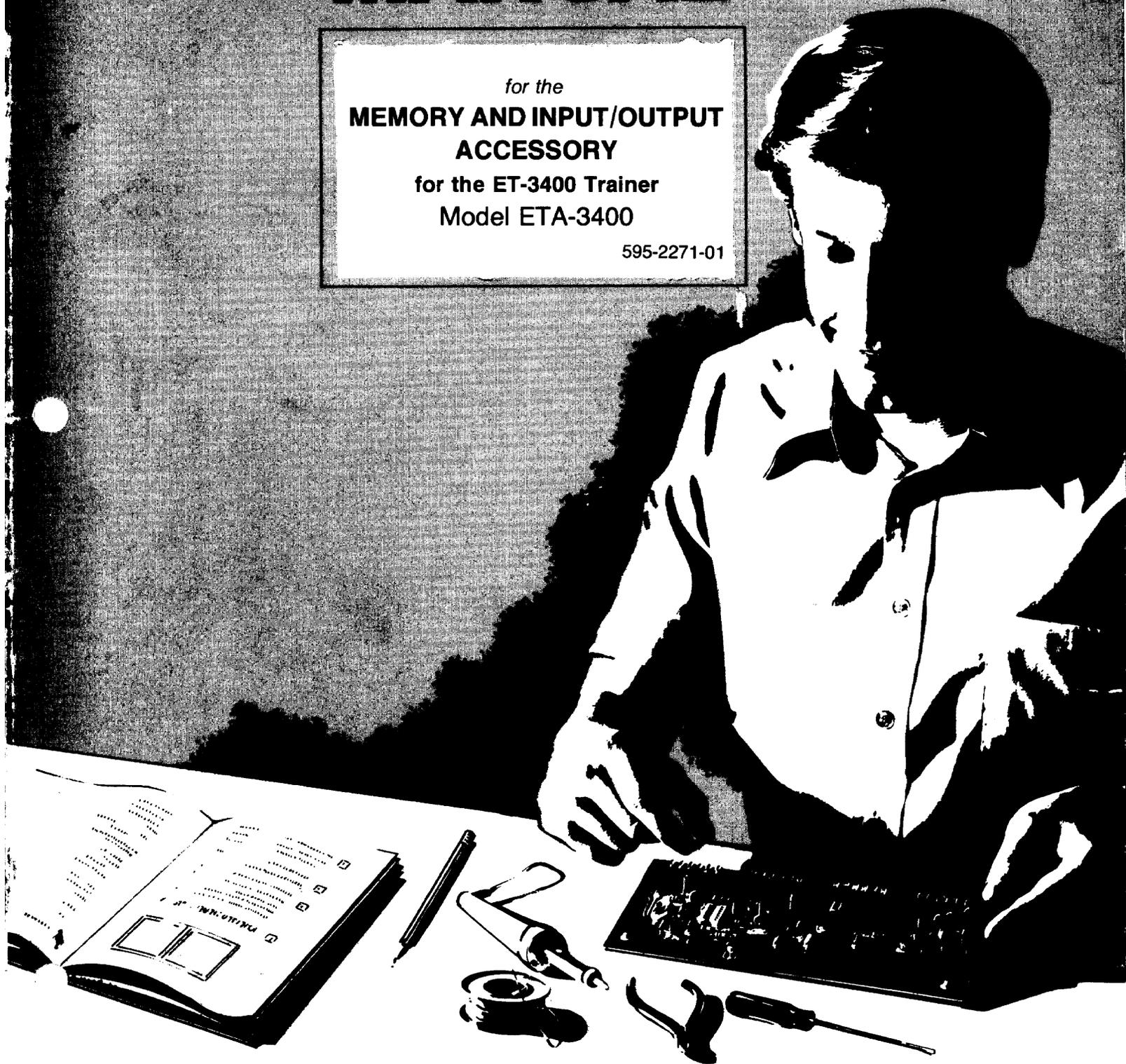


# HEATHKIT<sup>®</sup> MANUAL

*for the*  
**MEMORY AND INPUT/OUTPUT  
ACCESSORY**  
for the ET-3400 Trainer  
Model ETA-3400

595-2271-01



HEATH COMPANY • BENTON HARBOR, MICHIGAN

The following telephone numbers are direct lines to the departments listed:

Kit orders and delivery information ..... (616) 982-3411  
 Credit ..... (616) 982-3561  
 Replacement Parts ..... (616) 982-3571

*Technical Assistance Phone Numbers*

*8:00 A.M. to 12 P.M. and 1:00 P.M. to 4:30 P.M., EST, Weekdays Only*  
 R / C, Audio, and Electronic Organs ..... (616) 982-3310  
 Amateur Radio ..... (616) 982-3296  
 Test Equipment, Weather Instruments and  
 Home Clocks ..... (616) 982-3315  
 Television ..... (616) 982-3307  
 Aircraft, Marine, Security, Scanners, Automotive,  
 Appliances and General Products ..... (616) 982-3496  
 Computer Hardware ..... (616) 982-3309  
 Computer Software ..... (616) 982-3860  
 Heath Craft Wood Works ..... (616) 982-3423

## YOUR HEATHKIT 90-DAY LIMITED WARRANTY

### Consumer Protection Plan for Heathkit Consumer Products

Welcome to the Heath family. We believe you will enjoy assembling your kit and will be pleased with its performance. Please read this Consumer Protection Plan carefully. It is a "LIMITED WARRANTY" as defined in the U.S. Consumer Product Warranty and Federal Trade Commission Improvement Act. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

#### Heath's Responsibility

**PARTS** — Replacements for factory defective parts will be supplied free for 90 days from date of purchase. Replacement parts are warranted for the remaining portion of the original warranty period. You can obtain warranty parts direct from Heath Company by writing or telephoning us at (616) 982-3571. And we will pay shipping charges to get those parts to you . . . anywhere in the world.

**SERVICE LABOR** — For a period of 90 days from the date of purchase, any malfunction caused by defective parts or error in design will be corrected at no charge to you. You must deliver the unit at your expense to the Heath factory, any Heathkit Electronic Center (units of Veritechnology Electronics Corporation), or any of our authorized overseas distributors.

**TECHNICAL CONSULTATION** — You will receive free consultation on any problem you might encounter in the assembly or use of your Heathkit product. Just drop us a line or give us a call. Sorry, we cannot accept collect calls.

**NOT COVERED** — The correction of assembly errors, adjustments, calibration, and damage due to misuse, abuse, or negligence are not covered by the warranty. Use of corrosive solder and/or the unauthorized modification of the product or of any furnished component will void this warranty in its entirety. This warranty does not include reimbursement for inconvenience, loss of use, customer assembly, set-up time, or unauthorized service.

This warranty covers only Heath products and is not extended to other equipment or components that a customer uses in conjunction with our products.

SUCH REPAIR AND REPLACEMENT SHALL BE THE SOLE REMEDY OF THE CUSTOMER AND THERE SHALL BE NO LIABILITY ON THE PART OF HEATH FOR ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOSS OF BUSINESS OR PROFITS, WHETHER OR NOT FORSEEABLE.

Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

#### Owner's Responsibility

**EFFECTIVE WARRANTY DATE** — Warranty begins on the date of first consumer purchase. You must supply a copy of your proof of purchase when you request warranty service or parts.

**ASSEMBLY** — Before seeking warranty service, you should complete the assembly by carefully following the manual instructions. Heathkit service agencies cannot complete assembly and adjustments that are customer's responsibility.

**ACCESSORY EQUIPMENT** — Performance malfunctions involving other non-Heath accessory equipment, (antennas, audio components, computer peripherals and software, etc.) are not covered by this warranty and are the owner's responsibility.

**SHIPPING UNITS** — Follow the packing instructions published in the assembly manuals. Damage due to inadequate packing cannot be repaired under warranty.

If you are not satisfied with our service (warranty or otherwise) or our products, write directly to our Director of Customer Service, Heath Company, Benton Harbor MI 49022. He will make certain your problems receive immediate, personal attention.

# SOFTWARE REFERENCE MANUAL

*for the*  
**MEMORY AND INPUT/OUTPUT  
ACCESSORY**

**for the ET-3400 Trainer  
Model ETA-3400**

595-2271-01

**HEATH COMPANY  
BENTON HARBOR, MICHIGAN 49022**

**Copyright © 1979**  
Heath Company  
*All Rights Reserved*  
Printed in the United States of America

## TABLE OF CONTENTS

Introduction .....	3
Heath/Wintek Fantom II Monitor .....	4
Symbols .....	5
Using the Monitor .....	6
Display/Alter Register Contents .....	7
Display/Alter Memory Contents .....	9
Display Program Instructions .....	11
Block Memory Transfer .....	12
Program Execution Control .....	13
Program Storage and Retrieval .....	18
Using a Teletypewriter .....	20
Sample Program .....	22
Monitor Command Summary .....	23
Heath/Pittman Tiny BASIC .....	26
Editing Commands .....	27
Using Tiny BASIC .....	28
Modes of Operation .....	29
Instructions .....	30
Mathematical Expressions .....	32
Tiny BASIC Re-Initialization (Warm Start) .....	33
Functions .....	34
Sample USR Programs .....	36
Appendixes .....	40
Appendix A — Memory Map .....	40
Appendix B — Tiny BASIC Error Message Summary .....	41
Appendix C — Heath/Wintek Monitor Listing .....	43
Appendix D — Excerpts from “Kilobaud” .....	75
INDEX .....	95

# INTRODUCTION

This Manual describes the operation of your ET-3400/ETA-3400 microcomputer system. The major operational features of the system are explained in the sections titled "Heath/Wintek FANTOM II Monitor" and "Heath/Pittman Tiny BASIC." The keyboard commands, "Monitor Listing," sample programs, and memory maps are also included, as well as several article reprints from "Kilobaud" magazine that will help you more fully enjoy your ET-3400/ETA-3400 Microcomputer System.

The Microcomputer system easily interfaces to a video terminal and a cassette recorder. The increase in memory size and software support gives you a more flexible, general-purpose computer system, while the trainer itself still remains functional and useful. The following list summarizes the main features.

- The ETA-3400 uses an independent power supply.
- The system supports 1024 (1K) bytes of read/write random-access memory. This is expandable to 4K.
- A 2K ROM MONITOR.
- A 2K ROM Tiny BASIC interpreter.
- Expanded I/O support:
  - Audio cassette mass storage;
  - Video terminal.

## HEATH/WINTEK FANTOM II MONITOR

This Monitor consists of a group of individual computer programs linked together that operate as a single supervisory systems controller. These programs are permanently located in a 2K ROM (2048 bytes of Read-Only-Memory) on the ETA-3400 circuit board. FANTOM II schedules and verifies the operation of peripheral computer components. You use the Monitor to build, test, execute, store, and retrieve computer programs written in machine code.

The Monitor provides you with a means of communicating between the microprocessor, the terminal, and a cassette. You select a Monitor command by pressing a key on the console terminal associated with the particular command. This information is processed by the Monitor, which then directs the computer to the routine that performs the operation. Control is returned to the Monitor after the operation is completed.

This section of the Manual describes the function, operation and features of FANTOM II. Some of the major features are:

- Display/Alter register contents.
- Display/Alter memory contents.
- Display Program Instructions
- Program Execution Control.
- Program Storage and Retrieval.

NOTE: A knowledge of the Motorola 6800 microprocessor and common programming techniques is essential for understanding the FANTOM II Monitor. The HEATH EE-3401 microprocessor course provides this knowledge.

## SYMBOLS

This Manual uses symbols to describe some terms. Frequently used symbols and their meaning are listed below. In examples of keyboard dialogue, monitor and program output are underlined>.

### MICROPROCESSOR

- A Accumulator or register A. The 8-bit arithmetical or logical section of the computer that processes data.
- B Accumulator or register B. An 8-bit register similar to register A.
- C The condition code register. A 6-bit register that indicates the nature or result of an instruction.
- P The program counter. A 16-bit register that sequentially counts each program instruction.
- S The stack pointer. A 16-bit register that records the last address of an entry onto the stack.
- X The index register. This 16-bit register permits automatic program modification of an instruction address without destroying the address contained in memory. The index register is frequently used as a memory pointer.

### TERMINAL

- ESC The ESCape key. Press this key to return control to the Monitor.
- BRK The BReaK key. Press this key once to return control to the Monitor. Press it twice to return control to the ET-3400 trainer.
- CTRL The control key. When it is used in conjunction with another key, it creates a special function. For instance, if you hold CTRL and press P, the contents of the program counter will be displayed.
- ␣ The carriage return, or return key, on your video terminal.

### PROMPT CHARACTERS

- MON> The FANTOM II Monitor prompt character. It indicates that your system is functioning and ready to accept a Command.
- :
- Tiny BASIC prompt character.

## USING THE MONITOR

### POWER UP and MASTER RESET

When power is first applied to the ET-3400/ETA-3400 Microcomputer System, you should press the RESET key on the ET-3400 keypad. The display will then show CPU UP, and the next keypad entry will be interpreted as a command. Use the RESET key to initialize the system or escape from a malfunctioning program.

When you wish to use FANTOM II, after pressing the RESET key, press the DO (D) key on your trainer and enter the hexadecimal starting address 1400. This command causes FANTOM II to print the prompt characters (MON> )\* on the video terminal. This tells you that the system Monitor is functioning and is waiting for a command. For instance, the following sequence will initialize the Monitor, examine the contents of several memory locations, and return control to the ET-3400 microcomputer.

- Apply power to the microcomputer system.
- Press RESET on the ET-3400 keypad.
- Press DO on the keypad and enter hexadecimal address 1400.
- Look for the prompt character ( MON> ) on your terminal.
- Type M ( Memory ) on the terminal keyboard and enter the address 1400 followed by a carriage return.
- The video display responds by printing the address and the memory contents. (1400 0F)
- Enter several carriage returns and observe the display. You will notice that, for each carriage return, a sequential memory location and its corresponding data is shown.
- Press the ESCape or BReaK key on your terminal. The prompt character reappears and control is returned to the monitor.
- Press the BReaK key a second time and control is returned to the Trainer.

\*Throughout this Manual, the computer output has been underlined to set it off from the user response.

## DISPLAY/ALTER REGISTER CONTENTS

### DISPLAY REGISTERS

The ET-3400/ETA-3400 Microcomputer System manipulates all data through its registers. You can examine the contents of a single register or all the registers by selecting the appropriate command. When you use the correct format, displaying the contents of a selected register is simple. For instance, pressing R after the prompt character displays the contents of all microprocessor registers. In this and subsequent examples, unless specified, the data shown is only given as an example. You should expect to get different displays.

```
MON> R C=DB B=0B A=0B X=0B0B P=1401 S=00D2 CE 1000  
MON>
```

In this example, you can see that the condition code register was set to hexadecimal integer DB. The A and B registers equal 0B, while the index register X was set to 0B0B. The program counter (P) displays the address of the next instruction to be executed and S is the current address of the stack pointer. Finally, the next instruction that would be executed if the program were run is CE 1000. This information, when displayed on the video screen, is useful for correcting program errors.

The two most significant bits of the 8-bit RAM location that hold the condition code are neglected by the system hardware. In the example, DB (1101 1011) shows the status of the condition codes. By pressing CTRL/C and entering a different value, you can change the status of register C.

### DISPLAY/ALTER REGISTERS

The Monitor also lets you display or change the contents of individual registers, except the stack pointer. To display the contents of a register (other than the stack pointer), press the CTRL key on the terminal, and then select and press the key that corresponds to the register name. When you wish to change the contents of a register, enter the new value after displaying the original contents. The following examples show you how to display and alter the contents of each microprocessor register.

For instance, to display the program counter, simultaneously press the CTRL and the P keys. A return causes the Monitor to complete the command and display the prompts.

```
MON> CTRL/P P=1401 Ⓢ  
MON>
```

In the next example, the contents of register A are first displayed and then altered. Press CTRL/A to display the current contents of register A. Enter a new hexadecimal value, for instance 1B, and a carriage return. The return signals the Monitor to execute the command, and the displayed prompt character indicates a successful completion of the command. You can then press CTRL/A and verify that the register contents were changed.

```

MON> CTRL/A A=NN 1B Ⓢ
MON> CTRL/A A=1B Ⓢ
MON>

```

The Monitor uses the same format to display or alter the contents of each microprocessor register. In all subsequent examples, NN or NNNN represents a random hexadecimal value. The list summarizes the usage of register commands available to you through the Monitor.

```

MON> CTRL/A A=1B Ⓢ      (Display A)
MON> CTRL/B B=NN 12 Ⓢ   (Alter B to read 12)
MON> CTRL/C C=NN 00 Ⓢ   (Alter C to read 00)
MON> CTRL/P P=NNNN 1234 Ⓢ ( P = 1234 )
MON> CTRL/X X=NNNN 5678 Ⓢ ( X = 5678 )
MON> R C=00 B=12 A=1B X=5678 P=1234 S=NNNN *
MON>

```

\*You can neither alter the stack pointer, nor predict its value, with the FANTOM II Monitor. Also, machine instructions or data will be output after the stack pointer address is printed.

## DISPLAY/ALTER MEMORY CONTENTS

### DISPLAY MEMORY

The FANTOM II Monitor can access individual or sequential memory locations. This feature allows you to rapidly examine and correct program instructions or data. To display an area of memory on the video terminal, type D (display) and specify the range of the memory locations. The following example shows you how to display the contents of 16 sequential memory cells from address 1400 thru 140F. Because the area shown in the example is part of the Monitor, you should obtain the same results.

```
MON> D 1400,140F Ⓢ  
1400 OF CE 10 00 6F 01 6F 03 86 01 A7 00 86 7F A7 02  
MON>
```

The Monitor responds to the carriage return by typing the starting address and listing the memory contents. The address of each line displayed is always the first four-digit number, followed by the contents of the next sixteen sequential memory locations.

### DISPLAY/ALTER MEMORY

Use the M (Memory) command when you wish to examine or alter the contents of an individual or a sequence of memory locations. For instance, as shown below, type an M after the prompt character and the address 1400. FANTOM II responds by printing the address and the memory contents (OF) after you press the carriage return. To proceed to the next location, press the carriage return again. FANTOM II responds by printing an address and its contents. To exit the display mode and return to the Monitor, press ESC or BRK.

The following example shows you how to examine the contents of ROM memory locations. You can compare the data with the "Heath/Wintek Monitor Listing," ("Appendix C," Page 37) and/or examine additional locations. This feature provides a quick method of searching for useful Monitor or Tiny BASIC sub-routines.

```
MON> M 1400 Ⓢ  
1400 OF Ⓢ  
1401 CE Ⓢ  
1402 10 Ⓢ  
1403 00 Ⓢ  
1404 6F ESC  
MON>
```

You may use the same procedure to modify memory contents that you use to change register contents. In the next example, use the M command to alter the contents of several hexadecimal locations between 100 and 105. The procedure always gives you an option of changing or not changing the program data. You will not alter memory contents if you press a carriage return after the data is displayed.

```
MON> M 100 Ⓢ  
0100 NN A Ⓢ  
0101 NN 0B Ⓢ  
0102 NN C Ⓢ  
0103 NN 0D Ⓢ  
0104 NN E Ⓢ  
0105 NN BRK  
MON>
```

The previous example features free-format hexadecimal input. This means you do not have to enter leading zeros. For example, at location 0104 we entered the value E rather than 0E. Free-format allows you to correct or modify a bad entry simply by typing extra digits. For instance, assume that, in the previous example, you incorrectly entered 109 after the M command. Enter the address 0100 before the carriage return to correct the mistake. For example:

```
MON> M 1090100 Ⓢ  
0100 NN ESC  
MON>
```

Since a maximum of four digits is all that are needed for an address, only the last four are retained. Similarly, if only two digits are expected, then only two will be retained.

## DISPLAY PROGRAM INSTRUCTIONS

The FANTOM II Monitor offers an important extra feature. You may use the Instruction (I) command to display program instructions. The format is similar to the memory display instruction except that the Monitor prints a single micro-processor instruction per line rather than the contents of each memory cell. An instruction can be one, two, or three bytes. A carriage return, as with the M command, causes FANTOM II to display the next sequential instruction. The I command allows data changes using the same procedure as the M command. However, only the last byte of an instruction can be altered.

The next example displays the first four Monitor program instructions.

```
MON> I 1400 Ⓢ  
1400 0F Ⓢ  
1401 CE 1000 Ⓢ  
1404 6F 01 Ⓢ  
1406 6F 03 BRK  
MON>
```

When the data in the first byte of an instruction address memory location is not a machine instruction, the Monitor prints a DATA=NN message. The next instruction following the DATA=NN statement is printed after the carriage return. For instance, the command sequence:

```
MON> I 1A0D Ⓢ  
1A0D DATA=45 Ⓢ  
1A0E DATA=15 Ⓢ  
1A0F 39 ESC  
MON>
```

produces the DATA = NN message until the Monitor encounters a valid machine instruction. In this example, the Monitor recognizes the integer (39<sub>H</sub>) as a machine instruction.

## BLOCK MEMORY TRANSFER

The Monitor features a command that allows you to move the contents of a block of memory from one location to another. The SLIDE memory command simply copies one section of memory to another.

To use the SLIDE memory command, you must determine the parameters of the block of memory to be moved. These parameters include a hexadecimal starting address of both the source and destination of the memory block to be moved. In addition, a hexadecimal count of the number of memory cells to be transferred is also required. Press and hold the CTRL key on the keyboard while pressing the S key to initiate the SLIDE command after you determine the program parameters. FANTOM II prompts you with the keyword SLIDE. You respond to this keyword by typing the starting address of the origin and destination, followed by the count and a carriage return.

The SLIDE command in the next example transfers thirty-two (decimal) bytes of data from ROM into low memory. The starting address of data to be moved is 1400 and the data will be moved to an area of memory starting at location 200. The display (D) command only verifies the data manipulation before and after the SLIDE command is executed.

```

MON> D 200,21F ☞
0200 NN NN
0210 NN NN
MON> D 1400,141F ☞
1400 0F CE 10 00 6F 01 6F 03 86 01 A7 00 86 7F A7 02
1410 C6 04 E7 01 E7 03 A7 00 09 A6 00 63 00 43 01 00
MON> CTRL/S SLIDE 1400,200,20 ☞
MON> D 200,21F ☞
0200 0F CE 10 00 6F 01 6F 03 86 01 A7 00 86 7F A7 02
0210 C6 04 E7 01 E7 03 A7 00 09 A6 00 63 00 43 A1 00

```

## PROGRAM EXECUTION CONTROL

FANTOM II gives you two options when you execute a machine language program. With the first option, you execute the complete program by entering the GO (G) command and a starting address. The second option allows you to execute a program segment with the S or E command. It is primarily used for detecting errors in program logic.

### EXECUTING A PROGRAM

The ETA-3400 Microcomputer Accessory contains a machine language program (Tiny BASIC). We will use this routine to show program execution with the GO command, G. The G command and a program starting address causes the system to fetch the operational code in the memory location specified. Program execution begins from this location and continues until your program returns control to the FANTOM II Monitor, or the RESET key is pressed on the ET-3400. To run Tiny BASIC, enter:

```
MON> G 1000 Ⓞ  
HTB1 G 1000  
:10 REM HTB1 IS PRINTED OVER MON> Ⓞ  
:20 PRINT "HEATH TINY BASIC IS RUNNING" Ⓞ  
:30 END Ⓞ  
:RUN Ⓞ  
HEATH TINY BASIC IS RUNNING  
  
:BYE Ⓞ  
MON>
```

NOTE: Tiny BASIC writes over the MON> prompt with the HTB1 letters and then issues a carriage return. The prompt character (:) signifies that Tiny BASIC is in the command mode and waiting for an instruction.

Using the Tiny BASIC firmware is only one example of program execution. For another example, you should enter the program shown at the top of Page 14 using the M command. This routine prints a message on your video terminal. The format is similar to the listing printed in "Appendix C," and it illustrates a format that you might encounter in some computer magazines. The JSR (Jump to SubRoutine) mnemonic at hexadecimal location 100 is translated to machine code instructions BD 1618. BD is the machine equivalent of JSR and 1618 is the starting address of a Monitor subroutine that prints a character string. Likewise, FCB is a pseudo-mnemonic that reserves a block of memory for your character string (i.e. the message).

0100	BD	1618	MSG JSR	OUTIS		; OUTPUT CHARACTERS
0103	ODDA48		FCB	OD,DA,48		; INSERT ASCII MSG.
0106	454C4C		FCB	45,4C,4C		; CR,LF,HELLO,0
0109	4F00		FCB	4F,00		
010B	BD	1400	JSR	MAIN		; RETURN TO MONITOR

Machine language program to print a message on your video terminal.

The following operational sequence uses the Monitor to enter the machine code, check the accuracy of the instructions, and execute the program.

```

MON> M 100 Ⓜ      (... Enter machine code...)
0100 NN BD Ⓜ      (... JSR ...)
0101 NN 16 Ⓜ      (... High byte address ...)
0102 NN 18 Ⓜ      (... Low byte address ...)
0103 NN 0D Ⓜ      (... Sequentially enter ...)
      .           data from the
      .           machine code
      .           until complete.
010D NN 00 Ⓜ      (... JSR MAIN.....)
010E NN ESC
MON>

```

The display instruction (I) lets you sequentially verify the accuracy of your work.

```

MON> I 100 Ⓜ
0100 BD 1618 Ⓜ
      .
      .
      .
010B BD 1400 ESC
MON>

```

The program is ready for execution. Use the Go (G) instruction to run your program from address 100.

```

MON> G 100 Ⓜ
MON>

```

The computer prints a friendly greeting on the display when you execute the program.

