AN INTER-COMPUTER COMMUNICATIONS SYSTEM FOR A PERSONAL COMPUTER

by

DANIEL RAY VESTAL

B.S., Cameron University, 1973

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

[illegible]

Major Professor

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND: The development of the low cost micro processor chip has led to the introduction of the personal or home computer system. Mass production and the resulting low prices now offer everyone the opportunity to purchase their own computer. These relatively inexpensive "home computers" give their owners many capabilities that, fewer than twenty years ago, were available only on large "mainframe" computers costing millions of dollars . There are still many functions, however, that are better accomplished by larger computer systems due to the slower processing speed and restricted memory size of a micro computer.

Personal computers are often used as a remote terminals to access a larger host computer by connecting an acoustic modem between a standard telephone and the home computer (Figure 1) and then executing a simple terminal program. The owner is therefore saved the cost of a separate computer terminal and also has the convenience of working at the place of his choice rather than traveling to the actual location of the computer system to gain access to a terminal.

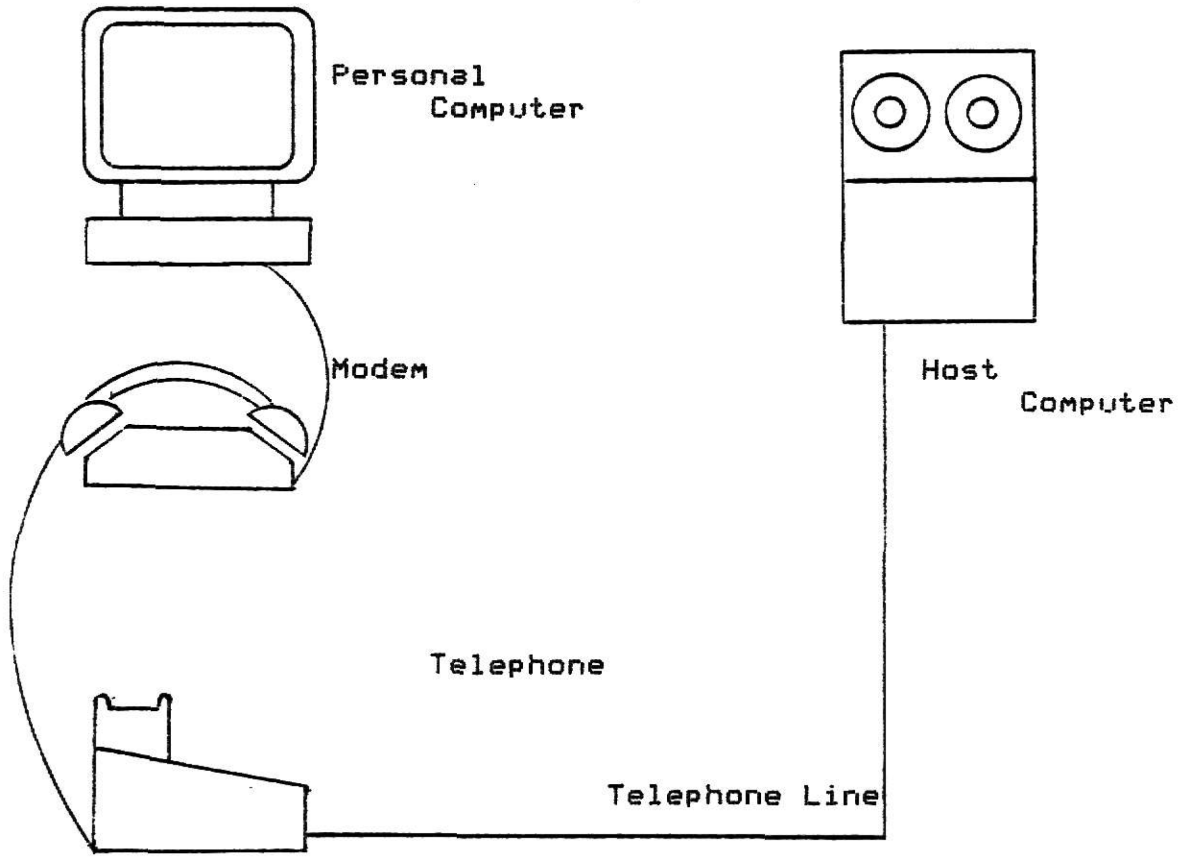

Fig. 1. A Personal Computer Used as a Remote Terminal

There are two principal limitations confronting the remote terminal user. First, the user may not be able to use the host system whenever he wants. The number of users that can be connected to a host system, at any one time, is limited. If all the connect lines are busy the remote terminal user wishing to use the host system must wait until one of the other users, that was already connected, completes his session and disconnects from the host system.

Second, while a terminal is connected, much of the time is spent waiting for input from the terminal user, and is therefore unproductive. For example, terminals that use unconditioned telephone lines to communicate with a remote host normally transfer data at 300 baud. At this transfer rate a terminal user would have to type at approximately 300 words/minute (300 baud = approx 300 bits/sec = approx 30 char/sec = approx 5+ words/sec = approx 300 wpm) to equal the speed at which the system can accept data. At a more realistic typing rate of 50-60 wpm, data is being entered at approximately 20% of the system's input capability. When "composing time" between entering statements is considered, the entry rate is, of course, much less. This time spent waiting is especially important if a user is being charged for connect time. It is simply not cost effective to pay for the time during which the terminal is connected to the host and data is not being transferred.

1.2 GENERAL REQUIREMENT - Many of the programs being marketed to allow a personal computer to act as a remote terminal perform only those functions normally handled by hardware in a standard terminal. An ASCII character is transmitted when a key is typed on the keyboard and any character received by the modem is displayed/printed. Figure 2 is a simple algorithm for a program to perform these functions.

```
Do forever
    Begin
        If keyboard character ready then
            begin
                read keyboard
                wait until transmitter not busy
                send character to transmitter
            end
        If receiver has data then
            begin
                read receiver
                display character from receiver
            end
     end
End Do_forever loop
```

Fig. 2. An Algorithm to Simulate a Hardware Terminal

A personal computer can easily be programmed to perform these functions. However, a solution to the limitations of using a remote terminal is to utilize the capabilities of the personal computer to make the maximum effective use of the time while the terminal is connected. More effective use of the connect time can be accomplished by limiting the transactions conducted to only those that must be passed between the machines and then passing those transactions at the maximum transfer rate. The number of transactions between the terminal and the host can be reduced by performing as much processing as possible on the micro computer before the connection is made with the host. For example, an editor sytem on the micro computer can be used to enter and edit the program text or data. This can be accomplished before the connection is made to the host. If a reasonably compatible compiler is available, preliminary debugging may also be performed on the micro computer system before the program is transferred to the host for final compilation. Once the program has been prepared, the connection can be made and the program transmitted to the host, under program control by the micro computer, at the maximum allowable speed. It follows that if each remote user's session is limited by these techniques more remote users would be able to access the system.

1.3 REPORT ORGANIZATION

- It is the intent of this project to identify and implement a minimum set of functions that make effective use of a home computer's capabilities when it is used as a remote terminal. The general requirement for this capability has been discussed in Chapter 1. Chapter 2 refines this requirement into a set of functions for implementation. In Chapter 3 the actual project implementation is discussed. Finally, Chapter 4 summarizes the results of this project and discusses possible enhancements.

CHAPTER 2

SYSTEM DESIGN

2.1 INTRODUCTION - In Chapter 1 the limitations imposed on a remote terminal user and the possible advantages of using a micro computer system as a remote terminal were discussed. The microcomputer's capabilities for stand-alone processing and data manipulation/storage can be used to assist the user in making the most effective use of his time when using a remote terminal. This chapter covers the design of a micro computer inter-computer communications system to provide these features to the remote terminal user.

2.2 DESIGN CONSIDERATIONS - During the design of this system the following broad areas were considered.

2.2.1 FUNCTIONALITY - To be more effective than a low cost hardware device or a simple terminal program, the communications system must improve the user's ability to transfer information. To accomplish this the system must be able to: (1) prepare text off-line and transmit it later by more efficient means than manually typing it on-line and (2) receive and save information so that it can be reviewed or processed after the communications link is disconnected.

2.2.2 PORTABILITY - The usefulness of the system can be increased by a design that allows it to be used with (portable between) a variety of host systems. *The only* thing reasonably standard about inter-computer communications is that most computers recognize the American Standard Code for Information Interchange (ASCII). To be portable between hosts, therefore, requires the ability to modify those other parameters that may be host system specific. (For example communications parameters such as baud rate and word length or security parameters such as sign-on and sign-off messages).

The source programs of this prototype are not intended to be readily transportable to other micro computers due to language and hardware incompatabilites; however, the system does provide a structure that could be modified or used as a model from which others could develop similar systems.

2.2.3 FLEXIBILITY - Since this system is being developed as a prototype, the design must allow the system to be easily modified so that future enhancements may be easily implemented. A modular approach must be used so that features can be added, modified or eliminated completely without affecting the remainder of the system.

2.2.4 - HARDWARE RESTRICTIONS - Since this project was personally funded it was implemented on the hardware already owned by the author:

a. Radio Shack TRS-80 Model II Micro Computer with 64K of random access memory.

b. Novation Cat Modem - acoustically coupled FSK Modem, EIA RS-232C Interface.

c. Centronics Model 737 Line Printer - parallel interface, maximum print speed approximately 22 lines per minute.

2.2.5 SOFTWARE SELECTION - Three programming languages were available and considered for implementing this application. Figure 3 shows the major advantages and disadvantages of each. BASIC was never seriously considered because of the limitations imposed by its interpretation/execution speed. Because this system is a prototype and many changes could be expected, PASCAL was chosen over Z-80 assembly code primarily because PASCAL program code is easier to modify.

Language	Advantages	Disadvantages
Z-80 Assembly	.Fast .Efficient	.Difficult to Debug/Modify
BASIC (interpreter)	.Easy to write	.Slow .Requires 18K of RAM for interpreter .Hard to understand large programs
PASCAL	.Easy to write .Fast .Easy to understand source code	.Generates 8080 assembly code .Not available on all micros

Fig. 3. Comparison of Available Programming Languages

2.3 SPECIFIC SYSTEM REQUIREMENTS:

2.3.1 MENU DRIVEN FORMAT - The number of commands a user has to memorize should be limited as much as possible. This can be provided by a menu-driven system that presents a list of commands to the user and then allows him to select from that list. Figure 4 depicts the algorithm for the system's primary command menu. The user's selection transfers the control of the system to the appropriate module and then displays the menu again when the function is completed.

2.3.2 ASCII TERMINAL - To meet the minimum requirements of a terminal program the system must have the capabilities of transmitting ASCII characters from the keyboard and displaying the received characters. The algorithm in Figure 5 includes these functions plus the capabilities to send a break sequence and to store the characters handled by the terminal in a buffer for later use. The function of the system buffer is discussed further in paragraph 2.3.4.

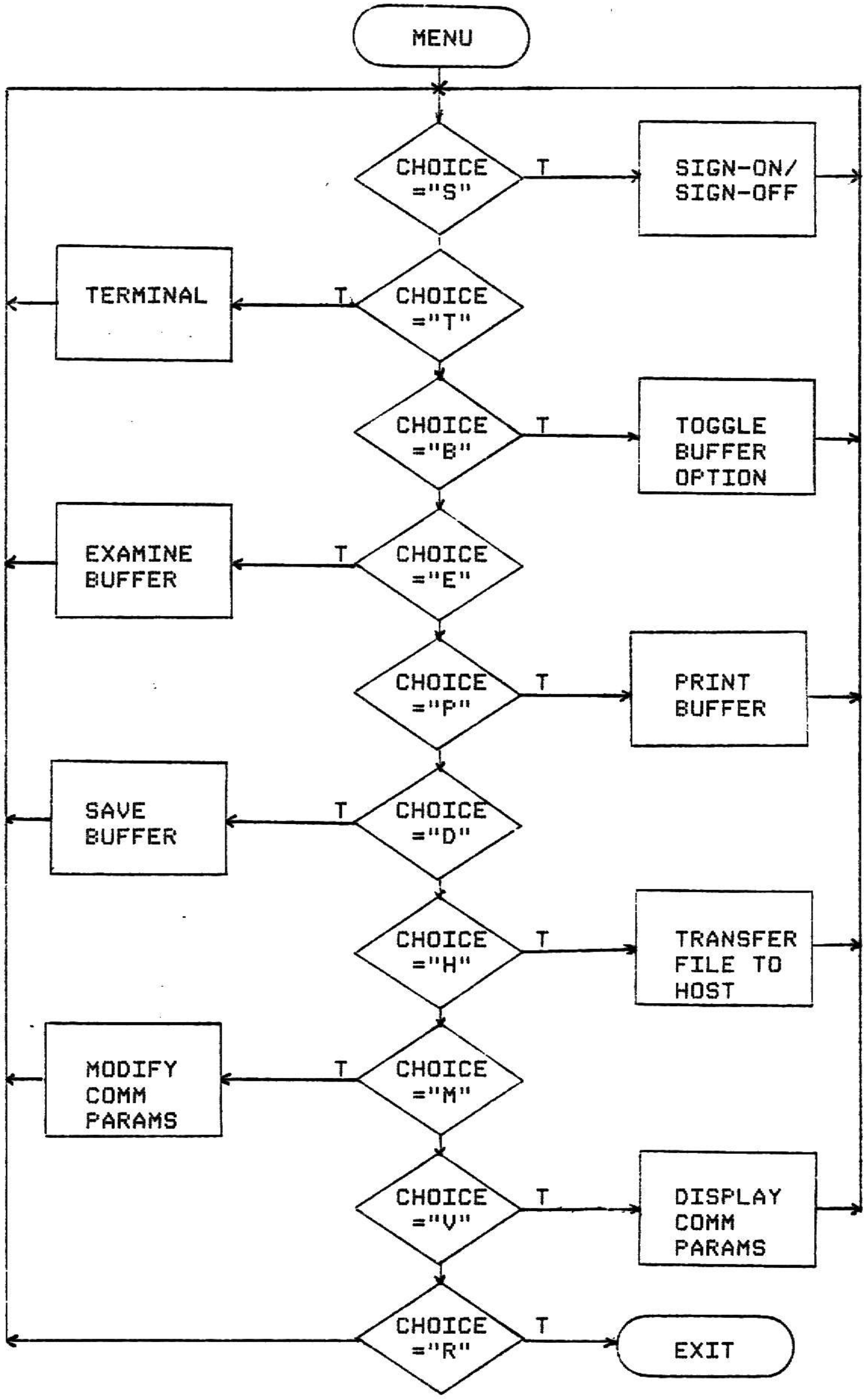

Fig. 4. System Menu

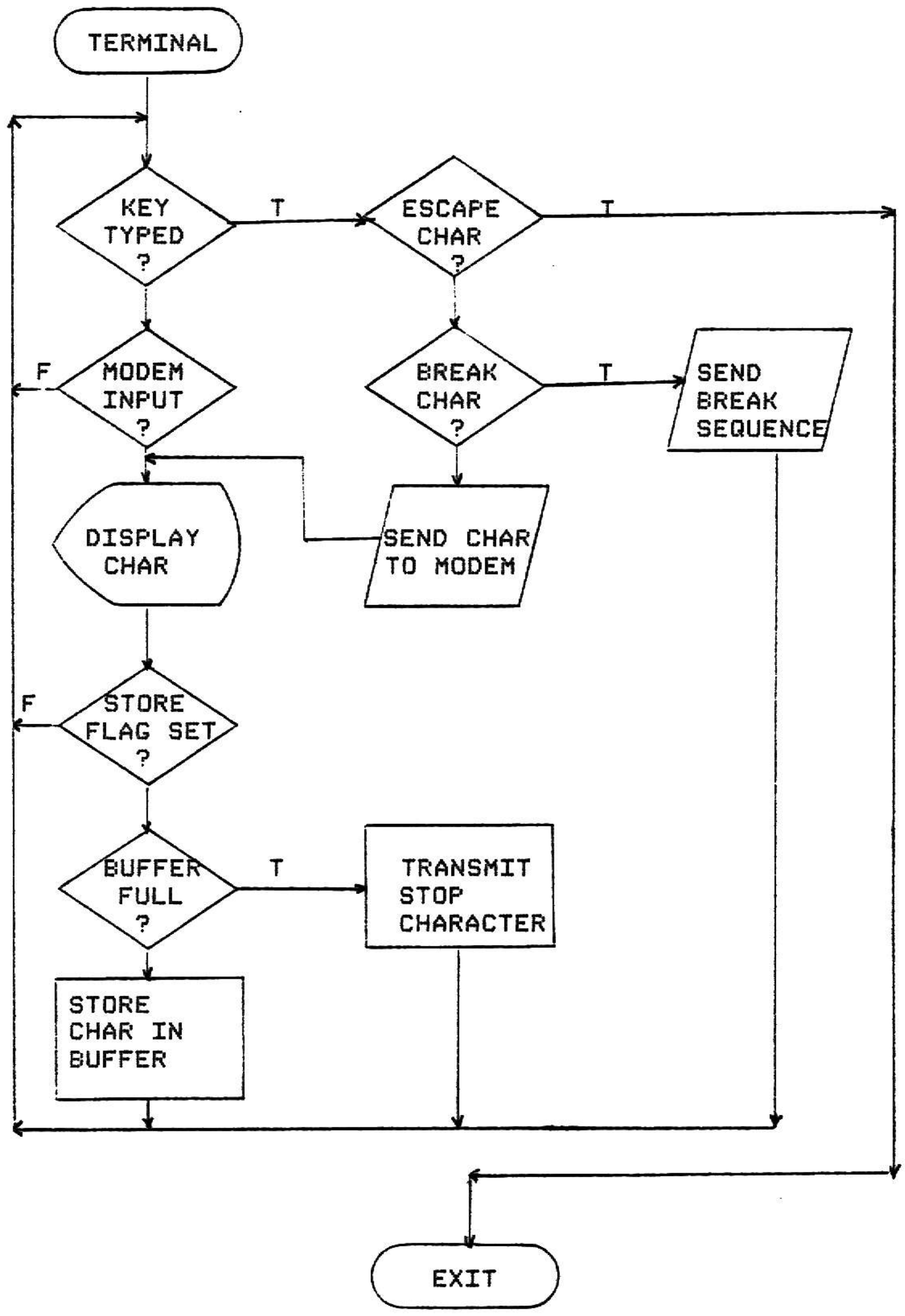

Fig. 5. Terminal Algorithm

2.3.3 FILE TRANSFER - The principal feature that allows the home computer to excel as a remote terminal is the home computer's capabilities to create, modify and store files. To make the best use of the time when a terminal is connected the user needs the capability of preparing messages before the connection is made and then transmitting them at maximum baud rate, rather than typing speed. Therefore, the most important function to design into the system is a capability to transmit the complete contents of a file created by the system editor as well as direct keyboard input (see Figure 6). This algorithm keys on the host system transmitting a cursor character to the remote terminal as an indication that the line of text transmitted has been received and the host is ready to accept the next line.

