IS PL
     AYED"
9030 PRINT " ON A CARD WITH THE FIRST 80 INTEGERS, ARRANGED IN A MATR
     IX"
9040 PRINT " OF 8 ROWS AND 10 COLUMNS.  YOU MAY SELECT FROM ONE TO 15
      OF"
9050 PRINT " THESE 'SPOTS', AND IN A CASINO YOU'D SIMPLY MARK THROUGH
      THEM"
```

Program continued

```
9060 PRINT " WITH A CRAYON.  20 OF THE POSSIBLE 80 NUMBERS ARE THEN D
     RAWN"
9070 PRINT " AT RANDOM.  THEN, DEPENDING UPON THE NUMBER OF SPOTS THA
     T YOU"
9080 PRINT " MARKED THAT WERE AMONG THE 20 NUMBERS SELECTED, YOUR PAY
     OFF"
9090 PRINT " IS CALCULATED."
9100 GOSUB 6000
9110 CLS :
     PRINT
9120 PRINT @ 86, "*** PAYOFF TABLE ***" :
     PRINT
9130 PRINT TAB(18) "NUMBER     NUMBER     $ PAID"
9140 PRINT TAB(18) "MARKED     CALLED     /$ BET"
9150 R1 = 1 :
     R2 = 8
9160 X1 = 404  :
     X2 = 415  :
     X3 = 422
9170 FOR R = R1 TO R2
9180  PRINT @ X1,""; :
      PRINT USING "##"; P(R,1);
9190  PRINT @ X2,""; :
      PRINT USING "##"; P(R,2);
9200  PRINT @ X3,""; :
      PRINT USING "#####.##"; P(R,3);
9210  X1 = X1 + 64 :
      X2 = X2 + 64 :
      X3 = X3 + 64
9220  IF R = 73
       THEN
        9280
9230  NEXT R
9240 R1 = R1 + 8 :
     R2 = R2 + 8
9250 GOSUB 6000
9260 PRINT @ 960, STRING$(63," ");
9270 GOTO 9160
9280 X = 466
9290 FOR R = 1 TO 8
9300  PRINT @ X, STRING$(63," ");
9310  X = X + 64
9320  NEXT R
9400 PRINT @ 720,"WANT TO SEE IT AGAIN ( YES / NO )";:
     INPUT Y$
9410 IF Y$ = "YES"
      THEN
       9110
9420 CLS
9999 RETURN
```

Tie Attack

by Bob Menten

Tie Attack is a real-time, shooting gallery type game that was written for Level I, 4K and converted to Level II. Level I operation of the game requires only about 2K of the available memory, but the animated title and instructions use up most of the remaining memory. I have 233 bytes remaining when I load the game into my TRS-80. In order to get it all in 4K, everything must be abbreviated as shown in Appendix A. Also, forget that there is a space bar on your keyboard—except in PRINT statements.

Playing the Game

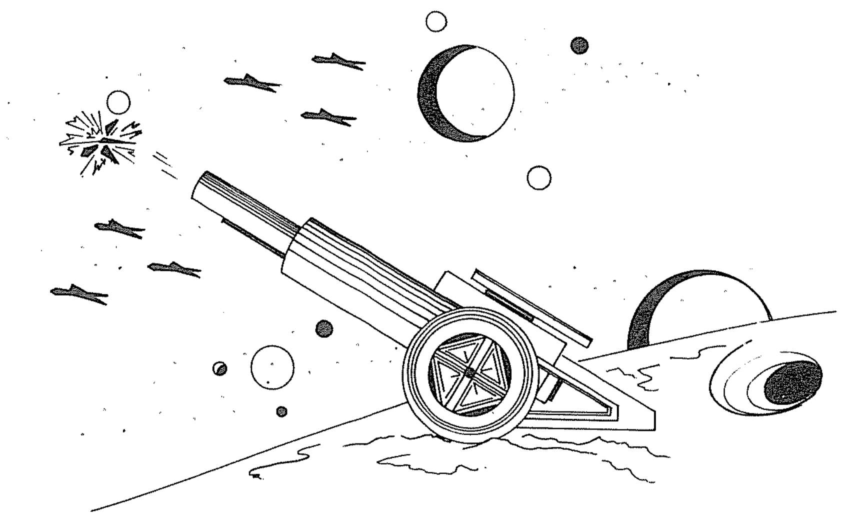
You are an artillery captain in command of an anti-neutron cannon on an isolated space outpost. Your outpost is being buzzed by a horde of Tie fighters. The Tie fighters fly by at different speeds and routes—covering every corner of your field of fire. Your mission: to destroy as many Tie fighters as possible with the 25 anti-neutron shells that remain in your armory. Fire your cannon by pressing the clear key. Theoretically, every Tie fighter will be within the range of your cannon. However, you will find it difficult to hit some of the faster moving fighters. If in doubt, it would be better to let a Tie fighter escape and save a shell for an easier target. A running tally is printed at the top of the screen. This will show shots left, misses, Tie fighters destroyed, and Tie fighters escaped.

Tie Animation

Line 1010 initializes the Tie animation sequence. Variable A determines which of six possible routes the Tie fighters will take. (See Table 2.) The apparent speed of the Tie fighters is determined by the value of E. In routes five and six, the Tie fighters are stepped E + 2 positions horizontally. In the four diagonal routes (A = 1, 2, 3, or 4), the Tie fighters are sequentially stepped one row vertically and E + 2 positions horizontally. Lines 1500 and 4100 to 4610 set the speed and the starting positions for the animation. Lines 2000 through 2070 perform the actual animation.

Shot Animation

After each step of Tie animation, the shot animation sequence is tested. If the shot sequence is not active (S = 1 in line 2100), the program is returned for another Tie animation step. If the shot sequence is already active, (S = 2), the shot (↑) is stepped upward two rows vertically (lines 2140-2160). Subroutine 8000 tests for a hit. A hit is sensed on two rows—the row where the shot (↑) is printed and the row directly below.



Line Number	Function
990	initialize game
1010	initialize animation sequence
1480	test for game end
1500	initialize Tie animation coordinates
2000-2040	animate Tie fighter
2050-2070	test for Tie animation stop—count escapes
2100	test for active cannon shot animation
2105	test for new cannon shot
2110	return to Tie animation if cannon shot is inactive
2115-2135	initiate new shot
2140	animate cannon shot
4100-4110	initialize route 1
4200-4210	initialize route 2
4300-4310	initialize route 3
4400-4410	initialize route 4
4500-4510	initialize route 5
4600-4610	initialize route 6
4700-4750	Tie animation stop coordinates
8000	test for hit
8020-8090	blowup tie fighter
9000-9020	delays
9050-9060	draw cannon
9065	count misses
9066-9070	print score

Table 1. *Line functions*

Variable	Functions
A	Tie Route
	A = 1 upper right to lower left
	A = 2 lower left to upper right
	A = 3 lower right to upper left
	A = 4 upper left to lower right
	A = 5 horizontal—right to left
	A = 6 horizontal—left to right
B	Tie start position
C	Tie stop position
E	Tie speed
F	speed steps
H	hit? (0 = no: 1 = yes)
I	count shots left
J	count Tie escapes
K	count misses
L	count hits
N	yes/no questions
P	FOR-NEXT loops
Q	print position—title and instructions
R	cannon shot print position
S	shot animation status
	S = 1 inactive
	S = 2 active
T	title X coordinates
U	title Y coordinates
V	determine yes/no answer
W	FOR-NEXT loops
X	FOR-NEXT loops
Y	yes/no questions
Z	FOR-NEXT loops

Table 2. *Variables list*

Tie Animation Speed Equalization

The purpose of lines 2005 and 2007 may not be readily apparent. After a step of Tie animation, the program tests for an active or inactive shot animation sequence. If the shot status is inactive, the program returns for another Tie animation step. However, if the shot status is active, the program goes through a number of additional steps (animating the shot) before another Tie animation step. The additional shot animating program steps add a little time delay between Tie animation steps. Without lines 2005 and 2007, the apparent speed of the Tie fighter would slow down very noticeably when the cannon is fired. A small time delay is added to the Tie animation sequence when the shot status is inactive.

Program Listing 1. *Tie Attack. This listing was originally in Level I and was converted to Level II. To use in Level I, change line 2105 to read: IFPOINT(60,42) = 0THEN2115 and abbreviate the program as indicated in Appendix A.*

```
  5 CLS :
    N = 0.01:
    Y = 1.1:
    A$ = ">-O-<"
100 Q = 186:
    FOR X = 1 TO 10:
     PRINT @Q,A$
105  GOSUB 9005
110  PRINT @Q,"     "
115  Q = Q + 59:
     NEXT X
120 GOSUB 9010
125 Q = 448:
    FOR X = 1 TO 19:
     PRINT @Q,A$
130  GOSUB 9005
135  IF Q = 475
      THEN
       155
140  PRINT @Q,"     "
145  Q = Q + 3:
     NEXT X
150 GOTO 160
155 PRINT @474,"TIE  ";:
    GOTO 145
160 Q = 128:
    FOR X = 1 TO 10:
     PRINT @Q,A$;
165  GOSUB 9000
170  IF Q = 478
      THEN
       190
175  PRINT @Q,"     ";
176  IF Q > 479
      THEN
       178
177  GOTO 180
178  Q = Q + 5:
     NEXT X
179 IF Q > 565
     THEN
      185
180 Q = Q + 70:
    NEXT X
185 GOTO 200
190 PRINT @Q,"ATTACK":
    GOTO 180
200 Q = 384:
    T = 48:
    U = 19
205 GOSUB 500
210 Q = 512:
    T = 48:
    U = 24
215 GOSUB 500
220 FOR X = 19 TO 24
225  SET(48,X):
     SET(49,X):
     SET(74,X):
     SET(75,X)
230  NEXT X
235 GOSUB 9020
240 CLS
```

```
245 PRINT @448,"DO YOU NEED INSTRUCTIONS-1=YES/0=NO";:
    INPUT V
250 IF V = 1
     THEN
      275
255 IF V = 0
     THEN
      270
260 GOSUB 9000
265 GOTO 245
270 V = 1.1:
    CLS :
    GOTO 9200
275 CLS :
    GOTO 700
500 FOR X = 1 TO 14:
     PRINT @Q,A$;
505  GOSUB 9005
510  PRINT @Q,"     ";
515  IF (X > = 7) * (X < = 11)
      THEN
       530
520  GOSUB 9005
525  GOTO 535
530  GOSUB 600
535  Q = Q + 4:
     NEXT X
540 RETURN
600 FOR P = T TO T + 5
605  SET(P,U)
610  NEXT P
615 T = T + 5
620 RETURN
700 PRINT "YOU ARE AN ARTILLERY CAPTAIN"
705 PRINT "COMMANDING AN ANTI-NEUTRON CANNON"
710 GOSUB 9010
715 PRINT "YOUR OUTPOST IS BEING BUZZED BY A HORDE OF TIE FIGHTERS"
720 GOSUB 9010
725 GOSUB 8100
730 GOSUB 9010
735 PRINT "YOUR MISSION---";:
    GOSUB 9010
740 PRINT "DESTROY AS MANY TIE FIGHTERS AS POSSIBLE"
745 GOSUB 9020
750 PRINT "YOUR CANNON IS FIRED BY PRESSING THE CLEAR KEY"
755 GOSUB 9020
760 PRINT "THERE ARE ONLY 25 ANTI-NEUTRON CANNON SHELLS"
765 PRINT "LEFT IN YOUR ARMORY"
770 GOSUB 9020
775 PRINT "FIRE YOUR CANNON CAREFULLY"
780 GOSUB 9020
785 PRINT "MAKE EVERY SHOT COUNT"
790 GOSUB 9010
795 GOSUB 9020
800 GOSUB 9020
810 V = 1.1:
    GOTO 9200
990 CLS :
    I = 25:
    J = 0:
    L = 0:
    K = 0:
    R = 0:
    B = 0
995 GOSUB 9010
1000 PRINT @R," ";:
     PRINT @B," ";:
     GOSUB 9050
1010 S = 1:
     H = 0:
```

Program continued

```
      A = RND(6):
      E = RND(4)
1480 IF I < = 0
      THEN
       9100
1490 GOSUB 9010
1500 ON A GOSUB 4100,4200,4300,4400,4500,4600
2000 PRINT @B,A$;
2005 IF S = 2
      THEN
       2010
2007 GOSUB 9005
2010 PRINT @B,"      ";:
     B = B + F
2020 ON A GOTO 2050,2060,2060,2050,2060,2050
2030 PRINT @B,A$;
2040 GOTO 2100
2050 IF B > C
      THEN
       2070
2055 GOTO 2030
2060 IF B < C
      THEN
       2070
2065 GOTO 2030
2070 J = J + 1:
     GOTO 1000
2100 IF S = 2
      THEN
       2140
2105 IF INKEY$ = CHR$(31) GOTO 2115
2110 GOTO 2000
2115 GOSUB 9050
2120 R = 862:
     S = 2:
     I = I - 1
2125 PRINT @R,"[";:
     GOSUB 8000
2130 IF H = 1
      THEN
       1000
2135 GOTO 2000
2140 PRINT @R," ";:
     R = R - 128
2142 IF R < 150
      THEN
       2165
2145 PRINT @R,"[";
2150 GOSUB 8000
2155 IF H = 1
      THEN
       1000
2160 GOTO 2000
2165 S = 1:
     GOTO 2000
4100 ON E GOSUB 4700,4700,4700,4710
4110 B = 186:
     F = 62 - E:
     RETURN
4200 ON E GOSUB 4730,4730,4730,4740
4210 B = 832:
     F = ( - 1) * (62 - E):
     RETURN
4300 ON E GOSUB 4730,4730,4730,4740
4310 B = 890:
     F = ( - 1) * (66 + E):
     RETURN
4400 ON E GOSUB 4700,4700,4700,4710
4410 B = 128:
     F = 66 + E:
      RETURN
4500 B = 314 + 64 * RND(7):
      C = B - 57
4510 F = (E + 2) * ( - 1):
      RETURN
4600 B = 256 + 64 * RND(7):
      C = B + 57
4610 F = E + 2:
      RETURN
4700 C = 889:
      RETURN
4710 C = 760:
      RETURN
4730 C = 128:
      RETURN
4740 C = 256:
      RETURN
4750 C = 320:
      RETURN
8000 IF (B < = R) * (B > = R - 4) + (B < = R + 64) * (B > = R
      + 60)
       THEN
        8020
8010 RETURN
8020 H = 1:
      L = L + 1
8030 PRINT @R," ";
8031 FOR P = 1 TO 3:
       PRINT @B,"I=*=I";
8032  GOSUB 9005
8033  PRINT @B,"     ";:
       GOSUB 9005
8034  NEXT P
8040 PRINT @B - 3,"-";:
      PRINT @B - 70,">";:
      PRINT @B - 64,"O";
8045 PRINT @B - 60,"-";:
      PRINT @B + 7,"<";
8050 GOSUB 9005
8055 GOSUB 8090
8060 PRINT @B - 11,"-";:
      PRINT @B - 138,">";:
      PRINT @B - 128,"O";
8065 PRINT @B - 121,"-";:
      PRINT @B + 14,"<";
8070 GOSUB 9005
8075 GOSUB 8090
8080 RETURN
8090 PRINT @B - 204," ":
      PRINT :
      PRINT :
      RETURN
8100 Q = 320
8105 FOR X = 1 TO 9:
       PRINT @Q,"     "
8110  Q = Q + 6:
       PRINT @Q,A$
8115  GOSUB 9000
8120  NEXT X
8125 PRINT @Q,"     ":
      GOTO 730
9000 FOR Z = 1 TO 200:
       NEXT Z:
      RETURN
9005 FOR Z = 1 TO 25:
       NEXT Z:
      RETURN
9010 FOR X = 1 TO 1000:
       NEXT X:
```

Program continued

```
      RETURN
9020 FOR X = 1 TO 2000:
       NEXT X:
      RETURN
9050 FOR X = 42 TO 44:
       SET(60,X):
       SET(61,X):
       NEXT X
9060 SET(59,44):
      SET(62,44)
9065 K = 25 - I - L
9066 PRINT @8,"SHOTS LEFT";I;
9067 PRINT @72,"MISSES";K;
9068 PRINT @27,"TIE FIGHTERS DESTROYED";L;
9069 PRINT @91,"TIE FIGHTERS ESCAPED";J;
9070 RETURN
9100 PRINT @448,"DO YOU WANT TO PLAY AGAIN-1=YES/0=NO";:
      INPUT V
9105 IF V = 1
       THEN
        V = 1.1:
        GOTO 9200
9106 IF V = 0
       THEN
        V = .01:
        GOTO 9200
9110 GOSUB 9000:
      GOTO 9100
9200 Q = 448
9205 CLS
9210 FOR X = 1 TO 10:
       PRINT @Q,A$
9215  GOSUB 9005
9220  IF (X = 4) * (V = Y)
        THEN
         9260
9225  IF (X = 5) * (V = Y)
        THEN
         9270
9230  IF (X = 5) * (V = N)
        THEN
         9280
9235  PRINT @Q,"     "
9240  Q = Q + 6
9245  NEXT X
9250 IF V = 1.1
       THEN
        990
9255 GOTO 9255
9256 GOSUB 9020
9257 GOTO 990
9260 PRINT @Q," GOOD"
9265 GOTO 9240
9270 PRINT @Q,"LUCK!"
9275 GOTO 9240
9280 PRINT @Q,"BYE  "
9285 GOTO 9240
```

GRAPHICS

Worksheet

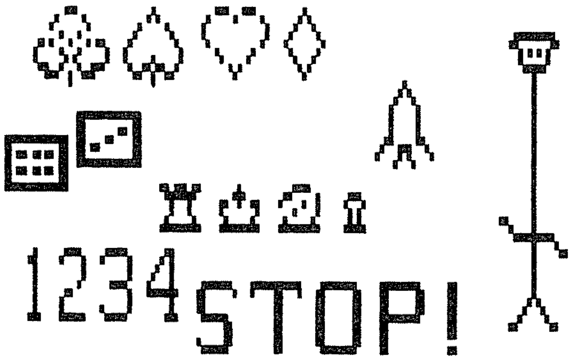
Curve Plotter

Worksheet

by Dan Rollins

There are three levels for working with graphics on the TRS-80. First, the SET and RESET commands offer a lot of flexibility, but can take forever to execute. Typing 10 or 20 lines of these commands is not my idea of fun. Next we have the POKE command. Individual graphics bytes may be read from DATA or included in program lines. POKEing these into screen memory is much faster, but flexibility is lost. Appendix C of the user's manual must be consulted to identify each of the chosen characters.

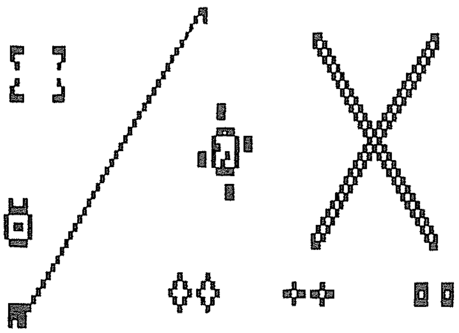
Finally, the preferred method is string packing, or as some call it, "fast graphics." This method uses strings which are built from those same TRS-80 graphic characters, with the possible inclusion of control codes such as CHR$ 24-27. These codes allow positioning of the cursor from within the string itself. The advantage to using this method is that the concatenated string may be placed anywhere on the screen with a simple PRINT@ command. It is also the fastest way to recall a complex figure which has a height of more than one line. With a little creative programming, you can achieve some fantastic visual effects! Example 1 might give you a few ideas. All of these examples were created using WRKSHEET. The printout is from my IDS 440 Paper Tiger.



Example 1. *Graphics ideas*

Traditionally, string packing is a tedious process. The would-be artist draws the desired figure on a Video Display Worksheet and, manual in hand, decodes individual bytes to be included in the string. These bytes and the necessary control codes are typed into DATA lines where they may be read and POKEd into a dummy string waiting on a program line. Finally, the DATA lines and POKE code are deleted, yielding the object of the whole process, a single line defining a string literal.

The TRS-80 can handle these mundane matters. WRKSHEET (Program Listing 1) handles all the messy work described above, with similar results, but the output is in a different form. Disk users end up with a sequential file for MERGEing with another program. Tape I/O is in the form of single-line, CLOADable program modules.


Example 2. *Intelligent LIFE forms*

How it Works

The doodle routine in lines 5000-5120 gives the user full control of a one-pixel cursor for creating the final design. It features repeat- and multiple-key action. In other words, holding down an arrow keeps the dot moving. Holding down vertical and horizontal arrows simultaneously causes motion along a diagonal. An important feature is the traverse function. While the spacebar is pressed, the cursor moves without changing the screen. Shifted arrows erase along the path of motion. Finally, the CLEAR key clears the screen with the CHR$ (128) character (this is necessary in some cases), and ENTER exits the doodle routine, sending control to the main program.

A few more notes on the doodle routine. The cursor is constantly flashing (line 5020), and moving off the screen returns you to the opposite side (lines 5080 and 5090). The logical variable is used in many of these functions to keep it concise. If it doesn't make any sense, just remember that the expressions enclosed in parentheses are evaluated as either −1 (true) or 0 (false). The current X,Y coordinates and the line and column number are printed on the top line. This is useful as an aid in transferring from a paper worksheet and will otherwise come in handy in some applications. For example, animation will require that placement of each frame be in the same position relative to screen line boundaries. If you are defining the alphabet as a set of billboard-sized characters, you'll want each to have the same height and width. Omit line 5025 if it's not needed, as it slows execution. I also suggest eliminating the remarks and using multiple-statement lines where possible.

The first section of the main program looks at the screen one line at a time. If it encounters a graphics byte, it saves the position in the integer array, A(X), and builds a temporary string (T$) of the rest of the line. When it reaches the end of the line, it counts backward until it finds a graphics character, truncating T$ so it includes only the minimum number of bytes. This is saved as an element of the A$ array which corresponds to the current line. The algorithm is repeated until the end of the screen is reached.

The next section processes the data just gathered. B$ is compiled from two components: (1) the graphics bytes in the A$ array and (2) the control codes required to reproduce the original image. This section also checks for an oversized string. I have set the limit at 225 characters, but it could conceivably be longer. That value leaves room for the line number, variable-name, and other necessities of the output string. Tall thin figures usually take less space than short wide ones, due to the backspacing requirement. A diagonal line, stretching from the top of the screen to the bottom, takes about 100 bytes of string space. An important facet of WRKSHEET is the edit feature. Should a string grow too long, it may be shortened or altered. The edit routine prints as much of the string as it was able to compile and jumps back to the doodle routine. Before saving the string, lines 8300-8360 provide for a review of the design. It may also do much to explain how WRKSHEET works. Follow the cursor as it spirals down a diagonal line if you want a real visual treat!

Lines 8520-8650 provide for disk output. Lines 9000-9190 are for Level II tape I/O. Either section may be omitted to suit your system. Both versions prompt for a line number which may range from 1 to 65529. Note that each ends with a COMPILE ANOTHER STRING (Y/N) query. A negative answer results in fall-through to program end, otherwise execution loops back to the doodle routine and starts over.

The disk section prompts for a variable name. LINEINPUT is very handy here as it allows for names such as GR$(1,9). If you intend to include a DEFSTR command in your target program, the dollar sign ($) will not be necessary. This prompt should be carefully considered, as EDIT won't be of any help later on. The EDIT mode thinks the graphic bytes are command codes and converts them to such after editing the line. You should *never* edit a line with graphics in a string! (LIST has the same delusion, but it is harmless.) You'll be asked for a filename in line 8520. Reply in standard TRSDOS format: filename /extension.password:drive#. I like to use the extension /GFX, though it may as well be /BAS or /TXT. The file created may be merged, loaded, and treated as a BASIC program. Notice that the line is executed only once.

Level II Tape

As a one-year veteran of Level II, my suggestion is to sell your car and buy a disk drive. Walking to work will be worth it. I can't imagine writing and debugging a complex program without the NEWDOS shorthands for LIST, EDIT, and the RENUM command.

The output routine employs a number of POKE gimmicks that will be of interest to the tape-bound hacker. My goal was to create a CLOADable file from the packed graphics string. Such a file could be appended to another program with a few POKEs from direct mode. I believe, however, that you'll probably want to use the program in Program Listing 2 for multiple mergings.

First a word about BASIC file format. Briefly, each line of Level II BASIC requires a Next Line Pointer (NLP), a line number, the program data, and an end of line marker of one byte of 00. The final line requires an end of program marker of three bytes of 00. The first program line is pointed to by the two-byte word at 16548,9. The last byte of program text may be located by subtracting two from the pointer at 16633,4. A CSAVE operation starts by sending a batch of zeros, a sync-byte, and the BASIC program header to the cassette. CSAVE then reads the bytes beginning at the address pointed to by the start-of-BASIC pointer, recording (verbatim) anything encountered up to the address held by the end-of-program pointer. CLOAD reads bytes until three zeros are encountered. It then adjusts the NLPs to jibe with the start-of-BASIC pointer, prints READY, and jumps to command mode.

After some reflection, you might realize that a CSAVE operation is not limited to BASIC text. Anything in memory may be sent to cassette. Just POKE the pointers with new values, type CSAVE"A" and watch the tape spin! But when including such instructions within a program, there's a catch. The pointer for the end of BASIC is also the spot where ROM looks to find where it's currently saving its list of simple variables. This one really stumped me for a while. BASIC evaluates LSB and POKEs it in, but by doing so, it changes the variables list pointer. When BASIC tries to execute the second POKE, it can't find it. Nobody likes to have furniture moved without a little warning! It may sound easy now, but the solution lies in using subscripted, or array variables. A different pointer keeps track of these variables.

Give B$ any value and send it to the output routine at line 9000. You'll wind up with a one-line program saved on tape. This simply defines a string literal on a line number of your choice. The difference between this and a data tape is that it may be CLOADed alone or appended to another program. Using the VARPTR of B$ as a reference, lines 9110-9130 calculate temporary values for the start and end of BASIC. It is very important to avoid any changes to 9100-9160. For example, don't change the program to CSAVE with a variable file name. This will affect the storage location for B$. Other changes can be equally disastrous. For a brief moment your BASIC interpreter has been tricked into believing that there is only one line of code in its memory. To show how thin the ice is, consider that the program is actually executing text that it isn't aware of! Mind the advice on line 9099, or become acquainted with the MEMORY SIZE prompt. After CSAVE"A" is issued, the pointers are immediately restored and execution either ends or branches to the doodle routine.

New Commands

After CSAVEing a tapeful of packed graphics, you'll want to include them in any of a number of display-oriented programs. Program Listing 2 POKEs a short machine-language routine into low memory to make the appending process considerably easier. No memory size need be saved as the code is written over your L3 ERROR jump vectors. One major advantage to this is simplicity of operation. To hide whatever BASIC text is in memory, just type PUT. Now a second program may be CLOADed, NEWed, EDITed, typed in, or executed with total disregard to the first. Another unused Disk BASIC command, GET, is used to restore the original or combined program. Furthermore, once in memory, these commands are available till powerdown or a jump to address 0 (e.g.: SYSTEM ENTER *? /0 ENTER).

The disadvantage of using the L3 ERROR jump-off area for program storage is that you might commit an L3 ERROR. However, in my experience with Level II, I never once received that message unintentionally. In fact, most of the keywords need to be in the right syntax for ROM to get even that far. Make your own choice (Program Listing 3 gives an alternative), but if you choose the L3 method, avoid typing CLOSE, LOAD, NAME, or KILL.

One note on appending program modules: Radio Shack's RENUM may be protected in high memory and used to straighten out line numbers. Line numbers must be in ascending sequence for the LIST and RUN processors, but they don't need to start out that way. Use the default values by typing ENTER three times in response to the prompts. This renumbers everything, starting with the first line and incrementing by 10 to the last line. Problems occur only when both modules of the merged program make reference to the same line number. Avoid this by first renumbering the base program with unlikely or even very high values. Type PUT, CLOAD the appendage, type GET, and renumber with default values.

For disk users, load your base program. Type MERGE"filename". The combined program can't be SAVEd in the ASCII format, as graphics bytes will be translated to their equivalent keywords.

Further WRKSHEET Information

Program logic requires that the design be drawn as one contiguous figure. A completely blank line is the cue to exit the compiler. Some trickery will allow blank Y-coordinate lines if they don't span an entire screen line. The processing time to compile a string is about 40 seconds, depending on the size and shape of the design.

All REMarks may be omitted. These have no effect on program logic. If you can think of any reason to do so, the PUT and GET commands of Program Listing 2 may be executed from a program line. Coincidentally, MERGE does the same as GET. This was not planned. It just happens that the entry point of MERGE is at a convenient address.

Ideas

The tape routine may be of greater significance than is immediately obvious; *TRS-80 BASIC Computer Games* includes a program called Animal (or Name My Animal). The program is tiny, but the concept is enormous. As you play the game, the program becomes more and more "intelligent." It builds a library of questions and responses which narrow the category, finally making a guess when it's down to one animal. Unfortunately, it starts out "dumb" on the next RUN. A creative programmer could modify the tape routine in WRKSHEET to create DATA lines instead of graphics strings. After some time working with Animal, the data array could be dumped to tape and appended to the program, effectively increasing its intelligence on each RUN.

Many data base management programs must read a master file from tape before updating any accounts. If this file is relatively permanent, why not include the values on DATA lines, saving scads of time and effort? Changes could be made with EDIT mode, and the verify (CLOAD?) option would also be helpful.

Suggested Improvements

Many times a single string just isn't long enough to hold the desired image. Include code to continue compilation on a second string. Just pass B$ to B1$, for example, and resume the loop. All the characters are preserved in the A$ array. A prompt would be needed for a second line number and variable name in the save routine. Alphanumeric data could be included in the string. Perhaps you could test for <SPACEBAR> <CLEAR> (K = 130) in the doodle routine, then GOSUB to an INKEY$ routine. As written, WRKSHEET compiles characters with a value greater than 128 (lines 8050 and 8110). Change this to 32 and replace GOSUB 6000 with CLS.

Lines 40-4998 are left open for a purpose. Here you may MERGE or type in code to create designs from other sources. Example: Graph a circular SINE function using the SET command. Then jump to 5000, edit in a smile and two eyes. Save the result and repeat the action, this time with one eye winking. Get the picture?

Program Listing 1. *Worksheet*

```
  * PROGRAM ID   : WRKSHEET/BAS
  * AUTHOR
    : DAN ROLLINS
  * DATE         : 11/10/80
2 ' *
    * PROGRAM ABSTRACT
:
      THIS PROGRAM AIDS IN THE CREATION OF "FAST-GRAPHICS"
    STRINGS.  THE USER DRAWS A DESIGN ON THE SCREEN USING
    THE ARROW KEYS.  THE SCREEN IS THEN PEEKED AND A
3 '  STRING VARIABLE IS COMPILED FROM THE RESULTING GRAPHICS
    BYTES.  THE PROGRAM INCLUDES OPTIONS FOR SAVING IT AS A
    PROGRAM LINE, USING EITHER DISK OR TAPE IO.
10 CLEAR 2000 :DEFINT X-Z :DIM A$(16),A(15)
30  GOSUB 6000 :GOTO 5000
4999 ' ALL-PURPOSE DOODLE ROUTINE
   ***********************
5000 X=60:Y=20
5020 IF POINT(X,Y),RESET(X,Y):FOR T=1TO10:NEXT:SET(X,Y)
     ELSE SET(X,Y):FOR T=1TO10:NEXT:RESET(X,Y)
5025 PRINT@0,"X=";X;" Y=";Y;" LINE #=";INT(Y/3);
              " COLUMN #=";INT(X/2);
5030 K=PEEK(14400)
5040 IF K=1 THEN 8000'                  * <ENTER> TO SAVE STRING
5050 IF K=2 GOSUB 6000:GOTO5000'        * <CLEAR> TO CLEAR-
                                      * SCREEN WITH GRAPHICS
5060 Y=Y+((KAND8)=8)-((KAND16)=16)'     * UP,DOWN ARROWS
5070 X=X+((KAND32)=32)-((KAND64)=64)'   * LEFT,RIGHT ARROWS
5080 X=X+(X>127)*127-(X<0)*127'         * SCREEN WRAP-AROUND
5090 Y=Y+(Y>47)*47-(Y<0)*47
5100 IF (KAND128)=128 THEN 5020'        * <SPACEBAR>+<ARROW>
                                        = TRAVERSE W/O CHANGE
5110 IF PEEK(14464)THEN RESET(X,Y)
     ELSE SET(X,Y)'                     * <SHIFT>+<ARROW>
                                        = ERASE DOT
5120 GOTO 5020'                         * GET NEXT INPUT
5130 '**********************

5999 ' * CLEAR THE SCREEN WITH GRAPHICS BYTES
6000 FOR J=0 TO 14 :PRINT STRING$(64,128);
        :NEXT :PRINT STRING$(63,128);:POKE 16383,128
        :RETURN

7999 ' * SEARCH THE SCREEN FOR GRAPHICS CHARACTERS
       * ARRAY A$(X) HOLDS ANY SUCH CHARACTERS ON LINE  X
       * ARRAY A(X) HOLDS THE POSITION OF THE FIRST CHARACTER
8000 PRINT @0,"****** COMPILING GRAPHICS STRING *****";
8010 FOR X=0 TO 15
8020    T$="" :FIRST =-1 :A$(X)="" :A(X)=0
8030    FOR Y=0 TO 63
8040      Z=PEEK(15360+X*64+Y)
8050      IF Z<129 AND FIRST=-1 GOTO 8080
8060      IF FIRST =-1 LET FIRST=Y
8070      T$=T$+CHR$(Z)
8080    NEXT Y
8090    IF FIRST=-1 THEN 8140
8100    LAST=63
8110    IF PEEK(15360+X*64+LAST)<129THEN LAST=LAST-1:GOTO 8110
8120    A$(X)=LEFT$(T$,LAST-FIRST+1) :A(X)=FIRST
8130    IF FP<>1 THEN SP= 64*X+A(X) :FP=1' * SP SAVES KEY POS.
8140 NEXT X :FP=0'                         * FOR EDIT ROUTINE
8199 ' * BUILD B$ FROM ARRAY A$(X)
      * INSERT DOWN-FEEDS AND BACK-SPACES AS NEEDED
        *
8200 B$=""
8210 FOR X=0 TO 15
8220    L=LEN(A$(X)) :L1=LEN(A$(X+1))
```

Program continued

```
8230     IF L=Ø THEN 829Ø'
               * SKIP IF BLANK LINE
8240     IF LEN(B$)+L>225 GOTO 84ØØ
         ELSE B$=B$+A$(X)'                * ADD SAVED CHARACTERS
825Ø     IF L1=Ø THEN X=16 :GOTO 829Ø     * EXIT IF LAST LINE
826Ø     LAST=A(X)+L:BACK=LAST-A(X+1)'    * CALCULATE NUMBER
                                            OF BACK-FEEDS NEEDED
827Ø     IF LEN(B$)+BACK>225 GOTO 84ØØ
828Ø     B$=B$+CHR$(26)+STRING$(BACK,24)'* ADD DOWN,BACK FEEDS
829Ø NEXT X
      : CLS: PRINT
      : PRINT" GRAPHICS STRING IS *   ";LEN(B$);"  * BYTES LONG"
83ØØ INPUT " DO YOU WANT TO REVIEW THE STRING (Y/N)";Q$
831Ø IF Q$="N" GOTO 85ØØ
832Ø CLS: PRINT@ SP,CHR$(14);'             * CURSOR ON
833Ø FOR X=1 TO LEN(B$)
834Ø   PRINT MID$(B$,X,1);
835Ø   FOR T=1 TO 75  :NEXT'               * SHOW BIZ
836Ø NEXT X :PRINT @Ø,;
837Ø INPUT" EDIT OR SAVE STRING (REPLY E OR S)";Q$
838Ø IF Q$<>"E" GOTO 85ØØ
      ELSE GOTO 843Ø
8399 ' *
       *  OVERSIZE STRING IS NOT LOST
       *   IT MAY BE EDITTED HERE
84ØØ PRINT :PRINT "        STRING TOO LONG
           ** COMPILATION ABORTED **

841Ø INPUT"RESTART PROGRAM OR EDIT STRING (REPLY R OR E)";Q1$
842Ø IF Q1$="R" GOTO 3Ø
843Ø GOSUB 6ØØØ:PRINT@ SP,B$; :GOTO 5ØØØ'     * BACK TO DOODLE
8499 ' *
       * STRING IS SAVED IN B$ , READY FOR FINAL PROCESSING
       *
85ØØ CLS :INPUT"OUTPUT TO TAPE OR DISK (REPLY T OR D)";Q1$
851Ø IF Q1$="T" THEN 9ØØØ ELSE IF Q1$<>"D" THEN 85ØØ
8519 ' ********************************
       *    THIS SECTION CREATES A     *
       *    SEQUENTIAL OUTPUT FILE     *
       *           TO DISK             *
       ********************************
852Ø IF OP<>1 THEN LINEINPUT"FILESPEC FOR OUTPUT  ";FS$
     : OPEN"O",1,FS$
     : OP = 1
853Ø INPUT"DESIRED LINE NUMBER FOR GRAPHICS STRING";LN
854Ø IF LN<1 OR LN>65529 OR INT(LN)<> LN GOTO 853Ø
855Ø LINEINPUT"DESIRED NAME FOR STRING VARIABLE
       (EXAMPLE :A$)  ";VN$
856Ø C$=STR$(LN)+" "+VN$+"="+CHR$(34)
      : IF LEN(C$)+LEN(B$) > 238 GOTO 84ØØ
857Ø PRINT:PRINT C$;"....STRING....";CHR$(34) :PRINT
     :INPUT "IS THIS FORMAT CORRECT (Y/N)";Q2$
858Ø IF Q2$<>"Y" THEN 853Ø
86ØØ B$=C$+B$+CHR$(34)+CHR$(13)'       * ADD QUOTE, <CR>
861Ø PRINT#1,B$
862Ø INPUT"COMPILE ANOTHER STRING (Y/N)"; Q3$
863Ø IF Q3$<> "N" THEN 3Ø
864Ø CLOSE 1
865Ø END
8999 '"*********************************
       * THIS SECTION IS FOR LEVEL II. *
       * A ONE - LINE BASIC PROGRAM IS *
       * CREATED IN B$.  IT IS USED AS *
       *         A CSAVE FILE          *
       *********************************
9ØØØ INPUT"NUMBER OF LINE TO HOLD GRAPHICS STRING";LN
9Ø1Ø IF LN<1 OR LN>65529 OR LN<> INT(LN) GOTO 9ØØØ
9Ø2Ø INPUT"NAME FOR THE STRING VARIABLE (EXAMPLE: A$)";VN$
9Ø3Ø PRINT : PRINT LN;" ";VN$;"="
            ;CHR$(34);"....STRING....";CHR$(34)
```

```
9040 PRINT : INPUT"IS THIS FORMAT CORRECT (Y/N)";Q3$
9050 IF Q3$ <>"Y" THEN 9000
9060 PRINT :INPUT"READY CASSETTE .... HIT <ENTER>";QA
9070 N2=INT(LN/256) :N1=LN - N2 * 256
9079 ' * C$ IS COMPRISED OF 2 "DUMMY" BYTES
       * FOR THE NLP, THE HEX VALUES FOR THE
       * LINE NUMBER, THE STRING NAME, THE BASIC
       * TOKEN FOR "=", AND A QUOTE MARK
       *
9080 C$="##"+ CHR$(N1)+ CHR$(N2)+ VN$+ CHR$(213)+ CHR$(34)
9090 IF LEN(B$)+ LEN(C$) > 240 GOTO 8500
9099  '***********************
    * VARIABLE ALLOCATION *
    *   INHIBITION ZONE   *
    *                     *
9100 B$=C$+B$+CHR$(34)+STRING$(3,0)'
   * ADD CLOSE QUOTE AND
                                          END-O-PROGRAM MARKER
9110 V=VARPTR(B$) :A(1)=PEEK(V+1)
     :A(2)=PEEK(V+2)'                    * CALCULATE START ,
9120 T=A(1)+A(2)*256 + LEN(B$)'          *  END OF STRING
9130 A(4)=INT(T/256) :A(3)=T-A(4)*256
9140 A(5)=PEEK(16548) :A(6)=PEEK(16549)'*    SAVE TRUE
9150 A(7)=PEEK(16633) :A(8)=PEEK(16634)'*    POINTERS
9159 ' * ADJUST THE START-O-BASIC AND
       * END-O-BASIC POINTERS TO THE START
       * AND END OF B$ WHICH IS IN HIGH MEMORY
       * CSAVE THE LINE AND RESTORE THE POINTERS.
       *
9160 POKE 16548,A(1) :POKE 16549,A(2)
    :POKE 16633,A(3) :POKE 16634,A(4)
    :CSAVE "A"
   :POKE 16548,A(5) :POKE 16549,A(6)
    :POKE 16633,A(7) :POKE 16634,A(8)
9169  '*                       *
    *  END  INHIBIT  ZONE *
    ***********************
9170 PRINT :INPUT"COMPILE ANOTHER STRING (Y/N)";Q3$
9180 IF Q3$<>"N" GOTO 30
9190 END
```

Program Listing 2

```
 1 :
   ' *THIS PROGRAM ADDS 2 COMMANDS TO YOUR
 2 :
   ' *LEVEL II BASIC DICTIONARY, PUT AND GET.
 3 :
   ' *
 4 :
   ' *PUT WILL "HIDE" ANY BASIC TEXT IN MEMORY.
 5 :
   ' *PROGRAMS MMAY THEN BE CLOADED, LISTED, & EXECUTED.
 6 :
   ' *
 7 :
   ' *GET RESTORES THE COMBINED PROGRAMS.
 8 :
   ' *LINE NUMBERS SHOULD BE IN ASCENDING ORDER.
10 X = 16768
20 READ Y:
   IF Y = - 1 GOTO 40
30 POKE X,Y:
   X = X + 1:
```

Program continued

```
   GOTO 20
40 STOP
50 DATA 141,65,217,42,249,64,43,43,34,164
60 DATA 64,217,201,217,33,233,66,24,245,-1
```

Program Listing 3

```
0 REM       CLOAD MODULE 1, THEN, AT READY, TYPE
1 REM   >X=16633:Y=16548:Z=PEEK(X):L=(Z<2):POKE Y+1,PEEK(X+1)+L:POK
  E Y,Z-2-L*256    <ENTER>
2 REM
3 REM       NOW CLOAD MODULE 2
4 REM           AND TYPE
5 REM   >POKE 16548,233: POKE 16549,66    <ENTER>
```

Curve Plotter

by David R. Cecil

How would you like to use the computer to create string art type figures? Or how about a constantly changing display of curves to fascinate and delight the kids on the block?

Here are six simple programs for screen displays. The names and equations of several exotic looking curves are included. The curves are in polar form, parametric equations, and in circles or lines. A scale factor must be input to determine the figure size (some suggested scale factors are included for experimentation).

The programs are written for the TRS-80's display area of 128 by 48. The origin of the curves is near the center of the screen (at 65, 23), the x-axis is horizontal, and the y-axis is vertical (the two axes are not displayed). To simplify the programs and allow the visible creation of the curves as the angle parameter changes, BASIC has been used for the curve generation instead of assembly language.

Polar Curves

The polar coordinate curves are plotted as if the screen were a sheet of polar coordinate paper with the pole at the center and the polar axis horizontal.

Program Listing 1 is to be used for curve equations, in polar form, with bounded extent. Enter the program and type RUN. When the input prompt, ?, appears, use 10, and then 2 for the second input prompt. A four-leaved rose (rhodonea) is sketched. To terminate the display, press the BREAK key. To obtain rhodonea with different numbers of leaves, try the following choices for the scale factor S and the constant A.

S	9	9	5	10	10	8
A	3	4	4	6	5	3.5

Be sure to use the BREAK key for every new curve. Note that the larger the S, the larger the figure; note also that when A is an integer, there are A leaves for A odd, but 2A leaves for A even. For non-integer A, the leaves overlap considerably and the figure appears incomplete. To complete the figure, a larger number than 2*PI in line 50 is needed (try 4*PI or some other multiple of 2*PI).

Now use R = 1 + A*COS(I) for line 60. If A = 1 (and S of perhaps 5), the figure is heart-shaped (a cardioid); if A = 2, we have a trisectrix (let S = 4); for A≤1, there is one loop, and for A>1, there are two loops. The curves are called limacons of Pascal.

Here are some other curves created by changing line 60 and by increasing 2*PI to 4*PI in line 50. Freeth's nephroid has R = 1 + 2*SIN(I/2 + A) for its equation (try S = 3, A = 10). Folia are given by R = COS(I)*(A*SIN(I)↑2 − 1)(try S = 10, A = 3), and the equation for Cayley's sextic is R = COS(I/3 + A)↑3(try S = 11, A = 3 and see where the loop is located). Changing the constant A in either Freeth's nephroid or Cayley's sextic rotates the figure. Experiment by observing the orientation of the loop in Cayley's sextic for the following S, A combinations.

S	10	9	9
A	1.7	2.4	0

These combinations have bounded values for the parameter I (such as 0 to 2π, or 0 to 4π). For interesting figures that have unbounded parameters for I, use Program Listing 2.

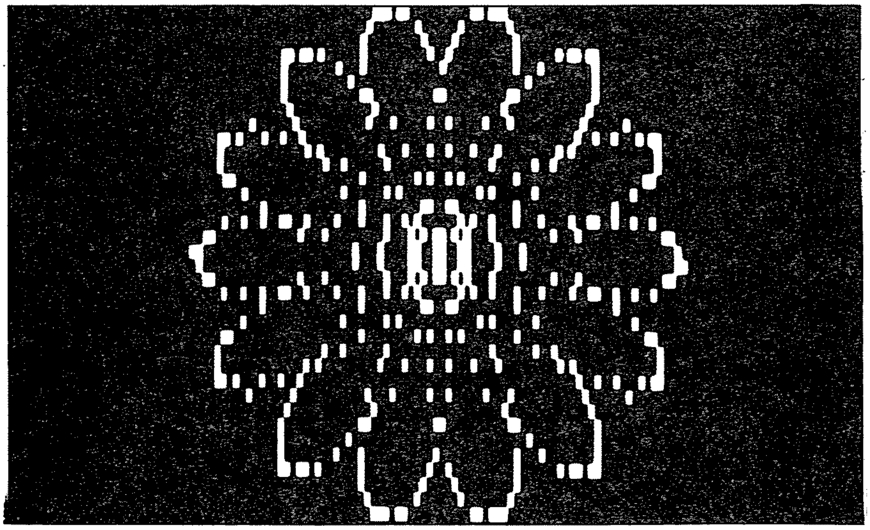


Photo 1. *A many-leaved rose using Program Listing 1 with S = 8, A = 3.5 and line 50 having 0 TO 4*PI STEP 2*PI/180. (Photograph by William R. Tinsley)*

Program Listing 2 creates a hyperbolic spiral (try S = 15, A = 1). A cross curve has R = A*SQR(1/COS(I)↑2) + (1/SIN(I)↑2) for line 50 (try S = 1, A = 3). You can increase the number of plotted points by changing the 60 in line 20 to 120. A folium of Descartes is plotted if line 50 is R = A*SIN(I)*COS(I)/(SIN(I)↑3 + COS(I)↑3). This has an interesting shape for S = 2, A = 5. Many other curves with polar equation forms can be handled with Program Listings 1 and 2. Some of these curves will be presented in terms of parametric equations.

Parametric Equation Curves

Program Listing 3 modifies Program Listing 1 to allow parametric equations. The program is designed to simulate plotting in the Cartesian plane with usual graph paper.

The parametric equations for the curves are in lines 60 and 70 with the x-coordinate called X1, the y-coordinate Y1, and the parameter I.

The equations already listed in lines 50 and 60 represent the roulettes called hypocycloids and give the curve traced by a point on the circumference of a circle rolling on the inside of a larger fixed circle. You might want to start experimenting using the following choices.

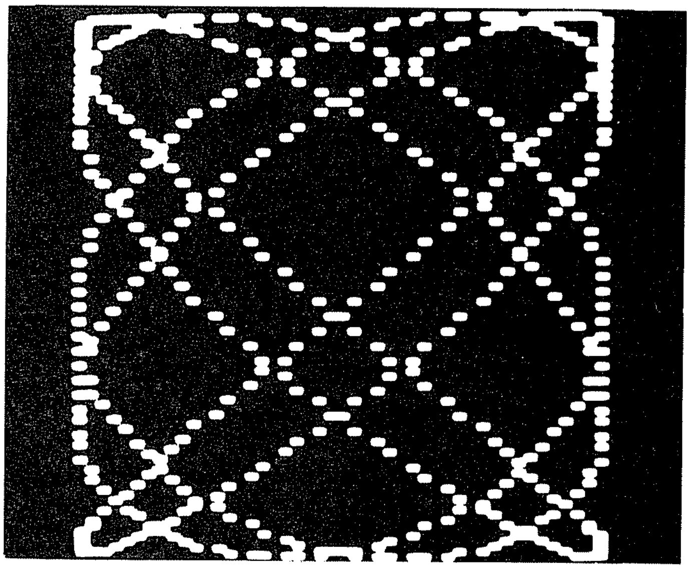


Photo 2. *Lissajous figure using Program Listing 3 with S = 11, A = .8, B = 2 and C = .9. (Photograph by William R. Tinsley)*

S	2	2	3	2.8	3	6
A	5	9	10	7	9	7.2
B	1	2	3	2	3	4.6

Did you notice that if (A − B)/B is an integer, there are A/B cusps (vertices)? Also if A − B is larger than B and if A − B and B are relatively prime (no common factors except ± 1), you see A cusps in B revolutions? Finally, did you note that when A and B are not integers, the figure is not completed symmetrically?

If you would like to see the larger fixed circle, add the following to Program Listing 3. Try this with some of the choices for S, A, and B suggested above.

```
 52 FOR J = 1 TO 2
 54 IF J = 1 THEN 60
 56 X1 = A/B*COS(I)
 58 Y1 = A/B*SIN(I)
 59 GOTO 80
105 NEXT J
```

Epicycloids are obtained by tracing a point on the circumference of a circle rolling on the outside of a larger fixed circle. To sketch these you need only change the + in line 60 to a − and change the A/B in lines 56 and 58 to (A − 2*B)/B. Some interesting patterns are obtained with the following choices:

S	2	2	1	3
A	5	9	16	9
B	1	2	2	3

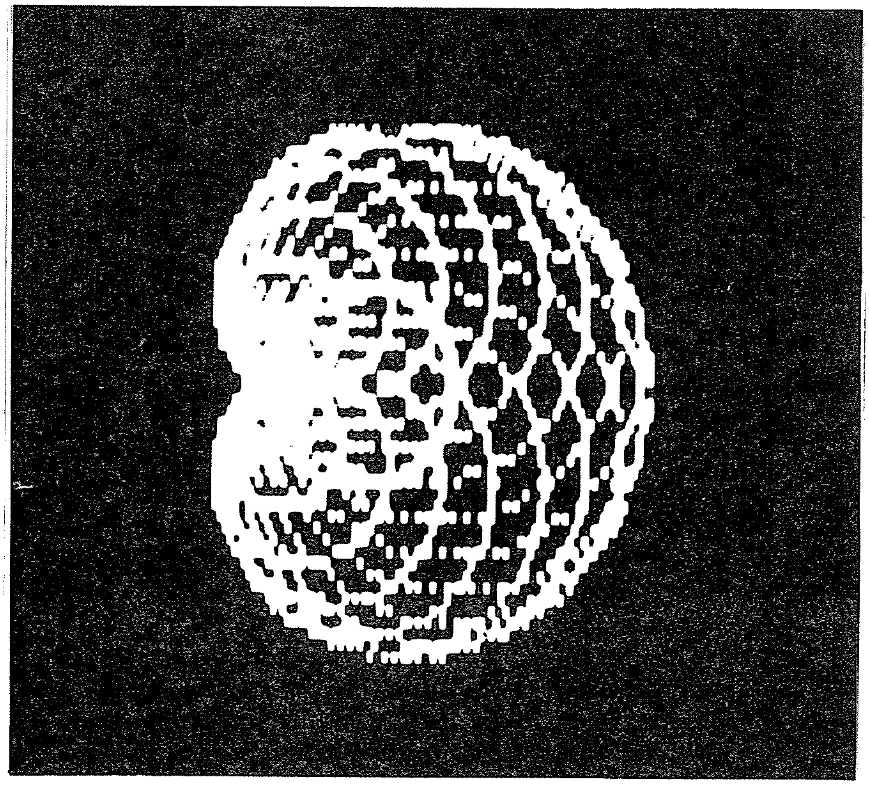


Photo 3. *Cardioid using Program Listing 4 with a fixed circle size of 9. (Photograph by William R. Tinsley)*

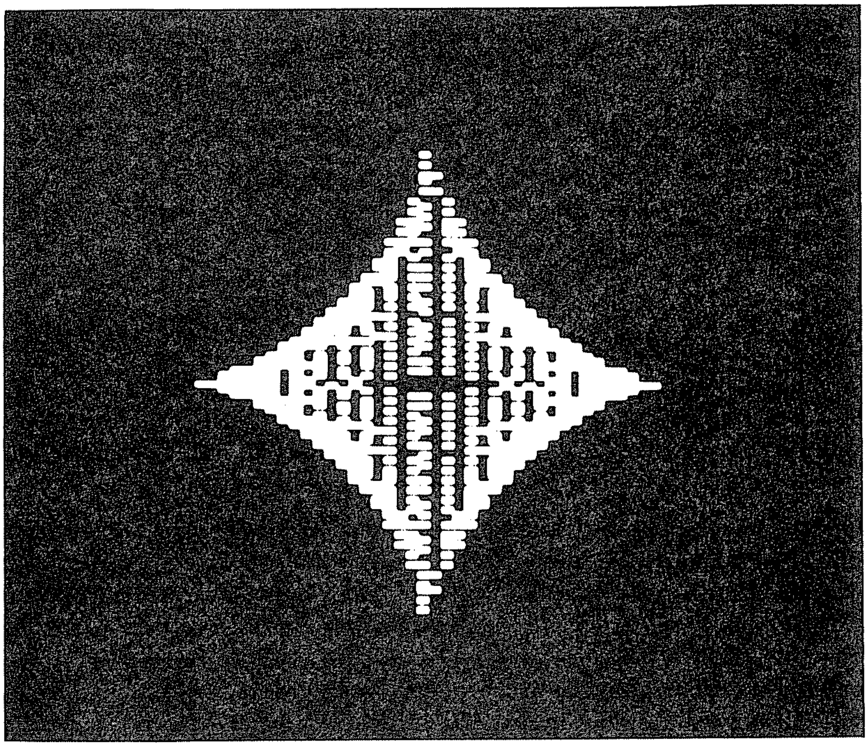


Photo 4. *Astroid using Program Listing 5 with A = 5.* (Photograph by William R. Tinsley)

Epicycloids have (A − 2*B)/B cusps when this number is an integer, and A − 2*B cusps (lying on the fixed circle) in B revolutions when A − 2*B and B are relatively prime with A − 2*B>B.

If you delete lines 52 through 59 and line 105 and change lines 60 and 70 to read:

```
60 X1=COS(I)/(1 + SIN(I)↑2)
70 Y1=X1*SIN(I)
```

the resulting figure is called a lemniscate of Bernoulli. A nice sized sketch is obtained with S = 9. (Note the constants A and B are not used here, but you can input A = 1 and B = 1 when asked by the program or delete the last half of line 30.)

Some very interesting curves, called Lissajous figures or Bowditch curves, can be obtained using parametric equations. Make the following changes in Program Listing 3.

