

adding power to your **commodore 64**



Vahé Guzelimian
Steve Cates

Adding Power to Your Commodore 64®

**Vahé Guzelimian
Steve Cates**

A Reston Computer Group Book
Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data

Guzelimian, Vahé.

Adding power to your Commodore 64.

I. Commodore 64 (Computer) I. Cates, Steve. II. Title.
QA76.8.C64G88 1984 001.64 84-4724
ISBN 0-8359-0060-6

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The authors have taken every precaution to ensure the effectiveness of the material in the book. The publisher and authors make no expressed or implied warranty of any kind pertaining to the programs in the book or on the diskette. The authors and publisher will not be liable for any form of damage associated with the use of these materials.

Apple® is a registered trademark of Apple Computer, Inc., Sunnyvale, California.

C-64™, CBM™, and Datassette™ are trademarks of Commodore Business Machines, Inc.
Commodore 64® and PET® are registered trademarks of Commodore Business Machines, Inc.
The Connection™ is a trademark of Tymac Controls, Inc.

IBM® is a registered trademark of International Business Machines Corporation.

© 1985 Reston Publishing Company, Inc.

A Prentice-Hall Company

Reston, Virginia 22090

All rights reserved. No part of this book may be reproduced, in any way or by any means, without permission in writing from the publisher.

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

CONTENTS

Preface, vii

PART ONE INTRODUCTION TO THE COMMODORE 64 COMPUTER

Chapter 1: The Commodore 64 Computer, 3

- A Computer in Every Home, 3
- Low Prices, 3
- The Microcomputer, 4
- Building the C-64 System, 6
- Computers That Listen and Speak, 9

Chapter 2: Programs and Languages, 11

- What is a Program?, 11
- Types of Software, 12
- Computer Languages, 14

Chapter 3: Computer Memory, 19

- ROM and RAM Chips, 19
- Bits, Bytes, Nibbles, and Kilobytes, 20

PART TWO THE COMPONENTS OF THE C-64 COMPUTER SYSTEM

Chapter 4: Setting Up the Computer, 25

- The Computer Work Area, 25
- The Computer, 26

Chapter 5: The Video Screen, 33

- Regular TV, 33
- Video Monitor, 36

Chapter 6: The Keyboard, 41

- Text Keys, 41
- Numeric Keys, 42
- Operation Keys, 43
- Symbol Keys, 43
- Graphic Keys, 43
- Color Keys, 45
- Function Keys, 47
- Special Control Keys, 47
- Editing Keys, 49

Chapter 7: The Tape Drive, 53

- Putting Programs Into Your Computer, 53
- Using the Datasette, 54
- Tape Tips, 57
- One Final Tip, 59

Chapter 8: The Disk Drive and Diskettes, 61

- The 1541 Disk Drive, 61
- Connecting the 1541 Disk Drive, 62
- Using Two Disk Drives, 64
- Disk Operating System (DOS), 65
- Floppy Diskettes, 66
- Files on Diskettes, 70
- Floppy Tips, 72
- Loading Programs, 74
- Saving Programs, 75
- Other Disk Commands, 76
- Disk Short Cuts, 79
- If Your Drive Isn't Performing Properly, 81
- The New 1541 Disk Drives, 82
- Care of the Disk Drive, 82

Chapter 9: The Printer, 85

- The Importance of the Printer, 85
- Types of Printers, 85
- Printer Characteristics, 87
- Connecting the Printer, 91
- Compatibility, 92

Chapter 10: The Modem and Telecomputing, 93

- The Future is Here, 93
- Access to the Biggest Computers, 93
- The Modem, 94
- Terminal Software, 95
- Commodore Modems, 98

PART THREE OPERATING THE C-64

Chapter 11: Using the C-64 in Direct Mode, 105

- What is Direct Mode?, 105
- Displays on the Screen, 106
- The Print Statement, 107
- Using Variables in Direct Mode, 109
- Cursor Control in Direct Mode, 111
- Graphics in Direct Mode, 112
- Other Direct Mode Functions, 113
- A Reminder About Color, 113

Chapter 12: Using the C-64 in Program Mode, 115

- Some Program Basics, 115
- Creating a Program from a Screen Display, 116
- The Screen Editor, 118
- Using Quote Mode in Programs, 127
- Your Name in Lights—A Demonstration, 129

Chapter 13: Using Your Printer, 133

- A Word of Caution, 133
- Non-Commodore Printers, 133
- Testing the Printer, 134
- Printer Commands, 136

Chapter 14: Using Sprites, 145

- Introduction to Sprites, 145
- Using the Sprite Editor, 147
- Sprites to Data Statements, 151
- Sprites in Programs, 153

Chapter 15: Creating Sounds with the C-64, 163

- Sound Capabilities of the C-64, 163
- The Sound Interface Device (SID), 163
- Time to Experiment, 171
- Multiple Voices, 171
- Advanced Sound Features, 172

PART FOUR ADVANCED SECTION

Chapter 16: File Handling, 175

- The Importance of Files, 175
- What is a File?, 175
- Program Files, 176
- Sequential Files, 176
- Relative Files, 183

Chapter 17: Machine Language, 191

- Machine Language through BASIC, 191
- Machine Code, 198
- Combining Machine Language and BASIC Programs, 205
- The Next Step, 211

PART FIVE POWER PLUS: A UTILITY PROGRAM

Chapter 18: Introduction to Power Plus, 215

- POWER PLUS Utilities, 215
- Loading and Running POWER PLUS, 216

Chapter 19: SUPERDOS, 217

- General Commands, 217
- Load Options, 220
- Save Options, 221
- Sending Commands to the Disk, 221
- SUPERDOS Quick Reference, 224

Chapter 20: SUPERBASIC, 227

- AUTO, 227
- DELETE, 229
- NUMBER, 229
- FIND, 231
- CHANGE, 233
- KILL, 234
- SUPERBASIC Quick Reference, 235

Chapter 21: SUPERMON, 237

- Uses of SUPERMON, 237
- SUPERMON Commands, 238
- SUPERMON Quick Reference, 244

PART SIX PROGRAMMING TRICKS AND PROGRAM DESCRIPTIONS

Chapter 22: Programming Aids, 247

- Appending Programs, 247
- Blank Screen, 250
- Character Switch, 251
- The Colon, 251
- Disable Commands, 254
- The Function Keys, 256
- The INPUT and GET Commands, 256
- Using the Joysticks, 258
- REM Statements, 260
- Random Numbers, 260
- System Reset, 261
- Time Clocks, 264

Chapter 23: Program Descriptions, 267

- Addresses, 267
- Backup 1541, 268
- Cataloger, 270
- Color POKEs, 271
- Data Writer, 272
- Disk Pac, 273
- Hex/Dec Convert, 276
- Kernel Revisions, 276

PART SEVEN APPENDICES, 281

Glossary, 315

Index, 327

Preface

Adding Power to Your Commodore 64 is intended for users with all levels of computing experience, from the new user who just bought the Commodore 64, to the user who has experience writing programs. The book starts with the fundamentals, and works up to intermediate and advanced levels. It's a book that you grow with. As you learn more about the C-64 by trying the various examples, you will be ready to tackle more advanced concepts.

To make it easier for you to understand the meaning of technical terms, we have included a Glossary. Each word in the Glossary is italicized when it is first used in the text.

For those of you who are both advanced in computing, and ambitious, we've included information in boxes that deal with more complex issues. Some of these boxes outline how to make hardware changes to your computer system.

The tutorial style will teach you, step-by-step, how to use the disk drive, design sprites, use sound, and program in machine language.

Adding Power to Your Commodore 64 may be purchased either with or without the support disk. This disk is not simply ready-to-run programs that repeat those that appear as program listings in the book: it contains additional valuable utility programs, some written in machine language. All of these programs are designed to make it easier for you to use your computer system, and to add more power to the C-64.

If you purchased this book without the support disk, you may obtain it by using the order form included in the book.

Acknowledgments

Many people have directly or indirectly contributed to the creation of this book, *Adding Power to Your Commodore 64*. Our students taught us that nothing can be ambiguous when you teach someone how to use a computer. They also helped us find gaps in the information they were acquiring from other sources.

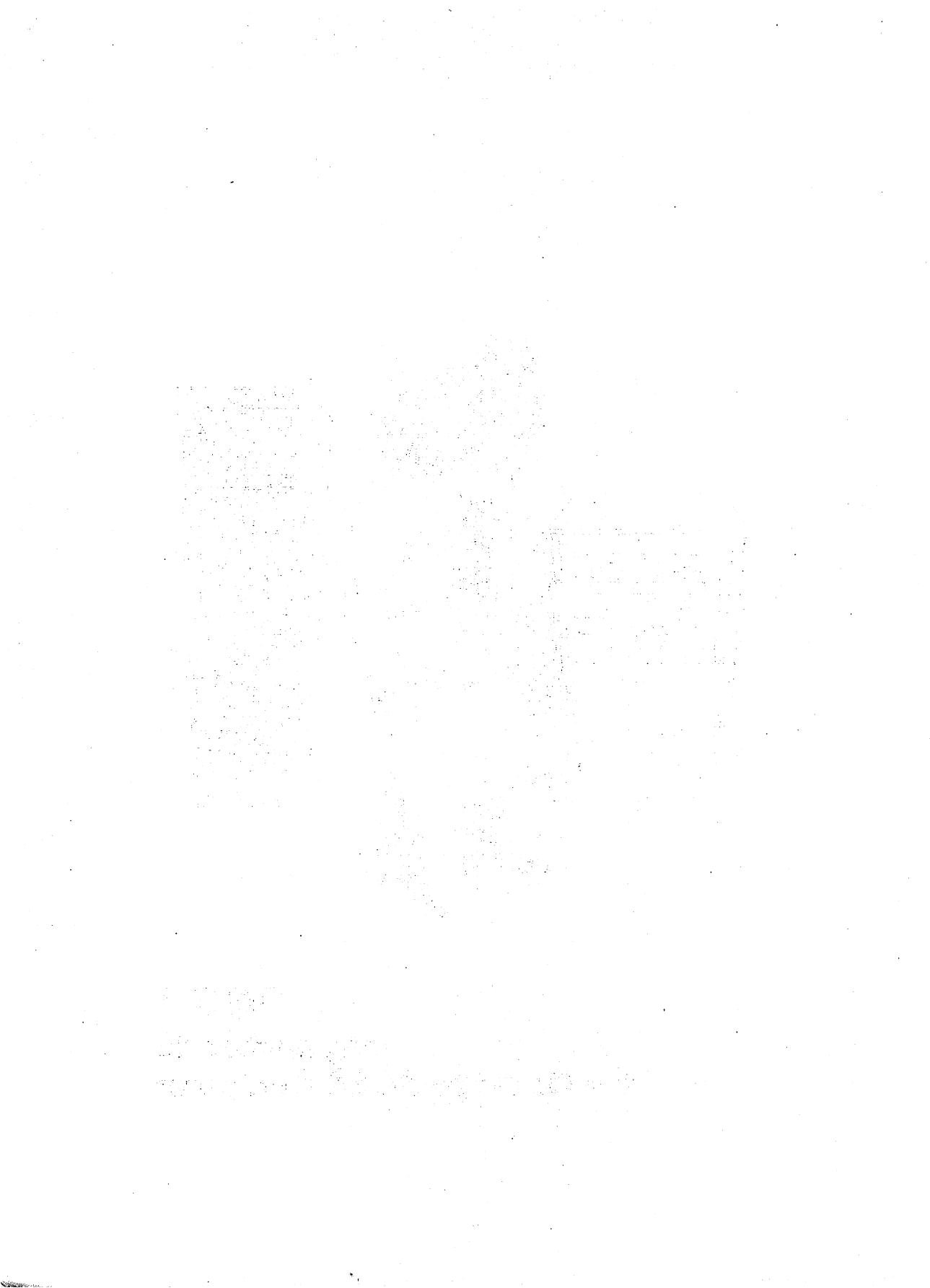
Jim Butterfield and Jim Strasma have, through their generous contribution of information to Commodore users, taught us a great deal about these wonderful machines.

Linnea Dayton provided valuable editorial advice. The San Diego PET User's Group kept us in touch with C-64 users and their needs. Dona Meilach had faith in our efforts from the start and gave us valuable advice throughout the preparation of the book.



PART 1

Introduction to the Commodore 64 Computer



The Commodore 64 Computer

George Orwell predicted that 1984 would mark the beginning of an era of increasing technological power. He predicted that this power would be in the hands of dictators seeking to control the populace. But in fact, all indicators suggest that with more sales of home computers, a great deal of technological power will be in the hands of the individual. The ever-increasing number of home computer enthusiasts, or "hackers," who have managed to break into large computers through the telephone system is supportive evidence that the power cannot be monopolized. Orwell didn't envision that the tables would be turned in this way.

Commodore Business Machines (CBM) has contributed a great deal to this technological revolution by marketing the most versatile, yet affordable home computers for the general public. CBM is currently selling three million Commodore 64 (C-64) computers each year. Computing power will soon be in the hands of virtually everyone.

Like higher-priced computers, the C-64 is useful both for entertainment and for business. You can manage your business with it by day, your children can use it for their lessons in the afternoon, and your family can enjoy it in the evening with educational or arcade-style games.

Programs, or *software*, are becoming more numerous as software publishers scramble to meet the high demand for C-64 programs. Many of these publishers are converting programs written for Apple and IBM computers to run on the C-64. Commodore has announced its plans to accelerate its involvement in software production. CBM wants to revolutionize the software market just as it revolutionized the home computer market. It intends to make powerful programs for the C-64 available at low prices. Software available at prices over \$200 for other machines will be under \$75 for the C-64. This is good news for you, the owner of a C-64.

**A COMPUTER
IN EVERY
HOME**

LOW PRICES

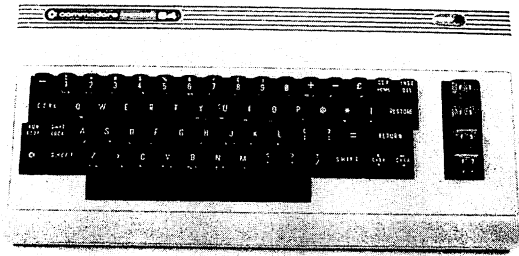


FIGURE 1-1. The Commodore 64 Computer.
(Courtesy of Commodore Business Machines, Inc.)

The C-64 (Figure 1-1) is a *microcomputer*, a term used to describe any small computer with its “brains” or computing power contained on one chip, the *microprocessor chip*.

THE MICRO-COMPUTER

Over 10,000 electronic circuits, the equivalent of a room-full of yesterday’s expensive apparatus, are now etched onto a single chip the size of a thumbnail! This chip (see Figure 1-2) carries out the arithmetic, logic, and control functions of the computer by moving *data* around and performing operations such as addition and subtraction. The microprocessor chip is the *central processing unit (CPU)* of the computer. Like the brain in the human body, its role is to control and regulate all of the parts that make up the whole unit.

The microprocessor chip is the most important part of the computer. A technician can tell much about a computer by learning which microprocessor it utilizes, because the microprocessor determines the character and power of the computer. The C-64 uses the 6510 microprocessor, manufactured by MOS Technology, a subsidiary of CBM. In most aspects, the 6510 is identical to the 6502 chips that are used in Apple and Atari home computers.

The Microprocessor Chip

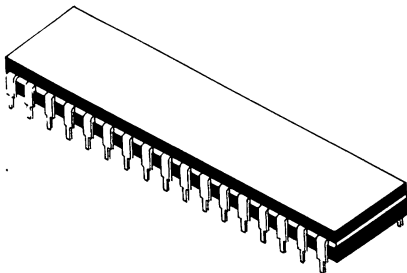


FIGURE 1-2. The microprocessor chip.

The original C-64 computers contained over 30 chips on a single board. Newer C-64s use fewer chips without any loss of power, yet with a reduction in manufacturing cost. It is safe to assume that in a few years the entire capability of today's C-64 computer will exist on a single microprocessor chip. Since microprocessor chips are inexpensive to produce once research and development have been paid for through sales, computer prices should continue to drop even as the computer's power increases.

A recent item from *The Washington Star* dramatizes the development of the computer: "Had the automobile developed at a pace equivalent to that of the computer during the last 20 years, today a Rolls Royce would cost less than \$3, get three million miles to the gallon, deliver enough power to drive the Queen Elizabeth II, and six of them would fit on the head of a pin!"

Fewer and Fewer Chips

The microcomputer is a machine that can perform both mathematical and logical operations at high speed. It can compare items of data and then automatically make decisions dependent on the outcome of the comparison, so we don't have to carry out these operations manually. Just as we take data such as numbers, perform calculations on them, and then write down our answers, the computer takes data in the form of *alphanumeric input* (letters and numbers), *processes*, or performs calculations on this data, and then presents the result as *output*. A study of Figure 1-3 will clarify the relationships between data, processing, and output.

The Work of Microcomputers

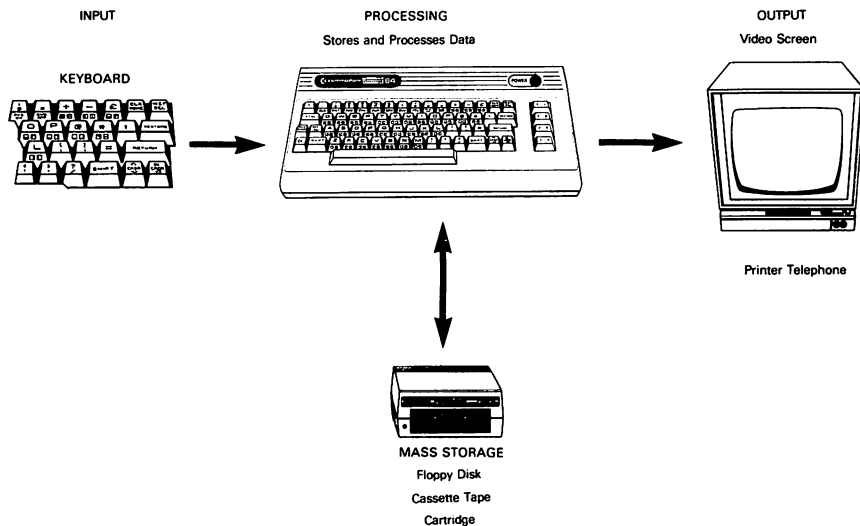


FIGURE 1-3. The components of the Commodore 64 system.

The microprocessor chip cannot carry out its operations alone. Just as our brain requires the use of systems such as vision and speech to interact with the world, so also the computer requires a link with us, the computer users. To achieve this link, the computer must be *interfaced* (connected) with devices that serve as its “eyes” and “voice.” With these *peripheral*, or external, devices, the computer can receive input data and provide output to the user. The computer receives this data through a set of wires called a *bus*, and this data is then stored in the memory chips of the computer. You can think of these memory locations, or memory *registers*, as post office boxes, each with its own *address*. By keeping track of which box each piece of data was put into, the computer can retrieve the information from that box whenever it is needed. Once the microprocessor chip has data to work on, it can perform the required calculations on this data. Interfacing peripheral devices with the computer itself will give us a complete computer system.

In this chapter, we examine the main components of the C-64 system. More detailed discussions of each of these components will follow in Part 2.

BUILDING THE C-64 SYSTEM

The keyboard is a device used to enter data into the computer. The C-64 keyboard is somewhat similar to a standard typewriter keyboard. With it you can type words, numbers, and graphic characters.

Because the keyboard shell of the C-64 houses the chips of the computer, it is easy to confuse the keyboard with the CPU itself. The keyboard is a peripheral device just like the TV or monitor you attach to the computer. Many computer manufacturers design their systems so that the circuit board containing the chips is entirely separate from the keyboard. A cord is used to attach the keyboard to the computer, just as you attach your TV or monitor to the C-64. The portable version of the C-64 (Figure 1-4) is configured in this way, with its keyboard separate from the circuit board.

The Keyboard

After thinking about something, we usually want to tell someone about the ideas we’ve come up with. The computer’s “voice,” or means of communication with the user, is usually the video screen—either a regular TV or a *CRT* (*cathode ray tube*) monitor. The screen displays the data that we type into the computer for processing as well as the results that the computer comes up with after processing the data.

The Video Screen



FIGURE 1-4. The Commodore SX-64 computer.
(Courtesy of Commodore Business Machines, Inc.)

The computer has a limited memory, just as we have a limited ability to keep long lists of data, such as names and phone numbers, in our heads. We normally keep long lists in written form in books or files. In a similar way, data such as programs or names and addresses can be stored by the computer on magnetic media—cassette tapes or *floppy disks*. To achieve such mass storage of data, we need to interface either a cassette recorder or a disk drive with the computer.

With such devices, an unlimited amount of data may be stored outside the computer. Mass storage media provide compact storage for large quantities of information. For example, a single floppy diskette for the C-64 can store the equivalent of over 120 typewritten pages of information. Using magnetic storage media rather than print on paper, we can store the information that would fill an entire building of archives in an area the size of a small room.

A video screen can display only a limited amount of data at one time. Because old information is lost each time the screen is updated with new information, we may want to make a *hard copy*, or permanent record, of our results. This may be accomplished by connecting a *printer* to the computer. Much like a typewriter, the printer can give us

Mass Storage

The Printer

neat, permanent records of the results. For word processing, a printer is a must.

A number of *controllers* are available as peripheral devices for the C-64 (see Figure 1-5). They permit you to direct the action of an arcade game, or to give commands to the computer, without using a keyboard. The *joystick* is the familiar game controller used in video arcades. Joysticks can direct motion in eight directions. *Game-paddles* are available in pairs. They provide smooth control of left and right directions. *Track balls* give you the freedom to control movement in any direction by rolling a ball in that direction. Of course, each of these devices has one or two fire buttons with which you can eliminate the “aliens” or other creatures that populate computer games.

Like the trackball, the *mouse* device allows the user to move a character on the screen in any direction. The user moves the mouse on a smooth surface, and the character on the screen copies the movement.

Joysticks, Paddles, Track Balls and the Mouse

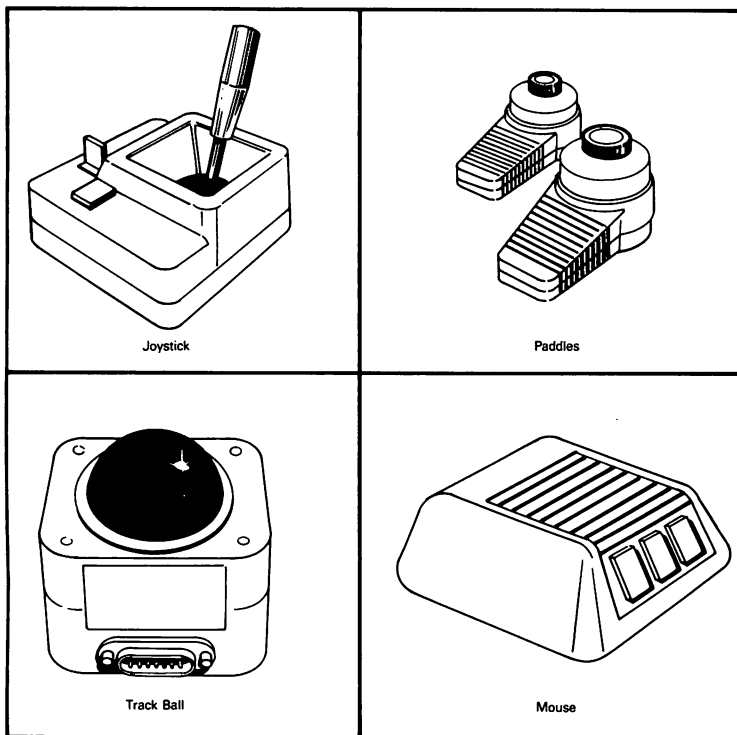


FIGURE 1-5. Game controllers.

A light pen is a peripheral device shaped like a pen, or stylus. It senses light from a point of contact on the screen and transmits this information to the computer as input. To use the light pen for drawing on the screen or to perform other tasks such as inputting your response to a question from the computer, you need a program written expressly for this purpose.

The Light Pen

When used with a drawing program, the light pen works like this: As you move the pen on the screen, the screen and pen act as an input device to tell the computer where the pen is located. The computer processes this information, keeping track of where the pen moves, and sends output to the screen so that the screen displays the form drawn by the pen. The screen is acting as both an input and an output device. The sensing, processing, and displaying all occur continuously and quickly, giving the user the illusion of drawing directly on the video screen.

Besides being used for drawing, light pens also make it easier for the user to enter information into the computer. Instead of typing on the keyboard, a user can simply touch the pen to specific parts of the video screen in order to select options. The light pen can be an especially useful tool for disabled persons who might have trouble with the keyboard.

Probably the most exciting device we can interface with a home computer is the telephone modem. Still at an early stage of development, the modem, or telephone-to-computer connection, makes it possible for users to access many sources of information. The modem can put a C-64 user, sitting at home, in contact with very large, central computers. These computers contain *databases* (stores of information) that are much bigger than any single user is likely to have at home.

The Modem

The telephone modem allows the user to access information that is constantly updated. For example, you can obtain court records, prices of stocks and commodities, weather information, and information from the best encyclopedias, all updated regularly. A relatively inexpensive computer such as the C-64 with a modem and a telephone can bring into your home most of the information available to mankind.

Many software manufacturers, including CBM, are developing new products to make it easier for users to input data into their home computers. Most home computers allow users to run programs that use devices such as a joystick and a mouse to give commands to the computer. These are the first steps toward the elimination of the keyboard as a primary input device. Not too far in the future, the keyboard may become obsolete. We will be able to speak to our computers, and our spoken words will provide the data input. Imagine a

COMPUTERS THAT LISTEN AND SPEAK

word processor that stores what you say and displays the text for you on the video screen for editing. Your computer will be able to take dictation as well as act as a word processor!

With a speech synthesizer, the C-64 can even talk to you! This device can make it possible for children who can't read and for those who are vision-impaired to receive output from the computer.

As computers become more sophisticated, they will take on more human characteristics. True creative thinking, however, will remain strictly a human trait.

Programs and Languages

The microprocessor chip, or “brain” of the computer, is part of the hardware of your computer system. Hardware is the physical part of computing—the circuit board with all the chips, the disk drive, the printer, and so on.

But the hardware is not capable of carrying out sophisticated functions on its own; something else is necessary. The computer system needs a program to direct its brain and its appendages, or peripherals, as an orchestra needs a conductor with a score to take charge of it. A program is an organized set of directions that the computer must follow to accomplish some specific task. It is the program that controls and shapes the power of the computer toward a specific goal. One program may turn the computer into an accountant, while another may turn it into a property manager, or a word processor.

The simple calculators that we are all familiar with are small computers that can carry out mathematical operations. A chip inside each calculator contains a set of instructions that enables the calculator to carry out these mathematical computations. This built-in program determines what tasks the calculator can perform. Most of the smaller calculators that we use are devoted to simple mathematics. They are not programmable; that is, you cannot alter the calculator’s program or put in a new one.

Home computers such as the C-64, on the other hand, offer us the opportunity to program them to carry out a large variety of different operations. This is what makes the computer a highly versatile tool. A computer without a program is like an orchestra without a musical score. It has the potential to do work for us, but does not know what we want it to do. The program gives the computer a set of instructions or commands that are carefully worded and organized to tell it how to successfully carry out some intended task.

WHAT IS A PROGRAM?

Programs for computers may be divided into two broad classifications, *applications software* and *systems software*. Applications software applies the computer to some specific task. Applications programs are those you buy so you can play a game, use a spreadsheet, or type text into a word processor. Systems software, on the other hand, is usually built into the computer and not a program that you purchase separately.

Computer manufacturers usually build systems software into the computer before they sell it to you. Whenever the computer is on, the systems software is working, even when an applications program is also running. Since we interact more directly with applications programs, such as a game we are playing, we often take the systems software for granted, forgetting that it's always there to do its work. Like the maintenance man who works behind the scenes to keep an office building running, the systems software is invisible to us. Systems software performs housekeeping and maintenance operations while the computer is doing its work for you. It also watches a clock to make sure that every component of the computer is working according to schedule.

The most important part of the systems software of a computer is a set of programs called the *operating system*. The operating system takes care of the input and output functions, manages all the memory locations of the computer, keeps track of the characters you type on the keyboard, and displays the characters on the screen, to mention just a few of its functions.

Given the right program, a computer can perform almost any task. Fortunately for you the user, many software manufacturers are producing excellent programs for the C-64 at low prices. Here are brief descriptions of the various types of applications programs available for the C-64.

The C-64's graphics capabilities are among the finest available. The C-64 was designed to make it easy for you to create and animate *sprites*, or movable object blocks, for games or video demonstrations. The sound capabilities of the C-64 are the best of any home computer. You can create almost any sound pattern, from footsteps to songs. These graphics and sound features make the C-64 an excellent game machine. It has the potential to duplicate most arcade-style games for your use at home.

TYPES OF SOFTWARE

Systems Software

Applications Software

Game Programs

Programs for the C-64 computer can turn it into a financial planner, an accountant, a payroll controller, a property manager, a filing system, a word processor, and much more. Your C-64 can be very helpful in running your small business. You can do payroll and have checks and W-2 forms filled out by the computer, prepare advertising flyers, send out form letters to your clients, schedule your business activities for the future, and carry out “if/then” calculations to determine how to make your business run more efficiently.

Business Programs

Your computer can be programmed to provide tutorial instruction on a variety of subjects—reading, math, history, geography, and computer education, to mention a few. Many teaching programs offer children a fun way to learn. For example, at least one typing tutor program has been designed to be both a game and an effective learning tool. The student keeps from being “zapped” by aliens by typing quickly and accurately.

Educational Programs

Educators have found that under certain circumstances, computers can be tutors for their students. Here are several of the reasons they give for the continuing use of computers in our schools:

1. Computer instruction gives their students the freedom to work at their own pace;
2. The computer can reward good answers and increase the confidence of students as they answer correctly.
3. The computer gives feedback on the student’s progress immediately; and
4. The computer is the most patient of all teachers. It can be programmed to wait as long as necessary for the answer.

Of course the computer tutor will never replace the personal contact with teachers and fellow students that is a crucially important part of our education. Furthermore, since the computer tutor often becomes somewhat repetitious after a while, students may lose interest in this form of instruction. Program writers must take up the challenge of producing better teaching aids that will not lose the student in an air of impersonality and repetition.

Utility programs make it easier to use the computer and its peripheral devices. They are designed to save you time. Some make it easier to enter programs into the computer and “debug” them (get rid of the

Utility Programs

problems in the program); others make it easier to send commands to peripheral devices such as the disk drive and printer. There are even utility programs that make it easier for you to use the graphics and sound capabilities of the C-64.

An extremely useful utility program called **POWER PLUS** is included on the **SUPPORT DISK** provided with this book, or available via the enclosed order form. Part 5 will show you how to use this program.

A *terminal program* is used in conjunction with a modem. Together they allow your computer to exchange information with another computer that may be hundreds of miles away. With a terminal program your computer can receive and send information using the standard codes sent over telephone lines by all computers. Chapter 10 treats modems and terminal programs in more detail.

Terminal Programs

To give the computer instructions on the tasks we want it to perform for us, we have to be able to “talk” to it. In response, the computer has to be able to “tell” us what results it has produced for us, or perhaps what errors it has detected in our program. For this interaction to take place, there has to be a language that makes this two-way communication possible.

COMPUTER LANGUAGES

Just as English is our language, the computer’s language is *machine* language. But machine language is a cumbersome language for the user to master, because it’s difficult to learn and remember. For this reason, other “user friendly” languages have been written for computers. Some of these languages are composed of code words made up of English words such as “load”, “run”, and “list”, so that users can communicate with the computer more easily. Computers are manufactured with the built-in capacity to translate from these other languages to machine language.

Low-Level and High-Level Languages

Programming languages that communicate most directly with the computer are called low-level languages, while languages that are most like human languages are called high-level languages. For example, machine language is low level; *assembly language*—one step up from machine language—is also quite low level; and **BASIC**, the built-in programming language of the C-64 as well as many other home computers, is a high-level language.

Machine language is the lowest level language for computers—that is, it’s the closest we can get to directly manipulating the electronic pathways of the machine. Machine language is made up of “words” com-

Machine Language

posed of ones and zeros. For example, the *binary number* 00101001 is a computer word.

When a programmer talks to a computer through a program written in machine language, the programmer has direct control over the many intricate details involved in the execution of a program. This means that machine language programs can be very efficient and effective in carrying out their tasks.

Assembly language is one level higher than machine language, but it is a much lower level language than BASIC. Instead of using words made up of ones and zeros, its words are abbreviations, such as LDA, which means Load the Accumulator. Each assembly language word is equivalent to one or more machine language words. You can see why these English-based codes are much easier for programmers to learn and remember than strings of ones and zeros.

Since the computer can only understand the ones and zeros of the binary number system, assembly language codes, which are convenient for programmers to use, are meaningless to the computer. For this reason, once a program is written in assembly language, an *assembler* program is used to translate the assembly language codes into the numbers that the computer understands. The assembled machine code program will then run quickly and efficiently.

Although programming in assembly language offers great speed, flexibility, and efficiency, one must have a great deal of technical knowledge to program in such a low-level language. It is not for beginners.

BASIC (Beginner's All-purpose Symbolic Instruction Code) is the high-level language built into the C-64 computer. It uses easy-to-remember English words for the commands that make up its vocabulary. This makes it a very easy language for beginners to learn and use. The C-64 actually translates each command word in BASIC into an entire *routine* of commands in machine language that are executed when the BASIC command word is used. BASIC has quickly become the most popular language for home computers.

Remember that the computer has to translate the English word commands of high-level languages such as BASIC into words made up of ones and zeros, or machine language. A BASIC *interpreter* is built into the operating system of the computer to perform this translation—to turn the BASIC words you use into language that is understood by the machine. Some other high-level languages also have their own interpreters.

Assembly Language and Assemblers

BASIC

The BASIC Interpreter

You know that when you have to speak through an interpreter to someone who doesn't understand English, it takes twice as long to communicate, because every spoken word has to be translated by the interpreter before the listener can understand it. For the same reason, high-level languages such as BASIC are slow languages for the computer. Programs written in BASIC or some other high-level language will run much more slowly than those written in machine language. Most computer games are written not in high-level languages like BASIC, but in machine language, so they can run quickly enough to give us the fast action that we all like in arcade-style games.

If you want to write programs that will run fast but you don't want to learn to program in a low-level language, there is help for you. You can purchase a *compiler* program that will take the program you write in BASIC and translate it into machine language. The machine language program that results will then run without need of an interpreter. Because it doesn't have to call on the BASIC interpreter to carry out translation, the compiled program will run up to 40 times faster than the original BASIC program. Compilers are available for other high-level languages also. Some computer languages do not have interpreters at all; they must be compiled before they can be run. Compilers offer programmers a bridge from high-level to low-level languages.

Compiler Programs

Although BASIC is the high-level language built into the C-64, you can purchase programs that will "teach" your computer to speak other high-level languages. The sections that follow describe some of the other languages available for the C-64. Each language has specific features that make it particularly appropriate for certain applications.

More High-Level Languages

Logo is a high-level language that is very easy to learn and use. It has been highly successful in teaching children how to program. Logo uses a "turtle" character which the student commands to move on the screen. The turtle outlines any of a variety of geometric shapes as it moves from one location to another. As they learn to move the turtle, students also learn how to program the computer in the Logo language. When they get more accustomed to the language, children learn more complex mathematical and logical concepts.

Logo

Pilot is another high-level language. It was designed to help teachers create Computer Assisted Instruction (CAI) on all subjects. With it, teachers can tailor their programs for drills and tests that are both fun and educational.

Pilot

The C-64 version of Pilot implements the powerful sprite graphics and sound capabilities of the computer. Pilot provides a truly interactive learning process in a friendly and entertaining format.

COMAL is another high-level programming language. Its promoters claim that it is superior to BASIC in speed and efficiency. *COMAL* has been attracting a lot of attention in Europe and will probably have an impact on the home computer industry in the United States.

The C-64 version of *COMAL* makes full use of the computer's graphics. Although *COMAL* has many advantages over BASIC, it probably will not be widely used until computer manufacturers begin to build *COMAL* into the operating systems of their computers.

FORTH is a language that is somewhat like a high-level assembly language. It is very flexible, allowing the programmer to add commands to the *FORTH* vocabulary. In this way, the language may be tailored for particular applications. It is more compact than other high-level languages, and consequently, *FORTH* programs use less of the computer's memory space. *FORTH* is fast and efficient; *FORTH* programs can run much faster than those written in BASIC. Some promoters claim that their programming productivity increased tenfold when they switched to *FORTH*.

FORTH is a tightly structured language. To use it effectively, you must have your programming strategies carefully planned and organized at the outset. It is difficult to write a program in *FORTH* without a clear, detailed understanding of what each part of the program will do. While this characteristic makes *FORTH* harder to learn than BASIC, for example, it can also help programmers to analyze and achieve their programming goals more efficiently.

Pascal is another high-level language available in software form for the C-64. It is easy to learn and encourages the careful design of programs. Like *FORTH*, it is a highly structured language that does not tolerate sloppy, badly organized programming strategies as BASIC often does. Most programmers learn Pascal only after they master BASIC.

Pascal's major application is for large, complex, business-oriented programs. Once a program is written in Pascal, it is compiled into a low-level code, called P-code. This helps Pascal programs to run much faster than those written in BASIC.

CP/M, or Control Program for Microcomputers, is available for the C-64. *CP/M* is not a programming language at all. It's an entire operating system built around the Z-80 microprocessor rather than the 6510, the C-64's native microprocessor.

CP/M is available in cartridge form and plugs into the cartridge port of the computer. The CP/M cartridge actually contains a Z-80 and several other chips on a small board.

When you plug the CP/M cartridge into the C-64, you have two brains in the computer. Are two heads better than one? Well, in this case they are. The Z-80 microprocessor takes control of the machine's data processing operations, while the 6510 maintains the input and output operations of the computer.

CP/M is very "transportable" because it is a machine-independent operating system. This means that CP/M is a standardized operating system. Without this standardization, a program written for one type of computer would run on that computer only. But because CP/M-oriented computers all use the same operating system, CP/M programs will run on all computers that have CP/M capability.

You can see why CP/M appeals to programmers. When they write a program to run on CP/M, they are writing a program for a wide variety of computers. This will give them a much larger market for their product than a program written for the operating system of a specific computer.

Computer Memory

Computers seem to be able to “think.” This illusion results from the memory capabilities of the computer system. By being able to use programs built into it as well as programs that are temporarily entered into its memory, a computer has the capacity to perform a wide variety of tasks.

ROM AND RAM CHIPS

When we talked about programs, we noted that some were systems software and others applications software. As you recall, systems software is built into the computer by the manufacturer. The systems software that makes up the operating system of the computer is fixed and permanent. We can’t easily change it.

Systems software programs are built into ROM chips. ROM means *Read-Only Memory*. The information on these memory chips may be “read” by the computer but not changed, or “written” to. When we turn off the computer, these programs remain in the chips. When we turn on the computer again, the programs jump to life and perform their housekeeping and maintenance functions automatically.

ROM

There is a ROM chip that contains the BASIC language interpreter—the program that converts BASIC to machine language. One ROM chip provides the shapes of the alphabetic, numeric, and graphic character sets that we see on the video screen.

When we put an application program into the computer, it will not stay there permanently. We know that when we turn the machine off, the application program, whether it be a game or a business program,

RAM

disappears from the computer's memory. We save applications programs on diskettes or tapes so that we can easily load a program the next time we need it. This kind of storage allows us to turn off the computer without permanently losing the application program.

Applications programs are loaded (usually from tape or diskettes) into memory chips of the computer called *RAM*, or *Random-Access Memory*. The operating system of the computer makes an exact duplicate of a program on RAM chips. Now the program can be run, or executed. When a new program is loaded or when the computer is turned off, the old contents of RAM are lost, so the memory chips can be used again to hold another program. RAM chips allow the computer to run many different programs, one after the other, without having to store all of them permanently.

We know that the computer only understands binary numbers—words made up of ones and zeros. For the C-64, these binary words are groups of eight digits. For example, the computer word 00000111 has a value of 7. Each number in this eight-digit word is called a binary *bit*. A bit is binary because it can hold one of only two values, 0 or 1.

The entire word of eight bits is called a *byte*. A single byte can represent any of 256 different values, depending on the positions of ones and zeros in the byte. Half a byte, or a group of four bits, is referred to as a *nibble* (see Figure 3-1).

The C-64 is an 8-bit computer. This means that each memory location is made up of eight bits. More powerful computers are 16-bit and 32-bit machines. They can move and store 16 or 32 bits of information at once rather than just eight. These machines can perform larger operations faster than 8-bit machines.

BITS, BYTES, NIBBLES, AND KILOBYTES

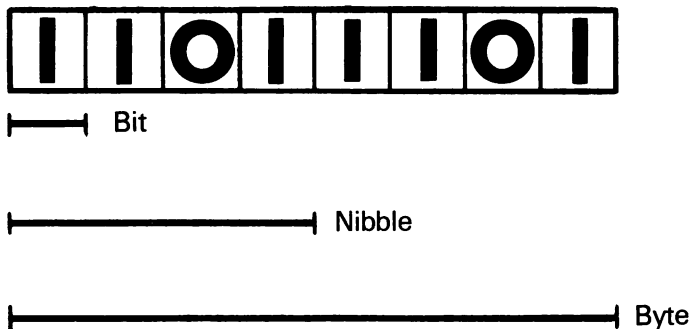


FIGURE 3-1. Bit, nibble, and byte.

A *kilobyte*, or *K* for short, is 1024 bytes of memory space. Normally *K* means 1000, but for computers *K* is 1024, or 4 times 256 (the largest value a byte can hold). *K* is the unit we use to describe the number of bytes of memory space a computer has. The C-64, as its name suggests, has 64K bytes of memory space, which means that it can hold up to 65,536 bytes (64 times 1024) of information in its memory registers.

One great advantage of the C-64 for experienced programmers is that the ability of ROM to reserve RAM can be turned off, making an entire 64K of RAM available for machine language programs. This is a very powerful feature of the C-64, and as more programmers learn how to use it, the software available for the C-64 will become increasingly complex, powerful, and efficient.

You may have noticed that when the computer is turned on, the top of the screen tells you that there are 38,911 bytes free. What happened to the rest of the 64K of memory? The 64K memory designation includes available RAM as well as RAM that is reserved for use by ROM. Recall that the operating system has programs built into the computer in ROM chips to carry out housekeeping and maintenance. Also, the character generator, the video chips, and the sound synthesizer are all in ROM. These ROM capabilities of the computer reserve some of the computer's 64K of RAM for housekeeping, language, and display functions. Figure 3-2 illustrates how ROM reserves some of RAM.

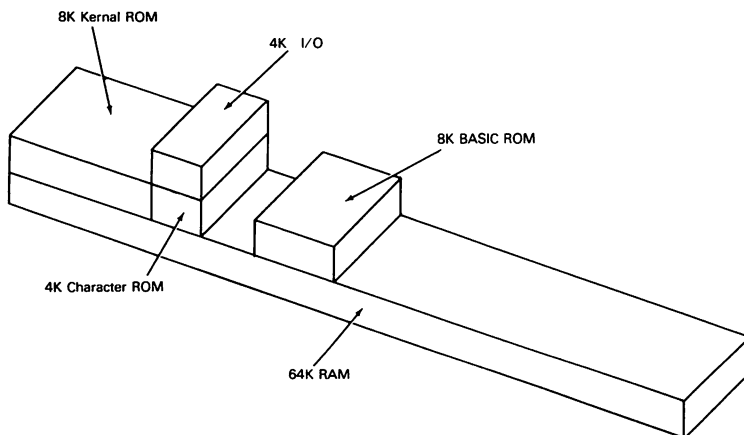


FIGURE 3-2. ROM and RAM



PART 2

The Components of the C-64 Computer System

Setting Up the Computer

Before you set up your computer, take a few minutes to plan where to install it. Follow these guidelines to avoid time-consuming reorganization later on.

THE COMPUTER WORK AREA

1. Find a room that will give you some privacy. You won't want to be disturbed unnecessarily when you're tackling a delicate aspect of programming strategy.
2. Plan your work area so that you'll have plenty of desk-top space for books, magazines, and floppy disk or cassette tape containers.
3. Anticipate expansion. If you don't have a printer now, you'll probably purchase one in the future, so reserve space for it.
4. You may want to use a telephone modem to communicate with other computers, so choose a part of the room that is close to a telephone line.
5. Choose a work area that isn't too close to any magnetic field generators, such as a fan, an air conditioner, or an audio speaker. Magnetic media such as cassette tapes and floppy disks may lose some of their information if they come into contact with these magnetic fields.
6. Your work area must be easy to clean, because dirt is an enemy of both the hardware and the magnetic media that store your programs and data. Keep away from fuel-burning appliances such as wood stoves and kerosene space heaters. They give off contaminants that may be harmful to both the electronics of your system and the data stored on floppy disks and cassette tapes. Tobacco smoke contains resins that will deposit on magnetic media. Keep your floppy disks and cassettes covered at all times.

7. Make sure that you have a grounded electrical outlet near your computer station. This is a safety precaution.
8. Don't choose a circuit that has heavy loads on it from appliances such as air conditioners and freezers. A sudden drain in the voltage when a large appliance is turned on could give the computer problems.
9. Plug the computer and all its peripheral equipment into a six-outlet power strip as shown in Figure 4-1. This item is available at most hardware stores. Use the switch on the power strip to turn on all your hardware at the same time. This will ensure that your equipment is turned on without any peripherals "locking up."
10. We recommend that you protect your computer system from damage that can result from power surges and voltage spikes by plugging your power strip into a surge protector. This device will provide a clean and precise flow of electricity to your system.

The major ports, or sockets, for attaching peripheral devices are at the rear of the computer. Figure 4-2 shows all these ports and their names. Figure 4-3 shows the side panel with the joystick/light-pen ports and the power switch.

THE COMPUTER

The Ports

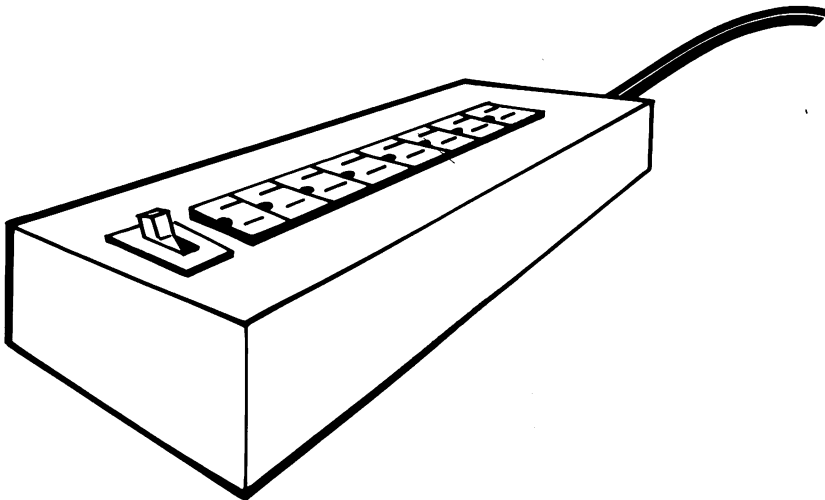


FIGURE 4-1. Power strip.

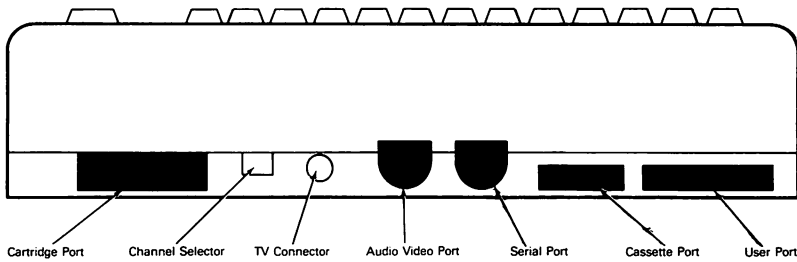


FIGURE 4-2. Rear view.

It is important not to plug a peripheral into a socket without first turning the power switches of both the computer and the peripheral device to the “off” position. If you neglect to do this, you may cause damage to the computer or the peripheral device. This caution does not apply to the video or joystick connections.

Refer to the user’s manual that came with the computer for a description of how to make the proper connections between the computer and its peripheral devices.

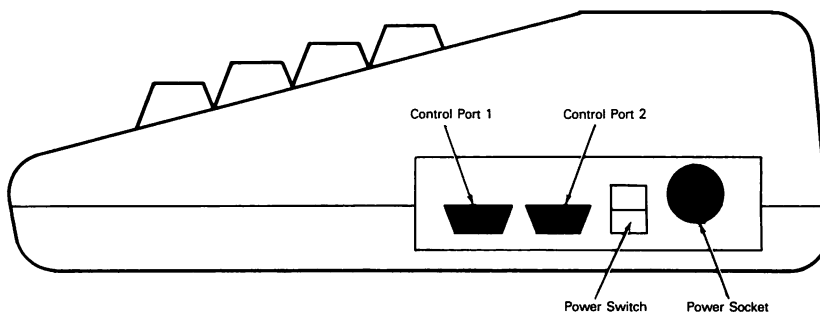


FIGURE 4-3. Side view.

The power switch on the right panel of the computer turns the C-64 on and off. Remember that when you turn the computer off, any programs or data that you have put in it will be lost. Before you turn the computer off, make sure you have your program or data stored on disk or tape.

Turn it on!

To power up the computer, do the following:

1. Turn the computer's power switch off. If you're using a power strip, make sure it is turned off.
2. Plug the cord from the power supply unit that came with the computer into the socket in the right panel of the computer, to the right of the power switch.
3. Plug the power supply unit into an outlet.
4. Turn the power switch of the C-64 to the "on" position. If you are using a power strip, turn it on.
5. After a moment you will see the following display:

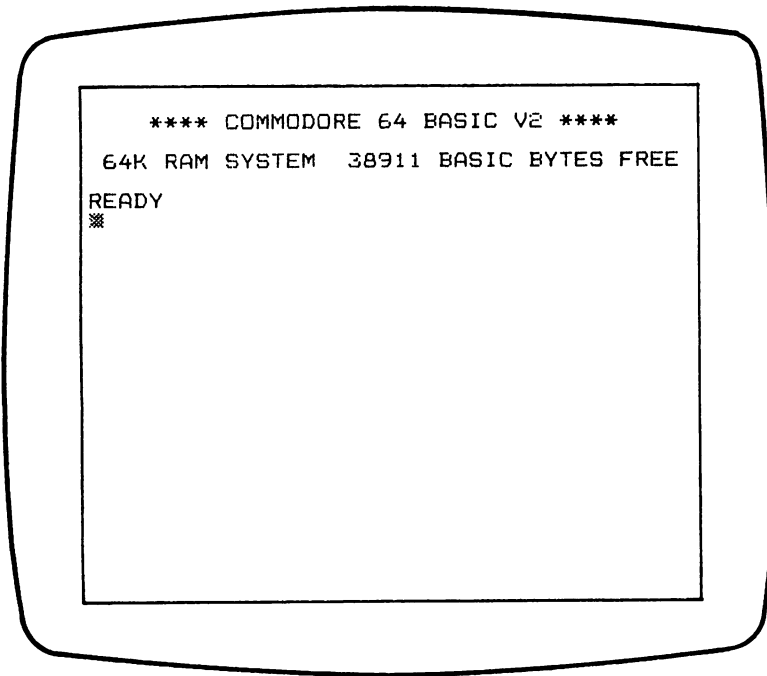


FIGURE 4-4. Power-up display.

If you are using a color monitor or color TV, the letters will be light blue against a dark blue background with a light blue border. If you are having trouble seeing the display because the contrast is too low, hold down [CTRL] while pressing [2] to change the character color to white. If you are still getting a display that is difficult to read, refer to Chapter 5.

Below the READY message you'll see a flashing square called the *cursor*. Whenever you see the cursor, it means that you have control of the computer; it's waiting for orders from you.

If you're not using a power strip with its central power switch to turn on your computer and its peripherals, you should keep in mind that your power supply is on as long as it's plugged into your wall socket. Unplug the power supply AC cord from your wall socket when you're not using the computer.

It is important that you use a three-hole grounded outlet for your computer system. Do not attempt to bypass grounding the computer by using a three-prong to two-prong adapter and a two-prong electrical outlet. One reason for this precaution is that surge protectors send surges of electricity to the third prong.

Manufacturers frequently make changes in the ROM programs they build into a particular computer, even while the computer is under production. The C-64 computer is no exception. At the time of this writing, we are aware of three versions of the C-64. The main differences among these versions are due to changes in the operating system, or *kernel* ROMs, of the C-64. You should find out which version you have, because it can make a difference when you start entering programs into the computer.

To determine which version you have, either follow the instructions below or, if you have the SUPPORT DISK supplied with this book, load and run the KERNEL REVISIONS PROGRAM described in Chapter 23.

Several Types of C-64 Computers

Which Version?

1. Clear the screen by holding down [SHIFT] and pressing [CLR/HOME]. This clears the screen and places the cursor, the flashing square, in the "home" position at the upper left side of the screen.
2. Now move the cursor down five lines by hitting [CRSR DN] (the CURSOR DOWN key) on the lower right side of the keyboard five times.
3. Type in the following command exactly:

POKE 1024,1

4. Press [RETURN].
5. If you see an 'A' in the home position, then you have Kernel ROM 1 or 3. Kernel 1 makes each character that is POKEd to the screen white. If the 'A' is white, you have Kernel 1. If the 'A' is the same color as the cursor, then you have Kernel 3. Kernel 3 always POKEs the characters to the screen in the same color as the cursor. With Kernel 3, you can determine what color to make the characters you POKE by changing the color of the cursor. If you do not see the 'A', then you have Kernel 2.
6. Those of you who did not see the 'A', press [HOME], this will home the cursor without clearing the screen. Now the 'A' will show "through" the cursor as it blinks on. Kernel 2 makes the color of the character the same as the background of the screen. That's why you couldn't see it until you moved the cursor to the same spot to provide a contrast.

SCREEN MEMORY LOCATIONS

The C-64 screen has 1000 locations (25 rows times 40 columns) which can display any character from the character sets built into the computer's ROM. Every one of the 1000 screen locations has an address number assigned to it. Each address can hold a number, just as a post office box can hold a letter. The 1000 addresses are numbered from 1024 to 2023, representing 1000 locations on the screen, as illustrated in the Screen Memory Map in Appendix F.

Location 1024 is the address of the home position on the screen. The number '1' in the command POKE 1024, 1 stands for the letter 'A'. We used this command to POKE, or put, the value '1' into the screen memory location in the computer that is responsible for character output at the home position (upper left-hand corner). This causes the letter 'A' to be displayed in the home position. A list of the codes for all the keyboard characters (letters, numbers, and graphic characters) appears in Appendix C. You may use this list to POKE any character into any of the 1000 possible positions on the screen. After you find out which Kernel you have, turn to Appendix F and try POKEing a few characters into different screen locations.

Those of you with Kernel 2 are probably wondering how you're going to see the characters that you POKE to the screen if they're always the same color as the background color of the screen. Follow the instructions in the box below to fix this problem.

SCREEN POKE FIX

With Kernel 2, you won't be able to see the characters that you POKE to the screen because the characters will be the same color as the screen background. You can use the FIX routine below to POKE characters to the screen that can be seen.

The FIX

Follow these steps to allow programs to POKE characters to the screen:

1. Before each [CLR] in your program (represented as an inverse heart character when the program is listed on the screen), POKE 53281 (the memory location for changing background colors) with the color you want your characters to be. Clearing the screen will now fill the color RAM memory with this color.
2. After the [CLR], POKE the background (53281) to any color other than the character color.

To show how this is done, let's POKE the letter 'A' into the first 40 screen memory locations, using a black background and green letters, with this program:

```
10 POKE 53281,5:PRINT"[CLR]":POKE 53281,0
20 A=1024:FOR B=A TO A+39
30 POKE B,1:NEXT B
```

Without the POKes in line 10, all the A's that appear on the screen when you run this program would have appeared as white with a Kernel 1 ROM computer. With a Kernel 2 computer, none of the characters would be seen, since the characters are the same color as the background.

Another way to solve this problem is to upgrade your computer by having a technician install the newest version of the Kernel ROM chip in your computer.

Commodore 64 computers are not all the same. Some have fewer chips on the circuit board, which enables the computer to run cooler. The more chips there are on the board, the warmer the computer will get. Heat may eventually cause permanent damage to the chips. A C-64 with more functions in fewer chips will probably last longer.

Some C-64s also have an 8-pin rather than a 5-pin audio/video socket. Both versions will perform equally well when properly connected to a video monitor (Chapter 5).

In addition, the power supply has been changed. The power plug from the power supply of some C-64s has only four pins, even though the power socket on the right panel of the computer can accept up to seven pins.

Other Changes

The Video Screen

The video screen is an important output device. It lets you see the data you are typing into the computer and also displays the computer's output after it processes this data.

The cheapest and easiest way to get a video output from your C-64 is to connect it to a regular color or black-and-white TV. Refer to the user's manual provided with the computer for a description of how to connect your TV to the C-64.

REGULAR TV

If you're planning to use your C-64 mostly for business applications involving only the manipulation of numbers and words, then use a black-and-white TV. It costs less than a color TV and will give you a sharper picture.

Color versus Black and White

If games and graphics are important to you, use a color TV to get the most out of the excellent, colorful programs now available for the C-64.

Because a TV has a receiver designed to pick up broadcasts, it also tends to pick up interfering signals that can cause wavy lines and indistinct characters. Here are a few tips on how to improve the quality of your TV screen display:

Improving the TV Picture

1. Move your connecting cable around to see if one position gives you a better picture than others.

2. Keep your disk drive and cassette tape player at least two feet from the TV.
3. Try replacing the antenna switch box with a standard 300 ohm to 75 ohm transformer, and the regular connecting cable with a 75 ohm coaxial shielded cable.
4. Buy a ferrite toroid core, about two inches in diameter, from an electronics shop. Wrap the video cable through and around the loop three or four times, as shown in Figure 5-1, and then plug the cable into the back of the computer. This should reduce the amount of interference that the cable itself picks up. Satisfactory results can also be obtained by wrapping the cable three or four times through a cardboard cylinder such as the cardboard core inside a roll of toilet paper.
5. If you feel technically competent to do so, you can make adjustments inside the computer to improve your video output.

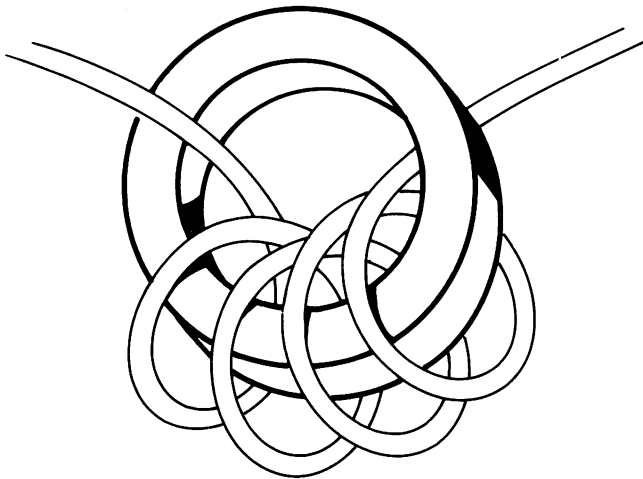


FIGURE 5-1. Toroid core.

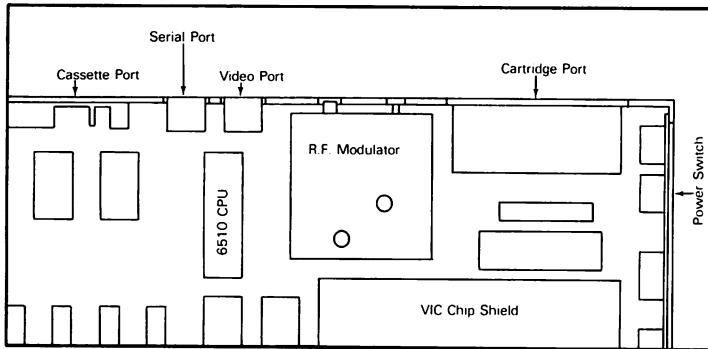


FIGURE 5-2. The RF modulator.

CAUTION: Because you must open the computer to perform these adjustments, this procedure may void your warranty.

Turn off the computer. Turn it over and remove the three retaining screws at the bottom. Turn the computer back over and carefully lift the keyboard portion up. Locate the RF modulator pots using Figure 5-2 as a guide. Type several lines of characters so that the screen contains some information that you can observe while testing. Adjust each of the modulator pots with a small screwdriver, carefully watching after each adjustment to see if there is any improvement.

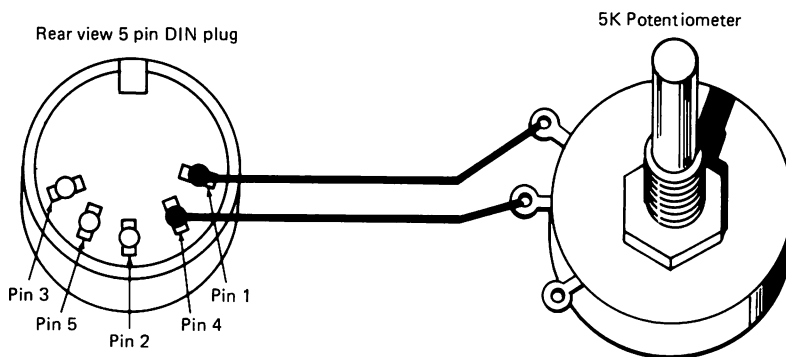


FIGURE 5-3. DIN plug connected to potentiometer.

A PROJECT FOR GETTING THE BEST OUT OF YOUR TV

When you examine the audio/video pin allocation diagram in Figure 5-3, you will find that there are both *luminance* and *video* output pins. Luminance output is the signal for monochromatic (black-and-white) monitors; video output is the signal for color output. Mixing the luminance and video signals that the computer puts out will produce a sharper, crisper picture. But if these two signals are mixed by crossing the two leads, the contrast of the picture will be too intense and, consequently, unacceptable.

To mix the signals effectively, connect a 5K variable resistor, or potentiometer, between the luminance (pin #1) and the video (pin #4) signals. You can buy a 5K variable resistor at an electronics store. Refer to Figure 5-3 to attach the potentiometer properly. Once it is connected, adjust the setting of the potentiometer to select the best possible mixture of the two signals.

Output from the computer may also be sent to a video monitor. A monitor is like a TV, but without a receiver for picking up broadcasts. The video monitor produces a much sharper picture than a regular TV because it is designed to provide high resolution output. It is an excellent choice for the business-oriented user.

VIDEO MONITOR

Some monochrome monitors display green characters on a black background. They are excellent for word processing or other applications that do not require the use of the C-64's color capabilities.

