

# USING & PROGRAMMING THE TI-99/4A

## INCLUDING READY-TO-RUN PROGRAMS



BY FREDERICK HOLTZ



**USING & PROGRAMMING THE  
TI-99/4A  
INCLUDING READY-TO-RUN PROGRAMS  
BY FREDERICK HOLTZ**

**TAB** **TAB BOOKS Inc.**  
BLUE RIDGE SUMMIT, PA. 17214

In February, 1983, Texas Instruments Inc. announced the possibility of an electrical shock hazard with the TI-99/4A computer. Please contact Texas Instruments Inc. directly or your dealer for more information.

FIRST EDITION

SIXTH PRINTING

Copyright © 1983 by TAB BOOKS Inc.

Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Library of Congress Cataloging in Publication Data

Holtz, Frederick.

Using and programming the TI-99/4A, including  
ready-to-run programs.

Includes index.

1. TI 99/4A (Computer)—Programming. 2. Basic  
(Computer program language) I. Title.  
QA76.8.T133H64 1983 001.64'2 83-5940  
ISBN 0-8306-1620-9  
ISBN 0-8306-0620-3 (pbk.)

Cover photograph by Zeigler Photography Studio of Waynesboro, PA.

# Contents

|  |            |
|--|------------|
| <b>List of Programs</b>  | <b>v</b>   |
| <b>Acknowledgment</b>  | <b>vi</b>  |
| <b>Introduction</b>  | <b>vii</b> |
| <b>1 Micros and Texas Instruments</b>  | <b>1</b>   |
| Primary Functions—A Microelectronic World—Electromechanical Computing Systems—Electronic Digital and Stored Program Computers  |            |
| <b>2 Home Computer System</b>  | <b>7</b>   |
| Home Versus Personal Computers—The Console—The Keyboard—Accessories  |            |
| <b>3 TI-99/4A BASIC</b>  | <b>37</b>  |
| <b>4 BASIC Programming</b>   | <b>49</b>  |
| Your First Program—Clearing the Screen—Continuous Loops—Erasing the Program—For-Next Loops—More Uses of the Print Statement—If-Then Statements—In and Out of a Loop—Input Statements—Let Statements—Variables—More on Strings—Rules on the Use of Variables—The VAL Function—The LEN Function—If-Then-Else and GOSUB—More on Functions—A Dice Game Program |            |
| <b>5 TI-99/4A Graphics</b>   | <b>75</b>  |
| Screen Coordinates and ASCII—HCHAR—VCHAR—CHAR—Color—Screen—Animation—Sound—Key—A True Game Program   |            |



|           |  |            |
|-----------|--|------------|
| <b>6</b>  | <b>Error Messages</b>  | <b>97</b>  |
|           | Types of Error Messages—Entering Errors—Symbol Table Errors—Program Run Errors   |            |
| <b>7</b>  | <b>The Microprocessor</b>  | <b>103</b> |
|           | Architecture—Instructions—Routines   |            |
| <b>8</b>  | <b>Programs</b>  | <b>119</b> |
|           | Numbers Guess Game—Loan Calculation—Fortune Teller—Teacher's Pet—Comical Intelligence Test—Julian Date—Random Partner Match—Alphabetizing Program—Math Practice—Stepping Sounds—Steps of Thirds—Keyboard |            |
| <b>9</b>  | <b>Other Programming Languages</b>   | <b>151</b> |
|           | TI Extended BASIC—Assembly Language—TI Logo  |            |
| <b>10</b> | <b>Software</b>  | <b>157</b> |
|           | Programming Aids—Engineering and Math Libraries—Business—Home/Personal—Education—Games   |            |
| <b>11</b> | <b>Converting to TI BASIC</b>  | <b>191</b> |
|           | <b>Glossary</b>  | <b>198</b> |
|           | <b>Appendix A Reserved Words in TI BASIC</b>   | <b>209</b> |
|           | <b>Appendix B ASCII Character Codes</b>  | <b>210</b> |
|           | <b>Appendix C Color Codes and Set Numbers</b>  | <b>212</b> |
|           | <b>Appendix D Musical Note Frequencies</b>   | <b>213</b> |
|           | <b>Index</b>   | <b>214</b> |

# List of Programs

|                                  |            |
|----------------------------------|------------|
| <b>Numbers Guess Game</b>        | <b>120</b> |
| <b>Loan Calculate</b>            | <b>122</b> |
| <b>Fortune Teller</b>            | <b>123</b> |
| <b>Teacher's Pet</b>             | <b>127</b> |
| <b>Comical Intelligence Test</b> | <b>131</b> |
| <b>Julian Date</b>               | <b>136</b> |
| <b>Random Partner Match</b>      | <b>139</b> |
| <b>Alphabetizing Program</b>     | <b>141</b> |
| <b>Math Practice</b>             | <b>144</b> |
| <b>Stepping Sounds</b>           | <b>147</b> |
| <b>Steps of Thirds</b>           | <b>148</b> |
| <b>Keyboard</b>                  | <b>149</b> |

# Acknowledgment

I would like to acknowledge and thank Texas Instruments Incorporated for supplying a wealth of technical information and photographs for use in this book. Without their assistance, this book would have been impossible to write.



# Introduction

The TI-99/4A computer from Texas Instruments is one of the finest microcomputer buys on today's market. This is partially due to its low cost, but more so to its excellent characteristics, characteristics that are more often associated with machines costing many times more than the TI-99/4A.

The TI-99/4A is a true family or home computer. It is not designed solely for adults, nor solely for children. It's designed for both. If you have never used a computer before, you will find the typewriter keyboard to be quite easy to adjust to and the entire complement of system and software to be user-friendly.

If you have used high-level machines before, you will still be quite impressed with the values that have been packed into this machine. Many high-level functions are included, making it easy to write and debug your

own programs. Editing of program lines is a snap, and a full complement of error messages is held within ROM to alert you to an error before and during a program run.

The nice thing about the TI-99/4A lies in the fact that it's ready to go in basic form. You don't need a long, expensive list of options to really put the machine to use. All that is required is a color or black-and-white television receiver to which the modulator can be connected. The modulator is a part of the basic machine package.

One outstanding feature of the TI-99/4A is the documentation that Texas Instruments supplies. Unlike some user's manuals, the Texas Instruments manual takes you on a trip through "programland" using language and instructions that are easy to follow. Most computer-world words are defined before they

are used. An easy-to-understand glossary is also included. Each statement, command, and function in TI BASIC is explained, and its use is then demonstrated in a typical program. This is especially helpful to the beginning programmer and is appreciated by the "old hands" as well who may have experience on other machines and in other dialects of BASIC.

Texas Instruments, of course, backs up its product with a surprisingly large selection of software from the TI Software Library. This is quite important, as nothing is quite so irksome as an excellent computer for which there is no available software.

While the TI-99/4A is ready to go in basic form, there are many expansion options that you may want to take advantage of. You can add a Peripheral Expansion System to allow for the use of plug-in cards. You may also wish to add a disk drive, and RS-232 interface, a telephone modem, and even different languages. Command Modules, which plug into a console slot; are available. These ROM modules can extend TI BASIC (Extended BASIC) and add new programming and program capabilities to the basic console. The list seems to be almost endless.

It is quite refreshing for a user of fairly high-level business computers, such as myself, to find so much wrapped up in one inexpensive package. The TI-99/4A at present may be the best all-around home computer for training, home finance, and family-oriented needs. It also has business applications, and many business software packages are available. This computer will probably be more comfortable in

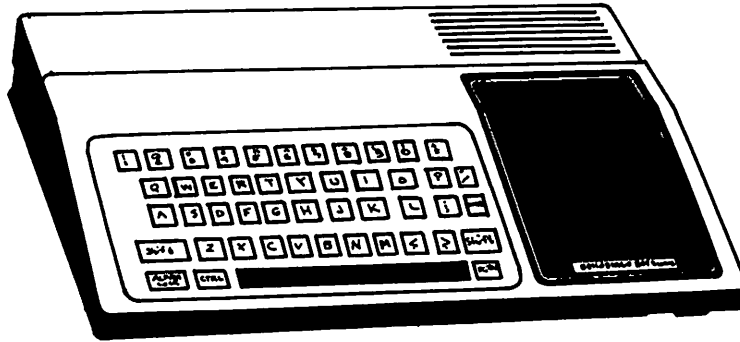
a home environment, but it can hold its own in other pursuits as well.

This book is designed to tell you about the TI-99/4A from the user's standpoint. If you've never touched a computer before, you'll find the chapters concerning writing your own programs on this computer to be a thorough education in the BASIC language itself. If you have some experience in computer programming, you will find the explanation of this machine and the uses of its language helpful in allowing you to quickly learn to operate this machine. A few of the "elite" may appreciate the chapter on the 16-bit microprocessor, and everyone can benefit from the chapter on converting other programs to TI BASIC.

The programs themselves have not been neglected either. You will find many programs in TI BASIC that describe what it's like to program the TI-99/4A. You will also find a separate chapter devoted entirely to presenting and discussing on a line-by-line basis programs that are ready to run on this machine.

Be assured that this book is written for the average TI-99/4A owner. All programs and all discussions involve the basic machine itself, which includes the console, 16K of resident RAM, the video modulator, and nothing else. The discussions in this book concern the basic machine and not the machine with many options attached, although these options are discussed for those who want to add them in the future. If you have purchased the least expensive TI-99/4A package, all discussions in this book will relate directly to you and your machine.

# Chapter 1



## Micros and Texas Instruments

An understanding of microcomputers and especially of the microprocessors around which they are built is especially appropriate for this book, because Texas Instruments Inc., the microprocessor, and the microcomputer all go hand in hand. The first single-chip microprocessor was invented by Gary W. Boone of Texas Instruments in 1970. Today, the microcomputer industry is booming, but up to the middle of 1975, it was really no industry at all. This chapter will clear up some misconceptions about microprocessors and microcomputers and tell you a bit about the role of Texas Instruments in the exciting development of one of our leading industries.

If you've ever looked inside a calculator, an electronic toy, a microwave oven, or any of the other electronic devices crowding today's stores, you're likely to see nothing but a few,

small rectangular shapes mounted on a board. These shapes are microcomputers and microprocessors, and they are the "brains" that run electronic devices.

Microcomputers and microprocessors are tiny silicon chips that can perform a wide range of control functions within a device. They interpret information that's coming into the device and then decide what the device should do next. They accomplish this far more cheaply and efficiently than the roomfuls of wiring, tubes, switches, and relays that were once required to do an equivalent job.

A microprocessor is generally defined as a computer CPU (central processing unit) that has been reduced to microscopic size. It makes decisions for the entire system based on its instructions (software) and incoming information (data). Additionally, it makes sure that



everything is done in the proper order.

Although the microprocessor is sometimes called a computer on a chip, it is actually only part of a computer, the part that performs the arithmetic and keeps the whole system working in harmony. It may be manufactured as a single chip or as several integrated circuits interconnected on a printed circuit board.

The microcomputer is a computer reduced to microscopic size. It contains all the elements of a computer, including the CPU, memory, input and output (I/O) circuitry, and a clock to make things happen in proper sequence.

When all this circuitry is manufactured on one silicon chip, the device is a single-chip microcomputer. When manufactured on one printed circuit board using a number of memory, control, I/O, and associated circuits, it is better described as a microcomputer module. Add to this a power supply, a software program, and I/O devices that allow an operator to communicate with the microcomputer (such as a keyboard to type in information and a monitor to display results), and the result is a microcomputer system.

The single-chip microcomputer, invented by Gary W. Boone and Michael Cochran of Texas Instruments in 1971, was designed to run a variable-function, fixed-program calculator. When introduced to the market, it was soon put to work in consumer and commercial products, including appliances, office equipment, automobiles, telephone equipment, and electronic games and toys.

This versatility is the result of a key distinctive feature of the single-chip microcomputer. Its program (the set of instructions that tell it what to do) is actually built into the chip.

This feature makes it possible for a single chip to control devices ranging from electronic games to microwave ovens.

## **PRIMARY FUNCTIONS**

The brain of a microprocessor is called the arithmetic and logic unit, or ALU. It is the adding machine of the computer, where numbers are added together or where logic decisions are made. The ALU cannot handle information all at once. The microprocessor must be able to fetch instructions and data as they're needed. It must also time the order in which each instruction or piece of data is acted upon. This job is handled by the timing and control circuits.

The timing circuits allow information to flow through the microprocessor in precise synchronization. They pass information along when triggered to do so by signals from the clock. The clock is an oscillator that provides timing signals so that all the information moving around in the device doesn't run into something else. Although the clock is sometimes incorporated in the microprocessor, it may also be included in another part of the computer system. The clock can be an oscillator circuit vibrating millions of times per second, or simply a counter that counts the cycles in alternating current.

The control circuits read the clock, fetching instructions and data as they're needed by the microprocessor. This incoming information is temporarily stored in areas called registers and then fed by the control circuits into the ALU in the proper order.

To carry the information in and out, the microprocessor contains buses. A bus is nothing more than electrical channels that

carry information from one part of a computer to another. Buses can be 4, 8, or 16 channels wide.

There are other segments of the computer system outside of the microprocessor. These include the memory and the input and output (I/O) functions.

The read-only memory, or ROM, is the place where the microcomputer's program is stored. A program is a set of instructions that tells the computer what it must do in step-by-step, exact detail. Since the program is permanently stored in ROM, think of the ROM as you would any instruction book. You can read it, but you can't change what is on the printed page. Memory is often organized into "pages," with a series of operations written in memory and stored on a particular page. A chip programmed to run an oven might have a buzzer or alarm program to signal when the cooking time is up. This program would be stored on a page of ROM and referred to when a buzzer signal is required.

The computer also needs a place to store information that is only needed temporarily. This form of temporary information is held in the random-access memory, or RAM.

While the ROM holds its information permanently, the RAM operates only when the power is on. Think of RAM as an electronic chalkboard on which a grid is drawn. Each of the columns and rows has an address in which a bit of information can be held, i.e., column 5, row 6. Once the information is used, it's erased. The numbers that you key into a calculator are stored in the RAM and then erased when the calculations are complete.

Both the ROM and RAM are composed of a three-dimensional grid of tiny memory cells.

In order for the CPU to locate a particular instruction or bit of data, it must send out an address code to specify the address of the required information. The memory locates the information stored at the specified address and then relays it via a bus to the CPU.

The input and output, or I/O, circuits enable the chip to communicate with the outside world. The input circuitry relays external signals (such as pressing the on key on an electronic toy) to the CPU. The output circuits relay the results of calculations from the CPU to the outside world.

All of these system components—the central processing unit, clock, read-only memory, random-access memory, and I/O circuitry—come together in one or a few silicon chips that are smaller and thinner than a baby's fingernail.

## **A MICROELECTRONIC WORLD**

Microcomputers and microprocessors are extremely efficient and cost-effective. By way of comparison, the vacuum-tube computer of the fifties cost about \$200,000. Its size was measured in cubic feet; its reliability, in hours between failures; and its power in thousands of watts. Today's single-chip microcomputer costs less than \$10.00. Its size is measured in thousands of an inch; its reliability, in hundreds of years; and its power, in milliwatts.

The factors of cost, size, reliability, and flexibility all play a part in the current explosive demand for single-chip microcomputers. The Texas Instruments TMS1000 family of single chip microcomputers, for instance, has been incorporated into over 500 different products, including appliances, toys, and au-

tomobiles, with an installed base totalling over 60 million units.

TI's single-chip microcomputer lines also include the TMS7000 family of 8-bit microcomputers and the TMS9900 family of 16-bit devices. TI pioneered the single chip, 16-bit microprocessor in 1976 when it introduced the TMS9900, on which the TI-99/4 is based. With the introduction of the TMS7000 family, Texas Instruments became the only manufacturer to offer single-chip microcomputers in 4-bit, 8-bit, and 16-bit versions. TI also offers the TM990 series of board-level microcomputer modules and the powerful DS990 series of minicomputer systems. The TM990 and DS990 series are based on the TMS9900 family of microcomputers. TI offers a full range of microcomputer options to the designer, from single-chip microcomputers, to microcomputer boards, to complete computer systems.

Each year more products are becoming computerized, and the benefits are not only immediate convenience and efficiency, but long-term economy as well. Through programming, the same chip(s) can satisfy a number of changing requirements without changing hardware.

A background of the microcomputer has been overviewed, but where did computers begin in the first place? Some say the first computers may be attributed to the Egyptians some thousands of years ago. These people developed some highly sophisticated mechanical devices that could do many things including tell time, add, and subtract. Most people concede that the Chinese, some six thousand years ago, developed the first mechanical predecessor to present-day digital computers, the abacus. It is totally mechanical and depends on

the alignment of beads on strings to indicate highly complex values. Figure 1-1 shows a history of evolutionary development which began the human race on its path to the digital computer. This chart takes us almost to the beginning of the twentieth century. It was during the 1900s that true computing devices were developed.

## **ELECTROMECHANICAL COMPUTING SYSTEMS**

Electromechanical computing devices became a reality about 1914 with the introduction of a punched card machine called a vertical sorter. This device was replaced thirteen years later with a machine much like those in use today with horizontal pockets. This device was called the printing card sorter.

A department store in Pennsylvania had the first point-of-sales recorder installed in 1929, with the recorder connected by wire to a central keypunch facility. The progress during the 1930s was great, and a number of relay-type computers were developed. Dr. Howard Aiken conceived the first fully automatic data processing system in 1937. This system was known as the Automatic Sequence Controlled Calculator or Harvard Mark I. Dr. Aiken's system was influenced by the ideas of Babbage's "Analytical Engine." The Mark I was capable of performing a sequence of arithmetic operations on numbers up to 23 decimal digits in length.

## **ELECTRONIC DIGITAL AND STORED PROGRAM COMPUTERS**

The first electronic computer was the Electronic Numerical Integrator and Computer (ENIAC). Work on ENIAC started in



| DATE             | WHO                                  | WHAT  |
|------------------|--------------------------------------|---|
| 4000-3000 B.C.   | Chinese                              | <b>Abacus</b> —Mechanical predecessor of present-day digital computers  |
| 1300 A.D.        | Hindus and Arabians                  | Final evolution of modern numbers and manipulation systems (adding, subtracting, multiplying, and dividing)   |
| 1617             | Napier                               | <b>Napier's Bones</b> —Mechanical device using logarithms   |
| 1630             | Oughtred                             | <b>Slide Rule</b> —Mechanical device using logarithms and antilogarithms permitting rapid calculations (the first analog computer)  |
| 1642             | Pascal                               | <b>Mechanical Adding Machine</b> —Ratchet-driven wheel device (capable of performing both addition and subtraction)   |
| 1671             | Leibniz                              | <b>Calculating Machine</b> —Stepped wheel device (capable of addition, subtraction, multiplication, and division)   |
| 1804             | Jacquard                             | <b>Punched Cards</b> —System for control of weaver's loom; predecessor of modern punched card accounting system   |
| 1820             | Thomas of Alsac                      | <b>Arithmometer</b> —First practical stepped wheel commercial calculator  |
| 1822             | Charles Babbage                      | <b>"Difference Engine"</b> —Concept and model of device which was intended to build up tables of mathematical functions utilizing their successive difference   |
| 1830-early 1900s | C. Babbage and his son H. P. Babbage | <b>"Analytical Engine"</b> —Concept of device which was intended to perform an assigned sequence of calculations and have the ability to store numbers, print results, go back and cycle over any desired part of the computation |
| 1850             | Parnallee                            | <b>Keyboard Adding Machine</b> (Parnallee Calculator)—Used a calibrated shaft that was raised through the machine's top when keys were depressed (limited to one column of digits at a time with a maximum total of 50)           |
| 1872             | Baldwin                              | <b>Calculator</b> —Invented pinwheel method of counter control  |
| 1886             | Felt                                 | <b>The Felt "Macaroni" Box</b> —The first multiple order key driven calculator; a forerunner of the comptometer   |
| 1890             | Hollerith                            | <b>Punched Card Code</b> and several machines for sorting and counting punched cards—founded corporation that began IBM   |

Fig. 1-1. A history of evolutionary development leading to the digital computer.

1939 at the University of Pennsylvania's Moore School of Electrical Engineering by a group under the direction of J. P. Eckert, an engineer, and J. W. Mauchly, a mathematician. ENIAC, completed in 1946, used about 1500 relays and 18,000 vacuum tubes and could complete an addition sequence in about 1/300 the time required by the Harvard Mark I.

ENIAC had several disadvantages that severely restricted its uses, among which were a limited storage capacity and the difficulty of presenting instructions. Instead of correcting the deficiencies of ENIAC, Eckert, Mauchly, and their associates immediately began work on a new machine. This new machine, known as the Electronic Discrete Variable Automatic Calculator (EDVAC), was a stored program-type computer, using an acoustic-delay storage device that greatly increased its storage capacity.

The first operational stored program computer was the Electronic Delay Storage Automatic Calculator (EDSAC) constructed at Cambridge University in England in 1949, under the direction of Dr. M. V. Wilkes, then Director of the Mathematical Laboratory.

During the 1940s, many other experimental computers were constructed. Among them were the IBM Selective Sequence Electronic Calculator, the Harvard Mark II, and the

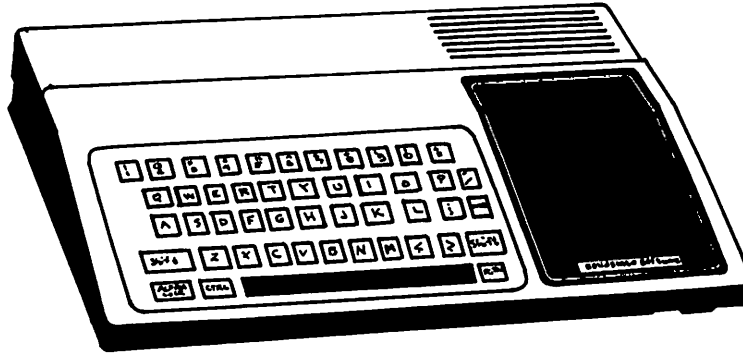
Bureau of Standards Eastern Automatic Computer (SEAC). After EDVAC, Eckert and Mauchly went into business for themselves and constructed BINAC for the Northrup Corporation. The first commercially available computer was the Universal Automatic Computer (UNIVAC) built originally in 1950 by the Eckert Mauchly Corporation and intended for use by the Census Bureau.

The original UNIVAC was a highly versatile computer built to handle both alphabetical and numerical data. It had a cycling rate of 2.25 million pulses per second, a 1,000-word internal memory using a mercury delay system, and ten servo units for handling magnetic tape inputs and outputs. It also had the ability to use a high-speed printer as an output device.

Many improvements have been made since the introduction of UNIVAC, and the computers of today are far superior. Cycling times are now on the order of picoseconds, and by using magnetic devices for internal storage (memory), capacities have been increased to hundreds of thousands of words.

Since 1950, many other companies, both large and small, have entered the automatic data processing field. These include UNIVAC division of Sperry-Rand, GE, IBM, CDC (Control Data Corporation), Honeywell, Datamatics, Texas Instruments, among others.

## Chapter 2



# Home Computer System

Texas Instruments Incorporated announced an enhanced version of their popular TI-99/4 home computer on May 29, 1981, at the Summer Consumer Electronics Show in Chicago, Illinois. The TI-99/4 was a multi-unit computer system. It contained a 16-bit microprocessor and was one of the first 16-bit home computers to be offered. The new model, announced in Chicago, is designated the TI-99/4A, and among other things, includes a new keyboard. The console retains the compact, profile, speech capability, and color graphics software capability of the older TI-99/4 but also offers improved functionality. The latter word belongs to Texas Instruments. What it means to the computer user is a lot more versatility. Now you get upper- and lowercase letters, along with numbers, punctuation, and symbols, all arranged on a standard typewriter

keyboard. The shift key activates the uppercase characters and can be locked in place by depressing the Alpha Lock key.

The TI-99/4A (Fig. 2-1) has a built-in automatic repeat function that was not available on the older TI-99/4. This is useful in formatting tabular data and in developing fairly complex graphic designs. By depressing and holding down any alphabetic or other symbol key, that character is repeated until the key is released.

One of the special keys is the Function key (FCTN). When this key is depressed, along with certain designated number keys, you get special computer functions such as Delete, Insert, Erase, Clear, Begin, Proceed, Aid, Redo, and Backspace. The Control key (CTRL) is used specifically for communications applications. This includes communicat-

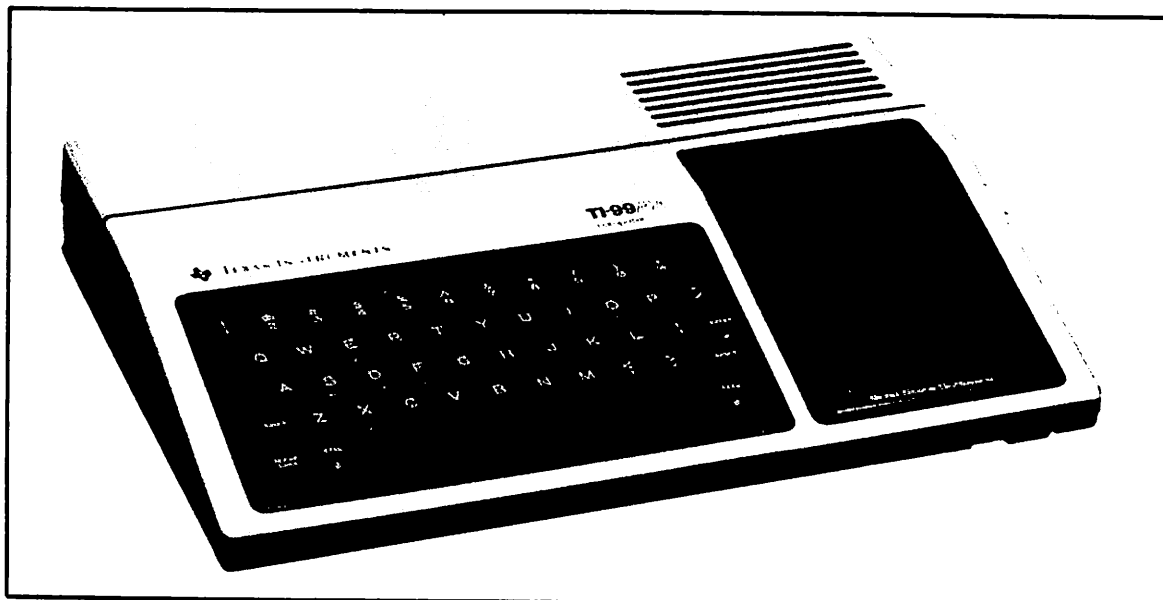


Fig. 2-1. The TI-99/4A microcomputer.

ing with another home computer or even with a remote home information service. A two-level strip overlay is included with the console to help you identify the keys that are used in combination with the FCTN and CTRL keys (Fig. 2-2). For additional identification, control keys and the CTRL key are specified by red symbols; the function keys are gray symbols.

TI BASIC, the standard language on the TI-99/4A, accepts both upper- and lowercase characters, except in a few special instances. When the List command is entered, the screen

displays all reserved words, variable names, and subprogram names in capital letters for easy identification. Actually, the lowercase letters are smaller reproductions of the uppercase character set. A lowercase R is not formed differently from an uppercase R, it is just physically smaller.

The TI-99/4A, like its predecessor, is equipped with a module slot so that Solid State Command Modules may be inserted. These modules may contain Extended BASIC, a more powerful version of TI BASIC, or any of a

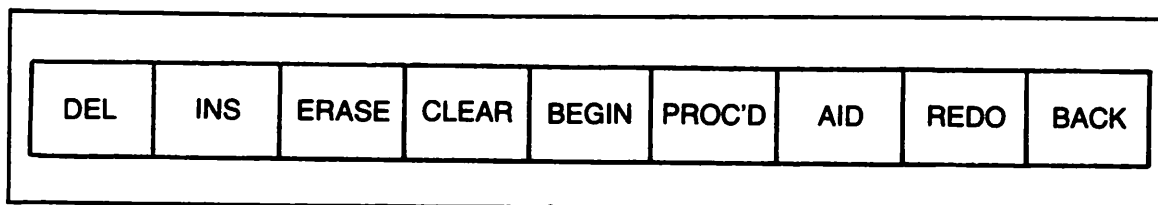


Fig. 2-2. A two-level strip overlay fits above the top row of keys to identify special function keys.

hundred or so different programs. This computer will also store and read programs on and from cassette tape with the optional Cassette Interface Cable. A disk drive system is available as well for those users who need the additional speed and convenience disks provide. In most instances, however, users will probably stick to the less expensive cassette tape storage medium.

The software for this computer is extensive and allows individuals with no prior computing experience to begin enjoying the machine. By adding peripherals such as disk drives, printers, speech synthesizers or telephone modems, the TI-99/4A becomes a quite powerful problem solver for advanced users. It is the only inexpensive home computer that can be programmed to include 16 colors, numerous sound effects, five musical octaves (with three-part harmony), and speech in the same program.

Another option is the 10-inch color monitor that accepts the composite video signal directly from the TI-99/4A. This is an option that many users won't buy because a modulator is supplied to connect the console with a television receiver. The picture quality is better with the 10-inch monitor, but the display on a standard color television screen is quite good. The fact that the color monitor option costs more than the basic computer itself is a big factor in selecting which type of monitor to use. If you're going to be heavily involved in color graphics work with the TI-99/4A, you will certainly want to consider the excellent reproduction provided by the small, 10-inch monitor.

Resident memory in the TI-99/4A is specified as 72K bytes. This can be a little

misleading since most companies rate their random-access memory only. The TI-99/4A includes 16K bytes of data storage memory. This is also known as random-access memory, read-write memory, or simply RAM. This is generally the minimum amount of RAM recommended for any type of computer system, although very few single programs will use even half of the available memory. The rest of the memory is taken up by ROM (read-only memory), and the memory circuits. With 16K RAM, any program you run cannot require more than 16K of memory.

This limitation doesn't mean that once you've written one program that consumes about 16K, your machine is useless until more memory is added. Two different programs cannot be run simultaneously on this microcomputer. When you write a program, you may want to store it in permanent memory. This is usually a cassette tape for the TI-99/4A, but it may also be a magnetic disk. Once the program information is transferred from RAM to disk or cassette, you are free to erase the program from RAM and begin a new one. When this one's completed, you can store it on cassette or disk as well. Suppose you want to run one of the stored programs. You simply load the information from the storage medium into RAM.

Any time a program is run, it must be contained in RAM. This applies whether you're entering the program from the keyboard or from any type of storage device. You can run any number of programs on a computer with 16K of RAM as long as any one of the programs does not exceed a length that requires more than 16K of memory. I find that the majority of individuals who buy a microcomputer actually



buy more memory than they will ever use. It takes a long program to fill 16K RAM, so I suggest you try the minimum configuration to determine what your memory requirements are.

In addition to the 16K RAM, the TI-99/4A contains 26K bytes of read-only memory (ROM). These chips contain the instructions needed for your computer to know it's a computer. Additionally, they include information to allow the microprocessor to accept information from the keyboard and perform all of the functions that allow you to enter a program in TI BASIC.

The 26K bytes of ROM and 16K bytes of RAM, a grand total of 42K bytes, is all the memory that comes built *into* the basic TI-99/4A computer. When Texas Instruments states that memory resident in the basic system is 80K bytes, they are including in that figure 38K bytes of ROM. This ROM memory is actually contained in an *accessory plug-in* memory module. This extra ROM may contain an optional program or new language. It accomplishes in ROM what might normally be accomplished in RAM through software.

Those of you who require more than 16K of RAM will be happy to know that RAM data storage is expandable to a total of 52K bytes through the Memory Expansion Unit. This option adds 32K bytes to the resident 16K bytes of RAM. A 4K Mini-Memory cartridge is also available.

## HOME VERSUS PERSONAL COMPUTERS

The TI-99/4A is a home, rather than a personal, microcomputer. The home computer is often called a low-level microcomputer, and the personal computer is often

called a high-level microcomputer. The differences between the two groups are becoming less distinct as home computer design and capabilities are being constantly upgraded. The 16-bit microprocessor in the TI home computer, for instance, is far more advanced than the 8-bit microprocessors used in many personal computers.

In general you will find fewer options (such as communications interfaces, disk drives, and mass storage systems) available for home computers than for personal computers, or if available, they will cost more than the cost of the computer itself. Home computers generally have fewer operating features than personal computers. Here too, the differences are disappearing. On the TI, for instance, it is easy to edit, or make program changes in a line without having to retype the entire line. This is a feature that was not available on most personal computers only a few years ago.

The character set and screen display on a personal computer will generally be of higher quality than found on home computers. A true personal computer can also more easily and more quickly display charts and graphs through direct programming methods using the features built into the machine. Because of this capability personal computers may also be referred to as small business computers.

Although many home computers may accomplish the same results, they most often require special software or hardware packages to do so and will do so much slower.

Screen displays are also handled differently. On most personal computers the first line of text appears at the top left-hand corner of the screen, and no scrolling (the automatic

movement of displayed lines upward) occurs until after the screen is full. On the TI micro-computer, as on most home computers, the first line of text appears at the bottom left-hand corner of the screen. Upward scrolling begins with the very next line that is typed. This is the easiest way to accomplish on-screen display because it requires less ROM. For most home computer uses this is really no drawback, but it can prove troublesome in more sophisticated personal or business applications.

Another difference in screen display is screen format. Most home computers display characters in only one format, and it is usually about 30 columns wide by 25 rows, that is approximately 30 letters can be printed in a single row and you can get up to 25 rows on the full screen. The TI-99/4A allows a 32-character wide format with 24 rows on the full screen. True personal computers usually display text in at least two different screen formats: one format approximately 80 columns wide and 24 or 25 rows deep (called high-resolution text mode) and one format 40 columns wide and 25 rows deep (called medium-resolution text format). This type of screen formatting can be a great aid in clearly displaying information without having to do a lot of hunting through screen garbage. Although most home computers display information quite accurately, in text mode you may have to look for a second or so longer to pull it out from other on-screen information.

Because personal computers generally offer more keyboard functions than home computers they make inputting information easier. For example, on the TI-99/4A home computer, you must press the FCTN key simul-

taneously with another key to move the cursor to the left or right. On most personal computers, cursor movement is controlled by a separate key panel that requires only one finger to operate.

While the extra features of a true personal computer may be very nice, you pay through the nose for them. Keep in mind that the TI-99/4A is available from most discount stores for a little less than \$300, and for even less while Texas Instruments continues to offer a \$100 rebate. In comparison, the personal computer I have sells for about \$2,500. With options such as a disk drive, special monitor, and memory expansion package, it costs well over \$6,000. In other words, my personal computer cost thirty times what the TI-99/4A home computer costs.

The question every potential computer owner must ask is, "Do I need everything the expensive machine offers?"

The TI-99/4A is a home computer designed to be used in the home by the average homeowner and home dweller. Texas Instruments states that adults and children with little or no knowledge of computers can easily use the TI-99/4A. Texas Instruments has attempted to achieve (and in my opinion, has) a simple machine built around a high-powered microprocessor. Simplicity is the key here, ergo the standard typewriter keyboard and excellent software packages that are offered.

As an experienced computer operator, I would love to see a separate numeric keypad on the TI-99/4A, just like the one on my personal computer. However, such a keypad is certainly not mandatory, nor even desirable for the beginning or casual home computer

operator. It would certainly add to the cost and the complexity of operating the machine.

For the money, the TI-99/4A is one of the best computer buys on today's market. I have tried many different home computers, and I especially like this one because of its standard typewriter keyboard. Many home computers do not offer this feature. The main reason I like the TI though is because its language and keyboard operation are identical or similar to those contained in personal computers. Some home computers seem to program in a completely different manner from most personal computers. They don't, therefore, make very good training aids for children who may someday hope to convert to serious personal computing. If you learn to efficiently program the TI-99/4A, you will be able to easily convert to a full-scale personal computer when the time comes. If you have programmed for quite some time on a personal computer, you will be able to easily convert to the TI-99/4A. There are marked differences in the two types of machines, but there are enough similarities to make converting to either an easy task.

## THE CONSOLE

The TI-99/4A console is shown in Fig. 2-4. This is the master unit and contains the microprocessor, the keyboard, Solid State Software™ Command Module input slot, and the video/audio interface.

The TI-99/4A console or master unit is designed around the TMS9900 16-bit microprocessor. Its architecture is 16-bit with 16 general registers. It can address up to 64K bytes of memory and contains four interrupt lines.

Also found in the console is a video dis-

play processor chip. It controls display memory and generates the composite video signal used to drive a composite monitor or a video modulator when a color television is used. It displays 24 lines of 32 characters in an 8-by-8 dot matrix. The processor provides 16 colors and 32 sets of 8 characters each, with different foreground/background colors. The video display processor addresses up to 16K bytes of RAM for the central processor or display purposes.

There is a third chip in the console unit, the sound controller chip. This chip offers three voices with a 5-octave musical range. It also contains a 15-bit programmable noise source and offers a 100-milliwatt audio output with 30 dB control in increments of 2 dB.

The keyboard is contained within the console. It's known as a 48-key Staggered Qwerty, full travel type. It is very similar to some typewriter keyboards, although you will note a few differences in the placement of certain keys. The number and letter keys, however, are in the usual location.

The console unit itself also contains the 14K byte BASIC Interpreter, along with a graphics language interpreter.

The console is powered from a standard 110-Vac source. The power supply is located in the power cord. There are two arrangements here. One model may locate the transformer a distance from the plug, while another type has the plug-in type of transformer arrangement. Here, the plug prongs exit the transformer body and the entire unit rests against the wall receptacle. It doesn't matter which type you get because both are electrically equivalent. The console draws a maximum of about 20 watts.

The removable power cord attaches at the rear of the unit by means of a 4-pin plug. The location of the various outlets on the console is shown in Fig. 2-3. At the far left, the cassette interface cable is connected to a 9-pin D outlet. Immediately to the right of this receptacle is the console power receptacle, and to the far right is the 5-pin connector for audio/video output from the unit. This connector accepts the cabling from a composite video monitor or from a video modulator when a television receiver is to be used. To one side of the console is another receptacle to accept the joysticks, or as Texas Instruments calls them, the Wired Remote Controllers. This receptacle is identical to that found on the rear of the unit for interfacing with a cassette tape recorder. Do not get these two receptacles mixed up.

Figure 2-4 shows the front of the console. The power switch is located at the lower front right near the Command Module Software slot. All of the module software is inserted in this slot. To the right of this slot is an output jack for optional peripheral accessories, such as the RS-232 interface. Since the power supply is located just beneath the module software slot, it is normal for there to be a bit of heat in this area. The plastic casing in this area will become warm, but not so hot that it's uncomfortable to touch.

## THE KEYBOARD

One big advantage of the TI-99/4A over the previous TI-99/4 is the typewriter keyboard (Fig. 2-5). For the most part, the keyboard looks and operates like a standard

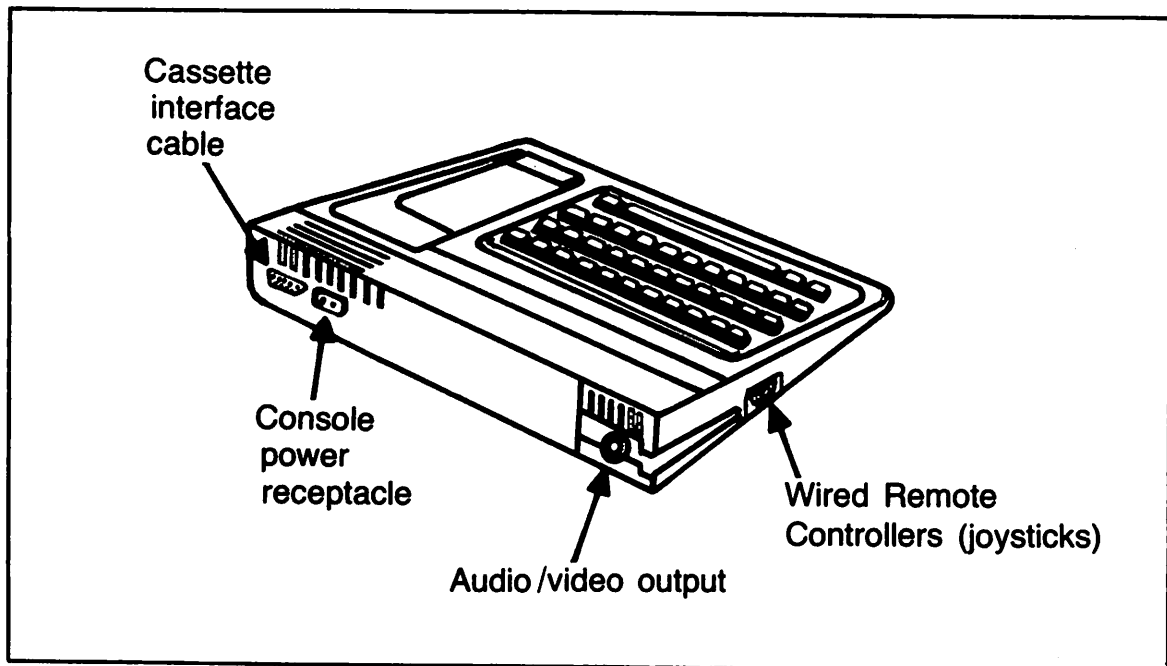


Fig. 2-3. The location of various outlets on the TI-99/4A console.

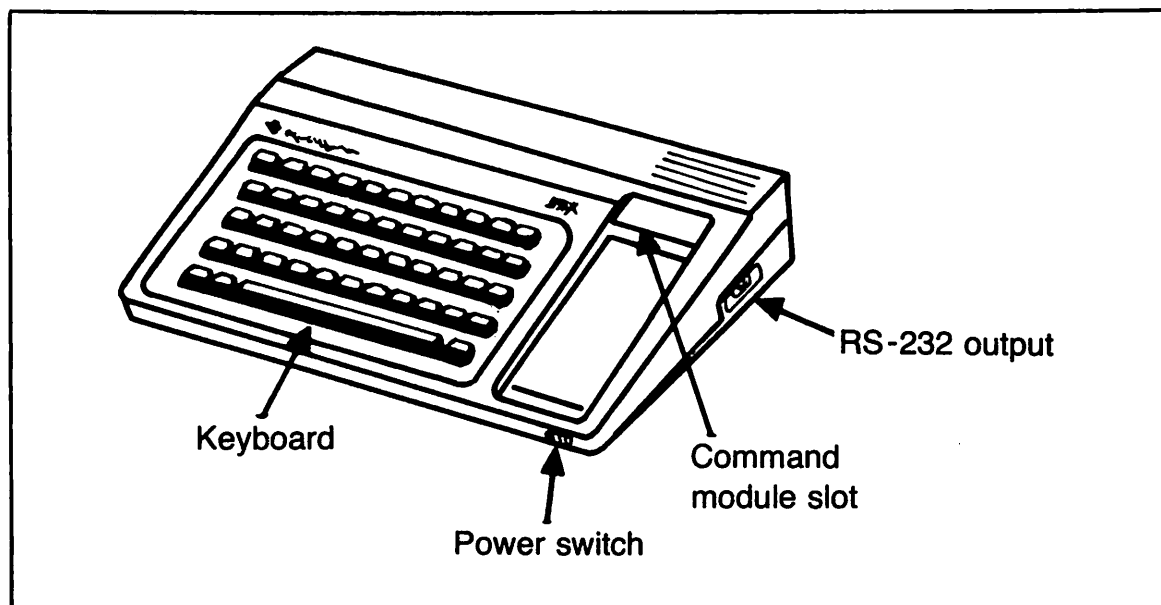


Fig. 2-4. Front view of console.

typewriter keyboard. When you press any key, its lowercase character appears on the screen unless you hold down the shift key at the left or right. When this is done and another key is simultaneously pressed, the uppercase character for that key appears on the screen. There is also an Alpha Lock key at the bottom left of the keyboard. When this is depressed, the keyboard is locked into uppercase mode. Except for the alphabetical keys, each key's uppercase character is printed at the top of the key face. The lowercase character is printed at the bottom. This is not done for the alphabetical keys because the characters are formed in exactly the same manner for both upper- and lowercase. The only difference between uppercase and lowercase letters will be their size. Some of the keys have special functions that are accessed by depressing the FCTN key

simultaneously with another key. Some characters formed with the aid of the FCTN key are printed on the front or side of the corresponding keys, rather than on the top, as is normally the case with most keyboards.

All alphabet letters are entered to the computer with the 26 alphabet keys. To enter upper- and lowercase letters, you use the Shift key, just like on a typewriter. To enter all uppercase (capital) letters, press the Alpha Lock key, which locks the alphabet keys into uppercase mode. The Alpha Lock key does not affect the number and punctuation keys. Pressing the Alpha Lock key one more time will "unlock" the uppercase mode, and return the keyboard to normal lowercase operation.

The number keys on the TI-99/4 console are on the top row. Unlike on some typewriters, you cannot type the lowercase letter

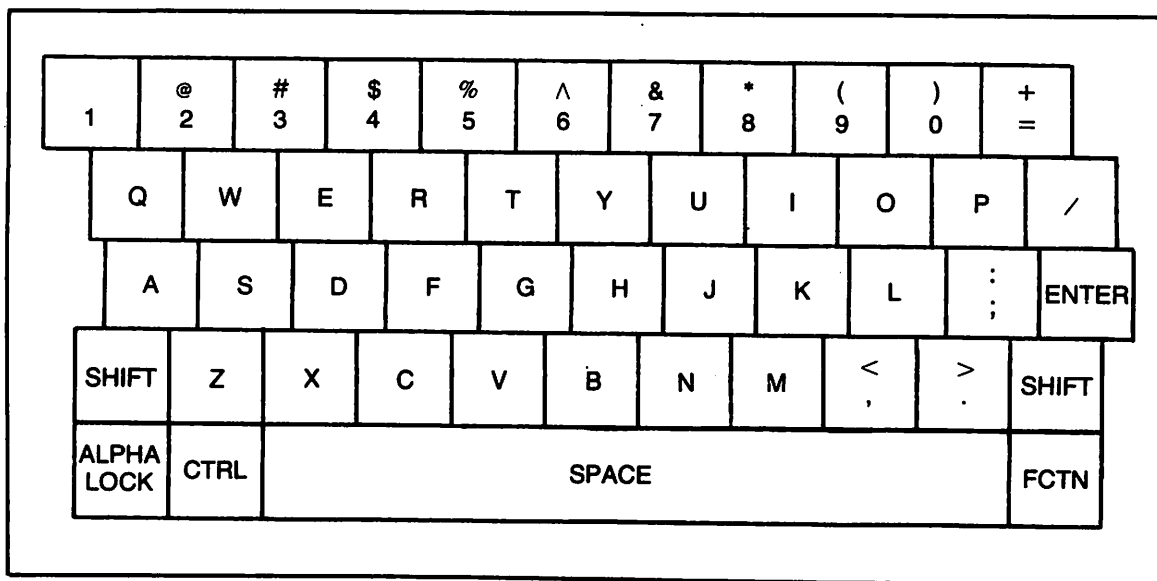


Fig. 2-5. The TI-99/4A contains a typewriter-like keyboard.

“elle” (l) as a substitute for the number one (1), and you can’t interchange a zero (0) and the uppercase letter “oh” (O). The computer screen displays the letter O with square corners and the number zero with rounded corners (Fig. 2-6) to make it easier for you to distinguish between them. By pressing the Shift key and number keys, symbols (rather than numbers) become available.

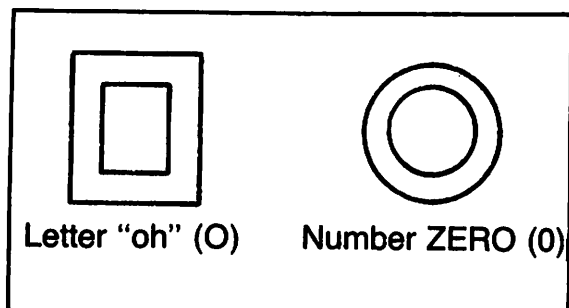


Fig. 2-6. Display of the letter O and the number zero on the TI-99/4A.

The keyboard has most of the punctuation and symbol keys of a standard typewriter. There are also a few special ones that have particular applications in computer programming and are not found on most typewriters. These punctuation and symbol keys follow the same upper- and lowercase format as other keys, and each has two symbols printed on its face. To print the top symbol, you must use the Shift key. To print the bottom symbol (lowercase), simply strike the key. Some punctuation marks (quotation marks, for instance) appear on the front of the key. The only way you can type quotation marks or any other symbols is by simultaneously holding down the FCTN key at the lower right of the keyboard while pressing the appropriate key. The special function keys with arrows on them do not print anything on the screen. These are used to move the cursor during the line editing process.

There are other special function keys as well. These are really the number keys that are pressed simultaneously with FCTN. The following is a rundown of the special FCTN key combinations.

When you press FCTN and the key with the arrow pointing toward the left, the cursor moves to the left. The cursor does not erase or change any characters on the screen, but in edit mode it allows you to insert or delete a character by means of other commands. The right arrow key moves the cursor to the right when pressed simultaneously with FCTN. The other two arrow keys, one pointing up and the other pointing down, have different functions according to the application where they are used and the software itself. When you enter a program, pressing FCTN and the up arrow key will cause the lines on the screen to scroll upward.

A special overlay is provided to owners of the TI-99/4A. This plastic strip is fitted over the top of the keyboard to indicate which number key performs which function when pressed simultaneously with the FCTN key. The key bearing the number 1 is labeled as DEL by the overlay. When this key is pressed simultaneously with the FCTN key, a letter is deleted from the screen. This could also be a number or other character. You usually use this while in the edit mode, or before a program line is entered. Using the cursor positioning keys (right and left arrow keys) discussed earlier, you place the cursor beneath an improper character and then press FCTN and the number 1 key to delete it. When the character is deleted, all other characters move one space to the left to fill in the empty space. By

continuing to hold down FCTN and 1, more letters or characters are deleted. All letters to the right of the cursor move toward the cursor.

When you press FCTN and the number 2 key, the insert mode (INS) is accessed FCTN plus 2 is used to add letters or characters to a program line. Let's assume that instead of typing `PRINT X`, you typed `PRIT X`. To correct this you enter the edit mode by typing `EDIT` and the line number that contains the error. Using FCTN and the cursor positioning keys, you move the cursor until it rests beneath the T. Now press FCTN and the number 2 key, and you are in insert mode. Type the letter N, and the N is inserted just before the T. Pressing the Enter key commits this line as edited. The Delete and Insert functions on the TI-99/4A can save a great deal of time by letting you quickly and easily correct mistakes.

When you press FCTN and the number 3 key, the entire program line you are presently typing is erased. This must be done before you press the Enter key to commit the line to memory. This is handy in situations where you might be entering a program that is printed in a book. Midway through the entering of a line, you discover that you skipped a line and this one is completely wrong. Press FCTN and the number 3 key, and it's gone. Then begin typing in the correct line.

When you press FCTN and the number 4 key, all execution stops. The key is also used to clear any information from the screen that was typed before pressing Enter. Its first feature (execution halt) is most important. On other computers, this might be called a break key or a halt key.

To stop a program in the midst of execu-



tion, press FCTN and the number 4 key simultaneously. You can now enter other commands in direct mode.

The other numerical keys have special functions in software applications and are labeled by the same overlay strip. Numeric keys 1 through 4, when used with FCTN, speed programming by making it easier to correct errors. Incidentally, the overlay strip is laid out in two levels. The top level of functions are identified by a red dot and are called control keys. The second level of functions is identified by a light gray dot. These are accessed by pressing that key while holding down the FCTN key. The control key level of functions is accessed by holding down the CTRL key and the numeric key simultaneously.

The TI-99/4A has several math keys used to insert characters to indicate math functions. The plus (+), minus (−), and slash (/) indicate addition, subtraction, and division. The asterisk key (\*) indicates multiplication, the equal sign key (=) means equal. There is also a caret key (^) used to indicate the raising of a number to a certain power. For instance,  $5 \wedge 2$  indicates 5 raised to the second power, or 5 squared. It is necessary to use the Shift key to obtain this character, which is found on the number 6 key in uppercase mode. Two other mathematical symbols are found on the keys at the lower right of the keyboard. In lowercase mode, these keys type the comma (,) and the period (.). In uppercase mode comma key becomes the less than symbol (<), and the period key is the greater than symbol (>).

Another feature of this keyboard is the automatic repeat. If you hold down the space

bar or any character key for more than about one second, it goes into repeat mode. To type a series of 5 spaces, press the space bar once and hold it down until your 5 spaces have been printed. The same applies to any character key. This comes in handy in certain graphics applications, where it may be necessary to print a series of 16 Fs, for instance.

The space bar may be used to delete or erase characters from a program line before the line has been committed to memory by pressing Enter. If you want to erase an entire word, you simply position the cursor at the beginning of the word and hold the space bar down until all letters in the word have been replaced by spaces. (Of course, you can use the FCTN key and Delete key for the same basic purpose.)

The feel of the keyboard is quite important to typists who depend on “feel” to put them into a typing rhythm. I would not call the TI-99/4A keyboard crisp, but rather pleasingly spongy and quiet. It does not have the feel of any keyboard I have ever used before. This doesn’t mean that it’s bad, just different. It’s a quiet keyboard; you don’t hear the various clicks present with other types of computer keyboards. After ten or fifteen minutes of practice, one becomes pleasingly adjusted to the keying action, and good typists can fly along at a comfortably rapid pace. I rate the keyboard as excellent for an inexpensive home computer.

## ACCESSORIES

A variety of accessories are available for the TI-99/4A computer.

## Video Modulator

One item listed as an accessory for the older TI-99/4 is now standard with the TI-99/4A. This is the video modulator (Fig. 2-7), which plugs directly into the console and attaches to the 300-ohm antenna terminals of your color television receiver. The TI900 Video Modulator is also called by several other names, such as Sup'R Mod, depending on the company from whom you buy it. This is a high-quality Korean-made modulator bought in bulk by many companies and sold under different names. Texas Instruments made a good choice with this modulator, as it's probably one of the most popular types for microcomputer users.

This is an audio and video modulator, so it transmits video information and audio information on the same carrier. The circuitry of your television receiver separates the sound and

picture information just as it does with the transmissions from a television station. The picture information is displayed on the picture tube, and the sound is emitted from the internal television speaker.

The Texas Instruments modulator is switch-selectable between channels 3 and 4. Connect the unit to the back of your television set by means of the short length of 300-ohm cable that exits the top. There is a terminal strip on the side of the modulator, to which you connect your television antenna leads. A switch at the center of the modulator lets you select either the computer or the television antenna for input to the television set. When you want to operate the computer into the television, set this switch in the "computer" position. The "television" position allows for normal television viewing.

The channel selector switch determining

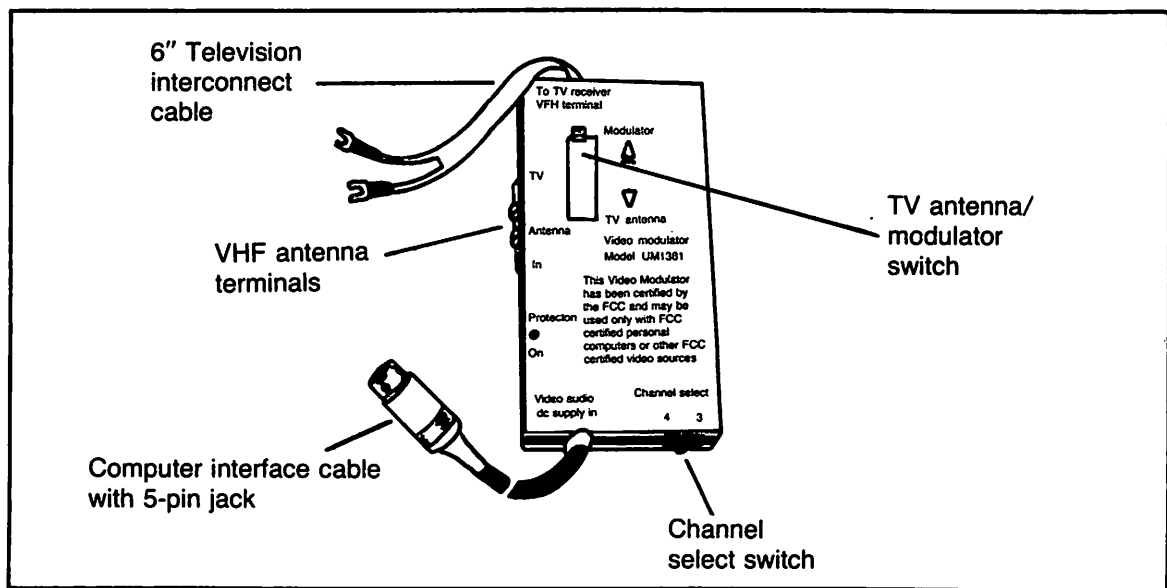


Fig. 2-7. TI900 video modulator.

the output frequency of the modulator is found at the bottom front. In the left position, the output is on television channel 4; in the right position, your computer output is seen on channel 3. If you have a strong local station on either of these channels, select the other one, or for that matter, whichever setting gives you the clearest screen.

Don't be surprised if you hear a few clicks, pops, and other sounds from your television speaker. This is common and can be corrected by turning down the volume. This assumes that you are running a program that is not using the TI-99's sound production capabilities. When using the computer to produce music, leave the television volume up, because the music comes from this speaker.

Another good feature of the modulator supplied by Texas Instruments is its built-in protection. If the computer overdrives the modulator (supplies too much video signal), the protective circuit temporarily disables the device. When the protection circuit is activated, a red light-emitting diode (LED) on the front panel is triggered. You can reset the modulator by turning the mode select switch on its front face to "television" and then back to "computer" again. If the light continues to be triggered, this could be an indication of a defect in the computer or even the modulator. For best results, try to place the computer console at least 3 feet away from the television receiver. This can avoid unwanted video and audio interference from the interaction of the two.

### **Color Monitor**

Shown in Fig. 2-8, the Model PHA 4100 is a high-quality color monitor with a 10-inch

screen specially matched for use with the TI home computer. The computer produces a display that has 24 lines of 32 characters per line and a 192 by 256 dot-density ratio. The monitor provides excellent color resolution and picture quality. It connects to the TI-99/4A computer via a special cable. This eliminates the chance for interference and distortion that can occur in the tuner of a standard television. This is a true color monitor and accepts the composite video directly. Therefore, the PHA 4100 has no tuning. The picture quality using a composite color video monitor is almost always superior to pictures from even the best color television receiver.

The monitor accepts the NTSC composite video signal at a nominal 1-volt peak-to-peak value. Audio input is delivered at 1 to 2 volts peak-to-peak. The operating, or scan, frequency of this monitor is 15.750 kHz. In addition to the standard on/off and volume controls, you will also find controls for sharpness, tint, color level, contrast, brightness, height, vertical hold, and horizontal hold. The monitor operates from the standard 110-volt household line, consumes about 65 watts, and weighs about 22 pounds.

Texas Instruments warrants all components of the color monitor with the exception of the picture tube for a period of three months from the date of purchase. The picture tube is warranted for a period of two years from the date of purchase.

### **Thermal Printer**

Shown in Fig. 2-9, the Solid State Thermal Printer gives printed copy of any program and/or data run on the TI-99/4A. The printer

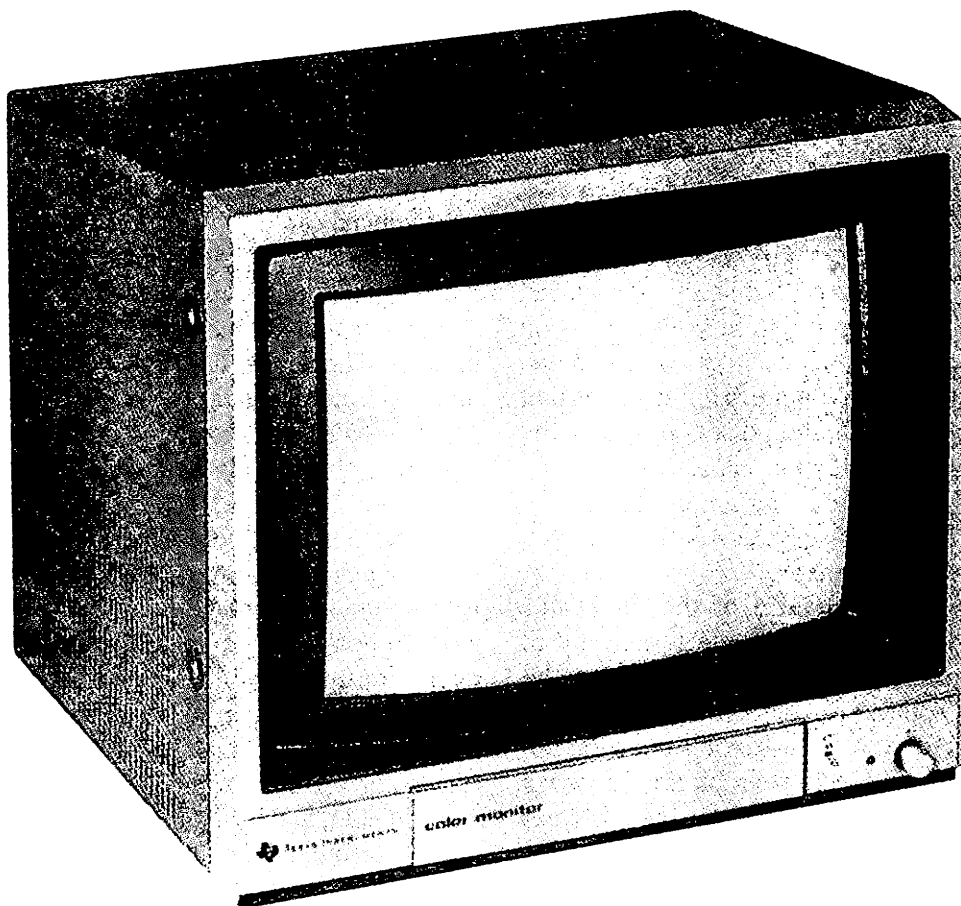


Fig. 2-8. The TI 10-Inch color monitor.

can also be used with some software applications to print screen displays or generate lists and reports.

The printer can print up to 32 characters per line. It contains its own resident character set, but it can also set special characters defined in software. Other special features included with this device allow you to control the amount of paper that is ejected and the spacing between lines. In many computer applications,

a hard copy printout is quite desirable and often necessary. The TI Solid State Thermal Printer is excellent for this purpose. In addition to standard letters and numbers, the printer has 32 predefined graphic symbols for printed charts and graphs.

Printing is done on a 3.5-inch thermally sensitive paper, the same type of paper used for some of TI's printing commercial calculators. The printer is quite tiny and mea-

sures approximately 10 inches by 7 inches by 5½ inches. Because thermal printers normally use fewer moving parts than impact-type printers, they can be far more reliable.

Several software programmable functions are available with this printer. When .U is listed in a program, the printer accepts user-defined characters. If not listed, the printer uses its resident character set. If .S is listed in a program, the printer does not leave any space between printed lines. If not listed, the printer leaves a space which is equivalent to the width of 3 rows of dots between the printed lines. When .E is listed in a program, the printer does not eject paper as the program runs. If not listed, the printer automatically ejects five lines of blank paper for each Open and Close statement for the printer. Five lines of blank

paper are also ejected before and after each List statement.

