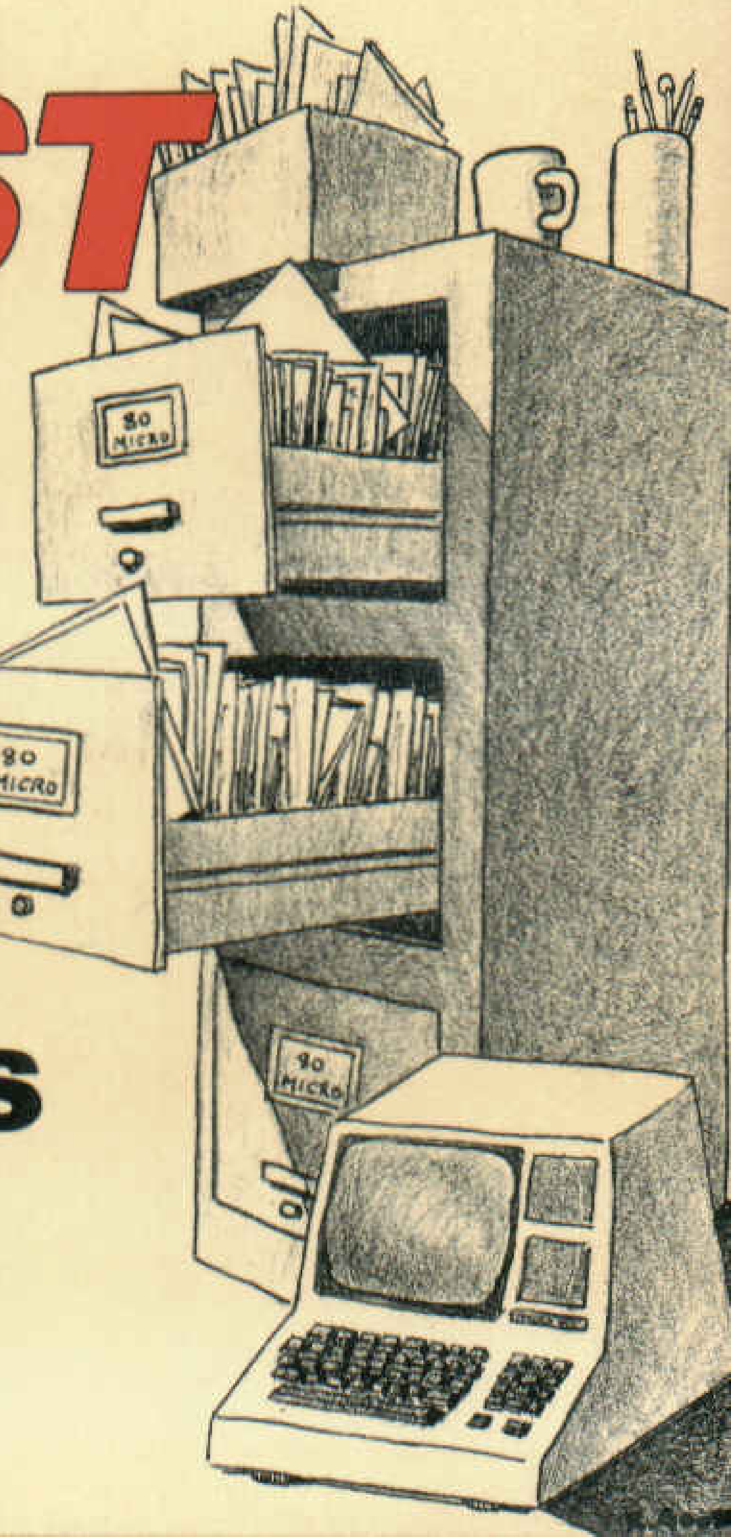
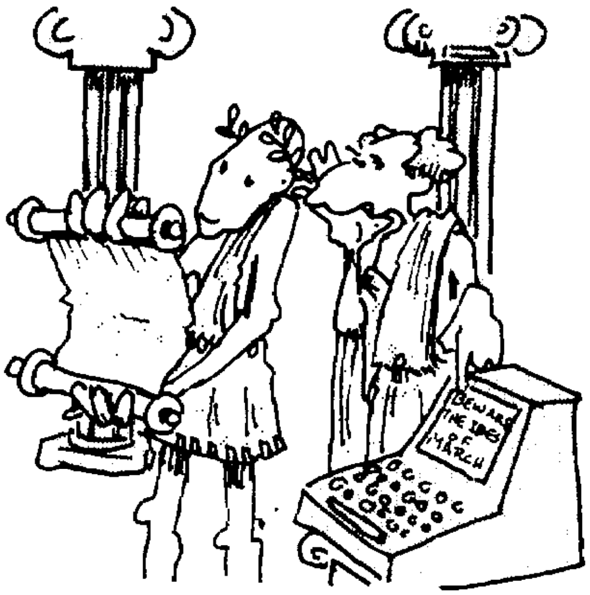
THE REST OF 80

all new tutorials and utilities



The Rest of 80


BEWARE THE IDES OF MARCH

The Rest of 80

A Wayne Green Publication
Peterborough, New Hampshire

Published by
Wayne Green Books
Peterborough, New Hampshire 03458

ISBN 0-88006-062-X

Cover design by Gary Ciocci
Cover illustration by Roger Goode

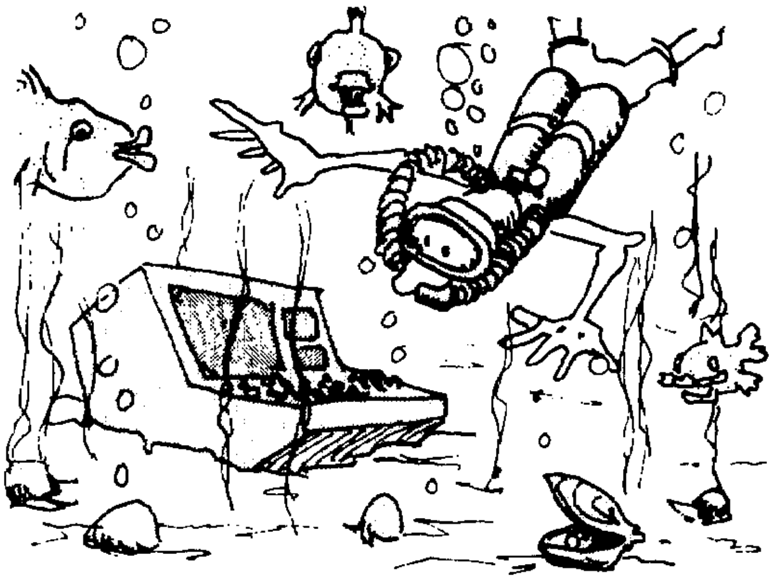
Table of Contents

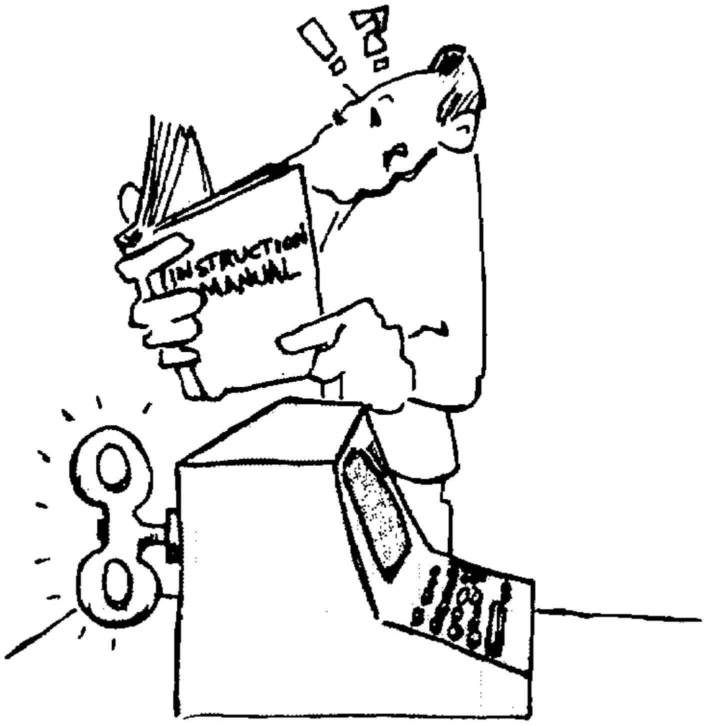
Introduction

The people at *80 Micro* hate to turn down good manuscripts. The problem is trying to fit them all in the magazine. So somebody came up with the great idea: "Let's take the best of the 'rest' of *80* and publish them in a book!"

Here it is.

We've taken special care to edit and design this book for easy use and long life. We know you're going to want it around. Never before published anywhere, these thirty-one tutorials and utilities represent some of the very best manuscripts ever sent to *80 Micro*. Enjoy!




!?
INSTRUCTION MANUAL

1

Going Beyond Sequential and Random Access

by Bradford Russo

All data processing requires the storage of data on a peripheral mass storage device. For most microcomputers, this means disks. TRSDOS and most other microcomputer operating systems only support two fundamental file access methods, sequential and random. Both have limitations.

Sequential Access

Sequential access was the first method used in data processing, because of the nature of the first mass storage media. To read any given piece of data from punched cards or from magnetic tape, it was necessary to read through all the records that came before (Figure 1). The sequential access technique led to batch processing, which has some major disadvantages. The most obvious problem is time lag, since data is not processed until a batch, typically a whole day's or week's worth of activity, accumulates. For example, data affected by activity on Monday may not be brought up to date until Friday.

This earliest of file access methods is also available on microcomputers. In sequential accessing, the computer stores data (in ASCII code) in the same order it is written, starting at the beginning of the file. Sequential filing is handy when you need to store only one or a few variables worth of data. It's also useful when data is needed in RAM in the form of a table or a set of subscripted arrays. You can use simple and effective FOR-NEXT loops to read the data into and out of RAM. The limited accessing capability becomes unimportant, since once the data resides in RAM it can be accessed in many ways.

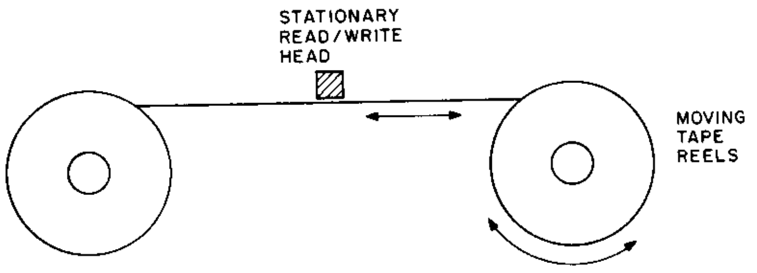


Figure 1

Random Access

The random file access technique was developed next, after the advent of mass storage media which could be read in non-serial order, including drum and disk. The read/write head on disks not only reads serially around one track, but can also move sideways from one track to the next (Figure 2). Multiple platter disk drives also read up and down along a third dimension by selecting the desired platter from a stack of several (Figure 3). These new storage devices led to interactive and real-time processing methods. The advantage to real-time processing is its instantaneous updating of data. This is very useful, for instance, in a business inventory control system. New shipments can be recorded immediately instead of waiting for a complete batch to be processed.

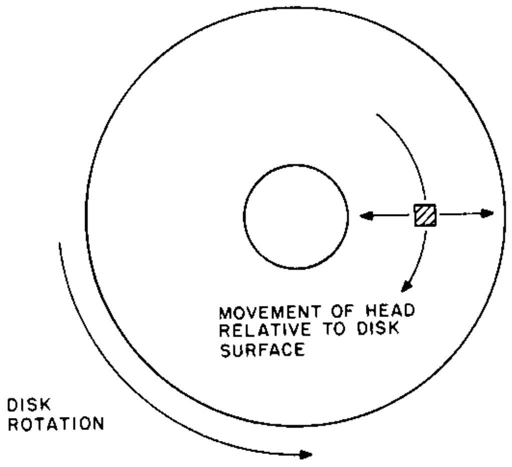


Figure 2

The process of jumping from one given record to any other record resembles random movement; however, the process is more accurately described as direct or relative. Direct means the computer can go directly to a given record, without reading all the intervening records; relative means that a particular record is referenced by its location relative to the beginning of the file. In other words, record number 126 is the one hundred twenty-sixth record from the beginning of the file. For example, assume the file contains records sized so that exactly ten of them fit on one track of the disk. The first track of the file contains record numbers 1-10. The second track holds 11-20, and so on. To locate record number 126, the computer first moves to the thirteenth track, which holds records 121-130, then reads around the track from 121 to 126. This two-dimensional read process only requires 13+6=19 reads to find record number 126. That's a lot faster than a sequential read (Figure 4).

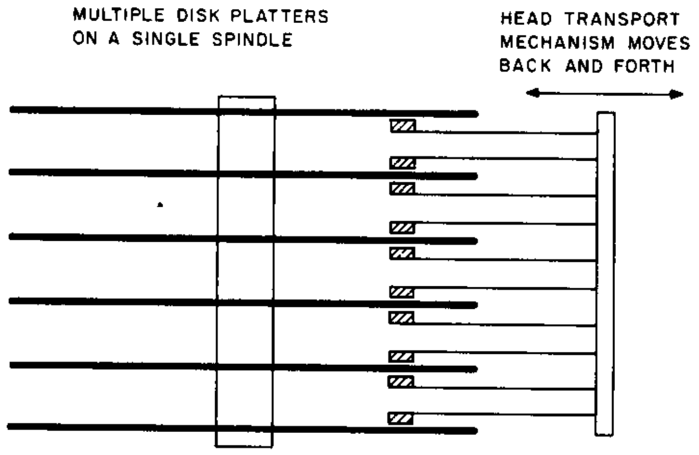

MULTIPLE DISK PLATTERS ON A SINGLE SPINDLE
HEAD TRANSPORT MECHANISM MOVES BACK AND FORTH

Figure 3

In random access files, a record is referenced by a numeric value called the record number. Each record of the file must have its own identifying number. For example, in a payroll package, the data pertaining to a given employee could be identified by the employee's clock number. If the employees are numbered serially, from 1 on up, clock number becomes synonomous with record number. The identifier (clock number) is directly related to record number. You can also use random access to store data whose identifiers are non-numeric. An example of this is a file storing a record for each day's activities, using dates as identifiers. The record storing data for the fifth of March would have an identifier of March 5, which translates into physical record number 64 by the following process: since there are 31 days in January, and 28 in February, that means March 5 is 31+28+5=64 days into the year.

Sometimes the identifier cannot be directly related to record number. Suppose the identifier is to be Social Security number instead of clock number. A Social Security Number is nine digits long, so any one person must fall between 100-00-0000 and 999-99-9999. That represents 899,999,999 possible combinations, which would require nine hundred million records. If your company has only one hundred employees, that means nine million records are wasted for every record actively used. Such a waste is impossible, of course, since there isn't that much on-line storage available on your computer.

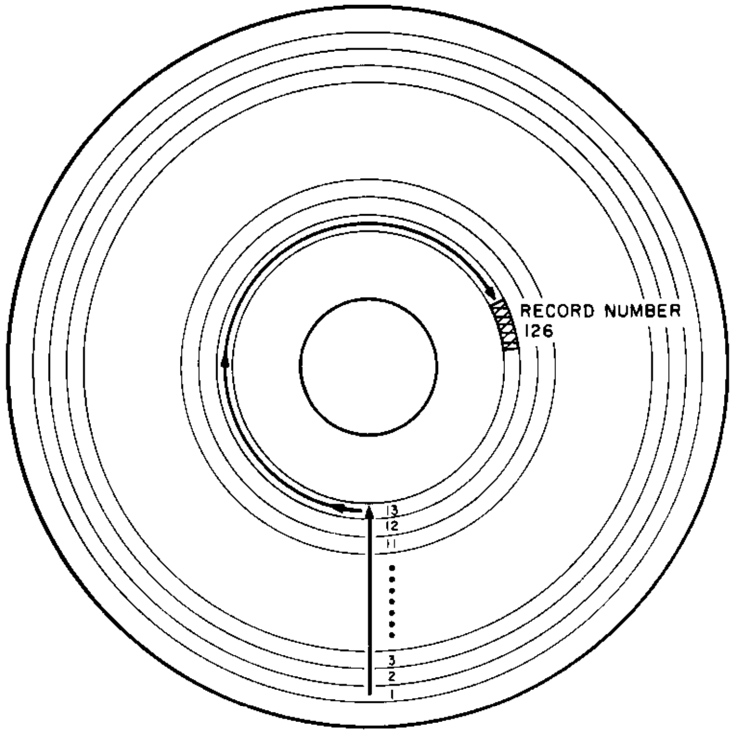


Figure 4

It would be more practical to relate the 100 identifiers (100 employees) to a file of a little more than 100 records. The earliest technique developed to do this is called hashing. In hashing, the relationship between identifier and record number is no longer direct, but is based on a mathematical formula—an algorithm devised for reducing the widely varying identifiers down to a condensed set of record numbers. Occasionally, two or more identifiers hash down to the same record number. This is called a collision, and the identifiers are called synonyms. The hashing algorithm must have provisions for handling these incidents.

ISAM or Keyed Index

Another technique for relating an identifier to a record number is commonly known as ISAM (Indexed Sequential Access Method), or keyed index. The unique identifier of a record is called the key field, or key. ISAM is not supported by TRSDOS or by most other microcomputer operating systems; however, it is available for the TRS-80 Model II in the compiler version of BASIC, as well as FORTRAN and COBOL. You can create an indexed access method under TRSDOS by combining the sequential and random methods.

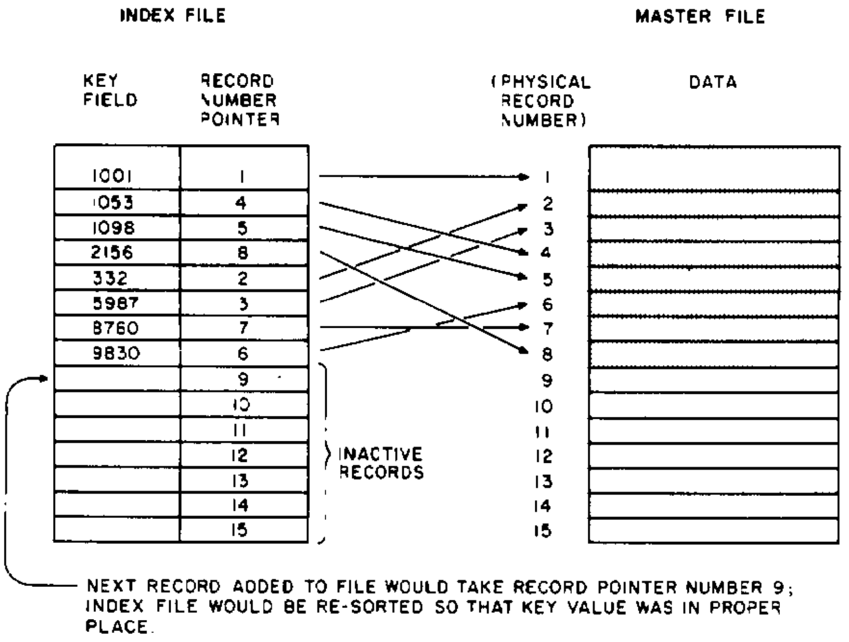
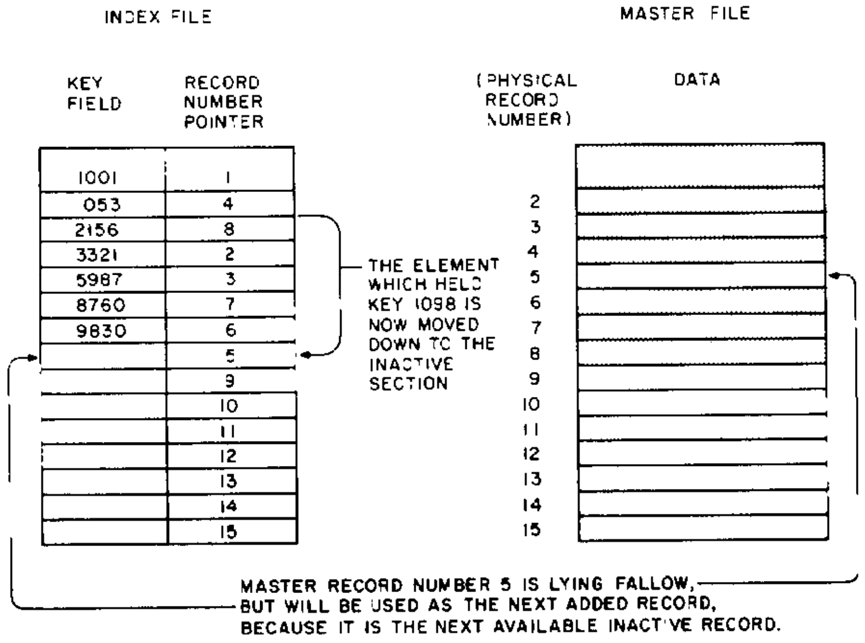


Figure 5

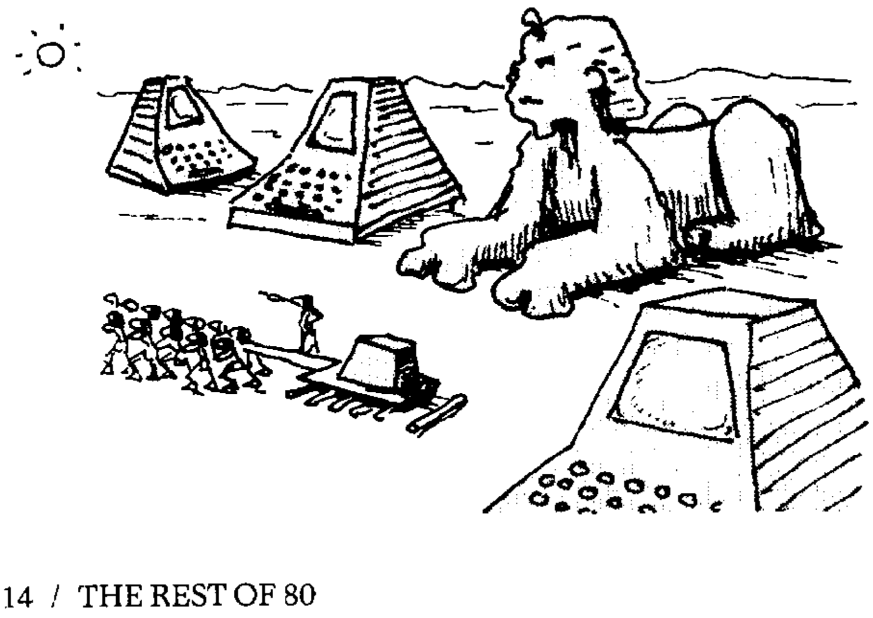
Two separate files are required for an indexed technique. One is the master file, which holds all the data which is to be stored. The other is an index to the master file. Unlike hashing, there is no mathematical relationship between a given key (identifier), and its associated record number. The number is not derived from the key, but assigned arbitrarily. When data is added to the master file, it is placed in the next available physical record on the disk. This record's number is placed in the index file beside its associated key (Figure 5). An indexed technique does not waste disk space for keys that might exist, as direct random access does. It only consumes space for keys which are actually being used. When a record is deleted from the master file, that space is recovered and can be used later, to hold new data (Figure 6).



INDEX FILE
MASTER FILE
KEY FIELD
RECORD NUMBER POINTER
(PHYSICAL RECORD NUMBER)
DATA
1001 1
053 4
2156 8
3321 2
5987 3
8760 7
9830 6
5
9
10
11
12
13
14
15
THE ELEMENT WHICH HELD KEY 1098 IS NOW MOVED DOWN TO THE INACTIVE SECTION
2 3 4 5 6 7 8 9 10 11 12 13 14 15
MASTER RECORD NUMBER 5 IS LYING FALLOW, BUT WILL BE USED AS THE NEXT ADDED RECORD, BECAUSE IT IS THE NEXT AVAILABLE INACTIVE RECORD.

Figure 6

In order to read data from the master file, the index file is first searched for the requested key. That key has an associated record number beside it, which is used to read the appropriate record from the master file. Usually, the index file is sorted by key. You can use a serial read of the index file, key by key, to produce reports in key sequence.



2

An Unlistable, Unbreakable Program

by Jon Boczkiewicz

System Requirements
Level II BASIC or
Disk BASIC
Model I or III

Have you teachers ever wanted instructional programs your students could not break to find the answers? A simple POKE command disables the BREAK key. However, students soon learn they can load and list a program before they run it. Here is my simple method for making your own programs unlistable and unbreakable. You can use it with any program, although Disk BASIC requires different numbers than Level I or II. The technique involves breaking the problem into two parts, because solving one problem creates another.

Make It Unlistable

The first step is to make the program unlistable. While the available numbers using two bytes run to 65535 (256 times 256 minus one), line numbers are limited to 0-65529. I do not know what these last six numbers are reserved for, but a program line number in this range causes a syntax error message. The computer keeps track of where to go by requiring the first two bytes of a program line to be the address of the beginning of the next program line. The second two bytes of the program line are the line number. After you write and debug a program, change the first line number to a number greater than 65529. Listing the program causes the first line to be read as a number higher than the limit, and a READY message appears. It does not affect the program to have a line number higher than acceptable.

You can change a line number with a POKE. The problem is knowing where to POKE. Remember the memory map in Appendix D of your

Level II BASIC Reference Manual? A look at the map shows the BASIC program text starts at memory location 17129. This location contains the first byte of the first program line. Bytes three and four (the line numbers) must be in memory locations 17131 and 17132. So after the program is running and debugged, save and back up your own copies, strip all the comments out of the program in the computer, and compress it, if you can. Then, from the command mode, POKE 17131,251 (or higher) :POKE 17132,255. It should now be unlistable. Run the program—it should be unchanged, except for a small problem with GOTOs or GOSUBs.

GOTOs or GOSUBs find their target line by going to the start of the program and looking for the line number, starting with the first line. But now the first line has a number that is higher than the highest allowed. The solution is simple--undo what you have done. You POKEd numbers into memory locations 17131 and 17132 to change the line number. Now POKE the original numbers back, but this time from *within* the program. Your first program line (the one which gets renumbered) can do this. You can use a special line:10 POKE 17131, 10:POKE 17132, 0. But this leaves you right where you were, able to load, run, break and list the program.

Make It Unbreakable

You could disable BREAK by POKEing 16396, 23 and later enable it by POKE 16396, 201, but if you have a mean streak, try the following sequence: POKE 16396, 62: POKE 16397, 187: POKE 16398,0. This redefines the BREAK key to NEW, and wipes out the program whenever the key is hit. Or, you can POKE 16397 with any of the internal codes for the BASIC keywords from Appendix E of the *Level II BASIC Reference Manual*. The program then runs as desired, including GOTO and GOSUB, but whenever you press BREAK, the program is NEWed. This will not work if you have had Radio Shack's lowercase modification installed; instead you get whatever symbol has the ASCII number you POKEd into 16397. If you are using Disk BASIC (version 2.2 or higher), POKEing 16396,23 causes the system to reboot when you press BREAK.

You must consider two other possibilities for program security. First, a student could run the program to its end, then list it. This is prevented by the END statement which POKEs 17131,251 and 17132,255, and enables BREAK with POKE 16396,201. The program should have only one END statement, and all other ending points should GOTO this statement. Second, a student could deliberately or accidentally make an error that would cause the program to break and list before running again. Lessen the chance of this by using error traps. These can be statements that require re-entry of wrong data, or a simple GOTO that sends execution to the END statement. The only essential condition is that the program end through a single statement.

The Code

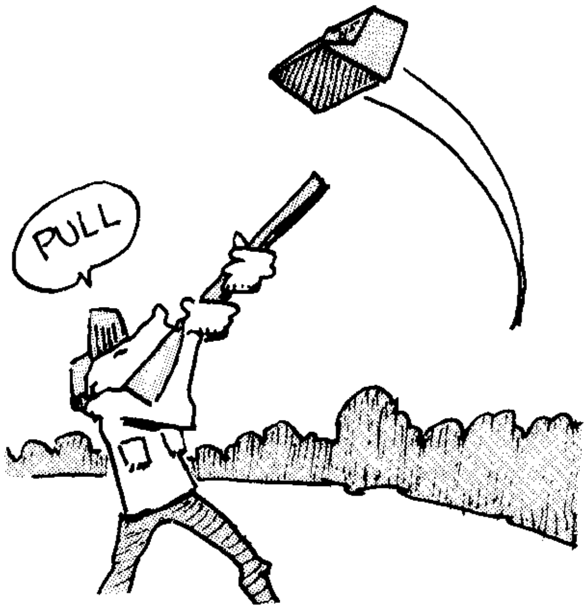
To use this technique, make the following line the first line of your program:

```
10 POKE 17131,10:POKE 17132,0:POKE 16396,62:POKE 16397,187:POKE 16398,0
```

If the line number you select is 5, POKE 17131,5. Now add the ending line as follows:

```
XXXX POKE 17131,251:POKE 17132,255: POKE 16396,201:END
                    (or NEW instead of END).
```

If the program is in Disk BASIC 2.2, replace 17131 with 26304 and 17132 with 26305. In Disk BASICR 2.2, use 26959 and 26960. Otherwise, everything is the same. To check, run the program. Everything should work except BREAK. Now, in command mode, POKE 17131,254:POKE 17132,255 or 27754 and 27755, if in Disk BASIC. Try to list it—it should not work. You can save the program on tape or disk, load it with no special effort, and run it.



3

Form Fillout Technique

by Gary S. Lindsey

System Requirements:
Level II BASIC
Model I or III

Have you ever considered the time and the number of programming steps it takes to write the code for the entry, viewing, editing, and reviewing of data? When your program requires a lot of data entry, using a form fillout program can save both programmer and user a lot of time and aggravation.

This process involves displaying a form, often with data included, on a screen with protected fields. The user enters into the free fields new or modified data by overwriting all or part of the data. Then the user presses a combination of keys to transmit the new or modified data back to the computer. The computer takes the data and converts it to the variables necessary for processing.

The programs in this chapter allow a programmer to implement a version of the form fillout technique on the Model I TRS-80. Program Listing 1 is written in assembly language for the Z80 microprocessor. Program Listing 2 is written in BASIC to load and demonstrate the assembly-language program. The use of assembly language is necessary for speed and the implementation of commands that are not available in the BASIC interpreter.

In my system, I have placed the BASIC subroutines necessary to implement the form fillout program in ASCII disk files that can be appended to any BASIC program. The program listings in this article are designed for a 48K disk system. I tested all conversions on my system by disconnecting the disks or expansion memory and by setting the memory size to the appropriate values.

Implement the form fillout technique by including two subroutines, 50000 and 60000, in a BASIC program. Subroutine 50000 loads the assembly-language program into memory. Subroutine 60000 places certain BASIC variables which describe the data to be processed into memory for use by the assembly-language program, and calls the assembly-language program. The BASIC program places variables, prompts, and titles on the screen. The assembly-language program accesses the variable data directly from the screen, allowing changes. The variables are then sequentially placed directly into the memory allocated by the BASIC interpreter for the variables A$(1) through A$(N). The assembly-language program handles the data from the screen as ASCII characters. When control is returned to the BASIC program, the new or modified data is present in the A$(I) variables in the order that they appeared on the screen. The BASIC program must convert the A$(I) variables into the appropriate program numeric and string variables.

The BASIC interpreter in the TRS-80 locates string variables through the use of a 3-byte description of the string. Byte 1 contains the length of the string; bytes 2 and 3 contain the memory address of the string pointer. The address of this 3-byte description is determined by using the VARPTR(A$(1)) BASIC command, which returns the address of byte 1 of the description of A$(1) variable through the use of this command, and POKEs it into particular memory locations for use by the assembly-language program. Since the 3-byte descriptions for all A$(I) variables are contiguous in memory, it is only necessary to pass the address of the A$(1) string to the assembly-language program.

Because the assembly-language program writes data directly into the A$(I) locations up to the allowable lengths of the variables, the A$(I) variables must be expanded to their maximum possible lengths before the assembly-language program is executed. Without this step, the A$(I) variables may be of any length when the assembly-language program is called, and you may destroy the validity of the 3-byte description by writing more or less data into a variable than the BASIC interpreter is expecting.

Using the Form Fillout Program

The assembly-language program interprets the six possible operator commands and responds appropriately.

- The left and right arrows move the cursor within the variable being worked.
- The up and down arrows move the cursor to the first position of the next or the preceding variable. The up and down arrows can be used at any time, independent of the cursor position within the variable, without changing the variable. Previous changes to the variable are not disturbed.

•The carriage return works like the down arrow if the cursor is in the first position of the variable. If the cursor is anywhere else and a carriage return is entered, the remaining portion of the variable is filled with blanks. This makes new data entry easy since, under BASIC, the carriage return terminates entries to the computer.
•Pressing the equal sign stores the variable data from the screen into the RAM used by the BASIC interpreter for storage of the A$(I) variable and returns control to the BASIC program.

The data on the screen is changed at the cursor location whenever a character is entered that is not one of the control characters mentioned above.

Implementation of the form fillout subroutines in a program is very simple, as illustrated in the accompanying demonstration program. The following steps must be taken in order:
•Set the memory size to 65217 for a 48K configuration.
•Load the assembly-language program with a GOSUB 50000 at the beginning of the program.
•Clear the screen and print the form and its variables on the screen. The variables are detected by the assembly-language program by the presence of a colon (:). The screen must be printed with a colon and two blank spaces preceding each changeable variable. Since the assembly-language program searches for a colon as the start of variables, you can protect variables from modification by preceding them with any character except a colon. The assembly-language program searches for the colons, skipping over any data on the screen prior to reaching a colon.
•Define N as the number and L(I) as the lengths of the variables. GOSUB 60000 stores these variables in memory for use by the assembly-language program and calls the USR program.
•Convert the A$(I) variables returned by the assembly-language program into the appropriate program string and numeric variables.
•Proceed with the user's program.

Sample Outputs

To illustrate the use of the form fillout program, I have included some sample outputs from the demonstration program. Figures 1, 2, and 3 show the three screen displays before any data has been entered into them. Note in Figure 2 that the numeric entries are shown as zeros. Also note that a blank space for the sign has been included for the numeric values. The BASIC interpreter adds these blanks. You don't need to include the space when you enter data onto the screen. Figure 4 shows Display 1 with a portion of the data entered, and Figure 5 shows Display 1 with a complete set of data.

Figure 6 shows Display 2 after the NAME has been entered using Display 1. Note that the NAME prompt and variable are separated by an asterisk. The asterisk means that the NAME cannot be changed using Display 2. The cursor jumps over the NAME variable and positions itself at the beginning of the EMPLOYEE NUMBER variable. Figure 7 shows a completed Display 2. Figure 8 shows Display 3 with the protected fields and titles before any data is entered. Figure 9 shows a completed Display 3.

```
                          DISPLAY 1
 DEMONSTRATES DATA ENTRY/EDIT WITHOUT PROTECTED FIELDS
NAME :                         DATE OF BIRTH :
STREET ADDRESS :
CITY/STATE :
PHONE NUMBER :
OCCUPATION :
```

Figure 1. *Data entry without protected fields*

```
                          DISPLAY 2
   DEMONSTRATES DATA ENTRY/EDIT WITH PROTECTED FIELDS
                 INCLUDING NUMERIC VALUES
NAME *                         EMPLOYEE NUMBER :    0
HIRE DATE :
SECTION NUMBER :    0
YEAR OF DEGREE :    0
SALARY :    0
```

Figure 2. *Data entry with protected fields*

```
                          DISPLAY 3
DEMONSTRATES FORMATTED SCREEN WITH EXTRA COMMENTS/TITLES
                        PERSONAL DATA
NAME*                                    PHONE NUMBER*
ADDRESS*
                                  RECREATIONAL DATA
HOBBIES   NUMBER 1:
          NUMBER 2:
SKILLS    NUMBER 1:
          NUMBER 2:
```

Figure 3. *Formatted screen with additional data*

```
                          DISPLAY 1
 DEMONSTRATES DATA ENTRY/EDIT WITHOUT PROTECTED FIELDS
NAME: JOHN Q. PUBLIC           DATE OF BIRTH: 7/4/1954
STREET ADDRESS :
CITY/STATE :
PHONE NUMBER :
OCCUPATION:
```

Figure 4. *Data entry without protected fields*

```
                         DISPLAY 1
   DEMONSTRATES DATA ENTRY/EDIT WITHOUT PROTECTED FIELDS
NAME: JOHN Q. PUBLIC        DATE OF BIRTH: 7/4/1954
STREET ADDRESS: 100 ANYSTREET
CITY/STATE: ANYTOWN, USA 00000
PHONE NUMBER: 555-1212
OCCUPATION: COMPUTER PROGRAMMER
```

Figure 5. *Screen display showing complete data entry*

```
                         DISPLAY 2
     DEMONSTRATES DATA ENTRY/EDIT WITH PROTECTED FIELDS
                  INCLUDING NUMERIC VALUES
NAME* JOHN Q. PUBLIC        EMPLOYEE NUMBER: 0
HIRE DATE:
SECTION NUMBER: 0
YEAR OF DEGREE: 0
SALARY: 0
```

Figure 6. *Data entry with protected fields*

```
                         DISPLAY 2
     DEMONSTRATES DATA ENTRY/EDIT WITH PROTECTED FIELDS
                  INCLUDING NUMERIC VALUES
NAME* JOHN Q. PUBLIC        EMPLOYEE NUMBER: 10000
HIRE DATE: 6/26/75
SECTION NUMBER: 1234
YEAR OF DEGREE: 1974
SALARY: 23000
```

Figure 7. *Display showing complete data entry*

```
                         DISPLAY 3
DEMONSTRATES FORMATTED SCREEN WITH EXTRA COMMENTS/TITLES
                       PERSONAL DATA
NAME* JOHN Q. PUBLIC              PHONE NUMBER* 555-1212
ADDRESS* 100 ANYSTREET
         ANYTOWN, USA 00000
                       RECREATIONAL DATA
HOBBIES  NUMBER 1:
         NUMBER 2:
SKILLS   NUMBER 1
         NUMBER 2
```

Figure 8. *Formatted screen with additional data*

```
                         DISPLAY 3
DEMONSTRATES FORMATTED SCREEN WITH EXTRA COMMENTS/TITLES
                       PERSONAL DATA
NAME* JOHN Q. PUBLIC              PHONE NUMBER* 555-1212
ADDRESS* 100 ANYSTREET
```

Figure continued

```
                     RECREATIONAL DATA
HOBBIES  NUMBER 1: COMPUTER HOBBYIST
         NUMBER 2: FLYING AIRPLANES
SKILLS   NUMBER 1: 45 WPM TYPIST
         NUMBER 2: ELEC MAINTENANCE
```
Figure 9. *Display showing complete data entry*

Assembly-Language Program Description

The assembly-language part of the program is discussed here in symbolic language format to show how it works. The assembled program has been converted to decimal values and is POKEd into memory by the subroutine at line 50000 in the BASIC program. The principal variables in this program are as follows:

NUM	The number of changeable variables on the screen as POKEd into memory by the BASIC program. NUM corresponds to the BASIC variable N.
LEN	A series of numbers representing the lengths of the variables which are also POKEd into memory by the BASIC program. These correspond to the L(1) through L(N) BASIC variables.
VARWK	Tracks which variable is being entered. The range is 1 to N, with N less than or equal to 32.
POS	Variable indicating position of cursor within the variable being worked. The range is 0 to LEN.
PTR	An address POKEd into memory by the BASIC program which points to the address in RAM containing the pointer to the 3-byte description of the A$(1) variable. The BASIC program derives this address using the VARPTR(A$(1)) command.

In the assembly-language program, lines 170–330 set up the initial register variables and position the cursor at the beginning of the first variable. The INPUT routine, lines 370–540, attempts to get a character from the keyboard. If no key has been pressed, the subroutine generates a block cursor, delays, restores the character to the screen, and then tries again to get a character from the keyboard. This process creates a flashing cursor on the video screen. Once the routine finds a character, control branches to the UPARR routine at line 580.

The first two lines of the UPARR routine check to see if the input character is an up arrow. If it is not an up arrow, control branches to the RTARR routine at line 810. If it is an up arrow, the routine checks to see if the variable being worked is the first variable on the screen. If so, control is returned to the INPUT routine with no further action. If the variable is not the first, the screen is searched backward, and the cursor is placed over the first position of the previous variable. The routine then passes control back to the INPUT routine.

The first two lines of the RTARR routine check to see if the input character is a right arrow. If the character is not a right arrow, the routine jumps to the LTARR routine at line 950, otherwise, a check is made to see if the cursor is at the last allowable position of the variable. If it is, control is passed to INPUT with no changes made. If the cursor is not at the last position, it moves forward one character space on the screen. Control is then passed to the INPUT routine.

The first two lines of the LTARR routine check to see if the input character is a left arrow. If the character is not a left arrow, control is passed to the DNARR routine at line 1070. If the character is a left arrow, a check is made to see if the cursor is in the first position of the variable. If it is, control is passed to INPUT with no changes made. If the cursor is not in the first position, the cursor is backed up one character space on the screen. Control is then passed to the INPUT routine.

The first two lines of the DNARR routine check to see if the input character is a down arrow. If it is not a down arrow, control is passed to the CARRET routine at line 1270. If the character is a down arrow, a check is made to see if the last variable on the screen is being processed. If so, control is returned to the INPUT routine with no changes. If not, the cursor is moved down to the first position of the next variable. Control is then passed to the INPUT routine.

The first two lines of the CARRET routine check to see if the input character is a carriage return. If the character is not a carriage return, control is passed to the EQUAL routine at line 1530. If it is a carriage return, a check is made to see if the cursor was in the first position of the variable. If so, control is passed to the DNARR routine, and the character is handled as a down arrow. If not, the number of blanks required to fill the remaining portion of the variable is calculated, and the remaining portion of the variable is filled with blanks. A check is then made to determine if the last variable on the screen was being processed when the key was pressed. If not, control is passed to the DNARR routine to move the cursor forward one variable. If so, control is passed to BEGIN, where the cursor is placed at position 0 of the first variable.

The first two lines of the EQUAL routine check to see if an equal sign was received. If not, control is passed to the CHAR routine at line 1820. If the character is an equal sign, the VARWK variable is set to 1 and the index register IX is loaded with the address of the BASIC pointer to the 3-byte description of the A$(1) variable in RAM. The cursor is then placed over position 0 of the first variable on the screen. The A$(1) pointer consists of three values: the length, the lower address, and the upper address of A$(1). First, B is loaded with the length, and DE is loaded with the address. The variable is then moved from the screen to the memory designated by the BASIC pointer to hold A$(1). This process continues for the other variables on the screen until the last variable is completed.

Control is then passed back to the BASIC program through the Z80 RET instruction.

The CHAR routine first checks to see if the cursor is at the last position of the variable. If so, control is then returned to the INPUT routine with no changes. If the cursor is not at the last position of the variable, the character is printed on the video screen and control returns to the INPUT routine.

Lines 1970-2010 allocate memory for the variables used by the assembly-language program.

BASIC Program Description

The BASIC portions of the form fillout program (Program Listing 2) are located in subroutines 50000 and 60000 in the demonstration program. The subroutine at 50000 loads the assembly-language portion of the program into high memory. The assembly-language routine, in decimal values, is located in lines 50090-50420. The data in the demonstration program is for a 48K disk configuration. The subroutine at 50000 need only be run once at the beginning of the BASIC program.

The subroutine at 60000 takes two types of variables from the BASIC program and stores them in appropriate memory locations for use by the assembly-language program. N represents the total number of variables on the display. L(1) through L(N) indicate the allowable lengths of the N variables. The subroutine then expands the A$(I) variables to the maximum lengths with line 60080 and derives the BASIC pointer to the A$(1) 3-byte description with the VARPTR command in line 60090. This decimal value is converted to a high and low memory address by lines 60110-60120 and POKEd into memory by line 60130. The assembly-language program is then called with the USR command in line 60140. Lines 60040 and 60150 save the variables used by the subroutine and restore these variables before exiting the subroutine. The only variables not available to the programmer are A$(I), L(I), N, A1, A2, A3, A4, A5, and A6. As a programming aid, the subroutine also checks to see that no more than 32 variables have been used. If this condition is not met, the program halts at that point.

Lines 100-230 set up the initial menu and load the assembly-language routine with a GOSUB 50000. Subroutine 1000 sets up the first display. Lines 1010-1100 print the display on the CRT. Line 1110 sets the number of variables to six and sets their lengths. The GOSUB 60000 sets up the assembly-language variables and calls the USR routine. Line 1130 converts the A$(I) variables returned into the appropriate program string variables.

The subroutine at 2000 is similar to the subroutine at 1000 with a few differences. Line 2060 prints the NAME on the screen but uses an asterisk delimiter to separate the prompt from the variable. The display contains numeric variables in addition to string variables. Line 2140 converts the ASCII strings returned into numeric variables where appropriate and converts the remaining strings into the program string variables.

The subroutine at 3000 illustrates another way to use the form fillout program. In this subroutine, the display is set up to include titles as well as prompts and protected variables. These demonstrate how the program ignores any data on the screen except the variables preceded by a colon and two blank spaces.

Conversion to 16K/32K

To convert the BASIC subroutines to run on a 16K or 32K TRS-80 with or without disks, the following lines must be substituted for the corresponding lines in the subroutines. For the 16K version, you must set memory size to 32449 before entering BASIC. For the 32K version, set memory size to 48833. Substitute the following lines for the 16K version:

```
50010  Z=32450
50090  DATA 221, 33, 255, 127, 253, 33, 254, 127, 17, 222, 127
50240  DATA 58, 221, 127
50300  DATA 58, 221, 127
50330  DATA 221, 42, 219, 127
50370  DATA 58, 221, 127
60050  Z=32450
```

Substitute the following lines for the 32K version:

```
50010  Z= -16702
50090  DATA 221, 33, 255, 191, 253, 33, 254, 191, 17, 222, 191
50240  DATA 58, 221, 191
50300  DATA 58, 221, 191
50330  DATA 221, 42, 219, 191
50370  DATA 58, 221, 191
60050  Z= -16702
```

Without Disk

To use this program without disks, the only additional changes are in lines 50030 and 60140. Be sure that you have changed the necessary lines if the memory size has been changed. In a 16K non-disk configuration, change these lines to read:

```
50030 POKE 16526, 194: POKE 16527, 126
60140 X=USR(X)
```

In a 32K, non-disk configuration, change these lines to read:

```
50030 POKE 16526, 194: POKE 16527, 190
60140 X=USR(X)
```

In a 48K, non-disk configuration, change these lines to read:

```
50030 POKE 16526, 194: POKE 16527, 254
60140 X=USR(X)
```

Program Listing 1.

Assembly-language program loaded by the subroutine at 50000 in the demonstration BASIC program

```
                          ;ASSEMBLY LANGUAGE PORTION OF FORM-FILLOUT
                          ;WRITTEN BY  :  GARY LINDSEY
                          ;               1Ø41 NOGALES AVENUE
                          ;               PALM BAY, FLORIDA 329Ø5
                  ;
                  ;=================================================
                  ;
FEC2                      ORG     ØFEC2H          ;SETS LOCATION OF PGM
FEC2 DD21FFFF             LD      IX,POS          ;IX POINTS TO POSITION
FEC6 FD21FEFF             LD      IY,VARWK        ;IY POINTS TO VARWK
                  ;
                  ;=================================================
                  ;
FECA 11DEFF       BEGIN   LD      DE,LEN          ;DE POINTS TO LEN OF VARWK
FECD 21FF3B               LD      HL,3BFFH        ;HL= TOP OF VIDEO MEM-1
FEDØ 23           INC1    INC     HL              ;HL=TOP OF VIDEO MEM
FED1 7E                   LD      A,(HL)          ;LOAD A WITH VIDEO CHAR
FED2 FE3A                 CP      ':'             ;CHECK IF A IS COLON
FED4 2ØFA                 JR      NZ,INC1         ;IF NOT COLON JP TO INC1
FED6 23                   INC     HL              ;INCREMENT VIDEO POINTER
FED7 23                   INC     HL              ;TO POS Ø OF VARIABLE
FED8 23                   INC     HL              ;AFTER THE COLON
FED9 FD36ØØØ1             LD      (IY),1          ;SETS VARWK TO 1
FEDD DD36ØØØØ             LD      (IX),Ø          ;SET POS TO Ø
                  ;
                  ;=================================================
                  ;
FEE1 D5           INPUT   PUSH    DE              ;SAVE LEN POINTER
FEE2 FDE5                 PUSH    IY              ;SAVE VARWK POINTER
FEE4 CD2BØØ               CALL    ØØ2BH           ;CALL SCAN KEYBOARD S/R
FEE7 B7                   OR      A               ;OR A
FEE8 FDE1                 POP     IY              ;RESTORE VARWK POINTER
FEEA D1                   POP     DE              ;RESTORE LEN POINTER
FEEB 2Ø15                 JR      NZ,UPARR        ;IF A NOT Ø JP TO UPARR
FEED 4E                   LD      C,(HL)          ;SAVE CHAR IN C
FEEE 3E8F                 LD      A,143           ;LOAD A WITH CURSOR CHAR
FEFØ 77                   LD      (HL),A          ;PUT CURSOR ON SCREEN
FEF1 C5                   PUSH    BC              ;SAVE BC WHICH HOLDS CHAR
FEF2 Ø1EAØØ               LD      BC,ØEAH         ;LOAD BC WITH DELAY COUNT
FEF5 CD6ØØØ               CALL    ØØ6ØH           ;CALL DELAY ROUTINE
FEF8 C1                   POP     BC              ;RESTORE BC
FEF9 71                   LD      (HL),C          ;PUT CHAR ON SCREEN
FEFA Ø1FØØØ               LD      BC,ØFØH         ;LOAD BC WITH DELAY COUNT
FEFD CD6ØØØ               CALL    ØØ6ØH           ;CALL DELAY ROUTINE
FFØØ 18DF                 JR      INPUT           ;JUMP TO INPUT
                  ;
                  ;=================================================
                  ;
FFØ2 FE5B         UPARR   CP      5BH             ;CHECK IF A IS UP ARROW
FFØ4 2Ø2Ø                 JR      NZ,RTARR        ;IF NOT JUMP TO RTARR
FFØ6 FD7EØØ               LD      A,(IY)          ;LD A WITH VARWK
FFØ9 FEØ1                 CP      1               ;CHECK IF FIRST VARIABLE
FFØB 28D4                 JR      Z,INPUT         ;IF TRUE JP TO INPUT
FFØD 2B           DEC1    DEC     HL              ;DECREMENT VIDEO POINTER
FFØE 7E                   LD      A,(HL)          ;LOAD A WITH VIDEO CHAR
FFØF FE3A                 CP      ':'             ;CHECK IF COLON
FF11 2ØFA                 JR      NZ,DEC1         ;IF NOT COLON JP TO DEC1
```

Program continued

```
FF13 2B       00670 DEC2     DEC     HL            ;DECREMENT VIDEO POINTER
FF14 7E       00680          LD      A,(HL)        ;LOAD A WITH VIDEO CHAR
FF15 FE3A     00690          CP      ':'           ;CHECK IF COLON
FF17 20FA     00700          JR      NZ,DEC2       ;IF NOT COLON JP TO DEC2
FF19 23       00710          INC     HL            ;INCREMENT VIDEO POINTER
FF1A 23       00720          INC     HL            ;TO POS 0 OF VARIABLE
FF1B 23       00730          INC     HL            ;AFTER COLON
FF1C FD3500   00740          DEC     (IY)          ;DECREMENTS VARWK
FF1F DD360000 00750          LD      (IX),0        ;SET POS TO 0
FF23 1B       00760          DEC     DE            ;DE NOW POINTS TO NEW LEN
FF24 18BB     00770          JR      INPUT         ;JUMP TO INPUT
              00780 ;
              00790 ;=====================================================
              00800 ;
FF26 FE09     00810 RTARR    CP      09H           ;CHECK IF A IS RT ARROW
FF28 200F     00820          JR      NZ,LTARR      ;IF NOT JUMP TO LTARR
FF2A 1A       00830          LD      A,(DE)        ;LD A WITH LENGTH OF VAR
FF2B DD4600   00840          LD      B,(IX)        ;LD B WITH POSITION
FF2E B8       00850          CP      B             ;COMPARE LEN WITH POS
FF2F 28B0     00860          JR      Z,INPUT       ;IF POS=LEN THEN INPUT
FF31 23       00870          INC     HL            ;INCREMENT VIDEO POINTER
FF32 DD3400   00880          INC     (IX)          ;INCREMENTS POS
FF35 18AA     00890          JR      INPUT         ;JUMP TO INPUT
              00900 ;
FF37 1891     00910 XFER1    JR      BEGIN         ;JUMP TO BEGIN
              00920 ;
              00930 ;=====================================================
              00940 ;
FF39 FE08     00950 LTARR    CP      08H           ;CHECK IF A IS LT ARROW
FF3B 200E     00960          JR      NZ,DNARR      ;IF NOT JP TO DNARR
FF3D 3E00     00970          LD      A,0           ;LOAD A WITH 0
FF3F DD4600   00980          LD      B,(IX)        ;LOAD B WITH POS
FF42 B8       00990          CP      B             ;COMPARE POS WITH 0
FF43 289C     01000          JR      Z,INPUT       ;IF POS=0 THEN INPUT
FF45 2B       01010          DEC     HL            ;DECRMENT VIDEO POINTER
FF46 DD3500   01020          DEC     (IX)          ;DECREMENTS POS
FF49 1896     01030 XFER2    JR      INPUT         ;JUMP TO INPUT
              01040 ;
              01050 ;=====================================================
              01060 ;
FF4B FE0A     01070 DNARR    CP      0AH           ;CHECK IF A IS DNARR
FF4D 201C     01080          JR      NZ,CARRET     ;IF NOT JP TO CARRET
FF4F FD4600   01090 DNARR1   LD      B,(IY)        ;LOAD B WITH VARWK
FF52 3ADDFF   01100          LD      A,(NUM)       ;LOAD A WITH NUM
FF55 B8       01110          CP      B             ;COMPAR VARWK WITH NUM
FF56 2889     01120          JR      Z,INPUT       ;IF VARWK=NUM THEN INPUT
FF58 23       01130 INC2     INC     HL            ;INCREMENT VIDEO POINTER
FF59 7E       01140          LD      A,(HL)        ;LOAD A WITH VIDEO CHAR
FF5A FE3A     01150          CP      ':'           ;CHECK IF COLON
FF5C 20FA     01160          JR      NZ,INC2       ;IF NOT COLON JP TO INC2
FF5E 23       01170          INC     HL            ;INCREMENT VIDEO POINTER
FF5F 23       01180          INC     HL            ;TO POS 0 OF VARIABLE
FF60 23       01190          INC     HL            ;AFTER COLON
FF61 DD360000 01200          LD      (IX),0        ;LOAD POS WITH ZERO
FF65 13       01210          INC     DE            ;DE POINTS AT NEW LEN
FF66 FD3400   01220          INC     (IY)          ;INCREMENTS VARWK
FF69 18DE     01230          JR      XFER2         ;JP TO INPUT VIA XFER2
              01240 ;
              01250 ;=====================================================
              01260 ;
FF6B FE0D     01270 CARRET   CP      0DH           ;CHECK IF A IS CAR RET
```

```
FF6D 2021       Ø128Ø          JR      NZ,EQUAL        ;IF NOT JUMP TO EQUAL
FF6F DD7EØØ     Ø129Ø          LD      A,(IX)          ;LOAD A WITH POSITION
FF72 FEØØ       Ø13ØØ          CP      Ø               ;COMPARE POS TO ZERO
FF74 28D9       Ø131Ø          JR      Z,DNARR1        ;IF POS=Ø THEN DNARR1
FF76 47         Ø132Ø          LD      B,A             ;LOAD B WITH POS
FF77 AF         Ø133Ø          XOR     A               ;ZERO A REGISTER
FF78 1A         Ø134Ø          LD      A,(DE)          ;LOAD A WITH LENGTH
FF79 9Ø         Ø135Ø          SUB     B               ;A= # OF BLANKS TO END
FF7A 3C         Ø136Ø          INC     A               ;ADD ONE TO BLANKS
FF7B 2B         Ø137Ø          DEC     HL              ;DECREMENT VIDEO POINTER
FF7C Ø62Ø       Ø138Ø          LD      B,2ØH           ;LOAD B WITH BLANK CHAR
FF7E 23         Ø139Ø CR1      INC     HL              ;INCREMENT VIDEO POINTER
FF7F 7Ø         Ø14ØØ          LD      (HL),B          ;WRITE BLANK TO SCREEN;
FF8Ø 3D         Ø141Ø          DEC     A               ;DECEREMENT # OF BLANKS
FF81 2ØFB       Ø142Ø          JR      NZ,CR1          ;IF BLANKS<>Ø THEN CR1
FF83 FD46ØØ     Ø143Ø          LD      B,(IY)          ;LOAD B WITH VARWK
FF86 3ADDFF     Ø144Ø          LD      A,(NUM)         ;LOAD A WITH NUM
FF89 B8         Ø145Ø          CP      B               ;COMPARE VARWK WITH NUM
FF8A 2ØC3       Ø146Ø          JR      NZ,DNARR1       ;IF VARWK<>NUM THEN DNARR1
FF8C 18A9       Ø147Ø          JR      XFER1           ;JUMP TO BEGIN VIA XFER1
                Ø148Ø ;
FF8E 18B9       Ø149Ø XFER3    JR      XFER2           ;RELATIVE JP XFER INST
                Ø15ØØ ;
                Ø151Ø ;==================================================
                Ø152Ø ;
FF9Ø FE3D       Ø153Ø EQUAL    CP      '='             ;CHECK IF A IS EQUAL
FF92 2Ø34       Ø154Ø          JR      NZ,CHAR         ;IF NOT JP TO CHAR
FF94 FD36ØØØ1   Ø155Ø          LD      (IY),1          ;SET VARWK TO 1
FF98 DD2ADBFF   Ø156Ø          LD      IX,(PTR)        ;IX=PTR TO BASIC STR PTR
FF9C 21FF3B     Ø157Ø          LD      HL,3BFFH        ;HL=TOP OF VIDEO MEM-1
FF9F 23         Ø158Ø INC3     INC     HL              ;HL=TOP OF VIDEO MEM
FFAØ 7E         Ø159Ø          LD      A,(HL)          ;LOAD A WITH CHARACTER
FFA1 FE3A       Ø16ØØ          CP      ':'             ;CHECK IF COLON
FFA3 2ØFA       Ø161Ø          JR      NZ,INC3         ;IF NOT JP TO INC3
FFA5 23         Ø162Ø          INC     HL              ;INCREMENT VIDEO POINTER
FFA6 23         Ø163Ø          INC     HL              ;TO POS Ø OF VARIABLE
FFA7 23         Ø164Ø          INC     HL              ;AFTER COLON
FFA8 DD4EØØ     Ø165Ø          LD      C,(IX)          ;C LOADED WITH BASIC LEN
FFAB Ø6ØØ       Ø166Ø          LD      B,Ø             ;B=Ø
FFAD DD23       Ø167Ø          INC     IX              ;IX= PTR TO LO VAR ADDRESS
FFAF DD5EØØ     Ø168Ø          LD      E,(IX)          ;E = LO ADDRESS OF VAR
FFB2 DD23       Ø169Ø          INC     IX              ;IX= PTR TO HI VAR ADDRESS
FFB4 DD56ØØ     Ø17ØØ          LD      D,(IX)          ;D = HI ADDRESS OF VAR
FFB7 EDBØ       Ø171Ø          LDIR                    ;MOVE VIDEO TO VAR MEM
FFB9 3ADDFF     Ø172Ø          LD      A,(NUM)         ;A=TOTAL # OF VARIABLES
FFBC FD46ØØ     Ø173Ø          LD      B,(IY)          ;B=VAR WORKED
FFBF B8         Ø174Ø          CP      B               ;CHECK IF LAST VARIABLE
FFCØ C8         Ø175Ø          RET     Z               ;RET TO BASIC IF LAST
FFC1 FD34ØØ     Ø176Ø          INC     (IY)            ;INCREMENT VARWK
FFC4 DD23       Ø177Ø          INC     IX              ;PTR TO NEXT A$ VARIABLE
FFC6 18D7       Ø178Ø          JR      INC3            ;START NEXT VARIABLE
                Ø179Ø ;
                Ø18ØØ ;==================================================
                Ø181Ø ;
FFC8 F5         Ø182Ø CHAR     PUSH    AF              ;SAVE CHARACTER IN AF
FFC9 DD46ØØ     Ø183Ø          LD      B,(IX)          ;LOAD B WITH POS
FFCC 1A         Ø184Ø          LD      A,(DE)          ;LOAD A WITH LEN
FFCD B8         Ø185Ø          CP      B               ;COMPARE LEN WITH POS
FFCE 2ØØ3       Ø186Ø          JR      NZ,START1       ;IF LEN<>POS JP TO START1
FFDØ F1         Ø187Ø          POP     AF              ;RESTORE CHAR IN AF
FFD1 18BB       Ø188Ø          JR      XFER3           ;JUMP TO INPUT VIA XFER3
```

Program continued

```
FFD3 F1         01890 START1  POP     AF                  ;RESTORE CHAR IN AF
FFD4 77         01900         LD      (HL),A              ;PUT CHAR ON SCREEN
FFD5 DD3400     01910         INC     (IX)                ;INCREMENT POS
FFD8 23         01920         INC     HL                  ;INCREMENT VIDEO POINTER
FFD9 18B3       01930         JR      XFER3               ;JUMP TO INPUT VIA XFER3
                01940 ;
                01950 ;=============================================================
                01960 ;
FFDB 0000       01970 PTR     DEFW    0                   ;PTR TO BASIC VAR PTR
FFDD 00         01980 NUM     DEFB    0                   ;NUMBER OF VARIABLES
0020            01990 LEN     DEFS    32                  ;UP TO 32 VAR LENGTHS
FFFE 00         02000 VARWK   DEFB    0                   ;# OF VAR BEING WORKED
FFFF 00         02010 POS     DEFB    0                   ;INDICATES POS WITHIN VAR
0000            02020         END                         ;END OF ASSY LANGUAGE PGM
00000 TOTAL ERRORS
```

Program Listing 2.

Demonstration BASIC program illustrating the use of the subroutines at 50000 and 60000 to implement the form fillout technique

```
10 REM
20 REM
30 REM   DEMONSTRATION OF FORM-FILLOUT PROGRAM
40 REM   WRITTEN BY:  GARY S. LINDSEY
50 REM                1041 NOGALES AVENUE
60 REM                PALM BAY, FLORIDA 32905
70 REM
80 REM=====================================================
90 REM
100 CLEAR 500
110 DIM A(32),A$(32),L(32)
120 GOSUB 50000
130 CLS
140 PRINT TAB(20) "FORM FILLOUT DEMONSTRATION"
150 PRINT
160 PRINT TAB(10) "1 - PROCESS PERSONAL DATA"
170 PRINT TAB(10) "2 - PROCESS WORK HISTORY DATA"
180 PRINT TAB(10) "3 - PROCESS RECREATIONAL DATA"
190 PRINT
200 INPUT "ENTER YOUR CHOICE:  ";C
210 IF C<1 OR C>3 THEN 130
220 ON C GOSUB 1000,2000,3000
230 GOTO 130
240 REM
250 REM=====================================================
260 REM
1000 REM FIRST SCREEN FOR DEMONSTRATION OF FORM FILLOUT
1010 CLS
1020 PRINT TAB(25) "DISPLAY 1"
1030 PRINT "  DEMONSTRATES DATA ENTRY/EDIT WITHOUT PROTECTED FIE
LDS"
1040 PRINT
1050 PRINT "NAME:  ";N$;TAB(30);"DATE OF BIRTH:  ";B$
1060 PRINT "STREET ADDRESS   :  ";AD$
1070 PRINT "CITY/STATE       :  ";C$
1080 PRINT "PHONE NUMBER     :  ";P$
1090 PRINT
```

```
1100 PRINT "OCCUPATION:  ";O$
1110 N=6:L(1)=20:L(2)=12:L(3)=30:L(4)=30:L(5)=15:L(6)=30
1120 GOSUB 60000
1130 N$=A$(1):B$=A$(2):AD$=A$(3):C$=A$(4):P$=A$(5):O$=A$(6)
1140 RETURN
1150 REM
1160 REM==============================================
1170 REM
2000 REM SECOND SCREEN FOR DEMONSTRATION OF FORM FILLOUT
2010 CLS
2020 PRINT TAB(25) "DISPLAY 2"
2030 PRINT TAB(5)"DEMONSTRATES DATA ENTRY/EDIT WITH PROTECTED FI
ELDS"
2040 PRINT TAB(5)"             INCLUDING NUMERIC VALUES"
2050 PRINT
2060 PRINT "NAME *  ";N$;TAB(30);"EMPLOYEE NUMBER:  ";E
2070 PRINT
2080 PRINT "HIRE DATE:  ";HD$
2090 PRINT "SECTION NUMBER:  ";S
2100 PRINT "YEAR OF DEGREE:  ";Y
2110 PRINT "SALARY:  ";SA
2120 N=5:L(1)=6:L(2)=10:L(3)=5:L(4)=5:L(5)=6
2130 GOSUB 60000
2140 E=VAL(A$(1)):HD$=A$(2):S=VAL(A$(3)):Y=VAL(A$(4)):SA=VAL(A$(
5))
2150 RETURN
2160 REM
2170 REM==============================================
2180 REM
3000 REM THIRD SCREEN FOR DEMONSTRATION OF FORM-FILLOUT
3010 CLS
3020 PRINT TAB(25) "DISPLAY 3"
3030 PRINT  " DEMONSTRATES FORMATTED SCREEN WITH EXTRA COMMENTS/
TITLES"
3040 PRINT TAB(20) "PERSONAL DATA"
3050 PRINT
3060 PRINT "NAME*  ";N$;TAB(30);"PHONE NUMBER*  ";P$
3070 PRINT "ADDRESS*";TAB(10);AD$
3080 PRINT TAB(10);C$
3090 PRINT
3100 PRINT TAB(20) "RECREATIONAL DATA"
3110 PRINT
3120 PRINT "HOBBIES";TAB(10);"NUMBER 1:  ";H1$
3130 PRINT TAB(10)"NUMBER 2:  ";H2$
3140 PRINT "SKILLS";TAB(10)"NUMBER 1:  ";S1$
3150 PRINT TAB(10)"NUMBER 2:  ";S2$
3160 N=4:L(1)=20:L(2)=20:L(3)=20:L(4)=20
3170 GOSUB 60000
3180 H1$=A$(1):H2$=A$(2):S1$=A$(3):S2$=A$(4)
3190 RETURN
3200 REM
3210 REM==============================================
3220 REM
50000 REM SETS Z=BEGINNING OF USR PROGRAM
50010 Z=-318
50020 REM SETS USR PROGRAM ENTRY POINT
50030 DEFUSR0=Z
50040 REM POKES USR ASSEMBLY LANGUAGE PROGRAM INTO MEMORY
50050 RESTORE:FOR I=Z TO Z+280
50060 READ X:POKE I,X
50070 NEXT I
```

Program continued

```
50080 RETURN
50090 DATA 221, 33, 255, 255, 253, 33, 254, 255, 17, 222, 255
50100 DATA 33, 255, 59, 35, 126, 254, 58, 32, 250, 35
50110 DATA 35, 35, 253, 54, 0, 1, 221, 54, 0, 0
50120 DATA 213, 253, 229, 205, 43, 0, 183, 253, 225, 209
50130 DATA 32, 21, 78, 62, 143, 119, 197, 1, 234, 0
50140 DATA 205, 96, 0, 193, 113, 1, 240, 0, 205, 96, 0
50150 DATA 24, 223, 254, 91, 32, 32, 253, 126, 0, 254, 1
50160 DATA 40, 212, 43, 126
50170 DATA 254, 58, 32, 250, 43, 126, 254, 58, 32, 250
50180 DATA 35, 35, 35, 253, 53, 0, 221, 54, 0, 0
50190 DATA 27, 24, 187, 254, 9, 32, 15, 26
50200 DATA 221, 70, 0, 184, 40, 176, 35, 221, 52, 0
50210 DATA 24, 170, 24, 145, 254, 8, 32, 14, 62, 0
50220 DATA 221, 70, 0, 184, 40, 156, 43, 221, 53, 0
50230 DATA 24, 150, 254, 10, 32, 28, 253, 70, 0
50240 DATA 58, 221, 255
50250 DATA 184, 40, 137, 35, 126, 254, 58, 32, 250, 35
50260 DATA 35, 35, 221, 54, 0, 0, 19, 253, 52, 0
50270 DATA 24, 222, 254, 13, 32, 33, 221, 126, 0
50280 DATA 254, 0, 40, 217, 71, 175, 26, 144, 60, 43
50290 DATA 6, 32, 35, 112, 61, 32, 251, 253, 70, 0
50300 DATA 58, 221, 255
50310 DATA 184, 32, 195, 24, 169, 24, 185
50320 DATA 254, 61, 32, 52, 253, 54, 0, 1
50330 DATA 221, 42, 219, 255
50340 DATA 33, 255, 59, 35, 126, 254, 58
50350 DATA 32, 250, 35, 35, 35, 221, 78, 0, 6, 0, 221, 35
50360 DATA 221, 94, 0, 221, 35, 221, 86, 0, 237, 176
50370 DATA 58, 221, 255
50380 DATA 253, 70, 0, 184, 200
50390 DATA 253, 52, 0, 221, 35, 24, 215
50400 DATA 245, 221, 70, 0
50410 DATA 26, 184, 32, 3, 241, 24, 187, 241, 119
50420 DATA 221, 52, 0, 35, 24, 179
50430 REM
50440 REM=================================================
50450 REM
60000 REM POKES NECESSARY NUMBER (N), LENGTHS (L(I)), AND
60010 REM THE POINTER TO A$(1) INTO MEMORY FOR USE BY
60020 REM THE ASSEMBLY LANGUAGE PORTION OF THE PROGRAM.
60030 IF N>32 THEN PRINT "TOO MANY VARIABLES":END
60040 A1=X:A2=I:A3=H:A4=L:A5=V:A6=Z
60050 Z=-318
60060 X=Z+283:POKE X,N:X=X+1
60070 FOR I=1 TO N:POKE X,L(I):X=X+1:NEXT I
60080 FOR I=1 TO N:A$(I)=STRING$(L(I)," "):NEXT I
60090 H=0:L=0:X=VARPTR(A$(1))
60100 IF X<0 THEN X=X+65536
60110 H=INT(X/256)
60120 L=X-H*256
60130 POKE Z+281,L:POKE Z+282,H
60140 X=USR0(X)
60150 X=A1:I=A2:H=A3:L=A4:V=A5:Z=A6
60160 RETURN
```

4

Idiosyncracies in Radio Shack's BASIC

by John Blommers

System Requirements:
Model I
Level II BASIC
or Disk BASIC
16K RAM

Radio Shack's Model I Disk BASIC has some bugs and idiosyncracies you should know. This chapter covers some of them, as well as ways to avoid numerical errors in solving problems.

The IF-THEN statement does not require the word THEN, provided there is no ambiguity in the statement. The following lines work properly:

```
100 A=1 : B=A : C=1 : D=2 : I=9 : J=I
110 IF A=B IF C<>D IF I=J PRINT "HELLO"
```

Since the three comparisons are all true, the word HELLO is printed.

When a comparison statement is executed, it results in a value which is either true (−1) or false (0). For example:

```
99  INPUT "Y,Z";Y,Z
100 X=Y=Z  ' sets X to 0 if Y isn't equal to Z
101 PRINT X ' sets X to  -1 if Y equals Z
102 GOTO 99
```

Compound statements work on a single line, but they also work with the IF-THEN-ELSE statement:

```
100 IF I=2 THEN A=B:C=D:E=F ELSE J=K:L=M:G=71
```

BASIC allows the user to define functions, usually at the beginning of the program. If a run-time error occurs as a result of a problem in the function definition statement, it is the statement calling the function—not the statement that defined it—that is flagged as the incorrect statement.

```
100 DEF FNA(X)=SQRT(X)
200 PRINT FNA(4)
```

This program causes a syntax error in line 200; however, the true error is in line 100 (the function for square root is SQR, not SQRT—that is the actual error).

To suppress the line feed in a PRINT statement, you place a semicolon after the last variable. This fails to work at location 1023, the last location on the screen.

```
100 PRINT @ 1023,"*";  ' still scrolls the screen.
```

BASIC occasionally interprets mathematically identical values differently. The numbers 3328D0 and 33.28D0*100D0 should be identical; however, the following statement results in a nonzero answer:

```
100 PRINT 3328D0-33.28D0*100D0
```

The result is −5.6843418860808 02D−14. The multiplication produces a round-off error in the last digit of the double-precision answer.

Sometimes BASIC can be forced to misinterpret a statement.

```
110 X=1.D0 : Y=2.D0
120 PRINT X+0D-(Y+0D)  ' will print two numbers.
 1   2
```

The confusion arises because the double-precision number 0D should be written 0D0 to help BASIC continue with the arithmetic. Because it thinks that X+0 is one variable and the D−(Y+0D) is a second variable, it prints two numbers. It prints the numbers 3 and 2 for the following:

```
100 PRINT 1+2X+X
```

The following PRINT statement prints a −1:

```
100 PRINT 1=1
```

This happens because 1=1 is a true statement and, therefore, has the value −1 assigned to it.

The VAL function converts a string into a number only if the first character is plus, minus, or a digit from zero to nine. This causes the following, otherwise legal conversions to fail and return a zero value:

```
100 PRINT VAL("  -32")   ' leading blank
110 PRINT VAL("&HFF")    ' in DOS BASIC should give 255
120                      ' but gives 0 instead !!
```

Another bug in the VAL function is that it attempts to convert a long numeric string, even though the extra digits cannot contribute to the precision of the answer. The resulting calculations in BASIC cause an overflow or underflow, giving a run-time error. Consider the following:

```
99 CLEAR 100
100 PRINT VAL("."+STRING$(39,"7"))
```

This program results in an OV error when you use 39, but it works with 38. BASIC supports exponents from −38 to +38.

When typing in DATA statements, you may inadvertently omit a number, leading to unexpected results, because no error message is generated by BASIC to tell you it didn't see any data between the commas. The variable you were trying to READ into the program is set to zero when this happens. Consider the following program:

```
100 FOR I=1 TO 10 : READ A : PRINT A; : NEXT : END
110 DATA 1,2,3,,,6,7,8,9,10 ' 4TH & 5TH ARE MISSING
```

The result of this program is 1 2 3 0 0 6 7 8 9 10.

The word THEN need not be used in the IF-THEN-ELSE statement, as shown in the following program:

```
700 INPUT "ENTER A STRING ";A$
800 IF A$="2000" ELSE PRINT "NOT 2000"
900 IF A$="1000" PRINT "IT'S 1000" ELSE
                 PRINT "NOT 1000"
990 GO TO 700
```

Since BASIC stores real numbers using four bytes in binary notation, certain arithmetic operations give rise to tiny errors. For example:

```
100 A=20.01 : B=20 : PRINT A-B  'prints .0100002.
```

It is true that 20.01 - 20.00 = 0.01, but because BASIC retains only about seven to eight digits of precision, subtracting nearly equal real numbers results in a lot of garbage digits. This result is not peculiar to the TRS-80, but applies to many computers.

It is not generally known that the file input statement INPUT #1, variablelist can be expressed more generally as INPUT #N,variablelist, where N is -1 or -2 for cassette port 1 or 2. This can be used as follows:

```
100 INPUT "WHICH CASSETTE PORT (1 OR 2) ";N
110 IF N<>1 AND N<>2 THEN 100 ELSE N=-N
120 INPUT #N,A$
130 PRINT "THE DATA IS ***" A$ "****"
140 GO TO 100
```

In DOS BASIC, N is a positive number that refers to the number of the file buffer.

BASIC calculates X raised to the Y power internally by the formula EXP(Y*LOG(X)). It does so even if both X and Y are integers. Try raising 2 to the thirteenth power. The answer is 8192.01 instead of 8192. If you know in advance that the calculation should result in an integer, you can obtain the correct result by adding 0.5 to the calculation and using the FIX function to create an integer.

BASIC generally allows you to put spaces wherever you want, ignoring them unless they appear inside quotation marks. This is not true for the TAB function. The following program gives a SUBSCRIPT OUT OF RANGE message:

```
100 PRINT TAB (20) "HELLO" : END
```

This happens because the token for TAB function represents the characters TAB(, rather than the characters TAB. BASIC sees the TAB (20) and assumes that this function refers to the twentieth element of the array

TA. Since the default dimension of all arrays is 10, BASIC gives you an error message.

BASIC prints only six significant figures of a number, even if you use PRINT USING *"#.############"*, as shown in the following program:

```
100 X=1/3
110 Y=0.333333
120 PRINT X,Y
130 IF X=Y PRINT " X AND Y ARE EQUAL "
140 IF X<>Y PRINT "X AND Y ARE UNEQUAL"
```

This prints the following:

```
0.333333            0.333333
X AND Y ARE UNEQUAL
```

Only if Y=0.333 333 33 is entered will X and Y be considered equal by BASIC.

The built-in math functions, such as SIN(X), return single-precision results regardless of the precision of the argument X. If X is less than 100, the SIN function returns an answer correct to at least five decimal places. For X near 1,000,000, the SIN function returns about one digit of accuracy, but for X greater than 100,000,000, the value 0 is returned. The SIN of 1E7 wastes all seven significant digits in specifying the input angle. Here is a summary of results:

X (in radians)	SIN(X)	CORRECT RESULT (6-digit)
100	-0.506368	-0.506366
10 000	-0.30566	-0.305614
1 000 000	-0.382683	-0.349994
10 000 000	0.707107	0.420548
100 000 000	0	0.931639
11 000 000 000	0	0.0681113

The solution to this inaccuracy with the trigonometric functions is to recompute the argument in double precision by subtracting multiples of two times pi. The SIN of the remainder should be fairly accurate. For example, run the following program:

```
100 DEFDBL A-X                      ' variables double precision
110 PI=3.14159265358979323846       ' accurate, eh?
120 INPUT "X=  ";X                  ' this is the argument
130 X1=X/(2.D0*PI)                  ' how many unit circles
140 XX=FIX(X1)                      ' # unit circles to remove
150 X2=X1-XX                        ' fraction of unit circle left
160 X3=X2*2.D0*PI                   ' reduced argument
170 Y=SIN(X3)                       ' accurate result desired
180 PRINT "SIN("X")="Y              ' ok. print answer!
190 GO TO 120                       ' let the human weary itself
```

Printing out the intermediate values X1,XX,X2, and X3 lets the programmer see what is happening at each stage.