Monitor Types

Green Screen

Other monochrome monitors display amber characters on a black background. The amber color may be easier on the eyes than green. If you plan to spend long hours in front of your monitor, you may wish to purchase an amber screen monitor.

Amber Screen

Color monitors are available from many manufacturers, and all of them will work fine with the C-64. The Commodore 1701, 1702, and 1703 Video Monitors (Figure 5-4), however, will give you the highest quality display at a very reasonable cost.

Color Monitor

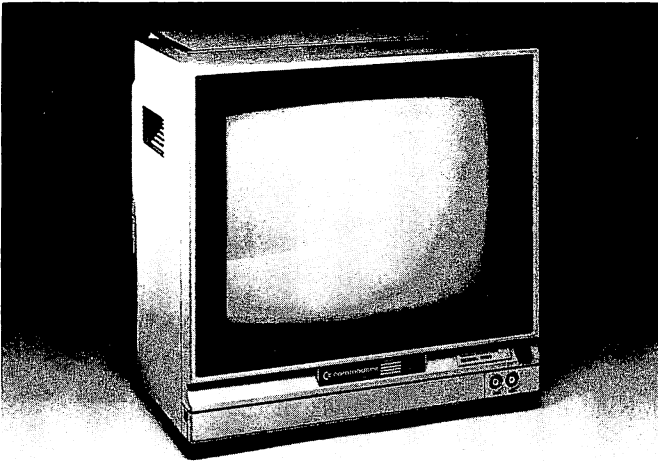


FIGURE 5-4. The Commodore 1701 video monitor.
(Courtesy of Commodore Business Machines, Inc.)

The monitor is connected using the audio/video socket of the computer, not the RF socket that a regular TV plugs into. You will need a connector made from a 5-pin DIN plug and two cables. Some Commodore dealers stock such connectors ready-made, so you might want to purchase one rather than make your own.

Plugging in the Monitor

If you decide to make your own connector, follow these steps to properly connect your monochrome monitor.

Connecting Monochrome Monitors

1. At an electronics store, purchase a 5-pin DIN plug and two cables with RCA-type leads as shown in Figure 5-5. One cable will go to the monitor and the other to a stereo to give you audio output.

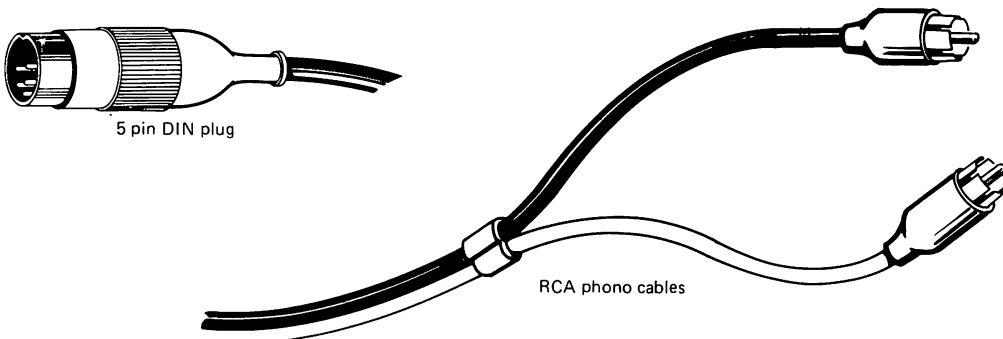


FIGURE 5-5. DIN plug and cables.

2. To take full advantage of the monochrome monitor's capability, you must use the luminance pin of the audio/video socket (pin #1 in Figure 5-6) to connect with monochrome monitors. Do not use the video output; it will give you an inferior picture.

Use Figure 5-6 to find the luminance pin of the DIN plug you purchased, and solder one lead of your cable to it and the other lead to the ground pin.

3. Solder the leads of the other cable to the audio and ground pins.

Plug the RCA connector for video output into the back of the monitor. To get audio output when using a monitor, you must connect the audio lead shown in the diagram to an amplifier and speaker.

For color monitors, follow the same instructions as for monochrome monitors, but use the video lead instead of the luminance lead out of the computer.

*Connecting Non-CBM
Color Monitors*

The 1701 Video Monitor: The Commodore 1701 Video Monitor (Figure 5-4) has connection sockets both in front and in the rear. The single cable supplied with the unit plugs into the front of the unit. The rear connection, however, can take both the video and luminance

*Connecting the CBM
1701 or 1702
Video Monitors*

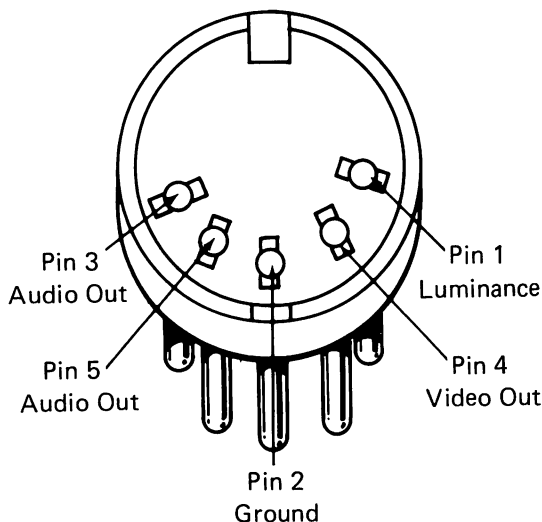


FIGURE 5-6. Rear-view, 5-pin DIN plug.

signals (via a double cable) and mix them to provide a very bright, sharp picture.

To connect the computer to the 1701 via the rear connections, you must obtain a 5-pin DIN plug with three or four RCA jacks, as shown in Figure 5-7.

1. Set the “front/rear” switch at the rear of the 1701 monitor to “rear.”
2. Try plugging one after another of the four leads into the audio socket at the rear of the unit until you find one that does not produce a humming noise. Leave this one plugged in.
3. Try various combinations of the three remaining leads in the other two sockets until you get a good color picture. Don’t worry—you can’t do any harm to the electronics of either the computer or the monitor by plugging a cable into the wrong socket.

The 1702 Video Monitor: The 1702 color monitor was designed for Commodore 64s with an 8-pin rather than a 5-pin DIN socket. A cable with three RCA connectors that plug into the rear of the monitor is provided with the 1702 monitor.

C-64s with the 5-pin DIN plug cannot be connected to the 1702 with the cable provided with the monitor. If you try to use a 5-pin DIN connector cable as described on page 38, you have trouble getting colors on the screen when the background is a light color. This is caused by a different chroma output used in the C-64 computers that have the 8-pin DIN socket.

CAUTION: The following procedure should be carried out only by someone experienced in electronics.

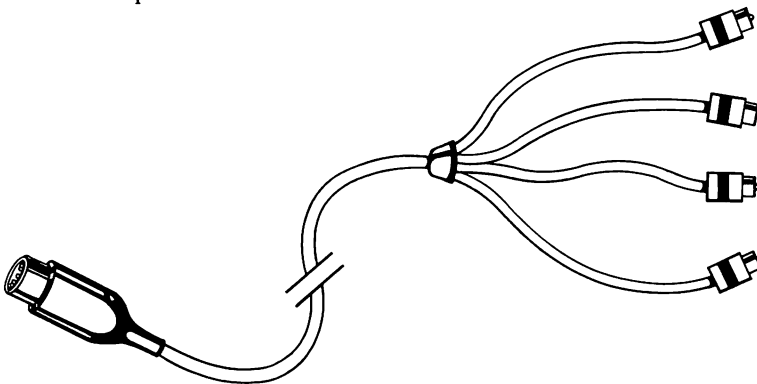


FIGURE 5-7. 5-pin DIN plug with 4 RCA connectors.

To get better output to a 1702 monitor with the 5-pin connector, you must lower the voltage of the chroma or video line by placing a 500-ohm resistor on the chroma/video output. Solder the resistor in-line on pin 4 either on the 5-pin DIN plug or inside the 1702.

6

The Keyboard

As you become more familiar with your C-64 system, you'll be entering data into the computer for processing. Data may be numbers for a computational program, the text of a document you're writing, or instructions for a program you're developing. Usually, you will use the keyboard to enter this data.

The C-64 keyboard is similar to a typewriter keyboard, but with some additional features. In this chapter, we'll examine the keys of the computer according to their functions. In each of the illustrations that follow, a group of keys is shaded. Each illustration is followed by text describing the function of the shaded keys. As you study the illustrations and accompanying text, you will find that some keys fall into more than one group.

The *text keys* shown in Figure 6-1 are the same as those on standard typewriters: the alphabetic keys, punctuation keys, space bar, shift and shift lock keys, and carriage return key. These are the keys most commonly used for word processing.

TEXT KEYS

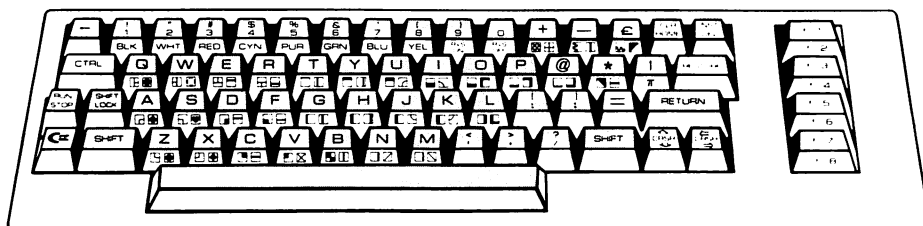


FIGURE 6-1. Text keys.

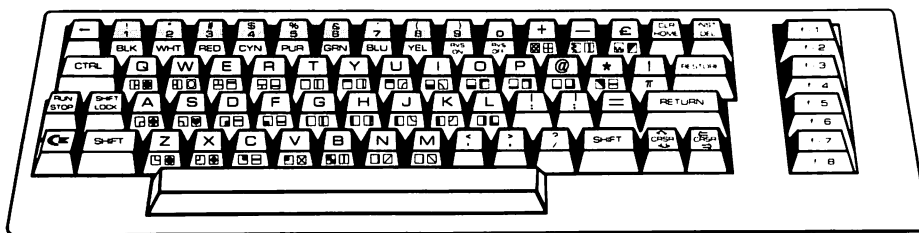


FIGURE 6-2. Numeric keys.

When the C-64 is turned on, the keyboard is in *upper-case/graphics* mode. In this mode, unshifted letters will give you upper-case letters. To type in upper- and lower-case text, press [CBM] [SHIFT] together to switch to the *upper/lower-case* mode. Now unshifted letter characters will give you lower-case letters, while shifted letter characters will give you upper-case letters.

The *numeric*, or *number keys*, are in the top row of the keyboard, as shown in Figure 6-2. They may be typed without the shift key in either upper/lower-case or upper-case/graphics mode. If you wish to use the C-64 for your business and expect to enter large amounts of numeric data, you may want to purchase a numeric keypad, which is similar to an adding machine and cash register keypad (Figure 6-3). The numbers on the keypad are usually clustered together in a three-by-three grid; an arrangement that makes numerical entries easier and quicker.

NUMERIC KEYS

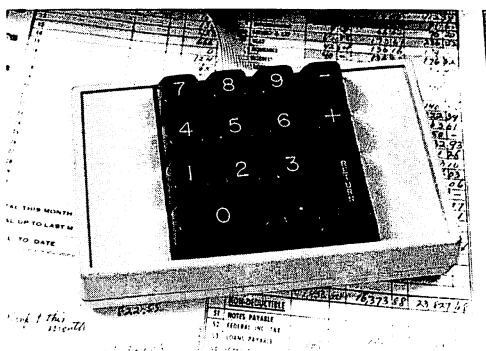


FIGURE 6-3. A numeric keypad.
(Courtesy of Computer Place.)

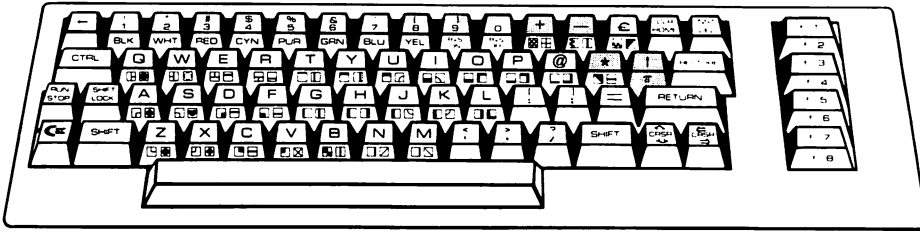


FIGURE 6-4. Operation keys.

The *operation keys* (shown in Figure 6-4) will permit you to perform mathematical calculations. The plus sign (+) and minus sign (-) are for addition and subtraction, the slash (/) is for division, and the asterisk (*) is for multiplication. The up arrow ($\uparrow$) will raise a number to some power; it signifies exponentiation. Pi (π) will give you the mathematical value of any circle's circumference divided by its diameter: 3.14159265.

OPERATION KEYS

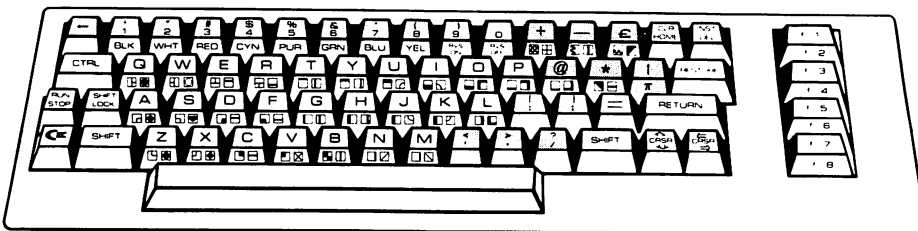
Symbol keys are all the non-alphabetic, non-numeric keys (see Figure 6-5). They have a variety of functions for programming and business use.

The check mark ($\checkmark$) is also available, but it does not appear on the keyboard. Press [SHIFT][@] in upper/lower-case mode to obtain the check mark.

SYMBOL KEYS

The *graphic characters* on the keyboard are marked on the fronts of the keys as indicated in Figures 6-6 and 6-7. The SHIFT key and a graphic key, pressed together, will give you the graphic character marked on the right front of that key. The CBM logo key, located on the lower

GRAPHIC KEYS



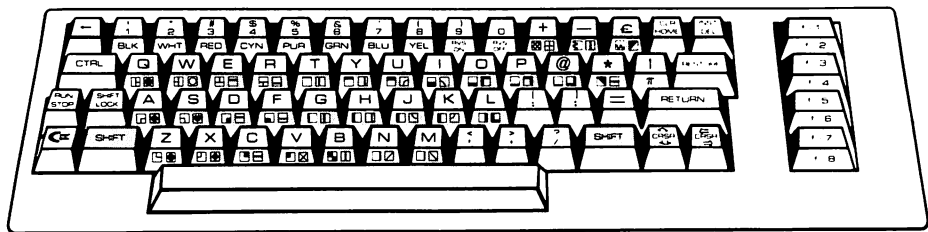


FIGURE 6-6. Graphic characters with SHIFT key.

left corner of the keyboard, is another kind of shift key that will produce the graphic characters marked on the left fronts of the graphic keys.

How can you remember which shift key to use for each graphic character?

The CBM key is to the left of the SHIFT key; it will give you the graphic character marked on the *left* side of the key you press.

The SHIFT key is to the right of the CBM key; it will give you the graphic character marked on the *right* side of the key you press.

Several more graphic characters are available that are not marked on the keyboard. They are obtained in upper/lower-case mode. Remember that you must press [CBM] [SHIFT] together to move from upper-case/graphics mode to upper/lower-case mode, and vice versa. These graphic characters are obtained in the following way:

LOWER-CASE MODE		
[SHIFT] [↑]	gives you	⌘
[SHIFT] [⓪]	gives you	✓
[SHIFT] [£]	gives you	⌘
[SHIFT] [π]	gives you	⌘
[CBM] [*]	gives you	⌘

TABLE 6-1. Graphic characters not marked on the keyboard.

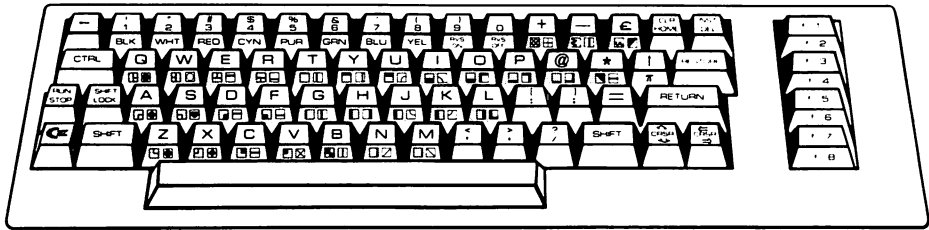


FIGURE 6-7. Graphic characters with CBM key.

The *color keys* are also the numeric keys 1 through 8. Color names are marked on the fronts of the keys (see Figure 6-8). The C-64 can actually display 16 different colors using these eight keys. Unfortunately, the names of only eight of these colors appear on these keys.

Hold down [CTRL] and press a color. The color of the cursor and all subsequently typed characters will be printed to the screen in the color marked on the front of the key. Hold down [CBM] and press any numeric key from 1 to 8 in order to get a color from the second set of eight colors. Table 6-1 shows how to key the 16 different colors.

COLOR KEYS

KEYBOARD	COLOR	KEYBOARD	COLOR
[CTRL] [1]	BLACK	[CBM] [1]	ORANGE
[CTRL] [2]	WHITE	[CBM] [2]	BROWN
[CTRL] [3]	RED	[CBM] [3]	LT RED
[CTRL] [4]	CYAN	[CBM] [4]	DK GRAY
[CTRL] [5]	PURPLE	[CBM] [5]	MD GRAY
[CTRL] [6]	GREEN	[CBM] [6]	LT GREEN
[CTRL] [7]	BLUE	[CBM] [7]	LT BLUE
[CTRL] [8]	YELLOW	[CBM] [8]	LT GRAY

TABLE 6-2. 16 color choices.

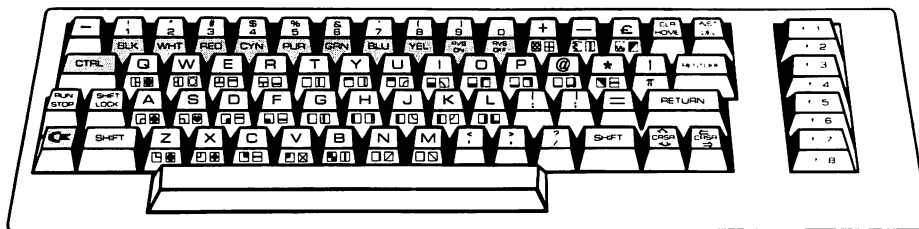


FIGURE 6-8. Color keys.

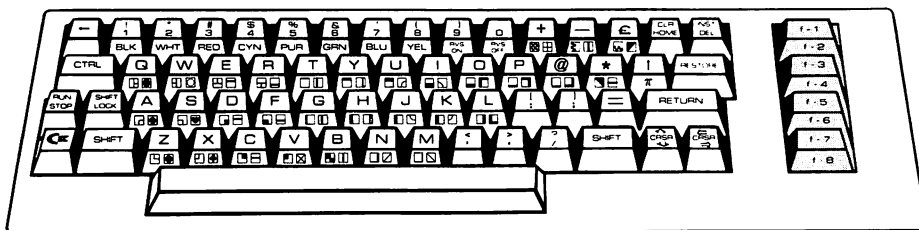


FIGURE 6-9. Function keys.

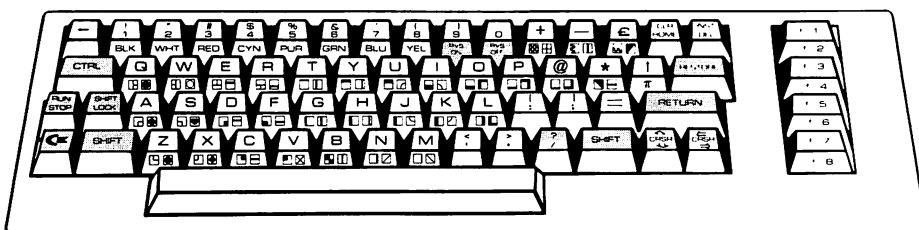


FIGURE 6-10. Special control keys.

THIS IS A SAMPLE OF REVERSE CHARACTERS

FIGURE 6-11. Example of reverse text.

The four keys to the right of the main keyboard are the *function keys* (see Figure 6-9). These keys do not have any assigned function. Without programming, they do nothing. Programmers can assign functions to these keys in their programs. For example, function keys may be programmed to change the color of the screen for a word processing program. Or they may be used to get input from a user; for example, the function keys are pressed by the user to start an arcade game or to end it.

A programmer may assign a specific function to any key on the keyboard, but when any key other than a function key is used this way, the original function of that key is lost for the duration of the program. In Chapter 22, you will learn how to program the function keys.

FUNCTION KEYS

The *control keys* (shown in Figure 6-10) do not print characters on the screen; they control specific functions. Here are brief descriptions of the functions of these keys.

SPECIAL CONTROL KEYS

The REVERSE ON and REVERSE OFF keys are the number keys 9 and 0. To use [RVS ON] and [RVS OFF], you must use [CTRL] as an access key, holding down [CTRL] and then pressing [9] or [0] at the same time. Once you have pressed [CTRL] [9], all the character keys that you press subsequently will be displayed on the screen in reverse mode, as shown in Figure 6-11.

REVERSE ON and REVERSE OFF

This means that if your normal display shows light blue characters on a blue background, pressing [CTRL] [9] will cause the characters you type in to appear blue on a light blue background. If you now press [CTRL] [0], the reverse feature will be turned off, and characters you type will be displayed normally again.

The SHIFT key accesses a second set of characters. As we mentioned earlier, in upper-case/graphics mode, the SHIFT key gives you the graphic characters on the right fronts of the keys. For keys without graphic characters, the SHIFT key accesses whatever appears on the upper half of the top surface of the key. For example, the exclamation point on the number 1 key, CLR on the CLR/HOME key, and CRSR UP on the vertical cursor key are all marked on the top half of these keys and are accessed with the SHIFT key (in both upper-case/graphics mode and upper/lower-case mode).

SHIFT

In upper/lower-case mode, the SHIFT key gives you the capital letters of the alphabetic keys, as it does on a regular typewriter. In

upper-case/graphics mode, the SHIFT key gives you the graphic characters marked on the right side of the front of the keys.

The SHIFT LOCK key will stay down when pressed and will cause all the keys that are subsequently pressed to be shifted. It serves the same function as it does on a standard typewriter.

SHIFT LOCK

This is the Commodore logo key. Like the SHIFT key, it is held down while another key is pressed. Remember, the CBM key accesses the second set of eight colors, and also the graphic characters marked on the left fronts of the keys in both upper-case/graphics mode and upper/lower-case mode.

CBM

The CONTROL key, like the SHIFT and CBM keys, is held down while another key is pressed. The CONTROL key is used for changing colors and for turning the reserve mode on and off.

CONTROL

The CONTROL key is also used to slow the scrolling rate of listings on the screen.

The question mark on the keyboard is the short form for the BASIC command PRINT. Whenever you wish to use this command, you may type "?" as a short form.

?

The RETURN key is one of the most important keys on the keyboard. When you type text on the screen, [RETURN] acts like the carriage return of a standard typewriter, sending the cursor to the beginning of the next line. When pressed on a command line, the RETURN key will execute that command and then send the cursor to the beginning of the next line. When you press [RETURN] after typing in a line of a BASIC program, that line will be entered into the memory of the computer. It will replace any BASIC line already in memory with the same line number. Nothing you type in will be executed or entered into the memory of the computer until the RETURN key is pressed.

RETURN

STOP: When the RUN/STOP key is pressed unshifted, STOP is executed. This will terminate the program that is running. It will also stop the listing of a program as it *scrolls* up the screen. To see how the STOP key stops a running program, try the following:

RUN/STOP

```
NEW [RETURN] (to erase any programs in memory)
10 PRINT "TERRIFIC COMPUTER"; [RETURN]
20 GOTO 10 [RETURN]
RUN [RETURN]
```

You will see the words 'TERRIFIC COMPUTER' appear all over the screen and continue to scroll upward as more words are printed at the bottom. Now press [STOP] to stop the program.

The STOP key will also terminate the loading of programs from tape and disk.

RUN: The RUN function is obtained by pressing [SHIFT] [RUN/STOP]. It will result in the prompt 'PRESS PLAY ON TAPE'. When you do this, the first program that the computer finds on the cassette tape will be LOAded and RUN. Since this function LOADs and then automatically RUNs a program, it is called an AUTO RUN.

The RESTORE key has no effect when it is pressed by itself. When it is pressed with [RUN/STOP], however, the RESTORE key will reset the graphic and sound registers back to zero. You may hold down [RUN/STOP] and then hit [RESTORE] to turn off any sound, clear the screen, and return to the colors you get when you first turn on the computer. Sometimes when a program doesn't end properly, [RUN/STOP] [RESTORE] will return the computer to normal, permitting you to run the program again. Using RESTORE in this way is sometimes called a *soft reset*.

RESTORE

The *editing keys* permit you to move the cursor to different locations on the video screen and make changes using the *screen editor* of the C-64 (see Chapter 12 for a description of the screen editor).

The four editing keys are: CURSOR DOWN/UP, CURSOR RIGHT/LEFT, CLEAR/HOME, and INSERT/DELETE.

EDITING KEYS

CURSOR DOWN: The CURSOR DOWN key is located in the lower right corner of the keyboard. It is used unshifted, as suggested by the down arrow being on the lower half of the key. This key will move the cursor down repeatedly as long as the key is held down. You may also hit the key just once to move the cursor down only one row. After the cursor reaches the bottom of the screen, the screen will scroll upward if you continue to press the CURSOR DOWN key. You have to be careful here, because any information at the top of the screen will disappear off the top when the scroll occurs.

CURSOR DOWN/ UP

CURSOR UP: The same key shifted will cause the cursor to move up. When the cursor reaches the top of the screen it will stop without causing the screen to scroll downward.

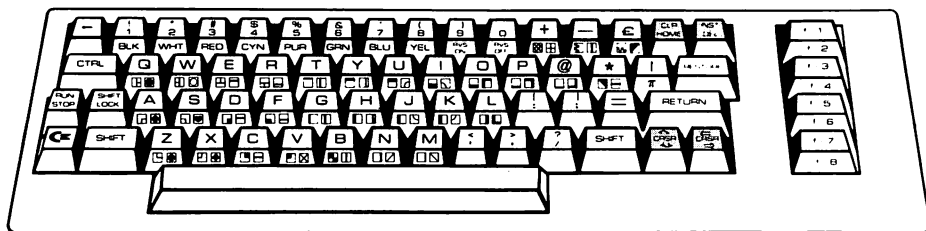


FIGURE 6-12. Editing keys.

CURSOR RIGHT: Unshifted, the horizontal cursor key at the lower right corner of the keyboard will move the cursor to the right. When it reaches the right side of the screen the cursor will “wrap around” to the beginning of the next line and, if the key is still pressed, the cursor will continue to move horizontally to the right. When the cursor reaches the bottom right corner of the screen, it will continue to wrap around to the next line, causing the screen to scroll up one line. If you are not familiar with the wrap around, hold the CURSOR RIGHT key down and watch the cursor move.

CURSOR RIGHT/ LEFT

CURSOR LEFT: When the horizontal cursor key is shifted, it will cause the same sort of motion to the left. This time, when the cursor reaches the left side of the screen it will “wrap around” upward and to the right side and continue left. When it reaches the upper left corner of the screen, or HOME position, however, the cursor will stop moving.

HOME: When the CLEAR/HOME key is pressed unshifted, the cursor moves to the HOME position at the upper left-hand corner of the screen. The HOME key is used to move the cursor to this position quickly.

CLEAR/HOME

CLEAR SCREEN: This editing function, which clears the screen, appears on the upper half of the CLEAR/HOME key and so must be pressed along with the SHIFT key. Press [SHIFT] [CLR] to clear the screen and to home the cursor.

DELETE: When the INSERT/DELETE key is pressed unshifted, the character immediately to the left of the cursor is deleted. If you continue to press the delete key, more characters will be deleted. This also moves the character under the cursor and all characters to the right side of the cursor to the left as each delete is executed, filling up the space left by the deleted characters. Type some text and then use the delete key to see how it performs.

INSERT/DELETE

INSERT: Sometimes you will want to insert a character somewhere. To achieve this, press [SHIFT][INST/DEL] to open a space to the left of the cursor. Now you can insert a character in the space that was created. As with most editing keys, INSERT will repeat if you continue to press it.

SHIFTED SPACE AND SHIFTED RETURN

Shifted SPACE

When you type [SHIFT] [SPACE], you create a space just as you do when you press [SPACE] alone. However, the computer stores **SHIFTED SPACE** as a different character than **SPACE**. Word processors sometimes use the shifted space when words with a space between them are not to be broken. Some programmers include shifted spaces in their file names to prevent unauthorized copying of their programs.

Shifted RETURN

The **SHIFTED RETURN** moves the cursor to the beginning of the next line as [RETURN] does, but with one important difference: Pressing [SHIFT] [RETURN] doesn't enter the information from that line into the memory of the computer. Hence, you can use the **SHIFTED RETURN** to exit a line without executing it and without storing it as part of a program. The **SHIFTED RETURN** is also used to get out of *quote mode*, as described on page 107.

The Tape Drive

In this chapter you'll learn about the Datasette tape drive, the machine that loads and saves programs on cassette tape. As discussed earlier, you need a program to take control of your computer and operate it to yield a desired objective. The program is a set of carefully structured commands that tells the computer what to do.

The tape drive is only one of several ways by which a program can be entered into the computer. We'll briefly discuss four of these methods before we discuss the tape drive itself.

PUTTING PROGRAMS INTO YOUR COMPUTER

Typing programs from sources such as books or magazines is time-consuming, but it gives you the opportunity to examine the structure of a program as you type it in. After you have typed in an entire program and debugged it, save the program to tape or disk (we'll show you how below) so you won't lose it when you turn off the computer.

Typing Programs

The ability to load and save your programs on magnetic media is one of the major advantages of using a computer. The tape system can load and save both programs and data, but it is slower and less efficient than a disk drive. A tape drive loads and accesses information *sequentially*, or one after another. Disk drives access information *randomly*, from any location on the diskette.

There are several tape utilities that increase the speed of the tape system by packing information more densely on the cassette tape medium.

In summary, a disk drive's advantages are:

1. It is able to load and save programs faster than a tape drive.
2. It can display a constantly updated disk DIRECTORY, which displays the names of all files on the diskette.
3. It can access a selected program randomly without having to go through programs that may precede it.

Loading Programs From Tape or Disk

A cartridge is a program stored on a ROM chip. As soon as it is plugged into the computer, and the computer is turned on, the cartridge becomes part of the computer's hardware. Plug in a game cartridge, for example, and the computer turns into a game machine entirely devoted to the operation of that game. Other cartridges are devoted to utilities such as machine monitors, sound and graphic utilities, and BASIC enhancements.

A cartridge program can run automatically as soon as the computer is turned on. It saves time by eliminating the need to load programs from tape or disk. That's especially helpful with long programs.

Because a cartridge reserves a large amount of memory in the computer, you may have to remove the cartridge when you want to use certain other programs. These programs may require the same memory locations as the cartridge program. Another disadvantage of cartridge programs is that they cannot be modified by the user because the programs exist only in hardware form. Finally, cartridges are generally more expensive than programs available on disk or tape.

CAUTION: A cartridge **MUST** be installed or removed with the computer turned off to avoid damage to the computer and the cartridge. Such damage might include shorting out certain pins, which will send destructive electrical spikes to the chips in the computer and the cartridge.

It is possible, with the proper software, to use your modem and telephone line to enter programs from remote computers into your computer. You can then save these programs on tape or disk. This is an inexpensive way to obtain many public domain programs and enlarge your software library.

The Commodore Datassette tape drive (Figure 7-1) and the operating system you get with the C-64 make up one of the best combinations available for tape storage and access. To become familiar with the way the Datassette works, type in the following short program so you can learn the various tape commands described in this chapter. Remember to hit [RETURN] at the end of each line.

```
10 FOR X=1 TO 100
20 PRINT "THIS IS A TAPE TEST[SPACE]";
30 NEXT X
```

Type RUN to see what the program actually does. Now you're ready to try some tape commands.

Plugging in a Cartridge

Downloading via Modem

USING THE DATASSETTE

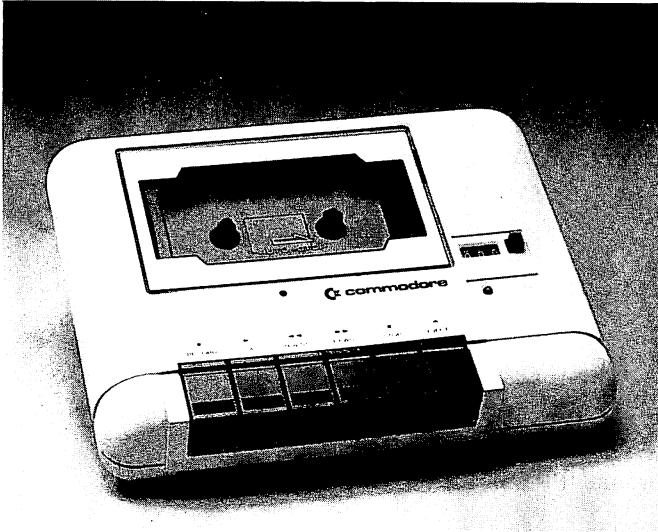


FIGURE 7-1. The Datassette tape drive.

(Courtesy of Commodore Business Machines, Inc.)

1. Insert a blank cassette tape into the tape player and rewind it to the beginning.
2. Type **SAVE** followed by the name of the program in quotes and hit [RETURN] as shown below.

```
SAVE "TEST" [RETURN]
```

3. The screen will now say 'PRESS PLAY AND RECORD ON TAPE'. Press down **PLAY** and **RECORD** together until both keys click and stay down. The Datassette automatically advances the tape enough so it doesn't try to record on the plastic "leader" at the beginning of the tape.
4. The screen will become blank while the computer performs the save. The computer blanks out the screen to avoid timing problems while the tape machine is working.
5. After the program is saved on tape, the 'READY.' signal will appear on the screen, letting you know that you can now give the computer other commands.

The program is now saved on tape under the name "TEST".

The program is now on the tape as well as in the computer. Type LIST [RETURN] to show that the program is still in the computer. Use the VERIFY command to verify that the program recorded on tape is identical to the one in the memory of the computer. The VERIFY command is typed in the same way as SAVE. It instructs the computer to compare the program in its memory with the program on tape. If the two programs are identical, it will say 'OK'. If there is a mismatch, it will say 'VERIFY ERROR', and you must resave the program. To demonstrate the VERIFY command, do the following:

VERIFY

1. Type:

```
VERIFY "TEST" [RETURN]
```

2. When you see the prompt 'PRESS PLAY ON TAPE', press the PLAY switch.
3. If the verification is successful, the computer will print 'OK' on the screen. If there was a mismatch between the program in the computer and the one on tape, it will say 'VERIFY ERROR'. Rewind the tape and resave the program, following the SAVE instructions 2 through 5. Then VERIFY the program again.

If you continue to get verify errors, your cassette tape or Datassette may be defective. Try a new tape. If you still get the error, have your Datassette examined by a service technician.

Cassette users should VERIFY every program SAVE to ensure that no errors have occurred.

After you have verified that your program is properly saved, you can safely erase the program in the memory of the computer by typing NEW [RETURN]. Next, type LIST [RETURN] and you will get the 'READY' prompt but no program is listed; the program is indeed gone from the computer's memory.

LOAD

Now LOAD the TEST program from tape into the memory of the computer by the following procedure:

1. Rewind the tape to the beginning.
2. Type LOAD followed by the name of the program in quotes and hit [RETURN].

```
LOAD "TEST" [RETURN]
```

3. The screen will now say 'PRESS PLAY ON TAPE'. Press the PLAY switch.
4. The screen will become blank while the computer reads the tape.
5. When the computer finds the name of the specified program as it reads the information on the magnetic tape, it displays the program's name on the screen: 'FOUND TEST'. At this time the tape player stops on orders from the computer and waits for you to press any key before loading the program. When you press a key, the tape player will start running again and the screen will become blank as the program is loaded into the computer. If you do not press a key, the tape will begin to load the program after 5 to 10 seconds (a small number of C-64s will wait indefinitely for you to press a key before starting the load procedure. See the KERNEL REVISIONS program described in Chapter 23).

When the computer is waiting for you to press a key, you may choose to stop the load by pressing [STOP].

If the name of the first program on the tape does not match the name you typed in, the computer will cause the Datasette to continue searching the tape for the program name you requested.

If you had not used a program name and had typed only LOAD followed by [RETURN], the name of the FIRST program encountered will be displayed on the screen, and that program will be loaded when you press any key.

If you want the program to be loaded and then RUN automatically without having to type in the command RUN, press [SHIFT] [RUN/STOP] instead of entering 'LOAD'.

You may terminate a load at any time by pressing [STOP].

When a program is loaded into the computer, a copy of the program is formed in the computer's memory. The tape still retains the original saved program. When you turn the computer off, the copy is lost from the memory of the computer, but the program is still on the tape.

1. As you type in a long program, you should save the material you have typed every 20 to 30 lines. This will ensure against loss of the program due to a power failure or other disruption of the normal electrical supply.

TAPE TIPS

2. Make at least one backup copy of each of your programs. The tape medium is susceptible to error; safeguards must be taken to prevent the complete loss of important programs.
3. When you have perfected the final version of a program, rewind the tape completely and then SAVE the same program over and over. If you are using C-20 ten-minute tapes, the best for recording programs, you can fill the entire side of the tape with copies of the program. You may do multiple SAVES easily by typing in:

```
FOR F=1 to 5: SAVE "STARWARS":NEXT F
```

Now you can rewind the tape and do all the VERIFYs.

After you load in the first program on this specially prepared tape, the computer will automatically send a signal to stop the tape machine. The tape is now at the beginning of the second copy of your program, ready for another load. This system eliminates the need for rewinding the tape to get to the beginning of a program after you have loaded it once before.

4. You can obtain a screen listing of the names of all the programs on tape by typing LOAD"ZZZ". Since none of your programs has that name, the screen will print the names of all the programs that are encountered as the computer searches for "ZZZ".
5. Don't use chromium dioxide tape, it will wear out the head too quickly. Don't use extra high-quality tapes, especially those called "computer quality"; they are far more costly than necessary. Use ferric oxide tape instead. Moderately priced ferric oxide tapes cost \$2.00 or less. However, don't buy unknown brands that are very cheap. If you buy the screw-assembled cassettes, you can take them apart if necessary.
6. Clean and demagnetize the head of the Datasette after each eight hours of use.
7. Fast forward and rewind new tapes from beginning to end several times before you use them. This reduces binding and sticking as the tape unwinds from the spool.
8. Erase old tapes before reusing them so they are noise free.

9. Keep your cassette player away from TVs and monitors. And keep your tapes away from TVs, fluorescent lights, speakers, and anything else that might be magnetic.

Because the C-64 has so much memory, many of the programs available are long and take a lot of time to load and save on tape. Purchasing a disk drive will save you many hours of time waiting for the tape drive to complete its task.

ONE FINAL TIP

The Disk Drive and Diskettes

The VIC-1541 Single Drive Floppy Disk (Figure 8-1) is a mass storage device that is fully compatible with the Commodore 64 computer. It is used to store programs and data on disks, also called *floppy diskettes*.

THE 1541 DISK DRIVE

The 1541 is called an “intelligent” peripheral because it has its own 6502 microprocessor as well as ROM and RAM chips. It also houses the drive mechanism that spins the floppy diskette at 300 revolutions per minute past the read/write head. The head is similar to those in tape recorders. The head of a tape recorder can read the music on a cassette tape in the “play” mode, and also record, or write, music to a tape when it is in “record” mode.

An Intelligent Peripheral

The 1541 is self-contained. All you need to connect it to the C-64 is the cable that was supplied with the drive unit. With the memory chips built into it, the 1541 can operate without using any of the computer’s memory. This leaves all of the computer’s RAM for your programs. Disk drives that do not have “intelligence” built into them require some of the computer’s memory for operation.

The TEST/DEMO disk that came with your 1541 disk drive has valuable programs on it. DO NOT try to save other programs on the TEST/DEMO disk. The tape that covers the notch on the diskette prevents you from accidentally recording, or saving, information on it. It is *write-protected*, so don’t remove the tape. Two of the programs on the TEST/DEMO disk give some brief instructions on how to use the disk drive. Additional programs on the diskette provide useful utilities that make it easier to use the disk drive.

Demo Disk

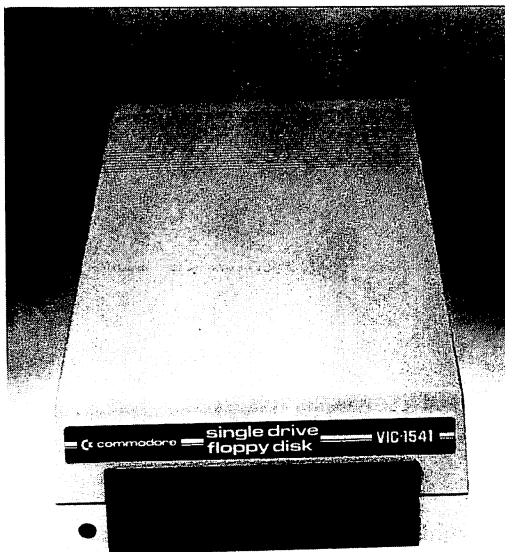


FIGURE 8-1. The 1541 disk drive.

(Courtesy of Commodore Business Machines, Inc.)

Other manufacturers' drive units cannot be used with the C-64 unless they were specifically designed for C-64 compatibility. Other Commodore disk drives such as the 4040 Dual Drive (Figure 8-2) and the 2031 Disk Drive (identical in appearance to the 1541) may be connected to the C-64 with a proper interface. It is also possible to connect a *hard disk drive* to the C-64 with the proper interface. These units use a rigid disk which can hold 5 to 10 megabytes of information, nearly 100 times more data than the 1541. Hard disk drives are very expensive and are rarely used with the C-64.

Other Disk Drives

There are two serial bus sockets at the back of the 1541. Insert one end of the drive cable into either of these sockets and the other end into the *serial bus* socket at the rear of the C-64, as shown in Figure 8-3. If you are connecting only one 1541, you can plug the printer cable into the second socket of the drive unit.

CONNECTING THE 1541 DISK DRIVE



FIGURE 8-2. The 4040 dual drive.
(Courtesy of Commodore Business Machines, Inc.)

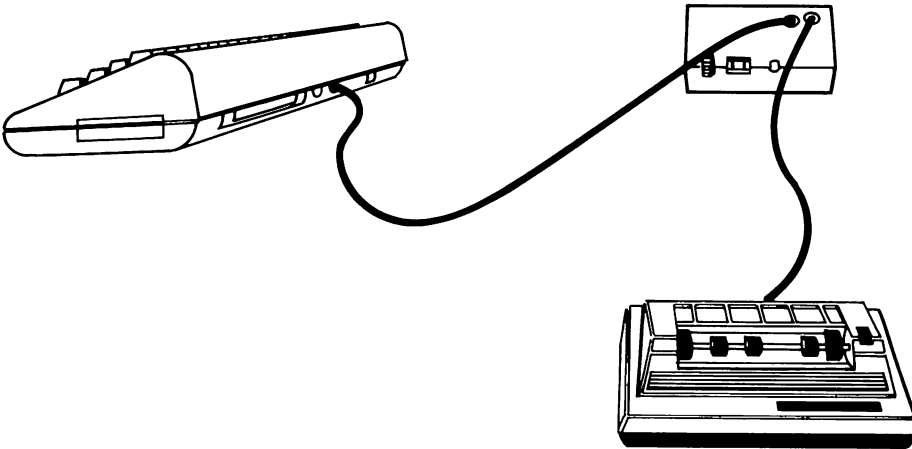


FIGURE 8-3. One drive connected.

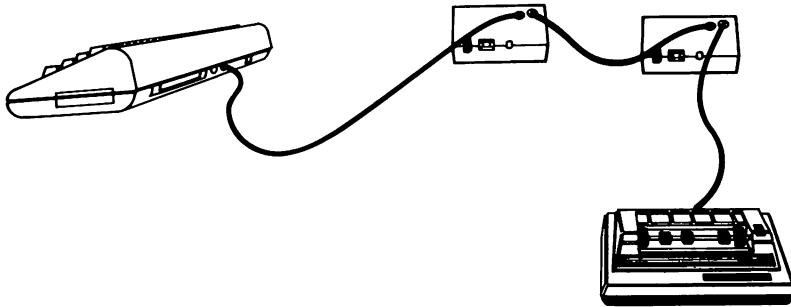


FIGURE 8-4. Two drives connected.

You can connect up to four disk drives by “daisy-chaining” them together. Figure 8-4 shows how two drives and a printer are connected.

The 1541 uses a cable called a serial bus to connect with the C-64. Data can move both ways along the serial bus: from the disk drive or other device to the C-64, or from the C-64 to the disk drive. Data moves in serial fashion along this bus, each piece of data following the previous piece like soldiers marching in single file.

The Serial Bus

Another type of cable connection between the computer and its peripheral devices is the *parallel cable*. The *IEEE bus* used by Commodore on its PET computers uses a parallel configuration. Data transfer with parallel devices is faster than with serial. Using our analogy again, data moves along the bus in parallel, like soldiers marching several abreast. Just as more soldiers will arrive at their destination sooner when walking several abreast rather than serially, or single file, more data can move over the parallel bus in a fixed period of time.

The Parallel Bus

The MSD single disk drive and dual drives also use the IEEE bus. With a proper IEEE interface, these MSD drives can be connected to the C-64. Data transmission with these drives is faster than with the 1541.

Programs that store data in the form of files on a floppy diskette (also referred to as disk for short) usually require two disks, a *program disk* and a *data disk*. When you run such a program with one disk drive, you must constantly remove one diskette to insert the other. When two

USING TWO DISK DRIVES

drives are used, one contains the program disk and the other the data disk. This access to both program and data without changing disks saves time. Serious business applications that require large data entries will need two disk drives for efficiency.

A dual disk drive is a pair of disk drives housed in one case. One drive is identified as drive #0 and the other as drive #1. We use these numbers to specify which drive we want to access. The two drives share one microprocessor built into the unit. With a dual drive such as the MSD, you may *back up*, or copy, disks from one drive to another directly. Data from the first disk drive is not loaded into the computer before being saved on the second disk. This method of backing up a diskette is more than 10 times faster than using two separate drives or a single drive.

Dual Disk Drives

Two single drives have two separate microprocessors. Using two single disk drives daisy-chained together does not give you the same configuration as a dual disk drive. With two single drive units, one drive is identified as device #8 and the other as device #9. But the computer considers both single drive units to be drive #0. To back up a diskette with two single drives, the data from a diskette in one drive must be loaded into the computer and then saved on the diskette in the second drive unit. Consequently, this indirect back-up technique is much slower than back-ups performed on dual drive units.

Two Single Disk Drives

Another factor that contributes to the speed of a dual drive versus two 1541 disk drives is the difference in the types of connecting cables used. The 1541 uses a serial bus cable; the 4040 and MSD dual drives use a parallel cable. As noted above, the serial bus transmits data more slowly than the parallel bus.

Just as the C-64 computer requires an operating system to interpret your commands and to execute them, the 1541 disk drive needs a *Disk Operating System* (DOS) to tell it how to perform disk commands. The DOS (rhymes with toss) is made up of programs built into the 1541 as ROM. It keeps track of where your various files are on the diskette and can fetch them for you quickly and automatically.

DISK OPERATING SYSTEM (DOS)

Many other manufacturers have designed their disk drives to store the DOS on a diskette rather than as ROM. This DOS must be loaded into the computer each time the computer is turned on. Typically, the DOS is stored on each diskette. You can see that this type of system uses valuable memory space in both the computer and the diskette. With a Commodore system, all the space on each diskette is available for data storage, except one area reserved for housekeeping.

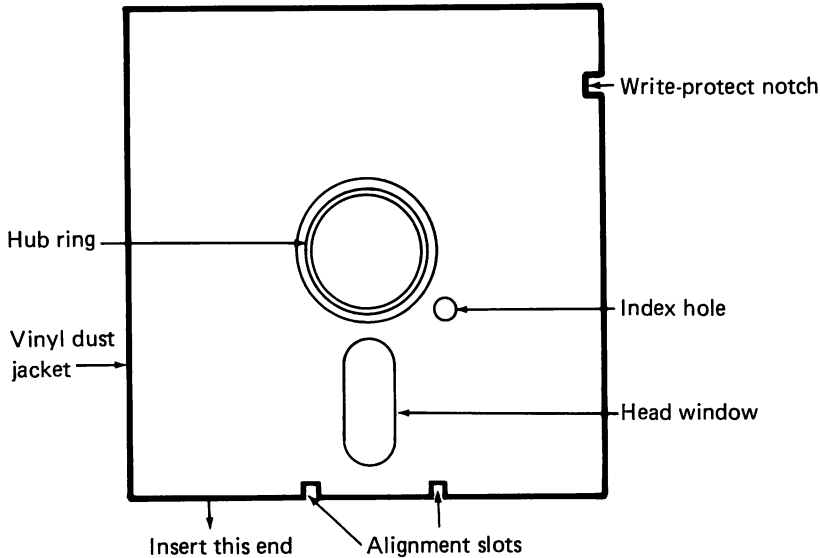


FIGURE 8-5. The floppy diskette.

Use 5-1/4 *single-sided, single-density, soft-sectored* floppy diskettes with *hub rings*. Figure 8-5 shows the major parts of a diskette.

FLOPPY DISKETTES

Types of Diskettes

The 1541 disk drive can access only one side of a floppy diskette at a time. *Single-sided* diskettes are designed and tested to store data faithfully on only one side even though both sides are coated the same way. Although double-sided diskettes may be used with the 1541, they are more expensive than single-sided diskettes; so using single-sided diskettes makes better sense. See page 72 for instructions on how to use the second side of a single-sided diskette.

Single-Sided

Some disk drives are designed to pack data more densely on *double-density* diskettes than is possible using single-density diskettes. The 1541 cannot do this. Although you can use double-density disks as single-density, double-density diskettes cost more. You can save money without any loss in performance or efficiency by purchasing single-density diskettes.

Single-Density

Your phonograph records may be described as *hard-sectored*. This means that the tracks (cuts or songs) are predefined by the manufacturer; you cannot change them. Similarly, hard-sectored floppy disk-

Soft-Sectored

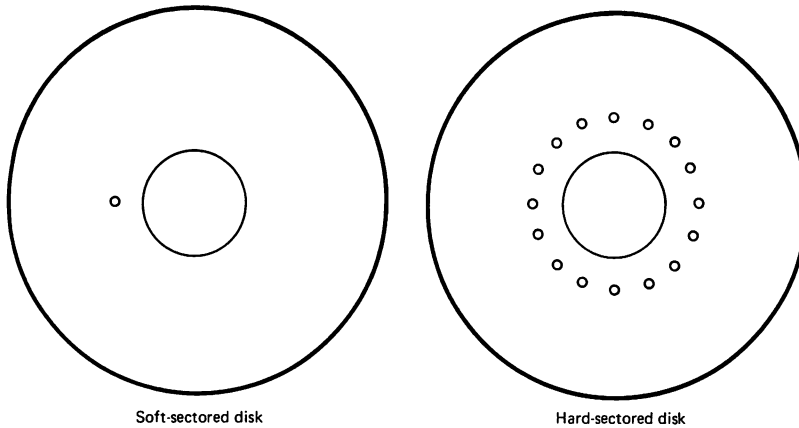


FIGURE 8-6. Soft- and hard-sectored disks.

ettes have their track locations predefined. Hard-sectored diskettes have holes punched between sectors as shown in Figure 8-6. The disk drive senses these holes and thus knows where each sector begins and ends. Soft-sectored diskettes have only one punched hole. The Disk Operating System of the disk drive defines the various sectors of the soft-sectored diskette; they are not predefined. The 1541 uses soft-sectored diskettes. Hard-sectored diskettes may also be used on the 1541, but they are more expensive.

Hub rings are similar to reinforcements that are glued around holes in sheets of paper for three-ring binders. The ring protects the rim around the center of the diskette from damage. When the door of the drive unit is closed, a circular assembly fits into the center hole to grip and rotate it. Sometimes if the diskette is not seated properly in the drive unit, the gripping action may crimp the area of the diskette around the hole. This will cause the diskette to be seated improperly on another occasion, resulting in disk errors. Buy diskettes with the hub rings installed. There are kits for installing hub rings on diskettes that do not have them.

Hub Rings

Every floppy diskette must be formatted before data can be stored on it. A blank diskette is like a blank piece of paper. If we want to arrange our words on a page neatly, without running them together, we can draw lines on the page to guide us as we write. This will make our writing easier to read. Similarly, the 1541 formats the diskette, creating a series of *tracks* and *sectors*. A total of 35 tracks are created; they are like the “cuts” or songs on a phonograph record except that tracks are single, circular rings rather than one continuous spiral inward (Figure

Organizing Information on Diskettes *Formatting*

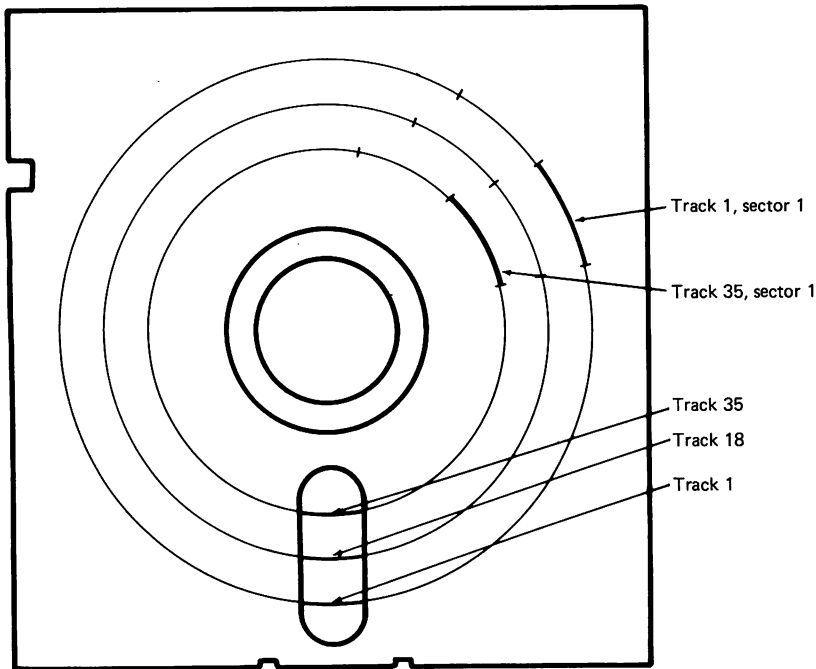


FIGURE 8-7. Tracks and sectors.

8-7). Sectors are sections of tracks. Formatting a Commodore diskette divides outer tracks into more sectors than inner tracks, since the outer ones have a greater circumference. Each sector can store 256 bytes of data.

One side of a 5-1/4" floppy diskette can store 174,848 bytes of data. This amount of data is equivalent to well over 120 pages of this book. A maximum of 142 different files can be stored on one side of a diskette. This means that you could theoretically store 142 very short files on a diskette. Each side can hold 664 blocks of data, each block equal to 256 bytes of data. Double-sided diskettes can store more than twice as much data.

Capacity

The disk *directory* is much like a book's table of contents (see Figure 8-8). The directory is updated automatically by the DOS each time a file is added or deleted from a diskette. To view a screen display of the directory of any diskette, use the appropriate command, described on page 77.

The Directory

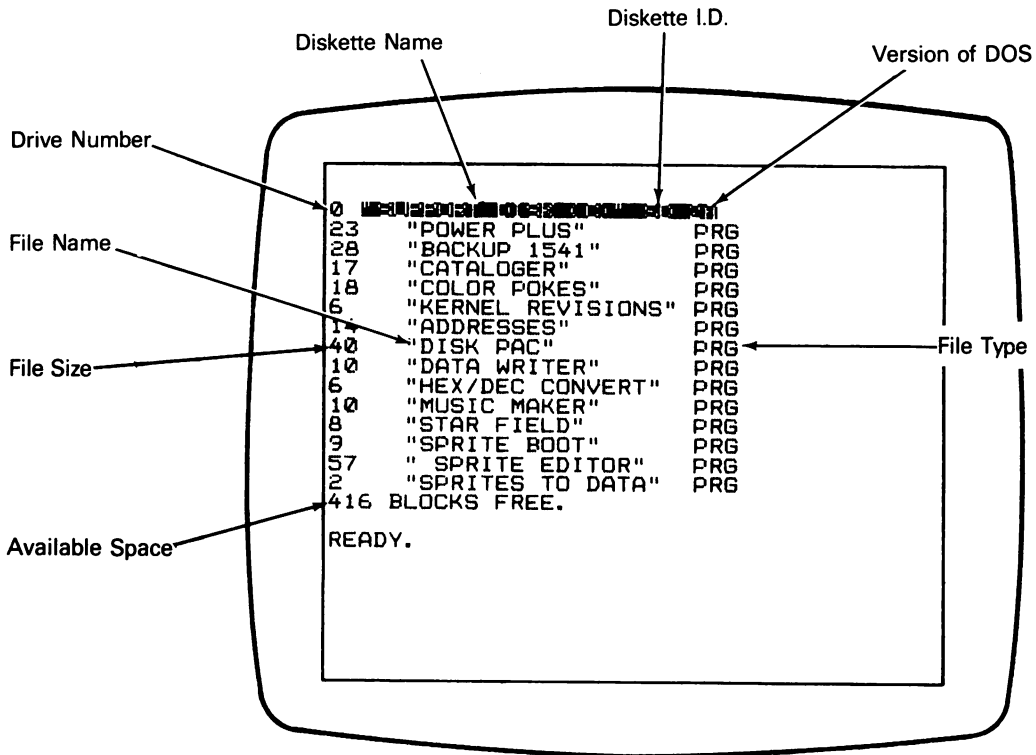


FIGURE 8-8. Diskette directory

The BAM of a diskette is located in the center of the diskette on track 18, as shown in Figure 8-7. The central location of the BAM enables the disk head to move quickly to the BAM from any position on the disk. This is important because the BAM must be accessed frequently.

*Block Availability
Map BAM*

The BAM area contains all the housekeeping data for the diskette. The DOS uses it to keep track of where all files are stored on the diskette, their names, and their length. The directory is located in the BAM area of each disk. Track 18 also contains an updated *directory header* made up of the name of the diskette and its ID number. Any single program or data file may have its parts stored in scattered sectors all over the diskette; the parts don't have to be saved on consecutive sectors. The DOS allocates the space for each segment and records on the BAM each location of the scattered parts of a file. The DOS can use these *pointers* to link all parts of a program or data file so they are recombined in the proper order when a LOAD of that file is requested.

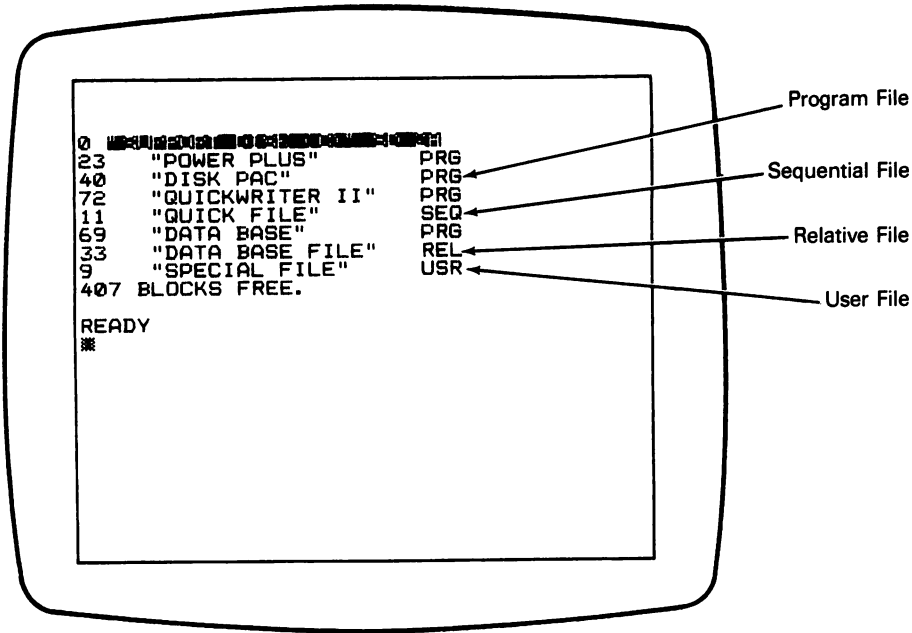


FIGURE 8-9. File types.

A file is any group of data stored under a single name on a diskette. There are four types of files:

1. PRoGrams (PRG)
2. SEQuential files (SEQ)
3. RELative files (REL)
4. USer files (USR)

Most of the time you will load and save program files. Some programs such as database and word processor programs will store the data that you type in as *sequential*, *relative*, or *user* files. Of these three types, sequential files are the most common.

Figure 8-9 illustrates how the file type is indicated to the right of each file name, as either PRG, SEQ, REL, or USR.

Programs, such as the ones you've been running, are files of instructions and statements that may be loaded into the computer directly.

FILES ON DISKETTES

Programs

Data files such as sequential files (indicated by the SEQ to the right of the file name on the disk directory) can be loaded only through a program that has already been loaded into the computer and is running; the program handles the loading and saving of these data files.

Data files are similar to manila file folders in which we store documents. To begin to store information in a diskette data file, the file must be opened and given a name. How is this done? By typing commands to open the file, or by using a program with these commands already in it. Then information is put inside the file as a series of *records*. A record is the information that might be on one piece of paper in the file folder. For example, if the file holds the names and addresses of clients, a record is the name and address of one client in the file. A file, then, is a group of records.

A *field* is one part of a record. In the example above, each element of one client's file is a field. The name is the first field, the street address is the second field, and the city/state/zip is the third field.

If you use a program to open a disk file to enter some records into it, the program will then close that file when you are finished. Otherwise, you will have to close it by typing in a closing command. For a detailed description of data files, refer to Chapter 16.

Data Files

Sequential files store data in the form of records, byte by byte, in sequence, just as data is stored on a length of cassette tape. Should you want to retrieve a record in the middle of a sequential file, the file must be searched by the program from the beginning of the file, in sequence, until the required record is found. This makes sequential files rather slow, especially for large applications.

Another disadvantage of sequential files is that the memory space of the computer limits their size. In order to manipulate a sequential file, the entire file must be loaded into the computer. If you keep adding to the sequential file on disk or tape, it may become too large to fit into the memory of the computer.

A third disadvantage is that once you start storing data in a sequential file, you must load the entire file to add or change data, and then resave the entire updated file. This also makes sequential files slow and inefficient for large business applications. Sequential files are discussed in more detail in Chapter 16.