## WARNING

Always originate your programs at or above hexadecimal location 100 because Tiny BASIC and FANTOM II frequently use the low memory as a buffer. "Appendix A" contains a memory map of the RAM locations that the firmware uses.

## EXECUTING A PROGRAM SEGMENT

Isolating and correcting program errors is another function of program execution control. This function is commonly referred to as breakpointing. For a more complete discussion on breakpointing, refer to the operation section of the ET-3400 Microprocessor Trainer Manual. The Monitor supports breakpointing techniques by providing you with both single STEP (S) and multiple step EXECUTE (E) commands. A third technique lets you enter breakpoint addresses into a table and then use the GO command to execute a program segment.

Assume that, in the previous example, machine instruction BD 1618 was incorrectly entered to read BD 160D. The simple method to detect this error is to set the program counter to address 100 and step through each instruction, comparing the computer activity with the results expected from your algorithm.

The single STEP command requires that you define the initial program parameters and preset any registers to their initial status. For this example, only the program counter is affected and must be preset to the starting address of the program (i.e. 100). Use the command to display/alter the program counter to read hexadecimal integer 100. Type S after presetting the initial parameters to execute a single instruction. The Monitor responds by executing the instruction located at the program address contained in the program counter, and then printing the contents of each CPU register on the terminal.

```
MON> CTRL/P P=NNNN 100 Ⓞ  
MON> R C=NN B=NN A=NN X=NNNN P=0100 S=NNNN  
MON> S C=NN B=NN A=NN X=NNNN P=160D S=NNNN  
MON>
```

Analysis of the program data displayed on your terminal, when compared with the algorithm (i.e. see Chart 1), shows an incorrect address for the JSR mnemonic. Once the initial parameters have been defined, you may continuously single step through a program by typing S.

A better technique for debugging large programs is to use the EXECUTE (E) multiple step command. The EXECUTE command is similar to the STEP command, except control is returned to the Monitor only after a specified number of steps have been executed. The step count is a hexadecimal integer. For example, the following sequence would execute 18 program steps, and then display the registers in the same format as the STEP command.

```
MON> CTRL/P P=NNNN 100 Ⓢ  
MON> E 12 Ⓢ  
C=NN B=NN A=NN X=NNNN P=NNNN S=NNNN NN NN *  
MON>
```

Breakpointing is another technique for isolating errors in your program. A breakpoint in your program interrupts the normal program execution and lets you test or analyze program parameters. Type H to set a breakpoint (Haltpoint), followed by the address and a carriage return.

For instance,

```
MON> H 10B Ⓢ  
MON>
```

would set a breakpoint in the table that would halt your program at address 10B.

\*NOTE: Be extremely careful when you are using ROM subroutines and the S, E, and H commands. In this example, it is not possible to accurately predict the program results because the FANTOM II Monitor and the ET-3400 Monitor share RAM locations. Occasionally, this sharing causes unpredictable results.

When you wish to examine the status of the breakpoint table, simply type CTRL/H. This command displays the contents of the breakpoint table. The Monitor forbids the entering of additional breakpoints into the table until one of the entries is cleared. A cleared table entry is displayed as FFFF.

```
MON> CTRL/H 010B FFFF FFFF FFFF  
MON>
```

The only way to delete a breakpoint from the table is to use the CLEAR (C) command. To remove a breakpoint, type C and the address. For instance:

```
MON> C 10B Ⓢ  
MON>
```

would remove the breakpoint 10B from the table.

A maximum of four breakpoints (Haltpoints) is permissible in the table. An attempt to set more than four breakpoints would return the following message:

```
ERROR!
```

Always place a haltpoint at a RAM location containing an operation code. Use the G command to execute the program until the haltpoint is reached. After it encounters a haltpoint address, the Monitor prints the current status of the microprocessor registers. You may examine or alter the contents of memory or registers before proceeding with program execution.

## PROGRAM STORAGE AND RETRIEVAL

The ETA-3400 Microcomputer Accessory lets you choose either of two different methods for controlling a cassette magnetic tape recorder. The simpler method allows you to use a recorder and the ET-3400 keypad. The other method lets you use a recorder and console terminal to store data. The advantage to the second method is the optional increase in speed with which you can LOAD or DUMP your routine. Either method lets you create and use an inexpensive library of computer routines. The information you store on cassette tape uses the Kansas City Standard (KCS) format with a five second leader and trailer.

The method you choose to LOAD or DUMP a magnetic tape is optional. However, using a console lets you select different baud rates to transfer data between cassette tape and computer memory. A baud rate is the measure (bits per second) of the speed of transmission of data pulses. We recommend that you use 300 baud. The important thing about baud rates is that they be the same for each device when you are reading or writing information between devices. For your convenience, always write the baud rate on the cassette label next to the program name.

### CASSETTE USAGE WITH A CONSOLE TERMINAL

To use the Tape (T) command, press CTRL/T after the Monitor prompt character. This command causes the terminal to print a T after which you specify the baud rate\* (1 to 8). A colon (:) separates the baud rate from the program starting address, and a comma is used between the starting and ending address of the memory block to be recorded. Prepare the cassette by installing and rewinding a tape before typing a carriage return. Always allow the recorder to attain a normal operating speed by waiting several seconds before hitting the return key. For instance, assume you wish to save sample program number one on Page (22).

```
MON> CTRL/T T1:100,126 Ⓢ  
MON>
```

This command writes the data from memory locations 100 through 126 to cassette tape at 2400 baud. When the data is completely written, program control is returned to the Monitor and the FANTOM II prompt character reappears. To specify 300 baud, type 8 rather than 1.

\*Any integer can be used to specify a baud rate. However, the common rates use: 300 for T8; 600 for T4; 1200 for T2; and 2400 for T1.

Because 300 baud is the recommended rate, the Monitor lets you select and type T rather than CTRL/T when writing data. With this feature, you may standardize all your tapes at 300 baud and, in so doing, be able to use either the keypad or the terminal to LOAD your tapes. For example, the following two commands are equivalent:

```
MON> CTRL/T T8:100,126 Ⓢ  
      or  
MON> T 100,126 Ⓢ
```

The LOAD (L) command allows you to read data from a cassette tape into memory. The baud rate with which the tape was written must agree with the baud rate at which you wish to read the data. If the baud rates do not agree or you find a tape error, possibly due to dirt on the recorder heads, a tape error message will be generated. To use the load command, type L followed by the integer code (1 to 8) that indicates the selected baud rate. For example:

```
MON> L 1 Ⓢ  
MON>
```

would load a tape written at 2400 baud. A tape written at 300 baud can be read by either an "L8" or "L" command.

### ET-3400 CASSETTE USAGE

You may use the ET-3400 keypad to save a block of memory on cassette tape. This routine prompts you for the first and last address of the memory block to be recorded. To execute the cassette dump routine from the keypad, use the DO function to transfer control to address 1A8F. The following two prompts are printed on the ET-3400 displays:

```
---- Fr.  
  
---- La.
```

You respond to the prompts by entering the first (Fr.) and last (La.) address of the block of memory to be saved on cassette tape. Before you enter the last digit, activate the cassette recorder by pressing the record button on the cassette. For instance, assume you wish to save sample program number one on Page 22.

- Press DO (D) on the ET-3400 keypad and enter address 1A8F.
- Enter the first address (0100) of the memory block to be transferred after the ---- Fr. prompt.

- Enter the first three digits of the last address (012) after the \_ \_ \_ \_ La. prompt.
- Install and rewind a magnetic tape. Then press the Record button. Be sure the leader passes the recording head.
- Enter the last digit (6) of the address. When the memory block is recorded, the ET-3400 displays will print CPU UP.

The ET-3400 cassette LOAD routine, located in the Monitor from address 1ABC through 1AD4, reads a block of memory data from cassette tape into computer memory. The routine proceeds until the last record is found or until a tape error occurs. An error can be caused by many diverse problems such as, dirt on the tape or tape heads, an incorrect baud rate, etc. If an error is found the ET-3400 display prints:

Error

If no error is found, the CPU UP message is printed after the data is completely loaded. Don't forget to turn off the recorder at this point. The following procedure transfers binary data from a cassette tape into computer memory:

- Press the DO (D) key on the trainer and enter the first three digits of the cassette loader routine, 1AB\_ .
- Install and rewind the cassette tape.
- Press the PLAY button on the recorder and enter the last digit (C) on the keypad.
- Wait for the message (CPU UP or Error) to be printed on the displays.

## USING A TELETYPEWRITER

Two commands let you Punch/List formatted absolute binary tapes using the Motorola MIKBUG\* format. The tape format is shown in Figure 1. When you want to load or store binary data from a teletypewriter, use the L or P monitor commands. For instance, to transfer binary data from a paper tape to memory, enter the following command from your console:

```
MON> L0
```

NOTE: Always activate the teletypewriter before you enter any monitor commands.

\*Registered Trademark, Motorola Inc.

To Print/Punch a formatted binary tape, enter the P command followed by a beginning and ending address. FANTOM II responds by outputting the data. The next example displays the sixteen bytes of memory from hexadecimal location 1400 to 140 F.

```

MON> P 1400,140F
S11314000FCE10006F016F03861A700867FA7022D
S9
MON>
    
```

Figure 1 is a breakdown of the Motorola MIKBUG\* format. Use the information only to decode programs stored in the MIKBUG\* format.

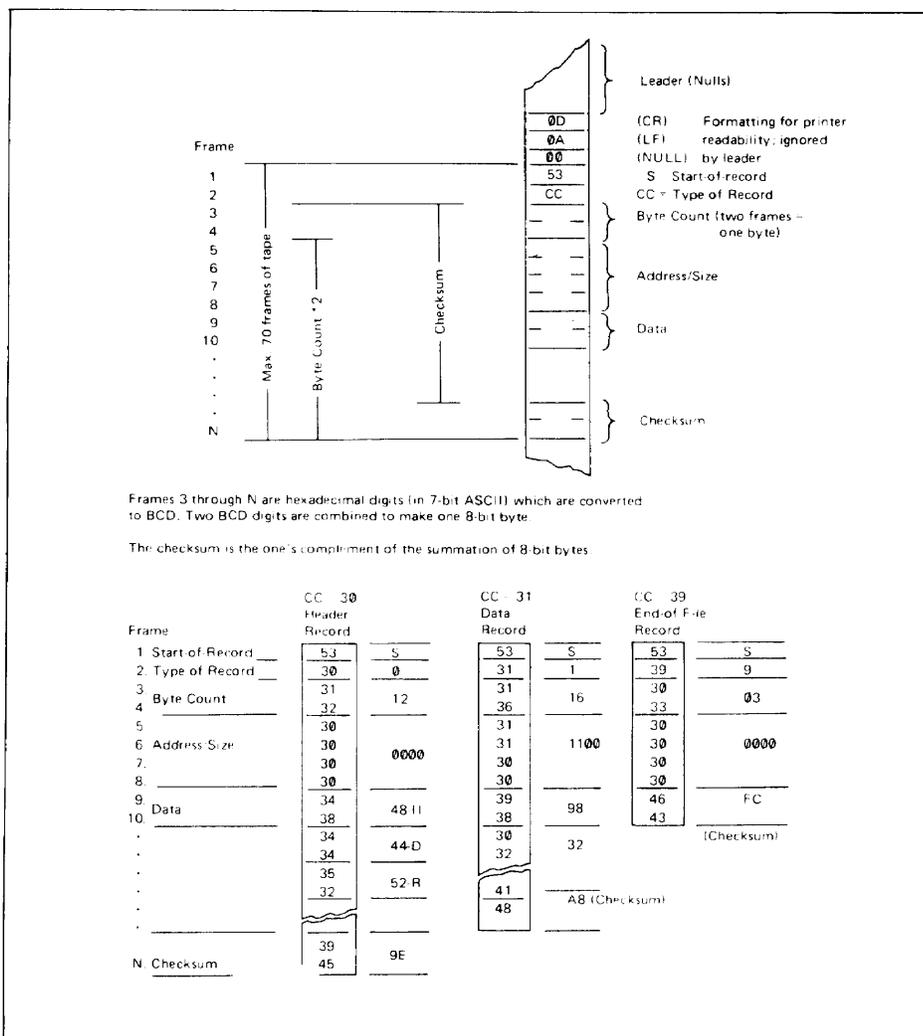


Figure 1  
Courtesy of Motorola Semiconductor Products Inc.

## A SAMPLE PROGRAM

The sample program provides you with a routine to test the operation of your ETA-3400 Microcomputer Accessory. You can use the routine to gain proficiency with the FANTOM II Monitor. The routine is a duplicate (with minor changes) of a program listed in the ET-3400 Manual.

```
0100 BD FCBC START JSR REDIS
0103 86 01 LDA A $01
0105 20 07 BRA OUT
0107 D6 F1 SAME LDA B DIGADD+1
0109 CB 10 ADD B $10
010B D7 F1 STA B DIGADD+1
010D 48 ASL A
010E BD FE3A OUT JSR OUTCH
0111 CE 2F00 LDX $2F00
0114 09 WAIT DEX
0115 26 FD BNE WAIT
0117 16 TAB
0118 5D TST B
0119 26 EC BNE SAME
011B 86 01 LDA A $01
011D DE F0 LDX DIGADD
011F 8C C10F CPX $C10F
0122 26 EA BNE OUT
0124 BD 1400 JSR MAIN
```

Use FANTOM II when you enter, verify, and execute the sample program. When the program is running, the LED display on the ET-3400 Trainer will sequentially turn each segment on and off and then return to the monitor.

## MONITOR COMMAND SUMMARY

### REGISTER

<u>COMMAND</u>	<u>FUNCTION</u>
R	Display all the registers.
CTRL/P	Display/alter the program counter.
CTRL/X	Display/alter the index register.
CTRL/A	Display/alter accumulator A
CTRL/B	Display/alter accumulator B
CTRL/C	Display/alter the condition codes.

### MEMORY

<u>COMMAND</u>	<u>FUNCTION</u>
D addr1, . . .,addrN	Display an area of memory on your console starting from location addr1 through addrN.
M addr1	Display/Alter sequential memory location starting from addr1.
I addr1	Display sequential program instructions starting from memory location addr1.
CTRL/S addr1, addr2,cnt	Transfer a block of memory contents starting from location addr1 to the memory location starting at addr2. The hexadecimal integer count (cnt<=FF ) is the number of bytes to be transferred.

**PROGRAM EXECUTION CONTROL**

<u>COMMAND</u>	<u>FUNCTION</u>
G addr1	Run the program starting from location addr1.
S addr1	Execute a single program instruction from location addr1.
E cnt	Using the present value of the program counter as a starting value, execute a series of instructions. (cnt<=FF )
H addr1	Insert a single haltpoint address into the breakpoint table.
C addr1	Remove a single haltpoint address from the breakpoint table.
CTRL/H	Examine the status of the breakpoint table.

**INPUT/OUTPUT OPERATIONS**

<u>COMMAND</u>	<u>FUNCTION</u>
T addr1, . . . ,addrN	Write the memory contents from location addr1 through addrN to a cassette tape at 300 baud.
CTRL/T #,addr1,addrN	Write the memory contents from location addr1 through addrN to a cassette tape. The symbol "#" refers to an integer value representing the desired output baud rate.
L	Read a cassette tape into memory at 300 baud.
L #	Read a cassette tape into memory. The symbol "#" refers to an integer value representing the desired output baud rate.

**ET-3400 USAGE**

<u>COMMAND</u>	<u>FUNCTION</u>
D 1A8F — — — — Fr — — — — La	Start the cassette and: enter the first address enter the last address
D 1ABC	Start the cassette and the monitor routine that reads a cassette tape.

**TELETYPEWRITER**

<u>COMMAND</u>	<u>FUNCTION</u>
P addr1,addrN	Punches a tape using the MIKBUG* format.
L 0	Reads a paper tape that was created with the MIKBUG format.

## HEATH/PITTMAN TINY BASIC

Tiny BASIC is a subset of BASIC\* that allows you to easily create your own computer programs. For instance, a program to balance your checkbook is easy to write using Tiny BASIC. The People's Computer Company (PCC), a nonprofit corporation in Menlo Park, Ca., conceived the idea of a compact computer language designed to teach programming skills. The implementation of Tiny BASIC follows the philosophy of the original idea.

In keeping with the "small is good" philosophy, Heath/Pittman Tiny BASIC employs a two-level interpreter approach with its consequent reduction in speed. The Heath Tiny BASIC firmware is permanently located in your computer system. The obvious advantage to this arrangement is the protection from a runaway program given to the Tiny BASIC interpreter. Also, you do not need to load the interpreter from cassette every time BASIC is used.

The following pages describe the function, operation, and features of Tiny BASIC. Some of the major features are:

- Integer Arithmetic (16-bit)
- Twenty six Variables (A, B, . . . ,Z)
- Fifteen BASIC statements:

LET	LOAD	INPUT	REM
RUN	SAVE	PRINT	IF (THEN)
END	GOTO	GOSUB	RETURN
BYE	LIST	CLEAR	
- FUNCTIONS: Random (RND)  
User (USR)

\*BASIC is a registered trademark of the Trustees of Dartmouth College.

## EDITING COMMANDS

Tiny BASIC lets you modify a program by inserting, changing, or deleting lines in the program. You can insert lines by typing a line with a line number that is not currently in the program. You can change lines by typing a new line with the same line number, and you can delete lines by typing a line number followed immediately by a carriage return.

Two control characters also permit you to edit a line as you enter it. Hold the control (CTRL) key down and then press a U or H to delete either a complete line of text or a single character, respectively.

CTRL/U This command deletes the current line.

CTRL/H This command deletes the previous character.

## USING TINY BASIC

Heath Tiny BASIC employs several FANTOM II Monitor subroutines. Therefore, you must always initialize the Monitor and use the Monitor command (G) to start BASIC. This causes Tiny BASIC to execute a CLEAR command. BASIC then prints a prompt character (:) on your terminal, indicating that the system firmware is functioning and awaiting a command. The entry to Tiny BASIC is at 1C00, so you must use "G 1C00" to start it.

For example, the following program prints a message on your terminal several times. The procedure to implement this program requires that you initialize the FANTOM II Monitor, start the Tiny BASIC interpreter, create and execute a BASIC program, and finally return control to the monitor.

- Initialize the FANTOM II monitor by entering "D0 1400 Ⓞ".
- Type "G 1C00 Ⓞ" on your console. This is the Tiny BASIC starting address.
- Enter the following program statements after the prompt ( : ) character.

```

:100 LET I=0
:200 PRINT "HEATH TINY BASIC"
:300 I=I+1
:400 IF I<5 GOTO 200
:500 END

```

- Type "RUN Ⓞ". The program prints  
HEATH TINY BASIC  
five times on your display, and then outputs a prompt character.
- Type "BYE Ⓞ". System control is then returned to the monitor.

The BReaK key is used to interrupt the execution of a Tiny BASIC program. This is particularly valuable if a program is in an infinite loop. You may stop it by pressing the BReaK key and holding it until Tiny BASIC responds "!0 AT NNN". This error message tells you that the BReaK key was pressed and line NNN is the next line to be executed. To continue running your program, you may type "GOTO NNN".

NOTE: When your program is at an INPUT statement, the BReaK key is disabled. You must either respond to the INPUT request with data or use a "MASTER RESET" from the ET-3400 keypad to regain system control.

## MODES OF OPERATION

You can use either the COMMAND mode or the PROGRAM mode when working with Tiny BASIC. An instruction in the COMMAND mode does not have a line number and is immediately executed after the carriage return. An instruction in the PROGRAM mode has a line number and will not execute until a RUN command is given. For example, the following two statements perform the same operation. However, the second statement will not be executed until you type RUN  $\text{\textcircled{R}}$  on the keyboard.

```
:PRINT "TESTING THE ETA-3400 ACCESSORY"  $\text{\textcircled{R}}$ 
```

```
:10 PRINT "TESTING THE ETA-3400 ACCESSORY"  $\text{\textcircled{R}}$ 
```

The important thing to remember about the modes of operation is: The COMMAND mode primarily assists you in detecting and debugging program errors, whereas the PROGRAM mode collects statements that will eventually become your finished computer program.

All Tiny BASIC instructions are valid in either mode. However, some of the instructions only make sense in one of the modes. For this reason, RUN and LIST should not be used in the PROGRAM mode. Also, END and RETURN should not be used in the COMMAND mode.

All instructions function the same in either mode except for INPUT and GOTO. In COMMAND mode, the data that is to be INPUTted must be on the same line. Thus,

```
:INPUT X,5,Y,7
```

will cause the variable X to be set to 5 and Y to be set to 7. In addition, in the COMMAND mode, a GOTO will not be accepted until the program has been started with a RUN command at least once.

## INSTRUCTIONS

A list of the instructions that Tiny BASIC recognizes is given below. It assumes that you are familiar with programming in the BASIC language. If you are not comfortable using BASIC, a course such as "BASIC Programming," Heath Model EC-1100, will help you to become proficient with BASIC.

<u>INSTRUCTION FORM</u>	<u>DESCRIPTION</u>
REM (text)	The remark (REM) is a nonexecutable statement, used only for commentary.
LET Var = Exp or Var = Exp	This instruction assigns the value of the expression to the variable. Variable values are not preset. Therefore, always assign an initial value to a variable before using it.
INPUT Var1,...,VarN	This instruction allows you to read data from the keyboard and assign values to the variables.
PRINT "message";Arg or PR Arg1,...,ArgN	The message or value of the argument is printed on the console terminal. Messages may be numbers or letters and are enclosed within quotations. If a comma is used between items in the PRINT list, items are printed in fields that start in columns 1, 8, 16, 32, and so on. If semicolons are used between the items, no space is left between them when they are printed.
GOTO NNN	The program is unconditionally transferred to the statement numbered NNN and execution continues.
GOSUB NNN	The go-to-subroutine (GOSUB) instruction transfers program execution to the statement number. When the RETURN instruction is encountered in the subroutine, program execution returns to the statement following GOSUB.
RETURN	Once program control is transferred to a subroutine, program execution continues until program control encounters a RETURN statement. A subroutine must always be terminated with a RETURN statement.

---

IF Exp1 rel Exp2 THEN Stmt	<p>If the test “Exp1 rel Exp2” is true, the statement after the “THEN” is executed. This statement can be any Tiny BASIC statement. The “THEN Stmt” part can be replaced by</p> <p style="text-align: center;">GOTO NNN</p> <p>Tiny BASIC recognizes the relational operators:</p> <p style="text-align: center;">= &lt; &gt; &lt;= &gt;= &lt;&gt; &gt;&lt;</p>
RUN	<p>This instruction starts the program at the statement with the lowest statement number.</p>
END	<p>When the interpreter encounters an END statement in your program, it stops program execution and returns control to the command mode.</p>
LIST LIST NNN LIST NNN1,NNN2	<p>The LIST instruction writes the entire buffer contents to your terminal. The LIST instruction followed by an argument writes either a single program statement or the range of statements between the arguments. (( NNN1 &lt; NNN2 ))</p>
CLEAR	<p>The interpreter removes all program statements from the buffer when it encounters a CLEAR instruction.</p>
BYE	<p>Executing a BYE instruction causes the interpreter to exit BASIC and return to the FANTOM II Monitor. The exit does not clear the buffer and you can return to BASIC with the buffer contents intact by using a warm start (see Page 33).</p>
SAVE	<p>The SAVE instruction directs Tiny BASIC to write the buffer contents at 300 baud to a cassette tape.</p>
LOAD	<p>The LOAD instruction reads a cassette tape at 300 baud and transfers a previously saved computer program into the buffer.</p>

## MATHEMATICAL EXPRESSIONS

A mathematical expression is the combination of one or more constants, variables, and functions connected by arithmetical operators. For instance, the Tiny BASIC statement: `LET A = 5+6/3-2*2` contains a mathematical expression.

### NUMERICAL CONSTANTS

All constants in Tiny BASIC are evaluated as 16-bit signed integers. An integer constant is written without a decimal point, using the decimal digits zero through nine. Unless they are preceded by a negative sign, integer constants are assumed to be positive.

### VARIABLES

A variable is any capital letter (A-Z). The letter is a symbol for a numeric value capable of changing during program execution. The value of this variable can range from  $-32768$  to  $32767$ . "Appendix A" contains the address of each of the 26 variables used by Tiny BASIC.

### OPERATORS

Tiny BASIC uses four arithmetical operators; addition (+), subtraction (-), multiplication (\*), and division (/). The statement `LET A = 5+6/3-2*2` is an example of a mathematical expression using these operators. Tiny BASIC processes these operators in the same fashion that you would use to solve an algebraic expression. For example, Tiny BASIC first evaluates  $6/3$  and  $2*2$  and then evaluates the expression to  $A=5+2-4$  and sets the variable A equal to 3. Because Tiny BASIC evaluates multiplication and division before addition and subtraction, you must be careful when writing any mathematical expression. If you are not certain of the order of operations, use parentheses to force the order you wish. Evaluation always proceeds from left to right, except that arguments enclosed within parentheses are evaluated first.

Tiny BASIC also uses two unary (+ or -) operators. These operators denote whether an expression is positive or negative. The expression `LET A = 5-(-3)` causes the variable A to equal eight.

## TINY BASIC RE-INITIALIZATION (Warm Start)

Tiny BASIC, in conjunction with the FANTOM II Monitor, allows you to exit Tiny BASIC and then re-enter it without clearing program statements and variables. In particular, the warm start re-entry preserves any remaining program and sets your memory limits. You can also reserve a block of memory by changing the high or low memory address (“Appendix A, Tiny BASIC Memory Map”) and combine a BASIC program with a routine written in machine code.