```
30 INPUT "SCALE FACTOR=";
   S: INPUT "CONSTANTS
   A,B,C";A,B,C
50 FOR I=0 TO 10*PI STEP
   2*PI/90
```

```
60 X1=SIN(A*I+B)
70 Y1=C*SIN(I)
```

Here are some choices you might want to try:

S	12	10	11	12
A	.75	.4	.8	.67
B	0	.75	2	2.8
C	.8	1	.9	.7

String Art on the Computer

Constructing curves with the computer is done in much the same way that string art figures are made. To illustrate the possibilities let's construct a cardioid (the heart-shaped figure mentioned earlier) and an astroid (a four-pointed star, or hypocycloid of four cusps).

For the cardioid, begin with a fixed circle C and a fixed point PF (use the point on the circumference at the extreme left of the circle for PF). Then choose a number of points (X0,Y0) on C and draw circles with centers at (X0,Y0) and with radii equal to the distance between PF and (X0,Y0). The curve generated by these circles is the desired cardioid. Program Listing 4 creates the circles and the cardioid. You might try an A of 8 or 9 for a nice sized display. If X0 + A in line 110 is changed to X0 − A, the cardioid is turned 180° since the point PF is now on the extreme right of the circle.

Lines instead of circles will generate the four-pointed astroid star. With x- and y-axes positioned so the origin is at the center of the screen and a line RS of fixed length 4*A, we draw several copies of the line with R always on the x-axis and S always on the y-axis.

Program Listing 5 allows different choices for A (values between 4 and 7 give nice displays) and draws the RS lines two at a time, one above and one below the x-axis. The astroid can also be generated using ellipses since the envelope of the ellipses X = A*COS(I), Y = (1 − A)*SIN(I) is the astroid X = ¼*(3*COS(I) + COS(3*I)), Y = ¼*(3*SIN(I) − SIN(3*I)).

Computer Spirograph

The last program (see Program Listing 6) presents a panorama of hypocycloids, epicycloids, and rhodonea (roses) generated in random order and sizes. If the curve suddenly disappears, the random size is too big for the screen, but don't worry! The curve will reappear in a smaller size.

After each curve is constructed, the values for T, N, and S are displayed for a short period at the bottom of the screen. If you don't want these values shown, delete line 180. If you prefer fewer points drawn, change the step

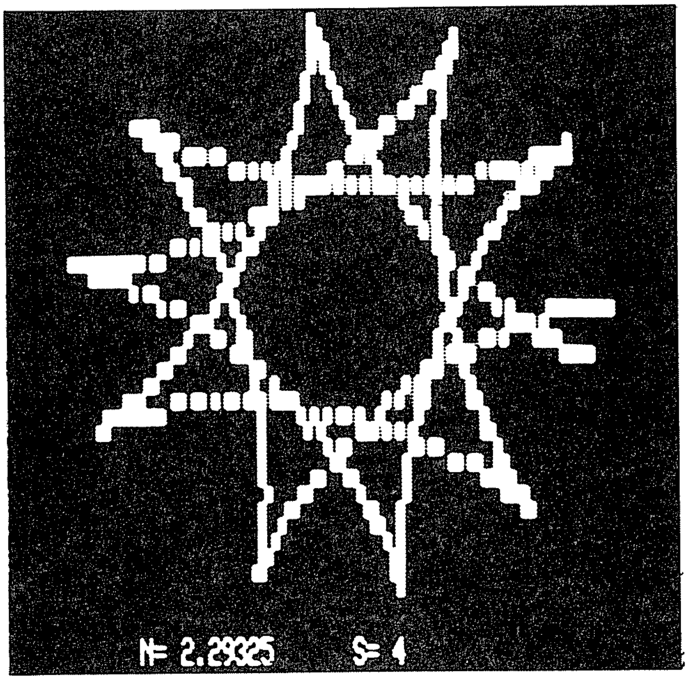


Photo 5. *Random pattern using Program Listing 6. The value of N and S is displayed. (Photograph by William R. Tinsley)*

size in line 70 to 2*PI/90 or 2*PI/60, etc.; more points may be obtained with 2*PI/360. In line 70 the parameter I makes three revolutions (0 to 6π). If more revolutions are desired, change the 6*PI to 10*PI (or 4*PI for fewer revolutions), etc.

References

A Book of Curves by E. H. Lockwood, Cambridge University Press, 1961.
A Catalog of Special Plane Curves by J. Dennis Lawrence, Dover Publications, 1972.

Program Listing 1

```
 10 ON ERROR GOTO 120
 20 PI = 3.14159
 30 INPUT "SCALE FACTOR=";S:
    INPUT "CONSTANT=";A
 40 CLS
 50 FOR I = 0 TO 2 * PI STEP 2 * PI / 180
 60  R = COS(A * I):
     REM  THIS LINE IS CHANGED FOR OTHER POLAR CURVES
 70  X = 65 + 4.9 * S * R * COS(I)
 80  Y = 23 - 2.3 * S * R * SIN(I)
 90  SET(X,Y)
100  NEXT I
110 GOTO 110
120 RESUME NEXT
```

Program Listing 2

```
 10 ON ERROR GOTO 130
 20 PI = 3.14159:
    W = 2 * PI / 60:
    I = W
 30 INPUT "SCALE=";S:
    INPUT "CONSTANT=";A
 40 CLS
 50 R = A / I:
    REM  THIS LINE IS CHANGED FOR OTHER POLAR CURVES
 60 FOR J = 1 TO 2.1:
     I = - I
 70  X = 65 + 4.9 * S * R * COS(I)
 80  Y = 23 - 2.3 * S * R * SIN(I)
 90  SET(X,Y)
100  NEXT J
110 I = I + W
120 GOTO 50
130 RESUME NEXT
```

Program Listing 3

```
 10 ON ERROR GOTO 130
 20 PI = 3.14159
 30 INPUT "SCALE=";S:
    INPUT "CONSTANTS A,B=";A,B
 40 CLS
 50 FOR I = 0 TO B * 2 * PI STEP 2 * PI / 90
 60  X1 = (A - B) / B * COS(I) + COS((A - B) / B * I)
 70  Y1 = (A - B) / B * SIN(I) - SIN((A - B) / B * I)
 80  Y = 23 - 2.3 * S * Y1
 90  X = 65 + 4.9 * S * X1
100  SET(X,Y)
110  NEXT I
120 GOTO 120
130 RESUME NEXT
```

Program Listing 4

```
10 ON ERROR GOTO 170
20 INPUT "SIZE OF FIXED CIRCLE";A
30 CLS
40 PI = 3.14159
50 FOR I = - PI TO PI STEP 2 * PI / 180
60  X1 = A * COS(I):
    Y1 = A * SIN(I)
70  SET(63,2.2 * X1,23 + Y1)
80  NEXT I
90 FOR J = - PI TO PI STEP 2 * PI / 20
100  XO = A * COS(J):
     YO = A * SIN(J)
110  R = SQR((XO + A) [ 2 + YO [ 2)
120  FOR K = - PI TO PI STEP 2 * PI / 90
130   X = XO + R * COS(K):
      Y = YO + R * SIN(K)
140   SET(63 + 2.2 * X,23 + Y)
150   NEXT K,J
160  GOTO 160
170  RESUME NEXT
```

Program Listing 5

```
10 ON ERROR GOTO 120
20 INPUT "A VALUE";A
30 CLS
40 Q = 4 * A * 19 / 20
50 FOR X = - Q TO Q STEP 4 * A / 20
60  FOR XX = X TO 0 STEP - X / 10
70   Y = (1 - XX / X) * SQR(16 * A [ 2 - X [ 2)
80   SET(63 + 2.4 * XX,23 + Y)
90   SET(63 + 2.4 * XX,23 - Y)
100  NEXT XX,X
110  GOTO 110
120  RESUME NEXT
```

Program Listing 6

```
10 ON ERROR GOTO 210
20 RANDOM
30 N = 7 * RND(0):
   T = RND(4)
40 S = RND(6)
50 CLS
60 PI = 3.14159
70 FOR I = 0 TO 6 * PI STEP 2 * PI / 180
80  ON T GOTO 90,100,120,130
90  X1 = N * COS(I) + COS(N * I):
    GOTO 110
100  X1 = N * COS(I) - COS(N * I)
110  Y1 = N * SIN(I) - SIN(N * I):
     GOTO 150
120  R = 3 + 2 * COS(N * I):
     GOTO 140
130  R = 5 * SIN(N * I)
```

Program continued

```
140  X1 = R * COS(I):
     Y1 = R * SIN(I)
150  X = 65 + 49 / S * X1:
     Y = 23 + 23 / S * Y1
160  SET(X,Y)
170  NEXT I
180 PRINT @960,"T=";T,"N=";N,"S=";S;
190 FOR J = 1 TO 1250:
     NEXT J
200 GOTO 30
210 S = S + 1:
    RESUME 50
```

HARDWARE

Chip Tester

Build a Light Pen

Chip Tester

by Joe Magee

Ever see someone selling bargain ICs at a sidewalk sale? Sure you have, and usually the vendor thinks they are good, but isn't sure. In a similar experience I recently obtained quite a few 4118 memory chips. These are very nice 1K × 8 static RAMs made by Mostek, but I don't have anything that uses them, so testing them is not very simple. Knowing that I also come across bargain 2114s from time to time, I decided to do something to let me test the memory devices. By the time the project was done, I learned how to test memory using a BASIC program, and using BASIC and assembly language together. I designed a test circuit, and devised a scheme to add other peripherals to my TRS-80.

Hardware

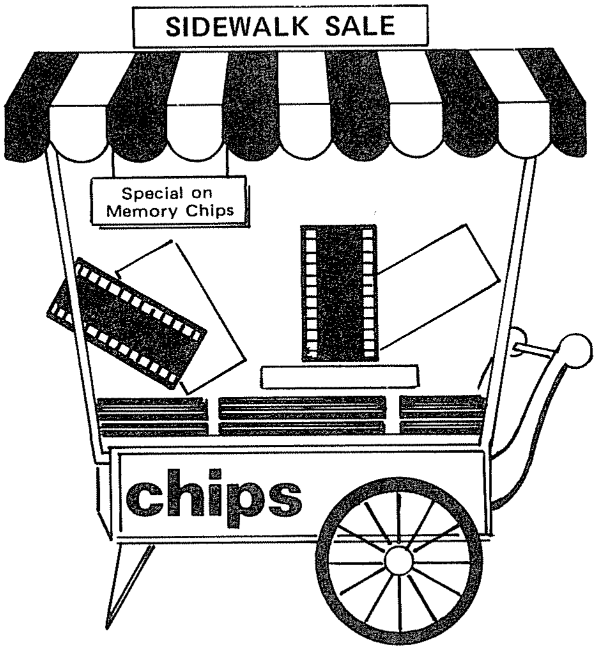
The goal was to test unknown memory devices while not disturbing the TRS-80 data and address buses. Thus the circuit can be divided into functional sections: One is the device under test (DUT), another is its address decoding, another is the DUT/TRS-80 isolation, and the last is an I/O port used for control. The DUT is simply a 4118 of unknown quality. The decoding for the DUT addresses is handled by U5, a 74LS138. The 4118 decodes A0 through A9 itself, and the 74LS138 decodes A12 through A15. U5 actually decodes eight 1K blocks from 8000 hex to 9FFF. Tying A13, A14, and A15 to G2B, G2A, and G1 enables U5 only when addresses 8000 through 9FFF appear on the address bus. Tying A10 through A12 to the A, B, and C inputs causes one of the Y inputs to be low, corresponding to the proper one of eight possible sets of 1K blocks between 8000 and 9FFF. The Y0 output of U5 is the chip select for the 4118,which puts it in the address range 8000 through 83FF.

There is another concern. During certain I/O operations, the top half of the address bus may have data on it, possibly causing U5 to decode an address. This does not cause unwanted memory accesses because the read and write lines are not active during this time. To get a read or write to the 4118, not only must the chip select be active, but also read or write. Read and write lines come from the TRS-80. They, as well as the address and data bus, are buffered to isolate the TRS-80 from the DUT. This isolation is required because if there is any type of short in the DUT it could cause erroneous data to appear on the data bus and thus foul up the operation of the TRS-80. These isolation devices, U1, U2, and U3, have three-state outputs. This is necessary for the lines hooked to the TRS-80, and also for the DUT. Many MOS devices can get confused if signals are applied to their inputs or outputs before their main power is applied. Early versions of this circuit seemed to indicate that the 4118 is susceptible to this confusion. Thus, these isolation devices hold all the 4118 inputs in a high impedance state while the power is off.

This brings us to the control section. This section activates the isolation circuit, it determines what direction data should be going in relation to the DUT, it applies and removes power to the DUT, and it lights certain status indicators. The isolation circuits are activated by a signal called Master Enable Not (labeled $\overline{\text{ME}}$). $\overline{\text{ME}}$ (and all other control signals) is an output of a 74LS374, U6. U6 is configured as an I/O port; we'll go into that later. $\overline{\text{ME}}$ can be set or reset under software control. When it is low, all isolation circuits, U1, U2, and U3, are enabled. Also, when U3 is enabled, the read line from the TRS-80 is allowed to go to the DUT. It also goes to a gate and inverter which is used to determine the direction of data flow to or from the DUT. This is a line called $\overline{\text{DIR}}$. Thus, the data from the DUT will never be placed on the data bus unless chip select, read, and Master Enable are all active.

Besides Master Enable, there are three other lines. Two of them are control status indicators. These indicate whether a DUT passed or failed. After a test is run, a pass or fail decision is made. One of these lights can then be turned on under software control. The last control line activates a reed relay. This relay supplies power to the DUT. When DUT power is on another LED is lit. It is called Test In Progress (or TIP). Thus, the status of DUT power is also under software control. The last piece of hardware to discuss is the I/O port. It consists of U6 and U7. U6 has already been discussed. What makes it an I/O port (actually, output only) is the address decoding provided by U7. In a TRS-80, I/O ports are defined by the conditions of the lower eight address lines (A0 through A7) and $\overline{\text{IN}}$ and $\overline{\text{OUT}}$.

I made certain assumptions about this test circuit which speeded and simplified its design. The biggest assumption was that it would be the only thing tied to the TRS-80 expansion port while it was being used. The final design presented here does allow other things to be on the port, but liberties were taken with the port address decoding. Specifically, U6 will accept data for certain port addresses greater than 240 decimal. This allowed me to use only one IC to decode the port address. By putting A5 through A7 on the A, B, and C inputs, I could decode eight blocks of 32 addresses. But because I put A4 on G1 and A3 on G2B, only those addresses with A3 low and A4 high can be decoded. Also, I tied OUT to G2A. This means that U7 will only decode an address when A3 is low, A4 is high, and an output to a port is being done. I then selected the Y7 output. This causes U6 to accept data whenever an OUT command is issued to a port whose address is between 240 and 247 decimal. Thus, while this circuit occupies eight port addresses, 248 are still available. But don't forget that the DUT is still sitting in the memory address space from 8000 to 83FF hex.



Software

Memory testing is a complex, controversial, and often emotional subject. The purpose of this project has been to provide the most testing for the least time and effort. The software can be divided into three sections: initialization, testing, and processing. Two programs are provided. One is in BASIC and the other is part BASIC and part machine language.

The BASIC program provides the address and data for each failure. The BASIC/machine-language program provides pass/fail indications only. The BASIC program takes one-and-one-half minutes to test a good chip while the mixed program takes about three-quarters of a second. (Most of this time is used up by the BASIC portion.) The initialization portion turns off all indicators, removes DUT power, and resets Master Enable. This is done by an OUT 240,15 instruction, which sets the four low-order bits of the data field to one. It also sets up the display on the screen for pass/fail data to come later.

The test gives you three options: first is TEST. In this mode, a test is run. A one is added to the total units tested, and a one is added to either the pass or fail count depending on the outcome. Second is RETEST. This option is the same test, but no counts are changed. It is primarily used to verify a failure.

The third option returns you to BASIC with the READY prompt. The test portion executes three different memory tests. In each case the memory is loaded with a known pattern. Then, after all 1024 bytes are loaded, they are read back and the read data is checked against what it should be. If the data does not match, an error is flagged. At this point the BASIC program will print a message giving the memory address checked, the data found there, and what was expected. The machine-language version simply returns with a fail indication. This means that the machine-language test should be used to weed out the bad chips because it runs very quickly, and the BASIC program can be used to go back and check bad devices to see what was wrong.

Each of the three tests uses a different pattern. Each pattern has its own purpose. The first is a number which goes from 0 to 255 (0 to FF hex). On read back, this will tell us if the address decoders are working. This is because if we write to memory and immediately read, or if we put the same data in each byte, we really can't tell for sure if we addressed more than one byte. So we fill

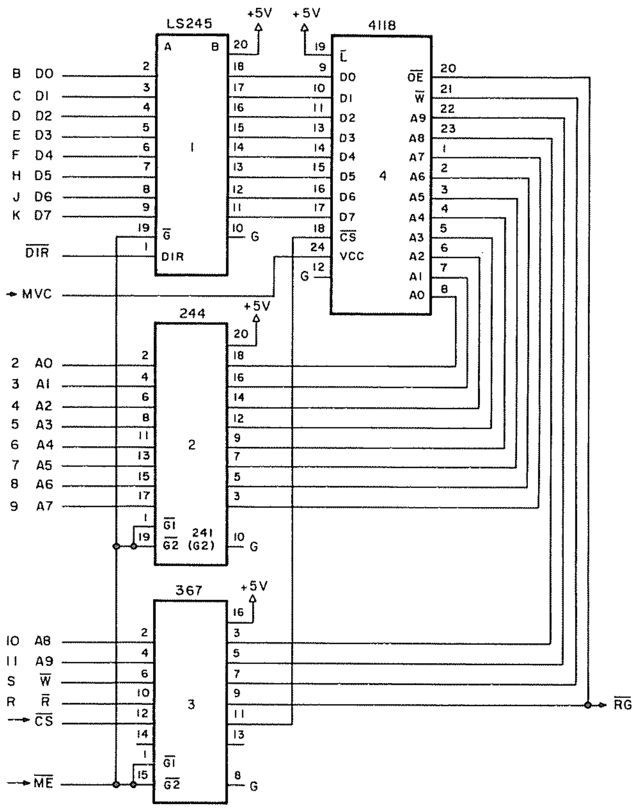


Figure 1. *4118 Test socket electronics*

the memory with different bytes and then read them back. The second pattern is all zeros, and the third is all ones. This checks that each memory cell can hold a 0 or 1.

The final portion of the program is the processing. It turns off DUT power and if an error is found, the red (fail) LED is turned on. If no error, then the green (pass) LED is lit. Also, if T is selected, the units tested and appropriate pass or fail counts are incremented. There you have it: a fairly simple, yet fairly complete memory test. The same concepts used here can be extended to other types of memory devices.

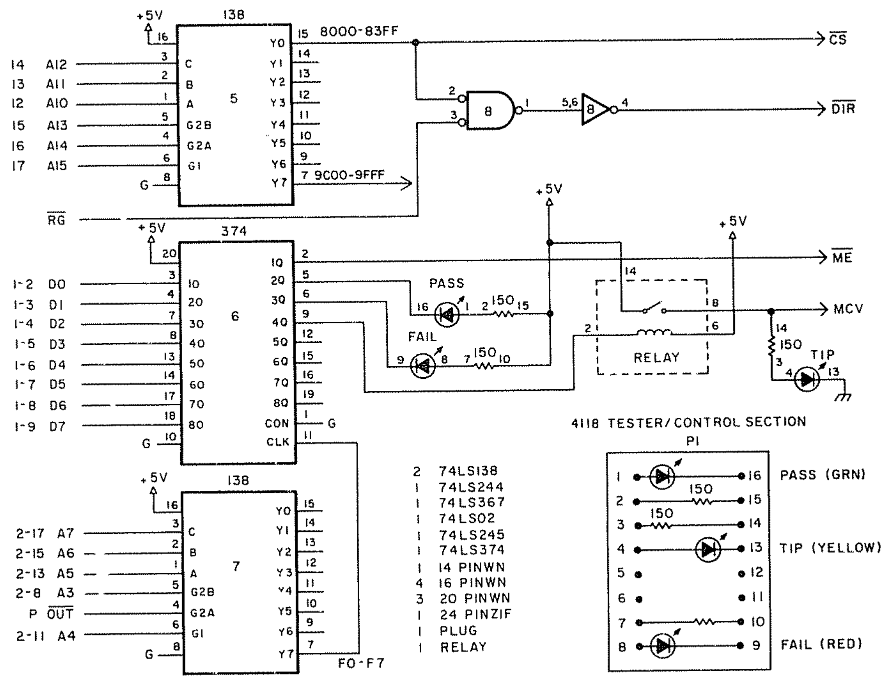


Figure 2. *4118 Tester control section*

Program Listing 1. *BASIC memory tester*

```
 100 REM  *******************************************************
 200 REM  *              4118 MEMORY TEST                       *
 300 REM  * COPYRIGHT 1980, JOE MAGEE                           *
 400 REM  *                 2405 BUNKER HILL                    *
 500 REM  *                 TEMPLE, TEXAS 76501                 *
 600 REM  *******************************************************
 700 OUT 240,15
 750 CLS
 800 PRINT TAB(15),"4118 MEMORY TEST"
 900 PRINT :
     PRINT "PRESS 'T' TO TEST"
1000 PRINT "       'R' TO RETEST"
1010 PRINT "       'E' TO EXIT"
1100 T = 0:
     R = 0:
     P = 0:
     F = 0
1300 PRINT @448,"UNITS TESTED","PASSED","FAILED"
1500 A$ = INKEY$:
     IF A$ = ""
      THEN
       1500
1600 IF A$ = "T"
      THEN
       2000
1700 IF A$ = "R"
      THEN
       2100
1800 IF A$ = "E"
      THEN
       END
1900 GOTO 1500
2000 T = T + 1
2100 DA = 0
2200 OUT 240,7
2300 FOR I = 1 TO 10:
      NEXT
2400 OUT 240,6
3000 FOR AD = 32768 TO 33791
3100  POKE (AD - 65536),DA
3200  DA = DA + 1
3300  IF DA > = 256
       THEN
        DA = 0
3400  NEXT
4000 DA = 0:
     FF = 0
4100 FOR AD = 32768 TO 33791
4200  D = PEEK(AD - 65536)
4300  IF D = DA
       THEN
        4700
4400  FF = 1
4500  GOSUB 15000
4700  DA = DA + 1
4800  IF DA > = 256
       THEN
        DA = 0
4900  NEXT
4950 IF FF > 0
      THEN
       5400
5000 DA = 0
5100 GOSUB 10000
5200 DA = 255
5300 GOSUB 10000
5400 IF A$ = "R"
```

```
      THEN
       7000
5500 IF FF = 1
      THEN
       5900
5600 P = P + 1
5700 OUT 240,13
5800 GOTO 6000
5900 F = F + 1:
     OUT 240,11
6000 PRINT @512,T,P,F;
6100 GOTO 1500
7000 IF FF = 0
      THEN
       5700
7100 OUT 240,11
7200 GOTO 6000
10000 FOR AD = 32768 TO 33791
10100  POKE (AD - 65536),DA
10200  NEXT
10300 FOR AD = 33768 TO 33791
10400  D = PEEK(AD - 65536)
10500  IF D = DA
        THEN
         11000
10600  FF = 1
10700  GOSUB 15000
11000  NEXT
11100 RETURN
15000 PRINT @576,"ERROR--ADR=";AD;"DATA=";D,"SHOULD=";DA;
15100 RETURN
```

Program Listing 2. *BASIC/machine-language version—BASIC portion*

```
 100 REM  ******************************************************
 200 REM  *               4118 MEMORY TEST                     *
 300 REM  * COPYRIGHT 1980, JOE MAGEE                          *
 400 REM  *                  2405 BUNKER HILL                  *
 500 REM  *                  TEMPLE, TEXAS 76501               *
 600 REM  ******************************************************
 700 OUT 240,15
 750 CLS
 800 PRINT TAB(15),"4118 MEMORY TEST"
 900 PRINT :
     PRINT "PRESS 'T' TO TEST"
1000 PRINT "      'R' TO RETEST"
1010 PRINT "      'E' TO EXIT"
1100 T = 0:
     R = 0:
     P = 0:
     F = 0
1300 PRINT @448,"UNITS TESTED","PASSED","FAILED"
1500 A$ = INKEY$:
     IF A$ = ""
      THEN
       1500
1600 IF A$ = "T"
      THEN
       2000
1700 IF A$ = "R"
      THEN
       2200
1800 IF A$ = "E"
      THEN
       CLS :
```

Program continued

```
      END
1900 GOTO 1500
2000 T = T + 1
2200 OUT 240,7
2300 FOR I = 1 TO 10:
      NEXT
2400 OUT 240,6
2500 POKE 16526,0:
     POKE 16527,125
2600 FF = USR(15)
5400 IF A$ = "R"
      THEN
       7000
5500 IF FF = 1
      THEN
       5900
5600 P = P + 1
5700 OUT 240,13
5800 GOTO 6000
5900 F = F + 1:
     OUT 240,11
6000 PRINT @512,T,P,F;
6100 GOTO 1500
7000 IF FF = 0
      THEN
       5700
7100 OUT 240,11
7200 GOTO 6000
```

Program Listing 3. *BASIC/machine-language combined memory test—machine-language portion.*

```
                 00100 ;*****************************************************
                 00200 ;*                4118 MEMORY TEST                   *
                 00300 ;* THIS MACHINE LANGUAGE PORTION STORES, READS       *
                 00400 ;* AND CHECKS DATA ONLY.                             *
                 00410 ;*                                                   *
                 00420 ;* WHEN LOADING THIS PROGRAM ANSWER THE "MEMORY SIZE" *
                 00430 ;* PROMPT WITH 31999.                                *
                 00440 ;*                                                   *
                 00500 ;* COPYRIGHT 1980, JOE MAGEE                         *
                 00600 ;*                  2405 BUNNKER HILL                *
                 00700 ;*                  TEMPLE, TEXAS 76501              *
                 00800 ;*****************************************************
0A9A             00900 MERTPT  EQU     0A9AH           ;RTN TO BASIC WITH HL
8000             01000 MESTRT  EQU     8000H           ;START OF FIXTURE ADR SPC
0400             01100 MEMCNT  EQU     400H            ;SIZE OF 2114/4118 MEMORY
7D00             01200         ORG     7D00H           ;START ADR
7D00             01300 MEMTST  EQU     $
7D00 210080      01400         LD      HL,MESTRT       ;GET STARTING ADR
7D03 010004      01500         LD      BC,MEMCNT       ;GET BYTE COUNT
7D06 3E00        01600         LD      A,0             ;DATA FIELD
7D08             01700 MEMLP1  EQU     $               ;LOAD LOOP
7D08 77          01800         LD      (HL),A          ;LOAD MEMORY CELL
7D09 3C          01900         INC     A               ;SET UP NEXT DATA FIELD
7D0A EDA1        02000         CPI                     ;DUMMY TO BMP HL, DEC BC
7D0C EA087D      02100         JP      PE,MEMLP1       ;DO NEXT CELL
7D0F 3E00        02200         LD      A,0             ;SET UP DATA FIELD
7D11 210080      02300         LD      HL,MESTRT       ;GET START ADR
7D14 010004      02400         LD      BC,MEMCNT       ;GET BYTE COUNT
7D17             02500 MEMLP2  EQU     $               ;CHECK LOOP
7D17 EDA1        02600         CPI                     ;COMPARE DATA
7D19 2031        02700         JR      NZ,MEMERR       ;IF ERROR-EXIT
7D1B E2217D      02800         JP      PO,MEMCN2       ;IF DONE, GO ON
7D1E 3C          02900         INC     A               ;NEXT DATA FIELD
7D1F 18F6        03000         JR      MEMLP2          ;DO NEXT CELL
```

```
7D21            03100 MEMCN2  EQU    $              ;NOW DO 1'S AND 0'S
7D21 3E00       03200         LD     A,0            ;0'S FIRST
7D23 CD317D     03300         CALL   MEMCHK         ;GO
7D26 3EFF       03400         LD     A,0FFH         ;NOW 1'S
7D28 CD317D     03500         CALL   MEMCHK         ;GO
7D2B 210000     03600         LD     HL,0           ;SET UP PASS FLAG
7D2E C39A0A     03700         JP     MERTPT         ;EXIT TO BASIC
7D31            03800 MEMCHK  EQU    $              ;1'S AND 0'S CHECK LOOP
7D31 210080     03900         LD     HL,MESTRT      ;GET START ADR
7D34 110180     04000         LD     DE,MESTRT+1    ;DEST ADR
7D37 010004     04100         LD     BC,MEMCNT      ;BYTE COUNT
7D3A 77         04200         LD     (HL),A         ;DATA TO PROPAGATE
7D3B EDB0       04300         LDIR                  ;LOAD ALL BYTES
7D3D            04400 MEMCK1  EQU    $              ;NOW CHK DATA
7D3D 210080     04500         LD     HL,MESTRT      ;GET START ADR
7D40 010004     04600         LD     BC,MEMCNT      ;GET BYTE COUNT
7D43            04700 MEMLP3  EQU    $              ;CHECK LOOP
7D43 EDA1       04800         CPI                   ;CHECK CELL
7D45 2004       04900         JR     NZ,MEMER1      ;ERROR EXIT
7D47 EA437D     05000         JP     PE,MEMLP3      ;DONE?-NO, DO NEXT CELL
7D4A C9         05100         RET                   ;DONE AND NO ERRORS
7D4B            05200 MEMER1  EQU    $              ;COME HERE FROM SUBRTN
7D4B E1         05300         POP    HL             ;POP RTN ADR FROM STK
7D4C            05400 MEMERR  EQU    $              ;ERROR EXIT
7D4C 210100     05500         LD     HL,1           ;SET UP ERROR RTN CODE
7D4F C39A0A     05600         JP     MERTPT         ;EXIT TO BASIC
7D00            05700         END    MEMTST         ;END
00000 TOTAL ERRORS

MEMCHK 7D31 03800   03300 03500
MEMCK1 7D3D 04400
MEMCN2 7D21 03100   02800
MEMCNT 0400 01100   01500 02400 04100 04600
MEMER1 7D4B 05200   04900
MEMERR 7D4C 05400   02700
MEMLP1 7D08 01700   02100
MEMLP2 7D17 02500   03000
MEMLP3 7D43 04700   05000
MEMTST 7D00 01300   05700
MERTPT 0A9A 00900   03700 05600
MESTRT 8000 01000   01400 02300 03900 04000 04500
```

Build a Light Pen

by Wayne Holder

Imagine the magic effect of moving chess pieces around your screen with the stroke of a light pen.

What is a light pen? It is a wand that guides and instructs your TRS-80 computer. To be more technical, the pen is a light-sensing peripheral that plugs into the cassette jack on the keyboard or expansion chassis, allowing you to select things on the video screen by pointing at them. A light pen lets your computer see where you are pointing.

The pen has thousands of uses. Use it for computer games or to draw pictures on the screen. Let it answer the questions in a quiz, act as a futuristic control panel for an automated house, or for menu selection in business applications.

How It Works

The schematic diagram of the light pen is shown in Figure 1. Its heart is photodetector PQ1. PQ1 is a VTA-1011 photoDarlington which conducts more current when illuminated. Transistors Q1 and Q2, resistors R1, R2, and R3, and capacitors C1 and C2 compose a bias source for the phototransistor. This is a kind of voltage regulator, except that it tracks only slow changes in current. Quick pulses of light cause a change in the voltage across the phototransistor, whereas slow changes in light do not. This allows the circuit to adapt to different ambient light conditions.

The ac signal that crosses the phototransistor in response to quick changes in light is coupled through capacitor C3 and is amplified by transistor Q3. The amplified signal is passed into the TRS-80 circuitry where it simulates a cassette input signal.

It is fortuitous that the cassette recorder's input circuit also serves as a light pen input circuit. I won't go into how the TRS-80 circuitry works, but if you are interested, I suggest you read the TRS-80 technical manual.

Construction Details

Homebrewers can build the light pen into the body of an old felt tip marker or similar housing. You can wire the circuit on perfboard or similar material. If you are skilled with machine tools you may elect to duplicate the housing shown in Photo 1.

Figure 2 shows the parts outlines and dimensions for the body and end caps. DELRIN^{TM} plastic is recommended for machining the end caps, because it is easily worked on a lathe without shattering or chipping. Figure 3 shows the foil pattern and outline for constructing the printed circuit card.

Figure 4 shows the component placement on the circuit card. I recommend using a bending guide to prebend the resistor and capacitor leads before inserting them into the circuit card.

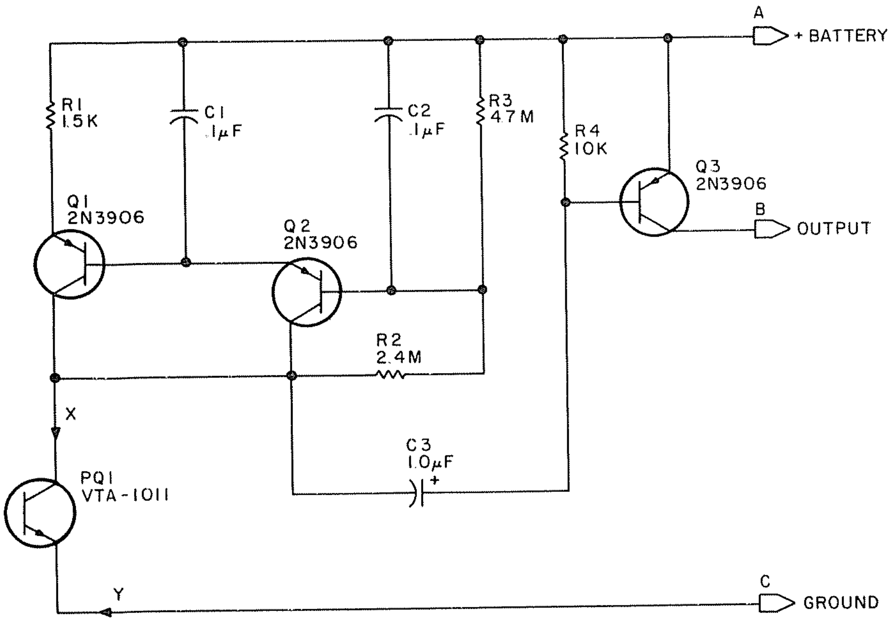
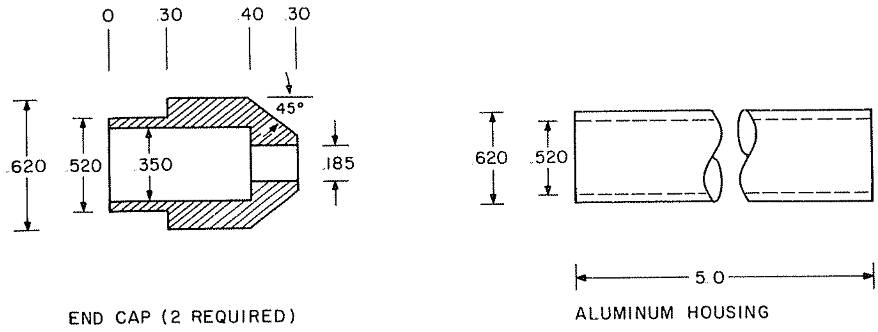


Figure 1. *Light pen schematic*



ALL DIMENSIONS IN INCHES

Figure 2. *Outlines and dimensions for body and end caps*

The three 2N3906 transistors are mounted flat against the circuit card so that the completed assembly will fit into the tubular housing with plenty of clearance. The three transistors are mounted such that the rounded side of the case presses against the circuit card and the flat side faces up.

Bend the leads at a 90-degree angle and insert them into the card. Notice that the pattern of holes is different for Q3. Photo 2 shows the assembled circuit card.

Capacitors C1 and C2 are axial lead, ceramic chip type. They were selected specifically for their small size, but any 0.1 uF nonpolarized capacitor with at least a 25-volt rating may be used. The same thing applies to capacitor C3, which is a miniature tantalum type. All resistors are 1/4-Watt composition for film types with a value tolerance of five percent.

Mounting the Photodetector

Mounting photodetector PQ1 is very important. Cut off the base lead (the center one), and bend the emitter and collector leads to the length specified in the placement diagram. The leads should bend in the direction of the now removed base lead. The emitter lead (the one nearest the tab) connects to the Y pad, and the collector lead connects to the X pad of the circuit card.

Insert all of the components into the circuit card. Be careful to check that they are all flat against the surface, so that the card will fit into the housing. Then solder them into place. Use a good grade rosin core solder, and be careful not to excessively overheat any of the components. Trim the leads

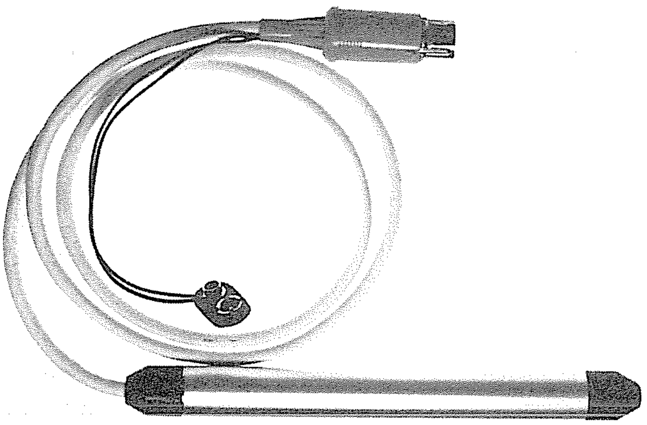


Photo 1. *Assembled light pen*

as close to the circuit card as possible to prevent them from shorting to the aluminum housing.

Check the position of the phototransistor by inserting the PC assembly into one of the end caps. If you have measured correctly, the narrow end of the card will fit into the end cap all the way. The end of the phototransistor should be recessed from the other end about 1/4 of an inch. Next, wire the connector to the cable and battery clip, using Figure 5 as a guide. Twist together the cable shield, the brown wire in the cable, and the black wire from the battery clip. Solder them to pin 2 of the connector.

Twist together the white wire from the cable and the red wire from the battery clip. Solder them, and insulate the soldered joint with tape or heat shrink tubing. Solder the green wire from the cable to pin 4 of the connector. Assemble the connector, being careful to crimp the cable and battery clip wires into the strain relief tabs.

Slip the other end of the cable through one of the end caps. Pull about four inches through and fasten a nylon ty-wrap around the cable near the end cap. The ty-wrap will prevent the cable from pulling out of the end cap once the pen is assembled.

Strip the insulation from the cable, and trim the wires so that they measure about two inches from the end of the end cap. Solder the white wire to point A on the circuit card (Figure 4). Solder the green wire to point B, and the brown wire to point C. Trim off the excess wire after soldering.

At this point your light pen is almost ready to operate. Before final assembly, test it by plugging it into the keyboard, connecting a battery, and running the program shown in Program Listing 1. I advise, however, that you first read the section on programming.

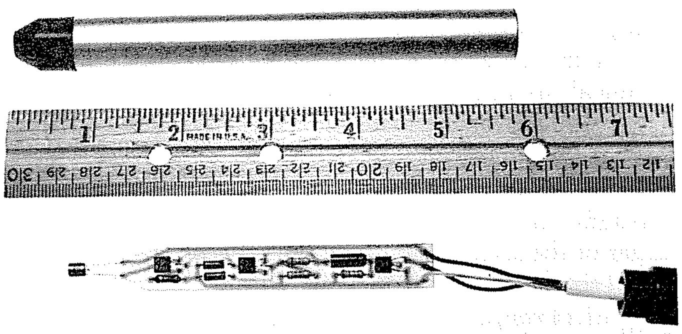


Photo 2. *Assembled circuit card and housing*

When you are satisfied that your light pen is working correctly, assemble it into the housing, using epoxy cement to fasten the end caps into the aluminum tube.

Programming

The pen is a device that can sense light coming from the video monitor. Software is used to distinguish exactly where the light pen is pointing, but the software must be accurate. Let's take a look at exactly what is involved in programming a light pen.

When we look at the screen of the video monitor, we see continuous light. Those of us who are savvy about how television works know that the light is not really continuous. Television creates images by scanning a point of light across the television screen. This point of light scans so rapidly that to our human eyes it appears to be everywhere at once. The photosensor of the light pen, however, is not so easily fooled. It sees a pulse of light every time the point scans past wherever the light pen is pointing.

The video monitor is designed to scan from the top to the bottom of the screen 60 times a second, or once every 1/60 of a second. This means the light pen sees 60 pulses of light every second whenever the light pen is pointed at an illuminated area on the screen. The light pen converts these light pulses into electrical pulses. The electrical pulses are fed into the TRS-80's cassette input latch. A latch is a device that stores information. It has two states: SET (on) or RESET (off). The TRS-80 is able to RESET the latch, while the light pen can SET it.

In addition, the TRS-80 can determine the state of the latch at any time. The light pen SETs the latch every time it senses a pulse of light from the video monitor. Once the latch is SET, it stays in that state until it is RESET by the TRS-80. Therefore, to detect whether or not the light pen is "looking" at light, we need only to RESET the latch, wait at least 1/60 of a second, and then check the latch to see if it is SET.

BASIC Program Listing 1 clears the screen (line 10) and puts a spot of light in the center of the screen (line 20). Line 30 RESETs the latch. (Read the TRS-80 technical manual for more details.) Line 40 delays execution for about 1/60 of a second. Line 50 reads the state of the latch and decides which message (ON or OFF) to print.

Type in the program, connect the light pen, and run it. You will find that pointing the light pen at the spot of light displays the ON message in the upper left corner of the screen; otherwise the OFF message will show.

Now look at the subroutine shown in Program Listing 2. This subroutine does all the work of testing to see if the light pen is looking at light. If a program performs a GOSUB 9000, upon return the variable LP will be set to 0 if the light pen is looking at darkness or 128 if the pen sees light. We will make use of this BASIC subroutine in developing more sophisticated programs.

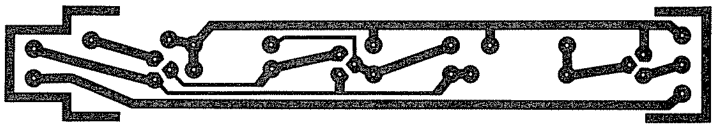


Figure 3. *PC foil pattern*

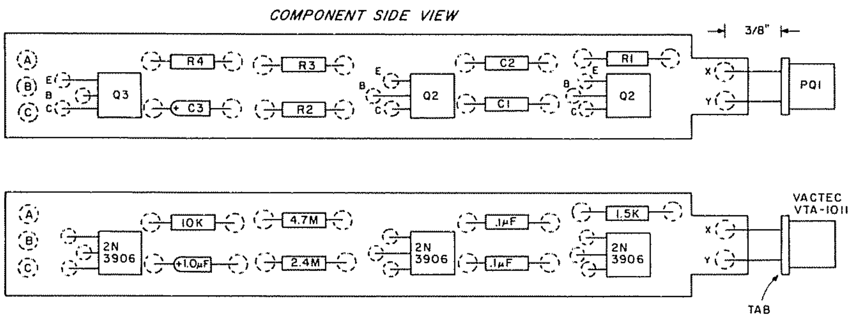


Figure 4. *Components placement guide*

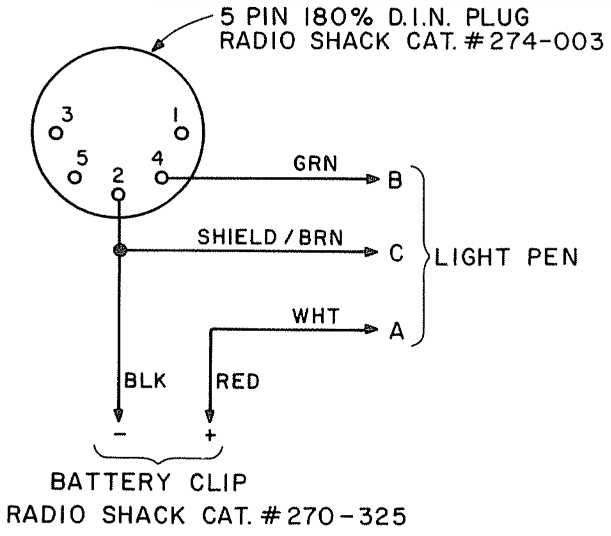


Figure 5. *Plug/cable wiring*

Some Programming Examples

Naturally, the software must do more than simply determine if the light pen sees light. The pen could easily pick up a stray pulse of light from a light fixture, or by being pointed at a line of text. Good software must, therefore, include checks to verify where the light pen is actually being pointed. Enter the BASIC program shown in Program Listing 3, and try it.

This program expects you to point the light pen at the * displayed on the screen. The program calls subroutine 9000 and tests the returned value in variable LP, in order to wait until the light pen senses light (lines 30 and 40). The program then tries turning the * on and off several times and calls 9000 each time. It is checking to see if the light pen sees light with the * on, and, also, if it sees no light with the * off. The program does not decide that the light pen is looking at the * until it can successfully repeat this test several times. This double checking is very important for proper operation of any light pen program.

Now let's look at another light pen application called menu selection. Most of us have used programs that required us to choose among various options. The game Star Trek, popular in most computer circles, is an example. To play the game, the player is usually requested to "TYPE 1 TO FIRE PHASORS, TYPE 2 TO FIRE PHOTON TORPEDOES, TYPE 3 TO SCAN QUADRANT FOR LIFE," etc. Wouldn't it be great if your computer had separate keys labeled with each option? You can use the light pen in this application.

Consider if the screen showed something like this:

```
*  FIRE PHOTON TORPEDOES
*  FIRE PHASORS
*  SCAN FOR LIFE IN QUADRANT
*  SURRENDER
```

You could play the game by touching the light pen to the * next to the appropriate command. Perhaps certain commands would call up other menus of commands for us to select from.

Imaginary Restaurant

We don't have space here to present a complete Star Trek program, but try the program shown in Program Listing 4. The program is called Imaginary Restaurant, and you use the light pen to select what you would like to eat from the bill of fare.

Lines 5 through 40 reserve string space and dimension the arrays A$ and B. Array B is then initialized to the values 192, 320, and 448, which correspond to areas on the screen that will be changed with the PRINT@ command.

Lines 100 through 110 read the contents of the DATA statements into the array A$. The array is initialized to contain messages that the program will display. A$ is a two-dimensional array organized so that the messages are put into groups of four. Each group corresponds to the X dimension of the array, and the particular message in a group corresponds to the Y. Line 120 selects the first group by setting X = 0.

Lines 200 through 240 display a group of messages on the screen. Message 0, Y = 0 of each group, is displayed as a prompting message at the top of the screen. Messages 1–3 are displayed as choices with a target spot preceding each one. The first group of messages appears as follows:

```
WELCOME TO THE IMAGINARY
RESTAURANT—PLEASE SELECT
* BREAKFAST
* LUNCH
* DINNER
```

The program expects the user to select one of the three choices it offers by pointing to the associated spot with the light pen. The program waits at statement 300 until the pen detects light. When it's detected, the program assumes that the user is pointing to one of the three spots.

Lines 310 through 350 determine which spot, by turning off each selection. The program assumes the light pen will see darkness when it finds the correct one. As a precaution should the test fail, the program will get to line 360. The program branches to line 400 when it thinks it has located the selection. Lines 400 through 450 perform a double check on this assumption by turning the suspect spot on and off several times, checking for the correct response. If this double check fails, the program branches back to line 300.

If the check passes, the program displays an arrow, <= , next to the spot and waits for the light pen to see darkness. This means the user has lifted the pen from the screen. Line 500 checks for further displayed choices. If they are there, the program sets X to the group requested and branches back to 200. If they are not, the program terminates by printing the final selection.

The Magic Subroutine

Imaginary Restaurant is a rather specialized program, as you probably noticed. If light pen input is ever going to be painless, we will need to develop a subroutine to do the hard work for us. Before we get into exactly how we are going to write such a subroutine, let's define how we would like it to work.

The PRINT@ command in Level II BASIC is a very convenient way to display messages anywhere on the screen. Using this command, you can refer to any location on the screen by using a value between 0 and 1023. Imaginary Restaurant made extensive use of the PRINT@ in setting up the display and in turning the light pen targets on and off. Suppose we had a subroutine that would accept a list of locations on the screen, where each location on the screen was described by a value from 0-1023.

Let us further suppose that this subroutine would display a light pen target at each of these locations, wait for the light pen to touch one of them, and determine which one was touched by returning a value. This will be our magic light pen subroutine. Let us design this subroutine such that the list of locations is contained in an array called LST, and that the first element of this array, LST(0), will contain a value indicating how many locations are in the list. Therefore, if we wish to display three targets on the screen at locations 120, 430, and 522, we would set up the array as follows:

```
100 LST(0) = 3
110 LST(1) = 120
120 LST(2) = 430
130 LST(3) = 522
```

We would then call our subroutine. (Let's assume it starts at line 9100.) We will design the subroutine to set a variable called SCAN equal to the list element index (in this case 1-3) which corresponds to the target location touched by the light pen. If, for example, the light pen had touched the target displayed at location 430, then SCAN would equal two upon returning from the subroutine. The listing for the Magic Light Pen subroutine is shown in Program Listing 5. Program Listing 6 is a short program that uses the magic subroutine to build a simple quiz program.

The subroutine makes use of variables names SCAN, I, LP, CNT, C$, and the array LST, so don't use them elsewhere. Also remember to DIMension the array LST and to put the proper values into it before doing a GOSUB 9100. The string variable C$, which is assigned a value at statement 9120, determines what character will be used as the light pen target. Currently it is set to a graphic character, but you can change it to something else if you like.

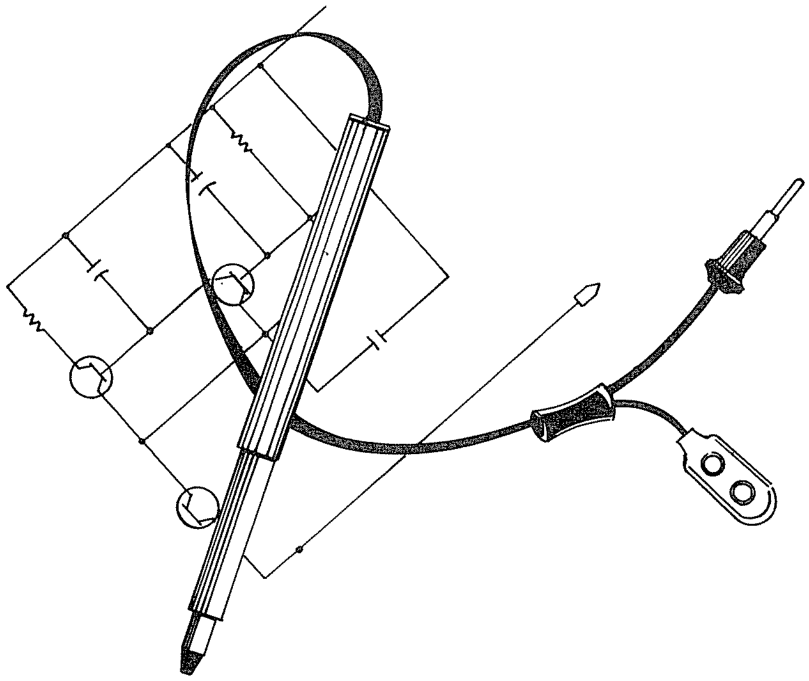
Assembly-Language Programming

The light pen can be used by assembly-language programs just as easily as it can be by BASIC programs. Assembly language has the advantage of being much faster than BASIC. This is a real help when writing light pen programs, because you can test more targets on the screen in less time. An assembly-language equivalent to the BASIC subroutine at 9000 is shown in Program Listing 7. This program is presented using Zilog standard Z-80 mnemonics. This simple subroutine is sufficient to perform all light pen input. I recommend, however, that anyone planning to write assembly-language programs for the light pen should be thoroughly familiar with assembly-language programming.

What to Do About Problems

Should you experience problems with your light pen, the following suggestions could be of help. If the light pen refuses to work at all, the problem could be a weak or drained battery. Try replacing the battery, and then try one of the simple programs such as the ON/OFF test in Program Listing 1. If the light pen still refuses to operate, try turning down the lights in the area of the video screen. Adjusting the brightness and contrast on the video monitor may also be of help. The light pen is most sensitive to the video screen when the contrast and brightness are adjusted for a bright, clear image with a dark background.

Note: Recent models of the TRS-80 computer and those which have had the cassette fix installed may not work correctly with the light pen, unless a 1k Ohm resistor is installed between points B and C on the circuit card. This resistor may also be installed between pins 2 and 4 inside the DIN connector.



Program Listing 1. *Test of light pen operation*

```
10 CLS
20 SET(64,24):
   SET(65,24)
30 OUT 255,0
40 FOR X = 1 TO 6:
    NEXT X
50 IF ( INP(255) AND 128) = 0 GOTO 70
60 PRINT @0,"ON ";:
   GOTO 30
70 PRINT @0,"OFF";:
   GOTO 30
```

Program Listing 2. *Light-detecting subroutine*

```
9000 OUT 255,0
9010 FOR Z = 0 TO 6:
      NEXT Z
9020 LP = ( INP(255) AND 128)
9030 RETURN
```

Program Listing 3. *Verification of light pen position*

```
 10 CLS
 20 PRINT @384,"*  <= TOUCH HERE";
 30 GOSUB 9000
 40 IF LP = 0 GOTO 30
 50 CNT = 2
 60 PRINT @384," ";
 70 GOSUB 9000
 80 PRINT @384,"*";
 90 IF LP < > 0 GOTO 30
100 GOSUB 9000
110 IF LP = 0 GOTO 30
120 CNT = CNT - 1
130 IF CNT < > 0 GOTO 60
140 CLS
150 PRINT "DON'T TOUCH ME!"
160 STOP
```

Program Listing 4. *Imaginary Restaurant*

```
  5 CLEAR 1200
 10 DIM A$(4,4),B(3)
 20 B(0) = 192
 30 B(1) = 320
 40 B(2) = 448
 99 REM  READ DATA INTO ARRAYS
100 FOR X = 0 TO 3:
     FOR Y = 0 TO 3
110   READ A$(X,Y):
      NEXT Y,X
120  X = 0
199  REM  DISPLAY A SCREEN GROUP
```