Sequential Files

Relative files overcome many disadvantages of sequential files. Since all records in a relative file are the same length, it's easy for a file-oriented program to keep track of where each record is with a short mathematical calculation. Any particular record in a relative file may be accessed directly, without having to work from the beginning of the

Relative Files

file sequentially, from one record to the next, until the required record is found. Just as we can play the sixth “cut” or song of a phonograph record by placing the needle at the beginning of that cut, we can retrieve the sixth record of a relative file without having to start at the beginning, as we must with a sequential file arrangement. Locating records in a relative file directly, without having to search sequentially from the beginning of the file, is called a random process. This speeds up the accessing of data from a relative file.

Relative file records are the same length whether all the space in a record is used or not. This allows us to change a record in the middle of the file without having to redesign the entire file. We can even add records to the end of a relative file. A relative file’s length is not limited by the memory size of the computer; it is limited only by the space available on the diskette.

For applications involving large files and the need to update records frequently, relative files are more desirable than sequential files. However, since all relative file records must be the same length, records with little data will waste disk space. Relative files do not economize diskette space as do sequential files. Relative files are discussed more fully in Chapter 16.

User files (not to be confused with the BASIC command `USR`) are rarely used. The format for user files is almost the same as that for sequential files.

User Files

Remember that whenever you see `SEQ`, `REL`, or `USR` next to the name of a file on a disk directory, this file **CANNOT** be loaded into the computer directly. A program that accesses and creates such files must be loaded and run. Database and word processor programs use such data files.

A Reminder

All diskettes are potentially double-sided. If you punch a hole using a single-hole paper punch opposite the write protect notch on your diskette, you’ll be able to format and record programs and data on the flip side (be warned, however, that such action voids the diskette warranty). If you decide to try this money-saving technique, understand that the manufacturer of the diskette does not always test for the reliability of the medium on the flip side of disks sold as single-sided.

FLOPPY TIPS

The Flip Side

We’ve found that when you use a diskette for some time, the dust and other particles that are picked up by the insulating material inside the jacket of the diskette become arranged in a configuration determined by the direction of rotation. When you flip the diskette over, reversing the rotation, the dust dislodges and results in errors during loading or saving.

We've also found that it can be difficult to keep track of programs on disks that have been recorded on both sides. Imagine going to the library to look for a book only to find that to save money, two books were bound inside one cover. However, many people do use both sides of their disks without complaint. The decision is yours.

It is important to use unique ID numbers when you format disks. This will help you keep track of your diskettes and prevent errors that can occur when you change diskettes in the drive unit frequently.

When you access a diskette by getting a directory or by loading a program, the DOS reads the diskette's BAM and stores it in the memory of the disk drive. The drive now knows where all the data is stored on the diskette. It also knows the locations of all empty spaces available for storing new data. If you change diskettes, inserting a second diskette with the same ID number into the same drive unit, the DOS may not realize that a new diskette has been inserted. This can cause some devastating problems. For example, if you try to save a program onto the second diskette, the BAM of the first diskette, which is in the memory of the DOS, may allocate space that was free on the first diskette but is occupied on the second. The SAVE may write over an important file on the second diskette. You can avoid this problem by using unique ID numbers.

Plan your ID coding system carefully. ID numbers can be made up of only two characters—letters, numbers, or both. When you use letters of the alphabet (26 possible characters) rather than numerals (10 possible characters), you'll have a wider range of possible character combinations for identification purposes. For example, you may use UA to UZ for utility disks, GA to GZ for game disks, BA to BZ for business disks, etc. This will give you different ID numbers for up to 26 disks in each of 26 categories.

If you have the SUPPORT DISK available with this book, you can change the ID numbers of your diskettes without having to reformat them by using the disk utility program DISK PAC, described in Chapter 23.

Always make a second copy, or *back-up* of your valuable disks. The BACKUP 1541 program on the SUPPORT DISK will allow you to copy the entire contents of a disk. This program is described in Chapter 23.

1. Keep disks away from sources of magnetism, heat, smoke, and dirt.
2. Store each diskette in its sleeve whenever possible.

ID Numbers

Unique Identification Numbers

Coding Your Disks

Back-up

Care of Floppy Diskettes

3. Don't touch the exposed parts of the floppy diskette with your hands.
4. Don't bend the disk as you put it into the drive. Work slowly and carefully. Do not force the lever that closes the door of the drive if it resists. Just remove the disk and reinsert it.
5. Don't turn the power to the disk drive on or off when there's a disk inside, or you may lose some information from the disk.
6. As long as the red light of the disk drive is off, it's all right to leave a disk in the drive when you turn the computer on and off to reset it, or when you use the reset switch you may have installed on the computer.

Commercial program diskettes are frequently formatted on disk drives other than the 1541. Some of them may cause problems when you try to access them. The red light of the disk drive may flicker on and off while the disk drive makes strange sounds. This usually means the drive is having problems reading the BAM. If, after several attempts, the drive still can't read the BAM properly, the drive will quit with its red light flashing. Read the error channel using the technique described on page 77 to discover the type of error encountered. (A list of disk errors appears in your disk drive manual.) If a commercially formatted disk won't load successfully, ask the retailer or the manufacturer for a replacement.

Diskettes formatted on the 4040 Dual Drive may load properly with a 1541, but will cause problems when you try to save programs or data on them.

To avoid the problems described here, copy disks formatted on other disk drives using a backup program that will allow you to copy the entire contents of a diskette onto another diskette using only one disk drive. If you have the SUPPORT DISK, use the BACKUP 1541 program described in Chapter 23.

The LOAD command instructs the computer to load a named program from the diskette into the computer's memory. Use the following steps to perform a LOAD:

1. Insert the floppy disk that contains the program you want into the disk drive. Be careful not to bend the disk as you're doing this. Now close the door.
2. To initiate a load, type:

```
LOAD "<fn>".8 [RETURN]
```

Using Diskettes Formatted on Other Drives

LOADING PROGRAMS Standard Loads

The <fn> inside the quotes stands for the file's name. The comma 8 tells the computer that you wish to load from disk drive #8 rather than from tape or another disk drive; 8 is the device number of the disk drive. The red light on the disk drive will come on, indicating that the drive is working.

3. The screen now says 'SEARCHING FOR <fn>'. In a moment, it will say 'LOADING', which means that it has found the program and is loading it. When the load has been completed, you will see the 'READY.' prompt and the cursor will reappear.

When you load a program into the computer, you put a copy of the program into the computer's memory. The original program is still on the disk. Listing the program and making changes to the program in the computer will not affect the program on the disk. Don't be afraid to make changes in this way. If you don't like the changes you've made, you can reload the original program from disk.

Usually, if you have a program in the computer, loading another program will eliminate the first program from the computer's memory. Save the first program, if you haven't already done so, before loading another program into the computer.

The above procedure loads BASIC programs into the BASIC memory area of the computer. Some machine language programs, however, must be loaded into the same area of memory that they were saved from. Loading in this way is called an *absolute load*. To perform an absolute load, type the load command as:

```
LOAD "<fn>",8,1
```

The comma 1 (,1) after the comma 8 (,8) tells the computer to load the named program into the same memory location from which it was saved. The BAM keeps track of where in memory it was saved from.

After you've entered a program into the computer, by typing it in, you can save it with the SAVE command. Since most machine programs load into areas of the computer's memory other than the BASIC area, you can't use this BASIC command to save machine language programs. Please refer to page 243 for a description of how to save machine language programs.

To save a BASIC program, type:

```
SAVE "<fn>",8
```

Absolute Loads

SAVING PROGRAMS

The red light will come on as the program is being saved. Do not turn the computer off or open the door of the disk drive while the red light is on. Either of these actions may cause loss of program or data files. If you can't regain control over the computer by hitting [RUN/STOP] [RESTORE] a few times, open the door of the disk drive and then turn the drive unit off and on to reset it.

The wide range of commands you can send to the disk drive makes the diskette an excellent medium for complex storage applications. To get the most from your C-64, you must master these commands.

OTHER DISK COMMANDS

As mentioned earlier, before you can use a new floppy diskette, you must format (also called *header* or *new*) the diskette and thus prepare it for use. Information cannot be saved on a diskette until it has been formatted. A diskette that already has information recorded on it may be reformatted, but the format command erases everything from the diskette, so be sure you no longer require the information on any diskette you reformat.

Formatting

The command for formatting or reformatting a diskette is:

```
OPEN 15,8,15 [RETURN]
PRINT#15, "N0:<dn>,<ID>" [RETURN]
```

- N is the short form for the format command; it stands for NEW.
- 0 indicates that you are using drive #0.
- <dn> Disk Name—the name you wish to give the diskette. It may be any string of 16 characters or less. This name will appear as the header, or title, on the disk directory.
- <ID> is the IDentification you wish to assign the diskette. The ID number also appears on the disk directory. Remember to use a coding system when you format your diskette.

When you press [RETURN] after the second line of the format command, the process of erasing and formatting the diskette will begin. The whole process takes about one and a half minutes. When the red light on the disk drive goes out, formatting is finished and your diskette is ready to store information.

Remember, always make sure you have the correct diskette in the drive before you format, because the formatting process erases any previously recorded files.

You can load the directory (a table of contents) of your diskette into the computer by using this command:

```
LOAD "$",8
```

This command loads the directory into the memory of the computer as a BASIC program, and so it will *overwrite*, or displace, any BASIC program already in the memory of the computer. Once the directory has been loaded, you can display it on the screen by typing LIST. Remember, the directory is only the table of contents of a diskette; you can't run it as a program. From the displayed directory, choose the program you want and follow the instructions for loading programs as outlined on page 74.

When the disk drive blinks its red light, it indicates some kind of error. When you run the program below, the drive will display a message on the screen, telling you the type of error that occurred.

To demonstrate the procedure for reading the error channel, first type NEW (don't confuse this BASIC command with the formatting command, NEW) to erase any BASIC program in the computer. Now open the disk drive door and use the directory command as described above—LOAD "\$",8. Since the door is open, the disk drive will register an error and the red light on the disk drive will blink. To identify the error, enter this program:

```
10 OPEN 15,8,15
20 INPUT#15,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
```

Now RUN the program. You should see a display that looks like this:

```
74,DRIVE NOT READY,00,00
```

The first number is the error number. It is followed by a description of the error. The last two numbers give the track and sector that contain the error. Since the error didn't involve a disk track or sector, zeros are displayed for the track and sector numbers. For a listing of disk errors by number, see your disk drive manual.

Sometimes you'll load a program, make changes, and then want to save the new version in place of the original. This command will save the new version of a program, check to be sure the save was successful, and automatically scratch, or erase, the old version. The name of the file will appear in the same place as before on the disk directory. The command is typed as:

```
SAVE "@0:<fn>",8
```

Using the Directory

Reading the Error Channel

Save and Replace

It's easy to change the name of a program without having to load it.
Use:

Rename

```
OPEN 15,8,15
PRINT#15,"R0:<new fn> = <old fn>"
```

To erase a program from the disk, use:

Scratch

```
OPEN 15,8,15
PRINT#15,"S0:<fn>"
```

If you want to scratch all the PRoGram files, use this format after the OPEN statement:

```
PRINT#15,"S0:*=P"
```

To scratch all SEQuential files:

```
PRINT#15,"S0:*=S"
```

To scratch all RELative files:

```
PRINT#15,"S0:*=R"
```

If the red light of the disk drive flickers on and off, or if you have trouble loading or saving programs, try the Initialize command:

Initialize

```
OPEN 15,8,15
PRINT#15,"I"
```

This will cause the disk drive to read the BAM of the diskette, and store this information in the memory of the disk drive. The drive now knows where to find all the files that are on the diskette.

Occasionally, a disk drive will lock up or "crash" the way the computer sometimes does. When this happens, the disk drive will not respond to your commands. You may correct this condition by using the reset command (U;) to reset the drive. This is similar to turning the disk drive off and on again, but the advantage is that it saves wear-and-tear on the drive power switch.

Resetting the Drive

```
OPEN 15,8,15
PRINT#15,"U;"
```

You may also reset the drive by turning the drive's power switch off and then on. Make sure you open the door of the drive unit first, so that the read/write head is no longer on the diskette surface.

When you have saved and scratched many files, the areas on the diskette can become disorganized, leaving scattered spaces that are not being used. You can consolidate these spaces and free up some extra space by using the Validate command shown here:

```
OPEN 15,8,15
PRINT#15,"V"
```

Before you use the validate command, list the directory and note the number of blocks free on the diskette. If the validate command accomplishes something for you, there should be more blocks free after validation than before.

There are ways to reduce the amount of typing you must do to enter disk commands. Some of these short cuts are described below.

Use the abbreviations for BASIC commands as they appear in Appendix B. For example, L[SHIFT]O for LOAD. Note that the shifted letter appears as a graphic character, and looks like this:

```
L┐
```

You may use shortened versions of file names by using the asterisk (*) as a delimiter. The asterisk stands for the letters omitted. Instead of typing:

```
LOAD "MY PROGRAM",8
```

you could type:

```
LOAD "MY PR*",8
```

The '*' stands for the missing letters. You can put the '*' anywhere you want, but to use the '*' effectively, you have to place it after enough letters to uniquely identify the program you want to load. For example, you could have done it this way:

```
LOAD "M*",8
```

Validate

DISK SHORTCUTS

Abbreviations

Pattern Matching

The first program on the directory that starts with the letter 'M' would be loaded. If a program called MOUSE was before MY PROGRAM on the disk directory, then MOUSE would be loaded. To differentiate MY PROGRAM from MOUSE, you can type the command this way:

```
LOAD "MY*",8
```

If MY PROGRAM was the first program on the diskette that started with 'MY', it would be loaded into the computer.

If you just turned on your computer system, or just reset your drive, when you type the command

```
LOAD "*",8
```

the disk drive will load the first program on the diskette. If you had already accessed a program, the last program you accessed would be loaded in again when you use the above command.

To access the first program on your diskette even after the drive has been used several times, use this command:

```
LOAD "0:*",8
```

In poker, a "wild card" can stand for any other card in the deck. Similarly, the "?" can stand for any character in the name of a file on a diskette. For example, if you're not sure whether the name of a program you want to load contains 'cat' or 'cot', you can use a wild card in the following way:

```
LOAD "C?T POKES",8
```

Wild Cards

You can use the screen editor of the C-64 to enter any command line (such as LOAD "MY*",8) that already appears on the screen by moving the cursor to a position anywhere on that line and hitting [RETURN]. The screen editor can save you time when you're typing in commands. For a more complete description of the Commodore 64 screen editor, see Chapter 12.

The Screen Editor

If you have the SUPPORT DISK, you may use SUPERDOS, a part of the POWER PLUS utility program, to send disk commands. SUPERDOS disk commands, described in Chapter 19, are short and easy to use. They can be helpful when you're sending disk commands. You can use C-64 WEDGE, a program on the TEST/DEMO disk that came with your disk drive, if you do not have the SUPPORT

SUPERDOS

DISK. The syntax of both these utilities is the same. However, **POWER PLUS** gives you many more features than the **WEDGE** program.

When you have trouble loading and saving files, follow the procedures listed in "Initial Troubleshooting" below, starting with the first item. If the first procedure doesn't correct the problem, try the next. Continue in this manner until you find the solution. If your problem persists after you've tried all the techniques listed, continue reading for information on repairs.

1. Open the door of your disk drive and carefully reseal the diskette in the drive, close the door, and try again.
2. Reset your disk drive using the **SUPERDOS** command **@U**; as described on page 222. Without the utility program, you can perform the reset by following the instruction on page 78.
3. Initialize the disk drive using the **SUPERDOS** command **@I** as described on page 221. See page 77 to do this without the utility program.
4. Open the door of the disk drive to eject your diskette; turn your disk drive off and then on, to reset it.
5. The next step is to determine whether the diskette itself is defective. Try another diskette to see if the same problem occurs.

If the problem only occurs with a particular diskette, follow these steps. The **VIC-1541 TEST/DEMO** disk that was provided with the 1541 contains a program called **DISK CHECK**. This program erases all data from the disk and then checks to see whether the diskette itself is defective.

If you suspect that a diskette has a defect, load the **DISK CHECK** program, insert the suspect diskette into the drive unit, and type **RUN** followed by **[RETURN]**. Remember, the program will erase all the data on the diskette, so make sure that you no longer require the programs or data files recorded on it. The program will write to each space on the diskette and then read back the information that was written to determine if the record was written and read accurately. This procedure takes a great deal of time.

You can also use **DISK PAC**, a disk utility program on the **SUPPORT DISK** available with this book, to check a diskette for defects without damaging the data on it. See Chapter 23 for instructions.

IF YOUR DRIVE ISN'T PERFORMING PROPERLY

Initial Troubleshooting

Checking Your Diskettes

You can use two or more 1541 disk drives by permanently changing one drive's device number by cutting jumpers as described in the disk drive manual. If you've used two drives with a 1525-E printer, you may have had trouble accessing the second drive after powering up. The following power-up procedure should eliminate this problem:

1. Apply power to the 1525-E printer.
2. Apply power to drive #8.
3. Apply power to the C-64.
4. Apply power to drive #9.

Some 1541 disk drives have been modified to avoid the multiple-drive lock-up problem. You may be able to improve your performance of the disk drive by replacing a ROM chip. Check with your Commodore dealer.

If you can't fix the problem you're having with the drive unit, run the PERFORMANCE TEST program on the TEST/DEMO disk. This program will test some aspects of the hardware and report drive failures.

When the drive unit is on, heat builds up inside the case and escapes through the vents on the top and bottom surfaces of the drive unit. If the disk drive gets too hot, metal parts may expand enough to cause alignment problems. Here are several ways to prevent heat build-up:

1. Always keep the ventilation slots of the disk drive uncovered.
2. Attach 5/8" plastic cups to the bottom of the unit to elevate it. This permits more air to move into the vents underneath, through the drive unit, and out the vents on top.
3. For best results, place a small, quiet fan over the vents on top of the unit to force cool air into the unit. Some filter material, foam or fine fabric, can be used to prevent the forced air from carrying dust into the unit.

Every four months or so, the head of the drive unit should be cleaned with a commercial disk drive head cleaning kit available from computer supply stores.

CAUTION: Any tinkering with hardware may void its warranty. If you're not familiar with electronics, we advise you to have a service technician perform some of the operations described below.

Multiple-Drive Lock-Up

THE NEW 1541 DISK DRIVES

Testing the 1541

CARE OF THE DISK DRIVE Heat Problems

Head Cleaning

Lubrication

If you hear a prominent squeaking sound when the drive unit is spinning the diskette, the spindle of the drive may need lubrication. Open the disk unit by removing the four bottom screws. Take out the screws that hold the circuit board. Look for the spindle above the center post of the drive motor. Place a small dab of lithium grease (available at bicycle shops) on the spindle.

The Spindle

We have found that problems with loading and saving programs can frequently be prevented if the rails on which the disk drive head moves are lubricated with a cotton swab dipped in lithium grease. Locate the pair of rails, and apply the grease sparingly.

The Rails

The following steps should be carried out only if the procedures suggested above do not correct a problem. Work at night so that the room you're working in may be darkened. You'll need a fluorescent light (not an incandescent light).

Adjusting the Timing

1. Take out the four bottom screws from the drive unit and remove the cover.
2. Take out the screws that hold the drive assembly to the bottom of the plastic case.
3. Lift up the drive assembly and look for the timing pattern shown in Figure 8-10.
4. Darken the room and turn on a fluorescent light near the drive unit.
5. Attach the disk drive to the computer, insert a diskette, and type in the command for getting a disk directory (LOAD "\$",8). As the drive unit spins the diskette, study the strobe pattern at the bottom of the drive unit. One of the rings of patterns should appear to stand still.
6. If the ring moves slowly one way or the other, it means the timing of the drive motor needs adjustment. Insert a small screwdriver in the pot near the strobe pattern and turn the screwdriver in one direction or the other until the pattern stands still.

If none of the above fixes work, the drive unit's head may require re-alignment. This should be done only by a trained technician.

Head Alignment

If you're still having trouble using your disk drive, have a trained service person examine it.

Servicing

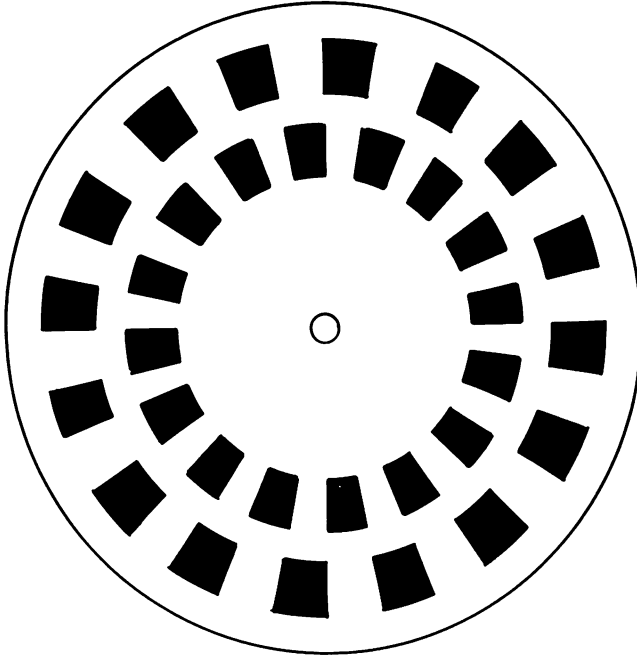


FIGURE 8-10. Timing pattern.

The Printer

The printer is the peripheral device that will probably complete your computer system. The printer provides output from your computer (referred to as *hard copy*) in permanent physical form—documents, program listings, and the results of computations. These results on paper are the only hard copy records of the temporary screen displays.

Selecting a printer should not be taken lightly. Think of the printer as you would the speakers of your stereo system. You may have a terrific stereo with the potential for true high fidelity sound, but without quality speakers, you will never hear the excellent sound it is capable of producing. Similarly, your printer's output is the end visual result of the computer's work, whether it be financial computations or word processing. If the appearance of your printed output is important, be sure your printer is capable of producing high quality print. The discussion of printers that follows will help you research and select the right printer for your specific needs.

First, we'll discuss three of the most popular types of printers: dot matrix printers, impact printers, and electronic typewriters. Second, there is a short discussion of features to consider when you compare printers. If you already own a printer, refer to Chapter 13 for a discussion of how to use it.

A dot matrix printer uses a matrix of pins to make patterns of dots on paper. These dots define the text and graphic characters that are output from the computer. Dot matrix printers are popular because of their low cost and great versatility. Since each wire in the head of the printer can be programmed individually, the printer can produce almost any form of output, from high resolution graphics to custom characters, with various print styles, print sizes, and emphasized characters. The sample output, Figure 9-1, was printed on an Okidata 92 dot matrix printer.

THE IMPORTANCE OF THE PRINTER

TYPES OF PRINTERS

Dot Matrix Printers

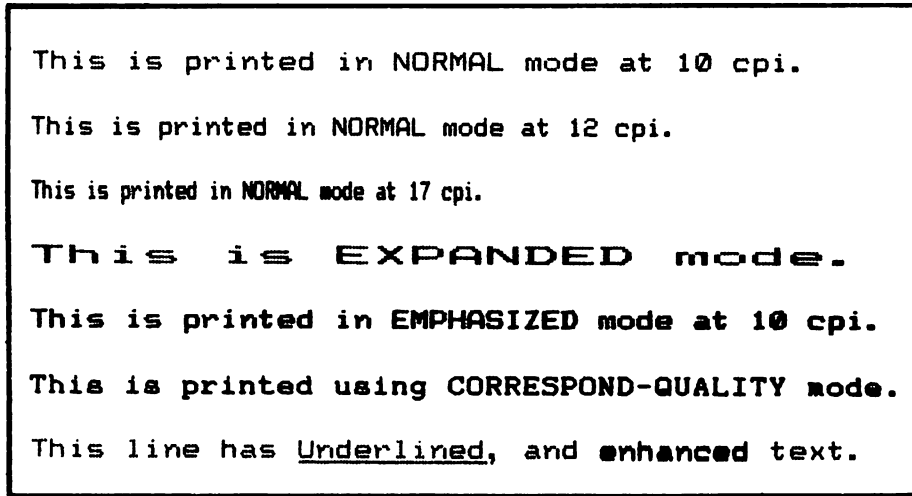


FIGURE 9-1. Sample dot matrix printout.

Impact printers (also called letter-quality printers) use fully formed, molded characters for printing that look like traditional letters. A hammer taps the molded letter of the element against the ribbon and paper, as it would for a typewriter. Those who plan to produce business correspondence should consider obtaining an impact printer.

Impact Printers

The current explosion in the production of electronic typewriters (not to be confused with electric typewriters) has some implications for computer users. An electronic typewriter has a microprocessor and memory chips built into it, making it an intelligent typewriter and text editor. These typewriters use impact printing and consequently produce high quality print.

Electronic Typewriters

Some electronic typewriters (Figure 9-2) are designed to be connected to computers for use as printers as well as to be used independently of the computer as typewriters. With an appropriate interface (Figure 9-2), an electronic typewriter can accept output signals from a computer. These signals go directly to the print head, bypassing the keyboard entirely. When used with a Commodore 64, this interface must be connected to the computer with a second interface that converts the electronic typewriter's parallel output to serial output (see the discussion of interfaces later in this chapter).



FIGURE 9-2. Electronic typewriter/interface.

Compare costs carefully. The cost of the electronic typewriter and both interfaces may be about the same as the cost of an impact printer. The hardware package, however, will give you the capability of using the unit both as a printer and as a typewriter separate from your computer.

Many factors influence a printer's quality. The important ones are discussed below. The discussion will help you compare capabilities of different printers.

Printers produce either dot matrix or fully formed characters. Fully formed characters are suitable for all applications, from program listings to formal documents. The output of dot matrix printers has a characteristic "grainy" appearance which may be unacceptable for some correspondence applications. For this reason, some of the better dot matrix printers offer a "double strike" mode for correspondence-quality output. In this mode, they print each line in two passes to fill in

PRINTER CHARACTERISTICS

Dot Matrix versus Impact Printers

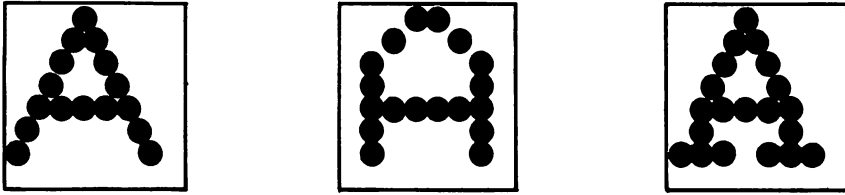


FIGURE 9-3. Dot representations of 'A'.

the spaces between dots. This usually slows the printer by a factor of two or more, but the results resemble fully formed letters more than the results of regular print mode.

Each printer has its own look. When you compare the output from several dot matrix printers, you will see that they produce characters of different styles. The clarity, or definition of the characters, will vary from printer to printer. Study samples from several printers until you find one that you like. Figure 9-3 shows three ways of representing the letter 'A' using a dot matrix.

Character Styles

The clarity of a dot matrix printer output depends on the density of pins in the matrix used to create characters. Some printers use a matrix of 5×7 pins, while others use a 9×9 or even denser matrix. The more pins used in the description of characters, the higher the resolution, or quality, of each character. Some of the more expensive dot matrix printers use a 9×24 dot matrix, which can produce output that closely resembles the output of impact printers.

Clarity

Some less expensive dot matrix printers, such as the CBM 1525-E (Figure 9-5), print lower case letters with tails ('g', 'j', 'p', 'q', and 'y') by placing the entire character above the line, as shown below:

True Descenders

Many users find these non-descending characters objectionable. If you

This is what non-descending characters look like:

gan9 jumP plump quick yawn

FIGURE 9-4. Non-descending characters.

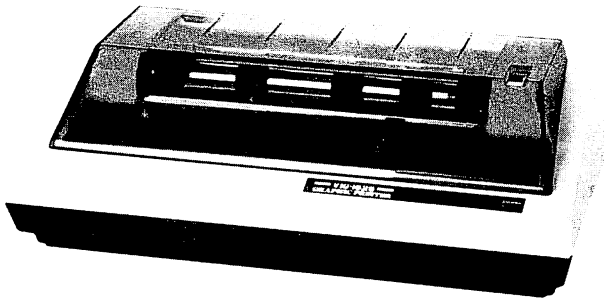


FIGURE 9-5. The Commodore 1525-E printer.
(Courtesy of Commodore Business Machines, Inc.)

feel this way, be sure the printer you purchase has true descenders.

When you want *justification* of your text so that text is aligned vertically at the right margin as well as at the left margin, many word processors put extra spaces between words to spread your line of text out to the right margin.

Some printers can produce text with *proportional spacing*, which means that different letters occupy different amounts of space. With a word processor that supports this feature of the printer, justified text can be spread out letter by letter. This smoother form of spacing is visually more pleasing. Most published books, for example, use proportionally spaced, justified text.

Proportional Spacing

Printing speed varies widely among printers. Dot matrix printers are known for their speed. Moderately priced dot matrix printers have speeds that range from 30 to 160 characters per second (cps). Impact printers are slower, with speeds from 10 to 55 cps. At first, even a slow printer seems fast, considering how slowly most of us type. But if you are planning to produce much printed output, you should consider purchasing one of the faster printers. Examine the written specifications of the printers you are considering to determine their speeds.

Printing Speed

Printers that print while moving from right to left as well as from left to right are called bi-directional printers. These printers are faster than those which print only in one direction.

Bi-Directional Printing

A bi-directional printer must be able to keep a full line of text in memory so it can type a line “backwards” as it is moving from right to left. Such printers have a *buffer*, or memory space, which can hold text. Some buffers are larger than others, permitting the storage of larger amounts of text in memory. A printer with a large buffer frees up the computer so you can continue to use the computer while the printer is printing out data stored in its buffer memory.

Buffers

It is possible to purchase add-on buffers with larger capacities so that several pages of information can be stored and printed out without tying up the computer.

Friction feed and *tractor feed* are the most common types of paper feed mechanisms used by computer printers.

Paper Feed

With friction feed, you can insert single sheets of paper just as you would with a regular typewriter. This will allow you to print on any kind of paper that fits the carriage of the printer.

Friction Feed

The most common form of paper for computer printers is continuous paper feed. A major industry standard is 11” sheets of fan-folded, perforated paper that are 9½” wide. On each side of this paper is a strip of sprocket holes that allows the tractor feed mechanism of the printer to pull the paper through as it is printing. After printing, you may tear off these strips and separate the sheets of paper to obtain standard 8½×11” pages.

Tractor Feed

Many dot matrix printers allow you to use either single sheets or continuous feed paper. This flexibility is highly desirable because it permits you to use continuous feed when you print out program listings or rough drafts of documents, and friction-feed sheets for correspondence or final drafts.

Some printers use standard spool-to-spool typewriter ribbons, others require cartridge ribbons. Spool-to-spool ribbons cost less, but can be messy to change.

Ribbons

The printer has many moving parts and a print head that may wear out and require occasional replacement. Consequently, printers may need servicing and maintenance frequently. When you have narrowed your choices to two or three printers, consider these points:

Durability and Service

1. Purchase the printer that seems more sturdy and durable than the others.

2. Pay attention to the length of the warranty period offered by the manufacturer, and purchase the printer with the longer warranty period whenever possible.
3. Buy your printer from a reliable local dealer who offers servicing for the equipment that he sells. Your printer is one item that you probably should not buy from a mail-order firm.

The Commodore 64 computer “speaks” to its peripheral devices such as the disk drive and the printer via a cable called a serial bus. Since a printer operates much more slowly than a disk drive, the slow transmission speed of a serial bus does not noticeably decrease printing speed. The computer industry standard, however, is a parallel bus. Commodore’s departure from the standard parallel bus has caused confusion and frustration for many C-64 owners who connect non-Commodore printers to their computers. We will discuss the advantages and disadvantages of Commodore and non-Commodore printers and provide some suggestions for interfacing your C-64 with the latter.

The easiest way to solve the problem of connecting a printer to the C-64 is to use a Commodore printer designed to operate with the C-64. Commodore printers plug directly into the serial port of the computer if a disk drive is not being used. When a disk drive is being used, the printer is plugged into either of the two serial ports at the rear of the disk drive. No interface is required.

The Commodore printer is capable of printing out all text and graphic characters that are marked on the keyboard of the C-64. Using a CBM printer will save you the trouble of interfacing with a non-Commodore printer.

Users often complain that Commodore printers are slow and that their print quality is not very good. There are printers on the market that certainly outperform Commodore printers in speed and print quality. However, non-Commodore printers require an interface to connect to the C-64. This interface converts the serial bus of the C-64 to a parallel bus so that a proper connection can be made between computer and printer. If you plan to use a non-Commodore printer, read the section below on interfaces and compatibility.

There are several types of printer interfaces for the C-64. Some plug into the serial port, others plug into the user port, and still others plug

CONNECTING THE PRINTER

Commodore Printers

Non-Commodore Printers

Interfaces

into the cartridge port. Your best choice is an interface that plugs into the serial port, because some programs cannot send data to printers via other ports.

There are two types of printer interfaces. One is a dumb interface, the other is an intelligent interface.

These interfaces are less expensive than intelligent interfaces. Dumb interfaces simply convert the C-64's nonstandard signals to the industry standard and convert the serial output to parallel. For most word processing applications, a dumb interface is sufficient.

Dumb Interfaces

To be truly compatible with Commodore printers, an intelligent interface employs a microprocessor along with a ROM chip to give the user full printer compatibility. This means that the printer will act just like a Commodore printer, producing even its graphics. Most non-Commodore printers can be connected to the C-64 with an appropriate intelligent interface. Tymac's The Connection™ and Cardco's Cardprint/+G™ are two examples of intelligent printer interfaces that emulate Commodore printers.

Intelligent Interfaces

Commodore printers designed for the C-64 are fully compatible with all software written for the C-64. When you use a non-Commodore printer, the printer's unique capabilities can be accessed only if the software you run in the computer knows which printer you are using and can adjust the output signal appropriately. For example, your printer can produce italics only if you can send a command through your word processing program to turn this feature on and off. Not all software supports a variety of printers. But if you use a non-Commodore printer and an intelligent interface, your printer will act just like a Commodore printer. Software written to work with the C-64 and the Commodore printer will work perfectly.

COMPATIBILITY

Non-Commodore printers used with dumb interfaces do not reproduce the graphic characters that the C-64 can output. Whenever you try to output a heart (♥), for example, most printers will either not print anything at all, or will print some unexpected character. With an intelligent printer interface, your printer will be able to print out the rich graphic character set of the C-64.

The Modem and Telecomputing

An intricate network of telephone lines connects us all. There are families who have televisions with cable service. There is new interest in satellite communications with the potential for world-wide communication from within our homes. Couple all these with a home computer, and within a few years, everyone can have instant access to the wealth of much of the world's information. Already, a device called the telephone modem, used with a telephone and a computer, is providing thousands of users around the world with the capability of *telecomputing*. Access to cable and satellite communications via the home computer is just around the corner.

THE FUTURE IS HERE

The modem is an inexpensive device that makes it possible to connect any home computer such as the C-64 to very powerful, central computers that contain vast amounts of updated information. Your computer will perform like a million-dollar computer for as long as it is connected to the *mainframe* computer by telephone.

ACCESS TO THE BIGGEST COMPUTERS

A wide variety of information services are available to the computer user with a modem. These information services offer a broad spectrum of textual and graphic information that can be sent quickly to your home computer. This information is easy and surprisingly inexpensive to access. Here is a list of some services available to users:

The Uses of Telecomputing

1. Information—news, weather, encyclopedias, and bibliographies.
2. Personal business—travel reservations, shopping, and banking.

3. Electronic mail—fast mail transmission and fast mail retrieval.
4. Computing—interactive gaming, programming, financial analysis, and word processing.
5. Publishing—fast manuscript delivery, fast editing, and fast typesetting.
6. Bulletin board services (BBS)—local computer news, access to computer programs, classified ads, and help with computer problems.

In this chapter, you will discover how you can use your C-64, a modem, and a terminal program for telecomputing. The modem is a hardware device that sends and receives information through a telephone. The *terminal program* turns your computer into a remote terminal (input-output device) of a distant computer.

Modem and Terminal Software

Your computer processes and stores data as electronic signals. Since telephone lines can only carry audio (sound) signals, the computer's digital information must be converted into audio signals. The modem transforms the telephone's audio signals into the computer's electronic signals and vice versa. Each computer that is to participate in telecomputing must have a modem device attached to it.

THE MODEM

Modem stands for "MOdulator/DEModulator." A *modulator* is an electronic circuit that converts a computer's electronic signals into audio signals and sends these tones through the telephone lines to a receiving computer.

Modulator/Demodulator

A *demodulator* is a circuit that does just the reverse of what the modulator does. It converts the audio telephone signals into the electronic signals that a computer understands.

The modulation/demodulation process that occurs with a modem is much like the process used by your computer system to store data on cassette tapes. During a SAVE, the computer converts the computer's electronic digital signals to analog audio signals and sends them through a set of wires to the cassette recorder, which records the data as a series of "beeps." When you LOAD this data from the cassette tape into the computer, the audio signals are converted back into electronic, digital signals that the computer understands.

The two most common types of modems for home computers are *acoustic* and *direct-connect* modems. Acoustic modems have two rubber cups that fit over the handset of the phone. The cups help

Types of Modems

reduce the amount of noise that the telephone picks up from the room.

A direct-connect modem plugs directly into the telephone line. Direct-connect modems eliminate the problem of room noise entirely.

Commodore markets two inexpensive modems that are compatible with the C-64: The VICMODEM and the 1650 AUTOMODEM.

The VICMODEM and AUTOMODEM

The VICMODEM is probably the most inexpensive modem available for any computer. It is a cartridge that plugs into the C-64's user port. It can be connected to a modular telephone by plugging the telephone wire that normally connects to the handset into the modem. The VICMODEM includes a terminal program and free time on some of the communication services available to telecommunicators. It is not loaded with features, but the VICMODEM and software are all you need to take advantage of the vast number of telecommunications services.

VICMODEM

The AUTOMODEM is a cartridge that plugs into the user port just as the VICMODEM does. However, the AUTOMODEM gives you two additional features: *auto-answer* and *auto-dial*. The auto-dial feature enables your modem to dial the telecommunication service number for you. With the VICMODEM, for example, you would have to use the telephone to dial the number yourself; and when you finished dialing, you would have to disconnect the cord from the handset and plug the cord into the modem. The AUTOMODEM can remain attached to the telephone line; it dials the numbers for you without the need for unplugging and plugging in wires.

AUTOMODEM

The auto-answer feature of the AUTOMODEM enables your computer to receive messages from other computers without you having to be present to answer the telephone. With this type of modem, you could set up your own bulletin board service. People could call your telephone number and obtain or leave messages even when you weren't home. Your computer could receive messages while you slept.

Terminal software temporarily converts your computer into a terminal—an input and output peripheral of a remote computer. This program temporarily changes your computer into a device that only sends data to or receives data from another computer. The other powerful features of your computer are not required for telecomputing.

TERMINAL SOFTWARE

One big advantage of telecomputing is that any computer, whether it's a Commodore, an Apple, or an IBM, can communicate with any other computer. Each of these individual computers, however, interprets electronic signals differently. How can two computers talk to

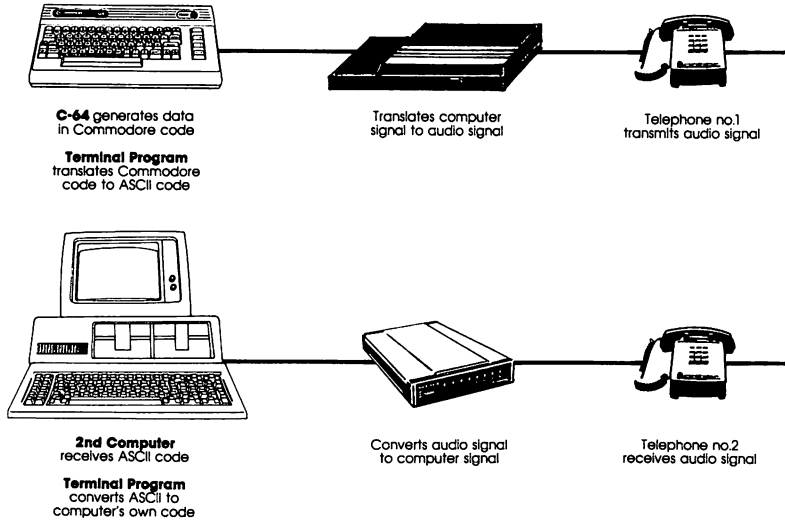


FIGURE 10-1. The C-64 communicating with an IBM PC via modem.

each other if each of them speaks, in a sense, a different language? It is the terminal program that converts each computer's unique electronic signals into standard *ASCII codes* (see page 133) before they are sent to the modem. When standard ASCII signals are received from a distant computer via the modem, the terminal program converts this data into the computer's own code so that the computer can interpret the information correctly. Figure 10-1 shows how a Commodore computer can communicate with an IBM Personal Computer.

There are two types of terminal programs: *dumb* terminal programs and *smart* terminal programs.

Types of Terminal Programs

A dumb terminal program permits you to send and receive information from your computer when you are on-line (connected via modem) with a distant computer. Everything that you type into your computer will be displayed both on your screen and on the screen of the remote computer as you type it. With a dumb terminal program, the keyboard is the only means of entering data you want to send. And reading the screen is the only way you can receive the information being sent to your computer. It will take time to examine large batches of received information. Also, unless you interrupt the flow of information to dump the screen contents to a disk or to the printer, you will have no permanent record of the information received.

Dumb Terminal Programs

Since long distance telephone charges may be incurred while you are on-line, it would be much more efficient if you could send and receive entire files of information using either a disk drive or printer. You could examine the information at your leisure after the telephone connection is terminated. To do this, you need a smart terminal program, one that gives you *upload* and *download* capability.

A smart terminal program allows you to do everything that a dumb terminal program does, and more. It gives you the ability to download and upload information. The download capability permits you to receive whole files of information in the form of data or programs. Data files or program files are downloaded to your computer from the remote computer. The smart terminal program creates a buffer memory area in your computer which temporarily stores this file. The buffer is then emptied to a diskette, tape, or printer.

Smart Terminal Program

With the ability to upload data, you can send program or data files to a distant computer. For example, if you and your friend 500 miles away both have modems and terminal programs that offer the ability to upload and download, you can exchange programs or data such as word processing files without having to use the mail. One computer can upload and send a program, and the second computer can download the program and save it on disk.

One of the advantages of telecomputing is that you can download programs from any other computer. Since the disks of many computer systems are incompatible with other systems, the modem is the only device that enables you to receive programs from a different type of computer without typing them in yourself. Once your computer receives a program into its buffer, the smart terminal program will allow you to save it on disk or tape.

Although you can save any program you receive from a different computer, you may have to alter it so that it will run on the C-64. Only programs from other Commodore 64 computers are certain to run on your C-64 without alteration, because some commands vary from computer to computer. If you get a BASIC program from an Apple computer through your modem, this program will have to be altered before it can be run on your computer, because Apple BASIC is different than C-64 BASIC.

Before you start communicating with a distant computer, you must set up the terminal program to operate properly. In using a terminal program, you can choose among several options for proper data transmission. Let's examine some of the options in a terminal program that may require input from the user.

Using a Terminal Program

Baud Rate. The standard rate of transmission of data through a modem is 300 baud. This is equivalent to 30 characters per second (cps). If your terminal program asks for a baud rate, use 300 if you are using a CBM modem. More expensive modems are able to communicate at 1200 baud, or 120 cps.

Duplexing. Modems may operate in *half-duplex* or *full-duplex* mode. If you choose the full-duplex mode, the information you send will be “echoed” back to your video screen exactly as it is displayed on the receiving computer’s screen. This is one way you can detect errors in the information that is received by the remote computer. If you send the word “love” and you see “gbzt” displayed on your screen, you will know instantly that the other computer is receiving garbage rather than proper signals. In half-duplex mode, the information is not echoed to your screen.

Word Length. You will recall that a computer word, or byte, is made up of eight bits. Standard ASCII uses seven bits to define alphanumeric characters, while Commodore uses eight. Your terminal program will ask you whether you wish to receive seven-bit or eight-bit words. Choose the eight-bit option.

Stop Bits. Usually one or two blank bits are placed after each character to mark the end of a character. Choose ‘1’ for CBM modems.

Parity. Parity check is a method of checking for transmission errors by including an extra bit in each character byte. Parity is either “odd” or “even.” In an “odd” parity system, the extra bit is used to make the total value of the byte odd. If the value of the byte (without the extra bit) is odd, the parity bit added is 0; if the original byte value is even, a parity bit of 1 is added to make the total byte value odd. Therefore, every byte the computer receives should be odd. The computer checks each byte it receives. If it receives an even byte, the computer knows there has been an error in transmitting that byte.

If the parity is set incorrectly for the terminal program, many of the characters that are received will be printed to the screen incorrectly.

The VICMODEM and the AUTOMODEM (Figures 10-2 and 10-3) are both direct-connect modems. Both of these modems are relatively inexpensive compared to other brands on the market. You can purchase VICMODEM for a price equivalent to the cost of some single software packages. For the price of five game programs, you can purchase the AUTOMODEM. Each of these modems comes complete with a dumb terminal program and free access time on information services.

COMMODORE MODEMS

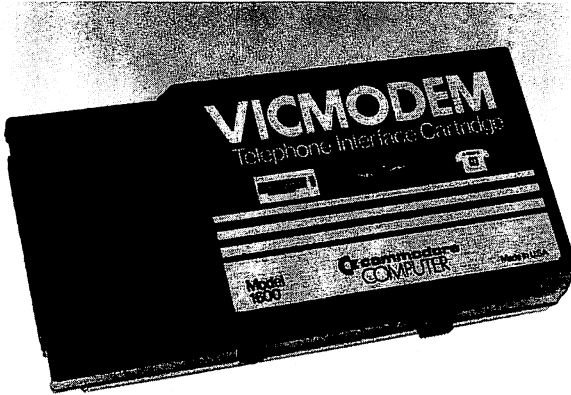


FIGURE 10-2. The Commodore VICMODEM.
(Courtesy of Commodore Business Machines, Inc.)

To connect the VICMODEM, you must have a modular telephone—one with a handset that is separate from the cradle. The handset must have a cord that can be unplugged from it. If you do not have modular telephone with a removable cord, you may have to purchase a VICMODEM Nonmodular Telephone Adapter from your local Commodore dealer. Follow these steps to prepare properly for telecomputing:

Setting up the VICMODEM

1. Turn off the computer system and plug the VICMODEM, with the label facing upward, into the user port.
2. Set the Originate/ Answer switch of the modem to "O" when you are using the computer to communicate with information services. The switch must be set to the "A" position, for Answer mode, to receive a call originated by a remote computer. If you want to telecommunicate with a friend, for example, you will dial his number after setting your modem to the "O" position. He will receive your call with his modem set on "A".
3. Turn on your computer system, and load and run your terminal program.
4. Dial the number of the remote computer you want to communicate with.



FIGURE 10-3. The Commodore AUTOMODEM.

(Courtesy of Commodore Business Machines, Inc.)

5. When you hear a high-pitched tone, detach the cord from the handset and plug it into the VICMODEM. Do not hang up the handset. It must remain off the hook until you have logged off (terminated your telecommunication).
6. If nothing happens on your screen at this point, hit [RETURN] a couple of times. In a moment you will see a greeting from the distant computer displayed on your screen. You are now *on-line* with another computer. Most services will display instructions to help you.

Follow these steps to properly set up the AUTOMODEM.

1. Turn off the computer system and plug the AUTOMODEM, with the label facing upward, into the user port.
2. Unplug the telephone line from the base of your modular telephone and insert it into the socket marked "LINE" on the AUTOMODEM. Plug one end of the connecting cable that came with the modem into the back of the telephone, and the other end into the socket on the modem called "PHONE."
3. Set the Originate/ Answer switch of the modem to "O" when you are using the computer to communicate with information services. The switch must be set to the "A" position (for

Setting Up the AUTOMODEM

Answer mode) to receive a call originated by a remote computer. If you want to telecommunicate with a friend, for example, you will dial his number after setting your modem to the "O" position. He will receive your call with his modem set on "A" for answer.

4. Set the "full/half duplex" switch according to the requirements of the service with which you're going to communicate.
5. Turn on your computer system and load and run your terminal program.
6. Set the "telephone/data" switch to "D" (Data) when you want to use the modem, and "T" (Telephone) when you want to use the telephone for regular calls.
7. Use the auto-dial feature of the AUTOMODEM to dial the desired number.
8. If nothing happens on your screen when you have connected with the remote computer, hit [RETURN] a couple of times. In a moment you will see a greeting from the distant computer displayed on your screen. You are now on-line with another computer. Follow the instructions to obtain help on how to use the system.

The entire world of information is now at your fingertips.



PART 3

Operating the C-64

Using the C-64 in Direct Mode

Your computer can operate either in *direct mode*—for writing and drawing on the screen or for doing calculations, or in *program mode*—for writing, editing, listing, and running programs. *Quote mode* is a kind of “submode” that can be used in either direct or program mode. Quote mode is discussed in this chapter and again in Chapter 12.

WHAT IS DIRECT MODE?

In direct mode, you can use your computer to carry out useful functions even without a program. You can print characters on the screen, perform mathematical calculations, and even find out how long you’ve been working on the computer. Of course, the simple, direct commands you use to accomplish these tasks do not take full advantage of the sophisticated capabilities of the computer as well-written programs do. However, you can explore much of the C-64 by trying some of the following examples of commands in direct mode.

Turn on your computer and all its peripherals. Remember, you should plug all of the components into a power strip and turn them all on together with the power strip’s main power switch. This way you can leave all the “on/off” switches on your equipment set to “on” all the time, saving a lot of wear and tear on the switches. If you are not using a power strip, turn on all peripherals before you turn on the computer. On power-up you will see the display shown in Figure 11-1, with a flashing cursor. The presence of this cursor indicates that the computer is waiting for you to tell it to do something; it means that you have control over the computer.

The computer is now in direct mode as opposed to program mode. It will remain in direct mode until you enter program mode by running a program or by entering BASIC program instructions identified by line numbers.

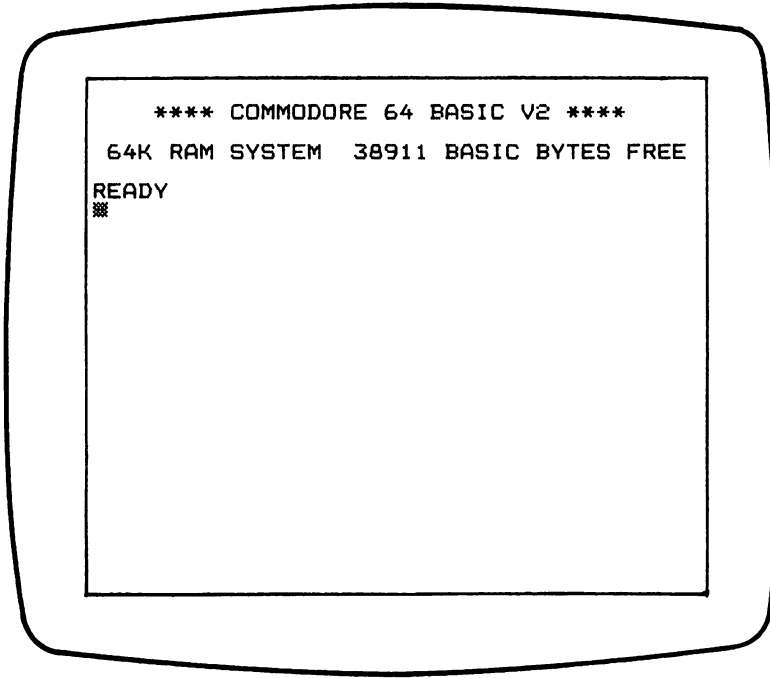


FIGURE 11-1. Power-up screen.

You can use the keyboard to write on the screen. First clear the screen using [CLR] (hold down [SHIFT] while pressing [CLR/HOME]). Type your name and then hit [RETURN]. You'll see a display on the screen similar to:

```

?SYNTAX ERROR
READY.
  
```

The [RETURN] made the computer think that the text or numbers you entered made up a BASIC command. Since your name is not understood by the BASIC interpreter as a proper BASIC line, the computer lets you know that it cannot understand what you have entered by displaying the "SYNTAX ERROR" message.

Should you want to leave a message on the screen for someone without getting an error statement, type the message and press [SHIFT][RETURN]. The shifted [RETURN] does not enter your text into the computer as a command, so the error statement is avoided. Try this method of leaving a message by clearing the screen and typing

DISPLAYS ON THE SCREEN

'DON'T FORGET TO GO TO THE BANK' followed by [SHIFT] [RETURN]. Your message now appears without the error message.

When you type graphic symbols on the screen rather than letters or numbers, the computer does not try to interpret them as BASIC command lines when you press the RETURN key, and consequently you will not get the '?SYNTAX ERROR' display. To avoid getting the 'READY' display, however, you will still have to use [SHIFT] [RETURN].

As you've seen, when you are working directly from the keyboard to the screen, you can use the CURSOR, INSERT, and DELETE keys to correct typographical errors.

The PRINT statement causes the computer to display strings, or groups of characters, as well as the results of mathematical calculations on the screen.

THE PRINT STATEMENT

To demonstrate the PRINT statement, type the following commands in direct mode:

Printing Strings

```
[CLR]
PRINT "HELLO" [RETURN]
?"HELLO" [RETURN]
```

Remember that the second and third commands must be followed by [RETURN] before they can be executed. From now on you will not be told to press [RETURN] each time you are to send a command to the computer; you can assume that each command is to be entered by [RETURN]. Exceptions will be noted.

These examples show that the PRINT command will display on the next line of the screen any string of one or more characters that follows it in quotation marks. The short form for PRINT is the question mark (?). Now try this:

```
?"6+2"
```

This is what you should see on the screen:

```
6+2
```

If you didn't see this, make sure you typed everything exactly as above. Computers have exacting syntax standards; they can't second-guess our instructions.

In the examples in "Printing Strings" above, you used quote mode. When you are operating in direct mode, the computer enters quote

Quote Mode

mode as soon as you hit the first quotation mark ("). It stays in quote mode until you either hit a second quotation mark or press [RETURN] or [SHIFT] [RETURN]. This is convenient for defining strings, but there can be some strange results if you make the mistake of forgetting you are in quote mode. For example, try the following. Type:

```
? "HELLO COMPUTER" (but this time don't press [RETURN])
```

To insert the 'O' between the 'C' and the 'M', first cursor left until the cursor is on the 'M' and then INSERT a space for the 'O'. Try it. What happened? Instead of the cursor moving to the left, a series of graphic characters appeared as the cursor control key was pressed. Your display now looks something like this:

```
? "HELLO CMPUTER" (with 5 graphic characters)
```

To make your correction properly, you must leave quote mode. Press [SHIFT] [RETURN] to perform a carriage return. This takes you out of quote mode so you can go back to the 'HELLO CMPUTER' line and make your correction. Remember, another way to exit quote mode is to press another quotation mark.

You have a display that looks like this:

```
? "HELLO CMPUTER"

```

Use the cursor keys to place the cursor over the 'M'.

```
? "HELLO CMPUTER" (cursor under M)
```

Insert a space to the left of the 'M' by holding down [SHIFT] while pressing [INST/DEL]. Your display looks like this:

```
? "HELLO C MPUTER" (space between C and M)
```

Type the O, cursor to the right, and type in the second quotation mark ("):

```
? "HELLO COMPUTER" (cursor under O)
```

Finally, use [SPACE] to erase the graphic characters, and hit [RETURN].

Quote mode will be discussed more fully in Chapter 12.

Type the following command, which looks similar to the one we used earlier:

```
?6+2
```

The screen displays '8', the sum of the two numbers. Mathematical calculations may be performed in direct mode using a PRINT statement followed by a mathematical expression. Notice that when you want to print 6+2 as a string, you had to use quote mode, but now when you want the computer to calculate and display the sum, you must leave out the quotation marks. This gives the results of the expression rather than the string. Try these calculations after clearing the screen:

```
?6-2
```

```
?6*2
```

```
?6/2
```

```
?6!2
```

You can see that '-', '*', '/', and '!' are the symbols for subtraction, multiplication, division, and exponentiation. Now try these calculations:

```
?2+3*6
```

```
?2+(3*6)
```

Both produce the same results. Priority in mathematical operations is as follows: exponentiation is performed first, then multiplication and division, then addition and subtraction. You can use parentheses to group terms together to make the order of operations clear at a glance. Or you can use parentheses to group terms together to change the order of operations. Try this:

```
?2+3*6
```

```
? (2+3) *6
```

The results of the two calculations are different because parentheses were used to change the order of operations. In the second calculation, grouping by parentheses caused the addition of 2+3 to be carried out before the multiplication.

A variable is a term that can assume any of a set of values. You give the variable a name, such as X or AS, and then set its value. You can change its value as many times as you wish. Variables allow you to store arithmetic expressions or strings in the computer's memory tem-

Performing Mathematical Calculations

USING VARIABLES IN DIRECT MODE

porarily and then to recall them to carry out other operations. You may use variables in this way as a shorthand technique.

You can use variables to represent single numbers or arithmetic expressions. Try these examples:

Mathematical Variables

```
X= 2
Y=2+3
Z= X+Y
?X
?Y
?X+Y
?Z
```

When you type in $X=2$, the computer assigns an area of its memory (think of it as a post office box) for X , putting the value 2 into it. A similar box is set up for Y ; 8, the calculated result of $2+3$, is stored in the box. When the computer is told to print the variable X , it prints out the value that it finds in the box labeled X ; likewise for Y .

When asked to print Z , the computer displays the number 10, the numerical result of the calculation $X+Y$.

We can change the value of X by typing:

```
X= 5
```

The computer replaces the 2 in the X box with 5. Enter the following statements:

```
?X
?Y
Z= X+Y
?Z
```

Note that the value of X has changed because it was redefined; the value of Y has not changed; and the value of Z has changed to reflect the change in X .

The values that a string variable can assume are expressed in quote mode. To demonstrate this use of string variables, type the following:

String Variables

```
A$= "COMPUTER" [RETURN]
```

Then type:

```
?A$
```


When you press [RETURN] you will see 'COMPUTER' printed in the middle of your screen.

After a quotation mark, the cursor control and [CLR/HOME] keys will give you graphic character symbols, also called control codes (summarized in Appendix D), rather than performing their function as they would in direct mode. When a command line containing these control codes is executed by the PRINT statement, the functions represented by the control codes are executed.

You can create designs on the screen in direct mode by using the graphic characters available directly from the keyboard. First be sure the computer is in upper-case/graphics mode. If you see lower-case characters, toggle into graphics mode by pressing [CBM][SHIFT].

Let's design several boxes in direct mode. Before you start, clear the screen. Locate the four angle keys ([O], [P], [I], and [L]). Now find four keys with different types of connecting straight lines, (one at the top of the character space '—', the bottom '—', the left '|', and the right '|'). These graphic characters are found on [Y], [P], [H], and [N] keys. You can access them with the Commodore logo key ([CBM]).

GRAPHICS IN DIRECT MODE

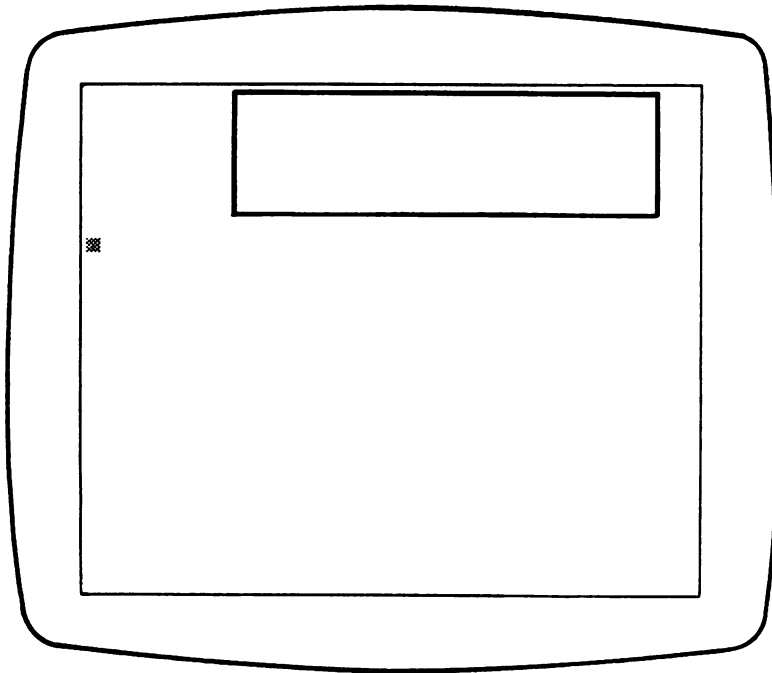


FIGURE 11-2. Finished box.

Beginning 10 spaces from the left margin, design a box like the one in Figure 11-2. It is important that your box begins in the tenth column of your video screen because we are saving the spaces at the left for a later demonstration.

Do not clear the screen at this point. Instructions for turning your direct mode box design into a program that may be saved on disk or tape will be presented in the next chapter. If you were to clear the screen at this point, your carefully designed box would be lost. In direct mode, the computer does not store the characters displayed on the screen in its memory.

The symbol π (on the right side of the keyboard) stands for *pi*, the mathematical value of any circle's circumference divided by its diameter. You may use this value in direct mode; for example:

```
?π
?π*2
```

OTHER DIRECT MODE FUNCTIONS Pi

If you want to use the symbol for *pi* rather than its mathematical value, use quote mode:

```
?“π”
```

There is a clock built into the C-64. You can find out the time elapsed since you turned on the computer by commanding the computer to print out the variable **TIS**. **TIS** is a variable name that the C-64 has reserved for this special use. The location that stores **TIS** contains the latest value of elapsed time. To find out how long the computer has been on, try the following command:

Time

```
?TIS
```

The first two digits show the hours, the next two the minutes, and the last two the seconds.

For more information on the clocks that are part of the C-64, see Chapter 23.

In direct mode, you can change the color of the cursor by holding down [CTRL] or [CBM] while pressing one of the color keys. You can also print characters in reverse text by holding down [CTRL] while pressing [RVS ON] (the number 9 key). The reverse mode can be turned off by pressing [CTRL][RVS OFF] (the number 10 key).

A REMINDER ABOUT COLOR

Using the C-64 in Program Mode

A program is a set of commands that follow the syntax, or rules, of the BASIC language. When a program is run, the computer will carry out these commands in the proper sequence to complete the task.

In this chapter we'll explore some of the especially good programming features of the C-64, and you'll have a chance to do some programming. If you need more information about BASIC programming, please refer to one of the many fine books written on the subject.

