SAMS

22456

The Waite Group

THE OFFICIAL BOOK FOR THE

COMMODORE 128™

PERSONAL COMPUTER

Mitchell Waite • Robert Lafore • Jerry Volpe

The Official Book for the Commodore™ 128 Personal Computer

Mitchell Waite,
Robert Lafore, and Jerry Volpe

The Official Book for the Commodore™ 128 Personal Computer

Howard W. Sams & Co., Inc.
A Subsidiary of Macmillan, Inc.
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

FIRST EDITION
SECOND PRINTING — 1985

International Standard Book Number: 0-672-22456-9
Library of Congress Catalog Card Number: 85-50977

Illustrated by *Bob Johnson*
Typography by *Walker Graphics*
Printed in the United States of America

The Waite Group has made every attempt to supply trademark information about company names, products, and services mentioned in this book. The trademarks indicated below were derived from various sources. The Waite Group cannot attest to the accuracy of this information.

8008 and Intel are trademarks of Intel Corp.
Adventure is a trademark of Adventure International.
Altair 8080 is a trademark of Altair.
Apple II is a registered trademark of Apple Computer, Inc.
Atari and Atari 800 are registered trademarks of Atari Inc.
Automatic Proofreader is a trademark of COMPUTE! Publications.
Bank Street Writer is a trademark of Broderbund Software.
BASIC Compiler-64 is a trademark of Abacus Software.
Blitz! is a trademark of Skyles Electric Works.
Byte is a registered trademark of Commodore Business Machines, Inc.
CalcResult is a trademark of Handic Software.
CBASIC is a registered trademark of Digital Research, Inc.
CBM is a registered trademark of Commodore Business Machines, Inc.
CP/M is a registered trademark of Digital Research, Inc.
Commodore 64, Commodore 128, and Commodore PET are trademarks of Commodore Business Machines, Inc.
Commodore *Microcomputers* and Commodore *Power Play* are trademarks of Contemporary Marketing, Inc.
COMPAQ and COMPAQ PLUS are trademarks of COMPAQ Computer Corp.
CompuServe is a registered trademark of CompuServe Corp.
COMPUTE!'s Gazette is a trademark of COMPUTE! Publications, Inc.
Context MBA is a trademark of Context Management Systems.
Crosstalk is a trademark of Microstuf, Inc.
Datasette is a registered trademark of Audio Devices, Inc.
dBASE II is a registered trademark of Ashton-Tate.
Deadline, Eyewitness, and Suspended are trademarks of Infocom.
Epson is a registered trademark of Epson Corp.
Evelyn Wood Speed Reading is a trademark of Timeworks.
Flight Simulator is a trademark of subLOGIC Corp.
FX-80 is a trademark of Epson America, Inc.
Home Accountant is a trademark of Continental Software.
IBM is a registered trademark of International Business Machines, Inc.
Learning With Leeper is a trademark of Sierra On-Line.
MATH BLASTER!, SPEED READER!, SPELL IT!, and WORD ATTACK! are trademarks of Davidson & Associates.
MBASIC, Microsoft, and Multiplan are trademarks of Microsoft.
MetaBASIC is a trademark of COMPUTE! Publications.
Mince is a trademark of Mark of the Unicorn.
MLX is a trademark of COMPUTE! Publications.
MOS Technologies and 6502 are trademarks of MOS Technologies.
PASCAL/MT + and SID are trademarks of Digital Research, Inc.
PeachText is a registered trademark of Peach/Tree Software.
Personal Accountant is a trademark of Continental Software.
Plus 14 is a trademark of Dentaration, Inc.
Popcom X100 is a trademark of Prentice Corp.
Popular Computing is a registered trademark of McGraw-Hill, Inc.
Power! is a registered trademark of Computing!
PractiCalc is a registered trademark of Computer Software Associates.
RUN is a trademark of CW Communications/Peterborough, Inc.

Screen-Graphics-64 is a trademark of Abacus Software.
Simon's BASIC, Easy Script, and Super Expander are trademarks of Commodore Business Machines, Inc.
Smart 64 Terminal Plus 3 is a trademark of Micro Technics Solutions Corp.
Smartmodem and Smartmodem 1200 are trademarks of Hayes Microcomputer Products, Inc.
Software Fitness System is a trademark of Open Systems.
Solo Flight is a trademark of Micro Prose Software.
Speedscript is a trademark of COMPUTE! Publications, Inc.
Subwar 64 is a trademark of Clockwork Computers, Inc.
SuperCalc, SuperCalc2, and Sorcim are registered trademarks of Sorcim Corp.
SuperForth 64 is a trademark of Parsec Research.
Super Term is a trademark of Midwest Micro, Inc.
Symphony, 1-2-3, and Lotus are trademarks of Lotus Development Corp.
The Manager is a trademark of Commodore Business Machines, Inc.
The Perfect Score is a trademark of Mindscape.
The Source is a trademark of Reader's Digest Association, Inc.
TI 99/4 is a trademark of Texas Instruments, Inc.
Trivia Fever is a trademark of PSI.
TRS-80 and Radio Shack are registered trademarks of Radio Shack.
Type Attack is a trademark of Serious Software.
UCSD Pascal is a trademark of Soft Tech Microsystems.
Ultra Font + is a trademark of COMPUTE! Publications, Inc.
Unix is a registered trademark of Bell Laboratories.
VIC, VIC II, and VIC 20 are trademarks of Commodore Business Machines, Inc.
VicTree is a trademark of Skyles Electric Works.
VisiCalc is a trademark of VisiCorp, Inc.
WordPro is a trademark of Professional Software.
WordStar is a registered trademark of MicroPro International Corp.
Z80 and Z80A are registered trademarks of Zilog, Inc.
Zorba is a registered trademark of Telcom Industries, Inc.
Zork I, II, and III are trademarks of Infocom.

Acknowledgments

The authors would like to acknowledge the assistance of the following people, without whom this book could neither have been written in the first place, nor changed from a manuscript into an actual product: Jim Gracely and Bob Kenny of Commodore Business Machines; Damon Davis, Phil Debrabant, Esther Eisman, John Obst, and Barbara Sams of Howard W. Sams, Inc.; and Lyn Cordell and Joan Frank of The Waite Group. All these people made sacrifices far beyond the call of duty; we salute their dedication.

Contents

1

Introduction to the Commodore 128

In this chapter you'll learn:

- **What this book is about**
- **The main features of the Commodore 128**
- **The three modes of operation of the Commodore 128**
- **Why you might want to buy a Commodore 128**

The Commodore 128 Personal Computer (shown in Figure 1-1) is one of the best values ever to appear on the home computer scene. It incorporates an amazing variety of features, at a price of less than $300. Some of these features represent firsts in the computer industry. The Commodore 128 is, of course, the first upgrade of the ubiquitous Commodore 64 computer, of which over three million have been sold to date. The Commodore 128 is the first low-cost personal computer to offer a full one-eighth megabyte (128K) of memory. It is also the first computer to combine the high resolution color graphics of the Commodore 64 with a serious business operating system (CP/M). The Commodore 128 is the first computer to offer two entirely separate microprocessor brains and video graphic chips, giving it essentially two different personalities: home computer and small business computer. It is also the first low-cost personal computer to offer an intelligent disk drive capable of reading dozens of storage formats. Finally, and most importantly, it is the first computer to combine, in one case, what amounts to three separate computers: the C64 mode, the C128 mode, and CP/M mode.

What's in This Book

This book describes the features of the Commodore 128, what it does, how it does it, and what kinds of things you can use it for. In this first chapter we'll give you a quick overview of the Commodore 128, explaining how it came to be and what its major features are. In later chapters we'll go into more detail about particular features of the C128, and how to use these features. We'll discuss the kinds of external devices (peripherals) you may need to operate the Commodore 128, and then we'll explore in some detail the three modes of operation available: the C128 mode, the C64 mode, and the CP/M mode. We'll finish up with chapters describing the sound and graphics capabilities of the C128, and how to make use of them in your own programs.

Our purpose in this book is not to give you an entire course in computers from the ground up, nor to teach you the fundamentals of BASIC or computer programming. Rather, we focus on those features of the Commodore 128 that make it unique and interesting.

From time to time in this book we'll mention the names of manufacturers of peripheral equipment and software, as well as organizations that are involved in the Commodore industry. The addresses of these manufacturers and organizations will be found in the back of the book.

Figure 1-1. The Commodore 128

If you have bought a Commodore 128, or if you are thinking of buying one, this book should answer most of your questions concerning the operation of the computer and whether it can do what you want.

A Word about Names

In a computer that contains three other computers, it's easy to get confused about whether you mean the main computer case or the individual computers inside. Commodore refers to the inside computers as *modes*, and that's what we'll do in this book. The main computer we'll refer to as the *Commodore 128*, or as the *C128*, for short. The three modes built into it are the *C128 mode*, the *C64 mode*, and the *CP/M mode*. The earlier computer on which the Commodore 128 was based we'll call the Commodore 64, or the C64, for short.

The Commodore 128: Three Computers in One

The main feature of the Commodore 128 — what sets it apart from almost every other computer ever manufactured — is that it is really three separate computers packed into one case. These three computers share the same keyboard, and can share external equipment, like the display screen and disk drives; so that when the C128 is sitting on your desk it may *look* as if there is only one computer system there. However, buried inside its handsome case there are really three separate computers, as shown somewhat fancifully in Figure 1-2. What are these three computers, and why did Commodore put them together in one case?

The first computer hidden inside the Commodore 128 is a faithful copy of an earlier computer: the Commodore 64. Introduced in 1982, the C64 turned out to be one of the most successful home computers of all time, with sales of over three million (and still climbing). The C64 was successful because it offered excellent color graphics and sound, which made it a perfect computer for such applications as education and entertainment. It was also, with its built-in Microsoft BASIC, easy for users to learn to program, and, most importantly, it was priced below the competition.

Because the Commodore 64 was so successful, a large software industry grew up to support it. Today there are more than six thousand programs which run on the C64, with new ones being written every day.

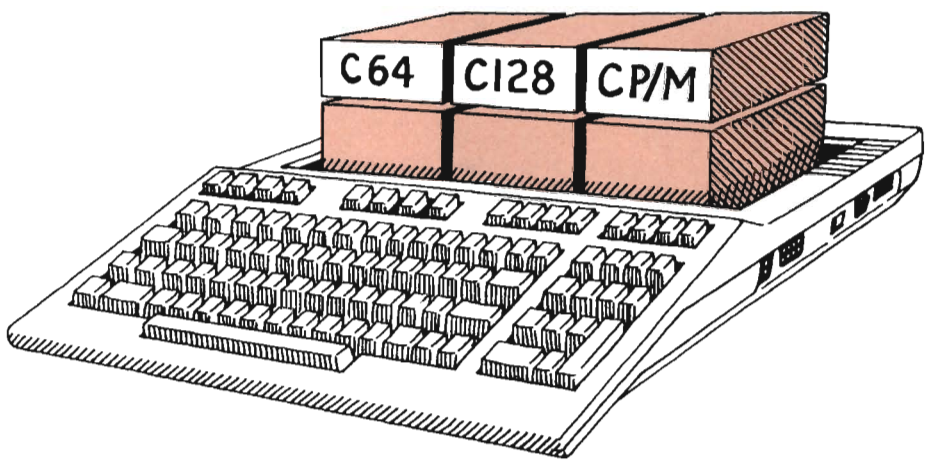
As we'll see, this huge existing software base has played a major part in the evolution of the Commodore 128.

The Commodore 64 was not without its faults. For one thing, the speed at which the disk system could transfer data was woefully slow. Users who needed to load a really long program or data file from a disk got used to brewing a cup of coffee while the transfer took place. Also, the C64 could display only 40 characters on one line of the screen. This was fine for simple business programs and for games, but users who wanted to use professional-quality software needed a screen capable of displaying 80 columns, the industry standard.

Commodore, in the face of competition from other home computer manufacturers, wanted to introduce a new computer which, using more advanced technology, would eliminate some of the problems found on the C64. However, because there were so many Commodore 64 programs in existence, they did not want a new machine which would be unable to run these programs. The millions of people who had invested heavily in programs designed to run on the C64 would not look kindly on a computer that was supposed to improve on the C64 but would not run the existing software.

However, it is difficult to make a computer *better*, without at the same time making it *different*. If Commodore made major changes in the Commodore 64, then the new computer would not have been able to run the old C64 programs. Many manufacturers have wrestled with this problem, and decided on one course or another: compatibility versus innovation.

Figure 1-2. Three Computers in One



Commodore decided to have their cake and eat it too: in one machine they would combine *both* the old C64, and a new computer called the C128, which would be similar to the C64, but better.

As we noted earlier, these two computers-within-a-computer are referred to as modes: the C64 mode, which lets the Commodore 128 emulate a Commodore 64; and the C128 mode, which is an entirely new computer with many enhancements and improvements over the C64. In addition to the C64 and C128 modes, Commodore added a third mode: the CP/M mode, which turns the Commodore 128 into a serious business computer. We'll describe these modes briefly here, and explore them in more detail in the chapters to come.

> ***It is difficult to make a computer better without, at the same time, making it different. You will soon see why the Commodore 128 is both — a different computer, and a better one.***

The C64 Mode

As we noted, the Commodore 64 featured excellent color graphics and sound; in fact, the most advanced in the industry for a home computer. The color graphics made use of sprites — sophisticated, intelligent graphics entities which could be programmed to represent objects like race cars and space ships. These sprites could be told to move anywhere on the screen and they would do so automatically, without slowing the program down. The sound-generating capability provided amazing versatility for a machine in this price range, including three voices and a wide variety of ways to shape the sound to produce almost any imaginable effect, from musical instruments to space ships to the human voice.

All these features have been retained in the C64 mode of the Commodore 128. In fact, in C64 mode the Commodore 128 is almost identical to the old Commodore 64. It will run all the same software. The screen display is the same, the keyboard operates the same way, and the disk drives and other peripherals that worked on the C64 will work the same way on the C128.

If you are a Commodore 64 owner you'll find this mode invaluable, because all your software will work on the new machine, without any modification, just as it did on the old one. Even if you never owned a

Commodore C64, you'll still profit from the C64 mode, because you'll be able to buy and use any of the thousands of programs already written for the Commodore 64. This mode will be discussed further in Chapter 4.

The C128 Mode

The C128 mode is, in effect, a brand new computer inside the Commodore 128. What improvements does this mode offer over the C64 mode? We've already mentioned two areas where the Commodore 64 needed improvement: the slow disk access, and the lack of an 80-column screen display. The C128 mode eliminates both problems, offering both much faster disk access and the choice — with the push of a button — of either 80-column or 40-column color displays.

Disk Data Transfer Speed

Data transfer times using the disk have been increased more than five times. This is possible using an entirely new family of accessory drives for the C128. These new drives are called the Commodore 1571 and 1572. The 1571 (shown with other peripherals in Figure 1-3) is a single drive, and the 1572 is a dual-drive unit. These drives will be discussed in more detail in the chapter on peripherals.

80-Column Mode

There are two different screen displays on the Commodore 128: 40-column, and 80-column, selected by a key on the keyboard. The 80-column display makes possible the use of much more sophisticated business-oriented software on the Commodore 128 than was possible on the Commodore 64. Note that not all monitors will work in 80-column mode; we'll have more to say about this in the next chapter.

The innovations introduced by the C128 mode don't end with faster disk access and the 80-column screen display. In addition, there is a vastly improved BASIC, increased memory capacity, and a machine language monitor, to name only the most obvious improvements.

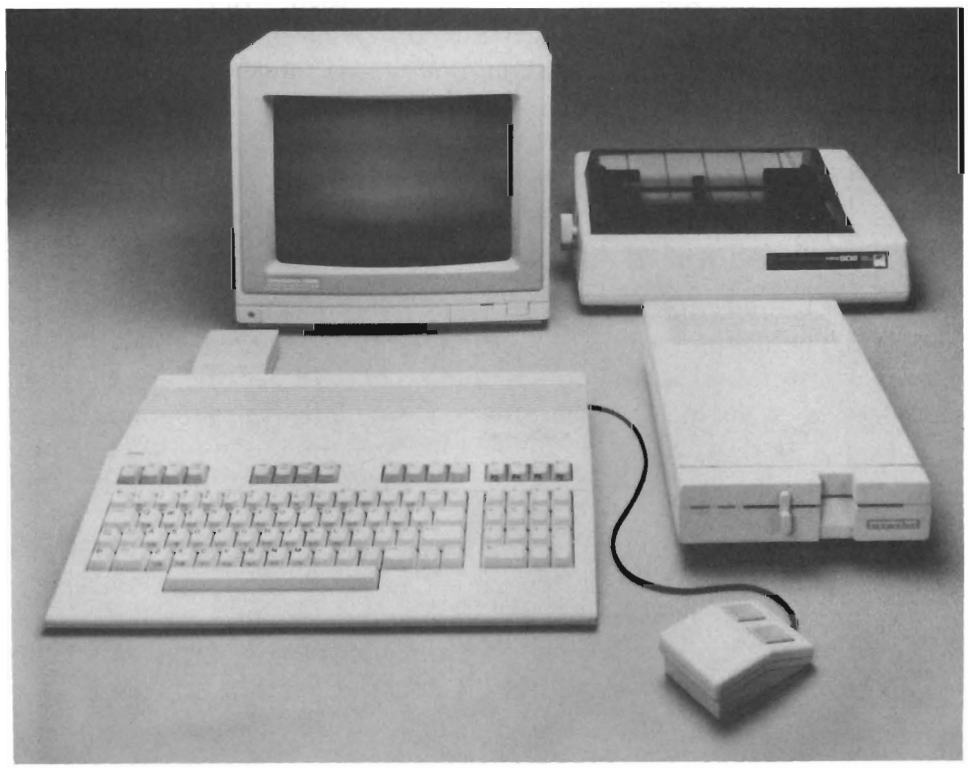
Advanced BASIC 7.0

Most home computers come with a programming language built in, so that users can learn programming, write their own programs, or customize programs written by others. The standard language built into most small computers is BASIC, which is probably the easiest programming language

to learn. The version of BASIC built into the Commodore 64, called BASIC 2.0, in general served very well, but it had some drawbacks. For one thing, it was difficult to take advantage of the excellent graphics and sound capabilities of the Commodore 64. Also, it was difficult to write programs in the modern "structured" style, since C64 BASIC did not have certain statements designed to make this kind of programming easy. Several accessories, notably Simons BASIC and SuperExpander, provided add-on commands to improve the BASIC, but they were not a part of the standard language.

The new version of BASIC that is accessible from C128 mode, called BASIC 7.0, remedies these deficiencies and adds many other new features as well. It provides a greatly increased number of statements, including those that make modern structured programming easier, such as DO...LOOP, BEGIN...BEND, and IF...THEN...ELSE. Graphics is simplified by the addition of commands that let you instantly draw circles, boxes, dots, and

Figure 1-3. The Commodore 128 System



lines; fill areas with color; and split the screen into a text section and a graphics section. Sprites (the colored graphics objects which can be programmed to move independently on the screen, like race-cars and space ships) are supported by simple commands which create sprites, move them, and check for collisions between them. There is even a built-in sprite editor that lets you design sprites on the Commodore screen and then save them for use in a program. There are also better statements to use the disk system (the DOS, or Disk Operating System), and a variety of new statements to help make the programming process easier, like AUTO, to automatically number program lines; RENUM, to renumber program lines; and TRACE, to allow line-by-line analysis of a program's operation.

In general, the new BASIC is a programmer's delight. It is probably the best and most powerful BASIC available to anyone buying a computer in the Commodore 128's price range.

Increased Memory

As its name implies, the Commodore 128 has 128K of memory, twice that of the Commodore 64. Because of various technicalities, the actual effect of this increased memory is that BASIC programs in C128 mode can be more than twice as large as they could be in C64 mode, as we'll discover in the chapter on C128 mode. And, if 128K of memory isn't enough, it's possible to expand it still further with optional plug-in memory modules, up to a maximum of 512K.

Improved Keyboard

A numeric keypad (the number keys clustered together as they are on a calculator) is useful for anyone entering large amounts of numeric data, but this feature was lacking on the Commodore 64. Since many Commodore 64 owners spent hours entering machine language programs in the form of long lists of numbers, this feature was often asked for. The Commodore 128 provides a full-featured numeric keypad, built into the computer on the right side of the keyboard.

In addition, the new keyboard has (among other new keys) additional cursor keys to make moving the cursor easier, a [HELP] key to ask for explanatory messages from those programs that use this capability, and the 40-column/80-column key for switching between display modes. We'll talk more about the new keyboard in the chapter on the C128 mode.

Built-in Machine Language Monitor

If you're a machine language or assembly language programmer, or even if you just like to explore various technical aspects of your computer, you'll appreciate this new feature of the Commodore 128: a built-in monitor program that lets you assemble, disassemble, and debug assembly language or machine language programs; and examine and modify memory.

The CP/M Mode

As if two computers in one weren't enough, Commodore also added a third, entirely different mode, to the Commodore 128. This is the CP/M mode. CP/M stands for "Control Program for Microcomputers." It's a *disk operating system* (a program which controls disk operations) developed by Digital Research, Inc., and available since the early days of microcomputers, almost a decade ago. You'll learn more about what operating systems are in general and about CP/M in particular in Chapter 5.

CP/M is an operating system that attempts to standardize program writing and usage, so that a program written to work on CP/M on one computer will work on CP/M on any other computer. CP/M has, in fact, been enormously successful, and tens of thousands of programs — mostly professional level business programs — have been written for it. Thus, by making CP/M available on the C128, Commodore is tapping into this vast reservoir of existing software. Since so much of this software is business-oriented, the addition of CP/M (along with the 80-column screen and faster disk drives) tranforms the Commodore 128 into a serious business-oriented machine. Chapter 5 details some of the more popular CP/M programs. It also describes the great amount of free "public domain" software available for CP/M.

The Bottom Line

If you're interested in owning one of the most versatile computers designed to date, with the ability to run thousands of existing educational, entertainment, and other programs, with many new enhanced features, and at the same time, a computer that can serve as a serious business machine, the Commodore 128 represents an excellent value. In the chapters that follow, we'll tell you more about this computer and explore its capabilities in detail. We'll begin by discussing peripherals for the C128: display

screens, disk drives, printers, and other external devices. Then we'll discuss the three modes available on the Commodore 128: the C128 mode, the C64 mode, and the CP/M mode; and finally we'll show you something about the C128's most exciting features: graphics and sound.

2

Peripherals: Displays, Disk Drives, Printers, and More

In this chapter you'll learn:

- **What "peripherals" are and why they're needed**
- **What peripherals are commonly used with the Commodore 128**
- **What peripherals are available from Commodore**
- **How peripherals attach to the Commodore 128**

When you buy a car, you probably also purchase a certain number of options: probably an automatic transmission and power steering, perhaps air conditioning, or tinted glass, or a stereo tapedeck. Some of these options are essential to the operation of the car, some are conveniences, and some — like racing stripes — are merely ego-boosters. A similar situation exists when you buy a computer: there are many options besides the computer itself — some essential to the operation of the computer, some less essential. In the computer world these options are called "peripherals," meaning that they sit around your computer, on its "periphery." Peripherals commonly used with the Commodore 128 are display devices, disk drives, printers, and modems.

If you want to see the output from your computer, you'll need some sort of display device: either a TV set or monitor. If you want to store programs or data when the computer is turned off, you'll need either a cassette tape storage device or a disk drive. If you want to print out programs or other data on paper, you'll need a printer; and if you want to

communicate over telephone lines with your computer, you'll need a modem. And this is just the beginning; there are many other peripherals as well.

Just as it is difficult to decide which options to buy (and which you can afford) when you buy a car, there may seem to be a bewildering array of peripherals to choose for your computer. In this chapter we're going to introduce you to the world of Commodore 128 peripherals, explaining what they are, reviewing what is available in the marketplace, and suggesting how to choose which equipment is right for you. We'll then explain just how these peripherals are attached to the Commodore 128. This is not a simple topic. There are so many plugs, openings, ports and connectors of different shapes and sizes on the Commodore 128 that it's necessary to explain which kind of peripheral plugs in where, and why.

We'll start off by talking about what's already built into the Commodore 128. From this discussion, the reason for needing some peripherals will begin to emerge.

What's Already Inside the Commodore 128

We've mentioned several different kinds of peripheral devices that may be necessary for your computer system. But what comes with the computer itself? What don't you need to buy?

The Keyboard

As you can see from Figure 1-1 in the last chapter, the Commodore 128 looks as if it's mostly a keyboard. This keyboard is the way that you, the user, will most commonly put information into the computer. However, the nicely styled plastic case that houses the keyboard contains many other things as well.

Chips

First and most importantly, the Commodore 128 contains the handful of "chips" (complex circuits etched onto a small crystal of silicon) that control the computer's operation. As a user you probably won't need to know much about these chips. If you're not interested in technical details, you can skip the next few paragraphs.

At the heart of the Commodore 128's circuitry is the 8502 microprocessor. This chip is a close cousin of the 6502 chip which powers the

Apple II, Atari, and other computers. The 8502 controls the operation of the Commodore when it is in C64 mode and C128 mode. In CP/M mode, an entirely different microprocessor is used, the Z80A.

In addition to the microprocessor chips, the Commodore 128 also has specialized chips to control graphics, sound, and communication with peripheral devices. Graphics is handled by a chip called the VIC II, for "Video Interface Chip." In 80-column mode (which we'll discuss below), another chip is used, the 8563, which can display twice as many dots per line as the VIC II chip. Sound is controlled by the SID, or "Sound Interface Device." The SID chip is the same one that was used in the Commodore 64 computer. There are two chips to handle communications with peripheral devices: they're called CIAs, for "Complex Interface Adaptors."

Memory

There are other chips which constitute the memory of the Commodore 128. The memory is divided into two parts: a permanent part which is filled with information at the factory, called ROM (for Read Only Memory); and a temporary part which can hold programs or data generated by the user, called RAM (Random Access Memory). The Commodore 128, as its name implies, contains 128K of RAM memory. (One "K" is equivalent to 1024 bytes (or characters), so the C128 actually holds 131,072 bytes). This is twice as much as the Commodore 64.

Something to remember about the RAM in your computer is that any programs or data that are stored in it will disappear when the power is turned off. Thus it is not a good place for permanent storage of programs or data. We'll have more to say about this when we talk about cassettes and disk drives.

Connectors

The last major category of components built into the Commodore 128 itself is the various plugs and connectors which allow it to communicate with its peripheral devices. We'll cover these in the last part of the chapter.

In the sections that follow we'll talk in turn about each of the broad categories of peripherals; then we'll discuss how these devices are connected to the C128. In later chapters we'll talk more about the kinds of peripherals which are particularly suitable for each of the three modes.

Display Devices

A display device is the only peripheral you absolutely must have to operate your Commodore 128. This display device can be an ordinary TV set, or it can be a video *monitor*. What's a monitor? It's very similar to a TV set, except that it can't receive TV broadcasts. Thus it offers a sharper picture than a TV set, because the signals from the computer don't need to go through the complex circuitry intended for TV broadcast reception. There are several different types of monitors, so choosing a display device is not a simple task. It depends on what you want to use your computer for, how good a picture you like to look at, and how much you want to spend. Let's review the available options.

The TV Set

The least expensive display device for your Commodore 128 is an ordinary home TV set. This is an excellent choice if you want to play games or use other simple programs. For these uses a color set is preferable, since almost all computer games make use of color. Since almost everyone has a TV set, this option usually doesn't cost you anything. Of course, if you only have one TV set, and you want to use your computer in a different room than where you usually watch TV, then you might find yourself wanting a second TV set; a somewhat more expensive proposition. In this case you should consider a monitor, to be described below.

To use your TV set with the computer you'll need a small box which contains a switch, so you can switch the TV back and forth between the computer and the normal TV antenna. This box is provided by Commodore along with the the C128. To use it, simply plug the cable from the computer into the switchbox, and plug another cable coming from the box into your TV set.

The major disadvantage of a TV set is that the *resolution*, or amount of detail you can see, in a TV image is fairly limited. For pictures — such as those you'll find in games and simple educational programs — a TV set works very well. But for text — such as program listings — it is less satisfactory: the letters and words are somewhat fuzzy, and can be tiring to read.

The Direct Video Color Monitor

Because text can be difficult to read on a TV set, different kinds of monitors are available, all of which offer clearer pictures than the standard TV does. You should consider a monitor if you will be using your computer

for writing with a word-processing program, or for working with spreadsheet or financial programs, or if you plan to spend a substantial amount of time writing your own programs. You might even want a monitor simply because you don't have a TV, or don't want to use your TV with your computer.

The least expensive color monitor available for the Commodore 128 is called a *direct monitor*, because the picture signals from the computer don't go through the broadcast-decoding circuitry as they do on a standard TV set. Here, one cable plugs into the computer, but is split into three separate plugs where it plugs into the monitor: one wire for sound, and two for special picture signals called *luma* and *chroma*. Commodore is unique among makers of small computers in using this kind of monitor. Most computer-makers use something called a *composite* monitor, in which the luma and chroma signals are combined in one wire. The Commodore approach offers a clearer picture, but you must buy a special monitor, made by Commodore, to take advantage of the system.

Commodore builds a direct monitor which has been very popular with users of the Commodore 64 computer: the 1702 color video monitor. You can also use a composite monitor, although the picture will not be quite as sharp. Other manufacturers produce inexpensive (under $200) composite monitors. Somewhat higher-quality composite monitors cost around $350.

Although a direct monitor offers a clearer picture than you might obtain with a TV set, it may still not be clear enough for all purposes. Specifically, it will display 40 columns of characters, but not 80. Let's see what this means.

40-Column Versus 80-Column

The Commodore 64 — the predecessor of the Commodore 128 — displays text of one size only. Forty characters (letters or numbers) just fit on one line of the screen: that is, there is one character in each of forty columns, so the letters are fairly large, and can be read even though they're a little fuzzy. This same way of displaying text is used when the Commodore 128 is used in C64 mode.

However, when you're typing a letter with a typewriter, you can usually get more than 40 characters on a line: closer to 80, in fact. This is also true of documents printed out on a computer's printer. Many people who use a computer for word processing, or for other business uses, want to see a line of text on the computer screen which corresponds more to

what they will see when the letter is printed out. For this reason, most business computers use an 80-column display mode, which fits 80 characters on each line.

The Commodore 128 computer lets you choose between a 40-column and an 80-column display. This is done by depressing a key on the keyboard. However, if you select the 80-column display, you'll need to use a monitor which will clearly display letters this small (half the size of the letters in the 40-column display). A TV set cannot be used in 80-column mode, and a composite monitor lacks sufficient resolution. To use this mode, you need a new kind of monitor called an *RGBI* monitor.

The RGBI Monitor

More expensive than the direct or composite monitors, but with a substantially better picture, is an RGBI monitor. RGBI stands for Red-Green-Blue Intensity. In this type of monitor, signals containing information about these three colors and the intensity are sent on four separate wires from the computer to the monitor. This results in a very clear color picture, since the signals for the colors are not first mixed in the computer and then unmixed in the monitor, as they are in direct and composite monitors.

Because an entirely different set of signals coming from the Commodore 128 is used for the RBGI monitor, a different connector on the back of the C128 is used for this kind of monitor than is used for a composite monitor or TV set. In fact, it's possible to have both kinds of monitors connected simultaneously, showing different pictures. (We'll show these connectors in the last section in this chapter.)

Many manufacturers make RGBI monitors. They vary widely in price: the better the picture you want, the more you must pay. The highest quality monitors sell for $750 or more, but good RGB monitors are available for around $400.

Figure 2-1 shows the connections for the various types of color displays.

A potential problem to note with the Commodore 128 is that you can't use an RGB monitor in 40 column mode. This is unfortunate, since it means that to use all the modes on your C128 you really need two monitors: direct (or composite) and RGB. Fortunately, Commodore has solved this problem.

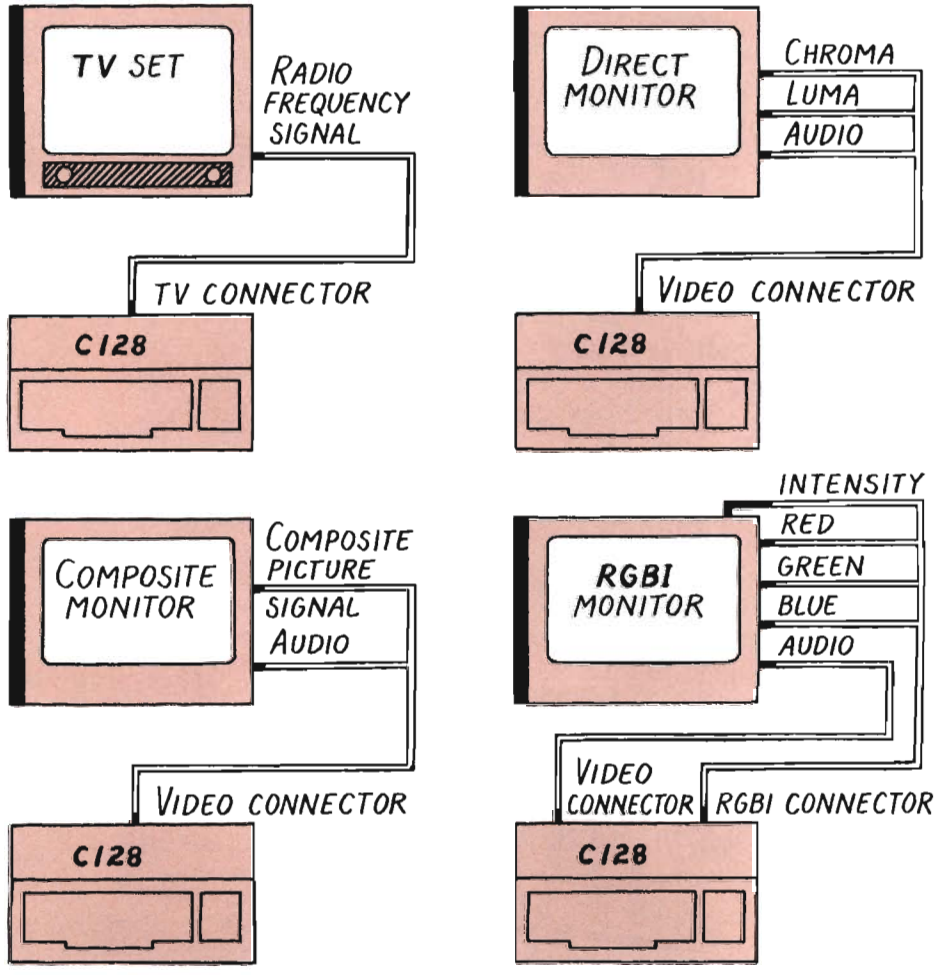
At the same time it introduced the C128 computer, Commodore also announced a new monitor: the 1902, shown in Figure 2-2. This monitor, which costs less than $300, actually combines several monitors in one. You can use it either as a composite monitor, or as an RGB monitor, switching from one to the other by means of a switch on the front panel.