The SQR function, while accurate to seven digits, may not return the exact answer. The following program line prints out the word INEXACT if the answer isn't exact:

```
100 IF 5=SQR(25) PRINT "EXACT" ELSE
                 PRINT "INEXACT"
```

There are many perfect squares for which the SQR function gives an inexact result. However, a simple formula based on Newton's Method makes the square root exact, as shown in line 140 below:

```
110 INPUT "ENTER A NUMBER "; N
120 X=SQR(N)
130 PRINT "FIRST ESTIMATE IS REALLY" CDBL(X)
140 Z=(X+N/X)/2  ' one Newton iteration
150 PRINT "SECOND ESTIMATE IS REALLY" CDBL(Z)
160 GO TO 110
```

PRINT SQR(25) prints out a 5 because the TRS-80 shows only six digits.

Other numerical oddities:

```
PRINT    100000+1        prints 100001
PRINT   1000000+1        prints 1E+06
PRINT  10000000+1        prints 10000001
```

But

```
PRINT CDBL(  100000+1)   prints    100001
PRINT CDBL( 1000000+1)   prints   1000001
PRINT CDBL(10000000+1)   prints  10000001
```

The TRS-80 stores single-precision numbers with four bytes, providing seven to eight digits of precision.

Repeatedly adding 0.1 eventually yields an error in the sixth digit.

```
100 FOR I=1 TO 20 STEP 0.1
110 PRINT I;
120 NEXT I        ' results in the following segment:
..... 7.6 7.7 7.79999 7.89999 7.99999 8.09999 8.2 8.3 8.4 ....
```

This happens because 0.1 cannot be represented exactly :

```
PRINT CDBL(0.1) ' yields 0.1000000014901161.
```

On general principle, when accumulating many numbers, do the arithmetic in double precision and convert the final result to single precision as required:

```
110 FOR I=1 TO 200 STEP 1  ' unit stepsize!
120 SUM#=SUM#+0.1D0
120 SUM#=SUM#+0.1D0
130 NEXT
140 PRINT SUM#      'GIVES 20.00000000000001
```

You should never compare real numbers for equality. Compare numbers within the precision possible on the computer, using the following technique:

```
100 TL=1E-7 ' seventh digit of precision
110 INPUT "ENTER X AND Y ";X,Y
```

```
120 IF ABS( (X - Y)/Y) <= TL PRINT "EQUAL"
                              ELSE PRINT "UNEQUAL"
```

If Y is equal to 0, modify the test by removing the division by Y in line 120. If X and Y are the result of other computer calculations, with only five significant digits, then TL = 1E - 5 is appropriate for the comparison.

Sometimes formulas from handbooks, textbooks, or articles are programmed directly without regard to the numerical difficulties that may be encountered. In general, whenever a formula subtracts two numbers, there is a loss of accuracy.

Consider the subtraction of nearly equal numbers. Suppose two single-precision numbers are equal in five places. The difference has only two significant figures. For example:

```
  1234.567
- 1234.523
  ---------
     .0440674     where 0674 = garbage digits
                          44 = two significant figures
```

Now consider the quadratic formula:

```
X = ( - B + SQR(B*B - 4*A*C))/(2*A)
```

The formula loses accuracy when 4*A*C is small compared with B*B. The resulting square root almost equals B, so that the difference in the numerator of the formula causes a loss of accuracy. To correct this, use the formula:

```
X = -2*C/( B + SQR(B¢B - 4*A*C) )
```

A second way to lose accuracy is to add up a mixture of large and small numbers. If possible, sort the numbers in ascending order so that all the small numbers can be summed accurately before adding the remaining large numbers.

Multiplication and division do not produce serious arithmetic errors, except that the answers are truncated to seven or eight significant digits. Therefore, the following lines:

```
100 X = 1/11 : PRINT X,CDBL(X)
110 X = 11*X : PRINT X,CDBL(X)
```

produce

```
.0909091          .09090909361839295
1                 1
```

One divided by 11 equals 0.0909090909090909. . . , an infinitely repeating decimal number. Its base 2 representation also repeats, so any attempt to store this number with a limited number of bytes will be inexact. Only numbers such as 0.5, 0.25, 0.375, 0.125, and 0.0625, which are all sums of negative powers of 2, can be represented exactly in a computer which does base 2 arithmetic. Therefore, the statement PRINT CDBL(0.0125) prints out 1.249999972060323D - 3.

Some formulas are sensitive to small changes in input values. For example, the equation of a line is Y = M∗X + B, where M is the slope of the line, and B is the y-axis intercept of the line. Suppose the slope is very small, say 1E − 6, and B is − 1. The line crosses the x-axis at X = − B/M = 1,000,000. If B is changed to − 1.001, a change of only 0.001, X becomes 1,001,000 (a change of 1000). The reason for the sensitivity is geometrically clear, but there are many other problems, such as curve-fitting, which are called ill-conditioned. The only way to cope with them in a program is to study the program outputs for extreme sensitivity to changes of the input values. To minimize the effects of single-precision arithmetic on such outputs, using double-precision arithmetic may help. In the above problem, if B was computed from other values, using a formula which included the subtraction of nearly equal numbers, the resulting X value would be very inaccurate.

In many statistical packages, the average (or mean) and standard deviation are computed after all the data has been input and converted to various sums, as follows:

```
90  DEFSNG A-Z                    'keep everything single precision
100 READ N                        'number of data points
110 S=0 : SS=0                    'zero these sums
120 FOR I=1 TO N
130 READ X                        'read in an input value
140 S=S+X                         'sum of the data points
150 SS=SS+X*X                     'sum of squares of data points
160 NEXT I
170 M=S/N                         'mean of the data points
180 SD=SQR((SS-S*S/N)/(N-1))      'standard deviation
190 PRINT "MEAN=" CDBL(M))        " STND DEV=" SD
200 DATA 3
210 DATA 6666123, 6666246, 6666369
220 END
```

The program prints out MEAN = 6666246 STND DEV = 0. The correct answers are MEAN = 6666246 STND DEV = 123. The arithmetic error occurs in line 180, where nearly equal quantities are subtracted, losing most of the seven significant figures. There are two solutions to this problem. You can convert all the variables to double precision or you can rewrite the program based on an alternative method of calculating mean and standard deviation:

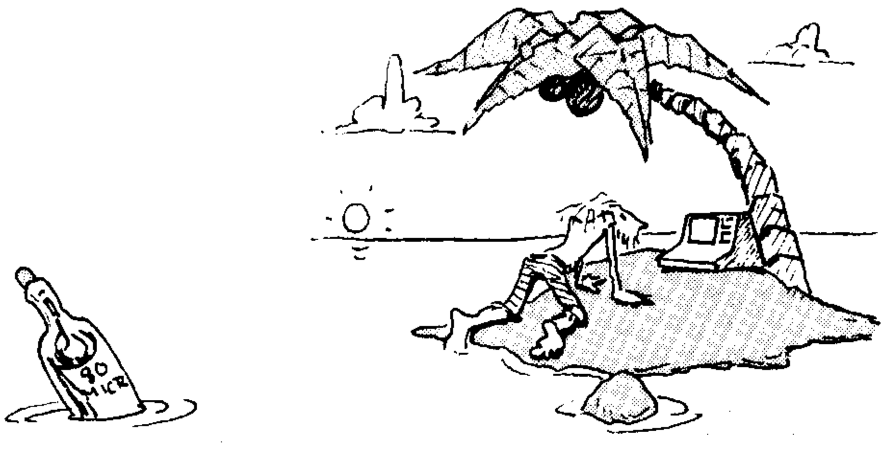
```
100 DEFSNG A-Z                    'all single precision
110 READ N                        'number of data points
120 S=0 : SS=0                    'zero the sums
130 FOR I=1 TO N                  'process n inputs
140 READ X                        'read in a data point
150 S=S+X                         'form only the sum of the points
160 NEXT I
170 M=S/N                         'mean of the points
180 RESTORE                       'reset the pointer to the data
190 READ N                        'input number of points
```

```
200 FOR I=1 TO N                    'process N inputs again
210 READ X                          'read A point
220 SS=SS+(X-M)*(X-M)               'sum of squared diff from mean
230 NEXT I
240 SD=SQR(SS/(N-1))
250 PRINT "MEAN = "M " STND DEV =" SD
260 DATA 3
270 DATA 6666123,6666246, 6666369
280 END
```

The following program calculates the best line through a set of N given points. It too suffers from arithmetic errors (in lines 200 and 210). The correct answers are M = 1 and B = −660,000. The program produces the values 1.52588E−5 and −10.1624:

```
100 DEFSNG A-Z                      'let all be single precision
110 READ N                          'N=number of point pairs
120 XY=0 : SX=0 : SY=0 : S2=0       'zero these sums
130 FOR I=1 TO N                    'process N points
140 READ X,Y                        'read a point pair
150 XY=XY+X*Y                       'sum of X*Y products
160 SX=SX+X                         'sum of the X values
170 SY=SY+Y                         'sum of the Y values
180 S2=S2+X*X                       'sum of X squared values
190 NEXT I
200 M=(XY-SX*SY/N)/(S2-SX*SX/N)
210 B=(SY-M*SX)/N
220 PRINT "THE BEST LINE THROUGH"N;
230 PRINT "DATA POINTS IS : Y=" M "*X+" B
240 STOP
250 DATA 3
260 DATA 665999,-1, 666000,0, 666001,1
```

If the DATA in line 260 were 999, −1, 1000,0, 1001,1, the program would correctly calculate M = 1 and B = −1000.



5

The Hidden Sort Routine

by Tom Mueller

System Requirements:
Model I, II, or III
Disk BASIC
16K RAM
One disk drive

You can sort numbers at incredible speeds using a machine-language routine already built into your TRS-80. All you have to do is fool the computer into thinking the numbers are line numbers in a program. It already has a BASIC sort function for line numbers, designed to allow the insertion of program lines. If you have a disk drive, you can easily adapt this ability to sorting whole numbers between 0 and 65,535. Duplicate numbers are lost, because BASIC does not allow a program to have identical line numbers.

Program Listing 1 generates 100 random numbers and saves them to disk in ASCII. It fools the computer into thinking that the random numbers in the SORT/DAT file are program line numbers. The computer then puts the line numbers into order.

The asterisk in line 100 of Program Listing 1 is required because the computer drops any line numbers not followed by a function, command, or character. Later, line 110 in Listing 2 removes the asterisk from the number and prints the number on the screen. Line 100 in Listing 1 also converts the random number into a non-numeric string. This lets you add one character before saving it to the disk file. Program Listing 3 lets you type in and sort your own numbers. After the prompt, type your numbers, pressing ENTER after each one. To stop, press ENTER without typing a number. The computer continues writing to the disk until the maximum number of entries is reached. If you enter a number less than 1 or greater than 65,535, the computer displays ERROR and asks you again to enter your number.

Program Listing 1

```
10 REM                  LISTING NUMBER 1
20 REM                  TOM MUELLER, PHOENIX, ARIZ.
30 REM
40 REM                  ME=MAX. NUMBER OF ENTRIES
50 ME=100
60 REM                  OPEN SEQUENTIAL FILE TO STORE NUMBERS IN

70 OPEN"O",1,"SORT/DAT"
80 REM                  GENERATE RANDOM NUMBERS
90 FOR I = 1 TO ME
100 A$=STR$(RND(9999))+"*"
110 PRINT LEFT$(A$,LEN(A$)-1)
120 REM                 WRITE RANDOM NUMBER ON DISK
130 PRINT #1, A$
140 NEXT I : CLOSE
150 REM                 DONE
160 PRINT"TYPE IN  LOAD ''SORT/DAT''  AND PRESS <ENTER>"
170 PRINT"THEN TYPE IN   SAVE ''SORT/DAT'',A  AND PRESS <ENTER>"

180 PRINT
190 PRINT"THE SORT IS DONE..."
200 PRINT"TO SEE THE NUMBERS,"
210 PRINT"TYPE IN THE SECOND PROGRAM (LISTING #2) AND"
220 PRINT"TYPE  RUN"
```

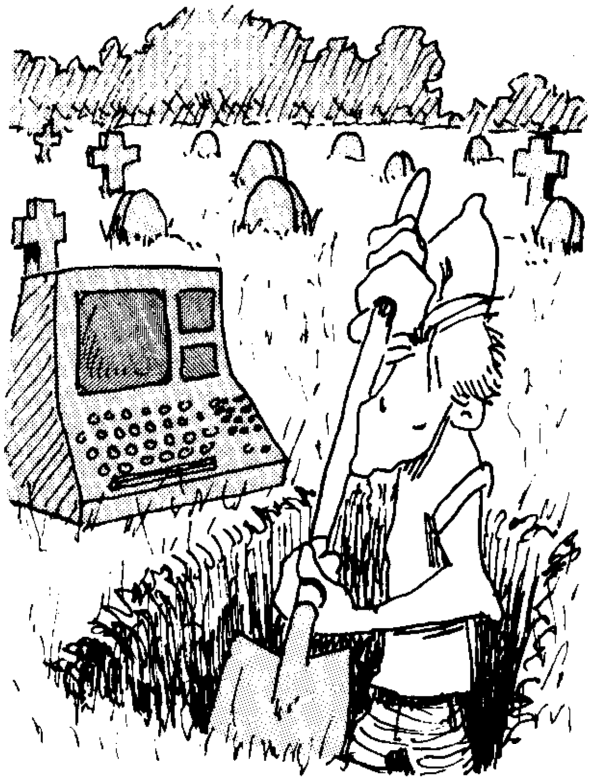
Program Listing 2

```
10 REM                  LISTING NUMBER 2
20 REM                  TOM MUELLER, PHOENIX, ARIZ.
30 REM
40 REM                  OPEN SEQUENTIAL FILE TO LOAD
50 REM                  SORTED NUMBERS BACK INTO COMPUTER.
60 OPEN"I",1,"SORT/DAT"
70 REM                  CHECK FOR END OF LIST
80 IF EOF(1) THEN CLOSE : END
90 INPUT #1, A$
100 REM                 DISPLAY NUMBER
110 PRINT LEFT$(A$,LEN(A$)-1)
120 GOTO 80
```

Program Listing 3

```
10 REM                  LISTING NUMBER 3
20 REM                  TOM MUELLER, PHOENIX, ARIZ.
30 REM
40 REM                  ME=MAX. NUMBER OF ENTRIES
50 ME=100
60 OPEN"O",1,"SORT/DAT"
70 REM                  ENTER NUMBERS HERE
80 FOR I = 1 TO ME
90 N$="" : INPUT "TYPE IN A NUMBER OR JUST PRESS <ENTER> TO QUIT
" ; N$
```

```
100 IF N$="" THEN 130
110 IF VAL(N$)<1 OR VAL(N$)>65535 THEN PRINT "ERROR" : GOTO 90
120 N$=N$+"*" : PRINT #1, N$ : NEXT I
130 CLOSE
140 PRINT"TYPE IN   LOAD''SORT/DAT''  AND PRESS <ENTER>"
150 PRINT"THEN TYPE IN  SAVE''SORT/DAT'',A   AND PRESS <ENTER>"
160 PRINT
170 PRINT"TO DISPLAY THE SORTED LIST, RUN THE PROGRAM SHOWN"
180 PRINT"IN LISTING NUMBER 2."
```



6

Testing Shell-Metzner Sort Routines

by T. W. Cleghorn

System Requirements
Model I or III
Level II BASIC
16K RAM

Recently I was looking for a good sorting algorithm to replace a binary search technique overloaded with increases in file size. I do not have a standard sort product for my TRS-80, so each application with data to sort must include its own sort routine. A diligent search of *80 Micro* revealed three programs that include Shell-Metzner sorts.

Doug Walker's "Beyond Shell Metzner," (*80 Microcomputing*, September 1980) provides an easy test data source, with its parameter driven, random string generator. Swapping pointers rather than swapping strings seems promising. Walker's program includes several other options: multiple field selection, ascending or descending sequences, and numeric or string data types. Stewart E. Fason's review of the B17 product (*80 Microcomputing*, March 1981) includes a driver test program with a variation on Shell-Metzner. Thomas C. Mehesan, Jr.'s "The Spare Time Generator" (May 1981) is also based on Shell-Metzner.

I defined and programmed a validation and timing experiment to prove and measure the three techniques. I report the results of my testing here. All times are in seconds, measured by the internal clock. I identify the three Shell-Metzner (S-M) algorithms by the author's name.

Binary vs. Fason

First to fall was the binary search algorithm. I used a production data file on disk as a data source and selected a varying number of data records from the file for several runs. I sorted the same set of records with each technique. It was no contest. The Fason S-M outperformed my binary technique by more than three to one at the 50-record level and reached eight to one at the 125 record mark.

Fason vs. Walker

I added the Fason algorithm to the Walker random string generation program. Fason won by a technical knock-out. I completed two runs with the same 50 records sorted by each algorithm. The sorting times, in seconds, are shown below.

Round	Fason	Walker	Ratio W/F
1	17	145	8.5/1
2	19	179	9.4/1

The Walker VARPTR swap routine hit an address greater than 32767 and died with a field overflow error. Since Walker was trailing by nine to one, I stopped the bout.

Mehesan vs. Fason

"The Spare Time Generator" by Mehesan includes a trim S-M algorithm in 15 lines of BASIC. I constructed another program combining Walker's data generator, the Mehesan algorithm and the Fason dreadnought and began a new set of time trials. Another clean sweep for Fason: after ten rounds the score was 1.36 seconds per record for Fason and 3.61 seconds per record for Mehesan (see Figure 1).

These statistics reveal a non-linear increase in sorting time per record as the number of records increase. I assumed this was due to the TRS-80 string packing algorithm. To test this theory I collected, combined and modified the various test programs used until now and developed the parameter-controlled sort test program SORT/TST (see the Program Listing).

		Fason		Mehesan	
Round	Records	Seconds	Seconds Per Rcd.	Seconds	Seconds Per Rcd.
1	10	2	0.2000	2	0.2000
2	25	6	0.2400	9	0.3600
3	50	14	0.2800	22	0.4400
4	75	27	0.3600	45	0.6000
5	100	52	0.5200	74	0.7400
6	100	54	0.5400	86	0.8600
7	125	71	0.5680	124	0.9920
8	150	110	0.7333	205	1.3667
9	200	351	1.7550	955	4.7750
10	500	1,129	2.2580	3,299	6.5980
Totals	1,335	1,816	1.3603	4,821	3.6112

Figure 1. *Fason vs. Mehesan*

Seconds Per Record

Run No.	Records	Fason String	Fason Floating Point	Fason Integer	Mehesan String	Mehesan Floating Point	Mehesan Integer
1	25	.32	.32	.32	.32	.32	.32
2	50	.34	.39	.40	.62	.50	.46
3	100	.51	.39	.40	1.08	.65	.57
4	150	.82	.4333	.4333	1.633	.6867	.6467
5	200	1.665	.46	.43	4.445	.75	.70
Ratio 5/1	8	5.2	1.43	1.34	13.89	2.34	2.18

Figure 2. *Data Type Analysis*

Statements		Function
10	70	Identification and prologue
80	210	Run parameters description
220		Run parameters—DATA statement
230	350	Interpret run parameters, establish the run environment
360	520	Generate the test data
530	540	Pass identification
550	610	Time and invoke Sort One
620	650	Reset the data array
660	720	Time and invoke Sort Two
730		Get parameter and test for another pass
740	750	End of job
760	810	Subroutine to verify sort results
820	850	Clock interface subroutine
860	870	Line print output
4000		Entry point to Sort One
5000		Entry point to Sort Two

Figure 3. *SORT/TST Structure*

Position	Form, Function and Use
1	A number that specifies the number of bytes to be reserved for strings, this number becomes the operand of a Clear
2	To specify Data Type, 0 = String, 1 = Numeric
3	0 = Integer, or 1 = Floating Point Numbers. Tested only if parameter 2 is a 1
4	1 = Print to Line Printer, 0 = Video Display only
5-N	This series defines the number of records per pass and the number of passes. The program terminates when a zero is read from this series.

Figure 4. *SORT/TST parameters*

SORT/TST

SORT/TST, a parameter driven program, facilitates the testing and measurement of two sorting algorithms. SORT/TST generates test data, constructs data arrays, invokes and times two sorting algorithms, and reports the results to the video display and, optionally, to the line printer. The structure of SORT/TST is shown in Figure 3. Line 220 contains all the run parameters used by SORT/TST to perform a five-pass integer data test and write the results to the line printer. The run parameters are positional; they are read and interpreted based on the order in which they appear in the data statement. Their meanings are tabulated in Figure 4.

GOSUB statements at lines 570 and 680 invoke the two routines under test. Array T stores the data to be sorted, and integer variable N contains the number of records. The sort under test must be written as a closed routine that retains control until array T is sorted into ascending sequence and exits via a RETURN statement. A numeric array, B, is available for use by the sort under test. Other variables that may be tested but not changed are listed in Figure 5.

Floating Numbers

I defined SORT/TST run parameters to perform five passes, varying the number of records from 25 to 200 with each pass. I ran SORT/TST three times, setting the data type option to string, floating point numeric, and integer. The results of three runs of five passes each are shown in Figure 2. These measures support the theory that the string-packing routine accounts for the non-linear increases in sort time for string data.

With an eight-fold increase in records between runs one and five, the time per record for strings increased 5.2 times (Fason) and 13.89 times (Mehesan), while the integer time increased 1.43 times and 2.18 times, respectively.

Variable	Contents
A, AX, N	Number of records
D	Parm 1
KT!	Start Time in seconds
P2	Parm 2
P3	Parm 3
P4	Parm 4
PN	Current Pass Number
T1	Storage Array for Test Data

Figure 5. *Reserved variables*

Program Listing

```
10 ' PROGRAM "SORT/TST", TO COMPARE SORT TIME OF TWO
20 ' SORTING ALGORITHMS. THIS PARAMETER DRIVEN PROGRAM
```

Program continued

```
30 ' GENERATES THE TEST DATA, TIMES, CALLS, AND VERIFIES THE
40 ' RESULTS OF TWO SORTING ROUTINES AGAINST THE SAME DATA.
50 ' PARAMETERS ARE DEFINED BY THE FOLLOWING DATA STATEMENT
60 ' BY TWC COMPUTING,
70 ' HOUSTON, TEXAS
80 ' ************************************************************
90 ' THE PARAMETERS, VALUES, AND USE ARE AS FOLLOWS
100 ' D = NUMBER OF BYTES OF STRING DATA TO RESERVE
110 ' P2; Ø = GENERATE AND SORT CHARACTER STRINGS
120 '      1 = GENERATE AND SORT NUMERIC DATA
130 ' P3: (TESTED ONLY IF P2 = 1)
140 '      1 = SINGLE PRECISION FLOATING POINT NUMBERS
150 '      Ø = INTEGER NUMBERS
160 ' P4: 1 = PRINTED SORT STATISTICS ON LINE PRINTER
170 '      Ø = STATISTICS TO DISPLAY ONLY
180 ' A(1) THRU A(N)  = NUMBER OF RECORDS TO GENERATE AND
190 '    NUMBER OF PASSES, TERMINATED WHEN A(N) = Ø
200 ' ************************************************************
210 '     D,      P2,     P3,     P4,     A(1) THRU A(N)
220 DATA 1000, 1,      Ø,      1,     25,5Ø,1ØØ,15Ø,2ØØ,Ø
230 CLS: DEFINT A-S,U-Z
240 READ D
250 CLEAR D
260 READ D,P2,P3,P4,AX
270 ' GET HIGHEST REQUESTED RECORD COUNT FOR ARRAY SET
280 READ A: IF A > AX THEN AX = A
290 IF A > Ø THEN 280
300 RESTORE: READ D,P2,P3,P4,A ' RESET PARMS
310 IF P4=1 THEN LPRINT"SORT TEST, PARMS =";D;P2;P3;P4;",";
320 IF P4=1 THEN IF P2=1 THEN LPRINT"NUMERIC DATA"; ELSE LPRINT"
STRING DATA"
330 IF P2 = 1 THEN 340 ELSE DEFSTR T: GOTO 360
340 IF P3 = 1 THEN DEFSNG T ELSE DEFINT T
350 IF P4=1 THEN IF P3=1 THEN LPRINT", FLOATING POINT" ELSE LPRI
NT", INTEGER"
360 ' GENERATE RANDOM SORT RECORDS
370 ' ************************************************************
380 DIM T(AX), T1(AX), B(AX)
390 N=A: FOR I = 1 TO A
400   IF P2 = Ø THEN 430 ELSE T=RND(32767)
410   IF P3 = 1 THEN T = T * RND(10)
420   GOTO 500
430 ' GENERATE CHARACTER STRINGS
440   T=""
450   B = RND(8)
460   FOR C = 1 TO B
470     T = T+CHR$(RND(26)+64)
480   NEXT C
490   T = T + " INPUT REC."+STR$(I)
500   T(I)=T
510   T1(I)=T
520 NEXT I
530 PN=PN+1: PRINT "*****  PASS  *****";PN;",";N;"RECORDS"
540 IF P4 = 1 THEN LPRINT:LPRINT "*****  PASS  *****";PN:LPRINT
550 GOSUB 820: KT!=TX!  ' START TIME
560 ' ************** >>>>>  SORT NO. 1  <<<< ********************

570 GOSUB 4000 '   <<<<<   PERFORM SORT NO. 1
580 GOSUB 820: PRINT "SORT NO. 1 ELAPSED TIME = ";(TX!-KT!);"SEC
ONDS"
590 IF P4=1 THEN LPRINT"SORT NO. 1 ELAPSED TIME ="(TX!-KT!);"SEC
ONDS"
```

```
600 IF P4=1 THEN GOSUB 860
610 GOSUB 760
620 ' NOW TIME AND PERFORM SORT NO. 2
630 FOR I = 1 TO A      ' RESTORE T ARRAY
640   T(I) = T1(I)
650 NEXT I
660 GOSUB 820: KT1=TX1 ' RECORD START TIME
670 ' ***************** >>>>  SORT NO. 2   <<<< ***************
680 GOSUB 5000  ' <<<<<   PERFORM SORT NO. 2
690 GOSUB 820  ' RECORD STOP TIME
700 PRINT "SORT NO. 2 ELAPSED TIME = ";(TX1-KT1);"SECONDS"
710 IF P4=1 THEN LPRINT"SORT NO. 2 ELAPSED TIME =";(TX1-KT1);"SE
CONDS": GOSUB 860
720 GOSUB760     ' TEST SORTED FILE
730 READ A: IF A < 1 THEN 740 ELSE 390
740 IF P4 = 1 THEN LPRINTCHR$(12);
750 END
760 ' TEST SORTED RESULTS
770 FOR I = 1 TO A-1
780   IF T(I) <= T(I+1) THEN 800
790   PRINT T(I);" > > > ";T(I+1)
800 NEXT I
810 RETURN
820 ' SUB-ROUTINE TO GET TIME IN SECONDS
830 X1$=RIGHT$(TIME$,8)          ' HH/MM/SS
840 TX1=(VAL(LEFT$(X1$,2)) * 3600) + (VAL(MID$(X1$,4,2)) * 60) +
 (VAL(RIGHT$(X1$,2)))
850 RETURN
860 LPRINT"SORTED";N;"RECORDS, AT";:LPRINTUSING"##.####";((TX1-K
T1)/N);:LPRINT" SECONDS PER RECORD"
870 RETURN
880 ' *************** END OF SORT/TST ****************
890 ' START SORT NO. 1 AT STATEMENT NO. 4000 AND SORT NO. 2 AT S
TATEMENT NO. 5000
4000 ' FROM "B17", STEWART E. FASON,
4010 ' 80/MICROCOMPUTING, MAR. '81
4020 L=1
4030 B(L) = N+1
4040 M=1
4050 J=B(L)
4060 I=M-1
4070 IF J-M < 3 THEN 4270
4080 M1= INT(RND(0)*(J-M))+M
4090 I = I + 1
4100 IF I = J THEN 4190
4110 IF T(I) <= T(M1) THEN 4090
4120 J = J - 1
4130 IF I = J THEN 4190
4140 IF T(J) >=T(M1) THEN 4120
4150 T = T(I)
4160 T(I) = T(J)
4170 T(J) = T
4180 GOTO 4090
4190 IF I >= M1 THEN I = I - 1
4200 IF J = M1 THEN 4250
4210 T = T(I)
4220 T(I) = T(M1)
4230 T(M1) = T
4240 L = L + 1
4250 B(L) = I
4260 GOTO 4050
4270 IF J-M < 2 THEN 4320
```

Program continued

```
4280 IF T(M) < T(M+1) THEN 4320
4290 T = T(M)
4300 T(M) = T(M+1)
4310 T(M+1) = T
4320 M = B(L) + 1
4330 L = L -1
4340 IF L > 0 THEN 4050
4350 RETURN
5000 ' "THE SPARE TIME GENERATOR", THOMAS C. MEHESEN, JR.
5010 ' 80/MICROCOMPUTING, MAY, 1981, PAGE 264
5020 M=N
5030 M = INT(M/2)
5040 IF M = 0 THEN RETURN
5050 L = 1: M1 = N - M
5060 I=L
5070 J = I + M
5080 IF T(I) <= T(J) THEN 5110
5090 T = T(I): T(I) = T(J): T(J) = T
5100 I = I - M: IF I < 1 THEN 5110 ELSE 5070
5110 L = L + 1: IF L>M1 THEN 5030 ELSE 5060
```

7

ASCII Converter

by David D. Busch

System Requirements:
Model I or III
Level II BASIC
16K RAM

What's ASCII, and why must it be converted? The answer lies in the different ways computers and humans process information.

People can handle mixtures of alpha and numeric characters, but computers recognize only binary numbers—ones and zeros. When string data is fed to a TRS-80, it must be converted to a series of numbers that the processor can handle. ASCII, or American Standard Code for Information Interchange, is one standard of communication that allows computers to exchange alphanumeric information in a form common to processors with differing operating systems and languages.

But, even if you have no modem, and aren't communicating with other computerists, it is often necessary to translate a string into the corresponding ASCII code, or vice versa. In some cases, when only a few characters need converting, a table of codes and their string values will do. Other times, longer messages need deciphering.

One good application for ASCII characters in programs is in gamewriting. Writers of BASIC adventure programs may wish to hide messages from those casually listing the program. The CHR$(n) function can assign the desired string values to string variables called at appropriate points in the program. CHR$(n) returns a one-character string that corresponds to the ASCII code of n. For example, PRINT CHR$(65) produces an uppercase A on the screen.

A BASIC adventure might use a message such as: "Look in the hollow stump." This hint could be labeled H1$, and concatenated using CHR$(n), and the ASCII codes:

```
100 DATA 76,111,111,107,32,105,110,32,116,104,101,32,104,111,108,108,
        111,119,32,115,116,117,109,112,32
110 FOR N=1 TO 25:READ A
120 H1$=H1$+CHR$(A)
130 NEXT A
```

Use additional DATA lines and FOR-NEXT loops to put any number of messages into difficult-to-read-accidentally string variables. Of course, any knowledgeable programmer can pick the BASIC game apart, or enter PRINT H1$ from command mode once the program has been run past the initialization point. The object of this technique, however, is to protect the game player who innocently lists the program and wants to save the fun.

The same method can hide program credits within BASIC code. ASCII Converter (see Program Listing) accepts any keyboard input (in batches up to 255 characters each) and translates the alphanumerics into ASCII code, or converts an ASCII message back into the equivalent string. Those with Disk BASIC, Model III BASIC, or some other patch for the TRS-80 with LINEINPUT can best use this program. LINEINPUT allows you to feed any keyboard character, including commas and quotes, into the string to be translated into ASCII. If you have no LINEINPUT you may still be able to use this program; try changing the LINEINPUT in line 420 to a simple INPUT. Remember to use no string delimiters in your message.

ASCII Converter allows you to input a message of up to 100 lines, stored in a string array, MESS$(n), which is dimensioned in line 50. When you run the program, your instructions are displayed, and you choose video output only, or output to both the screen and a printer. (If printer output is desired, PFLAG is set to a value of 1 in line 250.)

The next menu displayed (lines 260–350) allows you to convert a string to ASCII code, or translate a series of ASCII numbers into the equivalent string. Your choice sets another flag, CH, to a value of either one or two in line 340. The ASCII or string message is input within a FOR-NEXT loop at lines 370–450. Be careful to enter the message correctly, with no typographical errors. The program does not differentiate between string and ASCII codes at this point; and, therefore, there is no check to see if ASCII numbers, if applicable, are valid (less than 255). It's possible to input nonsense. Because spaces (CHR$(32)) indicate the end of a group of ASCII numerals, a space is added to the end of input in line 440, if the user excludes it.

If the first three characters of the data input equal the string 999, signaling the end of message input, control exits from the loop at line 430 to the parsing subroutine at lines 470–680. Here, a FOR-NEXT loop is initiated at line 470 and performed as many times as the number of message lines that have been input, less one (FOR N1=1 to N−1). The final message line input will be 999, and should not be translated.

As each line of MESS$(n) is brought on line, a second loop, nested within the first, looks at each character and assigns its value temporarily to A$ (line 490). When the CH flag does not equal two, it indicates that the line is an ASCII series to be decoded, and control drops down to line 510, where the ASCII value of the character is returned. This number is listed to the screen, followed by three spaces, and, if PFLAG = 1, it is also sent to the printer. At this point, control branches to line 620, where N2 is incremented by one, signaling the examination of the next character in the string.

Should CH = 2, a different series of operations is performed on the character in A$. At line 520, the program looks to see if A$ is a space (CHR$(32)), in which case it knows that the parsed ASCII group is complete. Spaces are used to delimit the groups, because the ASCII code input may consist of one, two, or three numbers.

If A$ does not equal CHR$(32), then at line 550 it is concatenated onto the end of N3$. Working from left to right, the string representation of each of the numerals in a given code group is added to N3$ until it is complete. Then, the value of the string is assigned to the variable B (line 530), and CHR$(B) is listed to the screen, or printed. Then N3$ is nulled, and the next group of ASCII code is examined.

Program Listing

```
10 '      ***********************************************
          *                                             *
          *               ASCII CONVERTER               *
20 '      *                                             *
          *               DAVID D. BUSCH                *
30 '      *               515 E. HIGHLAND AVE.          *
          *               RAVENNA, OHIO 44266           *
          *                                             *
          ***********************************************


40 CLEAR 4000
50 DIM MESS$(100)

60 '****** INSTRUCTIONS ******

70 CLS
80 PRINT
90 PRINT
100 PRINT "     ###### ASCII CONVERTER ******"
110 PRINT
120 PRINT "THIS PROGRAM WILL CONVERT ANY CHARACTER OR "
130 PRINT "MESSAGE INPUT FROM THE KEYBOARD TO ITS ASCII"
140 PRINT "CODE.  IT WILL ALSO CONVERT ASCII CODE"
150 PRINT "BACK INTO A STRING.  ENTER SPACES BETWEEN"
160 PRINT "WORDS OR NUMBER PAIRS, AND ENTER '999'"
170 PRINT "WHEN FINISHED."
```

Program continued

```
180 PRINT
190 PRINT
200 PRINT "DO YOU WANT YOUR MESSAGE ALSO"
210 PRINT "DIRECTED TO LINE PRINTER?"
220 PRINT
230 PRINT " Y/N";
240 INPUT AN$
250 IF LEFT$(AN$,1)="Y" THEN PFLAG=1

260 CLS
270 PRINT
280 PRINT
290 PRINT "DO YOU WANT TO :"
300 PRINT "        1.) CONVERT STRING TO ASCII CODE"
310 PRINT "        2.) CONVERT ASCII CODE TO STRING EQUIVALENT"
320 PRINT
330 INPUT " ENTER CHOICE :";CH$
340 CH=VAL(CH$)
350 IF CH<1 OR CH>2 GOTO 330
360 '****** ENTER CHARACTERS OR MESSAGE ******

370 :  FOR N=1 TO 100
380 :    CLS
390 :    PRINT
400 :    PRINT
410 :    PRINT "ENTER LINE OF MESSAGE   (TYPE 999 WHEN FINISHED ) :"
420 :    LINEINPUT MESS$(N)
430 :    IF LEFT$(MESS$(N),3)="999" GOTO 470
440 :    IF RIGHT$(MESS$(N),1)<>CHR$(32) THEN
         MESS$(N)=MESS$(N)+CHR$(32)
450 :  NEXT N

460 '****** PARSE CHARACTER OR MESSAGE ******
'
470 :  FOR N1=1 TO N-1
480 :    FOR N2=1 TO LEN(MESS$(N1))
490 :      A$=MID$(MESS$(N1),N2,1)
500 :      IF CH=2 GOTO 520
510 :      A=ASC(A$):
           PRINT A;"    ";:
           IF PFLAG=1 THEN LPRINT A;"    ";:
           GOTO 620
520 :      IF A$<>CHR$(32) GOTO 550
530 :      B=VAL(N3$):
           PRINT CHR$(B);:
           IF PFLAG=1 THEN LPRINT CHR$(B);
540 :      GOTO 600
550 :      N3$=N3$+A$
560 :      NU=NU+1
570 :      IF NU=2 GOTO 620
580 :      IF A$=CHR$(32) GOTO 620
590 :      GOTO 620
600 :      NU=0
610 :      N3$=""
620 :    NEXT N2
630 :    PRINT
640 :    INPUT "HIT ENTER WHEN READY FOR NEXT LINE";A$
650 :    CLS
660 :    PRINT
670 :  NEXT N1
680 PRINT"END OF MESSAGE(S)" : END
```

8

Capture RST 10H: Customized BASIC Commands

by James Cargile

System Requirements:
Model I
Level II BASIC
Editor/assembler

The problem: you want a clear and easy-to-use link between a BASIC program and an assembly-language program that does not impair functioning of the BASIC interpreter or any of the ROM or Disk BASIC routines. One solution is to use RST 10H, otherwise known as the parser, to implement customized commands.

RST 10H is one of seven, 1-byte calls to the ROM. On execution of RST 10H, a return address is pushed into the stack, and the program counter is loaded with the address 0010H for execution. At 0010H in the ROM is a jump to user RAM address 4003H. At 4003H is a jump to the ROM routine at address 1D78H. The routine at 1D78H increments the HL register pair, loads the byte addressed by HL into the A register, and sets flags to indicate whether the byte contains a colon or null (indicating end of a BASIC text line), or a digit zero to nine, or other characters such as letters. The annotated code for the subroutine at 1D78H is shown in Figure 1. RST 10H is used frequently by other ROM routines because it is a 1-byte call and because it is useful in deciphering strings of text. Specifically, it is called by the keyboard driver, text editor, and BASIC interpreter portions of the ROM. Since RST 10H vectors to a user RAM address at 4003H it may be temporarily diverted to assembly-language routines by replacing the jump address to 1D78H with the address of any desired user routine. The only registers that must be controlled are the HL register pair, which usually points to the position in a BASIC text line, and the stack pointer, which contains a return to the routine calling RST 10H.

```
; ************** RST1Ø  *********************
;
;
L1D78   INC     HL              ;INCREMENT READ POINTER
        LD      A,(HL)
        CP      3AH             ;IS CHARACTER A COLON?
        RET     NC              ;RETURN IF COLON OR
                                ;IF THIS IS A CHARACTER
        CP      2ØH             ;IS THIS A BLANK?
        JP      Z,L1D78H        ;IF SO, SKIP IT
        CP      ØBH
        JR      NC,1D8BH        ;CANNOT BE A NULL
                                ;GO CHECK FOR DIGIT
        CP      Ø9H
        JP      NC,1D78H        ;SKIP Ø9H AND ØAH
L1D8B   CP      3ØH             ;IF THIS A ZERO?
        CCF                     ;SET CARRY IF 'Ø' TO '9'
                                ;RESET CARRY IF NULL
        INC     A               ;RESET Z FLAG IF 'Ø'
        DEC     A               ;SET Z FLAG IF NULL
        RET
; ********** EXIT CONDITIONS ********************
;
; ** ZERO FLAG  *****  CARRY FLAG  ******  RESULT **
;     RESET       *       RESET       *  CHARACTER
;     SET         *       RESET       *  ':' OR NULL
;     RESET       *       SET         *  'Ø' TO '9'
; ******************************************************
```

Figure 1. *Annotated code for the subroutine at 1D78H*

Because of these features, RST 10H can easily intercept BASIC program execution, divert to a user subroutine, execute it, and return to the BASIC interpreter without the bother of DEFUSR and USR, error-trapping techniques, or use of BASIC tokens.

RST 10H *does* present three problems—all easily solved. Perhaps the most obvious problem is that you don't want to execute the assembly-language routine while entering or editing program text. The problem is solved by accepting only the calls to RST 10H from ROM address 1D5BH. The second problem may occur if your assembly-language routine uses RST 10H or calls a ROM routine which uses RST 10H. To prevent endless looping or stack overflow, give control of RST 10H back to the ROM before entering the user routine. After exiting the user routine, recapture RST 10H for further use. Finally, since you relinquish control of RST 10H upon entry to the user routine, any error in the user routine causes an exit to the BASIC interpreter, with no control over RST 10H. A simple error processor that recaptures RST 10H prior to processing the error eliminates this problem.

As an example of the potential uses of RST 10H, I have included portions of the code for a program MATRIX, which provides a set of matrix operations commands for use in BASIC programming. Each of the matrix operations subroutines is invoked by the command MAT and is followed by various operations and functions. The command to multiply matrices B and C and place the result in matrix A is "MAT A = B*C. The technique of capturing RST 10H so that the customized command MAT can be identified is illustrated in the Program Listing.

The initialization portion of the code sets the top of memory and loads the address of the command processor (CMDPRO) into the RST 10H vector at 4004H. It also diverts the Disk BASIC error processor by placing the address of the error processor (ERRPRO) into the address 41A7H, replacing the usual vector to 57E6H. The command processor section saves the stack pointer and examines it to see if the call is from 1D5BH in the ROM and, if so, exits the user routine (via NOCMD1) without executing it, because the call is from text entry or the editor. The command processor then determines whether the MAT command is being scanned. If it is not, a continuation exit (via NOCMD) to 1D78H is used. If the MAT command is being read, the command processor relinquishes control of RST 10H and executes the user subroutine.

The user routine exits normally (via JPOUT) to the BASIC interpreter after reestablishing control of RST 10H. Abnormal exits due to errors are handled by the error processor which also recaptures RST 10H. The result is an effective method to call assembly-language routines from the direct mode or from a BASIC program. The customized commands are entered, edited, and executed in exactly the same manner as those provided by BASIC. Give it a try the next time you're combining assembly-language and BASIC programs.

Program Listing. *Example of code to use RST 10H for customizing BASIC commands*

```
                ;*********** EXAMPLE OF CODE TO USE RST 1ØH ***********
                ;*********** IN IMPLEMENTING CUSTOMIZED      ***********
                ;*********** COMMANDS ON THE TRS-8Ø MOD i    ***********
                ;
                ;
                ;*********** INITIALIZATION ***********
                ;
7ØØØ                            ORG     7ØØØH
7ØØØ 21187Ø      INIT           LD      HL,CMDPRO        ;ADDRESS OF COMMAND PROCESSOR
7ØØ3 22Ø44Ø                     LD      (4ØØ4H),HL       ;LOAD INTO RST 1ØH VECTOR
7ØØ6 21FF6F                     LD      HL,INIT-1
7ØØ9 22B14Ø                     LD      (4ØB1H),HL       ;SET TOP OF MEM
7ØØC 22494Ø                     LD      (4Ø49H),HL       ;DITTO
7ØØF 21717Ø                     LD      HL,ERRPRO        ;ADDRESS OF ERROR PROCESSOR
7Ø12 22A741                     LD      (41A7H),HL       ;DIVERT DISK BASIC ERROR ROUTINE
7Ø15 C3CCØ6                     JP      Ø6CCH            ;EXIT TO BASIC AFTER INITIALIZATION
                ;
                ;*********** COMMAND PROCESSOR ***********
                ;
7Ø18 ED737E7Ø    CMDPRO         LD      (SPSAVE),SP      ;SAVE STACK POINTER
7Ø1C DDE1                       POP     IX               ;GET CONTENTS OF STACK
7Ø1E DD22807Ø                   LD      (RETURN),IX      ;SAVE IT
```

Program continued

```
7Ø22 DDE5               PUSH    IX              ;RESTORE CONTENTS OF STACK
7Ø24 227C7Ø             LD      (BRDPTR),HL     ;SAVE BASIC INTERPRETER POINTER
7Ø27 Ø8                 EX      AF,AF'          ;SWAP REGISTER SETS
7Ø28 D9                 EXX
7Ø29 2A8Ø7Ø             LD      HL,(RETURN)     ;EXAMINE CONTENTS OF STACK
7Ø2C 115B1D             LD      DE,1D5BH        ;TO SEE IF CALL IS FROM INTERPRETER
7Ø2F DF                 RST     18H
7Ø3Ø C2567Ø             JP      NZ,NOCMD1       ;IF TEXT ENTRY OR EDIT
7Ø33 D9                 EXX                     ;SWAP BACK TO PRIMARY REGISTERS
7Ø34 Ø8                 EX      AF,AF'
7Ø35 23          SKPBLK INC     HL              ;LOOK AT NEXT CHARACTER
7Ø36 7E                 LD      A,(HL)          ;TO SEE IF IT IS A SPACE
7Ø37 FE2Ø               CP      2ØH
7Ø39 28FA               JR      Z,SKPBLK        ;SKIP ALL SPACES
7Ø3B FE4D               CP      'M'             ;IS THIS 1ST LETTER OF COMMAN?
7Ø3D C2587Ø             JP      NZ,NOCMD        ;IF NOT, RETURN TO INTERPRETER
7Ø4Ø 23                 INC     HL              ;GET NEXT CHARACTER
7Ø41 7E                 LD      A,(HL)
7Ø42 FE41               CP      'A'             ;IS THIS 2D CHAR OF COMMAND?
7Ø44 C2587Ø             JP      NZ,NOCMD        ;IF NOT RETURN TO INTERPRETER
7Ø47 23                 INC     HL              ;GET NEXT CHARACTER
7Ø48 7E                 LD      A,(HL)
7Ø49 FE54               CP      'T'             ;IS THIS THE LAST CHAR OF COMMAND?
7Ø4B C2587Ø             JP      NZ,NOCMD        ;IF NOT RETURN TO INTERPRETER
7Ø4E E5                 PUSH    HL              ;SAVE POINTER TO END OF VALID COMMAND
7Ø4F 21781D             LD      HL,1D78H        ;ADDRESS OF RST 1ØH ROUTINE
7Ø52 22Ø44Ø             LD      (4ØØ4H),HL      ;RESTORE VECTOR TO RST 1ØH ROUTINE
7Ø55 E1                 POP     HL              ;GET POINTER
                 ;
                 ;********** INSERT PROGRAM HERE.  MUST MAINTAIN CONTROL
                 ;********** OF THE POINTER SO THAT A CLEAN RETURN TO THE
                 ;********** BASIC INTERPRETER CAN BE ACCOMPLISHED AT THE
                 ;********** END OF THE USER ROUTINE.  EXIT TO JPOUT.
                 ;
                 ;********** EXIT HERE IF NO MATCH WITH COMMAND **********
                 ;
7Ø56 Ø8          NOCMD1 EX      AF,AF'          ;SWAP BACK TO PRIMARY REGISTERS
7Ø57 D9                 EXX
7Ø58 2A7C7Ø      NOCMD  LD      HL,(BRDPTR)     ;GET INTERPRETER READ POINTER
7Ø5B ED7B7E7Ø           LD      SP,(SPSAVE)     ;GET STACK POINTER
7Ø5F C3781D             JP      1D78H           ;GO PROCESS RST 1ØH
                 ;
                 ;********** EXIT HERE AT COMPLETION OF USER PGM *********
                 ;
7Ø62 21187Ø      JPOUT  LD      HL,CMDPRO       ;ADDRESS OF COMMAND PROCESSOR
7Ø65 22Ø44Ø             LD      (4ØØ4H),HL      ;RECAPTURE RST 1ØH
7Ø68 2A7C7Ø             LD      HL,(BRDPTR)     ;GET INTERPRETER READ POINTER
7Ø6B ED7B7E7Ø           LD      SP,(SPSAVE)     ;GET STACK POINTER
7Ø6F AF                 XOR     A               ;CLEAR CARRY AND ZERO 'A'
7Ø7Ø C9                 RET                     ;TO BASIC INTERPRETER
                 ;
                 ;********** ERROR PROCESSOR **********
                 ;
7Ø71 E5          ERRPRO PUSH    HL              ;SAVE READ POINTER
7Ø72 21187Ø             LD      HL,CMDPRO       ;ADDRESS OF COMMAND PROCESSOR
7Ø75 22Ø44Ø             LD      (4ØØ4H),HL      ;RECAPTURE RST 1ØH
7Ø78 E1                 POP     HL              ;GET READ POINTER
7Ø79 C3E657             JP      57E6H           ;GO PROCESS ERROR
                 ;
7Ø7C ØØØØ        BRDPTR DEFW    Ø
7Ø7E ØØØØ        SPSAVE DEFW    Ø
```

```
7080 0000       00930 RETURN  DEFW    0
                00940 ;
7000            00950         END     INIT              ;SET TRA ADDRESS TO INIT ROUTINE
00000 TOTAL ERRORS
```

9 Adding Commands to BASIC

by Alan R. Moyer

System Requirements:
Level II BASIC
16K RAM
Editor/assembler

Have you ever said to yourself, "I wish I had that command in my BASIC?" You *can* add your own commands.

BASIC is an interpretive language. It moves through your program listing, reading each command, deciding what it is to be done, and doing it, unless an error occurs (in which case it usually stops and tells you about it). Each BASIC command is stored in memory as a one-byte representation of the command. These representations are called tokens. When one is recognized, the interpreter looks it up in a table. The table tells where to go to execute the command. This process of interpreting your commands is what makes BASIC slow compared to machine-language programs, which are directly executed by the machine.

To get the BASIC interpreter to jump to a machine-language routine of your choice, put the new routine's address in the look-up table. When the appropriate command is encountered, BASIC will jump to the new routine.

BASIC keeps track of where it is in the program by reserving the Z80's HL register pair, so that it always points to the memory location of the character the interpreter is looking at. The A register contains the character at which the HL register pair is pointing. LD A,(HL) is a Z80 instruction BASIC uses to load the contents of the memory location pointed to by the HL register pair into the A register. Anytime the HL register pair is used in the execution of a BASIC command, its contents are saved and restored before returning to the BASIC interpreter. If the HL register pair is not restored, BASIC gets lost and the computer locks up.

When the interpreter recognizes a command and jumps to the appropriate routine, the HL register pair points to the next valid character after the command. You can pass information to a routine by placing the information after the BASIC command keywords.

Using NAME

For example, the NAME command is usually not used, but you can use it to pass two pieces of information to a machine-language program for processing. Put the address of the new routine into the look-up table, where the jump address for the NAME command is stored, with POKE 16783, X:POKE 16784, Y. Addresses 16783 and 16784 are the jump address for NAME. The command to be implemented is NAME A$,B$. This command exchanges the contents of A$ and B$ without using any temporary variables (for example, TEMP$ = A$:A$ = B$: B$ = TEMP$). The interpreter sees the NAME command and jumps to the proper address. The HL register pair then points to the next non-space character after NAME. In the example, HL points to the A in A$,B$.

Useful ROM Routines

Figure 1 contains addresses for useful ROM routines that can be used to get information about the arguments that are passed to the routine. To swap the two string variables in the new command, find the address value of each of the string variables (VARPTR), and switch the data for each string. (For a complete explanation of VARPTR and how the variables are stored in memory, see Radio Shack's *Level II BASIC Reference Manual.*) Once you know how to find the information for each variable, you can switch the string data and return to BASIC.

Three New Commands

The machine-language program in Program Listing 1 adds three new commands to BASIC.

NAME STR$ (SWAP) exchanges the contents of two specified string variables without using any string workspace.

NAME CVS (hexadecimal to decimal) converts any hexadecimal number in either a string literal or a string variable into its decimal equivalent in a specified integer variable.

NAME CVI (decimal to hexadecimal) converts any decimal number, either a numeric constant or integer numeric variable, into its hexadecimal equivalent in a specified string variable.

NAME STR$ (SWAP) Command

Program Listing 2 demonstrates the SWAP command. Three identical sets of string array variables are created, then swapped 100 times. Three different methods of swapping strings are timed for comparison.

Lines 10-190 initialize the string variables and ask which method of swapping is desired. String workspace cleanup, also known as garbage collection, is forced before each method. Method one uses a dummy variable to swap in line 390. This routine takes two minutes, 18 seconds. The major reason for this long time is garbage collection. By providing more initial string workspace (CLEAR 15000 bytes), garbage collection does not occur, and the swap time is only three seconds. That sounds impressive, but is unrealistic. If 15,000 bytes of string space is provided in a program, especially in a business program, most will be used and garbage collection will occur.

The method in lines 420-520 is faster and does essentially the same

BASIC's Accumulator
Useful for temporary storage of values during execution

	INT	SNG	DBL	STRING
411DH—			LSB	
411EH—			LSB	
411FH—			LSB	
4120H—			LSB	
4121H—	LSB	LSB	LSB	LSB=>$ DATA
4122H—	MSB	LSB	LSB	MSB=>$ DATA
4123H—		MSB	MSB	
4124H—		EXP	EXP	

40AFH—NTF (Number Type Flag) 2=INT, 3=STRING, 4=SNG, 8=DBL
0A7FH—Converts ACCUM to INT. Loads HL with ACCUM.
0E6CH—Converts ASCII string pointed to by HL to ACCUM and sets NTF accordingly. Zero is returned if the string is non-numeric.
0FBDH—Converts ACCUM to ASCII string for display. HL = string's address, DE = end of string plus one. ASCII string is terminated with a zero (null).
2337H—Evaluates the expression pointed to by HL. The ACCUM contains the result and NTF set accordingly. The expression is terminated with any valid delimiter. The terminator is pointed to by HL when finished. All error routines are contained in this subroutine (it flags type mismatch, for example).
2540H—Loads the ACCUM with specified variable's value whose ASCII representation is pointed to by HL. Sets NTF accordingly. ACCUM is zero if the variable is not found. HL points to the character after the ASCII of the variable.
25D9H—(RST 20H) Tests the NTF (at 40AFH). A register = NTF - 3, Z set if NTF is a string, S set if INT, S reset and C set if SNG, S and C reset if DBL.
260DH—Gets the VARPTR of the specified variable. HL points to the ASCII representation of that variable (A$). If the variable does not exist in the variable table, it is created. DE points to the variable's address and NTF is set accordingly. HL points to the character following the ASCII representation of the variable.
2857H—Checks string workspace and reserves the number of bytes contained in the A register. The address of the work space is contained in 40D4H. Garbage collection takes place if necessary. An out-of-string error occurs if there's not enough room.
4467H—Outputs a message (DOS and Disk BASIC). HL points to the beginning of the ASCII message. Message is terminated with a zero (null).

Figure 1. *Important addresses and useful ROM routines*

thing as the NAME STR$ command. This method, however, swaps from BASIC, using the VARPTR statement to point to the string variable's data. This method takes 18 seconds, a definite improvement. The major advantage is that garbage collection never occurs, since all data manipulation is done with VARPTR pointers; no actual string operations are done.

The third method, in lines 530–580, uses the new NAME STR$ routine. All swaps are done as a single BASIC line (560) taking only two seconds. This improvement is especially noticed when doing a sort in BASIC.

NAME CVS and CVI

These are two common functions. Disk BASIC only partially addresses one, with its &H statement, which allows you to use hexadecimal constants instead of decimal. This Disk BASIC command only allows you to use hexadecimal literals, not string variables containing hexadecimal values. The NAME CVS command allows string variables with hexadecimal values to be used as hex constants. This machine-language routine is quite an advantage over a comparable BASIC routine. NAME CVI does conversions in the other direction, with the same advantages. Program Listings 3 and 4 are demonstration programs for NAME CVI and NAME CVS. For those without an editor/assembler, Program Listing 5 contains the NAME STR$/CVI/CVS routine as BASIC DATA statements. This program protects memory size automatically, then installs the new commands. Program Listings 6, 7 and 8 provide each of the commands in a single format. Each program protects memory size automatically and installs the appropriate command.

Don't be frightened by the length of the NAME STR$/CVI/CVS assembler listing. Much of it consists of comments explaining the concepts. The code only requires 290 bytes of memory.

Program Listing 1

```
00100 ;***********************************************************
00110 ;**    ** NAME STR$/CVI/CVS (SWAP/DECHEX/HEXDEC) **    **
00120 ;**                  By Alan R. Moyer                  **
00130 ;**                     07/10/81                       **
00140 ;**                   Version  1.0                     **
00150 ;**                                                    **
00160 ;** This supervisory routine checks the syntax of the  **
00170 ;** data following the BASIC NAME command word.         **
00180 ;** If the data is one of the tokens that represent     **
00190 ;** either STR$ (F4); indicating a SWAP command, or     **
00200 ;** CVI (E6); indicating a Dec to Hex conversion,      **
00210 ;** or CVS (E7); indicating a Hex to Dec conversion,   **
00220 ;** that command is then executed, otherwise a syntax  **
00230 ;** error is generated. See comments under the ind-    **
```

Program continued

```
                ØØ24Ø ;** ividual commands for their explanations.                 **
                ØØ25Ø ;**   To initialize in BASIC;                                **
                ØØ26Ø ;** POKE 16783,221: POKE 16784,254  (48K)                    **
                ØØ27Ø ;** POKE 16783,221: POKE 16784,19Ø  (32K)                    **
                ØØ28Ø ;** POKE 16783,221: POKE 16784,126  (16K)                    **
                ØØ29Ø ;**************************************************************
                ØØ3ØØ ;
                ØØ31Ø ;** Define labels
                ØØ32Ø ;
26ØD            ØØ33Ø VARPTR  EQU     26ØDH    ;VARPTR subroutine ADDR
ØAF4            ØØ34Ø CHKVAR  EQU     ØØAF4H   ;Check for $ VAR
7FFF            ØØ35Ø MEM16K  EQU     Ø7FFFH   ;Top of 16K memory
BFFF            ØØ36Ø MEM32K  EQU     ØBFFFH   ;Top of 32K memory
FFFF            ØØ37Ø MEM48K  EQU     ØFFFFH   ;Top of 48K memory
4Ø49            ØØ38Ø TOPMEM  EQU     4Ø49H    ;Top of memory pointer
418F            ØØ39Ø NAME    EQU     418FH    ;NAME vector
4Ø2D            ØØ4ØØ DOS     EQU     4Ø2DH    ;DOS reentry ADDR
ØØ72            ØØ41Ø BASIC   EQU     ØØ72H    ;BASIC reentry ADDR
2857            ØØ42Ø STRRM   EQU     2857H    ;String room check ADDR
2BØ2            ØØ43Ø EVALNO  EQU     2BØ2H    ;Evaluate the numeric exp.
2337            ØØ435 EVALST  EQU     2337H    ;Evaluate the string exp.
4121            ØØ44Ø ACCUM   EQU     4121H    ;ACCUM in BASIC
19A2            ØØ45Ø ERROR   EQU     19A2H    ;Error message routine ADDR
4ØAF            ØØ46Ø NTF     EQU     4ØAFH    ;NTF (Number Type Flag)
FFFF            ØØ47Ø MEMSIZ  EQU     MEM48K   ;Put your memory size here
                ØØ48Ø ;
                ØØ49Ø ;** Housekeeping
                ØØ5ØØ ;
FECE            ØØ51Ø         ORG     MEMSIZ-131H      ;
FECE 21DDFE     ØØ52Ø START   LD      HL,NAMCMD        ;HL=>New routine ADDR
FED1 228F41     ØØ53Ø         LD      (NAME),HL        ;New ADDR (LII only)
FED4 21DCFE     ØØ54Ø         LD      HL,NAMCMD-1      ;
FED7 22494Ø     ØØ55Ø         LD      (TOPMEM),HL      ;Set new memory size
FEDA C32D4Ø     ØØ56Ø         JP      DOS              ;Change to suit
                ØØ57Ø ;
                ØØ58Ø ;** Start of command decoder
                ØØ59Ø ;
FEDD FEF4       ØØ6ØØ NAMCMD  CP      ØF4H             ;Is it STR$?
FEDF CA1DFF     ØØ61Ø         JP      Z,SWAP           ;Yes, go process
FEE2 FEE6       ØØ62Ø         CP      ØE6H             ;Is it CVI?
FEE4 CAEFFE     ØØ63Ø         JP      Z,DECHEX         ;Yes, go process
FEE7 FEE7       ØØ64Ø         CP      ØE7H             ;Is it CVS?
FEE9 CA58FF     ØØ65Ø         JP      Z,HEXDEC         ;Yes, go process
FEEC C3A8FF     ØØ66Ø         JP      SNERR            ;And print error
                ØØ67Ø ;**************************************************************
                ØØ68Ø ;**                       ** DECHEX **                       **
                ØØ69Ø ;**                      By Alan R. Moyer                    **
                ØØ7ØØ ;**                          Ø7/Ø2/81                        **
                ØØ71Ø ;**                        Version 2.1                       **
                ØØ72Ø ;**                                                          **
                ØØ73Ø ;** DECHEX is a machine language routine that will           **
                ØØ74Ø ;** place a ASCII hex representation of an integer into      **
                ØØ75Ø ;** a specified string variable. It is executed as;          **
                ØØ76Ø ;**                      NAME CVI A$,255                     **
                ØØ77Ø ;**                            -or-                          **
                ØØ78Ø ;**               NAME CVI A$(X),(A!+B#)*C%+2                **
                ØØ79Ø ;**         (The CVI stands for ConVert from Integer)        **
                ØØ8ØØ ;** The string variable will be created if it does not   **
                ØØ81Ø ;** exist. This routine checks the string work space     **
                ØØ82Ø ;** and will issue a 'garbage collection' if work        **
                ØØ83Ø ;** space is insufficient. After that if there is         **
```

```
              ;** still not enough work space, an OUT OF STRING      **
              ;** SPACE error will occur. The numeric expression can **
              ;** can consist of variables and numeric values. This  **
              ;** expression will be evaluated and turned into an    **
              ;** integer value. This integer value will be in       **
              ;** standard RS BASIC integer convention, with values  **
              ;** over 32767 to be expressed as negative numbers.    **
              ;** Use the formula on page 8/6 of the Level II BASIC  **
              ;** Reference Manual (second edition) for number       **
              ;** conversions.                                       **
              ;*******************************************************
FEEF D7       DECHEX  RST     1ØH             ;Get next non blank char
FEFØ CDØD26           CALL    VARPTR          ;Get VARPTR of VAR
FEF3 CDF4ØA           CALL    CHKVAR          ;Make sure it's a $
FEF6 ED53F6FF         LD      (PTR1),DE       ;Save the VARPTR
FEFA E5               PUSH    HL              ;Save the Char Pointer
FEFB 3EØ4             LD      A,4             ;Need 4 chars work space
FEFD CD5728           CALL    STRRM           ;Room? (4ØD4)=>work space
FFØØ E1               POP     HL              ;HL=>next char
FFØ1 23               INC     HL              ;Jump over the delimiter
FFØ2 CDØ22B           CALL    EVALNO  ;Eval the expr. DE=CINT(expr)
FFØ5 E5               PUSH    HL              ;Save the char pointer
FFØ6 2AD44Ø           LD      HL,(4ØD4H)      ;HL=>$ work space
FFØ9 CDD2FF           CALL    INTHEX          ;Convert DE to HEX$
FFØC 21D34Ø           LD      HL,4ØD3H        ;Get the $ data
FFØF ED5BF6FF         LD      DE,(PTR1)       ;Get the $ VARPTR
FF13 Ø6Ø3             LD      B,3             ;Data counter
FF15 7E       AGAIN   LD      A,(HL)          ;A=HEX$ data
FF16 12               LD      (DE),A          ;VARPTR=HEX$ data
FF17 23               INC     HL              ;HL=>next HEX$ data
FF18 13               INC     DE              ;DE>=next VARPTR address
FF19 1ØFA             DJNZ    AGAIN           ;Do again if not done
FF1B E1               POP     HL              ;Done, restore char pntr
FF1C C9               RET                     ;And return to BASIC
              ;*******************************************************
              ;**                   ** SWAP **                       **
              ;**               By Alan R. Moyer                     **
              ;**                   Ø6/2Ø/81                         **
              ;**                 Version  1.1                       **
              ;**                                                    **
              ;** SWAP is a machine language routine that will SWAP  **
              ;** the contents of two string variables as a direct   **
              ;** command from BASIC. It is executed as;             **
              ;**               NAME STR$ A$,B$                      **
              ;**                    -or-                            **
              ;**         NAME STR$ A$(Z%),B$(B#,C!+2)               **
              ;** The string variables can be simple string var-     **
              ;** iables or array string variables. Any number of    **
              ;** array dimensions can be used. For array variables, **
              ;** the subscript descriptors can be numeric variables.**
              ;** This routine will swap the string variables WITH-  **
              ;** OUT USING ANY STRING WORKSPACE, WHICH MEANS THAT   **
              ;** GARGAGE COLLECTION WILL NEVER HAPPEN WHEN USING    **
              ;** THIS COMMAND! Both string variables will be        **
              ;** created if they do not exist, and their null       **
              ;** contents will be swapped.                          **
              ;*******************************************************
FF1D D7       SWAP    RST     1ØH             ;Get next non blank char
FF1E CDØD26           CALL    VARPTR          ;Get VARPTR of 1st VAR
FF21 CDF4ØA           CALL    CHKVAR          ;Make sure it's a $
FF24 ED53F6FF         LD      (PTR1),DE       ;Save the VARPTR
```

Program continued

```
FF28 DD21F8FF          LD      IX,STRNG1      ;IX=> $ data buffer
FF2C CDBCFF            CALL    STORE          ;Store the $ data
FF2F 23                INC     HL             ;Jump over the delimiter
FF30 CD0D26            CALL    VARPTR         ;Get VARPTR of 2nd VAR
FF33 CDF40A            CALL    CHKVAR         ;Make sure it's a $
FF36 ED53FBFF          LD      (PTR2),DE      ;Save the VARPTR
FF3A DD21FDFF          LD      IX,STRNG2      ;IX=> $ data buffer
FF3E CDBCFF            CALL    STORE          ;Store the $ data
FF41 DD2AF6FF          LD      IX,(PTR1)      ;IX=> VARPTR of 1st VAR
FF45 FD21FDFF          LD      IY,STRNG2      ;IY=> 2nd VAR data
FF49 CDBFFF            CALL    SWITCH         ;Switch the $ data
FF4C DD2AFBFF          LD      IX,(PTR2)      ;IX=> VARPTR of 2nd VAR
FF50 FD21F8FF          LD      IY,STRNG1      ;IY=> 1st VAR data
FF54 CDBFFF            CALL    SWITCH         ;Switch the $ data
FF57 C9                RET                    ;And all done!
                ;***********************************************************
                ;**                   ** HEXDEC **                        **
                ;**                     07/06/81                          **
                ;**                   Version  1.0                        **
                ;**                                                        **
                ;** HEXDEC is a machine language routine that will        **
                ;** convert hexadecimal numbers contained in string       **
                ;** variables into their integer equivalents. The         **
                ;** command is executed as such;                          **
                ;**                  NAME CVS A%,A$                        **
                ;**                       -or-                             **
                ;**               NAME CVS A%,"FF"+A$                      **
                ;**      (The CVS stands for ConVert from String.)        **
                ;** The integer variable will be created if it does      **
                ;** exist. The hexadecimal range is 0000 to FFFF.         **
                ;** The integer variable will be created if it does      **
                ;** not exist. The string characters can be up to four   **
                ;** hexadecimal characters contained in either a         **
                ;** string variable or string literal.                    **
                ;** The integer equivalent will be expressed as stand-   **
                ;** ard RS BASIC convention, with values over 32767 to   **
                ;** be expressed as negative numbers.                     **
                ;***********************************************************
FF58 D7         HEXDEC RST     10H            ;Get next non blank
FF59 CD0D26            CALL    VARPTR         ;Get VARPTR of variable
FF5C ED53F6FF          LD      (PTR1),DE      ;Save the VARPTR
FF60 3AAF40            LD      A,(NTF)        ;Check the NTF
FF63 FE02              CP      2              ;Was it an integer?
FF65 2046              JR      NZ,TMERR       ;No, output an TM error
FF67 23                INC     HL             ;Skip over the delimiter
FF68 CD3723            CALL    EVALST         ;Evaluate the expression
FF6B CDF40A            CALL    CHKVAR         ;Make sure it's a $
FF6E ED5B2141          LD      DE,(ACCUM)     ;Get VARPTR data
FF72 E5                PUSH    HL             ;Save char pointer
FF73 1A                LD      A,(DE)         ;Get length of $
FF74 FE05              CP      5              ;Is $ length < 5 chars?
FF76 303A              JR      NC,OVERR       ;No, output an OV error
FF78 47                LD      B,A            ;Put char count in A
FF79 13                INC     DE             ;DE=>chars
FF7A EB                EX      DE,HL          ;HL=>chars
FF7B 5E                LD      E,(HL)         ;Put ADDR of chars
FF7C 23                INC     HL
FF7D 56                LD      D,(HL)         ;into DE
FF7E 210000            LD      HL,00H         ;Zero the binary count
FF81 29         HEXBIN ADD     HL,HL          ;Bump the
FF82 29                ADD     HL,HL          ;number left
```

```
FF83 29        Ø207Ø         ADD    HL,HL              ;four
FF84 29        Ø208Ø         ADD    HL,HL              ;places
FF85 1A        Ø209Ø         LD     A,(DE)             ;A=char
FF86 FE3A      Ø21ØØ         CP     3AH                ;Higher than a "9"?
FF88 38Ø8      Ø211Ø         JR     C,NOATOF           ;No
FF8A FE41      Ø212Ø         CP     41H                ;Lower than UC A?
FF8C 3829      Ø213Ø         JR     C,FCERR            ;Yes
FF8E E6DF      Ø214Ø         AND    ØDFH               ;Adjust if LC
FF9Ø D6Ø7      Ø215Ø         SUB    7                  ;Adjust for A-F
FF92 D63Ø      Ø216Ø NOATOF  SUB    3ØH                ;ASCII HEX to Binary
FF94 3821      Ø217Ø         JR     C,FCERR            ;Error if < Ø
FF96 FE1Ø      Ø218Ø         CP     1ØH                ;> 15?
FF98 3Ø1D      Ø219Ø         JR     NC,FCERR           ;Yes
FF9A B5        Ø22ØØ         OR     L                  ;Add with current #
FF9B 6F        Ø221Ø         LD     L,A                ;Put total into L
FF9C 13        Ø222Ø         INC    DE                 ;DE=>Next char
FF9D 1ØE2      Ø223Ø         DJNZ   HEXBIN             ;Do again if chars left
FF9F EB        Ø224Ø         EX     DE,HL              ;DE=Integer value
FFAØ 2AF6FF    Ø225Ø         LD     HL,(PTR1)          ;HL=>VARPTR of Int VAR
FFA3 73        Ø226Ø         LD     (HL),E             ;(HL)=LSB
FFA4 23        Ø227Ø         INC    HL
FFA5 72        Ø228Ø         LD     (HL),D             ;(HL)=MSB
FFA6 E1        Ø229Ø         POP    HL                 ;Restore char pointer
FFA7 C9        Ø23ØØ         RET                       ;And return
               Ø231Ø ;************************************************************
               Ø232Ø ;**                  NAME COMMAND SUBROUTINES               **
               Ø233Ø ;************************************************************
FFA8 1EØ2      Ø234Ø SNERR   LD     E,2                ;SN error code
FFAA C3A219    Ø235Ø         JP     ERROR
FFAD 1E18      Ø236Ø TMERR   LD     E,24               ;TM error code
FFAF C3A219    Ø237Ø         JP     ERROR
FFB2 1EØA      Ø238Ø OVERR   LD     E,1Ø               ;OV error code
FFB4 C3A219    Ø239Ø         JP     ERROR
FFB7 1EØ8      Ø24ØØ FCERR   LD     E,8                ;FC error code
FFB9 C3A219    Ø241Ø         JP     ERROR
               Ø242Ø ;
               Ø243Ø ;*** STORE and SWITCH subroutine
               Ø244Ø ;*** Stores data in a working buffer.
               Ø245Ø ;*** IY=>Source ADDR
               Ø246Ø ;*** IX=>Destination ADDR
               Ø247Ø ;*** A is destroyed
               Ø248Ø ;
FFBC D5        Ø249Ø STORE   PUSH   DE                 ;Put the VARPTR into
FFBD FDE1      Ø25ØØ         POP    IY                 ;the IY register
FFBF FD7EØØ    Ø251Ø SWITCH  LD     A,(IY+Ø)           ;A contains $ length
FFC2 DD77ØØ    Ø252Ø         LD     (IX+Ø),A           ;
FFC5 FD7EØ1    Ø253Ø         LD     A,(IY+1)           ;A contains $ LSB
FFC8 DD77Ø1    Ø254Ø         LD     (IX+1),A           ;
FFCB FD7EØ2    Ø255Ø         LD     A,(IY+2)           ;A contains $ MSB
FFCE DD77Ø2    Ø256Ø         LD     (IX+2),A           ;
FFD1 C9        Ø257Ø         RET                       ;And return
               Ø258Ø ;*** INTEGER TO HEX$ UTILITY
               Ø259Ø ;*** DE contains integer value to be converted
               Ø26ØØ ;*** HL=>buffer to contain the HEX$
               Ø261Ø ;*** A is destroyed
               Ø262Ø ;
FFD2 7A        Ø263Ø INTHEX  LD     A,D                ;A=MSB
FFD3 CDD7FF    Ø264Ø         CALL   CONVRT             ;Convert it
FFD6 7B        Ø265Ø         LD     A,E                ;A=LSB
FFD7 F5        Ø266Ø CONVRT  PUSH   AF                 ;Save the digits
FFD8 CB3F      Ø267Ø         SRL    A
```

Program continued

```
FFDA CB3F       Ø268Ø           SRL     A
FFDC CB3F       Ø269Ø           SRL     A
FFDE CB3F       Ø27ØØ           SRL     A
FFEØ CDEEFF     Ø271Ø           CALL    CHEKIT          ;Convert to ASCII
FFE3 77         Ø272Ø           LD      (HL),A          ;1st ASCII into buffer
FFE4 23         Ø273Ø           INC     HL              ;Bump buffer pointer
FFE5 F1         Ø274Ø           POP     AF              ;Get original digits
FFE6 E6ØF       Ø275Ø           AND     ØFH             ;Mask off high digits
FFE8 CDEEFF     Ø276Ø           CALL    CHEKIT          ;Convert to ASCII
FFEB 77         Ø277Ø           LD      (HL),A          ;2nd ASCII into buffer
FFEC 23         Ø278Ø           INC     HL              ;Bump buffer pointer
FFED C9         Ø279Ø           RET
FFEE C63Ø       Ø28ØØ CHEKIT    ADD     A,3ØH           ;Convert
FFFØ FE3A       Ø281Ø           CP      Ø3AH            ;Ø-9?
FFF2 F8         Ø282Ø           RET     M               ;Yes, return it
FFF3 C6Ø7       Ø283Ø           ADD     A,Ø7H           ;Correct for A-F
FFF5 C9         Ø284Ø           RET                     ;& return it
                Ø285Ø ;*************************************************************
                Ø286Ø ;**                       WORKSPACE                          **
                Ø287Ø ;*************************************************************
FFF6 ØØØØ       Ø288Ø PTR1      DEFW    ØØØØH           ;General storage
ØØØ3            Ø289Ø STRNG1    DEFS    3               ;1st $ VARPTR storage
FFFB ØØØØ       Ø29ØØ PTR2      DEFW    ØØØØH           ;2nd $ VARPTR data
ØØØ3            Ø291Ø STRNG2    DEFS    3               ;2nd $ VARPTR storage
                Ø292Ø ;
FECE            Ø293Ø           END     START           ;Auto start
ØØØØØ TOTAL ERRORS
```

Program Listing 2

```
1Ø '** Demonstration program for the NAME STR$/CVI/CVS command
2Ø '** This routine demonstates the STR$ (SWAP) function
3Ø DEFINT D,X,Y
4Ø CLEAR 65ØØ
5Ø POKE16783,225: POKE16784,254 '** Install new name vector
6Ø D=1ØØ
7Ø DIM A1$(D),A2$(D),A3$(D),B1$(D),B2$(D),B3$(D),C1$(D),C2$(D),C
3$(D)
8Ø CLS: PRINT"Initializing strings"
9Ø FOR X=Ø TO D
1ØØ A1$(X)=STRING$(RND(1Ø),RND(25)+65): A2$(X)=A1$(X): A3$(X)=A1
$(X)
11Ø B1$(X)=STRING$(RND(1Ø),RND(25)+65): B2$(X)=B1$(X): B3$(X)=B1
$(X)
12Ø C1$(X)=STRING$(RND(1Ø),RND(25)+65): C2$(X)=C1$(X): C3$(X)=C1
$(X)
13Ø NEXT X
14Ø A1=Ø: A2=Ø: A3=Ø: B1=Ø: B2=Ø: B3=Ø: C1=Ø: C2=Ø: C3=Ø
15Ø CLS: PRINT"Which method ?? - 1, 2, OR 3": X=Ø
16Ø A$=INKEY$: IF A$="1"ORA$="2"ORA$="3" THEN 2ØØ
17Ø X=X+1: IF X>4 THEN X=1
18Ø IF X=1 THEN PRINT@13,"  ";: PRINT@27,"3";: ELSE IF X=2 THEN
PRINT@18," ";: PRINT@13,"??";: ELSE IF X=3 THEN PRINT@21," ";: P
RINT@18,"1";: ELSE PRINT@27," ";: PRINT@21,"2";
19Ø FOR Y=Ø TO 5Ø: NEXT :GOTO 16Ø
2ØØ M=VAL(A$): CLS: PRINT@15,"EXCHANGE #";: PRINT@32,"FREE STRIN
G WORK SPACE";
21Ø GOSUB 59Ø: PRINT@128,"Starting free string work space =";
```

```
220 PRINT @192,"Please wait...and wait... (garbage collection ta
king place)";: PRINT@162,FRE(X$);
230 PRINT@192,CHR$(30)
240 GOSUB 590
250 PRINT@256,"Press *ENTER* to start": X=0
260 A$=INKEY$: IF A$=CHR$(13) THEN 300
270 X=X+1: IF X>2 THEN X=1
280 IF X=1 THEN PRINT@262,"        ";: ELSE PRINT@262,"*ENTER*";
290 FOR Y=1 TO 50: NEXT: GOTO 260
300 PRINT@256,CHR$(30): ON M GOSUB 360,420,530
310 PRINT@704,"Press *ENTER* to continue";: X=0
320 A$=INKEY$: IF A$=CHR$(13) THEN PRINT@256,CHR$(31): GOTO 150
330 X=X+1: IF X>2 THEN X=1
340 IF X=1 THEN PRINT@710,"        ";: ELSE PRINT@710,"*ENTER*";
350 FOR Y=1 TO 50: NEXT: GOTO 320
360 PRINT@256,"Swapping variables using a dummy variable"
370 PRINT RIGHT$(TIME$,8)
380 FOR X=0 TO D : PRINT@84,X; :GOSUB 590
390 D$=A1$(X):A1$(X)=B1$(X):B1$(X)=C1$(X):C1$(X)=D$
400 NEXT X
410 PRINT@384,RIGHT$(TIME$,8): PRINT"Done!": RETURN
420 PRINT@256,"Swapping the strings using the VARPTR routine"
430 PRINT RIGHT$(TIME$,8)
440 FOR X=0 TO D : PRINT@84,X; :GOSUB 590
450 A1=PEEK(VARPTR(A2$(X))):A2=PEEK(VARPTR(A2$(X))+1):A3=PEEK(VA
RPTR(A2$(X))+2)
460 B1=PEEK(VARPTR(B2$(X))):B2=PEEK(VARPTR(B2$(X))+1):B3=PEEK(VA
RPTR(B2$(X))+2)
470 C1=PEEK(VARPTR(C2$(X))):C2=PEEK(VARPTR(C2$(X))+1):C3=PEEK(VA
RPTR(C2$(X))+2)
480 POKE(VARPTR(A2$(X))),B1:POKE(VARPTR(A2$(X))+1),B2:POKE(VARPT
R(A2$(X))+2),B3
490 POKE(VARPTR(B2$(X))),C1:POKE(VARPTR(B2$(X))+1),C2:POKE(VARPT
R(B2$(X))+2),C3
500 POKE(VARPTR(C2$(X))),A1:POKE(VARPTR(C2$(X))+1),A2:POKE(VARPT
R(C2$(X
510 NEXTX
520 PRINT@384,RIGHT$(TIME$,8): PRINT"Done!": RETURN
530 PRINT@256,"Swapping variables using the NAME command"
540 PRINTRIGHT$(TIME$,8)
550 FORX=0TOD:PRINT@84,X; :GOSUB 590
560 NAME STR$ A3$(X),B3$(X): NAME STR$ B3$(X),C3$(X)
570 NEXT X
580 PRINT@384,RIGHT$(TIME$,8):PRINT"Done!":RETURN
590 PRINT@100,(PEEK(&H40D6)+PEEK(&H40D7)*256)-(PEEK(&H40A0)+PEEK
(&H40A1)*256); :RETURN
```

