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 7

*Trademark of Tandy Corp.

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 7

wayne GREEN inc.
PETERBOROUGH NH 03458

FIRST EDITION
FIRST PRINTING MARCH 1982

Edited by Kate Comiskey and Katherine Putnam
Proofread by Ann Winsor
Production: Margaret Baker, Gary Ciocci, Linda Drew, Thomas Villeneuve, Robert Villeneuve, Sandra Dukette, and Karen Stewart
Technical Assistance by Jake Commander
Illustrations by Howard Happ

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.

BUSINESS

Point and Figure Charting
For Stocks and Commodities
Dividend Reinvestment Plan

Point and Figure Charting for Stocks and Commodities

by Christopher C. Marvel

Before I bought my computer, I had books of charts for the stock or commodity that I was interested in and a filing cabinet to store information on those companies. Now, my computer is both a filing cabinet and a chart book, and I spend my time analyzing data rather than entering it.

Technical approaches to trading stocks are many and are confusing to most people. Point and figure charting is in this category but is among the least confusing of these technical trading approaches. For more information, see *The Commodity Futures Game: Who Wins? Who Loses? Why?* by Richard J. Teweles, Charles V. Harlow, and Herbert L. Stone, McGraw-Hill, 1974.

Point and figure chartists make two important assumptions. First, they view the volume of trading as unimportant, a mere side effect of price actions with no predictive significance. Second, they dismiss the importance of how much time has elapsed as prices move from one level to another. The only thing that matters is the direction of the price change. Point and figure charts are designed to show only the direction of the price change.

Each point (one box on a piece of graph paper) can indicate any number of actual points, cents, or dollars that the chartist has decided on. I have included an illustration of the program decision model I used to develop the charting program. (See Figure 1.) This will be helpful if you wish to modify the program for your own use.

Point and figure charts are records of price reversals with no reference to time. The more reversals there are, the more columns show up on the chart. PTFIG (see Program Listing) can store up to 62 price reversals (columns). In other words, it can track three months or more of price information for a stock or commodity. The decision model defines one price reversal as a change of four points or more in the opposite direction from the previous day's move.

I tried to use the KISS concept in writing this program—Keep it Simple, Stupid. The program is documented to show how you can modify it for your own use. The program is written for a TRS-80 Model I with a disk drive and a printer. Because it is in BASIC, it is compatible with the Model III. Don't be deceived by the program's small size. It packs a lot of practical punch in a small amount of space. It maintains files on disk and adds a /PF extension to the file to identify it as a PTFIG file. Figure 2 gives an overview of the program.

1. Get high price for today
2. Is current field up or down

Up					**Down**
Is price one point Higher	YES	Enter new points Up	Is price one point Lower	YES	Enter new points up Down
NO			NO		
Is price three points Lower	NO	DONE	Is price three points Higher	NO	DONE
Move right one column Enter price Down			Move right one column Enter price Up		
DONE			DONE		

Figure 1. *Point and Figure decision model*

Using PTFIG

PTFIG is executed from BASIC. After DOS READY, type BASIC. Answer 1 to the number of files, press ENTER for memory size, and type RUN "PTFIG/BAS. The HELP menu is then displayed. Enter R after studying the display. The screen clears, and the following appears at the bottom:

```
STOCK:     LAST DATE:     FIELD:     COMMAND DEILNPS ??
```

This puts you in the PTFIG command mode, and you can enter any of the command keys shown in Table 1.

1. Charts and holds data for three months or more
2. Automatically rolls out old data and rolls in new data, always keeping your file current
3. Minimal hardware requirements. May be used with tape or disk
4. Provides easy scrolling for most recent entries for printing hard copy of charts
5. Originally configured for an Okidata Printer and can provide standard or reduced size hard copy. (10 characters per inch, 8 lines per inch or 16.5 characters per inch, 8 lines per inch)
6. Will run in Level II with 4K

Figure 2. *Point and Figure overview*

Command Key	Description
D	Displays stock or commodity chart currently stored in memory on the screen video
E	Ends the program
H	Displays HELP menu
I	Inputs new data to stock or commodity chart currently stored in memory
L	Loads stock or commodity chart file from disk
N	Enters a new stock or commodity. (This erases any previous file in memory. Enter up to eight characters but not more.)
P	Prints stock or commodity chart on line printer. (This uses a video screen display routine; so the chart must be displayed on the video screen.)
S	Saves stock or commodity chart in memory to disk. (All PTFIG files have a /PF extension added to their filename.)
↑	Scrolls chart up on screen
Space bar	Scrolls chart down on screen

Table 1. *Command keys and descriptions*

The program uses a screen print routine, and you can print three different scales just by changing the CHR$ value. It rolls out old information once the screen is filled; so, if you want to track more than three months worth of data, you should archive the file and start a new one once the screen is full. The following is the procedure for using PTFIG.

1) Choose the stock or commodity that you wish to track.

2) Determine how much of a price change would be equivalent to a point tracked by PTFIG/BAS.

Example: STOCK
IBM one point = a change of $.25. If the IBM stock were to go up $.75 from $30.00 to $30.75, enter +3 for the day's change.

COMMODITY
HOGS one point = a change of $.02. If hogs lost $.14 on the day, enter −7 for the day's change.

3) Run PTFIG/BAS. A HELP menu appears with a list of all available commands for command mode. Enter R to return to command mode from the HELP menu. If you wish to display the HELP menu again, press H.

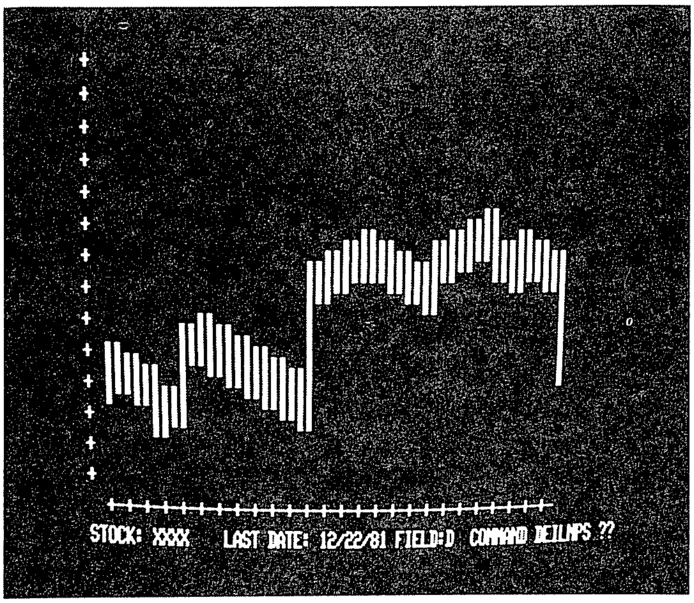
4) To enter a new commodity or stock, type N. Enter the name, date, and first day's point change. (See step 2.) If you wish to enter more data, press I after the initial display.

5) After data entry is complete, press S to store the information on disk. A /PF extension is added to the file to differentiate it from other files in the disk directory.

6) To enter the next day's data, run PTFIG/BAS. Go to the command mode. Press L (load). Enter the name of the stock or commodity stored on disk, and the file is loaded. Enter the data and repeat step 5 to save it.
7) Because the price charts could be larger than the screen, I added a scrolling provision to the program. To scroll the chart upward, press the up arrow. To scroll the chart down, press the space bar.
8) I added a filter provision to the program in order to screen whipsaw price gyrations. The current filter is 3. This means that if the current field is U (up), a price change of equal to or greater than 3 must be registered on the downside for the chart to show any downside movement. The reverse is true when the field is D (down).

Example: IBM has a $ – .50 for the day which equates to – 2 points (see step 2) for charting purposes. IBM has been in a solid rally, and its field is U (up). After you have entered the data, NO CHANGE appears in the chart because this is against the stock's major trend. This result is due entirely to the action of the filter.

To change the current filter from 3 to another value, edit line 18 of the program. The variable P is greater than 3 or less than 3 at two places in the line. Replace 3 with the value you prefer to use.



```
STOCK: XXXX   LAST DATE: 12/22/81 FIELD:D  COMMAND DEILNPS ??
```

Photo 1. *Screen display*

9) The current program is set up to produce a condensed printout (132 characters per line). Line 23 contains LPRINTCHR$(29). If you wish to have regular size printing, change this value to LPRINTCHR$(30). If you want larger type, change it to LPRINTCHR$(31).
10) To print hard copy, press P in the command mode, and the printer prints the chart that is currently displayed on the screen. Photo 1 shows a sample screen display.

Program Listing. *Point and Figure*

```
1 CLEAR 1000:
  DEFINT A - Z:
  DIM R(62),C(62),L(62),F(62):
  FOR I = 1 TO 61:
   F(I) = 0:
   R(I) = 0:
   C(I) = 0:
   L(I) = 0:
   NEXT I:
  N = 0:
  CLS :
  PRINT @400,"P O I N T   &   F I G U R E";:
  PRINT @471,"C H A R T I N G";:
  PRINT @540,"B Y";:
  PRINT @596,"CHRISTOPHER C. MARVEL";
2 PRINT @663,"COPYRIGHT 1981";:
  FOR I = 15824 TO 15829:
   POKE I,149:
   POKE I + 24,149:
   POKE I + 64,149:
   POKE I + 88,149:
   NEXT I:
  POKE 15952,149:
  POKE 15981,149:
  POKE 16016,149:
  POKE 16045,149:
  FOR I = 1 TO 29:
   POKE 16079 + I,131:
   NEXT I:
  POKE 16109,129:
  FOR I = 1 TO 3000:
   NEXT :
  GOTO 35
3 CLS
4 IF F(N) = 1
   THEN
    F$ = "U"
5 IF F(N) = - 1
   THEN
    F$ = "D"
6 PRINT @960,"STOCK: ";S$;:
  PRINT @975,"LAST DATE: ";D$;:
  PRINT @995,"FIELD:";F$;:
  PRINT @1004,"COMMAND DEILNPS ??";
7 A$ = INKEY$:
  IF A$ = "L"
   THEN
    12:
   ELSE
    IF A$ = "N"
     THEN
      14:
     ELSE
      IF A$ = "E"
       THEN
        CMD "S"
8 IF A$ = "H"
   THEN
    35:
   ELSE
    IF A$ = "P"
     THEN
      23:
     ELSE
      IF A$ = "S"
       THEN
        22:
```

```
        ELSE
         IF A$ = "D"
          THEN
           25:
          ELSE
           IF A$ = "I"
            THEN
             17:
            ELSE
             IF A$ = "["
              THEN
               9:
              ELSE
               IF A$ = " "
                THEN
                 11:
                ELSE
                 7
 9 IF R(N) = 0
    THEN
     7:
    ELSE
     FOR I = 1 TO N:
      R(I) = R(I) - 10:
      NEXT I:
     GOTO 25
10 IF R(N) = 0
    THEN
     7
11 FOR I = 1 TO N:
    R(I) = R(I) + 10:
    NEXT I:
   GOTO 25
12 CLS :
   INPUT "STOCK";S$:
   SA$ = S$ + "/PF":
   OPEN "I",1,SA$:
   INPUT #1,N,D$:
   FOR I = 1 TO N:
    INPUT #1,F(I),R(I),C(I),L(I):
    NEXT I:
   CLOSE :
   CLS
13 PRINT @448,"FILE LOADED":
   PRINT "ADD DATA (Y/N)";:
   INPUT " ";A$:
   IF LEFT$(A$,1) = "Y"
    THEN
     17:
    ELSE
     IF LEFT$(A$,1) = "N"
      THEN
       25:
      ELSE
       13
14 N = 1:
   F(N) = 0:
   S$ = "":
   D$ = "":
   CLS :
   PRINT :
   INPUT "STOCK NAME";S$:
   PRINT :
   INPUT "DATE: ";D$:
   PRINT :
   INPUT "PTS. + OR -  ";P:
   IF P = > 1
    THEN
     F(1) = 1:
     R(1) = 22 - P:
```

Program continued

```
     C(1) = 5:
     L(1) = P
15 IF P = < - 1
    THEN
     F(1) = - 1:
     R(1) = 22:
     C(1) = 5:
     L(1) = ABS(P)
16 GOTO 25
17 CLS :
   PRINT S$:
   PRINT :
   PRINT "DATA LAST ENTERED ON ";D$:
   PRINT :
   INPUT "NEW DATE: ";D$:
   PRINT :
   INPUT "PTS. + OR - ";P:
   IF F(N) = 1 AND P = > 1
    THEN
     R(N) = R(N) - P:
     L(N) = L(N) + P:
     GOTO 19:
    ELSE
     IF F(N) = - 1 AND P = < - 1
      THEN
       L(N) = L(N) + ABS(P):
       GOTO 19
18 IF F(N) = 1 AND P = < - 3
    THEN
     N = N + 1:
     F(N) = - 1:
     C(N) = C(N - 1) + 2:
     R(N) = R(N - 1):
     L(N) = ABS(P):
     GOTO 19:
    ELSE
     IF F(N) = - 1 AND P = > 3
      THEN
       N = N + 1:
       F(N) = 1:
       C(N) = C(N - 1) + 2:
       R(N) = R(N - 1) + L(N - 1) - P:
       L(N) = P
19 IF N < 62
    THEN
     20:
    ELSE
     FOR I = 1 TO N:
      C(I - 1) = C(I) - 2:
      R(I - 1) = R(I):
      L(I - 1) = L(I):
      F(I - 1) = F(I):
      NEXT I:
     N = N - 1
20 PRINT
21 PRINT @512," ";:
   INPUT "ENTER MORE INFO (Y/N)";A$:
   IF LEFT$(A$,1) = "Y"
    THEN
     17:
    ELSE
     IF LEFT$(A$,1) = "N"
      THEN
       25:
      ELSE
       21
22 SA$ = S$ + "/PF":
   OPEN "O",1,SA$:
   PRINT #1,N,D$:
   FOR I = 1 TO N:
     PRINT #1,F(I),R(I),C(I),L(I):
     NEXT I:
    CLOSE :
    CLS :
    PRINT @473," ";S$;" SAVED!!!!":
    GOTO 6
23  LPRINT CHR$(29):
    GOSUB 24:
    GOTO 6
24  FOR L = 0 TO 15:
     FOR I = 15360 + 64 * L TO 15422 + 64 * L:
      LPRINT CHR$( PEEK(I));:
      NEXT I:
     LPRINT CHR$(13);:
     NEXT L:
    RETURN
25  CLS :
    FOR I = 0 TO 832 STEP 64:
     PRINT @I,"+":
     NEXT :
    FOR I = 1 TO N:
     Y = R(I):
     X = C(I):
     L = L(I):
     F = F(I):
     IF R(N) = > 0
      THEN
       26:
      ELSE
       Y = Y + ABS(R(N))
26   D1 = R(N) + L(N):
     IF D1 < 42
      THEN
       27:
      ELSE
       Y = Y + 41 - D1
27   IF Y < 0
      THEN
       L = L - ABS(Y):
       Y = 0
28   IF Y > 41
      THEN
       Y = 0:
       L = 0
29   IF Y + L = > 41
      THEN
       L = 41 - Y
30   IF L < = 41
      THEN
       31:
      ELSE
       L = 41
31   IF Y = 0 AND L = 0
      THEN
       32:
      ELSE
       FOR K = 0 TO L:
        SET(X,Y + K):
        NEXT K
32   IF F = 1 PRINT @897 + I,"+"
33   IF F = - 1 PRINT @897 + I,"-"
34   NEXT I:
    GOTO 4
35  CLS :
    PRINT "************************ POINT & FIGURE ****************
    ******":
    PRINT :
    PRINT "                         HELP!!!!!":
    PRINT :
    PRINT "D =====> Display chart","N =====> Enter new commodity"
```

Program continued

```
36 PRINT "E =====> End & Return to DOS","P =====> Print chart":
   PRINT "H =====> Help       ","S =====> Save to disk":
   PRINT "I =====> Input data","[ =====> Scroll up":
   PRINT "L =====> Load from Disk","Sp. Bar =====> Scroll down":
   PRINT
37 PRINT "PRESS R to return main program"
38 A$ = INKEY$
39 IF A$ = "R"
    THEN
     3
40 GOTO 38
```

Dividend Reinvestment Plan

by Max Rosenzweig

The Economic Recovery Act of 1981 provides a way of paying less in taxes. Commonly called DRIP, the Dividend Reinvestment Plan went into effect January 1, 1982. It provides for deferral of taxes on dividend income from qualified utility companies when the dividends are reinvested in new shares of stock. The single taxpayer can defer taxes on up to $750 of dividends a year. The married taxpayer filing jointly can defer taxes on up to $1500. The deferral is valid until the stock is sold. If the stock is held for one year or more, the gain is considered a capital gain and is taxed at the capital gains rate; 40 percent of the gain is taxable. If you never sell the stock, you never pay any taxes on the dividends.

The law, as presently enacted, expires in 1985. It will be up to Congress to extend it at that time or let it die. In the meantime, you can build up a nest egg for retirement or college for the kids. Deferring taxes until you retire, when your income may be drastically reduced, putting you in a lower tax bracket, can ease the tax bite.

This program calculates the number of shares bought by reinvesting the dividend income, the new dividend payable at the end of the selected period, and the capital gains, if the stock is sold.

If you are a single taxpayer in the 30 percent tax bracket and have $750 in qualifying dividends, your taxes are $225. Under DRIP, if you hold the stock for a year and reinvest the dividends, your taxes will be $90 when you sell the stock, giving you savings of $135. As an example, suppose you own 500 shares of ABC Utility Company. You purchased them at $11.00 a share, and each share pays $1.48 in dividends (37 cents quarterly). The dividend income is $740 a year ($185 a quarter). Now refer to Figure 1. It compares investor 1, who does not reinvest the dividends, to investor 2 who has a DRIP plan. Note that investor 2 received more dividend income at the end of one year. This is due to compounding reinvestment each quarter. Note that the tax on capital gain is $90 ($750 × .40 × .30). The balance of $94.70 is taxed as ordinary income for $28.41 for total taxes on the dividends of $118.41. This is $103.59 less in taxes than investor 1 paid on $750 of dividends. A similar example shows the results for taxpayers filing a joint return. (See Figure 2.)

Now look at Figure 3. It shows two-year results from reinvesting. Again, investor 1 did not reinvest; so his dividend is still only $740.00, and his taxes again are $222. But investor 2 now has $964.22 in dividends and a total of

STARTING YEAR	1982	
ENDING YEAR	1982	
NO. OF SHARES	500	
DIVIDEND RATE	1.48	
COST PER SHARE	11.00	
TAX BRACKET	0.30	
FILING STATUS	S	
	INVESTOR #1	INVESTOR #2
DIVIDEND INCOME	740.00	844.70
CURRENT YEAR INCOME TAX	222.00	28.41
NEWLY ISSUED SHARES FROM DRIP	0	70.74
CAPITAL GAINS TAX	—	90.00
TAX SAVINGS	—	103.59
AFTER TAX PROCEEDS	518.00	726.29

Figure 1

151.5 new shares over a two-year period and a capital gains tax of $90 if stock is sold this year. The tax on the balance of $214.22 is $64.26 in the second year (remember it was $28.41 the first year), for total taxes the second year of $154.26.

STARTING YEAR	1982	
ENDING YEAR	1982	
NO. OF SHARES	500	
DIVIDEND RATE	1.48	
COST PER SHARE	11.00	
TAX BRACKET	0.30	
FILING STATUS	J	
	INVESTOR #1	INVESTOR #2
DIVIDEND INCOME	740.00	844.70
CURRENT YEAR INCOME TAX	222.00	0.00
NEWLY ISSUED SHARES FROM DRIP	0	70.74
CAPITAL GAINS TAX	—	101.36
TAX SAVINGS	—	120.64
AFTER TAX PROCEEDS	518.00	743.34

Figure 2

```
STARTING YEAR                   1982
ENDING YEAR                     1983
NO. OF SHARES                   500
DIVIDEND RATE                     1.48
COST PER SHARE                   11.00
TAX BRACKET                       0.30
FILING STATUS                     S

                                INVESTOR #1  INVESTOR #2
DIVIDEND INCOME                 740.00       964.22
CURRENT YEAR INCOME TAX         222.00        64.26
NEWLY ISSUED SHARES FROM DRIP     0          151.50
CAPITAL GAINS TAX                 —           90.00
TAX SAVINGS                       —           67.74
AFTER TAX PROCEEDS              518.00       809.95
```

Figure 3

As you can see in these examples, the amount over $750 or $1500, as the case may be, is taxable as ordinary income in the year it is earned. You have a choice of selling your stock that yields more than the allowed dividends or paying the taxes. It is still a good reduction in taxes while you build up your investment in stock. If you never sell the stock, you keep deferring $750 or $1500 each year. There are other fine points, such as discounts and return of capital which are beyond the scope of this article. Consult your broker or financial advisor for your particular situation.

The program through line 280 is an explanation of the plan and program. This program takes less than 4K of memory, but if you are short of memory, you can omit these lines. If you don't want to view them each time you run the program, enter RUN290.

Lines 300 to 560 perform the calculations for reinvesting the dividends and compounding them quarterly. Lines 580 to 680 print out the results. If you do not have a printer, omit the LPRINT statements and remove line 580 as well. The program does provide for use without a printer. Enter N when asked if you want hard copy. Lines 800 to 930 print the results on the screen, and the program asks if you want to run again. If you do, the program bypasses the explanation at the beginning of the program.

Program Listing. *Dividend reinvestment*

```
 10 CLS
 20 PRINT @132, CHR$(23),"DRIP"
 30 PRINT @256,"A DIVIDEND REINVESTMENT PROGRAM"
 40 PRINT @408,"WRITTEN BY"
 50 PRINT @532,"MAX ROSENZWEIG"
 60 PRINT @666,"OCT 1981"
 70 FOR I = 1 TO 1500:
     NEXT I
 80 CLS
 90 PRINT "UNDER THE PROVISIONS OF THE ECONOMIC RECOVERY ACT OF
100 PRINT "1981 AN INDIVIDUAL MAY REINVEST THE DIVIDEND INCOME RECEI
    VED"
110 PRINT "FROM STOCK OF A QUALIFYING UTILITY COMPANY AND DEFER"
120 PRINT "FEDERAL TAXES UNTIL THE STOCK IS SOLD. IF THE NEW"
130 PRINT "SHARES ARE HELD ONE YEAR OR MORE, THE GAIN IS THEN"
140 PRINT "TREATED AS LONG TERM CAPITAL GAINS."
150 FOR I = 1 TO 4000:
     NEXT I
160 CLS
170 PRINT "THIS PROGRAM WILL CALCULATE THE NUMBER OF SHARES BOUGHT"
180 PRINT "BY REINVESTING THE DIVIDEND INCOME, THE NEW DIVIDEND"
190 PRINT "PAYABLE AT THE END OF THE SELECTED PERIOD, AND THE"
200 PRINT "CAPITAL GAINS, IF THE STOCK IS SOLD."
210 FOR I = 1 TO 4000:
     NEXT I
220 CLS
230 PRINT "WHEN ASKED FOR, ENTER THE YEAR FROM WHICH YOU WANT TO"
240 PRINT "START THE CALCULATION; THE YEAR YOU WANT TO END THE"
250 PRINT "CALCULATIONS; THE NUMBER OF SHARES OF STOCK HELD AT THE"
260 PRINT "BEGINNING OF THE PERIOD; THE ANNUAL DIVIDEND RATE;"
270 PRINT "AND THE COST PER SHARE."
271 FOR I = 1 TO 4000:
     NEXT I:
    CLS
272 PRINT :
    PRINT "TWO EXAMPLES WILL BE CALCULATED.  THE FIRST IS"
273 PRINT "WHERE THE DIVIDEND INCOME"
274 PRINT "IS NOT REINVESTED IN THE DRIP.  THE SECOND EXAMPLE"
275 PRINT "ELECTS TO PARTICIPATE IN DRIP.  ASSUME NO CHANGE IN"
276 PRINT "STOCK PRICE, DIVIDEND AND NO DISCOUNT.
280 FOR I = 1 TO 4000:
     NEXT I
290 CLS
300 CLEAR 100
305 E$ = "####.##"
310 INPUT "STARTING YEAR";SY
320 INPUT "ENDING YEAR";EY
330 INPUT "NUMBER OF SHARES";S1
335 INPUT "COST PER SHARE";CS
340 INPUT "ANNUAL DIVIDEND RATE";DR
355 INPUT "TAX BRACKET (ENTER AS DECIMAL, I.E. .35)";TB
356 INPUT "FILING STATUS (<S>INGLE OR <J>OINT)";F$
357 IF F$ = "S"
     THEN
      MD = 750
358 IF F$ = "J"
     THEN
      MD = 1500
360 QR = DR / 4
370 SN = S1
390 FOR I = SY TO EY
400  FOR J = 1 TO 4
410   SN = (SN * QR) / CS + SN
420   Q1 = (SN * QR)
430   NEXT J
440  NEXT I
450 DI = S1 * DR
```

```
460 NS = SN - S1
510 ND = SN * DR
520 IT = DI * TB
540 GT = ND * TB * .4
560 AT = DI - IT
570 CLS :
    GOTO 800
580 INPUT "READY PRINTER, PRESS <ENTER>";A$
601 LPRINT :
    LPRINT TAB(5)"STARTING YEAR "; TAB(40);SY
602 LPRINT :
    LPRINT TAB(5)"ENDING YEAR   "; TAB(40);EY
603 LPRINT :
    LPRINT TAB(5)"NO. OF SHARES "; TAB(40);S1
604 LPRINT :
    LPRINT TAB(5)"DIVIDEND RATE"; TAB(40) USING E$;DR
605 LPRINT :
    LPRINT TAB(5)"COST PER SHARE "; TAB(40) USING E$;CS
610 LPRINT :
    LPRINT TAB(5)"TAX BRACKET "; TAB(40) USING E$;TB
611 LPRINT :
    LPRINT TAB(5)"FILING STATUS"; TAB(43)F$
612 LPRINT :
    LPRINT TAB(38)"INVESTOR #1"; TAB(54)"INVESTOR #2"
613 LPRINT TAB(38) STRING$(11,"-"); TAB(54) STRING$(11,"-")
620 LPRINT :
    LPRINT TAB(5)"DIVIDEND INCOME"; TAB(40) USING E$;DI;:
    LPRINT TAB(56) USING E$;ND
630 LPRINT :
    LPRINT TAB(5)"CURRENT YEAR INCOME TAX"; TAB(40) USING E$;IT;:
    LPRINT TAB(56) USING E$;OT
640 LPRINT :
    LPRINT TAB(5)"NEWLY ISSUED SHARES FROM DRIP"; TAB(43)"0";:
    LPRINT TAB(56) USING E$;NS
650 LPRINT :
    LPRINT TAB(5)"CAPITAL GAINS TAX"; TAB(43);"-";:
    LPRINT TAB(56) USING E$;GT
670 LPRINT :
    LPRINT TAB(5)"TAX SAVINGS"; TAB(43)"-";:
    LPRINT TAB(56) USING E$;TS
680 LPRINT :
    LPRINT TAB(5)"AFTER TAX PROCEEDS"; TAB(40) USING E$;AT;:
    LPRINT TAB(56) USING E$;AP
780 INPUT "DO YOU WANT ANOTHER RUN (Y/N)";A$
790 IF A$ = "Y"
     THEN
      RUN 290 :
     ELSE
      END
800 PRINT TAB(5)"STARTING YEAR"; TAB(25)SY; TAB(35)"ENDING YEAR";
    TAB(50)EY
810 PRINT "NO. SHARES="; TAB(11);S1; TAB(18)"COST PER SHARE=";
    TAB(33) USING E$;CS;:
    PRINT TAB(42)"DIV. RATE ="; TAB(53) USING E$;DR
820 PRINT TAB(5)"TAX BRACKET"; TAB(20)TB; TAB(30)"FILING STATUS";
    TAB(45)F$
830 PRINT TAB(20)"INVESTOR #1"; TAB(40)"INVESTOR #2"
840 PRINT TAB(20) STRING$(11,"-"); TAB(40) STRING$(11,"-")
850 PRINT "DIV. INCOME"; TAB(20) USING E$;DI;:
    PRINT TAB(40) USING E$;ND
860 PRINT "CURRENT INCOME TAX"; TAB(20) USING E$;IT;:
    PRINT TAB(44)"0"
870 PRINT "NEW SHARES"; TAB(24)"0"; TAB(40) USING E$;NS
875 IF ND = > MD
     THEN
      GT = 90:
      GOTO 877
876 IF ND < = MD
     THEN
      GT = ND * TB * .4:
```

Program continued

```
        OT = 0:
        GOTO 880
877 OT = (ND - MD) * TB
880 PRINT "CAP. GAINS TAX"; TAB(24)"-"; TAB(40) USING E$;GT
885 PRINT "ORD. INCOME TAX"; TAB(20);"";:
    PRINT TAB(40) USING E$;OT
886 TS = IT - (GT + OT)
890 PRINT "TAX SAVINGS"; TAB(24)"-"; TAB(40) USING E$;TS
895 AP = ND - (GT + OT)
900 PRINT "AFTER TAX PROCEEDS"; TAB(20) USING E$;AT;:
    PRINT TAB(40) USING E$;AP
905 PRINT
910 INPUT "DO YOU WANT A HARD COPY (Y/N) ";A$
920 IF A$ = "N" GOTO 780
930 IF A$ = "Y" GOTO 580
```

EDUCATION

Keeping Track— Student Scheduling and Attendance Part III

by Ulderic F. Racine

The first two parts of this series presented programs that allow you to enter schedule data on students, change existing student schedules, and print class rosters. Part III contains programs that allow you to enter attendance data for up to a month at a time and to print out schedules by students or a record of students/teacher/period by class name.

Program Listing 1 is the attendance initialization program (ATTENDIT). This program functions in the same manner as the schedule initialization program given in Part I. It allows you to specify the drive on which the initial attendance data will be written, select the number of days of attendance that will be entered, and to specify the method of input, either by teacher or by student. Finally, you must specify the number of class periods of attendance per day that count as a full day of attendance. If you select four or more class periods as constituting a full day of attendance, a student with three hours of class attendance is given a half-day of credit.

When you select option 5 of the master menu, Enter Attendance Data, ATTENDIT is loaded and run. It searches the disks currently in the drives for previously entered attendance data. If it finds no data, the program asks if you have a disk with attendance data on it. If you have not previously entered attendance data, the program asks a series of questions to initialize the attendance files.

```
ON WHICH DRIVE SHALL I WRITE THE ATTENDANCE DATA? (1-2-3)
HOW MANY DAYS OF ATTENDANCE DO YOU WISH TO ENTER? (1-23)
DO YOU WANT TO ENTER DATA BY TEACHER OR STUDENT?
ENTER 'T' FOR TEACHER OR 'S' FOR STUDENT (T/S)
HOW MANY PERIODS WILL BE USED FOR FULL-TIME ATTENDANCE?
```

The program creates the files necessary to record attendance data. After the initial designation of the drive number, it is not necessary to place that disk in the same drive. Each time the attendance cycle is completed (the data for all students or each class has been entered for the chosen number of days), you need to enter the number of days in the new cycle and decide whether you want to enter data by student or by teacher. You can enter attendance data by student for one cycle and by teacher for the next cycle. The program also displays the number of periods you select as equivalent to full-time attendance. You have the option to change this at the beginning of any attendance cycle.

Program Listing 2 is the attendance input by teacher program (TEATTEND). When you first enter attendance data, the program begins with the first teacher, first period, and first student. You must complete the input for all the students in a class period. You then have the option to enter another period. If you stop entering attendance data, the program records where you stopped and begins with the next period or teacher. The input display for attendance is shown below.

```
TEACHER: JONES      CLASS: MATH I      PERIOD 1
------------------------------------------------
STUDENT: JOHNSON FRED
------------------------------------------------
DAYS 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
WAS JOHNSON FRED PRESENT IN THIS CLASS
FOR ALL 15 DAYS? (Y/N)
```

If the student was present for all 15 days, type Y and press ENTER. The program displays the next student enrolled in that class. It may take a minute or so for the program to display the name of the next student. During this time, the machine rearranges the arrays it uses to store data. If the student was not present for all 15 days, the program asks you to enter each day individually, pressing P for present or A for absent. You do not have to press ENTER. Remember that you must enter the attendance data on each student before you can drop a student from a class. Once a student is dropped from a class, the student does not show up as being scheduled for that class.

Program Listing 3 is the class schedule by student printout program (PNTSTCHD). There are two options available. You can print all students currently on the file, or you can print the schedule for a specific student. In either case, you have the option to print hard copy or display the print on the screen. The printout follows this format:

```
------------------------------------------------
STUDENT: JOHNSON FRED
PERIOD          CLASS           TEACHER
------------------------------------------------
1               MATH            JONES
2               LITERATURE      BURTON
3               HOMEROOM        ADAMS
4               GEOGRAPHY       WARTON
5               ENGLISH         WEBSTER
6               SCIENCE I       EVERSON
------------------------------------------------
```

Program Listing 4 is the printout by class name program (PNTCLASS). It gives you the same options as the student schedule and class roster programs.

You can get a printout for a specific class name or for all class names. The same options in regard to hard copy or video print are also available. A sample printout appears below.

```
CLASS: MATH I
STUDENT          TEACHER      PERIOD
-------------------------------------
JOHNSON FRED     JONES        1
DELL CHARLES     JONES        1
ABBOT THOMAS     JONES        1
DEERING JOHN     EVERSON      2
```

Program Listing 1. *Attendance initialization*

```
 10 :
    '  ATTENDANCE INITIALIZATION PROGRAM ( ATTENDIT )
 20 :
    '  COPYRIGHT OCTOBER 1, 1981
 30 :
    '  ULDERIC F. RACINE
 40 :
    '  2520 S.E. ALEXANDER DRIVE
 50 :
    '  TOPEKA, KANSAS 66605
100 CLEAR 3000
110 ON ERROR GOTO 400
120 OPEN "R",1,"DATTEND":
    RN = LOF (1):
    IF RN = 0
     THEN
      130:
     ELSE
      320
130 CLOSE :
    KILL "DATTEND"
140 CLS :
    PRINT @448,"I HAVE READ THE DISKS CURRENTLY IN THE DRIVES.":
    PRINT "I CANNOT FIND A DISK WITH ATTENDANCE DATA.":
    LINE INPUT "DO YOU HAVE A DISK WITH ATTENDANCE DATA ? ( Y/N )  "
    ;AN$:
    GOSUB 390:
    IF AN$ = "Y"
     THEN
      150:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        140:
       ELSE
        160
150 PRINT @448, CHR$(31);:
    LINE INPUT "PLEASE PUT THE DISK IN A DRIVE ( 1 - 2 - 3 ) AND PRE
    SS <ENTER> ";AN$:
    GOTO 120
160 PRINT @448, CHR$(31);:
    PRINT "ON WHICH DRIVE SHALL I WRITE":
    LINE INPUT "THE ATTENDANCE DATA ? ( 1 - 2 - 3 ) ";DR$:
    IF VAL(DR$) < 1 OR VAL(DR$) > 3
     THEN
      160
170 CLS :
    PRINT @448,"HOW MANY DAYS OF ATTENDANCE":
    LINE INPUT "DO YOU WISH TO ENTER ? ( 1 - 23 )  ";CA$:
    CA = VAL(CA$):
    IF CA < 1 OR CA > 23
     THEN
      170
180 PRINT @448, CHR$(31);:
    PRINT "DO YOU WANT TO ENTER DATA BY TEACHER OR STUDENT ? ":
    LINE INPUT "<ENTER> 'T' FOR TEACHER OR 'S' FOR STUDENT ( T/S ) "
    ;AT$:
    IF AT$ = "T"
     THEN
      AT = 1:
      :
     ELSE
      IF AT$ = "S"
       THEN
        AT = 2:
```

```
       ELSE
        GOTO 180
190 IF UR = 1
     THEN
      220
200 PRINT @448, CHR$(31);"HOW MANY PERIODS OF ATTENDANCE PER DAY":
    LINE INPUT "WILL BE CONSIDERED A FULL DAY ? ";AN$:
    IF VAL(AN$) < 1 OR VAL(AN$) > 16
     THEN
      200:
     ELSE
      AV = VAL(AN$)
210 GOTO 240
220 PRINT @448, CHR$(31);"THE CURRENT FULL DAY ATTENDANCE IS ";AV;"
    PERIODS MINIMUM."
230 LINE INPUT "DO YOU WISH TO CHANGE IT ? ( Y/N ) ";AN$:
    IF AN$ = "Y"
     THEN
      200:
     ELSE
      IF AN$ < > "N"
       THEN
        230:
       ELSE
        240
240 IF UR = 1
     THEN
      360
250 TA = 0:
    SD = 1:
    SR = 0:
    NS = 0:
    CT = 1:
    TF = 2:
    PN = 0
260 IF AT = 1
     THEN
      CR = 1:
     ELSE
      CR = 2
270 DS$ = "TATTEND:" + DR$:
    DR$ = "DATTEND:" + DR$:
    OPEN "R",1,DR$:
    OPEN "R",2,DS$
280 FIELD 1,2ASXA$,2ASXB$,2ASXC$,2ASXD$,2ASXE$,2ASXF$,2ASXG$,2ASXH$,
    2ASXI$,2ASXJ$,2ASXK$
290 LSET XA$ = MKI$ (TA):
    LSET XB$ = MKI$ (CA):
    LSET XC$ = MKI$ (SD):
    LSET XD$ = MKI$ (CR):
    LSET XE$ = MKI$ (SR):
    LSET XF$ = MKI$ (NS):
    LSET XG$ = MKI$ (CT):
    LSET XH$ = MKI$ (AT):
    LSET XI$ = MKI$ (TF):
    LSET XJ$ = MKI$ (PN):
    LSET XK$ = MKI$ (AV)
300 PUT 1,1:
    CLOSE
310 IF AT = 1
     THEN
      RUN "TEATTEND":
     ELSE
      RUN "STDATEND"
320 FIELD 1,2ASXA$,2ASXB$,2ASXC$,2ASXD$,6ASDUMMY$,2ASXH$,4ASDV$,2ASX
    K$
330 GET 1,1
340 AT = CVI (XH$):
    AV = CVI (XK$):
    CA = CVI (XB$):
```

Program continued

```
    IF CA = 0
     THEN
      UR = 1:
      GOTO 170
350 CLOSE :
    GOTO 310
360 LSET XH$ = MKI$ (AT):
    LSET XB$ = MKI$ (CA):
    LSET XD$ = MKI$ (AT):
    LSET XK$ = MKI$ (AV)
370 PUT 1,1:
    CLOSE :
    GOTO 310
380 RUN "CLASMENU"
390 AN$ = LEFT$(AN$,1):
    RETURN
400 CLS :
    PRINT @394,"AN ERROR HAS OCCURED IN THE EXECUTION OF THE PROGRAM
     CALLED 'ATTENDANCE INITIALIZATION'."
410 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
420 PRINT TAB(5)"ERROR LINE = "; ERL
430 FOR V = 1 TO 5000:
     NEXT V
440 STOP
```

Program Listing 2. *Attendance initialization by teacher*

```
 10 :
    '  ATTENDANCE INPUT BY TEACHER ( TEATTEND )
 20 :
    '  COPYRIGHT OCTOBER 1, 1981
 30 :
    '  ULDERIC F. RACINE
 40 :
    '  2520 S.E. ALEXANDER DRIVE
 50 :
    '  TOPEKA, KANSAS 66605
100 CLS :
    PRINT CHR$(23):
    PRINT @450,"ATTENDANCE INPUT BY TEACHER"
110 UR = 1:
    GOTO 130
120 DEFINT A - Z
130 OPEN "R",1,"DATTEND"
140 OPEN "R",2,"STDSCHED":
    OPEN "R",3,"CLASSES"
150 FIELD 2,2ASFA$,2ASFB$,2ASFC$,2ASFD$,2ASFE$,2ASFG$,2ASFH$,2ASFI$,
    2ASFJ$,2ASFK$
160 GET 2,1
170 FS = CVI (FB$):
    UF = CVI (FC$):
    NC = CVI (FE$):
    NP = CVI (FG$):
    RN = CVI (FH$):
    Q = CVI (FI$)
180 X = ( LOF (2) - 1) * UF:
    RO = LOF (3) * 10
190 IF UR = 1
     THEN
      T = (X * FS) + (RO * 15) + (23 * X) + 4000:
      CLOSE :
      CLEAR T:
      GOTO 120
200 UR = 0
210 ON ERROR GOTO 2060
220 DIM SN$(X),CN$(RO),SA$(X + 3)
```

```
230 FIELD 1,2ASXA$,2ASXB$,2ASXC$,2ASXD$,2ASXE$,2ASXF$,2ASXG$,2ASXH$,
    2ASXI$,2ASXJ$,2ASXK$
240 GET 1,1
250 TA = CVI (XA$):
    CA = CVI (XB$):
    SD = CVI (XC$):
    CR = CVI (XD$):
    SR = CVI (XE$):
    CT = CVI (XG$):
    AT = CVI (XH$):
    TF = CVI (XI$):
    PN = CVI (XJ$):
    AV = CVI (XK$)
260 N1 = 0:
    NS = 0:
    Q = 0:
    RN = 2
270 G = Q * FS
280 FIELD 2,(G)ASDUMMY$,(FS)ASDA$
290 GET 2,RN
300 IF DA$ = STRING$(FS,88)
     THEN
      NS = N1:
      GOTO 350
310 N1 = N1 + 1
320 SN$(N1) = DA$
330 Q = Q + 1:
    IF Q = UF
     THEN
      Q = 0:
      RN = RN + 1
340 GOTO 270
350 Q = 0:
    RN = 1:
    N2 = 0
360 G = Q * 25
370 FIELD 3,(G)ASDUMMY$,25ASDB$
380 GET 3,RN
390 IF DB$ = STRING$(25,88)
     THEN
      NC = N2:
      GOTO 470
400 N2 = N2 + 1
410 FOR K = 1 TO 25
420  IF MID$(DB$,K,2) = "  "
      THEN
       CN$(N2) = LEFT$(DB$,K - 1):
       GOTO 450
430  NEXT K
440 CN$(N2) = DB$
450 Q = Q + 1:
    IF Q = 10
     THEN
      Q = 0:
      RN = RN + 1
460 GOTO 360
470 CLOSE :
    CN = 0
480 IF PN = 0
     THEN
      PN = 1
490 P$ = STRING$(63,45):
    CN$(0) = "NO CLASS":
    P1$ = "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
    23"
500 IF CA < 11
     THEN
      PT$ = LEFT$(P1$,CA * 2):
     ELSE
      PT$ = LEFT$(P1$,21) + MID$(P1$,22,(CA - 10) * 3)
```

Program continued

```
510 IF TF = 1
     THEN
      2110
520 OPEN "R",3,"TEACHER"
530 G = SR * 25
540 FIELD 3,(G)ASDUMMY$,25ASVN$
550 GET 3,CR
560 IF VN$ = STRING$(25,88)
     THEN
      CLOSE :
      GOTO 1500
570 FOR X1 = 1 TO 25
580  IF MID$(VN$,X1,2) = "  "
      THEN
       TN$ = LEFT$(VN$,X1 - 1):
       GOTO 610
590  NEXT X1
600 TN$ = VN$
610 IF LEFT$(VN$,7) = "DELETED"
     THEN
      620:
     ELSE
      650
620 SR = SR + 1:
    IF SR = 10
     THEN
      SR = 0:
      CR = CR + 1
630 CT = CT + 1
640 GOTO 530
650 SR = SR + 1:
    IF SR = 10
     THEN
      SR = 0:
      CR = CR + 1
660 TN$ = LEFT$(TN$,13)
670 CLOSE
680 Y = ((PN - 1) * 5) + 25:
    Z = Y + 2
690 FOR X1 = 1 TO NS
700  IF VAL( MID$(SN$(X1),Y,2)) = CT
      THEN
       CN = VAL( MID$(SN$(X1),Z,3)):
       GOTO 720
710  GOTO 1250
720  C = C + 1:
     IF C = 1
      THEN
       730:
      ELSE
       760
730  CLS
740  PRINT "TEACHER : ";TN$ TAB(24)"CLASS : "; LEFT$(CN$(CN),16)
     TAB(50)"PERIOD : ";PN
750  PRINT P$
760  PRINT @128, CHR$(31);
770  PRINT "STUDENT :  "; LEFT$(SN$(X1),24)
780  PRINT P$
790  PRINT "DAYS ";PT$
800  AN$ = "":
     PRINT @640,"WAS   "; LEFT$(SN$(X1),24):
     PRINT "PRESENT IN THIS CLASS FOR ALL";CA;:
     LINE INPUT "DAYS ? ( Y/N )  ";AN$:
     GOSUB 2050:
     IF AN$ = "Y"
      THEN
       810:
       :
      ELSE
       IF AN$ < > "N"
```

```
         THEN
          PRINT @640, CHR$(31);:
          GOTO 800:
          :
         ELSE
          920
 810   IF LEN(SA$(X1)) = 0
        THEN
         SA$(X1) = STRING$(CA,"1"):
         GOTO 1250
 820   SD$ = ""
 830   FOR X2 = 1 TO CA
 840    SB = VAL( MID$(SA$(X1),X2,1))
 850    SB = SB + 1
 860    IF SB > 9
         THEN
          SB = 9
 870    SC$ = RIGHT$( STR$(SB),1)
 880    SD$ = SD$ + SC$:
        SC$ = ""
 890    NEXT X2
 900   SA$(X1) = SD$
 910   GOTO 1250
 920   K = 325:
       PC = 1
 930   PRINT @640, CHR$(31);:
       PRINT "PLEASE ENTER THE DAYS ABSENT OR PRESENT BY PRESSING THE
       'A' KEY":
       PRINT "FOR ABSENT OR THE 'P' KEY FOR PRESENT FOR EACH DAY.":
       PRINT "YOU DO NOT HAVE TO PRESS ENTER.":
       PRINT "USE THE "; CHR$(93);" TO BACKSPACE."
 940   PRINT @K, CHR$(95);
 950   V$ = INKEY$
 960   IF V$ = ""
        THEN
         950
 970   IF ASC(V$) = 8 AND K = 325
        THEN
         940
 980   IF ASC(V$) = 8 AND K > 343
        THEN
         PC = PC - 1:
         K = K - 3:
         GOTO 940
 990   IF ASC(V$) = 8 AND K < = 344
        THEN
         PC = PC - 1:
         K = K - 2:
         GOTO 940
1000   IF V$ = "P" OR V$ = "A"
        THEN
         POKE 15360 + K, ASC(V$):
         GOTO 1010:
         :
        ELSE
         GOTO 940
1010   PC = PC + 1
1020   IF K = > 343
        THEN
         K = K + 3:
        ELSE
         K = K + 2
1030   IF PC > CA
        THEN
         1040:
        ELSE
         940
1040   PC = 1:
       PRINT @640, CHR$(31);:
       LINE INPUT "IS THIS DATA CORRECT ? ( Y/N )  ";AN$:
```

Program continued

```
      GOSUB 2050:
      IF AN$ = "Y"
       THEN
        1110:
        :
       ELSE
        IF AN$ < > "N"
         THEN
          1040:
         ELSE
          1050
1050  PRINT @640, CHR$(31);"<ENTER> THE NUMBER OF THE DAY THAT IS INC
      ORRECT ( 1 - ";CA;:
      LINE INPUT " ) ";IC$:
      IC = VAL(IC$)
1060  IF IC < 1 OR IC > CA
       THEN
        1050
1070  K1 = 15685:
      IF IC < 11
       THEN
        K = (IC - 1) * 2:
        K1 = K + K1:
        GOTO 1090
1080  K = 21 + ((IC - 11) * 3):
      K1 = K1 + K
1090  IF PEEK(K1) = 80
       THEN
        POKE K1,65:
       ELSE
        POKE K1,80
1100  GOTO 1040
1110  SD$ = ""
1120  FOR X2 = 15685 TO 15703 STEP 2
1130   IF PEEK(X2) = 80
        THEN
         SB = 1:
        ELSE
         SB = 0
1140   SC = VAL( MID$(SA$(X1),PC,1)):
       SC = SC + SB:
       IF SC > 9
        THEN
         SC = 9
1150   SD$ = SD$ + RIGHT$( STR$(SC),1)
1160   PC = PC + 1:
       IF PC > CA
        THEN
         1240
1170   NEXT X2
1180  FOR X2 = 15706 TO 15742 STEP 3
1190   IF PEEK(X2) = 80
        THEN
         SB = 1:
        ELSE
         SB = 0
1200   SC = VAL( MID$(SA$(X1),PC,1)):
       SC = SC + SB:
       IF SC > 9
        THEN
         SC = 9
1210   SD$ = SD$ + RIGHT$( STR$(SC),1)
1220   PC = PC + 1:
       IF PC > CA
        THEN
         1240
1230   NEXT X2
1240  SA$(X1) = SD$
1250  NEXT X1
1260 IF CN = 0
```

```
      THEN
       CLS :
       PRINT @448,TN$;" HAS NO STUDENTS SCHEDULED FOR PERIOD";PN;".":
       FOR X2 = 1 TO 500:
        NEXT X2
1270 C = 0:
     CN = 0:
     PN = PN + 1:
     IF PN > NP
      THEN
       PN = 1:
       GOTO 1290
1280 CN = 0:
     AN$ = "":
     CLS :
     PRINT @448,"";:
     LINE INPUT "ARE YOU READY TO ENTER THE NEXT PERIOD ? ( Y/N )  ";
     AN$:
     GOSUB 2050:
     IF AN$ = "Y"
      THEN
       680:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         1280:
        ELSE
         1310
1290 CT = CT + 1
1300 AN$ = "":
     CLS :
     PRINT @448,"";:
     LINE INPUT "ARE YOU READY TO ENTER DATA FOR THE NEXT TEACHER ? (
      Y/N )  ";AN$:
     GOSUB 2050:
     IF AN$ = "Y"
      THEN
       520:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         1300:
        ELSE
         1310
1310 NR = (256 / CA) - 1
1320 Q = 0:
     RN = 1:
     UR = 0:
     TF = 1
1330 OPEN "R",2,"TATTEND"
1340 FOR X = 1 TO NS
1350  G = Q * CA
1360  FIELD 2,(G)ASDUMMY$,(CA)ASDV$
1370  LSET DV$ = SA$(X)
1380  IF UR = 1
       THEN
        1420
1390  Q = Q + 1:
      IF Q = NR
       THEN
        Q = 0:
        PUT 2,RN:
        RN = RN + 1
1400  NEXT X
1410 SA$(X) = STRING$(CA,88):
     UR = 1:
     GOTO 1350
1420 PUT 2,RN
```

Program continued

```
1430 OPEN "R",1,"DATTEND"
1440 FIELD 1,2ASXA$,2ASXB$,2ASXC$,2ASXD$,2ASXE$,2ASXF$,2ASXG$,2ASXH$,
     2ASXI$,2ASXJ$,2ASXK$
1450 GET 1,1
1460 LSET XA$ = MKI$ (TA):
     LSET XB$ = MKI$ (CA):
     LSET XD$ = MKI$ (CR):
     LSET XE$ = MKI$ (SR):
     LSET XF$ = MKI$ (NS):
     LSET XG$ = MKI$ (CT):
     LSET XH$ = MKI$ (AT):
     LSET XI$ = MKI$ (TF):
     LSET XJ$ = MKI$ (PN):
     LSET XK$ = MKI$ (AV)
1470 PUT 1,1
1480 CLOSE
1490 RUN "CLASMENU"
1500 CLS :
     PRINT @448,"THAT COMPLETES THE TEACHER FILE.":
     PRINT "I AM NOW COMPUTING THE ATTENDANCE DATA."
1510 DIM SA(NS + 15)
1520 FOR X = 1 TO NS
1530  AC = 0
1540  FOR Y = 1 TO CA
1550   AC$ = MID$(SA$(X),Y,1):
       IF VAL(AC$) = > AV
        THEN
         AC = AC + 2:
         GOTO 1570
1560   IF VAL(AC$) > 0 AND VAL(AC$) < AV
        THEN
         AC = AC + 1
1570   NEXT Y
1580  SA(X) = AC
1590  NEXT X
1600 CLS :
     PRINT @448,"I AM WRITING THE DATA TO DISK NOW."
1610 IF TA = 0
      THEN
       X1 = NS:
       GOTO 1800
1620 OPEN "R",1,"DATTEND"
1630 RN = 2:
     X1 = 1
1640 NX = ( LOF (1) - 1) * 64:
     NX = NX + 5
1650 DIM SB(NX)
1660 X = 1
1670 FIELD 1,128ASRV$(1),128ASRV$(2)
1680 GET 1,RN
1690 FOR Y = 1 TO 128 STEP 4
1700  IF VAL( MID$(RV$(X),Y,4)) = 999
       THEN
        1760
1710  SB(X1) = VAL( MID$(RV$(X),Y,4))
1720  X1 = X1 + 1
1730  NEXT Y
1740 IF X = 1
      THEN
       X = 2:
       GOTO 1690
1750 RN = RN + 1:
     GOTO 1660
1760 CLOSE
1770 FOR X = 1 TO NS
1780  SA(X) = SA(X) + SB(X)
1790  NEXT X
1800 OPEN "R",1,"DATTEND"
1810 X = 1:
     RN = 2
```

```
1820 SA(X1 + 1) = 999:
     CW$ = ""
1830 FIELD 1,128ASRV$(1),128ASRV$(2)
1840 GET 1,RN
1850 FOR X2 = 1 TO X1 + 1
1860  IF SA(X2) < 10
       THEN
        CW$ = CW$ + "00" + STR$(SA(X2)):
        GOTO 1880
1870  IF SA(X2) < 100
       THEN
        CW$ = CW$ + "0" + STR$(SA(X2)):
        GOTO 1880:
       ELSE
        CW$ = CW$ + STR$(SA(X2))
1880  IF LEN(CW$) = 128
       THEN
        1890:
       ELSE
        1920
1890  LSET RV$(X) = CW$:
      CW$ = ""
1900  IF X = 1
       THEN
        X = 2:
        GOTO 1920
1910  X = 1:
      PUT 1,RN:
      RN = RN + 1:
      GET 1,RN:
      GOTO 1920
1920  NEXT X2
1930 LSET RV$(X) = CW$
1940 PUT 1,RN
1950 TA = TA + CA:
     CA = 0:
     CR = 0:
     SR = 0:
     CT = 1:
     AT = 0:
     TF = 2:
     PN = 0
1960 OPEN "R",2,"TATTEND"
1970 RN = LOF (2)
1980 A$ = ""
1990 FOR X1 = 1 TO RN
2000  FIELD 2,255ASDUMMY$
2010  LSET DU$ = A$
2020  PUT 2,X1
2030  NEXT X1
2040 GOTO 1440
2050 AN$ = LEFT$(AN$,1):
     RETURN
2060 CLS :
     PRINT @394,"AN ERROR HAS OCCURRED IN THE EXECUTION OF THE PROGRA
     MCALLED 'TEACHER ATTENDANCE INPUT'."
2070 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
2080 PRINT TAB(5)"ERROR LINE = "; ERL
2090 FOR V = 1 TO 5000:
      NEXT V
2100 STOP
2110 OPEN "R",2,"TATTEND"
2120 RN = 1:
     Q = 0:
     NR = (256 / CA) - 1:
     X = 0
2130 G = Q * CA
2140 FIELD 2,(G)ASDUMMY$,(CA)ASDW$
2150 GET 2,RN
2160 IF DW$ = STRING$(CA,88)
```

Program continued

```
      THEN
       CLOSE :
       GOTO 2210
2170 X = X + 1
2180 SA$(X) = DW$
2190 Q = Q + 1:
     IF Q = NR
      THEN
       Q = 0:
       RN = RN + 1
2200 GOTO 2130
2210 IF PN = 1
      THEN
       520
2220 IF SR = 0 AND CR = > 2
      THEN
       SR = 9:
       CR = CR - 1:
       GOTO 520
2230 SR = SR - 1:
     GOTO 520
```

Program Listing 3. *Class schedule by student printout*

```
 10 :
    '  STUDENT SCHEDULE PRINTOUT ( PNTSTCHD )
 20 :
    '  COPYRIGHT OCTOBER 1, 1981
 30 :
    '  ULDERIC F. RACINE
 40 :
    '  2520 S.E. ALEXANDER DRIVE
 50 :
    '  TOPEKA, KANSAS 66605
100 CLS :
    PRINT CHR$(23)
110 PRINT @446,"       STUDENT PRINTOUT"
120 PRINT :
    PRINT "  PERIOD / TEACHER / CLASS"
130 CLEAR 3000
140 OPEN "R",2,"TEACHER":
    RO = LOF (2) * 10
150 OPEN "R",3,"CLASSES":
    RP = LOF (3) * 10
160 IF UR = 0
     THEN
      CLOSE :
      GOTO 230
170 OPEN "R",1,"STDSCHED"
180 FIELD 1,2ASX1$,2ASX2$,2ASX3$,2ASX4$,2ASX5$,2ASX6$,2ASX7$,2ASX8$,
    240ASX9$:
    GET 1,1
190 T = CVI (X1$):
    FS = CVI (X2$):
    UF = CVI (X3$):
    NX = CVI (X4$):
    NY = CVI (X5$):
    NP = CVI (X6$):
    RN = CVI (X7$):
    Q = CVI (X8$)
200 DIM CN$(RP),TN$(RO),CN(NP),CT(NP)
210 TN$(0) = "NO TEACHER":
    CN$(0) = "NO CLASS"
220 IF UR = 1
     THEN
      260
```

```
230 T = (RO * 25) + (RP * 25) + 3000
240 CLEAR T
250 UR = 1:
    GOTO 140
260 Q1 = 0:
    RO = 1:
    X = 0
270 ON ERROR GOTO 1170
280 G = Q1 * 25
290 FIELD 2,(G)ASDX$,25ASDY$
300 GET 2,RO
310 IF DY$ = STRING$(25,88)
     THEN
      360
320 X = X + 1
330 TN$(X) = DY$
340 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RO = RO + 1
350 GOTO 280
360 Q1 = 0:
    RP = 1:
    X = 0
370 G = Q1 * 25
380 FIELD 3,(G)ASDX$,25ASDY$
390 GET 3,RP
400 IF DY$ = STRING$(25,88)
     THEN
      CLOSE :
      GOTO 800
410 X = X + 1
420 CN$(X) = DY$
430 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RP = RP + 1
440 GOTO 370
450 OPEN "R",1,"STDSCHED"
460 G = FS * Q
470 FIELD 1,(G)ASDZ$,(FS)ASNS$
480 GET 1,RN
490 IF NS$ = STRING$(FS,88)
     THEN
      CLOSE :
      GOTO 800
500 SN$ = LEFT$(NS$,24)
510 Y = 25:
    Z = 27
520 FOR X = 1 TO NP
530  CT(X) = VAL( MID$(NS$,Y,2))
540  CN(X) = VAL( MID$(NS$,Z,3))
550  Y = Y + 5:
     Z = Z + 5
560  NEXT X
570 IF UR = 3
     THEN
      GOTO 600
580 Q = Q + 1:
    IF Q = UF
     THEN
      Q = 0:
      RN = RN + 1
590 CLOSE
600 CLS :
    PS$ = STRING$(60,45)
610 PRINT PS$:
    PRINT SN$:
```