Since it also accepts a composite signal, this monitor can also be used to display regular TV pictures, if you connect it to your VCR.

Monochrome Monitors

If you plan to use your computer mostly for word processing, or for business programs such as spreadsheets and database programs, your best bet might be what is called a *monochrome* monitor. Like a black and white TV set, this monitor doesn't show a color picture; it is usually either black

Figure 2-1. Connections for Different Displays



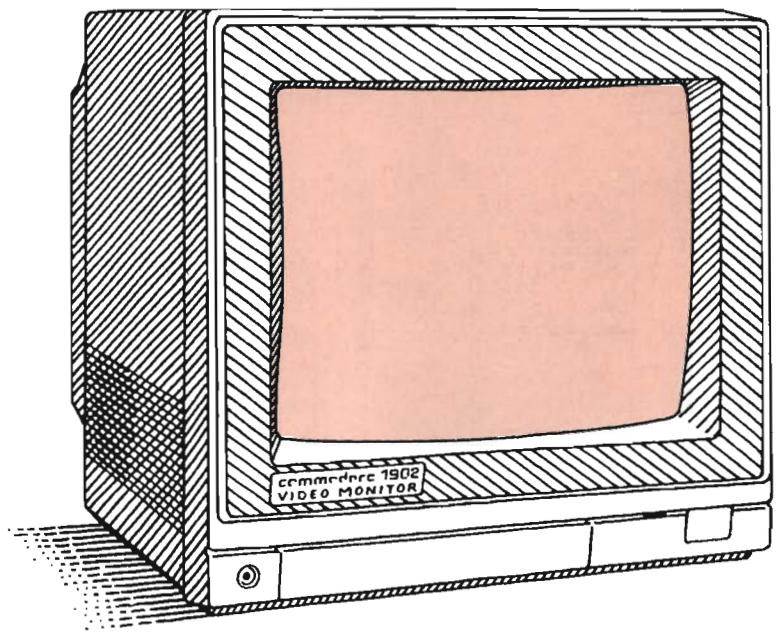
and amber or black and green. This monitor can be obtained for less than $200, and offers a much sharper picture than even an RGB monitor. If you're going to spend all day gazing at words or program listings on the screen, and you're not interested in full-color games, this option has a lot to offer.

If you're going to use the 80-column mode, make sure that the monochrome monitor you choose has enough resolution to handle this degree of detail; not all of them do.

Program Cartridges

Program cartridges aren't exactly peripherals in the sense a monitor or disk drive is. The cartridge *slot* (the connector where the cartridges plug in) is already built into the Commodore 128, so there is no major purchase to make to use this peripheral. The sole purpose of program cartridges is to provide a convenient way for software manufacturers to distribute their product. Only the manufacturer can put a program in the cartridge, so you can't use it to store programs of your own, as you can with a cassette or disk drive.

Figure 2-2. 1902 Color Monitor



The program cartridge itself is a plastic box about the size of a deck of cards. Some software manufacturers like it because it is almost indestructible, and the program inside can't be conveniently copied. However, cartridges are more expensive to produce than disks, so other manufacturers avoid them.

There is a considerable variety of commercially produced software available on program cartridges for the Commodore 64, which will also work on the Commodore 128 in the C64 mode. These programs are fairly inexpensive, ranging from around $19 to $40 for games, and up to $100 for more serious programs. It remains to be seen whether software developers will use this format for programs written specifically for the C128 mode.

Something to remember about program cartridges is that you should turn off the power to the computer before plugging in the cartridge. If you don't do this, you run the risk of damaging the computer, or the cartridge.

Cassette Units

When you turn off the power to your C128, everything you've stored in its internal RAM disappears. If you've written a long program and want to use it again without having to type it in again, this can be discouraging. You need some sort of permanent storage device, where programs and other data, such as letters you've written with a word-processor program, can be stored more or less permanently. Such long-term storage devices are sometimes called *mass storage*, since they can hold more data than can the memory in the computer. Program cartridges won't serve the purpose because, while they do store programs, you can't write your own programs on them; you can only read the manufacturer's program from them into your computer.

Cassettes offer the most inexpensive form of mass storage for your Commodore 128. A cassette unit is similar to the small cassette tape recorders used for playing music, and in fact it uses the same kind of cassette tapes. On many small computers it's possible to use any kind of cassette recorder, but the Commodore 128 uses a special kind of cassette recorder called a Datasette, made only by Commodore. (The Datasette records digital signals, rather than the analog signals used in other recorders. This makes it somewhat more reliable.) It is the most inexpensive way to store your programs and data, but it is also slow: you'll spend a long time waiting for a program to load.

Most users who start with a cassette recorder eventually upgrade to a disk drive, so our advice is to start off with a disk drive if you can

possibly afford it. Over 90 percent of Commodore 64 users chose a disk drive over the Datasette, and this figure will probably be similar for buyers of the Commodore 128.

Disk Drives

After a display device, a disk drive is probably the most popular peripheral for most personal computers.

Like cassettes, disk drives serve two principal purposes: they are a medium on which commercial software can be made available to your computer; and they also allow you to save, more or less permanently, programs or other data which you've created yourself. However, they offer far better performance than a cassette drive. At somewhat less than $300, the disk drive represents a relatively serious investment, but it is so much faster and more reliable than a cassette unit that it is worth the price for most users.

It is possible to use old-style Commodore drives, which were built for use with the Commodore 64, with the new Commodore 128. The most popular drive built by Commodore for the C64 was the 1541, a single drive unit (meaning it would hold only one disk at a time). Dual-drive units are also available from other manufacturers. All of these units will also work on the Commodore 128. However, they do not take full advantage of one of the major improvements of the Commodore 128 over the Commodore 64: the increased speed of the disk drives.

For the C128, Commodore has announced an entirely new line of disk drives. The single drive model is the 1571, shown in Figure 2-3. This drive operates in three modes, corresponding to the three personalities of the Commodore 128 computer. In C64 mode, it's completely compatible with the old 1541 drive, and runs at the same speed as the old drive. In C128 mode, it runs five times faster; and in CP/M mode, it runs faster still -- and is also compatible with most existing CP/M formats. The dual-drive version of the drive is called the 1572.

Data Transfer Speed

What does this speed advantage of the new drive mean to you? The old 1541 drive ran at about 320 characters per second. If you had a five-page letter or program listing of, say, 15,000 characters, it would take 46 seconds to transfer with the old drive (or with the new drive in C64 mode). In C128 mode, the new drive runs at 2000 characters per second, so the same document will take only 7.5 seconds to transfer. In CP/M mode, it's even faster: 3500 characters per second, for a 4.2-second transfer. If you've ever

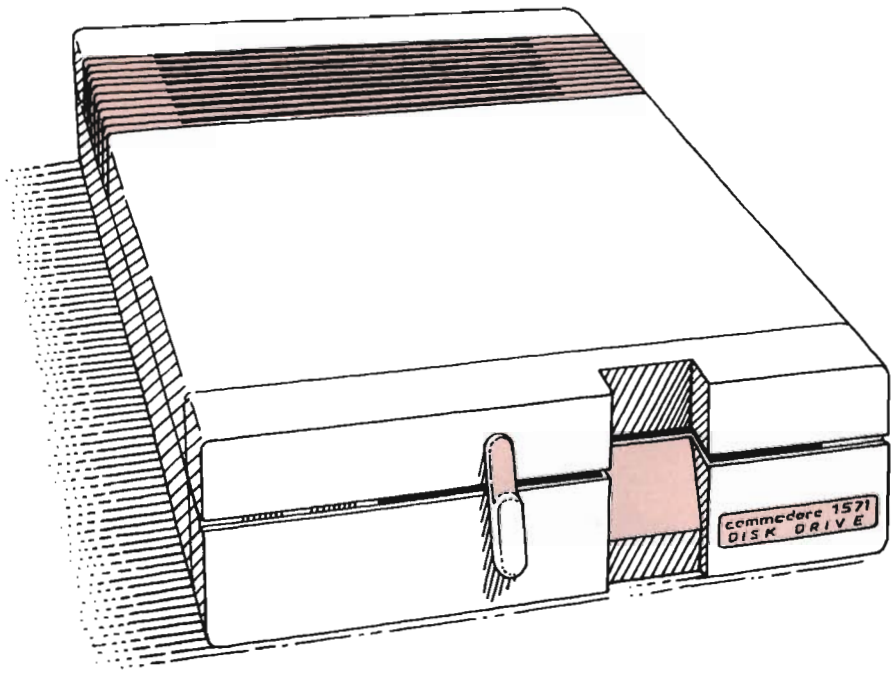
sat waiting for a long program to load on your Commodore 64, you'll know that this increased speed is is not just a gimmick: it can really make your life easier. The longer the programs you write or use, and the more you operate your computer, the more you'll appreciate the increased speed of these drives. The new 1571 drive also stores twice as much information on a disk: 340K bytes, versus only 170K for the older 1541 drive.

The new 1571 intelligent disk drive can transfer data up to ten times faster than the older 1541 disk drive.

Printers

You can have a lot of fun playing games and writing programs on the Commodore 128 without ever printing anything out. However, if you want to use your C128 for word processing, or for other serious business programs like spreadsheets, or if you want to write really long BASIC pro-

Figure 2-3. 1571 Disk Drive



grams, then you'll need a printer so you can save the results of your work on paper.

The kind of printer you need depends on how fast you want your output printed, how good you want it to look, and how much money you want to spend.

The least expensive kind of printer is called a *dot-matrix* printer. This kind of printer is fast, but the printed characters have that made-of-dots computer look to them. If you intend to write serious business letters on your computer, you will probably want what's called a *letter quality* printer. These printers are more expensive, and use some form of typewriter-like mechanism to form the letters, so the printing is much cleaner. However, these printers are usually slower than dot-matrix printers.

Commodore did not announce any new printers specifically for the Commodore 128, but they manufacture a variety of dot-matrix printers which work very well for the Commodore 64 and which are also perfect for the C128. These are the MPS-801, the MPS-802, and the MPS-803. The 801 and the 803 are the least expensive. The 802 is a heavier-duty machine suitable for business. All these printers can print not only text, but graphics (pictures). Being able to print graphics is useful if you are interested in computer art, or if you need to generate graphs and charts.

Other manufacturers also make printers that can be used with the Commodore 128. Notice however, that if a printer has not been built specifically to work with the Commodore, you'll need an adaptor (a small box or a special plug) to make the printer work with the computer. MSD Systems and CardCo are two companies making the adaptors, which are priced between $50 and $150 (addresses are shown in the back of the book). Even with the adaptor, not all printers will print graphics, so be sure to check that the printer does what you expect before you take it home.

For serious business use you'll probably want a letter quality printer, such as the Commodore DPS-1101. This produces copy like that typed on a good-quality office typewriter; but it's somewhat slower than the dot-matrix printers.

Modems

One of the fastest-growing uses of personal computers is the field of telecommunications. This means using your computer to send and receive data over the telephone lines. Why would you want to do that? You might want to send a program to a friend in another city. Or you might be

interested in accessing one of the on-line database services, such as CompuServe or The Source. These services give you instant access to the latest stock prices, weather reports, movie reviews, and other data. They also let you communicate with groups of people who share a common interest, such as stamp collecting or electronic music. These groups are called SIGs, for *special interest groups*. There is, for example, a SIG for Commodore computers on CompuServe. These groups offer free programs, free advice, and a forum for discussion.

The most important variance between different kinds of modems is that of speed: how fast can the modem send or receive a certain amount of information? A slow modem can take a long time to transmit even a small amount of information; so if you plan to use a modem extensively, a fast one is a good investment.

The slowest modems used today are rated at 300 baud. What does *baud* mean? Very roughly, the baud rate is the number of characters per second the modem can send, divided by 10. Thus, a 300 baud modem can send about 30 characters per second. (This definition isn't exactly right, but it's close enough to give you an idea how fast modems work.)

Commodore sells a variety of modems: the 1600 and the 1650, which have been around for some time, and the newer 1660 and 1670, which were introduced with the C128 computer. The first three are 300 baud modems, while the 1670, shown in Figure 2-4, runs at a brisk 1200 baud. While the 300 baud modems range in price from $40 to about $100, the 1670 will set you back about $200.

In addition to Commodore, there are a wide variety of other modem manufacturers, including Hayes, Human Engineered Software, and Anchor Automation. Some of these modems are designed to plug directly into the C128, but others will require an adaptor called an RS-232 interface module, which costs about $50 and can be obtained from several manufacturers, including Commodore and MSD Systems.

Besides speed, modems are also distinguished by features such as: Will the modem automatically answer the telephone? Will it automatically dial the phone to make an outgoing call? In the Commodore line, all but the 1600 have these autoanswer autodial features. However, the 1650 and 1660 require special software to use these features. For instance, if you want to dial a number, it is the software that generates the clicks for each digit you dial. In the 1670 on the other hand, the modem itself generates the clicks; all the software needs to tell it is what number to dial. The 1670 is thus a "smart" modem, which makes it compatible with a wider variety of communications software.

A modem should be one of the first peripherals you own. With it, you can shop from home, access the latest stock prices, weather reports, movie reviews, electronic bulletin boards, and perhaps best of all, obtain free programs.

Joysticks

For games and various other programs you'll probably want to buy one or more joysticks or similar devices. Joysticks are small handheld levers that let you input up-down left-right directional information directly into the computer. Most joysticks also include a button, which is used to fire guns or lasers in computer games.

Figure 2-4. 1670 Modem

Besides joysticks, there is a variety of other devices that act more or less like joysticks, including trackballs, mice, touch tablets, and paddles. Some users prefer one or the other of these devices, and some games or programs are designed to work with a particular device. However, the joystick is the most popular of such devices, and is a good start for anyone interested in games.

Note that not all such devices can be used with any given program. Joysticks and trackballs are mostly compatible with each other, in that they both transmit information about direction to the computer: is the stick pointed north, northeast, east, or where? Touch tablets, mice, and game paddles, on the other hand, send a position to the computer. They often work in conjunction with a pointer on the computer screen, which is positioned at a particular place.

Other Peripheral Devices

The variety of peripheral devices made for the Commodore 64 and the Commodore 128 is truly staggering. There are speech synthesis and recognition devices, as well as peripherals to open your garage door, check your house for burglars and fire, and operate your model trains. With the proper equipment you can get your C128 to draw engineering drawings, respond to your touching its screen, or synthesize music from a piano-like keyboard. We won't describe these esoteric devices here. What we've said about the more common peripherals should be enough to begin thinking about what you'll need for your Commodore 128.

Connecting Peripherals to the Commodore 128

Any peripheral device you buy is connected to the Commodore 128 by plugging it into one of the openings on the side or the back of the C128. There are quite a number of these connection ports, in a wide variety of shapes and sizes. It's easy to get confused and try to connect things in the wrong place, so this section will describe these connectors, and explain which peripheral gets plugged into which connector.

The first group of connectors we'll talk about is located on the right side of the Commodore 128, as shown in Figure 2-5.

Be Careful

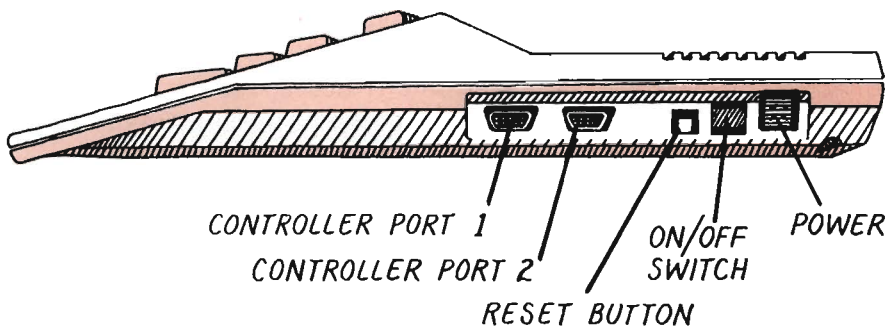
Connecting peripherals to the Commodore 128 is as simple as connecting the components of a stereo system — usually, there is only one way a connector can be plugged in. Because some of the Commodore 128 connectors contain voltages from inside the computer, it is a good practice to turn off power before plugging or unplugging peripherals. Never plug or unplug ROM cartridges or peripherals connected to the user port without first turning off the power.

Power and Reset

On the right side of the Commodore 128, closest to the back, is the *power connector*. One cord from the power supply plugs in here. The power supply is a small box, about 6 inches long, which converts house current to the lower voltage used by the C128. Next to the power connector is the *on/off switch*.

Notice that although this switch turns off the power to the computer, the power supply itself keeps going. You might want to unplug the power supply from the wall outlet if you're not going to use the computer for a while. This will make the power supply last longer. You can also buy a "power strip," a row of outlets with its own switch which plugs into the

Figure 2-5. Right Side of the Commodore 128



wall. This is useful because you can plug your computer and all your peripherals into it, and turn them all off at once with the switch on the power strip.

Next to the on/off switch is the *reset button*. This is used to restart your machine without turning it on and then off again. You may need to restart it if a program gets "hung up," and the computer will not respond to keyboard commands. Also, some types of copy-protected software require you to reset the computer before they will load. You can do this by turning the computer off and then on again, but that's hard on the electrical components inside the computer, so it's better to use the reset switch.

The Controller Ports

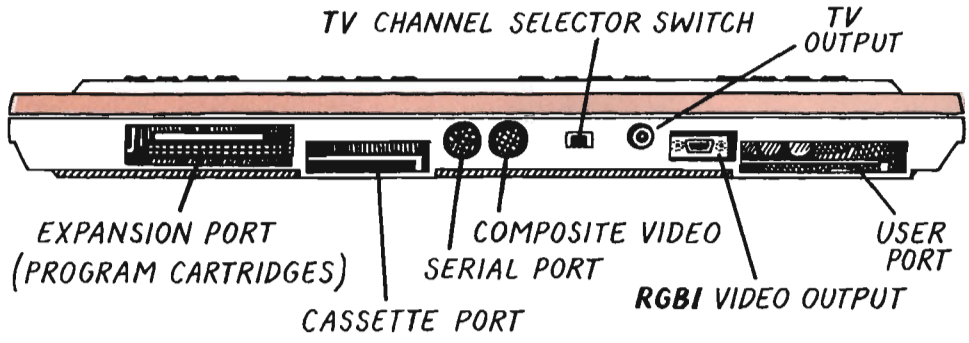
The controller ports are for joysticks and similar devices. Attaching such devices is fairly straightforward: you simply plug them in. Be careful, however, if you're only using one joystick, to plug it into the correct port. Software written by Commodore generally requires you to plug your joystick into port number 1, while that written by other developers often requires you to use port number 2.

The rest of the connections are found on the rear of the C128, shown in Figure 2-6.

The Serial Bus Connection

This connection is used for several different kinds of peripherals, most commonly disk drives and printers. Technically, the word *serial* means

Figure 2-6. Rear View of the Commodore 128



that each character or byte of information is sent in separate bits, one after the other. Since each byte is composed of eight bits, this way of sending information is theoretically eight times slower than sending each byte all at once, as is done in a parallel connection. However, sending all eight bits at once requires eight separate connectors, so the serial technique is less expensive.

The word *bus* refers to a kind of data path used in the computer. The bus is really more like a highway than a bus. Data is sent down the bus to a variety of different destinations. Like cities lying along a highway, peripherals can occupy different positions along the bus. Let's look at this arrangement in more detail.

Daisy-Chaining

An unusual feature of the serial bus connector is that more than one peripheral can be plugged into it. How is this possible? Through a technique called *daisy-chaining*. This means that the first peripheral is plugged directly into the serial bus connector on the back of the computer. Then the second peripheral is plugged, not into the computer, but into the first peripheral. The third peripheral is plugged into the second, and so on. There can be as many as five different peripherals connected to the Commodore's serial bus connector. This process is shown in Figure 2-7.

A question that might occur to you is, if all these peripherals are on one cable, how does the computer know which peripheral it's talking to when it sends a message out this connector?

> ***Because the Commodore 128 uses intelligent peripherals that can communicate over a daisy-chained serial bus, connections are simpler than on other computers, and expanding the system is also easier.***

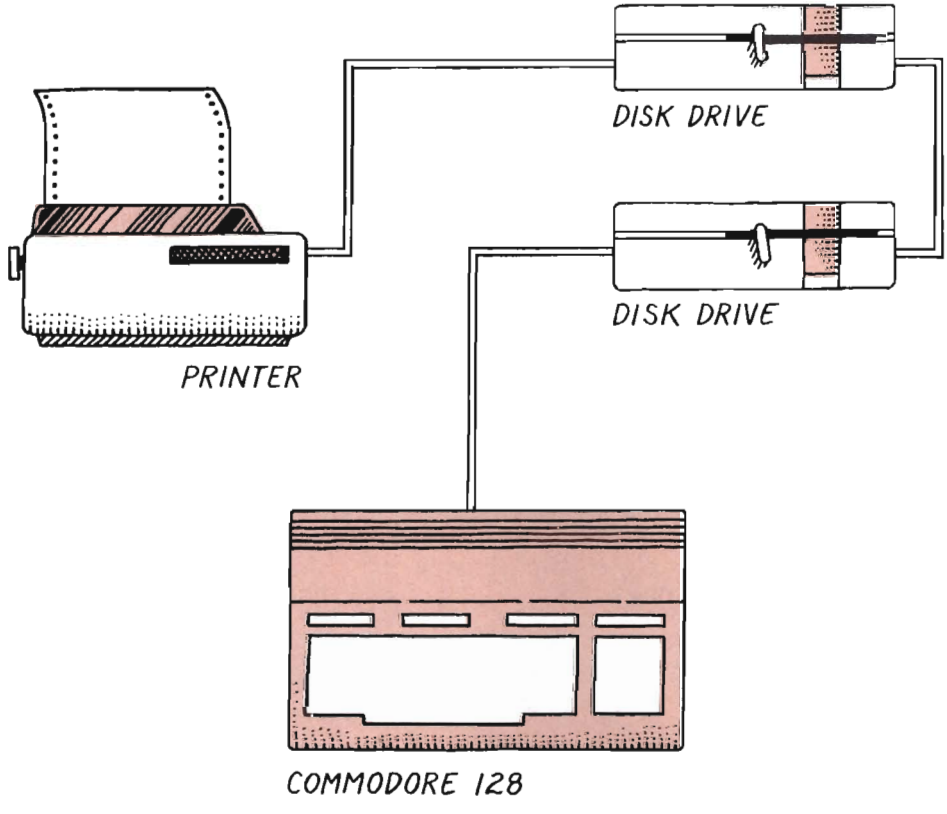
Intelligent Peripherals

To understand how the serial bus connector works, you should know that the Commodore 128 can make use of a special kind of peripheral device called an *intelligent* peripheral. In this regard the Commodore is unusual in the computer world. Most computer systems can only use what can be called "dumb" peripherals. The computer tells them exactly what to do, and they report back every little thing that's going on inside them, and can make no decisions without the computer's help. The C64 and C128, on the other hand, can use peripherals that perform complex tasks all by

themselves. The disk drive, for instance, can format a disk, keep track of what programs are on a disk, and read and write programs from the disk, without any help from the computer itself.

This intelligence also permits the computer to communicate with a particular peripheral, even though there are several peripherals connected to the single serial bus connector. Every time the computer sends a message to a peripheral, such as a request to the disk drive to send a file, it sends something called a *device number* along with the message. Each peripheral has its own device number, and when it sees the message going down the cable, it checks to see if the number on the message is addressed to it. If the disk drive is device #8, for example, and the message is for device #4 — the printer — the disk drive will ignore it; but the printer will accept the message and do what the message says, probably to print out a particular file.

Figure 2-7. Daisy-chained Peripherals



Now you know why only certain peripherals, those built especially for the Commodore system, will work on the C128. They must be intelligent peripherals, capable of interpreting the special messages sent from the computer.

Networks and File Servers

Another use for the serial bus connector is to connect a number of C128s or C64s together so that they can all share certain peripherals — commonly a disk drive and printer. This is a popular system in schools, where it's more economical to give each student only the keyboard unit, and buy a disk drive and printer that will serve an entire class. In such a system, each computer is connected — via the serial bus — to a box called a *file server*. The file server is, in turn, connected to the printer and disk drive.

The User Port

The user port is a connection on the right-hand side on the rear of the Commodore 128. It is most commonly used for modems, but it is also used for a variety of more esoteric peripherals, such as voice synthesizers, model train controllers, and so forth. It is, in many ways, like the RS-232 port which is standard on many other computers — but there are differences — so that standard RS-232 equipment, such as modems, will not work. Modems and other equipment which use this port must be specially modified to work with Commodore equipment. This port is the same on the C128 as it is on the old C64, however, so that any equipment that is attached to the user port on the C64 will work on the C128.

To attach a Commodore modem to this port no cable is needed: you simply push the modem right into the port. Of course, you must then connect the modem to the telephone, following the instructions for the particular modem. It is possible to use standard RS-232 equipment with this port, but you'll need a special adaptor to make it work with the C128.

The TV Output and Channel Selector Switch

This small round outlet is where you plug in the cable if you're using a standard TV set. The cable actually connects the computer to the switchbox, which is, in turn, connected to the TV. The channel selector switch is set to channel 3 or 4, whichever channel is not used (or is weakest) in your area.

Composite or Direct Video Connector

This connector is the leftmost of the two large round connectors on the rear of the C128. This kind of connector is called a DIN connector. The cable coming out of it separates into several RCA-type jacks (the kind you use to connect your hi-fi equipment together). If you're using the direct color monitor, these jacks go to three separate connections on the back of the monitor, labeled chroma, luma, and audio. If you're using the composite monitor, there are only two jacks to connect to the monitor: the video signal and the audio signal.

The RGB Video Connector

This is the small oblong D-connector. It connects directly to the back of an RGB monitor, using four different jacks: red, green, blue, and audio.

The Expansion (or Program Cartridge) Slot

This is a simple connector to use, but its effect on the computer is rather complicated. The most common use of this slot is for plugging in program cartridges. This is easily done: you turn the computer off, plug in the cartridge, turn the computer on again, and the game or whatever program is in the cartridge takes control of the computer. Besides games, you can also use this slot to change your computer from one which speaks BASIC into one which speaks other computer languages, such as FORTH.

If you know something about how computers work, you'll be interested to know that almost every interior signal in the Commodore 64 and Commodore 128 computers is made available on this connector, including the data bus and the address bus. Thus, whatever is plugged into the slot can completely take over the operation of the computer, changing it into a machine with different characteristics. It's even possible to plug in a different microprocessor chip here, so that the computer assumes an entirely different personality. The C64 uses this capability to turn itself into a CP/M computer with the aid of an optional cartridge; on the C128 this capability is built into the computer.

The Cassette Port

The type of connector used for the cassette port is called a Molex. A cable from the Commodore Datasette recorder or another recorder specially designed to work with the Commodore 128 plugs in here.

Now that you know what peripherals are available for the C128 and how to hook them up, you're ready to find out more about the three modes of the Commodore 128 Personal Computer: the C128 mode, the C64 mode, and the CP/M mode. In our discussion of each of these modes we'll have more to say about which peripherals are suitable for which mode.

3

The C128 Mode

In this chapter you'll learn:

- **What the C128 mode is**
- **What the C128 mode has to offer**
- **What equipment you will need to use this mode**
- **The new features of BASIC 7.0**
- **About the C128 Disk Operating System**
- **Sources for additional information**

The C128 Mode is one of three possible operating modes for the Commodore 128 personal computer. It is in this mode that the C128 distinguishes itself as more than just a slightly enhanced version of the Commodore 64. The purpose of the first part of this chapter will be to clarify this mode's more interesting features and its more unusual characteristics, and to explain the various applications suited to the C128 mode. The next part of the chapter explains which extra system components you should have to use this mode. Then we'll summarize the operations possible from the new version of BASIC, and the disk drive support provided by the new DOS. Lastly, we tell where you can go to get additional information or help about this mode.

What Is the C128 Mode?

As you will recall from the introduction, Commodore had three different markets in mind when it designed the Commodore 128. It wanted the C128 to be "absolutely compatible" with the Commodore 64 (the C64

mode), but it also wanted the C128 to have sufficiently enhanced features, as compared to the C64, to attract new buying interest (the C128 mode). It also wanted the C128 to run software written for CP/M-based computers (the CP/M mode). In order to meet these three design goals Commodore had to literally design three modes of operation into the C128. At first thought, such a feat might seem very difficult, especially if the three personalities were really different, as they are in the case of the C64, C128, and CP/M modes.

Building three computers into one *multimode* computer would have been an impossible task had Commodore limited itself to traditional computer designs. Instead, it compared the various hardware and software components that make up a C64, a C128, and a CP/M machine to see exactly how they overlapped with one another. Then it built the Commodore 128 using a composite pool of all these components, and used an organizer function (an intelligent memory/device manager) to arrange the parts to make a C64, a C128, or a CP/M computer on the fly. Figure 3-1 shows this arrangement of different components as used in the C128 mode. As you can see, some components of the Commodore 128 are not used in the C128 mode, and others are. The overall selection was to ensure the best possible arrangement of all C128 features. Details on how these components are rearranged for the C64 and CP/M modes will be found in chapters 4 and 5.

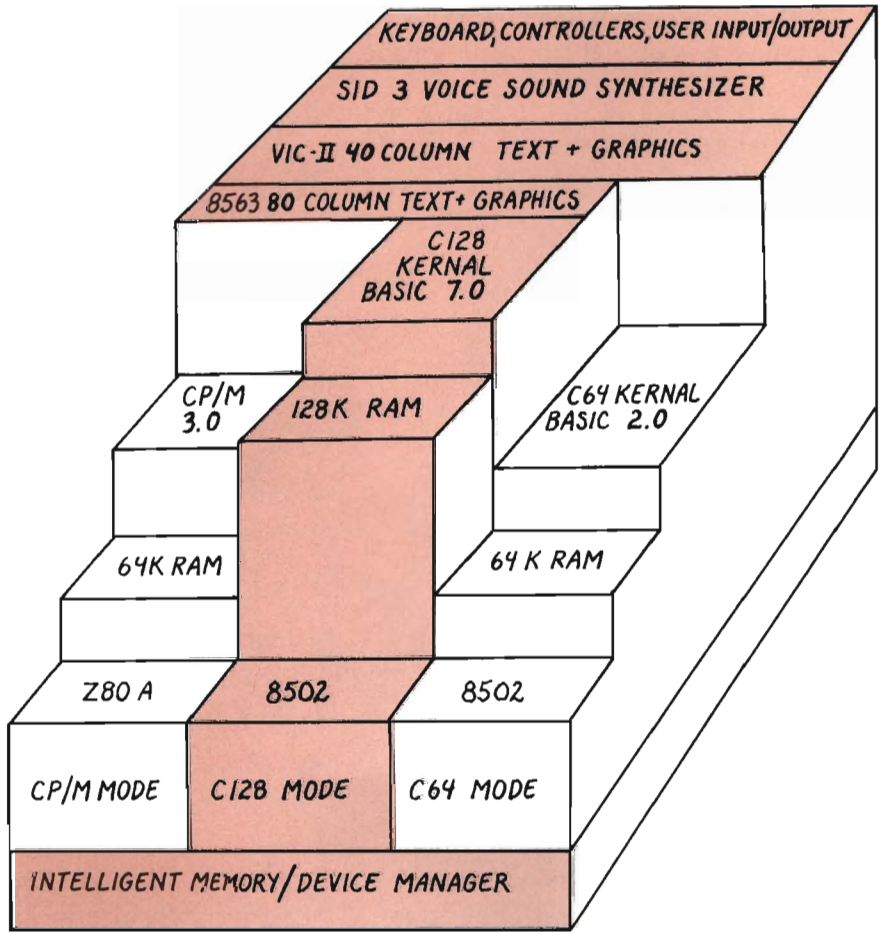
What Does the C128 Mode Offer?

When used in the C128 mode, the Commodore 128 is essentially an enhancement of the Commodore 64. As such, this mode offers all of the features of the C64 with very few of its disadvantages. The C128 mode has access to the same graphics and sound capabilities and the same input and output capabilities, including the expansion bus, the versatile programmable user port and the two internal real-time clocks. It will even operate with all C64 peripherals, such as the 1702 color monitor and the 1541 disk drive. On the other hand, the C128 mode removes its users from having to work within the somewhat primitive confines of the C64's BASIC 2.0 and its equally antiquated DOS structure.

