SAMS 22258

VIC 20™ Starter Book

Jonathan A. Titus
Christopher A. Titus
David G. Larsen



BLACKSBURG CONTINUING EDUCATION SERIES™
edited by Titus, Titus & Larsen

The Blacksburg Continuing Education™ Series

The Blacksburg Continuing Education Series™ of books provide a Laboratory—or experiment-oriented approach to electronic topics. Present and forthcoming titles in this series include:

- Advanced 6502 Interfacing
- Analog Electronics for Microcomputer Systems
- Analog Instrumentation Fundamentals
- Apple II Applications
- Apple II Assembly Language
- Apple Interfacing
- Basic Business Software
- Basic Robotics Concepts
- BASIC Programmer's Notebook
- Circuit Design Programs for the Apple II
- Circuit Design Programs for the TRS-80
- Computer Assisted Home Energy Management
- Computer Communication Techniques
- Design of Active Filters, With Experiments
- Design of Op-Amp Circuits, With Experiments
- Design of Phase-Locked Loop Circuits, With Experiments
- Design of VMOS Circuits, With Experiments
- 8080/8085 Software Design (2 Volumes)
- 8085A Cookbook
- Electronic Music Circuits
- Electronic Prototype Construction
- Entertainment Games in TI BASIC and Extended BASIC
- Fiber Optics Communications, Experiments, and Projects
- 555 Timer Applications Sourcebook, With Experiments
- FORTH Programming
- Guide to CMOS Basics, Circuits, & Experiments
- How to Program and Interface the 6800
- Introduction to Electronic Speech Synthesis
- Introduction to FORTH
- Microcomputer—Analog Converter Software and Hardware Interfacing
- Microcomputer Data-Base Management
- Microcomputer Design and Maintenance
- Microcomputer Interfacing With the 8255 PPI Chip
- NCR Basic Electronics Course, With Experiments
- NCR EDP Concepts Course
- Numerical BASIC
- PET Interfacing
- Programming and Interfacing the 6502, With Experiments
- Real Time Control With the TRS-80
- 16-Bit Microprocessors
- 6502 Software Design
- 6801, 68701, and 6803 Microcomputer Programming and Interfacing
- The 68000: Principles and Programming
- 6809 Microcomputer Programming & Interfacing, With Experiments
- STD Bus Interfacing
- TEA: An 8080/8085 Co-Resident Editor/Assembler
- TRS-80 Assembly Language Made Simple
- TRS-80 Color Computer Interfacing, With Experiments
- TRS-80 Interfacing (2 Volumes)
- TRS-80 More Than BASIC
- VIC 20 Programmer's Notebook
- Wordprocessing for Small Businesses

In most cases, these books provide both text material and experiments, which permit one to demonstrate and explore the concepts that are covered in the book. These books remain among the very few that provide step-by-step instructions concerning how to learn basic electronic concepts, wire actual circuits, test microcomputer interfaces, and program computers based on popular microprocessor chips. We have found that the books are very useful to the electronic novice who desires to join the "electronics revolution," with minimum time and effort.

Jonathan A. Titus, Christopher A. Titus, and David G. Larsen
"The Blacksburg Group"

Bug symbol trademark The Blacksburg Group, Inc., Blacksburg, VA 24060

VIC 20 Starter Book



Dr. Jonathan A. Titus is with the Blacksburg Group, Blacksburg, VA. Most of his current work involves writing about small computers, particularly their hardware side. He has written a number of articles about computers for both professional and popular publications, and got his start in microcomputers with the Intel 8008 in the early 1970s.

Jon is a radio amateur, holding an Advanced Class license (KA4QVK), and operates both phone and CW on the higher bands. He is also interested in astronomy, and looks forward to clear, calm evenings for good observing.

Jon received his B.S. degree from Worcester Polytechnic Institute in 1967, his M.S. degree from Rensselaer Polytechnic Institute in 1969, and his Ph.D from Virginia Polytechnic Institute and State University in 1978.



David G. Larsen is an instructor in the Department of Chemistry at Virginia Polytechnic Institute and State University, where he teaches electronics and automated instrumentation. He is a coauthor of many of the books in the Blacksburg Series and is a member of the Blacksburg Group. Dave is a coinstructor of a series of one-to-five-day workshops that cover the "microelectronic revolution," which are taught under the auspices of the Extension Division of the University. These programs attract professionals from all parts of the world.

Dave received his B.S. degree from Oregon State University in 1963, and worked for Varian before joining the staff at Virginia Polytechnic Institute and State University.



Dr. Christopher A. Titus currently writes and edits books for the Blacksburg Group, with many of them being in the microcomputer field. Chris has designed and programmed small systems based on the popular 8-bit microprocessors, and has worked with the 8086 and the Z8001/2. His current interests include computer-aided design and microcomputer-based industrial control systems. In his spare time, Chris enjoys camping and gardening.

Chris received his B.S. degree from Clarkson College of Technology in 1972, and his Ph.D from Virginia Polytechnic Institute and State University in 1977.

VIC 20 Starter Book

by

Jonathan A. Titus
Christopher A. Titus
and
David G. Larsen

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

Copyright © 1984 by Jonathan A. Titus, Christopher A. Titus, and David G. Larsen

FIRST EDITION
FIRST PRINTING—1984

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-22258-2
Library of Congress Catalog Card Number: 83-51224

Edited by: *C. W. Moody*
Illustrated by: *T. R. Emrick*

Printed in the United States of America.

Preface

Reading this book and learning about your VIC 20™* computer will be an interesting, rewarding, and fun experience. We've used a common sense approach to computers, and have included many experiments, questions, and problems that you can do on your own. *No computer experience is necessary.* You'll start using your VIC 20 computer right away, exploring the keyboard and display and then going on to other topics. This book is simple to understand, whether you have a specific use for a computer or you just want to know more about what a computer can do. To make computers easy to understand, we've avoided many of the terms and words that "computer experts" seem to use to confuse the rest of us. Where special words seem to fit, we'll tell you about them.

After reading this book and doing most of the experiments, you will be able to:

- Use the keyboard and the display.
- Use 20 simple program commands and know what they do.
- Use the special control keys on the keyboard.
- Know what kinds of information the VIC 20 computer can use.
- Know how labels are used to locate information.
- Use the computer to add, subtract, multiply, and divide values.
- Know how computers make decisions.
- Control the display of information on the TV screen in color and in "reverse."
- Know how "loops" are used in programs to control the computer.

---

*VIC 20 is a trademark of Commodore Electronics Limited.

- Know how the computer uses words and sentences.
- Use the VIC 20's digital clock.
- Use the computer to make special sounds.
- Know how large amounts of information are saved in the computer.
- Know how to test and evaluate programs for your computer.
- Know how to care for and protect your computer.
- Know how to type in and correct programs.
- Know how to add useful remarks and comments to programs.
- Know about some special math operations.

Our purpose is to give you a good understanding of what computers can do, and how they do it, without forcing you to become a computer expert. Since a computer is told what to do by using step-by-step instructions, or a *program*, you will see how a computer is *programmed* to do specific things. You will type in the programs shown in the experiments and you'll get a chance to try a few short programs of your own. Although you will learn about computer programs and some of the instructions they use, you won't be a programming expert when you finish this book. In fact, to use most small computers, you don't have to be a programming expert.

These days, small computers are easy to use, so you don't have to know all the details of exactly what is going on inside the computer box. If you had to be an expert at everything you did, you'd have to be a pump, water, and soap expert just to use a washing machine. Most of us just want to know enough to load the clothes, dump in the soap, and press a button. Computers can be used the same way; load a program and run it.

You'll work with your VIC 20 computer by using its keyboard and a TV set. Programs and information will be typed in on the keys and the computer will display information on the TV screen. The VIC 20 displays things in different colors, so a color TV set is recommended, but a black-and-white TV is fine, too. We have used a standard VIC 20 computer, just as it comes from the box, without add-ons or accessories. This is often called a "5K" VIC 20 computer, and it's all you'll need to use this book.

If all of a sudden the people around you started using a foreign language, it would be very hard to understand them. Your computer has the same problem. It wants you to talk to it in a language called BASIC, and as you'll see, it is fairly easy to understand. Most people learn a language by using simple words and

sentences; "Open the door," or "Where is the school?" You'll learn about the BASIC language the same way; using simple commands to do simple things.

Besides reading the text and doing the experiments, we want you to explore the computer on your own, too. There are problems at the end of most chapters, and if you are interested in programming, or learning more about what the computer can do, you can try and solve them. Solutions to selected problems are provided in Appendix D. You won't bother the computer by pressing the wrong keys or by programming it incorrectly. You may get some interesting sounds and displays, and things may not come out the way you expected, but YOU CAN'T HURT THE COMPUTER unless you get mad at it and give it a kick.

Here's a quick overview of what is included in this book. In *Chapter 1* you'll turn on your computer and learn about the keyboard and the display. Since these two parts of the computer let it "communicate" with you, they are very important. You will see how you can put letters, numbers, and special symbols on the TV screen, and how the color of the display can be changed.

Once you have used the keyboard and the display, you are ready to learn what a computer really does. They are simpler than you think, as you'll find out in *Chapter 2*. You will learn about a couple of BASIC instructions that you can use to get information into and out of the computer, and you'll find out about the two types of information that the VIC 20 can use

Simple arithmetic is very important to a computer, and *Chapter 3* shows you the four basic math operations of addition, subtraction, multiplication, and division, and how to get the computer to easily do them. Since everyone makes mistakes, several of the "correction" keys will be described, too.

Chapter 4 gives you a good understanding of how programs can control what the computer does. You'll find out how the computer can make simple decisions, and how decisions tell the computer what to do next. Although we think of computers as doing very complicated things, they can only make very simple decisions. You will also learn more about numbers, how the computer uses them, and how it can "dream up" a number for you, much like a throw of dice.

More "computer power" is described in *Chapter 5*. You'll see how the computer can be told to do something a specific number of times. Computers can store large quantities of information, and you'll find out how to do it in this chapter.

Computers can also work with names, addresses, and other "words." *Chapter 6* shows you how this type of information can be put into the computer and used in different ways. The VIC 20 has its own digital clock, and you will learn how to use it to keep track of time and to turn on an "alarm."

Some special math operations, such as logarithms and trig operations, are described in *Chapter 7*. We've provided some interesting history, so this isn't a dull, dry chapter. However, if you aren't interested in these operations and don't have a use for them, you can skip ahead to *Chapter 8* and learn how to control the TV screen and the information that it displays. Moving the cursor, changing the color, printing in reverse, and doing other interesting things with the TV display are all detailed for you.

The VIC 20 can also create interesting sounds that can be used in many types of programs. You will learn how to control these sounds in *Chapter 9*, and many experiments give you a chance to experiment with the VIC 20's sound effects.

The first nine chapters give you a detailed introduction to your VIC 20 computer, how it works, and what you can do with it. Most of the things you learn about BASIC programs can be used on other computers, and there are many books that will take you further into learning BASIC so that you can become a good programmer, if you want to. The last two chapters in this book discuss some of the accessories for your VIC 20 and what you can do with them, and how to take care of your computer, both hardware and software.

In *Chapter 10*, we'll tell you about the cassette unit, the VIC-1525 graphic printer, the joystick, and plug-in program and memory cartridges. In *Chapter 11*, you will learn how to care for your computer, from simple inexpensive steps you can take, to how to get service for the computer should it ever fail to operate correctly. We will also tell you about computer programs, or "software," how to review it, what to look for, what to avoid, and so on.

We want to thank Mr. Rik Andrews of Holdrens, Inc., Roanoke, Virginia, for loaning us a VIC-1525 printer that was used in preparing Chapter 10. Rik's help was invaluable. We also want to thank Mr. Bob Veltri, Blacksburg, Virginia, for working with us and doing most of the photography work included in this book. We also appreciate the continued support of our many friends at Howard W. Sams & Co., Inc.

Although we have mentioned some specific equipment, books,

magazines, and suppliers in this book, these are not necessarily endorsements of them. Check carefully to be sure that your purchases do what you want them to and that you get your money's worth.

If you have any questions or comments about this book, please write to us directly at:

The Blacksburg Group, Inc.
P.O. Box 242
Blacksburg, VA 24060 USA

JONATHAN A. TITUS
CHRISTOPHER A. TITUS
DAVID G. LARSEN

This book is dedicated to our families, both near and far.

Contents

Chapter 1. **Getting to Know Your VIC 20** 17

Getting Started 17
The Keyboard 20
Printing Keys 21
Experiment No. 1-1. Simple Typing 22
Action Keys 23
Experiment No. 1-2. The SHIFT Key 24
Experiment No. 1-3. Going Home and Clearing Out 26
Experiment No. 1-4. Moving the Cursor 27
Your Eraser 28
Experiment No. 1-5. The INST/DEL Key 29
Experiment No. 1-6. Deleting and Inserting 30
Drawing Pictures 32
Experiment No. 1-7. Drawing Things 34
Color Fun 35
Experiment No. 1-8. The Color Connection 36
Experiment No. 1-9. Printing in Reverse 37
Secrets 38
Questions 39
Food for Thought 40

Chapter 2. **Using a Small Computer** 41

What Can Your Computer Do? 41
Simple Programs 43

Getting Started 44
Experiment No. 2-1. A Simple Input/Output Program 45
Experiment No. 2-2. Running Your Program 47
Experiment No. 2-3. Bad Data 49
Labels, Labels, Everywhere 50
VIC 20 Labels 52
Experiment No. 2-4. Short Labels 53
The Computer Information Diet 54
Types of Information and Labels 55
Experiment No. 2-5. Inputting Strings 57
Experiment No. 2-6. Mixed Labels 58
Intelligent Questions and Answers 59
Experiment No. 2-7. Printing Messages 62
Conclusion 63
Questions 63
Food for Thought 65

Chapter 3. **Number Crunching** 67

Simple Arithmetic 67
Setting Up the Problem 68
A NEW Operation 69
Experiment No. 3-1. A Simple Math Program 69
Experiment No. 3-2. More Math Fun 70
Using Labels in Math Problems 72
Getting Your Values—the INPUT Command 73
Experiment No. 3-3. Your Calculator 75
One Line for Many 77
Experiment No. 3-4. More On a Line 77
Doing More Than One Thing at a Time 78
Using Parentheses in Math Problems 80
What's Going on Here? 82
Your Efficient Secretary 83
Experiment No. 3-5. The INST/DEL Key 83
Questions 86
Problems 87

Chapter 4. **Making Decisions** 89

The GOTO Command 89
Breaking Out 94
Experiment No. 4-1. Loops and Breaks 95
Real Decisions 98
The IF-THEN Operation 100
Experiment No. 4-2. The IF-THEN Operation 102
Experiment No. 4-3. Counting Loops 106

To Be AND/OR NOT to Be 108
Mixed Operations 109
The NOT Operation 111
A Little History 111
Pick a Number, Any Number 112
Experiment No. 4-4. Random Value Program 1 112
Experiment No. 4-5. Random Value Program 2 114
A Word of Caution 115
Questions 116
Food for Thought 118
Problems 118

Chapter 5. **More Power to You** 122

Specialty Printing 122
Experiment No. 5-1. Clearing the Screen 123
Typing Special Symbols 124
Playing for Position 125
Experiment No. 5-2. Printing in Columns and Rows 126
The TAB Command 126
Experiment No. 5-3. Using the TAB Command 127
More Decisions 128
Counting Loops—the FOR-NEXT Command 131
Experiment No. 5-4. The FOR-NEXT Loop 132
A Real Use for a Loop 135
Nested Loops 136
Take One Giant Step 140
Subroutines to the Rescue 142
Experiment No. 5-5. Using a Subroutine 144
Lots of Information 147
Experiment No. 5-6. Using an Array or Table 150
The Size of Your Arrays 151
A Special Kind of Array 152
Experiment No. 5-7. Using the DATA and READ Commands 153
Questions 155
Problems 157

Chapter 6. **Strings and Things** 160

Using Strings 161
Experiment No. 6-1. Building a String 164
Experiment No. 6-2. Peculiar Characters 166
String Decisions 167
Untangling Strings 169
The Length of a String 169

The LEFT$ and RIGHT$ Operations 170
Experiment No. 6-3. A Word-Guessing Program 171
In the Middle of a String 173
The GET Command 174
Experiment No. 6-4. Using the GET Command 175
Experiment No. 6-5. Building Your Own String 176
Experiment No. 6-6. Getting a Few Characters 177
Telling Time 178
Experiment No. 6-7. Using the VIC 20's Clock 179
Strings and Values, Values and Strings 180
Questions 182
Problems 184

Chapter 7. **Special Math Functions** 187

The Square Root 187
The Value of Pi 190
Trigonometry 193
VIC 20 Trigonometry 197
Using Trig Functions 198
Logarithms 203
VIC 20 Logs 205
Special Codes 206
The Tan "f" Keys 208
References 209
Problems 209

Chapter 8. **Lights, Action** 212

The Space Command 212
Moving from Place to Place 213
Experiment No. 8-1. Moving the Cursor 214
The Magic Color Screen 217
Experiment No. 8-2. Color Control 219
Experiment No. 8-3. Reversed Printing 221
Advanced Displays and Graphics 222
Experiment No. 8-4. Filling the TV Screen 223
Experiment No. 8-5. Using the POKE Command 230
Reversed Characters 232
Experiment No. 8-6. A Flashing Sign 232
Seeing What You Have Done 235
More Characters 236
Screens and Borders 236
A Note About Codes 237
Questions 238
Problems 239

Chapter 9. **Sound Off, 1, 2, 3, 4** 243

Sound Is Important 243
Sound from the VIC 20 244
Experiment No. 9-1. Setting the Volume 245
Experiment No. 9-2. Changing the Tone 249
Experiment No. 9-3. Changing the Speakers 251
Using Sound in a Program 252
Noise 255
Experiment No. 9-4. Making Noise 255
Questions 257
Problems 258

Chapter 10. **Accessories for the VIC 20** 259

Program and Memory Cartridges 259
The Cassette Unit 261
Saving Information on a Cassette 263
Experiment No. 10-1. Saving Information 265
Experiment No. 10-2. Getting Data from the Cassette 267
Experiment No. 10-3. Saving Strings on a Tape 269
The Cassette and the GET Command 271
Experiment No. 10-4. Using the GET Command with the Recorder 273
The Graphic Printer 274
Starting the Printer 275
The Joystick Controls 279
Suppliers and Distributors of Cartridges and Cassettes 282

Chapter 11. **Computer Care—Hardware and Software** 283

Hardware Care 283
A Special Place 284
Static 285
Protection 285
Power 285
Clean Up 288
Troubleshooting 289
Software 291
What's Available 291
Information and References 291
Demonstrations and Documentations 292
Testing Programs 293
Updates and Revisions 294
Flexibility 295
Protection 295

Appendix A. **VIC 20 Error Codes** 297
Appendix B. **VIC 20 BASIC Keywords** 300
Appendix C. **Answers to Questions** 301
Appendix D. **Solutions to Selected Problems** 311
References 339
Index 341

1

Getting to Know Your VIC 20

Before you can get your VIC 20 to do useful and helpful things, it's important for you to be familiar with the keyboard and the display. To get started, just sit down in front of your computer and continue reading. If you don't have your computer handy, you can still read through the chapters, using the illustrations as guides. We hope that you will go back later and do the experiments.

GETTING STARTED

The first thing you need to do is to check the various connections to the computer, so it will be ready when you turn it on. The basic computer system is shown in Fig. 1-1. Let's look at the power module first. It's the large, heavy plastic box that has two wires coming out of it, and it's shown in Fig. 1-2. One of the wires has a standard two-prong plug on the end, and it is plugged into a wall or floor outlet to get power. The other wire is somewhat thinner, and it is connected to the small, two-prong connector marked 9V AC, on the right side of the keyboard case. The power connections are shown in Fig. 1-3.

A cable assembly connects the computer and your television, or TV, as shown in Fig. 1-4. The VIC 20 works with either a color or black-and-white TV set, and the connections are the same for both.

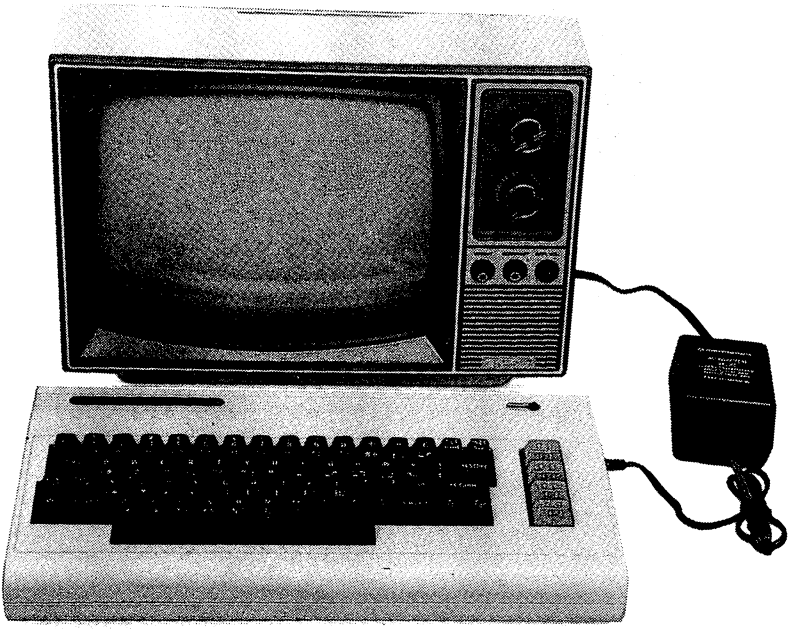


Fig. 1-1. The basic VIC 20 computer system—computer, power module, and TV set.



Fig. 1-2. The VIC 20 power module.

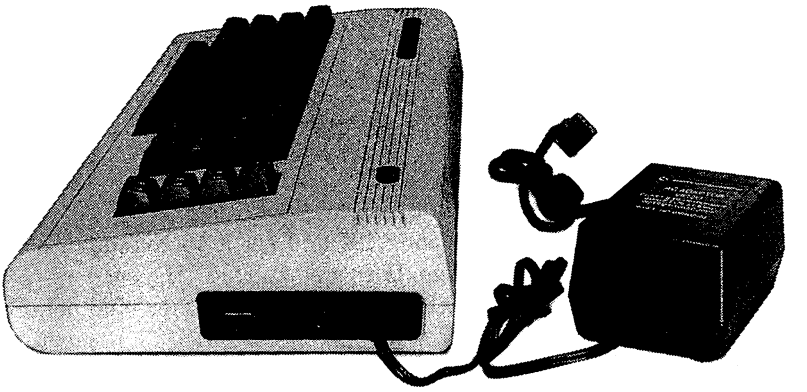


Fig. 1-3. The power module connected to the VIC 20.

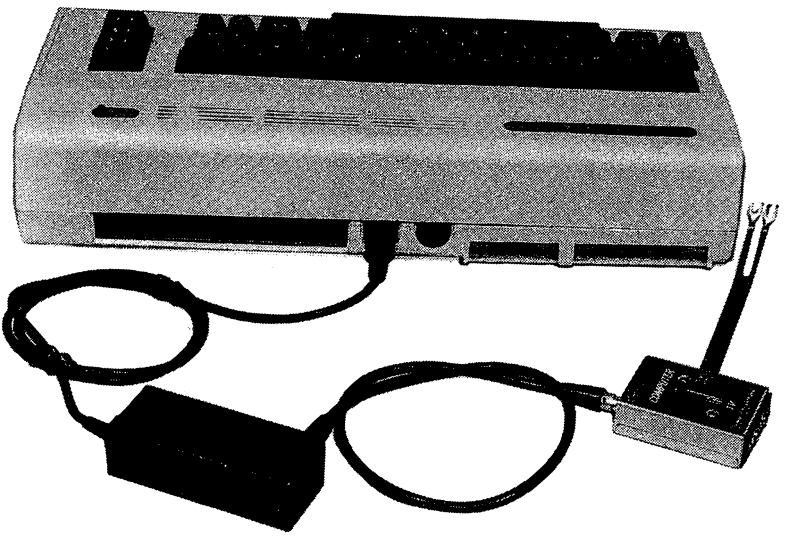


Fig. 1-4. The cable assembly for connecting the VIC 20 to a TV set.

The computer is connected to the VHF antenna screws on the back or side of the TV case through the TV/COMPUTER switch box supplied with the computer. When using the computer, be sure that this switch is in the COMPUTER position. When you are finished using the computer, switch it back to the TV position so that you can use the TV to watch a regular show.

There is a small box in the cable assembly, between the TV and the computer. This box contains a small switch marked CH3-CH4,

and it lets you switch the computer display to either channel 3 or channel 4 on the TV set. You can try either one to see which channel setting gives the best display.

Now it's time to turn on the computer. The computer's ON/OFF switch is on the right side of the keyboard, next to the power wire, as seen in Fig. 1-5. Push the switch to the ON position. The red light marked POWER on top of the keyboard should be lit when the computer is on. Turn on your TV and set it to channel 3 or 4. Give the computer a few seconds to get started. If you don't see anything on one channel, switch the TV to the other channel. Be sure the TV/COMPUTER switch is in the COMPUTER position. In case of difficulty, or for more information about the cables and connections, refer to the computer guide supplied with your computer.

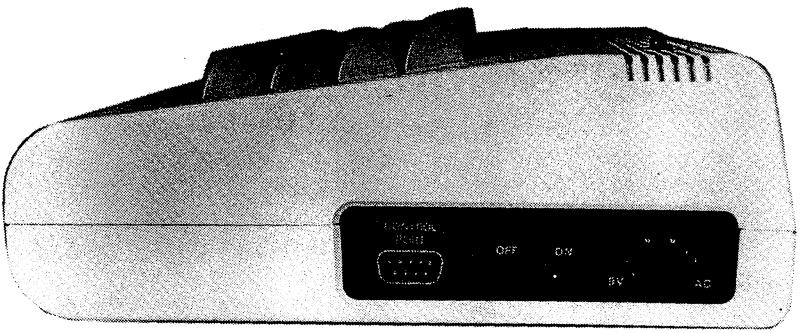


Fig. 1-5. Location of the ON/OFF power switch.

When the computer is first turned on, you'll see a large, white square area with a green border around it that goes to the edge of the TV screen. On a black-and-white TV, you'll just see that the large, central area is lighter than the border. The computer will display a message at the top of the square area, including the word READY. You will also see a small, flashing square, or *cursor*, on the screen, which is a "place marker," telling you where the computer is. We'll just show the cursor as a black square ■.

THE KEYBOARD

Now that your computer has been turned on, let's take a close look at the keyboard, since it's the most important and most-used part of the VIC 20 computer. You may have used some of the keys

when playing games or when using the computer for other things, but you may not know exactly what the keys do. Even if you have used a typewriter or other computer, all of the letters, numbers, words, and symbols can be confusing.

There are two types of keys on the VIC 20; *action keys* and *printing keys*, as shown in Fig. 1-6. The 50 *printing keys* are in the middle of the keyboard, and there are five *action keys* on the left and seven *action keys* on the right of the keyboard. (We will look at the four tan "f" keys later on in the book.) The printing keys are used to really "print" something on the TV screen, while the action keys tell the computer to do something, such as GO TO CAPITAL LETTERS, LIKE THIS. Of course, the computer won't print things on the screen; you need paper and ink to do that. It just *displays* them for you on the TV, but "print" is a handy word, so we'll use it.

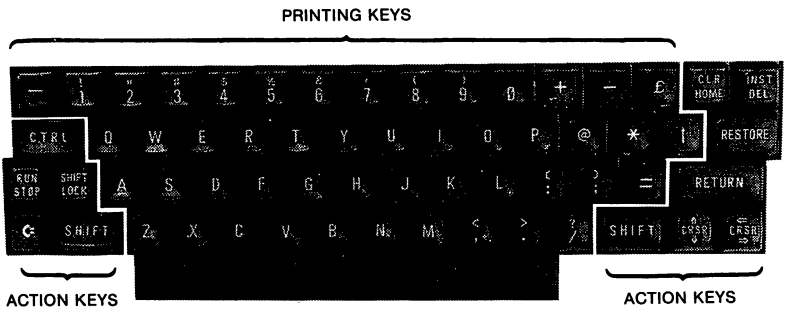


Fig. 1-6. The VIC 20 keyboard showing the *action* and *printing* keys.

PRINTING KEYS

First, let's look at the tops of the *printing keys* in the center of the keyboard, as shown in Fig. 1-7. We'll look at the special symbols on the front side of these keys later.

If you haven't used a keyboard like this before, the first thing you will see is that the letters are not in alphabetic order. You think of the alphabet as ABCDEFG . . . , but the letters on the VIC 20 keyboard are scrambled. It may take a while to get used to, but this type of standard keyboard is used on almost all computers.

The 26 letters, A to Z, are located in the bottom three rows, and toward the center of the keyboard, as shown in Fig. 1-7. You'll see

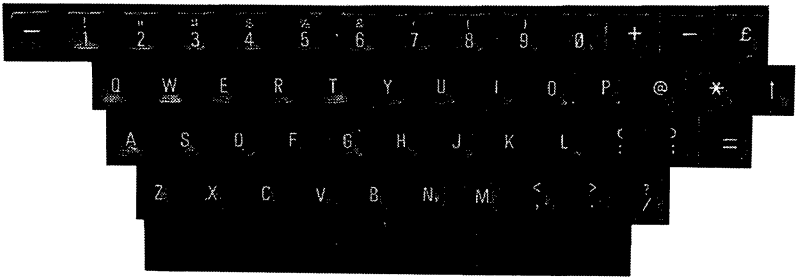


Fig. 1-7. The VIC 20 printing keys.

that the capital, or upper-case letters, are the only ones shown on these key tops.

The top row of keys is shown in Fig. 1-8, and the keys are in order, "1" through "9," with the "0" key just to the right of the "9" key. The key tops for keys "1" through "9" have other symbols on them, too. For example, the "4" key also has a dollar sign ($) on it. We'll talk more about this shortly.

Besides the letter and number keys, there are other *printing keys*, such as those for the asterisk (*), comma (,), semicolon (;), question mark (?), and period (.). Some of these keys also have two symbols, one above the other on the key tops.



Fig. 1-8. The 10 number keys.

Experiment No. 1-1. Simple Typing.

Now you will have a chance to use the computer and type some numbers and letters on the TV screen. If your computer is turned off, turn it on. Once the computer displays the word READY on the TV screen, press the "F," "G," "H," and "J" keys one at a time. They are in the center of the keyboard. Don't press any of the "action" keys. What happens as you press the four letter keys? You should see the letters F, G, H, and J displayed on the screen just as they are pressed, or "typed," on the keyboard. Type a few more letters or some numbers until your typing reaches the right side of the display.

What happens when the letters and numbers you are typing

reach the right side of the screen? What happens as you keep typing them? When your typing reaches the right side of the screen, and you keep typing, the cursor moves down and starts a new line at the left side of the screen.

Type more letters and numbers until you are typing on the bottom line of the display. You can use several fingers to quickly do the typing. What happens when the cursor reaches the bottom right-hand corner and you type a few more letters? The display seems to move up, providing "clean" lines for you to use. The computer automatically moves the display up one to four lines. The cursor moves up, too.

The long bar at the bottom of the keyboard is the "space bar." Does it print anything when you press it? It doesn't print anything, but it leaves a space so that your typing doesn't LOOKLIKETHIS. What happens if you hold the space bar down? The computer continues to leave spaces and the cursor moves for as long as you hold the space bar down. Do any of the other "printing" keys work this way? None of the other "printing" keys have this repeating action.

ACTION KEYS

The *action keys* cause the computer to do special things, and some of them change the way in which things are displayed on the TV screen. You've seen that some of the printing keys have two symbols on their tops. For example, the "4" key has a dollar sign ($) above the "4." In Experiment 1-1, you found that you could print the numbers on the TV screen, but you couldn't print the symbols. The SHIFT key is the *action key* that lets you print the "top" symbol on each of the "printing" keys. There are two SHIFT keys, both in the bottom row of keys, with one at each end. You may use either one if a SHIFT action is needed, since they both do the same thing.

The SHIFT key must be pressed down and held down while the other key is pressed to get its "top" symbol displayed on the TV screen. If you don't hold the SHIFT key down, the regular, or "bottom," symbol will be typed. Since the SHIFT key must be pressed and held down when it's used, we'll call this action *press-and hold,* or *press/hold* for short. After a little practice, you will remember when you need to use the SHIFT key.

Whenever an action key is noted in this book, its legend will be in upper-case, or capital letters; for example, the SHIFT key. If

the word "key" is not used, the name of the key will be found in brackets [SHIFT].

Experiment No. 1-2. The SHIFT Key

The SHIFT key is important, since it lets you "shift" between the different characters that can be displayed on the TV screen. In this experiment, you will see how it can be used.

Once your computer has been turned on, press the SHIFT key. Can you see anything happening on the TV screen? No, the SHIFT key doesn't do anything by itself. Press the SHIFT key and hold it down while you press the 10 number keys, "1" through "0". Now release the SHIFT key. You should see the symbols:

! " # $ % & ' ( ) 0

displayed on the TV screen. The zero key is not "changed" by the SHIFT key, and you'll see that there is no symbol above the "0" on the key top. You MUST hold down the SHIFT key for as long as you want to type the shifted or "top" symbols.

Try using the SHIFT key with the two keys to the right of the "L" key and the three keys to the right of the "M" key, as shown highlighted in Fig. 1-9. These keys will let you print the special symbols:

[ ] < > ?

All of the keys used so far pop right back up after you remove your finger from them. The SHIFT LOCK key on the left side of the keyboard (Fig. 1-6) is different. If you press it, it will stay pressed until you press it again, when it pops back up. Press the SHIFT LOCK key and it goes down, press it again, and it comes up. When it is pressed in, the computer acts as though the SHIFT key is being pressed all the time. Press the SHIFT LOCK key just to see how it operates. When it is pressed in, type some of the numbers. Now press the SHIFT LOCK key again so that it springs up. Type some numbers again. Did you see what happened?

When the SHIFT LOCK key is pressed in, the "top" symbols on the *printing keys* with two symbols are displayed on the TV screen. When the SHIFT LOCK key is in the "up" position, the "bottom" symbols on the keys are displayed.

Now let's look at three other action keys, as highlighted in Fig.

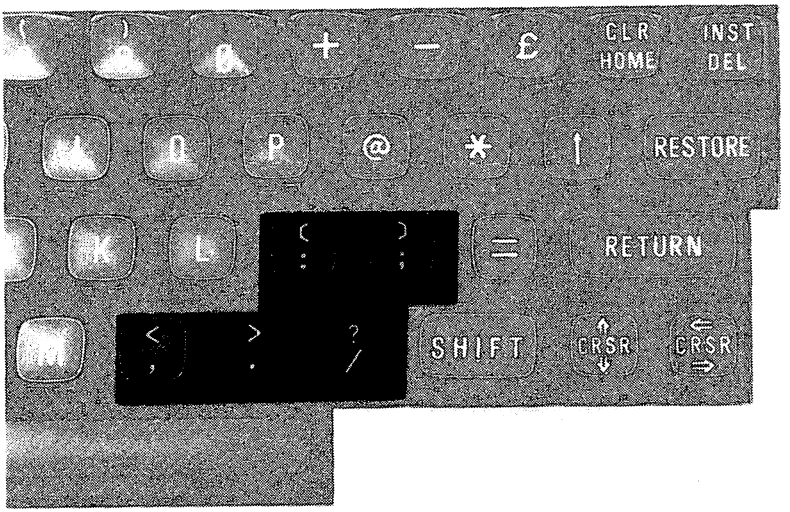


Fig. 1-9. Some of the special symbol printing keys.

1-10. The first of these is in the upper right-hand corner, and it is marked:

CLR
HOME

This key is used to do two things; it lets you move the cursor back to its home base in the upper left-hand corner of the TV display area, and it lets you clear the display area. The other two action keys are in the bottom right-hand corner, and they are both

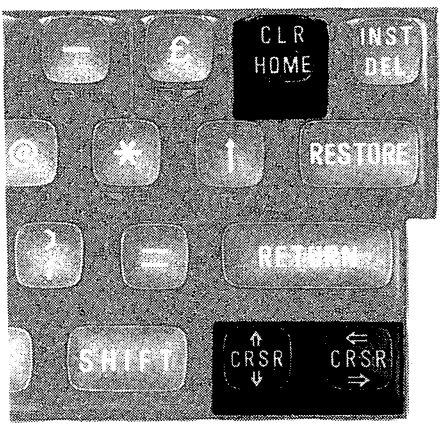


Fig. 1-10. The CLR/HOME and CRSR action keys.

marked CRSR. You can see in Fig. 1-11 that there are left and right arrows on one, and up and down arrows on the other. The two CRSR keys let you control the motion of the cursor. Here are two experiments that will show you how these three action keys work.

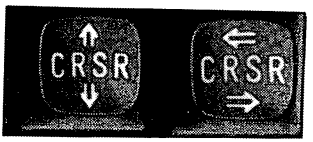


Fig. 1-11. The two cursor action keys.

Experiment 1-3. Going Home and Clearing Out

This experiment explores the uses of the CLR/HOME key. Before you get started, check the SHIFT LOCK key to be sure that it is in the up position. You can press it to check the different positions. Just leave it up when you're finished.

If you don't have many numbers, letters, or symbols on the TV screen, type several of the printing keys so that some are displayed. A few are all you need. Now, press the CLR/HOME key. What happens to the cursor, and where does it go? The cursor goes to its home base, which is in the upper left-hand corner of the large square display area on the TV screen. Is anything erased or cleared from the TV screen when the cursor goes to its home position? No, the unshifted CLR/HOME key just moves the cursor to its home base.

Letters, numbers, and other things displayed by the computer on the TV screen are called *characters*, so instead of saying numbers, letters, or symbols, from now on we'll just say they are "characters." Is the cursor now "on top of," or superimposed on a character? This will depend on what you have displayed on your TV screen. If the cursor is "on" a character, you'll see that the character isn't erased. Instead, when the cursor is off, the character is displayed normally. When the cursor is on, the character is "printed" on the cursor.

Now that you have a few characters on the screen, let's clear the TV screen. The CLR/HOME key is used for this, since it has two operations. The normal operation, HOME, places the cursor in its home position. The second operation, CLR, clears the screen. To clear the TV screen, you must press/hold the SHIFT key and then press the CLR/HOME key.

Press the SHIFT key and, while holding it down, press the CLR/HOME key. Now, let the SHIFT key pop up. What happens to the characters displayed on the TV screen? Where does the cursor go? The screen is cleared, and the central display area is "erased." The cursor is moved back to its home position in the upper left-hand corner of the TV screen. When the CLR operation is done, the TV screen is cleared *and* the cursor is moved.

Can you think of another way to clear the screen? If you keep the space bar pressed down long enough, everything will be "pushed up" off the screen. This can take up to 40 seconds, and it leaves the cursor at the bottom of the screen. The use of the CLR/HOME key is much easier and quicker.

Experiment No. 1-4. Moving the Cursor

Uses the CLR/HOME key to clear the screen. Remember to use the SHIFT key, too. Press the ASDFGHJKL keys located in the middle of the keyboard to put these nine characters on the screen. The keys are all in a row, from left to right on the keyboard.

Now use the CLR/HOME key to put the cursor at its home base. If you cleared the screen, you pressed the SHIFT key and the CLR/HOME key. Go back and type the characters again. Use the CLR/HOME key without pressing [SHIFT] to get the cursor back to its home base.

Now that you have a few characters on the screen, use the space bar to move the cursor to the letter J in your line on the screen. What happens as you press the space bar? The cursor can be moved with the space bar, but it erases characters as it passes over them. Actually, it doesn't erase them, it just puts a space in their place. The computer "sees" spaces as though they are characters, even though it doesn't print anything on the TV screen.

The two CRSR keys at the bottom right-hand corner of the keyboard, shown previously in Fig. 1-11, let you move the cursor without erasing or changing the display. The four arrows, two on each key, show the directions in which the cursor can be moved; up, down, right, or left.

Clear the screen with the SHIFT and CLR/HOME keys and use the two CRSR keys to move the cursor down and to the right so it is near the center of the TV screen. The arrows on the bottom of the key tops show the direction the cursor will move. The SHIFT key must be used to get the cursor to move in the direction shown on the top of the CRSR key tops.

This should be fairly easy to do, but you may need to practice using the CRSR keys. Remember that you can always move the cursor back to its home position by using the CLR/HOME key. Try and move the cursor back to the home base by using the SHIFT and the two CRSR keys. Can you do it? Just take your time and get used to the operation of the keys. *The CRSR keys are repeating keys,* just like the space bar. You can press them once and move the cursor one position, or you can hold them down and the cursor will move for as long as the CRSR key is pressed. Of course, you'll need to use the SHIFT key, too, to make the cursor go up or to the left.

Clear the screen again using [SHIFT] and [CLR/HOME], and type about 10 or 12 characters on the screen. Can you use the CRSR keys to put the cursor on top of, or superimposed on, any of the characters? Try it. When the cursor is "on top of" a character, the cursor and the character flash, as shown in Fig. 1-12. "On top of" doesn't mean that the cursor is above the letter.

The CRSR keys can put the cursor anywhere on the screen, within the central square area. The cursor does not erase characters when it is moved with the CRSR keys. You'll soon find out how you can use the CRSR keys to go back and make changes to the characters and other information shown on the TV screen.

Now that you have had a chance to try the CLR/HOME and the two CRSR keys, you will be able to use these keys when you need them, or when you're asked to clear the screen, send the cursor home, or move the cursor in an experiment. It will take some practice to get used to some of the special keys, but this is true for all of the computer operations. The keyboard has some other features, and we will look at these next.

YOUR ERASER

Everyone makes mistakes, so it's a good idea to have an eraser handy. The VIC 20 has a special key that can be used to "erase" mistakes, and this is the INST/DEL key located on the keyboard, as shown in Fig. 1-13. When this key is pressed, the cursor moves to the left, erasing characters as it goes. This key can be used when you are typing and discover a mistake. You can simply "back up," erase, and then continue typing.

We do not expect you to be an expert at using the INST/DEL key after you read this section and do an experiment. We just want you to know how this key can be used.

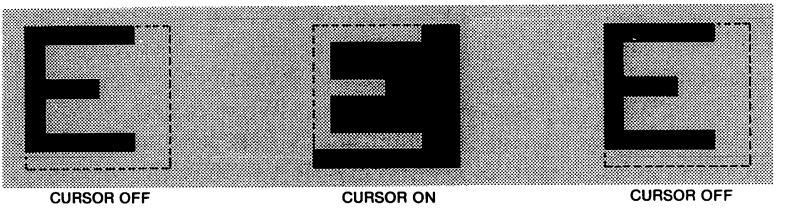


Fig. 1-12. Putting the cursor on a character makes it seem to "flash."

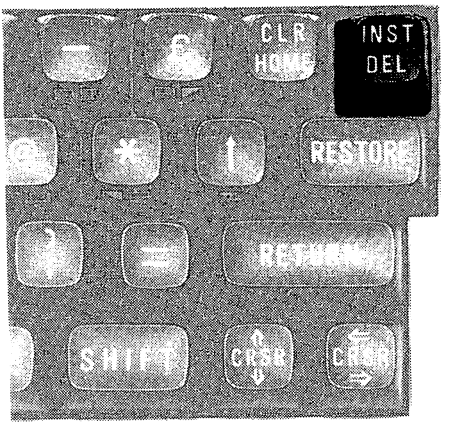


Fig. 1-13. The INST/DEL key located in the upper right-hand corner of the keyboard.

Experiment No. 1-5. The INST/DEL Key

In this experiment, you will find out how you can "erase" mistakes by using the INST/DEL key. Use the CLR/HOME key to clear the TV screen. Remember to use the SHIFT key, too, or you will simply move the cursor to its home position. Once the TV screen has been cleared, type the following:

```
THIS IS THE VIC■
```

Leave the cursor so that it is just to the right of the letter C in VIC.

Let's assume that you wanted to type THIS IS YOUR VIC 20 instead of THIS IS THE VIC. You can go back and correct the information typed on the TV screen by using the INST/DEL key. Press the INST/DEL key once. What happens to the cursor? Is

anything erased from the TV screen? The cursor moved one space to the left. Since it was placed just to the right of the C in VIC, the C was erased, leaving THIS IS THE VI■ on the TV screen. Press the INST/DEL key seven more times. What is left on the TV screen now? You should see only THIS IS■ on the TV screen. Now type a space and then type YOUR VIC 20. You should now have this on the TV screen:

```
THIS IS YOUR VIC 20■
```

Since the cursor is now to the right of the 0 in VIC 20, press the INST/DEL key and hold it down, but release it after a few seconds. What happens to the cursor now? The cursor moves to the left for as long as the INST/DEL key is pressed. It erases characters as it moves. When it reaches the left side of the TV screen, it will go on to the line above, if there is one, and will continue to erase whatever is there. When the cursor is moved back to the upper left-hand corner (home) with the INST/DEL key, it will go no further.

Clear the screen and move the cursor down four lines. Now type the following:

```
ERASE THIS LINE■
```

Now press the INST/DEL key and hold it down. What happens? The line just typed is erased and the cursor goes from the left side of the screen to the right side of the screen, but on the line above. If you keep the INST/DEL key pressed long enough, the cursor will move right back to its home position.

You can use the INST/DEL key to "erase" information on the screen whenever you wish. Just remember that it backs up and "erases" to the left of the cursor. The INST/DEL key also has an insert (INST) operation that is used to make "openings" in groups of characters so that other characters can be added with a minimum of retyping.

You can use the CRSR keys to place the cursor in the middle of a line of characters, then the INST/DEL key can be used to erase and open spaces for changes. Here is another experiment that will show you how the INST and DEL operations can be used together.

Experiment No. 1-6. Deleting and Inserting

What happens when the INST/DEL key is used to erase something in the middle of a line of characters? How do you insert characters? You'll see how it's done in this experiment.

Clear the screen and move the cursor down four lines. Type the following:

```
SEE HOW HE JUMPED
```

Move the cursor so that it is superimposed on the J in JUMPED, as shown below:

```
SEE HOW HE JUMPED
```

Use the SHIFT key and the left-hand CRSR key to do this. Once the cursor is on top of the J, press the INST/DEL key once. What happens to the characters on the TV screen? The space between HE and JUMPED is erased, so the line now looks like this:

```
SEE HOW HEJUMPED
```

Now press the INST/DEL key two more times. What happens now? Can you describe what the INST/DEL key does when it is used in the middle of a line of characters? The line is now, SEE HOW JUMPED, with the cursor still on top of the J in JUMPED. When the INST/DEL key is pressed, it "erases" a character to its left. Any character under it, and characters to its right are moved over to the left to take up the space left by the "erased" character.

Now we want to insert the word THEY, so that the line of characters looks like this:

```
SEE HOW THEY JUMPED
```

The insert operation on the INST/DEL key can be used to "open" spaces in a line of characters so that changes can be made. How many spaces would be needed in the line SEE HOW JUMPED to make it SEE HOW THEY JUMPED? You need five spaces, four for the letters in THEY and one for the space between THEY and JUMPED. To "open" five spaces, press/hold the SHIFT key and press the INST/DEL key five times. Don't hold it down, since the INST/DEL key is a repeating key, and too many spaces will be "opened." What happens to the display on the TV screen? When the SHIFT key is pressed and held and the INST/DEL key is pressed, the cursor stays in its place, but it "opens" spaces to its right. The characters to the right of the cursor are pushed to the right, one space at a time, "opening" spaces in the line of characters. Now that there are five spaces, you can simply type THEY . Remember to press the space bar once for the space between THEY and JUMPED.

Does the TV screen now display, SEE HOW THEY JUMPED?

The GRAPHICS key works the same way as the SHIFT key. It must be pressed and held when you want to display the left-hand graphic symbol on the front of a key. Again, if your computer is turned on, press/hold the GRAPHICS key and press the + key and then the "B" key. You'll see a part of a "flag" and a square full of "dots" displayed on the TV screen.

Experiment No. 1-7. Drawing Things

Use the CLR/HOME key to clear the screen. Press the SHIFT key and hold it down. Now, press some of the keys with the graphic symbols on the front. You should see the shapes, lines, and other special symbols displayed on the TV screen. The cursor moves one space to the right, just as it did when you were typing numbers and letters in some of the earlier experiments. The CLR/HOME and CRSR keys may be used to clear the screen, put the cursor at home base, or place the cursor anywhere you want it.

Clear the screen again, press the GRAPHICS key and hold it down. Now press some of the keys with the graphic symbols on the front. You should see that you are printing the left-hand figure shown on the key onto the screen. You can release the GRAPHICS or the SHIFT key at any time to print the "normal" characters on the screen.

The lines and shapes can be combined to make interesting figures and designs. For example, clear the screen and move the cursor away from the edges of the screen. Press the SHIFT key and then press the "U" and "I" keys. Let the SHIFT key come up again, and move the cursor down one space and to the left two spaces. You'll have to use the SHIFT key with the CRSR keys to move the cursor where you want it. You can probably do this without any more help. Now, press the SHIFT key and then press the "J" and "K" keys. What shape did you draw? You should find that you have drawn a circle on the screen. What happens when you print a shifted "W" key? A smaller circle appears.

Here is a more complicated figure you can create. Clear the screen and move the cursor so it's near the center of the TV screen. Press/hold the GRAPHICS key and type the letters "NBB." Release the GRAPHICS key and move the cursor down one line and three spaces to the left. Press/hold the GRAPHICS key and type the letters "NBB" again. Release the GRAPHICS key and move the cursor down one line and to the left three spaces. Press the GRAPHICS key and the "N" key. Move the cur-

sor down one line and to the left one space. Press the GRAPHICS key and the "N" key. You should have the checkered winner's flag in the center of your TV screen.

It won't take you long to learn how to use the graphic symbols, since it's more fun to draw things than it is to simply sit and type numbers and letters. You can move the cursor anywhere on the screen that you want to, and you can clear the screen as needed. Here are a few things to keep in mind as you draw figures and shapes with the special graphic symbols:

1. You can move the cursor "over" anything on the screen without destroying it. If you want to change something, simply use the cursor controls to get to the character or symbol and retype it.
2. When you clear (CLR) the screen, your graphic artwork or typing will be erased and you won't be able to get it back.
3. When the cursor reaches the bottom border, you can move it "down" another line, but this will cause everything on the screen to move "up" a line. The top line will be pushed up "off" the TV screen and lost. The display is said to "scroll" up.
4. When using the GRAPHICS key, you'll find that it acts just like the SHIFT key when you use keys with two symbols on the top. For example, pressing [GRAPHICS] and the "4" key will print a dollar sign ($) on the screen.

COLOR FUN

When the computer is turned on, the color of the central area on the TV screen is white, and the cursor is blue. If you have a color TV set, you can change the color of the cursor and the characters that are to be printed. If you have a black-and-white TV set, you will see the color changes as shades of grey. The color of the cursor is changed by using the number keys, "1" through "8." In Fig. 1-17, you can see that these keys have three-letter color names on the front of them, and these colors are also shown here:

KEY	COLOR
"1" BLK	Black
"2" WHT	White
"3" RED	Red
"4" CYN	Cyan
"5" PUR	Purple
"6" GRN	Green
"7" BLU	Blue
"8" YEL	Yellow

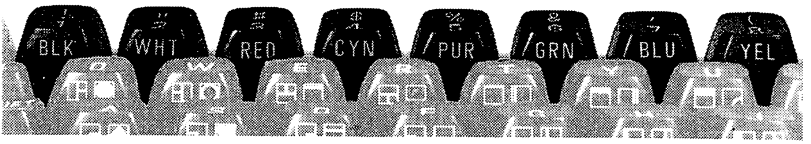


Fig. 1-17. The color codes are on the front of the number keys, 1 through 8.

The control key, marked CTRL, is used to change the cursor's color, and it is shown highlighted in Fig. 1-18, right below the "1" key. If you press and hold the CTRL key and one of the color keys, the color of the cursor will change to the color noted on the key. When you release the color key, the cursor will stay the color you have selected, unless you change it by pressing the CTRL key and one of the other color keys.

Experiment No. 1-8. The Color Connection

Clear the screen. Press/hold the CTRL key and press the "1" key to change the cursor to black. Now type the word, BLACK. You should see BLACK printed in black letters on the screen. Can you change the color to red? Press/hold the CTRL key and press the "3" key, the one with the RED notation on it. Type the word, RED. Change the color to cyan (CYN), which is light blue, and

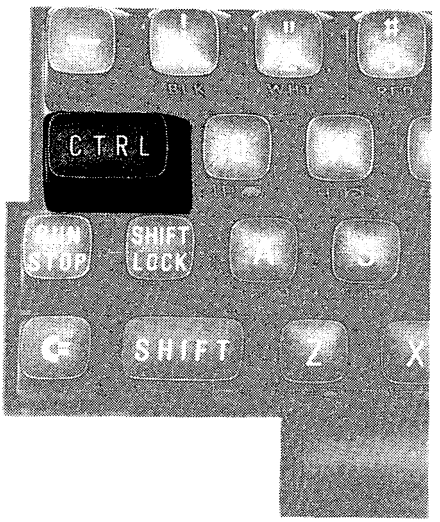


Fig. 1-18. The CTRL key is at the upper left side of the keyboard under the "1" key.

type CYAN. Change the color to purple (PUR) and type PURPLE, change it to green (GRN) and type GREEN. Change the color to blue (BLU) and type BLUE, and finally change it to yellow (YEL) and type YELLOW. Do you have all of the colors spelled out now?

If some of the colors look a bit odd, or if the shades don't seem to be right, you may want to adjust your color TV set, or have someone adjust it for you. If you have a black-and-white TV set, you will see that the cyan, green, and yellow "colors" look light grey. It is difficult to tell much of a difference between them.

Even when you have changed the color, you can still use all of the *printing keys*, including the special graphic characters. You can change the color at any time to put green dots, purple triangles, red squares, and other colorful figures on the TV screen. Many of these are used in games, educational programs, and other programs where it is useful to use colors for emphasis or special effects. When you change from one color to another, the characters already being displayed on the TV screen are NOT changed.

The computer can also "reverse" its display to highlight words, numbers, or special symbols. You can turn the "reverse" on and off, using the reverse on (RVS ON) and reverse off (RVS OFF) keys, found on keys "9" and "10," as shown in Fig. 1-19. The reverse operation is controlled by the CTRL key, as are the color keys.

Experiment No. 1-9. Printing in Reverse

Clear the screen and change the color to black. Try to do this without help, and without going back in this chapter. If you need help, press/hold [SHIFT] and press [CLR/HOME]. Now, press/hold [CTRL] and press [BLK] (the "1" key). That should do it.

To reverse the printing, press/hold the CTRL key and then

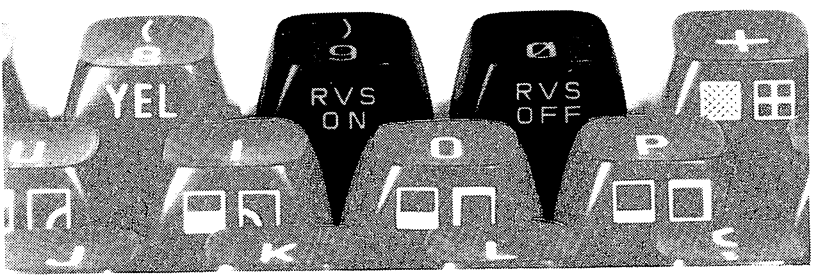


Fig. 1-19. The "9" and "0" keys also turn the reverse printing on or off.

press the RVS ON key (the "9" key). Do you see any changes on the screen? No, nothing has been printed yet. Type the word TEST. What do you see on the TV screen now? The word TEST is printed on the screen. The letters are the color of the big square area on the TV screen (white) and they are printed *on* the color you selected (black).

Change the color to yellow (press/hold [CTRL] and press [YEL]). The cursor should now be yellow, or light grey on a black-and-white TV. Type TEST again. What do you see on the screen? The letters are the same color as the large square area, but they are printed on a *yellow* background. You can experiment with reverse printing for the other colors, too.

To turn the reverse printing off, press/hold [CTRL] and press [RVS OFF] (the "0" key). Now you will be able to type characters normally. The areas that were printed in reversed colors *stay that way*. They do not change.

When the reverse printing is turned on with the CTRL and "9" keys, the characters already printed on the TV screen are not changed. Only the characters typed *after* the change in the display format are reversed.

We haven't talked about the color white (WHT), found on the "2" key. What do you think would happen if you change to this color? If you decide to try it, you'll see that the cursor disappears. When the printing color is changed to white and the background color is white, you won't be able to see the characters. White characters on a white background don't show up.

If you haven't already changed to the color white, do it now. Type some characters and change back to the color blue. Has the cursor moved from where it was when you changed to white? Yes, it has, since it still "prints," even though you can't see the characters you've typed in. Remember that changing back to blue doesn't change those characters already typed to blue. They stay the way they are, including the white characters you typed in (and still can't see).

SECRETS

We didn't talk about the RETURN, RESTORE, or RUN/STOP keys in this chapter. They are not secret keys, but they don't have much to do with the keyboard or TV display. These are *action keys* that control the operation of the computer. We will talk about them in another chapter.

QUESTIONS

1. How do you set up the computer to use channel 3 on the TV set for its display?

2. What types of keys are there on your VIC 20, and what kinds of things do they do?

3. What does the SHIFT key do?

4. What two operations does the CLR/HOME key do?

5. Why do you use the CRSR keys?

6. Do any of the keys "repeat" their action if they are held down for a while? Which keys do this?

7. What does the INST/DEL key let you do?

8. What does the GRAPHICS key look like, and what does it do?

9. How can you change colors on the TV screen?

10. What color do you get when you press the SHIFT key and the "6" key?

11. If you type several words and then change the color of the cursor, is the color of the previously typed words changed, too?

FOOD FOR THOUGHT

If you look at all the graphic symbols, you'll see that there isn't a symbol for a solid block, the size of the cursor. However, you can display a solid block of color (not the cursor) on the TV screen. How can this be done?

2

Using a Small Computer

WHAT CAN YOUR COMPUTER DO?

Now that you're familiar with the keyboard, you're ready to start using your VIC 20 computer for more than typing and drawing, but what exactly do computers do? If you think about computers in a very general way, you'll find that they simply *process information*. In fact, computers were originally called *electronic data processors*. Here are some ways in which computers are used: traffic-light control, bank accounting, cash register, calculator, spacecraft, or satellite control.

Computers have to get the information they're going to process and they have to do something with it when they are finished, so they also *transfer information*. Computers have two main functions; processing and transferring information.

Let's look at the satellite computer shown in Fig. 2-1 to see if computers really work this way. The satellite's computer gets information from various sensing devices that tell it where it is, how fast it's going, and so on. This information is processed and then transferred to various devices; to the rocket motors to reposition the satellite, and to a radio transmitter to send the new position back to an earth station. The principle is the same; information is transferred and processed.

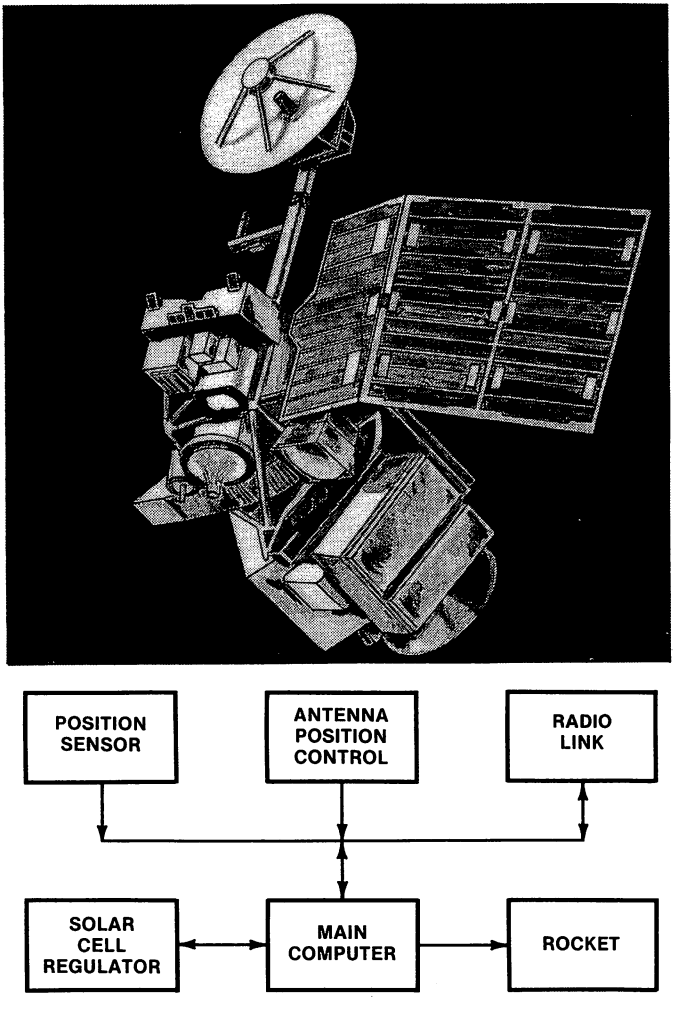


Fig. 2-1. Block diagram of a simple satellite computer system.

It's a bit awkward to talk about information being transferred to and from a computer, so the words *input* and *output* are commonly used. For example, information is input to the computer and after it is processed, it is output to a printer or a TV screen. In almost all cases, input refers to *input to the computer*, while output refers to *output from the computer*. The devices that put information into the computer are simply *input devices*, and the keyboard on your computer is a good example of one. Devices that receive informa-

tion from the computer are *output devices*, and the TV screen is a good example.

These devices are often put under the heading of input/output devices and are called I/O devices for short. Here are examples of some I/O devices that can be easily connected to your VIC 20 computer:

I/O DEVICES	USE
Game Paddle	INPUT
Printer	OUTPUT
Game Cartridge	INPUT
Disk	INPUT and OUTPUT
Cassette	INPUT and OUTPUT

The disk and cassette I/O devices can be both input and output devices, storing information from the computer and later transferring it back to the computer. The I/O devices are often called *peripherals*, and you will see this word used to describe all types of I/O devices.

You'll need to give the computer only a few simple instructions to get it to input or output information, and we will describe these later in this chapter.

SIMPLE PROGRAMS

Now that you have an idea of what your VIC 20 can do, you can write some simple programs. You have already used the computer to do some simple keyboard operations, and you may have played some games or used the computer for something else. It's really pretty easy. Here's a bit of history to ease you into programming your VIC 20.

Just as we think and communicate using the English language, the VIC 20 is set up to think and talk in a language called BASIC. We'll just call it BASIC from now on. BASIC was developed at Dartmouth College between 1963 and 1964, and its purpose has been to provide beginners with a simple, easy-to-understand way to use computers. BASIC *is* basic, and it gives you an ideal way to start using your VIC 20 computer.

You can't destroy your computer by programming it the wrong way. You may not get the answer you want, or the computer may tell you that you've done something wrong, but you can't destroy it. You're the computer's master, too. You can always pull the plug, walk away, and take a break. In fact, it's a good idea to do that every once in a while.

GETTING STARTED

Your computer can be set up to process information for hours at a time, but if you can't get information in or out, the processing doesn't do you any good. So, your first step in programming is learning how to get the computer to input information and to output it, too. There are two "words" in BASIC that you can use to tell the computer that it is to input and output information. They are INPUT and PRINT. In almost all computer languages, these are called *commands*, since you use them to command the computer to do something.