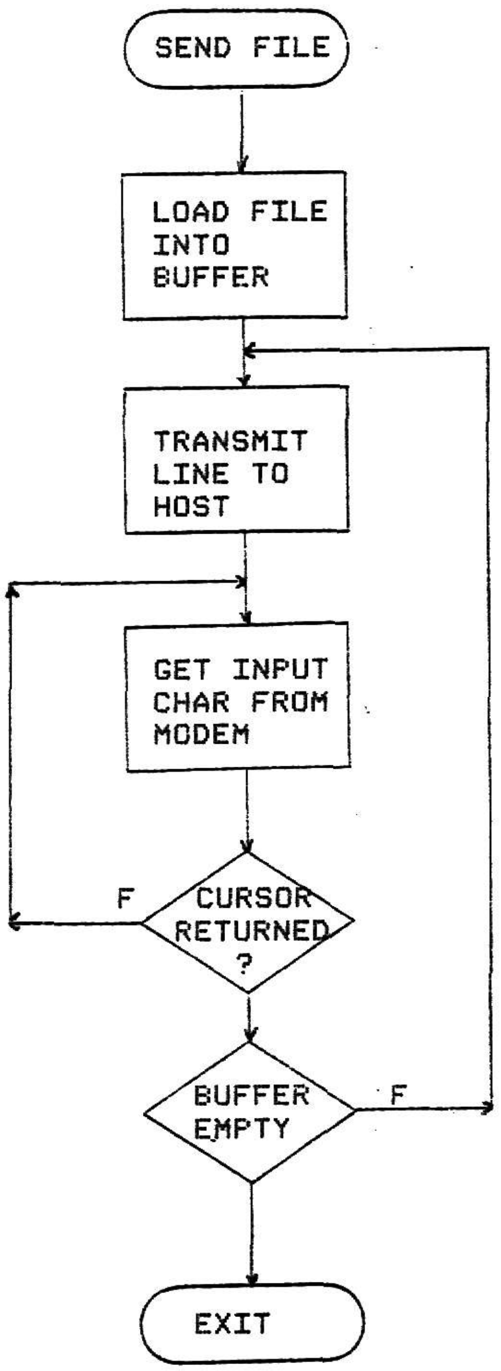

Fig. 6. File Transfer Algorithm

2.3.5 RECORD OF TRANSACTIONS - When using a video terminal the information displayed on the screen is lost when it scrolls off the screen. When the user is being charged for the time connected to the host it is inefficeint and expensive to make a record of terminal transactions by hand. Therefore, the capability to store the information and to review it later, off line, at the user's convenience is very useful. The terminal algorithm in Figure 5 includes the capability of storing characters in a system buffer for later use. Figures 7 through 9 are algorithms for examining, saving, and printing the contents of the system buffer. These functions provide the user a record of the terminal transactions and can be used to transfer a file from the host system to the micro computer by listing the program to the terminal screen and then saving the system buffer.

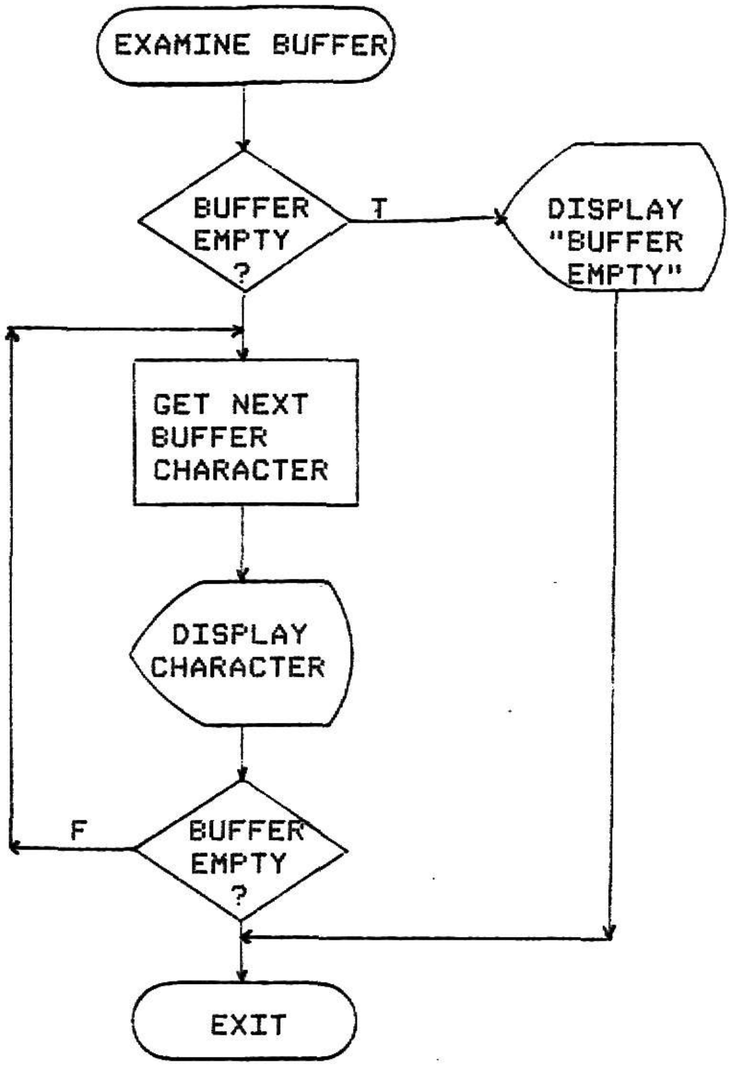

Fig. 7. Algorithm to View Contents of Buffer

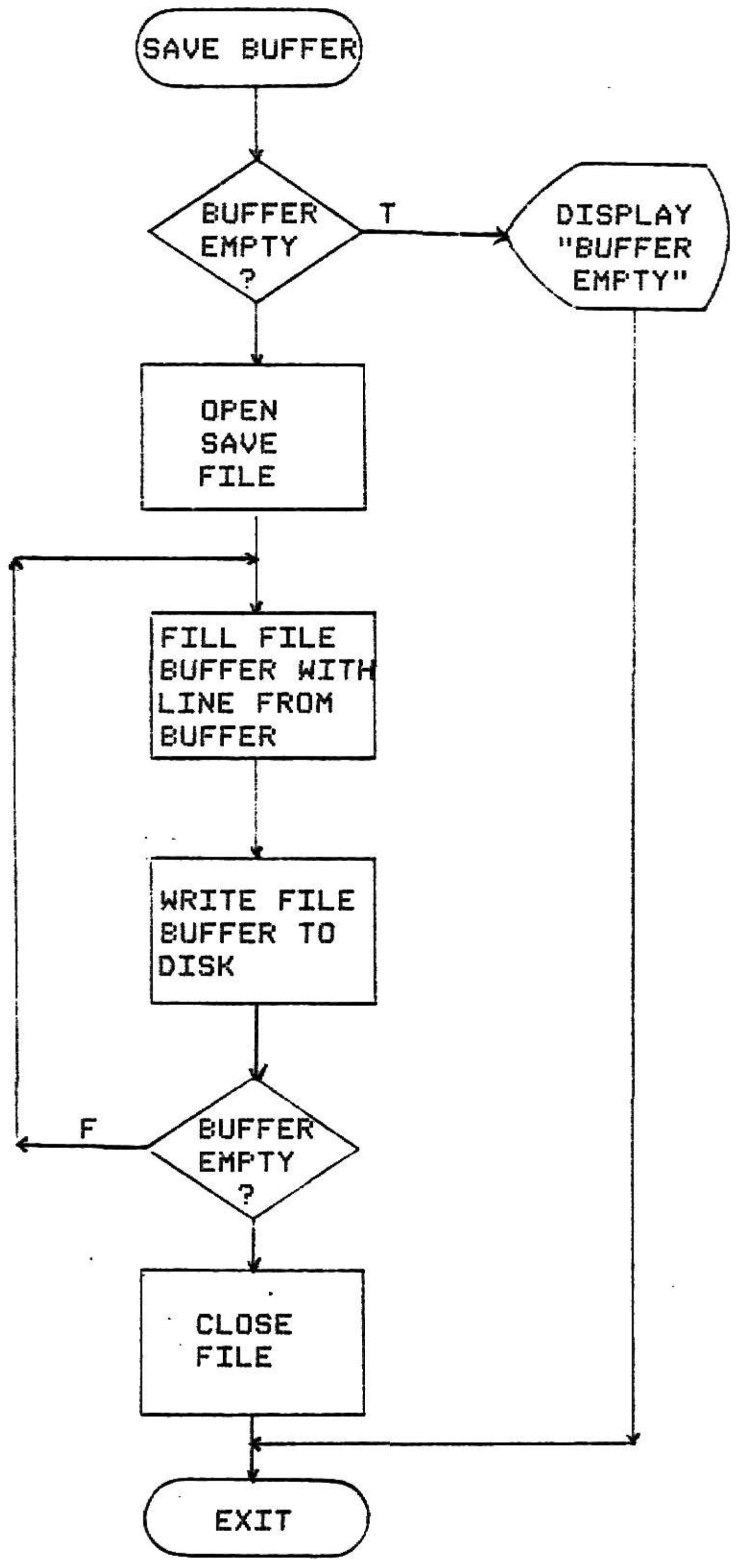

Fig. 8. Algorithm to Save Contents of Buffer

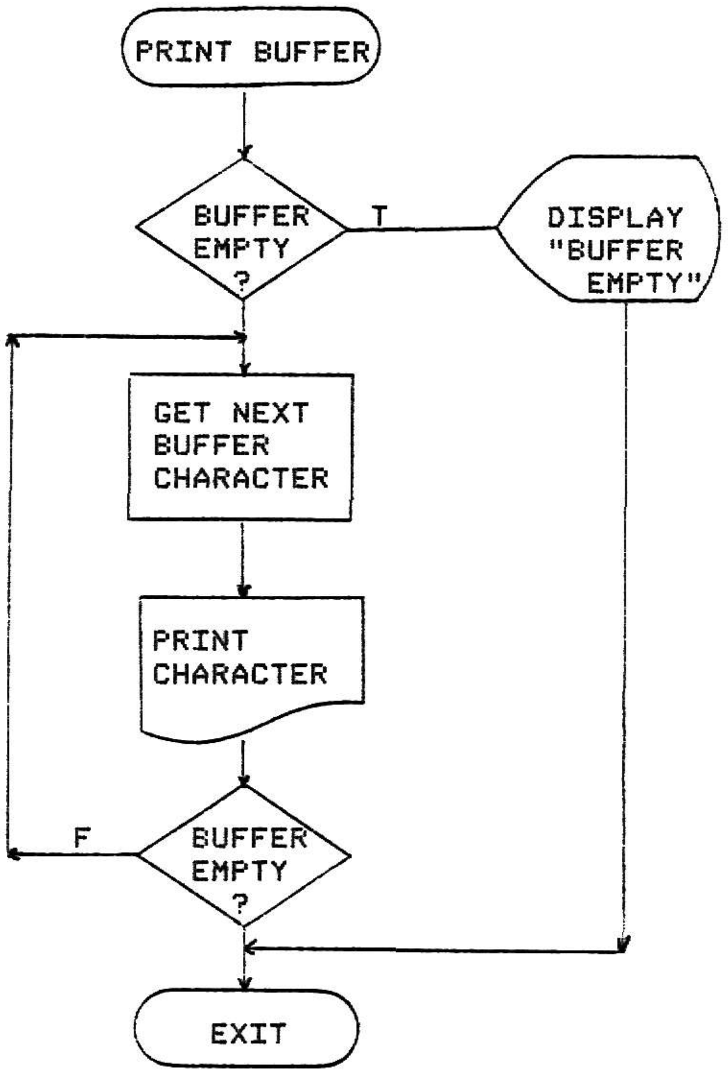

Fig. 9. Algorithm to Print Contents of Buffer

2.3.6 PROGRAM CONTROL OF COMMUNICATIONS PARAMETERS - If the system is to be used with more than one host system it must include the capability to configure the system's communications parameters to the new host. Flexibility is be built into the system by having the capability to change the more common communications parameters under program control. Figure 10 is a list of the parameters used by the system that can be modified.

Baud Rate
Parity
Word Length
Number of Stop Bits
Cursor Character
Number of Pad Characters
Sign-on Message
Sign-off Message

Fig. 10. Modifiable Communications Parameters

2.4 SUMMARY - The high level algorithms discussed in this chapter provide the essential functions of an inter-computer communications system. Chapter 3 discusses their implementaion into a prototype system.

CHAPTER 3

SYSTEM IMPLEMENTATION

3.1 INTRODUCTION - This chapter discusses the implementation and general operation of the SMART TERM system. It is the intent of this chapter to focus on the primary system operations. Where a function is trivial or the internal program documentation is sufficiently detailed, the corresponding program procedure may not be discussed. The users's manual, located in Appendix A, contains the specific operating procedures for the system. For clarity, specific program procedures and functions are depicted in uppercase characters (e.g. PROCEDURE MENU). Specific keyboard keys are shown between the "less than" and "greater than" symbols (e.g. <ESC>, <T> or <ENTER>). The actual PASCAL code for PROGRAM TERM and PROGRAM SETPARAMS is located in Appendices B and C respectively.

3.2 GENERAL ORGANIZTION - To allow for the inherent memory limitations of a microcomputer, the system was implemented as two modules/programs and a data file, see Figure 11. The primary module TERM remains resident in memory during operation of the system, executing the primary functions of the system, unless the <M> MODIFY PARAMETERS option is selected. If the modify option is selected, control is chained to the second system module SETPARAMS. The data file (TERM.DAT) is used to save the user defined communications parameters and to pass the current value of the parameters between the modules. Each of these three system components will be discussed in greater detail.

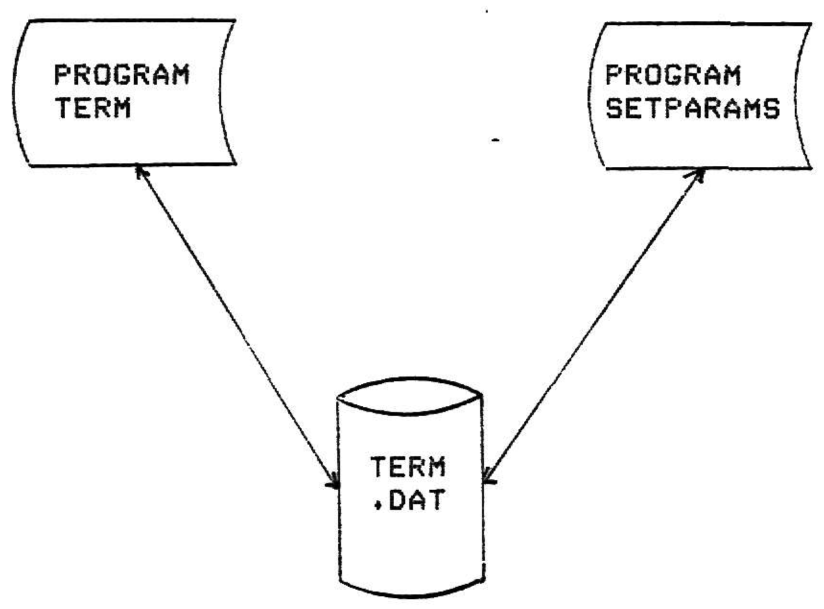

Fig. 11. General Organization

3.3 PROGRAM TERM - This module is composed of four primary functions: the menu of selections, the terminal, the program buffer (and those operations performed on it), and the programming of the serial I/O controller. The principal procedures that are used to perform these functions are discussed in the following paragraphs.