```
200  CLS
210  PRINT A$(X,0):
     PRINT :
     PRINT
220  FOR Y = 1 TO 3
230   PRINT CHR$(140);"   ";A$(X,Y):
      PRINT
240   NEXT Y
299  REM  WAIT FOR LIGHT PEN TO SENSE LIGHT
300  GOSUB 9000:
     IF LP = 0 GOTO 300
310  FOR W = 0 TO 2
320   PRINT @B(W)," ";
330   GOSUB 9000:
      IF LP = 0 GOTO 400
340   PRINT @B(W), CHR$(140);
350   NEXT W
360  GOTO 300
399  REM  GET HERE IF PROPER ONE LOCATED. DOUBLE CHECK
400  FOR S = 0 TO 1
410   PRINT @B(W), CHR$(140);
420   GOSUB 9000:
      IF LP = 0 GOTO 300
430   PRINT @B(W),"   ";
440   GOSUB 9000:
      IF LP = 128 PRINT @B(W), CHR$(140);:
      GOTO 300
450   NEXT S
460  PRINT @B(W), CHR$(140);"<=";
470  S = 0
480  S = S + 1:
     GOSUB 9000:
     IF LP = 128 GOTO 470
490  IF S < 5 GOTO 480
499  REM  GET HERE IF DOUBLE CHECK PASSES
500  IF X = 0
      THEN
       X = W + 1:
       GOTO 200
510  CLS
520  PRINT @192,"VERY GOOD! WE AT THE IMAGINARY"
530  PRINT " RESTURANT HOPE YOU ENJOY YOUR"
540  PRINT A$(X,W + 1);" FOR ";A$(0,X)
550  PRINT :
     PRINT :
     PRINT
560  PRINT "BY THE WAY. YOUR CHECK TOTALS---$37.50"
570  FOR X = 0 TO 5000:
      NEXT :
     GOTO 120
599  REM  DATA AREA
600  DATA WELCOME TO THE IMAGINARY RESTURANT- PLEASE SELECT
610  DATA BREAKFAST
620  DATA LUNCH
630  DATA DINNER
640  DATA FOR BREAKFAST WE ARE SERVING
650  DATA "HAM, EGGS, AND SAUSAGE SPECIAL"
660  DATA PANCAKES AND FRIED POTATOES
670  DATA COFFEE AND DANISH
680  DATA OUR LUNCH SPECIALS ARE
690  DATA "HAMBURGER, COKE, AND FRIES"
700  DATA ROAST BEEF SANDWICH
710  DATA VEGETARIAN SPECIAL (RICE AND BEAN CURD)
720  DATA FOR DINNER WE ARE PLEASED TO OFFER
730  DATA EXPENSIVE STEAK
740  DATA OVERPRICED SEAFOOD
750  DATA ECONOMY SOUP AND SALAD
760  :
     :
770  : REMEMBER TO INCLUDE SUBROUTINE 9000
```

Program Listing 5. *Magic light pen subroutine*

```
9100 :
     ' MAGIC LIGHT PEN ROUTINE
9110 L = LST(0)
9120 C$ = CHR$(140)
9130 FOR I = 1 TO L:
      :
      '                      DISPLAY TARGETS ON SCREEN
9140  PRINT @LST(I),C$
9150  NEXT I
9160 GOSUB 9000:
     :
     '                     WAIT FOR LIGHT INPUT FROM PEN
9170 IF LP = 0 GOTO 9160
9180 SCAN = 1
9190 PRINT @LST(SCAN)," ";:
     :
     '                             DETERMINE TARGET TOUCHED
9200 GOSUB 9000
9210 IF LP = 0 GOTO 9260
9220 PRINT @LST(SCAN),C$;
9230 SCAN = SCAN + 1
9240 IF SCAN < = L GOT 9190
9250 GOTO 9160
9260 PRINT @LST(SCAN),C$;:
     :
     '                               TARGET FOUND. DOUBLE CHECK
9270 GOSUB 9000
9280 IF LP = 0 GOTO 9160
9290 CNT = 2
9300 PRINT @LST(SCAN)," ";
9310 GOSUB 9000
9320 PRINT @LST(SCAN),C$;
9330 IF LP < > 0 GOTO 9180
9340 GOSUB 9000
9350 IF LP = 0 GOTO 9160
9360 CNT = CNT - 1
9370 IF CNT < > 0 GOTO 9300
9380 PRINT @LST(SCAN) - 2,"=>";:
     :
     '                              DISPLAY PROMPT ARROWS
9390 PRINT @LST(SCAN) + 1,"<=";
9400 GOSUB 9000:
     :
     '                       WAIT FOR PEN TO BE LIFTED
9410 IF LP < > 0 GOTO 9400
9420 RETURN
```

Program Listing 6. *Quiz routine*

```
100 DIM LST(4)
110 LST(0) = 3
120 LST(1) = 266
130 LST(2) = 394
140 LST(3) = 522
150 CLS
160 PRINT "WHO WAS THE SECOND PRESIDENT OF THE U. S. ?"
170 PRINT @LST(1) + 4,"THOMAS JEFFERSON"
180 PRINT @LST(2) + 4,"JOHN ADAMS"
190 PRINT @LST(3) + 4,"PAUL HARVEY"
200 GOSUB 9100
```

```
21Ø CLS
22Ø IF SCAN = 2 PRINT "THAT IS CORRECT!"
23Ø IF SCAN < > 2 PRINT "I'M SORRY. YOU ARE WRONG."
24Ø STOP
```

Program Listing 7. *Assembly-language version*

```
;SUBROUTINE TO READ LIGHT PEN STATUS AND RETURN
;A =Ø IF 'NO LIGHT' OR 128 IF 'LIGHT'
;
        PUSH    HL
        LD      HL,15ØØ
        LD      A,Ø
        OUT     (255),A
;DELAY FOR 1/6Ø OF A SECOND
LOOP    DEC     HL
        LD      A,H
        OR      L
        JP      NZ,LOOP
        IN      A,(255)
        AND     128
        POP     HL
        RET
```

HOME APPLICATIONS

BASIC Word Processor

by Delmer D. Hinrichs

You *do* always think of better phrasing after you see your thoughts in print, don't you? A word processor for your computer lets you write letters and articles without typing a series of rough drafts. All of your typing revisions and editing are done on the video display, and you produce hard copy only after you are satisfied with the text. You can also save the text on cassette so that you can make additional corrections and/or hard copies at a later time.

This BASIC word processor has the following features:

1. It accepts normal upper/lowercase typing on an unmodified 16K Level II TRS-80.
2. It is line-oriented. Every line of text has a line number for reference. This number need not be printed.
3. Editing is similar to Level II BASIC editing, except that the entire line is always visible under a transparent cursor.
4. You can change any of the ten variables to format for print or display.
5. Words can be automatically moved between lines to make all lines the correct format length.
6. Lines of text can be deleted, inserted, replaced, or moved.
7. Lines of text can be automatically right-justified.
8. Up to 120 lines of text, nearly 5 pages, double-spaced, can be in memory at one time.

Running the Program

When the program is loaded and RUN is entered, a title and prompt, COMMAND?, are displayed. The program will accept 17 single-letter legal commands, as shown in Table 1.

A	ADD	Add text at the end of the text file.
B	BLANK	Eliminate blank lines, and renumber text file.
C	COMPILE	Move words between lines in a specified block of lines to adjust lines to the correct length.
D	DELETE	Delete a specified block of lines.
E	EDIT	Edit a specified line.
F	FORMAT	Change the format for text display or printing.
H	HELP	List the 17 valid commands.
I	INSERT	Insert new lines into the middle of the text.
J	JUSTIFY	Right-justify the text file.
K	KILL	Eliminate the current text file, and start over.
L	LOAD	Load a text file from cassette tape.

M	MOVE	Move specified lines to a new location in text.
P	PRINT	Print the text file on the printer.
R	REPLACE	Replace a specified line with a new line.
S	SAVE	Record the text file on cassettė tape.
V	VIDEO	Display the text file on the video display unit.
X	EXIT	Exit from the program.

Table 1. *Single-letter legal commands*

Now, let's examine these commands in detail.

ADD. The final lines of the current text (if any) are displayed, and a flashing block cursor shows where text will be added. This command automatically turns on the line number display option. Text can now be typed continuously without having to ENTER each line. When the text file is filled, a FILE FULL message is shown. Subcommands in the ADD mode are shown in Table 2.

←	Moves the cursor left one position, and erases the last character.
Shift ←	Erases the entire current line.
→	Moves the cursor five positions to the right, adding five spaces.
Space bar	Moves the cursor one position to the right.
ENTER	Ends the line, and goes to the next line before the automatic end-of-line action.
Shift →	Ends the line, and moves the line text to the extreme right of the line.
↓	Ends the line, and leaves an end-of-page marker, the down arrow, at the end of the line.
CLEAR	Ends the line, and leaves a do-not-justify marker, left arrow, at the end of the line.
Shift ↓	Ends the line, centers the line text, and leaves a do-not-justify marker at the end of the line. (Some TRS-80s require Shift ↓ 2.)
Shift@ twice	Ends ADDing text, and returns to command mode. The first Shift@ stops the program, and the second returns. Occasionally, only one Shift@ is needed.

Table 2. *Subcommands in the ADD mode*

BLANK. All blank lines are eliminated from the text, and the line numbers are closed up. Note that blank lines are empty, and this command does not affect lines that contain only spaces. During operation, this command displays DELETING BLANK LINES; when done, it automatically displays the new text file using the VIDEO command.

COMPILE. After editing or reformatting the text, some lines may be too long or too short to properly fit into the specified line length. This command shifts words between the lines of a specified block to get the best possible fit. COMPILE works in two stages: It first checks for lines that are too long and pushes any extra words onto the following line. It then checks if any line can accept words from the following line and pulls any possible words back onto the preceding line. COMPILE should be used on only one paragraph at a time, as it left-justifies all lines except the first within its range. It can also bury any end-of-page or do-not-justify markers that are not on the last line within its range. This should be avoided, as the markers are then ineffective. Extra spaces between moved words are eliminated, except at the ends of sentences.

Following a period, question mark, exclamation point, or colon, three spaces are inserted. Trailing spaces are deleted, leaving only one space in a previously all-space line (to keep the BLANK command from eliminating the line). Note that COMPILE can push extra words forward several lines, but can pull words back only one line.

During operation, this command displays COMPILING. If, after using COMPILE, the last line of the specified block is still too long, a LINE n HAS x CHARACTERS message is displayed. To correct this, INSERT an empty line and COMPILE just those two lines. After a satisfactory COMPILE, the text file will be automatically displayed by the VIDEO command.

DELETE. This command eliminates a specified block of lines. If only one line is to be eliminated, enter that line number as both the first and last line number to DELETE. The text is displayed when done.

EDIT. The entire line—255 characters—is visible when in EDIT, including the character under the cursor. If a non-existent or empty line is specified, an ENTRY ERROR message is given. This command automatically turns on the line number display option. See Table 3 for subcommands in the EDIT mode.

n ←	Moves the cursor to the left n positions. The default value of n is always one.
n → or n (space)	Moves the cursor to the right n positions without adding spaces.
A	Again. Cancels previous editing changes, and reenters EDIT mode. (List makes editing changes permanent.)
n C	Change next n characters to next n entered characters. Cursor returns to start of changed block when done, as a signal that you have finished.
n D	Delete the next n characters, and close up the line.
H	Hack the rest of the line, and enter the INSERT mode.
I	Insert characters into the line, and move following characters to the right. While in the INSERT mode, you can move the cursor left or right without changes to the text by using ← or →.
L	List the line, and return cursor to the beginning. Also makes past editing changes immune to the Again subcommand.
n S c	Search for the nth occurrence of character c. Keeps upper and lower case separate, so can be used to find out if a letter on the video display is correct case.
X	Go to the end of the line, and enter the INSERT mode.

Table continued

Shift ↑	Exits the H, I, or X modes, and returns to EDIT.
Shift →	Moves current text to extreme right of the line.
Shift ↓	Centers current text, and adds a do-not-justify marker to the end of the line. (Some TRS-80s require shift ↓ Z.)
ENTER or Shift@	Exits from the EDIT command.

Table 3. *Subcommands in the EDIT mode*

After exiting from EDIT, if the line is too long, the LINE n HAS x CHARACTERS message is given. If the line is not too long, the text is displayed by the VIDEO command.

FORMAT. This command resets the ten text formatting parameters from their default values, either for the video display or for printing of the text.

1. Line length: Default value is 60 characters to fit on one video display line.

2. Line spaces: Default value is 0. Enter the number of blank lines that you want to appear between lines of text.

3. Line numbers: Default value is Y. To delete line numbers, enter N.

4. First line: Default value is 0 to show all lines of the text. To start the display or printing at a later line number, enter the desired beginning line number.

5. Left margin: Default value is 10 to print the default 60-character line centered on an 80-character printer. This setting affects only the printer.

6. Page length: Default value is 15. This is the number of lines to be printed on each page, so you might set it to 50 for printing unspaced text, or to 25 for spaced lines.

7. Page numbers: Default value is N, no page number. To print page numbers, enter "Y". Note: You must print page numbers if you want to print a page heading (see item 10 below).

8. First page: Default value is one to start page numbering with page number one. May be reset as required.

9. Page 1 number: Default value is Y to print page numbers for all pages. To print page numbers for all pages *except* page one, enter N.

10. Heading: Default value is one space. If a page heading is to be printed at the top of each page, enter it. For this entry only, press shift and the letter for lowercase and the letter for uppercase. If the heading is to have leading spaces (for centering) or punctuation, enclose it in quotes(""). The heading is printed or displayed only if the page numbering is on (items 7 and 9 above).

After going through these ten FORMAT parameters, you are returned to the command mode.

INSERT. This command inserts a line (or lines) of text into the middle of the current text. The following lines of text are moved down and renumbered. Specifying a non-existent line will give an ENTRY ERROR. To insert empty lines (for COMPILE or MOVE), just press ENTER. If the text file is filled, a FILE FULL message is given.

JUSTIFY. This command right-justifies all of the text, i.e., it makes the right ends of the lines even. The only exceptions are lines with a do-not-justify or an end-of-page marker at the end, or a line that has no spaces between words. Extra spaces are inserted between words, starting randomly, but evenly distributed. Spaces may be inserted between adjacent words, or every other word, depending upon whether there is an even or odd number of words in the line. Trailing spaces are eliminated, leaving only one space for an all-space line. Leading spaces are not affected, so that indentation may be maintained.

It is suggested that JUSTIFY be used only after the text is in its final form, as the extra spaces may be incorrectly left in by COMPILE (after EDITing). During its operation, this command displays JUSTIFYING; when done, the text is displayed by VIDEO.

KILL. This command eliminates all text from the text file, leaving the program ready to accept new text. The command asks twice if you really want to KILL the text to avoid accidental loss of a text file.

LOAD. This command loads a previously SAVEd text file from cassette. It can then be treated just like a keyed-in text file. After you enter this command, the program pauses to let you get the cassette in position and set the recorder to play. Since it may take several minutes to load the text file, the program displays LOADING to reassure you that the program has not hung up.

MOVE. This command transfers a specified block of lines either forward or backward in the text file. Lines MOVEd *to* must be empty (blank); lines MOVEd *from* are left empty. INSERT may be used to place empty lines where needed. If a non-empty *to* line is found, the transfer of lines stops, and a LINE n NOT EMPTY message is displayed. However, no text is lost. After the MOVE is complete, the text is displayed by VIDEO.

PRINT. This command prints the text file. Remember to reset FORMAT before trying to PRINT your text. If the printer is not ready to go, PRINTER NOT READY. ABORT (Y/N) is displayed, and you have a chance to get the printer ready or to go back to the command mode. The BLANK command is automatically executed before PRINTing, as a blank line would cause a FUNCTION CALL error. During its operation, first DELETING BLANK LINES, then PRINTING is displayed. If the printer encounters an end-of-page character (ASCII 17), that page will be terminated early, the normal between-pages spacing inserted, and the next full page started. Note: It is assumed that roll paper is being used; no form feeds are used.

Different printers operate somewhat differently, so you may have to change some things in lines 1680-1740. Specifically, my printer, a COMPRINT 912, interprets the ASCII control character 30 in line 1680 as continuous print (no pagination). The corresponding control character 28 in line

1740 means paginate (insert seven blank lines). Some printers may require line feed characters (usually ASCII 10) after each printed line.

Some printers will not accept the LPRINT at the end of line 1690, and require LPRINTCHR$(138) (or LPRINT" ") instead. If the 138 pseudo-control character in line 1710 does not work for you, use:

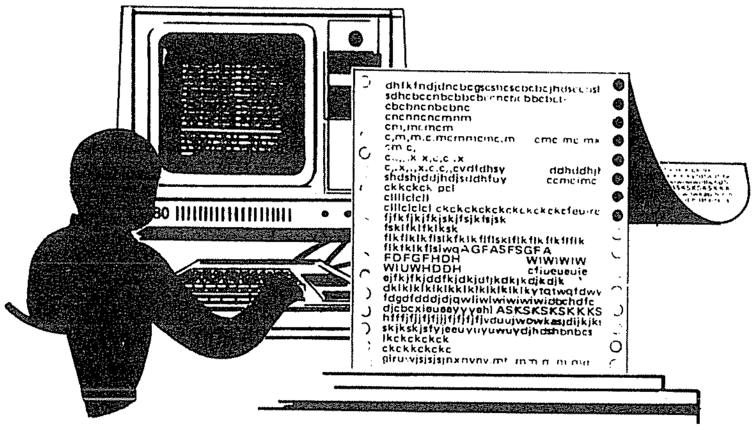
```
1710 M = M + 1:IFSFORK = 1TOS:LPRINT" ":NEXTK
```

A little revision should allow any printer to be used.

REPLACE. This command allows any specified line to be replaced with a newly entered line. REPLACE operates just like ADD, except that it applies to only one line. After a FILE FULL message, you cannot REPLACE a line.

SAVE. This command records the text file on cassette. The program pauses to let you get the cassette into position. It displays SAVING during operation. You can record four lines of text per data block to save time and tape. If the line length is greater than 61 characters, only three lines of text can be recorded per block. If it is greater than 82 characters, only two lines of text can be recorded per block. If it is greater than 123 characters, only one line of text may be recorded per block.

To make the changes (if necessary), determine the number of lines you can fit in a block. Change the four at the end of line 1780 to that number. Change the three in line 1790 to one less than that number. Finally, line 1810 must be changed so that the X$(0), X$(1), X$(2), X$(3) series has the same number of elements as lines in the block. For example, two lines to a block means the series is X$(0), X$(1).



To LOAD the text recorded with the revised SAVE routine, corresponding changes must be made to lines 1540 and 1550 of the program. If text lines are more than 60 characters long, NL should be redefined in program line 30 to approximately 7200/(characters per line). The exact value should be a number divisible by lines per block. For example, NL = 120, and 120 is evenly divisible by four.

VIDEO. This command displays the text file on the video display. If the line length is greater than 60 characters (64 if LINE NOS. was set N in FORMAT), the lines will "wrap around" to the next display line. After each displayed page of text, the program halts, showing PRESS ENTER?. To display the next page just press ENTER; to return to the command mode (to EDIT a line, etc.) key in any letter, then press ENTER. VIDEO may show one of three non-text markers at the end of a line: a left arrow for do-not-justify, a down arrow for end-of-page, or an underline after any trailing space. These markers allow you to keep better track of the text.

EXIT. This command allows for a graceful end to the program. More importantly, it CLEARs the string space to its normal value, so that the next program you run does not crash. It is easy to forget to CLEAR 50. It also returns the TRS-80 to its normal speed (OUT254,0—see below). This command asks again if you really want to EXIT from the program to avoid accidental loss of the text file.

HELP. This command displays all 17 legal commands and their one-word definitions, to refresh your memory. It also tells how to return to the command mode, for those commands that do not automatically return.

Potential Program Problems

Speed is the most noticeable problem, but this is inherent in BASIC strings. The lack of speed shows up as brief pauses in program operation. As the text file becomes full, the pauses become more frequent and longer. Each time BASIC manipulates a string, it must assign a new location for it in string space. This quickly fills up string space, so a garbage collection routine is used to delete all of the old, no-longer-needed versions. This periodic garbage collection causes the pauses. Some other BASICs avoid this string-handling problem by requiring that the length of all strings be specified in advance, thus trading off flexibility and memory space for speed.

One partial solution to the speed problem is to install the Archbold TRS-80 clock control board. The OUT254,1 at the end of line 50 automatically increases speed by 50 percent if this has been done. Otherwise, it has no effect.

A program halt accompanied by a BASIC error message or from accidentally touching BREAK does not necessarily mean that your text file is lost. In most cases you can recover by typing GOTO60, then pressing ENTER. This puts you back into the command mode.

A trailing space in the 60th position on a line will cause a wrap-around underline marker on the next line when displayed. Such a trailing space can be removed by using EDIT, COMPILE, or JUSTIFY.

Under some conditions, an extra blank line may be inserted in the video display when text is being keyed in. This is not an extra line in the actual text file, as can be verified by using VIDEO.

When using EDIT, if you delete all printable characters following the cursor, it stops flashing and becomes solid. This does no harm; press L to get back to normal operation.

If a text line has more than 22 words and requires over 21 spaces to justify, an error will occur in JUSTIFY. To avoid this, redimension arrays S and T in line 30, but not too much, or you will run out of memory.

Program Modifications

This program requires 7226 bytes of memory as loaded, but uses nearly all the memory of a 16K TRS-80 when RUN. For a 16K TRS-80, key in the Program Listing in compressed format, *without* any of the extra spaces added for legibility.

A 16K Model III TRS-80 will require changing CLEAR 7400 to CLEAR 7130 and changing NL = 120 to NL = 116 in line 30. For a 32K TRS-80, change CLEAR7400 to CLEAR22000 and change NL = 120 to NL = 360 in line 30. This not only triples the size of the text file, but also increases the program's speed.

This program uses the ASCII control characters 17 and 20 as markers representing end-of-page and do-not-justify, respectively. If these characters are not NOPs (no operation) on your printer, change 17 in program lines 320, 1040, 1420, 1730, and 1930 and/or change 20 in program lines 470, 1050, 1180, 1420, and 1940 to values that do not affect your printer.

You may want to add extra features to this program. Be warned: There are less than 100 bytes of memory left unused. For everything added, something will have to be deleted. I have already minimized REMarks, used multiple statement lines, reused variables, and left out all spaces to save memory, as well as to increase speed.

To help with possible modification, I have included Table 4 showing arrays and variables. A brief REMark shows the entry point for each command. Even more brief REMarks show the entry points for subcommands in ADD and EDIT.

Arrays	
A$(120)	Lines of text
X$(3)	Buffer text lines for LOAD and SAVE
S(20)	Position to enter space for JUSTIFY
T(20)	Number of spaces to enter for JUSTIFY

Table continued

Variables

A$	Input entry character
B$	Blank to end of line (ASCII 30)—code may differ on some systems
C$	Cursor (graphics character 143)
F$	Format string for line number ("### ")
H$	Heading for each page
L$	Left part of text string
N$	Line numbers? ("Y" or "N")
PN$	Page numbers? ("Y" or "N")
P1$	Print page number for page number 1? ("Y"/"N")
Q$	Number "n" string for EDIT subcommands
R$	Right part of text string
S$	Space character (" ")
X$	Temporary text manipulation string
X$(0)	Temporary text line string for EDIT
A	ASCII value of text character
B	Beginning display line
C	Cursor position
D	Cursor displacement for wrap-around lines; temporary cursor position
F	First line for COMPILE, DELETE, or MOVE; found character flag for EDIT search (0,1)
FL	First line of text for VIDEO or PRINT
FP	First page number to be listed
I	Multi-use integer counter
IT	Insert text flag for INSERT (0 or 1)
J	Multi-use integer counter
K	Multi-use integer counter, L + 1
L	Line number of text
LA	Last line number of text
LL	Line length in characters
LM	Left margin to start PRINTing
M	Multi-use integer counter
N	Number "n" for EDIT subcommands; number of spaces to skip for JUSTIFY; first new line number for MOVE
NL	Number of lines of text
P	Position in text line, PRINT line number
PL	Page length in number of lines
Q	Temporary position in text line
R	Return or REPLACE flag (0 or 1)
S	Spacing between text lines
U	ASCII value of space (32)
X	Length of A$(L) string, page number
Y	Length of A$(L + 1) string, move cursor flag for EDIT (− 1 or + 1)
Z	Last line for COMPILE, DELETE, or MOVE

Table 4. *Arrays and variables*

Program Listing

```
 10 CLS :
    PRINT TAB(20)"BASIC WORD PROCESOR"
 20 :
    '  BY D.D.HINRICHS
 30 CLEAR 7400:
    DEFINT A - Z:
    NL = 120:
    DIM A$(NL),X$(3),S(20),T(20)
 40 B$ = CHR$(30):
    C$ = CHR$(143):
    F$ = "### ":
    N$ = "Y":
    PN$ = "N":
    P1$ = "Y"
 50 S$ = " ":
    H$ = S$:
    LA = - 1:
    P = 1:
    FP = 1:
    PL = 15:
    LL = 60:
    LM = 10:
    U = 32:
    OUT 254,1
 60 L = LA:
    IT = 0:
    R = 0:
    A$ = "":
    PRINT :
    INPUT "COMAND";A$:
    IF A$ = "" GOTO 80
 70 A = ASC(A$) - 64:
    IF A > 0 ON A GOTO 90,480,510,760,790,1220,80,1320,1350,1390,151
    0,1520,1580,80,80,1640,80,1750,1770,80,80,1830,80,1970
 80 PRINT "** ENTRY ERROR **":
    GOTO 60
 90 CLS :
    D = 0:
    N$ = "Y":
    IF LA < 0
     THEN
      L = 0:
      GOTO 130:
      :
      ' ADD
100 IF NL = LA + 1
     THEN
      210:
     ELSE
      IF L > FL + 12
       THEN
        B = L - 12:
       ELSE
        B = FL
110 FOR I = B TO L:
     X = LEN(A$(I)):
     D = D + (X + 3) / 64
120  GOSUB 1910:
     NEXT I:
    L = L + 1
130 C = (L - FL + D) * 64:
    IF C > 896 PRINT :
    PRINT :
    C = 896
140 PRINT @C, USING F$;L;:
    PRINT A$(L);:
    P = LEN(A$(L)) + 1:
```

```
    C = C + P + 3:
    K = L + 1
150 PRINT @C,C$;:
    A$ = INKEY$:
    PRINT @C,S$;:
    IF A$ = "" GOTO 150
160 GOSUB 290:
    ON A - 7 GOTO 360,410,310
170 IF A = 13
     THEN
      A$ = S$:
      GOTO 210:
     ELSE
      IF A = 24
       THEN
        380:
       ELSE
        IF A = 31 GOTO 460
180 IF A = 25
     THEN
      430:
     ELSE
      IF A = 26
       THEN
        330
190 IF A = 96 IF LA < L
     THEN
      LA = L:
      GOTO 60:
     ELSE
      60
200 PRINT @C,A$;:
    A$(L) = A$(L) + A$:
    IF P < = LL
     THEN
      P = P + 1:
      C = C + 1:
      GOTO 150
210 IF R GOTO 60:
     ELSE
      IF NL < = K PRINT :
      PRINT "FILE FULL":
      LA = NL - 1:
      GOTO 60
220 IF LEN(A$(K))
     THEN
      L = K:
      GOSUB 1360
230 IF K > LA
     THEN
      LA = K
240 IF A$ = S$ GOTO 280
250 FOR M = LL + 1 TO 2 STEP - 1:
      A$ = MID$(A$(L),M,1):
      IF A$ < > S$ NEXT M:
    GOTO 280
260 A$(K) = RIGHT$(A$(L),LL - M + 1):
    A$(L) = LEFT$(A$(L),M - 1)
270 PRINT @C - LL + M - 1,B$;:
    L = K:
    GOTO 130
280 A$(L) = LEFT$(A$(L),LL):
    L = K:
    GOTO 130
290 A = ASC(A$):
    IF A > 64 AND A < 91
     THEN
      A = A + U:
     ELSE
      IF A > 96 AND A < 123
       THEN
```

Program continued

```
        A = A - U
300 A$ = CHR$(A):
    RETURN
310 IF P > LL GOTO 210:
    :
    ' D
320 PRINT @C, CHR$(92);:
    A$(L) = A$(L) + CHR$(17):
    A$ = S$:
    GOTO 210
330 IF P > LL GOTO 210:
    :
    ' S-D
340 C = (L - FL + D) * 64 + 4:
    IF C > 900
     THEN
      C = 900
350 GOSUB 1180:
    P = 1:
    A$ = S$:
    GOTO 210
360 IF P = 1 GOTO 150:
    :
    ' L
370 C = C - 1:
    PRINT @C,B$;:
    P = P - 1:
    A$(L) = LEFT$(A$(L),P - 1):
    GOTO 150
380 IF P = 1 GOTO 150:
    :
    ' S-L
390 A$(L) = "":
    P = 1:
    C = (L - FL + D) * 64 + 4:
    IF C > 900
     THEN
      C = 900
400 PRINT @C,B$;:
    GOTO 150
410 IF P > LL - 6 GOTO 150:
    :
    ' R
420 A$(L) = A$(L) + STRING$(5,S$):
    C = C + 5:
    P = P + 5:
    GOTO 150
430 IF P > LL GOTO 210:
    :
    ' S-R
440 C = (L - FL + D) * 64 + 4:
    IF C > 900
     THEN
      C = 900
450 GOSUB 1200:
    P = 1:
    A$ = S$:
    GOTO 210
460 IF P > L GOTO 210:
    :
    ' CL
470 PRINT @C, CHR$(93);:
    A$(L) = A$(L) + CHR$(20):
    A$ = S$:
    GOTO 210
480 CLS :
    PRINT "DELETING BLANK LINES":
    FOR J = LA TO 0 STEP - 1:
     :
     ' BLANK
```

```
490  IF A$(J) = "" FOR I = J TO LA:
      A$(I) = A$(I + 1):
      NEXT I:
     A$(LA) = "":
     LA = LA - 1
500  NEXT J:
    IF R
     THEN
      RETURN :
     ELSE
      1830
510 INPUT "FIRST LINE TO COMPILE";F:
    IF F < 0
     THEN
      F = 0:
      :
      ' COMPILE
520 INPUT "LAST LINE TO COMPILE";Z:
    IF Z > LA
     THEN
      Z = LA
530 IF F > = Z
     THEN
      80:
     ELSE
      CLS :
      PRINT "COMPILING":
      FOR L = F TO Z - 1:
       K = L + 1
540  X = LEN(A$(L)):
     X$ = "":
     IF X < 2
      THEN
       620:
      ELSE
       IF X < = LL GOTO 600
550  FOR I = X TO 1 STEP - 1:
      A$ = MID$(A$(L),I,1)
560   IF A$ < > S$
       THEN
        X$ = A$ + X$:
        NEXT I:
       GOTO 600:
        :
      ELSE
       IF X$ = "" NEXT I
570 A = ASC( RIGHT$(X$,1)):
    IF A = 33 OR A = 46 OR A = 58 OR A = 63
     THEN
      X$ = X$ + "  "
580 A$(L) = LEFT$(A$(L),I - 1):
    IF LEN(A$(K)) = 0
     THEN
      A$(K) = X$:
      GOTO 540
590 A$(K) = X$ + S$ + A$(K):
    GOTO 540
600 X = LEN(A$(L)):
    IF X < 2
     THEN
      620:
     ELSE
      FOR I = X TO 2 STEP - 1
610  IF RIGHT$(A$(L),1) = S$
      THEN
       A$(L) = LEFT$(A$(L),I - 1):
       NEXT I
620 NEXT L:
    FOR L = F TO Z - 1:
     K = L + 1
```

Program continued

```
 630  X = LEN(A$(L)):
      Y = LEN(A$(K)):
      X$ = "":
      IF X = Ø OR Y = Ø GOTO 75Ø
 64Ø  A = ASC( RIGHT$(A$(L),1))
 65Ø  IF A = 33 OR A = 46 OR A = 58 OR A = 63
       THEN
        A$(L) = A$(L) + "  ":
        X = X + 2
 66Ø  FOR I = 1 TO Y:
       A$ = MID$(A$(K),I,1)
 67Ø   IF A$ < > S$
        THEN
         X$ = X$ + A$:
         NEXT I:
       ELSE
        IF X$ = "" NEXT I
 68Ø IF LL - X < I GOTO 71Ø
 69Ø Y = Y - I:
     IF Y < Ø
      THEN
       Y = Ø
 7ØØ A$(L) = A$(L) + S$ + X$:
     A$(K) = RIGHT$(A$(K),Y):
     GOTO 63Ø
 71Ø X = LEN(A$(L)):
     IF X < 2
      THEN
       73Ø:
      ELSE
       FOR I = X TO 2 STEP - 1
 72Ø  IF RIGHT$(A$(L),1) = S$
       THEN
        A$(L) = LEFT$(A$(L),I - 1):
        NEXT I
 73Ø IF Y < 2
      THEN
       75Ø:
      ELSE
       FOR I = Y TO 2 STEP - 1
 74Ø  IF LEFT$(A$(K),1) = S$
       THEN
        A$(K) = RIGHT$(A$(K),I - 1):
        NEXT I
 75Ø NEXT L:
     X = LEN(A$(Z)):
     GOTO 9ØØ
 76Ø INPUT "FIRST LINE TO DELETE";F:
     IF F < Ø
      THEN
       F = Ø:
       :
       ' DELETE
 77Ø INPUT "LAST LINE TO DELETE";Z:
     IF Z > LA
      THEN
       Z = LA
 78Ø IF F > Z
      THEN
       8Ø:
      ELSE
       FOR I = F TO Z:
        A$(I) = "":
        NEXT I:
      GOTO 183Ø
79Ø INPUT "EDIT LINE";L:
    IF L < Ø OR L > LA OR A$(L) = "" GOTO 8Ø:
    :
    ' EDIT
8ØØ C = 4:
    P = 1:
    X$(Ø) = A$(L):
    N$ = "Y"
810 CLS :
    I = L:
    GOSUB 191Ø:
    N = 1:
    Q$ = ""
820 GOSUB 91Ø:
    IF A > 47 AND A < 58
     THEN
      Q$ = Q$ + A$:
      N = VAL(Q$):
      GOTO 82Ø
830 M = Ø:
    IF A = 8
     THEN
      Y = - 1:
      GOSUB 94Ø:
     ELSE
      IF A = 9 OR A = U
       THEN
        Y = 1:
        GOSUB 94Ø
840 IF A = 97
     THEN
      A$(L) = X$(Ø):
      GOTO 8ØØ:
      :
      ' A
850 IF LEN(A$(L)) > = LL GOTO 87Ø
860 IF A = 25 GOSUB 12ØØ:
     ELSE
      IF A = 26 GOSUB 118Ø
870 IF A > 98 ON A - 98 GOSUB 96Ø,1ØØØ,196Ø,196Ø,196Ø,1Ø2Ø,1Ø3Ø
880 IF A = 115 GOSUB 112Ø:
     ELSE
      IF A = 12Ø GOSUB 117Ø:
       ELSE
        IF A = 1Ø8 GOTO 8ØØ
890 IF M = 1
     THEN
      N = 1:
      Q$ = "":
      GOTO 82Ø:
     ELSE
      IF R PRINT @32Ø,;:
       ELSE
        81Ø
900 IF LL < X PRINT "LINE";L;"HAS";X;"CHARACTERS":
    GOTO 6Ø:
     ELSE
      183Ø
910 X$ = MID$(A$(L),P,1)
920 PRINT @C,C$;:
    A$ = INKEY$:
    PRINT @C,X$;:
    IF A$ = "" GOTO 92Ø
930 GOSUB 29Ø:
    X = LEN(A$(L)):
    IF A = 13 OR A = 96
     THEN
      R = 1:
      RETURN :
     ELSE
      RETURN
940 M = 1:
    FOR I = 1 TO N:
      P = P + Y:
      IF P > X
```

Program continued

```
       THEN
        P = X:
        RETURN
950  IF P < 1
       THEN
        P = 1:
        RETURN :
       ELSE
        C = C + Y:
        NEXT I:
       RETURN
960 Q = P:
     D = C:
     FOR I = 1 TO N:
      GOSUB 910:
      IF R OR A = 27
       THEN
        P = Q:
        C = D:
        RETURN :
        :
        ' C
 970  PRINT @C,A$;:
      GOSUB 1100:
      P = P + 1:
      GOSUB 1110:
      A$(L) = L$ + A$ + R$
 980  A = U:
      C = C + 1:
      IF P < = X NEXT I
 990 P = Q:
     C = D:
     RETURN
1000 IF P + N - 1 > X
      THEN
       N = X - P + 1:
       :
       ' D
1010 GOSUB 1100:
     Q = P:
     P = P + N:
     GOSUB 1110:
     A$(L) = L$ + R$:
     P = Q:
     RETURN
1020 GOSUB 1100:
     A$(L) = L$ + S$:
     PRINT @C,B$:
     :
     ' H
1030 GOSUB 910:
     IF R OR A = 27 RETURN :
     :
     ' I
1040 IF A = 10
      THEN
       A$(L) = A$(L) + CHR$(17):
       R = 1:
       RETURN
1050 IF A = 31
      THEN
       A$(L) = A$(L) + CHR$(20):
       R = 1:
       RETURN
1060 PRINT @C,A$;:
     IF A = 8
      THEN
       Y = - 1:
       GOSUB 940:
       GOTO 1030
```

```
1070 IF A = 9
      THEN
       Y = 1:
       GOSUB 940:
       GOTO 1030:
      ELSE
       IF P > X
        THEN
         X = P
1080 GOSUB 1100:
     GOSUB 1110:
     A$(L) = L$ + A$ + R$:
     PRINT @C,B$;A$ + R$
1090 C = C + 1:
     P = P + 1:
     GOTO 1030
1100 L$ = "":
     IF P < 2 RETURN :
      ELSE
       L$ = LEFT$(A$(L),P - 1):
       RETURN
1110 R$ = "":
     IF P > X RETURN :
      ELSE
       R$ = RIGHT$(A$(L),X - P + 1):
       RETURN
1120 GOSUB 910:
     Q = P:
     D = C:
     :
     ' S
1130 FOR I = 1 TO N:
      F = 0:
      FOR J = Q + 1 TO X:
       D = D + 1
1140   IF MID$(A$(L),J,1) = A$
        THEN
         F = 1:
         Q = J:
         J = X
1150   NEXT J:
      NEXT I:
     IF F
      THEN
       P = Q:
       C = D
1160 A = U:
     RETURN
1170 A$(L) = A$(L) + S$:
     P = X + 1:
     C = P + 3:
     GOTO 1030:
     :
     ' X
1180 A$(L) = STRING$((LL - LEN(A$(L))) / 2,32) + A$(L) + CHR$(20):
     :
     ' S-D
1190 PRINT @C,B$;A$(L); CHR$(93);:
     RETURN
1200 A$(L) = STRING$(LL - LEN(A$(L)),32) + A$(L):
     :
     ' S-R
1210 PRINT @C,B$;A$(L);:
     RETURN
1220 CLS :
     PRINT "LINE LENGTH =";LL,:
     INPUT "NEW =";LL:
     :
     ' FORMAT
1230 PRINT "LINE SPACES =";S,:
```

Program continued

```
     INPUT "NEW =";S
1240 PRINT "LINE NOS. =  '";N$;"'",:
     INPUT "NEW (Y/N)";N$
1250 PRINT "FIRST LINE = ";FL,:
     INPUT "NEW =";FL
1260 PRINT "LEFT MARGIN =";LM,:
     INPUT "NEW =";LM
1270 PRINT "PAGE LENGTH =";PL,:
     INPUT "NEW =";PL
1280 PRINT "PAGE NOS. =  '";PN$;"'",:
     INPUT "NEW (Y/N)";PN$
1290 PRINT "FIRST PAGE = ";FP,:
     INPUT "NEW =";FP
1300 PRINT "PAGE 1 NO. = '";P1$;"'",:
     INPUT "NEW (Y/N)";P1$
1310 PRINT "HEADING = '";H$;"'    ",:
     INPUT "NEW (Y/N)";H$:
     GOTO 60
1320 CLS :
     PRINT "LEGAL COMMANDS ARE:":
     :
     ' HELP
1330 PRINT "A ADD","B BLANK","C COMPILE","D DELETE","E EDIT","F FORMA
     T","H HELP","I INSERT","J JUSTIFY","K KILL","L LOAD","M MOVE","P
      PRINT","R REPLACE","S SAVE","V VIDEO","X EXIT"
1340 PRINT "KEY 'SHIFT-@' TWICE TO RETURN FROM A,E,I,R TO COMMAND MOD
     E":
     GOTO 60
1350 INPUT "INSERT AT LINE";L:
     IF L < 0 OR L > LA GOTO 80:
     :
     ' INSERT
1360 IF NL = LA + 1 PRINT "FILE FULL":
     GOTO 60:
      ELSE
       IF R GOTO 60
1370 FOR I = LA TO L STEP - 1:
      A$(I + 1) = A$(I):
      NEXT I
1380 A$(L) = "":
     LA = LA + 1:
     L = L - 1:
     IF IT RETURN :
      ELSE
       IT = 1:
       GOTO 90
1390 CLS :
     PRINT "JUSTIFYING":
     FOR L = 0 TO LA:
      X = LEN(A$(L)):
      :
      ' JUSTIFY
1400  IF X < 2 GOTO 1500:
       ELSE
        FOR I = X TO 2 STEP - 1:
         A = ASC( RIGHT$(A$(L),1))
1410   IF A = U
        THEN
         A$(L) = LEFT$(A$(L),I - 1):
         X = X - 1:
         NEXT I
1420  IF X > = LL OR A = 17 OR A = 20
       THEN
        1500:
       ELSE
        J = 0:
        K = 1:
        FOR I = 1 TO X
1430   IF MID$(A$(L),I,1) < > S$
        THEN
        K = Ø:
       ELSE
        IF K = Ø
         THEN
          K = 1:
          S(J) = I:
          J = J + 1
1440   NEXT I:
      IF J = Ø GOTO 1500
1450  K = RND(J) - 1:
      IF INT(J / 2) = J / 2 OR J = 1
       THEN
        N = 1:
       ELSE
        N = 2
1460  FOR I = 1 TO LL - X:
       T(K) = T(K) + 1:
       K = K + N:
       IF K > J - 1
        THEN
         K = K - J
1470   NEXT I:
      FOR I = J - 1 TO Ø STEP - 1:
       A$ = STRING$(T(I),S$):
       T(I) = Ø
1480   A$(L) = LEFT$(A$(L),S(I)) + A$ + RIGHT$(A$(L), LEN(A$(L))
       - S(I))
1490   NEXT I
1500  NEXT L:
     GOTO 1830
1510 CLS :
     INPUT "REALLY KILL (Y/N)";A$:
     IF A$ = "Y"
      THEN
       RUN :
      ELSE
       60:
       :
       ' KILL
1520 GOSUB 1820:
     CLS :
     PRINT "LOADING":
     :
     ' LOAD
1530 INPUT # - 1,LA,LL,S,N$,FL,LM,PL,PN$,FP,P1$,H$
1540 FOR I = Ø TO LA STEP 4:
      INPUT # - 1,X$(Ø),X$(1),X$(2),X$(3)
1550  FOR J = Ø TO 3:
       L = I + J:
       X = LEN(X$(J)):
       A$(L) = "":
       IF X < 1 GOTO 1570
1560   FOR K = 1 TO X:
        A$(L) = A$(L) + CHR$( ASC( MID$(X$(J),K,1)) - 128):
        NEXT K
1570   NEXT J:
      NEXT I:
     GOTO 60
1580 INPUT "FIRST LINE TO MOVE";F:
     IF F < Ø
      THEN
       F = Ø:
       :
       ' MOVE
1590 INPUT "LAST LINE TO MOVE";Z:
     IF Z > LA
      THEN
       Z = LA
1600 IF F > Z
      THEN
```

Program continued

```
        80:
       ELSE
        INPUT "FIRST NEW LINE";N:
        FOR I = F TO Z
1610   IF LEN(A$(N)) PRINT "LINE";N;"NOT EMPTY":
       GOTO 60
1620   A$(N) = A$(I):
       A$(I) = "":
       N = N + 1:
       IF N > LA
        THEN
         LA = N
1630   NEXT I:
      GOTO 1830
1640  IF PEEK(14312) < 128
       THEN
        X = FP:
        M = FL:
        GOTO 1670:
        :
        ' PRINT
1650  INPUT "PRINTER NOT READY.  ABORT (Y/N)";A$
1660  IF A$ = "Y"
       THEN
        60:
       ELSE
        INPUT "PRESS ENTER";A$:
        GOTO 1640
1670  R = 1:
      GOSUB 480:
      CLS :
      PRINT "PRINTING"
1680  LPRINT CHR$(30):
      IF PN$ < > "Y" OR (P1$ = "N" AND X = 1) GOTO 1700
1690  LPRINT AB(LM)H$; TAB(LL + LM - 7)"PAGE"; USING "###";X:
      LPRINT TAB(LM)
1700  FOR P = M TO M + PL - 1:
       IF P > LA GOTO 1740
1710  M=M+1:
      IF S LPRINT STRING$(S-1,138)
1720   LPRINT TAB(LM);:
       IF N$ = "Y" LPRINT USING F$;P;
1730   LPRINT A$(P):
       IF ASC( RIGHT$(A$(P),1)) < > 17 NEXT P
1740  LPRINT CHR$(28):
      IF P > LA
       THEN
        60:
       ELSE
        X = X + 1:
        GOTO 1680
1750  INPUT "REPLACE LINE";L:
      IF L < 0 OR L > LA GOTO 80:
      :
      ' REPLACE
1760  R = 1:
      A$(L) = "":
      L = L - 1:
      GOTO 90
1770  GOSUB 1820:
      CLS :
      PRINT "SAVING":
      :
      ' SAVE
1780  PRINT # - 1,LA,LL,S,N$,FL,LM,PL,PN$,FP,P1$,H$:
      FOR L = 0 TO LA STEP 4
1790   FOR J = 0 TO 3:
        I = L + J:
        X = LEN(A$(I)):
        X$(J) = "":
```

```
       IF X < 1 GOTO 1810
1800   FOR K = 1 TO X:
        X$(J) = X$(J) + CHR$( ASC( MID$(A$(I),K,1))
        NEXT K
1810   NEXT J:
      PRINT # - 1,X$(0),X$(1),X$(2),X$(3):
      NEXT L:
     GOTO 60
1820 INPUT "READY CASSETTE, THEN PRESS ENTER";A$:
     RETURN
1830 CLS :
     X = FP - 1:
     FOR M = FL TO LA STEP PL:
      X = X + 1:
      :
      ' VIDEO
1840  IF P1$ = "N" AND X = 1 GOTO 1860
1850  IF PN$ = "Y" PRINT H$; TAB(LL - 7)"PAGE" USING "###";X:
      PRINT
1860  FOR I = M TO M + PL - 1:
       IF I > LA GOTO 1890
1870   IF S PRINT STRING$(S - 1,10)
1880   GOSUB 1910
1890   NEXT I:
      A$ = "":
      IF I < = LA INPUT "PRESS ENTER";A$:
      IF A$ < > ""M = LA
1900  NEXT M:
     L = LA:
     GOTO 60
1910 Y = LEN(A$(I)):
     IF Y
      THEN
       A = ASC( RIGHT$(A$(I),1)):
      ELSE
       A = 0
1920 IF N$ = "Y" PRINT USING F$;I;
1930 PRINT A$(I);:
     IF A = 17 PRINT CHR$(92);
1940 IF A = 20 PRINT CHR$(93);:
      ELSE
       IF A = U PRINT CHR$(95);
1950 IF N$ < > "Y" OR Y < > 60 PRINT
1960 RETURN
1970 CLS :
     INPUT "REALLY EXIT (Y/N)";A$:
     IF A$ < > "Y" GOTO 60:
     :
     ' EXIT
1980 CLS :
     CLEAR 50:
     OUT 254,0:
     END
```

States Worked: A Program for Radio Amateurs

by Edward G. McCloskey KB3CK

There are many systems for keeping track of which states have been worked or confirmed on each band. I've tried index cards, notebooks, and filing systems, but I could never find one system that was both easy to maintain and allowed quick retrieval of information. The following program meets all my requirements, and it will most likely be an improvement over any written system you may be using. With this program, you can display the status of each state for an entire band, display each state with its status on each band, list all states not confirmed on any band, and change the status of any state with a single keystroke.

Three status codes indicate that a state is: (1) not worked, (2) worked, but not confirmed, and (3) confirmed. Table 1 explains each of the nine menu functions and provides the necessary keying instructions for each function. The state names are arranged alphabetically by call district, with Hawaii and Alaska at the end of the list (see lines 35120 and 35130 in the Program Listing). The abbreviations used are those found in the *U.S. Callbook.* Table 2 contains an alphabetical list of all symbols used in the program along with a comment, line number, and memory location for each.

Menu No.	Function Performed
1.	Start New Index: Displays the status of all states one band at a time, starting with 10 meters and allowing status changes to be made for any state. The cursor (left arrow) points to the state for which changes will be accepted. Press 1, 2, or 3 to change the displayed status. The new code will be displayed as the cursor moves to the next state. The cursor may be moved backwards to change any input errors by pressing B. After all desired changes have been made, press X to move to the next band. This function is intended for initialization of the status codes, but may be used at any time.
2.	Input Data Tape: Reads all status codes from tape, and returns to the menu when all status codes have been read.
3.	Review Specific Band: Displays one band only (your choice), and returns to the menu whenever any key (except shift) is pressed. This is a display function only. No changes can be made to the status codes.
4.	Record Data Tape: Records all status codes on tape, and returns to the menu when all status codes have been recorded. A new data tape must be recorded any time the status of a state has been changed.
5.	Review Specific State: Displays any state, along with its status on each band. The status code for any band may now be changed using the same keys as described under menu function 1. The cursor (right arrow) points to the status code that may be changed.

6.	Review All Data: Displays all bands in succession, starting with 10 meters and ending with mixed band. Press any key to move to the next band. This is a display function only.
7.	Review Mixed Band W.A.S.: Displays all states and the status of each. Press any key to return to the menu. This is a display function only.
8.	List Needed States: Displays all states not at status 3 on each band, starting with 10 meters and ending with mixed band. Press any key to return to the menu. This is a display function only.
9.	End Session: Displays a reminder message to record a new data tape if any changes have been made. Press any key to enter BASIC. Press X to return to the menu.

Table 1. *Menu functions*

I have attempted to document this program in a way that will be useful to beginners. The comments in the Program Listing are intended to detail what each line is doing, while the comments in Table 2 explain the operation of each individual routine.

Program Operation

Menu Display: Lines 300 through 500 display all menu choices and the input statement for choice selection. The message statement used to display the menu is contained in lines 35000 through 35020. Whenever a slash (/) is encountered in the message statement, program execution jumps to line 450, where the video location for the next character is updated. An exclamation point is used to signal the end of the message statement. These two methods avoid unecessary line numbers and reduce the number of spaces needed in each message line. Lines 520 through 760 examine the keyboard input from line 540 and send program execution to the selected function. Any character other than 1 through 9 returns execution to line 300, and the above procedure is repeated until proper keyboard input is received.

New Index: Lines 1000 through 1200 display all state names and draw the graphics display. Calls to A1, A5, A2, A4, and A8 are made to display the status codes for each state on 10, 15, 20, 40, and 80 meters, respectively. A call to UP is made for each display to allow the status codes to be changed.

Input Data: Lines 2000 through 2210 read the data from tape, loading 300 bytes of data into memory locations 48D4H through 49FFH. Execution returns to line 300 after all data has been read.

Review Band: Lines 3000 through 3750 contain the routines used to review each of the five bands. The first function of this section is to determine which band you want displayed. Lines 3050 and 3060 get two characters from the keyboard. The first character must be a 1, 2, 4, or 8, and the second character must be a 5 or 0, otherwise execution returns to line 3000, and the above procedure is repeated until correct input is received. Once a correct input is found, program execution jumps to X1, X5, X2, X4, or X8.

Record Data Tape: Lines 4000 through 4210 record 300 bytes of data from memory locations 48D4H through 49FFH onto tape, returning to the menu display when all data has been recorded.

Review Specific State: Lines 5000 through 5090 determine which state is to be reviewed, based on keyboard input at line 5050. Lines 5100 through 5310 search the state list until a match with the input is found, with the IX register being used to point to the location of the correct status code. Line 5110 checks for the end of the state list if the keyboard input does not match any state, and execution returns to line 5000 for another input. Lines 5320 through 5440 display a heading and the name of the state being reviewed. Lines 5450 through 5580 display the status codes for the state being reviewed and the key instructions. Lines 5590 through 5620 restore the HL register to the 10-meter status code and put the cursor on screen. Lines 5630 through 5740 check for correct keyboard input, sending program execution to the selected operation. Line 5750 puts the new status code into memory, and line 5760 displays the new code on the screen. Lines 5770 through 5780 are used to move the cursor and adjust the HL and IY registers accordingly.

Review All Data: Lines 6000 through 6120 display each band in succession, starting with 10 meters and ending with 80 meters, then going to the mixed band routine to display the mixed band status.

Review Mixed Band W.A.S.: Lines 7000 through 7130 display all data with calls to subroutines QP and MX. Lines 7300 through 7360 reset all status codes for mixed band to 1. Lines 7400 through 7520 check the status of each state on all five bands, sending execution to line 7900 each time a status code greater than 1 is found. Lines 7900 through 7920 put the status codes found into the proper memory locations for mixed band.

List Needed States: All bands are displayed by lines 8000 through 8340, with each band requiring seven program lines. Calls to LP and MP are made for each band being displayed. The LP routine (lines 8700 through 8770) displays the heading and sets the memory pointers to the location of the status codes for the band being displayed. The MP routine finds all states not at status 3 by calling P4 to display the state name.

End Session: Lines 9000 through 9060 display a message on the screen as a reminder to record a new data tape if any changes have been made to any of the status codes.

Print Subroutine: Lines 25000 through 25060 are used to display all standard message statements throughout the program. Upon entry to this routine, the IY register holds the address of the first character to be displayed, and the IX register holds the video location where the first character is to be printed. Line 25010 checks for the end of the message statement and returns to the calling routine whenever an ! is found.

Display Status Subroutine: Lines 25100 through 25290 are used to display the status codes of each state whenever an entire band is being displayed. Upon entry, the IX register holds the address of the first status code to be displayed. The C register is used to count five columns for display, and the B register is used to count 10 rows in each column. The IY register is used as a pointer for the screen locations, with the HL register helping out when moving from the bottom of one column to the top of the next.

Display States Subroutine: Lines 25500 through 25730 are used to display the state names whenever an entire band is displayed. The operation of this routine is basically the same as that of the display status subroutine with some differences. The roles of the IX and IY registers are reversed, and there is no need to count columns, since the end of this routine occurs when an exclamation point is encountered at line 25570. This frees the B register to be used to count the three bytes of each state name. After all state names are displayed, execution jumps to GH to draw the graphics display.

Graphics Display Subroutine: Lines 25740 through 25970 draw the graphics display whenever an entire band is displayed. The C register counts 10 graphics characters per vertical line, and the B register counts six vertical lines. The DE register is used to increment the video location by 12 spaces each time a graphics character is printed, and by four spaces each time a row of six is completed. Lines 25920 through 25970 draw the bottom line of the display.

Display Band Subroutines: Subroutines A1, A5, A2, A4, and A8 are used to display 10, 15, 20, 40, and 80 meters respectively. All are identical except for the data location (D1, D2, etc.) and message statement (MA, MB, etc.) used.

Update Data Subroutine: This routine is used along with the new index routine. Program control is passed to this routine after each band is displayed. Upon entry, the HL register contains the address of the first data byte of each band. Lines 27000 through 27030 set the column count, row count, and video pointers. Lines 27040 through 27180 get and evaluate keyboard input, sending program execution to the desired function when proper input is received. Lines 27190 through 27270 store and display new status codes, adjust all necessary counters and pointers, and return to line 27040 when completed. Lines 27280 through 27360 are used when the cursor has reached the bottom of any column, moving the cursor to the top of the next column, and resetting the row counter. Lines 27370 through 27900 are used when the cursor is to be back spaced. Special routines are used when the cursor is at the bottom of the last column, at the top of column one, and at the top of any other column.

When the cursor reaches the bottom of the last column, and a status code (1, 2, or 3) has been entered for Alaska, the cursor has reached a limit and will continue to point to AK. The only keyboard inputs that will be accepted are B and X.

The remainder of the Program Listing (lines 35000 through 35180) consists of the message statements used throughout the program. The ASCII codes for each message are listed in lines of 16 bytes each with the beginning address for each 16-byte line shown.

SYMBOL	ADDR	LINE	FUNCTION
A1	4F0F	26200	Display 10 meter status
A2	4F35	26400	Display 20 meter status
A4	4F48	26500	Display 40 meter status
A5	4F22	26300	Display 15 meter status
A8	4F5B	26600	Display 80 meter status
AP	4ECD	25680	Move to next column
B1	4EE6	25790	Display one graphics row
B3	4F04	25920	Draw bottom graphics line
BP	4EF1	25840	Move to next row
BS	4FC4	27370	Setup to back space cursor
CB	5022	27840	Back space or move to next function
CH	4A41	520	Determine menu choice selected
D1	4906	140	First byte of 10 meter data
D2	496A	160	First byte of 20 meter data
D4	499C	170	First byte of 40 meter data
D5	4938	150	First byte of 15 meter data
D8	49CE	180	First byte of 80 meter data
DM	4A08	240	Set all data to status 1
DP	4E92	25220	Move to next column
DS	4EA0	25500	Display state names
DX	48D4	130	First byte of mixed band data
EP	4EAF	25550	Display one column
ES	4E4E	9000	Start of end session routine
F2	4F9E	27190	Change status code
F3	4FB2	27280	Move cursor to next column
F4	4FDF	27520	Move cursor back one column
F5	4FFD	27690	Move cursor back to beginning
F6	5012	27780	At end of display, get input
F7	4FD0	27420	Move cursor back one row
FP	4F7A	27040	Get keyboard input
FS	4C2D	5100	Search for first letter match
G1	4C45	5210	Check second letter
GB	4B07	3030	Get keyboard input for band
GH	4ED9	25740	Display graphics
HP	4CA3	5630	Get one byte from keyboard
IN	4AC2	2000	Start of tape input routine
K1	4D46	7420	Setup to move data (50 bytes)
K2	4D4C	7440	Check data to be moved
LS	4D64	8000	Start of list needed states routine
LP	4E11	8700	Display message and set counters
M1	5034	35000	Menu display message
M2	5153	35030	Input data tape message
M3	5173	35040	Band choice message
M4	519A	35060	Record data tape message
M5	51C2	35070	State choice message
M6	51E4	35080	Mixed band message
M7	51EF	35090	List needed states message

continued

M8	520E	35110	End session message
M9	5270	35120	State list message
MA	5190	35050	10 meters message
MB	5204	35100	15 meters message
MC	534B	35160	20 meters message
MD	5355	35170	40 meters message
ME	4D5D	7900	Move data
MF	535F	35180	80 meters message
MG	5307	35140	Heading message for review state
MJ	5330	35150	Key instructions message
MN	4A22	340	Check message M1 for / or !
MP	4DFE	8600	Determine if state is needed
MU	4A14	300	Display menu
MV	4CC1	5770	Move cursor and data pointer
MX	4D40	7400	Setup to move data (5 bands)
NP	4C76	5450	Display data
NW	4A81	1000	Start of new index routine
O	4BEE	4090	Output data (150 bytes)
P2	4C5A	5320	Print heading
P3	4E7F	25140	Display one column
P4	4E29	8780	Display needed state and its status
PM	4A36	450	Add 32 spaces if / is found
PT	4E64	25000	Routine to print messages
Q1	4D38	7330	Put 1 in status location
QP	4D30	7300	Set mixed band data to 1
RA	4CDE	6000	Start of review all data routine
RB	4AFC	3000	Start of review band routine
RC	4BD1	4000	Start of record data routine
RS	4C0B	5000	Start of review state routine
RX	4D05	7000	Start of review mixed band routine
SS	4E74	25100	Display status
T	4BF6	4120	Record byte on tape
U	4AE7	2120	Read byte from tape
UP	4F6E	27000	Start of update data routine
V	3C00	190	Screen location 0
W	4ADF	2090	Input data tape (150 bytes)
X1	4B3B	3300	Display 10 meters
X2	4B77	3480	Display 20 meters
X4	4B95	3580	Display 40 meters
X5	4B59	3390	Display 15 meters
X8	4BB3	3670	Display 80 meters
XV	4B21	3160	Check keyboard input
XX	4B28	3200	Check keyboard input

Table 2. *Symbols used in the program*

As for reference material, the book I found most useful is *How to Program the Z80* by Rodnay Zaks. Other reference material included *TRS-80 Assembly Language Programming* by William Barden Jr., the Editor/Assembler manual, and many of the articles in *80 Microcomputing* and *Kilobaud Microcomputing*.

Program Listing