The warm start is used after you have left Tiny BASIC by typing “BYE” or by pressing RESET on the ET-3400 Trainer. From the FANTOM II Monitor, when you have the “MON>” prompt, type “B” to do a warm start of Tiny BASIC.

## FUNCTIONS

You may use either of two intrinsic functions in Tiny BASIC. The random (RND) function allows you to generate a positive pseudo-random integer. The user (USR) function is actually a call to a machine language subroutine that you have previously written. You can use either function in the COMMAND or PROGRAM mode.

### THE RND FUNCTION

The RaNDom function selects a positive pseudo-random integer between zero and one less than the argument. The argument is an integer or variable between 1 and 32767. For instance, the following statement, when inserted in the sample program, causes the computer to store a random integer between zero and eight in the variable J.

```
LET J = RND(9)
```

### THE USR FUNCTION

If a subroutine is written in Tiny BASIC, you simply use the GOSUB and RETURN commands to call and return from the subroutine. This is no problem. But suppose you wish to call a machine language subroutine from a program written in Tiny BASIC. This is the purpose of the USR function.

The USR function also permits you to call two routines in the Tiny BASIC interpreter. These two are commonly called PEEK and POKE, but they are not part of Tiny BASIC's vocabulary. You must implement the USR function to call the PEEK and POKE interpreter subroutines. These two routines let you get at nearly every feature of your microcomputer. As the name implies, you can examine the contents of selected memory locations with the PEEK routine. The POKE routine lets you enter data into memory locations.

First, how do machine language subroutines work? A subroutine is called with a JSR instruction. This pushes the return address onto the stack and jumps to the subroutine whose address is in the JSR instruction. When the subroutine has finished its operation, it executes the RTS instruction, which retrieves that address from the stack, returning control to the program that called it.

Depending on what function the subroutine is to perform, data may be passed to the subroutine by the calling program in one or more of the CPU registers and results may be passed back from the subroutine to the main program in the same way. The registers contain either addresses or more data. In some cases, the subroutine has no need to pass data back and forth, so the contents of the registers may be ignored.

The USR function may be called with one, two, or three arguments. These arguments are enclosed by parentheses, separated by a comma, and may be constants, variables, or expressions. The first of these is always the address of the subroutine to be called. The second and third arguments allow you to pass data through the CPU registers. The value of the second argument is placed in the index register while registers A and B contain the third argument. The forms of the USR statement are:

```
A = USR (sa)
A = USR (sa, x)
A = USR (sa, x, r)
```

The starting address (sa) and the index register (x) are 16-bit arguments. The third argument (r) is also 16 bits, but must be split between two registers. The most significant 8 bits of the third argument go into the B register, while the least significant bits are placed in the A register. However, it is important to realize that the three arguments in the USR function are decimal expressions and not the hexadecimal expressions that are normally associated with machine language programs. Any valid combination of numbers, variables, or expressions can be used as arguments.

The value returned by a USR function is a 16-bit number that is split between the A and B registers. The most significant byte is in the B register, and the least significant byte is in the A register. If your BASIC program does not use a returned value (such as POKE), the USR does not have to set up one. However, if the USR is supposed to return a value (such as PEEK), you must set up the value in the machine language of the USR.

The sample program on the next page shows you how to implement the USR function. The program accesses the Tiny BASIC interpreter subroutines "POKE" and "PEEK", which permit you to alter or examine the contents of memory locations. The program lets you store fifteen integer variables into an array that occupies the lowest memory in your computer system.

The program uses a simple loop to input and store data in memory locations zero through fourteen. After running the program, use the BYE command to exit Tiny BASIC and return to the Monitor. You can then examine the memory locations and verify that the program stores data in memory. By using a warm start, you can return to your Tiny BASIC program without deleting program statements.

The program accesses two machine language subroutines. PEEK and POKE. PEEK is permanently programmed into ROM starting at hexadecimal memory locations 1C14 (7188) and POKE is at location 1C18 (7192).

## SAMPLE USR PROGRAMS

```
10 REM THIS PROGRAM IS AN ADAPTATION OF A ROUTINE
11 REM PUBLISHED BY TOM PITTMAN FOR KILOBAUD MAGAZINE.
12 REM HEATH HAS OBTAINED PERMISSION FROM KILOBAUD TO
13 REM REPRINT SEVERAL ARTICLES AT THE END OF THIS
14 REM MANUAL ABOUT TINY BASIC. THESE ARTICLES PRESENT
15 REM AN INFORMATIVE DISCUSSION ON TINY BASIC.
16 REM
17 REM
18 REM
20 REM LET "L" REPRESENT THE VARIABLE FOR THE
21 REM ADDRESS OF THE INDEX REGISTER.
22 REM
23 LET L=0
24 REM
30 REM LET "J" REPRESENT THE VARIABLE DATA THAT
31 REM WILL BE STORED IN ARRAY MEMORY LOCATIONS 0-15.
32 REM
33 INPUT J
34 REM
40 REM "POKE" THE VARIABLE "J" INTO LOCATION "L" .
41 REM
42 LET J=USR(7192,L,J)
43 REM
50 REM USE THE "PEEK"COMMAND TO WRITE DATA FROM
51 REM ARRAY LOCATION "L" INTO VARIABLE "N", THEN
52 REM USE A PRINT STATEMENT TO VERIFY THAT THE DATA
53 REM WAS CORRECTLY STORED.
54 REM
55 LET N=USR(7188,L)
56 REM
57 PRINT "INTEGER ",N," IS LOCATED AT ADDRESS ",L
58 REM
60 REM INCREMENT INDEX REGISTER AND TEST FOR END OF ARRAY.
62 LET L=L+1
64 IF L<15 GOTO 30
70 END
```

In the next example, the USR function lets you call two separate machine language subroutines. A listing of these routines is provided in Figures 1A and 1B. The first routine, "LEDOFF", turns off the ET-3400 LED display, while the other routine, "LEDON", lights various LED segments. Both routines use accumulators A and B to pass a value from the USR function to the BASIC program.

```

0000 BD FE50    LEDOFF    JSR    OUTST1
0003 000000                    FCB    0,0,0
0006 000000                    FCB    0,0,0
0009 80                        FCB    80
000A 86 44                    LDAA  #$44
000C 5F                        CLR B
000D 39                        RTS

```

**Figure 1A**

```

0100 CE C16F    LEDON     LDX    DG6ADD
0103 BD FE50                    JSR    OUTST1
0106 3E5B05                    FCB    3E,5B,05
0109 47158D                    FCB    47,15,8D
010C 86 AA                    LDAA  #/AA
010E 5F                        CLR B
010F 39                        RTS

```

**Figure 1B**

The USR function requires that you either reserve an area of memory for machine code by adjusting the low memory address of BASIC user space upward, or you use the available bytes in low memory.\* Both methods are featured in this example.

\*NOTE: See "Appendix A" for a complete memory map. Always use caution when you are working in memory locations below 100<sub>H</sub> for subroutines. This area is generally used by BASIC and the Monitors to store program variables. This example only shows you that areas of memory are available. However, the accepted procedure is to reserve an area of memory above address 100<sub>H</sub> for your programs.

Use the following procedure to adjust BASIC's low memory limit. For example, the "LEDON" subroutine requires sixteen bytes of memory. Therefore, add the number of program bytes to the constant 0100<sub>H</sub> and insert the result in memory locations 20<sub>H</sub> and 21<sub>H</sub>. Replacing these values changes the low memory limit in BASIC.

```

0100 Tiny BASIC low memory address.
+ 10 Number of program bytes needed.
0110 New low memory address.

```

Reserve memory locations 0100<sub>H</sub> through 010F<sub>H</sub> for the program by using the following procedure. First, enter BASIC from the monitor. This will initialize the interpreter, and you will be able to set the new low memory limit by exiting BASIC and replacing the value with your new low memory limit. For example:

```

MON> G 1000
HTB1: BYE
MON> M 20 00
0020 01 00
0021 00 10
0022 NN ECS
MON>

```

Now use the Phantom II Monitor to enter the machine code from Figure 1A and 1B. The two subroutines are almost identical because they call another subroutine (OUTST1) located in the ET-3400 monitor. This routine outputs data to the LED displays. The major difference between the routines is in the program data. Changing this data changes the display.

Observe that the program statement, LDX DG5ADD, is missing from the LEDOFF routine. The operand, DG6ADD, corresponds to Hexadecimal value C16F, which is the address of the left-most digit on your ET-3400 Trainer. This value must be in the index register before the USR program inserts this value ( $49519_{10} = C16F_H$ ) into the index register for the second program.

The machine language subroutines performs one additional operation before returning to BASIC. The hexadecimal value entered into accumulators A and B is returned to the USR variable (i.e.  $A = USR(0)$ ). When the return from subroutine instruction is executed, these values are converted to a decimal equivalent and stored in variable A. The value stored in this variable determines the on/off delay time of the LED display. Changing the value in the accumulators lets you alter this delay time.

Always use a warm start to reenter BASIC after you adjust the memory limits and enter the machine code. If you do not use a warm start, BASIC will reinitialize the available memory and write over any program that you may have in memory. That is:

```
MON> B ☹  
:
```

Enter the following BASIC program statements after you adjust the low memory boundry and enter your machine language subroutines.

```
10 K=5  
20 PR " OBSERVE ET-3400 DISPLAY"  
30 A=USR(256)  
40 GOSUB 100  
50 A=USR(0,49519)  
60 GOSUB 100  
70 K=K-1  
80 IF K>0 GOTO 30  
90 END  
100 A=A-1  
110 IF A>0 GOTO 100  
120 RETURN
```

The LED display on the ET-3400 will display a message when you run the program. Program statement 30 calls the machine language routine that prints the "USr Fnc." message. After lighting the display, the program returns to BASIC and enters the time delay subroutine.

Program statement 50 calls the routine that turns off the LED display. Note that the decimal value, 49519, is equivalent to the hexadecimal value C16F. Setting the index register in the calling program reduces the memory requirements in the subroutine.

The starting address of each routine is supplied in decimal as the first argument in the USR function. If the address is not included, the program will never be executed. If the address is wrong, the jump will be to the wrong place in memory and unpredictable results will occur.

# APPENDIXES

## APPENDIX A

### *Tiny Basic Memory Map*

LOCATION	SIGNIFICANCE
0000-000F	Not used by Tiny BASIC.
0010-001F	Temporaries.
0020-0021	Lowest address of user program space.
0022-0023	Highest address of user program space.
0024-0025	Program end + stack reserve.
0026-0027	Top of GOSUB stack.
0028-002F	Interpreter parameters.
0030-007F	Input line buffer and Computation stack.
0080-0081	Random Number generator workspace.
0082-00B5	Variables: A,B,...,Z
00B6-00C7	Interpreter temporaries.
0100-0FFF	Tiny BASIC user program space.
1C00	Cold start entry point.
1C03	Warm start entry point.
1C06	Character input routine.
1C09	Character output routine.
1C0C	Break test.
1C0F	Backspace code.
1C10	Line cancel code.
1C11	Pad character.
1C12	Tape mode enable flag. (HEX 80 = enabled)
1C13	Spare stack size.
1C14	Subroutine (PEEK) to read one byte from RAM to B and A. (address in X)
1C18	Subroutine (POKE) to store A and B into RAM at address in X.

## APPENDIX B

### *Tiny Basic Error Message Summary*

<u>NUMBER</u>	<u>MEANING</u>
0	Break during execution.
8	Memory overflow; line not inserted.
9	Line number 0 is not allowed.
13	RUN with no program in memory.
18	LET is missing a variable name.
20	LET is missing an =.
23	Improper syntax in LET.
25	LET is not followed by END.
34	Improper syntax in GOTO.
37	No line to GOTO.
39	Misspelled GOTO.
40	Misspelled GOSUB.
41	Misspelled GOSUB.
46	GOSUB invalid. Subroutine does not exist.
59	PRINT not followed by END.
62	Missing close quote in PRINT string.
73	Colon in PRINT is not at end of statement.
75	PRINT not followed by END.
95	IF not followed by END.
104	INPUT syntax bad — expects variable name.
123	INPUT syntax bad — expects comma.
124	INPUT not followed by END.
132	RETURN syntax is bad.
133	RETURN has no matching GOSUB.
134	GOSUB not followed by END.

139	END syntax bad.
154	Cannot list line number 0.
158	LIST not followed by END statement.
164	LIST syntax error — expects comma.
183	REM not followed by END.
188	Memory overflow, too many GOSUB'S.
211	Expression too complex.
224	Divide by zero.
226	Memory overflow.
232	Expression too complex.
233	Expression too complex using RND.
234	Expression too complex in direct evaluation.
253	Expression too complex — simplify.
259	RND(0) not allowed.
266	Expression too complex.
267	Expression too complex for RND.
275	USR expects ( before argument.
284	USR expects ) after argument.
287	Expression too complex.
288	Expression too complex for USR.
290	Expression too complex.
293	Syntax error in expression — expects value.
296	Syntax error — missing ) .
298	Memory overflow — CHECK USR function.
303	Expression too complex in USR.
304	Memory overflow.
306	Syntax error.
330	Syntax error — check IF/THEN.
363	Missing statement. Type keyword.
365	Misspelled statement. Type keyword.

---

**APPENDIX C**

*Heath/Wintek Monitor Listing*

HEATH KEYBOARD MONITOR  
RAM AND CHARACTERS DEFINED

```

***      HEATH/WINTEK TERMINAL MONITOR SYSTEM
*
*      BY JIM WILSON FOR WINTEK CORPORATION
*      COPYRIGHT 1978 BY WINTEK CORP.
*      ALL RIGHTS RESERVED

**      CONDITIONAL ASSEMBLIES

0000      DEBUG      EQU      0          DEBUG CODE OFF

**      CHARACTER DEFINITIONS

000D      CR          EQU      0DH
000A      LF          EQU      0AH
0020      SPACE      EQU      ' '

**      PIA DEFINITION

1000      ORG          $1000
1000      TERM        RMB      1
1001      TERM.C      RMB      1
1002      TAPE        RMB      1
1003      TAPE.C      RMB      1

**      EXTERNALS

FE6B      SSTEP      EQU      0FE6BH
FEFC      SWIVE1     EQU      0FEFCH
FF76      OPTAB      EQU      0FF76H
FC8C      REDIS      EQU      0FC8CH
FD7B      DISPLAY   EQU      0FD7BH
FE20      OUTBYT     EQU      0FE20H
FD43      BKSP       EQU      0FD43H
FD25      PROMPT    EQU      0FD25H
FC86      OUTSTA     EQU      0FC86H
FE52      OUTSTR     EQU      0FE52H

**      RAM TEMPORARIES

00CC      ORG          0CCH
00CC      USERC     RMB      1          CONDX CODES
00CD      USERB     RMB      1
00CE      USERA     RMB      1          ACCUMULATORS
00CF      USERX     RMB      2          INDEX
00D1      USERP     RMB      2          P.C.
00E4      ORG          0E4H
0004      NBR        EQU      4          FOUR BREAKPOINTS ALLOWED
00E4      BKTEL     RMB      2*NBR
00EC      T0        RMB      2
00EE      T1        RMB      2
00F0      DIGADD    RMB      2
00F2      USERS     RMB      2
00F4      T2        EQU      *
00F4      SYSSWI    RMB      3
00F7      UIRQ      RMB      3

```

HEATH KEYBOARD MONITOR  
RAM AND CHARACTERS DEFINED

```

00FA          USWI      RMB      3
00FD          UNMI      RMB      3

FFFF          IF        DEBUG-1
              ELSE
1400          ORG        $1400
              ENDIF

              **        MAIN MONITOR LOOP
              *
              *        1) FEELS OUT MEMORY
              *        2) SEARCHES FOR PAST INCARNATIONS
              *           A) CLEARS BREAKPOINTS IF REINCARNATED
              *           B) CLEARS BREAKPOINT TABLE OTHERWISE
              *        3) SENDS PROMPT "MON>"
              *        4) ACCEPTS COMMAND CHARACTERS AND JUMPS
              *           TO APPROPRIATE HANDLER

1400 0F          MAIN    SEI
1401 CE 10 00    LDX      #TERM          TERMINAL FIA
1404 6F 01      CLR      1,X          IN CASE IRREGULAR ENTRY
1406 6F 03      CLR      3,X
1408 86 01      LDA A    #1
140A A7 00      STA A    0,X
140C 86 7F      LDA A    #01111111B
140E A7 02      STA A    2,X
1410 C6 04      LDA B    #4
1412 E7 01      STA B    1,X
1414 E7 03      STA B    3,X
1416 A7 00      STA A    0,X          IDLE MARKING!!

              *        NOW FIND MEMORY EXTENT

1418 09          MAIN1   DEX
1419 A6 00      LDA A    0,X
141B 63 00      COM      0,X
141D 43          COM A
141E A1 00      CMP A    0,X
1420 26 F6      BNE      MAIN1
1422 63 00      COM      0,X          RESTORE GOOD BYTE
1424 86 15      LDA A    #4*NBR+5
1426 09          MAIN2   DEX          GO TO MONITOR GRAVEYARD
1427 4A          DEC A
1428 26 FC      BNE      MAIN2
142A 35          TXS
142B 86 0C      LDA A    #2*NBR+4
142D EE 08      LDX      2*NBR,X          RETURN ADDRESS IF ANY
142F 8C 14 4C   CPX      #MAIN5
1432 27 09      BEQ      MAIN4          IS RE-INCARNATION
1434 C6 FF      LDA B    #FF
1436 30          TSX
1437 E7 0A      MAIN3   STA B    2*NBR+2,X
1439 08          INX
143A 4A          DEC A
143B 26 FA      BNE      MAIN3

```

HEATH KEYBOARD MONITOR  
 MAIN - MAIN MONITOR LOOP

```

143D 86 04      MAIN4  LDA A   #NBR          CLEAR BREAKPOINTS
143F 33                MAIN44 PUL B
1440 33                PUL B
1441 30                TSX
1442 EE 0C                LDX      2*NBR+4,X
1444 E7 00                STA B   0,X
1446 4A                DEC A
1447 26 F6                BNE     MAIN44
1449 0C                CLC          NO ERROR MESSAGE
144A 31                INS
144B 31                INS
144C 24 0D      MAIN5  BCC     MAIN6          NO ERROR
144E BD 16 18                JSR     OUTIS
1451 0D 0A 45                FCB    CR,LF,'ERROR!',7,0
145B BD 16 18      MAIN6  JSR     OUTIS
145E 0D 0A 4D                FCB    CR,LF,'MON> ',0
1466 7D 10 00      MAIN66 TST     TERM
1469 2A FB                BPL     MAIN66
146B BD 18 E1                JSR     INCH          INPUT COMMAND
146E CE 19 EF                LDX    #CMDTAB-3
1471 08                MAIN7  INX
1472 08                INX
1473 08                INX
1474 A1 00                CMP A   0,X
1476 25 F9                BCS     MAIN7
1478 26 D2                BNE     MAIN5          ILLEGAL COMMAND
147A 36                PSH A
147B BD 18 63                JSR     OUTSP
147E 32                PUL A
147F C6 4C                LDA B   #-MAIN5/256*256+MAIN5
1481 37                PSH B
1482 C6 14                LDA B   #MAIN5/256
1484 37                PSH B
1485 E6 02                LDA B   2,X
1487 37                PSH B
1488 E6 01                LDA B   1,X
148A 37                PSH B
148B 5F                CLR B
148C DE F2                LDX    USERS
148E 39                RTS

**      GO - GO TO USER CODE
*
*      ENTRY: (X) = USERS
*      EXIT:  UPON BREAKPOINT
*      USES:  ALL,T0,T1,T2

148F BD 16 25  GO      JSR     AHV
1492 24 04                BCC     GO1          NO OPTIONAL ADDRESS
1494 A7 07                STA A   7,X
1496 E7 06                STA B   6,X
1498 BD FE 6B  GO1     JSR     SSTEP          STEP FAST BKPT
149B C6 04                LDA B   #NBR
149D 30                GO2    TSX          COPY IN BREAKPOINTS
149E EE 0C                LDX    2*NBR+4,X

```

HEATH KEYBOARD MONITOR  
GO - GO TO USER CODE

14A0	A6 00		LDA A	0,X	
14A2	36		FSH A		
14A3	36		FSH A		
14A4	86 3F		LDA A	##3F	
14A6	A7 00		STA A	0,X	
14A8	5A		DEC B		
14A9	26 F2		BNE	G02	
14AB	20 3E		BRA	G07	
14AD	30	G03	TSX		
14AE	A6 06		LDA A	6,X	
14B0	26 02		BNE	G033	
14B2	6A 05		DEC	5,X	
14B4	E6 05	G033	LDA B	5,X	
14B6	4A		DEC A		
14B7	A7 06		STA A	6,X	DECREMENT USER PC
14B9	9F F2		STS	USERS	
14BB	9E EC		LIS	T0	
14BD	36		FSH A		
14BE	86 04		LDA A	#NBR	
14C0	97 EC		STA A	T0	
14C2	32		PUL A		
14C3	30		TSX		
14C4	08	G04	INX		SEARCH TABLE FOR HIT
14C5	08		INX		
14C6	A1 0D		CMP A	2*NBR+5,X	
14C8	26 19		BNE	G05	NO HIT HERE
14CA	E1 0C		CMP B	2*NBR+4,X	
14CC	26 15		RNE	G05	
14CE	BD 16 18		JSR	OUTIS	
14D1	0D 0A 00		FCB	CR,LF,0	
14D4	B6 04		LDA A	#NBR	
14D6	33	G044	PUL R		
14D7	33		PUL B		OP CODE INTO B
14D8	30		TSX		
14D9	EE 0C		LDX	2*NBR+4,X	
14DB	E7 00		STA B	0,X	
14DD	4A		DEC A		
14DE	26 F6		BNE	G044	
14E0	7E 15 53		JMP	REGS	DISPLAY REGISTERS
14E3	7A 00 EC	G05	DEC	T0	
14E6	26 DC		BNE	G04	
		*	SWI NOT MONITORS SO INTERPRET		
14E8	BD FE 6B		JSR	SSTEP	STEP PAST SWI
14EB	9F EC	G07	STS	T0	
14ED	CE 14 AD		LDX	#G03	
14F0	7E FE FC		JMP	SWIVE1	

HEATH KEYBOARD MONITOR  
BKPT - INSERT BREAKPOINT

```

**      BKPT - INSERT BREAKPOINT INTO TABLE
*
*      ENTRY:  NONE
*      EXIT:   'C' SET IF TABLE FULL
*      USES:   ALL,TO

14F3  30      BKPT   TSX
14F4  86 FF      LDA A  #FF
14F6  C6 04      LDA B  #NBR
14F8  08      BKFP1  INX
14F9  08      INX          LOOK FOR EMPTY SPOT
14FA  A1 04      CMP A  4,X
14FC  26 04      BNE     BKFP2  NOT EMPTY
14FE  A1 05      CMP A  5,X
1500  27 05      BEQ     BKFP3  IS EMPTY
1502  5A      BKFP2  DEC B
1503  26 F3      BNE     BKFP1  STILL HOPE
1505  0D      SEC
1506  39      RTS          FULL!!

1507  BD 16 25  BKFP3  JSR   AHV          GET BREAKPOINT VALUE
150A  24 04      BCC     BKFP4  NO ENTRY
150C  A7 05      STA A  5,X
150E  E7 04      STA B  4,X
1510  0C      BKFP4  CLC
1511  39      RTS

**      CLEAR - CLEAR BREAKPOINT ENTRY
*
*      ENTRY:  (X) = USERS
*      EXIT:   'C' SET IF NOT FOUND
*      USES:   ALL,TO

1512  86 04      CLEAR  LDA A  #NBR
1514  97 EC      STA A  TO
1516  BD 16 25  JSR   AHV          GET LOCATION
1519  25 04      BCS     CLE1    NO VALID HEX
151B  A6 07      LDA A  7,X
151D  E6 06      LDA B  6,X          USER PC FOR DEFAULT
151F  30      CLE1   TSX
1520  08      CLE2  INX
1521  08      INX
1522  A1 05      CMP A  5,X          SEARCH TABLE
1524  26 04      BNE     CLE3  NOT FOUND
1526  E1 04      CMP B  4,X
1528  27 07      BEQ     CLE4  FOUND
152A  7A 00 EC  CLE3  DEC   TO
152D  26 F1      BNE     CLE2
152F  0D      SEC
1530  39      RTS

1531  C6 FF      CLE4  LDA B  #FF
1533  E7 04      STA B  4,X          CLEAR ENTRY
1535  E7 05      STA B  5,X
1537  0C      CLC