3.3.1 PROCEDURE MENU - This routine directs the flow of control within the system. It displays a list of system functions to the user (see Figure 12) and then waits for a character to be typed. Once input is detected by FUNCTION KEY_PRESSED the keyboard is read by PROCEDURE READ_KEYBOARD and control is passed to the procedure selected. If the input was not a valid choice the menu is redrawn and the user is prompted for another input.

```
SELECT KEY                 OPTION
==========                 ======

    <S>                <SIGN-ON/SIGN-OFF>

    <T>                <TERMINAL>

    <B>                <RECEIVE TO BUFFER>

    <E>                <EXAMINE BUFFER>

    <P>                <PRINT BUFFER>

    <D>                <SAVE BUFFER TO DISK>

    <H>                <TRANSFER FILE TO HOST>

    <M>                <MODIFY PARAMETERS>

    <V>                <VERIFY PARAMETERS>


    <R>                <RETURN TO SYSTEM LEVEL>
```

Fig. 12. Menu Display

3.3.2 PROCEDURE TERMINAL - This procedure performs those functions normally provided by any "standard" terminal. It is essentially a loop that polls the keyboard and the serial I/O input port and, if a character is waiting, transmits or displays the character, respectively. Accomplishing this "poll and handle" sequence for the serial port is straightforward. The version of PASCAL used to implement this system allows the programmer to perform direct port reads through the "INP" function. FUNCTION MODEM_INPUT uses INP to read the serial I/O status port and then tests bit 0 of the byte returned. When the value of the bit is 1 there is an input data byte ready to be read. Reading the serial I/O data port returns the input character to be displayed.

Due to the TRS-80 Model II architecture a slightly more complex method had to be used to handle keyboard input. The Model II keyboard has a separate processor for keyboard control. When a key is typed the keyboard controller generates an interrupt to inform the operating system that the keyboard has data waiting to be handled. With this strategy there is no hardware status port to poll. While investigating methods of writing an interrupt handling routine that could be written within the PASCAL program, it was discovered that the CP/M operating system uses an internal polling strategy to handle input from the keyboard and this routine is accessible to a programmer. CP/M includes a jump table of operating system I/O routines as part of its Basic Input Output System (BIOS). To use these routines the programmer loads the CPU registers with the appropriate values and then makes a call to the location of the desired routine within the jump table. The compiler's INLINE function is used to load the CPU registers. The INLINE function allows a programmer to enter either assembly code, or a hexidecimal representation of machine code, into a PASCAL program. The program FUNCTION KEYPRESS, and PROCEDURE READ_KEYBOARD both make calls on the operating system BIOS routines to perform their functions. FUNCTION KEYPRESS checks the status of the keyboard for a character waiting. If KEYPRESS is true PROCEDURE READ_KEYBOARD reads the character from the keyboard.

PROCEDURE TERMINAL handles keyboard input as being one of three types. If an <ESC> key is typed the program returns to the MENU routine. If the <F1> key is typed a break sequence is transmitted by PROCEDURE SEND_BREAK. All other keyboard input is transmitted to the serial I/O controller by PROCEDURE MODEM_OUT. MODEM_OUT loops until the SIO controller is ready to accept the character and then passes the character to controller for transmission.

3.3.3 THE SYSTEM BUFFER - The system buffer is the primary data structure (an array of characters) for the system. PROGRAM TERM includes procedures that provide the user the capabilities to examine the current contents of the buffer, direct the contents to the printer, save the contents in a Model II disk file, or transfer a Model II disk file to a host using the program buffer as a temporary holding area.

3.3.3.1 PROCEDURE EXAMINE_BUFFER - This procedure displays the contents of the buffer on the Model II video display. The Model II keyboard includes a <HOLD> key that can be used to stop the display for viewing if the output scrolls too fast for viewing.

3.3.3.2 PROCEDURE PRINT_BUFFER - As implemented, this procedure is similar to PROCEDURE EXAMINE_BUFFER except that it directs the contents of the buffer to the printer instead of displaying it on the screen. This procedure was originally intended to allow the system to simulate a teletype, with each character being echoed to the printer as it is displayed on the screen. However, due to the operational characteristic of the printer firmware this was impractical. The Centronics 737-1 is a dot-matrix printer capable of printing 80 characters per second (cps). It accepts data at up to 2,200 cps and stores the input characters in a buffer within the printer. A line of data is not printed until a carriage return (CR) code is received by the printer or 80 characters are counted by the printer logic. While the printer is busy printing the contents of its buffer, it cannot accept new characters from the program, therefore, any characters that are received from the modem are lost.

3.3.3.3 PROCEDURE SAVE_BUFFER - This procedure writes the contents of the SMART TERM system buffer to a Model II disk file. This feature can be used to provide a record of terminal transactions or, when used in conjunction with other system options, to transfer a file from the host to the Model II. To transfer a file from the host, the file is first loaded into the SMART TERM system buffer. If the <R> RECEIVE TO BUFFER option has been previously selected, all terminal transactions are written to the buffer. Therefore any host system command that causes the program to be displayed on a remote terminal will cause the program to be loaded into the system buffer. If during a terminal session the number of characters in the buffer exceeds the maximum buffer size, a stop code (Control S) is sent to the host and a "BUFFER FULL" message is displayed to the user. Once the program is in the system buffer the user can then save the buffer contents to the Model II disk with the <D> SAVE BUFFER TO DISK command.

Since most line editor systems list a reference line number with each line, there is a feature within this procedure that allows the user to strip the line numbers from the file. As these line numbers may be part of the program, as in BASIC, or may only be for editor reference the user is given the option of stripping the line numbers before the file is saved. The line number is considered to include the line number digits and the blank spaces between the last digit and the column where the first character of a line may begin. The first algorithm implemented discarded the line number and all spaces until the first character of the line was found. This provided the desired stripping action but also removed all indentation of program lines. With the current algorithm, if the strip option is chosen, the user is prompted to supply the number of characters that the host's editor places between the line number and the first character of a line. The first character of each line written to the disk is then determined by discarding the line number and then skipping the given number of blank characters.

3.3.3.4 PROCEDURE TRANSFER FILE - This procedure is used to transfer a copy of a Model II disk file to the host system. To transfer a file the program is loaded into the program buffer and then transmitted a character at a time to the host. If the program is larger than the maximum buffer size one buffer is transmitted and the buffer is refilled. This cycle is repeated until the complete file has been transferred. Transferring a program to the host system is relatively easy on systems that have the capability to handle paper tape, since these systems are prepared to accept long strings of data from a paper tape reader. On these systems transferring a file is simply sending a stream of characters out the serial I/O port and letting the host system build the file. However, systems that are line oriented expect to receive a line of code from a remote terminal similar to punched card input. The text editor of the Kansas State University Computer Science Department's Interdata 8/32 expects to receive a string of fewer than 80 characters, followed by a carriage return character. After handling the line of code the host system's text editor returns a cursor character to the user terminal to indicate that it is prepared to accept the next line. To transfer a file to a host system PROCEDURE TRANSFER_FILE simulates a user entering text through the host system text editor. It transmits characters from the buffer until either a carriage return character is encountered, indicating the normal end of line has been reached, or until the number of characters tranmitted equals the maximum length of an editor line (which is declared as a global constant). When one of these conditions is met a carriage return is transmitted to the host and the host system text editor is allowed to accept the line. While the host is processing the line the SMART

TERM program loops awaiting the user defined cursor to be read from the modem input. For example, after a line of code is accepted the text editor of the Interdata 8/32 transmits the following hexidecimal string FF(pad), 0D(CR), 0A(LF), FF(pad), FF(pad), 2D("-"), FF(pad), 3E(">"), FF(pad), FF(pad) to the remote terminal to end the line and display the cursor "->". PROCEDURE WAIT_FOR_HOST ignores the characters until it receives the user defined cursor character and number of pad characters. Once this sequence of characters is received, indicating the line has been accepted, the next line is transmitted and this procees continues until the buffer is empty.

3.3.4 SERIAL I/O CONTROLLER PROGRAMMING - The Model II uses a Zilog Z-80 Serial I/O (SIO) Controller to provide serial-to-parallel, parallel-to-serial conversion. This device performs all the functions traditionally done by a Univeral Asynchronous Receiver Transmitter (UART) plus additional functions normally performed by the CPU. By using the INLINE function and accessing the CP/M jump table as previously discussed in paragraph 3.3.2, PROCEDURE PROGRAM_SIO changes the functions of the SIO controller from within the TERM program. PROGRAM SETPARAMS builds three bit patterns from the user defined communications parameters and these bit patterns are later used as input for the PROGRAM_SIO routine. PROCEDURE SEND_BREAK generates a "break" by causing the modem to send approximatly 200-450 ms of space tone.

3.4 PROGRAM SETPARAMS: This module consists of a series of questions (Figure 13) that allow the user to define the communications parameters and the sign-on and sign-off messages. Each question displays the current value of the parameter (shown in boldface in the figure) and, if there is a restricted set of values, the values that the system will accept. After the question/answer sequence is completed the new parameters and the corresponding bit patterns are written to file TERM.DAT and control is returned to PROGRAM TERM.

```
(1).    CURRENT BAUD RATE IS 300
        ENTER NEW RATE [1200,600,300,110]
              OR PRESS <ENTER> TO CONTINUE


(2).    CURRENT PARITY IS EVEN
        ENTER NEW VALUE [ODD,EVEN,NONE]
              OR PRESS <ENTER> TO CONTINUE


(3).    CURRENT WORD LENGTH IS 7 BITS
        ENTER NEW VALUE [5,6,7,8]
              OR PRESS <ENTER> TO CONTINUE


(4).    CURRENT NUMBER OF STOP BITS IS 1
        ENTER NEW VALUE [1,2]
              OR PRESS <ENTER> TO CONTINUE


(5).    CURRENT CURSOR IS >
        ENTER NEW VALUE [>,*,-,.]
              OR PRESS <ENTER> TO CONTINUE


(6).    CURRENT NUMBER OF PAD CHARACTERS IS 2
        ENTER NEW VALUE [1..9]
              OR PRESS <ENTER> TO CONTINUE


(7).    CURRENT SIGNON MESSAGE IS
        SIGNON
        ENTER A NEW STRING OF UP TO 30 CHARACTERS
              OR PRESS <ENTER> TO CONTINUE


(8).    CURRENT SIGNOFF MESSAGE IS
        SIGNOFF
        ENTER A NEW STRING OF UP TO 30 CHARACTERS
              OR PRESS <ENTER> TO CONTINUE
```

Fig. 13. Setparams Question/Answer Sequence

The user defined parameters are normally loaded from file "TERM.DAT" during system initialization by TERM PROCEDURE INITIALIZE. If for some reason this file cannot be read, the procedure uses the default values (Figure 14) that are defined internally.

DEFAULT COMMUNICATIONS PARAMETERS

PARAMETER	VALUE
BAUD RATE	300
WORD LENGTH	7 BITS
PARITY	EVEN
STOP BITS	1
SIGN-ON MESSAGE	SIGNON
SIGN-OFF MESSAGE	SIGNOFF
CURSOR CHARACTER	>
PAD CHARACTERS	2

Fig. 14. Default Communications Parameters

These values were chosen because they are commonly used by other systems (e.g. COMPUSERVE, Micronet, etc.). This feature was intended to allow for errors when reading the data file but may also be used to return the system to a known state by erasing the data file and allowing the defaults to be used.

3.5 DATA FILE "TERM.DAT" - This data file consists of one record of the structure shown in Figure 15. It contains the last set of user defined communications parameters. Whenever control is passed between the two system modules the current communications parameters are written to file "TERM.DAT" by the calling program and then read by the module accepting control. The data elements are primarily text or integer values that are displayed by PROCEDURE DISPLAY_PARAMETERS (a copy is in each system module). The fields C_PATTERN, D_PATTERN, AND E_PATTERN are bit patterns assembled by the program SETPARAMS and are used to program the serial I/O controller.

PARAMETER RECORD

Element	Type
Signon Message	Packed Array[1..Max_Message_Length] of Char
Signoff Message	Packed Array[1..Max_Message_Length] of Char
Active Flag	Boolean
Parity Parameter	String[4]
Baud Parameter	String[4]
Word Length	Char
Stop Bits	Char
C_Pattern	Integer
D_pattern	Integer
E_pattern	Integer
Cursor Character	Char
Pad Character	Integer

Fig. 15 Parameter Record Elements

CHAPTER 4

ENHANCEMENTS

4.1 GENERAL - This project has succeeded in identifying and implementing a minimum set of functions that allows a home computer owner to make effective use of his computer's capabilities when it is used as a terminal to access a remote host system. It has been tested on four separate host systems and has performed well with only minor discrepancies. The operation of the system during the past several months has identified several features that would increase the usefulness of the system if added in the future.

4.2 SUGGESTED ENHANCEMENTS:

4.2.1 Now that the system is more clearly specified and sized the method of passing parameters between the two programs of the system should be refined. Each time control is passed between PROGRAM TERM and PROGRAM SETPARAM the current values of the communications parameters are first written to the data file TERM.DAT by the calling program and then read by the called program. This method works well but requires four separate accesses of the data file during the modification sequence. If the global variables of each module were declared exactly the same and both programs were loaded at the same address, it would be possible to use a common data area that could be shared by the two programs and thereby eliminate the requirement for TERM.DAT and the data file I/O.

4.2.2 The file transfer algorithm should be modified to include the capability to handle errors. When transferring a file to a host computer the current algorithm keys on the return of the host systems's editor cursor as an indication that the editor is prepared to accept the next line of code. If an error occurs, the last character of the error message returned by the host is also a cursor character and will key the transmission of the next line. The enhanced version should inspect the complete string of returned characters and halt the transfer operation if an error message is returned.

4.2.3 Experienced users should have the capability to traverse the system without being required to select an option from the menu. The intent of requiring the user to select options from a menu is to minimize the number of commands that a new user must memorize in order to operate the system. However, after a short period of time operating the system, returning to the menu to make a selection becomes an irritant rather than an assistance.

The next version should permit the experienced user to enter designated codes to circumvent the menu.

4.2.4 A "Help" option should be provided to assist the user in operating the system. Users manuals always seem to be misplaced or sometimes it is difficult to find the appropriate explanation of a function. An enhanced version of this system should provide an online reference for the user. Entering "Help" and the function select key (e.g. HELP <P>) would return a display providing an expanded explanation of that function.

4.2.5 The system could be further automated by adding an auto-dial modem (models are now available in the $200-$350 range). This addition would allow the personal computer owner to take advantage of lower toll rates or decreased host usage during late hours. A routine could be implemented to dial the telephone number of the host system at a designated time, send the sign-on message when the host computer's modem tone is detected, and then transfer a file without any intervention by the personal computer owner.

4.2.6 A timer option should be added to prevent a user from "timing out". Many systems disconnect any remote terminal that has not initiated a transaction within an alloted period of time. If the personal computer system kept track of the time since the last transaction it could warn the user when his time was running out. The function could even be extended to transmit a "dummy" transaction to the host to reset the timeout clock and not require any action by the operator.

4.2.7 The capability to transfer data or machine code files should be added. The system was specifically limited to the ASCII character set and selected ASCII control codes (no byte values greater than 127). If data or the hexidecimal representation of machine code is transmitted any bit pattern might appear in the byte being transferred. An option should be added specifically for data and machine code that would transfer the values without displaying them on the screen.

SELECTED BIBLIOGRAPHY

Fernandez J., and Ashley R. 8080/8085 Assembly Language Programming. NEW YORK: John Wiley & Sons, Inc., 1981.

Grogono, Programming in PASCAL, Reading: Addison-Wesley Publishing Company, Inc., 1980.

Hogan, Thom, Osborne CP/M User Guide, Berkeley: Osborne/McGraw-Hill, 1981.

Hunt, Daniel S., " The General ...", Interface Age, Oct 1981 pp. 104-107.

Jensen, Kathleen and Wirth, Niklaus, PASCAL User Manual and Report, second edition, New York: Spronger-Verlag, 1978.

PASCAL/MT+, Release 5, Language Reference and Applications Guide, Cardiff-by-the-Sea:MT Microsystems, 1980.

P&T CP/M for the TRS-80 Model II, Users Manual, Goleta: Pickels and Trout, 1980.

TRS-80 Model II, Technical Reference Manual, Fort Worth: Radio Shack, 1980.

Z-80 SIO Serial Input/Output Controller, Product Specification, Zilog C8038-0111 C8038-0120, Feb 1980.

APPENDIX A

SMART TERM USER'S MANUAL

SMART TERM USERS MANUAL

1. INTRODUCTION: Smart Term is a general purpose communications program for The Radio Shack TRS-80 Model II Micro Computer. It is designed to enhance the Model II's capabilities for transmission and reception of ASCII text by providing many convenient features not available in the typical "dumb" terminal system.

2. MAJOR FEATURES:

2.1 One-key transmission of user defined "sign-on" and "sign-off" messages.

2.2 File Transfer from a host system to the Model II disk or from the Model II to a host.

2.3 Program selectable communications parameters.

2.4 Optional printed output of all terminal session transactions.

3. HARDWARE REQUIREMENTS:

3.1 Radio Shack Model II Micro Computer w/64k of memory.

3.2 Telephone Modem

a. EIA-RS232C standard w/host system compatable features.

b. Baud rate capability of 110, 300, 600 or 1200 baud (300 baud recommended).

3.3 Parallel interface line printer (optional).

4. OPERATION OF THE SYSTEM:

Notation: Commands to the system may be either words or single keys. Word commands are represented in capital letters, (e.g. COMMAND), and should be typed exactly as shown. If a particular key is to be typed it will be shown between the "less than" and "greater than" symbols (e,g. <T> or <ENTER>). A combination of these notations may be used together (e.g. TERM <ENTER>) indicating the four characters "T", "E", "R", "M" should be typed then the "enter" key typed.