```
                Ø0100
                ØØ110
                ØØ120
48D4            ØØ130 DX      EQU     48D4H         ;MIXED BAND STARTS HERE
49Ø6            ØØ140 D1      EQU     49Ø6H         ;1Ø METERS STARTS HERE
4938            ØØ150 D5      EQU     4938H         ;15 METERS STARTS HERE
496A            ØØ160 D2      EQU     496AH         ;2Ø METERS STARTS HERE
499C            ØØ170 D4      EQU     499CH         ;4Ø METERS STARTS HERE
49CE            ØØ180 D8      EQU     49CEH         ;8Ø METERS STARTS HERE
3CØØ            ØØ190 V       EQU     3CØØH         ;SCREEN LOCATION Ø
4AØØ            ØØ200         ORG     4AØØH         ;START PROGRAM AT 4AØØH
4AØØ DD21D448   ØØ210         LD      IX,DX         ;STARTING AT MIXED BAND
4AØ4 Ø696       ØØ220         LD      B,15Ø         ;     FOR 15Ø COUNTS
4AØ6 3E31       ØØ230         LD      A,31H         ;     PUT AN ASCII 1
4AØ8 DD77ØØ     ØØ240 DM      LD      (IX),A        ;     INTO THIS LOCATION
4AØB DD77Ø1     ØØ250         LD      (IX+1),A      ;AND THIS LOCATION
4AØE DD23       ØØ260         INC     IX            ;ADJUST MEMORY POINTER
4A1Ø DD23       ØØ270         INC     IX            ;ADJUST MEMORY POINTER
4A12 1ØF4       ØØ280         DJNZ    DM            ;IF B <>Ø THEN 240
4A14 CDC9Ø1     ØØ300 MU      CALL    Ø1C9H         ;CLEAR SCREEN
4A17 FD21345Ø   ØØ310         LD      IY,M1         ;POINT TO MESSAGE M1
4A1B DD21163C   ØØ320         LD      IX,V+22       ;PRINT AT 22
4A1F 21ØØ3C     ØØ330         LD      HL,V          ;POINT TO SCREEN LOC Ø
4A22 FD7EØØ     ØØ340 MN      LD      A,(IY)        ;GET MESSAGE CHARACTER
4A25 FE2F       ØØ350         CP      '/'           ;CHECK FOR A "/"
4A27 28ØD       ØØ360         JR      Z,PM          ;IF / FOUND THEN 45Ø
4A29 FE21       ØØ370         CP      '!'           ;CHECK FOR "!"
4A2B 2814       ØØ380         JR      Z,CH          ;IF ! FOUND THEN 52Ø
4A2D DD77ØØ     ØØ390         LD      (IX),A        ;PRINT MESSAGE CHARACTER
4A3Ø DD23       ØØ400         INC     IX            ;NEXT SCREEN LOCATION
4A32 FD23       ØØ410         INC     IY            ;NEXT MESSAGE CHARACTER
4A34 18EC       ØØ420         JR      MN            ;GO FOR NEXT CHARACTER
4A36 112ØØØ     ØØ450 PM      LD      DE,32         ;ADD 32 SPACES
4A39 19         ØØ460         ADD     HL,DE         ;  TO HL REGISTER
4A3A E5         ØØ470         PUSH    HL            ;  FOR FOR PRINT LOCATION
4A3B DDE1       ØØ480         POP     IX            ;  OF NEXT CHARACTER
4A3D FD23       ØØ490         INC     IY            ;ADJUST MESSAGE POINTER
4A3F 18E1       ØØ500         JR      MN            ;GET NEXT CHARACTER
4A41 DD21B93F   ØØ520 CH      LD      IX,V+953      ;SCREEN LOCATION 953
4A45 DD222Ø4Ø   ØØ530         LD      (4Ø2ØH),IX    ;STORE CURSOR LOCATION
4A49 CDB31B     ØØ540         CALL    1BB3H         ;GET KEYBOARD INPUT
4A4C 23         ØØ550         INC     HL            ;POINT TO INPUT
4A4D CDC9Ø1     ØØ560         CALL    Ø1C9H         ;CLEAR SCREEN
4A5Ø 7E         ØØ570         LD      A,(HL)        ;PUT INPUT IN A REGISTER
4A51 FE39       ØØ580         CP      '9'           ;IS INPUT A 9
4A53 CA4E4E     ØØ590         JP      Z,ES          ;IF YES THEN 9ØØØ
4A56 FE38       ØØ600         CP      '8'           ;IS INPUT AN 8
4A58 CA644D     ØØ610         JP      Z,LS          ;IF YES THEN 8ØØØ
4A5B FE37       ØØ620         CP      '7'           ;IS INPUT A 7
4A5D CAØ54D     ØØ630         JP      Z,RX          ;IF YES THEN 7ØØØ
4A6Ø FE36       ØØ640         CP      '6'           ;IS INPUT A 6
4A62 CADE4C     ØØ650         JP      Z,RA          ;IF YES THEN 6ØØØ
4A65 FE35       ØØ660         CP      '5'           ;IS INPUT A 5
4A67 CAØB4C     ØØ670         JP      Z,RS          ;IF YES THEN 5ØØØ
4A6A FE34       ØØ680         CP      '4'           ;IS INPUT A 4
4A6C CAD14B     ØØ690         JP      Z,RC          ;IF YES THEN 4ØØØ
4A6F FE33       ØØ700         CP      '3'           ;IS INPUT A 3
4A71 CAFC4A     ØØ710         JP      Z,RB          ;IF YES THEN 3ØØØ
4A74 FE32       ØØ720         CP      '2'           ;IS INPUT A 2
4A76 CAC24A     ØØ730         JP      Z,IN          ;IF YES THEN 2ØØØ
4A79 FE31       ØØ740         CP      '1'           ;IS INPUT A 1
4A7B CA814A     ØØ750         JP      Z,NW          ;IF YES THEN 1ØØØ
4A7E C3144A     ØØ760         JP      MU            ;BAD INPUT - GO AGAIN
4A81 CDC9Ø1     Ø1000 NW      CALL    Ø1C9H         ;CLEAR SCREEN
4A84 CDAØ4E     Ø1010         CALL    DS            ;DISPLAY STATES/GRAPHICS
4A87 FD212F53   Ø1020         LD      IY,MJ         ;GET MESSAGE LOCATION
4A8B DD21403F   Ø1030         LD      IX,V+832      ;PRINT AT 832
4A8F CD644E     Ø1040         CALL    PT            ;  MESSAGE MJ
```

```
4A92 CDØF4F    Ø1Ø5Ø           CALL    A1            ;DISPLAY 1Ø METERS
4A95 21Ø649    Ø1Ø6Ø           LD      HL,D1         ;POINT TO 1Ø METER STATUS
4A98 CD6E4F    Ø1Ø7Ø           CALL    UP            ;  UPDATE DATA
4A9B CD224F    Ø1Ø8Ø           CALL    A5            ;DISPLAY 15 METERS
4A9E 213849    Ø1Ø9Ø           LD      HL,D5         ;POINT TO 15 METER STATUS
4AA1 CD6E4F    Ø11ØØ           CALL    UP            ;  UPDATE DATA
4AA4 CD354F    Ø111Ø           CALL    A2            ;DISPLAY 2Ø METERS
4AA7 216A49    Ø112Ø           LD      HL,D2         ;POINT TO 2Ø METER STATUS
4AAA CD6E4F    Ø113Ø           CALL    UP            ;UPDATE DATA
4AAD CD484F    Ø114Ø           CALL    A4            ;DISPLAY 4Ø METERS
4ABØ 219C49    Ø115Ø           LD      HL,D4         ;POINT TO 4Ø METER STATUS
4AB3 CD6E4F    Ø116Ø           CALL    UP            ;UPDATE DATA
4AB6 CD5B4F    Ø117Ø           CALL    A8            ;DISPLAY 8Ø METERS
4AB9 21CE49    Ø118Ø           LD      HL,D8         ;POINT TO 8Ø METER STATUS
4ABC CD6E4F    Ø119Ø           CALL    UP            ;UPDATE DATA
4ABF C3144A    Ø12ØØ           JP      MU            ;GOTO 3ØØ -MENU-
4AC2 FD215351  Ø2ØØØ IN        LD      IY,M2         ;GET MESSAGE LOCATION
4AC6 DD21CØ3D  Ø2Ø1Ø           LD      IX,V+448      ;PRINT AT 448
4ACA CD644E    Ø2Ø2Ø           CALL    PT            ;  MESSAGE M8
4ACD DD21E33D  Ø2Ø3Ø           LD      IX,V+483      ;SCREEN LOCATION 483
4AD1 DD22204Ø  Ø2Ø4Ø           LD      (4Ø2ØH),IX    ;NEW CURSOR LOCATION
4AD5 CDB31B    Ø2Ø5Ø           CALL    1BB3H         ;GET KEYBOARD INPUT
4AD8 ØEØ2      Ø2Ø6Ø           LD      C,2           ;COUNT 2 DATA BLOCKS
4ADA Ø696      Ø2Ø7Ø           LD      B,15Ø         ;  15Ø BYTES PER BLOCK
4ADC 21D448    Ø2Ø8Ø           LD      HL,DX         ;STARTING AT MIXED BAND
4ADF 3EØØ      Ø2Ø9Ø W         LD      A,Ø           ;DEFINE TAPE DRIVE
4AE1 CD12Ø2    Ø21ØØ           CALL    212H          ;INPUT FROM TAPE
4AE4 CD96Ø2    Ø211Ø           CALL    296H          ;FIND SYNC BYTE
4AE7 CD35Ø2    Ø212Ø U         CALL    235H          ;READ ONE BYTE
4AEA 77        Ø213Ø           LD      (HL),A        ;STORE BYTE
4AEB 23        Ø214Ø           INC     HL            ;ADJUST MEMORY POINTER
4AEC 1ØF9      Ø215Ø           DJNZ    U             ;IF B <>Ø GET NEXT BYTE
4AEE CDF8Ø1    Ø216Ø           CALL    Ø1F8H         ;TURN ON CASSETTE
4AF1 Ø696      Ø217Ø           LD      B,15Ø         ;RESET BYTE COUNT
4AF3 216A49    Ø218Ø           LD      HL,D2         ;START AT 2Ø METERS
4AF6 ØD        Ø219Ø           DEC     C             ;ONE LESS DATA BLOCK
4AF7 2ØE6      Ø22ØØ           JR      NZ,W          ;GET 15Ø MORE BYTES
4AF9 C3144A    Ø221Ø           JP      MU            ;GOTO 3ØØ -MENU-
4AFC FD217351  Ø3ØØØ RB        LD      IY,M3         ;FIND MESSAGE LOCATION
4BØØ DD21CØ3D  Ø3Ø1Ø           LD      IX,V+448      ;PRINT AT 448
4BØ4 CD644E    Ø3Ø2Ø           CALL    PT            ;  MESSAGE M3
4BØ7 DD21F43D  Ø3Ø3Ø GB        LD      IX,V+5ØØ      ;SCREEN LOCATION 5ØØ
4BØB DD22204Ø  Ø3Ø4Ø           LD      (4Ø2ØH),IX    ;NEW CURSOR LOCATION
4BØF CDB31B    Ø3Ø5Ø           CALL    1BB3H         ;GET KEYBOARD INPUT
4B12 D7        Ø3Ø6Ø           RST     1ØH           ;POINT TO FIRST CHATACTER
4B13 4E        Ø3Ø7Ø           LD      C,(HL)        ;PUT CHARACTER IN C
4B14 23        Ø3Ø8Ø           INC     HL            ;POINT TO NEXT CHARACTER
4B15 46        Ø3Ø9Ø           LD      B,(HL)        ;STORE SECOND CHARACTER
4B16 78        Ø31ØØ           LD      A,B           ;   INTO A
4B17 FE3Ø      Ø311Ø           CP      'Ø'           ;IS 2ND CHARACTER Ø
4B19 28ØD      Ø312Ø           JR      Z,XX          ;IF YES THEN 32ØØ
4B1B FE35      Ø313Ø           CP      '5'           ;IS 2ND CHARACTER 5
4B1D 28Ø2      Ø314Ø           JR      Z,XV          ;IF YES THEN 316Ø
4B1F 18E6      Ø315Ø           JR      GB            ;BAD INPUT - GOTO 3Ø3Ø
4B21 79        Ø316Ø XV        LD      A,C           ;PUT 1ST CHARACTER IN A
4B22 FE31      Ø317Ø           CP      '1'           ;IS CHARACTER 1
4B24 2833      Ø318Ø           JR      Z,X5          ;IF YES THEN 339Ø
4B26 18DF      Ø319Ø           JR      GB            ;IF NO THEN 3Ø3Ø
4B28 79        Ø32ØØ XX        LD      A,C           ;PUT 1ST CHARACTER IN A
4B29 FE31      Ø321Ø           CP      '1'           ;IS CHARACTER 1
4B2B 28ØE      Ø322Ø           JR      Z,X1          ;IF YES THEN 33ØØ
4B2D FE32      Ø323Ø           CP      '2'           ;IS CHARACTER 2
4B2F 2846      Ø324Ø           JR      Z,X2          ;IF YES THEN 348Ø
4B31 FE34      Ø325Ø           CP      '4'           ;IS CHARACTER 4
4B33 286Ø      Ø326Ø           JR      Z,X4          ;IF YES THEN 358Ø
4B35 FE38      Ø327Ø           CP      '8'           ;IS CHARACTER 8
4B37 287A      Ø328Ø           JR      Z,X8          ;IF YES THEN 367Ø
4B39 18CC      Ø329Ø           JR      GB            ;BAD INPUT -GOTO 3Ø3Ø
4B3B CDC9Ø1    Ø33ØØ X1        CALL    Ø1C9H         ;CLEAR SCREEN
4B3E CDAØ4E    Ø331Ø           CALL    DS            ;DISPLAY STATES/GRAPHICS
```

Program continued

```
4B41 DD210649 03320           LD       IX,D1          ;FIND 10 METER DATA
4B45 CD744E   03330           CALL     SS             ;DISPLAY 10 METER STATUS
4B48 FD219051 03340           LD       IY,MA          ;FIND MESSAGE MA
4B4C DD219A3E 03350           LD       IX,V+666       ;PRINT AT 666
4B50 CD644E   03360           CALL     PT             ;   MESSAGE MA
4B53 CD4900   03370           CALL     049H           ;GET KEYBOARD INPUT
4B56 C3144A   03380           JP       MU             ;GOTO 300 -MENU-
4B59 CDC901   03390 X5        CALL     01C9H          ;CLEAR SCREEN
4B5C CDA04E   03400           CALL     DS             ;DISPLAY STATES/GRAPHICS
4B5F DD213849 03410           LD       IX,D5          ;FIND 15 METER DATA
4B63 CD744E   03420           CALL     SS             ;DISPLAY 15 METER STATUS
4B66 FD210452 03430           LD       IY,MB          ;FIND MESSAGE MB
4B6A DD219A3E 03440           LD       IX,V+666       ;PRINT AT 666
4B6E CD644E   03450           CALL     PT             ;   MESSAGE MB
4B71 CD4900   03460           CALL     049H           ;GET KEYBOARD INPUT
4B74 C3144A   03470           JP       MU             ;GOTO 300 -MENU-
4B77 CDC901   03480 X2        CALL     01C9H          ;CLEAR SCREEN
4B7A CDA04E   03490           CALL     DS             ;DISPLAY STATES/GRAPHICS
4B7D DD216A49 03500           LD       IX,D2          ;FIND 20 METER STATUS
4B81 CD744E   03510           CALL     SS             ;DISPLAY 20 METER STATUS
4B84 FD214A53 03520           LD       IY,MC          ;FIND MESSAGE MC
4B88 DD219A3E 03530           LD       IX,V+666       ;PRINT AT 666
4B8C CD644E   03540           CALL     PT             ;   MESSAGE  MC
4B8F CD4900   03560           CALL     049H           ;GET KEYBOARD INPUT
4B92 C3144A   03570           JP       MU             ;GOTO 300 -MENU-
4B95 CDC901   03580 X4        CALL     01C9H          ;CLEAR SCREEN
4B98 CDA04E   03590           CALL     DS             ;DISPLAY STATES/GRAPHICS
4B9B DD219C49 03600           LD       IX,D4          ;FIND 40 METER STATUS
4B9F CD744E   03610           CALL     SS             ;DISPLAY 40 METER STATUS
4BA2 FD215453 03620           LD       IY,MD          ;FIND MESSAGE MD
4BA6 DD219A3E 03630           LD       IX,V+666       ;PRINT AT 666
4BAA CD644E   03640           CALL     PT             ;  MESSAGE MD
4BAD CD4900   03650           CALL     049H           ;GET KEYBOARD INPUT
4BB0 C3144A   03660           JP       MU             ;GOTO 300 -MENU-
4BB3 CDC901   03670 X8        CALL     01C9H          ;CLEAR SCREEN
4BB6 CDA04E   03680           CALL     DS             ;DISPLAY STATES/GRAPHICS
4BB9 DD21CE49 03690           LD       IX,D8          ;FIND 80 METER STATUS
4BBD CD744E   03700           CALL     SS             ;DISPLAY 80 METER STATUS
4BC0 FD215E53 03710           LD       IY,MF          ;FIND MESSAGE MF
4BC4 DD219A3E 03720           LD       IX,V+666       ;PRINT AT 666
4BC8 CD644E   03730           CALL     PT             ;  MESSAGE MF
4BCB CD4900   03740           CALL     049H           ;GET KEYBOARD INPUT
4BCE C3144A   03750           JP       MU             ;GOTO 300 -MENU-
4BD1 FD219A51 04000 RC        LD       IY,M4          ;FIND MESSAGE M4
4BD5 DD21C03D 04010           LD       IX,V+448       ;PRINT AT 448
4BD9 CD644E   04020           CALL     PT             ;MESSAGE M4
4BDC DD21E93D 04030           LD       IX,V+489       ;SCREEN LOCATION 489
4BE0 DD222040 04040           LD       (4020H),IX     ;NEW CURSOR LOCATION
4BE4 CDB31B   04050           CALL     1BB3H          ;GET KEYBOARD INPUT
4BE7 0E02     04060           LD       C,2            ;COUNT 2 DATA BLOCKS
4BE9 0696     04070           LD       B,150          ;  OF 150 BYTES EACH
4BEB 21D448   04080           LD       HL,DX          ;STARTING AT MIXED BAND
4BEE 3E00     04090 O         LD       A,0            ;OUTPUT TO CASSETTE
4BF0 CD1202   04100           CALL     212H           ;DEFINE CASSETTE DRIVE
4BF3 CD8702   04110           CALL     287H           ;WRITE LEADER & SYNC BYTE
4BF6 7E       04120 T         LD       A,(HL)         ;PUT BYTE IN A
4BF7 CD6402   04130           CALL     264H           ;RECORD DATA BYTE
4BFA 23       04140           INC      HL             ;GET NEXT BYTE
4BFB 10F9     04150           DJNZ     T              ;IF B <>0 RECORD MORE
4BFD CDF801   04160           CALL     01F8H          ;TURN OFF CASSETTE
4C00 0696     04170           LD       B,150          ;RESTE BYTE COUNT
4C02 216A49   04180           LD       HL,D2          ;START AT 20 METERS
4C05 0D       04190           DEC      C              ;ADJUST BLOCK COUNT
4C06 20E6     04200           JR       NZ,O           ;IF NOT 0 DO NEXT BLOCK
4C08 C3144A   04210           JP       MU             ;GOTO 300 -MENU-
4C0B FD21C251 05000 RS        LD       IY,M5          ;FIND MESSAGE M5
4C0F DD21C03D 05010           LD       IX,V+448       ;PRINT AT 440
4C13 CD644E   05020           CALL     PT             ;MESSAGE M5
4C16 DD21E43D 05030           LD       IX,V+484       ;SCREEN LOCATION 484
4C1A DD222040 05040           LD       (4020H),IX     ;NEW CURSOR LOCATION
4C1E CDB31B   05050           CALL     1BB3H          ;GET KEYBOARD INPUT
```

```
4C21 D7         05060         RST     10H          ;POINT TO FIRST CHARACTER
4C22 FD217052 05070           LD      IY,M9        ;FIND MESSAGE M9
4C26 DD210649 05080           LD      IX,D1        ;FIND 10 METER DATA
4C2A CDC901     05090         CALL    01C9H        ;CLEAR SCREEN
4C2D FD7E00     05100 FS      LD      A,(IY)       ;GET FIRST CHARACTER
4C30 FE21       05110         CP      '!'          ;CHECK FOR END OF MESSAGE
4C32 28D7       05120         JR      Z,RS         ;IF FOUND - BAD INPUT
4C34 7E         05130         LD      A,(HL)       ;FIRST LETTER OF INPUT
4C35 FD4600     05140         LD      B,(IY)       ;LETTER FROM STATE LIST
4C38 B8         05150         CP      B            ;DO THEY MATCH
4C39 280A       05160         JR      Z,G1         ;IF YES CHECK 2ND LETTER
4C3B 110300     05170         LD      DE,3         ;IF NO GET NEXT STATE
4C3E FD19       05180         ADD     IY,DE        ;  BY MOVING 3 PLACES
4C40 DD23       05190         INC     IX           ;ADJUST DATA POINTER
4C42 C32D4C     05200         JP      FS           ;CHECK NEXT STATE
4C45 23         05210 G1      INC     HL           ;POINT TO 2ND CHARACTER
4C46 7E         05220         LD      A,(HL)       ;  & PUT IT IN A
4C47 FD23       05230         INC     IY           ;GET SECOND STATE LETTER
4C49 FD4E00     05240         LD      C,(IY)       ;  & PUT IT IN C
4C4C B9         05250         CP      C            ;DOES IT MATCH 'A'
4C4D 280B       05260         JR      Z,P2         ;IF YES GOTO 5320
4C4F 110200     05270         LD      DE,2         ;IN NO GET NEXT STATE
4C52 FD19       05280         ADD     IY,DE        ;  BY MOVING 2 PLACES
4C54 2B         05290         DEC     HL           ;GET FIRST INPUT LETTER
4C55 DD23       05300         INC     IX           ;ADJUST DATA POINTER
4C57 C32D4C     05310         JP      FS           ;CHECK NEXT STATE
4C5A CDC901     05320 P2      CALL    01C9H        ;CLEAR SCREEN
4C5D FD210653 05330           LD      IY,MG        ;FIND MESSAGE MG
4C61 DDE5       05340         PUSH    IX           ;SAVE DATA POINTER
4C63 E1         05350         POP     HL           ;  INTO HL REGISTER
4C64 DD21803D 05360           LD      IX,V+384     ;PRINT AT 384
4C68 CD644E     05370         CALL    PT           ;  MESSAGE MG
4C6B E5         05380         PUSH    HL           ;SAVE DATA POINTER
4C6C DDE1       05390         POP     IX           ;  INTO IX REGISTER
4C6E 21C23D     05400         LD      HL,V+450     ;PRINT AT 450
4C71 70         05410         LD      (HL),B       ;FIRST STATE LETTER
4C72 23         05420         INC     HL           ;PRINT AT 450+1
4C73 71         05430         LD      (HL),C       ;SECOND STATE LETTER
4C74 0E05       05440         LD      C,5          ;COUNT 5 DATA BYTES
4C76 110700     05450 NP      LD      DE,7         ;  7 SPACES APART
4C79 19         05460         ADD     HL,DE        ;MOVE 7 SPACES
4C7A DD4600     05470         LD      B,(IX)       ;GET FIRST DATA BYTE
4C7D 70         05480         LD      (HL),B       ;PRINT FIRST BYTE
4C7E 113200     05490         LD      DE,50        ;MOVE DATA POINTER 50
4C81 DD19       05500         ADD     IX,DE        ;  FOR NEXT DATA BYTE
4C83 0D         05510         DEC     C            ;DECREMENT BYTE COUNT
4C84 20F0       05520         JR      NZ,NP        ;IF C<>0 DO MORE
4C86 0E05       05530         LD      C,5          ;FOR 5 CURSOR LOCATIONS
4C88 FD212F53 05540           LD      IY,MJ        ;FIND MESSAGE MJ
4C8C DDE5       05550         PUSH    IX           ;SAVE DATA POINTER
4C8E E1         05560         POP     HL           ;  INTO HL
4C8F DD21403F 05570           LD      IX,V+832     ;PRINT AT 832
4C93 CD644E     05580         CALL    PT           ;MESSAGE MJ
4C96 11FA00     05590 JP      LD      DE,250       ;TO RETURN DATA POINTER
4C99 ED52       05600         SBC     HL,DE        ;  TO 10 METER STATUS
4C9B DD21C93D 05610           LD      IX,V+457     ;SCREEN LOCATION 457
4C9F DD36005E 05620           LD      (IX),5EH     ;DISPLAY CURSOR
4CA3 CD4900     05630 HP      CALL    049H         ;GET KEYBOARD INPUT
4CA6 FE58       05640         CP      'X'          ;IS INPUT X
4CA8 CA144A     05650         JP      Z,MU         ;IF YES GOTO 300 -MENU-
4CAB FE42       05660         CP      'B'          ;IS INPUT B
4CAD 2812       05670         JR      Z,MV         ;IF YES THEN 5770
4CAF FE31       05680         CP      '1'          ;IS INPUT 1
4CB1 280A       05690         JR      Z,NS         ;IF YES THEN 5750
4CB3 FE32       05700         CP      '2'          ;IS INPUT 2
4CB5 2806       05710         JR      Z,NS         ;IF YES THEN 5750
4CB7 FE33       05720         CP      '3'          ;IS INPUT 3
4CB9 2802       05730         JR      Z,NS         ;IF YES THEN 5750
4CBB 20E6       05740         JR      NZ,HP        ;BAD INPUT-GOTO 5630
4CBD 77         05750 NS      LD      (HL),A       ;STORE NEW STATUS
4CBE DD7701     05760         LD      (IX+1),A     ;DISPLAY NEW STATUS
```

Program continued

```
4CC1 113200   05770 MV    LD      DE,50       ;ADJUST DATA POINTER
4CC4 19       05780       ADD     HL,DE       ;   TO NEXT BAND
4CC5 DD360020 05790       LD      (IX),20H    ;ERASE OLD CURSOR
4CC9 110700   05800       LD      DE,7        ;MOVE CURSOR 7 SPACES
4CCC DD19     05810       ADD     IX,DE       ;   FOR NEXT BAND
4CCE DD36005E 05820       LD      (IX),5EH    ;DISPLAY NEW CURSOR
4CD2 0D       05830       DEC     C           ;ADJUST BAND COUNTER
4CD3 20CE     05840       JR      NZ,HP       ;IF NOT 0 GET INPUT
4CD5 0E05     05850       LD      C,5         ;RESET BAND COUNTER
4CD7 DD360020 05860       LD      (IX),20H    ;ERASE OLD CURSOR
4CDB C3964C   05870       JP      JP          ;  AND START OVER
4CDE CDC901   06000 RA    CALL    01C9H       ;CLEAR SCREEN
4CE1 CDA04E   06010       CALL    DS          ;DISPLAY STATES/GRAPHICS
4CE4 CD0F4F   06020       CALL    A1          ;DISPLAY 10 METER STATUS
4CE7 CD4900   06030       CALL    049H        ;GET KEYBOARD INPUT
4CEA CD224F   06040       CALL    A5          ;DISPLAY 15 METER STATUS
4CED CD4900   06050       CALL    049H        ;GET KEYBOARD STATUS
4CF0 CD354F   06060       CALL    A2          ;DISPLAY 20 METER STATUS
4CF3 CD4900   06070       CALL    049H        ;GET KEYBOARD INPUT
4CF6 CD484F   06080       CALL    A4          ;DISPLAY 40 METER STATUS
4CF9 CD4900   06090       CALL    049H        ;GET KEYBOARD INPUT
4CFC CD5B4F   06100       CALL    A8          ;DISPLAY 80 METER STATUS
4CFF CD4900   06110       CALL    049H        ;GET KEYBOARD INPUT
4D02 C3054D   06120       JP      RX          ;GOTO 7000
4D05 CDC901   07000 RX    CALL    01C9H       ;CLEAR SCREEN
4D08 CD304D   07010       CALL    QP          ;SET ALL DATA TO 1
4D0B 2632     07020       LD      H,32H       ;ASCII 2 IN H
4D0D CD404D   07030       CALL    MX          ;MOVE ALL 2'S
4D10 2633     07040       LD      H,33H       ;ASCII 3 IN H
4D12 CD404D   07050       CALL    MX          ;MOVE ALL 3'S
4D15 CDA04E   07060       CALL    DS          ;DISPLAY STATES/GRAPHICS
4D18 FD21E451 07070       LD      IY,M6       ;FIND MESSAGE M6
4D1C DD21993E 07080       LD      IX,V+665    ;PRINT AT 665
4D20 CD644E   07090       CALL    PT          ;MESSAGE M6
4D23 DD21D448 07100       LD      IX,DX       ;FIND MIXED BAND DATA
4D27 CD744E   07110       CALL    SS          ;DISPLAY STATUS
4D2A CD4900   07120       CALL    049H        ;GET KEYBOARD INPUT
4D2D C3144A   07130       JP      MU          ;GOTO 300 -MENU-
4D30 DD21D448 07300 QP    LD      IX,DX       ;FIND MIXED BAND DATA
4D34 0632     07310       LD      B,50        ;COUNT 50 BYTES
4D36 3E31     07320       LD      A,31H       ;ASCII 1 IN H
4D38 DD7700   07330 Q1    LD      (IX),A      ;  AND INTO DATA LOCATION
4D3B DD23     07340       INC     IX          ;ADJUST DATA POINTER
4D3D 10F9     07350       DJNZ    Q1          ;GOTO 7330 UNITL 50 BYTES
4D3F C9       07360       RET                 ;  ARE DONE THEN RETURN
4D40 0605     07400 MX    LD      B,5         ;COUNT 5 BANDS
4D42 DD210649 07410       LD      IX,D1       ;START AT 10 METER DATA
4D46 FD21D448 07420 K1    LD      IY,DX       ;POINT TO MIXED BAND
4D4A 0E32     07430       LD      C,50        ;COUNT 50 STATES
4D4C DD7E00   07440 K2    LD      A,(IX)      ;PUT DATA BYTE IN A
4D4F BC       07450       CP      H           ;IS IT A MATCH
4D50 CC5D4D   07460       CALL    Z,ME        ;IF YES THEN 7900
4D53 DD23     07470       INC     IX          ;ADJUST DATA POINTER
4D55 FD23     07480       INC     IY          ;ADJUST MIXED BAND
4D57 0D       07490       DEC     C           ;ADJUST STATE COUNT
4D58 20F2     07500       JR      NZ,K2       ;IF NOT 0 GOTO 7440
4D5A 10EA     07510       DJNZ    K1          ;ADJUST BAND COUNTER
4D5C C9       07520       RET                 ;RETURN AFTER 5 DONE
4D5D DD7E00   07900 ME    LD      A,(IX)      ;STORE NEW STATUS
4D60 FD7700   07910       LD      (IY),A      ;  IN MIXED BAND LOCATION
4D63 C9       07920       RET                 ;GET NEXT BYTE
4D64 CDC901   08000 LS    CALL    01C9H       ;CLEAR SCREEN
4D67 FD219051 08010       LD      IY,MA       ;FIND MESSAGE MA
4D6B DD21163C 08020       LD      IX,V+22     ;PRINT AT 22
4D6F CD644E   08030       CALL    PT          ;  MESSAGE MA
4D72 CD114E   08040       CALL    LP          ;DISPLAY MESSAGE M7
4D75 210649   08050       LD      HL,D1       ;START AT 10 METERS
4D78 CDFE4D   08060       CALL    MP          ;DISPLAY NEEDED STATES
4D7B CDC901   08070       CALL    01C9H       ;CLEAR SCREEN
4D7E FD210452 08080       LD      IY,MB       ;FIND MESSAGE MB
```

```
4D82 DD21163C          LD    IX,V+22     ;PRINT AT 22
4D86 CD644E            CALL  PT          ;  MESSAGE MB
4D89 CD114E            CALL  LP          ;DISPLAY MESSAGE M7
4D8C 213849            LD    HL,D5       ;FIND 15 METER DATA
4D8F CDFE4D            CALL  MP          ;DISPLAY NEEDED STATES
4D92 CDC9Ø1            CALL  Ø1C9H       ;CLEAR SCREEN
4D95 FD214A53          LD    IY,MC       ;FIND MESSAGE MC
4D99 DD21163C          LD    IX,V+22     ;PRINT AT 22
4D9D CD644E            CALL  PT          ;MESSAGE MC
4DAØ CD114E            CALL  LP          ;DISPLAY MESSAGE M7
4DA3 216A49            LD    HL,D2       ;FIND 2Ø METER STATUS
4DA6 CDFE4D            CALL  MP          ;DISPLAY NEEDED STATES
4DA9 CDC9Ø1            CALL  Ø1C9H       ;CLEAR SCREEN
4DAC FD215453          LD    IY,MD       ;FIND MESSAGE MD
4DBØ DD21163C          LD    IX,V+22     ;PRINT AT 22
4DB4 CD644E            CALL  PT          ;MESSAGE MD
4DB7 CD114E            CALL  LP          ;DISPLAY MESSAGE M7
4DBA 219C49            LD    HL,D4       ;FIND 4Ø METER DATA
4DBD CDFE4D            CALL  MP          ;DISPLAY NEEDED STATES
4DCØ CDC9Ø1            CALL  Ø1C9H       ;CLEAR SCREEN
4DC3 FD215E53          LD    IY,MF       ;FIND MESSAGE MF
4DC7 DD21163C          LD    IX,V+22     ;PRINT AT 22
4DCB CD644E            CALL  PT          ;  MESSAGE MF
4DCE CD114E            CALL  LP          ;DISPLAY MESSAGE M7
4DD1 21CE49            LD    HL,D8       ;FIND 8Ø METER DATA
4DD4 CDFE4D            CALL  MP          ;DISPLAY NEEDED STATES
4DD7 CD3Ø4D            CALL  QP          ;MIXED BAND ROUTINE
4DDA 2632              LD    H,32H       ;ASCII 2 IN H
4DDC CD4Ø4D            CALL  MX          ;MOVE ALL 2'S
4DDF 2633              LD    H,33H       ;ASCII 3 IN H
4DE1 CD4Ø4D            CALL  MX          ;MOVE ALL 3'S
4DE4 CDC9Ø1            CALL  Ø1C9H       ;CLEAR SCREEN
4DE7 FD21E451          LD    IY,M6       ;FIND MESSAGE M6
4DEB DD21163C          LD    IX,V+22     ;PRINT AT 22
4DEF CD644E            CALL  PT          ;  MESSAGE M6
4DF2 CD114E            CALL  LP          ;DISPLAY MESSAGE M7
4DF5 21D448            LD    HL,DX       ;FIND MIXED BAND DATA
4DF8 CDFE4D            CALL  MP          ;DISPLAY NEEDED STATES
4DFB C3144A            JP    MU          ;GOTO 3ØØ -MENU-
4DFE 7E             MP LD    A,(HL)      ;PUT STATUS CODE IN A
4DFF FE33              CP    '3'         ;IS CODE A 3
4EØ1 C4294E            CALL  NZ,P4       ;IF NO IT IS NEEDED
4EØ4 23                INC   HL          ;ADJUST STATUS POINTER
4EØ5 11Ø3ØØ            LD    DE,3        ;MOVE MESSAGE POINTER
4EØ8 FD19              ADD   IY,DE       ;  TO NEXT STATE
4EØA ØD                DEC   C           ;ADJUST STATE COUNT
4EØB 2ØF1              JR    NZ,MP       ;IF NOT Ø DO NEXT STATE
4EØD CD49ØØ            CALL  Ø49H        ;GET KEYBOARD INPUT
4E1Ø C9                RET               ;  AND RETURN
4E11 FD21EF51       LP LD    IY,M7       ;FIND MESSAGE M7
4E15 DD21ØØ3C          LD    IX,V        ;PRINT AT Ø
4E19 CD644E            CALL  PT          ; MESSAGE M7
4E1C FD217Ø52          LD    IY,M9       ;FIND MESSAGE M9
4E2Ø DD218Ø3C          LD    IX,V+128    ;START PRINTING AT 128
4E24 ØE32              LD    C,5Ø        ;  5Ø STATES
4E26 Ø6Ø6              LD    B,6         ;  IN 6 COLUMNS
4E28 C9                RET               ;  AND RETURN
4E29 FD7EØØ         P4 LD    A,(IY)      ;PUT FIRST LETTER IN A
4E2C DD77ØØ            LD    (IX),A      ;PRINT FIRST LETTER
4E2F FD7EØ1            LD    A,(IY+1)    ;2ND LETTER IN A
4E32 DD77Ø1            LD    (IX+1),A    ;PRINT 2ND LETTER
4E35 FD7EØ2            LD    A,(IY+2)    ;3RD LETTER IN A
4E38 DD77Ø2            LD    (IX+2),A    ;PRINT 3RD LETTER
4E3B 7E                LD    A,(HL)      ;PUT STATUS CODE IN A
4E3C DD77Ø3            LD    (IX+3),A    ;PRINT STATUS CODE
4E3F 11ØAØØ            LD    DE,1Ø       ;MOVE 1Ø SPAES
4E42 DD19              ADD   IX,DE       ;  FOR NEXT STATE
4E44 Ø5                DEC   B           ;ADJUST COLUMN COUNT
4E45 CØ                RET   NZ          ;IF NOT Ø DO NEXT STATE
4E46 11Ø4ØØ            LD    DE,4        ;MOVE 4 SPACES
```

Program continued

```
4E49 DD19       08910         ADD    IX,DE       ;  TO NEXT ROW
4E4B 0606       08920         LD     B,6         ;RESET COLUMN COUNT
4E4D C9         08930         RET                ;  AND RETURN
4E4E FD210E52   09000 ES      LD     IY,M8       ;FIND MESSAGE M8
4E52 DD21C03D   09010         LD     IX,V+448    ;PRINT AT 448
4E56 CD644E     09020         CALL   PT          ;  MESSAGE M8
4E59 CD4900     09030         CALL   049H        ;GET KEYBOARD INPUT
4E5C FE58       09040         CP     'X'         ;IS INPUT X
4E5E CA144A     09050         JP     Z,MU        ;IF YES GOTO 300 -MENU-
4E61 C37500     09060         JP     75H         ;IF NO GOTO MEMORY SIZE
4E64 FD7E00     25000 PT      LD     A,(IY)      ;PUT MESSAGE LETTER IN A
4E67 FE21       25010         CP     '!'         ;IS MESSAGE DONE
4E69 C8         25020         RET    Z           ;IF YES RETURN
4E6A DD7700     25030         LD     (IX),A      ;PRINT MESSAGE LETTER
4E6D FD23       25040         INC    IY          ;ADJUST MESSAGE POINTER
4E6F DD23       25050         INC    IX          ;ADJUST SCREEN POINTER
4E71 C3644E     25060         JP     PT          ;GET NEXT MESSAGE LETTER
4E74 0E05       25100 SS      LD     C,5         ;COUNT 5 COLUMNS
4E76 060A       25110         LD     B,10        ;COUNT 10 ROWS
4E78 FD21063C   25120         LD     IY,V+6      ;START PRINTING AT 6
4E7C 21063C     25130         LD     HL,V+6      ;COLUMN VIDEO POINTER
4E7F 114000     25140 P3      LD     DE,64       ;READY FOR NEXT ROW
4E82 DD7E00     25150         LD     A,(IX)      ;PUT STATUS CODE IN A
4E85 FD7700     25160         LD     (IY),A      ;PRINT STATUS CODE
4E88 DD23       25170         INC    IX          ;ADJUST STATUS POINTER
4E8A 05         25180         DEC    B           ;ADJUST ROW COUNT
4E8B 2805       25190         JR     Z,DP        ;IF 0 GOTO 25220
4E8D FD19       25200         ADD    IY,DE       ;ADJUST VIDEO POINTER
4E8F C37F4E     25210         JP     P3          ;GET NEXT STATUS CODE
4E92 0D         25220 DP      DEC    C           ;ADJUST COLUMN COUNT
4E93 C8         25230         RET    Z           ;IF 5 DONE RETURN
4E94 060A       25240         LD     B,10        ;RESET ROW COUNT
4E96 110C00     25250         LD     DE,12       ;MOVE 12 SPACES
4E99 19         25260         ADD    HL,DE       ;  TO NEXT COLUMN
4E9A E5         25270         PUSH   HL          ;TRANSFER COLUMN POINTER
4E9B FDE1       25280         POP    IY          ;  TO VIDEO POINTER
4E9D C37F4E     25290         JP     P3          ;GET NEXT STATUS CODE
4EA0 FD217052   25500 DS      LD     IY,M9       ;FIND MESSAGE M9
4EA4 DD21013C   25510         LD     IX,V+1      ;START PRINTING AT 1
4EA8 21013C     25520         LD     HL,V+1      ;COLUMN VIDEO POINTER
4EAB 0E0A       25530         LD     C,10        ;COUNT 10 ROWS
4EAD 0603       25540         LD     B,3         ;EACH STATE = 3 BYTES
4EAF 113D00     25550 EP      LD     DE,61       ;61 SPACES TO NEXT ROW
4EB2 FD7E00     25560         LD     A,(IY)      ;MESSAGE LETTER IN A
4EB5 FE21       25570         CP     '!'         ;IS MESSAGE DONE
4EB7 CAD94E     25580         JP     Z,GH        ;IF YES DRAW GRAPHICS
4EBA DD7700     25590         LD     (IX),A      ;PRINT MESSAGE LETTER
4EBD DD23       25600         INC    IX          ;MOVE 1 SPACE
4EBF FD23       25610         INC    IY          ;GET NEXT LETTER
4EC1 10EC       25620         DJNZ   EP          ;IF B <>0 GOTO 25550
4EC3 0603       25630         LD     B,3         ;RESET BYTE COUNT
4EC5 0D         25640         DEC    C           ;ADJUST ROW COUNT
4EC6 2805       25650         JR     Z,AP        ;IF C=0 NEXT COLUMN
4EC8 DD19       25660         ADD    IX,DE       ;  ELSE NEXT ROW
4ECA C3AF4E     25670         JP     EP          ;NEXT STATE-GOTO 25550
4ECD 110C00     25680 AP      LD     DE,12       ;MOVE 12 SPACES
4ED0 19         25690         ADD    HL,DE       ;  TO NEXT COLUMN
4ED1 E5         25700         PUSH   HL          ;TRANSFER COLUMN POINTER
4ED2 DDE1       25710         POP    IX          ;  TO VIDEO POINTER
4ED4 0E0A       25720         LD     C,10        ;RESET ROW COUNT
4ED6 C3AF4E     25730         JP     EP          ;GOTO 25550-NEXT STATE
4ED9 DD21003C   25740 GH      LD     IX,V        ;STARTING AT 0
4EDD 0E0A       25750         LD     C,10        ;COUNT 10 ROWS
4EDF 110C00     25760         LD     DE,12       ;PUT 12 SPACES BETWEEN
4EE2 0606       25770         LD     B,6         ; EACH OF 6 COLUMNS
4EE4 3E95       25780         LD     A,95H       ;GRAPHICS CHARACTER IN A
4EE6 DD7700     25790 B1      LD     (IX),A      ;DISPLAY ONE CHARACTER
4EE9 05         25800         DEC    B           ;ADJUST COLUMN COUNT
4EEA 2805       25810         JR     Z,BP        ;IF B=0 NEXT ROW
4EEC DD19       25820         ADD    IX,DE       ;ELSE MOVE 12 SPACES
```

```
4EEE C3E64E   25830          JP      B1             ;AND DO NEXT COLUMN
4EF1 0606     25840 BP       LD      B,6            ;RESET COLUMN COUNT
4EF3 110400   25850          LD      DE,4           ;ADD 4 SPACES
4EF6 DD19     25860          ADD     IX,DE          ;FOR NEXT ROW
4EF8 110C00   25870          LD      DE,12          ;RESTORE COLUMN OFFSET
4EFB 0D       25880          DEC     C              ;ADJUST ROW COUNT
4EFC 20E8     25890          JR      NZ,B1          ;IF C<>0 DO MORE
4EFE DD21803E 25900          LD      IX,V+640       ;START OF BOTTOM LINE
4F02 0E3D     25910          LD      C,61           ;61 SPACES LONG
4F04 3E83     25920 B3       LD      A,83H          ;GRAPHICS CHARACTER IN A
4F06 DD7700   25930          LD      (IX),A         ;PRINT GRAPHICS BLOCK
4F09 DD23     25940          INC     IX             ;MOVE 1 SPACE
4F0B 0D       25950          DEC     C              ;ADJUST COUNT
4F0C 20F6     25960          JR      NZ,B3          ;IF C<>0 DO MORE
4F0E C9       25970          RET                    ;  ELSE RETURN
4F0F DD210649 26200 A1       LD      IX,D1          ;FIND 10 METER DATA
4F13 CD744E   26210          CALL    SS             ;DISPLAY STATUS
4F16 FD219051 26220          LD      IY,MA          ;FIND MESSAGE MA
4F1A DD219A3E 26230          LD      IX,V+666       ;PRINT AT 666
4F1E CD644E   26240          CALL    PT             ;  MESSAGE MA
4F21 C9       26250          RET                    ;     AND RETURN
4F22 DD213849 26300 A5       LD      IX,D5          ;FIND 15 METER DATA
4F26 CD744E   26310          CALL    SS             ;DISPLAY STATUS
4F29 FD210452 26320          LD      IY,MB          ;FIND MESSAGE MB
4F2D DD219A3E 26330          LD      IX,V+666       ;PRINT AT 666
4F31 CD644E   26340          CALL    PT             ;  MESSAGE MB
4F34 C9       26350          RET                    ;     AND RETURN
4F35 DD216A49 26400 A2       LD      IX,D2          ;FIND 20 METER STATUS
4F39 CD744E   26410          CALL    SS             ;DISPLAY STATUS
4F3C FD214A53 26420          LD      IY,MC          ;FIND MESSAGE MC
4F40 DD219A3E 26430          LD      IX,V+666       ;PRINT AT 666
4F44 CD644E   26440          CALL    PT             ;  MESSAGE MC
4F47 C9       26450          RET                    ;     AND RETURN
4F48 DD219C49 26500 A4       LD      IX,D4          ;FIND 40 METER STATUS
4F4C CD744E   26510          CALL    SS             ;DISPLAY STATUS
4F4F FD215453 26520          LD      IY,MD          ;FIND MESSAGE MD
4F53 DD219A3E 26530          LD      IX,V+666       ;PRINT AT 666
4F57 CD644E   26540          CALL    PT             ;  MESSAGE MD
4F5A C9       26550          RET                    ;     AND RETURN
4F5B DD21CE49 26600 A8       LD      IX,D8          ;FIND 80 METER STATUS
4F5F CD744E   26610          CALL    SS             ;DISPLAY STATUS
4F62 FD215E53 26620          LD      IY,MF          ;FIND MESSAGE MF
4F66 DD219A3E 26630          LD      IX,V+666       ;PRINT AT 666
4F6A CD644E   26640          CALL    PT             ;  MESSAGE MF
4F6D C9       26650          RET                    ;     AND RETURN
4F6E 0605     27000 UP       LD      B,5            ;COUNT 5 COLUMNS
4F70 0E0A     27010          LD      C,10           ;COUNT 10 ROWS
4F72 FD21063C 27020          LD      IY,V+6         ;COLUMN VIDEO POINTER
4F76 DD21063C 27030          LD      IX,V+6         ;ROW VIDEO POINTER
4F7A DD222040 27040 FP       LD      (4020H),IX     ;SET CURSOR POSITION
4F7E DD36015D 27050          LD      (IX+1),5DH     ;DISPLAY CURSOR
4F82 CD4900   27060          CALL    049H           ;GET KEYBOARD INPUT
4F85 FE31     27070          CP      '1'            ;IS INPUT 1
4F87 2815     27080          JR      Z,F2           ;IF YES THEN 27190
4F89 FE32     27090          CP      '2'            ;IS INPUT 2
4F8B 2811     27100          JR      Z,F2           ;IF YES THEN 27190
4F8D FE33     27110          CP      '3'            ;IS INPUT 3
4F8F 280D     27120          JR      Z,F2           ;IF YES THEN 27190
4F91 FE42     27130          CP      'B'            ;IS INPUT B
4F93 282F     27140          JR      Z,BS           ;IF YES THEN 27370
4F95 FE58     27150          CP      'X'            ;IS INPUT X
4F97 20E1     27160          JR      NZ,FP          ;IF NO THEN 27040
4F99 DD360120 27170          LD      (IX+1),20H     ;ELSE ERASE OLD CURSOR
4F9D C9       27180          RET                    ;  AND RETURN
4F9E DD7700   27190 F2       LD      (IX),A         ;DISPLAY NEW STATUS
4FA1 DD360120 27200          LD      (IX+1),20H     ;ERASE OLD CURSOR
4FA5 77       27210          LD      (HL),A         ;STORE NEW STATUS
4FA6 0D       27220          DEC     C              ;ADJUST ROW COUNT
4FA7 2809     27230          JR      Z,F3           ;IF C=0 THEN 27280
4FA9 23       27240          INC     HL             ;ADJUST STATUS POINTER
```

Program continued

```
4FAA 114000     27250       LD      DE,64           ;MOVE 64 SPACES
4FAD DD19       27260       ADD     IX,DE           ;  TO NEXT LINE
4FAF C37A4F     27270       JP      FP              ;GOTO 27040
4FB2 05         27280 F3    DEC     B               ;ADJUST COLUMN COUNT
4FB3 285D       27290       JR      Z,F6            ;IF B=0 THEN 27780
4FB5 23         27300       INC     HL              ;ADJUST STATUS POINTER
4FB6 110C00     27310       LD      DE,12           ;MOVE CURSOR 12 SPACES
4FB9 FD19       27320       ADD     IY,DE           ;  TO NEXT COLUMN
4FBB FDE5       27330       PUSH    IY              ;TRANSFER IY TO IX
4FBD DDE1       27340       POP     IX              ;  FOR COLUMN POINTER
4FBF 0E0A       27350       LD      C,10            ;RESET ROW COUNT
4FC1 C37A4F     27360       JP      FP              ;GOTO 27040
4FC4 DD360120   27370 BS    LD      (IX+1),20H      ;ERASE OLD CURSOR
4FC8 79         27380       LD      A,C             ;PUT C IN A TO CHECK
4FC9 FE0A       27390       CP      10              ;    ROW COUNT
4FCB 2812       27400       JR      Z,F4            ;IF C=0 THEN 27520
4FCD 113F00     27410       LD      DE,63           ;ELSE PREPARE TO
4FD0 E5         27420 F7    PUSH    HL              ;  BACKSPACE CURSOR
4FD1 DDE5       27430       PUSH    IX              ;  BY 1 ROW
4FD3 E1         27440       POP     HL              ;   USING HL
4FD4 ED52       27450       SBC     HL,DE           ;    TO SUBTRACT
4FD6 E5         27460       PUSH    HL              ;TRANSFER HL TO IX
4FD7 DDE1       27470       POP     IX              ;  FOR POINTER
4FD9 E1         27480       POP     HL              ;RESTORE HL
4FDA 0C         27490       INC     C               ;ADJUST ROW COUNT +1
4FDB 2B         27500       DEC     HL              ;ADJUST MEMORY POINTER
4FDC C37A4F     27510       JP      FP              ;GOTO 27040
4FDF 78         27520 F4    LD      A,B             ;PUT COLUMN COUNT IN A
4FE0 FE05       27530       CP      5               ;CHECK COLUMN COUNT
4FE2 2819       27540       JR      Z,F5            ;IF 0 THEN 27690
4FE4 113402     27550       LD      DE,564          ;MOVE CURSOR BACK 564
4FE7 DD19       27560       ADD     IX,DE           ;  TO PREVIOUS COLUMN
4FE9 0E01       27570       LD      C,1             ;RESET ROW COUNT
4FEB 04         27580       INC     B               ;ADJUST COLUMN COUNT
4FEC 110C00     27590       LD      DE,12           ;ADJUST COULMN POINTER
4FEF E5         27600       PUSH    HL              ;SAVE HL
4FF0 FDE5       27610       PUSH    IY              ;PUT IY
4FF2 E1         27620       POP     HL              ; INTO HL
4FF3 ED52       27630       SBC     HL,DE           ;  IY MINUS DE
4FF5 E5         27640       PUSH    HL              ;PUT HL BACK
4FF6 FDE1       27650       POP     IY              ; INTO IY
4FF8 E1         27660       POP     HL              ;RESTORE HL
4FF9 2B         27670       DEC     HL              ;ADJUST MEMORY POINTER
4FFA C37A4F     27680       JP      FP              ;GOTO 27040
4FFD 117002     27690 F5    LD      DE,624          ;MOVE CURSOR TO BOTTOM
5000 DD19       27700       ADD     IX,DE           ;  OF NEXT COLUMN
5002 113000     27710       LD      DE,48           ;MOVE COLUMN POINTER
5005 FD19       27720       ADD     IY,DE           ;  48 SPACES
5007 0E01       27730       LD      C,1             ;RESET ROW COUNT
5009 0601       27740       LD      B,1             ;RESET COLUMN COUNT
500B 113100     27750       LD      DE,49           ;MOVE MEMORY POINTER
500E 19         27760       ADD     HL,DE           ;  49 SPACES
500F C37A4F     27770       JP      FP              ;GOTO 27040
5012 DD36015D   27780 F6    LD      (IX+1),5DH      ;DISPLAY CURSOR
5016 CD4900     27790       CALL    049H            ;GET KEYBOARD INPUT
5019 FE58       27800       CP      'X'             ;IS INPUT X
501B 2005       27810       JR      NZ,CB           ;IF YES THEN 27840
501D DD360120   27820       LD      (IX+1),20H      ;ELSE ERASE OLD CURSOR
5021 C9         27830       RET                     ;  AND RETURN
5022 FE42       27840 CB    CP      'B'             ;IS INPUT B
5024 20EC       27850       JR      NZ,F6           ;IF NO - BAD INPUT
5026 0601       27860       LD      B,1             ;ELSE RESET COLUMN COUNT
5028 0E01       27870       LD      C,1             ;RESET ROW COUNT
502A 114000     27880       LD      DE,64           ;RESET ROW OFFSET
502D DD360120   27890       LD      (IX+1),20H      ;ERASE OLD CURSOR
5031 C3D04F     27900       JP      F7              ;GOTO 27420
5034 2A         35000 M1    DEFM    '** MENU **////1.START NEW I
NDEX/2.INPUT DATA TAPE///3.REVIEW SPECIFIC BAND/4.RECORD DATA TA
PE
5091 2F         35010       DEFM    '///5.REVIEW SPECIFIC STATE/
```

```
6.REVIEW ALL DATA///7.REVIEW MIXED BAND W.A.S./8.LIST NEEDED STA
TES///
5105 20        35020          DEFM     '      9.END SESSION////////
                ENTER NUMBER OF FUNCTION YOU DESIRE!
5153 49        35030 M2       DEFM     'INSERT DATA TAPE - DEPRESS
PLAY!
5173 57        35040 M3       DEFM     'WHICH BAND--(10,15,20,40,80
)!
5190 31        35050 MA       DEFM     '10 METERS!
519A 49        35060 M4       DEFM     'INSERT DATA TAPE -- DEPRESS
 PLAY/RECORD!
51C2 57        35070 M5       DEFM     'WHICH STATE DO YOU WISH TO
REVIEW!
51E4 4D        35080 M6       DEFM     'MIXED BAND!
51EF 4C        35090 M7       DEFM     'LIST NEEDED STATES -!
5204 31        35100 MB       DEFM     '15 METERS!
520E 49        35110 M8       DEFM     'IF YOU FORGOT TO RECORD DAT
A TAPE PRESS X TO RETURN TO MENU.    ANY OTHER KEY RETURNS TO
BASIC.!
5270 43        35120 M9       DEFM     'CT-ME-MA-NH-RI-NT-NJ-NY-DE-
MD-PA-AL-FL-GA-KY-NC-SC-TN-VA-AR-LA-MS-NM-OK-TX-CA-AZ-ID-MT-NV-O
R-UT-WA-WY-MI-OH-WV-
52DF 49        35130          DEFM     'IL-IN-WI-CO-IA-KS-MN-MO-NB-
ND-SD-HA-AK!
5306 53        35140 MG       DEFM     'STATE      10       15       20
    40       80!
532F 58        35150 MJ       DEFM     'X = EXIT / B = MOVE CURSOR!