```

HEATH KEYBOARD MONITOR  
BKPT -- INSERT BREAKPOINT

```

1538 39          RTS

          **      EXEC - PROCESS MULTIPLE SINGLE STEP
          *
          *      ENTRY:  NONE
          *      EXIT:   REGISTERS PRINTED
          *      USES:   ALL,T0,T1,T2

1539 BD 16 25  EXEC  JSR      AHV          GET COUNT
153C 25 09          BCS      EXEC1
153E 86 01          LDA A   #1          DEFAULT COUNT
1540 20 05          BRA      EXEC1

1542 36          EXEC0 PSH A          SAVE COUNT
1543 BD FE 6B      JSR      SSTEP       STEP CODE
1546 32          PUL A
1547 4A          EXEC1 DEC A
1548 26 F8          BNE      EXEC0       MORE STEPS
154A BD 16 18      JSR      OUTIS
154D 0D 0A 00      FCB      CR,LF,0

          **      STEP - STEP USER CODE
          *
          *      ENTRY:  NONE
          *      EXIT:   REGISTERS PRINTED
          *      USES:   ALL,T0,T1,T2

1550 BD FE 6B  STEP  JSR      SSTEP       STEP USER CODE

          **      REGS - DISPLAY ALL USER REGISTERS
          *
          *      ENTRY:  NONE
          *      EXIT:   REGISTERS PRINTED
          *      USES:   ALL,T0

1553 5F          REGS  CLR B
1554 DE F2          LDX      USERS
1556 86 43          LDA A   #'C'
1558 8D 26          BSR      REGS1
155A 86 42          LDA A   #'B'
155C 8D 24          BSR      REGS3
155E 86 41          LDA A   #'A'
1560 8D 20          BSR      REGS3
1562 86 58          LDA A   #'X'
1564 8D 1B          BSR      REGS2
1566 86 50          LDA A   #'P'
1568 8D 18          BSR      REGS3
156A 86 53          LDA A   #'S'
156C 09          DEX
156D DF EC          STX      T0
156F CE 00 ER      LDX      #T0-1
1572 8D 0C          BSR      REGS1
1574 DE F2          LDX      USERS

```

HEATH KEYBOARD MONITOR  
REGISTER DISPLAY COMMANDS

1576	EE 06		LDX	6,X	(X) = USERPC
1578	DF EC		STX	TO	
157A	A6 00		LDA A	0,X	
157C	8D 63		BSR	TYPINO	TYPE INSTRUCTION
157E	0C		CLC		
157F	39		RTS		
1580	08	REGS1	INX		
1581	5C	REGS2	INC B		
1582	BD 18 65	REGS3	JSR	OUTCH	OUTPUT REGISTER NAME
1585	86 3D		LDA A	#'='	
1587	BD 18 65		JSR	OUTCH	
158A	20 67		BRA	TYPIN2	

\*\* REGISTER DISPLAY COMMANDS

\*  
\* ENTRY: (X) = USERSP  
\* (B) = 0  
\* EXIT: OPTIONAL REPLACEMENT VALUE STORED  
\* USES: ALL,TO

158C	08	REGP	INX		
158D	08		INX		
158E	08	REGX	INX		
158F	5C		INC B		
1590	08	REGA	INX		
1591	08	REGB	INX		
1592	8B 40	REGC	ADD A	##40	DISPLACE REG NAME
1594	8D EA		BSR	REGS1	OUTPUT WITH NAME
1596	37		PSH B		
1597	BD 16 25		JSR	AHV	
159A	24 2F		BCC	MEM4	
159C	8D 05		BSR	REG1	
159E	17		TBA		
159F	33		PUL B		
15A0	5A		DEC B		
15A1	27 08		BEQ	REG2	
15A3	09	REG1	DEX		
15A4	A7 00		STA A	0,X	
15A6	A1 00		CMF A	0,X	
15A8	27 01		BEQ	REG2	
15AA	0D		SEC		
15AB	39	REG2	RTS		

\*\* MEM - DISPLAY MEMORY BYTES

\*  
\* ENTRY: (B) = 0  
\* (X) = USER S.P.  
\* USES: ALL,TO

15AC	5A	MEM	DEC B		
------	----	-----	-------	--	--

HEATH KEYBOARD MONITOR  
MEM - DISPLAY MEMORY OR INSTRUCTION

```

**      INST - DISPLAY INSTRUCTIONS
*
*      ENTRY:  (B) = 0
*              (X) = USER S.P.
*      USES:   ALL,TO

15AD  37      INST  PSH B
15AE  EE 06      LDX   6,X          GET USER P.C.
15B0  8D 73      BSR   AHV
15B2  24 07      BCC   MEM1
15B4  36      PSH A
15B5  37      PSH B
15B6  30      TSX
15B7  EE 00      LDX   0,X
15B9  31      INS
15BA  31      INS
15BB  0C      MEM1  CLC
15BC  33      MEM2  PUL B
15BD  24 05      BCC   MEM3
15BF  8D E2      BSR   REG1
15C1  25 0A      BCS   MEM5
15C3  08      INX
15C4  8D 08      MEM3  BSR   TYPINS      TYPE THE DATA
15C6  37      PSH B          SAVE MODE FLAG
15C7  8D 5C      BSR   AHV          GET REPLACEMENT VALUE
15C9  23 F1      BLS   MEM2
15CB  0C      MEM4  CLC
15CC  33      PUL B
15CD  39      MEM5  RTS

```

```

**      TYPINS - TYPE INSTRUCTION IN HEX
*
*      ENTRY:  (X) = ADDRESS OF INSTRUCTION
*      EXIT:   (X) = ADDRESS OF NEXT INST.
*      USES:   ALL

15CE  A6 00      TYPINS LDA A  0,X          OP CODE
15D0  36      PSH A          ONTO STACK
15D1  DF EC      STX   TO
15D3  8D 43      BSR   OUTIS
15D5  0D 0A 00    FCB   CR,LF,0
15D8  CE 00 EC    LDX   #TO
15DB  8D 2D      BSR   OUT4HS
15DD  32      PUL A
15DE  5D      TST B
15DF  2B 0E      BMI   TYPIN1      ONE BYTE ONLY
15E1  8D 66      TYPINO BSR   BYTCNT
15E3  2A 5A      DEC B
15E4  5A 09      BPL   TYPIN1      IS VALID INST.
15E6  5C      INC B          RESTORE (B)
15E7  8D 2F      BSR   OUTIS
15E9  44 41 54    FCB   'DATA=',0
15EF  DE EC      TYPIN1 LDX   TO
15F1  8D 19      BSR   OUT2HS
15F3  C1 01      TYPIN2 CMP B  #1

```

HEATH KEYBOARD MONITOR  
MEM - DISPLAY MEMORY OR INSTRUCTION

```
15F5 2B 20      BMI      THB1
15F7 27 13      BEQ      OUT2HS
15F9 20 0F      BRA      OUT4HS
```

\*\* DISB - DISPLAY BREAKPOINTS

```
*
* ENTRY:  NONE
* EXIT:   BREAKPOINT TABLE PRINTED
* USES:   ALL
```

```
15FB C6 06      DISB   LDA B   #6           OFFSET INTO TABLE
15FD 30          TSX
15FE 08          DISB1  INX
15FF 5A          DEC B
1600 26 FC          RNE      DISB1
1602 C6 04          LDA B   #NBR
1604 8D 04          DISB2  BSR      OUT4HS
1606 5A          DEC B
1607 26 FB          BNE      DISB2
1609 39          RTS
```

\*\* OUT4HS, OUT2HS - OUTPUT HEX AND SPACES

```
*
* ENTRY:  (X) = ADDRESS
* EXIT:   X UPDATED PAST BYTE(S)
* USES:   X,A,C
```

```
160A 8D 05      OUT4HS BSR      THB           TYPE HEX BYTE
160C 8D 03      OUT2HS BSR      THB
160E 7E 18 63   JMP      OUTSP
```

\*\* THB - TYPE HEX BYTE

```
*
* ENTRY:  (X) = ADDRESS OF BYTE
* EXIT:   X INCREMENTED PAST BYTE
* USES:   X,A,C
```

```
1611 37          THB   PSH B
1612 5F          CLR B
1613 BD 17 E4    JSR      OCH
1616 33          PUL B
1617 39          THB1  RTS
```

\*\* OUTIS - OUTPUT IMBEDDED STRING

```
*
* CALLING CONVENTION:
* JSR      OUTIS
* FCB      'STRING',0
* <NEXT INST>
* EXIT: TO NEXT INSTRUCTION
* USES:   A,X
```

HEATH KEYBOARD MONITOR  
MEM - DISPLAY MEMORY OR INSTRUCTION

```

1618 30          OUTIS  TSX
1619 EE 00          LDX   0,X
161B 31          INS
161C 31          INS
161D 37          PSH B
161E 5F          CLR B
161F BD 17 C3     JSR   OAS
1622 33          PUL B
1623 6E 00        JMP   0,X

**          AHV - ACCUMULATE HEX VALUE
*
*          ENTRY:  NONE
*          EXIT:   (BA) = ACCUMULATED HEX VALUE OR
*                  (A) = ASCII IF NO HEX
*                  'C' SET FOR VALID HEX
*                  'Z' SET FOR TERMINATOR = CR
*          USES:   B,A,C

1625 5F          AHV   CLR B
1626 BD 18 A3     AHVD  JSR   IHD          GET FIRST DIGIT
1629 24 1D        BCC   AHV3          NOT HEX
162B 36          AHV1  PSH A
162C 37          PSH B
162D 48          ASL A
162E 59          ROL B
162F 48          ASL A
1630 59          ROL B
1631 48          ASL A
1632 59          ROL B
1633 48          ASL A
1634 59          ROL B          MAKE WAY FOR NEXT DIGIT
1635 37          PSH B
1636 36          PSH A
1637 BD 18 A3     JSR   IHD
163A 24 07        BCC   AHV2          THIS NOT HEX
163C 33          PUL B
163D 1B          ABA
163E 33          PUL B
163F 31          INS
1640 31          INS          DISCARD OLD VALUE
1641 20 E8        BRA   AHV1

1643 31          AHV2  INS
1644 31          INS          SKIP LATEST VALUE
1645 33          PUL B
1646 32          PUL A
1647 0D          SEC
1648 39          AHV3  RTS

```

HEATH KEYBOARD MONITOR  
BYTCNT - COUNT INSTRUCTION BYTES

```

**      BYTCNT - COUNT INSTRUCTION BYTES
*
*      ENTRY: (A) = OPCODE
*      EXIT:  (B) = 0,1,2 OR 3
*      'C' CLEAR IF RELATIVE ADDRESSING
*      'Z' SET IF ILLEGAL.

1649 36      BYTCNT PSH A
164A 16      TAB
164B CE FF 75 LDX      #OPTAB-1
164E 08      BYT1  INX
164F C0 08      SUB B   #8
1651 24 FB      BCC     BYT1
1653 A6 00      LDA A   0,X
1655 46      BYT2  ROR A
1656 5C      INC B
1657 26 FC      BNE     BYT2
1659 32      PUL A
165A 25 1E      BCS     BYT7
165C 81 30      CMP A   ##30      CHECK FOR BRANCH
165E 24 04      BCC     BYT3
1660 81 20      CMP A   ##20
1662 24 14      BCC     BYT5      IS BRANCH
1664 81 60      BYT3  CMP A   ##60
1666 25 11      BCS     BYT6      IS ONE BYTE
1668 81 8D      CMP A   ##8D
166A 27 0C      BEQ     BYT5      IS BSR
166C 84 BD      AND A   ##BD
166E 81 8C      CMP A   ##8C
1670 27 04      BEQ     BYT4      IS X OR SP IMM.
1672 84 30      AND A   ##30      CHECK FOR THREE BYTES
1674 81 30      CMP A   ##30
1676 C2 FF      BYT4  SBC R   ##FF
1678 5C      BYT5  INC B
1679 5C      BYT6  INC B
167A 39      BYT7  RTS

**      COPY - COPY MEMORY ELSEWHERE
*
*      ENTRY:  NONE
*      EXIT:   BLOCK MOVED
*      USES:   ALL
*
*      COMMAND SYNTAX: (CNTL-)D <FROM>,<TO>,<COUNT>

167B BD 16 18  COPY  JSR     OUTIS
167E 53 4C 49      FCB     'SLIDE ',0
1685 BD 16 25      JSR     AHV      GET *FROM*
1688 24 19      BCC     COP3      NO HEX
168A 36      PSH A
168B 37      PSH B
168C BD 16 25      JSR     AHV      GET *TO*
168F 24 10      BCC     COP2      NO HEX
1691 36      PSH A
1692 37      PSH B

```

HEATH KEYBOARD MONITOR  
COPY - COPY MEMORY ELSEWHERE

```

1693 BD 16 25      JSR      AHV      GET *COUNT*
1696 24 07        BCC      COP1     NO HEX
1698 36           PSH A
1699 37           PSH B
169A BD 19 6D     JSR      MOVE     MOVE DATA
169D 0C           CLC
169E 39           RTS      NO ERRORS

169F 31          COP1  INS
16A0 31          INS
16A1 31          COP2  INS
16A2 31          INS
16A3 0D          COP3  SEC
16A4 39          RTS

**          LOAD - LOAD DATA INTO MEMORY
*
*          ENTRY:  NONE
*          EXIT:   'C' SET IF ERROR
*          USES:   ALL,TO

16A5 BD 16 25    LOAD  JSR      AHV      GET OPTIONAL PARAMETERS
16A8 25 02      BCS      LOA00
16AA 86 08      LDA A   #8          DEFAULT TO CASSETTE
16AC 16          LOA00  TAB
16AD 34          LOA0   IES
16AE 34          IES          SCRATCHPAD ON STACK
16AF BD 18 DE    LOA1  JSR      ICT      INPUT CASSETTE/TERM
16B2 84 7F      AND A   #7FH
16B4 81 53      CMP A   #'S'
16B6 26 F7      BNE
16B8 BD 18 DE    JSR      ICT
16BB 84 7F      AND A   #7FH
16BD 81 39      CMP A   #'9'
16BF 27 36      BEQ      LOA4          IS EOF
16C1 34          IES
16C2 81 31      CMP A   #'1'
16C4 26 E9      BNE      LOA1          NOT START-OF-RECORD
16C6 B7 C1 6F    STA A   0C16FH        TURN ON D.P.
16C9 4F          CLR A
16CA 30          TSX
16CB BD 18 C2    JSR      IHB          COUNT
16CE BD 18 C2    JSR      IHB          ADDRESS (2 BYTES)
16D1 BD 18 C2    JSR      IHB
16D4 30          TSX
16D5 EE 01      LDIX   1,X          GET FWA OF BUFFER
16D7 D7 EC      STA B   TO
16D9 33          PUL B
16DA C0 03      SUB B   #3          ACCOUNT 3 BYTES
16DC 37          LOA2  PSH B
16DD D6 EC      LDA B   TO
16DF BD 18 C2    JSR      IHB
16E2 D7 EC      STA B   TO
16E4 33          PUL B
16E5 5A          DEC B

```

HEATH KEYBOARD MONITOR  
LOAD - FROM TAPE OR TERMINAL

16E6	26 F4		BNE	LOA2	
16E8	7F C1 6F		CLR	0C16FH	TURN OFF D.P.
16EB	D6 EC		LDA B	T0	
16ED	CE 00 EC		LDX	#T0	
16F0	BD 18 C2		JSR	IHE	
16F3	4C		INC A		
16F4	27 B9		BEQ	LOA1	
16F6	0D	LOA3	SEC		
16F7	31	LOA4	INS		
16F8	31		INS		
16F9	39		RTS		

\*\* TIME CRITICAL ROUTINES !!!!!  
\* SINCE CASSETTE I/O IS DONE USING ONLY SOFTWARE  
\* TIMING LOOPS, THE ROUTINE 'BIT' MUST BE CALLED  
\* EVERY 208 US. CRITICAL TIMES IN THESE ROUTINES  
\* ARE LISTED IN THE COMMENT FIELDS OF CERTAIN  
\* INSTRUCTIONS IN THE FORM "NNN US". THESE TIMES  
\* REPRESENT THE TIME REMAINING BEFORE THE NEXT  
\* RETURN FROM 'BIT'. THE TIME INCLUDES THE  
\* LABELED INSTRUCTION AND INCLUDES THE EXECUTION  
\* OF THE 'RTS' AT THE END OF 'BIT'. SOME  
\* ROUTINES HAVE "NNN US USED" AS A COMMENT  
\* ON THEIR LAST STATEMENT. THIS REPRESENTS  
\* THE TIME EXPIRED SINCE THE LAST RETURN  
\* FROM 'BIT' INCLUDING THE LABELED INSTRUCTION.

\*\* HIGH SPEED LOAD  
\*  
\* ACCEPTS ADDITIONAL BIT/CELL PARAMETER  
\*  
\* ENTRY: (A) = COMMAND  
\* (B) = 0  
\*  
\* USES: ALL,T0,T1,T2

16FA	8B 40	CTLT	ADD A	##40	DISPLACE TO PRINTING
16FC	BD 18 65		JSR	OUTCH	ECHO TO USER
16FF	BD 16 25		JSR	AHV	
1702	16		TAB		
1703	C4 7F		AND B	##7F	
1705	20 03		BRA	PTAF	

\*\* RCRD - RECORD MEMORY DATA IN 'KCS' FORMAT  
\*  
\* ENTRY: (B) = 0  
\* USES: ALL,T0,T1,T2

1707	CB 09	RCRD	ADD B	#9	
------	-------	------	-------	----	--

HEATH KEYBOARD MONITOR  
PUNCH - PUNCH MEMORY

```

**          DUMP - RAW MEMORY DUMP 16 BYTES PER LINE
*
*          ENTRY:  (B) = 0
*          USES:   T0,T1,T2

1709  5A          DUMP  DEC B

**          PTAP - PUNCH TO TAPE
*
*          ENTRY:  DEFAULT VALUES ON STACK
*                  BELOW RETURN ADDRESS
*          EXIT:   'C' SET FOR ERROR
*          USES:   ALL,T0,T1,T2

170A  30          PTAP  TSX
170B  37          FSH B          CASSETTE/TERMINAL FLAG
170C  BD 16 25    JSR   AHV          ACCUMULATE HEX
170F  24 0B      BCC   PTAP1        USE DEFAULT
1711  A7 03      STA A 3,X          STORE FWA
1713  E7 02      STA B 2,X
1715  BD 16 25    JSR   AHV
1718  A7 05      STA A 5,X
171A  E7 04      STA B 4,X
171C  A6 05      PTAP1 LDA A 5,X
171E  E6 04      LDA R 4,X          GET LWA, FWA
1720  EE 02      LDX   2,X
1722  DF EE      STX   T1
1724  97 F5      STA A T2+1
1726  D7 F4      STA B T2
1728  33          PUL B

**          PUNCH - WRITE LOADER FILE TO TERMINAL OR CASSETTE
*
*          ENTRY:  (T1) = FWA BYTES TO PUNCH
*                  (T2) = LWA BYTES TO PUNCH
*                  (B) = CASSETTE TERMINAL FLAG:
*                      (B) > 0 THEN TO CASSETTE
*                          USING (B) CELLS PER BIT
*                      (B) = 0 THEN TO TERMINAL
*                      (B) < 0 THEN TO TERMINAL WITH
*                          IMBEDDED SPACES AND NO S1,ETC.
*          USES:   ALL,T0,T1

1729  5D          PUNCH  TST B
172A  2F 07      BLE   PNCH0
172C  BD 18 27    JSR   OLT          OUTPUT LEADER
172F  86 07      LDA A #7
1731  20 02      BRA   PNCH1

1733  86 04      PNCH0 LDA A #4          186 US
1735  4A          PNCH1 DEC A
1736  26 FD      BNE   PNCH1
1738  37          FSH B          SAVE FLAG# 160 US
1739  D6 F4      LDA B T2          (BA) = END# 156 US

```

HEATH KEYBOARD MONITOR  
PUNCH - PUNCH MEMORY

173B	96	F5		LDA	A	T2+1	
173D	90	EF		SUB	A	T1+1	
173F	D2	EE		SBC	B	T1	(BA) = END - CURRENT
1741	25	58		BCS		PNCH9	DONE; 144 US
1743	81	0F		CMF	A	#15	140 US
1745	C2	00		SBC	B	#0	
1747	33			PUL	B		RESTORE FLAG
1748	24	02		BCC		PNCH2	AT LEAST FULL RECORD
174A	20	03		BRA		PNCH3	
174C	86	0F	PNCH2	LDA	A	#15	
174E	01			NOF			
174F	97	EC	PNCH3	STA	A	T0	COUNTER
1751	8B	04		ADD	A	#4	
1753	97	ED		STA	A	T0+1	BYTE COUNT
1755	CE	17	B6	LDX		#S1STR	114 US
1758	5D			TST	B		
1759	2A	03		BPL		PNCH35	
175B	CE	17	C0	LDX		#CRSTR	
175E	8D	63	PNCH35	BSR		OAS	OUTPUT ASCII STRING
1760	CE	00	EE	LDX		#T0+2	
1763	4F			CLR	A		(A) = CHECKSUM
1764	01			NOF			
1765	5D			TST	B		
1766	2B	03		BMI		PNCH5	
1768	09			DEX			
1769	A5	00		BIT	A	0,X	5 CYCLE NUTHIN'
176B	01		PNCH5	NOF			
176C	01			NOF			
176D	8D	75		BSR		0CH	182 US
176F	01			NOF			
1770	26	F9		BNE		PNCH5	
1772	DE	EE		LDX		T1	
1774	8D	62	PNCH6	BSR		0SH	182 US
1776	7A	00	EC	DEC		T0	
1779	2A	F9		BPL		PNCH6	
177B	43			COM	A		
177C	36			FSH	A		
177D	01			NOF			
177E	86	07		LDA	A	#7	
1780	4A		PNCH7	DEC	A		
1781	26	FD		BNE		PNCH7	
1783	32			PUL	A		
1784	5D			TST	B		
1785	2B	02		BMI		PNCH75	NO CHECKSUM
1787	8D	6E		BSR		0HR	
1789	B6	10	00	PNCH75	LDA	A	TERM
178C	43			COM	A		
178D	49			ROL	A		
178E	DF	EE		STX		T1	
1790	DF	EE		STX		T1	
1792	22	9F		BHI		PNCH0	NOT DONE; NO BREAK
1794	08			INX			
1795	37			FSH	B		
1796	86	06		LDA	A	#6	
1798	4A		PNCH8	DEC	A		
1799	26	FD		RNE		PNCH8	

HEATH KEYBOARD MONITOR  
PUNCH - PUNCH MEMORY

179B	33		FNCH9	FUL B		140 US
179C	01			NOF		
179D	86 03			LDA A	#3	
179F	4A		FNCHA	DEC A		
17A0	26 FD			BNE	FNCHA	
17A2	CE 17 BB			LDX	#S9STR	
17A5	5D			TST B		
17A6	2B 0D			BMI	FNCHC	RETURN
17A8	8D 19			BSR	OAS	
17AA	5D			TST B		
17AB	27 08			BEQ	FNCHC	NOT CASSETTE
17AD	86 13			LDA A	#19	
17AF	4A		FNCHB	DEC A		
17B0	26 FD			BNE	FNCHR	
17B2	8D 73			BSR	OLT	
17B4	0C			CLC		NO ERRORS
17B5	39		FNCHC	RTS		
17B6	0D 0A 53	S1STR		FCB	CR,LF,'S1',0	
17BB	0D 0A 53	S9STR		FCB	CR,LF,'S9',0	
17C0	0D 0A 00	CRSTR		FCB	CR,LF,0	
			**	OAS - OUTPUT ASCII STRING		
			*			
			*	ENTRY: (X) = ADDRESS OF STRING IN FORM:		
			*	'STRING',0		
			*	(B) = CASSETTE/TERM FLAG		
			*	EXIT: X POINTS PAST END OF STRING ZERO		
			*	USES: X,A,C		
17C3	A6 00		OAS	LDA A	0,X	97 US
17C5	08			INX		
17C6	8D 49		OAS1	BSR	OAB	88 US
17C8	01			NOF		
17C9	86 10			LDA A	#16	208 US
17CB	4A		OAS2	DEC A		
17CC	26 FD			BNE	OAS2	
17CE	A6 00			LDA A	0,X	
17D0	08			INX		
17D1	6D 00			TST	0,X	
17D3	26 F1			BNE	OAS1	NOT LAST BYTE
17D5	08			INX		
17D6	20 39			BRA	OAB	OUTPUT LAST AND RETURN
			**	OSH - OUTPUT OPTIONAL SPACE WITH HEX BYTE		
			*			
			*	ENTRY: (X) = ADDRESS OF BYTE		
			*	(A) = CHECKSUM		
			*	(B) = CASSETTE/TERMINAL FLAG		
			*	EXIT: (X) INCREMENTED, (A) UPDATED		
			*	USES: X,A,C		
17D8	AB 00		OSH	ADD A	0,X	174 US
17DA	36			PSH A		

HEATH KEYBOARD MONITOR  
OUTPUT ROUTINES

17DB	86 05		LDA A	#5	
17DD	5D		TST B		
17DE	2A 09		BPL	OCH0	NO SPACE
17E0	BD 18 63		JSR	OUTSF	OUTPUT SPACE
17E3	32		FUL A		
		**		OCH - OUTPUT AND CHECKSUM HEX BYTE	
		*			
		*	ENTRY:	(X) = ADDRESS OF BYTE	
		*		(A) = CHECKSUM	
		*		(B) = CASSETTE/TERMINAL FLAG	
		*	EXIT:	(X) INCREMENTED, (A) UPDATED	
		*		'Z' SET IF END OF HEADER INFO	
		*	USES:	X,A,C	
17E4	AB 00	OCH	ADD A	0,X	174 US
17E6	36		PSH A		
17E7	86 06		LDA A	#6	
17E9	01	OCH0	NOF		
17EA	4A	OCH1	DEC A		
17EB	26 FD		BNE	OCH1	
17ED	A6 00		LDA A	0,X	
17EF	8D 06		BSR	OHB	
17F1	32		FUL A		
17F2	08		INX		
17F3	8C 00 F0		CPX	#T1+2	
17F6	39		RTS		16 US USED
		**		OHB - OUTPUT HEX BYTE	
		*			
		*	ENTRY:	(A) = BYTE	
		*		(B) = CASSETTE TERMINAL FLAG	
		*	USES:	A,C	
17F7	36	OHB	PSH A		112 US
17F8	44		LSR A		
17F9	44		LSR A		
17FA	44		LSR A		
17FB	44		LSR A		
17FC	8D 08		BSR	OHB2	
17FE	86 12		LDA A	#18	208 US
1800	4A	OHB1	DEC A		
1801	26 FD		BNE	OHB1	
1803	32		FUL A		
1804	84 0F		AND A	##F	
1806	81 0A	OHB2	CMP A	#10	
1808	24 02		BCC	OHB3	IS A - F
180A	20 03		BRA	OHB4	
180C	01	OHB3	NOF		
180D	8B 07		ADD A	#7	
180F	8B 30	OHB4	ADD A	##30	

HEATH KEYBOARD MONITOR  
OUTPUT ROUTINES

```

**      OAB - OUTPUT ASCII BYTE
*
*      ENTRY:  (A) = ASCII
*              (B) = CASSETTE/TERMINAL FLAG
*      EXIT:   (A) PRESERVED
*      USES:   C

1811 5D      OAB      TST B              80 US
1812 2F 51      BLE      OUTCH

**      OCB - OUTPUT CASSETTE BYTE
*
*      ENTRY:  (B) = CELLS/BIT COUNT
*              (A) = CHARACTER
*      USES:   C

1814 0C      OCB      CLC              START BIT; 74 US
1815 8D 27      BSR      BIT1          72 US
1817 36      PSH A          208 US
1818 0D      SEC              STOP BIT
1819 46      ROR A
181A 8D 1B      OCB1     BSR      BIT          200 US
181C 01      NOP              208 US
181D 44      LSR A
181E 26 FA      BNE      OCB1
1820 8D 15      BSR      BIT
1822 32      PUL A
1823 08      INX
1824 09      DEX              8 CYCLE PSEUDO-NOF
1825 20 10      BRA      BIT

**      OLT - OUTPUT LEADER TRAILER
*
*      ENTRY:  NONE
*      EXIT:   5 SECONDS MARKING WRITTEN
*      USES:   C

1827 0D      OLT      SEC              78 US
1828 36      PSH A
1829 8D 13      BSR      BIT1
182B 37      PSH B
182C C6 6E      LDA B      #110
182E 17      TBA
182F 8D 06      OLT1     BSR      BIT
1831 01      NOP
1832 4A      DEC A
1833 26 FA      BNE      OLT1
1835 33      PUL B
1836 32      PUL A

```

HEATH KEYBOARD MONITOR  
OUTPUT ROUTINES

```

**      BIT - OUTPUT 'C' TO CASSETTE
*
*      ENTRY:  (B) = CELL/BIT COUNT
*              'C' = BIT
*      USES:   C EXCEPT 'C'

1837  36      BIT      PSH A              192 US
1838  86 15      LDA A      #21
183A  01      NOP
183B  01      NOP
183C  20 03      BRA      BIT3              182 US

183E  36      BIT1     PSH A              64 US
183F  86 01      LDA A      #1
1841  37      BIT3     PSH B
1842  8C      FCB      #8C              3 CYCLE SKIP
1843  86 1D      BIT4     LDA A      #29
1845  4A      BIT5     DEC A
1846  26 FD      BNE      BIT5
1848  4C      INC A
1849  8D 10      BSR      FLIP              43 US
184B  86 1E      LDA A      #30
184D  4A      BIT6     DEC A
184E  26 FD      BNE      BIT6
1850  07      TPA
1851  84 01      AND A      #1              MASK TO CARRY
1853  8D 07      BSR      FLIP1
1855  5A      DEC B
1856  26 EB      BNE      BIT4
1858  33      PUL B
1859  32      PUL A
185A  39      RTS      ----- ALL TIMES REFERENCED HERE !!!