4.1 Determine the following communications parameters as they apply to the host system:

a. Baud rate.

b. Word length.

c. Parity.

d. Number of stop bits.

e. Cursor character used by the host system's text editor.

f. Number of "pad characters" transmitted after the cursor for timing/delay purposes. (If this value cannot be determined, assume 0 pad characters. If later, during file transfer to the Model II, extraneous characters are printed after the line number, use the number of extraneous characters printed for the number of pad characters in the <M> MODIFIY PARAMETERS sequence).

4.2 Begin execution of the Smart Term program by typing - TERM <ENTER>. The current communications parameters will then be displayed as shown in Figure 1.

```
                          CURRENT PARAMETERS
                          ==================

 TERMINAL STATUS                              SIGNED OFF

 BUFFER OPTION                                OFF

     20000 CHARACTERS OF BUFFER SPACE REMAINING

 BAUD RATE         300           PARITY          EVEN

 WORD LENGTH       7 BITS        STOP BITS       1

 HOST CURSOR       >              PAD CHARACTERS  2

 SIGNON MESSAGE    SIGNON

 SIGNOFF MESSAGE   SIGNOFF

         PRESS <ENTER> TO CONTINUE
```

Fig. 1. Parameter Display

Compare the parameters listed on the screen to those you obtained for the host system and make note of any differences. Press <ENTER> to proceed. After a brief pause the screen will fill with a menu of selections (Figure 2). If there were no discrepancies between the values of the communications parameters obtained for the host and those values displayed as the current communications parameters, you are ready to begin operation. If there were, execute the parameter modification sequence, <M>, before attempting any of the other options.

SELECT KEY	OPTION
<S>	<SIGN-ON/SIGN-OFF>
<T>	<TERMINAL>
<B>	<RECEIVE TO BUFFER>
<E>	<EXAMINE BUFFER>
<P>	<PRINT BUFFER>
<D>	<SAVE BUFFER TO DISK>
<H>	<TRANSFER FILE TO HOST>
<M>	<MODIFY PARAMETERS>
<V>	<VERIFY PARAMETERS>
<R>	<RETURN TO SYSTEM LEVEL>

Fig. 2. Menu of Options

5. MODES OF OPERATION: This section discusses the system functions that are available to the user. Each function is discussed in the order it is displayed on the system MENU. Figure 3 shows the inputs (characters) required to move from one function to another. Referring to the figure may assist the new user in following the flow of the system while reading through this section.

Section 6, SAMPLE SESSIONS, shows how these independent functions may be combined to perform the more sophisticated features of the system.

5.1. MENU - The menu, Figure 2, lists the primary functions of the system and allows the user to select an operation by typing the appropriate "select key". After completion of all functions, except SIGNON/SIGNOFF, the system automatically returns to the MENU mode to allow selection of other functions.

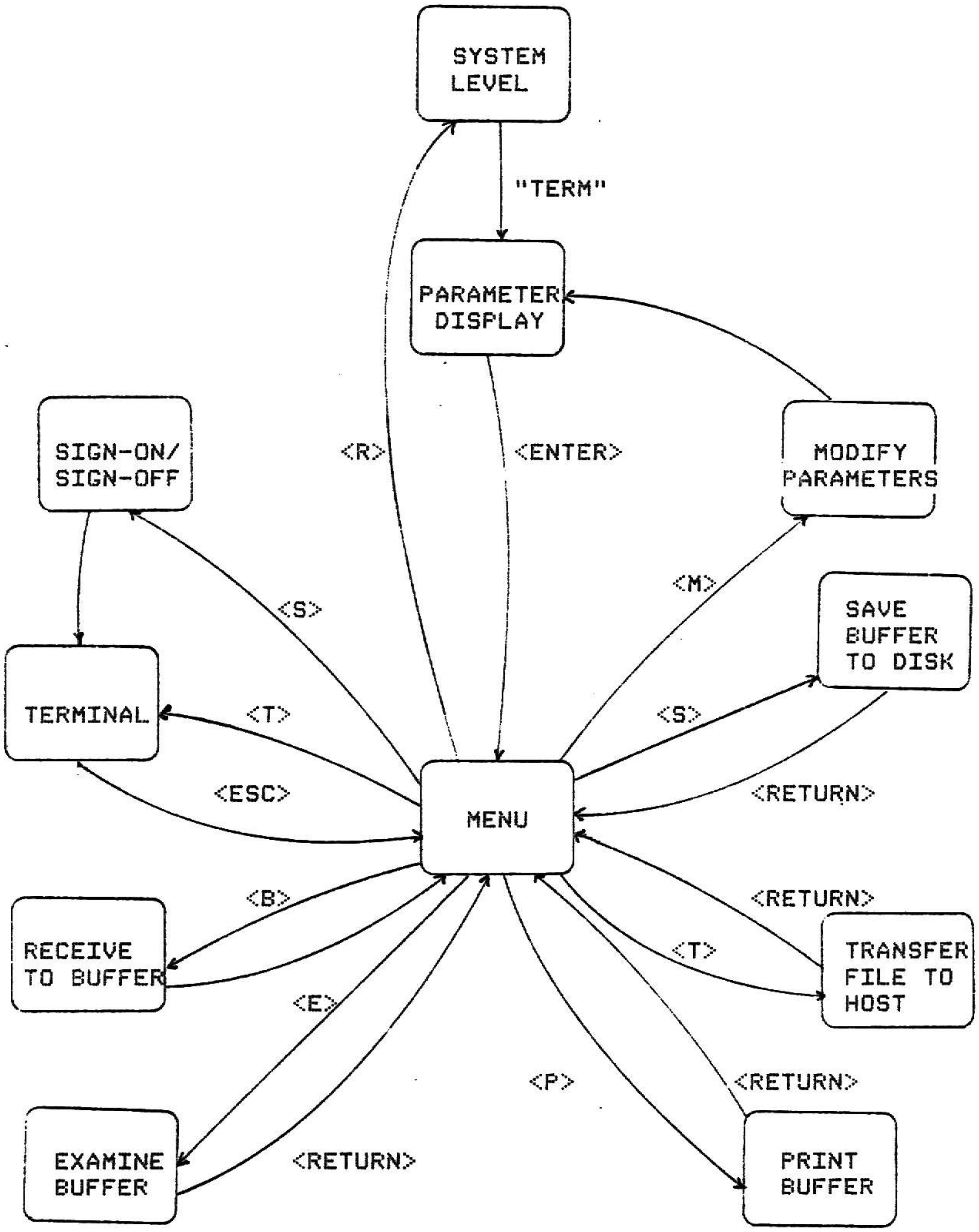

Fig. 3. Inputs Required to Move Between Functions.

5.2 <S> SIGN-ON/SIGN-OFF - This option allows the user to transmit either the default sign-on or sign-off messages, or a user defined message that has been entered with the <M> option, to the host computer system. The sign-on and sign-off messages alternate with each selection of this option starting with the sign-on message. After the message is transmitted, Smart Term automatically enters the TERMINAL mode to display any response from the host.

5.3 <T> TERMINAL - In the terminal mode the Model II acts as a standard communications terminal. All information transmitted by the host is displayed on the screen and any keyboard entry by the user is transmitted to the host. Due to peculiarities in the Model II keyboard, the <BREAK> key transmits a "Control C" character to the host; therefore the <F1> special function key on the numeric keypad has been programmed to transmit the break sequence. You may return to the MENU mode by typing the <ESC> key.

5.4 <B> RECEIVE TO BUFFER - This selection allows all transactions that take place in the terminal mode to be saved in the system buffer, a temporary storage area, for viewing with the <E> option, for printing with the <P> option, or to be saved to disk with the <D> option. This option may be "turned off" by typing the <B> while the Menu is displayed, and no further information will be saved. You may determine the amount of empty space remaining in the system buffer by selecting the <V> VERIFY PARAMETERS option from the menu. CAUTION #1. When the <R> RECEIVE TO BUFFER option is set to "ON" it clears the current contents of the buffer. Therefore if you want to save the contents of the buffer you must use the <P> or <D> options before resetting the buffer option to "ON". CAUTION #2. If while the buffer option is "ON" the number of characters input exceeds the maximum buffer size, a stop code will be sent to the host and a "BUFFER FULL" message will be displayed. The contents of the buffer must be handled (saved or discarded) before you may continue. If you were transferring a program to the Model II you must save the contents of the buffer currently received and then transfer the remainder of the program. Since there may be a delay between the time the stop code is sent and the host system actually stops transmitting characters, it may be necessary to have the host retransmit the last several lines of the program received before the "BUFFER FULL" condition was reached.

5.5 <E> EXAMINE BUFFER - This selection allows you to view the contents of the system buffer and then returns to the menu. If the contents scroll too fast for viewing you may stop the display by typing the <HOLD> key. Pressing the <HOLD> key again will restart the display.

5.6 <P> PRINT BUFFER - This option first prompts the user to insure the printer is ready and then prints the contents of the system buffer.

5.7 <D> SAVE BUFFER TO DISK - This selection allows you to save the contents of the system buffer in a Model II disk file. This option can be used to create a permanent record of a terminal session or of a file that has been listed to the terminal. (See Sample Session #1).

5.8 <H> TRANSFER FILE TO HOST - This option is used to send a copy of a program or text file to the host system. The user is prompted for the name of the file and the file is transmitted line by line to the host. (See Sample Session #2).

5.9 <M> MODIFY PARAMETERS - During this sequence the program will request various inputs from you in order to prepare the Model II to properly communicate with the host system. A list of valid responses will be displayed with each question. One of these responses must be entered or the system will repeat the question. If the current parameter displayed matches the host system parameter you may simply type the <ENTER> key to move to the next question. The last two questions of this sequence allow you to change the sign-on and sign-off messages that may be sent by the <S> SIGN-ON/SIGN-OFF option to the host. Each message may consist of a string of up to 30 characters. If you desire to change either of these strings simply type in the message you desire and type <ENTER>. If you attempt to exceed 30 characters the system will accept the first 30 characters and continue. You will have to perform the modification sequence again if you need to correct the message. After you have completed this sequence the system will save the parameters you have entered for future use, reprogram the Model II serial I/O controller and then display the new values. (see Sample Session #3).

5.10 <V> VERIFY PARAMETERS - This option displays the current values of the system parameters that can be modified by the user and the unused space remaining in the system buffer (see Figure 1).

5.11 <R> RETURN TO SYSTEM LEVEL - Selecting this option returns you to the Model II operating system where you may perform program editing, other processing, or terminate the session. If the sign-on message option was used during this session and the sign-off message was not, the message shown in Figure 4 is displayed to remind the user that he may still be signed-on.

```
YOU ARE STILL SIGNED ON....
         DO YOU WANT TO SIGN OFF - ENTER <Y>ES OR <N>O

    IF YOU ANSWER YES THE PROGRAM WILL TRANSMIT
        THE SIGNOFF MESSAGE.  HOWEVER, IT IS YOUR
         RESPONSIBILITY TO INSURE YOU ARE CURRENTLY AT A
         LEGITIMATE LEVEL WITHIN THE HOST SYSTEM TO ISSUE
          THIS COMMAND.
```

Fig. 4. Sign-off Reminder

Since many host systems will recognize the sign-off message only when it is sent at a specific level within the host program, the user is reminded of this fact.

6. SAMPLE SESSIONS: The following sample sessions demonstrate how a series of commands might be used to perform a typical terminal session. (These samples assume the user starts in the MENU mode)

6.1 SAMPLE #1. File Transfer from the host system to the Model II. If the user would like to transfer a file from the host system to the Model II disk for storage or editing he would perform the following sequence:

a. Enter the terminal mode by typing <T>. Access a level within the host system that will allow you to list the file to the terminal screen (e.g. a "list" or "print lines" command in the host system editor).

b. Return to the menu by typing <ESC>.

c. Press the <B> key to turn on the RECEIVE TO BUFFER option. If the buffer option was already active, typing the <B> twice will clear the buffer.

d. Return to the terminal mode by typing the <T>.

e. Enter the host system command that will list the host program to the screen. Each line will be displayed as it is received and will also be stored in the memory buffer.

f. When the last line has been received type <ESC> to return to the MENU.

h. Press <D> to begin the Save to Disk function and answer the "OUTPUT FILENAME QUESTION?.." with a valid file name.

i. The system will then offer the option "STRIP LINE NUMBERS - <Y>ES OR <N>O ???". If "<N>o" is selected the contents of the buffer will be saved exactly as they exists. If "<Y>es" is selected the system will then ask, "HOW MANY SPACES BETWEEN LINE NUMBER AND FIRST CHARACTER ???". If the host system inserts spaces between the line number and the first character of the line, enter that number of spaces (e.g. 5). If no spaces are inserted by the host type "0 <ENTER>" or <ENTER>. The system will then strip the line number and the number of spaces just entered from each line before saving it to the disk. After the buffer has been saved to disk, the user is returned to the MENU and may select the next function.

6.2 SAMPLE #2. Transfer file from Model II to a host computer. If the user would like to transfer a file from the Model II to a host system he would perform the following sequence:

a. Enter the terminal mode by typing <T> and access a level within the host system that would allow you to create a source file (e.g. the "create mode" of the host's editor system).

b. Return to the MENU by typing the <ESC> key.

c. Press the <H> key to select the TRANSFER FILE TO HOST option and answer the

"ENTER NAME OF FILE TO TRANSFER........?"

question with the name of the file you wish to transfer. When the transfer has been completed the system will return to the MENU.

d. Press <T> to return to the terminal mode and issue the appropriate commands to the host system editor to save the transferred file on the host system.

6.3 SAMPLE #3. Modification of the Communications Parameters

a. Begin the modification sequence by typing <M>.

b. The system will respond with the display shown in Figure 5.

```
                 **** W A R N I N G ****

   WHILE MODIFYING THE PARAMETERS ANY INFORMATION
   CONTAINED IN THE BUFFER WILL BE LOST AND THE
   RECEIVE TO BUFFER OPTION WILL BE TURNED OFF.

    PRESS <ENTER> TO CONTINUE
         OR <ESC> TO RETURN TO THE MENU
```

Fig. 5. Warning displayed during modification sequence.

If the buffer currently holds information you do not want discarded, return to the MENU by typing <ESC> and save or otherwise handle that information. If the buffer is empty, or its contents unimportant, type <ENTER> to continue.

c. The system will now step through the eight displays shown in Figure 6. Each display shows the current value of the parameter (Shown in boldface in the figure, reverse video on the terminal). If there is a restricted set of values, a list of entries the system will accept is shown in square brackets (e.g. [1200,600,300,110]). Type one of the values listed in the square brackets or type <ENTER> if the current value matches the host system parameter. When this sequence is completed the system will display the new parameter values, as in Figure 1, and then use these values to program the Model II Serial I/O controller.

```
(1).     CURRENT BAUD RATE IS 300
         ENTER NEW RATE [1200,600,300,110]
               OR PRESS <ENTER> TO CONTINUE


(2).     CURRENT PARITY IS EVEN
         ENTER NEW VALUE [ODD,EVEN,NONE]
               OR PRESS <ENTER> TO CONTINUE


(3).     CURRENT WORD LENGTH IS 7 BITS
         ENTER NEW VALUE [5,6,7,8]
               OR PRESS <ENTER> TO CONTINUE


(4).     CURRENT NUMBER OF STOP BITS IS 1
         ENTER NEW VALUE [1,2]
               OR PRESS <ENTER> TO CONTINUE


(5).     CURRENT CURSOR IS >
         ENTER NEW VALUE [>,*,-,.]
               OR PRESS <ENTER> TO CONTINUE


(6).     CURRENT NUMBER OF PAD CHARACTERS IS 2
         ENTER NEW VALUE [1..9]
               OR PRESS <ENTER> TO CONTINUE


(7).     CURRENT SIGNON MESSAGE IS
         SIGNON
         ENTER A NEW STRING OF UP TO 30 CHARACTERS
               OR PRESS <ENTER> TO CONTINUE


(8).     CURRENT SIGNOFF MESSAGE IS
         SIGNOFF
         ENTER A NEW STRING OF UP TO 30 CHARACTERS
               OR PRESS <ENTER> TO CONTINUE
```