Program Listing 3

```
10 '** Demonstration program for the NAME STR$/CVI/CVS command
20 '** This routine demonstrates the CVI (DECHEX) function
30 CLS: PRINT@192,"Initializing"
40 POKE 16783,221: POKE 16784,254 '** Install new name vector
50 DEF FN AD(A)=A+(A>32767)*65536 '** Convert to integer
60 DEFINT X,C,Y
70 DIM HX$(15)
80 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
90 FOR X=0 TO 15: READ HX$(X): NEXT X
```

Program continued

```
100 C=100
110 GOSUB 390
120 CLS: GOSUB 440: X=FRE(Z$)
130 PRINT@256,"Decimal to Hexadecimal conversion using BASIC"
140 PRINTRIGHT$(TIME$,8)
150 FOR X=1 TO C
160 D=PEEK(X)+PEEK(X+1)*256
170 B0=INT(D/4096)
180 B1=INT((D-B0*4096)/256)
190 B2=INT((D-((B0*4096)+(B1*256)))/16)
200 B3=D-((B0*4096)+(B1*256)+(B2*16))
210 PRINT@78,HX$(B0);HX$(B1);HX$(B2);HX$(B3);
220 PRINT@95,X;
230 NEXT X
240 PRINT@384,RIGHT$(TIME$,8)
250 PRINT"Done!"
260 PRINT: GOSUB 390
270 CLS: GOSUB 440: X=FRE(Z$)
280 PRINT@256,"Decimal to Hexadecimal conversion using NAME CVI"

290 PRINTRIGHT$(TIME$,8)
300 FOR X=1 TO C
310 NAME CVI A$,FN AD(PEEK(X)+PEEK(X+1)*256)
320 PRINT@78,A$;
330 PRINT@95,X;
340 NEXT X
350 PRINT@384,RIGHT$(TIME$,8)
360 PRINT"Done!"
370 PRINT: GOSUB 390
380 GOTO 120
390 X0=(PEEK(16416)+PEEK(16417)*256)-15360: PRINT@X0,"Press *ENT
ER* to continue";: X=0
400 A$=INKEY$: IF A$=CHR$(13) THEN RETURN
410 X=X+1: IF X>2 THEN X=1
420 IF X=1 THEN A$="       ": ELSE A$="*ENTER*"
430 PRINT@X0+6,A$;: FOR Y=1 TO 50: NEXTY: GOTO 400
440 PRINT@10,"HEX VALUE       STEP NUMBER";
450 RETURN
```

Program Listing 4

```
10 '** Demonstration program for the NAME STR$/CVI/CVS command
20 '** This routine demonstrates the CVS (HEXDEC) function
30 CLS: PRINT@192,"Initializing": CLEAR 500
40 POKE 16783,221: POKE 16784,254 '** Install new name vector
50 DEF FN AD(A)=A+(A>32767)*65536 '** Convert to integer
60 DEF FN HX(A$)=ASC(A$)+-48-7*(A$>"9")
70 DEFINT X,C,Y: DIM A$(100)
80 DIM HX$(15)
90 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
100 FOR X=0 TO 15: READ HX$(X): NEXT X
110 C=100
120 GOSUB 530: X=FRE(Z$)
130 FOR X=1 TO C
140 D=PEEK(X)+PEEK(X+1)*256
150 B0=INT(D/4096)
160 B1=INT((D-B0*4096)/256)
170 B2=INT((D-((B0*4096)+(B1*256)))/16)
```

```
180 B3=D-((B0*4096)+(B1*256)+(B2*16))
190 A$(X)=HX$(B0)+HX$(B1)+HX$(B2)+HX$(B3)
200 PRINT@78,A$(X);
210 PRINT@95,X;
220 NEXT X
230 PRINT"Done!"
240 PRINT: GOSUB 480
250 CLS: GOSUB 530: X=FRE(A$)
260 PRINT@256,"Hexadecimal to Decimal conversion using BASIC"
270 PRINTRIGHT$(TIME$,8)
280 FOR X=1 TO C
290 N=0: FOR Y=1 TO LEN(A$(X)): N=N*16+FN HX(MID$(A$(X),Y,1)): N
EXT Y: N=FN AD(N)
300 PRINT@78,N;
310 PRINT@95,X;
320 NEXT X
330 PRINT@384,RIGHT$(TIME$,8)
340 PRINT"Done!"
350 PRINT: GOSUB 480
360 CLS: GOSUB 530: X=FRE(A$)
370 PRINT@256,"Hexadecimal to Decimal conversion using NAME CVS"

380 PRINTRIGHT$(TIME$,8)
390 FOR X=1 TO C
400 NAME CVS A%,A$(X)
410 PRINT@78,A$(X);
420 PRINT@95,X;
430 NEXT X
440 PRINT@384,RIGHT$(TIME$,8)
450 PRINT"Done!"
460 PRINT: GOSUB 480
470 GOTO 250
480 X0=(PEEK(16416)+PEEK(16417)*256)-15360: PRINT@X0,"Press *ENT
ER* to continue";: X=0
490 A$=INKEY$: IF A$=CHR$(13) THEN RETURN
500 X=X+1: IF X>2 THEN X=1
510 IF X=1 THEN A$="       ": ELSE A$="*ENTER*"
520 PRINT@X0+6,A$;: FOR Y=1 TO 50: NEXTY: GOTO 490
530 PRINT@10,"HEX VALUE        STEP NUMBER";
540 RETURN
```

Program Listing 5

```
50000 '** NAME STR$/CVI/CVS (SWAP/DECHEX/HEXDEC) command using B
ASIC's NAME command
50010 M=65244 'for 48K, 48860 For 32K, 32476 for 16K
50020 X2=INT(M/256): X1=M-X2*256: POKE 16561,X1: POKE 16562,X2:
CLEAR 50 'This sets memory size from within BASIC
50030 SA=-291 'for 48K, -16675 for 32K, 32477 for 16K
50040 FOR X= SA TO SA+290: READY
50050 IF Y=888 THEN Y=254 'for 48K, 190 for 32K, 126 for 16K
50060 IF Y=999 THEN Y=255 'for 48K, 191 for 32K, 127 for 16K
50070 POKE X,Y: NEXT X
50080 POKE 16783,221: POKE 16784,254 'for 48K, 190 for 32K, 126
for 16K ** Install new NAME vector
50090 DATA 254,244,202,29,999,254,230,202,239,888,254,231,202,88
,999
50100 DATA 195,168,999,215,205,13,38,205,244,10,237,83,246,999,2
29
```

Program continued

```
50110 DATA 62,4,205,87,40,225,35,205,2,43,229,42,212,64,205
50120 DATA 210,999,33,211,64,237,91,246,999,6,3,126,18,35,19
50130 DATA 16,250,225,201,215,205,13,38,205,244,10,237,83,246,99
9
50140 DATA 221,33,248,999,205,188,999,35,205,13,38,205,244,10,23
7
50150 DATA 83,251,999,221,33,253,999,205,188,999,221,42,246,999,
253
50160 DATA 33,253,999,205,191,999,221,42,251,999,253,33,248,999,
205
50170 DATA 191,999,201,215,205,13,38,237,83,246,999,58,175,64,25
4
50180 DATA 2,32,70,35,205,55,35,205,244,10,237,91,33,65,229
50190 DATA 26,254,5,48,58,71,19,235,94,35,86,33,0,0,41
50200 DATA 41,41,41,26,254,58,56,8,254,65,56,41,230,223,214
50210 DATA 7,214,48,56,33,254,16,48,29,181,111,19,16,226,235
50220 DATA 42,246,999,115,35,114,225,201,30,2,195,162,25,30,24
50230 DATA 195,162,25,30,10,195,162,25,30,8,195,162,25,213,253
50240 DATA 225,253,126,0,221,119,0,253,126,1,221,119,1,253,126
50250 DATA 2,221,119,2,201,122,205,215,999,123,245,203,63,203,63

50260 DATA 203,63,203,63,205,238,999,119,35,241,230,15,205,238,9
99
50270 DATA 119,35,201,198,48,254,58,248,198,7,201,0,0,254,58
50280 DATA 248,0,0,201,28,108
```

Program Listing 6

```
50000 '** NAME CVS (HEXDEC) command using BASIC's NAME command
50010 M=65429 'for 48K, 49045 for 32K, 32661 for 16K
50020 X2=INT(M/256): X1=M-X2*256 :POKE 16561,X1: POKE 16562,X2:
CLEAR 50 'This sets memory size automatically from within BASIC
50030 SA=-106 'for 48K, -16490 for 32K, 32662 for 16K
50040 FOR X= SA TO SA+105: READY
50050 IF Y=999 THEN Y=255 'for 48K, 191 for 32K, 127 for 16K
50060 POKE X,Y: NEXT X
50070 POKE 16783,150: POKE 16784,255 'for 48K, 191 for 32K, 127
for 16K ** Install new NAME vector
50080 DATA 254,231,32,80,215,205,13,38,237,83,254,999,58,175,64
50090 DATA 254,2,32,70,35,205,55,35,205,244,10,237,91,33,65
50100 DATA 229,26,254,5,48,58,71,19,235,94,35,86,33,0,0
50110 DATA 41,41,41,41,26,254,58,56,8,254,65,56,41,230,223
50120 DATA 214,7,214,48,56,33,254,16,48,29,181,111,19,16,226
50130 DATA 235,42,254,999,115,35,114,225,201,30,2,195,162,25,30
50140 DATA 24,195,162,25,30,10,195,162,25,30,8,195,162,25,0
50150 DATA 0
```

Program Listing 7

```
50000 '** NAME CVI (DECHEX) command using BASIC's NAME command
50010 M=65442 'for 48K, 49058 for 32K, 32674 for 16K
50020 X2=INT(M/256): X1=M-X2*256: POKE 16561,X1: POKE 16562,X2:
CLEAR 50 'This sets memory size automatically from within BASIC
50030 SA=-93 'for 48K, -16477 for 32K, 32675 for 16K
50040 FOR X= SA TO SA+92: READY
```

```
50050 IF Y=999 THEN Y=255 'for 48K, 191 for 32K, 127 for 16K
50060 POKE X,Y: NEXT X
50070 POKE 16783,163: POKE 16784,255 'for 48K, 191 for 32K, 127
for 16K ** Install new NAME vector
50080 DATA 254,230,40,5,30,2,195,162,25,215,205,13,38,205,244
50090 DATA 10,237,83,254,999,229,62,4,205,87,40,225,35,205,2
50100 DATA 43,229,42,212,64,205,218,999,33,211;64,237,91,254,999

50110 DATA 6,3,126,18,35,19,16,250,225,201,122,205,223,255,123
50120 DATA 245,203,63,203,63,203,63,203,63,205,246,999,119,35,24
1
50130 DATA 230,15,205,246,999,119,35,201,198,48,254,58,248,198,7

50140 DATA 201,0,0
```

Program Listing 8

```
50000 '** NAME STR$ (SWAP) command using BASIC's NAME command
50010 M=65435 'for 48K, 49051 for 32K, 32667 for 16K
50020 X2=INT(M/256): X1=M-X2*256: POKE 16561,X1: POKE 16562,X2:
CLEAR 50 'This sets memory size automatically from within BASIC
50030 SA=-100 'for 48K, -16484 for 32K, 32668 for 16K
50040 FOR X=SA TO SA+99: READY
50050 IF Y=999 THEN Y=255 'for 48K, 191 for 32K, 127 for 16K
50060 POKE X,Y: NEXT X
50070 POKE 16783,156: POKE 16784,255 'for 48K, 191 for 32K, 127
for 16K ** Install new NAME vector
50080 DATA 254,244,40,5,30,2,195,162,25,215,205,13,38,205,244
50090 DATA 10,237,83,246,999,221,33,248,999,205,224,999,35,205,1
3
50100 DATA 38,205,244,10,237,83,251,999,221,33,253,999,205,224,9
99
50110 DATA 221,42,246,999,253,33,253,999,205,227,999,221,42,251,
999
50120 DATA 253,33,248,999,205,227,999,201,213,253,225,253,126,0,
221
50130 DATA 119,0,253,126,1,221,119,1,253,126,2,221,119,2,201
50140 DATA 191,104,0,0,0,0,0,0,0,0
```

10

Programming in Radio Shack's Tiny Pascal

by John Blommers

System Requirements:
Model I
Level II BASIC
16K RAM

Tiny Pascal can give you programs which execute quickly, up to ten times faster than interpreted BASIC programs. Despite the limited integer and integer array data types, Tiny Pascal provides all of the structured statements of standard Pascal. With a little ingenuity, you can write many useful programs. Each program in this chapter contains comment lines and is accompanied by an explanation in the text.

Celsius to Fahrenheit Conversion

The integer arithmetic limitation requires that the formula F = C*9/5 + 32 be rewritten as F = (C*18 + 5)/10 + 32. The 1.8 has been translated to 18/10, and adding 5 inside the parentheses before dividing by 10 results in rounding instead of truncation. This procedure gives the most accurate integer answer possible.

```
(* CELSIUS TO FAHRENHEIT CONVERSION *)
(* THIS PROGRAM IS WRITTEN IN RADIO SHACK'S TINY PASCAL.  SINCE ONLY
INTEGERS ARE SUPPORTED, MULTIPLICATION BY 1.8 IS DONE BY MULTIPLYING BY
18, ADDING 5 FOR ROUNDING, AND DIVIDING BY 10. *)
CONST LOW   = 0 ;        (* FIRST TEMPERATURE TO CONVERT *)
      HIGH  = 41 ;       (* LAST  TEMPERATURE TO CONVERT *)
      F32   = 32 ;
      ROUND = 5  ;       (* ROUNDING CONSTANT *)
VAR DEGREE, FLAG : INTEGER ;    (* GLOBAL VARIABLES *)
BEGIN                           (* OF MAIN PROGRAM *)
 WRITE(13) ; WRITE('CENTIGRADE TO FAHRENHEIT CONVERSION',13,13);
 WRITE('---------',13);
   FOR DEGREE := LOW TO HIGH DO
      BEGIN
```

```
           WRITE(DEGREE#,'C     ',((DEGREE*18+ROUND) DIV 10)+F32#,'F      ');
           IF DEGREE THEN WRITE(13)   (* CRLF IF DEGREE IS ODD *)
        END;
 WRITE('---------',13);
 WRITE(13,'*** PROGRAM STOP ***',13,13);
END.       (* OF CELSIUS TO FAHRENHEIT PROGRAM *)
```

Equation Solving Program

This program finds all the integer solutions to the equation 6*X+4*Y – 14*Z = 10 for values of X, Y, and Z from 0 to 100. The procedure is very elementary; it does an exhaustive search of all possible combinations of X, Y, and Z. The program uses INKEY to sense if the S key has been pressed during execution and stops if it has. Pressing the BREAK key twice is an alternative.

```
(* THIS PROGRAM PRINTS ALL 1454 OR SO OF THE SOLUTIONS TO THE FORMULA
6X+4Y-14Z=10. *)
VAR X,Y,Z : INTEGER ;
BEGIN
   WRITE(13,13,'SOLUTIONS (X,Y,Z) TO 6X+4Y-14Z=10',13,13) ;
   FOR X := 0 TO 100 DO          (* 3 EXHAUSTIVE LOOPS *)
   FOR Y := 0 TO 100 DO
   FOR Z := 0 TO 100 DO
      BEGIN   (* STRIKE THE 'S' KEY TO HALT THE PROGRAM *)
      IF INKEY = 'S' THEN BEGIN X:=100;Y:=100;Z:=100 END ;
      IF (6*X + 4*Y - 14*Z = 10) THEN
           WRITE('(',X#,' ',Y#,' ',Z#,')')
      END ;
   WRITE(13,13,13,13)   (* SEND A FEW LINE FEEDS *)
END.     (* OF EQUATION SOLVER PROG *)
```

Square and Cube Roots

This program lets you calculate square and cube roots for integers less than or equal to 32767. It does so by guessing initially and halving the error of its guess (by taking the middle between a low and a high limit) until the uncertainty is not greater than the constant tolerance.

```
(* SQUARE AND CUBE ROOT PROGRAM *)
(* PUT GLOBAL VARIABLES ETC UP HERE *)
VAR NUMBER : INTEGER ;
FUNC SQUAREROOT(NUMBER,POWER) ; (* POWER=1 NO ROOT, 2=SQUARE ROOT,
                                                    3=CUBE ROOT
 CONST TOLERANCE=1 ;
 VAR UPPER,LOWER,MIDDLE,PRODUCT : INTEGER ;
 BEGIN
   LOWER :=1 ; CASE POWER OF 1: UPPER:=1 ; (* FIX LOWER, UPPER LIMITS *)
                             2: IF NUMBER)182 THEN UPPER:=182
                                                   ELSE UPPER:=NUMBER;
                             3: IF NUMBER)032 THEN UPPER:=032
                                                   ELSE UPPER:=NUMBER
              END (* OF CASE *) ;
   WHILE (UPPER-LOWER) ) TOLERANCE DO (* ITERATE WHILE UNCERTAINTY *)
      BEGIN                           (* EXCEEDS TOLERANCE *)
         MIDDLE := (UPPER+LOWER) DIV 2 ;
         CASE POWER OF 1: PRODUCT := MIDDLE ;          (* NO ROOT *)
```

Program continued

```
                      2: PRODUCT := MIDDLE*MIDDLE ;    (* SQUARE ROOT *)
                      3: PRODUCT := MIDDLE*MIDDLE*MIDDLE (*CUBE ROOT *)
           END (* OF CASE *) ;
           IF PRODUCT <= NUMBER THEN LOWER := MIDDLE (* RECALCULATE THE *)
                                      ELSE UPPER := MIDDLE ; (* NEW *)
        END ;                                             (* BOUNDARIES *)
     SQUAREROOT := (UPPER+LOWER) DIV 2 (* CALCULATE FINAL ANSWER *)
  END (* OF SQUAREROOT *) ;

BEGIN (* OF MAIN PROGRAM *)
 WRITE('ENTER A NUMBER ') ; READ(NUMBER#) ;
 WRITE('THREE ROOTS ARE ',SQUAREROOT(NUMBER,1)#,'    ',
                          SQUAREROOT(NUMBER,2)#,'    ',
                          SQUAREROOT(NUMBER,3)#,13)
END.   (* OF ROOT PROGRAM *)
```

Another Square Root Program

This program demonstrates a unique approach to determining square roots. It counts the number of binary digits in the number and shifts half of them to the right as a first estimate of the square root. Three Newton iterations are done to produce a second estimate. This estimate is in error by, at most, plus or minus one, due to the round-off errors in the integer arithmetic. The program checks the second estimate, along with values of one greater and one less than the estimate. The one with the least error is returned as the square root of the argument passed. The following example calculates the square root of Y = 20:

```
20 is binary 10100 and has 5 binary digits
first estimate is 10 binary = 2
second estimate (after 3 Newtons) is 4
4-1=3 error = 20- 9=11
4    =4 error = 20-16= 4 (best of the three)
4+1=5 error = 20-25= -5
final answer = 4
```

```
VAR Z,I : INTEGER ;   (* SQUARE ROOT PROGRAM *)
FUNC SQRT(Y);
VAR X,E1,E2,E3,I,YY  : INTEGER ;
BEGIN
 IF Y < 0 THEN Y := 0 ; (* TRAP NEGATIVE ARGUMENTS *)
   CASE Y OF
 0,1 : SQRT := Y         (* TRIVIAL CASES *)
   ELSE  BEGIN
   I := 1 ;  YY := Y  ;
   WHILE YY > 0 DO BEGIN      (* COUNT # BINARY DIGITS IN ARGUMENT *)
                     I := I+1 ;
                     YY := YY SHR 1
                   END ;
   X := Y SHR ( I SHR 1 ) ;     (* FIRST ESTIMATE OF ROOT *)
   FOR I := 1 TO 3 DO X := Y DIV (X+X) + X SHR 1 ; (* 3 NEWTONS *)
   E1 := ABS(Y-SQR(X-1)) ; (* THREE ERRORS CALCULATED *)
   E2 := ABS(Y-SQR(X  )) ;
   E3 := ABS(Y-SQR(X+1)) ;
   SQRT := X ;
   IF E1 < E2 THEN SQRT := X-1 ; (* RETURN SQRT WITH LEAST ERROR *)
   IF E3 < E2 THEN SQRT := X+1
         END (* ELSE *)
   END (* CASE *)
END; (* OF SQRT *)
```

```
BEGIN          (* MAIN USER PROGRAM BEGINS HERE *)
    FOR I := 0 TO 127 DO BEGIN   (* PLOT THE SQRT FUNCTION *)
              PLOT(I,47-4*SQRT(I),1)
                        END ;
   Z := 1 ;
   WHILE Z > -1 DO BEGIN        (* LET USER TYPE IN VALUES TO TEST *)
                    WRITE('ENTER Z ') ;
                    READ(Z#) ;
                    WRITE('SQRT(',Z#,')= ',SQRT(Z)#,13,13) ;
                 END
END. (* OF SQUARE ROOT PROGRAM *)
```

Random Number Generator with Plotting

This program uses a function that generates a pseudo-random number, RND. The function depends on the overflow of integer multiplication. Note that when the result of such a multiplication is negative, the sign bit must be cleared with the MOD function. The number is not made positive with the ABS function. Run the program with a few different seeds to see just how pseudo-random the sequence is.

```
(* RANDOM NUMBER GENERATION FOR TINY PASCAL
   REF: PASCAL, BY DAVID L. HEISERMAN P 148, TAB BOOK 1205 *)
VAR I,RLOW,RHIGH,NUM,N : INTEGER;
FUNC RND(RLOW,RHIGH); (* RETURNS RANDOM # BETWEEN RLOW & RHIGH *)
   VAR M,P:INTEGER;    (* LOCAL VARIABLES *)
   BEGIN
   REPEAT
      M:=N*3125;        (* 3125 IS THE MULTIPLIER *)
      IF M<0 THEN M:=(M AND 32767)  (* MASK SIGN BIT IF REQUIRED *)
      N:=M ; P:=M;      (* N IS EXTERNAL TO THE RND FUNCTION *)
      P:= P MOD RHIGH
   UNTIL (P>=RLOW) AND (P<= RHIGH);  (* QUIT IF P IS IN RANGE *)
   RND:=P                         (* ASSIGN RND A VALUE AND RETURN *)
   END; (* OF RND *)

BEGIN                             (* MAIN PROGRAM EXERCISES RND *)
   WRITE('ENTER THE NONZERO RANDOM NUMBER SEED ');
   READ(N#);
   WRITE('HOW MANY RANDOM NUMBERS DO YOU WANT ');
   READ (NUM#);
   WRITE('STATE THE RANGE OF THESE POSITIVE NUMBERS ');
   READ(RLOW#,RHIGH#);
   FOR I:=1 TO NUM DO WRITE(RND(RLOW,RHIGH)#,'    ');
   WRITE(13,'HIT A KEY TO BEGIN PLOTTING 10000 RANDOM NUMBERS');
   READ(I);
   WRITE(28,31); (* CLEAR SCREEN *)
   FOR I:=1 TO 10000 DO PLOT( RND(0,127), RND(0,47) ,1)
END.                              (* OF RANDOM NUMBER PROGRAM *)
```

Another Random Number Generator

This program uses a different multiplier, MULT, and introduces an increment, INCR. The MOD function keeps the number in range, and the AND function keeps the numbers positive. The main program produces a histogram plot of the random numbers as they are generated.

```
VAR I,J,K,SEED : INTEGER ;   (* ANOTHER RANDOM # GENNER/PLOTTER *)
        XARRAY : ARRAY(127) OF INTEGER ;
FUNC RAND ;
CONST          (* DEFINE THE MULTIPLIER, INCREMENT AND MODULUS *)
   MULT = 25173 ;
   INCR = 13849 ;
   MODU = 16384 ;
BEGIN
   RAND := SEED ;    (* APPLY THE EXTERNAL SEED *)
   SEED := (MULT*SEED+INCR) MOD MODU ; (* CONTROL MAGNITUDE *)
   SEED := SEED AND 32767 (* AND ENSURE NUMBER IS POSITIVE *)
END ; (* OF RAND *)

BEGIN (* MAIN PROGRAM BEGINS HERE *)
SEED := 1 ;   (* USER MUST PREDEFINE A SEED *)
   FOR I := 0 TO 127 DO XARRAY(I) := 0 ; (* CLEAR COUNTERS *)
   FOR I := 1 TO 128*48 DO  (* PLOTS A NICE HISTOGRAM *)
      BEGIN
         J := RAND MOD 128 ; (* CREATE A RANDOM NUMBER *)
         XARRAY(J) := XARRAY(J)+1 ; (* INCR THE COUNTER *)
         PLOT(J,XARRAY(J),1)  (* PLOT THE POINT *)
      END
END.
```

Prime Number Generator Program

This program takes an array, FLAGS, initializes each element to 1, and marks all elements whose subscripts are multiples of 1, 2, 3, and so on. The remaining array elements are unmarked by default, and correspond to prime numbers. This algorithm is known as the Sieve of Eratosthenes. It runs slowly at first and picks up tremendous speed near the end. Prime numbers less than 8191 are printed.

```
CONST SIZE=8190 ; NUMITS=1 ; (* PRIME NUMBER PROGRAM *)
VAR FLAGS : ARRAY(SIZE) OF INTEGER ;
   I,PRIME,K,COUNT,ITER : INTEGER ;
BEGIN
   WRITE(NUMITS#,' ITERATIONS OF THE PROGRAM FOLLOW :',13) ;
   FOR ITER := 1 TO NUMITS DO
      BEGIN COUNT := 0 ; (* COUNTER OF # PRIMES CLEARED *)
            FOR I := 0 TO SIZE DO FLAGS(I) := 1 ; (* ALL FLAGS TRUE *)
            FOR I := 0 TO SIZE DO
               IF FLAGS(I) THEN
               BEGIN PRIME := I+I+3 ;
                     K := I+PRIME ;
                     WHILE K <= SIZE DO
                       BEGIN FLAGS(K) := 0 ;  (* FLAG A NON-PRIME *)
                             K := K+PRIME
                       END ;
               COUNT := COUNT +1 ; (* TALLY HOW MANY PRIMES GENERATED *)
               WRITE(PRIME#,' ')
               END ;
            WRITE(13,COUNT#,' PRIMES FOUND',13)
      END
END.  (* OF ERATOSTHENES' SIEVE *)
```

Trigonometry Functions Program

This program implements the SIN function scaled in amplitude by 32767. It is practical, because the SIN function normally returns a floating-point number between −1 and +1. The Tiny Pascal version returns integers between −32767 and +32767.

The argument of the SIN function is given in degrees. The function begins by reducing the argument to between 0 and 90 degrees and keeping track of a possible minus sign in the answer. Samples of the SIN function are stored in the procedure for every 10-degree step. The argument is further resolved to lie within one of these 10-degree steps. Then, a straight-line approximation is made to compute the returned value.

A separate general-purpose interpolation function, INTER, has been included in the program for those who want to experiment. The left-hand point (X1,Y1) and the right-hand point (X2,Y2) are joined with a line segment. For the x-value given, INTER returns the corresponding y-value lying on the line segment.

```
VAR Z,I : INTEGER ; (* SINE & COSINE FUNCTION PROGRAM *)
FUNC INTER(X,X1,X2,Y1,Y2) ;    (* INTERPOLATION FUNCTION *)
BEGIN
   INTER := Y1 + (Y2-Y1) DIV (X2-X1) * (X-X1) ;
END ;  (* OF INTER *)

(* SIN(X) FUNCTION FOR ANY X. -32767 <= SIN <= 32767 *)
FUNC SIN(X) ;
CONST N=9    ;  (*  N+1= NUMBER OF FIXED INTER'N POINTS *)
VAR I,X1,X2,Y1,Y2,SIGN : INTEGER ;
    XARRAY             : ARRAY(N) OF INTEGER ;
BEGIN
   FOR I := 0 TO N DO XARRAY(I) := 10*I   ; (* SETUP XARRAY*)
   X := X MOD 360 ; (* REDUCE ARGUMENT TO UNIT CIRCLE *)
   SIGN := +1 ;     (* ASSUME 0TH QUADRANT FIRST OFF *)
   IF X < 0 THEN SIGN := -1 ; (* FOR -VE ARG'S *)
   X := ABS(X)    ; (* YES, -1 MOD 360 GIVES -1 !!*)
   I := X DIV  90 ; (* DETERMINE WHICH QUADRANT *)
   CASE I OF         (* ADJUST TO FIRST QUADRANT *)
0:   BEGIN   X := X       ;   SIGN := SIGN   END ;
1:   BEGIN   X := 180-X   ;   SIGN := SIGN   END ;
2:   BEGIN   X := X-180   ;   SIGN := -SIGN  END ;
3:   BEGIN   X := 360-X   ;   SIGN := -SIGN  END
   END ; (* OF CASE *)
   I := 1 ;   (* DETERMINE X'S PLACE IN XARRAY *)
   WHILE XARRAY(I) < X DO I := I+1 ;
   CASE I OF           (* DETERMINE INTERPOL'N POINTS *)
1:   BEGIN   Y1 := 00000   ;   Y2 := 05690   END ;
2:   BEGIN   Y1 := 05690   ;   Y2 := 11207   END ;
3:   BEGIN   Y1 := 11207   ;   Y2 := 16384   END ;
4:   BEGIN   Y1 := 16384   ;   Y2 := 21062   END ;
5:   BEGIN   Y1 := 21062   ;   Y2 := 25101   END ;
6:   BEGIN   Y1 := 25101   ;   Y2 := 28377   END ;
7:   BEGIN   Y1 := 28377   ;   Y2 := 30791   END ;
8:   BEGIN   Y1 := 30791   ;   Y2 := 32269   END ;
9:   BEGIN   Y1 := 32269   ;   Y2 := 32767   END
   END; (* OF CASE *)
    X1 := XARRAY(I-1) ;  (* LEFT SIDE OF INTERVAL *)
    X2 := XARRAY(I  ) ;  (* RIGHT SIDE OF INTERVAL *)
    SIN := SIGN * INTER(X,X1,X2,Y1,Y2) (* LINEAR INTERPOLATE *)
END; (* OF SIN *)

FUNC COS(X) ;  (* COSINE FUNCTION, REQUIRES SIN *)
BEGIN
```

Program continued

```
   COS := SIN(X+90)
END ; (* OF COS *)

BEGIN           (* MAIN USER PROGRAM BEGINS HERE *)
   Z := 1 ;
   WHILE Z <> 999 DO BEGIN
                     WRITE('ENTER Z ') ;
                     READ(Z#) ; (* ENTERING 999 STOPS PROGRAM *)
                     WRITE('SIN (',Z#,')=', SIN(Z)#,13,13) ;
                     END (* OF DO *)
END.   (* OF SIN FUNCTION DEMONSTATOR *)
```

The Exponential Function

In integer-based Tiny Pascal, the EXP function is limited to arguments from 0 to 10. Its value exceeds 32767 for arguments over 10 and is set to 0 for arguments less than 0. Consequently, a simple set of CASE statements is used to implement the EXP function.

```
(* EXPONENTIAL FUNCTION FOR RADIO SHACK TINY PASCAL *)
VAR X : INTEGER ; (* GLOBAL VARIABLE *)
FUNC EXP(T) ;
BEGIN
CASE T OF 00 : EXP := 00001 ; (* HANDLE THE GOOD ARGUMENTS *)
          01 : EXP := 00003 ;
          02 : EXP := 00007 ;
          03 : EXP := 00020 ;
          04 : EXP := 00055 ;
          05 : EXP := 00148 ;
          06 : EXP := 00403 ;
          07 : EXP := 01097 ;
          08 : EXP := 02981 ;
          09 : EXP := 08103 ;
          10 : EXP := 22027
          ELSE EXP := 32767      (* THIS IS THE OVERFLOW CASE *)
END (* OF CASE *);
IF  T<0   THEN EXP := 00000      (* RETURN 0 FOR MINUS INPUTS *)
END (* OF FUNCTION *) ;

BEGIN (* MAIN PROGRAM BEGINS HERE *)
X := 0 ;     (* DEFINE X SO THE DO-WHILE WILL BEHAVE PREDICTABLY *)
WHILE X <> 9999 DO     (* QUIT IF USER ENTER 9999 *)
  BEGIN  WRITE(' ENTER AN ARGUMENT ') ;
         READ(X#) ;
         WRITE('EXP(',X#,')=',EXP(X)#,13) ;
  END
END. (* OF EXPONENTIAL PROGRAM *)
```

Morse Code Speed Timing Program

Connect your Morse code keyer's tone output (or your shortwave radio receive output) to the cassette earphone input of the TRS-80, and run this program. It measures the code speed in words per minute (WPM) based on both the dit and the dah durations. You tell the program how many dits and dahs to count, up to 100. The program stores the duration of each dit and dah in an array. The array is then sorted (simple bubble sort) by the SORT procedure, and the procedure AVERAGE computes the average duration of dits and dahs combined. This results in a first, crude estimate of the code speed. Next, the sorted array is searched from the beginning until the spot is found where the dahs begin. Two more code speed estimates are made—one based on just dits and one based on just dahs. The procedures DELAY, MARK, and SPACE are required to perform timing and detect the presence of dits and dahs (MARKS) and silence (SPACES). The program was tested on a 1.7 MHz machine.

This program was tested on a 32K machine and may not work on a 16K machine. Removing comments will help; use the TABs to align statements or eliminate them altogether. Note that the comments apply to the DELAY, SPACE, and MARK routines of the Morse code program.

```
VAR CODE,NUM,L,TEMP,AVG,J,TOTAL  : INTEGER; (* MORSE CODE WPM TESTER *)
    DITAVG,DAHAVG,THRESHOLD,S    : INTEGER; (* SUPPORT FUNCTIONS/PROCS *)
    X                            : ARRAY(100) OF INTEGER;
PROC DELAY(N);                  (* ADDS A LITTLE DELAY FOR TIMING *)
VAR K:INTEGER;
BEGIN K:=0; WHILE K<=N DO K:=K+1 END;

FUNC MARK;                       (* MEASURE DURATION OF MARK *)
VAR K:INTEGER;                   (* LOCAL INTEGER K *)
BEGIN
   K:=0;                         (* INITIALIZE MARK COUNTER *)
      REPEAT   OUTP(255,0);      (* CLEAR CASSETTE LATCH *)
             K:=K+1;             (* INCREMENT MARK COUNTER *)
            DELAY(1)             (* WAIT A LITTLE WHILE *)
        UNTIL (INP(255)=127);    (* EXIT IF BIT 7 STAYS OFF *)
MARK:=K                          (* RETURN VALUE OF MARK *)
END;   (* OF MARK *)

FUNC SPACE;                      (* MEASURE DURATION OF SPACES *)
VAR K:INTEGER;                   (* LOCAL COUNTER DIFFERENT FROM MARK *)
BEGIN
   K:=0;                         (* INITIALIZE SPACE COUNTER *)
      REPEAT OUTP(255,1);        (* KEEP TIMING SAME AS MARK *)
             K:=K+1;             (* INCREMENT SPACE COUNTER *)
            DELAY(1)             (* WAIT A LITTLE WHILE *)
        UNTIL (INP(255)=255);    (* TIL CASSETTE LATCH SEES SOMETHING *)
SPACE:=K                         (* RETURN VALUE OF SPACE *)
END;    (* OF SPACE *)

PROC SORT(START,STOP);  (* SORTS ARRAY X FROM LOW TO HIGH POINTS *)
VAR I,TEMP,II : INTEGER;   (* LOCAL VARIABLES *)
BEGIN          (* THE BUBBLE SORT *)
   FOR I:= START TO (STOP--1) DO
      FOR II:= 1 TO (STOP-1)-(I-1) DO
      BEGIN TEMP:=X(II);
            IF X(II+1)(X(II) THEN BEGIN X(II):=X(II+1);
                                      X(II+1):=TEMP
                                 END
      END
END;  (* OF SORT *)

FUNC AVERAGE(START,STOP); (* GIVES AVG OF X ARRAY FROM START TO STOP *)
VAR I,SUM :INTEGER;       (* LOCAL VARIABLES *)
BEGIN
   SUM:=0;                 (* INITIALIZE SUM TO ZERO *)
   FOR I:= START TO STOP DO SUM:=SUM+X(I); (* FORM THE SUM *)
   AVERAGE:= SUM DIV (STOP-START+1)        (* CALCULATE AVERAGE *)
END;  (* OF AVERAGE *)             (* END OF SUPPORT FOR WPM PROG *)
```

Program continued

```
BEGIN (* MAIN PROGRAM FOR MORSE CODE WPM MEASUREMENT *)
   WRITE('HOW MANY MARK SAMPLES ? ');READ(TOTAL#);
   IF TOTAL )100 THEN TOTAL:=100;  (* KEEP TOTAL LEGIT *)
   FOR J:=1 TO TOTAL DO BEGIN S:=SPACE; (* IGNORE SPACES *)(* GATHER *)
                              X(J):=MARK (* AND STORE MARKS *)(* DATA*)
                        END;
   FOR J:=1 TO TOTAL DO WRITE(X(J)#,9); (* PRINT RAW DATA *)
   WRITE(13,'===========================================',13);
   READ(NUM); (* USER HIT KEY WHEN READY TO VIEW MORE DATA *)
   SORT(1,TOTAL); (* SORT THE ENTIRE ARRAY OF MEASUREMENTS *)
   AVG:=AVERAGE(1,TOTAL); (* COMPUTE GROSS AVERAGE WPM *)
   FOR J:=1 TO TOTAL DO WRITE(X(J)#,9); (* PRINT SORTED MARK DATA *)
   WRITE(13,'=============================================',13);
   READ(NUM); (* USER HIT KEY WHEN READY TO VIEW MORE DATA *)
   J:=0;
   REPEAT   J:=J+1  (* FIND WHERE THE MARKS START AND SPACES END *)
   UNTIL (    (X(J))AVG) OR (J)TOTAL)    );
   DITAVG:=AVERAGE(1,J-1);      (* AVERAGE LENGTH OF THE DITS *)
   DAHAVG:=AVERAGE(J,TOTAL);    (* AVERAGE LENGTH OF THE DAHS *)
   THRESHOLD:= (DITAVG+DAHAVG)DIV 2;  (* MIDWAY BETWEEN DITS AND DAHS *)
   WRITE(13);                   (* REPORT THE FINDINGS *)
   WRITE('AGGREGATE AVERAGE = ',AVG#,13);
   WRITE(' DIVIDING SUBSCRIPT IN X-ARRAY = ',J#,13);
   WRITE('DIT AVERAGE         = ',DITAVG#,13);
   WRITE('DAH AVERAGE         = ',DAHAVG#,13);
   WRITE('THRESHOLD           = ',THRESHOLD#,13);
   WRITE(' WPM BASED ON DITAVG = ', (234 DIV DITAVG)#,13);
   WRITE(' WPM BASED ON DAHAVG = ', (659 DIV DAHAVG)#,13);
END. (* OF MORSE CODE WPM MEASURING PROGRAM *)
```

Morse Code Reading Program (32K Needed)

This program reads the audio signals from the radio receiver at the black earphone input plug. No special electronics is needed. It decodes a fair range of code speeds without special modification.

The main program detects the longer silent period between letters in order to separate the dits and dahs that form letters. It counts the number of code elements (NUM) in a character and forms a binary number (CODE) consisting of zeros and ones. The zeros represent dits and the ones represent dahs.

Armed with NUM and CODE, the procedure DECODE scans its CASE statements to find the matching letter, digit, or punctuation mark. If a legal character is found, it is returned for printing. If there is a long period of silence, the program prints a blank and ceases printing until another character is received. An error causes a dollar sign to be printed.

It's easy to add more codes to the DECODE procedure. Simply find the CASE for the number of code elements, NUM, and add an extra CASE for the CODE of that new character. The less-than sign (<), for example, would be assigned the code ..-- (dit dit dah dah). There are NUM=4 code elements. The binary code is 0011, so CODE=3. The source program is edited as follows:

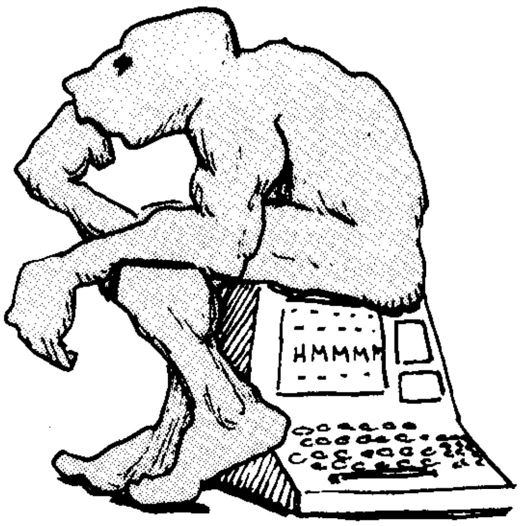
```
4: BEGIN
  CASE CODE OF
    0: L:="H" ; 1: L:="V" ; 2: L:="F" ; 3: L:="<" ;
```

The text of the Morse code decoder program follows:

```
VAR CODE,NUM,L,TEMP,LP : INTEGER; (* SUPPORT PROC'S FOR MORSE DECODER *)
                                  (* SEE WPM PROGRAM FOR INFO *)
PROC DELAY(N); (* CREATE A DELAY PROPORTIONAL TO N *)
VAR K:INTEGER;
BEGIN K:=0; WHILE K<=N DO K:=K+1 END;

FUNC MARK; (* RETURN DURATION OF MARKS (DITS/DAHS) *)
VAR K:INTEGER;
BEGIN
   K:=0;
      REPEAT   OUTP(255,0);
               K:=K+1;
               DELAY(1)
      UNTIL (INP(255)=127)OR(K)50); (* EXIT IF MARK TOO LONG *)
   MARK:=K
END;  (* OF MARK *)

FUNC SPACE; (* MEASURE DURATION OF SPACES BETWEEN THE MARKS *)
VAR K:INTEGER;
BEGIN
   K:=0;
      REPEAT OUTP(255,1);
             K:=K+1;
             DELAY(1)
      UNTIL (INP(255)=255)OR(K)50); (* EXIT IF SPACE TOO LONG *)
SPACE:=K
END;  (* OF SPACE *)


PROC DECODE; (* RETURNS CHARACTER IN GLOBAL L GIVEN NUM & CODE *)
BEGIN        (* SUPPORTS MORSE CODE DECODER PROGRAM *)
   CASE NUM OF
0: BEGIN L:=' ' END;           (* NO CODE ELEMENTS GIVEN *)
1: BEGIN                       (* ONE CODE ELEMENT GIVEN *)
      CASE CODE OF
         0: L:='E';    1: L:='T'
      END
   END;
2: BEGIN                       (* TWO CODE ELEMENTS GIVEN *)
      CASE CODE OF
         0: L:='I';    1: L:='A';    2: L:='N';    3: L:='M'
      END
   END;
3: BEGIN                       (* THREE CODE ELEMENTS GIVEN *)
      CASE CODE OF
         0: L:='S';    1: L:='U';    2: L:='R';    3: L:='W';
         4: L:='D';    5: L:='K';    6: L:='G';    7: L:='O'
      END
   END;
4: BEGIN                       (* FOUR CODE ELEMENTS GIVEN *)
      CASE CODE OF
         0: L:='H';    1: L:='V';    2: L:='F';
         4: L:='L';    6: L:='P';    7: L:='J';
         8: L:='B';    9: L:='X';   10: L:='C';   11: L:='Y';
        12: L:='Z';   13: L:='Q'
      END
   END;
5:   BEGIN                     (* FIVE CODE ELEMENTS GIVEN *)
        CASE CODE OF
            0: L:='5';    1: L:='4';    3: L:='3';    7: L:='2';
           15: L:='1';   16: L:='6';   24: L:='7';
           28: L:='8';   30: L:='9';   31: L:='0';   18: L:='/';
            10: BEGIN L:=' '; WRITE(' (END OF MESSAGE) ') END
        END
     END;
6:   BEGIN                     (* SIX CODE ELEMENTS GIVEN *)
```

Program continued

```
          CASE CODE OF
            12: L:='?';  21: L:='.';  33: L:='-';
            51: L:=','
          END
        END;
8:    BEGIN                        (* EIGHT CODE ELEMENTS GIVEN *)
          CASE CODE OF
             0: BEGIN L:=' ';WRITE(' (ERROR) ') END
          END
        END
END
END; (* OF DECODE PROCEDURE *)


BEGIN (* MORSE CODE DECODER MAIN PROGRAM BEGINS *)
REPEAT
   CODE:=0 ; NUM:=0 ; LP:=L ; L:='$';
   WHILE SPACE<=20 DO
      BEGIN   TEMP:=MARK;
              IF (TEMP>=20)AND(TEMP<50)
              THEN CODE:= 1+(CODE SHL 1) ELSE CODE:=CODE SHL 1;
              NUM:=NUM+1
      END;
   DECODE;
   IF (LP=' ')AND(L=' ') THEN ELSE WRITE(L) ; LP:=L
UNTIL 1=2     (* LOOP FOREVER *)
END. (* OF MORSE CODE DECODER PROGRAM *)
```



11

Beginning Scrolling

by Roger B. Wilcox

System Requirements:
Model I
16K RAM
Disk BASIC

You don't have to have an assembler or much experience to use this assembly-language program (see Figure 1). It combines a program by Roger Fuller to scroll a window of lines up and down (*80 Microcomputing*, January 1982, p. 30), and a program by M. Keller to scroll a number of lines down while protecting the top and bottom lines (*80 Microcomputing*, February 1982, p. 264).

This program lets you scroll forward and backward through long lists of data, using screen lines 3 through 14, while headings or instructions are protected on the top and bottom lines. It has two entry points, called by USR0 and USR1. There is one exit point, at the end. Let's look at the part of the program that moves the lines on the screen up (USR1 section of the Program Listing).

The key machine-language instruction is LDIR, and this section of the program is built around it. LDIR moves a byte of data from the HL register pair to the DE register pair, and checks to see if register pair BC is zero. If not, BC is decremented, and HL and DE are incremented. This sequence repeats until BC is zero. LDIR acts like a single instruction FOR-NEXT loop, using the BC, HL, and DE register pairs.

If HL has the starting address of a line in video memory, DE has the starting address of the next line up, and BC is initialized to the number of bytes to move (11 lines = 704). You can see how LDIR begins with the first line and moves it byte-by-byte up to the next line. LDIR automatically sets itself up to transfer the next line, and continues to drop down through the lines on the screen until BC = 0 at the end of screen line 14.

DATA	MNEMONIC	REMARKS
	**** MOVE DOWN, USR0 ****	
33,63,63	LD HL,16191	End of line 13
17,127,63	LD DE,16255	End of line 14
1,192,2	LD BC,704	Number bytes to move
237,184	LDDR	Move lines down
35	INC HL	Get start of top line
24,13	JR 13 (ERASE)	Erase top line
	**** MOVE UP, USR1 ****	
33,192,60	LD HL,15552	Start of line 4
17,128,60	LD DE,15488	Start of line 3
1,192,2	LD BC,704	Number bytes to move
237,176	LDIR	Move lines up
213	PUSH DE	Put DE in stack
225	POP HL	HL=DE
	**** ERASE ****	
62,32 (ERASE)	LD A," "	Load A with space
6,64	LD B,64	Number bytes to clear
119 (LOOP)	LD (HL),A	Clear one byte
35	INC HL	Next byte
16,252	DJNZ -4 (LOOP)	Loop until B=0
201	RET	Return to BASIC

Figure 1

The program begins by initializing HL, DE, and BC for use by the LDIR instruction. (*Initialization* means to establish a starting value.) This is done with the mnemonic LD dd,nn, where dd represents a register pair, and nn is the least significant byte (LSB) and most significant byte (MSB) of the number (video memory address, in this case). The Z80 code listing shows the format for this instruction as 00dd0001,n,n. The binary value of dd varies from 0 (00) to 3 (11) as follows: 0=BC, 1=DE, 2=HL, 3=SP. For HL the binary part of the format becomes 00100001, which is decimal 33 from the conversion table.

The BASIC program uses line 1 as a protected line, and line 2 is reserved as a space, so line 3 is the top of the scrolling lines. Line 4 is the first line to be moved up. Video memory starts at decimal address 15360, and line 4 starts at 15552 (15360+3*64), which converts to LSB=192 and MSB=60. Therefore, the data representing the first instruction of this section of the program is 33, 192, 60. This procedure is repeated for LD DE, 15488 (the starting address of line 3, the next line up) and for LD BC, 704.

Initialization complete, the next instruction is LDIR, represented by decimal 237, 176. When this section of the program is called by a USR1 statement, 11 lines on the screen (4-14) are moved up one-by-one. The bottom scrolling line must be erased to provide a clean space to print the next line of data from BASIC. Either LD (DE),A or LD (HL),r can be used to clear the last line, but here I chose LD (HL),r. After the last LDIR incrementing of DE, DE contains the starting address of line 14, which is to be erased using HL. A quick way to transfer this address to HL is to PUSH the value in DE into the stack and then POP it back into HL.

The erase operation loads register A with a space (ASCII code 32), then loads the HL address with the contents of A. Register B is used for a byte counter by loading it with the number of bytes to clear (one line equals 64 bytes).

The B register is used for the counter because the DJNZ e instruction used for the loop (like GOTO in BASIC) checks B for a non-zero status, and also decrements B, before performing a relative jump. The value of e is the number of bytes to jump, and requires an explanation. In this case, you want to jump backward in the code to the start of the LD (HL),A instruction. Counting the number of bytes to jump involves the program counter (PC). After an instruction is read, the PC has the address of the byte *after* the instruction. Relative jumps are counted from this byte, to the first byte of the target instruction. Here, it is a jump of four bytes. Since this is a backward jump, the value is a negative number. Negative numbers in assembly language must be represented by their two's complement. In simple terms that means that 255 = −1, 254 = −2, etc. The −4 becomes 256−4 = 252, and the data values for DJNZ −4 are 16, 252.

Just prior to the loop, HL is incremented, so the next byte is cleared each time the loop is performed. At this point, 11 lines of the screen have been moved up one line and the last line has been erased. A RET instruction returns control to the BASIC program.

The (earlier) section of the program that moves lines down is called by USR0. It is similar to the move-up routine, except that LDDR does the moving instead of LDIR. The difference is that LDDR decrements HL and DE. Therefore, the screen address used to initialize HL and DE must be the end of the lines. Line 16 is a protected line, and line 15 is a space between the scrolling lines and the bottom. Line 13 is the first to move down (to line 14). The starting address of line 13 is 15360+12*64 = 16128, and the end of line 13 is 16128+63 = 16191, which is loaded into HL. Register pair DE, for line 14, is handled in a similar manner. BC is loaded again with the number of bytes in 11 lines (704).

After the last LDDR decrement (BC = 0), HL has the address of the end of line 2, which is above the scrolling line. Incrementing HL moves it down to the starting address of the top scrolling line (line 3), so that it can be cleared. Since the erase routine uses HL to clear a line, it services both the move-up and move-down sections of the program.

This program listing can be incorporated into other programs, and modified if necessary. As long as there is a blank line above and below the scrolling lines, the ERASE routine can be deleted; make the number of bytes to be moved equal to 12 lines (768 bytes), and the blank line is moved to clear the last scrolling line. The RET instruction replaces the

LD A,"" instruction. The INC HL and PUSH/POP are no longer needed, and the 2-byte JR 13 can be replaced by a 1-byte RET. With these modifications you should end up with a 24-byte assembly program. Change the POKE limits in the BASIC program to agree with the number of bytes.

Program Listing

```
5 '*** SCROLL UP AND DOWN WITH PROTECTED LINES TOP & BOTTOM
10 POKE16561,87:POKE16562,127 '*** SET MEMORY SIZE TO 32599
20 CLEAR1000:DIM PA$(26):CLS
30 FORX=32600TO32635:READJ:POKEX,J:NEXT
40 DATA 33,63,63,17,127,63,1,192,2,237,184,35,24,13
50 DATA 33,192,60,17,128,60,1,192,2,237,176,213,225
60 DATA 62,32,6,64,119,35,16,252,201
70 DEFUSR0=32600:DEFUSR1=32614:DEFINT K,L,X:KA=14400
72 '****************
74 '*  THE PROGRAM THAT CREATES THE DATA TO BE SCROLLED
76 '*  GOES IN THIS AREA.  LINE 80 IS A DEMO PROGRAM.
80 FORX=1TO26:PA$(X)=STRING$(10,64+X):NEXT
82 '*  CHANGE LINES 90 AND 100 TO MATCH YOUR PROGRAM NEEDS.
84 '*  CHANGE PN TO EQUAL NUMBER OF DATA LINES.
86 '****************
90 PRINT@960,"PRESS: <[> TO SCROLL UP; <"CHR$(92)"> TO SCROLL DO
WN; <CLEAR> FOR MENU";
100 PRINT@0,"                    ITEM  NO.         DATA":PRINT
110 FORX=1TO12:PRINTTAB(20)X,PA$(X):NEXT
120 LE=12:LT=1:PN=26
130 K=PEEK(KA):IF K=8 AND LT>1 THEN LT=LT-1:LE=LE-1:PRINT@USR0(1
48),LT,PA$(LT);
140 IF K=16 AND LE<PN THEN LE=LE+1:LT=LT+1:PRINT@USR1(852),LE,PA
$(LE);
150 IF K=2 THEN CLS:STOP '*** REPLACE WITH RETURN TO MENU, ETC.
160 GOTO130
```

12

Line Drawing

by Daniel Lindsey

System Requirements:
Level II or Disk BASIC
Model I or III
16K RAM
Editor/assembler

BASIC is too slow for games with graphics. Level II BASIC is flexible, but falls short when high-speed drawing is required. The STRING$ function helps, but plots only horizontal lines.

There are ways around this problem. You can buy a BASIC compiler to convert programs into machine code, but this is expensive. You can suffer with BASIC, or you can write an assembly-language routine to draw lines, accessing it through the USR(n) function from BASIC.

Operation

Program Listing 1 draws a line between any two points on the screen given the (x,y) coordinates of both points. Program operation is easier to follow from the algorithm, outlining the flow of the program in non-computer language (Figure 1).

To understand the program's operation, divide it into four sections: the BASIC interface, the slope calculator, the x- or y-plot setup, and the plot routine.

The BASIC interface links the BASIC program to the plot subroutine. It compares the number received from the USR call to the known command numbers, jumping to whatever routine is requested, or returning on an illegal command.

Before calling the plot subroutine, each routine sets a flag signalling the set routine to erase or draw. If the command is a table operation, the program moves a pointer through the table of coordinates, drawing or erasing lines. The BASIC interface provides the necessary functions that use the plot subroutine from a BASIC program.

The slope of a line is the ratio between the change in y- and x-coordinates. Multiplying the slope of a line by an x-coordinate results in its corresponding y-coordinate. This lets the plot routine produce the x- and y-coordinates along the requested line. However, as the line approaches vertical, the slope approaches an infinite value. If the line is at an angle greater than 45 degrees, the routine switches the axis along which it counts.

The slope calculator gathers the necessary values, then generates the slope of the requested line. Since this is the entry point of the plot subroutine, the routine clears the temporary storage area. It then collects the two x-coordinates from the table and solves for the change in x. After doing the same for the y-coordinates, the routine compares the change in x with the change in y. If the change in y is greater than the change in x, then the angle of the line is greater than 45 degrees from the x-axis, and the routine must plot along the y-axis to avoid a slope that is greater than one. Once the routine has branched on a y-plot or continued on an x-plot, the registers are loaded with the change in x and the change in y, and the division subroutine is called to calculate the slope. The slope calculator produces a slope and the axis from which it was taken.

Figure 1. *Plot Algorithm*

```
BASIC INTERFACE
1. Find n of USR(n)
2. Cases—If:
   n=0  then (line draw)
        set flag to draw mode
        jump to draw program
   n=1  then (line erase)
        set flag to erase mode
        jump to draw program
   n=2  then (table draw)
        point to table of coordinates
        set flag to draw mode
        Until second x-coordinate is greater than 127
              call draw program (as a subroutine)
              move pointer down through the table
        End Until
        return to BASIC
   n=3  then (table erase)
        point to table of coordinates
        set flag to erase mode
        Until second x-coordinate is greater than 127
              call draw program (as a subroutine)
              move pointer down through the table
        End Until
   End Cases
```

```
SLOPE CALCULATOR
1. Clear storage area
2. Calculate the change in x-coordinates
3. Calculate the change in y-coordinates
4. If the change in y is greater than the change in x
         (plot along the y-axis)
         calculate slope: change in x/change in y
         jump to plot setup (y-plot)
   Else:
         (plot along x-axis)
         calculate slope: change in y/change in x
         jump to plot setup (x-plot)
Y-PLOT SETUP
1. Load the counter with the smaller y (starting count)
2. Load memory with the larger y (final count)
3. If the starting count x-coordinate is greater than the final count x-coordinate then
         signal a negative slope
         Else
                signal a positive slope
4. Jump to plot section
X-PLOT SETUP
1. Load the count with the smaller x (starting count)
2. Load memory with the larger x (final count)
3. If the starting count y-coordinate is greater than the final count y-coordinate, then
         signal a negative slope
   Else
         signal a positive slope
4. Jump to plot section
PLOT
1. While count is less than or equal to final count:
         If plot flag is equal to x-plot then,
                load the HL registers with the (x,y) coordinates
   Else
         load the HL registers with the (y,x) coordinates
   End If (the (x,y) coordinates are the starting count and the current alternate count,
         which was set to a starting value in the setup section)
   Call set subroutine to light pixel
   If angle of line is 45 degrees then
         If slope of the line is negative then
                decrement alternate count
         Else
                increment alternate count
   Else
         If the line's slope is negative then
                add the two's complement of the slope to the alternate count
         Else
                add the slope of the line to the alternate count
         End If
         Increment count
   End While
```

The x- or y-plot setup initializes the registers and memory necessary for the plot routine to generate the x- and y-coordinates between the two given points. The x- and y-plot routines work the same way. The only difference is that the x- and y-coordinates are switched. The x-plot routine compares the two x-coordinates, storing the larger as a final count in temporary storage, and the smaller as an initial count in the C register. The y-coordinate that corresponds to the initial count is stored in the D register as an initial y. The DE register pair stores the y-coordinate corresponding to the larger x-coordinate.

The D register contains the integer y-count, and the E register contains the decimal portion of the y-count. The routine compares the initial and final y, signalling a negative or positive slope in the temporary storage area. The routine signals an x-plot in the temporary storage, saves the slope in the B register, and passes control to the plot routine. The x- or y-plot setup prepares all values needed by the plot routine to generate a line.

The plot routine uses the information gathered by the previous sections to generate all the x- and y-coordinates between the two given points. Depending on the flag set in one of the setup routines, this routine counts along an axis from the smallest x or y to the largest x or y. As the program loops through the routine, the slope in the B register is added to the alternate coordinate in the DE register pair. The current (x,y) coordinates are sent to the set subroutine through each loop of the plot routine. In the case of a 45-degree angle line (slope equals one), the count is incremented, and the alternate coordinate is either incremented or decremented, depending on the sign (plus or minus) of the slope. Once the line has been plotted, control is returned to either BASIC or the BASIC interface. The plot routine generates the actual values of the x- and y-coordinates between the given points, setting pixels (picture elements) as it loops through the routine.

Using the Routines

After making a machine-readable copy of the program, you are ready to start using the line subroutine in your computer. Always set memory size to 32000 before loading the program.

Before you can draw a line, you must store the coordinates of the end points of the line in memory. Simply POKE the first x in 32436, its corresponding y in 32437, the second x in 32438, and its corresponding y in 32439.

Once the coordinates are in memory you can erase or draw a line. To draw a line use USR(0), and to erase a line use USR(1). For example, to plot a line diagonally on the screen, POKE 0 into 32436 and 32437, POKE 127 into 32438, POKE 47 into 32439, and type M=USR(0). A white line will instantly appear from the upper left corner to the lower right corner of the screen.

While the line functions serve many useful purposes, many applications require figures with more than one line. The table draw functions (draw and erase) solve this problem.

They are similar to the line functions with one exception. The line functions draw only one line, while the table functions draw as many lines (connected) as the table can hold. The following example will help you understand the operation of the table functions.

Location	Value
32436	0
32437	0
32438	127
32439	0
32440	127
32441	47
32442	0
32443	47
32444	0
32445	0
32446	255

Figure 2

To draw a border, place the numbers shown in Figure 2 into the indicated memory locations. After these values are in memory, USR(2) will draw the border, and USR(3) will erase it. To build a table, simply place the (x,y) coordinates of the points in the order they are to be drawn in ascending memory. Terminate the list with a number greater than 127.

Using these functions, you should be able to draw lines at an average speed of 50 lines per second. This greatly reduces the time that it takes for a program to set up a screen. Try the examples in Program Listings 2 and 3.

This program is designed for a cassette-based Model I or III machine. Both Level II and Model III BASIC require the starting address of any machine-language subroutines to be stored in memory addresses 16526 and 16527. The usual approach is to POKE the correct values in from the BASIC program and then do a USR(n) to call the machine-language routine. This program takes a different approach: lines 280 and 290 of the assembly-language routine take care of loading the correct value, 7D00H (32000 decimal) into locations 16526 and 16527.

Program Listing 1

```
00100 ;
00110 ;                * LINE DRAW *
00120 ;
00130 ; By Daniel Lindsey (1/1/81) (C)
```

Program continued

```
               ØØ14Ø ;
               ØØ15Ø ;       This program draws a graphics line between any
               ØØ16Ø ; two points on the screen; given the (X,Y) coordinates
               ØØ17Ø ; of both points.
               ØØ18Ø ;
               ØØ19Ø ;       Commands:
               ØØ2ØØ ;
               ØØ21Ø ; USR(Ø)-Draw a single line
               ØØ22Ø ; USR(1)-Erase a single line
               ØØ23Ø ; USR(2)-Draw a set of lines from a table
               ØØ24Ø ; USR(3)-Erase a set of line from a table
               ØØ25Ø ;
               ØØ26Ø ;       *** BASIC INTERFACE ***
               ØØ27Ø ;
4Ø8E           ØØ28Ø         ORG     16526
4Ø8E ØØ7D      ØØ29Ø         DEFW    BEGIN          ;Define entry-USR call
7DØØ           ØØ3ØØ         ORG     7DØØH          ;Actual start of program
7DØØ CD7FØA    ØØ31Ø BEGIN   CALL    ØA7FH          ;Get N of USR(N)
7DØ3 7D        ØØ32Ø         LD      A,L            ;Value of N
7DØ4 FEØØ      ØØ33Ø         CP      Ø              ;Check for draw
7DØ6 2815      ØØ34Ø         JR      Z,DRAW         ;Jump if draw
7DØ8 FEØ1      ØØ35Ø         CP      1              ;Check for erase
7DØA 28Ø9      ØØ36Ø         JR      Z,ERASE        ;Jump if erase
7DØC FEØ2      ØØ37Ø         CP      2              ;Check for table draw
7DØE 2815      ØØ38Ø         JR      Z,TDRAW        ;Jump if table draw
7D1Ø FEØ3      ØØ39Ø         CP      3              ;Check for table erase
7D12 2829      ØØ4ØØ         JR      Z,TERASE       ;Jump if table erase
7D14 C9        ØØ41Ø         RET                    ;Return to BASIC
7D15 21B37E    ØØ42Ø ERASE   LD      HL,ESTOR       ;Pointer to flag
7D18 36ØØ      ØØ43Ø         LD      (HL),Ø         ;Set flag to erase
7D1A C3557D    ØØ44Ø         JP      START          ;Go erase line
7D1D 21B37E    ØØ45Ø DRAW    LD      HL,ESTOR       ;Pointer to flag
7D2Ø 36Ø1      ØØ46Ø         LD      (HL),1         ;Set flag to draw
7D22 C3557D    ØØ47Ø         JP      START          ;Go draw line
7D25 FD21B47E  ØØ48Ø TDRAW   LD      IY,TABLE       ;Point to coordinates
7D29 21B37E    ØØ49Ø         LD      HL,ESTOR       ;Point to flag
7D2C 36Ø1      ØØ5ØØ         LD      (HL),1         ;Set flag to draw
7D2E FDCBØ27E  ØØ51Ø TD1     BIT     7,(IY+2)       ;Second X coordinate
7D32 CØ        ØØ52Ø         RET     NZ             ;Return if X>127
7D33 CD597D    ØØ53Ø         CALL    START2         ;Draw line
7D36 FD23      ØØ54Ø         INC     IY             ;Move pointer through
7D38 FD23      ØØ55Ø         INC     IY             ;   the table
7D3A C32E7D    ØØ56Ø         JP      TD1            ;Jump to top of loop
7D3D FD21B47E  ØØ57Ø TERASE  LD      IY,TABLE       ;Point to Coordinates
7D41 21B37E    ØØ58Ø         LD      HL,ESTOR       ;Point to flag
7D44 36ØØ      ØØ59Ø         LD      (HL),Ø         ;Set flag to erase
7D46 FDCBØ27E  ØØ6ØØ TE1     BIT     7,(IY+2)       ;Second X coordinate
7D4A CØ        ØØ61Ø         RET     NZ             ;Return if X>127
7D4B CD597D    ØØ62Ø         CALL    START2         ;Erase line
7D4E FD23      ØØ63Ø         INC     IY             ;Move pointer through
7D5Ø FD23      ØØ64Ø         INC     IY             ;   the table
7D52 C32E7D    ØØ65Ø         JP      TD1            ;Jump to top of loop
               ØØ66Ø ;
               ØØ67Ø ;       *** END OF BASIC INTERFACE ***
               ØØ68Ø ;
               ØØ69Ø ;       *** SLOPE CALCULATOR ***
               ØØ7ØØ ;
7D55 FD21B47E  ØØ71Ø START   LD      IY,TABLE       ;Pointer in table
7D59 DD21B17E  ØØ72Ø START2  LD      IX,STORE       ;Pointer in temp. storage
7D5D AF        ØØ73Ø         XOR     A              ;Clear A
7D5E DD77ØØ    ØØ74Ø         LD      (IX+Ø),A       ;Set storage area to zero
```

```
7D61 DD77Ø1   ØØ75Ø         LD      (IX+1),A
7D64 FD56ØØ   ØØ76Ø         LD      D,(IY+Ø)       ;Load in X coordinates
7D67 FD7EØ2   ØØ77Ø         LD      A,(IY+2)
7D6A 92       ØØ78Ø         SUB     D              ;Find change in X's
7D6B F27Ø7D   ØØ79Ø         JP      P,CONT1        ;Jump if positive
7D6E ED44     ØØ8ØØ         NEG                    ;Change to positive value
7D7Ø 57       ØØ81Ø CONT1   LD      D,A            ;Save change in X's
7D71 FD5EØ1   ØØ82Ø         LD      E,(IY+1)       ;Load in Y coordinates
7D74 FD7EØ3   ØØ83Ø         LD      A,(IY+3)
7D77 93       ØØ84Ø         SUB     E              ;Find the change in Y's
7D78 F27D7D   ØØ85Ø         JP      P,CONT2        ;Jump if result positive
7D7B ED44     ØØ86Ø         NEG                    ;Change to positive value
7D7D 5F       ØØ87Ø CONT2   LD      E,A            ;Save change in Y's
7D7E 92       ØØ88Ø         SUB     D              ;Find change in Y's-X's
7D7F F28A7D   ØØ89Ø         JP      P,CONT3        ;Jump if slope positive
7D82 4B       ØØ9ØØ         LD      C,E            ;Setup for divide routine
7D83 62       ØØ91Ø         LD      H,D            ;   X's/Y's
7D84 CD437E   ØØ92Ø         CALL    DIV            ;Calculate slope
7D87 C3927D   ØØ93Ø         JP      XPLOT          ;Jump to next section
7D8A 63       ØØ94Ø CONT3   LD      H,E
7D8B 4A       ØØ95Ø         LD      C,D
7D8C CD437E   ØØ96Ø         CALL    DIV
7D8F C3C67D   ØØ97Ø         JP      YPLOT
              ØØ98Ø ;
              ØØ99Ø ;       *** END OF SLOPE CALCULATOR ***
              Ø1ØØØ ;
              Ø1Ø1Ø ;       *** SETUP FOR PLOT ROUTINE ***
              Ø1Ø2Ø ;
7D92 FD7EØØ   Ø1Ø3Ø XPLOT   LD      A,(IY+Ø)       ;Load in the X's
7D95 FD46Ø2   Ø1Ø4Ø         LD      B,(IY+2)
7D98 B8       Ø1Ø5Ø         CP      B              ;Compare X's
7D99 F2A97D   Ø1Ø6Ø         JP      P,XP1          ;Jump if A>=B
7D9C 4F       Ø1Ø7Ø         LD      C,A            ;Save starting count
7D9D DD70ØØ   Ø1Ø8Ø         LD      (IX+Ø),B       ;Save final count
7DAØ FD7EØ1   Ø1Ø9Ø         LD      A,(IY+1)       ;Load in the Y's
7DA3 FD66Ø3   Ø11ØØ         LD      H,(IY+3)
7DA6 C3B37D   Ø111Ø         JP      XP2            ;Continue @ XP2
7DA9 48       Ø112Ø XP1     LD      C,B            ;Save starting count
7DAA DD77ØØ   Ø113Ø         LD      (IX+Ø),A       ;Save final count
7DAD FD7EØ3   Ø114Ø         LD      A,(IY+3)       ;Load in the Y's
7DBØ FD66Ø1   Ø115Ø         LD      H,(IY+1)
7DB3 57       Ø116Ø XP2     LD      D,A            ;Save initial Y
7DB4 1EØØ     Ø117Ø         LD      E,Ø            ;Clear E
7DB6 DDCBØ1CE Ø118Ø         SET     1,(IX+1)       ;Signal XPLOT
7DBA BC       Ø119Ø         CP      H              ;Compare the Y's
7DBB FAC27D   Ø12ØØ         JP      M,XP3          ;Jump if slope positive
7DBE DDCBØ1D6 Ø121Ø         SET     2,(IX+1)       ;Signal negative slope
7DC2 45       Ø122Ø XP3     LD      B,L            ;Save slope
7DC3 C3F77D   Ø123Ø         JP      PLOT           ;Continue @ PLOT
7DC6 FD7EØ1   Ø124Ø YPLOT   LD      A,(IY+1)       ;Load in the Y's
7DC9 FD46Ø3   Ø125Ø         LD      B,(IY+3)
7DCC B8       Ø126Ø         CP      B              ;Compare the Y's
7DCD F2DD7D   Ø127Ø         JP      P,YP1          ;Jump if A>B
7DDØ 4F       Ø128Ø         LD      C,A            ;Save starting count
7DD1 DD70ØØ   Ø129Ø         LD      (IX+Ø),B       ;Save final count
7DD4 FD7EØØ   Ø13ØØ         LD      A,(IY+Ø)       ;Load in the X's
7DD7 FD66Ø2   Ø131Ø         LD      H,(IY+2)
7DDA C3E77D   Ø132Ø         JP      YP2            ;Continue @ YP2
7DDD 48       Ø133Ø YP1     LD      C,B            ;Save starting count
7DDE DD77ØØ   Ø134Ø         LD      (IX+Ø),A       ;Save final count
7DE1 FD7EØ2   Ø135Ø         LD      A,(IY+2)       ;Load in the X's
```

Program continued

```
7DE4 FD66ØØ   Ø136Ø        LD      H,(IY+Ø)
7DE7 57       Ø137Ø YP2    LD      D,A            ;Save the intial X
7DE8 1EØØ     Ø138Ø        LD      E,Ø            ;Clear E
7DEA DDCBØ18E Ø139Ø        RES     1,(IX+1)       ;Signal YPLOT
7DEE BC       Ø14ØØ        CP      H              ;Check for negative slope
7DEF FAF67D   Ø141Ø        JP      M,YP3          ;Jump if slope positive
7DF2 DDCBØ1D6 Ø142Ø        SET     2,(IX+1)       ;Signal a negative slope
7DF6 45       Ø143Ø YP3    LD      B,L            ;Save slope
              Ø144Ø ;
              Ø145Ø ;      *** END OF SETUP FOR PLOT ***
              Ø146Ø ;
              Ø147Ø ;      *** PLOT ***
              Ø148Ø ;
7DF7 79       Ø149Ø PLOT   LD      A,C            ;Get current count
7DF8 DDBEØØ   Ø15ØØ        CP      (IX+Ø)         ;Compare with final count
7DFB 28Ø1     Ø151Ø        JR      Z,PL1          ;Jump if last loop
7DFD FØ       Ø152Ø        RET     P              ;RETURN TO BASIC !
7DFE DDCBØ14E Ø153Ø PL1    BIT     1,(IX+1)       ;1-XPLOT Ø-YPLOT
7EØ2 28Ø5     Ø154Ø        JR      Z,PL2          ;Jump on YPLOT
7EØ4 61       Ø155Ø        LD      H,C            ;Load HL with X,Y
7EØ5 6A       Ø156Ø        LD      L,D            ;   coordinates for XPLOT
7EØ6 C3ØB7E   Ø157Ø        JP      PL3            ;Continue @ PL3
7EØ9 69       Ø158Ø PL2    LD      L,C            ;Load HL with X,Y
7EØA 62       Ø159Ø        LD      H,D            ;   coordinates for YPLOT
7EØB E5       Ø16ØØ PL3    PUSH    HL             ;Save coordinates
7EØC D9       Ø161Ø        EXX                    ;Swap registers
7EØD E1       Ø162Ø        POP     HL             ;Restore coordinates
7EØE CD5F7E   Ø163Ø        CALL    SET            ;Set point
7E11 D9       Ø164Ø        EXX                    ;Restore registers
7E12 DD21B17E Ø165Ø        LD      IX,STORE       ;Restore pointer
7E16 DDCBØ15E Ø166Ø        BIT     3,(IX+1)       ;Ø-ANGLE/NOT45,1-ANGLE 45
7E1A 28ØF     Ø167Ø        JR      Z,PL6          ;Jump if angle not 45
7E1C DDCBØ156 Ø168Ø        BIT     2,(IX+1)       ;Ø-pos. 1-negative slope
7E2Ø 28Ø4     Ø169Ø        JR      Z,PL4          ;Jump if slope positive
7E22 15       Ø17ØØ        DEC     D              ;Coordinate - 1
7E23 C3277E   Ø171Ø        JP      PL5            ;Continue @ PL5
7E26 14       Ø172Ø PL4    INC     D              ;Coordinate + 1
7E27 ØC       Ø173Ø PL5    INC     C              ;Count=Count+1
7E28 C3F77D   Ø174Ø        JP      PLOT           ;Jump to top of loop
7E2B DDCBØ156 Ø175Ø PL6    BIT     2,(IX+1)       ;Ø-pos. 1-negative slope
7E2F 28Ø9     Ø176Ø        JR      Z,PL7          ;Jump if slope positive
7E31 26FF     Ø177Ø        LD      H,255          ;Set all bits in H
7E33 78       Ø178Ø        LD      A,B            ;Load with slope
7E34 2F       Ø179Ø        CPL                    ;Complement
7E35 6F       Ø18ØØ        LD      L,A            ;HL has 2's complement
7E36 23       Ø181Ø        INC     HL             ;Complete 2's complement
7E37 C33D7E   Ø182Ø        JP      PL8            ;Go add negative slope
7E3A 26ØØ     Ø183Ø PL7    LD      H,Ø            ;Clear H
7E3C 68       Ø184Ø        LD      L,B            ;Positive slope in HL
7E3D 19       Ø185Ø PL8    ADD     HL,DE          ;Add slope to count
7E3E EB       Ø186Ø        EX      DE,HL          ;Store result in DE
7E3F ØC       Ø187Ø        INC     C              ;Add one to count
7E4Ø C3F77D   Ø188Ø        JP      PLOT           ;Jump to top of loop
              Ø189Ø ;
              Ø19ØØ ;      *** END OF PLOT SECTION ***
              Ø191Ø ;
              Ø192Ø ;      *** SUBROUTINE DIVIDE ***
              Ø193Ø ;
7E43 7C       Ø194Ø DIV    LD      A,H            ;Check for 45 deg. angle
7E44 91       Ø195Ø        SUB     C              ;   & Jump if not 45
7E45 2ØØ5     Ø196Ø        JR      NZ,DV1         ;
```

```
7E47 DDCBØ1DE Ø1970          SET    3,(IX+1)      ;Signal a 45 deg. angle
7E4B C9       Ø1980          RET                  ;Return to prog.
7E4C 2EØØ     Ø1990 DV1      LD     L,Ø           ;Clear result
7E4E Ø609     Ø2000          LD     B,9           ;Number of loops
7E5Ø CB25     Ø2010 DV2      SLA    L             ;Shift result
7E52 79       Ø2020          LD     A,C           ;Get D2
7E53 94       Ø2030          SUB    H             ;Subtract D1
7E54 FA5A7E   Ø2040          JP     M,DV3         ;Jump if not positive
7E57 4F       Ø2050          LD     C,A           ;Update D2
7E58 CBC5     Ø2060          SET    Ø,L           ;Set bit in result
7E5A CB21     Ø2070 DV3      SLA    C             ;Shift D2 left
7E5C 1ØF2     Ø2080          DJNZ   DV2           ;Loop untill done
7E5E C9       Ø2090          RET                  ;Return to prog.
              Ø2100 ;
              Ø2110 ;        *** END OF DIVIDE ***
              Ø2120 ;
              Ø2130 ;        *** SUBROUTINE SET ***
              Ø2140 ; By William Barden, 8Ø-MICROCOMPUTING, JUNE 198Ø, PG.24
              Ø2150 ;
7E5F 5C       Ø2160 SET      LD     E,H           ;X
7E6Ø 7D       Ø2170          LD     A,L           ;Y
7E61 CB3B     Ø2180          SRL    E             ;Get char position in E
7E63 16ØØ     Ø2190          LD     D,Ø           ;Set COL# to Ø
7E65 3ØØ1     Ø2200          JR     NC,SET1Ø      ;Go if COL#=Ø
7E67 14       Ø2210          INC    D             ;COL#=1
7E68 Ø6FF     Ø2220 SET1Ø    LD     B,ØFFH        ;-1 to B
7E6A Ø4       Ø2230 SET2Ø    INC    B             ;Bump quotient in B=LINE#
7E6B D6Ø3     Ø2240          SUB    3             ;Successive subt. for /3
7E6D F26A7E   Ø2250          JP     P,SET2Ø       ;Go if not negative
7E7Ø C6Ø3     Ø2260          ADD    A,3           ;Add back for remainder
7E72 Ø7       Ø2270          RLCA                 ;(ROW#)*2
7E73 82       Ø2280          ADD    A,D           ;(ROW#)*2+COL#=BIT POS.
7E74 4F       Ø2290          LD     C,A           ;Save BIT POS. in C
7E75 68       Ø2300          LD     L,B           ;Line #
7E76 26ØØ     Ø2310          LD     H,Ø           ;Now in HL
7E78 Ø6Ø6     Ø2320          LD     B,6           ;Shift count
7E7A 29       Ø2330 SET3Ø    ADD    HL,HL         ;Multiply LINE#*64
7E7B 1ØFD     Ø2340          DJNZ   SET3Ø         ;Loop till done
7E7D 16ØØ     Ø2350          LD     D,Ø           ;DE now has CHAR POS.
7E7F 19       Ø2360          ADD    HL,DE         ;LINE#*64+CHARPOS. in HL
7E8Ø 11ØØ3C   Ø2370          LD     DE,3CØØH      ;Start of video
7E83 19       Ø2380          ADD    HL,DE         ;Points to memory
7E84 Ø6ØØ     Ø2390          LD     B,Ø           ;BC now has bit pos.
7E86 DD21AB7E Ø2400          LD     IX,MASK       ;Start of mask table
7E8A DDØ9     Ø2410          ADD    IX,BC         ;Point to mask
7E8C 7E       Ø2420          LD     A,(HL)        ;Load pixel
7E8D CB7F     Ø2430          BIT    7,A           ;Check for graphics
7E8F 2ØØ2     Ø2440          JR     NZ,SET4Ø      ;Jump if graphics
7E91 3E8Ø     Ø2450          LD     A,128         ;Blank graphics
7E93 F5       Ø2460 SET4Ø    PUSH   AF            ;Save character
7E94 3AB37E   Ø2470          LD     A,(ESTOR)     ;Get draw flag
7E97 FEØØ     Ø2480          CP     Ø             ;Check for erase
7E99 2ØØA     Ø2490          JR     NZ,SET5Ø      ;Jump if not erase
7E9B DD7EØØ   Ø2500          LD     A,(IX+Ø)      ;Get mask pattern
7E9E 2F       Ø2510          CPL                  ;Complement for erase
7E9F CBFF     Ø2520          SET    7,A           ;Make graphics
7EA1 C1       Ø2530          POP    BC            ;Restore character
7EA2 AØ       Ø2540          AND    B             ;Combine with mask
7EA3 77       Ø2550          LD     (HL),A        ;Save in video memory
7EA4 C9       Ø2560          RET                  ;Return to PLOT
7EA5 F1       Ø2570 SET5Ø    POP    AF            ;Restore character
```

Program continued

```
7EA6 DDB600    02580          OR      (IX+0)          ;Set bit
7EA9 77        02590          LD      (HL),A          ;Save in video mem.
7EAA C9        02600          RET                     ;Return to PLOT
7EAB 81        02610 MASK     DEFB    81H             ;Mask table for set
7EAC 82        02620          DEFB    82H
7EAD 84        02630          DEFB    84H
7EAE 88        02640          DEFB    88H
7EAF 90        02650          DEFB    90H
7EB0 A0        02660          DEFB    0A0H
0002           02670 STORE    DEFS    2               ;#1-final count #2-flags
0001           02680 ESTOR    DEFS    1               ;Draw-1 Erase-0 flag
0004           02690 TABLE    DEFS    4               ;Table of coordinates
0000           02700          END
00000 TOTAL ERRORS
```

Program Listing 2

```
10 '
20 '                    *** LISTING 2 ***
30 '
40 ' * LIGHTNING
50 '
60 ' MEM SIZE 32000
70 'FOR DISK - DEFUSR=32000
80 DI=32436
90 X1=RND(127):X2=RND(127):Y1=RND(47):Y2=RND(47)
100 POKEDI,X1:POKEDI+1,Y1:POKEDI+2,X2:POKEDI+3,Y2:CLS:M=USR(0)
110 GOTO 90
```

Program Listing 3

```
5 '
10 '                    *** LISTING 3 ***
15 '
20 ' * RADAR SIMULATOR *
25 '
30 ' MEM SIZE 32000
35 'FOR DISK - DEFUSR=32000
40 C=3.141593/180:DI=32436:POKEDI+2,64:POKEDI+3,24
45 FOR A=0TO 359 STEP 10
50 X1=COS(A*C)*63+64
55 Y1=SIN(A*C)*23+24:M=USR(1)
60 POKEDI,X1:POKEDI+1,Y1:M=USR(0):NEXT:GOTO45
```

13

Defeating the ROM: Scroll Protection

by Daniel J. Scales

System Requirements:
Model I
Level II BASIC
16K

Have you ever wanted to protect text or graphics on the screen so it wouldn't be scrolled off? This capability can be used to keep the headings of a table from disappearing off the top of the screen, or to build up a graphics image step-by-step in command mode without having it move up and off the screen. When I have a lot of disk copying or erasing to do, I find it convenient to protect the top part of the screen and display a directory which remains there until I clear the screen or change the scroll protection.