So what are the enhancements? To begin with, the C128 mode has access to at least twice the memory of a C64. It also provides a second powerful, 80-column color text and graphics system, a better-equipped 96-key keyboard, and access to a friendlier and faster single- or dual-disk drive system.

The C128 mode enhancements just mentioned are changes to the computer's hardware side. On the software side, Commodore updated the Kernal Operating System (its built-in operating instructions), added some important enhancements to the screen editor, built in a super powerful machine-language monitor utility, and threw in the most powerful BASIC ever in a Commodore computer. Lastly, when you use the C128 mode with one of the new multiformat disk drives, the 1571 or 1572, you get double-sided disk storage instead of single (twice as much room per disk) and a noticeably improved computer-to-disk-to-computer transfer rate. Let's examine each of these features in more detail.

Figure 3-1. How C128 Mode Stacks Up



What Types of Applications Does the C128 Have?

The Commodore 128 is an extremely capable computer, with features that broaden its appeal to a much wider range of applications than those the C64 focused on. For all its graphics and sound power, the C64 had a number of weaknesses that made it more useful for applications relying on eye and ear stimulation than for those requiring fast or versatile computing power. As a result, the C64 found its primary niche in applications such as entertainment and education. The C128, on the other hand, incorporates 128K of memory, a choice of both 40- and 80-column screens and a more business-capable disk storage system, plus a number of other enhancements that make it adaptable to just about any personal computer application.

Because of the C128's unique balance of computing features it will find equal acceptance in homes, schools, offices and even industry. For example, the types of real-life situations the C128 can be used in include:

Office productivity

Business management

Computer-aided design

Instrumentation control

Computer software design

Computer-aided instruction

Personal productivity

Home management

Entertainment

What Are the C128 Mode Enhancements?

In order for you to appreciate why the C128 is capable of such different working applications you need to understand how its new features affect its overall capabilities as a personal computer. That's what we'll explore in the following section.

More Memory

The C128 mode offers 128K of user memory, or twice that found in the C64. It also has memory space for larger *ROM-based software packages.*

A ROM-based software package is software that is permanently stored in a special memory chip that can then be used by the C128. This type of software is merged into the C128 memory either as a cartridge through the C128's expansion slot or as a "ROM chip" addition inside the C128.

For many C128 users, the expanded memory of the Commodore 128 means only one thing: they'll be able to run larger and more sophisticated software programs like Artificial Intelligence (AI), Expert Systems, large spreadsheets, and fuller-featured text editors. This is an important consideration, since many non-Commodore commercial application programs could only be made available to the C128 if it has a sufficient amount of memory. Whereas a few years ago, 64K of memory was more than satisfactory, today most popular business-oriented software packages are intended to run on systems with 128K or 256K of memory. The C128, with its ROM-based operating system and unique intelligent memory manager, frees up virtually all of its 128K of RAM for program use. In contrast, most other personal computers using a disk-based operating system (like CP/M or MS-DOS) forfeit a fair amount of their available RAM to hold their DOS. Since the C128 does not have this type of overhead there is more of the 128K to use for programs.

C128's memory power can be further enhanced through the external addition of up to 384K of additional RAM. This added memory is plugged into the C128 via the expansion bus in the same manner that you might plug in a software cartridge. Don't be misled, though — this additional 384K of memory is not intended to be an add-on to the existing RAM area. Commodore has chosen instead to use it as a *RAM disk*. A RAM disk is a section of RAM memory that is used as if it were another disk drive. Programs and data can be moved back and forth from regular memory to the RAM disk area just as if a disk drive were involved. The major advantage of this type of memory use is speed, since programs and files can be moved from one area of memory to another in a fraction of a second. This is a quantum improvement in speed; even when compared to the typical transfer rate of a very fast disk drive. A RAM disk option is especially useful with a large application that normally has to continually read and write to a disk during operation. Of course, when you are finished using a particular program all data in the RAM disk area must be saved on a disk or it will be lost. Further, before a program can be used from the RAM disk area it must first be copied to it from a disk.

A Larger Keyboard Selection

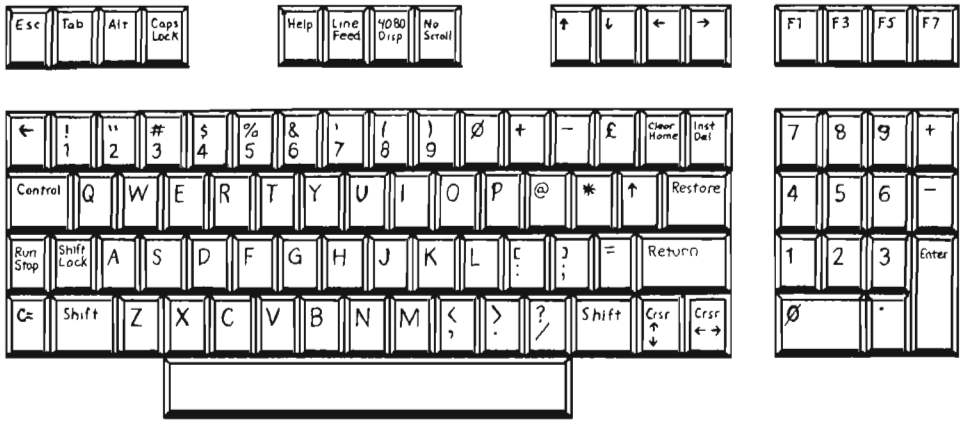
The Commodore 128 has a new keyboard. The C128's keyboard, shown in Figure 3-2, has 26 more keys than the C64. To begin with, there is the

original C64 keyboard area. This section has all the familiar multifunction text and graphics keys, as well as the standard C64 control keys: [Shift], [Control], [Run/Stop], [C=], [Inst/Del], [Clear/Home], and [Restore]. The Commodore 128 adds a 14-key numeric keypad just to the right of the main keyboard. This is perfect for long sessions of calculator-style data entry. The keypad even has its own plus, minus, decimal and [Enter] keys to add to its convenience. Directly above the main keyboard is a single horizontal row of 16 keys, divided into four banks of four each. These key banks include the original C64 Function Keys, a new set of individual up, down, left, and right cursor motion keys, and eight special-purpose keys.

The four physical "function keys," so familiar to C64 owners, are now located just above the numeric keypad section. As before, these keys can have two different values assigned to them (F1/F2, F3/F4, etc.); that is why Commodore refers to them as eight actual keys. Regardless of how you count them, they remain very important because of what they are not — they are *not* pre-defined in the same way as the other keyboard keys. Instead, they are user-definable keys. This means that their definitions, either simple or complex, can be changed through software — a nice plus for creating menu-driven application programs.

The Commodore 128's new keyboard has two very important keys — ESCape and ALTernative (located in the leftmost bank of new keys). [Esc] and [Alt] keys have been pretty common to business-style small computers for a number of years now, so it was only natural for Commodore to provide them on the business-oriented C128. The purpose of these two

Figure 3-2. The C128 Keyboard



keys is to further extend the range of different code combinations that can be output from the keyboard. By doing so they make the C128 more attractive to those established business software developers who are used to working with more traditional keyboard arrangements.

The [Help] key is of particular use to C128 users, especially those planning to write their own BASIC programs. When the C128 is in BASIC, pressing the [Help] key sends a special "give me help — where did I go wrong" signal to the BASIC interpreter. If used just after a BASIC program stops due to some error condition, then BASIC will respond with additional information as to where and possibly why the program ran into trouble. Of course, since the [Help] key, like all C128 keyboard keys, is also redefinable it is certain to become a favorite for application software developers too.

The remaining new keys, summarized in Table 3-1, all add certain operating conveniences to the C128.

40- and 80-Column Text and Graphics

Another new feature of the C128 mode is its ability to display 80 as well as 40 characters on a line. This is possible because of a second video processor chip inside the C128. This chip, an 8563, can be used instead of the standard 40-column chip by depressing the [40/80 Disp] key to its locked-down position before either applying power to the C128, or by pressing the RESET button when power is on. When this is done, the C128 initializes itself in the 80-column format. Of course, to use this new screen mode you must have a video monitor, capable of displaying 80 columns, connected to the C128's RGBI connector. Switching back to the 40-column mode also requires either turning the power off and on again or pressing RESET. Fortunately, there are ways to perform the screen mode switch from within a program, which is how most commercial software will ensure that the C128 is in the correct screen mode.

The advantages to using the new 80-column screen mode are numerous. Naturally, you have an 80-character by 25-line display so you can have up to 2000 characters on the screen at the same time, and they can be in color. What's more, because of the *bit-mapped* nature of this screen mode you can have the C128 display a total mix of all available text and graphics characters (not possible in the 40-column mode) and even mix text and graphics characters with high-resolution drawings. The character font definitions within the C128 (they determine what shapes will show up on the screen when you press a certain text or graphics key) can be changed to give the screen output a totally new set of characters — even a new language. It is possible by doing a little internal rearranging to use

a reduced-size character set and obtain an even larger 132-column by 25-line screen display.

Remember that the C128 has access to both the 40- and the 80-column screen video processors and the C128 can take advantage of that fact too. Not only can the C128 choose between these modes according to the position of the [40/80 Disp] key, the screen modes can also be switched from within a program. Therefore, it is possible to use two displays (a 40-column display and an 80-column one) to show outputs from the same program. Both displays can't be updated at the same time but changes can be alternated between the two. As long as both displays remain enabled, their outputs will stay visible. As the 40 and 80-column outputs come from separate screen processors, they can display different information and even be in totally different text and graphics modes. In order

Table 3-1. New Keys for the Commodore 128

Key	*Definition or Function*
[Esc]	Used in conjuction with one or more text keys to generate an ESCape code sequence (used in the C128 in the CP/M terminal emulator mode).
[Tab]	Used to move the cursor to the next Tab Stop.
[Alt]	Used simultaneously with some other text key to generate a special ALT code.
[Caps Lock]	Used in the 80-column mode to allow independent selection of upper- and lowercase letters and the two graphics sets.
[Help]	Used in BASIC 7.0 to locate errors while programming. May also be used as a general "HELP" key by some software applications.
[Line Feed]	Provides a LINE FEED without CARRIAGE RETURN.
[40/80 Disp]	Switches display from 40 to 80 columns.
[No Scroll]	Inhibits screen scrolling by preventing the cursor from going beyond the 25th display line.
[↑]	Moves cursor up one line.
[↓]	Moves cursor down one line.
[←]	Moves cursor left by one character position.
[→]	Moves cursor right by one character position.
Function Keys	Definable function keys (see BASIC 7.0 section).
14-Key Keypad	Calculator-style keypad with separate plus, minus, decimal, and [Enter] keys.

to take advantage of this dual-screen display ability the C128 must have both of its video outputs connected to appropriate video monitors. We'll discuss this more later.

The Commodore 128 can actually be speeded up internally by using the 80-column screen mode by itself. To do this, you must use software to actually turn the 40-column chip off. C128 data processing operations, when used in this manner, run at nearly twice the rate obtained when the 40-column screen processor is enabled. This increase in processing power is greatly appreciated when a computer is required to do the type of data processing and number crunching typical of most business, industrial, and scientific applications.

Enhanced BASIC

The enhanced BASIC, called BASIC 7.0, that is available to you in the C128 mode is the most powerful version of BASIC yet offered by Commodore. In all there are 140 commands, operations, and functions — 20 commands, 35 operations and 13 functions more than were found in the Commodore 64's BASIC version 2.0.

BASIC 7.0 gives you increased control of: program flow; conditional testing; logical evaluations; mathematical functions and procedures; text manipulation and output; data input, processing, storage, retrieval, and presentation; file construction and usage; program testing; and just about every conceivable graphics and sound operation that the 128 can perform. In addition, BASIC 7.0 supports a full array of commands that greatly simplify the tasks of disk drive control and file maintenance. Despite its enhancements, BASIC 7.0 remains downward-compatible with earlier Commodore BASIC versions 2.0 and 4.0. That is, a program written for the earlier Commodore BASICs will run under 7.0, although not vice versa. Note that this downward compatibility applies to the program as written on paper. The disk versions of BASIC programs are stored differently, and are not compatible.

BASIC 7.0 takes advantage of nearly the full 128K of the C128's RAM (122365 bytes), and does it in such a way as to eliminate the old "garbage collection" delays that occurred when larger BASIC programs were run on the Commodore 64. (C64 programs would often halt on their own for several minutes while BASIC 2.0 tried to make room in memory for more operations.) 64K of the Commodore 128's memory is set aside to hold just the program code, while all variables generated by that program are kept and maintained in the other 64K segment of memory. With so much memory space, it is possible to further divide the C128's memory so that a number of programs can be placed in memory at the same time.

Another feature of BASIC 7.0 is that the C128's eight function keys come predefined with often-used BASIC commands. Of course, they can be easily redefined by BASIC commands to perform other operations of your choice.

Later in this chapter we'll take a look at the various commands, statements, and functions found in BASIC 7.0.

Enhanced Screen Editor

Commodore computers have always been famous for the versatility of their built-in screen editors. These "full screen" editors act as temporary working windows where you write and organize your BASIC commands or program statements. Through them you can move the cursor anywhere on the screen, insert and delete any number of characters, and erase the screen completely. Commodore screen editors have some interesting side capabilities too. For example, you can list a group of lines from a program in memory to the screen and then use the screen editor's cursor motion and editing keys to change anything on any BASIC line within the editing screen, although nothing actually changes in BASIC's program memory until you press the [Return] key. Duplicating entire lines is as simple as typing a new line number over the old and pressing [Return]. The screen editor also has a special *quote* mode that allows you to mix text, graphics, and even cursor actions right in your program code.

Commodore has added some new features to the Commodore 128's screen editor to improve upon the C64's screen editor. For example, program lines can be 160 characters long, as compared to 80 on the C64. This is made possible by an ingenious form of auto-wraparound within the screen editor that keeps track of where on the screen each BASIC line begins. In the 80-column screen mode a BASIC statement can be two screen lines in length, while in the 40-column mode it can be up to four screen lines. Two other screen editor enhancements are line insert and delete and four-way margin control.

Built-In Monitor Program for Assembly Language Programmers

The Commodore 128 has a built-in *Machine Language Monitor*. This special programming utility was once a built-in feature on all Commodore PETs, but was later removed from the VIC-20 and C64 due to memory restrictions. The Monitor, as it is called, is an absolutely essential tool for anyone interested in writing, testing, or modifying programs written in

8502 machine language. It will be used by professional software developers as well as by C128 owners who want to write short and fast machine language routines to speed up critical parts of their own BASIC programs. The Monitor is also an excellent learning tool for students who want to understand how machine language programs work (most commercial software is written in machine language, and so are programs like BASIC 7.0, Logo, and FORTH).

The Monitor utility in the Commodore 128 is an enhanced version of the earlier Commodore monitors. A similar version was made available to Commodore 64 users as part of Commodore's "Assembler Development Package" for that machine. Table 3-2 summarizes the monitor commands built in to the C128. Complete details on how the monitor is used can be found in both the *Commodore 128 User's Guide* and the *Commodore 128 System Guide*.

Table 3-2. Commodore 128 Monitor Command Summary

Command	*Command Name*	*Command Description*
A	ASSEMBLE	Assembles a line of 6502 machine code
C	COMPARE	Compares two sections of memory and reports the differences
D	DISASSEMBLE	Disassembles a line of 6502 machine code
F	FILL	Fills memory with the specified byte
G	GO	Starts 6502 code execution at the specified address
H	HUNT	Hunts through memory within a specified range for all occurrences of a set of bytes
L	LOAD	Loads a file from tape or disk
M	MEMORY	Displays the hexadecimal and ASCII values of memory locations
R	REGISTERS	Displays the 8502 registers
S	SAVE	Saves memory to tape or disk
T	TRANSFER	Transfers code from one section of memory to another
V	VERIFY	Compares memory with tape or disk
E	EXIT	Exits Commodore 128 MONITOR
.	(period)	Assembles a line of 6502 code
>	(greater than)	Modifies memory
;	(semicolon)	Modifies 8510 register values
@	(at sign)	Displays disk status

Faster and Friendlier Disk System

The last really important enhancement that the Commodore 128 provides is improved disk storage capabilities. Not only can the C128 operate using the Commodore 64's 1541 single-sided disk drive, it can also work with its own special "super intelligent" double-sided disk systems — the 1571 (single drive) and the 1572 (dual drive).

You can choose to use the C128 with one or more of the older 1541 disk drives as they are totally compatible. However, doing so does not provide you with the disk storage improvements that the C128 is capable of — more storage capacity per diskette and vastly improved data transfer rates. The advantage of the new disk system's increased friendliness is a function of BASIC 7.0, not the drives, so the C128's friendlier disk commands hold true even for the 1541s.

The real improvement in disk storage capabilities happens when the C128 is used with the 1571 or 1572 disk drives. When these drives are used, the C128 has immediate access to twice the storage space per disk of the 1541s. For example, the 1541 could store 170K of programs and data per disk, but the 1571 and 1572 can store 340K per disk. Of course, the total disk storage space available to the C128 doubles to 680K when you consider both disks in a dual drive 1572. In addition to storage space improvements, these new drives are also fast. The 1541, with its typical data transfer rate of about 320 characters per second, is extremely slow in operation and, unfortunately, doesn't get any faster when it is connected to a C128. On the other hand, the new drives, when used in the C128 mode, operate at up to 5200 characters per second. High data transfer rates like this make the C128 a viable machine for even the most disk-intensive business application.

Finally, Commodore has provided the C128 with an easy-to-use set of DOS commands. The DOS commands are the same as those found in the late model Commodore PET and CBM machines (BASIC 4.0). DOS calls are used to perform a wide assortment of disk-related functions, including reading directories, formatting blank disks (using a command called HEADER), copying files, renaming files, or removing (scratching) files. When the 1572 dual drive is used there is even a command to copy one disk to another (BACKUP). These commands are very important to all C128 users, so we'll take a detailed look at them later in this chapter.

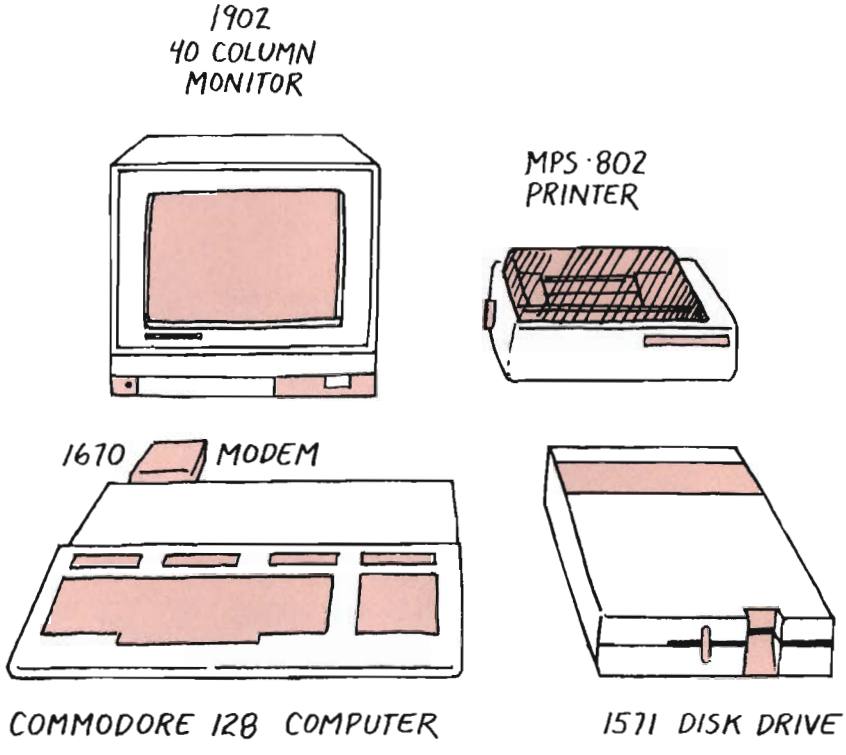
What Equipment Do I Need to Use the C128 Mode?

As you learned in the previous chapter, the Commodore 128, with all its power and flexibility, is just the computing part of a system. You still need

a monitor of some sort so you can view the messages it wants you to see, and you will probably want some form of external mass-storage device to hold your programs and data. Major C128 system accessories also include a printer for a permanent paper copy of important computing activities, and a modem to allow you to connect your C128 to other computers via the telephone. Other system add-ons would really depend on the requirements of your application.

Since the Commodore 128 is still compatible with all Commodore 64 peripheral devices, you have a very wide assortment of peripherals to choose from — both old and new. If you already have a complete assortment of C64 peripherals, than you can simply connect them to the C128. Of course, you won't be able to take advantage of all of the C128's features, like the 80-column option or the increased storage ability and speed of the newer disk drives; but you will be able to do most anything you want, as long as you remain within the limitations of those C64 peripherals.

Figure 3-3. Recommended Commodore 128 System



The best way to take full advantage of your C128's capabilities is to use the system components that were designed specifically for it. Figure 3-3 shows the C128 system configuration that we would recommend for general use.

If your computer needs are less complicated — for instance, if you simply want to experiment with BASIC 7.0 and 40-column graphics — then you might want to consider starting with a color television as a monitor and a Datasette as a mass storage device.

Now that you know the major improvements the Commodore 128 has made over the Commodore 64, let's examine two of these improvements in more detail: BASIC 7.0 and the new DOS.

All About BASIC 7.0

In this section we'll describe the new BASIC 7.0 that is built into the Commodore 128. This section is presented in three parts. The first part examines the most general facts about BASIC 7.0, giving you a bird's-eye view of the language. The second part explains and lists the BASIC statements and commands that are common to both the C64 (BASIC 2.0) and C128 (BASIC 7.0) modes of the Commodore 128. This way you'll get a feel for the available statements if you wanted to write programs that would run in both the C64 and C128 modes. The third part lists and explains the statements unique to the C128 mode. If you are only interested in BASIC 7.0 enhancements then you can read the third section directly, but if you want to know what BASIC 7.0 is like to program with, then you should read the first section as well. Chapters 6 and 7 explore the graphics and sound statements of BASIC 7.0.

Important Facts About BASIC 7.0

Let's examine the major features of the new BASIC 7.0 that come with the Commodore 128.

Modes

When you are in the C64 mode you are actually using BASIC 2.0; the same BASIC found in the Commodore 64. Therefore BASIC programs are written in exactly the same manner as they would be on a C64. When you are in the C128 mode, however, BASIC 2.0 is replaced with BASIC 7.0 and you have virtually an entirely new programming environment to work in.

Modified Microsoft BASIC

Commodore 128 BASIC is a modified version of Microsoft BASIC, commonly referred to in the CP/M universe as MBASIC (see description of MBASIC in Chapter 5, CP/M Mode). The new BASIC 7.0 is based on statements from the older C64 BASIC 2.0. Added to these are disk statements and commands from the Commodore PET/CBM BASIC 4.0 (the CBM line of business microcomputers). In addition there is a large number of new statements for controlling sound, graphics, windows, sprites, and peripherals. Microsoft BASIC was the first BASIC made available for the 8080, Z80, and 6502 microprocessors and enjoys a wide following. The new BASIC 7.0 has included several new statements that tend to completely remove several C64 BASIC 2.0 weaknesses, chiefly in the area of program control and looping. Also added are new statements for graphics, sound, windowing, external I/O devices, and keyboard access.

Onscreen Editor

The new BASIC 7.0 is built around a cursor-key-controlled onscreen editor. This is much like the BASIC editor of the IBM PC, where to alter or edit a line of the program, you move the cursor to the line you wish to change, type the correct information, and press [Return] to capture the change. When using the 40-column screen mode, BASIC 7.0 allows the same keyboard character and graphics set combinations as on the C64. Therefore you still can't display upper- and lower-case letters with both sets of keyboard graphics. On the other hand, when BASIC 7.0 is used to program for the 80-column screen mode, you can mix and match all available text and graphic symbols. Also when you enter a program in BASIC 7.0, putting spaces between keywords, variables, operators and such is strictly optional. Adding spaces will clarify the meaning of statements, and spaces consume very little additional memory space, so they can be used generously.

Graphics and PRINT Statements

Any characters, any colors, and any cursor activity can go inside of quotation marks. That is, you can use any key on the keyboard directly in PRINT statements. Thus you can insert dozens of cursor direction keys, graphics symbols, and colors in PRINT statements, and thereby create complex images on the screen.

Programmable Function Keys and Other Keys

There are four function keys, and each key can perform two functions. The keys are programmable from BASIC and thus can be used by your program for any function you want. When you first start BASIC the function keys are predefined, as shown in Table 3-3. F1, F3, F5, and F7 functions are accessed directly by pressing the appropriate key. You use the [Shift] key to access the F2, F4, F6, and F8 keys.

You program a function key with the KEY command to stand for some other BASIC sequence or string. For example:

```
KEY 7, "GRAPHIC 0" + CHR$(13)
```

would make the command GRAPHIC 0 followed by a carriage return occur when [F7] was pressed.

BASIC also responds to the [Help], [No Scroll], [Caps Lock], [Alt], [40/80 Disp], [Tab], and [Line Feed] keys. Each of these keys has its own initial definition and this definition can be changed in BASIC.

Memory and BASIC 7.0

BASIC 7.0 has access to all 128K of the Commodore 128's total RAM space. This 128K is split into two banks. The lower 64K bank is for the text portion of BASIC programs: the actual statements that make up the program. The upper 64K bank is used strictly as work space. This is where BASIC will keep the program's variables, arrays, and strings. The BASIC interpreter occupies 48K of ROM. Since arrays, strings, and variables are kept in a separate area of memory, this configuration allows the

Table 3-3. Predefined Functions Keys in BASIC 7.0 (C128 Mode)

Key	*BASIC Keyword*	*Purpose*
F1	GRAPHIC	Sets graphic mode you enter
F2	DLOAD"	Loads a filename you type
F3	DIRECTORY	Gives a directory of the disk
F4	SCNCLR	Clears the screen to current background color
F5	DSAVE"	Saves a filename you type
F6	RUN	Runs the program in memory or one on disk
F7	LIST	Lists the program in memory
F8	MONITOR	Enters the machine language monitor

text of a BASIC progam to occupy a full 64K regardless of the length of strings or arrays. This means a 64K program on the C128 is actually equivalent to a much larger program than on the C64; thus longer, more sophisticated programs are possible.

Variables and Limits of BASIC 7.0

BASIC 7.0 allows three kinds of variables: normal numeric (floating point), integer, and string variables. A floating point variable is assumed as a default (that is, BASIC will assume you mean floating-point unless you tell it otherwise). Floating point variables can have up to nine digits of significance and can range in size from ten to the –38 power to ten to the +38 power.

In BASIC 7.0, integer variables (whole numbers without fractions) range between –32768 and +32767 and are indicated in a program by appending the percent symbol (%) to the end of a variable name.

String variables, used to hold symbolic data like text and graphics, can be up to 255 characters long and are indicated by a $ sign appended to the end of the variable name.

A variable name in BASIC 7.0 can be as long as desired, but only the first two characters are significant. In other words, UNCOLA and UNLIMITED are legal variables, but are considered by BASIC to refer to the same variable. The first character has to be a letter but the second character may be a letter or a number. Graphic symbols are not allowed in variable names. Here are some legal BASIC 7.0 variables:

```
FOOBAR a floating point variable
NAME$ a string variable
P123% an integer variable
```

BASIC arrays are defined by the DIM statement. The number of dimensions is limited only by the length of the program line. Arrays follow the same name rules as regular variables do. Besides BASIC keywords, there are eight variable names that are reserved to BASIC 7.0 and can't be used in your programs. These are listed in Table 3-4.

The Commodore 128 has a built-in 24-hour clock function that is automatically updated every 1/60th of a second. It is both set and read using the reserved variable TI$. This clock is not a separate real-time clock operation, but instead a software *jiffy* clock just like the one used in the C64. It is most appropriate for programming operations where short term accuracy is required as it does lose count during certain system operations (for example, when the disk drive is being used). Since there

is no battery backup, the clock must be set each time the computer is powered up.

You can use the error trapping variables with the new TRAP command to process errors in more creative ways than BASIC normally offers. The disk channel variable DS lets you find out why the red error light on the disk drive is blinking.

Other Details

BASIC 7.0 offers all the normal math operators (+ – * / ^) and has the logic operators AND, OR, and NOT. The length of a BASIC 7.0 line is limited to 160 characters (counting spaces). Thus a BASIC line can occupy two lines on an 80-column screen and four lines on a 40-column screen.

Overview of C128/C64 Common BASIC Statements

In this section we will briefly review the statements and commands that are common to both BASIC 2.0 and BASIC 7.0. The BASIC DOS commands are covered in a separate section later in this chapter.

Assignments and Equates

Both BASIC 7.0 and BASIC 2.0 offer statements for clearing all variables, defining a custom function, and for dimensioning an array. These are shown in Table 3-5.

Table 3-4. Reserved Variable Names in BASIC 7.0

Variable	*Purpose*
ST	Status variable: tells success of an I/O operation
TI	Holds the value of the 24-hour jiffy clock
TI$	Sets the jiffy clock with this variable
DS	Reads disk drive command channel, returns status
EL	Line where the last error occurred
ER	Returns last error (number) since the program was run
ERR$	Contains error message for last error

Programming Commands

As Table 3-6 shows, both BASICs have commands that let you manipulate the program in memory.

Looping

The statements for controlling program flow and looping are listed in Table 3-7. The standard FOR...NEXT loop allows looping a fixed number of times. A STEP option allows making the loop index at some value other than one. GOSUB is used for branching to subroutines. A RETURN statement returns control to the line following the GOSUB. ONGOSUB is a more advanced GOSUB that allows indexed subroutine branching.

Program Control Statements

Several useful program control statements are provided, as shown in Table 3-8. The WAIT statement is a way to make the program stop and wait for

Table 3-5. BASIC Assignments and Equates

Statement	*Purpose*
CLR	Clears out variables and arrays
DEF	Defines a custom function
DIM	Dimensions an array
LET	Optional assignment statement

Table 3-6. BASIC Programming Commands

Statement	*Purpose*
LIST	Lists program in memory to screen
REM	Adds a remark or comment to a line
STOP	Program will stop and print out line number
NEW	Erases current program and all variables
RUN	Runs or executes current program or filename
CONT	Continues a stopped program

a particular bit in a specific memory location to change. It is mostly used to monitor the status of bits in input/output registers, such as sprite registers.

Machine and Memory Control

As shown in Table 3-9, there are several useful statements for controlling the memory of the C128, including PEEK (which reads the contents of a memory location) and POKE (which places an integer between 0 and 255 in a memory location). The SYS statement performs a call (like a GOSUB) to a machine language subroutine at a certain address. It can also optionally load the arguments a, x, y, and s into the accumulator, x-register, y-register, and stack register of the 8502 microprocessor. The address range of SYS is 0 to 65535 so the high bank of the 128K memory must be accessed indirectly with SYS. USR is another function that allows you access to a machine language subroutine. Unlike SYS, USR requires that you poke the starting address of the machine language routine into two specific memory locations (1281 and 1282). USR can also pass a parameter

Table 3-7. BASIC Looping Statements

Statement	*Purpose*
FOR...NEXT STEP	Loops a fixed number of times
GOSUB/RETURN	Used for branching and returning from subroutines
GOTO	Forces direct jump to a program line number
ONGOSUB	Indexed GOSUB
ONGOTO	Indexed GOTO
RETURN	Return from subroutine

Table 3-8. Program Control Statements

Statement	*Purpose*
END	End program execution
RESUME	Resume program execution at a certain line
WAIT	Wait for a memory location to change

(a variable value) from BASIC to that routine and from that routine back to BASIC. SYS will not allow you to pass this type of two-way parameter.

Input/Output and Data Control

Both BASIC 7.0 and BASIC 2.0, as shown in Table 3-10, provide numerous statements for controlling the input and output of information, as well as the storing of information and data within the program. The INPUT and GET statements are used to get information typed at the keyboard into variables (string or numeric) in a program. The Return key must always be pressed to terminate anything typed in response to the INPUT statement. The GET statement (used only with string variables) does not need a Return as it simply reads the keyboard for its current status and returns whatever key value (including none) that it received to its variable. INPUT# and GET# are two variations of INPUT and GET that are used to

Table 3-9. Machine/Memory Control

Statement	*Purpose*
PEEK	Returns contents of memory location
POKE	Places a value in a memory location
SYS	Calls a machine language subroutine and loads registers
USR	Calls a machine language subroutine and passes parameter

Table 3-10. Input/Output and Data Control Statements

Statement	*Purpose*
GET	Allows inputting a keyboard character without Return
GET#	Single character inputting from any legal input device
INPUT	Inputs a string or number from the keyboard
INPUT#	Inputs a string or number from any legal input device
PRINT	Displays a string or number on the screen
PRINT#	Displays a string or number to any legal output device
DATA	Holds a list of numbers or strings
READ	Reads numbers or strings in data statements into variables
RESTORE	Resets the DATA/READ pointer

read from any file structured input devices like the Datasette disk drive, modem, and even the keyboard. PRINT is the statement used to output information to the display. PRINT# is used to output information to any file structured output device. CMD is a redirection command that allows you to redefine the default output device. DATA is used to store a list of constants, numbers or letters, in a program in an easy-to-read format. READ is used to assign these constants in the DATA statements to variables.

Functions

Functions perform operations on numbers, strings, output devices, and memory. Table 3-11 shows that BASIC 7.0 and BASIC 2.0 contain a large number of these functions. The functions are used for obtaining the SIN or COS of a number, converting a string to a number, or vice versa, and so on. It also tells if a joystick fire button has been pressed.