In the VIC 20 computer, you use the INPUT command to get information from the keyboard. You use the PRINT command to put information on the TV screen so you can see what the computer has done. These commands are easy to use and you'll get a chance to try them very soon.

You will write simple BASIC programs using individually numbered instructions, or steps, for the computer. This lets you and the computer keep track of where you are and where you are going. In BASIC programs, each operation or command is put in a separate "sentence" or *line*. The lines are numbered in sequence with *line numbers*, much like the numbered steps in instructions for assembling a bicycle.

Here is a simple BASIC program that will give you an idea of what's ahead:

```
10 INPUT ALPHA
20 PRINT ALPHA
```

In this simple program, line numbers 10 and 20 are used to guide the computer through the program, one line at a time. Lines 11 through 19 aren't "missing," they just haven't been used in this program. The computer simply goes from one line number to the next highest one, so after finishing the command at line 10, it goes to the command at line 20. Most people who write computer programs in the BASIC language use line numbers that are multiples of 10, starting with 10; 10, 20, 30, and so on. You don't have to number your lines this way, but most programmers do. The following program will do the same thing as the preceding one:

```
 12 INPUT ALPHA
130 PRINT ALPHA
```

What does this simple program do? Here's a general description: The first thing it does is to command the computer to get a value

from the keyboard (INPUT ALPHA). We have told the computer that we want this number to be called ALPHA, so we can easily identify it later on in the program. Once the value has been typed in and labeled ALPHA, the computer prints it on the TV screen.

Experiment No. 2-1. A Simple Input/Output Program

In this experiment, you'll learn how to use the keyboard to put a short program into the computer. This is called *entering* a program.

Turn on your computer. If it is on, turn it off, wait a few seconds, and turn it back on. This will give you a fresh start, clearing the computer and the screen.

Type in the following program, one character at a time. Use the RETURN key on the right side of the keyboard to tell the computer when you have finished typing or entering each line. We'll show the RETURN key in brackets, [RETURN], so you'll know when to press it. Don't worry if you make a mistake. We'll show you how to correct it in a minute.

```
10 INPUT ALPHA [RETURN]
20 PRINT ALPHA [RETURN]
```

The RETURN key tells the computer that you have finished an operation. You'll find that using the RETURN key becomes second nature, but we'll tell you when to press it in the first few experiments to get you started. Remember, you don't type the word "RETURN," you just press the RETURN key.

Now you will check to be sure that your program has been entered into the computer correctly. You can ask the computer to "list" your program on the TV screen so that you can review it and check for any errors. This is called *listing your program.*

To list a BASIC computer program on the TV screen, simply type the word LIST and press [RETURN]. Go ahead and type LIST [RETURN]. What happens on the TV screen? You should now see your program displayed on the TV screen. This type of display is called a "listing." Check the listing displayed on the TV screen with the previous program to see if you have entered the program correctly. If you have made mistakes while typing or entering the program, don't worry, they are easy to correct.

If you slipped and used an incorrect line number, when you list your program it might look like this:

```
10 INPUT ALPHA
29 PRINT ALPHA
```

You can remove line 29 from your program by typing just the line number followed by [RETURN]; that is, 29 [RETURN]. You can also retype any line that is incorrect. You don't have to erase it first. Just start with the same line number, retype the information and press the RETURN key. The computer automatically replaces the old line in your program with the new one.

Now, retype any incorrect lines and erase any incorrect line numbers. You can retype incorrect lines as many times as needed until you get them right. You can also list your program as many times as you want to check it. Before going on, be sure your program looks like this:

```
10 INPUT ALPHA
20 PRINT ALPHA
```

Change line 10 by typing this into the computer:

```
10 INPUT ALFA [RETURN]
```

Type LIST [RETURN] and see what has happened to your program. You should see the following listing on the TV screen:

```
10 INPUT ALFA
20 PRINT ALPHA
```

Could you get line 10 back to: 10 INPUT ALPHA? It's not difficult to do, simply retype the correct line and press [RETURN]. Let's put another line in the program and change line 10 at the same time. Type in the following:

```
10 INPUT ALPHA [RETURN]
36 INPUT COUNT [RETURN]
```

Now list the program by typing LIST [RETURN]. The program listing should look just like this:

```
10 INPUT ALPHA
20 PRINT ALPHA
36 INPUT COUNT
```

When lines are entered from the keyboard, they can be entered in any order. The computer automatically sorts them out and will print them in the correct order when you list the program. Just remember to use the RETURN key when you finish typing each line.

Do you remember how to get rid of the command at line 36? You can simply type 36 [RETURN]. Do this, and then list the program

on the TV screen by typing LIST [RETURN]. You should see only lines 10 and 20 in the listing:

```
10 INPUT ALPHA
20 PRINT ALPHA
```

In this first programming experiment, you have already learned quite a bit. You have:

- Entered a simple computer program.
- Used several different line numbers.
- Corrected errors in line numbers.
- Added and deleted lines.
- Learned how to list a program on the TV screen to see that it is correct.

Leave your computer on if you're going to go ahead to the next section. Otherwise, you can turn it off. Remember that you will not "hurt" or "destroy" the computer by typing in incorrect lines or commands. The computer may not understand what you're asking it to do, and it will let you know quickly. You can't hurt it through errors in your programs, either.

Experiment No. 2-2. Running Your Program

Now you will learn how to tell the computer to run or "do" a simple two-line program, and you will learn how the RUN command is entered from the keyboard.

So far, you've simply entered a program and checked it to be sure it was entered correctly. If the following program is not in your computer, enter it now:

```
10 INPUT ALPHA
20 PRINT ALPHA
```

When typing in a program, remember to press the RETURN key after typing each line:

```
10 INPUT ALPHA [RETURN]
20 PRINT ALPHA [RETURN]
```

How do you get the computer to run the program? What do you think will happen when the program is run? You start a BASIC program by typing RUN [RETURN]. You would type the three letters R, U, and N, and press the RETURN key. Don't get confused by the RUN/STOP key on the left side of the keyboard. This

key has a special use, but it doesn't start programs. DON'T USE IT!

When a program is started with a RUN command, the computer goes to the lowest line number and starts to do the things that you have told it to, one line at a time. In this case, the computer goes to line 10 and starts an INPUT operation. You'll probably remember that an INPUT command asks you to enter information from the keyboard.

Before you run your program, clear the screen. Do you remember how to do this? (Press/hold [SHIFT] and press [CLR/HOME].) Type RUN [RETURN]. What happens on the screen? Your screen should look like this:

```
RUN
? ■
```

The question mark and flashing cursor show that the computer is waiting for you to enter information from the keyboard. In this case, the computer is waiting for you to type in a number, or value. To enter a value, you simply type in the numbers you want to enter and then press the RETURN key, for example, 1239 [RETURN].

Type in the three-digit number, 345. Does anything happen? You must press the RETURN key to tell the computer that you've finished entering information. Once you have entered your number and pressed the RETURN key, your screen should look like this:

```
RUN
? 345
 345

READY.
■
```

The computer has input the value with the INPUT ALPHA command, and it has printed it with the PRINT ALPHA command. The label ALPHA was used to identify the value for the computer. The value we chose for this experiment was 345. Any numeric value (within the computer's range) could have been entered with similar results. There is nothing special about the choice of 345.

Run the program again by typing RUN [RETURN]. Now enter a number of your choice between zero and a million. Don't forget to press [RETURN] to tell the computer that you have finished typ-

ing your value. Did the computer display the value properly? It should have.

When the READY message and the flashing cursor are displayed, this tells you that the computer is READY for you to tell it what to do. You could tell it to run a program again, print a listing of a program, or accept some lines typed in for a new program.

Do you think the computer "used up" your program? You can ask the computer to list your program to see if it's still there. Go ahead and type: LIST [RETURN]. Is the program still in the computer? It should be.

Experiment No. 2-3. Bad Data

Now you will program the computer to input a value labeled ALPHA. When the computer "asks" for the value, you will see what happens if you try and trick it by entering something else.

If the following program is not in your computer, enter it from the keyboard:

```
10 INPUT ALPHA
20 PRINT ALPHA
```

If you are unsure of how to do this, go back and do Experiment No. 2-1, then do this experiment.

Clear your screen and run the program. (Hint: Simply press/hold [SHIFT] and press [CLR/HOME]. Now type in: RUN [RETURN].) Let's try and fool the computer. Since the computer expects you to type in a value; that is, several numbers, what do you think will happen if you type in letters instead?

When you have started the program and the question mark can be found on the TV screen near the flashing cursor, type in the characters, WAS, and press the RETURN key. What shows up on the TV screen after you press the RETURN key? The computer prints:

```
RUN
? WAS
?REDO FROM START
? ■
```

This message tells you that your letters, "WAS," were "thrown out" by the computer, because it expected you to type in a value, instead. It knows that "WAS" isn't a value, so it gives you another chance to input a number.

Now type in 5.8.9. [RETURN]. What is displayed on the TV screen? The information you typed in, 5.8.9., isn't a value, so another REDO message appears and the computer gives you another chance to enter a value.

Type in $456 [RETURN] and see what happens. The REDO message appears again. Is this what you expected? Maybe you thought the computer would accept $456 as a value. Is $456 a number? It looks like four hundred and fifty six dollars to you and me, but the computer throws it out. Why? The dollar sign is a short-hand notation that is used by *people* in place of the word "dollars." The computer doesn't know that a dollar sign represents money. In fact, it doesn't "know" that a $ is a dollar sign. To the computer, it's just another *non-numeric character*.

So far, the computer works only with values, so you can't type in 15 POUNDS, 34 OUNCES, or 1250 SWISS FRANCS. It will only work with 15, 34, 1250, or some other value. Remember the computer wants you to enter values with *only the digits 0 through 9*. You can also use one decimal point and either a plus (+) or minus (−) sign, but that's all. Here are some examples:

0.00987 −987.084 +345 42 −0.081

Notice that no %, #, $ or other special symbols are used with these values. If they are, the computer will not accept your information. The plus sign in front of 345 is not really needed, since +345 and 345 are the same. As far as the computer is concerned, the following are *not values*:

#7 $47.98 15%

LABELS, LABELS, EVERYWHERE

Although you may not have realized it, the computer used a "label" to identify the values you typed in in the experiments you just did. Labels are very important, since they let the computer keep track of its information.

Can you imagine walking into a drug store that didn't have labels on any of the bottles or packages? How would you find the aspirin? It sure would be confusing. What about having a checking account with a bank that didn't use names or account numbers:

"I'd like to cash a check, please."
"Which account is yours, madam?"
"Well, I have $458.98 in it."

"Yes, madam but we have 15 accounts with $458.98 in each one. Which one is yours?"

It's pretty hard to imagine life without "labels" or names for things so we can identify them. Computers have the same problem; we need to label or identify information for them, so they know what to do with it and how to process it.

Here's a problem for you. You sold 14 cartons of eggs and there are 12 eggs per carton, how many eggs were sold? Everything in this problem has a "name"; cartons, eggs-per-carton, and eggs. You can set up the problem as shown in Fig. 2-2. When you're finished solving the problem, the answer is given the label, EGGS. In fact, this same type of "problem solving" can be used in a computer program. The only difference is that most computers use an asterisk, *, in place of the usual multiplication sign, ×. Here is the calculation part of the program.

```
EGGS PER CARTON = 12
CARTONS =14
EGGS = EGGS PER CARTON * CARTONS
```

In this problem, we first assigned the value of 12 to the label, EGGS PER CARTON, and we assigned the value 14 to the label, CARTONS. Of course, the word CARTONS isn't equal to 14, that doesn't make any sense. At the end of the problem, we've given the total number of eggs (168) the label, EGGS. Now, you could tell the computer to print the number of eggs you sold. It's not difficult at all.

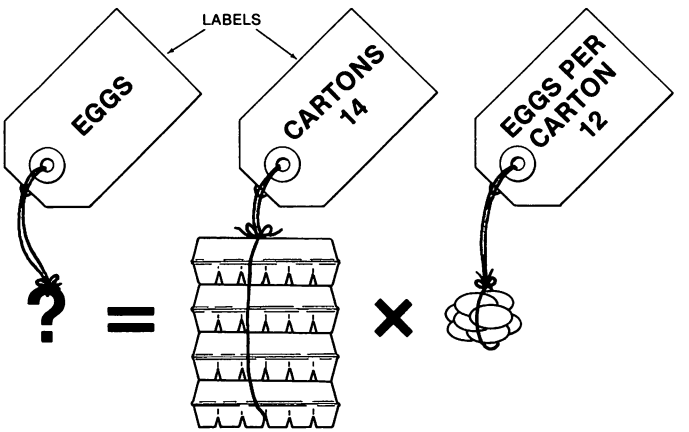


Fig. 2-2. Using labels to identify information.

What do you think the computer would do if you forgot to give a value to the label, CARTONS? The computer would look at the label, CARTONS, and find that there was no value attached to it. Without a value for CARTONS it would tell you that it couldn't figure out the number of EGGS. You couldn't figure it out either. The computer wouldn't say, "HOW MANY CARTONS, DUMMY?", but it might say "ERROR . . .". So, you can see how important labels are.

VIC 20 LABELS

In the VIC 20 computer, labels can have up to 255 characters, but *the first character must be one of the letters of the alphabet, A through Z*. The characters that follow the first letter can be numbers or letters, and you can use spaces as in EGGS PER CARTON to make labels readable. It's not too easy to read EGGSPERCARTON. Which of the following labels could be used in your VIC 20 computer?

DAYS	EGG ROLLS	MARKER
MOTHS	ACCOUNT NUMBER	DATE OF TEST
30TH	7A	ACT3

Except for 30TH and 7A the other labels are fine. These two don't follow the rule of using a letter to start a label.

It's more meaningful to use the label, EGG ROLLS, for the number of egg rolls sold than to use the totally unrelated label, MCW. Someone looking through a program is going to wonder what MCW stands for. There's not much question about the label, EGG ROLLS. Meaningful labels serve a purpose, allowing you to clearly identify things.

However, *the VIC 20 computer uses or "sees" only the first two characters in a label*. This means that in the preceding list, while *you* see ACCOUNT NUMBER and ACT3, the VIC 20 sees the two-character label, AC, for both. The same is true for the labels, *EG*GS and *EG*GS PER CARTON. Avoid choosing labels with the same first two letters, or the VIC 20 will think they are the same thing! This just happens to be the way the VIC 20 operates. Larger computers use many more characters in their labels, simplifying things a lot. Well, you can't have everything in a small computer.

Since the VIC 20 uses only the first two characters, how would you get around this in the egg-counting problem we discussed before? Try the following:

```
CARTONS = 14
AMT PER CARTON = 12
EGGS = AMT PER CARTON * CARTONS
```

Now there is no conflict between the first two letters in any of the labels. The computer sees:

```
EG as the label for EGGS
AM as the label for AMT PER CARTON
CA as the label for CARTONS
```

This seems to be fine, and it brings us to two programming rules:

Rule 1: Make your labels meaningful.

Use meaningful labels for things. If you use labels such as YT for eggs, QW for eggs per carton, and P for cartons, your calculation will look like this:

```
YT = QW * P
```

With these labels, it's not easy to see what you're doing.

Rule 2: Avoid using the numbers 0 and 1, or the letters O or I, in labels.

The numbers 0 and 1 are easily confused with the letters O and I, and if you look at a label quickly, it's easy to be confused. For example, VO; is it vee-oh or vee-zero? It may be difficult to tell.

Experiment No. 2-4. Short labels

This experiment will show you that the computer really uses only the first two characters in the label to identify information.

In the following program, the label BETA is used to identify a value, but the VIC 20 will use only the first two letters, BE, to identify information. Let's try and confirm this.

Type the following program into your computer. Don't forget to use the RETURN key at the end of each line:

```
10 INPUT BETA
20 PRINT BETA
```

List the program on the TV screen to check that it has been typed in correctly. Run the program and test it by entering 234 or any other three-digit number. You should see the 234 or your three-digit number printed on the screen, followed by the READY message.

If an error message, or something else unexpected is displayed on the TV screen, check your program again. You can list a program on the TV screen at almost any time, but you can't list it when the computer is waiting for you to input information. That is, when the flashing cursor is "waiting" after a question mark on the TV screen.

Once the two-line program has been run and tested successfully, change line 20 to:

```
20 PRINT BE
```

Remember, you just type in the new line 20, as shown, and press the RETURN key. The new line is substituted for the "old" line 20 in the program. List your program, and it should look like this:

```
10 INPUT BETA
20 PRINT BE
```

Run this program and type in a value. Does the computer operate the same way it did when the PRINT BETA command was used at line 20? It should, since the VIC 20 computer uses only the first two letters of a label to identify the information. This means that you shouldn't try and use the labels BETA and BE for different pieces of information in a program, since the computer will think they are the same. Be careful!

THE COMPUTER INFORMATION DIET

Your VIC 20 computer can process or "work on" two general types of information. The first type of information is simply a *value*, and you have already seen that the computer can input and print a value. Later you will see how the VIC 20 can add, subtract, multiply, divide, and use the values in other ways. The VIC 20 can work with some very large and small numbers, probably larger and smaller than you can imagine right now.

Suppose that you want the VIC 20 to keep a list of names and phone numbers. How does the computer work with a name such as Stew Watson? It's pretty obvious that a name such as this can't have a value, so this brings us to the second type of information that the computer can process; groups of characters, just like the ones in this sentence. You'll remember that we talked about characters in Chapter 1, and you learned that characters are simply symbols such as A through Z.

The computer "sees" Stew Watson's name as a group of char-

acters; S, T, and so on. Once the computer has groups of characters, it can see if two names match, it can put characters together to form sentences, and it can search a mailing list for all your friends in Washington, D.C. There are lots of useful things you can do with groups of characters, and computers use them every day. Since *groups of characters* is a bit of a mouthful, the word *string* is used to identify these groups.

The following list contains both "values" and "strings." Try and figure out which group each belongs in. You can circle your answers or use a sheet of paper.

Smith	String or value
12.87	String or value
Green paint	String or value
23	String or value
Main Street	String or value
24060	String or value

Smith, Green paint, and Main Street are all strings. It's not easy to tell if 12.87, 23, and 24060 are values or strings. The 24060 looks like a ZIP or mail code. Maybe it's not really twenty four thousand and sixty, but two-four-oh-six-oh. Since the computer lets you put labels on things, it's really not that difficult to tell the strings and values apart.

TYPES OF INFORMATION AND LABELS

Labels are used to identify information, and they are also used to tell the computer what *type of information* is going to be processed. If you want to give labels to values, just use letters and numbers in the label. The first character in a label must be a letter, A through Z. For example:

```
BALANCE = 234.76
TEST NUMBER = 70
M32 = 34689
MILLI = 0.001
```

Just remember that the longer labels are for *your* convenience; the VIC 20 sees them as the first two characters:

```
BA = 234.76
TE = 70
M3 = 34689
MI = 0.001
```

When the computer is going to process strings of characters, labels are also used, but you put a dollar sign ($) at the end; for example, TEST$. The dollar sign is a signal to the computer that the label is assigned to a string, and not to a value. Here are a few examples:

```
NAME$ = "BOB EBBS"
DATE$ = "SEPTEMBER 17, 1948"
TEST$ = "TESTING, 1, 2, 3."
```

The VIC 20 sees the string labels as the first two characters *and* the dollar sign:

```
NA$ = "BOB EBBS"
DA$ = "SEPTEMBER 17, 1948"
TE$ = "TESTING 1, 2, 3."
```

In a previous example, the *value* 70 was assigned to the label, TEST NUMBER. Would this confiict with the use of the label, TEST$, for a *string*?

```
TEST NUMBER = 70
TEST$ = "TESTING 1, 2, 3."
```

No, there is no conflict, since these are *two different types of information*, and the computer sees the label, TE, identifying a value and a different label, TE$, identifying the string.

```
TE = 70
TE$ = "TESTING 1, 2, 3."
```

. *Note that string labels always end with a dollar sign.* This information tells the computer that your string is simply a collection of characters and not a value, even though numbers may be used quite properly in a string of characters, as shown.

(NOTE: The closing quote mark at the end of a string is not required, but we think it makes good sense to use it. The closing quote mark clearly shows where the string ends. We will use both quotes throughout this book.)

Here are some examples of labels and the information assigned to them. Can you tell which ones are in the proper form, and which ones are not?

(a) SAMPLE = 34.8972
(b) GUN = "M-16 RIFLE
(c) NUMBER OF NAMES = 45

(d) CITY$ = "HAMILTON"
(e) LENGTH = 12
(f) TIME$ = ONE O'CLOCK
(g) DOLLARS = ONE
(h) DIRECTION = "ONE WAY"

Here are the answers:

(a) Correct.
(b) Incorrect. The label, GUN, needs to be GUN$, and we recommend a quote mark at the end of the string: GUN$ = "M-16 RIFLE"
(c) Correct.
(d) Correct.
(e) Correct.
(f) Incorrect. The ONE O'CLOCK must be put in quotes to be a string: TIME$ = "ONE O'CLOCK"
(g) Incorrect. It's hard to tell if it's supposed to be DOLLARS = 1 or DOLLARS$ = "ONE"
(h) Incorrect. The DIRECTION label needs a dollar sign at the end: DIRECTION$ = "ONE WAY"

Experiment No. 2-5. Inputting Strings

If the VIC 20 computer is to work with strings, there must be some way to input them. In this experiment, you'll see how labels for strings can be used in INPUT commands.

You can use the computer to input strings of characters in place of values. All you have to do is use a label for a string in the INPUT command in place of a label for a value. This tells the computer what type of information is to be typed in from the keyboard. Enter the following program into your computer:

```
10 INPUT TY$
20 PRINT TY$
```

You now have the label TY$ set up to identify the *string* that is to be input by the INPUT TY$ command at line 10. The PRINT command at line 20 uses this label to locate the string to be printed on the TV screen. The dollar sign at the end clearly identifies the string type of label.

Clear the screen and run the program. When the question mark and flashing cursor appear, type in:

```
VIC FUN
```

Did you forget to press the RETURN key after typing VIC FUN? If so, press it now. What does the computer print on the TV screen? You should see this on your screen:

```
RUN
? VIC FUN
VIC FUN

READY.
■
```

Run the program again and type in another short string of your own. Is it displayed on the TV screen? It should be.

Run the program again and type in the numbers 12345. What is displayed on the TV screen? Did you expect an error message or a REDO message because you entered a number? The computer simply displays 12345. There is no error message, nor a REDO message, because the computer simply treats 12345 as a string of five characters, which just happen to be digits. We used the string "September 17, 1948" in one of our examples earlier in this chapter. The date has no "value," it's just a string of characters to the computer. This means that you must distinguish between strings and values when you use labels in your programs.

Experiment No. 2-6. Mixed Labels

The VIC 20 computer can input and use strings and values in the same program. The proper labels are all that are needed. Load the following program into your computer:

```
10 INPUT XP
20 INPUT TR$
30 PRINT TR$
40 PRINT XP
```

Run the program. When the first question mark appears, type in a value. When the second question mark appears, type in your last name. What happens when you press the RETURN key after typing in your name? The computer prints your name and then it prints the value you typed. The computer was able to label each piece of information so it could use it later. The type of label used lets the computer tell whether it is to work with a value or a string. Change lines 20 and 30 to:

```
20 INPUT XP$
30 PRINT XP$
```

The complete program should now look like this:

```
10 INPUT XP
20 INPUT XP$
30 PRINT XP$
40 PRINT XP
```

Run the program and again enter a value after the first question mark, and then a string of five or six letters after the second question mark. Are the string and the value displayed as you expected? They should be. To the VIC 20, XP and XP$ are two *different* labels, one for a string (XP$), and the other for a value (XP).

If you were trying to use the computer, it would be very difficult to know when to type in a value and when to type in a string in response to the question mark and flashing cursor. Now you'll learn how to make your programs easier to use.

INTELLIGENT QUESTIONS AND ANSWERS

In the previous program examples and experiments, you used simple programs that input information from the keyboard and then displayed it on the TV screen. When the computer was waiting for you to input information, it displayed a question mark and the flashing cursor:

```
? ■
```

When you see this on the TV screen, there's no way of knowing what the computer is waiting for. Maybe it needs your name (a string), or perhaps it's waiting for your bank account total (a value); who knows? Imagine that a friend has sent you a check-account program to run on your VIC 20. Your friend promises that the program will quickly and painlessly balance your checkbook. This sounds great. You type in the long program and run it. The computer responds with the ? ■ display. What's next, the number of checks, the starting balance, or something else? You would need a complete instruction book just to run the checkbook program.

To help you use the computer, you can put special messages in the INPUT and PRINT commands, along with the usual labels for values and strings. These messages will be displayed on the TV screen when the computer is doing INPUT and PRINT commands

so that you'll know what you are to type in, and what is being "printed" on the TV screen.

When messages are used, you can get this type of display on the TV screen:

```
POUNDS OF CLAMS CAUGHT TODAY? 480
COST PER POUND IN DOLLARS ? 3.40
GROSS INCOME = $ 1632
YOUR WEEKLY CATCH = 1257 POUNDS
```

This display is much more meaningful than seeing the following on the TV screen:

```
? 480
? 3.40
1632
1257
```

It is much easier to see what is happening in the first example. Someone has caught 480 pounds of clams and sold them for $3.40 per pound. The gross income was $1632. The computer program is also keeping track of the weekly total catch. In the second example, the same information is typed in, processed, and displayed, but without the messages the user doesn't know what the computer is requesting or displaying.

It is easy to "build in" messages in INPUT and PRINT commands, so that "questions" are asked, or "answers" are printed. Here are some examples of how the computer can be programmed to provide this extra information:

```
INPUT "ENTER STRING"; TY$
PRINT "EGGS SOLD = " EGGS
INPUT "NUMBER OF STUDENTS"; NS
PRINT "NAME FOUND IS " NAME$
```

In each of these examples, a message has been put into the command. These messages are enclosed in quotes, and they have absolutely no effect on the INPUT or PRINT operations. The computer simply puts the information between the quote marks on the TV screen as an aid. You can put almost anything you want in these messages to make the computer easier to use.

There are just a few things to remember when these types of messages are to be used. First, *the message must be placed beween quote marks*. Second, for INPUT commands, there must be a semicolon (;) between the message and the label. A semicolon is

not used for the PRINT commands. The INPUT and PRINT commands shown above with messages in them follow these rules.

Let's look at an example. If the following INPUT command is used in a program:

```
INPUT "TYPE YOUR NAME"; NAME$
```

the computer will print, TYPE YOUR NAME, on the TV screen as part of the keyboard input process. Now, you know what you are supposed to type in when you see the flashing cursor on the screen. In fact, it will look like this:

```
TYPE YOUR NAME? ■
```

The question mark is placed right after NAME, and the cursor shows you that the computer is ready to accept information from the keyboard. Using the messages in INPUT commands certainly makes the computer easier to use. The same thing can be said for the PRINT commands, since messages are used to tell you what is being displayed on the screen.

You can also put messages in PRINT commands by themselves. They are used simply to print information on the TV screen. No labels are used with these PRINT commands, since they don't print anything but the "message." Here is an example:

```
100 PRINT "CHECK ACCOUNT PROGRAM"
120 PRINT "COPYRIGHT 1983"
140 PRINT "ELIZABETH SCOTT"
```

If these lines are used in a program, they will print the information between the quotes on the TV screen to identify the program and its author. In the TYPE YOUR NAME example shown previously, you might want to remind the user to press the RETURN key when entering information. You could put a message in several PRINT commands to print this reminder on the screen at the start of the program:

```
10 PRINT "WHEN ENTERING YOUR"
20 PRINT "INFORMATION TO THE"
30 PRINT "COMPUTER, DON'T"
40 PRINT "FORGET TO PRESS THE"
50 PRINT "      RETURN      "
60 PRINT "KEY WHEN YOU ARE"
70 PRINT "FINISHED."
```

In the fifth command, some spaces have been used so that the word RETURN will be printed toward the center of the screen to emphasize it. You can also print other things, such as:

```
100 PRINT "******************"
120 PRINT "* TEST PROGRAM *"
140 PRINT "******************"
```

In later chapters, we will tell you about some of the other interesting things that you can do with the PRINT commands.

Experiment No. 2-7. Printing Messages

Since messages are helpful to computer users, they are important parts of almost every computer program. Now you will see how they can be used in a practical program.

Type in the following program and use the LIST command to check it:

```
10 PRINT "STRING INPUT TEST"
20 INPUT "TYPE YOUR NAME"; NAME$
30 PRINT "SO YOU'RE " NAME$
```

Now run the program. Don't forget to press the RETURN key after typing your name. What is displayed on the TV screen? You should see something like this:

```
STRING INPUT TEST
TYPE YOUR NAME? JIMMY
SO YOU'RE JIMMY

READY.
■
```

Change line 30 to:

```
30 PRINT "SO YOU'RE" NAME$
```

Be sure that the closing quote mark comes right after the E in YOU'RE. Run the program and type in a name. What does the display look like? It should look something like this:

```
SO YOU'REJIMMY
```

The extra space at the end of YOU'RE in the message is used to space the message and the string. You can use extra spaces to center and align information in the messages on the TV screen.

The use of messages in PRINT and INPUT commands makes programs meaningful and easy to use. Without using messages, it is almost impossible to know what you are supposed to do with a program, or how to use it. In the programs that will be discussed and developed in the following chapters, you'll see that messages are used extensively.

CONCLUSION

You have seen quite a bit of new information about computers in this chapter, particularly if this is your first time using a small computer. You have seen that computers can process information and transfer it in and out. The VIC 20 computer can process both values and strings, and in this chapter you saw how labels can be used to identify information that the computer is to use. Labels were used to distinguish between actual values to be processed and strings of characters.

Although no information was "processed" in any of the experiments, you learned how INPUT and PRINT commands can be used to tell the computer to get information from the keyboard and to display, or print, it on the TV screen. You also found that the VIC 20 uses only the first two letters in a label, and that it can be confused by labels such as BEACH and BEETS, since the first two letters are the same. In the last section, you found that the INPUT and PRINT commands can be expanded to include messages that can be printed on the TV screen so that the input and output processes give the user guides as to what is happening.

In the next chapter, we will introduce you to the actual processing of information within the computer, and we will also tell you how you can control the operation of the computer. Some new keyboard operations will be introduced, too.

QUESTIONS

Here are some questions that you can answer to check your understanding of the ideas presented in this chapter. The answers are provided in the back of the book.

1. What types of information can be used with the VIC 20?

2. What are the two general types of operations that all computers perform?

3. Which of the following would be considered valid labels for information in the VIC 20 computer?

LABEL	$LABEL	DO NUTS	SPAGHETTI
TYPES$	31ST STREET	A1TESTS	EGGS$
VIC$	SPEND$	3STATE$	SPELLER

4. If the valid labels in the above list were all used in one computer program, would there be any conflicts? If there are conflicts, what are they?

5. Find the errors in the following labels or information. What are they?

   (a) TEST = $56
   (b) TIME = "TEN OF FIVE"$
   (c) TM = 56.89
   (d) LINK$ = NUMBER OF TESTS
   (e) SEEDS = 78
   (f) LINES = "10 LINES PER INCH

6. The following PRINT and INPUT commands have messages in them. Which ones are incorrect? What is wrong with them?

   (a) INPUT "YOUR NAME, PLEASE", NAME$
   (b) PRINT "TEST RESULTS
   (c) PRINT "BALANCE IN ACCT ="; BAL
   (d) INPUT " TODAY'S SPECIAL "
   (e) INPUT "ENTER THE YEAR"; YR$
   (f) PRINT "THIS IS A VIC 20"

7. If you enter a program line with the wrong line number, how do you "erase" it from your program?

8. If you make a typing error in entering a line in your program, how can you correct it?

9. Describe how you start a program once it has been typed into the computer.

10. Describe how to list a program that is already in your computer.

FOOD FOR THOUGHT

The following are provided as things to think about. You may not know the answers, but you can try and think out a logical answer for each. You can also try and run programs on the computer to see what happens.

1. If you run the following programs, what will happen?

   (a)
```
10 INPUT NAME$
20 PRINT NA
```

   (b)
```
10 PRINT NA
20 INPUT NA
```

   (c)
```
10 PRINT NA
20 INPUT NAMES$
```

(d)
```
10  INPUT "TEST VALUE"; VLU
20  PRINT VLU
30  PRINT VLU
40  PRINT "          " VLU
```

2. If the labels TEAM SPORTS, TEAMS, and TEETH$ are all used in a program, will there be any conflicts? Between which ones? Why?

3

Number Crunching

In the previous chapter you learned how the computer keeps track of information by assigning labels to it. You also learned that the VIC 20 computer can operate on numbers, or values, as well as work with strings of characters. In this chapter, we will concentrate on how the computer uses values, and you'll see how to make the computer do simple arithmetic operations such as addition and subtraction. You will also find out how the computer can do several things at once, and how programs can be made more efficient.

SIMPLE ARITHMETIC

The VIC 20 can do many interesting things with numbers, but in this chapter, we'll concentrate on the simplest arithmetic or math operations; addition, subtraction, multiplication, and division. Your VIC 20 can do many other complicated things with numbers and values, but we'll leave these for later. The four simple operations are a good place to start.

The four basic math operations are very common and we use them frequently without even thinking about them. You can quickly add a few prices to see if you have enough money, and you can multiply the kids at a birthday party by the cookies each will eat to see if you have enough. Even though you don't think about it, you're doing simple, and even complicated math in your head.

Unfortunately, a computer can't just run off a simple calculation without first having been "told" exactly what to do. This is called "programming" the computer, since you set up a step-by-step program of things for the computer to do. Computers don't know anything about the cost of groceries or how many cookies kids eat at parties, so every piece of needed information must be put into the computer right at the start.

SETTING UP THE PROBLEM

Here are a few examples that show how simple math problems can be set up. You can probably think of many others:

235	290	30	4782
+68	−79	×9	71

In these problems, the two values are put one over the other, but you could just as easily have written them on a line like this:

235 + 68 = ? 290 − 79 = ? 30 × 9 = ? 4782 / 71 = ?

It may not be easy for you to solve the second set of problems in your head, or on paper, since the columns of digits aren't lined up.

In the VIC 20, math problems are set up in almost the same way. There are two slight differences; (1) a label is used to identify the "answer," and (2) the problems are "reversed," with the label for the answer on the left side. Here's an example:

```
SUM = 43 + 3421
```

When the computer finishes this addition problem, it gives the label "SUM" to the answer, which is 3464. Remember that the word used as the label, SUM, is not equal to 3464. The computer simply needs a label so it can identify the anwer if you want to use it sometime later. There is nothing special about the label, SUM. You can choose almost any label you like; ANSWER, RESULT, Q9, or something else. Remember, the VIC 20 uses only the first two letters, or letter and number, in a label.

Now that you have a way of identifying the answer from the simple addition problem, what can you do with it? Why not use the computer to display the answer on the TV screen? A PRINT command can be used to do this, and you probably remember from Chapter 2 that it would look something like this:

```
PRINT SUM
```

By using both a math and a PRINT operation, you can put together a simple program for the VIC 20. Here it is:

```
50 SUM = 43 + 3421
60 PRINT SUM
```

Remember that line numbers are *always* used in BASIC programs for the VIC 20.

A NEW OPERATION

Before you do the first experiment in this chapter, we want you to know about a *new* command that will save you time when you want to clear the computer and get it ready for a new experiment or program. In a few of the experiments, you were told to turn your computer off and to turn it back on again. This "wiped out," or erased, old programs, clearing out the computer so a new program could be used. Whenever the computer's power is turned off, your BASIC-language programs are lost. Although this is a way of clearing the computer for new programs, it isn't really practical.

You can clear the computer for a new program by typing the word NEW and pressing the return key, [RETURN]. The NEW command completely clears the computer, so once you have used it, you cannot get back programs that have been erased. You'll have to retype your program if you accidentally use the NEW command.

Experiment No. 3-1. A Simple Math Program

This experiment will show you how a simple math problem can be set up and programmed into your VIC 20 computer. You'll also run the program to see what it does. This experiment will help break down some of the mystery of how computers are programmed.

With the computer turned on, type NEW [RETURN]. What happens? The computer just responds with the READY message to let you know that it is ready for your next command, or for a new program.

Type in the following short program. The plus sign is in the upper row of keys, near the right side of the keyboard.

```
50 SUM = 1590 + 398
60 PRINT SUM
```

List the program on the TV screen to be sure that you have entered it correctly. Just type the word LIST and press the RETURN key. If there are any errors in your program, retype the corrected lines. Don't forget to press the RETURN key at the end of each line.

Now run the program. (Remember, you just type the word RUN and press the RETURN key.) What is displayed on your TV screen? If you listed the program to check it, and then ran it, you will probably see something like this:

```
LIST
50  SUM = 1590 + 398
60  PRINT SUM
READY.
RUN
  1988
```

The result of the addition should be displayed on the TV screen (1988). Is this what you found? If your program is working properly, the result should be displayed. List your program again to be sure it's still in the computer. Now, type NEW [RETURN]. What is displayed when you list the program again? The program is no longer in the computer, since the NEW command told the VIC 20 to "erase" it and get ready for a *new* program. You can use the NEW command to completely "erase" old programs.

Experiment No. 3-2. More Math Fun

The program in this experiment will show you how the other three math operations can be used to do a subtraction, a multiplication, and a division problem.

Type NEW [RETURN] to clear the computer for this experiment. Load the following program into your computer. It is the same type of program you used in the previous experiment.

```
50  QX = 1590 + 398
60  PRINT QX
```

Use the LIST operation to list the program on the TV screen so you can check it to be sure it is correct. How would you substitute this subtraction problem for the addition problem used in the program?

```
371 - 83 = ?
```

The following program line could be used:

```
50  QX = 371 - 83
```

The same label, QX, is used, even though a subtraction is being done. This means that the label can be almost anything you choose. You could choose another label, for example, CT:

```
50  CT = 371 - 83
```

but you would have to change the PRINT part of the program so that it uses the same label, too. Just use QX as your label, since it saves time.

Put the following program line in your program. Just type it in. It will replace the "old" line 50 in the program:

```
50  QX = 371 - 83
```

Run the program. Is the correct answer displayed? What is it? The correct answer, 288, should be displayed on the TV screen. Now put the following program line in your program:

```
50  ZIP = 371 - 83
```

What do you think will happen if you run the program now? Go ahead and run the program and see what is displayed on the screen. Is this what you expected? The answer displayed is zero. Is that the result of 371 - 83? Something must be wrong. List your program and check it. It should look like this:

```
50  ZIP = 371 - 83
60  PRINT QX
```

In this program, the result of the subtraction is labeled ZIP, while the PRINT command prints the value that has been labeled QX. Two *different* labels are used, one for the subtraction and one for the PRINT command. When you tell the VIC 20 to RUN a program, it first clears all the labels to zero. Since QX hasn't had a new value assigned to it, the PRINT QX command at line 60 just prints the zero on the TV screen. To correct the program, simply type in:

```
60  PRINT ZIP
```

Now run the program. Is the answer from the subtraction displayed correctly? It should be. Now change program line 60 back to:

```
60 PRINT QX
```

and continue with the experiment.

Let's try multiplication and division. The computer uses a slash mark (/) for division and an asterisk (*) for multiplication. Use a two-line program to solve each of the following problems, one at a time. Use the label QX for the answer to each problem.

```
360 / 15        (Answer: 24)

34 * 56         (Answer: 1904)
```

The following programs can be used, one at a time. For the division problem, you would use:

```
50 QX = 360 / 15
60 PRINT QX
```

And for the multiplication problem, you would use:

```
50 QX = 34 * 56
60 PRINT QX
```

Suppose that you wanted the computer to solve a different multiplication problem. How would you get the new numbers into the program so that the computer could use them? Right now, the only way would be to put new instructions at line 50 in the program. For example, if you wanted to multiply 58 by 1295, you would have to change line 50 to:

```
50 QX = 58 * 1295
```

and you would have to run the program again to get the answer displayed on the TV screen.

As you saw in these experiments, getting the computer to do simple math problems isn't difficult, and the computer can do them quickly. However, it is going to be difficult to use the computer if you have to change the *program* whenever you want to do a math operation with new values. Can you think of a way to get around this?

USING LABELS IN MATH PROBLEMS

You may not remember it, but two values were multiplied in one of the examples in Chapter 2. Here is how the multiplication looked:

```
EGGS = AMOUNT PER CARTON * CARTONS
```

In this example, *labels* have been used instead of the actual values. The computer is being told to get the value labeled AMOUNT PER CARTON and multiply it by the value labeled CARTONS. When the values have been multiplied, the computer labels the answer EGGS.

No matter what, *the number of eggs will always be found by multiplying the number in each carton by the number of cartons.* Even though the numbers may change from day to day, the problem is always solved the same way. All of the math operations can use labels the same way. In the following examples, the problems are always solved the same way, but different values can be used:

```
MONEY = SAVINGS + CHECKING
PAY = WAGE - TAXES
DAYS = HOURS / HRS PER DAY
COST = CAKES * PRICE PER CAKE
```

In each of these examples, the labels were chosen with care so that there would not be conflicts between the first two letters in each label. Remember that the VIC 20 only "sees" or uses the first two letters in a label. So, the preceding examples are actually "seen" by the computer as:

```
MO = SA + CH
PA = WA - TA
DA = HO / HR
CO = CA * PR
```

When you use labels in a problem, you face another job. How do you get your values into the computer so the problem can be solved? The INPUT command is used in the program to "ask" for these values, so they can be typed in from the keyboard.

GETTING YOUR VALUES—THE INPUT COMMAND

It is easy to use the INPUT command to "ask" for values from the keyboard. Now a simple multiplication program can be set up, and it looks like this:

```
30 INPUT A
40 INPUT B
50 ANS = A * B
60 PRINT ANS
```

The two numbers are input at lines 30 and 40. One is labeled A and the other is labeled B. These values are multiplied at line 50, and the answer, ANS, is printed on the TV screen by the PRINT command at line 60. Here's what it looks like when it is run:

```
RUN
? 34
? 12
 408

READY.
■
```

This program isn't particularly useful, unless you just want to sit and fool with the computer. Here is a useful program that uses a multiplication operation:

```
30  INPUT HOURS
40  INPUT WAGE
50  PAY = HOURS * WAGE
60  PRINT PAY
```

When you run this program and type in two values, it looks like this on the TV screen:

```
RUN
? 34
? 12
 408

READY.
■
```

You can probably see that *the two sample programs do exactly the same thing*; input two numbers, multiply them, and display the answer. The only difference is that in the second example, meaningful labels have been used so you can see what is being calculated. However, the *computer programmer* is the only one who sees these labels; the *computer user* doesn't. It would be much more meangingful to use "messages" in the INPUT and PRINT commands so that the information to be input and displayed is meaningful to the *operator*. Here is a better program:

```
30  INPUT "HOURS WORKED"; HOURS
40  INPUT "HOURLY WAGE"; WAGE
```

```
50  PAY = HOURS * WAGE
60  PRINT "WEEKLY PAY IS $" PAY
```

If you run this program, here's what is displayed on the TV screen:

```
RUN
HOURS WORKED? 40
HOURLY WAGE? 5.50
WEEKLY PAY IS $ 220

READY.
■
```

Experiment No. 3-3. Your Calculator

Although you won't be writing long programs in this book, you will see how to write a simple one in this experiment. It is a math program that will add, subtract, multiply, and divide two numbers, all at the same time. At least it will seem to be happening at the same time. Type NEW and [RETURN] to clear the computer for this experiment.

Two numbers are to be input to the computer. The first number entered will be labeled A and the second number entered will be labeled B. Use INPUT commands and "messages" to get the user to enter the two values from the keyboard. Write your program lines below, starting with line 10:

10 _____

20 _____

What do your program lines look like? They should look something like this, but the message you put between the quote marks may be different:

```
10  INPUT "FIRST NUMBER"; A
20  INPUT "SECOND NUMBER"; B
```

Now write four program lines to calculate the following: S = A + B, T = A − B, Q = A * B, and R = A / B.

30 _____

40 _____

50 _____

60 _____

Your program lines should look like this, although they may be in a different sequence:

```
30 S = A + B
40 T = A - B
50 Q = A * B
60 R = A / B
```

Now, write four more program lines to print the results. Use "messages" so that the user will know which operation was performed:

```
70 _____
80 _____
90 _____
100 _____
```

Your complete program should now look like this:

```
10 INPUT "FIRST NUMBER"; A
20 INPUT "SECOND NUMBER"; B
30 S = A + B
40 T = A - B
50 Q = A * B
60 R = A / B
70 PRINT "A+B =" S
80 PRINT "A-B =" T
90 PRINT "A*B =" Q
100 PRINT "A/B =" R
```

Check the program lines you wrote in the spaces provided and make any corrections needed. Now enter your program, list it, and correct any errors. You may use any meaningful messages you like between the quotes.

Run your program and check to see if it operates as you would expect. You may want to use a calculator to check the results quickly.

In this experiment, you wrote a simple program to do math with two numbers. Although the program wasn't too complicated, it did show you several things:

1. The computer can do math operations quickly.
2. You can enter numbers from the keyboard using an INPUT command and a label.

3. Problems can be set up using labels.
4. Messages are useful in telling the user what is to be entered or what has been displayed.
5. Programs can be easy to write.

ONE LINE FOR MANY

In the math experiments and examples, a program line was used for each operation. Sometimes it is easier to write all of these operations on a single program line with a single line number. This can simplify a program and, in some cases, make it more efficient. You can put several operations on a single line, separating them with a colon (:). The VIC 20 "sees" the colon as separating each of the operations, so it does each one in turn, from left to right. You will probably see programs that use this technique. Here are some examples:

```
100  RATE = MILES/HRS : PRINT RATE
```

```
 75  S=TR+QW : INPUT "VALUE";VA : QX = VA + S
```

In the first example, two operations, a division and a print, were put in a single line. In the second example, three operations were put in a line. Extra spaces have been used in these examples so you can clearly see the positions of the colons. Some programmers try and squash everything into as little space as possible, writing a program line like this:

```
75S=TR+QW:INPUT"VALUE";VA:QW=VA+S
```

Not very easy to read, is it? The use of colons sometimes makes programming a bit easier, but it may be difficult to make simple changes in long program lines. There are some special keyboard operations that you can use to change program lines, and you'll read about them shortly.

Experiment No. 3-4. More On a Line

This experiment will show you how colons are used in program lines. You will be able to put several operations on a single program line.

Type in NEW [RETURN] to clear out any remaining programs or program lines. You can use the LIST operation to be sure that the computer has been "cleaned-out" for your new program. Type in the following program:

```
10 INPUT AZ
20 DX = AZ * 80
30 PRINT DX
```

To keep the program simple, no messages are used in the INPUT or PRINT commands. List the program and check it to be sure that it has been correctly entered. When you are sure it has been typed in correctly, run the program. What does it do? Does it operate properly? The program multiplies the value you type in by 80 and displays the result. Can you suggest how the complete program could be put on a single line? Use the space provided:

```
10 _____
```

The following program line will work properly:

```
10 INPUT AZ : DX = AZ*80 : PRINT DX
```

Type NEW [RETURN] to clear the computer for this one-line program, and then type in this program line. Since the complete program line cannot fit on one line on the TV screen display, it "wraps around," looking like this on the TV screen:

```
10 INPUT AZ : DX = AZ
* 80 : PRINT DX
```

The VIC 20 just continues the *program line* on the next line. The second line is not another program line, it's just a continuation of program line 10.

Run the new program. Does it give the same results as the three-line program you used earlier in the experiment? It should, since the same things are being done.

As you read this book, you will find some programs in which several operations are combined on a single line, using colons to separate each one. When program lines are described, remember that the program lines may have more characters than can fit on a single "line" on the TV screen. These lines will simply end at the right side of the screen and start up again at the left side of the following line.

DOING MORE THAN ONE THING AT A TIME

In many programs, the computer does only one math operation per line. There are times when it is useful to *combine* operations so that several are done at the same time. This means that several

operations can be "bundled" into one. When the computer is doing many things, this type of "bundling" saves computer time. Here is an example:

In the egg problem described before, the number of eggs sold was simply the number of cartons sold times the number of eggs in each carton. That's pretty simple, but let's assume that some eggs are also sold from a basket where customers can help themselves. Now we have to figure out the number of eggs sold in cartons and then add in the loose eggs sold from the basket. It can be done like this:

```
EC = EGGS PER CARTON * CARTONS SOLD
EL = LOOSE EGGS SOLD
TOTAL = EC + EL
```

In this example, EC labels the eggs sold in cartons and EL labels the loose eggs sold. The total number sold is just EC plus EL. The multiplication and addition operations could be put on a single program line and separated by a colon, but nothing much is gained. It would be better if the operations could be combined. Let's try using some numbers in this problem so we can see what is happening. Here's the problem:

```
TOTAL  =  EGGS PER CARTON  * CARTONS SOLD +  LOOSE EGGS SOLD
TOTAL  =        12         *       3      +        4
```

When the math operations are shown this way, it's a bit confusing, since we don't know which operation should be done first. Does the computer first multiply 12 and 3 and then add 4, or does it add 3 and 4 and then multiply the sum by 12? Which is it?

```
TOTAL = 12 * 7 = 84
```

or

```
TOTAL = 36 + 4 = 40
```

As you can see, the answers are different, depending on whether the addition or the multiplication is done first. Actually, this doesn't confuse the computer, since it follows two rules when doing math. here they are:

1. Multiplications and divisions are done first.
2. Subtractions and additions are done next.

This problem can be solved just as it is written, since the EGGS PER CARTON will be multiplied by the CARTONS SOLD and

then the LOOSE EGGS SOLD will be added in. Different math operations can be separated by using parentheses, and they can help you see what is happening in a complicated math problem. That's the next subject.

USING PARENTHESES IN MATH PROBLEMS

In complex math operations, you can use sets of parentheses to "contain" individual math operations. The egg example can be written as:

```
TOTAL = EG * CA + EL
```

or as:

```
TOTAL = (EG * CA) + EL
```

In this problem, the computer first does the math operation enclosed in the parentheses, then it does the other operation. When parentheses are used, there is no doubt about which operation is done first.

You will see parentheses used frequently in BASIC computer programs. Just remember that the "problem" in the inner set of parentheses is solved first. After that has been solved, the computer moves outward, solving the "problem" placed between the next set of parentheses, and so on. Here's an example of how parentheses are used:

```
? = (3 + ((9 * 4) / 12))
```

Go to the inside set of parentheses and multiply 9 and 4, which is 36. Remove the parentheses surrounding this part of the problem and you have:

```
? = (3 + (36 / 12))
```

Now go to the inside set of parentheses and divide 36 by 12, which is 3. Remove the parentheses surrounding this part of the problem and you have:

```
? = (3 + 3)
```

Just add 3 and 3, and you've solved the problem.

If you look at a problem that uses parentheses, how can you tell which set is the inner set? Let's look at the problem that we just went through:

```
? = (3 + ((9 * 4) / 12))
```

Use your finger or a pencil as a marker and move it from left to right across the problem line. Look at the parentheses as they pass your pointer. We've left out the math operations, but you'll first pass three parentheses that look like this: (((. When you reach one that curves in the opposite direction: (((9 * 4), you've reached the right side of the inner set. Solve the 9 * 4 problem between this set of parentheses. You can now "erase" or mark-off this inner set of parentheses and start over again looking for another "inner" set of parentheses. Just continue to do this until all the parentheses have been removed. Then, the problem has been solved.

This may sound complicated, but you can see in the example in Fig. 3-1 that it's not difficult to "break down" the parentheses so that the problem can be solved. In this example, only five steps were needed to get the answer.

Use paper and pencil to figure out the answers to each of the sample problems shown below:

```
A = (9 + (5 * 3))
B = ((6 / 2) * 12)
C = ((4 * 5) + 4) / 3
D = (((5 * 8)/4) + 10) * 5
```

Here are the answers: A = 24, B = 36, C = 8, D = 100. How did you do? You can check your answers on the VIC 20 by simply typing in:

```
PRINT (9 + (5 * 3)) [RETURN]
```

and so on for the other problems. This is using the VIC 20 as a *calculator* and it lets you do simple math problems quickly. You DO NOT use a line number, so the problem is not saved in the computer for later use. It is solved immediately for you. Use the VIC 20 as a calculator and check your answers to the preceding

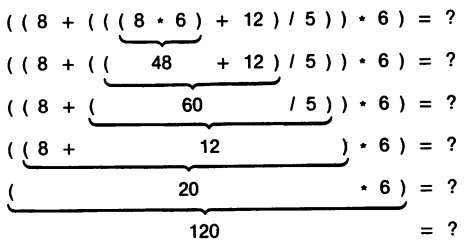


Fig. 3-1. Breaking down parentheses in a math problem.

problems. You should get A = 24, B = 36, C = 8, and D = 100. Of course, the VIC 20 won't print the "A =" part of the answer, it just displays the value of the answer.

WHAT'S GOING ON HERE?

Although there are more interesting things to learn about in this chapter, we want to tell you about a few things that will make your programming or computer-using experiences better. If you write computer programs, you need to keep track of what you do. We recommend that you buy and use a spiral-bound notebook in which you can write all of your programs, what goes wrong, what works, and so on. Make the notebook readable so you can go back and see what you've done and where you are, when you're working on a program. Even when you are using programs prepared by others, take notes about problems, times when the program wouldn't run, interesting results, and so on. Notes are important, and keeping them in one place will be helpful.

Good programmers also put helpful notes and comments right in their programs. The computer ignores these remarks and doesn't even "know" they are there. When you *run* a program that contains this information, it is not printed on the TV screen, and can only be read when listing a program on the TV or printer. If you decide to go back and work on a program sometime later, remarks can be useful in telling you what you've done. It is often difficult enough to remember why you did something in a program, but without the frequent use of remarks, it's almost impossible to reconstruct your "trail."

To put remarks in your programs, you simply type in a line number where you want the remark to be put, then type in REM, for remark, followed by a line of remarks or comments. You can use as many REM statements in a program as you like. Here are some examples from different programs:

```
290 REM  FEBRUARY 1983 VERSION
 20 REM  TAX CALCULATION PROGRAM
450 REM  THIS SETS UP SPECIAL VALUES
```

You can also put remarks in program lines, along with other operations. Just remember to separate the operations with a colon:

```
1090 TX=RT*GS: REM TAX = RATE * GROSS SALES
```

Good programmers make frequent use of REM lines in a program so that other programmers looking at the *program listing*

will have a pretty good idea of what is going on. Some of the program examples in this book will use REM commands to tell you what's going on, if it's not explained in the text. We recommend that you use REM comment or remark lines in BASIC-language programs you write.

YOUR EFFICIENT SECRETARY

In past experiments, you were told to make corrections in program lines by simply retyping the entire line. When complex program lines are used, there is a good chance that you will make even more mistakes when trying to correctly retype a line. To help you correct errors or make changes in programs, a special key is included on the keyboard. This is the INST/DEL key that is located in the upper right-hand corner. The INST stands for insert and the DEL stands for delete. You can use this key along with the SHIFT key to make program changes, even in complex program lines.

Rather than tell you how to use the key, we will show you how to use it in the next experiment.

Experiment No. 3-5. The INST/DEL Key

The purpose of this experiment is to have you investigate the use of the INST/DEL key and see how it can be used to make changes in programs.

Turn your computer on. Type in NEW [RETURN] to clear out any old programs and then type in the following program line:

```
10 PRINT " YOU PEST "
```

You should be able to list your programs without help. If your program is incorrect, retype the line and check it again. Run the program and be sure that YOU PEST is displayed on the TV screen.

We now want to change the program so that YOUR TEST is displayed when the program is run. To make the changes, first list the program on the TV screen. Then, use the cursor control keys at the bottom right-hand corner of the keyboard to position the cursor so it is right on top of the E in PEST. This means that the cursor will surround and flash the E character on the TV screen. Remember to use the SHIFT key with the CRSR keys to move the cursor in the directions shown on the top of the two cursor keys. If you have difficulty, simply clear the screen and type in

LIST again, to get a new listing. You can clear the screen by press/hold the SHIFT key and pressing the CLR/HOME key.

When the cursor is at the E, press the DEL key once. What happens? Press the DEL key again. What happens now? When the cursor is positioned at the E and the DEL key is pressed once, the P in PEST is removed and the letters EST move one space to the left. When the DEL key is pressed again, the space between YOU and EST is removed, leaving:

```
10 PRINT " YOUEST "
```

Two characters were removed from the line. Remember that the computer "sees" the space as a character, too. The cursor is still at the E.

Press the RETURN key. Now, clear the screen and list your program again. It should look like this:

```
10 PRINT " YOUEST "
```

Run the program. What is displayed on the TV screen? You should see YOUEST displayed.

Can you generally describe what the DEL key does? When the DEL key is pressed, the character just to its left is "erased," and the character under the cursor and those to its right move one space to the left, taking up the space of the "erased" character. The DEL key can be used to erase characters from anywhere in a program line.

When you ran the program, YOUEST was displayed on the TV screen. Since you want to display YOUR TEST, you need to insert three characters in the message part of the print command; R, space, and T.

List your program again and use the cursor keys to position the cursor at the E in YOUEST. The cursor will flash the E on the screen when it is in position. You can use the INST key and the SHIFT key to "open" three spaces in YOUEST for the needed characters. With the cursor positioned at the E, press and hold the SHIFT key and then press the INST/DEL key three times. What happens? When the SHIFT key is pressed and held, spaces are inserted in YOUEST each time the INST/DEL key is pressed. Now YOUEST has become YOU EST. The cursor remains in place, "pushing" the characters under it to the right. Remember, a space is a character, too. The program now looks like this:

```
10 PRINT " YOU■ EST "
```

Since the cursor is next to the U in YOU EST, type in an R, a space, and a T. Now press [RETURN]. What does the program look like? It should look like this:

```
10 PRINT " YOUR TEST "
```

Clear the screen and list the program to be sure the changes have been made. Then, run it to be sure it prints YOUR TEST as you would like it to.

The insert/delete key (INST/DEL) is very useful in helping you make changes in programs. To make changes, simply list the program and position the cursor where it is needed. To delete characters, simply press the DEL key and the cursor will "erase" characters to its left. *The changes made with the INST/DEL key only go into effect when you press the RETURN key.*

To make additions, again place the cursor where it's needed and press the INST key as many times as there are characters to be inserted. Remember that the cursor moves the character under it to the right, "opening" spaces for new characters. Remember, too, that spaces are characters.

When changes have been made, remember to press the RETURN key to tell the computer that you have completed the changes. It is a good idea to list the changed program again to check and be sure the changes are as they should be.

You can also use the DEL key as an "eraser" to "back up" the cursor and make changes in a program line as you are typing it. This is useful if you catch a mistake before pressing [RETURN] at the end of a command or program line. Once you press the RETURN key, you'll have to use the cursor and INST/DEL keys to make the correction, or retype the complete line.

For example, you want to list a program and you type LISR instead of LIST. If you catch the error before pressing the RETURN key, you can use the INST/DEL key to back up and "erase" the R in LISR. You can then type in the T at the end, so LIST is spelled correctly. Press the RETURN key and the program will be listed for you.

It takes a while to get used to the operation of the insert and delete keys. We suggest that you go through this experiment at least one more time so you become familiar with the operation of this useful key. As with the cursor keys, you must use the SHIFT key to cause the INST operation to take place.

QUESTIONS

1. What basic math operations can the VIC 20 do?

2. What math operations does the VIC 20 do first, and which does it do second?

3. How many program steps can you put in a single program line?

4. How are individual program steps separated when they are used in a single program line?

5. What are parentheses used for in math program steps?

6. What is the answer to each of the following math operations:

   (a) ? = 5 * 12 + 4
   (b) ? = 3 * 4 + 12 / 5 * 8
   (c) ? = ((((4 * 6) / 3) + 18) −7)

7. What does REM stand for and how is it used in a BASIC program?

8. What operation lets you put values into the computer from the keyboard?

9. How does the computer identify information?

10. What does the INST/DEL key do?

PROBLEMS

1. Write a program that lets you enter three numbers. The first two numbers are added together and the result is multiplied by the third number. The result is then displayed.
2. Using the formula $C=5/9*(F-32)$, write a program that lets you enter a temperature in degrees Fahrenheit (F) and displays the equivalent temperature in degrees Celsius (C). If you enter 68 (F), a 20 (C) should be displayed.
3. Write a program that lets you convert dollars into Swiss francs. Assume that there are 2.5 Swiss francs to the dollar (each franc is worth $0.40 or 40 cents). You should be able to enter any amount in dollars (such as 5.00) and the equivalent number of Swiss francs (12.5) should be displayed.
4. If the speed of light is 186,000 miles per second, how many seconds are required for light to travel 1 mile? How many seconds are required for light to travel 1 foot (assume that there are 5280 feet in a mile)?
5. If a 3-pound sack of apples costs $1.29, and a 5-pound bag of apples costs $2.20, which costs less per pound, the sack or the bag of apples?
6. Write a general-purpose program that lets you enter two prices and two weights for something that you want to purchase, and have the computer determine the cost per unit of measure, so that you can determine the less expensive item. For example, if a 24-ounce box of detergent costs $2.29 and a 52-ounce box costs $4.89, which is the better buy?
7. If a car travels for 2 hours and 20 minutes, and goes 120 miles, what is the average speed of the car in miles/hour? If 3.8 gallons of gasoline were used by the car, how many miles/gallon is the car getting?

8. If 14 aluminum drink cans weigh one pound, and you can sell them at the rate of 5 pounds for $1.28, how many cans do you need to earn $10.00?

9. Write a program that lets you enter the number of drink cans that you have, and calculates the amount of money that you will receive, based on the rate of 14 cans/pound, and $1.28/5 pounds.

10. If a recipe calls for 2 eggs, one cup of milk, and 1⅔ cups of flour, how much of each ingredient will be required if you have 5 eggs, and you want to use all of them in the recipe?

11. If an acre of land is a square, 208.71 feet on each side, how many square feet are there in an acre? How many acres are there in fields that are 425 feet × 1023 feet, 500 feet × 712 feet, and 3028 feet × 1120 feet?

12. If a farmer has a field in the shape of a circle, and the diameter of the field is 345 feet, how many acres of land are in the field? To calculate the area of a circle, use the formula 0.7854*D*D, where D is the diameter of the circle. There are 43,560 square feet per acre.

13. If the shortest day of the year has 7 hours and 20 minutes of sunlight, and the longest day has 14 hours and 52 minutes of sunlight, on the average, how many additional minutes of sunlight do we get each day, as we go from the shortest day (December 21 in the northern hemisphere) to the longest day (June 21 in the northern hemisphere)? Assume that there are 182 days between the shortest and the longest days.

14. A bank pays you 8.5% interest per year on your savings account. Thus, if you had $100.00 at the beginning of the year, you would have $108.50 at the end of the year. If you have $524.30, how much will you have at the end of the year (assume that no money is withdrawn from the account)?

15. The bank mentioned in Problem 14 has changed its policy so that instead of paying you 8.5% interest each year, it will pay you 8.5/12% interest each month. Assuming that your $524.30 is in the bank for a year, how much money will you have in your account after a year? If you had your choice of having your interest "compounded" monthly or yearly, which would you prefer?