The Program

The machine-language program in Program Listing 1 controls the scrolling of the screen. It is directly linked to the video output routine in ROM (read-only memory) that is used by Level II BASIC, Disk BASIC, and DOS. The scroll protection works in BASIC, DOS, and almost all machine-language utilities, as long as the memory for the routine is protected. Because it is linked to the ROM's video driver routine, the program is executed every time there is output to the screen, and is completely invisible to the user—no special command is needed every time there is a scroll.

The first part of the routine (see Program Listing 1), named PROT, intercepts the screen driver by placing its own starting address in the screen control block, which usually contains the address of the video output routine, and jumping to the video driver only after the scroll processing is completed. The other part of the machine-language program is called directly by the user to change the parameters of the scroll routine so that it protects a new specified text area, which may be any set of consecutive lines on the screen.

This SETUP routine is normally called from BASIC by a USR call. Its argument has the form 256*FIRST+LAST, where FIRST is the line number of the start of the text area that is to be scrolled, and LAST is the line number of the end of the text area. When the cursor is in the text area, only the text area is scrolled; the rest of the screen is protected. The cursor can still be placed anywhere on the screen by using a PRINT@ statement or by clearing the screen, and normal scrolling takes place if the cursor is not in the text area.

The scroll protection routine protects the top four lines of the screen (making the text area the last twelve lines of the screen) until SETUP is called to redefine the text area. The lines on the screen are numbered 0–15. If either the FIRST or LAST argument is not between 0 and 15, the argument is divided by 16, and the remainder is used as the line number. The SETUP routine does not check to see if FIRST is less than LAST. It is not possible for FIRST to equal LAST.

How It Works

Normally, the screen must be scrolled only if the cursor is at the last position of the screen and a character is to be printed, or if the cursor is anywhere on the last line of the screen and a line feed is to be printed. The machine-language routine, PROT, in Program Listing 1, does nothing unless the cursor is at the last position of the text area defined by SETUP or on the last line of the text area with a line feed to be printed. If one of these two conditions is true, it executes a special scroll routine that scrolls only the text area without spilling over into the protected regions of the screen. In this way, the scroll routine of the video print subroutine is defeated and is never invoked while the cursor is in the text area.

PROT starts by saving the contents of registers on the stack and loads the HL register with memory location 4020H. The screen image is stored in memory at locations 3C00H to 3FFFH, and 4020H contains the memory address associated with the cursor's location. The routine checks to see if the cursor is on the last character or last line of the text area and whether the character to be printed (which is in the C register on entry to PROT) is a carriage return/line feed (ASCII coded 0D hexadecimal). If both these conditions are true, the routine scrolls only the text area. The scrolling process uses the Z80 LDIR (load, increment, and repeat) instruction to move all the lines of the text area (except the first) up one line, then fills in the last line with blanks, and adjusts the cursor pointer in 4020H. The routine terminates by popping off the stack all the registers that were saved at the beginning of the routine and jumping to the video print routine.

SETUP is the routine which changes various instructions of the PROT routine so that it scrolls the right screen area according to the USR argument. It puts the USR argument in the HL register by calling a BASIC ROM routine, CVINT, at 0A7FH, as described in the Level II manual. It has a subroutine, M64, which takes a line number from 0 to 15 in the A register and returns the memory location asssociated with the first character of that line on the screen. It returns to BASIC after it has made all the necessary changes.

Using the Routine

The scroll protect routine is created using an editor/assembler and loaded in by the Level II SYSTEM command or the DOS LOAD command. The BASIC program in Program Listing 2 can be used to POKE the routine into memory. The POKE values are for a 32K machine. If the program is to POKE into the high memory of a 48K machine, all 191s in the DATA statements and in line 150 should be changed to 255s, and line 140 should start with FOR I= -256 TO -256+141. The routine as shown sets the bottom twelve lines of the screen as the text area and protects the top four lines. You can change the location of the text area by calling the SETUP routine with a USR call, so the address of SETUP must be known. This address is -16570 if the addresses of the BASIC program or assembly listing are not changed. If the 48K modification given above is used, then this address is -186. The lines of the screen are numbered 0-15, and the argument passed to SETUP must be 256*FIRST + LAST, where FIRST and LAST are the line numbers of the first and last lines of the text area. FIRST and LAST should be between 0 and 15, with FIRST less than LAST.

Only the text area scrolls as long as the cursor is in the text area, but if the cursor is not in the text area and is at the bottom of the screen, a normal unprotected scroll takes place. The CLEAR key is not disabled, so you can erase anything in the protected part of the screen at any time by clearing the screen. PROT can be modified to check whether the C register contains 1FH (the ASCII for clear screen), and if so, return to the caller without jumping to the video print routine, thus disabling the CLEAR key. The routine could be changed so that the clear code clears only the current text area.

Program Listing 1. *Scroll protection program*

```
                    00100 ;         SCROLL PROTECTION PROGRAM
                    00110 ;         MODEL I
                    00120 ;         DAN SCALES, 1981
                    00130 ;
4020                00140 CURLOC    EQU     4020H
0A7F                00150 CVINT     EQU     0A7FH    ;ROUTINE TO GET INTEGER ARGUMENT
```

Program continued

```
                                      ;OF USR CALL
3CØØ            VIDMEM  EQU   3CØØH   ;LOCATION OF VIDEO MEMORY
Ø458            VIDPR   EQU   458H    ;ROM ROUTINE TO PRINT CHARACTER
                                      ;ON SCREEN
                                      ;
4Ø1E                    ORG   4Ø1EH   ;LOCATION OF VIDEO DRIVER ADDRESS
4Ø1E ØØBF               DEFW  ØBFØØH  ;CHANGE DRIVER ADDRESS TO ADDRESS
                                      ;OF SCROLL ROUTINE
BFØØ                    ORG   ØBFØØH
BFØØ C5         PROT    PUSH  BC      ;C REGISTER CONTAINS CHARACTER
                                      ;TO BE PRINTED
BFØ1 D5                 PUSH  DE      ;DE CONTAINS ADDRESS OF VIDEO
                                      ;DEVICE CONTROL BLOCK
BFØ2 F5                 PUSH  AF      ;SAVE STATUS FLAGS
BFØ3 2A2Ø4Ø             LD    HL,(CURLOC)
BFØ6 7C                 LD    A,H
BFØ7 FE3F       ENDTXT  CP    3FH     ;CHECK IF NEAR END OF TEXT AREA
BFØ9 2Ø35               JR    NZ,POPALL       ;NO NEED TO SCROLL
BFØB 7D                 LD    A,L     ;CHECK IF CURSOR IS AT LAST
BFØC FEFF       LSTCHR  CP    ØFFH    ;POSITION OF TEXT AREA.
BFØE 28ØC               JR    Z,SCROLL ;IF SO, SCROLL.
BF1Ø 79                 LD    A,C
BF11 FEØD               CP    ØDH     ;IS LINEFEED TO BE PRINTED?
BF13 2Ø2B               JR    NZ,POPALL ;NO, DON'T SCROLL
BF15 7D                 LD    A,L
BF16 E6CØ               AND   ØCØH
BF18 FECØ       LSTLIN  CP    ØCØH
BF1A 2Ø24               JR    NZ,POPALL ;NOT ON LAST LINE, SO NO SCROLL
BF1C 214Ø3D     SCROLL  LD    HL,3D4ØH
BF1F 11ØØ3D             LD    DE,3DØØH
BF22 Ø1CØØ2             LD    BC,2CØH
BF25 EDBØ               LDIR          ;SCROLL THE SCREEN
BF27 162Ø               LD    D,2ØH
BF29 21CØ3F     LSTLN2  LD    HL,3FCØH
BF2C Ø14ØØØ             LD    BC,4ØH
BF2F 72         FILL    LD    (HL),D  ;FILL IN LAST LINE OF TEXT AREA
BF3Ø 23                 INC   HL      ;WITH BLANKS
BF31 ØB                 DEC   BC
BF32 78                 LD    A,B
BF33 B1                 OR    C
BF34 2ØF9               JR    NZ,FILL
BF36 2A2Ø4Ø             LD    HL,(CURLOC) ;SUBTRACT 4ØH FROM CURSOR
BF39 11CØFF             LD    DE,ØFFCØH   ;LOCATION TO ADJUST FOR SCROLL
BF3C 19                 ADD   HL,DE
BF3D 222Ø4Ø             LD    (CURLOC),HL
BF4Ø F1         POPALL  POP   AF
BF41 D1                 POP   DE
BF42 C1                 POP   BC
BF43 C358Ø4             JP    VIDPR   ;CONTINUE VIDEO PRINT ROUTINE
BF46 CD7FØA     SETUP   CALL  CVINT   ;GET USR ARGUMENT
BF49 4D                 LD    C,L
BF4A 7C                 LD    A,H     ;GET ADDRESS OF FIRST POSITION
BF4B E61F               AND   1FH     ;MAKE REGISTER A BETWEEN Ø AND 15
BF4D CD7BBF             CALL  M64     ;OF FIRST LINE OF TEXT AREA
BF5Ø 222ØBF             LD    (SCROLL+4),HL
BF53 E5                 PUSH  HL      ;SAVE TOP OF SCROLL AREA FOR LATER
BF54 114ØØØ             LD    DE,4ØH  ;TO MOVE TEXT UP ONE LINE (4ØH
BF57 19                 ADD   HL,DE   ;CHARACTERS
BF58 221DBF             LD    (SCROLL+1),HL
BF5B 79                 LD    A,C
BF5C E61F               AND   1FH     ;MAKE A REGISTER BETWEEN Ø AND 15
```

```
BF5E 3C         ØØ765          INC    A
BF5F CD7BBF     ØØ77Ø          CALL   M64      ;GET ADDRESS OF LAST POSITION
BF62 2B         ØØ78Ø          DEC    HL       ;OF LAST LINE OF TEXT AREA
BF63 7C         ØØ79Ø          LD     A,H
BF64 32Ø8BF     ØØ8ØØ          LD     (ENDTXT+1),A
BF67 7D         ØØ81Ø          LD     A,L
BF68 32ØDBF     ØØ82Ø          LD     (LSTCHR+1),A
BF6B E6CØ       ØØ83Ø          AND    ØCØH
BF6D 3219BF     ØØ84Ø          LD     (LSTLIN+1),A
BF7Ø 6F         ØØ85Ø          LD     L,A
BF71 222ABF     ØØ86Ø          LD     (LSTLN2+1),HL
BF74 D1         ØØ87Ø          POP    DE       ;RESTORE ADDR OF TOP OF SCROLL AREA
BF75 ED52       ØØ88Ø          SBC    HL,DE    ;HL EQUALS THE NUMBER OF CHARACTERS
BF77 2223BF     ØØ89Ø          LD     (SCROLL+7),HL    ;TO BE SCROLLED
BF7A C9         ØØ9ØØ          RET
                ØØ91Ø          ;
                ØØ92Ø          ;M64 IS A ROUTINE TO GET THE ADDRESS OF THE
                ØØ93Ø          ;FIRST POSITION OF THE LINE ON THE SCREEN WHOSE
                ØØ94Ø          ;NUMBER IS IN THE A REGISTER.
BF7B 47         ØØ95Ø M64      LD     B,A
BF7C 21ØØØØ     ØØ96Ø          LD     HL,ØH
BF7F 114ØØØ     ØØ97Ø          LD     DE,4ØH
BF82 B7         ØØ98Ø          OR     A
BF83 28Ø3       ØØ99Ø          JR     Z,CONT
BF85 19         Ø1ØØØ LOOP     ADD    HL,DE
BF86 1ØFD       Ø1Ø1Ø          DJNZ   LOOP
BF88 11ØØ3C     Ø1Ø2Ø CONT     LD     DE,VIDMEM ;HL NOW EQUALS 64 TIMES REGISTER A
BF8B 19         Ø1Ø3Ø          ADD    HL,DE
BF8C C9         Ø1Ø4Ø          RET
BFØØ            Ø1Ø5Ø          END    PROT
ØØØØØ TOTAL ERRORS
```

Program Listing 2. *BASIC program*

```
100 ' MODEL I SCROLL PROTECT
110 ' DAN SCALES, 1981
120 '    THE USR ADDRESS TO CHANGE SCROLL PROTECTION IS -1657Ø.
125 ' FOR INSTANCE, TO CHANGE THE TEXT AREA TO ONLY THE BOTTOM
126 ' FOUR LINES, DEFINE THE USR ADDRESS AS -1657Ø, AND THEN
127 ' EXECUTE "X = USR(12 * 256 + 15)".
13Ø '
14Ø FOR I=-1664Ø TO -1664Ø+141:READ X:POKE I,X:NEXT
15Ø POKE 16414,Ø:POKE 16415,191    'CHANGE SCREEN DRIVER ADDRESS
2ØØ DATA 197,213,245,42,32,64,124,254,63,32
21Ø DATA 53,125,254,255,4Ø,12,121,254,13,32
22Ø DATA 43,125,23Ø,192,254,192,32,36,33,64
23Ø DATA 61,17,Ø,61,1,192,2,237,176,22
24Ø DATA 32,33,192,63,1,64,Ø,114,35,11
25Ø DATA 12Ø,177,32,249,42,32,64,17,192,255
26Ø DATA 25,34,32,64,241,2Ø9,193,195,88,4
27Ø DATA 2Ø5,127,1Ø,77,124,23Ø,31,2Ø5,123,191
28Ø DATA 34,32,191,229,17,64,Ø,25,34,29
29Ø DATA 191,121,23Ø,31,6Ø,2Ø5,123,191,43,124
3ØØ DATA 5Ø,8,191,125,5Ø,13,191,23Ø,192,5Ø
31Ø DATA 25,191,111,34,42,191,2Ø9,237,82,34
32Ø DATA 35,191,2Ø1,71,33,Ø,Ø,17,64,Ø
33Ø DATA 183,4Ø,3,25,16,253,17,Ø,6Ø,25
34Ø DATA 2Ø1,Ø
```

14

Easier Formatting with SCREEN

by Don Robertson

System Requirements:
Model I or III
16K or 32K
Level II or Disk BASIC

The SCREEN program is for the BASIC programmer who wants professional-looking programs but gets frustrated with the time and energy required to format a screen using graphics and alphanumerics. The traditional way to lay out a screen is to PRINT AT, POKE, or PRINT USING. Traditionally, to use the graphics characters, you must look up their ASCII codes and laboriously POKE or use CHR$ to format them. SCREEN lets you move the cursor with complete freedom, using graphics to draw pictures, or putting alphanumeric characters where you want them. When the screen is set up exactly as you want it, the picture is POKEd into the program itself.

The program is easy to run (see Program Listing). First, the current screen appears. To create a new screen, type C. This clears the screen and displays the graphics cursor in the center. To modify the existing screen, type any character except C. Do not press ENTER after your choice, as this tells the program to save whatever is currently displayed on the screen.

The graphics cursor is controlled by the arrow keys. To draw, press the appropriate key. Press two keys simultaneously for diagonal lines. To erase or to move the cursor without drawing, hold down the SHIFT key and press the desired arrow. When SHIFT is pressed, the graphics cursor blinks. This is useful when the screen is almost full and you've forgotten where you are.

SCREEN lets you switch between the graphics and the alphanumeric modes, using the CLEAR key as the switch. When in the alpha mode, the blinking cursor shows the current position to be typed. Cursor position is controlled by the arrow keys. When the cursor is in the desired position, anything you type appears on the screen, much as it would on a word processor. Use the space bar to erase in the alpha mode.

When you're satisfied with the screen, press ENTER. The program scans the display and POKEs it into program lines 30000–30080. It takes about 20 seconds to dump the screen. When the READY prompt appears, delete lines 31010–33120 and write your BASIC program. Your screen is stored in A1$–A8$ and printed whenever line 31000 is accessed. You *must* type lines 30000–30070 exactly as they appear in the listing. Each of the strings A1$–A8$ must be precisely 128 characters long; these variables are where your new screen is stored.

Lines 31010–31030 find the beginning of A1$, check whether it is a modification or an entirely new screen, and position the graphics cursor in the middle of the display. The program starts in the graphics mode. Lines 31040–31110 check for cursor movement (arrows), alpha switch (CLEAR), and save screen (ENTER). Cursor movement in both modes is smooth and fast, because the program does not use INKEY$ to detect it. Keyboard memory is located in locations 3801H–3880H, with all control characters at 3840H. By PEEKing at 14400 (3840H) and testing for the different controls, the program can handle fast typing and support repeats—even on the Model I.

Alpha characters are handled in essentially the same way as graphics. Lines 33000–33090 detect controls, set the POKEing position, and display any characters you type. The cursor in the alpha mode is non-destructive; the character under the cursor is stored in CH. The ASCII code of any character typed is saved into KB and then POKEd to the screen. Lines 31120–31200 save the screen into variables A1$–A8$ when ENTER is pressed.

Program Listing

```
1 CLEAR1000
30000 A1$="SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCRE
EN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREE
N SCREEN SC"
30010 A2$="REEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN
 SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN
SCREEN SCRE"
30020 A3$="EN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN S
CREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SC
REEN SCREEN"
30030 A4$=" SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCR
EEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCRE
EN SCREEN S"
30040 A5$="CREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREE
N SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN
 SCREEN SCR"
```

Program continued

```
30050 A6$="EEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN
SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN S
CREEN SCREE"
30060 A7$="N SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SC
REEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCR
EEN SCREEN "
30070 A8$="SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCRE
EN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN SCREEN BY DO
N ROBERTSON"
31000 CLS:PRINTA1$;A2$;A3$;A4$;A5$;A6$;A7$;LEFT$(A8$,127);:POKE1
6383,ASC(RIGHT$(A8$,1))
31010 P=VARPTR(A1$)
'
   FIND LOCATION OF A1$
31020 A=PEEK(P+2)*256+PEEK(P+1)
31025 K$=INKEY$:IFK$=""THEN31025ELSEIFK$="C"THENCLS'    C --> CL
EARS SCREEN --> DO ENTIRELY NEW SCREEN  <<-->> ANY OTHER LETTER
LEAVES SCREEN FOR MODIFICATION ONLY
31030 X=63:Y=23:PO=15360'    INITIAL POSITIONS OF CURSORS
31040 M=PEEK(14400):FORI=1TO10:NEXT'    CHECK FOR ARROWS, (MOVE
CURSOR), ENTER (PUNCH SCREEN TO A1$-->A8$), CLEAR (SWITCH IN LET
TER POKER)
31050 IFM=1THEN31120'    ENTER KEYED --> SAVE SCREEN
31055 IFM=2THENFORI=1TO40:NEXTI:GOTO32000'    CLEAR KEYED --> SW
ITCH TO ALPHABETICS
31060 IFMAND8ANDY>0THENY=Y-1'    ARROW KEYED --> MOVE CURSOR
31070 IFMAND16ANDY<47THENY=Y+1
31080 IFMAND32ANDX>0THENX=X-1
31090 IFMAND64ANDX<126THENX=X+1
31100 SET(X,Y):IFPEEK(14464)THENRESET(X,Y)'    SHIFT IS ERASE
31110 GOTO31040
31120 '
 SAVE SCREEN INTO PROGRAM
31140 FOR II=15360TO16320STEP128
31150  FORJJ=0TO127
31160         POKEA+XX*139+JJ,PEEK(II+JJ)
31170  NEXTJJ
31180 XX=XX+1
31190 NEXTII
31200 END
32000 ' ALPHA NUMERIC CHARACTERS POKED HERE
32010 M=PEEK(14400)'    LOOK FOR CONTROL CHARACTER
32015 CH=PEEK(PO):POKEPO,254:P1=PO:POKEP1,CH'    FLASHING CURSOR
32020 IFM>2THEN33000'    NOT CLEAR, NOT ENTER --> CURSOR MOVE
32030 IFM=2THENFORI=1TO40:NEXTI:GOTO31040'    CLEAR  --> SWITCH
TO GRAPHICS
32040 IFM=1THEN31120'    ENTER  --> SAVE INTO PROGRAM
32050 K$=INKEY$:IFK$<>""ANDK$<>"["THENKB=ASC(K$):K$=""'    CHECK
 FOR LETTER
32060 IFKB>31THENPOKEPO,KB:IFPO<16383THENPO=PO+1'    CHECK FOR N
ON-CONTROL CHARACTER IN BUFFER, STORE ON SCREEN, MOVE CURSOR 1 P
OSITION RIGHT
32070 KB=0'    RESET BUFFER
32080 GOTO32010'    GET NEXT CHARACTER
33000 'GET POKING POSITION
33040 IFMAND8ANDPO>15424THENPO=PO-64'    CHECK FOR ARROW  --> MO
VE CURSOR
33050 IFMAND16ANDPO<16320THENPO=PO+64
33060 IFMAND32ANDPO>15360THENPO=PO-1
33070 IFMAND64ANDPO<16383THENPO=PO+1
33090 GOTO32010
```

15

DRAFTER: A Graphics Editor

by R.K. Fink

System Requirements:
Model I or III
32K or 48K
One disk drive

After years of searching for the ideal screen writer, I designed DRAFTER to be both easy to use, and foolproof. It operates quickly, uses few commands, and requires no computations to find TRS-80 block graphic codes. It automatically saves the screen as BASIC lines of data statements, in a file you can merge with another BASIC program. It uses machine-language block moves to speed up operation of screen saves and recalls. Two versions of DRAFTER are provided here—one for 32K systems, and one for 48K systems. Both require one disk drive.

DRAFTER (see Program Listing 1) was developed for a 48K disk system. Program Listing 2 is for a 32K system. The machine-language USR subroutines, which are used to clear buffers, save or recall a screen, or construct a data buffer, are POKEd into high memory from strings. They become invisible when a new data file is merged and POKEd to the screen. The memory maps in Figure 2 show where space is needed for storage buffers and for the operation codes. When you rerun the program, it loads its routines in memory, sets buffers to zero, and starts into the D (draw) mode with the (*) cursor ready for screen construction.

Using DRAFTER

The commands and construction loops are shown in Figure 1. While you are in the draw mode, hitting the SHIFT key saves the screen and displays the main menu. If the data on the screen is to be saved as a group of BASIC DATA statement lines, press S. To append and load a disk file of data previously made, press L. After loading the storage buffer and screen you are returned to the draw mode, ready for further construction. X exits to BASIC, but gives you a second chance with a prompt if your data has not been saved to disk.

While in the draw mode, ASCII keyboard characters may be entered at the cursor by pressing CLEAR while typing the desired character. Regular graphic blocks are built by pressing the keys shown on the numeric keypad. If you don't have a numeric keypad, use the numbers along the top key row—with a loss, however, of the relative position, or touch feel, of building a rectangular block. When a block is begun you are in a loop that can be erased and restarted with the space bar. When you're satisfied, press the ENTER key to enter the graphics equivalent. The cursor reappears.

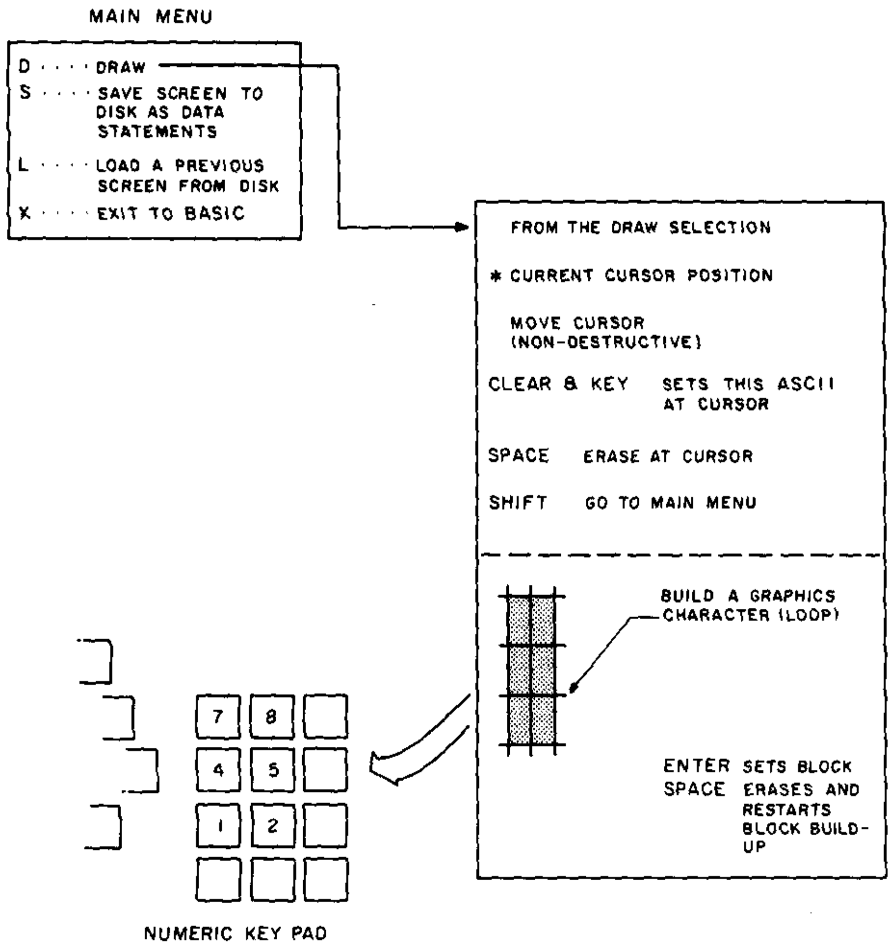

MAIN MENU
D · · · · DRAW
S · · · · SAVE SCREEN TO DISK AS DATA STATEMENTS
L · · · · LOAD A PREVIOUS SCREEN FROM DISK
X · · · · EXIT TO BASIC
FROM THE DRAW SELECTION
* CURRENT CURSOR POSITION
MOVE CURSOR (NON-DESTRUCTIVE)
CLEAR & KEY SETS THIS ASCII AT CURSOR
SPACE ERASE AT CURSOR
SHIFT GO TO MAIN MENU
BUILD A GRAPHICS CHARACTER (LOOP)
ENTER SETS BLOCK
SPACE ERASES AND RESTARTS BLOCK BUILD-UP
7
8
4
5
1
2
NUMERIC KEY PAD

Figure 1. *Commands*

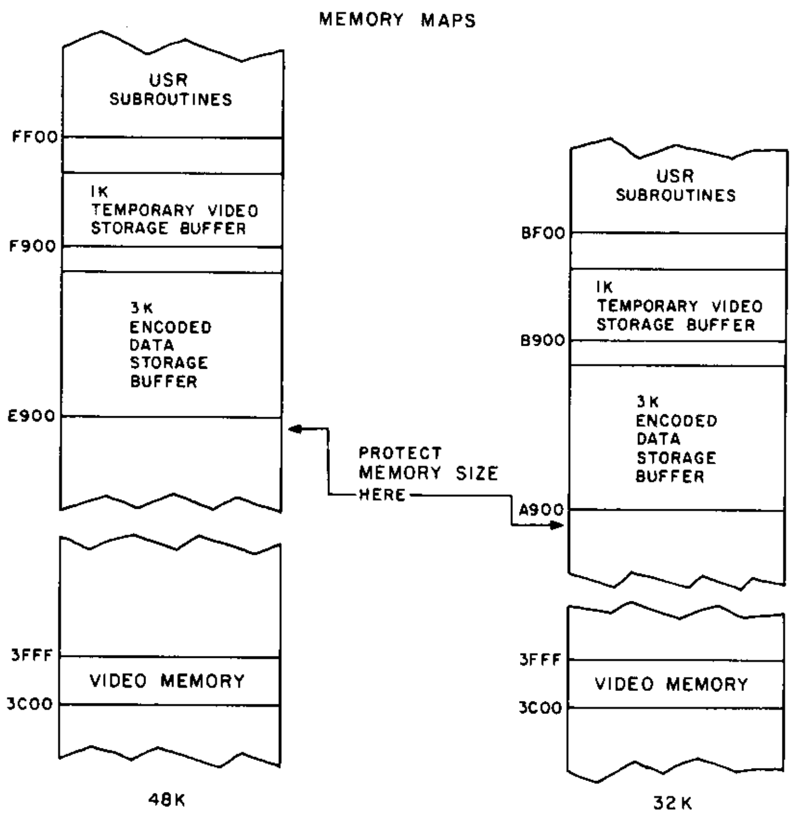


Figure 2. *Memory Maps*

The cursor has automatic repeat movement. The non-destructive cursor stops at any boundary position. To erase a position, hit the space bar, and the character at the cursor is erased. You can make a data file of your screen with the S command, then use MERGE"filespec" to append it to any other program. These lines always start at 20000, so put nothing above this in your other program. Only active screen locations with character contents other than a space are saved. It's assumed that the screen is cleared with a CLS command before POKEing the data in later programs. To use the created DATA statements note that there are three numbers per location: MSB, LSB of screen address (not in reversed order like Z80 operation code), and the screen character's ASCII code.

A simple program loop to fill the screen follows:

```
CLS
FOR N=1 TO (Length of the active data locations)
READ M,L,D : AD=M*16+L
POKE AD,D
NEXT N
```

Figure continued

NAME	DESCRIPTION
A$ AA$	Temporary Data Input
AD	POKE Address
CC	Cursor Character
DA	Current Character Buildup
F$	Filespec for Disk
I N	Closed Loop Counters
KD	Keyboard Input of NUMBERS Row
KB	Keyboard Input of ARROWS Row
LN$	ASCII DATA Statements Created
PS	Current Cursor Position
PO	Temporary POKE Address
S	Start of Screen
SA	Start Address of Op Codes
SL	Line No. of BASIC DATA Lines
U0$-U3$	Strings for USR Subroutine Codes

SUBROUTINES USED

LINE #	DESCRIPTION
1500	USR Subroutines
2000	Save DATA Statements To Disk
3500	Load Data From Disk

Figure 3. *Major symbols and subroutines*

DRAFTER also lets you construct and save a loadable file of a sketch in just a few minutes. To start, cover a video worksheet with celluloid and use a grease pencil to get rough starting proportions. Tó demonstrate the technique, I constructed a Giant Android in about eight minutes and saved it as Program Listing 3. DRAFTER put all those data lines automatically on disk. If you want a starting point, put the data lines into an ASCII file from the keyboard and then load them into DRAFTER.

Program Listing 1

```
1 ' ****  DRAFTING BOARD AND DATA ENCODER  ****
2 '   **   DRAFTER/BAS  MOD I DISK 32 OR 48K **
3 '   **    RK FINK  1/20/82   VER 1.0     - **
4 '   **     FOR THIS 48K VERSION ....       **
5 '   **      PROTECT 50000 TO ENTER BASIC   **
6 '   ** :::::::::::::::::::::::::::::::::: **
7 '
8 '
9 CLS:PRINTCHR$(23);"L O W
   R E S
     G R A P H I C S
       T A B L E T":FORN=1TO3000:NEXT
10 CLEAR500:CLS
15 DEFINTA-Z:GOSUB1500:GOSUB1700
```

```
20 ONERRORGOTO0:S=15360
30 X=0:Y=0:CC=42:DA=128
40 PS=S+X+64*Y:TP=PEEK(PS)
49 '  ::::: The Main program Loop :::::
50 POKEPS,CC:GOSUB200:GOSUB270
60 IFPEEK(14464)=0THEN50
68 '  :::::          :::          :::::
69 '  :::::  The Main Menu Routine   ::::
70 GOSUB1000
80 CLS:PRINT
90 PRINTTAB(8)"  DRAFTING BOARD AND DATA STATEMENT ENCODER"
100 PRINT:PRINTTAB(25)"M E N U"
110 PRINT"          D .... START OR CONTINUE DRAWING
          S .... SAVE A SCREEN & PUT DATA ON DISK
          L .... LOAD A PREVIOUS SCREEN FROM DISK
          X .... EXIT TO BASIC"
120 PRINT:PRINT"SELECTION......??"
130 AA$=INKEY$:IFAA$=""THEN130
140 IFAA$="X"THENCLS:PRINT"SURE DATA'S SAVED??    REALLY EXIT (Y
/N)":INPUTAA$:IFAA$="N"THEN90ELSEEND
150 IFAA$="S"THEN2000
160 IFAA$="D"THENCLS:Z=USR2(0):GOTO30
170 IFAA$="L"THENCLS:GOTO3500
180 GOTO130
200 KB=PEEK(14400):IFKB=0THENRETURN
201 IFKB<>2THEN210
202 IFPEEK(14400)=2THEN202
203 POKEPS,128:DA$=INKEY$:IFDA$=""THEN203
204 DA=ASC(DA$):POKEPS,DA:TP=DA:FORN=1TO200:NEXT:RETURN
210 IF(KB=32)AND(X<>0)THENX=X-1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS
):POKEPS,CC:RETURN
220 IF(KB=64)AND(X<>63)THENX=X+1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(P
S):POKEPS,CC:RETURN
230 IF(KB=8)AND(Y>0)THENY=Y-1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS):
POKEPS,CC:RETURN
240 IF(KB=16)AND(Y<15)THENY=Y+1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS
):POKEPS,CC:RETURN
250 IF(KB=128)THENPOKEPS,128:FORN=1TO200:NEXT:TP=128:RETURN
260 ' .... Construct A Graphics Block Loop Follows ....
270 KD=PEEK(14352):IFKD=0THENRETURN
280 IFPEEK(14400)=128THENPOKEPS,128:DA=128
290 IFPEEK(14400)=1THENPOKEPS,DA:TP=DA:DA=128:RETURN
300 IFKD=128THENDA=DAOR1ELSEIFPEEK(14368)=1THENDA=DAOR2
310 IFKD=16THENDA=DAOR4ELSEIFKD=32THENDA=DAOR8
320 IFKD=2THENDA=DAOR16ELSEIFKD=4THENDA=DAOR32
330 POKEPS,DA:KD=PEEK(14352):TP=DA:GOTO280
1000 POKEPS,TP
1010 Z=USR1(0):RETURN
1498 ' ... These are the string packers that set up
1499 '      the USR subroutines ....
1500 PRINT"INITIALIZING PROGRAM ROUTINES ........."
1505 U0$="1101E92100E9AF770100014EDB0C9"
1510 PO=&HFF00:FORI=1TOLEN(U0$)STEP2:L=ASC(MID$(U0$,I))-48
1520 R=ASC(MID$(U0$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
1530 POKEPO,16*L+R:PO=PO+1:NEXTI
1540 DEFUSR0=&HFF00
1541 '  USR0 ... Clears buffers with zeros
1550 U1$="1100F921003C010004EDB0C9"
1560 PO=&HFF10:FORI=1TOLEN(U1$)STEP2:L=ASC(MID$(U1$,I))-48
1570 R=ASC(MID$(U1$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
1580 POKEPO,16*L+R:PO=PO+1:NEXTI
```

Program continued

```
1590 DEFUSR1=&HFF10
1591 '  USR1 ... Saves screen to video buffer ...
1600 U2$="11003C2100F9010004EDB0C9"
1610 PO=&HFF20:FORI=1TOLEN(U2$)STEP2:L=ASC(MID$(U2$,I))-48
1620 R=ASC(MID$(U2$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
1630 POKEPO,16*L+R:PO=PO+1:NEXTI
1640 DEFUSR2=&HFF20
1641 '  USR2 ... Moves saved buffer back to screen ...
1650 U3$="0100FD11003C2100F9DD2100E9FD2100007EFE802819FE202815DD
7200DD23DD7300DD23DD7700DD23FD23FD23FD232313E5ED427CB52803E118D6
E1FDE5E1C39A0A"
1660 PO=&HFF30:FORI=1TOLEN(U3$)STEP2:L=ASC(MID$(U3$,I))-48
1670 R=ASC(MID$(U3$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
1680 POKEPO,16*L+R:PO=PO+1:NEXTI
1690 DEFUSR3=&HFF30:RETURN
1691 '  USR3 ... Construct Data Buffer from video buffer
1700 Z=USR0(0):CLS:RETURN:   'Clear Buffers TO ZEROS ...
1999 ' ... Construct Data Statements & Save To Disk ...
2000 CLS
2010 PRINT
2020 LE=USR3(0):IFPEEK(&HE900)=0THENPRINT"NO VIDEO TEXT IN BUFFE
R":FORN=1TO500:NEXT:CLS:GOTO30
2030 PRINT:LINE INPUT"FILESPEC FOR DATA SAVE? ";F$:OPEN"O",1,F$
2040 PRINT:PRINT"SAVING DATA LINES ....."
2050 SA=&HE900:SL=20000:GOSUB4000
2060 LN$=LN$+"DATA"
2070 FORN=SATOSA+LE-1
2075 IFLEN(LN$)<60THEN2110
2080 LN$=LEFT$(LN$,LEN(LN$)-1)
2090 GOSUB5000:GOSUB4000
2100 LN$=LN$+"DATA"
2110 A$=STR$(PEEK(N)):A$=RIGHT$(A$,LEN(A$)-1)
2120 LN$=LN$+A$+","
2140 NEXTN
2150 LN$=LEFT$(LN$,LEN(LN$)-1)
2160 GOSUB5000
2170 CLOSE
2180 GOTO90
3499 ' ... Load in a Diskfile of Data Statements ...
3500 CLS:LINE INPUT"FILENAME OF DATA? ";F$:PRINT
3510 PRINT"ENTER    RUN 3600 AT THE 'READY' PROMPT"
3520 PRINT:PRINT
3530 MERGEF$:END
3600 PRINT"RESETTING PROGRAM SUBROUTINES .....":GOSUB1505:GOSUB1
700
3610 ONERRORGOTO3650
3620 FORN=1TO3072:READM,L,D
3630 AD=M*256+L:POKEAD,D
3640 NEXT
3649 ' ... When all data is exhausted, jump below on error
3650 Z=USR1(0):RESUME20
4000 LN$=RIGHT$(STR$(SL),LEN(STR$(SL))-1)+" ":RETURN
5000 PRINT#1,LN$ : SL=SL+10 : RETURN
20000 DATA61,71,191,61,151,131,61,152,191,61,153,131,61,157
20010 DATA131,61,158,191,61,159,131,61,216,191,61,217,176,61
20020 DATA218,176,61,219,191,61,220,176,61,221,176,61,222,191
20030 DATA62,25,140,62,26,140,62,27,140,62,28,140,62,29,140
20040 DATA62,30,140,62,88,131,62,135,191
```

Program Listing 2. *Changes to provide a 32K system program*

```
1 ' ****  DRAFTING BOARD AND DATA ENCODER  ****
2 '   **   DRAFTER/BAS  MOD I DISK 32 OR 48K **
3 '   **    RK FINK  1/2Ø/82   VER 1.Ø       **
4 '   **     FOR THIS 32K VERSION ....       **
5 '   **      PROTECT 43ØØØ TO ENTER BASIC   **
6 '   ** :::::::::::::::::::::::::::::::::: **
7 '
8 '
9 CLS:PRINTCHR$(23);"L O W
   R E S
     G R A P H I C S
       T A B L E T":FORN=1TO3ØØØ:NEXT
1Ø CLEAR5ØØ:CLS
15 DEFINTA-Z:GOSUB15ØØ:GOSUB17ØØ
2Ø ONERRORGOTOØ:S=1536Ø
3Ø X=Ø:Y=Ø:CC=42:DA=128
4Ø PS=S+X+64*Y:TP=PEEK(PS)
49 '  ::::: The Main program Loop :::::
5Ø POKEPS,CC:GOSUB2ØØ:GOSUB27Ø
6Ø IFPEEK(14464)=ØTHEN5Ø
68 '  :::::          :::          :::::
69 '  :::::  The Main Menu Routine   ::::
7Ø GOSUB1ØØØ
8Ø CLS:PRINT
9Ø PRINTTAB(8)"  DRAFTING BOARD AND DATA STATEMENT ENCODER"
1ØØ PRINT:PRINTTAB(25)"M E N U"
11Ø PRINT"         D .... START OR CONTINUE DRAWING
         S .... SAVE A SCREEN & PUT DATA ON DISK
         L .... LOAD A PREVIOUS SCREEN FROM DISK
         X .... EXIT TO BASIC"
12Ø PRINT:PRINT"SELECTION......??"
13Ø AA$=INKEY$:IFAA$=""THEN13Ø
14Ø IFAA$="X"THENCLS:PRINT"SURE DATA'S SAVED??    REALLY EXIT (Y
/N)":INPUTAA$:IFAA$="N"THEN9ØELSEEND
15Ø IFAA$="S"THEN2ØØØ
16Ø IFAA$="D"THENCLS:Z=USR2(Ø):GOTO3Ø
17Ø IFAA$="L"THENCLS:GOTO35ØØ
18Ø GOTO13Ø
2ØØ KB=PEEK(144ØØ):IFKB=ØTHENRETURN
2Ø1 IFKB<>2THEN21Ø
2Ø2 IFPEEK(144ØØ)=2THEN2Ø2
2Ø3 POKEPS,128:DA$=INKEY$:IFDA$=""THEN2Ø3
2Ø4 DA=ASC(DA$):POKEPS,DA:TP=DA:FORN=1TO2ØØ:NEXT:RETURN
21Ø IF(KB=32)AND(X<>Ø)THENX=X-1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS
):POKEPS,CC:RETURN
22Ø IF(KB=64)AND(X<>63)THENX=X+1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(P
S):POKEPS,CC:RETURN
23Ø IF(KB=8)AND(Y>Ø)THENY=Y-1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS):
POKEPS,CC:RETURN
24Ø IF(KB=16)AND(Y<15)THENY=Y+1:POKEPS,TP:PS=S+X+64*Y:TP=PEEK(PS
):POKEPS,CC:RETURN
25Ø IF(KB=128)THENPOKEPS,128:FORN=1TO2ØØ:NEXT:TP=128:RETURN
26Ø ' .... Construct A Graphics Block Loop Follows ....
27Ø KD=PEEK(14352):IFKD=ØTHENRETURN
28Ø IFPEEK(144ØØ)=128THENPOKEPS,128:DA=128
29Ø IFPEEK(144ØØ)=1THENPOKEPS,DA:TP=DA:DA=128:RETURN
3ØØ IFKD=128THENDA=DAOR1ELSEIFPEEK(14368)=1THENDA=DAOR2
31Ø IFKD=16THENDA=DAOR4ELSEIFKD=32THENDA=DAOR8
32Ø IFKD=2THENDA=DAOR16ELSEIFKD=4THENDA=DAOR32
```

Program continued

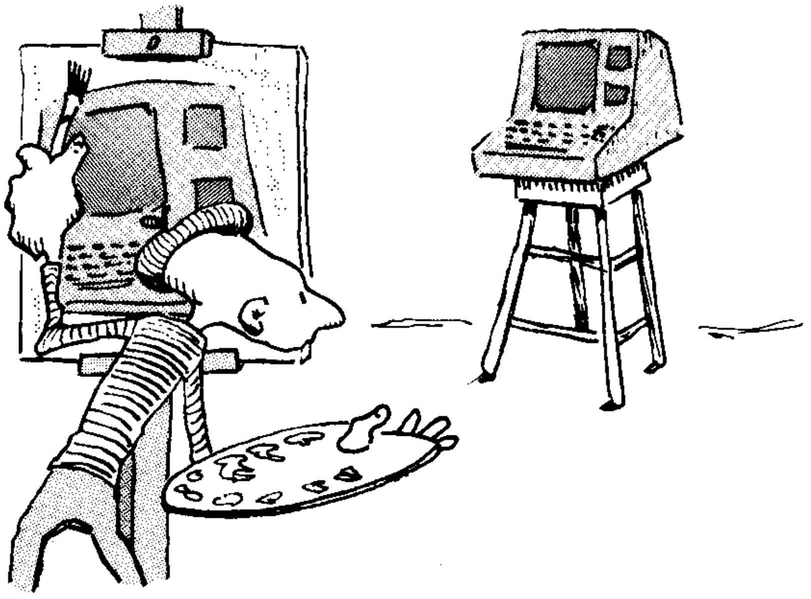
```
330 POKEPS,DA:KD=PEEK(14352):TP=DA:GOTO28Ø
1ØØØ POKEPS,TP
1Ø1Ø Z=USR1(Ø):RETURN
1498 ' ... These are the string packers that set up
1499 '      the USR subroutines ....
15ØØ PRINT"INITIALIZING PROGRAM ROUTINES ........."
15Ø5 UØ$="11Ø1A921ØØA9AF77Ø1ØØ14EDBØC9"
151Ø PO=&HBFØØ:FORI=1TOLEN(UØ$)STEP2:L=ASC(MID$(UØ$,I))-48
152Ø R=ASC(MID$(UØ$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
153Ø POKEPO,16*L+R:PO=PO+1:NEXTI
154Ø DEFUSRØ=&HBFØØ
1541 '  USRØ ... Clears buffers with zeros
155Ø U1$="11ØØB921ØØ3CØ1ØØØ4EDBØC9"
156Ø PO=&HBF1Ø:FORI=1TOLEN(U1$)STEP2:L=ASC(MID$(U1$,I))-48
157Ø R=ASC(MID$(U1$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
158Ø POKEPO,16*L+R:PO=PO+1:NEXTI
159Ø DEFUSR1=&HBF1Ø
1591 '  USR1 ... Saves screen to video buffer ...
16ØØ U2$="11ØØ3C21ØØB9Ø1ØØØ4EDBØC9"
161Ø PO=&HBF2Ø:FORI=1TOLEN(U2$)STEP2:L=ASC(MID$(U2$,I))-48
162Ø R=ASC(MID$(U2$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
163Ø POKEPO,16*L+R:PO=PO+1:NEXTI
164Ø DEFUSR2=&HBF2Ø
1641 '  USR2 ... Moves saved buffer back to screen ...
165Ø U3$="Ø1ØØBD11ØØ3C21ØØB9DD21ØØA9FD21ØØØØ7EFE8Ø2819FE2Ø2815DD
72ØØDD23DD73ØØDD23DD77ØØDD23FD23FD23FD232313E5ED427CB528Ø3E118D6
E1FDE5E1C39AØA"
166Ø PO=&HBF3Ø:FORI=1TOLEN(U3$)STEP2:L=ASC(MID$(U3$,I))-48
167Ø R=ASC(MID$(U3$,I+1))-48:L=L+7*(L>9):R=R+7*(R>9)
168Ø POKEPO,16*L+R:PO=PO+1:NEXTI
169Ø DEFUSR3=&HBF3Ø:RETURN
1691 '  USR3 ... Construct Data Buffer from video buffer
17ØØ Z=USRØ(Ø):CLS:RETURN:   'Clear Buffers TO ZEROS ...
1999 ' ... Construct Data Statements & Save To Disk ...
2ØØØ CLS
2Ø1Ø PRINT
2Ø2Ø LE=USR3(Ø):IFPEEK(&HA9ØØ)=ØTHENPRINT"NO VIDEO TEXT IN BUFFE
R":FORN=1TO5ØØ:NEXT:CLS:GOTO3Ø
2Ø3Ø PRINT:LINEINPUT"FILESPEC FOR DATA SAVE? ";F$:OPEN"O",1,F$
2Ø4Ø PRINT:PRINT"SAVING DATA LINES ....."
2Ø5Ø SA=&HA9ØØ:SL=2ØØØØ:GOSUB4ØØØ
2Ø6Ø LN$=LN$+"DATA"
2Ø7Ø FORN=SATOSA+LE-1
2Ø75 IFLEN(LN$)<6ØTHEN211Ø
2Ø8Ø LN$=LEFT$(LN$,LEN(LN$)-1)
2Ø9Ø GOSUB5ØØØ:GOSUB4ØØØ
21ØØ LN$=LN$+"DATA"
211Ø A$=STR$(PEEK(N)):A$=RIGHT$(A$,LEN(A$)-1)
212Ø LN$=LN$+A$+","
214Ø NEXTN
215Ø LN$=LEFT$(LN$,LEN(LN$)-1)
216Ø GOSUB5ØØØ
217Ø CLOSE
218Ø GOTO9Ø
3499 ' ... Load in a Diskfile of Data Statements ...
35ØØ CLS:LINEINPUT"FILENAME OF DATA? ";F$:PRINT
351Ø PRINT"ENTER    RUN 36ØØ AT THE 'READY' PROMPT"
352Ø PRINT:PRINT
353Ø MERGEF$:END
36ØØ PRINT"RESETTING PROGRAM SUBROUTINES .....":GOSUB15Ø5:GOSUB1
7ØØ
```

```
3610 ONERRORGOTO3650
3620 FORN=1TO3072:READM,L,D
3630 AD=M*256+L:POKEAD,D
3640 NEXT
3649 ' ... When all data is exhausted, jump below on error
3650 Z=USR1(0):RESUME20
4000 LN$=RIGHT$(STR$(SL),LEN(STR$(SL))-1)+" ":RETURN
5000 PRINT#1,LN$:SL=SL+10:RETURN
```

Program Listing 3.

A Giant Android. These BASIC lines were constructed automatically by DRAFTER after about 10 minutes of screen construction time.

```
20000 DATA60,17,71,60,19,73,60,21,65,60,23,78,60,25,84,60,28
20010 DATA65,60,30,78,60,32,68,60,34,82,60,36,79,60,38,73,60
20020 DATA40,68,60,90,188,60,94,188,60,154,191,60,158,191,60
20030 DATA216,191,60,217,191,60,218,191,60,219,191,60,220,191
20040 DATA60,221,191,60,222,191,60,223,191,60,224,191,61,23
20050 DATA170,61,24,191,61,26,170,61,27,191,61,28,191,61,29
20060 DATA191,61,30,149,61,32,191,61,33,149,61,87,170,61,88
20070 DATA191,61,89,149,61,95,170,61,96,191,61,97,149,61,151
20080 DATA130,61,152,131,61,153,131,61,154,143,61,155,143,61
20090 DATA156,143,61,157,143,61,158,143,61,159,131,61,160,131
20100 DATA61,161,129,61,213,160,61,214,176,61,215,176,61,216
20110 DATA176,61,217,176,61,218,150,61,219,131,61,220,131,61
20120 DATA221,131,61,222,131,61,223,180,61,224,176,61,225,176
20130 DATA61,226,176,62,21,191,62,25,191,62,32,191,62,35,191
20140 DATA62,84,190,62,89,191,62,96,191,62,100,189,62,147,186
20150 DATA62,152,170,62,153,191,62,160,191,62,161,149,62,165
20160 DATA181,62,211,142,62,216,170,62,217,191,62,224,191,62
20170 DATA225,149,62,229,141,63,25,191,63,32,191,63;88,191,63
20180 DATA97,191,63,152,191,63,161,191,63,214,188,63,215,188
20190 DATA63,216,191,63,225,191,63,226,188,63,227,188
```



16

HELP with Commands

by Phil Comeau

System Requirements:
Model I or III
16K RAM
One disk drive
Editor/assembler
TRSDOS-compatible DOS

Sometimes a short reminder can be more valuable than a three-page description in a user's manual. TRSDOS's LIB function displays a list of the available commands on the video screen, but many of them require extra parameters, and remembering the formats and possibilities of each command is not an easy task. The HELP program (Program Listing 1) does it for you. Type in the name of the command or feature you're unsure of, and HELP tells you about it. For example, if you type HELP DEBUG, HELP responds:

```
DEBUG (ON/OFF)
  ACTIVATES OR DEACTIVATES DEBUGGING MONITOR
FOR MORE INFORMATION TYPE "HELP DEBUG COMMANDS"
```

HELP can also tell you about the input formats expected by a general ledger program, or Scripsit formatting commands. It can also be used for purposes as diverse as an address and phone number list or a quick recipe reference.

How It Works

HELP begins by getting some information from the command line buffer, the place in memory where DOS stores the last command entered. You type HELP and a word, called the search keyword, describing the information you need. A list of available keywords and their associated informational messages is stored in the HELP program. HELP searches this list until it locates either the keyword or the end of the list. If it finds the keyword, the message following the keyword is displayed on the video screen. If it reaches end of the list without finding the keyword, HELP displays the message SORRY—NO HELP IS AVAILABLE FOR (the keyword). In either case, HELP then returns to DOS.

The HELP program, as written, provides information on DOS command formats. Both the keywords and the help messages are stored as strings. The last byte of each string is a special character called end-of-string, or EOS. EOS typically has the value zero.

There are two advantages to storing text in this manner. First, strings may have variable length. There is no arbitrary restriction that says the search keyword must be, for example, eight characters long. Second, because it uses a consistent format to store the text, some general purpose subroutines can be used. For instance, the subroutine DSPSTR, which is used to display a string on the video screen, can be lifted out of this program and used in another one. As long as DSPSTR is called correctly (with the HL register pair pointing to a string terminated by EOS), it will function properly.

In this program, the list of keywords and help messages is stored as strings in memory, starting at the symbol HLPLST and ending at the symbol ENDLST. Each entry in the list is formed by a keyword, followed immediately by the help message associated with that keyword.

I used EDTASM to assemble HELP. With this assembler, text can be represented with the DEFM (define message) pseudo instruction (it is called a pseudo instruction because it does not generate machine code). Each string defined with a DEFM consists of text followed by the symbol EOS, which is given the value zero in an EQUate near the start of the program.

Each line of the help message, except the last, ends with a carriage return (CR). This causes printing to resume at the beginning of the next line when the message is displayed. It does not mark the end of the string. Two carriage returns appearing together generate a blank line before printing resumes. Any number of carriage returns may appear within the message string.

If you want to replace or add to the help list with your own entries, Program Listing 2 shows how. If the strings in this program are included in your HELP program, typing HELP MARY SMITH results in this display:

```
MARY SMITH
111 1ST STREET
ATOWN, WHEREVER
PHONE: 123-4567
```

The first keyword in the help list consists only of the EOS code. This is a null string, used when no search keyword is entered in the command line. The null search keyword matches with the null help list keyword. HELP gives you a display of all the available keywords known to HELP.

Program Listing 1

```
                ;         HELP V1.1 25-JUL-82
                ;         AUTHOR: PHIL COMEAU
                ;         DATE WRITTEN: Ø3-DEC-81
                ;
                ;         CONSTANTS
                ;
ØØØ1            BREAK     EQU 1               ;<BREAK>
4318            CML       EQU 4318H           ;DOS COMMAND LINE
ØØØD            CR        EQU ØDH             ;CARRIAGE RETURN
4Ø2D            DOS       EQU 4Ø2DH           ;TRSDOS RE-ENTRY POINT
ØØ33            DSPC      EQU 33H             ;CHARACTER DISPLAY RTN
ØØØØ            EOS       EQU Ø               ;END OF STRING
ØØ2Ø            SPACE     EQU ' '
                ;
                ;         MAINLINE
                ;
52ØØ                      ORG 52ØØH
52ØØ 3EØD       HELP:     LD A,CR             ;LEAVE A BLANK LINE
52Ø2 CD33ØØ               CALL DSPC
52Ø5 11995F               LD DE,KEYWRD        ;GET KEYWORD FROM CMD LINE
52Ø8 CD4852               CALL GETCML
                ;
52ØB 21C252               LD HL,HLPLST
52ØE 11765F     SRCHLP:   LD DE,ENDLST        ;SRCH FOR KW UNTIL WE REACH
5211 E5                   PUSH HL             ;THE END OF THE TABLE
5212 B7                   OR A
5213 ED52                 SBC HL,DE
5215 E1                   POP HL
5216 F23452               JP P,NOTFND
                ;
5219 11995F               LD DE,KEYWRD        ;HAVE WE FOUND THE KEYWORD?
521C 23                   INC HL
521D CD8C52               CALL CMPSTR
522Ø CA2D52               JP Z,FOUND
                ;
5223 CD7F52               CALL SKPSTR         ;IF NOT, FIND END OF KW
5226 23                   INC HL
5227 CD7F52               CALL SKPSTR         ;THEN SKIP TO NEXT KW
522A C3ØE52               JP SRCHLP
                ;
522D 23         FOUND:    INC HL              ;IF FOUND, DISPLAY THE
522E CDB252               CALL DSPSTR         ;HELP MESSAGE
5231 C34Ø52               JP HLEXIT           ;AND END
                ;
5234 21765F     NOTFND:   LD HL,NFMSG         ;IF NOT FOUND, SAY SO
5237 CDB252               CALL DSPSTR
523A 21995F               LD HL,KEYWRD
523D CDB252               CALL DSPSTR
                ;
524Ø 3EØD       HLEXIT:   LD A,CR             ;LEAVE A BLANK LINE
5242 CD33ØØ               CALL DSPC
5245 C32D4Ø               JP DOS              ;BACK TO DOS
                ;
                ;
                ;         GETCML: GET PARAM STRING FROM CMD LINE
                ;         ENTRY: DE POINTS TO START OF PARAM STRING BFR
                ;         EXIT:  DE POINTS TO END OF PARAMETER STRING
                ;
```

```
5248 F5         ØØ59Ø GETCML: PUSH AF
5249 E5         ØØ6ØØ         PUSH HL
524A 211843     ØØ61Ø         LD HL,CML           ;POINT TO CMD LINE
                ØØ62Ø ;
524D 7E         ØØ63Ø SKIPNM: LD A,(HL)           ;SKIP PROGRAM NAME
524E FE2Ø       ØØ64Ø         CP SPACE            ;BY FINDING 1ST SPACE
525Ø CA5E52     ØØ65Ø         JP Z,SKIPSP
5253 FEØD       ØØ66Ø         CP CR               ;OR CR
5255 CA5E52     ØØ67Ø         JP Z,SKIPSP
5258 FEØ1       ØØ68Ø         CP BREAK            ;OR BREAK
525A 23         ØØ69Ø         INC HL              ;IN CMD LINE
525B C34D52     ØØ7ØØ         JP SKIPNM
                ØØ71Ø ;
525E 7E         ØØ72Ø SKIPSP: LD A,(HL)           ;FIND 1ST NON-SPACE CHAR
525F FE2Ø       ØØ73Ø         CP SPACE
5261 C26852     ØØ74Ø         JP NZ,LDPRM
5264 23         ØØ75Ø         INC HL
5265 C35E52     ØØ76Ø         JP SKIPSP
                ØØ77Ø ;
5268 7E         ØØ78Ø LDPRM:  LD A,(HL)           ;LOAD PARAM INTO (DE)
5269 FEØD       ØØ79Ø         CP CR               ;UNTIL A <CR>
526B CA7952     ØØ8ØØ         JP Z,GCEXIT
526E FEØ1       ØØ81Ø         CP BREAK            ;OR A <BREAK> IS FOUND
527Ø CA7952     ØØ82Ø         JP Z,GCEXIT
5273 12         ØØ83Ø         LD (DE),A
5274 23         ØØ84Ø         INC HL              ;TRY NEXT CHAR
5275 13         ØØ85Ø         INC DE
5276 C36852     ØØ86Ø         JP LDPRM
                ØØ87Ø ;
5279 3EØØ       ØØ88Ø GCEXIT: LD A,EOS        ;MARK THE END OF STRING
527B 12         ØØ89Ø         LD (DE),A
527C E1         ØØ9ØØ         POP HL
527D F1         ØØ91Ø         POP AF
527E C9         ØØ92Ø         RET
                ØØ93Ø ;
                ØØ94Ø ;
                ØØ95Ø ;       SKPSTR: FIND END OF STRING
                ØØ96Ø ;       ENTRY: HL POINTS TO START OF STRING
                ØØ97Ø ;       EXIT:  HL POINTS TO EOS
                ØØ98Ø ;
527F F5         ØØ99Ø SKPSTR: PUSH AF
528Ø 7E         Ø1ØØØ SKPLP:  LD A,(HL)           ;WHILE (HL)<>EOS
5281 FEØØ       Ø1Ø1Ø         CP EOS
5283 CA8A52     Ø1Ø2Ø         JP Z,SKEXIT
5286 23         Ø1Ø3Ø         INC HL              ;TRY NEXT CHAR
5287 C38Ø52     Ø1Ø4Ø         JP SKPLP
528A F1         Ø1Ø5Ø SKEXIT: POP AF
528B C9         Ø1Ø6Ø         RET
                Ø1Ø7Ø ;
                Ø1Ø8Ø ;
                Ø1Ø9Ø ;       CMPSTR: COMPARE STRINGS
                Ø11ØØ ;       ENTRY: HL POINTS TO START OF STRING1
                Ø111Ø ;              DE POINTS TO START OF STRING2
                Ø112Ø ;       EXIT:  HL POINTS TO END OF STRING1 OR
                Ø113Ø ;                 PLACE WHERE (HL)<>(DE)
                Ø114Ø ;              DE POINTS TO END OF STRING2 OR
                Ø115Ø ;                 PLACE WHERE (HL)<>(DE)
                Ø116Ø ;              AF = -1 IF (HL) < (DE)
                Ø117Ø ;                    Ø IF (HL) = (DE)
                Ø118Ø ;                   +1 IF (HL) > (DE)
```

Program continued

```
                 ;
528C C5          CMPSTR: PUSH BC
528D 1A          CSLP:   LD A,(DE)          ;WHILE (HL)=(DE)
528E BE                  CP (HL)
528F C29D52              JP NZ,NOTEQ
5292 7E                  LD A,(HL)          ;AND (HL)<>EOS
5293 FEØØ                CP EOS
5295 CAAE52              JP Z,EQ
5298 23                  INC HL             ;TRY NEXT CHAR
5299 13                  INC DE
529A C38D52              JP CSLP
                 ;
529D 1A          NOTEQ:  LD A,(DE)          ;IF (HL)<(DE)
529E 47                  LD B,A
529F 7E                  LD A,(HL)
52AØ 9Ø                  SUB B
52A1 F2A952              JP P,GT
52A4 3EFF                LD A,ØFFH          ;THEN RETURN -1
52A6 C3BØ52              JP CSEXIT
52A9 3EØ1        GT:     LD A,1             ;ELSE RETURN +1
52AB C3BØ52              JP CSEXIT
                 ;
52AE 3EØØ        EQ:     LD A,Ø             ;(HL)=(DE), SO RETURN Ø
                 ;
52BØ C1          CSEXIT: POP BC
52B1 C9                  RET
                 ;
                 ;
                 ;       DSPSTR: DISPLAY STRING
                 ;       ENTRY: HL POINTS TO START OF STRING
                 ;       EXIT:  HL POINTS TO EOS
                 ;
52B2 F5          DSPSTR: PUSH AF
52B3 7E          DSLP:   LD A,(HL)          ;WHILE (HL)<>EOS
52B4 FEØØ                CP EOS
52B6 CACØ52              JP Z,DSEXIT
52B9 CD33ØØ              CALL DSPC          ;DISPLAY CHAR
52BC 23                  INC HL
52BD C3B352              JP DSLP
52CØ F1          DSEXIT: POP AF
52C1 C9                  RET
                 ;
                 ;
                 ;       HELP LIST
                 ;       EACH HELP LIST ENTRY IS FORMATTED AS FOLLOWS:
                 ;       <KEYWORD STRING> <HELP MESSAGE STRING>
                 ;
52C2             HLPLST  EQU $
52C2 ØØ                  DEFB EOS
52C3 ØØ                  DEFB EOS
52C4 48                  DEFM 'HELP V1.1  25-JUL-82'
52D8 ØD                  DEFB CR
52D9 ØD                  DEFB CR
52DA 54                  DEFM 'TO GET HELP ON ONE OF THE FOLLOWING '
52FE 43                  DEFM 'COMANDS'
53Ø5 ØD                  DEFB CR
53Ø6 54                  DEFM 'TYPE "HELP <COMMAND>":'
531C ØD                  DEFB CR
531D ØD                  DEFB CR
531E 41                  DEFM 'APPEND   AUTO     ATTRIB   BASIC
```

```
5341 42         DEFM 'BASIC2   BASICR'
535Ø ØD         DEFB CR
5351 43         DEFM 'CLOCK    COPY     DATE     DEBUG    '
5374 44         DEFM 'DEVICE   DIR'
538Ø ØD         DEFB CR
5381 44         DEFM 'DUMP     KILL     FREE     LIB      '
53A4 4C         DEFM 'LIST     LOAD'
53B1 ØD         DEFB CR
53B2 5Ø         DEFM 'PRINT    PROT     RENAME   TIME     '
53D5 54         DEFM 'TRACE    VERIFY'
53E4 ØD         DEFB CR
53E5 ØD         DEFB CR
53E6 2Ø         DEFM '  < > INDICATE OPTIONAL VALUES'
54Ø3 ØD         DEFB CR
54Ø4 2Ø         DEFM '   /  INDICATES CHOICE OF VALUES'
5423 ØØ         DEFB EOS
              ;
5424 41         DEFM 'APPEND'
542A ØØ         DEFB EOS
542B 41         DEFM 'APPEND FILE1 TO FILE2'
544Ø ØD         DEFB CR
5441 ØD         DEFB CR
5442 2Ø         DEFM '  ADDS FILE1 ONTO THE END OF FILE2'
5463 ØØ         DEFB EOS
              ;
5464 41         DEFM 'AUTO'
5468 ØØ         DEFB EOS
5469 41         DEFM 'AUTO COMMAND'
5475 ØD         DEFB CR
5476 ØD         DEFB CR
5477 2Ø         DEFM '  SPECIFIES A COMMAND TO BE EXECUTED '
549B 57         DEFM 'WHEN DOS IS BOOTED'
54AD ØØ         DEFB EOS
              ;
54AE 41         DEFM 'ATTRIB'
54B4 ØØ         DEFB EOS
54B5 41         DEFM 'ATTRIB FILESPEC <(PARM...PARM)>'
54D4 ØD         DEFB CR
54D5 ØD         DEFB CR
54D6 2Ø         DEFM '  ALTERS PROTECTION STATUS OF FILESPEC'
54FB ØD         DEFB CR
54FC 2Ø         DEFM '  FOR MORE INFORMATION TYPE "HELP '
551D 41         DEFM 'ATTRIB PARM"'
5529 ØØ         DEFB EOS
              ;
552A 41         DEFM 'ATTRIB PARM'
5535 ØØ         DEFB EOS      ;2ND LEVEL ENTRY
5536 41         DEFM 'ATTRIB COMMAND PARAMETERS:'
555Ø ØD         DEFB CR
5551 ØD         DEFB CR
5552 2Ø         DEFM '  I            MAKE FILE INVISIBLE'
5573 ØD         DEFB CR
5574 2Ø         DEFM '  ACC=PSW1     ACCESS PASSWORD = PSW1'
5598 ØD         DEFB CR
5599 2Ø         DEFM '  UPD=PSW2     UPDATE PASSWORD = PSW2'
55BD ØD         DEFB CR
55BE 2Ø         DEFM '  PROT=LVL     ACCESS LEVEL = LVL'
55DE ØD         DEFB CR
55DF 2Ø         DEFM '               LVL: '
55F2 4B         DEFM 'KILL/RENAME/WRITE/READ/EXEC'
```

Program continued

```
560ØD ØØ       Ø239Ø          DEFB EOS
               Ø24ØØ ;
560ØE 42       Ø241Ø          DEFM 'BASIC'
5613 ØØ        Ø242Ø          DEFB EOS
5614 42        Ø243Ø          DEFM 'BASIC <*>'
561D ØD        Ø244Ø          DEFB CR
561E ØD        Ø245Ø          DEFB CR
561F 2Ø        Ø246Ø          DEFM '  LOADS THE DISK BASIC INTERPRETER.'
5641 ØD        Ø247Ø          DEFB CR
5642 2Ø        Ø248Ø          DEFM '  BASIC * RETURNS TO BASIC WITHOUT '
5664 44        Ø249Ø          DEFM 'DESTROYING PROGRAM'
5676 ØØ        Ø25ØØ          DEFB EOS
               Ø251Ø ;
5677 42        Ø252Ø          DEFM 'BASIC2'
567D ØØ        Ø253Ø          DEFB EOS
567E 42        Ø254Ø          DEFM 'BASIC2'
5684 ØD        Ø255Ø          DEFB CR
5685 ØD        Ø256Ø          DEFB CR
5686 2Ø        Ø257Ø          DEFM '  RETURNS TO LEVEL II (NON-DISK) BASIC '
56AC 49        Ø258Ø          DEFM 'INTERPRETER'
56B7 ØØ        Ø259Ø          DEFB EOS
               Ø26ØØ ;
56B8 42        Ø261Ø          DEFM 'BASICR'
56BE ØØ        Ø262Ø          DEFB EOS
56BF 42        Ø263Ø          DEFM 'BASIC <*>'
56C8 ØD        Ø264Ø          DEFB CR
56C9 ØD        Ø265Ø          DEFB CR
56CA 2Ø        Ø266Ø          DEFM '  SAME AS BASIC BUT INCLUDES '
56E6 52        Ø267Ø          DEFM 'RENUMBERING CAPABILITY'
56FC ØØ        Ø268Ø          DEFB EOS
               Ø269Ø ;
56FD 43        Ø27ØØ          DEFM 'CLOCK'
57Ø2 ØØ        Ø271Ø          DEFB EOS
57Ø3 43        Ø272Ø          DEFM 'CLOCK <(ON/OFF)>'
5713 ØD        Ø273Ø          DEFB CR
5714 ØD        Ø274Ø          DEFB CR
5715 2Ø        Ø275Ø          DEFM '  TURNS REAL-TIME CLOCK DISPLAY ON OR '
573A 4F        Ø276Ø          DEFM 'OFF'
573D ØØ        Ø277Ø          DEFB EOS
               Ø278Ø ;
573E 43        Ø279Ø          DEFM 'COPY'
5742 ØØ        Ø28ØØ          DEFB EOS
5743 43        Ø281Ø          DEFM 'COPY FILE1 TO FILE2'
5756 ØD        Ø282Ø          DEFB CR
5757 ØD        Ø283Ø          DEFB CR
5758 2Ø        Ø284Ø          DEFM '  MAKES A DUPLICATE COPY OF FILE1 '
5779 43        Ø285Ø          DEFM 'CALLED FILE2'
5785 ØØ        Ø286Ø          DEFB EOS
               Ø287Ø ;
5786 44        Ø288Ø          DEFM 'DATE'
578A ØØ        Ø289Ø          DEFB EOS
578B 44        Ø29ØØ          DEFM 'DATE MM/DD/YY'
5798 ØD        Ø291Ø          DEFB CR
5799 ØD        Ø292Ø          DEFB CR
579A 2Ø        Ø293Ø          DEFM '  SETS SYSTEM DATE'
57AB ØØ        Ø294Ø          DEFB EOS
               Ø295Ø ;
57AC 44        Ø296Ø          DEFM 'DEBUG'
57B1 ØØ        Ø297Ø          DEFB EOS
57B2 44        Ø298Ø          DEFM 'DEBUG <(ON/OFF)>'
57C2 ØD        Ø299Ø          DEFB CR
```

```
57C3 ØD         Ø3ØØØ           DEFB CR
57C4 2Ø         Ø3Ø1Ø           DEFM '  ACTIVATES OR DEACTIVATES DEBUGGING '
57E8 4D         Ø3Ø2Ø           DEFM 'MONITOR'
57EF ØD         Ø3Ø3Ø           DEFB CR
57FØ 2Ø         Ø3Ø4Ø           DEFM '  FOR MORE INFORMATION TYPE "HELP DEBUG '
5817 43         Ø3Ø5Ø           DEFM 'COMMANDS"'
582Ø ØØ         Ø3Ø6Ø           DEFB EOS
                Ø3Ø7Ø ;
5821 44         Ø3Ø8Ø           DEFM 'DEBUG COMMANDS'
582F ØØ         Ø3Ø9Ø           DEFB EOS
583Ø 2Ø         Ø31ØØ           DEFM '  DEBUG COMMANDS:
584Ø ØD         Ø311Ø           DEFB CR
5841 2Ø         Ø312Ø           DEFM '  A           SETS DISPLAY TO ASCII '
5865 46         Ø313Ø           DEFM 'FORMAT'
586B ØD         Ø314Ø           DEFB CR
586C 2Ø         Ø315Ø           DEFM '  C           SINGLE STEP, EXECUTE CALLS'
5894 ØD         Ø316Ø           DEFB CR
5895 2Ø         Ø317Ø           DEFM '  D ADR       DISPLAY MEMORY STARTING '
58BB 41         Ø318Ø           DEFM 'AT ADR'
58C1 ØD         Ø319Ø           DEFB CR
58C2 2Ø         Ø32ØØ           DEFM '  G A1,B1,B2  JUMP TO A1; B1 & B2 ARE '
58E8 4F         Ø321Ø           DEFM 'OPT. BREAKPOINTS'
58F8 ØD         Ø322Ø           DEFB CR
58F9 2Ø         Ø323Ø           DEFM '  H           SETS DISPLAY TO '
5917 48         Ø324Ø           DEFM 'HEXADECIMAL FORMAT'
5929 ØD         Ø325Ø           DEFB CR
592A 2Ø         Ø326Ø           DEFM '  I           SINGLE STEP'
5943 ØD         Ø327Ø           DEFB CR
5944 2Ø         Ø328Ø           DEFM '  M ADR       MODIFY MEMORY STARTING AT '
596C 41         Ø329Ø           DEFM 'ADR'
596F ØD         Ø33ØØ           DEFB CR
597Ø 2Ø         Ø331Ø           DEFM '  R RP VAL    LOADS VAL INTO REGISTER '
5996 5Ø         Ø332Ø           DEFM 'PAIR RP'
599D ØD         Ø333Ø           DEFB CR
599E 2Ø         Ø334Ø           DEFM '  S           SETS DISPLAY TO FULL '
59C1 53         Ø335Ø           DEFM 'SCREEN MODE'
59CC ØD         Ø336Ø           DEFB CR
59CD 2Ø         Ø337Ø           DEFM '  U           SETS DYNAMIC DISPLAY '
59FØ 55         Ø338Ø           DEFM 'UPDATE MODE'
59FB ØD         Ø339Ø           DEFB CR
59FC 2Ø         Ø34ØØ           DEFM '  X           SETS DISPLAY TO REGISTER '
5A23 46         Ø341Ø           DEFM 'FORMAT'
5A29 ØD         Ø342Ø           DEFB CR
5A2A 2Ø         Ø343Ø           DEFM '  ;/-         INCREMENTS/DECREMENTS PAGE'
5A52 ØØ         Ø344Ø           DEFB EOS
                Ø345Ø ;
5A53 44         Ø346Ø           DEFM 'DEVICE'
5A59 ØØ         Ø347Ø           DEFB EOS
5A5A 44         Ø348Ø           DEFM 'DEVICE'
5A6Ø ØD         Ø349Ø           DEFB CR
5A61 ØD         Ø35ØØ           DEFB CR
5A62 2Ø         Ø351Ø           DEFM '  LISTS ALL CURRENTLY DEFINED I/O '
5A83 44         Ø352Ø           DEFM 'DEVICES'
5A8A ØØ         Ø353Ø           DEFB EOS
                Ø354Ø ;
5A8B 44         Ø355Ø           DEFM 'DIR'
5A8E ØØ         Ø356Ø           DEFB EOS
5A8F 44         Ø357Ø           DEFM 'DIR <:D> (<S,I,A>)'
5AA1 ØD         Ø358Ø           DEFB CR
5AA2 ØD         Ø359Ø           DEFB CR
5AA3 2Ø         Ø36ØØ           DEFM '  DISPLAYS DIRECTORY OF DISK IN DRIVE :D'
```