Program continued

```
    PRINT PS$:
    IF HC = 1
     THEN
      620:
     ELSE
      650
620 LPRINT PS$
630 LPRINT SN$
640 LPRINT PS$
650 PRINT "PERIOD" TAB(10)"TEACHER" TAB(35)"CLASSES":
    PRINT PS$:
    IF HC = 1
     THEN
      660:
     ELSE
      680
660 LPRINT "PERIOD" TAB(10)"TEACHER" TAB(35)"CLASSES"
670 LPRINT PS$
680 FOR X = 1 TO NP
690  PRINT TAB(2)X TAB(10) LEFT$(TN$(CT(X)),20) TAB(35) LEFT$(CN$(CN
     (X)),20):
     IF HC = 1
      THEN
       700:
      ELSE
       720
700  LPRINT TAB(2)X TAB(10) LEFT$(TN$(CT(X)),20) TAB(35) LEFT$(CN$(C
     N(X)),20)
710  GOTO 730
720  IF X = 7 AND NP > 7
      THEN
       PRINT :
       LINE INPUT "PRESS <ENTER> TO CONTINUE ";AN$:
       PRINT @320, CHR$(31);
730  NEXT X
740 PRINT PS$:
    IF HC = 1
     THEN
      750:
     ELSE
      760
750 LPRINT PS$:
    LPRINT " "
760 IF UR = 3
     THEN
      UR = 0:
      GOTO 1050
770 IF PX = 1
     THEN
      790
780 PRINT :
    LINE INPUT "PRESS <ENTER> TO CONTINUE";A$
790 GOTO 450
800 CLS
810 PRINT TAB(10)"STUDENT SCHEDULE PRINTOUT":
    PRINT :
    PRINT "OPTIONS : "
820 PRINT :
    PRINT "   1 - PRINT SCHEDULE FOR ALL STUDENTS"
830 PRINT "   2 - PRINT SCHEDULE FOR A SPECIFIC STUDENT"
840 PRINT "   3 - RETURN TO MASTER MENU"
850 PRINT :
    LINE INPUT "<ENTER> OPTION SELECTED : ";OP$:
    OP = VAL(OP$)
860 IF OP < 1 OR OP > 3
     THEN
      800
870 IF OP = 3
     THEN
      1160
```

```
 880 RN = 2:
     Q = 0:
     UR = 0:
     HC = 0:
     PX = 0:
     GOSUB 1090
 890 ON OP GOTO 450,900,1160
 900 CLS :
     UR = 0:
     RN = 2:
     Q = 0
 910 PRINT @448,"PLEASE ENTER THE NAME OF THE STUDENT WHOSE SCHEDULE
     YOU WISH TO PRINT. IT SHOULD BE EXACTLY AS ENTEREDON THE STUDENT
      SCHEDULE FILE."
 920 INPUT "<ENTER> STUDENT'S NAME : ";SN$
 930 K = LEN(SN$):
     IF K < = 0
      THEN
       900
 940 OPEN "R",1,"STDSCHED"
 950 G = Q * FS
 960 FIELD 1,(G)ASDZ$,(FS)ASNS$
 970 GET 1,RN
 980 IF NS$ = STRING$(FS,88)
      THEN
       CLOSE :
       GOTO 1020
 990 IF SN$ = LEFT$(NS$,K)
      THEN
       CLOSE :
       UR = 3:
       GOTO 510
1000 Q = Q + 1:
     IF Q = UF
      THEN
       Q = 0:
       RN = RN + 1
1010 GOTO 950
1020 CLS :
     PRINT @448,"I CAN NOT FIND A STUDENT NAMED ";SN$:
     PRINT "IN MY STUDENT FILES. ARE YOU SURE THE NAME IS THE SAMEAS
     IT WAS ENTERED ?"
1030 PRINT :
     LINE INPUT "SHALL WE TRY AGAIN ( Y/N ) ";AN$:
     AN$ = LEFT$(AN$,1):
     IF AN$ = "Y"
      THEN
       900
1040 IF AN$ < > "N"
      THEN
       1020:
      ELSE
       800
1050 PRINT @704, CHR$(31);:
     HC = 0:
     PX = 0
1060 PRINT :
     LINE INPUT "DO YOU HAVE ANOTHER STUDENT WHOSESCHEDULE YOU WISH T
     O PRINTOUT ( Y/N ) ";AN$
1070 AN$ = LEFT$(AN$,1)
1080 IF AN$ = "Y"
      THEN
      GOSUB 1090:
      GOTO 900:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        1050:
       ELSE
```

Program continued

```
        800
1090 CLS :
     PRINT @448,"";:
     LINE INPUT "DO YOU WANT A HARDCOPY ( Y/N ) ";AN$:
     AN$ = LEFT$(AN$,1):
     IF AN$ = "Y"
      THEN
       1100:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         1090:
        ELSE
         RETURN
1100 HC = 1
1110 PRINT @448, CHR$(31);:
     LINE INPUT "SHALL I GENERATE A TEST LINE FOR THE PRINTER ( Y/N )
      ";AN$:
     AN$ = LEFT$(AN$,1):
     IF AN$ = "Y"
      THEN
       1120:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         1110:
        ELSE
         1130
1120 LPRINT STRING$(60,88):
     GOTO 1110
1130 IF OP = 1
      THEN
       PRINT @448, CHR$(31);:
       LINE INPUT "SHALL I PAUSE BETWEEN PRINTING SCHEDULES ( Y/N ) "
       ;AN$:
       AN$ = LEFT$(AN$,1):
       IF AN$ = "N"
        THEN
         1150:
         :
        ELSE
         IF AN$ < > "Y"
          THEN
           1130:
          ELSE
           RETURN
1140 RETURN
1150 PX = 1:
     RETURN
1160 RUN "CLASMENU"
1170 CLS :
     PRINT @394,"AN ERROR HAS OCCURRED IN THE EXEXCUTION OF THE PROGR
     AM'STUDENT SCHEDULE PRINTOUT'."
1190 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
1200 PRINT TAB(5)"ERROR LINE = "; ERL
1210 FOR V = 1 TO 5000:
      NEXT V
1220 STOP
```

Program Listing 4. *Printout by class name*

```
10 :
   '  PRINTOUT STUDENT/TEACHER/PERIOD BY CLASS ( PNTCLASS )
20 :
   '  COPYRIGHT OCTOBER 1, 1981
```

```
 30 :
    '  ULDERIC F. RACINE
 40 :
    '  2520 S.E. ALEXANDER DRIVE
 50 :
    '  TOPEKA, KANSAS 66605
100 CLS :
    PRINT CHR$(23):
    PRINT "        CLASS PRINT  STUDENT / TEACHER / PERIOD "
110 OPEN "R",1,"STDSCHED":
    FIELD 1,2ASX1$,2ASX2$,2ASX3$,2ASX4$,2ASX5$,2ASX6$,2ASX7$,2ASX8$:
    GET 1,1
120 T = CVI (X1$):
    FS = CVI (X2$):
    UF = CVI (X3$):
    NX = CVI (X4$):
    NY = CVI (X5$):
    NP = CVI (X6$):
    RN = CVI (X7$):
    Q = CVI (X8$)
130 X = LOF (1) * UF
140 OPEN "R",3,"TEACHER":
    RO = LOF (3) * 10
150 IF UR = 1
     THEN
      170
160 T = (X * FS) + (RO * 25) + 3000:
    CLOSE :
    CLEAR T:
    UR = 1:
    GOTO 110
170 CLOSE :
    UR = 0:
    ON ERROR GOTO 1020
180 DIM SN$(X),TN$(RO),NS(X + 10),PC(X + 10),NP(X + 10)
190 RN = 2:
    RO = 1:
    RP = 1
200 OPEN "R",1,"STDSCHED"
210 G = Q * FS
220 FIELD 1,(G)ASDU$,(FS)ASDA$:
    GET 1,RN
230 IF DA$ = STRING$(FS,88)
     THEN
      CLOSE :
      GOTO 270
240 N2 = N2 + 1:
    SN$(N2) = DA$
250 Q = Q + 1:
    IF Q = UF
     THEN
      Q = 0:
      RN = RN + 1
260 GOTO 210
270 OPEN "R",3,"TEACHER"
280 G = Q1 * 25:
    FIELD 3,(G)ASDV$,25ASDB$:
    GET 3,RO
290 IF DB$ = STRING$(25,88)
     THEN
      CLOSE :
      GOTO 620
300 N3 = N3 + 1:
    TN$(N3) = DB$
310 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RO = RO + 1
320 GOTO 280
```

Program continued

```
330 OPEN "R",2,"CLASSES"
340 G = Q2 * 25:
    FIELD 2,(G)ASDV$,25ASDC$:
    GET 2,RP
350 IF DC$ = STRING$(25,88)
     THEN
      CLOSE :
      GOTO 620
360 CN$ = DC$
370 Q2 = Q2 + 1:
    IF Q2 = 10
     THEN
      Q2 = 0:
      RP = RP + 1
380 CLOSE
390 N1 = N1 + 1
400 Y = 25:
    Z = 27
410 FOR X = 1 TO N2
420   IF VAL( MID$(SN$(X),Z,3)) = N1
       THEN
        430:
       ELSE
        440
430   SC = SC + 1:
      NS(SC) = X:
      NP(SC) = INT((Y - 24) / 5) + 1:
      PC(SC) = VAL( MID$(SN$(X),Y,2))
440   IF Z + 2 = FS
       THEN
        450:
       ELSE
        Y = Y + 5:
        Z = Z + 5:
        GOTO 420
450   Y = 25:
      Z = 27:
      NEXT X
460 CLS
470 P$ = STRING$(61,45)
480 PRINT P$:
    PRINT "CLASS : ";CN$:
    PRINT "STUDENT" TAB(27)"TEACHER" TAB(55)"PERIOD":
    PRINT P$
490 IF HC = 1
      THEN
       LPRINT P$:
       LPRINT "CLASS : ";CN$:
       LPRINT "STUDENT" TAB(27)"TEACHER" TAB(55)"PERIOD":
       LPRINT P$:
       IF SC = 0
        THEN
         LPRINT "NO STUDENTS CURRENTLY ENROLLED":
         GOTO 550
500 IF SC = 0
      THEN
       PRINT "NO STUDENTS CURRENTLY ENROLLED":
       GOTO 550
510 FOR X = 1 TO SC
520   PRINT LEFT$(SN$(NS(X)),24) TAB(27)TN$(PC(X)) TAB(57)NP(X)
530   IF HC = 1
       THEN
        LPRINT LEFT$(SN$(NS(X)),24) TAB(27)TN$(PC(X)) TAB(57)NP(X)
540   NEXT X
550 PRINT P$
560 IF HC = 1
      THEN
       LPRINT P$:
       LPRINT " "
570 IF PX = 1
```

```
      THEN
       590
580 INPUT "PRESS <ENTER> TO CONTINUE";A$
590 SC = 0
600 IF UR = 1
      THEN
       UR = 0:
       GOTO 760
610 GOTO 330
620 CLS
630 PRINT "PRINT BY CLASS TITLE":
    PRINT @128,"OPTIONS : ":
    PRINT @256,"1 - PRINT STUDENTS BY CLASS NAME FOR ALL CLASSES":
    PRINT "2 - PRINT STUDENTS BY CLASS NAME FOR A SPECIFIC CLASS"
640 PRINT "3 - EXIT THIS PROGRAM":
    PRINT :
    PRINT :
    LINE INPUT "<ENTER> OPTION SELECTED : ";OP$:
    OP = VAL(OP$):
    IF OP < 1 OR OP > 3
      THEN
       640
645 IF OP = 3
      THEN
       930
650 HC = 0:
    PX = 0:
    GOSUB 940
660 ON OP GOTO 330,670
670 CLS
680 PRINT @448,"";:
    LINE INPUT "<ENTER> CLASS NAME YOU WISH TO PRINT : ";CN$:
    K = LEN(CN$)
690 OPEN "R",3,"CLASSES":
    Q1 = 0:
    R0 = 1:
    N1 = 0
700 G = Q1 * 25:
    FIELD 3,(G)ASDY$,25ASDA$:
    GET 3,R0
710 IF DA$ = STRING$(25,88)
      THEN
       CLOSE :
       GOTO 790
720 N1 = N1 + 1
730 IF LEFT$(DA$,K) = CN$
      THEN
       CLOSE :
       UR = 1:
       GOTO 400
740 Q1 = Q1 + 1:
    IF Q1 = 10
      THEN
       Q1 = 0:
       R0 = R0 + 1
750 GOTO 700
760 AN$ = "":
    CLS
770 PRINT @448,"";:
    LINE INPUT "DO YOU HAVE ANOTHER CLASS NAME YOU WISH TO PRINTOUT
    ? ( Y/N ) ";AN$
780 GOSUB 1010:
    IF AN$ = "Y"
      THEN
       670:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         760:
```

Program continued

```
       ELSE
        620
790 CLS :
    PRINT @448,"I CANNOT FIND A CLASS NAMED ";CN$:
    PRINT "IN MY CLASS FILE. "
800 PRINT :
    LINE INPUT "SHALL WE TRY AGAIN ? ( Y/N ) ";AN$:
    GOSUB 1010
810 IF AN$ = "Y"
     THEN
      820:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        790:
       ELSE
        620
820 AN$ = "":
    CLS :
    PRINT @448,"WOULD YOU LIKE TO SEE A LIST OF CLASSES":
    LINE INPUT "CURRENTLY ON FILE ?  ( Y/N ) ";AN$:
    GOSUB 1010
830 IF AN$ = "Y"
     THEN
      840:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        820:
       ELSE
        670
840 OPEN "R",2,"CLASSE OPEN :RP = 1:Q1 = 0
850 CLS :
    JX = 0
860 G = Q1 * 25:
    FIELD 2,(G)ASDX$,25ASDG$:
    GET 2,RP
870 IF DG$ = STRING$(25,88)
     THEN
      CLOSE :
      GOTO 920
880 JX = JX + 1
890 PRINT DG$,
900 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RP = RP + 1
910 IF JX < 25
     THEN
      GOTO 860:
     ELSE
      PRINT :
      LINE INPUT "PRESS <ENTER> TO CONTINUE ";AN$:
      JX = 0:
      CLS :
      GOTO 860
920 PRINT :
    LINE INPUT "PRESS <ENTER> TO CONTINUE ";AN$:
    GOTO 670
930 RUN "CLASMENU"
940 CLS :
    PRINT @448,"";:
    LINE INPUT "DO YOU WANT A HARDCOPY ? ( Y/N ) ";AN$:
    GOSUB 1010:
    IF AN$ = "Y"
     THEN
      950:
```

```
      :
     ELSE
      IF AN$ < > "N"
       THEN
        940:
       ELSE
        RETURN
950 HC = 1
960 PRINT @448, CHR$(31);:
    LINE INPUT "SHALL I GENERATE A TEST LINE FOR THE PRINTER ? ( Y/N
     ) ";AN$:
    GOSUB 1010:
    IF AN$ = "Y"
     THEN
      970:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        960:
       ELSE
        980
970 LPRINT STRING$(60,88):
    GOTO 960
980 IF OP = 1
     THEN
      PRINT @448, CHR$(31);:
      LINE INPUT "SHALL I STOP BETWEEN PRINTING CLASSES ? ( Y/N )";A
      N$:
      GOSUB 1010:
      IF AN$ = "N"
       THEN
        1000:
        :
       ELSE
        IF AN$ < > "Y"
         THEN
          980:
         ELSE
          RETURN
990 RETURN
1000 PX = 1:
     RETURN
1010 AN$ = LEFT$(AN$,1):
     RETURN
1020 CLS :
     PRINT @394,"AN ERROR HAS OCCURRED IN THE EXECUTION OF THE PROGRA
     M 'CLASS PRINTOUT'."
1030 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
1040 PRINT TAB(5)"ERROR LINE = "; ERL
1050 FOR V = 1 TO 5000:
      NEXT V
1060 STOP
```

Keeping Track— Student Scheduling and Attendance Part IV

by Ulderic F. Racine

This part of the series contains the final three programs. Before proceeding, I suggest that you consider disabling the BREAK key in the data input programs. It is crucial to the operation of the programs that the only exit is through option 0 on the master menu. Each of the files the programs create is terminated by an end-of-file marker of Xs equal to the length of a record. If you break the program before this marker has been set, a SUBSCRIPT OUT OF RANGE error occurs the next time you run the program.

To disable the BREAK key on the Model III, add a CMD "B" near the beginning of the program. I use line 99 on the Model I. If you are running under NEWDOS 80, use POKE 17257,0 instead of the CMD function. I am not aware of any disables that work with TRSDOS 2.3 or NEWDOS 2.1. If you use one of these operating systems, warn the operator to exit only through the master menu. I also suggest that you try a test run once you have entered all the programs to identify any bugs. There are error routines in each program that identify the type of error and line number. You should test all the functions and options before you go into production.

Program Listing 1 is the attendance input by student program (STDATEND). It operates in the same manner as the input by teacher program in Part III. The program displays each student as entered on the schedule file. You enter data for each class period. The following is an example of the display screen.

```
STUDENT: JOHNSON FRED

____________________________________

PERIOD     TEACHER     CLASS

____________________________________

   1          JONES       MATH

____________________________________

DAYS 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

WAS JOHNSON FRED PRESENT IN THIS CLASS
FOR ALL 15 DAYS (Y/N) ?
```

If the student was present for the number of attendance days you enter, type Y and press ENTER. If the student was not present for the entire attendance period, the program displays the cursor under the first day. If the student was present, type P. If the student was absent, type A. There is no need to press ENTER. If you make a mistake you can back space by using the left-arrow key.

After you enter the data for each day, you have a final chance to correct any errors. The program asks if the data is correct. Type N if there is a mistake, and the program asks for the incorrect number. Type the number that is incorrect and press ENTER. The correction is made on the screen display, and the program again asks if the data is correct. If the data is correct, type Y and press ENTER. The next period appears, and when you have entered the data for that student, the program asks if you have another student to enter. You must respond with either a Y or an N.

Program Listing 2 is the teacher record change program (TEACHANG). This program allows you to change the name of a teacher previously entered in the teacher file through one of the scheduling options and replace it with another name, delete it, or alter the spelling. The program asks you to select

CLASSES	Contains all class names entered by schedule input. 10 class names per logical record. End of file designated by a string of 25 Xs.
TEACHER	Contains all teacher names entered through schedule input. 10 teacher names per logical record. End of file designated by a string of 25 Xs.
STDSCHED	Contains the names of all students. Each teacher/class entry is contained in a five-byte string. First logical record contains file management information. Student data begins with the second logical record. Number of students per logical record is determined by the number of class periods. The range is 1–4 student records per logical record. End of file designated by a string of 25 Xs.
DATTEND	Contains all full-time equivalent attendance data on students. First logical record contains file management information. Full-time equivalent attendance data begins with second logical record. 64 subrecords per logical record. End of file designated by 999.
TATTEND	Contains temporary attendance data on students when data is entered by teacher. This file is used until an attendance cycle is completed. The size of each record is equal to the number of days of attendance being entered in the current cycle.

Table 1. *Files created by Keeping Track*

T	Data being entered by teacher (1) or student (2)
FS	Number of bytes allocated to each student record
UF	Number of subrecords per logical record
NT	Number of teachers to be entered
NC	Number of classes to be entered
NP	Number of class periods per day
RN	Current logical record position
Q	Current position of subrecord
SN	Estimated number of students to be entered
PN	Current period being entered if input by teacher

Table 2. *STDSCHED file structure*

TA	Total full-time attendance to date for all students. Data is stored in half-day increments.
CA	Current number of attendance days being entered in the present attendance cycle.
SD	Number of next student to be entered on the attendance file if the input is by student.
CR	Next logical record number to be used for attendance input. If the input is by student, the record specified is the STDSCHED file. If the input is by teacher, the record specified is in the TEACHER file.
SR	This is the subrecord multiplier for either type of input.
NS	The number of students currently on file. This is used only when input is by student.
CT	The number of the teacher for whom attendance is currently being gathered. This is used only when input is by teacher.
AT	The type of attendance being entered currently. 1 = teacher. 2 = student.
TF	Indicates whether the temporary file TATTEND has been used. 1 = yes. 2 = no.
PN	Indicates the current period number for which attendance data by teacher is collected.
AV	The number of class periods per day that constitute full-time attendance per day.

Table 3. *DATTEND file structure*

one of the options. It displays the names of the first 20 teachers in the teacher file on the screen and asks if the teacher whose name you wish to replace, delete, or change is listed. If your response is YES, the program asks for the number of the teacher. Type the number and press ENTER. If you have entered a valid number, the program allows you to replace, delete, or change the spelling of that teacher's name. If you answer that the name of the teacher is not displayed, the computer displays the next 20 teachers, and so on, until it has displayed all the names or you have given a Y response.

Program Listing 3 is the year-to-date attendance printout program (PNTATTEND) which prints year-to-date attendance data stored in the attendance file. You can run this program at any time after one attendance cycle has been completed. The program gives you three choices. First, you can print out the attendance file for all students. The program prints the name of the student and the number of full-time equivalent attendance days currently on file. Each page of listing has a subtotal, and the final page shows the total full-time attendance and the total number of students. You can print just the totals, in which case the program prints only the total number of days of attendance on file, the total number of students, and the total number of full-time equivalent attendance days. The third option allows you to print out the attendance total for a specific student.

Table 1 is a summary of the files the programs create. Tables 2 and 3 are summaries of the program management variables stored in the first logical record of the schedule and attendance files.

Program Listing 1. *Attendance input by student*

```
 10 :
    '  ATTENDANCE INPUT BY STUDENT ( STDATEND )
 20 :
    '  COPYRIGHT OCTOBER 1, 1981
 30 :
    '  ULDERIC F. RACINE
 40 :
    '  2520 S.E. ALEXANDER DRIVE
 50 :
    '  TOPEKA, KANSAS 66605
100 CLS :
    PRINT CHR$(23):
    PRINT @452,"ATTENDANCE INPUT BY STUDENT"
110 OPEN "R",2,"TEACHER":
    RO = LOF (2)
120 OPEN "R",3,"CLASSES":
    RP = LOF (3)
130 IF UR = 0
     THEN
      CLOSE :
      T = (RO * 25) + (RP * 25) + 3000:
      CLEAR T:
      UR = 1:
      GOTO 110
140 ON ERROR GOTO 1620
150 DIM TN$(RO * 10),CN$(RP * 10)
160 Q1 = 0:
    RO = 1:
    X = 0
170 G = Q1 * 25
180 FIELD 2,(G)ASDX$,25ASDV$
190 GET 2,RO
200 IF DV$ = STRING$(25,88)
     THEN
      250
210 X = X + 1
220 TN$(X) = DV$
230 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RO = RO + 1
240 GOTO 170
250 Q1 = 0:
    RP = 1:
    X = 0
260 G = Q1 * 25
270 FIELD 3,(G)ASDX$,25ASDV$
280 GET 3,RP
290 IF DV$ = STRING$(25,88)
     THEN
      340
300 X = X + 1
310 CN$(X) = DV$
320 Q1 = Q1 + 1:
    IF Q1 = 10
     THEN
      Q1 = 0:
      RP = RP + 1
330 GOTO 260
340 OPEN "R",1,"STDSCHED"
350 FIELD 1,2ASY1$,2ASY2$,2ASY3$,2ASY4$,2ASY5$,2ASY6$,2ASY7$,2ASY8$
360 GET 1,1
370 FS = CVI (Y2$):
    UF = CVI (Y3$):
    NP = CVI (Y6$)
375 NS = ( LOF (1) - 1) * UF
```

```
380 CLOSE
390 DIM SA(NS),SB(NS)
400 OPEN "R",2,"DATTEND"
410 FIELD 2,2ASXA$,2ASXB$,2ASXC$,2ASXD$,2ASXE$,2ASXF$,2ASXG$,2ASXH$,
    2ASXI$,2ASXJ$,2ASXK$
420 GET 2,1
430 TA = CVI (XA$):
    CA = CVI (XB$):
    SD = CVI (XC$)
440 RQ = CVI (XD$):
    QA = CVI (XE$):
    NT = CVI (XG$):
    AV = CVI (XK$)
450 IF TA = 0 AND RQ = 2 AND QA = 0
     THEN
      CLOSE :
      GOTO 550
460 Q1 = 0:
    RS = 2:
    X = 0
470 G = Q1 * 4
480 FIELD 2,(G)ASDZ$,4ASZA$
490 GET 2,RS
500 IF VAL(ZA$) = 999
     THEN
      CLOSE :
      GOTO 550
510 X = X + 1
520 SA(X) = VAL(ZA$):
    ZA$ = ""
530 Q1 = Q1 + 1:
    IF Q1 = 64
     THEN
      Q1 = 0:
      RS = RS + 1
540 GOTO 470
550 IF NP > 10
     THEN
      DIM CT(NP),CN(NP)
560 DIM DP(CA)
570 PS$ = STRING$(60,45):
    Y1 = 15420:
    U$ = "NO TEACHER":
    U1$ = "NO CLASS":
    UR = 0:
    UZ = 0
580 PT$ = "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
    23"
590 OPEN "R",1,"STDSCHED"
600 G = QA * FS
610 FIELD 1,(G)ASDZ$,(FS)ASNS$
620 GET 1,RQ
630 IF NS$ = STRING$(FS,88)
     THEN
      CLOSE :
      UR = 1:
      CLS :
      PRINT @448,"THAT IS ALL THE STUDENTS ON THE FILE.":
      PRINT "I WILL NOW WRITE THE DATA TO DISK.":
      GOTO 1340
640 SN$ = LEFT$(NS$,24):
   FOR X = 1 TO 23
650  IF ASC( MID$(SN$,X,1)) < > 32
      THEN
       670
660  IF ASC( MID$(SN$,X + 1,1)) = 32
      THEN
       SN$ = LEFT$(SN$,X):
       GOTO 680
670  NEXT X
```

Program continued

```
680 Y = 25:
    Z = 27
690 FOR X2 = 1 TO NP
700  CT(X2) = VAL( MID$(NS$,Y,2))
710  CN(X2) = VAL( MID$(NS$,Z,3))
720  Y = Y + 5:
     Z = Z + 5
730  NEXT X2
740 CLOSE
750 FOR X2 = 1 TO NP
760  IF CT(X2) = 0 AND CN(X2) = 0
      THEN
       770:
      ELSE
       800
770  NEXT X2
780 SD = SD + 1:
    QA = QA + 1:
    IF QA = UF
     THEN
      QA = 0:
      RQ = RQ + 1
790 GOTO 590
800 CLS :
    PRINT "STUDENT : ";SN$
810 PRINT PS$
820 FOR X = 1 TO NP
830  PRINT "PERIOD : ";X; TAB(15)"TEACHER : "; LEFT$(TN$(CT(X)),15);
      TAB(35)"CLASS : "; LEFT$(CN$(CN(X)),15)
840  PRINT PS$
850  PRINT "DAYS ";
860  IF CA < 11
      THEN
       PT$ = LEFT$(PT$,CA * 2):
       GOTO 880
870  PT$ = LEFT$(PT$,21) + MID$(PT$,22,(CA - 10) * 3)
880  PRINT PT$
890  IF CT(X) = 0
      THEN
       PRINT TAB(10)U$;
900  IF CN(X) = 0
      THEN
       PRINT TAB(31)U1$:
       FOR K = 1 TO 250:
        NEXT K:
       GOTO 1240
910  PRINT @640,"WAS ";SN$;"PRESENT IN THIS CLASS ":
     PRINT "FOR ALL";CA;:
     LINE INPUT "DAYS ? ( Y/N ) ";AN$:
     GOSUB 1670:
     IF AN$ = "Y"
      THEN
       930:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         PRINT @640, CHR$(31);:
         GOTO 910
920  GOTO 970
930  FOR K = 1 TO CA
940   DP(K) = DP(K) + 1
950   NEXT K
960  GOTO 1230
970  PRINT @640, CHR$(31);:
     PRINT "PLEASE ENTER THE DAYS ABSENT OR PRESENT BY PRESSING THE
     'A' KEY":
     PRINT "FOR ABSENT OF THE 'P' KEY FOR PRESENT FOR EACH DAY.":
     PRINT "YOU DO NOT HAVE TO PRESS <ENTER>.":
     PRINT "USE THE "; CHR$(93);" TO BACKSPACE."
```

```
 980  K = 325:
      PC = 0
 990  PRINT @K, CHR$(95);
1000  V$ = INKEY$
1010  IF V$ = ""
       THEN
        1000
1015  IF ASC(V$) = 8 AND K = 325
       THEN
        990
1020  IF ASC(V$) = 8 AND K > 343
       THEN
        PC = PC - 1:
        K = K + 3:
        GOTO 990
1030  IF ASC(V$) = 8 AND K < = 344
       THEN
        PC = PC - 1:
        K = K - 2:
        GOTO 990
1040  IF V$ = "P" OR V$ = "A"
       THEN
        POKE 15360 + K, ASC(V$)
1050  PC = PC + 1
1060  IF K = > 343
       THEN
        K = K + 3:
       ELSE
        K = K + 2
1070  IF PC = CA
       THEN
        1080:
       ELSE
        990
1080  PC = 0:
      PRINT @640, CHR$(31);:
      LINE INPUT "IS THIS DATA CORRECT ? ( Y/N ) ";AN$:
      GOSUB 1670:
      IF AN$ = "Y"
       THEN
        1150:
        :
       ELSE
        IF AN$ < > "N"
         THEN
          1080:
         ELSE
          1090
1090  PRINT "ENTER THE NUMBER OF THE DAY THAT IS INCORRECT ( 1 - ";CA
      ;" ) ";:
      LINE INPUT IC$:
      IC = VAL(IC$)
1100  IF IC < 1 OR IC > CA
       THEN
        PRINT @896, CHR$(31);:
        GOTO 1090
1110  K1 = 15685:
      IF IC < 11
       THEN
        K = (IC - 1) * 2:
        K = K + K1:
        GOTO 1130
1120  K = 21 + ((IC - 11) * 3):
      K = K + K1
1130  IF PEEK(K) = 80
       THEN
        POKE K,65:
       ELSE
        POKE K,80
1140  GOTO 1080
```

Program continued

```
1150  FOR K = 325 TO 343 STEP 2
1160   PC = PC + 1:
       IF PC > CA
        THEN
         1230
1170   IF PEEK(15360 + K) = 80
        THEN
         DP(PC) = DP(PC) + 1
1180   NEXT K
1190  FOR K = 346 TO 379 STEP 3
1200   PC = PC + 1:
       IF PC > CA
        THEN
         1230
1210   IF PEEK(15360 + K) = 80
        THEN
         DP(PC) = DP(PC) + 1
1220   NEXT K
1230  IF X + 1 > NP
       THEN
        FTD = 0:
        GOTO 1250
1240  PRINT @128, CHR$(31);:
      NEXT X
1250 FOR X = 1 TO CA
1260  IF DP(X) = > AV
       THEN
        FTD = FTD + 2:
        GOTO 1280
1270  IF DP(X) < AV AND DP(X) > 0
       THEN
        FTD = FTD + 1
1280  DP(X) = 0
1290  NEXT X
1300 SB(SD) = FTD:
     SD = SD + 1
1305 FTD = 0
1310 QA = QA + 1:
     IF QA = UF
      THEN
       QA = 0:
       RQ = RQ + 1
1320 CLS :
     PRINT @448,"";:
     LINE INPUT "ARE YOU READY FOR THE NEXT STUDENT ? ( Y/N ) ";AN$:
     GOSUB 1670:
     IF AN$ = "Y"
      THEN
       GOTO 1330:
       :
      ELSE
       IF AN$ < > "N"
        THEN
         1320:
        ELSE
         1340
1330 GOTO 590
1340 FOR X = 1 TO NS
1350  SA(X) = SA(X) + SB(X)
1360  NEXT X
1370 IF UR = 1
      THEN
       RQ = 2:
       QA = 0:
       TA = TA + CA:
       CA = 0:
       SD = 1
1380 OPEN "R",2,"DATTEND"
1390 FIELD 2,2ASXA$,2ASXB$,2ASXC$,2ASXD$,2ASXE$,2ASXF$,2ASXG$,2ASXH$,
     2ASXI$,2ASXJ$,2ASXK$
```

```
1400 GET 2,1
1410 LSET XA$ = MKI$ (TA):
     LSET XB$ = MKI$ (CA)
1420 LSET XC$ = MKI$ (SD):
     LSET XD$ = MKI$ (RQ)
1430 LSET XE$ = MKI$ (QA):
     LSET XF$ = MKI$ (NS):
     LSET XK$ = MKI$ (AV)
1440 PUT 2,1
1450 Q = 0:
     RB = 2
1460 FOR X = 1 TO NS
1470  G = Q * 4
1480  FIELD 2,(G)ASDX$,4ASDY$
1490  IF UR = 1
       THEN
        GET 2,RB
1500  IF UZ = 1
       THEN
        1540
1510  IF SA(X) < 10
       THEN
        GH$ = "000" + RIGHT$( STR$(SA(X)),1):
        GOTO 1540
1520  IF SA(X) < 99
       THEN
        GH$ = "00" + RIGHT$( STR$(SA(X)),2):
        GOTO 1540
1530  GH$ = "0" + RIGHT$( STR$(SA(X)),3)
1540  LSET DY$ = GH$
1550  PUT 2,RB
1560  IF UZ = 1
       THEN
        1600
1570  Q = Q + 1:
      IF Q = 64
       THEN
        Q = 0:
        RB = RB + 1
1580  NEXT X
1590 GH$ = "0999":
     UZ = 1:
     GOTO 1470
1600 CLOSE
1610 RUN "CLASMENU"
1620 CLS :
     PRINT @394,"AN ERROR HAS OCCURRED IN THE EXECUTION OF THE PROGRA
     MCALLED 'STUDENT ATTENDANCE INPUT'."
1630 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
1640 PRINT TAB(5)"ERROR LINE = "; ERL
1650 FOR V = 1 TO 5000:
      NEXT V
1660 STOP
1670 AN$ = LEFT$(AN$,1):
     RETURN
```

Program Listing 2. *Teacher record change*

```
10 :
    '  TEACHER FILE NAME CHANGE ( TEACHANG )
20 :
    '  COPYRIGHT OCTOBER 1, 1981
30 :
    '  ULDERIC F. RACINE
40 :
    '  2520 S.E. ALEXANDER DRIVE
```

Program continued

```
 50 :
    '  TOPEKA, KANSAS 66605
100 CLEAR 5000
110 ON ERROR GOTO 690
120 OPEN "R",1,"TEACHER":
    RN = LOF (1):
    IF RN = 0
     THEN
      740
130 NR = RN * 10:
    IF NR > 100
     THEN
      NR = 99
140 RN = 1:
    Q = 0:
    X = 0:
    NT = 0:
    DIM TN$(NR + 5)
150 G = Q * 25
160 FIELD 1,(G)ASDUMMMY$,25ASDA$
170 GET 1,RN
180 IF DA$ = STRING$(25,88)
     THEN
      CLOSE :
      NT = X:
      GOTO 270
190 X = X + 1
200 T$ = DA$:
    K = LEN(T$)
210 FOR X1 = 1 TO K
220  IF MID$(T$,X1,2) = "  "
      THEN
       T$ = LEFT$(T$,X1 - 1):
       GOTO 240
230  NEXT X1
240 TN$(X) = T$:
    T$ = ""
250 Q = Q + 1:
    IF Q = 10
     THEN
      Q = 0:
      RN = RN + 1
260 GOTO 150
270 CLS :
    PRINT "TEACHER'S FILE NAME CHANGE OPTIONS :":
    A$(1) = "REPLACE":
    A$(2) = "CORRECT SPELLING":
    A$(3) = "DELETE"
280 PRINT @128,"1 - REPLACE A TEACHER CURRENTLY ON THE FILE WITH A N
    EW TEACHER"
290 PRINT "2 - CORRECT THE SPELLING OF AN EXISTING TEACHER'S NAME"
300 PRINT "3 - DELETE A TEACHER'S NAME FROM THE FILE"
310 PRINT "4 - RETURN TO THE MASTER MENU"
320 PRINT @448,"<ENTER> OPTION NUMBER ( 1 - 4 ) ";:
    LINE INPUT OP$:
    OP = VAL(OP$):
    IF OP < 1 OR OP > 4
     THEN
      270
330 IF OP = 4
     THEN
      570
340 CLS :
    QU = 0:
    QV = 4:
    T$ = ""
350 FOR X = 1 TO NT
360  IF QU = 1
      THEN
       390
```

```
370  IF X < 10
      THEN
       PRINT TAB(1)X;" -  ";TN$(X);:
       QU = 1:
       GOTO 410
380  PRINT X;" -  ";TN$(X);:
     QU = 1:
     GOTO 410
390  IF X < 10 PRINT TAB(31)X;" -  ";TN$(X):
     QU = 0:
     GOTO 410
400  PRINT TAB(30)X;" -  ";TN$(X):
     QU = 0:
     QV = QV + 1
410  IF QV < 10
      THEN
       430 :
      ELSE
       PRINT :
       PRINT "IS THE TEACHER'S NAME YOU WISH TO ";A$(OP):
       PRINT "LISTED ABOVE ?";:
       LINE INPUT " ( Y/N )  ";AN$:
       GOSUB 680 :
       IF AN$ = "Y"
        THEN
         450 :
         :
        ELSE
         IF AN$ < > "N"
          THEN
           PRINT @640, CHR$(31);:
           GOTO 410 :
           :
          ELSE
           420
420  IF QV = 10
      THEN
       QV = 4:
       CLS
430  NEXT X
440 PRINT @704,"IS THE TEACHER'S NAME YOU WISH TO ";A$(OP):
    PRINT "LISTED ABOVE ?";:
    LINE INPUT " ( Y/N )  ";AN$:
    GOSUB 680 :
    IF AN$ = "Y"
     THEN
      450 :
      :
     ELSE
      IF AN$ < > "N"
       THEN
        PRINT @704, CHR$(31);:
        GOTO 440 :
        :
       ELSE
        GOTO 270
450 A$(4) = "NEW TEACHER'S NAME":
    A$(5) = "NEW SPELLING":
    PRINT @640, CHR$(31);:
    PRINT :
    PRINT "<ENTER> THE NUMBER OF THE TEACHER YOU WISH TO ";A$(OP):
    LINE INPUT "TEACHER # ";NN$:
    N = VAL(NN$):
    IF N < 1 OR N > NT
     THEN
      450
460 PRINT @640, CHR$(31);:
    PRINT :
    PRINT "TEACHER #";N;"IS ";TN$(N):
    PRINT "IS THIS THE NAME YOU WISH TO ";A$(OP);:
```

Program continued

```
    LINE INPUT " ? ( Y/N )  ";AN$:
    GOSUB 680 :
    IF AN$ = "Y"
     THEN
      470 :
      :
     ELSE
      IF AN$ < > "N"
       THEN
        460 :
       ELSE
        340
470 IF OP = 3
     THEN
      560 :
     ELSE
      CLS :
      PRINT @448,"TEACHER #";N;" :   ";TN$(N)
480 PRINT :
    PRINT "PLEASE ENTER THE ";A$(OP + 3):
    PRINT "LAST NAME <SPACE> FIRST NAME <SPACE> MIDDLE INITIAL (IF A
    NY)"
490 IF OP = 2
     THEN
      510
500 LINE INPUT "NEW TEACHER'S NAME :  ";T$:
    GOTO 520
510 LINE INPUT "NEW SPELLING :  ";T$
520 IF LEN(T$) = 0
     THEN
      PRINT @512, CHR$(31);:
      GOTO 480
530 PRINT @512, CHR$(31);:
    PRINT "YOU WISH TO REPLACE ";TN$(N):
    PRINT "WITH ";T$;" .";:
    LINE INPUT " IS THIS CORRECT ? ( Y/N )  ";AN$:
    GOSUB 680 :
    IF AN$ = "Y"
     THEN
      540 :
      :
     ELSE
      IF AN$ < > "N"
       THEN
        520 :
       ELSE
        470
540 TN$(N) = T$
550 GOTO 270
560 TN$(N) = "DELETED":
    GOTO 270
570 OPEN "R",1,"TEACHER":
    TN$(NT + 1) = STRING$(25,88)
580 RN = 1:
    Q = 0
590 FOR X = 1 TO NT + 1
600  G = Q * 25
610  FIELD 1,(G)ASDUMMY$,25ASDA$
620  LSET DA$ = TN$(X)
630  Q = Q + 1:
     IF Q = 10
      THEN
       Q = 0:
       PUT 1,RN:
       RN = RN + 1
640  NEXT X
650 PUT 1,RN
660 CLOSE
670 RUN "CLASMENU"
680 AN$ = LEFT$(AN$,1):
    RETURN
690 CLS :
    PRINT @394,"AN ERROR HAS OCCURRED IN THE EXECUTION OF THE PROGRA
    MCALLED 'TEACHER NAME CHANGE'."
700 PRINT TAB(5)"ERROR TYPE = "; ERR / 2 + 1
710 PRINT TAB(5)"ERROR LINE = "; ERL
720 FOR X = 1 TO 5000:
     NEXT
730 RESUME 670
740 CLS :
    PRINT @448,"THERE IS NO TEACHER FILE ON THE DISKS CURRENTLY":
    PRINT "IN THE DRIVES. I AM RETURNING YOU TO THE MASTER MENU."
750 FOR X = 1 TO 1000:
     NEXT X
760 GOTO 670
```

Program Listing 3. *Year-to-date attendance printout*

```
 10 :
    '   YEAR-TO-DATE ATTENDANCE PRINTOUT ( PNTATEND )
 20 :
    '   COPYRIGHT OCTOBER 1, 1981
 30 :
    '   ULDERIC F. RACINE
 40 :
    '   2520 S.E. ALEXANDER DRIVE
 50 :
    '   TOPEKA, KANSAS 66605
100 CLS :
    PRINT CHR$(23):
    PRINT @454,"YEAR-TO-DATE ATTENDANCE"
110 UR = 1
120 OPEN "R",1,"STDSCHED"
130 FIELD 1,2ASFA$,2ASFB$,2ASFC$,14ASDUMMY$
140 GET 1,1
150 FS = CVI (FB$)
160 UF = CVI (FC$)
170 X = ( LOF (1) - 1) * UF
180 IF UR = 1
     THEN
      T = (X * FS) + 1500:
      CLOSE :
      CLEAR T:
      GOTO 120
190 ON ERROR GOTO 970
200 DIM SN$(X + 10)
210 RN = 2:
    X = 0
220 G = (UF * FS) - 1
230 FIELD 1,(G)ASDS$
240 GET 1,RN
250 FOR Y = 1 TO G STEP FS
260  SS$ = MID$(DS$,Y,24)
270  IF SS$ = STRING$(24,88)
      THEN
       NS = X:
       CLOSE :
       GOTO 320
280  X = X + 1
290  SN$(X) = SS$:
     SS$ = ""
300  NEXT Y
310 RN = RN + 1:
    GOTO 240
320 OPEN "R",2,"DATTEND":
    XX = LOF (2):
```

Program continued

```
    IF XX = 0
     THEN
      CLOSE :
      GOTO 950
330 FIELD 2,2ASXA$,20ASDUMMY$
340 GET 2,1
350 TA = CVI (XA$)
360 RN = 2:
    DIM SN(NS + 10):
    C = 1:
    X = 0
370 FIELD 2,128ASD$(1),128ASD$(2)
380 GET 2,RN
390 FOR Y = 1 TO 128 STEP 4
400  AC = VAL( MID$(D$(C),Y,4))
410  IF AC = 999
      THEN
       CA = X:
       CLOSE :
       GOTO 470
420  X = X + 1
430  SN(X) = AC
440  NEXT Y
450 IF C = 1
     THEN
      C = 2:
      GOTO 390
460 C = 1:
    RN = RN + 1:
    GOTO 380
470 CLS :
    PRINT "YEAR-TO-DATE ATTENDANCE PROGRAM":
    PRINT @128,"OPTIONS :":
    PRINT @256,"1 - PRINTOUT A LIST OF ALL STUDENTS ON THE FILE":
    PRINT "2 - PRINTOUT TOTALS ONLY":
    PRINT "3 - PRINTOUT ATTENDANCE FOR A SPECIFIC STUDENT":
    PRINT "4 - RETURN TO MASTER MENU"
480 UR = 0:
    GT = 0:
    ST = 0:
    PO = 0
490 PRINT @640,"";:
    LINE INPUT "<ENTER> OPTION # ( 1 - 4 )  ";O$:
    O = VAL(O$):
    IF O < 1 OR O > 4
     THEN
      PRINT @640, CHR$(31);:
      GOTO 490
500 ON O GOTO 510,690,770,880
510 GOSUB 890
520 P1$ = STRING$(40,45):
    P2$ = "YEAR-TO-DATE ATTENDANCE IN FULL-TIME EQUIVALENT DAYS":
    P3$ = "TOTAL DAYS ENTERED :":
    P4$ = "STUDENT":
    P5$ = "DATE :  ":
    P6$ = "NUMBER OF":
    P7$ = "DAYS PRESENT"
530 IF UR = 1
     THEN
      RETURN
540 CLS :
    IF PO = 1
     THEN
      LINE INPUT "PRESS <ENTER> WHEN YOU ARE READY TO PRINT ";AN$:
      POKE 16424,67:
      POKE 16425,1:
      CLS
550 C = 1
560 FOR X = 1 TO NS
570  PRINT SN$(X); TAB(25) USING "###.#";SN(X) / 2:
     C = C + 1
580  IF PO < > 1 AND C = 15
      THEN
       LINE INPUT "PRESS <ENTER> TO CONTINUE  ";AN$:
       C = 1:
       CLS :
       GOTO 620
590  IF PO = 0
      THEN
       620
600  IF PO = 1 AND C2 = 1
      THEN
       LPRINT P2$:
       LPRINT " ":
       LPRINT P5$;DT$:
       LPRINT P3$;TA:
       LPRINT TAB(26)P6$:
       LPRINT P4$; TAB(24)P7$:
       LPRINT P1$:
       C2 = 0
610  LPRINT SN$(X); TAB(25) USING "###.#";SN(X) / 2
620  ST = ST + SN(X) / 2
630  IF PO = 1 AND PEEK(16425) = > 60
      THEN
       LPRINT P1$:
       LPRINT "SUBTOTAL"; TAB(24) USING "#####.#";ST:
       GT = GT + ST:
       LPRINT CHR$(12):
       C2 = 1:
       ST = 0
640  NEXT X
650 GT = GT + ST
660 IF PO = 1
     THEN
      LPRINT P1$:
      LPRINT "SUBTOTAL"; TAB(24) USING "#####.#";ST:
      LPRINT " ":
      LPRINT "TOTAL DAYS ENTERED"; TAB(24) USING "#####.#";TA:
      LPRINT "TOTAL NUMBER OF STUDENTS"; TAB(24) USING "####";NS
670 PRINT P1$:
    PRINT "SUBTOTAL"; TAB(24) USING "####.#";ST:
    PRINT :
    PRINT "TOTAL DAYS ENTERED"; TAB(24) USING "####.#";TA:
    PRINT "TOTAL NUMBER OF STUDENTS"; TAB(24) USING "####";NS
680 PRINT :
    LINE INPUT "PRESS <ENTER> TO CONTINUE ";AN$:
    GOTO 470
690 GOSUB 890:
    UR = 1:
    GOSUB 520:
    UR = 0:
    GT = 0
700 FOR X = 1 TO NS
710  GT = GT + SN(X) / 2
720  NEXT X
730 CLS
740 PRINT P2$:
    PRINT :
    PRINT P3$;TA:
    PRINT P1$:
    PRINT "TOTAL ATTENDANCE"; TAB(24) USING "####.#";GT:
    PRINT "TOTAL NUMBER OF STUDENTS"; TAB(24) USING "####";NS
750 IF PO = 0
     THEN
      680
760 LPRINT P2$:
    LPRINT P5$;DT$:
    LPRINT P3$;TA:
    LPRINT P1$:
    LPRINT "TOTAL ATTENDANCE"; TAB(24) USING "#####.#";GT:
```

Program continued

```
    LPRINT "TOTAL NUMBER OF STUDENT"; TAB(24) USING "####";NS:
    LPRINT P1$:
    GOTO 680
770 CLS
780 PRINT @448,"ENTER THE NAME OF THE STUDENT":
    PRINT "LAST NAME <SPACE> FIRST NAME <SPACE> MIDDLE INITIAL (IF A
    NY)":
    LINE INPUT "STUDENT'S NAME :  ";SU$
790 GOSUB 890
800 K = LEN(SU$)
810 FOR X = 1 TO NS
820  IF LEFT$(SN$(X),K) = SU$
      THEN
       850
830  NEXT X
840 CLS :
    PRINT @448,"I CAN NOT FIND ";SU$;" IN THE FILE.":
    FOR X = 1 TO 400:
     NEXT X:
    GOTO 470
850 CLS :
    PRINT @448,SN$(X):
    PRINT "TOTAL ATTENDANCE TO DATE"; USING "###.#";SN(X) / 2:
    IF PO < > 1
     THEN
      870
860 LPRINT P5$;DT$:
    LPRINT SN$(X):
    LPRINT "TOTAL ATTENDANCE TO DATE - "; USING "###.#";SN(X)
    / 2
870 PRINT :
    PRINT "DO YOU HAVE ANOTHER STUDENT WHOSE ATTENDANCE":
    LINE INPUT "YOU WISH TO SEE ? ( Y/N )  ";AN$:
    GOSUB 930:
    IF AN$ = "Y"
     THEN
      770:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        870:
       ELSE
        470
880 RUN "CLASMENU"
890 CLS :
    PRINT @448,"";:
    LINE INPUT "DO YOU WANT A HARDCOPY OF THIS LIST ( Y/N )  ";AN$:
    GOSUB 930:
    IF AN$ = "Y"
     THEN
      PO = 1:
      GOTO 900:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        890:
       ELSE
        RETURN
900 GOSUB 940:
    C2 = 1:
    LINE INPUT "PLEASE ENTER TODAY'S DATE ( MM/DD/YY )  ";DT$:
    IF LEN(DT$) < > 8
     THEN
      900
910 GOSUB 940:
    LINE INPUT "SHALL I GENERATE A TEST LINE ? ( Y/N )  ";AN$:
    GOSUB 930:
    IF AN$ = "Y"
```

```
     THEN
      920:
      :
     ELSE
      IF AN$ < > "N"
       THEN
        910:
       ELSE
        RETURN
920 LPRINT "THIS IS A TEST LINE---------------------":
    GOTO 910
930 AN$ = LEFT$(AN$,1)
940 PRINT @448, CHR$(31);:
    RETURN
950 CLS :
    PRINT @448,"YOU MUST COMPLETE ONE ATTENDANCE CYCLE BEFOREI CAN P
    RINTOUT ANY ATTENDANCE DATA."
960 GOTO 880
970 CLS :
    PRINT @394,"AN ERROR HAS OCCURRED IN THE EXECUTION OF THE PROGRA
    M'YEAR-TO-DATE-ATTENDANCE'."
980 FOR X = 1 TO 5000:
     NEXT X
990 STOP
```

[illegible]

GAMES

Roulette
Five Short Games
Rubik's Cube™ Manipulator

Roulette

by Paul G. Ramsteyn

To play roulette in a casino, you select a combination of odds and risk. The ratio of the odds to the risk is fixed (36/37). You have a great chance of winning a little money or a small chance of winning a lot.

The roulette table consists of a roulette wheel which contains alternate red and black spaces numbered from 1 through 36. They are not in sequential order. The 0 space is green. On the green layout, the numbers are in numerical order. There are also compartments for passe, manque, pair, impair, noir, rouge, and dozens. The players can place their bets until the ball is rolling and "rien ne vas plus" has been called out.

When the ball falls into one of the numbered compartments, one of the croupiers rakes in the lost bets, which go into the bank, and pays the winning players.

About the Program

This program offers a limited number of possible bets. It makes no difference whether you bet $180 on red or 18 times $10 on the single red numbers. Any combination is acceptable. Every bet is broken down into bets on single numbers (0–36) and placed in the array G(A0,36), where A0 is the number of players. The players' names are in X$(A0), and their balances are in S(A0). In addition, some variables are used in loops and for the READ statements. Program Listing 1 contains the Roulette program.

Break Disable

You can not use the BREAK key without losing your variables. In line 690, a routine is POKEd into memory area 32743–32767 decimal. The data for this is in lines 130–140. Program Listing 2 shows you this routine in Editor/Assembler format.

If you type a BREAK character, the keyboard driver in Level II ROM ends in an RST 28. This jumps to 400CH and is normally sent back with a return. In this case, my check routine is met before the return. If necessary, the character is changed into 00. POKE 16384,201 enables the BREAK.

Graphics

A second machine-language routine (Program Listing 3) is POKEd into memory area 31894–32738. The data is in lines 150–670. The routine is used in line 700 to make up the roulette turntable. Because of these routines, you must answer MEM SIZE? with 31800 to reserve memory. When POKEing is finished, the first lines are deleted, and you need to run the program again to play the game.

The Flow of the Program

In lines 1110–1130, the program asks for the players' names and DIMensions the arrays. Then, every player gets a turn. You must type QUIT to stop the game. This can only be done by the first player before any bets are made.

In placing bets, you have 10 possibilities which are explained in lines 1290–1520 (EXPLANATION) and in lines 1530–1620 (LAYOUT). If necessary, you must give detailed information about the bet. G(A0,36) is filled to conform to the amounts (lines 1370–1500). You can enter @ to exit the betting mode, but this does not mean your turn is over. You can ask for an explanation and place more bets.

When everyone has had a turn, the turntable appears on the screen, the ball rolls to one of the numbers, and this number and its color are displayed (lines 1820–2040). Then, the gains and losses are computed, the balances are displayed (lines 1600–1720), G(A0,36) is cleaned, and the procedure starts over.

Once you QUIT the roulette, you see your net results. The bank's result is divided, and the computer is blocked. You can then turn it off.