4

Making Decisions

A computer cannot make up its "mind" about anything; it must be programmed by someone to make specific decisions. A computer can, however, make these programmed decisions quickly and choose among many alternatives. When spacecraft landed on Mars, on-board computers made almost instantaneous decisions about firing the rocket engines, where to land, how fast to land, and so on (Fig. 4-1). Of course, the computer didn't just decide to go to Mars one day; *people* built the lander and programmed the computer to do just what it did.

Your VIC 20 can be told to do many things, but these must be preset in programs that are no more than lists of instructions. So far, all of the example and experimental programs have been simple. The computer has done some math and it has input and output information, but the programs have not altered the path that the computer took through the program. The computer went from one program line number to the next higher line number, one at a time. This is called a *straight-line* program, as opposed to a *branch* program, both of which are shown in Fig. 4-2.

THE GOTO COMMAND

An easy thing for you to do is to tell the computer to go back and do a task again and again. In a previous chapter, you saw a pro-

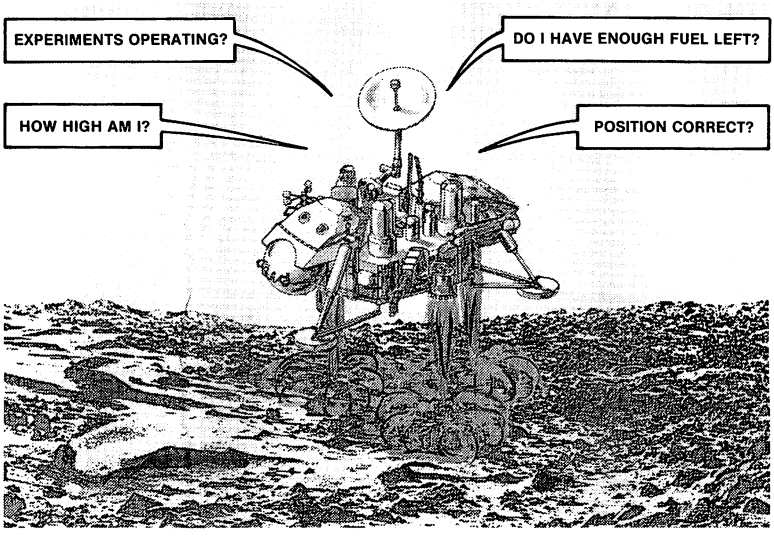


Fig. 4-1. Mars lander makes decisions quickly as it lands.

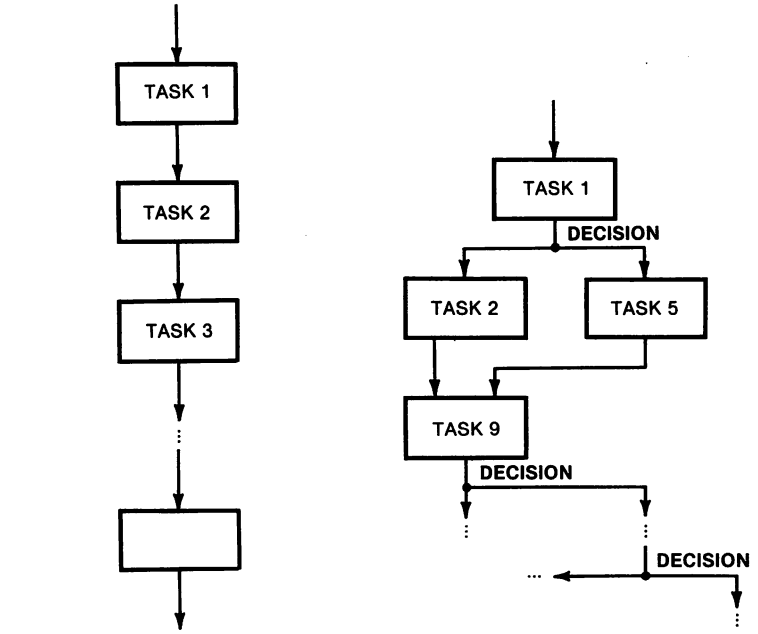
gram in which two values were multiplied to calculate a person's wage:

```
30  INPUT "HOURS WORKED"; HOURS
40  INPUT "HOURLY WAGE"; WAGE
50  PAY = HOURS * WAGE
60  PRINT "YOUR PAY THIS WEEK IS " PAY
```

The program is started by typing RUN. (We'll assume you remember to press the RETURN key, so we won't show it.) When the computer completes the calculation, it displays READY on the TV screen. To get it to run the same program again, for another employee, you would have to type RUN. This can be a bother, particularly if you have people waiting in line to get paid. It would be nice to simply tell the computer, "When you're finished, go back and do it again."

The GOTO command is used to tell the computer to go to a specific place in a program. The example in Fig. 4-3 shows how the GOTO command can be used to direct the computer to a specific line in a program. There are other program lines between the GOTO command at line 350 and the PRINT command at line 1200, but to make the example clearer, they have been left out.

When the computer reaches line 350 and reads the GOTO 1200



(A) Straight-line program. (B) Branching program.

Fig. 4-2. Comparison of straight-line and branching programs.

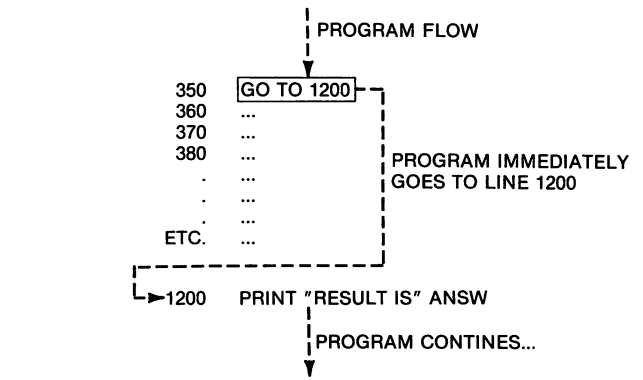


Fig. 4-3. Using the GOTO command to get to a specific line in a program.

command, it immediately goes to program line 1200 and starts work there, skipping over the program lines in between. In this example, the computer "jumps" from line 350 to line 1200 where it prints the message, RESULT IS, followed by the value labeled ANSW. The VIC 20 computer doesn't care what is done at line 1200, the GOTO command just tells it where to go. After completing the PRINT operation, the computer goes on to the next line that follows 1200.

Of course, if a GOTO 456 command is used in a program, the computer program *must contain line 456*, if the computer is to operate properly. For example, if you have only the following three-line program in the computer, what do you think the computer will do when it reaches the GOTO command at line 30?

```
10  INPUT V
20  PRINT V
30  GOTO 100
```

Since the program tells the computer to go to line 100, it will try to find line 100 in the program. Since there isn't a line numbered 100 in the program, an error message will be displayed on the TV screen:

```
RUN
? 45
 45

?UNDEF'D STATEMENT
 ERROR IN 30
READY.
```

The *undefined statement error* message tells you that something was undefined in line 30. In this case, the computer couldn't find line 100 in the program. There are other conditions that can cause errors, and the error messages for the VIC 20 computer are listed for you in Appendix A. Errors don't harm the computer, but they must be corrected if a program is to run properly.

By using the GOTO command and a line number, you can force the computer to go to any place in a program where there is a line number. This is a powerful addition to your list of BASIC instructions, since you can now cause the computer to do things again and again.

Here is a simple program that uses a GOTO command so you can see how it is used. Can you figure out what the computer will do?

```
10  INPUT "VALUE 1"; V1
20  INPUT "VALUE 2"; V2
30  SUM = V1 + V2
40  PRINT SUM
50  GOTO 10
```

Once the computer has printed the sum of the two values, V1 and V2, the GOTO command at line 50 points it back to line 10, and it continues working there. The computer does not "know" that it has been pointed back to the *start* of the program. It simply goes to the line number used in the GOTO command.

Here is what the TV screen would look like if this program were run:

```
VALUE 1? 5
VALUE 2? 6
 11
VALUE 1? 234
VALUE 2? 521
 755
```

and so on.

Here is an interesting variation on the VALUE program just shown. Can you see what is different?

```
10  INPUT "VALUE 1"; V1
20  INPUT "VALUE 2"; V2
30  PRINT V1 + V2
40  GOTO 10
```

The intermediate step of labeling the sum of V1 and V2 has been removed. In this program, the computer is told to print the result of V1 + V2 without labeling and saving the result. This type of operation can be used to speed up a program if a result is not going to be used somewhere else in the program. Since you only want to see the result, why have the computer take the time to label it and set it aside?

Here is another interesting program for you to think about:

```
100  Z = 0
110  PRINT Z
120  Z = Z + 1
130  GOTO 110
```

If you run it, this program will display numbers on the TV screen, starting with 0 and increasing by one; 0, 1, 2, 3, and so on.

It does it quickly, too. In this program, the GOTO command points the computer to program line 110 rather than to the start of the program.

Look at program line 120. This shows you another interesting way in which labels are used. The program line Z = Z + 1 simply says to get the value labeled with Z, add one to it, and then label the result Z. The program line Z = Z + 1 is NOT saying that Z *equals* Z plus 1. The label has been used to identify the starting value and it is then switched over to identify the result. The overall effect is to simply add one to the value labeled Z. This is done frequently in computer programs.

These computer programs are called *loops*, since the computer is forced to loop back and go through some of the same program steps again and again. Loops are important in computer programs, since they give you more control over what the computer is doing.

BREAKING OUT

Are there any problems with these "loop" programs? The big problem is that once the looping program is started, there is no way to get the computer out of it. The computer will complete its operations, go back and do them again, complete its operations, go back . . ., on and on and on. Is there a way to get the computer out of the loop? Actually, there are several ways, but only one is useful. Of course, you can always pull the plug, but that doesn't count.

The VIC 20 has a RUN/STOP key on the left side of the keyboard, shown in Fig. 4-4. When the computer is running a program, you can use this key to stop it. When the key is pressed, it will tell you where the computer was operating in the program by displaying a message such as:

```
BREAK IN 450
READY.
```

This tells you that you have caused the computer to break out of a program, and it gives you the line number where the computer was working. In this case, the computer was doing something at line 450 in the program when the RUN/STOP key was pressed.

Now that the computer has been stopped, the program can be listed, changed, corrected, and so on. You can also tell the computer to continue its operation by typing CONT [RETURN]. This tells the computer to continue on, even though you stopped it temporarily.

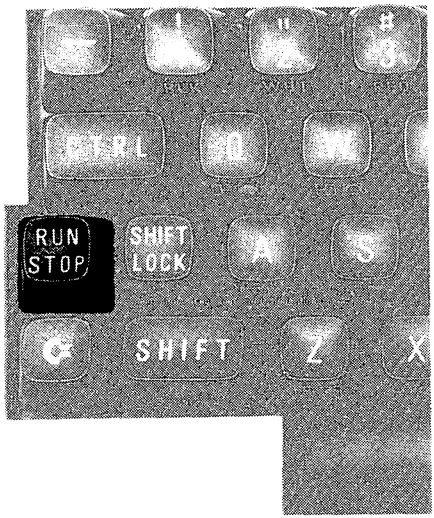


Fig. 4-4. Location of the RUN/STOP key on the left side of the keyboard.

You cannot use the RUN/STOP key to stop the computer if it is waiting for information from the keyboard; that is, if it is working on an INPUT command. Since the computer is waiting for information to be typed in, you must "reset" it. You can do this by pressing and holding the RUN/STOP key and then pressing the RESTORE key located on the right side of the keyboard, as shown in Fig. 4-5. When this is done, the READY message is displayed on the TV screen. If you stop a program by using the RUN/STOP and RESTORE keys together, you must type RUN to start it again.

When the RUN/STOP and RESTORE keys are pressed this way, the computer is reset, just as if you were restarting the program. Of course, the program is still in the computer ready to be used again. Remember that the RUN/STOP key must be pressed and held while the RESTORE key is pressed.

Experiment No. 4-1. Loops and Breaks

In this experiment you will see how you can use the RUN/STOP and RESTORE keys to break out of loops in computer programs.

Type NEW to clear out any programs in the computer. Type the following program into the computer and list it to be sure it is correct. Make any changes needed and then run it.



Fig. 4-5. Location of the RESTORE key on the right side of the keyboard.

```
100  X = 0
110  PRINT X
120  X = X + 1
130  GOTO 110
```

What is displayed on the TV screen? A column of numbers is displayed on the left side of the TV screen. The numbers seem to move up the side quickly, with new numbers being displayed in the bottom left corner of the display area.

While the program is running, press the RUN/STOP key. What happens? What is displayed on the screen? The numbers stop increasing and a message is displayed on the TV screen. The message should look like one of these:

```
BREAK IN 110
READY.
```

or

```
BREAK IN 120
READY.
```

or

```
BREAK IN 130
READY.
```

Why could three different messages be displayed? When you press the RUN/STOP key, it's impossible to know where the computer is in this program; it could be at line 110, 120, or 130. Run

the program again and stop it again. Just press the RUN/STOP key to stop it. Now what line number is shown in the BREAK message? Is it the same one?

Restart the program by typing RUN. Press the RUN/STOP key. Did the computer stop at the same line number? You can try this several times to see if you can stop the computer at different places in the program. It took us seven tries before we got the computer to stop at least once at each program line; 110, 120, and 130.

Can you stop the computer at line 100? Probably not, since this program step is done only once, right at the start of the program. It isn't included in the loop, so you'd have to be pretty fast in going from RUN to pressing the RUN/STOP key to catch it doing this at the start of the program.

Run the program again and wait until the numbers being displayed are into the hundreds. Press the RUN/STOP key. The number display will stop. Now, type CONT. What happens? The display continues. It does not start over again from zero. Press the RUN/STOP key again to stop the program. Now type X=10000 [RETURN]. Type CONT. What is displayed now? The display continues, but the numbers are in the ten thousands. When you used the RUN/STOP key to stop the program, you were able to change the value given the label X. You then restarted the program and the count continued, starting with the new value.

If your program is running, stop it. Just press the RUN/STOP key. Now, press/hold the RUN/STOP key and press the RESTORE key. What happens? The TV screen is cleared and the READY message appears in the upper left-hand corner.

Type NEW to clear out the program and enter the following program:

```
50 INPUT X
60 PRINT X
70 GOTO 50
```

Check it and make any corrections that are needed. Run the program and type a value when the computer displays the question mark on the screen. Is the value "printed" on the TV screen properly? Is the computer waiting for another value? The number should be displayed and another question mark should appear below it.

When the question mark appears again, press the RUN/STOP key. Does it have any effect on the program or the display? No,

there is no effect. The RUN/STOP key cannot be used to break the computer out of an INPUT command, where it is waiting for information from the keyboard.

Press/hold the RUN/STOP key and press the RESTORE key. What happens now? The screen is cleared and the READY message appears. When the RUN/STOP key seems to have no effect on the program, you can use the RUN/STOP and the RESTORE keys together to reset the computer.

REAL DECISIONS

Although the GOTO command can be used to force the computer to go to a specific program line, no decision is made. So, how can computers make complex decisions in medicine, finance, and science? Most complex "questions" can be broken down into smaller ones that are easier to understand and answer. Your doctor doesn't take one look at you and decide that you have an ear infection. He asks questions such as:

"What hurts?"
"Do you have a fever?"
"Are you taking any medicine?"
"Have you had any trouble hearing?"

and so on.

Computers make decisions in the same way, and it's the computer programmer who breaks down the complex questions into simpler, smaller ones that are easier to answer. Computers really use statements, instead of questions, and they *evaluate the statements*, telling whether they are true or false. Here are a few examples of statements that can be *evaluated* by a computer such as your VIC 20:

5 is greater than 78.	(True or False)
INCOME is less than EXPENSES.	(True or False)
WEIGHT is equal to 1000.	(True or False)

These statements are either true or false, and we'll assume we know the values for INCOME, EXPENSES, and WEIGHT. The computer can test similar statements and then use the true-or-false answer to make decisions. The computer can't give you a true-or-false answer for questions such as, "Margaret is prettier than Joan," or "Fred is nicer than Sam."

Here's an example of a statement that is either true or false, and two "actions" that are based on the answer:

It is raining today. (True or False)
If the answer is TRUE, take your umbrella.
If the answer is FALSE, don't take your umbrella.

You can see how the statement is evaluated and the proper action taken by looking at the *flowchart* in Fig. 4-6. Once the statement has been evaluated, the "program" branches in either of two directions, based on the answer.

You can simplify this type of operation by putting the statement and the "true" answer in a slightly different form: IF "It's raining" is true, THEN take your umbrella.

The "false" answer isn't included here, since if it isn't raining, you usually don't think about an umbrella at all. You can "test" the statement, "It's raining," by looking out the window or opening the door. If the statement is true, THEN you do something special. If the statement is false, no special action is needed. This type of operation is shown in Fig. 4-7.

Computers using BASIC-language programs can evaluate statements in a similar way to see whether they are true or false. Of course, your VIC 20 computer can't look out a window and tell you that it's raining, so what can it do to make decisions? Since you've used four simple math operations, you'll now see how the computer can compare values to see whether they are equal or unequal. You'll also see how the computer can test two values to see if one is greater-than or less-than the other. These operations give the computer a great deal of decision-making power.

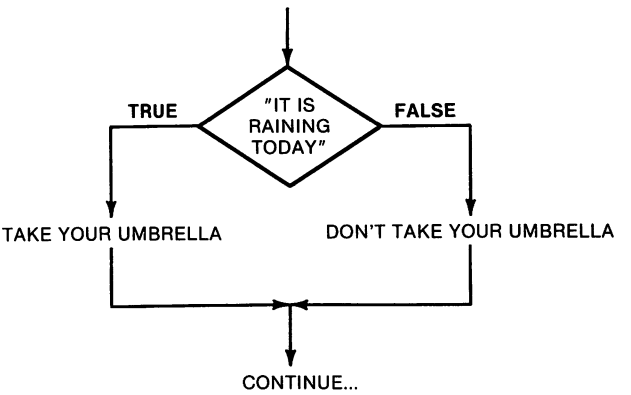


Fig. 4-6. The "umbrella" flowchart with true and false actions.

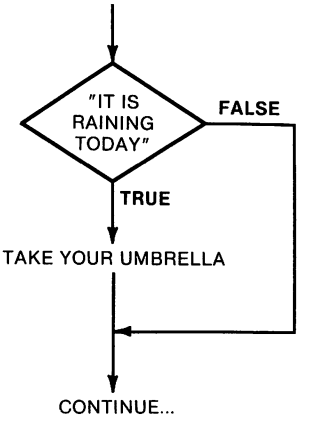


Fig. 4-7. The "umbrella" flowchart with only the true action.

THE IF-THEN OPERATION

Just as we used simple if-then sentences in the previous examples, the computer can use IF-THEN operations in programs. As part of an IF-THEN operation, the computer is given a "statement" to evaluate and a "command" to do if the statement is true. Here's the general form for the IF-THEN operation used with your VIC 20:

IF (this statement is true) THEN (do this command)

and it can be condensed as:

IF (statement) THEN (command)

Let's look at the types of statements that can be evaluated as either true or false by the VIC 20:

SCORE = 100	(Is "SCORE equal to 100" True or False?)
SCORE < 75	(Is "SCORE less than 75" True or False?)
SCORE > AVERAGE	(Is "SCORE greater than AVERAGE" True or False?)

The symbol < is used for "less-than" and the symbol > is used for "greater-than." You'll see these symbols used often in computer programs.

Here's how one of these statements might be used in a BASIC-language program for your VIC 20:

```
140 IF SCORE = 100 THEN PRINT "PERFECT"
```

In this example, the computer evaluates the statement, "SCORE = 100," to see if it is true or false. If it is false, the computer goes on to the next program line. If it is true, the command following THEN is done and PERFECT is "printed." After PERFECT is displayed on the TV screen, the computer goes on to the next program line. This can be shown with the help of the *flowchart* in Fig. 4-8.

All types of commands can be used in IF-THEN operations. For example, if you want the computer to go to line 1575 when SCORE is greater than AVERAGE, the IF-THEN operation would look like this:

```
IF SCORE > AVERAGE THEN GOTO 1575
```

A flowchart for this is shown in Fig. 4-9. Keep in mind that the computer only goes to line 1575 if the statement, "SCORE > AVERAGE," is true. *Labels* have been used in this statement to keep it general. Perhaps test scores have been typed in from the keyboard and the computer has figured out the average before it gets to this part of the program.

The command that follows the THEN is just like a separate program line in a BASIC program, and several commands can be used together as long as they are separated by colons. In this example, a PRINT and a GOTO command are used after the THEN:

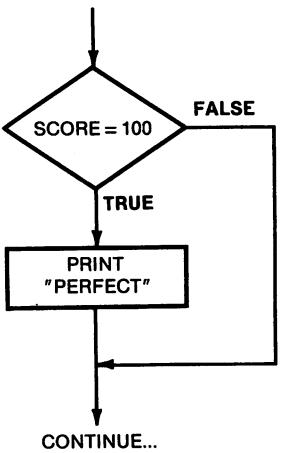


Fig. 4-8. A typical IF-THEN flowchart.

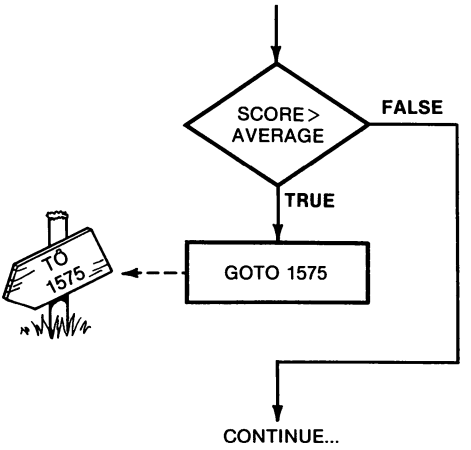


Fig. 4-9. An IF-THEN flowchart with a GOTO command.

```
IF SCORE < 75 THEN PRINT "NOT TOO GOOD" : GOTO 450
```

A flowchart for this is shown in Fig. 4-10.

Experiment No. 4-2. The IF-THEN Operation

This experiment will show you how an IF-THEN operation can be used in a BASIC-language program to make a decision. You will be able to type in numbers between 0 and 100 and the computer will tell you whether they are greater-than or less-than 50.

Clear out any old programs that are in your VIC 20 and load the following program. *Watch those line numbers*!

```
20  INPUT "NUMBER "; N
60  IF N > 50 THEN PRINT "GREATER THAN 50"
70  IF N < 50 THEN PRINT "LESS THAN 50"
90  GOTO 20
```

List your program to be sure that it has been entered correctly. Make any changes that are needed. Remember that "long" program lines will "wrap around" on the screen to the line below.

Run the program. Type in several values between 0 and 100, some greater-than and some less-than 50. Does the computer give you the right response for each? Is the computer really "smart"? The computer should give you the right answers as it determines whether the value is greater-than or less-than 50. If the program

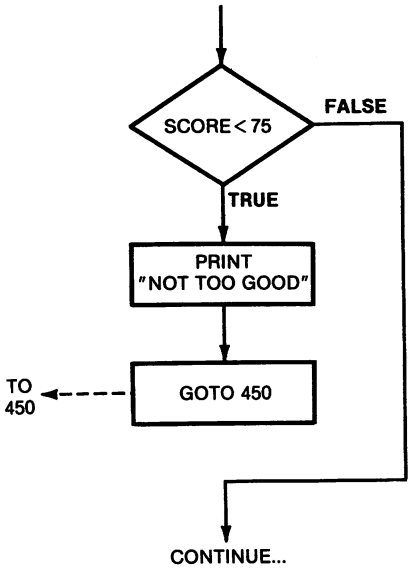


Fig. 4-10. An IF-THEN flowchart with multiple commands.

does not work correctly, stop the computer, list the program, and check it again. The computer isn't really very smart, since your program tells it exactly what to do. A computer just follows directions.

Your computer will continue to wait for numbers to "test." What happens when you type the number 50? Is anything printed? Is this what you expected? The computer program checks for values greater-than or less-than 50, but it does not check for the value 50, itself. Can you suggest a way to have the computer check the value entered to see if it is 50? Line 80 has not been used, so you can try and put in a program line of your own, if you wish to.

```
80 _____
```

You can check for the condition, N = 50, and the following program line can be used:

```
80 IF N = 50 THEN PRINT "EQUAL TO 50"
```

Put this program line in your program and run it. When the computer "asks" for a value, type in 50 from the keyboard. Does the computer program display the correct message? If line 80 has been typed in correctly, the computer should print the message, EQUAL TO 50, when you type in the value, 50.

At the start of this experiment, it was noted that values between 0 and 100 could be entered. When the program is running, type in 567. What happens? This value is accepted, and the "GREATER THAN 50" message is displayed. We would like to limit the numbers that can be input to between 0 and 100. Can the computer be used to detect numbers greater than 100? What about numbers less than 0? Yes, an IF-THEN operation can be used to check the keyboard entries and to reject any that are outside of the 0 to 100 range. You won't have to write these program steps, but think about what you'd like the computer to do. Here are several additional lines (marked with an asterisk (*)) that will solve the problem:

```
*  10  PRINT "NUMBERS BETWEEN 0 AND 100 ONLY"
   20  INPUT "NUMBER "; N
*  30  IF N > 100 THEN GOTO 10
*  40  IF N < 0 THEN GOTO 10
   60  IF N > 50 THEN PRINT "GREATER THAN 50"
   70  IF N < 50 THEN PRINT "LESS THAN 50"
   80  IF N = 50 THEN PRINT "EQUAL TO 50"
   90  GOTO 20
```

What effect do these program lines have? Line 10 just prints a message on the screen that tells the user what range of numbers is allowed. Once the value has been input by the computer at line 20, it is checked to be sure that it is in the range of 0 to 100. How is this done? At line 30, the computer tests the value to see if it is greater than 100. If it is, the computer goes back to line 10 and then asks for another value. At line 40, the computer tests the value to see if it is less than zero. If it is, the computer goes back to line 10 and then asks for another value. Only if the value is between 0 and 100 does the computer go on to line 60, where the value is tested to see if it is greater-than, less-than, or equal-to 50.

If you would like to, you can enter the new lines marked with an asterisk (don't enter the asterisks!), and see how the program works. Use a sheet of paper and try to sketch the flowchart for the complete program.

In Experiment 4-2, you saw how the IF-THEN operation can be used to make decisions. Although these decisions weren't critical, this type of decision-making is what makes computers so powerful. It can make these types of decisions quickly, without coffee breaks, and it will not make mistakes. Yes, mistakes are made, but because a *program doesn't work properly* or a circuit doesn't work

the way it should. Computers have survived millions of miles of space travel and have worked perfectly in the far reaches of our solar system. They are making true-false decisions all the time.

You have seen how a GOTO command can be used to force the computer to go to a specific line in a program. When used in a loop such as:

```
10 PRINT "TEST"
20 GOTO 10
```

there is no easy way to get out of the loop. If you wanted to go through the loop only 10 times, there is no way to tell the computer to stop after "printing" TEST on the TV screen 10 times. Although it is not often used this way, the IF-THEN operation can be used to count loops. (You'll see a better way to control loops in another chapter.)

Let's take a further look at how IF-THEN operations can be used to count loops and to provide a way out. Earlier in this chapter, a program was used to show how the GOTO instruction could be used:

```
10 X = 0
20 PRINT X
30 X = X + 1
40 GOTO 20
```

Remember that the X = X + 1 operation simply increased the value labeled X by one. If you run this program you'll see a column of numbers on the left side of the TV screen. The numbers are "pushed up" as new ones are "printed" at the bottom of the TV screen. The value of X is increased by one each time the computer goes through the loop. An IF-THEN statement could be used to break out of the loop after the computer goes through it a set number of times. Here is an example of that type of program:

```
10 X = 0
20 PRINT X
30 X = X + 1
40 IF X = 10 THEN END
50 GOTO 20
```

The IF-THEN operation tests the statement, X = 10, to see if it is true. If it is true, the command following THEN is done. The END command may be a new one to you, but it is easy to figure out what it does. It simply stops the computer. What happens if X

is not equal to 10? Well, the statement, X = 10, is then false, and the computer goes on to program line 50. It does not execute the END command unless the condition in the IF statement is true.

Can you figure out what numbers will be printed on the TV screen when this program is run?

Experiment No. 4-3. Counting Loops

This experiment will show you how to use an IF-THEN command to get the computer to "break out" of a loop after doing it a number of times. You will not have to put together any program lines of your own.

Clear the computer for a new program. Load the following program and check it. Correct any errors.

```
10  X = 0
20  PRINT X
30  X = X + 1
40  GOTO 20
```

Run the program. What do you see on the TV screen? Is this what you expected? You should see a column of numbers on the left side of the screen. The numbers seem to "move up" as new ones are displayed at the bottom of the column. Can you stop the computer? Yes, press the RUN/STOP key. Now, *change* program line 40 and *add* program line 50, as shown below:

```
40  IF X = 10 THEN END
50  GOTO 20
```

Check your program to be sure these lines have been entered correctly. Before you run the program, try and figure out what numbers will be displayed on the TV screen. Run the program and see if you are right. You should see the numbers 0 through 9 displayed in a column on the left side of the TV screen. Is this what you expected? Maybe you thought you'd see 0 through 10.

Remember that after the one is added to the value labeled X (line 30), it is tested by the IF-THEN operation (line 40). If X is not equal to 10, the computer goes on to line 50 where it is pointed back to line 20. If X is equal to 10, "X = 10" is true and the computer is stopped by the END command. The value 10 is never printed. However, the computer did go through the loop 10 times, printing 10 values on the TV screen. A flowchart for this program is shown in Fig. 4-11.

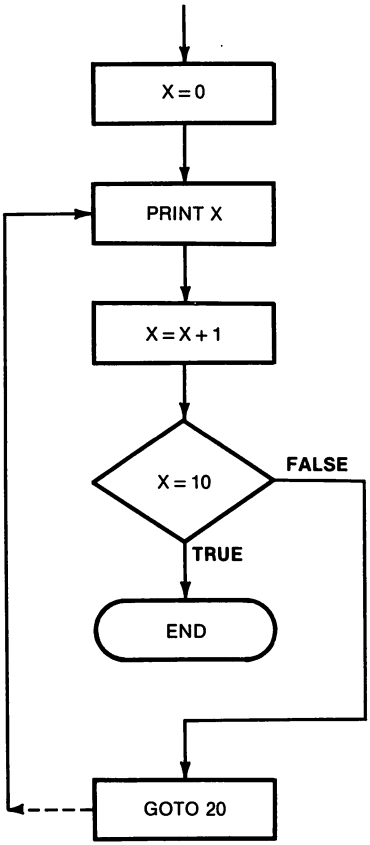


Fig. 4-11. A flowchart for the loop-counting program.

In the IF-THEN commands, GOTO commands are often used to point the computer to a line in the program:

```
IF XF = FR + FQ THEN GOTO 1265
```

However, you may find IF-THEN commands in which the word GOTO has been left out. The following program line has the same effect as the preceding one. The computer "knows" that you want it to go to line 1265 if XF is equal to (FR + FQ):

```
IF XF = (FR + FQ) THEN 1265
```

TO BE AND/OR NOT TO BE

Sometimes it is helpful to make several true or false decisions at the same time. These statements can be combined in one IF-THEN operation. Look at the following:

IF *we are out of eggs* AND IF *you want eggs for breakfast*, THEN *buy some on the way home.*
IF *it is raining* OR IF *rain is forecast*, THEN *take your umbrella.*

These can be simplified so they are easier to understand:

IF (we are out of eggs) AND (you want eggs for breakfast), THEN . . .

IF (it is raining) OR (rain is forecast), THEN . . .

By using AND and OR operations, the computer can test several different statements at the same time. This saves "computer time," and by combining statements this way, programs may be simplified, too. It is easy to put statements together in IF-THEN operations as long as you know the rules for using the AND and OR operations. Here they are:

1. When an AND is used, both of the statements must be true for the THEN command to be done. If either of the statements is false, the THEN command is not done. This is shown in Table 4-1. The results of testing or evaluating each statement are shown in parentheses.
2. When an OR is used, either or both of the statements must be true for the THEN command to be done. If both of the statements are false, the THEN command is not done. This is shown in Table 4-2.

In a previous experiment, two IF-THEN operations were used to test a number to see if it was between 0 and 100. If it was below 0 OR above 100, it was rejected. Here is how the operations were used:

```
10 PRINT "NUMBERS BETWEEN 0 AND 100 ONLY"
20 INPUT "NUMBER "; N
30 IF N > 100 THEN GOTO 10
40 IF N < 0 THEN GOTO 10
```

and so on.

Table 4-1. Table of Results for an AND Operation

Statement	Statement	THEN Command Done?
(False)	(False)	No
(False)	(True)	No
(True)	(False)	No
(True)	(True)	Yes

Table 4-2. Table of Results for an OR Operation

Statement	Statement	THEN Command Done?
(False)	(False)	No
(False)	(True)	Yes
(True)	(False)	Yes
(True)	(True)	Yes

If the number typed in is greater than 100, the IF statement at line 30 detects this and the computer is forced to line 10. If the number is less than 100, the computer goes on to see if the number is less than 0. If it is, the GOTO operation at line 40 forces the computer to line 10. Only when the number is between 0 and 100 will the computer go on to the program line following line 40.

The two IF statements can be combined into one by using an OR operation, and parentheses have been used to separate the statements:

```
10 PRINT "NUMBERS BETWEEN 0 AND 100 ONLY"
20 INPUT "NUMBER"; N
30 IF (N > 100) OR (N < 0) THEN GOTO 10
```

In this example, the IF-THEN operation checks two statements to see if the number is greater than 100 OR less than 0. It is possible to have the computer check and see if the number is *within* the range, instead. This uses an AND operation, and it is more complicated than the OR operation.

MIXED OPERATIONS

There are times when you will want to check the same two values for two different conditions. For example, it might be useful to check today's RAINFALL to see if it is *greater-than or equal-to* the RECORD rainfall that is on file in the computer. You could use the OR operation to do this:

```
IF (RAINFALL > RECORD)  OR  (RAINFALL = RECORD) THEN . . .
```

By looking at Table 4-2, you can see that the THEN command would be done if *either of the statements is true*. Since the same values, RAINFALL and RECORD, are being compared in each operation, you can have the computer do the greater-than and equal-to operations at the same time. Other combinations of greater-than, less-than, and equal-to operations can be used, too, as shown here:

```
GRADE <= SCORE(GRADE less-than or equal-to SCORE)
RAINFALL >= RECORD(RAINFALL greater-than
                        or equal-to RECORD)
TRY <> TARGET  (TRY less-than or greater-than TARGET)
```

This combination of operations is only useful when the same two values are being compared in different ways. In the TRY <> TARGET example, the computer is really checking for the not-equal condition; that is, TRY is *not equal* to TARGET.

These conditions can be checked by using simple IF-THEN operations, and several examples are shown here. Different types of commands are shown following "THEN" so you can see examples of how they are used.

```
IF RAINFALL >= RECORD THEN PRINT "NEW RECORD IS "
                              RAINFALL
```

In this example, if the RAINFALL is greater-than or equal-to the RECORD, the computer prints the message, "NEW RECORD IS," followed by the value assigned the label RAINFALL.

```
IF TRY <> TARGET THEN PRINT "YOU MISSED, TRY
                        AGAIN":GOTO 595
```

In this example, the program tests to see if the values labeled TRY and TARGET are not equal. If they are not equal, the computer prints the message "YOU MISSED, TRY AGAIN," and it goes to program line 595.

```
IF GRADE <= SCORE THEN GOTO 250
```

Here, the computer checks the value labeled GRADE to see if it is less-than or equal-to the value labeled SCORE. If this is true, the computer goes to program line 250.

THE NOT OPERATION

You have seen how the AND and OR operations can be used in IF-THEN operations so that several different statements can be tested at the same time. There is one other operation that isn't used very often, so we've left it for last. This is the NOT operation, and it allows you to reverse the meaning of a statement. For example, "You say white, I say black. You say black, I say white," or "If it's true, make it false. If it's false, make it true."

Here are two examples in which the NOT operation has been used to "reverse" the meaning of each statement:

```
NOT (COST = VALUE)    (COST is not equal to VALUE)
NOT (COST < VALUE)    (COST is not less-than VALUE)
```

It takes a bit of programming experience and trial and error to get used to the NOT operation. It isn't used very often, but we just wanted to show you that it is part of programming in BASIC.

A LITTLE HISTORY

The AND, OR, and NOT operations are called *logical operations*, and although you may find it difficult to believe, these operations have been in use for over *100 years*. In the nineteenth century, the English mathematician, George Boole, devised a way of solving logic problems with a special form of mathematics. This type of math is now known as *Boolean algebra*, but it was relatively unused until scientists and engineers applied it to solving problems with the first electronic computers in the 1940s.

These three logical operations are the foundation of all desk-top computers, just like the one you are using. These logical operations are done by tiny electronic circuits, of which there are thousands and thousands in your computer. These circuits make hundreds of thousands of decisions each second, and it is this high-speed decision making that makes small computers so powerful and useful. Even the biggest computers used by banks, universities, and research centers use electronic circuits that do AND, OR, and NOT operations. In the same way that all math operations depend on addition and subtraction, all modern computer operations depend on AND, OR, and NOT operations. Even small pocket calculators and digital clocks depend on these simple operations.

PICK A NUMBER, ANY NUMBER

In many games, you don't know how many spaces you'll move until you spin the spinner or toss the dice. Since computers are used in many games, they need some way to "spin a spinner," too. We don't know of any computers that have spinners or dice in them, but most computers do have a way of "picking a number," or coming up with a *random* value. This means you can get the computer to "pick a number" for you. Of course, you won't know ahead of time what number the computer will come up with.

The VIC 20 can "pick" a random number, or value between 0 and 1, by using the RND(1) operation which is part of its BASIC language. This means that the computer can come up with values between 0 and 1, such as 0.345, 0.914, and 0.310. The RND(1) operation NEVER gives you the value 0 and 1—that's just the way it works. You can use a BASIC operation like this in a program to get a random value:

```
SAMPLE = RND(1)
```

The computer labels the random number "SAMPLE."

Experiment No. 4-4. Random Value Program 1

You will enter and run a program that will print random values on the TV screen. The purpose of this experiment is to show you that the random number operation really works.

Use the NEW command to clear out the computer for a new program. Enter the following program and check it:

```
10 SAMPLE = RND(1)
20 PRINT SAMPLE
30 GOTO 10
```

Run the progam. You will see values displayed on the TV screen, but they will move up the screen too quickly to be read. Press the RUN/STOP key to stop the program, and you should see 19 or so random values displayed on the TV screen. The chances are very good that no two values will be the same. You may see a value such as 8.93456278E−03. Although it looks odd, it's just the computer's way of displaying a very small value and you don't need to know what it means right now.

Did you see the value 0 or 1 displayed? The display may have moved too quickly for you to see all the values clearly, so you

might have missed the value 0 or 1 if either were displayed. Can you think of a way to check for the values 0 or 1? An IF-THEN operation can compare the value labeled SAMPLE with the values 1 and 0. Here's how it could be done:

```
30  IF (SAMPLE = 1) OR (SAMPLE = 0) THEN END
40  GOTO 10
```

The program would now be:

```
10  SAMPLE = RND(1)
20  PRINT SAMPLE
30  IF (SAMPLE = 1) OR (SAMPLE = 0) THEN END
40  GOTO 10
```

The IF-THEN operation at line 30 tests for either the SAMPLE = 1 or the SAMPLE = 0 condition. If either of these conditions is true, the THEN command is done. In this case, the END command has been used. You saw the END command earlier, and its meaning is obvious; it just stops the computer.

Put the new line 30 in your program and add line 40 to complete the new program. Run your program. Does the computer stop? No, the computer continues to display random values on the TV screen. You can let it run all night if you want to, but it will continue to display new values without stopping. What is your conclusion? The RND(1) operation generates random values *between 0 and 1*, but it does not generate the value 0 or 1.

Since it's difficult to move your game marker by 0.984812986 spaces, the random values picked so far don't seem to be particularly useful. However, by simply multiplying the random values by another value, you can have the computer pick more useful numbers. For example, if you want to have the computer pick a random number between 0 and 10, you can do this by multiplying the value found in the RND(1) operation by 10. Here is what that part of the program would look like:

```
SAMPLE = RND(1) * 10
```

Now the results will be between 0 and 10 instead of between 0 and 1. If you want a random value between 0 and 50, you can get the computer to pick one in this range by using the following program line:

```
SAMPLE = RND(1) * 50
```

In both cases, the RND(1) operation "picks" a random value between 0 and 1, but the multiplication operation spreads the numbers over a larger range. In almost every case, the value used in the multiplication is a whole number, or integer.

Experiment No. 4-5. Random Value Program 2

In this experiment, you will find out how the computer can be made to "Pick a number between 0 and 10." Clear your computer and load and check the following program:

```
10  SAMPLE = RND(1) * 10
20  PRINT SAMPLE
30  GOTO 10
```

Run the program. You will see the random values "move up" the screen as new values are displayed on the bottom row on the TV screen. After the program has run for a few seconds, press the RUN/STOP key to stop the program. Are the values displayed between 0 and 10? Are any values displayed more than once? The values are between 0 and 10, and we didn't find any values that were equal when we ran the program. Although the chances are small, you may find two equal values displayed on your TV screen. The values 0 and 10 aren't found.

Change program line 10 in your program to the following:

```
10  SAMPLE = RND(1) * 90
```

Now run the new program. After a few seconds, stop the program and look at the values displayed on the TV screen. Are they between the values 0 and 90? The values should be between 0 and 90.

In this experiment, you saw that you could change the "range" of random values that the computer can "pick" for you.

Although the range of the random values can be changed, it is still difficult to move a game marker by 8.73481053 spaces. So, what can be done to have the computer pick values such as 9, 4, and 6, instead of 9.45271052, 4.8562321, and 6.46241739. The *integer operation*, INT, is used to remove or "strip-off" any decimal fractions; digits to the right of the decimal point. In this way, it "changes" a number that has a decimal fraction into an integer or whole number. Here is how it is used:

```
PRINT INT(9.45271052)
```

When this is done by the computer, only the 9 is displayed on the TV screen. The INT and RND operations can be used together to make the computer "pick" random whole numbers.

Once a random value between 0 and 1 has been "chosen" by the RND operation, a multiplication operation increases its "range." The INT operation is used last to "strip-off" the decimal fraction and "change" the value into an integer. This may seem a bit confusing, so look at the following example. This program line causes the computer to "pick" a number between, *and including*, 0 and 9:

```
SAMPLE = INT(RND(1) * 10)
```

The largest random values will always be one less than the value used in the multiplication. Here are some sample program lines that you can use in your programs. The label, X, has been used as the label for the value:

(a) Coin Flip: 1 = heads, 0 = tails

```
10 X = INT(RND(1) * 2)
```

(b) Dice Toss: Values 1 through 6

```
10 X = INT(RND(1) * 6) + 1
```

We're not trying to make you an expert programmer, but here are some general-purpose random-value program lines in which you can substitute your own ranges for games, teaching programs, magic tricks, puzzles, and other programs where a random integer is needed.

(a) To pick a number between 0 and T:

```
10 X = INT(RND(1) * (T + 1))
```

(b) To pick a number between 1 and T:

```
10 X = INT(RND(1) * T) + 1
```

(c) To pick a number between Q and R, where R > Q:

```
10 X = INT(RND(1) * (R - Q + 1)) + Q
```

A WORD OF CAUTION

Although you're not going to be a programmer, we want you to know that you cannot use "reversed words" as labels in your computer programs. It is unlikely that you will run into any problems

with prepackaged computer programs or cartridges that have been developed by a commercial group or competent programmer. However, if you decide to try your hand at even simple programs, you may run into problems without realizing it.

Reserved words are nothing more than the words in the BASIC language that represent operations. For example, IF, THEN, AND, and RND are all reserved words. So, what's the big deal? Well, if you're writing programs, be sure to check your labels so that you don't use any of the reserved words in them. There is a complete list of the reserved words in Appendix B, so these words can be found in one place.

Suppose that you wrote a short program to count something you've decided to call "Information Files." As a label, you've decided to use IF. Well, the computer "sees" the IF you've used as a label in one of your calculations:

```
145 IF = A5 + (TQ / 17)
```

The computer thinks that it is "looking at" the start of an IF-THEN operation. That really confuses the computer and it will give you an error message:

```
?SYNTAX
 ERROR IN 145
READY.
```

If you see this type of error message when you try to run a program, check to be sure that a reserved word hasn't been used.

QUESTIONS

1. What does a GOTO command do in a BASIC-language program?

2. What information must you provide in a GOTO command?

3. Can a GOTO command cause a computer error? If so, what kind of an error?

4. Does a GOTO command have to point the computer to the start of a program, or can it point it somewhere else?

5. How can you get out of an "endless" loop?

6. How can you stop the computer when it's waiting for you to type in information for an INPUT command?

7. What kinds of "statements" can the VIC 20 test, or evaluate?

8. How are statements used so the VIC 20 can make decisions?

9. What conditions can be checked in IF-THEN commands?

10. What kinds of commands can be used after the THEN in an IF-THEN command?

11. What do the AND, OR, and NOT operations do?

12. What operation gets a random number from the computer?

13. How can you get the integer 15 from the value 15.962?

FOOD FOR THOUGHT

1. In one of the experiments in this chapter, the following program was used to show you how an IF-THEN command could be used to get the computer out of a loop:

```
10 X = 0
20 PRINT X
30 X = X + 1
40 IF X = 10 THEN END
50 GOTO 20
```

   Rewrite the program so that it does the same thing, but use some other condition in an IF-THEN command. There's always more than one way to solve a problem.

2. In an example of the logic OR operation, an IF-THEN command was used to check a number to see if it was greater-than 100 OR less-than 0. Suggest a way in which the AND operation could be used instead. The operation must still check to be sure that values are between 0 and 100. Remember, the values 0 and 100 are allowed, too.

PROBLEMS

1. Write a program where three numbers are entered from the keyboard into the computer. The two largest numbers are multiplied, and the result is then divided by the smallest number. The division result is then displayed. Assume that the numbers can be entered in any order. Thus, if your program is properly written, the same result will be produced regardless of the sequence in which the numbers are entered. Assume that none of the numbers are the same.

2. Write a program that asks you if the washing machine is running. You answer with either a 1 for yes or a 0 for no. The computer then asks you if the water drain is plugged up, and again you answer with a 1 for yes or a 0 for no. If the washing machine is running and the water drain is plugged, the computer should display the words "STOP THE WASHING MACHINE."

3. Write a program that lets you enter an amount of money, such as 2.43, which represents $2.43. The computer then tells you how many quarters, dimes, nickels, and pennies would be required for this amount of money. You want to use as few coins as possible, so calculate how many quarters are required first, followed by dimes, nickels, and pennies.

4. Write a program where the computer generates an integer random number between 0 and 100. The number is then displayed on the TV screen.

5. Write a program that generates a random number between 35.46 and 48.7. The program should generate random numbers continuously and display them on the TV screen.

6. Have the computer generate two random numbers between 0 and 50. These numbers should be displayed on a single line, separated by a plus (+) sign. You then have to enter the *sum* of these two numbers. If you enter an incorrect sum, the computer displays the same "problem" again. If you enter the correct answer, the computer generates a new problem for you.

7. Modify the program from Problem 6 so that the computer asks you 10 problems and gives you a grade between 0 and 100 after all 10 problems have been answered. If an incorrect answer is entered, 10 points should be subtracted from the grade and the computer should generate a new problem. Any "missed" problems should not be repeated.

8. Write a program where the computer generates a random number between 0 and 100, and you have to guess the number by entering guesses on the keyboard. After you enter each guess, the computer should tell you if your guess is too high, too low, or correct. If your guess is too high or too low, you continue to enter guesses until the correct number is entered on the keyboard.

9. Write a program that converts degrees Celsius to degrees Fahrenheit. The equation for this is F=1.8*C+32, where C is the temperature in degrees Celsius and F is the resulting temperature in degrees Fahrenheit. When this program is running, enter the value 21. The computer should display the result (F) of 69.8.

10. Modify the program in Problem 9 so that the temperatures displayed (F) are rounded up or down. Thus, if a temperature of 21 is entered, the temperature displayed should be 70 rather than 69.8. Likewise, if the calculated temperature is 34.4, 34 should be displayed. If the fractional part of the temperature is .5 or greater, round up.

11. Write a program that lets you enter a day between 1 and 365. The computer then displays the month and the day of the month. For example, if you enter a 1 the computer displays January 1. If you enter a 45 the computer displays February 14. (NOTE: This is a long and tedious program to write.)

12. Write a program that generates a multiplication table for all possible combinations of numbers between 1 and 3. Thus, you should see:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
```

    on the TV screen.

13. Write a "checkbook" program. In this program, you should first enter the number of checks written the previous month, followed by the amount of each check. The computer should then display the total amount of money spent. You should next be able to enter the number of deposits that you made, followed by the amount of each deposit. You now enter the balance of your checkbook from last month, the amount of deposits are added to this, the amount of all checks is subtracted from the result, and this new balance is displayed on the TV screen.

14. One problem with the solution for Problem 13 is that you have to *know the number* of checks and deposits. If you write a lot of checks, or have a number of deposits, this can take time. Modify the solution to Problem 13 so that when an amount of 0 for a check or deposit is entered, the computer assumes that you have finished entering either check or deposit amounts. For instance, you might see the following:

```
CHECK AMOUNT ? 34.56
CHECK AMOUNT ? 89.50
CHECK AMOUNT ? 3.75
CHECK AMOUNT ? 0

TOTAL CHECKS: 127.81

DEPOSIT ? 100.37
DEPOSIT ? 0

PREVIOUS BALANCE ? 389.37
CURRENT BALANCE: 361.93
```

The CURRENT BALANCE was determined by adding 100.37 to 389.37, and then subtracting 127.81 from the result of 489.74.

5

More Power to You

Your VIC 20 gives you a lot of computer power in a small package. If you want to use this power, there are new things to learn about. For example, there are new ways that the computer can make decisions, ways of making programs simpler, new ways to store information, and so on. We'll talk about these things in this chapter. We'll start by showing you a few "tricks" that can be used to print information on the TV screen.

SPECIALTY PRINTING

So far, you have been able to display numbers and letters on the TV screen by using a PRINT command. The PRINT command prints messages that are enclosed in quotation marks along with any values that have been labeled. Here is an example:

```
10 PRINT "THIS IS A MESSAGE"
```

When the computer finds this PRINT command in a program, it prints THIS IS A MESSAGE on the TV screen. It looks just the same as if you had typed it on the keyboard. Since you can put letters and numbers in a "message," to have them printed on the TV screen, you can put other types of keyboard operations in messages, too.

In one of the first experiments you did, you used the CLR/HOME key to move the cursor to its home position in the upper left-hand corner of the screen. You also found that if you pressed the SHIFT key and then pressed the CLR/HOME key, the screen was completely cleared *and* the cursor moved to its home position. Remember, though, that this only clears the TV screen. Programs aren't cleared out of the computer.

It is often useful to start off with a clear or "clean" TV screen. Some programs clear the TV screen to erase an old game display, or to show a new problem to be solved. Well, the computer can't reach out and press the SHIFT and CLR/HOME keys, so there must be some other way to clear the TV screen.

The SHIFT and CLR/HOME key actions can be put in the message part of a PRINT command. So, when the computer reaches a PRINT command in the program where the SHIFT and CLR/HOME keys have been pressed as part of the message, it clears the TV screen automatically. Here's an experiment that shows you how this is done.

Experiment No. 5-1. Clearing the Screen

In this experiment, you will see how action keys can be used in the message part of a PRINT command to clear the TV display.

Clear the computer for a new program, and use the CLR/HOME key to clear the TV screen. Now type the following into the computer and leave the cursor as shown:

```
10 PRINT "■
```

You now have part of a PRINT command typed into the computer. Press the SHIFT key and then press the CLR/HOME key as if you were clearing the TV screen. What happens on the TV screen? Did you expect the computer to clear the screen? The computer displays a "reversed heart" symbol, ▚, right after the quote mark. Maybe you thought that the screen would be cleared instead. Your VIC 20 computer is pretty smart. It can tell that you're typing the SHIFT and CLR/HOME keys as part of a PRINT command's message; that is, between the quote marks. The computer "knows" that you don't want it to take any action right away.

Since the CLR/HOME key is an *action key*, it won't normally print anything on the screen when you press it. It clears the screen, instead. The "reversed heart" symbol is just the com-

puter's way of telling you that it knows that this clear-the-screen operation is to be done later, when the program is actually run. As you'll see later in this book, there are other special symbols that stand for other special operations.

Type the final quote mark and press the RETURN key, so the line looks like this:

```
10 PRINT " ♥ "
```

When the computer reaches this line in a program, it will recognize the special symbol and will know that it is supposed to clear the screen. Add the following line to your program:

```
20 PRINT "CLEAR TEST"
```

List the program and check it to be sure it is correct. Remember, you can simply retype a line if you've made an error. List the program three more times to put characters on the screen. Now run the program. What happens? The screen is cleared and the CLEAR TEST message appears at the top of the screen. Remember, the program simply cleared the screen and printed this message. (The usual READY message appears, too.) When this short program is finished, the computer is ready to do something else.

The clear-the-screen operation is quite useful to programmers. It is often used to clear the screen at the start of a program, or to clear it when a program has been run. It is used whenever a clear screen is needed.

TYPING SPECIAL SYMBOLS

When computer programs are provided as part of an experiment or problem, it would be confusing to see a special symbol, such as a "reversed heart," in the program listing. These special symbols are not found on the keyboard, so you might get confused trying to print them on the TV screen. So, when we provide computer programs for you to enter, you'll see the *keys* that you are to press instead of the special symbols:

```
10 PRINT "[SHIFT] [CLR/HOME]"
```

However, if you list this, you'll see:

```
10 PRINT " ♥ "
```

If a program is complex, we'll show you a program listing, too,

so you can compare your program with the one provided in the experiment. Just remember that you'll have to *press and hold* the SHIFT or CTRL key when either is shown.

PLAYING FOR POSITION

In several of the experiments you did in previous chapters, you found that it was fairly easy to display numbers in a column on the left side of the TV screen. The VIC 20 can also display numbers across the screen, in a row.

When the VIC 20 finishes a PRINT command it always places the cursor at the left side of the line below it. This automatically gives the computer a "clean" line on the TV screen for the next PRINT command. This is called a "line feed," and it is an automatic part of the PRINT command.

However, you can tell the computer that you don't want a "clean" line, so that printing will continue on the same line, instead. In order to keep the computer printing on the same line, you must put a semicolon at the end of the PRINT command. Here's an example:

```
670 PRINT "ONE LINE";
680 PRINT " OF PRINT"
```

After the computer prints ONE LINE on the TV screen, instead of moving the cursor to the left side of the line below, it leaves the cursor right after the E in LINE. When it reaches the following PRINT command, it continues printing OF PRINT right after ONE LINE. The result is ONE LINE OF PRINT, printed on a single line on the TV screen. The semicolon at the end of line 670 "told" the computer to leave the cursor where it was and not to go to a clean line.

Here's another example that shows how semicolons may be used to keep the computer printing on one line:

```
240 RZ = 2500
250 PRINT "SCORE =";
260 PRINT RZ;
270 PRINT " POINTS"
```

This causes the computer to print this as a single line on the TV screen:

```
SCORE = 2500 POINTS
```

Experiment No. 5-2. Printing in Columns and Rows

In this experiment, you'll see how a semicolon can be used with a PRINT command to tell the computer to print in rows instead of columns. Clear your computer for a new program and enter the following program:

```
10 PRINT X
20 X = X + 1
30 GOTO 10
```

Run the program. What do you see on the TV screen? Increasing numbers are displayed on the left side of the TV screen. Change line 10 in the program by typing in the following. Remember the semicolon at the end of the PRINT command:

```
10 PRINT X;
```

Your complete program should now look like this:

```
10 PRINT X;
20 X = X + 1
30 GOTO 10
```

Run the program. What do you see on the TV screen? The numbers are displayed across the screen, *line after line*. Stop the computer when the numbers reach 400 or more. Are any of the numbers at the right "edge" of the TV screen "broken," or are only complete numbers displayed?

Many of the numbers are broken at the right edge of the TV display, and several lines look like this:

```
387  388  389  390  39
 1  392  393  394  395
 396  397  398  399  4
00  401  402  403  404
```

The numbers are not lined up in columns, and a whole screen of numbers looks very confusing. However, we accomplished our goal of printing numbers across the screen rather than up and down one side.

THE TAB COMMAND

If you were running a program to help you balance a checkbook or put grocery prices on the TV screen, the "jumbled" display

shown in the previous section would be very difficult to read and understand. Help is on the way. The TAB command can be used to line up columns of numbers on the TV screen, and TAB is just short for "table." A printed "table" puts information in up-and-down columns that are easy to read, and the TAB command lets the computer display information in this easy-to-read form.

The TAB command uses a number that tells the cursor how far to move *from the left side of the TV screen.* For example, a TAB(9) command tells the cursor to move over nine spaces from the left side of the TV screen, while a TAB(15) command tells the cursor to move over 15 spaces from the left side of the TV screen. *The TAB command is used only as part of a PRINT command.* For example:

```
10 PRINT TAB(5) COST;
```

In this example, the computer moves the cursor over five spaces to the right and prints the *value* that has been labeled COST. Since a semicolon has been put at the end of the command, the cursor won't go on to a clean line below. Perhaps something else is going to be printed on the same line.

When using the TAB command, don't use any extra spaces between the word TAB and the parentheses, or the computer will not know what you mean. For example, this TAB command is incorrect:

```
TAB (9)
```

Experiment No. 5-3. Using the TAB Command

The purpose of this experiment is to show you how the TAB command can be used to control the display of information in neat columns that are easy to read.

Clear your computer for a new program and carefully enter the following program:

```
10 PRINT X;
20 X = X + 1
30 PRINT TAB(5) X;
40 X = X + 1
50 PRINT TAB(10) X
60 X = X + 1
70 GOTO 10
```

Be sure you have typed in the semicolons at the end of lines 10 and 30. Run the program. What do you find displayed on the TV screen? Three columns of numbers are displayed on the TV screen, and the numbers should be easy to read in this form. Change line 50 in the program to the following:

```
50 PRINT TAB(15) X
```

What do you think will happen when you run the program? Run it and see. The right-hand column has been moved farther to the right. You can use a TAB command with a PRINT command to print things on the TV screen in a neat form.

The TAB command is used when you want to have numbers, words, or other information displayed in neat columns that you can read without straining your eyes. If you decide to try the TAB command in a program, just remember that you can only use it with a PRINT command. Remember, too, that the display on the VIC 20 has room for only 22 characters in a row across the screen.

MORE DECISIONS

In the previous chapter, you saw how the computer made decisions by using the IF-THEN command. This is a useful command, since it lets the computer test a statement to see if it is true or false. If the condition is true, the command following the THEN is done. Here's an example:

```
50 IF ALFA = 100 THEN PRINT "LIMIT REACHED": END
```

In this example, if the statement, ALFA = 100, is true, the computer prints LIMIT REACHED and the program ends. If ALFA = 100 is found to be false (this means that ALFA is *not equal to 100*), then the computer continues to the next program line. Remember that ALFA is just a label that identifies a value saved in the VIC 20 computer.

The VIC 20 has another type of decision-making instruction. While it's not as flexible as the IF-THEN command, it can be used in some cases to save computing time and programming time. This new command is the ON command, and it can only be used to *test a value* to see if it is equal to 1, 2, 3, 4 . . . and so on up to a maximum value of 255. What does the ON command do once it has done the test? Look at this example:

```
100 ON ALFA GOTO 155, 300, 950
```

ALFA is first tested to see if it is equal to 1. If it is, the computer goes to line 155 in the program. If ALFA is not equal to 1, the computer tests to see if it is equal to 2. If it is, the computer goes to line 300 in the program. If the value isn't 2, the computer tests to see if it is equal to 3. If it is, the computer goes to line 950. If the value is not 1, 2, or 3, the computer just keeps going in the program. A flowchart for this ON command is shown in Fig. 5-1.

In this example, the ON command only tested the *three values*, 1, 2, and 3, since there were only *three line numbers* after the GOTO part of the command. The ON command always starts testing with a value of 1, and it runs as many tests are there are line numbers in the ON command.

How could the ON command be used in a program? Many programs ask you to type in a number; to answer a multiple-choice problem, to move a player in a game, or to select a special action from a list. In the followng example, the computer user has been asked to type in a number to tell the computer which one of several programs to run. The choices are:

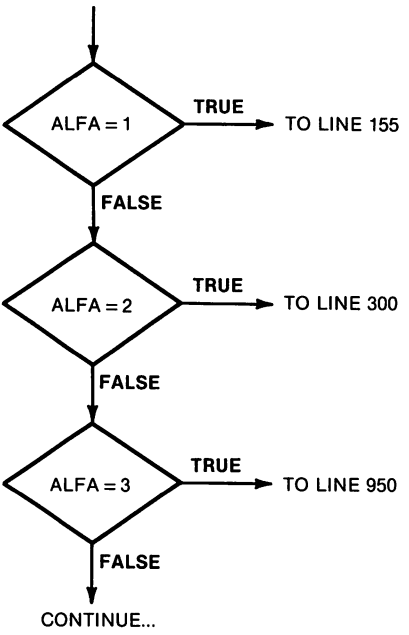


Fig. 5-1. Flowchart for an ON command.

1. ROAD-RACE GAME
2. MATH PROBLEMS
3. BUSINESS PROBLEMS
4. PHONE NUMBER FILE

Here are several program lines that could be used to "select" the program to be run. To make this simple, the program steps for each of the choices haven't been shown:

```
200 PRINT "TYPE YOUR CHOICE"
220 INPUT CHOICE
240 ON CHOICE GOTO 1250, 3570, 5790, 7100
260 GOTO 200

1250 REM START OF ROAD-RACE GAME PROGRAM
1255 Etc . . .

3570 REM START OF MATH PROBLEMS FOR KIDS
3575 Etc . . .

5790 REM BUSINESS CALCULATIONS PROGRAMS
5795 Etc . . .

7100 REM PHONE NUMBER AND ADDRESS FILES
7120 Etc . . .
```

In this example, if the CHOICE entered is not 1, 2, 3, or 4, the computer reaches the GOTO command at line 260 and it asks you to type in another choice. This is an important part of the program, since it prevents the computer from taking any unplanned action when some number other than 1, 2, 3, or 4 is entered. This is called on "error trap," since it "traps" numbers that can't be used for choices. Some programs use very complicated "error traps" so that programs can't be "messed-up" by someone who types in something a bit unusual.

When a computer asks you to tell it what you want it to do by listing several choices, this list is called a *menu*. Many programs use menus so that you can select one action from a list. When you hear people talk about a program that uses a *menu*, this list of choices is what they are talking about.

You can probably see that the ON command is limited in what it can do, but it can be used to replace many IF-THEN commands:

```
IF CHOICE = 1 THEN GOTO 1250
IF CHOICE = 2 THEN GOTO 3570
```

```
IF CHOICE = 3 THEN GOTO 5790
IF CHOICE = 4 THEN GOTO 7100
```

There is no reason why some of the line numbers in an ON command can't be the same. In the following example, if the value labeled CANS is 3 *or* 5, the computer goes to the same line in the program:

```
100  ON CANS GOTO 155, 200, 890, 1550, 890
```

COUNTING LOOPS—THE FOR-NEXT COMMAND

When we were discussing the IF-THEN command, we showed you how it could be used to get the computer out of a loop after it had gone through it a set number of times. The following program shows how an IF-THEN command can control a loop program so that "YOUR NAME" is printed on the TV screen five times:

```
10  T = 1
20  PRINT "YOUR NAME"
30  IF T = 5 THEN END
40  T = T + 1
50  GOTO 20
```

Computer programmers would say that the computer made five "passes" through this loop. "Pass" is a useful term, particularly when we are talking about loops, so we will use "pass" to mean that the computer has made a "trip" through the program loop.

Using an IF-THEN command to count passes through a loop is not very efficient, since you have to program the computer to count each pass and check to see if the limit has been reached. The FOR-NEXT command is used just to count passes through loops, and a *loop counter* is built right into the command. The FOR-NEXT command has two parts; a FOR command at the start of the loop and a NEXT command at the end of the loop. Here is how a FOR command starts a loop:

```
50  FOR GAME = 1 TO 10
```

The FOR operation first sets up a *loop counter* that has been labeled GAME in this example, but almost any label can be used. The example shown sets the counter to 1, and sets the maximum count to 10. The loop count has 1 added to it after each pass through the loop, so you can see that the computer will make 10 passes through this loop, with loop counts of 1, 2, 3 and so on up to 10.

The NEXT operation is put at the end of the loop:

```
100  NEXT GAME
```

This tells the computer that it has reached the end of the loop. The computer then adds 1 to the loop counter (GAME) to see if it has gone through the loop 10 times. When all 10 passes through the loop have been made, the computer continues with the program line that follows the NEXT command. Here is an example of how the FOR-NEXT command can be used:

```
50  FOR LTR = 1 TO 12
70    PRINT "I LOVE YOU!"
100 NEXT LTR
120 PRINT "  JACK"
```

When this program is run, it prints the message, I LOVE YOU!, on the screen 12 times, followed by JACK; a computerized Valentine. A flowchart for this program is provided in Fig. 5-2.

The PRINT command at line 70 has been indented several extra spaces so you can quickly identify the steps in the loop. This is a simple thing to do, and it lets you immediately identify the operations in the loop. If you decide to write programs that use loops, use the indentations. It helps keep track of what's going on.

Experiment No. 5-4. The FOR-NEXT Loop

This experiment will show you how the FOR-NEXT command can be used in programs to count passes through loops. Clear your computer for a new program and enter the following program:

```
10  FOR TR = 1 TO 10
50      PRINT "NAME"
90  NEXT TR
```

Run the program. What happens? You should see the message, NAME, printed on the TV screen 10 times. Go back and change line 50 so the computer will print your first name. Don't use more than 10 or so characters. Run the program again to be sure it works correctly.

How can you change the program so the computer goes through the loop 100 times? Just set the upper limit of the loop count to 100. Change line 10 in your program so that it is now:

```
10  FOR TR = 1 TO 100
```

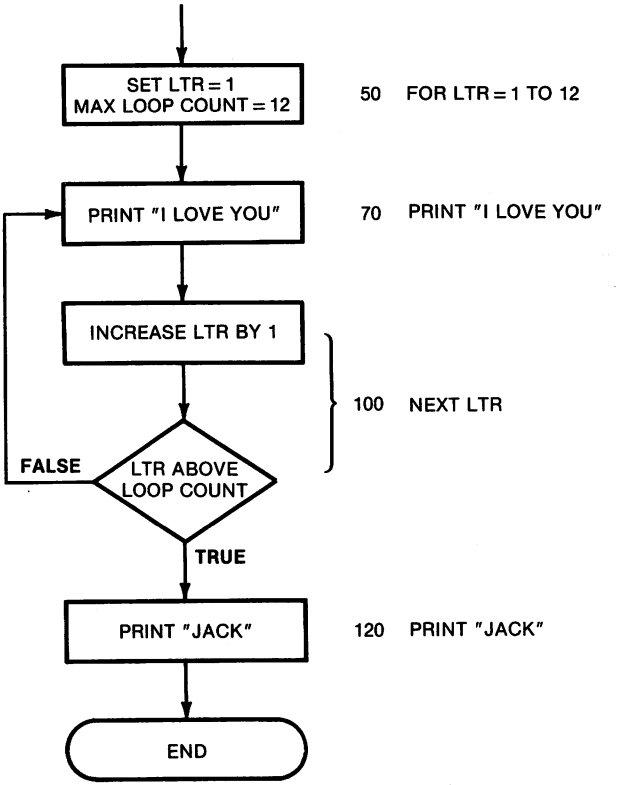


Fig. 5-2. Flowchart for a FOR-NEXT operation.

Run the program. Did the computer display your name all 100 times? Did you count them all? It's probably difficult to tell, since the display moved so quickly. It would be difficult to try and count names as they flash by.

You can display the value of the loop count along with the name, since it has been given a label. This will give you a way to keep track of the number of passes the computer has made through the loop. Add the following line to your program so that the loop count is displayed on the TV screen:

```
60 PRINT TR
```

Here is a listing of the complete program for you:

```
10  FOR TR = 1 TO 100
50    PRINT "NAME"
60    PRINT TR
90  NEXT TR
```

Run the program. Can you see the numbers as they flash by on the TV screen? They are hard to see, but 100 should be the last number displayed.

To convince you that the computer made 100 passes through the loop, try this: run the program again, but right after you start it, press the CTRL key and hold it down. What happens to the TV display? The display is slowed down so you can see what is happening on the TV screen. You can press the CTRL key to slow down the rate at which information is displayed on the TV screen.

You can also slow the TV display by giving the computer something else to do. Add the following line to your program:

```
70  FOR P = 1 TO 120 : NEXT P
```

Now run the program. Don't press the CTRL key this time. What has happened? The display has been slowed so you can read the numbers on the screen. Do you know why the added program line slowed the display? The added program line is a loop, too, but the FOR and NEXT operations have been put on one line. Here's what they look like when they are written one over the other:

```
70  FOR P = 1 TO 120
80  NEXT P
```

This loop doesn't have anything to do between the FOR and NEXT commands, so it's often called a "do-nothing" loop. It just causes the computer to go through this simple loop 120 times. Although the computer can do things quickly, making it do "nothing" 120 times slows it down quite a bit. This type of do-nothing loop is also called a *time-delay* loop, and it is often used to slow the computer so you can keep up with it.

Let's do one last thing with this program. Add a command so you can enter the number of loops that the computer is supposed to do. Here is the line to put in your program:

```
5  INPUT "LOOPS"; LP
```

Now change line 10 to:

```
10  FOR TR = 1 TO LP
```

List your program. It should look like this:

```
 5 INPUT "LOOPS"; LP
10 FOR TR = 1 TO LP
50   PRINT "NAME"
60   PRINT TR
70   FOR P =1 TO 120:NEXT P
90 NEXT TR
```

The number of passes that you want the computer to make through the loop is now entered from the keyboard. Run the program and type in a value for LOOPS. Since the time delay FOR-NEXT line (line 70) is still in your program, it will take quite a while for the computer to make a million passes through the loop, so keep your number below a thousand.

Can you take the time delay loop out of your program? You should be able to. How do you stop the program before all the passes have been done?

A REAL USE FOR A LOOP

There are many times when an *average* of several values is needed. Remember that the *average* of several values is simply the sum of the values divided by the *number of values used.* This type of calculation is used to figure an average of grades for students who are graded with numbers; 97, 75, 88, and so on. It could also be used to calculate the average balance in a checking account, the average cost of groceries over several months, the average cost of gasoline for a year, and so on. There are many uses for this type of program.

The following program can be used to figure or "compute" the average of values that are typed in from the keyboard. Let's say that we're averaging a family's grocery bills for a month. Since the number of bills will be different from month to month, it is helpful to be able to type in the number of bills that are to be averaged. This program is very flexible and you can use it to average many other things. If you want to, you can load it into your computer and try it. You don't have to load the REM line, line 15:

```
10 PRINT" ▪ "
15 REM THIS CLEARS THE SCREEN
20 INPUT "NUMBER OF VALUES ="; NV
30 SUM = 0
```

```
40  FOR LP = 1 TO NV
50    INPUT "VALUE ="; V
60    SUM = SUM + V
70  NEXT LP
80  PRINT "AVERAGE IS " SUM / NV
90  GOTO 20
```

If you decide to try this program, try averaging the following three values to test the program: 85.90, 97.50, and 68.72. The average is 84.04.

Several important things happen in this program. It lets you type in the number of values that are to be averaged. This number is labeled NV and it is used in the FOR command at line 40. This sets up the loop counter and also the maximum number of passes through the loop, or NV. The values to be averaged are input at line 50. Once a value has been input, it is added to the old sum and the new total is labeled SUM. To finally get the *average value*, the SUM is divided by the number of values, NV, at line 80. This line also prints the average on the TV screen.

NESTED LOOPS

You have probably used sets of measuring cups that could be placed one inside the other to make a neat stack that can be easily stored. Program loops can be placed one inside the other, too. Let's look at a program that uses two loops, one inside the other. In this program, one loop simply prints the numbers 1 through 10 on the TV screen, while the other loop prints five asterisks between each number. Here is the program.:

```
10  FOR N = 1 TO 10
20    PRINT N
30    FOR T = 1 TO 5
40      PRINT "*";
50    NEXT T
60    PRINT
70  NEXT N
80  GOTO 10
```

Let's look at this program, starting with the inner loop that prints the five asterisks. Here it is:

```
30    FOR T = 1 TO 5
40      PRINT "*";
```

```
50    NEXT T
60    PRINT
```

A flowchart for this program is shown in Fig. 5-3. The FOR command sets up the loop for five passes, and the computer prints five asterisks on the same line. The semicolon keeps the printing on the same line. The PRINT command at line 60 is not part of the FOR-NEXT loop, and it doesn't "print" anything. It's just used to return the cursor to the left side of the line below so that the next number will be printed on a "clean" line.

Now let's take this loop out of the program and look at what's left:

```
10  FOR N = 1 TO 10
20    PRINT N



70  NEXT N
80  GOTO 10
```

This is a simple loop, too, and it is shown in the flowchart in Fig. 5-4. The loop has been set up for 10 passes, and since the label N has been used to identify the loop count, you can print the loop count, too. The GOTO command at the end of the loop simply tells the computer to go back and start the program over again. A complete flowchart for the program is shown in Fig. 5-5.

When loops are used this way, one inside another, they are said to be "nested," and programmers use nested loops often. The "smallest" loop is called the "inside" loop and the "largest" loop is called the "outside" loop. Since an indented program listing was used in this example, it shouldn't be hard to tell the loops apart.

Let's look at another program that uses nested loops, but if you've had enough of nested loops, feel free to go ahead to the next section.

Although the next program looks complicated, it just uses two nested loops like those shown in the previous example. There are just more things done in each loop. This program was written by a housewife who wanted to do more than just average her food costs each month. This program also tells her the total amount spent on groceries over several months and the average amount spent each month.

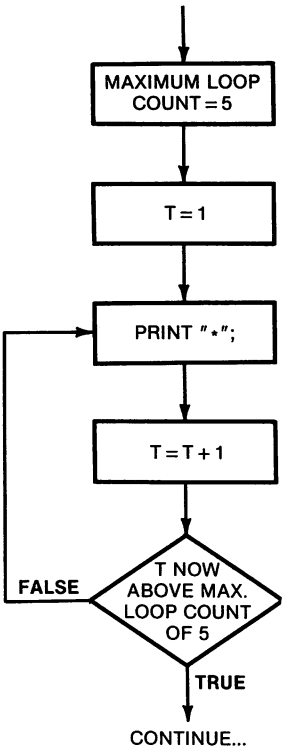


Fig. 5-3. Flowchart for the "five-asterisk" loop.

The illustration in Fig. 5-6 shows what the program calculates for the three monthly grocery bills. Here is the program. Although it looks complicated at first, you can break it down into smaller pieces if you want to look at it in detail:

```
10 PRINT "GROCERY PROGRAM"
100  INPUT "NUMBER OF MONTHS"; NM
120  TTL = 0
140  FOR M = 1 TO NM
160    PRINT "NUMBER OF BILLS"
180    INPUT "THIS MONTH"; NB
200    SUM = 0
220    FOR B = 1 TO NB
240      INPUT "AMOUNT ="; AMT
```

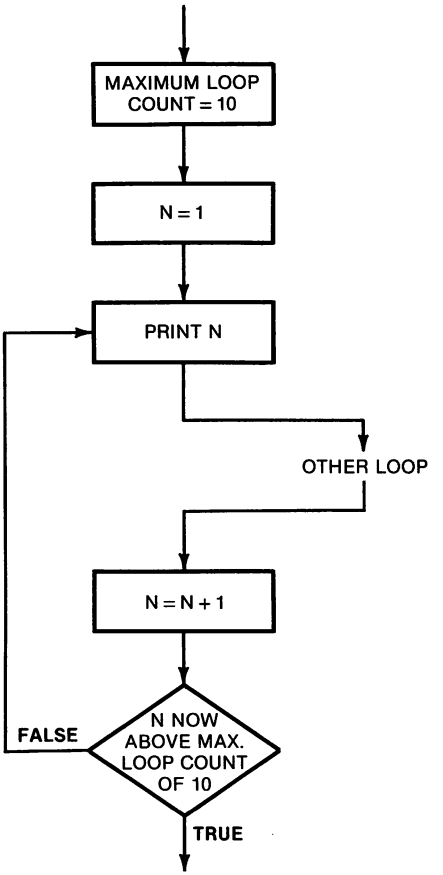


Fig. 5-4. Flowchart for the main printing loop.

```
260       SUM = SUM + AMT
280     NEXT B
300     PRINT "MONTHLY AVERAGE"
320     PRINT "IS " SUM / NB
340   TTL = TTL + SUM
360   NEXT M
380   PRINT "TOTAL OF ALL"
400   PRINT "BILLS IS " TTL
420   PRINT "AVERAGE MONTHLY"
440   PRINT "COST IS " TTL/NM
```

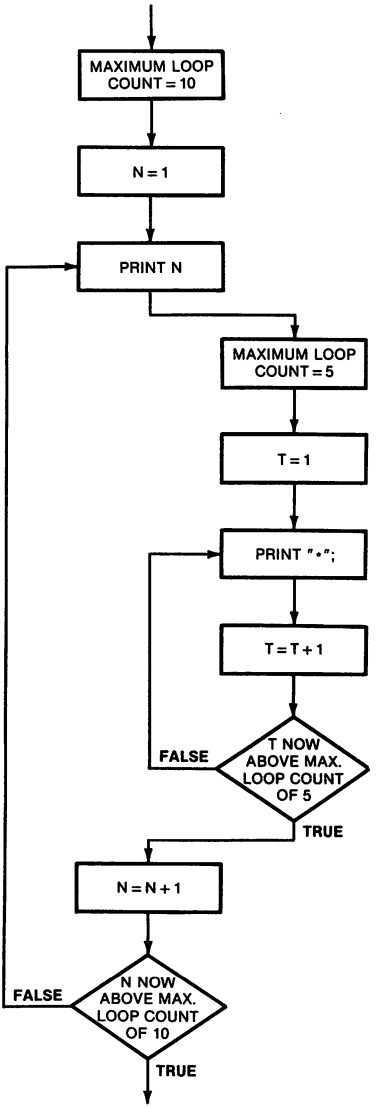


Fig. 5-5. Flowchart for the complete printing program.

TAKE ONE GIANT STEP

Let's look at a slightly different use of FOR-NEXT loops. In the FOR-NEXT loops that you've seen so far, a 1 is added to the loop count each time the computer passes through the loop. Well, there

```
RUN
GROCERY PROGRAM
NUMBER OF MONTHS? 3
NUMBER OF BILLS
THIS MONTH? 4
AMOUNT =? 76.98
AMOUNT =? 57.75
AMOUNT =? 78.90
AMOUNT =? 54.23
MONTHLY AVERAGE
IS 66.965
NUMBER OF BILLS
THIS MONTH? 2
etc...


TOTAL OF ALL
BILLS IS 671.95
AVERAGE MONTHLY
COST IS 223.98
```

Fig. 5-6. Typical printed output from the grocery-bill program.

may be times when you'd like to add some other value to the count when the computer goes through the loop. You can do this simply by telling the computer what value you want added to the count. Here's an example:

```
100  FOR TR = 1 TO 200 STEP 3
```

In this case, the computer adds 3 to the count after each pass through the loop. So, the loop count would be 1, 4, 7, and so on. If you don't tell the computer what size "step" is being used, it simply adds 1 to the count after each pass through the loop.

Since the maximum loop counter value in this example has been set to 200 in the FOR command, what happens as the loop count increases toward 200? When counting by 3s, the count looks like this:

```
... 187, 190, 193, 196, 199
```

In this case, the computer gets out of the loop after reaching a count of 199, since the next value, 202, is greater than 200. Remember that a maximum loop count of 200 was set up in the FOR command.

Do you want to count to 100,000 by 17s? Here's a program that does it for you:

```
100  FOR C = 1 to 100000 STEP 17
110    PRINT C
120  NEXT C
```

The computer quits at 99995, since the next value (100012) puts it over the 100000 mark.

Loops don't have to start with a count of 1. Any other value can

be used as long as the maximum value is greater than the starting value. The following program goes through this loop *six* times:

```
250  FOR TX = 15 TO 20
260    PRINT "HAVE FUN"
270  NEXT TX
```

Remember, the loop counts would be 15, 16, 17, 18, 19, and 20!

Loops can count down, too. Here's an example that we'll let you think about:

```
1020  FOR XZ = 100 TO 50 STEP -1
1030    PRINT XZ
1040  NEXT XZ
```

In this program, the loop count starts at 100. This is *higher* than the final loop count, which is set at 50. How can this be? Well, the step has been set at −1, which simply means that the loop count will have *1 subtracted from it each time the computer goes through the loop*. So, the first count is 100, the next count is 99, and so on. The loop is finished when the loop count goes down to 50.

SUBROUTINES TO THE RESCUE

We all do routine jobs, again and again, but at different times and different places; making a bed, fixing a sandwich, and mowing a lawn are typical things that we do frequently. Many computer programs have operations that do the same thing, but at different times and places in a program. Let's look at an example.

A cleaning service uses a computer program to do these three things:

1. Print bills to be sent to customers.
2. Print letters typed by a secretary.
3. Print envelopes.

The computer prints the company's name and address on each bill, letter, and envelope. Do the three parts of the program each have instructions to do this printing? Probably not, since it is wasteful of both the computer's time and the programmer's time to duplicate the same program steps three separate times. It is better to put the part of the program that prints the name and address in one place and let the different parts of the program share it. This is shown in Fig. 5-7. You can see that when the printing subroutine is shared, the program is shorter and any errors are limited to one part of the program.

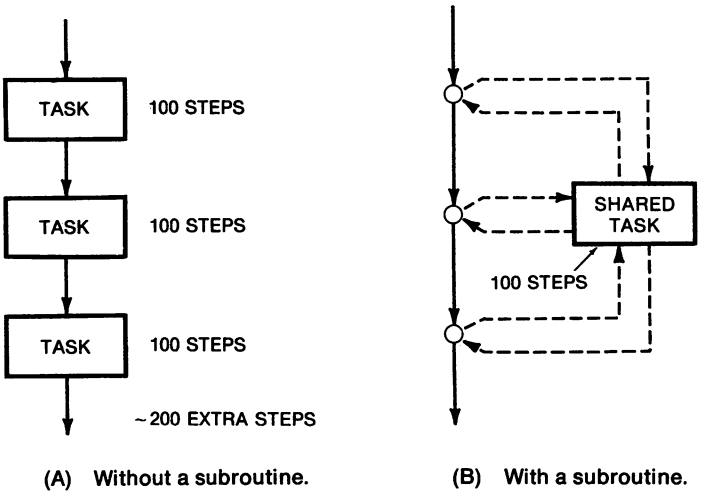


Fig. 5-7. Using a subroutine saves program space and limits errors.

Although our example may be a bit simple, this program-sharing technique is common. When programmers are writing long complex programs, they like to save as much time and energy as possible, so they look for tasks that can be shared. These shared tasks are called *subroutines*, and they provide a way of simplifying and shortening programs.

Here is another example of how a subroutine could be used. A hardware store uses a computer to keep track of the products that it sells. The products have been given part numbers between 1 and 10000, so that the computer can tell one from another. The product numbers are used by different parts of the hardware-store program, so a *subroutine* is used to get the product number from the computer keyboard whenever it is needed. Simple INPUT commands could have been used throughout the program, but the programmer decided to check the number to be sure it is in the range 1 to 10000. This takes more program steps, so why spend time duplicating these steps (and any errors) throughout the program, when the program can share one set of instructions that is easy to get to? Here's what the steps might look like:

```
2000  PRINT "TYPE THE PART NUMBER"
2020  INPUT PN  :  PN = INT(PN)
2040  IF (PN < 1) OR (PN > 10000) THEN PRINT
      "OUT OF RANGE": GOTO 2000
```

When a number is typed in, the computer uses the INT command to remove any decimal fractions, and the IF-THEN operation checks to be sure the number is between 1 and 10000. If it is outside this range, the message "OUT OF RANGE," is printed and the user gets another chance to type it in. This "traps" any incorrect numbers. Other typing errors could be checked for, too.

Whenever the programmer needs a part number for a task, he can "tell" the computer to go to this number-input subroutine and get it. But how is the computer pointed to this subroutine? Programs use a GOTO command to point the computer to a specific program line, and there is a similar instruction for subroutines. This is the go-to-subroutine, or GOSUB instruction, and a line number is used with it to tell the computer where the subroutine is. Unlike a GOTO command that simply tells the computer, "Go to a specific place," the GOSUB command tells it, "Go, but remember to come back when you're done."

A RETURN instruction is used at the end of every subroutine to tell the computer that it's time to go back to the main program. Here's what the complete number-input subroutine would look like:

```
2000  PRINT "TYPE THE PART NUMBER"
2020  INPUT PN  :  PN = INT(PN)
2040  IF (PN < 1) OR (PN > 10000) THEN PRINT
      "OUT OF RANGE": GOTO 2000
2060  RETURN
```

And here is how the main program would point the computer to it:

```
250  GOSUB 2000
260  etc. . .
```

Now, whenever the programmer needs to have the user type in a part number, he can simply use the GOSUB 2000 command to get it. He doesn't have to retype all of the input and error-checking steps. Subroutines are very useful to programmers.

Let's do an experiment to see how a subroutine can be used in a program to save programming time.

Experiment No. 5-5. Using a Subroutine

In this experiment, you'll see how a subroutine can be used for a simple task, and how the subroutine can be used at several places

in a program. Clear your computer for a new program, and enter the following subroutine:

```
2000 FOR P = 1 TO 22
2010   PRINT "*";
2020 NEXT P
2030 RETURN
```

Do you know what this subroutine will do? Let's just run the subroutine and see. Type GOSUB 2000 [RETURN] and see what happens. You should see a line of asterisks across the TV screen. If the asterisks go down the screen instead, you forgot the semicolon at the end of the PRINT command at line 2010.

Now let's use the subroutine in a program. Add the following lines to the ones already in the computer:

```
100 FOR L = 1 TO 10
110   PRINT L
120 NEXT L
130 GOSUB 2000
140 END
```

Run the program. You can type RUN or RUN 100. You can give the computer a specific line number in RUN, GOTO, or GOSUB commands that you type in from the keyboard if you want the computer to start at a particular place in a program, rather than at the lowest line number.

What is displayed on the TV screen when the program is run? You should see the numbers 1 through 10 displayed down the left side of the TV screen, with a row of asterisks across the screen below them. Now add the following lines to your program:

```
140 FOR Q = 1 TO 3
150   PRINT "VIC 20"
160 NEXT Q
170 GOSUB 2000
180 END
```

The complete program, including the subroutine, should look like this:

```
100 FOR L = 1 TO 10
110   PRINT L
120 NEXT L
130 GOSUB 2000
```

```
140  FOR Q = 1 TO 3
150    PRINT "VIC 20"
160  NEXT Q
170  GOSUB 2000
180  END

2000  FOR P = 1 TO 22
2010    PRINT "*";
2020  NEXT P
2030  RETURN
```

Run the program. You should see a display like this on the TV screen:

```
1
2
3
4
5
6
7
8
9
10
**********************
VIC 20
VIC 20
VIC 20
**********************
```

In this experiment, the subroutine has been used to do a simple task; print 22 asterisks across the TV screen. However, it saved considerable programming time and effort, since the same asterisk-printing instructions were used twice in the program. This meant that you didn't have to duplicate or copy the program lines that printed the asterisks across the TV screen when they were needed again. You simply used the GOSUB 2000 command.

Can you think of other tasks that might be done by using subroutines instead of writing the same program steps over and over again? Special math operations might be put in subroutines; for example, figuring sales tax.

Subroutines are very useful, even in short programs where it is necessary to have the computer do the same thing again and again.

The task might be as simple as printing a line of asterisks across the screen, or it might be as complex as figuring how to point a telescope at a particular star in the night sky.

LOTS OF INFORMATION

There are many times when we need to have the computer store more than one or two pieces of information. In all of our previous examples, we have shown you how the computer uses labels to identify information and save it for use somewhere else in a program. This has worked out well so far, but there is a limit to the amount of information that can be stored this way.

Let's assume that we are running tests on flashlight batteries to see how long they last. We load the batteries into a flashlight and leave it on. We want to keep a record of how long it takes for the light to go out. If we write the results on paper, they might look like this:

Battery type 1,	5½ hours
Battery type 2,	4¾ hours
Battery type 3,	6 hours

How can we use the computer to store the test results for eight kinds of batteries? A program could be written using eight different labels; B1, B2, B3, and so on. That seems to do the trick, but we still have to get the information into the computer. Since eight different labels have been used, one for each type of battery, we'll need *eight INPUT commands*:

```
160 INPUT "B1 VALUE"; B1
180 INPUT "B2 VALUE"; B2
200 INPUT "B3 VALUE"; B3
220 INPUT "B4 VALUE"; B4
240 INPUT "B5 VALUE"; B5
260 INPUT "B6 VALUE"; B6
280 INPUT "B7 VALUE"; B7
300 INPUT "B8 VALUE"; B8
```

Imagine testing 100 batteries and trying to save the test times this way! Computers would lose a lot of their power if we had to set up programs with many input commands such as those shown. There is an easier way to store this kind of information.

The computer can easily store a lot of similar information, and you can think of it as being stored in a chest of drawers, as shown

in Fig. 5-8. Each drawer has been numbered and can hold a piece of information. The chest is given a "name" so you can identify *all* of the information stored in it.

Computers use a *table* to store this type of information, and it is given a name, too. Individual pieces of information in the table are given a number to identify them. So, each piece of information in a table can be identified using the name of the table and the number where it is located. Here's what a table might look like:

TABLE "BAT"—BATTERY TEST TIMES

Battery Type	Test Time
1	5½ hours
2	4¾ hours
3	6 hours

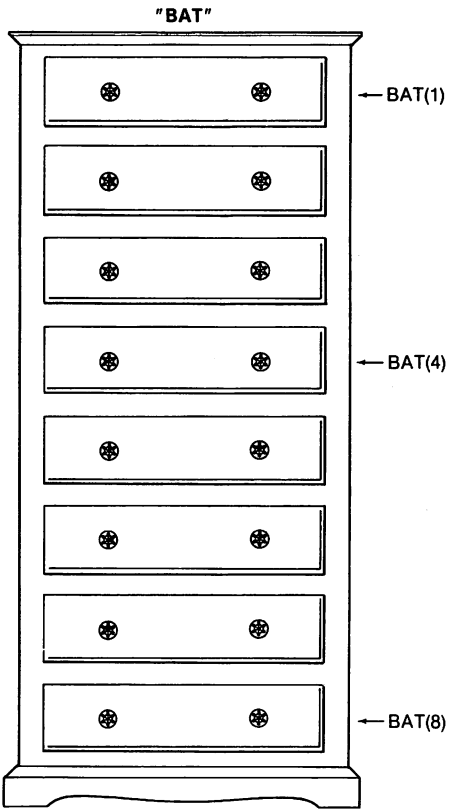


Fig. 5-8. An array of information is organized like a set of drawers.

This table has been named BAT, and the test information has been put in it as BAT(1), BAT(2), BAT(3), and so on for each type of battery tested. The number in parentheses identifies or "locates" each piece of information. When the computer "sees" a label that is followed by a number in parentheses, such as BAT(7), it knows that this information is in a table called BAT. This kind of table is also called an *array of information*, or just an *array*. This simply means that there is an orderly arrangement of the information.

However, it might still look as though individual INPUT commands are needed to get this information into the computer:

```
160  INPUT "B1 VALUE"; BAT(1)
180  INPUT "B2 VALUE"; BAT(2)
```

and so on.

When an *array* or a *table* is used, you can get at the different pieces of information by using the name of the table and a *label* in place of the number in the parentheses. For example, the command:

```
20  INPUT BAT(1)
```

does the same thing as this one:

```
10  TEST = 1
20  INPUT BAT(TEST)
```

The only difference is that a label has been used in place of the number in the second set of instructions. Since a label has been used, you can use many different types of instructions to change its value. In this way, the computer can quickly locate a specific piece of information in the table, or array.

Here's an example of how an array can be used to store eight values entered from the keyboard. The program has been kept simple:

```
10  FOR TEST = 1 TO 8
20          INPUT BAT(TEST)
30  NEXT TEST
```

You should be familiar with the FOR-NEXT loop, and you can see that the computer will go through this loop eight times. When the computer goes through the loop the first time, what is the value of the loop counter labeled TEST? The value is 1, so at line 20 the computer is really doing this:

```
20        INPUT BAT(1)
```

When the computer goes through the loop the next time, the loop counter is equal to 2, so the second value input is put into the table called BAT, and it is labeled BAT(2). The computer goes through this loop eight times, storing eight individual values in the table called BAT. They are stored as BAT(1), BAT(2), and so on up to BAT(8). You can get them out of the table just as easily. Let's do an experiment to see how the computer can store and get values using a table.

Experiment No. 5-6. Using an Array or Table

In this experiment, you will find out how the computer can be used to store an array or table of information, and you will use a simple program to store an array in the computer.

Clear your computer for a new program, and enter the following program:

```
10  FOR TR = 1 TO 8
20        INPUT BAT(TR)
30  NEXT TR
```

Now run the program. When the computer displays a question mark on the screen, enter the value 1. Remember to press [RETURN]. Now enter values of 2, 4, 8, 16, 32, 64, and 128 for the other question marks. Your TV screen should look something like this:

```
RUN
? 1
? 2
? 4
? 8
? 16
? 32
? 64
? 124

READY.
■
```

The eight values are stored in the array labeled BAT. How can you display one of these values on the TV screen? You can simply

type PRINT BAT(4) to get the fourth value printed on the TV screen. Do this. What is printed on the TV screen? The number 8 is displayed. You can display the other values this way, too. Just use a PRINT command followed by the label for the array and the location of the value in parentheses. Try to display some of the other values. Can you do it? You should be able to.

Let's add onto the program so that the computer prints the values on the TV screen and also gives us the sum of the values. Here is the complete program. The lines with an asterisk in front of them are the new ones. Don't type in the asterisks, though:

```
  10 FOR TR = 1 TO 8
  20       INPUT BAT(TR)
  30 NEXT TR
* 40 SUM = 0
* 50 PRINT "VALUES ARE:"
* 60 FOR X = 1 TO 8
* 70       PRINT "VALUE" X "IS " BAT(X)
* 80       SUM = SUM + BAT(X)
* 90 NEXT X
* 100 PRINT "TOTAL IS " SUM
```

Run this program. After you type in the eight values, the computer displays them for you. It also gives you the "location" of the value in the table. The sum is also displayed below the values.

Run the program again, but use some different numbers. You can check your results by using a calculator, if one is available. Can you change the program so that the array is set up to store only five values? You can try this if you want to.

This experiment shows you how the computer can be used to store information in a table, and you have seen how the computer can recall the information that it needs. You should be able to see that using an array is much easier than using individual labels and INPUT commands to type in many pieces of information.

THE SIZE OF YOUR ARRAYS

When arrays are used, they are given a name, and a number is used to "locate" a stored value. The number is put in parentheses at the end of the label or name. When a number is used this way with the label or name of an array, it is called a *subscript*. In the VIC 20, you can have arrays with *subscripts* from 0 through 10, so 11 pieces of information can be stored. In the previous examples,

arrays started with the subscript 1, since it's easy to remember that the first value is XYZ(1). Of course, if you can only put 11 pieces of information in an array, arrays aren't going to be very useful.

You can have larger arrays, *but you must tell the VIC 20 the maximum size of the array* before you try and use it. To set up a large array, you use the dimension command, DIM, and follow it with the name of your array and the *maximum* subscript that you plan to use. For example, an array named CHECKS with 100 pieces of information would be set up with this command:

```
10 DIM CHECKS(100)
```

Actually, 101 places in the array can be used, but let's assume that we start storing information at CHECKS(1), instead of at CHECKS(0). If you see a DIM command in a program, you'll understand that the programmer is simply telling the computer that an array is being set up to store information. This doesn't mean that you *must* use all 100 locations in the CHECKS array for information. At some times, the array may contain only a few pieces of information, while at others it may be filled with all 100. The DIM command just tells the computer the array's maximum size.

You can also set up several arrays at the same time; for example:

```
10 DIM CHECKS(50), AMOUNTS(50), DATES(50)
```

This statement sets up three arrays that can hold up to 50 pieces of information in each. Perhaps a checking account program is going to use arrays to store information. The arrays are always set up with a DIM command right at the start of the program. In fact, this is the first thing done in a program if an array is going to be used. Remember that you can set up and use an array with a subscript of up to 10 without using the DIM command. If you want to store more information, you'll have to use the DIM command.

A SPECIAL KIND OF ARRAY

Arrays or tables are very useful, since they let us store a lot of information in the computer. Let's assume that we want to store the weights of the nine planets, Mercury through Pluto, in the computer so they can be used in an action game for kids. A loop could be set up so that the weights are typed in at the start of each

game, but it would be a real bother to type them in each time the game is run. You could set up the weights in an array right at the start of the program, for example:

```
10  PLANET(1)=123456 : PLANET(2)=456789
```

and so on.

This can be a bother, too, and there's a much better way to store this kind of information in a program. You can simply use a DATA statement in your program and put the weights of the planets in a single line:

```
100  DATA 123456, 456789, 1298098
```

and so on.

This simply tells the computer, "store this information for later use, but don't give it a specific label." By the way, most programmers put DATA statements at the end of their programs.

Now that the information has been stored in a DATA statement, you can get at it with a READ operation or command. The READ command simply reads each piece of information from the DATA statements as if it were reading it from a list of values. Once the information has been read, you can give it a label or use it in any way you want. When you use the READ command again in your program, *the second piece of information is read from the DATA statement.* In this way, you can put information right in your programs and use it one piece at a time. Let's do an experiment to see how this works.

Experiment No. 5-7. Using the DATA and READ Commands

The purpose of this experiment is to show you how the DATA and READ commands can be used. Clear your computer for a new program and type in the following program:

```
10  FOR T = 1 TO 4
20        READ QX : PRINT QX
30  NEXT T
40  DATA 123, 456, 789, 88
```

Run the program. Are the data values displayed on the TV screen? They should be. You can change the four values in the data statement if you want to and run the program again to see them printed on the TV screen.

The values in the DATA statement are really part of your pro-

gram, and you cannot change them except by typing in a new program line. Change line 10 in the program to:

```
10  FOR T = 1 TO 5
```

Run the program. Are the values printed? What else is displayed on the TV screen? You should see an "error message" printed on the screen, since the FOR-NEXT loop tries to READ five values from the DATA statement, and there are only four values in it. The computer displays the following:

```
?OUT OF DATA
 ERROR IN 20
READY.
■
```

You cannot read more information from a DATA statement than is really there.

DATA statements are very useful, since they let you store a great deal of information in the computer for later use. For example, if you are using the computer to play a space-war game, the weight of each planet and its size can be stored in a DATA statement. Two READ commands are used to read them. Here's another example. Instead of using values, we'll just show you what would be stored in the DATA statement. Of course, the values would be used in a real program:

```
50  READ WEIGHT : READ SIZE
60  DATA Weight of Mercury, Size of Mercury, Weight of Venus,
      Size of Venus, etc . . .
```

The first READ command reads the first value in the DATA statement, *Weight of Mercury*, and labels it WEIGHT. The next READ command reads the second value, *Size of Mercury*, and labels it SIZE.

When the computer comes back to the READ commands, the first READ command reads the third value in the DATA statement, *Weight of Venus*, and labels it WEIGHT. The next READ command reads the fourth value, *Size of Venus*, and labels it SIZE. In this way, the weight and size of the planets are stored together, each being read in order by a different READ command.

Notice that you can assign any labels you want to the values the computer gets in the READ command or commands. However, when DATA and READ commands are used, programmers must pay very careful attention to how the information is stored and how it is read.

When a game or other program has finished, it may use a GOTO command to start over again. If this is done, the programmer must point the computer back to the start of the DATA statement, so it will "read" through the information again. The RESTORE command does this. For example:

```
 10 READ RX : PRINT RX
 20 RESTORE
 30 READ RX : PRINT RX
 40 READ RX : PRINT RX
 50 READ FZ : PRINT FZ
100 DATA 12, 67, 99
```

In this example, the first READ RX : PRINT RX operation prints the value 12 on the TV screen. The computer gets ready to go on to the second value in the DATA statement, but the RESTORE command says, "No, go back to the first value." The READ RX : PRINT RX commands at line 30 print the value 12 again. The following READ command at line 40 gets the value 67 and it is printed. The READ command at line 50 gets the value 99 and it is printed, too. Note that this value is labeled FZ. You can use any labels you want to get information from a DATA statement.

The RESTORE command simply tells the computer to go back to the first value in the DATA statement. This is useful if you want to go through the information again.

For long lists of information, you can use multiple DATA statements in your program. Here is an example:

```
200 DATA 200, 547, 6789, 2, 512, 43
220 DATA 400, 600, 53, 1, 0, 0, 45
```

The VIC 20 sees this as a *continuous list* of 13 values.

QUESTIONS

1. You can use special "commands" in the message part of a PRINT command to get the computer to do keyboard-type operations. How would you get the cursor to its home position without clearing the screen?

2. How would you use two PRINT commands in a program to print something on a single line on the TV screen?

3. What is the TAB command, and how is it used?

4. What do you think would happen if you tried to do a PRINT TAB(15) "FUN"; and then a PRINT TAB(8) "TIME" on the same TV screen line?

5. What does the ON command do? What is tested by this decision-making command?

6. When would you use the ON command in a program?

7. What does the FOR-NEXT command do?

8. Can you use steps other than 1 with a FOR-NEXT command? What kinds of steps can be used?

9. What is a nested loop?

10. What is a subroutine? How might it be used?

11. What is an array? How is it used?

12. Why are arrays so flexible? How big can an array be?

13. What are the READ, DATA, and RESTORE operations used for?

PROBLEMS

1. Write a program that draws a diagonal line from the top left-hand side of the TV screen to the bottom right-hand side of the screen. (HINT: To draw the diagonal line, use the graphics character produced by a [SHIFT] M. This graphics character will be enclosed in quotes in a PRINT statement.)
2. Modify the solution to Problem 1 so that the line is drawn from the top right-hand side of the screen to the bottom left-hand side of the screen.
3. Write a program that flashes the message "THIS MESSAGE GETS YOUR ATTENTION" on the TV screen. The message should flash on and off between 1 and 3 times every second.
4. Write a program that causes the "heart" character ([SHIFT]S) to move from the center of the left-hand side of the TV screen to the center of the right-hand side of the TV screen. Once the heart is on the right-hand side of the screen, the movement of the heart from left to right should be repeated.
5. Write a program that causes the following to be displayed on the TV screen:

```
ADD          - 1
SUBTRACT     - 2
MULTIPLY     - 3
DIVIDE       - 4

COMMAND ?
```

Once you enter a command number (between 1 and 4), the computer will ask you for two numbers, which will then be added, subtracted, multiplied, or divided, depending on the command (number) that you entered first. Once the result is displayed, the program should be repeated. For example,

```
ADD          - 1
SUBTRACT     - 2
MULTIPLY     - 3
DIVIDE       - 4

COMMAND ? 1

NUMBERS ? 3,4
RESULT IS 7

ADD          - 1
SUBTRACT     - 2
MULTIPLY     - 3
DIVIDE       - 4

COMMAND ? 3

NUMBERS ? 3,5
RESULT IS 15
```

6. Write a program that displays a multiplication table that contains all possible combinations of the numbers 1 through 3 times the numbers 10 through 13. Assume that multiplying 2 times 10 is the same as multiplying 10 times 2.

7. Write a program that calculates and displays all of the prime numbers between 1 and 100. A prime number is a number that can only be exactly divided by itself and by 1. Some examples of prime numbers are 2, 3, 5, 7, and 11. Nonprime numbers include 4 (which can be divided by 4, 1, *and* 2) and 12 (which can be divided by 1, 3, 4, 6, and 12).

8. Write a program that lets you enter between 2 and 20 numbers. These numbers are stored in the computer so that it can calculate and display the average of all the numbers (the sum of the numbers divided by the number of numbers entered), the lowest number entered, and the highest number entered. No calculations must be performed on the numbers as they are

entered into the computer. All calculations must be performed after all numbers have been entered.

9. Write a program where a single "die" is thrown 100 times (there are two "die" in a pair of dice). Keep track of how many times a 1, 2, 3, 4, 5, and 6 is thrown, and at the end of 100 throws, display the total number of times each number was "thrown."

10. Write a program that is used to keep track of car sales for each of five salesmen. At the end of a month the computer should display the name of each salesman, along with the number of cars sold.

11. Write a subroutine that draws a playing card from a deck of cards. After the card has been drawn, the computer displays its suit (heart, club, diamond, or spade) along with its value of either 2 through 10, Jack, Queen, King, or Ace. (HINT: Assume that each card is given a value of from 1 to 13. Once a card is "drawn," somehow "mark it" so that it can't be drawn again.)

12. Write a program that lets you enter an amount of money in dollars and cents. The computer then displays the equivalent number of French francs, German marks, and English pounds. Assume that there are 7.49 francs, 2.48 marks, and .642 pounds per dollar. These values should be stored in a DATA statement, along with the name of the country and its unit of currency.

6

Strings and Things

In previous chapters, you saw how the computer used values in math and decision-making operations. Computers can process values very quickly, doing thousands of math operations each second, and this is exactly what makes them so useful and powerful. Computers process strings quickly, too; sorting names on address lists, locating information in library records, and printing business reports. In this chapter, you'll see what the computer can do with strings; combine them, take them apart, and use them to make decisions.

Strings are simply groups of letters, numbers, punctuation marks, and other symbols, all of which we'll put under the heading of "characters." Strings such as "TEST REPORT 12" cannot be given a numeric value, and this type of information cannot be processed by "multiplying" or "adding" it to some value. The VIC 20 computer stores strings in almost the same way it stores values. Both are assigned a label so that the computer can identify them for later use. Here are some examples of strings and the labels assigned to them:

```
NAME$ = "JIMMY SMITH"
DATE$ = "MAY SIXTH"
UNIT$ = "KILOGRAMS"
```

Each string label ends with a dollar sign, and the characters in the string are enclosed in quotes. As we mentioned earlier, the last quote mark isn't really needed, but we've used it since it quickly shows the end of a string. Since the VIC 20 recognizes only the first two letters (or letter and number) of a label, the preceding labels and strings are the same as these:

```
NA$ = "JIMMY SMITH"
DA$ = "MAY SIXTH"
UN$ = "KILOGRAMS"
```

The VIC 20 can print a value on the TV screen by using a PRINT command and a label, and it can print a string in the same way. Here are two program lines that will display the word, KILOGRAMS, on the TV screen:

```
10 UNITS$ = "KILOGRAMS"
20 PRINT UN$
```

You can also use the computer to input a string of characters and store it for later use. The INPUT command does this, but you must remember to use a string label in the command. Here is how the INPUT and PRINT commands could be used together in a program:

```
250 INPUT R$
270 PRINT R$
```

When this program is run, individual characters are typed in and then the RETURN key is pressed. Pressing the RETURN key tells the computer that you have finished typing in your information, letting it go on to the next program step. The string of characters typed in is labeled R$, and it is can be used later in the program.

You can also put messages in the INPUT and PRINT commands that are used with strings to make them more useful. Here is another example:

```
250 INPUT "TYPE YOUR STRING"; R$
270 PRINT "THE STRING IS. . . " R$
```

USING STRINGS

The computer can do many useful things with strings, and we'll start by looking at how the computer puts strings together. You

can think of the strings as characters written on strips of paper. By using some adhesive tape, you can connect these "strings" in many ways (Fig. 6-1). In the VIC 20, strings are connected, or linked, by using the plus sign (+). The strings are not "added" to one another, since this would be meaningless. When the plus sign is used with strings, it simply means that the strings are *linked* to form a new string. Here is an example that shows how this works:

```
10  FIRST$ = "KAREN"
20  LAST$ = "SUMMERS"
30  PRINT FIRST$ + LAST$
RUN
KARENSUMMERS

READY.
```



Fig. 6-1. Using strips of paper and tape to make up strings.

The two strings labeled FIRST$ and LAST$ have been linked in the PRINT command so when the program is run, the computer prints KARENSUMMERS on the TV screen. The computer does not insert any spaces between the strings. Since there are none in either string, none are printed on the TV screen. If you want to have spaces printed in the strings, you must include them in each string. In this example, there is a space after KAREN, and one after SUMMERS:

```
10  FIRST$ = "KAREN  "
20  LAST$ = "SUMMERS  "
30  PRINT FIRST$ + LAST$
RUN
KAREN SUMMERS

READY.
```

If you don't include spaces or other special characters in your strings, they can be linked to the strings later. This example shows how an asterisk can be placed between the strings FIRST$ and LAST$:

```
30  PRINT FIRST$ + "*" + LAST$
```

The result is KAREN*SUMMERS. The asterisk is a one-character string that has been linked in between the strings FIRST$ and LAST$. Other characters or strings can also be linked this way.

When numbers or values are added, their order, or position, is not important. For example, 3 + 4 gives the same result as 4 + 3. When strings are linked, their order is *very* important. Here is a program example that shows what happens when strings are combined in different ways:

```
10  FIRST$ = "KAREN  "
20  LAST$ = "SUMMERS  "
30  PRINT FIRST$ + LAST$
40  PRINT LAST$ + FIRST$
```

When this program is run, two strings are printed on the TV screen:

```
KAREN SUMMERS
SUMMERS KAREN
```

In these examples, strings have been linked in various ways and then printed, but the combined strings were not saved in the com-

puter. Sometimes it is necessary to link strings and then store the result for later use. This is done by using a new label for the new string:

```
250  NAME$ = LAST$ + FIRST$
```

Strings can be put together this way, and the following experiment gives you a chance to try it.

Experiment No. 6-1. Building a String

In this experiment, you will run a program that uses a loop and an INPUT command to get characters from the keyboard. The program links the individual characters you type into a complete string that is displayed on the TV screen.

Clear your computer for a new program and type in the following:

```
10  FOR T = 1 TO 12
20    INPUT X$
30    R$ = R$ + X$
40  NEXT T
50  PRINT R$
```

You will use this program to type in 12 *letters*, one at a time. (You must press the RETURN key after typing each letter.) After checking the program to be sure that you have entered it correctly, run it and type in 12 letters. What happens? After the 12 letters have been typed in, the complete string is displayed on the TV screen. Your TV screen should display something like this:

```
RUN
? A
? C
? Z
? F
? Q
? F
? K
? E
? S
? T
? B
? P
```

```
ACZFQFKESTBP

READY.
```

Now, type the command, PRINT R$, and press the RETURN key. What happens? The 12-letter string is printed on the TV screen again. Since the letters form a complete string that has been labeled R$, the string can be easily identified and used again and again.

The letters are displayed from left to right. Are they shown in the same order in which they were typed? Yes, the order is correct. As new characters are typed in, they are linked to the right end of the string by the R$ = R$ + X$ command, as shown in Fig. 6-2.

Run the program again, but now type in *two or three letters* after each question mark appears. Again, you must press the RETURN key to actually enter the letters into the computer. Does the computer accept and store all of the letters? Yes, they are all saved. So, you are not limited to one character for each entry. Complete words or sentences might be entered this way.

Now, change your program so that line 30 looks like this:

```
30  R$ = X$ + R$
```

How do you think this will change the operation of the program? Run the modified program, but type only one letter for each question mark. What does the display look like? When the complete 12-letter string is displayed, it shows the letters in the *reverse* order, since the newly input letter is linked to the left end of the string, as shown in Fig. 6-3. In the original program, each letter was linked to the right end of the string.

One final note about typing in strings: When a string of characters *is typed in from the keyboard,* it can be only 89 characters long.

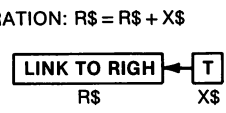


Fig. 6-2. Linking characters to the right-hand end of a string.

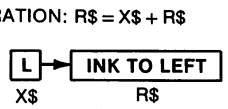


Fig. 6-3. Linking characters to the left-hand end of a string.

Experiment No. 6-2. Peculiar Characters

When you do this experiment, you'll see that not all of the characters on the keyboard are created equal, some of them do unusual things when you try and INPUT them.

In the previous experiment, you typed letters into the computer. We avoided having you type other characters, since some of them are not accepted by an INPUT command. Clear your computer and enter the following program:

```
500 INPUT H$
510 PRINT "        *"+ H$ +"*"
520 GOTO 500
```

Be sure to leave four or five spaces between the first quote in the PRINT command and the asterisk. This will place the display away from the left margin so that it is easy to see. Run the program. When a question mark appears, type in a *letter* and press the RETURN key. You should see the letter printed on the TV screen between two asterisks: *F*, for example. You can safely assume that the INPUT command will accept all of the letters, numbers, and special graphic symbols. Try the punctuation marks and symbols such as %, (, etc. Are they all printed between the asterisks?

There are four "characters" that cause problems for the VIC 20's INPUT command. These characters are the space, the quote mark, the comma, and the colon. The quote mark is simply ignored and nothing is printed on the TV screen for it. The comma and the colon are also ignored, but they cause the computer to display the message, EXTRA IGNORED. The space character is also ignored, but the previously typed character is printed again between the asterisks.

How can you enter these four peculiar characters and link them to a string if they are ignored? There is another type of keyboard input command that is much more useful for string input operations, and we will show you how the command works later in this chapter.

STRING DECISIONS

The computer can make decisions based on the information in strings. For example, a computer program can be used to "look through" a series of names to see if Chris Hendrie's name is there. The computer compares each of the names it has with the string, CHRIS HENDRIE, to see if there is a matching name on the list. At another time, the computer might be asked to print the names of all the people on a magazine's subscription list, *except for those people living in New Jersey.* Now, the computer would look at all the state names and print the information for those people whose state *did not match* the string, NEW JERSEY.

These string operations are similar to the *equal-to* and *not-equal-to* operations that were used with IF-THEN commands to compare values. In fact, these same operations can be used with strings. The IF-THEN command tests a statement to see if it is true or false. If the condition is true, the command following the THEN is done and the computer continues on in the program. If the condition is false, the computer simply goes on with the program.

When its tasks have been completed, a program might be programmed to ask you if you want to go through it again. You might have to answer by typing YES or NO, and the computer would make a decision using simple string comparisons. Here is an example in which the computer "asks" a question and expects you to answer YES or NO:

```
1040 PRINT "WANT TO DO IT AGAIN?"
1045 PRINT "  TYPE YES OR NO"
1050 INPUT A$
1060 IF A$ = "YES" THEN GOTO 100
1070 END
```

The string that you type is compared with the string "YES" by the IF-THEN command at line 1060. If the strings are the same, the computer is pointed back to line 100 in the program (GOTO 100). If the strings are not the same, the program ends.

There is a flaw in this program. The computer will go back and do the program again if you type YES, *but if you type anything else, the program will end.* The person who wrote this short program didn't bother to have the computer also check for the NO answer. Suppose you made a mistake and typed YEA instead of YES? The program would end.

A good programmer realizes that this might happen, so his program tests for a YES *or* a NO answer. If anything else is typed in, the user gets another chance to type in the correct response. Here is a better program:

```
1040  PRINT "DO YOU WANT TO TRY IT AGAIN?"
1045  PRINT "TYPE YES OR NO"
1050  INPUT A$
1060  IF A$ = "YES" THEN GOTO 100
1070  IF A$ = "NO" THEN END
1080  PRINT "JUST YES OR NO—PLEASE"
1090  GOTO 1040
```

The string typed in is checked with the string YES at line 1060, and if there is a match the computer goes to program line 100. If the string does not match YES, it is checked with NO at line 1070. If the string matches NO, the computer program stops. If the string doesn't match either YES or NO, the computer types out a short message and gives the user another chance to type in either YES or NO.

Here is a simple game in which the computer tests two strings for a match:

```
100  R$ = "MAGIC"
110  INPUT W$
120  IF R$ = W$ THEN PRINT "YOU WIN"  : END
130  PRINT "TRY AGAIN"
140  GOTO 110
```

The "hidden" word, MAGIC, is labeled R$, and the user is supposed to guess it by typing in one word at a time. The chances are slight that you could guess the word without a clue or two, but this sample program shows how the computer can compare strings. Of course, the strings aren't really "equal" in the sense that 5 = 5. However, the strings must have the exact same letters in exactly the same positions if they are to be seen as "equal" by the VIC 20.

Computer programs can also check to see if two strings are "unequal," or *not the same.* This is shown in the following example:

```
220  INPUT R$
230  IF R$ <> "BUGS" THEN END
240  etc. . .
```

When the greater-than and less-than symbols are used together, this is the same as saying "not-equal-to." In this example, if the

string typed in is not the same as BUGS, the computer is stopped by the END command. In order to continue the program, the user must type in the password BUGS. This program could have used the NOT command:

```
220 INPUT R$
230 IF NOT(R$ = "BUGS") THEN END
240 etc. . .
```

String-comparison operations are frequently used in games and learning programs such as "guess-a-word" or "hangman," in which the computer user must correctly spell a word within a certain number of tries. Most string tests only check to see whether or not two strings are the same, they don't usually check to see if one string is "greater-than" or "less-than" another.

In the examples, upper-case, or "capital" letters have been used. However, the VIC 20 can be set up to use lower-case letters, too. Since the VIC 20 "sees" "y" and "Y" as *different characters*, programmers must pay careful attention to the "case" of the letters being used. If the following command is used in a program to test for the string APRIL:

```
100 IF PX$ = "APRIL" THEN. . .
```

The string, PX$, must be APRIL and *not* April, april, or something else if the computer is to reach the THEN. . . command.

UNTANGLING STRINGS

There are three operations that let you use the computer to "cut up" strings so you can get at pieces of them. These operations let you take characters from the left, right, and middle of the string, once you have specified the string to be used and the number of characters you want. There is also a length operation that will tell you the number of characters in a string. These operations are used in many ways.

The Length of a String

The length operation, LEN, is used with a string so you can "count" the characters in it. Letters, numbers, spaces, and all special characters (except for quote marks) are included in the count. Here is an example of how the characters in a string are counted. It doesn't matter whether you count from the left or the right:

```
THIS IS A 30-CHARACTER STRING!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

The LEN command specifies a string and it gives you a value that can be printed, labeled, or used in a calculation. Here is how the LEN command might be used in a program:

```
100 PRINT LEN("BLUE RIDGE")
```

The computer would print the number 10 on the TV screen, since there are 10 characters in this string. The LEN command generally uses a string label, and here is an example that shows this:

```
500 F$ = "THIS IS A TEST STRING"
510 PRINT LEN(F$)
```

When the computer reaches line 510 in this program, it counts the number of characters in the string, F$, and prints this number on the TV screen. In this example, there are 21 characters in the string. The quote marks are not part of the string, they just show the computer where the string starts and ends.

The LEFT$ and RIGHT$ Operations

The computer uses the LEFT$ and RIGHT$ operations to get characters from each end of the string. In each case, the string's label and the number of characters wanted must be specified in the instruction. Fig. 6-4 is an example of how these instructions operate on the string.

The LEFT$ and RIGHT$ operations actually put together their own new strings, so the results of these operations can be labeled,

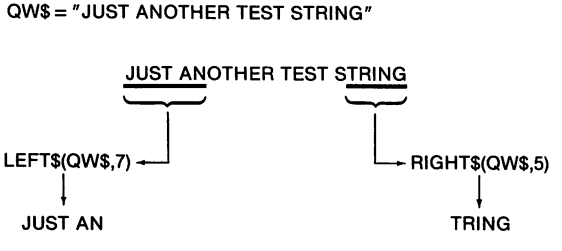


Fig. 6-4. Using the LEFT$ and RIGHT$ instructions.

as shown in the following program. This program has been listed and run:

```
350  H$ = "COMPUTER POWER"
370  W$ = LEFT$(H$,4)
390  K$ = RIGHT$(H$,3)
410  PRINT W$
430  PRINT K$
450  PRINT H$
RUN
COMP
WER
COMPUTER POWER

READY.
```

The computer printed the left-hand four characters, COMP, and the right-hand three characters, WER, on the TV screen. The original string has not been changed by the LEFT$ or RIGHT$ operations, and it is displayed, too.

Here is another example of how the LEFT$ and RIGHT$ operations work. What do you think this program will display on the TV screen? You can type this program into your computer and try it, but before you do, use some scrap paper to come up with the answer.

```
560  P$ = "TERMITE KILLING ACTION"
580  X$ = LEFT$(P$,7)
600  G$ = RIGHT$(X$,4)
620  PRINT G$
```

The program first sets up string X$ as the seven left-hand characters, or TERMITE. Next, the four right-hand characters *of this new string* are made into string G$ by the RIGHT$ command. The result is MITE.

The RIGHT$, LEFT$, and LEN operations can be combined in a word-guessing game program, as shown in the next experiment.

Experiment No. 6-3. A Word-Guessing Program

This experiment gives you a simple word-guessing program that you and a friend can use just for fun. You can see that the LEFT$, RIGHT$, and LEN operations have been used to give "clues"

about the word that is to be guessed. Clear your computer and enter the following program. Check it carefully:

```
 10  INPUT T$
 20  Z = LEN(T$)
 30  PRINT " [SHIFT] [CLR/HOME] " : PRINT
 40  PRINT Z "LETTERS"
 50  PRINT LEFT$(T$,1) "        " RIGHT$(T$,1)
 60  FOR G = 1 TO 5
 70    INPUT K$
 80    IF K$ = T$ THEN GOTO 500
 90  NEXT G
100  PRINT "SORRY, WORD IS . . ."
110  PRINT "" T$
120  END

500  PRINT "CORRECT"
510  PRINT "MATCHED IN " G "TRIES"
520  END
```

Before you run this program, let's see what it does. The first line lets you type in the word to be used in the game. The computer then determines its length, clears the screen, and prints the number of letters in the word. The PRINT command at line 50 prints the left-most and the right-most characters to give the user a clue about the word to be guessed. The steps that follow input the user's guesses and keep track of the number of tries used. Up to five tries are possible before the computer prints the correct answer for you.

Run the program and type in a simple word, such as FLAVOR. Your TV screen might look something like this:

```
  6 LETTERS
F       R
? FLOWER
? FLAMER
? FLAVOR
CORRECT
MATCHED IN 3 TRIES

READY.
■
```

Or, maybe it will look like this:

```
  6 LETTERS
F       R
? FLOWER
? FLAMER
? FRAMER
? FAVOUR
? FESTER
SORRY, WORD IS . . .
FLAVOR

READY.
■
```

It depends on whether or not the guesses lead to the correct answer. Although this may seem like a simple program, it can be developed into a more complicated program that uses colors, sounds, scores, and a large array, or table, of words that are to be matched or spelled correctly.

The LEN, RIGHT$, and LEFT$ operations "extract" valuable information about the string, so that clues can be provided for the person using the game program.

In the Middle of a String

You can get characters from the middle of a string by using the MID$ instruction. Fig. 6-5 is an example of how this instruction works. The MID$ operation identifies the string, and the first number (9) tells the computer where to start getting the characters, counting from the left-hand side of the string. The quote marks are not part of the string, so they aren't counted. The sec-

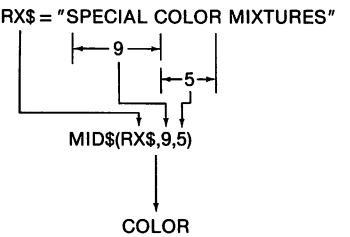


Fig. 6-5. Using the MID$ instruction.

ond number (5) tells the computer how many characters to get. Here is a short program that shows how the MID$ instruction might be used. You can load it into your computer and run it if you want to:

```
10 T$ = "VIC 20"
20 FOR F = 1 TO 6
30 PRINT MID$(T$,F,1)
40 NEXT F
50 END
```

THE GET COMMAND

In one of the experiments in this chapter, you found that there are four characters that are "ignored" by the INPUT command. The INPUT command is used mainly to input values and simple strings that do not use the space, colon, quote, or comma characters. For example, programs that require dollar amounts, weights, YES or NO answers, and so on, would use INPUT commands. However, if you want to type in information that contains commas, spaces, quotes, or colons, you will have to use the GET command.

The main difference between the INPUT and the GET commands is the way the keyboard is used. The INPUT command will input characters until you press the RETURN key. You can use a single INPUT command to get a string with many characters, or a value with many digits.

When the computer reaches a GET command in a program, it goes immediately to the keyboard and gets whatever key is pressed at exactly that moment. The computer doesn't wait for you to press a key.

If no key is pressed, the computer gets "nothing." The GET command does not wait for the RETURN key to be pressed, and it can input only a single character from the keyboard. However, it won't "ignore" any of the keys.

It may sound as though you have to know exactly when the computer reaches a GET command in a program so you can have your finger ready to type a key. It's easier than that, since in most cases the GET command is used over and over again in a loop to get individual characters from the keyboard.

The GET command is really easy to use, as you'll see in the next experiment.

Experiment No. 6-4. Using the GET Command

The GET command is very useful for getting information from the keyboard, since none of the characters are ignored. You will see how the GET command can be used in this experiment. Clear your computer and enter the following program:

```
100 GET A$
110 PRINT A$;
120 GOTO 100
```

This program simply gets the character for whatever key is pressed on the keyboard and displays it on the TV screen. The GOTO command points the computer back to the start of the program so it will do the same thing over and over again. Run the program. Watch the display and press several keys. What happens? The characters are displayed on the TV screen as they are typed in. The RETURN key has no effect except to cause the display to start at the left side of the line below it. Do the space bar, quote, comma, or colon keys have any special effect on the display? No, they act as you would expect, simply printing their respective characters on the TV screen. Of course, the space bar just moves the display over a space, as it would on a normal typewriter.

The computer is going through this three-line program very quickly. What do you think the GET command does when there are no keys pressed? Change the program by *removing the semicolon* in line 110:

```
100 GET A$
110 PRINT A$
120 GOTO 100
```

When you run this program, the display will be much different. Go ahead and start the program. Type some of the keys. What is shown on the TV screen? The characters are displayed on the left side of the TV screen, but they move up the side quickly. In fact, it is difficult to see them. Even when no keys are pressed, the display still moves up. Remember that the computer is going through this loop many times each second. If no key is pressed, the computer receives a "null" or "nothing" character from the keyboard, and the PRINT command prints "nothing." *However, since the semicolon has been removed from the PRINT command, the computer will still advance to a new line each time the PRINT com-*

mand is executed. This has the effect of "moving up" all the lines on the TV display.