Each BASIC statement line must begin with a line number—any number between 0 and 63999. Statements are executed in order, from the lowest to the highest line number. It's a good idea to start with the number 10 and increase each subsequent line number by 10. That way, if you need to, you'll be able to insert BASIC lines between these lines at a later time by using intermediate line numbers, for example, 15 or 17 between lines 10 and 20.

SOME PROGRAM BASICS Line Numbers

The RETURN key enters each BASIC line into the computer's memory. If you forget to press [RETURN], the line that you typed will not be entered into the computer even though it appears on the video screen, and your program will not run properly.

Entering a BASIC Line

If you wish to delete a specific program line, type its number on a blank line and press [RETURN].

Deleting a Line

You can tell the computer to show you a listing of the current program in its memory by typing the command LIST.

Listing a Program

To erase a whole program from the computer's memory, use the NEW command. If you erase a program by typing the NEW command and

Erasing a Program

then change your mind, you can still retrieve the program. The program is still in the memory of the computer, but the computer has “forgotten” where it is. Use the Un-new command built into the utility program POWER PLUS to get the program back, as described in Chapter 19.

To *execute*, or run a program, use the RUN command.

To stop a program while it is running, press [STOP].

To continue running a stopped program, type in the CONT command. If one of the cursor control keys or the RETURN key has been used, you will get a ‘CAN’T CONTINUE’ error.

To do this next example, you will need the box screen display that you designed in Chapter 11. If you erased the box, refer to page 112 to reconstruct it.

Follow these steps to create a program that prints this box to the screen whenever the program is run.

1. Move the cursor to a position just below these boxes in the left-most column. Type NEW and press [RETURN] in order to erase any program that might be in the computer's memory. Now type LIST to verify that there is no program stored in memory (Figure 12-1a).
2. Move the cursor to the far left position of the first line of the box, as shown in Figure 12-1b.
3. Type the number 10 followed by the BASIC statement PRINT and a quotation mark, as shown in Figure 12-1c.
4. Enter this BASIC line into the computer by hitting [RETURN].
5. Repeat the above steps 3 and 4 for all the remaining lines, incrementing the line number by 10 each time. Don't forget to hit [RETURN] for every line (Figure 12-1d).
6. Compare your results with those shown in Figure 12-1d. If you made a mistake, cursor up to the error and correct it. When you press [RETURN] after the correction is completed, this program line will replace the old line in the computer's memory. When you make corrections in this way, you are using a very powerful feature of the C-64, the

Running Programs

Stopping Programs

Continue

CREATING A PROGRAM FROM A SCREEN DISPLAY

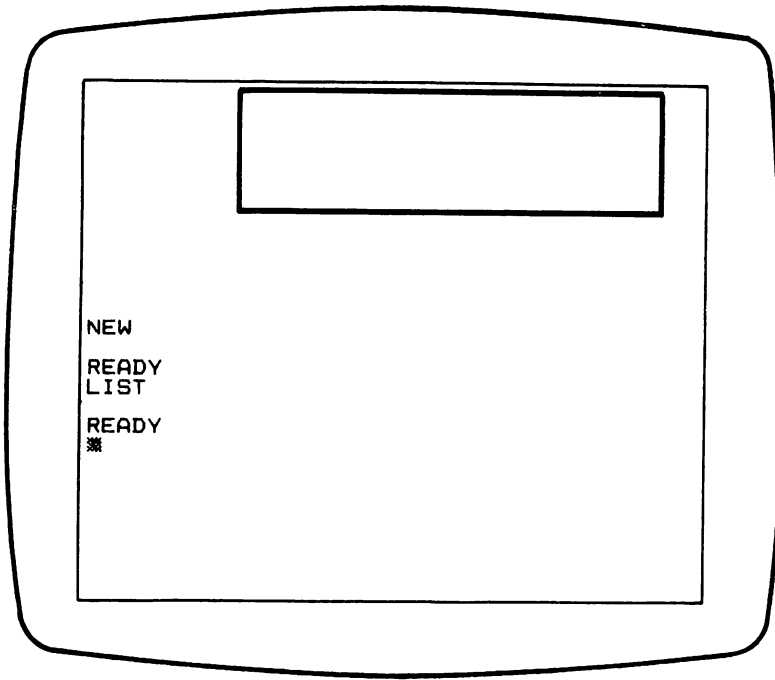


FIGURE 12-1a. Erasing the program in memory.

SCREEN EDITOR. The screen editor of the C-64 permits you to cursor to any line of information on the screen, make your correction, and enter that corrected line by hitting [RETURN]. This feature will be discussed more fully in a later section.

7. If you are certain that you have completed these steps precisely, type LIST to see the program you have created. Once again, if you see a mistake in the program, you can use the screen editor to correct it.
8. Type RUN to execute your program.
9. You can now SAVE this program on tape or diskette following the instructions on page 56 for tape, and page 74 for disk.

You may design any chart or graphic display in direct mode, as we have demonstrated above. To put these pictures into a program, just give each line of the graphic display a line number, a PRINT state-

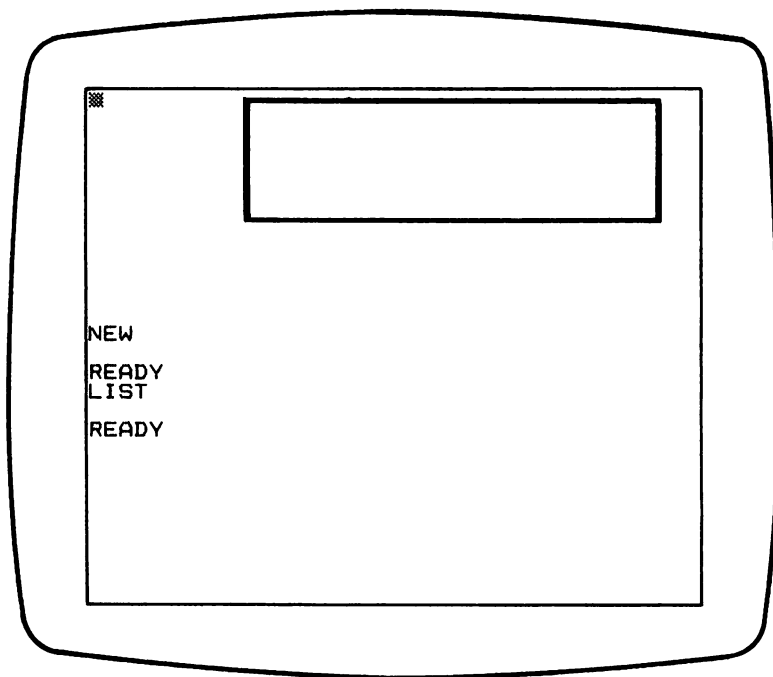


FIGURE 12-1b. The cursor in the HOME position.

ment, and a quotation mark. A second quotation mark must be used at the end of the string of characters if anything else is to follow the string in that BASIC program line. But if the second quotation mark would be at the end of the BASIC line, it may be left out.

NOTE. You can't enter a BASIC line longer than 80 characters. However, a line, when listed, may appear as longer than 80 characters if abbreviations were used for the BASIC commands used in the line. The computer prints abbreviated commands in their full length. A line that is listed longer than 80 characters can't be edited without the loss of all characters over 80.

In most computers, when a BASIC line is entered incorrectly, the entire line must be rewritten and re-entered. With a computer that uses a screen editor, however, you can simply move the cursor to the line with the mistake and make the correction. Hitting [RETURN] when the cursor is at any position on the corrected line will enter that line in place of the incorrect one.

THE SCREEN EDITOR

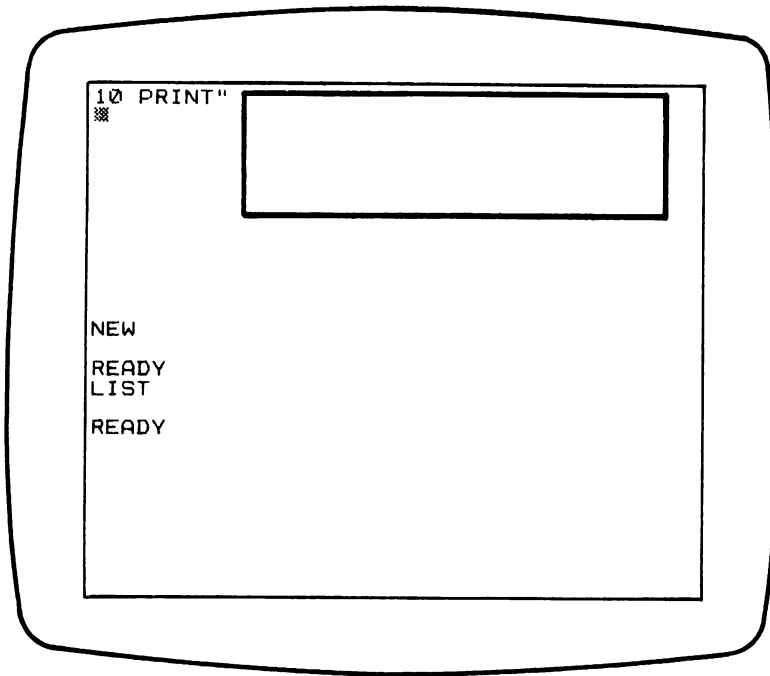


FIGURE 12-1c. Entering a line of BASIC.

The C-64 video displays 40 characters across, but you may write BASIC lines of up to 80 characters. Type NEW, clear the screen, and then type the following BASIC lines:

Additions

```
10 PRINT "COMPUTERS ARE REVOLUTIONIZING"
20 PRINT "DON'T YOU THINK SO?"
```

RUN the program to see the result. If you forgot to type NEW, you may find part of your box diagram below the two sentences. That's because the computer thinks you have changed line 10 and 20 of the box program in its memory but have left all the other lines the same.

Now LIST the program again, and use the screen editor to make a change. Cursor up to line 10 and over to the second quotation mark. Press [SPACE] once to erase the quotation mark and type:

```
EVERY ASPECT OF OUR LIVES"
```

When you did this, a line space was opened up for you so that the new information would not overwrite the next BASIC line. This is one of

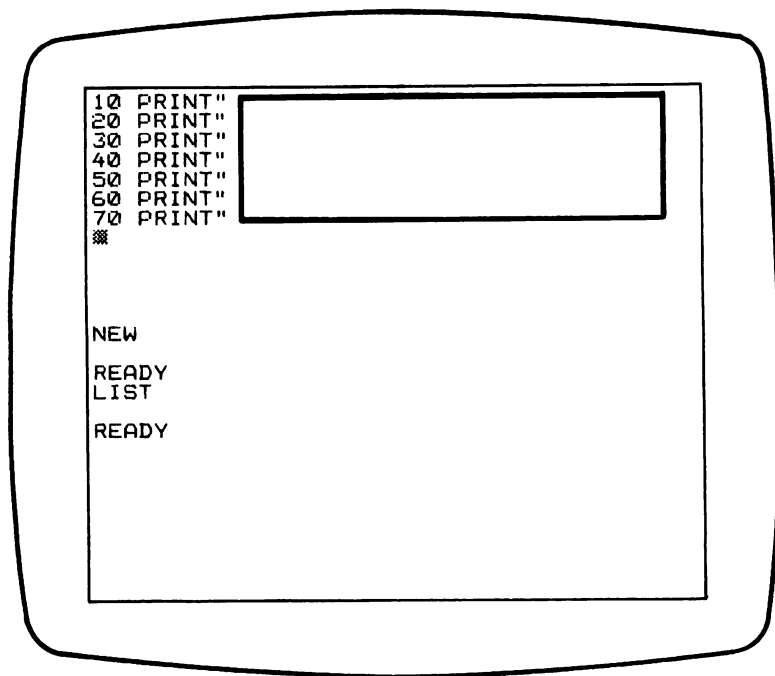


FIGURE 12-1d. The complete program.

the many features of the C-64's sophisticated screen editor. You now have a program that looks like this:

```
10 PRINT"COMPUTERS ARE REVOLUTIONIZING E
    VERY ASPECT OF OUR LIVES"
20 PRINT"DON'T YOU THINK SO?"
```

Hit [RETURN] and try running your revised program, if you like.

The screen editor also gives you the ability to delete and insert text in the middle of a BASIC line. If you want to delete some text, place the cursor one character to the right of the text to be deleted and hit [DEL].

Try deleting 'EVERY ASPECT OF' from line 10 of your program. You will notice that as each character is deleted, the rest of the line moves over to close up the space.

To insert text in a BASIC line, place the cursor at the place where you want to add text and hit [INST] once for each letter to be inserted. A space will be opened up and text to the right of the cursor will move

Insertions and Deletions

to the right. You can now type the text to be inserted. Try inserting 'REALLY' into line 20 to give this:

```
20 PRINT "DON'T YOU REALLY THINK SO?"
```

If you have just done an insertion and tried to type one of the editing keys (cursor keys, HOME/CLR, RVS ON, or RVS OFF), you will notice that a strange graphic character, a control character, appears. To see an example of this, position your cursor at the beginning of an empty line and perform four inserts by pressing [INST] four times. Now press [CRSR DN] eight times. You should have a display that looks like the one in Figure 12-2.

When you insert spaces, you are in QUOTE MODE for each of the inserted spaces. You were in quote mode for the first four cursor downs because you inserted four times, and so you obtained four cursor control codes as you would after a quotation mark. You were in direct mode when you pressed [CRSR DN] four more times, and the

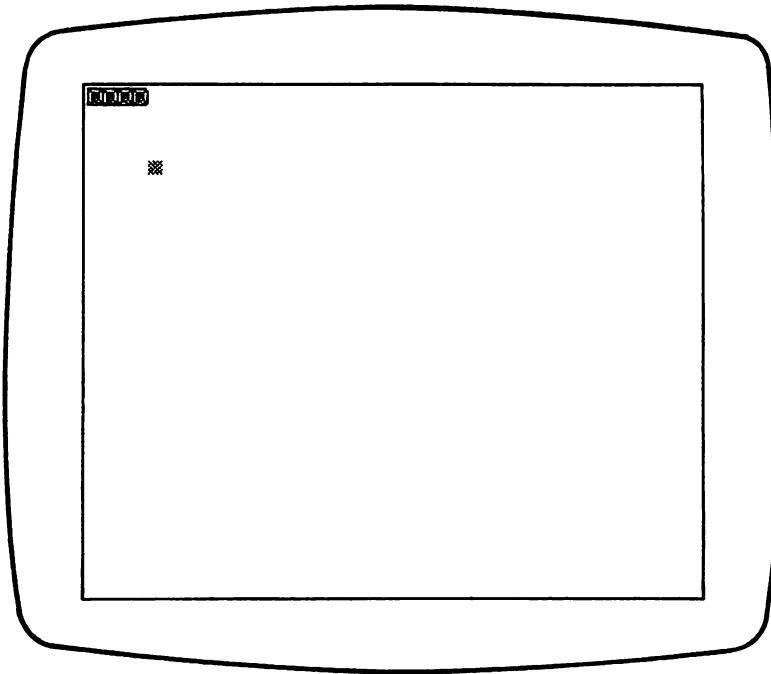


FIGURE 12-2. Inserts put you in quote mode.

cursor moved down four lines. A list of the control codes and their meaning appears in Appendix D.

You can also use the screen editor to duplicate lines while typing in a program. Let's demonstrate. Type NEW to get rid of any BASIC program in the memory of the computer, and press [CLR] to clear the screen. Type in all of line 10 below and hit [RETURN]. Don't type line 20 yet.

Line Duplication

```
10 PRINT "I AM FASCINATED BY COMPUTERS"
20 PRINT "YOU ARE FASCINATED BY COMPUTERS"
```

Line 10 has been entered into the computer's memory. Cursor up to line 10 and type 20 over the line number '10' and hit [RETURN]. You might think that you just erased line 10 and replaced it with the same line, now numbered as line 20. Line 10 is still in the computer. You have duplicated line 10 as line 20. Type LIST to see the program in memory; you should see this:

```
10 "PRINT "I AM FASCINATED BY COMPUTERS"
20 "PRINT "I AM FASCINATED BY COMPUTERS"
```

Now you can use the screen editor to cursor up to line 20 and change 'I AM' to 'YOU ARE.' Type 'YOU A'. The cursor is now on the 'F':

```
10 PRINT "I AM FASCINATED BY COMPUTERS"
20 PRINT "YOU AFASCINATED BY COMPUTERS"
```

Insert 3 spaces, and you will have this:

```
10 PRINT "I AM FASCINATED BY COMPUTERS"
20 PRINT "YOU A   FASCINATED BY COMPUTERS"
```

Type 'RE' and hit [RETURN]:

```
10 PRINT "I AM FASCINATED BY COMPUTERS"
20 PRINT "YOU ARE FASCINATED BY COMPUTERS"
```

READY.

Type LIST over 'READY.'. The screen will display 'LISTY.', which is not a proper BASIC command.

```
10 PRINT"I AM FASCINATED BY COMPUTERS"
20 PRINT"YOU ARE FASCINATED BY COMPUTERS"
```

```
LISTY.
```

If you press [RETURN] now, you will get a 'SYNTAX ERROR' message. You can avoid this by using the space bar to erase 'Y.', leaving the proper BASIC command, LIST. Press [RETURN] to get the following program listing:

```
10 PRINT"I AM FASCINATED BY COMPUTERS"
20 PRINT"YOU ARE FASCINATED BY COMPUTERS"
```

```
READY.
```

```
□
```

For the purpose of demonstration, move the cursor up to the 'R' in READY and type LIST again, so that the screen displays:

```
LISTY.
```

Instead of using the space bar to erase 'Y.', type a colon (:). This tells the computer to ignore everything after the word LIST. The screen display now looks like this:

```
LIST:□.
```

Hit [RETURN] to show that this works, avoiding the 'SYNTAX ERROR' message:

```
10 PRINT"I AM FASCINATED BY COMPUTERS"
20 PRINT"YOU ARE FASCINATED BY COMPUTERS"
```

```
READY.
```

```
□
```

1. To enter a BASIC line after you have made changes, the cursor can be anywhere on that BASIC line when you hit [RETURN]. Type NEW, clear the screen, and then type the following command:

```
10 PRINT"C-64 POWER!" (don't hit [RETURN])
```

Screen Editing Tips

Cursor back to the middle of this line and press [RETURN]. Then type RUN. You will now see this display:

```
10 PRINT "C-64 POWER!"
RUN
C-64 POWER!
READY.
```

This demonstration shows that the cursor can be on any part of the line to enter the line with [RETURN].

2. If the cursor is at the beginning of a line and you want it to be at the end, don't cursor forward. It's quicker to cursor down to the beginning of the next line and then cursor backward once to get to the end of the desired line.
3. Use [HOME] to get the cursor from the lower part of the screen to the upper left corner.
4. If you've been writing a program and need to insert a line between consecutively numbered lines, you may have yet another use for the screen editor. Here's another example for you to try. Type NEW and clear the screen:

```
10 PRINT "MY C-64 EATS"
19 PRINT "APPLES AND PEANUTS"
20 PRINT "FOR LUNCH"
```

To insert the words 'EVERY DAY' between lines 19 and 20, do the following. First, move the cursor to the beginning of line 19 and change '19' to '14' and press [RETURN]. The screen will look like this:

```
10 PRINT "MY C-64 EATS"
19 PRINT "APPLES AND PEANUTS"
20 PRINT "FOR LUNCH"
```

Type 'LIST:'. You can type it right over line 20 if you use the colon. Press [RETURN]. The screen will look like this:

```
10 PRINT "MY C-64 EATS"
14 PRINT "APPLES AND PEANUTS"
LIST:INT "FOR LUNCH"

10 PRINT "MY C-64 EATS"
14 PRINT "APPLES AND PEANUTS"
```

```
19 PRINT"APPLES AND PEANUTS"
20 PRINT"FOR LUNCH"
```

READY.

□

We have duplicated the second line of the program and must now delete the extra one. Type '19' followed by [RETURN].

Type LIST to see the program.

```
10 PRINT"MY C-64 EATS"
14 PRINT"APPLES AND PEANUTS"
20 PRINT"FOR LUNCH"
```

Now type this to enter our new line:

```
16 PRINT"EVERY DAY"
```

LIST the final version of the program to see this:

```
10 PRINT"MY C-64 EATS"
14 PRINT"APPLES AND PEANUTS"
16 PRINT"EVERY DAY"
20 PRINT"FOR LUNCH"
```

5. To delete an entire line, simply type in its line number and hit [RETURN]. We demonstrated this in number 4, above.
6. When you use abbreviations for BASIC commands as shown in Appendix B, you will be able to fit longer statements in a line. When you list such a line, it will appear on the screen in its expanded form. Type NEW, clear the screen, and enter the following BASIC line:

```
100 ?"TRY NOT TO PACK YOUR":?"BASIC LINE
S":?"LIKE WE'VE DONE HERE":GOTO100
```

The abbreviation for GOTO is GOTO. Now type LIST. The screen display will look like this:

```
100 PRINT"TRY NOT TO PACK YOUR":PRINT"BA
SIC LINES":PRINT"LIKE WE'VE DONE HERE":G
OTO100
```

If you try to modify such a line when it is listed in its expanded form, you'll lose all the characters over 80 (anything over two lines). Cursor up to the second line and change the word 'LIKE' to 'BIKE' and press [RETURN]. Cursor down two lines and type 'LIST' to see this:

```
100 PRINT"TRY NOT TO PACK YOUR":PRINT"BA
SIC LINES":PRINT"BIKE WE'VE DONE HERE":G
```

The rest of your BASIC line is gone. If you RUN this program now, you will get a 'SYNTAX ERROR' message.

The solution is either to avoid such long lines, or to shorten some of the expanded commands back to their short form when making the correction, so that the corrected line appears on the screen as no more than 80 characters.

7. If you type in 20 lines of a BASIC program and then discover that you want to keep only six of them in the program, rather than deleting each of the unwanted lines, one by one, by typing their numbers and pressing [RETURN], you can use the screen as a storage area. Let's try an example. Type NEW, clear the screen and enter this line of BASIC:

```
10 PRINT"TEST"
```

Now cursor up to the '1' in line 10, press [2], and hit [RETURN]. Change the '2' to a '3' and hit [RETURN] again. Do this for the numbers 4, 5, 6, 7, and 8. When you type LIST, you should see eight identical BASIC lines, each with a different line number:

```
10 PRINT"TEST"
20 PRINT"TEST"
30 PRINT"TEST"
40 PRINT"TEST"
50 PRINT"TEST"
60 PRINT"TEST"
70 PRINT"TEST"
80 PRINT"TEST"
```

Suppose we only want to keep lines 10, 20, and 70. First type NEW to erase the entire program from memory. Now cursor up to line 10 and hit [RETURN] to enter it into the

```
10 PRINT "TEST"  
20 PRINT "TEST"  
70 PRINT "TEST"
```

USING QUOTE MODE IN PROGRAMS

Cursor Control

```
10 PRINT "[12 CRSR DN][15 CRSR RT][put your name  
here]"
```

```
10 PRINT "XXXXXXXXXXXXXJOHN DOE" □
```

Screen Control

LIST your program and position the cursor on the first letter of your name and insert a space:

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXX JOHN DOE"
```

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"
```

YOUR NAME IN LIGHTS— A DEMON- STRATION

```
20 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXEJOHN DOE"  
READY.
```

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"  
20 PRINT "XXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"
```

List the program and cursor to the first letter of your name in line 10:

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"  
20 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"
```

Press [DEL] to delete the code for reverse mode, and hit [RETURN]. Run the program. It's hard to see what happens, because it's fast. The

screen is being cleared at the beginning of each line, so we are not able to see much of an effect. Change the program by listing the program and deleting the inverse heart in line 20.

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"  
20 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXFJOHN DOE"
```

Try running it now. The second BASIC line is causing a scroll downward, printing your name in reverse text mode 10 lines below your name printed in normal text mode. Let's get rid of all the cursor down codes in line 20 by deleting them:

```
10 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"
20 PRINT"XXXXXXXXXXXXXXXXXXXXJOHN DOE"
```

What happens when you RUN the program now? You will see two displays of your name in the middle of the screen, the reverse text mode below the normal one.

To create a flashing name, LIST the program and position the cursor just to the right of the first quotation mark in line 20 and press [INST]:

```
10 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"  
20 PRINT "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXJOHN DOE"
```

Now press [CRS UP] once and hit [RETURN]:

```
10 PRINT "XXXXXXXXXXXXXIIIIIIIIIIIIIIIIIIIIJOHN DOE"  
20 PRINT "●IIIIIIIIIIIIIIIIIIIIJOHN DOE"
```

When you RUN the program, the cursor up command will cause the second line to print your name in inverse text one line higher. You'll see only the reverse text version of your name because the display from line 10 appears on the screen only for an instant. That is, the output from line 20 will immediately print over line 10's output, eliminating it from view.

Using Your Printer

Although you can process, store, and retrieve data using your computer with a tape or disk drive, you need a printer to make full use of the computer's capabilities. The printer is a peripheral device that will give you a "hard copy" of your processed data on paper. In Chapter 9 we discussed various kinds of printers and how to connect them to your C-64. Discussions and examples in this chapter will show you how to use your printer with more confidence.

When you use a printer and a disk drive together, you should not turn either the printer or disk drive on or off when the other device is in use. If you are loading or saving data with the disk drive, don't turn your printer on or off, because doing so may "lock up" the disk drive. This could cause loss of your data. For a similar reason, don't turn the disk drive on or off when the printer is working.

A WORD OF CAUTION

The codes the C-64 uses to designate characters are not standard ASCII codes used by the industry. When you type a given character on your keyboard, the computer remembers that character as a number value, or code. The ASCII standard code is used by many makers of computers, printers, and other computer peripherals. For example, in ASCII standard code the value for the upper-case letter 'A' is 65; for the lower-case letter 'a', the value is 97. Commodore uses slightly different codes. The value 65 on the C-64 isn't the capital letter 'A', but the small letter 'a'. For a complete list of all Commodore characters and their equivalent codes, refer to Appendix C. If all your peripheral devices are made by Commodore, then each piece of equipment will properly send and recognize the Commodore character codes.

If you're using equipment designed for the industry standard, you'll need an interface to adjust for the variation in codes, as discussed in Chapter 9.

NON-COMMODORE PRINTERS ASCII versus Commodore

As we mentioned above, the main difference between standard ASCII codes and Commodore codes is the coding of the upper- and lower-case letters. The upper-case letters that the C-64 uses are received by a non-Commodore printer as lower-case letters, and the computer's lower-case letters are translated to upper case. To correct this problem, most interfaces have the capability of converting Commodore codes so that the printer responds to the codes the way a Commodore printer would. Make sure that your interface can do this. Otherwise, some software for the C-64 will not be able to give you proper printer output.

Many codes used by the C-64 (such as the Commodore graphic character codes) are not understood by non-Commodore printers. And some of the control codes used by the C-64 ([CLR], [HOME], [RVS], etc.) are also used by your printer, but for different purposes. To work effectively, an interface provided for the C-64 will "trap" these codes and convert them to the appropriate safe values before they reach your printer.

Commodore printers have the built-in capacity for printing the C-64's graphic characters. Some of the more advanced interfaces for non-Commodore printers also provide the ability to print the Commodore graphic characters. For complete compatibility with a Commodore computer, you should use a "smart" printer interface that permits non-Commodore printers to emulate Commodore printers completely.

To confirm that your printer is properly connected and set up, you can perform a short test in direct mode. Remember that when you want to have upper/lower-case characters on your screen, you must toggle out of the upper-case/graphics mode by pressing [CBM] and [SHIFT] together. Make sure that your printer is turned on and that you are in upper/lower-case mode before you type the following commands.

```
open 6,4 [RETURN]
print#6,"ABCDEFGHIJKLMnopqrstuvwxyz":clo
se 6 [RETURN]
```

If you have a non-Commodore printer, when you hit [RETURN], the printer will print the letters inside the quotes so that the first 13 characters are in upper case and the last 13 are in lower case.

If the letters output by your non-Commodore printer from this test don't match what is displayed on your screen, then consult your printer interface manual to find out how to correct for the ASCII conversion between the C-64 and your printer. Most printer interfaces

Translation

Graphics

TESTING THE PRINTER

designed to operate with the C-64 have the ability to “flip” characters so that if they appear as upper/lower case on the screen, they will print out the same way.

If you have a Commodore printer or an interface that allows your printer to emulate one, you will notice that the first 13 characters did not print as upper-case letters, but as graphic characters, and that the last 13 characters are in upper case. Commodore printers, like the computer, provide both an upper/lower-case character set, and an upper-case/ graphics set. When the printer is turned on, it defaults to the upper-case/graphics set, in the same way that the computer defaults to this character set when you first turn it on. When you switch the computer to the alternate upper/lower-case mode, the printer still thinks you want to print in upper-case/graphics mode. To correct this, you must tell the Commodore printer to switch to upper/lower-case. This is done by OPENing the printer channel with a secondary address of 7. Using the same example as above, you would write the BASIC command as follows:

```
open 6,4,7 [RETURN]
print#6,"ABCDEFGHJKLMnopqrstuvwxyz":clo
se 6 [RETURN]
```

If this example gives you the letters ‘A’ through ‘M’ printed in upper case, and the letters ‘n’ through ‘z’ printed in lower case, then continue by typing the following BASIC program. If you didn’t get this result, you may have made a mistake in typing the above commands.

Make sure that your printer is turned on before you run this program.

```
new [RETURN]

10 open 6,4,7
20 for a=1to10
30 print#6,"THIS IS A TEST";
40 print#6,"[2 SPACES]this is a test"
50 next a:close 6
```

If your printer is set up properly, this program will print 10 lines that read ‘THIS IS A TEST this is a test’.

If your printer is printing lines on top of each other, set the *line feed* switch of the printer to the “on” position. The C-64 does not send a

Character Sets

The Line Feed

line feed when sending a carriage return (the code for the RETURN key) to your printer. To have the proper spacing between your lines, you must set the line feed switch of the printer to the “on” position. Some printer interfaces send a line feed themselves to make up for the absence of a line feed from the C-64. This might cause two spaces between the lines when the above program is run. If you get such a result, turn the line feed switch of the printer (see your printer manual) to the “off” position.

In Chapter 9 we explained that you can have more than one peripheral device plugged into the C-64 at one time. A device number is assigned to each one. Whenever we want to access a particular peripheral, we use its device number to tell the computer which device we want to access. The printer’s device number can be either 4 or 5. In most cases, you will use device number 4 for your printer. If you have two printers connected to the C-64, one will be assigned device number 4, and the other device number 5.

To get the printer’s attention, we must OPEN a channel (also referred to as a file) to it with a command like this:

```
OPEN 6,4
```

The 4 is the device number of the printer. The number 6 is used to designate the channel number. It can be any number from 0 to 255. Channel numbers greater than 127 generate a line feed after each carriage return.

If you’re using a Commodore printer or an interface that makes the printer emulate a Commodore, you must include a secondary address of 7 to print in upper/lower-case:

```
OPEN 6,4,7
```

If you OPEN a channel to a device, remember to CLOSE it when you have finished. To CLOSE a file (channel), use the CLOSE command followed by the file number, like this:

```
CLOSE 6
```

After the OPEN command is executed, we can send information to the printer. This information may be in any form that the printer can print out. The simplest way to send data to your printer is with the CMD (CoMmanD) statement. It can be used when you want to print infor-

PRINTER COMMANDS

The OPEN Command

The Secondary Address

The CLOSE Command

The CMD Command

mation from direct mode because the CMD redirects the screen output to the chosen channel. The CMD command requires a channel number. Use the number of the file you have opened; for example, CMD 6.

To get a print-out of a disk directory, first place a disk in your disk drive and type: *Directory Listings*

```
LOAD "$",8
```

This command will load the disk directory into the memory of the computer. If you now type LIST, you should get a listing of the disk directory on the screen. To send the listing to your printer, type this line:

```
OPEN 6,4:CMD 6:LIST
```

You should have a listing of your disk directory on paper. To get out of the CMD mode to the normal screen output mode, type the following:

```
PRINT#6:CLOSE 6
```

You can also use the CMD command to get a BASIC program listing printed out. Load a BASIC program into the computer and type these commands: *Program Listings*

```
OPEN 6,4:CMD 6:LIST
```

The CMD command sends all output to your printer, so you'll get a print-out of the program listing.

The problem with using the CMD command is that all information that appears on the screen is sent to your printer. It prints all messages that normally appear on the screen, such as 'READY.' To get print-outs that exclude any unwanted messages, we can use the PRINT# command.

The PRINT# command enables you to send only selected information to the printer. Use this command in exactly in the same way you would use the PRINT command, or '?', when working in BASIC. However, you can't write PRINT# using the '?' as a short form, like this: ?#. Written this way, the command will give you a 'SYNTAX ERROR' message.

The PRINT# Command

To see how to use the PRINT# command, let's try the following example to print the string 'PRINTERS ARE GREAT!'.

```
OPEN 6,4 [RETURN]
PRINT#6,"PRINTERS ARE GREAT!" [RETURN]
```

The line 'PRINTERS ARE GREAT!' should be printed out, and 'READY.' should appear on the screen.

Use [CRSR UP] to place the cursor on the OPEN line and press [RETURN]. What you should get is a 'DEVICE OPEN ERROR'. This error message lets you know that you have not closed file number 6 before trying to open it a second time. Always close your files, or channels, when you no longer need them, like this:

```
CLOSE 6
```

Many printers have advanced features that provide a variety of character styles and sizes. To take advantage of these features, you must send special codes to the printer. Many of these codes can't be represented by any of the characters on the keyboard. To send special printer codes you'll need to use the BASIC command CHR\$.

The CHR\$ Command

In our first example of the use of CHR\$, let's send the code to cause the Commodore 1515 or 1525-E printers to print in expanded or (double width) mode. This will also work if you're using an interface that emulates the 1525-E. The following program will print the first line of text in the standard size, the second line in double size, and the third line in standard size.

With a Commodore Printer

```
NEW

10 OPEN 6,4
20 PRINT#6,"THIS SHOULD BE NORMAL"
30 PRINT#6,CHR$(14);
40 PRINT#6,"HOW DO I LOOK NOW?"
50 PRINT#6,CHR$(15);
60 PRINT#6,"AND BACK TO NORMAL"
70 CLOSE 6
```

Look at lines 30 and 50. By using the CHR\$ with the number '14', we have sent a code to the Commodore printer that causes the text to be printed in double width. The CHR\$(15) is used to turn the double width mode off. The semicolon after the CHR\$ command suppresses the RETURN and line feed so that the printer doesn't skip a line after

the codes are sent. The semicolon is required here because the PRINT# command will cause a line feed unless it is suppressed by a semicolon. This avoids extra line spaces between the three lines of text.

The codes for non-Commodore printers will be different unless you're using an interface that makes your printer emulate the Commodore printer. To demonstrate how to use the CHR\$ command on other printers, we'll use the Okidata 92 printer as an example. By referring to the printer manual, we'll find that we must send the code '31' to print in double width mode and that the code '30' returns you to normal mode. If you replace the '14' in line 30 with '31' and replace the '15' in line 50 with '30', the above program will work with the Okidata 92 printer. If you are using another type of printer, refer to its manual for the proper codes.

With Non-Commodore Printers

The ESCAPE Code. Many non-Commodore printers use a code referred to as the 'ESC' or escape code. For some printer codes, this escape code must be sent before the actual printer feature code is sent. The ASCII code value of the ESC code is the number 27. For example, to tell the Okidata printer to print in a bold typeface, you must send both the code '27' (for escape) and the code '84' (for bold face). Using the same example as above, change lines 30 and 50 as shown here:

```
10 OPEN 6,4
20 PRINT#6,"THIS SHOULD BE NORMAL"
30 PRINT#6,CHR$(27)CHR$(84);
40 PRINT#6,"HOW DO I LOOK NOW?"
50 PRINT#6,CHR$(27)CHR$(73);
60 PRINT#6,"AND BACK TO NORMAL"
70 CLOSE 6
```

When we send the code '27' and then the code '84', the Okidata printer switches into its bold text mode. Sending the codes '27' and '73' turns off the bold mode.

Sending Letter Codes. Another way to send control codes to your printer is by using the equivalent letter codes in place of the numbers. Most printer manuals list both the CHR\$ values and the matching ASCII characters. In the Okidata manual, the code for the bold command appears like this:

BOLD ON	(27)	(84)
BOLD OFF	(27)	(73)

It is also listed like this:

BOLD ON	ESC	T
BOLD OFF	ESC	I

Since the ESC, or escape code, is equal to 27, we can write our BASIC command like this:

```
CHR$(27) "T" (for BOLD ON)
CHR$(27) "I" (for BOLD OFF)
```

Since the ESC printer code has no equivalent character on your keyboard, this code must be sent as CHR\$(27). But the ASCII code is the same as the character 'T', and the ASCII code '73' is equivalent to the character 'I'.

Refer to your printer manual to find the proper codes for your printer's different printing modes.

Correcting for 'Flip': If you're using an interface that corrects for the ASCII flip, be aware that certain control codes will also be flipped. When the computer sends information to your printer, the interface converts Commodore character codes to their ASCII equivalent. A listing of all the Commodore characters and their codes is shown in Appendix C.

The characters that may cause trouble if flipped have codes from 65 through 90 and from 193 through 218. If you try to use these numbers as printer control codes, your interface will think you're sending the codes for upper- or lower-case letters, and will provide the flip to these values. To correct this problem, add 128 to the value of the control code if it's between 65 and 90, and subtract 128 if the control code is between 193 and 218. Then the flip that the interface performs will result in the correct codes.

The printer can print words from a word processor program as well as numerical calculations from business-oriented programs. It is also possible to use your printer to print graphic representations of numerical data. With the BASIC commands TAB, SPC, and POS, you can format text the way you want it, and you can design charts and graphs.

The TAB, SPC, and POS Commands

The TAB and SPC (SPaCe) commands are used to move the cursor a predetermined number of spaces to the right when printing to the screen. The TAB command moves the cursor to an absolute column position like a typewriter's tabbing function, and the SPC command moves the cursor to the right of the current cursor position. The

TAB and SPC With Numbers

column number for the TAB command and the number of spaces to the right of the cursor for the SPC command are placed in parentheses following the command, like this—TAB(17) or SPC(5).

The TAB and SPC commands can also be placed in PRINT# statements sent to the printer. However, with PRINT#, both commands act like a SPC command, sending the print head to the right of the cursor by the number of spaces in the parentheses. For example, to print the words 'THE TIME IS', and then move to the right 15 spaces and print "12:00 AM", you could do it this way:

```
OPEN 6,4:PRINT#6,"THE TIME IS [15 SPACES]
12:00 AM":CLOSE 6
```

You could do the same thing in a more elegant manner by using the SPC command:

```
OPEN 6,4:PRINT#6,"THE TIME IS";""SPC(15)
"12:00 AM":CLOSE 6
```

You can do the same thing with the TAB command:

```
OPEN 6,4:PRINT#6,"THE TIME IS";""TAB(15)
"12:00 AM":CLOSE 6
```

KERNEL 1 PRINTER PROBLEM

The null string ("") is required with Kernel 1 ROM computers before the TAB or SPC commands in order to correct for a "bug" in that ROM. This is one of the reasons that led Commodore to revise the Kernel ROM with version 2.

The TAB(15) and SPC(15) commands move the print head 15 spaces to the right of its position after printing 'THE TIME IS'. This places the print head in the 27th column. You'll probably find that using the TAB or SPC command is easier than manually using [SPACE] and keeping track of the count.

You can also use variables or mathematical expressions inside the TAB or SPC commands to produce a variety of shapes. To show how the SPC command can be used to accomplish this, type the line below

*TAB or SPC in
Variables*

in direct mode. Make sure your printer is on before you press RETURN.

```
OPEN 6,4:FOR A= 0TO10:PRINT#6,""SPC(A)"FF
F":NEXT A:CLOSE 6
```

Using the variable **A** inside the **SPC** command allows you to “dynamically” change the amount of the spacing to the right.

In the next example, see if you can guess what the output to the printer will be before you run the program.

```
10 OPEN 6,4
20 FOR A= 1 TO 20 STEP 2
30 FOR B= 1 TO 2:PRINT#6,""SPC(20-A/2);
40 FOR C= 1 TO A:PRINT#6,"*";
50 NEXT C:PRINT#6:NEXT B:NEXT A
60 FOR A= 1 TO 3
70 PRINT#6,""SPC(18)"***"
80 NEXT A:CLOSE 6
```

By using mathematical expressions inside the **SPC** command as we did in line 30, we can produce many types of shapes.

To use a true tabbing function with the Commodore printer, or a printer with an interface that allows the printer to emulate a Commodore printer, you can use the **POS** (**POS**ition) command. The **POS** command is sent to the printer by placing the code **CHR\$(16)** in a **PRINT#** statement. The two characters following the **POS** command tell the printer what column to move the head to before printing. Using the same example as above to print ‘THE TIME IS...’, we would use the **POS** command to print the time in column 27 like this:

```
OPEN 6,4:PRINT#6,"THE TIME IS";CHR$(16);
CHR$(50);CHR$(54);"12:00 AM":CLOSE 6
```

The codes **CHR\$(50)** and **CHR\$(54)** stand for the numbers ‘2’ and ‘6’ (Appendix C). The two digits together make up the number ‘26’. Since columns are numbered starting at 0 rather than 1, the 27th column is actually 26 for the C-64. The command to go to column 26 tells the printer to move its head to the 27th column.

The **POS** command will be useful when you want to print out columns of numbers across a page.

By combining the BASIC commands OPEN, CLOSE, CMD, PRINT#, SPC, and POS, you can produce hard copies of charts, graphs, or pictures with your printer. Many printers also provide multiple colors and custom features such as enhanced and bold printing. Some even offer special graphics features to help you design your own print-outs.

Summing Up

One of the most powerful features of the Commodore 64 is its ability to help you design and manipulate *sprites*. Sprites are figures you can use in arcade-style games or in visual displays within other applications programs. Any time you need a fine visual display with custom designs, it's time to use a sprite.

The difference between sprite graphics and standard graphics is that a chip built into the C-64 manages and maintains the sprites for you. This makes it easier and less time-consuming for a programmer to create exciting visual displays.

Sprites are movable object blocks that you can design. The Pac-Man* character is such an object block—a figure that was designed, stored, and moved as a unit of information. The VIC-II chip (the Video Interface Chip) keeps track of the sprites you design and helps you move them around the screen. Since the C-64 remembers a sprite as a single unit of information, it's very easy to move a sprite quickly from one part of the screen to another. The VIC-II chip can also detect collisions between sprites. It can even simulate three-dimensionality by letting you choose which sprite is to be transparent (in the background) and which sprite opaque (in the foreground) when two sprites pass over each other.

Up to 256 sprites may be stored in the computer's memory. Normally, only eight sprites can be displayed on the screen at one time, although programmers can display more than eight sprites through advanced programming techniques. Once you've designed your sprites and stored them in the memory of the computer, you can use the VIC-II chip to manage their interactions.

In this chapter you will learn how to design your own sprites by

INTRODUCTION TO SPRITES

*PACMAN and characters are trademarks of Bally Midway Mfg. Co. licensed by Namco-America, Inc. to Atari, Inc.

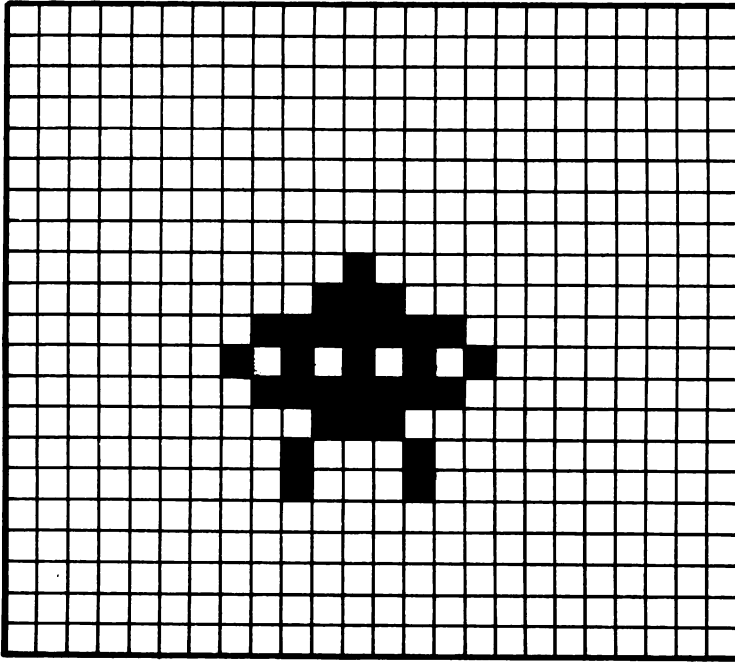


FIGURE 14-1. A small sprite.

using a sophisticated **SPRITE EDITOR** program* provided on the **SUPPORT DISK**. You'll also learn how to use sprites in your own programs by studying **STAR FIELD**, a game program you can type in or load from the **SUPPORT DISK**.

When you use the keyboard, each of the letter, number, and graphic characters that you see on the screen is made up of an 8×8 matrix of dots. A sprite, on the other hand, is made up of a 24×21 matrix of dots. You can design a very small sprite within this grid format (Figure 14-1), or you can design a sprite that uses the full size of the grid (Figure 14-2).

The first step in designing a sprite character is to draw your sprite on graph paper within a 24×21 grid. Then you must calculate 63 binary number values that will tell the computer which dots are to be on and which ones off in the 24×21 grid. These numbers are put into **DATA** statements within a **BASIC** program. The number values are then **POKEd** into the computer's memory.

*This program by Gary Kiziak was originally published in *Commander Magazine*.

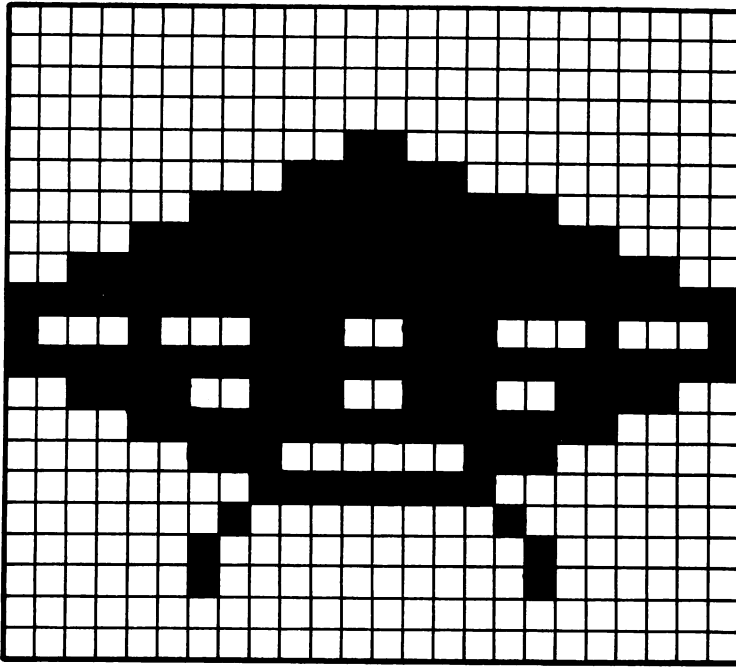


FIGURE 14-2. A large sprite.

You can see that designing sprites could be tedious and time-consuming if you had to do it all yourself. The **SPRITE EDITOR** program included on the **SUPPORT DISK** will make it much easier and faster.

The **SUPPORT DISK** has three programs that make up the **SPRITE EDITOR**: **SPRITE BOOT**, **SPRITE EDITOR**, and **SPRITES TO DATA**. Load and run the **SPRITE BOOT**. It will automatically load and run the main **SPRITE EDITOR** program. Do not try to load and run the **SPRITE EDITOR** by itself; it can only be done properly by running the boot program. The space in front of the file name '**SPRITE EDITOR**' on the directory of the **SUPPORT DISK** lets you know that a boot program should be used to load and run the **SPRITE EDITOR** program.

On the left side of the screen you'll see a 24×21 grid with a cursor blinking in the home position (upper left corner). This grid is your design slate. Each position of the grid represents one *pixel* on the screen. A pixel is one tiny dot on your TV or monitor screen.

You'll create your sprite in the grid. As you design the sprite, the actual sprite as it would look on the screen is displayed to the right of the

USING THE SPRITE EDITOR

Running the Sprite Editor

The **SPRITE EDITOR** Display *The Sprite Grid*

The Sprite Display

grid. You can change the color of the sprite, the border or the background at any time to see how it will look. The sprite can also be expanded along both its X- and Y-axes. The instructions below will show you how to make these changes in the sprite.

At the upper right of the screen you'll see the page number of the sprite you're working on. Page 200 is always the first page provided by the SPRITE EDITOR. It is also the only page that is provided "clean" and ready to use. Other pages will contain some random information, but it's easy to clean up these other pages. We'll show you how later.

Remember, sprites are 63 numbers stored in 63 consecutive bytes of memory. These 63 numbers tell the computer what pixels to turn on to display the sprite you created. The page number indicates where these 63 numbers are stored in the computer. Each sprite page is 64 bytes long. Since a sprite is defined by 63 bytes, the 64th byte within a sprite page is not used. However, a 64-byte page seems easier to keep track of, so we'll treat each sprite's data as 64 bytes long rather than 63.

For example, if a sprite is in page 0, it occupies the first 64 bytes of memory (memory registers 0-63). The sprite in page 1 occupies the next 64 bytes of memory (64-127), and so on. The last sprite (the 256th sprite) occupies locations 16320 to 16383. Since the operating system of the C-64 uses some of the memory locations between 0 and 16383, not all of this storage area is readily available for sprites. For this reason, various pages from 0 to 127 of the computer cannot contain sprite data.

The lower right side of the screen displays the current background, border, and sprite colors. When in multicolor mode, two additional colors are displayed below the other screen color names. A section later in this chapter will show you how to change these colors.

The period key (.) will turn on the sprite unit at the current cursor position. Press the period key several times. You'll see that each location the cursor occupied when you pressed [.] now has a white block in it. Notice that to the right of the grid, small dots are being formed as you press [.] This describes your sprite as it will appear, life-size, on the screen when you use a program.

The cursor control keys will move the cursor around the screen in four directions: up, down, left, and right. When you reach the end of the grid, the cursor will appear on the other side of the grid in the same column or row. Try moving the cursor around on the screen to see how this works.

Sprite Pages

The Color Display

Creating Sprites

Period (.)

Cursor Control

The SPACE key erases the block at the current cursor position. Move the cursor around some more and use [.] and [SPACE] to turn pixels on and off. Now you're on your way to designing your first sprite. *SPACE Key*

The HOME key will send the cursor to the upper left corner of the grid. *[HOME]*

The CLEAR key will clear the entire grid and its corresponding life-size sprite. We use [CLR] to clean up sprite pages to get them ready to use. *[CLR]*

The RETURN key moves the cursor to the beginning of the next line. *[RETURN]*

Editing Sprites

The DELETE key will delete the sprite unit to the left of the cursor. *[DEL]*

This command Inserts a blank line at the current cursor position. It causes everything below the line the cursor is on to move down one line. *[CTRL] [I]*

Use this command to Kill, or delete, the line at the cursor position. It moves everything below the line the cursor is on up one line, filling the deleted line. *[CTRL] [K]*

This command will Reverse the current sprite. Each pixel that is on will be turned off, and each pixel that is off will be turned on. *[CTRL] [R]*

Flips the entire sprite upside down, rotating it around row 11, on its X-axis. *[CTRL] [F]*

Flips the entire sprite sideways, rotating it around a Y-axis between columns 12 and 13. (This move takes some time, so be patient.) *[SHIFT] [F]*

The FUNCTION 1 key will move the entire sprite up one line. When this is done, the top line of the grid is lost and a new blank line is added at the bottom. The FUNCTION key commands are useful for adjusting the placement of a sprite within the grid. However, any parts of the sprite that are scrolled outside the grid are permanently lost. *[F1]*

The FUNCTION 3 key moves the sprite down one line. *[F3]*

The FUNCTION 5 key moves the sprite to the left. *[F5]*

The FUNCTION 7 key moves the sprite to the right. *[F7]*

Pressing [£] rotates the sprite 90 degrees each time it is pressed. Press [RETURN] to get the cursor back.

British Pound Sign [£]

Managing Sprite Pages

Pressing [+] displays the next consecutive sprite page.

Plus Sign (+)

Pressing [-] displays the previous sprite page.

Minus Sign (-)

[SHIFT] [C] Copies the current sprite to the next sprite page. This command can be useful when you're using several very similar sprites, or several versions of the same sprite, for animation.

[SHIFT] [C]

[CTRL] [C] Copies a range of sprites to other pages. When you press [CTRL] [C], the screen will prompt you for the pages you want to copy and where they are to go.

[CTRL] [C]

[CTRL] [P] selects a sprite Page for editing.

[CTRL] [P]

[CTRL] [D] Displays a range of sprites in various pages. When you press [CTRL] [D], you're prompted to give the beginning and ending page numbers. You can ask to see as many pages as you like. The SPRITE EDITOR will display eight sprites on the screen at a time. You'll be prompted to press [SPACE] to view eight more, and so on. After you've viewed the last of the pages you asked for, press [SPACE] to return to the page you were working on.

[CTRL] [D]

[CTRL] [X] expands or contracts the currently displayed sprite along its X-axis. This command toggles the expansion/contraction on and off. Only the life-size sprite expands or contracts; the grid pattern stays the same.

Sprite Expansions

[CTRL] [X]

[CTRL] [Y] expands or contracts the currently displayed sprite along its Y-axis. Again, only the life-size sprite expands or contracts.

[CTRL] [Y]

The two expansion/contraction commands can be used together to expand or contract the sprite in both directions.

Sprite Colors

[CTRL] [B] changes the Background color. Hold [CTRL] down and press [B] repeatedly to step through the 16 background colors available. The name of the current color is displayed to the right of the grid.

[CTRL] [B]

[CTRL] [E] changes the border color. Again, 16 colors are available.

[CTRL] [E]

The > ([SHIFT] [.]) key changes the sprite color. Only the life-size sprite changes color. >]

Pressing [M] selects Multicolor mode. [M]

The [0] and [1] keys allow you to change the colors in color registers 0 and 1 for the multicolor mode. [0] and [1]

The [CTRL] [V] command will display the current sprite on a blank screen in random motion. This feature allows you to see how your sprite will look as it moves on the screen. You can speed up the motion by pressing [+] and slow it down again by pressing [-]. While the sprite is in motion, you can use the color commands to change the border, background, and sprite colors. The sprite can also be expanded or contracted. The SPACE key will stop and restart the sprite's movement. Press [RETURN] to return to the EDITOR.

Sprite Movement

[CTRL] [V]

The [SHIFT] [S] command will Save a range of sprites pages to cassette tape. Any sprites you design will be lost from the computer's memory if you turn the C-64 off, so it's a good idea to save your sprites.

Sprite File Handling

[SHIFT] [S]

The unshifted [S] Saves a range of sprite pages to disk. You'll be prompted for a file name. [S]

The [SHIFT] [L] command Loads sprites from tape into the SPRITE EDITOR. [SHIFT] [L]

The [CTRL] [L] command Loads sprites from disk into the SPRITE EDITOR. You'll be prompted for a file name. [CTRL] [L]

Pressing [Q] for Quit causes the computer to leave the SPRITE EDITOR. Since the EDITOR program changes some BASIC pointers, you should either reset the computer using a reset switch, or turn the computer off and then on again before loading a new program. It's not necessary to reset before loading the SPRITES TO DATA program.

Exiting the Editor

[Q]

After you've designed a sprite using the SPRITE EDITOR program, the 64 bytes of data that describe this sprite are stored in the memory of the computer in the page location indicated on the upper right side of the EDITOR display. In order to use this sprite data within a program, you must convert the data into DATA statements that can be added to your BASIC program.

SPRITES TO DATA STATEMENTS

The SPRITES TO DATA program on the SUPPORT DISK will convert the 64-byte sprite description in the memory of the computer into DATA statements. Follow these steps to use it properly:

1. After designing your sprites, or after loading them from tape or disk, exit the SPRITE EDITOR program by pressing [Q].
2. Load and run the SPRITES TO DATA program.
3. The program will prompt you to type in the first line number for the DATA statements that will hold the sprite data numbers. Choose a large enough number (such as 60000), so that if you append these basic lines to the end of any BASIC program, the line numbers of the DATA statements won't overlap the line numbers you've already used in your program.
4. You'll be prompted to enter an increment for the line numbers. Use 10.
5. The next prompt asks for the number of the first sprite page you wish to convert to DATA statements.
6. The last sprite page number is then entered. If you're only converting one sprite, this page number will be the same as for 5 above.
7. The program now carries out the conversion.
8. Type LIST. You'll see a listing of the SPRITES TO DATA program with your sprite data statements at the end of it.
9. Use SUPERBASIC's DELETE command (or manually delete each line by typing its number followed by [RETURN]) to delete lines 0 to 11 (the SPRITES TO DATA program), leaving only the DATA statements.
10. You can now save this program and append it to your BASIC program (see page 247). Or you can use the screen dump command of POWER PLUS (page 219) to get a printout of the DATA statements. Then load your main program into the computer and type the DATA statements from the print-out, starting at the largest line number of your program, incremented by 10.
11. Refer to lines 220 and 270 of the STAR FIELD program described below to find out how to use your BASIC program to POKE your sprite into the memory of the computer.

Now that you have created your own sprites using the SPRITE EDITOR, you'll want to put them to use by incorporating them into BASIC programs. To discover how to use and manipulate sprites in your programs, we'll study a short BASIC game program called STAR FIELD. This program is on the SUPPORT DISK. If you don't have this disk, type the program as listed below and save it on disk or tape.

SPRITES IN PROGRAMS

```

100 REM *****
110 REM *
120 REM *      SPRITE STAR FIELD      *
130 REM *
140 REM *      BY  STEVE CATES      *
150 REM *
160 REM *      EDUCOMP  1983      *
170 REM *
180 REM *****
190 :
200 POKE 53280,6                :REM BORDER COLOR
210 POKE 53281,0                :REM BACKGROUND COLOR
220 PA=13:POKE 2040,PA          :REM SPRITE PAGE LOCATION
230 PRINT "[CLR][17 CRSR DN]
240 :
250 REM **** READ AND POKE SPRITE DATA ****
260 :
270 PA=PA*64:FOR A=PA TO PA+63:READ B:POKE A,B:NEXT A
280 :
290 DATA 0,0,0,0,0,0,0,0,0
300 DATA 0,0,0,0,60,0,0,255,0
310 DATA 3,129,192,63,255,252,64,0,2
320 DATA 127,255,254,63,129,252,7,255,224
330 DATA 14,24,112,0,0,0,0,0,0
340 DATA 0,0,0,0,0,0,0,0,0,0
350 DATA 0,0,0,0,0,0,0,0,0,0
360 :
370 V=53248                    :REM BEGINNING VIC-II LOC.
380 X=24:X1=0                  :REM X SPRITE LOCATION
390 Y=150                      :REM Y SPRITE LOCATION
400 S=40                      :REM STAR COUNTER
410 D=6                        :REM SPRITE MOVER
420 POKE V+39,4                :REM COLOR OF SPRITE 0
430 POKE V+31,0                :REM CLEAR COLLISION LOC.
440 POKE V,X:POKE V+1,Y:POKE V+16,X1:REM PLACE SPRITE 0
450 POKE V+21,1                :REM TURN ON SPRITE 0
460 J=NOT PEEK(56320) AND 15    :REM READ JOYSTICK 2
470 K=PEEK(203)                :REM READ KEYBOARD
480 POKE 646 ,RND(0)*15+1      :REM POKE RANDOM COLOR
490 PRINT "" SPC(RND(0)*S)"*"; :REM PRINT STAR

```

```

500 IF PEEK(V+31) THEN 610           :REM CHECK COLLISION
510 IF J>3 AND J<7 THEN X=ABS(X-D):GOTO 560:REM MOVE LEFT
520 IF K=47 THEN X=ABS(X-D):GOTO 560:REM MOVE LEFT IF '<'
530 IF J>7 AND J<11 THEN X=X+D:GOTO 560:REM MOVE RIGHT
540 IF K=44 THEN X=X+D:GOTO 560      :REM MOVE RIGHT IF '>'
550 FOR A=1 TO 60:NEXT A:GOTO 460:REM DELAY LOOP
560 IF X>255 THEN X=0:X1=1           :REM CHECK X BOUNDARY
570 IF X1=1 AND X<0 THEN X=255:X1=0:REM CHECK X BOUNDARY
580 IF X1=1 AND X>90 THEN 670        :REM CHECK IF OFF SCREEN
590 POKE V,X:POKE V+1,Y:POKE V+16,X1:REM MOVE SPRITE
600 GOTO 460                         :REM BACK TO READ JOYSTICK
AND KEYBOARD
610 PRINT "[HOME][12 SPACES][WHT]BOOM !!!!!!!!!!"
620 FOR A=1 TO 10
630 POKE V+23,1:POKE V+29,1          :REM EXPAND SPRITE 0
640 FOR B=1 TO 60:NEXT B
650 POKE V+23,0:POKE V+29,0          :REM SHRINK SPRITE 0
660 FOR B=1 TO 60:NEXT B:NEXT A:GOTO 710
670 FOR A=1 TO 10:PRINT "[HOME][11 SPACES][RVS ON][WHT]YOU
WIN !!!!!!!!!!"
680 FOR B=1 TO 50:NEXT B
690 PRINT "[HOME][11 SPACES]YOU WIN !!!!!!!!!!"
700 FOR B=1 TO 50:NEXT B:NEXT A
710 POKE V+21,0                      :REM TURN SPRITE 0 OFF
720 PRINT "[HOME][11 SPACES]ANOTHER GAME? (Y/N)"
730 POKE 193,0:GET A$:IF A$="Y" THEN RUN
740 IF A$<>"N" THEN 730

```

Run the program and play the game for a while. The object of the game is to maneuver your spaceship (the sprite) across the screen from the far left side to the far right side without being hit by the stars. Use either a joystick plugged into joystick port 2, or the '<' and '>' keys to move left or right.

STAR FIELD could have been written without the use of sprites, but it would have been difficult and time-consuming. For example, when the stars reach the bottom, the entire screen is scrolled up one line with the exception of the sprite. A sprite's position is independent of the screen. If you were using a regular graphic character to represent the spaceship, each time the screen scrolled up, you would have to erase and reposition the spaceship to keep it in its original position on the screen.

The Advantages of Sprites

Sprites Are Independent of the Screen

You can move a sprite in any direction on the screen, one pixel-row or pixel-column at a time. This gives you a much smoother motion than moving a regular graphic character one character space. The VIC-II

Sprite Movement

chip performs many complex functions that in most other computers would be very difficult, if not impossible.

Let's examine STAR FIELD line-by-line to see how the sprite is used.

Recall that the C-64 has the capability of displaying eight sprites on the screen at a time. You can store sprites anywhere you wish in the computer's memory, as long as the memory locations you choose are free. There is a sprite page pointer for each of the eight sprites that can be displayed on the screen, so you can pick the sprite (0 to 7) you would like to use by setting its pointer. You'll remember that each sprite page is 64 bytes long. A sprite can be stored in any of 256 available pages, but keep in mind that the computer also uses some of these locations for its own operations. If you need only a couple of sprites, store them in pages 13 through 15. This area is the cassette buffer, locations 832 to 1024. As long as you don't use the Datassette while running your program, your sprites will be safe there.

Suppose we used page 13 of the SPRITE EDITOR to design our spaceship for the game. Since each sprite occupies 64 bytes of memory, this puts our sprite in memory locations 832-895 ($13 \times 64 = 832$), the cassette buffer area. The spaceship will be sprite 0. We must tell the computer where it can find the data that describes sprite 0. This is done by POKEing our page value 13 into the sprite pointer for sprite 0. The sprite page pointers are located at memory locations 2040 to 2047, where location 2040 is the pointer for sprite 0, 2041 for sprite 1, and so on, as shown in Table 14-1.

Setting the Sprite Pointers

SPRITE NUMBER	SPRITE POINTER LOCATION
0	2040
1	2041
2	2042
3	2043
4	2044
5	2045
6	2046
7	2047

TABLE 14-1. Sprite Pointers.

Line 220 of the STAR FIELD program POKEs 13 into location 2040, telling the computer that the 64 bytes of data that define sprite 0 are in page 13, memory locations 832 to 895, the cassette buffer.

After you have set the sprite pointer, place your sprite data, as generated by the SPRITES TO DATA program described above, into the page location. In STAR FIELD, this is done in lines 270 through 350. Line 270 reads the DATA statements in lines 290 through 350, which contain the 64 values for the spaceship sprite. Each DATA statement value is POKEd by line 270 into the proper memory location of the cassette buffer. Line 220 lets variable **PA** equal 13, the sprite page. Line 270 sets **PA** to equal **PA** times 64. This gives us the starting memory location for page 13 ($13 \times 64 = 832$). If you would like to use your own sprite in the program in place of the spaceship, put your sprite DATA statements from the SPRITES TO DATA program in lines 290 through 350.

In line 370 we set the variable **V** to 53248. This is the beginning location for the VIC-II chip. From now on, we'll use variable **V** as our reference location; all other sprite locations will be accessed by adding offset values to variable **V**. By doing this, we only have to remember the offset values for the other VIC-II control registers.

An example of this is on line 420 where the color of the sprite is chosen. If you look at the VIC-II chip address chart in Appendix G, you'll see that $53248 + 39$ is the location for choosing the color for sprite 0. Since variable **V** is equal to 53248, we can place the desired color into this location by POKing it into location **V+39**, like this:

```
POKE V+39, 4
```

The color codes for sprites are identical to those used for the background and border colors, as listed in Appendix D.

Line 430 is used to clear the collision register to make sure that no false collisions are detected at the start of the game. A value of '1' in register **V+31** indicates that a collision between sprite 0 and another screen character has occurred. If you want to detect a collision between two sprites, use register **V+30**. We'll study collisions in more detail later.

To place the sprite on the screen, use values for the X, or horizontal position, and the Y, or vertical position, of the sprite. The range of possible values for the Y position is from 0 to 255. The range for the X position is from 0 to 346 (see Figure 14-3). Since 255 is the highest number a single memory register can hold, we must use an additional register to tell the VIC-II chip when we want to position our sprite past location 255.

Entering Your Sprite Data in Pages

The VIC-II Chip

Choosing the Sprite's Color

Clearing the Collision Register

The Sprite's Position

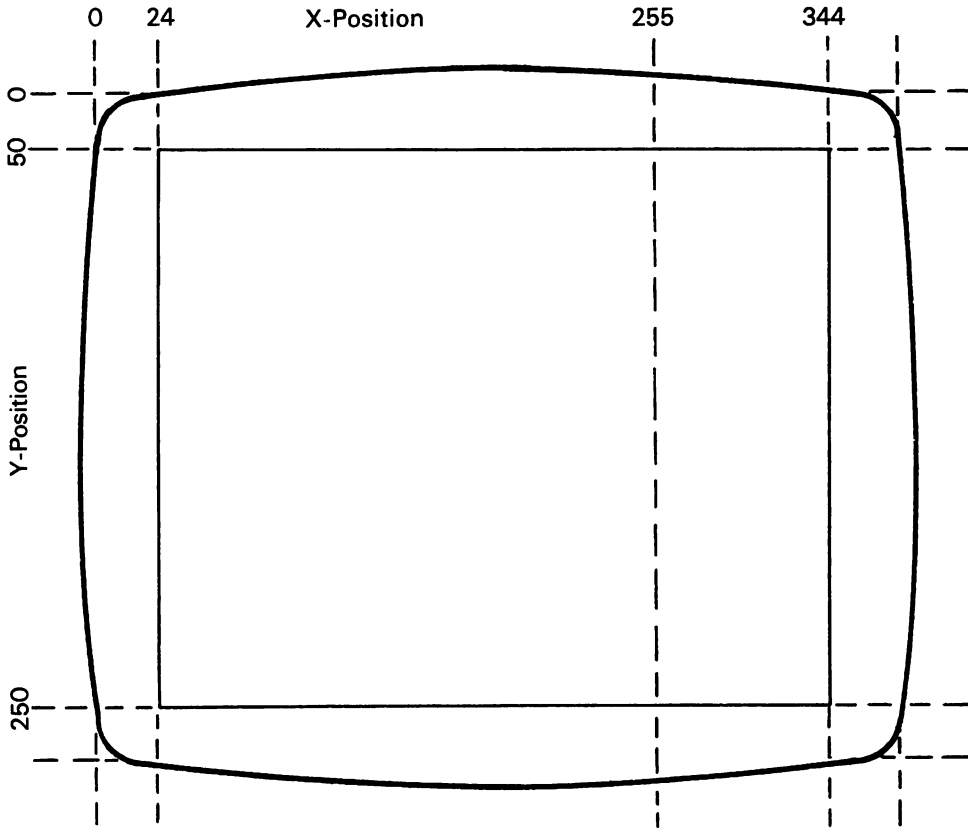


FIGURE 14-3. The sprite's screen position.

We'll use location **V** for the X position for sprite 0, and location **V+1** for the Y position for sprite 0. The first bit in location **V+16** is used to tell the VIC-II chip if we are beyond 255 in the X position for sprite 0. By looking at lines 380 and 390, you'll see that we have set **X** equal to 24, **X1** to 0, and **Y** to 150. In line 440, we use the variables **X**, **X1**, and **Y** to represent the X and Y positions. We then **POKE X** into register **V** for the X position for sprite 0, we **POKE Y** into register **V+1** for the Y position for sprite 0, and **X1** into register **V+16** in case the X position is greater than 255.

Notice that **X** is equal to 24, and **X1** is equal to 0. You would think that when this sprite is displayed on the screen, the sprite would be 24 pixels in from the left side of the screen, since the X-axis position is 24. What you actually see is the sprite flush with the left side of the screen, because the sprite screen is actually larger than the normal video

Placing the Sprites

screen. You can move sprites not only on the visible part of the screen, but also off this screen. When you ran STAR FIELD, you probably noticed that if you moved to the far left side of the screen, the spaceship started to disappear behind the border. If you get the sprite to the far right side of the screen (it isn't easy), the same thing will happen on the right side.

To turn sprite 0 on, POKE register V+21 with 1; our sprite will appear on the screen.

Turning on the Sprite

Line 460 is the beginning of the game loop. Line 460 is used to get a direction value from the joystick (for additional information on using joysticks, refer to Chapter 22). Line 470 reads the keyboard (see box below). Lines 480 sets our random star colors and line 490 prints a star at the bottom of the screen. When a star is printed on the bottom line, the next execution of the PRINT statement on line 490 causes the screen to scroll up, moving up the entire screen of stars. This is how we create the motion of the stars from the bottom to the top. The variable S in line 490 is used to define how far apart the stars will be placed. If you would like to make the game harder or easier, change the 40 in line 400 to a smaller number to place the stars closer together, or use a larger number to place the stars farther apart.

Manipulating the Sprite

PEEKING THE KEYBOARD

Game programs that use the keyboard to receive input from the user must be written to accept this input quickly and efficiently. The GET statement described in Chapter 22 can be used to find out what key is being pressed, but this command doesn't work well for fast games. The GET statement can only read the keyboard after the key has been released. Furthermore, the value for each key is read one at a time and stored in the *keyboard buffer*. This slows the game down.

A better way to detect pressed keys in a game is to have the computer survey the whole keyboard repeatedly and very quickly. This is done by PEEKing the keyboard matrix location, which is register 203 (see line 470 of the STAR FIELD program). First, put the value of the PEEKed location into a variable like this:

```
K = PEEK ( 203 )
```

Each time this command is executed in your program, the value of the key being pressed will be put into variable **K**.

In order to tell the computer what to do when a particular key is pressed (when **K** equals a particular value), you'll need to know the values for the keys. To find these keyboard values, type in and run the following program:

```
10 PRINT "[CLR]" PEEK(203):GOTO 10
```

When this program is run, the number 64 is printed at the home position. The number 64 is the value for no key being pressed. Press any key; its keyboard value will appear in the home position in place of the 64. In this way, you can find the values for any key you wish.

To illustrate how to use this way of reading the keyboard, type in the following program:

```
10 PRINT "[CLR][12 CRSR DN][20 CRSR RT]";
20 PRINT "[*][CRSR LT]";
30 K= PEEK(203)
40 IF K= 44 THEN PRINT "[SPACE]*[CRSR LT]";
50 IF K= 47 THEN PRINT "[SPACE][2 CRSR LT]*
[CRSR LT];
60 GOTO 30
```

Pressing either the '<' or '>' will move the '*' to the left or right.

The detection of collisions between a sprite and a screen character is done in line 500, using register **V+31**. We PEEK register **V+31** and check to see if this register contains a 0. If the value in this register is not equal to 0, then a collision has occurred. When a collision between sprite 0 and any screen character occurs, the value in **V+31** will be equal to 1. If you were using a sprite other than sprite 0, the collision values would be different. However, since we're using only sprite 0, the value in this register will be equal to 0 or 1. If a collision has occurred, the program will jump to line 610. If no collision has occurred, then the next line of the program (line 510) will be executed.

The ability of the VIC-II chip to detect collisions removes much of the burden of keeping track of the location of your sprites and other characters on the screen. Without this capability, you have to know

Detecting Collisions

where your sprite is on the screen at all times, and also where each star is, so you can determine if a collision has taken place.

Lines 510 and 530 check whether the joystick is deflected to the left or right. Lines 520 and 540 check whether '<' or '>' keys have been pressed. The variable **D** was set to 6 in line 410. **D** is the number of pixels that the sprite will move each time the user wants to move to the left or right. If the movement is to the left, the value of **D** is subtracted from the **X** variable (the **X** position of the sprite). We used the BASIC command ABS (ABSolute number) in lines 510 and 520 to ensure that variable **X** is always a positive number. If the movement is to the right, the value of **D** is added to variable **X**. Since we're only moving the sprite in a horizontal direction, the **Y** position is never changed from its original value of 150. This keeps the sprite horizontally in the middle of the screen.

Moving Sprites

Line 560 is used to see if the **X** variable is greater than 255. If **X** is greater than 255, we must set register **V+16** to 1, and then reset the value of **X** to 0 to move the spaceship beyond location 255. Line 570 checks to see if we have moved back to the first boundary (where **X** is less than 255). If the boundary has been reached, then **X1** must be set back to 0.

Movement along the **X**-axis is a little complicated, so we'll go over it again. To understand why we need to use two registers to position the sprite on the **X**-axis, remember that we POKE the value for **X** into memory. Since the range of values that we may POKE is from 0 to 255, we can't POKE any values greater than 255. The sprite position range along the **X**-axis, however, extends to 346 (the screen is wider than it is high). To access **X**-axis positions greater than 255, we must set a *flag* to indicate that we are beyond the 255 limit. This is done by placing a 1 into register **V+16** when using sprite 0. If the flag in **V+16** is set with a 1, then the VIC-II chip will add 256 to **X**. If our sprite is at the 255 position, and we want to move the sprite one more pixel to the right (to **X** position 256), we would set the flag at register **V+16**, and then reset the register for the **X** position back to 0. The VIC-II chip adds 256 to the **X** value 0 to get the required value of 256. We are now in the second boundary, and can move an additional 91 pixels to the right. To move back into the first boundary, we must reset register **V+16** back to 0.

Line 580 checks if the game player has successfully moved his spaceship to the far right side of the screen. This line first checks if the spaceship is in the second screen boundary by seeing if **X1** is equal to 1. It

Checking the Sprite Location

then checks if the spaceship is at the far right side of the screen by seeing if **X** is greater than 90. Line 590 repositions the sprite using the new values for **X** and **X1**. Line 600 completes our loop by telling the computer to go back to line 460 to read the joystick and the keyboard.

The VIC-II chip gives you the ability to expand the sprite in either the **X** or **Y** direction. When a collision has occurred in **STAR FIELD**, the sprite is expanded and shrunk a number of times. This is done in lines 630 through 660. The register to expand in the **Y** direction is **V+23**, and the register to expand in the **X** direction is **V+29**. In this program, sprite 0 is expanded in both the **X** and **Y** directions by **POKEing** both **V+23** and **V+29** with a value of 1. To return the sprite to its normal size, the same registers are **POKEd** with a value of 0, as in line 620.

Expanding the Sprite

After you have finished using your sprites, you should turn them off. If you've ever stopped a program that uses sprites by pressing **[STOP]**, you'll see that the sprites continue to be displayed on the screen. To demonstrate this, run **STAR FIELD**, and when the sprite has been placed on the screen, press **[STOP]**. The spaceship is still on the screen, even after you try to clear the screen by pressing **[CLR]**. To turn a sprite off and reset all the VIC-II registers, hold down **[STOP]** and press **[RESTORE]**. You can turn a sprite off within a program by **POKEing** a 0 into register **V+21** as shown in line 680. Since register **V+21** is used to turn any one of the eight sprites on or off, when you **POKE** a 0 into this register, you are actually turning all eight sprites off. To understand how one register may control all eight sprites, refer to the next section on using multiple sprites.

Turning the Sprite Off

The program **STAR FIELD** was designed to help you understand how to use the sprite feature of the **C-64**. In this example, we only used one sprite so that the program would be easy to follow. By referring to the chart in Appendix G, you can see that there are registers for up to eight sprites. To position sprite 1 (the second sprite), for example, you would use register **V+2** for the **X** position and **V+3** for the **Y** position. Additional sprite functions, such as sprite color and sprite page location, have their own memory registers. Some single registers, however, control all eight sprites. An example of this type of register is location **V+21**, which turns the sprites on or off. If you're using only sprite 0, as in **STAR FIELD**, **POKE 1** into register **V+21** to turn sprite 0 on, and **POKE 0** into this register to turn sprite 0 off. To control the "on" and "off" condition of more than one sprite at a time, use Table 14-2.

Multiple Sprites

SPRITE 0	ADD 1
SPRITE 1	ADD 2
SPRITE 2	ADD 4
SPRITE 3	ADD 8
SPRITE 4	ADD 16
SPRITE 5	ADD 32
SPRITE 6	ADD 64
SPRITE 7	ADD 128

TURN ALL SPRITES OFF
WITH 0

TABLE 14-2. Values for turning sprites on.

To turn both sprites 0 and 1 on, refer to the chart and write down the numbers 1 and 2. Now add these two numbers together to get 3. The value 3 is now **POKEd** into register **V+21** to turn both sprites 0 and 1 on. To turn sprite 0 off, and leave sprite 1 on, get the value for turning sprite 1 on from the chart (value of 2), and **POKE** that value into **V+21**.

Any time you wish to use more than one sprite, refer to the chart in Appendix G, and if any of the sprite registers are being used to control multiple sprites, use the values listed in Table 14-2 for turning on each sprite. You can turn on any number of sprites in any order you choose. You can choose additional sprites by changing the sprite page pointers at locations 2040 to 2047.

Now that you know something about sprites, change the **STAR FIELD** program to incorporate some of the techniques you've learned. Change the color of the sprite. Try to enter a second or third sprite to act as an obstacle. The more you try to put your knowledge into practice, the more you'll learn about sprites.

Creating Sounds with the C-64

The C-64 has the most powerful sound synthesizer of any microcomputer on the market. Other computers' sound generating devices can produce musical notes, but the C-64's Sound Interface Device (SID) gives you three voices and the ability to program the Attack/Decay/Sustain/Release (ADSR) qualities of each voice. With three voices, you can create three sounds at the same time to achieve chords and harmony. You can also have one voice produce percussion sounds while the other two voices carry the melody.

By changing the ADSR of each voice, you can simulate a large variety of sounds, both musical sounds and others. In addition, the C-64 has four different waveforms (distinct sound patterns) and programmable filters for each voice. You can program the SID chip to produce a variety of sounds, from the sound of a harpsichord to the sound of footsteps.

The SID chip occupies memory locations 54272 to 54300. The only location you'll need to remember is 54272, the beginning location of the SID chip. Since we'll be using the POKE command to access the SID chip, location 54272 will be our reference point for the other locations of the sound chip. Appendix H summarizes the sound chip locations.

The sample program below will help you understand how to use the various sound locations of the SID chip. This BASIC program will provide you with the capability of programming music using standard music notation. For example, the notes of the scale will go from 'C' to 'B'. If you're using a video monitor without sound capability, use an external amplifier and speaker so that you can hear the sounds.

SOUND CAPABILITIES OF THE C-64

THE SOUND INTERFACE DEVICE (SID)

A Sample Program

program lines have been numbered in increments of 10, starting at line 100.

```

100 REM *****
110 REM *           MUSIC MAKER           *
120 REM *           BY STEVE CATES       *
130 REM *****
140 :
150 DIM H(7,12),L(7,12)
160 DIM DU(1000),HI(1000),LO(1000)
170 DIM X$(12):X=1
180 SO=54272 :REM *** SID LOCATION ***
190 PRINT "[CLR]          READING DATA"
200 :
210 REM *****
220 REM *           MUSIC SETUP TABLE   *
230 REM *****
240 : VO=10 :REM SET VOLUME *
250 : TE=350:REM SET TEMPO *
260 : AD=9 :REM SET ATTACK/DECAY *
270 : SR=9 :REM SET SUSTAIN/RELEASE*
280 : WF=33 :REM SET WAVEFORM *
290 : P1=0 :REM PULSE WIDTH LOW *
300 : P2=0 :REM PULSE WIDTH HIGH *
310 REM *****
320 :
330 FOR A=0 TO 24:POKE SO+A,0:NEXT A
340 POKE SO+24,VO:REM * VOLUME *
350 POKE SO+5,AD :REM * ATTACK/DECAY *
360 POKE SO+6,SR :REM * SUST/RELEASE *
370 POKE SO+2,P1 :REM * PULSE WIDTH LO*
380 POKE SO+3,P2 :REM * PULSE WIDTH HI*
390 :
400 REM ***** NOTE TABLE *****
410 :
420 FOR A=1 TO 12:READ X$(A):NEXT A
430 DATA C,C#,D,D#,E,F,F#,G,G#,A,A#,B
440 :
450 REM ***** FREQUENCY TABLE *****
460 :
470 FOR A=3 TO 5:REM ** # OF OCTAVES **
480 FOR B=1 TO 12:READ H(A,B),L(A,B):NEXT B:NEXT A
490 REM ***** OCTAVE THREE *****
500 DATA 8,97,8,225,9,104,9,247
510 DATA 10,143,11,48,11,218,12,143
520 DATA 13,78,14,24,14,239,15,210

```

```

530 REM ***** OCTAVE FOUR *****
540 DATA 16,195,17,195,18,209,19,239
550 DATA 21,31,22,96,23,181,25,30
560 DATA 26,156,28,49,29,223,31,165
570 REM ***** OCTAVE FIVE *****
580 DATA 33,135,35,134,37,162,39,223
590 DATA 42,62,44,193,47,107,50,60
600 DATA 53,57,56,99,59,190,63,75
610 :
620 REM ***** READ NOTES *****
630 :
640 READ A$:IF A$="END" THEN 750
650 DU(X)=VAL(LEFT$(A$,1))
660 OC=VAL(RIGHT$(A$,1))
670 IF RIGHT$(A$,1)="R" THEN HI(X)=0:LO(X)=0:GOTO 730
680 A$=MID$(A$,2,LEN(A$)-2)
690 FOR A=1 TO 12
700 IF A$(<)X$(A) THEN NEXT A
710 HI(X)=H(OC,A)
720 LO(X)=L(OC,A)
730 X=X+1:GOTO 640
740 :
750 REM ***** PLAYER ROUTINE *****
760 :
770 FOR A=1 TO X-1
780 POKE SO,LO(A) :REM ** LOW FREQ **
790 POKE SO+1,HI(A) :REM ** HI FREQ **
800 POKE SO+4,WF :REM ** WAVEFORM **
810 FOR B=1 TO TE STEP DU(A):NEXT B
820 POKE SO+4,WF-1 :REM ** GATE OFF **
830 FOR B=1 TO TE STEP DU(A):NEXT B
840 NEXT A
850 FOR A=0 TO 23:POKE SO+A,0:NEXT A:END
860 REM *****
870 REM * ENTER MUSIC DATA BELOW *
880 REM *****
890 :
900 DATA 4C3,4C3,4G3,4G3,4A3,4A3,2G3
910 DATA 4F3,4F3,4E3,4E3,4D3,4D3,2C3
920 DATA 4G3,4G3,4F3,4F3,4E3,4E3,2D3
930 DATA 4G3,4G3,4F3,4F3,4E3,4E3,2D3
940 DATA 4C3,4C3,4G3,4G3,4A3,4A3,2G3
950 DATA 4F3,4F3,4E3,4E3,4D3,4D3,1C3
960 DATA END

```

Run the program after you have checked to make sure that you typed it in correctly.

Notes and Frequencies

The DATA statements in lines 900 to 940 are the musical notes that the program will play. The tune is *Twinkle, Twinkle, Little Star*. Each item of data tells the computer the duration of the note, the note itself, and which octave to play. Let's study the DATA statement on line 900:

```
900 DATA 4C3, 4C3, 4G3, 4G3, 4A3, 4A3, 2G3
```

The '4C3' provides the program with three pieces of information. The number 4 is the duration of the note. The syntax for stating the duration is as follows:

8 = an eighth note

4 = a quarter note

2 = a half note

1 = a whole note

The character "C" is the note we want to play. When you write your own DATA statements so that you can play a different tune, you can use any of the standard notes listed below:

C C# D D# E F F# G G# A A# B

The last character, 3, is the octave of the note. To keep the program short, only three of the eight standard octaves are used. You can see by the REM statements on lines 490, 530, and 570 that we are using octaves 3 through 5. For most musical applications, three octaves are more than enough to explore the capability of the SID chip. You can increase the number of octaves by typing in additional DATA statements for octaves 0 through 2, and octaves 6 and 7. To get the proper values for each note of the new octaves, use the conversion chart in Appendix H. Looking at the chart, you'll notice that two values are required to represent each note. For example, the first note in our program is the 'C' note in the third octave (4C3 in line 900). Referring to the chart in Appendix H, you'll find the values 8 and 97 for this note. These numbers are put into the DATA statements for octave 3 in line 500.

If you want to add more octaves to the MUSIC MAKER program, insert the DATA statements you need for the octaves you wish to add in the proper order, and adjust the FOR/NEXT loop on line 470 to include your additional octaves.

One additional special character is provided to allow for a pause, or rest. The syntax for a rest is the duration number of the rest and the character 'R'. To have a rest of a half note, you would use the syntax '2R'. Since you're not playing a note, you need not state an octave. For longer pauses, use multiple rests, like this:

```
DATA 4C#3, 4C#4, 1R, 1R, 4C#4, 1C#5, END
```

If you want to write your own song in place of the one in the program, you can type your note values over the ones in lines 900 through 950. When you have entered the notes of your song, include a last DATA statement: END. This tells the computer that the last note has been played. The computer will then turn all the sound registers off and terminate the program.

To get a more thorough understanding of what the SID chip can do, we need to program other locations in the SID chip. We'll get these other locations by using 54272 to designate the beginning of the SID chip, and adding offset values to 54272. Look at line 180 and notice that we are using the variable **SO** to set our beginning S**O**und location at 54272. By referring to line 180, you can program each of the important sound locations by using the set of fixed variables (**VO**, **TE**, **AD**, **SR**, **WF**, **P1**, and **P2**) listed in lines 240 through 300. The first variable, **VO**, is used to adjust the volume of the notes. You may set the volume of all three voices by setting **VO** equal to a value between 0 and 15, where 15 is the loudest setting, and 0 is the lowest setting. Line 340 **POKEs** the **VO** value into location **SO+24**, which is the volume location of the SID chip. For a listing of all the SID registers, refer to Appendix H.

Remember that location **SO+24** is the master volume control; it sets the volume for all three voices of the C-64. To demonstrate how this music program sets the volume, LIST line 240. You'll see that the variable **VO** is equal to 10. Change that value to 15, which is the highest setting you can use; then **RUN** the program again. You'll find that the volume is a little louder this time.

The SID chip uses four types of waveforms for producing different qualities of sounds. The SID chip can produce *TRIANGLE*, *SAW-TOOTH*, *VARIABLE PULSE*, and *NOISE* waveforms. By visualizing the shape of each of these waveforms, you can better anticipate the quality (timbre) of the sound that is generated by that particular waveform (see Table 15-1). The triangular waveform will have a mellow

The Rest

The Volume Switch

Waveforms

sound in contrast to the sawtooth waveform's coarse sound. The variable pulse waveform is a square wave. By adjusting the width of the pulse waveform, you can achieve a variety of sounds. The noise waveform is a combination of random waves. It is used to produce harsh noises such as explosions. Table 15-1 lists the correct 'on' and 'off' values for each waveform:

LOCATION	TRIANGLE	SAWTOOTH	PULSE	NOISE
VOICE 1 S+4 — 54276	17/16	33/32	65/64	129/128
VOICE 2 S+11 — 54283	17/16	33/32	65/64	129/128
VOICE 3 S+18 — 54290	17/16	33/32	65/64	129/128
				
	On/Off	On/Off	On/Off	On/Off

TABLE 15-1. ADSR and waveform control settings

Placing the 'on' value of any waveform in line 285 of our music program allows us to use that waveform. In line 800 we POKE our waveform value (**WF**) into the SID chip at location **SO+4**. This is the location for setting the waveform for voice 1 and for turning the sound of the note we wish to play on or off. Your music program should have completely prepared the computer to create your chosen sounds before you POKE to this location, because it starts the music.

Looking at line 820, you will notice that we again POKE into location **SO+4**; this time we use the variable **WF-1** to turn the waveform and the sound of voice 1 off (refer again to Table 15-1).

The triangle, sawtooth, and noise waveforms have 'on' values 17, 33, and 129. You may choose one of these waveforms by POKEing its 'on' value into location **SO+4**. To change the waveform in our music program, LIST 285 and note the value of the variable **WF**. This value is 33, the value for a sawtooth waveform. Let's change the 33 to 17, the value for the triangle waveform, and run the program. What's the quality of the sound now? You'll find that the notes are more mellow than with the sawtooth waveform.

Just for fun, set the waveform to noise by using the value 129 and RUN the program. Using the noise waveform is not recommended for musical notes, but it's useful for the creation of sound effects.

To use the three waveforms described above, we set only the **SO+4** location. To use the variable pulse waveform (value 65), we must set not only **SO+4** but also the “pulse-low” and “pulse-high” locations. These two values determine the pulse width of this rectangular waveform. Changes in the width alter the quality of the pulse sound from an organ-like sound to a soft, reedy one. In lines 370 and 380 of your music program, locations **SO+2** and **SO+3** set the pulse width for voice 1. You can put values from 0 to 255 in the pulse-low location, or **SO+2**, and values from 0 to 15 for the pulse-high location, or **SO+3**. To use the variable pulse waveform, set variable **WF** on line 285 to 65; then set **P1** and **P2** on lines 296 and 270 to your desired values. Feel free to try some of your own values by setting the variables on lines 260 and 270.

Every type of musical instrument has a different kind of sound “envelope,” or characteristic, that makes its sound unique. With the C-64, you can modify the envelope of your sound so that you can synthesize a wide variety of musical sounds and sound effects. The envelope is defined by four factors: the *ATTACK*, *DECAY*, *SUSTAIN*, and *RELEASE* (*ADSR*).

ADSR

The volume of any musical instrument has some variation from the time you first hear the note to the time the note dies out. The Attack is the time it takes the note to reach its maximum volume. The Decay is the note’s rate of drop in volume from its maximum volume to a mid-level volume. The level of the mid-volume of the note is called the Sustain level. The rate at which the sound dies out is called the Release. Study Figure 15-1 for a graphic description of the ADSR envelope.

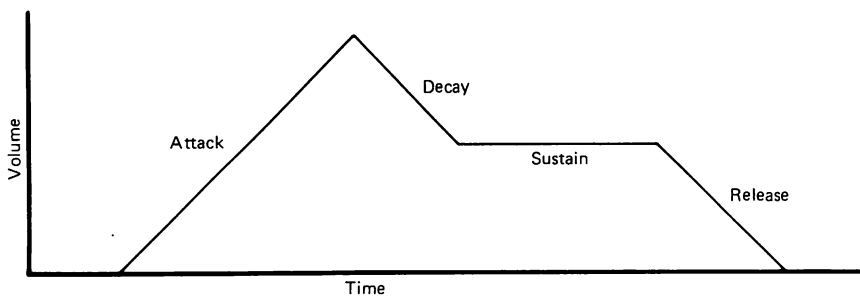


FIGURE 15-1. The ADSR sound envelope.

ATTACK/DECAY RATE SETTINGS

	LOCATION	HIGH ATTACK	MEDIUM ATTACK	LOW ATTACK	LOWEST ATTACK	HIGH DECAY	MEDIUM DECAY	LOW DECAY	LOWEST DECAY
VOICE 1	S+5 — 54277	128	64	32	16	8	4	2	1
VOICE 2	S+12 — 54284	128	64	32	16	8	4	2	1
VOICE 3	S+19 — 54291	128	64	32	16	8	4	2	1

TABLE 15-2. Attack/Decay rate settings.

To manipulate the ADSR envelope, you need to set two locations in the SID chip. The location for Attack and Decay for voice 1 is **SO+5**; **SO+6** is the location for Sustain and Release. Tables 15-2 and 15-3 list the values for each element of the ADSR envelope.

Since the **SO+5** and **SO+6** locations hold the values for two different aspects of the envelope, you must add the Attack and Decay values together to get the value for **SO+5**, and you must add the Sustain and Release values to get the **SO+6** value. This means that you'll be working with Attack and Decay as one value, and Sustain and Release as a different value. To illustrate how the ADSR works, let's set the Attack and Decay to a medium value. Referring to Table 15-2, you'll see that the value for a medium Attack is 64, and the value for a medium decay is 8. Adding these two values together, we get a value of 72. This is the value that we must POKE into location **SO+5**.

Let's set Sustain to a medium value and our Release to 0. Referring to Table 15-3, we'll set Sustain to 64, and Release to 0. We must now put the total value of 64 (64+0) into location **SO+6**. By listing and editing lines 268 and 279, we can set variable **AD** (Attack/Decay) to 72, and variable **SR** (Sustain/Release) to 64. Run the music program to hear the new envelope we've set.

To see where the program sets the envelope values, list lines 350 and 360. You'll see that variable **AD** is POKEd into location **SO+5**, and variable **SR** is POKEd into location **SO+6**.

These examples demonstrate that you can produce a wide range of different synthesized sounds by changing the waveform and the ADSR envelope.

Tempo is the speed of your music. The tempo is not normally set by a SID chip location, but by the program itself. The amount of time it takes for each note to complete its cycle depends on the rates you have picked for the ADSR envelope. For example, if your note has a long

Tempo

SUSTAIN/RELEASE SETTINGS

	LOCATION	HIGH SUSTAIN	MEDIUM SUSTAIN	LOW SUSTAIN	LOWEST SUSTAIN	HIGH RELEASE	MEDIUM RELEASE	LOW RELEASE	LOWEST RELEASE
VOICE 1	S+6 — 54278	128	64	32	16	8	4	2	1
VOICE 2	S+13 — 54285	128	64	32	16	8	4	2	1
VOICE 3	S+20 — 54292	128	64	32	16	8	4	2	1

TABLE 15-3. Sustain/Release rate settings.

Attack and a long Decay, you will need to have a pause after the note to allow the note to go through its complete cycle. When the note is turned on, it goes through the Attack, Decay, and Sustain cycles, but the Release cycle is not executed until the note is turned off (line 820). By using a delay loop (line 810) after the time the note is turned on, and another loop (line 830) after the note is turned off, we can provide the delays necessary to have the note complete its cycle. By adjusting the variable **TE** in line 300, we may increase or decrease the time between notes. This changes the tempo of the song.

Now that we've examined each part of our music program, try adjusting the variables in line 240 through 300 to create your own sounds. Also try programming your own songs by writing the appropriate **DATA** statements, starting at line 900. This program was provided so that you could experiment with the different parts of the SID chip, so don't be afraid to change the program. You can probably learn most quickly by changing one setting at a time and listening to its effect before making another change.

All our examples so far have used only one voice. You are now ready to program the other two voices of the SID chip. Remember that the SID chip starts at location 54272, and that we've set the variable **SO** to this value. Then we set the other locations of the SID chip by offsetting the **SO** value by the appropriate amount to give us all the other SID chip locations. You may recall that we used locations **SO** and **SO+1** to set our frequency, and locations **SO+2** and **SO+3** to set the pulse width. Location **SO+4** was used to set the waveform and **SO+5** and **SO+6** were used to adjust the ADSR. Altogether, we used seven consecutive locations to adjust the sound output for voice 1.

We can set the characteristics of voice 2 at the next location after the voice 1 characteristics. If you're using the variable **SO** as the beginning

TIME TO EXPERIMENT

MULTIPLE VOICES

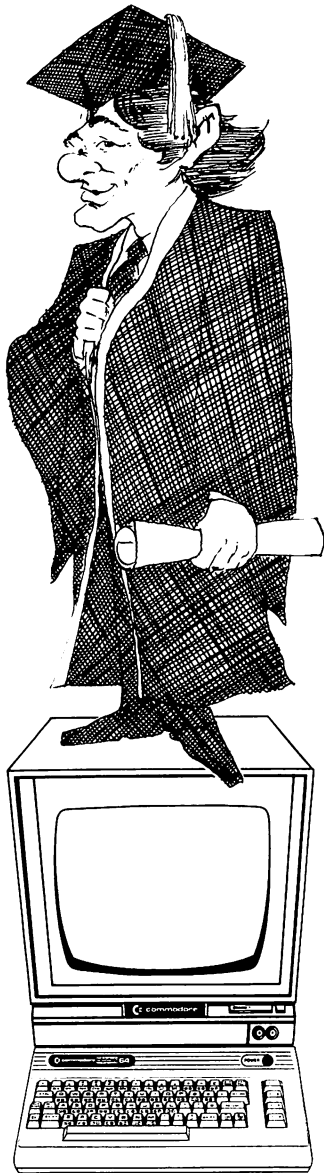
of the SID chip, this value is also the beginning location for voice 1. To use voice 2, set a new variable, **S2**, to equal **S0** plus 7 ($S2=S0+7$). To set any of the registers in voice 2, use the variable **S2** in place of variable **S0**.

If you want to program voice 3, use another variable with a value seven higher than voice 2 ($S3=S2+7$). See Appendix H for a complete table of all the sound register locations. Here's a list of the starting locations of the three voices:

VOICE # 1		54272
VOICE # 2		54279
VOICE # 3		54286

When you have become familiar with the various SID chip features described above, you may want to try some of the chip's more advanced features such as programmable filters, synchronization, and ring modulation. To find out more about these advanced features, refer to the *Commodore 64 Programmer's Reference Guide*.

ADVANCED SOUND FEATURES



PART 4

Advanced Section

One of the most important features of computers is their ability to manipulate data quickly and precisely. In part, this speed and accuracy depends on the computer's ability to store data in an organized way for future retrieval. For example, you can use a word processor program to write and edit a letter to a friend. When the letter is completed, it can be stored on tape or disk in the form of a data file. If you later want to make changes in the wording of the letter, you can load this data file back into the computer using the word processor.

THE IMPORTANCE OF FILES

Data files may also be used to store numerical data that is generated by database programs. These programs are used to create and manipulate numerical data files. Data files, with a computer and the appropriate software, allow you to retrieve or alter your numerical information very quickly. Also, storage of your documents or records as data files on tape or disk takes up far less physical space than keeping the same records on paper.

In this chapter, we'll see how the computer handles several types of files. We'll use some BASIC programming keywords such as STRINGS, INPUT, ARRAYS, IF/THEN statements, and FOR/NEXT LOOPS in our examples. If you're not familiar with these concepts, please refer to a book on BASIC programming that discusses them.

You'll recall from Chapter 8 that a file is a collection of information. Think of a file as a file folder that contains information on pieces of paper. Each of these pieces of paper, devoted to a single subject, is called a record. Each item within a record, such as the last name of a client, or his address, is called a field.

WHAT IS A FILE?

As we saw in Chapter 8, there are four different kinds of files we can use with the C-64 computer: PROGRAM, SEQUENTIAL, RELATIVE, and USER FILES. The two most commonly used kinds of files

are program and sequential files. They may be used with either tape or disk. The other two file types are relative and user files. These two kinds of files work only with a disk drive.

Because we have discussed programs in Chapter 8, and because user files are rarely encountered, we'll treat program and user files only briefly. Instead, the main emphasis of this chapter is on sequential and relative files.

The most commonly encountered type of file is the program file. By now, you have probably loaded and saved programs on tape or disk.

As a review, the format for loading and saving programs using tape is shown below (where <fn> stands for file name, the name of the program. Don't forget the quotation marks.

```
LOAD "<fn>"
SAVE "<fn>"
```

For disk, you must include the device number, like this:

```
LOAD "<fn>".8
SAVE "<fn>".8
```

The tape drive has a device number of 1. The number '1' is the default number for LOAD or SAVE. This means that if you leave out the device number, a LOAD will be initiated from the tape drive. Only one tape drive may be used with the C-64, so there's no need to identify which tape drive is to be accessed.

The disk drive's standard device number is 8. If you're using more than one disk drive, you can change the number of a drive other than the first to any value from 9 to 11 (see your disk drive manual for details). This means that you may have up to 4 disk drives connected to your C-64 at one time, each assigned a different device number. You can use the device number to tell the computer which disk drive you want to access for a LOAD or SAVE.

When we have used a program to generate a large amount of data, we can also use the program to store our data in a "sequential" format. Sequential files are pieces of information, or data, stored one after another on tape or disk. A sequential file can be compared to a long ticker tape with the data written one character after the other along the length of the tape, as shown in Figure 16-1.

PROGRAM FILES

LOAD and SAVE Commands

Device Numbers

SEQUENTIAL FILES

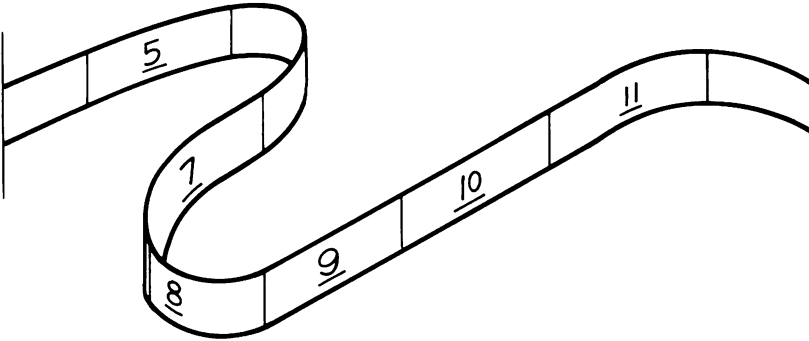


FIGURE 16-1. Sequentially stored data.

To return to our file folder analogy, fields of information are stored one after the other, with no regard for where they fall on a page.

This sequential file can then be read back into the computer at a later time in the same sequential order. It is quite easy for a programmer to set up sequential file storage when writing a data-oriented program. Unlike relative files, sequential files can be used with tape as well as disk. The main disadvantage of a sequential file is that when you make changes to it, the entire file must be reorganized and then resaved. That is, since data is stored one character after the other, if you want to insert data in the middle of the sequence, the entire file must be altered to accommodate the change, and then the file must be saved in the new form. You can't just change one part of a sequential file, leaving the rest intact.

To see how to use a sequential file, let's first go over a few BASIC commands that are reserved for writing and reading data in the form of files on tape or disk.

To create a sequential file, we must first tell the computer to OPEN the file and assign the file a number. This is like writing a number on a file folder for identification purposes, and then opening the file folder so that it can receive the pages of information that we want to place into it. A file may be assigned any number up to 127.

The OPEN Statement

Let's look at the example below for OPENing file number 6 to the tape drive. *For Tape*

```
OPEN 6,1,0,"NAMES"
```

The number 6 is the file number for the sequential file called NAMES. The file now has both a number and a name assigned to it. The name

makes it easier for us to remember what information is in the file. The second number is the device number. In this case the number is 1, the device number for the tape drive. The third number is called the secondary address. It is used differently by different peripheral devices. When you use a tape drive (device number 1), the secondary address is used to indicate whether we are reading from, or writing to, our sequential file. If we want to read the contents of a file so that the data is entered into the computer from tape, we must use a secondary address of 0. If we want to write, or store data, to the tape drive, we must use a secondary address of 1. By OPENing the file with a secondary address of 0, we are telling the computer that we want to read file number 6, called NAMES, from the tape drive.

To open the sequential file 'NAMES' so that we can write to it, the OPEN command is sent like this (but don't type in this example):

```
OPEN 6,1,1,"NAMES"
```

If we use a disk drive, the syntax is slightly different than for tape. The secondary address is not used to designate the direction of the data (read or write) as with tape. Instead, we use a syntax embedded inside the file name, like this (but don't type in this example):

For Disk

```
OPEN 3,8,3,"NAMES,S,R"
```

To tell the computer what type of disk file we want to open, and whether we wish to read or write data to this file, we include a two-letter code at the end of the file name. To indicate that we are OPENing a Sequential file, we use the letter 'S'. To indicate the direction that the data will be moving, we use an 'R' for Reading the file. To Write to the file, we use a 'W'.

Note that even though the secondary address is not needed to designate reading from, or writing to, sequential files on disk, you need to put a number in the secondary address position as a place holder. This number can be any number from 2 to 14. For the sake of convenience only, use the same number as the file number for the secondary address, unless the file number is 1. In the example above, the file number and secondary address numbers are both 6. If a secondary address is not used, you will get a syntax error.

Here is another example of opening a file. To OPEN sequential file number 3 for writing data to disk drive number 8, using a file name NAMES, type:

```
OPEN 3,8,3,"NAMES,S,W" (don't type this example)
```

After OPENing a file, data is sent to it. When we are finished with the file, it must be properly closed before we can go on to something else. Here is the proper way to close file number 3:

```
CLOSE 3
```

After a file is opened, we can either read or write our sequential data using tape or disk. This is done with the BASIC commands PRINT# and INPUT#. These two commands work exactly like the normal PRINT and INPUT commands, except that a file number is required for both PRINT# and INPUT# commands. PRINT# sends data to tape or disk; INPUT# reads stored data from tape or disk and puts it into the computer.

To see how to use the OPEN, PRINT#, and INPUT# commands, enter the following examples. The first example is for tape, the second is for disk.

```
10 FOR A=1 TO 10
20 READ A$(A)
30 NEXT A
40 OPEN 2,1,1,"TEST"
50 FOR A=1 TO 10
60 PRINT#2,A$(A)
70 NEXT A
80 CLOSE 2
90 DATA ZERO,ONE,TWO,THREE
100 DATA FOUR,FIVE,SIX,SEVEN
110 DATA EIGHT,NINE
```

Tape WRITE

```
10 FOR A=1 TO 10
20 READ A$(A)
30 NEXT A
40 OPEN 2,8,2,"TEST,S,W"
50 FOR A=1 TO 10
60 PRINT#2,A$(A);CHR$(13);
70 NEXT A
80 CLOSE 2
90 DATA ZERO,ONE,TWO,THREE
100 DATA FOUR,FIVE,SIX,SEVEN
110 DATA EIGHT,NINE
```

Disk WRITE

You'll notice that the programs above are identical except for lines 40 and 60. Figure 16-2 shows the differences in line 40. When using

Closing Files

Writing to Sequential Files

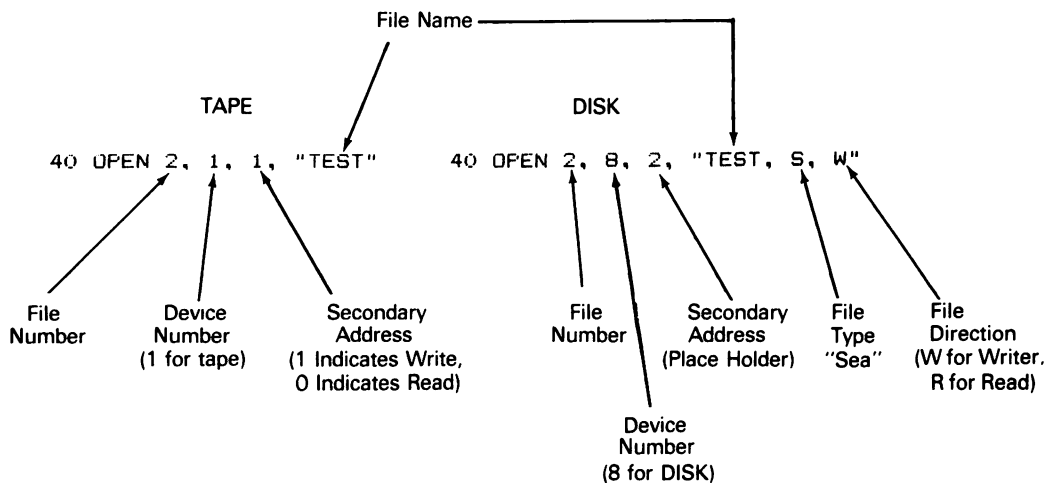


FIGURE 16-2. The OPEN statement.

disk, you must tell the computer that this file is being written to the disk by placing 'S,W' (for Sequential, Write) inside the quotes after the file name, as shown in line 40 for disk. With the tape version, you must use a secondary address of 1 to tell the computer that this file is to be written to tape, as shown in line 40 for tape. The secondary address is the third number of the tape OPEN statement in Figure 16-2.

You must keep in mind that, for tape files, this secondary address number of 1 means that we are going to write to this file. The secondary address for the disk file is simply a place holder, having no meaning for the computer.

Figure 16-3 compares the tape and disk versions of line 60. When you send data to the disk drive with the PRINT# command, you must suppress the line feed that is normally sent at the end of each piece of information, or field. A piece of information here is represented by the variable `AS(A)`. You can suppress the line feed by placing a semicolon (;) after printing a `CHR$(13)` at the end of the PRINT# statement. The `CHR$(13)` is the carriage return code, which is necessary to indicate the end of a field of information.

When the sequential write program for disk is run, the red light of the disk drive will come on, and you'll hear the motor of the drive whirl. If you look at your disk directory after the red light goes off, you'll see a sequential file called TEST.

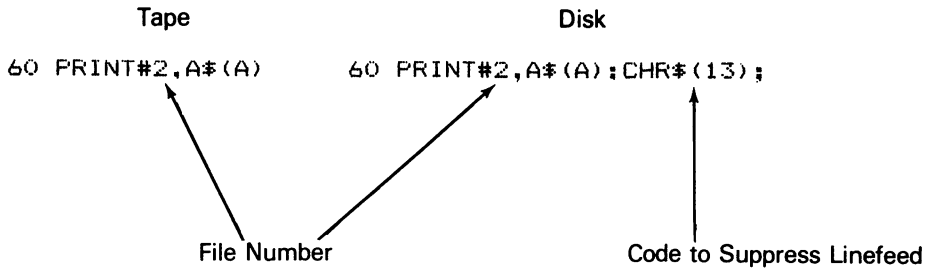


FIGURE 16-3. The PRINT statement.

In the next example we'll read our sequential file back into the computer and have the data printed to the screen.

Reading a Sequential File

```
10 OPEN 2,1,0,"TEST"
20 FOR A=1 TO 10
30 INPUT#2,A$(A)
40 PRINT A$(A)
50 NEXT A
60 CLOSE 2
```

Tape READ

```
10 OPEN 2,8,2,"TEST,S,R"
20 FOR A=1 TO 10
30 INPUT#2,A$(A)
40 PRINT A$(A)
50 NEXT A
60 CLOSE 2
```

Disk READ

The OPEN statements in the READ examples are similar to those used in the WRITE programs. The only differences are that the '1' was changed to a '0' in the tape version, and the 'W' was replaced with an 'R' (for Read) in the disk version. Also, you are now INPUTting the data into the computer instead of PRINTing the data out to a device.

Reading or writing a sequential file is usually a straightforward procedure, but there are a couple of things to keep in mind. First, remember that the PRINT# works the same as a normal PRINT statement, except that the data is sent to an external device. This means that all the rules that apply to the PRINT statement also apply

to the PRINT# statement. For example, you must not use commas between variables when using the PRINT# statement, otherwise unwanted extra spaces will be placed in your file, in the same way that spaces are created on the screen by commas in PRINT statements. (Remember, you can't abbreviate PRINT# using ?#.)

You're not restricted to the type of data you may write to your file. Both strings of characters and numbers may be written to tape or disk. Remember, however, to read the file back in the same way that you wrote it. For example, if the first piece of data that was written to the file was a word, (string), and the second a number, you must indicate when reading this file back that the first piece of data is to be a string, and the second a number. The examples below will write files with both string and number variables:

```
10 FOR A=1 TO 10
20 READ A$(A),A(A)
30 NEXT A
40 OPEN 2,1,1,"2 TEST"
50 FOR A=1 TO 10
60 PRINT#2,A$(A):PRINT#2,A(A)
70 NEXT A
80 CLOSE 2
90 DATA ZERO,0,ONE,1,TWO,2,THREE,3
100 DATA FOUR,4,FIVE,5,SIX,6,SEVEN,7
110 DATA EIGHT,8,NINE,9
```

Tape WRITE

```
10 FOR A=1 TO 10
20 READ A$(A),A(A)
30 NEXT A
40 OPEN 2,8,2,"2 TEST,S,W"
50 FOR A=1 TO 10
60 PRINT#2,A$(A);CHR$(13);:PRINT#2,A(A);
   CHR$(13);
70 NEXT A
80 CLOSE 2
90 DATA ZERO,0,ONE,1,TWO,2,THREE,3
100 DATA FOUR,4,FIVE,5,SIX,6,SEVEN,7
110 DATA EIGHT,8,NINE,9
```

Disk WRITE

You'll notice that the word 'ZERO' is read into array A\$(A), which is then PRINT#ed to file number 2. A\$(A) is a string variable and may be handled as any normal string. Also, in the same sequential file, we are PRINT#ing the numeric variable A(A). You may mix data for both types of variables in your sequential file, but when you READ

the data back into the computer, you must make sure to match each piece of data with the proper type of variable. If the data being read from the file is string data, then this data must be read into a string variable. In a similar way, numeric data must be read into a numeric variable. In the following example you will notice that the string and numeric data are read back in the same order as they were written.

```
10 OPEN 2,1,0,"2 TEST"  
20 FOR A=1 TO 10  
30 INPUT#2,A$(A),A(A)  
40 PRINT A$(A);A(A)  
50 NEXT A  
60 CLOSE 2
```

Tape READ

```
10 OPEN 2,8,2,"2 TEST,S,R"  
20 FOR A=1 TO 10  
30 INPUT#2,A$(A),A(A)  
40 PRINT A$(A);A(A)  
50 NEXT A  
60 CLOSE 2
```

Disk READ

Sequential files have many important applications for storing and retrieving data. To create your own sequential files, remember to first OPEN the file you want to read or write. Then, by following the procedures we've shown in the examples above, you can build your own data files. Keep in mind that you cannot load sequential files directly into your computer. You need a program like the examples you've seen to read this type of file.

Sequential files are a good way to store a fixed block of data that needs little manipulation after the file is created. Since data in a sequential file is written and read in a sequential order, you must load the entire file into the computer to make any corrections or changes to the data. Once corrections have been made, you must rewrite the entire file back to tape or disk in order to retain the corrections. If you need to make a data file that is too large for the computer to hold in its memory, or if you expect that you will need to alter a small part of a large data file, you should use a relative file rather than a sequential file.

Sequential files are commonly used for large batches of data that are to be loaded into the computer all at once. However, if you want to set up files for a mailing list, for example, you must have the ability to select the name and address of just one person from the file. With a sequential file, you would have to load in the data for your entire

RELATIVE FILES

mailing list, and then search this data from the beginning to locate a specific record you need—in this example, the person's name and address.

Relative files give us the capability for storing and using a mailing list. If we store data in a relative file rather than a sequential file, we can specify which record we wish to load into the computer, and it will load the record quickly and precisely. A relative file is set up so that the computer can locate a particular record on the diskette without looking at other records. The disk drive will find the specific name and address we need without having to search through all the other files, and will load the record into the computer. This sort of nonsequential data retrieval is called *random access*. Random access is what makes relative files so powerful.

The relative file may be used with a disk drive but not with a tape drive, because the tape is a sequential device without the capability for accessing information randomly.

In a relative file, each record has internal pointers to help the disk drive keep track of where each record is stored. The disk drive uses the BAM of the diskette to keep track of each record by its number. When you ask the disk drive to find a particular record, the BAM tells the disk drive where that record is on the diskette. Then the disk drive head can move quickly to the beginning of that record without going through the other records. Figure 16-4 illustrates how relative files are recorded on a diskette.

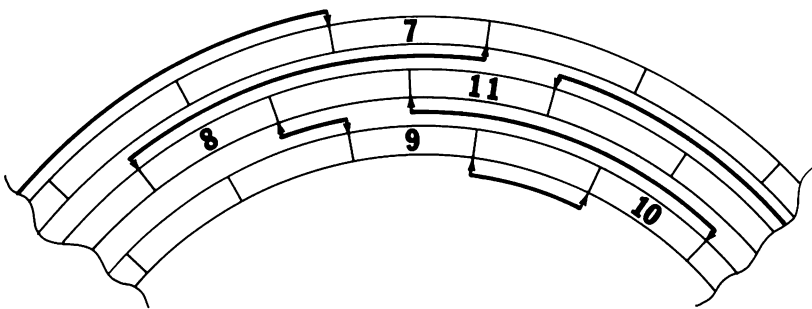


FIGURE 16-4. Data filed as relative file.

To set up a relative file, you need to open two files. The first OPEN statement is used to set the file number, file name, and record size. It opens the file that will accept our data. The second OPEN statement is used to open the disk channel for sending commands to the disk drive. We'll need the disk command channel open so that we can help the disk drive find the records that we save in our relative file.

Let's begin by writing a short BASIC program that will create a relative file. First, we'll open a relative file named NUMBERS. When we open a relative file, we must specify the maximum size of a record so that the disk drive can keep track of our records by computing where one ends and another starts. In this example, let's set our maximum record size to 35 characters. Then we'll open the disk drive channel for future use. This is how it's done:

```
10 OPEN 2,8,2,"NUMBERS,L,"+CHR$(35)
20 OPEN 15,8,15
```

In line 10, the file name NUMBERS is opened to device number 8 (the disk drive). Both the file number and the secondary number are 2. We could have used any number up to 127, except 1 and 15 for the secondary address. The 'L' inside the quotes is used to designate a relative file. The CHR\$(35) tells the disk drive to get ready to receive records with a maximum length of 35 characters. The 35 includes a carriage return at the end of each record, so each of our records can have no more than 34 characters. Note that when we open a relative file, we don't have to indicate whether we are reading or writing, as we must for sequential files.

The OPEN 15,8,15 in line 20 opens the command channel. We'll use this channel later to help the disk drive find where the record has been stored.

The next five lines of the program request the user to input information that is put into the first ten record blocks. Remember that we have already set the maximum length of each record to 34 characters plus one carriage return, so when the program asks you to input data into each of the 10 records, make sure that each piece of data is no more than 34 characters in length.