This completes the description of the BASIC statements and keywords common to both BASIC 7.0 and BASIC 2.0. In the next section we will examine those enhanced keywords that are particular to BASIC 7.0.

Table 3-11. BASIC Functions

Statement	*Purpose*
FRE	Tells how much memory space is left
ASC	Returns the ASCII value of a string
CHR$	Converts number to ASCII character
INSTR	Locates and returns a substring in another string
LEFT$	Returns the left N characters of a string
LEN	Returns the length of a string
MID$	Returns the middle N characters of a string
ABS	Gives the absolute value of a number
ATN	Returns the arc-tangent of a floating point number
COS	Gives the cosine of a number
DEC	Returns the decimal value of a hexadecimal string
EXP	Returns the value of e raised to the X power
FN	Allows custom functions with parameters
HEX$	Returns hexadecimal number string from decimal number
INT	Truncates decimal portion of floating point number
LOG	Returns the natural log of X

Enhanced BASIC 7.0 Statements

You can see the real power of the new BASIC 7.0 when you examine the new statements that have been added to it. Here we will present details on all of these except the new DOS, graphics, and sound commands, which we will explore later. The new statements are made up of bit-mapped graphics commands, sprite control statements, and very sophisticated three-voice sound and music control statements. There are also new looping statements that allow more structured programming constructs, and a new window statement that lets you control the rectangle where text will scroll.

New C128 Mode Bit-Mapped Graphics Statements

Table 3-12 lists the new statements that allow drawing on the bit-mapped display. You can draw lines, rectangles, polygons, and circles, and fill them with any color. You can capture a pixel area to a string (most BASICs capture a pixel area to an array) and then draw it anywhere on the screen quickly. There are statements for reading the color of any pixel, and scaling

Table 3-12. BASIC 7.0 Graphics Statements

Statement	*Purpose*
BOX	Draws a rectangle
CHAR	Displays characters at any pixel coordinate
CIRCLE	Draws a circle, oval, triangle, or polygon
COLOR	Chooses colors for foregrounds, backgrounds, border
DRAW	Draws lines and plots points
GRAPHIC	Chooses one of six graphics modes
LOCATE	Sets the pixel cursor to a certain pixel coordinate
PAINT	Fills any closed boundary with a color
SCALE	Scales the coordinates of the entire screen to 1024
SCNCLR	Clears the screen to the current background color
SSHAPE	Captures pixels of an area in a string
GSHAPE	Draws pixels captured in a string to the screen
WINDOW	Draws a window of defined boundaries
RCLR	Returns the color of a source
RDOT	Returns the color of a pixel on the screen
RGR	Returns the mode of the screen
RWINDOW	Reads the current window boundaries

the entire screen to a 1024 by 1024 grid. These statements and how they work are covered in Chapter 6 — Graphics.

New C128-Mode Sprite Control Statements

One of the greatest enhancements to the C128-mode BASIC 7.0 is its sprite handling statements (see Table 3-13). Sprites, which are covered in more detail in Chapter 6, are programmable graphic objects used often in games and simulations. Since sprites live on independent bit planes that don't erase each other, and since they can be moved by the VIC II chip automatically, it makes great sense to exploit them in programs. Unfortunately, in the C64, sprites had to be controlled via POKE and PEEK statements, rendering sprites quite difficult to control. The new BASIC 7.0 contains a set of clear, simple statements that allow sprites to be easily created, edited, saved, colored, expanded, and moved about on the screen. You can also find out when sprites cross over each other, or when they pass over background display data.

C128 Mode Sound Control Statements

The Commodore 64 has always been known for its powerful SID (sound interface chip) and the incredible noises that can emanate from it: Bach-like sonnets, powerful symphonies, or screeching sound effects. BASIC 7.0 adds a set of powerful statements for easily creating these effects. Table

Table 3-13. BASIC 7.0 Sprite Control Statements

Statement	*Purpose*
BUMP	Records what sprites (1-8) hit what
COLLISION	Causes a GOSUB when sprites collide or sprite hits display data
MOVSPR	Moves a sprite to new coordinates automatically
SPRCOLOR	Sets multicolor colors for all sprites
SPRDEF	Enters sprite editor
SPRITE	Turns on a sprite, colors it, expands size, sets priority and mode
SPRSAV	Moves picture string to a sprite or sprite to a string
RSPCOLOR	Checks what sprite multicolor values last set
RSPPOS	Checks speed and position of sprite
RSPRITE	Returns sprite attributes

3-14 lists the six sound control statements. These are covered in detail in Chapter 7 — Sound. Note that these statements control a three voice music synthesizer. You can create custom envelopes with your own attack, sustain, and release curves, that in turn control one of the three voices. It is easy to have a guitar, drum, and saxophone playing simultaneously. A FILTER statement lets you attach a frequency filter to the output of the SID chip, thereby allowing strange synthesizer effects, like ring modulation.

C128-Mode Program Control Statements

Table 3-15 shows the new program control statements that are offered. There is now a DO UNTIL and DO WHILE looping structure. You can say something like this:

```
100  DO
110  INPUT "DO YOU LIKE YOUR COMPUTER"; ANS$
120  LOOP UNTIL ANS$="YES"
130  PRINT "THANKS"
```

or you can do this:

```
100  DO UNTIL ANS$="QUIT"
.
.
   (statements)
.
200 INPUT "AGAIN OR QUIT"; ANS$
210 LOOP
```

Table 3-14. BASIC 7.0 Sound Control Statements

Statement	*Purpose*
ENVELOPE	Creates instrument envelopes: ADSR, sustain, waveform, pulse width
FILTER	Defines sound filters: low, band or hi-pass, cut-off frequency
PLAY	Defines and plays notes with sophisticated instrument voices
SOUND	Sound effects: any frequency with sweep control
TEMPO	Sets up standard note duration for three voices
VOL	Sets output volume for SOUND statement

You can also say DO WHILE something is true or false. The EXIT statement is provided to let you force a program to exit a DO LOOP when a special condition is met. The ELSE statement has been added to the IF THEN statement to allow more elegant decision statements. The BEGIN/BEND statements are a way of getting around the fact that you can't get a Microsoft BASIC IF THEN statement to allow multiple lines to be executed if the condition is true. Now you can do this in BASIC 7.0:

```
100 INPUT A
110 IF A<100 THEN BEGIN:
120 PRINT "NUMBER IS LESS THAN 100"
130 BEND: ELSE: PRINT "NUMBER IS GREATER THAN 100"
```

PRINT USING, PUTDEF, and GETKEY

The C128 now allows PRINT USING which means you can format how your numbers appear on the screen. This was one of the most missed statements among C64 BASIC 2.0 users. Decimal points can be fixed and so can the number of digits that are displayed. There is also a PUTDEF statement that lets you replace format characters ($, etc) with any character from the keyboard. This means you can display in British pounds, for example. A GETKEY statement is provided that is similar to the GET statement (used to loop on the keyboard and wait for a key to be pressed). GETKEY eliminates the need to loop: it waits on its own for a keypress.

Program Aids

Finally, the new BASIC 7.0 adds all the popular Microsoft program entry and aid commands, as shown in Table 3-16. Take special note that the

Table 3-15. BASIC 7.0 Advanced Program Control Statements

Statement	*Purpose*
DO/UNTIL/LOOP	Structured DO/UNTIL loop
DO/WHILE/LOOP	Structured DO/WHILE loop
EXIT	Causes immediate exit from a DO loop
IF THEN BEGIN/BEND	Allows multiple statements after a THEN
IF THEN ELSE	Allows processing for false and true conditions

TRAP statement allows you to process user errors and turn off BASIC's normal syntax error checking.

What Is DOS and How Do I Use It?

Commodore disk drives are random access mass storage devices. The term random applies somewhat loosely to a method of storing programs and their data (using files and records). The disk drive method of storage has many advantages over cassette storage, but at the cost of increased complexity. With cassette storage you are personally responsible for starting, stopping, and positioning the tape. Manual operations like these are simply inappropriate to the manner of storage used by disk drives. For example, a single floppy disk can store hundreds of programs or data files with thousands of individual records or both. And these programs and files are partitioned into very small data blocks that are spread all over the disk. It is ridiculous to attempt to control a disk device as you would a cassette. This type of storage control and accounting (knowing where things are, etc.) is better left to a computer. When you use a Commodore disk drive with your Commodore 128 you actually get a second computer with its own special Disk Operating System (DOS) program. Its only task is to manage all disk storage activities for you.

What is DOS?

The term DOS is a sort of collective name for a group of single-purpose disk utility programs. These utilities do things like preparing a brand new diskette for future storage operations, reading the current contents of a diskette, copying data from one diskette to another, and erasing old data

Table 3-16. BASIC 7.0 Programming Aids Statements

Statement	*Purpose*
AUTO	Turns on automatic line numbers
RENUMBER	Renumbers line numbers of program
DELETE	Deletes groups of lines
HELP	Shows where line number error occurred on screen
TRAP	Branches to subroutine containing user error handler
TRON, TROFF	Trace mode: displays line numbers as program runs

to make room for new data. Some DOS utilities have disk operations whose actual functions are less apparent. These "specialty" DOS operations are the ones requested by programs operating in the C128. For example, if a mailing list program needs to see the list of names and addresses already stored on a diskette, the mailing list program must make a series of properly organized requests to the disk drive's DOS. The DOS, in turn, uses the requested DOS operations to locate and then send back this data to the program. DOS operations like these are seldom recognized or even used by C128 users who primarily use commercially written programs, but they are important for C128 users who want to write their own programs. Commodore DOS has a number of disk operations that are primarily used within other C128 programs. The most familiar DOS operations, however, are those used by every disk drive user at one time or another.

Where is Commodore DOS Kept?

Commodore disk drives are *intelligent* disk drives. They are intelligent because they have their own internal computer, running under a special *Commodore DOS* program that allows them to perform routine disk functions and maintenance. What's more, since this DOS program is actually stored in the intelligent disk drive's memory, the drive can do any DOS operation on its own. Most computers need to instruct their drives with a continuous flow of instructions. All the C128 need do is send a single *command word* or phrase over the serial bus to a Commodore disk drive. The disk drive's computer compares this command against a list of known commands within its own DOS program, selects the appropriate disk function, and the proper DOS operation is performed. Since the Commodore disk drive does all the real DOS work, the C128 need only pass along the appropriate commands and then go back to its normal processing activities.

Commodore DOS is stored in ROM in the disk drive. The disk drive has its own computer, with a microprocessor, RAM and ROM memory, and I/O circuits. One advantage to storing DOS in ROM, as opposed to putting it on a diskette, is that a ROM-based DOS is there and ready for use as soon as the disk drive is turned on. Another advantage is that the memory space needed to store DOS is taken from the disk drive's internal computer, not the C128's. This means there will be more room in the C128's memory for application programs. Finally, keeping DOS in ROM eliminates the need to keep it on your diskettes, therefore all of the diskettes' available storage can be used for programs and data.

Are All DOSs the Same?

Commodore has three different disk drives that you can connect to the C128 — the 1541 (designed for the C64), the 1571, and the 1572 (designed to work on any current Commodore computer). All three drives have DOS already built-in. The 1541's DOS is slightly different than the DOS found in the 1571 and 1572, but there is a high level of compatibility between the two. They are programmed by the user in the same way, but their capabilities differ.

Many of the differences between the old and the new DOS have to do with the slower serial bus capabilities of the Commodore 64 and the fact that the 1541 could only access one side of a floppy disk (single-sided). The new drives are precision instruments that are able to take advantage of the C128's faster serial bus capabilities. They are double-sided (that is, they use both sides of a floppy disk). Despite these differences, the new drives can read and write to any disks originating from a 1541 (only one side of the disk is used). They do this by changing their DOS "personality" to that of a 1541.

What Tasks Can the DOS Perform for Me?

Since DOS is stored in the Commodore disk drive, it is running as soon as you turn on the disk drive's power. Thereafter, in order to get DOS to do something for you, all you need to do is get its attention and send it a command. Fortunately, the C128 has a complete set of easy to use DOS commands and statements that are accessible through BASIC 7.0. You will note that many of these commands and statements are similar to one another. This slight redundancy was necessary in order to remain compatible with the DOS structure used in BASIC 2.0 (Commodore 64), while providing an alternate set of disk-based commands and statements that are easier to remember and use. Here is a list of these new DOS commands and statements:

APPEND	CONCAT	HEADER
BACKUP	COPY	LOAD
BLOAD	DCLEAR	OPEN
BOOT	DCLOSE	RECORD
BSAVE	DIRECTORY	RENAME
CATALOG	DLOAD	SAVE
CLOSE	DSAVE	SCRATCH
COLLECT	DVERIFY	VERIFY
	DOPEN	

Let's review these DOS commands and statements one by one. Those that are referred to as "commands" can be entered directly on the keyboard and can often be included within a program as well. Those listed specifically as "statements" are generally used from within programs only, so they will be of most use to C128 owners who plan to do their own programming using BASIC 7.0. The descriptions we give here are just meant to give you a feel for their functions. Complete details on the actual syntax and modifiers can be found in *C128 Users Guide*.

APPEND

The APPEND command is a special "open file" command that is used to add new data to the end of an existing file.

BACKUP

The BACKUP command copies all files from one diskette onto another. This command can only be used with a dual drive system like the 1572 and it will not back up "copy protected" software.

BLOAD

This command is used to load *binary*-type files from a disk. The C128 has two disk specific file loading commands — DLOAD and BLOAD. DLOAD is used on files such as BASIC programs that are always placed in the same area of memory. However, not all programs are in BASIC so they don't necessarily go to that same portion of memory. Binary files can go just about any place in memory. Where they go during a load should be determined by the load address parameters that BSAVE automatically tags onto them when they are originally saved. BLOAD reads these parameters and puts the binary files back where they belong (see BSAVE).

BOOT

BOOT is a completely new DOS command for Commodore computers. It is used to load (and subsequently run) an executable binary file from a disk. Like BLOAD, it does not automatically relocate the file to the start of BASIC memory.

BSAVE

BSAVE is the binary file variation of DSAVE. It is used to save a segment of C128 memory to a binary disk file. Files created in this manner must be not be loaded with DLOAD as they would be automatically relocated to BASIC memory. Instead they should be loaded with BLOAD (see BLOAD).

CATALOG

This command is used to read the contents of a diskette's file directory. Its operation is quite different from the C64 method of viewing a directory — LOAD"$",8:LIST. The old command did not work well on long directory listings because you could not stop and start the display once it started scrolling. Also the old method actually loaded the directory into memory as if it were a BASIC file, thus overwriting any BASIC programs already in memory. The CATALOG command corrects both of these deficiencies: its display scrolling can be controlled, and it is loaded directly into screen memory so as not to disrupt BASIC programs (see DIRECTORY).

CLOSE

This statement completes and closes any files previously opened by either DOPEN or OPEN statements. CLOSE is normally used only in programs and is applicable to files of all types, including those meant for a printer or modem.

COLLECT

This command is used to reclaim disk space that is currently allocated to improperly closed files.

CONCAT

The CONCAT command (derived from the word concatenate) is used to merge two data files together.

COPY

COPY is used to make copies of files. It can be used to copy from one disk to another (which requires a dual-disk drive) and it can be used to create a second copy of a file under a different name on the same disk.

DCLEAR

This statement clears the contents of all currently open channels to disk files. It is used by programmers to insure the removal of any old data that may have been left in a channel from some previous file operation (see DCLOSE).

DCLOSE

DCLOSE is used to close one or all currently opened channels to the disk drive. The term *channel* is used to indicate a specific communication

path between the C128 and the disk drive's internal computer. Under normal operation there can be a number of channels open (each to different files), but there is a finite limit to just how many are open at any one time. The DCLOSE statement is issued by a program to close channels that are no longer needed (see DOPEN).

DIRECTORY

The DIRECTORY command displays a disk directory on the C128 screen (same as CATALOG).

DOPEN

This statement is used to open a communication channel to a disk drive. It is a disk-specific version of the OPEN statement.

DSAVE

The DSAVE command saves a program currently located in BASIC memory to a disk file. It is a disk-specific form of the SAVE command (see BSAVE).

DVERIFY

This command is used to compare a program in memory with one in a disk file. This is a disk-specific version of VERIFY.

HEADER

The HEADER command is used to prepare a new diskette for its first write and read operations. It formats a blank diskette into storage blocks and then establishes a fresh block allocation table and a blank disk directory. It also gives the disk a specific disk title and ID code. This is a destructive command as it totally erases any files already on a diskette.

LOAD

LOAD is used to load a file from either a Dataset or a disk drive. Variations of this command can be used for regular file types (BASIC programs) and binary file types (non-relocatable machine language programs of data) (see SAVE).

RECORD

This statement is used from within a program as a relative file pointer to select any byte of any record in a relative file.

RENAME

The RENAME command is used to change the name of a disk file. You should note that no two files on a single diskette can ever have the same name.

SAVE

This version of DSAVE can be used to save a file to any legal Commodore destination device such as a dataset, a screen, a modem, a printer, or a disk drive. It can only be used to save a program located in the standard BASIC memory area. Binary-type programs must be saved using the C128's built-in MONITOR utility (see LOAD).

SCRATCH

This command is used to erase or delete a file or group of files from a diskette. It is recommended that it be used to delete an existing file before attempting to save another file by that same name. Commodore DOS will not allow you to save a file if one by that name already exists.

VERIFY

This version of DVERIFY works for files found on the Datasette as well as those from a disk drive.

Can DOS Be Changed?

The DOS we've referred to thus far is one of two possible DOS systems that you can use with the Commodore 128. This particular DOS is the one already stored in the Commodore disk drives and is the easiest to access. It is also the only DOS you can use when the Commodore 128 is used in the C128 or C64 modes. However, the built-in DOS can be replaced with another DOS called CP/M. How, why, and when you would want to use a different DOS will be explained in Chapter 5.

Where Can You Learn More About the C128?

No matter how "friendly" and reliable a computer is, and how complete its operations manual may be, its owner will always want to have access to more in-depth documentation, special training, service, and a number of other forms of system support. Fortunately, there are numerous resources available that answer these needs for Commodore 128 owners.

Books You Should Own

If the past Commodore 64 market is any indicator as to the number of books that will be written about the Commodore 128 then you will have a large number of books to choose from. Titles will cover such subjects as C128 BASIC programming, getting started with the C128, inside the C128, applications for the C128, using C128 graphics and sound, business and the C128, and machine language programming. In addition to specific interest books like these you will also find a large assortment of books containing interesting programming hints and software. A quick check at your local book store will reveal an ever-increasing number of C128 book titles for you to choose from.

Magazines for Commodore Computer Owners

Commodore 128 owners will also have a wide range of magazines to choose from; many of them are completely devoted to Commodore computer owners. Their editorial formats differ sufficiently so that you can easily choose one or two magazines that are particularly suited to your needs. Here is a partial list of Commodore oriented magazines listed from those attempting to primarily reach the beginning C128 users to those whose aim is to reach the more sophisticated C128 hobbyists and programmers.

Commodore Power Play by Contemporary Marketing, Inc.

Commodore Microcomputers by Contemporary Marketing, Inc.

COMPUTE!'s Gazette by COMPUTE! Publications, Inc.

COMPUTE! by COMPUTE! Publications, Inc.

RUN by CW Communications/Peterborough, Inc.

The Transactor by Transactor Publishing, Inc.

Commodore User Groups

One of the best and easiest organizations for new Commodore 128 owners to get help from is their own local Commodore users group. There are over 600 of these organizations spread throughout 16 countries and 50 states. For example, there are 66 of these clubs in California alone. Complete lists of these user groups with their addresses are periodically published in magazines like *Commodore Microcomputers* and *RUN*.

A computer user group is basically a club that is open to all parties who share common interests. The typical membership of a user group will be primarily new or fairly new computer users. Their goals will be to help each other by sharing information. Many user groups sponsor classes in applications and programming and maintain a library of *public-domain* (free to the general public) software. Some Commodore user groups also maintain their own electronic bulletin boards (self-answering computer systems contacted with a C128 and a modem). These electronic bulletin boards provide many of the same functions as the clubs themselves, like question and answer forums and software libraries.

The Commodore Information Network

The Commodore Information Network is available to anyone with an interest in Commodore computers who has access to CompuServe, an electronic database system. Unlike other electronic bulletin boards, this network is maintained and run by the Commodore corporation itself. The Commodore Information Network is actually broken down into a number of smaller *special interest groups* (SIGs) that are quite distinct from one another. Each has its own help forum, a message database and a software library. More information about the various Commodore SIGs can be obtained by calling CompuServe, making a modem connection with your C128, and then following CompuServe's system of prompts till you get to their personal computing section (those more familiar with CompuServe's operation can use the command GO-PCS at any prompt). Details on CompuServe can be obtained from any computer products store.

Software Directories

Another publication of interest to Commodore 128 owners is the *Commodore Software Encyclopedia* published and periodically updated by Commodore Business Machines. This encyclopedia contains hundreds of references describing the many different software products currently available for Commodore computers, including those for the Commodore 128.

4

The C64 Mode

In this chapter you'll learn:

- **What the C64 mode is**
- **What the C64 mode has to offer**
- **What you will need to use this mode**
- **About popular Commodore 64 software products**
- **How BASIC 2.0 differs from BASIC 7.0**
- **The C64 mode DOS structure**

The Commodore 128's second operating mode is its C64 mode. In this chapter we'll explain what this mode is, why it is important to you, what external devices you will want, and how you get into and out of this mode. Then we'll describe a number of interesting Commodore 64 software products that cover a broad spectrum of applications. In the last part of this chapter we'll briefly state some of the changes that occur to the Commodore 128 when used in this mode and we'll identify some of the BASIC programming compatibility issues. We'll round off our discussion of this mode with a fast look at the C64 mode's changes to DOS users. Let's begin by defining just what the Commodore 64 mode means.

What is the C64 Mode?

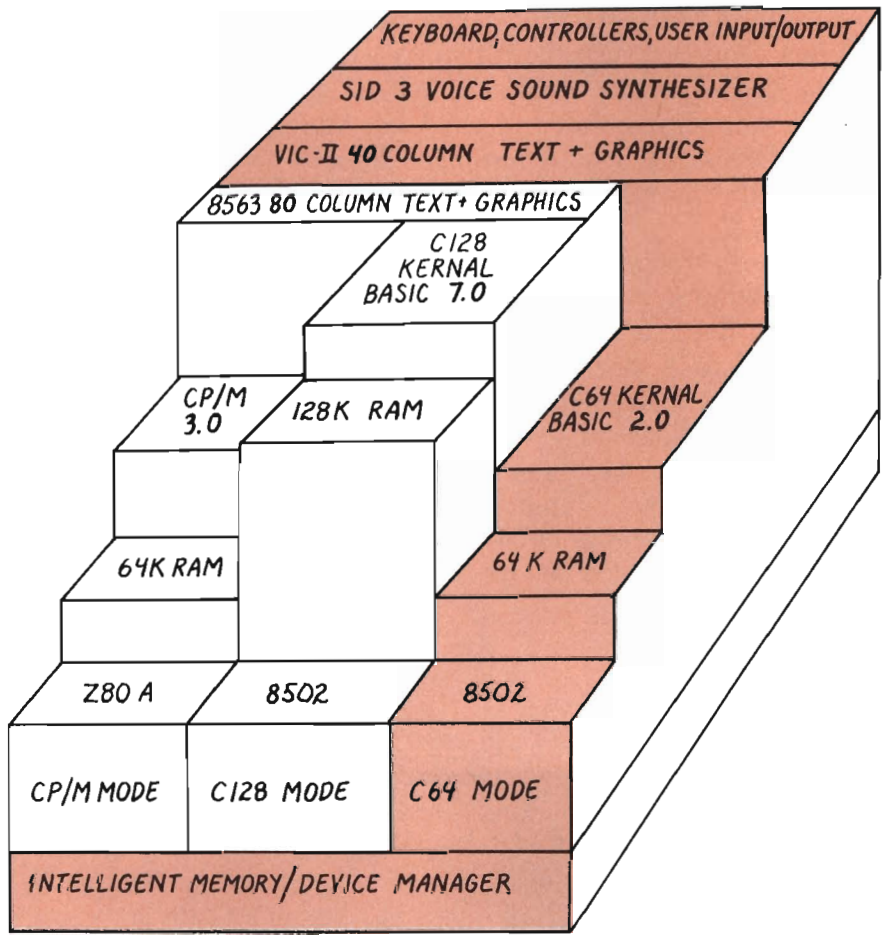
In the last chapter we explained how the Commodore 128 had the internal components for three different computers. One of those component arrangements forms the C128 mode; a second component arrangement forms the C64 mode. This new arrangement, shown in Figure 4-1, reorganizes the Commodore 128's internal hardware and software parts to form an exact replica of a Commodore 64. Thus, when you use the C128 in its C64

mode you are really using a Commodore 64. Naturally, how well you understand the implications of this mode will depend on how familiar you are with the Commodore 64. If you are not already familiar with what the C64 is, then a very quick review of this product is in order.

The Commodore 64 History

The period between 1979 and 1982 marked Commodore's first big success in the personal computer market — this was the era of the VIC-20 personal computer. The VIC was a small innocuous computer sporting a typewriter-

Figure 4-1. How the C128 Stacks Up in the C64 Mode



like keyboard, with built-in BASIC, vivid color graphics, pleasing sound effects, and 5K of RAM. It also had just about the lowest price tag ever seen for a computer. Commodore's price structure for the VIC-20 was no accident: it was designed to compete against home video game machines like the Atari 2600. The strategy worked well and by late 1981 over one million people had succumbed to the VIC's charm and low price. However, since technology does not stand still, Commodore knew that if it was to remain number one in the home computer market it would have to create a new and better computer by 1982.

The period from 1979 to 1981 was also the peak of the home video game craze. It was during this time that Commodore's semiconductor chip subsidiary, MOS Technology, began designing a series of super-powerful video game graphics and sound chips. In 1981 it completed the design of two high-technology microchips: the 6581 Sound Interface Device (SID) and the 6567 Video Interface Chip II (VIC-II). Together these two chips were capable of truly extraordinary graphics and sounds. Commodore originally intended to use these two chips as cornerstones for its next generation of video game machines. Instead, given the phenomenal success of the VIC 20, it decided to use these chips as central components for a new 64K-byte personal computer. No one had yet offered a home computer with this memory capacity. Thus the Commodore 64 concept was born.

When the Commodore 64 entered the home computer market in 1982 it had just the right combination of built-in features to favorably compete with the then popular Atari 800, TI 99/4, and Apple II computers. Like the VIC-20 before it, the C64 was priced significantly below its competition. Because of its low price and immense advertising campaign, the C64 quickly became a household word. By 1985 over two million people had bought a Commodore 64 computer system and over six thousand pieces of software had been developed to run on it. The only other computer to come close to this sales volume has been the Commodore VIC-20. The Apple II and CP/M machines are the only other computers to have so much software.

The C64 Concept

Commodore had a number of different considerations when it designed the Commodore 64 computer — these considerations determined the eventual shape and success of the C64.

Commodore chose the C64's design features to make it as attractive as possible to the *home* and *school* computer markets. It already had successful computers in both of these markets — VIC-20s in homes and

PETs in schools. Commodore merely had to merge some of VIC's and PET's more successful attributes in order to place the C64 in both markets. The Commodore 64 also had to have an impressive assortment of new features to make it more attractive to those same buyers. The C64, its peripherals, and its software also had to be inexpensive. This meant that Commodore had to perform some real hardware and software engineering feats if the finished C64 was to cost less than its competition and perform better.

Here is a list of features that the Commodore 64 borrowed from the VIC-20 and PET:

6502-based architecture

Kernal operating system

BASIC version 2.0

Typewriter style keyboard

40-column text screen

PET graphics

Cassette interface

Serial BUS for disk drives and printers

User port for peripherals such as modems and voice synthesizers

Here is a list of features new to the C64:

Color graphics

Sprite graphics

Three-voice eight-octave sound synthesizer

CP/M option

Redesigned cartridge slot

Direct high quality video and audio outputs

RF modulator

64K of RAM memory with advanced memory management

Two game controller ports for joysticks, light pens, etc.

New family of inexpensive peripherals — color monitor, disk drive, printers, and modems

As you will see from these lists, Commodore packed the C64 with a collection of important features. It introduced the C64 at $595 in 1982;

by mid-1983 this price had dropped to $300, and by mid-1985 it had dropped to $150.

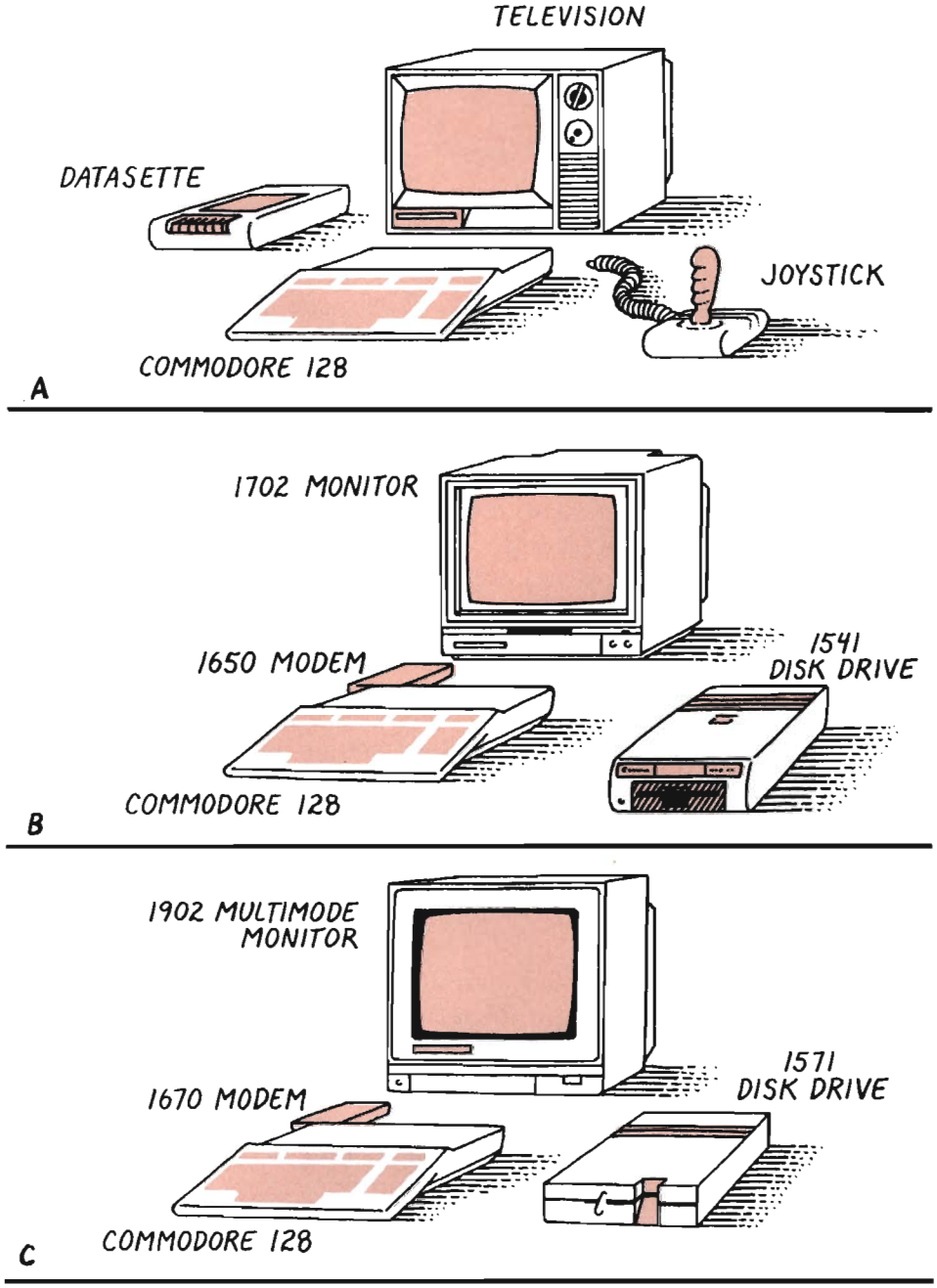
What External Devices Do You Need to Use the C64 Mode?

At the very least, to use the C64 mode you will need a 40-column compatible monitor or television. The kind of monitor you choose will mainly depend on what you want to do while your Commodore 128 is in its C64 mode. In this mode you can select from the same list of accessories as would a Commodore 64 owner. You can even use new Commodore accessories, like the 1902 color monitor and the 1571 and 1572 disk drives (designed specifically for the C128 and described in Chapter 2), as they have C64 modes as well.

Naturally, what equipment you will want depends on the C64 software you will use. If your interest in the C64 mode lies strictly in using C64 cartridge game and educational software, then all you may need is a monitor or a television set and maybe an external controller, like a joy stick or a set of game paddles. If you want to use cassette-based C64 software then you will need a Commodore Datasette. If you want to use more powerful disk-based C64 software then you will have to have one or more disk drives. Whether or not you will want other devices, like a printer or modem, will depend on the needs of your C64 software. For example, if you plan to use your computer for word processing then you will want a printer, but if you are only interested in entertainment software like arcade games then you wouldn't. You would want a modem if you planned to use your computer to access remote computer systems like CompuServe or The Source.