By changing the program this way, you can see that the computer was still going through the GET, PRINT, and GOTO commands, even though none of the keys is pressed.

Experiment No. 6-5. Building Your Own String

In this experiment, you'll get a chance to see how the computer can "build" a string by using the GET command. Any of the keyboard characters may be used in the string. Clear your computer and type in the following program:

```
100 GET R$
140 T$ = T$ + R$
160 GOTO 100
```

Run the program and type in two or three words, using the space bar to separate them. Is anything displayed on the TV screen? How can you get your information back on the TV screen? Nothing is displayed, since there is no PRINT command in the program. Stop the program and type in PRINT T$. Are the words displayed on the TV screen now? They should be. The string was put together from the characters you typed, and it was labeled T$. Although the computer didn't know what you might do with it, it was still set aside for later use.

Let's change the program so that the information is printed on the TV screen. Add this line to your program:

```
120 PRINT R$;
```

Now, the complete program should look like this:

```
100 GET R$
120 PRINT R$;
140 T$ = T$ + R$
160 GOTO 100
```

Run this program and type in several words. You can use the space bar, colon, comma, and quote keys if you want to. Do they work the way the other keys do? Are the characters printed on the TV screen? The keys simply print their characters on the TV screen. Stop the program and again type PRINT T$. Is your complete string printed again? It should be.

It is unfortunate that you have to stop the computer to get the

entire string printed out. Let's see if we can have the computer type out the complete string when we type the "*" key. Now, the program will have to use an IF-THEN command to check for the asterisk. Here is the complete program for you:

```
100  GET R$
110  IF R$ = "*" THEN GOTO 500
120  PRINT R$;
140  T$ = T$ + R$
160  GOTO 100

500  PRINT T$
510  GOTO 100
```

Run the program and type in a short, complete sentence. You can use the INST/DEL key to back space and correct errors as you're typing. When you're finished typing your sentence, press the * key and see what happens. You should see the computer display your complete sentence on the TV screen.

Experiment No. 6-6. Getting a Few Characters

This experiment lets you further explore the use of the GET command and how it might be used in a program to get only a few characters. This experiment is optional, and you can skip over it if you want to.

Suppose that you wanted to have the computer get only five characters from the keyboard. The GET command can be used in a loop that is done five times by the computer. Clear your computer and load this program:

```
120  PRINT "TYPE 5 LETTERS"
140  FOR LC = 1 TO 5
160  GET L$
180  TS$ = TS$ + L$
200  NEXT LC
220  PRINT TS$
240  PRINT "THE END"
```

The computer will go through the loop between lines 140 and 200 five times, linking each letter to the string TS$. When five characters have been obtained from the keyboard, the computer prints THE END. Run the program *but don't type any letters.* What is shown on the TV screen? The computer displays THE END, which

means that it reached the end of your program! How can this be, since you didn't type your five letters? The computer went through the loop the required five times, getting a "null" character from the keyboard each time. Your program can't tell the difference between a useful character and a "null" character. Remember that the string operations are not affected by a null.

The program can be modified to accept only non-null characters. Add the following line to your program:

```
170 IF L$ = "" THEN GOTO 160
```

In line 170, the quote marks are right next to each other; there is *no space* between them. This represents a null character to the computer, and this statement tells the computer that if a null character is obtained from the keyboard, ignore it and go back and do the GET command again. The computer will only pass the IF-THEN command when a non-null character has been typed in.

Once you have added line 170 to your program, run it. Is the message, THE END, displayed on the TV screen? It shouldn't be. Type in five letters. What is displayed after all five have been typed? The five letters should be displayed as a complete string. This program shows how the GET command can be used with an IF-THEN command that checks for and ignores the null characters.

TELLING TIME

The VIC 20 computer has a built-in 24-hour digital clock that is easy to use. The clock has been given the string label, TIMES$, or simply, TI$. Using this label, you can "set" and "read" the time. The clock is accurate to one second and it will run for as long as the computer is powered. The clock is completely independent of your programs, so no matter what you do, the clock still keeps accurate time. This type of clock is called a *real-time clock.*

Setting the clock is very easy. You set up a 6-character string of numbers that represents the hours, minutes, and seconds with two characters each. So, to set the clock for exactly 5:30 A.M., the string would look like this:

```
TI$ = "053000"
```

Since the clock has a 24-hour "cycle," times after 12:00 noon must have 12 added to them. For example, 6:00 P.M. is 1800, 11:00 P.M. is 2300, and so on. You can set the seconds, too. For example, 9:00 A.M. *and 45 seconds* would be:

```
TI$ = "090045"
```

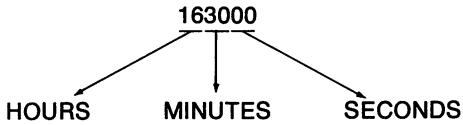
You don't have to do anything special with the string, TI$. Simply by labeling the string of digits, the clock's time is set. Reading the clock is also easy. You simply use the label TI$ in your program. For example, by asking the computer to PRINT TI$, you will have the time printed on your TV screen. If your computer is on, you can do this right now. If the clock doesn't show the right time, it's probably because it wasn't set. Here's an experiment that shows you how to use the clock.

Experiment No. 6-7. Using the VIC 20's Clock

In this experiment we will show you how to use the VIC 20's built-in clock. Clear your computer and enter the following program:

```
100 INPUT "TIME"; TI$
120 PRINT "[SHIFT] [CLR/HOME]"
130 PRINT TI$
150 PRINT "[CLR/HOME]"
160 GOTO 130
```

This program lets you type in the time to "set" the clock. After the clock has been set, the time is displayed on the TV screen. You can run the program and set the clock to any time you wish. Just remember that the format of the characters in the string is:



The characters must be typed in as a group of six, without any spaces or other characters used in the string. If you have a watch handy, you can synchronize the VIC 20's clock with the time on your watch. You'll find that the VIC 20's clock keeps good time.

Stop the program and start it again. Type in the following time:

```
235800
```

Do you know what time this is? It's 11:58 P.M., and no seconds. Watch the display. What happens when the seconds reach 59? What happens when the minutes reach 59? In about two minutes, the time changes to 000000, or midnight.

Since the time can be obtained from the computer by using a string label, can you suggest how you might change the program to include an "alarm"? You can set up an alarm by comparing the clock's time, TI$, with your own string that represents the alarm's time. Here's how you might do it:

```
100 INPUT "TIME"; TI$
110 INPUT "ALARM"; AL$
120 PRINT "[SHIFT] [CLR/HOME]"
130 PRINT TI$
140 IF AL$ = TI$ THEN PRINT "GET UP!!!"
150 PRINT " [CLR/HOME] "
160 GOTO 130
```

You can test this program by typing in a time that will start the clock and an alarm time that is only a minute or two later. For example:

```
RUN
? 040000
? 040200
```

What happens when the clock reaches the alarm time? The alarm message, GET UP!!!, is printed on the TV screen. This is not a very effective alarm unless you happen to be using the computer. In a later chapter, you'll learn how the VIC 20 can make sounds and generate a useful alarm signal.

At first, the clock may seem interesting but not very practical. However, many computer programs use a clock. For example, programs might "stamp" a time on a file so that you can tell when it was last used. Some games use a clock so that if you don't make a move or give an answer in a specific time, the computer just goes on. Some people use "appointment book" programs that keep appointments in a computer file. The clock can be used to tell them that they are scheduled for an important meeting in 30 minutes.

A clock is a useful device to have in a computer system, and there are many uses for it.

STRINGS AND VALUES, VALUES AND STRINGS

It is important to remember that strings are not values and values are not strings. You can quickly tell one from the other by the type of labels used; if the label ends with a dollar sign, it's for a string, otherwise, the information is a value.

Sometimes it is useful to change strings into values and values into strings. Of course, this only makes sense for "numbers." It doesn't make sense to try and get a numeric value for the strings "BETH SCOTT" or "SARA JANE."

Let's look at a use for these operations. The VIC 20 uses the string variable, TI$, to set and read the clock. Since you can't add strings together, how could you set an alarm for an hour from now? In the experiment you just finished, you typed in a string of numbers to set the alarm. If you didn't want to have to figure out the new alarm time and type in a new setting, you'd have to do something such as this:

1. Get the present time string, TI$.
2. Convert it to a value.
3. Add one hour to it.
4. Convert it back to a string.
5. Save it as the alarm string, AL$.

The actual program to advance the alarm can be quite complex, and it won't be discussed any further. However, the *operations* that perform these conversions are the STR$ for the value-to-string operation, and the VAL for string-to-value operation. *Remember, these operations are only used when you have strings of numbers and values that are to be converted.*

Here is an example of how the STR$ operation might be used to get the first digit of a mailing code:

```
500 INPUT "MAIL CODE"; MC
510 M$ = STR$(MC)
520 X$ = MID$(M$,2,1)
530 PRINT X$
```

The first step is to INPUT the mail code, no matter how many digits it has. The STR$(MC) operation converts the value into a string of characters, and the MID$ operation is used to get the left-hand digit.

There is a special reason why the MID$ operation has been used instead of the LEFT$ operation to get the left-most character in the mail code. When the STR$ operation is used, it automatically inserts the sign of the number at the start of the resulting string. A minus sign is included in the string for a negative number. The computer doesn't insert a plus sign for positive numbers, it leaves a blank space, instead. If a number doesn't have a minus sign, it's assumed to be positive.

So, if you typed in the *value* −11743, the *string* M$ would be "−11743" with the minus sign at the left end of the string. If you typed in the value 01609, the string M$ would be just " 01609" with a space at the left end of the string. To get the first digit without the sign, or the space for it, the MID$ operation is used. That's why the MID$ operation in the sample program specifies: go in two characters from the left and get one character.

The VAL operation performs the reverse type of operation, so if you wanted to do a math operation with a string of numbers, you could do this:

```
100  X = VAL(FM$)
```

These operations are not found in too many programs. For more information about the STR$ and VAL operations, check the *VIC 20 Programmer's Reference Guide.*

QUESTIONS

1. How can you tell if a label is used to identify a string?

2. Can a string of characters be input to the VIC 20 with a standard INPUT command? If so, how?

3. How are strings combined?

4. Can strings be combined in any order?

5. What characters are "ignored" by an INPUT command? How can they be input for use in a string?

6. What types of decisions are made for strings?

7. What operation tells you the length of a string? Are all characters counted?

8. How can you get the left three characters and the right six characters of the string, CAT$?

9. How can you get the middle five characters of the string, G$ = "QUESTIONS"?

10. If a key is not pressed when a GET command is done by the VIC 20, what is the result?

11. What kind of clock is built into the VIC 20?

12. What is a "real-time clock"?

13. Can values and strings be interchanged? Could this be a useful operation?

PROBLEMS

1. Write a program so you can enter a string and have the computer display the number of characters in the string.
2. Write a program that lets you enter a string. The computer then separates the string into two equal pieces and displays the two pieces of the string, each on a separate line. If the string contains an odd number of characters, then one piece will be one character shorter than the other.
3. Write a program that uses arrays, where you enter the names of five people along with their ages. Once this is done, enter a name and the person's age should be displayed. If you enter an age, the person's name should be displayed. A typical display would be:

   ```
   NAME OR AGE ? WENDY
   AGE IS: 19

   NAME OR AGE ? 23
   NAME IS: HENRY


   NAME OR AGE ? DAN
   NO SUCH PERSON!
   ```

4. Write a program so that once a string has been entered into the computer, the characters in the string are displayed on the TV screen, one character per line. For example:

   ```
   STRING? COMPUTER

   CHARACTERS ARE:
   C
   O
   M
   P
   U
   T
   E
   R
   ```

5. Write a game program that lets you enter a word (a string). The computer then "scrambles" the string and displays the result. To scramble the string, just randomly change the position of the characters in the word. The program should operate on

strings of any length. You then have to enter a guess as to what you thought the string (word) originally was. If your guess is wrong, the computer lets you enter another guess. If you guess correctly, you can enter another word to be scrambled.

6. Write a subroutine that causes a string to be displayed in the center of a line on the TV screen. You can assume that the string to be centered is called A$, and that it is less than 20 characters long.

7. Write a program so that once a string has been entered from the keyboard, the first character of the string is displayed on one line, the first two characters are displayed on the next line, the first three on the next, until finally the entire string is displayed on the last line. As an example:

```
STRING TO USE ? ORANGE

O
OR
ORA
ORAN
ORANG
ORANGE
```

8. Write a program that lets you enter two strings, and then the computer displays them in alphabetical order. Assume that the strings only contain letters; no numbers or special symbols.

9. If you have solved Problem 8, write a program that lets you enter five strings into an array. The computer then displays the strings in alphabetical order, one string per line.

   NOTE: In this problem, the strings do not have to be "sorted," that is, they do not have to be moved about inside the array. Instead, the program simply finds the first string and displays it, and then the next string, etc. The strings are not moved around in the array. If you want, you can eliminate the strings from the array (set them to null "" strings) once they have been displayed.

10. In Problem 11, Chapter 4, you had to write a program that converted a day between 1 and 365 to the month and day of the month. Now, solve the same programming problem using the additional statements and commands that you have learned about in the last two chapters. (HINT: Use DATA statements in your program).

11. Write a program that lets you enter a number, such as 1208976. This number is converted to a string and then the computer displays the string, with commas in it, such as 1,208,976. Don't worry about detecting a number that has been entered using scientific notation, that is, numbers such as 6.5E+30 or 4.203E−5.

12. Write a program that causes the VIC 20 to simulate a ticker-tape or electronic billboard. After entering a string (containing 20 or more characters), the first 21 characters of the string are displayed on the top line of the display. After a short period of time, for example ½ second, 21 characters are displayed again on the top line of the display, starting at the second character. This continues until the last character is displayed, followed by 20 spaces. Then the message is displayed again, starting at the beginning.

13. If you have solved Problem 12, modify the program so that instead of displaying the last character in the message and 20 spaces, the message "wraps around." This means that as you get toward the end of the message, the beginning of the message starts to appear on the right-hand side of the screen.

14. Using the clock built into the VIC 20 (TI$), write a program that lets you enter four or five times, such as 2 minutes, 30 seconds (enter as 230), 4 minutes (enter as 400), and 10 minutes (enter as 1000). After these times have been entered, the computer starts to keep track of the time. After 2 minutes and 30 seconds have elapsed, the computer displays the message, "TIME 1 HAS ELAPSED." After the second time period has elapsed, the computer displays the message, "TIME 2 HAS ELAPSED." This process should continue until all of the times that were entered on the keyboard have been "timed." This type of program would be useful in photography, where a number of the film development steps have to be timed, one right after the other.

15. Write a program that asks you the time, and then asks how many hours and minutes to wait before displaying the message, "ALARM," on the TV screen. Thus, if you tell the computer that it is 103015, and that you want the alarm to go off in 2 hours (020000), the computer will display the ALARM message at 123015.

7

Special Math Functions

There are some special math functions that you will come across less frequently than addition, subtraction, multiplication, and division. Since these new functions are not seen often in computer programs, we have put them in their own chapter. We will also tell you about the types of "codes" that computers use to transfer information. There are no experiments in this chapter, so you won't need to use the computer as you read it.

Most computer programming books that discuss BASIC-language programs simply list the special math functions without telling you much about them. They leave it to you to decide whether or not you need to use them, providing a few simple examples of how these instructions work and what they do. There is a great deal of interesting history behind these math operations, and we thought you would be more interested in this than in learning 20 ways of using the cosine function.

THE SQUARE ROOT

Most people who have attended high school have heard of a "square root" even if they haven't actually used this interesting function. Let's go back to the sixth and fifth centuries, B.C. and look at the work of Pythagoras and his colleagues, the Pythago-

reans, who lived in what is now southern Italy. The Pythagoreans were interested in the abstract side of mathematics and did not use their mathematical skills in any commercial ventures. One of their beliefs was that there was a common measure between any two lengths. Thus, no matter what their lengths, there was a common measure between two pieces of rope, as shown in Fig. 7-1. In this example, one piece of rope has two "units" while the other has three "units." Of course, other lengths would probably require different common "units."

Ancient Egyptian land surveyors used a rope triangle with sides of 3, 4, and 5 units during their surveying. Such a triangle always contained a "right angle," or an angle of 90°, which made it an ideal tool for laying out square or rectangular plots of land. No matter what size triangle was chosen, if the ratio of the sides was 3 to 4 to 5, a right angle was always formed. You can measure a piece of string and prove this for yourself.

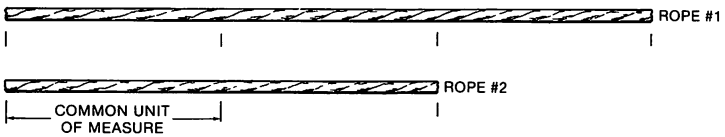


Fig. 7-1. There is a common unit of measure between the two pieces of rope.

Using this type of special triangle as an example, the Pythagoreans discovered an interesting thing about right triangles—those in which one angle is 90°. If a square was constructed for each side of the triangle, as shown in Fig. 7-2, the area of the large square (25) on the long side was found to be equal to the sum of the areas of the two smaller squares (9 + 16). It is thought that the Pythagoreans also knew this was the case for a right triangle in which the two short sides were of equal length, as shown in Fig. 7-3. This "sum-of-the-squares" relationship can be proven for other right triangles, too, but in their time the Pythagoreans probably weren't able to do so. Today, this relationship between the "squares" on the sides of a right triangle is called the *Pythagorean Theorem.*

The Pythagoreans looked at the type of triangle shown in Fig. 7-3 and found that there was no common measure between the two short equal sides and the longer side. No matter how small a "measure" was used to divide the sides, they could not find one that was common to the short sides and the long side. Since the

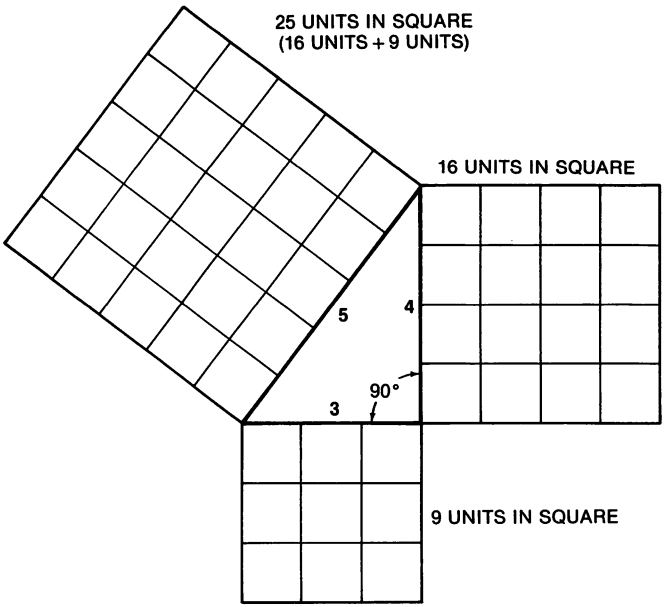


Fig. 7-2. A 3-4-5 right triangle shows the sum-of-the-squares rule.

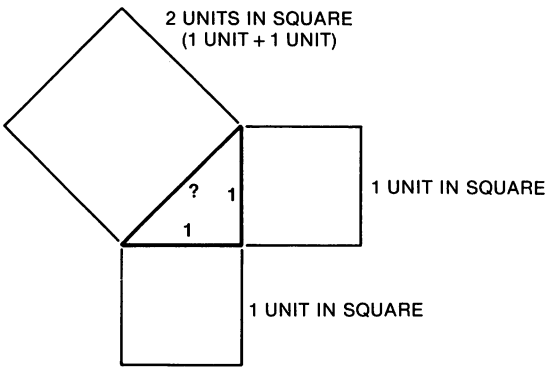


Fig. 7-3. Another right triangle that shows the sum-of-the-squares rule, but without a common measure between the short and long sides.

common-measure principle was central to their mathematical thinking, this posed a perplexing problem.

Obviously, the area of the large square is 2 square units, so the length of the long side must be a number that when multiplied by

itself yields 2. We know that this number is approximately 1.41, but the Pythagoreans did not have decimal numbers and could not find a number that would exactly represent this length. Since there was no common "measure" or *ratio* between the lengths of the short and long sides, the length of the long side was said to be "not a rational" number, or *irrational.*

As mathematics evolved, the side of a square was called the "root" of the square, or the number from which the square "grew." Now we simply say "square root," and calculators and computers can easily find the square root of a number. In the case of the value 2, the square root is close to 1.414. We say close, because the square root cannot be calculated to a final value. A pocket calculator gives the value as 1.4142136 . . ., and more decimal places may be calculated.

The VIC 20 calculates square roots very easily using the SQR operation, and here is what it looks like:

```
ROOT = SQR(number)     or     DX = SQR(2.00)
```

The SQR function expects the number you want the root of to be greater-than or equal-to zero.

Here is how you can use the square root operation to find the long side of a right triangle if you are given the lengths of the two short sides:

```
500  REM COURTESY OF PYTHAGORAS
520  PRINT "ENTER TWO SHORT SIDES"
540  INPUT S1 : INPUT S2
560  LG = SQR(S1 * S1 + S2 * S2)
580  PRINT "LONG SIDE IS . . ."
600  PRINT LG
```

There are many uses for the square root function in engineering, science, mathematics, statistics, surveying, astronomy, and other fields that evaluate numeric information. It is one of the more useful functions that we'll describe in this chapter.

THE VALUE OF PI

One of the problems facing ancient mathematicians was the exact measurement of the circumference, or the "length" around the outside of a circle. While the radius (r) and diameter (d) for the circle shown in Fig. 7-4 can be easily measured, there is no accurate way to measure the length of the circumference (c). You can

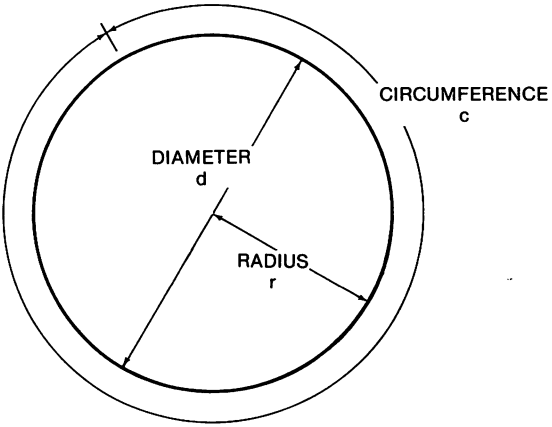


Fig. 7-4. The radius and diameter of a circle are easy to measure; the circumference isn't.

use a compass to mark out a circle and you can try to measure the circumference with a ruler. You'll find that it's difficult to measure a curved line with a straight-edged ruler, and the early mathematicians had the same problem. In their attempts to measure the circumference, they found that it was approximately three times the length of the diameter. Ancient Hebrew and Babylonian mathematicians fixed the ratio of the diameter to the circumference at exactly 1 to 3, or just 3.

The ancient Egyptians knew better, however, using the ratio of 1 to 3.1605 as early as 2200 B.C. Of course, they didn't have decimal numbers, so the ratio was written in a different form. The main point is that the Egyptians were able to come up with a more accurate ratio, realizing that the circumference was not simply three times the length of the diameter of a circle.

The Greek mathematician, Archimedes, who lived between 287 and 212 B.C., improved upon this ratio by using a process of approximations that is shown in Fig. 7-5. Using this method, a circle has a square placed inside it so that each *corner* touches the circle. Another square is placed outside the circle so that each *side* touches the circle. It is easy to measure the perimeter of both squares, and the circumference lies somewhere between these two values. Archimedes increased the number of sides on each square, making them into octagons, and he added more and more sides as he went on. In each case, the inside polygon had all its sides equal,

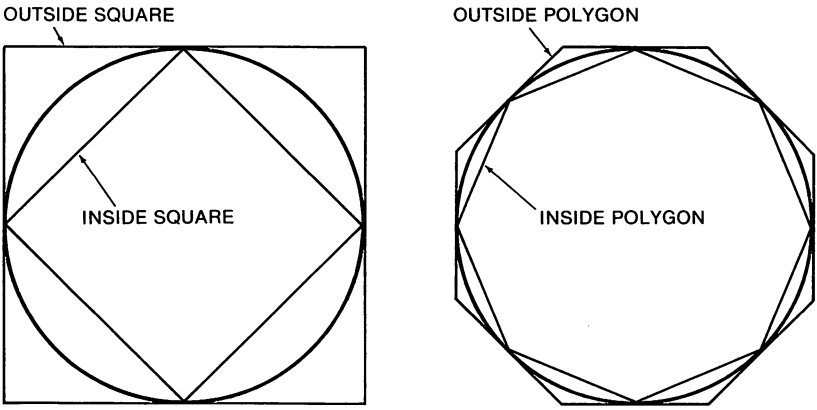


Fig. 7-5. Using approximations to measure the circumference of a circle.

and each corner touched the circle. The outside polygon had its sides of equal length, too, and each side touched the circle. As the number of sides of each of the polygons was increased, their perimeters more closely matched the circumference of the circle. When Archimedes had gotten the polygons so they had as many sides as possible, he was able to put the diameter-to-circumference ratio between 71 to 223 and 70 to 220. More simply, the circumference was between 3.14085 and 3.14286 times the length of the diameter.

A Hindu mathematician, Aryabhata, who lived in the sixth century A.D., was able to further refine the value to 3.1416, which is extremely close to the present value of 3.14159. This is a constant, and it is used by engineers, scientists, mathematicians, students, and almost anyone who has taken even an introductory course in geometry. The constant is called *pi*, for the Greek letter π, which is used to stand for the value 3.14159. You can simply think of it as a special label for this value.

Most people can tell you the simple relationships:

$$\text{circumference} = \pi * \text{diameter of the circle}$$

And, since the diameter is twice the length of the radius:

$$\text{circumference} = \pi * 2 * \text{radius of the circle}$$

The value of pi has been preset in your VIC 20 computer, and you will see the Greek pi symbol, π, on the up-arrow key on the right side of the keyboard. This is a special label used just for this value in the VIC 20 computer. You can type:

```
PRINT π
```

and you will see the value 3.14159265 displayed on the TV screen. You can use this value in a program by using the label, π, in your math formulas. Here is a sample program that calculates the circumference of a circle when you type in its radius:

```
250 INPUT "RADIUS ="; RD
260 PRINT "CIRCUMFERENCE IS = " 2 * π * RD
270 GOTO 250
```

Now, with an accurate value for the ratio between a circle's diameter and circumference, it was possible for mathematicians to look more closely at circles and some of their other measurements. We now move on to trigonometry.

TRIGONOMETRY

Although this subject often seems to strike fear into the hearts of students, trigonometry is nothing more than "tri-angle measurement," or the measurement of triangles. Aristarchus, another Greek mathematician, first used what we would consider to be "tri-angle" measurements when he attempted to measure the distance between the earth and the sun, as shown in the distorted diagram in Fig. 7-6. His methods used complex geometric reasoning, but his result was incorrect due to the inaccuracy of his measurements. However, his *line of reasoning*, using triangles and the lengths of their sides, was correct and it pointed the way to more accurate measurements and further investigations of triangles.

The next mathematician we encounter is Hipparchus, who is thought of as the "inventor of trigonometry." Modern trigonometry arises from his measurement of chords in circles. As shown in Fig. 7-7, a chord is nothing more than a straight line that cuts a circle into two pieces. The resulting curved line is called an arc, and Hipparchus realized that there was a definite relationship between the length of the chord and the length of the arc. He wrote several "volumes" about measuring the "arc in a circle."

If lines are drawn between each of the two points where the chord line cuts through the circle and the center of the circle (Fig. 7-8), you'll see that an angle is formed. Each circle was considered to have 360°, so these angles could be measured in degrees. However, chords were measured by their length, and this was based not on a standard of inches or centimeters, but on the length of the radius of the circle. This type of measurement could lead to prob-

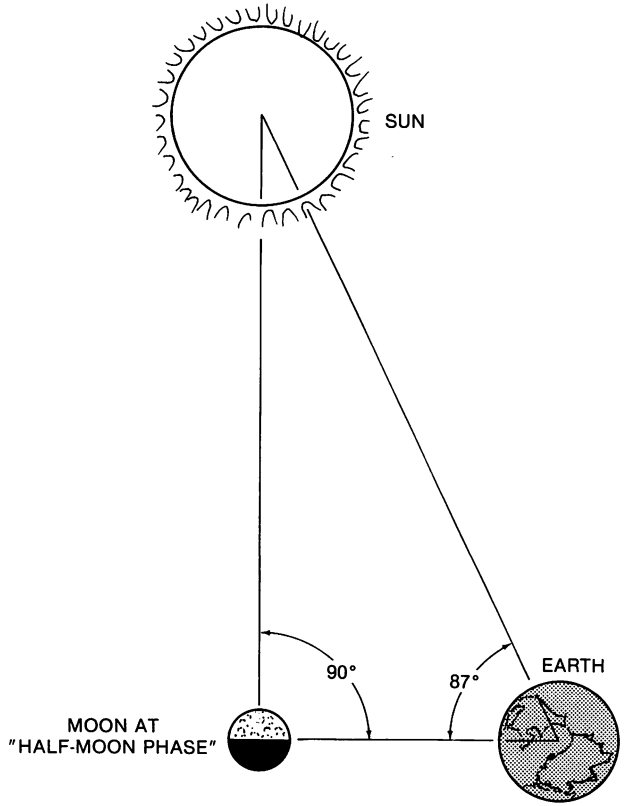


Fig. 7-6. Using angles to measure the distance from the earth to the sun.

lems, since the measurement of a chord depended on the units used to measure the radius of the circle.

This brings us back to the Hindu mathematician, Aryabhata. His contribution to trigonometry was to divide each chord in half, as shown in Fig. 7-9, and he also introduced a *right angle*, or 90° angle into the measurement. As trigonometry developed, it was realized that the *ratio* between the length of the radius and the length of the half-chord was more useful than the lengths by themselves. For example, consider the two triangles (or half-chords) shown in Fig. 7-10. The circles have been left out for clarity. The angles are exactly the same, but the lengths of the "radius" and the "half-chord" are different. However, when the lengths are measured and the *ratios* are calculated, they turn out to be exactly

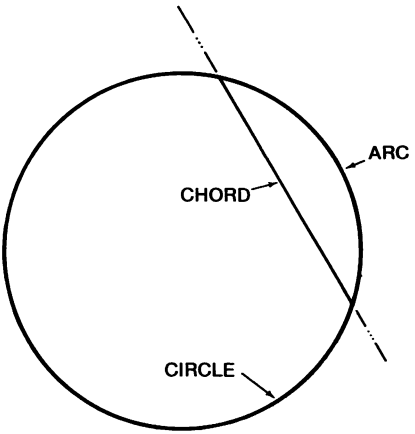


Fig. 7-7. An arc and a chord in a typical circle.

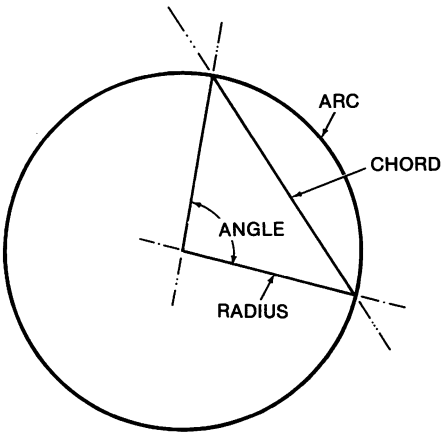


Fig. 7-8. An angle is formed between the center of a circle and the points where the chord crosses it.

the same. With an inexpensive ruler, we found that each "half-chord" was just about 0.46 times the length of the "radius." This is an important relationship, since no matter what size the triangle is, if the angles are the same, the ratios are the same.

The word to describe this relationship was translated and mistranslated several times, so we finally have the Latin word, *sinus*, which means "curve" or "bosom." This has been abbreviated to

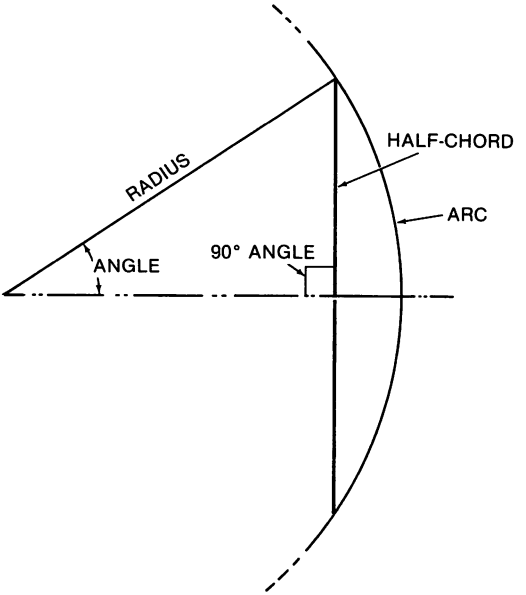
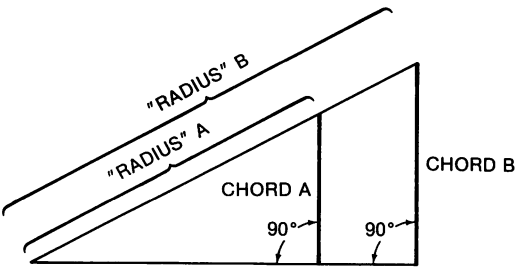


Fig. 7-9. The angle in a circle-chord can be simplified by looking at only half of it.



$$\frac{\text{CHORD A}}{\text{RADIUS A}} \cong \frac{\text{CHORD B}}{\text{RADIUS B}}$$

$$\frac{40}{86} = 0.465 \qquad \frac{54}{115} = 0.469$$

$$0.465 \cong 0.469$$

Fig. 7-10. Two triangles that have the same angles have equal ratios between the sides.

sine (which is pronounced "sign"), and is usually found in mathematics books as simply *sin*. The words "radius" and "half-chord" are no longer used in this relationship. Instead, we refer to the long, angular line as the *hypotenuse*, and the other lines are said to be *adjacent-to*, or *opposite-from*, the angle we're interested in. The *sine* of angle A is just the ratio of the length of the *opposite* side to the *hypotenuse*. Of course, this is only true for a right-triangle.

Without going into the history behind them, we'll simply tell you that there are two other ratios of interest. One is called the *cosine*, or *cos*, and the other is called the *tangent*, or *tan*, of the angle. These are also ratios of the lengths of the sides of the triangle. The three relationships are illustrated in Fig. 7-11.

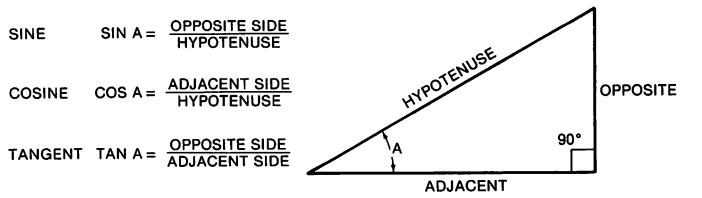


Fig. 7-11. The sine, cosine, and tangent ratios between the sides of a right triangle.

VIC 20 TRIGONOMETRY

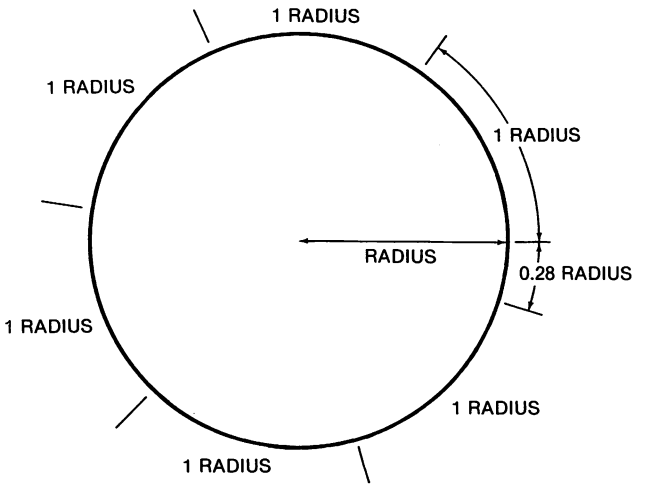
Your VIC 20 computer can give you the *sin*, *cos*, and *tan* for any angle you give it, so it is possible to use these ratios in many interesting ways. They are particularly useful in science, engineering, and mathematics. We'll look at an example in just a moment. The formats for these BASIC-language operations are fairly simple:

```
A = SIN(QX)        D = COS(FK)        R = TAN(YW)
```

However, there is one problem; the VIC 20 does not accept angles measured in degrees. A circle contains 360°, and this is convenient for many measurements. However, the VIC 20 trigonometric, or *trig* functions require that the angle be measured in a unit called the *radian*. A radian is nothing more complicated than a "radius." Since the circumference of a circle is equal to:

$$\text{circumference} = 2 * \pi * \text{radius}$$

there are 2 π radiuses, or radians, around the circle as shown in Fig. 7-12. No matter what the actual radius is, this will always be true. Since there are also 360° in a circle, it is not difficult to con-



CIRCLE = 2π RADIUSES = 6.28 RADIUSES = 6.28 RADIANS

Fig. 7-12. The relationship between the radius of a circle and the number of "radians" around its circumference.

vert from radians to degrees, or *vice versa*. It may take you a while to get used to the term *radian*. Just remember that there are about 57° per radian. You can convert from degrees to radians with this formula:

$$\text{radians} = \frac{(\text{angle in degrees}) * 2 * \pi}{360}$$

or from radians to degrees with this formula:

$$\text{degrees} = \frac{(\text{radians}) * 360}{2 * \pi}$$

In BASIC, these are:

```
RAD = DEG * 2 * π / 360
```

and:

```
DEG = RAD * 360 / (2 * π)
```

USING TRIG FUNCTIONS

Here are two examples that show you how trigonometry can be used. Not all of the mathematics or algebra is shown, since that is

really secondary. We want you to have an appreciation for the types of problems that can be solved with the trig functions. The BASIC programs are provided for you.

In the first example, several young people are testing model rockets and they want to know how high the rockets are going. The rockets are fired straight up and there is no wind to blow them off course. Two people use a home-built device at an observing point to measure the angle between the rocket and the level ground. This is shown in Fig. 7-13. Once the rocket has been fired, the students track it until it reaches its maximum height. At this point, they note the angle between the earth and the rocket. Since the distance from the launch site and the students doing the measurements is 650 meters, can we calculate the altitude of the rockets?

The diagram of the launch site can be simplified to a right triangle. The angle, A, has been measured and the side of the triangle marked "distance" is 650 meters. The tangent function can help us out, since:

$$\tan(\text{angle A}) = \frac{\text{opposite side}}{\text{adjacent side}} = \frac{\text{rocket eight}}{\text{distance from launcher}}$$

And, by rearranging the equation, you get:

rocket height = distance from launcher * tan (angle A)

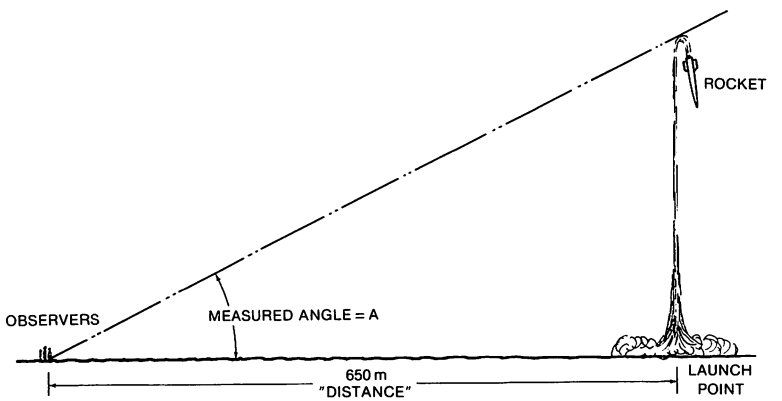


Fig. 7-13. Measuring the angle between the ground and the rocket to get its height.

The angle must be converted into radians, too. Here is a computer program that will solve the problem:

```
200  PRINT "ROCKET ALTITUDE"
220  PRINT " CALCULATOR"
240  INPUT "DIST TO LAUNCHER"; DTL
260  INPUT "MAXIMUM ANGLE"; MA
280  RAD = MA * 2 * π / 360
300  ALT = DTL * TAN(RAD)
320  PRINT "ALTITUDE" ALT
```

In the second example, we'll extend this application a bit by supposing that a light wind is changing the rocket's flight path so that it doesn't always go straight up. Is it possible to measure its maximum altitude? Since the rocket is no longer going straight up, it cannot be done with only one observation. Two observers are needed, as shown in Fig. 7-14. The only measurements needed are:

1. The base-line distance between observers (BL).
2. The angle between the ground and the rocket for each observer (GA and GB).
3. The angle between the base line and each observation (HA and HB).

Since it is important that the observers record their angles at exactly the same time, a set of walkie-talkie radios is used to coordinate the measurements.

There are two ways to solve this problem; (1) make a scale drawing using the angles and the base-line distance and "measure" the rocket's altitude, and (2) calculate the altitude using trigonometry. We'll take the latter course. Here's the BASIC program that will solve the problem:

```
100  PRINT "ROCKET ALTITUDE"
120  PRINT "CALCULATIONS FOR"
140  PRINT "TWO OBSERVERS"
160  PRINT "BASE LINE LENGTH?"
180  INPUT BL
200  PRINT "OBSERVER A"
220  INPUT "GROUND ANGLE"; GA
240  INPUT "HEIGHT ANGLE"; HA
260  PRINT "OBSERVER B"
280  INPUT "GROUND ANGLE"; GB
300  INPUT "HEIGHT ANGLE"; HB
```

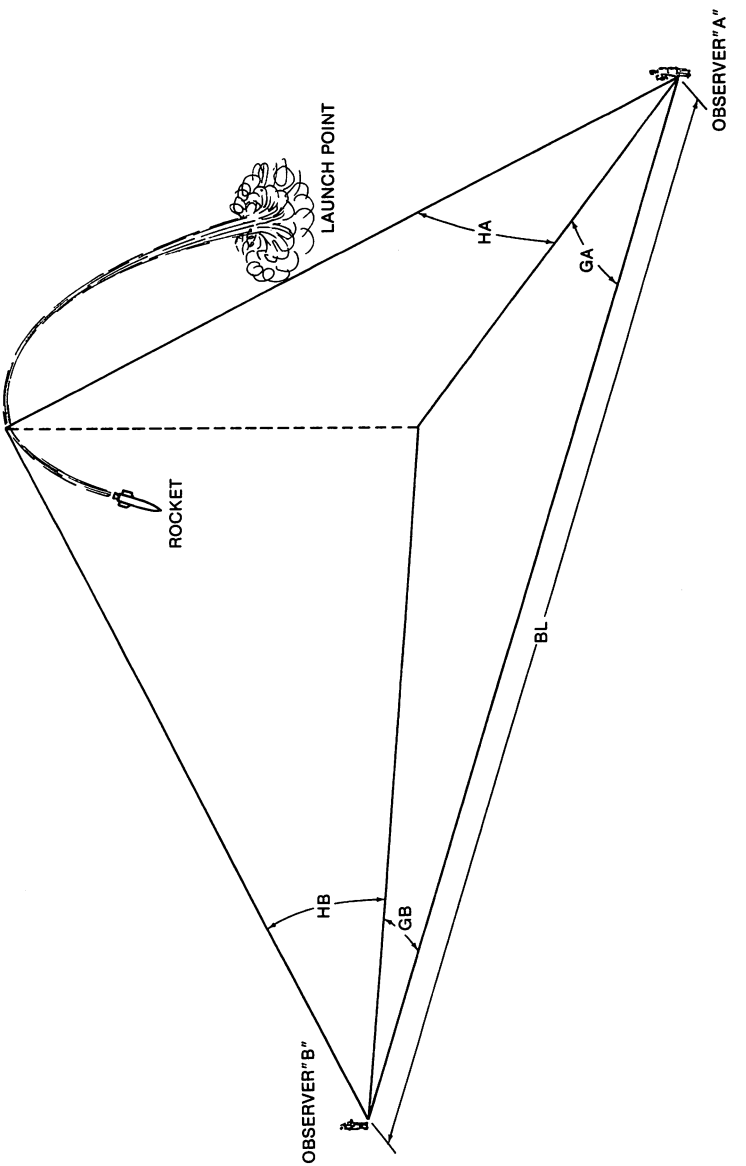


Fig. 7-14. Using two observers to measure the height of a curved flight for a rocket.

```
320 REM CONVERT TO RADIANS
340 C = 2 * π / 360
360 GA = GA * C : HA = HA * C
380 GB = GB * C : HB = HB * C
400 BX = BL/((TAN(GB)/TAN(GA)) + 1)
420 AX = BL - BX
460 PA = AX/COS(GA)
480 PB = BX/COS(GB)
500 REM ALTITUDE CALCULATIONS
520 A1 = PA * TAN(HA)
540 A2 = PB * TAN(HB)
560 AVG = (A1 + A2) / 2
580 PRINT "OBSERVER A ALT"
600 PRINT A1
620 PRINT "OBSERVER B ALT"
640 PRINT A2
660 PRINT "AVERAGE ALTITUDE"
680 PRINT AVG
```

If you decide to run this program, here are some values you can use:

Base line between observers:	3000 meters
Observer A ground angle:	45°
Observer A height angle:	37°
Observer B ground angle:	26.5°
Observer B height angle:	26°

You'll find that the altitudes calculated for observers A and B are 1066 and 1090, respectively. The average is 1078. Remember, these values are based on experimental measurements, and there are always errors in them.

The important point of these examples is to give you an appreciation for the use of trigonometry and for the tremendous work done by early mathematicians as they developed these relationships and came up with ratios used in the sine, cosine, and tangent operations.

There is one last trigonometry operation in the VIC 20, and it is called the arc-tangent, or arctangent. If you have the tangent of an angle, the arctangent operation will give you the angle, in radians. The form for this operation is:

ANGLE = ATN(tangent of the angle)

There are many other trig operations that can be done, based on the SIN, COS, TAN, and ATN functions of the VIC 20, and they are listed in the *VIC 20 Programmer's Reference Guide.*

LOGARITHMS

Early mathematicians spent years computing and compiling the values of the sine, cosine, and tangent for angles. Since these values were still thought of as "lengths of a chord," very small units were used to measure the radius of the circle so that fractional lengths were avoided as often as possible. The calculations of these "lengths" required tedious multiplications and divisions in the interest of greater accuracy, and they were not always error-free.

A better method was needed for manipulating large numbers, and the Scottish mathematician, John Napier, contributed two solutions. His first was the development of decimal fractions as we know them, something we probably take for granted. Earlier mathematicians often multiplied small numbers by a million so they wouldn't have to work with fractions. At least one table of square roots was published in which this was done. The square root of 3000000 is 1732, which doesn't require any fractions, while the square root of 3 is 1.732. The value 1.732 would have been 1 and 183/250ths to the mathematicians in the sixteenth century. Although the roots in the table were 1000 times too great, this was easy to remember and fractions were dispensed with.

It did not occur to many mathematicians to extend their digits to the right as we do today, using a decimal point and decimal fractions. Napier developed the idea of using a "period" or decimal point and extending the fractions as their decimal equivalents in tenths, hundredths, thousandths, and so on. This meant that fractions could be easily represented and worked with using a single line of decimal numbers.

Napier's second development was the *logarithm,* which he described in a book published in 1614. It had taken him 20 years to develop and refine this idea and to provide useful tables of *logarithms* that others could use to simplify calculations. This had a profound influence on the mathematics of Napier's time, and logarithms are widely used today.

Without going into more of the history of logarithms, or "logs" as they are commonly called, we'll just say that Napier developed a special way of treating numbers. Basically, all numbers can be rep-

resented as a *common,* or *base value* with an appropriate exponent. *Exponents* or superscripts are mathematical shorthand notes that are placed above and to the right of numbers or labels to indicate how many times they are multiplied by themselves. Since modern logarithms use a *base* of 10, here is an example:

$$10 * 10 * 10 = 10^3 = 1000$$

Thus, the base-10 logarithm of 1000 is 3. There are tables of logarithms that provide the exponent, or "log," for any value, and most scientific calculators and small computers can give you the log of a number.

Let's see how logs can be used. In the following example, a value, X, is multiplied by itself five times:

$$X * X * X * X * X = \text{answer}$$

Exponent's can be used to simplify the way the problem is written:

$$X*X*X*X*X \text{ or } X^2 * X^3 \text{ or } X^5$$

There is something very interesting about the way the problem has been rewritten. Instead of doing the actual multiplications, the exponents have been *added.* Napier used this approach in his system of logarithms. By using logarithms, multiplication of numbers became the addition of their exponents, and division of numbers became the subtraction of their exponents. Other complex functions can be simplified by using logarithms, too.

Using logarithms, you can easily multiply 3.98712 by 2.12678 without doing any multiplication at all. You can find the log for each of these values in a table of logarithms and, since each log represents an exponent, simply add them. The result would be an exponent which is the *log of the answer.* Here is what the problem looks like:

$$\begin{array}{rll} 3.98712 & \log(3.98712) = & 0.6006584 \\ \times 2.12678 & \log(2.12678) = & 0.3277226 \\ \hline ??? & & \end{array}$$

$$\log(3.98712) + \log(2.12678) = \log(\text{answer}) = 0.928381$$

Now, you would look in the table in the logs column until you found 0.928381. On the same line would be the value 8.47971, which is the result of the original multiplication. Although this has been simplified a bit, it still shows you how powerful logs were in easing the job of complex calculations.

Napier's original table of logarithms is no longer used. It was quickly improved upon by other mathematicians, and the base-10, or common, logarithms are the ones in general use today. However, 10 is not the only base possible for logarithms.

There is another type of logarithm called the *Napierian* or *natural* system of logarithms, and it is based on the value 2.7182818. The "natural" base is simply called "e" by people who use this form of logarithm. Although called Napierian logarithms, these are not the original logarithms Napier developed.

As long as you are just using the logarithms to simplify multiplication, division, and other complex math operations, the base is unimportant. However, in some calculations it is important to distinguish between the two systems of logs. In the common log system, logs are noted as log(x), while in the natural log system, they are noted as ln(x).

VIC 20 LOGS

The VIC 20 uses natural logs, so it is unfortunate that the LOG notation has been chosen for the logarithm command in BASIC. To have the computer give you the natural log of a value, you use the command:

```
QX = LOG(value) (The value must be greater than zero.)
```

You are asking the computer, "What exponent do you use with 'e' to get 'value'?"

Once you have obtained the logarithm of a number from the computer, how can you get back to the original number? The VIC 20 uses the EXP operation to do this. Here is an example:

```
DZ = EXP(value)
```

You are asking the computer, "What is the result of raising 'e' to the 'value' power?" In this case, the value must be less than 88. The 88th power of *e* is a very big number!

If there is a need to, you can convert between the natural and common logs. Here are these important relationships:

$$\ln(x) = 2.3026 * \log(x) \text{ or } 0.4343 * \ln(x) = \log(x)$$

and

$$e^x = 10^{(0.4343 * x)} \text{ or } e^{(2.3026 * x)} = 10^x$$

Logs are often used when people are comparing power levels, and the following relationship is used:

power ratio = 20 * log (power out / power in)

For example, if a hi-fi amplifier can produce 150 watts of power from a tape recorder that gives it only 0.1 watt, the relationship is:

power ratio = 20 * log (150 / 0.1)
= 20 * log (1500)
= 63.52

The power ratio is called the "decibel," or dB, named after Alexander Graham Bell, the inventor of the telephone. Bell worked with deaf people, and the telephone was an attempt to develop a device to increase their ability to hear. Decibel simply means tenth-of-a-bel, since the larger unit of bel is not frequently used. If you buy a hi-fi amplifier and the salesman tells you that the amplifier can deliver a 50-decibel power gain, you'll have a good idea of what he is talking about.

Since the VIC 20 uses natural logs, the power relationship would be:

power ratio = dB = 20 * 0.4343 = ln(power out / power in)

or, in a BASIC computer program:

```
DB = 20 * 0.4343 * LOG(PO / PI)
```

There are many other uses for logarithms in science, mathematics, and engineering.

SPECIAL CODES

Hundreds of years ago, if "information" was to be sent quickly over long distances, flags, fires, or other signaling devices had to be used. When the electric telegraph was developed by Samuel F. B. Morse, the "Morse" code of dots and dashes was used. In this code, dots and dashes were used to represent each letter, number, or other special symbol to be communicated from one telegraph station to the next. Morse code is still used today for radio communications.

As communication techniques advanced, the teletypewriter was developed. This is an electromechanical device that automatically transmits information from a keyboard to a device that prints actual letters and numbers on a strip of paper. Instead of using dots

and dashes, a special code was developed by Emile Baudot, and the "Baudot" code was used to transfer information from one teletypewriter to another over telegraph wires.

As more and more computers came into existence in the 1950s and 1960s, it was obvious that a more efficient and "universal" code had to be adopted so that information could be transferred from one computer to another. One such code is called the American Standard Code for Information Interchange. This is abbreviated ASCII, and most people who are familiar with it call it the "as-key" code.

All of the letters, numbers, punctuation marks, and special symbols such as $, have a unique ASCII "code." By using this code, computer designers and programmers can be sure that the code for the letter "A" will be the same from one computer to another. Without this standard, it would be difficult to transfer information from one computer to another.

Your VIC 20 computer can use the ASCII representation for information. There are two special operations that may be used in BASIC programs to convert from string characters to their equivalent code and back again. Although these operations are not frequently used, you may find them in a program or two, particularly if communications between several computers are involved.

These operations are the ASC and CHR$ operations. THE ASC operation lets you "convert" a string character into its equivalent coded representation. For example, PRINT ASC("H") would print the value 72 on the TV screen, since 72 is the ASCII equivalent for the letter "H". The CHR$ operation "converts" from ASCII to a string character. For example, the PRINT CHR$(87) would print the letter "W" on the TV screen, since W is the character with the ASCII value of 87.

Here is an example of how the ASC operations can be used:

```
50 INPUT A$
60 PRINT ASC(A$)
70 GOTO 50
```

When this program is run, the value of the code for each character will be displayed on the TV screen as its decimal value. For example, the code for the letter "X" is 88. This type of character-to-ASCII conversion process is used when the VIC 20 is used as a terminal to "talk" with another computer over a telephone line.

The CHR$ command does just the opposite, taking a code and converting it to its equivalent character. Here is an example:

```
100 INPUT CODE
120 PRINT CHR$(CODE)
140 GOTO 100
```

If the value 50 is typed in, the string character "2" is printed on the TV screen. The values for the CHR$ instruction can be between 0 and 255. However, in the VIC 20, *only the values of 13, and those between 32 and 95 correspond to the standard ASCII code.* The other codes have special functions within the VIC 20. Some display the special graphic symbols on the TV screen and do other things. However, the values for the letters, numbers, punctuation marks, and special symbols are the same.

THE TAN "f" KEYS

The four tan keys marked f1 through f4, and located on the right side of the keyboard, have been ignored so far. These keys are not used when you are programming the computer, since they don't print a special symbol or have any special meaning to the VIC 20. However, these keys are useful, and you may see a game or other program that tells you to press one of them to cause something special to happen.

Each of the keys has its own special ASCII value, as listed in Table 7-1. You can get at the even-numbered "f" keys by pressing [SHIFT] and one of these keys. The keys can be used in a program by using a sequence of instructions such as this:

```
200 INPUT FK$
220 F = ASC$(FK$)
240 IF F = 133 THEN GOTO. . .
260 IF F = 137 THEN PRINT. . .
```

Table 7-1. ASCII Values for the Tan "f" Keys

Key	Value
f1	133
f2	137
f3	134
f4	138
f5	135
f6	139
f7	136
f8	140

This simple sequence checks for the f1 and f2 key operations, and takes whatever action the programmer has put in the program. The program would probably trap errors, but we haven't shown that. The ON command could be used, too, by subtracting 132 from the code for the "f" keys. This gives you a 1 for the f1 key, a 2 for the f3 key, and so on. Now the following program lines could be used:

```
500 GET F$
520 IF F$ = " " THEN 500
540 F = ASC$(F$) - 132
560 ON F GOTO 3500, 4590, etc. . .
```

This can simplify things a bit if sequential "f" keys are being used. The main purpose of the "f" keys is to give you some extra keys that can serve special purposes in a complex program.

For additional information about the ASCII characters and their codes, we suggest that you look at the appendixes in your VIC 20 computer guide, supplied with the computer, or in the *VIC 20 Programmer's Reference Guide,* where this information is listed for you.

REFERENCES

1. Hooper, Alfred. *Makers of Mathematics.* New York: Random House, Inc., 1948.
2. *VIC 20 Programmer's Reference Guide.* Indianapolis: Howard W. Sams and Co., Inc., 1982.

PROBLEMS

1. You are standing on top of a building that is 650 feet high (Fig. 7-15). You look down at the roof of another building, 230 feet away, at a 20° angle. How tall is this building?
2. When measuring the acidity of a liquid, the acidity is usually expressed in units of pH. Water, a neutral liquid, has a pH of 7. A mildly acidic substance, such as vinegar, might have a pH of 5 or 6. A strong base, such as some bathroom cleaners, might have a pH of 13 or 14. The pH of a substance is defined as:

$$pH = -\text{Log}(H^+)$$

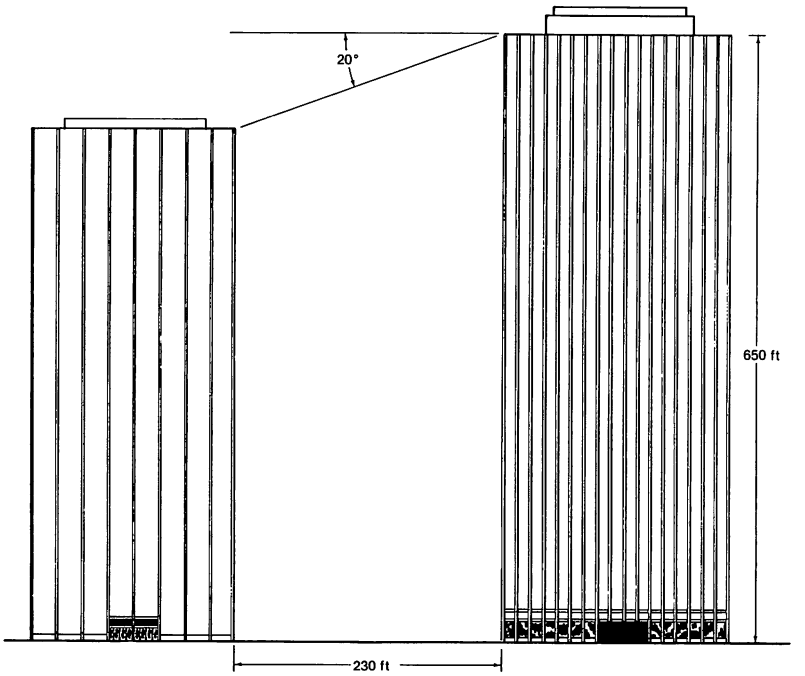


Fig. 7-15. Determining the height of a building.

Thus, if the hydrogen ion (H^+) concentration is 0.0000001 molar, or 1.0×10^{-7}, the pH will be 7. Write a general-purpose program that lets you enter a hydrogen ion concentration, such as 2.89E−6, and displays the pH (5.55 for 2.89E−6). (Hint: Remember, the computer LOG function is really the natural log (ln) function, so you will have to convert from LOG to ln!)

3. Write a program that displays the sine of the angles, 10, 20, 30, . . . , 350° on the TV screen. The program should display both the angle and the sine of the angle.

4. Modify the program from Problem 3 so that the sine function is "plotted" on the TV screen. The sine for 0° should be at the top of the screen, and the sine for 360° should be at the bottom of the screen. The TV screen should appear as shown in Fig. 7-16. (Hint: Use the TAB function.)

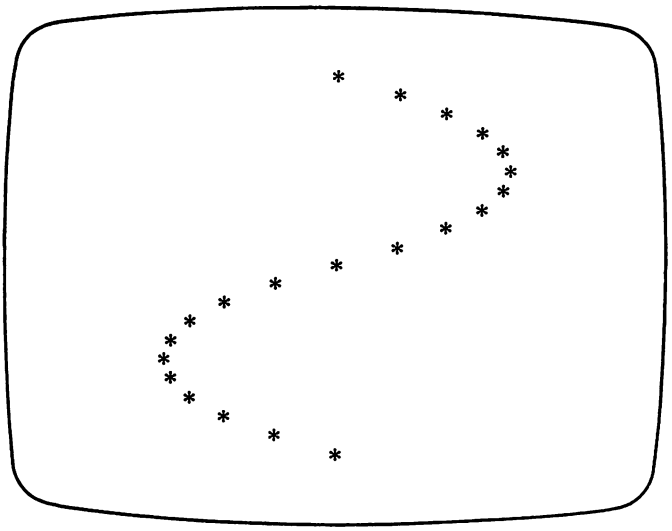


Fig. 7-16. Displaying the sine function on the TV screen.

5. Suppose the computer does not have a tangent (TAN) function. Write a program that compares the tangent of an angle to the sine divided by the cosine of the angle. Does this give us basically the same result? Can you quess how your computer determines the tangent of an angle?

8

Lights, Action

In this chapter, you will take a closer look at how you can control the TV screen display. You will see how you can use a computer program to position the cursor and how you can control the color of what is displayed on the TV screen. You'll also see how you can control the screen without using PRINT commands so that special displays can be put together.

THE SPACE COMMAND

In a previous chapter, you saw how the TAB command could be used to position columns of information so they were easy to read. There is another command that can be used to position information on the TV screen and it is called the space command, SPC. This is used within a PRINT command and it tells the computer to move the cursor a specific number of spaces to the right. At first it may seem as though the SPC and TAB commands do the same thing.

Both the TAB and SPC commands move the cursor to the right, but they differ in how they do it. A TAB(10) command moves the cursor 10 spaces to the right *from the left edge of the display area,* but an SPC(10) command moves the cursor to the right 10 spaces *from its present position.* These operations are shown in Fig. 8-1. Neither the SPC nor the TAB command will

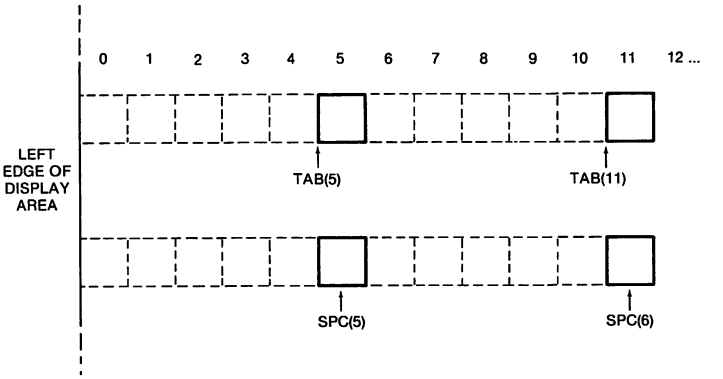


Fig. 8-1. How the TAB and SPC commands move the cursor.

erase or "destroy" any of the characters on the screen as they move the cursor from place to place.