```

```

**      FLIP - FLIP CASSETTE BIT
*
*      ENTRY:  (A) = 0 THEN NO FLIP
*              (A) = 1 THEN FLIP
*      USES:   A,C EXCEPT 'C'

185B  01      FLIP     NOP              35 US
185C  B8 10 02  FLIP1  EOR A      TAPE
185F  B7 10 02      STA A      TAPE
1862  39      RTS              24 US

```

```

**      OUTSP - OUTPUT SPACE TO TERMINAL
*
*      ENTRY:  NONE
*      EXIT:   (A) = ' '
*      USES:   A,C

1863  86 20      OUTSP  LDA A      # ' '

```

HEATH KEYBOARD MONITOR  
OUTPUT ROUTINES

```

**      OUTCH - OUTPUT CHARACTER TO TERMINAL
*
*      ENTRY:  (A) = CHARACTER
*      EXIT:   (A) PRESERVED UNLESS --LF--
*      USES:   C

1865 36      OUTCH  PSH A
1866 37      PSH B
1867 8D 21    BSR      BRD      BAUD RATE DETERMINE
1869 0D      SEC      STOP BIT
186A 8D 32    BSR      WOB
186C 0C      CLC      START BIT
186D 8D 2F    BSR      WOB
186F 0D      SEC
1870 46      ROR A
1871 8D 2B    OUTC1  BSR      WOB      WAIT - OUTPUT BIT
1873 44      LSR A
1874 26 FB    BNE      OUTC1
1876 8D 26    BSR      WOB      WAIT; OUTPUT STOP
1878 33      PUL B
1879 32      PUL A
187A 81 0A    CMP A     #LF
187C 26 0B    BNE      OUTC2
187E 36      PSH A
187F 4F      CLR A
1880 8D E3    BSR      OUTCH     OUTPUT FILL CHARACTER
1882 8D E1    BSR      OUTCH
1884 8D DF    BSR      OUTCH
1886 8D DD    BSR      OUTCH
1888 32      PUL A
1889 39      OUTC2  RTS

**      BRD - BAUD RATE DETERMINATION
*
*      ENTRY:  NONE
*      EXIT:   (B) = BAUD RATE DIVISOR
*              (COMPENSATED FOR 5*13 EXTRA
*              EXECUTION TIME!!)
*      USES:   B,C

188A 36      BRD    PSH A
188B C6 01    LDA B     #1      ASSUME 110 BAUD
188D B6 10 00 LDA A     TERM     BAUD SWITCH DATA
1890 43      COM A
1891 84 0E    AND A     #1110B   MASK TO SWITCHES
1893 44      LSR A
1894 27 06    BEQ      BRD2     IS 110
1896 56      BRD1  ROR B
1897 4A      DEC A
1898 26 FC    BNE      BRD1
189A C0 05    SUB B     #5      EXECUTION COMPENSATION
189C 32      BRD2  PUL A
189D 39      RTS

```

HEATH KEYBOARD MONITOR  
OUTPUT ROUTINES

```

**      WOB - WAIT AND OUTPUT BIT
*
*      ENTRY:  (B) = DELAY COUNT
*              'C' = BIT
*      EXIT:  (B), 'C' PRESERVED
*      USES:  C

189E  37      WOB      PSH B
189F  8D 72      BSR      DLB          DELAY ONE BIT
18A1  20 68      BRA      WIB1

**      IHD - INPUT HEX DIGIT FROM TERMINAL
*
*      ENTRY:  NONE
*      EXIT:  (A) = HEX VALUE IF VALID
*              ASCII OTHERWISE
*              'C' SET IF HEX
*              'Z' SET IF CR
*      USES:  A,C

18A3  8D 3C      IHD      BSR      INCH
18A5  81 20      CMP A     #SPACE
18A7  27 FA      BEQ      IHD          IGNORE SPACES

**      ASH - ASCII TO HEX TRANSLATOR
*
*      ENTRY:  (A) = ASCII
*      EXIT, USES:  SEE "IHD"

18A9  80 30      ASH      SUB A     #'0'
18AB  25 0C      BCS      ASH1          NOT HEX
18AD  81 0A      CMP A     #10
18AF  25 10      BCS      ASH3
18B1  80 11      SUB A     #'A'-'0'
18B3  81 06      CMP A     #6
18B5  25 08      BCS      ASH2          IS HEX
18B7  8B 11      ADD A     #'A'-'0'      DISPLACE BACK
18B9  8B 30      ASH1     ADD A     #'0'
18BB  81 0D      CMP A     #CR
18BD  0C
18BE  39          CLC
          RTS

18BF  80 F6      ASH2     SUB A     #-10
18C1  39          ASH3     RTS

**      IHB - INPUT HEX RYTE
*
*      ENTRY:  (B) = CASSETTE/TERMINAL FLAG
*              (X) = ADDRESS
*              (A) = CHECKSUM
*      EXIT:  A, X UPDATED
*              (B) PRESERVED

```

HEATH KEYBOARD MONITOR  
INPUT ROUTINES

18C2	36	IHB	PSH A		SAVE CHECKSUM
18C3	8D 19		BSR	ICT	INPUT CASSETTE/TERMINAL
18C5	84 7F		AND A	#7FH	
18C7	8D E0		RSR	ASH	ASCII - HEX
18C9	48		ASL A		
18CA	48		ASL A		
18CB	48		ASL A		
18CC	48		ASL A		
18CD	97 EC		STA A	T0	
18CF	8D 0D		BSR	ICT	INPUT CASSETTE/TERMINAL
18D1	84 7F		AND A	#7FH	
18D3	8D D4		BSR	ASH	ASCII - HEX
18D5	9B EC		ADD A	T0	
18D7	A7 00		STA A	0,X	PLACE IN MEMORY
18D9	32		PUL A		
18DA	AB 00		ADD A	0,X	
18DC	08		INX		
18DD	39	IHB2	RTS		

```

**      ICT - INPUT FROM CASSETTE OR TERMINAL
*
*      ENTRY:  (B) = CASSETTE/TERMINAL FLAG
*      EXIT:   (A) = CHARACTER
*      USES:   A,C

```

18DE	5D	ICT	TST B		
18DF	2E 54		BGT	ICC	IS CASSETTE

```

**      INCH - INPUT TERMINAL CHARACTER
*
*      ENTRY:  NONE
*      EXIT:   (A) = CHARACTER
*      USES:   A,C

```

18E1	37	INCH	PSH B		
18E2	8D A6		BSR	BRD	BAUD RATE DETERMINE
18E4	17		TBA		
18E5	16	INC1	TAB		
18E6	54		LSR B		
18E7	5C		INC B		
18E8	7D 10 00	INC2	TST	TERM	
18EB	2B FB		BMI	INC2	WAIT FOR SPACING
18ED	8D 15		BSR	WIB	WAIT, INPUT START
18EF	25 F7		BCS	INC2	WAS NOISE
18F1	16		TAB		
18F2	86 80		LDA A	#80H	
18F4	8D 0E	INC3	BSR	WIB	WAIT; INPUT BIT
18F6	46		ROR A		
18F7	24 FB		BCC	INC3	
18F9	8D 09		BSR	WIB	GET STOP
18FB	25 03		BCS	INC4	NO FRAME ERROR
18FD	7C 10 00		INC	TERM	SEND STOP BIT
1900	84 7F	INC4	AND A	#7F	MASK TO SEVEN BITS
1902	33		PUL B		

HEATH KEYBOARD MONITOR  
INPUT ROUTINES

```

1903 39          RTS

**          WIB - WAIT AND INPUT BIT
*
*          ENTRY:  (B) = DELAY COUNT
*          EXIT:   'C' = BIT
*          USES:   C

1904 37          WIB    PSH B
1905 8D 0C          BSR    DLB          WAIT ONE BIT TIME
1907 CB 80          ADD B  #80H
1909 C0 80          SUB B  #80H
190B C9 00          WIB1  ADC B  #0          COPY BIT INTO LSB
190D F7 10 00      STA B  TERM
1910 56            ROR B          RESTORE SMASHED 'C'
1911 33            PUL B
1912 39            RTS

**          DLB - DELAY ONE BIT AND RETURN (TERM) IN B
*
*          ENTRY:  (B) = DELAY CONSTANT
*          EXIT:   (B) = (TERM) .AND. 11111110 B
*          USES:   C EXCEPT 'C'

1913 C5 FE          DLB   BIT B  #0FEH
1915 26 11          BNE    DLB4          NOT 110 BAUD
1917 5A            DEC B
1918 27 02          BEQ    DLB1          110 FULL BIT TIME
191A C6 38          LDA B  #56
191C C8 31          DLB1  EOR B  #49
191E 36            PSH A
191F 86 12          DLB2  LDA A  #18
1921 4A            DLB3  DEC A
1922 26 FD          BNE    DLB3
1924 5A            DEC B
1925 26 F8          BNE    DLB2
1927 32            PUL A
1928 BC 19 13      DLB4  CPX    DLB          5 CYCLE NUTHIN'
192B 01            NOP
192C 5A            DEC B
192D 26 F9          BNE    DLB4
192F F6 10 00      LDA B  TERM
1932 C4 FE          AND B  #FE
1934 39            RTS

**          ICC - INPUT CASSETTE CHARACTER
*
*          GETS BITS FROM CASSETTE IN SERIAL FASHION
*          EACH BIT CONSISTS OF SEVERAL 'CELLS'
*          EACH CELL IS EITHER 1/2 CYCLE OF 1200HZ
*                               OR 1 CYCLE OF 2400HZ
*          AT 8 CELLS/BIT THE ROUTINE IS 'KCS'
*          COMPATIBLE

```

HEATH KEYBOARD MONITOR  
INPUT ROUTINES

```

*
*   ENTRY: (B) = CELLS PER BIT
*   EXIT:  (A) = CHARACTER
*           'C' = STOP BIT
*
*   USES:  A,C

1935 37      ICC      PSH B
1936 54      LSR B
1937 8D 1E   ICC1    BSR      TNC      TAKE NEXT CELL
1939 25 FC   BCS      ICC1    NOT START BIT
193B 5A      DEC B
193C 2A F9   BFL      ICC1    NOT ENOUGH CELLS
193E 33      PUL B
193F 86 7F   LDA A    #01111111B  PRESET ASSEMBLY
1941 37      ICC2    PSH B
1942 36      PSH A
1943 8D 12   ICC3    BSR      TNC      TAKE NEXT CELL
1945 5A      DEC B
1946 26 FB   BNE      ICC3
1948 32      PUL A
1949 33      PUL B
194A 46      ROR A
194B 25 F4   BCS      ICC2
194D 37      PSH B
194E 36      PSH A
194F 8D 06   ICC4    BSR      TNC      GET STOP BIT
1951 5A      DEC B
1952 26 FB   BNE      ICC4
1954 32      PUL A
1955 33      PUL B
1956 39      RTS

```

```

**   TNC - TAKE NEXT CELL
*
*   WAITS FOR 1/2 CYCLE OF 1200 HZ OR
*           1 CYCLE OF 2400 HZ
*   STRUCTURE ASSURES EXIT AT END OF
*   ZERO CELL
*
*   ENTRY:  NONE
*   EXIT:   'C' = CELL VALUE
*           (A) = NEW CASSETTE DATA
*
*   USES:   A,C

```

```

1957 B6 10 02 TNC      LDA A    TAPE
195A 8D 02      BSR      TNC1
195C 24 0E      BCC      TNC3      WAS ZERO
195E 37      TNC1    PSH B
195F 5F      CLR B
1960 5C      TNC2    INC B
1961 B1 10 02  CMP A    TAPE
1964 27 FA      BEQ      TNC2      NO TRANSITION
1966 B6 10 02  LDA A    TAPE
1969 C1 1D      CMP B    #29
196B 33      PUL B

```

HEATH KEYBOARD MONITOR  
INPUT ROUTINES

```

196C 39          TNC3  RTS

**          MOVE - REENTRANT MOVE MEMORY
*
*          ENTRY:  STACK>  RETURN (0,S)
*                  COUNT  (2,S)
*                  TO      (4,S)
*                  FROM    (6,S)
*          EXIT:   STACK CLEANED
*          USES:   ALL

196D 30          MOVE   TSX
196E EE 02       LDX    2,X          CHECK COUNT < 0
1970 27 74       BEQ    MOV4        NO MOVE
1972 30          MOVEA  TSX          ** ALTERNATE ENTRY **
1973 A6 05       LDA   A    5,X      (BA) = TO
1975 E6 04       LDA   B    4,X
1977 A0 07       SUB   A    7,X      (BA) = TO - FROM
1979 E2 06       SRC   B    6,X
197B 25 24       BCS   MOV2        IS MOVE DOWN
197D 26 03       RNE   MOV1
197F 4D          TST   A
1980 27 64       BEQ    MOV4        DISPLACEMENT = 0

*          HAVE MOVE UP - MUST START AT TOP
*          TO AVOID CONFLICT

1982 86 FF       MOV1  LDA   A    #-1      (BA) = -1
1984 16          TAB
1985 36          PSH   A            DELTA = -1
1986 37          PSH   B
1987 AB 03       ADD   A    3,X      (BA) = COUNT - 1
1989 E9 02       ADC   B    2,X
198B 36          PSH   A
198C 37          PSH   B
198D AB 05       ADD   A    5,X      TO = TO + COUNT - 1
198F E9 04       ADC   B    4,X
1991 A7 05       STA   A    5,X
1993 E7 04       STA   B    4,X
1995 33          FUL   B
1996 32          FUL   A
1997 AB 07       ADD   A    7,X      FROM = FROM
1999 E9 06       ADC   B    6,X      + COUNT - 1
199B A7 07       STA   A    7,X
199D E7 06       STA   B    6,X
199F 20 0E       BRA   MOV3

*          HAVE MOVE DOWN - MAY START AT TOP

19A1 86 01       MOV2  LDA   A    #1      DELTA = 1
19A3 5F          CLR   B
19A4 36          PSH   A
19A5 37          PSH   B
19A6 4F          CLR   A
19A7 A0 03       SUB   A    3,X      (BA) = -- COUNT

```

HEATH KEYBOARD MONITOR  
MOVE - MOVE SUBROUTINE

19A9	E2 02		SBC B	2,X	
19AB	A7 03		STA A	3,X	
19AD	E7 02		STA B	2,X	COUNT = - COUNT
		*	ACTUAL MOVE LOOP FOLLOWS		
19AF	30	MOV3	TSX		
19B0	EE 08		LDX	8,X	
19B2	A6 00		LDA A	0,X	
19B4	30		TSX		
19B5	EE 06		LDX	6,X	
19B7	A7 00		STA A	0,X	
19B9	30		TSX		
19BA	A6 01		LDA A	1,X	BUMP *FROM*
19BC	E6 00		LDA B	0,X	
19BE	AB 09		ADD A	9,X	
19C0	E9 08		ADC B	8,X	
19C2	A7 09		STA A	9,X	
19C4	E7 08		STA B	8,X	
19C6	A6 01		LDA A	1,X	BUMP *TO*
19C8	E6 00		LDA B	0,X	
19CA	AB 07		ADD A	7,X	
19CC	E9 06		ADC B	6,X	
19CE	A7 07		STA A	7,X	
19D0	E7 06		STA B	6,X	
19D2	A6 01		LDA A	1,X	BUMP *COUNT*
19D4	E6 00		LDA B	0,X	
19D6	AB 05		ADD A	5,X	
19D8	E9 04		ADC B	4,X	
19DA	A7 05		STA A	5,X	
19DC	E7 04		STA B	4,X	
19DE	26 CF		BNE	MOV3	COUNT <> 0
19E0	4D		TST A		
19E1	26 CC		BNE	MOV3	
19E3	31		INS		
19E4	31		INS		DISCARD DELTA
19E5	30		TSX		
19E6	EE 00	MOV4	LDX	0,X	
19E8	31		INS		
19E9	31		INS		
19EA	31		INS		
19EB	31		INS		
19EC	31		INS		
19ED	31		INS		
19EE	31		INS		
19EF	31		INS		
19F0	6E 00		JMP	0,X	

HEATH KEYBOARD MONITOR  
TABLES

		**	COMMAND TABLE		
19F2	54	CMDTAB	FCB	'T'	TAPE RECORD DATA
19F3	17 07		FDB	RCRD	
19F5	53		FCB	'S'	STEP USER CODE
19F6	15 50		FDB	STEP	
19F8	52		FCB	'R'	DISPLAY USER REGISTERS
19F9	15 53		FDB	REGS	
19FB	50		FCB	'P'	PUNCH TO PAPER TAPE
19FC	17 0A		FDB	PTAP	
19FE	4D		FCB	'M'	DISPLAY MEMORY (BYTE)
19FF	15 AC		FDB	MEM	
1A01	4C		FCB	'L'	LOAD FROM TAPE
1A02	16 A5		FDB	LOAD	
1A04	49		FCB	'I'	DISPLAY MEMORY (INST)
1A05	15 AD		FDB	INST	
1A07	48		FCB	'H'	HALTPOINT INSERT
1A08	14 F3		FDB	BKPT	
1A0A	47		FCB	'G'	GO TO USER CODE
1A0B	14 8F		FDB	GO	
1A0D	45		FCB	'E'	MULTIPLE STEP
1A0E	15 39		FDB	EXEC	
1A10	44		FCB	'D'	DUMP MEMORY
1A11	17 09		FDB	DUMP	
1A13	43		FCB	'C'	BREAKPOINT CLEAR
1A14	15 12		FDB	CLEAR	
1A16	42		FCB	'B'	GO TO BASIC WARM START ENTRY
1A17	1C 03		FDB	1C03H	
1A19	18		FCB	'X'--40H	DISPLAY INDEX
1A1A	15 8E		FDB	REGX	
1A1C	14		FCB	'T'--40H	HI SPEED TAPE
1A1D	16 FA		FDB	CYLT	
1A1F	13		FCB	'S'--40H	SLIDE MEMORY!!
1A20	16 7B		FDB	COPY	
1A22	10		FCB	'P'--40H	DISPLAY P.C.
1A23	15 8C		FDB	REGP	
1A25	08		FCB	'H'--40H	HALTPOINT LIST
1A26	15 FB		FDB	DISB	

HEATH KEYBOARD MONITOR  
TABLES

1A28	03		FCB	'C'--40H	DISPLAY CONIX
1A29	15	92	FDB	REGC	
1A2B	02		FCB	'B'--40H	DISPLAY B ACC.
1A2C	15	91	FDB	REGB	
1A2E	01		FCB	'A'--40H	DISPLAY A ACC.
1A2F	15	90	FDB	REGA	
1A31	00		FCB	'@'--40H	EXIT TO OLD MONITOR
1A32	FC	00	FDB	\$FC00	

```

**      MTST - MEMORY DIAGNOSTIC
*
*      DISPLAYS LWA IN 'ADDR' FIELD ON LEDS
*      CURRENT TEST PATTERN IN 'DATA'
*      ENTRY:  NONE
*      EXIT:   FAILED ADDRESS/PATTERN DISPLAYED
*      PROCESSOR HALTED
*      USES:  ALL,T0,T1,DIGADD
    
```

1A34	0F		MTST	SEI	
1A35	8D	49		BSR	F10P
1A37	35			TXS	FIND TOP OF MEMORY STACK AT TOP
1A38	31			INS	
1A39	6F	00	MTS2	CLR	0,X
1A3B	09			DEX	
1A3C	26	FB		BNE	MTS2
1A3E	6F	00		CLR	0,X
1A40	9F	EE		STS	T1
1A42	8E	00	EB	LDS	#T0-1
1A45	BD	FC	BC	JSR	REDDIS
1A48	CE	00	EE	LUX	#T1
1A4B	C6	02		LDA	B #2
1A4D	BD	FD	7B	JSR	DISPLAY
1A50	4F			CLR	A
1A51	5A			DEC	B
1A52	BD	FE	20	MTS3	JSR
1A55	36			FSH	A
1A56	BD	FD	43	JSR	BKSP
1A59	32			PUL	A
1A5A	DE	EE		LIX	T1
1A5C	A1	00	MTS4	CMF	A 0,X
1A5E	26	13		BNE	MTS6
1A60	6C	00		INC	0,X
1A62	09			DEX	
1A63	8C	00	F1	CPX	#DIGADD+1
1A66	26	03		BNE	MTS5
1A68	CE	00	DF	LIX	#T0-13
1A6B	8C	FF	FF	MTS5	CPX #-1
1A6E	26	EC		BNE	MTS4
1A70	4C			INC	A
1A71	20	DF		BRA	MTS3
1A73	DF	EE	MTS6	STX	T1

HEATH KEYBOARD MONITOR  
MEMORY DIAGNOSTIC

1A75	BD FC BC		JSR	REDIS	RESET DISPLAYS
1A78	CE 00 EE		LDX	#T1	
1A7B	5C		INC B		
1A7C	BD FD 7B		JSR	DISPLAY	
1A7F	3E		WAI		
		**	FTOP - FIND MEMORY TOP		
		*			
		*	SEARCHES DOWN FROM 1000H UNTIL FINDS		
		*	GOOD MEMORY		
		*			
		*	ENTRY:	NONE	
		*	EXIT:	(X) = LWA MEMORY	
		*	USES:	X	
1A80	36	FTOP	PSH A		
1A81	CE 10 00		LDX	#TERM	TOP OF MEMORY+1
FFFF			IF	DEBUG-1	
			ENDIF		
1A84	86 55		LDA A	#55H	TEST PATTERN
1A86	09	FTO1	DEX		
1A87	A7 00		STA A	0,X	
1A89	A1 00		CMP A	0,X	
1A8B	26 F9		RNE	FTO1	
1A8D	32		PUL A		
1A8E	39		RTS		
		**	CCD - CONSOLE CASSETTE DUMP		
		*			
		*	ENTRY:	NONE	
		*	EXIT:	TO LED MONITOR	
		*	USES:	ALL,T0,T1,T2	
1A8F	C6 08	CCD	LDA B	#8	
1A91	8D 42		BSR	IN.PIA	INITIALIZE PIA
1A93	8E 00 EB		LDS	#T0-1	
1A96	37		PSH B		
1A97	BD FC 86		JSR	OUTSTA	
1A9A	47 85		FCB	47H,05H+80H	'FR'
1A9C	CE 00 EE		LDX	#T1	
1A9F	C6 02		LDA B	#2	
1AA1	BD FC BC		JSR	REDIS	RESET DISPLAYS
1AA4	BD FD 25		JSR	PROMPT	PROMPT FWA
1AA7	BD FC 86		JSR	OUTSTA	
1AAA	0E FD		FCB	0EH,7DH+80H	'LA'
1AAC	BD FC BC		JSR	REDIS	RESET DISPLAYS
1AAF	CE 00 F4		LDX	#T2	
1AB2	BD FD 25		JSR	PROMPT	PROMPT LWA
1AB5	33		PUL B		
1AB6	BD 17 29		JSR	PUNCH	
1AB9	7E FC 00	CCD1	JMP	#FC00	EXIT TO MONITOR

HEATH KEYBOARD MONITOR  
LED MONITOR TAPE PROCESSORS

```

**      CCL -- CONSOLE CASSETTE LOAD
*
*      ENTRY:  NONE
*      EXIT:   TO CONSOLE MONITOR IF SUCCESS
*      USES:   ALL, TO, HIGHEST MEMORY

1ABC  C6 08      CCL      LDA B      #8
1ABE  8D 15      BSR      IN.PIA      INITIALIZE PIA
1AC0  8D BE      BSR      FTOP        FIND MEMORY TOP
1AC2  35
1AC3  31
1AC4  BD 16 AD    JSR      LOAO          LOAD MEMORY
1AC7  24 F0      BCC      CCD1          NORMAL RETURN
1AC9  BD FC BC    JSR      REDIS          PRINT ERROR MESSAGE
1ACC  BD FE 52    JSR      OUTSTR
1ACF  4F 05 05    FCB      4FH,05H,05H,10H,05H+80H
1AD4  3E      WAI

```

```

**      IN.PIA -- INITIALIZE PIA FOR LED MONITOR
*
*      INITIALIZE CASSETTE SIDE FOR LOAD OR DUMP
*      AND SET (TERM) SO THAT A BREAK IS NOT
*      SENSED.
*
*      ENTRY:  NONE
*      EXIT:   NONE
*      USES:   A,X

1AD5  CE 10 00    IN.PIA  LDX      #TERM
1AD8  6F 01      CLR      1,X
1ADA  6F 03      CLR      3,X
1ADC  86 80      LDA A    #10000000B
1ADE  A7 00      STA A    0,X          INTO DDR
1AE0  43
1AE1  A7 02      STA A    2,X          INITIALIZE CASSETTE
1AE3  86 04      LDA A    #4
1AE5  A7 03      STA A    3,X
1AE7  39      RTS

```

LON L

```

**      TTST -- TERMINAL TESTER
*
*      ENTRY:  NONE
*      EXIT:   NEVER

1AF6  86 01      TTST   LDA A    #1
1AF8  B7 10 00    STA A    TERM
1AFB  C6 04      LDA B    #4
1AFD  F7 10 01    STA R    TERM.C
1B00  BD 16 18    TTST   JSR      OUTIS
1B03  0D 0A 54    FCB      CR,LF,'THIS IS A TERMINAL TEST',0
1B1D  20 E1      BRA     TTST

```

HEATH KEYBOARD MONITOR  
TERMINAL TEST

1B1F

END

STATEMENTS =1632

FREE BYTES =16823

NO ERRORS DETECTED

## APPENDIX D

### *Excerpts from “Kilobaud”*

The following magazine articles have been reproduced with permission from Kilobaud. They provide entertaining and educational material that enables you to more fully appreciate and enjoy your ETA-3400 microcomputer accessory.