The printer is controlled from certain TI Command Modules and from TI BASIC. The Open, Print, and Close statements in a program control and output data to the printer to produce printed copy when the program is run. The List command tells the computer to print a copy of the program currently in memory.

The TI Solid State Printer prints approximately 30 characters per second and offers upper- and lowercase characters. It must be used with thermal printing paper (PHA-1950), available only from Texas Instruments. Other thermal papers may damage the printer and void the warranty, which is in effect for 90 days from the date of purchase.

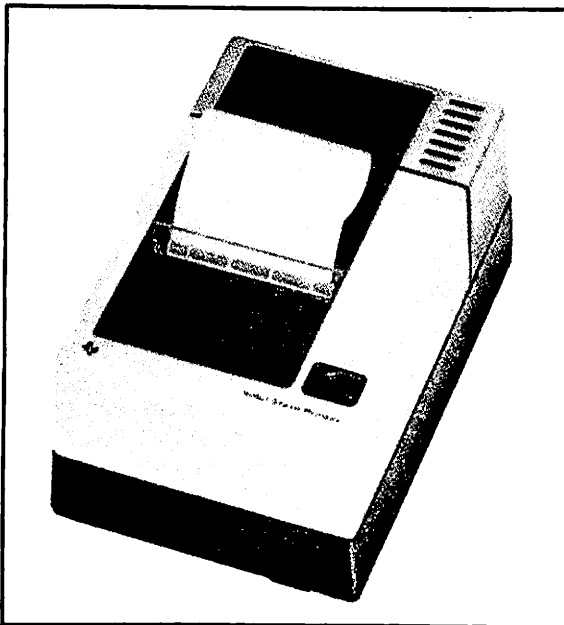


Fig. 2-9. The solid state thermal printer from Texas Instruments.

### Wired Remote Controllers

Shown in Fig. 2-10, these are often called joysticks and allow you greater freedom and versatility in the controlling of graphics, games, and sound on your computer. Without the joysticks, it is necessary to press one or more keys to effect similar control, and this may not be as precise as that offered by the remote controller. The remote controllers are required for certain software offered by Texas Instruments and are a must for programmers who wish to concentrate on developing complex computer games.

### RS 232 Interface

The Texas Instruments RS-232 interface (Fig. 2-11) is a communications adapter that lets you connect serially formatted devices, including those from other manufacturers, to the TI-99/4A. It is not required for the use of

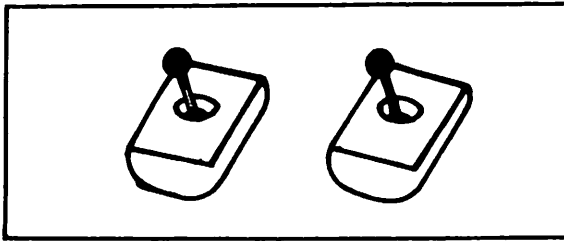


Fig. 2-10. The Wired Remote Controllers from Texas Instruments.

TI-99/4A peripherals and printers manufactured by Texas Instruments (with the exception of the Telephone Coupler where required). With the RS-232 interface, you can list programs on a printer, send and receive data from a terminal, exchange TI BASIC programs directly between TI home computers, etc.

With the addition of the Telephone Coupler (modem) or other standard modem or acoustic coupler and the RS-232 interface, the TI-99/4A can talk with other computers and terminals over standard telephone lines. You can access an office computer or time-sharing network using the TI-99/4A as a remote terminal to send and receive data. This two-way communication permits interactive programming and distributed processing functions to be performed between two or more TI-99/4A computers or by using the TI-99/4A as a remote terminal for another computer system.

The RS-232 interface is programmable so you can exchange data with a variety of serially formatted devices. Using TI BASIC, you can select baud rate, the number of bits, parity, and the number of stop bits. This lets you interface with low- and high-speed peripherals including printers, plotters, video display terminals, and other computers.

The interface is capable of outputting information at rates 110, 300, 600, 1200, 2400, 4800, or 9600 bits per second.

Several software programmable functions are available and include:

☐ *Carriage Return.* Automatically added to the end of all output records unless disabled. If disabled, forces Nulls and Linefeed to be disabled also.

☐ *Nulls.* Normally disabled, but if enabled, will automatically add 6 null characters between the carriage return and the linefeed characters.

☐ *Linefeed.* Automatically added after carriage return character unless disabled.

☐ *Echo.* Automatically echoes all received data on a particular port back to the device connected to that port. Also enables the remote terminal device to edit the data record before the console receives it.

☐ *Parity.* Normally disabled, but if

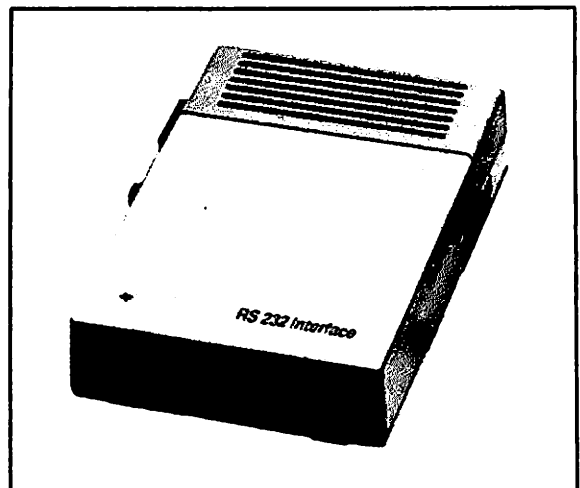


Fig. 2-11. The RS-232 Interface allows communications with serially formatted devices.

enabled, will check for parity errors and generate an error code if any are found.

This unit also contains all the software necessary to interface with the TI Home Computer File Management System and is controlled from TI BASIC. The Open, Close, Input, Print, Old, and Save statements can be used to input and output data through the two ports of the RS-232 interface. The Input and Print statements can input and output data to a terminal. The OLD and SAVE commands can transfer a copy of a TI BASIC program from one TI home computer to another.

Two serial ports are provided by this device, and connection is by means of cables using EIA RS-232-C standard 25-pin male connectors. Seven signals are used:

SERIAL DATA IN  
SERIAL DATA OUT  
CLEAR TO SEND  
DATA SET READY  
DATA CARRIER DETECT  
DATA TERMINAL READY  
SIGNAL GROUND

This device is operated from the ac line (115 volts) and consumes a maximum of 20 watts of power during normal operation.

### **Peripheral Expansion System**

The TI Peripheral Expansion System (Fig. 2-12) lets you add accessories to your computer in a single unit by inserting them in the slots provided. The package includes the expansion system and the Peripheral Expansion Card with a connecting cable. The latter pair combine to serve as an interface between

the computer console and the accessories mounted in the unit. With the Peripheral Expansion System attached to the TI-99/4A, you can quickly change computer capabilities by adding different accessory cards. You can also install a TI Disk Drive in the portion of the compartment designed for this purpose. To access the interior of this accessory, remove the top of the unit and slide in the accessory cards. The system can hold up to seven accessories, including the Disk Drive Controller card, the RS-232 interface, the TI Memory Expansion Card, and several other accessory boards. To handle the increased power drain of the many options this device can hold, a separate 150 watt power supply is provided. The unit weighs about 20 pounds and is operated from the ac line.

### **Disk Memory System**

The TI Disk Memory System is a combination of hardware and software that allows you to store and retrieve data on single-sided or double-sided disk measuring 5¼ inches in diameter. Disk systems accomplish the same thing as cassette tape storage systems, but faster. Each single-sided disk holds over 700,000 bits of information; a double-sided disk holds nearly 1,500,000 bits. The single-sided disk has a holding capacity of about 90K bytes, and the double-sided disk with this system will hold twice this amount.

The Disk Memory System consists of a Disk Controller Card, Disk Memory Drive, and the Disk Manager Command cartridge. The Disk Controller Card tells a disk drive where to position the magnetic head in order to read or write information properly. The con-

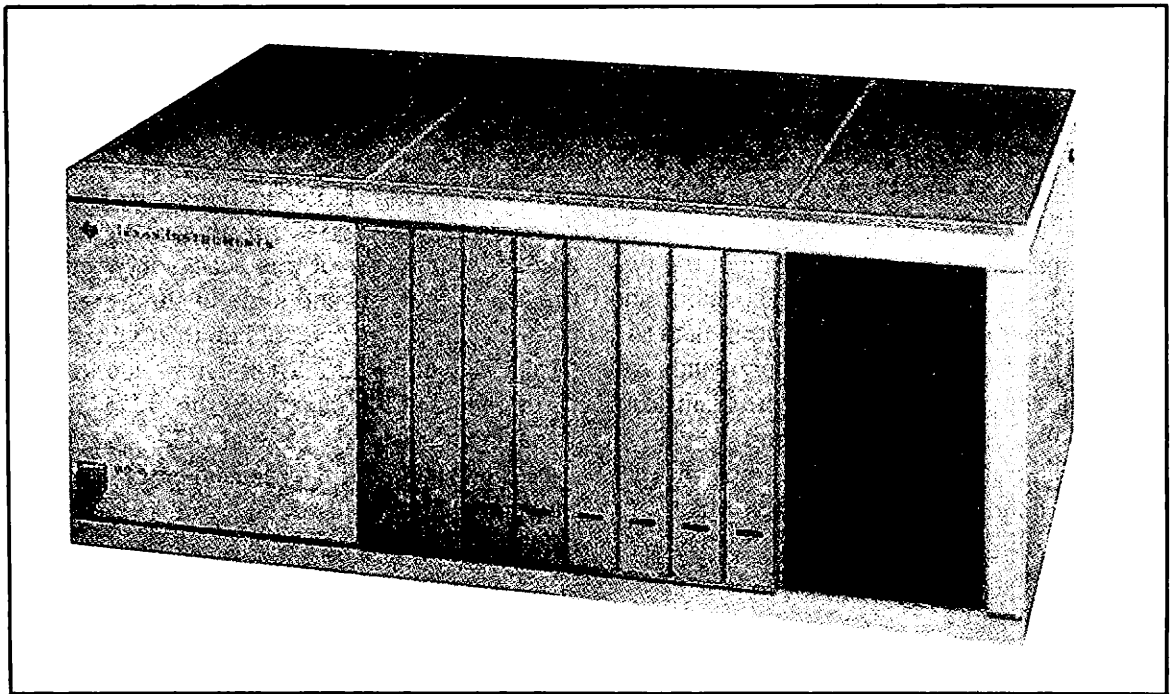


Fig. 2-12. The Peripheral Expansion System from Texas Instruments.

troller also puts an index on the disk, making the data that has been written easy to locate. It can control up to three Disk Memory Drives.

The disk drive spins the diskette at a constant speed and controls the movement of the magnetic head. There is a special compartment in the Peripheral Expansion System for installation of one TI Disk Memory Drive.

The Disk Manager Solid State Software Command Module helps you maintain the information on your disks. Naming and renaming diskettes, renaming files, deleting files, copying files, and copying disks is done with the Disk Manager Module.

Because the control software needed for the disk system is in permanent ROM, in the

Disk Manager Command Module, and in the controller, the disk system uses a relatively small amount of working space in the computer's available memory (RAM).

### **Memory Expansion Card**

The TI Memory Expansion Card adds 32K bytes of random-access memory to the 16K bytes of RAM resident in the TI-99/4A console. The expanded memory is designed for use with TI Extended BASIC and other languages contained on the Command Module. The Memory Expansion Card attaches to the Peripheral Expansion System and requires that TI Extended BASIC or another specialized Command Module be inserted in the



computer console. Most software packages cannot make use of the Memory Expansion Card without the addition of Extended BASIC or some other special Command Module.

### **Telephone Coupler (Modem)**

The Texas Instruments Telephone Coupler (Fig. 2-13) enables your TI-99/4A to send and receive messages through a standard telephone. Use of the Telephone Coupler requires an RS-232 interface unit.

The Telephone Coupler functions as a modulator to convert the data you enter on the console into signals that can be sent over telephone lines. It also functions as a demodulator to convert data received over telephone lines back to its original form. Using the Telephone Coupler is simple. It is powered by a UL-listed

low-voltage transformer, which is included. A cable connects the coupler to the RS-232 interface. A standard telephone headset inserts into the flexible acoustic couplers on the Telephone Coupler. This device may be used with many RS-232 compatible terminals or computer systems for communication over standard telephone lines.

The Telephone Coupler offers two basic modes of operation; called Originate mode and Answer mode. In the Originate mode, you are the party who begins all communications with the remote terminal. In the Answer mode, the remote terminal originates communications. This device is capable of transmitting at a data rate that is continuously variable up to 300 bps.

### **Cassette Interface Cable**

This interface (Fig. 2-14) cable plugs into

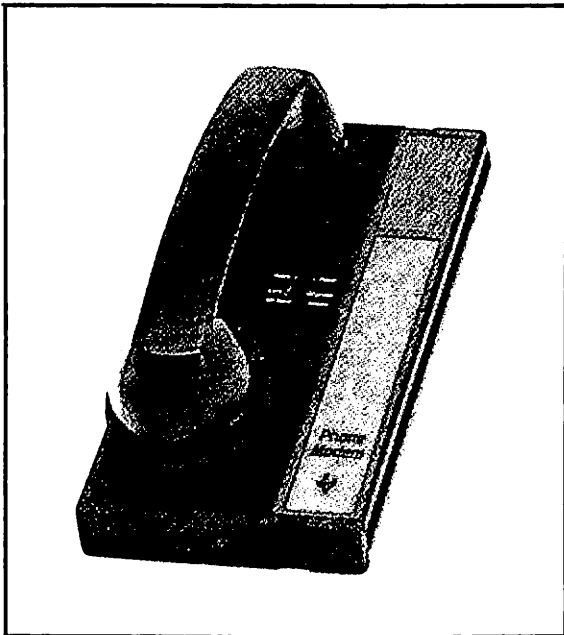


Fig. 2-13. The TI Telephone Coupler enables the TI-99/4A to send and receive messages via a standard telephone.

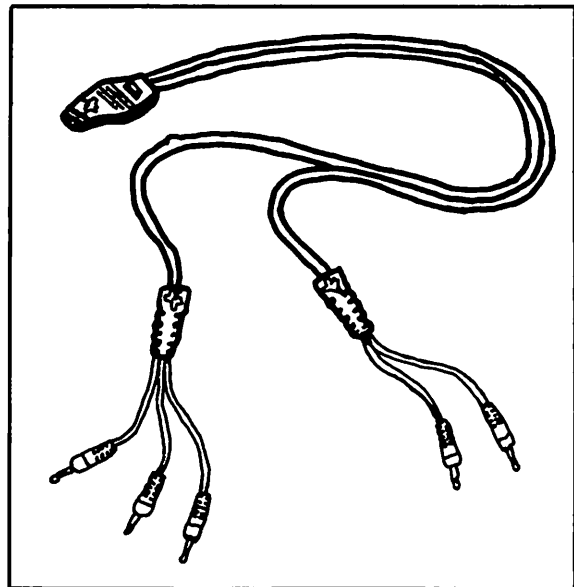


Fig. 2-14. The Cassette Interface cable turns a cassette recorder into a memory storage device for the TI-99/4A.

the TI-99/4A console and allows you to connect a cassette recorder to the computer. I stated earlier that the basic TI-99/4A package is complete, in that it allows you immediately to begin writing and running computer programs, providing you have a television or monitor. With the standard package, however, you cannot store any programs, even if you have a storage device such as a cassette tape recorder. I assume that TI includes the video modulator with their basic package on the assumption that most homes have a television and that a receiver is necessary to use the computer. It's my feeling that most homes also have cassette tape recorders; and therefore, that the Cassette Interface Cable should be provided as part of the base package to allow for the saving of programs. I guess I'm especially touchy about this particular cable, since I incorrectly assumed that one was included with my purchase of the basic console. When I returned home after a long drive, I found that the cable was an option and it was impossible to locate a substitute locally. I think the majority of TI-99/4A owners will use cassette storage since a disk drive costs more than the computer itself.

With the Cassette Interface Cable, you can use the recorder to save and load computer programs. Texas Instruments points out that the use of two cassette recorders is especially helpful for programming applications where a lot of memory space is required. Many cassette recorders can be used with the computer, although each should be equipped with a separate volume control, tone control, microphone jack, remote jack, earphone or external speaker jack, and a digital tape counter. The

latter is not mandatory, but it is a tremendous help in locating the correct tape position for a particular program.

To connect the computer to the cassette recorder(s), insert the 9-pin D connector into the 9-pin outlet on the rear of the computer console. This is the outlet directly to the left of the power cable outlet when facing the back of the unit. On the other end of the cable, the plug with the red wire goes to the microphone jack. The one with the black wire goes to the remote jack and the third one connects to the earphone jack. In most cases, the second set of cassette recorder plugs are not used, so these simply hang free.

Texas Instruments includes a list of recorders from various manufacturers whose products are known to work with the TI-99/4A. This does not include all of the recorders that work, and indeed, most types can be made to work. Some of the inexpensive recorders do not have a tone control, so it may be necessary to adjust the volume to make up for this.

There is one point that should be known. Most cassette recorders operate from internal batteries as well as from house current. I would shy away from the use of battery power when saving and loading computer programs. As the batteries deteriorate, motor speed will slow, and information may be erratically recorded or output to the computer. In many instances, replacing the batteries with a fresh set will correct this; in one instance, it may not. A set of weak batteries in the recorder causes the motor speed to slow up and the tape is pulled across the record head at a slower than normal rate. You may successfully save (re-

cord) the program on tape. With a new set of batteries, the motor speed will pick up to normal again, but the program that was saved while the other set of batteries was in place was recorded at a slower speed. With the fresh set, the playback of this recorded program is faster than intended. This can be disastrous, and you may not be able to retrieve the program.

Also when batteries become weak, motor speed may fluctuate. The tape may travel across the record head for a few minutes at one speed and a few more at a faster or slower speed. This is an even bigger problem and almost assures that you can never retrieve the program. The same thing can happen when ac power is used to drive the recorder, but only when there is a defect in the recorder circuitry. I highly recommend the use of an ac power supply for recording programs.

If you decide to use batteries, make absolutely certain that fresh batteries are installed at appropriate time intervals. You may wish to use rechargeable batteries that can be recharged from the ac line after every usage.

Cassette storage is slow compared with disk storage, but it's also quite inexpensive and the data is stored quite accurately. From a price standpoint, it is the most efficient data storage medium available today. For most owners of the TI-99/4A, cassette storage will be completely adequate.

### **Cassette Program Recorder**

Texas Instruments announced early in 1983 a new, compact (Cassette Program Recorder, (Fig. 2-15), designed for use with the TI-99/4A. The recorder package includes a computer interface cable for the TI-99/4A.

Features of the unit include the ability to be controlled from the TI-99/4A, an Automatic Recording Level Control (ALC), a digital tape counter, clearly marked optimum settings for volume and tone control, color-coded input jacks for easy setup, a pause control, and a built-in condenser microphone. The Program Recorder, with a suggested retail price of \$69.95, can operate either on four C batteries or on ordinary ac power through the included cord.

### **Speech Synthesizer**

The Texas Instruments Solid State Speech™ Synthesizer (Fig. 2-16) makes possible the exciting addition of speech to the TI-99/4A. The Speech Synthesizer requires an optional Command Module preprogrammed for speech, such as the Speech Editor Command Module. These preprogrammed modules allow the Speech Synthesizer to be used without the need to do any programming. Speech can also be included as part of your own programs in TI BASIC.

The Speech Synthesizer is entirely electronic. There are no taped voice recordings or any other traditional recording medium. A vocabulary of words and phrases is permanently stored on chips contained within the Speech Synthesizer. Each word has been transformed into a pattern of bits. When processed, each pattern drives electronic circuitry that rebuilds the requested word and audibly reproduces it through a loudspeaker. The Speech Synthesizer contains a resident vocabulary of over 300 words. Capacity is expandable with optional Plug-in Speech Modules.

The synthesizer docks into the TI-99/4A by means of built-in connectors. Insert one of

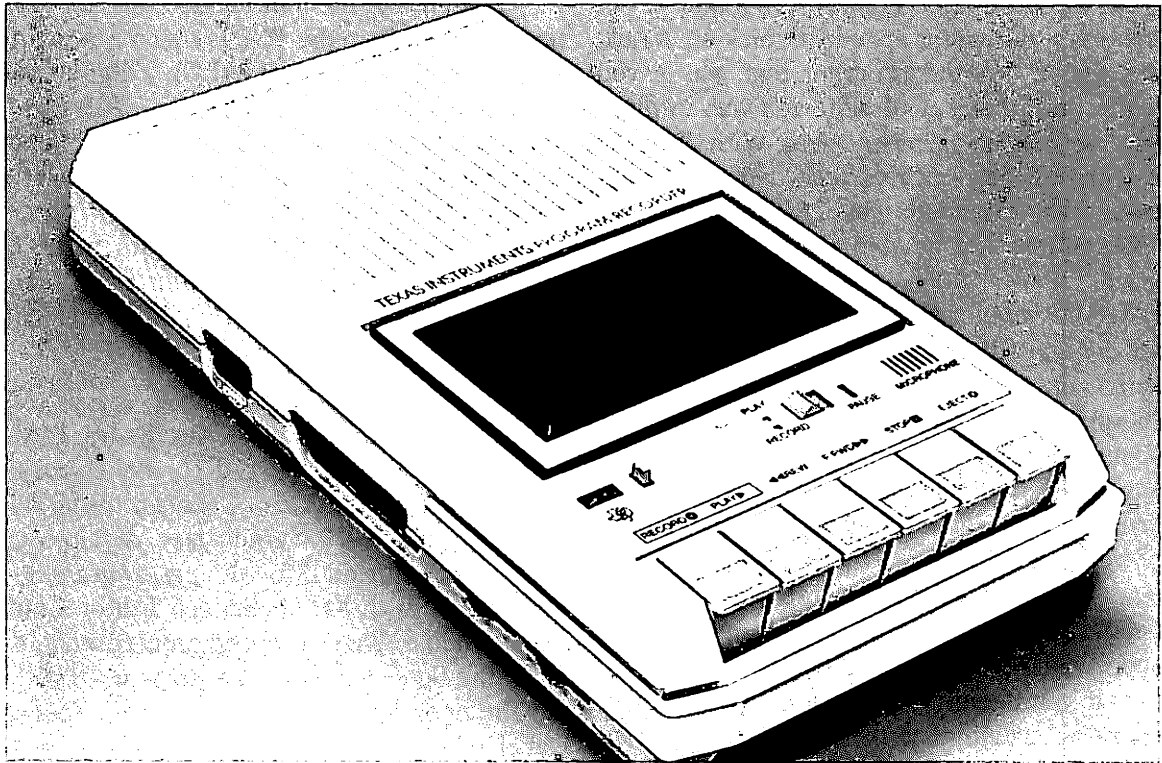


Fig. 2-15. The cassette program recorder from Texas Instruments sells for about \$70.00.

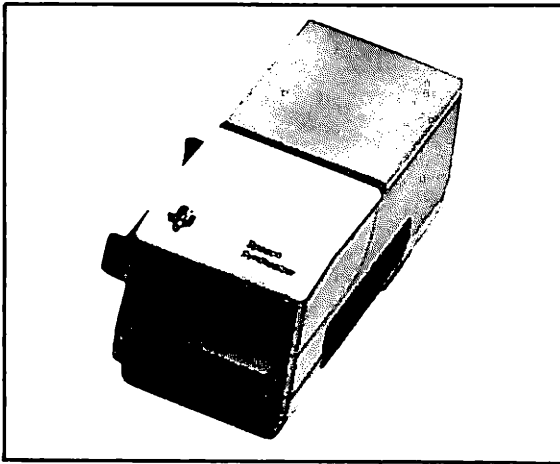


Fig. 2-16. The Solid State Speech Synthesizer made by TI adds a voice to your TI-99/4A.

the Command Modules designed to call up speech from the device, and you are ready to go.

The Speech Synthesizer provides a voice for the computer, creating many new applications and enhancing the effectiveness of existing ones. It can communicate with you even if you are not near the display. It can recite instructions to those unable to read, or where written instructions might interfere with the display. It can provide exciting comments and sound effects in games, and it can reinforce concepts in educational applications.

The Speech Synthesizer can be used in several ways. In one mode, it is controlled by a

Command Module other than the Speech Editor Command Module.

Other methods of operation require the use of the Speech Editor Command Module. Using TI BASIC, words, phrases, or sentences may be recited under program control.

The Speech Editor Command Module can also immediately recite words, phrases, and sentences without having to write a program. In this mode, just type in the desired word, push the return key, and the Speech Synthesizer says the word. Figure 2-17 provides a listing of the device's resident vocabulary.

Programs that utilize the Speech Synthesizer can be written using Extended BASIC. The Terminal Emulator II cartridge allows you to select the specific sounds you will hear (allophones) and thus provides an unlimited vocabulary.

### **Impact Printer**

The TI Impact Printer (Fig. 2-18) is a fairly new offering for the TI-99/4A. The printer itself has been out for a long time, because it's manufactured by Epson (probably the best-known manufacturer of computer printers in the world). The Epson printer, the MX-80, is almost a standard in the personal computer industry. The IBM Personal Computer, for instance, uses this same printer (with IBM's name on it).

This printer is capable of producing 80 characters per second and can handle 40-, 66-, 80-, and 132-column widths. It can print text or graphic data. This is a bidirectional printer, which means it prints from left to right and then from right to left. There is no nonprinting return stroke. The first line of a page of text is

printed from left to right, just like on a typewriter. However, the second line will be printed from right to left in reverse order.

This printer does not require special paper. It features a 9 by 9 dot matrix print head that can be easily replaced. A single-unit ribbon cartridge is easily inserted, and you can choose from several different ribbon colors.

Connecting the impact printer to the TI-99/4A requires the RS-232 interface and the printer cable supplied by Texas Instruments.

It produces excellent quality hard copy printouts, but is not a letter-quality printer. Letter-quality printers are used in word processing operations that require all letters and documents to appear as if they were typed on a high-quality typewriter. Because most letter-quality printers have typewriter-like mechanisms to do the actual printing, they are usually slower than dot matrix printers.

The TI Impact Printer produces neat and perfectly readable copy. The type will not appear to be as perfect as that produced by a good letter-quality printer or for that matter, a good typewriter, but because the TI-99/4A is not designed for sophisticated word processing (in my opinion), a letter-quality printer should not be required.

### **Cartridge Storage Cabinet**

If you collect a lot of TI-99/4A software, you've got to store the cartridges or cassettes when they're not in use. Figure 2-19 shows the TI storage cabinet, which sells for about \$15.00. The cabinet holds 12 cartridges in a sliding drawer. The case is designed to be stackable, so two or more may be combined vertically to increase storage capability.

|         |            |           |          |               |          |
|---------|------------|-----------|----------|---------------|----------|
| +       | (positive) | but       | draw     | gives         | it       |
| -       | (negative) | buy       | drawing  | go            | j        |
| .       | (point)    | by        | e        | goes          | joystick |
| 0       |            | bye       | each     | going         | just     |
| 1       |            | c         | eight    | good          | k        |
| 2       |            | can       | eighty   | good work     | key      |
| 3       |            | cassette  | eleven   | goodbye       | keyboard |
| 4       |            | center    | else     | got           | know     |
| 5       |            | check     | end      | gray          | l        |
| 6       |            | choice    | ends     | green         | large    |
| 7       |            | clear     | enter    | guess         | larger   |
| 8       |            | color     | error    | h             | largest  |
| 9       |            | come      | exactly  | had           | last     |
| a(ä)    |            | comes     | eye      | hand          | learn    |
| a1 (ah) |            | comma     | f        | handheld unit | left     |
| about   |            | command   | fifteen  | has           | less     |
| after   |            | complete  | fifty    | have          | let      |
| again   |            | completed | figure   | head          | like     |
| all     |            | computer  | find     | hear          | likes    |
| am      |            | connected | fine     | hello         | line     |
| an      |            | console   | finish   | help          | load     |
| and     |            | correct   | finished | here          | long     |
| answer  |            | course    | first    | higher        | look     |
| any     |            | cyan      | fit      | hit           | looks    |
| are     |            | d         | five     | home          | lower    |
| as      |            | data      | for      | how           | m        |
| assume  |            | decide    | forty    | hundred       | made     |
| at      |            | device    | four     | hurry         | magenta  |
| b       |            | did       | fourteen | i             | make     |
| back    |            | different | fourth   | I win         | me       |
| base    |            | diskette  | from     | if            | mean     |
| be      |            | do        | front    | in            | memory   |
| between |            | does      | g        | inch          | message  |
| black   |            | doing     | games    | inches        | messages |
| blue    |            | done      | get      | instruction   | middle   |
| both    |            | double    | getting  | instructions  | might    |
| bottom  |            | down      | give     | is            | module   |

Fig. 2-17. The resident vocabulary in the T1 Speech Synthesizer.

|          |                |             |                   |               |         |
|----------|----------------|-------------|-------------------|---------------|---------|
| more     | point          | shape       | that is incorrect | up            | you     |
| most     | position       | shapes      | that is right     | upper         | you win |
| move     | positive       | shift       | the1 (thē)        | use           | your    |
| must     | press          | short       | the (thē)         | v             | z       |
| n        | print          | shorter     | their             | vary          | zero    |
| name     | printer        | should      | then              | very          |         |
| near     | problem        | side        | there             | w             |         |
| need     | problems       | sides       | these             | wait          |         |
| negative | program        | six         | they              | want          |         |
| next     | put            | sixty       | thing             | wants         |         |
| nice try | putting        | small       | things            | way           |         |
| nine     | q              | smaller     | think             | we            |         |
| ninety   | r              | smallest    | third             | weigh         |         |
| no       | randomly       | so          | thirteen          | weight        |         |
| not      | read (rēd)     | some        | thirty            | well          |         |
| now      | read1 (rēd)    | sorry       | this              | were          |         |
| number   | ready to start | space       | three             | what          |         |
| o        | recorder       | spaces      | threw             | what was that |         |
| of       | red            | spell       | through           | when          |         |
| off      | refer          | square      | time              | where         |         |
| oh       | remember       | start       | to                | which         |         |
| on       | return         | step        | together          | white         |         |
| one      | rewind         | stop        | tone              | who           |         |
| only     | right          | sum         | too               | why           |         |
| or       | round          | supposed    | top               | will          |         |
| order    | s              | supposed to | try               | with          |         |
| other    | said           | sure        | try again         | won           |         |
| out      | save           |             | turn              | word          |         |
| over     | say            | t           | twelve            | words         |         |
| p        | says           | take        | twenty            | work          |         |
| part     | screen         | teen        | two               | working       |         |
| partner  | second         | tell        | type              | write         |         |
| parts    | see            | ten         | u                 | x             |         |
| period   | sees           | texas       | uhoh              | y             |         |
| play     | set            | instruments | under             | yellow        |         |
| plays    | seven          | than        | understand        | yes           |         |
| please   | seventy        | that        | until             | yet           |         |

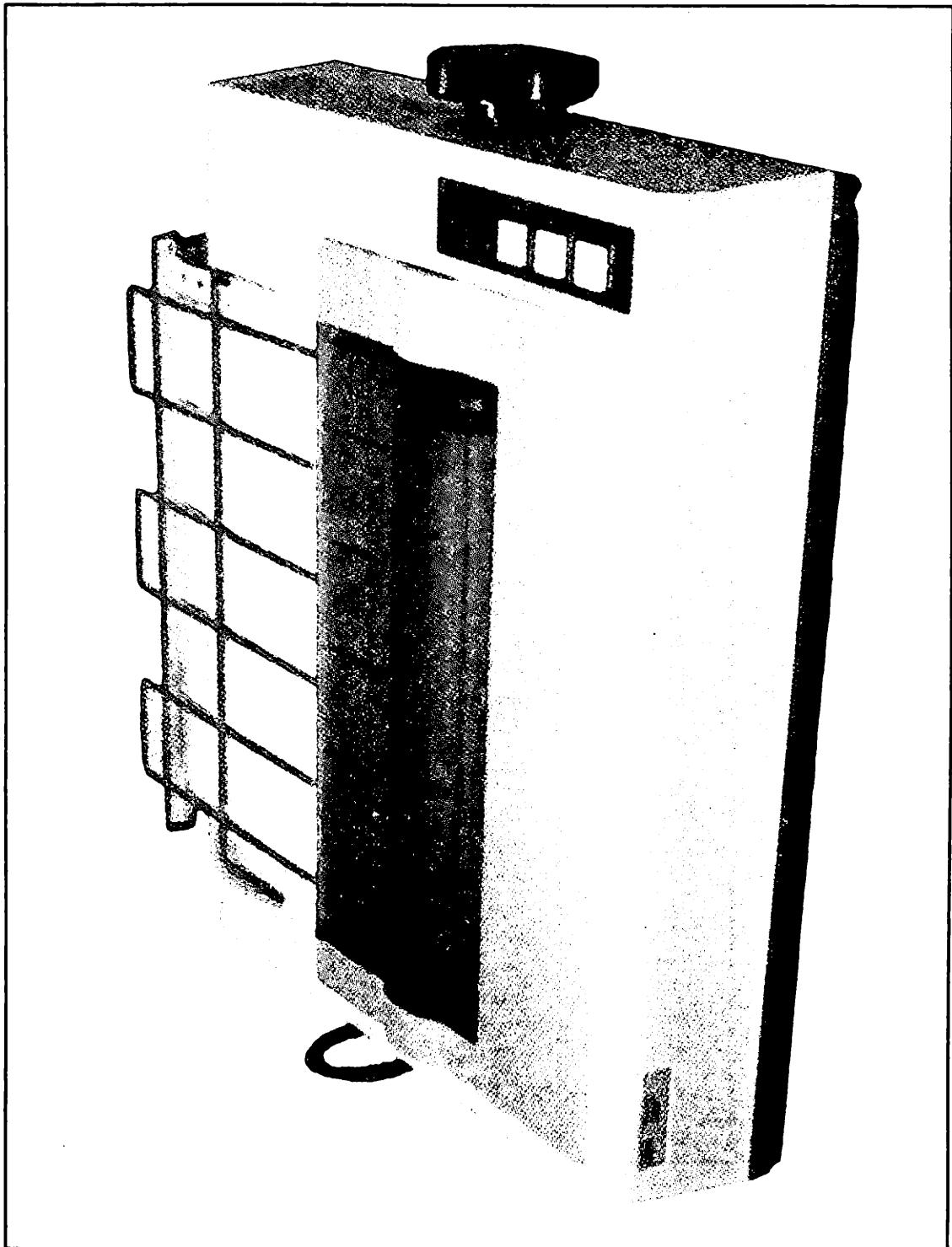


Fig. 2-18. The TI Impact Printer offers a speed of 80 characters per second and a maximum width of 132 columns.



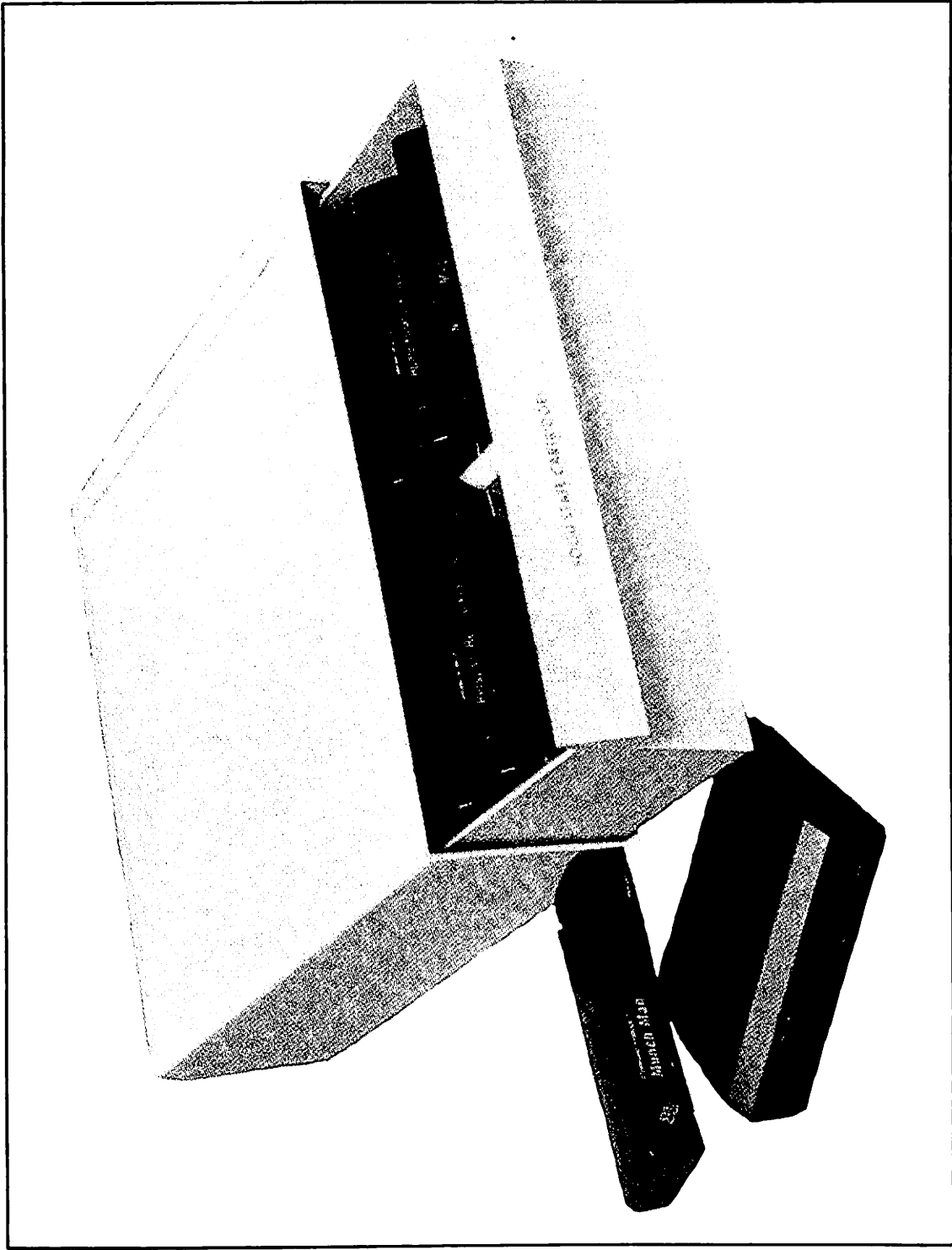


Fig. 2-19. A cartridge storage cabinet can protect valuable software.

### Compatible Computers

The next device may not be considered an option to the TI-99/4A, but it is available and it can be interfaced with this computer. Since this is a relatively new announcement from TI, it is appropriate that it be mentioned here.

**The Compact Computer 40 (CC-40)** was announced on January 6, 1983. It is the first member of a new series of small computers designed for professionals. Shown in Fig. 2-20, the computer is similar in appearance to the TI-99/2, but includes a numeric keypad and a

built-in LCD display. The CC-40 is programmable in Enhanced BASIC and can run preprogrammed applications software loaded from plug-in solid state cartridges or from small tape cartridges.

The system is battery-operated and fits unobtrusively on a desk or into a briefcase. It is designed to be used as a small personal desktop cordless computer and for data communications. Its small size and battery operation also provide extensive capability for portable computer applications.



Fig. 2-20. Compact Computer 40 (CC-40) is the first member of a new series of computers from TI which are small but designed for professionals (Courtesy Texas Instruments Inc.)

The computer console has a 34K byte ROM that contains a BASIC language interpreter allowing operation in BASIC. The BASIC language built into the CC-40 is compatible with TI BASIC. Calculator functions are available. The computer contains 6K bytes of RAM and can be expanded to 16K bytes. The CC-40 has a suggested retail price of \$249.95.

A plug-in module port is provided for application software of up to 128K bytes of ROM. This port can also be used to expand the random-access memory of the computer. The back of the console houses a Hex-bus intelligent peripheral interface connector, allowing connection of any Hex-bus compatible peripherals to this device, as well as future TI products.

Three low-cost peripherals will also be available: an RS-232 interface, a printer/plotter, and a Wafertape digital tape drive. Other peripherals such as a wand input device, modems, printers, and a black and white television monitor should be available late in 1983. Each peripheral includes a Hex-bus port and interface cable. Peripherals will also operate with the TI-99/2 and, with an adapter, will work with the TI-99/4A computer as well.

The RS-232 interface allows direct connection to serial-input printers and modems. With the addition of an optional cable, the interface can connect to a parallel-input printer. The RS-232 interface, HX-3000, has a suggested retail price of \$99.95.

The printer plotter is an x-y plotter with four-color capability using 2½ inch wide plain paper. In addition to x-y plotting, it can print up to 36 characters per line. The printer/plotter peripheral, HX-1000, has a suggested retail price of \$199.95.

The Wafertape digital tape drive can store up to 48K bytes and has a data transfer rate of 8,000 bits per second. The Wafertape unit, HX-2000, has a retail price of \$139.95.

Twenty-two software applications packages, including 8 plug-in Solid State Software cartridges and 14 Wafertape cartridges, are also available. The plug-in cartridges, which sell for prices ranging from \$39.95 to \$124.95, are: Mathematics, Finance, Perspective Drawing, Statistics, Business Graphics, Non-parametric Statistics, and Advanced Electrical Engineering (\$59.95 each); Editor/Assembler (\$124.95); and Games I and Games II (\$39.95 each). Wafertape cartridges, which have a suggested retail price of \$19.95 each, are: Elementary Dynamics, Regression/Curve Fitting, Pipe Design, Production and Planning, Inventory Control, Electrical Engineering, Thermodynamics, Photography, Solar Energy, Profitability Analysis, Quality Assurance: Sampling Plans, and Quality Assurance: Control Data. A total of 75 applications solutions cartridges (48 solid state and 27 Wafertape programs) are also available. TI is initiating aggressive third-party authorship programs as well as developing software internally.

The CC-40 console is 9½ inches by 5¼ inches by 1 inch and weighs 22 ounces. The display is a scrollable 31-character liquid crystal display (LCD) capable of displaying upper- and lowercase characters. In addition, there are 18 built-in indicators for user feedback including shift, control, function, degrees, radians, grads, and 6 user-settable flags.

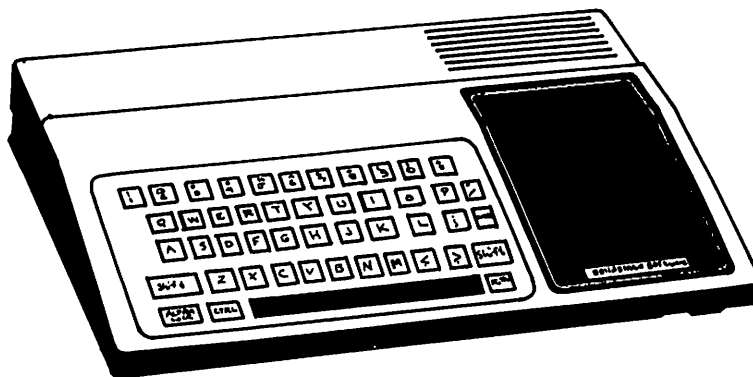
The keyboard has a staggered QWERTY key arrangement with a numeric keypad. Key spacing allows for easy key entry without making the unit excessively large. A tilt stand

is built into the back of the console to provide an optimum viewing and keying angle.

Four AA alkaline batteries provide power to the console for up to 200 hours. Memory contents are retained even when the unit is turned off. The unit may also be connected to a standard 115-volt ac power outlet using an optional adapter, AC9201, available for \$14.95.

Texas Instruments continues to offer new products and accessories, but usually attempts to provide methods of interfacing them with previous offerings. This speaks well of the company and assures that any product you purchase does not suddenly become antiquated by the introduction of a new product.

## Chapter 3



# TI-99/4A BASIC

The language used by most microcomputers is BASIC, an acronym for *B*eginners *A*ll Purpose *S*ymbolic *I*nstruction *C*ode. Unlike many computer languages, BASIC uses English words to represent computer commands. For example, the Print statement tells the computer to print something on the screen. The End statement tells the computer to stop execution, or end, a program. The BASIC commands, statements, and functions relate to the actual function that is to be carried out.

If you're already familiar with BASIC, you will have little trouble converting to TI BASIC. All dialects of BASIC are similar, although some contain special statements designed to perform a specialized function on a particular machine. These differences are always minor, and most of what you already know about BASIC will apply to the TI-99/4A.

This chapter overviews TI BASIC and explains what each command, statement, and function causes the machine to do. If you're familiar with BASIC, many of these pages will contain view material; otherwise, this chapter will serve as a BASIC primer for the TI-99/4A.

The nucleus of TI-99/4A BASIC is built into the machine. The BASIC interpreter is written into the on-board ROM contained in the console unit. (ROM stands for Read-Only Memory, as opposed to RAM, which is Random-Access Memory.) The programs contained in ROM are handled on the machine level: the integrated circuit chips that make up ROM have been electronically programmed at the factory. When the computer is turned on, it reads this information into its microprocessor. Nothing you can do at the keyboard affects the programming in ROM.

The programs you write are committed to RAM. RAM is also composed of integrated circuits, but you can change this programming based on your keyboard input. RAM is also known as read/write memory: you can write information into the memory and then the microprocessor reads information out. The language used to write information into RAM is the same one set up in ROM, supplying the language the computer understands.

This is really a language set, much like a dictionary, which contains all the words in the English language. In dictionary form the words are not connected to form meaningful sentences, and this is the way the words are organized in ROM. ROM simply tells you what words you can use. You must pull them out and arrange them in a meaningful order, which will then be committed to RAM.

Each TI BASIC statement, command, and function is explained below including what it means and how to use it in writing programs.

**ABS** The ABS function, for absolute value, gives the absolute value of an expression. This expression is often called the argument; it is the value obtained when the numeric expression is evaluated. If the argument is positive, the absolute value function gives you the argument itself. If the argument is negative, the absolute value is the negative of the argument (the absolute value of  $-20$  is  $20$ ). This function is useful when it is necessary to pull the absolute value from a long series of mathematical functions. It is used in the following format:

**ABS(38)**

The 38 in this case is the numeric expression.

It could be replaced by a variable or a complex series of mathematics, such as:

**ABS(20\*(14\*3.2)/-20)**

The absolute value will return the numerical value from this formula and delete the minus sign if the value is negative.

**ASC** The ASC function returns the ASCII code for the first character of a string variable or string of numbers inserted in parentheses following this function. Each character produced by the TI-99 is represented and accessed by an ASCII code number. For example, the ASCII code number for the uppercase letter O is 79. Using the ASC function followed by an O in parentheses would yield the number 79. A typical format for this function is:

```
10 X$ = "O"
20 PRINT ASC(X$)
```

When this simple program is run, the computer screen will display 79 (the ASCII value for X\$), which is equal to the uppercase letter O.

**ATN** This function is similar to **ABS**, except it returns the arctangent of the numeric expression which follows it in parentheses. The arctangent is the angle in radians whose tangent is equal to the numeric expression. (This sophisticated mathematical function will not be of immediate use to the beginning programmer.) ATN function formatting is handled in the same manner as the ABS function.

**Break** The Break command is entered via the keyboard: it is not normally included as part of a program. When the Break

command is entered, break points are set at the program lines listed in the line list (program listing). When you enter Break, you tell the computer to stop running the program before executing the statement on the next line.

**BYE** The BYE command lets you leave BASIC. When this command is entered, the computer closes all open files, the program in memory and all variables are erased, and the computer is reset so it's ready to receive programming when you return to BASIC. After the BYE command is entered and executed, the computer screen returns to the master mode, the first mode accessed when the computer is turned on. Don't execute this command until you are certain that any program currently in memory has been saved.

**CHR\$** The CHR\$ function is the reverse of the ASC function. Where the ASC function returned the ASCII code for a specific character, the CHR\$ function converts an ASCII code number into its character equivalent. The following will cause the computer screen to display the letter O:

```
10 V$ = CHR$(79)
20 PRINT V$
```

In the simple program shown here, V\$ equals CHR\$(79), which is the same as saying V\$ is equal to ASCII character 79 or the uppercase letter O. The Print statement in line 20 causes the character O to be printed on the screen.

**Close** The Close statement "closes" a file that was previously opened using an Open statement. Any open file must be closed before the computer can move to another part of

execution. The Close statement is discussed further under the Open statement entry.

**Call CHAR** CHAR is a subprogram standing for character definition. The Call statement is used to call up or initiate the subprogram. Call CHAR lets you arrange special graphics characters on the screen. It is followed by the ASCII character code and a pattern identifier expressed in hexadecimal code (a 16-character string expression which specifies the pattern of a character you want to use in your program). The graphics section of this book discusses this in more detail.

**Call Clear** The Call Clear subprogram clears or erases the monitor screen. A Call Clear command is often issued at the beginning of a program to clear the screen. Without this command new characters are added at the bottom of the screen, preceding lines above them continue to move upwards on the screen. When the screen is full the uppermost line scrolls off the top of the screen. With the Call Clear subprogram the screen clears immediately, thus decreasing screen congestion.

**Call Color** The Call Color subprogram lets you specify the colors of characters on the screen. This subprogram statement is followed by a character set number, foreground color code, and background color code, all numeric expressions.

**Call GCHAR** This subprogram lets you read a character anywhere on the screen, by specifying the row number, column number, and the numeric variable to read the character. The video screen is arranged in a series of blocks, 32 running horizontally and 24 running vertically. Row number 12 references

the middle far left of the screen, while column number 16 references the top center portion of the screen. When the two numbers are combined, as in 12,16 the center of the screen is referenced.

**Call HCHAR** This subprogram places a character anywhere on the screen and optionally repeats it horizontally. Input the row and column numbers, along with the character code (given in the ASCII equivalent) and, optionally, the number of repetitions.

**Call JOYST** This subprogram lets you input information directly to the computer by positioning the lever on a joystick. (Joysticks are available as options for the TI-99.)

**Call Key** The Call Key subprogram transfers one character from the keyboard directly to the program, eliminating the need for an Input statement. (The Call Key subprogram is similar to an INKEY\$ variable common to other dialects of BASIC.) This subprogram reads the keyboard input and branches the program according to the pressed key.

**Call Screen** This subprogram is used to display on-screen graphics and lets the screen be changed to any of 16 available colors. When a Call Screen subprogram is executed, only the screen background color changes. The Call Screen color code is a number from 1 to 16. To change a screen to a dark blue background, you would type **CALL SCREEN(5)** (5 is the color code for dark blue).

**Call Sound** The Call Sound subprogram generates tones and noises. Follow this statement by the time duration, frequency, and volume you wish the sound to follow. The duration is measured in milliseconds, numerically expressed by a value of from 1 to 4250. A value of 4250 holds the tone for 4.25 seconds.

Frequency is expressed in hertz, legal values are from 110 to 44,733. A chart in the Appendices indicates the frequencies that correspond to different musical notes. The final number in the string expresses volume, one of five values from 0 to 5. Zero is the loudest, 5 is the softest.

**Call VCHAR** This subprogram is like Call HCHAR, except it repeats characters on the screen vertically rather than horizontally.

To further demonstrate, take the following example:

```
CALL VCHAR(2,15,86,7)
```

This will cause 7 ASCII characters (86) to appear vertically on the screen starting at position 2,15. ASCII character 86 is the capital letter V, which is repeated 7 times. The HCHAR version is:

```
CALL HCHAR(2,15,72,7)
```

The ASCII code has been changed to 72, the letter H.

**Continue** This command is entered whenever program execution has been halted by a Break command. When a Continue command is input, execution continues until the program ends or another Break point is reached.

**COS** The COS function, for cosine, returns the cosine of a numeric expression. The format is **COS(X)**, where X is the numeric expression. If you entered the line **PRINT COS(4)**, the screen would display the cosine of the number 4. You can also use this function as follows:



```
10 I = COS(4)
20 PRINT I
```

**Data** The Data statement stores numeric and string constant data in a program. It is always used with a Read statement, which instructs the computer to pull information from the Data statement. The format for the Data statement is: Data item, item, item, . . . If you wanted to include the numbers 1 through 10 in a Data statement, they would have to be separated by commas: DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Whenever a Read statement is encountered, the information contained in the Data statement will be fed to the machine one item at a time. A Read statement would have to be accessed 10 times to read all Data items in the example.

**DEF** The define statement lets you define your own functions in a particular program. The specified function name may be any valid variable name. Any parameters following a DEF statement must be enclosed in parentheses.

**Delete** This command removes a program or a data file from a disk. To use this command, you must have the TI Disk Drive Controller and a disk drive. Once a file is established, the Delete command will erase it from the storage medium. The command must be followed by the file name or program name. If you opened a file under the name GAME, DELETE "GAME" will erase it from the disk.

**DIM** This may be used as a command or statement and reserves space for numeric and string arrays. DIM lets you set the maximum size of an array. For example DIM X(15) sets aside a one-dimensional array with

a maximum of 15 elements. Using the DIM statement, you may also establish two- and three-dimensional arrays.

**Display** The Display statement is identical to the Print statement. Both may be used to write information on the display screen. The Display statement causes information to be output only to the screen.

**Edit** The Edit command is entered in direct mode and used to call up a line from a previously written program to change it. For example, to make corrections in line 100, input EDIT 100, and that line will appear on the screen. The FCTN and cursor movement keys are used to align the cursor with the beginning of the word or letter to be changed. New information may now be typed over the old, or the Insert function may be used to place letters or words before this point. There is no need to retype the entire line. The line number cannot be changed. Press Enter to exit the edit mode and store all changes in memory.

**End** The End statement terminates your program. It may be used interchangeably with the Stop statement. Its presence as the last line of a program is not necessary, since the program will automatically terminate when there are no more lines to execute. The End command is useful when one or more subroutines are included at a program point which may follow the normal termination point. For example, if you write a program filling lines 100 through 1000 and then add a subroutine starting at line 1010 reached through a GOSUB or GOTO statement, the End statement might be inserted at line 1005 to avoid entering the subroutine at the end of the program.

**EOF** The end-of-file function deter-

mines if the end of a specific file has been reached. When files are accessed by the Open statement, their information is output until there is nothing left. On the next information loop, an end-of-file condition results. Using the EOF function, a branch may be built into a file-reading program which will terminate the program before an error message can occur or activate other programs. If a file has been opened as number 1, the EOF function might look like this:

```
IF EOF(1) THEN 1000
```

When an end-of-file condition results in file number 1, the program will branch to line 1000.

**EXP** This is the exponential function, the inverse of the natural logarithm function. It raises the number 2.718281828 to the  $X$  power. In this case, the variable  $X$  is the number you input. For example:

```
PRINT EXP(4)
```

will raise the number 2.718281828 to the fourth power.

**For-To-Step** This statement is used to create loops in a computer program. It is always used with a Next statement, which marks the end of a loop. While For and To must always be used to set up a For-Next loop, the Step command is necessary only when the loop is to cycle in increments other than 1. The following program demonstrates the use of this statement:

```
10 FOR X = 1 TO 10 STEP 1
20 PRINT X
30 NEXT X
```

This is a simple For-Next loop which causes the value of  $X$  to be printed on the screen. The For-To-Step statement in line 10 specifies that  $X$  is equal to a value of from 1 to 10 in steps of 1. In this Step 1 cycle,  $X$  is equal to 1 on the first cycle, 2 on the next, then 3, and so on, until the maximum value specified is reached. If the step were changed to 2, the count would skip every other number.

**GOSUB** The GOSUB statement is used to branch to another portion of a program. It may be typed as one word or two, as in GOSUB or GO SUB. This statement is always used with a Return statement allowing you to defer the program to a subroutine, execute each line in the subroutine, and then return to the next program line following the GOSUB statement.

**GOTO** The GOTO statement is similar to GOSUB, used to branch from one portion of a program to another. A line number follows this statement naming the program line to which to branch GOTO 100 or GO TO 100 will bring about a branch to line 100. Once a GOTO branch is made, there is no automatic return; the only way to return to the main program is with another GOTO statement.

**If-Then-Else** statement lets you change the sequence of program execution by using a conditional branch. GOSUB or GOTO will bring about an unconditional branch. With If-Then-Else, a certain condition must exist before the branch occurs. Else is often dropped. For example:

```
IF X = 40 THEN 500
```

This means there will be a branch to line 500

only when the value of  $X$  is equal to 40. If  $X$  is not equal to 40, the computer will execute the next line. When the Else statement is used, a branch will always occur, but the branch selected depends on a certain condition:

```
IF X = 40 THEN 500 ELSE 1000
```

There are two possible branches—one to line 500 and the other to line 1000. If the value of  $X$  is 40, there will be a branch to line 500; if  $X$  is not equal to 40, then there will be a branch to line 1000. If-Then-Else statements are often used to conditionally access subroutines using branches to lines containing GOSUB statements. The following program segment demonstrates this:

```
10 IF X = 1 THEN 20 ELSE 30
20 GOSUB 100
30 PRINT X
40 END
```

In line 10, the computer is told to branch to line 20 if the value of  $X$  is 1. Line 20 contains a GOSUB statement that branches to a subroutine starting at line 100. The content of this subroutine is unimportant for this discussion and is therefore not included here, but when it has been executed, there will be a Return statement that will cause line 30 to be the first executed after the subroutine.

If  $X$  is not equal to 1, the Else portion of line 10 branches to line 30, skipping line 20 altogether.

**Input** The Input statement temporarily halts program execution until information can be input via the keyboard. The Input prompt appears as a question mark on the

screen. The Input statement may be immediately followed by a prompt message in quotation marks. After the last quotation mark, a colon must be inserted and then a variable name. Either a numeric or string variable may be specified. If a numeric variable is used and the information is not input in numeric form, an error message will be displayed.

Another form of the Input statement lets you enter data from an accessory device. The Input statement can be used only with files open in Input or Update mode. The file number in the Input statement must be the file number of a currently open file.

**INT** The integer function gives you the largest integer not greater than the argument. The argument is the value obtained when a numeric expression is evaluated. With positive numbers, the decimal portion of the number is dropped. For negative numbers, the next smallest integer value is used. The following format is used with the INT function:

```
100 X = INT(113.876)
110 PRINT X
```

The value output to the screen will be 113, the integer of 113.876. If the value in line 100 was  $-113.876$ , the integer value would be  $-114$ , the next smallest integer value. INT is used whenever it is necessary to arrive at answers given as whole numbers only and not as whole numbers and fractions or decimal equivalents.

**LEN** The length function gives you the number of characters in a string:

```
10 A$ = "HELLO"
20 PRINT LEN(A$)
```

When this program is run, the screen will display the number 5. The number of characters in A\$, which is assigned the word HELLO. Hello has five letters; the string length therefore is 5. The LEN function counts spaces as well as characters; if A\$ were assigned the value of HELLO CATHY, the length would be 11.

**Let** The Let statement is optional and is used to assign values to variables within a program. In TI BASIC a variable may be assigned by using the equal sign. For instance, LET A = 10 is the equivalent of A = 10. Either method is acceptable.

**List** The List command is entered in direct mode (no line number) rather than as part of the program. It causes the screen to display the list of lines which make up a program. You may also specify the name of the device on which you want the lines listed. You may also specify a line or lines with the List statement; typing LIST displays all program lines. LIST 150 will display only line 150. LIST-150 will list all program lines to and including line 150. LIST 150- will list line 150 and all lines following it. LIST 90-150 will list lines 90 through and including line 150.

**LOG** This is the natural logarithm function. PRINT LOG(3.5) will give you the natural logarithm of the number 3.5. The number may also be represented by a previously assigned variable. LOG is the inverse of the EXP function.

**New** The New command erases the program currently in memory. It also closes any open files and clears all space previously allocated for special characters. The New command is often used after a program has been written, debugged, and stored on cas-

sette or disk. Typing NEW erases any program from memory and lets you begin on a new one. Don't use New before a program you wish to save has been committed to permanent storage.

**Next** The Next statement is never used by itself; it is always paired with a For statement (For-To-Step). The next statement controls whether the computer will repeat a loop or exit to the following program line. When a Next statement is encountered, the previously evaluated increment in the Step clause is added to the control variable, then tested to see if the control variable exceeds the previously evaluated limit.

**Number** This command may be entered as NUMBER or NUM. When the computer receives this command, it automatically generates line numbers to speed program writing. The NUM command is issued before a program is written. When you press Enter, a line number is automatically generated, starting with 100 and stepping up in increments of 10. When you have finished the program, hit Enter once more to remove the number generator feature.

**Old** The Old command reads a previously saved program into the computer's memory. This applies to programs which have been saved on cassette or disk and then removed from current memory. When you want to load information from cassette to current memory, input OLD CS1. A set of instructions will then appear on the screen telling you to rewind the cassette tape and press the Cassette Play button. If you are using a disk system, the Old command is followed by the name of the file you wish to load into memory.

**ON-GOSUB** The ON-GOSUB statement is used with a Return statement to tell the computer to perform one of several subroutines. It is another way of setting up a conditional branch to subroutines without using the If-Then-Else statement:

```
10 INPUT A
20 ON A GOSUB 150, 250, 350
```

This is not a conditional branch in the true sense, but it does bring about several branches whenever a value is input for A. The following is an example of a true conditional branch:

```
10 B = 10
20 INPUT A
30 ON B-1 GOSUB 1000
```

Here, a branch will occur only when A is equal to 9 (the value of B minus 1, as specified in the ON-GOSUB statement).

**ON-GOTO** This statement, like the ON-GOSUB statement, is used to access different portions of a program where Returns are unnecessary.

**Open** The Open statement prepares a BASIC program to use data files stored on cassette, disk, etc. It provides the necessary link between a file number in a program and the particular accessory device on which the file is to be located.

**Option Base** The Option Base statement is used to set the lower limits of an array subscript at 1 instead of 0.

**POS** The position function detects the occurrence of string-2 within string-1. The POS function compares two strings and indicates at what position a letter or series of

letters found in one string begins occurring in another.

**Print** The Print statement is used to display information on the screen. Words to be displayed follow the Print statement in quotation marks. When words and/or numbers assigned to variables or string variables are printed by following the Print statement with the name of the variable, no quotation marks are necessary.

**Randomize** The Randomize statement is used with the random number function (RND) to generate a pseudo-random sequence of numbers. When Randomize is used by itself, the random number function generates, a different sequence of random numbers each time. The Randomize statement may also be used with another number, called the seed. This sets the starting point for the random number generator. In this case, the sequence will be the same each time the program is run. The Randomize statement and RND function are used in programs simulating dice rolls, card selections, and other games of chance. The output numbers are pseudo-random, which means they make logical progressions to the computer, but the patterns are so complex as to appear random to users.

**Read** The Read statement is used to read data from Data statements within the same program. Read and Data statements must always be used together. Information is read an item at a time by the Read statement, sequentially from left to right.

**REM** The REM (remark) statement is a non-executable portion of a program. The computer simply skips over the lines that begin with REM. REM statements are used to insert information line by line in the program

that may be of importance to users.