Program Listing 1. *Roulette*

```
100 :
    '                                            ************
110 :
    '                                            ROULETTE 1.4
120 :
    '                                            ************
130 DATA 33,12,64,54,195,33,243,127,34,13,64,201
140 DATA 71,123,254,4,32,2,175,201,120,201,0,0,0
150 DATA -67,63,24,-83,-100,-40,-99,-36,-97,137,76,101
160 DATA 28,28,28,28,44,76,44,28,76,44,76,44,28,76,44,28,76,44
170 DATA 76,44,28,44,76,44,28,76,44,76,44,28,76,44,28,76,44,76
180 DATA 44,28,76,44,76,44,28,44,76,44,28,44,76,44,28,44,44,28
190 DATA 76,44,44,28,76,44,28,28,28,28
200 DATA 28,28,28,28,49,57,49,28,56,33,41,49,28,40,49,28,56,33
210 DATA 57,49,28,49,56,33,28,40,49,41,48,28,40,49,28,56,33,57
220 DATA 48,28,40,49,56,33,28,49,41,48,28,49,41,49,28,41,49,28
230 DATA 56,33,49,28,56,33,28,28,28,28
240 DATA 76,44,76,44,29,31,29,28,31,29,31,29,28,28,29,28,31,29
250 DATA 31,29,28,29,31,29,28,31,29,31,29,28,31,29,28,31,29,31
260 DATA 29,28,31,29,31,29,28,29,31,29,28,29,31,29,28,28,29,28
270 DATA 31,29,29,28,31,29,60,76,60,76
280 DATA 56,33,56,33,28,28,28,28,28,28,28,28,28,28,28,28,28,28
290 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28
300 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28
310 DATA 28,28,28,28,28,28,68,42,38,72
320 DATA 31,29,79,45,28,28,28,28,28,43,31,31,31,31,31,31,31,31
330 DATA 31,79,79,79,79,79,79,79,79,79,79,79,31,91,76,28,28,28
340 DATA 51,49,28,28,60,91,31,79,79,79,79,79,79,79,79,79,79,31
350 DATA 91,28,28,28,28,28,62,63,78,31
360 DATA 28,28,41,49,28,28,28,28,28,91,91,43,43,43,43,91,28,28
370 DATA 28,28,28,28,28,28,28,28,28,28,28,31,28,28,28,28,28,28
380 DATA 41,33,28,28,28,28,28,28,28,28,28,28,40,40,88,28,91,28
390 DATA 91,28,28,28,28,28,70,36,74,28
400 DATA 28,76,47,45,28,28,28,28,28,91,91,28,28,28,28,91,28,28
410 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28
420 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,31,28,91,28
430 DATA 91,28,28,28,28,28,62,76,62,60
440 DATA 28,40,49,49,28,28,28,28,28,91,91,31,31,91,31,31,68,56
450 DATA 72,28,88,28,48,68,48,28,88,40,36,72,56,36,72,56,68,56
460 DATA 32,28,28,28,28,28,28,28,28,28,28,28,28,40,43,28,91,28
470 DATA 91,28,28,28,28,28,36,74,38,74
480 DATA 28,47,45,45,28,28,28,28,28,91,91,28,28,91,76,28,70,81
490 DATA 86,28,91,76,49,70,81,44,91,77,28,70,49,28,70,49,70,83
500 DATA 44,28,60,76,76,76,76,76,76,76,79,79,79,79,79,79,79,76
510 DATA 91,28,28,28,28,28,62,79,28,30
520 DATA 28,49,41,49,28,28,28,28,28,28,28,28,28,28,28,28,28,28
530 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28
540 DATA 28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28,28
550 DATA 28,28,28,28,28,28,70,72,28,28
560 DATA 28,29,28,61,76,60,76,28,60,28,60,76,60,76,28,60,60,76
570 DATA 28,60,76,60,60,28,60,76,28,60,60,76,28,60,76,60,76,28
580 DATA 60,76,28,60,76,60,76,28,60,60,28,60,76,60,76,28,60,60
590 DATA 76,28,60,76,60,76,30,31,28,28
600 DATA 28,28,28,68,42,70,70,28,70,28,36,74,36,74,28,70,70,72
610 DATA 28,68,42,38,74,28,38,72,28,70,70,70,28,68,42,36,74,28
620 DATA 70,74,28,36,74,70,70,28,70,70,28,36,74,70,72,28,70,36
630 DATA 74,28,68,42,36,74,28,28,28,28
640 DATA 28,28,28,30,31,30,31,28,30,28,30,31,30,31,28,30,30,31
650 DATA 28,30,31,28,30,28,30,31,28,30,30,31,28,30,31,30,31,28
660 DATA 30,31,28,30,31,30,31,28,30,30,28,30,31,30,31,28,30,30
670 DATA 31,28,30,31,28,30,28,28,28,28
680 CLS :
    PRINT CHR$(23):
    PRINT @524,"FRENCH ROULETTE"
690 FOR X = 32743 TO 32767:
     READ A:
     POKE X,A:
     NEXT
```

Program continued

```
700 POKE 16526,231:
    POKE 16527,127:
    X% = USR(0)
710 FOR I = 1 TO 844:
     READ J:
     POKE I + 31894,J + 100:
     NEXT
720 POKE 16526,151:
    POKE 16527,124
730 CLS :
    PRINT @512,"NOW  R U N  THE PROGRAM. GOOD LUCK!"
740 DELETE 100 - 740
750 GOTO 1110
760 CLS :
    X = USR(0):
    PRINT @979,"NOW PRESS <ENTER> FOR ROLL";:
    X$ = ""
770 X$ = INKEY$:
    IF X$ < > ""
     THEN
      890
780 PRINT @989, CHR$(199);:
    FOR I = 1 TO 170:
     NEXT
790 PRINT @989,"<ENTER>";:
    FOR I = 1 TO 300:
     NEXT :
    GOTO 770
800 DATA 228,40,28,167,77,29,172,79,31,176,79,31,179,76,28,183
810 DATA 77,29,186,79,31,251,40,28,379,40,28,507,40,28,635,40
820 DATA 28,697,79,76,693,63,60,689,63,60,685,63,60,681,63,60
830 DATA 677,79,76,673,63,60,668,63,60,665,79,76,661,63,60,656
840 DATA 63,60,652,63,60,648,63,60,645,63,60,580,40,28,452,40
850 DATA 28,324,40,28,196,40,28,133,79,31,137,77,29,141,76,28
860 DATA 145,77,29,150,79,31,154,77,29,158,79,31,162,77,29
870 DATA 1,24,7,36,5,20,11,32,17,28,19,15,34,13,26,3,22,9,30
880 DATA 4,25,6,29,18,21,8,37,12,33,31,16,27,2,23,10,35,14
890 IF ASC(X$) < > 13
     THEN
      770
900 PRINT @979,"     RIEN NE VAS PLUS      ";:
    FOR I = 1 TO 100:
     NEXT
910 RANDOM :
    D0 = RND(37) - 1:
    D1 = RND(4) + 4:
    A9 = A9 + 1
920 RESTORE :
    FOR I = 1 TO 111:
     READ J:
     NEXT
930 PRINT @979, CHR$(220);
940 FOR I = 0 TO 36:
     READ J:
     IF NOT I = D0, NEXT
950 FOR I = 1 TO D1:
     RESTORE :
     FOR K = 1 TO 37:
      READ I1,I2,I3
960   POKE I1 + 15360,I2 + 100:
      FOR X = 1 TO I * 3:
       NEXT :
      POKE I1 + 15360,I3 + 100
970   NEXT :
     NEXT
980 RESTORE :
    FOR I = 1 TO J:
     READ I1,I2,I3:
     POKE I1 + 15360,I2 + 100
990  FOR X = 1 TO 12 + 4 * I:
       NEXT
1000   IF I < > J, POKE I1 + 15360,I3 + 100:
      NEXT
1010 IF DO = 0,X$ = ".....":
     GOTO 1030
1020 IF J / 2 = INT(J / 2),X$ = "ROUGE":
      ELSE
       X$ = "NOIR"
1030 PRINT @408,DO;": ";X$;:
     GOSUB 1570
1040 FOR J = 1 TO A0:
      FOR K = 0 TO 36:
       G(J,K) = 0:
       NEXT :
      NEXT
1050 GOSUB 1060:
     GOTO 1600
1060 PRINT @979,"PRESS <SPACE-BAR> TO CONTINUE";:
     X$ = ""
1070 X$ = INKEY$:
     IF X$ < > ""
      THEN
       1100:
      ELSE
       PRINT @985, CHR$(203);
1080 FOR X = 1 TO 150:
      NEXT :
     PRINT @985,"<SPACE-BAR>";
1090 FOR X = 1 TO 300:
      NEXT :
     GOTO 1070
1100 IF ASC(X$) < > 32
      THEN
       1070:
      ELSE
       RETURN
1110 CLS :
     INPUT "THE NUMBER OF PLAYERS PLEASE";A0:
     PRINT
1120 DIM X$(A0),S(A0),G(A0,36)
1130 FOR I = 1 TO A0:
      PRINT "NAME OF PLAYER #"I;:
      INPUT X$(I):
      NEXT
1140 PRINT "EACH PLAYER IS FREE TO BET AS MUCH AS HE WANTS,"
1150 PRINT "UNLESS THE GROUP WANTS TO OBSERVE LIMITATIONS."
1160 GOSUB 1060
1170 FOR I = 1 TO A0
1180  CLS :
      PRINT "YOUR TURN, "X$(I);:
      PRINT @128,"MENU :"
1190  PRINT "<B> = PLACE BETS":
      PRINT "<X> = EXPLANATION"
1200  PRINT "<L> = LAYOUT":
      PRINT "<E> = END OF YOUR TURN"
1210  PRINT "<QUIT> = END OF ROULETTE":
      C$ = ""
1220  INPUT "ENTER YOUR CHOISE PLEASE ?";C$
1230  H = 0:
      FOR J = 0 TO 36:
       IF G(I,J) < > 0,H = 1:
       NEXT :
       ELSE
        NEXT
1240 IF C$ = "B"
      THEN
       1270:
      ELSE
       IF C$ = "X"
        THEN
```

Program continued

```
        1730
1250 IF C$ = "L"
      THEN
       1820:
      ELSE
       IF C$ = "E", NEXT I:
       GOTO 760
1260 IF C$ = "QUIT" AND I = 1 AND H = 0
      THEN
       1660:
      ELSE
       GOTO 1180
1270 CLS :
     PRINT X$(I)", PLACE BETS PLEASE"; CHR$(204);
1280 PRINT "TO EXIT THIS MODE, ENTER <@>":
     C$ = ""
1290 PRINT @192,"OPTION    ";:
     INPUT C$:
     IF C$ = "C", GOSUB 1360
1300 IF C$ = "R", GOSUB 1380:
      ELSE
       IF C$ = "N", GOSUB 1400
1310 IF C$ = "E", GOSUB 1420:
      ELSE
       IF C$ = "O", GOSUB 1430
1320 IF C$ = "L", GOSUB 1450:
      ELSE
       IF C$ = "H", GOSUB 1440
1330 IF C$ = "P", GOSUB 1460:
      ELSE
       IF C$ = "S", GOSUB 1480
1340 IF C$ = "V", GOSUB 1510:
      ELSE
       IF C$ = "@"
        THEN
         1180
1350 GOTO 1270
1360 INPUT "COLUMN #  ";K:
     K = INT( ABS(K)):
     IF K > 3 OR K = 0
      THEN
       1360
1370 GOSUB 1560:
     FOR J = K TO 33 + K STEP 3:
      G(I,J) = G(I,J) + G / 12:
      NEXT :
     RETURN
1380 GOSUB 1560:
     RESTORE :
     FOR J = 1 TO 148:
      READ K:
      NEXT
1390 FOR J = 1 TO 18:
      READ K,K1:
      G(I,K) = G(I,K) + G / 18:
      NEXT :
     RETURN
1400 GOSUB 1560:
     RESTORE :
     FOR J = 1 TO 148:
      READ K:
      NEXT
1410 FOR J = 1 TO 18:
      READ K1,K:
      G(I,K) = G(I,K) + G / 18:
      NEXT :
     RETURN
1420 GOSUB 1560:
     FOR J = 2 TO 36 STEP 2:
      G(I,J) = G(I,J) + G / 18:
```

```
      NEXT :
     RETURN
1430 GOSUB 1560:
     FOR J = 1 TO 35 STEP 2:
      G(I,J) = G(I,J) + G / 18:
      NEXT :
     RETURN
1440 GOSUB 1560:
     FOR J = 19 TO 36:
      G(I,J) = G(I,J) + G / 18:
      NEXT :
     RETURN
1450 GOSUB 1560:
     FOR J = 1 TO 18:
      G(I,J) = G(I,J) + G / 18:
      NEXT :
     RETURN
1460 INPUT "NUMBER     ";K:
     K = ABS( INT(K)):
     IF K > 36
      THEN
       1460
1470 GOSUB 1560:
     G(I,K) = G(I,K) + G:
     RETURN
1480 INPUT "BEGINNING ";J:
     INPUT "LENGTH     ";K
1490 J = INT( ABS(J)):
     K = INT( ABS(K)):
     IF K + J > 37
      THEN
       1480
1500 GOSUB 1560:
     FOR K1 = J TO J + K - 1:
      G(I,K1) = G(I,K1) + G / K:
      NEXT :
     RETURN
1510 INPUT "CENTRE     ";K1:
     K1 = INT( ABS(K1)):
     IF K1 > 36
      THEN
       1510
1520 RESTORE :
     FOR J = 1 TO 150:
      READ K:
      NEXT :
     GOSUB 1560
1530 FOR J = 1 TO 41:
      READ K:
      IF K < > K1, NEXT :
      ELSE
       K = J:
       RESTORE
1540 FOR J = 1 TO 147 + K:
      READ K1:
      NEXT :
     FOR J = 1 TO 5
1550  READ K1:
      G(I,K1) = G(I,K1) + G / 5:
      NEXT :
     RETURN
1560 INPUT "AMOUNT  $ ";G:
     G = ABS(G):
     RETURN
1570 FOR I = 1 TO A0:
      FOR J = 0 TO 36:
       IF J = D0,S(I) = S(I) + 35 * G(I,J):
       GOTO 1590
1580   S(I) = S(I) - G(I,J)
1590   NEXT :
```

Program continued

```
       NEXT :
      RETURN
1600 CLS :
      PRINT "SITUATION AFTER"A9"ROLLS :":
      PRINT
1610 FOR I = 1 TO A0:
       PRINT X$(I),:
       PRINT USING "$ #####.##";S(I):
       NEXT
1620 GOSUB 1060:
      GOTO 1170
1630 DATA 32,15,19,4,21,2,25,17,34,6,27,13,36,11,30,8,23,10,5,24
1640 DATA 16,33,1,20,14,31,9,22,18,29,7,28,12,35,3,26,0,32,15
1650 DATA 19,4
1660 CLS :
      PRINT "FINAL RESULT :":
      PRINT
1670 G = 0:
      FOR I = 1 TO A0:
       G = G + S(I):
       NEXT
1680 FOR I = 1 TO A0:
       PRINT X$(I),:
       PRINT USING "$ #####.##";S(I);:
       PRINT ,
1690   PRINT USING "$ #####.##";(S(I) - G / A0):
       NEXT
1700 PRINT :
      PRINT "THE LAST COLUMN IS YOUR NET PROFIT OR LOSS :"
1710 PRINT "THE BANK'S RESULT IS DIVIDED."
1720 POKE 16413,8:
      GOTO 1720
1730 CLS :
      PRINT "OPTION :  INFORMATION REQUIRED :":
      PRINT
1740 PRINT "COLONNE   <C> AND NUMBER OF COLUMN"
1750 PRINT "ROUGE     <R>":
      PRINT "NOIR      <N>"
1760 PRINT "PAIR      <E> (=EVEN)":
      PRINT "IMPAIR    <O> (=ODD)"
1770 PRINT "MANQUE    <L> (=LOW)":
      PRINT "PASSE     <H> (=HIGH)"
1780 PRINT "PLEIN     <P> AND NUMBER"
1790 PRINT "SERIES    <S>, NUMBER OF BEGINNING AND LENGTH"
1800 PRINT "VOISINS   <V> AND NUMBER"
1810 GOSUB 1060:
      CLS :
      GOTO 1180
1820 CLS :
      PRINT @14,"PLEIN <P> :      0     (=SINGLE NUMBER)";
1830 PRINT @91,"1   2   3";
1840 PRINT @129,"PAIR <E>";:
      PRINT @155,"4   5   6";
1850 PRINT @181,"<O> IMPAIR";
1860 PRINT @193,"(=EVEN NUMBERS)";:
      PRINT @219,"7   8   9";
1870 PRINT @249,"(=ODD)";
1880 PRINT @283,".   .   .";
1890 PRINT @321,"PASSE <H>";:
      PRINT @347,".   .   .";
1900 PRINT @373,"<L> MANQUE";
1910 PRINT @385,"(19 THROUGH 36)        ..  ..  ..";
1920 PRINT @433,"(1 THROUGH 18)";
1930 PRINT @474,"31  32  33";
1940 PRINT @513,"NOIR <N>";:
      PRINT @538,"34  35  36";
1950 PRINT @566,"<R> ROUGE";
1960 PRINT "  (=BLACK NUMBERS)"; CHR$(201); CHR$(91); CHR$(91);
1970 PRINT "  "; CHR$(91); CHR$(91);"  "; CHR$(91); CHR$(91);
1980 PRINT @625,"(=RED NUMBERS)";
```

```
1990 PRINT @658,"COLONNE #1  #2  #3 <C>";
2000 PRINT @705,"SERIES <S> :";:
     PRINT @754,"<V> VOISINS :";
2010 PRINT @769,"NUMBERS IN SEQUENCE";
2020 PRINT @815,"NUMBER AND ITS 4";
2030 PRINT @885,"NEIGHBOURS";
2040 GOSUB 1060:
     GOTO 1180
2050 :
     '                            ***************************
2060 :
     '                            PROGRAM BY PAUL G. RAMSTEYN,
2070 :
     '                            RIJSWIJK, NL., DECEMBER 1980
2080 :
     '                            ***************************
```

Program Listing 2. *BREAK disable*

```
             00100                                ;USER OF LINE 700
             00110                                ;SET JUMP INSTRUCTION
7FE7         00120          ORG    7FE7H          ;ENTRY OF USER
7FE7 210C40  00130          LD     HL,400CH       ;RST 28 JUMPS TO 400C
7FEA 36C3    00140          LD     (HL),0C3H      ;MAKE IT JUMP
7FEC 21F37F  00150          LD     HL,7FF3H       ;TO CHECK ROUTINE
7FEF 220D40  00160          LD     (400DH),HL     ;AT 7FF3
7FF2 C9      00170          RET                   ;RETURN TO LINE 700
             00180                                ;400C IS JP 7FF3 NOW
             00190                                ;
             00200                                ;CHECK ROUTINE, MUST
             00210                                ;CHECK FOR BREAK FOR
             00220                                ;EVERY RST 28 COMES HERE
             00230                                ;ORG = 7FF3
7FF3 47      00240          LD     B,A            ;SAVE ACCUMULATOR
7FF4 7B      00250          LD     A,E            ;IF BREAK, E HOLDS 4
7FF5 FE04    00260          CP     04H            ;SEE IF SO
7FF7 2002    00270          JR     NZ,RETURN      ;RETURN IF NO BREAK
7FF9 AF      00280          XOR    A              ;ELSE CLEAR ACCUMULATOR
7FFA C9      00290          RET                   ;AND RETURN
7FFB 78      00300 RETURN   LD     A,B            ;RESTORE ACCUMULATOR
7FFC C9      00310          RET                   ;AND RETURN
7FFD 00      00320          NOP                   ;END OF RAM
7FFE 00      00330          NOP                   ;HAS NO
7FFF 00      00340          NOP                   ;OPERATIONS
7FFB         00350          END                   ;END OF 7FF3
00000 TOTAL ERRORS
```

Program Listing 3. *Graphics subroutine*

```
             00100                                ;ROUTINE TO DISPLAY GRAPHIC
             00110                                ;USER OF LINE 760
7C96         00120          ORG    7C96H          ;ORIGIN
7C96 21967C  00130          LD     HL,7C96H       ;POINT TO DATA
7C99 11003C  00140          LD     DE,3C00H       ;START OF DISPLAY
7C9C 014003  00150          LD     BC,0340H       ;COUNTER
7C9F EDB0    00160          LDIR                  ;MOVE IT
7CA1 C9      00170          RET                   ;RETURN TO LINE 760
0340         00180          END                   ;END OF 7C96
00000 TOTAL ERRORS
```

Five Short Games

by Michiel van de Panne

Have you ever typed in a game program, only to find out that you have run out of memory, or that the game falls short of your expectations? These five programs are fast action, exciting games which will not wear out your fingers when you type in the listings. All the programs run in Level II or Disk BASIC.

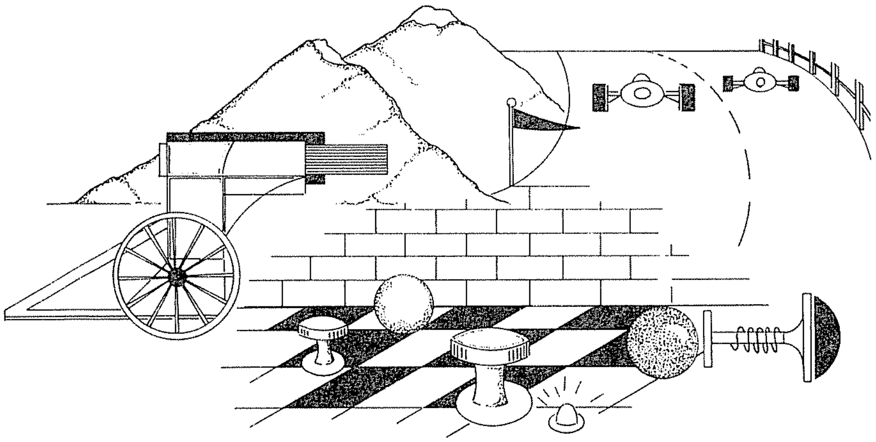
The first game, Roadrace, which is shown in Program Listing 1, is a short version of a popular game. The game features various skill levels and a car that moves in a particular direction as long as you are pressing the proper key. At the end of the game, the computer shows your score, which is determined by the length of time you drive without crashing into another car or driving off the road. Because the game is so short, modifying it is simple.

The second game is called Reflex (see Program Listing 2) and it tests your reflexes. In this game, you are positioned in the center of the screen. Targets appear one at a time, in random order, above you, below you, to the left, or to the right. You shoot at them by using the four arrow keys, and the computer keeps score. You are penalized for each shot you miss. The faster you shoot the targets down, the higher your score will be. The program also remembers your high score. I recommend starting out with a series of 10 targets. Modifications such as adding sound are easy to make.

The next game is the shortest version I know of a game known as Breakout. (See Program Listing 3.) In this game the player tries to knock bricks out of the wall above him with a ball. The object of the game is to knock out as many bricks as possible by keeping the ball in play with a paddle which is at the bottom of the screen. The bricks in the first row are worth one point each; those in the second row of bricks are worth three points each; those in the third row are worth five points; and so on. The paddle keeps moving in the direction of the key you have pressed until you let go of the key.

The fourth game is a short version of a game I call Target. (See Program Listing 4.) This game shows you a hill that you must shoot over to hit a randomly placed target. After you specify the angle and power of your shot, the computer traces the line of your shot. If your shot goes off the screen, the computer does not show the part of your shot that is off the screen, but does show it when it falls back onto the screen. When you do hit the target, the program tells you how many shots you took and allows you to run the program again with the target in a different position. The program contains no exceptionally difficult trigonometry, making it easy for you to follow the workings of the program.

Program Listing 5 is a computerized version of pinball. This pinball game has letters which disappear when the ball rolls over them. At the end of the game, you get 10 points for each letter you have knocked out. The ball does not gain speed when it goes downhill or lose speed going uphill, but the ball does move fast at all times. The flippers at the bottom close when you press the space bar and remain closed until you release the space bar. The object of the game is to keep the ball in play as long as you can to get as many points as possible. In this version, you get two points every time the ball bounces off anything and 10 points for each letter you knock out. Avoid using the flippers because you lose points each time you do. As in real pinball, sometimes you cannot stop the ball with the flippers because the ball slips between them. The program contains POKE and PEEK statements for the video screen memory.



Program Listing 1. *Roadrace*

```
 10 CLS :
    CLEAR 1000:
    DEFINT A - Z
 20 PRINT "HIT '" CHR$(93)"' TO TURN LEFT"
 30 PRINT "HIT '" CHR$(94)"' TO TURN RIGHT"
 40 INPUT "RATE YOURSELF AS A DRIVER (1=EXPERT, 5=NOVICE)";A
 50 CLS :
    S = 480:
    A$ = CHR$(191):
    B$ = CHR$(187) + CHR$(183)
 60 C$ = CHR$(194):
    M = 980:
    N = 1002:
    F = RND(3)
 70 PRINT @M + R,A$;:
    PRINT @N + R,A$;:
    R = R + F
 80 IF R > RND(9) + 5
     THEN
      F = RND(3) - 3
 90 IF R < RND(9) - 15
     THEN
      F = RND(3)
100 PRINT :
    D = 0:
    IF PEEK(14400) = 32
     THEN
      D = - 2
110 IF PEEK(14400) = 64
     THEN
      D = 2
120 S = S + D:
    IF PEEK(15360 + S) < > 32
     THEN
      160
130 IF PEEK(15361 + S) < > 32
     THEN
      160
140 PRINT @S,B$;:
    E = RND(A):
    IF E = 1
     THEN
      PRINT @M + R + RND(21),B$;
150 PRINT @S,C$;:
    I = I + 1:
    GOTO 70
160 CLS :
    FOR A = 1 TO 10:
     PRINT "***** CRASH *****":
     NEXT
170 PRINT :
    PRINT "YOUR SCORE IS";I:
    INPUT "PLAY AGAIN (Y/N)";A$
180 IF A$ = "Y"
     THEN
      RUN
```

Program Listing 2. *Reflex*

```
10 CLS :
   CLEAR 500:
   DEFINT A - Z
20 C$ = CHR$(182) + CHR$(179) + CHR$(185)
```

```
 30 B$ = CHR$(24) + CHR$(32):
    D$ = CHR$(191)
 40 FOR N = 1 TO 4:
     READ L(N):
     NEXT :
    DATA 448,30,509,990
 50 INPUT "HOW MANY TARGETS";P
 60 T = 0:
    PRINT @478,D$;D$;D$;:
    FOR O = 1 TO P
 70  K = 200:
     U = RND(4):
     PRINT @L(U),C$;:
     GOTO 90
 80  K = K - 15
 90  K = K - 1:
     A$ = INKEY$:
     IF A$ = ""
      THEN
       90
100  IF A$ = CHR$(91)
      THEN
       A = 2:
       GOTO 140
110  IF A$ = CHR$(9)
      THEN
       A = 3:
       GOTO 140
120  IF A$ = CHR$(10)
      THEN
       A = 4:
       GOTO 140
130  IF A$ = CHR$(8)
      THEN
       A = 1:
       GOTO 140 :
      ELSE
       80
140  IF U = A
      THEN
       T = T + K:
      ELSE
       80
150  PRINT @L(U), CHR$(195);:
     NEXT :
    CLS :
    IF T > M
     THEN
      M = T
160 PRINT CHR$(23):
    PRINT "YOUR SCORE WAS"T:
    PRINT "HIGH SCORE IS"M
170 INPUT "PLAY AGAIN (Y/N)";A$:
    IF A$ = "Y"
     THEN
      CLS :
      GOTO 60
```

Program Listing 3. *Breakout*

```
10 DEFINT A,D - W,Y:
   A$ = CHR$(179) + CHR$(145)
20 B$ = CHR$(140) + CHR$(132):
   C$ = CHR$(131) + CHR$(129)
30 FOR N = 1 TO 32:
```

Program continued

```
     D$ = D$ + A$:
     E$ = E$ + B$:
     F$ = F$ + C$:
     NEXT :
    CLS
 40 PRINT "USE " CHR$(93)" AND " CHR$(94)" TO CONTROL PADDLE"
 50 FOR N = 0 TO 2000:
     NEXT :
    CLS :
    H = 0:
    A = 0:
    P = 32:
    C = 64:
    D = 38:
    Y = - 2:
    X = RND(3) - 2
 60 FOR N = 0 TO 448 STEP 128:
     PRINT D$;E$;:
     NEXT
 70 L = 0:
    PRINT @512,F$;:
    G$ = STRING$(2,131)
 80 PRINT @P + 959, STRING$(2, CHR$(131));
 90 IF PEEK(14400) = 32
     THEN
      A = - 1
100 IF PEEK(14400) = 64
     THEN
      A = 1
110 IF D > 42
     THEN
      U = 1:
     ELSE
      IF D < 43
       THEN
        U = 0
120 IF A = 0
     THEN
      160
130 IF P + 2 * A < 1
     THEN
      P = 3
140 IF P + 2 * A > 60
     THEN
      P = 60
150 PRINT @959 + P, CHR$(194);:
    P = P + 2 * A:
    A = 0
160 PRINT @P + 959,G$;:
    IF D > 42
     THEN
      GOSUB 260
170 RESET(C,D):
    C = C + X:
    D = D + Y
180 IF C < = 0 OR C > = 127
     THEN
      X = - X:
      C = C + X
190 IF D = 0
     THEN
      Y = 2:
      X = RND(0) * 3 - 1.5
200 IF U = 1
     THEN
      240
210 IF POINT(C,D)
     THEN
      I = 4 * INT(C / 4):
     ELSE
      250
```

```
220 FOR N = 0 TO 2:
     RESET(I + N,D):
     NEXT N:
    L = L + (25 - D)
230 PRINT @889,L;:
    X = RND(0) * 3 - 1.5:
    Y = 2
240 IF C < = 0 OR C > = 127
     THEN
      X = - X:
      C = C + X
250 SET(C,D):
    GOTO 90
260 IF C > = 2 * P - 2 AND C < = 2 * P + 2
     THEN
      Y = - 2:
      U = 1:
     ELSE
      280
270 X = RND(0) * 3 - 1.5:
    RETURN
280 IF H > = 4
     THEN
      290 :
     ELSE
      H = H + 1:
      GOTO 300
290 IF H = 4 AND L > 400
     THEN
      H = H + 1:
     ELSE
      340
300 PRINT @768,"THIS IS BALL NUMBER";H + 1;
310 PRINT @832,"PRESS 'ENTER' TO CONTINUE";:
    INPUT A$:
    C = 45:
    D = 39
320 Y = - 1:
    X = RND(3) - 2:
    PRINT @832, CHR$(243):
    PRINT @768, CHR$(240);
330 PRINT @P + 959, CHR$(194);:
    P = 24:
    PRINT @983,G$;:
    RETURN
340 PRINT @705,"YOU GOT"L"POINTS";:
    A$ = INKEY$
350 PRINT @833,"PRESS ANY KEY TO PLAY AGAIN";:
    FOR N = 1 TO 400:
     NEXT
360 A$ = INKEY$:
    IF A$ = ""
     THEN
      360 :
     ELSE
      50
```

Program Listing 4. *Target*

```
10 CLS :
   PRINT TAB(25)"TARGET"
20 PRINT "THE OBJECT OF THIS GAME IS TO"
30 PRINT "DESTROY THE TARGET BY SHOOTING AT IT WITH A CANNON"
40 PRINT "WHICH IS LOCATED IN THE LOWER LEFT HAND CORNER OF THE"
```

Program continued

```
 50 PRINT "SCREEN. THE TARGET IS A '*'. THE AIMING IS DONE BY"
 60 PRINT "CHANGING THE ANGLE & POWER OF THE CANNON. AFTER EACH"
 70 PRINT "SHOT THE COMPUTER TELLS YOU BY HOW MUCH YOU MISSED."
 80 PRINT "IF IT IS NEGATIVE IT MEANS YOU OVERSHOT. IF IT IS"
 90 PRINT "POSITIVE THEN YOU UNDERSHOT THE TARGET. KEEP SHOOTING"
100 PRINT "AT THE SAME TARGET UNTIL YOU SUCCEED IN HITTING IT."
110 PRINT "YOU MUST SHOOT OVER THE HILL WHICH IS IN THE WAY"
120 HI = RND(10) + 962:
    T = RND(28) + 978:
    SH = 1
130 PRINT :
    PRINT "PRESS ANY KEY TO CONTINUE"
140 S$ = INKEY$:
    IF S$ = ""
     THEN
      140
150 CLS :
    PRINT @T,"*";:
    PRINT @960, CHR$(191);
160 PRINT @HI, STRING$(6,191);:
    PRINT @HI - 63, STRING$(4,191);
170 PRINT @HI - 126, STRING$(2,191);
180 PRINT @15,"";:
    INPUT "POWER (1-2)";PO:
    PRINT @15, CHR$(212);
190 PRINT @960, CHR$(191);:
    H = 0:
    PRINT @15,"";:
    INPUT "ANGLE";A
200 IF A > 90 OR A < 0
     THEN
      190
210 C = PO * SIN(A * .01745329):
    P = PO * COS(A * .01745329):
    D = 47:
    I = .005
220 D = D - C:
    H = H + P:
    IF D < 0
     THEN
      320
230 IF D > 47 OR H > 127
     THEN
      330
240 IF D < 39
     THEN
      310
250 IF H > (HI - 960) * 2 AND H < (HI - 954) * 2
     THEN
      260 :
     ELSE
      310
260 IF D > 44
     THEN
      350
270 IF D > 41
     THEN
      280 :
     ELSE
      290
280 IF H > (HI - 959) * 2 AND H < (HI - 955) * 2
     THEN
      350
290 IF D < 39
     THEN
      310
300 IF H > (HI - 958) * 2 AND H < (HI - 956) * 2
     THEN
      350
310 SET(H,D)
320 C = C - I:
    I = I + .001:
    GOTO 220
330 IF T - 959 = INT(H) / 2 OR T - 959 = INT((H) + 1) / 2
     THEN
      340 :
     ELSE
      350
340 CLS :
    PRINT "A HIT! YOU TOOK"SH" SHOTS TO GET ME!":
    GOTO 360
350 SH = SH + 1:
    PRINT @15,"MISSED BY"(T - 959) - INT(H) / 2;:
    GOTO 130
360 INPUT "RUN PROGRAM AGAIN (Y/N)";N$:
    IF N$ = "Y"
     THEN
      RUN
```

Program Listing 5. *Pinball*

```
 10 CLEAR 2000:
    DEFINT A - Z:
    PRINT "PRESS 'ENTER' TO START":
    INPUT A$
 20 A$ = STRING$(3,191):
    B$ = LEFT$(A$,2):
    D$ = CHR$(191):
    CLS
 30 PRINT @13, STRING$(41,191);:
    FOR N = 77 TO 973 STEP 64:
     PRINT @N,B$;
 40  PRINT @N + 39,B$;:
     NEXT :
    C$ = STRING$(10,191)
 50 FOR N = 192 TO 960 STEP 64:
     PRINT @N + 49,B$;:
     NEXT
 60 PRINT @815,B$;:
    PRINT @1011,D$;:
    PRINT @114, CHR$(130); CHR$(175);
 70 PRINT @783,B$;:
    PRINT @847,B$;B$;D$; CHR$(216);B$;B$;D$;
 80 FOR N = 1 TO 4:
     READ A:
     PRINT @A,C$;:
     NEXT :
    DATA 911,935,975,999
 90 FOR N = 1 TO 2:
     READ A:
     PRINT @A,B$;:
     NEXT :
    DATA 347,356
100 FOR N = 1 TO 2:
     READ A:
     PRINT @A,A$;:
     NEXT
110 B1 = 1:
    DATA 543,799:
    PRINT @154,"SUPERPINBALL";
120 C$ = " ":
    A$ = "O":
    B$ = " ":
    B = 64:
    N = 947:
    PRINT @120,"SCORE";
```

Program continued

```
130 PRINT @N,B$;:
    N = N - 64:
    PRINT @N,A$;:
    IF N = 179
     THEN
      150
140 FOR X = 1 TO 10:
     NEXT :
    GOTO 130
150 PRINT @N,B$;:
    N = N - 1:
    PRINT @N,A$;:
    F = 16333
160 PRINT @N,B$;:
    N = N - 65:
    PRINT @N,A$;:
    A = 15472:
    E = - 1:
    C = 63:
    G = 921
170 PRINT @113,D$;:
    PRINT @177,D$;
180 A$ = STRING$(5,191) + CHR$(197) + STRING$(5,191):
    B$ = CHR$(206)
190 IF PEEK(A + C) < 127
     THEN
      210
200 E = RND(3) - 2:
    B = - B:
    C = B + E:
    S = S + 1:
    PRINT @185,S;:
    GOTO 190
210 IF PEEK(14400) = 128
     THEN
      PRINT @G,A$;:
      S = S - 1:
     ELSE
      PRINT @G,B$;
220 IF A + C > F
     THEN
      A$ = INKEY$:
     ELSE
      POKE A,32:
      A = A + C:
      POKE A,79:
      GOTO 190
230 POKE A + C,79:
    FOR N = 1 TO 800:
     NEXT :
    PRINT @113,C$;:
    PRINT @177,C$;
240 PRINT @G,B$;:
    B1 = B1 + 1:
    IF B1 = 4
     THEN
      260 :
     ELSE
      PRINT @503,"BALL";B1;
250 A$ = INKEY$:
    IF A$ = ""
     THEN
      250 :
     ELSE
      POKE A + C,32:
      GOTO 120
260 A$ = INKEY$:
    A = 0:
    FOR N = 15514 TO 15525:
     IF PEEK(N) = 32
      THEN
```

```
       A = A + 10
270  NEXT :
    FOR N = 1 TO A:
     PRINT @185,S;:
     S = S + 1:
     FOR T = 1 TO 30:
      NEXT :
     NEXT
280 A$ = INKEY$:
    IF A$ = ""
     THEN
      280 :
     ELSE
      CLS :
      RUN
```

Rubik's Cube™ Manipulator

by Chuck Baird

Most of you have probably seen a Rubik's cube™, a puzzle made up of 27 small cubes arranged into one large cube. Each face of the large cube consists of three rows by three columns of smaller cubes, which you can rotate as a group, either clockwise or counterclockwise as you view the face. Each exposed surface of the small cubes is one of six different colors. One possible solution to the puzzle is to arrange the small cubes (through sequences of rotations of the faces) so that every side of the larger cube is a solid color. As anyone who has held a Rubik's cube knows, this is not an easy task.

My Rubik's cube (see Program Listing) is a BASIC program which shows you a Rubik's cube and allows you to manipulate it on the screen of a TRS-80. It starts with the cube solved, that is, with each face a solid color. By specifying the face and direction of rotation, you can move the faces. The computer executes your move and remembers it, giving you the option to take back a move.

The program allows the manipulation of the cube but does not solve it or attempt to solve it. Since it is written in BASIC, it runs quite slowly and takes over a second to make a move. It does, however, illustrate one method of representing a cube internally on a computer for anyone wishing to expand the program to solve the puzzle.

When you run the program, you see directions for operation and the six faces of the cube, labelled front, top, right, bottom, left, and posterior. To make a move, type the first letter of the desired face (F, T, R, B, L, P). The program scans the keyboard for input using the INKEY$ function; there is no need to press ENTER. This causes the middle cube of the face you select to flash on and off. Now type a plus sign for clockwise rotation or a minus sign for counterclockwise rotation. Any other character cancels the move.

You can recall any moves made one at a time by typing < instead of a letter. Thus, you are able to go back one move at a time to recover a previous arrangement.

Limitations

The program is written in Disk BASIC and uses the INSTR function to test acceptable keystroke inputs. To operate in Level II, use a FOR loop which uses the MID$ function. This causes only a minor loss of efficiency. For example, the BASIC statement:

```
J = INSTR("string",A$)
```

is equivalent to the following, assuming that the length of A$ is 1 and that the length of "string" is L:

```
        FOR J = 1 TO L
        IF A$ = MID$("string",J,1) THEN xxxx
        NEXT J
        J=0
xxxx next BASIC statement
```

The Program

GOSUB 570 moves face N either clockwise (if MM = 1) or counterclockwise (MM<>1) and updates the display. Each face has nine positions within it, but since the center never moves, only eight need to be numbered. One of the corners of each face is called position 0, and the remaining positions are numbered 1 through 7, going clockwise from position 0. The orientation of position 0 is not the same for all faces. The face numbers (1–6) also are assigned arbitrarily. The upper left-hand position of each face and face number as displayed on the screen are numbered as follows:

Front	face 1	0 is upper left-hand corner
Top	face 2	6 is upper left-hand corner
Right	face 3	6 is upper left-hand corner
Left	face 4	4 is upper left-hand corner
Bottom	face 5	4 is upper left-hand corner
Posterior	face 6	6 is upper left-hand corner

As a result, the numbering is consistent, simplifying rotation of the cubes.

If you draw this arrangement on a piece of paper, you will notice that any position on any face will always have fixed positions adjacent to it on the respective faces. For example, arrange one of the faces so that position 0 is in the upper left-hand corner. Now position number 6 will always be beside position 4 on the face below it (the face that would be on bottom if the chosen face were vertical) and it will be beside position 0 on the face to the left. Likewise, all positions are related to other positions on adjacent cubes.

Table 1 shows the variables the program uses. The MO array gives the information to locate the face number and the lowest numbered of the three positions that are closest (immediately adjacent) to the right, left, top, and bottom of any face. The array has four numbers for each of the six faces. These numbers show:

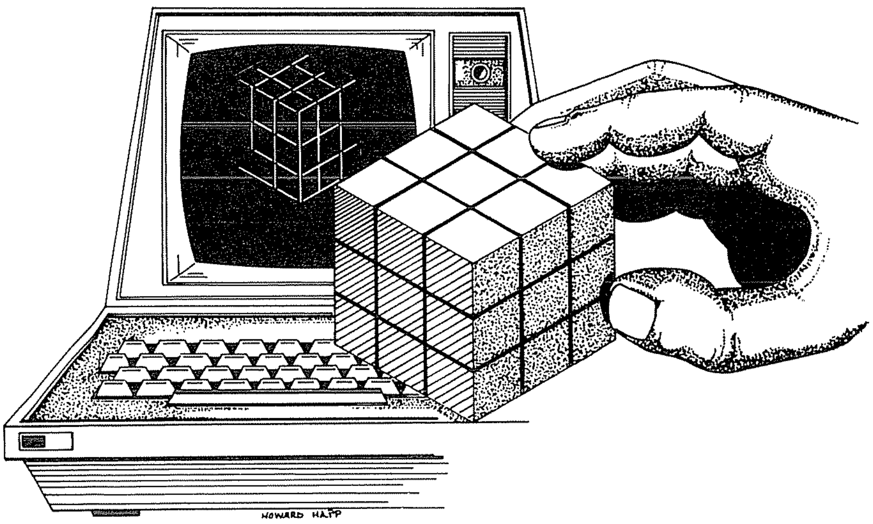
0) face number and position 4 of the top face
1) face number and position 2 of the right face
2) face number and position 6 of the bottom face
3) face number and position 0 of the left face

This information is packed as MO(I,Move) = Position*8 + Face, where I is given as 0–3 above; Move is the face to be moved (1–6); Face is the face number (1–6) of the adjacent face (top, right, bottom, or left depending on

I); and Position is the position number that is the smallest of the three positions that are adjacent to the Move face.

FA(position, face)—the color (1–6) of each small cube on a face. Faces are numbered 1–6, and positions are numbered 0–7 around the edges. The center cube on any face remains stationary and therefore is not represented in this array.
PA(position, face)—gives the number for the PRINT@ statement to display the color of this small cube.
CN(face)—gives the PRINT@ number for the center cube on each face (for flashing it).
C$(color)—has the six three-character color strings.
MO(adjacent face, face)—gives the mapping to show which small cubes on other faces must move when a face is rotated. The format is explained later.
P1(i),P2(i),P3(i)—used internally during a move to keep track of value, face, and position. Since each side of every small cube to be moved is in effect in a "ring" with three others and they will all jump to the next spot in the ring, all four are extracted prior to the move and then placed back into the next slot in the ring. There are five such rings, two on the face to be rotated, and three on the adjacent faces.
SV(move number)—used to save previous moves, up to 400 as originally written. Anyone who would type in more than 400 moves in one sitting is beyond hope.

Table 1. *Selected variables*



This scheme might be confusing at first, but there is sufficient information to make the rotations. Five of the six faces are affected by any move, although only three positions on four of those faces change. Within the rotated face, eight (all but the center) positions change.

Ideas for Expansion

A useful addition to the program would be to allow the user to blank out all small cube faces except one. This would clearly show how the individual cubes move when various faces are rotated. A simpler change would be to allow a 2 as input in addition to – and + so that two rotations (for a total of 180 degrees) could be made at once.

Program Listing. *Rubik's Cube manipulator*

```
 10 CLS :
    DEFINT A - Z:
    PRINT TAB(13);"RUBIK CUBE DIDDLE":
    PRINT
 20 PRINT "ANY OF THE SIX FACES OF THE CUBE MAY BE MOVED"
 30 PRINT "CLOCKWISE (+) OR COUNTERCLOCKWISE (-).":
    PRINT
 40 PRINT "TO SPECIFY A MOVE, TYPE THE FIRST LETTER OF THE"
 50 PRINT "NAME OF THE FACE (F,T,B,R,L, OR P).  THE MIDDLE"
 60 PRINT "SQUARE OF THAT FACE WILL THEN FLASH ON AND OFF."
 70 PRINT "THEN TYPE EITHER + OR - TO COMPLETE THE MOVE."
 80 PRINT :
    PRINT
 90 PRINT "TO TAKE BACK THE MOST RECENT MOVE, TYPE <"
100 DIM MO(3,6),FA(7,6),P1(3),P2(3),P3(3)
110 FOR I = 1 TO 6:
     FOR J = 0 TO 3:
      READ MO(J,I):
      NEXT J:
     NEXT I
120 DATA 34,19,53,4, 35,17,52,6, 33,18,54,5
130 DATA 37,22,50,1, 38,20,49,3, 36,21,51,2
140 REM
150 DIM PA(7,6),CN(6),C$(6),OF(8),SV(400)
160 NN = 0:
    M$ = "FTRLBP-+"
170 FOR I = 1 TO 8:
     READ OF(I):
     NEXT I
180 DATA 0,4,8,72,136,132,128,64
190 REM
200 PRINT :
    PRINT :
    LINE INPUT "<ENTER> TO CONTINUE ";A$
210 CLS :
    FOR I = 1 TO 6:
     READ CZ$,X,Y,W:
     C$(I) = CZ$
220  DATA WHT,36,18,0, GRN,36,3,6, RED,68,18,2
230  DATA BLU,4,18,4, ORN,36,36,4, YEL,100,18,6
240  REM
250  PZ = X / 2 + INT(Y / 3) * 64:
     FOR J = 1 TO 8:
      PU = PZ + OF(J)
260   PA(W,I) = PU:
      FA(W,I) = I:
      W = (W + 1) AND 7
270   PRINT @PU,CZ$;:
      NEXT J
280  PU = PZ + 68:
     CN(I) = PU:
     PRINT @PU,CZ$;
290  REM
300  X1 = X - 3:
     X2 = X + 24:
     Y1 = Y - 2:
     Y2 = Y + 9:
     FOR J = X1 TO X2
310   SET(J,Y1):
      SET(J,Y2):
      NEXT J:
     FOR J = Y1 + 1 TO Y2 - 1
320   SET(X1,J):
      SET(X2,J):
      NEXT J:
     NEXT I
330 REM
340 FOR I = 1 TO 6:
     READ PU,CZ$:
     PRINT @PU,CZ$;:
     NEXT I
350 DATA 140,TOP,646,LEFT,661,FRONT
360 DATA 677,RIGHT,691,POSTERIOR,841,BOTTOM
370 REM
380 A$ = INKEY$:
    IF A$ = ""
     THEN
      380
390 J = INSTR ("FTRLBP<",A$):
    IF J = 0
     THEN
      380
400 IF J < 7
     THEN
      460 :
     ELSE
      IF NN < 1
       THEN
        380
410 PU = SV(NN):
    NN = NN - 1:
    N = INT(PU / 2):
    MM = (PU + 1) AND 1
420 GOSUB 570 :
    IF NN > 0
     THEN
      540
430 PRINT @113, STRING$(13," ");
440 PRINT @177, STRING$(11," ");:
    GOTO 380
450 REM
460 PRINT @CN(J),"   ";:
    FOR G = 1 TO 20:
     NEXT G
470 PRINT @CN(J),C$(J);:
    FOR G = 1 TO 20:
     NEXT G
480 A$ = INKEY$:
    IF A$ = ""
     THEN
      460
490 PRINT @CN(J),C$(J);:
    IF A$ = "+"
     THEN
      520
500 IF A$ < > "-"
     THEN
      380
510 MM = 0:
    N = J:
    JS = J:
    GOSUB 570 :
    GOTO 530
520 MM = 1:
    N = J:
    JS = J:
    GOSUB 570
530 IF NN < 400
     THEN
      NN = NN + 1:
      SV(NN) = JS + JS + MM
540 PRINT @113,"LAST MOVE: "; MID$(M$,JS,1); MID$(M$,MM + 7,1);
550 PRINT @177,"NUMBER: ";NN;"  ";:
    GOTO 380
560 REM
570 M = MM:
    FOR I = 0 TO 1:
     P3(0) = I:
```

Program continued

```
     P1(0) = FA(I,N)
580  FOR J = 1 TO 3:
      P = (P3(J - 1) + 2) AND 7:
      P3(J) = P
590   P1(J) = FA(P,N):
      NEXT J:
     IF M < > 1
      THEN
       M = - 1
600  FOR J = 0 TO 3:
      P = (J + M) AND 3:
      PK = P3(P)
610   P = P1(J):
      FA(PK,N) = P:
      PRINT @PA(PK,N),C$(P);
620   NEXT J:
     NEXT I
630 REM
640 FOR I = 0 TO 2:
     FOR J = 0 TO 3:
      MV = MO(J,N)
650   F = MV AND 7:
      P = (((MV / 8) AND 7) - I) AND 7
660   P3(J) = P:
      P2(J) = F:
      P1(J) = FA(P,F):
      NEXT J
670  FOR J = 0 TO 3:
      PK = (J + M) AND 3:
      P = P3(PK)
680   F = P2(PK):
      PU = P1(J):
      FA(P,F) = PU:
      PRINT @PA(P,F),C$(PU);
690   NEXT J:
     NEXT I:
    RETURN
```

GRAPHICS

Easy CHR$ Graphics and Animation

Easy CHR$ Graphics and Animation

by Kenneth Lee Gibbs

Superior graphics help to stimulate interest in game and educational programs. Unfortunately, it is a difficult task to create involved graphics on the TRS-80. This program should make that task easier. It is written on a Model I Level II TRS-80 with 16K of memory. I believe it will work on the Model III.

Using SET and RESET statements to create graphics has major limitations. Graphics done this way are very slow to appear on the video display, making this method almost useless for any animation other than a single pixel moving around the screen, such as a ball or puck in a game program. Also, it is quite difficult to create complex graphics symbols using SET and RESET statements.

In most cases where complex graphics or animated graphics are desired, it is preferable to use the CHR$ graphics blocks which come as part of the TRS-80 video display character set. These blocks have ASCII code numbers from 128 to 191. Each of the graphics characters consists of six segments and occupies the space of one PRINT@ position on the video display. These 64 characters have various combinations of the six segments, on (SET) or off (RESET), ranging from all six being off, CHR$(128), to all six being on, CHR$(191).

There are three major methods for making use of the CHR$ graphics blocks. The first method is to use the PRINT statement. An example would be PRINT@ 544, CHR$(191). This will cause a solid white block to occupy a space in the middle of the video display.

The second method is to POKE the CHR$ onto the video display. TRS-80 CRT video memory addresses 15360 to 16383 correspond exactly to PRINT@ positions 0 to 1023. This means that if you POKE 15904,191, you accomplish the same thing as when you used PRINT@ 544, CHR$(191).

The third method, string packing, uses the PRINT statement but is an enhancement. There are other sources from which you can learn string-packing techniques. It is beyond the scope of this article to go into a detailed explanation.

POKEing graphics is about six times faster than using the SET statement. Using PRINT statements can be about 10 times faster than using POKE, if the CHR$s are concatenated. Concatenation is a method of tying CHR$s together with plus signs. String packing is the closest you can get to matching machine-language speed for BASIC graphics displays and it has the unique advantage of eliminating the need to clear string space in memory.

Any one of these three methods is much better than using SET statements for creating complex graphics and animation. The main problem with creating complex graphics with CHR$s is that it is a laborious process. First, you draw the intended graphic representation on a video worksheet, then you analyze each PRINT@ position to determine which CHR$ graphics block occupies that space. The solution to the problem is your TRS-80. Your computer is willing to assist you in your efforts to create CHR$ graphics, if only you ask it.

This program enables you to enlist the aid of your TRS-80. From now on, it will be much easier for you to create complex graphics and even animation. As a side benefit, you will learn a method of sketching graphics which your program can then use in any suitable manner. This program is most useful if you have a printer, but is still very useful if you don't. Using this program, you can sketch your graphics directly onto the video display. The computer then examines your sketch and tells you which CHR$ blocks can be used to duplicate it.

The routine contained in lines 420–580 is the real substance of the program. By PEEKing at each of the appropriate spaces in the video memory, you receive the ASCII code for the graphics character occupying that space. This number is stored as a subscripted variable and is later used to identify the CHR$ numbers and to reproduce the sketch.

Sketching

Program lines 270–400 are a simple sketching routine which allows the user to draw whatever is desired directly on the screen. This eliminates the

Figures From Display List

```
28  28  28  28  28  28  28  57  72  76  28  28  28
28  28  28  28  28  28  28  49  28  30  73  76  28
28  28  28  28  28  28  28  49  28  28  28  30  31
28  28  28  84  88  88  80  49  28  28  28  28  28
28  28  30  75  91  91  59  29  28  28  31  28  28
```

For this example, I drew the figure in the upper left corner of the sketch area; therefore, all numbers to the right of PRINT@ position 13 are 28s and are ignored. Notice how unnecessary 28s are left out of the program lines, but that the necessary 28s are used. Also notice how the numbers allow you to calculate easily the necessary quantity of back spaces, (CHR$(24)). Remember to include the missing 1 in these numbers. For example, 28 means CHR$(128).

Figure 1. *Display list of CHR$(n)s for sample sketch*

necessity of drawing in blocks on a video worksheet. I have limited the sketch area to eight lines of 32 PRINT positions each. This represents one quarter of the screen. You may alter this to suit your needs, but I find that almost anything I want to draw on the screen which is larger than that is not likely to be very complicated.

The box that surrounds the sketch area is there to help you determine the dimensions of your sketch. When you are designing graphics symbols which must fit into an area of specific size, it is best to use the upper left corner of the sketch area. Using the marks on the box, count off the number of PRINT spaces you want for the width of your sketch and the number of lines you want for the height. Etch in the boundaries for the size of your graphics symbol and erase the boundaries before requesting the list of CHR$ numbers. Don't worry about the box. It is not included in the display list.

Getting the CHR$(n)s

When your sketch is the way you want it, press the ENTER and CLEAR keys at the same time. This freezes the sketch on the screen and program execution goes to lines 420–580. These lines induce the computer to examine each PRINT@ position in the sketch area and assign a CHR$(n) to it. This takes about seven seconds.

Lines 600–710 produce the display list of the CHR$(n)s. I have the numbers displayed without the leading 1 so that all of the numbers, except the last one, can be displayed on the screen together. Remember to include the missing 1 when using the numbers. For example, if the number 56 is on the screen, it really means CHR$(156). The numbers are displayed consecutively from the top left of the sketch area. Each line of the sketch area is represented by two lines in the display list. This format will seem simple once you get used to it.

If you are fortunate enough to own a printer, you can have the numbers LPRINTed for you. I have set up the LPRINT routine (lines 1000–1470) to print out the numbers in a 32-column, eight-line matrix which corresponds to the layout of the sketch area. Just turn the printout sideways so that all of the leading 1s are facing the bottom of the page, and your list of numbers will be in the exact sequence and similar in form to the sketch area. If you don't have a printer, the easiest way to copy the numbers is to draw an 8-by-32 grid on a sheet of legal-size tablet paper. Copy the numbers from the display list, writing from left to right so that the first two rows of numbers from the video display list become the first line on your paper. This gives you the same type of layout as having the numbers LPRINTed.

Using the Numbers

Outline the numbers which are essential to your drawing. All of the CHR$(128)s outside of your actual sketch can be ignored. Next, determine how many CHR$(24)s are needed to back space the cursor to start the next string with the first CHR$ in the next line. Insert a CHR$(26) at the end of each line to drop the cursor. Any time you have the opportunity to turn consecutive CHR$(n)s into STRING$, do so. For example, if you have 12 consecutive CHR$(191)s, convert them into one STRING$(12,191).

Concatenate all of the CHR$(n)s for each line into one string. Add a CHR$(26) and the appropriate number (n) in a STRING$(n,24) to back space and drop the cursor one line. Do the same for each following line. Leave out the CHR$(26) and STRING$(n,24) on the bottom line. Concatenate all of those strings into one string if the total number of CHR$s used is less than 256. Your entire sketch appears on the video display screen almost instantly when you assign one PRINT@ position for the upper left corner of the sketch. Study Figures 1 and 2 to better understand this process. If you are using string-packing techniques, the numbers can be used exactly as they are for the DATA lines.

```
10   CLS : CLEAR 200
20   A$ = CHR$(157) + CHR$(172) + CHR$(176) + CHR$(26) + STRING$(3,24)
30   B$ = CHR$(149) + CHR$(128) + CHR$(130) + CHR$(173) + CHR$(176) +
      CHR$(26) + STRING$(5,24)
40   C$ = CHR$(149) + STRING$(3,128) + CHR$(130) + CHR$(131) + CHR$(26) +
      STRING$(10,24)
50   D$ = CHR$(184) + STRING$(2,188) + CHR$(180) + CHR$(149) + CHR$(26) +
      STRING$(6,24)
60   E$ = CHR$(130) + CHR$(175) + STRING$(2,191) + CHR$(159) + CHR$(129) +
      STRING$(2,128) + CHR$(131)
70   X$ = A$ + B$ + C$ + D$ + E$
80   PRINT@ 288, X$;
90   GOTO 90
```

Figure 2. *Program to generate sample sketch from Figure 1.*

Animation

Notice that the computer redraws your sketch after displaying the list of CHR$(n)s. This gives you the opportunity to make changes in the sketch. Making changes is how animation is done. To create animated graphics, erase small portions of your first sketch and add a few more graphics blocks in the appropriate places. Get the numbers for this new sketch. Do this until your sketch reaches the final position desired in the animation sequence. Compare the CHR$(n)s for each sketch in the sequence. Once your original sketch is on the screen, you need only change the PRINT@ positions for the CHR$(n)s that change during the sequence. Most of the numbers will remain the same unless you are creating a really complicated animation.

Using the Routine

With what you now know or can learn by studying the program, you can write programs which allow you to create your own graphics symbols. These user-created graphics could be personalized spaceships, buildings, mazes, and so on—use your imagination!

By creating a space of known proportions and a sketch routine, you can reproduce these user-created graphics in your program in the original space in which they were sketched, draw them in another place, or even animate them. When you have finished sketching, your program examines each of the video memory addresses through the PEEK function. Store each PEEK result as a subscripted variable. You can then reproduce the graphics symbols any time.

There you have it—a program that tells you what CHR$(n)s you are using instead of asking you what numbers you want to use. Notice line 60. This POKE will prevent accidental BREAKing of the program which could result in a sketch being disturbed. It also prevents intentional use of the BREAK key; so you will have to reach around to the back of your CPU and press the reset button in order to LIST the program.

There are several aspects of the program that can be changed. You can alter the size of the sketch area to better suit your needs, or you might want to change the format for displaying or LPRINTing the numbers. I hope this program makes creating your graphics easier, and that you will use more graphics in your programs to make them more interesting.