Program continued

```
5ACA ØD        Ø361Ø          DEFB CR
5ACB 2Ø        Ø362Ø          DEFM '  FOR MORE INFORMATION TYPE "HELP DIR '
5AFØ 4F        Ø363Ø          DEFM 'OPT"'
5AF4 ØØ        Ø364Ø          DEFB EOS
               Ø365Ø ;
5AF5 44        Ø366Ø          DEFM 'DIR OPT'
5AFC ØØ        Ø367Ø          DEFB EOS
5AFD 44        Ø368Ø          DEFM 'DIR OPTIONS:'
5BØ9 ØD        Ø369Ø          DEFB CR
5BØA ØD        Ø37ØØ          DEFB CR
5BØB 2Ø        Ø371Ø          DEFM '  S  DISPLAY ALL SYSTEM AND '
5B26 4E        Ø372Ø          DEFM 'NON-INVISIBLE FILES'
5B39 ØD        Ø373Ø          DEFB CR
5B3A 2Ø        Ø374Ø          DEFM '  I  DISPLAY ALL INVISIBLE AND '
5B58 4E        Ø375Ø          DEFM 'NON-SYSTEM FILES'
5B68 ØD        Ø376Ø          DEFB CR
5B69 2Ø        Ø377Ø          DEFM '  A  DISPLAY DISK SPACE ALLOCATION'
5B8A ØØ        Ø378Ø          DEFB EOS
               Ø379Ø ;
5B8B 44        Ø38ØØ          DEFM 'DUMP'
5B8F ØØ        Ø381Ø          DEFB EOS
5B9Ø 44        Ø382Ø          DEFM 'DUMP FILE (START=X''AAAA'',END=X''BBBB'''
5BA2 3C        Ø383Ø          DEFM '<,TRA=X''CCCC''>)'
5BA9 ØD        Ø384Ø          DEFB CR
5BAA ØD        Ø385Ø          DEFB CR
5BAB 2Ø        Ø386Ø          DEFM '  DUMPS MEMORY FROM ADDRESS AAAA TO '
5BCE 42        Ø387Ø          DEFM 'BBBB TO DISK,'
5BDB ØD        Ø388Ø          DEFB CR
5BDC 2Ø        Ø389Ø          DEFM '  WITH FILESPEC "FILE".  WHEN FILE IS '
5CØ1 4C        Ø39ØØ          DEFM 'LOADED, EXECUTION'
5C12 ØD        Ø391Ø          DEFB CR
5C13 2Ø        Ø392Ø          DEFM '  WILL BEGIN AT ADDRESS CCCC (IF '
5C33 53        Ø393Ø          DEFM 'SUPPLIED)'
5C3C ØØ        Ø394Ø          DEFB EOS
               Ø395Ø ;
5C3D 46        Ø396Ø          DEFM 'FREE'
5C41 ØØ        Ø397Ø          DEFB EOS
5C42 46        Ø398Ø          DEFM 'FREE'
5C46 ØD        Ø399Ø          DEFB CR
5C47 ØD        Ø4ØØØ          DEFB CR
5C48 2Ø        Ø4Ø1Ø          DEFM '  DISPLAYS FREE SPACE ON ALL DISKS'
5C69 ØØ        Ø4Ø2Ø          DEFB EOS
               Ø4Ø3Ø ;
5C6A 4B        Ø4Ø4Ø          DEFM 'KILL'
5C6E ØØ        Ø4Ø5Ø          DEFB EOS
5C6F 4B        Ø4Ø6Ø          DEFM 'KILL FILESPEC'
5C7C ØD        Ø4Ø7Ø          DEFB CR
5C7D ØD        Ø4Ø8Ø          DEFB CR
5C7E 2Ø        Ø4Ø9Ø          DEFM '  DELETES FILESPEC FROM DISK'
5C99 ØØ        Ø41ØØ          DEFB EOS
               Ø411Ø ;
5C9A 4C        Ø412Ø          DEFM 'LIB'
5C9D ØØ        Ø413Ø          DEFB EOS
5C9E 4C        Ø414Ø          DEFM 'LIB'
5CA1 ØD        Ø415Ø          DEFB CR
5CA2 ØD        Ø416Ø          DEFB CR
5CA3 2Ø        Ø417Ø          DEFM '  DISPLAYS NAMES OF DOS COMMANDS'
5CC2 ØØ        Ø418Ø          DEFB EOS
               Ø419Ø ;
5CC3 4C        Ø42ØØ          DEFM 'LIST'
5CC7 ØØ        Ø421Ø          DEFB EOS
```

```
5CC8 4C                 DEFM 'LIST FILESPEC'
5CD5 ØD                 DEFB CR
5CD6 ØD                 DEFB CR
5CD7 2Ø                 DEFM '  DISPLAYS CONTENTS OF FILESPEC ON '
5CF9 53                 DEFM 'SCREEN'
5CFF ØØ                 DEFB EOS
                   ;
5DØØ 4C                 DEFM 'LOAD'
5DØ4 ØØ                 DEFB EOS
5DØ5 4C                 DEFM 'LOAD FILESPEC'
5D12 ØD                 DEFB CR
5D13 ØD                 DEFB CR
5D14 2Ø                 DEFM '  LOADS FILESPEC FROM DISK TO MEMORY'
5D37 ØØ                 DEFB EOS
                   ;
5D38 5Ø                 DEFM 'PRINT'
5D3D ØØ                 DEFB EOS
5D3E 5Ø                 DEFM 'PRINT FILESPEC'
5D4C ØD                 DEFB CR
5D4D ØD                 DEFB CR
5D4E 2Ø                 DEFM '  PRINTS CONTENTS OF FILESPEC ON PRINTER'
5D75 ØØ                 DEFB EOS
                   ;
5D76 5Ø                 DEFM 'PROT'
5D7A ØØ                 DEFB EOS
5D7B 5Ø                 DEFM 'PROT <:D> <(PARM...PARM)>'
5D94 ØD                 DEFB CR
5D95 ØD                 DEFB CR
5D96 2Ø                 DEFM '  CHANGES PROTECTION STATUS OF ALL '
5DB8 46                 DEFM 'FILES ON DRIVE :D'
5DC9 ØD                 DEFB CR
5DCA 2Ø                 DEFM '  FOR MORE INFORMATION TYPE "HELP PROT '
5DFØ 5Ø                 DEFM 'PARM"'
5DF5 ØØ                 DEFB EOS
                   ;
5DF6 5Ø                 DEFM 'PROT PARM'
5DFF ØØ                 DEFB EOS
5EØØ 5Ø                 DEFM 'PROT PARAMETERS:'
5E1Ø ØD                 DEFB CR
5E11 ØD                 DEFB CR
5E12 2Ø                 DEFM '  PW      CHANGE MASTER PASSWORD'
5E32 ØD                 DEFB CR
5E33 2Ø                 DEFM '  UNLOCK  REMOVE PASSWORDS FROM USER '
5E58 46                 DEFM 'FILES'
5E5D ØD                 DEFB CR
5E5E 2Ø                 DEFM '  LOCK    ASSIGN MASTER PASSWORD TO '
5E82 55                 DEFM 'USER FILES'
5E8C ØØ                 DEFB EOS
                   ;
5E8D 52                 DEFM 'RENAME'
5E93 ØØ                 DEFB EOS
5E94 52                 DEFM 'RENAME FILE1 TO FILE2'
5EA9 ØD                 DEFB CR
5EAA ØD                 DEFB CR
5EAB 2Ø                 DEFM '  CHANGES NAME OF FILE1 TO FILE2'
5ECA ØØ                 DEFB EOS
                   ;
5ECB 54                 DEFM 'TIME'
5ECF ØØ                 DEFB EOS
5EDØ 54                 DEFM 'TIME HH:MM:SS'
5EDD ØD                 DEFB CR
```

Program continued

```
5EDE ØD         Ø483Ø          DEFB CR
5EDF 2Ø         Ø484Ø          DEFM '  SETS REAL TIME CLOCK'
5EF4 ØØ         Ø485Ø          DEFB EOS
                Ø486Ø ;
5EF5 54         Ø487Ø          DEFM 'TRACE'
5EFA ØØ         Ø488Ø          DEFB EOS
5EFB 54         Ø489Ø          DEFM 'TRACE <(ON/OFF)>'
5FØB ØD         Ø49ØØ          DEFB CR
5FØC ØD         Ø491Ø          DEFB CR
5FØD 2Ø         Ø492Ø          DEFM '  SETS DISPLAY OF PC REGISTER ON OR OFF'
5F33 ØØ         Ø493Ø          DEFB EOS
                Ø494Ø ;
5F34 56         Ø495Ø          DEFM 'VERIFY'
5F3A ØØ         Ø496Ø          DEFB EOS
5F3B 56         Ø497Ø          DEFM 'VERIFY <(ON/OFF)>'
5F4C ØD         Ø498Ø          DEFB CR
5F4D ØD         Ø499Ø          DEFB CR
5F4E 2Ø         Ø5ØØØ          DEFM '  SETS DISK WRITE VERIFICATION ON OR OFF'
5F75 ØØ         Ø5Ø1Ø          DEFB EOS
                Ø5Ø2Ø ;
5F76            Ø5Ø3Ø ENDLST   EQU $
5F76 53         Ø5Ø4Ø NFMSG:   DEFM 'SORRY -- NO HELP IS AVAILABLE FOR '
5F98 ØØ         Ø5Ø5Ø          DEFB EOS
5F99            Ø5Ø6Ø KEYWRD   EQU $
                Ø5Ø7Ø ;
52ØØ            Ø5Ø8Ø          END HELP
ØØØØØ TOTAL ERRORS
```

Program Listing 2

```
;
;       HELP LIST
;       EACH HELP LIST ENTRY IS FORMATTED AS FOLLOWS:
;       <KEYWORD STRING> <HELP MESSAGE STRING>
;
HLPLST  EQU $           ;THIS MARKS START OF HELP LIST
        DEFB EOS
        DEFB EOS
        DEFM 'HELP V1.1  25-JUL-82'
        DEFB CR
        DEFB CR
        DEFM 'THIS IS EXAMPLE 1.'
        DEFB CR
        DEFM 'THESE NAMES ARE KEYWORDS IN A SAMPLE'
        DEFB CR
        DEFM 'NAME AND ADDRESS LIST:'
        DEFB CR
        DEFB CR
        DEFM 'ALICE ARMSTRONG, MARY SMITH, BRIDGET TURNER'
        DEFB CR
        DEFM 'SHARON WILLIAMS'
        DEFB CR
        DEFB CR
        DEFM 'TO FIND AN ADDRESS AND PHONE NUMBER, TYPE'
        DEFB CR
        DEFM '  "HELP <NAME>"'
        DEFB EOS
;
```

```
        DEFM 'ALICE ARMSTRONG'  ;THIS IS THE KEYWORD
        DEFB EOS                ;EOS MARKS END OF KEYWORD
                                ;INFO FOR ALICE A. FOLLOWS
        DEFM 'ALICE ARMSTRONG'
        DEFB CR
        DEFM '377 OCTAL AVENUE'
        DEFB CR
        DEFM 'TROIS-RIVERES, QUEBEC'
        DEFB CR
        DEFM 'PHONE: 745-1263'
        DEFB EOS                ;EOS MARKS END OF INFO
;
        DEFM 'MARY SMITH'
        DEFB EOS
        DEFM 'MARY SMITH'
        DEFB CR
        DEFM '111 1ST STREET'
        DEFB CR
        DEFM 'ATOWN, WHEREVER'
        DEFB CR
        DEFM 'PHONE: 123-4567'
        DEFB EOS
;
        DEFM 'BRIDGET TURNER'
        DEFB EOS
        DEFM 'BRIDGET TURNER'
        DEFB CR
        DEFM '2 RADIUS CIRCLE'
        DEFB CR
        DEFM 'COMPASS, ONTARIO'
        DEFB CR
        DEFM 'PHONE: 314-1592'
        DEFB EOS
;
        DEFM 'SHARON WILLIAMS'
        DEFB EOS
        DEFM 'SHARON WILLIAMS'
        DEFB CR
        DEFM '8Ø8Ø PROCESSOR BLVD'
        DEFB CR
        DEFM 'MEMORY LAKE, MANITOBA'
        DEFB CR
        DEFM 'PHONE: 28Ø-4116'
        DEFB EOS
;
ENDLST  EQU $           ;THIS MARKS THE END OF HELP LIST
```

17

Automatic Master Disk Directory

by Jack R. Smith

System Requirements:
Model III
One disk drive
TRSDOS-compatible DOS
Printer optional

Do you get tired of thumbing through your collection of disks looking for a particular program? My program, INDEX, automatically reads the directory information from each disk, then sorts and saves the information on a master disk program index.

The basis of the first, extraction part of INDEX is a machine-language TRSDOS RAM call, $RAMDIR, which allows you to examine a disk directory one entry at a time or altogether. $RAMDIR is an easy routine to invoke from BASIC. Set the HL, B, and C registers of the Z80 to the initial condition described in the *Model III Disk System Owner's Manual* (Figure 1). $RAMDIR is then called with a USR 0 instruction. When the directory has been written into the specified memory locations, control is returned to the BASIC interpreter.

The initialization, calling and return functions are accomplished in the short assembler program shown in Program Listing 1. Line 100 exchanges the register set in current use with the alternate set (the prime registers), saving the contents of the normal register set. Line 110 loads the HL register pair with the starting address of the memory location where the directory information is to be placed.

Since approximately 1761 bytes of directory information have to be accommodated, as well as the calling program, a 48K machine should start storing the directory about 2000 bytes below the top of the memory, or at F700H. Line 120 tells the routine to read the directory using drive 1. Line 130 selects transfer of the entire directory into memory. Line 140 transfers control to the $RAMDIR routine. Line 150 restores the normal register set of the Z80 so that return to BASIC is possible without loss of information. Line 160 returns control to the BASIC interpreter.

The directory information is written sequentially into memory, starting with the address loaded into the HL register. According to Radio Shack, this directory information is written in blocks 21 bytes in length, in the format FILENAME/EXT:DR, left justified, and padded with trailing blanks for a length of 15 bytes, followed by the file's protection level information, its length, and other information which is not of interest here (see Figure 2).

Register	Contents	Purpose
HL	F700H	Points to start of destination address of directory information F700H for 64K machines.
B	1H	Drive to be read from
C	0H	Switch—0H copies entire directory

Figure 1. *Entry conditions to $RAMDIR*

```
                                    A
                          File names without /EXT

1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21
                  Filename:d                    P   E   L     S      G

                                    B
                           File names with /EXT

1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22
                  Filename:d                        P   E   L     S      G
```

P: Protection Level
E: End of File
L: Logical Record Length
S: Last Sector in File
G: Number of Granules in File

Figure 2. *Directory files returned by $RAMDIR*

FILENAME/EXT is the user-generated file name and optional extension under which the program is stored. DR is the drive number that the disk is read from, in this case drive number 1. A program saved with the command SAVE "HELLO/BAS" is transferred into memory by $RAMDIR as HELLO/BAS:1bbbbXXXXXXX, where b indicates a blank and X stands for numeric information not used in the indexing program. The last entry in the directory is a plus sign (+).

Any file stored with an extension takes 22 characters, while file names without an extension are transferred into memory using 21 characters. Consequently, each directory entry must be examined for a slash (/), which if found tells the program that the start of the next directory entry is found 22 characters after the file under examination. If no slash is found, then the next file entry begins 21 characters later.

The second part of the program reads the resulting files into the array A$(n), sorts the array, and prints the sorted array. Disks are identified with a user-supplied two-digit serial number from 01 to 99, which is then appended to the FILENAME/EXT extracted from the first part of the program. After the program has read the last disk, a dummy disk number of 0 is entered and control branches to the print routine.

To speed up the sort, I use the TRSDOS CMD"O" machine-language sort routine, which takes only five seconds to sort 400 entries. Since printing 400 directory entries—one per line—would use seven pages, I use a 4-column format, which resembles a dictionary in layout and permits the same amount of information to be contained on two pages. The print parameters are based on an Epson MX-100 printer, with automatic page performation skip.

Operating the program is simple. The program assumes that you have numbered each of your disks with a different ID number between 1 and 99. INDEX also requires that you leave a TRSDOS system disk in drive 0 while the program is reading your disk from drive 1. If you wish to read the disk that is in drive 0, substitute any other TRSDOS system disk.

When the program is run, it asks for the ID number of the first disk to be read, then if that disk is mounted on drive number 1. If your reply is Y, the directory is read and the directory names are displayed on the screen as they are read from memory by the program. When the last name has been read, you are asked for the next disk ID number. If no more disks are to be read, type 0 as an ID, and an alphabetized master directory will be printed as hard copy.

Program Analysis

Lines 100–160: Initialize program, set MEMSIZE

Lines 170–240: Set up disk ID and make sure disk is mounted.

Lines 250–330: Line 250 tells the BASIC program the address to transfer control to when the machine-language routine is called. Lines 260–330 are the machine-language routine and the POKE statements for putting it into memory.

Line 340: Calls the machine-language routine.

Lines 500–520: Initialize the memory reading section.

Lines 530–610: Examine the memory, byte by byte. Non-printing characters are stripped, and the colon (:) delimiter is searched for. Control is transferred out of the loop when each FILENAME/EXT is read or when the end of directory is reached.

Lines 1000–1030: Prompt operator for all disks after the first one.

Lines 1200–1270: Subroutine which is called after a complete record has been found in lines 530-610. The end of a record is identified by a colon (:). The disk ID is added to the file name and the resulting string stored in A$(K). Based on the presence or absence of a slash (/), the next file name is looked for 21 or 22 characters after the start of the file name which has just been completed.

Lines 4000-4060: Print a header at the top of the page in double wide characters, and the date of the run.

Line 4070: Activates automatic page perforation skip feature of Epson printers.

Lines 4080-4100: Use TRSDOS utility sort routine.

Lines 4110-4180: Break the sorted array into quarters, and then print the four columns.

Changes for 32K Machines

Line 130 POKE 16562,&HF7 : POKE 16561,&HB7
Line 140 CLEAR 12000
Line 250 DEFUSR0=&HBF00
Line 260 DATA 217,33,248,183,6,
1,14,0,205,144,66,217,201
Line 310 POKE (&HBF00+X),P
Line 530 A$=CHR$(PEEK(&HB7F8+NN))

Changes for Other 132-Column Printers

Line 4030 LPRINT TAB(53) "MASTER DISK INDEX LIST"
Line 4070 Delete this line.

Program Listing 1. *$RAMDIR calling program*

```
0000 D9        00100        EXX                       ;SWAP REGISTERS
0001 2100F7    00110        LD      HL,0F700H         ;LOAD WITH START ADDRESS
0004 0601      00120        LD      B,1               ;USE DRIVE NUMBER 1
0006 0E00      00130        LD      C,0               ;GET ENTIRE DIRECTORY
0008 CD9044    00140        CALL    4490H             ;CALL $RAMDIR
000B D9        00150        EXX                       ;SWAP REGISTERS BACK
000C C9        00160        RET                       ;BACK TO BASIC
0000           00170        END
00000 TOTAL ERRORS
```

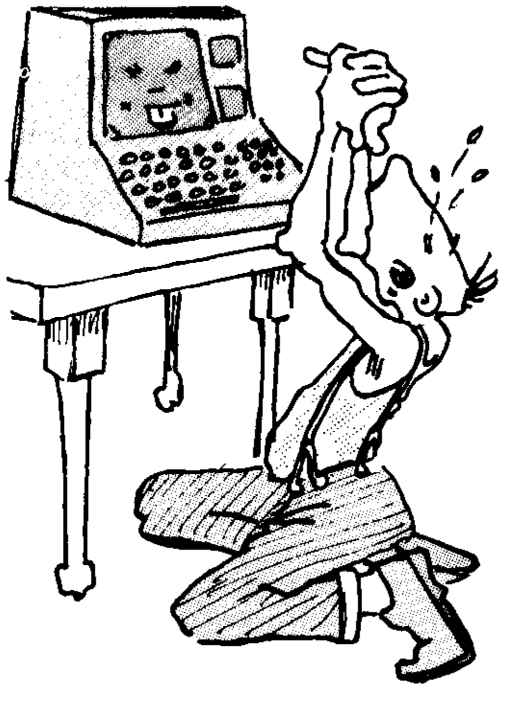
Program Listing 2. *INDEX*

```
100 'MODEL III DISK INDEXER
110 'BY JACK R. SMITH K8ZOA
120 'VERSION 1.0  APRIL 11,1982
125 'NO NEED TO SET MEM SIZE
130 POKE 16562,&HF6 : POKE 16561,&HFF:
```

Program continued

```
     'SETS MEMSIZE
135 'IF 32K POKE 16562,&HB7 : POKE 16561,&HF8
140 CLEAR 22000

       :'(48K) IF 32K CLEAR 12000
150 DIM A$(1200)        :'A$ HOLDS PROGRAM NAMES
160 CLS
170 PRINT@512,"";:INPUT "ENTER DISK ID NUMBER ";ID
180 IF ID<0 OR ID>99 THEN GOTO 160
190 IF ID=0 THEN GOTO 4000
200 ID$=STR$(ID)         :'CONVERT ID NO. TO STRING
210 PRINT@ 512, STRING$(60,32)
220 PRINT@512,"IS DISK NUMBER" ID$ ;:INPUT " MOUNTED ON DRIVE #1
";Y$
230 IF LEFT$(Y$,1)<>"Y" THEN GOTO 210
240 '
250 DEFUSR0=&HFF00
255 ' IF 32K THEN DEFUSR0=&HBF00
260 DATA 217,33,0,247,6,1,14,0,205,144,66,217,201
270 'LINE 260 FOR 48K IF 32K CHANGE TO
280 'DATA 217,33,248,183,6,1,14,0,205,144,66,217,201
290 FOR X=0 TO 12        :'POKE INTO HIGH MEMORY
300     READ P
310     POKE (&HFF00+X),P        :'CHANGE TO &HBF00 IF 32K
320     NEXT X
330 RESTORE                      :'FOR NEXT DISK
340 J=USR0(0)                    :'LOAD DIRECTORY INTO MEMORY
350 '
500 ' READ DIRECTORY INFORMATION FROM MEMORY AND PUT INTO A$( )
505 PRINT@512,STRING$(60,32)
510 M=0          :'POSITION COUNTER
520 NN=M         :'CHARACTER COUNTER
530 A$=CHR$(PEEK(&HF700+NN))     :'LOOK AT DIRECTORY ONE CHR AT A
 TIME
535 ' IF 32K THEN PEEK(&HB7F8+NN)
540  IF A$<"!" THEN A$=""        :'CLEAN OUT NONPRINT CHRS.
550 IF A$>"Z" THEN A$=""         :'DITTO
560 NN=NN+1                      :'ADVANCE COUNTER
570 IF A$="+" THEN GOTO 1000     :'END OF DIRECTORY MARKED BY ++
580 IF A$=":" THEN GOSUB 1200 : GOTO 520        :'END OF RECORD
590 B$=B$+A$
600 PRINT@512,B$
610 GOTO 530                     :READY FOR NEXT CHARACTER
1000 CLS
1010 PRINT@512,"LOAD NEXT DISK INTO DRIVE #1."
1020 PRINT@576,"ENTER DISK ID NUMBER OR 0 TO HARDCOPY SORTED LIS
T";:INPUTID
1030 GOTO 180
1200 CLS
1210 'PUT RECORD INTO A$( )
1220 A$(K)=B$+ ":" + ID$        :'RECORD FORMAT FILENAM/EXT :ID
1230 K=K+1
1235 ' TRSDOS HAS TWO LENGTHS -- 21 IF NO /EXT AND 22 IF USE /EX
T
1240 IF INSTR(B$,"/")=0 THEN M=M+21 ELSE M=M+22
1250 B$=""      :'READY FOR NEXT RECORD
1260 PRINT@512,STRING$(60,32)
1270 RETURN
4000 CLS
4010 'PRINT AND SORT SECTION
4020 'WRITTEN FOR EPSON MX-100 PRINTER
```

```
4030 LPRINT TAB(44) CHR$(14) CHR$(155) "E" "MASTER DISK INDEX LI
ST" CHR$(18) CHR$(155) "F" :'DOUBLE WIDE DOUBLE STRIKE HEADER
4040 LPRINT STRING$(2,10)
4050 LPRINT "DATE PREPARED: "LEFT$(TIME$,8)
4060 LPRINT STRING$(2,10)
4070 LPRINT CHR$(155) "N"       :'AUTO PAGE SKIP WITH EPSON
4080 N%=K+1
4090 'USE TRSDOS MACHINE SORT ROUTINE
4095 CLS: PRINT@512,"SORTING -- READY PRINTER"
4100 CMD "O",N%,A$(0)
4110 '
4120 K4=INT(K/4)+1      :'FOR FOUR COLUMN ACROSS
4130 FOR L=1 TO K4
4140 SP=33              :'132 COLUMN PRINTER
4150 LPRINT A$(L) TAB(SP) A$(L+K4) TAB(2*SP) A$(L+2*K4) TAB(3*SP
) A$(L+3*K4)
4160 NEXT L
4170 LPRINT CHR$(12)    :'FORM FEED
4180 END
```



18

Short Form Directory

by Salvatore Yorks

System Requirements:
Model III
TRSDOS
One disk drive
Editor/assembler

I've developed a short machine-language program, D/CMD, that uses a simple command to put the BASIC directory on the screen after the TRSDOS READY prompt. This saves you time and frustration when you can't remember how the program name was abbreviated. I keep a copy on every disk I own, usually as an invisible file that is transparent to other users, and cannot be killed accidentally. You can use this program on all existing versions of DOS, 1.1, 1.2, and 1.3.

TRSDOS uses a buffer located at 4225H for all DOS commands. It stops reading a program name when it encounters a space. To use D/CMD, type D, press the space bar,and type 0, 1, 2, or 3. If you call for a directory by pressing D without entering a number, drive 0 is the default drive. The resulting display is the same directory you get when you use the CMD"D:#" command in BASIC. It lists all the program names on any disk together, without scrolling off the screen.

D/CMD is a complete, stand-alone program. It occupies the same area as DEBUG, 5200H-5272H, so no BASIC program resident in memory is disturbed when the directory is called. The code is completely relocatable; you can move it by changing the ORG statement in line 2200 of the assembly listing.

Once you've entered the program using an editor/assembler, you can create a build file, using the BUILD command in TRSDOS. Type BUILD SB and press ENTER. After each prompt, type the following lines:

```
COPY D/CMD:0 :1
ATTRIB D/CMD:1 (I)
D:1
```

Now press the BREAK key and you're ready to update all your disks.

•Place the disk with the BUILD file in drive 0.

•Place the disk to receive D/CMD in drive 1.

•Type DO SD and press ENTER.

•Repeat as often as necessary to convert all disks.

•You might want to set the AUTO function to DO SD and simply press RESET after exchanging disks.

This copies the program from drive 0 to drive 1, then changes the file's visibility so that no one will mistakenly delete it. Then it runs the program from drive 1, to make sure it's there, and has been rendered invisible. Once converted, you can call up a usable directory that won't run off the screen. If you want or need the full file information, you have to run the fill DIR system function.

Program Listing. *Directory Routine*

```
                00200 ;* * * * * * * * * * * * * * * * * * * * * * * * * * *
                00300 ;*                                                    *
                00400 ;*                MODEL III - SHORT                   *
                00500 ;*                DIRECTORY ROUTINE                   *
                00600 ;*                     REVISED                        *
                00700 ;*                    07/04/82                        *
                00800 ;*                                                    *
                00900 ;*                                                    *
                01000 ;*                                                    *
                01100 ;*                                                    *
                01200 ;* * * * * * * * * * * * * * * * * * * * * * * * * * *
                01300 ;                     DEFINED ADDRESSES
01C9            01400 CLS     EQU     01C9H           ;CLEAR SCREEN .
4271            01500 DRIVE   EQU     4271H           ;POINTER TO DRIVE SELECTED FOR DIRECTORY .
4419            01600 DSPDIR  EQU     4419H           ;DIRECTORY DISPLAY - BASIC FORMAT .
4225            01700 TXT     EQU     4225H           ;DOS TEXT BUFFER .
4290            01800 RAMDIR  EQU     4290H           ;SELECTIVE DIR ENTRY .
0033            01900 VDCHAR  EQU     0033H           ;DISPLAY A SINGLE CHARACTER .
021B            02000 VDLINE  EQU     021BH           ;DISPLAY AN ENTIRE MESSAGE .
4210            02100 SPECL   EQU     4210H           ;SPECIAL CHARACTER SWITCH REGISTER.
5200            02200         ORG     5200H           ;DOS "DEBUG" PROGRAM AREA .
                02300 ;                     PROGRAM BEGINS HERE
5200 CDC901     02400 START   CALL    CLS             ;CLEAR THE SCREEN !
5203 3E28       02500         LD      A,40            ;SETS SPECIAL CHARACTERS
5205 321042     02600         LD      (SPECL),A       ;TURNS OFF KANA CHARACTERS!
5208 1600       02700         LD      D,0             ;0'S TO FILL MEMORY .
520A 217042     02800         LD      HL,DRIVE-1      ;START HERE - CLEARS DRIVE POINTER .
520D 010500     02900         LD      BC,5            ;FILL 5 BYTES .
5210 CD7B52     03000         CALL    FILL            ;DO IT TO IT !
5213 010A00     03100         LD      BC,10           ;FILL 10 BYTES .
5216 21C352     03200         LD      HL,FREE         ;START HERE .
5219 CD7B52     03300         CALL    FILL            ; DO IT TO IT AGAIN !
                03400 ;               DISPLAY BASIC FORMAT DIRECTORY
521C 3A2742     03500         LD      A,(TXT+2)       ;GET DRIVE # FROM DOS COMMAND BUFFER .
521F FE31       03600         CP      '1'             ;IS IT A "1" ?
5221 280A       03700         JR      Z,EXE           ;OK     DO IT!
5223 FE32       03800         CP      '2'             ;IS IT A "2" ?
5225 2806       03900         JR      Z,EXE           ;OK     DO IT!
5227 FE33       04000         CP      '3'             ;IS IT A "3" ?
5229 2802       04100         JR      Z,EXE           ;OK     DO IT!
522B 3E30       04200         LD      A,'0'           ;IF IT ISN'T 1,2 OR 3 IT MUST BE 0!
522D 327142     04300 EXE     LD      (DRIVE),A       ;POINT TO TARGET DRIVE .
5230 CD1944     04400         CALL    DSPDIR          ;DISPLAY THE TARGETED DIRECTORY .
```

Program continued

```
               Ø45ØØ ;               GET FREE SPACE INFORMATION
5233 21C352    Ø46ØØ         LD      HL,FREE         ;POINT TO FREE SPACE BUFFER .
5236 ØEFF      Ø47ØØ         LD      C,255           ;ASK FOR FREE SPACE INFO .
5238 3A7142    Ø48ØØ         LD      A,(DRIVE)       ;GET DRIVE NUMBER .
523B D63Ø      Ø49ØØ         SUB     3ØH             ;CONVERT TO TRUE HEX VALUE !
523D 47        Ø5ØØØ         LD      B,A             ;GET HEX # INTO B .
523E CD9Ø42    Ø51ØØ         CALL    RAMDIR          ;GET FREE SPACE INFO .
               Ø52ØØ ;              DISPLAY DIRECTORY TITLE
5241 218352    Ø53ØØ         LD      HL,MSG1         ;POINT TO FREE SPACE MESSAGE .
5244 CD1BØ2    Ø54ØØ         CALL    VDLINE          ;DISPLAY MESSAGE .
               Ø55ØØ ;          CONVERT FREE SPACE INFO FROM HEX TO DECIMAL
5247 2AC552    Ø56ØØ         LD      HL,(FREE+2)     ;GET ASCII INTO HL .
524A DD216E52  Ø57ØØ         LD      IX,DECTBL       ;POINTER .
524E AF        Ø58ØØ PDEC1   XOR     A               ;CLEAR A .
524F DD46Ø1    Ø59ØØ         LD      B,(IX+1)        ; BC HOLDS THE
5252 DD4EØØ    Ø6ØØØ         LD      C,(IX)          ;           DECIMAL DIGIT .
5255 B7        Ø61ØØ         OR      A               ;CLEAR CARRY .
5256 ED42      Ø62ØØ PDEC2   SBC     HL,BC           ;SUBTRACT BC .
5258 38Ø3      Ø63ØØ         JR      C,PDEC3         ;DIGIT DONE .
525A 3C        Ø64ØØ         INC     A               ;ELSE INC A .
525B 18F9      Ø65ØØ         JR      PDEC2           ;CONTINUE .
525D Ø9        Ø66ØØ PDEC3   ADD     HL,BC           ;ADD BACK .
525E C63Ø      Ø67ØØ         ADD     A,3ØH           ;"Ø" TO "9" .
               Ø68ØØ ;          DISPLAY DECIMAL FREE SPACE INFO
526Ø CD33ØØ    Ø69ØØ         CALL    VDCHAR          ;DISPLAY IT .
5263 79        Ø7ØØØ         LD      A,C             ;IF C=1
5264 FEØ1      Ø71ØØ         CP      1               ; THEN DONE .
5266 28ØC      Ø72ØØ         JR      Z,DSP           ; DISPLAY REST OF MESSAGE WHEN DEC. CONV. IS DONE .
5268 DD23      Ø73ØØ         INC     IX              ; DONE TWICE BECAUSE
526A DD23      Ø74ØØ         INC     IX              ;    THE TABLE IS STORED IN TWO BYTE WORDS .
526C 18EØ      Ø75ØØ         JR      PDEC1           ; CONTINUE CONVERSION .
526E 64ØØ      Ø76ØØ DECTBL  DEFW    1ØØ             ;TABLE USED FOR DECIMAL
527Ø ØAØØ      Ø77ØØ         DEFW    1Ø              ;CONVERSION .
5272 Ø1ØØ      Ø78ØØ         DEFW    1               ;
               Ø79ØØ ;           DISPLAY FREE SPACE MESSAGE
5274 21A852    Ø8ØØØ DSP     LD      HL,MSG2         ;POINT TO NEXT MESSAGE .
5277 CD1BØ2    Ø81ØØ         CALL    VDLINE          ;DISPLAY IT !
               Ø82ØØ ;         RETURN TO DISK CALLER WHEN DONE
527A C9        Ø83ØØ         RET                     ;RETURN - DONE
               Ø84ØØ ;               FILL MEMORY WITH SPECIFIC BYTE
527B 72        Ø85ØØ FILL    LD      (HL),D          ;STORE THE BYTE TO USE .
527C 23        Ø86ØØ         INC     HL              ;POINT TO NEXT BYTE .
527D ØB        Ø87ØØ         DEC     BC              ;ADJUST THE COUNT .
527E 78        Ø88ØØ         LD      A,B             ;MSB OF COUNT .
527F B1        Ø89ØØ         OR      C               ;MERGE LSB .
528Ø 2ØF9      Ø9ØØØ         JR      NZ,FILL         ;CONTINUE 'TILL DONE .
5282 C9        Ø91ØØ         RET                     ;RETURN WHEN DONE .
               Ø92ØØ ;                   SCREEN MESSAGE AREA
5283 15        Ø93ØØ MSG1    DEFB    21              ;SWITCH TO SPECIAL CHARACTERS
5284 EE        Ø94ØØ         DEFB    238             ;FAT "X"
5285 EE        Ø95ØØ         DEFB    238             ;FAT "X"
5286 EE        Ø96ØØ         DEFB    238             ;FAT "X"
5287 2Ø        Ø97ØØ         DEFM    ' Model III Short Directory '
52A2 8F        Ø98ØØ         DEFB    143             ;SHIRT CUFF FOR POINTING HAND
52A3 F4        Ø99ØØ         DEFB    244             ;POINTING
52A4 F5        1ØØØØ         DEFB    245             ;    HAND AND
52A5 F6        1Ø1ØØ         DEFB    246             ;        FINGER
52A6 2Ø        1Ø2ØØ         DEFB    2ØH             ;SPACE   .
52A7 Ø3        1Ø3ØØ         DEFB    Ø3H             ;END-OF-MESSAGE TERMINATOR - NO C.R.
52A8 2Ø        1Ø4ØØ MSG2    DEFM    ' Free Gran(s) '
52B6 EE        1Ø5ØØ         DEFB    238             ;FAT "X" FOR BOARDER
52B7 EE        1Ø6ØØ         DEFB    238             ;        FOR
52B8 EE        1Ø7ØØ         DEFB    238             ;          MESSAGE .
52B9 2Ø        1Ø8ØØ         DEFM    ' SLY '         ;I ALWAYS SIGN MY WORK !
52BE EE        1Ø9ØØ         DEFB    238             ;FAT "X" FOR
52BF EE        11ØØØ         DEFB    238             ;     BORDER .
52CØ EE        111ØØ         DEFB    238             ;       AT END OF LINE
52C1 15        112ØØ         DEFB    21              ; SWITCH BACK TO SPACE COMPRESSION CODES
52C2 ØD        113ØØ         DEFB    ØDH             ;END-OF-MESSAGE TERMINATOR - WITH C.R.
               114ØØ ;        STORAGE BUFFER FOR FREE SPACE INFORMATION
ØØØA           115ØØ FREE    DEFS    1Ø              ;STORAGE AREA FOR FREE SPACE INFORMATION .
52ØØ           116ØØ         END     START           ;PROGRAM ENDS HERE !
ØØØØØ TOTAL ERRORS
```

19

ONESTEP at a Time in BASIC

by Alan Sehmer

System Requirements:
Model I
32K RAM
Disk BASIC
One disk drive
TRSDOS

Writing a BASIC program is easy; the hard part is making it work. The best way to debug is to run the program one command at a time. ONESTEP lets you run a BASIC program one command at a time, while displaying the current line number and up to 26 user-defined variables. The program may be run at close to normal speed, or at a very slow speed.

Once ONESTEP is entered and loaded, load disk BASIC. Answer the Memory Size? question with 48128 to keep BASIC from overwriting ONESTEP. Load the program you want to debug. To turn ONESTEP on, add this line to the target program: DEFUSR 0=(&HBC00): * = USR 0(0), where the asterisk is a dummy variable. The line can be put anywhere. If you have a long program that runs fine until the last few lines, ONESTEP need not be activated until just before the problem lines.

ONESTEP understands six commands.

SHIFT S	Go into the one-step mode. Print the BASIC line number and the variables requested.
ENTER	Execute the next command. Display line number and variables.
N	Leave the one-step mode. Run program at normal speed. Display line numbers in upper right.
S	Leave the one-step mode. Run program at slow speed. Display line numbers in upper right.
C	Change the variables to monitor.

K Kill ONESTEP. This entirely disconnects ONESTEP. The DEFUSR line must be executed to restart it.

With ONESTEP on, a program can be stopped with the BREAK key only in modes N or S. The program can then be edited or restarted with CONT, RUN or GOTO. It is unwise to stop a program in the S mode; nothing is hurt, but the keyboard is slow to receive input. ONESTEP uses the variables QA$-QZ$ to store the names of the variables to be printed. Because all variables are destroyed by the EDIT, CLEAR and RUN commands, ONESTEP automatically executes a C command if any of these commands is used. ONESTEP also executes the C command immediately after the DEFUSR line. Any type of variable can be used with the C command. This includes variables that use other variables, such as B$(X) or A(I,J). ONESTEP executes only one command each time ENTER is pressed. You must press ENTER several times to complete a multi-statement line.

How It Works

ONESTEP is a machine-language program written for a 32K Model I with disk and TRSDOS. When ONESTEP is first called, it ties itself to the front of the BASIC keyboard driver. Every time BASIC calls the keyboard (after each command), it calls ONESTEP instead. ONESTEP tests a set of flags and variables to see what needs to be done, does it, and jumps to the real keyboard driver.

In all but three cases this causes no problems. However, the INPUT, LINEINPUT and INKEY$ commands must scan the keyboard, which requires calling the keyboard driver and puts you back into ONESTEP. INPUT and LINEINPUT do a repetitive keyboard scan, so the program would never get past them. INKEY$ does a one-shot scan, so it always returns a null string. Patching sections of code—INSET into the INPUT routine, LINSET into the LINEINPUT routine, and INRES into the main interpreter—solves the problem with INPUT and LINEINPUT. No simple fix works for INKEY$.

One way around the INKEY$ problem is to temporarily replace the INKEY$ function with a LET statement, making the INKEY$ variable whatever you would have entered. Because ONESTEP uses QA$-QZ$ to store variable names, they must not be used in the target program. To change ONESTEP's variable names, change line 2020. To adjust the slow speed, change line 390.

ONESTEP commands are straightforward, but the display can be confusing. A PRINT@ command spreads ONESTEP output all over the screen. This is intentional. If lines were set aside for ONESTEP, program output could easily be overwritten. It is best to have ONESTEP output and program output on separate devices. Insert the following lines if you have a printer:

```
552   LD  HL,058DH
554   LD  (401EH),HL
1712  LD  HL,0458H
1714  LD  (401EH),HL
1742  LD  HL,0458H
1744  LD  (401EH),HL
```

The last problem involves monitoring array variables and is best illustrated by the following example:

```
10 DEFUSR 0=(&HBC00) : 0=USR 0(0) : DIM Y(50)
20 FOR X=0 TO 50
30 Y(X)=X
40 NEXT X
```

If ONESTEP is told to monitor Y(X), a subscript-out-of-range error occurs at line 40. In a FOR-NEXT loop the NEXT command increments the FOR variables and tests it against the TO argument. If the TO argument has been satisfied, program flow falls through the FOR-NEXT loop. However, the FOR variable is now greater than the TO argument (in the example X=51). At line 40 ONESTEP tries to print the variable Y(51), but Y is only dimensioned to 50; hence, the error.

ONESTEP is a good aid in teaching as well as in debugging. It is hard for people to catch the ins and outs of programming when one moment there's a program on the screen and the next the computer is printing answers. ONESTEP lets you see step-by-step what the computer is doing.

Program Listing

```
              ;             ONESTEP BY AL SEHMER  Ø8/31/81
              ;
BCØØ                        ORG     ØBCØØH         ;BCØØ = 48128
BCØØ 21B2BD                 LD      HL,INRES       ;ON RETURN FROM "INPUT"
BCØ3 22C541                 LD      (41C5H),HL     ;COMMAND PT. TO INRES
BCØ6 21A5BD                 LD      HL,LINSET      ;LOAD LINSET JUMP INTO
BCØ9 22A441                 LD      (41A4H),HL     ;"LINEINPUT" COMMAND
BCØC 3EC3                   LD      A,ØC3H         ;LOAD INSET JUMP INTO
BCØE 32B857                 LD      (57B8H),A      ;"INPUT" COMMAND
BC11 2195BD                 LD      HL,INSET
BC14 22B957                 LD      (57B9H),HL
BC17 2171BE                 LD      HL,FLAG        ;INIT FLAG
BC1A 363Ø                   LD      (HL),3ØH       ;SINGLE STEP ON
BC1C 2123BC                 LD      HL,START       ;SET NEW DRIVER
BC1F 22164Ø                 LD      (4Ø16H),HL
BC22 C9                     RET                    ;RETURN FOR USR CALL
BC23 2171BE   START         LD      HL,FLAG        ;BASIC ENTRY POINT
BC26 CB7E                   BIT     7,(HL)         ;"INPUT" COMMAND ?
BC28 C2D843                 JP      NZ,43D8H       ;IF YES BACK TO BASIC
BC2B 2AA24Ø                 LD      HL,(4ØA2H)
BC2E 222141                 LD      (4121H),HL     ;BASIC LINE # TO REG1
BC31 Ø1ØAØA                 LD      BC,ØAØAH
BC34 213A3C                 LD      HL,3C3AH       ;PT. TO SCREEN
BC37 CD2F13                 CALL    132FH          ;TO ASCII & DISPLAY
```

Program continued

```
BC3A 3E20      00340          LD      A,20H
BC3C 323F3C    00350          LD      (3C3FH),A
BC3F 2171BE    00360          LD      HL,FLAG          ;TEST FOR NORM. OR SLOW
BC42 CB76      00370          BIT     6,(HL)
BC44 2808      00380          JR      Z,TEST
BC46 21FF1F    00390          LD      HL,1FFFH
BC49 2B        00400 DELAY    DEC     HL
BC4A 7C        00410          LD      A,H
BC4B B5        00420          OR      L
BC4C 20FB      00430          JR      NZ,DELAY
BC4E 2171BE    00440 TEST     LD      HL,FLAG
BC51 CB66      00450          BIT     4,(HL)           ;IS SINGLE SET FLAG SET
BC53 2011      00460          JR      NZ,MAIN
BC55 3A0438    00470          LD      A,(3804H)        ;IS 'S' KEY PRESSED
BC58 218038    00480          LD      HL,3880H         ;PT. TO 'SHIFT' KEY
BC5B B6        00490          OR      (HL)
BC5C FE09      00500          CP      9                ;ARE BOTH PRESSED
BC5E C2D843    00510          JP      NZ,43D8H         ;IF NOT BACK TO BASIC
BC61 2171BE    00520          LD      HL,FLAG
BC64 CBE6      00530          SET     4,(HL)           ;SET SINGLE STEP FLAG
BC66 21D843    00540 MAIN     LD      HL,43D8H         ;RESTORE OLD DRIVER
BC69 221640    00550          LD      (4016H),HL
BC6C 3E41      00560          LD      A,41H            ;DO I NEED TO GET VARS.
BC6E 323BBE    00570          LD      (BUF1+1),A       ;BUF1 = QA$
BC71 213ABE    00580          LD      HL,BUF1          ;PT. TO BUF1
BC74 CD0D26    00590          CALL    260DH            ;WHERE IS QA$ STORED
BC77 1A        00600          LD      A,(DE)           ;DE PTS. TO LENGTH OF QA$
BC78 A7        00610          AND     A                ;SET OR RESET ZERO FLAG
BC79 2035      00620          JR      NZ,PRTLNE        ;NO NEED, SKIP GETVAR
BC7B 3A71BE    00630 GETVAR   LD      A,(FLAG)
BC7E E6F0      00640          AND     0F0H             ;CLEAR VAR. COUNT
BC80 3271BE    00650          LD      (FLAG),A
BC83 CDC901    00660          CALL    01C9H            ;CLEAR SCREEN
BC86 21BFBD    00670          LD      HL,GETMSG        ;PT. TO MESSAGE
BC89 CD6F20    00680          CALL    206FH            ;PRINT IT
BC8C 3A71BE    00690          LD      A,(FLAG)         ;GET VAR. COUNT
BC8F E60F      00700 NEXT     AND     0FH              ;MASK OUT CONTROL BITS
BC91 C641      00710          ADD     A,41H            ;ADD OFFSET
BC93 323BBE    00720          LD      (BUF1+1),A       ;INSERT INTO BUFFER
BC96 213ABE    00730          LD      HL,BUF1          ;PT. TO INPUT BUFFER
BC99 CD9457    00740          CALL    5794H            ;INPUT VARIABLE
BC9C 213ABE    00750          LD      HL,BUF1          ;WAS INPUT A 'ENTER'
BC9F CD0D26    00760          CALL    260DH            ;WHERE IS ENTRY STORED
BCA2 1A        00770          LD      A,(DE)           ;DE PTS. TO ENTRY LENGTH
BCA3 A7        00780          AND     A                ;SET OR RESET ZERO FLAG
BCA4 CA2CBD    00790          JP      Z,PRTINS         ;IF ZERO DONE WITH INPUT
BCA7 3A71BE    00800          LD      A,(FLAG)         ;GET VAR. COUNT
BCAA 3C        00810          INC     A
BCAB 3271BE    00820          LD      (FLAG),A         ;RESAVE COUNT
BCAE 18DF      00830          JR      NEXT             ;GET NEXT VARIABLE
BCB0 21F6BD    00840 PRTLNE   LD      HL,LNEMSG        ;PT. TO MESSAGE
BCB3 CD6F20    00850          CALL    206FH            ;PRINT IT
BCB6 2AA240    00860          LD      HL,(40A2H)       ;GET BASIC LINE NUMBER
BCB9 CDAF0F    00870          CALL    0FAFH            ;TO ASCII & DISPLAY
BCBC 3A71BE    00880          LD      A,(FLAG)         ;GET VAR. COUNT
BCBF E60F      00890          AND     0FH
BCC1 FE00      00900          CP      0                ;IS COUNT EQUAL TO ZERO
BCC3 28B6      00910          JR      Z,GETVAR
BCC5 3E00      00920          LD      A,0              ;INIT # OF VARS. PRINTED
BCC7 3270BE    00930          LD      (TEMP),A
BCCA 2170BE    00940          LD      HL,TEMP
```

```
BCCD CD6F2Ø      ØØ95Ø          CALL    2Ø6FH           ;START NEW LINE
BCDØ C641        ØØ96Ø MORE     ADD     A,41H           ;ADD OFFSET
BCD2 323BBE      ØØ97Ø          LD      (BUF1+1),A      ;PUT IN BUF1
BCD5 213ABE      ØØ98Ø          LD      HL,BUF1         ;PT. TO BUF1
BCD8 CDØD26      ØØ99Ø          CALL    26ØDH           ;WHERE IS VAR. STORED
BCDB ED536EBE    Ø1ØØØ          LD      (BUF3),DE       ;STORE ADDRESS IN BUF3
BCDF 213EBE      Ø1Ø1Ø          LD      HL,BUF2         ;PT. TO STRING FOR PRINT
BCE2 3622        Ø1Ø2Ø          LD      (HL),22H        ;ADD " TO BUF2
BCE4 23          Ø1Ø3Ø          INC     HL
BCE5 EB          Ø1Ø4Ø          EX      DE,HL
BCE6 Ø1ØØØØ      Ø1Ø5Ø          LD      BC,Ø            ;CLEAR BC FOR LDIR
BCE9 DD2A6EBE    Ø1Ø6Ø          LD      IX,(BUF3)       ;PT. TO VAR. NAME LENGTH
BCED DD4EØØ      Ø1Ø7Ø          LD      C,(IX+Ø)        ;C=LENGTH OF NAME
BCFØ DD6EØ1      Ø1Ø8Ø          LD      L,(IX+1)
BCF3 DD66Ø2      Ø1Ø9Ø          LD      H,(IX+2)        ;HL=LOC. OF VARIABLE NAME
BCF6 EDBØ        Ø11ØØ          LDIR                    ;MOVE VAR. NAME TO BUF2
BCF8 EB          Ø111Ø          EX      DE,HL
BCF9 363D        Ø112Ø          LD      (HL),3DH        ;ADD = TO BUF2
BCFB 23          Ø113Ø          INC     HL
BCFC 3622        Ø114Ø          LD      (HL),22H        ;ADD " TO BUF2
BCFE 23          Ø115Ø          INC     HL
BCFF 363B        Ø116Ø          LD      (HL),3BH        ;ADD ; TO BUF2
BDØ1 23          Ø117Ø          INC     HL
BDØ2 EB          Ø118Ø          EX      DE,HL
BDØ3 Ø1ØØØØ      Ø119Ø          LD      BC,Ø
BDØ6 DD4EØØ      Ø12ØØ          LD      C,(IX+Ø)
BDØ9 DD6EØ1      Ø121Ø          LD      L,(IX+1)        ;SAME FUNCTION AS ABOVE
BDØC DD66Ø2      Ø122Ø          LD      H,(IX+2)
BDØF EDBØ        Ø123Ø          LDIR
BD11 EB          Ø124Ø          EX      DE,HL
BD12 362C        Ø125Ø          LD      (HL),2CH        ;ADD , TO BUF2
BD14 23          Ø126Ø          INC     HL
BD15 36ØØ        Ø127Ø          LD      (HL),Ø          ;ADD TERMINATOR
BD17 213EBE      Ø128Ø          LD      HL,BUF2         ;PT. TO OUTPUT BUFFER
BD1A CD6F2Ø      Ø129Ø          CALL    2Ø6FH           ;PRINT IT
BD1D 3A71BE      Ø13ØØ          LD      A,(FLAG)        ;GET VAR. COUNT
BD2Ø E6ØF        Ø131Ø          AND     ØFH             ;MASK OUT CONTROL BITS
BD22 217ØBE      Ø132Ø          LD      HL,TEMP         ;PT. TO TEMP
BD25 34          Ø133Ø          INC     (HL)
BD26 BE          Ø134Ø          CP      (HL)            ;AM I DONE ?
BD27 3A7ØBE      Ø135Ø          LD      A,(TEMP)
BD2A 2ØA4        Ø136Ø          JR      NZ,MORE
BD2C 3EØØ        Ø137Ø PRTINS   LD      A,Ø
BD2E 327ØBE      Ø138Ø          LD      (TEMP),A
BD31 217ØBE      Ø139Ø          LD      HL,TEMP
BD34 CD6F2Ø      Ø14ØØ          CALL    2Ø6FH           ;START NEW LINE
BD37 21ØCBE      Ø141Ø          LD      HL,INSMSG       ;PT. TO MESSAGE
BD3A CD6F2Ø      Ø142Ø          CALL    2Ø6FH           ;PRINT IT
BD3D CD49ØØ      Ø143Ø GETKEY   CALL    ØØ49H           ;SCAN KEYBOARD
BD4Ø FE4E        Ø144Ø          CP      4EH             ;IS IT 'N'
BD42 2ØØ9        Ø145Ø          JR      NZ,SKEY
BD44 2171BE      Ø146Ø          LD      HL,FLAG         ;PT. TO FLAG
BD47 CBA6        Ø147Ø          RES     4,(HL)          ;CLEAR SINGLE STEP FLAG
BD49 CBB6        Ø148Ø          RES     6,(HL)          ;CLEAR SLOW FLAG
BD4B 183F        Ø149Ø          JR      DONE            ;RETURN TO BASIC
BD4D FE53        Ø15ØØ SKEY     CP      53H             ;IS IT 'S'
BD4F 2ØØ9        Ø151Ø          JR      NZ,CKEY
BD51 2171BE      Ø152Ø          LD      HL,FLAG
BD54 CBA6        Ø153Ø          RES     4,(HL)          ;CLEAR SINGLE STEP FLAG
BD56 CBF6        Ø154Ø          SET     6,(HL)          ;SET SLOW FLAG
BD58 1832        Ø155Ø          JR      DONE
```

Program continued

```
BD5A FE43      01560 CKEY    CP      43H            ;IS IT 'C'
BD5C CA7BBC    01570         JP      Z,GETVAR
BD5F FE0D      01580         CP      0DH            ;IS IT 'ENTER'
BD61 2829      01590         JR      Z,DONE
BD63 FE4B      01600         CP      4BH            ;IS IT 'K'
BD65 20D6      01610         JR      NZ,GETKEY
BD67 21D843    01620         LD      HL,43D8H       ;RESTORE OLD DRIVER
BD6A 221640    01630         LD      (4016H),HL
BD6D 218657    01640         LD      HL,5786H       ;REMOVE LINSET JUMP FROM
BD70 22A441    01650         LD      (41A4H),HL     ;THE "LINEINPUT" COMMAND
BD73 DD21B857  01660         LD      IX,57B8H       ;REMOVE INSET JUMP FROM
BD77 DD3600E5  01670         LD      (IX+0),0E5H    ;THE "INPUT" COMMAND
BD7B DD3601FE  01680         LD      (IX+1),0FEH
BD7F DD360223  01690         LD      (IX+2),23H
BD83 21305A    01700         LD      HL,5A30H       ;REMOVE INRES JUMP
BD86 22C541    01710         LD      (41C5H),HL
BD89 C3D843    01720         JP      43D8H          ;BACK TO BASIC
BD8C 2123BC    01730 DONE    LD      HL,START       ;SET NEW DRIVER
BD8F 221640    01740         LD      (4016H),HL
BD92 C3D843    01750         JP      43D8H          ;BACK TO BASIC
BD95 F5        01760 INSET   PUSH    AF             ;SAVE REGS.
BD96 3A71BE    01770         LD      A,(FLAG)
BD99 F680      01780         OR      80H            ;SET "INPUT" CALLED FROM
BD9B 3271BE    01790         LD      (FLAG),A       ;TARGET PROGRAM FLAG
BD9E F1        01800         POP     AF             ;RESTORE REGS.
BD9F E5        01810         PUSH    HL
BDA0 FE23      01820         CP      23H
BDA2 C3BB57    01830         JP      57BBH          ;JP TO DOS EXIT
BDA5 F5        01840 LINSET  PUSH    AF             ;SAVE REGS.
BDA6 3A71BE    01850         LD      A,(FLAG)       ;SET "LINEINPUT" FLAG
BDA9 F680      01860         OR      80H
BDAB 3271BE    01870         LD      (FLAG),A
BDAE F1        01880         POP     AF             ;RESTORE REGS.
BDAF C38657    01890         JP      5786H          ;JP. TO DOS EXIT
BDB2 F5        01900 INRES   PUSH    AF             ;SAVE REGS.
BDB3 3A71BE    01910         LD      A,(FLAG)       ;RESET "INPUT" BIT IN
BDB6 E67F      01920         AND     7FH            ;FLAG REG.  (BIT 7)
BDB8 3271BE    01930         LD      (FLAG),A
BDBB F1        01940         POP     AF             ;RESTORE REGS.
BDBC C3305A    01950         JP      5A30H          ;JP TO DOS EXIT
BDBF 22        01960 GETMSG  DEFM    '"ENTER VARIABLES TO MONITOR, PRESS ENTER TO END INPUT"'
BDF5 00        01970         DEFB    00
BDF6 22        01980 LNEMSG  DEFM    '"WORKING ON LINE # ";'
BE0B 00        01990         DEFB    00
BE0C 22        02000 INSMSG  DEFM    '"<N>ORMAL  <S>LOW  <C>HANGE  <ENTER>  <K>ILL"'
BE39 00        02010         DEFB    00
BE3A 51        02020 BUF1    DEFM    'Q $'
BE3D 00        02030         DEFB    00
0030           02040 BUF2    DEFS    30H
BE6E 0000      02050 BUF3    DEFW    0000
BE70 00        02060 TEMP    DEFB    00
BE71 00        02070 FLAG    DEFB    00
0000           02080         END
00000 TOTAL ERRORS
```

20

Renumber Your BASIC Program

by Gary L. Simonds

System Requirements:
Model I or III
16K RAM
Level II BASIC

Have you ever had to retype part of your BASIC program because you didn't leave enough room between line numbers to make corrections? This renumber utility program is the answer. It also adds a professional look to your programs.

The renumber program lets you renumber a previously written BASIC program. You may specify any starting line (0–65535) and the increment between program lines (1–32767). The renumber program also searches your program for a reference to an existing line number and adjusts that reference accordingly. In Figure 1, the number 17 after the THEN command in line 10 is a reference to line 17.

For an example of the actual operations performed by the renumber program, see Figures 1 and 2. Figure 1 represents an ordinary BASIC program. Figure 2 shows the same program renumbered with a starting line number of 10 and an increment of 5. Not only are the lines renumbered, but the references to line numbers within a program line have also been updated to reflect the correct line numbers.

The renumber program is written in assembly language. The BASIC program in Program Listing 1 POKEs the machine-language code for the renumber program into memory. The actual renumber program is contained in the data statements of Program Listing 1. Program Listing 2 contains the assembly code for the renumber program, for those who prefer to enter the renumber program using an editor/assembler.

```
  5 CLS
 10 IF X=3 THEN 17 ELSE X=X+1
 11 GOTO 25
 17 X=1
 25 ON X GOTO 50,51,100
 50 PRINT "X=1" : GOTO 10
 51 PRINT "X=2" : GOTO 10
100 PRINT "X=3": GOTO 10
```

Figure 1. *A typical BASIC program*

The BASIC Program

Here is how to use Program Listing 1.

•Set the memory size to 31999. The renumber program occupies the memory addresses from 32000 to 32698. Setting the memory size prevents BASIC from using memory that contains the renumber program.

•Load Program Listing 1 and run it. If you have made any mistakes in typing, the program stops and prompts you to check for errors. A checksum is performed on the data statements. If the resulting checksum is not as expected, an error is indicated and you must revise the data statements to match the listing.

•Type NEW and load the program you wish to renumber. Do not run it.

•Enter the SYSTEM mode by typing SYSTEM, and press ENTER. The *? system prompt should appear.

•Type /32093 and press ENTER. This tells the computer to start execution of the machine-language program starting at memory location 32093. The renumber program is now in control. The screen should display the title of the renumber program and the statement ENTER STARTING LINE NUMBER.

•Enter the starting line number desired (0–65535), and press ENTER.

•Enter the increment desired between lines (1–32767).

The renumber program now renumbers your program, then returns to BASIC so you can list or save your program.

```
10 CLS
15 IF X=3 THEN 25 ELSE X=X+1
20 GOTO 30
25 X=1
30 ON X GOTO 35,40,45
35 PRINT "X=1" : GOTO 15
40 PRINT "X=2" : GOTO 15
45 PRINT "X=3" : GOTO 15
```

Figure 2. *The program in Figure 1 renumbered with a starting line number of 10 and an increment between lines of 5*

The Assembly-Language Program

To use the assembly-language program, set the memory size to 31999. Load Program Listing 2. You must load the program in the SYSTEM mode since it is a machine-language program. Do not run it. Return to BASIC, and continue from the third instruction for Program Listing 1 (type NEW, etc.)

Limitations and Bugs

The renumber program only changes references to other line numbers if the reference is preceded by a few special command tokens, including GOTO, GOSUB, ON-GOTO, THEN, ON-GOSUB, and ELSE. If the reference is not preceded by one of these tokens, it is not recognized by the renumber program as a reference.

The program to be renumbered should not contain lines with packed machine language. The renumber program can't tell whether the starting line number and the increment will overshoot the maximum line number of 65535. Be careful to choose a starting line number and increment that will not generate line numbers greater than 65535.

A bug shows up the first time you try to save the program that has just been renumbered. An OM (out of memory) error appears. The second time you try to save the program, it works without error. Almost all commands, except the LIST command, cause an OM error if they are the first command executed after the renumber program has been used. Since CSAVE turns on the tape recorder and wastes tape before the OM error occurs, I usually execute a CLEAR command first. The CLEAR command causes the OM error and doesn't waste any tape or damage the program.

The renumber program occupies the memory space 32000 to 32698, which is a little less than 1000 bytes of code, or 1K. The actual starting address of the program is 32093. There is approximately 15K of memory left for the BASIC program that is to be renumbered. The program execution time varies with the size of the BASIC program to be renumbered and the number of references in it.

How It Works

To understand how the renumber program works, you must understand that each BASIC program line contains a pointer to the next program line, its own line number, the instructions themselves, and an end-of-line indicator (00).

Each BASIC command is stored in memory as a token, not as ASCII characters. For instance, a token of 141 represents a GOTO statement. This saves memory space and makes decoding of the instructions easier. The renumber program searches for the tokens shown in Figure 3.

Command	Token
GOTO	141
GOSUB	145
THEN	202
ELSE	149
ON. . .GOTO,ON. . .GOSUB	161

Figure 3. *BASIC commands and their tokens*

When a token is found, the number following it is a reference. The new reference is then calculated and inserted.

Replacing a reference raises a few possible problems. If the new reference number has more digits than the old reference number, room must be made to accommodate the extra digits. This is done by shifting the remainder of the program higher in memory by the appropriate number of digits. You must also go back and update the pointer of each program line to the following program line. If there are fewer characters in the new reference number, you compress the remaining program lines by the appropriate number of digits. When all the references have been updated, you must update each program line number.

Program Listing 1

```
Ø REM RENUMBER (USING BASIC TO POKE IT IN MEMORY)
                                BY GARY L. SIMONDS
1 REM LINE 1Ø SETS UP VARIABLES
2 REM LINE 2Ø READS DATA, ADDS CHECKSUM, POKES DATA INTO MEMORY
3 REM LINES 4Ø TO 1ØØ CHECK CHECKSUM FOR ERRORS
4 REM LINES 12Ø-52Ø DATA STATEMENTS
5 REM LINE 53Ø PRINTS OUT ERROR IF ANY EXIST
1Ø CLS:DEFINT A,B,C,E,I:DEFSNG S
2Ø FORI=32ØØØ TO 32698: READ A :S=S+A: POKE I,A
3Ø C=S
4Ø IF I=32Ø98 THEN S=Ø:IF C<>6826 THEN B=12Ø:E=16Ø:GOSUB 53Ø
5Ø IF I=32198 THEN S=Ø:IF C<>11181 THEN B=17Ø:E=22Ø:GOSUB 53Ø
6Ø IF I=32299 THEN S=Ø:IF C<>1186Ø THEN B=23Ø:E=28Ø:GOSUB 53Ø
7Ø IF I=324ØØ THEN S=Ø:IF C<>12Ø12 THEN B=29Ø:E=34Ø:GOSUB 53Ø
8Ø IF I=32499 THEN S=Ø:IF C<>11861 THEN B=35Ø:E=4ØØ:GOSUB 53Ø
9Ø IF I=32597 THEN S=Ø:IF C<>12235 THEN B=41Ø:E=46Ø:GOSUB 53Ø
1ØØ IF I=32698 THEN S=Ø:IF C<>8876 THEN B=47Ø:E=52Ø:GOSUB 53Ø
11Ø NEXT I:END
12Ø DATA 82,69,78,85,77,66,69,82,32,66,89,32,71,65,82,89,32,76,4
6,32
13Ø DATA 83,73,77,79,78,68,83,13,54,45,5Ø,53,45,56,49,32,82,69,8
6,73
14Ø DATA 83,73,79,78,32,48,46,48,13,69,78,84,69,82,32,83,84,65,8
2,84
15Ø DATA 73,78,71,32,76,73,78,69,32,78,85,77,66,69,82,13,69,78,8
4,69
16Ø DATA 82,32,73,78,67,82,69,77,69,78,84,13,225,2Ø5,2Ø1,1,33,Ø,
125
```

```
170 DATA 205,212,125,33,28,125,205,212,125,62,13,205,51,0,33,49,
125
180 DATA 205,166,125,34,153,127,33,76,125,205,166,125,125,180,32
,2
190 DATA 46,1,34,155,127,124,254,128,48,207,42,164,64,34,157,127
,205
200 DATA 228,125,205,109,127,42,249,64,34,251,64,34,253,64,195,2
5
210 DATA 26,205,212,125,6,5,33,177,127,205,64,0,33,176,127,215,4
8
220 DATA 165,33,177,127,205,108,14,58,175,64,254,2,40,12,254,4,3
2
230 DATA 148,58,36,65,205,251,10,235,201,42,33,65,201,126,35,254
,13
240 DATA 202,224,125,205,51,0,24,244,205,51,0,201,34,185,127,94,
35
250 DATA 86,35,237,83,161,127,94,35,86,237,83,163,127,35,126,167
,40
260 DATA 22,254,141,40,39,254,145,40,35,254,149,40,31,254,202,40
,27
270 DATA 254,161,40,42,24,229,35,126,167,32,7,35,126,167,32,1,20
1
280 DATA 43,237,91,161,127,223,32,211,24,190,215,40,232,48,204,2
29
290 DATA 205,90,30,225,237,83,165,127,205,94,126,24,190,215,40,2
13
300 DATA 254,141,40,4,254,145,32,245,215,40,202,48,14,229,205,90
,30
310 DATA 225,237,83,165,127,205,94,126,24,237,254,44,194,247,125
,24
320 DATA 230,34,159,127,1,0,0,42,157,127,94,35,86,213,123,178,20
2
330 DATA 63,127,35,94,35,86,42,165,127,223,40,4,225,3,24,233,209
,17
340 DATA 0,0,121,176,40,4,42,155,127,235,42,153,127,25,121,176,4
0
350 DATA 5,11,121,176,32,246,34,169,127,34,33,65,33,171,127,1,7,
7
360 DATA 205,47,19,42,159,127,205,68,127,121,50,167,127,205,76,1
27
370 DATA 121,50,168,127,58,167,127,145,40,51,242,12,127,245,167,
42
380 DATA 249,64,229,237,75,159,127,237,66,229,193,225,229,209,19
,237
390 DATA 184,42,249,64,35,34,249,64,42,161,127,35,34,161,127,235
,42
400 DATA 185,127,115,35,114,205,252,26,241,60,32,208,58,168,127,
79
410 DATA 6,0,33,171,127,237,91,159,127,237,176,58,168,127,42,159
,127
420 DATA 133,111,62,0,140,103,201,245,167,42,249,64,237,75,159,1
27
430 DATA 237,66,229,193,42,159,127,229,209,35,237,176,42,249,64,
43
440 DATA 34,249,64,42,161,127,43,34,161,127,235,42,185,127,115,3
5
450 DATA 114,205,252,26,241,61,32,207,24,177,209,42,159,127,201,
14
460 DATA 0,43,215,208,12,24,251,14,0,33,171,127,43,215,208,254,4
8
470 DATA 40,5,12,215,208,24,251,229,197,213,1,5,0,229,209,35,237
,176
```

Program continued

```
480 DATA 209,193,225,24,228,42,155,127,235,42,153,127,167,237,82
,34
490 DATA 183,127,42,157,127,94,35,86,123,178,200,213,35,229,42,1
55
500 DATA 127,235,42,183,127,25,34,183,127,235,225,115,35,114,225
,24
510 DATA 228,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0
520 DATA 0,0,0,0,0,0,0
530 PRINT"CHECK DATA LINES ";B;" TO ";E;". THERE IS AN ERROR.":R
ETURN
```

Program Listing 2

```
7D00           00010           ORG     7D00H
7D00 52        00020 TEXT1     DEFM    'RENUMBER BY GARY L. SIMONDS'
7D1B 0D        00030           DEFB    0DH
7D1C 36        00040 TEXT2     DEFM    '6-25-81 REVISION 0.0'
7D30 0D        00050           DEFB    0DH
7D31 45        00060 TEXT3     DEFM    'ENTER STARTING LINE NUMBER'
7D4B 0D        00070           DEFB    0DH
7D4C 45        00080 TEXT4     DEFM    'ENTER INCREMENT'
7D5B 0D        00090           DEFB    0DH
7D5C E1        00100 BEGIN1    POP     HL       ;DUMMY POP
               00110 ;******************************************
               00120 ;*           BEGIN
               00130 ;*  BEGIN IS THE START OF THE RENUMBER
               00140 ;*  ROUTINE.
               00150 ;*  1. PROMPTS FOR PROPER INPUTS
               00160 ;*  2. RENUMBERS REFERENCES
               00170 ;*  3. RENUMBERS LINES
               00180 ;*  4. RETURNS TO BASIC
               00190 ;*  INTERNAL CALLS-DISPLY,INPKB,NEWLIN,UPDATE
               00200 ;*  EXTERNAL CALLS-01C9-CLEARS SCREEN,1A19-
               00210 ;*  RETURNS TO BASIC,0033-DISPLAYS CHARACTER
               00220 ;******************************************
7D5D CDC901    00230 BEGIN     CALL    01C9H    ;CLEAR THE SCREEN
7D60 21007D    00240           LD      HL,TEXT1
7D63 CDD47D    00250           CALL    DISPLY   ;DISPLAY TITLE & DATE
7D66 211C7D    00260           LD      HL,TEXT2
7D69 CDD47D    00270           CALL    DISPLY
7D6C 3E0D      00280           LD      A,0DH
7D6E CD3300    00290           CALL    0033H
7D71 21317D    00300           LD      HL,TEXT3     ;PROMPT FOR STARTING LINE #
7D74 CDA67D    00310           CALL    INPKB
7D77 22997F    00320           LD      (START),HL       ;SAVE IT IN 'START'
7D7A 214C7D    00330           LD      HL,TEXT4
7D7D CDA67D    00340           CALL    INPKB    ;PROMPT FOR INCREMENT
7D80 7D        00350           LD      A,L
7D81 B4        00360           OR      H
7D82 2002      00370           JR      NZ,NOT0    ;IF INCREMENT=0, FORCE IT TO 1
7D84 2E01      00380           LD      L,01
7D86 229B7F    00390 NOT0      LD      (INCR),HL        ;SAVE IT IN 'INCR'
7D89 7C        00400           LD      A,H
7D8A FE80      00410           CP      80H
7D8C 30CF      00420           JR      NC,BEGIN ;IF INCR IS NOT NUMERIC, START OVER
7D8E 2AA440    00430           LD      HL,(40A4H)
7D91 229D7F    00440           LD      (STRBSC),HL      ;GET STARTING ADDR. OF BASIC
```

```
7D94 CDE47D   ØØ45Ø          CALL    NEWLN   ;SEARCH AND CORRECT REFERENCES
7D97 CD6D7F   ØØ46Ø          CALL    UPDATE  ;UPDATE ALL LINE #S
7D9A 2AF94Ø   ØØ47Ø          LD      HL,(4ØF9H)
7D9D 22FB4Ø   ØØ48Ø          LD      (4ØFBH),HL  ;RESTORE, STACK & END OF BASIC
7DAØ 22FD4Ø   ØØ49Ø          LD      (4ØFDH),HL
7DA3 C3191A   ØØ5ØØ          JP      1A19H   ;RETURN TO BASIC
              ØØ51Ø ;**************************************************
              ØØ52Ø ;*                INPKB
              ØØ53Ø ;*  1.DISPLAY MESSAGE TEXT
              ØØ54Ø ;*  2.INPUT UP TO 5 ASCII CHARACTERS
              ØØ55Ø ;*  3.CONVERT THE NUMBER TO BINARY
              ØØ56Ø ;*  4.RETURN # IN HL-REG PAIR
              ØØ57Ø ;*   INTERNAL CALLS-DISPLY
              ØØ58Ø ;*   EXTERNAL CALLS-ØE6C-CONVERTS ASCII TO BINARY
              ØØ59Ø ;*   ØAFB-CONVERTS SINGLE PRECISION TO INTEGER
              ØØ6ØØ ;*   ØØ4Ø-INPUT FROM KEYBOARD
              ØØ61Ø ;**************************************************
7DA6 CDD47D   ØØ62Ø INPKB    CALL    DISPLY  ;DISPLAY TEXT
7DA9 Ø6Ø5     ØØ63Ø          LD      B,Ø5    ;INPUT 5 ASCII CHARACTERS
7DAB 21B17F   ØØ64Ø          LD      HL,KBUFF
7DAE CD4ØØØ   ØØ65Ø          CALL    ØØ4ØH
7DB1 21BØ7F   ØØ66Ø          LD      HL,KBUFF-1
7DB4 D7       ØØ67Ø          RST     1ØH     ;TEST INPUT
7DB5 3ØA5     ØØ68Ø          JR      NC,BEGIN1
7DB7 21B17F   ØØ69Ø          LD      HL,KBUFF
7DBA CD6CØE   ØØ7ØØ          CALL    ØE6CH   ;CONVERT IT TO BINARY
7DBD 3AAF4Ø   ØØ71Ø          LD      A,(4ØAFH)
7DCØ FEØ2     ØØ72Ø          CP      2       ;TEST FOR INTEGER
7DC2 28ØC     ØØ73Ø          JR      Z,INPK1   ;JUMP IF IT IS
7DC4 FEØ4     ØØ74Ø          CP      4       ;TEST FOR SINGLE PRECISION
7DC6 2Ø94     ØØ75Ø          JR      NZ,BEGIN1       ;START OVER IF NOT
7DC8 3A2441   ØØ76Ø          LD      A,(4124H)
7DCB CDFBØA   ØØ77Ø          CALL    ØAFBH   ;CONVERT TO INTEGER
7DCE EB       ØØ78Ø          EX      DE,HL
7DCF C9       ØØ79Ø          RET
7DDØ 2A2141   ØØ8ØØ INPK1    LD      HL,(4121H)       ;RETURN WITH #IN HL
7DD3 C9       ØØ81Ø          RET
              ØØ82Ø ;**********************************************
              ØØ83Ø ;*               DISPLY
              ØØ84Ø ;*  1. SEND MESSAGE TEXT TO DISPLAY
              ØØ85Ø ;*  2. STOP WHEN RETURN IS ENCOUNTERED
              ØØ86Ø ;*  EXTERNAL CALLS-ØØ33-DISPLAY A CHARACTER
              ØØ87Ø ;*  ON THE SCREEN
              ØØ88Ø ;**********************************************
7DD4 7E       ØØ89Ø DISPLY   LD      A,(HL)
7DD5 23       ØØ9ØØ          INC     HL
7DD6 FEØD     ØØ91Ø          CP      ØDH      ;NEXT CHARACTER A RET?
7DD8 CAEØ7D   ØØ92Ø          JP      Z,DISPL1        ;JUMP IF IT IS
7DDB CD33ØØ   ØØ93Ø          CALL    ØØ33H   ;DISPLAY CHARACTER
7DDE 18F4     ØØ94Ø          JR      DISPLY
7DEØ CD33ØØ   ØØ95Ø DISPL1   CALL    ØØ33H   ;SEND RETURN AND RETURN
7DE3 C9       ØØ96Ø          RET
              ØØ97Ø ;****************************************
              ØØ98Ø ;*              NEWLN
              ØØ99Ø ;*  1. CHECK LINE FOR REFERENCE
              Ø1ØØØ ;*  2. CALCULATE NEW REFERENCE
              Ø1Ø1Ø ;*  3. INSERT NEW REFERENCE
              Ø1Ø2Ø ;*  4. CONTINUE UNTIL ALL LINES CHECKED
              Ø1Ø3Ø ;*  INTERNAL CALLS-SUBTT
              Ø1Ø4Ø ;*  EXTERNAL CALLS-1E5A-CONVERTS ASCII TO BINARY
              Ø1Ø5Ø ;****************************************
```