534A 32        35160 MC       DEFM     '20 METERS!
5354 34        35170 MD       DEFM     '40 METERS!
535E 38        35180 MF       DEFM     '80 METERS!
0000           35200          END
```

Personal Property Inventory

by Robert James Lloyd

Returning home from a two week vacation can be as hectic as preparing for it. First, the long drive; then, picking up Rover at Aunt Mary's, stopping at the grocery store; and finally, arriving home. The last thing anyone would like to discover is a burglarized home.

While devastating to the individual, police consider burglary routine. Normally, for simple robberies, fingerprints aren't even taken. Actually, what is more helpful is a list of items stolen, giving as much information as possible. Naturally, serial numbers are extremely useful in recovering missing property.

Those maintaining homes, whether rented or owned, have property which should be documented. A minimum inventory includes credit card, driver's license, insurance policy, and savings bond numbers, as well as home furnishings having serial numbers. Your automobile's vehicle identification number is also a must.

If, like me, you haven't bothered to keep a record of your property, read on. The following paragraphs describe a program which helps in maintaining a Personal Property Inventory (PPI). A cassette version (Program Listing 1) is explained first, followed by the description of a disk version (Program Listing 2). Both are menu driven to simplify their use.

Cassette Version

Minimum requirements for the cassette version are a 16K Level II Model I TRS-80. While not absolutely necessary, hard-copy output is desirable. I find it best to use twenty-minute cassettes when recording files, and always make a backup copy.

PPI is divided into 13 sections, which are described below.

● Beginning variables: All variables and strings used by PPI are explained in Table 1. Note string PT$ prints a masthead via BASIC's string graphics feature whenever the menu is displayed. Character codes 194, 213, and 237 used by PT$ are space compression codes (SCCs), which print multiple blanks in lieu of STRING$. SCCs reduce the amount of memory needed for string storage.

● Menu display: Placing a restaurant order usually requires a menu of food items. Similarly, PPI makes use of a menu, allowing "orders" to be given. Lines 90 through 120 create a menu by reading data statements and storing them in M$(RD). Displaying the menu is handled by lines 130 through 160.

A flashing cursor indicates INKEY$ is in use. Simply press the letter that corresponds to the desired option. Only displayed options are valid. If an incorrect order is given, PPI ignores it and gives the user another try.

	INTEGER VARIABLES
A	Number of file items
C	Cursor location
F	Input/search loop
G	Search loop
H	Add/delete item number
L & U	Formats screen display of file
M	Input error flag
N	Page number for hard copy
P	VARPTR loop (add/delete)
Q	Screen display/hard-copy flag
S	Screen location for arrows (displayed during add/delete)
FS	Set to file size + 1
ED	Display error messages loop
RD	Read/display data (menu)
PL	Screen location when inputting items
TD	Time delay loop
A1-A2	PEEK/POKE (VARPTR)
B1-B2	PEEK/POKE (VARPTR)
C1-C2	PEEK/POKE (VARPTR)
D1-D2	PEEK/POKE (VARPTR)
LM*	Length of manufacturer input/2
LD*	Length of description input/2
LS*	Length of serial # input/2
	*LM-LD-LS—Centers input under headings
	DIMENSIONED ARRAYS
A$	Manufacturer
B$	Description
C$	Serial number
M$	Menu data
D	Value
	STRINGS
I$	INKEY$
K$	File date
Y$	PRINTUSING format
AS$	Manufacturer search
CS$	Serial number search
PT$	PPI masthead (menu)
	DISK VERSION
FILESPEC$	Sets file name to be used when writing file to disk

Table 1. *PPI variables, strings, and arrays*

● Creating a file: PPI includes four fields for each item within a file. They are: manufacturer (10), description (16), serial number (15), and value (9999.99). Numbers in parentheses show the maximum length, or value, of each field. Abbreviations may be necessary when inputting data. Should more information be entered than any one field can handle, an error message momentarily flashes. Check which field was entered incorrectly, and reenter the entire item.

All fields need not be used. For example, when inputting a charge card, value is not required; however, it may be used to show the applicable credit limit. Refer to the sample run in Figure 1.

After all items have been entered, type CLOSE to the manufacturer prompt. This closes the file and returns the menu. Should the maximum file size (50 items) be reached before CLOSE is entered, a message indicating FILE FULL flashes. The file automatically closes when this occurs.

● Displaying a file: Video display of a file is accomplished by selecting menu option B. It is formatted to the screen as shown by the sample run, except only ten items list at one time. Pressing C continues the display, again, ten at a time. During a screen listing, you can return to the menu by pressing M instead of C.

● Printing a file: As stated previously, hard-copy output is desirable when using PPI. File listings include the date and all items contained in the file, formatted to 8 1/2- by 11-inch paper. If more than one page is necessary, PPI automatically numbers each page. Whenever the value field equals zero, the program prints blanks for that field, whether displaying or printing a file.

● Recording a file: Insert a tape of proper length to hold your file, and set the cassette player to record. The program makes use of only one recorder, since everyone may not own an expansion interface. To change the record routine to access a second tape recorder, alter lines 460 and 470 from PRINT#-1 to PRINT#-2. As file information is recorded, item numbers are displayed. This acts to monitor the program so you can see how the recording is progressing.

● Inputting a file: As stated above, PPI uses only one recorder. Lines 500 through 510 determine from which recorder information is read. To use one recorder for input and one for output, change PRINT#s and INPUT#s as required. Inputting a file is similar to recording, in that item numbers are displayed as they are read. Once the file has been entered, the menu returns.

● Adding items: Sometimes, it may become necessary to add to a file. The program allows insertion of new items, either within a file or at the end. To add within, PPI first rearranges memory to make room for the new item. This is accomplished by VARPTR. Only one item may be inserted at a time. Adding items to a file's end merely continues the item numbering until CLOSE is entered or the maximum number of items is reached (50).

```
                        PERSONAL PROPERTY INVENTORY

      DATE: JANUARY 12 1981

ITEM #   MANUFACTURER      DESCRIPTION          SERIAL #          VALUE
   1       SYLVANIA        COLOR CON TV       B12 345 67 890      750.00
   2       RADIO SHK       TRS-80 COMPUTER       123456           849.00
   3       RADIO SHK        DISK DRIVE           098765           499.00
   4       RADIO SHK      LINE PRINTER III       1A123           1999.00
   5       UNISONIC          5 IN. TV            123 456           89.00
   6       RADIO SHK       TRS-80 MOD II         123 34 4        3500.00
   7       MSTR CHRG        CREDIT CARD       12345678901234
   8       PONTIAC          GRND PRIX         2J37AA3665939      9000.00
   9       BLUE CROSS     HOSPITALIZATION        511284
  10       LOCAL BANK       24 HR TELLER       0329668011
  11       LOCAL BANK     CHKNG ACCOUNT         3296 6896
  12       YOUR STATE      DRVR LICENSE          135791
```

Figure 1. *Sample run of PPI*

● Deleting items: Deletion is also done by the VARPTR command. To delete an item, enter its corresponding number. In a very short time, the unwanted item is purged.

● Editing a file: Occasionally, it becomes necessary to edit a file. Pressing H during the menu display allows you to change the file date or items. When editing an item, it is first shown as it is presently stored in the file. Enter the corrected item in the same format you used when you created the file. Editing the file date merely changes K$.

● Searching: File searches can be conducted by either manufacturer or serial number fields. Searches by description are not allowed, since PPI hunts for exact matches. The descriptions of the items do not remain constant as in the other fields. In other words, you might code a television as "Color cons. TV" one time and "Color TV" the next.

● Ending the program: Select J during the menu display to exit the program. You are then free to load other programs without powering down.

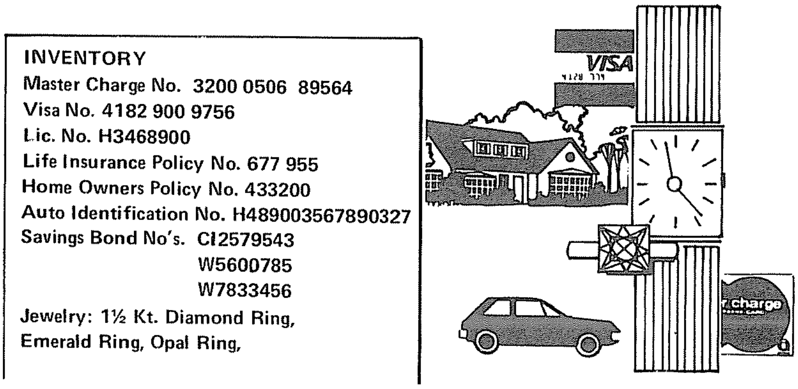
● Subroutines: Subroutines used by PPI are quite basic. They are used to check for input errors and carry out display/print formatting, file input, and time delay loops.

Disk Version

The disk version of PPI requires a 32K one-disk Model I system. Differences between the PPI cassette and disk programs have been kept to a minimum, so instructions are, for the most part, interchangeable.

The main differences are in the I/O routines. While INPUT (cassette) does not allow commas or other delimiters, LINE INPUT (disk) does. Therefore, you can input string data containing delimiters without seeing EXTRA IGNORED. Be sure to count delimiters when figuring the length of data being input to avoid INPUT ERROR messages.

● Output: Disk output is accomplished in three stages. First, the number of items (A) and file date (K$) are written. Next, string data (A$, B$, C$), and lastly, numerical data (D) are output. Data is written to disk sequentially, using one buffer. To see how this is done, create a file, save it, return to DOS, and use the LIST command to view the file.



INVENTORY
Master Charge No. 3200 0506 89564
Visa No. 4182 900 9756
Lic. No. H3468900
Life Insurance Policy No. 677 955
Home Owners Policy No. 433200
Auto Identification No. H489003567890327
Savings Bond No's. C12579543
W5600785
W7833456
Jewelry: 1½ Kt. Diamond Ring,
Emerald Ring, Opal Ring,

One caution when saving files: Remember, if you see a file name already on disk, that file will be replaced by the one you are saving. To avoid this, do a directory listing of the diskette when first loaded, and make sure you do not repeat any file names. Also, be aware that files are written in compressed format unless the ASCII option is specified in the file name.

● Input: Inputting from disk is done in a similar fashion. First, the number of items and date are input, followed by the string data and numerical data.

How to Make Changes in the PPI Program

● File size: To increase allowable file size in either version (make sure enough memory is available), proceed as follows: Clear additional memory and redimension A$, B$, C$, and D in line 40 to three more than the new file size, and change FS (line 80) to one more.

● Sort routine: Some of you may want to add a sort routine to PPI. If so, you may wish to modify the ADD routine, since new items need only be added at the end of a file, then sorted. Delete lines 530 through 570 and lines 620 through 710, inclusive, in the cassette version. For the disk version, lines 710 through 750 and lines 800 through 890 can be deleted. Now, enter your new items, execute the sort routine you added, and the items are in their proper places.

Program Listing 1. *PPI cassette version*

```
 10 REM  * PERSONAL PROPERTY INVENTORY BY ROBERT JAMES LLOYD *
 20 REM  * CASSETTE VERSION *
 30 CLEAR 3050:
    POKE 16424,66:
    POKE 16425,0
 40 DEFINT A - C,E - Z:
    DIM A$(54),B$(54),C$(54),D(54),M$(10)
 50 PT$ = CHR$(213) + CHR$(160) + CHR$(190) + CHR$(194) + CHR$(191)
    + CHR$(131) + CHR$(191) + CHR$(128) + CHR$(191) + CHR$(131)
    + CHR$(191)
 60 PT$ = PT$ + CHR$(128) + CHR$(131) + CHR$(191) + CHR$(131)
    + CHR$(194) + CHR$(189) + CHR$(144) + CHR$(237) + CHR$(130)
    + CHR$(175) + CHR$(194) + CHR$(191) + CHR$(131) + CHR$(131)
 70 PT$ = PT$ + CHR$(128) + CHR$(191) + CHR$(131) + CHR$(131)
    + CHR$(128) + CHR$(176) + CHR$(191) + CHR$(176) + CHR$(194)
    + CHR$(159) + CHR$(129)
 80 S = 330:
    FS = 51:
    A = FS:
    Y$ = "####.##"
 90 FOR RD = 1 TO 10:
     READ M$(RD):
     NEXT RD
100 DATA <A> CREATE FILE,<B> DISPLAY FILE,<C> PRINT FILE
110 DATA <D> RECORD FILE,<E> INPUT FILE,<F> ADD ITEMS,<G> DELETE ITE
    M
120 DATA <H> EDIT FILE,<I> SEARCH FILE,<J> END PROGRAM
130 L = 10:
    N = 1:
    H = 0:
    Q = 0:
    CLS :
    PRINT :
    PRINT PT$
140 PRINT :
    PRINT TAB(17)"PERSONAL PROPERTY INVENTORY"
150 PRINT :
    PRINT TAB(29)"MENU"
160 FOR RD = 1 TO 5:
     PRINT TAB(11)M$(RD); TAB(35)M$(RD + 5):
     NEXT RD
170 PRINT @854,"SELECT OPTION ";:
    GOSUB 1500
180 IF ASC(I$) = 65 OR ASC(I$) = 69 OR ASC(I$) = 74 CLS :
     ELSE
      IF ASC(I$) < 65 OR ASC(I$) > 74 GOSUB 1410 :
      GOTO 180 :
       ELSE
        GOSUB 1150
190 CLS :
    ON ASC(I$) - 64 GOTO 200 ,260 ,420 ,450 ,480 ,520 ,720 ,840 ,980
     ,1120
200 IF A$(1) < > "" GOTO 1350
210 K$ = "":
    PRINT :
    PRINT "INPUT FILE DATE (DO NOT USE COMMAS)"
220 INPUT K$
230 FOR F = 1 TO FS:
     CLS :
     IF F = FS GOTO 1220
240  GOSUB 1420 :
     IF M = 1M = 0:
     GOTO 240
250  NEXT F
260 IF Q = 1 GOTO 280 :
     ELSE
      PRINT TAB(15)"PERSONAL PROPERTY INVENTORY"
```

Program continued

```
270 PRINT "DATE: ";K$:
    GOSUB 1250 :
    GOTO 310
280 LPRINT TAB(24)"PERSONAL PROPERTY INVENTORY"
290 LPRINT STRING$(3,10):
    LPRINT TAB(10)"DATE: ";K$
300 LPRINT STRING$(3,10):
    GOSUB 1250
310 FOR F = 1 TO A - 1:
     GOSUB 1270 :
     U = A - 1:
     IF Q = 0 GOTO 340 :
      ELSE
       PRINT @128,"ITEM #";F;" IS BEING PRINTED":
       IF PEEK(16425) < > 60 GOTO 370
320  LPRINT CHR$(11):
     N = N + 1:
     LPRINT TAB(5)"DATE: ";K$:
     LPRINT TAB(60)"PAGE: ";N:
     LPRINT
330  IF F = A - 1 GOTO 360 :
      ELSE
       GOSUB 1250 :
       GOTO 370
340  IF F = A - 1 GOTO 380
350  IF U - L < 0 GOTO 370
360  IF F / L = 1 GOSUB 400 :
     CLS :
     GOSUB 1250
370  NEXT F:
    IF Q = 1 GOTO 390
380 PRINT :
    GOTO 960
390 LPRINT CHR$(11):
    GOTO 130
400 PRINT "PRESS C TO CONTINUE LISTING--PRESS M TO RETURN TO MENU ";
    :
    GOSUB 1500
410 IF ASC(I$) = 67L = L + 10:
    RETURN :
     ELSE
      IF ASC(I$) = 77 GOTO 130 :
       ELSE
        GOSUB 1410 :
        GOTO 410
420 PRINT "PREPARE PRINTER":
    PRINT :
    PRINT "WHEN READY, PRESS P ";:
    GOSUB 1500
430 IF ASC(I$) = 80 CLS :
     ELSE
      GOSUB 1410 :
      GOTO 430
440 IF PEEK(14312) < > 63 PRINT "THE PRINTER IS NOT READY":
    GOTO 1230 :
     ELSE
      LPRINT STRING$(3,10):
      Q = 1:
      GOTO 260
450 GOSUB 1390 :
    CLS :
    PRINT "SAVING DATA"
460 PRINT # - 1,A,K$:
    FOR F = 1 TO A - 1:
     PRINT @128,"ITEM #";F;
470  PRINT " IS BEING SAVED":
     PRINT # - 1,A$(F),B$(F),C$(F),D(F):
     NEXT F:
    GOTO 130
480 IF LEN(A$(1)) > 0 GOTO 1350
```

```
490 CLS :
    GOSUB 1390 :
    CLS :
    PRINT "INPUTTING DATA"
500 INPUT # - 1,A,K$:
510 FOR F = 1 TO A - 1:
     PRINT @128,"ITEM #";F;" IS BEING INPUTTED":
     INPUT # - 1,A$(F),B$(F),C$(F),D(F):
     NEXT F:
    GOTO 130
520 IF A = FS GOTO 1220
530 PRINT "DO YOU WISH TO:":
    PRINT
540 PRINT "<W>  ADD ONE ITEM WITHIN THE FILE"
550 PRINT "<E>  ADD ITEMS AT END OF THE FILE"
560 PRINT :
    GOSUB 1490
570 IF ASC(I$) = 69 CLS :
     ELSE
      IF ASC(I$) = 87 CLS :
      GOTO 620 :
       ELSE
        GOSUB 1410 :
        GOTO 570
580 FOR F = A TO FS:
     CLS :
     IF F = FS
      THEN
       A = F:
       GOTO 1220
590  GOSUB 1420 :
     IF M = 1M = 0:
     GOTO 590
600  IF A$(F) = "CLOSE"A = F:
     GOTO 130
610  NEXT F
620 INPUT "AFTER WHICH ITEM DO YOU WANT TO ADD IT";H:
    GOSUB 1200 :
    GOSUB 1340
630 FOR F = A TO H + 1 STEP - 1:
     PRINT @S, CHR$(94):
     FOR P = 2 TO 0 STEP - 1:
      A1 = PEEK( VARPTR(A$(F)) + P):
      A2 = PEEK( VARPTR(A$(F + 1)) + P):
      B1 = PEEK( VARPTR(B$(F)) + P):
      B2 = PEEK( VARPTR(B$(F + 1)) + P)
640   C1 = PEEK( VARPTR(C$(F)) + P):
      C2 = PEEK( VARPTR(C$(F + 1)) + P)
650   POKE ( VARPTR(C$(F)) + P),C2:
      POKE ( VARPTR(C$(F + 1)) + P),C1:
      POKE ( VARPTR(B$(F)) + P),B2:
      POKE ( VARPTR(B$(F + 1)) + P),B1
660   POKE ( VARPTR(A$(F)) + P),A2:
      POKE ( VARPTR(A$(F + 1)) + P),A1:
      NEXT P
670  FOR P = 3 TO 0 STEP - 1:
      D1 = PEEK( VARPTR(D(F)) + P):
      D2 = PEEK( VARPTR(D(F + 1)) + P):
      POKE ( VARPTR(D(F)) + P),D2:
      POKE ( VARPTR(D(F + 1)) + P),D1
680   NEXT P:
     PRINT @S, CHR$(32):
     S = S + 1:
     IF S = 341S = 330
690  NEXT F
700 F = H + 1:
    GOSUB 1420 :
    IF M = 1M = 0:
    GOTO 700
```

Program continued

```
710 A = A + 1:
    IF A = FS GOTO 1220 :
     ELSE
      GOTO 1230
720 INPUT "WHICH ITEM IS TO BE DELETED";H:
    GOSUB 1200 :
    IF H = ØM = 1:
    GOSUB 1180 :
    M = Ø:
    CLS :
    GOTO 720
730 GOSUB 1340
740 FOR F = H TO A:
     PRINT @S, CHR$(94):
     FOR P = Ø TO 2:
      A1 = PEEK( VARPTR(A$(F)) + P):
      A2 = PEEK( VARPTR(A$(F + 1)) + P):
      B1 = PEEK( VARPTR(B$(F)) + P):
      B2 = PEEK( VARPTR(B$(F + 1)) + P)
750   C1 = PEEK( VARPTR(C$(F)) + P):
      C2 = PEEK( VARPTR(C$(F + 1)) + P)
760   POKE ( VARPTR(C$(F)) + P),C2:
      POKE ( VARPTR(C$(F + 1)) + P),C1:
      POKE ( VARPTR(B$(F)) + P),B2:
      POKE ( VARPTR(B$(F + 1)) + P),B1
770   POKE ( VARPTR(A$(F)) + P),A2:
      POKE ( VARPTR(A$(F + 1)) + P),A1
780   NEXT P
790  FOR P = Ø TO 3:
      D1 = PEEK( VARPTR(D(F)) + P):
      D2 = PEEK( VARPTR(D(F + 1)) + P):
      POKE ( VARPTR(D(F)) + P),D2:
      POKE ( VARPTR(D(F + 1)) + P),D1
800   NEXT P:
     PRINT @S, CHR$(32):
     S = S + 1:
     IF S = 341S = 33Ø
810  NEXT F
820 A = A - 1:
    PRINT :
    PRINT "ITEM #";H;" HAS BEEN DELETED.":
    IF A < 2A$(1) = "":
    K$ = "":
    A = FS
830 GOTO 123Ø
840 CLS :
    PRINT "DO YOU WISH TO EDIT:":
    PRINT
850 PRINT "<F>  FILE DATE"
860 PRINT "<I>  ITEM INFORMATION"
870 PRINT :
    GOSUB 149Ø
880 IF ASC(I$) = 7Ø CLS :
     ELSE
      IF ASC(I$) = 73 GOTO 91Ø :
       ELSE
        GOSUB 141Ø :
        GOTO 88Ø
890 CLS :
    INPUT "ENTER NEW FILE DATE";K$
900 PRINT "FILE DATE IS NOW ";K$:
    GOTO 123Ø
910 CLS :
    INPUT "ENTER ITEM NUMBER TO BE EDITTED";F:
    H = F:
    GOSUB 12ØØ
920 PRINT :
    PRINT "ITEM #";H;" APPEARS IN FILE AS FOLLOWS:":
    PRINT
```

```
930 GOSUB 1250 :
    GOSUB 1270 :
    FOR TD = 1 TO 1000:
     NEXT TD:
    CLS
940 GOSUB 1420 :
    IF M = 1M = 0:
    GOTO 940
950 PRINT "THE ITEM NOW APPEARS AS FOLLOWS:":
    PRINT :
    GOSUB 1250 :
    GOSUB 1270
960 PRINT :
    PRINT "PRESS M TO RETURN TO MENU ";:
    GOSUB 1500
970 IF ASC(I$) < > 77 GOSUB 1410 :
    GOTO 970 :
     ELSE
      130
980 AS$ = "ZZZ":
    CS$ = "XXX"
990 CLS :
    PRINT "SEARCHES MAY BE CONDUCTED BY THE FOLLOWING FIELDS---"
1000 PRINT :
     PRINT "WHICH FIELD DO YOU WISH TO CONDUCT THE SEARCH BY"
1010 PRINT :
     PRINT "<A>  MANUFACTURER  <B>  SERIAL NUMBER":
     PRINT :
     GOSUB 1490
1020 IF ASC(I$) < 65 OR ASC(I$) > 66 GOSUB 1410 :
     GOTO 1020
1030 PRINT :
     PRINT :
     PRINT "WHAT DO YOU WISH TO SEARCH FOR ";:
     ON ASC(I$) - 64 GOTO 1040 ,1050
1040 INPUT AS$:
     GOTO 1060
1050 INPUT CS$
1060 PRINT :
     PRINT "SEARCHING FILE.....":
     GOSUB 1240 :
     GOSUB 1240
1070 FOR F = 1 TO A - 1:
      IF AS$ = A$(F) OR CS$ = C$(F) CLS :
      GOSUB 1250 :
       ELSE
        NEXT F:
       GOTO 1090
1080 FOR G = F TO A - 1:
      IF AS$ = A$(G) OR CS$ = C$(G) GOSUB 1270 :
      NEXT G:
     GOTO 1100 :
      :
      ELSE
       NEXT G:
       GOTO 1100
1090 PRINT :
     PRINT "A MATCH CANNOT BE FOUND"
1100 PRINT :
     PRINT "PRESS S TO CONDUCT ANOTHER SEARCH--PRESS M TO RETURN TO M
     ENU ";:
     GOSUB 1500
1110 IF ASC(I$) = 83 GOTO 980 :
      ELSE
       IF ASC(I$) = 77 GOTO 130 :
        ELSE
         GOSUB 1410 :
         GOTO 1050
1120 PRINT "IF A FILE WAS CREATED, BE SURE TO PROPERLY MARK THE CASSE
     TTE"
```

Program continued

```
1130 PRINT "WITH THE CORRECT FILE ID CODE AND DATE FOR FUTURE REFEREN
     CE."
1140 PRINT STRING$(10,10):
     END
1150 IF A$(1) = "" OR A < 2 CLS :
      ELSE
       RETURN
1160 FOR ED = 1 TO 5:
      PRINT @400,"YOU DO NOT HAVE A FILE IN MEMORY":
      GOSUB 1240 :
      PRINT @400, CHR$(224):
      GOSUB 1240 :
      NEXT ED:
     GOTO 130
1170 IF LEN(A$(F)) > 10 OR LEN(B$(F)) > 16 OR LEN(C$(F)) > 15
     OR D(F) > 9999.99M = 1:
     A$(F) = "":
     B$(F) = "":
     C$(F) = "":
     D(F) = 0
1180 IF M = 1C = (256 * ( PEEK(16417) - 60) + PEEK(16416) - 2)
     + 66:
      ELSE
       RETURN
1190 FOR ED = 1 TO 5:
      PRINT @C,"INPUT ERROR.....RE-ENTER":
      GOSUB 1240 :
      PRINT @C, CHR$(216):
      GOSUB 1240 :
      NEXT ED:
     RETURN
1200 IF H > A - 1 FOR ED = 1 TO 5:
      PRINT @400,"THERE AREN'T ";H;" ITEMS IN THIS FILE":
       ELSE
        RETURN
1210  GOSUB 1240 :
      PRINT @400, CHR$(242):
      GOSUB 1240 :
      NEXT ED:
     H = 0:
     GOTO 130
1220 FOR ED = 1 TO 5:
      PRINT @408,"THIS FILE IS FULL":
      GOSUB 1240 :
      PRINT @408, CHR$(209):
      GOSUB 1240 :
      NEXT ED:
     GOTO 130
1230 FOR TD = 1 TO 500:
      NEXT TD:
     GOTO 130
1240 FOR TD = 1 TO 100:
      NEXT TD:
     RETURN
1250 IF Q = 0 PRINT "ITEM #"; TAB(8)"MANUFACTURER"; TAB(24)"DESCRIPTI
     ON"; TAB(44)"SERIAL #"; TAB(57)"VALUE":
     RETURN
1260 IF Q = 1 LPRINT TAB(5)"ITEM #"; TAB(13)"MANUFACTURER"; TAB(29)"D
     ESCRIPTION"; TAB(49)"SERIAL #"; TAB(62)"VALUE":
     RETURN
1270 LM = INT( LEN(A$(F)) / 2):
     LD = INT( LEN(B$(F)) / 2):
     LS = INT( LEN(C$(F)) / 2)
1280 IF Q = 0 PRINT TAB(3)F; TAB(14 - LM)A$(F); TAB(29 - LD)B$(F);
     TAB(47 - LS)C$(F);
1290 IF Q = 0 AND D(F) < > 0 PRINT TAB(56);:
     PRINT USING Y$;D(F):
     RETURN
1300 IF Q = 0 AND D(F) = 0 PRINT TAB(56);"":
     RETURN
```

```
1310 IF Q = 1 LPRINT TAB(7)F; TAB(19 - LM)A$(F); TAB(34 - LD)B$(F);
     TAB(52 - LS)C$(F);
1320 IF Q = 1 AND D(F) < > 0 LPRINT TAB(61);:
     LPRINT USING Y$;D(F):
     RETURN
1330 IF Q = 1 AND D(F) = 0 LPRINT TAB(61)"":
     RETURN
1340 PRINT :
     PRINT "MEMORY IS BEING RE-ARRANGED":
     RETURN
1350 PRINT "THERE IS A FILE ALREADY IN MEMORY":
     PRINT
1360 PRINT "<D>  DELETE FILE AND RE-RUN PROGRAM"
1370 PRINT "<M>  RETURN TO MENU":
     PRINT :
     GOSUB 1490
1380 IF ASC(I$) = 68 GOTO 30 :
      ELSE
       IF ASC(I$) = 77 GOTO 130 :
        ELSE
         GOSUB 1410 :
         GOTO 1380
1390 PRINT "PREPARE RECORDER":
     PRINT :
     PRINT "WHEN READY, PRESS R ";:
     GOSUB 1500
1400 IF ASC(I$) = 82 RETURN :
      ELSE
       GOSUB 1410 :
       GOTO 1400
1410 C = 256 * ( PEEK(16417) - 60) + PEEK(16416) - 2:
     GOTO 1510
1420 CLS :
     PRINT "(ITEM #";F;")":
     PRINT "MANUFACTURER";
1430 PL = 85:
     PRINT @PL,"";:
     INPUT A$(F)
1440 IF A$(F) = "CLOSE"A = F:
     GOTO 130
1450 PRINT :
     PRINT "DESCRIPTION";:
     PL = PL + 128:
     PRINT @PL,"";:
     INPUT B$(F)
1460 PRINT :
     PRINT "SERIAL #";:
     PL = PL + 128:
     PRINT @PL,"";:
     INPUT C$(F)
1470 PRINT :
     PRINT "VALUE";:
     PL = PL + 128:
     PRINT @PL,"";:
     INPUT D(F)
1480 GOSUB 1170 :
     RETURN
1490 PRINT "SELECT OPTION ";
1500 C = 256 * ( PEEK(16417) - 60) + PEEK(16416)
1510 PRINT @C, STRING$(2,143);:
     GOSUB 1240 :
     I$ = INKEY$:
     IF I$ = "" PRINT @C,"  ";:
     GOSUB 1240 :
     GOTO 1510
1520 PRINT @C," ";I$;:
     FOR TD = 1 TO 50:
      NEXT TD:
     IF ASC(I$) < 65 GOTO 1510 :
      ELSE
       RETURN
```

Program continued

Program Listing 2. *PPI disk version*

```
 10 REM   * PERSONAL PROPERTY INVENTORY BY ROBERT JAMES LLOYD *
 20 REM   * DISK VERSION *
 30 CLEAR 3050:
    POKE 16424,66:
    POKE 16425,0
 40 DEFINT A - C,E - Z:
    DIM A$(54),B$(54),C$(54),D(54),M$(10)
 50 PT$ = CHR$(213) + CHR$(160) + CHR$(190) + CHR$(194) + CHR$(191)
    + CHR$(131) + CHR$(191) + CHR$(128) + CHR$(191) + CHR$(131)
    + CHR$(191)
 60 PT$ = PT$ + CHR$(128) + CHR$(131) + CHR$(191) + CHR$(131)
    + CHR$(194) + CHR$(189) + CHR$(144) + CHR$(237) + CHR$(130)
    + CHR$(175) + CHR$(194) + CHR$(191) + CHR$(131) + CHR$(131)
 70 PT$ = PT$ + CHR$(128) + CHR$(191) + CHR$(131) + CHR$(131)
    + CHR$(128) + CHR$(176) + CHR$(191) + CHR$(176) + CHR$(194)
    + CHR$(159) + CHR$(129)
 80 S = 330:
    FS = 51:
    A = FS:
    Y$ = "####.##"
 90 FOR RD = 1 TO 10:
     READ M$(RD):
     NEXT RD
100 DATA <A> CREATE FILE,<B> DISPLAY FILE,<C> PRINT FILE
110 DATA <D> SAVE FILE,<E> INPUT FILE,<F> ADD ITEMS,<G> DELETE ITEM
120 DATA <H> EDIT FILE,<I> SEARCH FILE,<J> END PROGRAM
130 L = 10:
    N = 1:
    H = 0:
    Q = 0:
    CLS :
    PRINT :
    PRINT PT$
140 PRINT :
    PRINT TAB(17)"PERSONAL PROPERTY INVENTORY"
150 PRINT :
    PRINT TAB(29)"MENU"
160 FOR RD = 1 TO 5:
     PRINT TAB(11)M$(RD); TAB(35)M$(RD + 5):
     NEXT RD
170 PRINT @854,"SELECT OPTION ";:
    GOSUB 1720
180 IF ASC(I$) = 65 OR ASC(I$) = 69 OR ASC(I$) = 74 CLS :
     ELSE
      IF ASC(I$) < 65 OR ASC(I$) > 74 GOSUB 1590 :
      GOTO 180 :
       ELSE
        GOSUB 1330
190 CLS :
    ON ASC(I$) - 64 GOTO 200 ,260 ,420 ,450 ,580 ,700 ,900 ,1020 ,11
    60 ,1300
200 IF A$(1) < > "" GOTO 1530
210 K$ = "":
    PRINT :
    PRINT "INPUT FILE DATE"
220 PRINT :
    LINE INPUT K$
230 FOR F = 1 TO FS:
     CLS :
     IF F = FS GOTO 1400
240  GOSUB 1600 :
     IF M = 1M = 0:
     GOTO 240
250  NEXT F
260 IF Q = 1 GOTO 280 :
     ELSE
      PRINT TAB(15)"PERSONAL PROPERTY INVENTORY"
```

```
270 PRINT "DATE: ";K$:
    GOSUB 1430 :
    GOTO 310
280 LPRINT TAB(24)"PERSONAL PROPERTY INVENTORY"
290 LPRINT STRING$(3,10):
    LPRINT TAB(10)"DATE: ";K$
300 LPRINT STRING$(3,10):
    GOSUB 1430
310 FOR F = 1 TO A - 1:
     GOSUB 1450 :
     U = A - 1:
     IF Q = 0 GOTO 340 :
      ELSE
       PRINT @128,"ITEM #";F;" IS BEING PRINTED":
       IF PEEK(16425) < > 60 GOTO 370
320  LPRINT CHR$(11):
     N = N + 1:
     LPRINT TAB(5)"DATE: ";K$:
     LPRINT TAB(60)"PAGE: ";N:
     LPRINT
330  IF F = A - 1 GOTO 360 :
      ELSE
       GOSUB 1430 :
       GOTO 370
340  IF F = A - 1 GOTO 380
350  IF U - L < 0 GOTO 370
360  IF F / L = 1 GOSUB 400 :
     CLS :
     GOSUB 1430
370  NEXT F:
    IF Q = 1 GOTO 390
380 PRINT :
    GOTO 1140
390 LPRINT CHR$(11):
    GOTO 130
400 PRINT "PRESS C TO CONTINUE LISTING--PRESS M TO RETURN TO MENU ";
    :
    GOSUB 1720
410 IF ASC(I$) = 67L = L + 10:
    RETURN :
     ELSE
      IF ASC(I$) = 77 GOTO 130 :
       ELSE
        GOSUB 1590 :
        GOTO 410
420 PRINT "PREPARE PRINTER":
    PRINT :
    PRINT "WHEN READY, PRESS P ";:
    GOSUB 1720
430 IF ASC(I$) = 80 CLS :
     ELSE
      GOSUB 1590 :
      GOTO 430
440 IF PEEK(14312) < > 63 PRINT "THE PRINTER IS NOT READY":
    GOTO 1410 :
     ELSE
      LPRINT STRING$(3,10):
      Q = 1:
      GOTO 260
450 GOSUB 1570 :
    CLS
460 PRINT "ENTER FILE NAME (EXAMPLE: PERSFILE/TXT:1)"
470 PRINT :
    PRINT "IF A DISK DRIVE IS NOT SPECIFIED, THE FIRST"
480 PRINT "NON-WRITE PROTECTED DRIVE WILL BE USED."
490 PRINT :
    PRINT "? ";
500 LINE INPUT FILESPEC$
510 OPEN "O",1,FILESPEC$
```

Program continued

```
520 CLS :
    PRINT "SAVING DATA"
530 PRINT #1,A
540 PRINT #1, CHR$(34);K$; CHR$(34)
550 FOR F = 1 TO A - 1:
     PRINT #1, CHR$(34);A$(F); CHR$(34); CHR$(34);B$(F); CHR$(34);
     CHR$(34);C$(F); CHR$(34):
     NEXT F
560 FOR F = 1 TO A - 1:
     PRINT #1,D(F):
     NEXT F
570 CLOSE 1:
    GOTO 130
580 IF LEN(A$(1)) > 0 GOTO 1530
590 CLS :
    GOSUB 1570
600 CLS :
    PRINT "ENTER FILE NAME"
610 PRINT :
    PRINT "? ";
620 LINE INPUT FILESPEC$
630 CLS :
    PRINT "INPUTTING DATA"
640 OPEN "I",1,FILESPEC$
650 INPUT #1,A
660 INPUT #1,K$
670 FOR F = 1 TO A - 1:
     INPUT #1,A$(F),B$(F),C$(F):
     NEXT F
680 FOR F = 1 TO A - 1:
     INPUT #1,D(F):
     NEXT F
690 CLOSE 1:
    GOTO 130
700 IF A = FS GOTO 1400
710 PRINT "DO YOU WISH TO:":
    PRINT
720 PRINT "<W>  ADD ONE ITEM WITHIN THE FILE"
730 PRINT "<E>  ADD ITEMS AT END OF THE FILE"
740 PRINT :
    GOSUB 1710
750 IF ASC(I$) = 69 CLS :
     ELSE
      IF ASC(I$) = 87 CLS :
      GOTO 800 :
       ELSE
        GOSUB 1590 :
        GOTO 750
760 FOR F = A TO 51:
     CLS :
     IF F = 51A = F:
     GOTO 1400
770  GOSUB 1600 :
     IF M = 1M = 0:
     GOTO 770
780  IF A$(F) = "CLOSE"A = F:
     GOTO 130
790  NEXT F
800 INPUT "AFTER WHICH ITEM DO YOU WANT TO ADD IT";H:
    GOSUB 1380 :
    GOSUB 1520
810 FOR F = A TO H + 1 STEP - 1:
     PRINT @S, CHR$(94):
     FOR P = 2 TO 0 STEP - 1:
      A1 = PEEK( VARPTR(A$(F)) + P):
      A2 = PEEK( VARPTR(A$(F + 1)) + P):
      B1 = PEEK( VARPTR(B$(F)) + P):
      B2 = PEEK( VARPTR(B$(F + 1)) + P)
820   C1 = PEEK( VARPTR(C$(F)) + P):
      C2 = PEEK( VARPTR(C$(F + 1)) + P)
```

```
 830    POKE ( VARPTR(C$(F)) + P),C2:
        POKE ( VARPTR(C$(F + 1)) + P),C1:
        POKE ( VARPTR(B$(F)) + P),B2:
        POKE ( VARPTR(B$(F + 1)) + P),B1
 840    POKE ( VARPTR(A$(F)) + P),A2:
        POKE ( VARPTR(A$(F + 1)) + P),A1:
        NEXT P
 850   FOR P = 3 TO Ø STEP - 1:
        D1 = PEEK( VARPTR(D(F)) + P):
        D2 = PEEK( VARPTR(D(F + 1)) + P):
        POKE ( VARPTR(D(F)) + P),D2:
        POKE ( VARPTR(D(F + 1)) + P),D1
 860    NEXT P:
       PRINT @S, CHR$(32):
       S = S + 1:
       IF S = 341S = 330
 870   NEXT F
 880  F = H + 1:
      GOSUB 1600 :
      IF M = 1M = Ø:
      GOTO 880
 890  A = A + 1:
      IF A = FS GOTO 1400 :
       ELSE
        GOTO 1410
 900  INPUT "WHICH ITEM IS TO BE DELETED";H:
      GOSUB 1380 :
      IF H = ØM = 1:
      GOSUB 1360 :
      M = Ø:
      CLS :
      GOTO 900
 910  GOSUB 1520
 920  FOR F = H TO A:
       PRINT @S, CHR$(94):
       FOR P = Ø TO 2:
        A1 = PEEK( VARPTR(A$(F)) + P):
        A2 = PEEK( VARPTR(A$(F + 1)) + P):
        B1 = PEEK( VARPTR(B$(F)) + P):
        B2 = PEEK( VARPTR(B$(F + 1)) + P)
 930    C1 = PEEK( VARPTR(C$(F)) + P):
        C2 = PEEK( VARPTR(C$(F + 1)) + P)
 940    POKE ( VARPTR(C$(F)) + P),C2:
        POKE ( VARPTR(C$(F + 1)) + P),C1:
        POKE ( VARPTR(B$(F)) + P),B2:
        POKE ( VARPTR(B$(F + 1)) + P),B1
 950    POKE ( VARPTR(A$(F)) + P),A2:
        POKE ( VARPTR(A$(F + 1)) + P),A1
 960    NEXT P
 970   FOR P = Ø TO 3:
        D1 = PEEK( VARPTR(D(F)) + P):
        D2 = PEEK( VARPTR(D(F + 1)) + P):
        POKE ( VARPTR(D(F)) + P),D2:
        POKE ( VARPTR(D(F + 1)) + P),D1
 980    NEXT P:
       PRINT @S, CHR$(32):
       S = S + 1:
       IF S = 341S = 330
 990   NEXT F
1000  A = A - 1:
      PRINT :
      PRINT "ITEM #";H;" HAS BEEN DELETED.":
      IF A < 2A$(1) = "":
      K$ = "":
      A = FS
1010  GOTO 1410
1020  CLS :
      PRINT "DO YOU WISH TO EDIT:":
      PRINT
```

Program continued

```
1030 PRINT "<F>  FILE DATE"
1040 PRINT "<I>  ITEM INFORMATION"
1050 PRINT :
     GOSUB 1710
1060 IF ASC(I$) = 70 CLS :
      ELSE
       IF ASC(I$) = 73 GOTO 1090 :
        ELSE
         GOSUB 1590 :
         GOTO 1060
1070 CLS :
     LINE INPUT "ENTER NEW FILE DATE";K$
1080 PRINT "FILE DATE IS NOW ";K$:
     GOTO 1410
1090 CLS :
     INPUT "ENTER ITEM NUMBER TO BE EDITTED";F:
     H = F:
     GOSUB 1380
1100 PRINT :
     PRINT "ITEM #";H;" APPEARS IN FILE AS FOLLOWS:":
     PRINT
1110 GOSUB 1430 :
     GOSUB 1450 :
     FOR TD = 1 TO 1000:
      NEXT TD:
     CLS
1120 GOSUB 1600 :
     IF M = 1M = 0:
     GOTO 1120
1130 PRINT "THE ITEM NOW APPEARS AS FOLLOWS:":
     PRINT :
     GOSUB 1430 :
     GOSUB 1450
1140 PRINT :
     PRINT "PRESS M TO RETURN TO MENU ";:
     GOSUB 1720
1150 IF ASC(I$) < > 77 GOSUB 1590 :
     GOTO 1150 :
      ELSE
       130
1160 AS$ = "ZZZ":
     CS$ = "XXX"
1170 CLS :
     PRINT "SEARCHES MAY BE CONDUCTED BY THE FOLLOWING FIELDS---"
1180 PRINT :
     PRINT "WHICH FIELD DO YOU WISH TO CONDUCT THE SEARCH BY"
1190 PRINT :
     PRINT "<A>  MANUFACTURER  <B>  SERIAL NUMBER":
     PRINT :
     GOSUB 1710
1200 IF ASC(I$) < 65 OR ASC(I$) > 66 GOSUB 1590 :
     GOTO 1200
1210 PRINT :
     PRINT :
     PRINT "WHAT DO YOU WISH TO SEARCH FOR ";:
     ON ASC(I$) - 64 GOTO 1220 ,1230
1220 INPUT AS$:
     GOTO 1240
1230 INPUT CS$
1240 PRINT :
     PRINT "SEARCHING FILE.....":
     GOSUB 1420 :
     GOSUB 1420
1250 FOR F = 1 TO A - 1:
      IF AS$ = A$(F) OR CS$ = C$(F) CLS :
      GOSUB 1430 :
       ELSE
       NEXT F:
      GOTO 1270
```

```
1260 FOR G = F TO A - 1:
      IF AS$ = A$(G) OR CS$ = C$(G) GOSUB 1450 :
      NEXT G:
     GOTO 1280 :
     :
      ELSE
       NEXT G:
       GOTO 1280
1270 PRINT :
     PRINT "A MATCH CANNOT BE FOUND"
1280 PRINT :
     PRINT "PRESS S TO CONDUCT ANOTHER SEARCH--PRESS M TO RETURN TO M
     ENU ";:
     GOSUB 1720
1290 IF ASC(I$) = 83 GOTO 1160 :
      ELSE
       IF ASC(I$) = 77 GOTO 130 :
        ELSE
         GOSUB 1590 :
         GOTO 1230
1300 PRINT "IF A FILE WAS SAVED, BE SURE TO PROPERLY MARK THE DISKETT
     E"
1310 PRINT "WITH THE CORRECT FILESPEC AND DATE FOR FUTURE REFERENCE."
1320 PRINT STRING$(10,10):
     END
1330 IF A$(1) = "" OR A < 2 CLS :
      ELSE
       RETURN
1340 FOR ED = 1 TO 5:
      PRINT @400,"YOU DO NOT HAVE A FILE IN MEMORY":
      GOSUB 1420 :
      PRINT @400, CHR$(224):
      GOSUB 1420 :
      NEXT ED:
     GOTO 130
1350 IF LEN(A$(F)) > 10 OR LEN(B$(F)) > 16 OR LEN(C$(F)) > 15
     OR D(F) > 9999.99M = 1:
     A$(F) = "":
     B$(F) = "":
     C$(F) = "":
     D(F) = 0
1360 IF M = 1C = (256 * ( PEEK(16417) - 60) + PEEK(16416) - 2)
     + 66:
      ELSE
       RETURN
1370 FOR ED = 1 TO 5:
      PRINT @C,"INPUT ERROR.....RE-ENTER":
      GOSUB 1420 :
      PRINT @C, CHR$(216):
      GOSUB 1420 :
      NEXT ED:
     RETURN
1380 IF H > A - 1 FOR ED = 1 TO 5:
      PRINT @400,"THERE AREN'T ";H;" ITEMS IN THIS FILE":
       ELSE
        RETURN
1390  GOSUB 1420 :
      PRINT @400, CHR$(242):
      GOSUB 1420 :
      NEXT ED:
     H = 0:
     GOTO 130
1400 FOR ED = 1 TO 5:
      PRINT @408,"THIS FILE IS FULL":
      GOSUB 1420 :
      PRINT @408, CHR$(209):
      GOSUB 1420 :
      NEXT ED:
     GOTO 130
```

Program continued