Program Listing. *Easy CHR$ Graphics and Animation*

```
 10 :
    ' ................EASY CHR$ GRAPHICS & ANIMATION..............
 20 :
    ' ...........2ND VERSION................12/4/81...............
 30 :
    ' ......................KENNETH LEE GIBBS.....................
 40 :
    ' ......................31 WILLOW STREET......................
 50 :
    ' ......................HIGHSPIRE, PA 17034...................
 60 POKE 16396,23:
    REM ...POKE16396,22 TO ENABLE BREAK KEY
 70 DEFINT A - Z
 80 CLS :
    DIM M(256)
 90 GOTO 730
100 :
    ' .............SUBROUTINE TO SET UP SKETCH AREA.................
    .
110 CLS :
    X = 63:
    Y = 24
120 FOR Z = 32 TO 96:
     SET(Z,10):
     SET(Z,37):
     NEXT
130 FOR Z = 34 TO 95 STEP 4:
     RESET(Z,10):
     RESET(Z + 1,10):
     RESET(Z,37):
     RESET(Z + 1,37):
     NEXT
140 FOR Q = 11 TO 36:
     SET(28,Q):
     SET(29,Q):
     SET(98,Q):
     SET(99,Q):
     NEXT
150 FOR Q = 12 TO 35 STEP 2:
     RESET(29,Q):
     RESET(98,Q):
     NEXT
160 FOR Q = 14 TO 35 STEP 3:
     SET(27,Q):
     SET(100,Q):
     NEXT
170 NU = 1
180 FOR L = 266 TO 714 STEP 64
190  PRINT @L,NU;:
     PRINT @L + 41,NU;
200  NU = NU + 1:
     NEXT L
210 PRINT @80,"1 3 5 7 9 1 3 5 7 9 1 3 5 7 9 1";
220 PRINT @26,"1 1 1 1 1 2 2 2 2 2 3";
230 FOR N = 144 TO 174 STEP 2:
     PRINT @N, CHR$(92) + " ";:
     NEXT
240 PRINT @838,"PRESS <ENTER> AND <CLEAR> WHEN READY TO GET NUMBERS.
    "
250 PRINT @909,"<SPACE BAR> AND <CLEAR> TO START OVER."
260 :
    ' .................SUBROUTINE FOR USER SKETCHING...............
    .
270 PK = PEEK(14400)
280 IF PK AND 8
     THEN
      Y = Y - 1
290 IF PK AND 16
```

```
     THEN
      Y = Y + 1
300 IF PK AND 32
     THEN
      X = X - 1
310 IF PK AND 64
     THEN
      X = X + 1
320 IF X > 95
     THEN
      X = X - 1
330 IF X < 32
     THEN
      X = X + 1
340 IF Y > 35
     THEN
      Y = Y - 1
350 IF Y < 12
     THEN
      Y = Y + 1
360 IF PK = 130
     THEN
      110
370 IF PK > 120
     THEN
      RESET(X,Y):
      GOTO 270
380 IF PK = 3
     THEN
      400
390 RESET(X,Y):
    FOR T = 1 TO 9:
     NEXT :
    SET(X,Y):
    GOTO 270
400 SET(X,Y):
    PRINT @0, CHR$(30):
    PRINT @64, CHR$(30):
    PRINT @128, CHR$(30);:
    PRINT @76,"GIVE ME A FEW SECONDS TO FIGURE THIS OUT...";
410 :
    ' ................COMPUTER DETERMINES CHR$(#)'S.................
    .
420 V = 1
430 FOR P = 15632 TO 15663:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
440 FOR P = 15696 TO 15727:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
450 FOR P = 15760 TO 15791:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
460 FOR P = 15824 TO 15855:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
470 FOR P = 15888 TO 15919:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
480 FOR P = 15952 TO 15983:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
490 FOR P = 16016 TO 16047:
     M(V) = PEEK(P):
```

Program continued

```
     V = V + 1:
     NEXT
500 FOR P = 16080 TO 16111:
     M(V) = PEEK(P):
     V = V + 1:
     NEXT
510 FOR V = 1 TO 256
520  IF M(V) = 32
      THEN
       M(V) = 128
530  NEXT
540 CLS
550 PRINT @512,"REMEMBER TO ADD 100 TO THE FOLLOWING NUMBERS."
560 PRINT "I HAD TO LEAVE OFF THE 1 TO FIT THEM ALL IN..."
570 PRINT :
    PRINT "PRESS <ENTER> TO SEE THE NUMBERS. THEY READ LIKE THIS -":
    PRINT "THE FIRST TWO ROWS OF NUMBERS ARE THE CHR$(NUMBERS)-100 F
    OR":
    PRINT "THE TOP LINE IN THE SKETCH BOX. THE 3RD AND 4TH ROWS ARE
    THE":
    PRINT "NUMBERS FOR THE 2ND SKETCH LINE...ETC."
580 INPUT DU$:
    CLS
590 :
    ' .................CHR$(#)'S ARE DISPLAYED ON SCREEN............
    .
600 FOR V = 1 TO 255
610  PRINT M(V) - 100;
620  NEXT V
630 PRINT @1019,"";:
    INPUT DU$
640 CLS :
    PRINT M(256)
650 PRINT :
    PRINT "  [ - THAT'S THE cHR$(NUMBER) FOR THE LOWER RIGHT CORNER.
    "
660 PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT "PRESS THE <D> KEY IF YOU WISH TO RESUME DRAWING.":
    PRINT :
    PRINT "PRESS THE <P> KEY IF YOU DESIRE TO HAVE THE NUMBERS LPRIN
    TED."
670 Z$ = INKEY$
680 IF Z$ = "D"
     THEN
      880
690 IF Z$ = "P"
     THEN
      1000
700 GOTO 670
710 CLS :
    GOTO 880
720 :
    ' ........................INSTRUCTIONS.........................
    .
730 PRINT "  THIS PROGRAM CONVERTS SKETCHES THAT YOU MAKE ON THE SCF
    EEN"
740 PRINT "INTO THE APPROPRIATE CHR$(NUMBERS) WHICH YOU CAN THEN USI
    "
750 PRINT "WITH POKE, PRINT@, OR STRING-PACKING TECHNIQUES TO REPROI
    UCE"
760 PRINT "THE SAME SKETCH WITHIN A PROGRAM."
770 PRINT :
    PRINT "  USE THE ARROW KEYS TO SKETCH UP, DOWN, OR DIAGONALLY."
780 PRINT "TO SKETCH DIAGONALLY, HOLD DOWN BOTH A L-R ARROW AND AN l
    -D"
790 PRINT "ARROW AT THE SAME TIME. TO ERASE, HOLD DOWN THE SPACE BAF
    "
```

```
800 PRINT "AND THE ARROW(S) WHICH WILL MOVE THE CURSOR IN THE DIRECT
    ION"
810 PRINT "THAT YOU WISH TO ERASE."
820 PRINT :
    PRINT "  WHEN YOU ARE READY FOR THE COMPUTER TO CONVERT YOUR SKE
    TCH"
830 PRINT "INTO CHR$(NUMBERS), LEAVE THE BLINKING CURSOR IN A SPOT '
840 PRINT "THAT YOU WANT TO BE SET."
850 PRINT :
    INPUT "PRESS THE <ENTER> KEY WHEN READY TO START DRAWING";DU$
860 CLS :
    GOTO 110
870 :
    ' ................SKETCH IS RE-DISPLAYED ON SCREEN.............
    .
880 CLS :
    V = 1
890 FOR P = 272 TO 303:
     GOSUB 980:
     NEXT
900 FOR P = 336 TO 367:
     GOSUB 980:
     NEXT
910 FOR P = 400 TO 431:
     GOSUB 980:
     NEXT
920 FOR P = 464 TO 495:
     GOSUB 980:
     NEXT
930 FOR P = 528 TO 559:
     GOSUB 980:
     NEXT
940 FOR P = 592 TO 623:
     GOSUB 980:
     NEXT
950 FOR P = 656 TO 687:
     GOSUB 980:
     NEXT
960 FOR P = 720 TO 751:
     GOSUB 980:
     NEXT
970 GOTO 120
980 PRINT @P, CHR$(M(V));:
    V = V + 1:
    RETURN
990 :
    ' .......................LPRINT SUBROUTINE....................
    .
1000 CLS :
     INPUT "ENTER A NAME OR REFERENCE NUMBER FOR THIS SKETCH";DN$:
     PRINT :
     PRINT :
     INPUT "ENTER A SEQUENCE NUMBER FOR THIS SKETCH";DS$:
     CLS
1010 IF PEEK(14312) > 127 PRINT "PRINTER NOT READY":
     PRINT :
     PRINT :
     INPUT DU$:
     GOTO 1010
1020 PRINT "PRINTING.........."
1030 LPRINT " "
1040 LPRINT "REFERENCE NUMBER OR NAME:     ";DN$
1050 LPRINT "SEQUENCE NUMBER:     ";DS$
1060 LPRINT " "
1070 LPRINT " "
1080 LPRINT "LINE LINE LINE LINE LINE LINE LINE LINE"
1090 LPRINT " #8   #7   #6   #5   #4   #3   #2   #1"
1100 LPRINT " "
1110 FOR V = 8 TO 1 STEP - 1:
      LPRINT M(V);:
```

Program continued

```
      NEXT :
     LPRINT " - COLUMN 1"
1120 FOR V = 16 TO 9 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 2"
1130 FOR V = 24 TO 17 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 3"
1140 FOR V = 32 TO 25 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 4"
1150 FOR V = 40 TO 33 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 5"
1160 FOR V = 48 TO 41 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 6"
1170 FOR V = 56 TO 49 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 7"
1180 FOR V = 64 TO 57 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 8"
1190 FOR V = 72 TO 65 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 9"
1200 FOR V = 80 TO 73 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 10"
1210 FOR V = 88 TO 81 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 11"
1220 FOR V = 96 TO 89 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 12"
1230 FOR V = 104 TO 97 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 13"
1240 FOR V = 112 TO 105 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 14"
1250 FOR V = 120 TO 113 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 15"
1260 FOR V = 128 TO 121 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 16"
1270 FOR V = 136 TO 129 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 17"
1280 FOR V = 144 TO 137 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 18"
```

```
1290 FOR V = 152 TO 145 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 19"
1300 FOR V = 160 TO 153 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 20"
1310 FOR V = 168 TO 161 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 21"
1320 FOR V = 176 TO 169 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 22"
1330 FOR V = 184 TO 177 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 23"
1340 FOR V = 192 TO 185 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 24"
1350 FOR V = 200 TO 193 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 25"
1360 FOR V = 208 TO 201 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 26"
1370 FOR V = 216 TO 209 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 27"
1380 FOR V = 224 TO 217 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 28"
1390 FOR V = 232 TO 225 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 29"
1400 FOR V = 240 TO 233 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 30"
1410 FOR V = 248 TO 241 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 31"
1420 FOR V = 256 TO 249 STEP - 1:
      LPRINT M(V);:
      NEXT :
     LPRINT " - COLUMN 32"
1430 LPRINT " "
1440 LPRINT " "
1450 LPRINT " "
1460 PRINT "FINISHED PRINTING.....PRESS <ENTER>"
1470 PRINT :
     INPUT DU$:
     CLS :
     GOTO 880
```

HARDWARE

Memory Size–20K!

Memory Size—20K!

by W. R. Stanley N4TF

When you upgrade your computer from 4K to 16K, you have a spare set of 4K memory chips. Since there are occasions when even the new 16K memory is not enough, this article provides information for putting that spare 4K set of memory chips to good use. Here is a way to use the chips without making internal changes to the keyboard unit.

Background

Correct operation of any external circuit connected to the keyboard will depend upon appropriate address selection, your use of the data bus, and the control signal you employ in an exact sequence. For example, an external memory cell location must be addressed to the exclusion of all other cell locations in that memory if it is to share in data transfers to and from the keyboard.

The parallel eight-bit data bus at the keyboard connector is an extension of the data bus routed inside the keyboard to several discrete memory blocks: ROM, RAM, video memory, and the keyboard itself. The external memory circuit to be added to the computer must be brought onto the data bus only when data is to be written to or read from that memory block. Address and control signals from the keyboard ensure that this external memory block is enabled only when called upon by the CPU. At all other times the external memory block must appear non-existent to the main computer.

The 4K dynamic chip as used in the TRS-80 is packaged in the standard 16-pin DIP package. One pin serves as the data input point, while a second one is used for data bit output. Three pins are used for power inputs, +12, +5, and −5 V. A sixth pin is grounded for signal and power path returns, and a seventh pin is used for a Chip Select control signal to activate the memory chip for a read or write operation. Still another pin receives a control signal telling the chip whether the current operation is a read or a write.

Eight of the 16 pins are now in use and eight pins are still available, but we must apply 12 bits of addressing information to the chip designating the exact memory cell locations to be affected by the chosen operation. To understand why 12 bits of information are needed, consider the number of address line bits necessary to address each of the 4,096 memory cell locations on the chip, 0000 to 0FFF hex. Two raised to the 12th power equals 4,096.

The Memory Matrix

The 16-pin memory chip addresses its memory as a matrix in rows and columns. Picture a grid of 64 horizontal wires overlaid by 64 vertical wires ($64 \times 64 = 4{,}096$). Any intersecting point in the grid can be located by specifying the row number and column number.

The memory chip is addressed by applying data that designates the row number of the desired memory bit location, latching that row address into the chip logic and control circuits, and then latching the column address presented a short time later. This address method, called multiplexing, can address any location on the 4K chip using only six address pins instead of 12. Two additional pins on the chip are used for signal inputs to tell the chip's internal circuits whether a row-address group or a column-address group is being entered. All 16 pins are now accounted for.

Consider one more function. The value of a given data bit stored at a particular address on the dynamic memory chip is represented by the charge level (high or low) of a capacitor at that memory matrix location. The level of charge on a practical capacitor changes over a period of time. This charge must be restored periodically at each capacitor cell location. This is known as the refresh operation. In the TRS-80, the Z-80 CPU performs refresh operations at the same time it decodes a machine instruction. A special register on the CPU chip keeps track of the row address groups and ensures that all memory matrix row addresses are accessed in the proper order. Regardless of the amount of dynamic memory on line, all cells will be refreshed approximately 500 times per second.

Block Diagram

Figure 1 is a block diagram of a 4K dynamic memory card that can be directly connected to the keyboard. Its address decoder is the principle circuit that determines when external memory is brought on line.

Since all locations in a 4K memory block can be addressed using 12 address lines (A_0-A_{11}), and an additional four address line bits (A_{12}-A_{15}) are available, those four bits can be used to arbitrarily assign an address block location to the external memory board. The logical place to locate the additional memory block is in the range of 8000–8FFF hex. This places the external memory immediately above the internal 16K block, whose highest address is 7FFF hex, without leaving a gap.

The address decoder constantly monitors the four highest-order address line bits. It outputs an enabling signal only when the computer address bus contains addresses in the 8xxx hex range (when A_{15} is high, and A_{12} through A_{14} are low). If the address decoder circuit senses any other combination of signals on the address bus, it disconnects the remainder of the memory board circuits.

The address multiplexer (Figure 1) performs as an electronic six-pole, double-throw switch. In one position of the switch, address lines A_0-A_5 connect to the memory chips. In the other position, lines A_6-A_{11} are routed to the same chips. The MUX (Multiplexer) signal, which is the output of a flip-flop in the keyboard, determines the electrical position of the double-throw switch. The MUX signal has a predetermined phase relationship to the $\overline{\text{RAS}}$ (Row Address Strobe Not) and $\overline{\text{CAS}}$ (Column Address Strobe Not) signals that are also generated inside the keyboard. When the $\overline{\text{RAS}}$ signal is present, either during the early part of a memory read or write cycle, or during a refresh operation, the phase of the MUX signal is such that the address multiplexer outputs the low-order address group (A_0-A_5) to the memory chips.

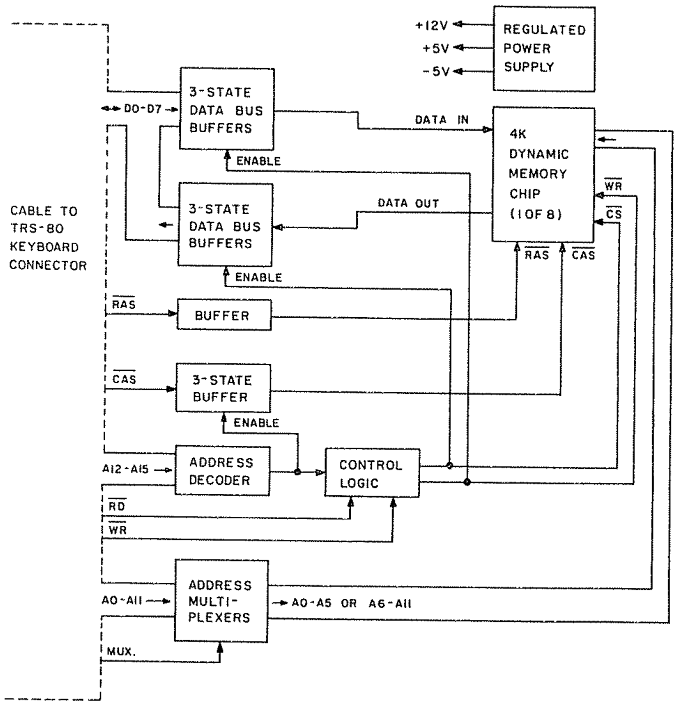


Figure 1. *Block diagram of the 4K dynamic memory board*

When the $\overline{\text{CAS}}$ signal is present, the MUX signal phase is reversed, and the multiplexer outputs the high-order address group (A_6-A_{11}) to the chips. Note that the $\overline{\text{RAS}}$ signal is applied through a buffer to memory. Buffering this signal causes the keyboard circuit that provides the $\overline{\text{RAS}}$ signal to see only one additional load rather than eight more. Note that the $\overline{\text{CAS}}$ signal is applied through a three-state buffer. This buffer is operational only when the address decoder senses addresses in the 8xxx hex range. The $\overline{\text{CAS}}$ signal is therefore applied to the external memory chips only when those chips are written to or read from.

Two additional control signals from the keyboard connector tell the external memory whether it is being written to or read from. These signals are, respectively, $\overline{\text{WR}}$ (Write Not) and $\overline{\text{RD}}$ (Read Not). Note that these signals are also combined with the address decoder output so that the memory chips are not placed in the read or write condition unless the proper address range is on the computer address bus. The $\overline{\text{CS}}$ (Chip Select Not) signal is present when the $\overline{\text{RD}}$ signal and the proper address decoder output appear in the same time frame.

The eight-bit data bus connections at the memory board are buffered in both directions. Three-state buffers are employed here so that the direction of data flow can be controlled by the keyboard. One set of buffers activates when the computer is writing to the external memory, and the other data buffer set is enabled when the computer wishes to read data from that memory section. When this particular block of memory is not being accessed for memory transfer operations, both data bus buffers go to the high-impedance state and, in effect, disconnect the memory chips from the data bus.

Construction

The schematic diagram of the 4K memory board is shown in Figure 2. The layout is not critical but I suggest that you spend a little time trying different component arrangements before you start wiring.

I constructed my circuit on a Radio Shack (cat. #276-152) Plug-In Breadboard. This card provides ample room for construction of the memory circuit. It also has 44 pins on its board edge, which means that its pin number assignments can be directly correlated with those of the keyboard connector, and still have four uncommitted pins available for connecting the card to an external power supply.

The mating connector is also available at Radio Shack. The proper mating connector for the keyboard is an AMP P/N 88103-1 or its equivalent. You can get a 40-pin connector with 0.1-inch contact spacing at Priority One Electronics.

Note that the pin number assignments on the keyboard connector do not necessarily agree with the order indicated on your mating 40-pin connector. With the keyboard in front of you, pin 1 will be on the top of the circuit board at the connector end farthest removed from the RESET button. The odd-numbered pins progress in ascending order to the left, with pin 39 on the top left edge. All even-numbered pins are on the underside of the keyboard connector, with pin 2 directly underneath pin 1. Pin 40 is closest to the RESET button.

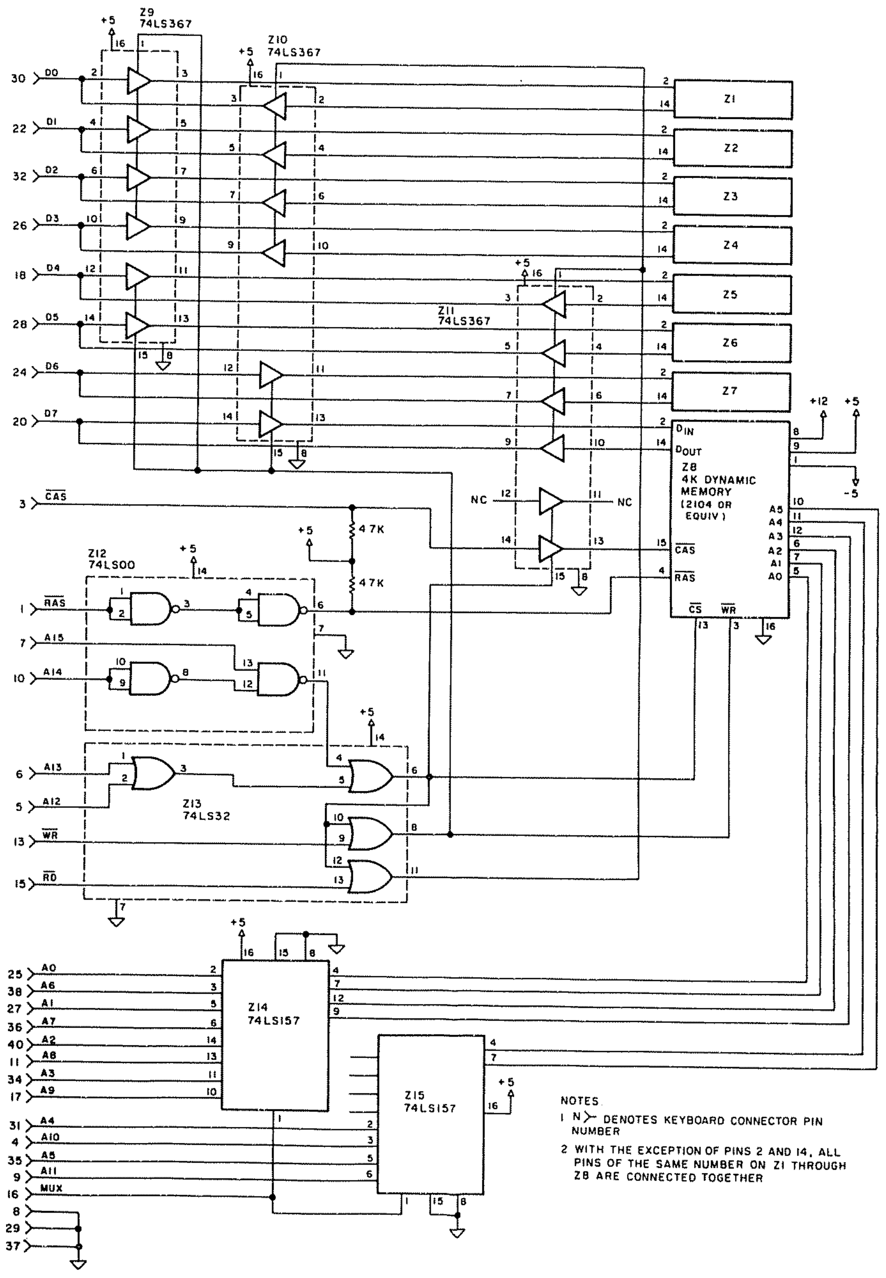

Z9 74LS367
Z10 74LS367
Z11 74LS367
Z12 74LS00
Z13 74LS32
Z14 74LS157
Z15 74LS157
Z8 4K DYNAMIC MEMORY (2104 OR EQUIV)
Z1
Z2
Z3
Z4
Z5
Z6
Z7
47K
47K
NC
+5
+12
-5
D0
D1
D2
D3
D4
D5
D6
D7
CAS
RAS
A15
A14
A13
A12
WR
RD
A0
A6
A1
A7
A2
A8
A3
A9
A4
A10
A5
A11
MUX
NOTES
1 N> DENOTES KEYBOARD CONNECTOR PIN NUMBER
2 WITH THE EXCEPTION OF PINS 2 AND 14, ALL PINS OF THE SAME NUMBER ON Z1 THROUGH Z8 ARE CONNECTED TOGETHER

Figure 2. *Schematic diagram of the 4K dynamic memory board*

Be sure to label the connector that plugs into the keyboard so that its position is correct before insertion. Use dots of fingernail polish or some other marking medium to indicate proper orientation between the memory card and its mating connector. If either cable connection is reversed, you might damage your system. You can use either ribbon cable or individual stranded wires to join the two connectors. I suggest a maximum length of one foot for either type. I urge you to employ sockets for all ICs. They are good insurance against heat and static damage for the memory chips. They also make troubleshooting and replacing components much easier. Incidentally, either solder or wire-wrap techniques can be used. The more ambitious constructor might consider fabricating a printed-circuit board.

Wiring Procedures

Make all necessary power and ground connections to all chip sockets. Next, connect all indicated like-numbered pins in parallel on the eight memory IC sockets. Make all connections between the memory chip sockets and the remainder of the components on the board. Following this, wire the address decoder and logic circuits. The last area to be wired is between the card edge pins and the appropriate ICs. Work slowly, and take a break now and then. Double-check your wiring. Unless printed-circuit techniques are used, you will end up with several layers of wires placed on top of each other. It becomes difficult to find and correct a wiring error that is buried.

If you arrange the memory sockets in parallel rows on the card, cross-connect the power buses between adjacent chips if possible. This method will lower the impedances of the power distribution leads. You are dealing with digital waveforms containing frequencies in the VHF range, and a short piece of wire at high frequencies appears to many circuit components as much more than just a low resistance current path for direct current flow.

Power supply pins should be bypassed, especially around memory ICs. A 0.1-uF disk capacitor bypassing the +12-V supply pin at every other chip and a 0.1-uF disk at the +5-V pin on alternate memory chip sockets are recommended. A 0.01-uF disk located at the −5-V supply pin on every other memory socket is also advisable. Several 0.01 to 0.1-uF disk capacitors located at the +5-V supply pins on several of the remaining IC sockets would help maintain smooth operation.

One reason for paying particular attention to supply pin bypassing around the memory chips is that high peak currents occur during certain portions of the memory IC operating cycle. Although the average current at each IC is not excessive, those sudden high-current demands could not be met by power supplies located several feet away. One solution to these abrupt current increases is to connect large capacitors in the immediate vicinity of the memory chips. The memory board doesn't require much power, but you will find that it costs only a little more to build a power supply that will operate with other circuits. In other words, overbuild the power supply, unless you are interested in the smallest possible package.

Junk-box Transformer

I used a junk-box transformer with two low-voltage, center-tapped secondaries. The wires between the windings and the terminals on the transformer appeared to be about the right size to handle at least one amp each. A bridge rectifier across one winding, with the center-tap left disconnected, feeds a 7812 regulator. Both input and output of this regulator chip are heavily bypassed with electrolytics. In addition, I placed a 0.1 uF disk at both of its active terminals, very close to the package.

A full-wave rectifier is connected across the other secondary winding, whose center-tap is grounded. This rectifier feeds a 7805 regulator whose input and output are also heavily bypassed. Another full-wave rectifier, with the diodes reversed from those for the +5-V supply, is connected to another regulator-filter combination. Here a 7905 negative voltage regulator is used. Many parts suppliers stock the equivalent of the 7905, listing it as the LM320K-5. A 5-V zener diode, bypass capacitor and series dropping resistor can be used in lieu of the 7905 (or equivalent), the total cost is about the same. However, this zener regulator does not provide the thermal and overload current protection of the regulator package.

Mount the 7812 and 7805 on separate small heat sinks, or attach suitable radiators to them. Both of these chips get rather warm. The 7905 does not require heat-sinking because its load current is much lower.

16K Memory Board

You can modify the 4K memory board making it plug compatible to that new set of 16K chips. You need change only the address decoder and address multiplexer circuits to operate the added 16K set in the address range from 8xxx to Bxxx hex. Figure 3 shows a partial schematic diagram.

Compare the schematics in Figures 2 and 3. Seven address lines are connected to the 16K chips. Only six lines went to the 4K set. This means that a total of 14 addressing bits are applied to the 16K chips, in two groups of seven. The row and column matrix in the 16K chip is a 128-by-128 grid; therefore, seven bits per multiplexed group are necessary.

Adding one address bit per group means that you must add two additional address lines to the multiplexer. This is shown in Figure 3.

Remember that all 16 pins on the 4K chip are used. What else must you change when the seventh address line is added to the 16K chip? The 4K chip has a $\overline{\text{CS}}$ pin which enables the chip. Note that the 16K chip has no such pin designation. The manufacturer designed the 16K chip so that it is fully functional when the $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and addressing signals are present at the correct times.

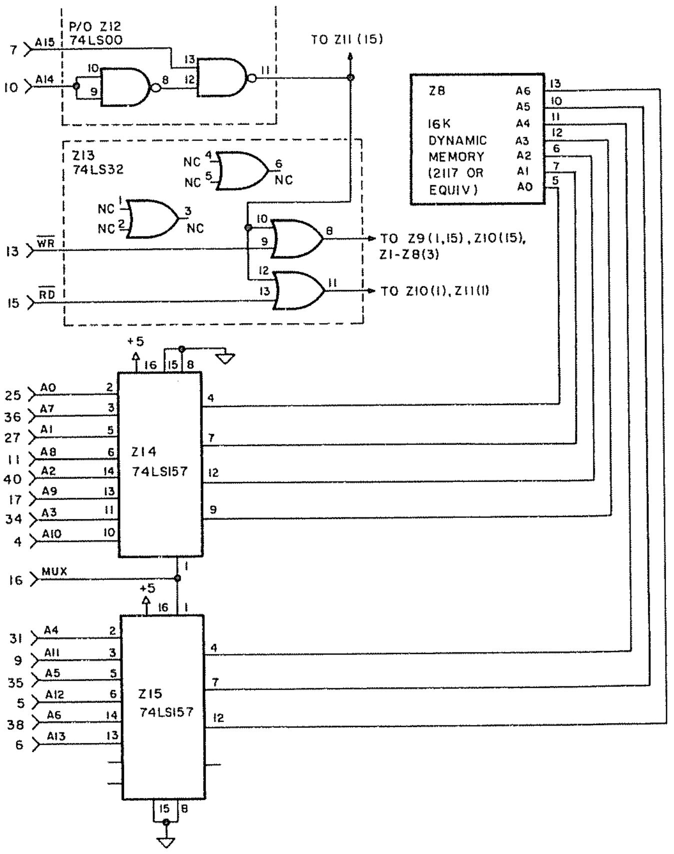


Figure 3. *Partial schematic diagram of the 16K dynamic memory board*

Look at the address decoder circuits on the two schematics. The decoder for the 16K memory circuit is simpler, because it responds to a wider range of addresses than the one for the 4K board. The 4K board is accessed only when A_{15} is high, and A_{12} through A_{14} are low. This represents an address within the 8xxx hex range. The 16K board is addressed in the 8xxx-Bxxx hex range. This particular range of addresses is present any time A_{15} is high and A_{14} is low. The address decoder on the 16K board, therefore, only has to monitor two address lines, rather than four.

When you substitute that set of 16K chips in the keyboard for the original 4K set, you have to reconfigure several jumpers. You are actually rearranging the address decoder output to respond to a wider range of dynamic memory addresses. You also change the $\overline{CS}$ signal line going to the 4K set to an additional multiplexed address line going to the 16K set.

Closing Comments

I have tried to present enough background information to enable you to substitute parts or rearrange logic and control circuits. There is always room for improvement on someone's ideas or techniques, and this construction article is no exception. You will find many instances when the external memory is unnecessary. Instead of wearing out the keyboard connector, merely turn off the power supplies feeding the board. One word of caution in this area: If you initialize the computer with the external memory energized and, later, remove power to the external circuit, the computer will, in some instances, use the external memory as if it were still available. The result is incorrect operation and/or lost data.

A good way of reserving a block of upper memory for machine-language programs is to power on while the external memory is still off. After the computer is initialized, turn on the external memory power supplies. The computer won't find the added memory, unless it gets trapped in a loop that asks you the MEMORY SIZE question (Level II).

Loss of Data

Concerning loss of data in the added memory block—especially the 16K add-on, I experienced a problem when the circuits for both the 4K and 16K blocks were built and under test. Difficulties with the 4K block disappeared when additional filtering and bypassing were added onto the −5-V line on the memory board. The −5-V supply is very lightly loaded by the memory chip; as a result this supply line is very susceptible to noise. Two or three 50 or 100 microfarad electrolytics and a handful of .01 to .1 microfarads distributed up and down the −5-V distribution line should bypass the noise picked up on this line.

If you connect an oscilloscope to a −5-V line that isn't properly bypassed, you will probably see at least one-half volt of noise. Add large and small bypass capacitors until the noise signal voltage is radically decreased. Bypassing will cure many ailments with the 16K memory board. The major cause of other difficulties is the addition of fast (200 or 250 ns) memory chips external to the keyboard if the chips inside the keyboard are the standard 450 ns speed. If all else fails to settle down operations of the 16K add-on memory board, swap those fast chips with the ones inside the keyboard.

HOME APPLICATIONS

Disk BASIC Word Processor
The Big Game

Disk BASIC Word Processor

by Delmer D. Hinrichs

This word processor program is written in BASIC for the TRS-80 Model I and Model III computers. It is based on my program published in the *Encyclopedia for the TRS-80*, Volume 2. I have revised it extensively for disk operation and for greater speed and convenience. It requires the extra enhancements of Disk BASIC. Reserve only one disk file when you load BASIC from DOS. Otherwise, you may run out of memory when you run the program.

The program checks to see if it is running on a Model I or a Model III and sets itself accordingly. It has a built-in machine-language case reversal and a lowercase video driver for the Model I. If your Model I does not have a lowercase keyboard modification, or if your Model I DOS has its own case reversal and lowercase video driver, check line 9 of the program for instructions before you run the program.

To use the program, load it and, if necessary, set the machine-language routine in lines 0–9 for your use. Typing RUN POKEs this routine and deletes lines 0–9. You can save a backup copy only before you run the program. Entering RUN a second time starts the program. A title appears with the prompt, Command? Respond by pressing one of the 18 single-letter commands shown in Table 1.

Before you enter any text, only A, F, H, L, and X are acceptable commands. The others do nothing or give an ENTRY ERROR message. After leaving a command, you return to the Command? prompt. Except as noted, all references to pressing a keyboard letter mean an unshifted (lowercase) letter.

Description of the Commands

● *Add*—This command adds material to the end of a current text file or, if the file is empty, starts a new text file. A flashing block cursor shows the place at which text will be added. This command turns on the line number display option. You can type material continuously without pressing ENTER for each line. The cursor position appears as a number at the bottom of the screen. When the file is full, a FILE FULL message is given. The capacity is 408 text lines, from line 0 through line 407. To reserve a spacer line without text, you must enter at least one space. The program eliminates trailing spaces from all text lines. Subcommands in the Add mode are shown in Table 2.

A	Add	Adds text to existing file or starts a new file
B	Blank	Removes blank text lines and renumbers lines
C	Compile	Moves words between lines to get the best fit
D	Delete	Deletes a block of text lines and renumbers
E	Edit	Edits a text line with subcommands similar to BASIC commands
F	Format	Changes formats for text display or printing
H	Help	Lists all commands and tells you how to exit each mode
I	Insert	Inserts text line(s) into the middle of a text file
J	Justify	Right-justifies text lines by spacing words
K	Kill	Removes all text, resets format, and starts over
L	Load	Loads previously saved text from tape or disk
M	Move	Moves a block of lines to a new place in the text file
O	Overlay	Searches for any word in text or replaces it
P	Print	Prints a text file on the printer
R	Replace	Replaces one existing text line with another
S	Save	Saves a text file on tape or disk
V	Video	Displays a text file on the video display unit
X	Exit	Exits from the program, resets string space, and so on

Table 1. *Program commands*

● *Blank*—This command eliminates blank lines from the text file you are working on and renumbers the lines. Blank lines are empty lines; this command does not affect lines containing only a space. During operation, the screen displays *Deleting Blank Lines.* When it has finished the Video command displays the text file.

● *Compile*—After Editing lines or changing line length with Format, some lines may be too long or too short to fit properly into the specified line length. The Compile command shifts words between the lines of a selected block to get the best possible fit. To abort (return to the Command? prompt without any action on the text file), press ENTER in place of line numbers. To Compile to the end of the text, enter a large number as the *Last Line to Compile.*

The Compile command works in three phases. It first spaces all words normally, inserting three spaces after a period, question mark, exclamation point, or colon, two spaces after a semicolon, and one space otherwise. Then it checks the line length and pushes any extra words in a line onto the following line. Finally, it checks if a line can accept words from the line after it and, if so, pulls words up to the preceding line. Compile can push words forward any number of lines but can pull words back only one line. If a blank line occurs between lines that have been Compiled, use the Blank command to remove it. Then Compile again, or the words may be left on the wrong line.

You should use the Compile command on one paragraph at a time, as it

Space bar	Moves the cursor one position to the right and adds one space
→	Moves cursor continuously to the right up to the length of the line, adding spaces to the end of the line
←	Moves the cursor continuously to the left, erasing characters to the beginning of the line
SHIFT ←	Erases the entire current line of text
<	Overstrike. Moves cursor one position to the left, over the previously entered character, followed by a < symbol. Then you must key in the overstrike character. A back spacing printer is needed to use overstrike. Overstrike puts two invisible characters in the line, which can cause difficulty in the Edit and Justify modes.
@	Caps lock. Pressing @ turns this function on or off. Only letters are affected. CAPS LOCK appears at the lower right of the screen when the function is on.
ENTER	Ends the current line and goes to the next line before the automatic end-of-line action
CLEAR	Ends the line and holds it secure from the Justify command. A ← marker appears at the end of the line.
SHIFT ↑	Ends the line, centers current line of text, and holds it secure from Justify. It leaves a ← marker at the end of the line.
SHIFT →	Ends the line and moves the current text to the extreme right end of the line
↓	Ends the line and inserts a blank line between lines of text by adding a line feed to the end of the line.
SHIFT ↓	Ends the line and leaves an end-of-page marker (↓) at the end of the line. (SHIFT ↓ Z for a Model III or a late Model I.)
SHIFT @	Escapes from the Add mode and returns to the Comand? prompt. (SHIFT @ usually must be keyed in twice.)

Table 2. *Subcommands in the Add mode*

left-justifies all lines except the first one within its range. It also buries any end-of-page, hold-justify, or linefeed markers that are not on the last line within its range. If buried, these markers do not work properly. During operation, *Compiling Line n m is displayed,* where n and m are first/second and third phase line numbers. When done, if 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 (or lines) and then Compile just the line that is too long and the blank line or lines. After a satisfactory Compile operation, the file is displayed by Video, starting with the first Compiled line.

● *Delete*—This command eliminates a specified block of lines. If you wish to eliminate only one line, enter that line number as both the first and the last line number. To Delete to the end of the text, enter a large number in answer to the *Last Line to Delete* prompt. During operation, the screen displays *Deleting* and *Deleting Blank Lines* messages. The modified text is

displayed by Video, starting with line 0. To abort, press ENTER in place of the line number.
● *Edit*—To edit a line, type the line number, then press E (defaults to line 0). If you give a nonexistent or empty line, you will get an ENTRY ERROR message. The entire line is visible in Edit, even the character above the smaller Edit cursor. The line number option is turned on by Edit, and the cursor position is shown at the screen bottom. Subcommands in the Edit mode are shown in Table 3. Variable n always defaults to one.

To see text lines that contain nonprinting characters, such as those inserted by overstrike, font changes, or U mode underlining, sweep the cursor over the line from right to left in the Edit command. Nonprinting characters show up as fixed cursor blocks, except for ASCII 14, which appears as an underlining character.

If you enter nonprinting characters by mistake with Edit, you can remove them using Delete or Again. After you exit from Edit, if the line is too long, the *Line n has x Characters* message appears. If the line is equal to or shorter than the specified line length, Video displays the text lines, starting with the

Space bar	Moves cursor one position to the right (no space)
→	Moves cursor continuously to the right (no spaces added)
←	Moves cursor continuously to the left (without deleting)
A	Again. Cancels previous editing changes and reenters Edit. The List command makes all editing changes permanent.
n C	Changes next n characters to next n keyed characters then returns cursor to start (as completed signal) Note that @ and <, which are control characters you cannot enter in the Add mode, can be entered using Change.
n D	Deletes the next n characters and closes up the line
E	Epson underlining for Epson MX-80 printers. Put the cursor under the first character to be underlined and press E for each character. Only one group per line can be underlined, not including double-width characters. After Editing a line, use List before using the E underline. Later, an underline, two numbers, and a second underline are shown at the end of the line. The first number is the space before the underline starts, and the second is the number of characters to be underlined.
H	Hacks the rest of the line and enters the Insert mode
I	Inserts characters into a line at the current cursor position and moves the following characters to the right. While in the Insert mode, you can move the cursor left or right without changing the line by using the left arrow or right arrow. You can use the < to insert overstrikes in Edit. See the "Font Change" section for printer font changes. See the "TRS-80 Graphics Printing" section for information on inserting graphics characters.

n K c	The Kill command. It deletes all characters from the current cursor position to the nth time that character c occurs.
L	Lists the line and returns the cursor to the start of the line. List makes editing changes immune to the Again command.
n S c	Searches for the nth occurence of character c. Keeps uppercase and lowercase separate (even with uppercase display).
U	Underlining for backspacing printers. Used like E but it is not restricted. Later, it shows only the underlines, not the underlined characters.
X	Extends a line. Enter the Insert mode at the end of the line.
SHIFT @	Leaves the C, H, I, or X modes and returns to Edit. SHIFT @ usually must be pressed twice.
ENTER	Exits from the Edit command (including exit from the C, H, I, and X subcommands).

The following five subcommands are not performed if entered in the C, H, I, or X subcommands:

CLEAR	Holds a line secure from Justify (adds a left-arrow)
SHIFT ↑	Centers text line and holds it secure from Justify
SHIFT →	Moves a line's text to the extreme right of a line
SHIFT ↓(Z)	Adds an end-of-page marker (↓) to the end of a text line
↓	Adds a line feed (ASCII 10) to the end of a line and exits from the Edit command

Table 3. *Subcommands in the Edit mode*

line you just Edited. Edit also deletes extra trailing spaces.

● *Format*—This command resets the 14 variables that control the display or printing of the text file. Each variable has a default value which is shown first. If the default value is correct, press ENTER. The 14 variables are as follows.

1) Line length: 60 characters, to fit with the line number on one video display line. Limits: 20–122. Long lines may overwrite the cursor position number.
2) Line spaces: None, for no extra spaces between lines. Enter the number of blank lines between text lines.
3) Line numbers: Enter y for yes to show numbers for lines. Enter n (no) to delete line numbers.
4) First print line: 0, to start printing from the initial line of text. To start printing at a later line, enter the corresponding line number. Limits: 0 to the last line in text file.
5) Last print line: Last line in file, to print to the end of the file. To end printing at an earlier line, enter the line number. Limits: First print line (set above) to the last line in the file.
6) Left margin: 10, to print the default 60-character line centered on an

80-character per line printer. This variable affects only the printer.
7) Page length: 15, to fill the video display. The number of lines per printed page is usually between 56 and 58. You must reset this value to print. Lines containing spaces are counted, but not line feeds (from item 2 above, or from the use of the down arrow).
8) Page spacing: 8, to use with a page length of 58 lines for a 66-line page. If page numbers are to be printed, use 6 for page spacing or 56 for page length.
9) Page numbers: n, for no page numbers. To show page numbers, enter y. Note that you must show a page number to show page heading. (See item 14.)
10) First page: 1, to start numbering pages with page 1. If you wish, you may enter a later initial page number.
11) Page 1 number: n, not to show a page number for page 1. To show page numbers for all pages enter y only if item 9 is also set to y.
12) Page stop: n, to continue printing after each page. For printing on single pages, enter y, and you will have time to insert a new page.
13) MX-80 graphics: n, if you do not want to shift graphics characters sent to the printer up by 32 to match the MX-80 printer's graphics codes. Enter y to print graphics on an MX-80. E mode underlining cannot be used.
14) Heading: " ", or null string. If you want to show a heading at the top left of each page, you must enter it. The heading is shown only if the page numbers are shown (items 9 and 11 above).

After going through these 14 Format variables, you return to the command mode. If there is nothing in the file, some variables may be skipped. Out-of-range entries generate an *Entry must be - - -* message so you can try again.

● *Help*—This command displays all legal commands and their definitions to refresh your memory. It also tells you how to return to the command mode for those commands that do not return automatically.

● *Insert*—This command inserts a line (or lines) of text into the middle of the current file, using the Add command. Key in the line number before which you want to insert lines, then press I (defaults to line 0). The following lines are moved down and renumbered. If you give a nonexistent line number, you get an ENTRY ERROR message. To insert empty lines (for Compile), press ENTER. At any time, to keep text you have entered and return to command mode, press SHIFT @ twice. If the text file gets full, you receive a FILE FULL message.

● *Justify*—This command right justifies all lines of the current file. The only exceptions are lines with a hold-justify, an end-of-page, or a linefeed marker at the end, a line without any spaces between words, or a line already longer than the specified line length (as you set in Format).

Extra spaces are inserted between words, starting at a random position, but evenly distributed. Spaces may be inserted between adjacent words or only every other word, depending on whether there is an odd or even

number of words in the line. Leading spaces are not affected to maintain indentation. You should use Justify before underlining, changing fonts, or overstriking, as these operations insert nonprinting characters into the text. During operation, *Justifying Line n* is displayed, where n is the line the program is working on. When it is done, the text is shown by Video.
● *Kill*—This command removes all text from the file and resets Formats to their default values, leaving the program ready to start over. This command asks again if you really want to Kill, to prevent accidental loss of the text file.
● *Load*—This command loads a saved text file from tape or disk. For tape operation it shows the message *Get cassette ready, press ENTER*. For disk, it shows the current filespec, if any, and asks if you want a different one. To abort, press SHIFT @ twice for tape, or SHIFT @ ENTER for disk.

For disk operation, to leave the filespec unchanged, press ENTER. The program loads the filespec you specified and displays *Loading (Filespec)*. For tape, *Loading (Heading)* appears. If there is already text in memory, Load appends the new text onto the old text. If there is not room for both, *Text too long* is shown, and no text is Loaded. In either case, the Format variables become those of the new text.
● *Move*—This command transfers a specified block of lines either forward or backward in the text file. A place to insert the block of lines is opened up automatically and the place where the block of lines came from is closed up automatically. The lines are then renumbered. Move cannot be used after a FILE FULL message. Only one line at a time is Moved; so large blocks can be handled even when the file is nearly full. While operating, Move shows *Moving, Deleting Blank Lines*; when done, it uses Video. To abort, press ENTER in place of line numbers.
● *Overlay*—This command gives a global search or replace function. You are asked the question, *Search or Replace (S/R)?* The search mode looks through the text file for any word you specify. If it finds the word, it enters the Edit mode, placing the cursor under the word's first letter. When you exit from Edit, you may continue the search from the current cursor position or return to the Command? prompt. If the word is not in the file, *Word Not Found* is displayed.

The Replace mode replaces any old word in the file with a new word. This is useful in correcting a misspelling, changing a name, and so on. The new word may be longer or shorter than the old word, but the same spacing of words is maintained. *Overlaying* is shown while operating; the file is displayed by Video when done. Either mode may find a word that is part of a longer word. To avoid this, put a space before and/or after the word. This may miss a word followed by punctuation. A word also may be missed if letters are capitalized.
● *Print*—This command prints the text file on the printer. Remember to

reset the Format variables before printing. To avoid a Function Call error, the program removes blank lines using the Blank command before printing. During operation, first *Deleting Blank Lines*, then *Printing* are displayed. If you set the Page Stop variable to y, the message, *Get new page ready, press ENTER* is displayed after each page is printed. If the program receives a down-arrow end-of-page character (ASCII 2), it inserts blank lines to fill out the page before printing the next page.

● *Save*—This command records the text file on tape or disk. For tape, it shows *Get cassette ready, press ENTER*. For disk, it shows the current filespec and asks if you want to use a different one. To abort, press SHIFT @ twice for tape or SHIFT @ ENTER for disk operation.

For disk operation, any legal filespec may be used. To leave a filespec unchanged, press ENTER. To save the file on a specified disk, end the filespec with :d where d is the desired disk drive number. To avoid possible disk errors, blank lines are deleted. During operation, first *Deleting Blank Lines*, then *Saving (Filespec)* are shown. For tape operation, *Saving (Heading)* is shown.

● *Video*—This command displays the text file on the video display. To start the display at a specific line, key in the line number before you press V (defaults to line 0). If the line length is greater than 60 characters (64 if *Line Nos.* was set n in the Format mode), the lines wrap around to the next display line.

After displaying each page, the program halts. To see the next page, press ENTER. To scroll text forward, press the down arrow; the up arrow to go backward. To return to the Command mode, press any letter. If you have added any down-arrow (linefeed) characters, or have wraparound lines, the top lines of the page may scroll off the top of the screen. To avoid this, reset the page length in the Format mode.

Video may show one of three non-text characters at the end of a line: A left arrow for hold-justify, a down arrow for end-of-page, or an underline after the space in a spacer line. These markers help to specify the text's format. In addition, a blank line with no line number follows any line that ends with a linefeed marker or between all lines if *line spaces* are specified in Format.

● *Exit*—This command allows 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. The program again asks if you really want to exit from the program to avoid accidental loss of the text file.

Font Changes

Some printers print characters of different fonts. For printers which require imbedded control characters in the text to set different fonts, this program can change fonts in the middle of the text. To do this, in the Insert subcommand of the Edit command, press SHIFT down arrow(letter), where letters A through Z insert ASCII codes 1 through 26 into the text. SHIFT up arrow gives the Escape code, ASCII 27. Since inserting invisible characters confuses the display, first move the cursor to the right end of the line, then work towards the beginning of the line if you have more than one code to insert. The Escape code is an upwards line feed on the video display. This eliminates some portions of the display; so you must work without seeing the whole line.

Since the Hack and Extend subcommands of Edit use the Insert subcommand, they too may be used to change fonts. Some printers, such as the Centronics 737 (Radio Shack Line Printer IV), consider underlining as a font change. For these, insert the appropriate codes for a font change.

The actual codes to use for different font changes for the Epson MX-80 printer are given in Table 4. Except for double-width characters, all font changes for the MX-80 are for whole lines only. Thus the turn on and the turn off codes must be in different lines so they do not cancel each other. You can combine these four fonts (except for Compressed and Emphasized) to get a total of 12 fonts.

	Turn On	Turn Off
Emphasized	Shift ↑ E	Shift ↑ F
Double strike	Shift ↑ G	Shift ↑ H
Compressed	Shift ↓ O	Shift ↓ R
Double width	Shift ↓ N	Shift ↓ T

Table 4. *Font control codes for the Epson MX-80 printer. Insert these codes into the text with Edit to change print fonts. See "Font Changes" section for details. Use uppercase letters only.*

The font change codes for the MX-80 with the addition of *Graftrax-80* are given in Table 5. With this addition, you can change fonts for any portion of a line, change line spacing within the text, and use italics. Counting italics as a font, 24 different fonts can be set.

Note that for SHIFT down arrow(letter), the SHIFT down arrow acts as a Control key and must be held down while the letter is keyed in. The same is true for the SHIFT up arrow(letter). In these cases only, the letter being keyed in is an uppercase letter. Where SHIFT up arrow is followed by a number or some other non-letter character, release the SHIFT up arrow before you key in the final character.

With the Graftrax-80 addition to the MX-80, the E mode underlining does not work. (U underlining does work.) The Print routine is easily

	Turn On	Turn Off
Line spacing 8 lines/inch	Shift ↑ 0	———
Line spacing 7/72 inch	Shift ↑ 1	———
Line spacing 6 lines/inch	Shift ↑ 2	———
Italics	Shift ↑ 4	Shift ↑ 5
Emphasized	Shift ↑ E	Shift ↑ F
Double strike	Shift ↑ G	Shift ↑ H
Compressed	Shift ↑ P	Shift ↑ Q
	Shift ↓ O	Shift ↓ R
Double width	Shift ↑ S	Shift ↑ T
	Shift ↓ N	Shift ↓ T
TRS-80 graphics (without setting Format)	Shift ↑ :	Shift ↑ ;

Table 5. *Font control codes for the Epson MX-80 printer with the Graftrax-80 addition. Insert as above. Note that there are sometimes two ways to obtain the same font change. Use upper-case letters.*

changed so that E underlining will work (but then it won't work on a normal MX-80). To change for E mode underlining with the Graftrax-80:

Line 2530, Change CHR$(133) to CHR$(5) and delete the final E$;"2"; from the line.
Line 2550, Delete E$;"A";CHR$(140); from the line.

When you use a Graftrax-80 addition to the MX-80, this change gives you the advantage of using both TRS-80 graphics and E mode underlining in the same text. For other printers, check the user's manual to find out what fonts can be used and which control codes are needed. This program should be able to insert the required codes.

TRS-80 Graphics Printing

This program allows direct keyboard entry of the 64 TRS-80 graphics characters, plus the [,/,],Λ,__,‘,{,¦,}, and ~ special characters. The graphics characters can be printed by the MX-80, Okidata, and some other printers. Most printers should be able to print the special characters. The MX-80 requires that the graphics characters be shifted up by 32 (ASCII 130 to ASCII 162, etc.). You can set this in the Format command, but as mentioned above, E mode underlining is excluded. With the Graftrax-80 addition to the MX-80, you can also follow the font changes in Table 5 to print TRS-80 graphics.

To enter either special characters or graphics, use the Edit command. In the Hack, Insert, or Extend modes, press SHIFT left arrow for a low graphics entry or SHIFT right arrow for a high graphics entry. The mode is displayed at the lower right of the video screen. Repeated pressing of either SHIFT left arrow or SHIFT right arrow toggles the graphics entry mode on and off. Pressing the keys indicated in Table 6 (unshifted) inserts the special characters or graphics characters into the text. The graphics characters appear normal on the screen, but print much wider than they are displayed unless you use the compressed print font. Special characters may look different on the screen than when they are printed. On the Model I, [,/,], and Λ are displayed as ↑,↓,←, and →.

Low Graphics						High Graphics					
Key	Chr	ASCII	Key	Chr	ASCII	Key	Chr	ASCII	Key	Chr	ASCII
Ø	[	91	H	🬆	135	Ø	🬘	154	H	🬪	173
1	\	92	I	🬇	136	1	🬙	155	I	🬫	174
2	]	93	J	🬈	137	2	🬚	156	J	🬬	175
3	^	94	K	🬉	138	3	🬛	157	K	🬭	176
4	_	95	L	🬊	139	4	🬜	158	L	🬮	177
5	`	96	M	🬋	14Ø	5	🬝	159	M	🬯	178
6	{	123	N	🬌	141	6	🬞	16Ø	N	🬰	179
7	¦	124	O	🬍	142	7	🬟	161	O	🬱	18Ø
8	}	125	P	🬎	143	8	🬠	162	P	🬲	181
9	~	126	Q	🬏	144	9	🬡	163	Q	🬳	182
:	nul	127	R	🬐	145	:	🬢	164	R	🬴	183
;	;	59	S	🬑	146	;	🬣	165	S	🬵	184
A	spc	128	T	🬒	147	A	🬤	166	T	🬶	185
B	🬀	129	U	🬓	148	B	🬥	167	U	🬷	186
C	🬁	13Ø	V	▌	149	C	🬦	168	V	🬸	187
D	🬂	131	W	🬔	15Ø	D	🬧	169	W	🬹	188
E	🬃	132	X	🬕	151	E	▐	17Ø	X	🬺	189
F	🬄	133	Y	🬖	152	F	🬨	171	Y	🬻	19Ø
G	🬅	134	Z	🬗	153	G	🬩	172	Z	█	191

TRS-8Ø graphics should be compressed for proper proportions:

Table 6. *Keyboard entry of TRS-80 graphics and special characters into text. Insert these codes into the text with Edit. See the "TRS-80 Graphics Printing" section. Do not use uppercase letters.*

Possible Problems

A program halt accompanied by a BASIC error message or one caused by accidentally touching the BREAK key usually does not mean that you have lost the text file. In most cases, you can recover it by typing GOTO 60 and pressing ENTER. This returns you to the Command? prompt.

Speed

Since this program is written in BASIC, its handling of each character is not fast enough to keep up with a good touch typist. A touch typist must key in text at a deliberate pace. This is especially important at the end of a line because moving a word to the beginning of the next line takes a little extra time.

The program occasionally pauses during operation due to the way BASIC handles strings. Each time a string is changed, it is assigned a new location in string space. As this quickly fills up all available string space, a garbage collection routine in BASIC must clear out all of the old versions of each string. As the text file fills up, these pauses become longer and more frequent. The best solution to this problem is to save the text on tape or disk as separate short files of about 150 lines, rather than trying to fill the text file to its maximum capacity. Use Load to combine the short text files for printing.

Tape saving or loading of text is relatively slow. This is because of the inherent slowness of the TRS-80 tape operations, plus having to "translate" the text to avoid the improper operation that some punctuation marks would cause.

The Archbold clock control board that I use speeds up the TRS-80's clock by 50 percent. The OUT254,1 statement in the program speeds up the clock, while the OUT254,0 statement slows it down again (for disk use, etc). If you do not have this board, these statements have no effect.

Model III TRS-80s

The program automatically sets itself to allow for the lowercase display capability of the Model III and allows for its slightly lower memory availability. The arrows for video display markers listed as ↑,↓,←, and → for the Model I appear as [,/,], and Λ on the Model III.

Printer

The routines the program uses for printing text work correctly on my Epson MX-80 printer, but may have to be changed for some other printers. If the printer is not ready, the program does not hang up, but gives a *Printer Not Ready* message and gives you a chance to get it ready. There is an underlining routine for MX-80 E mode underlining that works even with the standard Radio Shack printer cable. Some printers may require LPRINT-CHR$(32) instead of LPRINT at the end of line 2450. There are many variations; the printer manual should tell you how to set the program. Either TRS-80 graphics or E mode underlining may be printed on the MX-80, but not both in the same text. The special characters, [,/,],Λ,_,‘,{, ¦,}, and ~, print either way.

Memory

To avoid an *Out of Memory* error, key in the program listing without the extra spaces added for legibility. The program requires 12829 bytes of memory before you run it. This program is designed to use essentially all of the memory of the TRS-80. The six machine-language routines are POKEd into high memory. If you use a machine-language printer driver, etc., you have to relocate it and reduce the size of the program's text file to avoid getting the *Out of Memory* message. If you run the program and see Command? immediately, then press BREAK and PRINT MEM, you should have at least 350 bytes of free memory. If not, you have to reduce the CLEAR in line 30 and the value of NL, where NL equals the number of lines. CLEAR should be set at least 300–400 bytes greater than the value of NL*LL, where LL is the line length.