Program continued

```
7DE4 22B97F    NEWLN   LD      (LNSTRT),HL       ;SAVE ADDR. OF LINE
7DE7 5E                LD      E,(HL)
7DE8 23                INC     HL
7DE9 56                LD      D,(HL)
7DEA 23                INC     HL
7DEB ED53A17F          LD      (NXTLN),DE        ;SAVE POINTER TO NEXT LINE
7DEF 5E                LD      E,(HL)
7DFØ 23                INC     HL
7DF1 56                LD      D,(HL)
7DF2 ED53A37F          LD      (LNNUM),DE        ;SAVE LINE #
7DF6 23        GETCHR  INC     HL      ;POINT TO NEXT CHARACTER
7DF7 7E        GETCH1  LD      A,(HL)
7DF8 A7                AND     A
7DF9 2816              JR      Z,ENDTST          ;TEST FOR END OF LINE
7DFB FE8D              CP      141
7DFD 2827              JR      Z,GBNCH      ;TEST FOR 'GOTO'
7DFF FE91              CP      145
7EØ1 2823              JR      Z,GBNCH     ;TEST FOR 'GOSUB'
7EØ3 FE95              CP      149
7EØ5 281F              JR      Z,GBNCH     ;TEST FOR 'ELSE'
7EØ7 FECA              CP      2Ø2
7EØ9 281B              JR      Z,GBNCH     ;TEST FOR 'THEN'
7EØB FEA1              CP      161
7EØD 282A              JR      Z,ONBCH     ;TEST FOR 'ON'
7EØF 18E5              JR      GETCHR
7E11 23        ENDTST  INC     HL
7E12 7E                LD      A,(HL)   ;TEST NEXT CHARACTER FOR ZERO
7E13 A7                AND     A
7E14 2ØØ7              JR      NZ,NTLN1          ;JUMP IF NOT ZERO
7E16 23                INC     HL
7E17 7E                LD      A,(HL)   ;TEST FOR SECOND ZERO
7E18 A7                AND     A
7E19 2ØØ1              JR      NZ,NTLNE          ;JUMP IF NOT ZERO
7E1B C9                RET
7E1C 2B        NTLNE   DEC     HL
7E1D ED5BA17F  NTLN1   LD      DE,(NXTLN)        ;ADDR. = NEXT LINE ADDR.
7E21 DF                RST     18H
7E22 2ØD3              JR      NZ,GETCH1   ;IF NOT, GO TEST NEXT CHARACTER
7E24 18BE              JR      NEWLN   ;IF YES, GET NEW LINE
7E26 D7        GBNCH   RST     1ØH
7E27 28E8              JR      Z,ENDTST          ;TEST FOR END OF LINE
7E29 3ØCC              JR      NC,GETCH1         ;JUMP IF NOT NUMERIC
7E2B E5                PUSH    HL      ;FOUND REFERENCE
7E2C CD5A1E            CALL    1E5AH   ;CONVERT TO BINARY
7E2F E1                POP     HL
7E3Ø ED53A57F          LD      (LNREF),DE        ;SAVE LINE IN LNREF
7E34 CD5E7E            CALL    SUBTT   ;GO SUBSTITUTE NEW REFERENCE
7E37 18BE              JR      GETCH1  ;GET NEXT CHARACTER
7E39 D7        ONBCH   RST     1ØH     ;FOUND 'ON' REFERENCE
7E3A 28D5              JR      Z,ENDTST
7E3C FE8D              CP      141     ;TEST FOR GOTO
7E3E 28Ø4              JR      Z,ONBC1
7E4Ø FE91              CP      145     ;TEST FOR GOSUB
7E42 2ØF5              JR      NZ,ONBCH
7E44 D7        ONBC1   RST     1ØH
7E45 28CA              JR      Z,ENDTST          ;TEST FOR END OF LINE
7E47 3ØØE              JR      NC,ONBC2          ;JUMP IF NOT NUMERIC
7E49 E5                PUSH    HL
7E4A CD5A1E            CALL    1E5AH   ;CONVERT TO BINARY
7E4D E1                POP     HL
7E4E ED53A57F          LD      (LNREF),DE        ;SAVE LINE IN LNREF
```

```
7E52 CD5E7E             CALL    SUBTT   ;SUBSTITUTE NEW REFERENCE
7E55 18ED               JR      ONBC1
7E57 FE2C       ONBC2   CP      2CH     ;TEST FOR COMMA
7E59 C2F77D             JP      NZ,GETCH1       ;IF NOT GET NEXT CHARACTER
7E5C 18E6               JR      ONBC1   ;OTHERWISE, CONTINUE
                ;*****************************************
                ;*              SUBTT
                ;*  1. SEARCH FOR LINE THAT IS REFERENCED
                ;*  2. CALCULATE THE NEW LINE #
                ;*  3. SUBSTITUTE NEW REFERENCE INTO LINE
                ;*  INTERNAL CALLS-NCHK,FIXCHR
                ;*  EXTERNAL CALLS-132F-CONVERT BINARY TO ASCII
                ;*  1AFC-CALCULATE NEW LINE ADDRESSES
                ;*****************************************
7E5E 229F7F     SUBTT   LD      (PRSPS),HL      ;SAVE THE PRESENT POSITION
7E61 01ØØØØ             LD      BC,ØØØØ
7E64 2A9D7F             LD      HL,(STRBSC)
7E67 5E         SUBT1   LD      E,(HL)
7E68 23                 INC     HL
7E69 56                 LD      D,(HL)
7E6A D5                 PUSH    DE      ;DE = ADDR. OF NEXT LINE
7E6B 7B                 LD      A,E
7E6C B2                 OR      D
7E6D CA3F7F             JP      Z,SUEXT     ;IF DE = ZERO, WE ARE DONE
7E7Ø 23                 INC     HL
7E71 5E                 LD      E,(HL)
7E72 23                 INC     HL
7E73 56                 LD      D,(HL)      ;DE = LINE #
7E74 2AA57F             LD      HL,(LNREF)      ;HL = LINE WE ARE WORKING ON
7E77 DF                 RST     18H
7E78 28Ø4               JR      Z,SUBT2     ;JUMP IF THEY ARE EQUAL
7E7A E1                 POP     HL
7E7B Ø3                 INC     BC      ;ELSE GO TO NEXT LINE AND
7E7C 18E9               JR      SUBT1   ;INCREMENT BC
7E7E D1         SUBT2   POP     DE
7E7F 11ØØØØ             LD      DE,ØØØØ
7E82 79                 LD      A,C     ;CALCULATE NEW REFERENCE
7E83 BØ                 OR      B
7E84 28Ø4               JR      Z,SUBT3
7E86 2A9B7F             LD      HL,(INCR)   ;NEW REF=BC*(INCR)+START
7E89 EB                 EX      DE,HL
7E8A 2A997F     SUBT3   LD      HL,(START)
7E8D 19         SUBT4   ADD     HL,DE
7E8E 79                 LD      A,C
7E8F BØ                 OR      B
7E9Ø 28Ø5               JR      Z,SUBT5
7E92 ØB                 DEC     BC
7E93 79                 LD      A,C
7E94 BØ                 OR      B
7E95 2ØF6               JR      NZ,SUBT4
7E97 22A97F     SUBT5   LD      (NWLNM),HL  ;SAVE NEW REFERENCE IN NWLNM
7E9A 222141             LD      (4121H),HL
7E9D 21AB7F             LD      HL,BUFF
7EAØ Ø1Ø7Ø7             LD      BC,Ø7Ø7H    ;CONVERT NEW REF TO ASCII
7EA3 CD2F13             CALL    132FH
7EA6 2A9F7F             LD      HL,(PRSPS)
7EA9 CD447F             CALL    NCHK    ;CALCULATE # OF DIGITS IN OLD REF
7EAC 79                 LD      A,C
7EAD 32A77F             LD      (CHRNO),A   ;CHRNO=# OF DIGITS IN OLD REF
7EBØ CD4C7F             CALL    FIXCHR      ;FIX # TO PROPER FORMAT
7EB3 79                 LD      A,C
```

Program continued

```
7EB4 32A87F            LD      (CHRNN),A     ;CHRNN=# DIGITS IN NEW REF
7EB7 3AA77F            LD      A,(CHRNO)
7EBA 91                SUB     C
7EBB 2833              JR      Z,SUBT8     ;JUMP IF CHRNO=CHRNN
7EBD F20C7F            JP      P,SUBT7     ;JUMP IF CHRNO > CHRNN
7ECØ F5         SUBT6  PUSH    AF           ;SAVE THE DIFFERENCE
7EC1 A7                AND     A            ;CLEAR THE CARRY
7EC2 2AF94Ø            LD      HL,(4ØF9H)
7EC5 E5                PUSH    HL
7EC6 ED4B9F7F          LD      BC,(PRSPS)
7ECA ED42              SBC     HL,BC
7ECC E5                PUSH    HL
7ECD C1                POP     BC      ;BC=TOTAL # OF BYTES TO BE MOVED
7ECE E1                POP     HL
7ECF E5                PUSH    HL      ;HL=END OF BASIC ADDR.
7EDØ D1                POP     DE
7ED1 13                INC     DE      ;DE=END OF BASIC+1
7ED2 EDB8              LDDR            ;MOVE MEMORY FOR 1 NEW CHARACTER
7ED4 2AF94Ø            LD      HL,(4ØF9H)
7ED7 23                INC     HL
7ED8 22F94Ø            LD      (4ØF9H),HL  ;FIX END OF BASIC POINTER
7EDB 2AA17F            LD      HL,(NXTLN)
7EDE 23                INC     HL
7EDF 22A17F            LD      (NXTLN),HL  ;FIX NEXT LINE POINTER
7EE2 EB                EX      DE,HL
7EE3 2AB97F            LD      HL,(LNSTRT)
7EE6 73                LD      (HL),E
7EE7 23                INC     HL
7EE8 72                LD      (HL),D
7EE9 CDFC1A            CALL    1AFCH   ;UPDATE ALL NEXT LINE POINTERS
7EEC F1                POP     AF
7EED 3C                INC     A       ;JUMP IF MORE CHARACTERS EXIST
7EEE 2ØDØ              JR      NZ,SUBT6
7EFØ 3AA87F     SUBT8  LD      A,(CHRNN)
7EF3 4F                LD      C,A
7EF4 Ø6ØØ              LD      B,Ø
7EF6 21AB7F            LD      HL,BUFF
7EF9 ED5B9F7F          LD      DE,(PRSPS)  ;MOVE NEW REFERENCE TO
7EFD EDBØ              LDIR                ;PROGRAM LINE
7EFF 3AA87F     SUBT9  LD      A,(CHRNN)
7FØ2 2A9F7F            LD      HL,(PRSPS)
7FØ5 85                ADD     A,L
7FØ6 6F                LD      L,A
7FØ7 3EØØ              LD      A,Ø
7FØ9 8C                ADC     A,H
7FØA 67                LD      H,A     ;POINT TO NEXT CHARACTER
7FØB C9                RET             ;AND RETURN
7FØC F5         SUBT7  PUSH    AF      ;SAVE THE DIFFERENCE
7FØD A7                AND     A
7FØE 2AF94Ø            LD      HL,(4ØF9H)
7F11 ED4B9F7F          LD      BC,(PRSPS)
7F15 ED42              SBC     HL,BC
7F17 E5                PUSH    HL
7F18 C1                POP     BC      ;BC=TOTAL # OF BYTES TO BE MOVED
7F19 2A9F7F            LD      HL,(PRSPS)
7F1C E5                PUSH    HL      ;HL=PRESENT POSITION
7F1D D1                POP     DE      ;DE=PRESENT POSITION
7F1E 23                INC     HL
7F1F EDBØ              LDIR            ;MOVE MEMORY FOR 1 DIGIT
7F21 2AF94Ø            LD      HL,(4ØF9H)     ;ADJUST END OF BASIC POINTER
7F24 2B                DEC     HL
```

```
7F25 22F94Ø                LD      (4ØF9H),HL
7F28 2AA17F                LD      HL,(NXTLN)
7F2B 2B                    DEC     HL
7F2C 22A17F                LD      (NXTLN),HL       ;ADJUST NEXT LINE POINTER
7F2F EB                    EX      DE,HL
7F3Ø 2AB97F                LD      HL,(LNSTRT)
7F33 73                    LD      (HL),E
7F34 23                    INC     HL
7F35 72                    LD      (HL),D
7F36 CDFC1A                CALL    1AFCH       ;UPDATE ALL NEXT LINE POINTERS
7F39 F1                    POP     AF
7F3A 3D                    DEC     A
7F3B 2ØCF                  JR      NZ,SUBT7         ;JUMP IF MORE DIGITS EXIST
7F3D 18B1                  JR      SUBT8     ;GO PUT NEW REFERENCE IN LINE
7F3F D1             SUEXT  POP     DE
7F4Ø 2A9F7F                LD      HL,(PRSPS)       ;EXIT SUBSTITITE MODE
7F43 C9                    RET
                    ;****************************************
                    ;*             NCHK
                    ;*  1. COUNTS THE NUMBER OF ASCII
                    ;*  CHARACTERS POINTED TO BY HL-REGISTER
                    ;****************************************
7F44 ØEØØ           NCHK   LD      C,Ø
7F46 2B                    DEC     HL
7F47 D7             NCH2   RST     1ØH      ;TEST CHARACTER
7F48 DØ                    RET     NC       ;RETURN IF NOT NUMERIC
7F49 ØC                    INC     C        ;INCREMENT COUNT
7F4A 18FB                  JR      NCH2     ;CONTINUE
                    ;****************************************
                    ;*             FIXCHR
                    ;*  1. FIXES ASCII # SO THAT IT IS IN
                    ;*  THE CORRECT FORM FOR USE IN SUBTT
                    ;****************************************
7F4C ØEØØ           FIXCHR LD      C,Ø
7F4E 21AB7F                LD      HL,BUFF
7F51 2B             FIXCH1 DEC     HL
7F52 D7                    RST     1ØH      ;TEST CHARACTER
7F53 DØ                    RET     NC       ;RETURN IF NOT NUMERIC
7F54 FE3Ø                  CP      3ØH
7F56 28Ø5                  JR      Z,FIXCH3         ;JUMP IF IT IS AN ASCII ZERO
7F58 ØC             FIXCH2 INC     C
7F59 D7                    RST     1ØH      ;COUNT UNTIL NON-NUMERIC
7F5A DØ                    RET     NC       ;CHARACTER IS FOUND
7F5B 18FB                  JR      FIXCH2
7F5D E5             FIXCH3 PUSH    HL       ;ROTATE NUMBER LEFT TO
7F5E C5                    PUSH    BC       ;DELETE LEADING ZEROS
7F5F D5                    PUSH    DE
7F6Ø Ø1Ø5ØØ                LD      BC,ØØØ5
7F63 E5                    PUSH    HL
7F64 D1                    POP     DE
7F65 23                    INC     HL
7F66 EDBØ                  LDIR
7F68 D1                    POP     DE
7F69 C1                    POP     BC
7F6A E1                    POP     HL
7F6B 18E4                  JR      FIXCH1     ;GET NEXT CHARACTER
                    ;****************************************
                    ;*             UPDATE
                    ;*  1. UPDATES THE ACTUAL LINE NUMBERS
                    ;*
                    ;****************************************
7F6D 2A9B7F         UPDATE LD      HL,(INCR)
```

Program continued

```
7F70 EB         03510          EX      DE,HL
7F71 2A997F     03520          LD      HL,(START)      ;UPDATE ALL LINE #S
7F74 A7         03530          AND     A
7F75 ED52       03540          SBC     HL,DE
7F77 22B77F     03550          LD      (TOTAL),HL      ;TOTAL = START - INCR
7F7A 2A9D7F     03560          LD      HL,(STRBSC)
7F7D 5E         03570 UPDAT1   LD      E,(HL)     ;FIND NEXT LINE ADDR.
7F7E 23         03580          INC     HL
7F7F 56         03590          LD      D,(HL)
7F80 7B         03600          LD      A,E
7F81 B2         03610          OR      D
7F82 C8         03620          RET     Z        ;RETURN IF DONE
7F83 D5         03630          PUSH    DE
7F84 23         03640          INC     HL
7F85 E5         03650          PUSH    HL
7F86 2A9B7F     03660          LD      HL,(INCR)       ;CALCULATE LINE #
7F89 EB         03670          EX      DE,HL
7F8A 2AB77F     03680          LD      HL,(TOTAL)      ;TOTAL = TOTAL + INCR
7F8D 19         03690          ADD     HL,DE
7F8E 22B77F     03700          LD      (TOTAL),HL
7F91 EB         03710          EX      DE,HL
7F92 E1         03720          POP     HL
7F93 73         03730          LD      (HL),E      ;LINE # = TOTAL
7F94 23         03740          INC     HL
7F95 72         03750          LD      (HL),D
7F96 E1         03760          POP     HL       ;POINT TO NEXT LINE
7F97 18E4       03770          JR      UPDAT1      ;CONTINUE
0002            03780 START    DEFS    2        ;STARTING LINE #
0002            03790 INCR     DEFS    2        ;INCREMENT
0002            03800 STRBSC   DEFS    2        ;ADDR. OF START OF BASIC PROG.
0002            03810 PRSPS    DEFS    2        ;PRESENT POSITION
0002            03820 NXTLN    DEFS    2        ;NEXT LINE
0002            03830 LNNUM    DEFS    2        ;LINE NUMBER
0002            03840 LNREF    DEFS    2        ;LINE REFERENCE
0001            03850 CHRNO    DEFS    1        ;# OF DIGITS IN OLD REFERENCE
0001            03860 CHRNN    DEFS    1        ;# OF DIGITS IN NEW REFERENCE
0002            03870 NWLNM    DEFS    2        ;NEW LINE REFERENCE
0004            03880 BUFF     DEFS    4
7FAF 0000       03890          DEFW    00
0004            03900 KBUFF    DEFS    4
7FB5 0000       03910          DEFW    00
0002            03920 TOTAL    DEFS    2
0002            03930 LNSTRT   DEFS    2        ;STARTING LINE
7D5D            03940          END     BEGIN
00000 TOTAL ERRORS
```

21

Complete Variable Lister

by John M. Hammang

System Requirements:
Level II BASIC
Model I
16K RAM

With John Webster's variable lister article (*80 Microcomputing*, July 1981), I could finally document all my TRS-80 programs. Unfortunately, Mr. Webster's program fails to list the array variables. My modification of it does.

The key to listing these variables is to jump from array name to array name without stumbling over the intervening data and string pointers. The first array name immediately follows the last simple variable name.

Each array describes its contents with the following format. The first byte indicates variable type (2 = integer, 3 = string, 4 = single precision, and 8 = double precision). The next two bytes are the ASCII values of the variable name in reverse order. For instance, if your variable is named AB, the B appears first (66) and the A appears second (65). The HL register in the Z80 microprocessor reverses them when it retrieves the values for processing. The fourth and fifth bytes specify the total number of bytes which occur after the fifth byte. These two values, added together, determine the size of the array and are also reversed. The value in the fourth byte is multiplied by one. This is the least significant byte (LSB). The value in the fifth byte is multiplied by 256 (the decimal equivalent of the third place value of numbers in the hexadecimal numbering system). This fifth byte is the most significant byte (MSB). When you add the fourth and fifth bytes you know how far to jump, in decimal form, to the next array variable name.

This gives you the basic array names. The sixth byte tells how many dimensions this array has and each following pair of bytes tells the size or depth of each dimension in the array. Thus you can determine and report not only the number of dimensions for each array but the size of each dimension.

The dimension size byte pairs are reversed for normal Z80 operations. The sequence of dimension sizes is also stored in reverse order. For example, if you dimension AB(3,4,5), Level II will store the third size (5) first, the second size (4) next, and the first size (3) last. The Complete Variable Lister also reports the dimension sizes in reverse order.

The first part of the program listing approximates Mr. Webster's. Line 65000 finds the start of the simple variable storage area and dimensions a terminating test array. Line 65010 establishes the variable name values and checks for variable name ZV (the first variable name used in this routine). The first time this variable name is encountered, ZU is incremented and the program flows to the array listing routines. For an explanation of how the first segment of this program works, see page 259 of the July 1981 issue of *80 Microcomputing*.

When you find the ZV variable name the first time, 35 is added to the PEEK pointer value (the value of ZV + 35), to jump over the simple variable names in this program. I use five single precision variables (5*7 = 35). As in the first segment, the program prints the type of data in the array and the variable name. Line 65120 reports the number of dimensions (PEEK(ZV + 5)). Next, the program enters a FOR-NEXT loop to report the size of each dimension. Dimension size is computed (LSB*1) + (MSB*256) and printed in reverse order. Line 65140 computes the jump to the next array variable name and returns to the start of the reporting routine in line 65020. Five is added to the computed jump value in line 65140 because the array size, stored in bytes 4 and 5 of the array variable, reports only the number of bytes following the array size information.

This program uses very high line numbers so you can add this Complete Variable Lister to existing programs without renumbering. It also uses unusual variable names to avoid conflicts with variable names utilized in object programs.

1. CLOAD the object program
2. PRINT PEEK(16633),PEEK(16634) and write down the results
3. If PEEK(16633)≥2 then go to step 5
4. If PEEK(16633)<2 then POKE 16548, PEEK(16633) + 254: POKE16549,PEEK(16634) - 1 then go to step 6
5. POKE 16548,PEEK(16633) - 2: POKE 16549,PEEK(16634)
6. CLOAD the Complete Variable Lister program
7. POKE 16548,233: POKE 16549,66
8. Run the object program with all its subroutines and variations
9. BREAK and GOTO 65000 to execute the Complete Variable Lister

Figure 1. *How to merge the Complete Variable Lister with your object program*

Use the procedure listed in Figure 1 to add this program to the program for which you want the list of variables. Complete step 7 before listing the combined programs. Step 7 is a call to the Level II merge function. You have to run the object program with all its subroutines and variations to find all the variables.

The Level II manual erroneously states that the number and size of dimensions are limited only by the amount of memory available. The data pattern for array names and sizes limits the number of dimensions to 255 and a dimension's depth to 32,768.

The Complete Variable Lister program reports all simple and array variables with any type of data as well as the number and size of each array dimension. All arrays are reported whether directly dimensioned in the program or dimensioned by default.

Program Listing

```
64950 '*** COMPLETE VARIABLE LISTER ***'
64960 'BY: JOHN M. HAMMANG
64970 '21730 CHIPMUNK TRAIL EAST
64980 'WOODHAVEN, MICHIGAN 48183
64990 'VERSION 1.0
65000 ZV=PEEK(16633)+256*PEEK(16634):DIMZV$(1):ZU=1:LPRINT"    *
* * SIMPLE VARIABLES * * *":LPRINTCHR$(138)
65010 ZW=PEEK(ZV):ZX=PEEK(ZV+1):ZY=PEEK(ZV+2):IFZX=86ANDZY=90THE
NZU=ZU+1:GOSUB65160:GOTO65010
65020 LPRINTCHR$(ZY);CHR$(ZX);:GOSUB65030:LPRINT" ":LPRINTCHR$(1
38):GOTO65010
65030 IFZW=2THENLPRINT" % INTEGER";:GOTO65070
65040 IFZW=3THENLPRINT"$";:GOTO65070
65050 IFZW=4THENLPRINT" ! SINGLE PRECISION";:GOTO65070
65060 IFZW=8THENLPRINT" # DOUBLE PRECISION";:GOTO65070
65070 ONZUGOTO65080,65120
65080 IFZW=2THENZV=ZV+5:RETURN
65090 IFZW=3THENLPRINTPEEK(ZV+3);"CHAR. LONG";:ZV=ZV+6:RETURN
65100 IFZW=4THENZV=ZV+7:RETURN
65110 IFZW=8THENZV=ZV+11:RETURN
65120 LPRINT" ARRAY WITH";PEEK(ZV+5);"DIMENSION(S)":ZW=ZV+5
65130 FORZX=PEEK(ZV+5)TO1STEP-1:LPRINTTAB(5)"DIM SIZE NO.";ZX;"I
S";(PEEK(ZW+1)+(PEEK(ZW+2)*256))-1;"ELEMENTS":ZW=ZW+2:NEXTZX
65140 ZV=ZV+(PEEK(ZV+3)+(PEEK(ZV+4)*256))+5:RETURN
65160 IFZU=2THENZV=ZV+35:LPRINT" ":LPRINT"   * * * ARRAY VARIABL
ES * * *":RETURN
65170 IFZU=3THENLPRINT" ":LPRINTCHR$(138):LPRINT"ALL VARIABLES L
ISTED":END
```

22

MODSTRING for Packing Strings

by T.A. Wells

System Requirements:
Model I
16K RAM
Level II BASIC

MODSTRING, a utility written in BASIC, can be merged with another BASIC program. It enables you to pack strings for faster graphics or embed machine language in a BASIC program. With MODSTRING, you can pack, modify, and display strings wthout typing and later deleting FOR-NEXT loops, DATA statements, and POKE statements.

MODSTRING works by modifying itself. When you type the name of the string to be packed or modified, MODSTRING POKEs that name into one of its own lines, so the VARPTR and LEN functions can be used. After the information is obtained, MODSTRING unmodifies itself, to allow another string to be processed.

How to Use MODSTRING

If you have a disk system, it's easy. First load your program, in which you have put dummy strings. Make sure you don't use line numbers higher than 59999. Then merge MODSTRING with it. If you do not have a disk system, you will need a separate utility program that allows the merging of two BASIC programs.

After merging, run your program. When it has passed through all lines containing dummy strings, BREAK the program. This assures that the BASIC interpreter has positioned your string variables in memory just above the end of the program.

Type GOTO 60000. You are prompted from here on by MODSTRING. Type the name of the string. The ENTER key is not needed in any part of MODSTRING. Answer the other prompts and your string should print, or list on the screen. You must list the string to modify it. If you have printed it, press L. The listed string is in the form BYTE#>VALUE. When you press M, you are asked for the number of the byte you wish to modify. Three digits must be pressed. For example, to change byte 7, press 007. You are then asked for a value from 1 to 255 (except 34, the ASCII value for a quotation mark).

Here is an example to try. Load MODSTRING and type the following lines:

```
100 A$="1234567890"
200 EX$(3)="12345678901234567890"
```

Run the program. When the first prompt appears, press the space bar, then A. Answer the subscript quesion with N. Press P to print the length of the string (10). Below that, 1234567890 should appear. Now press L. The contents of the string (A$) should appear in the BYTE#>VALUE format. Press M; to modify the first byte, press 001. Select a value, say 191, and type it. This value appears next to the 1> near the upper left of the screen. Now press P. The string starts with the graphics block.

For fun, press Q, then type LIST 100. The line is now: 100 A$="USING234567890". The value 191 is the computer's code for the USING keyword. When you print the string, the graphics character is always printed. Now run MODSTRING again. List and modify EX$(3) in the same way, but press Y in response to the subscript prompt, then press 3.

Program Limitations

There are some restrictions which must be observed when using this program. Don't try to modify the program unless you understand it

Variable Name	Use
PQ%	Loop counter
PS%, PS!	Pointer to end of BASIC program
P1$, P2$, P3$, P4$	INKEY$ variables to input string name and choice of process
P5$, P5%	New value for byte to be modified
PZ!, PZ%	Location of string in memory
PP$	Name of string to be processed
PL%	Length of string being processed
PA%	Length of string being processed, minus one
PP!, PP%	Pointer to location of string in memory
PI%	Counter for byte number of string being processed
P1%	Counter variable
P4%	Byte position in string

Figure 1. *Variables*

completely. If you do modify it, or just type it in, save a copy before you run it.

String length can't be changed by MODSTRING. You should decide beforehand how long your strings must be. The longest string possible is 104 characters. If you like a challenge, try increasing the maximum length to 240 characters or so.

Variable names in your program must be different from those I have used (see Figure 1). All variables in MODSTRING have type declarations such as a percent sign (%) or exclamation point (!). This avoids confusion with variables in your program or such statements as DEFINTA-Z, DEFDBL or DEFSTR, which you are free to use. When in doubt, check the list of variables.

Do not manipulate any string that is or will be packed.

This program has only been verified on Radio Shack hardware tape and disk (with TRSDOS 2.3) systems. Any other hardware or operating systems that change the end-of-program pointers (16633,4) will produce problems.

Line-by-Line Program Description

60000 Ask for name of string variable to be processed.
60005 Subroutine to replace PP$ in the LEN and VARPTR statements in line 62000.
60009-60010 Get first letter of string variable name.
60020 Get second letter of string variable name. Put first plus second letters into PP$.
60022-60027 Ask if the string variable is subscripted. If so, get the value of the subscript and put the whole variable name into PP$.
60030-60040 Print or list option.
60060 Initialize variables. This keeps ROM from moving them around in later steps.
60100 Go to the heart of the program.
60200-60201 Set up pointers for the PEEK statements to follow.
60202 Reject string variables with LEN>104 bytes.
60208-60232 Print or list the contents of the variable.
60240-60250 Ask operator for next instruction.
60260-60320 If modify is selected, modify the screen display and the variable.
60330 Go back for further instructions.
61000 Find the end of the BASIC program. Memory locations 16633 and 16634 contain the pointer.
61220 STEP from the end of the program toward the front looking for "PP$" (in ASCII, PP$ is 80, 80, and 36). When found, exit to the next statement.
61900 POKE the name of the variable sought into the next BASIC

line, replacing the two occurrences of PP$ in line 62000 with the name of the variable the operator has specified. One is the argument of the LEN statement at the beginning of the line and the other is the argument of the VARPTR statement near the end. The blanks in line 62000 following each occurrence of PP$ are significant. They provide room for POKEing the name of a subscripted variable.

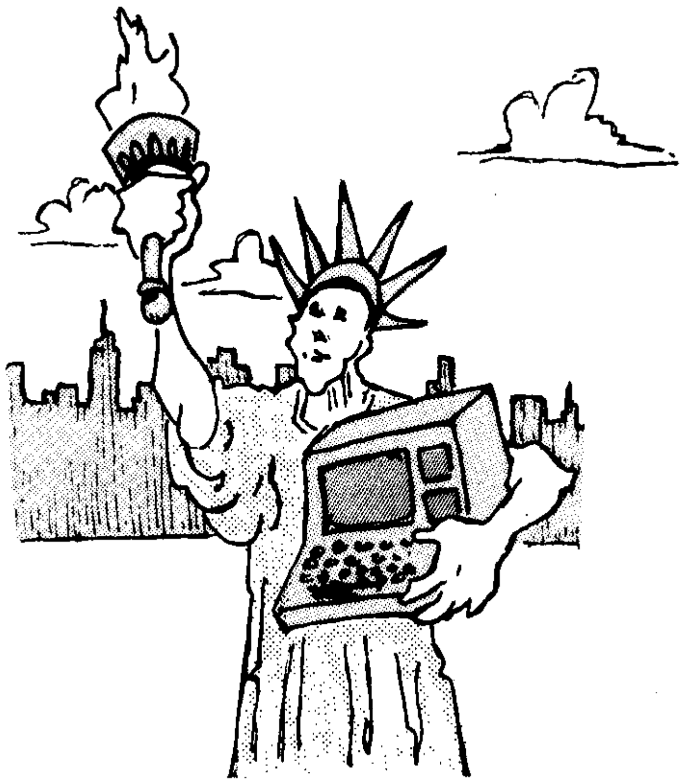
62000 Find the length of the requested function. If it's less than one, put the PP$ back into line 62000, print an error message and go get another name. If the length is okay, get PP!, the pointer to the variable in memory. Put the PP$ back into line 62000, then go process the variable (print, list or modify).

Program Listing

```
59998 REM          ** MODSTRING **
59999 REM - STRING MODIFICATION PROGRAM - COPYRIGHT 1981
            T. A. WELLS - N.O., LA. 70118
60000 CLS:PRINT"STRING MODIFICATION PROGRAM":PRINT:PRINT"ENTER T
HE NAME OF THE STRING USING ONLY ONE OF THESE 3 FORMATS:
<SPACE><LETTER> - <LETTER><LETTER> - <LETTER><NUMBER>
         ----> ? ";:GOTO60009
60005 FORPQ%=1TO2:POKEPS%-3+PQ%,80:POKEPS%-124+PQ%,80:NEXT:POKEP
S%,36:POKEPS%-121,36:FORPQ%=4TO6:POKEPS%-3+PQ%,32:POKEPS%-124+PQ
%,32:NEXT:RETURN
60009 P1$=INKEY$
60010 P1$=INKEY$:IF(P1$<"A"ORP1$>"Z")ANDP1$<>" "THEN60010ELSEPRI
NTP1$;:P2$=INKEY$
60020 P2$=INKEY$:IF(P2$>"/"ANDP2$<":"ANDP1$<>" ")OR(P2$>"@"ANDP2
$<"[")THENPP$=P1$+P2$+"$":PRINTP2$"$"ELSE60020
60022 PRINT"IS "PP$" SUBSCRIPTED (Y/N)?":P1$=INKEY$
60024 P1$=INKEY$:IFP1$="N"THENPP$=PP$+"   ":GOTO60030ELSEIFP1$="
Y"THENPP$=PP$+"("ELSE60024
60026 PRINT"VALUE OF SUBSCRIPT (0-9)?";:P1$=INKEY$
60027 P1$=INKEY$:IFP1$<"0"ORP1$>"9"THEN60027ELSEPP$=PP$+P1$+")"
60030 PRINT@0,CHR$(31)"<P>RINT "PP$" -OR- <L>IST ITS CONTENTS?"
60040 P1$=INKEY$:IFNOT(P1$="P"ORP1$="L")THEN60040
60060 PL%=0:PP%=0:PP!=0:PS%=0:PS!=0:PA%=0:PI%=0:PZ!=0:PZ%=0:P3$=
"":P4$="":P5$="":P1%=0:P4%=0:P5%=0:PQ%=0:PQ$=""
60100 GOTO61000
60200 PA%=PL%-1:PI%=1:IFPP!>32767THENPP%=65536-PP!ELSEPP%=PP!
60201 PZ!=PEEK(PP%+1)+256*PEEK(PP%+2):IFPZ!>32767THENPZ%=PZ!-655
36ELSEPZ%=PZ!
60202 IFPL%>104THENPRINTPP$" IS TOO LONG - IT HAS"PL%"CHARACTERS
.

 MAXIMUM IS 104 CHARACTERS.":FORPQ%=1TO1111:NEXTPQ%:GOTO60000
60208 PRINT"LEN("PP$")="PL%
60210 FORPS%=PZ%TOPZ%+PA%
60220 IFP1$="L"THENPRINTUSING"###";PI%;:PRINT">";:PRINTUSING"###
";PEEK(PS%);:PRINT" ";:PI%=PI%+1
60230 IFP1$="P"THENPRINTCHR$(PEEK(PS%));:PI%=PI%+1
60232 NEXT
60240 PRINT@960,"<A>NOTHER STRING  <Q>UIT ";:IFP1$="P"THENPRINT"
```

Program continued

```
 <L>IST ? ";ELSEPRINT"<P>RINT  <M>ODIFY  ? ";
6Ø25Ø P3$=INKEY$:IFP3$="M"ANDP1$="L"THEN6Ø26ØELSEIFP3$="L"ANDP1$
="P"THENCLS:P1$="L":PRINT:GOTO6Ø2ØØELSEIFP3$="P"ANDP1$="L"THENCL
S:PRINT:P1$="P":GOTO6Ø2ØØELSEIFP3$="Q"THENENDELSEIFP3$="A"THEN6Ø
ØØØELSE6Ø25Ø
6Ø26Ø PRINT@96Ø,CHR$(3Ø);"MODIFY WHICH BYTE (ØØ1 TO ";STRING$(4-
LEN(STR$(PL%)),"Ø")RIGHT$(STR$(PL%),LEN(STR$(PL%))-1)") ?";
6Ø27Ø P4$="":FORP1%=1TO3:P3$=""
6Ø28Ø P3$=INKEY$:IFP3$>"9"ORP3$<"Ø"THEN6Ø28ØELSEP4$=P4$+P3$:PRIN
TP3$;:NEXT:P4%=VAL(P4$):IFP4%<1ORP4%>PL%THEN6Ø26Ø
6Ø29Ø PRINT@96Ø,CHR$(3Ø);"NEW VALUE FOR BYTE #"P4%" = ? ";
6Ø3ØØ P5$="":FORP1%=1TO3:P4$=""
6Ø31Ø P4$=INKEY$:IFP4$>"9"ORP4$<"Ø"THEN6Ø31ØELSEP5$=P5$+P4$:PRIN
TP4$;:NEXT:P5%=VAL(P5$):IFP5%=ØORP5%=34ORP5%>255THEN6Ø29Ø
6Ø32Ø POKEPZ%+P4%-1,P5%:FORP1%=ØTO2:POKE1536Ø+132+P1%+8*(P4%-1),
ASC(MID$(P5$,P1%+1,1)):NEXT
6Ø33Ø GOTO6Ø24Ø
61ØØØ PS!=PEEK(16633)+256*PEEK(16634):IFPS!>32767THENPS%=PS!-655
36ELSEPS%=PS!
6122Ø FORPP%=PS%TOPS%-1ØØSTEP-1:IFPEEK(PP%)=36ANDPEEK(PP%-1)=8ØA
NDPEEK(PP%-2)=8ØTHEN619ØØELSENEXT
619ØØ FORPQ%=1TO6:POKEPP%+PQ%-3,ASC(MID$(PP$,PQ%,1)):POKEPP%-124
+PQ%,ASC(MID$(PP$,PQ%,1)):NEXT
62ØØØ PS%=PP%:PL%=LEN(PP$    ):IFPL%<1THENPRINT"ERROR - STRING HA
S TO BE DEFINED ALREADY & HAVE A LENGTH > Ø ...":FORPQ%=1TO167Ø:
NEXTPQ%:GOSUB6ØØØ5:GOTO6ØØØØELSEPP!=VARPTR(PP$    ):GOSUB6ØØØ5:GO
TO6Ø2ØØ
```



23 PASSWORD Utility

by Craig A. Lindley

System Requirements:
Model I
16K RAM
Editor/assembler
One disk drive
TRSDOS

PASSWORD lets you lock or unlock password protection so you can investigate the system, FORMAT/CMD, BACKUP/CMD and BASIC/CMD files. You can also perform many other operations while the system disk in drive 0 has its password protection unlocked. You can copy, kill, print, and list all those files that until now were only entries in your directories.

How the Program Works

PASSWORD patches the SYS2/SYS file. This file is called up as an overlay anytime the operating system tries to open a file for access. In order to open a file, the operating system must calculate a password code (two bytes in length) from the password specified in the disk control block (DCB). Subroutine CALPW (see Program Listing) contains the code to do this. The calculated password code is compared to the password code stored in the directory entry for the specified file. Usually, if the two match, the file opens and control passes back from the open routine to the calling program, with the Z flag set to indicate the operation was completed successfully. If the calculated and stored password codes do not match, the open routine ends, and an error message code passes back to the calling program in the A register, indicating that access to the file was denied. PASSWORD defeats this process by modifying the code contained in the SYS2/SYS file so that the calculated and stored password codes always appear to match. This is done by changing the relative jump-if-zero instruction to an unconditional jump instruction. The lock procedure replaces the jump-if-zero instruction, restoring password protection.

How to Use PASSWORD

Execute the program by typing PASSWORD while in the DOS command mode. After the sign-on message is displayed, you are prompted to indicate which disk in the system (0–3) to alter. The specified disk must have the SYS2/SYS file resident (it must be a system disk). If not, PASSWORD displays an error message.

The program prompts for the disk master password before performing the unlock/lock operation. If the password you supply is incorrect, the program ends.

It doesn't matter whether the disk in drive 0 is locked. If you enter the correct password, the area normally occupied by the SYS2/SYS file is loaded into memory from the disk drive specified. PASSWORD checks *the format of the data to make sure the disk is a system disk. If it isn't, an* error message is displayed, and the program ends. If the data format is correct, the program tells whether the specified disk is locked or unlocked and displays a menu showing the following three options:

L—Lock selected disk
U—Unlock selected disk
Q—Quit this program

The lock option restores password protection afforded by the operating system. The unlock option removes it. These functions patch the SYS2/SYS file accordingly and write it back to the specified disk before returning to the operating system. Every time you use the altered disk *as a system disk in drive 0, the password protection of all files in the* system is unlocked or locked, depending on which option you chose. The quit option returns you to the operating system without altering the selected disk password status. This option allows you to check a disk's status.

Program Listing

```
        ;*****************************************
        ;*****      PASSWORD UTILITY       *****
        ;***          VERSION  1.0           ***
        ;***          AUG 10, 1981           ***
        ;***              BY                 ***
        ;*****      CRAIG A. LINDLEY       *****
        ;*****************************************
        ;
7000            ORG     7000H
        ;
        ;SYSTEM EQUATES
        ;
```

```
ØØ49                CHRIN   EQU     ØØ49H         ;ROM CHAR IN ROUTINE
Ø33A                CHROUT  EQU     Ø33AH         ;ROM CHAR OUT ROUTINE
Ø1C9                CLRSCN  EQU     Ø1C9H         ;ROM CLEAR SCREEN ROUTINE
4Ø2Ø                CURSOR  EQU     4Ø2ØH         ;CURSOR STORAGE
ØØ6Ø                DELAY   EQU     ØØ6ØH         ;ROM DELAY ROUTINE
Ø5D9                LINEIN  EQU     Ø5D9H         ;ROM LINE INPUT ROUTINE
4Ø2D                OPSYS   EQU     4Ø2DH         ;OP SYS ENTRY POINT
                    ;
6FFF                STACK   EQU     $-1
                    ;
                    ;SYSTEM STORAGE LOCATIONS
                    ;
ØØØ1                DRIVE   DEFS    1             ;DRIVE # STORAGE
ØØØ1                SECTOR  DEFS    1             ;SECTOR # STORAGE
ØØØ1                TRACK   DEFS    1             ;TRACK # STORAGE
                    ;
                    ;COMPARE STRINGS FOR SYS2/SYS - MUST BE CORRECT FOR
                    ;PROGRAM OPERATION.
                    ;
7ØØ3 ED             CMPST1  DEFB    ØEDH
7ØØ4 52                     DEFB    Ø52H
7ØØ5 E1                     DEFB    ØE1H
                    ;
7ØØ6 21             CMPST2  DEFB    Ø21H
7ØØ7 79                     DEFB    Ø79H
7ØØ8 FE                     DEFB    ØFEH
                    ;
                    ;SYSTEM MESSAGES
                    ;
7ØØ9 913C           MSG1    DEFW    3C91H
7ØØB 55                     DEFM    'UNLOCK/LOCK PASSWORD UTILITY'
7Ø27 Ø3                     DEFB    3
7Ø28 593D                   DEFW    3D59H
7Ø2A 52                     DEFM    'REV. 1.Ø'
7Ø32 Ø3                     DEFB    3
7Ø33 1C3E                   DEFW    3E1CH
7Ø35 42                     DEFM    'BY'
7Ø37 Ø3                     DEFB    3
7Ø38 553E                   DEFW    3E55H
7Ø3A 43                     DEFM    'CRAIG A. LINDLEY'
7Ø4A Ø3                     DEFB    3
7Ø4B CD3E                   DEFW    3ECDH
7Ø4D 44                     DEFM    'DRIVE TO BE UNLOCKED/LOCKED (Ø-3) ? '
7Ø71 ØØ                     DEFB    Ø
                    ;
7Ø72 55             MSG2    DEFM    'UNLOCK/LOCK PASSWORD UTILITY'
7Ø8E ØD                     DEFB    13
7Ø8F ØD                     DEFB    13
7Ø9Ø 55                     DEFM    'UNLOCK/LOCK DISK DRIVE # '
7ØA9 ØØ                     DEFB    Ø
                    ;
7ØAA ØD             MSG3    DEFB    13
7ØAB ØD                     DEFB    13
7ØAC 49                     DEFM    'INPUT THE DISK MASTER PASSWORD - '
7ØCD ØØ                     DEFB    Ø
                    ;
7ØCE 55             MSG4    DEFM    'UNLOCK/LOCK PASSWORD UTILITY'
7ØEA ØD                     DEFB    13
7ØEB ØD                     DEFB    13
7ØEC 53                     DEFM    'SELECTED DISK IS CURRENTLY:  '
```

Program continued

```
71Ø9 ØØ                    DEFB   Ø
                  ;
71ØA 2A           MSG5     DEFM   '*** LOCKED ***'
7118 ØD                    DEFB   13
7119 ØØ                    DEFB   Ø
                  ;
711A 2A           MSG6     DEFM   '*** UNLOCKED ***'
712A ØD                    DEFB   13
712B ØØ                    DEFB   Ø
                  ;
712C ØD           MSG7     DEFB   13
712D ØD                    DEFB   13
712E 2A                    DEFM   '*** OPERATION SELECTION MENU ***'
714E ØD                    DEFB   13
714F ØD                    DEFB   13
715Ø 2Ø                    DEFM   '     L - LOCK SELECTED DISK'
716B ØD                    DEFB   13
716C 2Ø                    DEFM   '     U - UNLOCK SELECTED DISK'
7189 ØD                    DEFB   13
718A 2Ø                    DEFM   '     Q - QUIT THIS PROGRAM'
71A4 ØD                    DEFB   13
71A5 ØD                    DEFB   13
71A6 57                    DEFM   'WHICH ? '
71AE ØØ                    DEFB   Ø
                  ;
71AF 55           MSG8     DEFM   'UNLOCK/LOCK PASSWORD UTILITY'
71CB ØD                    DEFB   13
71CC ØD                    DEFB   13
71CD 4F                    DEFM   'OPERATION COMPLETE'
71DF ØD                    DEFB   13
71EØ ØD                    DEFB   13
71E1 ØD                    DEFB   13
71E2 ØØ                    DEFB   Ø
                  ;
                  ;SYSTEM ERROR MESSAGES
                  ;
71E3 ØD           EMSG1    DEFB   13
71E4 ØD                    DEFB   13
71E5 2A                    DEFM   '*** ERROR - IN DISK I/O ***'
72ØØ ØD                    DEFB   13
72Ø1 ØØ                    DEFB   Ø
                  ;
72Ø2 ØD           EMSG2    DEFB   13
72Ø3 ØD                    DEFB   13
72Ø4 2A                    DEFM   '*** ERROR - DRIVE NOT READY ***'
7223 ØD                    DEFB   13
7224 ØØ                    DEFB   Ø
                  ;
7225 ØD           EMSG3    DEFB   13
7226 ØD                    DEFB   13
7227 2A                    DEFM   '*** ERROR - INCORRECT DISK PASSWORD ***'
724E ØD                    DEFB   13
724F ØØ                    DEFB   Ø
                  ;
725Ø ØD           EMSG4    DEFB   13
7251 ØD                    DEFB   13
7252 2A                    DEFM   '*** ERROR - NOT SYSTEM DISK FORMAT ***'
7278 ØD                    DEFB   13
7279 ØØ                    DEFB   Ø
                  ;
```

```
                ;
                ;***** PROGRAM SUBROUTINES *****
                ;
                ;THIS ROUTINE CALCULATES THE MAIN DISK PASSWORD FROM
                ;THE OPERATOR INPUT
                ;
727A 21FFFF     CALPW   LD      HL,-1
727D Ø6Ø8               LD      B,8
727F 7B                 LD      A,E
728Ø C6Ø7               ADD     A,7
7282 5F                 LD      E,A
7283 3ØØ1               JR      NC,PW1
7285 14                 INC     D
7286 1A         PW1     LD      A,(DE)
7287 D5                 PUSH    DE
7288 57                 LD      D,A
7289 5C                 LD      E,H
728A 7D                 LD      A,L
728B E6Ø7               AND     7
728D ØF                 RRCA
728E ØF                 RRCA
728F ØF                 RRCA
729Ø AD                 XOR     L
7291 6F                 LD      L,A
7292 26ØØ               LD      H,Ø
7294 29                 ADD     HL,HL
7295 29                 ADD     HL,HL
7296 29                 ADD     HL,HL
7297 29                 ADD     HL,HL
7298 AC                 XOR     H
7299 AA                 XOR     D
729A 57                 LD      D,A
729B 7D                 LD      A,L
729C 29                 ADD     HL,HL
729D AC                 XOR     H
729E AB                 XOR     E
729F 5F                 LD      E,A
72AØ EB                 EX      DE,HL
72A1 D1                 POP     DE
72A2 1B                 DEC     DE
72A3 1ØE1               DJNZ    PW1
72A5 C9                 RET
                ;
                ;DISPLAY LINE OUTPUT ROUTINE
                ;
72A6 7E         LOUT    LD      A,(HL)          ;GET CHAR
72A7 B7                 OR      A               ;IS IT Ø ?
72A8 C8                 RET     Z               ;IF YES THEN
72A9 FEØ3               CP      3               ;OTHER TERMINATOR
72AB C8                 RET     Z               ;IF YES THEN
72AC CD3AØ3             CALL    CHROUT          ;DISPLAY IT
72AF 23                 INC     HL              ;NEXT CHAR
72BØ 18F4               JR      LOUT            ;UNTIL FINISHED
                ;
                ;FORMATTED MESSAGE OUTPUT ROUTINE
                ;
72B2 112Ø4Ø     MSGOUT  LD      DE,CURSOR       ;CURR CURSOR STORAGE
72B5 Ø1Ø2ØØ             LD      BC,2            ;MOVE ADDRESS
72B8 EDBØ               LDIR                    ;INTO CURSOR STORAGE
72BA CDA672             CALL    LOUT            ;DISPLAY LINE
```

Program continued

```
72BD F5       Ø2Ø2Ø        PUSH   AF               ;SAVE TERM CHAR
72BE 3EØF     Ø2Ø3Ø        LD     A,15             ;CURSOR OFF CODE
72CØ CD3AØ3   Ø2Ø4Ø        CALL   CHROUT           ;OUTPUT IT
72C3 F1       Ø2Ø5Ø        POP    AF               ;RESTORE CHAR
72C4 23       Ø2Ø6Ø        INC    HL               ;NEXT CHAR
72C5 FEØØ     Ø2Ø7Ø        CP     Ø                ;END OF MESSAGE ?
72C7 28Ø2     Ø2Ø8Ø        JR     Z,MSGO1          ;YES, FINISH UP
72C9 18E7     Ø2Ø9Ø        JR     MSGOUT           ;UNTIL FINISHED
72CB 3EØE     Ø21ØØ MSGO1  LD     A,14             ;CURSOR ON CHAR
72CD CD3AØ3   Ø211Ø        CALL   CHROUT           ;DO IT
72DØ C9       Ø212Ø        RET
              Ø213Ø ;
              Ø214Ø ;RESTORE DRIVE TO TRACK ØØ ROUTINE
              Ø215Ø ;
72D1 CDØ973   Ø216Ø RESTOR CALL   SELDSK           ;SELECT DRIVE
72D4 3EØ3     Ø217Ø        LD     A,3              ;SEEK ØØ CMD
72D6 32EC37   Ø218Ø        LD     (37ECH),A        ;GO
72D9 CDØ973   Ø219Ø RES1   CALL   SELDSK           ;SELECT DRIVE
72DC 3AEC37   Ø22ØØ        LD     A,(37ECH)        ;GET STATUS
72DF ØF       Ø221Ø        RRCA                    ;BUSY ?
72EØ DØ       Ø222Ø        RET    NC               ;IF NOT THEN
72E1 18F6     Ø223Ø        JR     RES1             ;LOOP UNTIL NOT BUSY
              Ø224Ø ;
              Ø225Ø ;SEEK ROUTINE - THIS ROUTINE SETS UP THE SELECTED DISK
              Ø226Ø ;WITH A SEEK TO THE SPECIFIED TRACK. ON RETURN, THE
              Ø227Ø ;HEAD IS POSITIONED AT TRACK (TRACK), SECTOR (SECTOR).
              Ø228Ø ;
72E3 CDØ973   Ø229Ø SEEK   CALL   SELDSK           ;SELECT THE DRIVE
72E6 21EC37   Ø23ØØ        LD     HL,37ECH         ;POINT AT STATUS/COM REG
72E9 3AØ27Ø   Ø231Ø        LD     A,(TRACK)        ;GET TRACK
72EC 32EF37   Ø232Ø        LD     (37EFH),A        ;SELECT THE TRACK
72EF 3AØ17Ø   Ø233Ø        LD     A,(SECTOR)       ;GET SECTOR
72F2 32EE37   Ø234Ø        LD     (37EEH),A        ;SELECT SECTOR
72F5 3E1F     Ø235Ø        LD     A,1FH            ;SEEK CODE
72F7 77       Ø236Ø        LD     (HL),A           ;SEEK POSITION
72F8 F5       Ø237Ø        PUSH   AF
72F9 F1       Ø238Ø        POP    AF
72FA F5       Ø239Ø        PUSH   AF
72FB F1       Ø24ØØ        POP    AF               ;WASTE TIME
72FC 7E       Ø241Ø CREADY LD     A,(HL)           ;GET STATUS
72FD Ø7       Ø242Ø        RLCA
72FE 3ØØ4     Ø243Ø        JR     NC,CBUSY         ;GO IF READY
73ØØ E1       Ø244Ø        POP    HL               ;FIX STACK
73Ø1 C34Ø73   Ø245Ø        JP     NREADY
              Ø246Ø ;
73Ø4 ØF       Ø247Ø CBUSY  RRCA                    ;ADJ BYTE
73Ø5 ØF       Ø248Ø        RRCA                    ;BUSY BIT TO FLAG
73Ø6 DØ       Ø249Ø        RET    NC               ;IF NOT BUSY THEN
73Ø7 18F3     Ø25ØØ        JR     CREADY           ;ELSE
              Ø251Ø ;
              Ø252Ø ;SELECT THE DISK DRIVE ROUTINE
              Ø253Ø ;
73Ø9 C5       Ø254Ø SELDSK PUSH   BC               ;SAVE "BC"
73ØA 3AØØ7Ø   Ø255Ø        LD     A,(DRIVE)        ;GET DRIVE #
73ØD 47       Ø256Ø        LD     B,A              ;"B" IS COUNTER
73ØE 3E8Ø     Ø257Ø        LD     A,8ØH            ;SELECTION BITS
731Ø Ø7       Ø258Ø SEL1   RLCA                    ;ROTATE
7311 1ØFD     Ø259Ø        DJNZ   SEL1             ;UNTIL FINISHED
7313 32E137   Ø26ØØ        LD     (37E1H),A        ;SELECT THE DRIVE
7316 C1       Ø261Ø        POP    BC               ;RESTORE "BC"
```

```
7317 C9         Ø262Ø          RET
                Ø263Ø ;
                Ø264Ø ;SECTOR READ ROUTINE ENTRY WITH BUFFER IN "BC"
                Ø265Ø ;DRIVE TRACK AND SECTOR NUMBERS ALREADY IN MEMORY
                Ø266Ø ;
7318 CDE372     Ø267Ø SREAD    CALL     SEEK           ;POSITION HEAD FOR
                Ø268Ø                                  ;REQUIRED OPERATION
731B 368C       Ø269Ø READ     LD       (HL),8CH       ;READ COMMAND
731D 11EF37     Ø27ØØ          LD       DE,37EFH       ;POINT TO DATA REGISTER
732Ø F5         Ø271Ø          PUSH     AF
7321 F1         Ø272Ø          POP      AF
7322 F5         Ø273Ø          PUSH     AF
7323 F1         Ø274Ø          POP      AF
7324 18Ø3       Ø275Ø          JR       READ2
                Ø276Ø ;
7326 ØF         Ø277Ø READ1    RRCA                    ;BUSY BIT TO FLAG
7327 3ØØA       Ø278Ø          JR       NC,STATCK      ;BUSY/DONE GO CK STATUS
7329 7E         Ø279Ø READ2    LD       A,(HL)         ;GET STATUS
732A CB4F       Ø28ØØ          BIT      1,A            ;DRQ BIT
732C 28F8       Ø281Ø          JR       Z,READ1        ;GO CHECK BUSY
732E 1A         Ø282Ø READIT   LD       A,(DE)         ;GET DATA BYTE
732F Ø2         Ø283Ø          LD       (BC),A         ;STORE IN BUF1
733Ø Ø3         Ø284Ø          INC      BC             ;NEXT BUF POSITION
7331 18F6       Ø285Ø          JR       READ2          ;GO CK DRQ
7333 7E         Ø286Ø STATCK   LD       A,(HL)         ;GET STATUS
7334 E65C       Ø287Ø          AND      5CH            ;MASK IT
7336 C8         Ø288Ø          RET      Z              ;IF NO ERROR
                Ø289Ø ;
                Ø29ØØ ;DISK ERROR OUTPUT ROUTINE
                Ø291Ø ;
7337 21E371     Ø292Ø DSKERR   LD       HL,EMSG1       ;DISK ERROR MSG
733A CDA672     Ø293Ø ERROR    CALL     LOUT           ;DISPLAY IT
733D C32D4Ø     Ø294Ø          JP       OPSYS          ;BACK TO DOS
734Ø 21Ø272     Ø295Ø NREADY   LD       HL,EMSG2       ;DRIVE NOT READY ERR MSG
7343 18F5       Ø296Ø          JR       ERROR          ;DISPLAY AND ABORT
                Ø297Ø ;
                Ø298Ø ;STRING COMPARISON ROUTINE FOR SYS2/SYS FORMAT VERIFY
                Ø299Ø ;
7345 1A         Ø3ØØØ STRCMP   LD       A,(DE)         ;GET DISK CHAR
7346 BE         Ø3Ø1Ø          CP       (HL)           ;IS IT = TO CMPSTR CHAR ?
7347 2ØØ5       Ø3Ø2Ø          JR       NZ,ERR1        ;IF MISCOMPARE THEN
7349 23         Ø3Ø3Ø          INC      HL             ;INC PTRS
734A 13         Ø3Ø4Ø          INC      DE
734B 1ØF8       Ø3Ø5Ø          DJNZ     STRCMP         ;CHECK ALL 7 CHARS
734D C9         Ø3Ø6Ø          RET                     ;WHEN COMPARE FINISHED
734E E1         Ø3Ø7Ø ERR1     POP      HL             ;FIX STACK
734F 215Ø72     Ø3Ø8Ø          LD       HL,EMSG4       ;FORMAT ERROR MSG
7352 C33A73     Ø3Ø9Ø          JP       ERROR          ;DISPLAY AND ABORT
                Ø31ØØ ;
                Ø311Ø ;WRITE SECTOR ROUTINE
                Ø312Ø ;
7355 CDE372     Ø313Ø SWRITE   CALL     SEEK           ;POSITION HEAD FOR THE
                Ø314Ø                                  ;REQUIRED OPERATION
7358 36AC       Ø315Ø WRITE    LD       (HL),ØACH      ;WRITE CMD
735A 11EF37     Ø316Ø          LD       DE,37EFH       ;POINT TO DATA REGISTER
735D F5         Ø317Ø          PUSH     AF
735E F1         Ø318Ø          POP      AF
735F F5         Ø319Ø          PUSH     AF
736Ø F1         Ø32ØØ          POP      AF             ;WASTE TIME
7361 18Ø3       Ø321Ø          JR       WRITE2
```

Program continued

```
                 Ø322Ø ;
7363 ØF          Ø323Ø WRITE1  RRCA                          ;BUSY BIT TO FLAG
7364 3ØCD        Ø324Ø         JR      NC,STATCK             ;BUSY/DONE GO CK STATUS
7366 7E          Ø325Ø WRITE2  LD      A,(HL)                ;GET STATUS
7367 CB4F        Ø326Ø         BIT     1,A                   ;DRQ BIT
7369 28F8        Ø327Ø         JR      Z,WRITE1              ;GO CHECK BUSY
736B ØA          Ø328Ø WRITIT  LD      A,(BC)                ;GET BYTE
736C Ø3          Ø329Ø         INC     BC                    ;POINT TO NEXT
736D 12          Ø33ØØ         LD      (DE),A                ;STORE ON DISK
736E 18F6        Ø331Ø         JR      WRITE2                ;GO CHECK DRQ
                 Ø332Ø ;
                 Ø333Ø ;WAIT ROUTINE - THIS ROUTINE SELECTS THE SPECIFIED DISK
                 Ø334Ø ;DRIVE AND WAITS FOR IT TO COME UP TO SPEED BEFORE
                 Ø335Ø ;RETURNING TO THE CALLING PROGRAM.
                 Ø336Ø ;
737Ø CDØ973      Ø337Ø WAIT    CALL    SELDSK                ;SELECT THE DRIVE
7373 Ø1ØØØØ      Ø338Ø         LD      BC,Ø                  ;MAX DELAY
7376 CD6ØØØ      Ø339Ø         CALL    DELAY                 ;ROM ROUTINE
7379 C9          Ø34ØØ         RET
                 Ø341Ø ;
                 Ø342Ø ;***********************************
                 Ø343Ø ;***** START OF MAIN PROGRAM *****
                 Ø344Ø ;***********************************
                 Ø345Ø ;
737A F3          Ø346Ø START   DI                            ;DISABLE INTERRUPTS
737B 31FF6F      Ø347Ø         LD      SP,STACK              ;LOAD STACK PTR
737E CDC9Ø1      Ø348Ø         CALL    CLRSCN                ;CLEAR DISPLAY
7381 21Ø97Ø      Ø349Ø         LD      HL,MSG1               ;SIGN ON MESSAGE
7384 CDB272      Ø35ØØ         CALL    MSGOUT                ;DISPLAY IT
7387 CD49ØØ      Ø351Ø ASK     CALL    CHRIN                 ;GET RESPONSE
738A FE34        Ø352Ø         CP      '4'                   ;MAX + 1
738C 3ØF9        Ø353Ø         JR      NC,ASK                ;ERROR IF > 3
738E FE3Ø        Ø354Ø         CP      'Ø'                   ;MIN
739Ø 38F5        Ø355Ø         JR      C,ASK                 ;ERROR IF < Ø
7392 F5          Ø356Ø         PUSH    AF                    ;SAVE ASCII DRIVE #
7393 D63Ø        Ø357Ø         SUB     3ØH                   ;CONVERT TO HEX
7395 3C          Ø358Ø         INC     A                     ;ADD ONE TO DRIVE NUMBER
7396 32ØØ7Ø      Ø359Ø         LD      (DRIVE),A             ;STORE DRIVE NUMBER
                 Ø36ØØ ;
                 Ø361Ø ;ANSWER IN RANGE CONTINUE
                 Ø362Ø ;
7399 CDC9Ø1      Ø363Ø         CALL    CLRSCN                ;CLEAR DISPLAY
739C 21727Ø      Ø364Ø         LD      HL,MSG2               ;MSG
739F CDA672      Ø365Ø         CALL    LOUT                  ;DISPLAY IT
73A2 F1          Ø366Ø         POP     AF                    ;RESTORE ASCII DRIVE #
73A3 CD3AØ3      Ø367Ø         CALL    CHROUT                ;DISPLAY IT
                 Ø368Ø ;
                 Ø369Ø ;READ THE DIRECTORY GAT SECTOR ØØ OF SELECTED DISK DRIVE
                 Ø37ØØ ;
73A6 CD7Ø73      Ø371Ø         CALL    WAIT                  ;WAIT FOR DRIVE TO
                 Ø372Ø                                       ;COME UP TO SPEED
73A9 3E11        Ø373Ø         LD      A,11H                 ;TRK NUM
73AB 32Ø27Ø      Ø374Ø         LD      (TRACK),A             ;STORE TRK NUM
73AE 3EØØ        Ø375Ø         LD      A,Ø                   ;SECTOR Ø
73BØ 32Ø17Ø      Ø376Ø         LD      (SECTOR),A            ;STORE SECTOR NUM
73B3 CDD172      Ø377Ø         CALL    RESTOR                ;SEEK TRK ØØ
73B6 Ø16A74      Ø378Ø         LD      BC,DBUF               ;SECTOR BUFFER AREA
73B9 CD1873      Ø379Ø         CALL    SREAD                 ;READ GAT IN DIR
                 Ø38ØØ ;
```

```
              Ø381Ø ;ASK FOR DISK PASSWORD ROUTINE
              Ø382Ø ;
73BC 21AA7Ø   Ø383Ø ASK1     LD      HL,MSG3         ;REQUEST MSG
73BF CDA672   Ø384Ø          CALL    LOUT            ;DISPLAY IT
73C2 216A75   Ø385Ø          LD      HL,INBUF        ;BUF FOR OPERATOR INPUT
73C5 Ø6Ø8     Ø386Ø          LD      B,8             ;MAX LENGTH OF PASSWORD
73C7 CDD9Ø5   Ø387Ø          CALL    LINEIN          ;GET PASSWORD
73CA 78       Ø388Ø          LD      A,B             ;LENGTH INTO "A"
73CB B7       Ø389Ø          OR      A
73CC 28EE     Ø39ØØ          JR      Z,ASK1          ;IF NO INPUT ASK AGAIN
              Ø391Ø ;
73CE EB       Ø392Ø          EX      DE,HL           ;POINT "DE" AT PASSWORD
73CF 83       Ø393Ø          ADD     A,E             ;POINT "HL" AT END
73DØ 6F       Ø394Ø          LD      L,A
73D1 7A       Ø395Ø          LD      A,D
73D2 CEØØ     Ø396Ø          ADC     A,Ø
73D4 67       Ø397Ø          LD      H,A
73D5 3EØ8     Ø398Ø          LD      A,8             ;MAX LENGTH
73D7 9Ø       Ø399Ø          SUB     B               ;- LENGTH
73D8 47       Ø4ØØØ          LD      B,A             ;INTO "B"
73D9 362Ø     Ø4Ø1Ø ASK2     LD      (HL),2ØH        ;STORE BLANKS
73DB 23       Ø4Ø2Ø          INC     HL              ;NEXT POSITION
73DC 1ØFB     Ø4Ø3Ø          DJNZ    ASK2            ;FILL REMAINDER WITH
              Ø4Ø4Ø                                  ;BLANK CODE 2ØH
73DE CD7A72   Ø4Ø5Ø          CALL    CALPW           ;CALCULATED PASSWORD
              Ø4Ø6Ø                                  ;IN "HL"
              Ø4Ø7Ø ;
              Ø4Ø8Ø ;COMPARE CALCULATED PASSWORD TO THE ONE STORED IN THE GAT
              Ø4Ø9Ø ;SECTOR OF THE SELECTED DISK.
              Ø41ØØ ;
73E1 ED5B3875 Ø411Ø          LD      DE,(DBUF+2Ø6)   ;PASSWORD LOCATION IN GAT
73E5 DF       Ø412Ø          RST     18H             ;COMPARE
73E6 28Ø6     Ø413Ø          JR      Z,ASK3          ;IF OK THEN
              Ø414Ø ;
73E8 212572   Ø415Ø          LD      HL,EMSG3        ;ACCESS DENIED MESSAGE
73EB C33A73   Ø416Ø          JP      ERROR           ;DISPLAY AND ABORT
              Ø417Ø ;
73EE 3E1Ø     Ø418Ø ASK3     LD      A,1ØH           ;SYS2/SYS LOCATION
73FØ 32Ø27Ø   Ø419Ø          LD      (TRACK),A
73F3 3EØ6     Ø42ØØ          LD      A,6             ;SECTOR NUM
73F5 32Ø17Ø   Ø421Ø          LD      (SECTOR),A
73F8 CD7Ø73   Ø422Ø          CALL    WAIT            ;WAIT FOR DRIVE TO
              Ø423Ø                                  ;COME UP TO SPEED
73FB Ø16A74   Ø424Ø          LD      BC,DBUF         ;SECTOR BUFFER AREA
73FE CD1873   Ø425Ø          CALL    SREAD           ;READ SYS2/SYS
              Ø426Ø ;
              Ø427Ø ;SYS2/SYS SHOULD NOW BE LOADED. CHECK FORMAT TO BE SURE.
              Ø428Ø ;
74Ø1 21Ø37Ø   Ø429Ø          LD      HL,CMPST1       ;POINT AT COMPARE STR 1
74Ø4 11CA74   Ø43ØØ          LD      DE,DBUF+96      ;PATCH POINT COMPARE
74Ø7 Ø6Ø3     Ø431Ø          LD      B,3             ;BYTES TO COMPARE
74Ø9 CD4573   Ø432Ø          CALL    STRCMP          ;COMPARE THE STRINGS
74ØC 21Ø67Ø   Ø433Ø          LD      HL,CMPST2       ;2ND COMPARE STRING
74ØF 11CE74   Ø434Ø          LD      DE,DBUF+1ØØ     ;PATCH POINT COMPARE
7412 Ø6Ø3     Ø435Ø          LD      B,3             ;BYTES TO COMPARE
7414 CD4573   Ø436Ø          CALL    STRCMP          ;COMPARE THE STRINGS
              Ø437Ø ;
              Ø438Ø ;DISPLAY CURRENT STATUS UNLOCKED OR LOCKED
              Ø439Ø ;
```

Program continued

```
7417 CDC901     ASK5    CALL    CLRSCN          ;CLEAR DISPLAY
741A 21CE70             LD      HL,MSG4         ;STATUS MSG
741D CDA672             CALL    LOUT
7420 21CD74             LD      HL,DBUF+99      ;PATCH DATA
7423 7E                 LD      A,(HL)          ;GET DATA
7424 FE28               CP      28H             ;LOCKED ?
7426 2805               JR      Z,LOCKED
7428 211A71             LD      HL,MSG6         ;UNLOCKED MSG
742B 1803               JR      DISIT           ;DISPLAY IT
742D 210A71     LOCKED  LD      HL,MSG5         ;LOCKED MSG
7430 CDA672     DISIT   CALL    LOUT            ;DISPLAY STATUS
                ;
                ;GET OPERATOR RESPONSE
                ;
7433 212C71             LD      HL,MSG7         ;OPTION MENU
7436 CDA672             CALL    LOUT            ;DISPLAY IT
7439 CD4900             CALL    CHRIN           ;GET RESPONSE
743C CD3A03             CALL    CHROUT          ;DISPLAY RESPONSE
743F FE51               CP      'Q'             ;QUIT ?
7441 CA2D40             JP      Z,OPSYS         ;IF YES THEN
7444 FE4C               CP      'L'             ;LOCK DISK ?
7446 2806               JR      Z,LOCK
7448 FE55               CP      'U'             ;UNLOCK DISK ?
744A 2806               JR      Z,UNLOCK
744C 18C9               JR      ASK5            ;IF NONE ASK AGAIN
                ;
744E 3E28       LOCK    LD      A,28H           ;PATCH VALUE - LOCK
7450 1802               JR      PATCH
7452 3E18       UNLOCK  LD      A,18H           ;PATCH VALUE - UNLOCK
                ;
                ;SYS2/SYS IS LOADED CORRECTLY NOW MAKE PATCH
                ;
7454 21CD74     PATCH   LD      HL,DBUF+99      ;PATCH LOCATION
7457 77                 LD      (HL),A          ;STORE INTO SECTOR DATA
7458 CD7073             CALL    WAIT            ;WAIT FOR DRIVE TO
                                                ;COME UP TO SPEED
745B 016A74             LD      BC,DBUF         ;POINT AT SECTOR DATA
745E CD5573             CALL    SWRITE          ;SECTOR BACK TO DISK
7461 CDC901             CALL    CLRSCN          ;CLEAR DISPLAY
7464 21AF71             LD      HL,MSG8         ;OPERATION COMPLETE MSG
7467 C33A73             JP      ERROR           ;DISPLAY AND ABORT
                ;
                ;
0100            DBUF    DEFS    256             ;SECTOR DATA STORAGE
756A            INBUF   EQU     $               ;INPUT BUFFER AREA
                ;
                ;
737A                    END     START
00000 TOTAL ERRORS
```

24

A Better LDOS KSM Builder

by Mike Tipton

System Requirements:
Model I or III
16K RAM
One disk drive
LDOS

One of the most useful features of the LDOS disk operating system is the filter routine, which you can use to enhance the capability of a peripheral driver. The keystroke multiplier (KSM), one of the LDOS filters, lets you build a file of commands or text strings. The file of commands is loaded into memory when KSM is activated. You select each command by pressing CLEAR and an alphabetic key. When programming in BASIC, for example, you might define CLEAR-G to display GOTO. All the 26 letters can define useful BASIC keywords.

The BUILD library command permits you to define KSM files. When you invoke BUILD, it displays a letter of the alphabet. You are to type the text string that is to be assigned to the letter. Each letter is displayed in succession. You can exit before Z, but you must start with A and you cannot skip any letters. BUILD has no editing capabilities. If you make an error, you must start over at A. This makes the BUILD utility awkward to use, but I solved this problem by writing a short program in BASIC (see Program Listing) that allows you to both create and edit KSM files. It only takes a few minutes to type it into your system. Comments were left out to make the typing easier, but it is short enough that you should be able to figure out how it works.

The first prompt is (B)UILD NEW FILE OR (R)EVISE OLD. If you type R you are asked for the name of the file you wish to revise. If you type B the program goes directly to the build and edit function. The next prompt is WHICH LETTER. If you type a single letter (A-Z), the program displays the text associated with the letter, if any, and permits you to change it. If no change is desired, press ENTER; otherwise, type the text you want. This process repeats as often as necessary.

There are also four commands you can use to respond to the prompt:

PRINT—Prints all your definitions on a line printer. The printout can be saved for future reference. Make certain that your printer is on-line before using this command. The computer will lock up if it is not.

LIST—Displays all your definitions on the CRT.

SAVE—Allows you to write your new KSM file to the disk. You are asked for a file name.

LOAD—Allows you to load and edit a new KSM file. The current file in memory is destroyed.

Program Listing

```
20 CLS : PRINT@384,  "KSM BUILD UTILITY"
40 CLEAR 2000
60 DEFINT A-Z: DIM AA$(26)
80 INPUT "(B)UILD NEW FILE OR (R)EVISE OLD"; T$
100 IF T$="B" THEN GOTO 260:
120 IF T$="R" THEN GOTO 160
140 GOTO 80
160 INPUT "FILENAME";T$
180 OPEN "I",1,T$
200 FOR J=1 TO 26: INPUT #1,AA$(J): NEXT J
220 CLOSE 1
240 GOTO 280
260 FOR J=1 TO 26: AA$(J)=" ": NEXT J
280 LINEINPUT "WHICH LETTER  ";T$
300 IF T$="" THEN PRINT "ERROR": GOTO 280
320 IF T$="PRINT" THEN GOTO 640
340 IF T$="LIST" THEN GOTO 720
360 IF T$="SAVE" THEN GOTO 540
380 IF T$="LOAD" THEN GOTO 160
400 IF LEN(T$)>1 THEN PRINT "ERROR": GOTO 280
420 T=ASC(LEFT$(T$,1))-64
440 IF T<1 OR T>27 THEN PRINT "ERROR": GOTO 280
460 PRINT T$;"=";AA$(T)
480 PRINT T$;"=?";:T$="":LINEINPUT T$
500 IF T$<>"" THEN AA$(T)=T$
520 GOTO 280
540 INPUT "FILE NAME";F$
560 OPEN "O",1,F$
580 FOR J=1 TO 26: PRINT #1,AA$(J): NEXT J
600 CLOSE 1
620 GOTO 280
640 FOR J=1 TO 26
660 LPRINT CHR$(J+64);"=";AA$(J)
680 NEXT J
700 GOTO 280
720 FOR J=1 TO 26
740 PRINT CHR$(J+64);"=";AA$(J),
760 NEXT J
780 PRINT
800 GOTO 280
```

25

T-EDASM: Link T-BUG and EDTASM

by Ron Anderson

System Requirements:
Level II BASIC
Model I
16K RAM
Radio Shack EDTASM and T-BUG

Are you tired of loading and reloading EDTASM and T-BUG again and again to assemble and debug *one* simple machine-language program? I was, until I developed a program, T-EDASM, that links these two utilities (EDTASM and T-BUG) into a unified assembly and check-out program.

T-EDASM lets you jump between EDTASM and T-BUG with single commands, and loads assembled object code directly into memory from EDTASM. No tape and play sequence is required. You only have to load your machine-language program once to fully develop, test, and debug it.

To use T-EDASM, load the single tape program and enter EDTASM. To move to T-BUG, type the new command B and press ENTER. To return to EDTASM, press A. T-EDASM gives T-BUG two other new commands. The I command tells EDTASM to dump object code in memory. Instead of producing an object code on tape (A command plus two ENTERs), it puts the code directly into memory at the ORG indicated in the code. The E command converts EDTASM to the normal tape dump mode again. After an EDTASM assemble command, the prompt READY CASSETTE appears in the E mode. In the I mode, however, the prompt is READY TO STORE.

Here is a typical programming session with T-EDASM. Develop your program using EDTASM, hop to T-BUG (with B), convert to internal load, and move back to EDTASM (I) to assemble and load the code into memory. Next, go back to T-BUG (B) and use it on your new program now in memory. Repeat as required. Finally, use the T-BUG E command to go back to the normal EDTASM and produce final object and source tapes.

You must move T-BUG to a higher than normal memory location because of a fundamental EDTASM and T-BUG conflict. I recommend that you extend T-BUG as described in "T-BUG and Then Some," by M. Paxton (*80 Microcomputing*, November 1980), then move the modified program into high memory. For this, see "Get T-BUG High," by I. Rappaport (*80 Microcomputing*, January 1980). The version of T-EDASM shown in the program listing is for use with the extended form of T-BUG starting at address 7380H.