**REM** statements are also helpful to the programmer. For example, when programs are quite long and complex, the beginning and ending of certain subroutines can be identified with **REM** statements, as can other major building block programs within a major program. When the program is reviewed for debugging, the **REM** statements let you quickly identify the sections you seek.

**Resequence** The **Resequence** command may also be entered as **RES**. It reassigns the line numbers for all lines in a program. It is often necessary during debugging to insert additional program lines, making the line number sequence confusing.

Also, when many additional lines must be input, you can often run out of space between lines. The **Resequence** command, when used alone will automatically renumber every line in sequences of 10, beginning with line number 100. All branch statements are also automatically changed to reflect the new numbers: if one branch statement was input as **GOTO 100** and line 100 was changed to line 120, during resequencing, the **GOTO** statement would read **GOTO 120** after resequencing. You can also renumber a program starting at certain lines and determine your own sequence: **RESEQUENCE 10,10** causes the first resequenced line number to be 10, followed by 20, 30, 40, etc.

**Restore** The **Restore** statement is used to return a **Data** statement to the beginning of its list of items.

**Return** The **Return** statement is used with a **GOSUB** statement to return program execution to the line immediately fol-

lowing the **GOSUB** statement which accessed the subroutine.

**RND** This is the random function, which provides the next pseudo-random number in the current sequence of numbers generated by the **Randomize** statement.

**Run** The **Run** command is used to begin execution of a program in memory. When used by itself, the **Run** command causes execution to begin at the first line number in the program. If the **Run** command is followed by a line number, execution will begin at that line.

**Save** The **Save** command lets you copy the current program in memory onto disk or cassette. The **Save** command must be followed by the name of the file you wish to establish.

**SEG\$** This is a function which gives you a portion of a designated string. The following program demonstrates the use of this function:

```
10 A$ = "PARADOXICAL"
20 PRINT SEG$(A$,4,5)
```

When this program is run, the screen will display **ADOXI**. This is the segment of the word "paradoxical" which has been assigned to **A\$** and begins with the fourth letter from the left and continues for five letters. **SEG\$** has been used to extract a substring from **A\$**.

**SGN** This is the signum function, giving you the algebraic sign of a value specified by an argument. This function tells you whether a number is positive, negative, or equal to 0:

```
10 A = 15
20 PRINT SGN(A)
```

Here a 1 will be displayed on the screen, indicating that the number is positive. If A were changed to -15 in line 10, the screen would display -1, indicating that the value of A is negative. If A were equal to 0, 0 would appear on the screen. Obviously, in the program shown for demonstration purposes, it is quite easy to tell whether a number is positive, negative, or equal to 0. The SGN function, however, may be used in a program that performs complex mathematical functions, most of which are not displayed on the screen.

Here, the SGN function may also be used to bring about branches to other portions of the program:

```
IF SGN(A) = -1 THEN 200
```

The value of the number is unimportant; the important quality is whether it is negative rather than positive or equal to 0.

**SIN** The sine function gives you the trigonometric sine of the argument. If the angle is in degrees, multiply the degrees by pi divided by 180 to get the equivalent angle in radians. The SIN function is useful when performing different types of vector math and in generating sine waves on a computer screen graph.

**SQR** The square root function returns the positive square root of the value specified by the argument:

```
10 PRINT SQR(9)
```

The output from this program will be the number 3, which is the square root of 9.

**Stop** The Stop statement terminates a program and is interchangeable with End.

**STR\$** This function converts the number specified by an argument into a string. It is the opposite of the VAL function.

**Tab** The Tab function is used with the Print statement and specifies a starting position on the line for the next print item. The Tab function works like a tab on a typewriter. **PRINT TAB(10); "HELLO"** will print HELLO on the screen starting 10 positions from the left.

**TAN** This returns the tangent of the argument X, where X is an angle in radians.

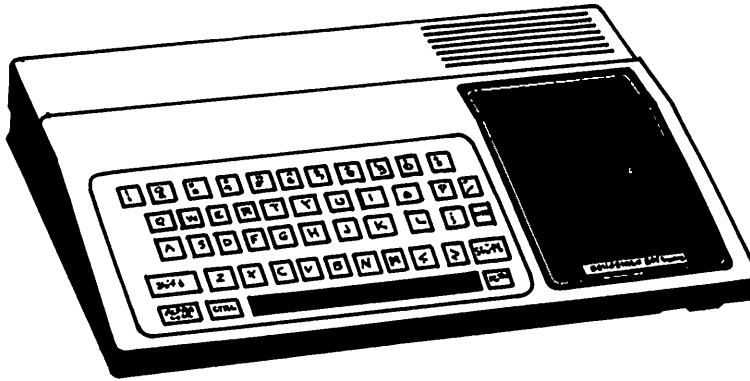
**Trace** This lets you see the order in which the lines of your program are executed. When the Trace command is input, the line numbers appear on the screen as they are executed. This can be a most valuable debugging aid, in that infinite loops and unwanted branches can be quickly detected. To remove the Trace feature, the Untrace command is input.

**VAL** The VAL function is used to extract a numeric value from a string variable. If the string variable is composed of numbers only, the VAL statement will extract these and assign them to a numeric variable when mathematical functions may take place. When VAL is used with a string variable containing letters and numbers, only the numeric portion will be committed to a numeric variable.





## Chapter 4



# BASIC Programming

If you've had some past computer experience and are simply making the transition from one machine to another, you may want to skip this chapter. If, on the other hand, the TI-99/4A is your first microcomputer, and you are a first-time user getting bogged down with all of the information covered so far, take heart. This chapter is specifically for you. The TI-99 is designed for first-time users as well as for those with considerable programming experience, and this chapter will help you decipher and understand the use of the TI BASIC commands, functions, and statements in Chapter 3 and show you how to begin writing programs step by step.

### YOUR FIRST PROGRAM

A computer program written in BASIC must consist of a minimum of one program line.

Some programs will have hundreds or even thousands of lines, but these programs are developed one line at a time. Complex programs are building blocks called subroutines; individual programs unto themselves. Subroutines usually contain only a few lines and are connected to other subroutines to bring about complex functions.

In BASIC, each line must have a number. Normally, lines are numbered in increments of 10, as in 10, 20, 30, 40, etc., or even 100, 110, 120, 130, etc. You can just as easily start numbering with 1 and increase in increments of 1, as in 1, 2, 3, 4, 5, etc. This is not normally done because few programs are written in finished form from the start. Usually you will want to go back through the program and insert additional lines in various locations. If you number in increments of 10, there is plenty of space to

insert additional lines. If your program contains 2 lines numbered 10 and 20 and you find it necessary to insert another line between them, you could number it 11 or 12 or 13, etc. Usually you would number it 15 so that you could, if necessary, add additional lines before and after it.

If you run out of spaces between line numbers, you can always renumber the entire program using the TI Resequence (RES) command, but we'll save this facet for later.

Your first program will consist of one line and will be used to display your name on the computer screen. Throughout this chapter, we will increase the size of this basic program by adding a few lines at a time.

To begin, turn on your computer. The computer screen will display the Texas Instruments logo, along with 16 color bars. You will be instructed to press any key to continue. When this is done, another instruction tells you to press key 1 to enter TI BASIC. After pressing this key, you will see the prompt **TI BASIC READY** at the bottom of the screen.

Another prompt will appear to tell you that the computer is ready to accept information. It looks like this: **>**. All information will appear on the screen to the right of this prompt.

For this programming exercise, make sure the Alpha Lock key (far left bottom of the keyboard) is in the down position. This will cause all letters to be printed in uppercase.

Begin by typing **10** on the screen. Hit the space bar and then type **PRINT "YOUR NAME"** inserting your own name between the quotation marks. This is the entire program. It should look like this:

## **10 PRINT "YOUR NAME"**

Examine the line carefully on your screen. Make sure it looks like the example above except with your name inserted between the quotation marks. If all appears to be in order, press the Enter key at the center right of the keyboard.

Congratulations! You have just written your first computer program and committed it to memory.

To see how the program works, you must "run" it. This is done by typing **RUN** on the keyboard. Now, press the Enter key again. Your name will appear at the bottom of the screen. **DONE** will also appear, indicating that the program is over.

If for some reason the program does not run correctly, you will get an error message indicating that the machine cannot run your program. Chances are you've made some minor error while entering the information. To check the line, type **EDIT 10**. This tells the machine that you wish to make some corrections to program line 10. When you press Enter, the line will again appear on the screen. Perhaps you failed to place a space between the number and the Print statement, or you may have failed to include one or both of the quotation marks. These are about the only mistakes that can be made in this simple program, assuming that you spelled **PRINT** properly. If there is an error, correct the program following the information in Chapter 2 on how to edit program lines. Then run the program again.

## **CLEARING THE SCREEN**

When you ran your program, you probably

noticed that while your name appeared at the bottom of the screen, all the previous screen information moved up a few inches. This is called *scrolling* and is helpful in allowing you to see the results of a short program run while viewing the program lines at the same time.

In some situations you will want to remove all information from the screen before the program run information is presented. To do this, you use the Call Clear statement. This simply replaces all information on the screen with spaces, effectively wiping the screen clean. To continue, type **CALL CLEAR**. For now, you do not need to put a number in front of this statement, as it is being used in direct mode to wipe the screen. Now, depress the Enter key, and all information will be erased from the screen. The computer memory still has your one-line program, so only the screen has been cleared. Now, type **LIST** and press Enter. Your one-line program should appear on the screen.

It's time to expand the program so that it clears all information from the screen before printing your name. To do this, the screen must be cleared before line 10 is executed. To do this you insert another program line prior to line 10. The actual line number may be any value from 1 to 9, so let's use 5. Type 5 then a space, and then **CALL CLEAR**. Examine the line and correct or edit it if necessary. Now press Enter.

It's time to run the program. Again, type **RUN**. Press Enter again. If your program is running properly, all previous information should be erased and your name will then appear at the bottom left side. Under it will be the word **DONE**, again indicating that the program has finished its run.

When you have successfully completed the program to this point, stop and think about the instructions you have given the computer. First you simply told it to print your name. Then you told it to clear all information from the screen and then print your name again.

## CONTINUOUS LOOPS

Now, let's tell the machine to print your name over and over again ad infinitum. This requires an additional program line. This one will use a **GOTO** statement to tell the computer what to do after the first two lines of the program have been executed. Type **CALL CLEAR** again (to clear the screen) and then type **LIST**. Hit Enter after each command is typed.

After typing **LIST** (and pressing Enter), your two-line program will again appear on the screen. The next line tells the computer what to do after the first two lines are executed, so this third line must follow line 10. This one will be numbered 20. Type 20 followed by a space, and then type in **GOTO 10**. Your program should look like this:

```
5  CALL CLEAR
10 PRINT "YOUR NAME"
20  GOTO 10
```

Examine this new three-line program. Here's how it works. Line 5 instructs the computer to clear the screen. Line 10 prints your name at the bottom of the screen. Line 20 tells the computer to go back and execute line 10 again after it is first run. The result is that line 10 will be executed for a second time, and line 20 is encountered once again, which branches (or goes back) to line 10. This process will go on

for as long as your computer is turned on. This is referred to as a *continuous loop*. Technically, the program goes on forever and never ends.

Type **RUN** and press Enter. You should see your name appear first at the bottom of the screen and then scroll upward as your name is printed again and again and again.

To stop the program, simultaneously press the FCTN key and the number 4 key. This will halt execution.

If the program does not work the way I have described here, re-list your lines and look for an obvious error. Make sure the lines on the computer screen correspond to those presented here.

Type **CALL CLEAR** again after you have halted the program run by pressing FCTN and 4. Type **LIST** and the three program lines will be displayed on the screen.

Let's change the contents of line 20 to bring about a different result when the program is run. Line 20 will still contain the GOTO statement, but this time, let's branch back to the beginning of the program, which is line 5. Using the Edit function, change 10 in line 20 to 5. You can forego the Edit function and simply retype line 20 completely. Type 20 followed by a space. Then type **GOTO 5**. Once you press Enter, your previous line 20 will be replaced by this new line. Type **RUN** and press Enter. You will now see your name printed only once at the bottom of the screen, but if you look closely, you will see that as soon as it's typed, it's erased and your name is written again. This occurs continuously as before, but in this program, the new printing of your name occurs in the same position each time.

This is still a continuous loop so you will have to manually halt it by simultaneously

pressing FCTN and 4.

If you can understand the concepts behind these two program versions, you are almost ready to begin writing some good programs on your own. You have learned a little about the Print command, erasing a screen, the GOTO statement, and returning computer execution to an earlier part of the program.

The last program example above could be duplicated by simply typing the first two lines over and over again, but the GOTO statement brings about the same results and requires only one additional line, for a total of three program lines.

## ERASING THE PROGRAM

Now, erase your program from memory. The best way to do this is to manually halt the program run and then type **NEW**. This erases the program forever and allows you to insert a new program. All memory storage space is cleared and the machine is ready to accept new program information. As soon as you press Enter after typing **NEW**, this command is executed.

## FOR-NEXT LOOPS

Don't let For-Next loops scare you; they are simple and one of BASICs most useful programming aids.

The following program is used to print your name five times on the computer screen:

```
10 CALL CLEAR
20 FOR X = 1 TO 5
30 PRINT "YOUR NAME"
40 NEXT X
```

Line 10 is used to clear the screen.

Line 20 begins a new area of study. In TI BASIC, the For statement is used to begin what is known as a loop. A *loop* is a part of the program that is executed for a specified number of times. The loop begins with For and ends with Next. The For statement is coupled with a variable, which in this case is the letter X. It could also be any other letter in the alphabet or combinations of letters. Line 20, specifies that X will be equal to a value of from 1 to 5.

Line 40 acts just like the GOTO statement and branches back to the For statement in line 20. Each time the branch occurs, the loop is said to have cycled one time.

When line 20 is executed, X will take on its low value of 1. X will continue to be equal to 1 until the Next statement is encountered in line 40. This branches back to line 20, and the loop cycles for the second time. Now, X is equal to 2. During the next cycle, X will be equal to 3, then 4, and finally, 5. The number 5 is the top value of the loop, and when the Next statement is encountered, the program ends. If you had additional program lines following line 40, these would be executed after the loop had cycled 5 times.

A For-Next loop acts like the continuous loop established by the GOTO statement; however, the For-Next loop establishes a beginning and ending point for the loop.

Let's go back to line 30 which is a part of the For-Next loop. Each time the loop cycles, the Print statement in line 30 will be executed. In this case, the Print statement will be executed 5 times before the loop times out.

Enter this program as shown above. Then type RUN and press Enter. Line 10 will clear the screen, and then lines 20 through 40 will

print your name 5 times. As before, your name will scroll up the screen, but unlike before the scrolling will stop after your name has been printed 5 times.

Remember the program using the GOTO statement to print your name at the bottom of the screen, erase it, and then write it again in the same location. You can duplicate the program using For-Next loops:

```
10  FOR X = 1 TO 5
25  CALL CLEAR
30  PRINT "YOUR NAME"
40  NEXT X
```

If you haven't already erased your four-line For-Next loop program you could upgrade it simply by typing 25 CALL CLEAR after line 20. This inserts a Call Clear statement within the For-Next loop. Each time the loop cycles, the screen will be cleared, your name will be printed, and the loop will repeat itself. Your name will be written at the bottom of the screen, erased, and then written in the same location again. This will occur 5 times.

The advantage of using the For-Next command is that the program run does not have to be manually halted. The halt is built in by the maximum number in the For-Next loop. If you want to print and erase your name 10 times, change 5 in line 10 to a 10. You control the loop cycles by specifying any number in line 10.

The next program makes use of the For-Next loop in a similar manner, but this time the screen will display the loop count rather than your name. Picking up on the program currently in memory, erase line 25 by typing 25 and pressing Enter. Modify line 30 by typing

30 followed by a space. Then type **PRINT X** and press Enter. Type **LIST** to see your modified program, which should look like this:

```
10 CALL CLEAR
20 FOR X = 1 TO 5
30 PRINT X
40 NEXT X
```

This program instructs the computer to print the value of **X**. The value of **X** is established in line 20 and will be equal to from 1 to 5. Now run the program (Type **RUN** and then press Enter.) Your screen should display:

```
1
2
3
4
5
```

When execution has been completed, the **DONE** will appear at the bottom of the screen.

This program provides a visual count for the For-Next loop.

### **MORE USES OF THE PRINT STATEMENT**

You may wish to see these numbers printed horizontally rather than vertically on the screen. This is easy to accomplish and requires that you modify line 30 as follows:

```
30 PRINT X;
```

The semicolon immediately following the **X** (no space) tells the computer to print **X** in a horizontal format. Now, when the program is run, the bottom of the screen will display:

```
1 2 3 4 5
```

You can use this trick with any of the earlier programs. The material in quotation marks should be immediately followed by a semicolon after the last quotation mark.

Without the semicolon, the computer automatically displays the information vertically. This is sometimes referred to as the default print state, which means this is the way the computer is set up to display information (vertically) unless instructed to do otherwise.

Let's make another modification to display the loop count and your name. Again, this modification occurs only in line 30. Type **30 PRINT X; "YOUR NAME"** substituting your own name between the quotation marks. Note that the semicolon has been dropped from the end of line 30. When you run this program, the screen should display:

```
1 YOUR NAME
2 YOUR NAME
3 YOUR NAME
4 YOUR NAME
5 YOUR NAME
```

The number appears first because it is the first thing encountered after the Print statement in line 30. Each number is represented by the changing value of **X**. If you change line 30 to **30 PRINT "YOUR NAME"; X** the screen will display:

```
YOUR NAME 1
YOUR NAME 2
YOUR NAME 3
YOUR NAME 4
YOUR NAME 5
```

If you add a semicolon at the end of either of these print lines, the information will be displayed horizontally on the screen.

## IF-THEN STATEMENTS

It's time to move on to other statements in TI BASIC. In the next set the computer tests for a certain condition and if this condition occurs, it performs another function.

In this example, we'll use an If-Then statement. All this statement does is say "if such and such is equal to so and so, *then* do this and that." Type the following:

```
10 CALL CLEAR
20 FOR X = 1 TO 5
30 IF X = 4 THEN 50
40 PRINT X
50 NEXT X
```

This program is similar to one used earlier to print the values of X, but here line 30 contains the If-Then test and branch instructions. What line 30 is saying is "if X is equal to 4, *then* branch to line 50." When X is equal to 4, line 40 is skipped and line 50 is executed. Line 50 signals the end of one cycle and branches back to line 30 for the beginning of another. The line that is skipped when X is equal to 4 tells the computer to display the value of X. When X is equal to 4, line 40 is skipped and this value is not printed on the screen.

Run the program and you should get a screen display which looks like this:

```
1
2
3
5
```

Whenever X is not equal to 4, line 30 does nothing. The branch to line 50 (specified following the Then statement) does not occur unless X is equal to 4. Line 40 is branched around in this latter condition, and line 50 is executed, causing the loop to recycle.

## IN AND OUT OF A LOOP

Let's find out how to temporarily leave a loop and then reenter it again. Change line 30 to 30 IF X = 4 THEN 60 and then type:

```
60 PRINT "YOUR NAME"
70 GOTO 50
```

The entire program (when listed) should look like this:

```
10 CALL CLEAR
20 FOR X = 1 TO 5
30 IF X = 4 THEN 60
40 PRINT X
50 NEXT X
60 PRINT "YOUR NAME"
70 GOTO 50
```

Line 50 ends the For-Next loop. Lines 60 and 70 lie outside of this loop. When the program is run, the loop will cycle 3 times and print the value of X each time. On the fourth cycle, line 30 detects the value of X as 4 and then branches outside of the loop to line 60, which instructs the computer to print your name. Line 70 uses a GOTO statement to branch back to line 50, causing the loop to be reentered. The loop is still in its fourth cycle. When you run this program, this is what you'll get:

```
1
2
```

```

3
YOUR NAME
5
YOUR NAME
FOR-NEXT ERROR

```

I've set you up. This is not a correct program. The purpose of this program was to cause the computer to count from 1 to 3, print your name, print 5, and then end. It did all of this, but then your name was printed again, followed by an error message telling you that there's something wrong with your program. In this case, it's a For-Next error.

When the program is first run, everything is fine. As instructed, the computer clears the screen and enters the loop that counts from 1 to 5. The numbers 1, 2, and 3 are printed on the screen. On the fourth cycle, X is equal to 4. Line 30 determines this and branches outside the loop to line 60. Lines 60 and 70 are outside the loop because they follow the Next statement in line 50. Everything is okay up to this point.

When line 60 is encountered, your name is printed and line 70 then branches back to the loop (GOTO 50). The loop must count from 1 to 5. When the branch to the outside occurs, the loop is in its fourth cycle and has one more to go. When line 70 branches back to the loop, the Next statement in line 50 causes it to cycle for the fifth and final time. The number 5 is printed on the screen below your name. The loop is ended... and this is where the problem occurs.

The loop ends in line 50 (after 5 cycles), and the computer goes on to the next line, which is line 60. This line instructs the computer to print your name, and it does so.

The real problem lies in line 70. It instructs the computer to GOTO line 50, which is the ending point of the For-Next loop. This was fine the first time it occurred, because the loop had not finished. But the second time line 70 is encountered, the loop was done and the branch statement to line 50 makes the computer think that a Next statement has been encountered without a matching For statement. The computer sees this new branch as a loop ending statement when no loop was ever begun. This is the reason for the error message.

We can correct this situation by adding a single line. This is placed following the loop and uses the End statement. This statement tells the computer the program is over. Your new program might look like this:

```

10 CALL CLEAR
20 FOR X = 1 TO 5
30 IF X = 4 THEN 60
40 PRINT X
50 NEXT X
55 END
60 PRINT "YOUR NAME"
70 GOTO 50

```

The End statement makes all the difference in the world. When this program is run, the screen will display:

```

1
2
3
YOUR NAME
5
"DONE"

```



A successful run, and no error messages! The reason is the End statement. All of the events described before have occurred again in this program, with one exception. When the loop completes its fifth cycle and times out, the next line encountered is 55, not 60. Line 55 contains the End statement, so the program is halted. Lines 60 and 70 are not executed again. These can be accessed only by the branch in line 30. The accidental running of branched to subroutines is a major problem for beginning computer programmers.

For-Next loops are not complete entries unto themselves. While the loop is cycling, no lines outside of it are executed (unless branches are inserted), but after the loop has timed out, all remaining program lines are executed. The End statement in line 55 stops execution on the spot. It is not encountered when X is equal to 4, and there is a branch outside of the loop because the Then statement branches *past* line 55 to line 60.

The following is a line-by-line breakdown for this completed program, discussing what each does in the run.

|                      |  |
|----------------------|--|
| 10 CALL CLEAR        | Clears the screen  |
| 20 FOR X = 1 TO 5    | starts the loop and sets minimum and maximum values for X      |
| 30 IF X = 4 THEN 60  | Tests for the value of X being equal to 4—branches to 60       |
| 40 PRINT X           | Displays the value of X on the screen                          |
| 50 NEXT X            | Ending point for the loop—branches to 20 during first 4 cycles |
| 55 END               | Ends program execution   |
| 60 PRINT "YOUR NAME" | Displays your name on the screen when branched to              |
| 70 GOTO 50           | Branches to line 50  |

This is the first program in this chapter that

does not execute *all* of its lines in the order in which they are presented. Lines 60 and 70 are executed prior to the execution of line 55. This is how branches are used in computer programs. They access other lines based upon certain parameters.

If you don't understand the uses of these statements, then re-read the chapter to this point and experiment. Moving forward at this point without a full understanding of all that has gone before will delay you in your goal of successfully programming the TI-99 computer.

## INPUT STATEMENTS

The next statement that we will use—the Input statement—can temporarily halt a program or halt a program long enough for you to enter information via the keyboard. The Input statement signals the computer that you wish to input (enter) information at a certain point in program execution. When the Input statement is encountered, program execution stops until you input the information and press Enter. The program will then continue to execute the remaining lines. An example of the Input statement follows:

```

10 CALL CLEAR
20 FOR X = 1 TO 5
30 INPUT A$
40 PRINT X
50 NEXT X

```

This program is a modification of the one I have been using. The original program demonstrated the operation of a For-Next loop. The modification is found in line 30, where INPUT A\$ is encountered. The Input statement must always be followed by a space and a variable.

Without it, an error message would occur. In this case, the variable is A\$, but it could be any other letter or combination of letters followed by the dollar sign (\$), which specifies a string variable. (A string variable contains letters and/or numbers that will not be used in mathematical operations.) Line 30 stops program execution and the computer waits for you to input a value that will be assigned to the variable A\$.

Here's how the program run operates. Line 10 clears the screen. The loop is entered in line 20. The Input statement is contained within the loop (line 30) and before the Print X statement. As soon as the loop is entered and before the value of X is printed, execution is halted.

To indicate that the computer is waiting for input, a question mark (?) will appear on the screen. What do you do now? Press Enter and the first value of X will appear on the screen (Print statement in line 40). Line 50 is then executed and recycles the loop. The second time around, the Input statement in line 30 is once again encountered, and you have to press Enter to cause the screen to display the second value of X. This process continues throughout the remaining cycles of the loop. Your screen should display:

```
?  
1  
?  
2  
?  
3  
?  
4  
?  
5
```

Each question mark indicates the point where the program is halted and where you have to press Enter to restart execution.

You can probably see the advantage of the Input statement used as a temporary program halt. Suppose the information printed by the loop was long and complex. You might not want it all displayed at one time, especially if the loop maximum value was 30 or more. Here, the displayed information would scroll off the screen, since the computer can only display 24 rows at one time. If your For-Next loop cycled 25 times, the first bit of printed information would scroll off the screen at the top. With the addition of the Input statement, you can control the rate at which the information is displayed to give you time to jot down these figures if necessary.

Sometimes the question mark prompt can be a bit confusing, especially if other printed information on the screen contains question marks. Fortunately, we can use the Input statement in a manner similar to the Print statement. To do this, change line 30 to **30 INPUT "PRESS ENTER TO CONTINUE":A\$**

With this change, whenever the Input statement is encountered, the screen will display the message **PRESS ENTER TO CONTINUE**. It is essential that a colon follow the last quotation mark. In turn, it is followed by the variable. When you run this program, your screen should display:

```
PRESS ENTER TO CONTINUE  
1  
PRESS ENTER TO CONTINUE  
2  
PRESS ENTER TO CONTINUE  
3
```

PRESS ENTER TO CONTINUE

4

PRESS ENTER TO CONTINUE

5

## LET STATEMENTS

Let's discuss another way the Input statement lets you type in information that can be directly used in a program. The program is:

```
10 CALL CLEAR
20 FOR X = 1 TO 5
30 INPUT "TYPE IN ANY NUMBER":A
40 LET Y = X + A
50 PRINT Y
60 NEXT X
```

This program contains the Let statement. You can use the Let statement, but it isn't necessary. Line 40 would run just as well if you typed  $Y = X + A$ .

What you're doing is asking the computer to Let the value of Y be equal to the value of X plus the value of A. This program lets you perform simple addition.

Here's how it works. Line 10 clears the screen. Line 20 starts the For-Next loop, which counts from 1 to 5. Line 30 uses the Input statement to temporarily halt execution and print the prompt **TYPE IN ANY NUMBER** on the screen. The variable A follows the Input statement and is assigned the number you enter. The Let statement in line 40 adds the values of X (loop value) and A (input value) and assigns this quantity to the variable Y. Line 50 prints the value of Y on the screen. As before, line 60 recycles the loop.

For demonstration purposes enter a value of 2 every time the prompt appears on the screen. Type 2 and then press Enter. Here's how your program run should look:

```
TYPE IN ANY NUMBER  2
3
TYPE IN ANY NUMBER  2
4
TYPE IN ANY NUMBER  2
5
TYPE IN ANY NUMBER  2
6
TYPE IN ANY NUMBER  2
7
"DONE"
```

By entering 2 each time, this number was added to the value of X in the For-Next loop. During the first cycle, X is equal to 1, and your input value is 2. During this first cycle, line 40 states that Y will be equal to X plus A, or Y is equal to 1 plus 2. The Print Y statement in line 50 prints the sum on the screen, which is 3. This same process continues throughout the next four cycles of the loop, with 2 being added to the value of X (providing 2 is the number typed in) each time.

In this case, the Input statement is used with a numeric variable (A), not a string variable. Therefore, you must type in a number before pressing Enter or the screen will display an error message.

The following program uses the Let statement in a very simple way:

```
10 LET A = 10
20 PRINT A
```

When you run this program, the screen will

display the number 10. However, the following program uses two Let statements to provide a different function.

```
10 LET A = 10
20 LET B = A/2
30 PRINT B
```

We're getting into some math now. As before, line 10 assigns the value of 10 to A. The second Let statement in line 20 performs division. It's saying "Let variable B be equal to the value of A divided by 2." Line 30 causes the value of B to be displayed on the screen. When the program is run, the computer will print 5, the value of 10 divided by 2. Notice, however, that line 20 uses the variable A. It does not read "let B = 10/2" but rather "let B = A/2."

The fact that we can divide an assigned variable is important, as shown by the following program:

```
10 INPUT "TYPE IN ANY NUM-
    BER":A
20 LET B = A/2
30 PRINT B
40 GOTO 10
```

Here is a "divide-by-two" program that lets you enter a number to be divided by 2. The Input statement, in line 10 asks for your part in this program. Any number you enter will be divided by 2. The answer will be displayed on the screen. Here's a sample program run:

```
TYPE IN ANY NUMBER    14
7
TYPE IN ANY NUMBER
```

This program is on an endless loop because of the GOTO statement in line 40. Once one problem has been worked, there is a branch to line 10, asking you to input another number. This process continues until you manually halt execution.

This program has many shortcomings, the main one being that the divisor is fixed at 2. This can be corrected by the following program:

```
10 INPUT "TYPE IN THE NUMBER
    TO BE DIVIDED":A
20 INPUT "TYPE IN THE DIVI-
    SOR":B
30 LET C = A/B
40 PRINT C
50 GOTO 10
```

This program uses two Input statements to let you type in a number to be divided and a number to be used as the divisor. The first is assigned to the variable A, and the divisor is assigned to B. Line 30 uses the Let statement to assign the value of A divided by B to variable C. Line 40 prints the answer on the screen, while line 50 branches back to the start of the program, allowing another problem to be input. Here's a sample program run:

```
TYPE IN THE NUMBER TO BE DI-
VIDED    100
TYPE IN THE DIVISOR    5
20
TYPE IN THE NUMBER TO BE DI-
VIDED
```

Let's improve this program by changing line 40

to 40 PRINT "THE CORRECT ANSWER IS";C Now the answer will be displayed as THE CORRECT ANSWER IS, given the same input values as before. To allow for a little more improvement, change line 50 to GOTO 5 and then insert the line 5 CALL CLEAR. Insert another line: 45 INPUT "PRESS ENTER TO CONTINUE":XY\$

Your completed program should look like this:

```
5 CALL CLEAR
10 INPUT "TYPE IN THE NUMBER
    TO BE DIVIDED":A
20 INPUT "TYPE IN THE DIVI-
    SOR":B
30 LET C = A/B
40 PRINT "THE CORRECT AN-
    SWER IS";C
45 INPUT "PRESS ENTER TO
    CONTINUE":XY$
50 GOTO 5
```

This is basically the same program as before, but it erases all information from the screen before a new problem is started. This is handled by the Call Clear statement in line 5. The GOTO statement in line 50 is changed to access this line. The Input statement in line 45 halts execution after one problem is worked. You continue by pressing Enter. This causes execution to pick up at line 50, which branches to line 5, clears the screen, and allows for the entering of a new problem. Without line 45, as soon as the value of C is printed in line 40, line 50 would cause the screen to be cleared before the answer could be read due to its branch to a line which contains a Call Clear statement. A sample program run follows:

```
TYPE IN THE NUMBER TO BE DI-
VIDED    100
TYPE IN THE DIVISOR    4
THE CORRECT ANSWER IS 25
PRESS ENTER TO CONTINUE
```

As soon as you press Enter again, the screen will be wiped clear and the same prompt will appear as before.

## VARIABLES

A variable is a representation of something else. The something else might be a number, a word, or a combination of numbers, letters, and characters. The term variable means that the letters used to represent quantities can be assigned at the beginning of a program and even changed in any portion of it. For example, if the beginning of a program starts with LET A = 10, then the letter A may be used in place of the number 10 throughout the program. However, the variable A may be reassigned within the program. If, at a later point, you input the line LET A = 20, then 20 will be substituted for A throughout the remainder of the program.

There are two types of variables to be concerned with. These are known as numeric variables and string variables. A *numeric variable* represents only numbers. In LET A = 10, A is a numeric variable. If we wanted A to represent a word, we would change it to a string variable, such as LET A\$ = "HELLO". The dollar sign is placed after the letter to indicate that it is a string variable and also that the value of the string variable (HELLO) is enclosed in quotation marks.

The quotation marks and the dollar sign

are mandatory. `LET A$ = HELLO` will result in an error message.

You cannot use a numeric variable to represent a string value. `LET A = HELLO` or `LET A = "HELLO"` will not work. A string variable can represent a number, as in `LET A$ = "1234"`. In this case, `A$` is equal to 1234, but you cannot use these numbers for mathematical functions.

Let's use two variables, one numeric and one string, as in:

```
10 LET A = 1234
20 LET A$ = "1234"
```

Now add `30 LET B = A/2`. The new variable is equal to `A` divided by 2. The computer will perform the mathematical function of dividing the value 1234 by 2. `B` will then be assigned the value of 617. `A$` is also assigned the value of 1234, but this is a string variable and cannot be used for mathematical functions. Therefore, `30 LET B = A$/2` or `30 LET B$ = A$/2` will not work.

If you want to perform mathematical functions, use numeric variables. Remember that `Input A$` allows you to halt your program until Enter is pressed, but `Input A` requires that a numeric value be typed in before pressing Enter.

I mentioned earlier that when using computers, there's always more than one way to skin a cat. The methods used to print your name in the following programs will be a bit different than the ways used earlier in this chapter. While the screen result will be the same, you will get a further education in the use of string variables.

## MORE ON STRINGS

A string variable can be used to represent letters, words, numbers, and combinations of these. The following program uses a string variable to which your name is assigned.

```
10 CALL CLEAR
20 LET A$ = "YOUR NAME"
30 PRINT A$
```

Following the `Call Clear` statement in line 10, a `Let` statement assigns `A$` the value of your name. In this case, the value is the letters that spell your name. These must be enclosed in quotation marks. The `Print` statement, is used in line 30. It prints whatever `A$` is equal to on the screen. When you run the program, your name should appear at the bottom of the screen. If not, you probably left out a quotation mark or made some other type of error.

Let's use the same basic program to print your name over and over again. The program is:

```
10 CALL CLEAR
20 LET A$ = "YOUR NAME"
30 PRINT A$
40 GOTO 30
```

The `GOTO` statement in line 40 makes this program a continuous loop by constantly branching to line 30. Your name will be printed over and over again until the program is manually halted by simultaneously pressing the `FCTN` and the 4 key.

The following program prints your name on the screen 5 times. It uses a `For-Next` loop:

```

10 CALL CLEAR
20 LET A$ = "YOUR NAME"
30 FOR X = 1 TO 5
40 PRINT A$
50 NEXT X

```

Here, the Print A\$ statement is enclosed within the loop. Since the loop counts from 1 to 5, the Print statement will be encountered 5 times, and your name will appear in a vertical format 5 times. By changing line 40 to **PRINT A\$**; your name will appear 5 times in the horizontal format. You can modify this program to print your name, erase it, and then print it again in the same screen location with the following changes:

```

20 LET A$ = "YOUR NAME"
30 FOR X = 1 TO 5
35 CALL CLEAR
40 PRINT A$
50 NEXT X

```

A Call Clear command in line 35 erases the screen each time a new loop cycle is begun. Your name will be written 5 times, but it will be erased immediately after it's written (except for the last time).

You may be wondering why the assignment of A\$ is done outside of the loop. In the two previous programs, A\$ is assigned a value in line 20, and the loop begins in line 30. It is all right to make the assignment within the loop, as in:

```

10 CALL CLEAR
20 FOR X = 1 TO 5
30 LET A$ = "YOUR NAME"

```

```

40 PRINT A$
50 NEXT X

```

In this situation A\$ is reassigned each time the loop cycles. It is always reassigned with the same value, so the end result displayed on the screen does not change. This is not considered to be an efficient programming step. The reason has nothing to do with the assignment itself, but rather with computer speed. This particular program will not slow the computer to a point where it is noticeable on the screen but, remember, the computer is not an instantaneous display device. It executes 30 program lines in half the time it takes to execute 60 program lines. This assumes that the first 30 lines are identical to the last 30, as certain statements and functions may take longer to process than others.

A loop containing only one Print statement executes faster than a loop containing two or more. This is because each time a statement or function is encountered within a loop, the computer's microprocessor must analyze it and decide what to do.

In this program, each time line 30 is encountered, the computer must reassign the value of A\$. This takes a small amount of time. Since it is not necessary to reassign A\$, placing this assignment line within the loop is slowing the program run. Of course you won't notice the delay because this is only one statement. However, some For-Next loops may contain many statement lines, and this can significantly slow the loop cycles. This is because each item within the loop must be read and identified by the computer. Additionally, the computer must decide whether or not a

particular line is to bring about branches or warrants the displaying of information on the screen (IF-THEN). The best rule to follow is to delete unnecessary items from your loops.

Here's another program that will allow you to enter a word, number, or combinations of letters and numbers and have it repeated 5 times on the screen.

```
10 CALL CLEAR
20 INPUT "TYPE IN THE PHRASE
   TO BE REPEATED":A$
25 CALL CLEAR
30 FOR X = 1 TO 5

40 PRINT A$

50 NEXT X
```

When it is first run, the screen will clear and the prompt **TYPE IN THE PHRASE TO BE REPEATED** will appear at the bottom of the screen. As soon as you've typed in what you want repeated, you press Enter. Line 25 will clear the screen again (in order to remove the prompt), and whatever you typed will be repeated 5 times on the screen. The assignment of the value of A\$ can change with each program run. Whatever you type in is assigned to A\$. If you enter **HELLO** this is the same as saying **LET A\$ = "HELLO"**.

## **RULES ON THE USE OF VARIABLES**

Caution: You cannot use all of the characters on the keyboard as part of a string variable name (value). You may use any number and any letter (upper- or lowercase), and you may also use the at sign (@) and the underline character

(\_). I'm speaking here of the variable itself and not the assignment made in quotation marks. You could use a variable name such as:

**@123RT\$ = "HELLO"**

You could not use:

**.,453W\$ = "HELLO"**

because commas are not allowed in a string name. **CAR\$ = "HELLO"** is fine. **CAR\$R\$ = "HELLO"** is not; the dollar sign correctly appears at the end of the name, but another dollar sign is included in the name.

When naming numeric variables, you cannot use any of the statements, functions, and commands found in TI BASIC. For example, **LISST = 1** is fine, because **LISST** is not a statement, function, or command. **LIST = 1** is unacceptable and will result in an error message, because this word is used in TI BASIC. This is not true with string variables, however. **LET LIST\$ = "HELLO"** is fine, because **LIST\$** is not a statement, command, or function. However, there are a few statements in TI BASIC that end with the dollar sign. These are **CHR\$**, **SEG\$**, and **STR\$**. These may not be used as string variable names. If you attempt to do this, an error message will appear on the screen.

## **THE VAL FUNCTION**

Sometimes it is necessary to extract a numeric value from a string variable. The **VAL** function is used to extract this value. It converts a string variable containing numbers to a numeric variable. The following program shows how **VAL** can be used:



```

10 A$ = "1234"
20 X = VAL(A$)
30 PRINT X

```

When this program is run, 1234 will appear on the screen. This is the numeric value of A\$. You could have skipped line 20 altogether and changed line 30 to **PRINT A\$**. You would come up with the same screen display. Let's go further and demonstrate the real value of the VAL function:

```

10 A$ = "1234"
20 X = VAL(A$)
30 Z = X/2
40 PRINT Z

```

Now we can perform mathematical computations with the numeric value of A\$. We can't do this with:

**Z = A\$/2**

This is an illegal call. However, line 20 assigns X the *numeric* value of A\$. So X is now a numeric variable. Line 30 assigns the variable Z the value of X divided by 2. Line 30 will run exactly as if it were entered as **LET Z = X/2**.

## THE LEN FUNCTION

The LEN function reads the number of characters in a string variable. It stands for length. It makes no distinction between letters, numbers, or spaces. A space in a string variable is considered a character. The LEN function assigns a numeric value to a numeric variable. The numeric value is equivalent to the number of characters in the string variable.

The following program demonstrates this:

```

10 A$ = "HELLO"
20 X = LEN(A$)
30 PRINT X

```

When this program is run, the screen will display the number 5. This is because there are 5 letters in HELLO, which has been assigned to A\$.

Here's another example:

```

10 A$ = "HOW ARE YOU"
20 X = LEN(A$)
30 PRINT X

```

When this program is run, 11 will appear on the screen, because there are 11 characters in A\$. Nine of these characters are letters, and two are spaces. If A\$ was equal to "HOW ARE YOU 1234", then the LEN value would be 16.

The LEN function can be used in programs that test typing accuracy. A phrase may be displayed by the computer, and its LEN value detected. The phrase entered by the typist is then tested for its LEN value, and the two are compared.

String variables can be used to decrease memory requirements in a program where certain words must be repeated over and over again. If these words are inserted using Print statements without variable assignments, each character in the word will consume a byte of memory. However, when these words are assigned to string variables, the Print statements are coupled to the proper variable, the word does not have to be re-spelled in the program line, and less memory is required to hold and run the program.

## IF-THEN-ELSE AND GOSUB

The If-Then statement tests for a certain condition and creates a branch when the condition is true. If the statement **IF X = 20 THEN 500** is used, a true condition occurs when X is equal to 20. If X is not equal to 20, then no branch occurs and the next line in the program is executed. The If-Then statement may also include an Else command. Here's one way it might be used:

```
10 IF X = 20 THEN 500 ELSE 1000
```

This statement tells the computer If X is equal to 20, Then branch to line 500 but If X is not equal to 20, then branch to 1000.

In TI BASIC, the If-Then-Else statement combination is often used with another type of branch statement. This is GOSUB, which is identical to GOTO in that it creates a branch to another portion of the program, but an automatic return is built in. GOSUB stands for go to a subroutine. A subroutine is a program segment used by another program.

The GOSUB statement must always have a Return statement. Just like a For-Next loop begins with the For statement and ends with a Next statement, a GOSUB branch begins with GOSUB and ends with Return. This program will show you how GOSUB works:

```
10 LET A = 10
20 GOSUB 70
30 PRINT B
40 PRINT C
50 PRINT D
60 END
70 LET B = A/2
80 LET C = A*3
```

```
90 LET D = A+5
100 RETURN
```

When the program is first run, A is assigned the value of 10 in line 10. Line 20 contains the GOSUB statement which branches to line 70. Lines 70 through 90 work A into different formulas (division, multiplication, and addition) and assign the values to numeric variables B, C, and D. Line 100 contains the Return statement, and this creates a branch back to the line that immediately follows the one containing the GOSUB statement. In this case the line is 30. Lines 30 through 50 print the values of B, C, and D.

The End statement in line 60 prevents the program from executing lines 70, 80, 90, and 100 again. These lines make up a subroutine, which is entered only upon execution of the GOSUB statement in line 20. A previous program where a For-Next loop was exited and then reentered with GOTO statements was given in improper form to demonstrate the error message that occurs when a Next is encountered without an appropriate For statement (For-Next error). You may recall that an End statement was inserted at the end of the For-Next loop and before the subroutine branched to from the loop had begun. The End statement in line 60 of this program accomplishes the same goal. Without it, the computer would reassign the values of B, C, and D in lines 70, 80, and 90. However, when it reaches line 100 and the Return statement, an error message would be generated indicating a GOSUB error. The actual message would be **CAN'T DO THAT** which would appear at the bottom of the screen.

The same program run can be accomplished by changing line 20 in the program to GOTO 70 (instead of GOSUB 70). Line 100 would have to include the branch back statement, which in this case, would be GOTO 30.

So why use GOSUB at all? There are many reasons that are not apparent when writing short programs. A GOSUB statement allows you to set up subroutines that may be several hundred lines away from the branch which accesses them, as in:

```
20 GOSUB 6000
```

You could also use GOTO 6000, but when you wrote the subroutine starting at line 6000, you might have to sift back through your program to find the correct line number to branch to when exiting the subroutine. By using GOSUB, however, the Return statement *automatically* branches back to the line immediately following the one that contains the GOSUB statement that accessed the subroutine in the first place. It is not necessary to include a line number with the Return statement, as the computer keeps track of the GOSUB statement line which accessed the subroutine. If you use a line number following a Return statement, you'll get an error message.

This is only part of the reason for the usefulness of the GOSUB-Return statement combination. In the original program using the GOSUB branch to line 70, a GOTO 70 statement would work as well, providing that the Return statement was replaced with another GOTO branch. However, many computer programs may use one subroutine at different times in the program. There may be a GOSUB 100 statement in program line 20. There may be another GOSUB 100 statement in line 50.

Both of these statements access the same subroutine, but after the subroutine is run, the program must branch back to the line following the one which contained the GOSUB line that initialized access. This program will clarify the situation:

```
10 LET A = 10
20 GOSUB 80
30 PRINT B
40 LET A = 20
50 GOSUB 80
60 PRINT B
70 END
80 LET B = A + 5
90 RETURN
```

When this program is run, your screen will display the number 15 and then below it, the number 25. Here's what is happening. In line 10, the variable A is assigned the value of 10. The GOSUB statement in line 20 branches to the subroutine beginning at line 80, where B is assigned the value of A + 5, or 15. The Return statement in line 90 branches to line 30 (the line following the program line that contained the GOSUB statement which accessed the subroutine). Here's where the difference comes in. After the value of B is printed in line 30, line 40 is executed, and the variable A is *reassigned*. The new value for A is 20. Another GOSUB statement is encountered in line 50. This one branches to the same subroutine and B is also reassigned in line 80 to the value of A (now 20) plus 5, or 25. When the Return statement is encountered in line 90, the branch is to line 60 which follows the GOSUB statement that accessed the subroutine.

You can never use GOTO statements in a program like this. If you changed line 20 to GOTO 80, line 90 would have to be changed to GOTO 30 in order to branch back properly. This is fine. However, when the second GOTO statement in line 50 accesses the subroutine, the branch would be GOTO 30, and program execution gets messed up. You can use 50 GOSUB statements to access one subroutine and only a single Return statement is required at the end of the subroutine.

GOSUB and Return statements make up one of the most powerful operating aids in BASIC language.

Subroutines may be accessed during every program run or during just a few program runs, depending on some other value. For instance, a subroutine might be used to graphically draw a playing card on the computer screen, such as the ace of spades. This subroutine might be entered only when a number was output from a random number generator which represented the ace of spades. If this number was not output, the subroutine would never be entered.

GOTO statements are more often used to branch to separate portions of the main program and more importantly, to skip over program portions.

## MORE ON FUNCTIONS

A function may be thought of as a command or statement that is used with another statement in TI BASIC. The INT or integer function is used often. An integer is a whole number. It contains no decimal portion. The numbers 1, 2, 3, 4, and 5 are integers, but while 2.349 is not. The INT function is used to chop off the fractional or decimal portion of a

number when that number is greater than or equal to zero, leaving just the whole number. For example, INT(2.349) is 2. Let's see how this function is used in an actual program.

```
10 X = 45.3896
20 A = INT(X)
30 PRINT A
```

When this program is run, the screen will display the number 45, which is the integer of 45.3896. The decimal portion has been truncated or chopped off.

Another function in TI BASIC is RND for random. It supplies you with a random number that is always less than 1. To take full advantage of the RND function, use the Randomize statement in your program as well. When this statement is used at the beginning of a program, the random number generator lets the RND function return a number that is completely random. The sequence of numbers follows no detectable pattern. The following program gives an example.

```
10 RANDOMIZE
20 A = RND*10
30 PRINT A
```

Each time you run this program, the value of the random number returned by RND will be printed on the screen. The value should be different each time you begin the program because of the Randomize statement in line 10. Now, remove line 10 from the program. Do this by typing 10 and then pressing Enter. Now run the program again. Each time you run this program, the same number will crop up. This demonstrates the importance of the Randomize statement.

## A DICE GAME PROGRAM

Let's combine much of what we've learned and write a game program to simulate the roll of a single die. It is necessary to use the RND function to get a sequence of numbers that cannot be predicted ahead of time. To do this, we multiply the random number returned by the RND function by another number. There are six sides on a die and six possible numbers of from 1 to 6, so the following line would do the trick:

$X = \text{RND} * 6$

We multiplied the random number times 6. This will come up with a different number each time that is no greater than 6. This is a good start, but there's more to it. First, you will never be able to get a 6 with this line. Remember, the number returned will always be less than 1, so you can never quite achieve a 6 at the output. Also, you're going to end up with outputs like 4.13867 instead of a 4 or a 5, which would be displayed by a die. Here's where the INT function comes into play. A program line such as:

$X = \text{INT}(\text{RND} * 6)$

means that the number output will always be an integer. But the 6 still has not been achieved, since the random number will always be less than 1. Also, if the random number is small enough, when it is multiplied by 6, it will still be less than 1, and the integer of a value which is more than 0 and less than 1 is 0. No dice game you ever played displayed a 0 on the cube

face. We can solve this problem very easily, however, with the following program line:

$X = \text{INT}(\text{RND} * 6) + 1$

$\text{RND} * 6$  will always be equal to less than 6. The integer of  $\text{RND} * 6$  will always be equal to 0, 1, 2, 3, 4, or 5. When you add 1 to any of these numbers, you come up with the equivalent of the numbers that can be anticipated from the roll of a single die. If the integer of  $\text{RND} * 6$  is 0, then the screen will display a 1. If the output of  $\text{RND} * 6$  as an integer is 5, then the screen will display a 6. It will also display every number between 1 and 6, so this is exactly what we need.

To get a screen display requires another line or two. Here's the dice game program:

```
10  RANDOMIZE
20  CALL CLEAR
30  X = INT(RND*6) + 1
40  PRINT X
50  INPUT A$
60  GOTO 20
```

Line 10 clears the screen, and line 20 contains the Randomize statement that gets us a random number each time RND is used. Line 30 has been discussed. This assigns X to a random number which is an integer between 1 and 6 and represents the face of a die. Line 40 prints the value of X on the screen. Line 50 is a pause command using the Input statement followed by a string variable. This stops execution until Enter is pressed and gives you the opportunity to note the random number that has been

printed on the screen. When you want to roll the cube again, press Enter. This causes line 60 to be executed, which branches to line 10 and starts the program over again. You could also have included a Call Clear command between lines 50 and 60 and then changed your GOTO branch in line 60 to GOTO 30. Either way, the program will return random numbers corresponding to the faces of a die. As a game program, line 50 can be changed to 50 INPUT "PRESS ENTER TO ROLL AGAIN":A\$ to provide an on-screen prompt that would get the player into the spirit of what the computer was trying to simulate.

Most dice games include two cubes, so how do you go about programming for two random numbers between 1 and 6? Simply add:

```
35 Y = INT(RND*6) + 1
```

and change line 40 to:

```
40 PRINT X;Y
```

That's all there is to it. You will now see two numbers displayed on the screen that correspond to a pair of dice. You can add one more line which will add your results, as in:

```
36 Z = X + Y
```

Add this line:

```
41 PRINT Z
```

Using these program additions, the two values will be printed on the screen side by side and below them will be the value of the two when

added together. The variable Z has been assigned the value of X plus Y, and line 41 prints the value of Z on the screen.

Let's go one step further. In a true dice game, certain combinations result in a win, others result in a loss. Normally, a 7 or 11 on the first roll is a winner, and a 2 (snake eyes) or a 12 (box cars) is a loser. We can modify the program to take this into account. We must use the If-Then statement described earlier. The program is:

```
10 CALL CLEAR
20 RANDOMIZE
30 X = INT(RND*6) + 1
40 Y = INT(RND*6) + 1
50 Z = X + Y
60 PRINT X;Y
70 IF Z = 7 THEN 110
80 IF Z = 11 THEN 110
90 IF Z = 2 THEN 150
100 IF Z = 12 THEN 150 ELSE 120
110 PRINT Z;"A WINNER!"
120 INPUT A$
130 CALL CLEAR
140 GOTO 30
150 PRINT Z;"YOU LOSE!"
160 INPUT A$
170 CALL CLEAR
180 GOTO 30
```

Line 10 clears the screen. Line 20 reseeds the random number generator. Lines 30 and 40 set up values that represent die faces and are assigned to the variables X and Y. Line 50 assigns the variable Z to the sum of X and Y, and line 60 prints X and Y on the screen. The sum of X and

Y is assigned to variable X, but Z is not always displayed on the screen. The value of Z is most important, as the If-Then statements in lines 70 through 100 test for its value and branch accordingly. Line 70 tests for any dice combination which is equal to 7 (5 and 2, 4 and 3, 6 and 1). Any of these combinations will make  $X + Y$  equal to 7. If Z is equal to 7, there is a branch to line 110, which instructs the computer to print the value of Z on the screen followed by the phrase **A WINNER!** Here, the value of Z is displayed, but this is only the case in the event of a win. Line 80 tests for the condition of Z being equal to 11 (6 and 5). If this is the case, there is a branch again to line 110, which prints Z (this time 11) followed by the same phrase **A WINNER!**

Lines 90 and 100 test for a loss, which is a 2 or 12. If a 2 is rolled, there is a branch to line 150 where Z is again printed, but, this time, it's followed by the phrase **YOU LOSE!** The same thing occurs in line 100 if a 12 is rolled. However, the If-Then statement is coupled with an Else as well in line 100 to make sure that some kind of branch occurs.

As soon as a win or a loss has been recorded, an Input A\$ statement is encountered (lines 120 and 160). This halts execution until Enter is pressed, giving you time to see the numbers displayed on the screen. When you press Enter, a Call Clear statement is executed, clearing the screen. There is then a branch to the start of the dice routine. This occurs anytime a 2, 7, 11, or 12 is rolled.

What happens when a number other than these four is rolled? First, the numbers are printed on the screen in line 60. Line 70 checks to see if the sum is a 7. When it's not, line 80

checks to see if it's an 11. Failing here, line 90 checks for a 2, and failing here, line 100 checks for a 12. If the number is not a 7, 11, 2, or 12, there is a branch to line 120, because of the Else statement in line 100. This hops over the Print statement in line 110 and executes the Input A\$ statement, which halts the run until Enter is pressed. When this occurs, the program runs again.

This does not exactly duplicate a true game of "craps." Here, players are given the opportunity to roll for points, assuming a win or loss has not occurred on the first roll. If you roll a 5 on your first try, you continue to roll until you get another 5 (a winner) or a 2, 7, 11, or 12 (losers). The last four numbers are always losers in the game of craps when going for a point. A 7 and 11 are automatic winners on the first roll only. A 2 and a 12 are always losers.

Here is the completed dice program that lets you win with a 7 or 11 on the first roll, lose with a 2 or 12 anytime, and even attempt to reach your point, should no win or loss occur on the first roll.

```

10 CALL CLEAR
20 RANDOMIZE
30 I = I + 1
40 X = INT(RND*6) + 1
50 Y = INT(RND*6) + 1
60 Z = X + Y
70 PRINT X;Y
80 IF (I>1)*(Z=7) THEN 220
90 IF Z = 7 THEN 160
100 IF (I>1)*(Z=11) THEN 220
110 IF Z = 11 THEN 160
120 IF Z = 2 THEN 220

```

```

130 IF Z = 12 THEN 220
140 IF (I>1)*(Z=B) THEN 160
150 IF I = 1 THEN 280 ELSE 320
160 PRINT Z;"A WINNER!"
170 INPUT "PRESS ENTER TO
    ROLL AGAIN":A$
180 CALL CLEAR
190 I = 0
200 B = 0
210 GOTO 30
220 PRINT Z;"YOU LOSE!"
230 INPUT "PRESS ENTER TO
    ROLL AGAIN":A$
240 CALL CLEAR
250 I = 0
260 B = 0
270 GOTO 30
280 B = Z
290 PRINT Z;"IS YOUR POINT"
300 INPUT "PRESS ENTER TO
    ROLL AGAIN":A$
305 CALL CLEAR
310 GOTO 30
320 PRINT "YOUR POINT IS";B
330 PRINT "YOU ROLLED";Z
340 INPUT "PRESS ENTER TO
    ROLL AGAIN":A$
350 CALL CLEAR
360 GOTO 30

```

Lines 10 through 70 are repeats of the previous program. Starting at line 80 things change a bit. It's quite acceptable to use a more complex test condition with If-Then statements. What line 80 is saying is If I is more than 1 and Z is equal to 7, Then branch to line 220. Two conditions have to be true before this branch can take effect. One additional line has been added

among the first seven. Line 30 counts the number of times line 30 is executed.

Line 30 is part of a loop. Each time the dice roll, the loop steps up by 1. When a win or loss occurs, the loop starts at 0 again. When line 30 is first executed, the variable I has been assigned no value, so it's equal to 0. However, in line 30, this value is added to 1, so I is now equal to 1. When the next roll occurs (assuming no win or loss), line 30 is executed again but during the second roll, the variable I has an assigned value of 1 (from the first roll). When the instructions in line 30 are carried out, the value of I is added to 1 and is now 2. The third roll of the dice lets I be equal to 3, and so on. It is necessary to differentiate only between the first roll and all subsequent rolls, since a 7 or 11 indicates a win on the first roll only.

Line 80 tests for Z being equal to 7 during any roll but the first one. There is then a branch to line 220, indicating a loss. Again, line 80 says If I is equal to a value which is more than 1 *and* Z is equal to 7, Then branch to line 220, to indicate a loss. Assume that a loss has occurred. Line 220 prints the dice value, along with the loss display. Line 230 tells you to press Enter to start again. Line 240 clears the screen, and lines 250 and 260 reset the value of I to 0. The same is done for B, which is another variable that will be discussed soon. Line 270 branches back to the beginning of the program. All this occurs when a 7 is rolled during any but the first roll. However, suppose we're in the first roll and Z is equal to 7. No problem. Line 80 will not bring about a branch, because I will not be more than 1. It will be equal to 1. The second condition in this line is true (Z=7), but for this type of branch to occur, both conditions



must be true. Line 90 is not so picky. It says If  $Z = 7$  Then branch to line 160 and record a win. If  $Z$  is equal to 7 on the second roll line 90 won't record a win here.

Line 80 is executed before line 90, and if it's the second roll and  $Z$  is equal to 7, then line 90 will not even be executed. It will be branched around (to line 220).

Lines 100 and 110 work in the same manner, but test for a condition of 11 on any but the first roll (line 100) and on the first roll (line 110). The appropriate branches occur, depending on whether the roll is a win or a loss. A 2 and 12 are always losers, so lines 120 and 130 remain identical to the earlier simple program.

Line 140 tests for a point being reached. To explain this, move to line 150 and assume that we're on the first roll and that a number other than 2, 7, 11, or 12 has occurred during this roll. Line 140 requires that  $I$  be more than 1, but remember that this is the first roll and none of the other If-Then conditions have been true, so these lines are skipped over.

Line 150, however, says If  $I$  is equal to 1, Then branch to line 280. If it's not equal to 1, Then branch to line 320. This falls at the end of all of our If-Then lines. The only way line 150 will be executed during the first roll is if a 7, 11, 2, or 12 have not been rolled. Line 150 will then branch to line 280, and this sets your point.

Assume you roll a 4 on the first roll. There is a branch to line 280, and here, the variable  $B$  which was previously assigned is given the value of  $Z$ , which is 4. Lines 280 and 290 print your point value and you are then prompted to press Enter to roll again. This branches to line 30, and you enter the dice roll routine again. Line 80 checks for a 7, and if this is rolled, you lose, because  $I$  is now equal to 2 and 7 can

never be a winner after the first roll. Line 90 has no effect, because if  $Z$  is equal to 7, line 80 has already made the losing branch. Line 100 checks for the roll of 11, as line 80 did for 7. If this condition is true, you lose. Line 110 has no effect, because 11 can never be a winner on the second roll. Lines 120 and 130 are armed and ready to indicate a loss if snake eyes or box cars are rolled.

Line 140 becomes important. The new value of  $Z$  was determined during the second roll. However, the value of  $B$  has been assigned the value of your first roll. If  $Z$  is equal to  $B$ , then you've reached your point and a win is recorded. There is a branch to line 160, which prints your point, along with the winner phrase. You are then prompted to press Enter to roll again. When this is done, the screen is cleared, the value of  $I$  is returned to 0, and the value of  $B$  is returned to 0.

If you get all the way down to line 150 without a branch again, it means that you rolled something other than a 2, 7, 11, 12, and finally, a 4, which is your point. Assume you rolled a 6 on the second go-around. You're down to line 150 again. Here,  $I$  doesn't equal 1 (it equals 2), so the branch to line 280 can't occur. The Else statement in line 150, however, branches to line 320. Here, the value of the point you're trying to roll is printed again followed by the value that was actually rolled. Press Enter to roll again, the screen clears, and you branch back to line 30. This time around,  $I$  will be equal to 3. Eventually, one of your rolls will have to satisfy one of the conditions in lines 80 through 140. If you finally roll your point, the win is recorded by branching to the win lines (beginning at line 160). You may lose by coming up with the wrong combination, and there

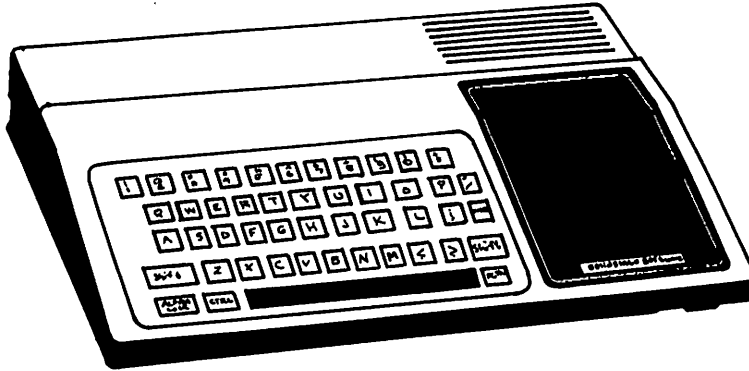
will be a branch to the loss lines, beginning at line 220. Whenever a win or a loss occurs, the values of I and B are reset to 0. Lines 80, 100, and 140 contain slight alterations of If-Then statements when compared to those previously encountered. These are used to specify two sets of conditions before a branch actually occurs. Other than this, everything else is pretty much standard. If you understand how this program operates (and it may take you several readings to get it all), then you're equipped to strike out on your own and explore TI-99 BASIC to a greater extent.

If you understand what has been presented thus far, then you know how to print information on the monitor screen, establish For-Next loops, use GOTO branches to access different portions of the main program, and branch to subroutines using GOSUB and Re-

turn statements. You also know how to use the Randomize statement and the RND function to get a random number output. You should also know that by using the INT function, the random number will always be an integer.