The programs will not necessarily run as is on your computer accessory, but with some modifications you can run the programs.

# Tiny Basic

ME THINK A MOMENT . . ." and that is what seems to be happening.

I've made my Hunt the Hurkle game a little more interesting for a first-time player by including a random 1 out of 15 chance of seeming confusion on the part of the computer. The result is that instead of the normal THE HURKLE IS HIDING message, the printout is as shown in Example 3.

Issue #1 of *Kilobaud* contained an article by Tom Pittman describing his Tiny BASIC. As a very optimistic owner of a new KIM-1, and with a SWTP CT-1024 TV terminal on order, I sent my order off to Tom's Itty Bitty Computer Company, and soon my Tiny BASIC listing arrived. Lacking the terminal, I spent a Saturday loading Tiny by hand with the hex keyboard and verifying it. When the last kit of the TV terminal arrived, I loaded Tiny. A close reading of the instructions indicated that I

ways to save memory:

1. PRINT may be abbreviated PR in all cases. For example:

```
50 PR "HI THERE!"
```

2. Tiny needs no spaces in the program statements. A listing is hard to read without them, but it is better than running out of memory.

3. Tiny has no absolute value function. This can be implemented easily as follows:

```
100 IF A < 0 A=-A
```

4. Tiny has no ON N GOTO statement (see Example 1).

```
150 ON N GOTO (100,110,120,130)
```

Example 1.

had to insert some I/O jump addresses. This done, Tiny ran with nothing more than operator problems.

It was not hard to begin programming some of the simpler games from *Basic Computer Games* published by Digital Equipment Corp.

As limited as it is, using only 2½K of memory (I had added an Econoram 4K expansion to my KIM), a great deal can be done with it that is not obvious on first glance.

At the bargain price of \$5 I didn't expect a full course in BASIC programming. But there are some features that are not obvious and could be expanded upon for those of us who are rank beginners.

First, here are a couple of

The following allows the same results:

```
60 GOTO 100+10*N
```

This is particularly useful in implementing a game like Bombers (see *Basic Computer Games*). Here the player is given a multiple choice, and the number he enters (N) determines a branch in the program.

My TV typewriter is the kind that "pages"; when the

```
110 IF T < (RND(150)+10) GOTO 105
115 RETURN
```

Example 2.

screen fills, it "flips" a page and starts to fill it from the top. If output such as instruc-

```
THE HURDLE IS HIKING (pause random time)
NO, THAT'S NOT RIGHT (pause random time)
THE HIDE IS HURKING. (pause random time)
NOW WAIT A MINUTE! (pause random time)
THE HURKLE IS HIDING.
```

Example 3.

tions extends to more than one full page, it is lost before it can be read. This would also be a problem with a scrolling display, particularly if the TVT is running at 1200 baud. The program can contain a "pause for read" which can be implemented easily at

Here the program resumes its regular course.

Last but not least, Tiny BASIC lacks any kind of string manipulation. It is possible to get around this by using Y and N for Yes and No responses as shown in Example 4.

```
50 PR "WANT TO PLAY AGAIN?";
60 Y=1
70 N=0
80 INPUT R
85 REMARK R FOR RESPONSE
90 IF R=1 GOTO 10
100 PR "THANKS FOR PLAYING. HOPE YOU ENJOYED IT"
999 END
```

Example 4.

the desired point.

```
100 T=0
105 T=T+1
110 IF T < 150 GOTO 105
```

The T less-than number may be adjusted for a suitable time delay. These steps may be a subroutine, and T may be randomized by Example 2.

A little ingenuity allows many tricks in Tiny BASIC. Use a little imagination, and it can be great fun.

I started out in this hobby with full intentions never to waste time playing games with my computer. Obviously I've changed my mind. The reason is that programming games seems to be a very good way to learn all the tricks and non-tricks of programming in BASIC. I still intend to do a lot of machine language programming, but I can't imagine a way to learn BASIC faster than by using it to program a game. Thanks, Tom Pittman, for Tiny BASIC. It really works. ■

The delay loop is used to add interest to a game, where the computer outputs "LET

Along with pointing out the differences between Tiny BASIC and standard BASIC, Tom offers here some comments and opinions on BASIC and structured programming. Interestingly, his manuscript is one of the few we've received which was prepared using a text editor (a Model 37 TTY driven by a COSMAC 1802 microprocessor). It would seem that more of us (including myself) should be at this stage by now. — John.

# Tiny Basic

## ... a mini-language

Tom Pittman  
PO Box 23189  
San Jose CA 95153

## for your micro

If you have an Altair or IMSAI computer or any 8080-based system, you have your choice of several versions of BASIC. There are rumors of BASIC for 6800 and 6502 within the next few months. But these require memory — probably more than you have with your low budget machine.

The alternative is *Tiny BASIC*. The language is a stripped down version of regular BASIC, with integer variables only — no strings, no arrays, and a limited set of statement types. It was first proposed by Bob Albrecht, the "dragon" of Peoples Computer Company (PCC) in Menlo Park, as a language for teaching programming to children. The PCC newspaper ran a series of articles (largely written by Dennis Allison) entitled "Build Your Own BASIC," suggesting how Tiny BASIC might be implemented in a microprocessor. The important portions of these articles have been reprinted in Dr. Dobb's *Journal of Computer Calisthenics and Orthodontia*, published by PCC and available in most computer stores.

### BASIC

Before we get into Tiny BASIC, let us look at high level languages in general and BASIC in particular.

When you program in machine language, each command, or statement, represents one operation from the machine's point of view. When we think of a single concept like, "A is the sum of B and C," a machine language program to perform this operation may take several operations, such as:

```
LDA B
LDA C
STO A
```

A high level language, on the other hand, lets you put a single human idea into a single program statement, for instance:

```
LET A = B + C
```

BASIC is one of a class of "algebraic" languages in that it permits the representation of algebraic formulae as part

of the language. Other languages in this class are FORTRAN and ALGOL. COBOL does not generally fall in this class (except for the "super" versions).

Of critical importance to all algebraic languages is the concept of an expression. An expression is the programming language notation for what we might think of as "the right-hand side of a formula." Alternatively, we can think of an expression as "a way of expressing the value of some number which the computer is to compute."

An expression may consist of a single number, a single variable name (all variables are referred to by name in high level languages), a single function call (discussed in detail later), or some combination of these, separated by operators and possibly grouped by parentheses. For this discussion, when we refer

to an operator, we mean one of the four functions found on a cheap pocket calculator: addition symbolized by "+"; subtraction by "-"; multiplication by "\*"; and division by "/". (The usual symbol for division does not appear on most typewriter and computer keyboards.) Thus,

$$\frac{A-B}{C-D}$$

becomes, in computerese,

$$(A - B) / (C - D)$$

Here the parentheses are used to indicate priority of operations. Normally multiplication and division are performed first, then addition and subtraction. Without the parentheses the expression,

$$\frac{A-B}{C-D}$$

would be understood by the high level language as,

$$a - \frac{B}{C} d$$

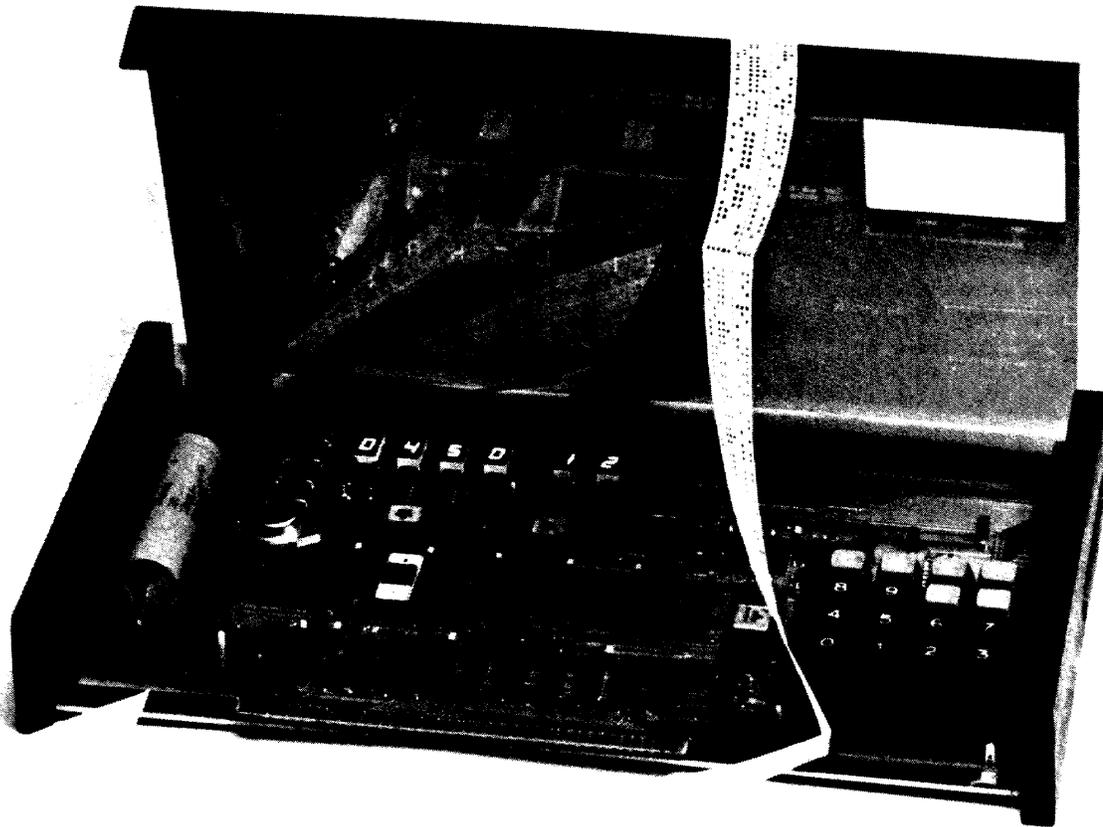


Photo courtesy of Electronic Product Associates, Inc., 1157 Vega Street, San Diego CA 92110.

which is not the same at all.

In BASIC, when an expression is encountered, it is evaluated. That is, the values of the variables are fetched, the numbers are converted (if necessary), the functions are called, and the operations are performed. The evaluation of an expression always results in a number which is defined to be the value of that expression.

The first example which we discussed showed a simple BASIC statement,

```
LET A=B+C
```

This is called an assignment statement, because it assigns the value of the expression "B + C" to the variable A. All algebraic high level languages have some form of assignment statement. They are characterized by the fact that when the computer processes an assignment statement, a single named variable is given a new value. The new value may not necessarily be

different from the old; for example:

```
LET A=A
```

This is also a valid assignment statement, even though nothing changes. Assignment statements are also used to put initial values into variables, for instance:

```
LET P=3
```

### Control Structures

One of the important characteristics distinguishing different high level languages is the control structure afforded to the programmer. The control structure is determined by the various permitted control statements, which alter the flow of program execution. Normally program execution advances from statement to statement in sequence, although there are however, circumstances in which this sequence is altered. The most common control structure allows one set of operations to be per-

formed if a certain condition is true, and another, if it is false. In "structured programming" this is referred to as the "IF... THEN... ELSE" construct; its general form is "IF condition is true, THEN do something, ELSE do some other thing." The full generality of this control structure is not directly available in BASIC, but, as we shall see, this is only a minor inconvenience.

Standard BASIC uses the IF... THEN construct, and makes it work something like a conditional GOTO:

```
IF A>3 THEN 120
```

If the value of the variable A is greater than three, then (GOTO) line 120, otherwise continue with the next statement in sequence. Actually, the condition to be tested consists of a comparison between two expressions, using any of the comparison operators which are given in Fig. 1.

In each case, if the comparison of the two expressions evaluates as true, the implied GOTO is taken; otherwise the next statement in sequence is executed. In Tiny BASIC the syntax is slightly different. Instead of a statement number, a whole statement follows the THEN part of the IF... THEN. The comparison above, in Tiny BASIC, would be:

```
IF A>3 THEN GOTO 120
```

But we could also validly write:

```
IF A<=3 THEN LET A=A+10
```

or some such. Note that this is *not* valid in standard BASIC.

The GOTO construct has been the subject of controversy in the last few years. A strong case has been made for "GOTO-less programming" which uses only certain other control structures to achieve structured programs which are more readable and less

=	Equality (the comparison is true if the two expressions are equal)
>	Greater than
<	Less than
< =	Less or Equal (not Greater)
> =	Greater or Equal
< >	Not Equal

Fig. 1. Comparison Operators.

prone to errors. I believe that both good and incomprehensible programs are possible regardless of the control structures used or not used, but I seem to be in a minority at this time. Suffice to say that BASIC is not conducive to structured programming in the technical sense of the term.

Standard BASIC has one control structure which has been omitted from Tiny BASIC. This is the FOR ... NEXT loop. Normally, if a program requires some sequence to be performed thirteen times, the following program steps might be used:

```
10 FOR I=1 TO 13
20 ...
30 NEXT I
```

Statement 20 would be executed 13 times, with the variable I containing successively the values, 1, 2, 3 ... 12, 13. In Tiny BASIC the same operation is a little more verbose:

```
10 LET I=1
20 ...
30 LET I=I+1
40 IF I<=13 THEN GOTO 20
```

but, as you can see, nothing is lost in program capability.

### Data Structures

Standard BASIC also has some data structures which have not been carried over into Tiny BASIC. The only data structure in Tiny BASIC is the integer number, which is further limited to 16 binary bits for a value in the range of -32768 to +32767. Compare this precision with the six

digit precision in standard BASIC, which also gives you fractional numbers (sometimes called "floating point"). Regular BASIC allows arrays, or variables with multiple values distinguished by "subscripts," and strings, which are variables with text information for values instead of numbers. We will see presently how these deficiencies in Tiny BASIC can be overcome.

### Input/Output

Thus far we have said nothing about input and output, how to see the answers the computer has calculated, or how to put in starting values. These needs are accommodated in BASIC by the PRINT and INPUT statements. Numbers are printed (in decimal, for us humans to read) at the user terminal by the PRINT statement:

```
PRINT A, B + C
```

This prints two numbers; the first is the value of the variable A, and the second is the value of the expression B+C. In general, the PRINT statement evaluates and prints expressions. It is perfectly valid to write

```
PRINT 1, 123, 0-0
```

although we know in advance what will be displayed on the terminal. To make our output more readable, BASIC permits the program to print out text labels on the data.

```
PRINT "THE SUM OF 1 + 2 IS", 3 + 2
```

will display the line:

```
THE SUM OF 1 + 2 IS 5
```

To feed new numbers from the terminal to the pro-

gram the INPUT statement is used.

```
INPUT A, B, C
```

will request three numbers from the input keyboard. The more popular versions of Tiny BASIC have an extra capability here beyond standard BASIC, in that the operator can type in numbers *and* whole expressions. Thus, in response to the INPUT request above, the operator types

```
1+2, 3*(4+5), B-A
```

the variable A will receive the value 3, B will receive the value 27, and C will receive the value 24 = 27-3. Therefore, a program in Tiny BASIC, which permits no text strings, can display and accept as input limited text information:

```
10 LET Y=1
20 LET N=0
30 PRINT "PLEASE ANSWER Y OR N".
40 INPUT A
50 IF A=Y THEN GOTO 100
60 IF A=N THEN GOTO 120
70 GOTO 30
```

This little program asks for an answer, which should be either the letter "Y" or the letter "N" (or their equivalents, the numbers 1 or 0, respectively). If the operator types anything else, the request is repeated. Obviously, this technique will not work for something like a person's name where any letters of the alphabet in any sequence must be expected, but it is certainly an improvement over no alphabetic input at all.

A generalized text output capability in Tiny BASIC depends on another characteristic peculiar to Tiny BASIC and not shared by standard. That is the fact that the line number in a GOTO or GOSUB statement is not limited to numbers only, but may itself be any valid expression which evaluates to a line number. The program which is shown in Fig. 2 prints A, B, or C, depending on whether the variable N has the value 1, 2, or 3. Note that, if N is out of range, nothing is printed.

### The USR Function

What about the fact that

there are no arrays? Let us turn to the USR function for a way to store and retrieve blocks of data. The remarks which follow apply only to my version of Tiny BASIC and are unique in that respect.

The USR function is invoked with one, two, or three arguments (expressions separated by commas within the parentheses). The first (or only) argument is evaluated to the binary address of a machine language subroutine somewhere in the computer memory. The USR function does a machine language subroutine call (JSR instruction) to that address. The user is obliged to be sure that there is in fact a subroutine at that address. If there is not, Tiny BASIC (and thus your computer) will execute whatever is there. The second and third arguments, if present, will be loaded into the CPU registers before jumping to this subroutine. On exit, any answer the subroutine produces may be left in the CPU accumulator, and it becomes the value of the function. Two machine language routines are already provided with the BASIC Interpreter; if S is the address of the beginning of the interpreter,

```
USR(S + 20, M)
```

has as its value the byte stored in memory at the address in the variable M (that is, the contents of the second argument is evaluated to a memory address). Also,

```
USR(S + 24, M, B)
```

stores the low order 8 bits of the value of B into the memory location addressed by M. The return value of this function is meaningless.

Consider the standard BASIC program in Fig. 3(a) to input ten numbers and print the largest as compared to the Tiny BASIC program in Fig. 3(b).

I have used this example for two reasons: First, it shows how the USR function may be used to simulate the operation of arrays. Second, it is typical of many of the applications commonly ad-

```

10 IF N>0 THEN IF N<4 THEN GOSUB 20+(N * 10)
20 RETURN
30 PRINT "A"
35 RETURN
40 PRINT "B"
45 RETURN
50 PRINT "C"
55 RETURN

```

Fig. 2. Program to Print A, B, or C, depending on the value of N.

to argue for arrays; however, neither real nor simulated arrays are required for this program! Here is the same program, with no arrays:

```

10 LET I=1
20 LET L=0
30 INPUT V
40 IF L<V THEN LET L=V
50 LET I=I+1
60 IF I<=10 THEN GOTO 30
90 PRINT L

```

### Summary

Tiny BASIC is not a super language. But, it also does not require a super computer to run. I've given here only a cursory examination of the power of Tiny BASIC. A full description of Tiny BASIC may be found in the Itty

Bitty Computers *Tiny BASIC User's Manual*. This comes with a hex paper tape of the program and is available for \$5 from: Itty Bitty Computers, PO Box 23189, San Jose CA 95153.

There are different versions for each of the following systems, so be sure to specify which system you are running:

M6800 with MIKBUG, EXBUG, or home brew (Executes in 0100-08FF); AMI Proto board (Executes in E000-E7FF); SPHERE (Executes in 0200-09FF); 6502 with KIM, TIM or homebrew (Executes in

0200-0AFF); JOLT (Executes in 1000-18FF); APPLE (Executes in 0300-0BFF); KIM-2 4K RAM (executes in 2000-28FF).

Although few people have paper tape systems, we are unable to provide the program on audio cassette. But if you request it, we will supply a hexadecimal listing of the program instead of tape which you can key in and then can save on cassette for future use.

If you have a small 8080 system, there are several widely differing versions of Tiny BASIC in the public

domain. Most of them have been published in Dr. Dobb's Journal, which is \$10 per year from: People's Computer Company, PO Box 310, Menlo Park CA 94025. This journal has also published a number of games which run in Tiny BASIC.

One final comment. Tiny BASIC was originally conceived as "free software" by the people at PCC. The 6800 and 6502 versions described in this article are not free; they are proprietary and copyrighted. Software is my only source of income, and, if I cannot make it from programs like Tiny BASIC, I won't write them. Please respect the labor of those of us who are trying to make quality software available to you: pay for the programs you use. ■

Fig. 3. Programs to input ten numbers and print the largest. (a) Standard BASIC; (b) Tiny BASIC.

```

10 FOR I=1 TO 10
20 INPUT V(1)
30 NEXT I
40 LET L=V(1)
50 FOR I=2 TO 10
60 IF L>=V(I) THEN 80
70 LET L=V(I)
80 NEXT I
90 PRINT L

```

```

10 LET I=1
20 INPUT V
25 LET V=USR(S=24,I,V)
30 LET I=I+1
35 IF I<=10 THEN GOTO 20
40 LET L=USR(S+20,I)
50 LET I=2
60 IF L<USR(S+20,I) THEN LET L=USR(S+20,I)
80 LET I=I+1
90 PRINT L

```

Bitty Computers, PO Box  
23189, San Jose CA 95153.)

These are not significant handicaps if you're estimating the effect of several alternatives. Round numbers are usually acceptable if you only want to get on base in some specific ball park (cliches are fun once in a while).

### Byte-saving Tips

Saving bytes of memory is a practical approach if your computer has limited memory (I have 1250 bytes of free space now). Let's talk about the memory-saving part first.

Fig. 1 is an example of a program with no statement shortcuts; Fig. 2 uses all the implied and abbreviated statements possible in this Tiny BASIC interpreter. Memory in Fig. 1 is 492 bytes, an average of 17 bytes per line, while Fig. 2 uses 410 bytes for an average of 14 bytes per line. REM comments were added later and used 470 bytes.

Using implied statements causes the program to run

# Tiny BASIC Shortcuts

*Tom Pittman's Tiny BASICs (6502, 1802, etc.) are somewhat limited in capabilities. This is the first of several articles discussing methods to expand those capabilities.*

Charles R. Carpenter  
2228 Montclair Place  
Carrollton TX 75006

**W**riting small but useful programs in Tiny BASIC (to paraphrase Tom Pittman) is a practical reality. Getting the most out of your programs is easier if you work with the inter-

preter's limitations. The utility program in Fig. 1 shows how to work with some of these limitations. This program is titled "Loans," but it could be any comparison of WHAT-IF alternatives. Here's what we'll be working with (and without):

- Decimal numbers not allowed.
- Number range limited from -32768 to +32767.

- 72 characters maximum on Input lines.

- Implied statements and abbreviations to save bytes of memory.