After you have moved T-BUG, the memory map for the final T-EDASM (for 16K of memory) is as shown in Figure 1. This leaves 6600H–71FFH (a little over 3K bytes) of storage for your new program. By using relative jumps in your program, you can use a new ORG just before making the final tapes to put your program anywhere else in memory.

Hex Memory Locations	**Program Portion**
42E9–5D40 plus EDTASM text	EDTASM (check end of text with T-BUG)
6600–71FF	Memory for object code (varies with text)
7200–72E4 (7200–72E9 for regular T-BUG)	T-EDASM
7380–797F	Regular T-BUG
7380–7B8A	Extended T-BUG
7B8B–7FFF	EDTASM Table

Figure 1

Type the T-EDASM program listing into EDTASM and record the object code. If you do not use the extended T-BUG, insert the following program lines into the listing in place of line 710:

```
710  CP  'F'       ;REPLACE ORIG T-BUG
712  JP  Z,780DH   ;BYTES
714  JP  73EAH     ;GOTO T-BUG(HIGH)
```

and use regular T-BUG, moved to start at 7380H.

Load T-EDASM, EDTASM and T-BUG (high) and enter T-BUG. Using the M command make the code change in T-BUG and EDTASM shown in Figure 2. The old bytes for T-BUG are for the extended versions. If you are not using this version, the old bytes from 73E5H to 73E9H are FE, 46, CA, 0D, and 78 (hex). Jump (J468A) to EDTASM's cold start and try everything. Load a fresh tape and use T-BUG's P command (P 42E9 7B8A 468A TEDASM) to save the programs. The 7B8A should be 797F if you have the non-extended T-BUG. T-EDASM is now ready with a file name of TEDASM.

If your experimental program crashes, Figure 3 shows a few entry points that may restore operation of T-EDASM. If that does not work, you have to reload the programs, so make a source code tape every now and then for a long machine language program. If T-BUG hangs up, just press ENTER.

T-BUG

Hex Mem	Old	New
73E5	C3	C3
73E6	06	6C
73E7	7A	72
73E8	00	00
73E9	00	00

EDTASM

Hex Mem	Old	New
4930	00	A0
4931	00	73

Figure 2

Program Portion	Entry (Hex)	Entry (Dec)
EDTASM (cold start)	468A	18058
EDTASM (warm start)	46A2	18082
T-BUG (high)	73A0	29600

Figure 3

Program Commentary

Lines 30-190 contain the program which stores the register-A byte passed from EDTASM (originally to be output to tape) in memory as an object code. The code here is similar to the tape-read portion of T-BUG (low) from 4668H to 469BH. Line 40 is the entry point for the first byte from EDTASM, if no call was made to EDTASM. Lines 200-310 complete EDTASM data passage to memory and reinitiate the first byte entry point. Lines 320-380 contain the first entry point from EDTASM where register A has data, without requiring a call. This portion of the program saves important EDTASM parameters and changes subsequent T-EDASM entry points to RENTRY.

Lines 390-470 save local CALL-RETURN addresses, store T-EDASM important variables, and restore EDTASM parameters. Lines 480-550 save EDTASM variables and retrieve T-EDASM parameters. Lines 560-580 set aside storage space to save T-EDASM parameters. Lines 590-720 are the extension of the T-BUG command input routine to check for A, I, and E commands and jumps as required. The delay prevents key bounce.

Lines 730-830 revise EDTASM to incorporate T-EDASM and assemble to memory. They also modify the prompt to READY TO STORE. Lines 840-940 restore EDTASM to its original configuration, which assembles to tape. They also restore the READY CASSETTE prompt.

The linkage between EDTASM and T-EDASM is accomplished when an EDTASM call statement attempts to pass an assembled byte to a record subroutine, and T-EDASM intercepts the byte, decodes the information it contains, and stores the actual object code in memory. T-EDASM then calls (RET) EDTASM for another byte.

Program Listing

```
               ØØØ1Ø ;TEDASM BY RON ANDERSON
72ØØ           ØØØ2Ø        ORG     72ØØH
72ØØ CD4572    ØØØ3Ø DO1    CALL    DATA      ;GO GET A BYTE FROM EDTASM
72Ø3 FE78      ØØØ4Ø ENTRY  CP      78H       ;END OF FILE?
72Ø5 281C      ØØØ5Ø        JR      Z,FIN     ;IF SO GO FINISH
72Ø7 FE3C      ØØØ6Ø        CP      3CH       ;DATA HEADER?
72Ø9 2ØF5      ØØØ7Ø        JR      NZ,DO1    ;NO SEARCH MORE
72ØB CD4572    ØØØ8Ø        CALL    DATA      ;YES, GET FIRST BYTE (DATA COUNT)
72ØE 47        ØØØ9Ø        LD      B,A       ;SAVE DATA COUNT
72ØF CD4572    ØØ1ØØ        CALL    DATA      ;GET LSB OF LOAD ADDR
7212 5F        ØØ11Ø        LD      E,A
7213 CD4572    ØØ12Ø        CALL    DATA      ;GET MSB OF LOAD ADDR
7216 57        ØØ13Ø        LD      D,A
7217 CD4572    ØØ14Ø DO2    CALL    DATA      ;GET LINE OF DATA
721A 12        ØØ15Ø        LD      (DE),A    ;LOAD A BYTE
721B 13        ØØ16Ø        INC     DE        ;MOVE UP
721C 1ØF9      ØØ17Ø        DJNZ    DO2       ;DO TIL DONE
721E CD4572    ØØ18Ø        CALL    DATA      ;GET CHECKSUM (DON'T USE)
7221 18DD      ØØ19Ø        JR      DO1       ;START OVER AGAIN
7223 CD4572    ØØ2ØØ FIN    CALL    DATA      ;GET LSB OF ENTRY ADDR
7226 6F        ØØ21Ø        LD      L,A
7227 CD4572    ØØ22Ø        CALL    DATA      ;GET MSB OF ENTRY ADDR
722A 67        ØØ23Ø        LD      H,A
722B 226672    ØØ24Ø        LD      (STORE1),HL     ;DUMP START ADDR
722E 213972    ØØ25Ø        LD      HL,FIRENT       ;FIX PROGRAM
7231 228A43    ØØ26Ø        LD      (438AH),HL      ;   START
7234 F1        ØØ27Ø        POP     AF        ;FIX EDTASM STUFF
7235 D1        ØØ28Ø        POP     DE
7236 C1        ØØ29Ø        POP     BC
7237 E1        ØØ3ØØ        POP     HL
7238 C9        ØØ31Ø        RET     ;GO BACK TO EDTASM
7239 E5        ØØ32Ø FIRENT PUSH    HL        ;SAVE EDTASM STUFF
723A C5        ØØ33Ø        PUSH    BC
723B D5        ØØ34Ø        PUSH    DE
723C F5        ØØ35Ø        PUSH    AF
723D 215672    ØØ36Ø        LD      HL,RENTRY       ;MODIFY ENTRY PT
724Ø 228A43    ØØ37Ø        LD      (438AH),HL      ;  SUBSQ DATA
7243 18BE      ØØ38Ø        JR      ENTRY   ;PRESS ON
7245 E1        ØØ39Ø DATA   POP     HL        ;GET LOCAL RET ADDR
7246 226672    ØØ4ØØ        LD      (STORE1),HL     ;SAVE LOCAL RET
7249 ED536872  ØØ41Ø        LD      (STORE2),DE     ;SAVE LOAD INDEX
724D ED436A72  ØØ42Ø        LD      (STORE3),BC     ;SAVE DATA COUNT
7251 F1        ØØ43Ø        POP     AF
7252 D1        ØØ44Ø        POP     DE        ;GET STUFF
7253 C1        ØØ45Ø        POP     BC
7254 E1        ØØ46Ø        POP     HL
7255 C9        ØØ47Ø        RET     ;BACK TO EDTASM
7256 E5        ØØ48Ø RENTRY PUSH    HL
7257 C5        ØØ49Ø        PUSH    BC        ;SAVE STUFF
7258 D5        ØØ5ØØ        PUSH    DE
7259 F5        ØØ51Ø        PUSH    AF
```

```
725A ED5B6872 00520         LD      DE,(STORE2)     ;RECOVER LOAD INDEX
725E ED4B6A72 00530         LD      BC,(STORE3)     ;RECOVER BYTE COUNT
7262 2A6672   00540         LD      HL,(STORE1)     ;RECOVER LOCAL RET
7265 E9       00550         JP      (HL)            ;   AND GOTO IT
0002          00560 STORE1  DEFS    2
0002          00570 STORE2  DEFS    2
0002          00580 STORE3  DEFS    2
726C FE41     00590         CP      'A'     ;CHK FOR EDTASM RETURN
726E 200B     00600         JR      NZ,NEXT ;NO, NEXT INPUT CHK
7270 010050   00610 DEL     LD      BC,5000H        ;YES, DELAY
7273 0B       00620 DO3     DEC     BC      ;    THE
7274 78       00630         LD      A,B     ;        JUMP
7275 B1       00640         OR      C       ;            TO
7276 20FB     00650         JR      NZ,DO3  ;              EDTASM
7278 C3A246   00660         JP      46A2H   ;GOTO EDTASM
727B FE49     00670 NEXT    CP      'I'     ;CHK FOR INTERNAL ASSEMBLY
727D 280C     00680         JR      Z,INT   ;YES,SET-UP FOR INT STORE
727F FE45     00690         CP      'E'     ;CHK FOR EXTERNAL ASSEMBLY
7281 282E     00700         JR      Z,EXT   ;YES, RESTORE INT TO EXT
7283 C3067A   00710         JP      7A06H   ;NO HITS, PRESS ON IN TBUG
0005          00720         DEFS    5       ;ROOM FOR TEDASM ADDITIONS
728B DD21A943 00730 INT     LD      IX,43A9H        ;PREVENT EDTASM
728F DD3600C9 00740         LD      (IX),0C9H       ;  TAPE ON AND SYNC
7293 DD218943 00750         LD      IX,4389H        ;SET-UP JUMP TO
7297 DD3600C3 00760         LD      (IX),0C3H       ;    THE
729B DD360139 00770         LD      (IX+1),39H      ;        NEW
729F DD360272 00780         LD      (IX+2),72H      ;            PROGRAM
72A3 010700   00790         LD      BC,7    ;CHANGE
72A6 21D772   00800         LD      HL,M1   ;   ASSM
72A9 11EA48   00810         LD      DE,48EAH        ;      PROMPT
72AC EDB0     00820         LDIR    ;    MESSAGE
72AE C37072   00830         JP      DEL     ;BACK TO EDTASM
72B1 DD21A943 00840 EXT     LD      IX,43A9H        ;RESTORE TAPE
72B5 DD3600CD 00850         LD      (IX),0CDH           ;   AND SYNC
72B9 DD218943 00860         LD      IX,4389H        ;RESTORE
72BD DD3600E5 00870         LD      (IX),0E5H       ;    ORIGINAL
72C1 DD3601C5 00880         LD      (IX+1),0C5H     ;        EDTASM
72C5 DD3602D5 00890         LD      (IX+2),0D5H     ;            PROGRAM
72C9 010700   00900         LD      BC,7    ;RESTORE
72CC 21DE72   00910         LD      HL,M2   ;   ASSM
72CF 11EA48   00920         LD      DE,48EAH        ;      PROMPT
72D2 EDB0     00930         LDIR    ;    MESSAGE
72D4 C37072   00940         JP      DEL             ;BACK TO EDTASM
72D7 54       00950 M1      DEFM    'TO STOR'
72DE 43       00960 M2      DEFM    'CASSETT'
0000          00970         END
00000 TOTAL ERRORS
```

26

Menu Program for NEWDOS/80 Version 2

by Dr. Walter J. Atkins, Jr.

System Requirements:
Model I
32K RAM
Level II BASIC
One disk drive
NEWDOS/80 Version 2

I have been using Barry Kornfield's disk menu program (*80 Microcomputing*, November 1980, p. 226) on all my game program disks since it appeared. That program, DIRPICK, reads the directory of the disk drive of your choice, numbers the programs, and runs any BASIC or /CMD file by entering its number. Unfortunately, it doesn't work with the latest version of Apparat's NEWDOS/80 operating system. My program is an adaptation of DIRPICK for use on the NEWDOS/80 Version 2.

Because NEWDOS/80 Version 2 is designed to work with double density as well as with single density disk drives, it uses a different format that allows more files to be displayed on the screen at a time. It displays the disk directory in four columns, of fifteen characters each.

I originally just changed line 230 of DIRPICK to accommodate the new format. The original line is L = 15488 + 64*Y + 20*Z. The variable Z counts the programs across the screen and the variable Y counts the lines on the screen. I changed the line to L = 15488 + 64*Y + 15*Z. I also changed line 200 from FOR Z = 0 TO 2 to FOR Z = 0 TO 3, but this left the screen too crowded.

I fixed that problem when I noticed that there is no need to display the file extensions when picking a program to run. /CMD and /BAS merely tell the computer whether a file is BASIC or machine language. Without the extensions the maximum length of a file name is eight characters, and the screen display is neater. Line 350 uses the INSTR function to find how many characters there are in the file name preceding the slash (/) used to begin the file extension. It then displays all characters up to the extension.

The original DIRPICK program could display a maximum of 31 programs on the screen. This modified version can display and select from a maximum of 48. If you have more than 48 programs on a disk, this program does not work correctly, because NEWDOS/80 Version 2 then displays the disk directory in pages of up to 48 files each.

If you run a single drive system, you must change line 200 to read 200 CMD"DIR". This reads the directory of the disk on drive 0 rather than the disk on drive 1.

Program Listing

```
100 REM  *****************  MENU8Ø  *********************
110 REM   ----- DISK MENU PROGRAM ---------
120 REM   - ADAPTED FROM 8Ø MOCROCOMPUTING NOV 8Ø P226 -
130 REM    --- CODED 25 AUG 1981 ---
140 REM ===== CODED BY DR. WALTER J. ATKINS ============
150 REM **************************************************
160 REM == PROGRAM PURPOSE :  TO PROVIDE A SELECTION
          MENU THAT ALLOWS ANY PROGRAM TO BE SELECTED BY
          NUMBER.
170 REM **************************************************
180 REM
190 CLEAR5ØØ
200 CMD"DIR :1"
21Ø TIT$="AUTOMATIC PROGRAM SELECTION":TL=LEN(TIT$)
22Ø SL=(64-TL)/2:SL=SL-1
23Ø PRINT@Ø,STRING$(SL,"*")+" "+TIT$+" "+STRING$(SL,"*");
24Ø REM
25Ø DIM A$(6Ø)
26Ø FOR Z=Ø TO 3
27Ø L=15488+64*Y+15*Z
28Ø P=P+1
29Ø C=Ø
3ØØ X=PEEK(L+C):IF X<32 THEN X=X+64
31Ø A$(P)=A$(P)+CHR$(X)
32Ø C=C+1
33Ø IF PEEK(L+C)<>32 GOTO 3ØØ
34Ø IF A$(P)=" " GOTO 4ØØ
35Ø ST=Ø:SP=Ø:SP=INSTR(^$(P),"/"):IF SP>Ø THEN ST=SP-1 ELSE ST=L
EN(A$(P))
36Ø IF P<1Ø THEN P$=" "+STR$(P) ELSE P$=STR$(P)
37Ø PRINT@(L-15424),CHR$(3Ø);RIGHT$(P$,LEN(P$)-1);". ";LEFT$(A$(
P),ST)+" ";
38Ø NEXT Z
39Ø Y=Y+1:IF Y>12 THEN 4ØØ  ELSE Z=Ø:GOTO 26Ø
4ØØ PRINTCHR$(31);:PRINT@832," "STRING$(63,"*");
41Ø PRINT@896,"     CHOOSE THE PROGRAM YOU WOULD LIKE BY ITS NUM
BER => ";
42Ø IF P>1Ø GOTO46Ø
43Ø X$=INKEY$:IF X$="" GOTO 43Ø
44Ø X=VAL(X$):IF X<1 OR X>P-1 THEN 4ØØ
45Ø GOTO47Ø
46Ø INPUT X:IF X<1 OR X>P-1 THEN 4ØØ
47Ø IF RIGHT$(A$(X),3)="CMD" THEN CMD A$(X) ELSE RUN A$(X)
```

27

TRANSCRIPT

by Brian Cameron

System Requirements:
Model I
48K RAM
Editor/assembler
Cassette or disk Scripsit
RS232 Interface

Before I could save enough money to buy a disk drive for my 32K Model I system, the tape-based version of Scripsit was the most useful program I had. To get the most from my system, I used the tape-based version of Scripsit as an off-line editor, to edit text and transmit it to the IBM computer where I work. This way I could create files even when the IBM was down or the limited phone lines were taken, and transmit later.

The trick was to modify Scripsit to allow a communications program to interface with it. With my program, TRANSCRIPT, you can create a text file, jump into communications mode, sign on to the computer at work, and transmit the formatted text using the Scripsit P,S (print to serial port) command.

TRANSCRIPT modifies the BREAK S (SAVE) command for tape, and the BREAK S,T command for disk. I used S,T for the disk version so you won't be annoyed by a terminal prompt each time you want to save to disk. Although a save to tape for the disk version of Scripsit is rare, it is still supported.

Normally the SAVE command saves the data to tape or disk. Now, when you invoke it, the program asks if you want to save the text via the terminal mode. Reply N, and a normal save is performed. Reply Y, and you are put in terminal mode. The terminal program lets you sign on to a host system and prepare to transmit data. You can send a break signal to the host computer by pressing BREAK, or the up-arrow key if the BREAK key is disabled.

Having established a link with the host computer, you enter the input mode of a system editor, in order to collect and save the data for transmission. Return to TRANSCRIPT by pressing CLEAR. The screen that was saved away is restored and you can send text to the main computer by issuing the P,S command. The cursor stops flashing temporarily while the data is being transmitted. Type BREAK S again to return to the communication mode and close the file on the host computer.

TRANSCRIPT contains a reverse bit routine to convert uppercase characters to lowercase and vice versa before sending the byte to the serial port. (When you type a lowercase letter on the TRS-80, the computer sends it uppercase, and when you press the shift key, it sends it lowercase.) I also modified the Scripsit program to allow a delay after a carriage return is sent. Without the delay, a few bytes get lost at the beginning of the next line.

When I purchased my disk drive, I refined TRANSCRIPT so it would be useful with both disk and tape versions of Scripsit. To use it, you must have 48K of memory in your system.

The program loads at F0000H. Enter the assembly source code into EDTASM and generate an object file. Then load Scripsit into memory without executing it. If you are running a tape version, you must load a monitor program at the address specified in the TRANSCRIPT patch program. Load and execute the patch program. It modifies Scripsit and moves the communications package, sitting at location F0000H, up against Scripsit. Save the memory addresses specified by the program, and you have the modified version of Scripsit I call TRANSCRIPT. It is no longer necessary to load two programs.

When you load and execute TRANSCRIPT, it moves the packed code above F000H. It also protects memory and prevents TRANSCRIPT text from overwriting the communications code. The tape version of Scripsit sits in memory locations 4300H–69C5H. On an NMI (non-maskable interrupt), memory locations 433FH–434BH are destroyed. This would mean that you could execute Scripsit only once without having to reload. To solve this problem I save the code originally at these addresses and restore it at each reset. The packed code is moved back to location F000H only once, and then the LDIR instruction is set to a 00H (NOP) instruction so it will fall through without moving data.

The modifications for the disk version of Scripsit are minor. I make a check of common memory locations, and determine which version is resident, if any, and apply the appropriate changes.

One change I make to the disk version is to alter the return address of the END command. Normally address 6595H contains 0000H which reboots the system. I add a jump to the display routine to clear the screen, and then go to the warm-start address.

Modifying Scripsit

•Load EDTASM
•L D=TRSCRIPT/SRC:1 (The source for the patch is on drive 1 and called TRSCRIPT)
•A/WE (specify TRSCRIPT/CMD:1 to prompt)
•Reset your system.
•LOAD Scripsit (do not execute).
•TRSCRIPT (will execute the patches)
•Reset your system.
•DUMP TRANS/CMD:1 5200H 8000H 5200H

I use the NEWDOS/80 Version 2 disk operating system, and I cannot guarantee the results with any other operating system. You should be able to make any changes necessary for your particular operating system:

In the procedure for the tape version, you must load a monitor program that will allow you to save the modified version to tape after the changes have been made.

•Load EDTASM
•Load the assembly source code
•Change the monitor jump address
•A/WE (write object to tape)
•Load Scripsit
•Load the object patch program
•Load a monitor
•Run the patch program at F000H
•Save the version of TRANSCRIPT

The monitor program I use is CPU80, an extended version of my CP80 monitor program (*80 Microcomputing*, April 1982), with tape I/O routines. I recommend the MON3 monitor, a relocating monitor you can move to location A000H. You can use MON3 at its normal load address 7000H, but you have to change the source-code listing to reflect the new jump address. If you are using T-BUG, relocate it to a higher address (*80 Microcomputing*, January 1980).

Program Listing

```
F000          01200          ORG   0F000H
002B          01300 KEYBRD   EQU   2BH
0000          01400 DISP     EQU   0000H
401E          01500 VIDRAM   EQU   401EH
0060          01600 DELAY    EQU   0060H
000D          01700 CR       EQU   0DH
00FF          01800 TRUE     EQU   0FFH            ;LOGICAL TRUE
0000          01900 FALSE    EQU   00H             ;LOGICAL FALSE
0000          02000 EOM      EQU   00H
```

```
Ø1C9            Ø21ØØ CLEAR     EQU    Ø1C9H
ØØ2B            Ø22ØØ KBD       EQU    2BH            ;KEYBOARD ROUTINE
ØØ33            Ø23ØØ DSP       EQU    33H
ØØE8            Ø24ØØ RESURT    EQU    ØE8H                ;IN=CONTROL BITS ,OUT=RESET
ØØE9            Ø25ØØ SWITCH    EQU    ØE9H                ;IN=SWITCH , OUT=BAUD RATE
ØØEA            Ø26ØØ CTRL      EQU    ØEAH           ;RS232 CONTROL
ØØEB            Ø27ØØ DATA      EQU    ØEBH           ;RS232 DATA
4Ø2Ø            Ø28ØØ CURPOS    EQU    4Ø2ØH
                Ø29ØØ ; FIND OUT WHICH VERSION OF SCRIPSIT
                Ø3ØØØ ; IS LOADED AND JUMP TO PROPER ZAP
                Ø31ØØ ; CODE
                Ø32ØØ ;
FØØØ 3AØ252     Ø33ØØ           LD     A,(52Ø2H)      ;GRAB A COMMON ADDRESS
FØØ3 FE98       Ø34ØØ           CP     98H            ;IS IT TAPE VERSION?
FØØ5 CAC3FØ     Ø35ØØ           JP     Z,MOVER        ;YES
FØØ8 FE43       Ø36ØØ           CP     43H            ;IS IT DISK VERSION?
FØØA 2825       Ø37ØØ           JR     Z,DMOVE        ;YES
FØØC 2114FØ     Ø38ØØ           LD     HL,LOADS       ;TELL USER
FØØF CD3DF3     Ø39ØØ           CALL   MSGDSP         ;... SCRIPSIT NOT LOADED
FØ12            Ø4ØØØ LOOP      EQU    $
FØ12 18FE       Ø41ØØ           JR     LOOP           ;BUZZ LOOP
FØ14 53         Ø42ØØ LOADS     DEFM   'SCRIPSIT NOT LOADED - RESET'
FØ2F ØD         Ø43ØØ           DEFB   CR
FØ3Ø ØØ         Ø44ØØ           DEFB   EOM
                Ø45ØØ ; THE FOLLOWING CODE WILL ZAP AND MOVE THE COMMUNICATION
                Ø46ØØ ; PROGRAM INTO PLACE SO IT CAN BE SAVED AWAY ...
                Ø47ØØ ; TO BE EXECUTED AS ONE MODULE LATER
                Ø48ØØ ;
                Ø49ØØ ; DISK VERSION
FØ31            Ø5ØØØ DMOVE     EQU    $
FØ31 3EC3       Ø51ØØ           LD     A,ØC3H         ;JUMP COMMAND
FØ33 32745F     Ø52ØØ           LD     (5F74H),A      ;ZAP IT
FØ36 321266     Ø54ØØ          LD      (6612H),A
FØ39 21Ø6F3     Ø55ØØ           LD     HL,DART
FØ3C 221366     Ø56ØØ          LD      (6613H),HL
FØ3F 214AF3     Ø56Ø1          LD      HL,XDOS
FØ42 229565     Ø5611          LD      (6595H),HL
FØ45 21DCF1     Ø57ØØ          LD      HL,OURS
FØ48 22F966     Ø57Ø1          LD      (66F9H),HL
FØ4B 212F52     Ø59ØØ           LD     HL,522FH       ;JUMP ADDRESS
FØ4E 22755F     Ø6ØØØ           LD     (5F75H),HL     ;ZAP
FØ51 3E21       Ø61ØØ           LD     A,21H          ;LOAD COMMAND
FØ53 326752     Ø62ØØ           LD     (5267H),A      ;ZAP ZAP
FØ56 21FFEF     Ø63ØØ           LD     HL,ØEFFFH      ;GET ADDRESS
FØ59 226852     Ø64ØØ           LD     (5268H),HL     ;ZAP ZAP ZAP
FØ5C Ø122ØØ     Ø65ØØ           LD     BC,DEND-DAME   ;GRAB COUNT
FØ5F 11ØØ52     Ø66ØØ           LD     DE,52ØØH       ;DESTINATION
FØ62 2191FØ     Ø67ØØ           LD     HL,DAME        ;SOURCE
FØ65 EDBØ       Ø68ØØ           LDIR                  ;MOVE IT
FØ67 Ø1B1Ø1     Ø69ØØ           LD     BC,STACK-CHECKX     ;COUNT
FØ6A 11ØØ7D     Ø7ØØØ           LD     DE,7DØØH       ;DESTINATION
FØ6D 21D3F1     Ø71ØØ           LD     HL,CHECKX      ;SOURCE
FØ7Ø EDBØ       Ø72ØØ           LDIR                  ;MOVE IT
FØ72 Ø11ØØØ     Ø73ØØ           LD     BC,16          ;COUNT
FØ75 112F52     Ø74ØØ           LD     DE,522FH       ;DESTINATION
FØ78 21B3FØ     Ø75ØØ           LD     HL,CRCK2       ;SOURCE
FØ7B EDBØ       Ø76ØØ           LDIR                  ;MOVE IT
FØ7D Ø11BØØ     Ø77ØØ           LD     BC,27
FØ8Ø 21B8F1     Ø78ØØ           LD     HL,LOGO
FØ83 11F757     Ø79ØØ           LD     DE,57F7H
FØ86 EDBØ       Ø8ØØØ           LDIR
```

Program continued

```
F088 2160F1   08100          LD     HL,DUND
F08B CD3DF3   08102          CALL   MSGDSP
F08E C38EF0   08104 BUZZ     JP     BUZZ
              09100 ;
F091          09200 DAME     EQU    $
F091 21FEEF   09300          LD     HL,0EFFEH         ;PROTECT ...
F094 22B140   09400          LD     (40B1H),HL        ;TOP OF BASIC
F097 22D640   09500          LD     (40D6H),HL        ;AND MEMORY SIZE
F09A 01B101   09600          LD     BC,STACK-CHECKX       ;COUNT
F09D 11D3F1   09700          LD     DE,CHECKX         ;DESTINATION
F0A0 21007D   09800          LD     HL,7D00H          ;SOURCE
F0A3 EDB0     09900          LDIR                     ;MOVE IT
F0A5 3E00     10000          LD     A,00H
F0A7 321252   10100          LD     (5212H),A
F0AA 321352   10200          LD     (5213H),A
F0AD CD1266   10210          CALL    6612H    ;INIT THE UART
F0B0 C33F52   10300          JP     523FH
F0B3          10400 DEND     EQU    $
              10500 ;
F0B3          10600 CRCK2    EQU    $
F0B3 FE0D     10700          CP     0DH               ;IS IT A CR
F0B5 C2785F   10800          JP     NZ,5F78H          ;NO - RETURN
F0B8 C5       10900          PUSH   BC                ;SAVE
F0B9 01FFFF   11000          LD     BC,0FFFFH         ;AMOUNT OF TIME
F0BC CD6000   11100          CALL   DELAY             ;WAIT
F0BF C1       11200          POP    BC                ;RESTORE
F0C0 C37C5F   11300          JP     5F7CH             ;RETURN
              11400 ; TAPE VERSION
F0C3          11500 MOVER    EQU    $
F0C3 3EC3     11600          LD     A,0C3H            ;JUMP COMMAND
F0C5 32294F   11700          LD     (4F29H),A         ;ZAP IT
F0C8 32AE56   11800          LD     (56AEH),A         ;ZAP THE S COMMAND
F0CB 32EA55   11900          LD      (55EAH),A
F0CE 21EEF2   12000          LD     HL,TART
F0D1 22EB55   12100          LD      (55EBH),HL
F0D4 21D3F1   12200          LD     HL,CHECKX
F0D7 22AF56   12300          LD     (56AFH),HL
F0DA 212F43   12400          LD     HL,432FH          ;JUMP ADDRESS
F0DD 222A4F   12500          LD     (4F2AH),HL
F0E0 3E21     12600          LD     A,21H             ;LD COMMAND
F0E2 326443   12700          LD     (4364H),A         ;ZAP IT
F0E5 21FFEF   12800          LD     HL,0EFFFH         ;GET ADDRESS
F0E8 226543   12900          LD     (4365H),HL        ;ZAP IT
F0EB 012D00   13000          LD     BC,SEND-SAME      ;GRAB THE COUNT
F0EE 110043   13100          LD     DE,4300H          ;DESTINATION
F0F1 218BF1   13200          LD     HL,SAME           ;SOURCE
F0F4 EDB0     13300          LDIR                     ;MOVE IT
F0F6 01B101   13400          LD     BC,STACK-CHECKX       ;COUNT
F0F9 11006A   13500          LD     DE,6A00H          ;DESTINATION
F0FC 21D3F1   13600          LD     HL,CHECKX         ;SOURCE
F0FF EDB0     13700          LDIR                     ;MOVE IT
F101 011B00   13800          LD     BC,27
F104 21B8F1   13900          LD     HL,LOGO
F107 11F448   14000          LD     DE,48F4H
F10A EDB0     14100          LDIR
F10C          14200 SAVEM    EQU    $
F10C 211BF1   14300          LD     HL,DUNT
F10F CD3DF3   14310          CALL   MSGDSP
F112          14320 KWT      EQU    $
F112 CD2B00   14330          CALL   KBD
F115 B7       14340          OR     A
```

```
F116 28FA       1435Ø         JR      Z,KWT
                1436Ø ;REPLACE ADDRESS OF NEXT LINE
                1437Ø ;WITH THE ADDRESS OF YOUR MONITOR
F118 C3ØØAØ     1438Ø         JP       ØAØØØH
F11B 53         146ØØ DUNT    DEFM 'SAVE MEMORY 43ØØ TO 6CØØ FOR TAPE VERSION'
F144 ØD         147ØØ          DEFB  CR
F145 5Ø         148ØØ          DEFM 'PRESS ANY KEY TO CONTINUE'
F15E ØD         149ØØ          DEFB  ØDH
F15F ØØ         15ØØØ          DEFB  ØØH
F16Ø 52         151ØØ DUND     DEFM 'RESET SYSTEM AND DUMP MEMORY 52ØØ TO 8ØØØ'
F189 ØD         152ØØ          DEFB  CR
F18A ØØ         153ØØ          DEFB  ØØH
F18B            154ØØ SAME     EQU   $
F18B 21FEEF     155ØØ          LD    HL,ØEFFEH        ;PROTECT ...
F18E 22B14Ø     156ØØ          LD    (4ØB1H),HL       ;TOP OF BASIC
F191 22D64Ø     157ØØ          LD    (4ØD6H),HL       ;AND MEMORY SIZE
F194 Ø1B1Ø1     158ØØ          LD    BC,STACK-CHECKX      ;COUNT
F197 11D3F1     159ØØ          LD    DE,CHECKX        ;DESTINATION
F19A 21ØØ6A     16ØØØ          LD    HL,6AØØH         ;SOURCE
F19D EDBØ       161ØØ          LDIR
F19F Ø11FØØ     162ØØ          LD    BC,OURE-OURCK  ;LENGTH
F1A2 112F43     163ØØ          LD    DE,432FH         ;DESTINATION
F1A5 21lEF3     164ØØ          LD    HL,OURCK         ;SOURCE
F1A8 EDBØ       165ØØ          LDIR
F1AA 3EØØ       166ØØ          LD    A,ØØH            ;NOP OUT ...
F1AC 321243     167ØØ          LD    (4312H),A        ;THE LDIR
F1AF 321343     168ØØ          LD    (4313H),A        ;INSTRUCTION
F1B2 CDEA55     1681Ø         CALL    55EAH
F1B5 C33F43     169ØØ          JP    433FH
F1B8            17ØØØ SEND     EQU   $
                171ØØ ;
F1B8 54         172ØØ LOGO     DEFM 'TRANSCRIPT BY BRIAN CAMERON'
F1D3            173ØØ CHECKX   EQU   $
F1D3 FE53       174ØØ          CP    'S'              ;IS IT SAVE TAPE?
F1D5 28Ø5       175ØØ          JR    Z,OURS           ;YES - OUR SAVE FIRST
F1D7 F5         176ØØ          PUSH  AF
F1D8 F1         18ØØØ          POP   AF               ;RESTORE REG
F1D9 C3B256     181ØØ          JP    56B2H            ;RETURN TO SCRIPSIT
                185ØØ ;
F1DC            186ØØ OURS     EQU   $
F1DC F5         187ØØ          PUSH  AF
F1DD C5         188ØØ          PUSH  BC
F1DE D5         189ØØ          PUSH  DE
F1DF E5         19ØØØ          PUSH  HL
F1EØ Ø1ØØØ4     191ØØ          LD    BC,4ØØH          ;GET THE COUNT
F1E3 11ØØFC     192ØØ          LD    DE,ØFCØØH        ;POINT TO DEST
F1E6 21ØØ3C     193ØØ          LD    HL,3CØØH         ;POINT TO SOURCE
F1E9 EDBØ       194ØØ          LDIR
F1EB            198ØØ MDSP     EQU   $
F1EB 2126F2     199ØØ          LD    HL,TERMM         ;GET MESSAGE
F1EE CDC86B     2ØØØØ         CALL    6BC8H   ;DISPLAY AT BOTTOM
F1F1            2Ø1ØØ MLP1     EQU   $
F1F1 CD2BØØ     2Ø2ØØ          CALL  KBD              ;GET ANSWER
F1F4 B7         2Ø3ØØ          OR    A                ;ANYTHING?
F1F5 28FA       2Ø4ØØ         JR      Z,MLP1  ;NO RETRY
F1F7 CBAF       2Ø43Ø         RES     5,A     ;INSURE UPPER CASE
F1F9 FE59       2Ø5ØØ          CP    'Y'              ;IS IT YES?
F1FB 284B       2Ø6ØØ         JR      Z,PRETOP
F1FD FE4E       2Ø7ØØ          CP    'N'              ;IS IT NO?
F1FF 2ØEA       2Ø8ØØ          JR    NZ,MDSP          ;INVALID - TRY AGAIN
F2Ø1 3EØF       2Ø9ØØ          LD    A,ØFH            ;TURN CURSOR ...
```

Program continued

```
F2Ø3 CD33ØØ    21ØØØ          CALL  DSP          ;OFF
F2Ø6 Ø1ØØØ4    211ØØ          LD    BC,4ØØH      ;LENGTH
F2Ø9 11ØØ3C    212ØØ          LD    DE,3CØØH     ;DESTINATION
F2ØC 21ØØFC    213ØØ          LD    HL,ØFCØØH    ;SOURCE
F2ØF EDBØ      214ØØ          LDIR
F211 3AC556    215ØØ          LD    A,(56C5H)    ;GET COMMON BYTE
F214 FECD      216ØØ          CP    ØCDH         ;IS IT TAPE?
F216 28Ø7      217ØØ          JR    Z,DISKE
F218 E1        218ØØ          POP   HL
F219 D1        219ØØ          POP   DE
F21A C1        22ØØØ          POP   BC
F21B F1        221ØØ          POP   AF
F21C C34F63    222ØØ          JP     634FH
               223ØØ ;
F21F           224ØØ DISKE    EQU   $
F21F E1        225ØØ          POP   HL
F22Ø D1        226ØØ          POP   DE
F221 C1        227ØØ          POP   BC
F222 F1        228ØØ          POP   AF
F223 C3C256    229ØØ          JP    56C2H
               23ØØØ ;
F226           231ØØ TERMM    EQU   $
F226 2Ø        2311Ø          DEFM ' '
F227 54        232ØØ          DEFM 'TERM MODE?'
F231 2Ø        234ØØ          DEFM ' (Y/N)'
F237 2Ø        2341Ø          DEFM '                    '
F247 ØØ        236ØØ          DEFB  EOM
               237ØØ ;
F248           2371Ø PRETOP   EQU     $
F248 CDC9Ø1    2372Ø          CALL    1C9H   ;CLEAR AND HOME
F24B 3EØE      23721          LD      A,ØEH  ;TURN CURSOR ON
F24D CD33ØØ    23731          CALL    DSP
F25Ø           238ØØ TOP      EQU   $
F25Ø CD2BØØ    239ØØ          CALL  KBD          ;SCAN KEYBOARD
F253 B7        24ØØØ          OR    A            ;ANYTHING?
F254 2838      241ØØ          JR    Z,CKIN       ;NO - CHECK INPUT
F256 FEØA      242ØØ          CP    ØAH          ;DOWN ARROW?
F258 2834      243ØØ          JR    Z,CKIN       ;YES IGNORE FOR NOW
F25A 214Ø38    244ØØ          LD    HL,384ØH     ;GET ROW
F25D CB66      245ØØ          BIT   4,(HL)       ;TEST CONTROL KEY
F25F 28Ø4      246ØØ          JR    Z,NOTCTL     ;NOT DOWN
F261 CBB7      247ØØ          RES   6,A          ;MAKE CONTROL
F263 1824      248ØØ          JR    NBRK         ; SHOW AND TELL
F265           249ØØ NOTCTL   EQU   $
F265 CDAAF2    25ØØØ          CALL  REVBIT       ;REVERSE BITS
F268 FE1F      251ØØ          CP    1FH          ;IS IT A CLEAR?
F26A CAC9F2    252ØØ          JP    Z,EXIT      ;JUMP TO CMD HANDLER
F26D FEØ1      2521Ø          CP     Ø1H     ;IS IT BREAK
F26F 28Ø4      2522Ø          JR     Z,BREAK ;YES
F271 FE5B      253ØØ          CP     5BH
F273 2Ø14      254ØØ          JR    NZ,NBRK         ;NO - TRANS CHAR
F275           255ØØ BREAK    EQU   $
F275 3A5ØF3    256ØØ          LD    A,(CTRLS)    ;GET READY TO BREAK
F278 CB97      257ØØ          RES   2,A              ;TURN ON BREAK
F27A D3EA      258ØØ          OUT   (CTRL),A     ;SEND BREAK
F27C Ø14A35    259ØØ          LD    BC,354AH     ;SET WAIT TIME
F27F CD6ØØØ    26ØØØ          CALL  DELAY        ;GO WAIT
F282 3A5ØF3    261ØØ          LD    A,(CTRLS)   ;TURN OFF BREAK
F285 D3EA      262ØØ          OUT   (CTRL),A     ;SEND RESET
F287 18Ø5      263ØØ          JR    CKIN         ;CHECK FOR INPUT
F289           264ØØ NBRK     EQU   $
```

```
F289 CD33ØØ                    CALL  DSP             ;ECHO
F28C              DSPIT        EQU   $
F28C D3EB                      OUT   (DATA),A
                  ;
F28E              CKIN         EQU   $
F28E DBEA                      IN    A,(CTRL)        ;GET STATUS
F29Ø CB7F                      BIT   7,A             ;ANYTHING WAITING?
F292 CA5ØF2                    JP    Z,TOP           ;NO - RETURN
F295 DBEB                      IN    A,(DATA)        ;GET BYTE FROM LINE
F297 FEØA                      CP    ØAH             ;IS IT A LF?
F299 28F3                      JR    Z,CKIN             ;YES - IGNORE
F29B FE7F                      CP    7FH             ;IS IT A DEL
F29D 28EF                      JR    Z,CKIN             ;YES - IGNORE
F29F FEØ8                      CP    Ø8H             ;IS IT BKSP?
F2A1 2ØØ2                      JR    NZ,TBELL          ;NO - CONTINUE
F2A3 3E18                      LD    A,18H           ;CHANGE TO OUR BKSP
F2A5              TBELL        EQU   $
F2A5 CD33ØØ                    CALL  DSP             ;ECHO
F2A8 18E4                      JR    CKIN          ;TRY AGAIN
                  ;-----------------------------------------------
                  ;            REVBIT - ROUTINE TO CONVERT
                  ;                     UPPER CASE TO LOWER
                  ;                     AND LOWER TO UPPER
                  ;-----------------------------------------------
                  ;
F2AA              REVBIT       EQU   $
F2AA FE41                      CP    41H             ;CAPS A?
F2AC FAC8F2                    JP    M,NOREV
F2AF FE7B                      CP    7BH             ;LOW CASE Z?
F2B1 F2C8F2                    JP    P,NOREV
F2B4 FE5B                      CP    5BH             ;SQUARE BRACKET?
F2B6 281Ø                      JR    Z,NOREV
F2B8 F2BDF2                    JP    P,TESTUL
F2BB 18Ø7                      JR    REV             ;REVERSE CODE
F2BD              TESTUL       EQU   $
F2BD FE6Ø                      CP    6ØH             ;ACCENT GRAVE
F2BF 28Ø7                      JR    Z,NOREV
F2C1 FAC8F2                    JP    M,NOREV
F2C4              REV          EQU   $
F2C4 EE2Ø                      XOR   2ØH             ;REVERSE CASE
F2C6 CBAF                      RES   5,A             ;CHANGE TO UPPER CASE
F2C8              NOREV        EQU   $
F2C8 C9                        RET
F2C9              EXIT         EQU   $
F2C9 3EØF                      LD    A,ØFH           ;TURN CURSOR OFF
F2CB CD33ØØ                    CALL  DSP
F2CE Ø1ØØØ4                    LD    BC,4ØØH         ;SET COUNT
F2D1 11ØØ3C                    LD    DE,3CØØH        ;POINT TO DESTINATION
F2D4 21ØØFC                    LD    HL,ØFCØØH
F2D7 EDBØ                      LDIR
F2D9 3AC556                    LD    A,(56C5H)       ;FIND OUT IF TAPE OR DISK
F2DC FECD                      CP    ØCDH            ;IS IT TAPE?
F2DE 28Ø7                      JR    Z,TAPEE         ;NO
F2EØ E1                        POP   HL
F2E1 D1                        POP   DE
F2E2 C1                        POP   BC
F2E3 F1                        POP   AF
F2E4 C38B6F                    JP    6F8BH
F2E7              TAPEE        EQU   $
F2E7 E1                        POP   HL
F2E8 D1                        POP   DE
```

Program continued

```
F2E9 C1                    POP    BC
F2EA F1                    POP    AF
F2EB C3FA5E                JP     5EFAH
                  ;
F2EE              TART     EQU    $
F2EE D3E8                  OUT    (RESURT),A
F2FØ DBE9                  IN     A,(SWITCH)
F2F2 3251F3                LD     (SWTCHS),A
F2F5 E6F8                  AND    ØF8H
F2F7 F6Ø5                  OR     Ø5H
F2F9 325ØF3                LD     (CTRLS),A
F2FC D3EA                  OUT    (CTRL),A
F2FE 3A51F3                LD     A,(SWTCHS)
F3Ø1 E6Ø7                  AND    Ø7H
F3Ø3 C3FE55                JP     55FEH
                  ;
F3Ø6              DART     EQU    $
F3Ø6 D3E8                  OUT    (RESURT),A
F3Ø8 DBE9                  IN     A,(SWITCH)
F3ØA 3251F3                LD     (SWTCHS),A
F3ØD E6F8                  AND    ØF8H
F3ØF F6Ø5                  OR     Ø5H
F311 325ØF3                LD     (CTRLS),A
F314 D3EA                  OUT    (CTRL),A
F316 3A51F3                LD     A,(SWTCHS)
F319 E6Ø7                  AND    Ø7H
F31B C32666                JP     6626H
                  ;
F31E              OURCK    EQU    $
F31E FEØD                  CP     ØDH           ;IS IT CR?
F32Ø C22D4F                JP     NZ,4F2DH      ;NO - RETURN
F323 C5                    PUSH   BC            ;SAVE REG
F324 Ø1FFFF                LD     BC,ØFFFFH     ;AMOUNT OF TIME
F327 CD6ØØØ                CALL   DELAY         ;WAIT
F32A C1                    POP    BC            ;RESTORE
F32B C3314F                JP     4F31H         ;RETURN
F32E 31FC41                LD     SP,41FCH
F331 3EØA                  LD     A,ØAH
F333 32E837                LD     (37E8H),A
F336 AF                    XOR    A
F337 32A36B                LD     (6BA3H),A
F33A 32A16B                LD     (6BA1H),A
F33D              OURE     EQU    $
F33D              MSGDSP   EQU    $
F33D 7E                    LD     A,(HL)
F33E B7                    OR     A
F33F 23                    INC    HL
F34Ø CA49F3                JP     Z,LP1
F343 CD33ØØ                CALL   DSP
F346 C33DF3                JP     MSGDSP
F349              LP1      EQU    $
F349 C9                    RET
                  ;
F34A              XDOS     EQU    $
F34A CDC9Ø1                CALL   1C9H     ;CLEAR AND HOME
F34D C32D4Ø                JP     4Ø2DH    ;WARM START DOS
                  ;
F35Ø ØØ           CTRLS    DEFB   ØØH
F351 ØØ           SWTCHS   DEFB   ØØH
ØØ32                       DEFS   5Ø
F384              STACK    EQU    $
FØØØ                       END    ØFØØØH
```

28

Tiny Pascal for Disk

by David R. Goben

System Requirements:
Model I
32K RAM
Disk BASIC
NEWDOS/80
One disk drive

Tiny Pascal gives you much of the power of Pascal, at low cost. Radio Shack sells Tiny Pascal for the 16K and 32K Model I. Unfortunately, it's a tape-based version that cannot perform as is from disk. John B. Harrell came to the rescue with alterations so Tiny Pascal can operate on a 32K disk system (*80 Microcomputing*, July 1981). I have modified and expanded Harrell's program to run faster on my 48K Model I, and operate all functions without complicated commands. I have also added printer capability and an option to convert tab characters created in the Pascal editor to three spaces, for easier reading on a program such as Scripsit.

The three BASIC programs in this chapter run unchanged on a NEWDOS/80 system, Versions 1 and 2, for the Model I. If you have another operating system, you must modify the BASIC code. Program Listing 1 is the introductory page; save it under the name INTRO. It runs on DOS boot. Program Listing 2 loads a program to memory, and to a printer if desired, from disk, then runs the Tiny Pascal program. Save it under the name LOAD. Program Listing 3 saves a Pascal program from memory to disk; save it under SAVE.

A user's manual explaining functions and options for the entire package is at the end of this chapter. It includes instructions for developing or editing Pascal programs with Scripsit. The LOAD program accepts Scripsit-created files. The Scripsit program is optional, since Tiny Pascal contains its own text editor. However, Tiny Pascal's editor is line-oriented, simple, and limited. You can use any other word processing program, such as Electric Pencil, or a BASIC program, in place of Scripsit, as long as it saves the entire text file to disk in ASCII format and concludes with a unique character recognized by LOAD.

LOAD recognizes 0FFH (255), the Tiny Pascal end-of-file marker, and Scripsit's block-end character, 1BH (27). Refer to line 9 of the LOAD listing for the recognition coding. You can use any unique ASCII character, other than 1BH, produced by the word processing program. A 35-track disk easily holds NEWDOS/80, the Tiny Pascal program, three utility programs, and a word processor, such as Scripsit.

The user's manual assumes you have a disk with the following programs loaded on it: NEWDOS/80, PASCAL/CMD (Modified PAS32K), SCRIPSIT/CMD (with patches for NEWDOS/80), and the three BASIC programs, INTRO, LOAD, and SAVE.

Patching PAS32K for PASCAL/CMD

Here's how to modify the tape-based PAS32K program to disk. Novices at NEWDOS/80's LMOFFSET and SUPERZAP programs should be able to use them without any experience. This should be a one-time operation. Have ready a NEWDOS/80 system disk with all visible files except Scripsit and the three BASIC programs removed. These patches use Harrell's Z80 assembly source.

Boot NEWDOS/80 and load LMOFFSET. Load the PAS32K tape to the recorder, set the volume, press PLAY, and type T and press ENTER. When the PAS32K program is loaded, the display shows that the program loads from 4D90–73C6. Type 7000, the new module load address, and press ENTER. Reply Y to the SUPPRESS APPENDAGE? question. You are now told that the module loads from 7000–9636. Answer by pressing ENTER.

If you have Version 2 of NEWDOS/80, you are asked if the destination is disk or tape. Make sure that you have the appropriate disk ready to accept the file, then type D and press ENTER. After the prompt, type PASCAL/CMD:d *and press ENTER, where :d is the drive number. If you* have Version 1, reboot DOS. If you have Version 2, type N and press ENTER after the next two prompts.

When the next READY prompt appears, type LOAD PASCAL/CMD and press ENTER. Then type DUMP PASCAL/CMD 7000H 969CH 9637H and press ENTER. After the READY prompt, type SUPERZAP and press ENTER. Type DFS and press ENTER. Type PASCAL/CMD and press ENTER. Type 38 and press ENTER, for the relative sector. When the sector is displayed, type MODD3. When you see the blinking block cursor over location D3, type the following lines:

```
            F3  AF  21  D2  06  11  00  40  01  36  00  ED  B0
3D  3D  20  F1  06  27  12  13  10  FC  11  80  40  21  F7  18
01  27  00  ED  B0  21  00  70  11  90  4D  01  37  26  ED  B0
```

The last two-hex-number entry should force the cursor to location 00. If not, recheck your entries.

When the cursor blinks over location 00, press ENTER and reply Y to the prompt. Press ENTER again, and the modified page is displayed. Press the plus sign (+), and page 39 is displayed. Type MOD00 and press ENTER, and the block cursor appears over location 00.

Type the following patch. When done, the next four hex digits should be 02 02. If not, check your work. Do not type the asterisks (*) surrounding three of the listed entries. If you have a 48K system, change BF to FF, and the two 28s to 68.

```
 21   72   96   11   D5  *BF*  01   2B   00   ED   B0   C3   90   4D   21   00
 98   11   F0   73   01   00  *28*  ED   B0   3E   FF   21   F0   73   01   00
*28*  22   80   41   22   8C   41   ED   B1   22   84   41   22   86   41   22
 96   41   2B   22   82   41   C3   3A   47
```

When the cursor blinks over the zero in the first 02, press ENTER, answer Y, and press ENTER again when prompted. When page 39 is displayed again, press K. Answer the prompt with 0 and press ENTER. When page 0 is displayed, type MOD10. When the cursor appears, for 32K machines type D5BF and press ENTER, and for 48K machines type D5FF and press ENTER. Answer Y and press ENTER when prompted. When the modified page 0 is displayed, press X to exit the mode, then type EXIT and press ENTER to leave SUPERZAP. To test your work, after the DOS READY prompt, type PASCAL and press ENTER, and Tiny Pascal should be up and running.

Tiny Pascal User's Manual

Here is a detailed list of operating instructions for the disk options of Radio Shack's Tiny Pascal package. Refer to all data related to the Pascal program itself in the *Tiny Pascal User's Manual.*

Running INTRO

When the Pascal disk is booted, an AUTO file, BASIC RUN "INTRO", initializes the system. A menu of seven choices is displayed (see Figure 1).

Option 1 loads the Tiny Pascal program with an empty buffer.

Option 2 loads a Pascal program from disk and runs Pascal.

Option 3 saves a program to disk and reruns INTRO.

Option 4 loads and runs Scripsit.

Option 5 asks which disk directory to read, reads it, and asks you to press ENTER to redisplay the menu.

Option 6 allows you to run DOS command strings from INTRO.

Option 7 leaves INTRO and returns to the DOS command mode.

Running LOAD

To answer yes in LOAD and SAVE, type Y or Yes. To answer no, type N or No or press ENTER.

When you run LOAD, you are asked for the source file's filespec, then whether the file is resident on a currently mounted disk. If you answer no, you are prompted to mount it. If you have a one disk system, the source program must be on a disk with the same operating system as the Pascal package. If you answer X, the filespec request repeats, which is useful when you enter an erroneous filename. You are next asked if you want to print the file.

If you answer Y when the printer is not ready, you are prompted, and the question is repeated. When the printer is ready, you are asked how many lines long each page is. If your pages hold a maximum of 66 lines, you would type 66 and press ENTER. You are next asked if you want pauses between pages. Last, you are asked if you want data sent to the line printer only. If so, after printing, INTRO runs rather than PASCAL/CMD.

The appropriate PASCAL program is loaded to RAM, and listed one line at a time to the screen (and printer, if specified). After the program is loaded, the Pascal file is run unless you need to mount another disk to load the file. In that case you are prompted to reload the Pascal disk to drive 0. Once you do this and press ENTER, Tiny Pascal loads and runs.

```
** TINY * PASCAL ** Disk Version

Pascal Menu

<1>  PASCAL program without data
<2>  *LOAD a PASCAL source program & run PASCAL
<3>  *SAVE a PASCAL program to disk
<4>  Load SCRIPSIT to edit or create files
<5>  Do a DIRECTORY read with 'A' option
<6>  Do a DOS command
<7>  Exit BASIC to DOS command mode

*Note that you must RE-BOOT DOS (which returns to this program) from PASCAL to
SAVE (or LOAD) a file via disks.

Key in your choice <1-7>?
```

Figure 1. *Screen display of Pascal menu*

Running SAVE

To run SAVE while in Tiny Pascal, reboot DOS, and INTRO reruns. Answer the prompt with option 3. When SAVE is run, you are asked if you want the file printed. If you answer yes, you are asked (as in LOAD) for the page length and whether you want pauses between pages. Next, you are asked if you want the file sent to the line printer only. If you answer yes, skip the next paragraph. A no answer sends the file to the printer, disk, and screen.

If you answered no to the print question, you are asked for the name of the program, then whether the destination disk is currently mounted. If you answer X, the filespec request repeats. If you answer no, you are prompted to load the destination disk. The drive 0 disk must contain the same operating system as the Pascal disk.

You are asked if you want the TAB characters (09H), used in Tiny Pascal's editor, converted to three spaces by the editor's display driver. This option is handy if you want to use Scripsit to edit a program created or edited by Pascal's editor. Otherwise, Scripsit prints each TAB character as a single left arrow. The program is displayed on the screen one line at a time, to show that it is being loaded to disk properly. Once the program is loaded, INTRO reruns.

Line-Print from Tiny Pascal

If you want data that is normally sent to the monitor printed instead, include the two procedures in Figure 2 in your Pascal program. The first diverts video output to the printer driver, and the second resets the video driver to normal.

```
PROC LPRINT; (* DIVERT VIDEO TO PRINTER *)
  BEGIN
    MEMW(%401E):=%058D;
  END;

PROC VIDEO; (* RESET VIDEO OUTPUT TO VIDEO *)
  BEGIN
    MEMW(%401E):=%0458;
  END;
```

Figure 2

Handling Pascal Files with Scripsit

Use uppercase (SHIFT@ command) to type programs if you have the lowercase modification. If you want a blank line separating program blocks, insert a space before pressing ENTER. Tiny Pascal must have data on a line, even if it is only a blank space. Do not include any special Scripsit printer options in the file. Tiny Pascal sees them as illegal entries.

LOAD recognizes Tiny Pascal files, and Scripsit files that meet specifications. Two other points are important. First, the Pascal program ends with an end-block character, followed by a carriage return. When the last line has been typed and entered, press control-block, control-end, ENTER, CLEAR. Second, the file *must* be saved in ASCII format. If you want to load it to a file that Scripsit has called from disk, press BREAK, type S,A and press ENTER. If it is a new file, or you want to change the file name, enter the command mode via BREAK. Then type S,A and press space and type the filename you want for the Pascal program.

Special Notes

It is a good idea to put a /PAS extension on Pascal filenames, for easy recognition. If you are editing in Scripsit an existing Pascal program which was loaded to disk in the SAVE program, you will note two graphics blocks at the end of the program listing, followed by a Scripsit carriage return character. If you modified the program listing and want to save the newer version to disk, you must delete the three graphics characters and add the end-block and carriage return characters as described earlier.

As an aid in writing the source programs, before keying in data, press BREAK and type T=3,6,9,12,15,18,21, then press ENTER. This allows you to use control-right arrow to tab three spaces at a time.

Program Listing 1. INTRO

```
1   '  I N T R O
2   '
3   POKE &H4ØB1,&HFF:
    POKE &H4ØB2,&H97:
    ON  ERROR  GOTO 27:
    CLS :
    PRINT "** TINY * PASCAL ** Disk version":
    PRINT
4   PRINT  TAB( 8);"Pascal Menu
5   PRINT "<1>        PASCAL program without data
6   PRINT "<2>       *LOAD a PASCAL source program & run PA
    SCAL
7   PRINT "<3>       *SAVE a PASCAL program to disk
8   PRINT "<4>        Load SCRIPSIT to edit or create files
9   PRINT "<5>        Do a DIRECTORY read with 'A' option
1Ø  PRINT "<6>        Do a DOS Command
11  PRINT "<7>        Exit BASIC to DOS Command mode":
    PRINT
12  PRINT "*Note that you must RE-BOOT DOS (which returns t
    o this program)
13  PRINT " from PASCAL to SAVE (or LOAD) a file via disks.
    ":
    PRINT
14  INPUT "Key in your choice <1-7>";CH$
15  IF  VAL (CH$) < 1 OR  VAL (CH$) > 7
       THEN
          PRINT "Wrong choice. Please try again":
             FOR
                X = 1 TO 75Ø:
             NEXT :
          GOTO 3
16  ON  VAL (CH$) GOTO 17,18,19,26,2Ø,28,25
17  CMD "S=PASCAL"
18  RUN "LOAD"
```

```
19  RUN "SAVE"
2Ø  INPUT "Which Drive <Ø-3>";D$
21  IF  VAL (D$) < Ø OR  VAL (D$) > 3
       THEN
          2Ø
22  D$ = "DIR " + D$ + " A":
    CMD "D$"
23  PRINT :
    INPUT "Press ENTER to redisplay Menu";CH$
24  GOTO 3
25  CLS :
    CMD "S"
26  CMD "S=SCRIPSIT"
27  CMD "E":
    FOR
       V = 1 TO 75Ø:
    NEXT :
    RESUME 3
28  CLS :
    LINE  INPUT "Key in Command string: ";ST$:
    IF ST$ = ""
       THEN
          3
       ELSE
    CMD "ST$":
    GOTO 23
```

Program Listing 2. LOAD

```
1  '  L O A D
2  '
3  POKE &H4ØB1,&HFF:
   POKE &H4ØB2,&H97:
   CLS :
   CLEAR 3ØØ:
   DEFINT A - Z:
   AD = &H98ØØ:
   PRINT  TAB( 14)"** TINY * PASCAL ** File Loader":
   PRINT
4  LINE  INPUT "Enter Filespec for Source File: ";FS$:
   IF FS$ = ""
      THEN
         RUN "INTRO"
      ELSE
   A$ = "":
   PRINT "Is Source on a mounted disk (if not, then it mus
   t be on a":
   INPUT "System Disk)";A$:
   IF A$ = "X"
      THEN
         4
      ELSE
   IF  LEFT$ (A$,1) = "N"
      THEN
         INPUT "Mount Source Disk and press ENTER";
5  T$ = "":
   INPUT "Output data to Line Printer";T$:
   IF  LEFT$ (T$,1) = "Y"
```

Program continued

```
      THEN
         IF  PEEK (14312) <  > 63
            THEN
               PRINT "Line Printer NOT ready!":
               PRINT :
               GOTO 5
            ELSE
         INPUT "What is your page leanth";LL:
         IF LL < 7
            THEN
               17
            ELSE
         LL = LL - 6:
         L = LL:
         INPUT "Pause between pages";G$
6   U$ = "":
    IF  LEFT$ (T$,1) = "Y"
       THEN
    PRINT :
    PRINT "Loading *** "FS$" *** from disk":
    IF  LEFT$ (U$,1) = "Y"
       THEN
          PRINT "* Data to Printer ONLY *":
          PRINT
       ELSE
    PRINT
8   L$ = "":
    LINE  INPUT #1,L$:
    IF  LEFT$ (T$,1) = "Y"
       THEN
          LPRINT  TAB( 6);L$:
          L = L - 1:
          IF L = Ø
             THEN
                GOSUB 16:
                GOSUB 15
9   PRINT L$:
    G =  ASC ( LEFT$ (L$,1)):
    IF G = 255 OR G = 27
       THEN
          11
       ELSE
    IF  LEFT$ (U$,1) = "Y"
       THEN
          1Ø
       ELSE
    FOR
       J = 1 TO  LEN (L$):
       POKE AD, ASC ( MID$ (L$,J,1)):
       AD = AD + 1:
    NEXT :
    POKE AD,13:
    AD = AD + 1
1Ø  POKE 143Ø4, PEEK (17161):
    GOTO 8
11  CLOSE :
    POKE AD,255:
    POKE AD + 1,255:
    IF A$ = "N"
       THEN
          INPUT "Mount PACAL Disk and press ENTER";
```

```
12  IF  LEFT$ (T$,1) = "Y"
       THEN
          GOSUB 15:
          IF  LEFT$ (U$,1) = "Y"
             THEN
                RUN "INTRO"
13  CMD "S=PASCAL"
14  CLS :
    CMD "E":
    FOR
       V = 1 TO 75Ø:
    NEXT :
    RESUME 3
15  LPRINT " "; CHR$ (13);" "; CHR$ (13);" "; CHR$ (13):
    RETURN
16  GOSUB 15:
    CLS :
    IF  LEFT$ (G$,1) <  > "Y"
       THEN
          RETURN
       ELSE
    INPUT "Press ENTER to continue";:
    L = LL:
    RETURN
17  PRINT "Illegal entry. Try this part again.":
    FOR
       X = 1 TO 75Ø:
    NEXT :
    GOTO 5
```

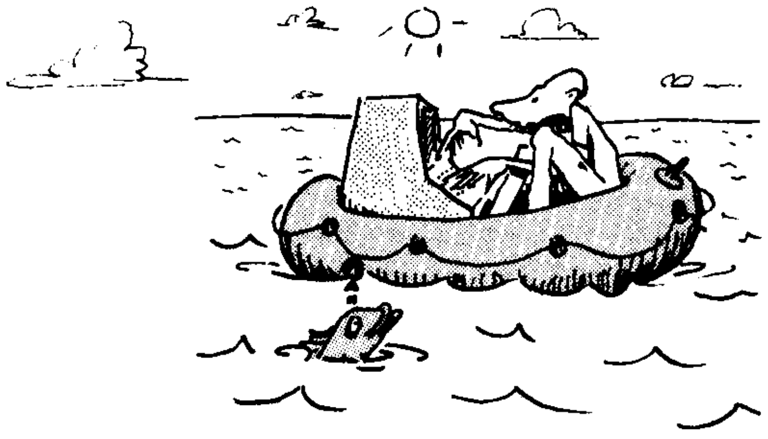
Program Listing 3. SAVE

```
1  '  S A V E
2  '
3  POKE &H4ØB1,&HEF:
   POKE &H4ØB2,&H73:
   CLEAR 3ØØ:
   CLS :
   DEFINT B - Z:
   PRINT  TAB( 16)"** TINY * PASCAL ** File Saver":
   PRINT
4  INPUT "Output data to Line Printer";T$:
   IF  LEFT$ (T$,1) = "Y"
      THEN
         IF  PEEK (14312) <  > 63
            THEN
               PRINT "Line Printer NOT ready!":
               PRINT :
               GOTO 3
            ELSE
         INPUT "What is your page leanth";LL:
         IF LL < 7
            THEN
               4
            ELSE
         LL = LL - 6:
         L = LL:
         INPUT "Pause between Pages";G$
```

Program continued

```
5   IF  LEFT$ (T$,1) = "Y"
       THEN
          INPUT "Data to Line Printer only";U$:
          U$ =  LEFT$ (U$,1)
6   INPUT "Should TABs be converted to 3 SPACES";J$:
    IF  LEFT$ (T$,1) = "Y"
       THEN
          GOSUB 22:
          IF U$ = "Y"
             THEN
                9
7   LINE  INPUT "Enter Filespec for PASCAL File: ";FS$:
    IF FS$ = ""
       THEN
          RUN "INTRO"
       ELSE
    ON  ERROR  GOTO 19:
    R$ = "":
    INPUT "Is Destination Disk mounted on Drive";R$:
    IF  LEFT$ (R$,1) = "N"
       THEN
          GOSUB 2Ø
       ELSE
    IF R$ = "X"
       THEN
          7
8   OPEN "O",1,FS$
9   AD = &H73FØ:
    PRINT :
    IF U$ = "Y"
       THEN
          PRINT "Putting *** "FS$" *** to Printer":
          PRINT
       ELSE
    PRINT "Saving *** "FS$" *** to disk":
    PRINT
1Ø  A$ = ""
11  J =  PEEK (AD):
    IF J = 255
       THEN
          16
       ELSE
    AD = AD + 1:
    IF AD > 32767
       THEN
          AD = AD - 65536
12  IF J = 9
       THEN
          IF  LEFT$ (J$,1) = "Y"
             THEN
                A$ = A$ +  STRING$ (3,32):
                GOTO 11
13  IF J = 13
       THEN
          14
       ELSE
    A$ = A$ +  CHR$ (J):
    GOTO 11
14  IF U$ <  > "Y"
       THEN
          POKE 143Ø4, PEEK (17161):
```

```
          PRINT #1,A$
15   PRINT A$:
     IF  LEFT$ (T$,1) = "Y"
        THEN
           LPRINT A$:
           L = L - 1:
           IF L = Ø
              THEN
                 GOSUB 23:
                 GOSUB 22:
                 GOTO 1Ø
              ELSE
           1Ø
        ELSE
     1Ø
16   IF U$ <  > "Y"
        THEN
           PRINT #1, CHR$ (255); CHR$ (255):
           CLOSE
17   POKE &H4ØB1,&HFF:
     POKE &H4ØB2,&HBF:
     IF  LEFT$ (R$,1) = "N"
        THEN
           GOSUB 21
18   RUN "INTRO"
19   CMD "E":
     CLOSE :
     RESUME 7
2Ø   INPUT "Load Destination System Disk and press ENTER";:
     RETURN
21   INPUT "Load PASCAL Disk and press ENTER";:
     RETURN
22   LPRINT " "; CHR$ (13);" "; CHR$ (13);" "; CHR$ (13):
     RETURN
23   GOSUB 22:
     CLS :
     IF  LEFT$ (G$,1) <  > "Y"
        THEN
           RETURN
        ELSE
     INPUT "Press ENTER to continue";:
     L = LL:
     RETURN
```



29

Faster Loading For the Model I

by Mark E. Tyler

System Requirements:
Model I
16K RAM
Editor/assembler

Get tired of waiting for your Model I to load cassette programs? It's possible to exceed the Model I's 500-baud rate. The programs in this chapter show you how.

The TRS-80 uses the double frequency encoding system for data storage. The timing in this system uses a synchronizing pulse at the start of each bit. The synchronizing pulse is followed by a data pulse if the bit to be read is a 1, or no pulse if the bit to be read is a 0. Figure 1 shows the different ways a 1 and a 0 bit are stored on tape.

The timing is important. The software controlling the reading looks for the synchronizing pulse. Once it's found, the software delays a predetermined amount of time, then looks for the data pulse. If it is there, the bit is considered a 1; if not, a 0. If noise on the tape causes the TRS-80 hardware to think there is a pulse where there is not, an error occurs. By controlling the delay times and the reading times, you can speed up the reading and writing of the tape and improve the reliability of data transfer.

The address of the software for reading a byte of data from the tape is 0235H in the Level II ROM. See Figure 2 for a flow diagram. The code at this location saves the BC and HL registers on the stack and sets up a loop to repeat eight times. The loop repeats eight times because data is read serially—one bit at a time—from the tape. The loop calls the code at 0241H where the actual reading takes place. Once the synchronizing pulse is detected, the data pulse will arrive exactly one millisecond later (assuming a 500-bit baud rate). The timing is accomplished automatically, because the software is written so that the time it takes to execute all the read instructions is one millisecond.

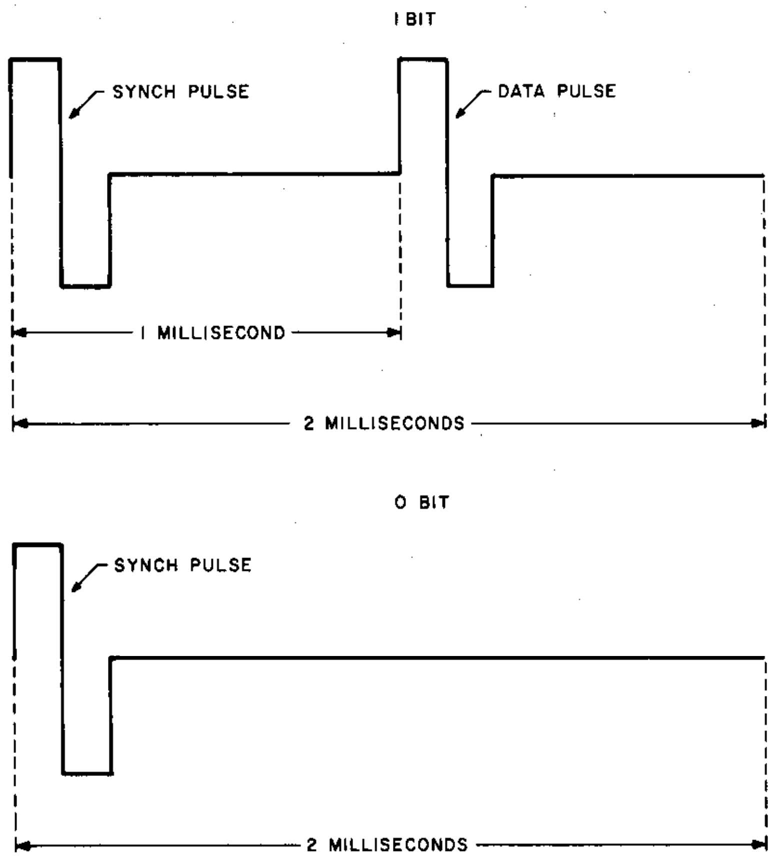


Figure 1. *No data pulse is present if bit is a 0. Timing is for 500 bits/second.*

The Z80 microprocessor used by the TRS-80 requires a definite number of T-states to execute each instruction. A T-state is one clock cycle. In an unmodified TRS-80, a clock cycle is 563 nanoseconds (5.63 × 10E-7 seconds). The instruction PUSH HL, which requires 11 T-states, would take 11 × 563, or 6,193, nanoseconds to execute. The complicated instructions are the conditional branches, which require a different number of T-states depending on whether the condition is true or false. For example, the DJNZ instruction requires 13 T-states if B does not equal zero, but only eight if B does equal zero.

Tape reading timing is automatic because the number of T-states has been accounted for by careful selection of the instructions and delay loops in the software. Take the code at 0241H in the ROM. Here the BC and AF registers are saved on the stack, and port 0FFH, the TRS-80's tape port, is checked for synchronizing pulse. Once the pulse is found, critical timing starts, with a delay loop to use up some T-states. In the later versions of ROM, this loop takes up 1250 T-states. This important loop, DJNZ1, is changed when the tape speed is increased.

Next, the code at 021EH is called. It resets the flip/flop in the TRS-80 tape reading circuit to prepare for the possible 1 pulse from the tape. This code requires 82 T-states. Another delay loop, DJNZ2, is started. It is 1731 T-states long. After this loop, the flip/flop is checked for the presence of a 1-bit pulse. The data, whether it is a 1 or a 0, is stored, and the flip/flop is reset. The program then returns to the code at 0235H. Here the B register is decremented and the whole process repeats until the full eight bits are read.

Remember that DJNZ1 is used to control the length of time between detection of the synchronizing pulse and the resetting of the flip/flop in preparation for the possible data pulse. The flip/flop shouldn't be reset too soon. That causes a long time between resetting and data pulse arrival, and any noise in the interval can set the flip/flop and cause a reading error. DJNZ2's length controls the time between resetting the flip/flop and checking it. Some time must be allowed for the electronics to stabilize, but too much time can cause an error. The two delay loops together

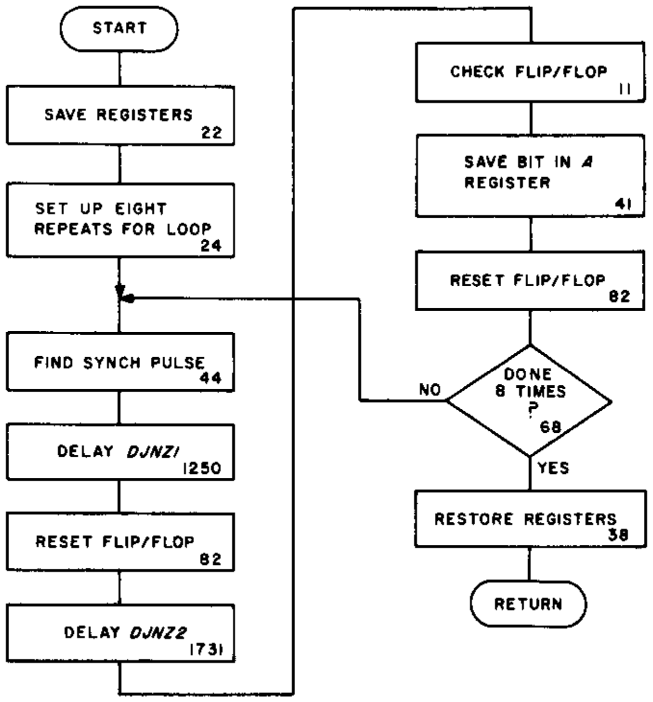


Figure 2. *Tape read subroutine flow chart. Number in bottom right hand corner is number of T-states required to execute the indicated section.*

control how long the software waits after the synchronizing pulse before it looks for a data pulse. Changing these two delay loops is the key to increasing the tape reading speed of the TRS-80.

In order to read a tape faster, it must be written faster. The code for writing a byte to tape is at memory address 0264H. The byte to be written should be in the A register. T-states are important in this code because timing is critical. See Figures 3A and 3B for a flow diagram of the software. First, the registers used by the code are saved. This includes the HL, BC, DE and AF registers. Next, a loop is set up to be executed eight times, and the byte to be written is transferred to the D register.

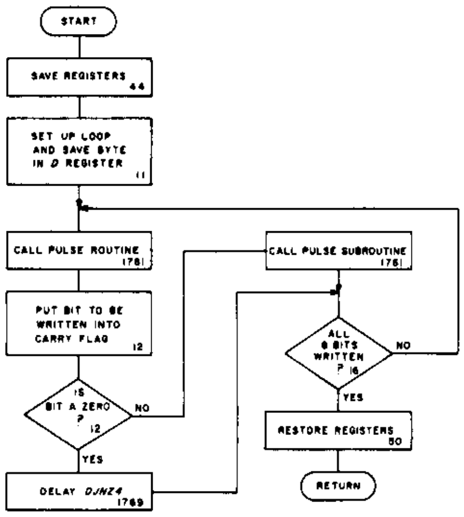


Figure 3A. *Tape write subroutine flow chart. Numbers indicate number of T-states required to execute the indicated section.*

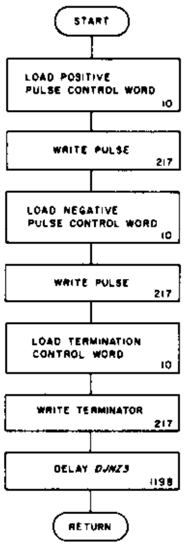

START
LOAD POSITIVE PULSE CONTROL WORD 10
WRITE PULSE 217
LOAD NEGATIVE PULSE CONTROL WORD 10
WRITE PULSE 217
LOAD TERMINATION CONTROL WORD 10
WRITE TERMINATOR 217
DELAY DJNZ3 1198
RETURN

Figure 3B. *Pulse subroutine called from write subroutine*

The code at 01D9H is called. This code writes the synchronizing pulse on the tape. The pulse is really two pulses, one after the other. The first pulse is positive and the second is negative. Each pulse lasts 227 T-states.

The A register is loaded with the byte to be written, from the D register. The A register is then rotated left so the bit to be written is put into the carry flag. The shifted byte is then loaded back into the D register for safekeeping. The bit in the carry flag can be easily tested to see if it is a 1 or a 0. If the carry flag is set (bit is a 1), the code continues by calling 01D9H. If the carry flag is reset (bit is a 0), the code branches to 027EH.

The code at 027EH is a delay loop. It is necessary because if the bit is a 0, no data pulse is written and you have to kill some time. This is done with another DJNZ loop, DJNZ4, which is *1769 T-states long. On completion,* control transfers to 0276H. This is the same location the program flow returns to after writing a data pulse, if the carry flag was set. The one data pulse is written with a call to 01D9H, the same code that wrote the synchronizing pulse.