The statements and functions discussed in this chapter are the very basic building blocks of computer programming. If you understand every one, learning the additional commands, statements, and functions of TI BASIC should be much easier for you. Use the TI manual, and this book, and you will overcome most difficulties and become quite proficient at programming your TI-99/4A. With each TI-99 computer sold, Texas Instruments include two excellent manuals. One is the *User's Reference Guide*; the other is *Beginner's BASIC*. Both will serve as excellent guides to the machine and its language.

## Chapter 5



# TI-99/4A Graphics

TI BASIC has a special set of subprograms built into the computer. These let you produce on-screen colors, graphics, and sounds. Whenever you want to use any of these special subprograms, you must call for them by name using the Call statement. Additionally, you will have to provide a few specifications to be used in the subprogram. From this point on, the subprogram does the rest.

The subprograms we are primarily interested in are CHAR, VCHAR, and HCHAR. When any one of these subprograms is to be accessed during a program, you use a Call statement, such as Call VCHAR.

### SCREEN COORDINATES AND ASCII

Before using the subprograms you need to understand the screen coordinates of the TI-

99/4A, as well as the ASCII character set. The TI-99/4A prints characters on the screen that fill tiny blocks. Figure 5-1 shows the display screen divided into 768 blocks, each of which is the same size. The blocks are numbered horizontally, from left to right, starting at 1 and ending at 32. Blocks are also numbered vertically from 1 to 24. When discussing display screens, we refer to a horizontal line of blocks as a row and a vertical line as a column. The TI-99/4A screen consists of 24 rows and 32 columns.

Each of the 768 blocks is broken down into 64 tinier blocks, as shown in Fig. 5-2. When your screen is filled with information, 49,152 blocks have been filled in. The character set for the TI-99/4A is determined by filling in some of the 64 tiny blocks and not filling in others.

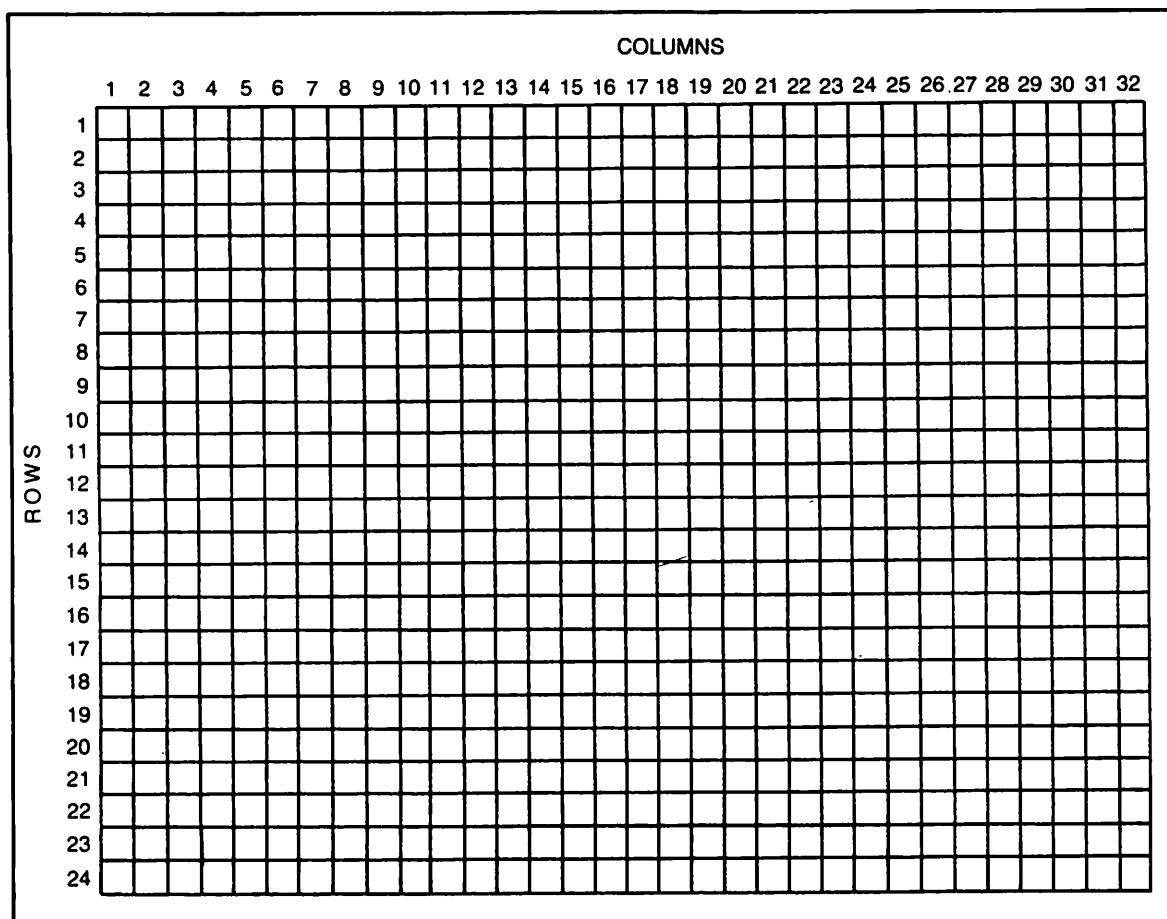


Fig. 5-1. Coordinate format of the TI-99/4A display screen.

Figure 5-3 shows how an O is formed on the screen by filling in some of the 64 blocks and leaving all the others vacant.

When doing on-screen graphics with the TI-99/4A, you must also understand the ASCII codes that represent the machine's character set. Each character is represented by an ASCII number. Appendix B contains the complete character set and ASCII number information

and should be used as a reference whenever you're programming graphics.

A capital letter A is generated by ASCII code 65. The lowercase A is generated by ASCII code 97. A comma is ASCII code 44, and a space is ASCII code 32.

Sometimes it is necessary to print a letter, number, or character at a certain location on the screen. In this case, we cannot specify

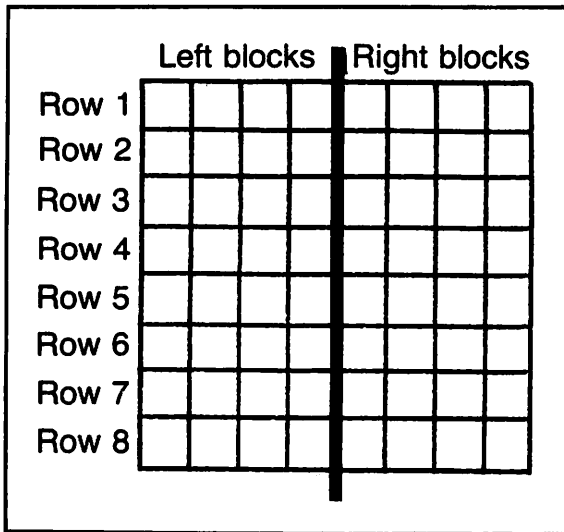


Fig. 5-2. Each character is created by filling in up to a total of 64 grid blocks.

the character by its keyboard designation. We have to use its ASCII code.

## HCHAR

The HCHAR subprogram is used to place a character anywhere on the screen by specifying the row and column coordinates. This subprogram can also repeat the character horizontally the number of times specified. This subprogram is used with the Call statement, as in:

```
CALL HCHAR(12,16,65)
```

The first number in parentheses identifies the screen *row* (vertical) where the character is to be printed. If you refer to Fig. 5-1, you see that row 12 is at the center left of the screen. The second character specifies the column position (horizontal). This lies at the top center

of the screen. However, when these two numbers are combined in this manner, you're telling the machine to print a character in row 12 and at the sixteenth column position. This falls in the exact center of the screen (Fig. 5-4). You first locate row 12 and then you move toward the right until you hit column 16. This is where the character will be printed.

There's a third number in parentheses, and this specifies the character to be printed, giving its ASCII code number. Here, the code is 65, so a capital letter A will be printed at screen position 12,16.

Although not shown in the previous example, you can add one more number to those already contained in the parentheses. This is called the repeat number, and it may be used to repeat the character specified any number of times. For example, in:

```
CALL HCHAR (12,16,65,5)
```

a capital letter A will be printed at the center of

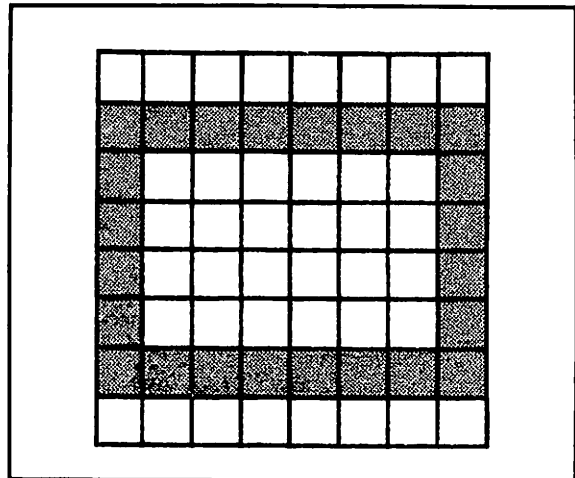


Fig. 5-3. The form of the letter O using the 64-block grid.

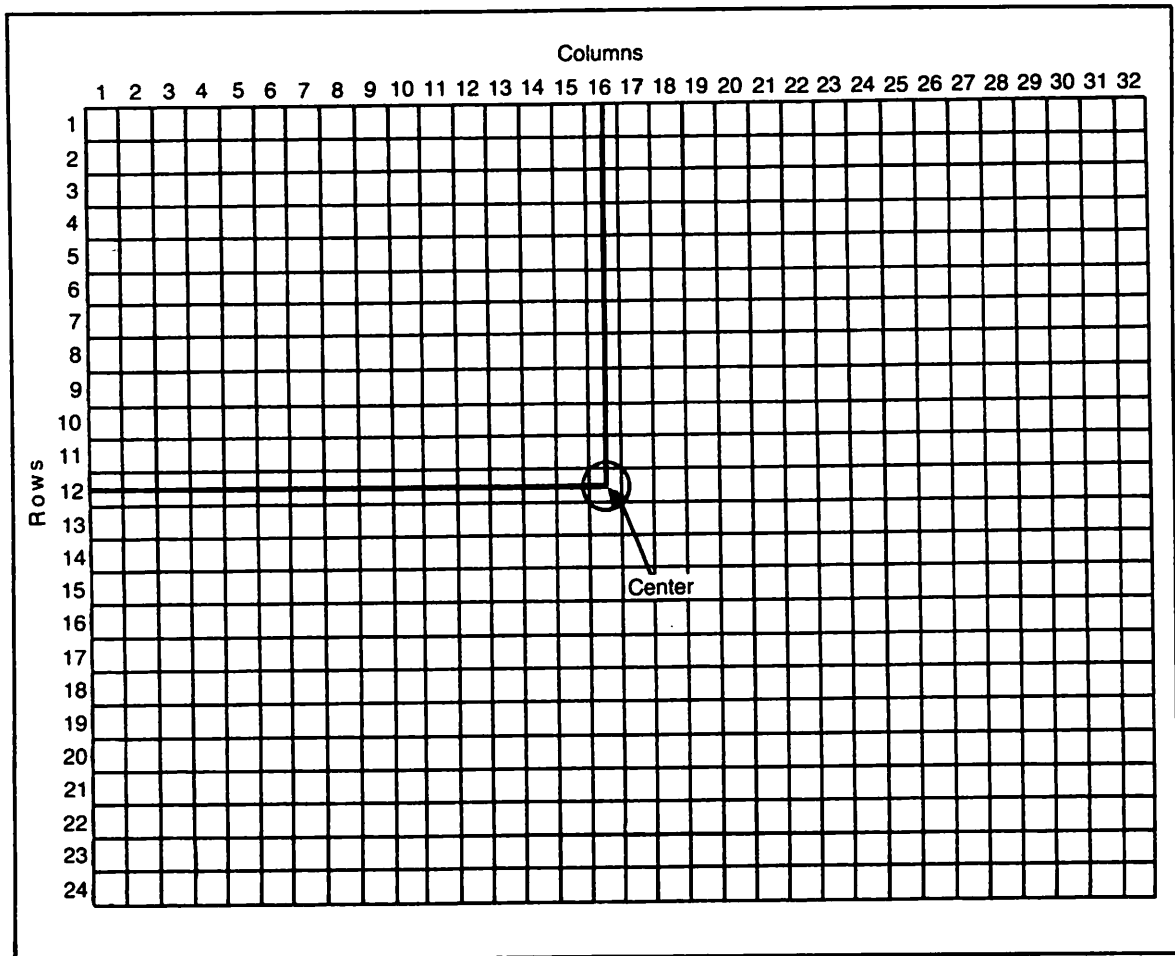


Fig. 5-4. The center of the screen is represented by the coordinate designations 12,16.

the screen, but the last number in parentheses (5) specifies that A is to be repeated 5 times. The first character will be printed at position 12,16. All other characters will be printed in a horizontal format to the right of the first character. The output from this program will be:

AAAAA

with the first character at screen position 12,16. The next character will be at screen position 12,17; then 12,18; etc. If you specify the repeat of 20 of these characters, as in:

**CALL HCHAR(12,16,65,20)**

you will run out of horizontal spaces in row 12. Remember that there are 32 columns to a

single row. Since you started at column 16, this means that the row can hold only 16 more characters. By specifying the repeat of 20 characters, you run out of columns in line 12, so the machine automatically advances to row 13 and prints the additional characters here. The result will be:

```

      AAAAAAAAAAAAAAAAAA
AAA

```

### VCHAR

The VCHAR subprogram is identical to HCHAR, except the optional character repeat occurs in a vertical format. The following demonstrates this:

```
CALL VCHAR(12,16,65,5)
```

This program causes a vertical column of capital As to appear at the center of the screen. The first A is printed at coordinate 12,16. The second one will be at 13,16; then 14,16, etc.

If you want to print a single character at a certain location on the screen, you may use either HCHAR or VCHAR. For instance:

```
CALL HCHAR(12,16,65)
CALL VCHAR(12,16,65)
```

will produce the same results on the screen.

We can use VCHAR and HCHAR together in programs to produce simple on-screen graphic displays and even to make a chart or two. The following program makes a large letter T on the screen using small capital Ts:

```
10 CALL CLEAR
```

```
20 CALL HCHAR(6,10,84,11)
30 CALL VCHAR(7,16,84,10)
```

Line 20 causes 11 letters (T) to be printed horizontally on the screen. Line 30 causes the same letters to be printed vertically at the center of the horizontal column.

The For-Next loop can be used with these subprograms to produce some interesting results, including pictures and graphs. The following program gives a simple demonstration:

```
10 CALL CLEAR
20 FOR X = 1 TO 10
30 CALL HCHAR(X,16,42,5)
40 NEXT X
```

This program produces the following results centered on the screen:

```

*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

You can also use Input statements to make an effective bar graph. The following program does this:

```
10 CALL CLEAR
20 INPUT A
30 INPUT B
```

```

40 INPUT C
50 CALL CLEAR
60 CALL HCHAR(5,1,42,A)
70 CALL HCHAR(10,1,42,B)
80 CALL HCHAR(15,1,42,C)

```

This program gives you the opportunity to enter three numeric values, A, B, and C. These values are then fed to the Call HCHAR subprograms in lines 60, 70, and 80. This generates horizontal bar graphs, starting at the left side of the screen. Line 60 specifies that the first character is printed in row 5 at column position 1. Line 70 begins the next graph by dropping down 5 rows, but again, the first character is printed at position 1. The same is true of the subprogram in line 80. Five more rows have been skipped, but the same column starting position is used. Assuming values of 8, 15, and 20 for variables A, B, and C, respectively, the following chart will be displayed on the screen:

```

*****
*****
*****

```

Here are three bar graphs that represent the numeric values by adjusting their lengths accordingly. The first bar contains 8 asterisks, the second 15, and the third 20.

## CHAR

The Call CHAR subprogram is used when it is necessary to generate characters that are not a part of the TI BASIC character set. This subprogram lets you design your own characters by filling in the proper number of squares

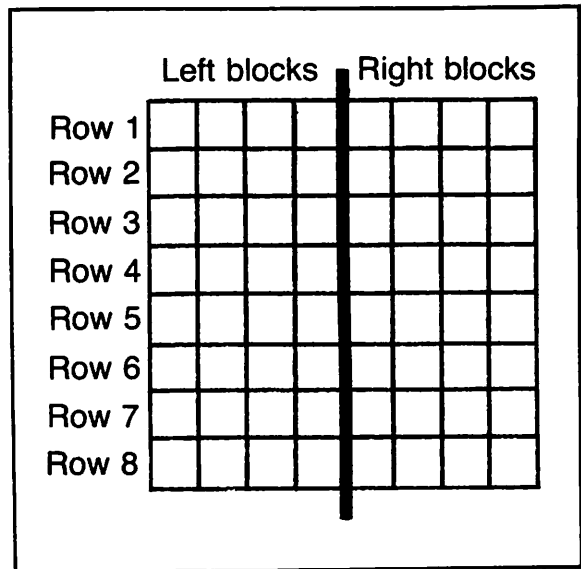


Fig. 5-5. Breakdown of the 64 blocks is handled in rows of 8.

that make up each character block.

Figure 5-5 shows the 64 blocks that make up a single screen character block. These are broken down into eight rows, with 2 block sets per row. Each block set contains 4 squares. The first 4 blocks in a row are given a certain numerical specification, followed by a numerical specification for the second set of 4 blocks.

There are 2 numerical specifications for each row, so each block character is defined by 16 numbers (2 times 8 rows). Just remember that there are 8 rows to each character and 8 possible columns in each row. The 8 columns are broken down into 2 major sets, each containing 4 columns. Remember now, I'm speaking here of the 64 tiny squares which make up one screen position.

Assume that we want to make a character that consists of filling in only one block of the 64. We have to provide a number for the one



block to be filled in, and we also have to provide numbers for those which are not to be filled in. Remember, zero is a number and is used to indicate the blocks that are not to be filled in. Figure 5-6 shows the 64-block grid with one square filled in to form a character. This square is the first one in row 1 and is specified with a certain number. The other squares are left blank, so these must be specified with another number.

We do not have to insert a number for each square, but rather for each set of four blocks. The first row requires two numbers to describe its two sets of character blocks. The same applies to the remaining seven rows. Any set of four blocks uses one number to describe the blocks that are to be filled in. If the first block is to be filled in and the rest left vacant, then one number will describe this situation. If the first two blocks are to be filled in, then one

number will describe this. If all blocks are to be filled in, yet another number will describe this.

While there are 64 total blocks in each grid, only 16 numbers need be given to describe any possible pattern that can be derived from this grid. Two numbers are given per row.

To make things a bit more complex, the numbers that describe each block set are given in hexadecimal notation. This is just another number system using 16 as a base instead of 10, which is the base in the decimal system. It is not important to know how the system is derived or even how to convert from decimal to hexadecimal. For programming graphic characters, all you need is the chart shown in Fig. 5-7. This tells you what number or letter to use in order to describe the blocks you wish to have filled in.

Hexadecimal code uses letters to describe numbers above 9. The letter F in hexadecimal code is really a number. Looking at the chart, we see that if no blocks are to be filled in in any 4-block set, use 0. In the next row, if you wish to fill in the fourth block only, use hexadecimal code 1. This does *not* mean that if you wish to fill only one block in a row, you use the number 1. It means that if you specifically want to fill in the last block in a 4-block row, use 1. If you wish to fill in the third block, use 2; the third and fourth blocks are filled in by hexadecimal code 3; and so forth.

Figure 5-8 shows a sample pattern using the 64-block grid. This pattern was chosen at random, and the hexadecimal code for each block set of 4 is given to the right. Row 1 contains no filled-in squares. We know the hexadecimal code for no blocks filled is 0. Therefore, this row is represented by the

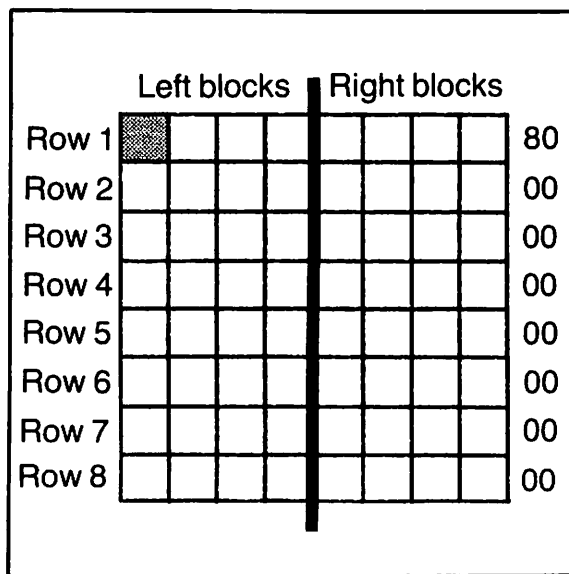


Fig. 5-6. The 64-block grid with one square filled in and each row numbered accordingly.

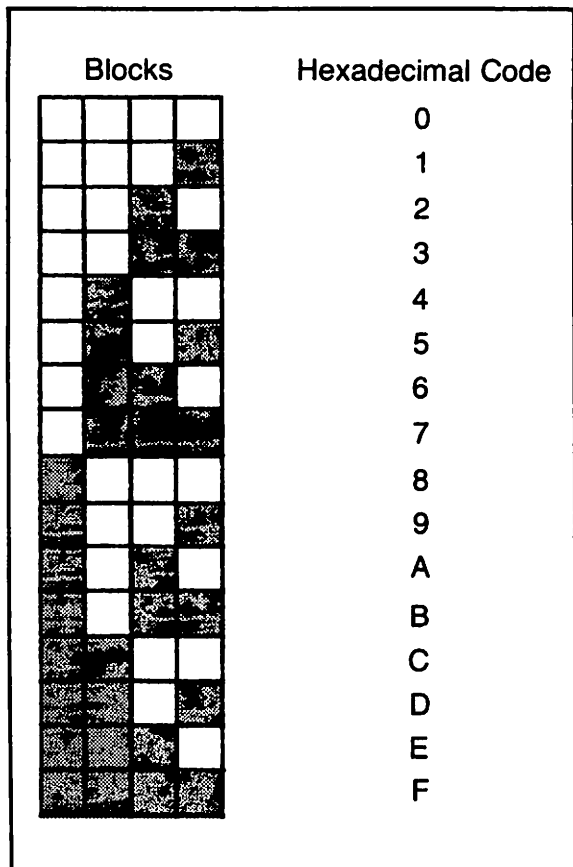


Fig. 5-7. The hexadecimal chart for filling in block grids.

hexadecimal code "00". The first 0 describes the left side of blocks in row 1, while the second 0 describes the right set. In row 2, the condition is the same so hexadecimal code "00" represents this row as well.

In row 3 the last block in the left block set is filled in, as is the first block in the right block set. The hexadecimal code to describe this row is "18". The 1 indicates that the last block in the left block section of row 3 is to be filled in. The 8 indicates that the first block in the right block section is to be filled in. Rows 4 through

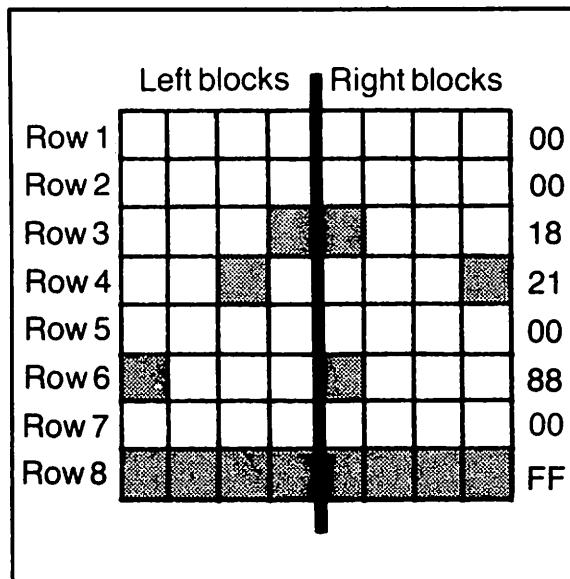


Fig. 5-8. A typical grid figure.

8 follow the same pattern. In the last row, all blocks are filled in in each block set. The code that describes a complete block set fill-in is F. Therefore, FF means fill in both block sets on this row.

We now have a sample character of our own design. We can display it on the screen using the Call CHAR subprogram. Our new character must be assigned an ASCII number. It can be any number on the ASCII chart used to represent a character already in the machine set. Any number from 32 to 127 will do. Let's use the character code (33) for this example. ASCII code 33 represents an exclamation point (!). However, we're going to use this number to represent our new character. The following program defines the new character and assigns it to ASCII code 33:

```
CALL CHAR(33,"00001821008100FF")
```

This line tells the computer to assign to ASCII code 33 character identified by the hexadecimal code. The code was derived from the grid shown in Fig. 5-8. The following program will display the character at the center of the screen:

```
10 CALL CLEAR
20 CALL CHAR(33,"00001821008
   100FF")
30 CALL HCHAR(12,16,33)
40 GOTO 40
```

The first line clears the screen. Line 20 then inputs the new character pattern, assigning it to ASCII code 33. Line 30 uses the Call HCHAR subprogram to locate screen position 12,16 and then print ASCII code character #33 on the screen. This would normally be "!" but since this character code has been reassigned in line 20, the pattern shown on the grid in Fig. 5-8 appears as a single character on the screen. An endless loop is set up in line 40 so that the program does not end. Without this loop, you would see the new character and it would then suddenly be replaced by the original ASCII character (!) as the program terminated.

Let's try some block graphics now by filling in an entire character block. The following line will do this:

```
CALL CHAR(33,"FFFFFFFFFFFFFF-
FF")
```

The sixteen Fs in the hexadecimal code indicate that all sixteen four-block sets (64 blocks) are to be completely filled in. This creates a solid block character on the screen. The fol-

lowing program prints a solid line from left to right across the center of the screen:

```
10 CALL CLEAR
20 CALL CHAR(33,"FFFFFFFFFFFF-
   FFFFF")
30 CALL HCHAR(12,1,33,32)
40 GOTO 40
```

After the block character has been established in line 20, the HCHAR subprogram is used to print a string of 32 characters horizontally on the screen from position 12,1. ASCII character 33 is the block character established in line 20. The number 32 tells HCHAR to repeat this character 32 times, which is the maximum number of columns on any line. Your screen will display a solid line running across the screen from left to right at its center.

You could use a similar program, only substituting VCHAR for HCHAR in line 30, to draw a vertical line at the center of the screen. Here, you might include:

```
30 CALL VCHAR(1,16,33,24)
```

This would draw a vertical line starting at the top center and ending at the bottom center of the screen. By combining block characters with VCHAR and HCHAR, it is possible to draw different kinds of simple pictures on the screen.

## COLOR

The Color subprogram lets you change screen character colors and even the screen background. Again, the Call statement is used with this subprogram. You can choose up to 16

foreground and background colors, and you can specify which set of characters will be given which color. In the TI-99/4A, there are 16 character set numbers. These are shown in Fig. 5-9. Set 1 is comprised of ASCII character codes 32 through 39. Any character represented by the numbers 32 through 39 falls into this particular set number. The set number is important, because it must be used with the Call Color subprogram to specify which characters are to be given a certain color.

Each character displayed on the monitor screen has two colors. This includes the color of the dots that make up the character itself and

the color that occupies the rest of the character position on the screen. The latter are the blank spaces in the 64-square character grid. The filled-in spaces in the grid are called foreground color, while the others are background color.

Using the Call Color subprogram, you must first specify the character set number, then the foreground color, and finally, the background color. Figure 5-10 shows the 16 color codes, along with the colors they represent. If code 1 is chosen (transparent), then the present screen color shows through when the character is displayed. The following program shows how Call Color might be used:

| Set Number | Character Codes |
|------------|-----------------|
| 1          | 32-39           |
| 2          | 40-47           |
| 3          | 48-55           |
| 4          | 56-63           |
| 5          | 64-71           |
| 6          | 72-79           |
| 7          | 80-87           |
| 8          | 88-95           |
| 9          | 96-103          |
| 10         | 104-111         |
| 11         | 112-119         |
| 12         | 120-127         |
| 13         | 128-135         |
| 14         | 136-143         |
| 15         | 144-151         |
| 16         | 152-159         |

Fig. 5-9. The 16 character set numbers.

| Color-code | Color        |
|------------|--------------|
| 1          | Transparent  |
| 2          | Black        |
| 3          | Medium Green |
| 4          | Light Green  |
| 5          | Dark Blue    |
| 6          | Light Blue   |
| 7          | Dark Red     |
| 8          | Cyan         |
| 9          | Medium Red   |
| 10         | Light Red    |
| 11         | Dark Yellow  |
| 12         | Light Yellow |
| 13         | Dark Green   |
| 14         | Magenta      |
| 15         | Gray         |
| 16         | White        |

Fig. 5-10. The 16 color codes.

## CALL COLOR(5,7,13)

This line instructs the computer to display all characters in character set 5 with a dark red foreground and a dark green background. Character set 5 includes all characters with ASCII codes of from 64 to 71. The foreground color number is 7, and this specifies dark red. The third number in parentheses is the background color. The number 13 specifies dark green.

Once the Call Color subprogram is entered, all characters represented by ASCII codes 64 to 71 will be printed on the screen in the colors previously outlined. This particular set (5) includes capital letters A through G. If you wanted all the capital letters in the character set to be displayed on the screen in the same color, several Call Color statements would be necessary. All of the capital letters are included in character sets 5, 6, 7, and 8 so four Call Color subprograms would do the trick.

## SCREEN

The Screen subprogram is very much like the Color subprogram, except it is used to specify the color of the screen itself. This is the palette upon which the characters are written. The same color code chart used with the Color subprogram applies to the Screen subprogram. The following program segment shows how the Screen subprogram is used with the previous Call Color subprogram:

```
10 CALL CLEAR
20 CALL SCREEN(11)
30 CALL COLOR(5,7,13)
```

This determines that the screen background color will be dark yellow (11) and that the character foreground color will be dark red with a dark green background color. Here, you have used lines 20 and 30 to control the coloration of three different screen elements: the screen itself, the character foreground, and the character background.

With these subprograms, you can highlight your displays, whether they be in alphanumeric form (text mode) or in pure graphics form. By changing the screen background colors, along with the character foreground and background colors, you can cause certain portions of a text display to be highlighted in comparison with the rest. You can also produce a myriad of multicolored images on the screen that can include kaleidoscopes and even fairly detailed pictures.

## ANIMATION

The HCHAR and VCHAR subprograms can be used to produce on-screen animation, or movement.

Animation or movement is created by drawing an image on the screen in one location, erasing it, and then drawing it again at another location on the screen. If this is done rapidly enough, you don't really see the erasure process and it appears as though the object is moving instead of being written, erased, and then written again.

The program shown in Fig. 5-11 displays numbers at the center of the screen and causes them to count upward, giving the impression of motion. All that's really happening is that one number is printed; then it is written over by the next number in the sequence. This process is

```

10 CALL CLEAR
20 FOR X = 48 TO 57
30 CALL HCHAR (12,15,X)
40 NEXT X

```

Fig. 5-11. A program to display numbers at the center of the screen.

continued until the program is over. This type of procedure can be used to produce the effect of on-screen motion from left to right, bottom to top, and/or vice versa. Here's how the program in Fig. 5-11 works.

Line 10 clears the screen, and a For-Next loop is entered in line 20. This causes X to count from 48 to 57 in steps of 1. These numbers represent ASCII codes in the Call HCHAR subprogram in line 30. The numbers 48 to 57 represent the ASCII codes for the numbers 0 to 9. During the first cycling of the loop, a 48 is output from the loop. This value of X is inserted into the Call HCHAR statement in line 30 at the character position. Therefore, the character represented by ASCII code 48 is displayed at screen position 12,15. This character is 0. When the value of X is equal to 49 during the second cycle of the loop, a 1 will be displayed at the screen position where the 0 formerly appeared. This will continue until the loop counts to 57 and times out. The program then ends. This won't take very long, so you may wish to add another line to set up an endless loop, such as:

```

50 GOTO 20

```

This causes the program to run over and over again until manually halted.

Remember that the loop numbers 48 to 57 represent machine characters specified by ASCII character codes. Only the characters ASCII codes 48 through 57 represented appear on the screen. By changing line 20 to

```

20 FOR X = 65 TO 90

```

the capital letters A through Z appear.

In this animation program movement was confined to a single character block. It is simple to produce movement of characters from one point to another on the display screen. Let's start with the program shown in Fig. 5-12. With a few modifications this program will produce animation.

After the screen is cleared in line 10, a For-Next loop is set up to count from 3 to 20 in steps of 1. Within the loop at line 30 is a Call HCHAR subprogram, which uses the value of X in the horizontal or column designator position. Line 30 tells the machine to print ASCII character 79 (O) at position 12,X. This means during the first cycle of the loop, the letter O will be printed at position 12,3; then at 12,4 during the next cycle, and so on, until 12,20 is reached and the loop times out. The result is 18 capital letter Os printed horizontally on the screen from position 12,3 to 12,20. All 18 appear on the screen at the same time, but you do see movement as each letter is printed in turn.

```

10 CALL CLEAR
20 FOR X = 3 TO 20
30 CALL HCHAR (12,X,79)
40 NEXT X

```

Fig. 5-12. A simple animation routine.

You could accomplish the same thing with a single program line, such as:

```
10 CALL HCHAR(12,3,79,18)
```

This would display 18 ASCII characters identified by the number 79 horizontally on the screen starting at position 12,3.

By using a For-Next loop we can set up some true animation. With one modification, a single letter (O) will travel from the left side of the screen to the right. The program is shown in Fig. 5-13. The addition is found in line 25. It's the Call Clear subprogram, which clears the screen before printing the letter O in its new position.

Here's how it works: As soon as the For-Next loop is entered, the screen is cleared. The letter O is then printed at position 12,3. The loop cycles once more, and the screen is erased again by line 25. Then the letter O is printed at the next screen position (12,4). This write, erase, and write again sequence continues until the loop times out. The overall result is that of a single letter moving from left to right on the screen.

```
10 CALL  
20 FOR X = 3 TO 20  
25 CALL CLEAR  
30 CALL HCHAR(12,X,79)  
40 NEXT X
```

Fig. 5-13. A modification to Fig. 5-12 produces true animation.

```
10 CALL CLEAR  
20 FOR X = 1 TO 30 STEP 5  
30 CALL CLEAR  
40 CALL HCHAR(12,X,79)  
50 NEXT X
```

Fig. 5-14. Animation program.

Try the program in Fig. 5-14 to make the letter travel all the way across the screen and in bigger jumps.

This program is almost identical to the previous one, but the coordinates specified by the For-Next loop have been modified, and the count is now in steps of 5. The first letter will be printed at position 12,1. The next will be printed at position 12,6; then 12,11, etc. The character will travel faster and in bigger jumps. You can repeat this process over and over again by adding

```
60 GOTO 20
```

This establishes an endless loop and the capital letter O will continue to race across the screen.

The program shown in Fig. 5-15 uses a trick learned earlier that causes a solid block character to race from one side of the screen to the other. Line 20 establishes the character with a Call CHAR subprogram. It assigns our new character to ASCII code 33. This character is represented by the hexadecimal code (FFFFFFFFFFFFFFFF), which fills in the character block completely. The For-Next

```

10 CALL CLEAR
20 CALL CHAR(33,"FFFFFFFFFFFFFFFF")
30 FOR X = 1 TO 32
40 CALL CLEAR
50 CALL HCHAR(12,X,33)
60 NEXT X

```

Fig. 5-15. Program to animate a solid block character.

loop in line 30 is followed by a Call Clear that is also part of the loop. The next loop instruction causes our new character to be printed on the screen at various locations using Call HCHAR.

When this program is run, the block will emerge from the left side of your screen, travel to the right side, and the program will then end. This is exactly what happened with the letter O, only substituting our filled-in block character.

This left to right travel is getting rather boring, so the program shown in Fig. 5-16 reverses it. The only change is found in line 30 where the For-Next loop counts from 32 to 1 instead of from 1 to 32. Loops can count up or down. However, if the starting value is more than the ending value, you must include the Step command, which will be a negative number. In this case, the -1 indicates that the loop is to count from 32 to 1 in steps of -1. If we wanted to have a loop take larger steps, we might use -5. Regardless of what step is specified, it must be given as a negative number in order to count from a high number to a lower one.

```

10 CALL CLEAR
20 CALL CHAR(33,"FFFFFFFFFFFFFFFF")
30 FOR X = 32 TO 1 STEP -1
40 CALL CLEAR
50 CALL HCHAR(12,X,33)
60 NEXT X

```

Fig. 5-16. This program reverses the travel of the block.

When this program is run, the block will first appear at screen position 12,32. The next position will be 12,31, and so forth until screen position 12,1 is reached. The program then ends. The result is that instead of moving from left to right on the screen, this new program causes the block to move from right to left.

Let's combine the left to right program with the one that moves the square from right to left. The program is shown in Fig. 5-17. Lines 10 through 60 are identical to the first program, and lines 70 through 100 are identical to the second program lines starting with the For-Next loop. It is not necessary to redefine character 33, since this was done for both loops in line 20. Once a character is defined with a Call CHAR subprogram, the character will remain in effect whenever called for in any other part of the program. The For-Next loop established in line 30 assigns X the values of from 1 to 32. When this loop times out, X is equal to 32. Line 70 is then executed, which establishes another loop, still using the variable X. Line 70 reassigns X from its former value to a value of from 32 to 1. Line 110 sets up an endless loop



```

10 CALL CLEAR
20 CALL CHAR(33,"FFFFFFFFFFFFFFFF")
30 FOR X = 1 TO 32
40 CALL CLEAR
50 CALL HCHAR(12,X,33)
60 NEXT X
70 FOR X = 32 TO 1 STEP -1
80 CALL CLEAR
90 CALL HCHAR(12,X,33)
100 NEXT X
110 GOTO 30

```

Fig. 5-17. Program to cause block to travel left to right and then right to left.

by branching back to line 30 after the second loop times out.

When this program is run, the block character moves from the left center of the screen to the right center. It then moves from right center to left center. This process continues until the program is manually halted.

We know that the program is really printing a multitude of characters at different positions on the screen, but erasing each old one before a new one is generated. The viewer seems to see a single cube in motion, but we know that the motion is really made up of a long series of *separate* blocks.

## SOUND

The TI-99/4A has a Sound subprogram

that can be used to generate a wide range of audio tones and a nice selection of audio sound effects. Most video game programs depend heavily on sound effects to make their displays and competitions more realistic.

Like the other subprograms, this one is used with the Call statement. You can produce 3 simultaneous tones. Each Call Sound statement must include the desired duration, frequency, and volume.

Duration is given in milliseconds (1/1000 of a second) and can range from 1 to 4250. One second is equal to a 1000 milliseconds. The longest any single tone can be held is 4.25 seconds, or 4250 milliseconds.

The frequency of the tone must follow the duration command. If the frequency is to be a tone or musical note, the number must be anywhere from 110 to 44733 Hertz. The numbers represent frequency in Hertz (cycles per second). Tones above 44,733 Hertz (44.733 kilohertz) falls well above the human hearing range and will not be detected.

If you want to generate a noise or sound effect, specify any number from -1 to -8 for frequency. The noise produced by the TI-99/4A falls into two categories, *white noise* or *periodic noise*. You will have to test these sounds with the computer yourself. Some sound like motors running, and others offer "space" sounds, etc.

The last parameter that must be specified is volume. Volume is represented by any number from 0 to 30. Zero represents the loudest output and 30 is the softest. The following program line generates a 1000-Hertz tone for approximately  $4\frac{1}{4}$  seconds at the loudest volume possible:

## CALL SOUND(4250,1000,0)

The number 4250 determines the length of the tone. The value 1000 determines the frequency, and 0 determines the volume.

Figure 5-18 shows a program that generates all 8 noises or sound effects available with the Call Sound subprogram. The first sound generated is represented by -1, while the next sound is -2, and so on, up to -8. Each sound is held for a little over 4 seconds. A For-Next loop is used to sequentially select the noise numbers that are fed to the frequency portion of the subprogram.

When this program is run, a For-Next loop is entered that encloses a Call Sound subprogram. The value of X is from 1 to 8. These are the values of the noise numbers. The noise specification numbers must have negative values. A value of minus X is specified in the frequency section of the Call Sound subprogram contained in the For-Next loop. This line tells the computer to output for 4.25 seconds the noise represented by the negative value of X. The 0 indicates that the noise is to be output at maximum volume. If the minus sign is not placed before the X, you will get an error message, because the lowest tone number that may be used is 110. When the program first runs,

you hear the noise generated by command -1. The next noise is that of -2, and so on, until the loop times out at a value of 8. Each positive value of X is converted to a minus value by the minus sign preceding the X variable in line 20.

If you want to generate a multitude of tones, try the program shown in Fig. 5-19. This one is similar to the noise-generating program, but the value of X is from 110 to 2010 in steps of 100. This time, X is inserted without the minus sign, since the numbers representing tones must always be positive and equal to or more than 110. They must also be less than 44733.

When the program is executed, the first tone output will be 110 Hertz. The next tone will be 210, then 310, and so on, until a maximum frequency of 2010 is reached. I shortened the duration command in line 20 to hold each tone for about half a second, if you use a long duration command here, the program can take several minutes to run.

### KEY

Another subprogram useful in maintaining control over the movement of graphic images is the Call Key subprogram. It lets you transfer one character from the keyboard directly to the program. This may sound similar to an Input statement, but it's not. When an

```
10 FOR X = 1 TO 8  
  
20 CALL SOUND(4250,-X,0)  
  
30 NEXT X
```

Fig. 5-18. Program to generate all eight noises using Call SOUND subprogram.

```
10 FOR X = 110 TO 2010 STEP 100  
  
20 CALL SOUND(500,X,0)  
  
30 NEXT X
```

Fig. 5-19. Program to produce a multitude of musical terms.

Input statement is used, program execution is halted until something is input from the keyboard and Enter is pressed. Using the Call Key subprogram, your program continues to execute in a certain manner until a key is pressed. When this occurs, there is usually a branch to another portion of the program and the program runs in a different way. It is not necessary to press Enter after striking the key. Once you "arm" a key, the computer is constantly monitoring the key's status. As soon as the status changes (when the key is pressed), the computer reads this condition and brings about the required branch.

The Call Key subprogram is followed by several specifications in parentheses. The first is the key unit. This can be any number from 0 to 5. A key unit of 0 activates any key on the console. A key unit of 1 activates only the keys on the left side of the keyboard. These are keys 1 through 5 on the top row, Q through T on the second row, A through G on the third row, and on the fourth row, Z through B. A key unit of 2 activates the remaining keys on the right side of the keyboard. Key units 3, 4, and 5 provide specific modes for the keyboard.

Figure 5-20 shows how the Call Key subprogram might be used to provide console control during a program run. This is similar to a previous graphics program discussed in this chapter. A block character moves from left to right across the top of the screen. This is set up by the For-Next loop beginning in line 30 and ending at line 90. The only thing unusual about this loop is in lines 60 through 80. In line 60, the Call Key subprogram is used to read the keyboard. R is the status factor and the third element of the Call Key subprogram. It's called

```

10  CALL CLEAR
20  CALL CHAR(33,"FFFFFFFFFFFFFFFF")
30  FOR X = 1 TO 32
40  CALL CLEAR
50  CALL HCHAR(2,X,33)
60  CALL KEY(0,KEY,R)
70  IF R = 0 THEN 90
80  IF R = 1 THEN 110
90  NEXT X
100 GOTO 30
110 PRINT "PROGRAM IS OVER"
120 END

```

Fig. 5-20. Use of Call Key subprogram.

the status bit, because it assumes the value or status of the keyboard. When no key is pressed, R has a value of 0. When a key is pressed, the value of R is 1. Lines 70 and 80 test for the condition of R. In line 70, if R is equal to 0, there is a branch to line 90, which simply causes the loop to cycle again. If R is equal to 1, this is detected in line 80, and there is a branch to line 110. When such a branch occurs, the moving character will freeze on the screen and the message "PROGRAM IS OVER" will be displayed.

The Call Key subprogram is often used in

text mode programming in place of Input statements. The program in Fig. 5-21 demonstrates this use.

This program is typical of the introductory lines of many game programs. This is only a sample used to demonstrate this use of Call Key and is not a workable program in itself.

Lines 10 through 120 print game instructions on the monitor screen. As soon as the instructions are printed on the screen, nothing further occurs until the operator presses any key on the keyboard. A screen prompt appears in line 150 and tells the operator to "press any key to continue." In line 140, the Call Key subpro-

```
10 CALL CLEAR
20 PRINT "THIS IS AN INTRODUCTION TO A NEW PROGRAM CALLED MOTORCADE"
30 PRINT
40 PRINT "WHERE YOU ACTUALLY DO THE DRIVING. THE GAME IS VERY SIMPLE TO PLAY"
50 PRINT
60 PRINT "ALONE OR WITH A FRIEND. THE TOP ROW OF KEYS CONTROL HORIZONTAL"
70 PRINT
80 PRINT "DIRECTION. THE SPACE BAR CONTROLS SPEED. THE OBJECT OF THE GAME"
90 PRINT
100 PRINT "IS TO COMPLETE TEN LAPS WITHOUT STRIKING AN OBSTACLE OR ANOTHER"
110 PRINT
120 PRINT "AUTOMOBILE. GOOD LUCK!!!"
130 PRINT
140 CALL KEY(0,KEY,R)
150 PRINT "PRESS ANY KEY TO CONTINUE"
160 IF R = 0 THEN 140
170 IF R = 1 THEN 500
```

Fig. 5-21. Using Call Key instead of Input.

gram is used. The 0 designator has been incorporated so that the entire keyboard is read. Line 160 brings about a branch to line 140 as long as R is equal to 0. It will be equal to this value as long as no key is pressed. This effectively sets up an endless loop within lines 140, 150, and 160. When a key is pressed, variable R will be equal to 1 and there will be a branch to another part of the program. This is detected in line 170. The branch to line 500 occurs when R is equal to 1. This fictitious line is used to represent the actual game portion of the program.

### **A TRUE GAME PROGRAM**

The subprograms offered on the basic TI-99/4A computer combined with the statements, commands, and functions in TI BASIC give you the tools necessary to program your own video games.

To give you an example of how most of the subprograms studied in this chapter can be combined into a game, look at the Shooting Gallery program shown in Fig. 5-22. It lets you try to "blow away" a little graphic man who runs across the top of the screen. Your weapon is a graphic pen that shoots square projectiles. Each time you press any key on the keyboard, your cannon will fire. The cannon will always fire its projectile to one position on the screen. An element of skill is involved since you must fire the cannon when the running figure is at the correct position to bring about a hit.

The game includes sound effects to make it more interesting. It should only take you a few minutes to enter this program to your machine, and you can begin playing as soon as the debugging procedure is complete.

Lines 10 through 40 use REM statements to title the program and give some basic details about the memory requirements and the language used. In line 50, a Call Color statement is used to color the moving characters with a dark red foreground and a black background. The screen is cleared in line 60, while lines 70 and 80 develop our on-screen characters.

These are produced with Call CHAR statements. In line 70, the graphic "target" is created from a single screen block. This will show a little man with arms extended. Line 80 prints a square block character that represents the projectile.

Line 90 brings sound effects into our program. It uses the Call Sound subprogram and causes the computer to generate a noise (-8) for approximately four seconds. When the Call Sound subprogram is used, the program lines following it are executed at the same time the sound is being heard. The sound data is fed to a buffer. This allows for simultaneous output of sound and execution of remaining program lines.

Lines 100 through 160 form a For-Next loop causing the character set in line 70 to run from left to right across the top of the screen. This is done by the HCHAR subprogram in line 120, whose horizontal coordinates are derived from the value of X. The Call Key subprogram is found in line 130, along with the test lines to bring about appropriate branches in lines 140 and 150. It takes this loop about four seconds to time out. The noise follows the little man across the screen and stops when he reaches the right side. When this occurs, the loop is timed out, and line 170 is executed. This is identical to the Sound subprogram in line 90,