(Note: Tom Pittman now has an experimenter's manual available that explains many of these features and how to work with them. They are not as simple as my approach. The manual is available from Itty

```

:
:LIST
10 REM TINY BASIC FOR KIM-1
11 REM 6502 V.1K BY T. PITTMAN.
12 REM
13 REM PROGRAMMED BY:
14 REM C.R. (CHUCK) CARPENTER W5USJ
15 REM 2228 MONTCLAIR PL.
16 REM CARROLLTON TX 75006
17 REM
18 REM THESE PROGRAMS ILLUSTRATE BYTE SAVING
19 REM TECHNIQUES IN LIMITED MEMORY SYSTEMS.
20 REM THE FIRST PROGRAM USED 492 BYTES. THE
21 REM OTHER USED 410 BYTES. AN INCREASE
22 REM (OR SAVING) OF 82 BYTES. IMPLIED
23 REM STATEMENTS AND ABBREVIATIONS ARE
24 REM THE REASON.
25 PR
26 PR
100 PRINT"LOANS : HOW MANY -"
110 INPUT N
115 PRINT
120 LET A=0
130 PRINT"INPUT: PRINCIPAL IN HUNDREDS (P)"
140 PRINT"      RATE IN PERCENT (R)"
150 PRINT"      TIME IN YEARS (T)"
160 PRINT"      PAYMENTS IN MONTHS (X)"
170 INPUT P,R,T,X
190 LET I=P*T*R
200 LET O=100*P+I
210 LET M=O/X
220 LET A=A+I
230 PRINT
240 PRINT"LOAN NUMBER -";A;" "
250 PRINT"INTEREST IS $";I
260 PRINT
270 PRINT"MONEY OWED IS $";O
280 PRINT
290 PRINT"PAYMENTS ARE $";M
300 PRINT
310 LET N=N-1
320 IF N>0 THEN GOTO 170
360 PRINT
370 PRINT"DONE"
380 PRINT
390 END
:
:
:
:
:I=0
:I I=I+2
:2 GOSUB 1
:RUN
:
:226 AT 1
:END
:PRINT"THESE ARE ";I;" BYTES LEFT"
THERE ARE 288 BYTES LEFT
:

```

Fig. 1. First program version using no shortcuts to write the program or save bytes. This program uses 492 bytes, exclusive of the REM statements. REM statements use 470 bytes. The short routine above illustrates how Tiny BASIC finds the number of bytes of free space remaining. The user's manual tells how to do it.

slower, but the increase in program lines is worth the loss of speed (if speed is your concern then Tiny BASIC may not be for you, anyway). Memory saving wasn't really necessary for this short program; but in a 100-line program over 200 bytes could be saved (12 to 15 lines' worth). Such significant savings allow you to write longer programs. The programs are still small, but even a few more lines make them more useful. And that's what we're trying to do. Bytes could be saved in a few more places, such as the spaces in the print input, lines 130 through 160, but in the interest of clarity, I left them alone.

### Decimal Values

Calculations involving decimal numbers can be handled several ways. Anytime a percentage or a calculation resulting in a fraction occurs, a decimal number results. Dollars and cents are decimal numbers, too. Tiny BASIC truncates decimal numbers down to the next lower whole number. If the number is less than one, the result is zero. (For this

reason, accountants would probably not want to use Tiny BASIC.)

Lines 130 through 180 are the input lines for this program. I used principal in hundreds and rate in percent to avoid decimal percentage entry and to prevent dividing percent by 100 (to get back to a decimal percentage). The math comes out right when it's printed out in line 250. I then multiplied the total loan value by 100 in line 200 to make the right amount print in lines 270 and 290.

Principal input in hundreds also helps avoid the number-limitation problem. Keeping the numbers to be operated on small limits precision but keeps the multiplication results in range. Adding a statement in a print line to multiply (or divide, etc.) by some factor will put the answer back in the right magnitude. This is sort of like using engineering notation with a slide rule. The difference is the lack of decimal numbers.

An input-line limitation of 72 characters restricts the amount of data you can input. Two character spaces are used

by the prompting question mark and following space. This reduces actual data input to 70 characters, including the required commas between the data entries. With the loan amount in hundreds, I was able to input values for six loans instead of five. To overcome the limited data-input situation, write programs that will perform calculations, hold the results and return for more

data. I've done this on some data-processing routines with good results.

There's another way to accommodate more data than the line will hold. Simply input as many loan numbers (or WHAT-IFs) as needed in line 100. When the program has used the data entered, it will ask for more until the number of N entries is reached in line 320. Question marks will show up each time

```
:LIST
```

```
100 PR"LOANS : HOW MANY -"
110 INPUT N
115 PR
120 A = 0
130 PR"INPUT: PRINCIPAL IN HUNDREDS (P)"
140 PR"    RATE IN PERCENT (R)"
150 PR"    TIME IN YEARS (T)"
160 PR"    PAYMENTS IN MONTHS (X)"
165 PR
170 INPUT P,R,T,X
190 I = P*T*R
200 O = 100*P + I
210 M = O/X
220 A = A + I
230 PR
240 PR"LOAN NUMBER - ";A;" "
250 PR"INTEREST IS $";I
260 PR
270 PR"MONEY OWED IS $";O
280 PR
290 PR"PAYMENTS ARE $";M
300 PR
310 N = N - 1
320 IF N > 0 GOTO 170
360 PR
370 PR"DONE"
380 PR
390 END
```

Fig. 2. Second program version using implied statements and abbreviations to save bytes. This version uses 410 bytes.

```
LOANS : HOW MANY -
?6
```

```
INPUT: PRINCIPAL IN HUNDREDS (P)
      RATE IN PERCENT (R)
      TIME IN YEARS (T)
      PAYMENTS IN MONTHS (X)
```

```
?40,10,3,36,40,12,4,48,40,18,5,60,50,10,3,36,50,1
2,4,48,50,18,5,60
```

```
LOAN NUMBER - 1
INTEREST IS $1200
```

```
MONEY OWED IS $5200
```

```
PAYMENTS ARE $144
```

```
LOAN NUMBER - 2
INTEREST IS $1920
```

```
MONEY OWED IS $5920
```

```
PAYMENTS ARE $123
```

```
LOAN NUMBER - 3
INTEREST IS $3600
```

```
MONEY OWED IS $7600
```

```
PAYMENTS ARE $126
```

```
LOAN NUMBER - 4
INTEREST IS $1500
```

```
MONEY OWED IS $6500
```

```
PAYMENTS ARE $180
```

```
LOAN NUMBER - 5
INTEREST IS $2400
```

```
MONEY OWED IS $7400
```

```
PAYMENTS ARE $154
```

```
LOAN NUMBER - 6
INTEREST IS $4500
```

```
MONEY OWED IS $9500
```

```
PAYMENTS ARE $158
```

```
DONE
```

Fig. 3. Sample run. Simple interest calculations of two different loan values at three rates.

From Fig. 3 Simple Int			From Fig. 5 Compound Int		
Interest%	Years	Amount	Equiv-Int%	Years	Amount
1. 10	3	\$5200.00	11	3	\$5320.00
2. 12	4	5920.00	15	4	6400.00
3. 18	5	7600.00	26	5	9200.00

Mult	Actual Loan Value	Difference
1. 1.331	\$5324.00	+\$ 4.00
2. 1.574	\$6296.00	- 104.00
3. 2.288	\$9152.00	+ 48.00

Fig. 4. For a loan of \$4000.

line 170 runs out of data and line 320 is still greater than zero.

This program only calculates simple interest loans. Compound-interest calculations require decimal numbers and raising numbers to some power. The multiplier for compounding over  $n$  periods is  $(1 + I)^n$ , where  $I$  is the interest expressed as a decimal and  $n$  is the number of years (or periods).

You can use this multiplier to calculate the approximate equivalent while percentage over the term of the loan. Your calculated answer will result in a much more realistic loan evaluation. I made some of these calculations, and Fig. 4 has some examples.

In the program itself, there are no unusual or unique programming techniques. There are two counting loops—one starting at line 110 and the other at line 120. Whatever

value is input for  $N$  is decremented in line 310 until the data sets, input in line 170, are used up. The counter that starts in line 120 numbers the printed output each time a pass through the program is completed.

I tried to use  $N$  to do both, but could not without using more program lines. Otherwise, this is simply a fundamental program with input between lines 100 and 170, calculations between lines 190 and 220 and output between lines 240 and 290.

#### Summary

It is easy to save bytes of memory if you remember to use implied statements and statement abbreviations. The user's manual for Tiny BASIC shows what is, and is not, allowed. Both the decimal number and number range limitation can be handled by using software math techniques (multipliers, dividers, engineering notation,

LOANS : HOW MANY -  
?3

INPUT: PRINCIPAL IN HUNDREDS (P)  
RATE IN PERCENT (R)  
TIME IN YEARS (T)  
PAYMENTS IN MONTHS (X)

?40,11,3,36,40,15,4,48,40,26,5,60

LOAN NUMBER - 1  
INTEREST IS \$1320

MONEY OWED IS \$5320

PAYMENTS ARE \$147

LOAN NUMBER - 2  
INTEREST IS \$2400

MONEY OWED IS \$6400

PAYMENTS ARE \$133

LOAN NUMBER - 3  
INTEREST IS \$5200

MONEY OWED IS \$9200

PAYMENTS ARE \$153

DONE

Fig. 5. Loan value two, rerun to show the effect of compound interest on the total loan value. Compare the results with the simple interest calculation.

etc.). Line input characters limited to 70 (72 with prompting question mark and space) can also be handled by programming techniques.

Remember, if you input more than a total of 72 characters in a single line, the program will stop. Nothing more will happen

until you reset your system. If you have to reset and want to save the program already in memory, then reenter the interpreter at the soft entry point. The Tiny BASIC user's manual explains how to do this, too. A program does not have to be big to be useful. ■

# Not So Tiny

*Perhaps after running this series we won't be calling it Tiny anymore!*



*KIM-1 and KIM-2 in redwood enclosure, ACT-1 TVT, Telpar Printer, Computerist power supply, Radio Shack recorders.*

```
:LIST
```

```
10 REM ORIGINAL VERSION
11 REM
100 FOR Y=1 TO 10
110 LET C=0
120 FOR X=1 TO 50
130 LET F=INT(2*RND(1))
140 IF F=1 THEN 180
150 PRINT "T";
160 GOTO 200
170 REM C COUNTS NO OF HEADS
180 LET C=C+1
190 PRINT "H";
200 NEXT X
210 PRINT
220 PRINT "HEADS ";C;" OUT OF 50 FLIPS"
230 NEXT Y
240 END
```

*Listing 1.*

of FOR-NEXT statements to LET, IF ... THEN GOTO statements, I have used the program in Listing 1. This is a coin-flipping routine with one counting loop inside another. The outside loop resides between lines 100 and 230; the inside loop is between lines 120 and 230. Lines 10 and 11 are my comment and are not part of the original program. It is not possible to run this program on my system because the Tiny BASIC interpreter would not recognize line 100 and would stop.

Listing 2 is my version rewritten in Tiny BASIC. I have added a couple of features, such as the INPUT N line, which lets you select N sets of 50 flips. Also, I like to see DONE (or something) at the end of a program. This way I know the program didn't quit in the middle (if the algorithm was right, anyway). Otherwise, Tiny BASIC used two more program lines than the larger BASIC version.

In my program, the two main loops comparable to the sample program are started with a LET statement. The outside loop is between lines 110 and 250 and controls the number of passes of 50 flips set in line 100. The inside loop is between lines 130 and 210 and controls the number of flips set in line 210. As I stated there are two additional lines—the counters for the two loops. The loop counter in line 200 increments by one on each pass through the program until it reaches the values in line 210. Incrementing the I loop (in line 240) by one occurs until the value in line 250 is reached. In this case, I is compared to N, the value input in line 100. The value of N lets the user select how many sets of 50 flips are to be run by the program before it ends.

**P**rograms written in Tiny BASIC and other small interpreters can be useful and fun. First, some changes in programming techniques and philosophy are needed, though, because there are fewer statements and commands in small interpreters.

One basic and very useful programming tool is the loop. Several articles have been written about the power and use of loops properly written and executed in a program. Usually in larger BASICs, these loops are written with FOR-NEXT statements. In Tiny BASIC, the equivalent statements are LET, IF ... THEN GOTO.

To illustrate the conversion

coin flipping, counting and printing are handled in lines 140 to 190. Line 140 randomizes the number 2 (1 is added so there are no zeros). If the random number is 1, it becomes a "head" and passes to the head counter in line 180. The head counter increments by one and prints an H, then increments the X loop by one. If X is less

than the limiting value (50), the program returns to the flip routine at line 140 and starts through again.

If F does not equal 1 in line 150, the value becomes a "tail," a T is printed, X is incremented (by jumping to line 200) and compared to the limiting value. This time; if 50 flips have occurred, the program falls through to the print statement in line 230. Heads (C) counted in line 180 are printed out and the program tests the relationships in lines 240 and 250. When I > N, the program prints DONE and ends.

Tiny BASIC, even though small in size, has power enough to produce significant programs. Applications are limited only by your imagination and user space in your computer's memory. In addition to some tricks using implied statements and commands to save memory, I have written programs to plot a graph, do simple graphics, do some limited data processing and simulate assembly processes in a small manufacturing company.

I plan to try several potential capabilities that include use of the USR function to save and load from a cassette tape. I would like to share my ideas with anyone interested, and I believe *Kilobaud* would be happy to publish programs for the development of a Tiny BASIC software library. ■

```

:LIST
10 REM TINY BASIC FOR KIM-1
11 REM 6502 V.IK BY T. PITTMAN.
12 REM
13 REM PROGRAMED BY:
14 REM C. R. (CHUCK) CARPENTER WSUSJ
15 REM 2228 MONTCLAIR PL.
16 REM CARROLLTON, TX. 75006
17 REM
18 REM FLIPS A COIN 'N' TIMES 50 AS SELECTED
19 REM IN LINE 100, THEN PRINTS THE NUMBER OF
20 REM HEADS IN EACH 50 FLIPS.
21 PR
22 PR
100 INPUT N
110 LET I=1
120 LET C=0
130 LET X=1
140 LET F=(RND(2)+1)
150 IF F=1 GOTO 180
160 PRINT "T";
170 GOTO 200
180 LET C=C+1
190 PRINT "H";
200 LET X=X+1
210 IF X<=50 GOTO 140
220 PRINT
230 PRINT "HEADS ";C;" OUT OF 50 FLIPS"
240 LET I=I+1
250 IF I<=N GOTO 120
260 PRINT
270 PRINT "DONE"
280 END

:RUN
? 5
HTHTTTTHTHTHTTTTTHHHHHHTHHHTHTHTHTTTHHTTTHHTTTTHTHTTH
HEADS 26 OUT OF 50 FLIPS
HHTHHHTHHHHHTTHTTHTTHTHTHTTTTTHHTTHTHTHTTTTTHHH
HEADS 28 OUT OF 50 FLIPS
TTHHTTTTHHHHTTTTHTTTHHHHHHTTHTTHTHTHTHHHTHTHTTTTTHHH
HEADS 28 OUT OF 50 FLIPS
THTHHHTTTTHTTTTHTTTTTHHHHTHTHTHHHHHTTTTHTHHHTHTHH
HEADS 25 OUT OF 50 FLIPS
TTHTTHTTTTHTTTTHTTTTHTTTTHTTHTTHTTHTTHTHTHTHTHT
HEADS 18 OUT OF 50 FLIPS

DONE

```

Listing 2.

# Tiny BASIC: Still Going Strong!

As an aid to those needing software to implement on a "Tiny" system, I present three game programs. Extensive personal research (I cornered my wife) demonstrated the appeal of these games to non-computer-oriented (i.e., normal) people. Each will run in a Tiny BASIC-equipped computer with 4K of memory. Although I used the SWTP M-68, programs should be interchangeable with any Tiny BASIC.

Remember, these programs are written in *Tiny* BASIC. Although with minor modifications, as in the RND function, they will run in standard BASIC, they will not be efficient. String handling and FOR-NEXT loops could simplify and speed up these programs, but then they wouldn't be Tiny BASIC.

Enough introduction. On to the programs.

## Battle of Numbers

Battle of Numbers, fre-

Marc I. Leavey, M.D.  
4006 Winlee Road  
Randallstown MD 21133

**A**fter assembling a home computer system, one of the first things hobbyists want to do is demonstrate to their friends and neighbors what their new machines can do. Unfortunately, those things we love to do, like

machine-language subroutines or vectored interrupts, don't come across well to "out-worlders." Furthermore, most of the games or educational programs available require BASIC with string capability. This implies eight to ten kilobytes of read-write memory, usually more than beginning systems have.

Fortunately, a language,

Tiny BASIC, exists that fits comfortably in the 4K generally available in a minimal system. Versions are available for most popular CPU's from Itty Bitty Computers of San Jose CA. Although Tiny BASIC does not have strings, FOR-NEXT loops or several other features of "standard" BASIC, it is still a useful language.

```

10 REM BATNUM [TINY BASIC]
20 REM VER 1.2 - 13 AUG 77
30 REM MARC I. LEAVEY, M.D.
40 REM *HOME UP, ERASE, PRINT HEAD*
50 PR "","BATTLE OF NUMBERS"
60 PR
70 PR "HOW MANY OBJECTS IN"
80 PR "THE PILE";
90 INPUT P
100 IF P <= 0 GOTO 70
110 PR "WHAT IS THE MINIMUM YOU"
120 PR "CAN TAKE";
130 INPUT A
140 IF A > 0 GOTO 180
150 PR "YOU HAVE TO TAKE AT"
160 PR "LEAST 1 EACH TIME!"
170 GOTO 110
180 PR "WHAT IS THE MAXIMUM"
190 PR "YOU CAN TAKE";
200 INPUT B
210 IF B >= A GOTO 250
220 PR "THE MINIMUM CAN'T BE"
230 PR "LARGER THAN THE MAXIMUM!"
240 GOTO 110
250 W=1
260 L=0
270 PR "DO YOU WIN OR LOSE BY TAKING"
280 PR "THE LAST OBJECT (W OR L)";
290 INPUT Z
300 IF Z=1 GOTO 320
310 L=A
320 T=A+B
330 Y=1
340 N=0
350 PR "DO YOU WANT TO GO FIRST";
360 INPUT Z
370 IF Z=1 GOTO 600
380 IF P > B GOTO 410
390 IF P <= A GOTO 540
400 IF L=0 GOTO 540
410 R=P-T*(P/T)

```

```

420 IF R >= A GOTO 450
430 IF R=0 GOTO 450
440 R=A
450 IF R=L GOTO 500
460 C=R-L
470 IF C > 0 GOTO 510
480 C=C+B
490 GOTO 510
500 C=A+RND(B-A+1)
510 PR "I TAKE ";C
520 P=P-C
530 GOTO 600
540 PR ""
550 IF L=0 GOTO 580
560 PR "I TAKE" ;P;"AND LOSE! [LUCKY!]"
570 GO TO 770
580 PR "I TAKE" ;P;"AND WIN!!"
590 GO TO 770
600 PR ""
610 PR "THERE ARE";P;"OBJECTS."
620 PR "HOW MANY DO YOU TAKE";
630 INPUT H
640 IF H < A GOTO 660
650 IF H <= B GOTO 700
660 IF H <> P GOTO 680
670 IF P < A GOTO 720
680 PR "YOU MAY TAKE FROM";A;"TO";B
690 GO TO 620
700 P=P-H
710 IF P > 0 GOTO 380
720 IF L=0 GOTO 750
730 PR ">> YOU LOSE! <<<"
740 GOTO 770
750 PR "*** YOU WIN! ***"
760 GOTO 770
770 PR ""
780 PR "ANOTHER MATCH";
790 INPUT Z
800 IF Z=1 GOTO 10
999 END

```

BATNUM program listing.

quently abbreviated BATNUM, is one of the oldest number games. In it, a pile of objects is established and items are removed until the game ends.

In the computer version, the size of the starting pile, minimum and maximum number per turn and win or lose on the last token are all determined by the player. The computer will go first or give you the option. It is a challenging game, and, with the proper strategy, you can win it.

As with all listings in this article, BATNUM is fairly self-explanatory, but a few points bear mentioning. Tiny BASIC allows PR for PRINT; all other commands are spelled out. The statement PR "" contains control characters used for homing the cursor and clearing the screen or line. Although Tiny has no string inputs, single-letter variables may be input at INPUT statements. Thus the

sequencey

```
100 Y=1
200 N=0
300 PR "ANOTHER GAME";
400 INPUT Z
```

could be answered by Y or N, and the variable Z would equal 1 for yes or 0 for no. Kind of a pseudo-string.

### Bagels

The second listing shows the Bagels program, which also has been around in various forms for some time. The theory of this game is that the computer selects a random number with three *different* digits. It then requests a guess from you. After first checking for other than three digits or double digits, the computer responds three ways (shown in Example 1).

Thus, if the computer's number was 439 and you guessed 497, it would respond: PICO FERMI, showing two correct digits — one in the right place and one in the wrong. PICOs come out

```
I HAVE A NUMBER
GUESS? 111
NO DOUBLE NUMBERS!
GUESS? 234
B A G E L S !
GUESS? 123
B A G E L S !
GUESS? 5678
THREE DIGITS, PLEASE!
GUESS? 567
PICO
GUESS? 890
PICO FERMI
GUESS? 590
FERMI
GUESS? 690
FERMI FERMI
YOU MUST BE NEW AT THIS GAME!
THE FIRST NUMBER IS 6
GUESS? 691
FERMI FERMI
GUESS? 698
```

```
CORRECT! IN 10 GUESSES!
TRY ANOTHER? Y
I HAVE A NUMBER
GUESS? 123
B A G E L S !
GUESS? 456
PICO PICO
GUESS? 789
PICO
GUESS? 457
PICO PICO
GUESS? 458
PICO PICO PICO
GUESS? 845
```

```
CORRECT! IN 6 GUESSES!
TRY ANOTHER? N
```

*Bagels run.*

```
HOW MANY OBJECTS IN
THE PILE? 21
WHAT IS THE MINIMUM YOU
CAN TAKE? 3
WHAT IS THE MAXIMUM
YOU CAN TAKE? 1
THE MINIMUM CAN'T BE
LARGER THAN THE MAXIMUM!
WHAT IS THE MINIMUM YOU
CAN TAKE? 1
WHAT IS THE MAXIMUM
YOU CAN TAKE? 3
DO YOU WIN OR LOSE BY TAKING
THE LAST OBJECT (W OR L)? L
DO YOU WANT TO GO FIRST? N
I TAKE 2
```

```
THERE ARE 19 OBJECTS.
HOW MANY DO YOU TAKE? 3
I TAKE 2
```

```
THERE ARE 14 OBJECTS.
HOW MANY DO YOU TAKE? 2
I TAKE 2
```

```
THERE ARE 10 OBJECTS.
HOW MANY DO YOU TAKE? 2
I TAKE 2
```

```
THERE ARE 6 OBJECTS.
HOW MANY DO YOU TAKE? 2
I TAKE 2
```

```
THERE ARE 2 OBJECTS.
HOW MANY DO YOU TAKE? 1
```

```
I TAKE 1 AND LOSE! [LUCKY!]
```

```
ANOTHER MATCH? N
```

*BATNUM run.*

```
BAGELS = No digit correct
PICO = Correct digit in wrong place
FERMI = Correct digit in correct place
```

*Example 1.*

*Bagels program listing.*

```
10 REM BAGELS < TINY BASIC >
20 REM VER 2.0 - 31 AUG 77
30 REM MARCI LEAVEY, M.D.
50 Y=1
60 N=0
70 PR "";
100 X=100+RND(900)
120 W=X
130 X=W/100
140 Y=(W-X*100)/10
150 Z=(W-X*100-Y*10)
200 IF X=Y GOTO 100
210 IF Y=Z GOTO 100
220 IF X=Z GOTO 100
290 PR "I HAVE A NUMBER"
300 G=0
310 G=G+1
312 IF G=9 PR "YOU MUST BE NEW AT THIS GAME!"
313 IF G=9 PR "THE FIRST NUMBER IS ";X
314 IF G=14 PR "I CAN'T BELIEVE IT!"
315 IF G=14 PR "THE FIRST TWO NUMBERS ARE";X;Y
320 PR "GUESS";
330 INPUT D
340 IF D=W GOTO 900
344 IF G=18 PR "I G I V E U P!"
346 IF G=18 PR "THE NUMBER WAS ";W
348 IF G=18 GOTO 920
350 IF D < 100 GOTO 950
360 IF D > 999 GOTO 950
370 A=D/100
380 B=(D-100*A)/10
```

```

390 C=(D-100*A-10*B)
400 IF A=B GOTO 850
410 IF A=C GOTO 850
420 IF B=C GOTO 850
430 F=0
440 P=0
450 IF A=X THEN F=F+1
460 IF A=Y THEN P=P+1
470 IF A=Z THEN P=P+1
480 IF B=X THEN P=P+1
490 IF B=Y THEN F=F+1
500 IF B=Z THEN P=P+1
510 IF C=X THEN P=P+1
520 IF C=Y THEN P=P+1
530 IF C=Z THEN F=F+1
540 IF P+F=0 PR "B A G E L S !";
550 IF P=0 GOTO 600
560 P=P-1
570 PR "PICO ";
580 GOTO 550
600 IF F=0 GOTO 640
610 F=F-1
620 PR "FERMI ";
630 GOTO 600
640 PR
650 GOTO 310
850 PR "NO DOUBLE NUMBERS!"
860 GOTO 310
900 PR
910 PR "CORRECT! IN";G:"GUESSES!"
920 PR "TRY ANOTHER";
930 INPUT Z
940 IF Z=Y GOTO 10
945 GOTO 999
950 PR "THREE DIGITS, PLEASE!"
960 GOTO 310
999 END

```

before FERMI, so their order is of no help in determining the correct sequence.