```
1410 FOR TD = 1 TO 500:
      NEXT TD:
     GOTO 130
1420 FOR TD = 1 TO 100:
      NEXT TD:
     RETURN
1430 IF Q = 0 PRINT "ITEM #"; TAB(8)"MANUFACTURER"; TAB(24)"DESCRIPTI
     ON"; TAB(44)"SERIAL #"; TAB(57)"VALUE":
     RETURN
1440 IF Q = 1 LPRINT TAB(5)"ITEM #"; TAB(13)"MANUFACTURER"; TAB(29)"D
     ESCRIPTION"; TAB(49)"SERIAL #"; TAB(62)"VALUE":
     RETURN
1450 LM = INT( LEN(A$(F)) / 2):
     LD = INT( LEN(B$(F)) / 2):
     LS = INT( LEN(C$(F)) / 2)
1460 IF Q = 0 PRINT TAB(3)F; TAB(14 - LM)A$(F); TAB(29 - LD)B$(F);
     TAB(47 - LS)C$(F);
1470 IF Q = 0 AND D(F) < > 0 PRINT TAB(56);:
     PRINT USING Y$;D(F):
     RETURN
1480 IF Q = 0 AND D(F) = 0 PRINT TAB(56);"":
     RETURN
1490 IF Q = 1 LPRINT TAB(7)F; TAB(19 - LM)A$(F); TAB(34 - LD)B$(F);
     TAB(52 - LS)C$(F);
1500 IF Q = 1 AND D(F) < > 0 LPRINT TAB(61);:
     LPRINT USING Y$;D(F):
     RETURN
1510 IF Q = 1 AND D(F) = 0 LPRINT TAB(61)"":
     RETURN
1520 PRINT :
     PRINT "MEMORY IS BEING RE-ARRANGED":
     RETURN
1530 PRINT "THERE IS A FILE ALREADY IN MEMORY":
     PRINT
1540 PRINT "<D>  DELETE FILE AND RE-RUN PROGRAM"
1550 PRINT "<M>  RETURN TO MENU":
     PRINT :
     GOSUB 1710
1560 IF ASC(I$) = 68 GOTO 30 :
      ELSE
       IF ASC(I$) = 77 GOTO 130 :
        ELSE
         GOSUB 1590 :
         GOTO 1560
1570 PRINT "PLACE DISKETTE IN DRIVE TO BE USED":
     PRINT :
     PRINT "WHEN READY, PRESS R ";:
     GOSUB 1720
1580 IF ASC(I$) = 82 RETURN :
      ELSE
       GOSUB 1590 :
       GOTO 1580
1590 C = 256 * ( PEEK(16417) - 60) + PEEK(16416) - 2:
     GOTO 1730
1600 CLS :
     PRINT "(ITEM #";F;")":
     PRINT "MANUFACTURER";
1610 PL = 85:
     PRINT @PL,"? ";
1620 LINE INPUT A$(F)
1630 IF A$(F) = "CLOSE"A = F:
     GOTO 130
1640 PRINT :
     PRINT "DESCRIPTION";:
     PL = PL + 128:
     PRINT @PL,"? ";
1650 LINE INPUT B$(F)
1660 PRINT :
     PRINT "SERIAL #";:
     PL = PL + 128:
```

```
     PRINT @PL,"? ";
1670 LINE INPUT C$(F)
1680 PRINT :
     PRINT "VALUE";:
     PL = PL + 128:
     PRINT @PL,"";
1690 INPUT D(F)
1700 GOSUB 1350 :
     RETURN
1710 PRINT "SELECT OPTION ";
1720 C = 256 * ( PEEK(16417) - 60) + PEEK(16416)
1730 PRINT @C, STRING$(2,143);:
     GOSUB 1420 :
     I$ = INKEY$:
     IF I$ = "" PRINT @C,"  ";:
     GOSUB 1420 :
     GOTO 1730
1740 PRINT @C," ";I$;:
     FOR TD = 1 TO 50:
      NEXT TD:
     IF ASC(I$) < 65 GOTO 1730 :
      ELSE
       RETURN
```

INTERFACE

Testing 1,2,3

Testing 1, 2, 3

by D. C. Nelson

Have you ever wanted to use your computer as part of a measurement system? Soon after getting a computer, I began to wish that I could use it for recording and analyzing measurements from test instruments. The thought of my computer functioning as a data logging tool was enticing, but the thought of having to manually key in the data was not.

Parallel Ports

Fortunately, there is a better way. I am referring to the parallel port, that neglected feature commonly used for little more than reading keyboards or driving printers. You can create both the hardware and software necessary to link your computer to outside measuring instruments without having to learn assembly language, since everything can be done using BASIC.

Let's look at some of the requirements for linking test equipment to a computer. First, the data must be in a computer-recognizable format and at TTL compatible levels. However, this does not mean that we are restricted to using only binary numbers. Binary code makes the most efficient use of space for a given number of bits, but it is not always the easiest code to use. By redefining the meaning of each bit position in an eight-bit word, we can make the code easier to work with in terms of the outside world. This is called binary coded decimal, or BCD for short.

BCD Data

A single eight-bit word can express any binary value from 00000000 to 11111111, or from 0 to 255 decimal. In a single-decimal digit, it takes four bits to specify any of the possible values from 0 to 9. The remaining binary combinations from 11 to 15 are discarded. Since the computer operates on eight-bit data, it is only logical to use the remaining four bit positions to represent another digit. Thus an eight-bit word can be used to represent any number between 0 and 99 in BCD format. Unlike a straight binary format,

BINARY	BCD	DECIMAL
01010010	10000010	82
01001111	01111001	79
00000011	00000011	3
10000000	ILLEGAL	128

01010110 BCD = 56 DECIMAL

Example 1. *Binary, BCD, and decimal equivalents. One byte can represent any two-digit number.*

we now have some bit patterns that are illegal. Example 1 shows how this process works.

Why go to BCD? A quick look through an IC catalog will give part of the answer. Many common components such as counters, A/D converters, and clock chips have BCD outputs already available. Many commercially available measuring instruments have auxiliary outputs that are BCD coded. In addition, it is easier to tailor your computer to the number of digits you wish to resolve by using BCD.

BCD also has some disadvantages. First, there is the need for a method of converting between BCD and binary. Second, to have a large number of BCD interfaces, you must also have a large number of interconnections. Remember that each digit requires four bits, therefore four leads will be needed. For a six-digit frequency counter, this means 24 bulky connections. (Multiplexing can cut this down, but that gets more complicated.)

Hardware

Let's look at what it takes to get parallel data in and out of our computer. The TRS-80 has an edge connector on the back left that provides access to the CPU bus inside. An explanation of the pinout can be found in the back of the Level I manual. Oddly enough, this diagram was omitted from the Level II manual in the system I worked on. You will need Level II BASIC to make both the hardware and software function properly.

The parts of the bus that interest us are the address lines, the data lines, and the two control lines labelled $\overline{\text{IN}}$ and $\overline{\text{OUT}}$. The Z-80 CPU can address up to 256 I/O ports. When this is done in the TRS-80, the CPU places the port address (0-255) on the lower eight address lines and issues the appropriate $\overline{\text{IN}}$ or $\overline{\text{OUT}}$ control command at the rear connector.

If a word is being written to the port, the data bus will contain the byte being output, and the $\overline{\text{OUT}}$ line will be strobed low. If an input byte is requested, the data bus is tri-stated to allow the input port hardware to place its byte on the bus, and the $\overline{\text{IN}}$ line is strobed low. Whether a specific port number is used for input or output depends upon the design of the external hardware. It can be both.

The first requirement in setting up a parallel port is for the computer to recognize its port address. This means we must decode a particular address. At the same time, we must check to see if either the $\overline{\text{IN}}$ or $\overline{\text{OUT}}$ line is valid. These control lines tell us that the address is a port and not a memory.

Figure 1 shows an example of how this works. First, we must pick an address that is not already in use. Let's use 31 decimal as an example. The address bus has the appropriate lines inverted to make the inputs to the eight-

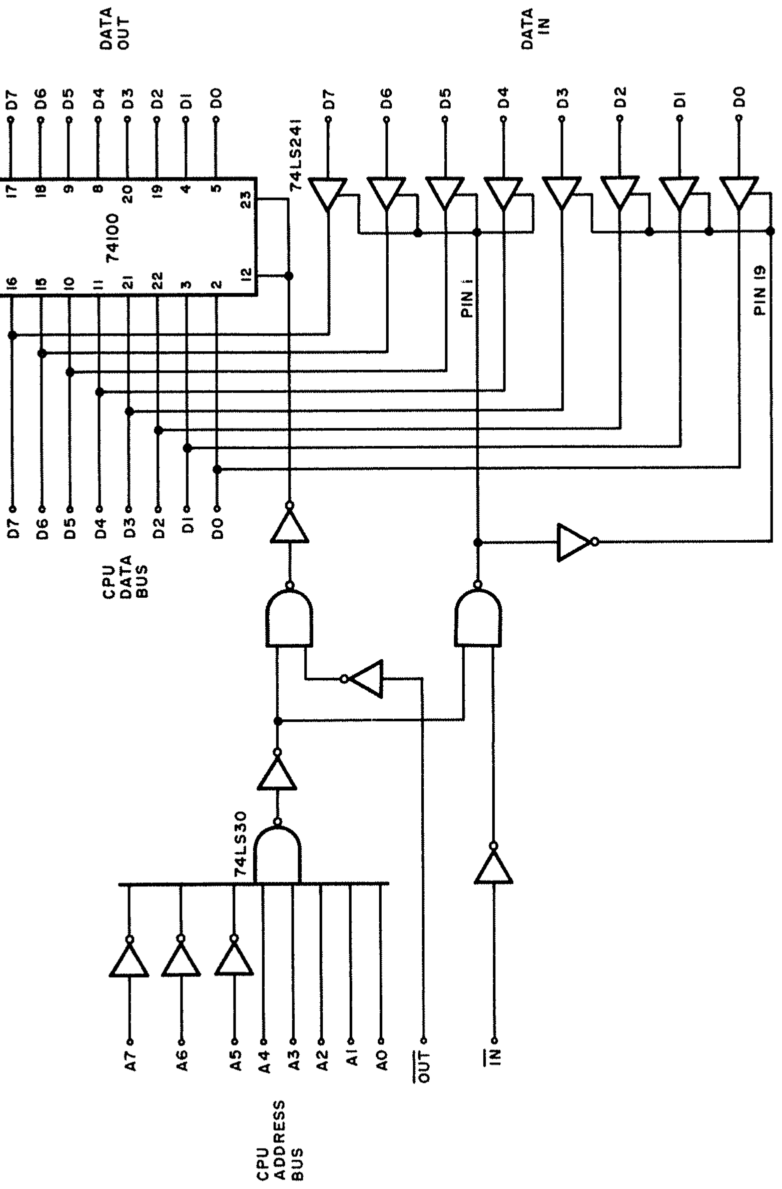


Figure 1. *One parallel I/O port. This example shows address 31 decimal (01F Hex) decoded.*

input NAND gate high when address 31 decimal is given. This signal is inverted and gated with the inverted $\overline{\text{OUT}}$ command to form a strobe pulse for the latch.

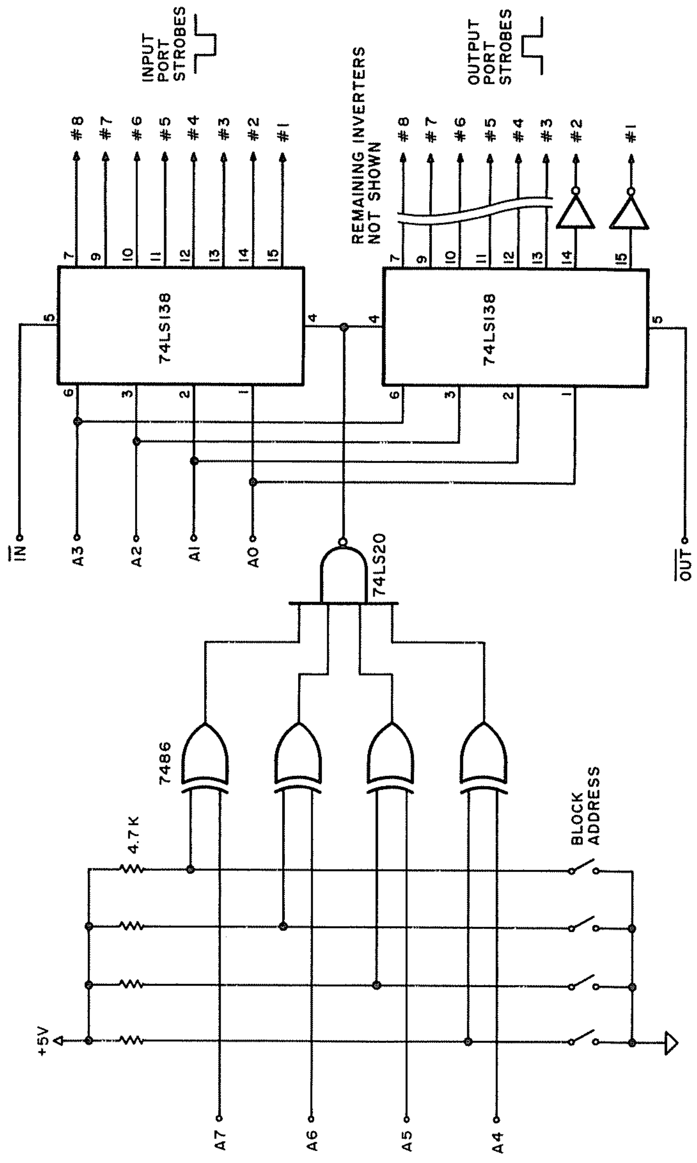


Figure 2. *Decoding eight parallel I/O ports. Input and output circuits are the same as Figure 1.*

For an input port, we decode the address in the same way, except that we use the $\overline{\text{IN}}$ line. Instead of a latch, we must now use a tri-state buffer to place the input byte on the data bus. Note that it is possible for an input and an output port to have the same numerical address. The control lines tell which of the two is to be active.

Multiple Ports

Obviously, a single I/O port will not suffice for all measurement applications. In order to address at least four—and preferably, eight—ports, the address is decoded differently.

First, we will say that all port addresses will be contiguous, or numerically sequential. The lowest address will be the boundary of the block. Since eight ports will require three bits for the address within the block, this leaves five bits to define the block boundary. Addressing a specific port now requires three events to be true: the proper boundary as defined by A3–A7, the proper three-bit address as defined by A0–A2, and the $\overline{\text{IN}}$ or $\overline{\text{OUT}}$ command as appropriate. Figure 2 illustrates this. (There are other ways of implementing parallel I/O ports. This method is shown because of its simplicity and the low cost of the parts.)

If a full eight ports are placed on the bus, it should be buffered, because there is the possibility of overloading the bus with all the additional decoding and latching circuitry. Also, if you locate your interface at the end of several feet of cable, you can introduce error-causing reflections onto the bus. As a final thought, consider the consequences if something goes haywire externally; you end up damaging some ICs, and your connections are straight onto the bus. Need I say more?

I built my buffer using tri-state gates on a small card that plugged straight into the edge connector. The buffer, in turn, had its own edge connector where the cable from the expansion interface was attached. I retained full use of the extra memory and the disk drive. Circuit speed posed no problem. The $\overline{\text{IN}}$ and $\overline{\text{OUT}}$ signals are 1.4 microseconds wide, more than adequate settling time.

Incidentally, the manual showed one of the pins on the edge connector providing +5 volts. Check before you use it, as mine turned out to be a ground. Because of the power draw of an eight-port interface, plan on building a separate power supply.

Check your work by instructing the processor to write a specific byte to a port. Verify this with any instrument, from an LED to an oscilloscope. At the same time, make sure that none of the other output ports are affected. To check an input port, selectively ground each bit, and verify that the binary number returned changes by the appropriate power of two.

Software

Now that the hardware has been resolved, let's look at the necessary software. We will make use of some special commands found in Level II BASIC. The first are INP(X) and OUT(X,Y). The statement A = INP(8) instructs the computer to read input port #8 and equate the value there to the variable A. (Note that at this time A can be from 0 to 255 decimal.) To write to a port, tell the computer to OUT 8,Y and the value of Y in binary (not BCD) will appear on port #8. In the examples I am describing, we cannot handle floating-point numbers.

Now we need the ability to selectively examine specific bits for deciphering the incoming BCD data. Earlier I said that one byte, and therefore, one port, can represent two BCD digits. Since the computer uses binary, and we are using BCD, some translating will be necessary. Assume we are using port #8 and that it contains the BCD representation of the number 95. Our program must read the port and return the number 95 in binary to the processor.

The first step reads the port and places the number in the variable N. Next we mask off, or zero, the four most significant bits, leaving only the units digit. This is directly equated to the variable N1 for temporary storage. Then the four least significant bits of N are masked, and the remaining bits are shifted four places to the right. This number is directly equated to N2 for storage. An artificial binary number is now created from each half of the BCD byte. Since the four least significant bits equate directly to 0–9 (not counting the illegals), the interpreted digit must lie in the four least significant bit positions. Anything else will give an erroneous result. The final step is to multiply N2 by 10, add it to N1, and return the number as N. If this process is made into a subroutine, any portion of the program can get a two-digit number from port #8. Example 2 shows the mechanics and the coding in more detail. Try it longhand to convince yourself that it works.

```
1000 REM BCD SUBROUTINE
1010 N = INP(8)              N = 10010101 = 95 BCD
1020 N1 = N AND 15           N1 = 00000101 = 5
1030 N2 = N AND 240          N2 = 10010000 = 90
1040 N2 = N2/16              N2 = 00001001 = 9
1050 N = 10 * N2 + N1        N = 10 * 9 + 5 = 95
1060 RETURN
```

Example 2. *Subroutine to read and decode two BCD digits. Bit masking and shifting are employed.*

Masking

The masking process is the key step in the conversion. Using the AND command makes it possible to selectively turn off any combination of bits in a word and leave the remainder unchanged.

Think of each bit in the input byte as one input to an AND gate and the other gate input as being the corresponding bit position in the mask word. If the mask bit is high, the gate output equals the second input bit. If the mask bit is a zero, then the gate output is a zero, regardless of the state of the other bit. This command will act on all eight bits at once. To determine the numerical value of the mask word, add the binary weight of the bit positions you want left unchanged.

Following the masking process (which leaves the tens digit), the shift right by four is accomplished through dividing by 16. Since dividing by two is the same as shifting every bit position right by one, it follows that a division by 16 will produce four shifts to the right. The code in Example 2 is shown one step at a time for clarity. To obtain more than two digits of resolution (one part in a hundred is inadequate in many cases), use some of the additional ports to give whatever amount is necessary.

Each port is treated as in Example 2, except that the decoded numbers are scaled by the appropriate power of ten before being added. Remember when I said how easy it was to scale the system for any number of digits? All you have to do is to allocate sufficient ports and do the appropriate decimal scaling on the numbers. Example 3 shows a subroutine for reading a six-digit frequency counter connected to three input ports.

```
1000 REM 6-DIGIT BCD READ
1010 N1 = INP(8)
1020 N2 = INP(9)
1030 N3 = INP(10)
1040 N1 = N1 AND 15 + (N1 AND 240)/16 * 10
1050 N2 = N2 AND 15 + (N2 AND 240)/16 * 10
1060 N3 = N3 AND 15 + (N3 AND 240)/16 * 10
1070 N = N1 + 100*N2 + 10000*N3
1080 RETURN
```

Example 3. *6-digit BCD read subroutine*

Output ports can be used to control relays, triacs, D/A converters, etc. One port can be used to control eight separate circuits by means of TTL compatible reed relays or optical isolators. The OR command controls any specific bit without affecting the others. To establish how fast I could transfer values in and out, I tried a couple of benchmark programs. In each case the program transferred 1000 values to a single port in a FOR-NEXT loop while doing nothing else. The output was finished in 6.4 seconds, while the input took 7.3 seconds. Using the subroutine in Example 3 took 115 seconds.

Obviously the calculations take their toll in speed. Consider, however, the application you will be involved with, and if your experience echoes mine, the comparatively low speed is of little matter. What is important is that the computer is now doing more work and the programmer less! Since it is almost always necessary for the computer to wait idly while the external instrument makes its measurement, a do-nothing FOR-NEXT loop can be used as a timer for such occasions. I have found that 340 iterations take almost one second.

Examples

Let's look at a circuit that uses the methods I have described. Figure 3 shows an analog-to-digital converter using the Motorola MC14433 IC. This chip is the basic building block for a 3 1/2-digit DVM. Note that most of the parts are used to convert the multiplexed BCD output to parallel, latched BCD, which is compatible with the software described. This is a typical requirement for most instruments. In this application the multiplexing works against us. It is possible to directly read a multiplexed output, but it requires an intimate knowledge of the timing of both the instrument and the computer.

To do this, the computer must test the digit position it needs to read. When that digit is enabled, it must be read before the scan continues. All digits must be read in one scan, so that an erroneous reading will not be returned. This would occur if the measuring instrument updated its reading to a new value while the computer was still trying to read the remainder of the original one.

Taking two readings and comparing them before returning to the calling program is a worthwhile check, if BASIC reads the data.

The circuit shown in Figure 3 illustrates a typical requirement for latching multiplexed data. If you intend to build it or need more information on this particular device, consult the Motorola CMOS manual.

You can also call a machine-language program from BASIC to demultiplex the data in software and POKE the digit pairs into successive memory locations. This eliminates the demultiplexing hardware. The subroutine in BASIC would then be modified to PEEK the memory locations as a source of data. Since the final program will depend on the specific instrument being interfaced, I will leave the rest to you.

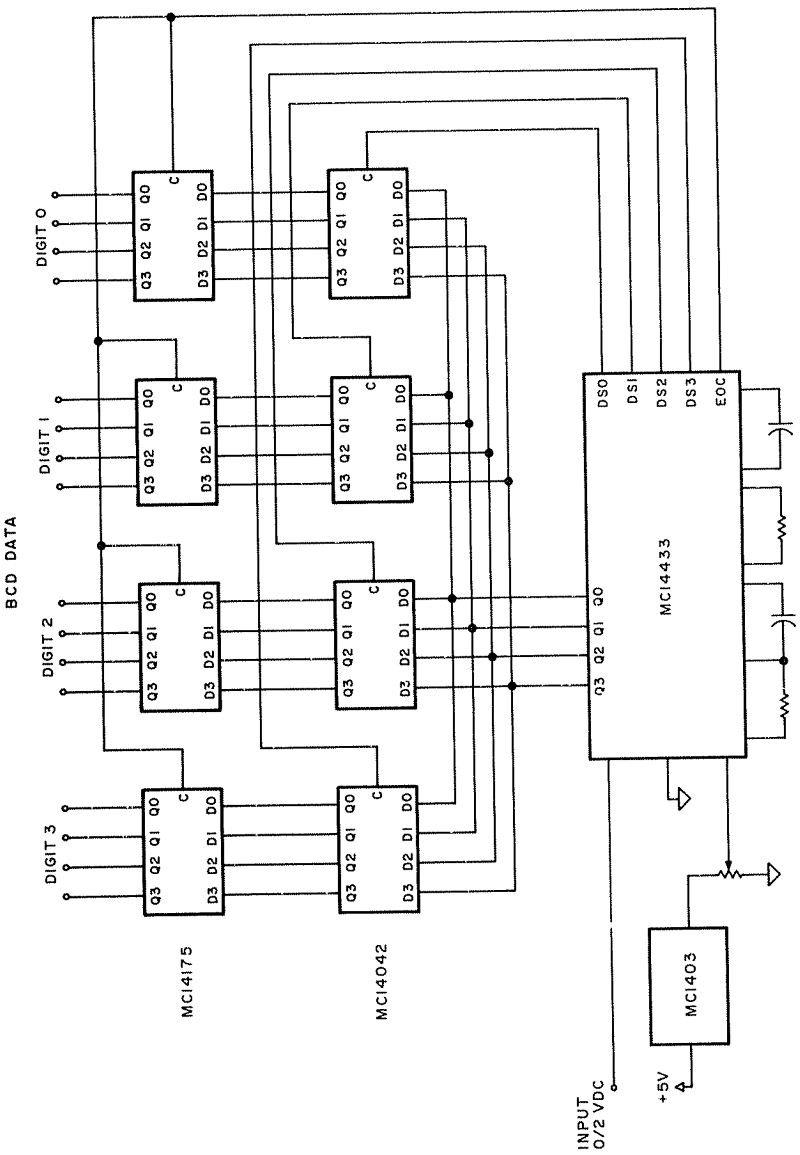


Figure 3. Interfacing a DVM chip. Multiplexed data is in latched, parallel format.

LANGUAGE

Decode CW Directly from the Cassette Earplug

Decode CW Directly from the Cassette Earplug

by Louis C. Graue

Would you like to have your computer copy Morse code directly off the radio without having to buy an expensive interface or having to build and apply some extra hardware? The cassette audio input is designed to process an audio signal. All we need is a program to transform CW audio, entered through the earplug, into text.

An Experiment

Go to your computer and run this experiment. This will help you to understand what to expect when audio is applied to the earplug. First enter the following program:

```
10 OUT 255,0:A = INP(255)
20 IF A = 127 PRINT "-";
30 IF A = 255 PRINT "*";
40 GOTO 10
```

Next, put a tape in the recorder, remove the mike, remote and auxiliary plugs, leave the earplug in, push play and record buttons, whistle dah, dit, dah, dit, and observe the CRT screen. You will see the following:

```
        ------------------------------------------------------------------
---------------*********---***---*********---***---------------
```

The stars appear as long as any sound is produced near the cassette mike, and the hyphens appear when there is silence. If you place your radio speaker near the mike while code is being broadcast, you will see the dots, dashes, and proper spacing on your screen. You have just programmed the simplest possible Morse code decoder. It can be used to determine how perfect the sending station is. Count the stars and hyphens to see the following: a) if dashes contain three times as many stars as do the dots, b) if the element spaces are composed of the same number of hyphens as there are stars in the dots, c) if there are character spaces three times as long as dots, and d) if word spaces are seven times as long as the dots. By placing your code practice oscillator near the mike, you can check out how well you send in the same way.

Now try inputting various tones while the above program is running. You will notice that low tones will not get through because the cassette audio input circuit in the keyboard contains an active high pass filter, with about a 2-kHz rolloff. This is used to filter out undesired noise and hum produced by the recorder. Thus, to copy a CW station, keep the output tone as high as possible.

A Basic Decoder

Enough experimenting. We want to produce text from those dots and dashes. We need a program which will make the translation. A simple program in BASIC will do the job for speeds up to about 20 wpm. For high speed work we will need a machine-language program. Program Listing 1 is a BASIC language program which is a modified version of one presented by Robert L. Kurtz in *Kilobaud*, November 1978, page 34. The steps taken are: wait for key down, measure down time (if less than ½T, a dot is stored; otherwise, a dash is stored), measure key up time (if end of character time, then print it; if end of word time, then print a space; otherwise get next code element).

For speeds from about 13 wpm to 20 wpm, set T = 6. For slower speeds, make T larger until you receive good copy. T can be changed quickly by hitting the BREAK key. Enter RUN and a prompt will ask for T. One value of T will cover a range of speeds, and I have found that it works better than a self-adaptive speed selection procedure. With T = 6 I get solid copy of the W1AW 18 wpm bulletin. I copied it on the recorder by placing it near the speaker of my radio and then played it back later on the computer. By changing the PRINT statements to LPRINT, you could get hard copy on your printer. You could even go first class and get an adaptor plug for the phone jack of the radio so that you can just plug the cassette ear plug in directly. However, I find it most rewarding to be able to both hear the CW and see the printout at the same time.

Another Experiment

What about those hams who are sending very fast? How do we find out what they are talking about? BASIC is too slow, so it's back to the drawing board and more experimenting, this time with machine language. Use T-BUG and enter Program Listing 2. This will do the exact same job as the first experimental program, except much faster. Do all the same things you did before. Whistle dah, dit, dah, dit. Play CW over the radio near the mike, or key your code practice oscillator. You will notice that the shortest dit will make a bunch of stars and a dah produces several lines of stars. This very fast action makes it possible to measure those 100 wpm dots and determine the difference between them and 100 wpm dashes. This extra sensitivity will cause a couple of quick lines of garbage to appear on the screen as a reaction to a static crash. The BASIC program in Program Listing 1 would produce only a couple of extra characters because of its slow action. Experimenting with this program will show you what to expect and why you get the outputs you will experience in the next program. The first six lines of this program (Program Listing 2) become the "check the input" routine for an assembly-language program to copy CW.

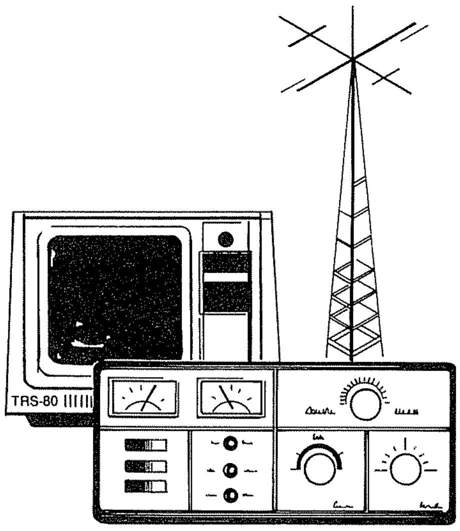
A Fast Decoder

Program Listing 3 is an assembly-language program with self-adaptive speed adjustment. It will copy at any speed with no attention from you. Just create an object program tape with your editor-assembler, enter the program, and hit the "/" key followed by RETURN to run the program. I have included detailed flowcharts for the main program and subroutines. I used the program published by Tim Ahrens in the January 1980 issue of *73 Magazine*, which was written in 6800 language, as a guide in writing this one.

Conclusion

If you have a computer and a good radio receiver, you can now copy Morse code on your terminal without making any attachments. Using the BASIC program is not difficult. If you want to copy faster than about 20 wpm, you will have to do a little more work and use the machine-language program.

Perhaps someone will come up with a super program that will turn our computer into a complete CW/RTTY station without the necessity of adding any hardware, except, of course, a transceiver.



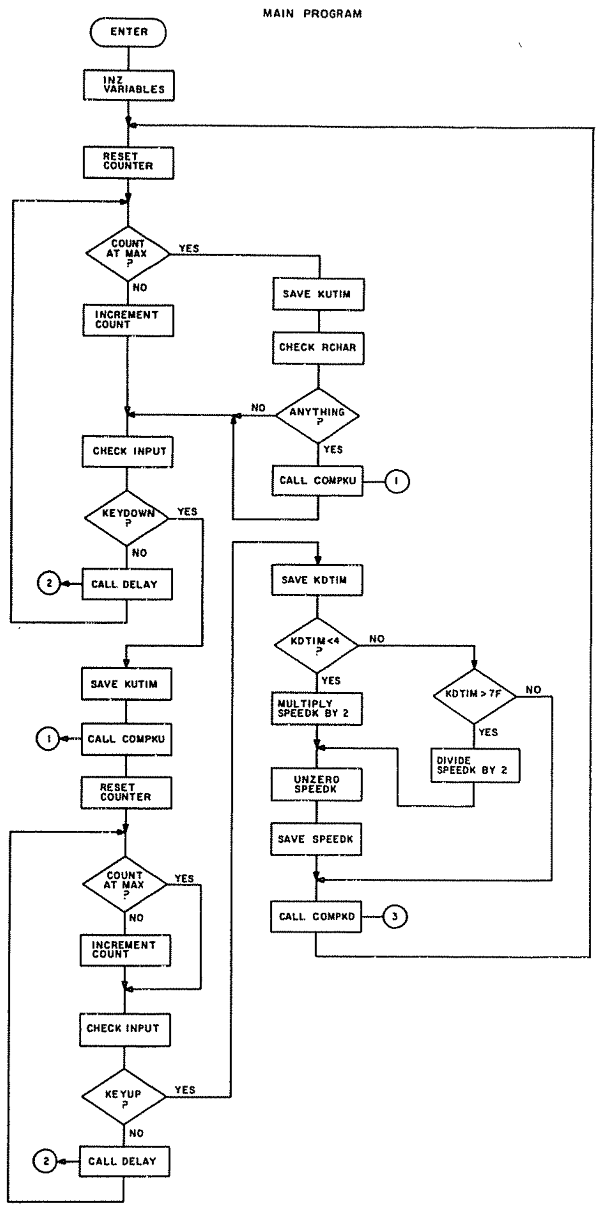

MAIN PROGRAM
ENTER
INZ VARIABLES
RESET COUNTER
COUNT AT MAX ?
YES
NO
INCREMENT COUNT
SAVE KUTIM
CHECK RCHAR
ANYTHING ?
NO
YES
CALL COMPKU
1
CHECK INPUT
KEYDOWN ?
YES
NO
CALL DELAY
2
SAVE KUTIM
CALL COMPKU
1
RESET COUNTER
COUNT AT MAX ?
YES
NO
INCREMENT COUNT
CHECK INPUT
KEYUP ?
YES
NO
CALL DELAY
2
SAVE KDTIM
KDTIM<4 ?
NO
YES
MULTIPLY SPEEDK BY 2
KDTIM > 7F
NO
YES
DIVIDE SPEEDK BY 2
UNZERO SPEEDK
SAVE SPEEDK
CALL COMPKD
3

Figure 1

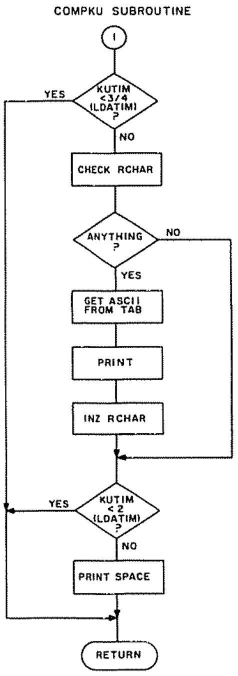


Figure 2

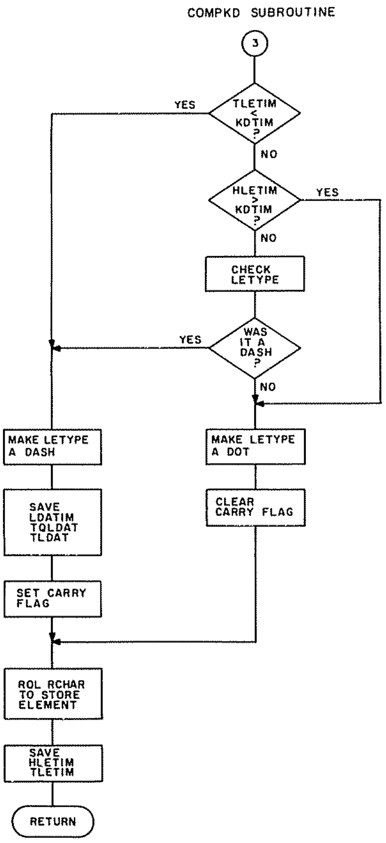


Figure 3

Program Listing 1

```
 10 :
    ' CASSETTE EAR PLUG CW READER - BY K8TT (6/15/80)
 20 CLS
 30 PRINT "HIGH FREQUENCY TONE COPIES BEST."
 40 DIM A$(100)
 50 FOR N = 1 TO 100:
     READ A$(N):
     NEXT N
 60 INPUT "DASH TIME(6 FOR 13 TO 20 WPM, HIGHER FOR SLOW)";T
 70 OUT 255,0:
    A = INP(255)
 80 IF A = 127
     THEN
      70
 90 B = 0
100 OUT 255,0:
    A = INP(255):
    B = B + 1
110 IF A = 127
     THEN
      DO = 2 * DO:
      DA = 2 * DA:
      DO = DO + 1:
      GOTO 160
120 IF B < .5 * T
     THEN
      100
130 DO = 2 * DO:
    DA = 2 * DA:
    DA = DA + 1
140 OUT 255,0:
    A = INP(255)
150 IF A = 255
     THEN
      140
160 B = 0
170 OUT 255,0:
    A = INP(255):
    B = B + 1
180 IF A = 255
     THEN
      90
190 IF B < .5 * T
     THEN
      170
200 GOSUB 260
210 OUT 255,0:
    A = INP(255):
    B = B + 1
220 IF A = 255
     THEN
      90
230 IF B < 2 * T
     THEN
      210
240 PRINT " ";
250 GOTO 70
260 DA = DA * 2:
    D = DA + DO:
    IF D > 100
     THEN
      D = 100
270 PRINT A$(D);
```

```
280 DA = Ø:
    DO = Ø:
    RETURN
290 DATA E,T,I,A,N,M,S,U,R,W,D,K,G,O,H,V,F,-,L,-,P,J,B,X,C
300 DATA Y,Z,Q,-,-,5,4,-,3,-,-,-,2,-,-,-,-,-,-,-,1,6,-,/,-
310 DATA -,-,-,-,7,-,-,-,8,-,9,Ø,-,-,-,-,-,-,-,-,-,-,-,-,?
320 DATA -,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-
```

Program Listing 2

```
4CØØ          ØØ1ØØ          ORG     4CØØH
4CØØ AF       ØØ11Ø START    XOR     A                ;ZERO A
4CØ1 D3FF     ØØ12Ø          OUT     (ØFFH),A         ;RESET
4CØ3 Ø6FF     ØØ13Ø          LD      B,ØFFH           ;DELAY FOR HARDWARE
4CØ5 1ØFE     ØØ14Ø LOOP     DJNZ    LOOP             ;      TO SETTLE
4CØ7 DBFF     ØØ15Ø          IN      A,(ØFFH)         ;CHECK INPUT
4CØ9 CB7F     ØØ16Ø          BIT     7,A              ;      AT BIT 7
4CØB 28Ø7     ØØ17Ø          JR      Z,SPACE          ;GO IF NO INPUT
4CØD 3E2A     ØØ18Ø          LD      A,2AH            ;ASCII FOR *
4CØF CD33ØØ   ØØ19Ø          CALL    33H              ;PRINT *
4C12 18EC     ØØ2ØØ          JR      START
4C14 3E2D     ØØ21Ø SPACE    LD      A,2DH            ;ASCII FOR -
4C16 CD33ØØ   ØØ22Ø          CALL    33H              ;PRINT -
4C19 18E5     ØØ23Ø          JR      START
ØØØØ          ØØ24Ø          END
ØØØØØ TOTAL ERRORS
```

Program Listing 3

```
              ØØ1ØØ ;DECODE CW THROUGH CASSETTE EAR PLUG (6/2Ø/8Ø)
              ØØ11Ø
4AØØ          ØØ12Ø          ORG     4AØØH
              ØØ13Ø ;TEMPORARY STORAGE FOR VARIABLES
6AØØ          ØØ14Ø LETYPE   EQU     6AØØH            ;LAST ELEMENT TYPE
4AØ1          ØØ15Ø HLETIM   EQU     4AØ1H            ;HALF LAST EL TIME
4AØ2          ØØ16Ø TLETIM   EQU     4AØ2H            ;TWICE LAST EL TIME
4AØ3          ØØ17Ø SPEEDK   EQU     4AØ3H            ;SPEED CONSTANT
4AØ4          ØØ18Ø RCHAR    EQU     4AØ4H            ;RCVD CHARACTER
4AØ5          ØØ19Ø LDATIM   EQU     4AØ5H            ;LAST DAH TIME
4AØ6          ØØ2ØØ TQLDAT   EQU     4AØ6H            ;3/4 LAST DAH TIME
4AØ7          ØØ21Ø TLDAT    EQU     4AØ7H            ;TWICE LAST DA TIME
4AØ8          ØØ22Ø KDTIM    EQU     4AØ8H            ;KEY DOWN TIME
4AØ9          ØØ23Ø KUTIM    EQU     4AØ9H            ;KEY UP TIME
              ØØ24Ø
              ØØ25Ø ;LOOKUP TABLE
4A2Ø          ØØ26Ø          ORG     4A2ØH
4A2Ø 21       ØØ27Ø          DEFB    21H
4A21 ØØ       ØØ28Ø          DEFB    Ø
4A22 4A       ØØ29Ø          DEFB    4AH
4A23 ØØ       ØØ3ØØ          DEFB    Ø
4A24 ØØ       ØØ31Ø          DEFB    Ø
4A25 ØØ       ØØ32Ø          DEFB    Ø
```

Program continued

```
4A26 00                   DEFB    0
4A27 7A                   DEFB    7AH
4A28 B6                   DEFB    0B6H
4A29 B2                   DEFB    0B2H
4A2A 00                   DEFB    0
4A2B 00                   DEFB    0
4A2C CE                   DEFB    0CEH
4A2D 86                   DEFB    86H
4A2E 56                   DEFB    56H
4A2F 94                   DEFB    94H
4A30 FC                   DEFB    0FCH
4A31 7C                   DEFB    7CH
4A32 3C                   DEFB    3CH
4A33 1C                   DEFB    1CH
4A34 0C                   DEFB    0CH
4A35 04                   DEFB    4H
4A36 84                   DEFB    84H
4A37 C4                   DEFB    0C4H
4A38 E4                   DEFB    0E4H
4A39 F4                   DEFB    0F4H
4A3A E2                   DEFB    0E2H
4A3B AA                   DEFB    0AAH
4A3C 00                   DEFB    0
4A3D 8C                   DEFB    8CH
4A3E 00                   DEFB    0
4A3F 32                   DEFB    32H
4A40 00                   DEFB    0
4A41 60                   DEFB    60H
4A42 88                   DEFB    88H
4A43 A8                   DEFB    0A8H
4A44 90                   DEFB    90H
4A45 40                   DEFB    40H
4A46 28                   DEFB    28H
4A47 D0                   DEFB    0D0H
4A48 08                   DEFB    8H
4A49 20                   DEFB    20H
4A4A 78                   DEFB    78H
4A4B B0                   DEFB    0B0H
4A4C 48                   DEFB    48H
4A4D E0                   DEFB    0E0H
4A4E A0                   DEFB    0A0H
4A4F F0                   DEFB    0F0H
4A50 68                   DEFB    68H
4A51 D8                   DEFB    0D8H
4A52 50                   DEFB    50H
4A53 10                   DEFB    10H
4A54 C0                   DEFB    0C0H
4A55 30                   DEFB    30H
4A56 18                   DEFB    18H
4A57 70                   DEFB    70H
4A58 98                   DEFB    98H
4A59 B8                   DEFB    0B8H
4A5A C8                   DEFB    0C8H

             ;RESTART ROUTINE
4A5B 3E00    RESRT        LD      A,0                ;CLEAR TEMP STORE
4A5D 21004A               LD      HL,4A00H
4A60 060A                 LD      B,0AH
4A62 77      LOOP1        LD      (HL),A
4A63 23                   INC     HL
4A64 05                   DEC     B
4A65 20FB                 JR      NZ,LOOP1
4A67 3E0F                 LD      A,0FH
4A69 32034A               LD      (SPEEDK),A         ;INZ SPEED CONSTANT
4A6C 3E01                 LD      A,1
4A6E 32044A               LD      (RCHAR),A          ;INZ RCHAR
4A71 31FF49               LD      SP,49FFH           ;INZ STACK POINTER

             ;MAIN PROGRAM
4A74 0EFF    KEYUP        LD      C,0FFH             ;RESET INTERVAL COUNTER
```

```
4A76 3EFE    KULOOP  LD      A,0FEH          ;MAX?
4A78 B9              CP      C
4A79 2803            JR      Z,NOINC         ;NO INC IF AT MAX
4A7B 0C              INC     C               ;INC INTERVAL COUNTER
4A7C 180F            JR      KUCONT
4A7E 79      NOINC   LD      A,C             ;SAVE KU INTERVAL TIME
4A7F 32094A          LD      (KUTIM),A
4A82 3A044A          LD      A,(RCHAR)       ;GET RCVD CHAR
4A85 FE01            CP      1               ;ANYTHING THERE?
4A87 2804            JR      Z,KUCONT
4A89 79              LD      A,C
4A8A CDE94A          CALL    COMPKU
4A8D CDD24A  KUCONT  CALL    CKINPT          ;CHECK INPUT
4A90 2005            JR      NZ,KD           ;BRANCH IF KEY DOWN
4A92 CDDE4A          CALL    TIMER           ;TIME DELAY
4A95 18DF            JR      KULOOP
4A97 79      KD      LD      A,C             ;SAVE KU INTERVAL
4A98 32094A          LD      (KUTIM),A
4A9B CDE94A          CALL    COMPKU
4A9E 0EFF    KEYDWN  LD      C,0FFH          ;RESET INTERVAL COUNTER
4AA0 3EFE    KDLOOP  LD      A,0FEH
4AA2 B9              CP      C
4AA3 2801            JR      Z,MAXKD
4AA5 0C              INC     C
4AA6 CDD24A  MAXKD   CALL    CKINPT
4AA9 2805            JR      Z,KU
4AAB CDDE4A          CALL    TIMER
4AAE 18F0            JR      KDLOOP
4AB0 79      KU      LD      A,C
4AB1 32084A          LD      (KDTIM),A
4AB4 FE04            CP      4
4AB6 3007            JR      NC,CKHI
4AB8 3A034A          LD      A,(SPEEDK)
4ABB CB3F            SRL     A
4ABD 1809            JR      UNZERO
4ABF FE7F    CKHI    CP      7FH
4AC1 380A            JR      C,CMPTKD
4AC3 3A034A          LD      A,(SPEEDK)
4AC6 CB27            SLA     A
4AC8 F601    UNZERO  OR      1
4ACA 32034A          LD      (SPEEDK),A
4ACD CD164B  CMPTKD  CALL    COMPKD
4AD0 18A2            JR      KEYUP

             ;SUBROUTINE TO CHECK INPUT
4AD2 AF      CKINPT  XOR     A
4AD3 D3FF            OUT     (0FFH),A
4AD5 06FF            LD      B,0FFH
4AD7 10FE    LOOP    DJNZ    LOOP
4AD9 DBFF            IN      A,(0FFH)
4ADB CB7F            BIT     7,A
4ADD C9              RET

             ;SUBROUTINE TO CREATE TIME DELAY
4ADE 3A034A  TIMER   LD      A,(SPEEDK)
4AE1 0640    DELOP2  LD      B,40H
4AE3 10FE    DELOOP  DJNZ    DELOOP
4AE5 3D              DEC     A
4AE6 20F9            JR      NZ,DELOP2
4AE8 C9              RET

             ;SUBROUTINE TO COMPUTE KU
4AE9 F5      COMPKU  PUSH    AF
4AEA 3A064A          LD      A,(TQLDAT)
4AED 57              LD      D,A
4AEE F1              POP     AF
4AEF BA              CP      D
4AF0 3823            JR      C,MOREL         ;KUTIM<3/4LDATIM?
4AF2 3A044A          LD      A,(RCHAR)       ;GET CHAR BEING RECVD
4AF5 FE01            CP      1
```

Program continued

```
4AF7 28ØB                JR      Z,CKFSP
4AF9 CD734B              CALL    GAFT            ;GET ASCII FM TBL
4AFC CD33ØØ              CALL    Ø33H            ;PRINT CHARACTER
4AFF 3EØ1                LD      A,1             ;READY FOR NEW CHAR
4BØ1 32Ø44A              LD      (RCHAR),A
4BØ4 3AØ74A      CKFSP   LD      A,(TLDAT)       ;GET 2*LAST DAH TIME
4BØ7 F5                  PUSH    AF
4BØ8 3AØ94A              LD      A,(KUTIM)
4BØB 57                  LD      D,A
4BØC F1                  POP     AF
4BØD BA                  CP      D
4BØE 3ØØ5                JR      NC,MOREL        ;BRANCH IF<TLDATIM
4B1Ø 3E2Ø                LD      A,2ØH           ;ASCII SPACE
4B12 CD33ØØ              CALL    Ø33H            ;PRINT SPACE
4B15 C9          MOREL   RET

                 ;SUBROUTINE TO COMPUTE KD
4B16 3AØ24A      COMPKD  LD      A,(TLETIM)      ;GET TWICE LAST EL TIME
4B19 57                  LD      D,A
4B1A 3AØ84A              LD      A,(KDTIM)
4B1D BA                  CP      D               ;COMPARE WITH KD
4B1E 3Ø17                JR      NC,DASHEL       ;TLETIM<KD TIME?
4B2Ø F5                  PUSH    AF
4B21 3AØ14A              LD      A,(HLETIM)
4B24 57                  LD      D,A
4B25 F1                  POP     AF
4B26 BA                  CP      D               ;COMPARE WITH KD
4B27 38Ø6                JR      C,DOTEL
4B29 3AØØ6A              LD      A,(LETYPE)      ;CK LAST ELEMENT TYPE
4B2C B7                  OR      A               ;CLEAR FLAGS
4B2D 2ØØ8                JR      NZ,DASHEL       ;GO IF LAST EL WAS DAH
4B2F 3EØØ        DOTEL   LD      A,Ø
4B31 32ØØ6A              LD      (LETYPE),A      ;MAKE LETYPE DOT
4B34 B7                  OR      A               ;CLEAR CARRY FLAG
4B35 1824                JR      ADDEL
4B37 3EØ1        DASHEL  LD      A,1
4B39 32ØØ6A              LD      (LETYPE),A
4B3C 3AØ84A              LD      A,(KDTIM)       ;GET KD INTERVAL
4B3F 32Ø54A              LD      (LDATIM),A      ;STORE IN LAST DA TIME
4B42 57                  LD      D,A             ;SAVE IN REG D
4B43 CB3F                SRL     A               ;DIVIDE KD INT BY 2
4B45 32Ø64A              LD      (TQLDAT),A      ;SAVE 1/2 KD
4B48 CB3F                SRL     A               ;DIVIDE 1/2 KD BY 2
4B4A F5                  PUSH    AF
4B4B 3AØ64A              LD      A,(TQLDAT)
4B4E 67                  LD      H,A
4B4F F1                  POP     AF
4B5Ø 84                  ADD     A,H             ;ADD 1/2 TO 1/4 KD
4B51 32Ø64A              LD      (TQLDAT),A      ;STORE RESULT
4B54 7A                  LD      A,D
4B55 CB27                SLA     A               ;MULT KD INT BY 2
4B57 32Ø74A              LD      (TLDAT),A       ;STORE RESULT
4B5A 37                  SCF
4B5B 3AØ44A      ADDEL   LD      A,(RCHAR)
4B5E CB17                RL      A               ;ADD NEW ELT TO CHAR
4B6Ø 32Ø44A              LD      (RCHAR),A
4B63 3AØ84A              LD      A,(KDTIM)       ;GET KD INTERVAL
4B66 57                  LD      D,A             ;SAVE IN REG D
4B67 CB3F                SRL     A               ;DIVIDE KD BY 2
4B69 32Ø14A              LD      (HLETIM),A      ;STORE 1/2 KD INT
4B6C 7A                  LD      A,D
4B6D CB27                SLA     A               ;MULT KD BY 2
4B6F 32Ø24A              LD      (TLETIM),A      ;STORE TWICE KD INT
4B72 C9                  RET

                 ;SUBROUTINE TO GET ASCII CHAR FROM CODE TABLE
4B73 37          GAFT    SCF                     ;RCHAR FORM CHNG
4B74 CB17                RL      A
```