Line Length

If you set the line length in Format equal to the maximum print line length of your printer, some printers insert blank lines between lines of text. The easiest way to avoid this is to set the line length shorter, for example to 79 for an 80-character per line printer. Be sure to set the left margin in Format to 0 also.

For saving lines of text on tape, if the lines are set to longer than 60 characters, it is necessary to modify the tape SAVE and LOAD routines. These routines handle four lines (240 characters) at a time to save time and tape. 240 is almost the maximum number the tape PRINT and INPUT statements can handle at once. To handle 80-character lines, you have to use only three lines at a time instead of four. Both the SAVE and LOAD routines need to be changed (STEP4 to STEP3, delete "X$(3)", and 0TO3 to 0TO2. See lines 2070, 2080, 2740, 2750, and 2770).

Lowercase Modifications

There are a number of different keyboard modifications available for the Model I TRS-80 that allow the display of lowercase letters on the screen. This program has a built-in lowercase driver. If you do not have a lowercase modification, delete the last two POKEs in line 5 of the program. If you use a lowercase modification or DOS that contains its own driver, delete the part of line 5 following NEXT I. See line 9 of the program for instructions.

If you do not have a lowercase modification on your Model I TRS-80, all letters appear as uppercase letters on the video display. Both uppercase and lowercase letters print; you can use the Search subcommand of Edit to check the case.

With a lowercase modification on a Model I, or with a Model III, the filespec in disk LOAD and SAVE appears in lowercase letters unless you shift the letters; this does no harm. The filespecs are interpreted as uppercase by the DOS.

Program Listing. *BASIC Word Processor*

```
 0 CLS :
   PRINT @ 320, CHR$(23); "Poke Scroll, LC Patch & Shift" :
   '  M/L Graphics Shift by Leo Christopherson, 80-US, Jan-F 1980
 1 :
   '  M/L Scroll-Down by Bob Boothe, 80-Micro, April 1981, p. 116
 2 :
   '  M/L Video Patch by Tim Mann, TRS-80 Computing, V1, N2 (CIE)
 3 :
   '  M/L Case Shift by Martin Hambel, 80-Micro, May 1981, p. 260
 4 POKE 16561,149 :
   POKE 16562,255 :
   POKE 16409,0 :
   CLEAR 50 :
   DEF USR0 = &HFFDC
 5 RESTORE :
   FOR I = - 106 TO - 1 :
    READ B :
    POKE I,B :
    NEXT I :
   IF PEEK(84) = 1 POKE - 23, PEEK(16406) :
   POKE - 22, PEEK(16407) :
   X = USR0(B) :
   POKE 16414,190 :
   POKE 16415,255 :
   '  POKE in six M/L routines,   Divert to Keyboard Case Shift and
    Video Patch (if Model I).
 6 POKE - 95, PEEK(16422) :
   POKE - 94, PEEK(16423) :
   DEF USR0 = &HFFA3
 7 DATA 245,121,254,128,56,2,198,32,79,241,195,0,0, 217,17,255, 63,
   33,191,63,1,192,3,237,184,33,0,60,17,1,60,1,63,0,54,32, 237,176,
   217,201, 221,110,3,221,102,4,218,154,4,221,126,5, 183,40,1,119,1
   21,254,32,218,6,5,254,128,210,166,4,195,125,4
 8 DATA 33,227,255,34,22,64,201, 225,33,235,255,229,195,0,0, 254,65
   ,56,14,254,123,48,10,254,91,56,4,254,97,56,2,238,32, 195,221,3
 9 CLS :
   PRINT @ 520, CHR$(23); "Key in:      RUN   <ENTER>" :
   DELETE 0 - 9 :
   '  For Model I TRS-80s without an LC keyboard mod, put a ' in
   line 5 before the ":POKE 16414,190 :POKE 16415,255".   For TRS-8
   0s with an LC DOS, put ' in line 5 after "NEXT I".
10 CLS :
   PRINT TAB(10)"BASIC Word Processor, 48K Tape/Disk Version
20 :
   '  (C) by D.D.Hinrichs  1981
30 CLEAR 24880:
   DEFINT A - Z:
   U = 32:
   V = 64:
   W = 992:
   CMD "T":
   OUT 254,1
40 NL = 408:
   DIM A$(NL),X$(3),S(U),T(U)
50 B$ = CHR$(30):
   E$ = CHR$(27):
   F$ = "### ":
   M$ = "n":
   N$ = "y":
   P$ = M$:
   PN$ = M$ :
   P1$ = M$:
   S$ = " ":
   U$ = CHR$(95):
   FP = 1:
   LA = - 1:
```

Program continued

```
    LL = 60:
    LM = 10:
    PL = 15:
    PS = 8
 60 CLOSE :
    CL = 0:
    H = 1:
    I = LA:
    IT = 0:
    N = 0:
    R = 0:
    PRINT :
    PRINT "Command? ";
 70 GOSUB 3160:
    PRINT :
    IF A > 96 ON A - 96 GOTO 90,570,610,890,940,1490,80, 1780,1820,1
    860,1980,2000,2140,80,2210,2380,80,2620,2650,80,80, 2870,80,3200
 80 PRINT "** ENTRY ERROR **":
    GOTO 60
 90 CLS :
    D = 0:
    P = 1:
    N$ = "y":
    C$ = CHR$(143):
    IF I < 0 GOTO 130 :
    '  ADD
100 IF NL = LA + 1
     THEN
      280:
     ELSE
      IF I > 13
       THEN
        B = I - 13:
       ELSE
        B = 0
110 FOR L = B TO I:
     GOSUB 3040:
     D = D + (Y + 3) / V:
     IF A = 10
      THEN
       D = D + 1
120  NEXT L
130 L = I + 1:
    B = 0:
    IF L + D > 14
     THEN
      D = 14 - L
140 IF P > 61
     THEN
      D = D + 1:
     ELSE
      IF P < 60 OR P < 62 AND B
       THEN
        150:
       ELSE
        IF H GOTO 160
150 PRINT
160 GOSUB 530
170 PRINT @C,B$;:
    GOSUB 3040:
    P = Y + 1:
    C = C + P + 3:
    K = L + 1:
    H = 1:
    GOSUB 550
180 PRINT @W,P;
190 PRINT @C,C$;:
    A$ = INKEY$:
    PRINT @C,S$;:
    IF A$ = "" GOTO 190
```

```
200 B = 0:
    A = ASC(A$):
    IF CL IF A > 96
     THEN
      A = A - U:
      A$ = CHR$(A)
210 IF A > V
     THEN
      260:
     ELSE
      IF A > = U GOTO 250
220 A$ = S$:
    IF A > 7 ON A - 7 GOTO 400,440,470,80,80,280
230 IF A > 23 ON A - 23 GOTO 420,460,380,390,80,80,80,480
240 GOTO 80
250 IF A = 60 GOTO 490:
     ELSE
      IF A = V GOTO 360
260 IF A = 96
     THEN
      LP = LA:
      IF LA < L
       THEN
        LA = L:
        LP = L:
        GOTO 60:
       ELSE
        60
270 PRINT @C,A$;:
    A$(L) = A$(L) + A$:
    IF P < = LL
     THEN
      P = P + 1:
      C = C + 1:
      GOTO 180
280 IF R GOTO 60:
     ELSE
      IF NL < = K PRINT "FILE FULL":
      LA = NL - 1:
      LP = LA:
      GOTO 60
290 IF LEN(A$(K))
     THEN
      L = K:
      GOSUB 1830
300 IF K > LA
     THEN
      LA = K
310 IF A$ = S$ GOTO 350
320 FOR M = LL + 1 TO 2 STEP - 1:
     A$ = MID$(A$(L),M,1):
     IF A$ < > S$ NEXT M:
    GOTO 350
330 A$(K) = RIGHT$(A$(L),LL - M + 1):
    A$(L) = LEFT$(A$(L),M - 1)
340 PRINT @C - LL + M - 1,B$;:
    L = K:
    B = 1:
    GOTO 140
350 PRINT B$;:
    A$(L) = LEFT$(A$(L),LL):
    L = K:
    GOTO 140
360 IF CL
     THEN
      CL = 0:
     ELSE
      CL = 1 :
      '  C-L
370 GOSUB 550:
```

Program continued

```
    GOTO 180
380 IF P > LL GOTO 280:
     ELSE
      GOSUB 1460:
      GOTO 520 :
      '   S-D
390 IF P > LL GOTO 280:
     ELSE
      GOSUB 1440:
      GOTO 520 :
      '   S-U
400 IF P = 1 GOTO 180:
     ELSE
      C = C - 1:
      P = P - 1:
      A$(L) = LEFT$(A$(L),P - 1) :
      '   L
410 PRINT @C,C$;S$;:
    GOSUB 510:
    IF PEEK(14400) = U
     THEN
      400:
     ELSE
      180
420 IF P = 1 GOTO 180 :
    '   S-L
430 A$(L) = "":
    H = 0:
    PRINT @960,B$;:
    GOTO 160
440 IF P > = LL GOTO 180:
     ELSE
      PRINT @C,S$;C$;:
      C = C + 1:
      P = P + 1 :
      '   R
450 A$(L) = A$(L) + S$:
    GOSUB 510:
    IF PEEK(14400) = V
     THEN
      440:
     ELSE
      180
460 IF P > LL GOTO 280:
     ELSE
      GOSUB 1480:
      GOTO 520 :
      '   S-R
470 IF P > LL GOTO 280:
     ELSE
      GOSUB 1470:
      D = D + 1:
      R = 0:
      PRINT :
      GOTO 280 :
      '   D-A
480 IF P > LL GOTO 280:
     ELSE
      GOSUB 1450:
      GOTO 520 :
      '   CL
490 IF P = 1 GOTO 180 :
    '   <
500 PRINT @C,"<";:
    A$(L) = A$(L) + CHR$(8):
    C = C - 1:
    P = P + 1:
    GOTO 180
510 PRINT @W, CHR$(15);P;:
    FOR I = 1 TO 10:
     NEXT I:
    RETURN
520 H = 0:
    GOSUB 530:
    PRINT @C,B$;:
    GOSUB 3040:
    GOTO 280
530 C = (L + D) * V:
    IF C > 896
     THEN
      C = 896:
      IF H PRINT
540 RETURN
550 PRINT @1014,;:
    IF CL = 1 PRINT "CAPS-LOCK";:
     ELSE
      PRINT STRING$(9,U);
560 RETURN
570 IF LA < 0 GOTO 80 :
    '  BLANK
580 CLS :
    PRINT "Deleting Blank Lines":
    FOR J = LA TO 0 STEP - 1
590  IF A$(J) = "" FOR I = J TO LA:
      A$(I) = A$(I + 1):
      NEXT I:
     A$(LA) = "":
     LA = LA - 1
600  NEXT J:
    IF R
     THEN
      RETURN :
     ELSE
      2870
610 F = 0:
    INPUT "First Line to Compile";F:
    IF F < 0
     THEN
      F = 0 :
      '  COMPILE
620 Z = 0:
    INPUT "Last Line to Compile";Z:
    IF Z > LA
     THEN
      Z = LA
630 IF F > = Z
     THEN
      80:
     ELSE
      CLS :
      PRINT "Compiling Line":
      J = 1
640 FOR L = F TO Z:
     PRINT @15,L:
     GOSUB 3090:
     IF Y < 2 GOTO 710
650  X$ = "":
     P = 1:
     K = L + 1
660  Q = INSTR (P,A$(L),S$):
     IF Q > P
      THEN
       J = 0:
      ELSE
       IF Q = 0
        THEN
         Q = Y + 1
670  IF J = 0 IF P = Q
      THEN
       P = Q + 1:
```

Program continued

```
      GOTO 660
680  X$ = X$ + MID$(A$(L),P,Q - P + J):
     P = Q + 1
690  IF Q < Y GOSUB 860:
     IF A = U GOTO 660:
      ELSE
       X$ = X$ + S$:
       GOTO 660
700  A$(L) = X$
710  GOSUB 3090:
     IF Y < = LL OR L = Z GOTO 770
720  X$ = "":
     FOR I = Y TO 1 STEP - 1:
      A$ = MID$(A$(L),I,1)
730   IF A$ < > S$
       THEN
        X$ = A$ + X$:
        NEXT I:
       GOTO 770
740  GOSUB 860
750  A$(L) = LEFT$(A$(L),I - 1):
     IF LEN(A$(K)) = 0
      THEN
       A$(K) = X$:
       GOTO 710
760  A$(K) = X$ + S$ + A$(K):
     GOTO 710
770  NEXT L:
    FOR L = F TO Z - 1:
     K = L + 1:
     X$ = A$(L):
     PRINT @20,L
780  X = LEN(A$(K)):
     Y = LEN(X$):
     IF X * Y = 0 GOTO 850
790  IF X > 1 IF ASC(A$(K)) = U
      THEN
       A$(K) = RIGHT$(A$(K),X - 1):
       GOTO 780
800  GOSUB 860:
     Y = Y + R
810  Q = INSTR (A$(K),S$):
     IF Q
      THEN
       Y$ = LEFT$(A$(K),Q - 1):
       GOTO 830
820  Y$ = A$(K):
     Q = X + 1
830  IF LL - Y < Q GOTO 850:
      ELSE
       X = X - Q:
       IF X < 0
        THEN
         X = 0
840  X$ = X$ + S$ + Y$:
     A$(K) = RIGHT$(A$(K),X):
     GOTO 780
850  A$(L) = X$:
     NEXT L:
    X = LEN(A$(Z)):
    N = F:
    GOTO 1080
860 A = ASC( RIGHT$(X$,1)):
    R = 0:
    IF A = 59
     THEN
      X$ = X$ + S$:
      R = 1
870 IF A = 33 OR A = 46 OR A = 58 OR A = 63
     THEN
```

```
      X$ = X$ + "  ":
      R = 2
 880 RETURN
 890 F = 1:
     Z = 0:
     INPUT "First Line to Delete";F:
     IF F < 0
      THEN
       F = 0 :
       '  DELETE
 900 INPUT "Last Line to Delete";Z:
     IF F > Z
      THEN
       80:
      ELSE
       IF Z > LA
        THEN
         Z = LA
 910 CLS :
     PRINT "Deleting":
     J = Z
 920 FOR I = F TO LA:
      J = J + 1:
      IF J > LA
       THEN
        A$(I) = "":
       ELSE
        A$(I) = A$(J)
 930  NEXT I:
     LA = LA - Z + F - 1:
     R = 1:
     GOSUB 570:
     GOTO 2870
 940 H = 0:
     L = N:
     C$ = CHR$(176):
     IF L < 0 OR L > LA OR A$(L) = "" GOTO 80 :
     '  EDIT
 950 N$ = "y":
     IF IT
      THEN
       P = Z - 1:
       GOTO 970
 960 P = 1
 970 C = P + 3:
     O$ = A$(L)
 980 CLS :
     D = - 1:
     R = 0:
     GOSUB 3040
 990 N = 1:
     Q$ = ""
1000 GOSUB 1100:
     GOSUB 3190:
     F = 0:
     M = 0:
     IF A = U
      THEN
       A = 9
1010 IF A = 8 OR A = 9 GOSUB 1130
1020 IF A = 97
      THEN
       A$(L) = O$:
       GOTO 960 :
       '  A
1030 IF X > = LL GOTO 1050:
      ELSE
       IF A = 10 GOSUB 1470
1040 IF A > 24 ON A - 24 GOSUB 1480,1460,1440,540,540,540,1450
1050 IF A > 98 ON A - 98 GOSUB 1160,1190,1210,540,540,1230,1240,540,1
```

Program continued

```
     370
1060 IF A = 115 GOSUB 1380:
      ELSE
       IF A = 117 GOSUB 1410:
        ELSE
         IF A = 120 GOSUB 1430
1070 IF A = 108 GOTO 960:
      ELSE
       N = L:
       IF M
        THEN
         990:
        ELSE
         IF R PRINT @320,;:
          ELSE
           980
1080 IF LL < X PRINT "Line";L;"has";X;"Characters":
     IF IT = 0 GOTO 60
1090 IF IT
      THEN
       RETURN :
      ELSE
       2870
1100 X$ = MID$(A$(L),P,1):
     B = 1:
     PRINT @W,P;
1110 PRINT @C,C$;:
     A$ = INKEY$:
     PRINT @C,X$; CHR$(15);:
     IF A$ = "" GOTO 1110
1120 A = ASC(A$):
     X = LEN(A$(L)):
     IF A = 13
      THEN
       R = 1:
       RETURN :
      ELSE
       RETURN
1130 M = 1:
     P = P + A * 2 - 17:
     IF P < 1
      THEN
       P = 1:
      ELSE
       IF P > X
        THEN
         P = X :
         ' MV
1140 PRINT @C,X$;:
     C = P + 3:
     X$ = MID$(A$(L),P,1):
     PRINT @C,C$;:
     GOSUB 510
1150 K = PEEK(14400):
     IF K = U OR K = V
      THEN
       1130:
      ELSE
       RETURN
1160 Q = P:
     D = C:
     FOR I = 1 TO N:
      GOSUB 1100:
      IF A < U OR A = 96 GOTO 1180 :
      ' C
1170  PRINT @C,A$;:
      MID$(A$(L),P) = A$:
      P = P + 1:
      C = C + 1:
      IF P < = X NEXT I
```

```
1180 P = Q:
     C = D:
     A = U:
     RETURN
1190 IF P + N - 1 > X
      THEN
       N = X - P + 1 :
       '  D
1200 GOSUB 1350:
     Q = P:
     P = P + N:
     GOSUB 1360:
     A$(L) = L$ + R$:
     P = Q:
     RETURN
1210 PRINT @C + V,U$:
     C = C + 1:
     P = P + 1:
     M = 1:
     IF D = - 1
      THEN
       D = P - 2 :
       '  E
1220 A$(L) = 0$ + U$ + STR$(D) + "," + STR$(P - D - 1) + U$:
     RETURN
1230 GOSUB 1350:
     A$(L) = L$ + S$:
     PRINT @C,B$ :
     '  H
1240 R = 0:
     GOSUB 1100:
     IF R OR A = 96 GOSUB 3090:
     X = Y:
     A = U:
     RETURN :
     '  I
1250 IF A = 8 OR A = 9 GOSUB 1130:
     GOTO 1240
1260 IF A = 26 IF F = 0
      THEN
       F = 1:
       GOTO 1240
1270 IF A = 25 IF F < > 1 PRINT @1010,;:
     IF F = 2 PRINT STRING$(13,U);:
     F = 0 :
     GOTO 1240:
      ELSE
       PRINT "High Graphics";:
       F = 2:
       GOTO 1240
1280 IF A = 24 IF F < > 1 PRINT @1010,;:
     IF F = 3 PRINT STRING$(13,U);:
     F = 0 :
     GOTO 1240:
      ELSE
       PRINT "Low Graphics ";:
       F = 3:
       GOTO 1240
1290 IF F = 2 IF A > 47 AND A < 60
      THEN
       A = A + 106:
      ELSE
       IF A > 96
        THEN
         A = A + 69
1300 IF F = 3 IF A > 47 AND A < 54
      THEN
       A = A + 43:
      ELSE
       IF A > 53 AND A < 59
```

Program continued

```
          THEN
           A = A + 69 :
          ELSE
           IF A > 96
            THEN
             A = A + 31
1310 IF F = 1
      THEN
       F = 0
1320 A$ = CHR$(A):
     IF P > X
      THEN
       X = P
1330 GOSUB 1350:
     GOSUB 1360:
     IF A = 60
      THEN
       X$ = CHR$(8):
       B = - 1:
      ELSE
       X$ = A$ :
       '  <
1340 A$(L) = L$ + X$ + R$:
     PRINT @C,B$A$ CHR$(15)R$:
     C = C + B:
     P = P + 1:
     GOTO 1240
1350 L$ = "":
     IF P < 2 RETURN :
      ELSE
       L$ = LEFT$(A$(L),P - 1):
       RETURN
1360 R$ = "":
     IF P > X RETURN :
      ELSE
       R$ = RIGHT$(A$(L),X - P + 1):
       RETURN
1370 D = P:
     GOSUB 1380:
     N = P - D:
     P = D:
     C = P + 3:
     GOTO 1190 :
     '  K
1380 GOSUB 1100:
     Q = P :
     '  S
1390 FOR I = 1 TO N:
      Q = INSTR (Q + 1,A$(L),A$):
      IF Q NEXT I:
     C = C + Q - P:
     P = Q
1400 A = U:
     RETURN
1410 P = P + 1:
     GOSUB 1350:
     GOSUB 1360:
     A$(L) = L$ + CHR$(8) + U$ + R$ :
     '  U
1420 PRINT @C + V,U$:
     C = C + 1:
     P = P + 2:
     M = 1:
     RETURN
1430 A$(L) = A$(L) + S$:
     P = X + 1:
     C = P + 3:
     GOTO 1240 :
     '  X
1440 A$(L) = STRING$((LL - LEN(A$(L))) / 2,U) + A$(L) :
     '  S-U
1450 A$(L) = A$(L) + CHR$(3):
     RETURN :
     '  CL
1460 A$(L) = A$(L) + CHR$(2):
     RETURN :
     '  S-D
1470 A$(L) = A$(L) + CHR$(10):
     R = 1:
     RETURN :
     '  D-A
1480 A$(L) = STRING$(LL - LEN(A$(L)),U) + A$(L):
     RETURN :
     '  S-R
1490 CLS :
     PRINT ,"FORMAT:" :
     '  FORMAT
1500 PRINT "To leave Formats unchanged, press <ENTER>":
     PRINT
1510 PRINT "Line Length =";LL,:
     X = LL:
     GOSUB 1720:
     LL = X
1520 IF LL < 20 OR LL > 122 GOSUB 1770:
     PRINT "20 to 122":
     LL = 60:
     GOTO 1510
1530 PRINT "Line Spaces =";S,:
     X = S:
     GOSUB 1720:
     S = X
1540 PRINT "Line Nos. =  '";N$;"'",:
     X$ = N$:
     GOSUB 1740:
     N$ = X$
1550 IF LA < 0 GOTO 1690:
      ELSE
        IF LP > LA
         THEN
          LP = LA
1560 PRINT "First Print Line =";FL,:
     X = FL:
     GOSUB 1720:
     FL = X
1570 IF FL > LA GOSUB 1770:
     PRINT "0 to";LA:
     FL = 0:
     GOTO 1560
1580 PRINT "Last Print Line =";LP,:
     X = LP:
     GOSUB 1720:
     LP = X
1590 IF LP < FL OR LP > LA GOSUB 1770:
     PRINT FL;"to";LA:
     LP = LA:
     GOTO 1580
1600 PRINT "Left Margin =";LM,:
     X = LM:
     GOSUB 1720:
     LM = X
1610 PRINT "Page Length =";PL,:
     X = PL:
     GOSUB 1720:
     PL = X
1620 PRINT "Page Spacing =";PS,:
     X = PS:
     GOSUB 1720:
     PS = X
1630 PRINT "Page Nos. =  '";PN$;"'",:
     X$ = PN$:
```

Program continued

```
     GOSUB 1740:
     PN$ = X$
1640 PRINT "First Page = ";FP,:
     X = FP:
     GOSUB 1720:
     FP = X
1650 PRINT "Page 1 No. = '";P1$;"'",:
     X$ = P1$:
     GOSUB 1740:
     P1$ = X$
1660 PRINT "Page Stop =  '";P$;"'",:
     X$ = P$:
     GOSUB 1740:
     P$ = X$
1670 PRINT "MX-80 Graphics = '";M$;"'",:
     X$ = M$:
     GOSUB 1740:
     M$ = X$
1680 IF M$ = "y"
      THEN
       POKE 16422,150:
       POKE 16423,255 :
      ELSE
       POKE 16422, PEEK( - 95):
       POKE 16423, PEEK( - 94)
1690 PRINT "Heading = '";H$;"'     ","New =? ";:
     LINE INPUT X$
1700 IF X$ < > ""
      THEN
       H$ = X$
1710 GOTO 60
1720 PRINT "New =? ";:
     N = - 1:
     GOSUB 3160:
     IF A = 13 AND N > - 1
      THEN
       X = N:
       RETURN
1730 IF A = 13 RETURN :
      ELSE
       PRINT :
       GOSUB 1770:
       PRINT "a number",:
       GOTO 1720
1740 PRINT "New (Y/N)? ";:
     N = - 1:
     GOSUB 3160:
     IF A = 13 AND N = - 1 RETURN
1750 IF A = 121 OR A = 110 PRINT :
     X$ = A$:
     RETURN :
      ELSE
       IF A < > 13 PRINT
1760 GOSUB 1770:
     PRINT "Y/N (Unshifted)",:
     GOTO 1740
1770 PRINT "Entry must be ";:
     RETURN
1780 CLS :
     PRINT "Legal Commands are:":
     PRINT :
     '  HELP
1790 PRINT "A  ADD","B  BLANK","C  COMPILE","D  DELETE", "E  EDIT","F
       FORMAT","H  HELP","I  INSERT","J  JUSTIFY", "K  KILL","L  LOAD
     ","M  MOVE","O  OVERLAY","P  PRINT", "R  REPLACE","S  SAVE","V
     VIDEO","X  EXIT":
     PRINT
1800 PRINT "Key 'Shift-@' twice to return from A,I,L,R,S to Command m
     ode"
1810 PRINT "From E & R only, press <ENTER> to return":
     GOTO 60
1820 L = N:
     IF L < 0 OR L > LA GOTO 80 :
     '  INSERT
1830 IF NL = LA + 1 PRINT "FILE FULL":
     GOTO 60:
      ELSE
       IF R GOTO 60
1840 FOR I = LA TO L STEP - 1:
      A$(I + 1) = A$(I):
      NEXT I
1850 A$(L) = "":
     LA = LA + 1:
     L = L - 1:
     IF IT RETURN :
      ELSE
       I = L:
       IT = 1:
       GOTO 90
1860 CLS :
     PRINT "Justifying Line":
     FOR L = 0 TO LA:
      GOSUB 3090 :
      '  JUSTIFY
1870  PRINT @16,L:
      IF Y < 2 OR Y > = LL GOTO 1970
1880  IF A = 2 OR A = 3 OR A = 10
       THEN
        1970:
       ELSE
        J = 0:
        FOR I = 1 TO Y
1890    Q = INSTR (I,A$(L),S$):
       IF Q = 0
        THEN
         I = Y:
         GOTO 1910
1900    IF Q > I
        THEN
         S(J) = Q:
         J = J + 1:
         I = Q
1910    NEXT I:
      IF J = 0 GOTO 1970
1920  K = RND(J) - 1:
      IF INT(J / 2) = J / 2 OR J = 1
       THEN
        N = 1:
       ELSE
        N = 2
1930  FOR I = 1 TO LL - Y:
       T(K) = T(K) + 1:
       K = K + N:
       IF K > J - 1
        THEN
         K = K - J
1940   NEXT I:
      FOR I = J - 1 TO 0 STEP - 1:
       A$ = STRING$(T(I),S$):
       T(I) = 0
1950   A$(L) = LEFT$(A$(L),S(I)) + A$ + RIGHT$(A$(L), LEN(A$(L))
       - S(I))
1960   NEXT I
1970  NEXT L:
     N = 0:
     GOTO 2870
1980 CLS :
     PRINT "Really Kill (Y/N)? ";:
     GOSUB 3160 :
     '  KILL
```

Program continued

```
1990 IF A = 121
      THEN
       RUN :
      ELSE
       60
2000 GOSUB 2800:
     IF A = 96
      THEN
       60:
      ELSE
       PRINT "Loading "; :
       '  LOAD
2010 Q = LA + 1:
     IF DK
      THEN
       OPEN "I",1,FS$:
       PRINT FS$;:
      ELSE
       2050
2020 INPUT #1,LA,LL,S,N$,FL,LP,LM,PL,PS,PN$,FP,P1$,P$,M$,H$
2030 GOSUB 2110:
     IF R GOTO 2790
2040 FOR L = Q TO LA:
      LINE INPUT #1,A$(L):
      GOSUB 3090:
      NEXT L:
     GOTO 2780
2050 INPUT # - 1,LA,LL,S,N$,FL,LP,LM,PL,PS,PN$,FP,P1$,P$,M$,H$
2060 GOSUB 2110:
     IF R GOTO 2790
2070 PRINT H$;:
     FOR I = Q TO LA STEP 4:
      INPUT # - 1,X$(0),X$(1),X$(2),X$(3)
2080  FOR J = 0 TO 3:
       L = I + J:
       X = LEN(X$(J)):
       A$(L) = X$(J):
       IF X = 0 GOTO 2100
2090   FOR K = 1 TO X:
        MID$(A$(L),K) = CHR$( ASC( MID$(A$(L),K,1)) - V):
        NEXT K
2100   NEXT J:
      NEXT I:
     GOTO 2780
2110 IF H$ < > "" IF ASC(H$) = 1
      THEN
       H$ = ""
2120 IF LA + Q < NL
      THEN
       LA = LA + Q:
       LP = LA:
       RETURN
2130 CLS :
     PRINT "Text too long":
     LA = Q - 1:
     LP = LA:
     R = 1:
     RETURN
2140 F = 1:
     INPUT "First Line to Move";F:
     IF F < 0
      THEN
       F = 0 :
       '  MOVE
2150 Z = 0:
     INPUT "Last Line to Move";Z:
     IF Z > LA
      THEN
       Z = LA
2160 IF F > Z
```

```
       THEN
        80:
       ELSE
        N = 0:
        INPUT "Insert before Line";N:
        IN = N
2170 IF N < 0 OR N > LA OR N > = F AND N < = Z
       THEN
        80:
       ELSE
        IT = 1:
        FOR M = F TO Z:
         CLS :
         R = 0
2180   PRINT "Moving":
       GOSUB 1820:
       IF IN < F
        THEN
         K = M + 1:
        ELSE
         K = F
2190   A$(N) = A$(K):
       A$(K) = "":
       IF IN < F
        THEN
         N = N + 1
2200   R = 1:
       GOSUB 580:
       NEXT M:
     N = 0:
     GOTO 2870
2210 CLS :
     PRINT ,"OVERLAY:":
     PRINT :
     '  OVERLAY
2220 PRINT "Search or Replace (S/R)? ";
2230 GOSUB 3160:
     PRINT :
     PRINT :
     IF A = 114
      THEN
       2270:
      ELSE
       IF A < > 115 GOTO 80
2240 PRINT "Word to Search for? ";:
     GOSUB 3110
2250 Y$ = X$:
     IT = 1:
     GOSUB 2310:
     PRINT :
     IF Z = 0 PRINT "Word not found"
2260 GOTO 60
2270 CLS :
     PRINT "Old Word to Overlay? ";:
     GOSUB 3110
2280 PRINT "New Word to Replace old Word? ";:
     Y = X:
     Y$ = X$:
     GOSUB 3110
2290 IF X * Y = 0
      THEN
       80:
      ELSE
       CLS :
       PRINT "Overlaying":
       Z$ = X$:
       GOSUB 2310
2300 N = 0:
     GOTO 2870
2310 FOR L = 0 TO LA:
```

Program continued

```
      Z = 1
2320  Z = INSTR (Z,A$(L),Y$):
      IF Z
       THEN
        X = LEN(A$(L)):
        Z = Z + 1:
       ELSE
        2370
2330  IF IT = 0 GOTO 2360
2340  A = U:
      N = L:
      GOSUB 940:
      PRINT :
      PRINT "Continue Search (Y/N)? ";
2350  H = 1:
      GOSUB 3160:
      Z = P + 1:
      IF A = 121
       THEN
        2320:
       ELSE
        L = LA:
        GOTO 2370
2360  A$(L) = LEFT$(A$(L),Z - 2) + Z$ + RIGHT$(A$(L),X - Y - Z
      + 2):
      GOTO 2320
2370  NEXT L:
     RETURN
2380 X$ = "Printer":
     IF PEEK(14312) < 128 X = FP:
     M = FL:
     GOTO 2410 :
     '  PRINT
2390 PRINT :
     PRINT X$;" not ready.   Abort (Y/N)? ";:
     GOSUB 3160
2400 IF A = 121
      THEN
       60:
      ELSE
       PRINT :
       GOSUB 3150:
       GOTO 2380
2410 R = 1:
     GOSUB 580:
     R = 0:
     IF LP > LA
      THEN
       LP = LA
2420 CLS :
     I = M + PL - 1:
     PRINT "Printing ";H$;
2430 IF PN$ = "n" OR PI$ = "n" AND X = 1 GOTO 2460
2440 LPRINT TAB(LM)H$; STRING$(LL - 7 - LEN(H$),U);
2450 LPRINT "Page"; USING "###";X:
     LPRINT
2460 FOR P = M TO I:
      O$ = A$(P):
      IF P > LP GOTO 2570
2470  M = M + 1:
      IF S LPRINT STRING$(S - 1,13)
2480  LPRINT TAB(LM);:
      IF N$ = "y" LPRINT USING F$;P;:
      R = 4
2490  X$ = "":
      Y = LEN(O$) + 1:
      GOSUB 2610:
      IF Q$ < > U$ LPRINT O$:
      GOTO 2560
2500  GOSUB 2610:
       IF Q$ < > ","
        THEN
         X$ = Q$ + X$:
         GOTO 2500
2510   N = VAL(X$):
       X$ = ""
2520   GOSUB 2610:
       IF Q$ < > U$
        THEN
         X$ = Q$ + X$:
         GOTO 2520
2530   D = VAL(X$):
       GOSUB 2610:
       LPRINT E$;"A"; CHR$(133);E$;"2";
2540   LPRINT O$:
       LPRINT E$;"1"; STRING$(D + LM + R,U); STRING$(N,45)
2550   LPRINT TAB(LM)E$;"A"; CHR$(140);E$;"2";
2560   IF ASC(Q$) < > 2 NEXT P:
       ELSE
        IF I > = M LPRINT STRING$(I - M,13)
2570 IF PS LPRINT STRING$(PS - 1,13)
2580 IF P > LP GOTO 2780
2590 IF P$ = "y"
      THEN
       X$ = "new page":
       PRINT :
       GOSUB 3150
2600 X = X + 1:
     GOTO 2420
2610 Y = Y - 1:
     O$ = LEFT$(O$,Y):
     Q$ = RIGHT$(O$,1):
     RETURN
2620 IF N < 0 OR N > LA
      THEN
       80 :
       '  REPLACE
2630 CLS :
     PRINT "Really Replace Line";N;"(Y/N)? ";:
     GOSUB 3160
2640 IF A = 121
      THEN
       R = 1:
       A$(N) = "":
       I = N - 1:
       GOTO 90:
      ELSE
       60
2650 GOSUB 2800:
     IF A = 96
      THEN
       60:
      ELSE
       PRINT "Saving "; :
       '  SAVE
2660 IF H$ = ""
      THEN
       H$ = CHR$(1):
       GOTO 2680
2670 IF RIGHT$(H$,1) = S$
      THEN
       H$ = LEFT$(H$, LEN(H$) - 1):
       GOTO 2660
2680 IF DK = 0 PRINT H$;:
     GOTO 2730
2690 R = 1:
     GOSUB 570:
     OPEN "O",1,FS$:
     CLS :
     PRINT "Saving ";FS$;
```

Program continued

```
2700 PRINT #1,LA;LL;S;N$;",";FL;LP;LM;PL;PS;PN$;",";FP;P1$;","; P$;",
     ";M$;",";H$
2710 FOR L = 0 TO LA:
      IF ASC( RIGHT$(A$(L),1)) = 10
       THEN
        A$(L) = A$(L) + S$
2720  PRINT #1,A$(L):
      NEXT L:
     GOTO 2780
2730 PRINT # - 1,LA,LL,S,N$,FL,LP,LM,PL,PS,PN$,FP,P1$,P$,M$,H$
2740 FOR L = 0 TO LA STEP 4
2750  FOR J = 0 TO 3:
       I = L + J:
       X = LEN(A$(I)):
       X$(J) = A$(I):
       IF X < 1 GOTO 2770
2760   FOR K = 1 TO X:
        MID$(X$(J),K) = CHR$( ASC( MID$(X$(J),K,1)) + V):
        NEXT K
2770   NEXT J:
      PRINT # - 1,X$(0),X$(1),X$(2),X$(3):
      NEXT L
2780 PRINT " Completed"
2790 OUT 254,1:
     GOTO 60
2800 PRINT :
     PRINT "Disk or Tape (D/T)? ";:
     GOSUB 3160:
     PRINT :
     PRINT
2810 IF A = 116
      THEN
       X$ = "cassette":
       GOSUB 3150:
       CLS :
       DK = 0:
       RETURN
2820 IF A < > 100 GOTO 80
2830 IF FS$ = ""
      THEN
       FS$ = "TEXT/BWP"
2840 PRINT "Current Filespec = '";FS$;"'",:
     LINE INPUT "New =? ";A$
2850 IF A$ < > ""
      THEN
       A = ASC(A$):
       IF A = 96
        THEN
         RETURN :
        ELSE
         FS$ = A$
2860 CLS :
     OUT 254,0:
     DK = 1:
     RETURN
2870 CLS :
     H = 0:
     X = FP - 1 :
     '  VIDEO
2880 FOR M = N TO LA STEP PL:
      X = X + 1:
      IF PN$ = "n" OR P1$ = "n" AND X = 1 GOTO 2900
2890  PRINT H$; STRING$(LL - 7 - LEN(H$),U);"Page"; USING "###";X:
      PRINT
2900  FOR L = M TO M + PL - 1:
       IF L > LA GOTO 2930
2910   IF S PRINT STRING$(S - 1,10)
2920   GOSUB 3040
2930   NEXT L:
      IF L > LA GOTO 2970
```

```
2940  PRINT "#### COMMAND (LETTER):   SCROLL ( [ OR "; CHR$(92); " ):
         NEXT PAGE <ENTER>";
2950  GOSUB 3160:
      IF A > 31 AND A < > 91
       THEN
        M = LA:
       ELSE
        PRINT E$
2960  IF A = 91
       THEN
        L = M:
        GOTO 2980:
       ELSE
        IF A = 10 GOTO 3020
2970  NEXT M:
     GOTO 60
2980 NP = 0:
     Z = 0:
     IF L < 1
      THEN
       L = 0:
       Z = 1
2990 IF PEEK(14400) < > 8 OR Z
      THEN
       M = L:
       L = L + 15:
       PRINT @960,;:
       GOTO 2940
3000 Z = USRO(B):
     NP = 1:
     L = L - 1:
     IF LEN(A$(L)) > 60
      THEN
       Z = USRO(B)
3010 PRINT @0,;:
     GOSUB 3040:
     FOR Z = 0 TO 50:
      NEXT Z:
     GOTO 2980
3020 IF PEEK(14400) < > 16 OR L > LA
      THEN
       M = L - 15:
       GOTO 2940
3030 GOSUB 3040:
     L = L + 1:
     FOR Z = 0 TO 50:
      NEXT Z:
     GOTO 3020
3040 GOSUB 3090:
     IF N$ = "y" PRINT USING F$;L;
3050 PRINT A$(L); CHR$(15);:
     IF A = 2 PRINT CHR$(92);
3060 IF A = 3 PRINT CHR$(93);:
      ELSE
       IF A = U PRINT U$;
3070 IF NP = 0 PRINT B$;:
     IF N$ < > "y" OR Y < > 60 PRINT
3080 RETURN
3090 Y = LEN(A$(L)):
     IF Y
      THEN
       A = ASC( RIGHT$(A$(L),1)):
      ELSE
       A = 0
3100 IF Y > 1 AND A = U
      THEN
       A$(L) = LEFT$(A$(L),Y - 1):
       GOTO 3090:
      ELSE
       RETURN
```

Program continued

```
3110 X$ = ""
3120 GOSUB 3160:
     X$ = X$ + Q$:
     X = LEN(X$):
     IF A = 13 RETURN
3130 IF A = 8 IF X = 0 GOTO 3120:
      ELSE
       X$ = LEFT$(X$,X - 1):
       GOTO 3120
3140 X$ = X$ + A$:
     GOTO 3120
3150 PRINT "Get ";X$;" ready, press <ENTER>";
3160 Q$ = "":
     IF H PRINT U$;B$; CHR$(24);
3170 A$ = INKEY$:
     IF A$ = "" GOTO 3170
3180 A = ASC(A$):
     IF H IF A = 13 PRINT B$:
      ELSE
       PRINT A$;
3190 IF A > 47 AND A < 58
      THEN
       Q$ = Q$ + A$:
       N = VAL(Q$):
       GOTO 3170:
      ELSE
       RETURN
3200 CLS :
     PRINT "Really Exit (Y/N)? ";:
     GOSUB 3160 :
     '  EXIT
3210 IF A < > 121
      THEN
       60:
      ELSE
       CLS :
       CLEAR 50:
       OUT 254,0:
       END
```

The Big Game

by Ken Lord and Joe Boudreau

This program does not provide a crystal ball, but it takes the numbers you give it and analyzes them to assist you in selecting a winning lottery number. The program does a distribution analysis of previous winning numbers. It takes the numbers as you enter them or allows you to add daily to the list of numbers stored within the program. Using the program, you can determine the number of times each digit of the number has appeared, the percentage of the time each digit has occurred and the number of occurrences in each position of a four-digit number which is what most lotteries use.

Examine the DATA lines in lines 920 to 1010 of the Program Listing. Note that there are 16 numbers per line. There are actually four groups of four numbers per line, and a total of 40 groups of four numbers. These represent the actual numbers from a state lottery for a period of 40 days. To use this program with an imbedded data base, you must add one group of four numbers each day as the numbers are announced. As you add the four numbers each day, you also must change the variables R and C in lines 840 and 850. Variable R indicates the number of sets of four numbers to be contained in the DATA lines. Variable C indicates the columns for printing purposes. Variable R is usually incremented by one for every four numbers you add. You may wish to leave variable C unchanged.

When you run the program, you receive the following message:

```
WILL YOU INPUT THE DATA (I) OR
OBTAIN FROM DATA BASE (D)
```

Only the responses I and D are acceptable. If you select option D, the computer processes the DATA lines, then presents the report.

If you wish to enter data directly and not use the DATA lines, the program allows you to state how many rows and columns you will enter, then provides this message:

```
ENTER NUMBERS IN SEQUENCE
LEFT-TO-RIGHT
FOLLOW EACH WITH <ENTER>
```

The program accepts the numbers which result from the product of the rows and columns of figures you provide. Figure 1 shows four four-column numbers given to the program in this manner.

```
STATE LOTTERY TICKET NUMBER FREQUENCY REPORT
1  2  3  4
5  6  7  8
9  0  1  2
3  4  5  6

FINDS   NR   HIT PCT    NUMBER OF TIMES IN POSITION
  1      0     6 %            0     1     0     0
  2      1    12 %            1     0     1     0
  2      2    12 %            0     1     0     1
  2      3    12 %            1     0     1     0
  2      4    12 %            0     1     0     1
  2      5    12 %            1     0     1     0
  2      6    12 %            0     1     0     1
  1      7     6 %            0     0     1     0
  1      8     6 %            0     0     0     1
  1      9     6 %            1     0     0     0
```

Figure 1. *Lottery number frequency report*

Figure 2 is the printout after the DATA BASE (D) option has been specified. It lists 40 numbers in sequence, then produces a report. The reports are distribution reports. Note the four titles: FINDS, NR, HIT PCT, and NUMBER OF TIMES IN POSITION. Under the NR column are 10 digits, 0 through 9. The FINDS column tells the number of times each was found in the data. The HIT PCT tells what percentage of the time each particular number was found. If you add the numbers in the column, you find that the total is not 100 percent. This is because the INTeger function in statement 690 truncates all decimal places. If you require more precision, you can change the INTeger function to a rounding function by adding .5. The HIT PCT indicates a 13 percent incidence for the digits 1 and 7. Examining the 7 first, you see that the highest incidence occurs in the second position of the number. Thus, the final number should include the number 7 in that position. Here's the working number:

_ 7 _ _

The number 1 also has a 13 percent incidence. Had the incidence of the number 1 been higher than the incidence of the number 7 in that position, you would have put a 1 in the second position. At this level of precision, the number had equal usage in positions 3 and 4. Thus, there are now two working numbers:

_ 7 1 _
_ 7 _ 1

The next highest percentage of incidence is 12 percent for the number 4. Its greatest usage is in the fourth position for a value of 8, and its incidence is higher than that of 1, the number previously placed in that position. Thus, there is again one working number:

_ 7 1 4

The number 2 has an 11 percent incidence and has the same number and position as the number 4 placed in the fourth position. Thus, the number 2 is rejected.

Two numbers, 5 and 9, have a 10 percent occurrence. The highest incidence of each number falls in the first position. The values are identical. Thus, there are again two working numbers:

5 7 1 4
9 7 1 4

To discriminate, look at the FINDS column. The number 5 occurred a total of 17 times in the sample data, while the number 9 occurred 16 times. 5, then, has the edge. If you require more precision, you must make the rounding change.

We have not found further precision necessary. This system picked the correct number three times within a 40-day span, and the number 5714 was a winning lottery number shortly after we derived it. Combinations of two-digit and three-digit numbers are also valid as winners. This system has picked several of those.

```
STATE LOTTERY TICKET NUMBER FREQUENCY REPORT
7  2  7  5
9  6  5  1
4  5  6  4
5  4  9  2
9  7  9  4
1  7  3  4
4  2  6  1
2  3  1  2
3  8  7  2
2  7  6  2
4  1  7  1
9  4  6  8
7  8  3  1
6  9  2  8
9  2  3  1
7  8  3  0
1  7  7  0
5  1  1  4
5  0  5  7
5  6  2  1
0  5  9  5
1  7  0  8
```

Figure continued

```
7  7  5  4
4  7  1  9
0  4  7  9
1  6  1  2
9  9  1  6
2  8  4  4
8  4  8  5
3  0  0  4
9  3  5  3
7  2  0  2
2  0  4  3
5  3  5  7
0  1  2  6
9  3  9  4
8  6  4  2
5  7  1  7
5  9  6  1
4  6  1  2

FINDS   NR   HIT PCT    NUMBER OF TIMES IN POSITION
 11      0     6 %             3      3      3      2
 21      1    13 %             4      3      7      7
 19      2    11 %             4      4      3      8
 12      3     7 %             2      4      4      2
 20      4    12 %             5      4      3      8
 17      5    10 %             7      2      5      3
 13      6     8 %             1      5      5      2
 21      7    13 %             5      8      5      3
 10      8     6 %             2      4      1      3
 16      9    10 %             7      3      4      2
```

Figure 2. *Frequency report after DATA BASE option*

Program Listing. *Lottery number frequency*

```
100 CLS
110 CLEAR 2000
120 PRINT CHR$(23);"     STATE LOTTERY"
130 PRINT "     TICKET NUMBER"
140 PRINT "   FREQUENCY  REPORT"
150 PRINT
160 PRINT "WILL YOU INPUT THE DATA (I) OR"
170 PRINT "OBTAIN FROM DATA BASE (D)"
180 R$ = INKEY$
190 IF R$ = "I"
     THEN
      230
200 IF R$ = "D"
     THEN
      830
210 GOTO 180
220 GOTO 100
230 PRINT :
    INPUT "ROWS    ";R
240 INPUT "COLUMNS ";C
250 PRINT "ENTER NUMBERS IN SEQUENCE"
260 PRINT "LEFT - TO - RIGHT"
270 PRINT "FOLLOW EACH WITH <ENTER>"
280 DIM P(R,C),T(10),N(10,C)
290 FOR X = 1 TO R
300  FOR Y = 1 TO C
310   IF R$ = "D"
       THEN
        GOSUB 880
320   IF R$ = "I"
       THEN
        GOSUB 900
330   NEXT Y
340  NEXT X
350 PRINT "PROCESSING DATA"
360 FOR X = 1 TO 10
370  T(X) = 0
380  NEXT X
390 FOR X = 1 TO 10
400  FOR Y = 1 TO C
410   N (X,Y) = 0
420   NEXT Y
430  NEXT X
440 FOR X = 1 TO R
450  FOR Y = 1 TO C
460   T(P(X,Y)) = T(P(X,Y)) + 1
470   NEXT Y
480  NEXT X
490 FOR Y = 1 TO C
500  FOR X = 1 TO R
510   Z = P(X,Y)
520   N(Z,Y) = N(Z,Y) + 1
530   NEXT X
540  NEXT Y
550 INPUT "HOW MANY COPIES: ";D
560 FOR J = 1 TO D
570  LPRINT CHR$(12)
580  LPRINT "STATE LOTTERY TICKET NUMBER FREQUENCY REPORT"
590  LPRINT " "
600  FOR X = 1 TO R
610   FOR Y = 1 TO C
620    LPRINT P(X,Y);
630    NEXT Y
640   LPRINT " "
650   NEXT X
660  LPRINT " " :
     LPRINT " " :
     LPRINT " "
```

Program continued

```
 670  LPRINT "FINDS  NR    HIT PCT       NUMBER OF TIMES IN POSITION"
      :
      LPRINT " "
 680  FOR X = 0 TO 9
 690   PU = INT(T(X) / (R * C) * 100)
 700   LPRINT T(X);"    ";X"    ";PU;"% "; TAB(30);
 710   FOR Y = 1 TO C
 720    LPRINT N(X,Y);"  ";
 730    NEXT Y
 740   LPRINT " "
 750   NEXT X
 760  LPRINT " "
 770  NEXT J
 780 PRINT "WANT MORE COPIES (Y/N)"
 790 PA$ = INKEY$
 800 IF PA$ = "Y"
      THEN
       550
 810 IF PA$ = "N"
      THEN
       1020
 820 GOTO 790
 830 :
     ' THE FOLLOWING TWO CONSTANTS WILL CHANGE WITH EACH RUN
 840 R = 40:
     :
     ' ROWS
 850 C = 4:
     :
     '  COLUMNS
 860 PRINT "PROCESSING DATA"
 870 GOTO 280
 880 READ P(X,Y)
 890 RETURN
 900 INPUT P(X,Y)
 910 RETURN
 920 DATA 7,2,7,5,9,6,5,1,4,5,6,4,5,4,9,2
 930 DATA 9,7,9,4,1,7,3,4,4,2,6,1,2,3,1,2
 940 DATA 3,8,7,2,2,7,6,2,4,1,7,1,9,4,6,8
 950 DATA 7,8,3,1,6,9,2,8,9,2,3,1,7,8,3,0
 960 DATA 1,7,7,0,5,1,1,4,5,0,5,7,5,6,2,1
 970 DATA 0,5,9,5,1,7,0,8,7,7,5,4,4,7,1,9
 980 DATA 0,4,7,9,1,6,1,2,9,9,1,6,2,8,4,4
 990 DATA 8,4,8,5,3,0,0,4,9,3,5,3,7,2,0,2
1000 DATA 2,0,4,3,5,3,5,7,0,1,2,6,9,3,9,4
1010 DATA 8,6,4,2,5,7,1,7,5,9,6,1,4,6,1,2
1020 END
```

INTERFACE

Using the Useful UART

Using the Useful UART

by James N. Devlin

There are a number of good reasons for adding a serial interface to your TRS-80. Information is most economically sent long distances over a single line, and it is difficult to imagine an eight-bit data bus being sent to distant parts of a building to communicate to a remote device. Many output devices require serial data, and, with holding registers, individual bits or sections of a byte can be stripped from a computer word as it goes flying by. Dozens of signals can be sent out from or received by a central computer on a single pair of wires. One of the most inexpensive and simplest devices for converting the parallel data from your computer into serial messages is the UART or Universal Asynchronous Receiver/Transmitter.

I was interested in obtaining hard-copy output from an old Teletype™. Although a number of hardware and software methods exist to do this, I thought that with the wide availability and low cost of UARTs it would be worthwhile to explore their potential. The finished interface required only four chips and I able was to activate it using only Level II BASIC commands. The board is shown in Photo 1. The heart of the interface is an Intel 8251 UART which is also available from NEC, Mostek, Advanced Micro, Motorola, Radio Shack, Poly-Packs, and others.

The special clock circuit is the key to the ease of implementing this interface. One of the problems with most projects involving oscillators is that they must be tuned to some exact frequency. In order to get the proper frequency the circuit must be functioning, but in order to get the circuit working, the clock must be properly tuned—catch 22! The CMOS programmable clock chip 4563 needs no tuning because it has a crystal to drive its on-board oscillator. If the circuit is operating, you know the frequency is correct. The clock chip never has to be tuned. In fact, there are no adjustments of any kind in this interface. The 4536 chip allows you to program or select the desired (or required) frequency. See Figure 1.

Two frequencies originate from the crystal oscillator for use by the UART. The Teletype requires data to be input at the rate of 110 baud, and the rate required for the serial conversion of the data in the UART is 1760 Hz, or 16 times the baud rate. 1760 Hz is the first frequency that you need. The clock circuit allows the UART to compose the bits for the data and attach a start bit, two stop bits, and a parity bit. Other codes, such as Baudot, requiring more or less than the seven ASCII bits can also be assembled. The assembled word is transmitted at 110 baud automatically, and, when the UART is done, it informs you by dropping the transmit flag in the status word.

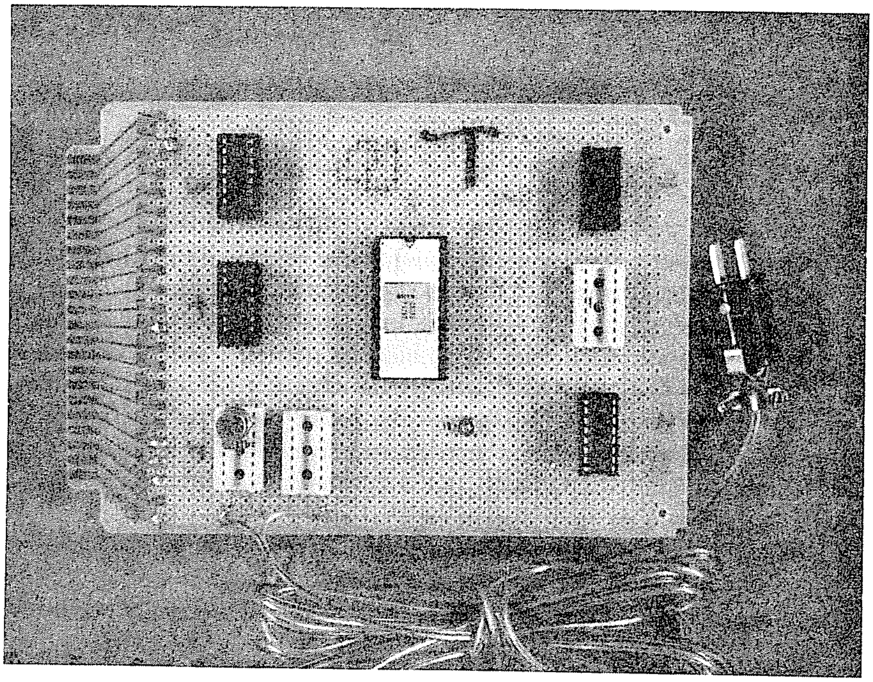


Photo 1. *Interface board*

The UART itself needs a higher clock for its own internal operation; what could be more convenient than to use the crystal frequency itself. The crystal that I use is a 901 kHz radio type. This frequency divided by 2^9 (or 512) yields 1760 Hz, within a few tenths of a cycle. The 4536 chip allows you to divide by any power from 2^0 to 2^{24} by simply grounding the appropriate input pins. Any crystal that can be divided by a power of 2 to yield 1760 Hz will be adequate. The high frequency for the UART is totally arbitrary. It just needs to be greater than 4.5 times the frequency of the transmit data clock.

The 4536 chip oscillator is on-board, so it is unnecessary to construct a circuit. Plenty of gain is provided by the internal amplifiers, and any even number of amplifiers will provide the required 360 degrees of phase shift.

In order not to load the oscillator, I ran the signal through a CMOS gate to isolate it from the UART. I have found CMOS logic to be fast enough for any of the signals encountered in these parts of the computer circuits. The clock chip draws 5 microamps at 5 volts. The entire current drawn by the interface is just the current used by the UART itself, which is approximately 45 milliamps. The small amount of current drawn allows you to derive all the power directly from the TRS-80's internal 5-volt supply. The regulator chip in the TRS-80 should be able to handle this additional amount of current, as it is adequately heat sunk. If the voltage drops slightly, you can readjust the chip. There was no drop when I hooked up the interface, and I have been running it for a year. I did find, however, that for some mysterious reason, the 5-volt line had been disabled by cutting the trace and shorting the pin to ground. If this is the case with yours, you will have to remove the short and bridge the cut trace.

A small, external power supply can be added if needed. If you add any additional circuitry, such as opto-couplers, or if you use TTL devices, you will need the additional power.