```
30 FOR A=1 TO 10
40 PRINT "ENTER RECORD[SPACE] ";A
50 INPUT A$
60 PRINT#15,"P"+CHR$(2)CHR$(A)CHR$(0)CHR$(1)
70 PRINT#2,A$;CHR$(13);:NEXT A:CLOSE2
```

Writing a Relative File

Line 60 provides the power characteristic of a relative file. This line of BASIC sets the pointer ("P") for reading or writing to your desired record block. Working from left to right, let's look at exactly how this line works.

The PRINT#15 sends this command line to the disk drive's command channel, which we opened earlier in line 20. The rest of the line sets our record pointer.

PRINT#15

The '2' in the first CHR\$ tells the computer that this pointer is set up for file number 2, which we opened in line 10.

CHR\$(2)

The next two CHR\$s tell the computer which record we want to read or write. CHR\$(A) and CHR\$(0) are in the low-byte high-byte format. CHR\$(A) points to record numbers 1 to 255; CHR\$(0) points to record numbers above 256.

CHR\$(A) and CHR\$(0)

We have defined A in line 30 as a number variable that goes from 1 to 10. At first, A=1. This means that the first record entered into the file is labelled number 1. The second record is number 2, the third 3, and so on until we reach record 10. Once the program is running and we have entered each of the 10 records, each record will have its own assigned number. Whenever we want to retrieve a record from our relative file, we indicate what its number is, and the disk drive will send the drive head directly to that record, read the record, and display it for us on the screen.

In this program, we have fewer than 255 records per file, so the second (high-byte) CHR\$ is equal to 0.

If you were using more than 255 records, you would use the following format to calculate the CHR\$ values for these two CHR\$s:

$$\text{RECORD NUMBER} = \text{low byte} + (\text{high byte} * 256),$$

where high byte is the quotient of the record number divided by 256 and low byte is the remainder. For example, to access record number 470 using the record pointer, we would use the PRINT#15 command like this:

```
PRINT#15, "P"CHR$(2)CHR$(214)CHR$(1)CHR$(1)
```

The CHR\$(2) identifies the file. Since $470/256 = 1$ (high byte) with a remainder of 214 (low byte), the second CHR\$ is 214, and the third CHR\$ is 1.

The last CHR\$ is used to set the location within the record. This pointer is primarily used when we want multiple fields in each record. Multiple fields are discussed later in this chapter. Since we're using only one field in this example, the CHR\$ is set to 1, the first location within the record. *CHR\$(1)*

Line 60 does not write records to the disk; it only sets the pointer for a read or write. In our example, the PRINT# command in line 70 is the command that actually writes our records to the disk. After all 10 records are written, we close file 2 to empty the BAM buffer of the disk drive. Always remember to close a file when you've finished writing to it. *Writing to the Disk*

When the program is run, when we've entered all 10 records, and the program has written them to the disk, we will need a routine to read our data back into the computer. Remember that the main advantage of a relative file is its ability to access any record immediately through a random, rather than a sequential, searching process. To demonstrate the power of relative files, let's add a routine that gives us the option of choosing which record block we want to retrieve from the disk. **Reading a Relative File**

```
80 OPEN 2,8,2,"NUMBERS"
90 INPUT "ENTER RECORD NUMBER"; A
100 IF A=0 THEN CLOSE 15:CLOSE 2:END
110 PRINT#15,"P"CHR$(2)CHR$(A)CHR$(0)
CHR$(1)
120 INPUT#15,D,E$,F$,G$:IF D=50 THEN
PRINT E$:GOTO 90
130 INPUT#2,A$
140 PRINT A$:GOTO 90
```

Since we closed our file in line 70 after writing to it, we must reopen the file in line 80 before we can read it. You can see by examining lines 60 and 110, that the record pointer is used the same way for either reading or writing files. Run the program above and enter the records called for (after the question mark displayed on the screen). Then enter any record number from 1 to 10 to have that particular record printed to the screen. In this example, we put 10 records in our file. If we asked for record number 11, we would get a 'RECORD NOT PRESENT' error. To check for the occurrence of this error, we used line 120 to read the error channel. If the error number (D) is equal to 50 (the error

for RECORD NOT PRESENT), then we print the error message (E\$), and go back to line 90.

Any time you open a file or channel, you should make sure that it is closed again before you leave the program. Line 100 allows us to close channel 15 and file 2 by entering a 0 for our record number. Then it ends the program.

In the example program above, we created a relative file with 10 records. Each record had only one piece of data, or field, within it. There are many applications that require more than one field within each record. An example of such a file is a mailing list. Usually three fields are used: the first field is for the name, the second field is for the street address, and the third field is for the city, state, and zip code. In relative files, the maximum number of characters for each record is 254. To create a record with three fields, we have to remember that the total length must be no more than 254 characters, including the carriage return that is needed at the end of each field as a separator between fields.

In the example below, you may enter up to 255 records, with each field within a record having no more than 64 characters. The 65th character is reserved for the carriage return. When you have finished entering your data, type 'DONE' in response to the prompt for a name. This will cause the computer to leave the write file routine and enter the read file routine.

Multiple Fields

```

10 OPEN 2,8,2,"ADDRESSES,L,"+CHR$(196)
20 OPEN 15,8,15
30 A=1:X1=1:X2=66:X3=131
40 PRINT "NAME"
50 INPUT NA$:IF LEN(NA$)>64 THEN 40
60 IF NA$="DONE" THEN A=A-1:GOTO 150
70 PRINT "STREET ADDRESS"
80 INPUT SA$:IF LEN(SA$)64 THEN 70
90 PRINT "CITY STATE & ZIP (NO COMMAS )"
100 INPUT CS$:IF LEN(CS$)>64 THEN 90
110 A$=NA$:X=X1:GOSUB 270
120 A$=SA$:X=X2:GOSUB 270
130 A$=CS$:X=X3:GOSUB 270
140 A=A+1:GOTO 40
150 CLOSE 2:OPEN 2,8,2, "ADDRESSES"
160 INPUT "ENTER RECORD NUMBER";B
170 IF B=0 THEN CLOSE 2:CLOSE 15:END
180 INPUT "ENTER FIELD NUMBER";C
190 IF C<1 OR C>3 THEN 180

```

```

200 IF C=1 THEN X=X1
210 IF C=2 THEN X=X2
220 IF C=3 THEN X=X3
230 PRINT#15,"P"CHR$(2) CHR$(B)' CHR$(0)
CHR$(X)
240 INPUT#15,D,E$,F$,G$:IF D=50 THEN PRINT
"[RVS ON]"E$:GOTO 160
250 INPUT#2,A$
260 PRINT A$:GOTO 160
270 PRINT#15,"P" CHR$(2) CHR$(A) CHR$(0)
CHR$(X)
280 PRINT#2,A$;CHR$(13);:RETURN

```

At first glance, it would seem that using multiple fields is more complex than using a single field. The only difference between the single field and multiple field examples is that we have included a checking routine to make sure our maximum field length is kept to 65 characters or under. If you look at lines 230 and 270, you will notice that the only difference from the single field example (lines 60 and 110) is the last CHR\$ in both lines. This is the character pointer within the record block. We need to adjust this pointer so that it points to the first character of whichever field we need to read or write. The possible values for this pointer location are on line 30, with X1 equal to 1 (the first character in field 1), X2 equal to 66 (the first character in field 2), and X3 equal to 131 (the first character in field 3).

In writing your programs, you may increase or decrease the number of fields or the size of any field, but keep in mind that all fields combined must total no more than 255. If you'd like to have more than 255 records, you must increase the third CHR\$ by 1 after every 255 records.

As you can see, using a relative file requires a more complex program than a sequential file, but the amount of time spent loading records, changing them, and then resaving them on disk is greatly reduced. The more complex database programs combine sequential and relative files in the same program. A sequential file is used to keep track of major field headings such as NAME, ADDRESS, and CITY/STATE/ZIP. A relative file is used to hold the actual data. In this type of *hybrid file* system, you can use the sequential file to keep track of the relationship between the record number and the name assigned to each record. Instead of asking for record number 36, you would be able to request the record by the name that you have associated to that record. For example, you could request the file for SMITH rather than file number 36.

HYBRID FILES

Our discussion of various types of files will prepare you for writing your own small database programs using either sequential or relative files. Even if you don't plan to write your own programs, you will understand better how to use commercially available programs now that you understand how the programs use files.

**NOW THAT
YOU KNOW
SOMETHING
ABOUT FILES**

Machine Language

If you have played any of the arcade-quality games available for the C-64, you have probably noticed that some of these games are much faster-paced and more exciting than others. Those of you who use your computers for business applications may have found that some business programs run much faster than others. These differences in speed are usually due to differences in the program languages.

You'll recall that BASIC is a high-level language, easy to learn and use for programming. Since the computer understands only machine language, a BASIC interpreter is required to translate the English BASIC words into numbers that the machine understands. This translation process greatly slows the running of BASIC programs.

Game programs that run fast enough to make your joystick hand wet with perspiration, and business programs that run quickly and efficiently, were probably written in either assembly language or machine code. After you've become somewhat familiar with BASIC, you may feel an urge to explore the exciting world of machine language.

Although programming in machine language is not easy to master, certain BASIC commands can help you learn about machine language programming. The BASIC commands PEEK, POKE, and SYS will help you write machine code instructions within BASIC programs. Your first entry into the world of machine language will probably be with the BASIC keywords PEEK and POKE. PEEK is used to read a value from a particular memory location, and POKE is used to place a value into a particular location.

Imagine a set of post office boxes. If we wish to see the contents of box number 15560, we can "peek" into box 15560 and examine its contents. In a similar way, we can PEEK into a specific memory address in the memory registers of the computer to find out what number is in that memory location.

MACHINE LANGUAGE THROUGH BASIC

Why Machine Language?

Using Machine Language With PEEK and POKE

The PEEK Command

If we want to put something into post office box 23432, we first locate the correct box and then put into it whatever we wish. Similarly, when we want to put a value into some memory location of the computer, we must have the address of the location. Unlike a post office box, which can hold several pieces of mail at once, each location of the computer can hold only one value at a time. POKEing a value to a memory location causes the computer to find the location and, in effect, to remove the old value and replace it with the new value.

The POKE Command

When you use the PEEK command, you must include a memory location to PEEK into. Here's an example for you to try; type in:

Using PEEK

```
PRINT PEEK(650)
```

The screen displays this:

```
PRINT PEEK(650)
0
```

This PEEK which you entered in direct mode prints a 0, the contents of memory location 650, to the screen. You can also place the PEEK value into a variable. Type in:

```
A=PEEK(650)
```

In this way, you may store the PEEK value in variable A for future use. To show how this works, type:

```
?A
```

Once again, the screen displays '0', the contents of memory location 650. You can PEEK any memory register address—from 0 to 65535. Recall that since the C-64 has 64K of memory space, the largest memory location address that it can have is 65535 (64 times 1024 minus one). The PEEK value can be any number from 0 to 255. Remember that there are 256 possible numbers (0-255) that an 8-bit computer can hold in a single byte.

The POKE command is used to place a value into a particular memory location. Let's try POKEing the value of 128 into location 650:

Using Poke

```
POKE 650, 128
```

Notice that no parentheses are used around 650, and that a comma is used to separate the memory location from the value we want to place

into that location. To see what we've done, let's PEEK location 650 and see if our POKE was successful. Type in:

```
PRINT PEEK(650)
```

The screen will show:

```
PRINT PEEK(650)  
128
```

If you don't have a 128 printed to the screen, check carefully that you have typed the location and value numbers correctly in both the POKE and PEEK statements.

What function do you think POKEing this value had? To find out what our POKE has done, hold any key down and see what happens. You will find that all keys will repeat when held down. Without the POKE, only the SPACE bar, cursor control keys, and INSERT and DELETE keys would repeat. Our POKE changed the value of the location 650, called the "REPEAT FLAG", from 0 to 128, which caused the computer to repeat any key pressed. Actually, repeat is turned on by POKEing location 650 with any number over 127. POKE 650, 127 (or any number from 64 through 127) turns off the repeat feature of all keys, even the SPACE, cursor control, and INSERT/DELETE keys. This repeat on all keys will stay in effect until we either change the value in location 650 back to a number between 0 and 63, or until we turn the computer off and on again.

Some other important memory locations are the border, background, and character color locations. To change the color of the border, we use memory location 53280. For example, to change the border color to black, do the following POKE:

```
POKE 53280,0
```

You'll see that when you press [RETURN], the border color instantly changes to black.

To change the background to red, we'll POKE the code for red into location 53281:

```
POKE 53281,2
```

You may also use a POKE to change the character color by POKEing memory location 646:

```
POKE 646,1
```

The '1' changes the character color to white. A list of all the colors and their POKE values appears in Appendix D.

When you PEEK into memory locations 53280 and 53281 (the locations that contain the border and background color values) you must subtract 240 from the value the screen displays to get the true color value. Try this example—type in:

```
PRINT PEEK( 53281)
```

The screen displays the value 242, which is 240 more than the '2' we POKEd in earlier to make the screen red. Now type:

```
PRINT PEEK( 53281) - 240
```

The screen displays '2', the correct screen color value.

The numbers we've been POKEing and PEEKing certainly have significance for the computer—for instance, '1' means white, and '2' means red. But the significance of the number isn't due to the number alone. The memory location into which the number is stored is also of critical importance for the computer. The POKEd number '0', for example, can have many different meanings depending on the assigned function of the memory location that the '0' is POKEd into. In location 53280, the '0' will cause the border color to turn to black. As we saw in the above example, a '0' in location 650 causes a repeat on the CURSOR keys, the SPACE key, the INSERT/DELETE key and the RETURN key.

So how are we to anticipate what effect various numbers in different locations will have on the computer? What we need is a reference table that contains all the important RAM locations of the computer, their functions, and the effects of different values placed in each of these locations. Such a chart is called a *memory map*. There is a good memory map in *The Commodore 64 Programmer's Reference Guide*, published by Commodore Business Machines and Howard W. Sams and Company, 1982. Another excellent publication that provides even more information in a memory map format is *The Master Memory Map for the Commodore 64*, published by Reston Publishing Company.

We've been using the PEEK and POKE commands to examine and change the values in various memory locations, and we've used the PEEK and POKE commands in direct mode to do some simple operations. Now let's try using these commands in a BASIC program. Enter the following lines of BASIC:

*Values PEEKed
and POKEd*

*Programming With
PEEKs and POKEs*

```
NEW [RETURN]
10 PRINT "[CLR]";
20 A=INT(RND(0)*(16)
30 POKE 646,A
40 PRINT "[RVS ON][SPACE]";
50 GOTO 20
```

Make sure to include the **SPACE** after the **REVERSE ON** command inside quotes on line 40. When this program is run, the screen is cleared, and then randomly colored boxes (**SPACES**) are printed to the screen. To break out of the program, press **[STOP]**. Line 20 sets the value of **A** to a random number between 0 and 15 (random numbers are discussed in Chapter 22). In line 30, this random number is **POKEd** into location 646 (the location that contains the value for the character color).

Let's now write a short program that will reverse all 1000 locations of the screen—that is, each character will be printed in the background color:

```
NEW

10 A=1024:B=55296:C=PEEK(646)
20 FOR I=0 TO 999
30 X=PEEK(I+A)
40 X=X+128:IF X>255 THEN X=X--256
50 POKE I+A,X
60 POKE I+B,C
70 NEXT I
```

When this program is run, all characters on the screen will be flipped to their reverse. This happens because the values 128-255 **POKEd** into screen memory locations produce characters that are the reversed images of the characters whose values are 0-127. The number 1024 in line 10 is the beginning memory location of the screen. There are 1000 bytes starting at 1024 which are reserved for the screen memory (40 columns times 25 rows). In line 30, the variable **X** is set to the value that is stored in location **A** plus the variable value of **I**. The value **I+A** is the screen location. The variable **I** is increased by one each time until it reaches 1000. When it is added to the first screen location of 1024 (variable **A**), the variable **I** keeps track of each of the 1000 screen locations. Line 40 performs the actual flip. This is done by adding 128 to the **PEEKed** value for the character codes. The result is stored in variable **X**. Line 50 **POKEs** the new value of **X** back to location **I+A**.

The number 55296 (variable **B**) is the beginning location for the screen colors. Any time you POKE a value to screen memory, it is wise to POKE a color in its matching location in screen color memory. For the example above, we used the value stored in location 646, which is the value for the current character color. This character color value is POKEd into all 1000 color memory locations to make the screen the same color as the character (line 60). The screen memory goes from 1024 to 2023, and the color memory goes from 55296 to 56295. Both the screen and color memory locations have 1000 post office boxes reserved for holding information. The number 1000 is obtained by multiplying 40 (screen columns) by 25 (screen rows). The difference between screen memory and screen color memory is that the code for the actual character to be displayed to the screen is stored in the appropriate screen memory location, while the code for the color of the character is stored in color memory. To demonstrate this, type the following line in direct mode:

```
A=19:POKE 1024+A,1:POKE 55296+A,5
```

When this line is typed in and [RETURN] is pressed, a green letter 'A' is POKEd to the top center of the screen. Notice that the variable **A** is added to the values of both the screen and color memory locations. A screen and color memory chart is in Appendix F. For a complete listing of color and character POKE values, see Appendices D and C.

Now let's change the program above by first deleting lines 30 and 60. This is done by typing '30' on a blank screen line and hitting [RETURN] and doing the same for '60'. Then change line 40 to this:

```
40 X=32
```

The program will now clear all 1000 bytes of screen memory, leaving you with a blank screen. The value '32' is the POKE value for the SPACE key, and since we POKEd the SPACE character into all 1000 screen locations, all screen characters are overwritten by a space. This is similar to using the [CLR] key, except that it takes about 100 times longer. The reason for this is that the clear screen routine that is executed when you press the [CLR] key is a machine language routine built into the operating system of the C-64.

The short routine you typed in can be useful when you wish to clear only part of the screen, so that other information on the screen is left untouched. For example, to clear only the top half of the screen, we would use a program like the one listed below:

```

10 A=1024
20 FOR I=0 TO 519
40 X=32
50 POKE I+A,X
70 NEXT I

```

There are many ways to use the PEEK and POKE commands to increase the power of your computer, but if speed is your main objective, then you should program in machine language to get the best results. The next section will introduce you to machine language, showing you different ways to write and use machine language programs.

We have seen that we can control the computer by POKEing values into its memory registers. In fact, we could write an entire machine language program by POKEing numbers into the memory registers of the computer manually. This, however, would be an extremely slow and inefficient way of writing programs.

Another way to enter machine instructions in BASIC is by using DATA statements. Type in the following program to see how DATA statements can enter machine language programs into the computer:

Using Machine Language With DATA Statements

```

1000 FOR A=828 TO 868
1010 READ X
1020 POKE A,X
1030 NEXT A
1040 DATA 160,0,169,4,132,140,133,141,169
1050 DATA 216,132,142,133,143,177,140,73,128
1060 DATA 145,140,173,134,2,145,142,200,192,0
1070 DATA 208,240,230,141,230,143,169,8
1080 DATA 197,141,208,230,96

```

This program POKes the 40 numbers in lines 1040 to 1080 into 40 memory locations from 828 to 867 (the cassette buffer). Note that there must be exactly 40 addresses (post office boxes) into which the 40 numbers are put, or POKEd. After checking to make sure that you have entered all the numbers correctly, save the program; this will permit you to use the program in a later demonstration. Now type RUN.

When this program is run, all the POKes are executed, storing a machine language program in locations 828 through 867 (the cassette buffer). You won't see the machine language program running yet

because the BASIC program has only POKEd the machine language program into the memory of the computer. To execute the machine language that has been POKEd into the computer, type this:

```
SYS 828
```

The SYS 828 command tells the computer to go to location 828 and execute all the machine language instructions that it finds starting there. SYS is a command that is similar to the RUN command. When we use the SYS command, we must always specify where the beginning of the machine language program is.

If the DATA statements were entered accurately, when you press [RETURN], all 1000 screen characters are flipped so that the text on the screen is now the color the background used to be, and the new background color is the color that the characters used to be. For example, if the letter 'A' was a black character on a blue background, it will now appear as a blue 'A' on a black background. If we type SYS 828 again, the screen characters will be flipped back. Try it a few times.

This machine language program carries out the same operation as the BASIC program that you typed in earlier. But as you probably noticed, there is a difference in the speed of execution. The first BASIC program took about 15 seconds to execute, the machine language version took less than a second.

The DATA statement values that were POKEd into the memory locations may appear to be random values to the uninitiated. However, these numbers are machine code instructions that have special meanings for the computer. The computer can understand these numbers immediately, without the need for a BASIC interpreter. That's why the machine language program works so much faster than the equivalent BASIC program. It's fine that the computer can understand these numbers, but what about us, the users? How can we look at these numbers and make some sense of their meaning?

A utility program called the *machine language monitor* will help you translate the machine code into assembly language words, or symbols, called *mnemonics*. These symbolic codes are easier for us to understand than groups of numbers. The monitor will not only help you examine machine code, it will also allow you to change the values stored in any memory registers. The machine language monitor allows you to step inside the brains of the computer to examine, and even change, the contents.

MACHINE CODE

The Machine Language Monitor

In this section we will use SUPERMON, a machine language monitor that is part of the utility program POWER PLUS included on the SUPPORT DISK available with this book. For a complete description of how to use SUPERMON, see Chapter 21. If you have POWER PLUS, load and run it now. Then enter SUPERMON by pressing [*] and then [RETURN].

The syntax of other monitors you might obtain for your C-64 may be different from the syntax of SUPERMON, but the operations should be similar. If you are using another machine language monitor, load the program now and enter the monitor.

When you turn on SUPERMON or another machine language monitor, you actually “break,” or go, to a new operating system designed to assist you in writing in machine language. If you are using SUPERMON, you will see a display that looks something like this:

```

      PC  SR  AC  XR  YR  SP
.;CEB6  33  C5  F8  01  F8
.□

```

You probably can't make any sense out of these strange codes and numbers. In order to use machine language, we must learn to write and read numbers in a different number system called the *hexadecimal* number system. The line above the cursor in your screen display is a series of hexadecimal numbers.

The difference between hexadecimal (hex for short) and the familiar decimal system is that with hex numbers we have 16 numerals rather than the 10 we have with decimal numbers. In decimal, we use numerals 0 through 9. In the hex system, six more units are added—the letters 'A' through 'F'. As we count up from 0 in hex, when we reach 9, we use 'A' to represent the number 10. The letters 'B' through 'F' represent the numbers 11 through 15. When we reach 16, we add a second digit; so in hex, the number 16 is represented by 10. To signify that a number is a hex number rather than a decimal number, we'll put a dollar sign (\$) in front of the number. For example, we'll express the hex equivalent of the number 16 as \$10. Study Table 17-1 to become more familiar with hex numbers.

You can see that hex numbers take up less space than decimal numbers, and of course they are much shorter than their binary equivalents comprised of ones and zeros.

With practice, you'll master the hex number system. Is it worth the effort? Yes. Hex is important for working with the computer. Identification and allotment of blocks of memory in the computer are based on the hexadecimal system. For example, the beginning of the screen

Hexadecimal Numbers

DEC	HEX	DEC	HEX
0	0	20	14
1	1	21	15
2	2	22	16
3	3	23	17
4	4	24	18
5	5	25	19
6	6	26	1A
7	7	27	1B
8	8	28	1C
9	9	29	1D
10	A	30	1E
11	B	40	28
12	C	50	32
13	D	60	3D
14	E	70	46
15	F	80	50
16	10	90	5A
17	11	100	64
18	12	200	C8
19	13	1000	3E8

TABLE 17-1. Decimal and hexadecimal numbers.

Let's now return to the machine language program that we typed in earlier. Remember, to enter the machine language monitor in POWER PLUS, press and [RETURN]. If you're using another monitor, refer to the instruction manual for proper entry into the monitor.

memory in decimal is 1024; in hex it is \$400. The beginning of color RAM memory in decimal is 55296, but in hex it is \$D800. Although the computer can interpret either decimal or hex address numbers, when we use machine language through the machine language monitor, we must use hex numbers. It's easier for us to remember hex numbers such as \$400 and \$D800 than the equivalent decimal numbers.

First we'll look at the machine code that we placed at location 828. Remember that 828 is a decimal number. When we use the machine language monitor, we must express all numbers in hex. For this reason, you must convert all the decimal numbers that you're going to use to hex. You may obtain the conversions by running the program HEX/DEC CONVERT provided on the SUPPORT DISK.

The C-64's microprocessor works with numbers. However, we, the masters of these wonderful machines, need an easier way to talk to the computer. Let's review how we can do this.

Languages such as BASIC permit us to use English words when we want to command the computer. The BASIC interpreter translates the English words into numbers that the computer understands. The disadvantage of BASIC is that programs written in this high-level language take up much more memory space, and run much more slowly than those written in machine language.

With a machine language monitor, we can write programs in machine language, but in a form that is easier for us to master. Built into most machine language monitors are two utilities called the *assembler* and the *disassembler*. The assembler allows us to enter machine code using assembly code words, or mnemonics. The mnemonic codes that are used in the C-64 (as well as any 6502 microprocessor derivative computer) are three-letter command codes that stand for instructions to the computer. For example, neither the binary number 10101001 nor the hex number A9 gives you a clue as to its meaning to the computer. However, the code LDA is easily remembered as LoAD the Accumulator. To learn more about machine codes and their mnemonics, refer to a book on programming the 6502 chip using machine language.

Mnemonics

The disassembler allows us to view and change the instructions that make up a machine language program.

The Disassembler

If you just typed in and ran the sample program listed above that POKes a machine language program into locations 828 to 868, you still have that program in the computer. To show you what a machine

Viewing the Program

language program looks like, we'll display a disassembly of this machine language program. The disassembler takes the machine code and converts it to mnemonic codes that are easier for us to understand.

Before we start, we need to know the hex number that represents the address, or storage location, of the beginning of the machine language program. You can use the HEX/DEC CONVERT program that is on the SUPPORT DISK to convert the decimal number 828 (the first location of the cassette buffer where our machine language program starts) to its hexadecimal equivalent of \$33C. Now let's disassemble the machine code starting at memory location \$33C. To disassemble using SUPERMON, type the following:

```
.D 033C [RETURN]
```

A zero is put in front of '33C' because the monitor requires that hex numbers be written as two-character or four-character numbers. For example, the hex number '8' would be written '08', and '9E1' would be written as '09E1'. When you press [RETURN] you should get one full screen page of disassembled machine code that looks like this:

```
.,033C A0 00      LDY #$00
.,033E A9 04      LDA #$04
.,0340 84 8C      STY $8C
.,0342 85 8D      STA $8D
.,0344 A9 DB      LDA #$DB
.,0346 84 8E      STY $8E
.,0348 85 8F      STA $8F
.,034A B1 8C      LDA ($8C),Y
.,034C 49 80      EOR #$80
.,034E 91 8C      STA ($8C),Y
.,0350 AD 86 02    LDA #$01
.,0353 91 8E      STA ($8E),Y
.,0355 C8         INY
.,0356 C0 00      CPY #$00
.,0358 D0 F0      BNE $034A
.,035A E6 8D      INC $8D
.,035C E6 8F      INC $8F
.,035E A9 08      LDA #$08
.,0360 C5 8D      CMP $8D
.,0362 D0 E6      BNE $034A
.,0364 60         RTS
.,0365 00         BRK
```

The '#' symbol indicates that the hex number that follows is a numerical value to be stored, not an address number. A number in the right column that is not preceded by '#' represents a location address. If you don't see the above listing, but see random numbers and a lot of question marks, it's probably because you did one of two things. The memory locations from \$033C to \$0364 where the machine language program resides are also used by the C-64 as a cassette buffer. If you used the cassette drive after you entered the machine code program, then the code would have been overwritten by tape data. Also, turning the computer off, or resetting it, would have cleared the program from the memory locations of the cassette buffer. If you lose your program, load the BASIC program containing the DATA statements you saved earlier, and run it to put the machine language program back into the cassette buffer.

To understand what the displayed disassembly means, let's examine the disassembled code starting at the top left. The first 4-digit number hex number, \$033C, is the first of the memory locations where our machine language program is stored. The next two numbers are the actual values stored in locations \$033C and \$033D. The disassembler has translated these two numbers into a mnemonic syntax that is easier for us to understand. For example, the 'A0' is translated into the mnemonic LDY or, in long form, LoaD register Y. The '00' is the value we want to load into the temporary storage location called the 'Y' register.

To make a change in the machine code, cursor up and place the cursor on the first '0' of '00' on the first line, and type '88' over the '00'. You will notice that the number has been changed, but the mnemonic in the right column has not. To enter your change, press [RETURN]. Now the LDY #\$00 is replaced with LDY #\$88.

*Altering The
Machine Code:*

The disassembler takes machine code and converts it to symbolic codes, or mnemonics, which are easier for us to understand. The assembler does the opposite. It allows us to enter symbolic (assembly) code instructions which are translated by the assembler to true machine code numbers.

Assembly

Now let's enter a program using the assembler built into the machine language monitor. First erase the program that is currently in memory. You may do this by clearing memory from locations \$033C to \$0364 using the method below. Before you type in the following command, press [CRSR DN] once and hit [RETURN] to get back the decimal.

```
.F 033C 0364 00
```


This command fills locations \$033C to \$0364 with the value '00'. If we get another disassembly display starting at \$033C we would see only '00's and the mnemonic 'BRK'. Try this by typing 'D 033C' beside the decimal (.) on the screen and then hit [RETURN].

Now you can begin to write your own machine language programs. First clear the screen by pressing [SHIFT] [CLR/HOME]. Hit [RETURN] a couple of times so that the cursor is next to a decimal. This decimal lets you know that you are in the machine language monitor, not in BASIC.

To start entering machine code use the 'A' command (for Assembler) like this:

```
.A 033C LDY #$00 [RETURN]
.A 033E □
```

When you type this example and press [RETURN] on the first line, the symbolic code LDY is converted by the assembler to the proper hex numbers and placed in memory at location \$33C. Then the next memory location is printed to the screen, your cursor is placed in the proper place, and the computer is ready for you to enter your next command line. From now on, the assembler will provide the 'A' and the location number for you. All you have to enter is the assembly code itself. Let's finish the program by typing the following assembly code statements next to the 'A' prompts:

```
.A 033E LDA #$20
.A 0340 STA $0400,Y
.A 0343 INY
.A 0344 CPY #$A0
.A 0346 BNE $0340
.A 0348 RTS
.A 0349 [RETURN]
```

Pressing [RETURN] on line 0349 took you out of the assembler. To see what the hex version of the machine code for your assembly language program looks like, display a disassembly by typing 'D 033C'. Your screen should now show this:

```
..033C A0 00            LDY #$00
..033E A9 20            LDA #$20
..0340 99 00 04        STA $0400,Y
..0343 C8              INY
..0344 C0 A1            CPY #$A0
```

.,0346 D0 F8	BNE \$0340
.,0348 60	RTS
.,0349 00	BRK
.,034A 00	BRK
etc.	

On the left side are the memory locations. Next to the location numbers are the machine code instructions expressed in hex. On the right side are the same instructions in assembly code.

Let's go back to BASIC and execute this machine language program. To exit most machine language monitors, you type 'X', for eXit to the right of the decimal and press [RETURN]. You can tell when you are out of the monitor—you don't get '.' printed to the screen after you press [RETURN].

Clear the screen and enter some text on the top six to eight lines. Cursor down a few lines and type SYS 828 and press [RETURN]. If the program was entered correctly, the top four lines of the screen should have gone blank in a fraction of a second.

We now have a simple, but fast machine language program that may be used with a BASIC program. Frequently, programmers tie machine language programs to the end of BASIC programs. The main part of the program is written in BASIC because it's an easy language to use. In order to speed up parts of the program that run too slowly in BASIC, a machine language program is added to the BASIC program. But how do you tie the machine language program to your BASIC program?

You may convert the machine code program to a series of DATA statements that are then appended to the main BASIC program. If you have the SUPPORT DISK, a program called DATA WRITER does the tedious job of converting machine code into DATA statements for you. For a complete description of this program, see Chapter 23.

After the machine code has been placed as DATA statements in the main program, you can use a simple FOR/NEXT loop to POKE each value into memory. The first example at the beginning of this section used such a DATA statement "loader."

Another way to tie BASIC and machine language together is to place the machine code at the end of the BASIC program. Figure 17-1 is a diagrammatic representation of a machine language program tied directly to the end of a BASIC program.

COMBINING MACHINE LANGUAGE AND BASIC PROGRAMS

Writing DATA Statements

Appending Machine Code to BASIC Programs

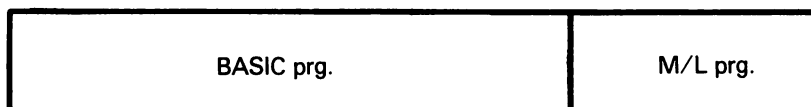


FIGURE 17-1. A machine language program tied to the end of a BASIC program.

The machine language program that is to be tied to a BASIC program can be temporarily stored in a safe place in memory so that the BASIC program is not overwritten by machine code. Normally, a BASIC program will load into the Commodore 64 starting at hex location \$0801, and can continue up as far as memory location \$9FFF. Memory space is also available in other locations of the C-64. For example, the cassette buffer is a small, convenient area for storing short machine language programs like the one we wrote earlier. The cassette buffer has 194 bytes of RAM available, as long as we don't use the cassette drive. Another safe place to put machine code is in memory location \$C000 to \$CFFF. This block of memory gives us 4096 bytes (4K) of RAM memory for our machine language program. (The program POWER PLUS that is provided on the SUPPORT DISK loads into this "hidden" RAM area.)

Suppose we have a BASIC program in the computer and a machine language program in the cassette buffer. First, we must transfer the machine language program to the end of the BASIC program. Once the machine language program is appended to the end of the BASIC program, we must let the computer know that it is to consider the machine language program as part of the BASIC program and execute it. In other words, we must trick the computer into thinking that the BASIC program extends beyond the actual BASIC program to include the machine code.

Let's first type in a short BASIC program to which we will tie the machine language program that we entered earlier. Type in the following program exactly, including all spaces between characters:

```
NEW [RETURN]

10 PRINT "[CLR]";
20 FOR A=1 TO 8
30 PRINT "THIS IS A TEST"
40 NEXT A
```

So that our BASIC program will run the machine language program that will be tied to it, we must include a SYStem call in the BASIC program. Any time you add characters to your BASIC program, it will get bigger; if you delete characters, it will get smaller. The machine code that is tied to the end of this BASIC program will “float” from one location to another as the BASIC program gets larger or smaller. This means that the beginning address of the machine language portion of the program will change as it “floats” with the BASIC program. Therefore, any SYStem call that we might have inserted into the BASIC program to start the machine language program will no longer refer to the correct address for the beginning of the machine language program.

One way to avoid having to keep track of where the machine language portion starts is to use a program instruction that automatically keeps track of where the machine code resides, even after changes are made to the BASIC program.

When a BASIC program is in the computer, the beginning and ending locations of the program are stored by the computer in memory locations 43 through 46. We say that these locations store the *pointers* for the beginning and end of BASIC programs. Remember, the ending address of the BASIC program is the beginning address of the machine language program. You can find the beginning of BASIC by PEEKing locations 43 and 44, and the end by PEEKing 45 and 46.

Each memory location of the C-64 can store only one byte of information, so only numbers up to 255 (the number 255 is 11111111 in the binary system) can be stored in a single location. Consequently, we must use two bytes, or two memory locations, to store numbers greater than 255. Since the ending address of the BASIC program will almost certainly be a number larger than 255, it will be stored as two bytes of information in two locations: 45 (\$2D) and 46 (\$2E). PEEKing these two locations gives us two numbers, in *low byte/high byte* format, which indicate the ending address of BASIC programs. The low byte is stored in location 45, and the high byte in location 46 instead of the other way around, as you might expect. For example, if the two numbers that we get from PEEKing locations 45 and 46 are '50' and '08', we must reverse their order; the 16-bit number is \$0850 rather than \$5008. When we know that \$0850 is the end of the BASIC program, we can transfer the machine language program to this location so that it will be attached to the end of the BASIC program.

Before we transfer our machine language program, let's insert an expression into our program that will calculate the end of BASIC (the

The SYStem call

BASIC Pointers Defined

Low Byte/High Byte

Finding The End of BASIC

beginning of the machine language program) automatically. Add this line of BASIC to the BASIC program that you have typed in:

```
50 SYS( (PEEK(45)+PEEK(46)*256)-X)
```

This line of BASIC takes the decimal value of the high byte at memory location 46 (the high byte for the end of the entire program) and multiplies it by 256, and then adds this to the decimal value of the low byte at memory location 45 (the low byte for the end of the entire program). The decimal value *X*, the exact length of the machine language program that was appended to the end of the BASIC program, is then subtracted to give the end of the BASIC portion of the program, regardless of how the length of the program may change.

To find the end of BASIC, break to your machine language monitor by pressing [*] [RETURN] and list the Memory from hex locations \$2B to \$2E (43 through 46) by typing this command:

```
.M 002B 002E
```

You should have a display like this on your screen.

```
.002B 01 08 56 08 56 08 56 08
```

The first two hex numbers after the '002B' should be '01' and '08'. To properly interpret these numbers, we must flip the values to give us '0801'. This is the beginning location of the BASIC RAM memory. When you load a BASIC program into the computer, it will start loading at location \$0801. The next two hex numbers are the values for the ending of our BASIC program. If you typed the program exactly as it is listed in the example above, these two values should be \$56 and \$08. When flipped, these values form the 16-bit hex number \$0856. This is the ending address of our BASIC program. If you have a hex number slightly different than this, write down whatever hex number you have. Figure 17-2 shows how the end of BASIC value points to the end of a BASIC program.

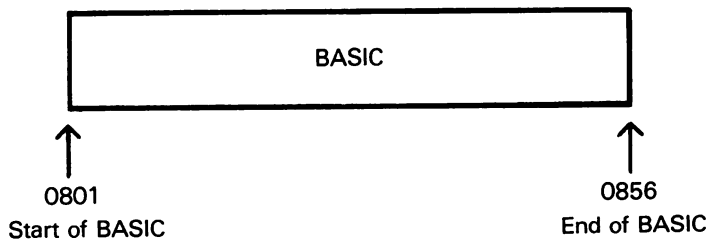


FIGURE 17-2. The end of the BASIC program.

Attaching Programs

Now we must append our machine language program that is still located in the cassette buffer to the end of our BASIC program by using the machine language monitor's Transfer command. First we must find out how long our machine language program is. To accomplish this, disassemble the machine code starting at location \$033C by using this command:

```
.D 033C
```

You will see a display that looks like this:

```
.033C A0 00      LDY #$00
.033E A9 20      LDA #$20
.0340 99 00 04    STA $0400,Y
.0343 C8         INY
.0344 C0 A0      CPY #$A0
.0346 D0 F8      BNE $0340
.0348 60         RTS
.0349 00         BRK
.034A 00         BRK
etc.
```

Note that the program ends at location \$0348. Cursor down one line and hit [RETURN] in order to position the cursor on a new line next to a decimal. To transfer this code, type:

```
.T 033C 0348 0856 (or the number you got when you used
                  the M command)
```

In this transfer command, \$033C and \$0348 are the beginning and ending addresses of our machine language program. The number \$0856 (or the number you wrote down) is the address to which we want to transfer the machine code. This is the location that marks the end of BASIC. To see where the appended machine code now ends, we must disassemble starting with the hex number that we got from locations #2D and \$2E, like this:

```
.D 0856 (or your number)
```

```
.0856 A0 00      LDY #$00
.0858 A9 20      LDA #$20
.085A 99 00 04    STA $0400,Y
.085D C8         INY
.085E C0 A0      CPY #$A0
.0860 D0 F8      BNE $0340
.0862 60         RTS
.0863 00         BRK
```

The appended machine code now ends after location \$0862. Count the machine code hex numbers to the right of the memory locations \$0856 through \$0862. There are 13 of them. This means that the total length of our machine language program is 13 bytes; keep this in mind for later. Figure 17-3 shows how the machine language program is now tied to the end of BASIC. The end of BASIC value (stored in memory locations 45 (\$2D) and 46 (\$2E)) is \$0856. You can demonstrate this by PEEKing these locations again.

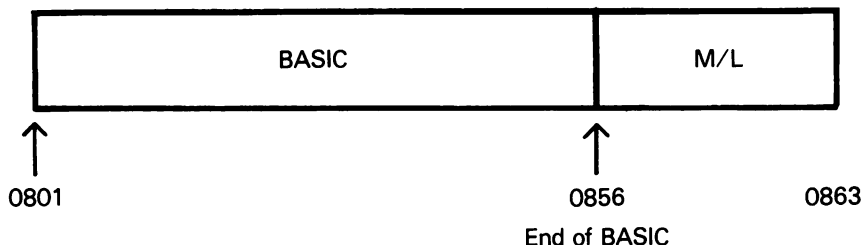


FIGURE 17-3. The machine language program tied to a BASIC program.

To tie this machine code to the BASIC program, we must readjust memory locations \$2D and \$2E to point to one byte past the end of our machine language program (\$0863). Use the Memory dump command to list the memory from hex locations \$2B to \$2E like this:

```
.M 002B 002E
```

You should have a display like this on your screen:

```
.002B 01 08 56 08 56 08 56 08
```

Now change the first \$56 (bold) to \$63 in location \$2D so that it looks like this:

```
.002B 01 08 63 08 56 08 56 08
```

Don't forget to press [RETURN] to enter your correction to the computer.

By changing this memory location, we have adjusted the end of BASIC to include the machine language program. Figure 17-4 shows how the end of BASIC value in locations \$2D and \$2E now points to the end of the machine language program.

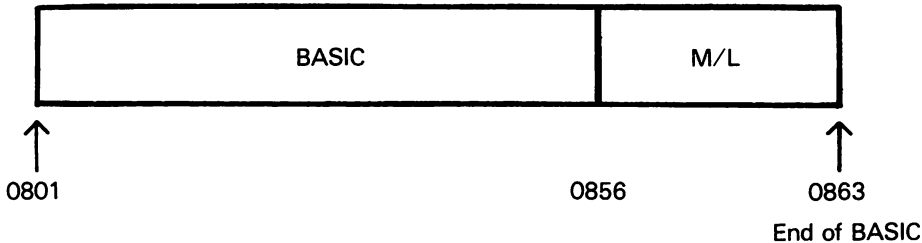


FIGURE 17-4. The end of BASIC is now at the end of the machine language program.

Before we can run our new program, we must exit to BASIC by using the 'X' command. Now enter this BASIC line:

```
5 X= 13 [RETURN]
```

The variable **X** is now set to the length of the appended machine language program (13 bytes). Before you run this program, save it on a disk or tape to ensure that you have a copy in case it "locks up" when you run it. After you have saved the program using a normal BASIC SAVE command, type RUN and [RETURN]. If you typed everything correctly, you should have eight lines printed to the screen, and then the top four lines are instantly erased.

Since we have written line 50 to automatically adjust where the machine language program is residing in memory, you may add BASIC lines to this program, or delete them, without having to recalculate the SYStem call number that starts the machine language component of the program. As the BASIC program grows or shrinks, line 50 will adjust the SYStem call to the proper location by looking at locations 45 (\$2D) and 46 (\$2E). You can demonstrate this by typing a new basic line:

```
35 PRINT "WHAT A TEST!"
```

Run the program again and show that the computer will still find and run the entire program.

There are many exciting applications that can be achieved only with programs written in machine language. This chapter has introduced you to the power of machine language programming. For more information on this subject, refer to one of the many books on programming the 6502 microprocessor.

THE NEXT STEP

1891

1891

1891

1891

1891

1891

1891

1891

1891

1891

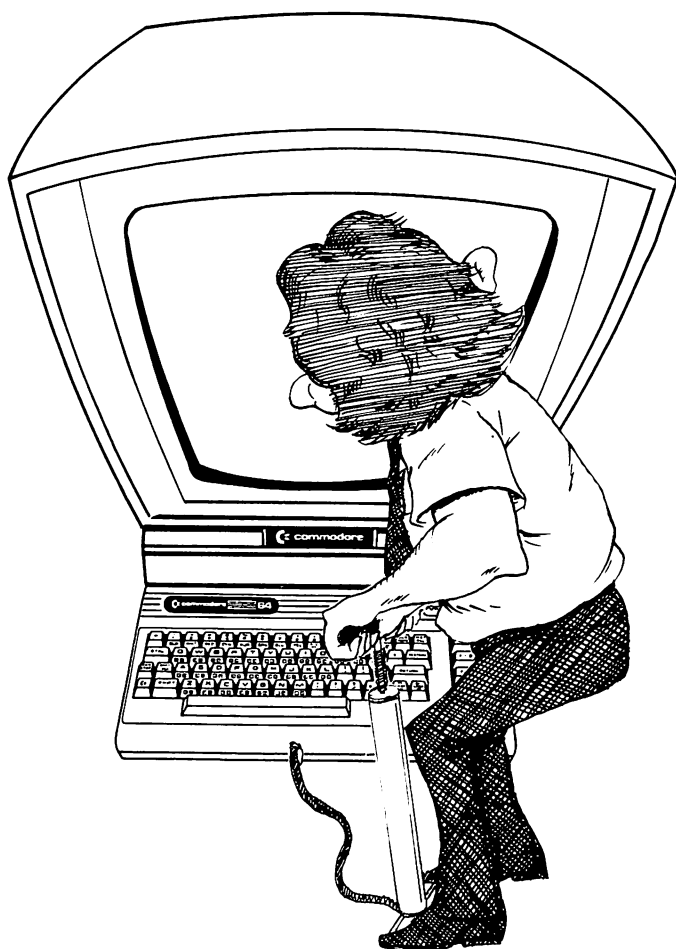
1891

1891

1891

1891

1891



PART 5

POWER PLUS: A Utility Program

Introduction To POWER PLUS

POWER PLUS is a package of three different utility programs designed to make using your computer easier. POWER PLUS provides over 40 commands that add more power to the C-64. Because it loads into a non-BASIC area of memory in the computer, POWER PLUS doesn't use up any BASIC Memory. As long as your computer is on, you have access to all of this utility program's power. Since POWER PLUS is written in machine language, it performs its tasks very quickly.

The three utility programs that make up POWER PLUS are SUPERDOS, SUPERBASIC, and SUPERMON.

POWER PLUS UTILITIES

SUPERDOS provides many powerful commands that make it much easier for you to operate your disk drive. It also includes a number of very convenient commands for computer and printer control. The SUPERDOS utility has been written with a syntax similar to that used in DOS 5.1 (also called C-64 WEDGE), so that users familiar with this syntax need not learn a new one.

SUPERDOS

SUPERBASIC is an editing and debugging tool for BASIC programs. It adds six new commands to the BASIC vocabulary. They will save you many hours of time and frustration when you're working with BASIC programs.

SUPERBASIC

SUPERMON* is a machine language monitor utility that gives you the ability to work with your computer in its own language, machine language. SUPERMON has all the important features that costly machine language monitors provide.

SUPERMON

*This program was modified from Jim Butterfield's SUPERMON 64.

Load POWER PLUS as you would normally load any BASIC program:

```
LOAD "POWER PLUS",8    or
LOAD "POW*",8          or
LF "P*",8
```

LOADING AND RUNNING POWER PLUS

POWER PLUS loads into the BASIC area of the computer's memory even though it's a machine language program. Use the LIST command to see that it is in BASIC memory. Now type RUN. When run, POWER PLUS moves into an area of memory outside of the BASIC area. Type LIST and you will see that it's no longer in BASIC memory.

19

SUPERDOS

SUPERDOS is a disk utility that makes it much easier for you to send commands to your disk drive. SUPERDOS commands are shorter than the commands you must use without it (see Chapter 8). If you don't have the SUPPORT DISK available with this book, you can use a program called C-64 WEDGE, which is on the TEST/DEMO disk included by Commodore with the disk drive.

The SUPERDOS commands described below must be entered with the cursor flush with the left side of the screen. No blank spaces are allowed in front of the commands. If you type a command incorrectly, you will get a 'SYNTAX ERROR' message.

The general commands that SUPERDOS gives you are all entered with '*' as the first character, as shown below.

GENERAL COMMANDS

Change Colors

*C
*C1
*C2

These commands change the border, background, and character colors of the screen to predetermined values. You have a total of three preset combinations to choose from: *C, *C1, and *C2. It is possible for you to alter the SUPERDOS program so that your own color choices rather than the predetermined combinations will result from these commands. Follow the instructions in the box to reprogram the color values of the CHANGE COLORS commands.

REPROGRAMMING THE COLORS

Find three border, background, and character color combinations that you like. (If you have the SUPPORT DISK that came with this book, load and run COLOR POKES to view and select all the color combinations that are possible, and also to record the appropriate POKE numbers. Remember, POWER PLUS is in a memory area other than BASIC, so loading COLOR POKES will not overwrite it. If you don't have the SUPPORT DISK, you can use the chart in Appendix D to find the POKE numbers that correspond to the colors you want. For example, suppose you wish *C to give you a black border, a white background, and blue characters. Using COLOR POKES or referring to the chart in Appendix D, you'll see that the POKE value for black is 0, white is 1, and blue is 6. Write down the color POKES for your favorite colors. Now follow these steps:

1. Load POWER PLUS into the computer, but don't run it.
2. Perform the three POKES for the three colors you want to obtain with the *C command, as shown:

```
POKE 7437,0
POKE 7434,1
POKE 7431,6
```

You have now reprogrammed the command *C to give you a black border, a white background, and blue characters. Now carry out this same procedure for the *C1 and *C2 commands, selecting two other sets of border, background, and character colors.

*C	POKE 7437,X	(border)
	POKE 7434,X	(background)
	POKE 7431,X	(character)
*C1	POKE 7452,X	(border)

	POKE 7449,X	(background)
	POKE 7446,X	(character)
*C2	POKE 7467,X	(border)
	POKE 7464,X	(background)
	POKE 7461,X	(character)

3. In order to make these changes a permanent part of your POWER PLUS utility program, you must save the altered program, on tape or disk before you RUN it. Call it 'MY POWER PLUS'. Each time you load and run 'MY POWER PLUS', *C, *C1, and *C2 will give you your chosen color combinations.

*B

The *B command turns on SUPERBASIC, a utility that makes it easier for you to write and debug BASIC programs. Chapter 20 provides a detailed list of commands for SUPERBASIC.

Turn on SUPERBASIC

*

Pressing [*] followed by [RETURN] will turn on SUPERMON. You'll see 'SUPERMON' printed on the screen.

You'll find a full description of SUPERMON in Chapter 21. If you're not familiar with what a machine language monitor is and does, don't worry. As you become more familiar with the C-64, the uses for a machine language monitor will be more evident.

To get out of SUPERMON, press [X] and [RETURN].

Turn on SUPERMON

*D

This command will DUMP everything that is on the screen to your printer. That is, the printer outputs the entire contents of the screen in the same format as the screen display. The command defaults to device number 4, but may be changed by entering a number after the *D as in this example:

*D5

*D5 will dump the screen to the printer that has a device number of 5.

You can use the dump screen command in program control by using the following SYSTem call whenever you want one of your

Dump Screen to Printer

programs to stop in order to dump the screen to the printer and then to continue:

```
SYS 53059
```

NOTE: This SYStem call will work only with a printer that has a device number of 4.

*U

If you have accidentally NEWed a BASIC program, you will no longer be able to LIST it or RUN it. The program seems to have been erased. Actually, the program is still in the memory of the computer. It seems to have been erased because the computer no longer knows where the program is. You can get your program back by using the Un-new command.

Try the UN-NEW command by loading or typing in a short program and then typing NEW. If you type LIST, you'll see that the program is no longer available for listing. Now press [*][U] and [RETURN] and then type LIST. The program is back!

If you have installed a reset switch by following the instructions on page 262, you can use this switch to reset the computer. The program or programs that were in the memory of the computer are still there, but the computer doesn't know where they are. After the reset button is pressed, we can turn POWER PLUS back on by sending the command SYS 52426. Now that POWER PLUS is on, you can use the *U (Un-new) command to bring back the BASIC program that was in your computer.

*K

This KILLS, or turns off, POWER PLUS. There are some programs that will need the same memory space of the computer that POWER PLUS uses. To see if a program will occupy the same space as POWER PLUS, use the DISK PAC or CATALOGER program described in Chapter 23. POWER PLUS resides in memory locations 49152 and 53247 (\$C000 to \$CFFF). When you want to run a program that uses this same area, turn off POWER PLUS before you load the other program.

SYS 52426 will bring back SUPERDOS after it has been killed, or turned off.

Let's try some LOAD commands. First try this command using the name of a program (indicated by <fn>, which stands for "file name") on one of your disks:

```
/<fn>
```

UN-NEW BASIC Program

Kill POWER PLUS

LOAD OPTIONS

This command will load a BASIC program into the BASIC area of the computer's memory. It replaces the LOAD"<fn>,"8 command, which is used without SUPERDOS. Note that this SUPERDOS command eliminates the need for the quotation marks and the comma 8.

Now use the command that loads and then automatically runs your program once the load has been completed:

↑<fn>

The following command will load a program that was designed to load into an area of memory other than BASIC, to the area of memory it was saved from. This is called an *absolute load*.

%<fn>

To save a program with SUPERDOS, use this command:

←<fn>

To replace a program that is on the diskette with one that's in the memory of the computer, under the same name, use this command:

←@0:<fn>

Each of the following commands can be typed in using either the prefix '@' or '>'. Since pressing [@] does not require the SHIFT key, it may be easier for you to type than [>] is.

@\$ List disk directory; retain BASIC program.

This command permits you to view the directory of any diskette without losing the BASIC program in memory. In contrast, recall that if you use 'LOAD"\$",8', the directory is loaded into the computer as a BASIC program that will overwrite any BASIC program in the memory of the computer.

@\$XYZ* List all files starting with 'XYZ'

This version of the directory command allows you to list to the screen only files that start with the letters 'XYZ'. If you want to list all the programs that begin with the letter 'X', use:

@SX* List all files starting with 'X'.

SAVE OPTIONS Save

Save and Replace

SENDING COMMANDS TO THE DISK

Directory Commands

Listing All File Names

*Listing Selected
File Names*

Here are two more ways to use @\$:

@\$???? List all files with four-letter file names.

@\$X??? List all files with four-letter file names starting with 'X'.

You can see that with combinations of pattern matching (*) and wild cards (?), you can request precisely the kind of file names you wish to see.

The variations of the DIRECTORY command listed below allow you to list files according to their file type rather than their name.

*Listing a Selected
File Type*

@\$*=P List all Program files.

@\$*=S List all Sequential files.

@\$*=R List all Relative files.

@\$*=U List all User files.

Pressing [SPACE] sharply will freeze, or stop, the directory listing. Pressed again, [SPACE] will continue the listing. Pressing [STOP] will also halt the listing, but without the restart option.

Freezing the Listing

@I Initialize disk drive.

If you are having problems with your disk drive, use the INITIALIZE command to read the BAM of the diskette you wish to access. This will ensure that the disk drive knows which diskette is currently in the drive and where the information on the diskette can be found.

**Initializing the
Disk Drive**

@U; Reset disk drive.

The RESET command resets the memory chips of the drive to the power-up condition. You can use this command in place of turning the drive's power switch off and on.

**Resetting the
Disk Drive**

@ Get disk status.

When you try a disk command that cannot be carried out, the red light of the disk drive will start blinking, letting you know that an error has occurred. Without SUPERDOS, you must type in and run a short program (described on page 77) to discover what type of error has occurred. The '@' is much easier to use.

**The Disk Status
Command**

@#<dn> Change disk device number that POWER PLUS addresses.

The device number <dn> is any number from 8 to 12. POWER PLUS defaults to device #8.

This command is used when you have two 1541 disk drives connected to the C-64. It changes the device number that POWER PLUS sends its commands to. You can use a device number from 8 to 12 for the second drive. For example, if you want POWER PLUS to send its disk commands to disk drive 9, use the following command:

@#9

@N0:<n>, ID New or Format the disk.

On page 76 we discussed the NEW (or FORMAT) command without SUPERDOS. Remember that this command will erase all the information that was on the disk. Don't use periods, colons, or semicolons in the disk name <n>. Here's an example of the NEW command in its SUPERDOS short-form:

@N0:MY PROGRAM, VG

@S0:<fn> Scratch file on disk.

This command will remove the names of one or more files from the disk directory. When we SCRATCH a file, the BAM will no longer have a record of where the program is, and the space that the file occupied will be made available for other files. However, if you accidentally scratch a file, you may retrieve it by using a program called DISK PAC, which is described in Chapter 23.

@S0:XYZ* Scratch all files that start with XYZ

You can use pattern matching as shown above to scratch all files starting with 'XYZ'. Keep in mind that it is rare to scratch all files starting with certain letters. In order to avoid accidents, you should probably not use pattern matching when using the SCRATCH command.

@S0:*=P Scratch all Program files.

@S0:*=S Scratch all Sequential files.

@S0:*=R Scratch all Relative files.

@S0:*=U Scratch all User files.

Addressing Other Disk Drives

Writing to and Erasing from the Disk

Formatting a Disk

Scratching A File

@R0: <new fn>=<old fn> Rename a file on disk.

Renaming a File

This command lets you change the name of any file. For example, if you want to change the name of a file from 'HATS' to 'HITS', you can do it this way:

@R0:HITS=HATS

@C0:<fn2>=0:<fn1> Copy file 1, naming it file 2.

Copying a File

The COPY command will duplicate file 1, giving it the name you specify as file 2.

@C0:<fn3>=0:<fn1>,0:<fn2> Concatenate file 1 and file 2 to make file 3.

Linking Data Files

You can use this command to put two data files together into one file. Use this command only for data files; it will not merge programs successfully. If you want to concatenate a file called PART ONE with another called PART TWO to form a third file called PART A, you type the command in this way:

@C0:PART A=0:PART ONE,0:PART TWO

@V Validate extra space on disk.

Validating the Diskette

The validate command will rewrite the diskette's BAM and will get rid of files that have not been properly saved.

***C, *C1, *C2** Change screen colors.

***B** Turn on SUPERBASIC.

***** Break to SUPERMON.

***D<H>** Dump screen to printer.
<H> = printer device number.
Default is device # 4.

***U** Un-new a BASIC program.

***K** Kill all POWER PLUS utility routines.

/<fn> Load a particular BASIC program.

/XYZ* Load the first program that starts with XYZ.

SUPERDOS QUICK REFERENCE General Commands

Ways of Loading Programs

/??? Load all programs with three-letter titles.

↑ <fn> Load and run a BASIC program.

%<fn> Load a program to an absolute address.

← <fn> Save a BASIC program.

← @0:<fn> Replace the program on disk with the program in the computer under the old program's name.

Ways of Saving Programs

SYS 52426 Turn on POWER PLUS after a reset or the *K command.

System Calls

SYS 53059 Screen dump command for BASIC programs.

SUPERBASIC is the second part of the three-part utility package POWER PLUS. SUPERBASIC adds six new BASIC commands to the C-64 to help you write and edit BASIC programs. POWER PLUS commands can be used only in direct mode, not within BASIC programs.

These new BASIC commands become part of BASIC and may be abbreviated like most other BASIC words by shifting the second letter of the command. For example, AUTO may be typed as [A] shifted [U], and DELETE may be written as [D] shifted [E].

AUTO provides you with automatic line numbers whenever you're typing BASIC program lines. It keeps track of what line number you're on so that when you hit [RETURN], the next line number will be displayed below your current line, automatically incremented by an interval you've chosen. Your cursor will be in position for the entry of your next line of BASIC. The format for this command is as follows:

AUTO

```
AUTO<interval>
```

To show how this command works, first type NEW to remove the BASIC program in the computer, then type the following and press [RETURN]:

```
AUTO 10
```

The AUTO command is now on. To test it, type the following line:

```
100 REM THIS IS A TEST
```


You will notice that when you pressed [RETURN], the next line number was printed to the screen, and your cursor is now in the right place to continue writing your program. Type PRINT "SUPERBASIC SAVES YOU TIME" [RETURN] and you should have this display:

```
100 REM THIS IS A TEST
110 PRINT "SUPERBASIC SAVES YOU TIME"
120 □
```

To exit AUTO mode temporarily, simply press [RETURN] with only the line number present and the cursor will drop to the next line without giving you a new line number. At this point the AUTO feature is still on; if you enter a new line of BASIC with a line number, or edit an old line, the next line number incremented by your chosen interval will be provided by the AUTO mode. Try hitting [RETURN] on line 120 and type this line:

```
200 PRINT "PROGRAMMING IS EASIER"
```

When you used line 200, the AUTO feature gave you the next line after 200, incremented by 10.

```
100 REM THIS IS A TEST
110 PRINT "SUPERBASIC SAVES YOU TIME"
120
200 PRINT "PROGRAMMING IS EASIER"
210 □
```

To turn the AUTO command off, press [RETURN] on the line you're on without typing anything, and then type AUTO by itself with no line number interval and hit [RETURN].

```
100 REM THIS IS A TEST
110 PRINT "SUPERBASIC SAVES YOU TIME"
120
200 PRINT "PROGRAMMING IS EASIER"
210
AUTO
□
```

Line numbers may not exceed 63999. If you use a large increment with a program that is very long, you may run out of line numbers. For this reason, you should limit your increment sizes.

The DELETE command is used to remove a range of lines from a BASIC program. This is particularly useful when you wish to delete many lines from a long BASIC program. The syntax for DELETE is the same as for the LIST command:

```
DELETE<starting line no.>-<ending line no.>
```

For example, if you wish to delete all lines from 100 through 200 from our sample program, type the following:

```
DELETE 100-200
```

If you wish to delete all line numbers from 400 to the end of the BASIC program, type this:

```
DELETE 400-
```

To delete the first line numbers up to line 400, type the command this way:

```
DELETE -400
```

You'll notice how the '-' is used to define the range for the DELETE command. If you type DELETE without any number range, no deletions will occur. This prevents the complete loss of a BASIC program by accident.

The NUMBER command will renumber the entire BASIC program in the memory of your computer. It will not only change the line numbers, but also all line number references that occur in GOTO, GOSUB, IF THEN, ON GOTO, ON GOSUB, and RUN statements. To use the NUMBER command, you must give both the starting line number and the line number interval. A comma must be placed between these two numbers. The syntax of the NUMBER command is as follows:

```
NUMBER<start line no.>,<line no. interval>
```

To see how the NUMBER command works, type in the following program:

```
1 REM THIS IS A TEST
10 PRINT "HELLO"
67 GOTO 200
```

DELETE

NUMBER

```

130 END
200 IF A=10 THEN 130
210 GOSUB 1000
300 END
1000 PRINT "[SPACE]THERE"
1001 RETURN

```

Now type this:

```
NUMBER 100,10
```

You will notice that some numbers are printed to the screen. These are the new line numbers that will be used in GOTO and GOSUB statements to maintain accurate references to these renumbered lines in the program. Now, if you type LIST the program will look like this:

```

100 REM THIS IS A TEST
110 PRINT "HELLO"
120 GOTO 140
130 END
140 IF A=10 THEN 130
150 GOSUB 170
160 END
170 PRINT "[SPACE]THERE"
180 RETURN

```

If there are any GOTO or GOSUB references to BASIC line numbers that don't exist in the program, these line references will appear as 65535. To see how this works, cursor up to line 120 and change the 140 in that line to 500 by typing over 140. When you have done this, press [RETURN] to enter the new line. Now type the reNUMBER command like this:

```
NUMBER 10,10
```

You will see the numbers 65535, 40, and 80 printed to the screen as the reNUMBER command is being executed. The 65535 lets you know there's a line reference in the BASIC program to a line that doesn't exist.

This feature of the reNUMBER command is an excellent debugging tool. Any time you make a reference to a line that doesn't exist in your BASIC program, the program will not run properly. You can use the reNUMBER command to detect the error before you run the program.

If you would like to have the GOTO and GOSUB reference numbers printed to your printer instead of your screen, do the following:

```
OPEN 4,4
CMD 4
NUMBER 1000,10
```

This will renumber your BASIC program starting with line number 1000 with a line interval of 10, and all line references will be printed to your printer. To get control back to the screen do the following:

```
PRINT#4
CLOSE 4
```

See Chapter 13 for a discussion of commands sent to the printer.

The FIND command will search through your BASIC program and display all lines that contain the specified BASIC keyword or string. You may specify the range of line numbers that you want to search for in the same way that you would for a LIST or DELETE command. The syntax for FIND for searching for BASIC keywords is as follows:

FIND

```
FIND:<BASIC word>:,<start line no.>-
<end line no.>
```

If you want to search the whole program, the syntax is:

```
FIND:<BASIC word>:
```

If you use FIND on large programs, the lines may fill the screen and scroll off the top. To slow this down, hold down [CTRL]. Press [STOP] to stop the listing entirely.

There are two ways to use the FIND command. You can use it to search for BASIC words or expressions, like the BASIC words FOR, NEXT, or GOTO, or to find expressions such as X=12. You can also use the FIND command to search for quoted strings like "A" or "THIS IS A TEST". The syntax for using FIND to search for strings is as follows:

```
FIND "<string>",<start line no.>-
<end line no.>
```

To see the difference between the two FIND formats, type NEW and then enter the following program:

```
10 POKE 53280,1:POKE 53281,1
20 PRINT "[CLR][4 CRSR DN]THIS IS A TEST"
30 FOR A=1 TO 20
40 PRINT "HELLO THERE"
50 NEXT A
60 PRINT "THIS IS ALL FOR NOW"
70 POKE 53280,4
```

Run it to make sure you typed it correctly. First we'll look for all occurrences of the BASIC keyword FOR by typing the following FIND command:

```
FIND :FOR:
```

The result will be this:

```
30 FOR A=1 TO 20
```

You will notice that the word (string) 'FOR' in line 60 is not listed to the screen. 'FOR' in this line is not a BASIC keyword but rather the word 'for' as it is commonly used in English. It is part of a string, because it is inside quotation marks. To find all occurrences of 'for' as part of a string, use the following command:

```
FIND "FOR"
```

To find strings, we use quotes rather than colons to delimit the characters that are being searched for.

One thing to keep in mind is that BASIC keywords are stored in your computer as tokens, or code numbers, not as sequences of letters. This means that to find BASIC words like PRINT or NEXT, you must type out the entire word. If you typed 'FIND:NE:' you would not find the BASIC words NEXT, but rather any expressions that use those two letters by themselves such as AB=NE+20.

Let's try another example. Type the following:

```
FIND :POKE:,-30
```

The following BASIC lines appear on the screen:

```
10 POKE 53280,1:POKE 53281,1
10 POKE 53280,1:POKE 53281,1
```

You'll notice that line 10 was listed twice. This lets you know that there are two POKEs in line 10. Also note that because we defined our range as -30, only lines 10 through 30 were searched. That's why the POKE on line 70 is not listed.

You may search for control or graphic characters by placing them inside quotes like this:

```
FIND "[CLR]"
```

The screen will display:

```
20 PRINT "[CLR][4 CRSR DN]THIS IS A TEST"
```

Line 20 is listed because the control code for CLEAR (the reverse heart) is inside quotes.

Remember that the computer stores strings of characters as numbers. There are some BASIC words that have the same numerical value as certain control or graphic characters. If you get a line listed that doesn't have the BASIC word you were searching for, it probably has a control or graphic character that matches the BASIC keyword token.

The CHANGE command is a very powerful command and can save you a great deal of time when you are changing programs to suit your specific needs. With this command, you can automatically change a specific BASIC word, expression, or string in your program to another BASIC word, expression, or string. The syntax for CHANGE is similar to that of the FIND command, in that you must use quotes or colons as delimiters, and you can define a line number range. In addition, you must specify what BASIC word, expression, or string you want the specified word, expression, or string to be changed to. This is the syntax of the CHANGE command:

CHANGE

```
CHANGE:<word 1>:<word 2>:,
<start line no.>-<end line no.>

CHANGE"<string 1>"<string 2>",<start line
no.>-<end line no.>
```

To see how CHANGE works, type LIST to view the following program:

```
10 POKE 53280,1:POKE 53281,1
20 PRINT "[CLR][4 CRSR DN]THIS IS A TEST"
30 For A=1 to 20
40 PRINT "HELLO THERE"
50 NEXT A
60 PRINT "THIS IS ALL FOR NOW"
70 POKE 53280,4
```

To change all occurrences of 'PRINT' to 'PRINT#4,', do the following:

```
CHANGE:PRINT:PRINT#4,:
```

You'll see this on your screen:

```
20 PRINT#4, "[CLR][4 CRSR DN]THIS IS A TEST"
40 PRINT#4, "HELLO THERE"
60 PRINT#4, "THIS IS ALL FOR NOW"
```

This change makes it easy for you to send output from a running program to a printer instead of the screen. To retype each 'PRINT' to 'PRINT#4,' in a large program would take a long time. With the CHANGE command, you can do it in a few seconds.

Like the FIND command, CHANGE searches for BASIC tokens. It also interprets your keyboard entries as these BASIC word tokens whenever possible. So be careful not to use words that have a BASIC word embedded in them. For example, the word 'FORM' has the BASIC keyword 'FOR' in it. Other examples are 'TTotal' and 'ANDroid'. The CHANGE command will 'tokenize' the BASIC word in these words and the replacement will not work properly.

Now let's change the string on line 40 to 'GOOD-BYE' by doing the following:

```
CHANGE "HELLO THERE"GOOD-BYE"
```

The screen will show:

```
40 PRINT#4, "GOOD-BYE"
```

This command will KILL, or turn off, the SUPERBASIC part of POWER PLUS. You may choose to do this if you know that some other program you wish to load into the computer uses the same

KILL

memory location as SUPERBASIC. You can turn SUPERBASIC back on by using the *B command.

AUTO<interval>

DELETE<start line no.> - <end line no.>

NUMBER<start line no.> - <line no. interval>

FIND:<BASIC word>:, <start line no.> - <end line no.>

FIND"<string >",<start line no.> - <end line no.>

CHANGE:<word 1>:<word 2>:, <start line no.> - <end line no.>

CHANGE"<string 1>"<string 2>",<start line no.> - <end line no.>

KILL Turns off SUPERBASIC.

SUPERBASIC QUICK REFERENCE

SUPERMON is the third utility program that makes up the POWER PLUS utility package. SUPERMON is a machine language monitor, a utility that allows you to study and manipulate the inner workings of the C-64. When you use SUPERMON, you're dealing with the machine language of your computer. If you're not familiar with machine code or hexadecimal numbers, study a book that deals with machine language programming of the 6502 microprocessor. With time, patience, and the SUPERMON utility, you will be able to write programs in machine language that can run more than 100 times faster than those written in BASIC. As you read this chapter, keep in mind that hex numbers are preceded by a dollar sign (\$).

SUPERMON may be used to:

1. save machine language programs,
2. write machine language programs,
3. modify machine language programs, and
4. change the memory registers of the computer without using PEEK and POKE.

USES OF SUPERMON

In this chapter you'll learn the commands for SUPERMON. Turn your computer off and then on, load POWER PLUS, and then run it. If you do this, you'll get the expected results from the examples presented here.

To "break to", or turn on, SUPERMON, press [*] [RETURN]. You'll see the REGISTER DISPLAY and a '.' to the left of the flashing cursor, like this:

```

      PC      SR      AC      XR      YR      SP
.;CEB6      32      00      0B      01      F8

```

- □

If you now press [RETURN], a new '.' will be printed on the next line, showing you that the monitor is on. You can't use normal BASIC commands while in SUPERMON. To demonstrate this, try typing LIST and press [RETURN]. You should now see this:

```

      PC      SR      AC      XR      YR      SP
.;CEB6      32      00      0B      01      F8

```

```

.LI?T

```

. □

The question mark lets you know that you've used a word that is not in the syntax of SUPERMON.

Let's examine some of the operations that can be performed with SUPERMON. A detailed description of how to use SUPERMON is in Chapter 17, which deals specifically with machine language programming.

The simple Assembler is used to enter machine code into the computer using standard 6502 mnemonics (assembly language codes). If you don't know assembly language, the next few paragraphs won't mean much to you, but we suggest that you do the exercises anyway to get a feel for the operation of SUPERMON.

SUPERMON COMMANDS Assembler (A)

To begin, type [A] and [SPACE]. Next type the beginning memory location (in hex) at which you wish to start writing assembly code, and then type the code itself. In the example below, after the first line is entered, the next 'A' and the next memory location are printed to the screen, and SUPERMON is ready to receive your next line of mnemonic code:

```

.A 1000 LDA #$56
.A 1002 □

```

Complete this line as follows:

```

.A 1002 STA $8000,X

```

The screen now shows:

```
.A 1000 LDA #$56
.A 1002 STA $8000,X
.A 1005 □
```

To exit the assembler, simply press [RETURN] after the memory location, without entering any assembly code.

The Disassembler is used to view and change machine code, using the screen editor of your computer. If you type a [D] [SPACE] and then the beginning memory location (in hex) followed by [RETURN], 22 lines of disassembled machine code will be printed to the screen. By using the cursor keys and the screen editor, you can move around and change the hex numbers to alter the contents of those memory locations. After you have changed a line, press [RETURN] to enter the changes. The computer will clear the screen and reprint the disassembly with the changes.

Disassembler (D)

To get a disassembly of the code starting at location \$C000, do this:

```
.D C000 [RETURN]
```

You will see 22 lines displayed on your screen:

```
., C000 A2 00      LDX #$00
., C002 BD D1 C4   LDA $C4D1,X
., C005 F0 06      BEQ $C00D
.,      .         .
.,      .         .
.,      .         .
., C02C 85 7E      STA $7E
```

Since the POWER PLUS utility program starts at \$C000, you are actually looking at the first 22 lines of code of this utility program. Machine language code is on the left, assembly code is on the right.

To list the next 22 memory locations, type a 'D' at the cursor position on the last line, and hit [RETURN]:

```
., C02C 85 7E      STA $7E
., C02E 60          RTS
```

```

., C02F 85 8E      STA $8E
.,      .          .          .
      .          .          .
      .          .          .
., C057 C0 20      CMP #$20
., _

```

You can use the FILL command to fill a block of memory with a particular number. Press [CRSR DN] once to move out of the disassembled lines you have on the screen, and hit [RETURN] to get your '>'. Then type [F], [SPACE], the beginning memory location followed by a [SPACE], the ending location, another [SPACE], and any two-digit hex number. Try this example:

Fill Memory (F)

```

.F 1000 1005 FF [RETURN]

```

This will fill all memory locations from 1000 hex to 1005 hex with the hex value 'FF'. Use the 'D' command to verify that it worked:

```

.D 1000

```

You'll find that the screen shows all lines between \$1000 and \$1015. Lines \$1000 to \$1005 will look like this:

```

., 1000 FF      ???
., 1001 FF      ???
., 1002 FF      ???
., 1003 FF      ???
., 1004 FF      ???
., 1005 FF      ???

```

After you've written a machine code program, you can run that program by using the GO RUN command. For example, if you want to run a program that starts at 1000 hex, you would type the command as follows:

Go Run (G)

```

.G 1000

```

The HUNT MEMORY command may be used in two ways. The first is to search for a sequence of bytes in memory. In the following example we'll search from \$C000 to \$C100 for the occurrence of the bytes 'FD' and '02'.

Hunt Memory (H)

```
.H C000 C100 FD 02
```

The computer lists (in hex) the locations that have the values \$FD and \$02, like this:

```
.H C000 C100 FD 02
C01C C072 C07D
```

The next example shows how to search the memory locations from \$C000 to \$CFFF for the ASCII string 'ABCD'.