```
4B76 CB27       02410 GAFT2   SLA     A
4B78 30FC       02420         JR      NC,GAFT2
4B7A 063B       02430         LD      B,59
4B7C 215A4A     02440         LD      HL,RESRT-1      ;POINT TO ENTRY
4B7F BE         02450 STAB1   CP      (HL)
4B80 2806       02460         JR      Z,TABM          ;FOUND MATCH
4B82 2B         02470         DEC     HL
4B83 10FA       02480         DJNZ    STAB1           ;END OF TABLE?
4B85 3E2D       02490         LD      A,'-'           ;"-" FOR NO MATCH
4B87 C9         02500         RET
4B88 7D         02510 TABM    LD      A,L             ;L IS ASCII FOR CHAR
4B89 C9         02520         RET
4A5B            02530         END     RESRT
00000 TOTAL ERRORS
```

TUTORIAL

Into the 80s
Part IV
Part V

Into the 80s Part IV

by Ian R. Sinclair

In the last chapter (in Volume 1) we were faced with the problem of selecting a pair of words at random from our data list, doing it by running through a random number of items and discarding the ones we didn't need. All this effort was necessary because we couldn't pick a single word at random out of the list. Now we're going to look at a method of doing that.

This method is a darn sight simpler than the name suggests. We READ the data as usual, but as each item is read we label it as a string variable, and *number* the variables. When we read off the first animal name, we number it as Q$(1), and we make its answer A$(1). Similarly, the next pair become Q$(2) and A$(2), the next are Q$(3), A$(3), etc.

Storing Variables

The computer stores these variables and refuses to be confused by any similarities between variable names. A$, A1$ and A$(1) are three separate variables which will be stored in different parts of memory and can be called up only if you use the correct titles.

One advantage of storing words like this is that we can retrieve any question or answer pair without having to sort through *all* the words. If our random choice comes up with the number two, we can then print Q$(2), and match the answer at the INPUT stage with A$(2). Remember that RND(6) generates the random numbers.

A set of strings tagged with numbers in this way rejoices in the splendid title of an array of subscripted string variables. Array means a list, and subscripted means that we've tagged each item with numbers so that we can identify them.

Nothing could be easier than setting up an array now that you know about the FOR-NEXT loop. The array in Program Listing 1 starts with the FOR N = 1 TO 6 statement, which means we start with the value of the variable N set to one. The next command is READ Q$(N), A$(N). This reads in the first word of data and assigns it the variable name Q$(1), because N is set to one. The next word is also read and assigned the variable name A$(1). That's the first question and answer pair dealt with, so the next command is NEXT. This causes the computer to increase the value it has assigned to N, and compare it with the limit we set at the start, which was six. We've just increased N from one to two, so the NEXT command moves to the READ command with N set to two. The next two words are read, assigned the variable names Q$(2) and A$(2), the control returns to the NEXT statement, and N is set at three and compared with six. This goes on until N has been set to six. At this value, the last pair of question and answer words are read in and assigned the variable names Q$(6) and A$(6). When N is increased to seven by the NEXT step, the loop is broken because seven is greater than six. All the data words have been read and converted into an array so we can pick them out as we want, using a piece of program like the one shown in Program Listing 2.

Dimensioning

Before you start using tagged variables, there's one more instruction you need to know. When you set up an array of tagged variables, the computer stores the variables in one part of its memory and keeps a note of them, along with the tags in another part. So it can organize this process efficiently, it needs to be told how many tags you might use. *Might* use, notice, not *did* use. If you specify that you might use 50 tags, but use only 20, that's all right by the TRS-80, but if you specify that you might use 50 and then try to use 51, you'll get an error message (BS) whenever you try to use the last tag, meaning that you haven't reserved enough memory space.

The number of tags which you might use on a variable is called the dimension of the variable. If you're going to enter 12 names, assigned to L$ and tagged L$(1),. . .,L$(12), the dimension of L$ is 12. The TRS-80 allows you to use dimensions of up to ten on any subscripted variable without any extra work, but if you are going to use more than this number of tags you have to enter the dimension early in the program, by using the DIM (for DIMension) statement.

DIM L$(12) means that you plan to use a subscripted variable L$ with subscript numbers which do not exceed 12. If your program has several subscripted variables, you don't need to write a separate line of DIM for each. For example, you can write DIM L$(12), P(20). Make sure that you haven't reserved more memory space than your computer has, and make sure that the DIM statement comes early in the program, well before you are going to use any of these subscripted variables. Remember also that you can use 0 as a subscript, so you can have L$(0), L$(1),. . . which lets you have an extra subscript without having to reserve any more memory.

Loading the Strings

Program Listing 3 allows you to enter word pairs from the keyboard and tag them as subscripted variables. As an added refinement, the tag number is shown alongside the word as you type it in. This is a powerful piece of program. It starts with FOR N = 1 TO 100. I picked 100 as an upper limit, but we could have used anything else. The point of doing this is to ensure that we don't put in more words than we've allowed for in the DIM statement.

N is set to one at the beginning, and the PRINT N; command prints a one on the screen, followed by the question mark which always goes along with the INPUT statement. The semicolon after PRINT N is there to make sure that the N and the INPUT word are on the same line. You can then type a question word, which will be assigned the variable name of Q$(1). Follow this with a comma, and type the answer word, A$(1), and ENTER. What happens if you forget the comma and then use ENTER? Disaster, because the two words will simply be strung together as a single word assigned as Q$(1), and the computer will print two query marks to tell you that it's waiting for the answer word. The TRS-80 is a great little computer, but it can't correct your mistakes. If you enter each word separately, you'll get a single query mark as a prompt for the question word, and a double query mark as the prompt for the answer word.

The next section of the line is an IF-NEXT-ELSE decision. If Q$(n) is not assigned to X, then the next value of N is taken and another pair of words can be used. If the letter X is input, the program breaks out of the FOR-NEXT loop, and makes N take a value one less than that assigned to it last. In this way you avoid using the value X in the program. If, for example, X is the tenth item which you entered, then N is reduced from ten to nine, because there are only nine actual items. We can now pick out any pair of words for our program and enter them from the keyboard. The idea of subscripted string variables (you can have subscripted number variables as well, but strings are more fun) is useful, but it can be extended.

A Matrix

Take a look at the program in Program Listing 4. It starts at line 10 with something which is new to you, a pair of FOR-NEXT loops together, one inside the other. This is called nested, because the first one completely surrounds the second one and the second the third (if you have one) and so on. There are two here, and they are reading what looks at first sight to be a single string variable, A$. A$, however, is a subscripted string variable, and it's not singly subscripted like A$(1), but doubly subscripted like A$(1,1). This arrangement of subscripts is set up by the use of I and J in the FOR-NEXT loops and makes for very economical programming, because in place of question string and answer string we simply have A$. It's important but difficult to grasp if you've never used anything like it before, so we'll spend some time looking at this one closely.

The first, or outside, loop starts with FOR I = 1 TO 4, so that on the first run I is given the value one. The program then moves to the second FOR instruction, and sets J at one. The READ instruction causes the first word of data to be read and labeled as A$(1,1) because I = 1 and J = 1. We would normally have two separate NEXT statements, but in this type of array we can get away with the one which is shown, NEXT J,I, which means take the next J if there is one, and if there isn't, take the next I.

Notice that you have to be fussy about the order of these variables. The NEXT variables have to be in reverse order from the FOR variables, so that if the first FOR uses I, then I must be the last variable in the NEXT. If you don't do this, your nest has holes in it. For example, if we opened with FOR X = 1 TO 5: FOR Y = 1 TO 4: FOR Z = 1 TO 2, we would have to finish with NEXT Z,Y,X.

So far we have read the data word HORSE and assigned it as A$(1,1). We then take the next J, keeping I at one, and so making J = 2. The next word, FOAL, is assigned the string coding A$(1,2). Starting to look interesting?

We're out of Js now, so the next I is taken, and I now has the value of two. This time around, with I = 2 and J = 1 (because we started back at the FOR J = 1 TO 2 again), we'll read PIG and assign it as A$(2,1). The inner loop will then cause PIGLET to be read, and assigned as A$(2,2). In fact we're assigning four sets of two words when the program has run. If you like a more abstract description, it's four lines of animals with two columns, one for parents, the other for the young. Mathematicians (may they be preserved. . . preferably in aspic) call this arrangement a matrix.

The nicest thing about a matrix of this sort is that it's easy to make neat arrangements. Line 30 gives you some idea of what can be done. Starting with the FOR statements which set up the matrix arrangement, it uses J in a PRINTTAB( ) statement to space the two columns of words neatly on the video screen. The semicolon after the A$(I,J) makes sure that the young animal's name gets printed in the same line as the old one. Follow this up with a separate PRINT command between the NEXT J and the NEXT I, or the computer will try to print everything on the same line, and fail miserably. It seems a shame to abandon that NEXT J,I already, but the results are quite satisfying. Run it and see!

Cutting Strings

And now, as they say, for something different. Remember when learning to recognize a string, using instructions like IF A$ = P$ THEN. . . ? One of the hazards of that type of recognition is that if you print one of these string variable words with a space or a misspelling, the computer simply won't recognize it. We're now going to look at ways around that problem, making use of three very powerful string selection instructions, LEFT$, RIGHT$, and MID$. Let's take them slowly, one by one.

LEFT$, as its name suggests, selects the left part of a string. You have to specify which string you want a chunk selected from, and how many letters you want to take. For example, suppose we have the instruction LEFT$ (A$,3). Whatever word is used as A$, the instruction will select the first three letters on the left-hand side. If A$ = "HORSE", then LEFT$ (A$,3) gives HOR. A$ is not affected by this, it is still HORSE. If you had spelled it as HORRES, the computer won't care if it has been instructed to look only at the first three letters. RIGHT$ does the same sort of thing. Suppose we have the instruction RIGHT$(A$,3) and A$ = "RABBIT". This time BIT is selected from the word, and A$ is still RABBIT. LEFT$ and RIGHT$ do not delete letters from words, they simply select which letters can be used for other purposes.

LEFT$ and RIGHT$ are useful weapons, but MID$ is a real missile. To use MID$, specify what string you want to operate on, at which letter you want to start, and how many letters you want to select. Suppose we take MID$(A$,2,3). If A$ = ANTELOPE the value of the MID$(A$2,3) is NTE—the selection starts with the second letter and takes in three letters.

Suppose we have a list of names stored as subscripted string variables, L$(N), which means L$(1), L$(2), L$(3) and so on. How many of these names start with the letter D? No, don't sit there and count them, write a program! Something along the lines of Program Listing 5 might suit us very well, assuming we've used a program to read in 50 names. For each value of N, the string name has its first letter compared to D. If the string doesn't start with a D, the next one is taken, but if it does, line 120 commands a printout of the name before going to the next one. We can have two commands of NEXT. This can get us into trouble if the last name does not start with D, because in line 110, the FOR. . . NEXT loop will end, and line 120 will then print L$(N), which we don't need, and asks for the NEXT again. This could cause an error report (BS), meaning that we have exceeded the dimensions we asked for.

Program Listing 6 shows a neater and flawless method of sorting out these Ds. The IF statement sorts out the Ds and prints the string, and the ELSE causes the NEXT N to be selected if there isn't a D around. The dimension is chosen to allow the NEXT command to take N to 51 without causing an error message. Now how about selecting all the phone numbers which have the same area code? Let's suppose we have 50 numbers stored in an array K(N). Not K$? Tough luck, you can't do it. All these string commands operate only on strings, not on numbers, which is why so many programs store numbers in string form by simply entering them as strings. STR$ converts any number variable into a string variable. For example, if we have the statement K$ = STR$ (234), then 234 becomes assigned to the string variable K$ just as thoroughly as if we had written K$ = "234" in the first place. If we have 50 number variables, K(1) through K(50), you just add the line:

```
FOR N=1 TO 50: K$(N) = STR$(K(N)).
```

Watch the double set of parentheses, because if you miss one, you'll get the SN error message. Now that you've got your phone numbers in the form of a subscripted string variable array you can pick off the area codes by using

```
A$ = LEFT$ (K$(N),3)
```

Once again, we have parentheses within parentheses, and you have to be sure that you've included all of the parentheses.

Packing Strings

Suppose you stored each name and number together in one string as K$(N). Quite a savings in memory is gained by doing this, because there's only one string to store for each name/number, instead of a separate pair of strings or a string and a number. How do we separate them so that we can print out something that looks rather more civilized than JOHN W DOE 2141673802? One neat and simple method makes use of the statement VAL.

VAL means: Find the number value inside a string. It's usually used when a number has been converted into a string by using STR$ and you now want to convert back. As it happens, you often want to do things with numbers which you can't do with strings, like multiplication, division and subtraction, for example. Addition is a bit different, and we'll be looking at what happens when we use the + sign on strings in a moment.

Many computers use VAL just for converting a number string back to a number, but the TRS-80 BASIC goes one better. If you have a string which starts with a number, like 1024 SUNRISE AVENUE, you can extract the number out of the string by using VAL. If, for example, you run the little program:

```
10 A$ ="1024 SUNRISE AVENUE"
20 PRINT VAL(A$)
```

What is printed out is 1024, the number which the VAL statement finds at the start of the string. VAL can only find a number at the start of a string, however. If you have A$ = JOHN DOE 2174267803, then VAL(A$) is zero, because the number follows the letters.

This doesn't prevent you from writing your own routine, using MID$(A$,N,1) to strip characters off the string one by one and test their ASCII codes to find if they are numbers. The ASCII codes for numbers are 48 through 57, so you could detect numbers anywhere in the string and print them out.

To separate numbers from names by using VAL, we have to place the number first, coding our number/name in the form of 2172677803JOHN DOE.

Program Listing 7 assumes that you have a set of data lines which contain your telephone number and name strings. Line 10 is straightforward—we are just reading each item and labeling it as a string array A$(N), allowing for 50 items. If you don't want to try 50 for starters, make it two and use just the data in line 50.

Line 20 asks for the surname of the person whose number you want typed. From what you know of computers by now, you should not be surprised to learn that your typing of DOE had better match exactly with the DOE which you have stored in the data line!

In line 30, L = LEN(X$), X$ is the string variable assigned to the name you typed at the INPUT stage, and LEN means: Measure how many characters are in this string. The answer here is three. If we know how many characters are in the surname, and that the surname is at the right hand side of the string, we can pick the surname out of A$ in line 40, by setting a variable L equal to the length of the name string X, and then find RIGHT$(A$(N),L). We could equally as easily have saved a line by writing in line 40:

```
RIGHT$(A$(N),LEN(X$))
```

making sure not to leave out any of the parentheses. It's a useful feature of our BASIC that we can use expressions like LEN(X$), as well as simple numbers and variables in the RIGHT$, LEFT$, MID$ and other expressions.

In line 40 each item of data is examined, and the correct number of letters on the right hand side is stripped off to compare with X$, which might be DOE. If the last three letters are not DOE, then the next string is taken, and if the last three match up, the final part of line 40 instructs the computer to print the telephone number by taking VAL(A$(N)).

Here's another use for RIGHT$. Suppose you have a set of numbers which lie between one and 25, and you want to put them into string form so that each has two digits, like 21, 01, 18, 06, ... If you write numbers in this way, you can put them into a string and get them back easily, because you always want the same number of characters back—two in this example. If you had these at the end of a string, you could use RIGHT$ (A$(N),2), for example.

Program Listing 8 shows how this operation of padding numbers out can be achieved. The number in this example is typed in as an answer to the INPUT query, and we take the chance to assign it to a string variable, N$. In line 20, we redefine N$ as being equal to one space plus the old value of N$. When we use a + sign with two string quantities, the quantities are simply run together, or concatenated. If we had typed N$ = "*" + N$, then with N$ = "2", the result would be *2. As it is, line 20 uses a zero between quote marks so that the new N$ consists of the number we read in with a zero in front of it.

In line 30, we define another N$, this time the RIGHT$ of the N$ with a zero in front. If that N$ were 02, the RIGHT$ will give 02, but if N$ were 021, RIGHT$ would give just 21. Either way, the number consists of just two characters, and can be selected again by picking off two characters from the string we put it in. Both words and numbers can be padded out in this way to a standard size (two characters, 10, 20, whatever you like so long as it doesn't exceed 255) so that they can be easily selected again.

One small problem arises here. If you have converted a number into a string by using STR$(number), the computer will automatically put a space in front of the number to make room for a negative sign if one is needed. That way, if you use STR$(5), you get a string which is two characters long, and STR$(50) is three characters long, though STR$(– 50) is also three characters long. If this might cause problems, one way out of it is to use RIGHT$. To pad out to two characters we use:

```
A$(N) = RIGHT$(" " + A$(N),2)
```

Your numbers will be two-character strings no matter what STR$ has done to them, but watch out for negative numbers!

Time to leave the LEFT, RIGHT, MID business, and look at other things, but before we do, look at Program Listing 9. This is one answer to the word recognition part of a mailing list program. If you have the first two letters correct, the comparison is good enough. You have to use this type of recognition carefully, however, because if you have two names which start with the same letters, like ANT or ANTELOPE, the computer doesn't know the difference. In fact, the MONKEY, DONKEY problem is the worst you're likely to get!

Presentation

How do you underline a word you have printed on the screen? There's no way of underlining on the same line as the letters of the word, but if there is space on the next line (*make space*) the problem has a solution. Program Listing 10 shows one solution—the title words are printed, and, on the next line, using quotation marks, the characters of underlining. The equality sign and the asterisk are useful for this job. Big BASIC, however, offers a lazy way of underlining in the form of the STRING$ function. STRING$ is a statement which instructs the computer to print identical characters.

There are two ways of specifying which characters we want to string together. If we type in:

```
PRINT STRING$(24," = ")
```

the computer will print a string of 24 equality signs. Similarly, PRINTTAB(20)STRING$ (24,"A") will produce a row of 24 As starting at tabulator position 20. Another way makes use of the ASCII codes for the numbers, letters, and characters.

Who is this ASCII, you ask me? It stands for American Standard Code for Information Interchange, and it's a number-code method of transmitting characters. How do you find the ASCII code for a character? The hard way is to look it up in the TRS-80 manual. The easy way is to ask the computer.

PRINT ASC("*") will bring up the code which represents the asterisk. We can easily find other codes.

The character which is represented by ASCII code 128 is a blank. It's not the same blank as the one which is represented by ASCII 32. It's possible to have two different blanks. If that sounds weird to you, think of this. The blank represented by 32 can be entered from the keyboard (by using the space bar), but 128 can't. When the computer finds 128 in a string, it can be instructed that this is the end of a string and the start of another. It's a useful distinction.

We can now redesign our underlining statement in line 20, using STRING$, so it looks like Program Listing 11. There's no reason for these two statements not going into one line, saving memory. Each time you start a new line, you use five bytes of memory, so it pays well to pack the lines as much as possible this way.

CHR$ stands for the character or action represented by the number in parentheses following CHR$. For example, PRINT CHR$(68) causes a D to be printed because 68 is the ASCII code for the letter D. Of course, there's a catch: A lot of ASCII codes don't represent letters. They represent actions, and we can have the computer carry them out by using the PRINT CHR$( ) command.

One pair of codes which are peculiar to the TRS-80 are 23 and 28. PRINT CHR$(23) causes the display to print double-size letters and numbers until the command is cancelled by one of a variety of methods. After CHR$(23) has been printed, we have to be careful how we use TAB and PRINT@ numbers because with double-size characters, there are only 32 characters per line, and eight lines on the screen! A command to PRINTTAB(35) isn't going to produce a letter in the middle of the screen. In the same way, the PRINT@ instructions go only to 256, not to 1023.

Double-sized lettering is excellent for titles and for drawing attention to error messages, but the uses suggested in the manual are limited. The character size returns to normal when the CLEAR key is pressed, a CLEAR command used, or CLS used.

Sometimes you don't want to lose what has been printed in the large characters, yet you want more lettering on the screen in smaller print. You can't have a mixture of large and small letters. The PRINT CHR$(23) command operates on the part of the memory which stores the video display characters, and affects either none of it, or all of it at once. One method I use in my own programs is shown in Program Listing 12.

Line ten skips the first line of the screen. It could be done just as easily by using a PRINT@ command in line 20, but we've opted to use TAB. The CHR$(23) sets up the big letters, and we print the title and underline it. Line 40 simply arranges a time delay so we can sit back and in line 50, the PRINT

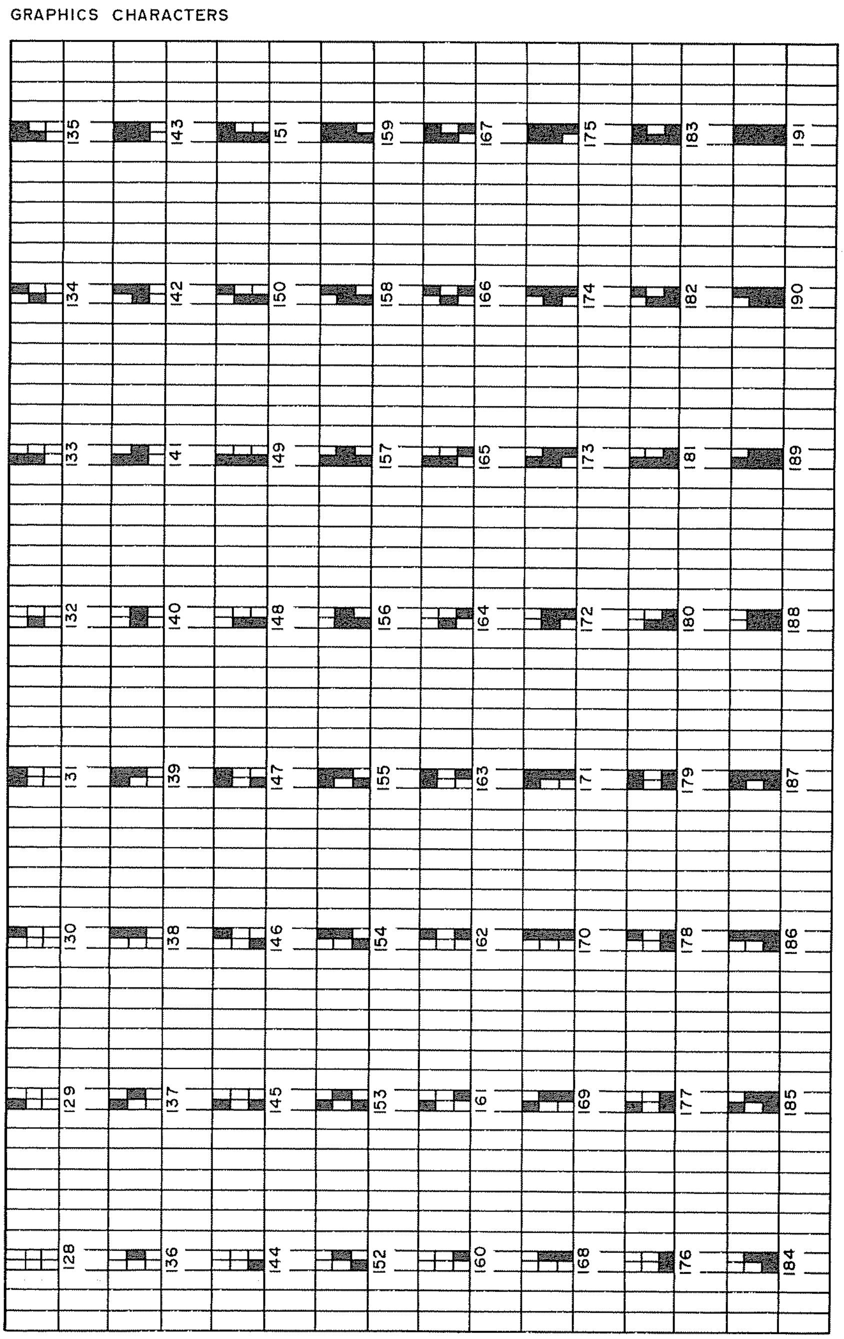

GRAPHICS CHARACTERS
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191

Figure 1

CHR$(28) then restores the lettering to normal size. It was not intended as a way of restoring normal size letters but as a method of wiping out the top line! The top line was left blank as it will otherwise be wiped clean by the PRINT CHR$(28) command. Notice also that the lettering which was printed double-size is now normal size but double spaced. It still looks good as a title.

We've also had to position the next line using PRINT@ to keep out of the way and avoid wiping out any other lines.

Another way of getting double-spaced print and returning again uses the instruction POKE. The method is shown in Program Listing 13. We'll deal with the POKE instruction later; all we need to know for now is that it can change the contents of the memory directly, and more quickly than the usual BASIC instructions. You can mix these commands, using PRINT CHR$(23) to start the big print, and POKE 16445,0 to stop it. For some curious reason, however, the stop command does not work in every program. I have one program in which POKE 16445,0 works perfectly, and another in which it has no effect. I still haven't discovered why.

The sharp-eyed folks will already have sensed that there's more to tell about the PRINT CHR$( ) instruction. It's not particularly useful for printing letters or even punctuation marks. It's just as easy (and easier to follow) if you just type PRINT A or PRINT ; or whatever. CHR$( ) has been found useful for producing effects in a program which we can't get directly from the keyboard, such as the CHR$(23) and CHR$(28). Table 1 shows more of these effects taken from the Level II manual.

Code	**Function**
0-7	None
8	Backspaces and erases current character
9	None
10-13	Carriage returns
14	Turns on cursor
15	Turns off cursor
16-22	None
23	Converts to 32 character mode
24	Backspace ← Cursor
25	Advance → Cursor
26	Downward ↓ linefeed
27	Upward ↑ linefeed
28	Home, return cursor to display position(0,0)
29	Move cursor to beginning of line
30	Erases to the end of the line
31	Clear to the end of the frame

Table 1. *Control codes 1-31*

There are a lot of ASCII codes which can't be entered from the keyboard but which make their appearance in many programs. These are the graphics characters. One unit of memory, a byte, can store a number of size up to 255; since the highest number of ASCII code for letters or characters is 128, however, that leaves a large number of unused codes. In the TRS-80 these are used for graphics characters. The later Level II manuals include a printout of these characters, but the earlier manuals didn't. For everyone who is now struggling with an old manual, Figure 1 shows what the graphics characters look like, with their code numbers. To see any of these characters for yourself, look up its code number. Use the command PRINT CHR$(number).

Bigger Graphics

Going on to Sinclair's Second Law—that what they don't tell you in an instruction manual is as important and as useful as what they *do* tell you. You may have sensed that there's a lot more to this business. Each printing position on the video screen consists of six small blocks or cells, and the graphics characters are formed by lighting up various combinations of these cells. Why shouldn't we light up more than one cell at a time in a given block? And there's also no reason why we shouldn't light up more than one block at a time. We can do this by combining codes; Program Listing 14 shows an example. G$ is defined as the combination of two graphics characters. Each time we command PRINT G$, we'll get that combination, and we can, of course, use the usual printing options of PRINTTAB( ) G$ and PRINT@, G$ to position the set of characters where we want them. We can also use tricks like defining two sets of graphics characters, G$ and H$, and then writing

```
PRINT@N,G$:PRINT@(N + 1),H$
```

which will print the two sets side by side, starting at the position set by the value of the number N (between 0 and 1022).

Alternately we can use:

```
PRINT@N,G$:PRINT@(N + 64),H$
```

which will print G$ at position N, and H$ directly underneath it. Adding 64 to N moves the printing position to the line space immediately below, since we now have 64 print positions in a line. When we used large print we used only 32 characters per line, and there are 64 in the program now.

In Part V we'll be looking at the SET and RESET commands, which are a free range way of creating shapes. Then in the final section of this series we'll investigate the POKE command which can speed up the process of drawing shapes.

INKEY$

INKEY$ can make your program a lot more interesting; it's always by a statement like

```
K$ = INKEY$
```

What is INKEY$? It refers to the value of the character which is fed into the computer when pressing the key, just as the computer is scanning the keyboard contacts looking for a key being closed. This scanning takes place continuously when the computer is being used to enter a program, and during much of the time when a program is running in order to detect the BREAK key being pressed. It is halted during a CLOAD or CSAVE, an LLIST or LPRINT. You can't affect what goes on during these operations by punching keys. The RESET button alone, located at the back of the computer, will stop a CLOAD or CSAVE (and will usually corrupt the tape as well). Incidentally, having the continuous keyboard scan means that if you are using a simple keyboard delay routine as a bounce fix, your programs are running slower!

This scan operation is fast but chances are, if you just wrote K$ = INKEY$ into a program and let it run, there wouldn't be a key pressed down at the instant when that line of program was carried out, so K$ would be a blank string at first. We get around this by looping around the instruction, forcing it to repeat itself until something is entered by pressing a key. A line such as:

```
50 K$ = INKEY$ : IF K$ = " " THEN 50
```

does just that. If the value of K$ is a blank, the line runs again. It keeps running until a value is entered from a key. The value of that key is then stored as K$. If the key is just a letter key or a number key, its value can be printed by making the line read

```
50 K$ = INKEY$: IF K$ = " " THEN 50 ELSE PRINT K$
```

This is a useful way of entering letters or numbers without hitting ENTER. An example of this sort of thing is shown in Program Listing 15. In line 10, K$ is set equal to INKEY$ and looped back waiting for a key to be pressed. The number value of K$ is found by using K = VAL(K$). If the key is a letter key, its number value is zero, and the message in line 20 is printed. If the key is a number key, its number value is not zero (unless it is the zero key) and the message in line 30 is printed.

Take a look at Program Listing 16. This is useful for YES/NO choices, because it lets you see the word build up on the screen, and returns at once when the correct word is selected, without needing to hit ENTER. In addition, it signals back to the main program what has been typed, using M = 1 to mean YES and M = 0 to mean NO. Try it out, and then think how it might be improved. Perhaps a flashing asterisk or dash to remind you when you need to enter another letter?

Program Listing 1

```
500 FOR N=1TO6:READ Q$(N),A$(N):NEXT
510 FOR N=1 TO 6:PRINT Q$(N),A$(N):NEXT
```

Program Listing 2

```
150 DATA"LION","PRIDE","WHALE","SCHOOL","FISH","SHOAL",
    "SHEEP","FLOCK","COWS","HERD","GEESE","GAGGLE"
500 FOR N=1TO6:READ Q$(N),A$(N):NEXT
510 R=RND(6):PRINT Q$(R)
```

Program Listing 3

```
10 DIM Q$(100),A$(100)
20 FOR N=1TO100:PRINT N".";:INPUT Q$(N),A$(N):IF Q$(N)<
    >"X" THEN NEXT ELSE N=N-1
30 FOR Y=1TO N:PRINT Q$(Y),A$(Y):NEXT
```

Program Listing 4

```
10 DIM A$(6,6):FOR I=1TO4:FOR J=1TO2:READ A$(I,J):NEXT
    J,I
20 DATA "HORSE","FOAL","PIG","PIGLET","DOG","PUPPY","CO
    W","CALF"
30 FOR I=1TO4:FOR J=1TO2:PRINTTAB(20*J)A$(I,J);:NEXT J:
    PRINT:NEXT I
```

Program Listing 5

```
10 DIM L$(50):FOR N=1TO5:INPUT L$(N):NEXT
100 REM INTO 80'S FIG 4.5 FAULTY EXAMPLE
110 FOR N=1TO50:IF LEFT$(L$(N),1)<>"D"THEN NEXT
120 PRINT L$(N):NEXT
```

Program Listing 6

```
10 DIM L$(51):FOR N=1TO5:INPUT L$(N):NEXT
110 FOR N=1TO50:IF LEFT$(L$(N),1)="D" THEN PRINT L$(N):
     NEXT:ELSE NEXT
```

Program Listing 7

```
10 DIM A$(51):FOR N=1TO50:READ A$(N):NEXT
20 INPUT "SURNAME"; X$
30 L=LEN(X$)
40 FOR N=1TO50:IF X$<>RIGHT$(A$(N),L) THEN NEXT ELSE PR
     INT VAL(A$(N))
50 DATA "217467803JOHN DOE","2170322104TIM BUCK" :REM Y
     OU NEED A TOTAL OF FIFTY ENTRIES!
```

Program Listing 8

```
10 INPUT "NUMBER BETWEEN 1 AND 25, PLEASE";N$
20 N$="0"+N$
30 N$=RIGHT$(N$,2)
40 PRINT N$:GOTO10
```

Program Listing 9

```
10 IF LEFT$(A$,2)=LEFT$(L$(N),2) OR RIGHT$(A$,2)=RIGHT$
     (L$(N),2) THEN PRINT "CORRECT, WELL DONE!"
```

Program Listing 10

```
10 CLS:PRINTTAB(23)"THIS IS THE TITLE"
20 PRINTTAB(23)"==== == === ====="
```

Program Listing 11

```
10 CLS:PRINTTAB(21)"THIS IS ANOTHER TITLE"
20 PRINT TAB(21) STRING$(21,42)
```

Program Listing 12

```
10 CLS:PRINT:PRINT
20 PRINT CHR$(23)TAB(12)"TITLE"
30 PRINTTAB(12)STRING$(5,42)
40 FOR N=1TO1000:NEXT
50 PRINT CHR$(28):PRINT@384,"NEXT LINE OF MESSAGE"
```

Program Listing 13

```
10 POKE 16445,8
20 PRINT"HAPPY BIRTHDAY!":FOR N=1TO1000:NEXT
30 POKE 16445,0
40 PRINT "TO YOU...."
```

Program Listing 14

```
10 G$=CHR$(153)+CHR$(166)
20 CLS:PRINT G$
30 PRINTTAB(32)G$
40 PRINT@350,G$
50 END
100 REM TYPE RUN 100 TO RUN THIS ONE
110 G$=CHR$(154)+CHR$(165):H$=CHR$(183)+CHR$(187)
120 CLS:PRINT@150,G$;:PRINT@155,H$
130 FOR N=1TO1000:NEXT
140 CLS:PRINT @480,G$:PRINT@544,H$
```

Program Listing 15

```
5 PRINT "PRESS ANY LETTER OR NUMBER KEY"
10 K$=INKEY$:IF K$="" THEN 10
20 K=VAL(K$): IF K=0 THEN PRINT"YOU ENTERED THE LETTER
     ";K$
```

```
30 IF K<>0 THEN PRINT "YOU ENTERED THE NUMBER "; K$
40 END
```

Program Listing 16

```
1000 A$=""
1010 K$=INKEY$:IF K$="" THEN 1010 ELSE PRINT K$;
1020 A$=A$+K$:IF LEN(A$)<2 THEN 1010
1030 IF LEN(A$)=2 AND A$="NO" THEN M=2:GOTO2000
1040 IF LEN(A$)=3 AND A$="YES" THEN M=1:GOTO2000
1050 IF LEN(A$)=2 THEN 1010
1060 IF LEN(A$)>3 OR A$<>"NO" OR A$<>"YES" THEN GOTO201
     0
1070 END
2000 IF M=1 THEN PRINT "  THE ANSWER IS YES":ELSE PRINT
      "  THE ANSWER IS NO"
2005 END
2010 PRINT "  YOU HAVE MADE A MISTAKE- PLEASE TRY AGAIN
     .":FOR N=1TO500:NEXT:GOTO1000
```

Into the 80s
Part V

by Ian R. Sinclair

In parts I through IV of "Into the 80s" we learned how to program a computer, with hardly a word about math. It was too good to last, and in this chapter we're going to dive into some of the mathematical capabilities of the TRS-80.

Simple Calculator?

Let's start at the beginning.

The + sign is the add command of the TRS-80, and when you use it with numbers or variables, which have number values, it does what you expect it to do. If you type: PRINT 25 + 37 and ENTER, the screen will show the number 62 below your line. This is using the TRS-80 just like a hand-held calculator, but that's not exactly what you bought it for, is it?

Program Listing 1 is a step in the right direction. In line 10, you are reminded of what the program should do. Then type in two numbers, separated by a comma, and ENTER. In line 20, the numbers are added, giving the total, T. Line 30 prints this lot, helpfully indicating that the number being printed is the total. The program then prints a blank line, waits, and asks for another pair of numbers. If you want to break out of the endless loop, hit the BREAK key. It's a simple program, but it does illustrate the big difference between a computer and a calculator. As we go on, that difference will become more obvious.

Suppose we want to keep a running total. We're going to enter many numbers, and we want to keep a record of how many we've entered and what the total is. Just to make it work for its money, we'll make it print the total and the number of entries each time we enter a new number. Program Listing 2 shows the method. Start by setting two number variables T and N to zero. We set them at zero to start with and add to them during the program, and thereby maintain control over the total. It's like saying "Here's a dollar. Put it in your pocket. How much is now in your pocket?" If you knew that your pocket was empty, the problem would be pretty simple.

At line 20, the program asks for a number to be typed in and entered, and this number is assigned the letter A. We use line 30 to end the program; if an entry is zero, steps 40 through 60 are skipped, and the program ends. If a number is not zero, line 40 does the arithmetic. The statement T = T + A adds the input number to the total. The first time we do this, T has been set to zero, so if the number we fed in was 16, then T = T + A sets T to the value 0 + 16, which is 16. Next time T will start at 16, and whatever number you type will be added. This is the part of the program which totals up the numbers entered.

The second part of line 40 is N = N + 1. Once again, variable N is set to zero in line 10, and on the first step it becomes 1, because 0 + 1 is 1. Second time around, it's made equal to 2, and so on. This variable keeps note of how many numbers have been entered. At line 50, the number of entries and the total are displayed, and the program then loops back to line 20 for another number. Looping back to line 10 would set the count numbers T and N to zero again, and we would lose our totals.

Look at Program Listing 3, which produces the same effect as Program Listing 2, only by adding four sets of numbers at the same time and printing out four totals each time. Unless you can punch four calculator keyboards at once, you're not going to find much competition for the TRS-80 in tasks like this!

Subtraction is so similar to addition that we needn't spend any time on it. The subtract sign is on the keyboard, and it's used in programs the same way as the add sign. The difference is that subtraction can cause negative numbers to be printed, as when you subtract 5 from 3 leaving − 2. This is no hassle for the TRS-80, which simply prints − 2.

Multiplication

Multiplication uses the asterisk sign *. We can't use × for multiplication the way we do on paper because X is a letter symbol, and the TRS-80 can't tell the difference. We can check multiplication in action without writing a program by typing: PRINT (16*1.5) and ENTERing. The parentheses are not needed in this expression, but using parentheses is a good habit, as I'll explain.

As you've probably gathered by now, using the computing power of the TRS-80 just to multiply two numbers is a bit of a waste. The computer scores when a large number of operations are carried out and a result displayed. As an example, take a look at Program Listing 4, a simple program which prints out a multiplication table (up to 12 times) for any number you enter in line 10. Notice, we've made use of a FOR-NEXT loop to get the sequence of numbers from one through 12. Similarly, we can make use of division in programs by using the / sign, so that division problems such as 38/4 are written easily into a program.

There's nothing difficult about any of these four operations, but it's not difficult to get into a muddle when performing different bits of arithmetic. For example, suppose you saw 3 + 3 * 6 − 8/2. The answer you get from this depends upon which order you carry out the operations. If you take it as it's written, you'll add three to three to get six, multiply by six to get 36, subtract eight to get 28, and then divide by two to end with 14. Some calculators would also solve the problem this way. Another scheme depends on what's called a hierarchy of order, where multiplication and division are done before addition and subtraction.

Your TRS-80 has been well trained to decide which operations to carry out first and to obey your instructions. If there are no parentheses around any quantities, multiplication and division are carried out first, in left to right order. Then, addition and subtraction, also left to right. This is only part of the order which is printed on page 1/6 of the Level II manual. I never feel entirely happy letting a machine decide what order it will take for these operations, so I use parentheses. The computer will carry out any operation inside parentheses before it does anything else. If you have nested parentheses (one pair inside another), the innermost are done first, followed by the next set outwards. Within a set of parentheses, left-to-right priority rules apply.

As an illustration, look at Program Listing 5. It's an electrical problem concerning the internal resistance of a battery. A battery has a voltage which is steady when not drawing any current, but which decreases when drawing current because of internal resistance. The formula which is used is V = E − r * I, where E is the voltage (called the open-circuit voltage when no current is taken), r is the amount of internal resistance, V is the voltage present when current flows, and I is the amount of current. Suppose we want a table demonstrating the effect of a range of currents on the output voltage of a battery. Program Listing 5 does that and also checks that the value of internal resistance looks reasonably sensible. The STEP instruction is one we haven't used before. It ensures that the step is 0.1, whereas if no STEP is given, a step of one would be automatic. The display used in this program shows the superiority of the computer over the calculator.

In line 40, two headings are printed, one for current and the other for voltage. Line 50 sets up another FOR-NEXT loop, using the same values of current, and in line 60 these are printed at the correct place. The voltage values are printed using the format command, PRINTUSING, so that no more than two decimal places are printed.

Program Listing 5 is one example of a program which works out results from a formula and sets them in table form. This sort of thing has wide applications in engineering, statistics, and finance, among other uses. Before we go further along this track we need to know what other math operations the TRS-80 can do.

First is exponentiation, which means multiplying a number by itself. The expression 2^3 means multiply 2 by itself three times, meaning 2 * 2 * 2 = 8. In BASIC, this is written as 2 ↑ 3, so that entering PRINT 2 ↑ 3 should come up 8.

Exponentiation will always be carried out first, unless there are other expressions inside parentheses in the same line. A fractional exponent has the same meaning as a root. For example, an exponent of 0.5 gives the same result as a square root, and an exponent of 0.33333 is the same as a cube root. For convenience, the square root is always separately coded as SQR, so that entering PRINT SQR(25) comes back with the value five, as if we used PRINT 25 ↑ .5.

Eternal Triangles

If you know the lengths of the two short sides of a right triangle, A and B, you can find the length of the long side, C (called the hypotenuse), by using the formula $C^2 = A^2 + B^2$. Program Listing 6 prints out the length of the hypotenuse for any pair of other sides entered. For good measure, we've made it show the total perimeter (equal to A + B + C) as well. Lines 20 and 30 ask for the side lengths, in any units you like, as long as they are the same measure. The calculation is carried out in line 40, and then there's a step which may have caused your eyebrows to lift slightly. What does C = (INT(100*C))/100 do?

The INT instruction means "take the integral part of"—chop off the decimal point and anything which follows. Suppose C starts as 26.2615. Since the order of carrying out instructions starts on the inside parentheses, 100 * C is first of all evaluated as 2626.15. This is inside another set of parentheses, so the next step is the INT step, taking the whole number part of 2626.15, which is 2626. This is finally divided by 100 to give 26.26, which is allocated the variable name C. The answer is down to two decimal places so that we don't have too many in the answer, printed in line 50.

Is this desirable? If we are entering values of A and B, which are numbers greater than one, fine, but if A = 0.3 and B = 0.4, then C should be 0.5. This works out all right, but if A = 0.003 and B = 0.004 then the value for C, which should be 0.005, comes out zero. There are two ways to avoid this. One is to reject (upon entry) any values of A or B which are too small. The other is to ignore the C = (INT(100*C))/100 step if A and B are less than 0.01. You're not really a beginner now, so you can try these out.

Translating other formulae into BASIC is not difficult, but you need to be familiar with algebra. The TRS-80 can also cope with trigonometrical functions. The main functions can be obtained by typing SIN, COS, or TAN, but the angles have to be in *radians*, not in more familiar degrees. The Level II manual shows how you convert, by multiplying the angle in degrees by 0.0174533, so that you can have SIN(A*.0174533) as a way of finding the value of SIN A, with A in units of degrees. If you are going to use several conversions, it saves a lot of memory and running time if you have, early in your program, a step such as F = .0174533, and then write the formulae as SIN(A*F), or COS(A*F), or TAN(A*F). The manual also lists the other trigonometrical functions and formulae. Program Listing 7 uses trigonometry to calculate the side of a triangle.

Imprecisions

Before we break away to other things, there are a few important points about using numbers in the TRS-80. You need to know about them if you are not to be mystified by the results of some of your own programs. At some time, you may try to write a simple financial program which involves adding and subtracting sums of money, and you'll be intrigued (if it's not your money) or infuriated (if it is your money) to find that sums are often a cent or so off. How can a computer do such a thing?

The answer is the problem of precision. The degree of precision of a quantity is measured by the number of digits it can handle—you are probably familiar with calculators which work with eight figures. Look at some examples: The number 741.36 has five digits of precision, 42.5 has only three, and 1024.76 has six. Level II BASIC makes use of three levels of precision, and a lot of the odd results you get arise from "rounding off" within the computer, when numbers are cut to fit the level of precision chosen.

Unless you instruct the computer to the contrary, a variable is stored and printed as a single-precision variable. Single precision, as far as the TRS-80 is concerned, means that it will store seven digits and print out six. The sixth digit will be rounded up, and if this happens often, the errors will add up (a cumulative error) to something noticeable. If you don't want this (or if you want it to happen in a bigger way!), you can change things.

An integer is a whole number—no fractions allowed—and the permitted range on the TRS-80 is − 32768 to + 32767. This is the range of numbers we can obtain by using two bytes to store the binary numbers that the TRS-80 uses, so that by declaring a variable to be an integer, we need reserve only two bytes of memory for it. We can declare a letter to be an integer variable by using DEFINT at the start of a program, or by using a "type declaration" character, in this case %. N% means that N is an integer variable, just as N$ would mean that N is a string variable. If we use DEFINT N at the start of a program, then N must be used as an integer throughout, but if we use N%, then we can also use N$, N#, and N!, all meaning different values. The hashmark # means a double-precision variable, and the ! means single precision. Notice, by the way, that if you use integers, no fractions can appear, so that if you type N% = 5:PRINT N%/2, you get 2, and not 2.5.

The other degrees of precision, as mentioned above, are single and double precision; all variables are treated as single precision if we don't make any effort to declare them as anything else. A single-precision variable needs four bytes of memory; a double-precision variable needs eight and contains

17 digits, of which 16 can be printed. A string variable will need as many bytes as there are characters in the string (up to 255).

If your programs use a lot of counting loops, with variables like N,Z,T, and so on, you can make them run faster and use less memory if the first line is formulated as DEFINT N,Z,T (and any others like them). This way, the numbers will take less memory and can be taken in and out of memory more quickly.

The other point comes back to those missing cents. The rounding down which is done when a number is printed can also cause errors. The most suspicious steps in any program are where numbers containing decimals are multiplied together because, when you multiply two single-precision numbers, the result may have too many digits to store as a single-precision number. Consequently, a rounding-off error results. If the quantities are added, more errors of the same type will occur.

There are two useful wrinkles for avoiding this problem. One is to work all money amounts in cents. If you work in cents and use S = INT(S) every now and again after a step which might cause fractions to appear, you should avoid trouble. The other is to round up occasionally (and close the corral gate after you). We do this with the instruction C = INT(C + .5). How does it work? Suppose C has taken its value from multiplying two numbers, and rounding off has caused this to be 176.999 instead of 177. Adding .5 to this makes it 177.499, and INT(177.499) is 177, since INT chops off the decimal part of the number.

Free Range Methods

We took a brief look in Part IV at the graphics characters of the TRS-80 which allow you to put shapes on the screen by using the CHR$( ) command, or a PRINT A$, where A$ is defined as a number of graphics strings. This time we're going to look at free range methods, including those used to display bar charts and graphs.

The commands which make this possible are SET and RESET. SET means light up a graphics cell, one of the block of six at each PRINT position. RESET means turn it off. If you command SET and the cell has been lit, there is no change. Similarly, if you command RESET and the cell has not been lit, there is no change. SET and RESET are followed by numbers in parentheses to tell the computer which cell to SET or RESET. The first number measures how far on the width of the screen the SET position is. If you're into graphs, this is the x-direction. We have a maximum of 64 print positions, each two graphics cells wide, making 128 cells, numbered 0 to 127. In the vertical direction we have 16 lines, each three cells deep, making 48, numbered 0 to 47. The SET or RESET must be followed by (X,Y), where X is a number (an integer) between 0 and 127, and Y is another integer between 0 and 47.

These commands open up possibilities for interesting graphics work, not the least of which is the opportunity to do a bit of animation. Look at Program Listing 8, which flashes a graphics block on and off. To get out of this you need to use BREAK, because the loop is endless, but you already know how to make this flash a number of times and then stop. Program Listing 9 is a crawling worm graphic which we're going to develop a bit further. It starts by clearing the screen (line 10) and setting Y = 5, which is the vertical setting for the worm's path. The worm is created in line 30 by setting a line of five graphics blocks. Line 40 simply adds a delay. The animation starts in line 50. Taking values from 0 to 127, we reset the left-hand cell of the worm and set a new right-hand cell, making it appear that the worm crawled one cell to the right. The FOR-NEXT loop uses Z, then another delay, and then the process is repeated. If we are not careful, we will get an error message, because the SET(N + 5,Y) instruction will not operate when N exceeds 122, as we have only 127 cell numbers along the line. We get around that by using an IF-THEN statement. If the value of N is 122 or less, the line runs normally, but if N is 123 or more, the ELSE part of the statement simply bypasses the SET command, returning to the next value of N.

Want a snake rather than a worm? We'll need to stretch it out a bit in line 30, or you won't notice the wiggle. To make it "wiggle," we'll make the value of Y change now and again, and that's more difficult. A reasonable way of making Y vary is to make use of the SIN function. The math majors will tell you that the sine of an angle is the ratio of two sides of a right-angled triangle, but I prefer to think that the name suggests more interesting things. The word sine comes from the Latin word for snake, because if you plot a graph of the sine of an angle against the angle (Program Listing 10), the shape is the wiggle.

Take the value of Y as Y + (5*SIN(N)). Sine values repeat every 360°, so that if we use angle values in degrees we would see the shape repeating. As we noted though, the SIN function of the TRS-80 does not use angles in degrees but in radians. In Program Listing 10, we use the correcting factor of .1745, taken from the Level II manual, which converts degrees to radians.

Program Listing 11 is the wiggling program. We set up a series of subscripted number variables, Y(N), not forgetting to dimension this correctly in line 10. With the screen cleared, line 30 introduces the snake from the left-hand side by setting values of N, and a value of Y equal to 7 + Y(N). Y(N) takes on values which can range between + 3 and − 3 because of the 3*SIN(N/4) function in line 20, and this creates the wiggle between values for Y of 10 and 4 (7 + 3 and 7 − 3, see?). The value doesn't just leap from one extreme to the other, but snakes its way there, which is what we want.

To animate a track across the screen, we need line 40. It advances the "head" of the snake and rubs out the "tail" at each step, using a short delay to make sure that progress is slow enough to follow. If you fancy faster or slower snakes, you only have to alter the delay loop which starts with FOR Z = 1 TO 100. The reason for putting the wiggle values into a subscripted variable is so that we can rub them out correctly as the snake moves along. It's not the only way of doing this, but it's the easiest.

Graphs and Bar Charts

The uses of SET and RESET aren't confined to games and amusements; there are several serious and useful applications in math and statistics. For our purposes, the most useful are for drawing graphs and bar charts. The conventional directions of a graph are X and Y, with X being used to represent the size of the quantity which we can control, and Y the other quantity which is varying. Program Listing 12 illustrates this by drawing the shape of a graph of X^2 plotted against X, for a range of values of X which will cover the screen, but leave room for a flashing asterisk on the top line. In this example, SET has been used as the command which prints the graph spot.

Because we use only 128 cells across the screen, and 48 down, graph drawing is a bit limited, but the use of a printer makes it possible to draw more extensive graphs. A graph-plotter is the ultimate luxury. For the beginner, however, a printer is a luxury item, so we won't spend time looking at graph techniques which make use of a printer, except to say that we turn out graphs on their sides when printing. That way, we have all 64 print points available in one direction, and as many as we like in the other.

Most graph programs require you to change a line of the program to enter the equation. Program Listing 13 doesn't. It uses TRS-80 BASIC to create a line of data from the input in line 60. Then it draws the graph using this data. The program is by Ian O'Neill of Ealing, London, England. A complete description of how this program works is a bit beyond us now, but it deserves a description of how it should be used. It depends on changing the expression entered in line 60 into the data statement in line 500. To do this, the computer has to find the address of line 500 by searching through memory for the character @, whose ASCII code is 64. This causes a slight pause, as the computer searches. If, by any chance, line 500 has been zapped, line 20 deals with the problem and reports the bad news. The program then ends, so you can type in a new line 500.

All being well, the title "Graph Plotter" will come up, followed by the instruction "PRINT THE FUNCTION IN TERMS OF X", followed by a query caused by INPUT in line 60. At this point you have to type in the equation to be graphed, in the form of Y = function (X), with no Y^2 or Y^3 or $\sqrt{Y}$

permitted. This is usually straightforward if the equation to be graphed is already in this form, such as $Y = 2X^2 + 3$, which can be entered as: 2*X↑2 + 3; or the equation $Y = \sqrt{X^2 + 2C^2}$, which can be entered as: SQR(X↑2 + 2*C↑2). It becomes harder when the equation has a form like $Y^2 = 2aX + 7$ because the program does not allow you to use Y^2. To enter this equation, you have to rearrange it by taking the square root of each side of it, transforming it to $Y = \sqrt{2aX + 7}$, which is then entered as SQR(2*A*X + 7).

Practically any equation you graph is catered to because the standard BASIC functions, + − * / ↑ SGN, INT, ABS, RND, SQR, LOG, EXP, SIN, COS, TAN, and ATN can be used. The quantity entered into line 60 should be typed so that if it were a line of BASIC in another program, it would run without an error signal. An important point: *No spaces are permitted.* The permitted characters can be seen in line 40.

If you've mistyped your expression, line 90 rejects it. Once the expression is entered correctly, line 100 transfers it into the form of data in line 500.

You are then asked a few more questions which affect the appearance of the graph. The first question is about the equation you have typed. Is it symmetrical about the x-axis? That sounds unfair because you probably want to see the graph to know the answer. A useful hint here is that if the expression uses SQR(X), then you should probably answer YES to the symmetry question, otherwise NO. The reason is that a square root can have a positive or negative value so that there are two possible values of Y for a given value of X. For example, if Y = SQR(X), then for X = 4, Y can be +2 or −2; and for X = 9, Y can be +3 or −3. The symmetry question lets you see both parts of a function like this. If you haven't the faintest idea, just answer YES to the question and if there is only one graph line, run it again, this time answering NO.

The next question is for limits. The computer will print the previous limits of X and Y, if any, so that you can use these again if you like. They must be entered when the questions, "X-AXIS: LOWER LIMIT?" and "X-AXIS: UPPER LIMIT?" appear. You can't expect the computer to know you want one function plotted from 0 to 100 and another from −10 to +10. You'll be asked for a lower limit for Y. You can type AUTO and the computer will calculate its own limits so that the graph will fit the video screen. If you've never seen the shape of the graph, it's wiser to opt for AUTO because you'll see the complete graph, with no chance of points disappearing. You can then try setting lower and upper limits for Y in order to view an expanded section. If you enter a lower limit for Y, you will be prompted for an upper limit.

A flashing bar (cursor) appears to warn you that everything is ready for action. You can now issue a command by pressing any one of the keys D,F,L,N,P, or # without using ENTER. D means display the limits, to tell which X and Y limits are being used. This can be done before or after drawing and will show what limits the computer chose for Y if you opted for AUTO. F causes the equation (function) to be displayed again. If you have a print routine which transfers the screen information to a printer, this is useful. L will allow you to insert new limits. If you want to see more or less of the graph, you don't have to run the program again from start. N selects a new function, so that you can enter another equation. Press P and the equation is plotted in lines 310 and 330. You can look at your work with admiration. The prompt cursor will then flash to remind you that you can choose any of the command letters again.

If you hit SHIFT and 3 together, the program resets itself, ready to run again. If you use the BREAK key at any time, line 500 will be left as a data line, containing the expression you previously entered. You'll have to restore the line or use GOTO200 to get back to the command cursor. If you choose a letter which is not part of the command set, the computer will display a query (line 270) and return to the set.

This program is such a joy to use I had to include it when considering graph drawing. When you finish this series, look back on this one and try to unravel it. You'll learn a lot about programming and how your TRS-80 operates.

"Into the 80s" will continue in Volume 3 of the Encyclopedia.

Program Listing 1

```
10 CLS:PRINT "PLEASE TYPE NUMBERS TO BE ADDED";:INPUT A
   ,B
20 T=A+B
30 CLS:PRINT "THE TOTAL IS ";T: PRINT
40 FOR N=1TO1000:NEXT:GOTO10
```

Program Listing 2

```
10 T=0:N=0
20 INPUT "NUMBER, PLEASE";A
30 IF A=0 THEN 70
40 T=T+A:N=N+1
50 PRINT N ;" ENTERED, TOTAL IS ";T
60 GOTO20
70 PRINT "TOTAL OF ";N;" NUMBERS IS ";T:PRINT"END OF TO
   TALLING RUN":END
```

Program Listing 3

```
10 X=0:T1=0:T2=0:T3=0:T4=0
20 INPUT "FOUR NUMBERS, PLEASE";N1,N2,N3,N4
30 IF N1=0 THEN 70
40 T1=T1+N1:T2=T2+N2:T3=T3+N3:T4=T4+N4:X=X+1
50 CLS:PRINTTAB(20);;X;" SETS ENTERED, TOTALS ARE: ":PR
   INT T1,T2,T3,T4
60 GOTO20
70 PRINT "FINISHED":END
```

Program Listing 4

```
10 INPUT "NUMBER, PLEASE";X:CLS
20 FOR N = 1 TO 12
30 PRINT N; " TIMES ";X;" IS ";N*X
40 NEXT
```

Program Listing 5

```
10 INPUT "WHAT IS THE OPEN-CIRCUIT VOLTAGE";E
20 INPUT "WHAT IS THE AMOUNT OF INTERNAL RESISTANCE";R
30 IF R>E PRINT "VALUE IS RATHER HIGH - PLEASE RECHECK"
   :GOTO20
40 CLS:PRINTTAB(10)"CURRENT";TAB(30)"VOLTAGE":A$="##.##
   "
50 FOR I=.1 TO 1 STEP .1
60 PRINTTAB(10)I;TAB(33)USING A$;E-R*I
70 NEXT
80 END
```

Program Listing 6

```
10 PRINT"THIS PROGRAM CALCULATES THE LENGTH OF THE HYPO
     TENUSE OF":PRINT"A RIGHT-ANGLED TRIANGLE, GIVEN TH
     E OTHER TWO SIDES."
20 INPUT"PLEASE TYPE IN LENGTH OF SIDE A";A
30 INPUT "PLEASE TYPE IN LENGTH OF SIDE B";B
40 C=SQR(A↑2 + B↑2):C=(INT(100*C))/100
50 PRINT "THE HYPOTENUSE LENGTH IS ";C:PRINT"THE PERIME
     TER LENGTH IS ";A+B+C
```

Program Listing 7

```
10 CLS:PRINT"THIS PROGRAM FINDS THE LENGTH OF A SIDE OF
      A TRIANGLE,":PRINT"GIVEN TWO SIDES AND THE ANGLE
     BETWEEN THEM"
20 INPUT"TWO SIDE LENGTHS, PLEASE";B,C
30 INPUT"ANGLE, IN DEGREES,PLEASE";A:IF A/180 =INT(A/18
     0) THEN 70:ELSE IF A=90 THEN X=SQR(B[2+C[2):GOTO50
40 X=SQR(B↑2+C↑2-(2*B*C(COS(A*.0174533))))
50 PRINT "LENGTH OF THIRD SIDE IS ";X; " UNITS LONG"
60 END
70 PRINT "IMPOSSIBLE ANGLE - PLEASE TRY ANOTHER VALUE"
```

Program Listing 8

```
10 CLS
20 SET(63,23):FOR N=1TO100:NEXT
30 RESET(63,23):FOR N = 1TO100:NEXT:GOTO20
```

Program Listing 9