Fig. 6. Parameter Modification Sequence

APPENDIX B

PROGRAM SMART_TERM SOURCE CODE

```
(**********************)
PROGRAM SMART_TERM;
(**********************)
(*==============================================================
 PROGRAM TITLE:  SMART TERM

 PROGRAM AUTHOR: DAN VESTAL

 PROGRAM FILE: TERM.PAS

 LAST UPDATE: 3 NOV 81

 PROGRAM SUMMARY:

   THIS SYSTEM PROVIDES A COMPLETE INTERCOMMUNICATIONS
   PACKAGE FOR THE RADIO SHACK TRS-80 MODEL II
   MICRO COMPUTER.  IT PROVIDES THE STANDARD TERMINAL
   FUNCTIONS OF ASCII INPUT AND OUTPUT THROUGH THE
   SERIAL PORT, TRANSFERS FILES BETWEEN THE HOST AND
   THE MODEL II, AND PROVIDES OTHER FUNCTIONS TO ASSIST
   THE USER.


==============================================================*)

CONST
  MAX_LINE_LENGTH=80;     (* LENGTH OF TRANMITTED LINES *)
  MAX_MESSAGE_LENGTH=20;  (* LENGTH OF SIGNON/OFF MESSAGE *)
  MAX_BUFFER_SIZE=20000;  (* RECEIVE BUFFER SIZE *)
  BREAK_DELAY=500;        (* LOOP COUNTER FOR CARRIER BREAK *)

  (* MODEL II MODEM PORT NUMBERS *)
  STAT_PORT=$F6; (* STATUS PORT  *)
  MOD_PORT=$F4;  (* DATA PORT    *)

  NULL_CHR=0;             (* ASCII CHARACTER CODES *)
  F1_KEY=1;
  LINE_FEED_CHAR=10;
  CLEAR_CHR=12;
  CARRIAGE_RTN_CHR=13;
  REV_VIDEO=14;
  GRAPH_CHR=17;
  CNTRL_R=18;
  CNTRL_S=19;
  ESCAPE_CHR=27;
  BLANK_CHR=32;
  DELETE_CHR=127;
```

```
TYPE
   CHAR_FILE = FILE OF CHAR;
   PARAM_RECORD =
      RECORD
         ON_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH] OF CHAR;
         OFF_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH] OF CHAR;
         ACTIV_FLAG:BOOLEAN;
         PARITY_PARAM:STRING[4]; (* THIS RECORD IS READ FROM AND WRITTEN
         BAUD_PARAM:STRING[4];   (* TO FILE TERM.DAT. TO SAVE THE CURRENT
         WORD_LNGTH:CHAR;        (* USER DEFINED PARAMETERS             *)
         STP_BITS:CHAR;
         C_PATTERN:INTEGER;
         D_PATTERN:INTEGER;
         E_PATTERN:INTEGER;
         CURSOR_CHAR:CHAR;
         PAD_CHARACTERS:INTEGER;
      END;
   PARAM_FILE = FILE OF PARAM_RECORD;

VAR
   IN_BUFFER:ARRAY[1..MAX_BUFFER_SIZE] OF CHAR;
   END_OF_BUFFER_POINTER,NR_OF_PADS:INTEGER;
   SIGNON_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH]OF CHAR;
   SIGNOFF_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH]OF CHAR;
   STORE_FLAG,ACTIVE_FLAG_SET, PRESS:BOOLEAN;
   PARITY,BAUD_RATE:STRING[4];
   PRINT_FILE:CHAR_FILE;     (* TYPE THE PRINTER *)
   ENTER_PRESS,ANSWER,HOST_CURSOR,WORD_LENGTH,STOP_BITS:CHAR;
   CHAIN_FILE:FILE;
   C_REGISTER,D_REGISTER,E_REGISTER:INTEGER;


(*******************************************)
FUNCTION UPPERCASE(IN_CHAR:CHAR):CHAR;
(*******************************************)
(* CONVERT THE INPUT CHARACTER INTO AN UPPERCASE
 CHARACTER BY STRIPPING BIT 6 WITH A LOGICAL "AND" *)

VAR
   CHAR_ORD:INTEGER;
BEGIN
   IF (IN_CHAR >= 'a') AND (IN_CHAR <= 'z') THEN
      UPPERCASE := CHR(ORD(IN_CHAR) & 95)
   ELSE
      UPPERCASE := IN_CHAR;
END;
```

```
(*************************)
PROCEDURE CLEAR_SCREEN;
(*************************)
(* CLEAR THE SCREEN AND HOME THE CURSOR *)

BEGIN
   WRITE(CHR(CLEAR_CHR));
END;

(************************************************)
PROCEDURE READ_KEYBOARD(VAR KEY_CHAR:CHAR);
(************************************************)
(* THIS ROUTINE IS WRITTEN IN 8080 ASSEMBLY CODE AND IS
 ASSEMBLED BY THE COMPILER. IT PERFORMS A DIRECT READ
 OF THE KEYBOARD BY CALLING CPM BIOS ROUTINE CONIN    *)

CONST
   BIOS_ENTRY_POINT = $EE09;
VAR
   CONSOLE_IN,RESULT : INTEGER;
BEGIN
 INLINE( "CALL /BIOS_ENTRY_POINT/ (* CALL CPM BASIC I/O ENTRY POINT
       "MOV L,A    /         (* MOVE VALUE IN A REGISTER TO L REGISTER *)
       "MVI H/$0   /         (* ZERO OUT H REGISTER                  *)
       "SHLD / RESULT );       (* STORE VALUE IN HL REGISTER IN 'RESULT' *)
 KEY_CHAR := CHR(RESULT);
END;

(*****************************)
FUNCTION KEY_PRESS:BOOLEAN;
(*****************************)
(* THIS ROUTINE IS WRITTEN IN 8080 ASSEMBLY CODE AND IS
 ASSEMBLED BY THE COMPILER. IT PERFORMS A DIRECT READ
 OF THE KEYBOARD INPUT STATUS BY CALLING THE CPM BIOS
 ROUTINE CONST                              *)

CONST
   BIOS_ENTRY_POINT = $EE06;
VAR
   CONSOLE_STATUS,RESULT : INTEGER;
BEGIN
 INLINE( "CALL /BIOS_ENTRY_POINT/ (* CALL CPM BASIC I/O ENTRY POINT
       "MOV L,A    /         (* MOVE VALUE IN A REGISTER TO L REGISTER *)
       "MVI H/$0   /         (* ZERO OUT H REGISTER                  *)
       "SHLD / RESULT );       (* STORE VALUE IN HL REGISTER IN 'RESULT' *)
 CONSOLE_STATUS := RESULT;
 KEY_PRESS := (CONSOLE_STATUS = $FF);
END;
```

```
(**********************************)
FUNCTION BUFFER_EMPTY: BOOLEAN;
(**********************************)
(* THIS PROCEDURE RETURNS TRUE IF THE BUFFER IS EMPTY *)

BEGIN
   BUFFER_EMPTY := END_OF_BUFFER_POINTER <= 1;
END;

(**********************************)
FUNCTION BUFFER_FULL: BOOLEAN;
(**********************************)
(* THIS PROCEDURE RETURNS TRUE IF THE BUFFER IS FULL *)

BEGIN
   BUFFER_FULL := END_OF_BUFFER_POINTER >= MAX_BUFFER_SIZE;
END;

(*********************************************************)
PROCEDURE GET_CHAR_FROM_MODEM(VAR MODEM_CHAR:CHAR);
(*********************************************************)
(* THIS PROCEDURERETURNS THE CHARACTER READ FRON THE INPUT PORT*)

BEGIN
   MODEM_CHAR := INP(MOD_PORT);
END;

(**********************************)
FUNCTION MODEM_INPUT: BOOLEAN;
(**********************************)
(* BIT 0 OF THE MODEM STATUS PORT GOES HIGH WHEN THERE IS A
 CHARACTER IN THE MODEM DATA PORT.  THIS FUNCTION TESTS
 BIT 0 AND RETURNS 'TRUE' IF THERE IS MODEM INPUT        *)

BEGIN
   MODEM_INPUT := TSTBIT(INP(STAT_PORT),0);
END;

(**********************************)
FUNCTION MODEM_READY: BOOLEAN;
(**********************************)
(* THIS FUNCTION RETURNS TRUE IF THE SERIAL PORT IS READY TO
 ACCEPT OUTPUT. IF BIT 2 OF THE STATUS PORT IS HIGH, THE
 PORT IS READY.                                  *)

BEGIN
   MODEM_READY := TSTBIT(INP(STAT_PORT),2);
END;
```

```
(*****************************************)
PROCEDURE MODEM_OUT(OUT_CHAR:CHAR);
(*****************************************)
(* THIS PROCEDURE LOOPS UNTIL THE MODEM PORT IS READY TO
 ACCEPT A CHARACTER AND THEN SENDS THE CHARACTER TO THE
 THE PORT AND RETURNS *)

VAR
   CHAR_SENT:BOOLEAN;
BEGIN
   CHAR_SENT := FALSE;  (* INIT BOOLEAN *)
   REPEAT
      IF MODEM_READY THEN
         BEGIN
            OUT[MOD_PORT]:=OUT_CHAR;
            CHAR_SENT := TRUE;
         END;
   UNTIL CHAR_SENT;
END;

(**********************************)
PROCEDURE DISPLAY_PARAMETERS;
(**********************************)
(* THIS PROCEDURE DISPLAYS A LIST OF THE USER DEFINED
 PARAMETERS AND THEIR CURRENT STATUS *)

VAR
   ENTER_PRESS:CHAR;
   TERM_STRING,PRINT_STRING,STORE_STRING,ON_STRING,OFF_STRING:STRING;
   FREE_SPACE:INTEGER;
BEGIN
   CLEAR_SCREEN;
   IF ACTIVE_FLAG_SET THEN
      TERM_STRING := 'SIGNED ON'
    ELSE
       TERM_STRING := 'SIGNED OFF';
   IF STORE_FLAG THEN
      STORE_STRING := 'ON'
    ELSE
       STORE_STRING := 'OFF';
   FREE_SPACE := MAX_BUFFER_SIZE - END_OF_BUFFER_POINTER +1;
```

```
     (* PROCEDURE DISPLAY_PARAMETERS CONT'D *)

  WRITELN('                  CURRENT PARAMETERS');
  WRITELN('                  ==================');
  WRITELN;
  WRITELN('      TERMINAL STATUS        ',CHR(14),TERM_STRING,CHR(15));
  WRITELN;
  WRITELN('      BUFFER OPTION         ',CHR(14),STORE_STRING,CHR(15));
  WRITELN;
  WRITELN('        ',FREE_SPACE,' CHARACTERS OF BUFFER SPACE REMAINING');
  WRITELN;
  WRITE('    BAUD RATE           ',CHR(14),BAUD_RATE,CHR(15));
  WRITELN('     PARITY             ',CHR(14),PARITY,CHR(15));
  WRITELN;
  WRITE('    WORD LENGTH         ',CHR(14),WORD_LENGTH,' BITS',CHR(15));
  WRITELN('    STOP BITS           ',CHR(14),STOP_BITS,CHR(15));
  WRITELN;
  WRITE('    HOST CURSOR         ',CHR(14),HOST_CURSOR,CHR(15));
  WRITELN('       PAD CHARACTERS       ',CHR(14),NR_OF_PADS,CHR(15));
  WRITELN;
  WRITELN('      SIGNON MESSAGE       ',CHR(14),SIGNON_MESSAGE,CHR(15));
  WRITELN;
  WRITELN('      SIGNOFF MESSAGE      ',CHR(14),SIGNOFF_MESSAGE,CHR(15));
  WRITELN;
  WRITELN('          PRESS <ENTER> TO CONTINUE');
  READ(ENTER_PRESS);
END;

(****************************************)
PROCEDURE SAVE_COMM_PARAMETERS;
(****************************************)
(* THIS PROCEDURE WRITES THE CURRENT COMMUNICATIONS   *)
(* PARAMETERS TO THE FILE TERM.DAT.                   *)

VAR
  PARAMETER_RECORD:PARAM_RECORD;
  CURRENT_PARAMS_FILE:PARAM_FILE;
  CLOSE_CODE:INTEGER;
  GOOD_OPEN,GOOD_IO:BOOLEAN;
```

```
        (* PROCEDURE SAVE_COMM_PARAMETERS CONT'D *)

BEGIN
  WITH PARAMETER_RECORD DO
    BEGIN
      ON_MESSAGE:=SIGNON_MESSAGE;
      OFF_MESSAGE:=SIGNOFF_MESSAGE;
      ACTIV_FLAG:=ACTIVE_FLAG_SET;
      PARITY_PARAM:=PARITY;
      BAUD_PARAM:=BAUD_RATE;
      WORD_LNGTH:=WORD_LENGTH;
      STP_BITS:=STOP_BITS;
      C_PATTERN:=C_REGISTER;
      D_PATTERN:=D_REGISTER;
      E_PATTERN:=E_REGISTER;
      CURSOR_CHAR:=HOST_CURSOR;
      PAD_CHARACTERS:=NR_OF_PADS;
    END;
  ASSIGN(CURRENT_PARAMS_FILE,'A:TERM.DAT');
  REWRITE(CURRENT_PARAMS_FILE);
  GOOD_OPEN := (IORESULT <> 255);
  IF GOOD_OPEN THEN
    BEGIN
      WRITE(CURRENT_PARAMS_FILE,PARAMETER_RECORD);
      GOOD_IO := (IORESULT = 0);
      IF NOT GOOD_IO THEN
        BEGIN
          WRITELN(' **** ERROR - BAD WRITE TO TERM.DAT ****');
          WRITELN(' PRESS <ENTER> TO CONTINUE ');
          READ(ENTER_PRESS);
        END;
      CLOSE(CURRENT_PARAMS_FILE,CLOSE_CODE);
      IF CLOSE_CODE = 255 THEN
        BEGIN
          WRITELN('      **** ERROR - CANNOT CLOSE TERM.DAT ****');
          WRITELN(' PRESS <ENTER> TO CONTINUE ');
          READ(ENTER_PRESS);
        END;
    END
  ELSE
    BEGIN
      WRITELN(' **** ERROR - UNABLE TO OPEN TERM.DAT ****');
      WRITELN(' PRESS <ENTER> TO CONTINUE ');
      READ(ENTER_PRESS);
    END;
END;
```

```
(**********************************)
PROCEDURE CHANGE_PARAMETERS;
(**********************************)
(* THIS PROCEDURE TRANSFERS CONTROL BY CHAINING TO
 PROGRAM "SETPARAM.COM" . THE CALLED PROGRAM
 CONSISTS OF ONE SUBPROCEDURE TO CHANGE EACH
 COMMUNICATIONS PARAMETER AND A CALL TO CPM TO
 PROGRAM THE UART. CONTROL IS THEN CHAINED BACK
 TO THIS PROGRAM. *)

BEGIN
  CLEAR_SCREEN;
  WRITELN('                *** W A R N I N G ***');
  WRITELN;
  WRITELN('           WHILE MODIFYING THE PARAMETERS ANY INFORMATION');
  WRITELN('           CONTAINED IN THE BUFFER WILL BE LOST AND THE ');
  WRITELN('           RECEIVE TO BUFFER OPTION WILL BE TURNED OFF. ');
  WRITELN;
  WRITELN('           PRESS <ENTER> TO CONTINUE ');