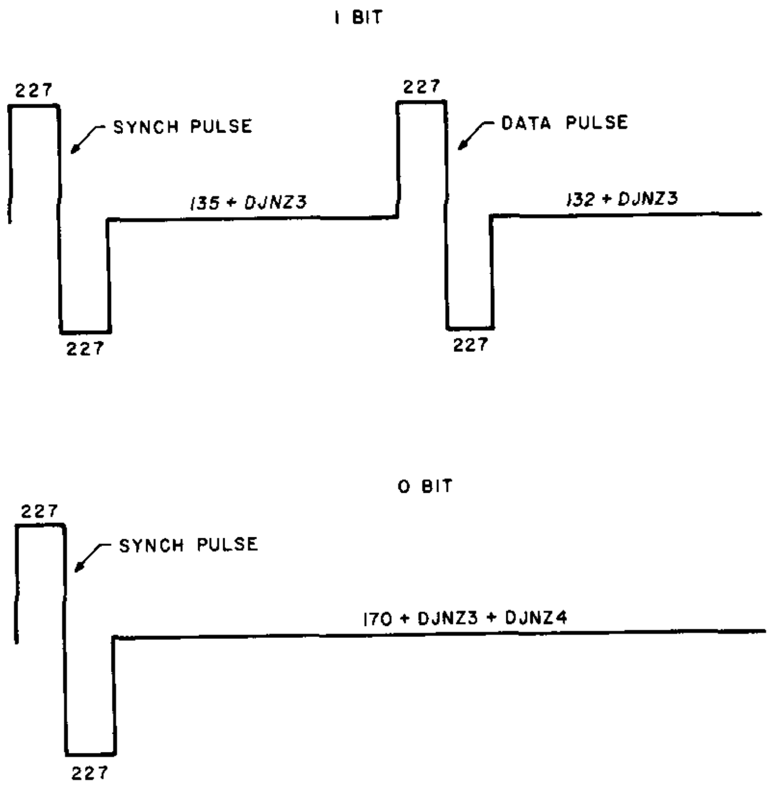


Figure 4. *Software timing measured in T-states*

At 01D9H, a control word is loaded into the HL register and the code at 0221H is called (Figure 3B). This outputs a positive pulse to the tape recorder. The length of the pulse is controlled by a DJNZ loop. Another control word is loaded into the HL register, and the routine at 0221H is called again, but this time a negative pulse is written to the tape. Its length is also controlled by a DNJZ loop. To terminate the negative pulse another control word is loaded into the HL register, and the routine at 0221H is called once more. Finally, another DJNZ loop, DJNZ3, is executed. It is important in controlling the writing time, and is 1198 T-states long. After executing this loop, control transfers back to 0276H.

The C register is decremented and checked to see if all eight bits have been written. If not, control goes back to 026BH, where the synchronization pulse is again written. If all eight bits have been written, the registers pop off the stack, and control returns to the calling routine. Figure 4 is Figure 1 redrawn to show the length of time, measured in T-states, for each part of the stored signal. DJNZ3 alone controls the timing of the 1 bit. You can shorten the delay of DJNZ3 and increase the number of 1 bits you can write per second. The 0 bit is controlled by both DJNZ3 and DJNZ4. It is a simple matter to shorten the length of the 0 bit so you can write faster. Just make sure the 0 bit is about the same length as the 1 bit. They need not have exactly the same length because there is some excess time in the reading code, and the reading timing always starts with the detection of the synchronization pulse.

Figure 5 shows how DJNZ1 and DJNZ2 control the reading time: the positive pulse represents the reading of the TRS-80 tape flip/flop, and the negative pulses represent the resetting of the flip/flop. The figure starts with the detection of the synchronization pulse. The numbers represent the length of time, measured in T-states. By adjusting DJNZ1 and DJNZ2, you can control when the flip/flop is reset and how far into the data pulse you check for the presence of the pulse.

By controlling these four delay loops, you control the tape writing and

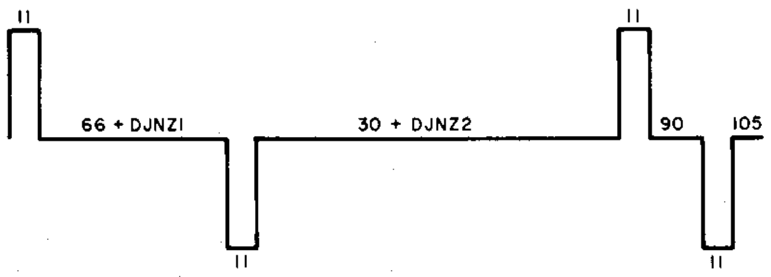


Figure 5. *Positive pulse represents checking status of flip/flop. Negative pulse represents resetting the flip/flop.*

reading speeds. I have done experiments at various speeds of writing and reading that range from the 500 bits/second standard to over 1500 bits/second. Figure 6 is a table of the various delay times, in T-states, for the four DJNZ loops in the reading and writing routines. The control bytes to be loaded into the B register to obtain the necessary delays are in parentheses.

Bits/Sec	Total Pulse Length	DJNZ1	DJNZ2	DJNZ3	DJNZ4
500	3554	1243(60H)	1724(85H)	1191(5CH)	1750(87H)
600	2959	1035(50H)	1412(6DH)	892(45H)	1438(6FH)
700	2540	879(44H)	1204(5DH)	684(35H)	1230(5FH)
800	2228	775(3CH)	035(50H)	528(29H)	1074(53H)
900	1968	671(34H)	918(47H)	398(1FH)	944(49H)
1000	1760	554(2BH)	840(41H)	294(17H)	840(41H)
1100	1604	476(25H)	736(39H)	216(11H)	762(3BH)
1200	1474	424(21H)	684(35H)	151(0CH)	697(36H)
1300	1370	372(1DH)	645(32H)	99(08H)	645(32H)
1400	1266	320(19H)	593(2EH)	47(04H)	593(2EH)
1500	1188	294(17H)	489(26H)	8(01H)	554(2BH)

Figure 6. *Delay time constants for read and write subroutines. Numbers in parentheses are control bytes needed for the delay loops.*

Keep in mind when using modified tape speeds that the speed is for writing the bits of a byte. If you use a 1500 bit/second reading or writing subroutine, it does not control the length of time between bytes. The calling program does that. That is how you can still do all the things between bytes that were done before. You are not really writing everything at the 1500 bits/second rate; if you could see the bits on the tape, you would see groups of eight bits spaced at 0.666 millisecond intervals and the 8-bit groups spaced at some other interval, dependent on the program calling the faster writing subroutine.

If these high-speed routines are to be of any practical value, you must be able to use them to read and write both BASIC and system tapes. You should be able to read in existing tapes at the normal speed and write them again at the higher speed. You can do this with machine-language programs that patch themselves into BASIC ROM, giving you two new commands and a third, modified command. The two new commands are LOAD and SAVE. They operate like CLOAD and CSAVE, but at a higher speed. The third command is a modified SYSTEM command which allows system tapes to be read at normal or faster speed. Another program (Program Listing 4) shows how to modify John Harrell's ZBUG monitor (*80 Microcomputing*, January 1981, p. 130) to read and write tapes at both normal and fast speed.

The first three programs add new commands to ROM. The LOAD and

SAVE commands are already reserved words in the ROM, but are used only on disk systems. If you try to execute the SAVE command without Disk BASIC, you get the L3 ERROR message. This is because Level II BASIC branches to RAM memory location 41A0H after a SAVE command. A jump to the error message routine is placed there during the initialization of this area of RAM from the MEM SIZE message. When Disk BASIC is loaded and initialized, it places new jump instructions at this and other RAM locations that patch into Disk BASIC. You can do the same thing with this program by placing jump instructions to the locations of our new code.

When the ROM encounters the SYSTEM command, a call is made to 41E2H in RAM. Normally there is a RETurn instruction there so program control goes back into ROM. By placing a jump instruction here, you can branch to the new SYSTEM code very easily. The initialization portion of each program sets up all the jumps needed.

Each program is similar to the corresponding program in ROM written by Microsoft, with the new read and write codes patched in. The similarity means the programs operate exactly like the corresponding BASIC command. The start of the code is the initialization portion of the program. Jump instructions are placed into the communications section of RAM at the correct locations for LOAD, SAVE, and SYSTEM.

The first program (see Program Listing 1) is SAVE, which is the high speed equivalent of CSAVE. The first thing that SAVE does is turn on the recorder, by calling a ROM subroutine at 01FEH, which selects the correct recorder and turns it on. Next, it writes 256 zeros to tape followed by the 0A5H marker. A call is made to 2337H in the ROM. This code looks for a file name. The address of the string containing the file name is in the HL register on returning from the subroutine. A call to 2A13H gets the address into the DE register. Next, the three 0D3H leader bytes are written on the tape, followed by the file name whose address is in the DE register.

The DE register is set up to point to the start of the BASIC program. The HL register contains the address of the end of the program. The A register is loaded with the contents of the address in the DE register. The A register is written to tape using the fast writing program. The DE register is incremented and compared with the HL register. This procedure repeats until the end of the BASIC program (HL = DE). The recorder is turned off with a call to 01F8H, the HL and BC registers are restored, and control returns to ROM.

The rest of the SAVE program involves the speeded-up write subroutine. The program defines a word called WRSP, which contains the DJNZ3 and DJNZ4 delay loop variables to be loaded into the B register in order to obtain the desired tape speed. The WRSP variable in Program Listing 1 is for writing at 1000 bits/second. Use Figure 6 to select a speed. DJNZ4 is the most significant byte and DJNZ3 is the least significant byte of

WRSP. To use WRSP, load HL with the address of WRSP, then load B with the contents of the address pointed to by the HL register.

The next program (Program Listing 2) is called LOAD. It loads a fast tape written by SAVE. First, the proper RDSP variable is created. Next, a check is made to see if a question mark (?) follows the LOAD command. If so, the load is being verified as in CLOAD? If it is a LOAD? command, a flag is set and saved. A LOAD command resets the flag before saving it. The file name is searched for, and when found, combined with the LOAD/LOAD? flag and stored in RAM. If this is a LOAD command, the NEW subroutine at 1B4DH is called. The tape recorder is turned on and the tape is read, looking for the synchronizing zeros in the leader. Once these are found, the LOAD/LOAD? flag and the file name to the DE register are restored. The three 0D3H bytes are read from the tape. The file name is read from the tape and compared with the one stored in the E register. If it is not right, a jump is made to LD4 to wait for the right file name to be read.

After the right file name has been found, the tape is read again, looking for the next byte. If the LOAD/LOAD? flag in the D register is set, the byte is compared with the data in memory. If an error occurs, a jump is made to LD2, where the BAD message is displayed, before the return to BASIC at 1A18H. If the byte was the same as in the program or if the LOAD/LOAD? flag is reset, the bit is saved and the free memory pointer is updated with a call to 196CH.

A test is made to see if the byte is a zero. If so, the star in the right hand corner of the screen blinks, and the process starts over, until three zeros are found. When that is accomplished, the READY message appears, the recorder is turned off, and the starting address is saved on the stack before returning to BASIC at 1AE8H. The LOAD routine uses the READ and SYNCH routines following the SYSTEM program. A storage area called RDSP is defined, with the DJNZ1 and DJNZ2 delays already loaded into them. Then the B register is loaded with these delays from the RDSP word. DJNZ2 is the most significant byte of RDSP and DJNZ1 is the least significant byte.

The third program (Program Listing 3) is the SYSTEM code. It functions like the standard SYSTEM command, except that whenever a file name is specified, a short message which says (FAST/SLOW) is displayed. Responding F to this message loads the RDSP word with the faster reading speed variable. Any other response loads the RDSP word with the standard 500 bit/second speed variable. The tape then reads at the selected speed with the same results as the regular SYSTEM command.

The SYSTEM code starts by popping the return address of the calling program off the stack. This keeps the stack in order. Next a jump is made to SYSTM2, to set up a new stack pointer and skip down one line on the video with a call to 20FEH. The system prompt (*?) displays, and execution waits on input from the keyboard. This input is evaluated and a branch is made if the first character is a slash (/). If it is, a jump is made back into the ROM SYSTEM routine at 031DH. If there was no slash, the input is assumed to be a file name. The SPEED subroutine is called where the (FAST/SLOW) message is displayed, and the input is evaluated so the proper DJNZ delays are loaded into RDSP.

The tape recorder is turned on and the synchronizing byte is looked for. The 55H byte is found next, followed by a search for the correct file name. Once it's found, the star in the top right-hand side of the screen is turned on or off. Another byte is then read and compared to 78H. This is the end of the tape flag and it branches to SYSTM1, where the transfer address is stored in 40DFH, and the recorder is turned off. The stack pointer is readjusted, the system prompt (*?) displays, and the program waits for input.

If the 78H flag is not found, the tape is read until a 3CH is detected. This is followed by the number of bytes to read and the address where the bytes are to be stored. A checksum is set up, and the bytes are read from the tape and stored in the appropriate place. Once all the bytes are ready, the checksum is compared with the one read from tape. If they are different, a C displays in the upper right-hand corner of the screen, and the tape is scanned for the start of the next program. If the checksum is right, the program branches to SYSTM5, to look for the 78H byte and continue from there. The SYSTEM program sequence ends by turning off the tape recorder and displaying the system prompt (*?).

These three programs provide the commands to read and write fast BASIC tapes and to read fast system tapes, but you need a method of writing fast system tapes. The easiest way is to modify a monitor program, as in Program Listing 2. I chose John Harrell's ZBUG program (*80 Microcomputing,* January 1981) for its very good tape copying and verification commands. Modification requires finding all the locations in the monitor where a write-from-tape or read-to-tape function occurs and substituting calls to the new routines. You can use the ZBUG monitor to find these locations. The find address (A) command in ZBUG is loaded with one of the subroutine addresses to be searched for, followed by the area in memory where ZBUG resides. The addresses needed are 0235H(READ), 0264H(WRTAPE), 0287H(WSYNCH), 0214H(RDBIT) and 0296H(SYNCH). Once found, the address portion of the call instructions is changed to point to the appropriate new subroutine patched in at the end of ZBUG. Some editor/assembler programs make the address changing easier because they have a macro command. The EDTASM+ program allows the use of macros. In Program Listing 2, the macro NEW is defined with variables #ADD and #NAME. The macro is simply ORG #ADD, followed by CALL #NAME, and is used 43 times to set up all the necessary patches.

In Program Listing 4, TAPE is the subroutine that prints an F/S message after the selected ZBUG command. It accepts a one letter response. An F response loads the WRSP and RDSP words with the high-speed DJNZ variables. Any other response loads the words with the regular speed variables. The COPY and WRITE subroutines in Program Listing 4 are necessary because these ZBUG commands need slightly different processing than the others.

These programs can also be assembled on an editor/assembler like EDTASM+. Program Listings 1–3 reside in upper memory and are shown at 7E00H for a 16K machine. This is a multiple-origin set of programs. Only the origin indicated in the comments on each program should be changed when relocating the program. Once the programs have been entered and debugged, save them to tape. They can then be loaded under the SYSTEM command. Always load Program Listing 1 last, because there the changes to the RAM addresses for SYSTEM, LOAD and SAVE are made. If Program Listing 1 is loaded before Program Listing 3, the SYSTEM command jumps to the new address, but there is no program there. No initialization of the programs is necessary, so control returns to BASIC with a slash (/) ENTER after the last part is loaded under the SYSTEM command. You have the three new commands in your Level II BASIC until you return to the memory size question, when you must enter the programs again. Be sure to protect sufficient memory (32255) when answering the memory size question.

Program Listing 4 can be entered by using an editor/assembler that supports macros, such as Microsoft's Editor/Assembler+. Once ZBUG (or a similar monitor) has been modified, it can make copies of itself at slow or fast speed. You can use it to read any SYSTEM tape at 500 baud, and write it to tape at a higher speed using the COPY (,) command. Using the 1000 bit/second loading speed can decrease your loading time by more than half.

Cautions

Higher speed tapes are more sensitive to the volume control setting. Usually it must be turned down when you load them. Once the proper setting is found, the tapes load very reliably; however, reliability decreases as speed increases. I believe that the volume control problem could be easily overcome with the addition of a pulse-shaping device such as The Data Dubber from the Peripheral People, or a similar device.

The slow DJNZ delays used in both listings are from the new Level II ROM with its improved tape loading features. Some commercial tapes will not load using these delays. This has been true for system tapes more than BASIC tapes. The easiest way to fix the problem is to use the old delays from the original ROM. This would mean changing DJNZ1 to

41H, and DJNZ2 to 76H. After you have made a copy of the troublesome tape, it can be read more reliably using the new delays already in the programs.

Program Listing 1. *SAVE*

```
                 ;===================================
                 ;         LISTING I
                 ;       FSTAPE  VER 1.2
                 ;        BY MARK TYLER
                 ;===================================



41A0                             ORG     41A0H    ;SET UP
41A0 C3007E                      JP      SAVE     ;'SAVE' VERB
4188                             ORG     4188H    ;SET UP
4188 C37E7E                      JP      LOAD     ;'LOAD' VERB
41E2                             ORG     41E2H    ;SET UP
41E2 C3117F                      JP      SYSTEM   ;'SYSTEM' MODIFICATION





7E00             MEMORY  DEFL    7E00H    ;CHANGE THIS ADDRESS ONLY.
                                          ;TO RELOCATE
0000             DIS     DEFL    MEMORY-7E00H
7E00                     ORG     MEMORY


                 ;--------------------------------
                 ;              SAVE
                 ;       1000 BIT/SEC VERSION
                 ;--------------------------------



7E00 CD2D7E      SAVE    CALL    WSYNCH   ;TURN ON CASSETTE
                                          ;AND WRITE HEADER
7E03 CD3723              CALL    2337H    ;EVALUATE REST OF
                                          ;'SAVE' COMMAND
7E06 C5                  PUSH    BC       ;SAVE REGISTER
7E07 E5                  PUSH    HL       ;SAVE ADD OF FILE NAME
7E08 CD132A              CALL    2A13H    ;PUT ADD IN 'DE' REGISTER
7E0B 3ED3                LD      A,0D3H   ;WRITE THE THREE
7E0D 0603                LD      B,03     ;LEADER BYTES
7E0F CD3A7E              CALL    WRITE    ;ON TAPE
7E12 10FB                DJNZ    $-3
7E14 1A                  LD      A,(DE)   ;FILE NAME TO 'A' REGISTER
7E15 CD3A7E              CALL    WRITE    ;WRITE IT ON TAPE
7E18 2AA440              LD      HL,(40A4H)  ;POINTER TO BEGINNING
7E1B EB                  EX      DE,HL    ;OF BASIC PROGRAM TO 'DE'
7E1C 2AF940              LD      HL,(40F9H)  ;END OF BASIC PROGRAM
                                          ;POINTER TO 'HL' REGISTER
7E1F 1A                  LD      A,(DE)   ;GET NEXT BYTE
7E20 13                  INC     DE       ;STEP POINTER
```

Program continued

```
7E21 CD3A7E     ØØ52Ø          CALL    WRITE     ;WRITE BYTE TO TAPE
7E24 DF         ØØ53Ø          RST     18H       ;COMPARE 'HL' AND 'DE'
7E25 2ØF8       ØØ54Ø          JR      NZ,$-6    ;AGAIN IF 'DE'<>'HL'
7E27 CDF8Ø1     ØØ55Ø          CALL    Ø1F8H     ;TURN OFF CASSETTE
7E2A E1         ØØ56Ø          POP     HL        ;RESTORE REGISTERS
7E2B C1         ØØ57Ø          POP     BC
7E2C C9         ØØ58Ø          RET               ;GO BACK TO BASIC
                ØØ59Ø
7E2D CDFEØ1     ØØ6ØØ WSYNCH   CALL    Ø1FEH     ;TURN ON CASSETTE
7E3Ø 06FF       ØØ61Ø          LD      B,ØFFH    ;WRITE 255
7E32 AF         ØØ62Ø          XOR     A         ;ZEROES
7E33 CD3A7E     ØØ63Ø          CALL    WRITE     ;TO TAPE
7E36 1ØFB       ØØ64Ø          DJNZ    $-3       ;DONE YET ?
7E38 3EA5       ØØ65Ø          LD      A,ØA5H    ;WRITE SYNCH BYTE FIRST
                ØØ66Ø
7E3A E5         ØØ67Ø WRITE    PUSH    HL        ;SAVE NEEDED REGISTERS
7E3B C5         ØØ68Ø          PUSH    BC
7E3C D5         ØØ69Ø          PUSH    DE
7E3D F5         ØØ7ØØ          PUSH    AF
7E3E ØEØ8       ØØ71Ø          LD      C,8       ;COUNTER FOR 8 BITS
7E4Ø 57         ØØ72Ø          LD      D,A       ;SAVE BYTE TO BE WRITTEN
7E41 CD5D7E     ØØ73Ø WR2      CALL    WR5       ;WRITE SYNCH PULSE TO TAPE
7E44 7A         ØØ74Ø          LD      A,D       ;RETURN BYTE TO 'A' REG.
7E45 CBØ7       ØØ75Ø          RLC     A         ;ROTATE BIT TO BE WRITTEN
                ØØ76Ø                            ;TO CARRY FLAG
7E47 57         ØØ77Ø          LD      D,A       ;SAVE IN 'D' AGAIN
7E48 3ØØB       ØØ78Ø          JR      NC,WR4    ;GO IF BIT IS ZERO
7E4A CD5D7E     ØØ79Ø          CALL    WR5       ;WRITE A ONE BIT
7E4D ØD         ØØ8ØØ WR3      DEC     C         ;ADJUST THE COUNTER
7E4E 2ØF1       ØØ81Ø          JR      NZ,WR2    ;AGAIN IF 'C'<> Ø
7E5Ø F1         ØØ82Ø          POP     AF        ;RESTORE REGISTERS
7E51 D1         ØØ83Ø          POP     DE
7E52 C1         ØØ84Ø          POP     BC
7E53 E1         ØØ85Ø          POP     HL
7E54 C9         ØØ86Ø          RET               ;RETURN TO CALLING PROGRAM
                ØØ87Ø
7E55 21FD7F     ØØ88Ø WR4      LD      HL,WRSP+1
7E58 46         ØØ89Ø          LD      B,(HL)    ;GET DJNZ4
7E59 1ØFE       ØØ9ØØ          DJNZ    $         ;DELAY AWHILE
7E5B 18FØ       ØØ91Ø          JR      WR3       ;GO BACK TO MAIN PROGRAM
7E5D 21Ø1FC     ØØ92Ø WR5      LD      HL,ØFCØ1H ;POSITIVE CONTROL WORD
7E6Ø CD21Ø2     ØØ93Ø          CALL    Ø221H     ;WRITE POSITIVE PULSE
7E63 Ø6ØB       ØØ94Ø          LD      B,ØBH     ;DELAY
7E65 1ØFE       ØØ95Ø          DJNZ    $         ;AWHILE
7E67 21Ø2FC     ØØ96Ø          LD      HL,ØFCØ2H ;NEGATIVE CONTROL WORD
7E6A CD21Ø2     ØØ97Ø          CALL    Ø221H     ;WRITE NEGATIVE PULSE
7E6D Ø6ØB       ØØ98Ø          LD      B,ØBH     ;DELAY
7E6F 1ØFE       ØØ99Ø          DJNZ    $         ;AWHILE
7E71 21ØØFC     Ø1ØØØ          LD      HL,ØFCØØH ;TERMINATOR CONTROL WORD
7E74 CD11Ø2     Ø1Ø1Ø          CALL    Ø211H     ;WRITE TERMINATOR
7E77 21FC7F     Ø1Ø2Ø          LD      HL,WRSP   ;GET DJNZ3
7E7A 46         Ø1Ø3Ø          LD      B,(HL)
7E7B 1ØFE       Ø1Ø4Ø          DJNZ    $         ;DELAY AWHILE
7E7D C9         Ø1Ø5Ø          RET               ;RETURN TO CALLING PROGRAM
                Ø1Ø6Ø
                Ø1Ø7Ø
                Ø1Ø8Ø ;   FSTAPE PROGRAM DEFINITIONS
7E7E            Ø1Ø9Ø LOAD     EQU     7E7EH+DIS
7F11            Ø11ØØ SYSTEM   EQU     7F11H+DIS
7FFC            Ø111Ø WRSP     EQU     7FFCH+DIS
                Ø112Ø
1A19            Ø113Ø          END     1A19H
```

Program Listing 2. *LOAD*

```
                   ;*********  FSTAPE  PART 2  VER 1.2     *********
                   ;*********       BY MARK TYLER          *********


7EØØ               MEMORY  DEFL    7EØØH    ;CHANGE ONLY THIS ADDRESS
                                            ;TO RELOCATE


ØØØØ               DIS     DEFL    MEMORY-7EØØH
7E7E                       ORG     7E7EH+DIS

                   ;-------------------------------
                   ;               LOAD
                   ;      1ØØØ BIT/SEC VERSION
                   ;-------------------------------


7E7E E5            LOAD    PUSH HL          ;SAVE REGISTER
7E7F 212B41                LD      HL,412BH   ;DJNZ1 AND DJNZ2
7E82 22FE7F                LD      (RDSP),HL  ;INTO RDSP
7E85 E1                    POP     HL       ;RESTORE REGISTER
7E86 D6B2                  SUB     ØB2H     ;TEST FOR LOAD?
7E88 28Ø2                  JR      Z,$+4    ;JUMP IF LOAD?
7E8A AF                    XOR     A        ;SET UP LOAD FLAG
7E8B Ø12F23                LD      BC,232FH  ;USE 2ND AND 3RD BYTES
                                            ;TO CPL AND INC 'HL'
7E8E F5                    PUSH    AF       ;SAVE FLAG
7E8F 7E                    LD      A,(HL)   ;GET FILE NAME IF THERE
7E9Ø B7                    OR      A        ;ZERO 'A' IF NO FILE NAME
7E91 28Ø7                  JR      Z,$+9    ;JUMP IF NO FILE NAME
7E93 CD3723                CALL    2337H    ;EVALUATE FILE NAME
7E96 CD132A                CALL    2A13H    ;FILE NAME ADD INTO 'DE'
7E99 1A                    LD      A,(DE)   ;PUT FILE NAME
7E9A 6F                    LD      L,A      ;INTO 'L' REGISTER
7E9B F1                    POP     AF       ;GET BACK LOAD/LOAD? FLAG
7E9C B7                    OR      A        ;SET A ACCORDING TO FLAG
7E9D 67                    LD      H,A      ;SAVE IN 'H' REGISTER
7E9E 222141                LD      (4121H),HL  ;SAVE IN RAM
7EA1 CC4D1B                CALL    Z,1B4DH ;CALL 'NEW' IF FLAG RESET
7EA4 21ØØØØ                LD      HL,ØØØØH   ;CASSETTE CONTROL WORD
7EA7 CD8Ø7F                CALL    SYNCH    ;TURN ON CASSETT AND
                                            ;FIND ØA5H BYTE
7EAA 2A2141                LD      HL,(4121H) ;RESTORE LD/LD? FLAG
7EAD EB                    EX      DE,HL    ;PUT IN 'DE' REGISTER
7EAE Ø6Ø3          LD3     LD      B,Ø3H    ;SET UP LOOP
7EBØ CD967F                CALL    READ     ;TO LOOK FOR THREE
7EB3 D6D3                  SUB     ØD3H     ;ØD3H BYTES
7EB5 2ØF7                  JR      NZ,LD3   ;JUMP IF NOT ØD3H
7EB7 1ØF7                  DJNZ    $-7      ;DO UNTIL FOUND
7EB9 CD967F                CALL    READ     ;READ FILE NAME
7EBC 1C                    INC     E        ;USER SPECIFIED NAME?
7EBD 1D                    DEC     E        ;SET FLAG IF NAME EXIST
7EBE 28Ø3                  JR      Z,$+5    ;JUMP IF NO FILE NAME
7ECØ BB                    CP      E        ;COMPARE TO CALLER'S NAME
7EC1 2Ø37                  JR      NZ,LD4   ;WAIT UNTIL NAME IS FOUND
7EC3 2AA44Ø                LD      HL,(4ØA4H)  ;BASIC PROGRAM POINTER
                                            ; TO 'HL' REGISTER
7EC6 Ø6Ø3                  LD      B,Ø3H    ;NEED THREE ZEROES FOR
                                            ;END OF FILE
```

Program continued

```
7EC8 CD967F     LD1      CALL   READ      ;READ A BYTE
7ECB 5F                  LD     E,A       ;SAVE IN 'E' REGISTER
7ECC 96                  SUB    (HL)      ;COMPARE WITH PROGRAM
7ECD A2                  AND    D         ; AND WITH LOAD/LOAD? FLAG
7ECE 2021                JR     NZ,LD2    ;PRINT ERROR MESSAGE
                                          ;IF MISMATCH
7EDØ 73                  LD     (HL),E    ;OTHERWISE SAVE BYTE
7ED1 CD6C19              CALL   196CH     ;ADJUST FREE MEMORY PT.
7ED4 7E                  LD     A,(HL)    ;TEST BIT
7ED5 B7                  OR     A         ;FOR A ZERO
7ED6 23                  INC    HL        ;ADJUST 'HL'
7ED7 20ED                JR     NZ,$-17   ;LOOP IF NOT ZERO
7ED9 CD2CØ2              CALL   Ø22CH     ;BLINK '*'
7EDC 1ØEA                DJNZ   LD1       ;NEED THREE ZEROES
7EDE 22F94Ø              LD     (4ØF9H),HL  ;SAVE END OF PROGRAM
                                          ;ADDRESS IN RAM
7EE1 212919              LD     HL,1929H     ;'READY' MESSAGE ADD
7EE4 CDA728              CALL   28A7H     ;DISPLAY MESSAGE
7EE7 CDF8Ø1              CALL   Ø1F8H     ;TURN OFF CASSETTE
7EEA 2AA44Ø              LD     HL,(4ØA4H)  ;STARTING ADD OF
7EED E5                  PUSH   HL        ;PROGRAM TO STACK
7EEE C3E81A              JP     1AE8H     ;RETURN TO BASIC

7EF1 21ØC7F     LD2      LD     HL,MSG1 ;'HL'=ERROR MESSAGE ADD
7EF4 CDA728              CALL   28A7H     ;DISPLAY MESSAGE
7EF7 C3181A              JP     1A18H     ;RETURN TO BASIC
7EFA 323E3C     LD4      LD     (3C3EH),A  ;SAVE FILE NAME
7EFD Ø6Ø3       LD5      LD     B,Ø3H     ;SEARCH FOR
7EFF CD967F     LD6      CALL   READ      ;THREE ZEROES
7FØ2 B7                  OR     A         ;DO UNTIL
7FØ3 2ØF8                JR     NZ,LD5    ;YOU FIND
7FØ5 1ØF8                DJNZ   LD6       ;THREE IN A ROW
7FØ7 CD837F              CALL   SYNCH+3 ;START READING TAPE AGAIN
7FØA 18A2                JR     LD3       ;CONTINUE

7FØC 42         MSG1     DEFM   'BAD'     ;ERROR MESSAGE
7FØF ØD                  DEFB   ØDH
7F1Ø ØØ                  DEFB   ØØH


                ;    FSTAPE PROGRAM DEFINITIONS
7FFE            RDSP     EQU    7FFEH+DIS
7F8Ø            SYNCH    EQU    7F8ØH+DIS
7F96            READ     EQU    7F96H+DIS


1A19                     END    1A19H
```

Program Listing 3. *SYSTEM*

```
                ;*********  FSTAPE    PART 3   VER 1.2   *********
                ;*********           BY MARK TYLER        *********

7EØØ            MEMORY   DEFL   7EØØH     ;CHANGE THIS ADDRESS ONLY
                                          ;TO RELOCATE

ØØØØ            DIS      DEFL   MEMORY-7EØØH
7F11                     ORG    7F11H+DIS
```

```

               ;-------------------------------
               ;            SYSTEM
               ;      1ØØØ BIT/SEC VERSION
               ;-------------------------------


7F11 E1        SYSTEM  POP   HL         ;KEEP STACK IN ORDER
7F12 18Ø9              JR    SYSTM2     ;CONTINUE
7F14 CDE77F    SYSTM1  CALL  RDHL       ;GET TRANSFER ADD
7F17 22DF4Ø            LD    (4ØDFH),HL  ;SAVE IN RAM
7F1A CDF8Ø1            CALL  Ø1F8H      ;TURN OFF TAPE
7F1D 318842    SYSTM2  LD    SP,4288H   ;SET UP NEW STACK
7F2Ø CDFE2Ø            CALL  2ØFEH      ;SKIP DOWN ONE LINE
7F23 3E2A              LD    A,2AH      ;'*' TO A REGISTER
7F25 CD2AØ3            CALL  Ø32AH      ;DISPLAY STAR
7F28 CDB31B            CALL  1BB3H      ;WAIT ON INPUT
7F2B DACCØ6            JP    C,Ø6CCH    ;JUMP IF BREAK KEY
7F2E D7                RST   1ØH        ;LOOK FOR '/'
7F2F CA9719            JP    Z,1997H    ;JUMP TO SN ERROR
7F32 FE2F              CP    2FH        ;COMPARE TO '/'
7F34 CA1DØ3            JP    Z,Ø31DH ;JUMP BACK TO RAM IF '/'
7F37 CDC67F            CALL  SPEED      ;GET SPEED WANTED
7F3A CDØØ7F            CALL  SYNCH      ;START CASSETTE AND
                                        ;FIND ØA5H BYTE
7F3D CD967F    SYSTM3  CALL  READ       ;READ NEXT BYTE
7F4Ø FE55              CP    55H        ;LOOK FOR 55H BYTE
7F42 2ØF9              JR    NZ,SYSTM3  ;DO UNTIL YOU FIND IT
7F44 Ø6Ø6              LD    B,Ø6H      ;LOOK FOR
7F46 7E        SYSTM4  LD    A,(HL)     ;6 LETTER
7F47 B7                OR    A          ;FILE NAME
7F48 28Ø9              JR    Z,SYSTM5   ;POINTED TO
7F4A CD967F            CALL  READ       ;BY 'HL' REGISTER
7F4D BE                CP    (HL)       ;AND COMPARE
7F4E 2ØED              JR    NZ,SYSTM3  ;JUMP IF NOT THE SAME
7F5Ø 23                INC   HL         ;DO UNTIL
7F51 1ØF3              DJNZ  SYSTM4     ;MATCH IS MADE
7F53 CD2CØ2    SYSTM5  CALL  Ø22CH      ;BLINK '*'
7F56 CD967F    SYSTM6  CALL  READ       ;READ NEXT BYTE
7F59 FE78              CP    78H        ;COMPARE WITH 78H
7F5B 28B7              JR    Z,SYSTM1;JUMP IF MATCH
7F5D FE3C              CP    3CH        ;COMPARE WITH 3CH
7F5F 2ØF5              JR    NZ,SYSTM6  ;DO UNTIL 3CH OR
                                        ;78H FOUND
7F61 CD967F            CALL  READ       ;GET # OF BYTES TO LOAD
7F64 47                LD    B,A        ;SAVE IN 'B' REGISTER
7F65 CDE77F            CALL  RDHL       ;READ NEXT 2 BYTES AND
                                        ;PUT INTO 'HL' REGISTER
7F68 85                ADD   A,L        ;START CHECK SUM
7F69 4F                LD    C,A

7F6A CD967F    SYSTM7  CALL  READ       ;READ NEXT BYTE
7F6D 77                LD    (HL),A     ;STORE IT
7F6E 23                INC   HL         ;ADJUST STORAGE ADD
7F6F 81                ADD   A,C        ;USE FOR CHECK SUM
7F7Ø 4F                LD    C,A        ;SAVE NEW CHECK SUM
7F71 1ØF7              DJNZ  SYSTM7     ;ARE ALL BYTES READ?
7F73 CD967F            CALL  READ       ;READ CHECK SUM
7F76 B9                CP    C          ;COMPARE WITH OUR'S
7F77 28DA              JR    Z,SYSTM5   ;GO IF MATCH
7F79 3E43              LD    A,43H      ;LOAD A WITH 'C'
```

Program continued

```
7F7B 32ØØ3D          LD      (3DØØH),A  ;AND DISPLAY
7F7E 18D6            JR      SYSTM6  ;CONTINUE

7F8Ø CDFEØ1   SYNCH  CALL    1FEH    ;TURN ON CASSETTE
7F83 E5              PUSH    HL      ;SAVE 'HL'
7F84 AF              XOR     A       ;LOAD A WITH 'Ø'
7F85 CDA27F          CALL    RDBIT   ;READ FROM TAPE
7F88 FEA5            CP      ØA5H    ;IS IT ØA5H BYTE?
7F8A 2ØF9            JR      NZ,$-5  ;NO -- KEEP TRYING
7F8C 3E2A            LD      A,2AH   ;'*' TO A REGISTER
7F8E 323E3C          LD      (3C3EH),A  ;AND DISPLAY
7F91 323F3C          LD      (3C3FH),A  ;ON VIDEO
7F94 E1              POP     HL      ;RESTORE 'HL'
7F95 C9              RET             ;RETURN TO CALLER

7F96 C5       READ   PUSH    BC      ;SAVE REGISTERS
7F97 E5              PUSH    HL      ;USED
7F98 Ø6Ø8            LD      B,8     ;READ 8 BITS PER BYTE
7F9A CDA27F          CALL    RDBIT   ;READ ONE BIT FROM TAPE
7F9D 1ØFB            DJNZ    $-3     ;READ ALL EIGHT BITS
7F9F E1              POP     HL      ;RESTORE
7FAØ C1              POP     BC      ;REGISTERS
7FA1 C9              RET             ;RETURN TO CALLER

7FA2 C5       RDBIT  PUSH    BC      ;SAVE NEEDED
7FA3 F5              PUSH    AF      ;REGISTERS
7FA4 DBFF            IN      A,(ØFFH)  ;LOOK FOR
7FA6 17              RLA             ;SYNCH PULSE
7FA7 3ØFB            JR      NC,$-3  ;DO UNTIL FOUND
7FA9 21FE7F          LD      HL,RDSP ;GET DJNZ1
7FAC 46              LD      B,(HL)  ;AND USE IT
7FAD 1ØFE            DJNZ    $       ;FOR A DELAY
7FAF CD1EØ2          CALL    21EH    ;RESET FLIP/FLOP
7FB2 21FF7F          LD      HL,RDSP+1  ;GET DJNZ2
7FB5 46              LD      B,(HL)  ;AND USE IT
7FB6 1ØFE            DJNZ    $       ;FOR A DELAY
7FB8 DBFF            IN      A,(ØFFH)  ;CHECK FLIP/FLOP
7FBA 47              LD      B,A     ;SAVE IN 'B'
7FBB F1              POP     AF      ;AND ON STACK
7FBC CB1Ø            RL      B       ;CHECK TO SEE IF
7FBE 17              RLA             ;PULSE WAS FOUND
7FBF F5              PUSH    AF      ;SAVE ON STACK
7FCØ CD1EØ2          CALL    21EH    ;RESET FLIP/FLOP
7FC3 F1              POP     AF      ;RESTORE
7FC4 C1              POP     BC      ;REGISTERS
7FC5 C9              RET             ;RETURN TO CALLER

7FC6 E5       SPEED  PUSH    HL      ;SAVE 'HL'
7FC7 21FØ7F          LD      HL,MSG2 ;DISPLAY
7FCA CDA728          CALL    28A7H   ;PROMPT
7FCD CD49ØØ          CALL    Ø49H    ;WAIT ON KEYBOARD INPUT
7FDØ FE46            CP      'F'     ;COMPARE WITH 'F'
7FD2 28Ø8            JR      Z,FAST  ;GO IF MATCH
7FD4 216Ø85          LD      HL,856ØH  ;SLOW RDSP
7FD7 22FE7F          LD      (RDSP),HL ;VARIABLE
7FDA 18Ø6            JR      PT1     ;CONTINUE
7FDC 212B41   FAST   LD      HL,412BH  ;FAST RDSP
7FDF 22FE7F          LD      (RDSP),HL ;VARIABLE
7FE2 CDFE2Ø   PT1    CALL    2ØFEH   ;SKIP DOWN ONE LINE
7FE5 E1              POP     HL      ;RESTORE 'HL'
7FE6 C9              RET             ;RETURN TO CALLER
```

```

7FE7 CD967F       RDHL    CALL    READ      ;READ NEXT BYTE
7FEA 6F                   LD      L,A       ;STORE IN 'L'
7FEB CD967F               CALL    READ      ;READ NEXT BYTE
7FEE 67                   LD      H,A       ;STORE IN 'H'
7FEF C9                   RET               ;RETURN TO CALLER

7FF0 28           MSG2    DEFM    '(FAST/SLOW)"'

7FFC 1741         WRSP    DEFW    4117H     ;1000 BITS PER SECOND
                                            ;DJNZ4 AND DJNZ3
7FFE 2B41         RDSP    DEFW    412BH     ;1000 BITS PER SECOND
                                            ;DJNZ2 AND DJNZ1

1A19                      END     1A19H
```

Program Listing 4. *ZBUG Modification*

```
;=================================
;            LISTING 4
;     ZBUG MODIFICATION  VER.1.2
;        1000 BITS PER SECOND
;           BY MARK TYLER
;=================================

; ----->  REQUIRES EDTASM THAT SUPPORTS
; ----->  MACROs (such as EDTASM+)
; ---------------------------------

;       SET UP MACRO TO CHANGE
;       CALLS TO ROM


NEW     MACRO   #ADD,#NAME
         ORG    #ADD
        CALL    #NAME
        ENDM
        NEW     4543H,TAPE
        NEW     459FH,WRITE
        NEW     4653H,TAPE
        NEW     4751H,TAPE
        NEW     47C9H,COPY
        NEW     45CCH,WSYNCH
        NEW     47DFH,WSYNCH
        NEW     454AH,SYNCH
        NEW     4663H,SYNCH
        NEW     4761H,SYNCH
        NEW     45D1H,WRTAPE
        NEW     45DAH,WRTAPE
        NEW     45EEH,WRTAPE
        NEW     45F3H,WRTAPE
        NEW     45F7H,WRTAPE
        NEW     45FBH,WRTAPE
        NEW     4601H,WRTAPE
        NEW     460AH,WRTAPE
        NEW     4618H,WRTAPE
        NEW     461CH,WRTAPE
        NEW     4620H,WRTAPE
```

Program continued

```
ØØ39Ø           NEW     4624H,WRTAPE
ØØ4ØØ           NEW     462AH,WRTAPE
ØØ41Ø           NEW     4633H,WRTAPE
ØØ42Ø           NEW     4638H,WRTAPE
ØØ43Ø           NEW     463FH,WRTAPE
ØØ44Ø           NEW     4643H,WRTAPE
ØØ45Ø           NEW     47F6H,WRTAPE
ØØ46Ø           NEW     455ØH,READ
ØØ47Ø           NEW     4557H,READ
ØØ48Ø           NEW     4562H,READ
ØØ49Ø           NEW     456CH,READ
ØØ5ØØ           NEW     4573H,READ
ØØ51Ø           NEW     457CH,READ
ØØ52Ø           NEW     466AH,READ
ØØ53Ø           NEW     4674H,READ
ØØ54Ø           NEW     467FH,READ
ØØ55Ø           NEW     4689H,READ
ØØ56Ø           NEW     469EH,READ
ØØ57Ø           NEW     46A5H,READ
ØØ58Ø           NEW     48ØFH,READ
ØØ59Ø           NEW     484EH,READ
ØØ6ØØ           NEW     4852H,READ
ØØ61Ø
ØØ62Ø
ØØ63Ø WRCMD     EQU     4AA8H    ;THESE ARE THE
ØØ64Ø GETCH     EQU     ØØ49H    ;ADDRESSES NEEDED
ØØ65Ø OUTSTR    EQU     28A7H    ;FOR SUBROUTINE CALLS
ØØ66Ø SETUP2    EQU     4A9EH    ;TO ROM OR ZBUG
ØØ67Ø
ØØ68Ø           ORG     4F2ØH    ;STORE OUR PATCHES
ØØ69Ø                            ;STARTING HERE
ØØ7ØØ
ØØ71Ø WSYNCH    LD      B,ØFFH   ;SET UP LOOP
ØØ72Ø           XOR     A        ;TO WRITE 256
ØØ73Ø           CALL    WRTAPE   ;Ø'S TO TAPE
ØØ74Ø           DJNZ    $-3      ;FOLLOWED BY
ØØ75Ø           LD      A,ØA5H   ;ØA5H BYTE
ØØ76Ø WRTAPE    PUSH    HL       ;SAVE NEEDED
ØØ77Ø           PUSH    BC       ;REGISTERS
ØØ78Ø           PUSH    DE       ;THEN SET UP
ØØ79Ø           PUSH    AF       ;TO WRITE
ØØ8ØØ           LD      C,8      ;8 BITS PER BYTE
ØØ81Ø           LD      D,A      ;STORE BYTE TO BE
ØØ82Ø                            ;WRITTEN IN 'A'
ØØ83Ø LP2       CALL    PULSE    ;WRITE SYNCH PULSE
ØØ84Ø           LD      A,D      ;RESTORE BYTE TO BE
ØØ85Ø                            ;WRITTEN TO 'A'
ØØ86Ø           RLCA             ;ROTATE BIT TO BE WRITTEN
ØØ87Ø                            ;TO CARRY FLAG
ØØ88Ø           LD      D,A      ;SAVE ROTATED BYTE IN 'D'
ØØ89Ø           JR      NC,LP4   ;JUMP IF CARRY FLAG RESET
ØØ9ØØ           CALL    PULSE    ;WRITE DATA PULSE
ØØ91Ø LP3       DEC     C        ;HAVE WE WRITTEN
ØØ92Ø           JR      NZ,LP2   ;ALL 8 BITS ?
ØØ93Ø           POP     AF       ;IF SO
ØØ94Ø           POP     DE       ;RESTORE ALL
ØØ95Ø           POP     BC       ;THE REGISTERS
ØØ96Ø           POP     HL
ØØ97Ø           RET              ;RETURN TO CALLER
ØØ98Ø
ØØ99Ø LP4       LD      HL,WRSP+1  ;GET DJNZ4
```

```
Ø1ØØØ          LD      B,(HL)   ;AND WASTE
Ø1Ø1Ø          DJNZ    $        ;SOME TIME
Ø1Ø2Ø          JR      LP3      ;GO BACK TO MAIN PROGRAM
Ø1Ø3Ø
Ø1Ø4Ø PULSE    LD      HL,ØFCØ1H  ;POSITIVE CONTROL WORD
Ø1Ø5Ø          CALL    Ø221H    ;WRIT POSITIVE PULSE
Ø1Ø6Ø          LD      B,ØBH    ;DELAY
Ø1Ø7Ø          DJNZ    $        ;138 'T' STATES
Ø1Ø8Ø          LD      HL,ØFCØ2H  ;NEGATIVE CONTROL WORD
Ø1Ø9Ø          CALL    Ø221H    ;WRITE NEGATIVE PULSE
Ø11ØØ          LD      B,ØBH    ;DELAY
Ø111Ø          DJNZ    $        ;138 'T' STATES
Ø112Ø          LD      HL,ØFCØØH ;TERMINATOR CONTROL WORD
Ø113Ø          CALL    Ø221H    ;WRITE TERMINATOR
Ø114Ø          LD      HL,WRSP  ;GET DJNZ3
Ø115Ø          LD      B,(HL)   ;AND DELAY
Ø116Ø          DJNZ    $        ;AWHILE
Ø117Ø          RET              ;RETURN TO CALLER
Ø118Ø
Ø119Ø READ     PUSH    BC       ;SAVE REGISTERS
Ø12ØØ          PUSH    HL       ;USED
Ø121Ø          LD      B,Ø8H    ;READ 8 BITS PER BYTE
Ø122Ø          CALL    RDBIT    ;READ 1 OF THE BITS
Ø123Ø          DJNZ    $-3      ;DONE 8 TIMES?
Ø124Ø          POP     HL       ;RESTORE
Ø125Ø          POP     BC       ;REGISTERS USED
Ø126Ø          RET              ;RETURN TO CALLER
Ø127Ø
Ø128Ø RDBIT    PUSH    BC       ;SAVE NEEDED
Ø129Ø          PUSH    AF       ;REGISTERS
Ø13ØØ          IN      A,(ØFFH)   ;LOOK FOR
Ø131Ø          RLA              ;SYNCH PULSE
Ø132Ø          JR      NC,$-3   ;DO UNTIL FOUND
Ø133Ø          LD      HL,RDSP  ;GET DJNZ1
Ø134Ø          LD      B,(HL)   ;AND DELAY
Ø135Ø          DJNZ    $        ;AWHILE
Ø136Ø          CALL    Ø21EH    ;RESET FLIP/FLOP
Ø137Ø          LD      HL,RDSP+1   ;GET DJNZ2
Ø138Ø          LD      B,(HL)   ;AND DELAY
Ø139Ø          DJNZ    $        ;AWHILE
Ø14ØØ          IN      A,(ØFFH)   ;CHECK FLIP/FLOP
Ø141Ø          LD      B,A      ;SAVE IN 'B'
Ø142Ø          POP     AF       ;RESTORE 'A' FROM STACK
Ø143Ø          RL      B        ;CHECK TO SEE IF
Ø144Ø          RLA              ;FLIP/FLOP WAS SET
Ø145Ø          PUSH    AF       ;SAVE RESULTS ON STACK
Ø146Ø          CALL    Ø21EH    ;RESET FLIP/FLOP
Ø147Ø          POP     AF       ;RESTORE
Ø148Ø          POP     BC       ;REGISTERS
Ø149Ø          RET              ;RETURN TO CALLER
Ø15ØØ
Ø151Ø SYNCH    PUSH    HL       ;SAVE ON STACK
Ø152Ø          XOR     A        ;LOAD 'A' WITH Ø
Ø153Ø          CALL    RDBIT    ;READ FROM TAPE
Ø154Ø          CP      ØA5H     ;IS IT THE ØA5H BYTE?
Ø155Ø          JR      NZ,$-5   ;NO----KEEP TRYING
Ø156Ø          LD      A,2AH    ;'*' TO 'A' REGISTER
Ø157Ø          LD      (3C3EH),A   ;AND DISPLAY
Ø158Ø          LD      (3C3FH),A   ;ON VIDEO
Ø159Ø          POP     HL       ;RESTORE 'HL'
Ø16ØØ          RET              ;RETURN TO CALLER
```

Program continued

```
01610
01620 RDSP     DEFW     412BH    ;1000 BITS/SECOND
01630                            ;DJNZ4 AND DJNZ3
01640 WRSP     DEFW     4117H    ;1000 BITS/SECOND
01650                            ;DJNZ2 AND DJNZ1
01660
01670 TAPE     CALL     WRCMD    ;ZBUG SUBROUTINE TO
01680          DEFM     ' (F/S) ,'  ;WRITE MESSAGE
01690          CALL     GETCH    ;ZBUG SUBROUTINE
01700                            ;TO GET REPLY
01710          CP       'F'      ;IS REPLY AN 'F'
01720          JR       Z,FAST   ;YES---GOTO FAST
01730          LD       HL,8560H  ;OTHERWISE LOAD
01740          LD       (RDSP),HL ;RDSP AND WRSP
01750          LD       HL,875CH  ;WITH SLOW
01760          LD       (WRSP),HL ;SPEED VARIABLES
01770          RET               ;RETURN TO CALLER
01780
01790 FAST     LD       HL,412BH  ;LOAD RDSP
01800          LD       (RDSP),HL ;AND WRSP
01810          LD       HL,4117H  ;WITH FAST
01820          LD       (WRSP),HL ;SPEED VARIABLES
01830          RET               ;RETURN TO CALLER
01840
01850 COPY     CALL     OUTSTR   ;NEED MORE ROOM
01860          CALL     TAPE     ;IN ZBUG PROGRAM
01870          RET               ;RETURN TO CALLER
01880
01890 WRITE    CALL     TAPE     ;NEED MORE ROOM
01900          JP       SETUP2   ;AGAIN
01910          END      4338H    ;ZBUG STARTING POINT
*
```

30

Blinking and Repeating on the Model I

by Craig A. Lindley

System Requirements:
Model I
16K RAM
One disk drive
TRSDOS-compatible DOS

The Model I, unlike newer and more expensive computers, lacks a blinking cursor and repeating keys. This utility adds these capabilities to your Model I. Like most other special keyboard drive routines, NEWDVR patches itself into the keyboard device control block (DCB) so that it, not the normal keyboard driver routine, is executed when the keyboard is polled. The computer is continually polling the keyboard in order to respond to input immediately.

Essentially, NEWDVR directs the computer to:
•Blink cursor on then off until a key is pressed.
•Transmit the valid key code immediately to the operating system.
•If the key is held down, delay before repeating the key.
•After delay is up, repeat key until key is released.
I have successfully tested this routine in BASIC and machine-language programs.

How It Works

The first portion of code, labeled PATCH, puts the address of the NEWDVR routine into the keyboard DCB (see Program Listing). The PATCH routine also places the address of the normal keyboard scan routine, taken from the DCB, into the NEWDVR code at two places, so you can call the keyboard scan routine to get keyboard characters from the operator. PATCH also protects NEWDVR from being overwritten, by storing its address, minus one, in the DOS top-of-memory pointer at 4049H.

The first part of the NEWDVR code checks to see if the routine calling the keyboard scan is the wait-for-key routine at 49H in ROM. If so, the blinking cursor routine is executed. If not, the normal key scan routine is initiated. This function is checked by looking back on the stack to see what the return address of the calling routine is. An address of 4CH, ten bytes back on the stack, indicates the routine at 49H is calling. Any other address indicates the calling program is doing a keyboard scan, probably just to check whether the BREAK key is active, not waiting for input. This feature was added because quite a few TRS-80 programs scan the keyboard constantly for a BREAK command. This is normally invisible, but becomes painfully obvious if the cursor has to blink before the calling program can continue execution.

If the NEWDVR routine decides that the blinking cursor routine should be executed, the first thing it does is turn off the normal cursor character, by outputting the cursor-off character code of 15 decimal to the ROM display routine at 33H. Then the address of the cursor is loaded into HL and the character at the current address of the cursor is loaded into register C for safekeeping. The new block cursor character (code 143 decimal) is stored over the original character at the current address of the cursor. The RDKEYS routine, described below, is called to read the keyboard repeatedly until either a key press is detected or a predetermined time period elapses. This time period is controlled by the BLINK equate in the listing.

The RDKEYS routine returns with the Z flag reset, if a key was detected; or set, if not. The original character at the cursor position is replaced. If the RDKEYS routine detected a key, control returns to the operating system with that character in the A register. If not, the RDKEYS routine is again called to see if a key has now been pressed. Control returns unconditionally to the operating system after this call, with the Z flag set accordingly. Calling the RDKEYS subroutine twice, first with the block cursor character at the cursor address, and then with the original character at the cursor address, causes the cursor to blink at a 50/50 duty cycle (half on, half off).

The RDKEYS routine performs the actual polling of the keyboard. The number of times it polls before returning to the NEWDVR routine is controlled by the BLINK time constant, which is contained in DC. This count decrements one each time the keyboard is polled and no key has been pressed. The delay count (IX) and the first-time flag (IX + 1) are reset to zero to indicate that a key was not found. The delay count determines how long after the key is held down the repeat function begins. The first-time flag is zero before a new key is pressed, and one afterwards.

If the BLINK count reaches 0FFH, or 255, before a key is detected, the RDKEYS routine returns to the NEWDVR routine with Z = 1 indicating this. Clearing the 7-byte keyboard work area at 4036H to zero before polling the keyboard causes the normal keyboard driver routine to return a key code, even if the key has not been released. Normally, the keyboard driver routine does not return a key code until another key is pressed, or the original key was released and pressed again.

If the RDKEYS routine determines that a key has been pressed before the BLINK count is exhausted, control transfers to the FOUND routine for additional processing. There, the key character code is placed in the B register for temporary storage. The delay count is loaded into the accumulator and a comparison is done to see if the count equals zero. The count equals zero when a key is first pressed, or after the delay time counter has overflowed. If a key was just pressed, control transfers to F1, and the keyboard character stored in B is returned to the A register.

A test is performed to see whether this is a *new* keypress, signified by the first-time flag equaling zero. If so, the delay count is incremented to one and the first-time flag is set to one, so that these instructions are skipped the next time the F1 routine is entered. The F2 routine is executed whether or not this was the first time through F1. F2 resets the Z flag to indicate that a key has been pressed, before returning the key character in the A register to the NEWDVR routine and eventually to the operating system.

If the key remains pressed the next time the keyboard is polled, control does not pass to the F1 routine because the delay count is not zero. This causes the delay count to shift, the accumulator to clear, the Z flag to be set, and control to return to the NEWDVR routine. This means a delay before the keys start to repeat, because when the delay count doesn't equal zero, this routine passes the keyboard null code of 00 back to the operating system, just as if no keys were being pressed. When shifting the count causes the count to again equal zero, the F1 routine is executed again. This time, however, the F1 routine doesn't increment the delay count, so the keys repeat at full speed.

Using this Program

Place this program in memory one byte at a time, by using a monitor program such as DEBUG and writing it to tape or disk. Or, assemble it as a DOS/CMD file, using an editor/assembler. Change the program ORG, as shown on the listing, to reflect your computer's memory size. If your system is non-disk, change the jump to the operating system in line 250 of the listing to a return to BASIC (1A19H). Make sure the memory location of this program doesn't conflict with any other high memory programs.

Program Listing

```
                ;****************************************
                ;***          BLINKING CURSOR         ***
                ;***         REPEAT KEY UTILITY       ***
                ;***            VERSION  1.2          ***
                ;***         NOVEMBER 26, 1981        ***
                ;***                BY                ***
                ;***          CRAIG A. LINDLEY        ***
                ;****************************************
                ;
                ;SYSTEM EQUATES
                ;
0400            BLINK   EQU     0400H           ;CURSOR BLINK TIME CONST.
0033            CHROUT  EQU     0033H           ;VIDEO CHAR OUT ROUTINE
4020            CURSOR  EQU     4020H           ;CURSOR ADDR STORAGE
4016            KEYDCB  EQU     4016H           ;KEYBOARD DCB
402D            OPSYS   EQU     402DH           ;DOS REENTRY POINT
4049            TOPMEM  EQU     4049H           ;DOS TOP MEM PTR
4036            WKAREA  EQU     4036H           ;KEY DVR WORK AREA
                ;
                ;**********
                ;PROGRAM ORGS FOR VARIOUS MEMORY SIZES
                ;48K - FF70H
                ;32K - BF70H
                ;16K - 7F70H
                ;**********
                ;
FF70                    ORG     0FF70H
                ;
                ;**********
                ;START OF PROGRAM
                ;**********
                ;
FF70 2A1640     PATCH   LD      HL,(KEYDCB)     ;GET NORM KEY DRIVER ADDR
FF73 2206FF             LD      (SCAN2+1),HL    ;PLACE IN CODE FOR CALL
FF76 2296FF             LD      (NEWD1+1),HL
FF79 2188FF             LD      HL,NEWDVR       ;GET NEW DVR ADDR
FF7C 221640             LD      (KEYDCB),HL     ;PLACE IN KEYBOARD DCB
FF7F 2187FF             LD      HL,NEWDVR-1     ;MEMORY PROTECT LIMIT
FF82 224940             LD      (TOPMEM),HL     ;STORE FOR DOS
FF85 C32D40             JP      OPSYS           ;BACK TO OPSYS
                ;
                ;**********
                ;NEW KEYBOARD DRIVER ROUTINE
                ;**********
                ;
FF88 ED73B3FF   NEWDVR  LD      (STKPTR),SP     ;GET STACK PTR ADDR
FF8C DD2AB3FF           LD      IX,(STKPTR)     ;INTO IX
FF90 3E4C               LD      A,4CH           ;LSB OF CALL TO 49H
                                                ;RET ADDRESS
FF92 DDBE0A             CP      (IX+10)         ;CALL FROM 49H ?
FF95 C20000     NEWD1   JP      NZ,$-$          ;IF NOT THEN
                ;
FF98 DD21B1FF           LD      IX,STATUS       ;PT AT STATUS
FF9C 3E0F               LD      A,15            ;CURSOR OFF CODE
FF9E CD3300             CALL    CHROUT          ;OUTPUT CODE
FFA1 2A2040             LD      HL,(CURSOR)     ;GET CURSOR LOCATION
FFA4 4E                 LD      C,(HL)          ;GET CHAR AT CURSOR
FFA5 3E8F               LD      A,143           ;BLOCK CURSOR CODE
FFA7 77                 LD      (HL),A          ;STORE AT CURSOR LOCATION
```

```
FFA8 CDB5FF   ØØ69Ø         CALL    RDKEYS          ;READ KEYBOARD
FFAB 71       ØØ7ØØ         LD      (HL),C          ;PUT ORIGINAL CHAR AT
              ØØ71Ø                                 ;CURSOR BACK
FFAC CØ       ØØ72Ø         RET     NZ              ;IF KEY FOUND THEN
FFAD CDB5FF   ØØ73Ø         CALL    RDKEYS          ;READ KEYBOARD AGAIN
FFBØ C9       ØØ74Ø         RET
              ØØ75Ø ;
              ØØ76Ø ;SYSTEM STORAGE LOCATIONS
              ØØ77Ø ;
ØØØ1          ØØ78Ø STATUS  DEFS    1               ;DELAY COUNT
ØØØ1          ØØ79Ø         DEFS    1               ;1ST TIME FLAG
ØØØ2          ØØ8ØØ STKPTR  DEFS    2               ;STACK PTR STORAGE
              ØØ81Ø ;
              ØØ82Ø ;***********
              ØØ83Ø ;SUBROUTINES
              ØØ84Ø ;***********
              ØØ85Ø ;
FFB5 11ØØØ4   ØØ86Ø RDKEYS  LD      DE,BLINK        ;LOAD TIME CONSTANT
FFB8 E5       ØØ87Ø SCAN    PUSH    HL              ;SAVE REGS
FFB9 D5       ØØ88Ø         PUSH    DE
FFBA C5       ØØ89Ø         PUSH    BC
FFBB 21364Ø   ØØ9ØØ         LD      HL,WKAREA       ;CLEAR KEYBOARD WORK AREA
FFBE Ø6Ø7     ØØ91Ø         LD      B,7             ;TO MAKE DRIVER RETURN A
FFCØ 36ØØ     ØØ92Ø SCAN1   LD      (HL),Ø          ;KEY CODE EVERYTIME IT IS
FFC2 23       ØØ93Ø         INC     HL              ;CALLED.
FFC3 1ØFB     ØØ94Ø         DJNZ    SCAN1
FFC5 CDØØØØ   ØØ95Ø SCAN2   CALL    $-$             ;READ KEYBOARD USING
              ØØ96Ø                                 ;NORMAL KEY DRIVER
FFC8 B7       ØØ97Ø         OR      A               ;NO KEY PRESSED ?
FFC9 2ØØ8     ØØ98Ø         JR      NZ,SCAN3        ;IF KEY PRESSED THEN
FFCB DD36ØØØØ ØØ99Ø         LD      (IX),Ø          ;RESET DELAY COUNT
FFCF DD36Ø1ØØ Ø1ØØØ         LD      (IX+1),Ø        ;RESET 1ST TIME FLAG
FFD3 C1       Ø1Ø1Ø SCAN3   POP     BC              ;RESTORE REGS
FFD4 D1       Ø1Ø2Ø         POP     DE
FFD5 E1       Ø1Ø3Ø         POP     HL
FFD6 2ØØ5     Ø1Ø4Ø         JR      NZ,FOUND        ;IF KEY PRESSED THEN
FFD8 1B       Ø1Ø5Ø         DEC     DE              ;DEC BLINK COUNT
FFD9 BA       Ø1Ø6Ø         CP      D               ;BLINK TIME FINISHED ?
FFDA 2ØDC     Ø1Ø7Ø         JR      NZ,SCAN         ;IF NOT READ KEYS AGAIN
FFDC C9       Ø1Ø8Ø         RET
              Ø1Ø9Ø ;
FFDD 47       Ø11ØØ FOUND   LD      B,A             ;SAVE KEY CHAR
FFDE DD7EØØ   Ø111Ø         LD      A,(IX)          ;GET DELAY COUNT
FFE1 FEØØ     Ø112Ø         CP      Ø               ;IS IT ZERO ?
FFE3 28Ø6     Ø113Ø         JR      Z,F1            ;IF YES THEN
FFE5 DDCBØØ26 Ø114Ø         SLA     (IX)            ;SHIFT COUNT
FFE9 AF       Ø115Ø         XOR     A               ;"A" = Ø
FFEA C9       Ø116Ø         RET
FFEB 78       Ø117Ø F1      LD      A,B             ;RESTORE CHAR
FFEC DDCBØ146 Ø118Ø         BIT     Ø,(IX+1)        ;1ST TIME ?
FFFØ 2ØØ7     Ø119Ø         JR      NZ,F2           ;IF NOT THEN
FFF2 DD34ØØ   Ø12ØØ         INC     (IX)            ;BUMP COUNT
FFF5 DDCBØ1C6 Ø121Ø         SET     Ø,(IX+1)        ;RESET 1ST TIME FLAG
FFF9 FEØØ     Ø122Ø F2      CP      Ø               ;SET NZ FLAG
FFFB C9       Ø123Ø         RET
              Ø124Ø ;
              Ø125Ø ;
FF7Ø          Ø126Ø         END     PATCH
```

31

SYNC: Automatic Start and Memory Size Setting

by Theodore J. LeSarge

System Requirements:
Model I
16K RAM
Editor/assembler

I wrote SYNC to provide automatic start and automatic memory size setting for my tape-based BASIC programs. SYNC assigns a number to each program CSAVEd and lets you CLOAD it by entering its identification number. You can choose any number from 1-255 (except 165, a code used by the computer to signal the end of the header pulses used to synchronize the data pulse and the computer's clock).

It's easy to relocate the program. Just make sure the new origin in line 190 and the END statement in line 1160 are the same. Lines 200-210 reset the automatic start. Lines 220-250 answer the memory size question at power-up. The computer stores the memory size, minus two, in memory location 40B1H. The TRS-80 automatically sets string space to 50 bytes and stores that number at location 40A0H. The HL register pair is loaded with the location in memory of the label DEF, minus 52. This number is placed at 40A0H. The process then repeats, but minus two. This number is stored at 40B1H.

When your non-disk TRS-80 is ready, memory locations 4152H-41A5H are set to jump to the L3 (Disk BASIC only) error display. Since these locations are in user RAM, you can easily detour the interpreter and use disk commands to execute your own machine-language programs. Lines 270-320 do just that. These locations already contain the JP (195 decimal) op code; just enter the 2-byte address of your program's entry point.

Line 340 returns you to BASIC. Note that I used 06CCH rather than the popular 1A19H for the entry point. A jump to 1A19H often produces the ?OM (out of memory) error when the next command in BASIC is executed. When you type DEF in the command mode, the computer jumps to the memory location specified in line 360. Line 620 uses a ROM routine at 01C9H to clear the video display. Lines 630-660 set the cursor to the eighth line of the screen, set up the HL register pair to the message address at line 1060, and use ROM routine 28A7H for display. Line 670 sets the HL pair to the start of the buffer location. Lines 680-800 allow up to three numbers to be entered without pressing ENTER. Line 690 uses the ROM keyboard scan operation.

The A register now contains the character(s) entered. This is compared with 0DH, the code for ENTER. If ENTER is pressed for the first number, the computer responds with the ?MO (missing operand) error. Lines 730-760 trap non-numeric input. When a number is entered, line 770 loads it into a buffer. Line 780 uses the ROM display location 033H. The character in the A register is displayed at the current cursor location. Line 790 moves the HL pair to the next memory location in the buffer, while line 800 decrements the B register and checks for a non-zero result. After three numbers are entered, the program loop falls through and line 810 is executed. If fewer than three numbers are entered, the program goes back to the keyboard scan. Lines 810 and 820 set the last location in the buffer to zero, used in the next instruction.

Decimal Conversions

When you enter a number in response to the computer's prompt, the program uses the ROM keyboard scan at 049H, and loads the A register with the ASCII value of your input. For decimal 145, these would be 31H, 34H, and 35H. Each number is converted to binary coded decimal (BCD) by subtracting 30H from the ASCII value. Then each digit is multiplied by the power of ten it represents. This is a long process; the ROM routine at location 2B02H does the work. Each number input is stored in a buffer. The buffer is terminated with a zero (lines 810 and 820), the HL pair is set to the beginning address of the buffer, and 2B02H is called. The result is put in the DE register pair.

You can use this routine for numbers up to 32767; numbers beyond that give you an error. Because this program uses numbers up to 255 (the maximum for a single register), the answer from 2B02H is in the E register. Line 860 stores this number in memory. Lines 870-920 check for a synchronization byte of less than one or more than 255. The A register is set to zero and the HL pair is loaded with 255. If the E register is equal to the A register or greater than the HL pair, you get an error message. Lines 950-1040 set up and display the error message. A delay routine holds the display for about three seconds before returning you to line 620 for another try. If everything is entered correctly, the operation returns to 370, which jumps to the BASIC command mode.

To save your BASIC program, type SAVE and press ENTER. A file name is not needed. The program jumps to line 390, the cassette recorder starts, a leader of 255 zeros is written to tape, and return points used by ROM are saved on the stack. Line 480 loads the A register with your next synchronization byte; this also goes on the tape. Line 490 jumps to the remainder of the normal CSAVE instructions in ROM. Your BASIC program is now saved on tape with its code number. Confirm the save with LOAD?.

To load a program type DEF, type the identification number and enter the word LOAD. Lines 510-600 start the cassette recorder, load the DE register pair, and compare the value in the DE register to what the TRS-80 reads from the tape. When there is a match, lines 570-590 display two asterisks in the upper corner. If one of them doesn't blink, you entered the wrong code number; press RESET to continue.

Now, for the automatic start: lines 1140-1160 place the instruction "jump to where HL points" at 41E2H. The END statement loads the program start into HL. Then the first system command calls 41E2H, normally C9H or a return function. When this program loads, the computer jumps to the start of the program. Lines 200 and 210 reset this location so the system command operates correctly the next time.

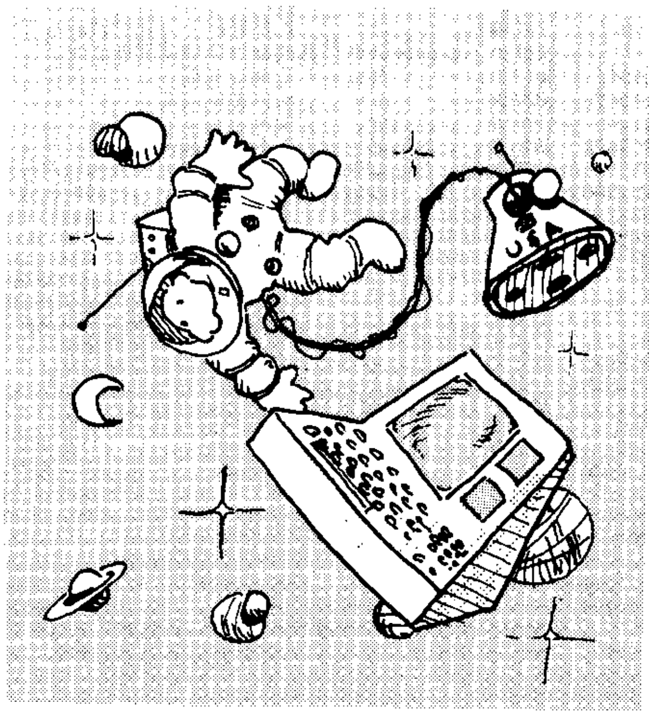
Program Listing

```
               00100 ;BASIC CODING PROGRAM
               00110 ;CREATES NEW SYNC BYTE
               00120 ;THEODORE J. LESARGE
               00130 ;6027 W. DECKER ROAD
               00140 ;LUDINGTON, MI. 49431
               00150 ;(616) 845-6905
               00160 ;JULY 20, 1981
               00170
               00180
7F20           00190       ORG    7F20H        ;32544
7F20 3EC9      00200       LD     A,0C9H       ;RESET
7F22 32E241    00210       LD     (41E2H),A    ; AUTO START
7F25 21157F    00220       LD     HL,DEF-52    ;SET
7F28 22A040    00230       LD     (40A0H),HL   ; STRING SPACE
7F2B 21477F    00240       LD     HL,DEF-2     ;SET
7F2E 22B140    00250       LD     (40B1H),HL   ; MEM SIZE
               00260 ;SET DISK BASIC ENTRY WORDS
7F31 214F7F    00270       LD     HL,CSAVE
7F34 22A141    00280       LD     (41A1H),HL   ;"SAVE" WORD
7F37 21687F    00290       LD     HL,LOAD
7F3A 228941    00300       LD     (4189H),HL   ;"LOAD" WORD
7F3D 21497F    00310       LD     HL,DEF
7F40 225C41    00320       LD     (415CH),HL   ;"DEF" WORD
7F43 CD491B    00330       CALL   1B49H        ;NEW
7F46 C3CC06    00340       JP     06CCH        ;READY//NOT 1A19H
               00350
7F49 CD827F    00360 DEF   CALL   SCN
7F4C C3CC06    00370       JP     06CCH        ;READY
```

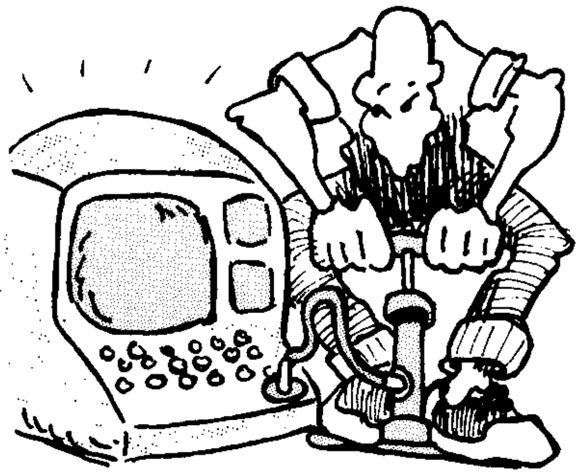
```

7F4F 21CCØ6       CSAVE  LD     HL,Ø6CCH       ;RET POINTS
7F52 E5                  PUSH   HL             ; ROM WILL
7F53 21FB2B              LD     HL,2BFBH       ;  USE
7F56 E5                  PUSH   HL
7F57 CDFEØ1              CALL   Ø1FEH          ;ON TAPE
7F5A Ø6FF                LD     B,ØFFH         ;255
7F5C AF                  XOR    A              ; ØØØØØ'S
7F5D CD64Ø2              CALL   Ø264H          ;WRITE
7F6Ø 1ØFB                DJNZ   $-Ø3H          ;B=Ø?
7F62 3AFE7F              LD     A,(SYNC)       ;NEW SYNC BYTE
7F65 C364Ø2              JP     Ø264H          ;BACK TO ROM

7F68 CDFEØ1       LOAD   CALL   Ø1FEH          ;ON TAPE
7F6B E5                  PUSH   HL
7F6C CD41Ø2              CALL   Ø241H          ;READ
7F6F ED5BFE7F            LD     DE,(SYNC)      ;NEW SYNC BYTE
7F73 BB                  CP     E              ;A SAME AS E
7F74 2ØF6                JR     NZ,$-Ø8H       ;NO? GO AGAIN
7F76 3E2A                LD     A,2AH          ; *
7F78 323E3C              LD     (3C3EH),A      ;SCREEN
7F7B 323F3C              LD     (3C3FH),A      ;SAME+1
7F7E E1                  POP    HL
7F7F C3222C              JP     2C22H          ;BACK TO ROM
7F82 CDC9Ø1       SCN    CALL   Ø1C9H          ;CLS
7F85 21CØ3D              LD     HL,3DCØH       ;SCREEN LOC
7F88 22204Ø              LD     (4Ø2ØH),HL     ;SET CURSOR
7F8B 21D97F              LD     HL,MESG1       ;SET PTR
7F8E CDA728              CALL   28A7H          ;DISPLAY IN ROM
7F91 21FA7F              LD     HL,BUFFER      ;SET PTR
7F94 Ø6Ø3                LD     B,Ø3H          ;THREE NUMBERS
7F96 CD49ØØ              CALL   Ø49H           ;KEYBD SCN
7F99 FEØD                CP     ØDH            ;ENTER PRESSED?
7F9B 2811                JR     Z,$+13H        ;YES! CONTINUE
                  ;NUMBER TEST ROUTINE/NUMBERS ONLY
7F9D FE3Ø                CP     Ø3ØH
7F9F FA967F              JP     M,$-Ø9H
7FA2 FE3A                CP     Ø3AH
7FA4 F2967F              JP     P,$-ØEH
7FA7 77                  LD     (HL),A         ;NO. TO BUFF
7FA8 CD33ØØ              CALL   Ø33H           ;DISPLAY IN ROM
7FAB 23                  INC    HL             ;BUMP ONE
7FAC 1ØE8                DJNZ   $-Ø16H         ;B=Ø? NO GO AGAIN
7FAE AF                  XOR    A              ;ZERO IN A
7FAF 77                  LD     (HL),A         ;DELIMITER FOR BUFF
7FBØ 21FA7F              LD     HL,BUFFER      ;SET PTR
                  ;ROM ROUTINE CONVERTS ASCII NUMERIC/RESULTS IN DE
7FB3 CDØ22B              CALL   2BØ2H          ;VERY USEFUL
7FB6 ED53FE7F            LD     (SYNC),DE      ;STORE
7FBA AF                  XOR    A              ;ZERO A REG
7FBB BB                  CP     E              ;E=Ø?
7FBC 28Ø7                JR     Z,ERROR        ;YES/END
7FBE 21FFØØ              LD     HL,ØFFH        ;255 MAX NUMBER
7FC1 DF                  RST    18H            ;DE>HL?
7FC2 38Ø1                JR     C,ERROR        ;YES IF CARRY
7FC4 C9                  RET

7FC5 21EC7F       ERROR  LD     HL,ERRMSG      ;SET PTR
7FC8 CDA728              CALL   28A7H          ;DISPLAY
                  ;DELAY ROUTINE FOR ERROR DISPLAY--ABOUT 3 SECONDS
7FCB Ø6Ø3                LD     B,Ø3H          ;THREE LOOPS
```

Program continued

```
7FCD 21FFFF   00990          LD      HL,0FFFFH      ;65535
7FD0 2B       01000          DEC     HL             ;ONE LESS
7FD1 7C       01010          LD      A,H            ;H TO A
7FD2 B5       01020          OR      L              ;ZERO TEST
7FD3 20FB     01030          JR      NZ,$-03H       ;NOPE?
7FD5 10F6     01040          DJNZ    $-08H          ;B=0?
7FD7 18A9     01050          JR      SCN            ;BACK AGAIN
7FD9 45       01060 MESG1    DEFM    'ENTER CODE NUMBER '
7FEB 00       01070          DEFB    0              ;STOPS OUTPUT
7FEC 20       01080 ERRMSG   DEFM    '    (1 TO 255)'
7FF9 00       01090          DEFB    0
0004          01100 BUFFER   DEFS    04H            ;BUFFER LENGTH
7FFE 0000     01110 SYNC     DEFW    0              ;NEW SYNC BYTE STORE
              01120
              01130 ;SET UP FOR AUTO START
41E2          01140          ORG     41E2H          ;SET UP
41E2 E9       01150          JP      (HL)           ;  AUTO START
7F20          01160          END     7F20H
00000 TOTAL ERRORS
```



Index



31 all new tutorials and utilities

From the overflowing files of *80 MICRO*: The very best articles from files full of manuscripts too good to pass up. These 31 useful, clearly documented tutorials and utilities were chosen, edited, and formatted to last for years of practical computing. Beginning and advanced programmers alike will find time- and effort-saving articles for the TRS-80 Model I and Model III. From line drawing to debugging, from sorting routines to adding commands and automatic features, you'll find THE REST OF 80 a goldmine of useful information.

ISBN 0-88006-062-X

$9.97