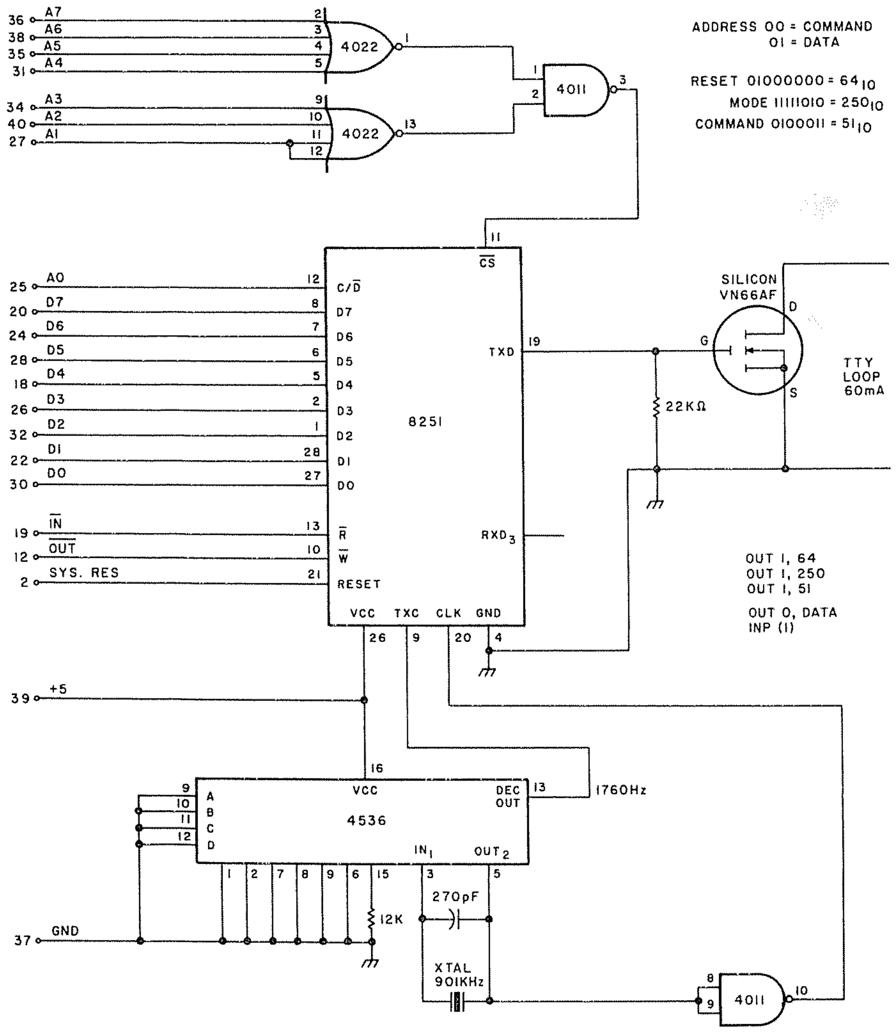


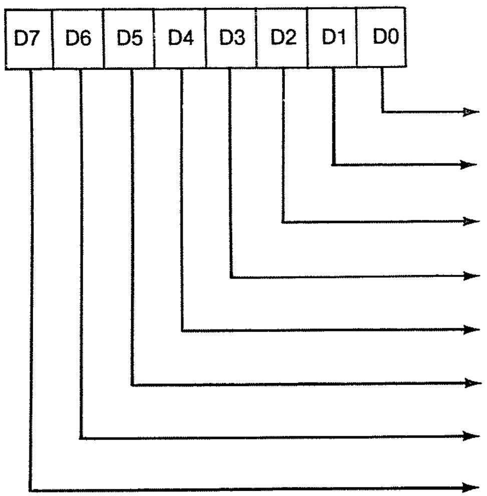
Figure 1. *Clock circuit*

Radio Shack decodes a single port (FF) for the cassette recorder. I used I/O port addressing and selected the 0 and 1 address to send out the two kinds of words required by the UART. This is an easy address to decode with a 4002 dual-4-input NOR gate. The upper seven bits go to the 4002, and the A0 bit (LSB) goes to the C/D input of the UART. This input determines whether the word that is on the data bus is a command word or a data word. The least significant bit, A0, facilitates the selection of the command word and the data word.

The decoded address enables the UART on the chip select pin CS. A memory-mapped address could also be used, but would require decoding all 16 address lines. The rest of the lines of importance are the control signals. These are the read and write strobes which are combined in the TRS-80 with the I/O REQUEST line to produce the IN and OUT signals on the interface output pins. SYSRES is also brought out to the UART's external reset pin. The UART has an internal software reset, but I chose to take the hardware approach.

On the output side, I used a relatively new device that has such outstanding features that it may someday dominate the computer interface field. The device is a VMOS Power FET. It was originally introduced by Siliconix, but is now produced by many other manufacturers. One of the unique properties of this device is that it will pass 2 amps while being driven by nothing more than a CMOS gate. It operates to 100 MHz, has an excellent linear region, and is just as effective as a switch. The output acts similar to a variable resistance that goes as low as 2 Ohms in my device. Other models go to .02 Ohms and withstand 500 volts. The VMOS Power FET does not exhibit secondary breakdown as transistors do, and it has a self-limiting temperature coefficient that prevents thermal runaway. You can parallel two Power FETs and have them share current. I used the VMOS Power FET to short the Teletype loop directly from the UART. Be cautious when doing this. A 20 mil loop is supplied by 110 volts, and you probably need a current-limiting resistor. This particular device works for a 48-volt, 60-mil loop.

The driver required to activate the UART is written in assembly code, however, a short BASIC loader is also given; so it is not necessary to use SYSTEM. Look ahead to the first two lines of the BASIC driver program in the Program Listing. These lines wake up the UART and start the communication with it. The first line, which must appear early in your program or at least once as a direct output, is needed to set up the UART. The first command that you must give the UART is the internal reset, OUT 1,64. Port address 1 tells the UART that this is a command word and that the decimal 64 is the reset bit. The specification sheet recommends that this reset be preceded by three OUT 1,0 commands. The next word, decimal 250, sets up the mode: It selects the baud rate, the character length, parity, and the number of stop bits, as shown in Figure 2. After the mode instruction another com-



a MODE WORD	b COMMAND WORD	STATUS WORD
baud rate	trans enb.	TxRDY
"	term ready	
char. length	rec enb.	
"	send break	
parity enb.	err. reset	
even/odd	req. to send	
stop bits	internal reset	
"	sync search	

D1	D0	
0	1	= 1X
1	0	= 16X
1	1	= 64X

D3	D2	
0	0	= 5 bits
0	1	= 6
1	0	= 7
1	1	= 8

D7	D6	
0	1	= 1 bits
1	0	= 1½ bits
1	1	= 2 bits

Figure 2

mand, OUT 1,51, sets the specific operation of the format selected by the mode word, which in this case is to transmit. See Figure 2.

The driver assembly code is given in Table 1. In Level II 16K machines, you must POKE the starting address into locations 16422 and 16423. When this is done, the LLIST and LPRINT statements vector to the driver location. The driver itself is placed in upper memory. In order to prevent inadvertent destruction of the driver during subsequent programming, you should answer the MEM SIZE? request with 32720. This protects the driver program locations.

The assembly code shown in Table 1 shows the address, the Z-80 code, the assembly code, and the decimal value of the code. A string of zeros is imbedded in the code. These are necessary because the computer is operating about 10 times faster than the UART is running. The delay allows the two systems to get together. The driver will return to BASIC when the TTY has finished printing the designated character.

The BASIC program (see Program Listing) places the assembly code into the desired locations and POKEs the starting address into locations 16422 and 16423. The decimal equivalents of the assembly code are incorporated as DATA statements and read in via a FOR-NEXT loop. This section of the program is executed with a RUN 5000. The lines from 100 to 200 are a short printer test that asks for a number and requests how many times you wish to print it. This routine helps to verify that the driver program is correctly

Address	Hex Code	Mnemonic	Decimal
7FD0	79	MOV A,C	121
7FD1	FE0D	CP CR	254,13
7FD3	C2E87F	JNZ E87F	194,232,127
7FD6	D300	OUT 0,A	211,0
7FD8 to 7FDC	0	0	0,0,0,0,0
7FDD	DB01	INA,1	219,1
7FDF	E601	AND,1	230,1
7FE1	FE01	CP 1	254,1
7FE3	C2DD7F	JNZ DD7F	194,221,127
7FE6	3E0A	LDLF	62,10
7FE8	D300	OUT 0,A	211,0
7FEA to 7FEE	0	0	0,0,0,0,0
7FEF	DB01	INA,1	219,1
7FF1	E601	AND 1	230,1
7FF3	FE01	CP 1	254,1
7FF5	C2EF7F	JNZ EF7F	194,239,127
7FF8	C9	RET	201

Table 1. *Driver assembly code for Z-80 loader*

loaded or to test the driver-printer operation when you hook things up. The first two statements, 5 and 10, should be incorporated in any program that you wish to have printer output from. They can, of course, be directly executed from the keyboard at any time and need only be executed once during a program session.

```
                              Mnemonic    Mnemonic
Address          Line         Op Codes    Operands
7FD0             00100        ORG         7FD0H
7FD0 79          00110        LD          A,C
7FD1 FE0D        00120        CP          0DH
7FD3 C2E87F      00130        JP          NZ,7FE8H
7FD6 D300        00140        OUT         (00H),A
7FD8 00          00150        NOP
7FD9 00          00160        NOP
7FDA 00          00170        NOP
7FDB 00          00180        NOP
7FDC 00          00190        NOP
7FDD DB01        00200        IN          A,(01H)
7FDF E601        00210        AND         01H
7FE1 FE01        00220        CP          01H
7FE3 C2DD7F      00230        JP          NZ,7FDDH
7FE6 3E0A        00240        LD          A,0AH
7FE8 D300        00250        OUT         (00H),A
7FEA 00          00260        NOP
7FEB 00          00270        NOP
7FEC 00          00280        NOP
7FED 00          00290        NOP
7FEE 00          00300        NOP
7FEF DB01        00310        IN          A,(01H)
7FF1 E601        00320        AND         01H
7FF3 FE01        00330        CP          01H
7FF5 C2EF7F      00340        JP          NZ,7FEFH
7FF8 C9          00350        RET
0000             00360        END
00000 TOTAL ERRORS
```

Table 2. *Disassembled version of the Z-80 loader*

You can input from your TTY keyboard through the same UART by putting a connection between the TTY send loop and the RxD input pin on the UART. The UART mode has to be set and the INP(0) statement used. If you have a punch or tape reader, they can also be accommodated by the UART. My purpose was to print from the Teletype; so that is the only mode that I have covered in detail.

When using the Teletype, just load the driver tape, RUN 5000, then load or write the desired program. By typing or by placing the UART set-up lines into your program, LLIST and LPRINT are yours to command. A simple serial interface opens up an entire world of communications possibilities and control capabilities. I hope that this introduction to the UART and its simplicity will add a new dimension to your home computing.

Program Listing. *BASIC loader*

```
   3 REM  UART TTY SUBROUTINE ...... J. DEVLIN
   5 FOR J = 1 TO 3:
      OUT 1,0:
      NEXT J
  10 OUT 1,64:
     OUT 1,250:
     OUT 1.51
 100 INPUT "VALUE";A
 110 INPUT "CYCLES";N
 120 FOR J = 1 TO N
 130  PRINT A
 140  LPRINT A;
 150  NEXT J
 200 END

5000 REM  TTY DRIVER
5040 POKE 16422,208
5050 POKE 16423,127
5060 FOR I = 1 TO 41
5070  READ D
5080  POKE 32719 + I,D
5090  NEXT I
5100 DATA 121,254,13,194,232,127
5110 DATA 211,0,0,0,0,0,0
5120 DATA 219,1,230,1,254,1
5130 DATA 194,221,127,62,10
5140 DATA 211,0,0,0,0,0,0
5150 DATA 219,1,230,1,254,1
5160 DATA 194,239,127,201
5170 END
```

TUTORIAL

String Problems in the TRS-80
Hex, Octal, and Binary To Decimal Conversions

String Problems in the TRS-80

by Arthur R. Jackman

There is a string gremlin in the TRS-80 that occasionally raises its head and causes problems. The problem is related to the use of the string space allocated in the CLEAR statement. When it occurs, the computer comes to a screeching halt, does not respond to the BREAK key, and after a delay, suddenly continues as if nothing had happened.

The problem is that the string space has been used up but not filled and must be cleaned up. During its lapse of consciousness, the computer repacks all of the valid strings in the string space to free the unused space. When the process is complete, operation continues. The problem is directly related to string usage and string space allocation. It can occur under the following conditions:
1) Extensive string processing, such as sorting large string arrays or reading and/or writing large random access data files when string variables are used.
2) A large string array is held in memory.
3) A large string space has been allocated.
Each time a string variable is assigned a value, it is placed in the reserved string space as expected. If that variable is assigned a new value, however, even if it has the same length, it is allocated new string space.

Program Listing 1 demonstrates how the string space is allocated and used. Line 110 sets aside a space of 500 bytes for strings. In line 140, you repeatedly assign A$ different values from B$ and the string equivalent of the index loop I. To see where the strings are placed, set the variable P equal to the address at which A$ is stored. This is done in line 150. Next, print the values of P and A$ to see how they are changing. When you have finished, print out the values of P and A$ one last time to see where they finally ended up.

Look at Figure 1 which shows the output of Program Listing 1. I run this program in a 48K machine with a printer driver located at 64715 and above. Notice that the strings are assigned places at the top end of the string space first and progress downward in memory. The first string is at 64693, and the second is at 64671. The difference is 22 bytes, but the string is only 20 bytes long. The other two bytes are used in line 140 to convert I into a string to attach to B$.

Notice that as the strings are assigned different values the address gets lower and lower until you reach string number 21. At this point, the 500 bytes are used up, even though about 24 bytes are actually used for valid strings. The next string, number 22, is placed at location 64670. This allows 21 bytes for A$ and three bytes for the string equivalent of I. At this point, between strings number 21 and 22, there is a slight delay in program operation while the string space is cleaned up. All of the old strings are purged, and the valid strings move to the top of the string space. The time the program takes to do this is not noticeable because both the string space and the number of strings to be moved are small.

Program Listing 2 expands the program shown in Program Listing 1 to 5000 bytes of string space and stores the strings in an array. Figure 2 shows the output. The program works fine until string 200 is placed. There is a delay of 8.7 seconds while the computer mops up the string area. It moves up all of the strings in the array and packs them to free more string space. Processing continues to string 227. There is a delay of 10.5 seconds for the mop-up operation. The program proceeds to string 231 and a delay of 11.9 seconds, and then on to string 232 and a delay of 21.6 seconds before giving the out of string space error message.

```
64693   THIS IS STRING NO. 1
64671   THIS IS STRING NO. 2
64649   THIS IS STRING NO. 3
64627   THIS IS STRING NO. 4
64605   THIS IS STRING NO. 5
64583   THIS IS STRING NO. 6
64561   THIS IS STRING NO. 7
64539   THIS IS STRING NO. 8
64517   THIS IS STRING NO. 9
64493   THIS IS STRING NO. 10
64469   THIS IS STRING NO. 11
64445   THIS IS STRING NO. 12
64421   THIS IS STRING NO. 13
64397   THIS IS STRING NO. 14
64373   THIS IS STRING NO. 15
64349   THIS IS STRING NO. 16
64325   THIS IS STRING NO. 17
64301   THIS IS STRING NO. 18
64277   THIS IS STRING NO. 19
64253   THIS IS STRING NO. 20
64229   THIS IS STRING NO. 21
       (NOTE SHIFT IN STRING ALLOCATION ADDRESS)
64670   THIS IS STRING NO. 22
64646   THIS IS STRING NO. 23
64622   THIS IS STRING NO. 24
64598   THIS IS STRING NO. 25
64598   THIS IS STRING NO. 25
```

Figure 1. *Program Listing 1 output*

```
64693     THIS IS STRING NO. 1
64671     THIS IS STRING NO. 2
64649     THIS IS STRING NO. 3
64627     THIS IS STRING NO. 4
. . .       . . .
59809     THIS IS STRING NO. 197
59783     THIS IS STRING NO. 198
59757     THIS IS STRING NO. 199
59731     THIS IS STRING NO. 200
      (DELAY OF 8.7 SECONDS)
60397     THIS IS STRING NO. 201
60371     THIS IS STRING NO. 202
60345     THIS IS STRING NO. 203
60319     THIS IS STRING NO. 204
. . .       . . .
59799     THIS IS STRING NO. 224
59773     THIS IS STRING NO. 225
59747     THIS IS STRING NO. 226
59721     THIS IS STRING NO. 227
      (DELAY OF 10.5 SECONDS)
59803     THIS IS STRING NO. 228
59777     THIS IS STRING NO. 229
59751     THIS IS STRING NO. 230
59725     THIS IS STRING NO. 231
      (DELAY OF 11.9 SECONDS)
59715     THIS IS STRING NO. 232
      (DELAY OF 21.6 SECONDS)
OUT OF STRING SPACE IN 160
READY
```

Figure 2. *Program Listing 2 output*

The larger the string space and the greater the number of valid strings in the string space, the longer the delay. I have experienced delays of more than two minutes with string space of 10000 bytes and 7000 to 8000 bytes of valid strings. This becomes significant when you have two or three minutes of delay after every 15 or 20 seconds of disk file processing.

One obvious solution is to avoid assigning strings. You have to assign new values to the strings as you do in a sort subroutine or in processing a disk file. If the new string will fit in the old string space, you can keep the computer from using new string space for the same string name. For a random access disk file, each time you read a record, you have to field a buffer.

```
100 FIELD 1,255 AS X$
200 FIELD 2,20 AS A$, 20 AS B$, 20 AS C$
```

In these examples X$, A$, B$, and C$ are all part of the file buffer area and do not count as part of the string space. When you assign new values to these variables, use LSET and RSET.

```
300 LSET X$=Y$
310 LSET A$=D$(I)
320 RSET B$=STR$(B)
330 RSET C$=E$(E+1)
```

One function of LSET and RSET is to force the computer to recognize and use the current string location to hold the new string value. The new string length must be equal to or smaller than the old string length. With random access files and string file sorting this is usually the case.

```
64693     THIS IS STRING NO. 1
64693     THIS IS STRING NO. 2
64693     THIS IS STRING NO. 3
64693     THIS IS STRING NO. 4
64693     THIS IS STRING NO. 5
64693     THIS IS STRING NO. 6
64693     THIS IS STRING NO. 7
64693     THIS IS STRING NO. 8
64693     THIS IS STRING NO. 9
64693     THIS IS STRING NO. 10
```

Figure 3. *Program Listings 3 and 4 output*

The nice thing about Disk BASIC in this case is that you can use LSET and RSET on any string variable to force the use of the same string location for the new string value. In Program Listing 3, which is similar to Program Listing 1 except for the use of LSET, A$ is always placed in the same memory location. If you do not have Disk BASIC, consider Program Listing 4. In this example, pack the information from B$ into A$ manually. The variable P tells you where A$ is located in memory, and R tells you where B$ is located. At line 180, assign N$ the string equivalent of the index I. Now Q can indicate where N$ is actually located. The value of Q must be determined inside the FOR loop because N$ is always placed in a new area. Figure 3 shows the output of Program Listings 3 and 4.

Beginning at line 200, manually construct A$ by POKEing into A$ the characters in B$ found by PEEKing. This is done through the length of B$ using the index J. You actually use zero to one less than the length, but the character count is the same. After packing B$ into A$, add N$, also by packing. The variable J is used in this loop as an offset since it is pointing to the next character in A$ after the FOR loop ends in line 220. After constructing A$, print the address value and the string itself to verify the solution to the problem of reassignment.

When the strings are in an array and must be moved around, it may be advantageous to move the string pointer instead of the string itself. (See Doug Walker's article "Beyond Shell Metzner" in *80 Microcomputing*, September 1980.) In some special cases, you can convert the strings to numeric data and then process them. An example of this is extracting a record number and a sort key from a disk file to generate a string of the form: 'key'-'record#'. Now convert this to a double precision variable, sort the array numerically, and convert back to strings to process the records into a sorted order in a new file.

Extensive string processing can cause time delays. This happens with large string space allocation, large string arrays, and repetitive string processing. The delay becomes evident when a variable is printed as an indicator during extensive operations. Careful program planning using LSET, RSET, POKE, and PEEK eliminates most or all of the problem and keeps the bits moving.

Program Listing 1

```
100 :
    ' PROGRAM  STRINGS1
110 CLEAR 500
120 B$ = "THIS IS STRING NO."
130 FOR I = 1 TO 25
140  A$ = B$ + STR$(I)
150  P = PEEK( VARPTR(A$) + 1) + 256 * PEEK( VARPTR(A$) + 2)
160  PRINT P,A$
170  NEXT I
180 P = PEEK( VARPTR(A$) + 1) + 256 * PEEK( VARPTR(A$) + 2)
190 PRINT P,A$
200 END
```

Program Listing 2

```
100 :
    '  PROGRAM: STRINGS2
110 CLEAR 5000
120 DIM A$(250)
130 B$ = "THIS IS STRING NO."
140 FOR I = 1 TO 250
150  A$(I) = B$ + STR$(I)
160  P = PEEK( VARPTR(A$(I)) + 1) + 256  * PEEK( VARPTR(A$(I))
     + 2)
170  PRINT P,A$(I)
180  NEXT I
190 FOR I = 1 TO 200
200  P = PEEK( VARPTR(A$(I)) + 1) + 256 * PEEK( VARPTR(A$(I))
     + 2)
210  PRINT P,A$(I)
220  NEXT I
230 END
```

Program Listing 3

```
100 :
    '  PROGRAM: STRINGS3
110 CLEAR 1000
120 A$ = STRING$(22,32)
130 B$ = "THIS IS STRING NO."
140 FOR I = 1 TO 10
150  LSET A$ = B$ + STR$(I)
160  P = PEEK( VARPTR(A$) + 1) + 256 * PEEK( VARPTR(A$) + 2)
170  PRINT P,A$
180  NEXT I
190 END
```

Program Listing 4

```
100 :
    '  PROGRAM: STRINGS4
110 CLEAR 1000
120 A$ = STRING$(22,32)
130 B$ = "THIS IS STRING NO."
```

```
140 N$ = STRING$(3,32)
150 P = PEEK( VARPTR(A$) + 1) + 256 * PEEK( VARPTR(A$) + 2):
    IF P > 32767 P = - 1 * (65536 - P)
160 R = PEEK( VARPTR(B$) + 1) + 256 * PEEK( VARPTR(B$) + 2):
    IF R > 32767 R = - 1 * (65536 - R)
170 FOR I = 1 TO 10
180  N$ = STR$(I)
190  P = PEEK( VARPTR(A$) + 1) + 256 * PEEK( VARPTR(A$) + 2):
     IF P > 32767 P = - 1 * (65536 - P)
200  Q = PEEK( VARPTR(N$) + 1) + 256 * PEEK( VARPTR(N$) + 2):
     IF Q > 32767 Q = - 1 * (65536 - Q)
210  FOR J = 0 TO LEN(B$) - 1
220   POKE P + J, PEEK(R + J)
230   NEXT J
240  FOR K = 0 TO LEN(N$) - 1
250   POKE P + J + K, PEEK(Q + K)
260   NEXT K
270  PRINT 65536 + P,A$
280  NEXT I
290 END
```

Hex, Octal, and Binary to Decimal Conversions

by Clay Lansdown

My intent in writing this program was to learn something of the string commands I knew very little about. A program that would input the hexadecimal numbers 0 through F and convert them to decimal values seemed like a good way to practice using at least two or three string commands. As it turned out, writing the program was a good learning experience, especially since I decided to add a decimal to hex conversion, an octal to decimal conversion, a decimal to octal conversion, and then to round it out, binary to decimal and decimal to binary.

To understand the program (see Program Listing), you must first understand how the conversion between the various bases is done. Each digit of a number has a weight based on its position. For example, the number 343 equals (3 × 100) + (4 × 10) + (3 × 1) or 300 + 40 + 3. Notice that the 3 on the far right has a weight of 3 × 1 or 3, while the 3 on the far left has a weight of 3 × 100 or 300. The weights for five digits can be written as follows: 10,000, 1,000, 100, 10, and 1. If the number 10,269 is broken apart and written under the weights:

10,000	1,000	100	10	1
1	0	2	6	9

a pattern is evident. Notice that each weight differs from the preceding weight by a factor of 10. The number base our number system uses is base 10.

The pattern can be represented in the following manner: 10^4, 10^3, 10^2, 10^1, 10^0. Since $10^4 = 10,000$, $10^3 = 1,000$, $10^2 = 100$, $10^1 = 10$, and $10^0 = 1$, the pattern can be seen to consist of the base raised to increasing powers. The same system works for other bases as well.

The pattern for base 2 (binary) is as follows: 2^4, 2^3, 2^2, 2^1, 2^0. The weights are 16, 8, 4, 2, and 1, and the binary number 10101 is (1 × 16) + (0 × 8) + (1 × 4) + (0 × 2) + (1 × 1) or 16 + 4 + 1 or 21 decimal. This simple series of multiplication and addition is a binary to decimal conversion.

Hexadecimal to decimal conversions are done in a similar fashion, using 16 as the base. The weights are 16^4, 16^3, 16^2, 16^1, 16^0 or 65536, 4096, 256, 16, and 1. The numbers get large quickly, but follow the same pattern as base 10 and base 2. The hex number 24031 can be written under the appropriate weights and handled like the other bases:

65536	4096	256	16	1
2	4	0	3	1

This is (2 × 65536) + (4 × 4906) + (0 × 256) + (3 × 16) + (1 × 1) or 131,072 + 16,384 + 48 + 1; or 147,505 decimal. The TRS-80 can only address 64K (65535) words of memory; so four hex digits, instead of the five used in the example, are normally used. The letters A through F used in hex notation represent the numbers 10 through 15. In base 10, the number after 9 is not a new number. If you add 9 + 1, you get a 0 in the units column and carry a 1 to the tens column, the result is the number 10. 10 is made up of two old numbers, 1 and 0.

In hex you don't get a carry when you add 9 + 1; so you need a new number. A is that number. Similarly, 9 + 2 = B, 9 + 3 = C, 9 + 4 = D, 9 + 5 = E, 9 + 6 = F. To convert the hex number BE3F, place the digits under the weights:

4096	256	16	1
B	E	3	F

Since B = 9 + 2 or 11, E = 9 + 5 or 14, and F = 9 + 6 or 15, the multiplication and addition series look like this: (11 × 4096) + (14 × 256) + (3 × 16) + (15 × 1) or 45,056 + 3,584 + 48 + 15 or 48,703 decimal.

The base 8 or octal number system only uses the digits 0 through 7; so no new numbers are needed. The conversion process is performed the same as for the other bases. The octal weights are 8^4, 8^3, 8^2, 8^1, 8^0, or 4096, 512, 64, 8, and 1.

To convert octal 72317 to decimal, write the number under the weights:

4096	512	64	8	1
7	2	3	1	7

(7 × 4096) + (2 × 512) + (3 × 64) + (1 × 8) + (7 × 1) equals 28,672 + 1,024 + 192 + 8 + 7 equals 29,903 decimal.

Notice that the conversion process is the same for both base 2 and base 16. Only the weights change. There are programs which take advantage of this and convert from base 10 to almost any other practical base. These universal base conversion programs are difficult to use since they do not allow entry of numbers larger than 9 as letters.

This program is simple since it is essentially self prompting. If you enter a number that is out of range for the base being converted, the program tells you. When you are doing binary to decimal conversions, you can put a space between each group of four binary numbers. The binary number 1101001110111110 can be entered as 1101 0011 1011 1110. Since I tend to get confused by a lot of 1s and 0s, this feature is very helpful for me.

Converting from base 10 is even easier because this process involves successive subtractions. To convert 535 from base 10 to hexadecimal, look at the hex weights, 4096, 256, 16, and 1 and subtract the largest number possible (leaving a positive result) from 535. 256 is the largest number that meets this requirement; 535 – 256 = 279. 256 can be subtracted from 279 leaving 23. Since you subtracted 256 from 535 twice, put a 2 in the column three places to the left of the decimal point. Now subtract the largest number possible from the remainder 23. That is 23 – 16 = 7. You can only subtract 16 once; so put a 1 in the second column. You can subtract 1 from 7 seven times. This puts a 7 in the first column. The result is 217 hexadecimal. If you had been able to subtract a hex weight 10 times, you would have put an A in the appropriate column. For example, the decimal number 3,584 converts to E00 hexadecimal, since 256 is the largest number that can be subtracted from 3,584, and it can be subtracted fourteen times with no remainder. 14 is expressed as E in hex; so an E goes in the column three places to the left of the decimal point, yielding E00.

A conversion from 742 decimal to octal is performed in the same way. The octal weights are 4096, 512, 64, 8 and 1. 512 can be subtracted once leaving 230. 64 can be subtracted three times leaving 38. You can subtract 8 four times leaving 6. One can be subtracted six times leaving 0. The result is 1346 octal.

Decimal to binary conversion follows the same scheme, but since the binary system uses only two digits, 0 and 1, a given weight is subtracted only once. To convert from 43 decimal to binary, look at the binary weights 32, 16, 8, 4, 2, and 1 and subtract the largest weight possible from 43; 43 – 32 = 11. Put a one in the sixth column, then subtract the largest weight possible from 11. Subtracting 16 from 11 leaves a negative number; so you put a 0 in the fifth column and subtract 8 from 11; 11 – 8 = 3. Put a 1 in the fourth column. A 0 goes in the third column because subtracting 4 from 3 gives a negative result. Subtract 2 from 3, leaving 1. Put a 1 in the second column and subtract 1 from 1, leaving 0. Put a 1 in the first column. The converted number is 101011.

If this seems confusing, a few practice sessions using the conversion program to check your results should make it more clear. If you are not interested in how the conversion process works, you can use the program just to make conversions. There are numerous safeguards and prompts built in to keep you from going astray.

Program Listing. *Base conversion*

```
  5 :
    ' BASE CONVERSION - SEPTEMBER 1980  - N.C.L.
 10 CLS
 20 CLEAR :
    PRINT
 30 PRINT "TO CONVERT FROM HEX TO DECIMAL,ENTER 1":
    PRINT
 40 PRINT "TO CONVERT FROM DECIMAL TO HEX,ENTER 2":
    PRINT
 50 PRINT "TO CONVERT FROM OCTAL TO DECIMAL,ENTER 3":
    PRINT
 60 PRINT "TO CONVERT FROM DECIMAL TO OCTAL,ENTER 4":
    PRINT
 70 PRINT "TO CONVERT FROM BINARY TO DECIMAL,ENTER 5":
    PRINT
 80 PRINT "TO CONVERT FROM DECIMAL TO BINARY,ENTER 6":
    PRINT
 90 PRINT "TO RETURN TO MENU,ENTER ANY NON HEX LETTER":
    PRINT
100 INPUT "ENTER 1, 2, 3, 4, 5, OR 6";P
110 ON P GOTO 470,130,920,730,1410,1150
120 GOTO 10
130 CLS
140 CLEAR :
    ' BEGIN DECIMAL TO HEXADECIMAL CONVERSION
150 D(1) = 1:
    D(2) = 16:
    D(3) = 256:
    D(4) = 4096
160 INPUT "INPUT DECIMAL NUMBER";DS$
170 IF ASC(DS$) > 70
     THEN
      10
180 D = VAL(DS$)
190 IF D > 65535
     THEN
      PRINT "NUMBER ENTERED IS TOO LARGE,IT MUST BE SMALLER THAN 655
      36":
      GOTO 140
200 DC = D
210 IF DC > = 4096
     THEN
      DC = DC - 4096:
     ELSE
      GOTO 240
220 D1 = D1 + 1
230 GOTO 210
240 IF DC > = 256
     THEN
      DC = DC - 256:
     ELSE
      GOTO 270
250 C1 = C1 + 1
260 GOTO 240
270 IF DC > = 16
     THEN
      DC = DC - 16:
     ELSE
      GOTO 300
280 B1 = B1 + 1
290 GOTO 270
300 IF DC > = 1
     THEN
      DC = DC - 1 :
     ELSE
      GOTO 330
310 A1 = A1 + 1
```

Program continued

```
320 GOTO 300
330 IF D1 < 10
     THEN
      D$(1) = STR$(D1):
     ELSE
      :
      X = D1:
      Y = 1:
      GOSUB 400
340 IF C1 < 10
     THEN
      D$(2) = STR$(C1):
     ELSE
      :
      X = C1:
      Y = 2:
      GOSUB 400
350 IF B1 < 10
     THEN
      D$(3) = STR$(B1):
     ELSE
      :
      X = B1:
      Y = 3:
      GOSUB 400
360 IF A1 < 10
     THEN
      D$(4) = STR$(A1):
     ELSE
      :
      X = A1:
      Y = 4:
      GOSUB 400
370 PRINT :
    PRINT "THE HEXADECIMAL EQUIVALENT IS :"
380 PRINT D$(1) + D$(2) + D$(3) + D$(4):
    PRINT
390 GOTO 140
400 IF X = 10
     THEN
      D$(Y) = "A"
410 IF X = 11
     THEN
      D$(Y) = "B"
420 IF X = 12
     THEN
      D$(Y) = "C"
430 IF X = 13
     THEN
      D$(Y) = "D"
440 IF X = 14
     THEN
      D$(Y) = "E"
450 IF X = 15
     THEN
      D$(Y) = "F"
460 RETURN
470 CLS
480 CLEAR :
    ' BEGIN HEXADECIMAL TO DECIMAL CONVERSION
490 C(4) = 1:
    C(3) = 16:
    C(2) = 256:
    C(1) = 4096
500 INPUT "ENTER HEX NUMBER";H$
510 IF ASC(H$) > 70
     THEN
      10
520 V = LEN(H$)
530 IF V = 1
```

```
      THEN
       H$ = "0" + "0" + "0" + H$
540 IF V = 2
      THEN
       H$ = "0" + "0" + H$
550 IF V = 3
      THEN
       H$ = "0" + H$
560 IF V > 4
      THEN
       PRINT "ENTER NO MORE THAN 4 DIGITS PLEASE":
       GOTO 500
570 FOR X = 4 TO 1 STEP - 1
580  H$(X) = MID$(H$,X,1)
590  NEXT X
600 FOR X = 1 TO 4
610  IF ASC(H$(X)) < 58
       THEN
        S = S + VAL(H$(X)) * C(X):
       ELSE
        GOSUB 660
620  NEXT X
630 PRINT :
    PRINT "THE DECIMAL EQUIVALENT IS :"
640 PRINT S:
    PRINT
650 GOTO 480
660 H$(P) = H$(X)
670 IF ASC(H$(X)) > 70
      THEN
       PRINT "A,B,C,D,AND E ARE THE ONLY LETTERS YOU CAN USE.":
       GOTO 480
680 FOR L = 65 TO 70
690  K = L - 55
700  IF ASC(H$(P)) = L
       THEN
        S = S + C(X) * K
710  NEXT L
720 RETURN
730 CLS
740 CLEAR :
    ' BEGIN DECIMAL TO OCTAL CONVERSION
750 O(1) = 1:
    O(2) = 8:
    O(3) = 64:
    O(4) = 512:
    O(5) = 4096:
    O(6) = 32768
760 INPUT "ENTER DECIMAL NUMBER";O$
770 IF ASC(O$) > 70
      THEN
       10
780 O = VAL(O$)
790 IF O > 65535
      THEN
       PRINT "NUMBER ENTERED IS TOO LARGE,IT MUST   BE SMALLER THAN 6
       5535":
       GOTO 740
800 Y = 1
810 FOR X = 6 TO 1 STEP - 1
820  IF O > = O(X)
       THEN
        O = O - O(X) :
       ELSE
        GOTO 840
830  D(Y) = D(Y) + 1:
     GOTO 820
840  Y = Y + 1
850  NEXT X
860 FOR Y = 1 TO 6
```

Program continued

```
870  D$(Y) = STR$(D(Y))
880  NEXT Y
890 PRINT :
    PRINT "THE OCTAL EQUIVALENT IS :"
900 PRINT D$(1) + D$(2) + D$(3) + D$(4) + D$(5) + D$(6):
    PRINT
910 GOTO 740
920 CLS
930 CLEAR :
    ' BEGIN OCTAL TO DECIMAL CONVERSION
940 C(6) = 1:
    C(5) = 8:
    C(4) = 64:
    C(3) = 512:
    C(2) = 4096:
    C(1) = 32768
950 INPUT "ENTER OCTAL NUMBER";F$
960 IF ASC(F$) > 70
     THEN
      10
970 V = LEN(F$)
980 IF V > = 6
     THEN
      1030
990 FOR X = 1 TO 6 - V
1000  FOR Y = 1 TO Y
1010   F$ = "0" + F$
1020   NEXT X
1030  IF VAL(F$) > 177777
       THEN
        PRINT "THE NUMBER ENTERED IS TOO LARGE,IT MUST BE SMALLER THA
        N 200,000":
        GOTO 930
1040  FOR X = 6 TO 1 STEP - 1
1050   F$(X) = MID$(F$,X,1)
1060   NEXT X
1070  FOR X = 1 TO 6
1080   IF VAL(F$(X)) > 7
        THEN
         PRINT "OCTAL NUMBERS USE NO DIGITS LARGER THAN 7":
         GOTO 930:
        ELSE
         GOSUB 1130
1090   NEXT X
1100  PRINT :
      PRINT "THE DECIMAL EQUIVALENT IS :"
1110  PRINT D:
      PRINT
1120  GOTO 930
1130  D = D + ( VAL(F$(X)) * C(X))
1140  RETURN
1150  :
      ' BEGIN DECIMAL TO BINARY CONVERSION
1160  CLS
1170  CLEAR 200:
      DIM D(17),D$(17)
1180  Y = 1
1190  FOR X = 1 TO 16
1200   D(X) = Y
1210   Y = 2 * Y
1220   NEXT X
1230  INPUT "ENTER DECIMAL NUMBER";B$
1240  IF ASC(B$) > 70
       THEN
        10
1250  D = VAL(B$)
1260  IF D > 65535
       THEN
        PRINT "NUMBER ENTERED IS TOO LARGE, IT MUST BE SMALLER THAN 6
        5536":
         GOTO 1230
1270   FOR X = 16 TO 1 STEP - 1
1280    IF D > = D(X)
         THEN
          D = D - D(X):
          D(X) = 1 :
         ELSE
          D(X) = 0
1290    NEXT X
1300   FOR X = 1 TO 16
1310    D$(X) = STR$(D(X))
1320    NEXT X
1330   FOR X = 16 TO 1 STEP - 1
1340    D$ = D$ + D$(X)
1350    NEXT X
1360   PRINT :
       PRINT "THE BINARY EQUIVALENT IS :"
1370   A$ = LEFT$(D$,8):
       B$ = MID$(D$,9,8):
       C$ = MID$(D$,17,8):
       E$ = RIGHT$(D$,8)
1380   PRINT USING "%      %      %      %      ";A$;B$;C$;E$
1390   PRINT
1400   GOTO 1170
1410   :
       ' BEGIN BINARY TO DECIMAL CONVERSION
1420   CLEAR
1430   CLS
1440   DIM B(18)
1450   DEFDBL D
1460   Y = 1
1470   FOR X = 1 TO 16
1480    B(X) = Y
1490    Y = 2 * Y
1500    NEXT X
1510   L = 0:
       Y = 0:
       S = 0:
       INPUT "ENTER BINARY NUMBER";B$
1520   IF ASC(B$) > 70
        THEN
         10
1530   D = VAL(B$):
       B$ = STR$(D)
1540   L = LEN(B$)
1550   FOR X = L TO 2 STEP - 1
1560    Y = Y + 1
1570    T$ = MID$(B$,X,1)
1580    T = VAL(T$)
1590    IF T > 1 PRINT "1 AND 0 ARE THE ONLY DIGITS USED IN THE BINARY
         NUMBER SYSTEM.":
        GOTO 1510
1600    IF T = 1
         THEN
          S = S + T * B(Y)
1610    NEXT X
1620   PRINT "THE DECIMAL EQUIVALENT IS:"
1630   PRINT S:
       PRINT
1640   GOTO 1510
```

UTILITY

EMOD—EDTASM Modifications for the Model III

by Winford Rister and Rick Steinberg

What's that? EDTASM was supposed to be available for your Model III when? We ran into the same problem and fortunately were able to do something about it. This chapter will show you how you can patch Radio Shack's Editor/Assembler to run on your Model III computer.

We tried to run EDTASM on the Model III. We loaded the tape at 500 baud, and it gave every indication of loading correctly, but when we tried to execute the program, all we got were some strange characters printed at the top of the screen.

The challenge issued by those Greek letters was too strong to ignore. Even though we were told that EDTASM for the Model III would be out shortly, we decided that we would try to modify the Editor/Assembler we had. Our copy of EDTASM was version 1.2. If you have a different version of EDTASM, most of the specifics of this article do not apply to you.

Disassembling EDTASM

Armed with a copy of the article "Custom EDTASM" by John T. Blair which appeared in the August 1980 issue of *80 Microcomputing*, we planned a course of action. First we needed a disassembled listing of EDTASM. We were sure that once we got the source listing of EDTASM we would be able to patch it to run on the Model III.

We had already written a disassembler program in BASIC, all we needed to do was figure out how to load EDTASM and the disassembler into the computer at the same time. (Both normally load into lower memory.) We finally discovered a method of CLOADing the BASIC program anywhere we wanted. We simply had to change the contents of locations 16548/9 to point to the address where we wanted to start loading, then type CLOAD.

We were ready to disassemble EDTASM. First, we used the SYSTEM command to load EDTASM on the Model III computer. We then pressed RESET (to reinitialize system RAM), POKEd our CLOAD address into locations 16548/9, and CLOADed the disassembler program into upper memory (starting at location X'6000'). Everything worked fine, and within a couple of hours we had our assembly-language listing of EDTASM.

Using the listing we had generated, along with the "Custom EDTASM" article, we found out some very important information about memory usage within EDTASM. Let us caution you once more that the EDTASM we're going to discuss is version 1.2. Although other versions are similar, they probably won't be identical to this one. The Editor/Assembler (version 1.2) uses memory starting at location X'4100' as follows:

X'4100'	Start of temporary storage and I/O buffers
X'42FF'	Start of stack (increases toward X'4100')
X'4300'	Start of device control blocks (DCB)
X'4318'	Start of cassette I/O driver software
X'43CE'	Start of general DCB handler routine
X'45F6'	Start of I/O linkage code
X'468A'	EDTASM cold start location
X'5CDC'	Start of user program buffer

Unfortunately, the Model III uses several memory locations between X'4100' and X'43EB' for system functions (cassette I/O status, time storage, etc.). This, coupled with a difference in hardware, prevents the Model I EDTASM from running on the Model III computer. The I/O driver software (cassette, keyboard, display, and line printer functions) is slightly different for the two hardware configurations. It also means that the I/O drivers contained within EDTASM will not work on the Model III computer.

Changing the I/O driver addresses inside EDTASM was not hard, thanks to the way Radio Shack had structured their program. All calls to the keyboard, display, and line printer go to subroutines located at X'4605', X'460A', and X'460F' respectively. Each of these routines loads the address of the appropriate device control block (DCB) into the DE register pair and then jumps to a general DCB handler routine. We changed the code within the I/O linkage routines to load the DE pair with the addresses of the system's DCBs. Then we changed the jump address for the general DCB handler routine to the one in ROM. Program Listing 1 is a disassembler listing of the I/O linkage code from EDTASM. Program Listing 2 shows the code for I/O linkage found in the ROM, and Program Listing 3 is the code in EDTASM after the changes were made. Radio Shack has not formally documented the address of the general DCB handler routine in the ROM; therefore, this address is subject to change without notice in later versions of the ROM.

At this point, we had redirected the keyboard, display, and line printer I/O to the corresponding ROM routines. Cassette I/O was another matter entirely. Radio Shack does not use the DCB concept for cassette control. We had to search the EDTASM listing to find all jumps and calls to the resident cassette I/O routines which start at location X'4318' and relink them to the ROM routines. This was rather tedious, but we succeeded at last. By redirecting all of the I/O to the ROM, we created a hole in EDTASM between locations X'4300' and X'45F6'. The machine code in this area was no longer necessary!

Now another tedious task had to be performed—changing all references to the temporary storage area starting at X'4100'. We searched out all addresses in the program listing that began with X'41' and changed them to start at X'44', and changed the stack start address to X'45F5' (immediately below the start of our program code). When we had made a list of all required corrections, we went back to the Model III computer and loaded EDTASM along with a monitor program we had written. Use of the monitor allowed us to insert all of our revisions and make a SYSTEM-format tape of the resulting editor/assembler.

We were ready to start testing the revised program. We used the SYSTEM command to load the new EDTASM, and it seemed to load correctly. With great eagerness, we pressed / ENTER, and watched the screen. Lo and behold, MODEL III ED/ASM (VER 1.2M) appeared in the upper left-hand corner. We then checked out all of the EDTASM functions and found a few problem areas. We quickly worked out patches for these, and before the week was over, we had an editor/assembler that would run on a Model III computer! We then wrote an assembly-language program using EDTASM on the Model III computer. The program is named EMOD, and its purpose is to produce a tape of the editor/assembler that is compatible with a 16K Model III computer. The final version of this program appears in Program Listing 5, but before we discuss what it does, let's look at the BASIC program in Program Listing 4.

All this BASIC program does is POKE the EMOD program into memory and execute it. It does have an important feature that you should be aware of. Since typographical errors are a normal occurrence, and since EMOD *must be correct*, we have installed a method for verification of all the data statements in this program. The data statements are divided into groups of 64 elements, each with an associated checksum. As EMOD is POKEd into memory, the checksum is developed. The sum is verified after each group of data statements has been processed by comparing it to the proper element of the C array. If any errors are detected, a message is printed which identifies the group containing the error.

Running the BASIC Program

Before you run the program, you need to buy Radio Shack's Editor/Assembler, version 1.2. Have the salesperson verify the version number before you buy the tape by loading the tape on a Model I computer and executing it. The title is printed at the top of the screen along with the version number. If you use someone else's EDTASM, you will be violating Tandy's copyright (not to mention that you won't have an instruction manual for the program).Now that you have EDTASM, here's what to do:
1) Press RESET on your Model III and answer the memory size question with 28600.

2) Type in the BASIC program from Program Listing 1 and run it.
3) If you get a CHECKSUM ERROR message, count the data statements in groups of four until you get to the group with the error. Correct the error and rerun the program.
4) If there are no errors the BASIC program will end. CSAVE the final copy of the BASIC program, and then execute EMOD by typing:

```
X = USR(0) ENTER
```

5) EMOD will set the baud rate to 500 baud, destroy your BASIC program, and ask you to insert EDTASM in the cassette unit.
6) Set the cassette recorder to PLAY and then press any key on the keyboard. If you press BREAK, EMOD will return to Step 6.
7) When you press a key, EMOD will read the tape of EDTASM. The familiar asterisks will flash in the upper right corner of the screen.
8) If you have inserted a tape other than EDTASM, or if any errors are detected while EDTASM is being read, an E will be displayed in place of the left asterisk, and EMOD will return to Step 6
9) When EDTASM has been read, EMOD will verify the version number. If this is wrong, a message will be printed, and EMOD will return to Step 6.
10) If the version number of EDTASM is correct, EMOD will make the required changes and ask you to ready a tape to record the new version of EDTASM.
11) Insert a fresh tape and set the cassette unit to RECORD.
12) Press any key on the keyboard to continue. If you press BREAK, EMOD will return to Step 6.
13) When you press a key, EMOD will set the baud rate to 1500 baud and write a SYSTEM-format tape of the new version of EDTASM. EMOD will then go back to Step 12. To get out of this loop, press RESET.
You should now have an editor/assembler for your Model III computer!

Let's Talk About EDTASM

There are a few things you should know about the editor/assembler version you have created. It will run only on a Model III computer with the Model III's BASIC. Along with the modifications required for machine compatibility, we have added some other features to make the editor/assembler easier to use.

One command in your EDTASM manual is the B command. Execution of this command normally causes a jump to location 0 (reset). This is necessary on the Model I computer because EDTASM uses some system RAM which must be reinitialized before you can use BASIC again. On the Model III version, however, we don't use any reserved memory; so a reset is unnecessary. Therefore, we have changed the jump address for the B command to the BASIC warm start address (X'1A19'). In most instances, this will allow you to get out of EDTASM and back in without disturbing text in your source code buffer. To get back to EDTASM, enter the SYSTEM command and then type / ENTER. The editor/assembler will have changed the default system execution address to its own warm start address (18138–X'46DA').

The original Editor/Assembler had no provision for protecting any memory at the top of RAM. If you had a machine-language routine loaded, EDTASM might write over it. The Model III version corrects this problem by using the memory size from BASIC to determine the top of usable memory.

Another feature of the Model III is its ability to operate at two different cassette baud rates. There is no facility within the Editor/Assembler to change the baud rates; so we added one. To do this, we had to delete a command from the original EDTASM. In looking over this list of EDTASM commands, we decided that the most dispensable was the T command, so we deleted it and installed a C command. The C command calls the ROM routine $SETCAS which allows you to switch baud rates.

Using the high (1500) baud rate is acceptable for reading and writing source tapes from EDTASM, but the low baud rate must always be used to produce SYSTEM-format (machine-language) tapes. The Editor/Assembler produces object code too slowly for the 1500-baud cassette speed; therefore, to insure that the low baud rate is used, we have installed a patch in EDTASM that automatically sets the low baud rate whenever an object module is being written to cassette. When the assembly is complete, the baud rate is restored to its previous setting.

A Look at EMOD

Program Listing 5 contains the assembly-language program, EMOD. Since you now have an editor/assembler for your Model III, you may be interested in some of the routines we have used in this program.

The first part of the program contains the variables and lookup tables required to modify EDTASM. The program actually begins execution at the label START. The first section is a modified form of a routine that loads a SYSTEM-format tape into memory. We have set it up to ignore anything loaded below address X'45F6', and we also ignore the execution address that is read by the two calls at label LDEND.

The middle sections of code perform the modifications to EDTASM based on the tables at the start of the program. In several places we have used RST 18H, a call to a very handy ROM subroutine that performs a double-precision compare on the HL and DE register pairs and alters the accumulator and the flags. On return to the caller, the carry flag is set if the number in the HL register pair is less than the number in the DE pair. The carry flag is reset if HL is greater than or equal to DE, and the zero flag is set if HL equals DE. This call is especially handy because it uses only one byte of memory in your program.

The last section of the program writes the contents of memory to cassette tape in SYSTEM tape format. The first byte of a SYSTEM-format tape must be the ASCII character U (X'55'). This is followed by a six-character program name. If the program name is less than six characters, it must be filled with spaces on the end. Following the name are the DATA blocks and an END block.

Each DATA block starts with an X'3C', followed by a byte indicating the block size. A block size of zero indicates that there are 256 bytes in the block. The block size byte is followed by two bytes that give the starting load address for the block. The least significant byte of the load address comes first, and then the most significant byte. After the load address are the data bytes to be loaded. The number of data bytes must equal the block size.

Following the last data byte in the block is a byte which represents the checksum. The checksum is calculated by adding all of the bytes after the block size (including the load address). Only the eight least significant bits of the sum make up the checksum. The byte after the checksum byte starts another DATA block or an END block. The first byte of an END block is an X'78'. The X'78' is followed by two bytes which represent the execution address of the program. The execution address, like the load address, has its least significant byte first.

We'll add a couple of *don'ts* about reading and writing cassette tapes at 1500 baud. In a machine-language program, there is very little time between calls to the cassette I/O routines. If you execute too many instructions between calls to the $CSOUT routine, you'll produce a tape that gives data errors when you try to read it back in. When reading a cassette tape, don't enable interrupts between calls to $CSIN. This gives very unpredictable results. Also, don't attempt to call any of the video display routines during a cassette read. These routines enable interrupts before returning to your program.

We hope this article has given you the capability to produce an editor/assembler for your Model III computer. The revised EDTASM has run successfully on two Model IIIs with different versions of the Model III BASIC ROM: the earliest version which did not have keyboard accessible control characters, and a later version which did. We have used a few undocumented ROM routines and system RAM locations, however, so it's possible that the program will not work on your computer. If this happens, a slight modification to EMOD may be necessary.