  WRITELN('              OR <ESC> TO RETURN TO THE MENU ');
  READ(ANSWER);
  IF ANSWER = CHR(BLANK_CHR) THEN
    BEGIN
      SAVE_COMM_PARAMETERS;   (* SAVE ACTIVE FLAG *)
      ASSIGN(CHAIN_FILE,'A:SETPARAM.COM');
      RESET(CHAIN_FILE);
      IF IORESULT = 255 THEN
        WRITELN('UNABLE TO OPEN SETPARAM.COM ')
      ELSE
        CHAIN(CHAIN_FILE);
    END;
   (* ANY INPUT EXCEPT POSITIVE RESPONSE RETURNS TO MENU *)
END;
```

```
(******************************)
PROCEDURE EXAMINE_BUFFER;
(******************************)
(* IF THERE IS DATA IN THE BUFFER THIS PROCEDURE DISPLAYS *)
(* THE CONTENTS OF THE BUFFER ON THE SCREEN              *)

VAR
   ENTER_PRESS:CHAR;
   POINTER:INTEGER;
BEGIN
   CLEAR_SCREEN;
   IF NOT BUFFER_EMPTY THEN
      FOR POINTER := 1 TO END_OF_BUFFER_POINTER DO
         WRITE(INBUFFER[POINTER])
   ELSE
      BEGIN
         WRITELN;
         WRITELN('                 **** BUFFER EMPTY ****');
      END;
   WRITELN; WRITELN;
   WRITELN('                    PRESS <ENTER> TO CONTINUE');
   READ(ENTER_PRESS);
END;

(****************************************)
PROCEDURE WAIT_FOR_HOST_TO_ACCEPT;
(****************************************)
(* THIS PROCEDURE LOOKS FOR THE PROMPT CHARACTER ACKNOWLEDGING
         THAT THE TRANSMITTED LINE HAS BEEN ACCEPTED, DISCARDS THE
 PAD CHARACTERS FOLLOWING IT AND RETURNS TO THE CALLING ROUTINE*)

VAR
   COUNTER:INTEGER;
   IN_CHAR:CHAR;
BEGIN
   COUNTER := 0;
   REPEAT
      IF MODEM_INPUT THEN          (* LOOP UNTIL CURSOR RETURNS *)
         GET_CHAR_FROM_MODEM(IN_CHAR);
   UNTIL IN_CHAR = HOST_CURSOR;
   REPEAT
      IF MODEM_INPUT THEN
         BEGIN
            GET_CHAR_FROM_MODEM(IN_CHAR);     (* SHUCK PAD CHARS *)
            COUNTER := COUNTER + 1;
         END;
   UNTIL COUNTER = NR_OF_PADS;
   WRITE('*');   (* SHOW USER THAT HOST HAS RESPONSED *)
END;
```

```
(**************************)
PROCEDURE TRANSFER_FILE;
(**************************)
(* THIS PROCEDURE TRANSFERS A FILE TO ANOTHER COMPUTER
 BY TRANSMITTING ONE LINE AND THEN WAITING ON THE HOST
 TO RESPOND WITH A CURSOR CHARACTER BEFORE TRANSMITTING
 THE NEXT.                                  *)

VAR
   CLOSE_CODE,LINE_POINTER,CHAR_COUNTER:INTEGER;
   XMIT_FILENAME:STRING;
   VALUE_ENTERED:CHAR;
   IN_FILE:CHAR_FILE;
   CR_LAST_CHAR,ALL_DONE,GOOD_OPEN,RESULT:BOOLEAN;

(*********************************)
PROCEDURE OPEN_TRANSFER_FILE;
(*********************************)
(* LOOP UNTIL THE TRANSFER FILE IS OPENED *)
BEGIN
   REPEAT
      CLEAR_SCREEN;
      WRITELN;WRITELN;WRITELN;WRITELN;
      WRITELN('        ENTER NAME OF FILE TO TRANSFER.......? - ');
      READLN(XMIT_FILENAME);
      IF LENGTH(XMIT_FILENAME) > 0 THEN
         BEGIN
            ASSIGN(IN_FILE,XMIT_FILENAME);
            RESET(IN_FILE);
            GOOD_OPEN := IORESULT <> 255;
            IF NOT GOOD_OPEN THEN
               BEGIN
                  WRITELN(' **** BAD OPEN ON ',XMIT_FILENAME,' ****');
                  WRITELN(' PRESS <ENTER> TO CONTINUE ');
                  WRITELN('      <ESC>  TO RETURN TO MENU ');
                  READ(VALUE_ENTERED);
                  IF VALUE_ENTERED = CHR(ESCAPE_CHR) THEN EXIT;
               END;
         END;
   UNTIL GOOD_OPEN;
END;

(****)

BEGIN
   OPEN_TRANSFER_FILE;
   END_OF_BUFFER_POINTER := 1;  (* FLUSH BUFFER *)
   ALL_DONE := FALSE;  (* INIT BOOLEAN *)
 (* PROCEDURE TRANSFER_FILE CONT'D *)

  WHILE NOT ALL_DONE DO  (* FILL AND SEND BUFFER UNTIL EOF *)
    BEGIN
      WHILE (NOT BUFFER_FULL) AND (NOT ALL_DONE) DO
        BEGIN
          INBUFFER[END_OF_BUFFER_POINTER] := IN_FILE^;
          WRITE(INBUFFER[END_OF_BUFFER_POINTER]);
          GET(INFILE);
          IF EOF(INFILE) THEN
            ALL_DONE := TRUE;
          END_OF_BUFFER_POINTER := END_OF_BUFFER_POINTER+1;
        END;
      LINE_POINTER := 1; CHAR_COUNTER := 1;
      WHILE LINE_POINTER < END_OF_BUFFER_POINTER DO
        BEGIN
(* XMIT LINE *)     WHILE (INBUFFER[LINE_POINTER] <> CHR(CARRIAGE_RTN_CHR
                    AND (CHAR_COUNTER <  MAX_LINE_LENGTH) DO
              BEGIN
                IF (ORD(INBUFFER[LINE_POINTER]) > 31) AND
                   (ORD(INBUFFER[LINE_POINTER]) < 127) THEN
(* LOOP HERE UNTIL *)       BEGIN
(* 80 CHAR SENT OR *)         MODEM_OUT(INBUFFER[LINE_POINTER]);
(* CR IS READ      *)       CR_LAST_CHAR := FALSE;
                  END;
                LINE_POINTER := LINE_POINTER + 1;
                CHAR_COUNTER := CHAR_COUNTER +1;
              END;
            IF CR_LAST_CHAR = TRUE THEN (* SEND BLANK LINE *)
              BEGIN
                MODEM_OUT(CHR(BLANK_CHR));
                MODEM_OUT(CHR(BLANK_CHR));
              END;
(* SEND CR  *)     MODEM_OUT(CHR(CARRIAGE_RTN_CHR));
            CR_LAST_CHAR := TRUE;
(* WHEN EOL *)     IF INBUFFER[LINE_POINTER] = CHR(CARRIAGE_RTN_CHR) THEN
              LINE_POINTER := LINE_POINTER + 1; (* SHUCK CR *)
(* RESET COUNTER *) CHAR_COUNTER := 0;
            WAIT_FOR_HOST_TO_ACCEPT;
            IF KEY_PRESS THEN
              BEGIN
                READ_KEYBOARD(VALUE_ENTERED);
                  IF VALUE_ENTERED = CHR(ESCAPE_CHR) THEN EXIT;
              END;
        END;
      END_OF_BUFFER_POINTER := 1; (* FLUSH BUFFER *)
    END;
  CLOSE(IN_FILE,CLOSE_CODE);
END;
```

```
(************************)
PROCEDURE SAVE_BUFFER;
(************************)
(* THIS PROCEDURE SAVES THE CONTENTS OF THE
 BUFFER TO A DESIGNATED OUTPUT FILE. THE
 USER IS GIVEN THE OPTION OF STRIPPING
 THE LINE NUMBERS FROM THE CODE BEFORE IT
 IS SAVED TO DISK *)

VAR
   CLOSE_CODE,LINE_POINTER,BUFFER_POINTER:INTEGER;
   ENTER_PRESS:CHAR;
   OUT_FILENAME:STRING;
   OUT_FILE:CHAR_FILE;
   STRIP_NUMBERS,GOOD_WRITE,RESULT,GOOD_OPEN:BOOLEAN;

(****************************)
PROCEDURE FIND_FIRST_CHAR;
(****************************)
(* THIS PROCEDURE FINDS THE FIRST CHARACTER OF
 A PROGRAM LINE BY READING CHARACTERS UNTIL
 THE LINE NUMBER IS FOUND OR THE END OF THE
 BUFFER IS REACHED. IT THEN DISCARDS THE
 LINE NUMBER AND THE LEADING SPACE BETWEEN
 THE LINE NUMBER AND THE FIRST CHARACTER OF
 THE PROGRAM LINE. *)

VAR
   NUMBER_FOUND:BOOLEAN;
BEGIN
   (* FIND FIRST LINE NUMBER *)
   NUMBER_FOUND := FALSE;
   WHILE (NOT NUMBER_FOUND) AND
        ( BUFFER_POINTER <= END_OF_BUFFER_POINTER ) DO
   BEGIN
         IF INBUFFER[BUFFER_POINTER] IN ['0'..'9'] THEN
            NUMBER_FOUND := TRUE
         ELSE
            BUFFER_POINTER := BUFFER_POINTER + 1;
      END;
   (* STRIP LINE NUMBER *)
   WHILE (INBUFFER[BUFFER_POINTER + 1] IN ['0'..'9'] ) AND
        ((BUFFER_POINTER + 1 ) <= END_OF_BUFFER_POINTER) DO
         BUFFER_POINTER := BUFFER_POINTER + 1;
   (* REMOVE LEADING SPACES *)
   BUFFER_POINT := BUFFER_POINTER + LEADING_SPACES;
END;
```

```
(****************************)
PROCEDURE OPEN_SAVE_FILE;
(****************************)
(* LOOP UNTIL AN OUTPUT FILE IS OPENED *)

BEGIN
  REPEAT
    WRITELN;WRITELN;WRITELN;WRITELN;
    WRITELN('                    - OUTPUT FILENAME....? - ');
    READLN(OUT_FILENAME);
    IF LENGTH(OUT_FILENAME) > 0 THEN
      BEGIN
        ASSIGN(OUT_FILE,OUT_FILENAME);
        REWRITE(OUT_FILE);
        GOOD_OPEN := IORESULT <> 255;
        IF NOT GOOD_OPEN THEN
          BEGIN
            WRITELN(' **** BAD OPEN ON ',OUT_FILENAME,' ****');
            WRITELN(' PRESS <ENTER> TO CONTINUE ');
            WRITELN('      <ESC> TO RETURN TO THE MENU ');
            READ(ENTER_PRESS);
            IF ENTER_PRESS = CHR(ESCAPE_CHR) THEN EXIT;
          END;
      END;
  UNTIL GOOD_OPEN;

END;

(****)

BEGIN
  CLEAR_SCREEN;
  IF NOT BUFFER_EMPTY THEN
    BEGIN
      OPEN_SAVE_FILE;
      BUFFER_POINTER := 1;
      GOOD_WRITE := TRUE;     (* INIT BOOLEAN *)
      WRITELN('STRIP LINE NUMBERS - <Y>ES OR <N>O ???');
      READ(ENTER_PRESS);
      IF ENTER_PRESS = 'Y' THEN
        BEGIN
          STRIP_NUMBERS := TRUE;
          FIND_FIRST_CHAR;      (* PRIME FIRST LINE *)
        END
      ELSE
        STRIP_NUMBERS := FALSE;
      WHILE (BUFFER_POINTER < END_OF_BUFFER_POINTER) AND GOOD_WRITE I
```

```
(* PROCEDURE SAVE_BUFFER CONT'D *)

        BEGIN
          IF (INBUFFER[BUFFER_POINTER] = CHR(LINE_FEED_CHAR))
             AND (STRIP_NUMBERS) THEN
(* WHEN EOL FOUND *)    BEGIN
(* XMIT LF AND    *)      OUT_FILE^ := CHR(LINE_FEED_CHAR);
(* FIND FIRST OF  *)      PUT(OUT_FILE);
(* NEXT LINE      *)      FIND_FIRST_CHAR; (* OF NEXT LINE *)
                        END;
(* WRITE BUFFER *)  OUT_FILE^ := INBUFFER[BUFFER_POINTER];
(* CHARACTER TO *)  PUT(OUT_FILE);
(* DISK         *)  GOOD_WRITE := (IORESULT = 0);
            BUFFER_POINTER := BUFFER_POINTER + 1;
            IF BUFFER_POINTER MOD 5 = 0 THEN
               WRITE('*');    (* SHOW USER ITS WORKING *)
        END;
      CLOSE(OUT_FILE,CLOSE_CODE);
      IF NOT GOOD_WRITE THEN
        BEGIN
          CLEAR_SCREEN;
          WRITELN; WRITELN; WRITELN;
          WRITELN(' **** WRITE ERROR TO',OUT_FILENAME);
          WRITELN; WRITELN;
          WRITELN('         PRESS <ENTER> TO CONTINUE');
          READ(ENTER_PRESS);
        END;
    END
  ELSE
    BEGIN
      WRITELN;
      WRITELN('             **** BUFFER EMPTY ****');
      WRITELN('                  PRESS <ENTER> TO CONTINUE');
      READ(ENTER_PRESS);
    END;
END; (* END PROCEDURE *)
```

```
(***************************)
PROCEDURE PRINT_BUFFER;
(***************************)
(* IF THERE IS DATA IN THE BUFFER, THIS PROCEDURE PROMPTS THE USER
 TO TURN ON THE PRINTER AND THEN PRINTS EACH LOCATION IN THE
    BUFFER. *)


VAR
   KEY_PRESS:CHAR;
   CLOSE_CODE,POINTER:INTEGER;
   PRINT_FILE:CHAR_FILE;
BEGIN
   CLEAR_SCREEN;
   IF NOT BUFFER_EMPTY THEN
      BEGIN
         WRITELN('        **** INSURE THE PRINTER IS READY ****');
         WRITELN('              THEN PRESS <ENTER> TO CONTINUE');
         WRITELN('               OR <ESC> TO RETURN TO THE MENU');
         READ(KEY_PRESS);
         IF KEY_PRESS = CHR(BLANK_CHR) THEN (* <ENTER> PRESSED *)
            BEGIN
               ASSIGN(PRINT_FILE,'LST:');
               REWRITE(PRINT_FILE);
               FOR POINTER := 1 TO END_OF_BUFFER_POINTER DO
                  BEGIN
                     PRINT_FILE^ := INBUFFER[POINTER];
                     PUT(PRINT_FILE);
                  END;
               CLOSE(PRINT_FILE,CLOSE_CODE);
            END;
       (* ELSE <ESC> PRESSED *)
      END
   ELSE
      BEGIN
         WRITELN;
         WRITELN('                **** BUFFER EMPTY ****');
      END;
   WRITELN; WRITELN;
         WRITELN('                   PRESS <ENTER> TO CONTINUE');
   READ(ENTER_PRESS);
END;
```

```
(**************************************************************************