Here is a sample program that uses the SPC command. You can type it into your computer and run it:

```
20 FOR S = 1 TO 21
30   PRINT SPC(S)"*"
40 NEXT S
50 FOR S = 20 TO 2 STEP -1
60   PRINT SPC(S)"*"
70 NEXT S
80 GOTO 20
```

MOVING FROM PLACE TO PLACE

The SPC and TAB operations are used to move the cursor along a line of information on the TV screen to display it in an orderly way. Sometimes, it is useful to have complete control of the cursor, moving it wherever you want. You can move the cursor in four directions by using the two cursor control keys and the SHIFT key. These same keyboard operations can be included in a program so that the computer moves the cursor when it runs the program. This is easy to do, since the cursor movement operations can be placed within the "message" part of a PRINT command.

If you are typing a "message" in a PRINT command and you use the cursor control keys, you'll find that the cursor doesn't move at all. Instead, each time you press one of the cursor control keys,

a special symbol is printed in the "message." Here is an example in which the DOWN cursor control key was pressed twice while the "message" was being typed:

350 PRINT " [reversed Q][reversed Q] "

The reversed "Q" displayed after the quote mark is the computer's notation to do a down-cursor movement when the "message" is used by the program.

When the cursor keys are pressed as a message is being typed, the VIC 20 realizes that you don't want the cursor moved right now. However, when the computer gets to this "message" in the program, it recognizes the special symbols and then it moves the cursor. Each of the four cursor movements prints a special symbol in a "message," and these are listed in Table 8-1. For example, when the computer gets to the following PRINT command, it will move the cursor down two lines and to the right two spaces before printing BIKES:

550 PRINT" [reversed Q][reversed Q][reversed] BIKES"

The cursor commands move the cursor from its *present position* to a *new position.* So, to use the cursor commands properly in a program, you need to know exactly where the cursor is before you start to move it. Maps in shopping centers and large buildings have a brightly colored arrow saying "YOU ARE HERE." Without this guide, it might be difficult to figure out where you are and where you want to go. Since the cursor can be easily sent to its home position in the upper left-hand corner of the TV screen, that's a convenient starting point for cursor movements.

Experiment No. 8-1. Moving the Cursor

This experiment will show you how a program can be used to position the cursor on the TV screen. Clear your computer and

Table 8-1. Cursor Control Symbols Used in VIC 20 Messages

Key	Symbol
DOWN	[reversed Q]
UP	[reversed]
RIGHT	[reversed]
LEFT	[reversed]

(NOTE: The UP and LEFT operations use the SHIFT key.)

type in the following program. You must press the cursor keys as shown in the PRINT messages. Check your program for any mistakes. Be sure to enter the semicolon (;) at the end of line 30:

```
10 PRINT "[SHIFT] [CLR/HOME]"
20 PRINT "[DOWN] [DOWN] [DOWN] [DOWN] [DOWN]"
30 PRINT "[RIGHT] [RIGHT] [RIGHT] [RIGHT] [RIGHT]";
40 PRINT "VIC 20"
```

When this program is listed, it will appear on the TV screen with the special symbols as follows:

```
10 PRINT "  ▗▖  "
20 PRINT "  ▚▚▚▚▚  "
30 PRINT "  ▌▌▌▌▌  ";
40 PRINT "VIC 20"
```

After you have checked your program and corrected any errors, run the program. What do you see? The "message," VIC 20, is displayed in the center of the screen. The READY message is also displayed when the computer finishes the program.

Here's something a bit more complicated. Let's try a counting program that displays increasing numbers but only at one place on the screen. *DO NOT CLEAR THE COMPUTER*, since the program you just typed in will still be used. Here's the new program for you to enter: (Remember to press the [SHIFT] and [UP/DOWN] keys for the [UP] operation.)

```
100 PRINT "[SHIFT] [CLR/HOME]";
110 PRINT "[DOWN] [DOWN] [DOWN] [DOWN] [DOWN] [DOWN]";
120 PRINT "[RIGHT] [RIGHT] [RIGHT] [RIGHT] [RIGHT] [RIGHT]";
130 PRİNT X
140 X = X + 1
150 PRINT "[UP]";
160 GOTO 120
```

What happens when you list this program on the TV screen? You'll see the program lines from the first part of the experiment, followed by the program lines you just entered. Both programs are present in the computer. The program lines 100 to 160 should look like this:

```
100 PRINT "  ▗▖  ";
110 PRINT "  ▚▚▚▚▚  ";
120 PRINT "  ▌▌▌▌▌  ";
```

```
130  PRINT X
140  X = X + 1
150  PRINT "  ❑  ";
160  GOTO 120
```

If you are confused by seeing all of the program lines on the TV screen at the same time, you can tell the computer to list only the lines that you want to look at. In this case, type LIST 100–200 [RETURN] and the computer will list only the program lines with numbers between 100 and 200.

Are you ready to run the program? Type RUN and put your finger above the RETURN key, but don't press it yet. Carefully watch the center of the TV screen as you press the RETURN key. Did you see the "VIC 20" message flash on quickly near the center of the TV screen when you started the program? What else happens on the TV screen?

The "VIC 20" message was displayed quickly, just above the number. You can stop the program and start it again if you missed seeing the "VIC 20" message displayed on the TV screen. You should be able to see "VIC 20" flash quickly on the TV screen.

Each higher number in the count "writes over" the one displayed before it, so the display counts quickly, much like a digital clock when it is being set to a new time.

The illustration in Fig. 8-2 shows how the cursor is moved by this program. The cursor is first moved down and to the right to place it near the center of the TV screen. After a number has been printed, the computer moves the cursor to the left side of the line below. This is the normal line-feed operation that automatically takes place after each PRINT operation unless a semicolon is used to prevent it. The cursor operation in the PRINT command at line 150 moves the cursor back up to the line above so that the next number will be printed "over" the previous one.

This is one way in which values are displayed in the same place in games, educational programs, and business programs. Do you know why the "VIC 20" message was displayed quickly on the TV screen when you started the program with the RUN command? The first program doesn't have an END command to tell the computer that it has reached the end of that program. So, after the computer displays the "VIC 20" message, it keeps going right into the second program. An END command is used to stop the computer at the end of each program so that this sort of thing doesn't happen.

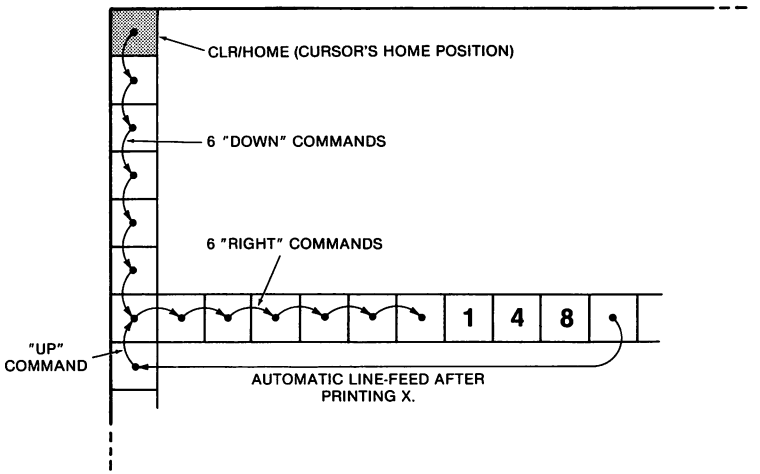


Fig. 8-2. The cursor movements for the number display program in Experiment 8-1.

In this experiment you have seen how the cursor control keys can be used in the "message" part of a PRINT command to control the movement of the cursor from one position to the next.

THE MAGIC COLOR SCREEN

You can also use a BASIC program to change the color of the characters displayed on the TV screen. This is done using special keyboard operations in a "message" in the same way that the cursor was moved. There are many times when it is useful to change to a different color to call attention to a word, number, or sentence on the TV screen.

The color of the cursor can be changed by using the CTRL key and the eight color keys that share their function with the keys numbered 1 through 8 in the top row of the keyboard, as shown in Fig. 8-3. If you press and hold the CTRL key and then press one of the color keys, the cursor will change to the color shown on the

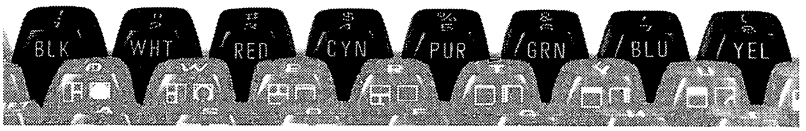


Fig. 8-3. The color control keys.

key. Now, when characters are typed, they will be displayed in the color you have selected. If you have a black-and-white TV set, you will see changes in the shades of grey as you change colors.

When the color of the cursor is changed, the characters already on the TV screen stay the same. *Only the characters printed after the color has been changed are displayed in the new color.*

You can also "reverse" the characters being displayed by using the CTRL key and the RVS ON and RVS OFF keys that share their functions with the 9 and 0 keys. When the display is "reversed" by using the CTRL and RVS ON keys, the cursor color becomes the background color, and the background color is used for the characters being displayed. This change won't be obvious from looking at the cursor, since its color and position stay the same. The "reverse" printing may be turned off by using the CTRL and RVS OFF keys. The reversed type of printing is often used to add emphasis to a display, or for a special visual effect. *Only the characters printed after the display has been "reversed" will be displayed in the reverse form.*

The color-change and reverse-printing operations can be included in the "message" part of a PRINT command, and they use special symbols to represent their actions. The symbols are shown in Table 8-2. The computer changes the color or displays characters in reverse only when the computer reaches these special symbols in the program. For example, when the computer gets to the following print command:

```
100 PRINT "  THIS IS RED"
```

it recognizes the symbol and switches the printing color to

Table 8-2. Color Control Symbols Used in VIC 20 Messages

Key	Symbol
BLK (Black)	
WHT (White)	
RED (Red)	
CYN (Cyan)	
PUR (Purple)	
GRN (Green)	
BLU (Blue)	
YEL (Yellow)	
RVS ON	
RVS OFF	

red. The message, THIS IS RED, is then printed on the TV screen as red letters on the background color. Once the color has been switched, the VIC 20 continues to use the selected color until it is changed by another color-change operation in the program, or by a keyboard operation. Several special operations can be put in a "message." For example, clear, home, cursor, color, and reverse operations have been combined in this command:

```
100 PRINT "                COMBO"
```

This PRINT operation clears the screen and moves the cursor to its home position . It then moves the cursor down three lines and over three spaces . The color is changed to green and the color is reversed . The word COMBO is printed as background-colored letters surrounded by a green area.

Experiment No. 8-2. Color Control

This experiment will show you how you can control the color of the TV display from within a program. If you have a black-and-white TV set, you will see changes in the shades of grey. Clear your computer for this experiment. Type in the following program:

```
 10  PRINT "[SHIFT] [CLR/HOME]"
 20  PRINT "[CTRL] [6]"
 30  X = 1
 40  PRINT X
 50  X = X + 1
100  GOTO 40
```

Here is how the program will appear on your TV screen:

```
 10  PRINT "  "
 20  PRINT "  "
 30  X = 1
 40  PRINT X
 50  X = X + 1
100  GOTO 40
```

What do you think this program will do? This program will display an increasing value on the screen; 1, 2, 3, 4, and so on.

What special symbol was displayed in line 20 when you typed the [CTRL] [6] color change? What will this do when the computer reaches this command? When you are typing characters on the

keyboard and you press/hold the CTRL key and then press the 6 key, the next characters you type will be colored green. However, when the CTRL and 6 keys are pressed in a "message," a special symbol is displayed. It looks like a tree or an umbrella. When the computer reaches the statement:

```
20 PRINT "  "
```

in the program, it will change the printing color to green. Anything that is printed will be green. Run the program. What is displayed on the TV screen? The TV screen is cleared and the numbers are displayed on the left side of the screen. The numbers are green. Stop the program by pressing the RUN/STOP key. Now list your program. What color are the characters in the program listing? The characters are all green. Do you know why?

There are no other color-change commands in your program, so the TV display will continue to stay green until you change it to another color. Can you change the color to black right from the keyboard? Try to do this. The color can be changed to black by pressing the CTRL key and then pressing the 1 key. This changes the color of the cursor and any characters that are then printed. If you list the program now, the listing will be printed in black.

Can you add a program line to change the color of the numbers when the value 100 is displayed? How can the computer make this decision? It can check for the condition, X = 100. Here is a program line that will change the display to blue when the value labeled X reaches 100:

```
60 IF X = 100 THEN PRINT "[CTRL] [7]";
```

When the program is listed, it should look like this:

```
 10 PRINT "  "
 20 PRINT "  "
 30 X = 1
 40 PRINT X
 50 X = X + 1
 60 IF X = 100 THEN PRINT "  ";
100 GOTO 40
```

After correcting any mistakes, run your program. You can press the CTRL key by itself to slow the display. What happens when the number 100 is displayed? Did the display change color? The number 100 is displayed in blue, and so are the numbers that follow it.

Experiment No. 8-3. Reversed Printing

In this experiment, you will find out how the reverse printing operation works and how it can be used in a program. Clear your computer and type the following program:

```
20  PRINT "[SHIFT] [CLR/HOME]"
40  FOR N = 1 TO 5
60    PRINT N
80  NEXT N
```

Run this program to test it. The numbers 1 through 5 should be displayed in the upper left-hand corner of the TV screen. When the program works properly, change line 20 so that a green color-change operation is included in the message:

```
20  PRINT "[SHIFT] [CLR/HOME]  [CTRL] [6]"
```

This will change the display color to green. The program should now look like this:

```
20  PRINT "      "
40  FOR N = 1 to 5
60    PRINT N
80  NEXT N
```

Run the program again. Do the numbers appear on the screen in green? The numbers should be printed in green on the normal background color. Now change the program so that line 20 also contains a reverse-on operation, RVS ON:

```
20  PRINT "[SHIFT] [CLR/HOME] [CTRL][6] [CTRL][9]"
```

This will change the listing of the program so it looks like this:

```
20  PRINT "      "
40  FOR N = 1 TO 5
60    PRINT N
80  NEXT N
```

Run the program. Are the numbers printed in reverse, as the background color surrounded by green? No, they are not. Unlike the color-changing operations, you cannot set the VIC 20 to the reverse mode and have it stay there. *The reverse mode must be set for each printing command in which it is to be used.* Here is how you can do it. Change line 20 so it contains only the clear-home and green color-change operations:

```
20 PRINT "[SHIFT] [CLR/HOME]   [CTRL][6]"
```

Change line 60 so that the reverse printing operation is contained in the "message" part of the PRINT command:

```
60 PRINT "[CTRL][9]" N
```

Check your program with the listing shown below:

```
20 PRINT "  ■■  "
40 FOR N = 1 TO 5
60   PRINT "  ■  " N
80 NEXT N
```

Run the program. Are the numbers printed in reverse? You should see that the numbers are printed in the background color, with a green area around them.

NOTE: If you want to use the reverse printing operation in a program, you must set it in each of the PRINT commands in which you want to use it. The reverse printing will be set back to the normal mode whenever (1) the VIC 20 does a line-feed on the display, (2) the RETURN key is pressed, or (3) the program stops. The colors are not changed by these operations, and the cursor's color will stay as it is unless you change it.

The computer can do many interesting things on the TV screen by using the PRINT command and the cursor, color, and reverse operations in the "message." All of the letters, numbers, punctuation marks, and graphic symbols may be printed on the screen to make colorful displays of information or art. By adding color to displays, you can add a great deal of value and interest to games and to education, business, and other programs.

The PRINT command is very useful and flexible. There are many ways in which it may be used, and you will find that the PRINT command will answer almost all of your needs to display information on the TV screen. However, there are some times when it would be helpful to have even greater control of the TV screen.

ADVANCED DISPLAYS AND GRAPHICS

The advanced displays and graphics that we will describe in this section are based on the "5K VIC 20 system." This is the basic VIC 20 computer system as it comes from the store or computer dealer. The 5K VIC 20 must not be expanded with any extra memory cartridges if you want to do the experiments in the rest of this

chapter. If you look at the back of the computer, you'll find a large open slot. This is where game cartridges and other plug-in modules can be added. If there is a module in this slot, we suggest that you turn off the power and remove it.

The VIC 20 can display up to 23 lines of information, and each line can have up to 22 characters. So, there are 506 positions for characters on the TV screen. Do you think it's possible to "fill" them all with information?

Experiment No. 8-4. Filling the TV Screen

This experiment will let you see whether or not you can completely fill the TV screen with characters.

Here is a program that will print 506 "@" signs on the TV screen. Clear your computer for a new program and type this program into the computer:

```
100  PRINT "[SHIFT] [CLR/HOME]";
120  FOR W = 1 TO 506
140    PRINT "@";
160  NEXT W
```

Run the program. Is the screen completely filled with 506 "@" symbols? It is difficult to tell, since as soon as the program ends, the READY message is displayed on the TV screen. Add the following program line to your program:

```
180  GOTO 180
```

This will keep the computer in an "endless" loop so the program will never end and the READY message will not appear.

Run the program again. Are all 506 "@" signs displayed? It looks as though they might be, but as soon as the last one is printed, the display moves up, leaving four blank lines at the bottom of the display. Stop the program with the RUN/STOP key and change line 120 to:

```
120  FOR W = 1 TO 505
```

Now run the program again. What is displayed on the TV screen? The TV screen is completely filled with "@" symbols, except for the last space in the TV display, in the lower right-hand corner.

When the VIC 20 displays a character in the last space on a line, or in the last space on the display, it automatically moves the cur-

sor to the next line. At the bottom of the display, it will move the display up by one to four lines. You can't always tell how many. This is an annoyance, and it makes it difficult to display information in the last place on a line if you don't want the cursor to automatically move.

When your program printed the last "@" symbol on the page, the VIC 20 moved the display up by four lines. If you had had useful information at the type of the display, the first four lines would have been "lost."

The PRINT command is used extensively in computer programs, but in the VIC 20 it has some limitations, and there is another way to control the TV display. So far, the screen has appeared to be individual lines, each of which can hold characters. You can also think of the TV display as a grid of squares, each of which can hold one character, as shown in Fig. 8-4. The grid, or "map," is laid out in the same way in which characters are displayed on the TV screen; 22 squares across the top and 23 squares down the side.

This is the way that the VIC 20 actually arranges the information that is shown on your TV set. When information is to be printed on the TV screen, the computer places the necessary characters in the map squares, one by one. As far as we are concerned, this is automatic and completely taken care of by the computer.

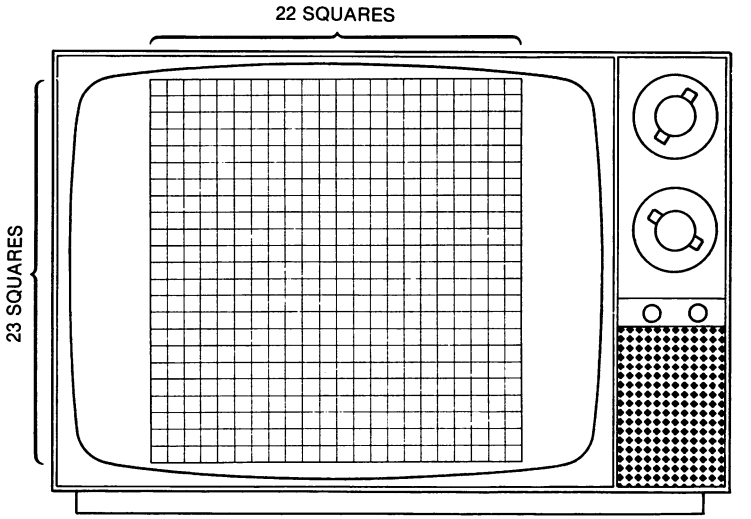


Fig. 8-4. The TV screen grid: 22 characters across by 23 display lines.

The overall result is that the characters show up in matching positions on the TV screen.

The squares used to hold the TV display information are grouped in a *display map*, and they are numbered in sequence, just like bins in a warehouse or boxes in a bank safe. In the basic 5K VIC 20, the display map numbers start at 7680 and end with 8185, as shown in Fig. 8-5. Any square on the map can be easily located, since each has a unique number or "address." Individual characters can then be placed in each square so they are displayed on the TV screen.

The VIC 20 has a second map that looks exactly like the display map, except that its addresses start with 38400 and end with 38905. This is the *color map*, and it is used to set the color of each of the characters displayed on the TV screen. Each position on the TV screen has a matching position in the display map and in the color map, as you can see in Fig. 8-6. You can think of the display map as holding the character outlines and the color map as holding colored lights that shine through to the TV screen.

The two maps are used together so that when a character is placed in the display map at address 8010, its color must be placed

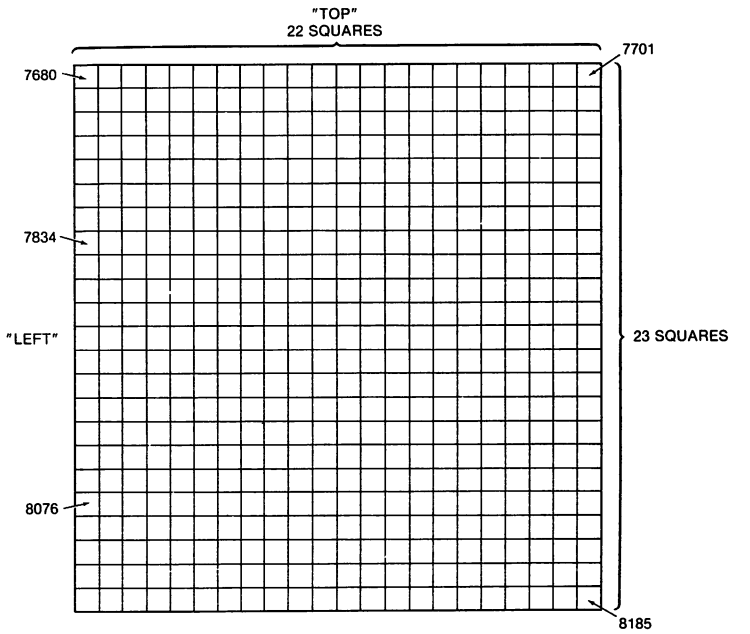


Fig. 8-5. The screen display map.

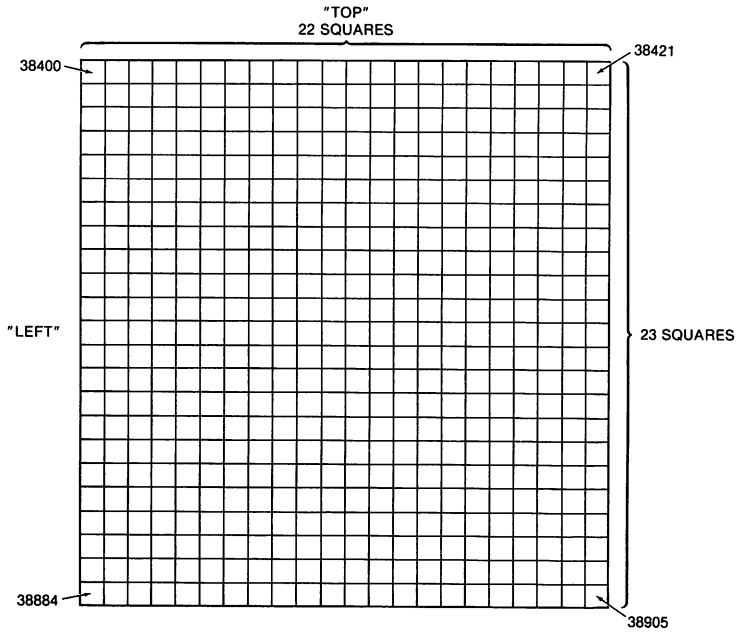


Fig. 8-6. The screen color map.

in the color map at address 38730, which is right "behind" it, providing the colored light. The addresses in the color map are 30720 greater than the matching addresses in the display map, so it's not too difficult to keep track of them.

If you could look into the squares in the maps, you wouldn't find the actual characters and colors. The VIC 20 uses a special "screen code" for each character and a "color code" for each color. For example, the screen code for "P" is 16 and the color code for red is 2. So, by placing the proper *screen code* in one of the display-map squares *and* a *color code* in the matching color-map square, the colored character appears on the TV screen. This is shown in Fig. 8-7.

The characters and their screen codes are shown in Table 8-3. The VIC 20 has two sets of characters, either of which you can use. We'll show you how to switch between them later in this chapter. The color codes are listed in Table 8-4.

A new command, POKE, is used to "address" a square in the display and color maps and place, or "poke," a code into it. The POKE command looks like this:

```
POKE address, code
```

Table 8-3. Screen Codes for the Display Characters

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	U	u	21	*		42
A	a	1	V	v	22	+		43
B	b	2	W	w	23	,		44
C	c	3	X	x	24	—		45
D	d	4	Y	y	25	.		46
E	e	5	Z	z	26	/		47
F	f	6	[		27	ø		48
G	g	7	£		28	1		49
H	h	8	]		29	2		50
I	i	9	↑		30	3		51
J	j	10	←		31	4		52
K	k	11	SPACE		32	5		53
L	l	12	!		33	6		54
M	m	13	"		34	7		55
N	n	14	#		35	8		56
O	o	15	$		36	9		57
P	p	16	%		37	:		58
Q	q	17	&		38	;		59
R	r	18	'		39	<		60
S	s	19	(		40	=		61
T	t	20	)		41	>		62

Courtesy Commodore Business Machines, Inc.

Table 8-3. Screen Codes for the Display Characters (cont)

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
?		63		T	84			106
		64		U	85			107
	A	65		V	86			108
	B	66		W	87			109
	C	67		X	88			110
	D	68		Y	89			111
	E	69		Z	90			112
	F	70			91			113
	G	71			92			114
	H	72			93			115
	I	73			94			116
	J	74			95			117
	K	75	SPACE		96			118
	L	76			97			119
	M	77			98			120
	N	78			99			121
	O	79			100		✓	122
	P	80			101			123
	Q	81			102			124
	R	82			103			125
	S	83			104			126
					105			127

Courtesy Commodore Business Machines, Inc.

Table 8-4. Screen Color Codes

Color	Color Code
Black	0
White	1
Red	2
Cyan	3
Purple	4
Green	5
Blue	6
Yellow	7

To display a character on the TV screen at a specific spot, the matching display map and color map squares must have a screen code and a color code poked into them. Two POKE commands, one for the display map and one for the color map, are used to print the letter "M" in red near the middle of the screen, at display map address 7932. The matching color address is 7932 + 30720, or 38652:

```
100  POKE 7932, 13  :  POKE 38652, 2
```

In this example, the screen code (13) was poked into the display map and then the color code (2) was poked into the color map. The same thing happens even if the POKE commands are reversed:

```
100  POKE 38652, 2  :  POKE 7932, 13
```

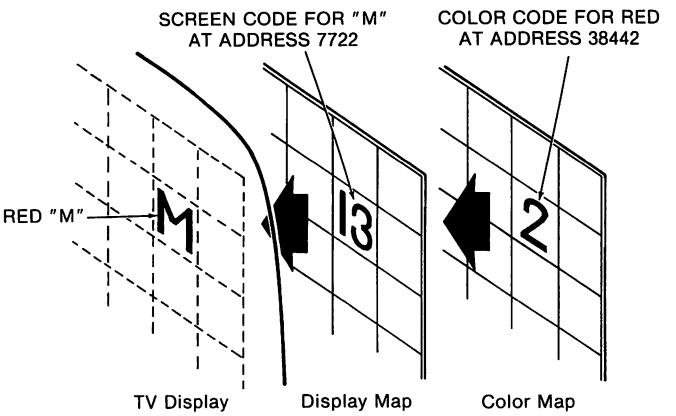


Fig. 8-7. Alignment of the screen display and color maps for TV display control.

A POKE command can be used to address any of the display-map or color-map squares at any time in a program. When the POKE command is used, its action is independent of the cursor and it will not cause line feeds to take place, nor will it advance or "push up" the display.

Experiment No. 8-5. Using the POKE Command

In this experiment, you will learn how the POKE command can be used to control the display of characters and their colors on the TV screen. You will use the POKE command to control the display and color maps. Clear your computer and type in the following program:

```
220  FOR AD = 7680 TO 8185
240    POKE AD, 42
260    POKE AD + 30720,2
280  NEXT AD
300  GOTO 300
```

This program will do two things. The FOR-NEXT loop will go through all of the addresses in the display map, and the first POKE command will put the display code (42) for an asterisk (*) in a display-map square. The second POKE command will put the red color code (2) in the matching color-map square.

Run the program. What is shown on the TV screen? Are all of the positions "filled" with an asterisk? The entire screen fills with red asterisks. Change line 220 to:

```
220  FOR AD = 7680 TO 8184
```

This should leave the last space on the display "blank." Run the program and see if this is what happens. The last space should be left blank. Change line 220 back to:

```
220  FOR AD = 7680 TO 8185
```

and run the program. The complete display should be filled again.

The display can also be "filled" from the bottom. Change line 220 to:

```
220  FOR AD = 8185 TO 7680 STEP -1
```

and run the program again. The display of asterisks should start at the bottom right-hand corner and end in the upper left-hand corner. It would be difficult to do this type of printing with a PRINT command. Change line 220 back to:

```
220  FOR AD = 7680 TO 8185
```

Now, let's change the program so that different colors are selected as the program is running. The random number command (RND) can select a random color code that can be poked into the color map. Add the following line to your program:

```
250  CL = INT(RND(1) * 8)
```

and change line 260 to:

```
260  POKE AD + 30720, CL
```

These new program lines will come up with a random number between 0 and 7, and will place it in the color map as the asterisks are printed on the TV screen. Here is a listing of the complete program:

```
220  FOR AD = 7680 TO 8185
240    POKE AD,42
250    CL = INT(RND(1) * 8)
260    POKE AD + 30720, CL
280  NEXT AD
300  GOTO 300
```

Run the program and watch the display. Are different colored asterisks shown as you expected? There are colored asterisks printed on the screen, but some asterisks seem to be "missing." Do you know why? These "missing" asterisks aren't missing at all. The color code used for them is 1, for white. A white character printed on a white background cannot be seen on the TV display. Change line 300 to:

```
300  GOTO 220
```

This command will point the computer back to the beginning of the program. What will this do to the TV display? Run the program and find out. The asterisks stay on the screen, but their colors change as the computer goes through the program again and again.

This experiment has shown you how POKE commands can be used to control the TV display so that characters of any color can be displayed anywhere on the TV screen. You also saw an interesting use for the random number operation.

Once a character has been displayed on the TV screen, you can change its color by changing the color code placed in its matching square in the color map. You can also change the character being

displayed and do many other interesting things with the display maps.

REVERSED CHARACTERS

If you look at the characters in Table 8-3, you'll see that none of them is shown in its reversed mode; that is, the character printed in the background color in a block of another color. Each character can be reversed just by adding 128 to its screen code. For example, 3 is the code for "C" and 131 is the code for a reversed "C," . It is easy to switch to the reverse mode by adding 128 to a character's screen code. By subtracting 128 from a reverse character's screen code, the character goes back to normal.

Experiment No. 8-6. A Flashing Sign

Flashing signs are used to get your attention, and computer displays use the same technique. The cursor flashes to show you where it is, and flashing legends and messages can be useful, too. In this experiment, you'll see how a message can be flashed on the screen.

Clear your computer and type this program into your computer:

```
100  PRINT "[SHIFT] [CLR/HOME]";
120  DATA  4, 1, 14, 7, 5, 18
140  FOR AD = 7908 TO 7913
160    READ CH
180    POKE AD, CH
200    POKE AD + 30720, 2
220  NEXT AD
```

This program uses a DATA statement to hold screen codes for the letters D, A, N, G, E, and R so that "DANGER" can be printed on the TV screen. The FOR-NEXT loop reads the codes from the DATA statement and pokes them into the display map. The POKE command at line 200 sets the color of each letter to red.

Run the program. You should see the word DANGER displayed near the center of the TV screen. If it is not displayed, carefully check your program. To flash the word DANGER on the TV screen, you display it in the same place, but in reverse. This is done by using the same screen codes, but with 128 added to each one. Add the following lines to your program:

```
240  FOR T = 1 TO 1000: NEXT T
260  RESTORE
280  FOR AD = 7908 TO 7913
300    READ CH
320    POKE AD, CH + 128
340  NEXT AD
360  RESTORE
380  FOR T = 1 TO 1000: NEXT T
400  GOTO 140
```

These program lines place "reversed" screen codes in the same display map squares as were used in the first part of the program. The difference is that 128 has been added to the original codes. Since the color code has already been poked into the map once, this doesn't have to be done again. The FOR-NEXT loops at lines 240 and 380 are "do-nothing" loops used to slow the computer so you can see the display change. The RESTORE commands reset the READ command so it starts from the beginning of the list of codes in the DATA statement. Here is the complete program:

```
100  PRINT "[SHIFT] [CLR/HOME]";
120  DATA  4, 1, 14, 7, 5, 18
140  FOR AD = 7908 TO 7913
160    READ CH
180    POKE AD, CH
200    POKE AD + 30720, 2
240  FOR T = 1 TO 1000: NEXT T
260  RESTORE
280  FOR AD = 7908 TO 7913
300    READ CH
320    POKE AD, CH + 128
340  NEXT AD
360  RESTORE
380  FOR T = 1 TO 1000: NEXT T
400  GOTO 140
```

Run the complete program and see if DANGER is flashed on the screen. It should be. If it is not, carefully check your program.

This program flashed the word DANGER by changing the display from the normal letters to the reversed letters. Another type of display might turn the letters on and off. Let's try that, too.

Once the screen codes have been placed in the display map, the characters are "lit" by poking a color code into the color map. In

the asterisk display experiment, you saw that if the color code was the same as that for the background, an asterisk wasn't seen. You can turn off characters, or "remove them," simply by setting their color code to the background color code.

Change your program by removing lines 200 through 400. Type the line number for a line to be "erased" and press [RETURN]. This removes the line from your program. The lines remaining should be:

```
100  PRINT "[SHIFT] [CLR/HOME]";
120  DATA  4, 1, 14, 7, 5, 18
140  FOR AD = 7908 TO 7913
160    READ CH
180    POKE AD, CH
```

Add the following line to the program:

```
200  NEXT AD
```

Run the program. Does the word DANGER appear on the TV screen? No, it doesn't, since no color codes were placed in the color map as part of this program. Add the following lines to the program:

```
220  FOR AD = 7908 TO 7913
240    POKE AD + 30720, 2
260  NEXT AD
```

Now run the program. Lines 220, 240, and 260 set the matching squares in the color map to red, so the DANGER message should show up on the TV screen. Now add these lines to the program to make the message flash:

```
280  FOR T = 1 TO 1000: NEXT T
300  FOR AD = 7908 TO 7913
320    POKE AD + 30720, 1
340  NEXT AD
360  FOR T = 1 TO 1000: NEXT T
380  GOTO 220
```

The complete program will look like this:

```
100  PRINT "[SHIFT] [CLR/HOME]";
120  DATA  4, 1, 14, 7, 5, 18
140  FOR AD = 7908 TO 7913
160    READ CH
```

```
180    POKE AD,CH
200  NEXT AD
220  FOR AD = 7908 TO 7913
240    POKE AD + 30720, 2
260  NEXT AD
280  FOR T = 1 TO 1000: NEXT T
300  FOR AD = 7908 TO 7913
320    POKE AD + 30720, 1
340  NEXT AD
360  FOR T = 1 TO 1000: NEXT T
380  GOTO 220
```

This seems like a lot of trouble to go to to flash a message on the screen, but it shows you how the display of information can be controlled by a program. Since you changed the color of the letters from red to white, back to red, and so on, you can change the color code in line 320 so that DANGER goes back and forth between red and yellow letters. Try it.

SEEING WHAT YOU HAVE DONE

You have used the POKE command to place codes in the display and color maps. The VIC 20 also gives you a command that you can use to take a look into the maps to see what code is stored in a particular square. This operation uses the PEEK command, and it lets you "peek" into any square to get the code that has been put there. Here is what the PEEK command looks like:

```
PEEK (address)
```

When you give the PEEK command an address, the computer will *get a copy of the code in that square.* The PEEK command is often used with a label so you can use the coded information from the map in a program. For example, the PEEK command:

```
100  DR = PEEK(38400)
```

will get the color code from the upper left-hand square in the color map (see Fig. 8-6). The PEEK operation *copies* the code, and it doesn't change the color map in any way.

How could this operation be used in a program? One simple use would be to have the program get screen codes from the display map so the characters could be "reversed." Here is how it might be done:

```
680 SC = PEEK (AD)
700 RC = SC + 128
720 POKE AD, RC
```

We assumed that the screen code wasn't already "reversed." The PEEK command isn't used as often as the POKE command, but it serves a useful purpose, and it's helpful to know that this type of look-and-see operation is available.

MORE CHARACTERS

Two sets of characters are shown in Table 8-3, but so far only Set 1 has been used. You can switch to the second set of characters by pressing the SHIFT and GRAPHICS keys at the same time. If you do this, you will see that the entire display is changed as the computer switches to the second set of characters. The second set of characters will continue to be used until you switch the computer back to Set 1.

You can also change to the second set of characters from within a program, and this is done with a POKE command that uses address 36869. For example, this instruction will switch the VIC 20 to the characters in Set 2:

```
100 POKE 36869, 242
```

while this command:

```
200 POKE 36869, 240
```

will switch the VIC 20 back to Set 1. Unlike the color and reverse printing operations, *the switch from one character set to the other immediately switches the whole TV display to the new character set.*

Since the computer uses Set 1 when it is turned on, you don't have to do anything to the computer to use the characters in that set. In most cases, you won't be switching back and forth between the two sets of characters.

SCREENS AND BORDERS

When you turn on the VIC 20 computer, the color of the square central area is white, and the surrounding border color is light blue, or cyan. You can change these colors, too, by using a POKE command. There are 128 combinations of these colors, and they are

listed in Table 8-5. To change the colors, simply choose the set of colors you want, get the code from the table, and use it in the command:

```
POKE 36879, code
```

The code used by the VIC 20 when it is turned on, or reset, is 27 for a white central area and a cyan border.

So far, you have seen how the computer uses "maps" and codes to control the characters displayed on the TV screen and their colors. You'll find that the VIC 20 has other maps, with their own addresses, and these control other parts of the computer. For example, there is a small "map" that controls the sounds that the computer can make.

A NOTE ABOUT CODES

In the Special Math Functions chapter, we introduced the American Standard Code for Information Interchange, or ASCII. This code is a standard one that has been adopted by computer equipment manufacturers so that information can be easily transferred from one computer to another. You learned about the CHR$ and ASC operations that can convert string characters to their ASCII equivalent and back again.

The screen codes use in the VIC 20 computer are different from

Table 8-5. Screen and Border Color Combination Codes

SCREEN COLOR	BORDER COLOR							
	BLK	WHT	RED	CYN	PUR	GRN	BLU	YEL
BLACK	8	9	10	11	12	13	14	15
WHITE	24	25	26	27	28	29	30	31
RED	40	41	42	43	44	45	46	47
CYAN	56	57	58	59	60	61	62	63
PURPLE	72	73	74	75	76	77	78	79
GREEN	88	89	90	91	92	93	94	95
BLUE	104	105	106	107	108	109	110	111
YELLOW	120	121	122	123	124	125	126	127
ORANGE	136	137	138	139	140	141	142	143
LT. ORANGE	152	153	154	155	156	157	158	159
PINK	168	169	170	171	172	173	174	175
LT. CYAN	184	185	186	187	188	189	190	191
LT. PURPLE	200	201	202	203	204	205	206	207
LT. GREEN	216	217	218	219	220	221	222	223
LT. BLUE	232	233	234	235	236	237	238	239
LT. YELLOW	248	249	250	251	252	253	254	255

the ASCII values assigned to each letter, number, and punctuation mark. Most of the special graphic characters that can be displayed with the VIC 20 have no ASCII equivalent. Likewise, there are no ASCII equivalents for reversed characters. Although you won't be concerned with this right now, *remember that the screen code and ASCII value for a character are not the same thing!*

QUESTIONS

1. What does the SPC, or space, command do?

2. How does the SPC command differ from the TAB command?

3. What happens when you are typing a "message" for a PRINT command and you try and use the CRSR keys?

4. What commands can be used in a PRINT command to control the cursor?

5. What does the END command do? Why is it useful?

6. How do you get the VIC 20 computer to change the color of its display?

7. When the color is changed, does this change the color of everything being displayed?

8. How can the color be changed from within a program?

9. How do you get the computer to print in reverse? Can you set the computer to the reverse mode and use it as you wish?

10. What two maps control the TV display? How many locations are there in each map?

11. What information must be placed in the two maps to control the TV display?

12. How can a character be "reversed" on the TV screen?

13. What does the POKE command do? What does the PEEK command do?

14. How many sets of display characters does the VIC 20 computer have? How can you switch from one to the other? Does this "switch" change all of the characters on the TV screen?

PROBLEMS

1. Write a program so that the background color on the TV screen is changed to green.
2. Using POKEs, move a red ball ([SHIFT][Q]) along the top line of the TV screen from left to right. Once the ball reaches the right-hand side of the TV screen, it should bounce back toward

the left-hand side of the screen. The red ball should continue to bounce off the sides of the TV screen.

3. Write a program so that a black diamond ([SHIFT][Z]) is moved about on the top line of the TV screen in a random manner. This means that the diamond can either (1) stay where it is, (2) move one position to the left, or (3) move one position to the right.

4. Display a sine wave on the TV screen (Fig. 8-8), assuming that the angle varies between 0 and 2 π radians (0 and 360°). If you aren't comfortable using the SIN function (from Chapter 7), skip this and the next problem.

5. Modify the program from Problem 4 so that in addition to plotting a sine wave on the TV screen, a coordinate system is also plotted (Fig. 8-9).

6. Write a program so that a red asterisk (*) moves around the edge of the TV screen in a clockwise manner.

7. Write a program that asks you to enter a number that represents the border color and a number that represents the background color. Once both numbers have been entered, these

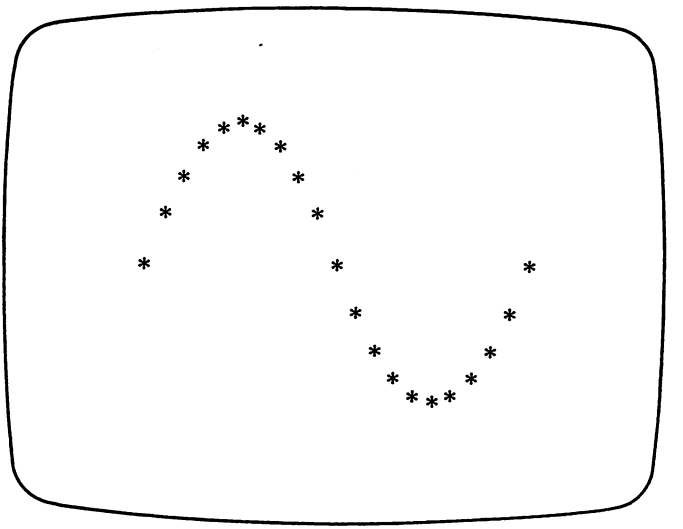


Fig. 8-8. Displaying a sine wave on the TV screen.

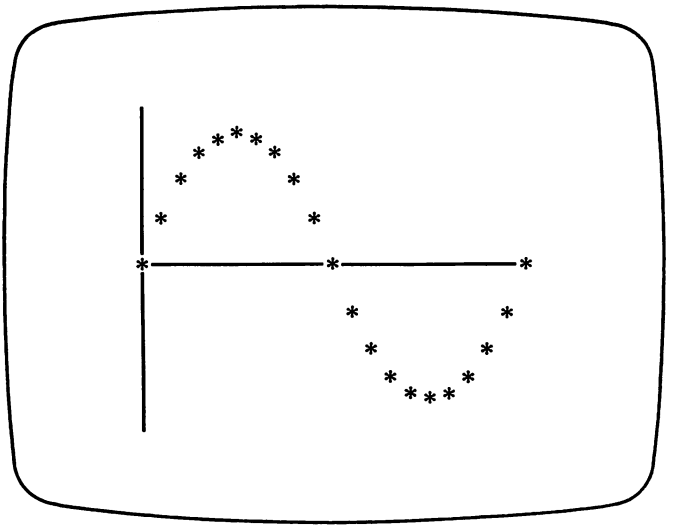


Fig. 8-9. Adding a coordinate system to the sine-wave display.

colors are actually changed on the TV screen. For this problem, use the "menu" approach of entering a selection's number shown in Problem 5 in Chapter 5.

8. Write a program that displays the up-arrow character (↑) in the middle of the bottom line of the TV screen. By pressing the "V" key, the arrow moves to the left, and by pressing the "N" key, the arrow moves to the right. (HINT: Use the GET$ command in your program).

9. Write a program that lets you enter a character (A–Z) from the keyboard, which is then displayed in the center of the TV screen. The cursor keys can then be used to move the character around on the TV screen.

10. Modify the program from Problem 9 so that as the character is moved, it leaves a trail of characters, the same way the *Etch-O-Sketch®* toy works.

11. Modify the program from Problem 8 so that the arrow shoots the period character (.) toward the top of the screen. Once a shot has been taken, no other shots can be taken until the first bullet goes off the top of the screen. As the bullet moves

toward the top of the screen, the "V" and "N" keys should still be able to move the arrow around on the bottom line.

12. Write a program that puts three asterisks (*) on each line of the TV screen, in random places in each line. The color for each line of asterisks should also be randomly chosen. The program should not allow the screen color to be used, thus "invisible" asterisks are avoided.

13. Write a program so that an asterisk is placed on the left-hand side of the TV screen on a randomly chosen line. The asterisk is then moved from the left-hand side of the TV screen to the right-hand side. Once the asterisk disappears off of the right-hand side of the screen, another asterisk should appear on the left-hand side of the screen on a randomly chosen line, and it, too, should move over to the right-hand side of the screen.

14. Modify the program from Problem 13 so that an asterisk appears randomly on either the left-hand or right-hand side of the TV screen on a randomly chosen line. The asterisk should then move toward the other side of the screen. Once the asterisk reaches the other side of the screen, this process should be repeated.

15. Modify the program from Problem 14 so that at the beginning of the program the user can enter a skill level between 0 and 9. The higher the skill level, the faster the asterisk travels across the screen.

16. Modify the program from Problem 12 so that once the TV screen is filled, a new line of three randomly placed asterisks will appear at the bottom of the screen. This will cause the top line on the screen to be scrolled off of the screen. The three new asterisks should all be the same (randomly chosen) color.

17. If you have solved every problem so far, and you are a VIC 20 "expert," try the following for some extra credit: Modify the character set of the VIC 20 so that the expression, X↑2, will be displayed with the 2 actually appearing as a superscript, slightly above the X. To properly solve this problem, the 2 must be the same height as the X character.

9

Sound Off, 1, 2, 3, 4

SOUND IS IMPORTANT

If you can use more of your senses in something, the more enjoyable it can be. Pictures of animals just can't be compared to a trip to a zoo where the animals can be seen, heard, touched, and even smelled. Many an early video game used only a black-and-white TV set to display a set of "paddles" and a "ball" that moved around the screen. Later, sound effects and realistic color displays were added, and more people started playing video games.

Unfortunately, many computers just use a TV screen to display letters and numbers, so using a computer isn't as interesting as it could be. In an earlier chapter, you learned how to use colors to call attention to information and to interest people in what the computer is doing. In this chapter, you'll see how the VIC 20 can make special sounds, so your sense of hearing can be involved, too.

Besides being used in game programs for sound effects, sounds have other important uses in small computers. In some, a quick "tick" is heard each time a key is pressed. If you are typing information without looking at the keyboard, this lets you know that each key has really been pressed. The "tick" is reassuring, and it takes the place of the "snap" you hear from a typewriter as the typeface hits the paper.

Sounds can also be used to call attention to something the computer is doing. Perhaps a quick "beep" will let you know you've pressed a wrong key, or a warbling tone will tell you that something special is about to happen. Sounds add a lot to computer systems.

SOUND FROM THE VIC 20

Your VIC 20 computer can put together or create many sounds, and they are easy to control with a BASIC program. The VIC 20 has four sources of sounds, or tones, and they are called "speakers," since they are similar to the speaker in a radio or hi-fi. The computer's "speakers" can be used individually, or together, depending on what kind of sound is needed. The sounds don't come out of the keyboard unit, but are sent to the TV set and are heard on its built-in speaker.

Three of the "speakers" create musical tones, from low tones at "speaker 1" to high tones at "speaker 3." We'll call them SPKR1, SPKR2, and SPKR3. The fourth speaker, SPKR4, creates a "hiss," or noise, like that from a radio set that isn't receiving anything. This sound is also called "white noise." A simplified diagram of the computer's sound system is shown in Fig. 9-1.

The VIC 20 uses a map to control these sounds, and the *sound map* is shown in Fig. 9-2. There are five adresses in the map, one for each "speaker" and one for the volume. Each of these is controlled by placing a code in the matching map address. The codes for the sounds go from 128 to 255, so each "speaker" can create 128 tones. The codes for the "musical" tones are shown in Table 9-1.

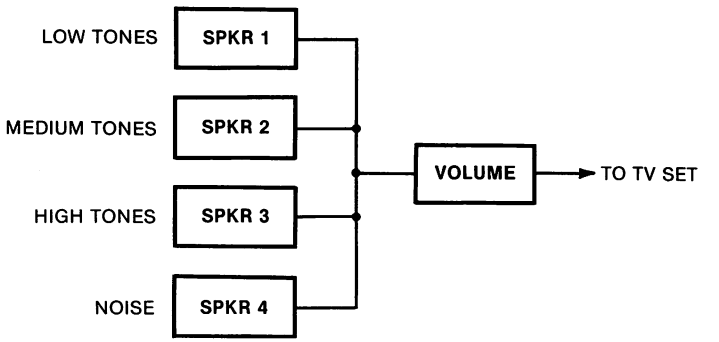


Fig. 9-1. Block diagram of the VIC 20 sound system.

ADDRESS	FUNCTION	(RANGE)
36874	LOW TONE	(128-255)
36875	MEDIUM TONE	(128-255)
36876	HIGH TONE	(128-255)
36877	NOISE	(128-255)
36878	VOLUME	(0-15)

Fig. 9-2. The sound map and addresses for the VIC 20.

The tones have been put into groups of eight, and each is called an "octave." Not all of the tones have been shown, since many don't correspond to specific musical notes.

There is some overlap between the tones, so that a code of 223 at SPKR3 creates the same tone as a code of 239 at SPKR2. This means that the same tone may be created by using a code at one speaker, and a different code at another speaker.

The volume control sets the volume of the complete sound created by the computer, whether it is coming from only one speaker or from all of them. The volume settings go from 0 to 15, with 0 the code for OFF and 15 the code for the highest volume.

The sound map operates in the same way as do the display and color maps that were described in an earlier chapter. The tones and the volume are controlled by using POKE commands to place the codes in the proper squares in the sound map. You can use the PEEK command to find out what code is stored in any square in the sound map.

Experiment No. 9-1. Setting the Volume

Sounds are easy to create and you will find out how you can do this in this experiment. You will also find out how to use a tone code and the "volume control."

Clear your computer for a new program. Turn the volume control on your TV set so that it is at the lowest volume setting. At the highest setting you will hear a hum or a slight hiss. You want the position opposite from this. Type the following instructions into your computer:

Table 9-1. Musical Tone Codes for the VIC 20 Speakers

SPKR 3		SPKR 2		SPKR 1	
Note	Code	Note	Code	Note	Code
B	239	—	—	—	—
A	237	—	—	—	—
G	235	—	—	—	—
F	232	—	—	—	—
E	231	—	—	—	—
D	228	—	—	—	—
C	225	—	—	—	—
B	223	B	239	—	—
A	219	A	237	—	—
G	215	G	235	—	—
F	209	F	232	—	—
E	207	E	231	—	—
D	201	D	228	—	—
C	195	C	225	—	—
B	191	B	223	B	239
A	183	A	219	A	237
G	175	G	215	G	235
F	163	F	209	F	232
E	159	E	207	E	231
D	147	D	201	D	228
C	135	C	195	C	225
—	—	B	191	B	223
—	—	A	183	A	219
—	—	G	175	G	215
—	—	F	163	F	209
—	—	E	159	E	207
—	—	D	147	D	201
—	—	C	135	C	195
—	—	—	—	B	191
—	—	—	—	A	183
—	—	—	—	G	175
—	—	—	—	F	163
—	—	—	—	E	159
—	—	—	—	D	147
—	—	—	—	C	135

```
POKE 36878,15
POKE 36876,200
```

These instructions immediately set the computer's sound volume to its highest setting and set up a tone in SPKR3. Adjust the volume control on your TV set so that the tone can be heard at a normal level. Since you will be using the tones again, you want to be able to easily hear the tone.

Once the volume has been set, type the following instruction into your computer. This will turn off the tone:

```
POKE 36878,0
```

Leave the TV set's volume control at its present setting for now. Load the following program into the computer:

```
100  POKE 36876,200
120  FOR VL = 0 TO 15
140    INPUT Z
160    PRINT VL
180    POKE 36878, VL
200  NEXT VL
```

This program sets up a tone on SPKR3 and increases the volume code by 1 each time you press the RETURN key. Run the program and listen for the tone. Just press the RETURN key to advance the computer to the next volume setting. Do you hear a difference in the volume of the sound after pressing the RETURN key? You will probably be able to hear the volume change at the lower settings, but once above a volume code of 12, or so, it is difficult to hear much of a change. To turn off the tone, type:

```
POKE 36878,0
```

You can also turn off the sound by pressing the RUN/STOP and RESTORE keys together. This is the easiest way to turn off the sound and it can be done at any time. Of course, it will stop any program that you are running, too. Both of these operations set the volume code back to 0, or OFF. Change program line 140 to:

```
140    FOR T = 1 TO 100: NEXT T
```

Here is a complete listing of the modified program:

```
100  POKE 36876,200
120  FOR VL = 0 TO 15
140    FOR T = 1 T0 100: NEXT T
160    PRINT VL
180    POKE 36878, VL
200  NEXT VL
```

Now run the program. What happens? The volume slowly increases as the volume codes are printed on the TV screen. Add the following program lines to your program:

```
220  FOR VL = 15 TO 0 STEP -1
240    FOR T = 1 TO 100: NEXT T
```

```
260    PRINT VL
280    POKE 36878, VL
300  NEXT VL
```

Check your program listing against the listing of the complete program shown below:

```
100  POKE 36876,200
120  FOR VL = 0 TO 15
140    FOR T = 1 TO 100: NEXT T
160    PRINT VL
180    POKE 36878, VL
200  NEXT VL
220  FOR VL = 15 TO 0 STEP -1
240    FOR T = 1 TO 100: NEXT T
260    PRINT VL
280    POKE 36878, VL
300  NEXT VL
```

Run this complete program. Now what happens to the volume? The volume increases and then it decreases. Change the program by adding the following program line. This will make the program run over and over again:

```
320  GOTO 120
```

Run the program. The volume should increase, decrease, increase, and so on until you stop the program. The time delay (lines 140 and 240) and printing steps (lines 160 and 260) can be removed from the program to increase its "speed." Remove lines 140, 160, 240, and 260 from your program. The program should now look like this:

```
100  POKE 36876,200
120  FOR VL = 0 TO 15
180    POKE 36878, VL
200  NEXT VL
220  FOR VL = 15 TO 0 STEP -1
280    POKE 36878, VL
300  NEXT VL
320  GOTO 120
```

Run the program, What do you hear? The sound seems to be pulsed, creating an interesting effect. You have not changed the tone. The program has only changed its volume.

Experiment No. 9-2. Changing the Tone

This experiment will show you how you can change the tone that is created by a "speaker" in the VIC 20. Clear your computer for a new program and load the following program into the computer:

```
100  POKE 36878,15
120  INPUT T
140  POKE 36874, T
160  GOTO 120
```

This program will let you type in values that will be poked into the sound map for SPKR1, which generates the lowest tones. Run the program and type in values between 128 and 250. You can adjust the volume of the TV set if you want to. What do you hear? The pitch of the tone increases as the value typed in is increased. Type in the value 128. This should give you a low rumble. Type in 250. This should give you a medium-pitch tone. Type in 254. This will give you a high-pitch whistle. If you can't hear this sound on your TV set, try the value 253. You should be able to hear its tone.

Type in the value 255. Did you expect a higher-pitch sound? We expected that 255 would create the highest tone for a "speaker." IT DIDN'T. We found that the tones created for codes 128 and 255 were exactly the same. THE HIGHEST TONE WILL BE CREATED BY CODE 254. AVOID CODE 255.

Type in values that are between 0 and 127. What tone is heard? You should find that for speaker codes less than 128, the tones are turned off and you can't hear anything from the TV set. While the volume of all the "speakers" together is controlled by the volume code, individual "speakers" can be turned off by giving them a code that is between 0 and 127. We suggest just using 0, since it is easy to remember.

Change program lines 120 and 160 to:

```
120  FOR T = 128 TO 254
160  NEXT T
```

The complete program will look like this:

```
100  POKE 36878, 15
120  FOR T = 128 TO 254
140  POKE 36874, T
160  NEXT T
```

Run the program. What do you hear? You should be able to hear the tones increase quickly from the lowest tone to the highest tone for SPKR1. Change the program by adding the following program lines:

```
180  FOR T = 254 TO 128 STEP -1
200    POKE 36874, T
220  NEXT T
240  GOTO 120
```

Here is a listing of the complete program so you can easily check it:

```
100  POKE 36878, 15
120  FOR T = 128 TO 254
140  POKE 36874, T
160  NEXT T
180  FOR T = 254 TO 128 STEP -1
200    POKE 36874, T
220  NEXT T
240  GOTO 120
```

Run this program. What do you hear now? The tones increase, decrease, increase, and so on until you stop the program.

In this experiment, you have changed the tone generated by one of the "speakers," SPKR1. This speaker generates the lowest set of tones available from the VIC 20 computer. Just for fun, change the program line as shown below. The other program lines should remain as they are:

```
120  FOR T = 220 TO 254

180  FOR T = 254 TO 200 STEP -1

240  POKE 36874,0
```

Add the following program lines to your program:

```
260  FOR D = 1 TO 500: NEXT D
280  GOTO 100
```

The complete program will now look like this:

```
100  POKE 36878, 15
120  FOR T = 220 TO 254
140  POKE 36874, T
```

```
160  NEXT T
180  FOR T = 254 TO 200 STEP -1
200    POKE 36874, T
220  NEXT T
240  POKE 36874,0
260  FOR D = 1 TO 500: NEXT D
280  GOTO 100
```

Run this program. If you use your imagination, you may be able to hear laser pistols firing, or perhaps you can think of something else that it sounds like. Do you hear laser pistols being fired? Maybe you hear something else.

Experiment No. 9-3. Changing the Speakers

In this experiment, you will see how you can switch from one "speaker" to another. You will also be able to hear the difference between each "speaker." Clear your computer for a new program. Type in the following program:

```
200  POKE 36878, 15
220  INPUT "TONE"; TN
240  FOR X = 0 TO 2
260    POKE 36874 + X, TN
280    INPUT Q
300    POKE 36874 + X, 0
320  NEXT X
340  GOTO 220
```

This program will let you type in a tone code. It will then use the tone code in SPKR1. When you press the RETURN key, the computer will go on to SPKR2, and so on, so that the same code is used in the three different "speakers."

Run the program and type a value between 128 and 254 when the computer prints "TONE ?". Press the RETURN key to get the computer to go to the next "speaker." You should be able to hear the difference between the "speakers" as the same tone code is used by each one in turn. Try several tone codes.

It would be interesting to hear the result of having the computer go through all of the tone codes by itself. To have it do this, change the following lines in your program:

```
220  FOR TN = 128 TO 254

340  NEXT TN
```

Remove line 280 from the program.

Now add the following program line:

```
360  GOTO 220
```

Here is the complete program. We have added extra spaces to indent the FOR-NEXT loops for clarity:

```
200  POKE 36878, 15
220  FOR TN = 128 TO 254
240      FOR X = 0 TO 2
260          POKE 36874 + X, TN
300          POKE 36874 + X, 0
320      NEXT X
340  NEXT TN
360  GOTO 220
```

Run the program. The tones will be fairly low at first, but you should be able to hear the effect of using the same code in each "speaker," one after the other. Now add a time delay loop to your program:

```
270  FOR M = 1 TO 20: NEXT M
```

You will be able to hear the effect of this simple program change. Are you ready for take-off? Run the program

This interesting sound "effect" was created by increasing the tone code and using each code in each of the "speakers." There is nothing very creative about this, but the result sounds interesting. Try changing the limits of the FOR-NEXT loop at line 220 so only the codes between 230 and 254 are used.

USING SOUND IN A PROGRAM

In the experiments, you saw how the computer can create different sounds, and how changes in the tones and in the volume can create interesting effects. These effects can be used in programs to add the sense of hearing to the experience of using a computer. In a typical program, you may be asked to type in a value that the computer will use in some way. If an incorrect value is typed in, a sound could be used to tell you that there is a problem.

In this program, you are asked to type in a value between 0 and 10000. If the value is outside those limits, the computer will ask for another value. Sound has been used to alert you to an incorrect value.

```
1000  INPUT "TYPE YOUR VALUE"; VA
1020  IF VA>=0 AND VA<=10000 THEN GOTO 1300
1040    POKE 36878, 15
1060    FOR Y=1 TO 15
1080      POKE 36876, 240
1100      FOR T = 1 TO 50: NEXT T
1120    POKE 36876, 0
1140    FOR T = 1 TO 20: NEXT T
1160  NEXT Y
1180  PRINT "ONLY VALUES FROM"
1200  PRINT "  0 TO 10000"
1220  GOTO 1000

1300  REM VALUE IS OK, SO USE IT. . .
```

The IF-THEN command at line 1020 checks the value to see if it is between 0 and 10000. If it is, the computer goes on to line 1300, where the value is used in some way by the program. However, if you have typed in a value that is outside the range 0 to 10000, the computer generates a "warbling" tone using program lines 1040 to 1160. After generating the sound, the computer prints a two-line message and you get another chance to type in a value. You can type in this program and try it if you want to.

Sometimes several sounds are generated in a program for different reasons. The warbling tone just described indicates an incorrect value has been entered. Other tones might tell you that an answer to a question is correct or incorrect. Sounds can also be used as "rewards" in teaching programs. Here are the input and sound parts of a "multiple-choice" test program that uses three sounds: a medium warbling tone when an incorrect value is typed in, a high warble for a correct answer, and a low tone for an incorrect answer. This is not a complete program, but it shows how tones can be used. Since the program is a bit complex, it has been broken into segments for you.

```
1000  INPUT "YOUR ANSWER 1-4"; AN
1020  AN = INT(AN)
1040  IF AN>=1 AND AN<=4 THEN 2040
```

```
1060  HT = 240 : LT = 0 : GOSUB 5010
1080  PRINT "ANSWERS FROM"
1100  PRINT "  1 TO 4 ONLY"
1120  GOTO 1000

2000  REM PROCESS THE ANSWER HERE
2020  REM CA = CORRECT ANSWER VALUE
2040  IF AN <> CA THEN 2200
2060  PRINT "GOOD FOR YOU"
2080  HT = 240 : LT = 235 : GOSUB 5010
2100  GOTO. . .

2200  PRINT "INCORRECT"
2220  HT = 140 : LT = 130 : GOSUB 5010
2240  GOTO. . .

5000  REM SOUND GENERATOR SUBROUTINE
5010  POKE 36878, 15
5020  FOR Y = 1 TO 5
5030    POKE 36876, HT
5040    FOR T = 1 TO 50: NEXT T
5050    POKE 36876, LT
5060    FOR T = 1 TO 20: NEXT T
5070  NEXT Y
5080  POKE 36876, 0
5090  RETURN
```

This program expects you to type in a value of 1, 2, 3, or 4 as an answer to a multiple-choice question. The computer inputs the value and then uses the INT operation to remove any decimal fraction. For example, 4.5 would be converted to 4. The program checks to see if the value is between 1 and 4, and if it isn't, a tone is generated and a message displayed. You get another chance to type in a value between 1 and 4.