Program Listing 1. *EDTASM I/O linkage routine prior to modification*

```
45F6   C5         PUSH   BC
45F7   06 01      LD     B,01
45F9   18 19      JR     4614H
45FB   C5         PUSH   BC
45FC   0602       LD     B,02
45FE   18 14      JR     4614H
4600   C5         PUSH   BC
4601   06 04      LD     B,04
4603   18 0F      JR     4614H
4605   11 00 43   LD     DE,4300H   ;GET ADR OF KBD DCB
4608   18 EC      JR     45F6H
460A   11 08 43   LD     DE,4308H   ;GET ADR OF DSPL DCB
460D   18 EC      JR     45FBH
460F   11 10 43   LD     DE,4310H   ;GET ADR OF LP DCB
4612   18 E7      JR     45FBH
4614   C3 CE 43   JP     43CEH   ;GO TO DCB HANDLER
```

Program Listing 2. *I/O linkage routines from the Model III ROM*

```
0013   C5         PUSH   BC
0014   06 01      LD     B,01
0016   18 2E      JR     0046H
0018   C3 06 40   JP     4006H
001B   C5         PUSH   BC
001C   06 02      LD     B,02
001E   18 26      JR     0046H
 .
 .
 .
002B   11 15 40   LD     DE.4015H ;GET KBD DCB ADR
002E   18 E3      JR     0013H
0030   C3 0F 40   JP     400FH
0033   11 1D 40   LD     DE,401DH ;GET DSPL DCB ADR
0036   18 E3      JR     001BH
0038   C3 12 40   JP     4012H
003B   11 25 40   LD     DE,4025H ;GET LP DCB ADR
003E   18 DB      JR     001BH
 .
 .
 .
0046   C3 74 06   JP     0674H ;GO TO DCB HANDLER
```

Program Listing 3. *EDTASM I/O linkage routine after modification*

```
45F6   C5          PUSH   BC
45F7   06 01       LD     B,01
45F9   18 19       JR     4614H
45FB   C5          PUSH   BC
45FC   0602        LD     B,02
45FE   18 14       JR     4614H
4600   C5          PUSH   BC
4601   06 04       LD     B,04
4603   18 0F       JR     4614H
4605   11 15 40    LD     DE,4015H   ;GET ADR OF KBD DCB
4608   18 EC       JR     45F6H
460A   11 1D 40    LD     DE,401DH   ;GET ADR OF DSPL DCB
460D   18 EC       JR     45FBH
460F   11 25 40    LD     DE,4025H   ;GET ADR OF LP DCB
4612   18 E7       JR     45FBH
4614   C3 74 06    JP     0674H   ;GO TO DCB HANDLER
```

Program Listing 4. *EMOD BASIC listing*

```
  1 REM    EMOD  GENERATOR     STEINBERG/RISTER  5/81
 10 CLEAR 100
 15 RESTORE :
    DIM C(14)
 20 CLS :
    PRINT @128,"THIS PROGRAM WILL GENERATE AND EXECUTE 'EMOD'."
 25 INPUT "PRESS ENTER TO CONTINUE.  READY";ZX$
 30 IF PEEK(16561) = 184 IF PEEK(16562) = 111
     THEN
      100
 35 PRINT "YOU FORGOT TO SET MEMORY SIZE.":
    PRINT
 40 PRINT "I WILL SET IT FOR YOU BUT YOU WILL HAVE TO "
 45 PRINT "TYPE RUN AGAIN."
 50 POKE 16561,184:
    POKE 16562,111:
    END
100 GOSUB 200
130 FOR I = 1 TO 14:
     READ C(I):
     NEXT
140 N = 28672
150 GOTO 300
200 PRINT @256,"READING, CONVERTING, AND POKING DATA"
210 RETURN
300 FOR J = 1 TO 14
305  CS = 0
315  GOSUB 200
320  M = N
325  PRINT " 64 ELEMENT BLOCK BEGINNING AT";
330  B(1) = INT(M / 4096)
335  B(2) = INT((M - B(1) * 4096) / 256)
340  B(3) = INT((M - (B(1) * 4096) - (B(2) * 256)) / 16)
345  B(4) = INT((M - B(1) * 4096 - B(2) * 256 - B(3) * 16))
350  B1$ = ""
355  FOR I = 1 TO 4
360   B$(I) = STR$(B(I))
```

```
  365   IF B(I) > 9
         THEN
          B$(I) = CHR$(B(I) + 55)
  370   B1$ = B1$ + RIGHT$(B$(I),1)
  375   NEXT
  380  PRINT " ";B1$;" HEX"
  390  FOR K = 1 TO 64
  395   IF N > 29546
         THEN
          7000
  400   READ H$:
        P$ = H$:
        H = 0
  420   A$ = RIGHT$(H$,1)
  440   IF A$ > = "A" AND A$ <. = "F"
         THEN
          A = ASC(A$) - 55
         ELSE
          A = VAL(A$)
  460   H = A:
        A$ = LEFT$(H$,1)
  480   IF A$ > = "A" AND A$ < = "F"
         THEN
          A = ASC(A$) - 55
         ELSE
          A = VAL(A$)
  500   A = A * 16:
        H = H + A
  510   POKE N,H
  520   N = N + 1 :
        PRINT P$;"  ";
  530   CS = CS + H
  540   NEXT K
  550  IF CS < > C(J)
        THEN
         1000
  600  CLS
  700  NEXT
 1000 PRINT :
      PRINT " CHECKSUM ERROR - AT BLOCK ";J;" CHECK DATA"
 1200 GOTO 9999
 7000 IF CS < > C(J)
       THEN
        1000
 8000 POKE 16526,72:
      POKE 16527,114
 8020 PRINT :
      PRINT :
      PRINT "THIS PROGRAM IS SELF DESTRUCTIVE"
 8030 PRINT :
      PRINT :
      PRINT "IF YOU HAVE NOT CSAVED THIS PROGRAM - DO IT NOW!!"
 8050 PRINT :
      PRINT "EMOD HAS BEEN LOADED - TO EXECUTE: "
 8100 PRINT :
      PRINT " TYPE  X=USR(0)  (ENTER)"
 9999 END
20000 DATA 4432,7014,7277,8320,6372,5318,5524
20010 DATA 4814,6338,5193,6203,5783,5796,4475
24999 REM  7000H  BLOCK 1
25000 DATA 00,78,8A,46,49,4E,53,45,52,54,20,45,44,54,41,53
25010 DATA 4D,20,46,4F,52,20,52,45,41,44,0D,52,45,41,44,59
25020 DATA 20,54,41,50,45,20,54,4F,20,57,52,49,54,45,20,45
25030 DATA 44,54,41,53,4D,0D,57,52,4F,4E,47,20,56,45,52,53
25039 REM  BLOCK 2
25040 DATA 49,4F,4E,20,4F,46,20,45,44,54,41,53,4D,0D,46,08
25050 DATA 8F,A1,A7,AA,B6,B9,E7,F0,47,03,0F,1C,3D,49,0C,34
25060 DATA 43,46,78,91,AF,C4,CF,D4,F7,FB,FE,4A,0D,0E,2E,49
25070 DATA 63,88,8E,97,AD,B2,C0,D8,E1,FF,4B,0B,04,18,59,6A
25079 REM  BLOCK 3
```

Program continued

```
25080 DATA 99,AD,B5,CA,D8,DF,EC,4C,14,03,1E,32,3F,43,51,5B
25090 DATA 5E,80,8D,92,96,9D,AB,CC,DD,E1,E4,ED,F0,4D,0D,02
26000 DATA 05,0A,21,33,3B,63,66,89,9D,AC,DA,E3,4E,01,44,4F
26010 DATA 0C,0D,22,2A,32,59,5D,62,76,7E,9F,AA,E0,50,03,AB
26019 REM  BLOCK 4
26020 DATA BC,D0,51,0A,23,3B,43,50,C7,CB,CF,E9,EC,F1,52,16
26030 DATA 0F,30,33,3A,5D,60,67,8D,A2,BD,C0,C4,C8,CB,D0,D3
26040 DATA D6,DA,E3,E9,ED,F4,53,07,22,8B,A8,AE,BA,BE,D3,54
26050 DATA 04,66,93,9E,BD,55,0B,0A,17,1C,2C,2F,44,63,9A,B9
26059 REM  BLOCK 5
26060 DATA C0,E3,56,04,5C,91,96,B2,57,05,03,26,57,80,F6,58
26070 DATA 04,17,4E,64,F3,59,13,00,07,0B,0F,28,31,34,3C,3F
26080 DATA 43,58,62,71,74,7E,98,A0,A7,CB,5B,01,ED,5C,13,04
26090 DATA 1B,2C,5E,61,6E,73,76,79,88,92,9A,9E,A9,AC,B2,BE
26099 REM  BLOCK 6
27000 DATA C4,D8,06,46,15,40,0B,46,1D,40,10,46,25,40,15,46
27010 DATA 74,06,B1,46,D0,45,DB,46,D0,45,DE,46,DB,45,43,47
27020 DATA 64,02,30,49,19,1A,C0,4B,F0,45,46,4D,64,02,58,4D
27030 DATA 96,02,5B,4D,35,02,70,4D,35,02,79,4D,35,02,96,4D
27039 REM  BLOCK 7
27040 DATA 35,02,B4,4D,35,02,BB,4D,35,02,35,4F,87,02,3A,4F
27050 DATA 64,02,40,4F,64,02,49,4F,64,02,98,52,E5,45,9B,52
27060 DATA D0,45,AF,59,64,02,B3,59,64,02,B7,59,64,02,CD,5C
27070 DATA 64,02,D3,5C,64,02,95,46,0A,00,21,DA,46,22,DF,40
27079 REM  BLOCK 8
27080 DATA 2A,B1,40,BA,46,08,00,00,21,DA,46,22,04,42,1E,47
27090 DATA 02,FE,05,C9,48,1B,4D,4F,44,45,4C,20,49,49,49,20
28000 DATA 45,44,2F,41,53,4D,20,28,56,45,52,20,31,2E,32,4D
28010 DATA A9,0B,49,1E,44,E7,4C,52,D1,4C,4E,02,4A,41,E7,51
28019 REM  BLOCK 9
28020 DATA 57,23,4D,46,C8,4B,5B,76,4C,0A,78,4C,45,C5,4D,43
28030 DATA D1,45,A6,4D,03,00,00,00,D1,45,25,CD,42,30,3A,11
28040 DATA 42,32,EF,45,C9,CD,F8,01,3A,EF,45,32,11,42,C9,CD
28050 DATA D4,45,AF,32,11,42,C3,25,4F,0D,CD,F8,01,C3,2B,47
28059 REM  BLOCK 10
28060 DATA CD,1B,02,CD,49,00,3D,C0,31,00,70,21,48,72,22,04
28070 DATA 42,21,04,70,CD,40,72,11,F6,45,21,00,44,36,00,23
28080 DATA 7C,FE,60,38,F8,AF,32,11,42,CD,96,02,CD,35,02,FE
28090 DATA 55,20,3F,21,0B,70,06,06,CD,35,02,BE,20,34,23,10
28099 REM  BLOCK 11
29000 DATA F7,CD,35,02,FE,78,28,34,FE,3C,20,26,CD,35,02,47
29010 DATA CD,35,02,6F,CD,35,02,67,85,4F,CD,35,02,32,00,70
29020 DATA 81,4F,DF,38,04,3A,00,70,77,23,10,EE,CD,35,02,B9
29030 DATA 28,CF,3E,45,32,3E,3C,CD,F8,01,18,8C,CD,35,02,CD
29039 REM  BLOCK 12
29040 DATA 35,02,CD,F8,01,3A,E1,48,FE,31,20,07,3A,E3,48,FE
29050 DATA B2,28,09,21,36,70,CD,1B,02,C3,48,72,21,4E,70,3E
29060 DATA 44,56,23,46,23,5E,23,12,10,FB,11,42,71,DF,38,EF
29070 DATA 5E,23,56,23,7E,23,12,13,7E,23,12,11,B6,71,DF,38
29079 REM  BLOCK 13
29080 DATA EF,5E,23,56,23,46,23,7E,23,12,13,10,FA,11,40,72
29090 DATA DF,38,EE,21,1B,70,CD,40,72,11,F0,5C,32,11,42,CD
30000 DATA 87,02,3E,55,CD,64,02,21,0B,70,06,06,7E,23,CD,64
30010 DATA 02,10,F9,21,D0,45,3E,3C,CD,64,02,AF,47,CD,64,02
30019 REM  BLOCK 14 (SHORT)
30020 DATA 7D,CD,64,02,7C,CD,64,02,85,4F,7E,CD,64,02,81,4F
30030 DATA 23,10,F7,79,CD,64,02,DF,38,DC,21,01,70,06,03,7E
30040 DATA 23,CD,64,02,10,F9,CD,F8,01,18,A8
```

Program Listing 5. *EMOD assembly-language listing*

```
                 00010 ;  EMOD - 4/31/81 - R. STEINBERG
                 00020 ;PURPOSE:  THIS PROGRAM LOADS EDTASM (V 1.2) INTO
                 00030 ;   MEMORY, MODIFIES IT TO RUN ON A MODEL III
                 00040 ;   COMPUTER, AND THEN PRODUCES A SYSTEM-FORMAT
                 00050 ;   TAPE OF THE MODIFIED PROGRAM AT 1500 BAUD.
                 00060 ;
7000             00070 ORGN     EQU     7000H
```

```
                00080 ;
                00090 ;EQUATES
021B            00100 VDLINE  EQU     021BH
0049            00110 KBWAIT  EQU     0049H
3042            00120 SETCAS  EQU     3042H
0296            00130 CSHIN   EQU     0296H
0235            00140 CSIN    EQU     0235H
0287            00150 CSHWR   EQU     0287H
0264            00160 CSOUT   EQU     0264H
01F8            00170 CSOFF   EQU     01F8H
45D0            00180 STK     EQU     45D0H
                00190 ;
                00200 ;PROGRAM START
7000            00210         ORG     ORGN
7000 00         00220 CSAVE   DEFB    0
7001 78         00230 ENDBLK  DEFB    78H
7002 8A46       00240         DEFW    468AH
7004 49         00250 RDMSG   DEFM    'INSERT '
700B 45         00260 EDNAME  DEFM    'EDTASM FOR READ'
701A 0D         00270         DEFB    0DH
701B 52         00280 WRMSG   DEFM    'READY TAPE TO WRITE EDTASM'
7035 0D         00290         DEFB    0DH
7036 57         00300 VMSG    DEFM    'WRONG VERSION OF EDTASM'
704D 0D         00310         DEFB    0DH
                00320 ;
                00330 ;T44 - THIS TABLE DESCRIBES REFERENCES TO THE EDTASM RAM
                00340 ;   STORAGE AREA.
                00350 ;
704E 4608       00360 T44     DEFW    0846H
7050 8FA1       00370         DEFW    0A18FH
7052 A7AA       00380         DEFW    0AAA7H
7054 B6B9       00390         DEFW    0B9B6H
7056 E7F0       00400         DEFW    0F0E7H
                00410 ;
7058 4703       00420         DEFW    0347H
705A 0F1C       00430         DEFW    1C0FH
705C 3D         00440         DEFB    3DH
                00450 ;
705D 490C       00460         DEFW    0C49H
705F 3443       00470         DEFW    4334H
7061 4678       00480         DEFW    7846H
7063 91AF       00490         DEFW    0AF91H
7065 C4CF       00500         DEFW    0CFC4H
7067 D4F7       00510         DEFW    0F7D4H
7069 FBFE       00520         DEFW    0FEFBH
                00530 ;
706B 4A0D       00540         DEFW    0D4AH
706D 0E2E       00550         DEFW    2E0EH
706F 4963       00560         DEFW    6349H
7071 888E       00570         DEFW    8E88H
7073 97AD       00580         DEFW    0AD97H
7075 B2C0       00590         DEFW    0C0B2H
7077 D8E1       00600         DEFW    0E1D8H
7079 FF         00610         DEFB    0FFH
                00620 ;
707A 4B0B       00630         DEFW    0B4BH
707C 0418       00640         DEFW    1804H
707E 596A       00650         DEFW    6A59H
7080 99AD       00660         DEFW    0AD99H
7082 B5CA       00670         DEFW    0CAB5H
7084 D8DF       00680         DEFW    0DFD8H
7086 EC         00690         DEFB    0ECH
                00700 ;
7087 4C14       00710         DEFW    144CH
7089 031E       00720         DEFW    1E03H
708B 323F       00730         DEFW    3F32H
708D 4351       00740         DEFW    5143H
708F 5B5E       00750         DEFW    5E5BH
7091 808D       00760         DEFW    8D80H
7093 9296       00770         DEFW    9692H
```

Program continued

```
7095 9DAB      00780          DEFW    0AB9DH
7097 CCDD      00790          DEFW    0DDCCH
7099 E1E4      00800          DEFW    0E4E1H
709B EDF0      00810          DEFW    0F0EDH
               00820 ;
709D 4D0D      00830          DEFW    0D4DH
709F 0205      00840          DEFW    0502H
70A1 0A21      00850          DEFW    210AH
70A3 333B      00860          DEFW    3B33H
70A5 6366      00870          DEFW    6663H
70A7 899D      00880          DEFW    9D89H
70A9 ACDA      00890          DEFW    0DAACH
70AB E3        00900          DEFB    0E3H
               00910 ;
70AC 4E01      00920          DEFW    014EH
70AE 44        00930          DEFB    44H
               00940 ;
70AF 4F0C      00950          DEFW    0C4FH
70B1 0D22      00960          DEFW    220DH
70B3 2A32      00970          DEFW    322AH
70B5 595D      00980          DEFW    5D59H
70B7 6276      00990          DEFW    7662H
70B9 7E9F      01000          DEFW    9F7EH
70BB AAE0      01010          DEFW    0E0AAH
               01020 ;
70BD 5003      01030          DEFW    0350H
70BF ABBC      01040          DEFW    0BCABH
70C1 D0        01050          DEFB    0D0H
               01060 ;
70C2 510A      01070          DEFW    0A51H
70C4 233B      01080          DEFW    3B23H
70C6 4350      01090          DEFW    5043H
70C8 C7CB      01100          DEFW    0CBC7H
70CA CFE9      01110          DEFW    0E9CFH
70CC ECF1      01120          DEFW    0F1ECH
               01130 ;
70CE 5216      01140          DEFW    1652H
70D0 0F30      01150          DEFW    300FH
70D2 333A      01160          DEFW    3A33H
70D4 5D60      01170          DEFW    605DH
70D6 678D      01180          DEFW    8D67H
70D8 A2BD      01190          DEFW    0BDA2H
70DA C0C4      01200          DEFW    0C4C0H
70DC C8CB      01210          DEFW    0CBC8H
70DE D0D3      01220          DEFW    0D3D0H
70E0 D6DA      01230          DEFW    0DAD6H
70E2 E3E9      01240          DEFW    0E9E3H
70E4 EDF4      01250          DEFW    0F4EDH
               01260 ;
70E6 5307      01270          DEFW    0753H
70E8 228B      01280          DEFW    8B22H
70EA A8AE      01290          DEFW    0AEA8H
70EC BABE      01300          DEFW    0BEBAH
70EE D3        01310          DEFB    0D3H
               01320 ;
70EF 5404      01330          DEFW    0454H
70F1 6693      01340          DEFW    9366H
70F3 9EBD      01350          DEFW    0BD9EH
               01360 ;
70F5 550B      01370          DEFW    0B55H
70F7 0A17      01380          DEFW    170AH
70F9 1C2C      01390          DEFW    2C1CH
70FB 2F44      01400          DEFW    442FH
70FD 639A      01410          DEFW    9A63H
70FF B9C0      01420          DEFW    0C0B9H
7101 E3        01430          DEFB    0E3H
               01440 ;
7102 5604      01450          DEFW    0456H
7104 5C91      01460          DEFW    915CH
7106 96B2      01470          DEFW    0B296H
```

```
                ;
7108 5705                DEFW    0557H
710A 0326                DEFW    2603H
710C 5780                DEFW    8057H
710E F6                  DEFB    0F6H
                ;
710F 5804                DEFW    0458H
7111 174E                DEFW    4E17H
7113 64F3                DEFW    0F364H
                ;
7115 5913                DEFW    1359H
7117 0007                DEFW    0700H
7119 0B0F                DEFW    0F0BH
711B 2831                DEFW    3128H
711D 343C                DEFW    3C34H
711F 3F43                DEFW    433FH
7121 5862                DEFW    6258H
7123 7174                DEFW    7471H
7125 7E98                DEFW    987EH
7127 A0A7                DEFW    0A7A0H
7129 CB                  DEFB    0CBH
                ;
712A 5B01                DEFW    015BH
712C ED                  DEFB    0EDH
                ;
712D 5C13                DEFW    135CH
712F 041B                DEFW    1B04H
7131 2C5E                DEFW    5E2CH
7133 616E                DEFW    6E61H
7135 7376                DEFW    7673H
7137 7988                DEFW    8879H
7139 929A                DEFW    9A92H
713B 9EA9                DEFW    0A99EH
713D ACB2                DEFW    0B2ACH
713F BEC4                DEFW    0C4BEH
7141 D8                  DEFB    0D8H
                ;
7142            T44E     EQU     $
                ;
                ;WDTBL - THIS TABLE DESCRIBES ALL FULL-WORDS WITHIN
                ;  EDTASM THAT REQUIRE MODIFICATION.  MOST OF THESE
                ;  RELINK JUMP AND CALL ADDRESSES TO THE ASSOCIATED
                ;  ROM FUNCTIONS.
                ;
7142 0646       WDTBL    DEFW    4606H
7144 1540                DEFW    4015H     ;KBD DCB
7146 0B46                DEFW    460BH
7148 1D40                DEFW    401DH     ;DSPL DCB
714A 1046                DEFW    4610H
714C 2540                DEFW    4025H     ;LP DCB
714E 1546                DEFW    4615H
7150 7406                DEFW    0674H   ;ADR OF ROM DCB HANDLER
7152 B146                DEFW    46B1H
7154 D045                DEFW    STK
7156 DB46                DEFW    46DBH
7158 D045                DEFW    STK
715A DE46                DEFW    46DEH
715C DB45                DEFW    45DBH
715E 4347                DEFW    4743H
7160 6402                DEFW    CSOUT
7162 3049                DEFW    4930H
7164 191A                DEFW    1A19H
7166 C04B                DEFW    4BC0H
7168 F045                DEFW    45F0H
716A 464D                DEFW    4D46H
716C 6402                DEFW    CSOUT
716E 584D                DEFW    4D58H
7170 9602                DEFW    CSHIN
7172 5B4D                DEFW    4D5BH
7174 3502                DEFW    CSIN
```

Program continued

```
7176 704D                     DEFW    4D70H
7178 3502                     DEFW    CSIN
717A 794D                     DEFW    4D79H
717C 3502                     DEFW    CSIN
717E 964D                     DEFW    4D96H
7180 3502                     DEFW    CSIN
7182 B44D                     DEFW    4DB4H
7184 3502                     DEFW    CSIN
7186 BB4D                     DEFW    4DBBH
7188 3502                     DEFW    CSIN
718A 354F                     DEFW    4F35H
718C 8702                     DEFW    CSHWR
718E 3A4F                     DEFW    4F3AH
7190 6402                     DEFW    CSOUT
7192 404F                     DEFW    4F40H
7194 6402                     DEFW    CSOUT
7196 494F                     DEFW    4F49H
7198 6402                     DEFW    CSOUT
719A 9852                     DEFW    5298H
719C E545                     DEFW    45E5H
719E 9B52                     DEFW    529BH
71A0 D045                     DEFW    STK
71A2 AF59                     DEFW    59AFH
71A4 6402                     DEFW    CSOUT
71A6 B359                     DEFW    59B3H
71A8 6402                     DEFW    CSOUT
71AA B759                     DEFW    59B7H
71AC 6402                     DEFW    CSOUT
71AE CD5C                     DEFW    5CCDH
71B0 6402                     DEFW    CSOUT
71B2 D35C                     DEFW    5CD3H
71B4 6402                     DEFW    CSOUT
                      ;
71B6                  WDTBLE  EQU     $
                      ;
                      ;VARTBL - THIS TABLE DESCRIBES ALL SECTIONS OF PROGRAM
                      ;   CODE TO BE OVERLAID WITHIN EDTASM.
                      ;
71B6 9546             VARTBL  DEFW    4695H
71B8 0A                       DEFB    10
71B9 00                       NOP
71BA 21DA46                   LD      HL,46DAH    ;GET EDTASM WARM START ADR
71BD 22DF40                   LD      (40DFH),HL  ;STORE AT 'SYSTEM' DEFAULT
                      ;                            EXECUTION ADR ( /<ENTER>)
71C0 2AB140                   LD      HL,(40B1H)  ;GET BASIC TOP OF MEMORY
                      ;
71C3 BA46                     DEFW    46BAH
71C5 08                       DEFB    8
71C6 00                       NOP        ;DELETE MODEL I CASSETTE TAPE
71C7 00                       NOP        ;  INITIALIZATIONS.
71C8 21DA46                   LD      HL,46DAH ; AND INSERT BREAK PROCESSING
71CB 220442                   LD      (4204H),HL
                      ;
71CE 1E47                     DEFW    471EH
71D0 02                       DEFB    2
71D1 FE05                     CP      05
                      ;
71D3 C948                     DEFW    48C9H
71D5 1B                       DEFB    27         ;INSTALL NEW PGM TITLE
71D6 4D                       DEFM    'MODEL III ED/ASM (VER 1.2M'
71F0 A9                       DEFB    ')'+80H
                      ;
71F1 0B49                     DEFW    490BH
71F3 1E                       DEFB    30
71F4 44                       DEFB    'D'
71F5 E74C                     DEFW    4CE7H      ;ALTER COMMAND TABLE TO
71F7 52                       DEFB    'R'        ; DELETE 'T' COMMAND AND
71F8 D14C                     DEFW    4CD1H      ; ADD 'C' COMMAND.
71FA 4E                       DEFB    'N'
71FB 024A                     DEFW    4A02H
```

```
71FD 41                      DEFB    'A'
71FE E751                    DEFW    51E7H
7200 57                      DEFB    'W'
7201 234D                    DEFW    4D23H
7203 46                      DEFB    'F'
7204 C84B                    DEFW    4BC8H
7206 5B                      DEFB    5BH
7207 764C                    DEFW    4C76H
7209 0A                      DEFB    0AH
720A 784C                    DEFW    4C78H
720C 45                      DEFB    'E'
720D C54D                    DEFW    4DC5H
720F 43                      DEFB    'C'
7210 D145                    DEFW    45D1H
                    ;
7212 A64D                    DEFW    4DA6H
7214 03                      DEFB    3
7215 00                      NOP             ;DELETE CALL TO ASTERISK
7216 00                      NOP             ;  FLASH ROUTINE.  (BUILT
7217 00                      NOP             ;  INTO MODEL III ROM)
                    ;
7218 D145                    DEFW    45D1H ;CASSETTE CONTROL PATCHES
721A 25                      DEFB    37
721B CD4230                  CALL    SETCAS ; 'C' COMMAND
721E 3A1142                  LD      A,(4211H)
7221 32EF45                  LD      (45EFH),A ;SAVE NEW BAUD RATE
7224 C9                      RET
7225 CDF801                  CALL    CSOFF ;WARM START INIT PATCH
7228 3AEF45                  LD      A,(45EFH) ;RESTORE BAUD RATE
722B 321142                  LD      (4211H),A
722E C9                      RET
722F CDD445                  CALL    45D4H ;SAVE CURRENT BAUD
7232 AF                      XOR     A
7233 321142                  LD      (4211H),A ;SET LOW BAUD FOR OBJ TAPE
7236 C3254F                  JP      4F25H
7239 0D                      DEFB    0DH ;BAUD RATE TEMP STORAGE
723A CDF801                  CALL    CSOFF ;PATCH TO GIVE FASTER CSOFF FOR
723D C32B47                  JP      472BH ;  1500 BAUD OPERATION.
                    ;
7240                VAREND   EQU     $
                    ;
                    ;DISPLAY MSG AND WAIT FOR KBD ENTRY.
                    ;     <BREAK> CAUSES RESTART.
                    ;
7240 CD1B02         MSGOUT   CALL    VDLINE
7243 CD4900                  CALL    KBWAIT
7246 3D                      DEC     A
7247 C0                      RET     NZ
                    ;
7248 310070         START    LD      SP,ORGN
724B 214872                  LD      HL,START
724E 220442                  LD      (4204H),HL ;SET UP BREAK PROCESSING
                    ;
                    ;LOAD EDTASM AT 500 BAUD
                    ;
7251 210470                  LD      HL,RDMSG
7254 CD4072                  CALL    MSGOUT
7257 11F645                  LD      DE,45F6H   ;WON'T LOAD ANYTHING BELOW HERE
725A 210044                  LD      HL,4400H
725D 3600           ZERO     LD      (HL),0     ;ZERO OUT PROGRAM AREA
725F 23                      INC     HL
7260 7C                      LD      A,H
7261 FE60                    CP      60H
7263 38F8                    JR      C,ZERO
                    ;
7265 AF                      XOR     A
7266 321142                  LD      (4211H),A ;SET LOW BAUD RATE
7269 CD9602                  CALL    CSHIN
726C CD3502                  CALL    CSIN
726F FE55                    CP      'U'
```

Program continued

```
7271 203F      03580          JR      NZ,LDERR
7273 210B70    03590          LD      HL,EDNAME
7276 0606      03600          LD      B,6
7278 CD3502    03610 NMCK     CALL    CSIN     ;CHECK PROGRAM NAME
727B BE        03620          CP      (HL)
727C 2034      03630          JR      NZ,LDERR
727E 23        03640          INC     HL
727F 10F7      03650          DJNZ    NMCK
               03660 ;
7281 CD3502    03670 NXTBLK   CALL    CSIN
7284 FE78      03680          CP      78H      ;END RECORD?
7286 2834      03690          JR      Z,LDEND
7288 FE3C      03700          CP      3CH      ;DATA RECORD?
728A 2026      03710          JR      NZ,LDERR
728C CD3502    03720          CALL    CSIN
728F 47        03730          LD      B,A      ;BLOCK SIZE
7290 CD3502    03740          CALL    CSIN
7293 6F        03750          LD      L,A      ;LSB OF BLOCK START ADR
7294 CD3502    03760          CALL    CSIN
7297 67        03770          LD      H,A      ;MSB OF BLOCK START ADR
7298 85        03780          ADD     A,L
7299 4F        03790          LD      C,A      ;START CHECKSUM
               03800 ;
729A CD3502    03810 NXTBYT   CALL    CSIN
729D 320070    03820          LD      (CSAVE),A
72A0 81        03830          ADD     A,C      ;UPDATE CHECKSUM
72A1 4F        03840          LD      C,A
72A2 DF        03850          RST     18H      ;THIS ROM CALL COMPARES HL:DE.
               03860 ;                CARRY SET IF HL<DE; ZERO SET IF HL=DE.
72A3 3804      03870          JR      C,NOLOAD ;BELOW MIN LOAD ADDR.
72A5 3A0070    03880          LD      A,(CSAVE)
72A8 77        03890          LD      (HL),A
72A9 23        03900 NOLOAD   INC     HL
72AA 10EE      03910          DJNZ    NXTBYT
72AC CD3502    03920          CALL    CSIN     ;GET CHECKSUM
72AF B9        03930          CP      C
72B0 28CF      03940          JR      Z,NXTBLK
               03950 ;
72B2 3E45      03960 LDERR    LD      A,'E'
72B4 323E3C    03970          LD      (3C3EH),A   ;REPLACE LEFT ASTERISK WITH 'E'
72B7 CDF801    03980          CALL    CSOFF
72BA 188C      03990          JR      START
               04000 ;
72BC CD3502    04010 LDEND    CALL    CSIN     ;READ EX ADR & DISCARD
72BF CD3502    04020          CALL    CSIN
72C2 CDF801    04030          CALL    CSOFF
               04040 ;
               04050 ;CHECK FOR CORRECT VERSION  (MUST BE 1.2).
               04060 ;
72C5 3AE148    04070          LD      A,(48E1H)
72C8 FE31      04080          CP      '1'
72CA 2007      04090          JR      NZ,VERSER
72CC 3AE348    04100          LD      A,(48E3H)
72CF FEB2      04110          CP      '2'+80H
72D1 2809      04120          JR      Z,MOD44    ;VERSION CORRECT
72D3 213670    04130 VERSER   LD      HL,VMSG
72D6 CD1B02    04140          CALL    VDLINE
72D9 C34872    04150          JP      START
               04160 ;
               04170 ;
               04180 ;START EDTASM MODIFICATIONS
               04190 ;
               04200 ;MOD44 - THIS ROUTINE INSERTS A 44H WHERE REQUIRED
               04210 ;  BASED ON TABLE 'T44'
               04220 ;
72DC 214E70    04230 MOD44    LD      HL,T44
72DF 3E44      04240 MD1      LD      A,44H
72E1 56        04250          LD      D,(HL)   ;GET PAGE # FOR THIS SECTION
72E2 23        04260          INC     HL
72E3 46        04270          LD      B,(HL)   ;GET # OF BYTES TO DO
```

```
72E4 23                   INC     HL
72E5 5E           MD2     LD      E,(HL)  ;GET ADDR ON PAGE
72E6 23                   INC     HL
72E7 12                   LD      (DE),A
72E8 10FB                 DJNZ    MD2
72EA 114271               LD      DE,T44E
72ED DF                   RST     18H     ;HL-DE TO FLAGS
72EE 38EF                 JR      C,MD1   ;HL < DE
                  ;
                  ;
                  ;MODWDS - THIS ROUTINE INSERTS A SPECIFIED WORD AT A
                  ;  GIVEN ADDRESS.  TABLE 'WDTBL' IS USED.
                  ;
                  ;
72F0 5E           MODWDS  LD      E,(HL)  ;GET LSB OF WORD ADDRESS
72F1 23                   INC     HL
72F2 56                   LD      D,(HL)  ;GET MSB OF WORD ADDRESS
72F3 23                   INC     HL
72F4 7E                   LD      A,(HL)  ;GET LSB OF WORD TO BE INSERTED
72F5 23                   INC     HL
72F6 12                   LD      (DE),A  ;STORE LSB OF WORD
72F7 13                   INC     DE
72F8 7E                   LD      A,(HL)
72F9 23                   INC     HL
72FA 12                   LD      (DE),A
72FB 11B671               LD      DE,WDTBLE
72FE DF                   RST     18H
72FF 38EF                 JR      C,MODWDS
                  ;
                  ;MODVAR - THIS ROUTINE INSTALLS THE SPECIFIED PROGRAM
                  ;  CODE FROM TABLE 'VARTBL'.
                  ;
7301 5E           MODVAR  LD      E,(HL)
7302 23                   INC     HL
7303 56                   LD      D,(HL)
7304 23                   INC     HL
7305 46                   LD      B,(HL)
7306 23                   INC     HL
7307 7E           MDV1    LD      A,(HL)
7308 23                   INC     HL
7309 12                   LD      (DE),A
730A 13                   INC     DE
730B 10FA                 DJNZ    MDV1
730D 114072               LD      DE,VAREND
7310 DF                   RST     18H
7311 38EE                 JR      C,MODVAR
                  ;
                  ;WRITE EDTASM AT 1500 BAUD
7313 211B70       WRITE   LD      HL,WRMSG
7316 CD4072               CALL    MSGOUT
7319 11F05C               LD      DE,5CF0H ;END OF EDTASM
731C 321142               LD      (4211H),A
731F CD8702               CALL    CSHWR
7322 3E55                 LD      A,'U'
7324 CD6402               CALL    CSOUT
7327 210B70               LD      HL,EDNAME
732A 0606                 LD      B,6
732C 7E           WRNM    LD      A,(HL)
732D 23                   INC     HL
732E CD6402               CALL    CSOUT
7331 10F9                 DJNZ    WRNM
7333 21D045               LD      HL,45D0H ;START ADR
7336 3E3C         WRBLK   LD      A,3CH ;DATA BLOCK
7338 CD6402               CALL    CSOUT
733B AF                   XOR     A
733C 47                   LD      B,A
733D CD6402               CALL    CSOUT
7340 7D                   LD      A,L
7341 CD6402               CALL    CSOUT
7344 7C                   LD      A,H
```

Program continued

```
7345 CD6402    04980          CALL    CSOUT
7348 85        04990          ADD     A,L
7349 4F        05000          LD      C,A
734A 7E        05010 WRBYT    LD      A,(HL)
734B CD6402    05020          CALL    CSOUT
734E 81        05030          ADD     A,C
734F 4F        05040          LD      C,A
7350 23        05050          INC     HL
7351 10F7      05060          DJNZ    WRBYT
7353 79        05070          LD      A,C
7354 CD6402    05080          CALL    CSOUT
7357 DF        05090          RST     18H
7358 38DC      05100          JR      C,WRBLK
735A 210170    05110          LD      HL,ENDBLK
735D 0603      05120          LD      B,3
735F 7E        05130 WREND    LD      A,(HL)
7360 23        05140          INC     HL
7361 CD6402    05150          CALL    CSOUT
7364 10F9      05160          DJNZ    WREND
7366 CDF801    05170          CALL    CSOFF
7369 18A8      05180          JR      WRITE
               05190 ;
7248           05200          END     START
00000 TOTAL ERRORS
```

Renumber One

by Dr. Stephen Mills

If you use a TRS-80 primarily for self-education and entertainment, chances are you have a Level I system. If your expectations aren't too demanding, and you can live without Level II's extra number-crunching, string-untangling functions, you're probably satisfied. But if you're honest, you'll have to admit that there are times when you envy the realm of the classier systems. The one thing I envied most about Level II was its line-renumbering software. Radio Shack supplied it, and it seemed that every software publisher who marketed a utility program had a line renumbering program. But no one provided renumbering programs for Level I.

For me, the main attraction of such a program was mostly aesthetic. My raggedly numbered lines of BASIC were a bit embarrassing. Naturally, I felt creative pride in my programs, but was a little ashamed of exposing their secret workings. I felt as though I had dressed my creations in tuxedos with tattered underwear and holey socks. But, I admit that there are better reasons for wanting a renumbering utility in Level I. The processes of debugging and amplifying programs sometimes fill the spaces between initially well-separated lines of BASIC. A renumbering program gives you more room. Also, a neatly arranged final version is desirable if you're considering publishing your work. It not only looks more professional, but also aids typists and transcribers, because errors are easier to detect within a regular and predictable pattern of line numbers. Finally, in combination with other utilities, a renumbering program facilitates modular programming. Tested and effective subroutines can be sorted and reintegrated into other programs, increasing programming efficiency.

So, when I began expanding my Level I BASIC programming with assembly-language flexibility, a renumbering utility was a high priority. This utility should be of interest to Level I users with a penchant for neatness. You will need a monitor such as the Editor/Assembler or T-BUG to do this. The program is written for 16K Level I, but a table of modifications for 4K (Table 4) is included, and applications for Level II and Model III users are discussed at the end of the article. NUMBR 1 is both simple to use and effective. It corrects all GOTO, GOSUB, and THEN addresses to the new numbers, and it also corrects multi-address ON-GOTO and ON-GOSUB statements. Standard Level I abbreviations are accepted and processed, as are unofficial, but functional, phrases like GOT. or TH. NUMBR 1 isn't disturbed by spacing between the statement and its address (as in GOTO 100), or by statements which address nonexistent lines (deleted lines). But, it does not accept a numbering pattern which generates negative numbers or numbers over 32767. Finally, it operates quickly on a BASIC program already in memory without requiring an initial CSAVE, but it does permit you to CLOAD the BASIC program while it is running.

Structure of NUMBR 1

The operation of NUMBR 1 can be more easily understood if you know something of its development. Once I had the logic of the renumbering program settled, there were two structural decisions to be made. First, NUMBR 1 could take either of two general tape formats:

1) NUMBR 1 could be operated via T-BUG (250 baud) or Radio Shack's SYSTEM tape (a tape supplied with EDTASM that allows the Level I to load 500 baud, Level II SYSTEM tapes). The BASIC program to be renumbered would then have to be CLOADed after NUMBR 1 is loaded.
2) NUMBR 1 could be loaded and used on a BASIC program already in memory.

Option 1 is the easiest to compose and to explain, but it is much more cumbersome to execute. It requires four cassette operations for every renumbering: CSAVE the BASIC program, CLOAD T-BUG or SYSTEM, load NUMBR 1, and then CLOAD the BASIC program again. The first and last of these operations are necessary because T-BUG and SYSTEM compete with BASIC programming for specific regions of RAM. This fact makes it impossible to operate NUMBR 1 under T-BUG or SYSTEM. Although NUMBR 1 was originally designed this way, once it was debugged, I chose option 2. This format is a breeze to use but, unfortunately, more difficult to explain. You cannot produce NUMBR 1 directly with T-BUG or with Editor/Assembler. As a result, the code presented for NUMBR 1 is actually a parent program, containing the source code for the renumbering utility. The purpose of the parent program is to beget lots of little renumber programs. Although the logic of NUMBR 1 can be understood from the listing, some relocation and self-modification takes place when the parent program runs.

The second format question concerned what part of memory NUMBR 1 would occupy. In Level I, there is a section of memory below address 4200H available for machine-language programming during the command mode. Here NUMBR 1 would not interfere with BASIC variables, or with BASIC program text. A second advantage is that a single version of NUMBR 1 can be written for any size RAM. Unfortunately, NUMBR 1 is too long to fit into this small, protected area. This narrowed the possibilities to two:

1) Load part of NUMBR 1 outside the protected area
2) Successively load and execute different parts of NUMBR 1 in the same area

I tried option 2 but abandoned it. It is cumbersome and susceptible to

110	Checksum error test for part one of NUMBR 1
120	Determine top of available memory (same for 4K)
120–150	Load end-of-program in DE register, and make sure that a program is present
160–220	If no BASIC is present, execution pauses for CLOAD
230–270	The RSD routine insures that there is enough space available to load part two of NUMBR 1.
280–300	If not enough memory, return to BASIC command mode
410–420	Load part two of NUMBR1 and do checksum test
430–450	Print opening title and jump to entry point of part two of NUMBR1
780–1050	ZAP routine generates the Level I format cassette
780	Moves the stack to a safe place
790–820	Prepare to move part one into place and calculate the checksum adjustment
830–870	Block move part one and insert checksum modifier
860	Subtract MSB of entry address to produce 40H checksum (CLOAD will automatically insert this at address 41FFH)
890–960	250-baud output loop, leaves cassette running
970	Set up delay
980–1050	PSE routine generates extra leader between dumps and looks for BREAK
1350–1370	Statement pointers to Level I ROM
2460–2560	Input for number of first line and increment (default procedure uses 100 and 10 respectively)
2570–2740	Make sure input values will work
2750–2800	Successive processing of three BASIC statements, THEN, GOTO and OSUB
2810–3020	Do actual line renumbering
3000–3010	(Program modifies itself here to load the contents of the IX register into the address pointed to by HL)
3030	The RNTHRU routine begins here; address pointer for statement being processed is set in IX
3040–3090	BASIC source code is moved into high memory
3180–3300	Test to determine if all of program has been processed, by comparing HL to top of memory
3310–3320	Shift bytes back to low memory while testing
3330–3340	Test for end of BASIC line (0DH)
3350–3400	If byte marks end of line, program line pointer is updated, and line number bytes are shifted without further processing
3410–3420	Compare byte just moved to statement being processed; if no match proceed to 3180
3430–3440	String comparison loop. Test BASIC statement pointer to determine if a full string match has been made (for Level I ROM this is indicated by bit 7 in the (IX + 1) location). If match, proceed to 3520
3450–3480	A test for double letters, e.g., IFS = TTHEN . . ., allowed in Level I
3490–3500	Test succeeding byte for period, indicating an abbreviation. If none, continue string comparison
3520–3540	Test for blank spaces between statement and number
3550–3590	Decode address in old BASIC into hexadecimal
3600–3770	Find line number and determine its new value, convert value to ASCII and insert in program
3780–3830	Test for comma (for ON GOTO and ON GOSUB)

Table 1. *Summary of NUMBR 1 by line number*

problems. Even if the variables are sacrificed, 2 requires breaking the program into at least four separately loaded segments. But using 1, with NUMBR 1, as written, requires a 16K memory and at least 549 bytes of free memory (PRINT MEM with the BASIC program in place must yield a number greater than 548). Table 4 lists the modifications needed for 4K operation, but the 549-byte requirement remains. With this format, part of NUMBR 1 still loads into the safe area. This is necessary to make the program automatic, and it also reduces the demand for free memory.

The Parent Program

The first-generation, or parent program, cannot do any renumbering. Its purpose is merely to output to the cassette a second-generation program which *will* renumber. If the code is transcribed through an Editor/Assembler, the object code is designed to start at the ZAP label (the Z-80 Assembly Parent). The Editor/Assembler produces a 500-baud recording of the parent, which is executed via the SYSTEM tape. You might want to write out the listing for easy correction later on. If the code is entered through T-BUG, it is necessary to transcribe it into two locations, the first beginning at 44F0H and the second at 7DDBH. Before running the parent program, you should save what you have transcribed. Otherwise, an undetected bug could gobble up your work, and you'll have to start from scratch. To save your transcription, you must do two Punches:

```
P 44F0 4642
P 7DDB 8000 [P 4DDB 5000 for 4K systems]
```

To run the program, prepare a fresh cassette and type: J 45FE.

My practice with this and other assembly-language utilities is to wind a C-30 or C-45 cassette to the midway point, dumping the parent program and the EDTASM listing. Then I rewind to the start of the tape for the 250-baud, second-generation dump. I allow as many dumps as possible, until the recording approaches the middle of the tape. Because the utility is so short, it is useful to have many recordings of it for quick execution. This also avoids frequent rewinding which may wear out a small segment of tape.

The cassette-output segment of the parent program is written as a continuous loop to facilitate multiple recordings. After both parts of NUMBR 1 are dumped, the parent program runs the recorder for a moment, to provide extra leader, and then starts outputting the program again. When you have enough recordings, simply hold down the BREAK key. This terminates the process as soon as the dump in progress is completed. Control returns to the BASIC command mode.

The Logic of NUMBR 1

The Program Listing and Tables should suffice for those curious about the details of NUMBR 1. Remember, though, that the working version of

LABEL	LINE # #	Function
BCOUNT	1380–1470	Loads active BC register pair with difference between HL and DE
BUILD	3840–4040	Search source code for the line number stored in DE, while renumbered value is calculated in IX register. When DE is matched, FND is executed
CKSUM	670	Place where checksum adjustment will be stored before part one of NUMBR1 is dumped (41ECH in this program unless modified)
DESAVE	1920	Address where contents of DE are temporarily stored during renumbering
DIF	770	The value calculated by the Editor/Assembler used to modify direct addresses of par one after relocation
FND	4050–4090	Conclusion of BUILD, where matched line number is converted to ASCII and stored in new object code
INCV	1980–2390	Converts ASCII string to two-byte hexadecimal in HL. Terminates when nonnumeric symbol is met or overflow occurs
INDEX	3170	Stores pointer address of BASIC statement being processed
INPUTC	1070–1290	Keyboard input routine using ROM calls but protecting program execution from CLEAR key and excessive backspacing. Evaluates first five non-blank characters input. INPUTC is destroyed during program execution. The label is also used by part one calculations because it is the first item in part two.
MAINE	3350	Entry point for MAINLN loop
MAINLN	3180ff.	Loop in which renumbering algorithm is executed
NEXT20	3520ff.	A statement match has been found. Program ignores any blanks between statement and address, then converts address to hexadecimal. Line number addressed is located, and its new number is calculated. Value is converted back to ASCII and stored in new program code
NPLACE	1480–1580	Performs machine numbering into ASCII decimal by subtraction. Stores result in buffer pointed to by BC. Used repeatedly by OUTCV
OUTCV	1590–1970	Converts two-byte hexadecimal in HL to a decimal ASCII string. 1860–1880 skips leading zeros in setting buffer
PSE	980–1050	During cassette output writes extra leader between separate programs, and awaits termination command
STUMBL	3360	Location where current BASIC line number is stored
TIDY	2400–2450	Test to insure proper input for line numbers and increments
ZAP	780–1050	Executed part of first-generation program, which generates the second-generation renumbering utility. ZAP relocates part one of program, adjusts the checksum value and makes multiple cassette recordings of two-part NUMBR1

Table 2. *Some important parts of NUMBR 1 by label name*

NUMBR 1 involves only lines 100–660 (which are relocated first) and lines 1060–4090. The following discussion focuses on some of the more difficult features of the program.

The touchy part of a renumbering program is not the actual line renumbering, but the correction of GOTOs and GOSUBs for the new numbering system. The algorithm for this is the MAINLN loop (starting at line 3180), which is initialized for each BASIC statement by RNTHRU (3030–3160). MAINLN tests each byte of the BASIC code as it is shifted into place. This procedure, flowcharted in Figure 1: (1) checks for the end of program and (2) checks for the end of a line, while (3) shifting one byte, which (4) it compares to the BASIC statement. This loop continues until there is a match at (4), in which case a secondary loop is executed. Here, the BASIC statement pointer advances to the next letter until (5) a complete match of the statement is found, (6) a period, signalling an abbreviation, is found, or (7) a mismatch occurs. If there is a mismatch, the BASIC statement pointer is reset (0). Otherwise, the program proceeds to process any addresses which follow the recognized BASIC statements.

If your entries generate numbers which are too high for Level I BASIC, the program will restart. If you press ENTER in response to the input queries, a default procedure sets the first line at 100 and the increment at 10. Program execution time varies, depending on the length of the program and the number of addresses to be converted. With short programs, it seems almost instantaneous, but allow up to a minute for very long ones. Control returns automatically to the command mode, and LIST reveals the renumbered program.

If for some reason (possibly deliberate, in incomplete or modular programs), a line is addressed which does not actually exist, the old address is deleted. For example:

```
ON Z GOSUB 100 , , ,
```

might result if the second, third, and fourth addresses could not be found in the original program. Any spaces originally placed between the statement and address remain. If an operation, rather than an address, follows an occurrence of THEN, it is not affected. Finally, note that NUMBR 1 takes no heed of quotation marks or REMarks. I thought this preferable since, for example, a REM statement might mention GOTO 253, and I would prefer having that changed, too. Occasionally, this may require some modification (e.g., PRINT"IF 2 OF US LEAVE THEN 1 WILL REMAIN", which contains a nonstatement occurrence of THEN, would be converted).

Parts of NUMBR 1 may hold some interest for programmers with systems other than Level I. As a renumbering program, it is adaptable to Level II, but it is probably not the most expedient technique, since Level II does not have to contend with string searches and abbreviations. But, the algorithm for the Z-80 could be applied to other functions. To facilitate any adaptation for the Model III, I have included Table III, which contains every use of ROM and dedicated RAM utilized by NUMBR 1. Modifications, or substitutions of equivalent functions in other formats, may be required.

Level I programmers interested in successively loading and executing different parts of NUMBR 1 in the same memory area, should note the following features of Level I dedicated RAM: video memory, the memory actually

Address	Function
0EEF	Reentry point for Level I command mode, used by ROM after a bad CLOAD. Displays WHAT? message automatically.
406C	RAM location where end of BASIC program is indicated. It contains the address of the carriage return of the last line + 1.
4200	This is (a) the RAM address where BASIC programs are stored, and (b) the value assigned to the SP register. The first value placed on the stack by a CALL instruction goes into 41FE–41FF.
01C9	Reentry point for command mode of Level I ROM.
0EF4	ROM's CLOAD function, inputs program or other data from cassette.
0010	Routine in ROM to display contents of A register on video.
3840	This actually addresses the keyboard. If ENTER, BREAK or CLEAR is pressed, a value will be at this address.
41FE–41FF	Address, actually occurring in the stack area, which must receive CLOAD input in order to take control away from Level I ROM. After a CLOAD, program will continue at the address stored here.
0FE9	Turns on the cassette relay.
0F4B	Level I ROM's CSAVE function. Will record contents of memory between HL and DE—1.
7FFF	Last byte of memory with 16K. Value is 4FFF for 4K.
3801	Keyboard memory, addresses the C key.
4068	RAM location where position of video cursor is stored.
0B40	ROM routine which scans for keyboard input, and displays the result automatically (including CLEAR and ENTER).
0020	Subroutine which compares the contents of DE and HL.
0028	Subroutine which checks the DE address for a blank (20H), and increments DE until a non-blank character is found.
094F	ROM subroutine which displays an extended message on video. DE points to the message and terminates when a carriage return (13H) is met, or byte = contents of B (which is 0).
0287	The word string GOTO occurs at this ROM location.
028E	The string OSUB begins at this location. The first letter is skipped for reasons explained in the text.
0338	The word THEN occurs here.
406A	RAM location where ROM stores the address of the highest byte of memory available, i.e., 7FFF for a 16K TRS-80.

Table 3. *Table of references to Level I ROM and dedicated RAM*

represented on the CRT, occupies 3C00H to 3FFFH. No machine-language code can be written here, since this memory is not a full eight-bit memory. But it is possible to load text directly from cassette to screen, as the Microchess program does. Following video memory, bytes 4000H to 4067H are dedicated to the variables A through Z, available to the Level I user. It is possible to pack programming into this area at the expense of the variables. But, the Level I cassette input technique stores one more byte than output normally writes. This means that your code only goes through address 4066H, because the block from 4068H to 406DH is vital to the Level I interpreter and cannot usually be written over (see Table 3). Addresses 406EH to

408FH are dedicated to the string variables A$ and B$. They are expendable, but 4090H should be left alone, since this is the reference point for ROM's cassette input and output operations. After that, 4091H to about 41F5H are available; a few bytes between 41F5H and 41FFH should be left open for stack operations. This leaves the Level I programmer with about 500 bytes available below BASIC storage, provided certain critical addresses are avoided.

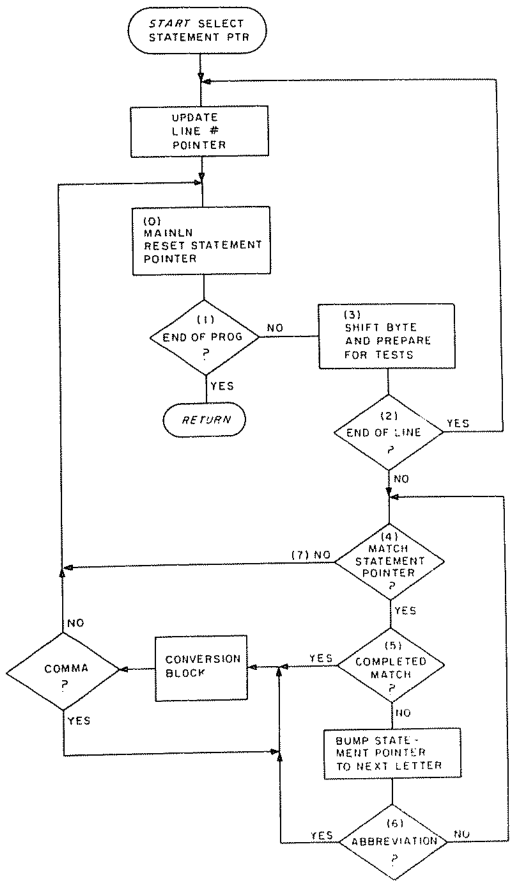

START SELECT STATEMENT PTR
UPDATE LINE # POINTER
(0) MAINLN RESET STATEMENT POINTER
(1) END OF PROG ?
NO
YES
RETURN
(3) SHIFT BYTE AND PREPARE FOR TESTS
(2) END OF LINE ?
YES
NO
(4) MATCH STATEMENT POINTER ?
(7) NO
YES
(5) COMPLETED MATCH ?
YES
NO
CONVERSION BLOCK
COMMA ?
NO
YES
BUMP STATEMENT POINTER TO NEXT LETTER
(6) ABBREVIATION ?
YES
NO

Figure 1. *Flowchart of MAINLN subroutine. Numbers in parentheses refer to discussion in text.*

The method used for step 1, which checks for the end of the program, is comparatively slow and lengthy (19 bytes). This is to allow for simple modification for hybrid programming. I don't expect that the length or speed will really trouble anyone.

The commands GOSUB, GOTO, and THEN do not actually occur in the program itself. Instead, I have referred to their addresses in Level I ROM.

For 4K machines, the instruction addresses from line 01060 will be different. The encoding must start at 4DDBH instead of 7DDBH. However, locations will be quite easy to read since only the first numeral will be changed. The following are modifications in the instruction code itself for LD, CALL and JP instructions:

Line number	New code	Line number	New code
240	21DB4D	3030	22544F
520	C3AB4E	3060	CD0B4E
930	21DB4D	3070	11094E
940	11FF4F	3110	CD704F
1450	CD0B4E	3550	ED53634E
1960	ED53634E	3560	CD6D4E
2480	CDDB4D	3620	DD2A124F
2490	CDA34E	3640	EDSB634E
2510	22124F	3650	CD0B4E
2530	CDDB4D	3680	CDDC4F
2540	CDA34E	3740	11094E
2560	220E4F	3750	CD0B4E
2570	ED5B124F	3770	CDD74F
2620	CD0B4E	3790	CD5B634E
2720	CDA34E	3820	C2524F
2760	CD334F	3910	22E84F
2780	CD334F	4010	ED4B0E4F
2790	CD334F	4070	CD254E
2830	CD0B4E		
2940	222F4F		

Table 4. *List of instruction changes for 4K systems*

The program recognizes a complete string match when a succeeding byte has bit 7 = 1 (see line 3430). Furthermore, the GOSUB pointer actually points to the O rather than the G. The reason for this is that, since the GOTO string is processed first, any occurrence of G will already have been converted by the MAINLN algorithm. If the GOSUB pointer addressed the initial G, every occurrence of G would put the search into the string test subloop. Then, the period-match at step 6 would cause NUMBR 1 to attempt a reconversion of the addresses which follow, with erroneous results.

After an address conversion has been made (3810–3830), ON-GOTO and ON-GOSUB are converted by checking the source code for a comma. Depending on the results, the program branches either to the MAINLN loop or to the conversion section. If you have a long program, numbered with small numbers, and you want to renumber it with higher numbers, the length of the program will increase by renumbering. As the Level I manual tells us, the line number itself always takes two bytes, whether it is 1 or 19999. But, GOTO 19999 does take more memory than GOTO 1. To allow for this without increasing its own memory requirements, NUMBR 1 permits part of itself to be obliterated during use (from 7DDBH to 7E09H, or 47 bytes in 16K).

Execution of NUMBR 1

After the second-generation renumbering utility is produced, using it is easy. When you have a program you want to renumber, simply insert the NUMBR 1 cassette and type CLOAD. If there is a problem with loading NUMBR 1, the monitor displays the message:

```
WHAT?
>READY_
```

Unlike a bad BASIC CLOAD, this will not cause the loss of your resident program. Should a loading error occur, wind the tape to a good recording and start over. If the load is okay, three things might happen:
1) If you have loaded NUMBR 1 without a BASIC program already in memory, NUMBR 1 will note that fact and display:

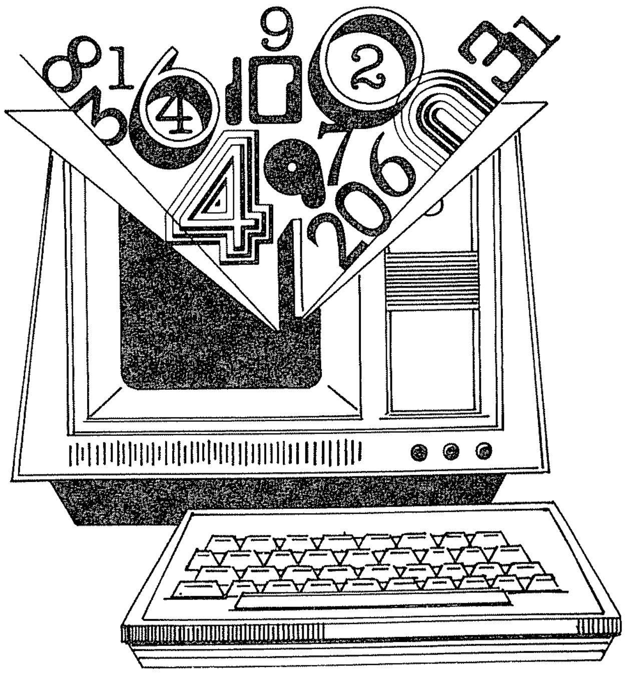
```
INSERT BASIC PROGRAM CASSETTE & PRESS C TO LOAD
```

When you have done so, if everything is okay, the monitor will read:

```
LOAD REST OF LEVEL I BASIC LINE RENUMBER
PRESS C TO CLOAD
```

and part two, the high memory section of NUMBR 1, will be loaded.
2) Second, if there is already a BASIC program in memory, but one which is too long to process, the monitor will display:

```
INSUFFICIENT MEMORY
```



3) Third, if there is a BASIC program in memory of manageable size, both parts of NUMBR 1 load and automatically start. The monitor displays:

```
LEVEL I LINE RENUMBER
FOR RESIDENT PROGRAM
FIRST LINE?
```

Here, type in the number to be assigned to the first line of code. When that is entered, the program solicits:

```
INCREMENT?
```

for which the step between lines will be entered.