```
10 CLS
20 Y=5
30 FOR N=0TO4:SET(N,Y):NEXT
40 FOR Z=1TO50:NEXT
50 FOR N=0TO127:RESET(N,Y):IF N<122 THEN SET(N+5,Y):FOR
      Z=1TO50:NEXT Z:ELSE FOR Z=1TO50:NEXT Z
60 NEXT N:Y=Y+1:IF Y=48 THEN END ELSE 30
```

Program Listing 10

```
10 CLS:FOR X=1TO 127
20 SET (X,10+10*(SIN(.1745*X))):NEXT
30 PRINT@640,""
```

Program Listing 11

```
10 DIM Y(128):CLS
20 FOR N=ØTO127:Y(N)=5*SIN(N/4):NEXT
30 FOR N=ØTO24:SET(N,7+Y(N)):FOR Z=1TO5Ø:NEXT Z,N
4Ø FOR N=24 TO 127:SET(N,7+Y(N)):RESET(N-24,7+Y(N-24)):
    FOR Z=1TO5Ø:NEXT Z,N
5Ø FOR N=1Ø3 TO 127:RESET (N,7+Y(N)):FOR Z=1TO5Ø:NEXT Z
    ,N
6Ø END
```

Program Listing 12

```
1Ø CLS:Y=47:FOR X=ØTO127
2Ø SET(X,Y-(X↑2)/384)
3Ø NEXT
4Ø PRINT@Ø,"*":FOR Z=1TO5Ø:NEXT:PRINT@Ø," ":FOR Z=1TO5Ø
    :NEXT:GOTO4Ø
```

Program Listing 13

```
1Ø CLEAR 4ØØ:CLS:PRINT@474,"PLEASE WAIT.":DEFINTA-P:DEF
    STRQ-W:ON ERROR GOTO35Ø:FORL=19ØØØTO2ØØØØ:IF PEEK(
    L)=64 THEN 3Ø
2Ø NEXTL:PRINT@471,"NO DUMMY LINE 5ØØ.":END
3Ø FORJ=L TO L+4:IF PEEK(J)=64 THEN NEXT ELSE 2Ø
4Ø DIMV(2Ø),R(2Ø):FOR J=ØTO2Ø:READ V(J),I:R(J) = CHR$(I
    ):NEXT:DATA+,2Ø5,-,2Ø6,*,2Ø7,/,2Ø8,[,2Ø9,(,4Ø,),41
    ,.,46,EXP,224,X,88,SGN,215,INT,216,ABS,217,SQR,221
    ,RND,222,LOG,223,COS,225,SIN,226,TAN,227,ATN,228,E
    ,69
5Ø CLS:PRINT:PRINTTAB(25)"GRAPH PLOTTER":PRINTTAB(24)ST
    RING$(15,62):PRINT:PRINT:PRINT"TYPE THE FUNCTION I
    N TERMS OF X:":PRINT
6Ø INPUT" Y=";T:J=1:U="":IFT="" THEN 5Ø
7Ø IF MID$(T,J,1)>"/" AND MID$(T,J,1)<":" THENU=U+MID$(
    T,J,1):J=J+1:GOTO1ØØ
8Ø FOR I=Ø TO2Ø:IF MID$(T,J,LEN(V(I)))=V(I) THEN U=U+R(
    I):J=J+LEN(V(I)):GOTO1ØØ ELSE NEXT
9Ø PRINT"ILLEGAL REFERENCE:   Y="LEFT$(T,J)"?"RIGHT$(T,
    LEN(T)-J):PRINT"RETYPE FUNCTION.":GOTO6Ø
1ØØ IF J<=LEN(T)THEN7Ø ELSE U="Y"+CHR$(213)+U+":"+CHR$(
    147):FOR J=1 TO LEN(U):POKE L+J-1,ASC(MID$(U,J,1))
    :NEXT:H=Ø:GOSUB5ØØ:IF H=2 THEN 5Ø
11Ø PRINT:INPUT"IS FUNCTION SYMMETRICAL ABOUT X-AXIS (Y
    /N)";S:S=LEFT4(S,1):IF S<>"Y" AND S<>"N" THEN 11Ø
12Ø CLS:PRINT:PRINT"LIMITS":PRINT"======":PRINT:M=Ø
13Ø PRINT"PREVIOUS LIMITS:    X="XL"TO"XU CHR$(8)", Y=
    "YL"TO "YU: PRINT@384,"";:INPUT" X-AXIS:    LOWER
    LIMIT"; XL:INPUT"    UPPER LIMIT";XU:XS=(XU-XL)/12
    8:PRINT:INPUT"Y-AXIS:    LOWER LIMIT" ; Q
14Ø IFQ="AUTO"THEN15ØELSE YL=VAL(Q):INPUT"     UPPER LI
    MIT";YU:YS = (YU-YL)/48:IF XS=Ø OR YS=Ø THEN PRINT
    "     ILLEGAL LIMITS:  AXIS LENGTH ZERO.":FOR I=1TO
     9ØØ:NEXT:GOTO12ØELSE M=1:GOTO19Ø
15Ø M=Ø:X=XL:GOSUB5ØØ:YL=Y:YU=Y:FORX=XL+XS TO XU STEP 3
    *XS:GOSUB5ØØ:IFY>YU THEN YU=Y ELSE IF Y<YL THEN YL
    = Y
16Ø NEXT:IF YU<>YL THEN M=1:Y=YU-YL+.Ø4*Y:YL=YL-.Ø4*Y:Y
```

```
    S=Y/48
170 IF S="Y" AND M=1 THEN YU=ABS(YU+YL+ABS(Y))/2:YL=-YU
    :YS=YU/24
180 PRINT@576,CHR$(30)" Y-AXIS:   AUTO LIMITS  ="YL" TO
    "YU:Q=STR$(YL)
190 AT=16040:IF W = "P" THEN AT=15360
195 PRINT@3,"d-LIMITS:F-FUNCTION:L-NEW LIMITS:N-NEW FUN
    CTION:P-PLOT:#-END PROGRAM"
200 POKE AT,143:FOR I=1 TO 40:W= INKEY$:IF W="" THEN NE
    XT:POKE AT,32:FOR I=1 TO 32:W=INKEY$:IF W="" THEN
    NEXT:GOTO200
210 POKE AT,ASC(W):FORI=1 TO 250:NEXT:IF W="#" THEN 370
220 IF W = "P" THEN 280
230 IF W = "L" THEN 120
240 IF W = "N" THEN 50
250 IF W = "F" THEN PRINT@5,CHR$(30)"Y = "T;:GOTO200
260 IF W = "D" THEN PRINT@5, "LIMITS:  X= "XL" TO "XU C
    HR$(8)",  Y = "YL" TO "YU;:GOTO 200
270 POKE AT,63:FOR I = 1 TO 300:NEXT:GOTO200
280 IF M=0 THEN CLS:PRINT:PRINT"ILLEGAL LIMITS:   AXIS
    LENGTH ZERO.":FOR I=1 TO 900:NEXT:GOTO120 ELSE CLS
290 A=INT(.5-XL/XS):IF 0<A AND A<=127 THEN FOR I=0 TO 4
    7:SET(A,I):NEXT
300 A=47 - INT(.5-YL/YS):IF 0<A AND A<=47 THEN FOR I =
    0 TO 127:SET(I,A):NEXT
310 FOR N=0 TO 127:X=XL + N*XS:H=0:GOSUB500:IF H=1 THEN
     340
320 P=47-INT((Y-YL)/YS+.5):IF P>=0 AND P<=47 THEN SET (
    N,P)
330 IF S="Y" THEN P=47-INT(.5-(Y+YL)/YS):IF P>=0 AND P<
    =47 THEN SET(N,P)
340 NEXT:GOTO190
350 IF ERR=2 OR ERR=40 THEN CLS:PRINT"    Y= "T:PRINT:P
    RINT "ERROR IN FUNCTION. RETYPE CORRECTLY.":FOR I=
    0 TO 2000:NEXT:H=2:RESUME NEXT
360 H=1:RESUME NEXT
370 FOR I=L TO L+10:POKE I,64:NEXT:CLS:PRINT "RUN COMPL
    ETE.":END
500 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
510 RETURN
```

UTILITY

EDTASM for Model III

by Richard Koch

Under distressing circumstances, I recently converted from the TRS-80 Model I computer to the Model III. My Model I was stolen. The burglar conveniently left all my software behind, and I have been converting it to Model III format. Among the indispensible programs left was Radio Shack's Editor/Assembler. Unhappily, this old program does not work on the Model III, and at the time I wrote this article, Radio Shack had not released a Model III version.

Those who know something about the internal structure of EDTASM may be surprised that it will not work on the Model III. The original program was written for both Level I and Level II BASIC. Since the ROMs for these BASICs differ, the Editor/Assembler never calls ROM routines. Instead, it contains a duplicate version of the keyboard, video display, cassette, and printer routines. You might think that EDTASM would thus have been immune from conversion problems, but in fact almost everything goes wrong—the screen fills with garbage, the keyboard has a glitch, the cassette fails, and the printer remains silent.

Luckily, it is easy to change EDTASM so it will work on the Model III. I'll explain how in a minute. My explanation will be very mechanical. Later on I'll explain what the method actually does. Those who are interested can thus learn the details of changes made in the Model I by Radio Shack when they produced the Model III.

How to Convert

First, I'll explain the conversion process for those without disk. Program Listing 1 contains a short BASIC program. Type this program into the computer, and save it on cassette. Whenever you want to run the Editor/Assembler, load this program and run it. Then load the EDTASM tape as you ordinarily would. Do not respond to the second system prompt with the usual /. Instead respond with /31000. Immediately, the EDTASM program will run.

As a matter of fact, you'll notice a few pleasing changes in EDTASM. The screen will display lowercase characters. To obtain them from the keyboard, simply press SHIFT-ZERO as usual on the Model III. The keyboard repeat feature will also work.

The converted version of EDTASM outputs to cassette at low baud rate, so system tapes produced by EDTASM must be loaded at low rate.

Conversion for Disk Owners

If you use disks, the conversion process must be changed a little. As a bonus for the extra steps, the process yields a copy of EDTASM on disk. Type the program from Program Listing 1 into the computer. Add the program fragment in Program Listing 2 (this fragment will overwrite line 280 of the first program). Change line 130 of the resulting program to:

```
130 IF A= -2 THEN END ELSE IF A= -1 THEN I= -30208: GOTO 120
```

Run the program. Next, hold the BREAK key down and press RESET. You will enter non-disk BASIC—respond to the cassette prompt with L and to the memory prompt with ENTER. Load the EDTASM tape. Do not respond to the second system prompt with the usual /. Instead respond /31000. You will return to DOS mode. Type DUMP EDTASM/CMD (START = 7000, END = 8A19, TRA = 8A00). Whenever you wish to run the assembler, just type EDTASM.

Explanation

It is fun to load the original EDTASM tape and see what happens. The screen fills with garbage. Inspection shows that this garbage consists of ordinary numbers and of the new characters Radio Shack has provided for ASCII codes between zero and 31. Moreover, the garbage occurs precisely where EDTASM should have printed its start-up message.

The explanation is easy enough. ASCII codes between 40H and 5FH correspond to capital letters; ASCII codes between 60H and 7FH give lowercase letters. On the Model I, numbers with ASCII codes between 00H and 1FH also give capital letters when POKEd to the video memory. The EDTASM contains several lines of code which subtract 40H from numbers between 40H and 5FH and subtract 60H from numbers between 60H and 7FH before POKEing these numbers to video memory. We can fix this display routine by simply bypassing these pieces of subtraction code. That is very simple—change the code at location 447BH to 18H (the machine code for JR).

Once this single byte of code is corrected, EDTASM begins to look familiar. The keyboard works correctly *except* that the right-hand shift key no longer works.

Incidentally, this same shift key difficulty occurs when Model I SCRIPSIT is run on the Model III. The difficulty is caused by a small wiring change Radio Shack made in the keyboard. Location 3880H contains the status of the shift keys for both models. On the Model I, it contains zero when neither key is pressed and one otherwise. On the Model III, it contains zero when neither is pressed, one when the left shift is pressed, two when the right is pressed, and three when both are pressed. (An admirable change, Radio Shack.)

Ironically, it is easier to write machine-language code testing for zero than to write code testing for one. If EDTASM had been coded that way, there would have been no difficulty, but EDTASM tests for one. Rather than fix the EDTASM keyboard routine, I suggest replacing the entire routine with the ROM keyboard routine which starts at 2BH. That way, EDTASM gains lowercase and repeating keyboard capability. It is easy to switch the keyboard routines—simply change the code beginning at 4605H to CALL 2BH, RET.

To be honest, it is somewhat risky to use ROM routines with EDTASM because ROM routines sometimes refer to memory locations in RAM. For instance, the keyboard routine looks at location 4019H to see if it should use lowercase. EDTASM uses many locations in low RAM for its own purposes, so some ROM routines fail when used with the assembler. Luckily, the keyboard ROM routine works fine.

EDTASM contains a routine which calls the keyboard; if the keyboard returns a lowercase letter, this routine rejects the letter and calls the keyboard again. We want to accept lowercase letters now, so location 46FAH must be changed to 7FH.

Fixing the printer is easy. On the Model I, printing is done using location 37E8H. The status of the printer is determined by reading this location, and letters are printed by POKEing them to the location. Model III printing is handled in a different way. Location 37E8H still contains the printer status, but letters are printed by placing them in the output port 0F8H. Thus the code LD(37E8H),A (which occurs at locations 45CA and 45D8) must be changed to OUT(0F8H),A.

The cassette routines must be changed. Again, we will simply use ROM routines. Six routines are involved. The routine to read the leader and sync byte occurs at 43B8; we insert here the code CALL 296H, RET. The routine to write the leader and sync byte starts at 43A9, and we change it to CALL 287H, RET. The routine to input a byte is at 435D and must be replaced by CALL 235H, RET. The routine to output a byte is at 4389 and is replaced by CALL 264H, RET. The routine to turn off the cassette recorder at 4337 is changed to CALL 1F8H, RET. Finally, the routine at 4354 to print a flashing asterisk can be eliminated by inserting a RET.

Postscript

If you run across a Model I computer with serial number B-004539, it's mine.

Program Listing 1

```
100 POKE 16913,0
110 I = 31000
120 READ A
130 IF A = - 1
     THEN
      END
140 POKE I,A
150 I = I + 1
160 GOTO 120
170 DATA 62,24,50,123,68,62,205,50,05,70,62,43,50,06,70
180 DATA 62,00,50,07,70,62,201,50,08,70,62,127,50,74,70
190 DATA 62,211,50,202,69,50,216,69,62,248,50,203,69
200 DATA 50,217,69,62,00,50,204,69,50,218,69,62,201
210 DATA 50,140,67,50,58,67,50,96,67,50,172,67,50,187,67
220 DATA 50,84,67,62,205,50,137,67,50,55,67,50,93,67
230 DATA 50,169,67,50,184,67,62,100,50,138,67,62,02
240 DATA 50,139,67,62,248,50,56,67,62,01,50,57,67
250 DATA 62,53,50,94,67,62,02,50,95,67,62,135
260 DATA 50,170,67,62,02,50,171,67,62,150,50,185,67
270 DATA 62,02,50,186,67
280 DATA 195,138,70,-1
```

Program Listing 2

```
280 DATA 195,25,138,-1
290 DATA 243,33,00,112,17,00,67,01,00,26,237,176,62
300 DATA 201,50,12,64,62,00,50,17,66,195,138,70
310 DATA 33,00,67,17,00,112,01,00,26,237,176
320 DATA 195,00,00,-2
```

Put Some Flash into Your Menus

by John Acres

If you have a disk-based TRS-80 Model I or III, this program can help dress up dull program menus. Most menu-driven systems display a list of options on the screen with a number beside each. To execute one, you type in the number of the desired function and press ENTER. This routine does precisely the same thing—except that the option selected flashes on the screen. Press ENTER and that option is executed. Press a different option number and the new one flashes. Keystrokes that do not represent a valid option are ignored.

A Useful Idea

A flashing menu has practical advantages. I write business software for the TRS-80. Most of my programs are used by secretaries and salespeople who know and care nothing about computers. They have a job to do and want to complete it as soon as possible. Many of these people have difficulty in relating a written description to an option number. The flashing cursor helps. All they have to do is press the key they think will work. If the desired selection flashes, they can press ENTER and perform the task at hand. If no flashing is visible, or the improper item flashes, they know something is wrong and can try again.

Program Setup

The Program Listing lists the flashing menu program. The particular menu shown is for a small business accounting package and is typical of program menus. Central to the operation of this routine are LSET and VARPTR, two of the least utilized commands in Disk BASIC's vocabulary. Through them, a BASIC string is set to video memory. Menu options displayed on the screen are held within the string. By periodically filling the string with blanks and then restoring its original contents, menu options appear to flash.

Line 1500 of the Program Listing initializes B$ to a single character, then uses VARPTR to put the B$ descriptor block address (Figure 2) into B. Next, the length of B$ is forcibly changed by POKEing a new value into the descriptor block. That length is calculated by subtracting the left margin offset (XX) from the video line length (LE). Besides length, descriptor blocks keep the address at which variable information can be found. Lines 1900-2300 calculate address values for specific video screen lines and convert them to LSB/MSB (Least Significant Byte/Most Significant Byte) format for later use. Because each MSB value can represent four lines of video display, four values are calculated and assigned to arrays MS(I) and LS(I).

Lines 2500-3300 display the menu options. XY represents the line number on which the first option will be shown. XX is the left-hand indentation. Using these variables makes it easy to center the options list on the display. Line 3500 executes a keyboard scan, INKEY$, and line 3600 checks the results for a valid input. This menu uses 1-8 to select options. Any other value, including a null (which indicates no key was pressed), returns to line 3500 for another scan. When a valid entry is detected, line 3700 converts the string value to a number which line 3800 uses to branch on. Depending on which number is pressed, one of the lines in 4000-4700 is executed.

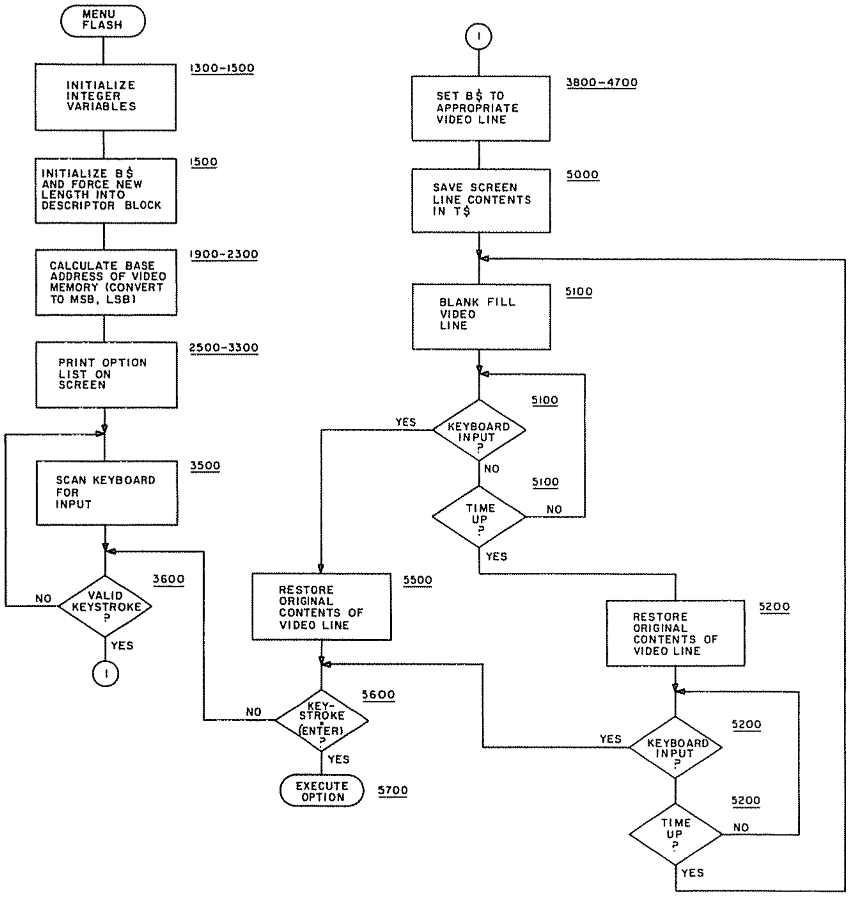

MENU FLASH
INITIALIZE INTEGER VARIABLES
1300-1500
INITIALIZE B$ AND FORCE NEW LENGTH INTO DESCRIPTOR BLOCK
1500
CALCULATE BASE ADDRESS OF VIDEO MEMORY (CONVERT TO MSB, LSB)
1900-2300
PRINT OPTION LIST ON SCREEN
2500-3300
SCAN KEYBOARD FOR INPUT
3500
VALID KEYSTROKE ?
3600
NO
YES
1
SET B$ TO APPROPRIATE VIDEO LINE
3800-4700
SAVE SCREEN LINE CONTENTS IN T$
5000
BLANK FILL VIDEO LINE
5100
KEYBOARD INPUT ?
5100
TIME UP ?
5100
RESTORE ORIGINAL CONTENTS OF VIDEO LINE
5500
KEY-STROKE (ENTER) ?
5600
EXECUTE OPTION
5700
RESTORE ORIGINAL CONTENTS OF VIDEO LINE
5200
KEYBOARD INPUT ?
5200
TIME UP ?
5200

Figure 1. *Program flow chart. Underlined numbers are the line numbers that implement each function.*

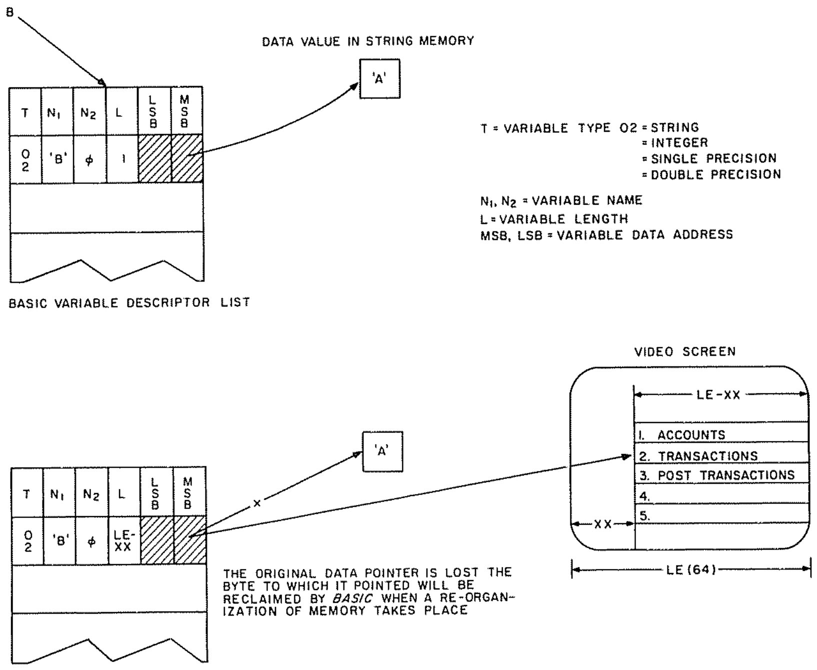

B
DATA VALUE IN STRING MEMORY
'A'
T
N1
N2
L
LSB
MSB
02
'B'
ϕ
1
T = VARIABLE TYPE 02 = STRING
= INTEGER
= SINGLE PRECISION
= DOUBLE PRECISION
N1, N2 = VARIABLE NAME
L = VARIABLE LENGTH
MSB, LSB = VARIABLE DATA ADDRESS
BASIC VARIABLE DESCRIPTOR LIST
VIDEO SCREEN
LE-XX
1. ACCOUNTS
2. TRANSACTIONS
3. POST TRANSACTIONS
4.
5.
XX
LE (64)
LE-XX
THE ORIGINAL DATA POINTER IS LOST THE BYTE TO WHICH IT POINTED WILL BE RECLAIMED BY BASIC WHEN A RE-ORGANIZATION OF MEMORY TAKES PLACE

Figure 2. *Reassignment of string variables to video screen*

Flashing the Option

The display address of the option selected is POKEd into the B$ descriptor block and B$ becomes a part of display memory. Line 5000 transfers that portion of the display to T$ for safekeeping while line 5100 blanks the screen area. LSET and its companion RSET are intended for moving string information into a disk buffer. However, they work just as well in moving information to other strings. Since LSET "blank fills," only a single character is assigned by the program. After the video area is blanked, line 5100 executes a FOR-NEXT loop as it scans the keyboard for another input. If none is found, line 5200 is executed. Here, the original screen data saved in T$ is restored and the keyboard is again scanned. If no inputs are found, control passes back to line 5100. This alternate blanking/restoration generates a flashing effect whose rate can be controlled by adjusting the FOR-NEXT loops in 5100-5200. Line 5100 controls the blank or "off" time while 5200 is the "on" time. These values can be altered to match your own tastes.

Final Selection

While the flashing process is going on, the keyboard is constantly scanned. When a keystroke is detected, line 5500 restores the original screen contents

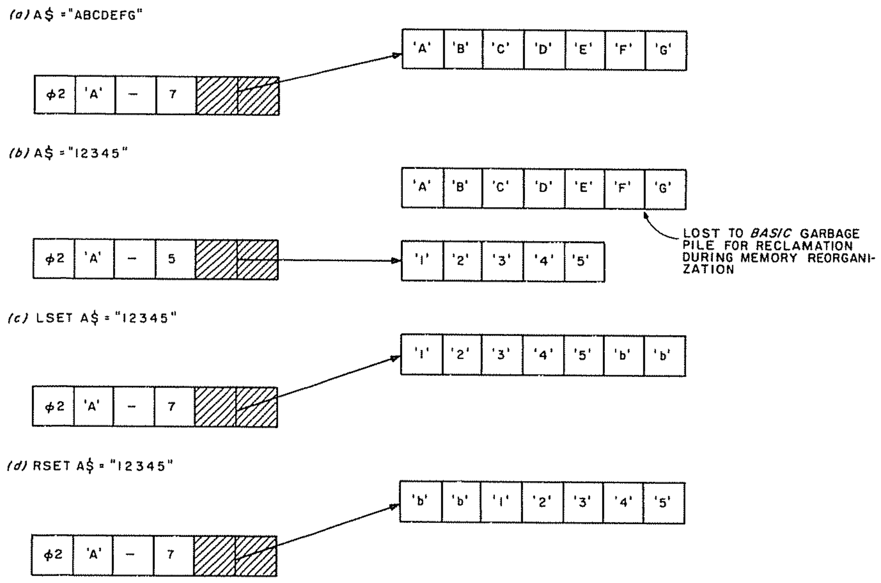

(a) A$ ="ABCDEFG"
'A' 'B' 'C' 'D' 'E' 'F' 'G'
φ2 'A' – 7
(b) A$ ="12345"
'A' 'B' 'C' 'D' 'E' 'F' 'G'
LOST TO BASIC GARBAGE PILE FOR RECLAMATION DURING MEMORY REORGANIZATION
φ2 'A' – 5
'1' '2' '3' '4' '5'
(c) LSET A$ = "12345"
'1' '2' '3' '4' '5' 'b' 'b'
φ2 'A' – 7
(d) RSET A$ = "12345"
'b' 'b' '1' '2' '3' '4' '5'
φ2 'A' – 7

Figure 3. *BASIC assignments: LSET, RSET and =. (a) initializes A$ to a seven-byte value "ABCDEFG". When (b) is executed, the original data block is lost and the A$ descriptor block is set to a new data block. Executing (c) instead of (b) left justifies the new data in the original data block. Excess space is blank-filled. (d) is the same as (c), except data is right justified.*

and line 5600 checks to see if the keystroke is ENTER. If so, line 5700 is executed and branches to the program line for the selected option. If the keystroke isn't ENTER, line 3600 is executed again. If a valid selection key is pressed, that option area of the screen begins to flash. If an invalid key is pressed, flashing stops as lines 3500-3600 monitor the keyboard for a new input value.

I hope this program is useful to you. To learn more about LSET and RSET, consult your Disk BASIC manual. To find out about VARPTR, PEEK, and POKE, check the Level II manual.

Program Listing. *Flashing cursor demonstration*

```
1000 :
     ' FLASHING CURSOR DEMONSTRATION MENUFLSH/BAS. 11/20/80
1100 :
     ' COPYRIGHT 1980 JOHN ACRES
1200 :
     '
1300 CLEAR 500:
     CLS :
     DEFINT A - Z
1400 LE = 64:
     CRT = 15360:
     XY = 3:
     XX = 20:
     :
     ' VIDEO MEMORY, LINE LENGTH, 1ST MENU LINE + OFFSET
1500 B$ = " ":
     B = VARPTR(B$):
     POKE B,LE - XX:
     :
     ' INITIALIZE FLASHING STRING FIELD
1600 PRINT @13,"ACR CONSULTANTS FLASHING MENU DEMO":
     :
     ' MENU HEADING
1700 :
     '
1900 X = CRT + XY * 64 + XX:
     MS = X / 256:
     LS = X - MS * 256:
     :
     ' CONVERT TO LSB/MSB BASE
2000 FOR I = 0 TO 3
2100  IF LS > 255
       THEN
        LS = LS - 256:
        MS = MS + 1:
        :
        ' KEEP LS WITHIN 0-255. UPDATE MS TOO
2200  LS(I) = LS:
      LS = LS + LE:
      MS(I) = MS:
      :
      ' NEXT LINE LSB AND MSB
2300  NEXT
2400 :
     '
2500 PRINT @(XY * LE) + XX,"1. ACCOUNTS"
2600 PRINT @(XY + 1) * LE + XX,"2. TRANSACTIONS"
2700 PRINT @(XY + 2) * LE + XX,"3. POST TRANSACTIONS (AND WRITE CHECK
     S)"
2800 PRINT @(XY + 3) * LE + XX,"4. RECONCILE CHECKS"
2900 PRINT @(XY + 4) * LE + XX,"5. PRINT INCOME STATEMENT"
3000 PRINT @(XY + 5) * LE + XX,"6. BACKUP DATA DISKETTE"
3100 PRINT @(XY + 6) * LE + XX,"7. RECALCULATE BALANCE"
3200 PRINT @(XY + 7) * LE + XX,"8. CREATE NEW SYSTEM"
3300 PRINT @(XY + 9) * LE + 10,"PRESS THE NUMBER OF YOUR SELECTION AN
     D <ENTER>."
3400 :
     '
3500 A$ = INKEY$:
     :
     ' GET OPTION
3600 IF A$ < "1" OR A$ > "8"
      THEN
       GOTO 3500:
       :
       ' IGNORE INVALID ENTRIES
3700 A = ASC(A$) - 48:
```

Program continued

```
     :
     ' CONVERT STRING$ INPUT TO NUMBER
3800 ON A GOTO 4000,4100,4200,4300,4400,4500,4600,4700
3900 :
     '
4000 POKE B + 1,LS(0):
     POKE B + 2,MS(0):
     GOTO 5000:
     :
     ' OPTION 1
4100 POKE B + 1,LS(1):
     POKE B + 2,MS(1):
     GOTO 5000:
     :
     '         2
4200 POKE B + 1,LS(2):
     POKE B + 2,MS(2):
     GOTO 5000:
     :
     '         3
4300 POKE B + 1,LS(3):
     POKE B + 2,MS(3):
     GOTO 5000:
     :
     '         4
4400 POKE B + 1,LS(0):
     POKE B + 2,MS(0) + 1:
     GOTO 5000:
     :
     '       5
4500 POKE B + 1,LS(1):
     POKE B + 2,MS(1) + 1:
     GOTO 5000:
     :
     '       6
4600 POKE B + 1,LS(2):
     POKE B + 2,MS(2) + 1:
     GOTO 5000:
     :
     '       7
4700 POKE B + 1,LS(3):
     POKE B + 2,MS(3) + 1:
     GOTO 5000:
     :
     '       8
4800 :
     '
5000 T$ = B$:
     :
     ' SAVE CONTENTS OF LINE TO BE FLASHED
5100 LSET B$ = " ":
     FOR I = 0 TO 12:
      A$ = INKEY$:
      IF A$ < > ""
       THEN
        GOTO 5500 :
       ELSE
        NEXT :
       :
       ' FLASH OFF
5200 LSET B$ = T$:
     FOR I = 0 TO 24:
      A$ = INKEY$:
      IF A$ < > ""
       THEN
        GOTO 5500 :
       ELSE
        NEXT :
       :
       ' FLASH ON
```

```
5300 GOTO 5100:
     :
     ' KEEP FLASHING WHILE WAITING FOR KEYBOARD ENTRY
5400 :
     '
5500 LSET B$ = T$:
     :
     ' RESTORE ORIGINAL LINE CONTENTS
5600 IF A$ < > CHR$(13)
      THEN
       GOTO 3600:
       :
       ' IF KEYSTROKE WASN'T ENTER, START OVER
5700 ON DA GOTO 6000,6100,6200,6300,6400,6500,6600,6700:
     :
     ' EXECUTE OPTION
5800 :
     '
5900 :
     ' OPTION LIST
6000 GOTO 3500:
      '1
6100 GOTO 3500:
      '2
6200 GOTO 3500:
      '3
6300 GOTO 3500:
      '4
6400 GOTO 3600:
      '5
6500 GOTO 3500:
      '6
6600 GOTO 3500:
      '7
6700 GOTO 3500:
      '8
6800 '
6900 END
```

FILEX: A Communication Package For File Exchange

by Larry Rudner

The RS-232 interface board provides Radio Shack's TRS-80 with serial input output (I/O) for connecting the machine to peripheral devices such as a printer or a modem. FILEX provides software for using this I/O hardware for telephone exchange of ASCII files between two TRS-80 disk systems. Minimum configuration requires two machines, each with 16K RAM, one disk drive, RS-232 interface, and a modem (one set to originate, the other to answer).

FILEX has four modes of operation: (1) diskette to RS-232, (2) keyboard to RS-232, (3) RS-232 to display, and (4) RS-232 to diskette. Modes one and four are used to send and receive files to and from diskette. Modes two and three are for inter-user communication.

Programming Considerations

There are several ways to accomplish telephone file transfer from one TRS-80 to another. Using Radio Shack's Host program, one system can be controlled by a second remote system. The controlled system can be placed into an edit-input mode, with the characters being sent by the first machine, and a file created on the controlled machine. An alternate method involves the receiving station sequentially loading characters into memory and then, using the TAPEDISK routine, transferring the characters to disk.

The approach taken in FILEX is to load the characters directly, one sector (256 bytes) at a time, onto diskette (see Program Listing 1). This approach has several advantages: Little operator activity is required, programming is relatively simple, and the program can be written almost entirely in BASIC. Since the program uses independent string sending and string receiving routines, it can be modified readily for custom requirements. I use a modified version to control a DEC PDP-10 for automatic downloading of several files at a time.

The TRS-80 requires relatively large amounts of time to extract data from memory via the PEEK command or data placed onto diskette via a PRINT n command. The PRINT n command first places data into a 256-byte buffer and when the buffer is full, writes the sector on diskette. While the computer is performing a PEEK or a PRINT, it can't do anything else. Any characters sent during this time would be lost. I solved this problem in FILEX by using "handshaking" between machines. In handshaking, the receiving station issues control characters which tell the sending station when to wait and when to send characters.

The other major consideration in developing FILEX was program operation after reaching the end of file (EOF). If the receiving station does not know when to expect the EOF, it will process the last line of a file exactly as it does every other line. After a line is received, it is processed and then the program loops back to receive the next line. In the case of the last line, there is no more input and the program would loop infinitely waiting for another, albeit nonexistent, line. The problem is solved in FILEX by using another handshaking character. A different control character is issued by the transmitting station once the entire file has been sent. Upon receiving this special character, the receiving station writes whatever is in the buffer to diskette, closes the file; then jumps to command input mode (line 150).

Program Logic

Initialization (lines 560 to 780): The Radio Shack RS-232 system uses ports 232 through 235 for various functions related to I/O. This application uses: the master reset, baud rate select, the UART control, the UART status, the data in, and the data out functions. Modem sense and configuration switch sense functions are ignored.

An OUT of any value to port 232 performs the master reset of the RS-232 system and is accomplished in the initialization subroutine on line 580. The UART control is accomplished by an OUT to port 234. The value on line 580, 167 decimal or 10100111 binary, sets even parity, a seven-bit word length, one stop bit, and parity enable (see page 17 of the Radio Shack *RS-232 Interface Manual*).

Port 233 is used for setting baud speed. In lines 600 to 660, a user-specified speed is compared to the possible speeds loaded into array BD. Once a match is found, baud is set by an OUT of the associated value in array BT. While 300 baud is maximum with an acoustical coupler, higher speeds are possible with a special modem or with a direct connection, such as with a large computer or a RS-232 tape system. The initialization section also enters the USR program used to receive characters (lines 700–760) and clears the character storage buffer (line 770).

Character Transmission (lines 250–350; 370–430; 800–850): The subroutine used to transmit a string of characters is located in lines 800 to 850. One character is extracted at a time from string AS$ and its ASCII value placed into variable CH. Port 234 is then input in a test to determine whether the computer is ready to send a character. An AND of B6 to the received character provides a test of the sixth bit. This value is 1 when the machine is ready to send and 0 when it is not. This occurs, for example, when the previous character is still in the process of being sent. Once the machine is ready, the value in CH is OUT to port 235 and the RS-232 line. The process is then iterated sequentially for each character in the string.

In diskette to RS-232 mode (lines 250–350), characters are read one line at a time from the user specified file, and appended with a carriage return. After the entire string is sent, the sending machine waits for the receiving machine to issue a control-S (lines 280–300). This character, an ASCII 19 decimal, initiates a handshake and puts the sending station into a loop (line 310). This activity allows the necessary time for the receiving station to transfer the string either between buffers or from buffer to diskette. When the receiving station is again ready to receive characters, it issues a control-Q, the sending station exits the loop, and the next string is gathered and sent. After the entire file is sent, an EOF marker is issued (lines 340–350) to tell the receiving station to close the newly created file.

In keyboard to RS-232 mode (lines 370–430), a string is input from the keyboard and sent to the RS-232 line. After each string is sent, the program cycles and waits for a new string until you type a control-Z (shift down-arrow) as the first character. A control-Z is then sent and both stations return to command input mode.

Character Reception (lines 450–480; 510–540; 870–910): The subroutine in lines 870–910 receives a string of characters. String AS$ is zeroed and then the USR machine-language routine is called. As characters are received, they are placed into memory starting at location 30450. The characters are then taken from the buffer and placed into string AS$ (lines 890–900).

Program Listing 2 shows the Z-80 assembly language mnemonics for the USR routine. When called, the routine issues a control-Q (ASCII 17D) to indicate readiness to receive characters. The keyboard and the input port are then alternately scanned. Pressing any key aborts the program and returns to command input mode. As characters are received, they are placed into memory starting at location 30450 until a carriage return is detected. A control-S is then issued to tell the sending station to wait. The location of the last character is returned to the caller and placed into variable DD. The wait permits the receiving station to transfer the characters from memory to string AS$ (lines 890–900) and then to the display (lines 470 or 520) or the write buffer (line 520). Once the write buffer is full, or the file closed, text is written to the diskette.

The heart of the reception subroutine is written in assembly language in order to overcome the inherent slowness of TRS-80 BASIC. The routine shown in Program Listing 3 uses the same logic as the subroutine listed in lines 880–900 and in Program Listing 2. However, starting above 110 baud, the machine requires too much time to interpret this routine and will lose characters.

Summary

FILEX provides software for the exchange of files between two TRS-80 microprocessors. Because the program uses handshaking and an end-of-file marker, a minimum of operator intervention is required. To use FILEX, both stations load and run FILEX in BASIC. When the program enters command input mode, one station selects mode four and becomes the receiving station, while the other selects mode one and becomes the sending station. The receiving station types in a name for the new file and the machine is then ready to receive characters. The sending station types in the name of the ASCII file to be sent and that machine immediately starts sending lines of data. At the end of each line, both machines pause while the receiving station processes the string. Once the end of the file is reached, the file automatically closes and the stations return to command input mode.

Note that this version of FILEX is limited to ASCII files such as those created using Microsoft's Edit-80 program, data files, and BASIC files created with a SAVE"filename/ext", A command.

Program Listing 1. *FILEX/BAS*

```
100 :
    ' FILEX-A TRS-800 COMMUNICATIONS PACKAGE FOR FILE EXCHANGE
110 :
    ' VERSION 3.2
120 :
    ' LAR RUDNER
130 :
    '
140 CLEAR 500:
    GOSUB 560
150 CLS :
    PRINT "FILEX VER. 3.2":
    PRINT :
    PRINT :
    PRINT
160 PRINT "TYPE 1) TRS-80 DISKETTE TO RS-232"
170 PRINT "     2) KEYBOARD TO RS-232"
180 PRINT "     3) RS-232 TO DISPLAY"
190 PRINT "     4) RS-232 TO DISKETTE"
200 PRINT "     5) QUIT":
    INPUT WH
210 IF WH < 1 OR WH > 5 PRINT "?REDO";:
    GOTO 150
220 OUT 235,0:
    ON WH GOTO 240,370,450,500,930
230 :
    '
240 :
    ' DISKETTE TO MODEM
250 INPUT "SOURCE FILESPEC";FS$
260 OPEN "I",1,FS$
270 LINE INPUT #1,AS$:
    AS$ = AS$ + CS$:
    GOSUB 810
280 FOR I = 1 TO 100:
     CH = INP(235):
     IF CH = 19
      THEN
       310
290  IF CH = 26
      THEN
       330
300  NEXT I:
    GOTO 320
310 IF INP(235) < > 17
     THEN
      310
320 IF EOF (1) = 0
     THEN
      270
330 CLOSE 1
340 IF ( INP(235) AND B6) = 0 GOTO 340
350 OUT 235,26:
    GOTO 150
360 :
    '
370 CLS :
    PRINT :
    PRINT "TYPE MESSAGE (END WITH SHIFT DOWN ARROW)"
380 PRINT STRING$(41,"-"):
    PRINT
390 FC$ = INKEY$:
    IF INP(235) = 26
     THEN
      FC$ = CHR$(26)
400 IF FC$ = ""
     THEN
      390
```

```
41Ø IF ASC(FC$) = 26
     THEN
      34Ø :
     ELSE
      PRINT FC$
42Ø LINE INPUT AS$:
    AS$ = FC$ + AS$ + CR$
43Ø GOSUB 81Ø:
    GOTO 39Ø
44Ø :
    '
45Ø CLS :
    PRINT :
    PRINT "RS-232 TO DISPLAY":
    PRINT STRING$(16,"-")
46Ø PRINT
47Ø GOSUB 88Ø:
    PRINT AS$
48Ø IF INSTR (AS$,CZ$) = Ø
     THEN
      47Ø :
     ELSE
      34Ø
49Ø :
    '
5ØØ :
    ' RECIEVE CHARACTERS RS-232 TO DISKETTE
51Ø INPUT "DESTINATION FILESPEC";FS$:
    OPEN "O",2,FS$
52Ø GOSUB 88Ø:
    PRINT AS$:
    PRINT #2,AS$
53Ø IF INSTR (AS$,CZ$) = Ø
     THEN
      52Ø :
     ELSE
      CLOSE :
      GOTO 34Ø
54Ø PRINT #2,AS$:
    PRINT AS$:
    GOTO 52Ø
55Ø :
    '
56Ø :
    ' INITIALIZATION
57Ø DIM BD(9),BT(9):
    B6 = 2 [ 6:
    CR$ = CHR$(13):
    CZ$ = CHR$(26)
58Ø OUT 232,1:
    OUT 234,167
59Ø :
    ' BAUD SET
6ØØ FOR I = 1 TO 9:
     READ BD(I):
     READ BT(I):
     NEXT I
61Ø DATA 11Ø,34,15Ø,68,3ØØ,85,6ØØ,1Ø2,12ØØ,119
62Ø DATA 18ØØ,136,24ØØ,17Ø,48ØØ,2Ø4,96ØØ,238
63Ø INPUT "TTY SPEED";SP
64Ø FOR I = 1 TO 9:
     IF SP = BD(I)
      THEN
       67Ø
65Ø  NEXT I
66Ø PRINT "?REDO";:
    GOTO 63Ø
67Ø OUT 233,BT(I)
68Ø :
    ' SET UP USR PROGRAM
```

Program continued

```
690 DEF USR0 = 30390
700 FOR I = 30390 TO 30449:
     READ DD:
     POKE I,DD:
     NEXT I
710 DATA 33,242,118,219,234,203,119,40,250,62,17
720 DATA 211,235,205,43,0,183,40,06,62,26,211,235
730 DATA 24,14,219,234,203,127,40,238,219,235
740 DATA 230,127,254,13,40,8,119,254,26,40,13
750 DATA 35,24,222,219,234
760 DATA 203,119,40,250,62,19,211,235,195,154,10
770 FOR I = 30450 TO 30706:
     POKE I,0:
     NEXT I
780 RETURN
790 :
    '
800 :
    ' SUBROUTINE TO SEND STRING AS$
810 FOR I = 1 TO LEN(AS$)
820  CCH = ASC( MID$(AS$,I,1)):
     IF WH = 1 PRINT CHR$(CH);
830  IF ( INP(234) AND B6) = 0 GOTO 830
840  OUT 235,CH:
     NEXT I
850 RETURN
860 :
    '
870 :
    ' RECIEVE A STRING OF CHARACTERS
880 AS$ = "":
    DD = USR(DM)
890 FOR I = 30450 TO DD
900  AS$ = AS$ + CHR$( PEEK(I)):
     POKE I,0:
     NEXT I
910 RETURN
920 :
    '
930 END
```

Program Listing 2. *A Z-80 assembly-language routine to receive a string of characters*

```
00100 BEG     LD      HL,30450D       ;BUFFER START
00110 OUT1    IN      A,(234D)        ;READY TO SEND?
00120         BIT     6,A
00130         JR      Z,OUT1
00140         LD      A,17D           ;HANDSHAKE GO
00150         OUT     (235D),A
00160 IN1     CALL    002BH           ;KEYBOARD SCAN
00170         OR      A
00180         JR      Z,GETCH         ;INTERPRET IF HIT
00190         LD      A,26D
00200         OUT     (235D),A        ;SEND EOF
00210         JR      CTRLZ
00220 GETCH   IN      A,(233D)        ;READY TO RECIEVE?
00230         BIT     7,A
00240         JR      Z,IN1
00250         IN      A,(235D)        ;GET CHARACTER
00260         AND     7FH             ;STRIP PARITY BIT
00270         CP      13D             ;CR?
00280         JR      Z,OUT2
00290 CTRLZ   LD      (HL),A          ;FILL BUFFER
00300         CP      26D             ;EOF?
00310         JR      Z,RETRN         ;QUIT IF SO
```

```
00320          INC     HL
00330          JR      IN1              ;GET NEW CHARACTER
00340 OUT2     IN      A,(234D)
00350          BIT     6,A
00360          JR      Z,OUT2
00370          LD      A,19D            ;HANDSHAKE WAIT
00380          OUT     (235D),A
00390 RETRN    JP      0A9AH            ;BASIC
00400          END
```

Program Listing 3. *An alternate subroutine*

```
880 AS$ = "":
    B7 = 2 ↑ 7
890 IF ( INP(234) AND B6) = 0
     THEN
      890
900 OUT 235,17
910 B$ = INKEY$:
    IF B$ < > ""
     THEN
      930
920 CH = 26:
    OUT 235,CH:
    GOTO 950
930 IF ( INP(234) AND B7) = 0
     THEN
      910
940 CH = INP(235) AND 127:
    IF CH = 13
     THEN
      970
950 AS$ = AS$ + CHR$(CH):
    IF CH = 26
     THEN
      990
960 GOTO 910
970 IF ( INP(234) AND B6) = 0
     THEN
      970
980 OUT 235,19
990 RETURN
```

APPENDIX

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P*.

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200-450 nanoseconds for TRS-80 RAM.

accumulator—traditionally the register where arithmetic (the accumulation of numbers) takes place.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—adjustment of hardware to achieve proper transfer of data. In the TRS-80 this usually applies to cassette heads and disk drives.

alphanumerics—refers to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

ALU—Arithmetic-Logic Unit. Internal, and inaccessible to the programmer, it is the interface between registers and memory, manipulating them as necessary to perform the individual instructions of the microprocessor.

analog—data is represented electrically by varying voltages or amplitudes.

AND—a Boolean logical function. Two operators are tested and if both are true the answer is true. Truth is indicated by a high bit, or "1" in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately.

APL—a popular high-level mathematical language.

argument—any of the independent variables accompanying a command.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—a piece of software that translates operational codes into their binary equivalents.

B

backup—refers to making copies of all software and data stored externally and having duplicate hardware available.

base—a mathematical term that refers to the number of digits in a number system. The decimal system, using digits 0 through 9, is called base 10. The binary system is base 2.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing where many of the same type jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud rate—a measure of the speed at which serial data is sent. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

Boolean algebra—a mathematical system of logic named after George

Boole. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. Sometimes it is keyed in, and on other machines it is in read only memory (ROM). Using this program is called "booting" the system or cold-starting.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specifically designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

carrier—a steady signal that can be slightly modified (modulated) continuously. These modulations can be interpreted as data. In microcomputers the technique is used primarily in modem communications and tape input/output (I/O).

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for that block of data.

chip—a physical package containing electrical circuits. They vary from aspirin-size for a simple timer to about the size of a stick of gum for a complete microprocessor.

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—a piece of software that will convert a program written in a high-level language to binary code.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

constant—a value that doesn't change.

CPU—Central Processing Unit. The circuitry that actually performs the functions of the instruction set.

CRT—Cathode Ray Tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to read/write the right section of tape.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

data base—refers to a series of programs each having a different function but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

digital—all data is represented in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—(DOS) the system software that manipulates the data to be sent to the disk controller.

DMA—direct memory access. A process where the CPU is disabled or bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot matrix printer—instead of each letter having a separate type head (like a typewriter), the single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to make.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrically Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editor.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Electrically Programmable Read Only Memory. The chip is programmed by voltages higher than normal for computer chips. Once programmed, it is used like ROM, but can be erased by exposure to ultraviolet light.

exclusive OR—see XOR.

execution cycle—a cycle during which a single instruction actually occurs.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occured.

flowcharting—a method of graphically displaying program steps, used to develop and define an alogrithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

H

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Contrast with buffer.

hangup—a situation where it seems the computer is not listening to you.

hard copy—a printout.

hardware—refers to any physical piece of equipment in a computer system.

hexadecimal—a number system based on sixteen. The decimal digits 0-9 are used along with the alpha characters A-F, which are also recognized as digits.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming code that does not require a knowledge of the CPU structure.

high order—see most significant.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

host computer—the primary computer in a multi-computer or terminal hookup.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit. See chip.

immediate addressing—the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—the information is addressed by a specified value, or by the value in a specified register.

indirect—the address given points to another address, and the second address is where the information actually is.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compile.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

J

jack—a socket where wires are connected.

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

least significant—refers to the lowest position digit of a number, or rightmost bit of a byte. In 19963 the 3 is the least significant digit. Opposite of most significant.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory that this rule applies to. The last piece of data pushed into the stack is the first piece popped out.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

loop—a set of instructions that executes itself continuously. If the programmer had the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

memory—the hardware that stores data for use by the CPU. Each piece of data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

microprocessor—a CPU on a single chip.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

monitor—1. a CRT. 2. a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

most significant—refers to the highest value position of a number of the left most bit of a byte. In the number 1923 the 1 is most significant because it represents thousands.

multiplexing—a method allowing several sets of data to be sent at different times over the same bus lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, *each* requiring four data lines, can all be written to with only one group of four data lines.

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers have a limit to the number of loops that can be nested.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1).

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0-7.

OEM—Original Equipment Manufacturer.

offset value—a value that can be added to an address. Most addressing modes allow an offset value.

off-the-shelf—a term referring to software. Means the program is generalized so that it can be used by a greater number of computer owners, thus it can be mass produced and bought "off-the-shelf."

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

overflow—a situation that occurs when an arithmetic function requires more than the machine is capable of handling. Most computers have a flag so that this condition can be tested.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word is depends on the computer. Most micros are 8 bit word machines. The term is important because many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

PC board—stands for Printed Circuit board. A piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

peripheral—any piece of hardware that is not a basic part of the computer.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters.

PL/1—a programming language used by very large computers. It incorporates most of the better features from other programming languages.

plotter—a device that can draw graphs and curves controlled by the computer through an interface.

port—a single addressable channel used for communications.

PROM—Programmable Read Only Memory. A memory device that is written to once, and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

R

RAM—an acronym for Random Access Memory. Memory that can be written to or read from. It is addressed by the address bus.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a memory location used by the CPU and not addressed by the address bus. It cannot be used by the programmer.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricty. Transistors and integrated circuits are made from semiconductive material and called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (−) and 0 is positive (+).

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

status register—the register that contains the status flags set and tested by the CPU operations.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

text editor—see word processor.

time-sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

TTL—Transistor-Transistor Logic. Means that the electrical values for logic highs and lows fall within the values necessary to run transistors. See semiconductor.

U

utility—a program designed to aid the programmer in developing other software.

V

variable—a labeled entity that can take on any value.

von Neumann, John (1903-1957)—Mathemetician. Put the concept of games, winning strategy, and different types of games into mathematical formulae. Also advanced the concept of storing the program in memory as opposed to having it on tape.

W

word—in computing it refers to a number of bits that are in a parallel format. If the CPU works with 8 bits then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

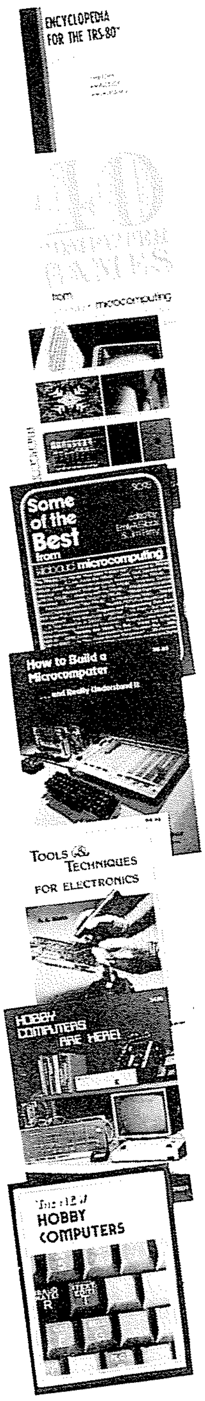
zero page—refers to the first page of memory.

INDEX

INDEX

index

INDEX COMPILED BY NAN MCCARTHY


ENCYCLOPEDIA FOR THE TRS-80*
Some of the Best from kilobaud microcomputing
How to Build a Microcomputer ... and Really Understand It
TOOLS & TECHNIQUES FOR ELECTRONICS
HOBBY COMPUTERS ARE HERE!
THE NEW HOBBY COMPUTERS

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

ISBN 0-88006-030-1