```

10 REM SHOOTING GALLERY
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 1/26/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM MEMORY USED TO RUN THIS PROGRAM I
S 768 BYTES
50 CALL COLOR(1,7,2)
60 CALL CLEAR
70 CALL CHAR(33,"1818FF3C3C3C2424")
80 CALL CHAR(34,"FFFFFFFFFFFFFFFF")
90 CALL SOUND(4000,-8,0)
100 FOR X=1 TO 32
110 CALL CLEAR
120 CALL HCHAR(2,X,33)
130 CALL KEY(0,KEY,R)
140 IF R=0 THEN 160
150 IF R=1 THEN 270
160 NEXT X
170 CALL SOUND(4000,-8,0)
180 FOR X=32 TO 1 STEP -1
190 CALL KEY(0,KEY,R)
200 IF R=0 THEN 220
210 IF R=1 THEN 270
220 CALL CLEAR
230 CALL HCHAR(2,X,33)
240 NEXT X
250 CALL SOUND(4000,-8,0)
260 GOTO 100
270 FOR Y=22 TO 1 STEP -5
280 CALL CLEAR
290 CALL HCHAR(Y,15,34)
300 NEXT Y
310 IF X=15 THEN 330
320 GOTO 160
330 CALL CLEAR
340 CALL SOUND(4000,1000,0)
350 PRINT "DIRECT HIT"
360 FOR Q=1 TO 800
370 NEXT Q
380 GOTO 60

```

Fig. 5-22. Shooting Gallery program.

and sets up the sound effect for the next loop. Lines 180 through 240 cause the little man to now run from right to left across the top of the screen. This loop is identical to the previous one, except for the reverse order for the count value of X. Note the Step - 1 command used at the end of line 180.

When this loop times out, there is another Call Sound subprogram (line 250) and then a branch to line 100, where the entire sequence begins again. As long as no key is pressed after the program begins running, the target will run back and forth across the screen while sound effects occur.

However, if a key is pressed, the value of R is equal to 1, and there will be a branch to line 270. Two Call Key subprograms are used in lines 130 and 190 to make sure you have control when either directional loop is executing.

When a key is pressed, the branch to line 270 causes another For-Next loop to be executed. This is another negative step loop, determining the vertical coordinates for the HCHAR subprogram in line 290. The loop causes character 34 (the projectile) to move from the bottom of the screen (position 22) to the top of the screen in steps of 5 screen places.

Line 310 tests for a hit. The projectile is fired from horizontal position (column) 15. It starts at 22,15, then rises to 17,15, then 12,15, etc. Line 310 states that If X is equal to 15, Then branch to line 330. Here, the variable X is the last horizontal position of the little man. This was his position when the loop controlling his movement was exited by pressing a key. If the little man is in the 15th horizontal position, the projectile strikes him. When the program

branches to line 330, there is a musical tone set up by the Call Sound subprogram in line 340. The screen then displays **DIRECT HIT**.

The For-Next loop in lines 360 and 370 are a time delay loop. These two lines cause the computer to count from 1 to 800. This takes a few seconds and gives the player time to note the "DIRECT HIT" prompt on the screen. As soon as this loop times out, a branch to line 60 starts the target running again.

Going back to line 310, If X is not equal to 15, Then there is a branch to line 160, that starts the program near its beginning. Each time you press a key, a single shot will be fired. It takes a bit of practice to get anywhere close to a perfect score.

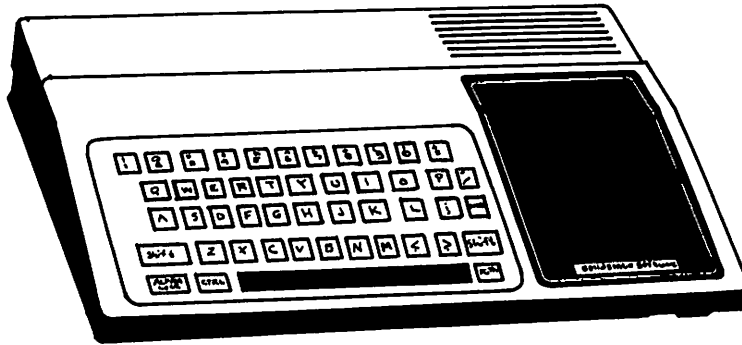
This is a simple computer game program. Through a few more programming steps, a great deal more sophistication could be built in, but the program as shown is sufficient as an example of how to begin. Don't attempt to do too much too soon. The best thing to do is come up with a game idea and then program it in its simplest possible form. Once you have the basics working properly, add features to expand the enjoyment of the game.

I hope that the discussions in the last two chapters have provided you with a solid grounding in TI BASIC. If you feel there are areas which you do not fully understand, go back and re-read the discussions surrounding the subjects. Not all of the attributes of TI BASIC have been discussed in detail in the last two chapters. All other statements, functions, and commands work along similar format lines. You will be able to understand how they are used by referring to Chapter 3 and to the manuals supplied by TI.





## Chapter 6



# Error Messages

An error message results when the computer detects an error in one of its programs. It appears as a message or prompt on the monitor screen. Error messages are taken for granted today by most computer operators, but they were not available on early microcomputers. At that time, following several hours of typing in a program, you could discover that the program would not run and have no clue as to what the program might be. All modern microcomputers have built-in check functions which will test each program line and let you know if there is a problem, and give you a good clue as to what the problem is so you can make immediate corrections.

### TYPES OF ERROR MESSAGES

The TI-99/4A prints three types of error messages and performs three types of checks.

The first check is made as soon as a program line is committed to memory, as soon as you have entered the line by pressing Enter. If something is incorrect the computer will emit a beep and print the error message.

The second check is made when a symbol table is generated. This occurs when a program has been completely input and a Run is attempted. Before the program begins to run, the computer scans it to establish an area in its memory where the variables, arrays, and functions can be stored. During this scanning process, the computer can recognize some errors that could not be detected during first line-by-line check. When this second type of error is discovered, the number of the line containing the error is printed as part of the error message. Occasionally the error is not a part of the line named, but has to do with using a variable

in that line which was not correctly assigned in another line. At the line named by the error message the computer can continue no further.

The third type of check is performed during the program run. When a program is run, the computer may encounter statements that it cannot perform. An error message will then be printed, and usually the program run will be terminated. The number of the line containing the error will be printed as part of the error message.

There is a fourth automatic computer check, but it has to do with the information retrieval from memory files and will not be discussed here.

The first check prevents obvious erroneous line entry errors which the machine can immediately detect. Try typing **PERFORM** immediately following a line number. This word has no meaning as a statement, command, or function in TI BASIC; you will immediately hear a beep and an error message will indicate that you entered an incorrect statement.

The second type of error is encountered when, for instance, you mismatch the number of For and Next statements in a program. Assume you use For three times, but use only two Next statements. This won't be detected by the first error check.

The third type of programming error is discovered during the program run and can be artificially set up by including a line which commands **CALL SOUPD** instead of **CALL SOUND**. This type of error message would include the program line, along with the clue **BAD NAME** indicating that you probably misspelled a word somewhere.

The following section of this chapter lists and explains the various error messages. With this information it should be easy for you to debug any program that will not run due to some type of input error.

Keep in mind that it's possible to enter a program in correct form, as far as the computer is concerned and still not have it to do what you want it to do. You might write a program to print a mathematical answer on the screen, but if you leave out the Print statement, you will never see the answer. The automatic check procedure will be of no value in debugging this type of problem. The only thing the computer checks for is "poor grammar." The computer cannot tolerate misspelled statements, functions, or commands or any of a number of other possible mistakes.

## ENTERING ERRORS

The following error messages are the type that are generated as soon as the faulty line is entered. These errors must be corrected before the computer will accept that line.

**Bad Line Number** This means a line number is outside machine limits or a line number has been referenced in a branch statement which is outside these limits. The lowest legal value line number is 1. The highest line number that can be used with this micro is 32767. This same error message will occasionally arise after executing a Resequene command that necessitates the generation of a line number greater than 32767.

**Bad Name** A variable name has been used which contains more than 15 characters. Most variable names will consist of a single letter or just a few letters.

**Can't Do That** This error message is generated by many errors. In direct mode, it means you used a program statement as a command, and the statement is not valid in direct mode. These statements include For, GOTO, GOSUB, If and others. This error message may also be generated when you attempt to use direct mode commands as part of a program. These commands include Edit, List, New, Run, Number, and several others. One other way this error message can be generated is when you try to List, Run, or Save a program not in memory.

**Incorrect Statement** This error message can be generated for several reasons. It might mean the command keyword is not the first word in a line, but most often it will mean that a string in quotation marks following a Print statement has an opening quotation mark but no closing quotation mark, or vice-versa. The message will also be generated when you sequentially enter two variable names with no valid separator, such as a colon, comma, or quotation mark. It may also mean you have used invalid print separators between numbers while using the List, Number, or RES commands. This is one of the most common error messages seen.

**Line Too Long** This means you tried to enter a program line which contained too many characters for the input buffer. Maximum length is three lines. This can be corrected by stopping a phrase in quotation marks following a print statement at the end of the third line and then numbering another program line beginning with another Print statement followed by the rest of the phrase, in quotation marks.

**Memory Full** This means the avail-

able memory (RAM) has been exceeded by the line you just added. You can try to remove unnecessary portions of your program or add a memory expansion option.

## SYMBOL TABLE ERRORS

Symbol table error messages are generated immediately after the Run command is entered but before the program is executed.

**Bad Value** This may mean the dimensions established for an array are greater than 32767 (the machine maximum). It may also mean a dimension for an array is 0 when Option Base = 1.

**Can't Do That** When this error message occurs during the symbol table error check mode, it means you used more than one Option Base statement in your program, or that the Option Base statement has a higher line number than an array definition.

**For-Next Error** This means you used a For statement without a Next or a Next without a For. It may also mean you did not properly identify the loop variable when the Next statement was entered. Check your For-Next loops carefully and you will certainly discover the problem. This error sometimes occurs when you improperly branch into a For-Next loop from another portion of the program.

**Incorrect Statement** This means a DIM statement has no dimensions or more than three dimensions, or even that you've used a letter instead of a number to define the DIM statement. It may also mean you used Option Base without a 0 or a 1, which must follow it. You may even have used Option without the word Base.

**Memory Full** This means you have

specified an array size too large for resident memory or that your program has taken up so much RAM there's not enough memory left to allocate a variable or function.

**Name Conflict** This is a common error message, especially when a program uses a number of arrays. It usually means the same name has been given to more than one array. It may also mean you assigned a name to an array and then later assigned it to a simple variable.

## PROGRAM RUN ERRORS

These encompass the largest area where error messages are generated. While the first two checks will catch many errors, the third check catches most of them. Any time an error message is generated in this mode, the number of the line where the problem has occurred will follow it.

**Bad Argument** This has to do with functions, such as INT, VAL, etc. If the VAL function is used (and indicated by the error line number), this probably means the string expression is not a valid representation of a numeric constant, or that the string expression has a zero length. It may also mean you tried to use one of these functions with a bad argument. In the case of VAL, this means the string sign (\$) may have been omitted from the argument variable.

**Bad Line Number** This probably means you tried to branch to a line that doesn't exist.

**Bad Name** This means a subprogram name in a Call statement is illegal. You probably mistyped the second portion of the Call statement. Check for a spelling error.

**Bad Subscript** Often this means the subscript 0 is used when an Option Base of 1 is specified. It may also mean a subscript has a value greater than the specified dimensions of an array.

**Bad Value** This can mean many things. It is often generated due to an incorrect For-To-Step statement. A Bad Value message will occur when you include the word Step in a statement but fail to follow it with a number, or when the step value is 0. This can also mean the character code is out of range when using a CHAR statement, or you've included a bad argument number in the CHR\$ function. You might also watch for an improper row or column number in a GCHAR, HCHAR, or VCHAR statement. Also, a color-set-number may be out of range in a Color statement. You may have also used the Tab function which included the value of a character position in an amount greater than 32767. A Screen statement may have been used with a color code that is out of range. In the Sound statement, this message can occur when the duration, frequency, and/or volume specifications are outside the established ranges. This message indicates a numerical value error which is generally the result of a typographical error when entering via the keyboard. The error should be obvious in the line listed with this error statement.

**Can't Do That** This error message is common to all three types of checks. When it is printed following a Run command, it's usually a sign of a branch error or an error occurring within a loop. More precisely, you used a statement requiring a matching statement, but didn't include the match. This applies to GOSUB and Return, as well as to For-Next.

The same error message is generated when a Return is encountered with no previous GOSUB.

**Data Error** When this error message occurs, it often means you forgot to put commas between items in a Data statement. It may also mean you ran out of Data statements or that you used a Read statement with no Data statement remaining.

**File Error** When this error occurs, it may mean you tried to close a file you hadn't opened or you tried to open a file that was already opened. It's also an indication you have attempted to Read or Write data to a file improperly.

**Incorrect Statement** This can be caused by many errors, but almost always points to improper use of BASIC. It can occur when you omit a comma from a line using statements that require commas. You may have used mathematic operators such as plus, minus, multiply, etc., not followed by numeric expressions. You may even have used string variables in mathematic operations. In short, you either left something out or put too much in. Other ways of generating this message include forgetting to follow the statement For with a numeric variable, such as **FOR A\$ = 1 TO 20**. This is a string variable, and the For statement will work only with a numeric variable. You may also have forgotten to include the word To, as in **X = 1 - 5**, instead of **FOR X = 1 TO 5**. You may also have used If without Then, On without GOSUB, or used an illegal word or character following a Return statement. You may also have forgotten to include the control variable following the Next statement.

**Input Error** This often means that keyboard input asked for during the middle of a program is too long for the keyboard buffer. Also, an Input statement may have requested a numeric entry via the keyboard and none was typed in. This is often seen when the programmer attempts to halt execution of a program until the Enter key is pressed by the user, using **INPUT A** rather than **INPUT A\$**. This same error message will occur when a numeric input is needed, but a string input is supplied via the keyboard.

**I/O Error** Most often this means there is a problem with a peripheral device or at least a problem communicating with it. This error message can occur when you try to save or retrieve programs from cassette storage when the recorder is not on, or the cable is not attached. It can also occur when you try to write information to a storage medium which is write-protected. If you use an illegal Input/Output command, the same message will be generated. It may also mean you tried to retrieve a file from storage which is not found in storage because of an incorrect cassette, disk, or even an incorrect file name.

When an I/O error occurs, a two-digit error code is displayed on the screen: **I/O ERROR 14 IN 400**. This means program line 400 contains an error and the number 14 will help to indicate what type of error. In this case, the 14 is really two numbers. The 1 indicates the operation which was attempted, and the 4 indicates the type of error. Either number may be given a value from 0 to 7. Figure 6-1 provides a key for the error message numbers.

**Memory Full** This may mean a program contains too many subroutine branches

| First Number  |  |
|---------------|--|
| Number        | Operation  |
| 0             | OPEN   |
| 1             | CLOSE  |
| 2             | INPUT  |
| 3             | PRINT  |
| 4             | RESTORE  |
| 5             | OLD  |
| 6             | SAVE   |
| 7             | DELETE   |
| Second Number |  |
| Number        | Error  |
| 0             | Invalid device or filename                                   |
| 1             | Write Protect in effect                                      |
| 2             | Bad open attribute   |
| 3             | Illegal operation  |
| 4             | Insufficient space on storage medium                         |
| 5             | File past end  |
| 6             | Peripheral device not installed, defective, or not activated |
| 7             | File does not exist  |

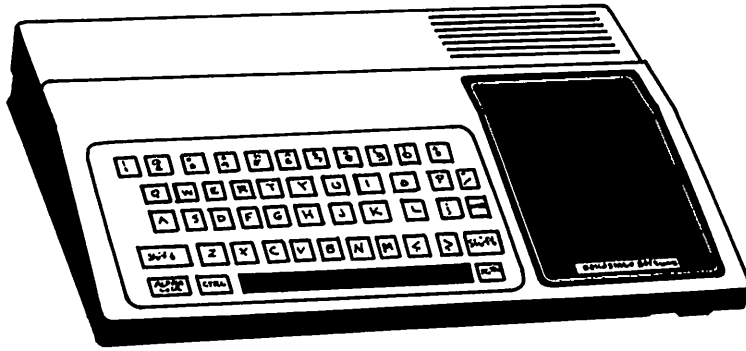
Fig. 6-1. Error message number key.

with no Return command, a relational, string, or numeric expression is too long; or a GOSUB statement branches to its own line number: 20 GOSUB 20.

**Number Too Big** This means a numeric operation produces a number greater than the largest number the micro is capable of handling.

**String-Number Mismatch** This type of error message can be caused by many different factors, such as attempting to assign a number to a string variable, by a non-string value found in a specification requiring a string value, or by a non-numeric file number in an Open, Close, Input, Print, or Restore statement.

## Chapter 7



# The Microprocessor

The heart of any microcomputer is its microprocessor. The TI-99/4A contains a 16-bit microprocessor. While the 16-bit microprocessor is considered a fairly new innovation and one of the main selling points of the IBM Personal Computer, the TI-99/4 computer was using this chip in 1979, long before the IBM PC ever surfaced.

This chapter includes some highly technical overviews of the TMS9900 microprocessor. If you are a beginning programmer, you probably won't understand much of this chapter, which is written especially for the serious programmer who is contemplating writing programs in machine language. Information for this chapter was obtained from Texas Instruments and from *Microprocessor Cookbook* by Michael F. Hordessi, published by TAB Books, Inc.

The TMS9900 microprocessor is a single-chip, 16-bit central processing unit which was produced using NMOS silicon-gate technology. This technology results in higher speeds and TTL input and output levels that operate directly with TTL memories.

The TMS9900 has a 16-bit capability on address and data buses. The buses use a parallel configuration allowing access to 16-bit words in a single cycle. The architecture of this chip is similar to a minicomputer, in that there are no general-purpose registers on the CPU; these registers are housed in memory. All data goes directly from memory to the arithmetic and logic unit, or to special-purpose registers for interrupts or data status and then back to memory. Since all data resides in external memory, register capacity is not limited by on-chip register capacity. This system also

saves time during interrupts, since register data does not have to be saved.

## ARCHITECTURE

Figure 7-1 shows the architecture of the TMS9900. The heart of this chip is the arithmetic and logic unit (ALU). Three internal registers are accessible to the user. The program counter (PC) contains the address of the next instruction. This address is referenced by the processor for fetching the next instruction from memory and is then incremented automatically. The status register is used to contain the present state of the processor, including the interrupt mask level and information for the instruction operation.

The work-space pointer register is a 16-bit register that holds the address of the first general register (RO). A work-space register can hold data or addresses and function as accumulators, address registers, operand registers, or index registers. During program execution, the processor addresses a register in the work-space by adding the register number to the contents of the work-space pointer register and initiating a memory request for the word.

The work-space system is useful when there is a change from one program environment to another. In a conventional register arrangement, at least a part of the file must be stored and reloaded. A memory cycle is used to store or fetch each word.

The TMS9900 accomplished a program environment change by exchanging the program counter, the status register, and work-space pointer register in three store cycles and three fetch cycles. After the exchange, the

work-space pointer holds the starting address of a new 16-word work space for the new routine. Time is also saved when returning to the original program environment.

The TMS9900 interrupt system has the capability for 17 vectored interrupts. Two of these interrupts are predetermined, while the remaining 15 can be determined by the programmer.

The TMS9900 compares the interrupt code with the interrupt mask contained in status register bits 12 to 15. When the level of the incoming interrupt is less than or equal to the interrupt mask, the interrupt is recognized and initiated, following completion of the current instruction. A new WP and PC is fetched from the interrupt vector locations. The previous WP, PC, and status register values are stored in the new work space.

The processor forces the interrupt mask to one less than the level of the interrupt being serviced, except for the zero level interrupt, which loads zero. Only higher priority interrupts can interrupt a service routine. Interrupts are inhibited until the first instruction of the service routine is executed, to save the program linkage if a higher priority interrupt should occur. All interrupt requests remain active until recognized by the processor. The service routines must reset the interrupt requests. When a higher priority interrupt occurs, the processor switches to service it. When that routine is complete, a return instruction is used to complete the processing of the lower priority interrupt. The TMS9900 flowchart is shown in Fig. 7-2.

Instruction execution is made up of four sequences occurring in the control ROM sec-



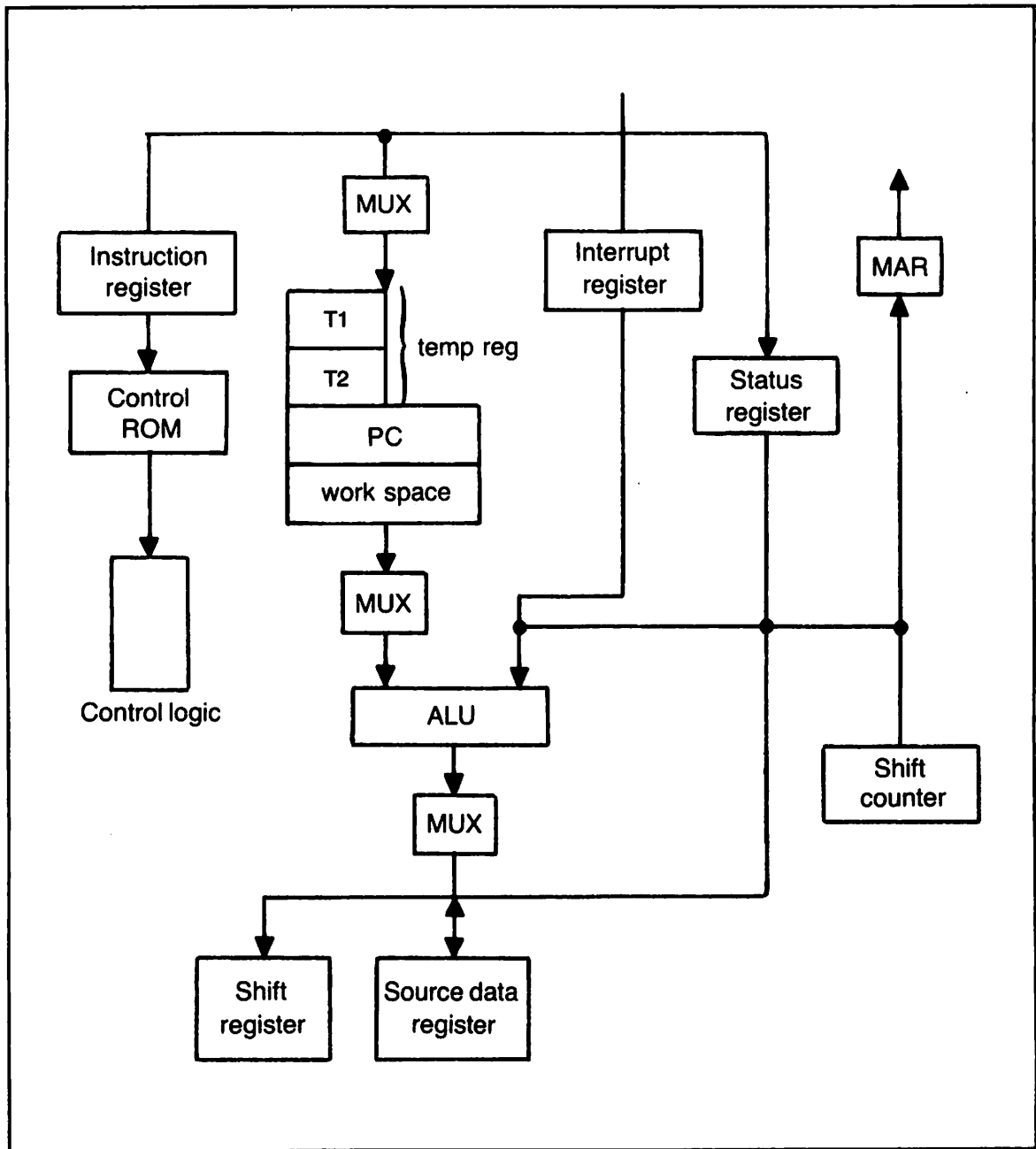
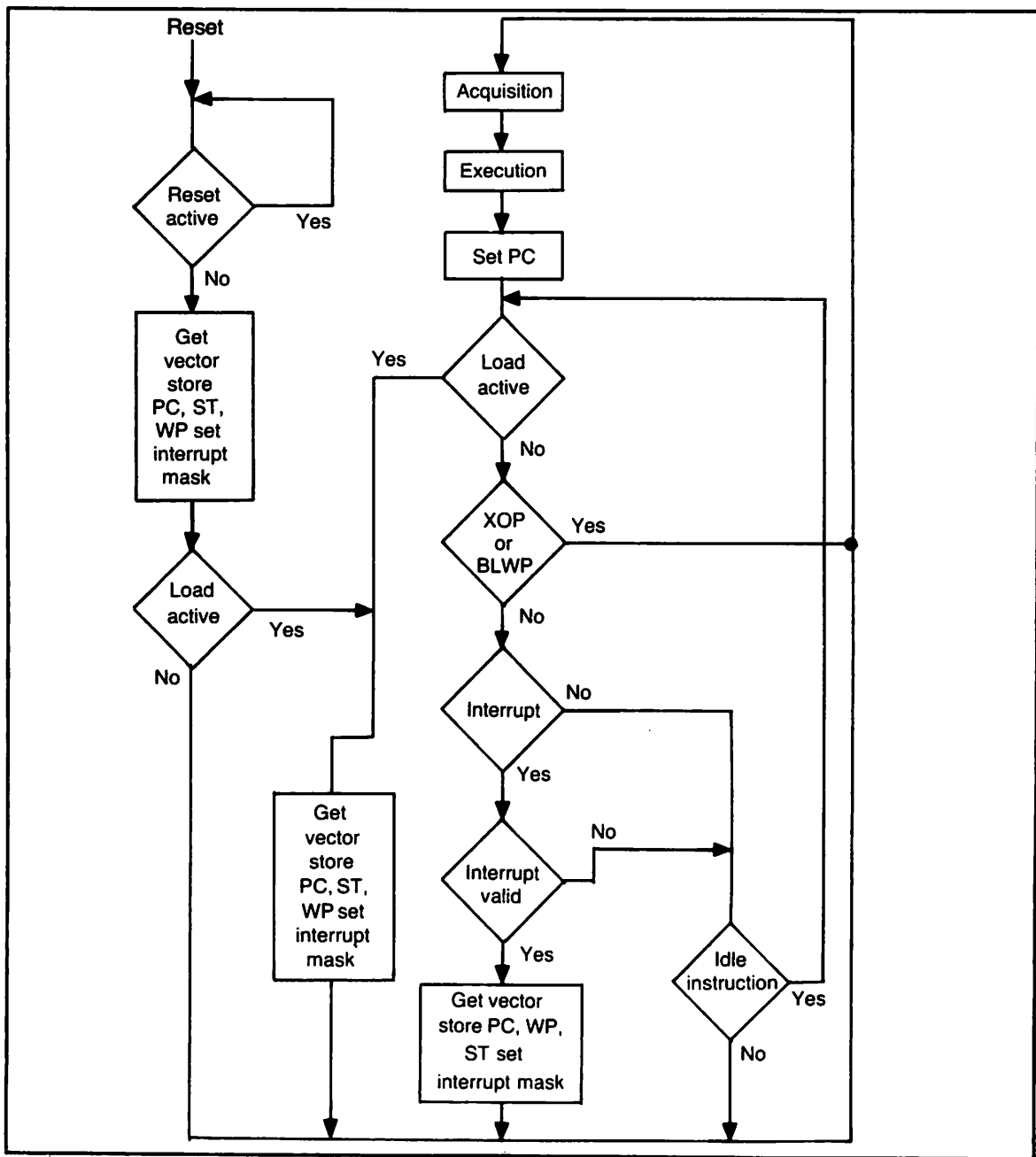


Fig. 7-1. TMS9900 architecture.



tion of the CPU. First there is the instruction acquisition phase, in which the program counter is set and then passed through the ALU and multiplexer to the memory address register (MAR). From a memory-read-cycle, the instruction passes from the data bus to the instruction register, where it initiates the other three sequences

- Source-operand derivation
- Destination-operand derivation
- Instruction execution

The derivation phases are control-ROM sequences that acquire operands based on the instruction and address mode. Operand addresses are generated from the values in the work-space pointer register and the contents of the instruction.

The source data, destination data, and destination address are stored in temporary registers T1 and T2. Instructions are then executed using the source and destination data. The result is stored in an external memory location specified by the destination address.

During the instruction execution phase, the INTREQ and Load lines are checked for interrupts. If the INTREQ line is active, a four-bit code is generated on the interrupt code lines and compared with the status register data to determine the priority for acceptance of the interrupt. If accepted, a vector address is generated by the ALU and sent to the memory address register to initiate the interrupt sequence. If there are no active enabled interrupts, the program counter phase is begun again.

## INSTRUCTIONS

The TMS9900 uses 69 16-bit instruc-

tions. Each of these instructions performs one of the following functions:

- ☐ Arithmetic, logic, comparison, or manipulation operations on data.
- ☐ Control operations.
- ☐ Data transfers between memory and external devices.
- ☐ Loading or storage of the program counter, work-space point, or status registers.

The instruction set for the TMS9900 is shown in Fig. 7-3 in summary form.

The TMS9900 uses a variety of addressing modes to address random memory data, such as program parameters and flag, or formatted memory data such as character strings and data lists. The 9900 addressing modes include register, register indirect, register indirect with auto increment, indexed, immediate, direct, and relative.

In register addressing, a register in memory contains the operand. The address of register R is  $WP+2 \cdot R$ , where WP is the work-space pointer. The work space is always preset at the start of the program as a starting reference.

In register indirect addressing, the operand is found in the memory location whose address lies in a register in memory:

**INSTRUCTION—**

**ADDRESS IN MEMORY—OPERAND**

An example of an indirect address instruction is:

**MOV \*R2,R3 ;move word, R3 = R2**

This instruction loads register R3 with the memory location that has its address in R2.

|                                      |   |                                       |
|--------------------------------------|---|---------------------------------------|
| A : add word                         | JEQ : jump if equal                     | RSET : reset                          |
| AB : add byte                        | JGT : jump if greater than              | RTWP : return workspace pointer       |
| ABS : absolute value                 | JH : jump if high                       | S : subtract word                     |
| AI : add immediate                   | JHE : jump if high or equal             | SB : subtract byte                    |
| ANDI : AND immediate                 | JL : jump if low                        | SBO : set CRU bit to one              |
| B : branch                           | JLE : jump if low or equal              | SBZ : set CRU bit to zero             |
| BL : branch and link                 | JLT : jump if less than                 | SETO : set ones                       |
| BLWP : branch load workspace pointer | JMP : jump unconditional                | SLA : shift left, zero fill           |
| C : compare word                     | JNC : jump if carry clear               | SOC : set ones corresponding, word    |
| CB : compare byte                    | JNE : jump if not equal                 | SOCB : set ones corresponding, byte   |
| CI : compare immediate               | JNO : jump if no overflow               | SRA : shift right, algebraic          |
| CKOF : clock off                     | JOC : jump if carry set                 | SRC : shift right circular            |
| CKON : clock on                      | JOP : jump if odd parity                | SRL : shift right, logarithmic        |
| CLR : clear operand                  | LDCR : load communications register     | STCR : store from CRU                 |
| COC : compare ones corresponding     | LI : load immediate                     | STST : store status register          |
| CZC : compare zeroes corresponding   | LIMI : load interrupt mask immediate    | STWP : store workspace pointer        |
| DEC : decrement by one               | LREX : load ROM and execute             | SWPB : swap bytes                     |
| DECT : decrement by two              | LWPI : load workspace pointer immediate | SZC : set zeroes corresponding, word  |
| DIV : divide                         | MOV : move word                         | SZCB : set zeroes corresponding, byte |
| IDLE : idle                          | MOVB : move byte                        | TB : test CRU bit                     |
| INC : increment by one               | MPY : multiply                          | X : execute                           |
| INCT : increment by two              | NEG : negate, twos complement           | XOP : extended operation              |
| INV : invert, ones complement        | ORI : OR immediate                      | XOR : exclusive OR                    |

Fig. 7-3. The instruction set for the TMS9900.

In the automatic increment mode of register indirect addressing, the general register contains the address of the operand. After acquiring the operand, the general register is incremented by two:

```
MOV *R2+,R3 ;move word, R3 =
              R2 + 2
```

loads R3 with the memory location that has its address in R2. After the move, R2 is incremented by two.

With indexed addressing, the operand is contained in the memory location whose address is a constant (the base address) plus the

contents of a register (the index value). The sum of the base address and the index value results in the effective address for the operand. If the first register (R0) is used, the operand is the base address. To move the contents of a variable (VA) to register R1 code:

```
MOV atVA,R1 ;move word, R1 = VA
```

The instruction does not specify an index value, so the assembler assumes R0 is desired. To add an index value, code:

```
MOV at 12(R1),R2 ;move word, R2 =
                  R1 + 12
```

This instruction loads R2 with the location addressed by the contents of R1 + 12.

In immediate addressing, the word following the instruction contains the operand:

```
LI  R2,>2134 ;load R2 with 2134
           hexadecimal
```

This instruction loads register R2 with the hexadecimal value 2134 which is designated by >.

In direct or symbolic addressing, the word following the instruction contains the address of the operand. This is shown as:

#### INSTRUCTION/LABEL — OPERAND

Relative addressing can be used to obtain destination addresses for most jump instructions in the TMS9900. The relative address is found by multiplying the second byte of the instruction by two and adding the result to the address of the next sequential instruction. The addition is done in two's complement arithmetic, which allows the transfer of control to an address between +258 and -250 locations from the current instruction. Since all instructions are stored as two-byte words, this translates to a transfer of control of a word from +129 to 825 locations of the current instruction. For examples:

```
JMP 12 ;jump unconditional
```

transfers control to the address of the next sequential instruction plus 24 (12\*2).

The 9900 instruction set is much more powerful than many microprocessors. However, since it is word-oriented, memory re-

quirements must be larger, and the 64-pin package tends to increase costs.

Memory access is organized so that all 16-bit memory addresses specify the location of one byte of data. The memory space for a system then becomes 65,536 bytes, which is organized as 32,768 16-bit words. Access to memory is via 15 bits on the memory bus for all 32,768 words. The sixteenth bit is maintained in a register to specify the byte which the processor must use during instruction execution. During byte operations, the unused byte is held, but at the end of an instruction, the two bytes are merged and returned to memory. This allows the instructions to automatically control operations.

Each operational code (op code) is one word long, and if register indirect, indexed, or immediate addressing is used, the constant is located in the word or words that follow the op code. The constant for each source operand is found in the first word following the op code, and the constant for the destination operand is found in the next available word. The instructions can then be one to three words long, or two to six bytes in length. A six-byte instruction is shown below:

```
MOV  at VA2, at VA3 ;move word,
                   VA3 = VA2
```

This instruction transfers the contents of variable VA2 to variable VA3.

When the 9900 addresses a register in the byte mode, it uses the left byte of the register, not the right byte. Whenever the processor references memory, it reads a full word and selects the proper byte for the word.

The 9900 can be used with byte operands

almost as well as full-word operands. For example, to add two bytes, use:

**AB at B1, at B2 ;add, B2 = B2+B1**

This instruction adds the contents of B1 to B2.

The 9900 does not use a stack like other microprocessors for subroutine return addresses. It saves each return address in register R11:

**BL ROUT ;branch and link to**  
**REP \* call ROUT**  
**\***

This instruction saves the address of REP in R11 and transfers control to ROUT. To return from ROUT, a jump is used to the contents of R11: B\*R11. For one subroutine to call another, it must first save the contents of R11. One method involves saving the return address in a general register. Let ROUT1 be one subroutine which calls another subroutine ROUT2, and then code:

**MOV R11,R1 ;move to save return**  
**address**  
**BL ROUT2 ;branch and link to call**  
**\* next subroutine**  
**\***  
**B \*R1 ;branch to exit**

For only a few subroutine levels, this technique is usually the most efficient, but more complex applications may have too many subroutine levels to store all the return addresses in registers. Storage in RAM is used instead, with an instruction like the following for saving the return address:

**MOV R11,at TEMP ;move to save**  
**return address**

To exit the subroutine:

**MOV at TEMP,R11 ;move to get**  
**return**  
**B \*R11 ;branch to exit**

This method uses four words of instruction memory for the exit in addition to the word for the return address. Another method that uses less memory is:

**MOV R11,at EX+2 ;move to save**  
**return in exit**  
**branc-**  
**\***  
**EX B at ;exit**

This routine will save the return address in the second word of the branch instruction and does not require the second move. One problem is that the routine modifies itself so it cannot be used in a ROM.

Another method is to use a stack for saving the return addresses. A software stack can be created in the 9900 by using one of the general registers. For example, use R14 to load the address of the first location. Then, to place an entry on the stack, code:

**MOV R11,\*R14+ ;move to stack R11**

Since the stack pointer is incremented after each storage, the stack moves up instead of down. To retrieve an entry from the stack, code:

**DECT R14 ;decrement by**

```

                                two,R14 = R14-2
MOV  *R14,R11  ;move to get the
                                top entry

```

The software stack can save the general registers for other uses in cases with limited subroutine levels.

Another approach requires the use of the BLWP instruction. BLWP is a subroutine call, but before execution, it resets the work-space pointers. This gives the subroutine a new set of registers for storage without having to store the old registers. The only requirement for BLWP is the additional memory for the call and the new registers.

For restarts and software interrupts, the 9900 offers the XOP instruction. This can be used as a reset or software interrupt. It also allows the passing of parameters. The XOP instruction execution is similar to BLWP, but the target address is determined by the transfer vectors of the instruction. For the 16 possible XOP instructions, the source operand goes to R11 of the new work space as in:

```

XOP  at x,14      ;extended operation 14

```

This instruction performs an extended operation and places the contents of the variable x in R11.

## ROUTINES

In the 9900, the return address of a routine is stored in one of the general registers making passing parameters easily accomplished. For a set of floating point arithmetic routines for addition, subtraction, multiplication,

and division, three parameters are required. The addresses of the parameters will occur after the subroutine call. To calculate  $X1=X2*X3*X4$ , use the routine shown in Fig. 7-4.

In this sequence, the *address* of the parameter is passed rather than the *value*. Lines 1 through 3 are used to store the addresses of the three parameters in the three general registers (R1, R2, and R3). Line 4 is used to set TMP equal to  $X2*X3$ , and line 8 sets  $X1$  equal to  $TMP + X4$ . With the indirect auto increment addressing, there is no need for intermediate incrementing.

To allow subroutines to return results to their calling programs, the general registers can be used if the call is via a branch and link instruction (BL). However, a call using a BLWP or XOP instruction requires a different technique, since the data will be lost if placed in the general registers when control returns to the calling program and the work-space pointer is reset.

Since the general registers are in memory, the following technique can be used. Let R0 and R1 be the old work space. When the processor executes a BLWP instruction, the work-space pointer is saved in R13. Now, code:

|                 |              |
|-----------------|--------------|
| 1. MOV *R11+,R1 | 7. DATA TMP  |
| 2. MOV *R11+,R2 | 8. BL @ FADD |
| 3. MOV *R11+,R3 | 9. DATA TMP  |
| 4. BL @ FMUL    | 10. DATA X4  |
| 5. DATA X2      | 11. DATA X1  |
| 6. DATA X3      |              |

Fig. 7-4. Routine to calculate  $X1 = X2 * X3 * X4$ .

```

MOV  R0,*R13      ;move, set
                   old R0 = new
                   R0
MOV  R1,at 2(R13)  ;move, set
                   old R1 = new
                   R1

```

Since the old register is the same as memory location  $R13+2*i$ , the location is addressed as  $at\ i+i(R13)$ , where  $i$  is the register number. In the first instruction,  $R0$  becomes a special case, since  $at\ 0(R13)$  becomes  $*R13$ .

The 9900 has no decimal-adjust instruction for BCD operations, but one can be created, and BCD numbers are commonly encountered. Figure 7-5 contains a routine for converting  $R1$  to a BCD number in  $R3$ .

Line 1 sets  $R0$  equal to the digit count. Then  $R1$ ,  $R2$  are set equal to the value to be converted and  $R1$  is cleared. A division on line 4 produces  $R1 = \text{value}$  and  $R2 = \text{digit}$ . Shift left on line 5 for the reposition digit, shift right on line 6 to make room for the addition on line 7, then decrement to continue until completed.

The 9900's multiply and divide instructions use unsigned operands, while the rest of the microprocessor uses two's complement operands. A signed two's complement multiply can be formed by noting that if  $X1$  and  $X2$  are two numbers, then the sign of  $X1*X2$  is the Exclusive OR of the signs of  $X1$  and  $X2$ . The

|    |      |           |     |     |         |
|----|------|-----------|-----|-----|---------|
| 1. | LI   | R0,4      | 6.  | SRL | R3,4    |
| 2. | LOOP | MOV R1,R2 | 7.  | A   | R2,R3   |
| 3. | CLR  | R1        | 8.  | DEC | R0      |
| 4. | DIV  | @TEN,R1   | 9.  | JNE | LOOP    |
| 5. | SLA  | R2,12     | 10. | TEN | DATA 10 |

Fig. 7-5. Routine to convert  $R1$  to a BCD number in  $R3$ .

|    |     |        |     |     |       |
|----|-----|--------|-----|-----|-------|
| 1. | MOV | @X1,R1 | 6.  | ABS | R3    |
| 2. | MOV | @Z2,R3 | 7.  | MOV | R2,R2 |
| 3. | MOV | R1,R2  | 8.  | MPY | R3,R1 |
| 4. | XOR | R3,R2  | 9.  | JGT |       |
| 5. | ABS | R1     | 10. | NEG | R2    |

Fig. 7-6. A multiply and divide operation.

sequence in Fig. 7-6 uses this fact to calculate  $X3 = X1*X2$ .

The multiply routine can be used to produce a 32-bit product stored in  $R1$  and  $R2$ . It does not change any of the condition bits, so the sign test is made before the multiply. The first lines are used to load  $R1$  with  $X1$  and  $R3$  with  $X2$ . The absolute value instruction is used prior to the sign test on line 7, then the multiply is performed with a jump to  $GOE$  if the product is positive. After the multiply, the routine converts the lower 16 bits of the result back to a two's complement number. Only the lower word is converted, so as to allow additional arithmetic operations. The required  $GOE$  instruction is:

**GOE          MOV R2, at X3**

If no additional operations are desired, the following sequence can be used to convert both words to two's complement form.

```

INV  R2    ;invert R2 to ones
          complement
NEG  R3    ;convert R3 to nega-
          tive, twos complement
JNE  ZRO   ;jump if R3 equals zero
INC  R2    ;increment, R2 = twos
          complement of R2

```



ZRO \*  
\*

A similar routine can be used for signed division of two's complement numbers. The sign of X1/X2 will be the exclusive OR of X1/X2. For 16-bit numbers, the routine in Fig. 7-7 can be used to calculate X1/X2.

Lines 1 and 2 are used to load R2 with X1 and R3 with X2. The sign of R4 is set equal to the sign of X1/X2. The upper bits of the numerator are cleared on line 7. The 32-bit operand is divided by a 16-bit operand from the lower register of the pair. A sign test is made on line 9, and a jump is used to make a correction if the result is minus.

An expanded version of this routine for full division is shown in Fig. 7-8. The first few lines load X1 into R1 and R2, and line 3 stores

|               |              |
|---------------|--------------|
| 1. MOV @X1,R2 | 8. DIV R3,R1 |
| 2. MOV @X2,R3 | 9. MOV R4,R4 |
| 3. MOV R2,R4  | 10. JGT GOE  |
| 4. XOR R3,R4  | 11. NEG R1   |
| 5. ABS R2     | 12. GOE .    |
| 6. ABS R3     | 13. .        |
| 7. CLR R1     |              |

Fig. 7-7. Signed division operation.

|                 |            |
|-----------------|------------|
| 1. MOV @X1,R1   | 7. ABS R1  |
| 2. MOV @X1+2,R2 | 8. JGT L1  |
| 3. MOV @X2,R3   | 9. ABS R2  |
| 4. MOV R1,R4    | 10. JNE L1 |
| 5. XOR R3,R4    | 11. DEC R1 |
| 6. ABS R3       |            |

Fig. 7-8. Expanded version for full division.

X2 in R3. The sign is calculated in the next two lines. The absolute values are then taken prior to the jump instruction, which is used to invert the lower half if X1 is negative. Another jump is used if R2 is not equal to zero on line 10. For the required loop for the jumps, code:

```

LI   DIV   R1,R3 ;divide
     MOV   R4,R4 ;test sign
     JGT   GOE   ;jump to GOE to
                        correct if minus
     NEG   R1     ;negate
GOE  *
     *
```

Most multiply operations will be between operands which represent integers. For fractional numbers, a scaling approach can be used. This method makes use of the location of the decimal point in the register. For example, to multiply VAR by 0.75, first define the constant:

```

CON DAT >C0000 ;decimal point at
                left for constant
                of .75
     MOV at VAR,R1 ;move to get op-
                        erand for R1
     MPY at CON,R1 ;multiply CON
                        by VAR
```

To perform double-precision multiply operations on unsigned 32-bit numbers with the 9900, a cross multiply technique can be used. The method combines four single-precision multiples, as shown in Fig. 7-9. The cross-multiply method uses a double-precision addition which can be coded as follows:

```

A    R4,R2 ;add word of lower
      half
```

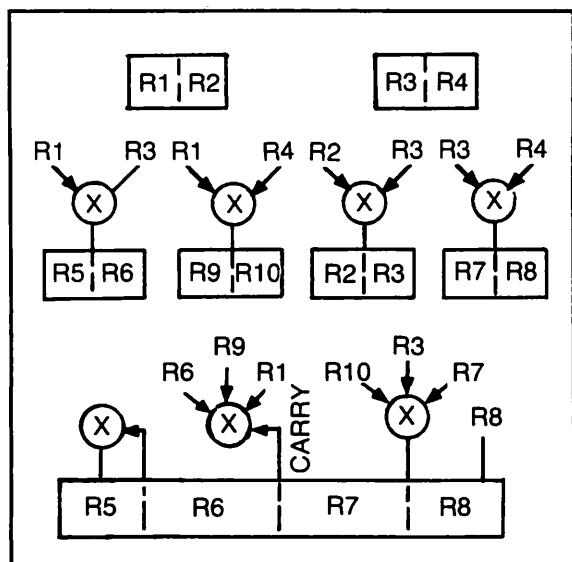


Fig. 7-9. Double-precision multiply operation.

```

JNC LOOP1 ;if carry clear correct
              upper half
INC  R1
LOOP1 A  R3,R1 ;add word of
              upper half

```

In this sequence, R1,R2 is used to store the sum of R1,R2 and R3,R4.

Expanding the concept used in this sequence, the double-precision multiply shown in Fig. 7-10 can be coded. Line 1 sets R5, R6 equal to R1\*R3 for the multiply. Then R7, R8 is set to R2\*R4, R9,R10 is set to R1\*R4, and R3,R4 is set to R2\*R3. The accumulator is cleared for the carry, and the jump to the first loop is made. A succession of loops is used to complete the routine. On line 15, R1 is set equal to the carry from the accumulator. The carry is added beginning on line 22, and the other carry on Fig. 7-9 is added in line 25.

An algorithm for calculating sines and cosines is useful in many alternating current applications, including signal processing. The coordinate rotation digital computer (CORDIC) algorithm is used in hand calculators. The algorithm makes use of the fact that any angle from zero to ninety degrees can be represented as the sum or difference of a set of base

```

1.      MOV  R1,R5
2.      MPY  R3,R5
3.      MOV  R2,R7
4.      MPY  R4,R7
5.      MOV  R1,R9
6.      MPY  R4,R9
7.      MPY  R2,R3
8.      CLR  R0
9.      A    R3,R7
10.     JNC  L1
11.     INC  R0
12. LOOP1 A    R10,R7
13.     JNC  LOOP2
14.     INC  R0
15. LOOP2 CLR  R1
16.     A    R2,R6
17.     JNC  LOOP3
18.     INC  R1
19. LOOP3 A    R9,R6
20.     JNC  LOOP4
21.     INC  R1
22. LOOP4 A    R0,R6
23.     JNC  LOOP5
24.     JNC  R1
25. LOOP5 A    R1,R5

```

Fig. 7-10. Expansion of double-precision multiply.

angles. For bases that are multiples, such as ninety, forty-five, twenty-two and a half, etc., the sine and cosine of an angle are found by computing:

$$\begin{aligned} X_0 &= 0 & X_i &= X_{i-1} + \tan(D_i A_i) * Y_{i-1} \\ Y_0 &= 0 & Y_i &= Y_{i-1} + \tan(D_i A_i) * X_{i-1} \end{aligned}$$

where  $D_i = \pm 1$  and  $A_i = \text{base angle}$ , then:

$$\begin{aligned} X_n &= R_n \sin u \\ Y_n &= R_n \cos u \end{aligned}$$

where  $R_n = 1/(\cos(D_1 A_1) * \dots * (\cos(D_n A_n)))$ . To simplify the multiply operations, let:

$$A_i = \tan^{-1}(1/2^{i-1})$$

so

$$\tan(A_i) = 1/2^{i-1}$$

The multiply operations then become a right-shift problem and the algorithm can be rewritten as:

$$\begin{aligned} V_0 &= -u \\ X_0 &= 0 \\ Y_0 &= 1/R_n \cdot .60725 \text{ for } 90, 45, 22.5 \dots \text{degrees} \\ X_i &= X_{i-1} - \text{sign}(V_{i-1}) * Y_{i-1} / 2^{i-1} \\ Y_i &= Y_{i-1} + \text{sign}(V_{i-1}) * X_{i-1} / 2^{i-1} \\ V_i &= V_{i-1} - \text{sign}(V_{i-1}) * (\arctan(1/2^{i-1})) \end{aligned}$$

Store the arctangent values in the table and use shift, addition, and subtraction operations to implement the algorithm. A variable shift con-

stant is required. This is easily done with the 9900. If the shift count is stored in R0, code:

```
SRA R1,R0 ;shift R1 right by
           the shift count
           in R0
```

The input must be scaled, since the sine and cosine are fractional values. The angle can be scaled so  $R_1 = \text{angle} * 256$  to provide eight bits for the integer and eight bits for the fraction. Then the X and Y values are scaled so  $X = \sin * 32768$  and  $Y = \cos * 32768$  for 16 bits of signed fraction. The complete routine for the sine and cosine calculations is shown in Fig. 7-11.

For an angle  $u$ , R1 is set equal to  $u * 256$  on entry, and R2 is used for the output sine and R3 for the output cosine. Line 1 clears R2 for  $X = 0$ , and line 2 loads  $Y = 6072526 * 32768 (2^{**}15)$ . A Clear instruction sets  $X_0$  to zero and then  $Y_0$  is set to Y. The shift and the count are cleared to zero and R1 is then negated for  $-u$ . The sign of  $u$  is tested on line 8, and a jump to LOOP2 is made if negative. A subtraction for  $X = X - Y/2^{**}i$  is made, and an addition sets  $Y = Y + X/2^{**}i$ . On line 16, the table is referenced for the subtraction;  $u = u - \arctan(1/2^{**}i)$ . LOOP2 starts by adding for R5, R2 to compute  $X = X + Y/2^{**}i$ ; then by subtracting R4, R3 calculates  $Y = Y - X/2^{**}i$ . The table is referenced again for  $u = u + \arctan(1/2^{**}i)$ , then LOOP3 begins with increment instruction to update the shift count. R4 is set to  $X/2^{**}i$  on lines 19 and 20, while R5 is set to  $Y/2^{**}i$  on lines 21 and 22. Line 23 allows continuation for the 12 iterations required, and the branch on line 25 returns control to the subroutine caller.

|     |       |      |            |
|-----|-------|------|------------|
| 1.  |       | CLR  | R1         |
| 2.  |       | L1   | R3,19898   |
| 3.  |       | CLR  | R4         |
| 4.  |       | MOV  | R3, R5     |
| 5.  |       | CLR  | R0         |
| 6.  |       | CLR  | R6         |
| 7.  |       | NEG  | R1         |
| 8.  | LOOP1 | MOV  | R,R1       |
| 9.  |       | JLT  | LOOP2      |
| 10. |       | S    | R5,R2      |
| 11. |       | A    | R4,R3      |
| 12. |       | S    | TAB(R6),R1 |
| 13. |       | JMP  | LOOP3      |
| 14. | LOOP2 | A    | R5,R2      |
| 15. |       | S    | R4,R3      |
| 16. |       | A    | TAB(R6),R1 |
| 17. | LOOP3 | INC  | R0         |
| 18. |       | INCT | R6         |
| 19. |       | MOV  | R2,R4      |
| 20. |       | SRA  | R4,R0      |
| 21. |       | MOV  | R3,R5      |
| 22. |       | SRA  | R5,R0      |
| 23. |       | CI   | R0,12      |
| 24. |       | JNE  | LOOP1      |
| 25. |       | B    | *R11       |
| 26. | TAB   | DATA | 11520      |
| 27. |       | DATA | 6800       |
| 28. |       | DATA | 3593       |
| 29. |       | DATA | 1824       |
| 30. |       | DATA | 916        |
| 31. |       | DATA | 458        |
| 32. |       | DATA | 229        |
| 33. |       | DATA | 115        |
| 34. |       | DATA | 57         |
| 35. |       | DATA | 29         |
| 36. |       | DATA | 14         |
| 37. |       | DATA | 7          |

Fig. 7-11. Sine and cosine calculation routine.

If the tangent is required, it can be computed from an additional division of the sine and cosine values.

An interrupt interface for the 9900 can be constructed from standard TTL components. A single interrupt requires no additional hardware, less than eight interrupts requires one priority encoder and from eight to fifteen interrupts requires two priority encoders and the AND gates and inverter shown in Fig. 7-12.

Four bits of the status register are used to provide the code used for masking the interrupt signals. The mask is under program control, and the processor uses it to determine the interrupt sequence. If the mask level is at six, the program allows interrupt levels zero to six and inhibits levels seven through fifteen. If level four is requested, the processor will start the interrupt sequence at the end of execution of the current instruction. The mask is then automatically moved down one step to level three, which masks out the lower interrupts and allows the higher ones. When the return instruction occurs, the status register value is changed back to level four. The sequence continues until all interrupts are processed to allow nested interrupts without polling. For a single interrupt, the interrupt request input is used and IC0 through IC3 are hard-wired.

A minimum 9900 microprocessor system is shown in Fig. 7-13. Eight bits of input and output interface are used for a total package count of thirteen devices. The memory contains a 16-bit by 1024-word ROM system and a 256 by 16 RAM system.

A system for 65,536 bytes of memory is shown in Fig. 7-14. The I/O interface can support 4096 input bits and 4096 output bits. Ex-

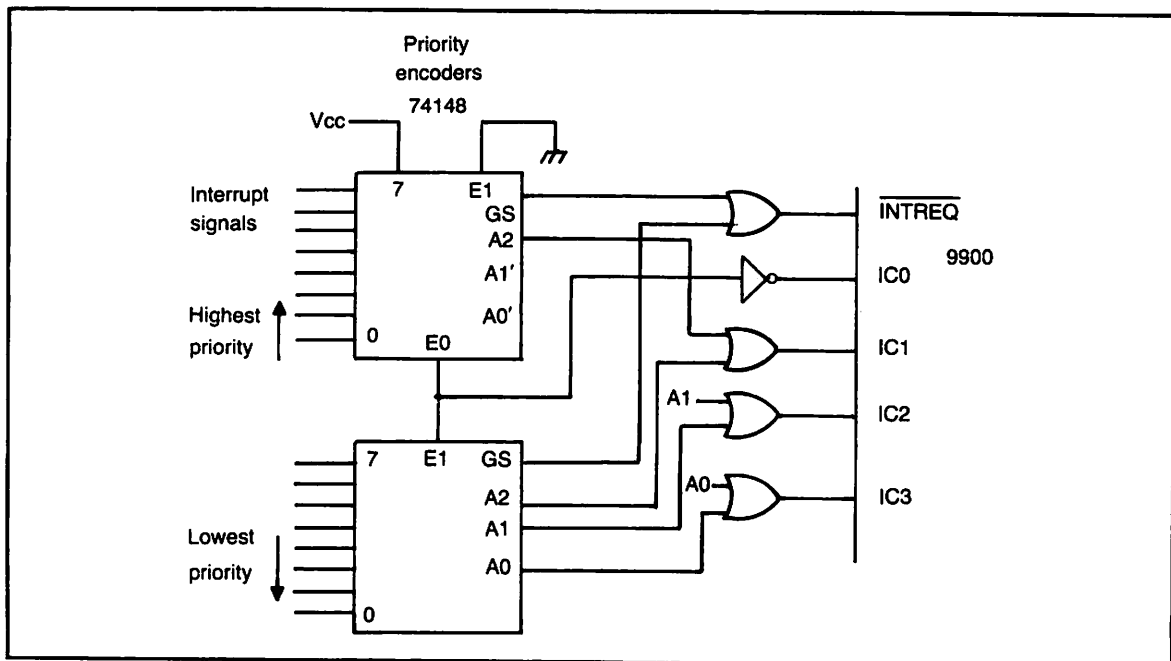


Fig. 7-12. Interrupt interface for the TMS9900.

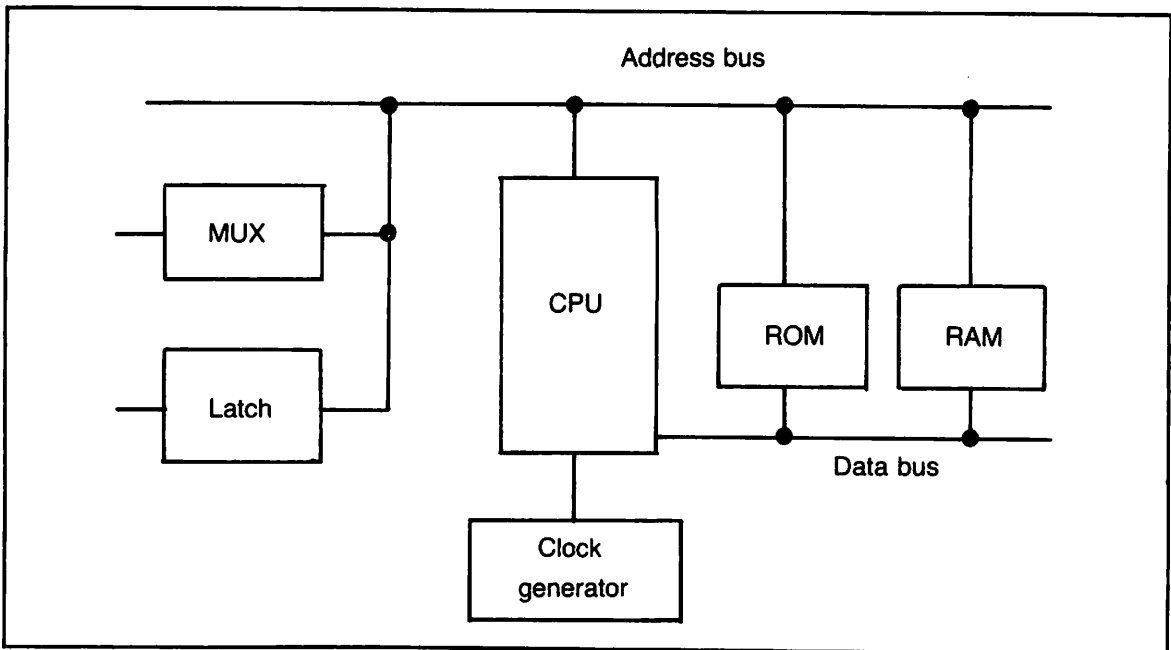


Fig. 7-13. The minimum TMS9900 system.

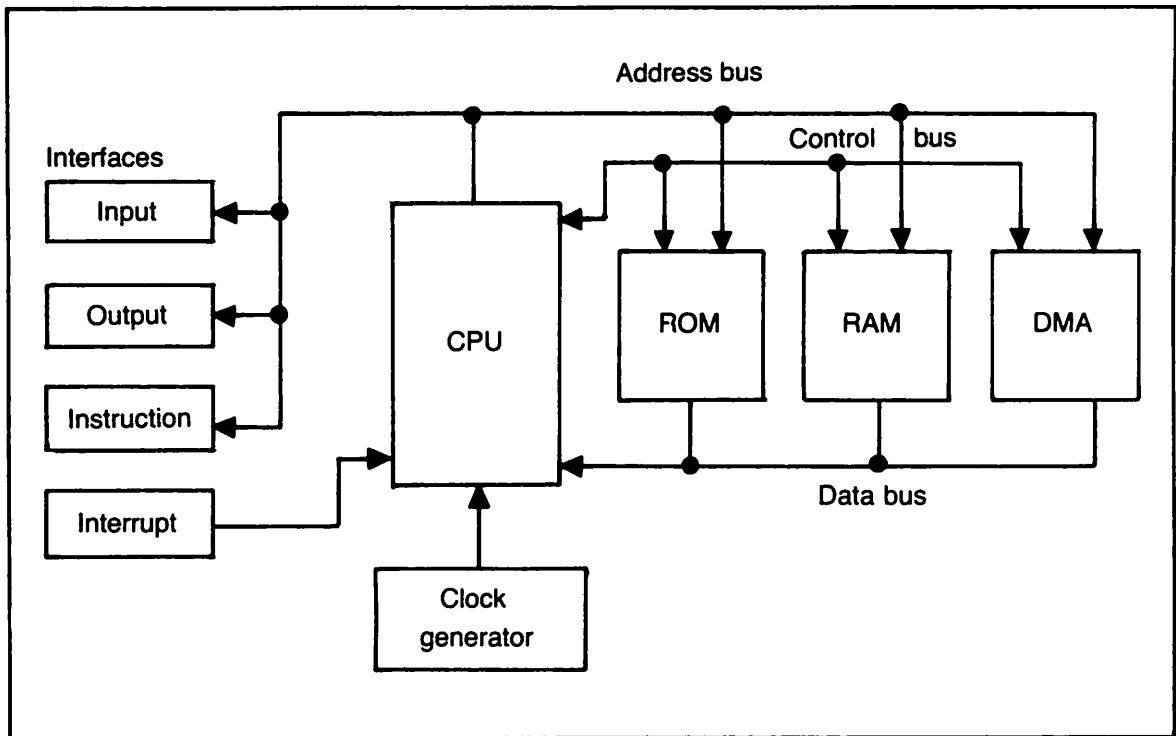
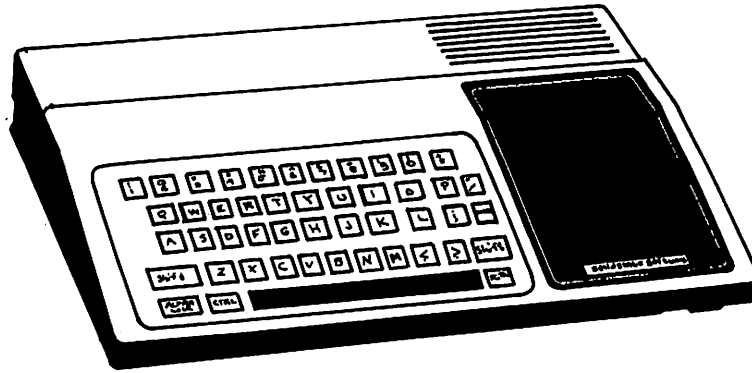


Fig. 7-14. A system for 65,536 bytes of memory.

ternal interrupts (15) are on the interrupt interface. The clock generator also includes memory-decode and control-synchronization logic. The system buses require buffers.

## Chapter 8



# Programs

This chapter is designed to give the new TI-99/4A owner an opportunity to enter his or her own programs from program line listings. Even if you've owned your computer for some time, you should find the programs in this chapter interesting, fun, and useful. A myriad of program types are included; some are devoted to fun and games; others have practical uses in home or business management. BASIC is all that's needed to run these programs.

When entering program lines from printed pages, take your time. If you rush, you're bound to make typing errors, greatly lengthening the debugging time. Remember, if you accidentally type a semicolon in place of a colon, the program will not run, and if you leave out a quotation mark or omit a comma, the program will not run.

The built-in debugging system will detect

many errors quickly, but there are other errors that can create problems even though the program will run with them. For instance, if you type a comma following a Print statement and preceding a variable, the next variable will be printed on the line below the first. If you insert a semicolon between two variables in a Print statement, the variables will be displayed side by side. If you type in a semicolon where a comma should be, the program will run, but the display may not be set up as originally intended. By taking time and rechecking each line with the line in the book before pressing Enter, you will be assured of a fairly clean program run on the first attempt.

Modify these programs to satisfy your individual needs. First enter and debug any program discussed in this chapter as shown in the line listing. Once the published program works

properly, it is the time to begin making changes. This way you will know if these changes are creating an execution problem or some typographical error.

### **NUMBERS GUESS GAME**

In Numbers Guess the computer selects a number from 1 to 100. It is up to you to guess what the number is in the smallest number of

guesses. Each time you guess a number, a clue will be given. If your guess is low, the computer will display **THAT'S TOO LOW!! TRY AGAIN!!** If your guess is too high, a prompt appears telling you so. This information will let you quickly hone in on the correct number. When you guess correctly, the computer will respond and also print the number of guesses it took. The program listing follows.

```
10 REM NUMBERS GUESS GAME
20 REM PROGRAM RUNS IN TI-BASIC
30 REM PROGRAM REQUIRES 896 BYTES OF RAM

40 REM COPYRIGHT FREDERICK HOLTZ & AS
   SOCIATES 2/5/83
50 RANDOMIZE
60 CALL CLEAR
70 A=INT(RND*100)+1
80 PRINT"THE COMPUTER HAS CHOSEN A NUMBE
   R"
90 PRINT
100 PRINT"BETWEEN 1 AND 100. WHAT DO YOU
   THINK"
110 PRINT
120 PRINT"THIS NUMBER IS?"
130 INPUT B
140 CALL CLEAR
150 NT=NT+1
160 IF B=A THEN 250
170 IF B<A THEN 230
180 PRINT"THAT'S TOO HIGH!! TRY AGAIN!!"
190 FOR T=1 TO 1000
200 NEXT T
210 CALL CLEAR
220 GOTO 80
230 PRINT"THAT'S TOO LOW!! TRY AGAIN!!"
240 GOTO 190
250 PRINT"THAT IS THE CORRECT NUMBER !!!"
```



```

260 PRINT "IT TOOK YOU A TOTAL OF";NT;"GUESSES"
270 FOR T=1 TO 2000
280 NEXT T
285 NT=0
290 CALL CLEAR
300 INPUT"WOULD YOU LIKE TO PLAY AGAIN (
YES/NO)  ":A$
310 IF A$="YES" THEN 60
320 IF A$="NO" THEN 340
330 GOTO 380
340 CALL CLEAR
350 PRINT"THE GAME IS OFFICIALLY OVER."
360 PRINT"THANK YOU FOR PLAYING"
370 END
380 CALL CLEAR
390 PRINT"YOU HAVE NOT RESPONDED WITH A
YES OR NO ANSWER!!!!"
400 FOR T=1 TO 1000
410 NEXT T
420 CALL CLEAR
430 GOTO 300

```

Look at the program; you will see the Randomize statement is used in line 50. This assures a different number on each program run. The screen is cleared in line 60. Line 70 assigns the random number to variable A. The INT function is used here because the number must always be an integer (a whole number).

Lines 80 through 120 print a short set of instructions, telling you the computer has chosen a number between 1 and 100. Print statements have been used in lines 90 and 110 without any variable or quoted phrase following them; the Print statements here are used as spacers between printed lines. When a Print statement is used alone, the entire line is printed as a series of blanks. Line 90 effectively double-spaces the text found in lines 80 and 100. Line 110 performs the same function

for the text found in lines 100 and 120. Line 130 uses the Input statement to let you enter your guess as to the number. Your guess is assigned to the numerical variable B. When you press Enter, line 140 clears all information from the screen and line 150 starts the guess count process. The value of NT has not been assigned, therefore it is equal to 0. In line 150 NT equals its present value plus 1. When line 150 is first encountered, NT is given a value of 1. Each time a guess is made, NT is incremented by one.

Lines 160 and 170 use If-Then statements to test the value entered for B. Line 160 checks if B is equal to A. If B is equal to A, the program branches to line 250. The computer screen displays **THAT IS THE CORRECT NUMBER!!!**, and **IT TOOK YOU A TOTAL**

OF \_\_\_\_\_ GUESSES. Lines 270 and 280 are then accessed. These encompass a time delay loop. The loop does nothing but count from 1 to 2000. This lets the screen information be displayed for a few seconds before being automatically cleared by the Call Clear statement in line 290, accessed only when the For-Next loop times out.

You are then asked if you would like to play again, requiring a yes or no response. If you type in an improper answer, line 330 branches to line 380, where the screen is cleared and you are chastised by the prompt **YOU HAVE NOT RESPONDED WITH A YES OR NO ANSWER!!!!** Lines 400 and 410 encompass another time-delay loop. When this times out, the Call Clear statement in line 420 clears the screen, and line 430 branches to line 300, where you are asked once more if you'd like to play again.

Returning to line 170, let's see what happens when your guess is not equal to the number the computer has chosen. Line 170 tests for a condition of B less than A. When B is less than A, there is a branch to line 230, and the screen displays a prompt telling you your guess is too low. Line 240 branches to line 190. This line, along with line 200, forms another time-delay loop to allow the low guess prompt to establish itself on the screen for a few seconds. When the loop times out, the Call Clear statement in line 210 clears the screen, and line 220 branches back to line 80. Here, you are

asked to guess again. During your second guess, line 150 is accessed for the second time, and NT is assigned a value of 2. If you guess correctly this time, it's detected in line 160, and the win sequence is accessed.

There is no If-Then statement in this listing to cover a condition where your guess is higher than the computer's number. One could be added, but it's certainly not necessary, as line 180 prints the **TOO HIGH** prompt. Here's how it works: if B is not equal to A, line 160 does nothing. If B is less than A, line 170 creates a branch. If it's more than A, line 170 does nothing. This leaves line 180. If line 180 is accessed, this means B cannot be equal to A or less than A, so it has to be more than A, and the **TOO HIGH** prompt appears. Lines 190 through 220 are accessed and let you guess again.

## LOAN CALCULATION

This is an unusual financial program. Most loan programs let you enter a mortgage amount, an interest rate, and the number of years over which the mortgage is to be amortized. The screen then displays the monthly payment. This program works differently; it will calculate the maximum amount of money you can afford to borrow based on interest rate and the maximum amount you can afford to pay for a set number of months. The program listing follows.

Look at the program; line 60 clears the

```
10 REM PROGRAM TO CALCULATE THE MAXIMUM  
LOAN AFFORDABLE  
20 REM BASED UPON CURRENT INTEREST AND A  
BILITY TO PAY  
30 REM PROGRAM RUNS IN TI-BASIC
```

```

40 REM PROGRAM REQUIRES 640 BYTES OF RAM

50 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/3/83
60 CALL CLEAR
70 INPUT "HOW MANY MONTHLY PAYMENTS DO YO
U WANT TO MAKE":A
80 CALL CLEAR
90 INPUT "WHAT IS THE CURRENT INTEREST RA
TE ON BORROWED MONEY":I
100 I=I/1200
110 CALL CLEAR
120 INPUT "WHAT IS THE MAXIMUM MONTHLY PA
YMENT YOU CAN AFFORD":M
130 CALL CLEAR
140 X=M*(1-(1+I)^-A)/I
150 X=INT(X)
160 PRINT "YOU CAN AFFORD TO BORROW UP TO
$"; X

```

screen and line 70 prompts you to enter the number of monthly payments you want to make. Line 80 then clears the screen and another prompt appears, asking for the current interest rate on money borrowed. This should be entered as a percentage (i.e., 13 = 13% = .13). Line 100 reassigns the value of I to I divided by 1200. This breaks the yearly interest rate of 13 percent (in this example) into monthly interest of .0108333. Line 110 clears the screen again, and line 120 asks you to enter the maximum monthly payment you can afford. This is assigned to the numerical variable M. Line 130 clears the screen, while lines 140 and 150 perform the calculations. The amount you can afford to borrow is assigned to the variable X. In line 140, we see that X is equal to the maximum monthly payment (M) times the monthly interest plus 1 raised to the  $-A$  power and divided by the monthly interest. Line 150

converts the value of X to an integer and line 160 prints this value on the screen.

This program can be put to good use around the home.

## FORTUNE TELLER

Fortune Teller is a simulation of the fortune-telling booth, which used to be popular at carnivals and circuses in the middle part of this decade. It's also similar to many different types of mechanical games that provide a response to any question that can be answered with a yes or no.

Computer fortune-telling is attractive to many individuals. This program in no way attempts to predict the future. The answer to each question is determined before the question is asked. The required keyboard response is built into the program simply for game effect. The program listing follows.

```

10 REM FORTUNE TELLER
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/7/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 1920 BYTES OF RA
M
50 CALL CLEAR
60 PRINT"ASK THE COMPUTER ANY QUESTION W
HICH CAN BE ANSWERED"
70 PRINT
80 PRINT"WITH A YES OR NO. IT WILL TELL
YOU WHAT YOU NEED TO KNOW.
90 PRINT
100 INPUT"PRESS <ENTER> TO ACCESS THE CO
MPUTER FORTUNE TELLER." :A$
110 CALL CLEAR
120 PRINT"WHAT IS IT YOU WISH TO ASK. THE
ALL-SEEING, ALL-KNOWING"
130 PRINT
140 PRINT"COMPUTER NOW?"
150 PRINT
160 INPUT B$
170 IF B$="" THEN 180 ELSE 190
180 GOTO 160
190 RANDOMIZE
200 X=INT(RND*20)+1
210 IF X=1 THEN 220 ELSE 240
220 Q$="YES"
230 GOTO 790
240 IF X=2 THEN 250 ELSE 270
250 Q$="NO"
260 GOTO 790
270 IF X=3 THEN 280 ELSE 300
280 Q$="POSSIBLY"
290 GOTO 790
300 IF X=4 THEN 310 ELSE 330
310 Q$="ABSOLUTELY NOT"
320 GOTO 790
330 IF X=5 THEN 340 ELSE 360
340 Q$="MOST ASSUREDLY"
350 GOTO 790

```

```

360 IF X=6 THEN 370 ELSE 390
370 Q$="THERE IS A SLIGHT POSSIBILITY"
380 GOTO 790
390 IF X=7 THEN 400 ELSE 420
400 Q$="I WILL NOT ANSWER THAT"
410 GOTO 790
420 IF X=8 THEN 430 ELSE 450
430 Q$="THE SIGNS ARE GOOD"
440 GOTO 790
450 IF X=9 THEN 460 ELSE 480
460 Q$="I MUST GIVE YOU A NEGATIVE ON TH
AT ONE"
470 GOTO 790
480 IF X=10 THEN 490 ELSE 510
490 Q$="ASK ME AGAIN LATER"
500 GOTO 790
510 IF X=11 THEN 520 ELSE 540
520 Q$="THAT QUESTION IS AN INSULT TO MY
ARTIFICIAL INTELLIGENCE"
530 GOTO 790
540 IF X=12 THEN 550 ELSE 570
550 Q$="THE SIGNS ARE NOT GOOD"
560 GOTO 790
570 IF X=13 THEN 580 ELSE 600
580 Q$="MOST CERTAINLY"
590 GOTO 790
600 IF X=14 THEN 610 ELSE 630
610 Q$="CHANCES ARE FIFTY/FIFTY"
620 GOTO 790
630 IF X=15 THEN 640 ELSE 660
640 Q$="THINK IT OVER AND ASK AGAIN"
650 GOTO 790
660 IF X=16 THEN 670 ELSE 690
670 Q$="VERY DOUBTFUL"
680 GOTO 790
690 IF X=17 THEN 700 ELSE 720
700 Q$="THERE IS A VERY GOOD POSSIBILITY
"
710 GOTO 790
720 IF X=18 THEN 730 ELSE 750
730 Q$="THE CHANCES ARE SLIM"

```

```

740 GOTO 790
750 IF X=19 THEN 760 ELSE 780
760 Q$="CONCENTRATE HARDER. I'M NOT RECE
IVING YOU"
770 GOTO 790
780 Q$="PROBABLY"
790 PRINT Q$
800 INPUT "PRESS ENTER TO CONTINUE":AA$
810 CALL CLEAR
820 GOTO 120

```

Like the Numbers Guess Game, this game depends on the output of a random number. Lines 50 through 140 initialize the screen and print a brief set of instructions. Line 100 lets you start the output sequence by pressing Enter. Line 160 uses an Input statement to enter the question from the keyboard. It is necessary to use a string variable following the Input statement, because the keyboard entry will consist of letters and possibly even numbers. Remember that when the Input statement is followed by a string variable it is not mandatory that keyboard information be entered. Such a line can be used to temporarily halt execution until Enter is pressed. In this case, it is not desirable to continue execution if Enter is pressed without entering some type of keyboard information. Line 170 takes care of this situation; if B\$ is equal to no keyboard input, the program branches to 180.

No keyboard input is represented by two back-to-back quotation marks. When there is no keyboard input, line 180 is accessed and branches back to line 160, where the input prompt appears again. Execution will not continue until you have entered something via the keyboard. This can be a single number or letter, but you've got to do something before you press Enter.

When the input requirement has been satisfied, the program continues by executing line 190, the Randomize statement. Line 200 uses the RND function to return a value to variable X which can be anywhere from 1 to 20. The output number will always be an integer due to the use of the INT function.

The heart of the program is found in lines 210 through 780. These are the If-Then-Else statements which determine what answer will be given based on the random number assigned to the variable X. If X is equal to 1, this is detected in line 210, which branches to line 220. The answer is always assigned to the variable Q\$. In this case, Q\$ is equal to yes. There is then a branch to line 790, which prints Q\$ on the screen. The player is given the opportunity to ask another question by pressing Enter. This clears the screen and there is a branch to line 120, which starts the program over again.

Going back to line 210. if X is not equal to 1, the Else portion of the statement branches to line 240. Here, X is tested for a value of 2. If this condition is true, Q\$ equals No, and this answer is printed on the screen. If X is not equal to 2, there is a branch to line 270, where X is checked against a possible value of 3. This process continues until the proper value of X is

matched. Each differing value of X results in a different response which is printed on the screen.

This program is meant as an amusement or party game. While a bit of chicanery is necessary to get the players set up to participate, you should always let them in on the secret once the game is complete.

You can shorten this program considerably by deleting lines 270 through 780. You will also have to change the 20 in line 200 to 2. This weakens the program, restricting it to two answers, yes or no. The way the program is originally set up is far more fun.

## TEACHER'S PET

Here's a practical program designed to aid teachers to establish a grading system and keep track of scores on tests. It allows for the input of many grades, which will be sorted and assigned to grade categories. The resulting printout will look similar to that shown in Fig.

8-1, which shows a sample printout of a geometry test. The category can be typed in, along with the classroom designation and the name of the instructor.

Scores may be entered in any order (randomly). The program will extract the highest score, the lowest score, and the average score, and display each. It will then provide a breakdown of average scores in any particular grade category (seen in the bottom five lines of Fig. 8-1). Here there were two scores in the A category, and the average of these two was a score of 97. There were four scores in the B category, with an average of 87, and so on.

This program assigns grades (A-F) based on the following criteria: any score higher than 89 is considered to be an A. Anything above 79 is a B, above 69 a C, above 59 a D, and below 59 an F. Adjust these values in lines 210, 260, 310, 360, and 410 to match your own scoring system.

The program listing for teacher's pet follows on page 128.