The second part of the program checks to see if the answer, AN, matches the correct answer, CA. If there is a match, a high tone is generated and the computer goes on. If an incorrect answer is found, a low tone is generated and the computer goes on.

In this sample program, the tone program is used as a *subroutine*, and the main program simply points the computer to the subroutine to generate a tone. Each part of the program that uses this must supply a high tone (HT) and a low tone (LT) code. This simplifies the program quite a bit, and it means that by supplying different high and low tone codes, different effects can be gener-

ated with one subroutine. This example shows how sound can be added to a program to increase its value to the person using it.

This program has also shown a good use for a subroutine. The tone-creating instructions were used only once in the program, but three parts of the program can use these steps.

NOISE

The noise "speaker," SPKR4, is used to add noise, or "hiss" to a sound. This is most often used in games and recreational programs to create special "explosions," "blasts," and other sound effects. There are not many other uses for "noise."

Experiment No. 9-4. Making Noise

This experiment will give you a chance to see how "noise" can be used for a sound effect in a simple computer program. The "noise" sounds generated are really the mixture of a tone and some noise. Clear your computer and type in this program:

```
200 POKE 36878, 15
220 FOR TN = 128 TO 254
240   POKE 36877, TN
260 NEXT TN
```

This program will cause the computer to have SPKR4 generate each of its "noise" sounds, starting with the lowest one. Run the program. Were you able to hear the tone increase? Was the sound "noisy"? You should hear the tone increase, and it should sound a bit "fuzzy." Add the following time-delay program line to your program:

```
250 FOR M = 1 TO 100: NEXT M
```

Run the program. Can you hear the noise as the tone increases? You should be able to hear the noisy signal. You can press the RUN/STOP key to stop the computer before it reaches the end of the program so that a single "noise" can be heard. Run the program again and as the tone is increasing, press the RUN/STOP key. Can you hear the noise now? You can type CONT [RETURN] to have the computer continue with the program.

Here is a sound effect you can try. Change line 220 to:

```
220 FOR TN = 200 TO 252
```

and change line 250 to:

```
250  FOR M = 1 TO 10: NEXT M
```

Now add the following lines to your program:

```
280  FOR VL = 15 TO 0 STEP -1
300    POKE 36878, VL
320    FOR M = 1 TO 100: NEXT M
340  NEXT VL
```

The complete program is:

```
200  POKE 36878, 15
220  FOR TN = 200 TO 252
240    POKE 36877, TN
250    FOR M = 1 TO 10: NEXT M
260  NEXT TN
280  FOR VL = 15 TO 0 STEP -1
300    POKE 36878, VL
320    FOR M = 1 TO 100: NEXT M
340  NEXT VL
```

Run this program and listen to the effect. Remember that you're using *only* the noise "speaker" and the volume control in the VIC 20. Here is another program you can try. Clear your computer and type it in:

```
100  POKE 36878, 15
120  POKE 36877, 215
140  FOR T = 1 TO 150: NEXT T
160  FOR VL = 15 TO 0 STEP -1
180    POKE 36878, VL
200    FOR T = 1 TO 10: NEXT T
220  NEXT VL
```

Run the program and listen for the sound. You can add the following line to your program to get it to repeat the sound again and again:

```
240  GOTO 100
```

Most often, noise is used in creating special sound effects. It is not used too frequently in other types of programs. In any case, it is one of the sounds that can be generated by the VIC 20 computer.

In this chapter, you have learned how the VIC 20 computer's

"speakers" can be used to create, or "generate," interesting sounds. There are many interesting sounds that you can put together, and you can even turn the computer into a simple musical instrument. There are several interesting sound programs in the manual supplied with your computer.

QUESTIONS

1. How many sources of sound does the VIC 20 computer have?

2. What kinds of sounds can be created?

3. Can you control the volume of the sounds? If so, how do you do it?

4. How many different tones can each "speaker" create?

5. Is there any overlap between the tones generated by each "speaker"?

6. What is "noise"?

7. Can all of the sources of sound be used at the same time?

8. Can you suggest some uses for sound in computer programs?

PROBLEMS

1. Write a program that plays one of 10 possible tones when the keys 0 through 9 are pressed. You can choose any tones that you like, along with the volume value and the sound channel or "speaker" used.

2. Write a program that generates a "jet plane" (whoosh) sound.

3. Write a program that lets you enter numbers for one of the "speakers." Ordinarily, the numbers entered will be between 128 and 254. As these numbers are entered, they are stored in an array until the number 99 is entered, at which time the values entered will cause the appropriate sounds to be generated. Each tone should be played for the same amount of time.

4. Modify your solution to Problem 3, so that along with the values that are entered, a number that represents the *duration* of the tone can also be entered. Once the value 99 is entered, the "tune" is played.

5. Modify the solution to Problem 4 so that as keys are pressed, the appropriate tone is also generated. If the tone isn't what you want, you can enter other numbers. Only when the value 0 is entered is the tone value saved. The computer then lets you enter the duration of the tone, which is then played. You can continue to enter different "durations" until you get the proper sound, at which time a 0 is entered so that the duration is saved in the array.

6. Write a program that generates random tones on a sound channel, for random durations of up to one second per tone.

10

Accessories for the VIC 20

There are many useful accessories for your VIC 20 computer that can be added to it so it can do new and interesting things. These accessories range from plug-in cartridges to floppy-disk storage systems, and their prices range from about $25 to several hundred dollars. In this chapter, the plug-in cartridges, the cassette recorder, the graphic printer, and the joystick control will be discussed. You will be given some programs that you can use to control the cassette to save information on it, and you will see how you can use the joystick. Some printer control programs are also listed for you.

Several experiments are provided in the cassette section, and you can do these if you have a Commodore Cassette Unit and are interested in how the cassette works. If you are only interested in using the recorder to save and load *programs*, you can skip over the cassette recorder experiments.

PROGRAM AND MEMORY CARTRIDGES

Plug-in cartridges are used to preprogram the VIC 20 with ready-to-run programs for games, education, home finance, and other uses. The cartridges measure about 14 cm by 7 cm (5.5 in by 2.75 in) and they plug into the large open slot in the back of the

VIC 20 keyboard case. Typical cartridges are shown in Fig. 10-1, and one of the cartridge cases has been opened so you can see what is inside. The small rectangular device on the flat card is an electronic circuit that permanently stores the program instructions. A cartridge is shown inserted into the VIC 20 computer in Fig. 10-2.

Most of the programs that are available in cartridges will "take over" the computer so the special program starts as soon as the computer is turned on. Other plug-in cartridges, such as the Machine Language Monitor (VICMON), a tool for advanced programmers, are started by using a special command. Although cartridges will hold programs almost forever, they can be destroyed if you plug them in or take them out of the computer when the computer is turned on. Turn off the power before a cartridge is changed.

Many programs are available in cartridge form, so it is easy (but not necessarily inexpensive) to have many interesting programs that can be plugged in and used quickly. Most of the popular computer magazines contain advertisements for companies that produce cartridges for the VIC 20 computer.

The available storage space in the VIC 20 computer is about 3500 characters, and this is a bit small for some programs that are long, complex, or require extra space to store a lot of information. Special memory cartridges can be used to expand the "memory space" of the VIC 20 up to about 16,000 characters. This is generally enough for most uses. People who use computers a lot use the abbreviation "K" for one thousand, particularly when they are talking about the available "memory space" in a computer. So, 16,000

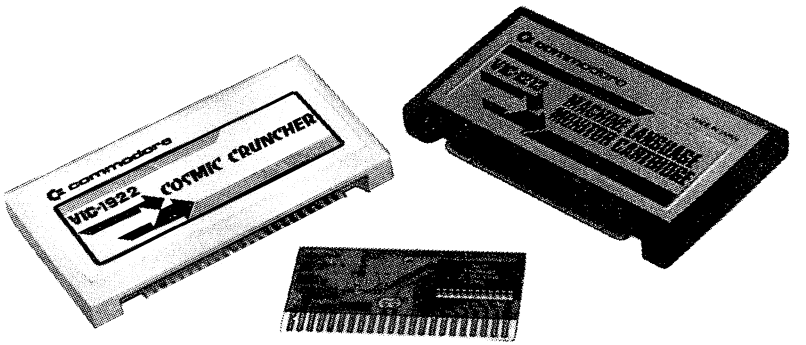


Fig. 10-1. Typical plug-in cartridges for the VIC 20.

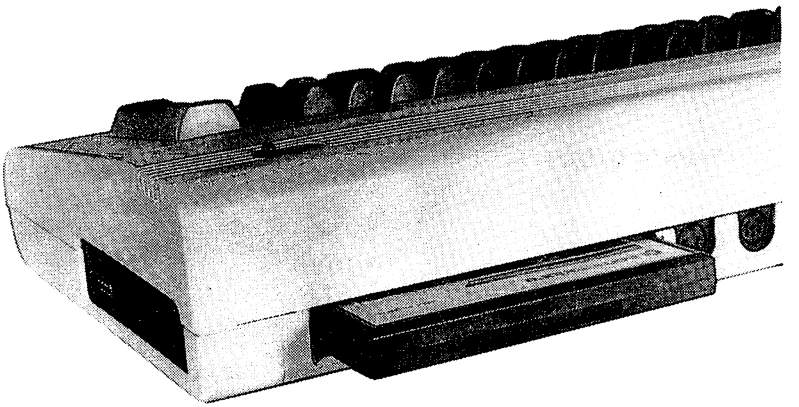


Fig. 10-2. A cartridge inserted into the VIC 20.

becomes 16K and it is pronounced, *sixteen kay*. (Actually, one "K" of memory contains 1024 characters, so 16K is 16,384 characters.)

"Memory cartridges" are available from many manufacturers, some of whom are listed at the end of this chapter.

THE CASSETTE UNIT

Besides plug-in cartridges, you will find that the cassette recorder is one of the most useful accessories for your computer. The cassette recorder is called the C2N Cassette Unit by Commodore, and it is easily connected to the computer, as shown in Fig. 10-3. The C2N cassette recorder was designed specifically to work with the VIC 20 computer, and a regular audio cassette recorder cannot be used instead without modifications or extra add-ons. We will call the C2N Cassette Unit the cassette recorder, or just recorder. (At the computer end of the connecting cable, you may find a long piece of silver, braided wire. This is not used in VIC 20 systems, and you can let it hang free, or you can tape it to the cable to keep it out of the way.)

The cassette recorder is easy to use and it will let you save both programs and information so that they can be put back into the computer at another time. Many commercial programs are available on cassette tapes instead of in cartridges, so you will need the recorder if you want to use these programs.

BASIC programs that you have written can be stored for later use by recording them on a cassette tape. This is easy to do, and

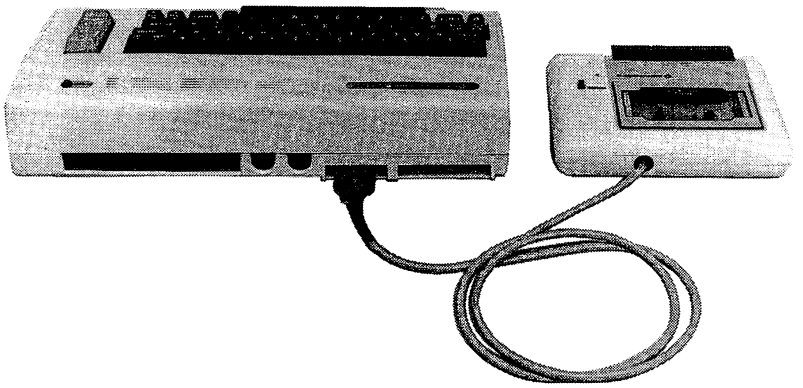


Fig. 10-3. The cassette recorder connected to the VIC 20.

you can store several programs on one cassette tape. Good audio-quality cassette tapes are recommended. Since it is unlikely that you will record an hour's worth of information, you can use the shorter audio cassettes, marked C-5 (5 minutes) or C-10 (10 minutes).

Instructions that tell you how to store a program on the cassette tape are contained in your computer instruction manual, and in the C2N Cassette Unit instruction manual. Three commands are used: SAVE, LOAD, and VERIFY. The VIC 20 saves your program twice on the tape, placing the second recording right after the first. In case of a problem with one, the second can be used as a backup.

After a program has been recorded, the VERIFY command can be used to check the tape to be sure that the program has been recorded correctly. We recommend verifying your programs, even though it takes a little extra time to rewind the tape and have the computer check it.

When programs are saved, verified, and loaded, you can give them a name so that they can be easily identified by the computer. If you are putting only one program on each tape, you don't need to do this, but be sure you carefully label each cassette. In any case, putting a name on the tape right along with the program makes the program easy to identify and locate. The name may have up to 16 characters, and any of the VIC 20's characters, numbers, letters, or symbols may be used. Unlike labels used in your programs, the VIC 20 uses *all* 16 characters in a name to locate a program on a tape.

SAVING INFORMATION ON A CASSETTE

The SAVE, LOAD, and VERIFY commands are used only to save, load, and verify *programs*. If you want to use the cassette recorder to save information, such as values and strings, you will have to use two new commands, OPEN and CLOSE. The recorder can be thought of as a drawer that will hold your information. Before you can use it, it must be opened, and when you are finished, it must be closed. The OPEN and CLOSE commands are used to control some of the other VIC 20 accessories, and each accessory has been given a number so the computer knows which one is being used. The computer has preset the cassette recorder to be device #1. The computer uses device code #1 for its own purposes, and you don't have to be concerned with this number.

When the recorder is to be used, the computer must be told whether you are going to put information on the tape; that is, "write" information, or whether you are going to get information from the tape; that is, "read" information. The *read* and *write* operations each have their own code: 0 for read and 1 for write.

Information is stored on the tape as a "file," and you can give each of your information files a "file name" so they are easy to locate. This means that several files can be saved on the same cassette tape. The command to "open" the cassette recorder looks like this:

```
OPEN your id number, 1, read/write code, "file name"
```

Once a device, such as the cassette recorder, has been "opened," you can refer to it by using the identification (id) number *you gave it* in the OPEN command. This makes it easy to point the computer to the cassette recorder when you want information transferred back and forth. Here is a typical example of how the OPEN command is used:

```
OPEN 9,1,1,"BANK"
```

This command tells the computer to "open" the cassette recorder so that a file named "BANK" can be recorded on a tape. The cassette recorder is identified as #9. This number was chosen at random, and another number would do just as well, for example:

```
OPEN 12,1,1,"BANK"
```

Once information has been placed on a tape, the cassette recorder must be "closed" until you want to use it again. The CLOSE

command is simple; CLOSE9 for the first example and CLOSE12 for the second one.

Getting information to and from a tape is very easy. The PRINT and INPUT commands are used, just as they were when information was displayed on the TV screen and input from the keyboard. The only difference is that you identify the recorder by using the identification number you gave it in the OPEN command. When the INPUT and PRINT commands are used with the keyboard and the screen, identification numbers are not used. Here is an example in which 100 data values are saved on a cassette tape with the file name "NUMBERS." The recorder has been given the identification number, 9:

```
120  OPEN 9,1,1,"NUMBERS"
140  FOR NB = 1 TO 100
160    PRINT NB
180    PRINT#9, NB
200  NEXT NB
220  CLOSE 9
```

The OPEN command tells the computer to set up the cassette recorder so that the file called "NUMBERS" can be *recorded* on the tape. It also tells the computer that *you* are going to call the recorder "#9" later on in your program. The PRINT#9, NB command tells the computer to "print" the value labeled NB on the tape. Once all 100 values have been recorded, the CLOSE 9 command tells the computer that you are finished with the cassette recorder.

It is your responsibility to place a cassette tape in the recorder, rewind the tape to the start, and follow the directions that the computer automatically displays on the TV screen. We recommend that you practice recording and loading some simple programs using the SAVE, VERIFY, and LOAD commands before you try using the recorder to save other types of information.

Most audio-quality cassette tapes have a clear or colored plastic "leader" at the start of the tape. If you rewind a tape, you can see this leader in the opening in the side of the plastic case. The computer will generally go past the leader before it starts to record information. However, if you have difficulty getting information stored on the tape, we recommend rewinding the tape and resetting the counter wheels on the cassette recorder (press the small black button). This will put the count to 000. Now "play" the cassette until the count reaches the 010 mark. This should advance

the tape past any plastic leader so the tape is ready to record information.

Getting information back off a tape is not difficult. The difference is that the recorder must be "opened" so that the information can be read, and an INPUT command must be used to read the information from the tape:

```
500 OPEN 9,1,0,"NUMBERS"
520 INPUT#9, DV
540 PRINT DV
560 GOTO 520
```

The read/write code in the OPEN command is now a 0, since this program *reads* information from the cassette recorder into the computer. This particular program will read information from the tape, *but it will never stop.* There are no instructions in the program or on the tape that tell the computer there is no more information on the tape.

When information is read from the tape into the computer, the computer must have a way of knowing that all of the information has been received. In the preceding example, only 100 values were saved, so a FOR-NEXT loop could be used to count the values as they are received. Here is a program that shows how this can be done:

```
500 OPEN 9,1,0,"NUMBERS"
520 FOR NV = 1 TO 100
540   INPUT #9, DV
560   PRINT DV
580 NEXT NV
600 CLOSE 9
```

Now the computer searches the tape for the file called "NUMBERS" and uses a loop to count off the first 100 values from it. After these values have been received, the CLOSE command turns off the recorder and "closes" it so it can be used later. The information from the tape doesn't have to be displayed. It can be placed in an array, or used in some other way.

Experiment No. 10-1. Saving Information

In this experiment you will find out how you can save information on a cassette tape so that it can be used later by the computer. You will need a C2N Cassette Unit (recorder) and a blank audio cassette tape in this and the following cassette experiments.

Turn off your computer and be sure that the cassette unit is properly connected to the VIC 20. If it is not connected, refer to the operating instructions that came with the cassette unit. Once the recorder has been connected and checked, continue to the next step.

There is an opening on the cassette tape and you can see the narrow, brown tape or plastic leader pass through it. On the opposite edge of the cassette, you will see two square areas near the corners. To use the cassette in this experiment, there must be plastic tabs in these square areas. If either of the plastic tabs is missing, leaving a square hole, another cassette must be used. A broken tab will prevent you from recording information on the tape.

Turn on your computer, place the cassette tape in the recorder, and rewind it by pressing the REWIND button. After the tape has been rewound, press the STOP button. Clear your computer for a new program and type in the following program:

```
100  OPEN 7,1,1,"DATA"
120  FOR DV = 0 TO 100
140    PRINT DV
160    PRINT#7, DV
180  NEXT DV
200  CLOSE 7
```

Run the program. The computer will display a message on the TV screen telling you to press the RECORD and PLAY buttons on the recorder. Press these buttons *at the same time.* They should both stay down. What does the computer do? A message, "OK," is displayed on the TV screen and the recorder starts running. It runs for several seconds and then stops. After it stops, the numbers start to be displayed on the TV screen. Do the numbers immediately go up to 100? Eventually they do, but they go up to about 40 first and then the recorder runs for a while. When the recorder stops, the numbers go up to about 78 and the recorder starts again. When the recorder stops, the numbers go up to 100 and the recorder starts again. When the recorder finally stops, the READY message is displayed on the TV screen.

When the READY message is displayed, press the STOP button on the recorder and rewind the cassette tape. You have saved the numbers from 0 to 100 on the tape. In the next experiment you will find out how you can read the numbers back into the computer.

In this experiment, you found that when the recorder is

"opened" so that you can save information on a tape, the recorder immediately starts running. The computer runs the tape forward a bit to put some space between the last file that was recorded, if any, or to get past any plastic leader at the start of the tape. The computer also puts the name of the data file on the tape so it can be identified later when you want to read it.

The computer did not wait for all of the information to be displayed before it started to record the information. The computer works much the way we do when we use a clothes washer. We don't wash each piece of clothing, one at a time, we wait until we have a full load and then run the washing machine. The computer does not record each value, one at a time. Only when the computer has enough information will it send a complete "block" to the recorder. This saves time and space on the tape.

Once the computer has accumulated a complete "block" of information, it automatically records the information on the tape. If the computer doesn't have a full "block" of information to be recorded, it will not send the information to the recorder until it reaches a CLOSE command that "closes" the recorder. This is why the CLOSE command is so important. It causes the computer to save any last pieces of information that don't make up a complete "block."

Experiment No. 10-2. Getting Data from the Cassette

In this experiment, you will see how data stored on a cassette tape can be read into the computer. Clear the computer for a new program. The cassette tape prepared in the previous experiment will be used in this experiment. If you don't have a tape with that information on it, go back and do Experiment No. 10-1. Load the following program into your computer:

```
300  OPEN 8,1,0,"DATA"
320  FOR L = 0 TO 100
340    INPUT#8, DV
360    PRINT DV
380  NEXT L
400  CLOSE 8
```

Be sure that your tape has been rewound, then run this program (RUN 300 or GOTO 300). The TV screen will display a message telling you to press the PLAY button on the cassette recorder. Press *only* the PLAY button! It should stay down. The OK mes-

sage is displayed on the TV screen, the recorder starts to operate, and the computer is now searching for the file named "DATA." When the computer has located DATA, you will see the recorder start and stop as "blocks" of information are read into the computer and displayed on the TV screen. Change the following line in your program:

```
320  FOR L = 200 TO 300
```

Rewind the tape and run the program again. The same display of numbers from 0 to 100 should be seen on the TV screen. The loop count values were changed just to show you that these values and the values stored on the tape are not related. Change line 300 to:

```
300  OPEN 8,1,0,"NMBR"
```

Rewind the tape and run the program. What happens? When the PLAY button is pressed, the OK message appears on the TV screen and the recorder starts to operate. The recorder continues running, but no values are displayed.

Stop the recorder and rewind the tape. Do you know why no values were displayed? The information was recorded on the tape as a file called "DATA." The computer is told to "open" the recorder for a read operation, and to look for a file called "NMBR." The computer comes across the DATA file, but it cannot find NMBR. The computer will continue searching for NMBR right up to the end of the tape! Change line 300 back to:

```
300  OPEN 8,1,0,"DATA"
```

and change line 320 to:

```
320  FOR L = 0 TO 50
```

Now the program looks like this:

```
300  OPEN 8,1,0,"DATA"
320  FOR L = 0 TO 50
340    INPUT#8, DV
360    PRINT DV
380  NEXT L
400  CLOSE 8
```

Rewind the tape and run this program. Follow the directions for the cassette recorder displayed on the TV screen. What values are eventually displayed? Only the values from 0 to 50 are displayed. Even though there is more information on the tape, you can take only a small part of it if that's all you want.

In this experiment you saw how the computer could read information from a cassette tape. The file name used in the OPEN command must match a file name used to record information on the tape. A loop was used to control the amount of information taken from the tape.

In some cases, you may not know ahead of time how much information is to be stored in a file, so some flexible and adaptable method of reading and writing must be used. A special value or character can be put at the end of a file so that when the computer reads the character, it knows that the end of the file has been found. Here is a sample program that gets strings of characters from the keyboard and puts them on the tape in a file called "STRING." When the computer gets a one-character string, *, from the keyboard, the computer stops. The asterisk is the last character in the file named "STRING":

```
200  OPEN 9,1,1,"STRING"
220  INPUT W$
240  PRINT#9, W$
260  IF W$ <> "*" THEN GOTO 220
280  CLOSE 9
```

Here is how the strings can be read back into the computer:

```
500  OPEN 9,1,0,"STRING"
520  INPUT#9, W$
540  PRINT W$
560  IF W$ <> "*" THEN GOTO 520
580  CLOSE 9
```

In this program, the computer tests each string to see if the string is a single asterisk. If it is, the end of the file has been found, and there is no more information to get from the tape. The program "closes" the recorder as the last operation, so the recorder can be used somewhere else in the program. A "marker" is used to end a cassette tape file when you don't know how much information is to be saved.

Experiment No. 10-3. Saving Strings on a Tape

Saving strings of different lengths on a cassette tape isn't difficult. You will see how it is done in this experiment. You will also see how a "marker" is used to end a file. Clear your computer for a new experiment and rewind the tape in the cassette recorder.

Type the following program into your computer:

```
200  OPEN 9,1,1,"STRING"
220  INPUT CS$
240  PRINT#9, CS$
260  IF CS$ <> "*" THEN GOTO 220
280  CLOSE 9
```

This program will let you type in a string, which will then be saved on the tape. If you type an asterisk, *, it will be saved on the tape, too, but it will also cause the computer to go to line 280. The CLOSE 9 command "closes" the recorder and records any remaining information that doesn't make up a complete "block."

Start the program and follow the instructions on the TV screen for the recorder. The recorder will run for several seconds, and then a question mark will be displayed on the TV screen. When you see the question mark, type in your strings. Remember to end each string by pressing the RETURN key. Type in several strings, such as:

```
? THIS IS AN EXPERIMENT
? MY NAME IS FRED
? MY ADDRESS IS GREENLAWN
? WHAT TIME IS IT?
```

When the next question mark appears on the TV screen, type * [RETURN]. What happens? The recorder will probably run for a while, and then the READY message will appear on the TV screen. When the READY message appears, rewind the tape.

Clear your computer and type in the following program; it will be used to read the strings from your tape:

```
500  OPEN 9,1,0,"STRING"
520  INPUT#9, RS$
540  PRINT RS$
560  GOTO 520
```

Rewind the tape, then run the program and follow the instructions displayed on the TV screen. After pressing the recorder controls and giving the computer a chance to get the information from the tape, what is displayed on the TV screen? You should see your original strings displayed on the TV screen. You *may* see other information, too, or you may get an *error message*, such as:

```
?STRING TOO LONG
 ERROR IN 520
```

Why do you think the other information or the error message was displayed? There may be other information left on the tape from old programs, or data that were saved sometime earlier. The program does not have any way of knowing where to stop the tape, so it keeps the recorder running, reading this "unknown" information from the tape. To stop this from happening, the program needs to be changed to look for the asterisk. Here are the program lines to *add* to your program:

```
560 IF RS$<>"*" THEN GOTO 520
580 CLOSE 9
```

Rewind the tape and run the program again. What do you see on the TV screen when the recorder finally stops? The original strings are seen. No other information or error message is seen. When the computer saw that an asterisk had been received from the tape, the program "closed" the recorder and stopped it. Save your tape with the STRING file on it for use in the next experiment.

THE CASSETTE AND THE GET COMMAND

The GET command is used to get individual characters from the keyboard, and the GET command can do the same thing with a cassette tape. (You may remember that the GET command gets a "null" or "nothing" character if a key isn't pressed when the computer "looks" at the keyboard.) When used with the recorder, the GET command gets characters from the tape, one at a time. There are no "null" characters between the characters on the tape, or anywhere else in the information. Since the GET command gets *single characters* from the tape and the keyboard, strings and values must be put back together from the individual characters. Short BASIC programs can be used to do this.

In most cases, the INPUT command will be used to control the recorder, since this command gets a complete string or a complete value from the tape. When the INPUT command is used, no special operations are needed to put the information back in a meaningful form.

It isn't difficult to end a file with a "marker," and this can be done with a few extra software steps. However, if you are using a tape that contains information from someone else's VIC 20 computer, you may find that they didn't put a "marker" at the end of their data files. Don't worry, since the VIC 20 automatically puts a null character at the end of each data file. The null character can be

looked for as a file is read from the tape using a GET command. Here is how it can be done for strings:

```
700  OPEN 9,1,0,"STRING"
720  GET#9, R$
740  IF R$="" THEN GOTO 800
760  PRINT R$
780  GOTO 720
800  CLOSE 9
820  etc . . .
```

This program gets individual characters from the tape and checks each one to see if it is a null. If it isn't a null, the program displays the character and gets the next one. If a null is found, that's the end of the file, so the recorder is "closed" and the program ends. The value of the GET command is in getting information from a tape when you don't know how much information there is in a file.

When this technique is used with *values* that have been stored on a tape, the program is more complex, since individual digits must be put back into strings, and the strings converted to the values originally stored. Here is how it is done:

```
1000  OPEN 7,1,0,"STRING" : B$ = ""
1020  GET#7, H$
1040  IF H$ = CHR$(13) THEN GOTO 1140
1060  IF H$ = "" THEN : CLOSE 7 : GOTO 1200
1080  B$ = B$ + H$
1100  GOTO 1020
1120  REM GET THE VALUE OF THE STRING
1140  X = VAL(B$) : B$ = ""
1160  PRINT X
1180  GOTO 1020
1200  etc . . .
```

This program first "opens" the cassette and then "clears" the string labeled B$. The computer checks for two conditions. If a null character is received from the tape, the computer stops the recorder and goes to the program steps starting at line 1200. When the computer puts values or strings on a tape with a PRINT#-type command, it places a RETURN character between the values or strings to separate them. The return character is CHR$(13), and if it is found, it means that the computer has reached a "separator" between two values or strings that have been put on the tape. The computer then converts the *string of individual digits* into a real

value, X. This value is displayed, but something else could have been done with it, too. Perhaps it could be stored in an array for later use. The B$ = "" operation clears the string B$ so it starts "empty" for the next string that is to be converted to a value.

Experiment No. 10-4. Using the GET Command with the Recorder

This experiment will demonstrate how the GET command can get individual characters from a tape. The cassette tape with the STRING file from the previous experiment will be used. If you do not have a cassette tape with that file on it, go back and do the previous experiment. Clear your computer for a new program and type in the following program:

```
400 OPEN 9,1,0,"STRING"
420 GET#9, H$
440 PRINT H$
460 GOTO 420
```

Rewind the tape and run the program. Follow the directions displayed on the screen for the recorder. You can use the CTRL key to slow down the speed of the display. What is displayed? The screen displays the original strings, one character at a time, with the characters going down the left side of the TV screen. Does the program or the recorder stop? The recorder continues to run, and no READY message is seen on the TV screen. If your tape was used previously to store other information, you may see other characters displayed on the TV screen, too.

The GET command gets individual characters from the tape. *It does not put the characters together to form the complete strings that you originally put on the tape.* Here is a new program you can use to link the individual characters back into the original strings:

```
400 OPEN 9,1,0,"STRING"
420 GET #9, H$
440 IF H$ = CHR$(13) THEN GOTO 600
460 IF H$ = "" THEN GOTO 700
480 TS$ = TS$ + H$
500 GOTO 420

600 PRINT TS$ : TS$ = ""
620 GOTO 420

700 CLOSE 9
720 END
```

This program gets the individual characters from the tape. Characters are linked at line 480 into a total string, TS$. If a RETURN character, CHR$(13), is found, that means that the end of a string has been found, and the complete string can be displayed (line 600). The TS$ string is reset to "nothing" to give it a fresh start for the next string.

If a null character is found, that means that the end of the file has been found and the recorder is "closed."

Rewind your tape and run the program. You should see the complete strings displayed on the TV screen.

A program to do basically the same thing with values stored on a tape has been provided in this section. The only extra operation needed for values is to convert from a string of digits to a real value that the computer can use.

The recorder will not be used again in this chapter, so you can remove the tape and disconnect the recorder, if you want to, but *turn off the power first.*

Some of this programming is a bit advanced, but it is useful. For example, if you are trying to keep track of household expenses for the last 12 months, it would be a lot of work to type in many of the same values every month. By using the recorder, you can save the values for the past 12 months on a tape. You can also save your household calculation program, so everything is on one cassette tape. Student grades, sales reports, car expenses, patient information, and other things can be stored on a tape using the types of programs shown in this section.

THE GRAPHIC PRINTER

The Commodore VIC-1525 Graphic Printer is a useful accessory for your VIC 20 computer, since the printer gives you the opportunity to actually print information, program listings, and graphics on a sheet of paper. The user's manual supplied with the printer is fairly complete, so we won't go into details about connecting the printer to the computer or setting it up. We will give you some information about how the computer and printer are used together.

The printer is a dot-matrix printer that uses small needles to form the individual characters. The dots are used in a 6-column, 7-row format, as shown in Fig. 10-4 for the letter "R." The printer can print 80 characters in a line, which is fairly standard among computer printers. The printer can duplicate all of the characters,

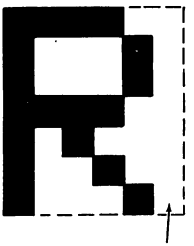


Fig. 10-4. A 6 by 7 dot matrix for the letter "R."

symbols, and graphic characters that can be displayed on the TV screen, and characters can be printed in "reverse," too. Of course, the printer can't print in color.

STARTING THE PRINTER

The printer is just like the cassette recorder in that it must be "opened" and "closed" with the OPEN and CLOSE commands. The VIC 20 computer can use up to two printers, and device codes 4 and 5 have been set aside for them. If only one printer is to be used, code 4 should be used, and this is set on a small switch on the back of the printer where the computer-printer cable is connected. The OPEN command for the printer looks like this:

```
OPEN your id number, 4, character set code
```

The identification (id) number you assign to the printer can be anything you choose, as long as it hasn't already been used to identify some other device, such as the cassette recorder. The character set code is also a *secondary address*, and it lets you select the set of characters that will be displayed on the printer. Commodore calls these two sets of characters the "CURSOR UP" and "CURSOR DOWN" sets. The CURSOR UP set is the one normally used on the TV screen (Set 1), and it has a code of 0. The CURSOR DOWN set has a code of 7, and it corresponds to Set 2 for the TV screen display. Here are examples of the OPEN commands for both character sets:

```
OPEN 33, 4, 0   (id #33, Set 1 characters)

OPEN 52, 4, 7   (id #52, Set 2 characters)
```

The identification numbers, 33 and 52, have been chosen at random.

The printer can be used to get a printed result from a program, and this is often called "hard copy," which just means that you can hold onto the copy and save it for later use. The PRINT# command is used to control the printer, and here is an example of a simple program that prints 10 numbers on a line:

```
100  OPEN 46, 4, 0
120  FOR T = 1 TO 10
140    PRINT#46, T;
160  NEXT T
180  CLOSE46
```

The semicolon supresses the printer's line feed, so that all of the numbers are printed on one line. When this program is run, *the numbers are not printed on the TV screen.* They are only printed on the paper in the printer. Since it is often better to test a program using the TV screen (it's faster and doesn't use paper), it will be a bother to try and convert all of the PRINT commands to PRINT# commands when you want to switch over to the printer for a "hard copy" of the program results.

The CMD command lets you switch *all* of the printing operations over to the printer, and only a single CMD command is needed, right at the start of the program. For example, the following program can be used to print a message on the TV screen:

```
200  PRINT  "**************************"
220  PRINT  "*                        *"
240  PRINT  "*     TEST MESSAGE       *"
260  PRINT  "*                        *"
280  PRINT  "**************************"
```

Two commands can be added to this program so that the message is printed on the printer, instead:

```
160  OPEN 43, 4, 0
180  CMD43
200  PRINT  "**************************"
220  PRINT  "*                        *"
240  PRINT  "*     TEST MESSAGE       *"
260  PRINT  "*                        *"
280  PRINT  "**************************"
```

The CMD43 command switches all of the PRINT commands to print their information on the device we've called 43; the graphic printer. Since the printer can print 80 characters on a line, long printing lines such as those shown here will "wrap around" on the TV screen when the program is listed.

When the CMD command is used, *all* of the printing is switched over to the printer, so even the READY message will be printed on the paper. To switch the computer back to the TV screen, you simply use a PRINT#43 command at the end of your programs. (We assumed that your id #43 is still being used, as in the preceding examples.)

Here is another example of how this can work when a line of letters, A–Z, is printed on the printer:

```
500  OPEN 43,4,0
520  CMD43
540  FOR LT = 65 TO 90
560    PRINT CHR$(LT);
580  NEXT LT
600  PRINT#43
620  CLOSE43
```

The OPEN and CMD commands set up the printer and switch printing over to the graphic printer. The letters are printed on a line, and the PRINT#43 command is a "dummy" command that switches back to the TV display for normal computer use. The CLOSE command simply "closes" the printer, since it won't be used again in the program.

The CMD command is useful when you want to print *everything* on the printer. If you want to print some information on the printer and some on the TV screen, you will need to use the PRINT and PRINT# commands individually.

In the preceding printing program, the CHR$ command was used to "convert" a value into its equivalent string character, 65 = A, 66 = B, and so on. The VIC-1525 printer uses some special string characters to set it up for special printing operations. The codes are listed in Table 10-1.

For example, the CHR$(14) "character" sets the printer so that it prints double-wide characters. The double-wide characters are useful for labeling tables, charts, and figures, and for chapter headings in manuscripts and reports.

There are also CHR$ "characters" that can be used to switch to and from the CURSOR UP and CURSOR DOWN sets of char-

Table 10-1. Special Printer Control Codes for the VIC-1525 Graphic Printer

Code	Description
CHR$(8)	Enter graphics mode[1]
CHR$(10)	Line feed after printing
CHR$(13)	Carriage return[2]
CHR$(14)	Start double width
CHR$(15)	Start standard width
CHR$(16)	Tab setting[1]
CHR$(17)	Cursor Down (Set 2) characters
CHR$(18)	Start reverse printing
CHR$(26)	Repeat graphics[1]
CHR$(27)	Specify dot address[1]
CHR$(145)	Cursur UP (Set 1) characters
CHR$(146)	End reverse printing

[1]These operations require additional commands or information.
[2]Carriage-return returns printing head to the start of the same line, without advancing the paper.

acters, set printing positions, and display graphic information on the paper.

You can list your programs on the printer once it has been opened. The following can be typed in to do it:

```
CMD43 : LIST [RETURN]
```

This tells the computer to send the listing to the printer. However, when the listing has been completely printed, the READY message will be printed on the printer and not on the TV screen. To switch back to the TV, just type:

```
PRINT#43 [RETURN]
```

and the computer will again use the TV screen.

The use of the printer to print the standard graphic symbols found on the keyboard is not difficult, and these characters can be put right in PRINT and PRINT# commands. The printer can also be used to print complicated graphs and special characters that can be set up in programs.

Although there are several examples in the printer user's manual, we don't recommend that you try and use or understand these programs, since they are *quite complex*, and it won't be obvious from the program listings what is being done. The special graphics modes for the printer aren't the type of thing that we'd recommend to beginners who are just becoming familiar with BASIC programs and the BASIC commands.

THE JOYSTICK CONTROLS

The joystick is another useful accessory for your VIC 20, since it is used in many games to move a "player" or other object on the TV screen. A typical joystick is shown connected to a VIC 20 computer in Fig. 10-5. The connector on the VIC 20 is marked "CONTROL PORT." The joystick unit contains four switches that are attached to the "stick," and as the stick is moved, the switches are actuated. The four switches are located in the NORTH, EAST, SOUTH, and WEST positions, as shown in Fig. 10-6. The joystick can be placed in between adjacent positions, actuating two switches, so the NORTHWEST, NORTHEAST, SOUTHWEST, and SOUTHEAST positions are possible, too.

Most joysticks have a "fire" or control button that is used to tell the computer to shoot missiles, launch rockets, and cause other things to happen. This control button is a separate switch that is independent of the joystick, and it can be pressed at any time.

Although joysticks are most frequently used with games, they have other uses, too. For example, the joystick might be used to point to answers in a multiple-choice test. Instead of numbering the possible answers, they are places in NORTH, SOUTH, EAST, and WEST positions on the TV screen. Moving the joystick moves a colored square to the selected answer, and pressing the control button enters the answer into the computer. People who have limited physical movement can control many things with a simple joystick, having the computer make decisions based on the joystick's position. The joystick can be used in more ways than as a simple game control.

The five joystick switches are electrically connected to the computer, and they can be "looked at" by using a BASIC program. Like the display map, the joystick has its own set of "addresses," so the position switches and the control button can be checked. Here is a program that can be used to give the positions of the joystick and the control button, as values:

```
1000  REM SET UP JOYSTICK MAP
1020  PA = 37137 : PB = 37152
1040  POKE 37139, 0
1060  POKE 37154, 127
1080  REM GET POSITION VALUES
1100  TH = PEEK(PB) AND 128
1120  TL = PEEK(PA) AND 28
1140  JS = TH/16 + TL/4
```



Fig. 10-5. A joystick connected to the CONTROL PORT on the VIC 20.

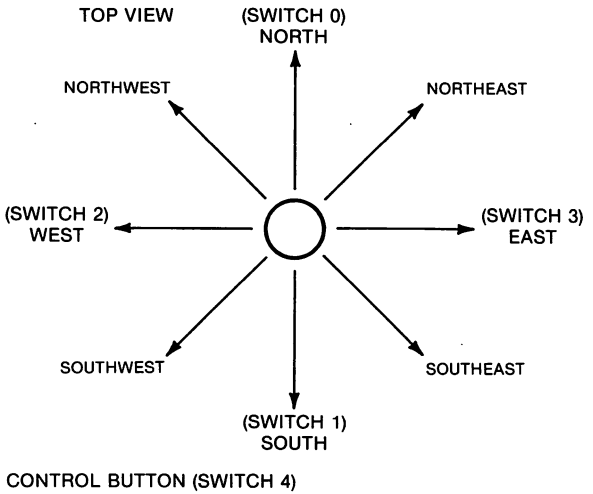


Fig. 10-6. Joystick positions can be related to compass directions.

```
1160  FB = (PEEK(PA) AND 32)/32
1180  PRINT JS, FB
1200  GET A$ : IF A$ = "S" THEN POKE 37154, 255 : END
1220  GOTO 1100
```

This program will display values that correspond to the positions shown in Fig. 10-7. These values can be used in IF-THEN or ON

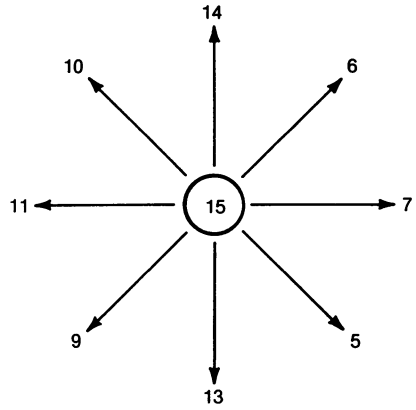


Fig. 10-7. Joystick positions and values for the BASIC control program.

commands to tell the computer what to do with the different joystick positions (JS). The control button value (FB) will be 0 when the button is pushed, 1 when it isn't. This progam displays the values of JS and FB on the TV screen. (NOTE: If you decide to use a program such as this to determine the position of the joystick and the conrol button, *you must clear the joystick map with the POKE 37154, 255 operation,* as shown at line 1200, when your program finishes using the joystick. In the example program, simply press the "S" key to stop the program and clear the joystick map.) If you forget to do this, the keyboard won't work properly, and you'll have to press the RUN/STOP and RESTORE keys at the same time to reset it.

You may clear your computer for a new program and type in the joystick program if you want to see how the joystick works. Plug your joystick into the CONTROL PORT connector on the right side of the computer's keyboard case, as shown earlier in Fig. 10-5.

There are other ways to figure out the position of the joystick, and one is listed in the *VIC 20 Programmer's Reference Guide,* along with more information about exactly how the joystick works.

SUPPLIERS AND DISTRIBUTORS OF PLUG-IN CARTRIDGES AND PROGRAM CASSETTES FOR THE VIC 20 COMPUTER

Academy Software
P. O. Box 9403
San Rafael, CA 94912
415-499-0850

Human Engineered Software
71 Park Lane
Brisbane, CA 94005

Microphys Programs
1737 West Second Street
Brooklyn, NY 11223

Micro Systems Development, Inc.
11105 Shady Trail, Suite 104
Dallas, TX 75229
1-800-527-5285

Micro-Ware Distributing, Inc.
1342 B Route 23
Butler, NJ 07405
201-838-9027

Precision Technology, Inc.
P. O. Box 15454
Salt Lake City, UT 84115
801-487-6266

Pyramid Computerware
278 Warren Street
Edgewater Park, NJ 08010
609-386-9353

Skyles Electric Works
231G South Whisman Road
Mountain View, CA 94041
415-965-1735

11

Computer Care—Hardware and Software

By taking good care of your computer, you can extend its useful life by many years, and it will reward you with good service and operation. Most "computer care" is common sense, but most *computer users* ignore this important area, and they are more likely to take better care of garden tools and household appliances. Taking care of your computer also means a careful evaluation of software for it. In the last part of this chapter, we will suggest some ways to evaluate software products to be sure you get what you want. Some of this evaluation is common sense, too, and other people may have their own ways of testing and checking software.

HARDWARE CARE

One of the first things to do is to protect the computer from physical abuse. If youngsters are going to use the computer to play games, for educational programs, or something else, they need to be told that the computer is a delicate piece of electronic equipment, and that it must be taken care of. Most people need to be reminded of this from time to time, since as they become more and more familiar with the computer, they will take it for granted. Be sure that users don't bang on the keys or hit the computer if something goes wrong. As silly as it sounds, it does happen.

A Special Place

A good way to protect your computer is to set aside a special place for it. If you are using the computer on the kitchen table, there is always the possibility of dropping or damaging the computer when it is put away during meals or other activities. On the other hand, if a desk or small table can be set aside for your "computer center," this will decrease the possibility of damage to the computer since it won't be moved as much.

Once your computer has been set up in a special place, you'll have a convenient place for all of the other computer materials that you're bound to collect; books, magazines, program cartridges, cassettes, and so on. A neat, logical layout is helpful, and storage bins or small sets of drawers are recommended as a way of organizing your materials. A disorganized work area means that program listings, cassettes, and special notes can be lost or damaged under the "junk pile."

Cassettes and disks store information and programs using small magnetic material. If you don't take good care of the cassettes and disks, the information on them can mysteriously disappear. Don't leave cassettes and disks near electrical equipment, and don't put them on top of the computer or TV set.

Most cassettes come in their own plastic cases. Be sure that the cassettes are returned to the cases when you are finished using them. Don't leave a cassette in the recorder for any longer than is necessary. Disks and cassettes can be further protected by using some sort of a "file" for them. Many record stores sell cassette organizers, and most computer stores have similar storage cases for disks. If you value your programs and information, take care of your cassettes and disks.

When you set up your computer equipment, place the cables and wires neatly. You can coil unused lengths of wire and cable, securing them with plastic bag ties or rubber bands. Keeping the wires and cables behind the computer equipment decreases the chances that something will get tangled in them and that someone will mistakenly unplug them.

The VIC 20 uses an antenna switch to connect the TV antenna (or cable) or the computer's TV cable to the TV set. If the TV set you are using is going to be used by others for normal TV watching, remember to switch back to the "TV" or the "ANTENNA" position when you are finished using the computer. This will avoid someone looking for you when the midnight late show comes on and they find, ". . . the TV isn't working right . . ."

Static

If you find that you are getting shocks from static electricity every time you move across the room in which you have the computer, it's a good idea to do something about it to protect your computer equipment. You might first try changing your shoes, since the material on the soles may be the culprit. If that doesn't reduce the static electricity, try using another chair, since plastic or plastic-covered chairs often generate static electricity. You can also use a humidifier in the computer room. Static electricity is reduced when the humidity is increased, so static is most often a problem during dry weather (usually during winter). Your nasal passages and throat will also thank you.

Protection

There are some other steps you can take to keep your computer running smoothly. One is the use of dust covers for the equipment. Plastic, nonstatic dust covers are recommended for all of your computer equipment, particularly the keyboard, disk drives, and printer.

Other equipment may be covered, but the items just listed are the ones that are most prone to damage. Dust covers don't only protect the equipment from dust, but also from deodorant, wax, polish, and other aerosol sprays that can leave a film on the equipment, causing malfunctions.

Dust covers can also protect against accidents such as liquid spills, cigarette ashes, and other types of material (Fig. 11-1). Make it a rule not to eat, smoke, drink, or do other noncomputer things around your computer. The authors have seen a can of soda disappear into a printer mechanism and have seen the disastrous effects of nondairy creamer on a small computer. It is easy for crumbs and small pieces of "junk" to accumulate between the keys. Such material can also get on your disks and cassettes, causing problems when you try to retrieve or store information and programs.

The two photographs in Fig. 11-2 illustrate the difference between a clean and a dirty disk "head," the part that actually reads and writes the information to and from the disk surface.

Power

It is a good idea to use a "plug strip" as the power source for your computer equipment. Many of these devices have a master switch and a circuit breaker, so all of the equipment can be con-

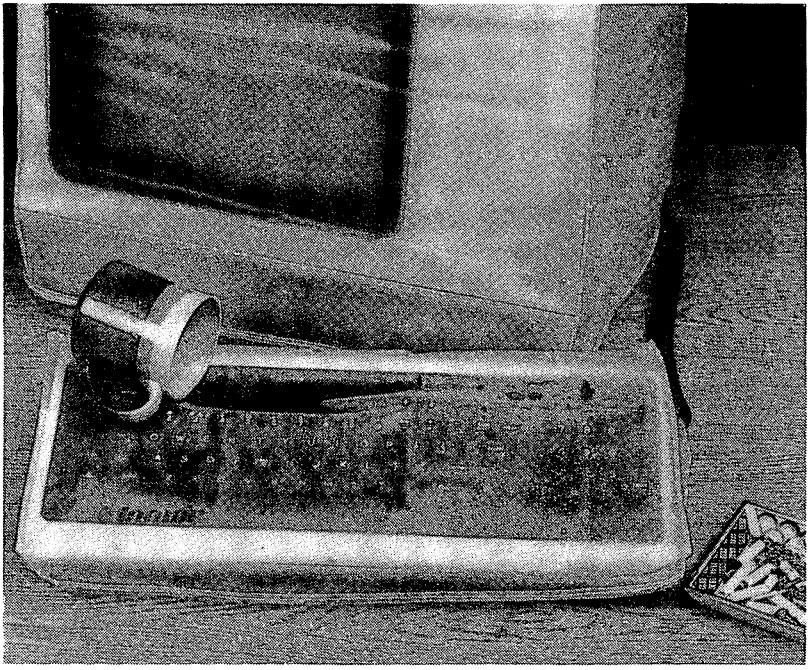


Fig. 11-1. A cover can protect a small computer from dirt, ashes, and other "pollution." *(Courtesy Inmac)*

trolled from one place. With this type of a setup, you won't go away on a trip and suddenly remember that the computer is still on. Remember that in the VIC 20 computer, the "power cube" that is plugged into the wall outlet *has power connected to it as long as it is plugged in.* The power switch on the VIC 20 just connects the cube's power to the computer.

If you're going to turn off your computer for a while, we recommend unplugging the power cube or using a "plug strip" to turn off all the equipment. A typical power strip is shown in Fig. 11-3. These strips are available from most hardware and electrical supply stores.

Power surges aren't normally a problem, although there are companies that advertise surge protectors that are built into plug adapters and strips. Your local situation will determine whether or not you need this kind of protection. You can ask some of the local computer users about their experiences. Lightning often strikes power lines, and it can damage delicate electronic equipment that

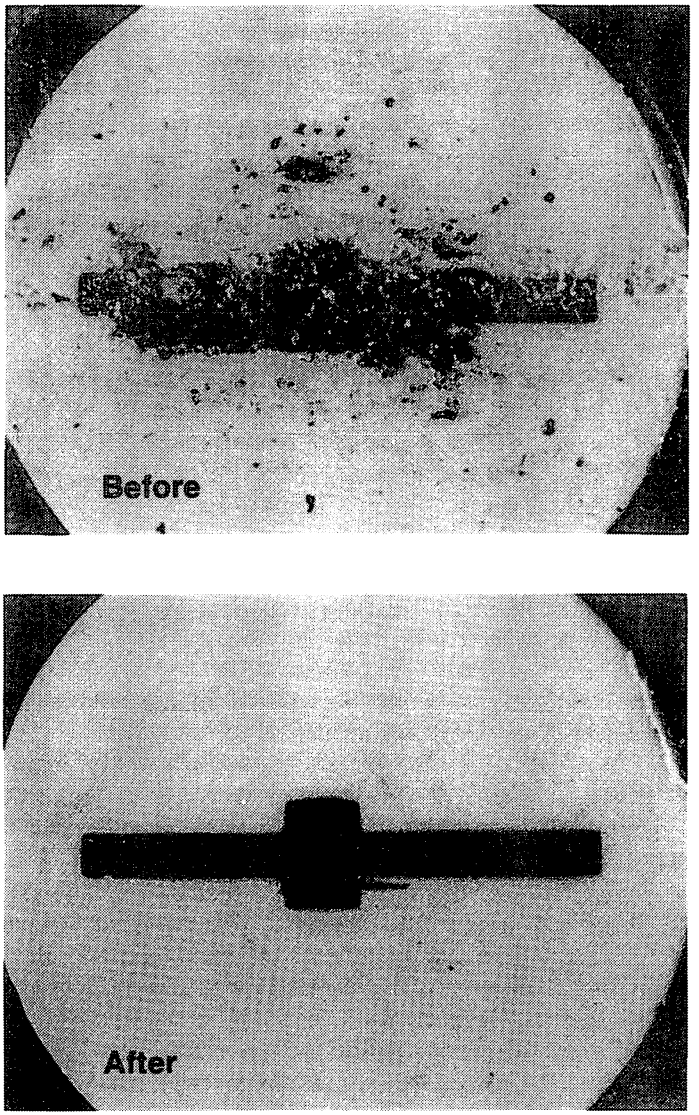


Fig. 11-2. Clean and dirty disk read/write heads.
(Courtesy Radio Shack, a Division of Tandy Corp.)

is plugged in, *even if the equipment is turned off.* We have seen television sets that have been damaged by nearby, indirect lightning strikes to the power line. If lightning storms occur in your

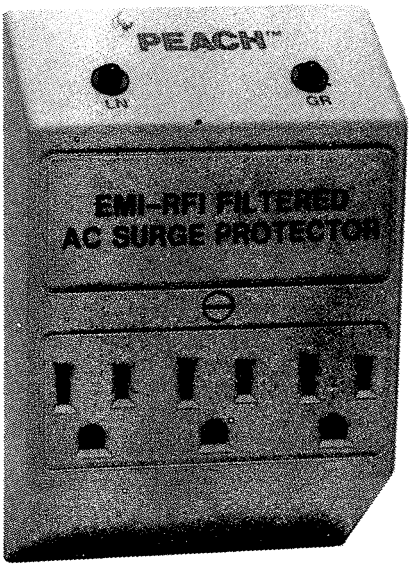


Fig. 11-3. A power strip provides a central source of power.
(Courtesy Electronic Protection Devices, Inc.)

area fairly frequently, it might be wise to unplug your computer equipment when you aren't using it.

Clean Up

All computer equipment needs to be cleaned once in a while. You can easily dust the keyboard unit and television set with a vacuum cleaner, but this isn't recommended for cassette and floppy-disk units. Printers can be dusted carefully with a vacuum cleaner on a low suction setting. Vacuuming can remove paper lint and dust that can cause problems. When dusting, don't use any commercial spray materials that leave a film or residue. If anyone else will be cleaning near the computer equipment, remind them NOT to use such products on the equipment. A commercial window-cleaning product can be used to clean the TV screen, but take the TV set out of the "computer center" so that spray droplets of the cleaner don't get into the other equipment.

Cleaning disk drives and cassette units calls for care. Special kits are available for cleaning both of these devices, with cassette head-cleaning kits available at most record and radio-TV stores. Cleaning kits for floppy-disk drives are available from many com-

puter suppliers, and a typical kit is shown in Fig. 11-4. This kit contains a special cleaning disk and cleaning solvent. The cleaning solvent is squirted onto the cleaning disk and the disk is placed into the disk drive unit. The disk is then activated so that the cleaning pad gently removes any dirt and residue from the disk-reading mechanism. This takes about 30 seconds, and it is a worthwhile task, prolonging the life of both the disk drive unit *and* your disks. Disk heads should be cleaned *once a week,* or more frequently if the disks are used heavily.

Troubleshooting

Most computer manufacturers provide service for the equipment they sell, either through a direct service facility or through service outlets. In most cases, you will have to deliver your computer



Fig. 11-4. A typical disk-cleaning kit. *(Courtesy 3M Co.)*

equipment to the service center, either by shipping it or by hand-carrying it. We suggest checking the information supplied with the computer for service information, or you can call the VIC 20 service desk at Commodore to get more information about having your computer serviced. If servicing is required, be sure to send a complete description of the problem along with the piece of equipment to be repaired. A note telling the service people that the equipment doesn't work isn't very helpful. If the problem can be localized, it will speed the repair job and get the computer back to you quickly.

It isn't too difficult to try and track down some computer problems on your own. The first thing is to be sure that the computer is set up properly, and that you aren't doing something odd to it. For example, check the cables and cords to be sure they are all firmly mated or "seated" in position. Also check to make sure all of the equipment has been turned on. Many people forget to turn on power or connect needed accessories. If the computer still isn't working properly, set up just the computer and the TV set and run a simple BASIC program. Something as simple as typing a few numbers on the TV screen will do fine as a test.

If the basic computer system is operating properly, reconnect the accessories, one at a time, and continue to test the computer. If you have reconnected the cassette unit, try loading and saving a program to test the system. This will help you localize the problem. For example, (1) you have your system connected, except for the printer, and all seems to be operating properly; then (2), the printer is connected, and the system doesn't work. Set up the computer with just the TV and the printer and try it again. If the printer still doesn't work, the problem may be in *either* the computer or the printer.

If you know of another person with a VIC 20 who lives close by, you might arrange to try your computer *and* printer with the other person's equipment. This may narrow the problem even further. Many towns and communities have enough VIC 20 users so a club or users' group has planned activities and even a newsletter. A similar group may exist in your town. Sales people at the store may know about such a group, or you may see a meeting notice in a local newspaper.

There are several computer magazines that contain lists of local interest groups and clubs. You'll find people in these groups who have a wide range of interests, and many of them will be happy to help you.

SOFTWARE

Software has been said to be the most important ingredient of any computer system. It takes a great deal of time to put together computer programs, check them, and get them to work properly, and you will find that software represents your biggest computer investment, even if you purchase only a few programs, or "packages," as they are often called. In this section, some of the details of program selection and evaluation will be described. It isn't particularly difficult to evaluate a simple game or recreational program, but programs for special applications can cost several hundred dollars, so you can't afford to make the wrong choices.

What's Available

Once you have decided that a particular type of program is needed, it is a good idea to try and find out what programs are available to meet your needs. For example, if you are interested in an address-file program, you will have to look for that specific type of program. If you subscribe to a magazine about computers, particularly the VIC 20, you may find several manufacturers or distributors who sell the type of program you are looking for.

A local user's group or computer club may be able to tell you about someone who has a similar program or who may have looked at several. Most people are glad to tell you about their experiences.

Information and References

Once you have found some programs that sound interesting, ask the manufacturers and suppliers for information about them. The responses you get are often good indicators of what the programs are like. A professional response with interesting informative literature generally means that the company operates in a professional manner. On the other hand, photocopied typewritten information that is unreadable or full of computer terms may mean that the company doesn't understand its market or its potential customers.

Use the information about the programs to make comparisons between them. How quickly do the programs run? How much information can they use? Your questions should be based on your needs. For example, if you are looking at a word processing program, you want to know how many pages of typewritten material can be saved on a disk or cassette. You may also want to know if

the program requires a special printer, or if the standard VIC-1525 printer can be used.

Ask the manufacturer, distributor, or dealer for the names of several people who are using the program. Most dealers will share this information with you, and these people will be able to give you an unbiased opinion about the program. Ask them how they like the program, how they are using it, how long they have been using it, and so on. Also ask if they have had any problems with the program, with the dealer, or with the manufacturer. If one or two of these people are located in your area, you might ask if you could visit them and actually see how the program works. Don't ask for a copy of the program to take home and try. This is asking the other person to violate the copyright on the program, and it is illegal.

Demonstrations and Documentation

Visit several dealers, if you can. Ask for demonstrations of the computer program and also ask to look at the user's manual or other written materials that come with the program. This is called "documentation," and it is what you will use to guide you when you use the program. One thing to look for is clearly written information that is easy to read and understand. If it seems to have been written for a computer expert, more than likely the program will be set up the same way—for an expert.

The written materials and instructions are probably the most important part of a computer program, since they tell you how to effectively use the program. Unfortunately, many program manufacturers do a poor job of preparing a manual for the user. In looking at a manual, check for an index so you can quickly locate information. A set of examples is also important, since it lets you see how the program works and what it actually looks like when it is running. Many manuals contain exercises that you can do to become familiar with the program, along with photographs or examples of the TV screen display

If a visit to a dealer isn't possible, you should at least review the instructions for the program package. Many software suppliers realize that people want to look at this material, so it is sold separately. In some cases, suppliers will give you credit for this purchase if you decide to buy the complete software package within a limited time. A smaller number of manufacturers supply demonstration or "demo" copies of their programs so you can try them. These "demo" programs may not be able to do everything the regular program can, but they will give you a good idea of how the program operates.

Testing Programs

If you can actually try the program, so much the better. You'll be able to see how it will work in your application. No program is perfect in the sense that it does everything exactly as you want it to. Of course, the program should give the correct results, but it may display the results on the bottom of the screen, while you'd rather have them at the top. These things are usually minor, unless you find that the final result of the program is not what you want.

If a dealer will not supply you with a program that you can test, you may be able to try the program in his store or office. Other users may let you try the program at their office or home, giving you an hour or two to use it. A few hours of testing time is a worthwhile "investment."

If you are spending a considerable amount of money to buy the program, and if you expect to use the program frequently, testing is almost mandatory. How you test the program depends on what kind of program it is. Business accounting programs and word processing programs do different things, so they are tested differently. During your test, make an effort to cause problems for the computer, to see what happens. For example, if the computer asks for a YES or NO answer, type something else to see if the computer will "trap" your incorrect answer and give you another chance to type in a correct answer. If the computer does something unexpected, there may be other flaws in the program, too. Try as many combinations of commands and conditions as you can.

As you test a program, be sure that it is easy to use and that the "menus" displayed on the TV screen are easy to understand and are meaningful. For example, if the program displays this type of menu at one time:

```
MORE DATA TO BE ENTERED?
    1. YES
    2. NO
TYPE 1 OR 2
```

and this at another:

```
DISPLAY RESULTS AGAIN?
    0. YES
    1. NO
TYPE 0 OR 1
```

it will be difficult to remember whether a 1 is a YES or a NO answer. The answers should be consistent, and in this example it

would have been much easier to just type YES or NO, or, Y or N. If the program is difficult to use, you will spend as much time mastering it as actually using it. Even though many people have "standardized" some difficult-to-use programs doesn't mean that they are the best ones available.

Many programs are confusing just because of the many different things they can do. Many people find that they can do a very good job without many of the special operations provided. In many cases, a simpler less-expensive software package will take care of your needs as well as a more-expensive package that has operations you won't use.