As you can see there is no easy solution to the question of which external devices you should connect to a Commodore 128 when it is used in the C64 mode. Three of the many possible system configurations are shown in Figure 4-2. The first example is really a minimum system designed for casual programming and entertainment. In this system a television is used as a monitor, a Datasette as the mass storage device and a single joystick is used for an alternate input device. The second example illustrates how a C64 owner who already owned a full C64 system upgrades it by simply exchanging the C64 for a C128. A Commodore 1702 is used as the monitor, a 1541 disk drive is used for mass storage, and an MPS-801 is used as the system's printer. Finally, the last example is a C64 mode system using new C128 system components; 1571 disk drive, 1902 multi-mode monitor, an MPS-802 printer and a model 1670 modem.

Figure 4-2. Three Possible Configurations for C128 Systems in the C64 Mode



Naturally, there are other system configurations than those just mentioned; just remember to base your external equipment selection according to the needs of your C64 software (consult the instructions provided with your C64 software for its hardware requirements). Then, if you are still not sure of your total options concerning the different peripheral devices, refer back to Chapter 2.

How and When Do You Change to the C64 Mode?

You must first be in the C64 mode before you can load or run C64-specific software. There are two ways to tell the Commodore 128 to change to its C64 mode. When the computer is in its C128 mode (normal start up) you can simply command it to go to its C64 mode by typing "GO 64." The second way to change to the C64 mode is to turn on the C128 with any Commodore 64-type software cartridge plugged into the cartridge slot (the expansion bus). In either case, once the C128 is switched to the C64 mode, there is no way to command it to go back to its C128 mode (the C64 did not have a GO 128 command, so to be truly compatible, neither can the C128 when used in its C64 mode). The only way to return to the C128 mode is to turn the C128 off and on again.

What Can You Do While in This Mode?

One of the main reasons for using the C64 mode is that so much high-quality software already exists for the Commodore 64 computer; software which can be used without modification in this mode. There are software programs for business, home, and school applications. In this section we're going to look at some of the different types of Commodore 64 software products. We'll somewhat arbitrarily divide this world of software into five categories: personal productivity, education, edutainment, entertainment, and software development.

Please note it would be impossible for us to cover every piece of software in these categories. To do so would require a book in itself (a very large book). So rather than make this section a "buyer's guide to software," what we want to do is give you a quick glimpse of the kinds of software that are available, and perhaps refer to a few popular products in each category — or to products with interesting or unusual features — as examples of what is available. There are so many excellent C64 software products already available, and new ones are being created all the time, so you should conduct your own research before deciding what to buy.

We do not list the prices of the products referred to in this section, since they change so fast over time and also because they vary from dealer to dealer. In general, programs in the personal productivity and software writing categories can be expected to cost from $50 up to about $150, while games and other entertainment software will cost from $20 to $50, with educational programs somewhere in between these ranges. These prices are often considerably below comparable software products available on more expensive computers.

Personal Productivity Software

By personal productivity software we mean those programs which help you personally to accomplish various tasks better or more efficiently at home or at work. Word processors, spreadsheets, databases, personal finance packages, and telecommunication programs fall into this category.

Word Processors

Word processing is probably the most popular serious use for personal computers such as the Commodore 64 and Commodore 128. A word processing program lets you type a document (which can be almost anything you'd write on a typewriter — from memos to novels) on the screen of your computer. Your screen provides a window into a much longer document. You can modify this document as much as you like, by inserting or deleting words or sentences, until it is just the way you like it. At this point you can print it out on a printer.

Word processors typically have commands to insert and delete individual letters, words, sentences, and paragraphs, to write a document to the disk, to read a document already stored on the disk into the computer, to set margins and tabs, to automatically number each page, and to print a document on the printer. Most word processing programs have a multitude of other features as well. With many programs you can search for specific words or phrases in your document (in case you decide to change every occurrence of "Jane" to "Mary", for example), underline words or phrases, and format the printed output in a variety of ways.

Most word processors use some form of automatic *word wrap*. This means that when you get to the end of a line, the program automatically breaks the line at the end of a word, and moves the word you're typing down to the next line, so you can keep typing without having to press Return or any other key.

A useful feature available on many word processing programs is the ability to *merge* parts of one document into another. This is typically used

in form letters, where one document consists of a list of names and addresses, and another document consists of a letter which is to be sent to everyone on the list. A variety of forms can be handled in the same way.

There are a number of very good word processing programs available for the Commodore 64. Bank Street Writer is renowned for its ease of operation: it is a favorite of school-age children and adults who rarely use a wordprocessor enough to get familiar with all the operations typical of larger full-featured word processors. Speedscript, a program available through COMPUTE! Publications, Inc., has more features, but not at the expense of its ease of operation. Speedscript is also an excellent choice for casual word processing, especially if you consider its cost — basically the price of a single magazine. If your word processing needs require more features than either of these two programs offer then you have a wide list of professionally competitive products to choose from. Easy Script from Commodore and WordPro 3 Plus/64 are two typical examples. Let's get a better look at what you get with one of these more elaborate and more expensive word processors.

WordPro 3 Plus/64, from Professional Software, has 23 format commands and 47 control functions which basically equate to a large number of word processing operations. In addition to the word processing functions available in smaller programs such as Speedscript, WordPro can perform such tasks as printing sub-and super-scripted text, adding and subtracting columns of numbers in your text, and using boldface type. WordPro also has its own spell checking dictionary — a very nice feature typical of these better word processor programs. WordPro is a complicated software package, but like most quality software products it comes with a very well written and complete users manual. There are also sample documents available on the disk to help you get started.

Spreadsheet Programs

The very first spreadsheet program, VisiCalc, was instrumental in establishing the personal computer as a serious business tool. The existence of this program sold millions of computers to American businesses, and changed the way people thought about personal computers. This same capability is available on the Commodore 64.

What is a spreadsheet program? Essentially, it's an electronic equivalent of an accountant's spreadsheet: a piece of paper with rows down the side and columns across the top, but electronic spreadsheets automatically perform whatever calculations are needed. Spreadsheets are used for a variety of purposes. For instance, you might use a spreadsheet to show

different years in the columns across the top, and sales figures for different items in the rows listed down the side. At the bottom of each column you might show totals for all items sold for each year. On the right you might show the percent of increase or decrease in the sales of particular items, over the number of years shown across the top. Once you have designed a spreadsheet *template* you can use it over and over again just as it is, or modify it to fit future needs.

Microsoft MultiPlan is one of many spreadsheet programs available for the Commodore 64. It is attractive to many C64 owners because it is also available on computers like the IBM PC and the Apple Macintosh. The version for the Commodore 64 retains many of the same features. Multiplan comes with a large, well written manual, a nice feature if you're new to spreadsheets.

Calc Result from Handic Software is a somewhat less ambitious program. It's easy to learn and offers a number of advanced features. It comes in two versions. Calc Result Easy gives you a capacity of 1000 cells, while Calc Result Advanced gives you 2000 cells and a number of extra features. Both versions are equally easy to learn and use.

PractiCalc from Computer Software Associates is a smaller and simpler spreadsheet program which has the advantage of being such a small program that it leaves a great deal of room in memory for storage of the actual spreadsheet. It's also one of the least expensive spreadsheet programs around, which makes it a good way to get started in spreadsheets if you're on a limited budget.

Database Programs

Database programs, like The Manager from Commodore, find a wide assortment of uses with C64 owners. Basically a database is like an electronic filing cabinet that you fill with reports you design. These reports can hold the particulars of daily sales reports, a stamp collection, favorite recipes, abstracts from books and magazines, or mailing lists of all your family, friends, or clients. Once the records are in the database you can use the speed of your computer to sort them, search them, or even update them. In some ways databases are like spreadsheets in that they require some initial setup work by you. After all, you have to design the electronic report forms for the database before you can use it. Fortunately, programs like the Manager come with a few sample report templates and easy-to-understand instructions.

Personal Finance

As the Commodore 64 was primarily aimed at the home computer market, it is no surprise that there are a number of software packages that deal expressly with personal financial matters. The Personal Accountant and the Home Accountant, from Continental Software, are two such programs. Both of these programs let you do such things as balance your checkbook, print checks, establish budget accounts, and keep a record of your financial status. Tables and easy-to-understand graphic displays are used to show rates of change or comparisons. It is even possible to take the financial data eventually stored by these programs and pass it to a tax preparation program at the end of the year.

Telecommunication Programs

Hundreds of thousands of Commodore 64 owners use their C64s to communicate with other computers and they do this with the aid of modems and modem software. Telecommunication programs, as they are commonly called, come in a variety of flavors, but underneath they are quite similar in what they do for you. Their primary function is to teach your computer how to send and receive data from a modem (the modem provides your C64's physical connection to the telephone line).

The simplest telecommunication programs are those that transform the C64 into a "dumb terminal." Dumb terminals have just two functions — one is to provide a keyboard that you use to send data to the modem and the other is to display for you the data received by the modem. You can't use a dumb terminal to store incoming messages or to edit and arrange your messages before sending them out, so their applications are basically limited to casual telecommunications. This kind of C64 telecommunications software is typical of what you get as free bonus software when you buy a modem like the 1660 or 1670 from Commodore.

If your telecommunication needs are more demanding, then you might want a program like Super Term from Midwest Micro, Inc., or Smart 64 Terminal Plus 3 from Micro Technics Solutions Corp., as they can provide you with better telecommunication capabilities. For instance, both of these programs let you use the C64's mass storage capabilities to save incoming data or to hold data that you want to send. They also let you *dump* incoming data to a printer. Other features supported by programs like these might include auto-dialing, and binary file transfers with special *protocols* that insure the integrity of file transfers. Overall, these kinds of telecommunication programs are the most flexible, but they are often difficult to learn and use.

If your telecommunication needs are limited to using CompuServe then you would want to use VIDTEX software. This special program will do everything the other general purpose programs can do with one important enhancement — VIDTEX provides a universal operating interface between CompuServe and you. This interface has one purpose and that is to ease the burden of telecommunicating to CompuServe and it does this in a number of important ways. For example, with VIDTEX your C64 automatically understands the special screen codes that CompuServe sends out to clear screens and move cursors. This makes your screens appear cleaner and easier to read. These screen codes will also let CompuServe send more information in less time, thereby saving you money. VIDTEX also has automatic features that make it easier for you to send and receive ASCII and binary files. VIDTEX has many other automatic features designed to simplify a number of CompuServe activities.

Education

Probably the second largest category of software for the Commodore 64 after games and entertainment is educational software. Within this category you will find a number of smaller categories such as preschool, ages 5-9, ages 9-13, ages 13-17, and adult. The subjects of these programs range from basic language skills (such as nouns, verbs, adverbs, and adjectives) to subjects such as geography, science, chemistry, and mathematics.

Over a hundred general education programs are already in the public domain and can be obtained free of charge from most larger Commodore user groups or Commodore computer dealers. Hundreds more of these programs can be obtained from listings in Commodore-oriented magazines like *RUN*, *COMPUTE!'s Gazette*, *Commodore Power Play*, and *Commodore Microcomputers*. There are even a number of different books filled with Commodore 64 educational software. High-quality educational software is also available commercially. For example, Davidson & Associates markets SPELL IT!, MATH BLASTER!, WORD ATTACK! and SPEED READER!, all excellent packages. Educational programs for the C64 extend to the college level with programs like THE PERFECT SCORE from Mindscape, a program specifically designed to prepare you for college SAT exams. There are even C64 educational programs for personal self-improvement like Timeworks' Evelyn Wood Speed Reading, which you can use to improve your reading comprehension, retention and speed.

Edutainment

Edutainment is a newly coined phrase that describes a whole new category of Commodore 64 software. Edutainment programs are specially designed to teach a specific subject and in a fun way — *edu*cation and enter*tain-ment*. Many programs fit so well into this format that it is often difficult for people using the program to realize that they are actually in a controlled learning environment — they are simply having too much fun. Edutainment programs are available for virtually all age groups. One excellent edutainment program for preschoolers is Learning With Leeper from Sierra On-Line. Leeper teaches basic skills like hand-eye coordination; letter, number, and object identification; counting; problem solving; and artistic expression. For the teenager, try Type Attack by Serious Software. This unusual program teaches touch typing in the guise of a fast-paced arcade game! A new entry to C64 edutainment is Trivia Fever by PSI. This engaging package offers thousands of challenging trivia questions from seven categories with three levels of play. Up to eight people can play it. Edutainment doesn't stop here as there are educational games designed to teach a number of diverse skills, such as word processing, programming, and spreadsheet management.

Entertainment

Entertainment is the largest category of available Commodore 64 software. In fact, somewhere between 60 and 70 percent of all Commodore 64 software is entertainment oriented. This very large base of software can be divided into a number of categories. *Arcade games* make up the largest group of entertainment programs and *adventure games* make up a significantly smaller group; some games fit into a *simulations* category, and others fit into an area called *art and music recreation*. Programs from these different entertainment subcategories can be obtained from a wide assortment of sources. Of course a large number of them are commercially prepared and marketed programs, but hundreds more are available through magazines, books, and local Commodore user groups. We can't possibly describe or list all of these highly entertaining software products here, but we can give you a brief idea of what it is like to use some of them.

Arcade Games

Arcade games — such as Pitstop II, Beach Head, and Impossible Mission — are designed to stimulate your sense of sight, hearing, and touch in an exciting and captivating way. Pitstop II, for example, puts you in the

driver's seat of a powerful race car which you take to various international car-racing circuits. In this game you are the driver looking out the race car's windscreen as you speed along the winding, twisting tracks. Beach Head is a war game in which you have a number of objectives to accomplish — such as surviving a kamikaze attack — before you land your forces on the enemy's highly fortified beach and try to take it. Impossible Mission pits you against an evil scientist in his underground labyrinth of tunnels and rooms. Your objective is to survive his evil robots and numerous other traps, and at the same time to gather enough hidden clues so that you will know how to turn off his doomsday device.

Arcade games are typically very fast paced and methodical in their approach. For example, all of their activities will be focused on some scenario laced with a heightening pattern of obstacles and rewards. Competition is also a key element in their design. Performance scores, based on elapsed time and goals completed, are used to show comparisons and progress. You may be competing against what you did the day before or directly against other players. Arcade games are eye-, ear-, hand-, and mind-oriented, therefore they provide an excellent exercise for improving manual coordination and concentration.

Adventure Games

The very nature of adventure games makes them totally different from arcade games in they way they look, act, and play. Unlike the repetitive, quickly learned, and often-repeated events of arcade games, adventure games work to continually stimulate your curiosity and imagination. They challenge your ability to solve an ever-widening assortment of riddle-like problems. Adventure games are not fast-paced or sense-stimulating. They are mind games designed to test your thinking and reasoning faculties in ways that arcade-style games simply don't. Adventure games require a great deal of concentration and a sizable investment of time.

The game scenario is the most important ingredient of a good adventure game. On the surface it must be easy to understand, but underneath it must have a highly complex inner structure filled with a wide range of event probabilities. What happens during the game depends a lot on what you do (you control the actual threading of events). There are adventure games like Zork I, Zork II, and Zork III that take you into the completely imaginative world of science-fantasy where you battle ghastly creatures and seek vast treasures. There are other adventures that throw you deep into the world of science fiction where you must learn how to survive in the deep reaches of space (Suspended). There are two adventures based on Agatha Christie-like stories that bring out the real sleuth in you (Dead-

line and Eye Witness). The Zork series, Suspended, Eye Witness, and Deadline are all products of Infocom.

Simulations

Another category of entertainment games is that called *simulations*. These programs combine all the computing, graphics, and sound power of the Commodore 64 to create a true-to-life simulation of a complex, mind-coordinated, motor-reflex, real-life activity — such as flying an airplane. We are not referring to the typical arcade game where you control a little biplane and make it climb, dive, turn, and land with simple motions of the computer's joystick, but to a real-life flight simulator where you are in the cockpit surrounded by realistic-looking and completely functional instruments. In front of you and to your sides you see the sky and land around you change as your plane moves, and this view changes in correct response to your handling of the plane's throttle, ailerons, flaps, and rudder. There is only one way to win at this type of game and that is to gain a lot of the same knowledge and skills actually needed to fly a real plane.

There are a number of simulation games available for the Commodore 64. Flight Simulator, from subLOGIC Corporation, and Solo Flight, from MicroProse Software, are two simulators that put you at the controls of your own airplane. Subwar 64, by Clockwork Computers, Inc., takes the art of simulation into the conning tower of a submarine. In this simulation you are given the choice of a number of different roles. For example, you can be an observer along for the ride, or a Ship Control Trainee, or a dive control officer, or even an attack center officer. Each role you assume has its own set of unique responsibilities.

Art and Music Recreation

The last area of entertainment programs for the Commodore 64 contains programs designed as noncompetitive recreational activities. These programs let you use the Commodore 64 as an extension of your own creative talents. You can paint people or landscapes with programs like Koalapad, create and animate your own computer graphics with Sprite Maker, listen to computer-generated music, play music, or even compose music of your very own with programs like MusiCalc and the Kawasaki Synthesizer. Of course, these are just a few examples of the dozens of art and music programs available for the C64. If your personal needs are for more relaxing forms of entertainment, then you will definitely want to try a number of these unusual programs.

Software Development

There are a number of Commodore 64 programs that help you develop, write, and test your own C64 programs. Some work right alongside BASIC 2.0 while others totally replace BASIC with an entirely new programming language. Many of these programs are complete software development systems; others are utilities designed to perform a single, but necessary, program development function; some are used as add-on enhancements to existing development systems; and a few are primarily aids that reduce some of the tedium normally associated with programming.

Development Systems

If you want an alternative to BASIC 2.0, there are a number of different software development systems that you can choose from. For those who need to write extremely fast programs that are compact in size, there are a number of different assembler development systems to choose from. For example, Commodore has its own Assembler 64 package. This package includes a full macro assembler, editor, loader, and two machine language monitors. If you want to retain many of the advantages of assembly language programming, but also have some of the benefits unique to an interpretive language then you might want to use a Forth development system such as SuperForth 64 from Parsec Research. This Forth implementation includes a full floating-point math package and it has the ability to create stand-alone applications (programs you can run without Forth). However, if you don't mind sacrificing program speed and some programming flexibility for an easier-to-use and –understand program structure then you might want to try a more structured compiler-based language such as C, Pascal, or Fortran. Excellent implementations of these languages are available from Abacus Software. You may choose Logo or Pilot as your software development tools. Both of these excellent interpretive languages are available from Commodore for the C64.

Programming Utilities

There are a number of Commodore 64 utility programs designed to help you with certain aspects of program writing. For example, there are special screen editors like Screen-Graphics-64 from Abacus Software that let you use the C128's screen as a visual scratchpad to create your program's text and graphic displays. There are font editors, like Ultra Font+ from COMPUTE! Publications, Inc., that allow you to redefine some or all of the standard character and graphics sets. There are sprite editors that you can use to create, test (animate), and then store your sprite definitions in a

form usable by your programs. There are even machine language monitors, such as the one available in the C128 mode, that you can use to write your own smaller machine language programs or create short and fast routines usable from a BASIC program. Micromon and Supermon are two excellent machine language monitors that are in the public domain so they are free (a version of Micromon was published in COMPUTE!'s *First Book of the Commodore 64*).

BASIC Enhancement Programs

BASIC enhancers make up another category of programming oriented programs for the Commodore 64. The purpose of many of these programs is to enhance BASIC 2.0 by increasing its overall power and flexibility. This is done by adding new features such as sprite, graphics, and sound commands and improving the operation of older commands such as those of BASIC 2.0's complicated DOS. You have a number of different BASIC enhancement packages that you can use with the C64. Skyles Electric Works has a cartridge called VIC Tree that adds over 40 new commands to BASIC 2.0 –including improved DOS commands and programming assistance commands such as AUTO, RENUMBER and TRACE. Simon's BASIC from Commodore, with all its graphics, sound, and structured programming commands, offers even greater enhancements to BASIC 2.0's operations. If you are just interested in adding graphics and sound commands similar to those found in BASIC 7.0 then you can get Commodore's Super Expander cartridge.

BASIC compilers are another type of enhancement program. Compilers, like BLITZ! from Skyles Electric Works or BASIC Compiler-64 from Abacus Software, are used to perform a one-time translation of a BASIC program into a much faster program format. Compiled BASIC programs, although physically large in comparison, often run as fast as similar programs written in Pascal, Pilot or Logo.

Programming Aids

The final category of programming-oriented Commodore 64 software includes those programs especially designed to ease some of the burden normally associated with BASIC and 6502 assembly language programming. These programs don't actually enhance or change the finished programs in any way, but they do make them easier to write.

Two very popular C64 programs, COMPUTE!'s Automatic Proofreader for BASIC programs and COMPUTE!'s MLX for machine language programs, were specifically designed to help C64 owners enter programs found in COMPUTE! Publication's magazines and books. With these two

programming aids it is possible for C64 users to obtain hundreds of guaranteed-to-work BASIC and machine language programs, even if they know nothing about the languages themselves. Programming aids similar to these are also available from other Commodore marketed magazine publishers.

Another programming aid, also available from COMPUTE! Publications, is MetaBASIC. This program performs functions similar to many found in full BASIC enhancement packages, but unlike those programs, MetaBASIC does not add any operations to the programmable portion of BASIC. Therefore, programs written using MetaBASIC run without it. What MetaBASIC does is add important *immediate mode* commands like AUTO, RENUM, TRACE, DUMP, CAT, ERR, DELETE, MERGE, and LLIST — commands that really simplify BASIC 2.0 programming.

What Can't You Do While in the C64 Mode?

Chapter 3 described many Commodore 128 features, primarily those classified as enhancements to the standard features found in the Commodore 64. When the C128 is used in its C64 mode those enhancements are gone. To begin with, only 64K of the computer's RAM can be used, because that's all the RAM the C64 knows how to use (the remaining 64K of RAM is disabled). Any externally added RAM — for example, a RAM disk — would be equally unknown and therefore unusable. ROM memory areas are also swapped around when the computer enters the C64 mode. When this happens the C128 operating system, with its built-in machine language monitor and improved screen editor, is replaced by the operating system used in the C64 (no monitor and a less flexible screen editor). Switching modes also changes other less obvious memory and device configurations which, among other things, alters the C128's I/O capability. For example, the C128 can no longer access the 14-key numeric keypad, or any of the other C128-only keys. The changed I/O also eliminates the C128's 80-column screen mode option. Lastly, subtle but important changes in both the C128's I/O circuits and internal operating system prevent the C128 from utilizing the advanced storage and speed capabilities of the newer 1571 and 1572 disk drives.

How Does BASIC Differ in the C64 Mode?

One of the consequences of commanding the Commodore 128 into its C64 mode is that BASIC 7.0 is replaced with BASIC 2.0, which is the same

exact BASIC furnished with the Commodore 64. Therefore the C128 can no longer understand any of the new BASIC commands, statements, and functions that are found only in BASIC 7.0. What you have left are those commands and functions listed in Table 4-1.

BASIC 2.0 and BASIC 7.0 Compatibility

As you probably noted from Table 4-1, BASIC 2.0 does not have specific graphics, sprite, and sound operations, nor does it have any statements like PRINT USING, or AUTO, or WHILE...DO. In fact there are a large number of operations available in BASIC 7.0 that you no longer have access to when the C128 is in its C64 mode. If you are thinking about writing BASIC programs in this mode, or just modifying an existing C64 BASIC program, then you should familiarize yourself with the differences between these two BASICs. Programs can be written in such a way as to make them compatible with both modes and both BASICs, but you will need to know both BASICs. Fortunately, as you may recall, BASIC 2.0 (C64 mode) is a subset of BASIC 7.0 (C128 mode). We did a complete comparison between these two BASICs in Chapter 3.

About C64 Mode DOS

The last noticeable change in the way the Commodore 128 operates in its normal C128 mode and the way it operates in its C64 mode is the DOS

Table 4-1. BASIC 2.0 Command and Function Summary

System	*Assignments*	*String Functions*	*Numeric Functions*	*Other Functions*	*Input/ Output*	*Program Flow*
NEW	LET	ASC	ABS	FRE	GET	GOTO
CLR	DIM	CHR$	ATN	POS	GET#	ON...GOTO
CMD	DEF FN	LEFT$	COS	SPC	INPUT	GOSUB
LOAD	DATA	LEN	EXP	TAB	INPUT#	ON...GOSUB
SAVE	READ	MID$	FNxx	DEF FN	PRINT	RETURN
VERIFY	RESTORE	RIGHT$	INT	PEEK	PRINT#	FOR...NEXT...STEP
OPEN	POKE	STR$	LOG		WAIT	IF...THEN
CLOSE		VAL	PEEK			SYS
RUN			RND			USR
STOP			SGN			
CONT			SIN			
END			SQR			
LIST			TAN			
REM						

command structure. First of all the new BASIC 7.0 DOS commands are no longer usable. The same disk operations can be performed, but to do so you have to familiarize yourself with the DOS command structure used in the Commodore 64.

Different Commands

Single word commands like HEADER, CATALOG, RENAME, and SCRATCH are now replaced with short direct mode BASIC 2.0 statements. For example, in order to format a new disk you first open a *command file channel* to the disk drive with a command such as this:

```
OPEN1,8,15
```

Then you use a second BASIC 2.0 statement to actually send the DOS command down that channel:

```
PRINT#1,"NEW:<disk name>,<id>"
```

At this point you could send more DOS commands without having to reopen the disk command channel. A list of the different 2.0 DOS commands with their meanings is shown in Table 4-2. After you have sent your last DOS command you should close the command channel like this:

```
CLOSE1
```

Similarly formatted statements can be used to send most DOS commands. However, there are two other BASIC 2.0 DOS-related operations

Table 4-2. Examples of DOS Commands under BASIC 2.0

BASIC 2.0 Command String	*DOS Meaning*
"NEW:<disk name>,<id>"	Formats a new diskette
"SCRATCH:<filename>"	Erases a file
"RENAME:<newname>=<oldname>"	Renames a file
"COPY:<destination>=<source>"	Copies a file
"VALIDATE"	Collects all unused disk space
"INITIALIZE"	Initializes a new disk allocation table (BAM)

that are even more cryptic in their nature: reading a disk directory and reading the disk drive error channel.

In BASIC 2.0, a disk's directory must be loaded into memory as if it were a BASIC program file. Once loaded, it must then be LISTed from BASIC in order for its contents to be viewed. Here is the command syntax used to read an entire disk directory:

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

Since BASIC is treating the disk directory as a program file, the directory gets placed right into BASIC memory and any program already there is replaced. Thus you can't check a disk's directory just before saving a program to see if the filename you want to use is already there. This is often inconvenient and sometimes disastrous. Just viewing a directory can be equally troublesome, especially if it is rather long, as there is no way to easily display just one screenful of directory information at a time.

Should something go wrong and the disk drive is not able to fulfill a DOS request, read the disk drive's error channel to find out the exact nature of the mistake. For example, you may have accidentally put an unformatted diskette in the drive or you may have tried to save a file using an existing file name. These are typical "operator errors." Of course, the error channel will also report more serious problems like a read error at a particular track and sector location due to a faulty diskette. Unfortunately, the Commodore 128 does not automatically display disk error messages or even check to see if an error occurred. It must instead be told

Table 4-3. Comparisons Between Different C128 DOS Operations

DOS Wedge	*BASIC 2.0*	*BASIC 7.0*
@$	LOAD"$":LIST	DIRECTORY
/<filename>	LOAD"<filename>",8	DLOAD"<filename>"
%<filename>	LOAD"<filename>",8,1	BLOAD"<filename>"
←<filename>	SAVE"<filename>",8	DSAVE"<filename>"
@:<diskname>,id	PRINT#1,"N:<diskname,id>"	HEADER"<diskname>,id"
@S:<filename>	PRINT#1,"S:<filename>"	SCRATCH"<filename>"
@V	PRINT#1,"V"	COLLECT
@	100 OPEN1,8,15	?DS$
	110 INPUT#1,ER,ER$,T,S	
	120 PRINT ER;ER$;T;S	
	130 CLOSE1	

to report the disk drive status by having it check the error channel. This manual error checking applies to both C128 and C64 modes, but the methods used to get the disk status are completely different. In BASIC 7.0, all you have to do to check for a disk error condition is type:

```
?DS$
```

This operation is far more difficult to perform using BASIC 2.0. Here is what you would have to do:

```
100 OPEN1,8,15
110 INPUT#1,ER,ER$,T,S
120 PRINT ER;" ";ER$;" ";T;" ";S
130 CLOSE1
RUN
```

As you can see, you have to write a small program just to get the disk drive's error status.

Improving 2.0 DOS

The purpose of these last examples was to give you an appreciation that sometimes BASIC 2.0 DOS operations can be difficult. Don't be dismayed, however. If you must do DOS operations while in the C64 mode you won't be able to use BASIC 7.0, but you will be able to use the *C64 DOS Wedge* just like two million other Commodore 64 users.

The C64 DOS Wedge is a short utility program that Commodore developed to take some of the pain out of BASIC 2.0 DOS operations. Once this program is loaded into a normally unused part of the C64's memory, many BASIC 2.0/Wedge DOS operations are easier to remember and require less typing than their equivalent BASIC 7.0 statements. The C64 DOS Wedge is supplied free of charge with every 1541 disk drive. A number of different books and magazines have detailed the wedge's operations, including COMPUTE!'s *First Book of the Commodore 64*. Table 4-3 will give you a quick look at some Wedge comparisons.

Complete details on DOS operations in the C64 mode can be found in the documentation provided with your disk drive and in *The Commodore 128 System Guide*.

5

The CP/M Mode

In this chapter you'll learn:

- **What CP/M is and why you might want to use it**
- **What equipment you need to run CP/M on the Commodore 128**
- **What software exists in the CP/M universe**
- **Where to find CP/M software for your Commodore 128**
- **How CP/M is organized**
- **What CP/M commands there are and what they do**
- **Where to go for more information about CP/M**

If you have never heard or seen the word CP/M before then this chapter is for you. This chapter will also be useful if you're familiar with CP/M on another computer and want to know how Commodore's CP/M is different. CP/M stands for Control Program for Microcomputers. That doesn't tell you much but it does give a hint of what CP/M is: a way to control programs. CP/M is a special kind of *disk operating system*. A disk operating system is a sort of supervisory program that is used to do chores, such as run programs, make copies of programs, erase files, measure the capacity of a disk, and so on. In this chapter we will tell you what CP/M is, its value and features, what you need to use it, what important and powerful CP/M software exists and where to find it, and finally a bit about its commands and what they do. We will also tell you where to go for more detailed information on CP/M.

What Is CP/M and What Is the CP/M Mode?

If you are familiar with the Commodore 64 and the 1541 disk drive you know that you can use commands in BASIC to access programs stored on a diskette plugged into that drive. And if you read the chapter in this book on the new C128 mode you know that Commodore has provided an even more powerful set of BASIC statements and commands for accessing the disk (borrowed from the Commodore CBM line of small business computers). From within BASIC you use these commands to find out what files and programs are on the diskette in the drive, to erase programs or to duplicate them on other diskettes, and even to run programs written in languages besides BASIC. Unfortunately, these disk management features exist inside the shell of another language, and have more complex syntax then we would desire. What they do is insulate us from having complete control over files and programs on the disk.

What is a Disk Operating System?

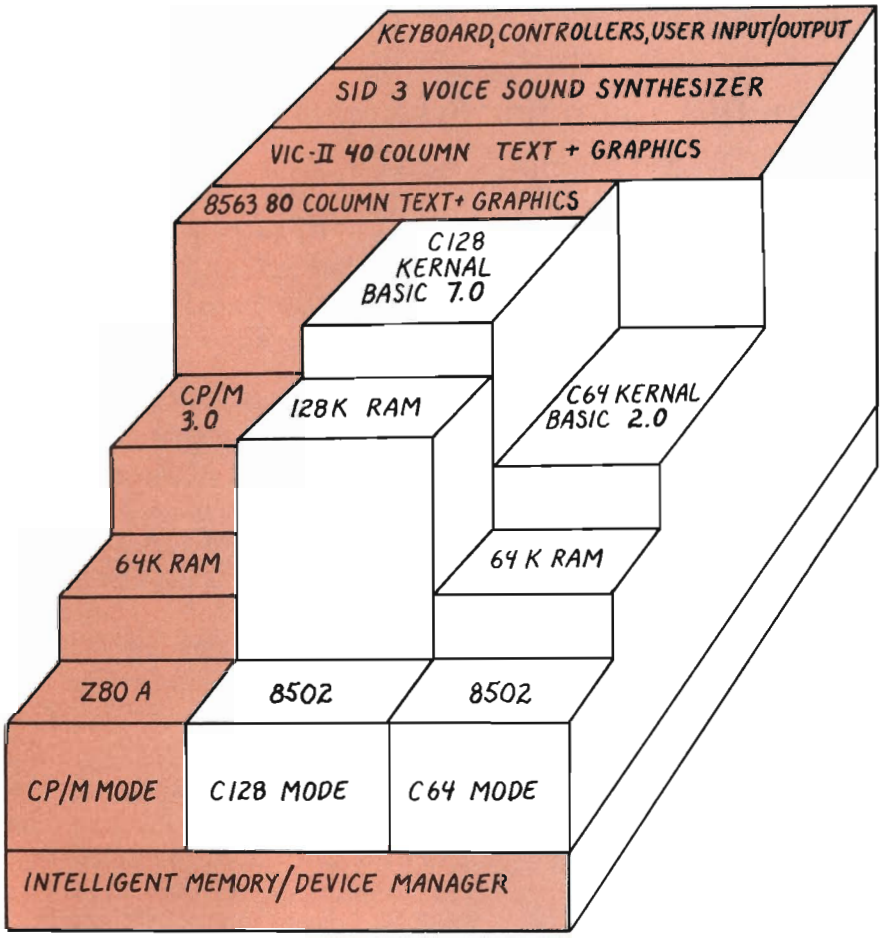
CP/M is a full-fledged operating system that gives complete control over the disk. One purpose of a disk operating system is to *simplify the handling of programs and files*. This is done in CP/M by providing a set of commands that are easy to use and remember. Since CP/M is not inside or part of another language, such as the disk commands of Commodore 64 or BASIC 7.0, its use is more straightforward. A disk operating system like CP/M is actually a kind of "mother" or supervisor program. It is the program you use to control other programs — a sort of master that conducts the operations of any "application" program on the disk.