```
CATEGORY: GEOMETRY
CLASSROOM: 2-A
INSTRUCTOR: HOLTZ
```

```
HIGHEST SCORE: 98
```

```
LOWEST SCORE: 57
```

```
AVERAGE OF TOTAL SCORES: 82
```

```
IN 'A' CATEGORY, AVERAGE SCORE OUT OF 2 SCORES WAS      97
IN 'B' CATEGORY, AVERAGE SCORE OUT OF 4 SCORES WAS      87
IN 'C' CATEGORY, AVERAGE SCORE OUT OF 2 SCORES WAS      78
IN 'D' CATEGORY, AVERAGE SCORE OUT OF 1 SCORES WAS      67
IN 'F' CATEGORY, AVERAGE SCORE OUT OF 1 SCORES WAS      57
```

Fig. 8-1. Sample printout for a run of the Grading Program.

```

10 REM GRADING PROGRAM
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/14/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 2304 BYTES OF RA
M
50 CALL CLEAR
60 GOSUB 910
70 INPUT"ENTER THE ACADEMIC CATEGORY(I.E
. HISTORY,ALGEBRA,ETC.)":CAT$
80 CALL CLEAR
90 INPUT"ENTER CLASSROOM DESIGNATION IF
ANY":CLASS$
100 CALL CLEAR
110 INPUT"ENTER NAME OF INSTRUCTOR":INST
RUCT$
120 CALL CLEAR
130 INPUT"HOW MANY SCORES WILL BE ENTERE
D":N
140 DIM S(200)
150 CALL CLEAR
160 PRINT"TYPE IN A SINGLE GRADE AT A TI
ME(ANY ORDER) AND PRESS <ENTER>"
170 FOR X=1 TO N
180 INPUT S(X)
190 NEXT X
200 FOR X=1 TO N
210 REM GRADE 'A' SEQUENCE
220 IF S(X)>89 THEN 230 ELSE 270
230 AC1=AC1+S(X)
240 C1=C1+1
250 GOTO 440
260 REM GRADE 'B' SEQUENCE
270 IF S(X)>79 THEN 280 ELSE 320
280 AC2=AC2+S(X)
290 C2=C2+1
300 GOTO 440
310 REM GRADE 'C' SEQUENCE
320 IF S(X)>69 THEN 330 ELSE 370
330 AC3=AC3+S(X)
340 C3=C3+1

```



```

350 GOTO 440
360 REM GRADE 'D' SEQUENCE
370 IF S(X)>59 THEN 380 ELSE 420
380 AC4=AC4+S(X)
390 C4=C4+1
400 GOTO 440
410 GRADE 'F' SEQUENCE
420 AC5=AC5+S(X)
430 C5=C5+1
440 TOTAL=TOTAL+S(X)
450 NEXT X
460 TOTAV=TOTAL/N
470 L=100
480 H=0
490 FOR X=1 TO N
500 IF L>S(X) THEN 510 ELSE 520
510 L=S(X)
520 IF H<S(X) THEN 530 ELSE 540
530 H=S(X)
540 NEXT X
550 IF C1=0 THEN 560 ELSE 580
560 AV1=0
570 GOTO 590
580 AV1=AC1/C1
590 IF C2=0 THEN 600 ELSE 620
600 AV2=0
610 GOTO 630
620 AV2=AC2/C2
630 IF C3=0 THEN 640 ELSE 660
640 AV3=0
650 GOTO 670
660 AV3=AC3/C3
670 IF C4=0 THEN 680 ELSE 700
680 AV4=0
690 GOTO 710
700 AV4=AC4/C4
710 IF C5=0 THEN 720 ELSE 730
720 AV5=0
730 AV5=AC5/C5
740 CALL CLEAR

```

```

750 PRINT "CATEGORY:";CAT$
760 PRINT "CLASSROOM:";CLASS$
770 PRINT "INSTRUCTOR:";INSTRUCT$
780 PRINT
790 PRINT "HIGHEST SCORE:";H
800 PRINT
810 PRINT "LOWEST SCORE:";L
820 PRINT
830 PRINT "AVERAGE SCORE:";INT(TOTAV)
840 PRINT
850 PRINT "IN 'A' CATEGORY,AVERAGE SCORE
  OUT OF";C1;"SCORES WAS";INT(AV1)
860 PRINT "IN 'B' CATEGORY,AVERAGE SCORE
  OUT OF";C2;"SCORES WAS";INT(AV2)
870 PRINT "IN 'C' CATEGORY,AVERAGE SCORE
  OUT OF";C3;"SCORES WAS";INT(AV3)
880 PRINT "IN 'D' CATEGORY,AVERAGE SCORE
  OUT OF";C4;"SCORES WAS";INT(AV4)
890 PRINT "IN 'F' CATEGORY,AVERAGE SCORE
  OUT OF";C5;"SCORES WAS";INT(AV5)
900 END
910 PRINT"THIS PROGRAM WILL ALLOW EDUCAT
ORS TO INPUT STUDENT GRADES"
920 PRINT
930 PRINT"FOR A PARTICULAR TEST. THE SCO
RES WILL THEN BE ACCURATELY"
940 PRINT
950 PRINT"SORTED AND COMPARED. THE OUTPU
T SCREEN WILL THEN DISPLAY"
960 PRINT
970 PRINT"THE HIGHEST SCORE, LOWEST SCOR
E, NUMBER OF SCORES IN "
980 PRINT
990 PRINT"A GIVEN CATEGORY AND AVERAGE S
CORES. 200 SCORES MAXIMUM."
1000 PRINT
1010 PRINT
1020 PRINT"PRESS <ENTER> TO BEGIN"
1030 INPUT AA$
1040 CALL CLEAR
1050 RETURN

```

When the program is first run, you will be asked to enter the academic category, such as history, algebra, etc., assigned to the string variable CAT\$. Line 80 clears the screen and you are asked to enter a classroom designation, assigned to the variable CLASS\$. Line 110 lets you enter the name of the instructor (INSTRUCT\$). You are finally asked for the total number of scores to be entered. This is assigned to the numerical variable N. Line 140 establishes an array (S). This array will hold up to 200 scores.

You are prompted to begin entering a single grade at a time and to press Enter after each entry. Line 170 begins a For-Next loop which counts from 1 to N (number of grades). Line 180 lets you enter each grade to the array. When the loop times out, another loop is entered which reads back the contents of the array and assigns each score a letter grade based on its numerical value. Line 460 establishes a grade average by assigning the variable TOTAV the value of all the scores added together and divided by the number of scores. Lines 500 through 750 perform the sorting routine, arranging the elements of the array in descending order.

The screen print begins in line 740. The screen is cleared and lines 750 through 770 print the category, classroom, and instructor information. Line 790 prints the highest score, while the lowest score is printed in line 810. Line 830 prints the average of total scores, which is an integer of the variable TOTAV. The individual category printout is han-

dled in lines 850 through 890.

This is a highly useful program. It may take a half hour or so to enter into the machine, but once debugged and stored on cassette tape, it is readily available and can save teachers hours of work assigning grades and especially rating the performance of a particular class. This program lets the teacher see the overall grading results of a class. One can easily tell when an overall average is beginning to rise or decline. Also, some grades are given based upon the highest and lowest grades on a single quiz or exam, instead of on the 100-point system. This program can be useful in determining this as well.

### COMICAL INTELLIGENCE TEST

Here is a fun program for older children and adults. It's a comical intelligence test that asks a question and then gives a choice of three possible answers. The questions are easy for adults. The most fun is obtained from purposely entering wrong answers and noting the machine's responses. This program is intended to be used only as a guide, since you undoubtedly will want to enter your own questions, choice of answers, and responses. Using this program as a format guide, you can custom-tailor other comical intelligence tests or even a very serious one. Near the end of the program you are asked to enter the number of questions you missed and you are then assigned a grade value.

The listing for Comical Intelligence Test follows.

```
10 REM INTELLIGENCE TEST (COMICAL)
20 REM COPYRIGHT FREDERICK HOLTZ &
ASSOCIATES 6/82
```

```

40  REM PROGRAM RUNS IN TI-BASIC
50  CALL CLEAR
60  PRINT"HOW SMART ARE YOU? THIS TEST W
ILL TELL IT ALL!"
70  PRINT
80  PRINT
90  PRINT"ANSWER THE FOLLOWING QUESTIONS
"
100 PRINT
110 INPUT"PRESS ENTER WHEN READY TO BEG
IN":B$
120 CALL CLEAR
130 PRINT"WHO IS THE PRESIDENT OF THE U
SA?"
140 PRINT
150 PRINT"1.JIMMY CARTER"
160 PRINT"2.LAWRENCE QUARK"
170 PRINT"3.RONALD REAGAN"
180 PRINT
190 PRINT
200 PRINT"SELECT 1,2, OR 3 AND PRESS EN
TER"
210 PRINT
220 PRINT
230 INPUT A
240 PRINT
250 PRINT
260 IF A=1 THEN 270 ELSE 280
270 PRINT"***DUMMY, HE WAS A FORMER PRE
SIDENT***"
280 IF A=2 THEN 290 ELSE 300
290 PRINT"***LARD HEAD, HE'S AN OBSTETR
ICIAN***"
300 IF A=3 THEN 310 ELSE 320
310 PRINT"***YOU ARE VERY SMART***"
320 PRINT
330 PRINT
340 INPUT"PRESS ENTER TO CONTINUE":B$
350 CALL CLEAR
360 PRINT"WHO DISCOVERED AMERICA?"

```

```

370 PRINT
380 PRINT
390 PRINT"1.CHRISTOPHER COLUMBUS"
400 PRINT"2.FELIX THE CAT"
410 PRINT"3.JOAN OF ARC"
420 PRINT"MAKE YOUR SELECTION AS BEFORE
--BE SURE TO PRESS ENTER"
430 PRINT
440 PRINT
450 INPUT B
460 IF B=1 THEN 470 ELSE 480
470 PRINT"***YOU MUST BE A GENIUS***"
480 IF B=2 THEN 490 ELSE 500
490 PRINT"***YOU'VE GOT TO BE KIDDING**
*"
500 IF B=3 THEN 510 ELSE 520
510 PRINT"***WRONG! THAT ANSWER BURNS M
E UP"
520 PRINT
530 PRINT
540 INPUT"PRESS ENTER TO CONTINUE":B$
550 CALL CLEAR
560 PRINT"WHY DO YOU HEAR THUNDER BEFOR
E SEEING THE LIGHTNING FLASH?"
570 PRINT
580 PRINT
590 PRINT"1.BECAUSE YOU DIDN'T LOOK QUI
CKLY ENOUGH"
600 PRINT"2.BECAUSE LIGHT TRAVELS FASTE
R THAN SOUND"
610 PRINT"3.ARE YOU KIDDING? YOU SEE TH
E FLASH THEN HEAR THE THUNDER"
620 PRINT"MAKE YOUR SELECTION--PRESS EN
TER"
630 PRINT
640 PRINT
650 INPUT C
660 IF C=1 THEN 670 ELSE 680
670 PRINT"***THAT'S A WRONG ANSWER, THU
NDERHEAD***"
680 IF C=2 THEN 690 ELSE 700

```

```

690 PRINT"***THE STATEMENT IS CORRECT--
YOUR ANSWER IS WRONG***"
700 IF C=3 THEN 710 ELSE 720
710 PRINT"***GEE, THAT'S RIGHT! DID YOU
EVER THINK OF BECOMING A COMPUTER***"
720 PRINT
730 PRINT
740 INPUT"IF YOU DARE TO CONTINUE, PRES
S ENTER AGAIN":B$
750 CALL CLEAR
760 PRINT"WHAT IS 5 TIMES 6?"
770 PRINT
780 PRINT
790 PRINT"1.30"
800 PRINT"2.56"
810 PRINT"3.88"
820 PRINT
830 PRINT"SELECT ONLY ONE ANSWER, THEN
PRESS ENTER"
840 PRINT
850 PRINT
860 INPUT D
870 IF D=1 THEN 880 ELSE 890
880 PRINT"***THAT IS A CORRECT ANSWER**
*"
890 IF D=2 THEN 900 ELSE 910
900 PRINT"***WRONG! GO BACK TO SCHOOL**
*"
910 IF D=3 THEN 920 ELSE 930
920 PRINT"***WRONG! WAS THE QUESTION TO
O HARD FOR YOU?***"
930 PRINT
940 PRINT
950 INPUT"PRESS ENTER TO RATE YOUR SCOR
E":B$
960 CALL CLEAR
970 PRINT"HOW MANY ANSWERS DID YOU MISS
"
980 INPUT E
990 IF E=0 THEN 1060
1000 IF E=1 THEN 1090
1010 IF E=2 THEN 1120

```

```

1020 IF E=3 THEN 1150
1030 IF E=4 THEN 1180
1040 IF E<1 THEN 970
1050 IF E>4 THEN 970
1060 CALL CLEAR
1070 PRINT"PERFECT SCORE!! CONGRATULATI
ONS!!"
1080 GOTO 1200
1090 CALL CLEAR
1100 PRINT"NOT BAD AT ALL!! THE NEXT TI
ME YOU'LL GET IT PERFECT!!"
1110 GOTO 1200
1120 CALL CLEAR
1130 PRINT"YOU DIDN'T TRY VERY HARD!! S
CORE:50%!!"
1140 GOTO 1200
1150 CALL CLEAR
1160 PRINT"YOU ONLY GOT 1 RIGHT!! DO YO
U UNDERSTAND ENGLISH?"
1170 GOTO 1200
1180 CALL CLEAR
1190 PRINT"YOUR SCORE IS A BIG,FAT ZERO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
1200 FOR ZZ=1 TO 2500
1210 NEXT ZZ
1220 CALL CLEAR
1230 END

```

When the program is first run, a brief set of instructions are provided. The screen is cleared and the first question is asked (set up in line 130). Lines 150 through 170 give you a choice of three answers. You are prompted to select one of the three numbers and then press Enter. The number you enter is assigned to the variable A (line 230). Line 260 tests for the value of A and assigns an appropriate response. Lines 280 and 300 test for other values of A. Lines 270, 290, and 310 contain the computer responses, which run from quite insulting to

extremely complimentary, depending on whether or not the correct answer was entered.

When you press Enter again, the old question and response is cleared from the screen and a new question is asked. Lines 390 through 540 perform exactly the same function as lines 150 through 340 above. In all, you are asked four different questions, although this can be expanded upon by inserting more question and answer lines following program line 940.

As soon as the last question has been answered (right or wrong) you are asked to enter the number of answers you missed. This is assigned the variable E in line 980. Lines 990 through 1050 tests for the value of E. Lines 1040 and 1050 check for an inappropriate answer. If E is less than 1 or more than 4, there is a branch to line 970, where you are asked once again for the number you missed. Lines 1000 through 1030 branch to different program lines depending on the value of E. The scoring responses are found in lines 1100, 1070, 1130, 1160, and 1190. Lines 1200 and 1210 form a time delay loop which displays the scoring response on the screen for a few seconds. When the loop times out, the screen is cleared and the program ends.

Adding questions will involve more time at the computer keyboard. However, once a very long program has been typed in and debugged, it can be saved on cassette or disk and accessed almost immediately. One should not

look so much at the input time required as the pleasure or advantage of using and running a program which has already been stored.

## JULIAN DATE

The Julian date is the day of a 365- or 366-day year, expressed as a number rather than a standard calendar date. For example, calendar day February 1 is equivalent to Julian date 32. This is determined this way: the year begins with January 1, January has 31 days, and February 1 is one day past 31, or 32. December 31 will have a Julian date of either 365 or 366, depending on whether the year is a Leap Year.

This program will give you Julian date for any years from 1901 through 1999 based upon a calendar date input. You must enter the month, day, and year numerically: 10,31,52, for example. Each set of numbers must be separated by a comma. All numbers must be entered before pressing Enter. The program listing follows.

```
10 REM JULIAN DATE
20 REM THIS PROGRAM WILL OUTPUT THE JULI
AN DATE FOR ANY
30 REM DATE YOU INPUT
40 REM THE JULIAN DATE IS THE SEQUENTIAL
   DAY OF THE YEAR
50 REM WHICH ANY CALENDAR DATE REPRESENT
   S
60 REM COPYRIGHT FREDERICK HOLTZ AND
   ASSOCIATES 2/1/83
70 REM PROGRAM RUNS IN TI-BASIC
80 REM PROGRAM REQUIRES 1664 BYTES OF RA
   M
90 CALL CLEAR
100 INPUT"TYPE THE MONTH, DAY AND YEAR":M
   ,D,Y
110 IF Y>99 THEN 310
120 IF M>12 THEN 310
```



```

130 IF Y<=0 THEN 310
140 IF M<=0 THEN 310
150 IF D>31 THEN 310
160 IF M=2 THEN 170 ELSE 180
170 IF D>29 THEN 310
180 IF M=4 THEN 190 ELSE 200
190 IF D>30 THEN 310
200 IF M=6 THEN 210 ELSE 220
210 IF D>30 THEN 310
220 IF M=9 THEN 230 ELSE 240
230 IF D>30 THEN 310
240 IF M=11 THEN 250 ELSE 260
250 IF D>30 THEN 310
260 IF Y/4=INT(Y/4) THEN 370
270 IF Y/4<>INT(Y/4) THEN 280 ELSE 370
280 IF M=2 THEN 290 ELSE 370
290 IF D=29 THEN 310 ELSE 370
300 PRINT
310 PRINT
320 PRINT
330 PRINT"INCORRECT DATE!!!"
340 FOR GG=1 TO 800
350 NEXT GG
360 GOTO 100
370 IF M=1 THEN 730
380 IF M=2 THEN 710
390 IF M=3 THEN 690
400 IF M=4 THEN 670
410 IF M=5 THEN 650
420 IF M=6 THEN 630
430 IF M=7 THEN 610
440 IF M=8 THEN 590
450 IF M=9 THEN 570
460 IF M=10 THEN 550
470 IF M=11 THEN 530
480 IF M=12 THEN 510
490 PRINT
500 PRINT
510 JD=334+D
520 GOTO 740
530 JD=304+D
540 GOTO 740

```

```

550 JD=273+D
560 GOTO 740
570 JD=243+D
580 GOTO 740
590 JD=212+D
600 GOTO 740
610 JD=181+D
620 GOTO 740
630 JD=151+D
640 GOTO 740
650 JD=120+D
660 GOTO 740
670 JD=90+D
680 GOTO 740
690 JD=59+D
700 GOTO 740
710 JD=32+D
720 GOTO 740
730 JD=D
740 IF Y/4=INT Y/4 THEN 750 ELSE 800
750 IF M>=2 THEN 760 ELSE 800
760 IF M=2 THEN 770 ELSE 850
770 IF D=29 THEN 780 ELSE 800
780 JD=60
790 GOTO 860
810 PRINT "THE JULIAN DATE FOR";M;"/";D;"/";Y;"IS";JD
820 IF Y/4=INT Y/4 THEN 830 ELSE 840
830 PRINT "LEAP YEAR"
840 GOTO 880
850 JD=JD+1
860 PRINT "THE JULIAN DATE FOR";M;"/";D;"/";Y;"IS";JD
870 PRINT "LEAP YEAR"
880 END

```

Lines 110 through 150 test for an incorrect date entry. This program is limited to Julian dates in the 20th Century, starting with 1901 and ending at 1999. Line 110 tests for a condition of Y being equal to more than 99. If you enter a date of 10,31,500, there would be a branch to line 310 and a prompt that you entered an incorrect date would appear on the screen. The same branch will occur if the month numeral is equal to more than 12, the day is equal to more than 31, or other conditions specified in lines 110 through 250.

Lines 160, 180, 200, 220, and 240 test for other incorrect dates. (This involves the number of days possible in a given month.) Line 150 covers incorrect dates for all months: no calendar month can contain more than 31 days. Line 160 tests for the month of February alone ( $M = 2$ ). The maximum number of days February can have is 29 (Leap Year). If  $M = 2$  and  $D$  is more than 29, the incorrect date prompt will appear. Similar tests are made for April, June, September, and November in lines 180, 200, 220, and 240, respectively. Line 260 tests for a Leap Year. Since Leap Year occurs every four years, by dividing four into the date numeral, a Leap Year can be determined (if  $Y$  divided by 4 is a whole number ( $\text{INT } Y/4$ ), branch to line 370). Lines 270 through 480 tests for the value of  $M$  (month) and branch to other portions of the program to determine the

days in all preceding months (total) and the days which have passed in the specified month. Line 780 prints the Julian date for the entered calendar date on the screen. Line 800 prints LEAP YEAR.

### RANDOM PARTNER MATCH

This program is fun, but it can also serve a very worthwhile function. You can enter the names of up to 100 persons or objects of one sex or type and then up to 100 names of persons or objects of a different type or the opposite sex. The computer randomly pairs these persons or things. It's equivalent to drawing names out of two different hats, but the hats are arrays established in line 100. The first array contains the names of the girls, while the second contains the names of the boys. The program listing follows.

```

10 REM RANDOM PARTNER MAT
CHING
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/10/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 1280 BYTES OF RA
M
50 CALL CLEAR
60 GOSUB 480
70 CALL CLEAR
80 INPUT "HOW MANY COUPLES":NP
90 CALL CLEAR
100 DIM A$(NP),B$(NP)
110 FOR N=1 TO NP
120 PRINT "NAME OF GIRL"
130 INPUT A$(N)
140 CALL CLEAR
150 NEXT N
160 FOR N=1 TO NP
170 PRINT "NAME OF BOY"

```

```

180 INPUT B$(N)
190 CALL CLEAR
200 NEXT N
210 CALL CLEAR
220 RANDOMIZE
230 I(1)=INT(RND*NP)+1
240 FOR X=2 TO NP
250 L=INT(RND*NP)+1
260 FOR Y=1 TO NP
270 IF I(Y)=L THEN 250
280 NEXT Y
290 I(X)=L
300 NEXT X
310 J(1)=INT(RND*NP)+1
320 FOR Z=2 TO NP
330 K=INT(RND*NP)+1
340 FOR M=1 TO NP
350 IF J(M)=K THEN 330
360 NEXT M
370 J(Z)=K
380 NEXT Z
390 FOR X=1 TO NP
400 S1=I(X)
410 PRINT X;".";A$(S1)
420 NEXT X
430 FOR Z= 1 TO NP
440 S2=J(Z)
450 PRINT Z;" ". "B$(S2)
460 NEXT Z
470 END
480 PRINT"THIS PROGRAM WILL RANDOMLY MAT
CH ANYTHING. MOST OFTEN"
490 PRINT
500 PRINT"SUCH A PROGRAM IS USED TO MATC
H COUPLES...AGAIN, ON A RANDOM"
510 PRINT
520 PRINT "BASIS. SIMPLY INPUT THE TOTAL
NUMBER OF COUPLES INVOLVED"
530 PRINT
540 PRINT"IN THIS MATCH. THE COMPUTER WI
LL THEN ASK YOU FOR THE NAMES"
550 PRINT

```

```

560 PRINT"OF THE GIRLS AND THEN THE BOYS
. WHEN ALL NAMES HAVE BEEN"
570 PRINT
580 PRINT"INPUT, THE COMPUTER WILL THEN
DECIDE HOW THE MATCHES ARE"
590 PRINT
600 PRINT"TO BE MADE."
610 PRINT
620 PRINT
630 INPUT"PRESS <ENTER> TO BEGIN":AA$
640 RETURN

```

When you run the program, you will first be asked the total number of couples. This cannot be a figure higher than 100; and no more than 10 pairs can be displayed on the screen at any one time. The number of couples is assigned to the variable NP.

The screen is cleared in line 90 and the arrays are established. Lines 110 through 150 set up a For-Next loop which counts from 1 to NP. Each time the loop cycles, you are prompted to enter the name of a girl. No two entries should be the same name. When the loop times out, all the female names will have been entered, and a new loop is established in lines 160 through 200 to input the names of the boys. The only difference is in line 180, where the Input statement feeds the names to the array identified by B\$.

When all names have been entered, the screen is cleared and the Randomize statement is accessed in line 220. The RND function is used in lines 230 and 250, to randomly mix the contents of each array. Lines 260 through 380 sort the names in each array. Lines 390 through 460 print the names of the girls first, giving each a sequential number. Then the

names of the boys are printed, also with sequential numbers. The girl with number 1 is matched with the boy assigned number 1.

This is a handy program for grade school mixers, Sadie Hawkins dances, and the like. It can also be used experimentally to match study partners at random.

The program may also be used to match any type of objects or even farm animals. No matchings are made based upon what we would call true logic, although the pseudo-random numbers are arrived at through machine logic.

## ALPHABETIZING PROGRAM

Whereas Random Partner Match matched items in two arrays according to random selection, this program arranges all items in alphabetical order. This is a true alphabetizing program that will let you enter up to 100 items in any order. It will then alphabetize the items, and upon command, print the alphabetical listing on the screen. The alphabetical listing is printed horizontally on the screen with each word separated by commas. The program listing follows.

```

10 REM ALPHABETIZING PROGRAM
20 REM PROGRAM RUNS IN TI-BASIC

```

```

30 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/12/83
40 REM PROGRAM REQUIRES 768 BYTES OF RAM

50 CALL CLEAR
60 PRINT"ENTER EACH WORD AS REQUESTED; W
HEN COMPLETE,TYPE 'END.'"
70 PRINT
80 PRINT"ENTER UP TO 100 WORDS"
90 PRINT
100 PRINT
110 DIM A$(100)
120 DIM J$(100)
130 PRINT"TYPE WORD TO BE ALPHABETIZED:
"
140 I=I+1
150 INPUT A$(I)
160 IF A$(I)="END" THEN 180
170 GOTO 140
180 N=I-1
190 FOR I=1 TO N
200 J$(I)=A$(I)
210 NEXT I
220 CALL CLEAR
230 FOR I=1 TO N
240 PRINT A$(I);", ";
250 NEXT I
260 PRINT
270 PRINT
280 PRINT"PRESS (ENTER) TO ALPHABETIZE"
290 INPUT Z$
300 CALL CLEAR
310 PRINT"COMPUTING ---PLEASE STAND BY"
320 FOR P=1 TO N-1
330 FOR I=1 TO N-P
340 IF J$(I)<= J$(I+1) THEN 380
350 X$=J$(I)
360 J$(I)=J$(I+1)
370 J$(I+1)=X$
380 NEXT I
390 NEXT P
400 CALL CLEAR

```

```

410 FOR I=1 TO N
420 PRINT J$(I);", ";
430 NEXT I
440 END

```

When the program is run, the screen is cleared and you are prompted to enter each word as requested. You are also instructed to **END** when your list is complete. This is the only word that cannot be alphabetized; if you anticipate using this word in an alphabetical list, you can change this word to any other key word of your choice to mark the end of your list. If you change this word in line 60, be certain to change it in line 160 as well.

Two arrays are established in lines 110 and 120. Each may have a maximum of 100 elements. Line 130 causes the prompt to appear indicating that you are to type the words to be alphabetized. Type one word at a time, and then press Enter. A loop is established by the branch statement in line 170, allowing you to enter as many words as you want until your list is complete. Each time the loop cycles, the value of I in line 140 increases by 1, providing a count of the number of words entered. When you type **END**, line 160 detects this and branches to line 180. This assigns a value to N, which is equal to I - 1. This will be the total number of words entered for alphabetizing. It is necessary to subtract 1 because entering **END** caused I to increase by 1.

A loop is established in lines 190 through 210 and transfers all the items in the first array to the second array identified by J\$. Lines 230 through 250 form another loop and print out the words you entered in the order in which they were entered. If you forgot to include something, you may detect it during this printout

and start over again. Once you have typed **END**, you cannot go back and add words. At this point, you are prompted to press Enter to alphabetize the list.

The sorting process may take some time, so line 310 prints a prompt on the screen telling you the machine is computing, please stand by. This lets the operator know the machine hasn't shut down. Long lists may take several minutes to sort fully.

Lines 320 through 390 form a loop within a loop (nested loop) to compare the values of J\$. Computers automatically arrange items alphabetically or at least think of them alphabetically. For example, an A is given a smaller value than B, just like 1 is given a smaller value than 2. Using this principle, it's not too difficult to enter words at random and then have them print out alphabetically. The sorting procedure takes place in lines 320 through 390. Line 340 tests for the condition of an item being equal to or less than another item in an array. If this occurs, there is a branch to line 380, which causes the nested loop to cycle once again. The value of each item in the array is temporarily assigned to X\$ in line 350. The value of X\$ is then read back into the J\$ array in alphabetical order. When all the sorting is complete, the loop times out.

At this point, line 400 clears the screen, and another loop is begun in line 410. This loop pulls the information stored in the J\$ array, which is now in alphabetical order. The Print statement in line 420 displays this information

on the screen followed by a comma. The loop recycles and the next item in J\$ appears, until J\$ has been emptied and the complete list is shown in alphabetical order.

To test this program, run it and type all the letters from the keyboard into the array one at a time. Do not type them in alphabetical order. When you reach the last letter, type END and press Enter. After a few minutes you will get a printout of A,B,C,D,E,F,G, etc. This indicates that the program is working perfectly. You may also enter numbers in any order. They will ultimately be listed in ascending order, starting with the smallest number and ending with the largest number in the array.

If your computer is equipped with a printer, this program can be useful when indexing books, and to provide other hard-copy printouts of alphabetical listings.

## MATH PRACTICE

A computer is a mathematical device, and while most of its mathematical operations on the machine level are carried out in binary arithmetic, it is quite easy to convert this information to the decimal system. This conversion is actually done almost automatically by the machine. This can provide a very capable mathematics drill routine which will delight youngsters and challenge even the best mathematical minds.

The program is designed to appeal to a broad range of individuals from high school level on. The math problems are selected at random. This applies to the numbers used and the mathematical operations to be performed with them. All problems deal with two separate numbers.

The program listing for Math Practice follows.

```
10 REM MATH PRACTICE
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/7/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 1408 BYTES OF RA
M
50 CALL CLEAR
60 PRINT"THIS PROGRAM WILL PROVIDE YOU W
ITH A CONSTANT ARRAY"
70 PRINT
80 PRINT"OF MATH PROBLEMS WHICH INVOLVE
ADDITION,SUBTRACTION,"
90 PRINT
100 PRINT"DIVISION, AND MULTIPLICATION.
EACH PROBLEM WILL BE"
110 PRINT
120 PRINT"DISPLAYED ON THE SCREEN AND YO
U WILL HAVE THREE CHANCES"
```



```

130 PRINT
140 PRINT"TO ARRIVE AT THE CORRECT ANSWER. IF YOU CAN'T FIGURE"
150 PRINT
160 PRINT"OUT THE CORRECT ANSWER, THE COMPUTER WILL SUPPLY IT"
170 PRINT
180 PRINT"AFTER THE THIRD WRONG ANSWER."

190 PRINT
200 PRINT
210 INPUT"PRESS ENTER TO CONTINUE":AA$
220 CALL CLEAR
230 RANDOMIZE
240 X=INT(RND*100)+1
250 Y=INT(RND*100)+1
260 Z=INT(RND*4)+1
270 IF Z=1 THEN 290 ELSE 320
280 REM OPERATIONAL SEQUENCE
290 I=X+Y
300 A$="+"
310 GOTO 420
320 IF Z=2 THEN 330 ELSE 360
330 I=X-Y
340 A$="-"
350 GOTO 420
360 IF Z=3 THEN 370 ELSE 400
370 I=X/Y
380 A$="/"
390 GOTO 420
400 I=X*Y
410 A$="*"
420 CALL CLEAR
430 PRINT X;A$;Y;"=";
440 INPUT Q
450 PRINT
460 PRINT
470 IF Q=I THEN 480 ELSE 560
480 CALL CLEAR
490 PRINT Q;"IS THE CORRECT ANSWER"
500 PRINT

```

```

510 PRINT
520 INPUT "PRESS ENTER TO CONTINUE":AA$
530 W=0
540 CALL CLEAR
550 GOTO 240
560 CALL CLEAR
570 W=W+1
580 IF W=3 THEN 620
590 PRINT "THAT IS AN INCORRECT ANSWER--
PRESS ENTER TO TRY AGAIN"
600 INPUT AA$
610 GOTO 420
620 PRINT "THAT'S THREE WRONG GUESSES. TH
E ANSWER IS";I
630 PRINT
640 PRINT
650 INPUT "PRESS ENTER TO CONTINUE":AA$
660 W=0
670 CALL CLEAR
680 GOTO 240

```

The first eighteen lines tell you about the program through REM statements and also print a limited set of instructions on the screen. In line 210, you are asked to press Enter to continue. At this point, the screen is cleared and the Randomize statement in line 230 reseeds the random number generator.

Lines 240 through 260 determine the problem. Lines 240 and 250 assign a random value to X and Y which can be any number between 1 and 100. Due to the use of the INT function, the assigned value of either of these values will always be an integer. The RND function makes it impossible to predict what the two numbers will be.

Line 260 also uses the RND function, where, variable Z is assigned a value which may be equal to 1, 2, 3, or 4. The value of Z will determine the mathematical operation to be

performed on or with the numbers assigned to X and Y.

Lines 270 through 410 make the operational assignments based upon the value of Z. In line 270, there is a test for Z equal to 1. If Z is equal to 1, the two numbers assigned to X and Y will be added together, and I is equal to the sum. In line 300, the string variable A\$ is assigned the plus sign, indicating an add operation. There is then a branch to line 420, where the screen is cleared and the computer prints the problem in line 430.

Line 430 tells the computer to display X followed by A\$, which in this case is a plus sign, followed by variable Y and an equal sign, which is enclosed in quotation marks. A semicolon is placed after the last quotation mark. This works with the Input statement in line 440 to let the answer you type in appear to

the right of the equal sign. Without the semicolon at the end of line 430, your answer would appear in the line below the problem. In this particular case, the problem might look like:

$$40 + 62 = ?$$

The question mark is automatically displayed through the use of the Input statement.

You now have three chances to come up with the correct answer. The information you enter is committed to the numerical variable Q. When you have typed your answer and pressed Enter, line 470 tests for the condition of Q being equal to I. I represents the correct answer. If Q is indeed equal to I, there is a branch to line 480, which clears the screen, prints the answer, and prints a prompt telling you the answer is correct. You are then asked to press Enter to continue and the program starts again.

If Q is not equal to I, there is a branch to line 560. The screen is cleared, and a count line is activated in line 570. The variable W will count the number of incorrect answers based on the number of times this routine is entered. The first incorrect answer will cause the INCORRECT prompt to appear on the screen, and you will be asked to press Enter to try again. If the next answer is wrong, this same program section will be entered, and the value of W is advanced to 2. On the third incorrect answer, W is equal to 3; line 580 detects this

condition and branches to line 620. The screen then announces that you have three wrong guesses and tells you the correct answer. You are asked to press Enter to continue with a new problem. When you press Enter, line 660 returns the value of W to 0, the screen is cleared, and there is a branch to line 240 to access a new problem.

When a correct answer is given after one or two incorrect answers, you will notice that the value of W is also returned to 0 in line 530. It is always necessary to return W to 0 before beginning a new problem. If you would prefer to allow for fewer or more incorrect guesses, change the If-Then statement in line 580 to reflect the number of guesses you prefer.

This is a very interesting game, and the problems range from very easy to moderately difficult. Some of the problems can be done in your head, but many will require pencil and paper or even a calculator. You can make the problems more difficult by assigning another random number to a variable which can be used in the problem displayed. You can arrange many different problem formats by adding just a few lines.

## STEPPING SOUNDS

The Call Sound subprogram can be used to generate simple songs, sound effects, and fairly complex scales. The program listing that

```
10 REM STEPPING SOUNDS
20 REM PROGRAM RUNS IN TI-BASIC
30 REM COPYRIGHT FREDERICK HOLTZ AND ASSOCIATES 2/7/83
40 REM PROGRAM REQUIRES 256 BYTES OF RAM
50 FOR X=500 TO 1000 STEP 10
60 CALL SOUND(50,X,0,X-300,0,X+300,0)
70 NEXT X
```

follows will take you only a couple of minutes to enter and will demonstrate how a musical scale may be created using For-Next statements and the Call Sound subprogram.

Line 50 establishes the parameters of the For-Next loop. It will start at 500 and advance to 1000 in steps of 10. The Call Sound subprogram is found in line 60. Three frequency commands are included so the output from the speaker will be three separate, but simultaneous, tones. The value of X is used to determine the frequency of the tones.

Looking at line 60, you will see a 50 as the first number within the frequency, represented by the variable X. During the first cycle of the loop, X will be equal to 500, so the first tone will be 500 Hertz. This frequency numeral is followed by a 0, which accesses the loudest volume possible. The next frequency command is also determined by X, but 300 is subtracted from the value of X. During the first cycle of the loop,  $X - 300$  will equal 200, and a 200-Hertz tone will be output. Following another volume command, the value of X is *added* to 300, so the third tone during the first cycle will be 800 Hertz.

When this program is run, you will hear three simultaneous tones. One will be 300 Hertz below the next, and the following one will be 300 Hertz above the center frequency. This provides a combination of 200, 500, and 800 Hertz tones during the first cycle of the loop and 700, 1000, and 1300 Hertz during the last loop cycle. This provides an interesting audio effect, and you can experiment with this program by changing the step values and the duration command in the Call Sound subprogram.

### STEPS OF THIRDS

This program is similar to Stepping Sounds above. However, the musical notes will step from 130 Hertz to 1100 Hertz in increments of 30. The Call Sound subprogram in line 60 uses three values of X roughly spaced by musical thirds. Again, three simultaneous tones will be output. The first is the value of X, the second is the same value times 1.26, while the third is 1.26 times the latter value. This separates all three notes by rough thirds, and you will hear a chord step routine as the loop advances. The program listing follows.

```
10 REM STEPS OF THIRDS
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/7/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 256 BYTES OF RAM

50 FOR X=130 TO 1100 STEP 30
60 CALL SOUND(50,X,0,X*1.26,0,(X*1.26)*1
.26,0)
70 NEXT X
80 END
```

## KEYBOARD

Here is a simple program that will turn your TI-99/4A into a basic keyboard. When using the Sound subprogram, it is necessary to specify frequency in Hertz. This one lets you enter a musical note, such as A, B, C, etc., and have that frequency output from the speaker. The program listing follows.

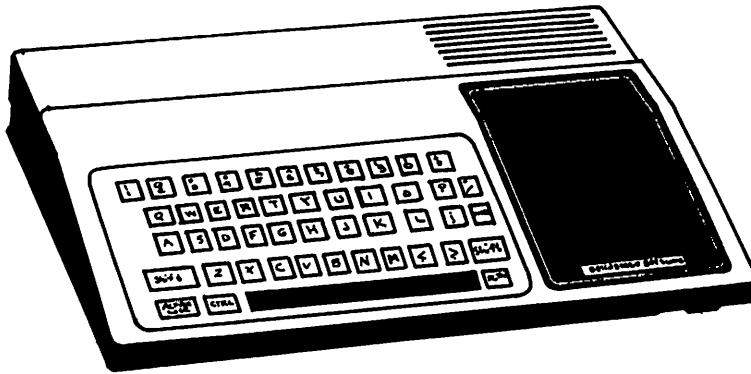
Line 60 lets you enter note A, B, C, D, E, F, or G. The note is assigned to the string variable A\$. Lines 70 through 240 test for the value of A\$ and assign frequencies which correspond to the entered note. If the note is A, this is detected in line 70, which branches to line 80. Here a value of 440 is assigned to X. The musical note middle A is a tone of ap-

```
10 REM KEYBOARD
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 2/7/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM PROGRAM REQUIRES 640 BYTES OF RAM
50 CALL CLEAR
60 INPUT"TYPE IN NOTE A,B,C,D,E,F, OR G"
:A$
70 IF A$="A" THEN 80 ELSE 100
80 X=440
90 GOTO 290
100 IF A$="B" THEN 110 ELSE 120
110 X=493
115 GOTO 290
120 IF A$="C" THEN 130 ELSE 150
130 X=523
140 GOTO 290
150 IF A$="D" THEN 160 ELSE 180
160 X=587
170 GOTO 290
180 IF A$="E" THEN 190 ELSE 210
190 X=659
200 GOTO 290
210 IF A$="F" THEN 220 ELSE 240
220 X=698
230 GOTO 290
240 IF A$="G" THEN 250 ELSE 270
250 X=783
260 GOTO 290
270 PRINT"ILLEGAL NOTE!!!"
280 GOTO 60
290 CALL SOUND(1000,X,0)
300 GOTO 50
```

proximately 440 Hertz. There is then a branch to line 290, where the Call Sound subprogram is used to play middle A for approximately one second. Line 300 branches back to the beginning of the program, letting you enter other notes. If you enter a note outside the range of this program, an **ILLEGAL NOTE** prompt will

occur, and you will be given the opportunity to try another. This program will not let you use your TI-99/4A like a piano or organ keyboard, but it does demonstrate a method by which musical manuscript may be programmed into the computer once a routine has been set up to assign frequencies to musical note values.

## Chapter 9



# Other Programming Languages

Your computer standardly uses TI BASIC as its programming language. It is capable of being programmed in other languages as well, and optional language packages are available from the Texas Instruments Software Library.

### TI EXTENDED BASIC

This is a language option many people will select. It is not a new language so much as an extension of the resident BASIC language found in the TI-99/4A ROM. All the programs in this book run without Extended BASIC. However, this language extension package is highly desirable, and sooner or later most users will purchase it.

TI Extended BASIC is available on a plug-in command module. Extended BASIC is a powerful, high-level programming language offering a large number of features not available in standard TI BASIC.

One limitation I found in TI BASIC lay in the fact that multiple statement lines are not permitted. Most dialects of BASIC allow the use of multiple statements on single lines, separated by colons, slash bars, or some other keyboard symbol. When two statements are combined in a single line, they use less memory space than the same statements input in two separate lines. From an execution standpoint, multi-statement lines and single-statement lines result in the same program run. When writing very large programs, some of which will tax your resident memory to the utmost, a language which allows multi-statement lines can mean the difference between a successful run and running out of memory.

The If-Then-Else statement in Extended BASIC has been greatly expanded as compared

to the version in TI BASIC. The expanded statement lets you immediately execute a statement based on the results of the comparison instead of requiring you to branch to another line. In other words, the program line, **40 IF X = 10 THEN PRINT "YOU ARE A WINNER" ELSE PRINT "YOU LOSE"** which is illegal in TI BASIC would be legal in Extended BASIC.

In TI BASIC, the If-Then-Else statement must be used to bring about a branch to another portion of the program. In Extended BASIC, however, certain statements may be executed depending on the results of the If-Then comparison. This decreases the number of program lines required and can take greater advantage of the on-board memory. The expanded If-Then-Else statement also provides more flexibility by allowing the addition of And, Or, XOR, and Not to the comparison.

In Extended BASIC, comments may be accepted on the same line on which a statement occurs without requiring another line number. Multiple assignments may be made: a value may be assigned to more than one variable.

There are additional commands and statements as well. The Size command tells you how much memory remains unused in the computer (RAM). There is an automatic load which can load and run programs automatically as soon as the computer is turned on. This may be referred to as a boot load.

Several commands have been expanded in Extended BASIC. These include Run and Save. The expanded Run command lets you specify which program to run, so that one disk program can load and run another disk program (called program chaining). The expanded Save

command protects a program to prevent unauthorized listing, editing, or copying. When this Save Protect is initiated, the program information is stored in a compressed format and cannot be listed on a line-by-line basis.

Another important statement added by Extended BASIC is Merge. This lets you combine programs or subprograms stored on disk with the program already in memory. There is also a List Pause feature, which will stop and start a program listing in this Extended form of BASIC. And you can write subprograms with parameter lists and local variables.

For the graphics programmer, TI Extended BASIC offers Sprites. A Sprite is a point of light on the screen, with which you can define up to 28 different color graphics which move smoothly on the screen through computerized animation. Each Sprite's definition can also include size, position, speed, and direction.

TI Extended BASIC also has built-in error handling. This lets you determine the action taken when a minor error, a major error, or a breakpoint is encountered. The Display statement is complemented by an Accept statement in Extended BASIC. This lets you display any input data at any position on the screen. This is probably one of the most important features of Extended BASIC, allowing the screen to be formatted under the programmer's direction.

TI Extended BASIC will also let you use the memory expansion unit, giving you additional memory and accessory capabilities. This package also supports the loading and running of TMS9900 assembly language programs if the memory expansion unit is attached and activated.



## ASSEMBLY LANGUAGE

Assembly-language routines may be entered and assembled using the TI-99/4A Editor/Assembler software. This following discussion provides an overview of the capabilities, how they are accessed from TI Extended BASIC, and how assembly routines may be developed.

Assembly language routines called from Extended BASIC may be used to make algorithms execute more rapidly, or to provide complex control of the video screen or sound chip. Assembly routines are normally loaded in a relocatable form into a dedicated 8K byte block of the memory expansion. Using relocatable code lets routines be written and used independently of the actual loaded address in memory. Also, combinations of several routines may be used from one BASIC program without concern for the load address. Assembly routines are called from the BASIC program by name. The actual address of the routine is resolved during execution. One or more entry names for a routine are defined when the routine is written.

The 8K-byte reserved block of the memory expansion may be used by assembly language. In addition, a portion of the other 24K bytes in the Memory Expansion is available depending on the size of the BASIC program being run. An assembly routine may or may not return control to BASIC. It may take control of the TI-99/4A and act as an entire application. In this instance, the entire memory resources of the computer are available for use by the assembly subprogram.

A set of utility routines is provided for assembly language subprograms to easily access unique hardware resources such as video

display processor, and to pick up the values of arguments or return values to the BASIC program. A set of subroutines may be linked with an assembly routine to provide access to peripherals. With these routines, an assembly subprogram may easily execute I/O to a disk, RS-232, or other peripheral.

Three statements in the Extended BASIC language (Call INIT, Call Load, and Call Link) are provided to support the assembly language capability.

The BASIC subprogram for Call INIT is used to initialize the RAM expansion for assembly subprograms. This subprogram ensures that the memory expansion has been attached. It then loads a set of utilities from ROM memory in the Extended BASIC command module into the memory expansion. Call INIT also causes any previously loaded assembly routines to be undefined. Any subsequently loaded routines will remain defined until another Call INIT is executed or the memory expansion is turned off.

The BASIC subprogram Call Load is used to load an assembly language object file into the memory expansion. Alternately, direct data may be specified in the form of addresses and data in the Call Load statement. An object file may contain relocatable or absolute address code. Direct data may only be absolute. Because of the difficulty in defining the entry point name from direct data, it is not recommended for general use when loading an entire program. However, if this is necessary, technical assistance can be provided from TI to help accomplish this. Examples of Call Load statements follow:

**CALL LOAD("CS1")**

```
CALL LOAD("DSK1.MYOBJECT",  
"DSK1.OBJECT2")  
CALL LOAD(12000,04,01,02,0,255,  
255)
```

The first two are examples of loading from a file, while the third uses direct data.

The BASIC subprogram Call Link is used to link to an assembly language subprogram. This statement specifies the routine to be called and any arguments to that routine. For example:

```
CALL LINK("NSORT",A(),)
```

could be written to sort array elements in row A(2,n) so elements in row A(1,n) are in ascending order. In this example, NSORT is the name of the routine. One argument is passed as two-dimensional array A. Values in this array are changed and returned to the BASIC program.

Development of assembly language programs to run under Extended BASIC may also be done using the PASCAL development system. This system includes a powerful text editor, macro assembler, and link editor. A set of macros is provided during assembly to facilitate accessing specified hardware features, such as the video display processor or the sound chip. An I/O utility is available which may be linked with an assembly routine to provide access to peripherals such as disk drive, RS-232, etc. Although the disk format of the PASCAL system is different from that used by Extended BASIC, a utility is provided to convert a PASCAL object file to the format required by Extended BASIC.

## TI LOGO

While assembly language is an exercise

for the highly serious computer programmer, Texas Instruments offers another language perfect for teaching children. Called TI LOGO, it is advertised as a child-appropriate computer language. Texas Instruments explains that the role of computers in education has traditionally cast the computer as the teacher and the student as a passive learner of the computer's lesson. While this approach is useful for teaching a prescribed curriculum, it does little to promote a feeling for what computers can and cannot do. TI LOGO is an innovative approach which not only develops computer awareness, but enriches a child's math, logic, and communication skills as well.

TI LOGO is a derivation of LOGO, a computer language based on a philosophy of education developed over a 12-year period by Professor Seymour Papert and the staff of the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT). The core principle of LOGO is to create computer-based environments in which mathematics and other areas of format learning can occur in a natural manner. Many of LOGO's premises are based on Jean Piaget's theory of intellectual development, which describes a child's development as taking place in a series of stages. TI LOGO, which is the result of collaboration between the MIT staff and Texas Instruments, is the first implementation of LOGO on a low-cost microcomputer system—the TI Home Computer.

For computers to become a valuable learning tool, the computer must understand a language a student can easily learn, and the computer must be able to do something for the student immediately. TI LOGO is a child-appropriate computer language, which means

it lets students of all levels of ability communicate with the computer using an easy-to-understand language. With its Sprite, Tile, MAKESHAPE, MAKECHARacter, and Turtle Geometry capabilities, TI LOGO places children in a virtually unlimited creative environment.

TI LOGO accomplishes specific tasks using an easy-to-follow, step-by-step approach. The student must first teach the computer what to do and then tell it how to do it. This reverses the role of the student, placing him or her in the position of teacher, at least to the computer. It lets students determine a level of challenge they want to explore in areas of problem solving and communications. The computer enables students to pace their levels of achievement. It gives immediate action no matter how fast or slow a student works. As it is always ready and waiting for the student's next step, the computer constructs an appropriate environment for creating a positive self-image. From this point of view, learning occurs naturally as the student works through the skills necessary to accomplish a task. This step-by-step discovery method of learning encourages the student to gain control over the learning process, in addition to helping to make learning a fun and satisfying experience.

With the help of Turtle Graphics, students can learn a computational style of geometry by acting out the role of the Turtle. The commands in TI LOGO for controlling the movement of the Turtle on the drawing surface of the display are Forward (a number of Turtle steps), Back (a number of Turtle steps), Right (a degree of turn), and Left (a degree of turn). A student thinks of a design. He or she acts it out, draws it, or just tells the Turtle how to do it.

The commands can be given one at a time or put into a procedure and named. From this simplistic stage, a student can progress to more complicated designs.

Suppose a house was the next project. A student divides the house into a square and a triangle. The steps for drawing the components of the house then become subprocedures of the procedure House. Calling a procedure by its name is an example of the process of communication. The name represents shared knowledge between the student and the computer.

The procedure can be saved on a disk or cassette tape and re-used just by recalling it by name.

For all students, even those with learning disabilities, TI LOGO serves as an unhurried guide to a world of animated shapes. The student creates and designs this world by using the computer's 32 Sprites. A Sprite is an undefined space on the display that carries any shape. The shape can be one the computer knows (Plane, Rocket, Truck, Ball, or Box) or any shape a student designs with MAKESHAPE. The Sprites carrying a shape can then be given other features—color, speed, heading, and velocity.

Sprites can be used in simple procedures, like one that makes a black truck move across the display, or advanced procedures that actually test a situation. The computer has a special role in education, as it can help to make practicable new styles of teaching and learning, not only for classroom situations, but also for education in the home and throughout life.

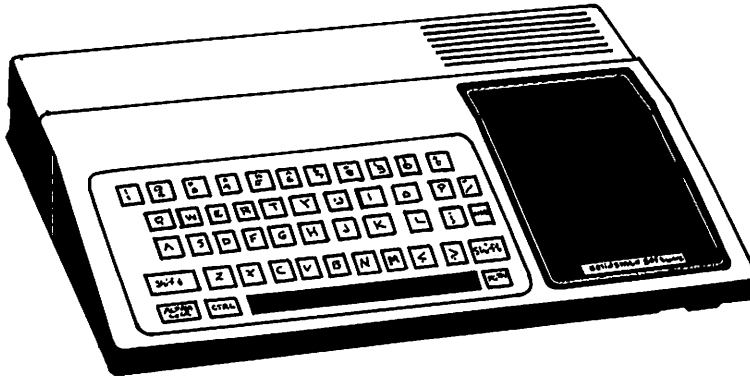
In order to fully utilize TI LOGO, you will need the TI-99/4A with a color monitor or color television receiver (RF modulator re-

quired). You will also need the Memory Expansion Unit. To take advantage of TI Extended BASIC all that is required is a standard TI-99/4A home computer. The Extended BASIC module plugs into the ROM expansion slot. Price of the Extended BASIC module is

approximately \$100.

There is a surprising amount of software available for the TI-99/4A. The TI-99/4A has some very powerful language packages which, in a home computer, are the exception rather than the rule.

## Chapter 10



# Software

The TI-99 home computer has been available since the late seventies; the newer TI-99/4A model was released in 1982. This computer has always enjoyed popularity, an advantage from the software standpoint. Texas Instruments and other software manufacturers offer a wide variety of programs for this machine in preprogrammed modules (which plug into a slot on the computer console), cassette tapes, and disks. Some software is available in all three media. A complete software directory is available for \$5.95 by calling (800) 858-4565 or by writing to Texas Instruments, Customer Relations, P.O. Box 53, Lubbock, Texas 79408.

This chapter overviews many of the programs offered in ready-to-run form on your TI-99 computer.

When consulting this chapter, keep in

mind several guides: the programs have been listed under general headings to make your search easier. Many programs, however, serve several purposes, but are listed only once. A word-processing program, for instance, will be listed under the Business section, but is also useful in Home/Personal applications and Educational applications. If you don't find something to suit your need under one heading, check the others. All prices are approximate and printed only to give you an idea of the price range.

### PROGRAMMING AIDS

The following programs will help you become a better programmer and user:

**Programming Aids I** is for programmers with some knowledge of TI BASIC who wish to go further. Many features are included

in this package, such as disk cataloging, screen print, lowercase letters, second ASCII set sub-routines, and user-defined characters. This package is available on disk (PHD 5004) and on cassette (PHT 6004) from Texas Instruments, for \$15.00 and \$10.00, respectively.

**Programming Aids II** takes the experienced programmer further by introducing the ability to sort information in different formats, such as disk sort, RAM sort, etc. Also included are a file dump and a merge program. It is available on disk only (PHD 5005) from TI, for about \$25.00.

**Programming Aids III** is a further educational package in the art of computer programming for the experienced computerist. It introduces such useful functions as allowing the programmer to cross-reference variables, arrays, keywords, functions and line number references. You may also resequence, delete program parts, and merge a sequence of code. The package is useful for advanced programming applications and sells for \$20.00 on disk (PHD 5012) from TI.

**Course Designer Authoring System** consists of a series of Extended BASIC programs that let the inexperienced programmer design lessons on the TI home computer. CDAS requires absolutely no previous programming experience, and consists of menus and user-friendly prompts that will guide a training specialist every step of the way. This package requires Extended BASIC and a disk drive and controller. The TI Impact Printer, RS-232 Interface, and video controller may be used optionally. It is available on disk (order PHD 5068) from TI for \$49.95

## Improving Your Computer

**Terminal Emulator II**, a plug-in module, directly links you to information services and time-sharing computer systems. Some of the many features of this package include file transfer with error definition, improved keyboard interface, the ability to hand color, speech, sound and graphics, and access of text to speech from user-written programs. This package from TI is priced at \$50.00 (PHM 3035).

**The Mini-Memory Command Module** makes it possible to increase the total memory of your system by 14K, composed of 6K of graphics read-only memory, 4K of read-only memory, and 4K of RAM. The package also contains additional tools for program development and writing. This module has its own battery, allowing you to store programs in the 4K of RAM retained in this memory even when the main computer is shut down. The package from TI is priced at \$100.00 (order PHM 3058).

## Special Effects

**Statistics** is designed to perform a wide variety of statistical calculations quickly and accurately, leaving you with more time to analyze and apply the results. This package may be used to output linear regression analysis, descriptive statistics, correlation, and other types of inferential evaluations. A data storage system is recommended for use with this program, which is available from TI as a plug-in module (PHM 3014) for \$45.00.

**Video Graphs** introduces you to the world of graphics on the TI-99/4A by showing

how to design pictures and patterns. This program lets you create your own designs by interacting with preprogrammed graphics in full color. Video Graphs is available in Module form only and sells for approximately \$20.00 from Texas Instruments (order PHM 3005).

**Graphing Package** provides plotting techniques, such as scatter plot, Cartesian and polar plots, and XYZ plots. This is designed for users interested in the art of making graphs in computerized form. It is available from TI either on disk (PHD 5013) or cassette (PHT 6013) for \$20.00 and \$15.00, respectively.

**Text-To-Speech (English)** requires the TI Speech Synthesizer, and is available on disk only. It lets you input English words via the keyboard. The words are then output via the synthesizer. This package teaches the user to construct words, phrases, and sentences using various pitch contours, inflection, pauses, etc. You learn to create sentences in different dialects of English. The package requires Extended BASIC, 32K memory, disk drive and controller. It is priced from TI at about \$30.00 (order PHD 5075).

**Speech Editor** will introduce the user to the many capabilities of the speech synthesizer from Texas Instruments. It lets you experiment by outputting words, phrases, and sentences entered via the keyboard. This is an excellent introductory lesson in the art of speech generation. It is available as a plug-in module from TI (PHM 3011) for approximately \$50.00.

### **Demonstration Programs**

**Music Maker Demonstration** is de-

signed for use with the Music Maker Solid State Command Module. This program is a demonstration package that illustrates the many types of musical compositions possible on the TI-99/4A. It contains five music files, with a wide variety of song types, including Christmas carols, childrens' songs, and even classical pieces. Available from Texas Instruments on disk only, it is priced at \$15.00.

### **Languages**

**Teach Yourself BASIC** is designed to teach the user how to program in BASIC, with the computer as teacher. Included in this package are on-screen lessons to provide experience in commands, graphics, and sound. The program was developed by Wolfdata Corporation and sells for approximately \$35.00 in disk form and \$30.00 in cassette form.

**The Extended BASIC** plug-in module provides you with the many features of this advanced programming language. It includes direct screen accessing, output formatting with using clause, multi-statement lines, If-Then-Else statements, subprograms, booting, control of up to 28 sprites from BASIC, chaining, merging, protected programs and Boolean functions, including And, Or, Not, etc. This package is highly recommended for the more experienced computerists who wish to expand their use of the BASIC language. It is priced at \$100.00 from TI (PHM 3026).

**Teach Yourself Extended BASIC** will take you through a complete education into Extended BASIC in a progressive manner. It teaches everything that can be done in Extended BASIC in individual chapter lessons.

Extended BASIC is required to use this package, which is available on disk (\$25.00) or cassette (\$20.00) from Texas Instruments.

**TI Pilot** is a comprehensive package that enables you to program in TI Pilot, a language designed for Computer Assisted Instruction (CAI). With this language you can perform many operations, including the development of interactive programs to teach student specific subjects, develop drill programs to reinforce concepts, develop test programs, store student information, etc. The package includes graphics, sound, and speech capability. Sample programs and the TI Interpreter are included in this package. A disk controller, disk drive, 32K memory expansion and TI P-Code peripheral or card are required. The synthesizer may also be used, although it is not mandatory. The package is available on disk from TI (PHD 5066) for \$80.00.

**UCSD Pascal Version IV.0** is a highly structured and efficient programming language. UCSD Pascal and UCSD p-System are trademarks of the Regents of the University of California. Pascal is a faster and substantially more powerful language than BASIC. With this package you can write highly efficient programs on the TI-99/4A, and run many existing Pascal programs with little or no modification. The UCSD Pascal compiler requires 32K of memory and a P-Code card which fits in the peripheral expansion system. You will also need a disk drive and controller. The compiler itself sells for approximately \$125.00 on disk.

**Editor Assembler** is available in command module form for about \$100.00. The Editor Assembler lets you program the TI-99/4A in TMS 9900 assembly language. This enables direct access to all machine features,

including speech, graphics, sound, and I/O. The assembler also provides the fastest speed possible from the 16-bit microprocessor. Assembly language routines can be run either as complete programs or linked into TI BASIC or TI Extended BASIC programs. While the Editor Assembler comes in command Module form, the package also includes two disks, and an owner's manual to provide documentation of the software's architecture. The Editor Assembler package also includes the source and object code for an interactive assembly language debugger and for a computer game called Tombstone City. System requirements are 32K RAM, the peripheral expansion system, disk drive and controller.

## ENGINEERING AND MATH LIBRARIES

The following library programs will be useful to programmers and users with special needs.

**The Ac Circuit Analysis Library** is ideal for electronics technicians who must design and build circuits to certain specifications. Two programs are included to aid in determining how a circuit will perform using alternating current. The circuit is analyzed in the first program, and the second provides a plot of the circuit. Standard components are allowed, including resistors, capacitors, inductors, and voltage-controlled current sources. Analysis may be provided in a number of different ways, and the analysis can be printed or saved for plotting on the monitor display. You can even make multiple passes, varying circuit parameters each time, to experiment with design specifications. The TI Impact Printer and RS-232 may be used with this package, although they are not necessary for operation. The pro-



gram is available on disk (order PHD 5044) for about \$30.00 from TI.

**The Math Routine Library** contains a wealth of reference information that may be used in a wide variety of mathematical applications. Included in this package are Fourier series calculations, function analysis, ordinary differential equations, base conversions, prime factorization, hyperbolic functions, and other calculations. The package is available either on disk (PHD 5006) or cassette (PHT 6006) from TI for \$25.00.

**Electrical Engineering Library** is a computerized reference source for electrical engineers. It includes root locus, filter design, Smith chart, phase-locked loop, and a myriad of other tools designed specifically to aid persons involved in electrical engineering. It is available on disk (PHD 5008) or cassette (PHT 6008) from TI for approximately \$30.00.

**Structural Engineering Library** is designed to aid the structural engineer in calculations associated with the job. It allows the user to make complex calculations and evaluations in a fraction of the time normally associated with such computations. The package is available on disk (PHD 5016) from TI for \$30.00, or on a cassette (PHT 6016) for \$25.00.

**SMU Electrical Engineering Library** was developed by Southern Methodist University for Texas Instruments, and is designed to aid electrical engineering students. It may be used both by instructors and students and is based around standard electrical engineering concepts. Three basic laws are taught—Ohm's Law, Kirchhoff's Current, and Kirchhoff's Voltage. The package consists of ten lessons, and may be used in combination with any college-level textbook on introductory circuit analysis.

It may be purchased in command module/disk form (order PHM 3045-D) or command module/cassette form (order PHM 3045-T) for about \$150.00.

## **BUSINESS**

Large and small businesses alike will find the programs in this section useful.

**TI-Count Business Series** is a series of six software packages implementing basic accounting functions for persons who conduct business at home. The TI-Count Series, developed for Texas Instruments by Pike Creek Computer Co., Inc., comprises six disk based packages written in TI Extended BASIC. The programs include General Ledger, Accounts Payable, Accounts Receivable, Payroll, Inventory, and Mail List. The first four packages are integrated. All packages sell for about \$100.00 each.

User's will need a TI-99/4A Console, an Extended BASIC cartridge, a peripheral expansion system, a disk memory drive, a disk controller card, an RS-232 card, and a printer. For optimum utilization, an additional disk memory drive and memory expansion card are recommended.

**Tax/Investment Record Keeping** is designed to let you keep all the data in a single filing system. You can keep track of assets, income, liabilities, etc., and record both your taxable and tax-exempt income. This system lets you determine your net worth, along with other financially-based records. A disk drive controller and disk drive are required to operate this package, which sells for approximately \$70.00 from TI (order PHM 3016).

**Securities Analysis** is for brokers and serious investors. It is an important financial

tool, providing bond analysis, stock analysis, option spreads, calculations of compound interest, along with annuities and cash flow. While intended for brokers and serious investors, it has possibilities for semi-serious investors who might like to play the stock market. Order PHM 3012 from TI; approximately \$55.00.

**Mailing List** stores, sorts, alphabetizes, and searches for mailing list information. The user enters names, addresses, telephone numbers, and other information relating to mailers. You may also search for a particular client's name and include a brief history on each. This program is available only on disk from TI, and sells for approximately \$70.00 (order PHD 5001).

**TI Writer** is an easy-to-understand word-processing package which provides many features of much more complex word processing systems with high price tags. This package lets the user insert and delete lines of text, automatically indent for paragraphs, and includes such features as overstriking and underlining, moving and copying text, and reformatting documents. The package is available as a plug-in module only (PHM 3111) and requires a disk drive and controller, 32K memory expansion, TI Impact Printer, and the RS-232 interface. It sells for about \$100.00.

**Microsoft Multiplan**, developed by Microsoft, is designed to aid in planning a business or budget. It may be used to plan a simple family budget, including personal investments, or capital budgeting for a small business. It requires a disk drive and controller, 32K memory, and a printer and RS-232 interface are optional. The package is available as a plug-in

module from TI (PHM 3113) and sells for \$100.00.

**Financial Management** sells for approximately \$40.00. It is used to project the amount of money needed to sustain a business and to predict how much capital costs will be. The disk also contains tools to calculate amortization, depreciation, cash flows, and annuities. This program, which is part of TI's Business Aids Library, requires Extended BASIC (order PHD 5022).

**Inventory Management**, another package in the series, Business Aids Library, allows inventory update and movement tracking. It is available only on disk, and requires the Personal Record Keeping or the Statistics Solid State Software Command module. Price is the approximately \$70.00 (order PHD 5024).

**Invoice Management** keeps accurate customer information, including addresses, applicable discounts, taxes, and a variety of other invoice information. The package includes seven programs to help maintain and update invoice records. A Personal Record keeping or statistics solid state software command module is also required and must be purchased separately. This Business Aids Library package sells for approximately \$70.00 (order PHD 5027).

**The Cash Management** Business Aids Library program provides a way to forecast the amount of cash available to your company, and estimates cash flow. It offers up to six forecasts with a maximum of twelve time periods per forecast. Beginning and ending balances are maintained in each period for cash, receivables, payables, investments, and inventory. The package offers 20 categories of data in

each forecast, 18 categories for incoming expenses, and 1 category for sales unit. There is also an automatically calculated gross-margin category. This program runs in Extended BASIC, and a printer is desirable. Price is approximately \$40.00 (order PHD 5029).

**Lease/Purchase Decisions**, the last in the Business Aids series, is available on disk (for \$69.95) and cassette (for \$59.95). It helps determine whether an investment will be economically beneficial. The program lets you rank investments according to desirability, allowing you to choose those which offer the best profit margin (order PHD 5038 for disk or PHC 6038 for cassette).

## HOME/PERSONAL

The programs in this section will prove useful for a variety of home and personal uses.

**Personal Record Keeping** requires an external data storage system. The storage system is not absolutely mandatory, but is recommended for maintaining (more or less) permanent records. This package lets you create a computer-based filing system, to update it, and pull information from it on command. Use possibilities include automotive maintenance records, medical records, a complete insurance filing system, etc. It can also be used to maintain a day-by-day calendar of duties to perform. Order PHM 3013 from TI, for approximately \$50.00.

**Personal Report Generator** is available as a plug-in module, designed to produce reports based on information and files created with the Personal Record Keeping or Statistics Command Module, (sold separately by TI). The package is divided into two sections. The

first is the Report Management section, which lets you perform operations on existing files, test formats, modify formats, print reports, and even save report formats to be used with other data files. The second part of the package is called Data File Management, which lets you add items to previously defined files, delete items, and combine files to create only one file. This package requires the TI Impact Printer and the RS-232 interface. It is priced at \$50.00.

**Personal Real Estate** sells for approximately \$70.00. This package can be a valuable tool for real estate agents and investors. It lets you evaluate personal real estate investments and closely follows the techniques used by the Realtors National Marketing Institute. It is also a valuable educational tool in real estate investing. Order PHM 3022 (module only) from TI.

**Personal Tax Plan** is ideal for the average homeowner, and provides valuable information regarding all aspects of tax planning and preparation. Designed by Aardvark Software, Inc., this program will perform many tax planning calculations in a fraction of the time it would take to do by hand. You are prompted to enter all income and expense information, and the program will analyze tax effects for you. Tax plans may be saved on disk. Results may be displayed on the monitor screen or printed in a letter format if desired, and if your system is equipped with the TI Impact Printer and RS-232 interface. Also, up to three disk drives may be used with this package. The package, which requires 32K and the P-Code module, is available on disk only from TI (PHD 5077) for approximately \$100.00.

**Home Financial Decisions** is a pro-

gram available in module form only. This program is an easy to understand guide to everyday financial questions that the homeowner may have. For example, it will provide information on the purchase of a home or car, as well as calculating personal savings data. This program lets you compare the difference between buying and leasing, renting versus owning, etc. It is available from Texas Instruments (PHM3006) for \$30.

**Household Budget Management** is an interesting program that helps the average homeowner establish budget guidelines for home financial management. Using this program, you can spot problem areas, keep records, and track income and expenses. This program lets you set up an entire budget system on a month-by-month basis, which makes it especially useful. It also provides a graphic analysis feature which prints charts and tables. This program comes on a plug-in module, but also requires a data storage system such as disk or cassette tape. Price is approximately \$40.00 from TI (PHM 3007).

**Personal Financial Aids** is available on disk for \$19.95, or on cassette for \$14.95. It lets the user to deal with many home financial problems, including loan amortization, depreciation computations, and mortgage analysis. Order PHD 5003 (disk) or PHC 6003 (cassette) from TI.

**Checkbook Manager** maintains records of checks, deposits, and running balances, and will even reconcile your bank statement. You can easily add entries and your balance is automatically updated. Checks can also be entered by account to see how much is spent on specific items. It also provides a sort and a sum-by-account feature. It is available only on

disk (PHD 5021) and sells for approximately \$20.00 from TI.

## **Self Improvement**

**Know Yourself** contains three separate lessons covering psychoanalysis, sex roles, alcohol, and general behavior patterns. This program package is for adults, and it is presented in a serious manner. Available from Creative Computing, the package is available on cassette only (order CS-6301).

**Weight Control and Nutrition** is designed to aid in preparing nutritional and diet-conscious meals, improving fitness through proper diet and caloric intake. Your own information is entered regarding weight, height, etc. The program determines recommended weight range and caloric requirements. Weekly menus can be created based upon the information and requirements of the user. The package is available from Texas Instruments in module form and sells for approximately \$60.00.

**Physical Fitness** is a program to aid you in planning and maintaining a physical fitness program. It is based upon input from the President's Council on Physical Fitness, and it is possible to design a specific program just for you. It may be used by persons of all ages and costs approximately \$30.00, from TI.

## **EDUCATION**

Many educational programs have been written for the TI-99/4A. The following programs are presented according to age group and curriculum.

### **Very Young Children**

**Early Learning Fun** is designed for

children between the ages of 3 and 6. The activities help a child learn shapes, numbers and letters, counting and sorting in a colorful and fun way. Also, the child will be learning some computer skills. Available from Texas Instruments, this educational program comes in module form and sells for approximately \$30.00 (order PHM 3002).

**Match 'Em I**, by Microcomputers Corporation, introduces children aged 3 to 6 with lessons in counting, recognition of numerals and knowledge of letters. The package can be purchased on disk (MCD0004) or cassette (MCT0004).

**Match 'Em II**, also from Microcomputers Corporation, is the next step in recognition of small letters and Roman numerals. Designed for ages 4-7, the package teaches these fundamentals in a unique and entertaining matching type of format. It is available on disk (order MCD0005) or cassette (MCT0005).

**Preschool IQ Builders: Letter Builders plus Same and Different**, developed by Program Design Incorporated, this package was designed for preschoolers and starts with very simple problems in letter matching. Two letters are used first, then the program takes the preschooler through all the letters in the alphabet in an entertaining manner. The package includes different types of matching drills which use colors, shapes, and letters. This educational tool is available only on cassette.

### **Math-Grades 1-9**

**Count 'em** is a simple educational tool designed for children of kindergarten and first grade levels. It produces a number of rabbits on the screen. The child is prompted to enter the correct number of rabbits on the screen.

Once the answer has been entered, the child is informed in a colorful and engaging manner whether his answer is correct. Available from Micro-Ed Inc. The package is offered on disk only (MA-2).

**Number Magic** provides drills and practice in the basics of mathematics for children ages 6 and up. It provides full color displays and sound and is based upon the popular Little Professor™ and Dataman™, also from Texas Instruments. Number Magic is available from Texas Instruments for \$20.00. Order PHM 3004, module form only.

**Number Readiness**, meant for children in Kindergarten through grade 2, helps students work basic math problems at 16 different problem levels. This program features drill and practice with color graphics and sound effects for correct responses. The program comes from TI.

Educational publisher Scott Foresman has two packages priced at \$39.95:

**Numeration I** is designed for children in grades 1 through 3. The cartridge introduces basic number concepts, including grouping, greater than, less than, and place value through the use of some tutorial examples.

**Numeration II**, for children in grades 4 through 6, teaches more about the basic math facts introduced in Numeration I, and adds number rounding, and expands on the basic concepts learned in the previous package. This program also includes some tutorial examples.

In **Alligator Mix**, hungry alligators lurk in a colorful swamp preying on apples that contain correct answers to addition and subtraction problems appearing on their stomachs. Children open the alligator's mouth when an apple with the correct answer appears and

close it rapidly when the answers and the problems do not match. Hits and misses are recorded at the bottom of the screen for problems using numbers 0 through 9. This and other Developmental Learning Materials (DLM) games use either a joystick or keyboard to control the action. Cost is \$39.95 on disk from TI.

In **Alien Addition**, children must answer addition questions correctly before their missile bases will fire on waves of alien invaders trying to attack the earth. Quick reflexes, in recognizing answers and firing on attackers, help children defend their missile bases while practicing addition with numbers 0 through 9. A DLM program, players can use either a joystick or the keyboard to answer questions and fire on aliens. Cost is \$39.95 on disk from TI.

In **Minus Mission**, a robot fights to defend his territory from the creeping slime, as children work to solve subtraction problems contained in the blobs of slime. Correct answers to problems with numbers 0 through 9 enable the robot to shoot and destroy the blobs. This is a DLM program from TI, costing \$39.95 on disk.

**Addition/Subtraction 1** was developed in conjunction with Scott, Foresman and Company. This package is designed to teach the user basic arithmetic skills. It reinforces these skills by producing drills. The program is designed for grade 1 arithmetic level, and a speech synthesizer is recommended for use with it. It is priced at approximately \$40.00 and is available in plug-in module form.

**Addition/Subtraction 2**, second in the series by Scott, Foresman and Company, is a tutorial package introducing more difficult

problems and techniques. It is designed for grade levels 1 and 2. Again, a speech synthesizer is recommended, and the package costs approximately \$40.00.

**Addition** is the first in a series of specific mathematics program packages from Microcomputer Corporation. This program consists of instructions which will teach elementary students aged 6 to 10 addition habits that will guide them in their further education. The package is available on disk (MCD0007) or cassette (MCT0007).

**Subtraction** is designed to teach the user standard approaches to the art of subtraction. The format of the package takes the student through the subtraction process step by step, showing how this particular math function is performed. Designed for students ages 6 to 10, the package can be purchased on disk (MCD0008) or cassette (MCT0008).

**Addition** from Milliken Publishing Company is available in command module form only. The package consists of 75 problem levels for students in grades 1 through eight. It provides drills and practice exercises using color graphics and sound effects in response to correct answer. Price is \$40.00 order PHM 3090.

In **Meteor Multiplication**, the inhabitants of a star station must defend themselves against meteors (that contain multiplication problems) by firing a cannon with the correct answers. Quick multiplication skills and rapid firing help children battle this threat from space while practicing multiplication problems with numbers 0 through 9. This is a DLM program available on disk for \$39.95.

In **Demolition Division**, four tanks ad-

vance simultaneously, each with a division problem. The player's only defense is the correct answer to each problem the tanks carry as he moves from one tank to the next to solve problems and fire before the foremost tank reaches its goal. Quick division skills and rapid fire save children from advancing tanks as they practice division with numbers 0 through 9. Another DLM program available on disk for \$39.95.

In **Dragon Mix**, the vigilant dragon stands guard over the city, but can defend the city only if children provide him with the correct answers to multiplication and division problems the invaders carry. Match the correct answer and aim the dragon's mouth at the enemy to fire. Miss and watch the enemy advance to bomb the city. Children practice multiplication and division problems with numbers 0 through 9 as they help the vigilant dragon. This DLM program is available on disk for \$39.95.

**Division** takes the student through the techniques used in division in a step-by-step manner. This is a highly useful teaching aid designed for children ages 8 through 12 by Microcomputer Corp. It is available on disk (MCD0010) or cassette (MCT0010).

**Division I**, developed in conjunction with Scott, Foresman and company, package combines animation, color, and graphics to teach the fundamentals of division to children in grades 3 through 5. All concepts are first taught, and the user is then allowed to practice what has just been learned. Although designed for grades 3 through 5, enrichment materials are provided for earlier grades as well as remedial materials for later grades. The package

comes with a Teacher's Guide which includes forms and worksheets for use in keeping records on individual students. The solid state speech synthesizer may be used with this package, although it is not necessary. It is available in command module form for \$40.00 (order PHM 3049).

**Multiplication 1** teaches the basics of multiplication in a fun and easy manner for grade levels 3 and 4. A speech synthesizer is recommended for use with this package, which was developed by Scott, Foresman and Company.

The **Multiplication** program package from Microcomputer Corporation is an entertaining and highly useful learning tool. It is designed for children between the ages of 8 and 12 and teaches the techniques of multiplication in a unique and challenging manner. It is offered on disk (MCD0009) and cassette (MCT0009).

**Bar Graph** is designed for use as an exercise in designing and using bar graphs for the elementary grade student. The information is presented in a colorful and entertaining manner. A number of different graphs are presented, and the student is drilled in the information presented, as well as provided with a means of practicing what has been taught. This package is available on disk only (MA-12).

**Laws of Arithmetic** helps children in grades 4 through 8 learn the laws of arithmetic, such as the Distributive Law and the Commutative Law, as they work problems at 19 different drill and practice levels with color graphics and reward sound effects. This is a DLM program available from TI on cassette.

**Equations** present drill and practice

problems for children grades 6 through 8 using 26 problem levels with color graphics and sound effects. This DLM program is available from TI on cassette.

**Measurement Formulas**, developed for children in grades 6 through 8, has students work through drill and practice problems in 25 different problem levels. Color graphics and sound effects reward correct responses. This is a DLM program available on cassette.

Texas Instruments also announced three Math Games programs to be available in the second quarter of 1983. The games, which include color graphics and are designed with progressive levels of difficulty for children in grades 1 through 9, and have a suggested retail price of \$39.95.

**Computer Math Games I** has progressive skill levels for children in grades 1 through 9. Five games are used to help children learn math skills. One game features basic math facts drill and practice and problems with percentages. Two games feature plotting numbers and ordered pairs of numbers. Another game features addition, subtraction, multiplication, and division; the last game features practice with place values

**Computer Math Games III** is for children in grades 1 through 9. Students learn by using seven card-type games with progressive skill levels to practice basic math facts, find squares of sums, reduce fractions, and practice counting and the concept of betweenness on a number line.

**Computer Math Games IV:** five games in this module for children in grades 1 through 9 feature progressive skill levels to provide drill and practice with basic math facts, whole

numbers, and compact and expanded forms of numbers. This module also includes a strategy game called Nim 25, where players take turns answering questions to remove 1 to 3 pieces from the total of 25 on the board. The object of the game is to be the one who removes the last piece. Timed response periods in this module challenge children to answer questions quickly.

In **Decimals**, 75 problem levels are included in this package to help the student learn the fundamentals of using decimals. Designed for use by students in grades 1 through 8 it also includes color graphics and sound effects. It is priced at \$40.00.

Milliken Publishing Company has developed additional packages designed in the same format as Decimals and Addition. All are for students in grades 1 through 8, and all feature 75 different levels. The packages are each priced at \$40.00. Titles are Fractions, Integers, Multiplication, Percents, and Subtraction.

**Speak & Math Program** introduces the child to addition, subtraction, multiplication, division, number relationships and problem solving. To use this package, your computer must be equipped with a speech synthesizer and the Terminal Emulator II command module, providing the child with a reproduction of the human voice. Many different activities are included in this package, which will introduce mathematics in a fun and challenging way. The package is available on disk for \$29.95 (order PHD 5031) or on cassette for \$24.95 (PHT 6031) from Texas Instruments.

**Computer Math Games II and VI**, are available as command modules, were de-



veloped by Addison Wesley Publishing Company, and are designed for students in grades 1 through 9. Different levels are provided in a progressive fashion to enable the student to go at his or her own speed in learning math skills. Each package contains educational material presented in a challenging and entertaining manner, complete with color graphics, sound, and music. Some games are designed for use by two persons, adding to the entertaining and educational value. Order PHM 3083 for Math Games II and PHM 3088 for Math Games VI. Both are priced at \$40.00.

### **Reading and Writing Skills**

**Reading Flight** is designed for students in sixth grade. Three different stories are presented with accompanying drills. The student is drilled in classifying, summarizing, and outlining information. The package was developed by Scott, Foresman and Company, and comes as a plug-in module for about \$55.00 (order PHM 3082).

**Reading Roundup**, also developed by Scott, Foresman and Company in conjunction with Texas Instruments, is meant for use by students in grade four. The module is divided into three main sections designed to aid in the development of reading skills associated with figures of speech, word meanings, and idioms. Once the material has been absorbed, the student is taken through drills. Available as a plug-in module, this educational package sells for about \$55.00 (order PHM 3047).

**Reading Fun** is designed for students in the fourth grade. This package combines reading skills tutoring with information about the world. The package contains a four-part program which includes three stories and

drills. The reading skills portion includes Problems in Stories, Why Things Happen, and How People Feel. The Solid State Speech Synthesizer is optional with this package, which was developed by Scott, Foresman and Company. It is available in command module form and sells for \$55.00 (order PHM 3043).

**Reading On** was developed by Scott, Foresman and Company and is designed to teach third grade students how to read and understand maps, graphs, and schedules. Again, this is a four-part learning program which includes stories and drills. The first three parts of the package take the student through stories and drills. The final part incorporates all the skills learned in the previous three parts into a story, ideal for children at this level. The package is available on a command module for \$55.00 (order PHM 3046).

**Reading Rally** was developed by Scott, Foresman and Company for children in the fifth grade. Basic reading skills are taught through a presentation of stories, which include Fact and Opinion, Author's Purpose, and Bias and Connotation of Words. Drills are provided as a review of the material, and the final part of the package combines everything learned into a roundup story. Available as a command module, this package sells for \$55.00 (order PHM 3048).

**Early Reading** requires a speech synthesizer; it combines color graphics and computer speech to introduce and reinforce reading skills to beginning readers. Early Reading was developed by the educational staff of Scott, Foresman and Company. It sells for approximately \$55.00 and is available in module form only.

**Grade 2 Spelling** is a comprehensive teaching program for elementary grade students. It consists of 36 lessons in which a specific rule or pattern regarding the fundamentals of spelling is introduced. The student chooses which lesson he would like to work on, and the information is displayed on the screen. Various types of problems are presented to enable the user to become familiar with that particular rule, such as a sentence presented with a word left out, etc. The package is available from Texas Instruments on disk only (SP-2).

**Grade 3 Spelling**, created by Micro-Ed, Inc., is an advanced version of the Grade 2 Spelling package. The package is designed to instruct the student in basic spelling rules and patterns. It is available on disk only (SP-3). Micro-Ed, Inc. also offers similar packages for grade 4 (SP-4), grade 5 (SP-5), and grade 6 (SP-6), all available on disk only.

**Speak & Spell program** is similar to the Texas Instrument's teaching aid sold as a separate unit. Children are taught to pronounce words and spell them correctly via a number of activities. A word is first pronounced and the child is asked to enter the correct spelling. Another part of the package provides prompts which instruct the child to pronounce a word and then spell it. Many exercises are provided which introduce the child to the world of pronunciation and spelling. Available from Texas Instruments on a module, this package sells for approximately \$30.00 (order PHD 5030).

**Spell Writer** requires the Terminal Emulator II package and speech synthesizer from Texas Instruments. It uses text-to-speech technology to introduce the user to

spelling in a number of entertaining and challenging ways. One program is designed so the user may create his own spelling lessons. Also included is a word game, as well as a file transfer program which lets the user produce extra copies of word lists. This is a highly useful and fun package. It is available on disk for \$30.00 (order PHD 5042) or cassette for \$25.00 (order PHT 6042).

**Scholastic Spelling:** the packages in this series were developed by Scholastic, Inc. Each package is designed for a specific grade level and contains 36 lessons and spelling games. Words are presented in each lesson. Some lessons are a review of the previous five or six lessons; some of the lessons are simple presentations of spelling words and drills. Others develop proficiency in different rules regarding spelling, while others provide lessons in spelling, in a game format. All are available in command module form and sell for \$60.00. Order PHM 3059—Level 3, PHM 3060—Level 4, PHM 3061—Level 5, and PHM 3062—Level 6. The solid state speech synthesizer from Texas Instruments is required for each of these packages.

**TI-Jotto** was developed by the Microcomputers Corporation, and consists of a game-formatted package that teaches vocabulary. It is also designed to develop the user's ability to analyze patterns and introduce the user to logical thinking. Designed for students age 8 and older, there are different difficulty levels of play to enable the user to progress at his own speed. The package is available on disk (order MCD0002) or cassette (MCT0002).

**Racing Letters** was designed by Microcomputers Corporation for children ages 5

to 7. This is a fast-moving package which teaches the user the alphabet and numbers in an appropriate format. It is available on disk (MCD0006) or cassette (MCT0006).

**Beginning Grammar** introduces the basic parts of speech for grades 2 through 5 in a colorful and engaging manner. Sentence construction is covered, and the activities are presented in a format which will keep a child's attention for hours. Available as a plug-in module, the Beginning Grammar program is from Texas Instruments and costs approximately \$30.00.

**Identifying Complete Sentences**, from Micro-Ed, Inc., has been designed for the elementary grade student, and introduces the user to sentence construction. The student is first presented with a group of words on the display, and is then asked whether the grouping constitutes a complete sentence. The sequences are presented randomly so the student can continue to use the program over and over without becoming familiar with the information. The package is available on disk only (RE-9).

**Code Breakers** was developed by Program Design Incorporated and consists of three educational programs designed to test the ability of the user to decode sentences that contain scrambled words and letters. It has been created for students in the fourth grade and higher grades as well. It can be purchased on cassette.

**The Verb** was designed by Micro-Ed, Inc. for elementary grade students. It presents information dealing with action verbs, linking verbs, and verb phrases, as well as a host of other lessons dealing with the use of verbs.

Presented first in a teaching format, the student is provided with simple tests on the subject matter to determine his proficiency in the lessons. The package is available on disk only (order GR-3).

**Antonym Machine**, designed for the elementary school student by Micro-Ed, Inc., is a teaching aid to develop knowledge of antonyms. The computer displays two words with opposite meanings which are randomly selected. The student is then prompted to enter the antonym. There are 50 pairs of antonyms in this package, providing many hours of educational lessons in this area. The program is available on disk only (VO-3).

**Homonym Machine** is similar to the antonym package. In this package, the computer selects a pair of words with different meanings that sound the same. The user is prompted to enter the word which has not been displayed. Fifty pairs of words are provided. The program is designed for elementary grade students and is available on disk only (VO-4).

### **Music Programs**

**Music Maker** is a music composition program that allows the user to create computer music. This is accomplished in a simple format in which the user arranges notes on an electronic music staff. You can create your own compositions or program familiar songs as well. A data storage system is recommended for this package, which sells for \$40.00 and comes in plug-in module form. It is available from Texas Instruments.

**Music Skills Trainer** is designed for persons 10 years and older. It consists of four musical drills that test music ability and also

improve previously-learned music skills. The drills include interval recognition, chord recognition, pitch guess, and phrase recall. It is a beginning music program for students with little or no experience in music skills. Cost for the disk version is \$30.00, while the cassette version sells for \$25.00.

**Computer Music Box**, designed for students 10 and older, is a music composition package that allows for programming three-part music. It also allows for playing, editing, and saving any compositions entered into the computer via the keyboard. The automatic chord feature included in this program package makes it possible to compose computer-generated tunes as well. The package is available on disk for \$20.00 and on cassette for approximately \$15.00.

**Higher, Same, Lower**, from Micro-Ed, Inc., is a music concept learning aid. Sound is used to introduce the student to music. Two notes are played, then the student is prompted to enter whether the second note is higher, lower, or the same as the first. This is a very basic introduction to music and provides the user with logic understanding as well as the very basics of music and sounds. The package is offered on disk only (MU-2).

### **General**

**Making an Outline:** yet another educational tool from Micro-Ed, Inc., this package teaches the elementary student in a multiple choice fashion how to make an outline. The student is first instructed to read an article. The student is then presented with a listing of topics included in the article and must arrange them in sequential order as they appear in the

article. This package is offered only on disk (RS-3).

**Direction and Distance** is arranged in a game format and is designed for students in the primary grades. It introduces the student to directions (north, south, east, west, etc.) and to the method of determining distance between two objects displayed on the monitor screen. This teaches the student a bit about logical decision-making, as well as introducing the basic directions and means of determining distance. The package is available on disk only (OT-2).

**Clock**, by Micro-Ed, Inc., is a program that teaches primary grade children how to tell time. There are a number of lessons included in this package, each consisting of random selections of clock faces. The child is prompted to enter the proper time and is provided with a display informing him or her whether the answer is correct. Clock is available on disk only (OT-5).

### **For Older Students**

**Key to Spanish** is a new software package for people who want to learn conversational Spanish, developed by Texas Instruments under license with Westinghouse Learning Corporation. This package consists of a three-ring binder containing four Solid State Software™ cartridge, four audio cassettes, and an instruction manual. The software is designed to teach vacation travelers or businessmen the Central and South American dialects of Spanish.

As shown in Fig. 10-1, an introductory lesson and six subsequent lessons and word games are contained in the cartridges. The

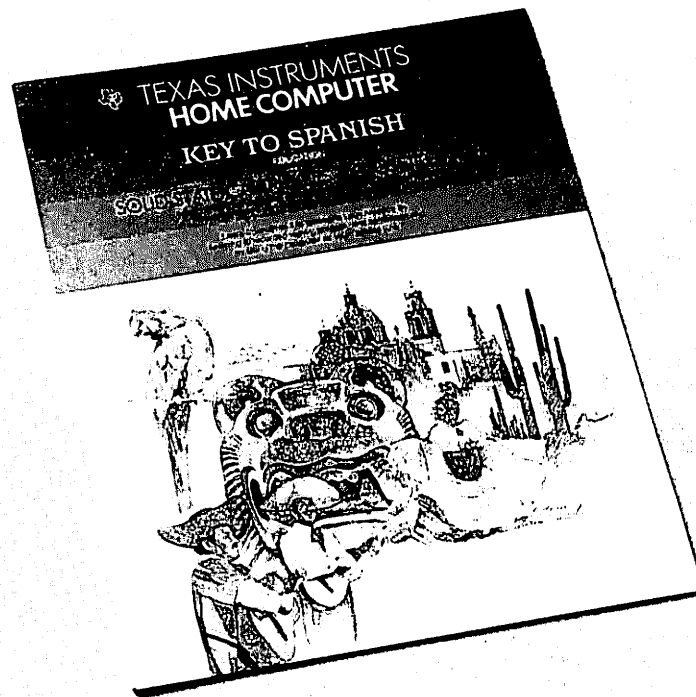


Fig. 10-1. The Key to Spanish Software Package (Courtesy Texas Instruments Inc.).

audio cassettes, which are controlled by the cartridges, help beginning speakers learn to pronounce Spanish. The system concentrates on useful phrases and words most common in day-to-day Spanish usage. Because the system is designed to let students learn at their own pace, they can disconnect the cassette player from the computer and operate it manually to control the pace.

Users will need a TI-99/4A computer and a cassette player, such as the Texas Instruments Program Recorder. Suggested price for this software album is \$149.95.

**Touch Typing Tutor** is designed for the beginning typist and uses the TI-99/4A computer keyboard. It is divided into a variety of lessons, and the user is taught to improve speed and accuracy. Each lesson covers a specific area, such as letters, numbers, and symbols, diagnostic timing system, keystroke analysis, as well as a game. It is available as a command module from TI and sells for about \$40.00, (order PHM 3064).

**IQ Builder Series: Analogies Lesson** was developed by Program Design Incorporated. This package consists of six separate

drills designed to prepare high school students for SAT exams and similar types of tests. The user is required to determine the analogy type for the examples provided. Each lesson is more difficult than the previous, taking the user through this type of learning in a logical progressive manner. The package is on cassette only.

**The Vocabulary Series** from Micro-Ed, Inc. is designed for upper elementary and high school students. It presents 72 lessons in word recognition, using commonly used words found in newspapers and news magazines. This comprehensive lesson sequence in vocabulary is designed to introduce the user to the type of language found in these periodicals. Each lesson presents a definition and a sample sentence. The user is then instructed to select the best word to insert in the sentence. The package is available on disk only (VO-2).

### **Logic Games and Simulations**

**Hat in The Ring Presidential Election Game**, created by Micro-Ed, Inc., can be used by children of all ages. This is a two-player program which first introduces the user(s) to some of the fundamentals of political considerations normally involved in an election campaign. One student plays a Republican, and the other a Democrat. The package is an introduction to our political system for students in the middle and high school grades. It is available on disk only (OT-9).

**Wall Street—A Market Simulation** provides the user with a computerized simulation of the stock market, in which he is required to make as much money as possible. You are given ten years in which to make a

fortune. This is an educational game, which can be used in BASIC (C1060) or Extended BASIC (C106-X). It is available on cassette only.

**Mind Masters** is a strategy and logic-teaching simulation game in which the computer creates problems and the user must solve them. The game is designed for multi-player use and different skill levels may be used by each player. Logic problems are created by the computer as well. The principles of deductive logic are taught in a fun and challenging manner which tests the users skill, ability, and patience. The package is designed for persons 10 and older and was designed by Image Producers. It is available on cassette only (#9405).

**Wall Street Challenge**, developed by Image Producers, is a computer simulation of Wall Street. It simulates the stock exchange in a challenging and educational manner, in which the user is allowed to make investments in different types of stocks. Charts are provided, as well as a Dow Jones report to keep you up to date on the current trends. The package is designed for users aged 13 and older and is available on cassette only (order #9402).

Another program from Image Producers is **Wildcatting** available on cassette only (#9403). In this game, the computer sets up hidden oil deposits, and the user must try to locate them. Geological survey data is provided to help determine whether oil is in a given location. Drilling costs are given. The oil deposits are in different locations each time the game is played. Designed for users 10 and older, this is an educational package which will teach the user logical decision-making based upon provided facts.

## **Programs for Educators**

**The advoCAIte Course Authoring System**, was designed by EduCAIte Incorporated, the developer of Computer Aided Instruction. They also offer a package in course curriculum and customized course development. The package is designed to help an educator with little or no data processing experience to develop CAI courses. The package is very simple to use, providing the educator with a means of entering simple yes and no answers, very simple numerical inputs, and a means of entering screens of text. This package and the others from this company can be purchased on disk only.

**Basketball Statistician** is designed for use by a basketball coach or record-keeper. It includes provision for entering such statistics as shots taken, shots made, rebounds, as well as a host of other major basketball statistics. This package requires Extended BASIC and is available from Texas Instruments on disk for approximately \$25.

**TI LOGO** is based on a philosophy of education developed by Seymour Papert and the staff of the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. It is a computer language that develops computer awareness in children and teaches a child math, logic and communications. Using this program, a student learns to teach the computer. The memory expansion unit from Texas Instruments is required for use with this package, which sells for approximately \$124.95. It comes in module form only.

**TI LOGO II** is designed specifically to aid in teaching children computer skills, including math, logic, and general communica-

tions skills. TI LOGO creates environments to teach mathematics to students in an easy-to-understand manner. This is the second version of TI LOGO, which was developed by Seymour Papert and the staff of the Artificial Intelligence Laboratory. TI LOGO also features printer capability, music, and enlarged sprites. Requirements for this package include 32K memory, TI Impact Printer (optional) and the RS-232 interface (optional). The package is available as a plug-in module for about \$130.00.

**Plato Courseware** was developed for use in schools primarily via terminals from a mainframe computer. It is now available on disk for the TI-99/4A. PLATO is a trademark of Control Data Corporation.

Shown in Fig. 10-2, the courseware packages cover reading, mathematics, and grammar in the basic skills for grades 3 through 8 and mathematics, writing, science, social studies and reading in the High School skills for grades 9 through 12. Reading and mathematics in basic skills and High School skills will be available in the first quarter of 1983, with the remaining four topics available in the second quarter. The 108 PLATO courseware packages contain 64 packages in basic skills and 44 packages in High School skills, and contains a total of more than 450 programs.

The initial PLATO package includes an interpreter solid-state cartridge and disks containing a basic skills survey to let parents or teachers help select courseware for individual needs. For those not familiar with the operation of the computer, the program is designed to teach beginners to use the keyboard on the TI-99/4A. A parent questionnaire is also included with the first package to help parents or



Fig. 10.2. The PLATO Courseware Package (Courtesy Texas Instruments Inc.).

teachers direct students to appropriate courseware for their skill level.

The first package, containing the interpreter, survey, and questionnaire, is priced at \$49.95. PLATO courseware for the TI-99/4A also sells for \$49.95. To take advantage of the PLATO software with the TI-99/4A, users will need a TI peripheral expansion system, a memory expansion card, a disk memory drive, and a disk controller card.

### School Management Applications

Scott, Foresman and Company have developed a series of programs for the Texas Instruments computers. Each package comes with five disks, a detailed reference manual, and a command module.

**Student Data Recorder** is designed to keep up-to-date records on students, including the name, address, birthdate, emergency phone numbers, student identification number, homeroom, locker combination, bus number, etc. Provision is made for storage, updating, and various means of summarizing student data in a manner which is convenient and efficient to use.

**Attendance Recorder** makes it possible for school personnel to keep track of student attendance. This information is entered on a daily basis, and the information can be summarized in a number of formats, including weekly, quarterly, semesterly, and annually. Alphabetical lists can be output which will provide information as to students absent, students tardy, etc.



**Class Data Recorder** provides a means to keep an accurate grade book on the computer. Using this program, the teacher can enter individual student's grades, calculate averages by grade, class rank, etc.

**Test Scorer** provides a means of scoring and analyzing test grades for individual students. It can also perform item analysis, calculate frequency distribution, determine grades based on various criteria for individual tests, convert numerical scores into letter scores, and produce various types of summaries based upon entered information.

**School Mailer** provides a multitude of uses for school administrators and teachers. It can be used to serve as a data base for parent telephone numbers, addresses, and other emergency information. Alternately, it can print many different types of mailing lists, as well as self-adhesive labels that can be selected by grade, teacher, building, or even destination zip code. This is a multi-function program that can increase overall efficiency many times.

**Mark Reporter** can produce progress reports, labels for cumulative records, and incomplete lesson lists. Also, it will perform a myriad of calculations on the information which is entered, such as class ranks for one or more classes, grade distributions, grade point averages, etc.

**Scheduling Assistant** is designed to help school administration officials in the proper and accurate scheduling of students, thus providing a means of scheduling each student in a manner that does not produce conflicts in classes and conforms to the students' requests. It will provide the administrators with a variety of listings, such as course-conflict lists,

student schedules for each individual student, class lists, etc.

**Payroll Assistant** does not actually produce payroll checks, but it will provide many different types of summary information that can be of considerable value to school administrators. Such reports as FICA and tax withholding, and other payroll information are provided in an easy-to-understand format.

**Personnel Data Recorder** enables administrators to keep an accurate and current file on each employee. Information such as name, address, phone number, birth date, salary, degrees and certificates, seniority, etc., can be stored and retrieved upon request in a number of different formats and by category.

**Activity Accountant** is designed to help administrators maintain an activity listing, keep track of petty cash, and provide current information on food service and other types of special accounts.

**Accounting Assistant** is a budgeting type of program that is quite useful in the overall accounting process of an educational institution. It provides many types of reports upon request. It can record purchase orders, keep track of vendors, and provide a check register listing.

**Salary Planner** is a type of analysis program and will provide administrators with data concerning proposed salary schedules and the overall effect such budgeted information will have on the budget as a whole. A number of reports can be obtained, such as cost calculations, changes by percentage, salary increments, etc.

**Property Manager** is designed to keep inventory records of equipment commonly used in school systems. It will maintain an

accurate record of the name of the equipment, the manufacturer, purchase date, serial number, stored or use location, condition, etc. This is a convenient way to keep track of all equipment.

**Data Analyzer** is an analysis program which can be used to evaluate test scores in the school system by grade, classroom, school level, etc. Also, the package is designed to analyze surveys, as well as school district data.

**Course Manager** enables the teacher to maintain records on each individual student's progress in individual programs. A great deal of information can be stored and output in many different formats. Objectives can be input, and test items can be summarized. The package is designed for grades 3 through 8.

## **GAMES**

The final section of this chapter presents some game programs.

### **Card Games**

**Bridge Bidding I** is designed to teach the user how to bid a bridge game. It is designed for the intermediate and advanced bridge player, and pits you as south. You select the bid, while the computer provides all other bids. You are given three turns to enter a bid, after which the computer provides the best or most recommended bid and an explanation. Available from Texas Instruments on disk (\$30) and cassette (\$25), the package is an excellent bridge-building tutor. Order PHD 5026 (disk) or PHT 6026 (cassette).

**Bridge Bidding II** is designed for the more experience bridge player and provides instruction in slam bidding as well as some of

the bridge bidding conventions. This package was developed by Robert Hammon and Robert Wolff of the Dallas Aces and it reflects their experience in the art of bridge bidding. Available on disk for \$29.95 (order PHD 5039) or cassette for \$24.95 (order PHT 6039), this package introduces ace asking, sources of tricks, cue bidding, and many other techniques.

**Bridge Bidding III** is the third in the Texas Instruments bridge series and provides detail on competitive bidding. Details in this package include discipline, partner trust, judgment decisions, and other aspects of the art of bridge bidding. The package is available from Texas Instruments on disk for approximately \$30.00 (order PHD 5041) or cassette for about \$25.00 (order PHT 6041).

**Draw Poker** pits one person against the computer. The computer is the dealer. You have a number of options once the cards are dealt, such as raise, call, fold, discard, etc. You are able to see all your cards, while the computer's cards are all face down. You are given a bankroll at the beginning which is equal to the computer's bankroll. The game is over when you or the computer run out of money. This game requires Extended BASIC and is available on cassette (PHT 6037) for \$25.00 or disk (PHD 5037) for approximately \$30.00.

**Challenge Poker**, designed for use by up to four players, can be played against the computer as well. Points are scored by creating the best poker hand. The winner is the player who is able to amass 100 points first. Designed by Pewterware, the game is available on cassette only (CPW67070).

**Casino Pack** was developed by Ehn-

inger Associates, Inc. In this game, the computer is the house and you are betting. The package includes a slot machine and blackjack tables. Based on the popular Las Vegas game, this package is available on disk only (D1010).

**Blackjack** was developed by Color Software, and up to four may play at the same time. All the cards in a standard deck are displayed face up in a colorful and easy-to-recognize manner. The game is designed for users 12 and older and is available on cassette only.

Developed by Milton Bradley Company, this package contains **Blackjack and Poker**, two betting games with which most people are familiar. You are given a certain amount of money at the beginning of each game. When your money runs out, the game is over. Both games can be played by users aged 10 and older, and the package is available as a plug-in module for \$25.00 (order PHM 3033).

### Sports Games

**All\*Star Baseball**, designed by Ehninger Associates, Inc., is a two-player game in which each player has control over pitching, field, base-running, and a number of other tactics common to the game of baseball. Active participation is the key in this exciting and challenging baseball simulation. The game may be played by persons of all ages. It is available on cassette only in either BASIC (C1020) or Extended BASIC (C1020X).

**Extended Baseball** was developed by Extended Software Company and requires Extended BASIC. The user controls the pitcher and batter. The user controls a number of skills such as balls, strikes, inning changes, scoring,

hitting, fielding, etc. You are also given such statistics as batting averages. The game is available on disk or cassette.

**All\*Star Bowling** is a simulated bowling game which may be played by as many as eight players at the same time. You may throw the ball and knock down the pins at various speeds. Before rolling the ball you must position it in the lane. This game is designed for use with Extended BASIC and was developed by Ehninger Associates, Inc. It is available on cassette only (C1110X).

**Decathlon**, also developed by Pewterware, is based on the Olympic Decathlon event. The user is required to compete in all ten events, and timing is very important. You have one second to complete the first event, with the time increasing in one-second increments with each event, so the final event must be completed in ten seconds. This is an exciting and challenging game designed for users aged 10 and older. The game is available on cassette only (CPW67030).

**Football** has been designed by TI for users 8 years old and up. It provides a simulation of football based on actual pro football statistics. The player can select offensive plays, defensive plays, and acts as the quarterback on offense.

**Indoor Soccer** is a five-man soccer game. The user controls the players and can make passes, shots, interceptions, saves, tackles, and a number of other common tactics. This is a fast-paced game. One of the unique features of this game is that you can view an instant replay of each score. The game is designed for users 8 and older and is available on module only for \$30.00 (order PHM 3024) from TI.

## Games of Skill

In **The Attack**, the user is the commander of a spaceship and must destroy enemies. The user must maneuver his ship to avoid contact with the alien ships and fire missiles at the same time. Developed by Milton Bradley Company, this is an exciting and entertaining game which can be played by users of all ages. It is available in plug-in module form only and is priced at \$40.00 (order PHM 3031).

**Blasto** can be played by one or two players. It is a tank game in which the user must destroy a mine field while avoiding opponent fire at the same time. This is a fast-paced game which is timed, so you must quickly destroy as many mines as possible. A number of options are provided, and there are dangers involved. If you hit a mine at close range, for example, you must start over. This game is designed for users 10 or older and is available as a plug-in module for \$25.00 (order PHM 3032).

**Bluegrass Sweepstakes** developed by Pewterware, this game displays a field of horses, and the user is shown the complete race. It is available on cassette only (CPW67020).

In **Cars and Carcasses**, developed by Not-Polyoptics, you are the driver of a car on a randomly generated board. The object of the game is to save an imaginary city from monsters such as Frankensteins, Draculas, and weird space creatures, by running them over as you move around the board. This exciting and challenging adventure game is available on cassette.

**Car Wars** is a speed-racing game of skill in which the player is pitted against the computer. The object is to maneuver your car

around the track while avoiding obstacles. This game has several difficulty levels and may be used with the TI wired remote controllers if desired. It is available as a plug-in module and sells for \$40.00.

**Chutes & Sharks**, also developed by Ehninger Associates, Inc., is designed for single-player use. In this game you are in control of a boat waiting for paratroopers who are dropped from a helicopter positioned overhead. To further complicate the situation, there are sharks in the waters waiting to destroy the paratroopers if you do not position the boat to receive the falling paratroopers poorly. This program requires Extended BASIC and the memory expansion may also be used, although it is not required. The game is available on cassette only (C1120XM).

**Galactic Gunfight**, developed by Inter-soft, puts you in control of a spaceship which is invaded by aliens. The purpose of the game is to defend a colony against these invaders, which appear on the screen in squadrons of five at increasing speed levels. The game is designed for users 7 and older and is available on cassette (CIS64540).

**Hustle** is a fast-paced game designed for one or two players. The user is in control of a snake-like object with which he must try to hit targets while avoiding his opponent, the edge of the screen, and his own object, at the same time. This is an excellent tool for learning quick reflexes, as well as eye and hand coordination. It can be used by users 10 and older and is available in module form only for \$25.00 (order PHM 3034).

**Ships!** is another adventure game from Not-Polyoptics. This game can be played by more than one player. You are in command of a

ship on the high seas. You must steer your ship in a variety of changing conditions, such as wind changes, which make sailing more difficult. The graphics in this game are good. The game is available on cassette.

**Speedway 100** is a well-illustrated graphic game in which the player is one of six cars on a speedway. The user first selects the number of laps he wishes to make and then maneuvers his car around all other cars, which appear in different lanes at varying speeds. Other obstacles are provided to increase driving proficiency. The game is available on cassette only, and was developed by Intersoft (CIS 64540).

**Tickworld** may be unpleasant for some; you are pitted against eight gigantic, hungry ticks. This game creates a different game board each time it is played, and your job is to capture the ticks and imprison them before they get you. There are three skill levels. This game was developed by Not-Polyoptics and is available on cassette.

**TI Invaders** is a one-player game in which you are under attack from strange space creatures. You must be quick to use your missiles before these multi-color creatures get you. The wired remote controllers from TI are optional, and the game comes in plug-in module form for approximately \$40.00.

**TI-Trek** is an exciting and challenging game using the speech capabilities of the TI-99/4A. You are responsible for the safety of a galaxy and have the ability to fire phasors, torpedoes or multiple torpedoes in an effort to destroy the enemy before he endangers your galaxy. A warp control is provided for additional encounters. The package is available on disk only for \$15.00 (order PHD 5002).

**Tournament Brick Bat** was designed by Image Producers and pits the player against the computer or a human opponent in a fast-action skill game. The computer keeps a record of the score and the game increases in difficulty as skills improve. Designed for users 10 and older, the game is available on cassette only (order #9041).

**Video Games I** consists of three games: Pot-Shot, Pinball, and Doodle. Each is designed for use by persons of all ages. Pot-Shot is an aim-practicing game, Pinball is designed in the format of the pinball games found in arcades, and Doodle is a game in which the user tries to trap his opponent. Each game is sure to provide many hours of entertainment. Available on module, it is priced at \$30.00 (order PHM 3018).

In **ZeroZap**, the user is provided with a computerized pinball game that includes electric lights and fascinating sound effects. The user may also create an individualized playing field, providing an educational as well as entertaining game. This program was developed by Milton Bradley Company and may be used by children as young as 6. Available as a plug-in module, it sells for \$20.00 (order PHM 3036).

### **Games of Strategy and Logic**

**Advance** is a board game developed by Not-Polyoptics. In this strategy game, two or three players compete in moving up the board. You can purchase squares on the board, and the whole game provides a number of random selections. Each square has a point value, or it may take points away from the player after purchase. Blocking other players stops them from reaching the end. This game is available on cassette only.

**A-Maze-Ing** is an exciting and challenging game of mazes. The user is able to select from options providing many mazes, from the very simple to the very complex. The chances of seeing the same maze twice are rare—this game provides 5,200 variations. The package is available on module and sells for \$25.00; order PHM 3030.

Strategy is used to the fullest in **Barrier**. Two players are required; the first player tries to draw a continuous line on the screen from left to right, while the second does the same from top to bottom. The object is to reach your goal in the shortest time possible, while preventing your opponent from reaching his goal. The game is designed for persons of all ages and is available on cassette only.

**Brain Games** was developed by Creative Computing. It consists of five challenging games designed to test your brain power. The games are Dueling Digits, Parrot, Tunnel Vision, European Maps, and U.S. Maps. All games are designed for users of all ages, and the package is available on cassette (CS-6002).

**Challenge I** displays ten frogs on the screen. The user is prompted to use logic to take the frogs through leaps which will reverse the color pattern of the frogs. This strategy game is designed for use by two players and was developed by Ehninger Associates, Inc. It is available on cassette only (C1030).

**Challenge II** contains two programs. The first is NIM, a game two players may play. Objects are arranged in rows and the players begin removing the objects, one at a time, until the winner is determined. The winner is the one who removes the final object. The second game is the popular game of Tic-Tac-Toe. De-

signed by Ehninger Associates, Inc., Challenge II is available on cassette only (C1040).

**Connect Four** was developed by Milton Bradley Company and is the computerized version of the popular game. In this strategy game the user has to place four markers in a row (down, across, or diagonal) to win. The game may be played by persons 10 and older and is priced at \$20.00. It is available as a plug-in module only; order PHM 3038.

**Corner Bound** is a combination skill and strategy game in which a snake line is presented on the screen. You are in control of the line, and you must maneuver it to hit targets placed in the corners of the screen. This is an excellent teaching aid, using both eye and hand coordination, and three skill levels are provided to increase your proficiency. The game was developed by Microcomputers Corporation and can be played by persons aged 8 and older. It is available on disk (MCD0001) or cassette (MCT0001).

**Crosses** is a computer simulation of a combination of Go and Othello, two popular board games. This game is presented in board form, and two players are required to strategically compete, placing markers on the board and trying to capture the opponent's marker while making a cross on the board. This game was developed by Not-Polyoptics, and is available on cassette only.

**The Cube** was developed by Linear Aesthetic Systems, and provides a graphic simulation of Rubik's Cube on the display screen. High resolution graphics make this game exciting visually and challenging as well. You have complete control over the movements of the cube and can command it to display any of

six sides, as well as rotate it clockwise, counterclockwise, spin it to see another side, etc. This skill-challenger is available on cassette only.

**Doctor Nuttier** was developed by Ehninger Associates, Inc. It is more an entertainment program than game. Dr. Nuttier is the computer, and the player is prompted to enter a question. Using psychoanalysis techniques developed by Carl Rogers, the doctor provides advice based upon the input. The program is available on disk only (D1050).

**Hangman** was developed by Hall Software. Two players are required; one selects a word, and the other must guess the word before his man is hung. You have only seven guesses in this game. It is available on cassette only.

**Hangman** is the computerized version of the popular game. The computer selects a mystery word, or the user can enter his own word. An opponent is asked to guess the letters in the mystery word. Each incorrect guess causes another portion of the hanged man's body to be drawn on the gallows. The object is to guess the word before the man is hanged. Designed by Milton Bradley Company, this package sells for \$20.00 and can be played by users 6 years and older. It is available as a plug-in module; order PHM 3037.

**Extended Hangman** is yet another computerized version of this popular game. This package was developed by Extended Software Company and requires Extended BASIC for operation. It uses color, graphics, and speech, making this extended form of hanging unique and entertaining. Over 500 words are included in the computer's word

vocabulary, and you can add your own words to this list to make the game more difficult or easy. The game is available on disk or cassette.

**Hidden Numbers** was developed by Hall Software. It is designed to test the memory skills of the player. Numbers from 1 to 10 are hidden behind several rows of squares. The player or players must locate the squares hiding the same number. The game may be played by persons of all ages and is available on cassette only.

**Hunt the Wumpus** is a search for the Wumpus through a hidden maze of caverns and tunnels. Clues are provided, and the user must evaluate the clues and avoid any dangers while traveling through the maze. This program is available in module form only and sells for \$25.00; order PHM 3023.

**Market Simulation** is a simulation program that pits two persons against each other in a business competition. Many possibilities are programmed into this package, such as economic fluctuations, strikes, etc. Each person selects units in production, advertising cost, and other parameters, to make the game realistic. This is a fun and exciting educational tool that will put its users through the paces of operating a business and dealing with many of the things that occur in the business world. It is available on disk for \$20.00 and on cassette for \$15.00.

**Match Wits** is a game which tests the connection of the player. The game is designed for up to four players aged 16 and older and is available on cassette only (CPW67050). It was developed by Pewterware.

**Mind Challengers** includes two chal-

lenging games. The first game is similar to many hand-held games available commercially and prompts the user to repeat a sequence of notes. A second user is then prompted to repeat the previous sequence plus an additional note. The game continues until one player is unable to echo the correct sequence, up to 64 notes. The second game is a code-breaking game which uses colors and shapes. Both games can be used by persons 10 and older. It is available in plug-in module form for \$25.00; order PHM 3025.

In **Mystery Melody**, a challenging musical game, the user and his opponent are prompted to guess the title of a song based on notes provided. The winner is the person able to name the song in the least number of guesses. One person may play this game alone. This game can be played by the entire family and will provide hours of entertainment. It is available on cassette (PHT 6010) or disk (PHD 5010) for \$10.00 and \$15.00, respectively.

**Oldies But Goodies—Games I** is five games in one. It includes Number Scramble, Word Scramble, Tic-Toe-Toe, Biorhythm and Factor Foe. Each game may be played with the computer or with a human component. The package is designed for use by persons of all ages. It can be purchased on cassette (PHT 6015) or disk (PHD 5015) for \$15.00 and \$20.00, respectively.

**Oldies But Goodies—Games II** contains five games. It contains Hammurabi, Hidden Paris, Peg Jump 3D Tic-Tac-Toe, and Word Safari. Designed for all family members, it is available on cassette (PHT 6017) for \$20.00 or on disk (PHD 5017) for approximately \$25.00

**Othello™** is the computerized version of this popular board game. The players (2) are required to place as many disks on the board as possible, while blocking your opponent. The players take turns trying to amass more disks and outflanking each other. The winner is the player with the most disks on the board at the end of the game. The game is over when neither player is able to make a move. Alternately, one player can be pitted against the computer. This game is available in plug-in module form only (PHM 3067) for \$40.00.

**Peg Jump**, developed by Hall Software, simulates the popular pegboard game in computerized form. You are required to make jumps which will win the game in the fast-packed action game. The game is available on cassette only.

**Pow Wow**, created by Microcomputers Corporation, is a logical deduction number game. The computer first generates a random number, and the user is prompted to guess what the number is. Proficiency is determined by the number of guesses needed. The package is designed for users aged 9 or older and can be bought on disk (MCD0003) or cassette (MCT003).

**Saturday Night Bingo** is a computer simulation of that highly popular game, Bingo. It is a multi-player game in which the computer randomly selects numbers and then reads them out loud through the TI speech synthesizer, an option. Two modes are provided, automatic and manual, so the user may select the speed at which the game is played. This package may be used at home with the entire family, or it may be used by church and other types of organizations that stage Bingo games.



The game is available on cassette (PHT 6025) for approximately \$25.00 or disk (PHD 5025) for \$30.00.

**Scrambled Letters Puzzle & Number and Alphabet Hi-Lo** is a two-game package developed by Hall Software. In the first game you are required to unscramble 15 letters. In the second game, Number and Alphabet Hi-Lo, a number is randomly selected by the computer, and you are prompted to guess the number (or letter). The computer responds to your guesses in a manner which gives you clues as to how close you are to guessing the correct number or letter. This package is available on cassette only.

**Skill Builder I**, designed by Image Producers, consists of two games of skill at different skill levels. The first is Bingo Duel, in which one or two players are provided with problems to solve. In the second game, Number Hunt, the user must match numbers at increasing levels of difficulty. Designed for users 10 and older, the package is available on cassette only (#9406).

**Strategy Games** was developed by Creative Computing to promote learning skill and strategy. Four games are included in this package, including Blockade, Checkers, Darts, and Depth Charge. This package is available on cassette (CS-6003).

**Strategy and Brain Games**, also developed by Creative Computing, combines the programs included in the Brain and Strategy packages previously discussed. This package is available only on disk (CS-6501).

**Strategy Pack I** contains two strategy games in which the user can play against the computer or a human opponent. The first is

Roman Checkers, based on the popular board game of the same name. The second game is called Frame Up. In this game you must use strategy and skill to outwit either the computer or your human opponent. The package was designed by Image Producers and is available on cassette only (#9404).

**Video Chess**, developed with the help of International Master David Levy, is designed for chess players of all ages. It is simple to use, providing help with moves if desired. You can play with the computer or another person, and different levels of play are provided. If desired, a game can be stopped and stored for return to the same game at another time. The computer keeps track of each move, and although it is simple to use, will prove challenging to even the most experienced chess player. The cost is approximately \$70.00, and is available from Texas Instruments in module form; order PHM 3008.

**Yahtzee** was developed by Milton Bradley Company. This is a challenging dice game that many people may be familiar with in its uncomputerized format. Points are garnered by varying dice combinations, and the winner is the person who obtains the most points after a specific number of rolls. Designed for users 8 years old and older, this package is available as a module and sells for \$25.00; order PHM 3039.

### **Fantasy and Adventure**

**Adventure** is a series of games developed by Adventure International. Each require the adventure command module, the TI disk memory system for the disk version, or a cassette recorder and the TI cassette interface cable for the cassette version. These games

were developed to create fantasies which may take as long as a few weeks to complete. The games include Mystery Fun House, Ghost Town, Adventureland, and a host of others. The disk version (PHM3041D) is available for \$50.00, as is the cassette version (PHM3041T).

**Adventureland Adventure Database:** the player in this game is taken on a fantasy trip to a forest in an enchanted world. You must explore the world, searching for treasures and avoiding all obstacles. You are also required to locate the secret place where the treasures are stored. The game is available on disk (PHD 5046) or cassette (PHT 6046) and requires the adventure command module from Texas Instruments. It is priced at \$30.00 for either cassette or disk.

**Alpiner** is a mountain-climbing game which may be played by one or two players. You are presented with a number of obstacles during your climb up a choice of mountains, including Matterhorn, Kenya, McKinley, Garmo, Everest, and Hood. Dangers you must confront include the abominable snowman, lions, bears, skunks, forest fires, avalanches, and rockfalls. The game is colorful and has sound effects as well. Alpiner is available as a plug-in module (PHM 3056) for approximately \$40.00.

**Airmail Pilot** takes you back to the early days of aviation. You are the pilot, and you are given the responsibility of piloting a plane from Columbus to Chicago in the shortest time possible. Many factors are involved in this flight, however, to make it challenging and educational. You must contend with weather conditions, electrical storms, etc. The game was developed by Instant Software and is designed

for persons 10 and older. It is available on cassette only (0274 TI).

In **Chisolm Trail** the user is in control of a steer which must move through a series of mazes, avoiding obstacles and killing monsters blocking the way. Chisolm Trail may be used with the TI optional joysticks and comes in plug-in module form (PHM 3110). Price is about \$40.00.

**The Count Adventure Database** also requires the adventure command module, and sets you back in the days of Dracula in Transylvania. You are given a number of clues and then must determine who you are, what you are doing in Transylvania, and a number of other things. The game is available on disk (PHD 5049) or cassette (PHT 6049) for \$30.00.

**Galactic War** is a space game in which you are in control of a spaceship. You must avoid all obstacles and prevent any danger to your spaceship while trying to destroy all enemy spaceships. This game requires Extended BASIC and is available on cassette (X1100X). The package was designed by Ehninger Associates, Inc.

**Ghost Town** is a treasure-hunting game placed in a setting of an old ghost town. The player must go through the deserted buildings, looking for treasure and avoiding ghosts. This game requires the adventure command module from TI, as well as a disk drive and controller if purchased on disk (PHD 5053) or a cassette recorder and cable if purchased on cassette (PHT 6053). It is priced at \$30.00 for either disk or cassette.

**The Golden Voyage Adventure Database** sends you back in time to a royal palace in a Persian City. You are introduced to a very old king, who is dying. Your missing is to find a

way to restore his youth, equipped with only a bag of gold. Your quest is to find the Fountain of Youth before the kind dies. This adventure requires the adventure command module from TI and can be purchased on disk (PHD 5056) or cassette (PHT 6056) for approximately \$30.00.

**Gorfia Pestulitas** is an unusual adventure game that places you in outer space and pits you against alien ships constantly trying to attack you. You may also opt to have space mines in the game, and two skill levels are provided. This game requires Extended BASIC and was developed by Extended Software Company. It comes on disk or cassette.

**Khe Sanh** is a game in which you are in Vietnam and must defend your base against the enemy. This tactical skill game plots you against approaching forces. It is your job to locate the enemy by means of search and destroy tactics, as well as defend supply convoys approaching, defoliate forests, etc. This game was developed by Not-Polyoptics and can be purchased on cassette only.

**Maze of Ariel** is a game which displays random mazes on the screen consisting of different rooms and paths through them. You are pitted against the computer, and to create difficulty there is a dragon which you must avoid. You are equipped with only a flashlight and a supply of grenades. Developed by Not-Polyoptics, the game comes on cassette only.

**Mission Impossible Adventure Database** is loosely based on the popular television program of the same name which spellbound many persons during its long run. As the game begins, you are listening to a recording in a briefing room. Your mission is to locate a person who has set out to destroy a nuclear reac-

tor, thus destroying the entire world. This action-packed game must be used with the adventure command module and is available on disk (PHD 5047) or cassette (PHT 6047) for approximately \$30.00.

**Mystery Fun House Adventure Database** the player must get inside the fun house, which may not be very easy. Once inside, you are confronted with all the sights you would see in a fun house, and you must search for a prize hidden somewhere inside. The game requires the adventure command module and is available on disk (PHD 5051) or cassette (PHT 6051) for approximately \$30.00.

In **Parsec**, you are the commander of a spaceship in outer space. You are required to do battle with alien ships, which attack in varying patterns, while guiding your ship through refueling tunnels. At different levels of the game, you are attacked by different types of alien ships, and as the levels change, you must guide your ship through obstacles such as asteroid belts. A number of controls are at your disposal, such as speed/sensitivity control, pause capability, and even a female voice which apprises you of your current situation. The graphics and speech capabilities make it quite exciting and entertaining. Joysticks and a speech synthesizer are optional for this game which is available in plug-in module form only (PHM 3112) and sells for \$40.00.

In **Pyramid of Doom Adventure Database** the player is positioned in a desert. Sticking out of the sand is a pole, which marks the point where a pyramid has been discovered. It is your job to find the entrance to the interior of the pyramid, find and collect the treasures, and then escape. This is an exciting adventure game which requires the adventure

command module. It is available on disk (PHD 5052) or cassette (PHT 6042) for \$30.00.

**SAM (surface-to-air missile) Defense** is a simulation game. In this game you are the viewer, watching the firing of a surface-to-air missile. This package is available on cassette only (C1080), and was developed by Ehninger Associates, Inc.

**Santa Paravia and Fiumaccio** was designed by Instant Software. In this game, up to six players can compete to become the ruler of a medieval state (king or queen). Various situations are set up and the players must create a kingdom. This game may be played by persons 12 and older. It is available on cassette only (0273 TI).

**Savage Island I and II Adventure Database** require the adventure command module. This game places you on the edge of a wild and impenetrable jungle. You are required to enter the jungle, with all its creepy creatures, and travel through it successfully. This is a two-part game which is available on disk (PHD 5054) or cassette (PHT 6054) for approximately \$40.00.

**Sengoku Jidai** is a fantasy game set in medieval Japan. Designed for one to three players, you are given a castle and three armies consisting of archers, foot soldiers, and samurai. You must then defend your castle as well as attack the enemy. Each mapboard is randomly selected by the computer, which makes this adventure game different each time it is played. This package was designed by Not-Polyoptics, a division of Synchronet, and is available on cassette.

**Starship Pegasus** is another space adventure game. This game is designed for play by one person only, and provides a different set

of randomly selected circumstances each time it is played. You are positioned inside a spaceship, looking out at the solar system. You are provided with indications of what is occurring by sensors which tell you conditions of a planet, for example. The purpose is to conquer the planet or solar system, while avoiding space pirates. Judgment decisions are involved, as well as skill and coordination. This game was developed by Not-Polyoptics and is available on cassette.

**3D Star Trek** is a three-dimensional computerized version of Star Trek. It is a very challenging and exciting program package, designed for persons 16 and older. Available on cassette, this game was designed by Color Software.

In **Strange Odyssey Adventure Database** the player finds himself stranded on a small planet with a spaceship badly in need of repair. The object is to find the parts needed to repair the ship, while collecting treasures from the ancient civilization's ruins. The game requires the adventure command module and can be purchased on disk. (PHD 5050) or cassette (PHT 6050) for \$30.00.

In **Tombstone City: 21st Century** you are in an Old West ghost town which is being invaded by strange alien creatures which eat people and tumbleweeds. You are given a security force consisting of prairie schooners, and it is your responsibility to protect the ghost town from the aliens. This is a one-player game which will test skill and strategy. It may be used with the optional wired remote controllers from Texas Instruments and is available as a plug-in module only (PHM 3052) for \$40.00.

In **Trail West** you are required to travel

west to California to reach the gold mines. Many obstacles are introduced on this 2,000-mile trip, and you are required to reach your destination without running out of ammunition or supplies. Random events such as storms and overturned wagons are introduced during the trip, and you must use skill and logic to ration supplies and ammunition so they will last the entire trip. The game can be played by young and old and was developed by Micro-Ed, Inc. It is available on disk only (#OT-1).

**Treasure Dive**, developed by Tutorex, takes the player through a search for sunken treasure. You are a scuba diver and are presented with many obstacles such as sea monsters and a limited air supply. Users of all ages will enjoy this game, which is available on cassette only (TUT 6861).

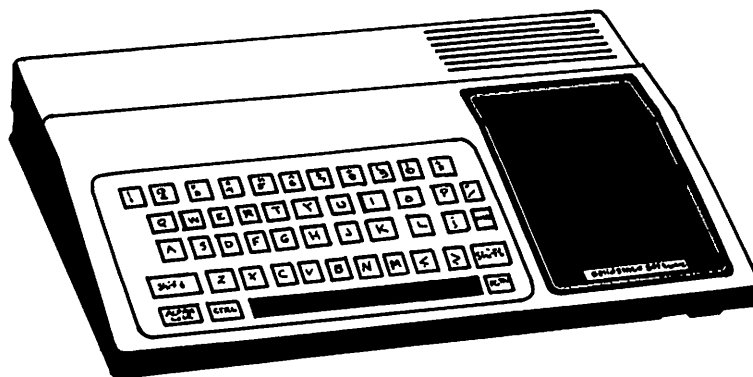
**Tunnels of Doom** is a fantasy/adven-

ture game which takes you back to the age of kings and queens in a search for treasure. This is a role-playing game in which you have many options to choose from in your rescue attempt of the king, while fighting off monsters and other dangers. This game requires the disk drive and controller if purchased as a command module or disk, or a cassette recorder and cable if purchased on cassette. It is priced at \$60.00 for either medium. Order PHM 3042-D for disk and PHM 3042-T for cassette.

**In Voodoo Castle Adventure Database** you are presented with a closed coffin, and it is your job to locate the information needed to free the Count from a curse which has placed him in the coffin. The game requires the adventure command module and may be purchased on disk (PHD 5048) or cassette (PHT 6048) for \$30.00.



## Chapter 11



# Converting to TI BASIC

There are still not too many books around listing programs for the TI-99/4A. Texas Instruments offers a lot of software for the TI-99/4A, but you may want to write your own programs. It is often helpful to see examples of other programs written by computer hobbyists and run them on the TI-99/4A. Later, these programs can be modified to perform other functions that are personally tailored to your needs.

To convert programs written in other dialects of BASIC to TI BASIC requires a bit of modification. This shouldn't be difficult once you have a firm grasp of the statements, commands, and functions in TI BASIC. Many of these are directly interchangeable with other dialects of BASIC.

The following discussion is an overview of statement, command, and function modifica-

tions that may be necessary to get a program written for another machine to run on the TI-99/4A. It doesn't take every other dialect of BASIC into account, but this information will be a worthwhile guideline as to how to proceed.

**ABS** This is the absolute value function and is used the same way in every dialect of BASIC. If you see this function in a BASIC program written for another computer, you should not have to make any modifications to get it to run on the TI-99/4A.

**ATN** The ATN function returns the arctangent of an argument. I have never seen it used in a different manner than that specified in the Texas Instruments manual, and you should not have to make any modifications.

**ASC** The ASC function gives you the ASCII character code that corresponds to the

first letter in a string argument. It is used in about the same way by all BASIC dialects.

**Break** The Break command is not encountered outside of TI BASIC, but it is similar to Stop and Wait commands in other dialects. In converting programs to TI BASIC, you will probably never have to use the Break command or the Unbreak command.

**Bye** I have not seen this command outside of TI BASIC. It is equivalent to the New command where BASIC is exited to return to the main operating system. Since this is a command, it should not be encountered in any BASIC program lines.

**Call Char** This subprogram is found only in TI BASIC. When one becomes involved in graphics programming, the differences in the dialects of BASIC can be so great that it's easier to write a new program from scratch than to convert. On other machines, this kind of feature may not be possible, at least not through a single command.

**Call Clear** This is identical to the CLS or HOME command found in most other BASIC dialects. This may be used as a command or a statement. When it occurs within a program, it is a statement. Call CLEAR may be directly substituted for CLS when converting programs to TI BASIC.

**Call Color** This is equivalent to Color statements in other dialects of BASIC. The numerical commands that follow Color may vary from machine to machine, depending on the number of foreground and background colors offered. You will probably have to experiment with the Call Color/Color conversion. This can be done after the major portion of the program has been modified and is running. Color statements deal mostly with

graphics, and it is far easier to convert programs from one dialect of BASIC to another that deal with text mode operations. Graphics changes often encompass rewriting an entire program.

**Call HCHAR/Call VCHAR** These subprograms do not really correspond to any other statements in other BASIC dialects. They are roughly equivalent to Locate statements found in IBM BASIC and TRS-80 Color Computer Extended BASIC or to PRINT @ statements in TRS-80 BASIC or Apple BASIC. In these other dialects, the statements are used to print information at a specific point on the screen, whereas in TI BASIC, the subprograms are more often associated with graphics rather than text. In most applications involving a printout of text mode information, Locate statements can be omitted from the TI version of the program and PRINT@ replaced with Print. These statements are often used to display information at certain points on the screen to make the output more readable. In this case, the omission of Locate or the changing of Print @ to Print will not make a significant difference. If those statements are used to form charts, a conversion may not be possible.

**Call JOYST** This is the joystick subprogram, which may roughly correspond to the Stick function in other dialects of BASIC.

**Call Key** The Call Key subprogram is quite similar to the On Key, INKEY\$, GET, and Key On statements in other dialects of BASIC. These statements are used to activate a certain key or set of keys on the keyboard and create branches when the key is activated.

**Call Screen** This subprogram loosely corresponds to Screen statements in other dialects of BASIC. However, the numbers



which designate foreground and background colors are probably different from dialect to dialect. In most instances, the Call SCREEN subprogram can be used to replace a Screen statement, but it will be necessary to coordinate the numbers between the two dialects. In IBM BASIC, the Screen statement is used to establish screen mode, which has to do with screen width and resolution.

**Call Sound** This corresponds with Sound, Beep, and Play statements in other dialects of BASIC. The Call Sound subprogram may be used to directly replace BEEP. The Sound statement in IBM BASIC can also be converted with Call Sound, although the duration portion of the command must be altered to reflect TI-99 nomenclature. There is no volume command, but this will be of little significance in making the conversion to TI BASIC.

**CHR\$** This function returns the character corresponding to the ASCII character code and is common to all dialects of BASIC. However, different computers have different character sets. It is necessary to compare character sets and ASCII codes to determine which character is being specified by an ASCII number. In IBM BASIC, ASCII code 219 is a block character used in text mode graphics. There is no equivalent to this character in the TI-99 character set. In such instances, a conversion may not be possible.

**Close** The Close statement is used to discontinue access to a file. In many cases, it may be used interchangeably from dialect to dialect.

**Continue** This command continues execution after a program halt has been performed. In TI BASIC, you may use CON, as

well as Continue. Both are the equivalent of CONT, which is common to most other dialects of BASIC.

**COS** This is the cosine function and returns a cosine of the argument. It is used in the same way for all dialects of BASIC.

**Data** Data statements may be used interchangeably in most dialects of BASIC.

**DEF** The DEF statement is used to define your own function if used within a program. Most dialects of BASIC expand on DEF, but most contain a DEF FN statement, which is interchangeable with DEF in TI BASIC.

**Delete** The Delete command in TI BASIC is used to erase a program or data file from a disk. Its use in TI BASIC is different from other dialects. In TRS-80 and IBM BASIC, the Delete command is used to delete lines from a program held in memory. In IBM DOS, Delete is used for the same purpose as in TI BASIC. In most dialects of BASIC, the Kill command is used to erase programs from disks, Delete in TI BASIC corresponds to Kill in other dialects in most instances.

**DIM** The dimension statement is common to all dialects of BASIC. No changes are usually necessary.

**Display** You probably won't find this elsewhere. It's rarely used in TI BASIC, as the Print statement really does the same thing.

**Edit** This command displays a line for editing purposes and usually works the same on all machines.

**End** The End statement stops program execution and is interchangeable with all other dialects of BASIC.

**EOF** The EOF function indicates an end of file condition. This function is common

to most dialects of BASIC, and no changes should be necessary as long as you use the number specified in all other Open and Close statements as an argument in the EOF function.

**EXP** The exponential function may be used interchangeably in most dialects of BASIC.

**For-To-Step** This is a statement common to all forms of BASIC and may be used interchangeably.

**GOSUB/GOTO** These branch statements are common to most forms of BASIC. They may be used interchangeably.

**If-Then-Else** This statement is common to most forms of BASIC, but there are subtle differences. In TI BASIC, If-Then must be followed by a branch line. In other forms of BASIC, If-Then may be followed by statements or commands, as in If A = 6 Then Print "HELLO". This last line is not legal in TI BASIC and would have to be modified to:

```
10 IF A = 6 THEN 20 ELSE 30
20 PRINT "HELLO"
30 END
```

If-Then-Else statements in other dialects of BASIC may also include Boolean operators, such as And, Or, etc., as in:

```
10 IF A = 6 and B = 10 THEN 30
   ELSE 20
20 END
30 PRINT "HELLO"
```

In TI BASIC, this would have to be modified to:

```
10 IF (A=6) * (B=10) < > 0
```

```
      THEN 30 ELSE 20
20 END
30 PRINT "HELLO"
```

All If-Then-Else statements in other dialects of BASIC can be converted to run in TI BASIC, although more lines will be required. In TI Extended BASIC, statements may follow If-Then-Else statements and Boolean operators are allowed.

**Input** The Input statement is common to most dialects of BASIC. TI BASIC requires that a colon follow any quoted phrase in an Input statement, as in:

```
10 INPUT "PRESS ENTER TO
CONTINUE":A$
```

In other dialects of BASIC, a colon is rarely used. Instead, you will find a comma or a semicolon. This applies only when a quoted phrase follows Input. Such program lines as Input A, Input A\$, etc. need no changes at all to run in TI BASIC.

**INT** This is the integer function and usually requires no modification to run in TI BASIC.

**LEN** This function is common to most dialects of BASIC, reading the characters in a string statement. No modification is usually required.

**Let** This statement is optional in TI BASIC, as is the case with IBM BASIC and a few other dialects. Let statements require no modifications to run in TI BASIC, nor is it necessary to add them when modifying programs from dialects where Let is not used.

**List** List is common to most dialects of BASIC and lists the lines of the program currently in memory.

**LOG** The LOG function returns the natural logarithm of a number. This is usually an interchangeable function and needs no modification to run in TI BASIC.

**New** New is a command used to erase the current program from memory in order to make way for the writing of a new program. It is common to most dialects.

**Next** See For-To-Step.

**Number** This is a command in TI BASIC that may also be input as NUM. It is used to automatically generate line numbers each time Enter is pressed. It has equivalents in most other dialects of BASIC. Other equivalents include NUMS, Numbers, and Auto.

**OLD** This command loads a program from cassette. It most often corresponds to the Load command in other dialects of BASIC.

**ON-GOSUB/ON-GOTO** These statements are common to most dialects of BASIC. See GOSUB/GOTO.

**OPEN** The Open statement is used to open a file and may often be used interchangeably.

**Print** All dialects of BASIC contain a Print statement, which displays information on the monitor screen. It is necessary to make modifications in some cases. For example, the following lines are legal in TI BASIC:

```
PRINT A
PRINT A$
PRINT "HELLO",A
PRINT "HELLO";A
PRINT A,"HELLO"
PRINT A;"HELLO"
```

The following lines may be legal in other dialects of BASIC, but not in TI BASIC:

```
PRINT A"HELLO"
PRINT "HELLO" A
PRINT "HELLO" A "HELLO AGAIN"
```

Whenever a variable is used on the same line with a quoted phrase, the variable must be separated from the phrase by either a comma or a semicolon. Many dialects of BASIC do not require this.

**Randomize** In TI BASIC, Randomize can usually be used to directly replace Randomize statements in other programs. Some dialects require that a number follow the Randomize statement, such as Randomize 32, or RANDOMIZE X, where X is an assigned numeric variable. In most instances, you can replace the latter two examples with the single Randomize statement in TI BASIC.

**Read** The Read statement is common to all dialects of BASIC and can usually be used interchangeably.

**REM** Remark statements are non-executable and may be used interchangeably.

**Resequence** You may also use RES. This command is used to sequentially renumber computer program lines. This is equivalent to RENUM in other dialects of BASIC.

**Restore** Restore statements can usually be used interchangeably when used to return a data line to the first item.

**Return** Always paired with the GOSUB statement, Return may be used interchangeably.

**RND** The RND function can usually be used interchangeably, but some machines randomize in a slightly different manner.

**RUN** The Run command is common to most dialects of BASIC.

**Save** The Save command is used to copy the current program into memory. This command is used similarly in most dialects of BASIC.

**SEG\$** In TI BASIC, the SEG\$ function takes the place of Left\$, Right\$, and MID\$ in some other dialects of BASIC. Any of these three functions can be replaced by SEG\$. You will rarely encounter this function outside TI BASIC.

**SGN** This is the signum function, which gives you the algebraic sign (plus, minus, zero) of an argument. It may be used interchangeably in most dialects of BASIC.

**SIN** This function returns the sine of a number and may be used interchangeably.

**SQR** This function returns the square root of a number and is used identically in most dialects of BASIC.

**Stop** The Stop statement is common to most dialects of BASIC and should not have to be modified when encountered in other programs.

**STR\$** This function returns a string representation of the value of an argument and may be used interchangeably.

**Tab** The Tab function may be used interchangeably, but it is often necessary to know the differences between screen formats of machines. The TI-99/4A is capable of a 32-column screen, and some other machines may have 40 or 80 column capability.

**TAN** The TAN function returns the tangent of a number and can usually be used interchangeably.

**VAL** The VAL function converts a string variable to a numeric variable and may be used interchangeably.

In its basic form, the TI-99/4A does not allow multiple statements on a single program line. Other dialects of BASIC may, and indeed, TI Extended BASIC adds this capability.

Machines that allow multi-statement lines separate statements in different manners. One machine may use the following format:

```
10 LET A = 10/PRINT A
```

Two statements are contained on this line. The first assigns the variable A a value of 10, while the second prints the value of A on the screen. The TI-99/4A would require the following modification:

```
10 LET A = 10
20 PRINT A
```

Multiple statement lines are illegal in TI BASIC, so each statement must go on a separate line. In another dialect of BASIC, the multiple statement equivalent might be:

```
10 LET A = 10:PRINT A
```

Here, the colon is used to delineate multiple statements. Some dialects may use a semi-colon or brackets.

Multiple statement lines that include If-Then-Else statements can be a bit tricky. Take the following line, for example:

```
10 IF A = 10 THEN PRINT
   "HELLO":PRINT "YELLOW"
20 PRINT "GOODBYE"
30 END
```

You might think that a conversion to TI BASIC would involve the following modification:

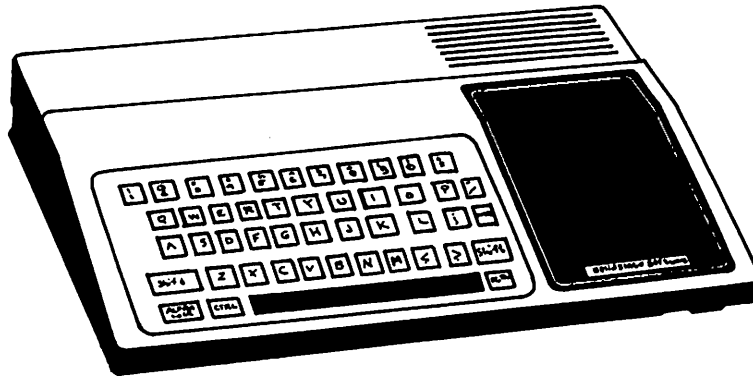
```
10 IF A = 10 THEN 15 ELSE 20
15 PRINT "HELLO"
20 PRINT "YELLOW"
25 PRINT "GOODBYE"
30 END
```

This won't work. Line 10 contains an If-Then statement. Therefore, the second statement on that line (PRINT "YELLOW") will be executed only if A equals 10. In the modified program, PRINT "YELLOW" will occur whether or not A is equal to 10. The correct modification is

```
10 IF A = 10 THEN 15 ELSE 25
15 PRINT "HELLO"
20 PRINT "YELLOW"
25 PRINT "GOODBYE"
30 END
```

If A equals 10, the word "HELLO" is printed. If not, the word "GOODBYE" is printed and the

program ends. If A is not equal to 10, there is a branch to line 25, where the word "GOODBYE" is printed and the program ends. This program is encountered because the modification must take place to the If-Then statement as well as to the second statement following on the same line. If you stick to text mode programs written in other dialects of BASIC for your first few exercises in program conversion, this will allow you to ease into such conversions. As you become more familiar with your TI-99, you will discover the programming methods used to accomplish certain functions. You may then be able to look at other programs and duplicate the same functions, not by modifying these programs, but by writing a whole new program on your own. This can only come from close familiarity with the TI BASIC language. If you plan to eventually go with TI Extended BASIC (a \$100 option), a lot of conversion problems will disappear because Extended BASIC is more powerful than TI BASIC and allows other programming features common to other dialects of BASIC.



## Glossary

**access**—The manner in which files are referred to by a computer.

**accessory device**—Any equipment that attaches to the computer to allow it to expand its functions. Accessory devices are usually limited to hardware or firmware, as opposed to software.

**access time**—The interval between the application of an input pulse and the availability of data signals at the output.

**algorithm**—An algorithm is a set of rules or a standard procedure that provides the solution to a problem. In a computer program, an algorithm is the most efficient method of achieving a specific goal. In this case, efficient would most likely refer to a minimum

number of statements, functions, and commands or the shortest program possible to achieve the goal.

**alphanumeric**—Alphanumeric describes characters which include the letters of the alphabet, numerals, and symbols used for punctuation and mathematical operations.

**array**—An array is a group or table of values referenced by the same name when programming in BASIC. Each item or value in the array is often referred to as an element. Array elements are variables and can be used in expressions and in BASIC statements or functions which allow the use of variables.

**ASCII**—ASCII is an acronym for American

Standard Code for Information Interchange. This is an 8-level code (7 bits plus parity check) that is widely used for information interchange. This code structure is used in the TI-99/4A and most personal computers to represent letters, numbers, symbols, and special characters.

**assigned statement**—An assigned statement is a line in a computer program that assigns a value of an expression to a variable. On the TI-99/4A the LET statement or the equal sign may be used.

**asynchronous**—Asynchronous is a mode of computer operation in which performance of the next command is activated by a signal indicating that the previous command has been completed.

**BASIC**—An acronym for Beginners All Purpose Symbolic Instruction Code, BASIC is a programming language that is used to write programs. A BASIC program consists of one or more statements, functions, or commands preceded by line numbers. These numbers control the sequence in which the instructions are run. The TI-99/4A is programmed in TI BASIC. The BASIC language used with all computers is quite similar. Different types of computers may alter the BASIC language slightly to conform to certain machine standards.

**binary**—Binary is a number system based on only two digits, 0 and 1. The internal language and operations of digital computers

are most often based on the binary system.

**bit**—Bit is an abbreviation for binary digit. This is an information unit equal to one binary decision or the designation of one or two possible values. These values may be referred to as high/low, 1/0, yes/no, off/on, etc.

**Boolean algebra**—Boolean algebra is a deductive system or process of reasoning named after George Boole, an English mathematician. It is a system of theorems that uses symbolic logic to denote classes of elements, true or false propositions, and on-off logic circuit elements. Symbols are used to represent operators such as And, Or, Not, Except, If-Then, etc. This system is now recognized as an effective method of handling single-valued functions with two possible output states. When Boolean algebra is applied to binary arithmetic, the two states become 0 and 1. When applied to switching theory, the two states become open and closed.

**branch**—A branch is a break in the sequential execution of a program. A branch causes the computer to jump to another portion of the program. In TI BASIC, statements that set up branches include GOTO, GOSUB, and If-Then-Else. There are two types of branches, conditional and unconditional. An unconditional branch is conducted each time the line that includes the branch instruction is executed. A conditional branch brings about the jump only upon the result of some

## **branch—character set**

---

logical or arithmetic operation. GOTO and GOSUB statements are used most often to bring about unconditional branches, and If-Then-Else is used only for conditional branching.

**breakpoint**—In TI BASIC, a breakpoint is the point in a program at which execution is halted by the BREAK command. After execution has been suspended, you can perform operations in Command Mode to help locate program errors. To resume execution after a breakpoint, type **CONTINUE** and the press Enter.

**buffer**—A buffer is a device or area of computer memory that serves as an isolator or interface to dissimilar elements. In computer terminology, a buffer is usually thought of as a storage device. It may store input or output information transmitted at one rate until another station can use the data. The output from the Texas Instruments computer to the printer is transmitted at a much faster speed than the printer can transfer to paper. The print buffer receives the output from the computer at its normally transmitted rate. It stores the information until the printer could accept it all at its own speed.

**bug**—A bug is an error. The term applies especially to software errors. When a program is first written, it must often go through a debugging process. This is a matter of removing all errors.

**bus**—A bus is a conductor through which in-

formation is transmitted or received.

**byte**—A byte is a string of binary digits that form one unit. A byte is equal to one character letter, number, space, or punctuation mark. Computer memory capacity is specified in bytes. The TI-99/4A makes available 16,000 bytes of read/write memory (RAM). This may also be specified as 16K bytes, with the K means multiply by 1000.

**card**—In microcomputer jargon, a card is a plug-in circuit board. The Peripheral Expansion System used with the TI-99/4A makes available slots for inserting these circuit boards, or cards. The plug-in modules contain internal cards, but since they are enclosed in one unit, the term module applies.

**cassette storage**—Cassette storage is a system for reading information from a previously programmed cassette tape and/or the ability to write information onto a blank cassette tape.

**cathode ray tube**—Abbreviated CRT, a cathode-ray tube is a device that displays information. Your television picture tube is a cathode ray tube, and all monitors, such as the TI 10-Inch Color Monitor, contain them.

**character**—A character is a letter, number, or symbol that can be produced (usually on the screen) by a computer.

**character set**—A character set is a set of



representations, called characters, from which selections are made to denote and distinguish data. A set may include the numerals 0 to 9, the letters A to Z, punctuation marks, and a blank or space.

**chip**—A chip is a thin piece of silicon material. Solid state devices use a single chip to produce highly complex circuits, all contained on the chip surface. More common terminology lets chip be used to describe integrated circuits.

**code**—A code is a system of symbols for representing data or instructions in computers. Code also means to translate the program instructions acceptable to that computer.

**collate**—An operation in which two or more sets of data are merged to produce one or more sets that still reflect the original ordering relations.

**command**—A command is an instruction that is not a part of a program and which the computer can perform immediately upon input.

**command module**—The Texas Instruments Command Modules are preprogrammed read-only memory circuits enclosed in an insertable package for connection to the TI-99/4A console. These modules contain different languages and programs.

**concatenation**—Concatenation is the process of linking together in a series.

**constant**—A constant is a numeric or string value specified by specific letters and/or numbers. A numeric constant is any real number, and a string constant is any combination of letters and numbers.

**cursor**—A cursor is a symbol that appears on the monitor to indicate where the next character will appear. In TI BASIC, the cursor is usually represented by flashing block.

**data**—Data are a graphic or textual representation of facts, concepts, numbers, letters, symbols, or instructions for communication, interpretation, or processing. Data form the basic elements of information. (See DATA Statement)

**Data statement**—In TI BASIC, a Data statement allows for items to be contained within a program line. A matching Read statement reads each data item when commanded to do so. A Data statement may contain as many constants as will fit on a program line, and any number of Data statements may be used within a program. The items contained in Data statements are always read sequentially.

**debug**—Debug is a computer term used to describe the detecting and removing of errors and malfunctions from a program or from the computer itself.

**default**—A default is a characteristic or value that the computer assumes to be true unless otherwise negated.

## **digital system—flowchart**

**digital system**—A digital system is a device or circuit that deals in digital rather than analog form. It operates on a binary number configuration using 2 as a base and the digits 0 and 1 as values which are referred to as bits. Combinations of these bit values provide the code by which data can be processed through its electronic circuitry.

**DIP**—DIP is an abbreviation for dual in-line package. This describes many types of integrated circuit packaging. DIP packages resemble long, flat wafers with pins extruding the longer edges.

**directory**—A directory is the area on a disk in which the names of files are stored. Included in a directory may be information about the size of the file, its location on the disk, and the date it was created.

**disk**—A disk (also called a floppy disk) is a mass storage device. It is a flexible circular object coated with a magnetic substance. When in use, the disk spins inside a permanent protective jacket. The disk drive contains a magnetic head to read and write information from the disk.

**edit**—Editing involves the deletion, insertion, and rearrangement of data.

**error message**—An error message is a screen prompt that appears after entering a line incorrectly, when trying to run a program with incorrect lines, or during the program run itself. The error message indicates

what the problem may be and may generate an error message number for easy reference in the TI reference manual.

**executable statement**—Executable statements are program instructions that tell BASIC what to do while executing a program.

**execute**—Execute means to run a program.

**exponent**—An exponent is a number which indicates the power to which another number or expression is to be raised. In TI BASIC, the exponent is written to the right of the number or expression, separated from it by the ^ symbol which is typed in via the keyboard.

**expression**—An expression is a combination of variables, constants, and operators that can be evaluated to a single result.

**file**—A file is a collection of information usually stored on cassette tape or disk.

**fixed-length record**—Fixed-length records are data in a file that are all the same length. If the data does not fill the record, the computer adds blanks.

**flowchart**—A flowchart is a graphical representation of the definition or solution of a problem, in which symbols are used to represent functions, operations, and execution flow. A flowchart contains the logical steps in a program so the designer can concep-

tualize and visualize each step. It defines the major phases of the processing, as well as the path to problem solution.

**formatting**—Formatting is the process of setting up a disk to receive information. This process checks the disk for bad spots and builds a directory to hold information about the files that will be written on it.

**function**—A function is a feature that lets you specify as single operations a variety of procedures. Each procedure may actually contain a large number of steps.

**function keys**—On the TI-99/4A, the keyboard function keys perform special functions when pressed simultaneously with the FCTN key. These functions can include the printing of quotation marks in a program, or the deletion, insertion, or complete erasure of characters or lines in a program while in Edit Mode.

**graphics**—Graphics include simple drawings, random patterns, and graphs.

**hardware**—Hardware refers to the physical components that make up the microcomputer. A monitor, printer, cassette recorder, and disk drive, are hardware.

**hexadecimal**—Hexadecimal describes a number system which has a base of 16 and uses 16 symbols. These symbols are the numbers 0 through 9 and the letters A through F.

**instruction**—An instruction defines an operation and causes the computer to actually perform the operation.

**integrated circuit**—Abbreviated IC, an integrated circuit is an interconnected array of components fabricated from a single crystal of semiconductor material (usually treated silicon).

**integer**—An integer is a positive or negative whole number, such as 1, 2, 3, 4, or 5. Zero is an integer, but numbers such as 1.12, 2.333, and 4.115 are not.

**interface**—An interface lets a computer operate into a communications line, a terminal, or into peripheral devices.

**I/O**—I/O is an abbreviation for input/output. An I/O channel is a circuit path that allows communications between the processor and devices including the keyboard, a disk drive, a cassette player, etc.

**iteration**—A programming technique of repeating a group of program statements. For-Next loops are often used for this purpose.

**joystick**—A joystick is a lever that provides coordinate data of a display surface. The data can control operations, such as the movement of one or more display elements. Joysticks are often used in computer games or to manipulate data about the screen.

## **keyboard—modem**

---

**keyboard**—The keyboard contains keys for entering data or information into the system.

**light pen**—A light pen is a photosensitive device that causes the computer to modify the display on the monitor screen. The light pen signals the computer using an electronically produced pulse. The light pen can draw impressions on the monitor screen, as well as read points of light from computer-generated displays.

**logical operator**—Logical operators perform logical, or Boolean, operations on numeric values. Logical operators are usually used to connect two or more relations and return a true or false value to be used in a decision. A logical operator takes a combination of true-false values and returns a true or false result. An operand of a logical operator is considered to be true if it is not equal to zero, or false if it is equal to zero. The result of the logical operation is a number that is true if it is not equal to zero, or false if it is equal to zero. The number is calculated by performing the operation bit by bit. The logical operators are Not (logical complement) And (conjunction), Or (disjunction), XOR (exclusive OR), IMP (implication), and EQV (equivalence).

**logic expression**—A logic expression consists of variable array elements, function references, logic constants, and combinations of operands separated by logical operators and parentheses. Typically, logi-

cal expressions may contain arithmetic expressions separated by relational operators.

**loop**—A loop is the repeated execution of a series of instructions usually for a specific number of times. For-Next statements are often used to establish such loops.

**machine language**—Machine language is used directly by a microprocessor. All other languages must be translated or compiled into binary code before being executed by the processor.

**matrix printer**—A matrix printer is a device that uses an array of dots to form characters.

**memory**—Memory in a computer stores information. Random-access memory (RAM) is the memory section to which operator programs are written and stored for execution. Read-only memory (ROM) is not accessible for storage, having been programmed at the factory to allow the computer to perform its built-in functions.

**microprocessor**—A microprocessor is normally a single-chip computer element that performs the logical and controlling functions in a microcomputer.

**modem**—A modem is an electronic device that performs the modulation and demodulation functions required for communications. A modem can be used to connect computers and terminals over telephone circuits.

**module**—A module is an assembly which contains a complete circuit or subcircuit. Printed circuit boards designed to be plugged into a computer may be classified as modules.

**monitor**—A monitor most often refers to the display screen. It also means a unit in a computer that prepares machine instructions from a source code. It may use built-in compilers for one or more program languages. The machine instructions are sequenced into the processing unit once compiling is complete.

**nanosecond** —A nanosecond is an amount of time equal to  $10^{-9}$  second. It is abbreviated ns and is equivalent to 1/1,000,000 of a second. A time interval of 1,000,000 nanoseconds is equal to 1 second.

**non-executable statement** —A non-executable statement does not cause any program action. The statements are there within the program, but they are passed over and not acted on. Two examples of non-executable statements are REM and DATA.

**null string** —A string that has no value is a null string. It contains no characters and has a length of 0.

**number mode** —This is an automatic line numbering mode set up by the NUM command in TI BASIC. Once this command is

executed, a number will be generated on the screen whenever the Enter key is pressed.

**numeric comparison**—In numeric comparison, when arithmetic and relational operators are combined in one expression, the arithmetic is always performed first. For example, the expression:

$$X + Y < (T - 1)/Z$$

will be true (–1) if the value of X plus Y is less than the value of T–1 divided by Z.

**numeric expression**—A numeric expression may be a numeric constant or variable, or may be used to combine constants and variables using operators to produce a single numeric value. Numeric operators perform mathematical or logical operations mostly on numeric values, and sometimes on string values. They are referred to as numeric operators because they produce a value that is a number.

**numeric function**—A function is used like a variable in an expression to a predetermined operation that is to be performed on one or more operands. BASIC has intrinsic functions that reside in the system, such as SQR (square root) or SIN (sine).

**operator**—An operator is a symbol used in performing arithmetic calculations. In the calculation  $1 + 1 = 2$ , the operator is +. Common operators are \*, +, –, /, ^, These

### **operator—scientific notation**

stand for multiply, add, subtract, divide, and raise to.

**power supply**—A power supply is an electric circuit that supplies operating voltage and current to the computer.

**program**—A program is a set of instructions that direct a computer in performing a desired operation, such as the solution of a mathematical problem or the sorting of data.

**program line**—Any line preceded by a line number and containing a statement is called a program line.

**prompt**—A symbol or phrase that appears on the display screen to signal that an input is needed is called a prompt.

**pseudo-random-number**—A pseudo random number is logically produced, but the set of calculations used to generate it are so complex that an outcome cannot be logically deduced by a human being. Computers output pseudo-random numbers as opposed to random numbers, the latter being generated by chance.

**RAM**—A pseudonym for random-access memory, RAM is the user programmable internal memory of a computer. The computer can store values in distinct locations in random access memory and recall them again, or alter and restore them. The values in RAM are lost when the power is turned off.

**record** —A record is a collection of related data elements. When records are combined into relational groups, they are called a file.

**register** —A register is a storage area in memory having a specified storage capacity, such as a bit, a byte, or a computer word, and intended for a special purpose.

**reserved word** —A reserved word has a pre-defined meaning in a specific computer language. Reserved words have special meaning and include all BASIC commands, statements, function names, and operator names. Reserved words may not be used as variable names.

**RF modulator**—An RF modulator accepts an audio or video input and then place this information on a carrier, usually at radio frequencies. They connect computers to television receivers.

**ROM**—An abbreviation for read-only memory, ROM holds important programs or data that must be available to the computer when first activated. Information in ROM is unalterable and does not disappear when the power is turned off.

**scientific notation**—A method of expressing numbers that are extremely large or small by using a base number or mantissa multiplied by 10 raised to some power is known as scientific notation. 2,000,000 may be represented in scientific notation by 2E6.

### scientific notation—truncation

The E6 stands for 10 raised to the sixth power, or 1,000,000.

**scrolling**—When a screen is filled, the display moves upward, or scrolls, one line at a time to allow additional lines to be entered at the bottom. When this occurs, the top line disappears.

**software**—Software encompasses all types of computer programs, including programs contained in ROM, those stored on magnetic media, and those entered from the keyboard. Any program that can be executed by the computer may be considered as software.

**statement**—A statement is an instruction preceded by a line number. Some computers allow program lines to contain multiple statements.

**storage**—Storage is used to describe a device or medium on or into which data can be entered, held and retrieved at a later time. Storage may use electrostatic magnetic, acoustic, optical, electronic or mechanical methods. This term is synonymous with memory.

**string**—A string is a sequence of items grouped in series according to certain rules.

**string constant** —A string constant is a sequence of up to 108 characters enclosed in

double quotation marks. It is a type of actual value which BASIC uses during execution.

**subprogram**—In TI BASIC, a subprogram is a general-purpose procedure made accessible by using the Call statement. Subprograms are held in ROM and extend the capability of BASIC.

**subroutine**—A subroutine is a segment of a program that can be executed by a single call. Subroutines are used to perform the same sequence of instructions at different places in a single program.

**telecommunications**—Telecommunications is data transmission between a computer and remotely located devices.

**terminal** —A terminal is a part of a computer system that is used for entering or outputting information. It usually includes a keyboard and monitor.

**Trace**—Trace is a command in BASIC that lets you see the order in which the computer executes statements during a program run. When the trace is in effect, the program line number is generated as it is executed. This is a valuable debugging aid.

**truncation** —Truncation is the deletion or omission of a portion of a string. It may also be the termination of a computation process before its final conclusion or natural termination.

### **update—variable**

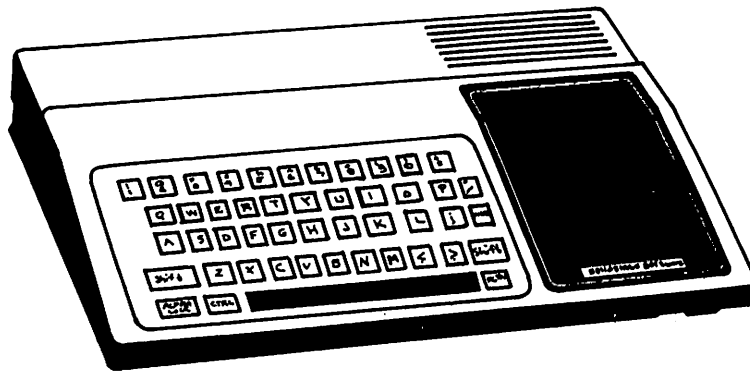
**update**—To update means to modify current information with new information.

**variable**—Variables are names used to represent values being used in a BASIC pro-

gram. There are two types of variables: numeric and string. A numeric variable always has a value that is a number. A string variable may only have a character string value.



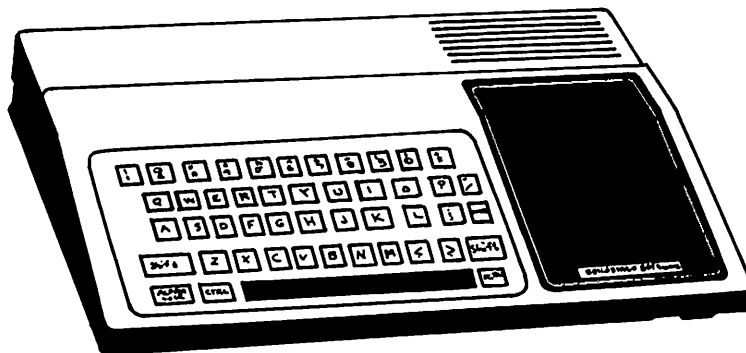
## Appendix A



## Reserved Words in TI BASIC

|          |          |           |            |          |
|----------|----------|-----------|------------|----------|
| ABS      | DIM      | LEN       | RANDOMIZE  | SQR      |
| APPEND   | DISPLAY  | LET       | READ       | STEP     |
| ASC      | EDIT     | LIST      | REC        | STOP     |
| ATN      | ELSE     | LOG       | RELATIVE   | STR\$    |
| BASE     | END      | NEW       | REM        | SUB      |
| BREAK    | EOF      | NEXT      | RES        | TAB      |
| BYE      | EXP      | NUM       | RESEQUENCE | TAN      |
| CALL     | FIXED    | NUMBER    | RESTORE    | THEN     |
| CHR\$    | FOR      | OLD       | RETURN     | TO       |
| CLOSE    | GO       | ON        | RND        | TRACE    |
| CON      | GOSUB    | OPEN      | RUN        | UNBREAK  |
| CONTINUE | GOTO     | OPTION    | SAVE       | UNTRACE  |
| COS      | IF       | OUTPUT    | SEG\$      | UPDATE   |
| DATA     | INPUT    | PERMANENT | SEQUENTIAL | VAL      |
| DEF      | INT      | POS       | SGN        | VARIABLE |
| DELETE   | INTERNAL | PRINT     | SIN        |          |

## Appendix B

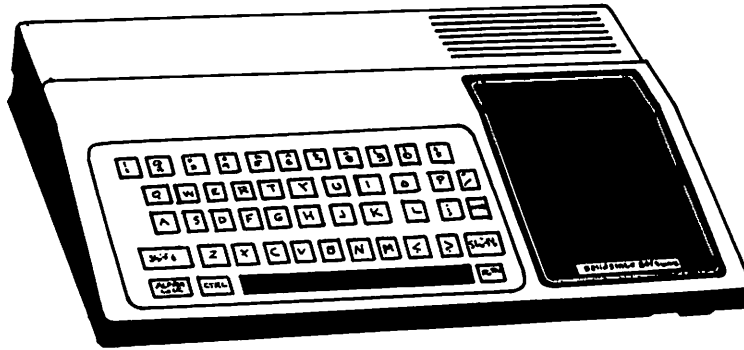


# ASCII Character Codes

| ASCII<br>CODE | CHARACTER                | ASCII<br>CODE | CHARACTER         | ASCII<br>CODE | CHARACTER   |
|---------------|--------------------------|---------------|-------------------|---------------|-------------|
| 32            | (space)                  | 48            | 0                 | 64            | @ (at sign) |
| 33            | ! (exclamation point)    | 49            | 1                 | 65            | A           |
| 34            | " (quote)                | 50            | 2                 | 66            | B           |
| 35            | # (number or pound sign) | 51            | 3                 | 67            | C           |
| 36            | \$ (dollar)              | 52            | 4                 | 68            | D           |
| 37            | % (percent)              | 53            | 5                 | 69            | E           |
| 38            | & (ampersand)            | 54            | 6                 | 70            | F           |
| 39            | ' (apostrophe)           | 55            | 7                 | 71            | G           |
| 40            | ( (open parenthesis)     | 56            | 8                 | 72            | H           |
| 41            | ) (close parenthesis)    | 57            | 9                 | 73            | I           |
| 42            | * (asterisk)             | 58            | : (colon)         | 74            | J           |
| 43            | + (plus)                 | 59            | ; (semicolon)     | 75            | K           |
| 44            | , (comma)                | 60            | < (less than)     | 76            | L           |
| 45            | - (minus)                | 61            | = (equals)        | 77            | M           |
| 46            | . (period)               | 62            | > (greater than)  | 78            | N           |
| 47            | / (slant)                | 63            | ? (question mark) | 79            | O           |

| ASCII<br>CODE | CHARACTER         | ASCII<br>CODE | CHARACTER | ASCII<br>CODE | CHARACTER                                 |
|---------------|-------------------|---------------|-----------|---------------|---|
| 80            | P                 | 97            | A         | 114           | R   |
| 81            | Q                 | 98            | B         | 115           | S   |
| 82            | R                 | 99            | C         | 116           | T   |
| 83            | S                 | 100           | D         | 117           | U   |
| 84            | T                 | 101           | E         | 118           | V   |
| 85            | U                 | 102           | F         | 119           | W   |
| 86            | V                 | 103           | G         | 120           | X   |
| 87            | W                 | 104           | H         | 121           | Y   |
| 88            | X                 | 105           | I         | 122           | Z   |
| 89            | Y                 | 106           | J         | 123           | { (left brace)                            |
| 90            | Z                 | 107           | K         | 124           | :   |
| 91            | [ (open bracket)  | 108           | L         | 125           | } (right brace)                           |
| 92            | \ (reverse slant) | 109           | M         | 126           | ~ (tilde)                                 |
| 93            | ] (close bracket) | 110           | N         | 127           | DEL (appears on<br>screen as a<br>blank.) |
| 94            | ^                 | 111           | O         |               |   |
| 95            | _____ (line)      | 112           | P         |               |   |
| 96            | ` (grave)         | 113           | Q         |               |   |

## Appendix C

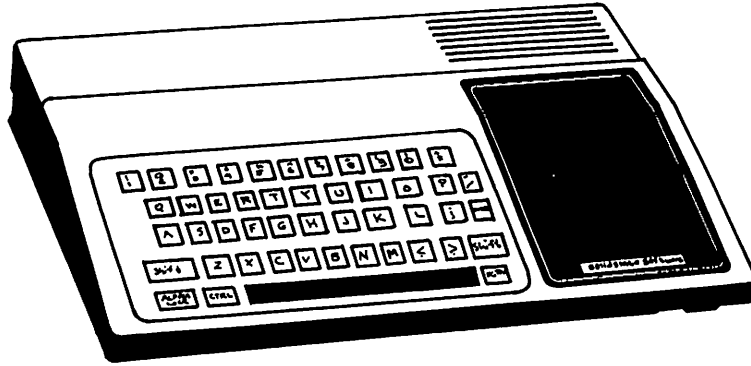


## Color Codes and Set Numbers

| Set # | Character Codes | Set # | Character Codes | Set # | Character Codes |
|-------|-----------------|-------|-----------------|-------|-----------------|
| 1     | 32-39           | 5     | 64-71           | 9     | 96-103          |
| 2     | 40-47           | 6     | 72-79           | 10    | 104-111         |
| 3     | 48-55           | 7     | 80-87           | 11    | 112-119         |
| 4     | 56-63           | 8     | 88-95           | 12    | 120-127         |

Two additional characters are predefined on TI-99/4A computer. The *cursor* is assigned to ASCII code 30, and the *edge character* is assigned to code 31.

## Appendix D



## Musical Note Frequencies

|   |     |          |      |
|---|-----|----------|------|
| C | 130 | Middle C | 523  |
| D | 146 | D        | 587  |
| E | 164 | E        | 659  |
| F | 174 | F        | 698  |
| G | 196 | G        | 783  |
| A | 220 | A        | 880  |
| B | 246 | B        | 987  |
| C | 261 | C        | 1046 |
| D | 293 | D        | 1174 |
| E | 329 | E        | 1318 |
| F | 349 | F        | 1396 |
| G | 392 | G        | 1568 |
| A | 440 | A        | 1760 |
| B | 493 | B        | 1975 |

# Index

## A

ABS function, 38, 191  
 Absolute value, 38  
 Accept statement, 152  
 Addressing, 107  
 Adventure games software, 186  
 Alpha Lock key, 16  
 Animation, 85  
 Arc tangent function, 38  
 Argument, bad, 100  
 Arithmetic and logic unit, 2, 104  
 Arrays, reserving space for, 41, 193  
 Array subscript, setting limits of, 45  
 Arrow key, 15  
 ASC function, 38, 191  
 ASCII code, 38, 39, 75, 191, 193, 210  
 Assembly language, 152, 153  
 Asterisk (\*) symbol, 17  
 ATN function, 38, 191  
 Auto command, 195

## B

Bad Argument error message, 100  
 Bad Line number error message, 98, 100  
 Bad Name error message, 98, 100  
 Bad Subscript error message, 100  
 Bad Value error message, 99, 100  
 BASIC, 8, 37, 159  
 BASIC interpreter, 12, 37  
 Beep, 98  
 Beep statement, 193  
*Beginner's BASIC*, 73  
 Boolean operators, 194  
 Branch, conditional, 42, 55, 66  
 Branching, 42, 45  
 Break command, 38, 192  
 Break key, 14  
 Bus, 2, 103  
 BYE command, 39, 192

## C

Call CHAR subprogram, 39, 79, 192  
 Call Clear subprogram, 38, 41, 192  
 Call Color subprogram, 39, 83, 192  
 Call GCHAR subprogram, 39  
 Call HCHAR subprogram, 40, 76, 85, 192  
 Call INIT subprogram, 153  
 Call JOYST subprogram, 40, 192  
 Call Key subprogram, 40, 89, 192  
 Call Link subprogram, 153  
 Call Load subprogram, 153  
 Call Screen subprogram, 40, 84, 192

Call Sound subprogram, 40, 89, 147-149, 193  
 Call statement, 75  
 Call VCHAR subprogram, 40, 79, 85, 192  
 Can't Do That error message, 99, 100  
 Carat (ˆ) symbol, 17  
 Card games software, 179  
 Carriage returns, 22  
 Cartridge storage cabinet, 29  
 Cassette interface cable, 25  
 Cassette program recorder, 26  
 Cassette tape, 9  
 CC-40, 34  
 CHAR, 38, 81  
 Characters, repeating screen, 40, 192  
 Character set, 8, 79  
 Check functions, built-in, 97  
 CHR\$ function, 38, 193  
 Clearing the screen, 50  
 Clock, 2  
 Close statement, 38, 193, 194  
 CLS command, 192  
 Colon, 194  
 Color, 39, 83  
 Color, on-screen, 75  
 Color codes, 212  
 Color monitor, 19  
 Color statement, 192  
 Colors, specifying screen, 39  
 Column, screen, 77  
 Comma, 17, 64, 119  
 Comments, 152  
 Communications adaptor, 22  
 Compact Computer, 34, 38  
 Computer enhancement programs, 158  
 Computer users, first-time, 49  
 Computers, digital, 4  
 Computers, stored program, 4  
 Computers compatible with the TI-99/4A, 32  
 Computing devices, electromechanical, 4  
 CON command, 193  
 Console, 12  
 Console control during program run, 91  
 Constants, 41  
 CONT command, 193  
 Continue command, 40, 193  
 Control circuits, 2  
 Control key, 8  
 COS function, 40, 193

Cosine function, 40  
 CTRL, 8

## D

Data, 1  
 Data, reading, 45  
 Data bus, 103  
 Data error message, 101  
 Data printed horizontally, 54  
 Data statement, 41  
 Debugging, vii, 98, 119  
 Decimal-adjust instruction, 112  
 Default print state, 54  
 DEF FN statement, 193  
 DEF statement, 41, 193  
 Delete command, 41, 193  
 Delete function, 16  
 Deleting stored programs, 41, 193  
 Demonstration programs, 159  
 Diagnostics program, 158  
 Dice game program, 68  
 DIM command, 41, 193  
 Direct addressing, 109  
 Disk drive, 9, 23  
 Disk memory system, 23  
 Displaying information on the screen, 41, 45, 193  
 Display statement, 41, 92, 152  
 Division, 60, 113  
 Documentation, TI-99/4A, vii  
 Dollar sign, 62

## E

Echo, 22  
 Edit command, 41, 50, 193  
 Editing, vii, 41, 193  
 Edit mode, 16  
 Educational software, 164, 170, 178  
 End-of-file function, 41, 193  
 End statement, 41, 55, 193  
 Engineering software, 160  
 EOF function, 41, 193  
 Equals (=) sign, 17  
 Erasing programs, 16, 52  
 Error handling, built-in, 152  
 Error messages, vii, 50, 97  
 Errors, program, 119  
 Exiting BASIC, 39  
 Expansion options, vii, 9, 17  
 EXP function, 42, 194  
 Exponential function, 42, 194  
 Extended BASIC, viii, 8, 151, 159

**F**  
 FCTN, 8  
 File, closing a, 39  
 File error message, 101  
 Files, retrieving, 46, 195  
 Financial program, 122  
 For-Next command, 53  
 For-Next error, 56  
 For-Next error message, 99  
 For-Next loops, 8, 55  
 For-To-Step statements, 42, 194  
 Function key, 8, 12, 14, 15  
 Function keys, special, 15  
 Functions, 68  
 Functions, defining, 41  
 Functions, special, 7

**G**  
 Geometry, 155  
 GOSUB statement, 42, 66, 194  
 GOTO statement, 42, 51, 67, 194  
 Grammar, poor, 98  
 Graphic images, controlling, 90  
 Graphics, 7, 19, 39, 158  
 Graphics, on-screen, 40, 192  
 Graphics, Turtle, 155  
 Graphics language interpreter, 12  
 Graphics programming, 152  
 Greater than (>) symbol, 17

**H**  
 Halt key, 16  
 HCHAR, 40, 77, 85  
 Hexadecimal code, 81  
 Home computer, vii, 10

**I**  
 If-Then-Else statements, 42, 66, 152, 194, 196  
 If-Then statement, 55  
 Impact printer, 29  
 Incorrect Statement error message, 99, 101  
 Indexed addressing, 108  
 Input error message, 101  
 Input/output (I/O) functions, 3  
 Input statement, 45, 58, 80, 194  
 Input statement, eliminating the, 40  
 Insert mode, 16  
 Instruction execution, 104  
 Instructions, 1  
 Instructions, microprocessor, 107  
 Integer function, 43  
 Integers, 68  
 Interrupt mask, 104  
 Interrupt system, 104  
 INT function, 43, 68  
 I/O error message, 101

**J**  
 Joysticks, 21, 40, 192

**K**  
 Key, 42, 90  
 Keyboard, 7, 12, 16, 149  
 Keyboard overlay strip, 15  
 Key On statement, 192  
 Keypad, numeric, 11  
 Kill command, 193

**L**  
 Language for children, computer, 154  
 Languages, programming, 151  
 Left\$ function, 196  
 LEN function, 43, 65, 194  
 Length function, 43  
 Less than (<) symbol, 17  
 Let statement, 44, 59, 194  
 List command, 8, 44, 51, 194  
 Linefeed, 22  
 Line number, bad, 98, 100  
 Line numbers, 46, 195  
 Line Too Long error message, 99  
 Locate statement, 192  
 Logarithm function, 44  
 LOG function, 46, 195  
 Logic software, games of, 182  
 Logo, 167  
 Logo language, 154  
 Loop, 52  
 Loops, continuous, 51  
 Loops, creating, 42, 194  
 Loops, for-next, 52, 55, 87  
 Loops, getting in and out of, 57  
 Lowercase letters, 14

**M**  
 Machine language, 103  
 Makecharacter command, 155  
 Makeshape command, 155  
 Mathematical computations, 65  
 Mathematical functions, 62  
 Math keys, 17  
 Memory, 3, 9  
 Memory, erasing a program in, 44  
 Memory address register, 107  
 Memory expansion card, 23  
 Memory Expansion Unit, 9  
 Memory Full error message, 99, 101  
 Memory module, plug-in, 10  
 Merge statement, 152  
 Microcomputer, single-chip, 2  
 Microcomputer history, 4-6  
 Microprocessor, 1, 2, 103, 152  
*Microprocessor Cookbook*, 103  
 Microprocessor instructions, 107  
 MID\$, 196

Minus (-) symbol, 17

Misspelled words, 98  
 Modem, 22, 26  
 Modulator, video, 18  
 Monitor, 9, 19  
 Multiplication, double-precision, 113  
 Multiply instruction, 112  
 Musical note frequencies, 213  
 Musical notes, 147, 148

**N**  
 Name, bad, 98, 100  
 New command, 44, 52, 192, 195  
 Next statement, 44, 195  
 Noise, 89  
 Noises, generating, 40, 193  
 NUB command, 44  
 Nulls, 22  
 Number command, 44, 195  
 Numbering program lines, 48  
 Number keys, 14  
 Numbers command, 185  
 Number Too Big error message, 102  
 NUM command, 195  
 NUMS command, 195

**O**  
 Old command, 46, 195  
 ON-GOSUB statement, 45, 195  
 ON-GOTO statement, 45, 195  
 On Key statement, 192  
 Op code, 109  
 Open statement, 45, 194, 195  
 Option Base statement, 45

**P**  
 Parity, 23  
 Pascal, 153, 160  
 Period (.), 17  
 Peripheral expansion system, 23  
 Personal computer, 10  
 Pilot, 160  
 Play statement, 193  
 Plus (+) symbol, 17  
 Pointer register, 104  
 POS function, 45  
 Power cord, 12  
 Power switch, 12  
 Printer, impact, 29  
 Printer, thermal, 19  
 Printer programmable functions, 21  
 Print statement, 45, 50, 54, 192, 193, 195  
 Program, erasing, 52  
 Program, terminating a, 47  
 Program counter, 104  
 Program errors, 119

Program execution, beginning, 46  
 Program execution, halting, 44, 57  
 Program execution, resuming, 40, 193  
 Program lines, determining the order of execution of, 47  
 Program lines, numbering, 48  
 Programming aids, 157  
 Programming the TI, 12  
 Program portions, accessing, 45  
 Program run errors, 100  
 Program sequence, changing, 42  
 Program termination, 41, 193  
 Programs, complex, 49  
 Programs, erasing, 17  
 Programs, stored, 44  
 Prompts, 50, 58  
 Punctuation keys, 15

## Q

Question mark prompt, 58  
 Quotation marks, 54, 62, 119, 126, 194

## R

RAM, 3, 10  
 Random function, 46  
 Randomize statement, 45, 121, 195  
 Random number generator, 45, 70, 121, 126  
 Random numbers, 68  
 Read statement, 46, 195  
 Register indirect addressing, 107  
 Relative addressing, 109  
 Remark, 46, 195  
 Remote controllers, 21  
 REM statement, 46, 195  
 RENUM command, 195  
 Repeat function, 7, 17  
 RES command, 46, 195  
 Resequencing command, 46, 50, 195  
 Reserved words in TI BASIC, 209  
 Restarts, 111  
 Restore statement, 46, 195  
 Return statement, 46, 68, 195  
 Right\$ function, 196  
 RND function, 46, 68, 195  
 ROM, 3, 10  
 Row, screen, 77  
 RS 232 Interface, 22  
 Run command, 46, 50, 195

## S

Save command, 46, 196  
 Save command, expanded, 152

Screen, 40, 85  
 Screen, changing colors of the, 83  
 Screen, clearing the, 39, 50  
 Screen, placing a character on the, 40, 192  
 Screen, reading characters on the, 38  
 Screen coordinates, 75  
 Screen display format, TI, 11  
 Screen statement, 192  
 Scrolling, 10, 51  
 SEG\$ function, 46, 196  
 Semicolon (;), 54, 119  
 SGN function, 46, 196  
 Shift key, 17  
 Shooting Gallery program, 92  
 Signum function, 46, 196  
 Sine function, 47, 196  
 SIN function, 47, 196  
 Size command, 152  
 Slash (/) symbol, 17  
 Software for inexperienced programmers, 158  
 Software interrupts, 111  
 Software stack, 111  
 Solid State Command Module, 9  
 Sound, 78, 89  
 Sound, generating, 40, 193  
 Sound, specifying duration, frequency, and volume of, 89  
 Sound generation, 147, 148  
 Sound statement, 193  
 Speech, 7  
 Speech Editor Command Module, 27  
 Speech modules, 29  
 Speech synthesizer, 27  
 Sprite command, 155  
 Sprites, 152  
 SQR function, 47, 196  
 Square root function, 47, 196  
 Stack register, 110  
 Statement, incorrect, 99, 107  
 Statement lines, multiple, 151  
 Stick Function, 192  
 Stop command, 192  
 Stop statement, 47, 196  
 Storing programs in memory, 46  
 Strategy and logic software, 182  
 STR\$ function, 47, 196  
 String, 47, 196  
 String, finding the number of characters in a, 43  
 String, getting a portion of a, 46  
 String-number Mismatch error message, 102

Strings, 62  
 Strings, detecting strings within, 45  
 String variable, extracting numeric value from, 47  
 String variables, 66  
 Subprograms, 38, 40, 75, 93, 153  
 Subroutine, returning from a, 46  
 Subroutines, 68, 111  
 Subscript, bad, 100  
 Symbolic addressing, 109  
 Symbol keys, 15  
 Symbol Table error messages, 99

## T

TAB function, 47, 196  
 TAN function, 47, 196  
 Tangent function, 47, 196  
 Telephone coupler, 22, 25  
 Thermal printer, 19  
 TI BASIC, converting to, 191  
 TI BASIC reserved words, 209  
 Tile command, 155  
 TI logo, 154, 167  
 Timing, circuits, 2  
 TI-99/4A, cost of, 11  
 TI-99/4A computer, the basic, vii  
 TI-99/4A microcomputer, 7  
 TI-99/4A UCSD PASCAL development system, 153  
 TI-99/2 computer, 34  
 TMS9900 microprocessor, 103, 152  
 Trace command, 47  
 Typing accuracy, testing, 66  
 Typing program lines, 119

## U

Unbreak command, 192  
 Uppercase mode, 14  
*User's Reference Guide*, 74

## V

VAL function, 47, 65, 196  
 Value, bad, 99, 100  
 Variables, 61  
 Variables, assigning values to, 44  
 Variables, naming, 64  
 Variables, rules on the use of, 64  
 Variables, string, 66  
 VCHAR, 40, 79, 85  
 Video modulator, 18

## W

Wait command, 192  
 Work-space system, 104



# Using and Programming the TI-99/4A, including Ready-to-Run Programs

by Frederick Holtz

*"This is the best book about the TI-99/4A I've seen. It will be a difficult task to top this one!",* writes Charles LaFara, President of the International 99/4A User's-Group in his review of this exceptional programming handbook. He goes on to say:

"The book answers the questions that arise during a user's first encounters with the TI-99/4A. And even long-time members of our Group will benefit from the information on BASIC programming and TI's LOGO language. The chapter on converting programs from other BASIC dialects to TI BASIC and the glossary of microcomputer terms are practical references for users of all levels of experience."

"The first half of the book makes the first-time user feel comfortable with his new microcomputer by describing the hardware, including the peripherals available for the TI-99/4A. The second half is devoted to programming the computer. I applaud the author for his recognizing the tremendous potential of this machine by seeing beyond its capability to run packaged software."

"Now, thanks to Frederick Holtz, tens of thousands of first-time users will know what we in the Group have known for a long time—that the TI-99/4A is a great microcomputer."

Thousands of other TI-99/4A users have been equally enthusiastic about this outstanding guide to using and programming this versatile and amazingly powerful micro. Of special interest are the many ready-to-run programs and a listing of more than 200 packaged programs that are commercially available for use on the TI-99/4A. If the TI-99/4A is your choice in a home and family computer, this book is your key to getting the most use and enjoyment from your machine!

---

Also available from TAB is TI-99/4A Game Programs (No. 1630—\$10.95 paper; \$17.95 hard)

---

**TAB TAB BOOKS Inc.**

Blue Ridge Summit, Pa. 17214

---

Send for FREE TAB Catalog describing over 750 current titles in print.

FPT > \$10.25

ISBN 0-8306-1620-9

PRICES HIGHER IN CANADA

995-0483