PROCEDURE STORE_IN_MEMORY(STORE_CHAR:CHAR; VAR MEMORY_FULL:BOOLEA
(**************************************************************************
(* IF THE BUFFER IS NOT FULL, THIS PROCEDURE TAKES THE INPUT
 CHARACTER AND STORES IT IN THE NEXT FREE LOCATION IN THE BUFFER
 A BOOLEAN IS RETURNED TO THE CALLING ROUTINE - "TRUE" IF THE
 BUFFER IS FULL. *)

BEGIN
   IF BUFFER_FULL THEN
      BEGIN
         MEMORY_FULL := TRUE;
         WRITELN('***  MEMORY BUFFER FULL  ***');
      END
   ELSE
      BEGIN
         MEMORY_FULL := FALSE;
         INBUFFER[END_OF_BUFFER_POINTER] := STORE_CHAR;
         END_OF_BUFFER_POINTER := END_OF_BUFFER_POINTER + 1;
      END;
END;

(************************)
PROCEDURE SEND_BREAK;
(************************)
(* THIS PROCEDURE TRANSMITS A BREAK SEQUENCE BY XMITTING AN  *)
(* UNINTERRUPTED SPACE TONE FROM THE MODEM FOR THE TIME IT   *)
(* TAKES TO COMMPLETE THE DELAY LOOP                  *)

VAR
   DELAY_COUNTER,INSIDE_COUNTER:INTEGER;
BEGIN
   OUT[STAT_PORT] := $5;
   OUT[STAT_PORT] := $0BA;
   DELAY_COUNTER := 0;
   WHILE DELAY_COUNTER < BREAK_DELAY DO
      DELAY_COUNTER := DELAY_COUNTER + 1;
   OUT[STAT_PORT] := $5;
   OUT[STAT_PORT] := $0AA;
END;
```

```
(**********************)
PROCEDURE TERMINAL;
(**********************)
(* THIS PROCEDURE IS AN INFINITE LOOP THAT HANDLES KEYBOARD AND MODEM
 INPUTS UNTIL THE  ESCAPE CHARACTER IS TYPED. IF THE STORE FLAG IS
 SET ALL TRANSACTIONS ARE STORED IN THE PROGRAM BUFFER.   *)

VAR
   IN_CHAR, OUT_CHAR:CHAR;
   BUFFER_FULL, TERMINAL_WANTED:BOOLEAN;
   INVALID_INPUTS:SET OF CHAR;
   CHAR_IN_BUFFER,CLOSE_CODE:INTEGER;
BEGIN
   INVALID_INPUTS := [CHR(CLEAR_CHR),CHR(REV_VIDEO),
                 CHR(DELETE_CHR),CHR(GRAPH_CHR)];
   (* INPUTS THAT MIGHT DESTROY DATA ON THE SCREEN ARE IGNORED *)
   TERMINAL_WANTED := TRUE;
   WHILE TERMINAL_WANTED DO
    BEGIN
     IF KEY_PRESS THEN
        BEGIN
           READ_KEYBOARD(OUT_CHAR);
           IF OUT_CHAR = CHR(ESCAPE_CHR) THEN
              TERMINAL_WANTED := FALSE       (* RETURN TO MENU *)
           ELSE
              IF OUT_CHAR = CHR(F1_KEY) THEN
                 SEND_BREAK  (* XMIT BREAK SEQUENCE *)
              ELSE
                 MODEM_OUT(OUT_CHAR); (* XMIT KEYBOARD CHARACTER *)
        END;
     IF MODEM_INPUT THEN
        BEGIN
           GET_CHAR_FROM_MODEM(IN_CHAR);
           IN_CHAR := CHR(ORD(IN_CHAR & 127)); (*  STRIP PARITY BIT  *)
           IF NOT (IN_CHAR IN INVALID_INPUTS) THEN
              BEGIN
(* DISPLAY INPUT *)    WRITE(IN_CHAR);
(* FROM MODEM   *)    IF STORE_FLAG_SET THEN
                    BEGIN
                       IF BUFFER_FULL THEN
                          BEGIN
                             MODEM_OUT(CHR(CNTRL_S));
                             WRITELN(' **** BUFFER FULL ****');
                             WRITELN(' PUSH <CTRL><R> TO CONTINUE');
                          END;
                       STORE_IN_MEMORY(IN_CHAR,BUFFER_FULL);
                    END; (* ENDIF *)
              END;
        END;
    END;
END;
```

```
(*****************************)
PROCEDURE SIGNON_SIGNOFF;
(*****************************)
(* THIS ROUTINE TOGGLES THE ACTIVE FLAG AND TRANSMITTS THE
 APPROPRIATE MESSAGE DEPENDING ON THE CURRENT VALUE OF
 THE ACTIVE FLAG. *)

VAR
   POINTER:INTEGER;
BEGIN
   CLEAR_SCREEN;
   IF ACTIVE_FLAG_SET THEN
      BEGIN
         FOR POINTER := 1 TO MAX_MESSAGE_LENGTH DO
            BEGIN
               MODEM_OUT(SIGNOFF_MESSAGE[POINTER]);
               WRITE(SIGNOFF_MESSAGE[POINTER]);
            END;
         MODEM_OUT(CHR(CARRIAGE_RTN_CHR));
         ACTIVE_FLAG_SET := FALSE;
      END
   ELSE
      BEGIN
         FOR POINTER := 1 TO MAX_MESSAGE_LENGTH DO
            BEGIN
               MODEM_OUT(SIGNON_MESSAGE[POINTER]);
               WRITE(SIGNON_MESSAGE[POINTER]);
            END;
         MODEM_OUT(CHR(CARRIAGE_RTN_CHR));
         ACTIVE_FLAG_SET := TRUE;
      END;
END;
```

```
(*****************)
PROCEDURE MENU;
(*****************)
(*THIS PROCEDURE DISPLAYS THE MENU OF OPTIONS, ACCEPTS
 THE USERS CHOICE, AND THEN PASSES CONTROL TO THE
 APPROPRIATE ROUTINE. IF THE INPUT DOES NOT MATCH A
 VALID CHOICE THE MENU IS REDRAWN AND THE USER IS
 PROMPTED FOR ANOTHER CHOICE *)

VAR
  LOOP_COUNTER : INTEGER;
  CHOICE : CHAR;
BEGIN
  CHOICE := 'A';
  WHILE CHOICE <> 'R' DO
    BEGIN
      CLEAR_SCREEN;
      WRITELN;
      WRITELN('       SELECT KEY       OPTION ');
      WRITELN('       ==========       ====== ');
      WRITELN;
      WRITELN('          <S>      <SIGN-ON/SIGN-OFF>');
      WRITELN(' ');
      WRITELN('          <T>      <TERMINAL>      ');
      WRITELN(' ');
      WRITELN('          <B>      <RECEIVE TO BUFFER>');
      WRITELN(' ');
      WRITELN('          <E>      <EXAMINE BUFFER>       ');
      WRITELN(' ');
      WRITELN('          <P>      <PRINT BUFFER>  ');
      WRITELN(' ');
      WRITELN('          <D>      <SAVE BUFFER TO DISK>');
      WRITELN(' ');
      WRITELN('          <H>      <TRANSFER FILE TO HOST>');
      WRITELN(' ');
      WRITELN('          <M>      <MODIFY PARAMETERS>    ');
      WRITELN;
      WRITELN('          <V>      <VERIFY PARAMETERS>    ');
      WRITELN;WRITELN;
      WRITELN('          <R>      <RETURN TO SYSTEM LEVEL>');

      REPEAT ; UNTIL KEYPRESS; (* POLL KEYBOARD UNTIL CHAR READY *)
      READ_KEYBOARD(CHOICE); (* GET CHARACTER *)
```

```
        CASE UPPERCASE(CHOICE) OF

          'S': BEGIN
                 SIGNON_SIGNOFF;
                 TERMINAL;
               END;

          'T': BEGIN
                 CLEAR_SCREEN;
                 TERMINAL;
               END;

          'P': PRINT_BUFFER;

          'B': IF STORE_FLAG THEN
                 STORE_FLAG := FALSE
                ELSE
                  BEGIN  (* TOGGLE STORE FLAG *)
                    STORE_FLAG := TRUE;
                    END_OF_BUFFER_POINTER := 1;
                  END;

          'D': SAVE_BUFFER;

          'H': TRANSFER_FILE;

          'E': EXAMINE_BUFFER;

          'M': CHANGE_PARAMETERS;

          'V': DISPLAY_PARAMETERS;

          'R': WRITELN(' -  RETURNING');

          ELSE ;  (* INVALID INPUT CAUSES MENU TO PRINT AGAIN *)

         END; (* END CASE *)
       END; (* END WHILE *)
END;
```

```
(****************************************************************)
PROCEDURE PROGRAM_SIO(C_ENTRY,D_ENTRY,E_ENTRY:INTEGER);
(****************************************************************)
(* THIS PROCEDURE IS WRITTEN IN 8080 ASSEMBLY CODE AND IS
 ASSEMBLED DURING COMPILATION BY THE MTPLUS COMPILER.
 IT LOADS THE REGISTERS WITH THE BIT PATTERNS PREVIOUSLY
 READ FROM TERM.DAT AND CALLS THE OPERATING SYSTEMS
 SETSIO ROUTINE TO PROGRAM THE SERIAL I/O CONTROLLER *)

CONST
   CPM_ENTRY_POINT = $40;
BEGIN
   INLINE( $2A / C_ENTRY   / (* LOAD HL REGISTER PAIR WITH C MASK  *)
         $4D /          (* MOVE VALUE IN L TO C REGISTER      *)
         $2A / D_ENTRY   / (* LOAD HL REGISTER PAIR WITH D MASK  *)
         $55 /          (* MOVE VALUE IN L TO D REGISTER      *)
         $2A / E_ENTRY   / (* LOAD HL REGISTER PAIR WITH E       *)
         $5D /          (* MOVE VALUE IN L TO E REGISTER      *)
         $2E / $11      / (* SET XMIT ON CHAR TO CRTL Q *)
         $26 / $13      / (* SET XMIT OFF CHAR TO CRTL S *)
         $CD / CPM_ENTRY_POINT);
END;

(************************)
PROCEDURE INITIALIZE;
(************************)
(* THIS PROCEDURE READS THE COMMUNICATIONS PARAMETERS FROM
 FILE TERM.DAT AND PROGRAMS THE MODEL II UART.
 IF TERM.DAT CANNOT BE READ DEFAULT VALUES ARE USED      *)

VAR
   PARAMETER_RECORD:PARAM_RECORD;
   IN_FILE:PARAM_FILE;
   CLOSE_CODE:INTEGER;
   ENTER_PRESS:CHAR;
   GOOD_OPEN,GOOD_IO:BOOLEAN;
BEGIN
   CLEAR_SCREEN;
   ASSIGN(IN_FILE,'A:TERM.DAT');
   RESET(IN_FILE);
   GOOD_OPEN := IORESULT <> 255;
   IF GOOD_OPEN THEN
      BEGIN
         READ(IN_FILE,PARAMETER_RECORD);
         GOOD_IO := IORESULT = 0;
         IF GOOD_IO THEN
```

```
(* PROCEDURE INITIALIZE CONT'D *)

          WITH PARAMETER_RECORD DO
              BEGIN
                SIGNON_MESSAGE:=ON_MESSAGE;
                SIGNOFF_MESSAGE:=OFF_MESSAGE;
                ACTIVE_FLAG_SET:=ACTIV_FLAG;
                PARITY:=PARITY_PARAM;
                BAUD_RATE:=BAUD_PARAM;
                WORD_LENGTH:=WORD_LNGTH;
                STOP_BITS:=STP_BITS;
                C_REGISTER:=C_PATTERN;
                D_REGISTER:=D_PATTERN;
                E_REGISTER:=E_PATTERN;
                HOST_CURSOR:=CURSOR_CHAR;
                NR_OF_PADS:=PAD_CHARACTERS;
             END;
     END;
   IF (NOT GOOD_OPEN) AND (NOT GOOD_IO) THEN
     BEGIN
       WRITELN('          **** UNABLE TO READ PARAMETER FILE ****');
       WRITELN('             DEFAULT PARAMETERS WILL BE USED');
       WRITELN('             PRESS <ENTER> TO CONTINUE');
       READ(ENTER_PRESS);
       SIGNON_MESSAGE := 'SIGNON          ';
       SIGNOFF_MESSAGE := 'SIGNOFF          ';
       ACTIVE_FLAG_SET := FALSE;
       PARITY:='EVEN';
       BAUD_RATE:=' 300';
       WORD_LENGTH:='7';
       STOP_BITS:='1';
       (* REGISTER PATTERNS FOR VALUES ABOVE *)
       C_REGISTER:=3;
       D_REGISTER:=198;
       E_REGISTER:=3;
       HOST_CURSOR:='>';
       NR_OF_PADS:=2;
     END;
   PROGRAM_SIO(C_REGISTER,D_REGISTER,E_REGISTER); (* SETUP SIO CONTROLLE
   END_OF_BUFFER_POINTER := 1;
   STORE_FLAG := FALSE;
   DISPLAY_PARAMETERS;
END;
```

```
BEGIN (* MAIN PROGRAM *)

(* THIS IS THE MAIN ROUTINE OF THE PROGRAM. AFTER
 INITIALIZING IT DISPLAYS A MENU SELECTION OF
 FUNCTIONS AVAILABLE TO THE USER. WHEN CONTROL
 RETURNS TO THE ROUTINE, IF THE USER HAS NOT
 SIGNED OFF, HE IS GIVEN THE OPPORTUNITY TO
 TRANSMIT THE SIGNOFF MESSAGE BEFORE RETURNING
 TO O.S.CONTROL. BEFORE EXITING THE CURRENT
 COMMUNICATIONS PARAMETERS ARE SAVED. *)

INITIALIZE;
MENU; (* DISPLAY SYSTEM FUNCTIONS *)
IF ACTIVE_FLAG_SET THEN (* USER HAS NOT SIGNED OFF *)
   BEGIN
      CLEAR_SCREEN;
      WRITELN('         YOU ARE STILL SIGNED ON ....');WRITELN;
      WRITELN('         DO YOU WANT TO SIGN OFF - ENTER <Y>ES OR <N>O');
      WRITELN;WRITELN;WRITELN;
      WRITELN('         IF YOU ANSWER YES THE PROGRAM WILL TRANSMIT     ');
      WRITELN('         THE SIGN OFF MESSAGE. HOWEVER,IT IS YOUR        ');
      WRITELN('         RESPONSIBILITY TO INSURE YOU ARE CURRENTLY AT A ');
      WRITELN('         LEGITIMATE LEVEL WITHIN THE HOST SYSTEM TO ISSUE ');
      WRITELN('         THIS COMMAND.                           ');
      READ(ANSWER);
      IF ANSWER = 'Y' THEN
         SIGNON_SIGNOFF;    (* DEFAULT IS SIGNON *)
   END;
SAVE_COMM_PARAMETERS;
END.
```