So why another DOS? Commodore chose CP/M not just because it offers more or better DOS commands, but because it offers a huge body of software that can now be run on the Commodore 128. CP/M has a unique position in the world of microcomputers. It was the first operating system developed for microcomputers (CP/M came out around 1975 — three years after the introduction of the first microcomputer: the Altair 8080). It was first developed so that many different computers could run the same program. A program that was written to run under the CP/M operating system on one computer would work on another computer if it also ran CP/M. Over the course of many years, many manufacturers produced machines that ran CP/M, a huge body of software was developed for these computers, and a new industry was born. Today there are dozens of companies that manufacture CP/M computers and a fantastic selection of software you can choose from.

The CP/M Mode: A Third Personality

Unlike the Commodore 128 DOS, which resides in ROM and can run immediately after the computer is turned on, CP/M resides on a diskette as a program, and this program must first be loaded into the computer memory before it can do anything. Figure 5-1 shows how this is done. This distinction is important because you must learn to "boot" CP/M before you can start using it. In addition, the Commodore 128 and Commodore 64 modes use the new 8502 microprocessor. CP/M, on the other hand, requires a special Z80A microprocessor to run. Commodore has

Figure 5-1. How the CP/M Mode Stacks Up



built this microprocessor into the Commodore 128. Thus the Z80 and CP/M turn the C128 into a completely different personality, which Commodore calls the CP/M Mode. Figure 5-1 also shows the various chips that the CP/M mode can access.

A Little History of CP/M

CP/M was developed by a (now very rich) man named Gary Kildall, who was working, at the time, for Intel, a large manufacturer of microprocessor chips. Gary developed a package of compactly written subroutines for the tiny (and now ancient) four-bit Intel 4004 microprocessor. These subprograms could be used by other programs, simplifying the work for other Intel programmers. As technology advanced, more powerful chips were developed, including the Intel 8008 and 8080 microprocessors, and the 8080-compatible Zilog Z80 microprocessor. Kildall went on to develop more routines for the 8080 and the compatible Z80. His overall goal was to create an environment that would allow storing and running programs on a disk. Because no one believed that microcomputers would ever become very popular or that they would ever use an expensive contraption like a disk drive, Intel gave Kildall the okay to market CP/M on his own. He started up a company called Digital Research, which to this day is still going strong.

> ***CP/M was the first disk operating system for microcomputers and now has over 1.5 million users.***

Keep in mind that up to this point there was no operating system for microcomputers. So CP/M was quickly seized upon by most users and manufacturers of 8080 or Z80 computers. Since there were no other contenders, CP/M quickly became a standard in the industry. Since everyone's computer had an 8080 or compatible Z80 chip, CP/M made it possible for one program to run on many different computers.

Most CP/M-based systems had a minimum of a keyboard and monitor or terminal, one or two disk drives, and 48K or 64K of memory. These computers were not designed to be compatible with each other: they each had differently sized screens, different disk capacities, and different keyboard layouts. But CP/M, the great equalizer, made them all act alike.

What Does CP/M Offer?

Today there are many microcomputers available that run CP/M. Most of these computers are moderately expensive (in the $1,000-$2,000 range),

and generally, most have been purchased by small- and medium-sized businesses. However, because CP/M microcomputers offer such great value, a large number of CP/M computers have been sold, and a huge body of software has grown up to support these machines. Until the Commodore 128 appeared, there had never been a low-cost CP/M computer for the home.

CP/M's Incredible Base of Software Programs

CP/M offers a huge world of software — a world where literally thousands of programs exist for your C128. Some of these programs are extremely good. Many are free if you know how to get them. To give you an idea of how much software exists, one software directory lists over 1,000 commercial programs that run under CP/M, and experts estimate that somewhere around 10,000 relevant CP/M programs exist. These programs would consume over 3,000 C128 CP/M diskettes.

The software that exists for CP/M is oriented for professionals and businesses. You will find powerful word processing programs, spreadsheets, financial packages, database managers, languages of all kinds, utilities, and much more. In fact, the Commodore 128 running CP/M represents a bargain for small businesses. The price of the Commodore 128 with the 1571 disk drive is competitive in performance with an IBM PC and is much less expensive. The software is time-tested and bug-free. So much software is available that you will easily find special interest software — programs that would not have mass appeal but are just what you are looking for. For example, you might find an accounting program written especially for a law practice, a database program for a stamp collector, or an engineering spreadsheet for a metallurgist.

There is also a huge amount of what is called free "public domain software" available for CP/M. We will detail both commercial and public domain software after we tell you what equipment you need to run CP/M.

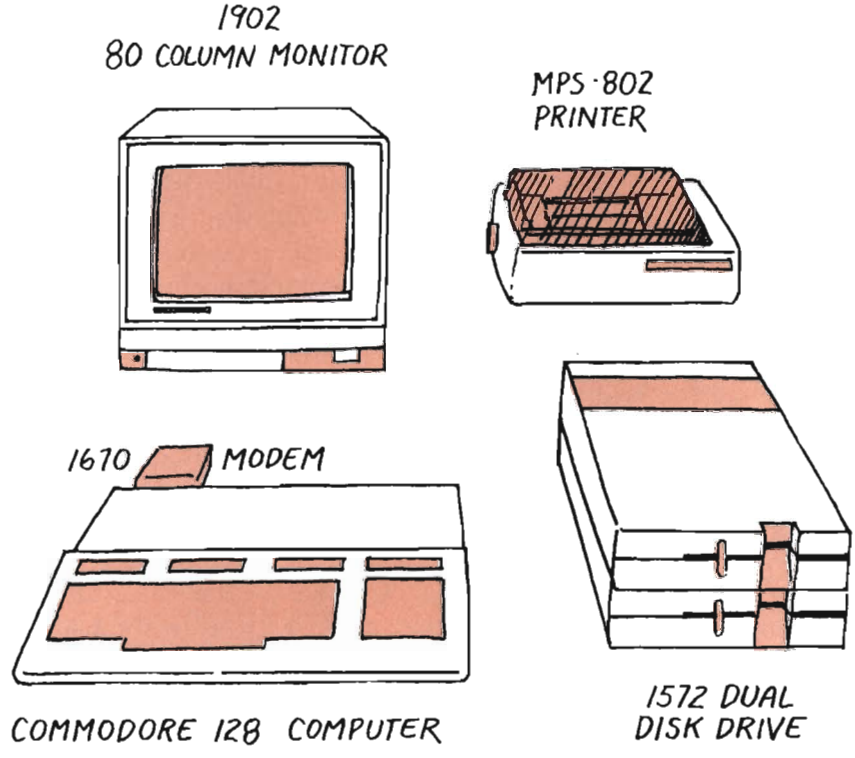
What Equipment Do You Need to Run CP/M?

In order to use CP/M on the Commodore 128, you will need either a 1571 single disk drive or a 1572 dual disk drive, a 40- or 80-column monitor, and, of course, a set of CP/M "system" diskettes. You may also want to purchase an MPS-802 or DPS-1101 printer and a 1660 or 1670 modem. See Figure 5-2.

Disk Drives

The Commodore 64 uses the Commodore 1541 disk drives. These drives were slow because they used a serial method to transfer data from the disk to the computer, and because of certain design elements in the way the disk drive communicated with the computer. The new 1571 and 1572 drives are much faster then the 1541 drive. The old drives transfer data at 320 characters per second. The new drives in the CP/M mode can transfer at a rate of 3500 characters per second, which is more than ten times faster. These new drives contain an intelligent microprocessor, which gives the Commodore 128 an advantage over all other CP/M computers. In the early days of CP/M, all software was distributed on 8-inch diskettes and only one storage format was used. When 5¼ inch disk drives became popular, manufacturers used different format techniques. Dozens of storage formats now abound, making compatibility a problem. It's not that a program cannot run under another machine's version of CP/M; rather, the

Figure 5-2. Recommended CP/M System



program simply cannot be entered into the computer because the disk drive doesn't understand the way it is stored. The Commodore 1571 disk drive is smart enough to automatically adjust to the numerous storage formats found in the CP/M world. This means that, regardless of what computer your CP/M program runs on, the 1571 will be able to read it.

The choice of single versus dual disk drive is strictly one of convenience and cost. A single drive will allow you to run CP/M programs right away. However, making backup copies of programs and files with a single drive is tedious if it must be done often. The dual drive 1572 allows you to easily copy files from one diskette to another.

Display

There are two ways to display text for CP/M. The best and most expensive way is to purchase a new Commodore 1902 80-column color monitor. This new monitor produces beautiful color and will work with the Commodore 64, Commodore 16, Plus/4, and Commodore 128. The display has a resolution of 640 by 200 pixels and can display an array of 16 colors. A new video chip in the Commodore 128, which is much like the one used in the IBM PC, drives this display in the 80-column mode. It is apparent that Commodore expects that programs now running on the IBM PC in the color mode will easily be reverse engineered to work in the Commodore 128 CP/M mode.

A neat feature of the Commodore 128 80-column mode is that the characters that are displayed are stored in RAM and are "bit-mapped". When the computer is first turned on, a set of character fonts stored in the ROM is moved into RAM. When CP/M wants to display a character, it gets it from this memory area. Since the characters are stored in RAM, it is possible to modify them and to create custom characters for your programs. It also means you can mix graphics and text on the same screen with CP/M.

You can also use a high-resolution 80-column *monochrome monitor*. A monochrome monitor is one that displays in only one color. On the color monitor the text characters are stretched a bit too much in the vertical direction, but are still quite readable. The Commodore color monitor includes built-in speaker, volume, color, tint, brightness, and contrast controls.

The second way to display text in CP/M is in the 40-column mode. This mode uses the standard VIC display chip found in the 64 and can be used with a standard 40-column Commodore 1702 color monitor. The 1702 monitor is less expensive than the new 1902, but it cannot display the entire 80-column line that CP/M programs expect. Instead, the first

40 columns are displayed normally. A special key on the Commodore 128 keyboard shifts the remaining rightmost 40 columns into view. Reading CP/M programs this way requires constant toggling between screen views, and therefore is only recommended if you will be creating CP/M programs that need 40 columns. Another justification might be writing CP/M games that use the Commodore 128 color and sound. A subtle advantage of using the 40-column mode is that the clock speed of the system is doubled from 1 to 2 Megahertz.

CP/M Diskettes

You will need a set of Commodore 128 CP/M diskettes. Presently CP/M is distributed on two disks. One contains special CP/M startup programs and the other contains various utility programs. We will explain this in more detail later.

Important Programs on a CP/M Owner's List

In 1976 there were 100 programs that ran under the CP/M operating system. In 1983 there were over 10,000 programs. With so much CP/M software to choose from, users of the Commodore 128 running CP/M are faced with a real problem of overwhelm. In this section we will help you presift and select the best software from the wide and abundant variety of CP/M programs. Not all software is created equal. We will focus on the programs that we feel are the best on the market for the price. Since there are so many good programs, our list is by no means to be considered exhaustive. By reading this section you will become aware of some of the most popular programs, and at the same time learn just what is possible in CP/M software.

The software that is popular for CP/M includes word processing programs for typing everything — from short letters to long chapters in a book. Most experts agree that really powerful word processing programs may have been responsible for starting the CP/M software marketplace. Eventually everyone needs to write something on the computer. Word processing *tools* exist that allow you to automatically check the spelling of your letters or documents, build a table of contents, create an index, or insert names at specific spots in letters.

Spreadsheets are popular programs used in businesses. These general purpose programs allow you to manipulate numbers and figures in a large matrix of rows and columns. Some of the best spreadsheets ever created

were developed first for CP/M. Database management programs are popular for CP/M and allow you to manipulate large amounts of information that can be specified as a collection of "data fields", such as names, addresses, part numbers, and the like. Databases are used for organizing a company's customers, identifying amounts due, setting up inventories, analyzing data, printing reports, and so on.

Very good financial software has been created for CP/M. There are accounts payable and receivable programs, journals, checkbook balancing programs, balance sheet and income statement generators, and even income tax programs. Thousands of companies run their accounting systems under CP/M. You will also discover a fine assortment of communications software that runs under CP/M. These programs allow you to send files and programs over telephone lines, use remote database services like The Source and CompuServe, and even to get Dow Jones stock reports. In fact, CP/M includes a famous telecommunication program called Modem7 that is free. Languages for programming under CP/M exist in great numbers. Microsoft BASIC (called MBASIC) is the most popular CP/M language, but you will also find Pascal, the new C language, Forth, COBOL, Lisp, Prolog, and others.

Don't look for much in games for CP/M. There are adventure games that are good, but Kildall designed CP/M to work with "generic" terminals and keyboards. Therefore, standard CP/M does not support video graphics, sound, or function keys and most CP/M programs you find will just use the 80-column text mode. However, as we shall see, it is possible for programs to be written in Commodore 128 CP/M that access the C128's graphics, windows, and sound effects, and we can expect to see these developed over the next few years.

A Caveat: Disk Format Compatibility

As we mentioned previously, a plethora of time-proven software exists for CP/M that is directly usable on your Commodore 128. The only requirement is that the CP/M software be available on 5¼ inch floppies. One particularly thorny problem with distribution of CP/M software is the disk format. When CP/M was originally created it used 8-inch single density, single-sided diskettes and a single storage format called IBM System 34. There now exist several new floppy storage possibilities, including double-density, and double-sided. When the 5¼ inch disks arrived, several manufacturers of CP/M computers, including KayPro, Morrow, and Osborne — makers of CP/M portables — slightly altered the storage format to suit their particular hardware. As a consequence there is incompatibility between the drives of different CP/M machines.

Commodore gets around this incompatibility problem by using an intelligent peripheral drive. The 1571 and 1572 drives are capable of reading any particular CP/M-formatted diskette. When you insert a diskette the 1571 disk drive tries to determine the type of storage format. Once the Commodore 128 has figured out how the data is stored, it is an easy matter to read it. Commodore 128 CP/M is also capable of formatting a CP/M disk in any of several formats, so you can write a program that can be read by another popular CP/M computer.

Now that you know a little about storage formats, let's see what particular software you might want to own.

Word Processing Programs

Word processing programs are among the first software purchased by many users. A word processor is a text editing program that lets you do what a typewriter does and a lot more. Besides allowing you to instantly insert a letter, word, or sentence anywhere in the document you are typing, a word processing program can automatically reformat an entire document in a few seconds, straightening out margins, changing width of lines, and so on.

WordStar

WordStar was the first "what-you-see-is-what-you-get" editing program for CP/M. What this means is that the formatting features, such as double spacing, justification, and margin settings are visible on the screen before the text is printed. The way the text looks on the screen is the way it will look when it is finally printed. Older text editors were broken into two parts: an editor for entering text and a formatter/printer for printing text. You could not see what the printed copy would look like until it was printed. The ED.COM program that comes with your copy of Commodore 128 CP/M is an ancient relative of WordStar.

WordStar, like many CP/M programs we will describe, must be "installed" on the particular CP/M computer system it will be working with. This installation process is necessary because each CP/M system may have a different size screen or type of terminal, different printer, and even different kinds of keyboards. Different terminals have different protocols for how they put things on the screen. The manufacturers of CP/M application programs, such as WordStar, do not want to inventory dozens of different disks for each machine, so they create an installation procedure into their software. When you install a CP/M program, you are basically telling it the characteristics of your screen and printer. The program then

uses a built-in table to produce a custom version of the product. WordStar has a powerful "install" program that provides a very comprehensive menu of printers and terminals it can work with.

Although WordStar is one of the oldest, most mature word processing programs on the CP/M market, it has a complex set of commands that you must memorize to work with it effectively. A well-designed set of on-screen "help" menus makes this fairly easy. WordStar's documentation is massive, but many books are out that simplify learning about it. A "mail-merge" option available for WordStar allows you to print copies of the same letter to a list of people stored in a database file. You can have specific information entered into each letter, too.

Perfect Writer

Perfect Writer is part of a family of Perfect Software, including Perfect Calc and Perfect File. Like WordStar, Perfect Writer uses on-screen formatting, but in addition it allows split-screen editing. This means you can split the screen into two horizontal windows, and view different parts of the same document, so you can read one section while modifying another. In fact, Perfect Writer allows you to open seven files at the same time and edit them simultaneously. Another reason that split windows are important is that you can cut and paste between two documents while both are on the screen. In WordStar, you have to save the text to be pasted in a file, quit the current document, reload the new document, and read in the file of text.

Embedded commands that set spacing and set up headers and footers are placed in the text enclosed by special symbols. These commands affect the formatting of the entire file and take some time to get used to. But they are more powerful than WordStar's formatting, where every paragraph must be individually reformatted. Perfect Writer is a bit more user friendly than WordStar because it focuses on two keys (Escape and Control) for most of its commands.

Perfect Writer has an excellent install program and can even allow customization for hardware not on its menu. A version of Perfect Writer for the Commodore 128 is available.

Database Managers

Database managers are programs used to manipulate data that exist in the form of "records". Records can be thought of as a single line of information that contain, for example, names, addresses, amounts invoiced, number of parts on hand, and so on. A database manager is a

program that lets you enter the data, arrange how it will be stored, search for certain data, and print out summary reports. Businesses are frequent users of database managers, since they frequently deal with customer information that can easily be represented in a database.

dBASE II

Probably the most popular database program for CP/M is dBASE II from Ashton-Tate. It is known as a relational database, meaning that any one piece of information can be linked to another. For example, a name can lead to an address and a phone number, and a phone number can lead to an address and a name. An actual dBASE II application program requires technical skill to program, but no technical skill is needed to use the final program. dBASE II has a programming language that you use to create a database. Although difficult to learn at first, once mastered you can create very complex data management applications. Many good books are available to teach you dBASE II programming.

Spreadsheets

A spreadsheet is simply an electronic ledger for "what-if" analysis. It gives you a way to test different possibilities by manipulating numerical data entered in column and row format. Not only can you enter numbers, but also formulas that operate on the data in rows and columns. Typical "what-if" questions that a spreadsheet answers are: How should you invest if interest rates are 15 percent? What is the effect of producing more purple socks than green socks? Financial modeling and forecasting are one of the main uses of spreadsheets. The important characteristics of a spreadsheet are its worksheet size, report capability, speed of calculation, interaction with other programs, and ease of entering data.

MultiPlan

MultiPlan was one of the first spreadsheets for CP/M. It is organized around the worksheet, which shows on your screen organized in rows and columns, much like an accountant's ledger. MultiPlan actually uses the screen as a window to a much larger worksheet — it shows the area of 7 rows and 19 columns — out of a total area of 63 columns by 255 rows. You use commands to move around on the worksheet and enter values. You can also enter formulas that automatically perform calculations on the data in the columns and rows. Recalculation is almost instantaneous. In a way MultiPlan works a lot like a word processor, except it uses the cursor keys for moving through the rows and columns. You can delete

large areas of the worksheet with a few keystrokes. You can enter text as well as numbers to nicely format the way the worksheet appears, giving headings to the figures, for example. There is an online help system that allows you to leap headlong into the program. However, the large manual is excellent for reference. Several good books exist for learning MultiPlan. MultiPlan's only lack is that there is no simple way to get graphics output from the program. With the Commodore 128 and its excellent graphics screen we may find that a version of MultiPlan with graphics output becomes available.

SuperCalc2

Another excellent spreadsheet program is SuperCalc2. This spreadsheet is much like MultiPlan, except its worksheet holds slightly more rows and columns. SuperCalc has an interesting batch execution system that lets you put a series of SuperCalc commands in a file and execute it. For example, if you want to print a report that uses different sections of the worksheet, the commands in the file can specify these ranges, one after another, unattended. This is especially useful to experts who wish to develop complex packages that can be used by novices. SuperCalc also uses a kind of storage for spreadsheets called SDI for *Super Data Exchange*. It can store in a comma-delimited format which is suitable for other CP/M languages like Microsoft BASIC, CBASIC, and Pascal (see the section on CP/M Programming Languages). SuperCalc2 is easy to use and has a help key feature.

Financial Software

Financial software programs are CP/M packages that allow a business to do all its bookkeeping on the computer. Most financial packages offer a general ledger program and a payroll system. A complete system would contain programs for handling receivables, payables, inventory, and job costing. Financial software is a difficult product to define, and there are many packages that fall into this catagory that may be quite different. When purchasing such a system for your Commodore 128 in the CP/M mode, you will want to pay attention to how flexible the structure of the product is (what functions have been preallocated in the program) and how flexible the reporting is. You'll also want to check the integration of the package; that is, can the different parts of the system, like the checkbook and the income statement generator, talk to each other? You may also want to be concerned with security and make sure there is some kind of password protection, so that only certain employees can write checks.

Software Fitness Program

Here is a solid, screen-oriented, seven-part accounting system. Software Fitness Program from Open Systems offers: general ledger, accounts receivable, accounts payable, sales order, inventory, payroll, and job cost programs. The entry screens are amazingly comprehensive. WordStar-like editing commands let you move easily around Software Fitness screens and enter and modify data in them. The Software Fitness Program can adapt to your business' accounting practices, rather then forcing you to adapt to its system. Everything about the program is stored in tables that can be modified. There are many types of reporting functions, including audit trails, comparisons and analyses. Custom reports with custom formatting is possible. The system is full of features. The accounts receivable program handles finance charges on forward balances and sends statements and invoices. The payroll handles sick pay, vacation, FICA, state and local taxes, and even double time.

The program runs under a special BASIC Interpreter from a company called Control C and that program must be purchased separately. The Software Fitness System earns its name: it is comprehensive, easy to learn, and the documentation is good. One possible special feature for Commodore 128 users is that the Software Fitness Program is capable of dealing with up to twenty-four concurrent users working on the same files. If a company creates a network scheme for C128s running under CP/M and installs the Software Fitness Program, it could be the most cost-effective corporate computer financial system ever sold.

Communications

The latest craze is computer networking and system-to-system communications, and the Commodore 128 user will most likely want to try it out. Networking and communicating with computers involves being able to access remote computers, often over the telephone lines. This may be for sending electronic MCI mail or getting a file or program from someone's computer. Electronic mail is a method whereby you can write a letter and transmit it to a computer owned by the electronic mail company. The file is converted into a printout and delivered to the recipient overnight. Sending and receiving programs is another popular reason for allowing computers to communicate. You may also use computer communications to call up far-away computers and then search and roam through the fantastic information databases they contain. Or maybe you want to access tons of free CP/M software (we're getting to that soon).

To get started in communications you need a modem and a communications program. The modem encodes and decodes information from the computer into a standard format of tones. If the remote computer has a modem, the two computers can understand each other and trade information. A communications program allows you to easily send and receive files. For example, if you use electronic mail you will need to be able to send memos, letters, and information. Your software should be able to look on your disk for a file containing a letter, and then be able to send it to the mail network. When receiving information from another computer, it needs to be able to open a file on your disk and capture the information. Your communications software also needs to be able to emulate a popular terminal of some kind via your communications program. This is because most information services set up a communications link that expects a certain kind of terminal to be attached.

Your program should also allow handling *protocol* file transfer. Protocol means that special error checking is set up between sender and receiver so that no errors occur in the transmission. The reason protocol is needed is that phone lines can have electrical static, or noise, on them which can alter the characters they are sending or receiving. The protocol mode guarantees that the character is sent properly. The most popular protocol is the XMODEM protocol, which was developed by Ward Christensen and placed in the public domain. XMODEM is part of another modem program, called Modem7, developed by Ward, and is described later in this chapter in the section called "Free Software". The mode for most communications between computers starts out with no protocol. The character that is sent is always echoed so you can see it appear on your screen (this is called *full duplex* mode), but no check is made to see if the right character is echoed. This is called the non-protocol mode. Which mode you use depends on the kind of file you are sending. A program file, like a COM file, is composed mostly of pure machine codes for the Z80A microprocessor. Sending this file demands protocol mode for it to work. On the other hand, you might not care about accuracy of a memo you are sending to a person via electronic mail.

Crosstalk

Although Crosstalk, manufactured by MicroStuff, wasn't the first CP/M communications program on the market, it has risen to the ranks of the most popular. Crosstalk is designed mainly to work with the Hayes Smartmodem, because this particular modem has built-in intelligence that Crosstalk can exploit. It can perform autodialing, both in pulse and touchtone, and has numerous selectable protocol features. Crosstalk cap-

tures incoming data in memory, lets you go in and edit it, and then save it to disk when ready. Like all good communications programs, Crosstalk lets you adjust important parameters, like the phone number, baud rate, number of data bits, number of stop bits, parity, and full or half duplex. What is really nice is that you set all of these parameters on a screen control panel which is always in view, so that their state can always be known. Since each computer system you talk to may require different sets of parameters, Crosstalk lets you store the settings under a filename, like SOURCE.XTK. Then you simply type LO SOURCE.XTK and Crosstalk LOads in the parameters you stored.

In addition to having a well-designed terminal-emulation mode, Crosstalk has a file-transfer mode. Even though the computers may be quite different hardware-wise — that is, you might have a Commodore 128 communicating with an Osborne — if they are both running Crosstalk, file transfer will be easy. People who use Crosstalk to trade public-domain "squeezed" (compressed) COM files will appreciate the unattended transfer mode. This allows you to use Crosstalk to call up an unattended computer also running Crosstalk, examine the files on that computer's drives, and transfer just the ones you desire. The transfer can be either an XMODEM protocol transfer, where all the characters are checked as they are sent, or in the non-protocol mode, where there is no accuracy checking. Crosstalk is bulletproof and almost impossible to crash. An online help command gets you descriptions of each command without needing to turn to the manual. Crosstalk is available for many machines, including the IBM PC and the full line of CP/M portables, such as Osborne, Morrow, and Kaypro.

Languages

If you are dissatisfied with a CP/M application, or want to create the next best-selling application program, you must know how to use a computer language. A programming language is what was used to create what ultimately became the final "instructions" to your Z80 chip, which make the program do what it does. For example, CP/M, WordStar, and MultiPlan were written in a very low-level language called assembly language. On the other hand, the Software Fitness Program was written in BASIC, a higher-level language. There are many different types of computer languages available for CP/M. Since the CP/M universe was made up in its first years primarily of hackers (computer hobbyists), it has spawned a great number of experimental computer languages, some of which went on to become great commercial products.

There are dozens of differences between languages, and this book cannot really compare them with any justice. Generally speaking, BASIC is the most popular language today, primarily because it is easy to learn and use, is available in versions for almost all computers, and is interactive in its operation with the user. Pascal and the newer C language are the next most popular. Pascal is a more serious language, and requires more expertise on the part of the user, but also offers in return more flexibility (and sometimes more speed) than BASIC. Pascal has traditionally been used in the schools to teach upper-division programming. C is slowly gaining ground on Pascal because it offers even faster and more compact code, and because the famous Unix operating system is based on it. Many professional application products are based on the C language.

Microsoft BASIC

Microsoft BASIC has a fantastic history in the world of CP/M and the personal computer industry in general. Bill Gates, now C.E.O. of one of the largest software publishers in the industry, and John Allen, his partner, created Microsoft BASIC back in 1975 to run on the world's first real microcomputer, the Altair 8080. At that early date there were no floppy disks (CP/M was about to arrive), and so Gates and Allen got BASIC running using a paper tape reader and loader. When CP/M came out Gates and his partner quickly brought out a disk-based version of MBASIC. The program instantly caught on, and since that time Microsoft BASIC has become the de facto standard programming language for CP/M.

Microsoft BASIC is available as an interpreter and a compiler. The interpreter, the most popular form, accepts BASIC commands as you type them and executes programs immediately. The compiler form reads BASIC instructions you have created in a text file using an editor, and usually takes a few minutes to tell you if there are any mistakes. The compiler, however, produces much faster code. Frequently programmers try to use the more friendly interpreter version to get the BASIC program working, and then use the compiler to speed it up. Using Microsoft BASIC under CP/M is very easy. In BASIC each instruction is on a "line" and the line is preceded by a line number. You can then control the program contents through a line editor; you can delete a line by typing its number with nothing after it, or insert a new instruction line by typing the number and the BASIC commands after it. You can get a listing of the entire program or just specific lines. Microsoft BASIC has its own built-in line editor, but you can also create a program using any of the popular word processors, or even CP/M's crude ED.COM editor. In Microsoft BASIC you are given a complete set of built-in functions, like SIN, LOG, and EXP, as well as

the standard operators. The control structures of Microsoft BASIC are not as complete as Pascal and C, and you are provided with WHILE-WEND, IF-THEN-ELSE, and FOR-NEXT.

With Microsoft BASIC you can chain programs too large to fit in memory, as well as use either random or sequential files. A fully formatted PRINT USING function is provided for output decimal numbers in precise columns. Error checking is good, and you can write your own custom routines to handle user mistakes. Microsoft BASIC provides an interface to memory with the POKE and PEEK commands. A CALL and USR statement allows you to run Z80 machine language programs from Microsoft BASIC. The reference manual for Microsoft BASIC is very good, and dozens of programmimg books are available that teach you how to use it.

Microsoft BASIC for CP/M is not as powerful as BASIC 7.0 on the Commodore 128, but it does offer compatibility with many other machines. For example, Microsoft BASIC and BASICA are distributed with every IBM PC and are almost identical to CP/M MBASIC. Since the 80-column color chip in the Commodore 128 has the same capability as the color board in the IBM PC, it is likely that many programs written on the IBM PC under BASIC that work with color will be easy to rewrite to run on the C128.

Pascal/MT+

Aficionados and devotees of the more sophisticated Pascal language will find Pascal/MT+ from Digital Research, the originators of CP/M, tough to beat. Pascal is a compiled language (although of late interpretive Pascals are occurring), and is best known for its "structuring". Pascal is really more in tune with what professional progammers need than a language like BASIC (although also of late BASIC is beginning to shed its toy-like reputation and more Pascal-like structured versions are appearing).

Pascal/MT+ is one of the fastest and most efficient Pascal languages on the CP/M market. It is fully compatible with another popular Pascal called UCSD Pascal, but it is faster and produces more compact code (less bytes per program). The language allows you to create a large number of program overlays that can be pulled off the disk and affixed in specific memory locations. This allows the Pascal programmer to write much longer programs than would seem possible. Very powerful math processing is provided in this language. Numbers can be as large as 32 bits, and BCD arithmetic for up to 18 digits is provided for computations involving money. (Rounding errors are a real problem with most languages used for business — BCD math avoids them at a slight decrease in speed.)

There are a lot of additional support utilities provided with Pascal/MT+, including a complete debugging system, a disassembler, and a linker with Microsoft assembler compatibility for using Pascal/MT+ with machine language programs. One of the most beneficial utilities provided is the speed programming package. This is a stripped-down WordStar-like editor that has an automatic formatter and syntax checker built in. If you are pretty good at Pascal it will catch almost all your errors. Your file is then given to a special fast compiler that lacks the syntax checking stage. The final version of the Pascal program runs just as fast, but the development of the programming is speeded up by this utility.

BDS-C

C is a programming language whose popularity is on the rise. C was created by Kernighan and Ritchie of Bell Labs to help them write an operating system called Unix. Pascal would not do because it did not produce compact enough code, was too slow, and didn't have a neat interface to the central processing unit (CPU). Although C has not been available on CP/M for very long, its use has blossomed because of its superiority over other languages for developing powerful programs. C is about halfway between Pascal and assembly language in complexity. It is more cryptic looking than Pascal (at least you can make it look that way if you want), and it is, in general, much faster. C was designed because an operating system is a very complex piece of software that must be fast when used. Writing it in machine language may appear the best approach at first, but when the size of the program becomes huge enough, assembly language becomes unwieldy and uncontrollable. C is well-structured like Pascal so it can look clear, while it also allows speed-sensitive parts to be easily interfaced in machine language.

Some excellent applications programs for CP/M were written in BDS-C, including the Mince text editor (from Mark of the Unicorn) and the PeachText word processor (from PeachTree).

BDS-C was the first version of C on the market for CP/M. It has evolved over the years into a very bug-free product preferred by many users. The language is not as complete as the one described by Kernighan and Ritchie's *The C Programming Language* (Prentice-Hall, 1978), but it does contain a large subset of the important features. One of BDS-C's best features is that it consumes very little memory for even the largest of programs. The language supports a feature called pointers, which is very important to C. The floating point math routines are incompatible with those of Unix C. The compilation speed of programs written in BDS-C is amazingly fast compared to languages like Pascal, sometimes taking mere

seconds when in Pascal the same program might take several minutes. The BDS-C compiler is easy to use because it does not need a lot of complex linkages to run. You compile the program and then link it, and it is ready to run.

BDS-C's only weak point is that the relocatable files created by the compiler are not compatible with the Microsoft assembler, so linking to machine language is more difficult.

Utilities

A galaxy of programs falls into the utilities category. Utilities are aids that help you work within the CP/M environment. Utilities are also programs that integrate other applications, give you desk accessory programs that can be called up as needed, like calendars and notepads, or provide tools that make programming CP/M easier. Some of the most popular utility programs provide an interface and set of commands that make CP/M easier to harness.

Power!