This program demonstrates a few useful techniques. The sequence from lines 100 to 220 breaks the three-digit number W down to three integers: X, Y and Z. They are then checked for duplicate digits; if one is found, another number is selected. Similar statements at lines 370 to 390 break the guess D down to integers A, B and C. Comparisons between A, B and C, and X, Y and Z increment the PICO and FERMI flags (P and F, respectively). These flags are used in a pseudo FOR-NEXT loop to print the PICO and FERMI. If neither is set (P+F=0), BAGELS gets printed. A guess counter (G) is also tallied in to offer the player some form of feedback.

### Lunar Lander

Another popular game is the simulated landing of a spacecraft on the moon. Versions have been published in all major books and magazines, including *Kilobaud*. The object is quite simple: Land your lunar excursion module (LEM) without crashing. In this program, constants for fuel, velocity, height and gravity are randomized at each play. This adds a degree of difficulty because the same strategy does not always work.

The loop at lines 92 to 96 counts to 50, giving the player a chance to read the introduction. Subroutine 600 produces a line feed and line erase for each 40 feet or so below 500 feet. This makes the LEM, which is drawn by lines 700 to 720, descend the screen as the game progresses.

```

10 REM LUNAR LANDER [TINY BASIC]
20 REM VER 3.0 - 30 AUG 77
30 REM MARC I. LEAVEY, M.D.
40 PR "", "LUNAR LANDER"
50 PR
55 PR
60 PR "TRY TO LAND THE LEM ON THE"
65 PR
70 PR "SURFACE OF THE MOON BY ENTERING"
75 PR
80 PR "FUEL BURN RATES WHEN REQUESTED."
85 PR
90 PR "GOOD LUCK!"
92 I=50
94 I=I-1
96 IF I > 0 GOTO 94
100 F=100+RND(75)
110 V=RND(50)-100
120 D=400+RND(200)
130 G=1+RND(8)
200 GOSUB 600
210 GOSUB 700
220 IF F > 0 GOTO 240
230 B=0
235 GOTO 250
240 GOSUB 750
250 IF B > F THEN B=F
255 F=F-B
260 C=B-G
270 D=D+V+C/2
280 V=V+C
400 IF D > 0 GOTO 200
410 IF D < -1 GOTO 500
420 GOTO 530
500 GOSUB 660
510 GOTO 800
530 GOSUB 900
540 GOTO 800
600 PR "";
610 S=12-D/40
615 IF S <= 0 GOTO 650
620 PR ""
630 S=S-1
640 IF S > 0 GOTO 620

```

```

650 RETURN
660 PR ""
665 PR "CRASH","CRASH","CRASH"
670 PR "*****","*****","*****"
675 PR
680 PR "IMPACT VELOCITY: ";V
685 PR "LEM BURIED";-D;"FEET"
690 PR
695 GOTO 1010
700 PR "0";"FUEL: ";F
710 PR "{#}";"SPEED: ";V
720 PR " - "; "HEIGHT: ";D;
730 RETURN
750 PR " BURN: ";
760 INPUT B
770 RETURN
800 PR
810 PR "ANOTHER GAME";
820 Y=1
830 INPUT A
840 IF A=Y GOTO 100
850 END
900 PR ""
910 PR
920 PR "LEM ON SURFACE OF THE MOON"
930 IF V < -5 GOTO 1000
935 PR
940 PR "CONGRATULATIONS!"
945 PR
950 PR "", "PERFECT LANDING!"
955 PR
960 PR "TOUCHDOWN VELOCITY: ";V
970 PR "FUEL REMAINING: ";F
980 RETURN
1000 PR "EXCESSIVE SPEED ON IMPACT!"
1005 PR
1010 IF F=0 GOTO 1050
1020 PR F;" UNITS OF FUEL REMAINING"
1030 PR "PRODUCED EXPLOSION COVERING"
1040 PR 100*RND(F+3);"SQ MILES OF LUNAR SURFACE"
1050 PR
1060 PR "LEM DESTROYED!"
1070 PR "***** YOU BLEW IT! *****"
1080 RETURN

```

In the sample run, this routine has been bypassed since it makes little sense on hard copy. It does add some flavor to the CRT version, though.

I hope the reader will be able to introduce his or her acquaintances to the world of personal computers by implementing these simple programs. Comments or questions are welcome; readers interested in Tiny BASIC should write (I have no connection with IBC): Itty Bitty Computers, P.O. BOX 23189, San Jose CA 95153. (A self-addressed stamped envelope should accompany requests for replies.) ■

```

TRY TO LAND THE LEM ON THE
SURFACE OF THE MOON BY ENTERING
FUEL BURN RATES WHEN REQUESTED.
GOOD LUCK!

```

```

0 FUEL: 111
[#] SPEED: -84
/\ HEIGHT: 431 BURN: ? 5

```

```

0 FUEL: 106
[#] SPEED: -80
/\ HEIGHT: 349 BURN: ? 3

```

```

0 FUEL: 103
[#] SPEED: -78
/\ HEIGHT: 270 BURN: ? 0

```

```

0 FUEL: 103
[#] SPEED: -79
/\ HEIGHT: 192 BURN: ? 0

```

```

0 FUEL: 103
[#] SPEED: -80
/\ HEIGHT: 113 BURN: ? 6

```

```

0 FUEL: 97
[#] SPEED: -75
/\ HEIGHT: 35 BURN: ? 12

```

```

CRASH CRASH CRASH
*****

```

```

IMPACT VELOCITY: -64
LEM BURIED 35 FEET

```

```

85 UNITS OF FUEL REMAINING
PRODUCED EXPLOSION COVERING
300 SQ MILES OF LUNAR SURFACE

```

```

LEM DESTROYED!
***** YOU BLEW IT! *****

```

```

ANOTHER GAME? N

```

*Lunar Lander run.*

# Match Pennies: A Game That Learns

Here is a program that demonstrates a computer's ability to show adaptive (artificial) intelligence and pattern recognition. The program is in the form of a simple penny-matching game and is planned as follows.

The computer guesses whether you are going to pick heads or tails. If it guesses correctly, it will subtract a point from your score. If it is wrong, your score is increased by a point.

To perform this task, the computer must decide whether to pick heads or tails. In the program, I have established criteria for making this decision. The computer has to keep a record of the human's previous plays. It will then look up in this record previous plays that match the situation with which it is now presented. Using earlier results, it now has a basis to make a decision on whether to play heads or tails.

Here's an outline of this basic concept:

1. Situation memory (16 cells)

2. Situation comparer
3. Input data (heads or tails)
4. Decision maker
5. Decision output (heads or tails)
6. Win/lose detector
7. Scorekeeper (from human's view)

The implementation of the outline has a different appearance. The program is written in Pittman Tiny BASIC. To set up the situation memory, I selected 16 variables. These act as 16 memory cells, each to contain a 0 = heads or a 1 = tails. The 16 cells are addressed by a memory address register that represents the last four human plays (head, tail, head, tail, etc.). This address (situation) register is contained in four variables. As a new play is generated, the play that occurred four plays ago is shifted out and each play is shifted one position, with the present play being shifted in as the least significant part of the address (situation) register. Thus, the address (situation) register is at all times a representation of

the last four human plays.

The computer uses this address register to compute a cell number (address). This is done by giving each of the four plays contained in the address register a value (power of 2). The oldest play, if it was a tail (= 1), is represented by 8; next, if it was a tail, by 4, and so on until the latest play equals 1. These are then added to compile a number (0-15). This corresponds to a cell number. The program stores the human's latest play (input data – heads = 0; tails = 1) in the cell whose

cell number is computed from the address register. This tells the computer that the human played H, T, H, T, for example, and then played heads again.

The next part of the program shifts the latest play into the address register. It then compares the latest play to the variable V (computer's guess from the end of the last play) to determine if the guess is a match or not. Depending on the results of the comparison, the human's score is incremented or decremented, and the human is shown the results. Then the computer (using the latest shift address register value) looks up the cell number and gets the human's play the last time this situation occurred. This is then used for computer's next guess (variable V).

Fig. 1 is a flowchart of the entire program and shows the four main parts of the program's main loop:

1. Store (present data with last situation).
2. Shift (to get latest situation).
3. Check Win/Lose.
4. Fetch guess (based on latest situation).

At first, the program will tend to make the computer appear dumb. This is because the memory cells and address register are initialized with data that is not derived from data

```

10 LET H = 0
15 LET T = 1
20 PRINT "TYPE HEADS OR TAILS (H OR T)"
25 INPUT X
  
```

Example 1.

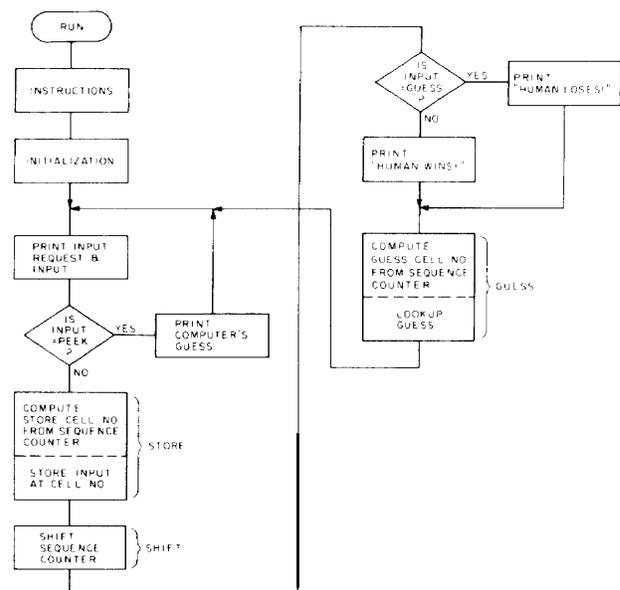


Fig. 1. Flowchart.

the human is presently playing. As soon as the memory contains data acquired from playing, the computer adapts and seems to get progressively more intelligent.

The chart in Table 1 shows how the program gradually

adapts to different patterns of play. The program uses a little-known aspect of Pittman Tiny BASIC: that a variable may be set to a given value and an input requested. The letter of the preset variable may then be typed, and the input will be

equal to the preset value, as in Example 1. If a player types H, the value of X will be 0; if he types T, the value of X will be 1. So, try your luck playing the computer at matching pennies. Remember, it may sucker you at first. You may think that the

computer cheats, so I have included a PEEK command in the program. If you type 2 instead of H or T, the computer will show you its next guess. It is not fair to "peek" every time as you may cause the program to have a nervous breakdown. ■

Game No.	Computer's Play	Human's Play	Win/Lose	Write Cell No.	Read Cell No.	Game Total	Read from Comment Game No.
0	T		W			0	* Reset
1	T	H	W	H-0	H-1	1	8
2	H	H	L	H-1	H-3	0	*
3	H	H	L	H-3	H-7	-1	*
4	H	H	L	H-7	H-15	-2	*
5	H	H	L	H-15	H-15	-3	5
6	H	T	W	T-15	H-14	-2	*
7	H	T	W	T-14	H-12	-1	*
8	H	H	L	H-12	H-9	-2	*
9	H	H	L	H-9	H-3	-3	3
10	H	T	W	T-3	H-6	-2	*
11	H	T	W	T-6	H-12	-1	8
12	H	H	L	H-12	H-9	-2	9
13	H	H	L	H-9	T-3	-3	10
14	T	T	L	T-3	T-6	-4	11
15	T	T	L	T-6	H-12	-5	12
16	H	H	L	H-12	H-9	-6	13
17	H	T	W	T-9	H-2	-5	*
18	H	H	L	H-2	H-5	-6	*
19	H	T	W	T-5	H-10	-5	*
20	H	H	L	H-10	T-5	-6	19
21	T	T	L	T-5	H-10	-7	20
22	H	T	W	T-10	H-4	-6	*
23	H	T	W	T-4	H-8	-5	*
24	H	T	W	T-8	H-0	-4	1
25	H	T	W	T-0	T-0	-3	25
26	T	T	L	T-0	T-0	-4	26
27	T	H	W	H-0	H-1	-3	2
28	H	H	L	H-1	T-3	-4	14
29	T	H	W	H-3	H-7	-3	4
30	H	H	L	H-7	T-15	-4	6
31	T	H	W	H-15	H-15	-3	31
32	H	H	L	H-15	H-15	-4	32
33	H	H	L	H-15	H-15	-5	33
34	H	T	W	T-15	T-14	-4	7
35	T	H	W	H-14	H-13	-3	*
36	H	H	L	H-13	H-11	-4	*
37	H	H	L	H-11	H-7	-5	30
38	H	T	W	T-7	H-14	-4	35
39	H	H	L	H-14	H-13	-5	36
40	H	H	L	H-13	H-11	-6	37
41	H	H	L	H-11	T-7	-7	38
42	T	T	L	T-7	H-14	-8	39
43	H	T	W	T-14	H-12	-7	16
44	H	T	W	T-12	T-8	-6	24
45	T	H	W	H-8	H-1	-5	28
46	H	T	W	T-1	H-2	-4	18
47	H	T	W	T-2	T-4	-3	23
48	T	T	L	T-4	H-8	-4	45
49	H	H	L	H-8	T-1	-5	46
50	T	T	L	T-1	T-2	-6	47
51	T	T	L	T-2	T-4	-7	48
52	T	T	L	T-4	H-8	-8	49
53	H	H	L	H-8	T-1	-9	50

\*Reset State (initialization)

Table 1. Penny-match game.

```

50 PR"MATCH PENNIES WITH THE COMPUTER!"
60 PR"IF THE COMPUTER GUESSES THE SAME AS YOU PICK "
70 PR"THEN THE COMPUTER WINS AND THE HUMAN LOSES!"
86 PR"TYPE YOUR FAVORITE NUMBER(0-100)"
87 INPUT X
100 GOSUB 600
105 PR"HEADS OR TAILS(H OR T)"
110 INPUT X
120 IF X=2 GOSUB 210
130 IF X>1 GOTO 105
140 GOSUB 300
150 GOSUB 400
160 GOSUB 215
170 IF X=V PR"HUMAN LOSES!"
175 IF X=V W=W-1
180 IF X<>V W=W+1
185 IF X<>V PR"HUMAN WINS!"
190 PR"YOUR SCORE IS ";W
195 GOSUB 500
200 GOTO 105
210 PR"YOU PEEKED!! -- NOT FAIR!"
215 PR"THE COMPUTER GUESSED ";
220 IF V=0 PR"HEADS"
225 IF V=1 PR"TAILS"
230 RETURN
300 Y=(8*A)+(4*B)+(2*C)+D
305 IF Y=0 F=X
310 IF Y=1 G=X
315 IF Y=2 E=X
320 IF Y=3 I=X
325 IF Y=4 J=X
330 IF Y=5 K=X
335 IF Y=6 L=X
340 IF Y=7 M=X
345 IF Y=8 N=X
350 IF Y=9 O=X
355 IF Y=10 P=X
360 IF Y=11 Q=X
365 IF Y=12 R=X
370 IF Y=13 S=X
375 IF Y=14 Z=X
380 IF Y=15 U=X
390 RETURN
400 D=C
405 C=B
410 B=A
415 A=X
420 RETURN
500 Y=(8*A)+(4*B)+(2*C)+D
505 IF Y=0 V=F
510 IF Y=1 V=G
515 IF Y=2 V=E
520 IF Y=3 V=I
525 IF Y=4 V=J
530 IF Y=5 V=K
535 IF Y=6 V=L
540 IF Y=7 V=M
545 IF Y=8 V=N
550 IF Y=9 V=O
555 IF Y=10 V=P
560 IF Y=11 V=Q
565 IF Y=12 V=R
570 IF Y=13 V=S
575 IF Y=14 V=Z
580 IF Y=15 V=U
590 RETURN
600 A=0
605 B=0
610 C=0
615 D=0
617 E=0
620 F=0
625 G=0
630 H=0
635 I=0
640 J=0
645 K=0
650 L=0
655 M=0
660 N=0
665 O=0
670 P=0
675 Q=0
680 R=0
685 S=0
687 T=1
690 U=0
692 V=0
695 W=0
696 Z=0
697 RETURN

```

*Program listing.*

Gregory L. Oliver  
P.O. Box 184  
Euless Tx 76039

# Why Not Trig Functions For Your 4K BASIC?

**A** while back, a neighbor's kid was looking through my copy of *101 Basic Computer Games* and asked if he could play *Gunner*. "No," I replied, "my computer can't do this line

with SIN(X) in it." So he settled for *Lunar Lander*. While he was occupied, I wondered if it was possible to simulate this and other math functions, included in 8K BASIC but missing in my

4K version. They weren't called often, but used up lots of programming space whether needed or not. So, why not just have subroutines to add only when necessary?

I recalled from calculus classes that any function can be approximated by a series equation, a method using successive iterations—ideal for a computer. After a lot of research and some trial and error, I had subroutines to calculate SIN(X), COS(X), TAN(X), EXP(X) and LOG(X). Since they're all based on the same principle, let's use SIN(X) to demonstrate.

In 4K BASIC, you can approximate the sine of X by following the function in Example 1—provided that X is in radians, and  $X^n/n!$  is less than some predetermined value, such as 1E-7.

I chose this value to compare with the 8K version. Actually, you could speed things up by

stopping at 1E-4. This is more than enough accuracy for most games. For those of you unfamiliar with the term  $n!$  (called factorial), it is defined as the multiplication of all integers (whole numbers) from one to  $n$ .  $3!$  equals 6,  $5!$  equals 120 and  $7!$  equals 5040.

You can see that  $X^n/n!$  very quickly becomes smaller and smaller. This is called converging, because the more terms you add, the closer you get to the actual answer.

Here's the procedure for finding SIN(X):

1. Convert X in degrees to R in radians.
2. Set X equal to R.
3. Set S equal to R.
4. Set counter N equal to 1.
5. Add 2 to N.
6. Convert term R to  $(-R)^*(S * S)/[-N * (N - 1)]$ .
7. Add R to X.
8. If the absolute value of R is

```

100 REM artillery game by G.L. Oliver
110 REM demonstrates 4K SIN(X) subroutine
200 Let T = 50000 - INT (RND (0) *45000)
205 REM T is distance to target
210 Let A = 0
215 REM A is shot count
220 Input X
230 If X<90 then go to 260
240 Print "Bad Angle"
250 Go to 220
260 If X<1 then go to 240
270 Let X = X * 2
280 Gosub 1000
290 Let A = A + 1
300 Let H = T - INT (50000*X)
310 If H<100 then go to 350
320 Print "Over By"; H; "Yards"
330 Go to 370
350 If H<-100 then go to 400
360 Print "Under By"; H; "Yards"
370 If A<5 then go to 220
380 Print "You Got Hit!"
390 Go to 500
400 Print "Got Him In"; A; "Shots"
410 Print H; "Yards"
500 Print "Try Again? (1 = Yes; 0 = No)";
510 Input A
520 If A = 1 then go to 200
530 Stop

```

Program B.

$$\text{SIN}(X) = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \dots + \frac{X^n}{n!}$$

Example 1.

less than  $1 \times 10^{-7}$ , you are done and should return with X equal to SIN(X).

9. Otherwise, go back to step 5.

Fig. 1 is the flowchart for this procedure, and Program A shows the completed subroutine. As to application, I freely changed and simplified the Gunner program to demonstrate my subroutine (see Program 2).

Now that we have SIN(X), how about COS(X)? All you need to do is add 90 degrees to the angle, and then use the same subroutine you use for SIN(X). Believe me. So, that gives us SIN(X) and COS(X).

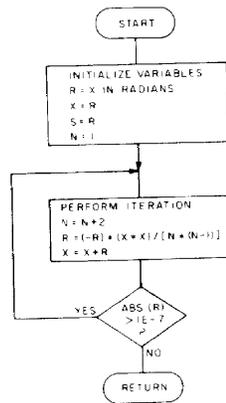


Fig. 1. Flowchart.

TAN(X) is just SIN(X) divided by COS(X). It may take a bit longer to calculate since you have to

```

1000 Let R = X *.01754293
1010 Let X = R
1020 Let S = R
1030 Let N = 1
1040 Let N = N + 2
1050 Let R = -R * S * S / [N * (N - 1)]
1060 Let S = X + R
1070 If ABS (R) < 1E-7 then return
1080 Go to 1040
  
```

Program A.

call the same subroutine twice and juggle a few numbers; but look at the space you save! That was the reason for using 4K to begin with.

You save a lot of space—as I stated earlier—but what are

you giving up? Time, of course. It takes about a second for angles less than 90 degrees, and maybe two seconds when you are up to 360 degrees. So what! You now have 4K extra of programmable memory. ■

# INDEX

- Appendix D, Excerpts from Kilobaud, 75
- Appendix A, Memory Map, 40
- Appendix B, Error MSG Summary, 41
- Appendix C, Monitor Listing, 43
  
- Block Memory Transfer, 12
  
- Display Memory, 9
- Display Program Instructions, 11
- Display Registers, 7
- Display/Alter Memory, 9
- Display/Alter Memory Contents, 9
- Display/Alter Register Contents, 7
- Display/Alter Registers, 7
  
- Editing Commands, 27
- ET-3400 Cassette Usage, 19
- Executing a Program, 13
- Executing a Program Segment, 15
  
- FANTOM II Monitor, 4
- Functions, 34
  
- HEATH/PITTMAN Tiny BASIC, 26
  
- Introduction — FANTOM II, 3
  
- Mathematical Expressions, 32
- Modes of Operation, 29
  
- Numerical Constants, 32
  
- Operators, 32
  
- Power Up and Master Reset, 6
- Program Execution Control, 13
- Program Storage and Retrieval, 18
  
- Sample Program, 22
- Symbols, 5
  
- The RND Function, 34
- The USR Function, 34
- Tiny BASIC Instructions, 30
- Tiny BASIC Re-Initialization (Warm Start), 33
  
- Using an ASR 33, 21
- Using the MONITOR, 6
- Using Tiny BASIC, 28
  
- Variables, 32

## CUSTOMER SERVICE

### REPLACEMENT PARTS

Please provide complete information when you request replacements from either the factory or Heath Electronic Centers. Be certain to include the **HEATH** part number exactly as it appears in the parts list.

### ORDERING FROM THE FACTORY

Print all of the information requested on the parts order form furnished with this product and mail it to Heath. For telephone orders (parts only) dial 616 982-3571. If you are unable to locate an order form, write us a letter or card including:

- Heath part number.
- Model number.
- Date of purchase.
- Location purchased or invoice number.
- Nature of the defect.
- Your payment or authorization for COD shipment of parts not covered by warranty.

Mail letters to: Heath Company  
Benton Harbor  
MI 49022  
Attn: Parts Replacement

**Retain original parts until you receive replacements. Parts that should be returned to the factory will be listed on your packing slip.**

### OBTAINING REPLACEMENTS FROM HEATH ELECTRONIC CENTERS

For your convenience, "over the counter" replacement parts are available from the Heath Electronic Centers listed in your catalog. Be sure to bring in the original part and purchase invoice when you request a warranty replacement from a Heath Electronic Center.

### TECHNICAL CONSULTATION

Need help with your kit? — Self-Service? — Construction? — Operation? — Call or write for assistance. you'll find our Technical Consultants eager to help with just about any technical problem except "customizing" for unique applications.

The effectiveness of our consultation service depends on the information you furnish. Be sure to tell us:

- The Model number and Series number from the blue and white label.
- The date of purchase.
- An exact description of the difficulty.
- Everything you have done in attempting to correct the problem.

Also include switch positions, connections to other units, operating procedures, voltage readings, and any other information you think might be helpful.

**Please do not send parts for testing**, unless this is specifically requested by our Consultants.

Hints: Telephone traffic is lightest at midweek — please be sure your Manual and notes are on hand when you call.

Heathkit Electronic Center facilities are also available for telephone or "walk-in" personal assistance.

### REPAIR SERVICE

Service facilities are available, if they are needed, to repair your completed kit. (Kits that have been modified, soldered with paste flux or acid core solder, cannot be accepted for repair.)

**If it is convenient, personally deliver your kit to a Heathkit Electronic Center. For warranty parts replacement, supply a copy of the invoice or sales slip.**

If you prefer to ship your kit to the factory, attach a letter containing the following information directly to the unit:

- Your name and address.
- Date of purchase and invoice number.
- Copies of all correspondence relevant to the service of the kit.
- A brief description of the difficulty.
- Authorization to return your kit COD for the service and shipping charges. (This will reduce the possibility of delay.)

Check the equipment to see that all screws and parts are secured. (Do not include any wooden cabinets or color television picture tubes, as these are easily damaged in shipment. Do not include the kit Manual.) Place the equipment in a strong carton with at least **THREE INCHES** of *resilient* packing material (shredded paper, excelsior, etc.) on all sides. Use additional packing material where there are protrusions (control sticks, large knobs, etc.). If the unit weighs over 15 lbs., place this carton in another one with 3/4" of packing material between the two.

Seal the carton with reinforced gummed tape, tie it with a strong cord, and mark it "Fragile" on at least two sides. Remember, the carrier will not accept liability for shipping damage if the unit is insufficiently packed. Ship by prepaid express, United Parcel Service, or insured Parcel Post to:

Heath Company  
Service Department  
Benton Harbor, Michigan 49022



**HEATH COMPANY • BENTON HARBOR, MICHIGAN**  
**THE WORLD'S FINEST ELECTRONIC EQUIPMENT IN KIT FORM**