APPENDIX C

PROGRAM SETPARAM SOURCE CODE

```
(*************************)
PROGRAM SET_PARAMETERS;
(*************************)
(*=========================================================
 PROGRAM TITLE: SET PARAMETERS

 PROGRAM AUTHOR: DAN VESTAL

 PROGRAM FILE: SETPARAM.PAS

 LAST UPDATE: 3 NOV 81

 PROGRAM SUMMARY:

   THIS PROGRAM IS A MODULE OF PROGRAM  SMART TERM,
   A MICRO-COMPUTER INTERCOMMUNICATIONS PACKAGE FOR
   THE RADIO SHACK TRS-80 MODEL II MICRO COMPUTER.
   THE PROGRAM READS THE CURRENT COMMUNICATIONS
   PARAMETERS FROM FILE TERM.DAT, DISPLAYS THEM TO THE
   USER AND ALLOWS CHANGES TO BE MADE TO THEIR VALUES.
   THE NEW VALUES AND THEIR CORRESPONDING BIT MASKS
   ARE THEN WRITTEN BACK TO TERM.DAT AND CONTROL
   CHAINED BACK TO THE CALLING PROGRAM.



=========================================================*)
CONST
  MAX_MESSAGE_LENGTH=20; (* LENGTH OF SIGNON/OFF MESSAGES *)
  BLANK_CHR=32;
TYPE
  CHAR_FILE = FILE OF CHAR;
  PARAM_RECORD =
    RECORD
      ON_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH] OF CHAR;
      OFF_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH] OF CHAR;
      ACTIV_FLAG:BOOLEAN;
      PARITY_PARAM:STRING[4];
      BAUD_PARAM:STRING[4];
      WORD_LNGTH:CHAR;
      STP_BITS:CHAR;
      C_PATTERN:INTEGER;
      D_PATTERN:INTEGER;
      E_PATTERN:INTEGER;
      CURSOR_CHAR:CHAR;
      PAD_CHARACTERS:INTEGER;
    END;
  PARAM_FILE = FILE OF PARAM_RECORD;
```

```
VAR
   SIGNON_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH]OF CHAR;
   SIGNOFF_MESSAGE:PACKED ARRAY[1..MAX_MESSAGE_LENGTH]OF CHAR;
   GOOD_READ,ACTIVE_FLAG_SET:BOOLEAN;
   PARITY,BAUD_RATE:STRING[4];
   ENTER_PRESS,ANSWER,WORD_LENGTH,STOP_BITS,HOST_CURSOR:CHAR;
   CHAIN_FILE:FILE;
   C_REGISTER,D_REGISTER,E_REGISTER:INTEGER;
   NR_OF_PADS:INTEGER;
(************************)
PROCEDURE CLR_SCREN;
(************************)
(* CLEAR THE SCREEN AND HOME THE CURSOR *)
BEGIN
   WRITE(CHR(12));
END;
(*********************************)
PROCEDURE DISPLAY_PARAMETERS;
(*********************************)
(* DISPLAY THE CURRENT VALUE OF THE COMMUNICATIONS PARAMETERS *)

VAR
   TERM_STRING,PRINT_STRING,STORE_STRING,ON_STRING,OFF_STRING:STRING;
   FREE_SPACE:INTEGER;
BEGIN
   CLR_SCREN;
   WRITELN('          CURRENT COMMUNICATIONS PARAMETERS');
   WRITELN('          =================================');
   WRITELN;
   WRITELN('      BAUD RATE            ',CHR(14),BAUD_RATE,CHR(15));
   WRITELN;
   WRITELN('      PARITY            ',CHR(14),PARITY,CHR(15));
   WRITELN;
   WRITELN('      WORD LENGTH            ',CHR(14),WORD_LENGTH,'
BITS',CHR(15));
   WRITELN;
   WRITELN('      STOP BITS            ',CHR(14),STOP_BITS,CHR(15));
   WRITELN;
   WRITELN('      HOST CURSOR            ',CHR(14),HOST_CURSOR,CHR(15));
   WRITELN;
   WRITELN('      PAD CHARACTERS         ',CHR(14),NR_OF_PADS,CHR(15));
   WRITELN;
   WRITELN('      SIGNON MESSAGE         ',CHR(14),SIGNON_MESSAGE,CHR(15));
   WRITELN;
   WRITELN('      SIGNOFF MESSAGE        ',CHR(14),SIGNOFF_MESSAGE,CHR(15));
   WRITELN;
   WRITELN('          PRESS <ENTER> TO CONTINUE');
   READ(ENTER_PRESS);
END;
```

```
(**********************************************)
PROCEDURE GET_BAUD_RATE(VAR MASK:INTEGER);
(**********************************************)
(* THIS PROCEDURE LOOPS UNTIL A VALID BAUD RATE IS ENTERED
 AND THEN RETURNS A CORRESPONDING BIT MASK. A VALID
 ENTRY CAN BE A CARRIAGE RETURN *)

VAR
  GOOD_RATE:BOOLEAN;
  NEW_BAUD_RATE:STRING[4];
BEGIN
  CLR_SCREN;
  GOOD_RATE := FALSE  (* INIT BOOLEAN *);
  WHILE NOT GOOD_RATE DO
    BEGIN
      WRITELN;
      WRITELN('                CURRENT BAUD RATE IS ',CHR(14),
           BAUD_RATE,CHR(15));
      WRITELN('                ENTER NEW RATE [1200,600,300,110] ');
      WRITELN('                    OR PRESS <ENTER> TO CONTINUE ');
      WRITELN;
      READ(NEW_BAUD_RATE);
      IF LENGTH(NEW_BAUD_RATE) > 0 THEN
        IF (NEW_BAUD_RAT = '1200') OR (NEW_BAUD_RATE = '600') OR
          (NEW_BAUD_RATE = '300') OR (NEW_BAUD_RATE = '110') THEN
          BEGIN
            BAUD_RATE := NEW_BAUD_RATE;
            GOOD_RATE := TRUE;
          END
        ELSE
          WRITELN('    IS NOT A VALID BAUD RATE')
      ELSE
        GOOD_RATE := TRUE; (* NOTHING ENTERED - USE CURRENT RATE *)
    END;
  IF BAUD_RATE = '110' THEN
    MASK := 0
  ELSE
    IF BAUD_RATE = '300' THEN
      MASK := 3  (* SET BITS 0 & 1 ON *)
    ELSE
      IF BAUD_RATE = '600' THEN
        MASK := 4 (* SET BIT 2 ON *)
          ELSE  (* RATE = 1200 *)
            MASK := 5; (* SET BIT 2 & 0 ON *)
END;
```

```
(*****************************************)
PROCEDURE GET_PARITY(VAR MASK:INTEGER);
(*****************************************)
(* THIS PROCEDURE LOOPS UNTIL A VALID PARITY VALUE IS ENTERED
 AND THEN RETURNS A CORRESPONDING BIT MASK.  A VALID ENTRY
 ENTRY CAN BE A CARRIAGE RETURN *)

VAR
   GOOD_PARITY:BOOLEAN;
   NEW_PARITY:STRING[4];
BEGIN
   GOOD_PARITY := FALSE;
   WHILE NOT GOOD_PARITY DO
      BEGIN
         WRITELN;
         WRITELN('                  CURRENT PARITY IS ',
               CHR(14),PARITY,CHR(15));
         WRITELN('                  ENTER NEW VALUE [ODD,EVEN,NONE] ');
         WRITELN('                     OR PRESS <ENTER> TO CONTINUE ');
         WRITELN;
         READ(NEW_PARITY);
         IF LENGTH(NEW_PARITY) > 0 THEN
            IF (NEW_PARITY = 'ODD') OR (NEW_PARITY = 'EVEN') OR
              (NEW_PARITY = 'NONE') THEN
              BEGIN
                 PARITY := NEW_PARITY_RATE;
                 GOOD_PARITY := TRUE;
              END
            ELSE
              WRITELN('   IS NOT A VALID PARITY')
         ELSE
            GOOD_PARITY := TRUE;
      END;
   IF PARITY = 'NONE' THEN
      MASK := 0   (* SET BIT 0 OFF *)
   ELSE
      IF PARITY = 'EVEN' THEN
         MASK := 3  (* SET BITS 0 & 1 ON *)
      ELSE
         MASK := 1; (* SET BIT 0 ON *)
END;
```

```
(****************************************)
PROCEDURE GET_LENGTH(VAR MASK:INTEGER);
(****************************************)
(* THIS PROCEDURE LOOPS UNTIL A VALID WORD LENGTH IS ENTERED
 AND THEN RETURNS A CORRESPONDING BIT MASK. A VALID ENTRY
 CAN BE A CARRIAGE RETURN *)

VAR
   GOOD_LENGTH:BOOLEAN;
   NEW_LENGTH:CHAR;
BEGIN
   GOOD_LENGTH := FALSE;
   WHILE NOT GOOD_LENGTH DO
      BEGIN
         WRITELN;
         WRITELN('               CURRENT WORD LENGTH IS '
              ,CHR(14),WORD_LENGTH,' BITS',CHR(15));
         WRITELN('               ENTER NEW VALUE [5,6,7,8] ');
         WRITELN('                    OR PRESS <ENTER> TO CONTINUE ');
         WRITELN;
         READ(NEW_LENGTH);
         IF NEW_LENGTH IN ['5','6','7','8'] THEN
            BEGIN
               WORD_LENGTH := NEW_LENGTH;
               GOOD_LENGTH := TRUE;
           END
         ELSE
            IF NEW_LENGTH = ' ' THEN
               GOOD_LENGTH := TRUE
            ELSE
               WRITELN(' IS NOT A VALID WORD LENGTH');
      END;
   CASE WORD_LENGTH OF
      '5': MASK := 00;
      '6': MASK := 32;
      '7': MASK := 64;
      '8': MASK := 96;
   END; (* END CASE *)
END;
```

```
(****************************************)
PROCEDURE GET_BITS(VAR MASK:INTEGER);
(****************************************)
(* THE PROCEDURE LOOPS UNTIL A VALID NUMBER FOR STOP BITS IS
 IS ENTERED AND THE RETURNS THE CORRESPONDING BIT MASK
 A VALID ENTRY CAN BE A CARRIAGE RETURN *)

VAR
  GOOD_BITS:BOOLEAN;
  NEW_BITS:CHAR;
BEGIN
  GOOD_BITS := FALSE;
  WHILE NOT GOOD_BITS DO
    BEGIN
      WRITELN;
      WRITELN('                CURRENT NUMBER OF STOP BITS '
          ,CHR(14),STOP_BITS,CHR(15));
      WRITELN('                ENTER NEW VALUE [1,2]  ');
      WRITELN('                    OR PRESS <ENTER> TO CONTINUE ');
      WRITELN;
      READ(NEW_BITS);
      IF NEW_BITS IN ['1','2'] THEN
        BEGIN
          STOP_BITS := NEW_BITS;
          GOOD_BITS := TRUE;
        END
      ELSE
        IF NEW_BITS = ' ' THEN
          GOOD_BITS := TRUE
        ELSE
          WRITELN(' IS NOT A VALID NUMBER OF BITS');
    END;
  CASE STOP_BITS OF
    '1': MASK := 4;
    '2': MASK := 12;
  END;  (* END CASE *)
END;
```

```
(************************)
PROCEDURE GET_CURSOR;
(************************)
(* WHEN TRANSFERRING A FILE TO THE HOST THE "TERM.COM"
 PROGRAM WAITS FOR THE HOST TO RETURN A CURSOR AS AN
 INDICATION THE LINE TRANSMITTED WAS ACCEPTED.
 THIS PROCEDURE ALLOWS THE USER TO CHANGE THE VALUE
 OF THE CHARACTER THE PROGRAM KEYS ON. *)

VAR
  GOOD_CURSOR:BOOLEAN;
  NEW_CURSOR:CHAR;
BEGIN

  GOOD_CURSOR := FALSE;
  WHILE NOT GOOD_CURSOR DO
    BEGIN
      WRITELN;
      WRITELN('                CURRENT CURSOR '
           ,CHR(14),HOST_CURSOR,CHR(15));
      WRITELN('                ENTER NEW CHARACTER VALUE [>,*,-,.] ');
      WRITELN('                     OR PRESS <ENTER> TO CONTINUE ');
      WRITELN;
      READ(NEW_CURSOR);
      IF NEW_CURSOR IN ['>','*','-','.'] THEN
        BEGIN
          HOST_CURSOR := NEW_CURSOR;
          GOOD_CURSOR := TRUE;
        END
      ELSE
        IF NEW_CURSOR = ' ' THEN
          GOOD_CURSOR := TRUE
        ELSE
          WRITELN(' IS NOT A VALID CURSOR CHARACTER');
    END;
END;
```

```
(*********************************)
PROCEDURE GET_PAD_CHARACTERS;
(*********************************)
(* WHEN TRANSFERRING A FILE TO THE HOST THE "TERM.COM"
 PROGRAM WAITS FOR THE HOST TO RETURN A CURSOR AS AN
 INDICATION THE LINE TRANSMITTED WAS ACCEPTED.
 IF THERE ARE PAD CHARACTERS (CHARACTERS USED FOR
 TIMING OR DELAY) FOLLOWING THE CURSOR CHARACTER,
 THIS PROCEDURE ALLOWS THE USER TO CHANGE THE NUMBER
 OF CHARACTERS IGNORED AFTER THE CURSOR. *)

VAR
  GOOD_PADS:BOOLEAN;
  NEW_PADS:CHAR;
BEGIN
  GOOD_PADS := FALSE;
  WHILE NOT GOOD_PADS DO
    BEGIN
      WRITELN;
      WRITELN('                CURRENT NUMBER OF PAD CHARACTERS '
          ,CHR(14),NR_OF_PADS,CHR(15));
      WRITELN('                ENTER NEW CHARACTER VALUE [0..9] ');
      WRITELN('                    OR PRESS <ENTER> TO CONTINUE ');
      WRITELN;
      READ(NEW_PADS);
      IF NEW_PADS IN ['0'..'9'] THEN
        BEGIN
          NR_OF_PADS := ORD(NEW_PADS)-48; (* CONVER TO INTEGER *)
          GOOD_PADS := TRUE;
        END
      ELSE
        IF NEW_PADS = ' ' THEN
          GOOD_PADS := TRUE
        ELSE
          WRITELN(' IS NOT A VALID NUMBER OF PADS');
    END;
END;
```

```
(****************************)
PROCEDURE GET_ON_MESSAGE;
(****************************)
(* THIS PROCEDURE ACCEPTS A NEW SIGN ON STRING OF
 MAX MESSAGE LENGTH OR LESS. OR ACCEPTS A
 CARRIAGE RETURN IF NO CHANGE IS DESIRED *)

VAR
  SUBSCRIPT:INTEGER;
  TEMP_NEW_ON:STRING[MAX_MESSAGE_LENGTH];
BEGIN
  WRITELN;
  WRITELN('                THE CURRENT SIGNON MESSAGE IS ');
  WRITELN('                ',CHR(14),SIGNON_MESSAGE,CHR(15));
  WRITELN('                ENTER A NEW STRING OF ',
       MAX_MESSAGE_LENGTH,' CHARACTERS');
  WRITELN('                      OR PRESS <ENTER> TO CONTINUE ');
  WRITELN;
  READLN(TEMP_NEW_ON);
  IF LENGTH(TEMP_NEW_ON) > 0 THEN
    BEGIN
      FOR SUBSCRIPT := 1 TO MAX_MESSAGE_LENGTH DO
        SIGNON_MESSAGE[SUBSCRIPT] := TEMP_NEW_ON[SUBSCRIPT];
      FOR SUBSCRIPT := (LENGTH(TEMP_NEW_ON) + 1)
             TO MAX_MESSAGE_LENGTH DO
             SIGNON_MESSAGE[SUBSCRIPT] := CHR(BLANK_CHR);
    END;
END;

(****************************)
PROCEDURE GET_OFF_MESSAGE;
(****************************)
(* ACCEPT A NEW SIGN OFF MESSAGE OR A CARRIAGE RETURN
INDICATING NO CHANGE *)

VAR
  SUBSCRIPT:INTEGER;
  TEMP_NEW_OFF:STRING[MAX_MESSAGE_LENGTH];
  CONSOLE:TEXT;
BEGIN
  WRITELN;
  WRITELN('                THE CURRENT SIGNOFF MESSAGE IS ');
  WRITELN('                ',CHR(14),SIGNOFF_MESSAGE,CHR(15));
  WRITELN('                ENTER A NEW STRING OF ',
              MAX_MESSAGE_LENGTH,' CHARACTERS');
  WRITELN('                     OR PRESS <ENTER> TO CONTINUE ');
  WRITELN;
```

```
(* GET OFF_MESSAGE CONT'D *)

   READLN(TEMP_NEW_OFF);
   IF LENGTH(TEMP_NEW_OFF) > 0 THEN
      BEGIN
         FOR SUBSCRIPT := 1 TO MAX_MESSAGE_LENGTH DO
            SIGNOFF_MESSAGE[SUBSCRIPT] := TEMP_NEW_OFF[SUBSCRIPT];
         FOR SUBSCRIPT := (LENGTH(TEMP_NEW_OFF) + 1)
                  TO MAX_MESSAGE_LENGTH DO
                  SIGNOFF_MESSAGE[SUBSCRIPT] := CHR(BLANK_CHR);
      END;
END;

(********************************)
PROCEDURE CHANGE_PARAMETERS;
(********************************)
(* THIS PROCEDURE CALLS A ROUTINE TO GET THE MASK FOR EACH
 PARAMETER AND THEN ASSEMBLES THE MASK FOR EACH REGISTER *)

VAR
   D_MASK,RATE_MASK,PARITY_MASK,LENGTH_MASK,BIT_MASK:INTEGER;
BEGIN
   GET_BAUD_RATE(RATE_MASK);
   E_REGISTER := RATE_MASK;

   GET_PARITY(PARITY_MASK);
   C_REGISTER := PARITY_MASK;

   GET_LENGTH(LENGTH_MASK);
   GET_BITS(BIT_MASK);
   D_MASK := 130;          (* SETS DTR AND RTS HIGH FOR REQUESTS *)
   D_REGISTER := (BIT_MASK | LENGTH_MASK | D_MASK);

   GET_CURSOR_CHAR;
   GET_PAD_CHARACTERS;
   GET_ON_MESSAGE;
   GET_OFF_MESSAGE;

END;
```

```
(***********************************)
PROCEDURE SAVE_COMM_PARAMETERS;
(***********************************)
(* THIS PROCEDURE WRITES THE VALUES OF COMMUNICATIONS
 PARAMETERS TO FILE "TERM.DAT" *)

VAR
  PARAMETER_RECORD:PARAM_RECORD;
  CURRENT_PARAMS_FILE:PARAM_FILE;
  CLOSE_CODE:INTEGER;
  ENTER_PRESS:CHAR;
  GOOD_OPEN,GOOD_IO:BOOLEAN;
BEGIN
  WITH PARAMETER_RECORD DO
    BEGIN
      ON_MESSAGE := SIGNON_MESSAGE;
      OFF_MESSAGE := SIGNOFF_MESSAGE;
      ACTIV_FLAG := ACTIVE_FLAG_SET;
      PARITY_PARAM := PARITY;
      BAUD_PARAM := BAUD_RATE;
      WORD_LNGTH := WORD_LENGTH;
      STP_BITS := STOP_BITS;
      C_PATTERN := C_REGISTER;
      D_PATTERN := D_REGISTER;
      E_PATTERN := E_REGISTER;
      CURSOR_CHAR := HOST_CURSOR;
      PAD_CHARACTERS := NR_OF_PADS;
    END;

WRITELN('C,D,E ',C_REGISTER,D_REGISTER,E_REGISTER);

  ASSIGN(CURRENT_PARAMS_FILE,'A:TERM.DAT');
  REWRITE(CURRENT_PARAMS_FILE);
  GOOD_OPEN := (IORESULT <>255);
  IF GOOD_OPEN THEN
    BEGIN
      CURRENT_PARAMS_FILE^ := PARAMETER_RECORD;
      PUT(CURRENT_PARAMS_FIEL);
      GOOD_IO := (IORESULT = 0);
      IF NOT GOOD_IO THEN
        BEGIN
          WRITELN(' **** ERROR - BAD WRITE TO TERM.DAT ****');
          WRITELN(' PRESS <ENTER> TO CONTINUE ');
          READ(ENTER_PRESS);
        END;
```

```
(* SAVE_COMM_PARAMETERS CONT'D *)

         CLOSE(CURRENT_PARAMS_FILE,CLOSE_CODE);
         IF CLOSE_CODE = 255 THEN
            BEGIN
               WRITELN(' **** ERROR - CANNOT CLOSE TERM.DAT ****');
               WRITELN(' PRESS <ENTER> TO CONTINUE ');
               READ(ENTER_PRESS);
            END;
      END
   ELSE
      WRITELN(' **** ERROR-- UNABLE TO OPEN TERM.DAT ****');
END;

(*********************************************************)
PROCEDURE GET_COMM_PARAMETERS(VAR GOOD_IO:BOOLEAN);
(*********************************************************)
(* THIS PROCEDURE READS THE COMMUNICATIONS PARAMETER RECORD
 FROM FILE "TERM.DAT" AND ASSIGNS THE VALUES TO PROGRAM
 VARIABLES. *)

VAR
   PARAMETER_RECORD:PARAM_RECORD;
   IN_FILE:PARAM_FILE;
   CLOSE_CODE:INTEGER;
   GOOD_OPEN:BOOLEAN;
BEGIN
   ASSIGN(IN_FILE,'A:TERM.DAT');
   RESET(IN_FILE);
   GOOD_OPEN := IORESULT <> 255;
   IF GOOD_OPEN THEN
      BEGIN
         READ(IN_FILE,PARAMETER_RECORD);
         GOOD_IO := IO_RESULT = 0;
```

```
(* GET_COMM_PARAMETERS CONT'D *)

        IF GOOD_IO THEN
           WITH PARAMETER_RECORD DO
              BEGIN
                 SIGNON_MESSAGE := ON_MESSAGE;
                 SIGNOFF_MESSAGE := OFF_MESSAGE;
                 ACTIVE_FLAG_SET := ACTIV_FLAG;
                 PARITY := PARITY_PARAM;
                 BAUD_RATE := BAUD_PARAM;
                 WORD_LENGTH := WORD_LNGTH;
                 STOP_BITS := STP_BITS;
                 C_REGISTER := C_PATTERN;
                 D_REGISTER := D_PATTERN;
                 E_REGISTER := E_PATTERN;
                 HOST_CURSOR := CURSOR_CHAR;
                 NR_OF_PADS := PAD_CHARACTERS;
             END
         ELSE
            BEGIN
               WRITELN(' **** ERROR - UNABLE TO READ TERM.DAT');
               WRITELN(' PRESS <ENTER> TO CONTINUE ');
               READ(ENTER_PRESS);
            END;
     END
  ELSE
     BEGIN
        WRITELN(' **** ERROR - UNABLE TO OPEN TERM.DAT ****');
        WRITELN('      PRESS ENTER TO CONTINUE ');
        READ(ENTER_PRESS);
     END;
END;
```

```
(************************)
BEGIN (* MAIN PROGRAM *)
(************************)
(* THIS DRIVER READS THE PREVIOUSLY SAVED COMMUNICATIONS
 PARAMETERS, DISPLAYS THEIR CURRENT VALUES, EXECUTES
 A CHANGE PROCEDURE, BY CALLING CHANGE_PARAMETERS, FOR
 EACH PARAMETER AND THEN SAVES THE NEW VALUES. *)

GET_COMM_PARAMETERS(GOOD_READ);  (* GET THE CURRRENT VALUES *)
IF GOOD_READ THEN
   BEGIN
      DISPLAY_PARAMETERS;
      CHANGE_PARAMETERS;
      SAVE_COMM_PARAMETERS;
   END;
(* RETURN TO PROGRAM TERM *)
ASSIGN(CHAIN_FILE,'A:TERM.COM');
RESET(CHAIN_FILE);
IF IORESULT = 255 THEN (* ERROR *)
   BEGIN
      WRITELN(' **** ERROR - UNABLE TO OPEN TERM.COM ****');
      WRITELN(' PRESS <ENTER> TO CONTINUE ');
      READ(ENTER_PRESS);
   END
ELSE
   CHAIN(CHAIN_FILE); (* RETURN TO PROGRAM TERM *)
END.
```

AN INTER-COMPUTER COMMUNICATIONS SYSTEM FOR A PERSONAL COMPUTER

by

DANIEL RAY VESTAL

B.S., Cameron University, 1973

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

An Inter-Computer Communications System for a Personal Computer

by

Daniel R. Vestal

ABSTRACT

Although the owner of a home computer now has many capabilities that were available only on expensive main frame computer systems several years ago, there may still be many times when it is necessary for him to use a larger computer system. A personal computer may be used as a remote terminal to access a larger host computer by connecting an acoustically coupled modem between a standard telephone and the home computer, and then executing a simple terminal program. Many of the terminal programs being marketed to allow the personal computer to act as a remote terminal perform only those functions normally handled by hardware in a standard terminal. A personal computer can easily be programmed to perform the functions of a hardware terminal; however, it also has other capabilities that can be used to offset the limitations of using a remote terminal.

This project identifies and implements a set of functions that make effective use of a home computer's capabilities when it is used as a remote terminal, especially the personal computer's capabilties to internally manipulate and store data as local files. This prototype and the enhancements that are proposed are intended to serve as a model for others in developing similar systems.