Program Listing. *NUMBR 1*

```
               ;NUMBR1
               ;(C) 1981 STEPHEN MILLS
               ;LEVEL I LINE RENUMBER PROGRAM
               ;EDTASM VERSION PREPARES A MULTI-DUMP 2-PART
               ;250 BAUD TAPE FOR LEVEL I.  PART 1 LOADS IN
               ;PROTECTED AREA BELOW LEVEL I BASIC.  PART 2
               ;IS BOOTSTRAPPED INTO HIGH MEM (16K).
               ;
44F0                    ORG     44F0H
40F1           HEAD     DEFL    $-DIF
44F0 C2EF0E             JP      NZ,0EEFH
44F3 ED5B6C40           LD      DE,(406CH)
44F7 210042             LD      HL,4200H
44FA ED52               SBC     HL,DE
44FC 3814               JR      C,RSD
44FE 113B41             LD      DE,LDP-DIF
4501 CD8A41             CALL    REIN-DIF
4504 CDF40E             CALL    0EF4H
4507 20F5               JR      NZ,$-9
4509 226C40             LD      (406CH),HL
450C 119B41             LD      DE,RSMS-DIF
450F CD8A41             CALL    REIN-DIF
4512 110042    RSD      LD      DE,4200H
4515 21DB7D             LD      HL,INPUTC
4518 AF                 XOR     A
4519 ED52               SBC     HL,DE
451B 3053               JR      NC,BOOT
451D 112741             LD      DE,TOOMS-DIF
4520 CD4F09             CALL    PRINT
4523 C3C901             JP      01C9H
4526 49        TOOMS    DEFM    'INSUFFICIENT MEMORY'
4539 0D                 DEFB    13
453A 49        LDP      DEFM    'INSERT BASIC PROGRAM CASSETTE & '
455A 00                 NOP
415C           SETBSE   DEFL    $-DIF
455B 46                 DEFM    'FIRST LINE'
4565 00                 NOP
4167           SETINC   DEFL    $-DIF
4566 49                 DEFM    'INCREMENT'
456F 00                 NOP
4570 CDF40E    BOOT     CALL    0EF4H
4573 C2EF0E             JP      NZ,0EEFH
4576 3E0C               LD      A,12
4578 D7                 RST     10H
4579 3C                 INC     A
457A D7                 RST     10H
457B D7                 RST     10H
457C 11A841             LD      DE,TITLE-DIF
457F CD4F09             CALL    PRINT
4582 CD4F09             CALL    PRINT
4585 D7                 RST     10H
4586 C3AB7E             JP      ENTRY
4589 CD4F09    REIN     CALL    PRINT
458C 11D941             LD      DE,KYBM-DIF
458F CD4F09             CALL    PRINT
4592 3A0138             LD      A,(3801H)
4595 E608               AND     8
4597 28F9               JR      Z,$-5
4599 C9                 RET
459A 4C        RSMS     DEFM    'LOAD REST OF '
45A7 4C        TITLE    DEFM    'LEVEL I BASIC LINE RENUMBER'
45C2 0D                 DEFB    13
45C3 46                 DEFM    'FOR RESIDENT PROGRAM'
45D7 0D                 DEFB    13
45D8 50        KYBM     DEFM    'PRESS  C  TO CLOAD'
45EA 0D                 DEFB    13
45EB 0000      CKSUM    DEFW    0
45ED 0000               DEFW    0
```

```
45EF 0000       00690          DEFW    0
45F1 0000       00700          DEFW    0
45F3 0000       00710          DEFW    0
45F5 0000       00720          DEFW    0
45F7 0000       00730          DEFW    0
45F9 0000       00740          DEFW    0
45FB 0000       00750          DEFW    0
45FD F1         00760          DEFB    HEAD&0FFH
03FF            00770 DIF      DEFL    $-41FFH
45FE 31004D     00780 ZAP      LD      SP,4D00H
4601 010E01     00790          LD      BC,ZAP-44F0H
4604 21FD45     00800          LD      HL,ZAP-1
4607 11FE41     00810          LD      DE,41FEH
460A AF         00820          XOR     A
460B 96         00830          SUB     (HL)
460C EDA8       00840          LDD
460E EA0B46     00850          JP      PE,$-3
4611 D640       00860          SUB     40H
4613 32EC41     00870          LD      (CKSUM-DIF),A
4616 CDE90F     00880          CALL    0FE9H
4619 21F140     00890 DUMP     LD      HL,HEAD
461C 11FF41     00900          LD      DE,41FFH
461F CD4B0F     00910          CALL    0F4BH
4622 CDE90F     00920          CALL    0FE9H
4625 21DB7D     00930          LD      HL,INPUTC
4628 11FF7F     00940          LD      DE,7FFFH
462B CD4B0F     00950          CALL    0F4BH
462E CDE90F     00960          CALL    0FE9H
4631 210000     00970          LD      HL,0
4634 23         00980 PSE      INC     HL
4635 7C         00990          LD      A,H
4636 B5         01000          OR      L
4637 28E0       01010          JR      Z,DUMP
4639 3A4038     01020          LD      A,(3840H)
463C A7         01030          AND     A
463D 28F5       01040          JR      Z,PSE
463F C3C901     01050          JP      01C9H
7DDB            01060          ORG     7DDBH
7DDB CD4F09     01070 INPUTC   CALL    PRINT    ;PRINT WHAT DE POINTS TO
7DDE 3E3F       01080          LD      A,'?'
7DE0 D7         01090          RST     10H      ;DISPLAY ?
7DE1 ED5B6840   01100          LD      DE,(4068H)       ;GET CURSOR
7DE5 CD400B     01110 WTZ1     CALL    0B40H    ;INPUT ROUTINE
7DE8 2A6840     01120          LD      HL,(4068H)       ;CURRENT CURSOR
7DEB FE03       01130          CP      3        ;BREAK?
7DED CAC901     01140          JP      Z,01C9H
7DF0 FE0D       01150          CP      13       ;ENTER?
7DF2 2805       01160          JR      Z,PASSES
7DF4 E7         01170          RST     20H      ;ROM'S HL/DE COMPARE
7DF5 30EE       01180          JR      NC,WTZ1
7DF7 18E5       01190          JR      INPUTC+3
7DF9 EF         01200 PASSES   RST     28H      ;HUNTS FOR NO-BLANK VIDEO
7DFA 21E844     01210          LD      HL,OUTBUF
7DFD EB         01220          EX      DE,HL    ;GET WORK OFF SCREEN
7DFE D5         01230          PUSH    DE       ;SAVE ADDRESS
7DFF 010500     01240          LD      BC,5
7E02 EDB0       01250          LDIR    ;MOVE INPUT TO BUFFER
7E04 E1         01260          POP     HL       ;PUT IN HL & DE TO
7E05 E5         01270          PUSH    HL       ;PREPARE FOR INCV
7E06 D1         01280          POP     DE
7E07 1864       01290          JR      INCV
7E09 00         01300 TOP      NOP
7E0A 01         01310          DEFB    1
094F            01320 PRINT    EQU     094FH    ;ROM PRINT SUB
40E9            01330 OUTBUF   DEFL    HEAD-8   ;LOW 7-BYTE BUFFER
0000            01340 NN       EQU     0        ;DUMMY ADDRESS
0287            01350 GOTO     EQU     0287H    ;STRINGS
028E            01360 GOSUB    EQU     028EH    ;IN LEVEL I ROM
0338            01370 THEN     EQU     0338H    ;FOR COMPARISON
7E0B E5         01380 BCOUNT   PUSH    HL
```

Program continued

```
7E0C B7                  OR      A          ;CLEAR CARRY
7E0D ED52                SBC     HL,DE
7E0F E3                  EX      (SP),HL
7E10 C1                  POP     BC         ;PLACE COUNT
7E11 D0                  RET     NC         ;RETURN IF HL HIGHER
7E12 EB                  EX      DE,HL      ;EXCHANGE OTHERWISE
7E13 CD0B7E              CALL    BCOUNT     ;DO IT THE OTHER WAY
7E16 EB                  EX      DE,HL      ;RESTORE REGISTERS
7E17 C9                  RET
7E18 AF          NPLACE  XOR     A          ;ZERO ACCUM
7E19 3C                  INC     A          ;RAISE IT
7E1A ED52                SBC     HL,DE      ;CARRY CLEAR FROM ABOVE
7E1C 30FB                JR      NC,NPLACE+1         ;REPEAT IF DE HASN'T
                         ;EXHAUSTED HL
7E1E 19                  ADD     HL,DE      ;RESTORE FROM NEGATIVE
7E1F 3D                  DEC     A          ;ADJUST DOWN 1
7E20 C630                ADD     A,30H      ;MAKE ASCII
7E22 02                  LD      (BC),A     ;BC HAS BUFFER POINTER
7E23 03                  INC     BC         ;ADJUST TO NEXT PLACE
7E24 C9                  RET
7E25 E5          OUTCV   PUSH    HL         ;SAVE # TO BE CONVERTED
7E26 21E940              LD      HL,OUTBUF           ;BUFFER ADDRESS
7E29 E5                  PUSH    HL
7E2A 0605                LD      B,5        ;COUNT
7E2C 3630                LD      (HL),30H            ;INIT BUFFER
7E2E 23                  INC     HL
7E2F 10FB                DJNZ    $-3
7E31 3620                LD      (HL),20H
7E33 23                  INC     HL         ;LOAD BLANK & STOP
7E34 3600                LD      (HL),0
7E36 C1                  POP     BC         ;RESTORE ADDRESS IN BC
7E37 E1                  POP     HL         ;RESTORE #
7E38 C5                  PUSH    BC         ;SAVE BUF ADR AGAIN
7E39 111027              LD      DE,10000            ;5TH PLACE
7E3C CD187E              CALL    NPLACE
7E3F 11E803              LD      DE,1000
7E42 CD187E              CALL    NPLACE
7E45 116400              LD      DE,100
7E48 CD187E              CALL    NPLACE
7E4B 110A00              LD      DE,10
7E4E CD187E              CALL    NPLACE
7E51 3E30                LD      A,30H      ;FOR UNIT PLACE
7E53 F5                  PUSH    AF         ;SAVE A BRIEFLY
7E54 85                  ADD     A,L        ;MAKE ASCII
7E55 02                  LD      (BC),A
7E56 F1                  POP     AF         ;RESTORE 30H
7E57 E1                  POP     HL         ;RESTORE BUFFER ADDR
7E58 010500              LD      BC,5       ;MAXIMUM MOVE
7E5B BE                  CP      (HL)       ;LOOK FOR NON-ZERO
7E5C 2004                JR      NZ,$+6
7E5E 23                  INC     HL
7E5F 0D                  DEC     C          ;ADJ MOVE COUNT
7E60 20F9                JR      NZ,$-5
7E63             DESAVE  EQU     $+1
7E62 110000              LD      DE,NN      ;WILL BE DESAVE LATER
7E65 C8                  RET     Z          ;RETURN IF ZERO CALC
7E66 EDB0                LDIR       ;PUT AT DE'S ADDRESS
7E68 ED53637E            LD      (DESAVE),DE
7E6C C9                  RET
7E6D D5          INCV    PUSH    DE         ;SAVE DEST PTR
7E6E AF                  XOR     A          ;CLEARS THIS ACCUM
7E6F 08                  EX      AF,AF'     ;MAKE IT ALT COUNTER
7E70 08          DEASC   EX      AF,AF'     ;COUNT LOOP
7E71 3C                  INC     A          ;RAISE 1
7E72 08                  EX      AF,AF'     ;RET TO INPUT ACCUM
7E73 3E30                LD      A,30H
7E75 AE                  XOR     (HL)       ;DEASC TEXT
7E76 FE0A                CP      10         ;MUST BE LESS THAN 10
7E78 3005                JR      NC,PROCES
7E7A 77                  LD      (HL),A     ;DOES NOTHING FOR INPUTC
```

```
7E7B EDA0                LDI               ;WHICH HAS DE=HL
7E7D 18F1                JR     DEASC      ;KEEP GOING
7E7F 08         PROCES   EX     AF,AF'     ;SWITCH TO COUNTER
7E80 D1                  POP    DE         ;RESTORE POINTER
7E81 D9                  EXX    ;MAKE IT ALTERNATE
7E82 210000              LD     HL,0
7E85 4F                  LD     C,A        ;BUT IT IN BC
7E86 0600                LD     B,0
7E88 D9         BUMP     EXX    ;GO TO POINTER DE
7E89 1A                  LD     A,(DE)     ;GET DIGIT FROM BUFFER
7E8A 13                  INC    DE         ;BUMP POINTER
7E8B D9                  EXX    ;BACK TO OTHER REGS
7E8C 0D                  DEC    C          ;LOWER COUNT
7E8D C8                  RET    Z          ;END IF ZERO
7E8E E5                  PUSH   HL         ;SAVE CURRENT SUM
7E8F 6F                  LD     L,A        ;BUT CURRENT DIGIT IN L
7E90 2600                LD     H,0
7E92 79                  LD     A,C        ;GET PLACE COUNT
7E93 1806                JR     LAST1
7E95 29         SHIFT    ADD    HL,HL
7E96 E5                  PUSH   HL         ;SAVE DOUBLE
7E97 29                  ADD    HL,HL
7E98 29                  ADD    HL,HL
7E99 D1                  POP    DE
7E9A 19                  ADD    HL,DE      ;NOW HL*10
7E9B 3D         LAST1    DEC    A          ;MORE?
7E9C 20F7                JR     NZ,SHIFT
7E9E D1                  POP    DE         ;GETS FORMER SUM
7E9F 19                  ADD    HL,DE
7EA0 D8                  RET    C          ;RETURN IF OVERFLOW
7EA1 18E5                JR     BUMP
7EA3 3806       TIDY     JR     C,ENTRY ;RESTART IF CARRY
7EA5 7C                  LD     A,H        ;GET H FOR
7EA6 07                  RLCA   ;ROTATE
7EA7 3802                JR     C,ENTRY ;RESTART IF NEG
7EA9 B5                  OR     L          ;OR IF ZERO
7EAA C9                  RET
7EAB 31FF41     ENTRY    LD     SP,41FFH
7EAE 115C41              LD     DE,SETBSE
7EB1 CDDB7D              CALL   INPUTC  ;INPUT BASE #
7EB4 CDA37E              CALL   TIDY    ;MAKE SURE GOOD INPUT
7EB7 2803                JR     Z,$+5   ;DEFAULT
7EB9 22127F              LD     (BASE),HL
7EBC 116741              LD     DE,SETINC
7EBF CDDB7D              CALL   INPUTC
7EC2 CDA37E              CALL   TIDY
7EC5 2803                JR     Z,$+5
7EC7 220E7F              LD     (INCMNT),HL
7ECA ED5B127F            LD     DE,(BASE)       ;GET BASE AMT
7ECE EB                  EX     DE,HL   ;SWITCH FOR TEST
7ECF D9                  EXX    ;ADD IN ALTERNATE REGS
7ED0 210242              LD     HL,4202H
7ED3 ED5B6C40            LD     DE,(406CH)      ;TOP OF MEM
7ED7 CD0B7E              CALL   BCOUNT  ;GET COUNT FOR TEST
7EDA 0B                  DEC    BC      ;NO NEED TO COUNT
7EDB 0B                  DEC    BC      ;LAST CAR RET
7EDC 3E0D       TESTLP   LD     A,13
7EDE EDB1                CPIR   ;TEST MEM
7EE0 E2EF7E              JP     PO,OKAY ;GO OUT IF OKAY
7EE3 EDA1                CPI
7EE5 EDA1                CPI    ;TAKES CARE OF LINE # LOCS
7EE7 D9                  EXX
7EE8 19                  ADD    HL,DE   ;MAKE SURE INPUTS
7EE9 CDA37E              CALL   TIDY    ;WILL FIT
7EEC D9                  EXX
7EED 18ED                JR     TESTLP
7EEF 218702     OKAY     LD     HL,GOTO ;LOOK FOR THIS COMMAND
7EF2 CD337F              CALL   RNTHRU
7EF5 218E02              LD     HL,GOSUB
7EF8 CD337F              CALL   RNTHRU
```

Program continued

```
7EFB 213803   02790         LD      HL,THEN
7EFE CD337F   02800         CALL    RNTHRU
7F01 210242   02810         LD      HL,4202H        ;START OF PROGRAM
7F04 ED5B6C40 02820         LD      DE,(406CH)      ;END OF PROGRAM
7F08 CD0B7E   02830         CALL    BCOUNT  ;GET COUNT
7F0B 0B       02840         DEC     BC      ;SKIP LAST BYTE
7F0C 0B       02850         DEC     BC
7F0E          02860 INCMNT  EQU     $+1     ;WILL BE LOADED LATER
7F0D 110A00   02870         LD      DE,10   ;RECOVER INCREMENT
7F12          02880 BASE    EQU     $+2     ;LOADED ABOVE
7F10 DD216400 02890         LD      IX,100
7F14 DD220042 02900         LD      (4200H),IX      ;SET FIRST LINE
7F18 3E0D     02910         LD      A,13
7F1A EDB1     02920 NUMBR   CPIR
7F1C C2C901   02930         JP      NZ,01C9H
7F1F 222F7F   02940         LD      (POKEPL),HL     ;SET OUT ADDRESS
7F22 EDA1     02950         CPI
7F24 EDA1     02960         CPI     ;SKIP TO START OF TEXT
7F26 CB78     02970         BIT     7,B     ;SEE IF CARRIED
7F28 C2C901   02980         JP      NZ,01C9H
7F2B DD19     02990         ADD     IX,DE   ;MAKE NEW LINE #
7F2F          03000 POKEPL  EQU     $+2
7F2D DD220000 03010         LD      (NN),IX ;LOAD NEW LINE #
7F31 18E7     03020         JR      NUMBR
7F33 22547F   03030 RNTHRU  LD      (INDEX),HL      ;STORE CMD REF
7F36 110042   03040         LD      DE,4200H
7F39 2A6C40   03050         LD      HL,(406CH)      ;END +1
7F3C CD0B7E   03060         CALL    BCOUNT
7F3F 11097E   03070         LD      DE,TOP
7F42 13       03080         INC     DE      ;MOVE 1 BYTE BEYOND
7F43 EDB8     03090         LDDR    ;BLOCK MOVE IT
7F45 EB       03100         EX      DE,HL   ;REVERSE FOR MOVE BACK
7F46 CD707F   03110         CALL    MAINE   ;MAINLN ENTRY POINT
7F49 3E0D     03120         LD      A,13    ;CAR RET
7F4B 12       03130         LD      (DE),A  ;MUST BE AT (DE)
7F4C 13       03140         INC     DE      ;MAKE 1 GREATER
7F4D ED536C40 03150         LD      (406CH),DE
7F51 C9       03160         RET
7F54          03170 INDEX   EQU     $+2     ;DEFINES INDEX LOC
7F52 DD210000 03180 MAINLN  LD      IX,NN   ;GET CMD ADDR IN IX
7F56 3E7E     03190         LD      A,TOP<-8
7F58 BC       03200         CP      H       ;TEST AGAINST H
7F59 3807     03210         JR      C,CL2
7F5B 200C     03220         JR      NZ,CL3
7F5D 3E09     03230         LD      A,TOP&0FFH
7F5F BD       03240         CP      L       ;TEST AGAINST LOW
7F60 3006     03250         JR      NC,CL3-1
7F62 1B       03260 CL2     DEC     DE      ;BACK UP DE PTR
7F63 1A       03270         LD      A,(DE)  ;IF MEM OVERSHOT
7F64 FE0D     03280         CP      13
7F66 20FA     03290         JR      NZ,CL2
7F68 C8       03300         RET     Z
7F69 7E       03310 CL3     LD      A,(HL)  ;GET NEXT BYTE TRANSFERRED
7F6A EDA0     03320         LDI     ;FOR TEST
7F6C FE0D     03330         CP      13
7F6E 200D     03340         JR      NZ,NEXT10
7F70 22757F   03350 MAINE   LD      (STUMBL),HL
7F75          03360 STUMBL  DEFL    $+2     ;GETS LINE # IN IY
7F73 FD2A0042 03370         LD      IY,(4200H)
7F77 EDA0     03380         LDI     ;SKIPS LINE # BYTES
7F79 EDA0     03390         LDI
7F7B 18D5     03400         JR      MAINLN
7F7D DDBE00   03410 NEXT10  CP      (IX)    ;COMPARE WITH COMMAND LETTER
7F80 20D0     03420         JR      NZ,MAINLN
7F82 DDCB017E 03430         BIT     7,(IX+1)        ;SEE IF END OF CMD
7F86 200E     03440         JR      NZ,NEXT20
7F88 7E       03450         LD      A,(HL)  ;GET BYTE
7F89 DD23     03460         INC     IX      ;BUMP CMD PRT
7F8B DDBE01   03470         CP      (IX+1)  ;AVOID DOUBLE LETTERS
7F8E 28C2     03480         JR      Z,MAINLN;WHICH THROW OFF LOOP
```

```
7F90 FE2E      03490          CP      '.'      ;TEST FOR ABBR
7F92 20D6      03500          JR      NZ,CL3+1
7F94 EDA0      03510          LDI     ;MOVE PERIOD/BLANK/COMMA
7F96 7E        03520 NEXT20   LD      A,(HL)   ;TEST FOR BLANK
7F97 FE20      03530          CP      20H      ;MOVE & RETEST
7F99 28F9      03540          JR      Z,NEXT20-2
7F9B ED53637E  03550          LD      (DESAVE),DE      ;SAVE DESTINATION
7F9F CD6D7E    03560          CALL    INCV     ;CONVERT THE #
7FA2 E5        03570          PUSH    HL       ;MOVE LINE
7FA3 D9        03580          EXX     ;GO TO ALT REGS
7FA4 D1        03590          POP     DE
7FA5 E5        03600          PUSH    HL       ;SAVE SOURCE
7FA6 D5        03610          PUSH    DE       ;SAVE # AGAIN!
7FA7 DD2A127F  03620          LD      IX,(BASE)
7FAB 210042    03630          LD      HL,4200H
7FAE ED5B637E  03640          LD      DE,(DESAVE)
7FB2 CD0B7E    03650          CALL    BCOUNT
7FB5 0B        03660          DEC     BC
7FB6 D1        03670          POP     DE
7FB7 CDDC7F    03680          CALL    BILDLO   ;SEE IF LOWER ADDR
7FBA A7        03690          AND     A        ;RETURNS BLANK IF
7FBB 280D      03700          JR      Z,PUTIT  ;LOC FOUND
7FBD E1        03710          POP     HL       ;RECOVER PTR
7FBE E5        03720          PUSH    HL       ;FOR COUNT
7FBF D5        03730          PUSH    DE       ;SAVE AGAIN
7FC0 11097E    03740          LD      DE,TOP
7FC3 CD0B7E    03750          CALL    BCOUNT
7FC6 D1        03760          POP     DE
7FC7 CDD77F    03770          CALL    BUILD
7FCA E1        03780 PUTIT    POP     HL       ;RESTORE HI POINTER
7FCB ED5B637E  03790          LD      DE,(DESAVE)
7FCF 7E        03800          LD      A,(HL)
7FD0 FE2C      03810          CP      ','      ;TEST FOR MULT ADDR
7FD2 C2527F    03820          JP      NZ,MAINLN
7FD5 18BD      03830          JR      NEXT20-2;MOVE COMMA & CONVT
7FD7 3E0D      03840 BUILD    LD      A,13
7FD9 EDB1      03850          CPIR    ;CT IN BC & MATCH ADDR IN DE
7FDB C0        03860          RET     NZ       ;NO MATCH
7FDC 0B        03870 BILDLO   DEC     BC
7FDD 0B        03880          DEC     BC
7FDE CB78      03890          BIT     7,B      ;CHECK COUNTER CARRY
7FE0 C0        03900          RET     NZ
7FE1 22E87F    03910          LD      (CKPNT),HL
7FE4 23        03920          INC     HL
7FE5 23        03930          INC     HL       ;SKIP LINE #
7FE6 E5        03940          PUSH    HL
7FE8           03950 CKPNT    EQU     $+1      ;HL HAS CONTENTS
7FE7 2A0000    03960          LD      HL,(NN)  ;OF ADDR IT PTS TO
7FEA E7        03970          RST     20H      ;COMPARE THEM
7FEB E1        03980          POP     HL       ;RESTORE PTR
7FEC 280A      03990          JR      Z,FND
7FEE C5        04000          PUSH    BC       ;SAVE CT
7FEF ED4B0E7F  04010          LD      BC,(INCMNT)
7FF3 DD09      04020          ADD     IX,BC    ;ADJ NEW NO
7FF5 C1        04030          POP     BC       ;RESTORE CT
7FF6 18DF      04040          JR      BUILD
7FF8 DDE5      04050 FND      PUSH    IX       ;NOW IX HAS NEEDED #
7FFA E1        04060          POP     HL
7FFB CD257E    04070          CALL    OUTCV    ;CONVERT IT
7FFE AF        04080          XOR     A
7FFF C9        04090          RET
45FE           04100          END     ZAP
00000 TOTAL ERRORS
```

Command

by Arthur B. Rosenberg

TRSDOS allows the execution of only one command or program through the use of the AUTO command when powering up or resetting the computer.

Shortly after getting my first disk, I found that before I could run my BASIC program, I had to load BASIC and several machine-language programs, as well as specify the number of files, and set the memory size. I also found that for a given disk I usually used the same start-up sequence, which I did not always remember. As my disk collection grew, the problem became worse, and I started to keep a written start-up procedure for each disk.

I wanted to do away with the need for entering commands from the keyboard. The easiest way to accomplish this was to use a program which replaced and simulated the keyboard when the computer requested an input. Furthermore, I wanted to use the keyboard, if needed, to supply the date, time, or other input. I also wanted the ability to use different commands or to easily generate other versions for different disks.

COMMAND/CMD, shown in Program Listing 1, is a machine-language program which contains the code necessary to simulate the keyboard, the text to be used by the simulator, and a self-loader to place the program on a particular disk with a given filespec. The second program, COMMAND/BAS, shown in Program Listing 2, is a BASIC program. It is used to rewrite the text in COMMAND/CMD, name the rewritten program, place it on any disk, and, if you wish, make it invisible.

You can assemble COMMAND/CMD using the Editor/Assembler and load it any place in memory. I located it starting at AA00H, because that is below the area in which I usually load machine-language programs and above where COMMAND/BAS resides. You will have to modify lines 530 through 560 and line 5030 in COMMAND/BAS if COMMAND/CMD is assembled in another location. (See Table 1.)

COMMAND/CMD

Lines 360 through 380 of COMMAND/CMD load the address of the start of the first command to be executed into the text pointer. The address of the simulator is then placed into the keyboard device control block at 4016H, and the program jumps to DOS READY. The computer would normally wait for a keyboard input, but here it passes control to the simulator. The simulator gets one character of text from the text buffer and passes control to the display routine. This continues until the program detects a carriage return, 0DH. The computer then executes the command, returns to DOS READY, and the process starts again.

Line Number	Change	=	To
530	XY(0)		originate + 9CH
540	XY(1)		originate + DBH
550	XX		originate + D2H
560	PZ		originate + 1EH
5030	DEFUSR 1		originate + 08H

Table 1. *COMMAND/BAS line modifications*

Lines 920 through 970 temporarily restore keyboard operation if user interaction is required. This is necessary if you wish to set the time or enter other data. The above processes continue until a 0H is detected. This signifies that the next character is the last one of the last command. Lines 820 through 870 restore normal keyboard operation and output the last character. The text of commands is stored in memory starting at AB1EH and can continue through ACFFH. Changing line 1570 from ACFFH to FFFFH would allow text to continue to the end of memory.

The program is entered at AA08H, line 450, when used with COMMAND/BAS. This loads the text pointer with the address of the self-loading commands, loads the keyboard device control block with the simulator address, and jumps to DOS READY, which causes the simulator to output the command. (See lines 1490 through 1700.) DUMP filename/CMD:d (START = X'AA00',END = X'ACFF',TRA = X'AA00), and if the file is to be invisible, ATTRIB filename/CMD:d (I) , which loads the program onto the disk and gives the invisible attribute. The program then clears the screen, displays two messages, and jumps to 0000H, which restarts the computer and executes the new text.

COMMAND/BAS

COMMAND/BAS generates new text for COMMAND/CMD to execute, provides a filename for the program, and tells the computer which disk drive to place the file on and whether it is to be visible or not.

The program is completely self-prompting. Lines 520 through 1000 initialize the program. Note that the program allows for 20 lines of text to be entered. Change the CLEAR and DIM values in line 520 and the FOR X = 1 TO nn in line 2030 if you need more text space. Lines 3520 through 3620 allow entry of text until a slash (/) is entered. The text is then displayed and you are asked if you wish to change the text. If you do, the computer shows you each line of text separately. Pressing ENTER will leave that line unchanged. Newly entered text will replace the existing line of text. Entering a # will terminate the editing session. You can change the text again or proceed.

Lines 3020–3120 name the command file. Any valid TRSDOS filename may be used. The extention /CMD is automatically added to the filename. The default filename is COMMAND. You are then asked to supply a drive number and to indicate whether the program is to be visible or not. The defaults are drive 0 and visible.

The program then POKEs the text, filespec, and invisible attributes, if required, into memory and redisplays the text. Pressing ENTER causes the program to pass control to the self-loader portion of COMMAND/CMD (line 5030), which then loads itself on the selected disk with the chosen filespec. It then restarts the computer so that it can execute the new command file.

To use COMMAND/CMD, enter and save COMMAND/BAS. Then enter, assemble, and save COMMAND/CMD using the Editor/Assembler. The BASIC program, Poker (shown in Program Listing 3), which POKEs the code into memory, may be used instead. Run Poker, enter DEBUG, and type GAA08. This will cause a jump to AA08H which will cause COMMAND/CMD to load itself onto drive 0 with a filespec of COMMAND/CMD. Enter BASIC and run COMMAND/BAS, enter a filename, and you are ready to go. Don't forget to load or execute COMMAND/CMD immediately before you run COMMAND/BAS. You can't modify the text if it isn't in memory.

Almost any command you can enter from the keyboard can be used. If you enter a command which will result in a display on the screen, such as PRINT TIME $, then the next line of text must be blank. In other words, after a PRINT or similar command, skip the next line by entering a carriage return.

Program Listing 1. *COMMAND/CMD*

```
               ;***********************************************************
               ;*                                                         *
               ;*               ***** COMMAND/CMD  *****                  *
               ;*               *****     7.2      *****                  *
               ;*               *****   Ø4/11/81   *****                  *
               ;*                                                         *
               ;*                                                         *
               ;*               *****      BY      *****                  *
               ;*                 ARTHUR B. ROSENBERG                     *
               ;*                  497 MADISON DRIVE                      *
               ;*               EAST WINDSOR N.J. Ø852Ø                   *
               ;*                                                         *
               ;***********************************************************
               ;
               ;PROGRAM SIMULATES KEYBOARD INPUTS.  TEXT CAN BE CHANGED
               ;USING "COMMAND/BAS". REWRITTEN TEXT AND THIS PROGRAM
               ;THEN BE LOADED ON THE DISK VIA A USR CALL TO THIS
               ;PROGRAM.
               ;ENTRY AT AAØØ EXECUTES THE SIMULATED TEXT.
               ;ENTRY AT AAØ8 LOADES THIS PROGRAM ON THE DISK.
               ;
               ;                          * COMMAND *
               ;
AAØØ                    ORG     ØAAØØH
               ;
               ;
               ;                * START "COMMAND/CMD" *
               ;
AAØØ 211EAB             LD      HL,DATA         ;GET ADDRESS FIRST COMMAND
AAØ3 2212AA             LD      (POINT),HL      ;SAVE IT
AAØ6 18ØC               JR      LOAD            ;GO TO LOAD
               ;
               ;
               ;                * START USR1--"COMMAND/BAS"-- *
               ;
AAØ8 2197AA             LD      HL,DATA1        ;GET ADDRESS SAVE ROUTINE
AAØB 2212AA             LD      (POINT),HL      ;SAVE IT
AAØE 18Ø4               JR      LOAD            ;GO TO LOAD
               ;
               ;
               ;
4Ø16           KBADDR   EQU     4Ø16H           ;KEYBOARD DRIVER ADDRESS
ØØØ2           TRKBAD   DEFS    2               ;SAVE SYS KB DVR HERE
ØØØ2           POINT    DEFS    2               ;PUT START OF DATA HERE
3CØØ           VIDEO    EQU     3CØØH           ;START SCREEN ADDRESS
               ;
               ;
               ;
               ;        * LOAD SIMULATOR ADDRESS INTO KB DVR ARDDRESS *
               ;
AA14 2A164Ø    LOAD     LD      HL,(KBADDR)     ;GET SYS KB DVR ADDRESS
AA17 221ØAA             LD      (TRKBAD),HL     ;SAVE IT
AA1A 2123AA             LD      HL,SIMUL        ;GET SIMULATOR ADDRESS
AA1D 22164Ø             LD      (KBADDR),HL     ;STORE IN KB BCD
AA2Ø C32D4Ø             JP      4Ø2DH           ;GO TO "DOS READY"
               ;
               ;
   .           ;                * KB SIMULATOR *
               ;
AA23 2A12AA    SIMUL    LD      HL,(POINT)      ;POINT TO DATA
AA26 7E                 LD      A,(HL)          ;GET FIRST LETTER
AA27 23                 INC     HL              ;POINT TO NEXT LETTER
AA28 2212AA             LD      (POINT),HL      ;SAVE POINTER
AA2B FEØØ               CP      Ø               ;IS LAST LETTER NEXT ?
AA2D 28ØB               JR      Z,RESTOR        ;GO IF DONE
AA2F FEØ1               CP      1               ;IS KB INPUT REQUIRED?
AA31 CC42AA             CALL    Z,INPUT         ;GO KB INPUT
AA34 FEØ2               CP      2               ;IS SAVE DONE ?
AA36 CA5AAA             JP      Z,END           ;GO TO CLOSE
AA39 C9                 RET                     ;RETURN IF NOT
               ;
               ;
               ;        * RESTORE SYSTEM TO NORMAL KB OPERATION *
               ;
```

Program continued

```
AA3A 7E          ØØ84Ø RESTOR  LD      A,(HL)          ;GET LAST LETTER
AA3B 2A1ØAA      ØØ85Ø         LD      HL,(TRKBAD)     ;GET SYS DVR ADDRESS
AA3E 22164Ø      ØØ86Ø         LD      (KBADDR),HL     ;RESTORE SYS KB DVR
AA41 C9          ØØ87Ø         RET                     ;RETURN
                 ØØ88Ø ;
                 ØØ89Ø ;
                 ØØ9ØØ ;       * KB INPUT DURING SIMULATOR OPERATION *
                 ØØ91Ø ;
AA42 2142AA      ØØ92Ø INPUT   LD      HL,INPUT        ;GET INPUT DVR ADDRESS
AA45 22164Ø      ØØ93Ø         LD      (KBADDR),HL     ;STORE IN KB DCB
AA48 CDD843      ØØ94Ø         CALL    43D8H           ;SCAN KB
AA4B B7          ØØ95Ø         OR      A               ;WAS THERE AN INPUT?
AA4C 28F4        ØØ96Ø         JR      Z,INPUT         ;NO? THEN LOOK AGAIN
AA4E FEØD        ØØ97Ø         CP      ØDH             ;IS IT CARRIAGE RTN?
AA5Ø 28Ø1        ØØ98Ø         JR      Z,RETURN        ;GO IF SO
AA52 C9          ØØ99Ø         RET                     ;RETURN IF NOT
                 Ø1ØØØ ;
                 Ø1Ø1Ø ;
                 Ø1Ø2Ø ;       * RETURN TO SIMULATOR OPERATION *
                 Ø1Ø3Ø ;
AA53 2123AA      Ø1Ø4Ø RETURN  LD      HL,SIMUL        ;GET SIMULATOR ADDRESS
AA56 22164Ø      Ø1Ø5Ø         LD      (KBADDR),HL     ;STORE IN KB DCB
AA59 C9          Ø1Ø6Ø         RET
                 Ø1Ø7Ø ;
                 Ø1Ø8Ø ;
                 Ø1Ø9Ø ;               * RESTART WITH NEW TEXT *
                 Ø11ØØ ;
AA5A 7E          Ø111Ø END     LD      A,(HL)          ;GET LAST LETTER
AA5B 2162AA      Ø112Ø         LD      HL,RUN          ;GET RESTART
AA5E 22164Ø      Ø113Ø         LD      (KBADDR),HL     ;STORE IN SYS KB ADDRESS
AA61 C9          Ø114Ø         RET
                 Ø115Ø ;
                 Ø116Ø ;
                 Ø117Ø ;       * RUN NEW "COMMAND/CMD" *
                 Ø118Ø ;
AA62 CDC9Ø1      Ø119Ø RUN     CALL    1C9H            ;CLEAR SCREEN
AA65 21EFAA      Ø12ØØ         LD      HL,MESS1        ;GET ADDRESS OF MESSAGE
AA68 11173E      Ø121Ø         LD      DE,VIDEO+535    ;SEND TO CENTER OF SCREEN
AA6B Ø117ØØ      Ø122Ø         LD      BC,MESSL1       ;LENGTH OF MESSAGE
AA6E EDBØ        Ø123Ø         LDIR                    ;DISPLAY MESSAGE
AA7Ø CD8AAA      Ø124Ø         CALL    DELAY           ;KEEP MESSAGE ON SCREEN
AA73 CDC9Ø1      Ø125Ø         CALL    1C9H            ;CLEAR SCREEN
AA76 21Ø6AB      Ø126Ø         LD      HL,MESS2        ;GET ADDRESS OF MESSAGE
AA79 11173E      Ø127Ø         LD      DE,VIDEO+535    ;SEND TO CENTER OF SCREEN
AA7C Ø118ØØ      Ø128Ø         LD      BC,MESSL2       ;LENGTH OF MESSAGE
AA7F EDBØ        Ø129Ø         LDIR                    ;DISPLAY MESSAGE
AA81 CD8AAA      Ø13ØØ         CALL    DELAY           ;KEEP MESSAGE ON SCREEN
AA84 CD8AAA      Ø131Ø         CALL    DELAY           ;FOR A WHILE LONGER
AA87 C3ØØØØ      Ø132Ø         JP      Ø               ;RESTART WITH NEW "COMMAND/CMD"
                 Ø133Ø ;
                 Ø134Ø ;
                 Ø135Ø ;               * DELAY *
                 Ø136Ø ;
AA8A Ø6Ø2        Ø137Ø DELAY   LD      B,2             ;COUNTER 2
AA8C 21FFFF      Ø138Ø LP1     LD      HL,ØFFFFH       ;COUNTER 1
AA8F 2B          Ø139Ø LP2     DEC     HL              ;DECREMENT COUNTER 1
AA9Ø 7C          Ø14ØØ         LD      A,H
AA91 B5          Ø141Ø         OR      L               ;COUNTER 1=Ø?
AA92 2ØFB        Ø142Ø         JR      NZ,LP2          ;IF NOT DECREMENT AGAIN
AA94 1ØF6        Ø143Ø         DJNZ    LP1             ;DEC. COUNTER 2.IS=Ø ?
AA96 C9          Ø144Ø         RET
                 Ø145Ø ;
                 Ø146Ø ;
                 Ø147Ø ;       * "SAVE" INPUT AND "MESSAGE" DATA *
                 Ø148Ø ;
AA97 44          Ø149Ø DATA1   DEFM    'DUMP '
AA9C 43          Ø15ØØ NAME1   DEFM    'COMMAND/CMD:Ø '        ;POKE FILE SPEC HERE
AAAA 2Ø          Ø151Ø         DEFM    ' (START=X'
AAB3 27          Ø152Ø         DEFB    27H
AAB4 41          Ø153Ø         DEFM    'AAØØ'
AAB8 27          Ø154Ø         DEFB    27H
AAB9 2C          Ø155Ø         DEFM    ',END=X'
AABF 27          Ø156Ø         DEFB    27H
AACØ 41          Ø157Ø         DEFM    'ACFF'
AAC4 27          Ø158Ø         DEFB    27H
AAC5 2C          Ø159Ø         DEFM    ',TRA=X'
AACB 27          Ø16ØØ         DEFB    27H
AACC 41          Ø161Ø         DEFM    'AAØØ'
AADØ 27         Ø162Ø          DEFB    27H
AAD1 29         Ø163Ø          DEFB    ')'
AAD2 Ø2         Ø164Ø VIS      DEFB    2       ;IF VISIBLE FILL WITH Ø2H
AAD3 ØD         Ø165Ø          DEFB    ØDH             ;CARRAIGE RETURN
AAD4 41         Ø166Ø          DEFM    'ATTRIB '
AADB 2Ø         Ø167Ø NAME2    DEFM    '             '         ;POKE FILE SPEC HERE
AAE9 2Ø         Ø168Ø          DEFM    ' (I)'
AAED Ø2         Ø169Ø          DEFB    2               ;LAST LETTER NEXT
AAEE ØD         Ø17ØØ          DEFB    ØDH             ;CARRAIGE RETURN
AAEF 2A         Ø171Ø MESS1    DEFM    '* TEXT STORED ON DISK *'
ØØ17            Ø172Ø MESSL1   EQU     $-MESS1
ABØ6 2A         Ø173Ø MESS2    DEFM    '* EXCUTING NEW PROGRAM *'
ØØ18            Ø174Ø MESSL2   EQU     $-MESS2
                Ø175Ø ;
                Ø176Ø ;
                Ø177Ø ;                * INPUT "COMMAND" DATA *
                Ø178Ø ;
AB1E 43         Ø179Ø DATA     DEFM    'CLOCK'
                Ø18ØØ ;
AB23 ØD         Ø181Ø          DEFB    ØDH             ;CARRIAGE RTN
AB24 54         Ø182Ø          DEFM    'TIME '
AB29 Ø1         Ø183Ø          DEFB    1               ;KB INPUT REQUIRED
AB2A 44         Ø184Ø          DEFM    'DATE '
AB2F Ø1         Ø185Ø          DEFB    1               ;KB INPUT REQUIRED
AB3Ø 42         Ø186Ø          DEFM    'BASIC'
AB35 ØD         Ø187Ø          DEFB    ØDH             ;CARRAIGE RTN
AB36 ØD         Ø188Ø          DEFB    ØDH
AB37 ØD         Ø189Ø          DEFB    ØDH
AB38 52         Ø19ØØ          DEFM    'RUN"COMMAND/BAS"'
AB48 ØØ         Ø191Ø          DEFB    Ø               ;LAST LETTER NEXT
AB49 ØD         Ø192Ø          DEFB    ØDH
AAØØ            Ø193Ø          END     ØAAØØH
ØØØØØ TOTAL ERRORS
```

Program Listing 2. *COMMAND/BAS*

```
1Ø :
   ' **********************************************************
15 :
   ' *                                                        *
3Ø :
   ' *              ***** COMMAND/BAS  *****                  *
35 :
   ' *              *****    2.2       *****                  *
4Ø :
   ' *              *****  Ø8/24/8Ø    *****                  *
45 :
   ' *                                                        *
5Ø :
   ' *              *****     BY       *****                  *
6Ø :
   ' *                ARTHUR B. ROSENBERG                     *
7Ø :
   ' *                 497 MADISON DRIVE                      *
75 :
   ' *              EAST WINDSOR N.J. Ø852Ø                   *
8Ø :
   ' *                                                        *
85 :
   ' **********************************************************
1ØØ :
    '
11Ø :
    '
5ØØ :
    '               ***** INITIALIZE *****
51Ø :
    '
52Ø CLEAR 1ØØØ:
    DEFSTR A:
    DIM A(2Ø):
```

Program continued

```
     DEFINT X,Y,P,I,B:
     GOTO 5520
 530 XY(0) = &HAA9C:
     ' =NAME1
 540 XY(1) = &HAADB:
     ' =NAME2
 550 XX = &HAAD2:
     ' =VIS
 560 PZ = &HAB1E:
     ' =DATA
 570 :
     '
1000 :
     '
1010 :
     '
1020 CLS :
     PRINT CHR$(23):
     PRINT @ 530,A1:
     FOR X = 1 TO 1000:
      NEXT
1030 PRINT @ 600,"-2.2-":
     FOR X = 1 TO 250:
      NEXT
1040 CLS :
     PRINT @ 27,A1
1050 :
     '
1060 :
     '
1500 :
     '                  ***** INSTRUCTIONS *****
1510 :
     '
1520 PRINT :
     INPUT "DO YOU WISH INSTRUCTIONS";A:
     IF A > = "Y"
      THEN
       GOSUB 6020
1530 :
     '
1540 :
     '
2000 :
     '                  ***** WRITE TEXT *****
2010 :
     '
2020 PRINT @ 27, A1:
     PRINT :
     PRINT AH
2030 FOR X = 1 TO 50
2040  LINE INPUT A(X)
2050  IF A(X) = "/" GOTO 2520
2060  NEXT
2070 :
     '
2080 :
     '
2500 :
     '                 ***** DISPLAY COMPLETED TEXT *****
2510 :
     '
2520 Y = X - 1
2530 CLS :
     PRINT @ 27,A1
2540 PRINT AI:
     PRINT
2550 FOR X = 1 TO Y
2560  PRINT A(X);
2570  IF RIGHT$(A(X),1) = "I"
       THEN
        PRINT TAB(24);"***** REQUIRES KEYBOARD INPUT *****";
2580  PRINT
2590  NEXT
2600 PRINT :
     PRINT "TYPE ";A2;"C";A2;" IF YOU WISH TO CHANGE THE TEXT: ELSE T
     YPE ";A2;"OK";A2;:
     INPUT A
```

```
2610 IF A = "C" GOTO 3520
2620 :
     '
2630 :
     '
3000 :
     '                    ***** NAME FILE *****
3010 :
     '
3020 CLS :
     PRINT @ 450,"ENTER COMMAND FILE NAME ===> ";
3030 LINE INPUT AZ
3040 IF AZ = ""
      THEN
       AZ = "COMMAND"
3050 AZ = AZ + "/CMD"
3060 PRINT @ 514,"WHICH DRIVE IS THIS FILE TO BE PLACED? ";
3070 LINE INPUT AX
3080 IF AX = ""
      THEN
       AX = ":0" :
      ELSE
       AX = ":" + AX
3090 AZ = AZ + AX
3100 PRINT @ 578,"IS THIS FILE TO BE INVISIBLE";:
     INPUT AY
3110 IF AY = > "Y"
      THEN
       POKE XX,32 :
      ELSE
       POKE XX,2
3120 GOTO 4020
3130 :
     '
3140 :
     '
3500 :
     '                    ***** EDIT TEXT *****
3510 :
     '
3520 GOSUB 6090
3530 A = ""
3540 FOR X = 1 TO Y + 1
3550  IF X = Y + 1 PRINT "LAST ENTRY":
      FOR Q = 1 TO 1000:
       NEXT :
      GOTO 2530
3560  PRINT A(X)
3570  LINE INPUT A
3580  IF A = "#" GOTO 2530
3590  IF A = ""
       THEN
        3620
3600  A(X) = A
3610  A = ""
3620  NEXT
3630 :
     '
3640 :
     '
4000 :
     '               ***** STORE TEXT IN MEMORY *****
4010 :
     '
4020 P = PZ
4030 FOR X = 0 TO 13
4040  POKE XY(0) + X,32
4050  POKE XY(1) + X,32
4060  NEXT
4070 FOR X = 1 TO LEN(AZ)
4080  POKE XY(0) + X - 1, ASC( MID$(AZ,X,1))
4090  POKE XY(1) + X - 1, ASC( MID$(AZ,X,1))
4100  NEXT
4110 FOR X = 1 TO Y
4120  IF Z = 0 PRINT @ 978,"* STORING TEXT IN MEMORY *";:
      Z = 1:
      :
      ELSE
```

Program continued

```
        PRINT @ 978, STRING$(26," ");:
        Z = 0
4130  FOR I = 1 TO LEN(A(X))
4140   IF ( LEN(A(X)) = 0) AND (X = Y)
        THEN
         POKE P,0:
         P = P + 1:
         POKE P,13:
         GOTO 4520 :
         :
        ELSE
         IF LEN(A(X)) = 0
          THEN
           POKE P,13:
           P = P + 1:
           NEXT X:
          GOTO 4520
4150  B = ASC( MID$(A(X),I,1))
4160  IF I = LEN(A(X)) AND B = 73
       THEN
        POKE P,1:
        P = P + 1:
        IF X = Y GOTO 4520 :
        :
       ELSE
        NEXT X:
       GOTO 4520
4170 IF I = LEN(A(X)) AND B < > 73
      THEN
       POKE P,B:
       P = P + 1:
       IF (X = Y) AND I = LEN(A(X))
        THEN
         POKE P,0:
         P = P + 1:
         POKE P,13:
         P = P + 1:
         GOTO 4520 :
         :
        ELSE
         POKE P,13:
         P = P + 1:
         NEXT X:
         GOTO 4520
4180 POKE P,B:
     P = P + 1
4190 NEXT I,X
4200 :
     '
4210 :
     '
4500 :
     '            ***** DISPLAY TEXT IN MEMORY *****
4510 :
     '
4520 CLS :
     PRINT @ 20,AM:
     P1 = P:
     P = PZ
4530 FOR X = P TO P1 - 1
4540  IF PEEK(X) = 1
       THEN
        PRINT TAB(24) "***** REQUIRES KEYBOARD INPUT *****":
        GOTO 4560
4550  PRINT CHR$( PEEK(X));
4560  NEXT
4570 PRINT :
     PRINT :
     LINE INPUT "PRESS ENTER TO STORE ON DISK";A
4580 :
     '
4590 :
     '
5000 :
     '               ***** STORE ON DISK *****
5010 :
     '
```

```
5020 CLS :
     PRINT @ 528,"***** STORING TEXT ON DISK *****"
5030 DEF USR1 = &HAA08
5040 X = USR1 (0)
5050 STOP
5060 :
     '
5070 :
     '
5500 :
     '                    ***** INSTRUCTIONS *****
5510 :
     '
5520 A1 = "* COMMAND *"
5530 A2 = CHR$(34)
5540 A3 = "THIS PROGRAM WILL ALLOW YOU TO REWRITE THE MACHINE LANGUAG
     EPROGRAM "
5550 A4 = "COMMAND/CMD"
5560 A5 = ".  THE EXISTING "
5570 A6 = " PROGRAM WILL BE DESTROYED UNLESS YOU RENAME IT."
5580 A7 = "SIMPLY TYPE IN THE COMMANDS OR OPERATIONS IN THE ORDER YOU
      WISHTHEM TO BE EXECUTED WHEN YOU ARE PROMPTED. "
5590 A8 = "IF YOU WISH TO ENTER KEYBOARD DATA AS PART OF A COMMAND OR
      OPERATION SIMPLY TYPE AN "
5600 A9 = " (FOR INPUT) AS THE LAST CHARACTER OF THAT COMMAND. TYPE "
5610 A0 = "INSTEAD OF A COMMAND WHEN YOU HAVE COMPLETED YOUR TEXT.YOU
      WILL THEN BE SHOWN THE COMPLETE TEXT.IF YOU WISH TO CHANGE THE
     TEXT TYPE "
5620 AA = " FOR CHANGE.TYPE "
5630 AB = " IF THE TEXT IS OK."
5640 AC = " RESET THE       COMPUTER AND EXECUTE THE PROGRAM."
5650 AD = "AN EXAMPLE OF TEXT FOLLOWS:NOTE:  THE COMMENTS WHICH FOLLO
     W "
5660 AE = "*****"
5670 AF = " ARE NOT PART OF THE    TEXT."
5680 AG = "CLOCKDATE I                          ***** REQUIRES KEYBOARD INPUT
     TIME I                    ***** REQUIRES KEYBOARD INPUTVERIFYBASIC
     FILES 3MEMORY SIZE 33000RUN "
5690 AQ = "COMMAND/BAS"
5700 AH = "START ENTERING YOUR TEXT:"
5710 AI = "THIS IS THE COMPLETE TEXT:"
5720 AJ = "THE TEXT WILL BE DISPLAYED ONE LINE AT A TIME.ENTER NEW TE
     XT IF YOU WISH TO CHANGE THAT LINE.PRESS ENTER IF YOU WISH TO LE
     AVE THAT LINE AS IT IS.  THE NEXTLINE OF TEXT WILL THEN BE DISPL
     AYED."
5730 AK = " TYPE A "
5740 AL = " AS THE NEXTENTRY WHEN YOU HAVE CHANGED THE LAST LINE OF T
     EXT YOU WISH TOCHANGE."
5750 AM = "* TEXT STORED IN MEMORY *"
5760 AN = "* TEXT STORED ON DISK *"
5770 AO = "* EXCUTING PROGRAM *"
5780 AP = "YOU WILL THEN BE ASKED TO SUPPLY A NAME FOR THE FILE.  ANY
      NAME UP TO 8 CHARACTERS BEGINNING WITH A LETTER MAY BE USED.THE
      DEFAULT NAME IS "
5790 AQ = "COMMAND"
5800 AR = ".  THE EXTENSION "
5810 AS = "/CMD"
5820 AT = " IS ADDED TOALL FILE NAMES."
5830 AU = "  YOU WILL   THEN BE ASKED TO SPECIFY IF THE FILE  IS TO
     BE INVISIBLE.  THE  DEFAULT IS VISIBLE."
5840 AV = "NOTE:   IF A FILE ALREADY EXISTS WITH THE SAME NAME AND IS
     INVISIBLE THE NEW FILE WILL ALSO BE INVISIBLE.  SEE TRDOS &DISK
     BASIC MANUAL, PAGE 4-12."
5850 AW = "THE COMPUTER WILL THEN PLACE THE TEXT IN MEMORY, DUMP THE
     NEW   FILE TO THE DISK, WITH THE SPECIFIED FILE NAME,"
5860 AX = "  NEXT YOU WILL SPECIFY THE DRIVE NUMBER ON WHICHTHE FILE
     IS TO BE PLACED.  THE DEFAULT IS DRIVE 0."
5870 GOTO 530
5880 :
     '
5890 :
     '
6000 :
     '          ***** INSTRUCTION PRINT SUBROUTINE *****
6010 :
     '
6020 PRINT :
     PRINT :
```

Program continued

```
     PRINT :
     PRINT :
     PRINT A3;A2 + A4 + A2;A5;A2 + A4 + A2;A6
6030 PRINT STRING$(4, CHR$(13)):
     GOSUB 6100
6040 PRINT @ 27,A1:
     PRINT :
     PRINT :
     PRINT A7;A8;" ";A2;"I";A2;A9;A2;"/";A2;" ";A0;A2;"C";A2;AA;A2;"O
     K";A2;AB:
     PRINT :
     PRINT :
     PRINT :
     GOSUB 6100
6050 PRINT @ 27,A1:
     PRINT :
     PRINT AP;A2 + AQ + A2;AR;A2 + AS + A2;AT;AX;AU:
     PRINT :
     PRINT AV:
     PRINT :
     GOSUB 6100
6060 PRINT @ 27,A1:
     PRINT STRING$(4, CHR$(13)):
     PRINT AW;AC:
     PRINT STRING$(5, CHR$(13)):
     GOSUB 6100
6070 PRINT @ 27,A1:
     PRINT :
     PRINT AD;A2 + AE + A2;AF
6080 PRINT :
     PRINT AG;A2 + AQ + A2:
     PRINT :
     GOSUB 6100 :
     RETURN
6090 CLS :
     PRINT @ 27,A1:
     PRINT :
     PRINT AJ;AK;A2;"#";A2;AL:
     PRINT :
     GOSUB 6100 :
     RETURN
6100 LINE INPUT "PRESS ENTER TO CONTINUE";A:
     CLS
6110 RETURN
```

Program Listing 3. *Poker*

```
 10 :
    ' **********************************************************
 15 :
    ' *                                                        *
 20 :
    ' *              *****  POKER/BAS   *****                  *
 25 :
    ' *              *****     2.1      *****                  *
 30 :
    ' *              *****  04/11/81    *****                  *
 35 :
    ' *                                                        *
 40 :
    ' *              *****      BY      *****                  *
 45 :
    ' *                ARTHUR B. ROSENBERG                     *
 50 :
    ' *                 497 MADISON DRIVE                      *
 55 :
    ' *              EAST WINDSOR N.J. 08520                   *
 60 :
    ' *                                                        *
 65 :
    ' **********************************************************
100 :
    '
```

```
110 CMD "T"
120 DEFINT A,X
130 FOR X = - 22016 TO - 21687:
     READ A:
     POKE X,A:
     PRINT X;:
     NEXT
140 END
150 DATA 33, 30, 171, 34, 18, 170, 24, 12, 33, 151, 170, 34, 18, 170
    , 24, 4, 71, 70, 69, 68, 42, 22, 64, 34, 16, 170, 33, 35, 170, 3
    4, 22, 64, 195, 45, 64, 42, 18, 170, 126, 35, 34, 18, 170, 254,
    0, 40
160 DATA 11, 254, 1, 204, 66, 170, 254, 2, 202, 90, 170, 201, 126, 4
    2, 16, 170, 34, 22, 64, 201, 33, 66, 170, 34, 22, 64, 205, 216,
    67, 183, 40, 244, 254, 13, 40, 1, 201, 33, 35, 170, 34, 22, 64,
    201
170 DATA 126, 33, 98, 170, 34, 22, 64, 201, 205, 201, 1, 33, 239, 17
    0, 17, 23, 62, 1, 23, 0, 237, 176, 205, 138, 170, 205, 201, 1, 3
    3, 6, 171, 17, 23, 62, 1, 24, 0, 237, 176, 205, 138, 170, 205, 1
    38, 170
180 DATA 195, 0, 0, 6, 2, 33, 255, 255, 43, 124, 181, 32, 251, 16, 2
    46, 201, 68, 85, 77, 80, 32, 67, 79, 77, 77, 65, 78, 68, 47, 67,
     77, 68, 58, 48, 32, 32, 40, 83, 84, 65, 82, 84, 61, 88, 39, 65,
     65
190 DATA 48, 48, 39, 44, 69, 78, 68, 61, 88, 39, 65, 67, 70, 70, 39,
     44, 84, 82, 65, 61, 88, 39, 65, 65, 48, 48, 39, 41, 2, 13, 65,
    84, 84, 82, 73, 66, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
    32, 32
200 DATA 32, 32, 32, 40, 73, 41, 2, 13, 42, 32, 84, 69, 88, 84, 32,
    83, 84, 79, 82, 69, 68, 32, 79, 78, 32, 68, 73, 83, 75, 32, 42,
    42, 32, 69, 88, 67, 85, 84, 73, 78, 71, 32, 78, 69, 87, 32, 80,
    82, 79
210 DATA 71, 82, 65, 77, 32, 42, 67, 76, 79, 67, 75, 13, 84, 73, 77,
     69, 32, 1, 68, 65, 84, 69, 32, 1, 66, 65, 83, 73, 67, 13, 13, 1
    3, 82, 85, 78, 34, 67, 79, 77, 77, 65, 78, 68, 47, 66, 65, 83, 3
    4, 0
220 DATA 13
```

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—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

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, generally on large-scale computers.

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

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

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

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

AND—a Boolean logic 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. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—A Programming Language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

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—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

B

backup—1) refers to making copies of all software and data stored externally; 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

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 in which many of the same types of 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—1) a unit of data transmission speed equal to the number of code elements (bits) per second; 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

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.

binary digit—the two digits, 0 and 1, used in binary notation. Often shortened to bit.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. 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. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

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 or hardware.

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 specially 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.

card reader—a device for reading information from punched cards.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode— in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

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 the block of data.

chip— the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit— a conductor or system of conductors through which an electric current may flow.

circuit card— a printed circuit board containing electronic components.

clear— to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

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—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement— a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

computer interface— a device designed for data communication between a central computer and another unit such as a programmable controller processor.

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.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

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 a read/write section of tape.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

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.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

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.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

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.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

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), a 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 manufacture.

downtime—the time when a system is not available for production due to required maintenance.

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 Electrical 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 editors.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation is performed.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power to which a floating-point number is raised.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

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 occurred.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

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.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

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. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit.

immediate—addressing mode in which 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—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one particular operation.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

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 compiler.

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.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that displays alphanumeric characters when supplied with a specified voltage.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line printer—a high-speed printing device that prints an entire line at one time.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has 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.

lsb—see least significant bit.

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.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

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.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—us. One millionth of a second: 1×10^{-6} or 0.000001 second.

millisecond—ms. One thousandth of a second: 10^{-3} or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

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.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

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.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication 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. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

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 limit the number of loops that can be nested.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

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.

off-line—describes equipment or devices which are not connected to the communications line.

off-the-shelf—a term referring to software. A generalized program that can be used by many computer owners. It is mass produced and can be 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.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

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 depends on the computer. Most micros are eight-bit word machines. 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. Contrast with serial.

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.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check— a check that tests whether the number of 1s in an array of binary digits is odd or even.

PC board—see printed circuit board.

peripheral devices— a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permutation— arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT— a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1— an acronym for Programming Language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter— a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

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.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

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.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

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 fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of dividend remaining after a divide has been completed.

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.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. Rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

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

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

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 electricity. Transistors and integrated circuits are made from semiconductive material and are 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 (+).

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

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.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

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.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularlyfor storage of register values during interrupt routines. See LIFO.

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.

storage—see memory.

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.

subtrahend—the number that is subtracted from the minuend.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

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.

tape reader—a unit which is capable of sensing data from punched tape.

Teletype™—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

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.

transistor—an active component of an electronic circuit consisting of a small block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1, and 0 volts is low or 0; 5V = 1, 0V = 0).

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

von Neumann, John (1903–1957)—mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. 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 flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

INDEX

INDEX COMPILED BY NAN MCCARTHY

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

$10.95 ISBN 0-88006-043-3