Power!, from Computing! in San Francisco, is a program that Digital Research should have provided. It is a set of utilities that take over where PIP, TYPE, ERASE, and RENAME fear to tread. To run Power!, you simply type its name and it installs itself inside your CP/M, then sits there like your slave. The most useful Power! command is COPY. When you do a DIR with Power! it displays a number before every file. If you then want to copy selected files, you can say COPY 1-3 7 13 and only files 1 through 3, 7, and 13 will be copied. ERASE works the same way. There are also utilities for checking if there are any bad sectors on your diskette and unerasing files you accidentally erased. Power! uses a friendly menu-like front end that is good for beginners to learn with. There is also a powerful debugging system for programmers that allows you to read sectors on the disk or memory contents. Several of the ideas for utilities in Power! can be found in various programs in the public domain (see next section).

Entertainment

Entertainment programs in CP/M are somewhat lacking due to the nature of the user interface. This is because to keep a CP/M program truly generic it can't do anything beyond placing text characters in different places on a screen. There is no color or sound in standard CP/M. This has not stopped games from being developed, just held back the sophistication,

compared to color graphics programs. You will find a host of text-based games like Zork and Adventure. There are millions of CP/M games written in MBASIC such as Tic-Tac-Toe, Animals, Guess the Number, and Blackjack, but don't expect much. Remember, many people cut their computer-programming teeth with CP/M, and the games reflect this to a certain extent. The best games are written in machine language or C.

Adventure

Adventure, from Adventure International, is a game where you explore a cave full of strange, magical creatures. You communicate with the game in plain English, typing instructions about what you want to do. The purpose is to find one of several treasures without getting killed in the process. You type things to Adventure like "kill dragon" or "pick up magic lantern". Adventure tells you things that happen along the way, like "you are in a large cavern and the walls are coated with jelly — you can't climb them". (You may find later that the only way out is to eat through the walls!) In Adventure games, you can get an inventory of what things you are carrying, look at your surroundings, pick stuff up, and so on. People become so enraptured by games like Adventure that winning becomes an obsession.

Free Software for CP/M

Earlier we mentioned that there existed free CP/M software — for example, we mentioned XMODEM. What we didn't tell you is that some of this free software is incredibly good, and that obtaining it is not hard. Public domain software is programs that have been written and made available for free to anyone who wants them. They are, in fact, owned by the public.

Since CP/M was one of the first operating systems, a large number of hobbyists, hackers, and esoteric programmers purchased CP/M computers and began to write programs for themselves. Keep in mind that, prior to the microcomputer revolution, programmers and hobbyists were locked out of using computers because only large companies could afford them. With the advent of the low-cost microcomputer and CP/M, literally thousands of hackers said, "Now I can write that program I've dreamed about," and set to work. The result of this was an incredible barrage of homespun, public-donated code, most of it poor and shoddy, but some of it real gems. It was offered on the free market in the form of floppy disks distributed at computer "user groups". It was also made available on on-line computer systems (we'll say more about these topics later).

Free CP/M programs are the equivalent of a software gold mine. Buried behind the rubble of poor documentation and zero customer support exist hundreds of excellent programs, provided you can weed them out. How much free software exists? Public domain software is measured in volumes. A volume is a single diskette containing between 10 and 20 programs and holding 250,000 characters. One catalog lists over 250 volumes of free CP/M software! This free software is available from user groups located across the country who will send numerous volumes just for the cost of copying. A second major source is *electronic bulletin board systems*, called Remote CP/M computer systems or RCPMs. These on-line CP/M computers have large hard disks containing hundreds of free CP/M programs. They are accessed by modem, and the programs are "downloaded" to your computer with special free communications software.

In this section, we will examine the CP/M public domain phenomenon, explore some of the better free programs, and tell you how to find them, how to buy them, and how to download them.

What Good Free Software is Available

The most popular programs in the public domain are programmer tools and system utilities. There are programs for fixing damaged disks and unerasing disk files, as well as for backup utilities and enhanced directory programs. There are also a dozen communication programs. Let's look at the most popular CP/M public domain programs that a Commodore 128 CP/M 3.0 user might want.

Modem7

Modem7 (also associated with a companion program called XMODEM) is the most popular free program in the public domain, and, unlike most software, has literally hundreds of versions. Originally written by Ward Christensen, Modem7 has been revised and improved by dozens of programmers. Modem7, or a similar communications program with XMODEM protocol capability, is the key to getting free software from bulletin boards. A special accuracy-checking feature of Modem7 insures that not one single error occurs in the transfer. Without such checking, it is almost certain the program will not run properly. Since modems for CP/M are not standardized, Modem7 must be customized for each CP/M computer it runs on. Luckily, a version of Modem7 has been adapted for most of the CP/M machines and modems on the market, and it will not be long before you can find a version of Modem7 for the Commodore 128. As a tribute to the importance and ubiquity of Modem7, today most commercial CP/M

communications programs on the market support the Modem7 file transfer protocol.

Squeeze and Unsqueeze (SQ and USQ)

The SQ program is used to squeeze files, that is, make them more compact in size, so that they can be transferred by modem in quicker time. Then the USQ, or unsqueeze program, is used to bring them back to normal. This can save telephone charges, as well as give more capacity than is obvious on a disk.

Disk Utility (DU)

The disk utility (DU) lets you perform intelligent disk repair to recover erased files, isolate bad sectors from use, and make data transplants on a disk when the disk seems totally dead. DU is a fairly complex programmer tool, but is extremely popular because it allows a CP/M progammer to become a disk surgeon. DU is also popular for cracking disk protection schemes and analyzing CP/M storage formats.

FindBad and UNERA

FindBad locates bad sectors on your disk and isolates them from future use so you can use a disk you might normally discard. UNERA helps you recover files you have accidentally erased from a disk. (Even though you erase a file with the ERASE command, the file still exists on the disk — only its name has been removed from action.)

Dan's Information Management System (DIMS)

Here is a complete database management system that is said to rival dBASEII. The program is a system of Microsoft BASIC programs chained together and designed so it can be used by anyone. Written by Dan Dugan, it is suitable for medium-scale mailing-list operations, indexes, ledgers, and similar operations. Records can be up to 256 characters long with up to 30 data fields, and file size can be the limits of the disk itself.

Other Good Public Domain Software

The Osborne/McGraw-Hill CBASIC2 Accounting System provides a complete small business accounting system with books to back it up, provided by the publisher Osborne/McGraw-Hill. There are dozens of languages, including a BASIC-E compiler from Lawrence Livermore Laboratory, BDS-C, several versions of Forth, a PILOT, and even a Pascal compiler.

You'll also find game programs like the original StarTrek program, and other pre-video-game games. There are also assemblers, search-and-rescue programs, speedup utilities, music programs that play three-part harmony, disk cataloging systems, ham radio programs, fast copy programs, and lots more.

How to Get Free Software

The two methods of exchanging free software are by disk via user groups and by computer-computer connection over the telephone lines via bulletin boards. Let's look at these in turn.

CP/M User Groups

A CP/M user group is a group that is devoted to cataloging and exchanging information about CP/M for nonprofit. User groups usually meet twice a month and are located across the country. You can find out about user groups in your area by reading the magazines listed at the end of the chapter, by reading the local newspapers that advertise computer consulting, and by asking around at a local school or computer store. User groups usually offer free software to anyone, even nonmembers, and some of the larger ones will duplicate and mail disks of software for a nominal charge.

One of the largest groups offering CP/M public domain software suitable for Commodore 128 users is called CP/MUG, which stands for CP/M User Group. This group is supported by Lifeboat Associates in New York. CP/MUG provides 5¼ inch CP/M single-density disks in the Kaypro formats for $18 per disk.

Another large user group is the First Osborne Group (FOG) located in Daly City, California. FOG distributes CP/M public domain software in Osborne, Kaypro II, Morrow, CompuPro, PCM Micromate, and Zorba formats. FOG has over 15,000 members worldwide and costs $24 to join.

The New Jersey or New York SIG/M (Special Interest Group/Microcomputers) offer perhaps the largest catalog of CP/M public domain software. Unfortunately, this is not provided on 5-1/4 inch formats.

Bulletin Boards

The second way to get free public domain CP/M software is by calling up a remote CP/M system bulletin board and downloading it. Thousands of these systems exist across the country, maintained by kind, diligent, benevolent individuals, who simply live to see people use their CP/M systems. Most of these systems contain the latest releases of volumes of

SIG/M, FOG, and C/MUG software. You can preview these programs before downloading them by reading a special description file first. These RCP/Ms serve as distribution points for SIG/M and FOG software, and their telephone numbers are distributed by these user groups. There are over twenty-five FOG RCP/Ms across the United States. Contact FOG at the address in Appendix A for the exact phone numbers for your area.

CompuServe and The Source are other sources of free CP/M software. Both these on-line services have CP/M special interest groups. Programs written in CP/M are available for downloading to your computer, and a way of leaving mail for other CP/M members is provided.

The Structure of CP/M: Layout, Commands, and Utilities

Now that you are familiar with what kinds of software can be found for CP/M, you are ready to get an overview of how CP/M is structured, the numerous commands it offers, and the utility programs provided to make your work easier.

Welcome to CP/M 3.0

CP/M has been around for some time. It first appeared in 1975 as CP/M version 1.3, and, shortly after, as version 1.4. In 1979, CP/M version 2.2 was introduced. It took advantage of larger memories and cheaper disk drives made available by the microcomputer boom, and has remained the most popular version to date. It is estimated that there were over 1.5 million copies of CP/M in circulation in early 1983.

In 1983, Digital Research developed a new version of CP/M called CP/M 3.0. Its name was switched to CP/M PLUS at the last minute, but the name CP/M 3.0 caught on and has been used ever since. CP/M 3.0 was an improved version of 2.2, with a couple of new commands and one major difference: it was capable of working with larger memories then 64K. Two versions of CP/M 3.0 were created: a banked version for larger memories and a nonbanked version for smaller 64K memories. The banked version of CP/M was intended to allow two programs to run at the same time, each in its own area of memory. For technical reasons, the Commodore 128 uses the nonbanked version of CP/M 3.0 and does not directly use the extra memory in the computer. However, it is possible through rather subtle methods, to access portions of the extra memory area.

Although programs are not written in CP/M 1.3/1.4, some still exist. You should know that from a compatibility standpoint, programs designed

to run under CP/M version 1.3/1.4 will not always run under version 2.2 and 3.0. Further, programs that run under 2.2 and 3.0 won't always work under version 1.3/1.4. However, most programs written for version 2.2 will run under version 3.0 without alteration.

Getting Started in CP/M

Unlike BASIC 7.0, which is installed in a ROM inside the Commodore 128, the program for CP/M is stored on the disk and must be read into memory before it can be used. This process is called *booting* CP/M because it is like kicking it to get it running (see Figure 5-3). There are two programs on the disk that are loaded when the CP/M disk is started: CPM.SYS and CCP.COM (we will explain these names later — for now they are just programs on the disk). When this booting process is completed, CP/M has fully "installed" itself in memory and will keep running until you turn the computer off. When you start a program under CP/M, it will run until you tell it to stop, and then you will end up back with the CP/M program running. Assuming you have an 80-column monitor, after you boot your CP/M disk, your screen should show something like this:

```
CP/M 3.0  On the Commodore 128  13 March 85
80 column display

A>_
```

The A> will be at the bottom of the screen. It represents the CP/M *prompt*. A prompt is a symbol or character sequence the operating system displays when it's awaiting your input. It means, "I'm waiting for you to type something." The letter A refers to the drive that we are using; the *booting drive* is called the A drive. By booting drive, we mean the drive that CP/M was started from. If you have a dual drive connected to your Commodore 128, the second drive is referred to as the B drive. This lettering allows you to switch between drives and to specify programs as being on either drive. The "greater than" symbol (>) is CP/M's way of saying "type your input right here". Following it is the Commodore 128 CP/M blinking underline cursor. The cursor represents where the next character you type will actually appear.

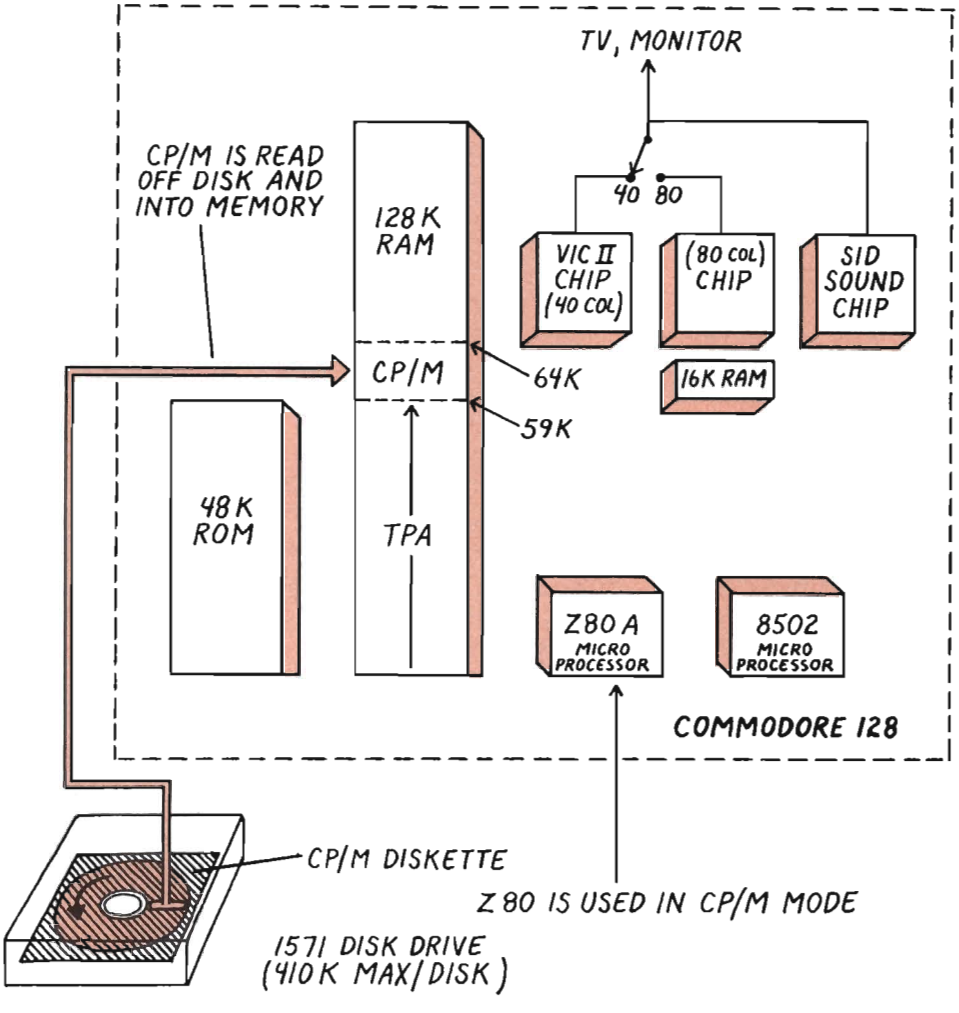
So Where Is CP/M?

Keeping track of where CP/M is located in memory is not important to its everyday use. However, it is important when it comes to purchasing ap-

plication software that was not designed specifically for the Commodore 128 CP/M. The reason is that different CP/M programs expect and need different amounts of memory space to operate. You want to make sure you understand how much memory the C128 version of CP/M provides, so you can be sure these programs will run.

Inside the Commodore 128 memory, the part of CP/M that was loaded off the disk is now situated starting at about the 59,000th memory location

Figure 5-3. The CP/M Mode Uses the Z80A Chip and Either Display Chip



and extending up to approximately the 64,000th location. Below this is a larger memory area where CP/M programs are loaded and run from. This area below the location CP/M is called the TPA (which means transient program area: it is where transient programs are loaded and run). As long as the CP/M program you wish to run does not need more than 59K, it will work fine in CP/M 3.0. Very few CP/M programs on the market will require a TPA larger than 48K. Thus the Commodore 59K TPA provides more than enough room.

As long as the CP/M program you wish to run does not need more than 59K, it will work fine in CP/M 3.0.

Files in CP/M

CP/M accesses and manipulates information stored on disks in files. A file is any information that is stored as a single entity and given a unique name. A file can be a single program, a chapter of this book, an entire mailing list, the data for a program, a single letter, or a complete novel. A file can be as short as a single character or as long as all the capacity of the disk. There are no basic rules for what goes in a file — you define its contents when you create the file.

There are rules for telling one file from another; these rules are to give each file a distinctive and unique name. In CP/M, all files are referred to by a name consisting of one to eight characters called the *filename*. In CP/M, no two files can have the exact same name and only uppercase letters are allowed in filenames. There are also limitations in CP/M as to what letters and symbols may be used in filenames. Characters you can't use include: < > . ; : = ? * [ ] ( ) / or tab. Some valid filenames are:

CHAP-12

PIP

BASKET

PROGRAM5

A347689

ME@

BIG-FILE

RT+XT

CP/M contains numerous commands that allow you to manipulate these files. Usually you must specify the name of the file you wish to

manipulate. To make this a little easier, CP/M provides special characters, called *wildcards*, that let you operate on more than one file at a time, and let you perform operations on groups.

Filetypes

When you use CP/M it is useful to be able to identify files in a way that lets you know what they are going to be used for. For example, a text file and a program file are really different, but you might not know it by their filenames. CP/M allows you to tell what kind of file you are using by way of a three-character extension added to the right of the filename and chosen from the list in Table 5-1. The *filetype* is separated from the filename by a period; that is why a period can't be used in the filename.

If you can identify the filetype, you can pretty easily figure out what kind of file you are dealing with. First, it is important to distinguish between two kinds of files: data files and program files. A data file is usually one that contains text, numbers, and collections of characters. A typical data file is a document created on your word processor that is made up of text. In CP/M, this would normally have the extension TXT to stand for text or DOC to stand for document. Data files can usually be examined from CP/M using the TYPE command (as we shall soon see).

A program file, on the other hand, contains instructions and codes in a particular computer language. For example, a file with the extension COM is called a command file and contains codes that run the Z80A in your C128. A program you wrote in Microsoft BASIC might have the extension BAS. Table 5-1 lists the most common filetypes for CP/M. The books listed at the end of this chapter, as well as the Commodore 128 manual, explain these in more detail.

Note that you do not have to stick to the list of filetypes given in Table 5-1. You can be creative and use names that make the type of file apparent. Here are some typical examples of complete filenames with filetype extensions:

MYPROG.BAS (a BASIC program)

JOE-3-14.LET (a letter to Joe written on 3-14)

PIP.COM (a CP/M program for copying files)

CHAP2.BAK (a backup of the text file CHAP2.TXT)

PERFECT.OVL (an overlay file for Perfect Writer)

A347689.PBS (a poor choice for a name)

CP/M's Built-in Commands

Commands are instructions that tell CP/M what to do. We give commands to CP/M by typing the command's name after the A> prompt and pressing the [Return] key. There are two kinds of commands in CP/M and the distinction between them is a subtle one. There are "built-in commands" and "transient commands". The short programs that carry out built-in commands are always in memory when CP/M is started up and are a part of CP/M itself. The programs that carry out transient commands are not part of CP/M and are not automatically loaded when CP/M is first started. Instead, when you issue a transient command, CP/M goes out to the disk and searches for the program containing the command, loads it into memory, and then executes it.

The CP/M built-in commands execute quickly because they are already sitting in memory. The transients, on the other hand, are slower

Table 5-1. CP/M 3.0 Filetypes

Extension	*Type*
.ASM	Assembly language source file
.BAK	Backup file
.BAS	BASIC source program
.COM	Z80 or 8080 program
.DAT	A data file
.DOC	A document file
.FOR	FORTRAN program
.HEX	Hexadecimal file created by ASM and MAC
.INT	Intermediate work file
.LIB	Library file used by ASM, MAC, and RMAC
.PAS	Pascal source program
.OVL	Overlay file used by another program
.OVR	Overlay file used by another program
.PL1	PL/1 source program
.PRN	Print file created by ASM, MAC, and RMAC
.REL	Relocatable file created by RMAC
.SUB	Source file used by SUBMIT
.SYM	Symbol table created by ASM, MAC, and RMAC
.SYS	A CP/M system file (used by CP/M)
.TXT	A text file
.XRF	A cross reference file
.$$$	A temporary file

because they are programs that must first be located on the disk and then loaded into memory. However, transients can be much larger in size than the resident memory-based commands and, consequently, can do more. Transients are frequently referred to as "utility" commands because they normally help you manage and perform utility operation on files and disks.

In the case of CP/M 3.0, the distinction between built-in and transient commands is even more vague. Four of the built-in commands have transient "extensions" that are executed if you need additional features. The six built-in commands are listed in Table 5-2. As a user of CP/M, the only reason you need to care about whether a command is transient or built-in is that, if it is transient, it must be on the disk. If you type the name of a transient command and it is not on the disk, you will get a message: File not found, meaning CP/M could not find a file with the name you gave. Thus, built-in commands are more convenient to use than transients.

Of these six built-in commands, two do not have transient extensions (that is, counterparts), namely DIRSYS and USER. When you look on your CP/M system disks, you will find, in addition to numerous utilities, the programs DIR.COM, ERASE.COM, RENAME.COM, and TYPE.COM. These are the transient extensions that are executed if we request additional parameters that the in-memory built-in commands can't offer. Additional parameters are called "options" or option parameters in CP/M 3.0 and are indicated by placing additional information in brackets after the command itself.

For example, to get a simple listing of the files on a disk, you can use the built-in DIR command, but to get the listing along with the size of the files, you use DIR[SIZE] which will automatically invoke the transient version of DIR. All options in CP/M 3.0 are typed between brackets.

Table 5-2. Built-In CP/M 3.0 Commands

Command	*Function Provided*
DIR	Displays the disk directory
DIRSYS	Displays the directory of system files
ERASE	Erases a disk file
RENAME	Renames a disk file
TYPE	Displays a text file on the screen
USER	Changes the user area

The new rules for CP/M 3.0 are as follows: If RENAME, ERASE, or TYPE are given without a filename, then the transient version is executed and you are prompted for more information. For example, if you enter the command TYPE without any filename after it, the transient version is invoked (read off the disk and executed), and you'll be prompted with the message "Enter filename:". If the commands DIR, ERASE, or TYPE are given with a filename and with an option parameter, the extended transient version will operate. If this seems overly complicated, you're right, it is. Before CP/M 3.0, transients and built-in commands did not overlap. Basically, the reason for the overlap is to allow the built-in commands to have additional and more powerful actions without changing their names.

Another CP/M built-in command changes the *default* disk drive (default means the one showing the prompt). If the current drive is A and you enter the new drive followed by a colon (that is, you type B:), CP/M will change the default drive to B.

Now that you understand the differences between the built-in and transient commands, let's examine the built-in commands in more detail.

DIR and DIRSYS

The DIR command (pronounced dee-eye-are) is used to display the names of the files on a diskette on the screen. The files can be on either the A or the B drive. The built-in version of DIR simply displays the names of any or all files. On the other hand, the transient version of DIR allows eighteen different options, including the ability to get the size of each file in bytes, the total number of bytes consumed by all the files, and a sorted listing of the filenames. You can also have DIR.COM tell you the attributes of the files — that is, if they are read-only, system files, read/write, or directory files.

ERASE, RENAME, AND TYPE

The ERASE command is used when you want to remove a file from the CP/M diskette. Once the file is removed, you cannot get it back (unless you have a special utility that unerases — see the previous section on CP/M software utilities), so you must be very careful when using this command.

The RENAME command is used to change the names of any files. For example, you might want to rename a file CHAPONE.TXT to CHAPTWO.TXT. RENAME will do this without affecting the contents of the file. When you use DIR to list the files, you will see CHAPTWO where you used to see CHAPONE.

TYPE is the CP/M built-in command for viewing the contents of a text file or data file. TYPE is one of the first commands to become familiar with, as it provides an easy method for examining the contents of a file on the screen. You do not want to use TYPE to try to view the contents of a nontext file, such as a COM file. This may send improper characters to the video screen that could make it lock up and require a rebooting of CP/M.

USER

Each CP/M disk can be partitioned or divided into 16 regions or areas, called user areas, which are numbered 0 to 15. Each area can be assigned to hold files for different users of the disk or different subject areas. User's areas are best used on a hard disk where several people may share it. Since you probably won't have multiple users on the Commodore 128, the USER command will most likely be used occasionally for storing different categories of files.

When you first start CP/M, the system automatically selects area 0. You can change to another area by giving the USER command followed by the user number. The files in one user area are not accessible by a different user area unless they are first copied to that area. All the built-in commands are available when you are in a user area. When you switch to a new user area, the CP/M prompt reflects the new area. For example, typing USER 9 will change to user area 9 and produce the prompt 9A>. Use the user areas with caution, as it is easy to misplace files in the areas, and finding them is time consuming.

CP/M Transients and Utilities

There are six important CP/M 3.0 transients that are used fairly often, and these are listed in Table 5-3. Let's go over each, one at a time.

CP/M Video Aid: HELP

One new feature of CP/M 3.0 that is lacking from earlier versions is the HELP transient. HELP.COM is used to display on the screen information about the different commands and transients in CP/M. It uses a very large work file called HELP.HLP, that contains most of the text it uses. To use it, you type HELP followed by a topic, such as HELP PIP. It in turn displays a terse listing of what the command (PIP) does. HELP can't be used when the computer is running a program. If you have a good book on CP/M, you may not want to use the HELP command and its large file to save room on the disk.

Setting Up CP/M Disks: FORMAT, COPYSYS

Information is stored on the surface of the CP/M diskette as a series of blocks (called sectors). There are many different arrangements possible for locating these sectors on a disk, and because of this each disk must first be prepared to receive information in the desired format. We call this preparation "formatting", and it must be done before a blank, out-of-the-box diskette can be used to store CP/M files. The transient command for formatting a CP/M disk is called FORMAT. Once you have run FORMAT on a blank disk, it is in a state that can be read, but it contains no information. A DIR performed on a formatted blank disk will give the message: no files. Because of its intelligent disk drives, the Commodore 128 version of FORMAT requests you to tell it the type of CP/M computer you wish the diskettes to be compatible with.

Once a CP/M disk is formatted, it is ready to receive files. But such a disk cannot be used to "boot" or start CP/M because it still does not have the CP/M program itself on it. In CP/M 3.0, the CP/M program or "system", as it is called, is stored partially in some invisible sectors of the disk and partially as two visible files: CPM.SYS and CCP.COM. When the C128 is first started, it checks if a CP/M disk is in the drive. If one is found, the first thing that happens is the CP/M system is loaded off the invisible sectors and into memory. Control is passed to the system program in memory, and it in turn reads the larger files, CPM.SYS and CCP.COM, into memory. These files contain the bulk of the code to complete the CP/M system. The COPYSYS program is used to both format a disk and to copy the CP/M system in the invisible sectors from one disk to another.

Table 5-3. Often-Used CP/M Transient Commands

Command	*Function Provided*
HELP	Provides help on CP/M subjects on screen
FORMAT	Used to format a blank diskette
COPYSYS	Creates a new bootable CP/M disk
PIP	Used to copy files and backup files
SHOW	Gives statistics on files
ED	Standard CP/M editing program

Copying Files: PIP

Once you have formatted a diskette and installed the CP/M system with COPYSYS, you are ready to copy files to the disk with the PIP utility. PIP stands for Peripheral Interchange Program (PIP), and allows CP/M to move files not only between disks, but also from disk files to printers or from disk files to the screen. In CP/M, the disk drive, printer, and screen are all "peripherals" that can be accessed directly from CP/M.

PIP's most important function is copying files from disk to disk. There are a large number of options you can use with PIP. For example, you can ask PIP to "archive" files, meaning you only copy a file if it was modified since it was last copied. This feature saves time when you are "backing-up" the files on a disk. Other PIP options request permission to continue, echo the copying on the screen, filter out special characters, or convert uppercase letters to lowercase. PIP can copy single files or groups of files, change names as it copies, print a group of files, extract a portion of a file, join several files into a single file, and much more.

Determining File Size: SHOW

Before you can copy files from disk to disk, you need to make sure there is enough room or space on the destination disk to hold the file. The SHOW transient is used to find the remaining disk space. When it is executed, SHOW returns the amount of Read/Write space on the disk. SHOW is also used to tell the number of directory entries. CP/M limits the maximum number of files that can be on the disk to 64. A third feature of SHOW, invoked with the option [USER], will reveal the number of active user areas and the number of files contained in each area. Another option of SHOW, called [DRIVE], will produce a listing of the disk characteristics, such as total capacity, directory space, and block size.

The System Editor: ED

ED is a very primitive text-editing program called a "line editor" that is provided on every CP/M diskette. ED is not useful for serious word processing, but is rather for entering very short text files, such as batch files or short messages. Anyone considering serious text editing should purchase a professional word processing program, such as Perfect Writer or WordStar.

The Less-Used CP/M Transients

There are many more transient commands in CP/M 3.0 which are less often used than the previous ones we have learned about. Table 5-4 lists these. Let's examine these further.

DATE and FKEYS

The DATE transient command is used to set or to display the internal time and date. The Commodore 128 keeps this information in a real-time clock. When you use DATE without options, it displays the date and time. With options, it sets the date and time.

FKEYS is used to customize the Commodore 128 function keys so they can elicit complete CP/M commands. For example, you could set up the F1 function key so that when pressed it reproduces DIR *.COM [FULL, EXCLUDE], and RETURN. This will cause a full listing of all the non-COM files on the disk and save typing 26 keystrokes.

DEVICE and Logical/Physical Devices

CP/M was designed from the start to work with different kinds of peripheral units or "devices". In Commodore 128 CP/M, the keyboard, printer, screen, and phone modem are all considered peripheral devices. CP/M also has what are called "logical devices". A logical device represents a particular function of your computer, while a physical device is the specific piece of equipment you choose to perform that function. We convey our choices to the Commodore 128 by using the DEVICE transient command.

Table 5-4. Less Used CP/M Transient Commands

Command	*Function Provided*
DATE	Displays and alters the time and date
DEVICE	Displays and alters peripheral assignments
FKEYS	Assigns commands to C128 function keys
GET	Inputs information from a file instead of keyboard
INITDIR	Prepares a disk for time- and date-stamping of files
PUT	Sends ouput to a disk file instead of screen or printer
SET	Changes file attributes, assigns a label, sets up a password
SETDEF	Sets up the disk search path for finding files
SUBMIT	Executes commands from a disk file

This selection of choices makes it possible to assign different physical devices to different logical entities that CP/M communicates through. This may come in handy when you are trying to control several different devices connected to your Commodore 128. The DEVICE transient is used both to set the assignments and to display their current settings.

GET, PUT, and INITDIR

The GET command allows what is known as true I/O redirection, that is, it lets you take input from a source other than the keyboard — in this case, a disk file. GET is handy for automatically running programs, or portions of programs, that require the same input each time they are run, such as the repetitive and annoying requests that programs sometimes make. To use GET, you would produce a file with your text editor or word processor that contains the information you want to feed to the next program. Then you type GET FILE filename.typ. Next you run the program that you want the GET information sent to. Your program will run, but it will automatically be filled in by the contents of the GET file you created. You might use GET to start a word processing program and give it the commands to log in the B drive and open a certain file.

PUT is the opposite of GET, and, instead of receiving input from a file, PUT writes a program's output to a file for later use. For example, you might want to capture several program outputs as text files and then enter them in a word processing program for editing.

CP/M 3.0 includes the feature of time- and date-stamping of files. What this means is that when you create or alter a file in CP/M 3.0 and time- and date-stamping is enabled, the file will be saved with the time and date attached to it. This same time and date will be displayed in the directory with DIR and SHOW. To enable time- and date-stamping, the INITDIR command must be issued after the disk has been formatted and before any files are copied to it.

Passwords, SET, SETDEF, and SUBMIT

SET is used for several purposes. The most common use of SET is in controlling the read/write attributes of the disk files. File attributes are the way we indicate what kind of file we are dealing with. A file can be R/O, or read only, R/W or read/write, SYS, meaning a system file, or DIR, meaning a directory file (this is the normal attribute given to all files). With SET, you can prevent anyone from modifying a file or a group of files. You can also prevent anyone from even reading or copying a file by specifying that a *password* first be typed before the file can be accessed.

Passwords are names that are assigned to individual files. When a password has been assigned, a user cannot copy, delete, or read that file without typing in the correct password. SET also allows you to turn the time- and date-stamping feature of CP/M 3.0 on and off.

First-time users of CP/M often have problems when they are logged onto another drive (say B) and type in a command that is stored on drive A. SETDEF, among other things, allows you to set up a "search path" for CP/M to try when responding to a typed command. Thus you could tell CP/M to first look on the A drive, then on the B for a particular program.

A sequence of commands that is normally given to CP/M from the keyboard can be placed in a disk file and processed by the SUBMIT transient. Input to a program can be included. CP/M will execute the file just as if these commands were coming from the keyboard. When the list of commands is exhausted, control returns to the keyboard. This operation is known as batch processing. You would normally use SUBMIT when you wanted to give CP/M a frequently used list of commands — for example, if you wanted to copy ten different files to ten different disks. A submit file could contain the ten filenames and the PIP operations, and you would only have to submit the file to SUBMIT ten times instead of 1,000. You could even make the submit file automatically request that you enter the next diskette.

Programmer's CP/M Utilities

There are several utilities used primarily by assembly language programmers provided in CP/M 3.0. If you are not going to be writing assembly language programs, you won't need to worry about them. Table 5-5 lists these utilities.