```
.H C000 CFFF 'ABCD'
```

A maximum of 32 characters or bytes may be the subject of your search. The closing apostrophe (') should not be used.

To load a machine language program into your computer while in the monitor, use the 'L' or LOAD command as follows:

Load (L)

```
.L "TEST",08
```

If there were a "TEST" program on the disk, this command would load it into the computer from device number 8 to its absolute address. You may load from disk by using device number '08', or from cassette by using '01'. Remember to specify the device number as a two-digit number.

To view and change a large block of memory, use the MEMORY command. Press [M], [SPACE], the beginning and ending locations, and then [RETURN]. You can now use the screen editor to change the memory values. Remember to press [RETURN] after the line has been changed in order to enter your changes into the computer. To view memory from \$C000 to \$C010, do this:

View Memory (M)

```
.M C000 C010
. C000 A2 00 BD D1 C4 F0 06 20
. C008 D2 FF E8 D0 F5 A9 4C 85
. C010 7C A9 2F 85 7D A9 C0 85
```

If the block is larger than 24 rows, the display will scroll off the top of the screen. To stop the scrolling, press [STOP].

The **PRINTER DISASSEMBLER** command will provide you with a printout of disassembled code of a defined area of memory. It works the same way the normal **DISASSEMBLER** works, except that the '.' is suppressed to give you a cleaner-looking listing. To get a printout, do the following in **BASIC** before you enter the monitor (if you're already in the monitor, press [X] to exit the monitor to **BASIC**, and then type in this command in direct mode):

```
OPEN4,4:CMD4
```

If your printer is on, it will print 'READY.' Now, instead of breaking to the monitor using the '*' command, type the following:

```
SYS 50517
```

You won't see the '.' because it is suppressed, but you are now back in **SUPERMON**. You may use the 'P' command like the 'D' command, except that in the **PRINTER** mode, you must specify both the beginning and ending locations. Try this example:

```
P C000 C005
```

The printer will print:

```
C000 A2 00          LDX #$00
C002 BD D1 C4       LDA $C4D1,X
C005 F0 06          BEQ $C00D
```

The **PRINTER DISASSEMBLER** command can be used to print out a large section of memory. Try:

```
P C000 CFFF
```

The disassembly will continue until the **STOP** key is pressed. To turn off the printer, use the 'X' command to exit **SUPERMON**. To re-enter **SUPERMON**, use the '*' command.

The **REGISTER** values are displayed for you whenever you "break" to the monitor using the '*' command, or when a running program encounters a '00' (the **BRK <BREAK>** command). You can also obtain a display of the registers at any time by using the 'R' command:

Printer Disassembler (P)

Register Display (R)

```
.R [RETURN]
      PC  SR  AC  XR  YR  SP
.; CEB6 33 C5 F8 01 F8
```

You can change the values of the registers by using the screen editor to make the changes and then hitting [RETURN] to enter your changes.

While you're in the monitor, you can SAVE a machine language program that's in the computer by using the 'S', or SAVE, command as follows:

Save (S)

```
.S "TEST",08,2000,2100
```

The program TEST, which resides in the memory of the computer from memory location 2000 hex to 2100 hex, will be saved on drive number 8. The LOAD pointer will be set so that the next time the program is loaded into the machine using an *absolute load*, the program will be stored in the computer starting at 2000 hex, the memory location from which it was saved.

You can save to disk or cassette by using '08' or '01', respectively. Remember to use a two-digit number when specifying the device number.

You can use the 'T', or TRANSFER, command if you want to transfer a block of machine code from one memory location to another. First type [T], [SPACE], the beginning memory location, [SPACE], the ending location, [SPACE], and the beginning address to which you want to move the block. For example, to transfer the contents of the block of memory \$C000 to \$C100 to location \$1000, send this command:

Transfer Memory (T)

```
.T C000 C100 1000
```

Disassemble the code starting at \$1000 to show that the transfer has taken place. You should note that the transfer command only copies the code to the new location; it doesn't actually remove it from the original location. To confirm this, disassemble the code starting at \$C000. The code is still there.

To EXIT SUPERMON and return to BASIC, type this:

.X [RETURN]

A simple Assembler
D Disassembler
F Fill memory
G Go run
H Hunt memory
L Load program from disk or tape (absolute load)
M display Memory
P Printing disassembler
R Register display
S Save program to disk or tape
T Transfer memory
X eXit back to BASIC

Exit to BASIC (X)

SUPERMON QUICK REFERENCE



PART 6

Programming Tricks and Program Descriptions

In this chapter you will learn several programming tricks and short-cuts. We've assumed that you have a fundamental knowledge of BASIC programming. This chapter is set up alphabetically by topic.

You can add a short subroutine to a main program or combine two or three programs into one larger program by using an *append* routine. This will save your fingers a lot of unnecessary work.

The C-64 has an *auto relocater*. This means that you can load BASIC programs from tape or disk into any location of the BASIC RAM area. They normally load at the beginning of the BASIC RAM area, but we can change this. We can load a series of BASIC programs, one after the other, and then save them all as one large program.

Let's append two short programs. Type in the first program as follows:

PART 1

```
NEW [RETURN]
10 FOR A=1 TO 10
20 PRINT "THIS IS A TEST"
30 NEXT A
```

Save this program to tape or disk, naming it 'PART 1'. Now type the second program:

PART 2

```
NEW [RETURN]
50 PRINT "IT WORKS !!!"
60 GOTO 50
```

APPENDING PROGRAMS

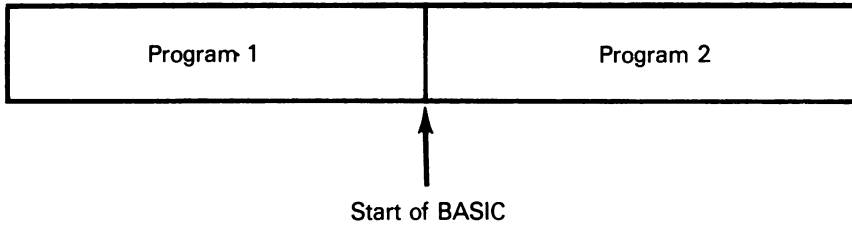


FIGURE 22-1. Start of BASIC pointer at the end of program 1.

Save this second program as 'PART 2'. To append PART 2 to the end of PART 1, follow this procedure:

1. Load PART 1 into the computer.
2. List the program and write down the last line number. In this case, it is '30'.
3. Now perform these POKES and PEEKs:

```
POKE 43,PEEK(45)-2:POKE 44,PEEK(46)
```

These commands tell the computer to find the end of the BASIC program in memory and POKE this value into the register that keeps track of the beginning of BASIC memory. These commands tell the computer that the beginning of BASIC is at the end of the program in the memory of the computer (see Figure 22-1). Make sure you have typed this line correctly; there's no margin for error when you are working with POKES and PEEKs. Type LIST and hit [RETURN]. No program lines are listed because we told the computer that BASIC memory starts at the end of our program. When it looks at BASIC memory, it finds nothing.

4. LOAD the next BASIC program (in this case LOAD PART 2):

```
LOAD "PART 2",8
```

The auto relocater of the C-64 will properly load PART 2. Type LIST and hit [RETURN] to see that the second program has loaded correctly. (If you want to append additional programs, you can type the POKES and PEEKs listed above. The only limitation to the number of programs you can append is the amount of memory of the C-64).

5. We now have both PART 1 and PART 2 in the computer, but only PART 2 will list. The computer thinks that the beginning of BASIC is at the end of PART 1. We must now tell the computer that the beginning of BASIC is at the original location: 2049 in decimal or \$801 in hexadecimal. We do this by typing the following POKEs:

```
POKE 43,1:POKE 44,8
```

To see if our programs are appended properly, type LIST and hit [RETURN]. Both PART 1 and PART 2 should be listed on your screen:

```
10 FOR A=1 TO 10
20 PRINT "THIS IS A TEST"
30 NEXT A
50 PRINT "IT WORKS !!!"
60 GOTO 50
```

A program that is to be appended to another should not have any line numbers lower than those of the program it is to be appended to. If the line numbers of the program you want to append overlap those of your first program, you'll need the services of POWER PLUS (on the SUPPORT DISK) or a similar utility to renumber the second program so that the beginning line number is larger than the ending line number of the first program. If this isn't done, you may have duplicate line numbers, or the GOTOs or GOSUBs in the two programs may find conflicting line numbers, causing the program to run improperly.

Sometimes when you enter the POKEs and PEEKs for the appending procedure, you may get a syntax error such as 'OUT OF RANGE'. This error occurs if the memory location value stored at 45 is less than 2, because when 2 is subtracted from this value in the command line, a negative number results. Since the range for a POKE value is from 0 to 255, the negative number will cause an error. To correct this, add a new line to the program using an unused line number:

```
0 REM
```

By adding the above line to your BASIC program, the end-of-BASIC pointer at location 45 will be increased by more than 2. You can now type the append POKEs and PEEKs without getting the error

message. Remember, it's only necessary to add the above line if you get an 'OUT OF RANGE' error when you enter the appending POKES.

You can blank the screen by typing this POKE:

```
POKE 53265, 43
```

To turn the screen back on, type the following POKE "blind", without being able to see a screen display:

```
POKE 53265, 27
```

Some situations in which blanking the screen is useful or even essential are described here.

Those of you who have used the Datassette may have wondered why the screen is blanked out when the computer is loading or saving programs using tape. The computer turns off the video processing chip to correct for timing problems that the CPU encounters when data is input or output using the Datassette. The 1541 disk drive doesn't have this timing problem, so the screen remains on during its input/output processes.

Like the Datassette, the older 1540 disk drives designed for the VIC-20 also have this timing problem when they are used with the C-64. To avoid loading and saving problems when you use the 1540 disk drive with the C-64, you must blank out the screen before LOADs or SAVEs. If you have a 1540 disk drive, have a technician install a new ROM chip to have the 1540 perform like a 1541.

The 1515 and 1525 printers (as opposed to the 1525-E printer) also have this timing problem. When using these printers, you must POKE the screen blank when the computer is sending output to these printers. The 1525-E printer has corrected this timing problem. You can upgrade the 1525 to a 1525-E by installing an upgrade ROM chip.

You can also blank the screen to hide a screen display that is being prepared by a BASIC program. If the screen isn't blanked, the user can see the screen output as it is being prepared. Use the screen blank POKE to hide the screen display until it has been completed. Then enable the screen with POKE 53265,27 to show the completed screen display.

BLANK SCREEN

The Datassette

The 1540 Disk Drive

The 1515 Printer

Hiding Information

Remember, the C-64 has two different character sets built into it. Press [CBM] and [SHIFT] together several times to toggle back and forth between them. One set is called the upper-case/graphics character set and the other the upper/lower-case set.

When you write programs, decide which character set you want. You can put the computer into that character set mode for the duration of the running program by putting a POKE or PRINT command at the beginning of your BASIC program.

For upper-case/graphics mode use:

```
POKE 53272,21      or
PRINT CHR$(142)
```

For upper/lower case use:

```
POKE 53272,23      or
PRINT CHR$(14)
```

There are several ways the colon (:) can be used to make it easier for you to use your computer. Let's study three of these.

The C-64 can store BASIC program lines up to 80 characters long. You can have more than one BASIC command on each line if you use the colon to separate these multiple commands. This will save you time typing your programs, and your programs will take up less memory. Here's an example:

```
10 FOR A=1 TO 10:PRINT"HELLO":NEXT A
```

You can also use the colon outside BASIC programs as a *delimiter*. The colon serves to mark the end of a set of characters or commands. Any information after the colon is ignored by the computer.

When you list a directory to the screen, you can save yourself some typing by loading programs directly from the directory listing. First get a directory listing by typing:

```
LOAD"$",8          (or @$ with POWER PLUS)
```

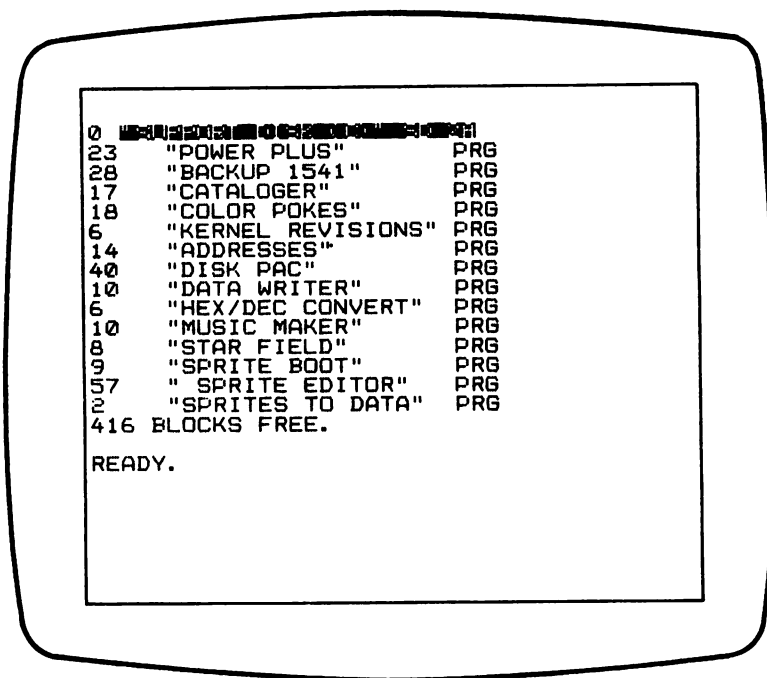
Your listing might look something like the one in Figure 22-2.

CHARACTER SWITCH

THE COLON

Separate Multiple Commands

The Colon as a Delimiter

**FIGURE 22-2. Directory listing.**

To load the program called DISK PAC from this directory listing, move the cursor to the line that lists the program DISC PAC and type 'LOAD' over the block numbers. Cursor past the file name and type ',8:' after the second quotation mark. Then press [RETURN]. (See Figure 22-3.)

The colon causes the computer to ignore the rest of that line, so we don't have to erase the extra characters in the line by using [SPACE].

You can also use the colon within BASIC programs to make your programs easier to read. Extra spaces in front of BASIC commands are normally removed automatically when they are entered. For example, try indenting your BASIC line by typing in this line:

Indenting BASIC Lines

```
NEW [RETURN]
```

```
10 [7 SPACES]PRINT"I WANT THIS INDENTED"
```

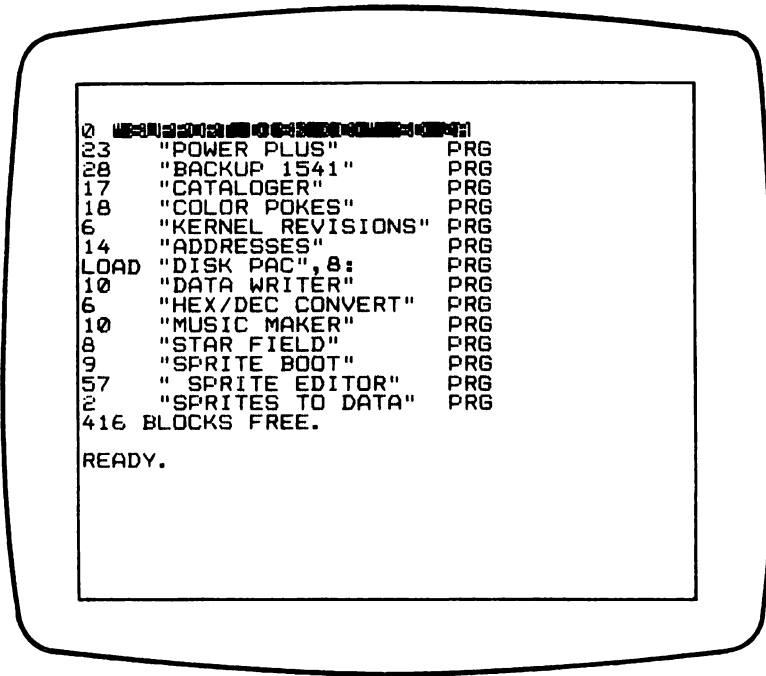


FIGURE 22-3. Loading a program from the directory.

Type LIST to see the line of BASIC in memory:

```
10 PRINT "I WANT THIS INDENTED"
```

The computer took your indentation spaces out.

There is a way of using colons to force the indentation of your BASIC lines after the line number. Look at the two program listings below to see how the colon can be used to achieve this. Which program is easier to read and understand?

```

10 FORB=1TO100
20 A=A+1
30 IFA=10THEN70
40 FORC=1TO10
50 PRINTC+A
60 NEXT
70 A=0
80 NEXT

```

Program 1

```

10 FORB=1TO100
20 :   A=A+1
30 :   IFA=10THEN70
40 :   FORC=1TO10
50 :       PRINTC+A
60 :   NEXT
70 :   A=0
80 NEXT

```

Program 2

Showing visually how the FOR NEXT loops are nested makes it much easier for you to follow the logic of the program. If you enter a few lines of PROGRAM 2 above and then list them, you'll see that the colons maintain the spacing you established.

It is possible for you to disable particular functions of the computer. Here we'll discuss several disable commands you can incorporate into your programs to give you better control over them even as they are running.

**DISABLE
COMMANDS**

There may be times when you don't want the user to switch from one character set to the other by using the character set toggle ([CBM] [SHIFT]). This toggle may be disabled by the following PRINT statement:

**Disable Character
Set Toggle**

```
PRINT CHR$(8)
```

To turn the character set toggle back on, use this PRINT statement:

```
PRINT CHR$(9)
```

To turn the entire keyboard off, try:

Disable Keyboard

```
POKE 649,0
```

To turn the keyboard back on:

```
POKE 649,10
```

Be careful when using this command. When you've disabled the keyboard, you can't enable it again by pressing keys. If the keyboard-disabling POKE is used in direct mode, you'll lose the use of the keyboard until the computer is reset or turned off and on. You can put the keyboard-enabling POKE into a program to turn the keyboard back on.

To demonstrate the effect of this disable function and the next one, you will need a short program to test them:

Disable LIST Command

```
NEW  
10 FOR B=1 TO 100  
20 PRINT B  
30 NEXT B
```

If you don't want anyone to list your BASIC program when it is in the computer, put this POKE in your program:

```
POKE 775,200
```

Try it to see how it works. Try listing the program in memory. To enable the LIST command, use this POKE:

```
POKE 775,167
```

You can disable the STOP key to prevent the user from inadvertently pressing [STOP], breaking out of a running program. Use the following POKE to disable the STOP key:

Disable [STOP] Key

```
POKE 788,52
```

Remember to turn the STOP key back on before you exit your program, with this POKE:

```
POKE 788,49
```

The following POKE will disable both the STOP and RESTORE keys. This prevents the user from terminating the program by RESTOREing.

Disable [STOP]/ [RESTORE] Keys

```
POKE 808,225
```

Remember, this POKE command leaves the user no way of breaking out of a running program except by turning the computer off, or by using a reset switch. Use this POKE carefully.

To enable the STOP and RESTORE keys:

```
POKE 808,237
```

CAUTION: Do not disable the STOP and RESTORE keys if you plan to use the cassette under program control. The cassette drive will not operate properly under these conditions.

The C-64 has four function keys on the right side of the keyboard. These keys provide the programmer with a convenient means for allowing the user to interact with a running program.

To use the function keys, you must program the computer to test whether any of these keys has been pressed. The GET command is the easiest way to do this. If you're not familiar with the GET command, see page 000. The following sample program will print a message when any function key—F1, F3, F5, or F7—is pressed.

```

10 GET A$
20 IF A$="[F1]" THEN GOSUB 100
30 IF A$="[F3]" THEN GOSUB 200
40 IF A$="[F5]" THEN GOSUB 300
50 IF A$="[F7]" THEN GOSUB 400
60 GOTO 10
100 PRINT "THIS IS FUNCTION KEY ONE":RETURN
200 PRINT "THIS IS FUNCTION KEY THREE":RETURN
300 PRINT "THIS IS FUNCTION KEY FIVE":RETURN
400 PRINT "THIS IS FUNCTION KEY SEVEN":RETURN

```

When you type a quotation mark and a function key, a reverse graphic character is printed to the screen because the quotation mark puts the computer in quote mode (see Chapter 12). Each function key has its own character code and an ASCII number value. You tested for these character codes in the example above. A listing of all eight function key character codes and their ASCII values is given in Appendix D.

You can use the BASIC command GOSUB to have the computer execute each of your subroutines when a particular function key is pressed. These subroutines can be as long as you wish. The RETURN command causes the computer to go back to get another character from the keyboard buffer. If a key hasn't been pressed, it will wait for you to hit another key.

The INPUT command is a convenient way of receiving data from the user in a running program. The INPUT command allows the user to type up to 80 characters. Pressing [RETURN] enters the entire line into the computer.

The INPUT command halts the program until this is done. If only a single character is to be entered by the user, the GET command will work better than INPUT.

When the computer encounters the GET command in a running program, the computer checks the keyboard buffer to see if any keys have

THE FUNCTION KEYS

THE INPUT AND GET COMMANDS INPUT

GET

been pressed. You don't have to press [RETURN] to enter the character. As soon as you press a key, the GET statement does the rest.

Let's write a short program to show how we can use the GET command to move a character around the screen:

```

10 PRINT "[CLR][12 CRS DN][20 CRS RT]*[CRSR LT]";
20 GET A$
30 IF A$="H" THEN PRINT "[SPACE][2 CRSR LT]
*[CRSR LT]";
40 IF A$="K" THEN PRINT "[SPACE]*[CRSR LT]";
50 IF A$="U" THEN PRINT "[SPACE][CRSR UP]
[CRSR LT]*[CRSR LT]";
60 IF A$="M" THEN PRINT "[SPACE][CRSR DOWN]
[CRSR LT]*[CRSR LT]";
70 GOTO 20

```

When this program is RUN, you can make the character '*' move around the screen by pressing any of the keys shown in Figure 22-4.

The GET command on line 20 tells the computer to look at the keyboard buffer to see if any keys have been pressed. The computer takes the character from the buffer and stores it as A\$. Lines 30 through 60 test the variable A\$ to see which key was pressed. Once the PRINT statement is carried out for one of the keys that was pressed, the GOTO command in line 70 sends the computer back to line 20 to see which key is pressed next. The GET command can only get one character at a time. If no key is pressed, a null value ("") is stored as the variable A\$.

In the above example we used the GET command to move an object on the screen, to get quick action in response to the keyboard. You can

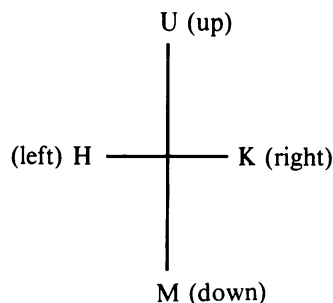


FIGURE 22-4. The direction keys.

also use the GET command to make your program pause until the user presses a key. This way, the user can choose when the program should continue running. The short BASIC line below uses the GET command to cause this sort of pause:

```
10 GET A$:IF A$="" THEN 10
```

When the computer encounters this line in a BASIC program, the GET command takes any key that is pressed and stores this key in variable A\$. When no key is pressed, the variable A\$ is equal to zero or null (""), and the command after the colon A\$="" is true. The IF/THEN statement is executed since A\$ equals "". This makes the computer go back to the beginning of the line for another try. The program remains frozen in this way until a key is pressed. When a key is pressed, the test for A\$="" is false, and the program will continue to the next line of BASIC.

Up to two joysticks can be plugged into the ports on the right side of your C-64. As we mentioned in Chapter 1, the joystick is an input device for drawing on the screen and for moving objects from one location of the screen to another. When the user moves the joystick, it sends a signal to the joystick memory registers in the computer. The computer can be directed to read the values in the appropriate joystick register to see which direction the stick is being pushed.

We can read the joystick registers by using the PEEK command. To read the value of joystick 1, we PEEK location 56321; for joystick 2, location 56320. In the example below we'll use variables J1 and J2 to hold our PEEK values.

```
J1=NOT PEEK(56321) AND 15
J2=NOT PEEK(56320) AND 15
```

By using NOT and AND, both BASIC keywords, we will be able to get a range of numbers from 0 to 10. These values and their associated joystick directions are shown in Figure 22-5.

To read the fire buttons, use the following PEEKs:

Fire Button 1:

```
F1 = - (PEEK(56321) AND 16)=0
```

Fire Button 2:

```
F2 = - (PEEK(56320) AND 16)=0
```

USING THE JOYSTICKS

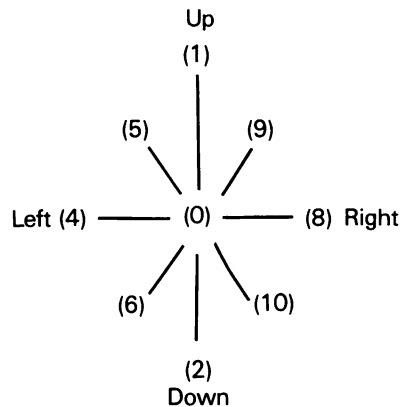


FIGURE 22-5. Joystick directions and their values.

F1 or **F2** will each have a value of '0' if the fire button is pushed, and '1' if it is not.

To see how to use the joystick **PEEK** statement in a program, run the following program:

```
NEW [RETURN]
100 A$="[HOME][10 CRSR DN][10 CRSR RT]"
110 B$="[HOME][10 CRSR DN][25 CRSR RT]"
120 FOR A=1 TO 10:READ X$(A):NEXT A
130 PRINT"[CLR]"
140 J1=NOT PEEK(56321)AND15
150 J2=NOT PEEK(56320)AND15
160 F1=-(PEEK(56321)AND16)=0
170 F2=-(PEEK(56320)AND16)=0
180 IF J1 THEN PRINT A$;X$(J1)
190 IF J2 THEN PRINT B$;X$(J2)
200 IF F1 THEN PRINT A$;"[2 CRSR DN]BUTTON ONE"
210 IF F2 THEN PRINT B$;"[2 CRSR DN]BUTTON TWO"
220 FOR I=1 TO 20:NEXT I:GOTO 130
230 DATA UP,DOWN,BLANK,LEFT,UP-LEFT,DOWN-LEFT
240 DATA BLANK,RIGHT,UP-RIGHT,DOWN-RIGHT
```

This program prints on the screen the direction you are aiming the joystick and whether the fire button is pressed. By adding your own subroutines, you can write a program to execute particular movements on the screen depending on which way the joystick is moved.

The REM statement is normally used to add REMarks, or comments, to a program. When the computer encounters REM statements during a program run, it ignores the characters that follow the REM statement and continues to execute program instructions. To see how the REM statement can be used in a program, try:

```
10 REM THIS IS ONLY A TEST
20 PRINT "WARNING: USING COMPUTERS MAY BE HABIT
FORMING"
```

The REM statement can be used to help debug a BASIC program. You can use the REM statement to remove the function of a program line temporarily, without erasing the line from the program. To see what happens when a particular line is not executed as a program is run, insert a REM at the beginning of the BASIC line that you wish to eliminate and press [RETURN]. This will change the BASIC instruction line into a REM line. Here's an example:

```
230 PRINT "THIS IS THE TIME"
240 PRINT "AND THIS IS THE PLACE"
250 IF A=1 THEN 300
```

We want to find out how the program will operate without line 240. To prevent line 240 from being executed, we could have deleted it by typing 240 and [RETURN]. However, to recover line 240 for later use, we insert a REM at the beginning of the line:

```
240 REM PRINT "AND THIS IS THE PLACE"
```

If you now run this program, the REM statement keeps line 240 from executing. By deleting the REM statement, you can return 240 to active status.

The random function built into the C-64 will generate random numbers for you. You can use this function for games of chance requiring random outcomes such as the roll of dice. You can also use randomly selected numbers to direct the random motion of characters in arcade-style games. You can generate a random number by using the BASIC keyword RND:

```
PRINT RND(1)
```

This will give you a number between 0 and 1. The '1' inside the parentheses is a "dummy" character. Write down the random number

REM STATEMENTS

RANDOM NUMBERS

that the above command printed to the screen. To simulate the roll of dice, type this short program:

```
10 PRINT "PRESS ANY KEY TO ROLL DICE [ 3 SPACES ] ";
20 GET A$:IF A$="" THEN 20
30 X=INT(RND(1)*6+1)
40 PRINT X
50 GOTO 10
```

Line 30 generates a decimal fraction between 0 and 1 (RND(1)), multiplies it by 6 (*6) to get a number greater than or equal to 0 and less than 6, and then adds 1 (+1) and gets rid of the fractional part (rounds it off with INT, the INTeger command), to arrive at a number between 1 and 6.

Turn off the computer and type:

```
PRINT RND(1)
```

This random number is the same as the one you wrote down. To get a true random number, you must set the random number generator with a random seed. This is done best by using the **TI** jiffy clock (discussed later in this chapter) to initialize the random function:

```
RND(-T1)
```

Since the jiffy clock value is constantly changing, initializing the random function with a negative **TI** value at the beginning of your program will ensure a different sequence of numbers.

On occasion, some change that you or your running program makes to the memory registers of the computer will “lock up” the computer, causing the program to stop running. If you can’t get a flashing cursor on the screen, you no longer have control of the computer. Try regaining control by pressing [RUN/STOP][RESTORE]. If nothing happens, try pressing these keys again, hitting the [RESTORE] key sharply, while holding down [RUN/STOP]. If you regain the cursor by doing this, you should perform a soft reset, as described below, to set the important memory registers of the computer back to their power-up values without losing the programs in the memory of the computer.

SYSTEM RESET

If your computer has locked up, but you’ve regained control over the keyboard, type:

```
SYS 64738
```

Soft Reset

This SYStem call unlocks the computer as turning the computer off and on would, but without erasing the programs in memory. If POWER PLUS was in the computer when it locked up, type SYS 52426 to get back POWER PLUS. To get your BASIC program back, use the UN-NEW command (*U) as described in Chapter 19.

If the RESTORE key doesn't give you back the cursor, the computer has crashed, and you must turn the computer off and then on again. This method of resetting the computer has several drawbacks:

Hard Reset

1. You lose the program that was in the memory of the computer;
2. You'll eventually wear out the on/off switch; and
3. Your video monitor and possibly other peripherals may suffer from the electrical "spike" sent to them from the computer when you turn it on.

The reset switch avoids these three problems. Have a service technician install the reset switch for you. If you're familiar with electronics, follow the instructions in the box to install your own reset switch.

INSTALLING A RESET SWITCH

This installation voids Commodore's 90-day warranty on the computer, so wait until it has expired before you install the switch.

Purchase a miniature momentary-on switch from an electronics shop. Solder one end of a 6" length of double-strand wire to each of the poles of the switch, as shown in Figure 22-6. Open the computer housing by removing the three screws at the bottom and lift up the hinged cover. Find the user port pins inside the housing. Solder one wire to the lead from pin one and the other wire to the lead from pin three. Now drill a 9/32-inch hole in the left panel of the computer, as shown in Figure 22-7. Secure the switch in the hole using the nut that came with the switch. The job is done.

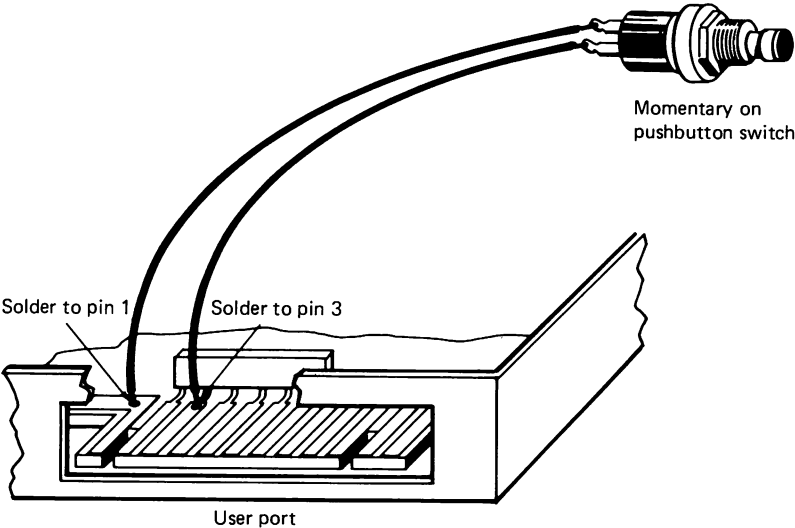


FIGURE 22-6. Installing a reset switch.

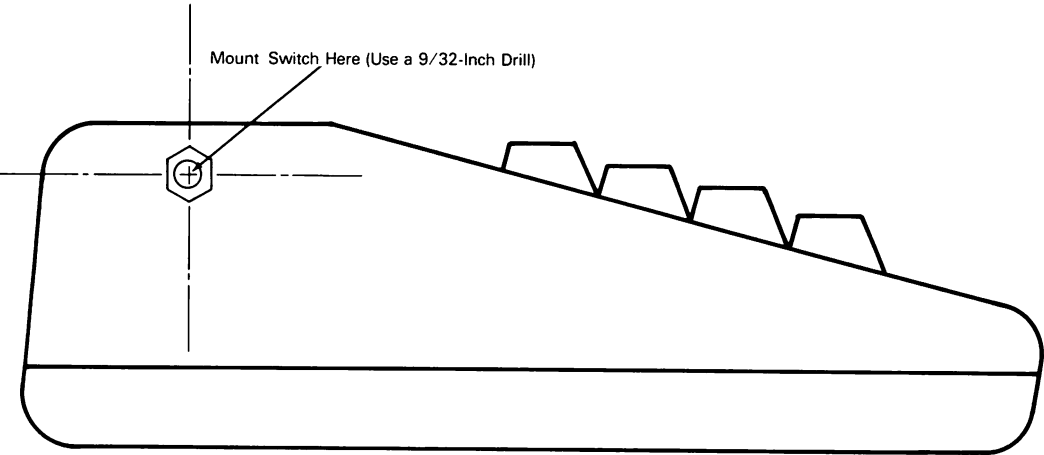


FIGURE 22-7. Mounting the reset switch.

To reset the computer with the reset switch, push the button for a moment. The video display will shrink and then pop out with the power-up display. This type of reset retains the programs in memory. To retrieve your programs, follow the instructions for UN-NEW described in Chapter 19. Although the programs themselves will be retrieved, any data that you entered will be lost.

The C-64 has a real-time clock that you can access. Two BASIC variables are reserved for these clocks.

TIME CLOCKS

The first is called the jiffy clock. Its reserved variable is **TI**, which stands for TIME. The jiffy clock increments every 1/60 of a second, or one jiffy. The **TI** variable is automatically set to zero when the computer is first turned on.

The TI Function

To use the jiffy clock as a timer, first set the value of a variable equal to **TI**. In the example below, the value of **TI** is put into variable **X**. To see when **TI** reaches our desired time (in this case 10 seconds), we compare **TI** with the present variable **X**, using the IF/THEN:

```
100 X=TI
110 IF TI<X+(60*10) THEN 110
120 PRINT "TIME'S UP"
```

When this program is RUN, the computer will pause for 10 seconds and then print 'TIME'S UP'. The product of the expression 60*10 was added to the **X** variable because each jiffy is equal to 1/60 of a second, so one second is equal to 60 jiffies.

To see how many seconds your computer has been on, try printing the **TI** value divided by 60:

```
PRINT TI/60 [RETURN]
```

The second reserved time variable, **TI\$**, is the 24-hour clock. To find out how long the computer has been on, use this command:

The TI\$ Function

```
PRINT TI$
```

The format for **TI\$** is as follows:

TI\$ = "hhmmss",

where: 'hh' is hours,
 'mm' is minutes,
 'ss' is seconds.

You can set this clock to the actual time by having **TI\$** equal the 6-digit number as shown above, with the first two digits being hours, the second two digits minutes, and the last two digits seconds. Let's set the clock to 11:04 and 20 seconds PM:

```
TI$ = "230420" [RETURN]
```

All six digits must be used to avoid a syntax error.

You can use the **TI\$** function to have your computer act as an alarm clock. Run this alarm clock program:

```
100 INPUT "CURRENT TIME '000000' ";TI$
110 INPUT "ALARM TIME '000000' ";X$
120 PRINT "[CLR][23 RIGHT] ALARM TIME=";X$
130 PRINT "[HOME]CURRENT TIME=";TI$
140 IF TI$<X$ THEN 130
150 S=54272
160 POKE S+1,20:POKE S+5,9
170 POKE S+6,9:POKE S+24,15
180 POKE S+4,17
190 FOR T=1 TO 40:NEXT:POKE S+4,16
200 FOR T=1 TO 500:NEXT:GOTO 160
```

The **TI\$** variable will continue to keep the current time until you either turn the computer off or reset the **TI\$** to another time value. (Using the Datassette will affect the accuracy of this clock.) Since setting the **TI\$** variable also sets the jiffy clock, be careful when you use both the **TI** and **TI\$** variables at the same time. The **TI** jiffy clock may be reset to zero by having **TI\$**="000000".

Program Descriptions

The discussions in this chapter will show you how to operate programs on the SUPPORT DISK. You can learn a great deal about BASIC programming by studying how these programs are put together. If you purchased this book without the SUPPORT DISK, you can order the disk from the authors by mailing the enclosed order form with your payment.

This program can be used to keep track of names, addresses, and telephone numbers. Load and run the ADDRESSES program. The first prompt asks if you want to output to a printer. Press [N] if you don't want printer output. The next prompt asks if you are printing envelopes. This option will be explained later; press [N] for now.

Now you'll be asked if you want to Add to or Search the file. If this is the first time you are running this program, you should put some information in your file, so answer this question by pressing [A]. You can now enter a name, street address, city/state/zip, and telephone number. When you are prompted for comments, enter anything you like (account number, description, etc.).

After you enter the comment line, ADDRESSES again provides the Add [A] or Search [S] prompt. You may choose either to add new records or to search through existing ones. If you choose to search the records by pressing [S], it will prompt you with 'name'. Enter either the last name, or the first name, or both. If you enter both, make sure that last name and first name are in the same order as you entered them originally. If you want to view all your entries, hit [RETURN] when you see the cursor over the '*'. To exit from search, enter an up arrow (↑) when you are asked for a name. If you choose to exit the program altogether, type 'end' and hit [RETURN].

ADDRESSES

Adding Data

Searching for Data

If you said 'yes' to printer output, all files that you search will be sent to the printer. If you said 'yes' to envelopes, the complete address record will be printed without the telephone number and comments. This will properly address an envelope in your printer (printers with friction feed only). If you want a printout of all data, press [RETURN] when the cursor is blinking on the '*'.

Printer Output

By studying how this program works, you'll learn how to incorporate data directly into a program, rather than into a separate file. After you enter an address file and press [RETURN], the screen clears, and a number of lines are printed at the top of the screen for just a second. Then the prompt returns. Break out of the program by pressing [STOP], and list the program. At the end of the ADDRESSES program you will see all the files you entered in the form of DATA statements. The process of entering BASIC lines while a program is running is called the *dynamic keyboard*. When a searched name is displayed on the screen, you'll see that there is a number above the address for each name. This number tells you which BASIC line stores the data for that name. Each of your entries is added to the program in the form of a DATA statement at the end of the program. Then the entire program, along with your data, is saved automatically, replacing the original program on the disk.

How it Works

If you want to make corrections to the names and addresses you typed in, you can list the program starting with the line number that was displayed above the address, and use your BASIC screen editor to make corrections. Resave the program when you have finished, using the SAVE AND REPLACE command described on page 77.

Making Changes

When you have finished adding new names to your address file, enter Search mode and type 'END'. The program will now automatically resave the entire program, along with your data, on disk.

Saving Your Data

If you want to scratch all your name and address records, list the program to find out where your data is. Use the SUPERBASIC utility that is part of POWER PLUS (on the SUPPORT DISK) to delete the range of lines containing your data. Now you can start a new file of names and addresses.

Deleting Names

In order to ensure that your programs and data are safe from accidental loss, keep copies of them in a cool, dry place. To make it copy the entire contents of a disk to a second disk using only one disk drive.

BACKUP 1541

easier for you to make copies of your disks, we have included a backup program on the SUPPORT DISK. BACKUP 1541* allows you to

The BACKUP program creates a buffer, or temporary storage area, in the computer. It then reads information from your “source” disk into this buffer area. When the buffer is full, its contents are dumped to a second disk called the “destination” disk. Since the buffer isn’t big enough to hold all the information that a diskette is capable of holding, you must reinsert the source disk to fill the buffer with more data, then dump this additional data to the destination disk. This exchange of disks is repeated until the entire contents of the source disk are recorded on the destination disk.

The program will tell you what to do by printing instructions on the screen as you go through each step of the backup. This program can perform two different types of backups, the BAM Select Backup and the Direct Backup.

When you choose the BAM Select Backup, the program will copy all the files that are listed in the directory of the diskette. Some commercially available programs delete the names of files from the directory as a protection technique, so the BAM Select Backup will not copy them. If you use the BAM Select mode and your program copy does not run properly, try the Direct Backup mode.

The BAM Select Backup

The Direct Backup mode will copy every byte of information on the source diskette exactly, even all the spaces. This ensures that any hidden files are copied properly. To copy the entire disk, about six disk exchanges are required. This procedure will take over 20 minutes, so don’t use the Direct Backup mode unless you are certain it is necessary.

The Direct Backup

1. When the screen for the backup program is displayed, you will see a checkered cursor in a box called “BACKUP COMMAND”. Hit [RETURN] for BAM Select Backup. Press [D] if you want a Direct Backup instead.
2. Now your cursor is in the box called “DISK DESTINATION”. Type in a name for your destination disk and hit [RETURN].
3. The program now goes to the right of this name and waits for you to type in a two-character ID number. Do not use the same ID number as the one on the disk you are copying.

Steps for Backing Up Disks

*BACKUP 1541 was modified from a program that Commodore Business Machines put in the public domain.

4. You will now be prompted to insert the destination disk. You may use a blank, unformatted disk, or one that already has data on it. However, a disk that has information on it will be completely erased by the backup process, so be sure that you no longer need the information on the diskette before you use it for the backup.
5. Hit any key to start formatting the destination disk. Formatting takes about one and a half minutes.
6. At the prompt, remove the destination disk and insert the source disk. Hit any key.
7. At the next prompt, remove the source disk and insert the destination disk.
8. Repeat steps 5 and 6 as you are prompted by the program until you see the message 'BACKUP FINISHED'. Your backup disk now contains a complete copy of the source disk.

The CATALOGER program permits you to print out (alphabetically if you like) directories of your disks. These printed directories make it easy for you to keep track of the programs and data files you have without having to load and view the directories of the disks using the computer and disk drive.

CATALOGER also gives you the beginning and ending addresses of all the programs on your disk. Most programs load into the computer where the BASIC memory area lies, at location \$0801 (hexadecimal), or 2049 (decimal). Machine language programs load into other areas of the computer's memory. You may have noticed that these machine language programs cannot be saved using our normal SAVE command. To save them, you must know where they will load in the computer's memory. CATALOGER goes through each program on the disk and lists its beginning and ending address. These addresses are the registers into which the program will load. Once you have these addresses, you can use SUPERMON to save your machine language program on another disk (see page 243).

CATALOGER is easy to operate. Load and run it. Place the disk you want to catalog in the drive, and press any key when you're ready to begin. After the program finds the name of the disk and its ID number, it will display them and then it will ask if you want the directory to be ordered alphabetically. The default is 'N' for No. Change the 'N' to 'Y' for Yes if you want the disk directory printed in alphabetical order. If you press [RETURN] on the 'N' default, CATALOGER will order the directory as it appears on your disk.

CATALOGER

NAME = 'SUPPORT DISK (C)' : ID = 'SD'

FILE NAME	TYPE	BLKS	ADDRESSES
-----	----	----	-----
SPRITE EDITOR	PRG	57	0801 3FB7
ADDRESSES	PRG	14	0801 155D
BACKUP 1541	PRG	28	0801 23B2
CATALOGER	PRG	17	0801 1828
COLOR POKES	PRG	18	0801 197B
DATA WRITER	PRG	10	0801 111A
DISK PAC	PRG	40	0801 2EFE
HEX/DEC CONVERT	PRG	6	0801 0D24
KERNEL REVISIONS	PRG	6	0801 0D46
MUSIC MAKER	PRG	10	0801 118C
POWER PLUS	PRG	23	0801 1E00
SPRITE BOOT	PRG	9	0801 1015
SPRITES TO DATA	PRG	2	0801 09F6
STAR FIELD	PRG	8	0801 0FEB

BLOCKS FREE 416

FIGURE 23-1. An alphabetical catalog of the SUPPORT DISK.

CATALOGER will then display the name of each file as it encounters and examines it. After it has looked at all the files, it will list them either alphabetically or as they appear on the disk directory, depending on your choice. Figure 23-1 shows an alphabetically ordered listing of the contents of the SUPPORT DISK.

CATALOGER's output will tell you the file type for each file on the disk. For programs, it also gives you the beginning and ending addresses in hex. For data files, it tells you how long they are in bytes.

Make a catalog of each of your disks, and put these directories in a binder for future reference.

If you want to get the beginning and ending addresses of only one machine language program, CATALOGER may take some time to give you the result, because it goes through the files as they are ordered on the disk directory. To save time, use the View Load Addresses option of the DISK PAC program (on the SUPPORT DISK), as described on page 274.

Each memory location of your computer has an address number similar to a post office box number. These locations contain numbers

COLOR POKEs

that are special instructions for the computer. You may change the value stored in a location by POKEing a new number into it.

The POKE values for colors are listed in Appendix D. You can change the screen, border, or character color by POKEing a different value into the appropriate memory location.

The COLOR POKES program shows what all the possible screen-border-character color combinations look like so that you may choose the one that suits your needs. You can then put these values into the program of your choice.

DATA WRITER is a program that will convert machine language programs in the memory of the computer into DATA statements that can be appended to a BASIC program. When the BASIC program containing the DATA statements is run, the values in the DATA statements are POKEd into the memory of the computer to recreate the machine language routine.

Machine language programs written as DATA statements are easier to append to other programs than actual machine code. You can use DATA WRITER to transform sprite data, or other forms of machine code, into DATA statements that are then appended to a BASIC program.

To use DATA WRITER, the machine language routine that you want to convert to DATA statements must be stored in the computer already. Make sure that the machine code is in an area that will not be overwritten by your BASIC program. If the machine language routine is less than 200 bytes, you can store it in the cassette buffer (from memory location 828 to 1019). If it is longer, you can store it in the RAM area from \$C000 to \$CFFF.

When DATA WRITER is run, you will be asked for the beginning memory location of the machine code routine that you want to put into DATA statements. You may enter the location address in either decimal or hexadecimal numbers. If you are entering your location in hex, put a dollar sign (\$) before the number. When [RETURN] is pressed, you'll be asked to enter the ending location of your machine code. Again, you can use either a decimal or a hex number.

When you press [RETURN], each memory location that contains your machine code is PEEKed, and its value is placed into a DATA statement. When all memory locations have been transformed into DATA statements, delete lines 100 to 850, leaving only the DATA statements and a LOADER routine that takes care of performing the POKES. The LOADER routine and DATA statements start at line 60000. Now save this BASIC program to tape or disk.

To use the DATA statements, you must append them along with the LOADER to your main BASIC program (see page 247 for instruc-

DATA WRITER

DISK PAC (C)	EDUCOMP
1. DELETE DISK FILES 2. CHECK DISK BLOCKS 3. VIEW LOAD ADDRESSES 4. VIEW DISK DIRECTORY 5. UNSCRATCH DISK FILES 6. CHANGE THE DISK NAME 7. CHANGE THE DISK ID NUMBER 8. CHANGE THE DISK DEVICE NUMBER 9. EXIT PROGRAM	
PRESS KEYS 1-9 TO CHOOSE SELECTION	

FIGURE 23-2. The DISK PAC menu.

tions on appending programs). After the DATA routine is appended to the main program, enter a BASIC line containing GOSUB 60000 at the beginning of the program. As soon as the program is run, the computer will go to line 6000 and put the values in the DATA statements into the appropriate memory locations, the same locations the machine code was originally loaded into. At the point (or points) in the BASIC program that you need to execute your machine language program, insert the proper SYStem call to the beginning location of your machine language program.

DISK PAC is a utility program package that offers you eight different disk utilities that help you reorganize your diskettes easily and quickly. When you load and run DISK PAC, you'll see a menu of nine different options, as shown in Figure 23-2.

A description of each of these utilities follows.

When you choose this disk utility by pressing [1], the program will read the directory of the diskette in the disk drive. It will then display the number of blocks free. Then it will display the name of the first file on the directory, and file type. For example, it might display "MONOPOLY PRG", or "LETTER SEQ". If you wish to delete the file displayed from the diskette, press [Y] for 'Yes'. If you press [N] for 'No' at this point, the next file name will be displayed. The program will continue to display each file name that is on the disk and wait for you to decide whether you wish to delete that file.

After you've decided whether to delete each file, the names of the files you selected for deletion will remain on the screen. The program will scratch all these files automatically. When the deletions have been

DISK PAC

1. Delete Disk Files

completed, you will be returned to the main menu. If you unintentionally scratch a file, you can unscratch it with option 5, described below.

If you are having trouble with a particular disk when you load files from it, you may use this routine to check the disk to see if you have a bad sector. Each sector is examined for read errors, and if any are found, the track and sector containing the error is identified on the screen. When the entire disk has been examined, the total number of errors is indicated. Unlike other disk check programs, this routine will not erase any data from the diskette. You can exit the program at any time by pressing any key.

If this process indicates any errors on the diskette, it's a good idea to make a backup of the entire diskette (in order to save any good files) using the BACKUP 1541 program which is on the SUPPORT DISK. Once the diskette is backed up, run the CHECK DISK program, provided on the TEST/DEMO disk that came with the 1541 disk drive, on your original diskette. This program checks the diskette by writing to, and reading from, each sector of the diskette.

CAUTION: The CHECK DISK program on the TEST/DEMO disk will erase all the information on your diskette.

If your diskette passes this test without showing any bad sectors, it may be used again to record your programs or data files. If it fails the CHECK DISK test, the diskette should be discarded.

This option will give you the beginning and ending load addresses for any program you choose. You'll need to know these addresses to save machine language programs, as described on page 243. When you choose this option from the menu, you'll be prompted to type in a file name. If this file name is found, the beginning and ending load addresses will be displayed for you in both hexadecimal and decimal number systems. If the file name is not found, or if the file is something other than a program file, an error will be displayed.

If you want to see what files you have on a disk, you may use this option to view the disk directory without leaving the DISK PAC program. After you finish studying the directory, press [RETURN] to go back to the menu.

If you accidentally Scratch a file from a diskette, you may retrieve it by selecting this option from the menu. When a file is scratched using the Scratch command (described in Chapter 8), the program is still on the diskette. However, the BAM does not retain the name of that file

2. Check Disk Blocks

3. View Load Addresses

4. View Disk Directory

5. Unscratch Disk Files

in the directory. Furthermore, the BAM now considers the sectors occupied by this file to be free for the storage of new information. However, as long as you have not saved a new file in one of these sectors, you can UNSCRATCH the scratched file with this routine.

When you choose the unscratch, the BAM of the diskette is read into the computer. Then the diskette is searched for a scratched file. If a scratched file is found, its name will be displayed, and you will be prompted to press [Y] if you wish to unscratch this file, or [N] to leave it scratched. If you press [N], the program will then search for another scratched file.

When you press [Y], you will be prompted to press [1] if it is a sequential file, [2] if it is a program file, [3] if it is a user file, and [4] if it is a relative file. The file will then be unscratched. The file's name will now be included in the disk directory, and the BAM will consider the file's sectors allocated rather than free.

You must validate the diskette after using the unscratch option so that the directory listing will indicate the correct number of blocks. Use the 'V' command of the POWER PLUS utility, the C-64 WEDGE program, or another equivalent utility.

This program will permit you to change the name of a diskette. The diskette name is the name you entered when the diskette was formatted, or NEWed. This function can be particularly useful when you're organizing your diskette library.

When you choose this option from the menu, the current name of the diskette will be displayed. You'll then be asked if you wish to change this name to a new one. If you press [N], you'll be returned to the menu. If you press [Y], you'll be asked to type in a new name for the diskette. Once the name change is completed, the new name will be displayed for a moment, and then you'll be returned to the menu. Note that you can't use more than 16 characters for the name of a diskette. If you try to, you'll be prompted to enter the name again.

6. Change the Disk Name

This option of DISK PAC will change a diskette's current two-character ID number to a new one. This can be particularly useful when you are coding and organizing your diskettes by their ID numbers.

When you choose this option, the old ID number will be displayed for you, and you'll be asked if you want to change it. If you press [N], you'll be returned to the menu. If you press [Y], you'll be asked to type in a new ID number. ID numbers must be one or two characters in length. If you type three or more characters, you'll be prompted to type in the new ID number again.

7. Change the Disk ID Number

This option allows you to change the device number of your disk drive from 8 to any number from 9 to 12. This is useful only if you're using more than one disk drive. All drives are assigned drive number 8 by the manufacturer, but this number may be changed temporarily through programming. If you're using more than one disk drive, each drive must be assigned a different number so that you can specify which drive you want to access with a particular disk drive command.

When you choose this option, you'll be instructed to turn off all drives except the one that needs a new device number. You're also asked to input the old device number of that drive unit. The default for this value is 8. If the old drive number is 8, simply hit [RETURN]. If the old drive number is not 8, type in the appropriate number over the 8 and hit [RETURN]. You're then asked to enter the new device number. The default for this value is 9. If you wish to change your drive device number to 9, hit [RETURN]. To assign another drive number, type a different number over the 9 and hit [RETURN].

When you have finished using DISK PAC, press [9] to exit the program.

HEX/DEC CONVERT is a program that will convert decimal numbers to hexadecimal numbers, and hexadecimal numbers to decimal numbers. You'll find this program useful when you use the machine language monitor, SUPERMON, which is part of the POWER PLUS utility provided on the SUPPORT DISK.

To use HEX/DEC CONVERT, load and run the program in the normal manner. Type in the number you want to convert. If it's a decimal number, simply type the number and press [RETURN]. If you want to convert a hexadecimal number to decimal, type '\$', the hex number, and then [RETURN]. You may enter any number, as long as it's in the range listed below:

DECIMAL: 1 to 65535

HEXADECIMAL: \$1 to \$FFFF

To exit the program, enter '0' and press [RETURN].

Over time, Commodore has used more than one version of the Kernel ROM in the C-64. This means that some of the built-in functions of the computer are different from one computer to another. Commodore has changed these chips in an effort to eliminate some bugs that were discovered in earlier versions. Unfortunately, these chips introduced additional bugs.

8. Change the Disk Device Number

9. Exit Program

HEX/DEC CONVERT

KERNEL REVISIONS

With a Kernel 1 ROM, if you POKE a character to one of the 1000 screen locations, you'll see that character as white against the background color. If the background color is white, you won't see the character at all. Try POKEing the letter 'A' (code is 1) into the HOME position of the screen (location is 1024), like this:

```
POKE 1024,1
```

Did you see a white 'A' in the HOME position? If so, you have Kernel 1 ROM.

When you use the Datassette with a C-64 with Kernel 1, after a program is found, the computer will wait for you to press [CBM] before it loads the program from tape.

The screen POKE as described above will not work properly with Kernel 2. You will not see the character that you POKE to the screen because the color of the character is the same as the background color. But when you press [HOME], you'll see that the letter 'A' is there; it appears as the cursor blinks on and off. If your C-64 has Kernel 2, the SCREEN POKE FIX on page 31 will enable you to POKE characters to the screen in any color you want.

When you use the Datassette with a Kernel 2 Commodore 64, there will be a short pause after the program name is found, but after several seconds, the LOAD will begin without your having to press [CBM].

A Kernel 3 ROM computer will display characters POKEd to the screen in the same color as the cursor color. In most other aspects, Kernel 3 computers are similar to those with Kernel 2.

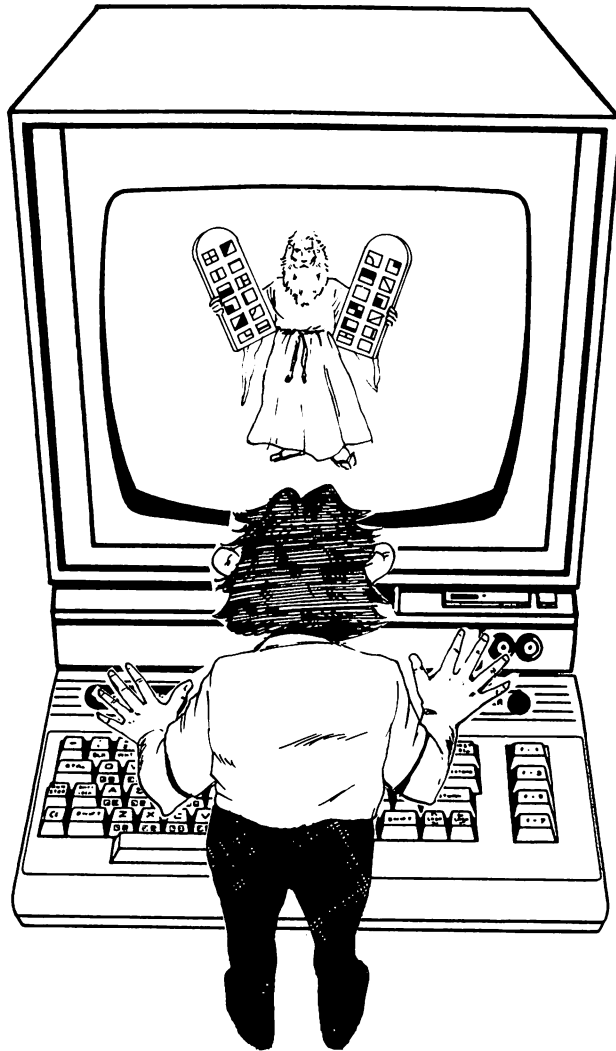
Load and run the KERNEL REVISIONS program. It will help you discover which Kernel you have.

Kernel 1

Kernel 2

Kernel 3

The Program



PART 7

Appendices

APPENDIX A

Recommended Products for the C-64

MAGAZINES

AHOY!

45 W. 34 St., Suite 407
New York, NY 10001

COMMANDER: THE COMMODORE USER'S MONTHLY JOURNAL

P.O. Box 98827
Tacoma, WA 98498

Commander provides in-depth articles on telecomputing, BASIC programming, machine language programming, etc. It will keep you informed of all new products that become available for the C-64.

COMMODORE: THE MICROCOMPUTER MAGAZINE

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380

Commodore is published six times a year. Articles are intended for intermediate-level users. A broad range of subjects including programming tips, program listings, and new hardware are presented in a clear, concise way.

COMPUTE!

P.O. Box 5406
Greensboro, NC 27403

Published each month, *Compute!* covers several computers, including the C-64. The program listings and discussions of new software are valuable for users.

COMPUTE!'S GAZETTE

P.O. Box 5406
Greensboro, NC 27403

The *Gazette* is intended for VIC 20 and C-64 users. It is an excellent source of current information on hardware and software for the C-64. Many excellent short utility and game programs are included.

THE MIDNITE SOFTWARE GAZETTE

635 Maple Street
Mt. Zion, IL 62549

The *Midnite* is an independent review journal for Commodore computers. It is published bi-monthly. Honest and unbiased reviews of hardware and software products fill its pages.

RUN

80 Pine Street
Peterborough, NH 03458

Run is a monthly magazine for VIC 20 and C-64 computers. It covers programmer's tips, hardware and software reviews, and provides programs you can type in.

WORD PROCESSORS

EASY SCRIPT

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380

PAPERCLIP (Batteries Included)

186 Queen Street West
Toronto, Ontario
Canada, M5V1Z1

QUICKWRITER III

Educomp
2139 Newcastle Avenue
Cardiff, CA 92007

WORDPRO 3+/64

Professional Software, Inc.
51 Fremont Street
Needham, MA 02194

DATA BASE PROGRAMS**MIRAGE CONCEPT'S DATABASE MANAGER**

Mirage Concepts, Inc.
2519 West Shaw, #106
Fresno, CA 93711

THE CONSULTANT (Batteries Included)

186 Queen Street West
Toronto, Ontario
Canada, M5V 1Z1

SUPERBASE 64

Precision Software, Inc.
1675 York Avenue, Suite 11D
New York, NY 10128

BUSINESS SOFTWARE**CALC RESULT: ADVANCED**

Computer Marketing
300 West Marlton Pike
Cherry Hill, NJ 08002

PRACTICALC 64

Micro Software International, Inc.
The Silk Mill
44 Oak Street
Newton Upper Falls, MA 02164

COMPLETE PERSONAL ACCOUNTANT

Futurehouse
P.O. Box 3470
Chapel Hill, NC 27514

THE TAX ADVANTAGE

Continental Software
11223 South Hindry Avenue
Los Angeles, CA 90045

PORTFOLIO MANAGER

Basic Byte, Inc.
P.O. Box 924
Southfield, MI 48037

GRAPHICS SOFTWARE

SPRITEMASTER 64

Access Software, Inc.
925 East 900 South
Salt Lake City, UT 84105

SPRYTEBYTER

Foxsoft
P.O. Box 507
Deer Park, TX 77536

DOODLE! II

City Software Distributors, Inc.
735 West Wisconsin Ave.
Milwaukee, WI 53233

SOUND SOFTWARE**SYNTHY 64**

Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510

STUDIO 64

En-Tech Software
P.O. Box 881
Sun Valley, CA 91353

MUSICALC

Waveform Corp.
1912 Bonita Way
Berkeley, CA 94704

UTILITIES**POWER PLUS**

Educomp
2139 Newcastle Avenue
Cardiff, CA 92007

SYSRES

Solidus International
West 15th Street, Suite 6
North Vancouver, B.C.
Canada, V7M 1R5

DISK PAC

Educomp
2139 Newcastle Avenue
Cardiff, CA 92007

ULTRABASIC

Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510

CLONE MACHINE WITH UNGUARD

Micro-W. Distributing, Inc.
P.O. Box 113
Pompton Plains, NJ 07444

COMPILERS

PETSPEED

Limbic Systems, Inc.
1056 Elwell Court
Palo Alto, CA 94303

DTL COMPILER 64

Cimarron
2158 South Hathaway Street
Santa Ana, CA 92705

ASSEMBLERS

M.A.E. ASSEMBLER

Eastern House Software
3239 Linda Drive
Winston-Salem, NC 27106

TERMINAL SOFTWARE

VIDTEX

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380

SUPERTERM

Midwest Micro, Inc.
311 West 72nd Street
Kansas City, MO 64114

ENTERTAINMENT

ARCHON

PINBALL CONSTRUCTION SET

MUSIC CONSTRUCTION SET

M.U.L.E.

Electronic Arts
2755 Campus Drive
San Mateo, CA 94403

Q*BERT

POPEYE

Parker Brothers
50 Durham Road
Beverly, MA 01915

POGO JOE

KEN USTON'S PROFESSIONAL BLACKJACK

Screenplay
P.O. Box 3558
Chapel Hill, NC 27514

NIGHT MISSION PINBALL

FLIGHT SIMULATOR II

Sublogic Corporation
713 Edgebrook Drive
Champaign, IL 61820

BLUE MAX
FORT APOCALYPSE
NECROMANCER
PHARAOH'S CURSE
PROTECTOR II
SHAMUS II
SURVIVOR
ZAXXON
ZEPPELIN

Synapse Software
5221 Central Avenue
Richmond, CA 94804

COMBAT LEADER
PRO TOUR GOLF

Strategic Simulations, Inc.
883 Stierlin Road, Building A-200
Mountain View, CA 94043

CENTIPEDE
DIG DUG
DONKEY KONG

Atarisoft
P.O. Box 2943
South San Francisco, CA 94080

BEACH-HEAD

Access Software
925 East 900 South
Salt Lake City, UT 84105

B.C.'S QUEST FOR TIRES

Sierra On-Line, Inc.
Sierra On-Line Building
Coarsegold, CA 93614

EDUCATIONAL SOFTWARE

THE FACTORY

Sunburst Communications
39 Washington Avenue
Pleasantville, NY 10570

TYPING TUTOR/WORD INVADERS

Academy Software
P.O. Box 9403
San Rafael, CA 94912

JOT-A-WORD

Micro-W. Distributing, Inc.
P.O. Box 113
Pompton Plains, NJ 07444

LEARNING WITH LEEPER

Sierra On-Line, Inc.
Sierra On-Line Building
Coarsegold, CA 93614

MASTERTYPE

Scarborough Systems, Inc.
25 North Broadway
Tarrytown, NY 10591

FLIGHT SIMULATOR II

Sublogic Corp.
713 Edgebrook Dr.
Champaign, IL 61820

USER INPUT DEVICES

POWERPAD

Chalk Board, Inc.
3772 Pleasantdale Road
Atlanta, GA 30340

KOALA PAD

Koala Technologies Corp.
3100 Patrick Henry Drive
Santa Clara, CA 95050

FLEXIDRAW

Inkwell Systems
7760 Vickers St.
San Diego, CA 92138

PRINTER INTERFACE

THE CONNECTION

Micro-W. Distributing, Inc.
P.O. Box 113
Pompton Plains, NJ 07444

DUAL DISK DRIVE

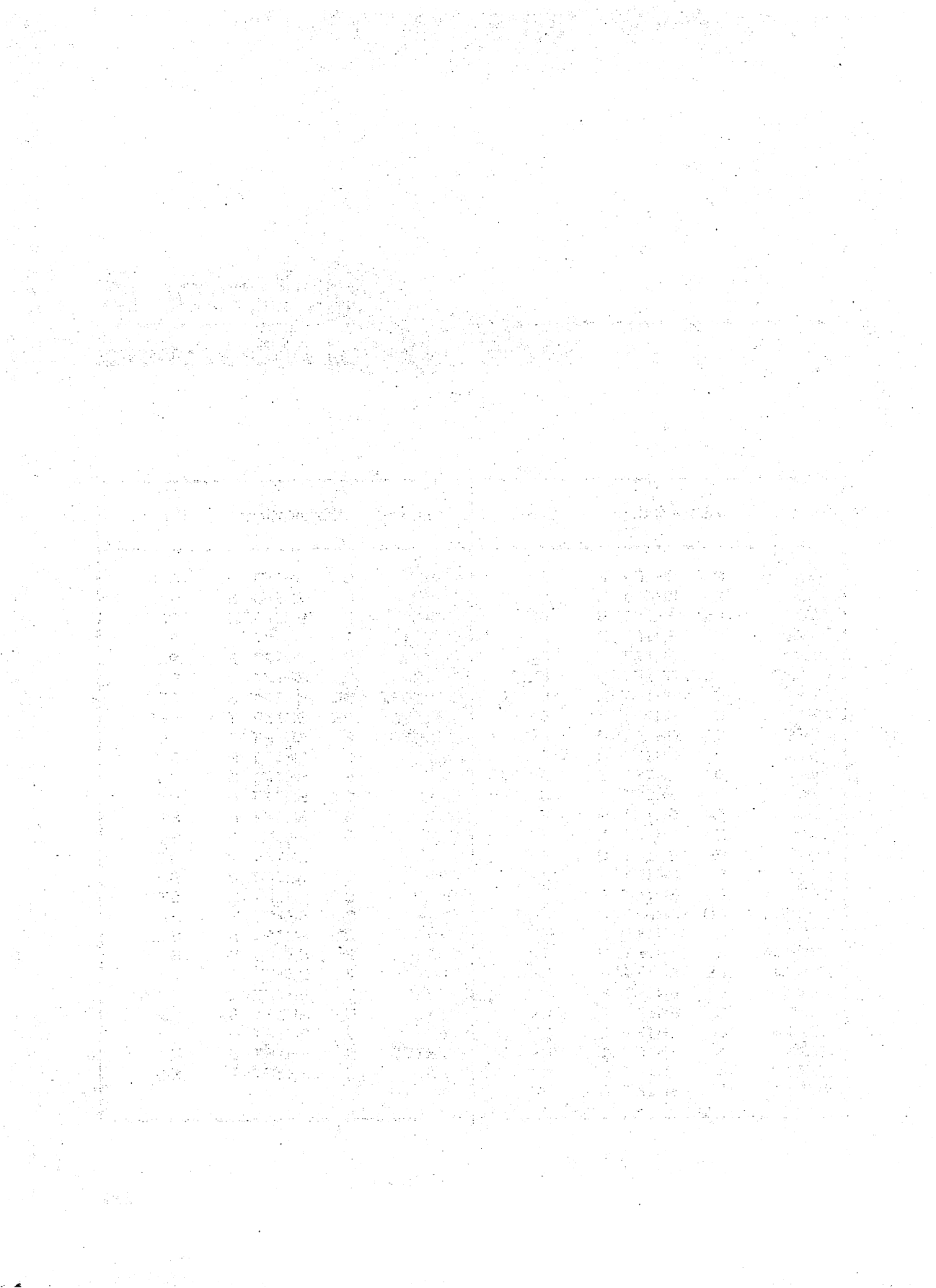
MSD SD-2

MSD Systems, Inc.
10031 Monroe Drive, Suite 206
Dallas, TX 75229

Appendix B

BASIC Keyword Abbreviations

Command	Abbreviations		Screen	Command	Abbreviations		Screen
ABS	A	SHIFT B	A I	OPEN	O	SHIFT P	O I
AND	A	SHIFT N	A /	PEEK	P	SHIFT E	P I
ASC	A	SHIFT S	A ♥	POKE	P	SHIFT O	P I
ATN	A	SHIFT T	A I	PRINT	?		?
CHR\$	C	SHIFT H	C I	PRINT#	P	SHIFT R	P _
CLOSE	CL	SHIFT O	CL I	READ	R	SHIFT E	R I
CLR	C	SHIFT L	CL	RESTORE	RE	SHIFT S	RE ♥
CMD	C	SHIFT M	C \	RETURN	RE	SHIFT T	RE I
CONT	C	SHIFT O	C I	RIGHT\$	R	SHIFT I	R \
DATA	D	SHIFT A	D ♣	RND	R	SHIFT N	R /
DEF	D	SHIFT E	D I	RUN	R	SHIFT U	R /
DIM	D	SHIFT I	D \	SAVE	S	SHIFT A	S ♣
END	E	SHIFT N	E /	SGN	S	SHIFT G	S I
EXP	E	SHIFT X	E ♣	SIN	S	SHIFT I	S \
FOR	F	SHIFT O	F I	SPC(	S	SHIFT P	S I
FRE	F	SHIFT R	F _	SQR	S	SHIFT Q	S ●
GET	G	SHIFT E	G I	STEP	ST	SHIFT E	ST I
GOSUB	GO	SHIFT S	GO ♥	STOP	S	SHIFT T	S I
GOTO	G	SHIFT O	G I	STR\$	ST	SHIFT R	ST _
INPUT#	I	SHIFT N	I /	SYS	S	SHIFT Y	S I
LEFT\$	LE	SHIFT F	LE _	TAB(	T	SHIFT A	T ♣
LET	L	SHIFT E	L I	THEN	T	SHIFT H	T I
LIST	L	SHIFT I	L \	USR	U	SHIFT S	U ♥
LOAD	L	SHIFT O	L I	VAL	V	SHIFT A	V ♣
MID\$	M	SHIFT I	M \	VERIFY	V	SHIFT E	V I
NEXT	N	SHIFT E	N I	WAIT	W	SHIFT A	W ♣
NOT	N	SHIFT O	N I				



Appendix C

Master Character Chart

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
0	00					
1	01				a	
2	02				b	
3	03				c	
4	04				d	
5	05	CTRL 2	white		e	
6	06				f	
7	07				g	
8	08		toggle off		h	
9	09		toggle on		i	
10	0A		line feed		j	
11	0B				k	
12	0C				l	
13	0D	RETURN	carriage return		m	
14	0E				n	
15	0F				o	
16	10				p	
17	11		cursor down		q	
18	12	CTRL 9	reverse on		r	
19	13		home cursor		s	
20	14		delete char.		t	
21	15				u	
22	16				v	
23	17				w	
24	18				x	
25	19				y	
26	1A				z	
27	1B					
28	1C	CTRL 3	red			
29	1D		cursor right			
30	1E	CTRL 6	green			
31	1F	CTRL 7	blue			

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
32	20	SPACE				
33	21	!	!	!		
34	22	"	"	"		
35	23	#	#	#		
36	24	\$	\$	\$		
37	25	%	%	%		
38	26	&	&	&		
39	27	'	'	'		
40	28	(	(	(		
41	29	)	)	)		
42	2A	*	*	*		
43	2B	+	+	+		
44	2C	,	,	,		
45	2D	-	-	-		
46	2E	.	.	.		
47	2F	/	/	/		
48	30	0	0	0		
49	31	1	1	1		
50	32	2	2	2		
51	33	3	3	3		
52	34	4	4	4		
53	35	5	5	5		
54	36	6	6	6		
55	37	7	7	7		
56	38	8	8	8		
57	39	9	9	9		
58	3A	:	:	:		
59	3B	;	;	;		
60	3C	<	<	<		
61	3D	=	=	=		
62	3E	>	>	>		
63	3F	?	?	?		

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
64	40	@	@	▬		
65	41	A	A	♣	a	A
66	42	B	B	▬	b	B
67	43	C	C	▬	c	C
68	44	D	D	▬	d	D
69	45	E	E	▬	e	E
70	46	F	F	▬	f	F
71	47	G	G	▬	g	G
72	48	H	H	▬	h	H
73	49	I	I	▬	i	I
74	4A	J	J	▬	j	J
75	4B	K	K	▬	k	K
76	4C	L	L	▬	l	L
77	4D	M	M	▬	m	M
78	4E	N	N	▬	n	N
79	4F	O	O	▬	o	O
80	50	P	P	▬	p	P
81	51	Q	Q	▬	q	Q
82	52	R	R	▬	r	R
83	53	S	S	▬	s	S
84	54	T	T	▬	t	T
85	55	U	U	▬	u	U
86	56	V	V	▬	v	V
87	57	W	W	▬	w	W
88	58	X	X	▬	x	X
89	59	Y	Y	▬	y	Y
90	5A	Z	Z	▬	z	Z
91	5B	£	£	▬		
92	5C	£	£	▬		
93	5D	J	J	▬		
94	5E	↑	↑	▬		
95	5F	←	←	▬		

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHR\$	POKE	CHR\$	POKE
96	60					
97	61					
98	62					
99	63					
100	64					
101	65					
102	66					
103	67					
104	68					
105	69					
106	6A					
107	6B					
108	6C					
109	6D					
110	6E					
111	6F					
112	70					
113	71					
114	72					
115	73					
116	74					
117	75					
118	76					
119	77					
120	78					
121	79					
122	7A					
123	7B					
124	7C					
125	7D					
126	7E					
127	7F					

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHR S	POKE	CHRS	POKE
128	80			Ⓔ		
129	81	Ⓒ 1	orange	A		a
130	82			B		b
131	83			C		c
132	84			D		d
133	85	F-1	function key 1	E		e
134	86	F-3	function key 3	F		f
135	87	F-5	function key 5	G		g
136	88	F-7	function key 7	H		h
137	89	SHIFT F-1	function key 2	I		i
138	8A	SHIFT F-3	function key 4	J		j
139	8B	SHIFT F-5	function key 6	K		k
140	8C	SHIFT F-7	function key 8	L		l
141	8D			M		m
142	8E			N		n
143	8F			O		o
144	90	CTRL 1	black	P		p
145	91	SHIFT Ⓒ	cursor up	Q		q
146	92	CTRL 0	reverse off	R		r
147	93	SHIFT CLR HOME	clear screen	S		s
148	94	SHIFT INST DEL	insert space	T		t
149	95	Ⓒ 2	brown	U		u
150	96	Ⓒ 3	light red	V		v
151	97	Ⓒ 4	dark gray	W		w
152	98	Ⓒ 5	medium gray	X		x
153	99	Ⓒ 6	light green	Y		y
154	9A	Ⓒ 7	light blue	Z		z
155	9B	Ⓒ 8	light gray	[		
156	9C	CTRL 5	purple	Ⓔ		
157	9D	SHIFT Ⓒ	cursor left	]		
158	9E	CTRL 8	yellow	↑		
159	9F	CTRL 4	cyan	←		

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
160	A0	SHIFT SPACE				
161	A1	K				
162	A2	I				
163	A3	T				
164	A4	@				
165	A5	G				
166	A6	+				
167	A7	M				
168	A8	£				
169	A9	SHIFT £				
170	AA	N				
171	AB	Q				
172	AC	D				
173	AD	Z				
174	AE	S				
175	AF	P				
176	B0	A				
177	B1	E				
178	B2	R				
179	B3	W				
180	B4	H				
181	B5	J				
182	B6	L				
183	B7	Y				
184	B8	U				
185	B9	O				
186	BA	SHIFT @				
187	BB	F				
188	BC	C				
189	BD	X				
190	BE	V				
191	BF	B				

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
192	C0	SHIFT *				
193	C1	SHIFT A			A	A
194	C2	SHIFT B			B	B
195	C3	SHIFT C			C	C
196	C4	SHIFT D			D	D
197	C5	SHIFT E			E	E
198	C6	SHIFT F			F	F
199	C7	SHIFT G			G	G
200	C8	SHIFT H			H	H
201	C9	SHIFT I			I	I
202	CA	SHIFT J			J	J
203	CB	SHIFT K			K	K
204	CC	SHIFT L			L	L
205	CD	SHIFT M			M	M
206	CE	SHIFT N			N	N
207	CF	SHIFT O			O	O
208	D0	SHIFT P			P	P
209	D1	SHIFT Q			Q	Q
210	D2	SHIFT R			R	R
211	D3	SHIFT S			S	S
212	D4	SHIFT T			T	T
213	D5	SHIFT U			U	U
214	D6	SHIFT V			V	V
215	D7	SHIFT W			W	W
216	D8	SHIFT X			X	X
217	D9	SHIFT Y			Y	Y
218	DA	SHIFT Z			Z	Z
219	DB	SHIFT +				
220	DC					
221	DD	SHIFT -				
222	DE	SHIFT ↑				
223	DF	*				

VALUE		PRESS KEY	UPPER CASE		LOWER CASE	
DEC	HEX		CHRS	POKE	CHRS	POKE
224	E0					
225	E1					
226	E2					
227	E3					
228	E4					
229	E5					
230	E6					
231	E7					
232	E8					
233	E9					
234	EA					
235	EB					
236	EC					
237	ED					
238	EE					
239	EF					
240	F0					
241	F1					
242	F2					
243	F3					
244	F4					
245	F5					
246	F6					
247	F7					
248	F8					
249	F9					
250	FA					
251	FB					
252	FC					
253	FD					
254	FE					
255	FF					

Appendix D

Color, Editing, and Function Key Codes

POKE 53280, X Border color 0 through 15
 POKE 53281, X Background color 0 through 15
 POKE 646, X Character color 0 through 15

Also [CTRL] and [CMDR] plus the number keys 1 through 8 may also be used to select colors.

TABLE 1

COLOR	CTRL	ASC	POKE	SYMBL	COLOR	CBM	ASC	POKE	SYMBL
BLACK	1	(144)	0		ORANGE	1	(129)	8	
WHITE	2	(5)	1		BROWN	2	(149)	9	
RED	3	(28)	2		LIGHT RED	3	(150)	10	
CYAN	4	(159)	3		GRAY 1	4	(151)	11	
PURPLE	5	(156)	4		GRAY 2	5	(152)	12	
GREEN	6	(30)	5		LIGHT GREEN	6	(153)	13	
BLUE	7	(31)	6		LIGHT BLUE	7	(154)	14	
YELLOW	8	(158)	7		GRAY 3	8	(155)	15	

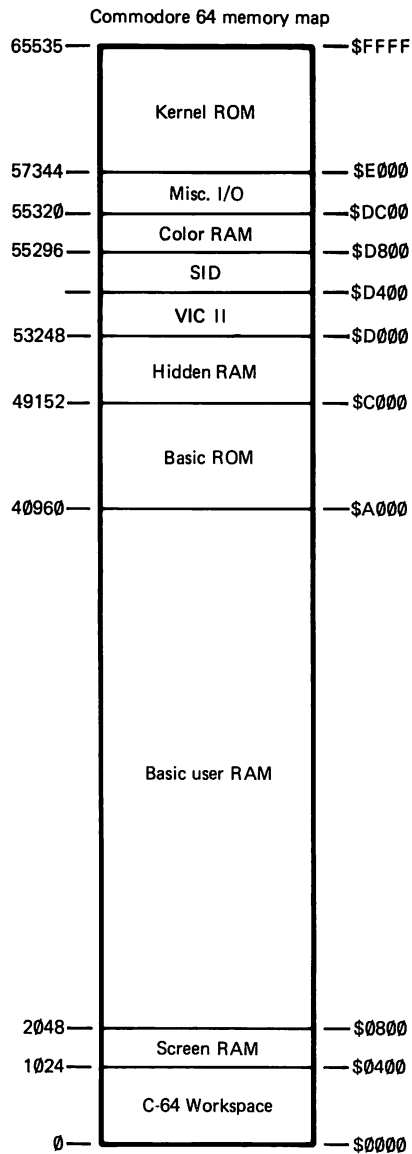
302 Color, Editing, and Function Key Codes

KEY	ASC	SYMBOL	KEY	ASC	SYMBOL
HOME	(19)		FN1	(133)	
CLR	(147)		FN2	(137)	
CRSR DOWN	(17)		FN3	(134)	
CRSR UP	(145)		FN4	(138)	
CRSR RIGHT	(29)		FN5	(135)	
CRSR LEFT	(157)		FN6	(139)	
RVS ON	(18)		FN7	(136)	
RVS OFF	(146)		FN8	(140)	

Each function key has an ASCII value and a unique graphic symbol that may be used in program control.

Appendix E

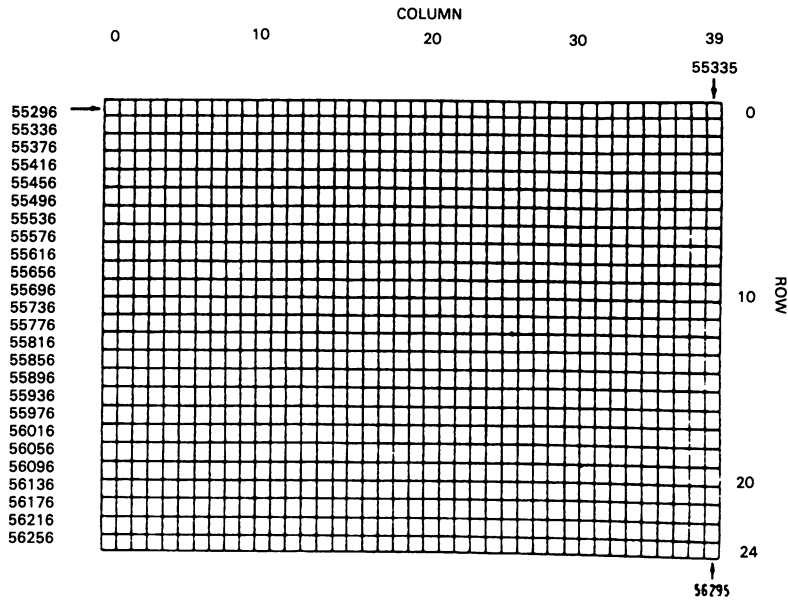
Commodore 64 Memory Map



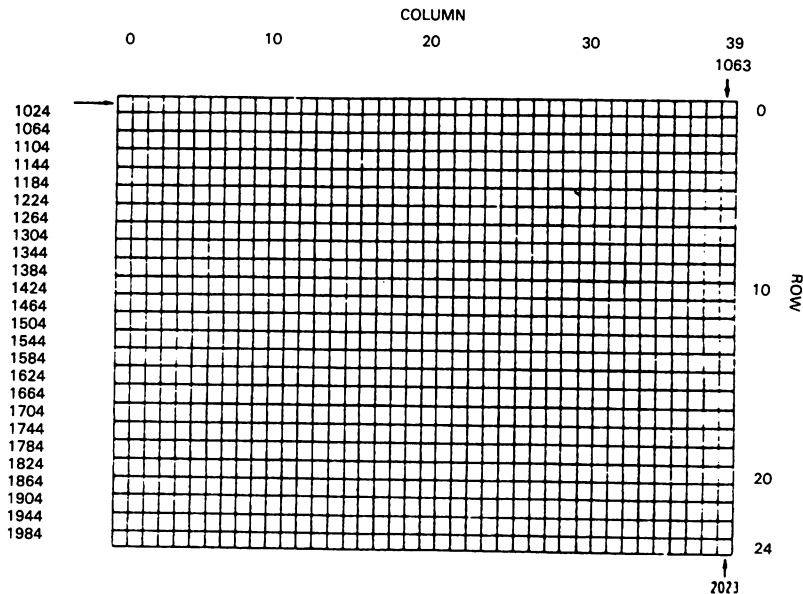
Appendix F

Screen and Color Memory Maps

COLOR MEMORY MAP



SCREEN MEMORY MAP



Appendix G

VIC-II Register Map

VIC-II REGISTER MAP

MEMORY	LOCATION	DESCRIPTION
V	— 53248	Sprite 0 X Position
V+1	— 53249	Sprite 0 Y Position
V+2	— 53250	Sprite 1 X Position
V+3	— 53251	Sprite 1 Y Position
V+4	— 53252	Sprite 2 X Position
V+5	— 53253	Sprite 2 Y Position
V+6	— 53254	Sprite 3 X Position
V+7	— 53255	Sprite 3 Y Position
V+8	— 53256	Sprite 4 X Position
V+9	— 53257	Sprite 4 Y Position
V+10	— 53258	Sprite 5 X Position
V+11	— 53259	Sprite 5 Y Position
V+12	— 53260	Sprite 6 X Position
V+13	— 53261	Sprite 6 Y Position
V+14	— 53262	Sprite 7 X Position
V+15	— 53263	Sprite 7 Y Position
V+16	— 53264	Sprites 0-7 Max. X Position
V+17	— 53265	VIC II Control register #1
V+18	— 53266	Read/Write Raster
V+19	— 53267	Light-Pen Latch X Position
V+20	— 53268	Light-Pen latch Y Position
V+21	— 53269	Sprites 0-7 Display Register
V+22	— 53270	VIC II Control Register #2
V+23	— 53271	Sprites 0-7 Expand Y Register
V+24	— 53272	VIC II Memory Control Register
V+25	— 53273	VIC II Interrupt Flag Register
V+26	— 53274	IRQ Mask Register

V+27	—	53275	Sprites 0-7 Display Priority
V+28	—	53276	Sprites 0-7 Multi-Color Mode Register
V+29	—	53277	Sprites 0-7 Expand X Register
V+30	—	53278	Sprite to Sprite Collision Detect
V+31	—	53279	Sprite to Character Collision Detect
V+32	—	53280	Border Color
V+33	—	53281	Background Color 0
V+34	—	53282	Background Color 1
V+35	—	53283	Background Color 2
V+36	—	53284	Background Color 3
V+37	—	53285	Sprites 0-7 Multi-Color Register 0
V+38	—	53286	Sprites 0-7 Multi-Color Register 1
V+39	—	53287	Sprite 0 Color
V+40	—	53288	Sprite 1 Color
V+41	—	53289	Sprite 2 Color
V+42	—	53290	Sprite 3 Color
V+43	—	53291	Sprite 4 Color
V+44	—	53292	Sprite 5 Color
V+45	—	53293	Sprite 6 Color
V+46	—	53294	Sprite 7 Color

Appendix H

Sound Interface Device

ADSR AND WAVEFORM CONTROL SETTINGS

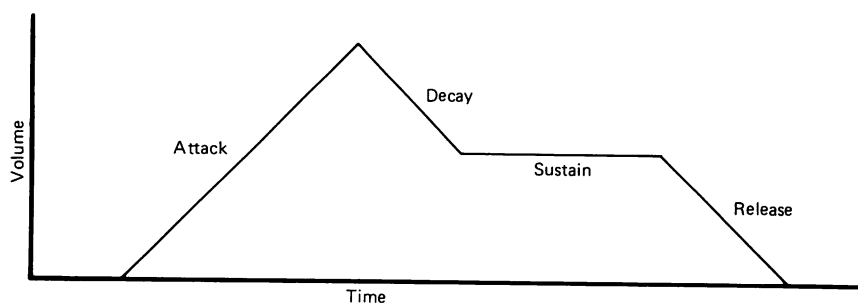
	LOCATION	TRIANGLE	SAWTOOTH	PULSE	NOISE
VOICE 1	S+4 — 54276	17/16	33/32	65/64	129/128
VOICE 2	S+11 — 54283	17/16	33/32	65/64	129/128
VOICE 3	S+18 — 54290	17/16	33/32	65/64	129/128

ATTACK/DECAY RATE SETTINGS

	LOCATION	HIGH ATTACK	MEDIUM ATTACK	LOW ATTACK	LOWEST ATTACK	HIGH DECAY	MEDIUM DECAY	LOW DECAY	LOWEST DECAY
VOICE 1	S+5 — 54277	128	64	32	16	8	4	2	1
VOICE 2	S+12 — 54284	128	64	32	16	8	4	2	1
VOICE 3	S+19 — 54291	128	64	32	16	8	4	2	1

SUSTAIN/RELEASE SETTINGS

	LOCATION	HIGH SUSTAIN	MEDIUM SUSTAIN	LOW SUSTAIN	LOWEST SUSTAIN	HIGH RELEASE	MEDIUM RELEASE	LOW RELEASE	LOWEST RELEASE
VOICE 1	S+6 — 54278	128	64	32	16	8	4	2	1
VOICE 2	S+13 — 54285	128	64	32	16	8	4	2	1
VOICE 3	S+20 — 54292	128	64	32	16	8	4	2	1



SID REGISTER MAP

MEMORY	LOCATION	DESCRIPTION
VOICE #1		
S	— 54272	Lower half of tone frequency value
S+1	— 54273	Upper half of tone frequency value
S+2	— 54274	Lower half of pulse-width value
S+3	— 54275	Upper half of pulse-width value
S+4	— 54276	Waveform control register
S+5	— 54277	Attack and Decay control register
S+6	— 54278	Sustain and Release control register
VOICE #2		
S+7	— 54279	Lower half of tone frequency value
S+8	— 54280	Upper half of tone frequency value
S+9	— 54281	Lower half of pulse-width value
S+10	— 54282	Upper half of pulse-width value
S+11	— 54283	Waveform control register
S+12	— 54284	Attack and Decay control register
S+13	— 54285	Sustain and Release control register
VOICE #3		
S+14	— 54286	Lower half of tone frequency value
S+15	— 54287	Upper half of tone frequency value
S+16	— 54288	Lower half of pulse-width value
S+17	— 54289	Upper half of pulse-width value
S+18	— 54290	Waveform control register
S+19	— 54291	Attack and Decay control register
S+20	— 54292	Sustain and Release control register
SOUND FILTERS		
S+21	— 54293	Lower half of cutoff filter value
S+22	— 54294	Upper half of cutoff filter value
S+23	— 54295	Resonance and sound filter values
S+24	— 54296	Mode and volume controls

MUSIC NOTE VALUES

MUSICAL NOTE		OSCILLATOR FREQ.		
NOTE	OCTAVE	DECIMAL	HI	LOW
0	C-0	268	1	12
1	C#-0	284	1	28
2	D-0	301	1	45
3	D#-0	318	1	62
4	E-0	337	1	81
5	F-0	358	1	102
6	F#-0	379	1	123
7	G-0	401	1	145
8	G#-0	425	1	169
9	A-0	451	1	195
10	A#-0	477	1	221
11	B-0	506	1	250
16	C-1	536	2	24
17	C#-1	568	2	56
18	D-1	602	2	90
19	D#-1	637	2	125
20	E-1	675	2	163
21	F-1	716	2	204
22	F#-1	758	2	246
23	G-1	803	3	35
24	G#-1	851	3	83
25	A-1	902	3	134
26	A#-1	955	3	187
27	B-1	1012	3	244
32	C-2	1072	4	48
33	C#-2	1136	4	112
34	D-2	1204	4	180
35	D#-2	1275	4	251
36	E-2	1351	5	71
37	F-2	1432	5	152
38	F#-2	1517	5	237
39	G-2	1607	6	71
40	G#-2	1703	6	167
41	A-2	1804	7	12
42	A#-2	1911	7	119
43	B-2	2025	7	233
48	C-3	2145	8	97
49	C#-3	2273	8	225
50	D-3	2408	9	104
51	D#-3	2551	9	247

52	E-3	2703	10	143
53	F-3	2864	11	48
54	F#-3	3034	11	218
55	G-3	3215	12	143
56	G#-3	3406	13	78
57	A-3	3608	14	24
58	A#-3	3823	14	239
59	B-3	4050	15	210
64	C-4	4291	16	195
65	C#-4	4547	17	195
66	D-4	4817	18	209
67	D#-4	5103	19	239
68	E-4	5407	21	31
69	F-4	5728	22	96
70	F#-4	6069	23	181
71	G-4	6430	25	30
72	G#-4	6812	26	156
73	A-4	7217	28	49
74	A#-4	7647	29	223
75	B-4	8101	31	165
80	C-5	8583	33	135
81	C#-5	9094	35	134
82	D-5	9634	37	162
83	D#-5	10207	39	223
84	E-5	10814	42	62
85	F-5	11457	44	193
86	F#-5	12139	47	107
87	G-5	12860	50	60
88	G#-5	13625	53	57
89	A-5	14435	56	99
90	A#-5	15294	59	190
91	B-5	16203	63	75
96	C-6	17167	67	15
97	C#-6	18188	71	12
98	D-6	19269	75	69
99	D#-6	20415	79	191
100	E-6	21629	84	125
101	F-6	22915	89	131
102	F#-6	24278	94	214
103	G-6	25721	100	121
104	G#-6	27251	106	115
105	A-6	28871	112	199

106	A#-6	30588	119	124
107	B-6	32407	126	151
112	C-7	34334	134	30
113	C#-7	36376	142	24
114	D-7	38539	150	139
115	D#-7	40830	159	126
116	E-7	43258	168	250
117	F-7	45830	179	6
118	F#-7	48556	189	172
119	G-7	51443	200	243
120	G#-7	54502	212	230
121	A-7	57743	225	143
122	A#-7	61176	238	248
123	B-7	64814	253	46

GLOSSARY

absolute load Transfer of a program from a peripheral storage device to a particular memory location of the computer. Most machine language programs must be loaded in this way.

address A name or number that identifies an exact storage location in the memory of the computer.

ADSR Attack, Decay, Sustain, and Release; four aspects that describe the quality of a sound produced by the Commodore 64's sound chip.

alphanumeric Data that is made up of letters of the alphabet, punctuation marks, and numerals.

applications software (or applications program) Software (one or more programs) designed to solve a particular problem or accomplish a particular task or set of tasks.

ASCII codes A code system for representing characters as numbers; ASCII stands for American Standard Code for Information Interchange.

assembler A program that translates assembly language instructions (mnemonic codes) into machine code.

assembly language A low-level language that uses mnemonic codes to represent machine language instructions.

auto relocater A capability built into the C-64 that allows the user to load BASIC programs anywhere in the BASIC RAM area. Most other computers load BASIC programs only at the predefined beginning of the BASIC RAM area. This feature is particularly useful when appending programs.

auto-answer A feature of a modem that allows it to receive input via telephone automatically, without the user having to answer the phone.

auto-dial A feature of a modem that allows it to dial and send information via telephone lines to a distant computer automatically, without human intervention.

back up To duplicate a program or file to a second storage medium to ensure against accidental loss; a copy made for this purpose.

BASIC Beginner's All-purpose Symbolic Instruction Code; an easy-to-use, high-level programming language built into many microcomputers, including the C-64.

binary number A number in the base 2 number system; binary numbers are made of the digits 0 and 1 and express numbers as the sum of successive powers of 2.

bit An acronym for Binary digIT. A digit (0 or 1) in the binary number system; the smallest unit of information that can be stored in the computer.

buffer An area of memory storage used to temporarily hold data being transferred from one piece of hardware to another. Some printers and printer interfaces have a buffer that can store text to be printed.

bus Paths between various parts of the computer for the transfer of information.

byte A group of 8 bits which can represent one character or value.

cathode ray tube (CRT) The screen element of a television receiver or computer monitor.

central processing unit (CPU) The part of the computer that fetches, decodes and processes data, and transmits it to the peripherals.

channel (communication channel) A path for the transmission of data

between the CPU and a peripheral device. Before we can send data to a printer, for example, we must first open a channel to the printer.

chip An integrated circuit created on a tiny piece of silicon; the paths and gates of the circuit are made of very thin films of conductive metal.

color keys The numeric keys of the C-64, which, when used with the [CTRL] or [CBM] key access the 16 colors available with the C-64.

Comal A high-level programming language used primarily in Europe; claimed to be faster and more efficient than BASIC.

compiler A program that translates a program written in a high-level language to machine code. By eliminating the need for a translator, compiled programs run faster than those written in BASIC, for example.

control keys The keys of the C-64 that control specific functions; for example, [RVS ON], [RVS OFF], [SHIFT], [SHIFT LOCK], [CBM], [CTRL], [?] (as a short form of PRINT), [RETURN], [RUN/STOP], and [RESTORE].

controller An external device that makes it possible to enter input without the use of the keyboard; for example, the manual controllers — joystick, game paddles, and track balls.

CP/M Acronym for Control Program for Microcomputers; standardized, machine-independent operating system developed for the Z-80 microprocessor; heavily used for business-oriented software: available in cartridge form for the C-64.

CPU See central processing unit.

CRT See cathode ray tube.

cursor A flashing indicator on the CRT screen that shows where the next character entered from the keyboard will appear.

data Any information input to a computer system.

data base A collection of related information on a given subject. A book or recipe collection, for example, may be the subject of a data base. Data base programs organize your data so that information can be retrieved or sorted (numerically or alphabetically) very rapidly.

data disk A disk on which the data output from a program such as a word processor or data base is stored; compare program disk.

debug To find and correct “bugs,” or errors, in programs.

delimiter A character, such as a slash or comma, used to mark the beginning and end of a group of characters.

demodulator A device that converts analog data to digital data; for example, the part of a modem that converts a signal transmitted over telephone lines to electronic signals that a computer can use.

direct mode The mode of operation of the C-64 in which the user types in commands and gets an immediate result; compare program mode.

directory A listing of all the files (program and data) on a disk; roughly similar to a book's table of contents.

directory header An identifier for a diskette; made up of the name and ID number of the diskette; stored on track 18.

disassembler A program that allows the user to view and change the instructions that make up a machine language program; operates by converting machine code to mnemonic code and displaying the mnemonic code listing.

disk (magnetic disk) A flat, circular storage medium that is coated with a magnetic material on which data can be recorded and read by the head of a disk drive unit; also called a diskette.

disk drive A peripheral hardware unit that reads data stored on a disk and also writes data to the disk.

disk operating system (DOS) An operating system built into the 1541 disk drive that regulates disk drive functions. Among other things, it keeps track of where files are located on a disk.

diskette See disk.

DOS See disk operating system.

double-density Characterizing a disk or other magnetic storage medium designed to store twice as much information as the usual single-density disk; compare single-density.

downloading The process by which a computer stores information transmitted over telephone lines by another computer.

dumb Not having a microprocessor for controlling tasks. For example, dumb terminal program, dumb printer interface; compare smart.

dynamic keyboard editing keys The keys of the C-64 that are used for inserting and deleting text and moving the cursor around the screen; for example, the [CRSR] keys, [CLR/HOME], and [INST/DEL].

execute To interpret an instruction or a program of instructions and perform the operation(s) indicated.

field A piece of information within a record; for example, name, street address, city and state, and zip code might be fields within a mailing label record.

file A collection of related records stored together and accessed under a single name; the C-64 uses program files and three kinds of data files — sequential, relative, and user.

flag An indicator for the achievement or nonachievement of a condition.

floppy disk A flexible disk, usually made of Mylar coated with a material that can be magnetized as a way of storing bits of data; the standard size for floppy disks for the C-64 is 5-¼ inches.

formatting (headering, newing) A procedure carried out when a disk is first used; the process titles the disk, defines the sectors, and establishes the BAM (block allocation map) and the directory (table of contents).

Forth A compact, highly structured language somewhat like a high-level assembly language.

friction feed The technique of moving paper through a printer by friction; for example, a standard electric typewriter uses friction feed, holding the paper between the rollers and the platen; see also tractor feed.

full-duplex Designating transmission of information over a line in both directions at once; for example, from a terminal to a computer and vice versa; compare half-duplex.

function keys The four keys at the far right of the C-64 keyboard (F1-F2, F3-F4, F5-F6, and F7-F8); these keys have no assigned functions; instead, they may be programmed with functions of particular use to the operator, and they may be assigned different functions in different programs.

graphic characters Lines, curves, triangles, squares and other displays accessible as keyboard characters; the C-64 has over 60 graphic characters and also allows for user-designed characters; the standard graphic characters are marked on the fronts of the keys.

half-duplex Designating a transmission of information over a line in only one direction at a time; compare full-duplex.

hard copy Computer output printed on paper.

hard disk drive A unit housing a mass storage disk made of rigid material, such as aluminum or ceramic, coated with a magnetic mate-

rial; a hard disk can typically store 3 to 10 million characters (or bytes of information).

hard-sectored Characterizing a floppy disk in which holes have been punched to mark the boundaries of each sector; compare soft-sectored.

hardware The physical components that make up a computer system.

hexadecimal (hex) A base 16 number system; numerals include the digits 0-9 and the letters A-F.

hub ring A device, analogous to a paper reinforcement ring, that is attached to the edge of the hole in the center of a floppy disk to make it easier for the disk drive to grasp and rotate the disk.

hybrid file A combination of a relative and a sequential file; sometimes used in data base programs to allow access to fields by name rather than number.

IEEE bus A group of wires that transmit data or electronic signals between two devices; standardized by the Institute of Electrical and Electronic Engineers.

input Data entered or to be entered into the computer.

interface A hardware device that allows interaction between two devices that handle data in different ways; for example, a device that connects the C-64 with a non-Commodore printer; also, to connect two devices with such a device.

interpreter A program usually built into the computer that translates high-level language instructions, such as those written in BASIC, into machine code.

joystick A manual control device with a stick or lever that can be moved or tilted in various directions to move the cursor or a graphic element around the CRT screen.

justification Vertical alignment at the left or right margin or both.

K See kilobyte.

kernel The operating system of the C-64.

keyboard buffer A storage area in the computer that stores characters entered via the keyboard.

kilobyte (K) 1024 (2 raised to the 10th power) bytes of data or memory space. The Commodore 64 can store 64K or 65,536 bytes of data.

line feed A mechanism or electronic signal that causes paper in a printer to advance one line.

low byte/high byte format A way of indicating a memory address or other number greater than 255 using two bytes of memory; the first byte (the byte stored in the memory location with the lower address) expresses the lower portion of the number; the second byte expresses the higher portion; for example, the hex number 1850 expressed in low byte/high byte format might be 50 (low byte), stored in location 45 and 18 (high byte) stored in location 46.

luminance output The C-64 output signal that transmits the screen picture to a monochrome monitor.

machine language Programming language made up of a set of machine codes composed of 1s and 0s that represent the ON and OFF states of the “switches,” or gates, within the computer.

machine language monitor A utility program that allows the user to examine and modify the contents of the memory locations of a computer.

mass storage A device, external to the CPU, that is capable of storing large amounts of data.

memory A device or the capacity to store data for later retrieval.

memory map A chart that shows diagrammatically the important memory areas of the computer.

microcomputer Any small computer that uses a microprocessor as its “brain”; also called personal computer or home computer.

microprocessor A chip that contains the “brain” of the computer; that is, the device that controls the flow of data in the computer and allows the computer to perform logical and arithmetic operations on data.

mnemonics Three-letter symbolic codes that represent machine language instructions; example, LDA #\$01.

modem (modulator/demodulator) A device that converts digital into analog signals for the purpose of transmitting data from one computer to another, and analog signals into digital signals for the purpose of receiving information.

modem, acoustic A modem that emits and receives sound signals; attaches to a telephone in place of the handset.

modem, direct-connect A modem that emits and receives the kind of electronic signals used in the transfer of telephone information; attaches to the telephone line without displacing the handset.

modulator A device that converts digital to analog signals; for exam-

ple, that part of a modem that converts machine code used by a computer into a signal to be transmitted over a telephone line.

mouse A small, hand-held device that can be moved over a surface to cause the cursor to move to a corresponding location on the screen.

new See formatting. Also a BASIC command that erases a BASIC program from the computer's memory.

nibble Half a byte; four bits.

numeric keys The number keys, on the top row of the C-64 keyboard.

on-line Designating equipment that is under the control of an active computer.

operating system A group of programs that control the overall operation of the computer system.

operation keys The keys of the C-64 that access the subroutines necessary to perform mathematical calculations; for example, +, -, *, /, !, and pi.

output Data that results from computer processing of input data.

overwrite To record over a previously recorded program, thus erasing the first program.

paddle, game A manual control device with a knob that can be rotated to control the movement of the cursor on the video screen.

parallel cable A cable over which all bits of a byte can be transmitted simultaneously; a parallel cable transmits data faster than a serial cable; compare serial cable.

Pascal A highly structured, high-level programming language, often used in large, complex business-oriented programs.

peripheral A device, usually for input or output, connected to a computer and under its control to some degree; for example, a printer, disk drive, or monitor.

Pilot A high-level programming language designed to help teachers create Computer-Assisted Instruction (CAI) on all subjects.

pixel The smallest unit of the video screen.

pointer The contents of a certain storage or memory location that indicates where in external storage or computer memory other information is to be found.

potentiometer A device for controlling or changing an electromotive force.

power strip A group of electrical outlets mounted in a single housing and controlled with a single ON-OFF switch; a power strip allows the user to turn on the C-64 and its peripherals at the same time, making it unnecessary to remember the order in which to turn equipment on and off.

printer A peripheral output device for producing hard copy.

process To perform a series of operations to obtain specified results.

programs A logically arranged set of instructions (programming statements) to enable it to perform a particular operation; to write such a set of instructions.

program disk A disk on which a program is recorded; the program may call for files from another disk, the data disk.

program mode The mode of operation of the C-64 in which the user uses the keyboard to enter instructions for later execution; compare direct mode.

proportional spacing A system for placement of printed characters on a line, in which different amounts of space are allowed for letters of different widths.

quote mode A "submode" of both direct and program modes; in direct mode, used to define strings to be PRINTed; in program mode, used to define cursor movement, screen scrolling, display color, and strings to be printed within a program.

RAM See random access memory.

random access Direct access to information at any location on disk; that is, nonsequential data retrieval; compare sequential access.

random-access memory (RAM) A type of computer memory in which records are accessed directly by their addresses; data can be read from or written to RAM by the user; data stored in RAM must be stored/saved to peripheral storage before the computer is turned off or it will be lost; compare ROM.

random seed A value that provokes the random number generator built into the C-64 to produce more random results than without it.

read-only memory (ROM) A type of memory whose contents are permanently built into the computer; the user cannot change the contents of ROM; compare RAM.

record A group of logically related fields of information treated as a unit; a file may be made up of many records.

registers, memory Storage locations in the CPU that hold data to be operated on.

relative file A coherent group of data stored on a mass storage medium. Data can be accessed much more quickly than data stored in sequential files. Relative files are more efficient than sequential files.

ROM See read only memory.

routine A set of instructions for solving a particular problem.

screen editor A program built into the C-64 that provides the capability of editing data as it is displayed on the screen. The C-64 has one of the most advanced screen editors available for home computers.

scroll The movement of text vertically or horizontally on the video screen.

sector A portion of a track on a disk; sectors are numbered and can hold 255 bits of information.

sequential access Access to data in the order in which it was stored; for example, data stored on tape is retrieved by sequential access; compare random access.

sequential file A coherent group of data stored sequentially on a mass storage medium. Sequential files can hold data in a compact way, but access to this data has certain restrictions. Access to data within a sequential file is much slower than for a relative file.

serial bus An electrical circuit used for transmitting bytes of data one bit after another; compare parallel cable.

single-density Designating a disk or other magnetic storage medium that stores data at the standard concentration of bits per sector; compare double-density.

single-sided Designating a disk that allows data to be recorded on only one side; compare double-sided.

smart Having a microprocessor for controlling tasks. For example smart terminal program, smart printer interface; compare dumb.

soft reset A way of resetting, or regaining control of the CPU, without turning the machine off, which would destroy all programs and data in memory.

soft-sectored Designating a disk whose sector locations are established by the DOS when disks are formatted; compare hard-sectored.

software Programs to be executed by a computer.

sprite A high-resolution programmable graphic figure that can be changed and animated through BASIC.

symbol keys All the nonalphanumeric and nongraphic keys of the C-64.

systems software Programs, usually built into the computer or supplied separately by the computer manufacturer, that control the execution of applications programs; for example, operating systems, compilers and assemblers.

telecomputing Information transfer from one computer to another over telephone lines.

terminal software A program that temporarily converts a computer into an input and output peripheral of a remote computer: usually used with a modem.

text keys The keys of the C-64 that are equivalent to the standard typewriter keys used to produce nonnumeric text; for example, the alphabetic, punctuation, [SHIFT], [SHIFT LOCK], and [RETURN] keys and [SPACE].

tractor feed The method of feeding paper through a printer by means of sprockets, or pins, that fit through holes in the edges of the paper.

track Any of the concentric recording paths on a disk; each track is divided into sectors.

track ball A manual control device that consists of a sphere mounted in a box so that it can be rotated to move the cursor or other graphic element to any position on the screen.

uploading The process by which a computer sends information over telephone lines to another computer; also, the process by which a CPU accesses information from a mass storage device and stores it in its memory.

upper-case/graphics mode The mode of keyboard operation of the C-64 in which the upper-case letters, two sets of graphic characters, and both symbols on two-symbol key faces can be accessed.

upper/lower-case mode The mode of keyboard operation of the C-64 in which the upper-case and lower-case letters, the graphic characters marked on the left side of the key fronts, and both symbols on two-symbol key faces can be accessed.

user file A rarely used form of data stored on a mass storage medium. The structure of a user file is determined through programming.

utility program A program that enhances the computer's built-in software.

video output The output signal from the C-64 that transmits the screen picture to a color monitor.

wave forms The shapes of various sound waves that can be produced by the C-64; triangle, sawtooth, variable pulse, and noise.

write-protected Characterizing a diskette that has been guarded against overwriting, which destroys stored data; write protection is provided by placing a piece of opaque tape over the notch of the diskette jacket.

- Abbreviations for BASIC
- ABS, 160
- Absolute load 75, 315
- Accumulator 201
- Address 178, 180, 315
- ADSR 168, 169-171
- Appending programs 247-250
- Applications programs 315
 - (see also specific applications)
- Arithmetic operations 109
- ASCII codes 96, 133
- Assembler program 15, 316
- Assembly language 14, 15, 316
- Asterisk, for pattern matching 79-80
- Audio-video socket 32
- AUTO 227-228
- Auto relocater 316
- Automatic line numbering 227-228
- AUTOMODEM 95, 98, 100

- Backup 58, 73, 74, 269-270, 316
- BAM (see Block availability map)
- BASIC 15, 48, 205-211
 - 163-167
- BASIC interpreter 15, 201
- BASIC programming 115-131,
- Baud 98
- Binary code 20, 207, 316
- Bit 20, 98, 316
- Blank screen 250
- Block availability map 69, 184
- Buffers 90, 197, 203, 206, 316
- Bulletin board services 94
- Bus 6, 64, 316
- Business programs 13
- Byte 20, 316

- C-64 4, 6, 12, 32
- Cable connections 27, 37-40
- Calculations, mathematical 109
- Calculators, differences from computers 11
- Carriage return 48
- Cartridges 54
- Cassette buffer 197, 203, 206
- Cathode ray tube 6, 316
- CBM key 48
- Central processing unit 4, 316
- CHANGE 233-234
- Character chart 293-300
- Character sets, switching 251
- CHR\$ 138-140, 186-187
- Chip, microprocessor 4-5
- Clearing screen 250
- CLEAR/HOME 50
- Clock, built-in 113, 264-265
- CLOSE 136

- CMD 136-137
- Codes 96, 133, 139, 301-302
- Colon (:) 123, 251-254
- Color 45-46, 113, 193-197, 271-272
- Comal 317
- Command channel 185
- Commands, short form BASIC 79, 107, 125
- Commodore Business Machines (CBM) 3, 5
- Compatibility, printer 92
- Compilers 16, 317
- CONT 116
- Control codes 127-129
- Control keys 45, 47
- Controllers 8, 317
- CP/M 317
- CPU (see central processing unit)
- CRT (see cathode ray tube)
- CTRL key 45, 48
- Cursor 29, 49-50, 105, 111, 317

- Data 317
- Data processing 5
- DATA statements 197-198
- Data bases 9, 189-190, 317
- Datassette, using 33, 54-59
- Debugging 215
- DELETE 50, 229
- Deleting disk files 273-274
- Direct mode 105-113
- Directory, diskette 68, 77, 137, 274
- Dirt, effect on components 25
- Disassembler 201-203, 239-240
- Disk, floppy (see diskette)
- Disk, hard 62
- Disk drive 61-65, 78-79, 82-84
- Disk operating system 65
- Disk status 222
- Diskette, floppy 7, 25, 54, 66-79, 275-276
- Dot matrix printer 85-88
- Downloading 97

- Editing 107, 108
- Editor, screen 117, 118-127
- Educational programs 13
- Electronic mail 94
- Electronic publishing 94
- Error channel 77, 187
- Error messages 77
- Executing a program 116

- Files 70-72, 78, 175-190, 274-275
- FIND 231-233

- Flag 160
- Formatting a diskette 76
- FOR/NEXT loop 166
- Function keys 47, 256

- Games 12
- GET 158-159, 256-258
- Hardware 4-10, 11
- Hexadecimal numbers 199-202
- HOME 50
- Home position 30
- Hub rings 67
- ID, diskette 73
- Impact Printer 86-88
- Initializing disk drive 78, 222
- Input, alphanumeric 5
- INSERT 51
- Interface 6, 91-92
- Interference, controlling, in screen display 33-34
- Interpreters 15

- Joystick 8, 258-259

- K (see kilobyte)
- Kernel 2, 29-30, 276-277
- Keyboard 6, 41-51, 158, 254-255
- Keys 42-49
- KILL 234-235
- Kilobyte 20

- Language 14, 15, 191-211
- Languages available for the C-64 14-17
- Letter-quality printer 86, 87-88
- Light pen 9
- Line duplication 122
- Line numbers 115
- LIST 115, 255
- Listing a program 58, 115
- LOAD 56-57, 74-75
- Loading 53, 56-57, 74-75
- Low byte/high byte 186, 207-208
- Luminance pin 38

- Machine language 14, 191-211, 215
- Magazines 281-282
- Magnetic fields 25
- Memory 6, 29, 241-242
- Microcomputer 4-5
- Microprocessor 4-5, 18, 61
- Mnemonics 198, 203
- Modem 9, 25, 94-95, 98-101
- Modes of operation 105-129
- Modulator/demodulator 94
- Monitor 6, 36-40

- Mouse 8
- Multicolor mode 148
- Multiple fields 187-189
- Music 163-167
- Nibble 20
- Notes 165, 166, 168, 169
- Number 15, 195, 260-261
- NUMBER 229-231
- OPEN 136
- Operating systems 12, 18
- Operations, arithmetic 109
- Outlet, grounded electrical 29
- Output 5
- Paddles 8
- Parity 98
- Pattern matching 79-80
- PEEK 191-197
- PEEKing the keyboard 158-159
- Peripherals 6
- Pi 113
- Pilot 16-17
- Pixel 147
- Pointers 69, 184, 207, 248-250
- POKE 30, 192-197
- Ports 26-27
- POWER PLUS 215-245
- Power strip 26
- Power supply 29, 32
- Powering up 28
- PRINT 48, 107-109
- Print 87-90
- Printer disassembler 242
- Printers 7-8, 85-92, 133-143
- Printing, bi-directional 89
- Programs 11-13, 15, 16, 53-54, 78, 115, 247-250, 282-290
- Quote mode (submode) 105, 107-108, 110, 121, 127-131
- RAM 19-21
- Random-access memory (see RAM)
- Read-only memory (see ROM)
- Register display 242
- Relative files 71-72, 183-189
- REM statements 260
- Remarks 260
- Renaming 78, 224
- Renumbering program lines 229-231
- Repeat keys 192-193
- Reset 49, 203, 222, 261-264
- RESTORE 49
- RETURN 51
- ROM 19, 21, 29-30, 54
- RUN 116
- Running a program 116
- RUN/STOP 49
- SAVE 55, 75-76
- Save and replace 77
- Scratching a file 78, 223
- Screen, clearing 50
- Screen display 33-40, 106-107, 116-118
- Screen editor 80, 117-122
- Screen memory locations 30-31, 195-196, 201
- Secondary address 136
- Sequential files 71, 176-183
- SHIFT 51
- SHIFT LOCK 48
- SHIFT/RETURN 107, 108, 127
- SID (sound interface device) 163, 165, 166, 309-313
- Smoke 25
- Software (see programs)
- Sound 163-172
- SPACE, shifted 51
- SPC 140-142
- Speech synthesizers 9-10
- Sprite editor 145-152
- Sprites 145-162
- STOP, disabling 255
- Stopping a program 116
- STOP/RESTORE, disabling 255
- Storage, mass data 7
- Strings 107-108, 111
- SUPERBASIC 152, 215, 219, 227-234
- SUPERDOS 80-81, 215, 217-225
- SUPERMON 199-205, 215, 219, 237-243, 244
- SUPPORT DISK 145-147, 152-153, 164, 199, 201-202, 205-206, 215, 217-218, 267-277
- Surge protection 26
- Switch, reset 262-264
- Syntax 115
- SYNTAX ERROR 106-107, 123
- SYS 198, 207
- TAB 140-142
- Tapes 25, 53-59
- Telecomputing 93-94, 96-97
- Television as a monitor 33-34
- Terminal, C-64 as 95-96
- Terminal programs 14, 94-98
- Text, inserting and deleting 50-51
- TI 264
- Time clocks 264-265
- TIS 113, 264-265
- Track balls 8
- Typewriter, electronic 86-87
- Typing, to enter programs 54
- Unscratching files 274-275
- Uploading information 97
- User files 72
- User input devices 289-290
- Utility programs 13
- Validating diskettes 79, 224
- Variables 109-111
- VERIFY 56
- VIC II chip 145, 154, 156-161
- VICMODEM 95, 98, 99-100
- Voltage surges and spikes 26
- Waveforms of sound 167-169
- WEDGE, C-64 215, 217
- Wild cards 80
- Word length 98
- Word processors 175
- Work area, planning and organizing 25

\$14.95

adding power to your **commodore 64[®]**

Sprites! Color! Sound! These are just part of what makes the Commodore 64[®] such a special and exciting computer! **ADDING POWER TO YOUR COMMODORE 64[®]** shows you how to take advantage of the C-64's versatility and steer clear of programming and operating problems.

Written for both beginners and experienced programmers, **ADDING POWER TO YOUR COMMODORE 64[®]** covers everything you need to know to use your C-64, from operating fundamentals to advanced programming tricks. Here's what you'll find in the seven easy-to-read sections:

Introduction to the Commodore 64 Computer • The Components of the C-64 Computer System • Operating the C-64 • Advanced Section • Power Plus: A Utility Program • Programming Tricks and Program Descriptions • Appendices

Whatever your level of expertise, you'll find useful information, programming hints and suggestions, and operating techniques that will enhance your C-64. You'll learn how to design sprites, use sound, and write machine language programs. With **ADDING POWER TO YOUR COMMODORE 64[®]**, you'll add excitement to your computer programming!

Commodore 64[®] is a registered trademark of Commodore Business Machines.

Cover Design by Kevin McMahon

A Reston Computer Group Book
Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

0-8359-0060-6



21898 00605