Updates and Revisions

Before you purchase a program, ask the dealer or supplier if it is the latest version. You might ask if a revised or updated program is expected in the near future. Sometimes, minor "bugs" or problems are found in a program, and the manufacturer modifies the program to take care of this. Many manufacturers continue to change their programs to make them easier to use, to change the display format, and so on, in respone to their customers' needs. Ask whether updated programs are going to be available, and what the terms are. Here are some typical arrangements:

1. The current program is the only one available. No revisions, updates, or changes are going to be developed or provided.
2. New programs or documents may be available, but you will have to buy them just as if you were buying a new program.
3. New programs and manuals may be available, and there is a small charge for exchanging your "old" program for the new one.
4. New programs and manuals may be available, and the exchange is free for the first six months you own the program.

The last two choices are the best, since they tell you that the manufacturer is interested in continuing to work with you, and that you are a valued customer.

Some manufacturers provide a "hot-line" telephone service that you can use if you have a problem or difficulty in using the program. Ask if this is available for the software packages you are looking at. This type of service is expensive, so it adds to the cost of the program. Don't expect to have this kind of service for an inexpensive program or game. You might also ask if there is a users' group that is centered around the software packages you are

interested in. Some of the most popular programs have spawned their own groups, so additional information, hints, and assistance is available from an amateur group. Some dealers and suppliers are familiar with software, so you might also ask if they provide any local assistance to their customers. This is a good reason to deal with a local dealer instead of a mail-order dealer.

Flexibility

If you expect your uses for your computer to increase, as they almost always do, you want to be sure that the program will be able to expand, too. For example, a program for teachers that will handle up to 25 students is going to be useless if the next class has 27 students in it. It is often difficult to think about the future and how things may change, but programs should be adaptable to new situations and requirements.

Protection

We know one person who buys phonograph records and immediately makes a tape recording of them. He plays the tapes, saving the record as a "master" or "source" so it won't be damaged by day-to-day use. It is a good idea to do the same thing with programs. Be sure that you can copy or duplicate any programs you buy to provide "backup" copies. In most cases, the original is set aside for an emergency, and the backup copies are used in the computer. Since some manufacturers protect their programs by making it difficult or impossible to copy them, ask the supplier if you will be able to make backup copies for your own use. If the answer is no, ask how you get backup copies of the program. If such copies aren't readily available, look elsewhere for programs.

It is a good idea to make backup copies of your own programs and information. This preserves them so that if a disk or cassette tape is ruined, you still have your information. Disks and tapes don't last forever, although most people have very good luck with them, using them for years. One company we know of recommends throwing out disks after only a month's use. These disks are used every day, so they go through a lot of use very quickly. Don't expect tapes and disks to last forever.

A

VIC 20 Error Codes

COMMON ERROR CODES

BAD SUBSCRIPT—The computer tried to use a subscript that didn't exist in the array. For example, A = RD(57) where the array, RD, only goes from RD(0) to RD(40).

CAN'T CONTINUE—The computer cannot continue after you typed the CONT command. You may have changed something in the program itself, or the program may not have been started. You can't continue after an error message, either.

DIVISION BY ZERO—The computer just can't divide zero into another number. Neither can you, since the result is infinity.

EXTRA IGNORED—You typed in something "extra" when the computer asked for information. The "extra" thing depends on what type of information you are typing in; string or value.

ILLEGAL DIRECT—You have asked the computer to do something it cannot do. For example, you can tell the computer to PRINT 5 + 89 right from the keyboard, but you can't tell it to *INPUT X* : PRINT X * 80, unless this is put in a program with a line number.

ILLEGAL QUANTITY—You have asked the computer to do something with a number that is too big or too small for it to handle. The computer can handle numbers up to 1 with 35 zeros after it, and down to a 1 with 35 zeros before it!

NEXT WITHOUT FOR—You always need to use NEXT and FOR together. The computer will tell you if you have forgotten the FOR command.

OUT OF DATA—This error takes place when you try and get more pieces of information from a DATA statement than are actually there. Make sure you don't try and READ more information than there is in the DATA statement.

OUT OF MEMORY—Wow, you completely filled up the computer's memory! The solutions are to cut your program into smaller chunks, save some information on tape or disk, or add a memory expansion plug-in cartridge.

OVERFLOW—The result of an operation is just too big for the computer to handle. Look for errors in your math, or have the computer work with smaller numbers. Since the computer can count the number of water molecules in a *cubic kilometer* of water, you're talking about a big, big number.

REDO FROM START—The computer expected a value, and you typed in some other character. The complete entry has been ignored, so just type it in again and try and get it right.

RETURN WITHOUT GOSUB—The computer found a RETURN command in the program, but the program never told it to GOSUB. Check your program and be sure that you have a GOSUB command for the subroutine. If you have unused subroutines, that's fine, but just be sure that the computer hasn't gotten to them with a GOTO or other command. Remember to use the END command to stop the computer when necessary so it doesn't go off into other programs.

STRING TOO LONG—Strings saved in the computer can have up to 255 characters. Strings input with an INPUT W$ command can have up to 89 characters in them.

SYNTAX—This means that something was messed up and the computer couldn't understand what you wanted it to do. Check the line noted in the error message.

NOT-SO-COMMON ERROR CODES

BAD DATA—The computer expected to receive numeric data from a "file" or peripheral device. It got non-numeric characters, or a string, instead.

DEVICE NOT PRESENT—You have told the computer to use a peripheral that hasn't been hooked up or turned on yet.

FILE NOT FOUND—The computer has looked at a disk or tape, but it couldn't find the file you asked it to read. The file name cannot be found.

FILE NOT OPEN—You can't use a file without opening it first. Go back and be sure that you have the appropriate OPEN commands in your program.

FILE OPEN—You have tried to open a file that is already open. You can't open something unless it's closed. Check to be sure that you closed your files, or that you haven't opened them twice.

FORMULA TOO COMPLEX—Split your formula into smaller pieces. You have asked the computer to do too much at one time.

LOAD . . .—The computer is having a problem loading your program from the cassette recorder.

NOT INPUT FILE—You have tried to get information from an output-only file. The printer is an output-only device, so don't try and get information *from* it.

NOT OUTPUT FILE—You have tried to output data to an input device.

REDIM'D ARRAY—Once you set up an array, you can't change its size within the program. If you need to change the size of an array, stop the program, change the DIM command, and rerun the program.

TYPE MISMATCH—You have tried to use a value in place of a string, or a string in place of a value. The two are separate types of information, so check what you are doing.

UNDEF'D FUNCTION—The computer was told to do a "special function," but you never told it what the function was.

UNDEF'D STATEMENT—You told the computer to go to a line number that isn't included in your program. This type of error occurs with GOTO and GOSUB commands.

VERIFY . . .—The computer cannot verify your program, saved on tape or disk.

B

VIC 20 BASIC Keywords

ABS	GET	ON	SPC(
AND	GET#	OPEN	SQR
ASC	GOSUB	OR	STATUS
ATN	GOTO	PEEK	STEP
CHR$	IF	POKE	STOP
CLOSE	INPUT	POS	STR$
CLR	INPUT#	PRINT	SYS
CMD	INT	PRINT#	TAB(
CONT	LEFT$	READ	TAN
COS	LEN	REM	THEN
DATA	LET	RESTORE	TIME
DEF	LIST	RETURN	TIME$
DIM	LOAD	RIGHT$	TO
END	LOG	RND	USR
EXP	MID$	RUN	VAL
FN	NEW	SAVE	VERIFY
FOR	NEXT	SGN	WAIT
FRE	NOT	SIN	π

C

Answers to Questions

CHAPTER 1

1. To use channel 3, set the TV channel selector switch to channel 3 and set the switch on the box in the TV cable assembly to CH3. Remember to move the TV antenna switch to the COMPUTER position.

2. There are *printing* and *action* keys on your VIC 20 computer. The printing keys print one or more characters on the TV screen. The action keys can perform actions on their own; for example, the CLR/HOME key; or they can change the operation of the printing keys; for example, the GRAPHICS key doesn't do anything by itself, it just lets you print special graphic characters.

3. The shift key has two uses:

   (a) It lets you print the upper symbol for those keys that have an "upper" and a "lower" symbol on their keytops.
   (b) It lets you print the right-hand graphic symbol for those keys that have a graphic symbol on their front surface.

4. The CLR/HOME key can be used by itself to move the cursor to the upper left-hand corner of the TV screen's display area. When used with the SHIFT key, the CLR/HOME key clears the display area *and* moves the cursor to the upper left-hand corner.

5. The two cursor control keys, CRSR, are used to move the cursor on the TV screen. The cursor can be moved down, right, up, or left. The

SHIFT key must be used to move the cursor up or left. When the cursor is moved, no characters are "erased."

6. Yes, several of the keys "repeat" their action. The space bar and the two cursor control keys are examples.

7. The INST/DEL (insert/delete) key is used to correct errors. You can use it to remove or "erase" characters and to insert new ones on the TV display. The insert operation uses the SHIFT key.

8. The GRAPHICS key is located in the lower left corner of the keyboard, and it looks as though it has a large "C" with an equal sign on it. It lets you type the left-hand graphic symbols for the keys that have graphic symbols on the front of them.

9. Colors are changed by using the CTRL key and the number keys, "1" through "8." The colors are noted on the front of these keys. You must press and hold the CTRL key when you press the color keys.

10. The SHIFT and "6" keys will print the "and" symbol, &, on the TV screen. No color change takes place. The CTRL key and a number key must be used to change the color of the cursor.

11. No, changing the color of the cursor doesn't change the color of anything else on the screen.

CHAPTER 2

1. The VIC 20 can process both *values* and *strings of characters.*

2. In general, computers are used to *process* information and to *transfer* information.

3. Of the 12 labels, several are not valid. These are $LABEL, 31ST STREET, and 3STATE$. Here is why they cannot be used:

   (a) $LABEL (Labels cannot start with a dollar sign.)
   (b) 31ST STREET (Labels cannot start with a number.)
   (c) 3STATE$ (Labels cannot start with a number.)

4. Yes, there is a conflict between SPAGHETTI and SPELLER, since the computer would see both as the label SP for use with a value. There is no conflict between SPEND$ and either SPAGHETTI or SPELLER, since SPEND$ is a label for a string, and its two-letter label would be ST$.

5. There are several errors in these labels and information. They are:

   (a) TEST = $56. (This must be TEST = 56. The dollar sign cannot be used as part of a value.)

(b) TIME = "TEN OF FIVE"$	(This must be: TIME$ = "TEN OF FIVE" The dollar sign is in the wrong place.)
(c) TM = 56.89	(This is correct.)
(d) LINK$ = NUMBER OF TESTS	(The string must be enclosed in quotes: LINK$ = "NUMBER OF TESTS")
(e) SEEDS = 78	(This is correct.)
(f) LINES = "10 LINES PER INCH"	(The label, LINES, must have a dollar sign after it to label a string: LINES$ = "10 LINES PER INCH"

6. There are errors in several of these commands. The corrections have been underlined for you:

   (a) INPUT "YOUR NAME, PLEASE"; NAME$
   (b) PRINT "TEST RESULTS"
   (c) PRINT "BALANCE IN ACCT ="BAL (No semicolon needed)
   (d) INPUT " TODAY'S SPECIAL "; SPEC
   (e) INPUT "ENTER THE YEAR"; YR$
   (f) PRINT "THIS IS A VIC 20"

7. An entry with an incorrect line number can be erased by simply typing the old line number and [RETURN]. This "erases" the line from the computer program.

8. You can finish the line, type [RETURN] and re-enter it. You can also use the INST/DEL key to back-up and erase errors.

9. To start a BASIC program, just type RUN [RETURN].

10. To list a program on the TV screen, simply type LIST [RETURN].

CHAPTER 3

1. So far, the VIC 20 can do addition, subtraction, multiplication, and division.

2. The VIC 20 does the multiplication and division operations in a program step first, and addition and subtraction operations second.

3. You can put almost as many program steps on a line as you want. Programs with too many steps are difficult to "read" and understand, so don't try and cram too many on a line.

4. Individual program steps are separated by colons when they are combined in a single program line.

5. Parentheses are used to separate math operations so they can be easily broken down. Individual operations are easy to understand when parentheses are used, and there is no question about which operations are done in what order.

6.

(a)	? = 5 * 12 + 4	(Answer: 64)
(b)	? = 3 * 4 + 12 / 5 * 8	(Answer: 31.2)
(c)	? = ((((4 * 6) / 3) + 18) −7)	(Answer: 19)

7. REM stands for REMark, and it is used to put a comment or note in a program so that the programmer will understand why something was done, what value is being used, and so on. The program doesn't *do* anything with a remark, it skips over it to the next real program step.

8. The INPUT command lets you have the computer get information from the keyboard. A label must be used with an INPUT command so the computer knows how to identify the information.

9. The computer always uses labels to identify and locate information.

10. The INST/DEL (insert/delete) key is used to help you correct program lines and other information on the TV screen. The INST/ DEL key uses the cursor as its "marker." The changes only go into effect after the RETURN key is pressed.

CHAPTER 4

1. The GOTO command causes the computer to go to a specific line in the program. It doesn't care where the line is or what command is there.

2. The GOTO command must be followed by a line number that is really part of the program.

3. The GOTO command will cause an error condition when the line number used is not found by the computer. The error tells you that something (the line number) is not properly defined in the GOTO command.

4. The GOTO command can point the computer to any line number in the program.

5. You can get out of an "endless" loop by pressing the RUN/ STOP key. You can also pull the power plug to the computer, but that will "erase" your program!

6. You can use the RUN/STOP and the RESTORE keys to get the computer out of an INPUT command when it's waiting for information from the keyboard. You can pull the plug, too, but that is not the recommended solution.

7. The VIC 20 can evaluate, or test, almost any statement that can be answered with true or false.

8. The statements are used in IF-THEN commands. The computer evaluates the statement to see if it is true or false. The answer is used to make the decision.

9. The conditions of equal-to, greater-than, or less-than can be tested. These conditions can be combined; for example, less-than OR equal-to.

10. Any of the BASIC commands can follow the THEN in an IF-THEN command. Multiple commands can be used, too, as long as they are separated by a colon.

11. The AND and OR operations let the computer test for several conditions in a single IF-THEN command. For example, IF (A = GR) AND (FX > 6) THEN . . . The NOT operation reverses the true-false result when a statement is evaluated by the computer. For example, the statement "7 = 7" is true, while the statement "NOT (7 = 7)" is false.

12. The RND(1) operation is used to get a random number between 0 and 1. Other operations are used to "scale" the value and to "strip off" any decimal fractions that aren't wanted.

13. You can remove the decimal fraction part of a value by using the INT operation. For example, PRINT INT(15.962) would display 15 on the TV screen.

CHAPTER 5

1. Just use (press) the CLR/HOME key in the message part of the PRINT command:

```
130  PRINT "[CLR/HOME]"
```

   This gives you a "reversed S" and it looks like this:

```
130  PRINT "  ▌  "
```

2. You can combine printing operations on a single line by using a semicolon after each PRINT command. For example:

```
200  PRINT "TESTING";
220  PRINT " 1, 2, 3"
```

   will print:

```
TESTING 1, 2, 3
```

   on the TV screen.

3. The TAB command is used to space columns of information on the TV screen in an orderly form. The TAB(5) command moves the cursor five spaces along the line, away from the left margin.

4. You cannot move the cursor back to the left with a TAB operation. Once you have done a TAB(15) operation to column 15 on the TV screen, you can't move back to column 8 on the same line with a TAB(8) operation.

5. The ON command acts like a group of IF-THEN commands. It checks a "statement" to see if it is equal to 1, equal to 2, equal to 3, and so on.

6. The ON command would be used when answers of 1, 2, 3, 4, and so on are expected. These results are used to point the computer to specific line numbers in a program. If the condition is not found, the program just continues with the next following instruction.

7. The FOR-NEXT command is used to count the number of passes through a program loop. You must set the starting and final count, and you must give the loop counter a label. For example:

```
20  FOR TX = 1 TO 14
30      PRINT "*"
40  NEXT TX
```

   This program would print 14 asterisks on a line on the TV display.

8. You can use almost any size steps you want to. You must specify them in the FOR part of the command with a step size:

```
20  FOR TX = 5 TO 200 STEP 5
```

   Fractional and negative steps can be used also:

```
30  FOR GM = 1 TO 50 STEP 0.5
```

   or

```
40  FOR DSA = 100 TO 20 STEP -2
```

9. A nested loop is one loop within another:

```
20  FOR T = 1 TO 20 ------------------------------+
30      FOR GH = 1 TO 10 ---------+               |
40          INPUT F               |  Inner        |  Outer
50          PRINT F * 12          |  Loop         |  Loop
60      NEXT GH ------------------+               |
70      PRINT " * "                               |
80  NEXT T ---------------------------------------+
```

10. A subroutine is a "piece" of a program that is used over and over again by different parts of a "main" program. It is easier to put the

group of often-used instructions in a subroutine than to duplicate the steps everywhere they are needed in the main program. A subroutine might be used to calculate the sales tax on several different types of sales transactions.

11. An array is a collection of similar information that has an overall label or name. It is used to store or save information that would require many different labels. Arrays are particularly useful for saving check values, game scores, test results, and so on.

12. Arrays are flexible since you can use a label as a subscript to identify a piece of information in the array. In this way, you can easily change the subscript used to point to the information. In effect, you are changing the label so it can point to new information. For example, the following program would get seven values from the array called DR and print them on the TV screen:

```
400 FOR AX = 1 TO 7
420     PRINT DR(AX)
440 NEXT AX
```

    There is no easy way to do this with usual, nonarray labels. The arrays used in the VIC 20 can save up to several hundred pieces of information. If you are going to go above 11 pieces of information (INFO(0) to INFO(10), you'll need to use a dimension statement, DIM, at the start of your program.

13. The DATA statement lets you put data right in your program without labeling it. The READ statement gets the information from the DATA statement, one piece at a time, just as if it were a list of information. Once you get the information this way, you can print it, process it, or do almost anything you want with it. The RESTORE command points the computer back to the start of the DATA statment, so the READ command will get information from the start of the list again.

CHAPTER 6

1. All string labels end with a dollar sign; for example, EST$.

2. Yes, a string of up to 89 characters can be input with an INPUT command. The INPUT command must use a string label: INPUT WET$.

3. Strings are combined with a "plus" sign (+). This links the strings as if they were written one after another: A$ + TR$ + "FUN" + RZ$.

4. Yes, strings can be combined in any order you want.

5. The comma, quote, space, and colon are ignored by an INPUT command. The GET command can be used to input them in a string.

6. The most common decisions are based on comparing two strings to see if they are the same, or not the same. The greater-than and less-than comparisons don't make a lot of sense when used with strings.

7. The LEN command will give you the number of characters in a string, or its length. All of the characters except the quotes are counted. For example, there are seven characters in this string: "TXSD.8*"

8. The left three characters and the right six characters can be obtained from the string CAT$ by using the LEFT$ and RIGHT$ operations: LS$ = LEFT$(CAT$,3) and RS$ = RIGHT$(CAT$,6).

9. You can get the middle five characters from G$ = "QUESTIONS" by using the MID$ operation: MS$ = MID$(G$,3,5). This gets the middle five characters, starting with the third one from the left "end" of the string.

10. If the GET command is done by the computer and no key is pressed, the computer gets a "null," or "nothing" character. It has no meaning in a string and is totally ignored.

11. A six-digit, 24-hour digital clock is built in. It tells time to the nearest second and you can use it by using the TI$ string label.

12. A real-time clock is one that continues to keep accurate time no matter what the computer is doing (unless it's unplugged!).

13. Yes, values and strings can be interchanged, but this only makes sense for changing values into strings of numbers, and *vice versa*. You cannot change the string "BLUE SKY" into a value.

CHAPTER 8

1. The SPC command moves the cursor to the right. The number of spaces to be moved is included in the command. The SPC command is used in a PRINT command to move the cursor.

2. The SPC command moves the cursor from its present position, while the TAB command moves the cursor based on the left edge of the display. For example, the TAB(7) command moves the cursor seven spaces to the right from the left edge of the display, while SPC(7) moves the cursor seven positions to the right *from its present position*, wherever that is.

3. When you try and use the CRSR keys while typing the "message" part of a PRINT command, special symbols are displayed on the line. This means that the computer will only move the cursor when it reaches this message in the program. Each cursor movement has its own special symbol, as shown earlier in Table 8-1.

4. There are many commands and keys that can be used—the four cursor control keys, the CLR/HOME key, the space key, and the SPC and TAB commands. There are many ways to control the position of the cursor on the TV screen.

5. The END command tells the computer that it has reached the end of a program. It is useful when there are several programs in the computer at one time. The END command prevents the computer from running from one program into the next.

6. The color of the display can be changed by using one of the color keys, located on the number keys, 1—8, and the CTRL key. These color-change operations can be used in the "message" part of a PRINT command, too.

7. When the color is changed, only the cursor's color changes. After the cursor has changed its color, anything typed or printed is shown in the new color. Information, displays, or graphics already on the TV screen are not changed.

8. You can change the color of the information to be printed by using the CTRL and color keys within the "message" part of a PRINT command.

9. The RVS ON and RVS OFF keys can be used to control the printing in reverse. These can be typed directly from the keyboard, or the actions can be placed in the "message" part of a PRINT command. The computer does not stay in the reverse mode until it is changed back to the normal mode. There are several built-in computer actions that change it back. In general, you must set the reverse mode each time you need it.

10. The TV is controlled by the display and color maps. There are 23 × 22, or 506 locations in each map.

11. Special codes must be placed in *both* maps to control the TV display. The display map requires a screen code for the character to be displayed, and the color map requires a color code for the color to be used. This information must be placed in the corresponding "squares" in the map for a single location on the TV display.

12. A character can be reversed by adding 128 to its screen code. Subtracting 128 returns the character to normal.

13. The POKE command lets you place a code in a map, while the PEEK command lets you get a *copy* of the code from a location in a map.

14. The VIC 20 has two sets of characters, Set 1 and Set 2, as shown in Table 8-3. Normally, Set 1 is used. You can switch from set to set by using the GRAPHICS and SHIFT keys together, or you can use the

POKE 36869, 242 command to get Set 2, and POKE 36869, 240 command to get Set 1. Switching from one set to the other changes the entire display.

CHAPTER 9

1. The VIC 20 has four "speakers" or sources of sound.
2. Three of the "speakers" create tones, while the fourth one creates "noise" that is a mixture of a hissing sound and a tone.
3. Yes, you can control the volume. You can set the volume of the TV set, or you can set the volume by poking a control code into the volume address in the sound map. You can also turn each speaker off by poking a 0 into its corresponding address in the sound map.
4. Each "speaker" is supposed to create 128 different tones, using codes of 128 through 255. We found that in our computer, only the codes from 128 to 254 created unique tones. We avoided code 255, since it created the same tone as code 128.
5. Yes, there is some overlap between the tones, as shown in Table 9-1.
6. Noise is a collection of random sounds all mixed together. The result is a "hiss."
7. Yes, all of the sounds can be used at the same time, but the overall result may be a mish-mash of sounds. Some interesting effects are possible, and it can be fun to experiment with the sound part of the VIC 20 computer.
8. There are many uses, beyond the ones we described. Sounds can call your attention to something, tell you that a limit has been reached, sound an alarm, beep and bop for games, and on and on.

D

Solutions to Selected Problems

CHAPTER 3

Problem 3-1 Listing

```
10 INPUT "FIRST NUMBER";F
20 INPUT "SECOND NUMBER";S
30 INPUT "THIRD NUMBER";T
40 PRINT
50 PRINT "ANSWER IS";(F+S)*T
60 END
```

Problem 3-1 Sample Run

```
FIRST NUMBER?  8
SECOND NUMBER?  4
THIRD NUMBER?  5

ANSWER IS 60

READY.
```

Problem 3-3 Listing

```
10 PRINT "HOW MANY DOLLARS"
20 PRINT "DO YOU WANT TO"
30 INPUT "CONVERT ?";D
40 PRINT
50 PRINT "YOU WILL GET";2.5*D
60 PRINT "SWISS FRANCS FOR THIS"
70 PRINT
80 END
```

Problem 3-3 Sample Run

```
HOW MANY DOLLARS
DO YOU WANT TO
CONVERT ? 100

YOU WILL GET 250
SWISS FRANCS FOR THIS


READY.

HOW MANY DOLLARS
DO YOU WANT TO
CONVERT ? 25.3

YOU WILL GET 63.25
SWISS FRANCS FOR THIS


READY.
```

Problem 3-6 Listing

```
10 PRINT "HOW MUCH DOES THE"
20 PRINT "FIRST PACKAGE"
30 INPUT "WEIGHT ";FW
40 INPUT "ITS PRICE ";FP
50 PRINT
60 PRINT "HOW MUCH DOES THE"
70 PRINT "SECOND PACKAGE"
80 INPUT "WEIGHT ";SW
90 INPUT "ITS PRICE ";SP
100 PRINT
110 PRINT "FIRST PACKAGE COSTS"
120 PRINT "$";FP/FW; "PER UNIT"
130 PRINT "WEIGHT"
140 PRINT
150 PRINT "SECOND PACKAGE COSTS"
160 PRINT "$";SP/SW;"PER UNIT"
170 PRINT "WEIGHT"
180 END
```

Problem 3-6 Sample Run

```
HOW MUCH DOES THE
FIRST PACKAGE
WEIGHT ? 24
ITS PRICE ? 1.49

HOW MUCH DOES THE
SECOND PACKAGE
WEIGHT ? 36
ITS PRICE ? 1.85

FIRST PACKAGE COSTS
$ .0620833333 PER UNIT
WEIGHT

SECOND PACKAGE COSTS
$ .0513888889 PER UNIT
WEIGHT

READY.
```

Problem 3-9 Listing

```
10 PRINT "HOW MANY DRINK CANS"
20 INPUT "DO YOU HAVE ";N
30 PRINT
40 PRINT "ASSUMING:"
50 PRINT "14 CANS/POUND"
60 PRINT "AND $1.28/5 POUNDS"
70 PRINT
80 PRINT "YOU WILL RECEIVE $"(N/14)*(1.28/5)
90 PRINT
100 END
```

Problem 3-9 Sample Run

```
HOW MANY DRINK CANS
DO YOU HAVE ? 300

ASSUMING:
14 CANS/POUND
AND $1.28/5 POUNDS

YOU WILL RECEIVE $ 5.48571429


READY.

HOW MANY DRINK CANS
DO YOU HAVE ? 70

ASSUMING:
14 CANS/POUND
AND $1.28/5 POUNDS

YOU WILL RECEIVE $ 1.28


READY.
```

Problem 3-11 Listing

```
10 PRINT "ALL DIMENSIONS ARE IN"
20 PRINT "FEET !!!"
30 PRINT
40 INPUT "LENGTH ";L
50 INPUT "WIDTH ";W
60 PRINT
70 PRINT (L*W)/43560 "ACRES"
80 END
```

Problem 3-11 Sample Run

```
ALL DIMENSIONS ARE IN
FEET !!!

LENGTH ? 300
WIDTH ? 456

 3.14049587 ACRES

READY.

ALL DIMENSIONS ARE IN
FEET !!!

LENGTH ? 3028
WIDTH ? 1120

 77.8549128 ACRES

READY.
```

Problem 3-15 Listing

```
10 PRINT "HOW MANY DOLLARS DO"
20 INPUT "YOU HAVE IN THE BANK";M
30 PRINT
40 PRINT "WHAT IS THE YEARLY"
50 PRINT "INTEREST RATE"
60 PRINT "(THIS WILL BE DIVIDED"
70 PRINT "BY 12 TO GIVE YOU THE"
80 INPUT "MONTHLY RATE)";I
90 MR=(I/100)/12
100 X=1
110 M=M+(M*MR)
120 X=X+1
130 IF X<13 THEN 110
140 PRINT
150 PRINT "YOU NOW HAVE";M
160 PRINT "IN THE BANK"
170 END
```

Problem 3-15 Sample Run

```
HOW MANY DOLLARS DO
YOU HAVE IN THE BANK? 100

WHAT IS THE YEARLY
INTEREST RATE
(THIS WILL BE DIVIDED
BY 12 TO GIVE YOU THE
MONTHLY RATE)? 8

YOU NOW HAVE 108.299951
IN THE BANK

READY.
```

CHAPTER 4

Problem 4-1 Listing

```
10 INPUT "FIRST NUMBER";A
20 INPUT "SECOND NUMBER";B
30 INPUT "THIRD NUMBER";C
40 IF (A<B) AND (A<C) THEN X=(B*C)/A
50 IF (B<A) AND (B<C) THEN X=(A*C)/B
60 IF (C<A) AND (C<B) THEN X=(A*B)/C
70 PRINT "ANSWER";X
80 PRINT: GOTO 10
```

Problem 4-1 Sample Run

```
FIRST NUMBER ? 5
SECOND NUMBER ? 6
THIRD NUMBER ? 2
ANSWER 15

FIRST NUMBER ? 8
SECOND NUMBER ? 3
THIRD NUMBER ? 6
ANSWER 16
```

Problem 4-3 Listing

```
10 PRINT "HOW MUCH MONEY DO"
20 INPUT "YOU WANT TO CHANGE";M
30 IF M<=0 THEN 10
40 PRINT
50 Q=0
60 D=0
70 N=0
80 M=M-.25
90 IF M<0 THEN 120
100 Q=Q+1
110 GOTO 80
120 M=M+.25
130 M=M-.10
140 IF M<0 THEN 170
150 D=D+1
160 GOTO 130
170 M=M+.10
180 M=M-.05
190 IF M<0 THEN 220
200 N=N+1
210 GOTO 180
220 P=INT((M+.051)*100)
230 PRINT "QUARTERS:";Q
240 PRINT "DIMES:";D
250 PRINT "NICKELS:";N
260 PRINT "PENNIES:";P
265 PRINT
270 GOTO 10
```

Problem 4-3 Sample Run

```
HOW MUCH MONEY DO
YOU WANT TO CHANGE?  2.39

QUARTERS: 9
DIMES: 1
NICKELS: 0
PENNIES: 4

HOW MUCH MONEY DO
YOU WANT TO CHANGE?  1.47

QUARTERS: 5
DIMES: 2
NICKELS: 0
PENNIES: 2
```

Problem 4-6 Listing

```
10 A=INT(RND(1)*50)
20 B=INT(RND(1)*50)
30 PRINT A;" + ";B;" =";
40 INPUT Q
50 IF Q<>A+B THEN 30
60 GOTO 10
```

Problem 4-6 Sample Run

```
44  +  27  =? 71
6  +  13  =? 19
12  +  10  =? 22
14  +  11  =? 20
14  +  11  =? 24
14  +  11  =? 25
10  +  45  =? 55
15  +  32  =? 47
33  +  48  =
```

Problem 4-8 Listing

```
10 Q=INT(RND(1)*100)
20 INPUT "YOUR GUESS";G
30 IF G<Q THEN PRINT "TOO LOW"
40 IF G>Q THEN PRINT "TOO HIGH"
50 IF G<>Q THEN 20
60 PRINT "THAT'S RIGHT"
70 GOTO 10
```

Problem 4-8 Sample Run

```
YOUR GUESS ? 50
TOO LOW
YOUR GUESS ? 75
TOO HIGH
YOUR GUESS ? 62
TOO HIGH
YOUR GUESS ? 56
TOO LOW
YOUR GUESS ? 59
TOO LOW
YOUR GUESS ? 61
THAT'S RIGHT

YOUR GUESS ? 50
TOO LOW
YOUR GUESS ? 75
TOO HIGH
YOUR GUESS ? 62
TOO HIGH
YOUR GUESS ? 56
TOO LOW
YOUR GUESS ? 59
THAT'S RIGHT
```

Problem 4-11 Listing

```
10 INPUT "DAY OF YEAR";N
12 IF N>31 THEN 20
14 PRINT "JANUARY";
16 GOTO 200
20 N=N-31
22 IF N>28 THEN 30
24 PRINT "FEBRUARY";
26 GOTO 200
30 N=N-28
32 IF N>31 THEN 40
34 PRINT "MARCH";
36 GOTO 200
40 N=N-31
42 IF N>30 THEN 50
44 PRINT "APRIL";
46 GOTO 200
50 N=N-30
52 IF N>31 THEN 60
54 PRINT "MAY";
56 GOTO 200
60 N=N-31
62 IF N>30 THEN 70
64 PRINT "JUNE";
66 GOTO 200
70 N=N-30
72 IF N>31 THEN 80
74 PRINT "JULY";
76 GOTO 200
80 N=N-31
82 IF N>31 THEN 90
84 PRINT "AUGUST";
86 GOTO 200
90 N=N-31
92 IF N>30 THEN 100
94 PRINT "SEPTEMBER";
96 GOTO 200
100 N=N-30
102 IF N>31 THEN 110
104 PRINT "OCTOBER";
106 GOTO 200
110 N=N-31
112 IF N>30 THEN 120
114 PRINT "NOVEMBER";
116 GOTO 200
120 N=N-30
124 PRINT "DECEMBER";
200 PRINT N
210 PRINT
220 GOTO 10
```

Problem 4-11 Sample Run

```
DAY OF YEAR?  1
JANUARY 1

DAY OF YEAR?  15
JANUARY 15

DAY OF YEAR?  100
APRIL 10

DAY OF YEAR?  365
DECEMBER 31
```

Problem 4-13 Listing

```
10 PRINT "HOW MANY CHECKS DID"
20 PRINT "YOU WRITE THIS"
30 INPUT "MONTH";NC
40 PRINT
50 X=1
60 TC=0
70 INPUT "AMOUNT OF CHECK";A
80 TC=TC+A
90 X=X+1
100 IF X<=NC THEN 70
110 PRINT
120 PRINT "TOTAL OF CHECKS";TC
130 PRINT
140 X=1
150 TD=0
160 PRINT "NUMBER OF"
170 INPUT "DEPOSITS";ND
180 PRINT
190 INPUT "DEPOSIT AMT.";D
200 TD=TD+D
210 X=X+1
220 IF X<=ND THEN 190
230 PRINT
240 INPUT "PREVIOUS BALANCE";PB
250 PRINT "CHECKS";TC
260 PRINT "DEPOSITS";TD
270 PRINT "NEW BALANCE";PB+TD-TC
280 END
```

Problem 4-13 Sample Run

```
HOW MANY CHECKS DID
YOU WRITE THIS
MONTH?  3

AMOUNT OF CHECK? 10
AMOUNT OF CHECK? 20
AMOUNT OF CHECK? 5

TOTAL OF CHECKS 35

NUMBER OF
DEPOSITS? 2

DEPOSIT AMT.? 35
DEPOSIT AMT.? 65

PREVIOUS BALANCE? 300
CHECKS 35
DEPOSITS 100
NEW BALANCE 365
```

CHAPTER 5

Problem 5-1 Listing

```
100 PRINT "[SHIFT] [CLEAR/HOME]";
110 FOR I=0 TO 20
120 PRINT TAB(I) "[SHIFT] [M]"
130 NEXT
140 GOTO 140
```

Problem 5-3 Listing

```
10 PRINT "[SHIFT] [CLEAR/HOME]"
20 PRINT "THIS MESSAGE GETS YOUR ATTENTION"
30 FOR J=1 TO 200: NEXT J
40 PRINT "[SHIFT] [CLEAR/HOME]"
50 FOR J=1 TO 200: NEXT J
60 GOTO 20
```

Problem 5-5 Listing

```
10 PRINT
20 PRINT "ADD      - 1"
30 PRINT "SUBTRACT - 2"
40 PRINT "MULTIPLY - 3"
50 PRINT "DIVIDE   - 4"
60 PRINT: INPUT "COMMAND ";C
70 C=INT(C)
80 IF (C<1 OR C>4) THEN 60
90 ON C GOSUB 200,300,400,500
95 GOTO 10
100 PRINT: INPUT "NUMBERS ";X,Y
110 RETURN
200 GOSUB 100
210 A=X+Y
220 PRINT "RESULT IS ";A
230 RETURN
300 GOSUB 100
310 A=X-Y
320 GOTO 220
400 GOSUB 100
410 A=X*Y
420 GOTO 220
500 GOSUB 100
510 A=X/Y
520 GOTO 220
```

Problem 5-5 Sample Run

```
ADD      - 1
SUBTRACT - 2
MULTIPLY - 3
DIVIDE   - 4

COMMAND ?  1

NUMBERS ?  2 , 3
RESULT IS  5

ADD      - 1
SUBTRACT - 2
MULTIPLY - 3
DIVIDE   - 4

COMMAND ?  3

NUMBERS ?  6.5 , 7.2
RESULT IS  46.8
```

Problem 5-7 Listing

```
10 FOR I=2 TO 100
20 FOR X=2 TO I
30 IF (I/X)<>(INT(I/X)) THEN 40
35 B=B+1
40 NEXT X
50 IF B=1 THEN PRINT I
60 B=0: NEXT I
70 END
```

Problem 5-7 Sample Run

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

Problem 5-9 Listing

```
10 DIM T(6)
20 FOR I=1 TO 100
30 X=1+INT(RND(0)*5.999)
40 T(X)=T(X)+1
50 NEXT
60 FOR I=1 TO 6
70 PRINT I;": ";T(I)
80 NEXT
90 END
```

Problem 5-9 Sample Run

```
1 :  25
2 :  7
3 :  21
4 :  14
5 :  17
6 :  16
```

Problem 5-11 Listing

```
10 DIM C(52)
20 FOR I=0 TO 3
30 FOR J=1 TO 13
40 C(I*13+J)=J
50 NEXT J
60 NEXT I
70 S=INT(RND(1)*3.9999)
80 V=1+INT(RND(1)*12.9999)
90 A=C(S*13+V)
100 IF A=0 THEN 70
110 C(S*13+V)=0
120 ON S+1 GOSUB 200,210,220,230
125 PRINT TAB(10);
130 IF A=1 THEN 170
131 IF A=11 THEN 171
132 IF A=12 THEN 172
133 IF A=13 THEN 173
140 PRINT A
150 GOTO 70
170 PRINT "ACE": GOTO 70
171 PRINT "JACK": GOTO 70
172 PRINT "QUEEN": GOTO 70
173 PRINT "KING": GOTO 70
200 PRINT "SPADES:   ";: RETURN
210 PRINT "HEARTS:   ";: RETURN
220 PRINT "DIAMONDS: ";: RETURN
230 PRINT "CLUBS:    ";: RETURN
```

Problem 5-11 Sample Run

```
SPADES:          10
CLUBS:           6
HEARTS:          KING
SPADES:          9
SPADES:          2
DIAMONDS:        3
CLUBS:           ACE
DIAMONDS:        7
CLUBS:           4
CLUBS:           5
CLUBS:           KING
CLUBS:           2
CLUBS:           JACK
HEARTS:          ACE
HEARTS:          QUEEN
SPADES:          QUEEN
DIAMONDS:        10
CLUBS:           7
HEARTS:          6
HEARTS:          4
DIAMONDS:        ACE
DIAMONDS:        JACK
CLUBS:           QUEEN
SPADES:          JACK
HEARTS:          9
CLUBS:           8
HEARTS:          3
HEARTS:          5
DIAMONDS:        QUEEN
CLUBS:           9
DIAMONDS:        2
HEARTS:          8
CLUBS:           3
DIAMONDS:        4
CLUBS:           10
DIAMONDS:        9
SPADES:          4
HEARTS:          JACK
SPADES:          3
SPADES:          ACE
HEARTS:          2
HEARTS:          7
HEARTS:          10
SPADES:          7
DIAMONDS:        6
DIAMONDS:        5
DIAMONDS:        8
SPADES:          8
SPADES:          KING
SPADES:          6
DIAMONDS:        KING
SPADES:          5
```

Problem 6-2 Listing

```
10 PRINT: INPUT "STRING TO SPLIT";A$
20 L=LEN(A$)
30 H=INT(L/2)
40 PRINT MID$(A$,1,H)
50 PRINT MID$(A$,H+1,L-H)
60 GOTO 10
```

Problem 6-2 Sample Run

```
STRING TO SPLIT? MICROCOMPUTER
MICROC
OMPUTER

STRING TO SPLIT? SPLIT THIS STRING !
SPLIT THI
S STRING !
```

Problem 6-3 Listing

```
10 DIM N$(5), A$(5)
20 FOR X = 1 TO 5
30 PRINT
40 INPUT "NAME"; N$(X)
50 INPUT "AGE"; A$(X)
60 NEXT X
70 PRINT: INPUT "NAME OR AGE"; B$
80 K = 1
90 IF B$ = N$(K) THEN PRINT "AGE IS "; A$(K): GOTO 70
100 IF B$ = A$(K) THEN PRINT "NAME IS "; N$(K): GOTO 70
110 K = K + 1: IF K<6 THEN 90
120 PRINT "NO SUCH PERSON!": GOTO 70
```

Problem 6-3 Sample Run

```
NAME? WENDY
AGE? 19

NAME? BILL
AGE? 34

NAME? LOIS
AGE? 33

NAME? HENRY
AGE? 23

NAME? MARGARET
AGE? 36

NAME OR AGE? WENDY
AGE IS 19

NAME OR AGE? 23
NAME IS HENRY

NAME OR AGE? JIM
NO SUCH PERSON!

NAME OR AGE? 45
NO SUCH PERSON!
```

Problem 6-5 Listing

```
5 DIM S$(50)
10 PRINT: PRINT: PRINT "STRING TO": INPUT "SCRAMBLE";W$
20 L=LEN(W$)
30 FOR I=1 TO L
40 S$(I)=MID$(W$,I,1)
50 NEXT
60 FOR I=1 TO L
70 Z=INT(RND(1)*(L+.999))
80 IF S$(Z)="" THEN 70
90 PRINT S$(Z);: S$(Z)=""
100 NEXT
110 B$="": PRINT: PRINT: PRINT "? ";
120 FOR I=1 TO L
130 GET A$: IF A$="" THEN 130
140 PRINT A$;: B$=B$+A$
150 NEXT
160 IF B$=W$ THEN 10
170 PRINT: PRINT "TRY AGAIN": GOTO 110
```

Problem 6-6 Listing

```
100 L=(22-LEN(A$))/2
110 PRINT TAB(L) A$
120 RETURN
```

Problem 6-7 Listing

```
10 INPUT "STRING TO USE";A$
20 PRINT
30 FOR K=1 TO LEN(A$)
40 PRINT MID$(A$,1,K)
50 NEXT
60 PRINT: GOTO 10
```

Problem 6-7 Sample Run

```
STRING TO USE? ORANGE

O
OR
ORA
ORAN
ORANG
ORANGE

STRING TO USE? MICROCOMPUTER

M
MI
MIC
MICR
MICRO
MICROC
MICROCO
MICROCOM
MICROCOMP
MICROCOMPU
MICROCOMPUT
MICROCOMPUTE
MICROCOMPUTER
```

Problem 6-9 Listing

```
10 DIM S$(5)
20 FOR I=1 TO 5
30 PRINT "STRING";I;: INPUT S$(I)
40 NEXT
50 FOR I=1 TO 5
60 T$="ZZZZZZ"
70 FOR J=1 TO 5
80 IF S$(J)<T$ THEN T$=S$(J)
90 NEXT
100 FOR G=1 TO 5
110 IF S$(G)=T$ THEN S$(G)="ZZZZZZ"
120 NEXT
130 PRINT T$
150 NEXT
160 END
```

Problem 6-9 Sample Run

```
STRING 1 ?HOUSE
STRING 2 ?DOORMAT
STRING 3 ?KITCHEN
STRING 4 ?HALLWAY
STRING 5 ?WINDOW
DOORMAT
HALLWAY
HOUSE
KITCHEN
WINDOW
```

Problem 6-10 Listing

```
10 DATA "JANUARY",31
20 DATA "FEBRUARY",28
30 DATA "MARCH",31
40 DATA "APRIL",30
50 DATA "MAY",31
60 DATA "JUNE",30
70 DATA "JULY",31
80 DATA "AUGUST",31
90 DATA "SEPTEMBER",30
100 DATA "OCTOBER",31
110 DATA "NOVEMBER",30
120 DATA "DECEMBER",31
130 INPUT "DAY";D
140 READ M$,A
150 IF D>A THEN D=D-A: GOTO 140
160 PRINT M$;D: RESTORE: GOTO 130
```

Problem 6-11 Listing

```
10 PRINT: INPUT "INPUT A NUMBER";N
20 N$=STR$(N)
30 L=LEN(N$)-1
40 B$=""
50 K=0
60 B$=MID$(N$,L+1,1)+B$
70 L=L-1: IF L=0 THEN 110
80 K=K+1: IF K<3 THEN 60
90 B$=","+B$
100 GOTO 50
110 PRINT B$: GOTO 10
```

Problem 6-11 Sample Run

```
INPUT A NUMBER?  1
1

INPUT A NUMBER?  12
12

INPUT A NUMBER?  123
123

INPUT A NUMBER?  1234
1,234

INPUT A NUMBER?  12345
12,345

INPUT A NUMBER?  123456
123,456

INPUT A NUMBER?  1234567
1,234,567
```

Problem 6-13 Listing

```
10 INPUT "MESSAGE";M$
20 L=LEN(M$)
30 M$=M$+" [4 SPACES] "+M$
40 PRINT "[SHIFT] [CLEAR/HOME]";
50 I=1
60 PRINT MID$(M$,I,21)
70 FOR K=1 TO 300: NEXT
80 PRINT "[SHIFT] [CLEAR/HOME]";
90 I=I+1: IF I=L+5 THEN 40
100 GOTO 60
```

Problem 6-14 Listing

```
10 PRINT: PRINT "HOW MANY TIME DELAYS"
20 INPUT "WOULD YOU LIKE";N
30 DIM A$(N)
40 FOR I=1 TO N
50 PRINT "DELAY ";I;: INPUT B$
60 A$(I)=RIGHT$("000000"+B$,6)
70 NEXT
80 FOR I=1 TO N
90 TI$="000000"
100 IF TI$<A$(I) THEN 100
110 PRINT "TIME";I;"HAS ELAPSED"
120 NEXT
130 END
```

CHAPTER 7

Problem 7-3 Listing

```
300 PRINT "ANGLE                SIN"
305 FOR I=0 TO 350 STEP 10
310 A=(I/360)*2*3.14159
320 PRINT I,SIN(A)
330 NEXT
340 END
```

Problem 7-3 Sample Run

```
ANGLE                SIN
 0                  0
 10                  .173648033
 20                  .342019866
 30                  .499999598
 40                  .642787158
 50                  .766043969
 60                  .866024962
 70                  .939692268
 80                  .984807548
 90                  1
 100                  .984808009
 110                  .939693175
 120                  .866026288
 130                  .766045674
 140                  .64278919
 150                  .500001915
 160                  .342022359
 170                  .173650645
 180                  2.65227048E-06
 190                 -.17364542
 200                 -.342017373
 210                 -.49999732
 220                 -.642785126
 230                 -.766042265
 240                 -.866023636
 250                 -.93969136
 260                 -.984807088
 270                 -1
 280                 -.984808469
 290                 -.939694083
 300                 -.866027614
 310                 -.76604738
 320                 -.642791222
 330                 -.500004212
 340                 -.342024853
 350                 -.173653257
```

Problem 7-5 Listing

```
2000 PRINT "ANGLE              TAN                  SIN/COS"
2010 INPUT A
2020 PRINT A,
2030 A=(A/360)*2*3.14159
2040 PRINT TAN(A), SIN(A)/COS(A)
2050 GOTO 2010
```

Problem 7-5 Sample Run

```
ANGLE            TAN                 SIN/COS
 10          .176326829           .176326829
 60          1.73204727           1.73204727
 130         -1.19175823          -1.19175823
 190          .176324093           .176324093
 230          1.19174539           1.19174539
 305         -1.42816167          -1.42816166
 355         -.0874939371          -.0874939371
```

CHAPTER 8

Problem 8-1 Listing

```
10 POKE 36879,91
```

Problem 8-3 Listing

```
10 X=11: PRINT "[SHIFT] [CLEAR/HOME]"
20 D=1-INT(RND(1)*2.9999)
30 POKE 7680+X,32:   REM WIPE OUT PREVIOUS DIAMOND
40 X=D+X          :   REM CALCULATE NEW POSITION
45 REM DON'T EXCEED THE SCREEN LIMITS
50 IF X<0 THEN X=0
60 IF X>21 THEN X=21
65 REM POKE THE CHARACTER THEN THE COLOR
70 POKE 7680+X,90: POKE 38400+X,0
80 FOR J=1 TO 100: NEXT
90 GOTO 20
```

Problem 8-6 Listing

```
10 PRINT "[SHIFT] [CLEAR/HOME]"
20 X=0: Y=0
30 FOR X=0 TO 20.5
40 GOSUB 300
50 NEXT
60 FOR Y=0 TO 21.5
70 GOSUB 300
80 NEXT
90 FOR X=21 TO 0.5 STEP -1
100 GOSUB 300
110 NEXT
120 FOR Y=22 TO 0.5 STEP -1
130 GOSUB 300
140 NEXT
150 GOTO 10
300 S=7680+X+(Y*22)
310 C=38400+X+(Y*22)
320 POKE S,42
330 POKE C,2
340 FOR J=1 TO 20: NEXT
350 POKE S,32
360 RETURN
```

Problem 8-9 Listing

```
10 X=11: Y=10
20 INPUT A$
30 A=ASC(A$)-64
35 PRINT "[SHIFT] [CLEAR/HOME]";
40 P=X+Y*22
50 S=7680: C=38400
60 POKE S+P,A: POKE C+P,0
70 GET A$: IF A$="" THEN 70
80 B=ASC(A$)
90 IF B=17 THEN Y=Y+1
100 IF B=29 THEN X=X+1
110 IF B=145 THEN Y=Y-1
120 IF B=157 THEN X=X-1
130 IF X<0 THEN X=0
140 IF X>21 THEN X=21
150 IF Y<0 THEN Y=0
160 IF Y>22 THEN Y=22
170 POKE S+P,32
180 GOTO 40
```

Problem 8-11 Listing

```
10 X=11:S=0: PRINT "[SHIFT] [CLEAR/HOME]"
20 POKE 8164+X,30: POKE 38884+X,0
30 GET A$: IF A$<>"" THEN 90
40 IF S=0 THEN 30
50 POKE D+P,32
60 D=D-44: DC=DC-44
70 IF D<7680 THEN S=0: GOTO 30
80 POKE D+P,46: POKE DC+P,2: GOTO 30
90 IF A$<>"V" THEN 120
100 POKE 8164+X,32: X=X-1: IF X<0 THEN X=0
110 GOTO 140
120 IF A$<>"N" THEN 150
130 POKE 8164+X,32: X=X+1: IF X>21 THEN X=21
140 POKE 8164+X,30: POKE 38884+X,0: GOTO 40
150 IF S=1 THEN 30
160 P=X: S=1: D=8142: DC=38862: GOTO 50
```

Problem 8-13 Listing

```
10 PRINT "[SHIFT] [CLEAR/HOME]";: X=0
20 Y=INT(RND(1)*21)          :REM GET RANDOM Y POSITION
30 FOR X=0 TO 21             :REM X POSITION
40 POKE 7680+X-1+(Y*22),32 :REM CLEAR PREVIOUS ASTERISK
50 POKE 7680+X+(Y*22),42   :REM POKE THE ASTERISK
60 POKE 38400+X+(Y*22),0   :REM POKE THE COLOR (BLACK)
70 FOR J=1 TO 30: NEXT J   :REM WAIT A LITTLE
80 NEXT                      :REM GENERATE A NEW X POSITION
90 GOTO 10                   :REM DO ANOTHER LINE
```

CHAPTER 9

Problem 9-1 Listing

```
10 GET A$: IF A$ = "" THEN 10
20 A = VAL(A$) + 1
30 FOR K = 1 TO A: READ T: NEXT
40 POKE 36874,T: POKE 36878,8
50 RESTORE: GOTO 10
60 DATA 135, 140, 150, 160, 170
70 DATA 180, 190, 200, 210, 220
```

Problem 9-3 Listing

```
10 DIM A(50): K=0
20 INPUT "SOUND VALUE";X
30 IF X=99 THEN 50
40 A(K)=X: K=K+1: GOTO 20
50 FOR J=0 TO K
60 POKE 36878,8
70 POKE 36876,A(J)
80 FOR L=0 TO 100: NEXT
90 NEXT
100 GOTO 50
```

Problem 9-6 Listing

```
10 S=128+(RND(1)*127)
20 D=RND(1)*1020
30 POKE 36878,8
40 POKE 36875,S
50 FOR J=0 TO D: NEXT
60 GOTO 10
```

References

INTERESTING MAGAZINES FOR VIC 20 OWNERS

Commodore–The Microcomputer Magazine. 1200 Wilson Drive, West Chester, PA 19380. Specific to the Commodore line of computers—VIC 20, Commodore 64, and PET. Adult level.

Power/Play–The Home Computer Magazine. 1200 Wilson Drive, West Chester, PA 19380. Specific to the Commodore line of computers. Aimed at the individual user who wants programs to run. Also contains general articles. Adult level.

Compute! P.O. Box 5406, Greensboro, NC 27403. General computer magazine that covers most popular small computers. Contains programs, articles, columns, and reviews. High school through adult level.

Creative Computing. P.O. Box 5214, Boulder, CO 80321. General computer magazine that contains mainly recreational and game programs, plus reviews. High school through adult level.

BUSINESS-RELATED BOOKS

Barden, William Jr. *What Do You Do After You Plug It In?* Indianapolis: Howard W. Sams & Co., Inc., 1983.

Beasley, Jack O. *Microcomputers on the Farm.* Indianapolis: Howard W. Sams & Co., Inc., 1983.

Blumenthal, Susan. *Understanding and Buying a Small-Business Computer.* Indianapolis: Howard W. Sams & Co., Inc., 1982.

Brooner, E. G. *Basic Business Software.* Indianapolis: Howard W. Sams & Co., Inc. 1980.

Brooner, E. G. *Microcomputer Data-Base Management.* Indianapolis: Howard W. Sams & Co., Inc., 1982.

Digital Equipment Corp. *Guide to Personal Computing.* Maynard, MA: Digital Equipment Corp., 1982.

Eischen, Martha. *Compuguide.* Beaverton, OR: Dilithium Press, 1982.

Jong, Steven F. *Word Processing for Small Businesses.* Indianapolis: Howard W. Sams & Co., Inc., 1983.

Index

A

Action keys, 23-32
AND operation, 108-109
Arithmetic operations, 67-82
  combining operations, 78-80
  inputting values, 73-75
  multiple operations on a single line, 77-78
  order of precedence, 79-81
  setting up the problem, 68-69
  use of parentheses, 80-82
  using labels, 72-73
  using messages, 74
Array of information, 145
  DATA and READ commands, 153
  size of, 151-152
  use of, 150-151

B

BASIC, 43
Basic VIC 20 system, 17-20
Boole, George, 111
Boolean algebra, 111
Branching program, 89, 91

C

Cable assembly, 17, 19-20
Cassette unit, 261-274
  GET command, 271-273
  getting data, 267-269
  LOAD, 262
  SAVE, 262
  saving information, 263-267
  VERIFY, 262
Clearing the screen, 26-27
CLR/HOME key, 25-28
Codes, ASCII, 237
Color keys, 35-37
Colors, changing on screen, 217-220
  controls, 219-220
    symbols, 218
  CTRL key, 217-218
  reversing characters, 218
Commands, 44
Comments, use of in programs, 82-83
CONT command, 94

CTRL key, 36-38
Cursor, 20
  action keys, 25-28

D

DATA and READ commands, use of 153-155
Deleting characters, 30-32
Displays and graphics, advanced, 222-238
  characters, second set of, 236
  flashing sign, 232-235
  filling the TV screen, 223-230
  PEEK command, 235-236
  POKE command, 230-232
  reversed characters, 232
  screens and borders, 236-237
Drawing pictures, 32-35

E

END command, 105
Entering a program, 45-47

F

FOR-NEXT Command, 131-132
  counting loops, 131-132, 141

G

GET command, 174-178
  building a string, 176-177
  getting a few characters, 177-178
  use of, 175-176
  with cassette, 271-274
Getting data from a cassette, 267-269
GOTO command, 89-94
Graphic keys, 32-35
Graphic printer, 274-278
  control codes, 278
Graphic printer—cont
  starting, 275-278
    secondary address, 275

H

Hardware care, 283-295
  cleaning, 288-289
  positioning, 284
  power, 285-288
  protection, 285
  static, 285
  troubleshooting, 289-290

I

IF-THEN
  command, 128
  operations, 100-110
Information storage, 147-150
INPUT command, 44, 73-75
INPUT, unusual characters, 166
Input/output devices, 43
Inputting strings, 57-58
Inserting characters, 30-32
INST/DEL key, 28-32, 83-85
INT operation, 114-115
  precautions, 115-116

J

Joystick controls, 279-281

K

Keyboard, 20-21

L

Labels, 50-59

Labels —cont
in math problems, 72-73
LEFT$ and RIGHT$ operations, use of in strings, 170-171
Line numbers, 44
LIST command, 45
Listing a program, 45-47
Logarithms, 203-205
Logical operations, 108-111
AND, 108, 109
NOT, 111
OR, 108-109
Logs, VIC 20, 205-206
Loop
nested, 136-139
use of, 135-136
average of several values, 135
number of values used, 135
Loop counter, 131
Loops, 94-98
IF-THEN, 100-110

M

Messages in INPUT and PRINT commands, 59-63
Mixed labels, 58-59
Moving the cursor, 27-28

N

NEW command, 69
Noise, adding to program, 255-257
NOT operation, 111

O

ON command, 128-131
On/off power switch, 20
OR operation, 108-109

P

Parentheses, in math operations, 80-82
PEEK command, 235-236
PI, value of, 190-193
use of, 193
POKE command, 230-234
Positioning numbers on the screen, 125
Power
module, 17-19
switch, 20
PRINT command, 44
Printing in columns and rows, 126
Printing keys, 21-23
Printing messages in INPUT and PRINT commands, 59-63
Program lines, multiple operations on, 77-78
Program and memory cartridges, 259-261

R

Random numbers 112-115
REM statement, 82
Remarks, use of in programs, 82-83
RESTORE key, 95-96
RETURN key, 45
Reverse printing, 37-38
Reversed printing, 221-222
RND(1) operation, 112-115
RUN command, 47
Running a program, 47-49
RUN/STOP key, 94-98
RVS OFF key, 38
RVS ON key, 38

S

Saving information on a cassette, 263-267
Saving strings on a tape, 269-271
Screens and borders, 236-237
SHIFT key, 23-26
SHIFT LOCK key, 25

Software selection, 291-295
demonstrations and documentation, 292
flexibility of programs, 295
information and references, 291-292
protection, 295
testing programs, 293-294
updates and revisions, 294-295
whats available, 291
Sound from the VIC 20, 244-257
changing the speakers, 251-252
changing the tone, 249-251
musical tone codes, 246
noise, 255-257
setting the volume, 245-248
use in a program, 252-255
SPC command, 212-216
and TAB operations, 212-213
moving the cursor, 213-216
Special codes, 206-208
ASCII, 207
value, 208
Special symbols, 124-125
Specialty printing, 122-124
clearing the screen, 123-124
Square root, 187-190
calculations, 190
Pythagorean Theorem, 188
Straight-line program, 89, 91
Strings, 54-58, 160-186
building of, 164-165
GET command, 174
middle, MID$, 173
untangling, 169-170
use of, 161-164
decisions regarding, 167-169
values, 180-182
Subroutines, 142-147
use of, 143, 144-146
Subscripts, arrays, 151
Supplies of VIC-20 cartridges and cassettes, 282
Switch box, TV/computer, 19-20

T

TAB command, 126-128
use of, 127-128
Tables, information storage, 148
array of information, 149
labels, 149
subscripts, 151
use of, 150-151
TAN "f" keys, 208-209
Telling time, 178-180
real-time clock, 178
using VIC 20's clock, 179-180
Testing a value, ON command, 128-131
Trigonometry, 193-202
VIC 20, 197-198
using trig functions, 198-203
TV/Computer switch box, 19-20

U

Untangling strings, 169-170
length, 169-170
Using sound in a program, 252-255

V

Values, 54-57
and strings, 181
VIC 20 labels, 52-54

W

Word-guessing program, 171-173

The Blacksburg Group

According to Business Week magazine (Technology July 6, 1976) large scale integrated circuits or LSI "chips" are creating a second industrial revolution that will quickly involve us all. The speed of the developments in this area is breathtaking and it becomes more and more difficult to keep up with the rapid advances that are being made. It is also becoming difficult for newcomers to "get on board."

It has been our objective, as the Blacksburg Group, to develop timely and effective educational materials that will permit students, engineers, scientists, technicians and others to quickly learn how to use new technologies and electronic techniques. We continue to do this through several means, textbooks, short courses, seminars and through the development of special electronic devices and training aids.

Our group members make their home in Blacksburg, found in the Appalachian Mountains of southwestern Virginia. While we didn't actively start our group collaboration until the Spring of 1974, members of our group have been involved in digital electronics, minicomputers and microcomputers for some time.

Some of our past experiences and on-going efforts include the following:

—The design and development of what is considered to be the first popular hobbyist computer. The Mark-8 was featured in Radio-Electronics magazine in 1974. We have also designed several 8080-based computers, including the MMD-1 system. Our most recent computer is an 8085-based computer for educational use, and for use in small controllers.

—The Blacksburg Continuing Education Series™ covers subjects ranging from basic electronics through microcomputers, operational amplifiers, and active filters. Test experiments and examples have been provided in each book. We are strong believers in the use of detailed experiments and examples to reinforce basic concepts. This series originally started as our Bugbook series and many titles are now being translated into Chinese, Japanese, German and Italian.

—We have pioneered the use of small, self-contained computers in hands-on courses for microcomputer users. Many of our designs have evolved into commercial products that are marketed by E&L Instruments and PACCOM, and available from Group Technology, Ltd., Check, VA 24072.

—Our short courses and seminar programs have been presented throughout the world. Programs are offered by the Blacksburg Group, and by the Virginia Polytechnic Institute Extension Division. Each series of courses provides hands-on experience with real computers and electronic devices. Courses and seminars are provided on a regular basis, and are also provided for groups, companies and schools at a site of their choosing. We are strong believers in practical laboratory exercises, so much time is spent working with electronic equipment, computers and circuits.

Additional information may be obtained from Dr. Chris Titus, the Blacksburg Group, Inc. (703) 951-9030 or from Dr. Linda Leffel, Virginia Tech Continuing Education Center (703) 961-5241.

Our group members are Mr. David G. Larsen, who is on the faculty of the Department of Chemistry at Virgina Tech, and Drs. Jon Titus and Chris Titus who work full-time with the Blacksburg Group, all of Blacksburg, VA.

SAMS

VIC 20™ Starter Book

- An interesting and enjoyable introduction to programming and understanding the Commodore VIC 20 computer.
- Written especially for the first-time computer user who wants to learn quickly about computer programming and what the VIC 20 can do.
- Avoids the highly technical details and terminology that only tend to confuse the novice programmer.
- Contains many programming experiments to help the reader learn about the VIC 20 and to reinforce the material being discussed.
- Includes questions and programming problems for readers to try and solve on their own—answers to questions and solutions to selected problems are provided at the end of the book.
- Covers accessories for the VIC 20, including the cassette tape unit, the graphics printer, the joystick, and plug-in program and memory cartridges. Also discusses proper care of the computer and how to intelligently evaluate and select packaged programs or "software."

ward W. Sams & Co., Inc.
st 62nd Street, Indianapolis, Indiana 46268 U.S.A.

ISBN: 0-672-22258-2