Table 5-5. Programmer's CP/M Transient Commands

Command	*Function Provided*
DUMP	Displays hexadecimal contents of COM files on screen
HEXCOM	Generates an executable COM file from an Intel HEX file
LINK	Creates executable COM file from relocatable modules
MAC	The CP/M Macroassembler
RMAC	The CP/M relocatable macroassembler
SAVE	Saves a portion of memory as a disk file
XREF	Creates a cross reference listing from MAC/RMAC tables

The MAC and RMAC transients are very powerful programs called assemblers. An assembler converts a program typed in by a programmer into a group of special codes that are instructions to the Z80 chip in your Commodore 128. The original source file is written with a word processor. The language used is called assembly language. The code produced by the assembler is called object code. MAC is an enhancement of an older CP/M assembler called ASM and has the addition of "macros". A macro is a compact single-line instruction that actually represents a much more elaborate collection of instructions. RMAC is a relocatable assembler, which means it allows the programmer to write code that can be relocated anywhere in memory and still run. MAC creates absolute nonrelocating code.

HEXCOM, LINK, and XREF are tools that MAC and RMAC programmers use. SAVE is a transient that is used to take the contents of the TPA and save it to a file. Only assembly language programmers will normally use SAVE.

Where Can You Learn More About CP/M?

There are numerous ways to find out more about CP/M. Many good books are available which explain in great detail how to use the CP/M operating system. There are books that show you how to write programs that run under CP/M and that expose the internal structure of CP/M. There are even books which describe the best CP/M software. Here are some useful books:

CP/M Primer, Second Edition, by Stephen Murtha and Mitchell Waite (Indianapolis: Howard W. Sams & Co., Inc., 1983). A simple, low-pressure introduction to CP/M and its utilities. Unique in its accessibility to beginners.

Osborne/McGraw-Hill CP/M User Guide, Third Edition, by Thom Hogan (Berkeley: Osborne McGraw-Hill, 1984). This book is more detailed and comprehensive than CP/M Primer but also more complex to learn from. Includes all versions of CP/M, including CP/M 3.0 as used in the Commodore 128.

The CP/M Plus Handbook, by Alan R. Miller (Berkeley: Sybex Computer Books, 1984). This book is completely devoted to CP/M 3.0 and, except for a few details, is compatible with the CP/M used in the Commodore 128. Not as comprehensive as the Osborne/McGraw-Hill CP/M User Guide.

Soul of CP/M, by Mitchell Waite and Robert Lafore (Indianapolis: Howard W. Sams & Co., Inc., 1983). This is a very easy introduction into the internal structure and programming of CP/M. Shows how to write assembly language programs that use the built-in CP/M system calls.

CP/M Bible, by Mitchell Waite and John Angermeyer (Indianapolis: Howard W. Sams & Co., Inc., 1984). This book catalogs all the CP/M commands and transients in a unique, accessible format.

Another source of CP/M information is magazines. In these, you will find names and addresses of companies that sell hardware for your CP/M computers, as well as commercial software and articles on using CP/M programs. Here are a few of the better magazines:

Byte, Peterborough, NH. Byte often discusses CP/M and provides reviews of hardware and software. Jerry Pournelle is a spokesperson for CP/M and gives his sometimes caustic opinions each month.

InfoWorld, Menlo Park, CA. This magazine has a long history of breaking the latest news in the micro and personal computer industry. It has several reviews each month and rates all products. Its coverage of CP/M has waned over the years as the IBM PC has grown.

Newsletters are a good source of CP/M news.

Digital Research News is produced by the creators of CP/M and presents the newest product releases, program developments, and bugs and fixes.

Lifelines is a newsletter/magazine produced by Lifeboat Associates that provides many helpful hints about CP/M, including the latest public domain software offerings.

6

Graphics on the C128

In this chapter you'll learn:

- **What kinds of graphics are offered with the Commodore 128**
- **The history of Commodore 128 graphics**
- **How to use the new Commodore 128 graphics modes**
- **How bit-mapped graphics work**
- **How to make and manipulate sprites**
- **What the 80-column mode offers in the way of graphics**

In this chapter we will take a look at the exceptional graphics offered by the Commodore 128 personal computer. We will first briefly review the graphics of the C64 mode. However, the bulk of the material in this chapter focuses on the new C128 mode graphics. Since CP/M was designed to work only with text-based ASCII terminals, it has no commands for manipulating video graphics, and we will not discuss it in this chapter. It is possible however, using a CP/M programming language such as BASIC or C, to create graphics in the CP/M mode via the 8563 video chip.

C128 Graphics Overview

This chapter is not meant to provide an exhaustive study of Commodore 128 graphic programming techniques, but rather to give an overview of the features you get with C128 graphics, to provide insights into how the Commodore 128 uses the VIC II and new 8563 chip to send video to the

screen, and finally to reveal the graphics operations now available from the BASIC 7.0. language.

What Do We Mean by Graphics?

What do we mean by graphics? In this book graphics means the display of pictures instead of just letters and numbers. A picture can be anything you can represent on the Commodore 128 screen: a rocket, a robot, an artistic pattern, an arrangement of specially designed characters, the outline and boxes of a report form, and so on.

6566 VIC II and 8563: The Chips Behind the Video

Before we get too far into learning graphics, a major point is in order. What makes Commodore graphics so special when compared to other low-cost home computers is the way in which graphics are created and controlled inside the computer. Most computers use arrangements of cheap, off-the-shelf circuits to control graphics. But the Commodore 64 and 128 share a complex chip called the VIC II, custom-built by Commodore's MOS Technology semiconductor division. The VIC II chip evolved from the VIC I chip used in the old Commodore VIC 20 and is actually an entire microprocessor and video processor. Officially known as a video interface chip, because it translates computer signals into television or monitor signals, it is the fast and smart VIC II chip that gives the C128 and C64 its edge in graphics performance. In addition to the VIC II, the Commodore 128 also contains the brand new 8563 80-column video chip. This new device gives the C128 the kind of graphics that business users demand: highly detailed 80-column color text and graphics. Keep in mind as you read this chapter that everything you are doing from BASIC ultimately boils down to instructions to the powerful VIC II or 8563 80-column chip and that these chips are ultimately the reason the graphics work so well.

What Kind of Graphics Does Commodore 128 Offer?

The C128 mode offers the same graphics modes as the old Commodore 64, plus a few important new ones. There are six different display modes, and sixteen different colors. There is a new 80-column mode. There are also eight programmable, movable objects called "sprites". Access to these modes, as we will soon see, has been greatly improved.

Like the Commodore 64, the Commodore 128 has a 40-column text and character graphics mode that is programmable in up to four colors. It has the same graphics and text character set. The C128 also has a high-resolution 320 by 200 dot *bit-mapped* mode. The new high-resolution C128 bit-mapped mode is now much more accessible and easy to use from BASIC. We'll say more on this soon.

New Graphics 80-Column Mode

The Commodore 128 offers one new graphics mode not available on the Commodore 64, namely the new 80-column mode. As we mentioned in the introduction and in the chapter on CP/M, the 80-column mode is created by the new 8563 video processor chip. This chip has features that are nearly identical to the color circuitry used in the popular IBM PC color board, namely 640 x 200 resolution, and 16 colors. This chip allows twice the horizontal resolution of the Commodore 64 graphics (640 dots versus 320), four times more color (16 colors versus 4), and twice as many characters (80 versus 40). This chip does require an 80-column monitor to use and will not work with a regular television set.

This new mode also offers a bit-mapped character set, meaning that the shapes of the characters that appear on the monitor are stored in memory. This means you can create your own custom characters and even do bit-mapped graphics on this chip. Since there are no BASIC 7.0 statements to easily control the 8563, you will need to resort to using BASIC's PEEK and POKE statements to directly manipulate the registers of the new 8563 chip. This may make your programs more complex, but high-quality business graphics are now perfectly feasible from BASIC. With the 80-column chip and its high resolution, you can expect to see much more intense and detailed graphics programs, especially high-quality, business programs for the "power user".

New Additional Memory

The C128 has twice the memory capacity of the C64. This extra memory is desperately needed. In graphics the amount of memory that is available to the programmer has a strong bearing on the degree of complexity and the size of the program that can be written. The problem is that the graphics information that you see on the Commodore 128 screen is actually represented in the computer's memory as a series of bytes. As we start to draw more complex or more detailed images on the C128 screen, we begin to consume more and more RAM memory. For example, one popular technique with games is to store several different graphics pictures (scenes) in the computer's memory at one time. You then make the

VIC II rapidly switch between them, thereby creating brilliant animation effects, like those found in arcade games. Since each of these scenes consumes a piece of memory, we eventually reach a limit on how many can be stored.

The extra 64K of memory in the Commodore 128 means that programs can have more complex and memory-hungry graphics programs. There can be many more graphics scenes stored in memory, as well as more depth and realism to the graphics. This means you can expect to see far richer games than ever before, with many more variations. This same idea applies to the user of custom character sets and sprites, which we will explain soon.

New Graphics BASIC Statements

A major change in the graphics potential of the new C128 mode has to do with the way the graphics are now accessed and manipulated by the BASIC programmer. In the Commodore 64 the BASIC programmer was forced to use the commands POKE and PEEK to manipulate graphics. (Very briefly, the POKE command is used to change the contents of memory locations and the PEEK command is used to read the contents of memory locations.) Plotting a dot on the screen using these commands is quite a bizarre process. First you would have to figure out where in memory the bit corresponding to the dot you want to change is located (as we shall see, dots and bits are related). Then you must use a complex combination of POKE and PEEK statements to change this single bit without changing the neighboring bits around it. Another use of the POKE was to change the way the VIC II handled memory. Certain addresses in the VIC II would allow you to change where the chip looked for screen memory and how it interpreted this memory. One would frequently see statements in C64 programs like this; one that simply changes the mode to multicolor:

```
POKE 53270,PEEK(53270) OR 16
```

Thus, without a clear understanding of the VIC II chip, and the knowledge that address 53270 was the mode register of the chip, no one but experts could really do wondrous things with graphics. Most programmers were left in the dark. Although we could give a long story about why Commodore chose to do things this way, we'll shorten it by simply saying that the reason was an early mandate to keep the cost of the Commodore 64 as low as possible. Adding any more statements to the Commodore 64 BASIC to make graphics easier would have consumed more memory and required higher costs.

The Commodore 128 changed all that. We now have 128K of RAM and a large group of BASIC statements for manipulating the Commodore graphics. In the rest of this chapter we will describe these statements and show how they drastically increase the efficiency of your programs. You can still use the POKE and PEEK statements, and there are times when they will still make sense. But generally these new BASIC graphics statements greatly simplify the job of the graphics programmer: these statements, coupled with the other powerful new BASIC 7.0 statements, will create a new generation of exciting games and educational and business programs that use graphics.

Where Did the New BASIC Statements Come From?

Most technological improvements evolve from earlier designs, and the statements that make up the new C128 mode are no exception. In 1983 Commodore brought out a cartridge called the Super Expander 64. When plugged into the cartridge slot on the Commodore 64, the Super Expander 64 adds a set of new graphics, sound, function key, and game port statements to the built-in C64 BASIC language. Although these statements greatly improved BASIC, the Super Expander did not sell in very large quantities.

What Commodore has done in the new C128 mode is take the enhanced statements from the Super Expander 64 and place them in the ROM of the BASIC 7.0 language, thereby forcing them to be a standard. Anyone who buys a C128 gets the extended graphic statements. We can now expect to see dramatic use of these statements by the progamming community, and by magazines creating new sections that explore the C128 mode.

Now that you have a little history of the graphics for the Commodore 128, let's get a better look at just what modes are available and what they offer. The remainder of this chapter is divided into three sections: character (block) graphics, bit-mapped graphics, and sprite graphics.

Overview of the C128 Modes

There are six different graphics modes on the Commodore 128. Here we will give a brief overview of these modes.

Standard 40-Column Text and Block Graphics Mode

This mode is identical to the standard C64 graphics mode. It allows up to 1000 text or graphics characters to be displayed in a format of 25 lines of

40 characters each. Each character in this mode consists of an 8 by 8 dot region and may be in any one of the 16 available colors. This is one of the most popular modes for doing graphics because the 8 by 8 dot regions can be easily controlled. We'll have a lot to say about this mode soon.

Standard Bit Map Mode

The standard bit map mode is also referred to as the high-resolution mode. Its purpose is to allow the highest degree of precision and detail. In this mode, the screen is treated as a rectangular array of 320 horizontal dots by 200 vertical dots. In the 40-column text mode we cannot directly access these individual dots, but in the bit-mapped mode we can. As far as color is concerned, your choice is any two colors out of the available 16. One color is called foreground; the other, background. In the past, accessing the bit-mapped mode from the C64 involved the complex usage of PEEKs and POKEs. The bit-mapped mode is now easily accessed in a logical manner with BASIC 7.0 statements. In fact, as we shall see, the bulk of the new BASIC 7.0 graphics statements are specifically used to manipulate the bit-mapped mode.

Standard Bit-Mapped Mode — Split Screen

The split screen mode mixes both the standard bit-mapped mode and the standard 40-column text modes. When you request this mode from BASIC the top 75 percent of the screen is in the standard bit-mapped graphics mode, and the lower 25 percent is in the standard 40-column text mode. Thus you get five lines of text that start on line 19 and extend down to line 24. This leaves the vertical size of the bit-mapped mode at 160 pixels (200 minus 5 lines by 8 dots per line). You can even alter the line that the text mode starts on by specifying the s parameter to be any line from 0 to 24. The split screen mode is useful when you want to print information about events in high resolution. A frequent use of this technique is in games where, for example, the top of the high-resolution screen gives a view into space from a space cruiser, and the bottom of the screen gives text messages from the mother ship. You could also exploit this split screen for business graphics, perhaps to share the screen between a chart in the top half and a word processor document in the bottom half.

Multicolor Bit Map Mode

In the multicolor mode horizontal resolution is sacrificed for increased color capability. The screen horizontal resolution is cut in half to 160 dots

and the vertical resolution stays the same at 200 dots. Each dot is twice as wide as each standard bit-mapped dot, so instead of a tiny dot you get more of an extended rectangle. However, you can now display up to four different colors in each 8 by 8 dot region. The four colors are the background, foreground, and two new colors called multicolor1 and multicolor2. The Multicolor Bit Map Mode is also available in the split screen mode. The multicolor mode is not used very much in Commodore software because the large size of the pixels makes the pictures look crude.

80-Column Text Mode

In 80-Column mode we are using the 8563 chip and are allowed 80 characters per line and 25 lines, for a total of 2000 characters. In the text mode each character can be any of 16 colors. The 8563 is actually a bit-mapped chip with a bit-mapped custom character set. However, Commodore has provided no direct way (for example, via BASIC statements) to access the bit map. We will no doubt see books that describe a way to access this graphics mode via POKEs and PEEKs, and eventually there should appear an extension to BASIC that will allow direct access. This is desirable because the 8563's graphics resolution is 640 by 200 dots, which is twice that of the VIC II. In fact, this chip has identical graphics and color resolutions to that found on the IBM PC, so once BASIC is extended in this way, transferring applications from the IBM PC to the C128 should be easy.

Character (Block) Graphics Mode

Before we teach you the new BASIC 7.0 graphics statements that operate on the bit-mapped screen, it will be useful to understand how the 40-column text mode is exploited for block graphics. There is much power in this mode and no discussion would be complete without it. Since there is a split screen mode, a mixture of bit-mapped and block graphics is feasible and practical.

Character, or block, graphics are accomplished by using the C128 in the 40-column text mode. This mode is what the Commodore 128 starts in when first turned on, and can be accessed at any time with the statement GRAPHIC 0. Although the 40-column text mode appears to be the least attractive for doing graphics, it is in fact perhaps the most popular mode in terms of the developed and high-performance software that exists today. This is the mode in which most Commodore 64 software is written.

In this 40-column mode several techniques can be used for creating graphics. All these techniques make use of character graphics: the use of

special 8 dot by 8 dot pictures to form larger pictures. When using BASIC the simplest way to do graphics in this mode is to place combinations of keyboard graphic characters inside of PRINT statements. Like the C64, the C128 has a full complement of graphics characters that are shown on each key. In fact there are four possible graphics character sets you can use. When you press either the Shift or C= keys and the respective graphics key, the respective graphics character will appear on the screen. You can design a shape or object made up of graphics keys. You use the keyboard and the cursor keys to design the shape on-screen. The shape is then captured in BASIC PRINT statements.

For example, to make up an automobile image you could use type in BASIC statements like these:

```
100 PRINT "  ___"
110 PRINT "|     \__"
120 PRINT "|        |"
130 PRINT "  O     O"
```

When you run the program, an automobile-like image will appear on the screen. You can write a program that will redraw the automobile in different locations so that it appears to "run" across the screen.

With 128 different Commodore graphics characters to choose from, this mode is very powerful. Colors can also be set by using the Commodore color keys along with the graphics keys. You type the keys for the color you want, then the next graphics character typed into the PRINT statement appears in that color. Let's take a closer look at this mode.

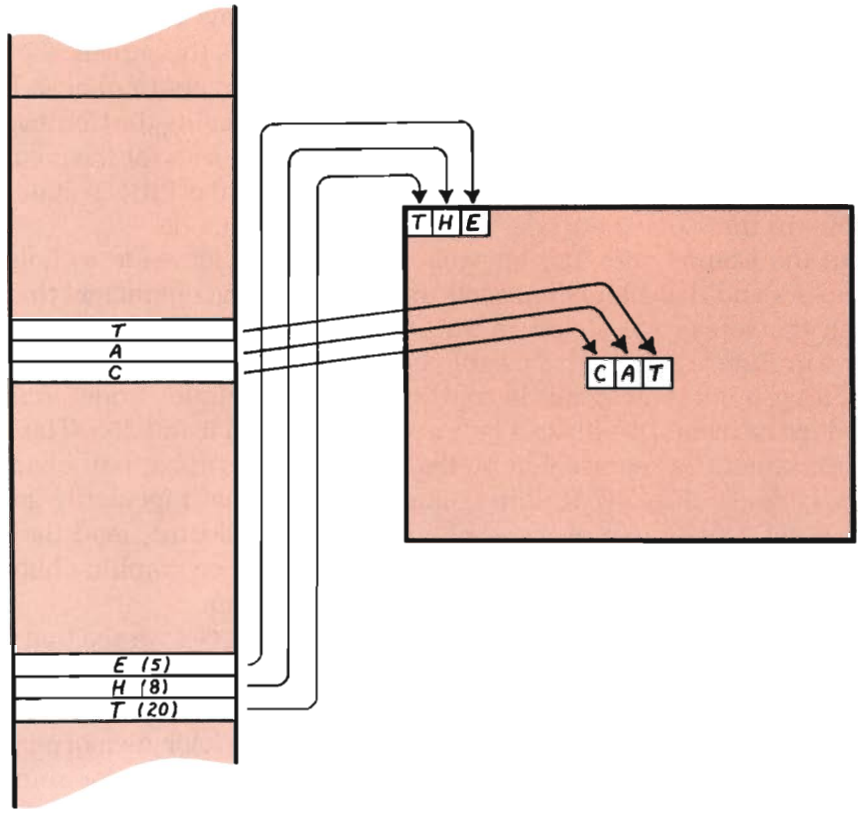
In the Commodore 128 an area of memory is set aside to hold the characters and the colors displayed in this mode. The characters that you see on the screen are stored in an area of memory in a unique way, as shown in Figure 6-1. Each possible character that you see on the screen is an 8 by 8 dot matrix and is represented by a unique "code" number stored in the memory. This can be a number between 0 and 255. The letter M, for example, is represented by the code 13; the graphic ball character by the code 81. The VIC II chip contains circuits that repeatedly go into the special screen area of memory one location at a time, read the code found there, convert it to a particular 8 by 8 letter or graphic character, and then display it at the right location on the screen.

How does each character get its color? In Figure 6-2 we see that there are actually two important areas of memory for graphics. One is the screen memory holding the character codes. The other is the color memory, which is used to hold the color for each screen location. Color memory is also 1000 characters long, but instead of codes it contains color numbers

between 0 and 15 (only the lower 4 bits of each location is used). Each location in color memory maps to a location in the screen memory. Thus when you change a color code in the color memory you are actually changing the color of a certain character on the screen. Also shown in this figure are the locations that are used to control the border, background and foreground colors if you are going to POKE them. Since the color and text memory locations are organized in the same format, you can imagine that the two memory areas are like "planes" that overlap each other. This is shown in Figure 6-2 as well.

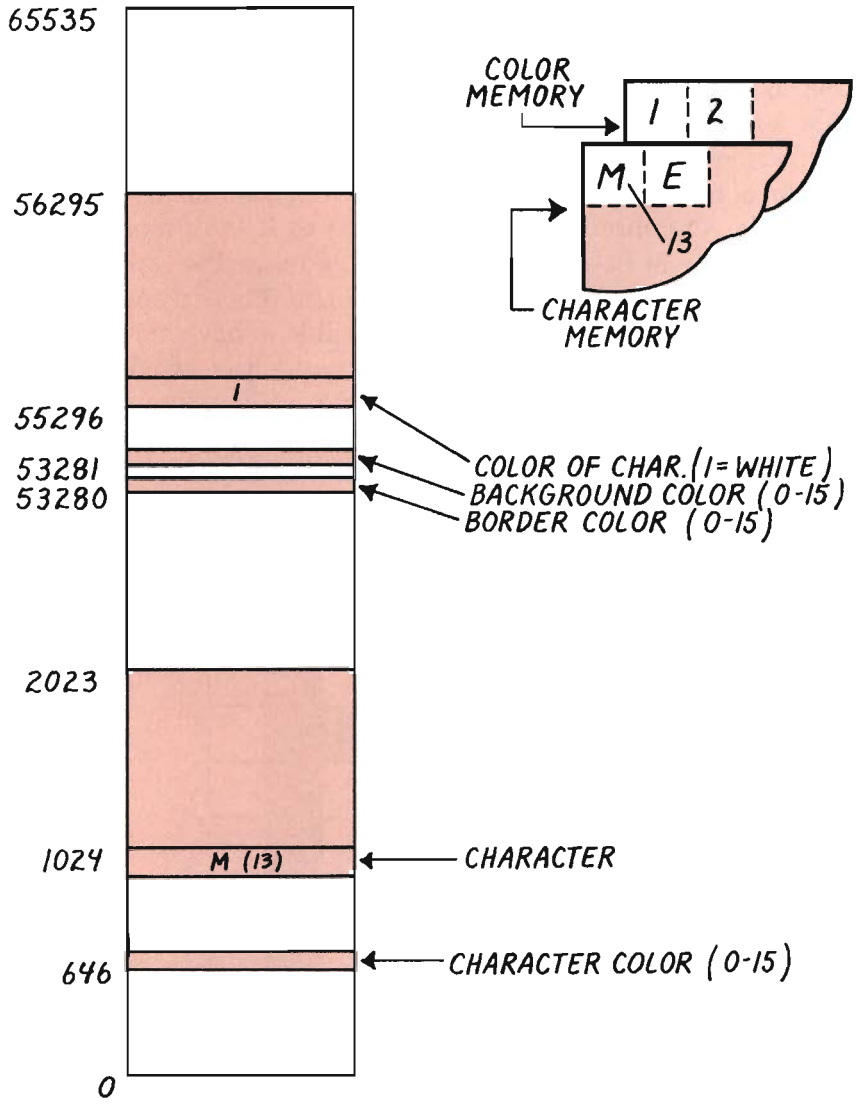
Note that although there are only 1000 characters on a 40 by 25 screen, there are 1,024 locations set aside for the text screen. The extra 24 characters are used for holding special Commodore system data.

Figure 6-1. Screen Codes in Memory Are Mapped to the Screen Display



There is a third area of memory that is critical to maximizing use of the 40 column text mode. That area is the one that holds the character set for the screen. After the VIC II chip picks up a character code from screen memory, it uses it to "look up" a particular dot pattern stored in the character font area. There dot patterns tell the VIC II chip what kind of 8

Figure 6-2. Screen Text and Color Memory Areas



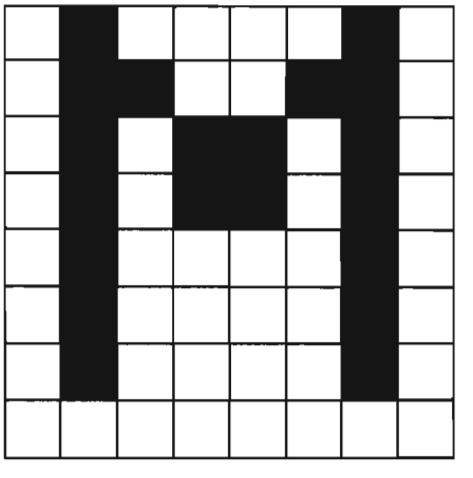
by 8 pattern to draw on the screen for the code. For example, the dot pattern for the M is shown in Figure 6-3.

Here the dark areas stand for the dots that will be turned into the foreground color, and the light areas are the dots that will be turned into the background color. Just like the screen memory, the character set memory can be made to contain any set of patterns you want. This character set can contain custom characters that, when assembled together, create pictures of great detail. In fact, several popular font editors are available for the C64 that make creating custom character sets easy.

The characters on the C128 keyboard are just not flexible enough to allow really detailed scenes. The most interesting software for the 40-column text mode exploits the use of custom character sets. In fact it is possible to have several sets of characters in memory at one time and make the VIC switch between them.

Furthermore, the location of screen memory (the part of memory holding the actual codes that tell the C128 what characters to put on the screen) in the 40-column mode is not fixed as it is in most computers. The VIC II chip can be made to go instantly to another area of memory and treat it like the new display memory. Since this screen memory only consumes about 1K bytes of RAM, it is possible to have many predefined 40-column screens waiting in memory. This idea is exploited in many games where you want the user to be faced with different "rooms" or "caves" to explore. The initialization phase of the program builds the screens; then the VIC II chip is simply directed to the beginning of the desired area.

Figure 6-3. Dot Pattern for Letter M



Besides controlling characters in the 40-column mode with PRINT statements, you can also directly POKE them into screen memory. The use of this mode is not simple, and moving the screen memory and using custom characters requires intimate knowledge of the registers of the VIC II chip. See references in the back of this chapter for books that explain this.

The 40-column text mode is particularly useful for games and educational programs, as well as for text and menu presentations. One popular game uses a custom character set and a large database of uniquely placed screen codes to simulate flying over a land mass. Trees, lakes, railroads, houses, factories, and gun encampments are made up of custom characters. From our view above a bomber airplane, we see the shadow of the plane on the ground. A feature of the VIC II chip called "smooth scrolling" moves the information on the screen by one dot row or column. This is exploited to make it appear we are flying over real terrain. In reality, different character codes are being moved in and out of the screen memory.

The Bit-Mapped Mode

Now that you understand the block graphics mode, you will find the bit-mapped mode much more straightforward. The bit-mapped mode is the mode affecting many of the new BASIC 7.0 graphics statements (the new sprite statements work with either bit-mapped or text modes). The bit-mapped mode is what you use to plot graphs and charts, draw lines, and control individual points on the display screen. In bit-mapping, each dot (called a pixel) on the screen is assigned its own bit (location) in memory. If that memory bit is a one, the dot it is assinged to is "on". If the bit is set to zero, the dot is "off". "On" means that dot shows in the foreground color; "off" means it shows in the background color.

A primary drawback to the bit-mapped mode is it consumes a great amount of memory and delivers only one color plus background, or 4 colors if you use the multicolor bit-mapped mode (but it offers only 160 horizontal dots). For example, in the C128 bit-mapped mode the resolution is 320 by 200, meaning there are 64000 dots and each needs a bit of memory. Since 8 bits are in a byte, a full 8000 bytes are needed for the bit-mapped display. Recall that in the 40-column text mode only 1000 bytes were needed to store the information for the entire screen.

Bit-mapping is best confined to programs where high speed or animation is not critical, as in engineering, CAD/CAM, business graphs, and so on. You would use the bit-mapped mode to draw highly detailed objects or complex mathematical graphs.

Setting the Graphics Mode: GRAPHICS

The Commodore 128 starts up in the 40-column text mode. To get into the high resolution bit-mapped modes, as well as to switch to the 80-column and split screen modes, you use the GRAPHIC statement. Let's see how that works. The format for the GRAPHIC mode statement is as follows:

```
GRAPHIC mode ,c,s
```

The variable mode is a digit between 0 and 5 which selects the desired mode, as shown in Table 6-1. The c parameter means clear or do not clear the screen of text after switching to the mode. If c=1 it means clear, if c=0 it means don't clear. The s parameter sets the line number at which the 40-column text screen starts in the split screen mode. It then defaults to line 19, giving 5 lines of text at the bottom of the screen.

Look at these examples:

```
GRAPHIC 0,1:REM 40 column text mode, clear screen
GRAPHIC 2,0,15: REM split screen, no clear, text starts on 15th
                line
```

The first statement switches us into the 40-column text mode and clears the screen. The second statement switches into a split screen, doesn't clear, gives us 10 lines of text starting on line 15 and a bit-mapped display of 320 by 120 pixels.

To enter the bit mapped mode and clear the screen, we would put this statement in our program:

```
GRAPHIC 1,1
```

Table 6-1. Graphics Modes for the C128

Mode	*Description*
0	Standard 40-column text
1	Standard bit map
2	Standard bit map split screen
3	Multicolor bit map
4	Multicolor bit map split screen
5	80-column text

When the GRAPHIC statement is issued with the bit-mapped mode, it allocates an area in memory for your bit-mapped screen information. The GRAPHIC CLR statement clears out this area and returns it to memory. This means the area is now available for program code. However, if you switch back into the graphics mode and your program has grown too long, you may get an error. Now that we have selected the bit mapped mode with the GRAPHIC statement, let's see how color is selected.

Choosing Colors: COLOR

The statement for choosing what colors to use for the screen foreground, background and border is set by the COLOR statement. The format of COLOR is:

```
COLOR source, color
```

where source is the section of the screen you wish to color (the foreground, background or border for a specific mode) and color is the color code (1-16) for the color you want assigned to that source. Table 6-2 lists the colors

Table 6-2. Color Codes for the COLOR Statement

Color	*Color Code*
1	Black
2	White
3	Red
4	Cyan
5	Purple
6	Green
7	Blue
8	Yellow
9	Orange
10	Brown
11	Yellow-Green
12	Pink
13	Blue-Green
14	Lt. Blue
15	Dk. Blue
16	Lt. Green

and codes used with the Commodore 128. Table 6-3 lists the six color sources and their codes for this statement. The trick here is that you can assign different colors to different modes.

Here are some examples of the use of the COLOR statement:

```
10  COLOR 0,15: REM background dark blue
20  COLOR 1,4:  REM  set foreground color bit map (1) to cyan (4)
30  COLOR 4,11: REM  set border color (4) to yellow-green (11)
```

In this example we are changing the normal color Commodore 128 defaults. The first line sets the background for text and graphics to dark blue. The second line sets the foreground color for the bit map to cyan and the last line sets the border color for the bit-mapped and text display to yellow-green.

Other Color Oriented Statements

The SCNCLR statement is used to clear the screen in any mode. For example, SCNCLR 0 clears the 40-column text mode and SCNCLR 2 clears the standard bit-mapped mode (by "clear" we mean sets it to the background color). If your program needs to know what color was assigned to a particular source, such as the color of the foreground of the bit-mapped mode, you use the RCLR(N) function. You assign a number to N to represent the source mode you want to check, and it returns a color number between 1 and 16. A similar new function is RGR, which simply returns the current mode number 0 to 5.

Table 6-3. Source Codes for the COLOR Statement

Number	*Source Affected*
0	Background 40-column text mode
1	Foreground bit-mapped mode
2	Foreground multicolor1 mode
3	Foreground multicolor2 mode
4	Border 40-column or bit-mapped mode
5	Character color 40-or 80-column text mode
6	Background 80-column text mode

Bit-Mapped Coordinate System

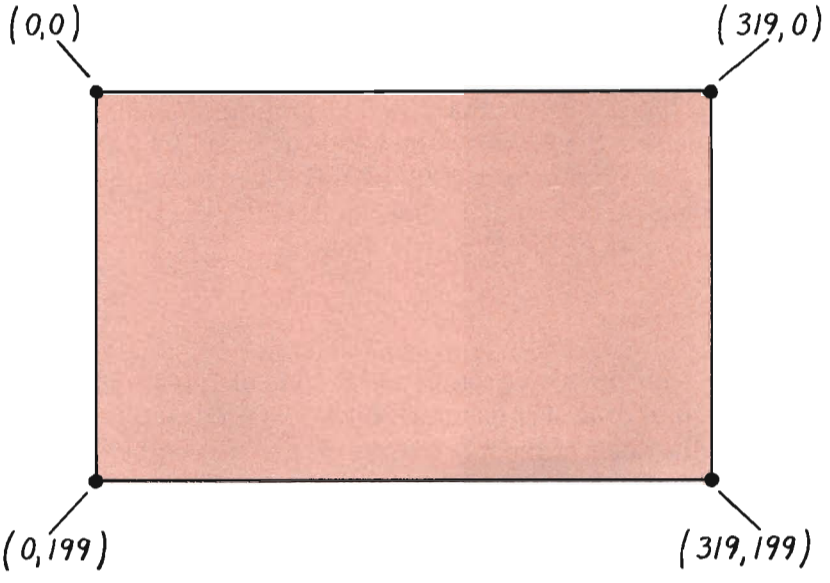
There are several statements that help you draw pictures on the standard graphics mode screen. There are statements for plotting dots and for drawing lines, boxes, circles, and polygons.

The coordinate system for the bit-mapped mode is shown in Figure 6-4. As you can see, the upper left corner is the home position (0,0) and the bottom right corner is the maximum value of the coordinate system (319,199). Each pixel on this display is indicated by an x and a y pair of "coordinates". For example, the pixel in the middle of the screen is at locations 160,100. Trying to plot a coordinate value outside the screen limits will cause the computer to stop and give a syntax error. The numbering of the axis (0 on top and maximum running toward the bottom) may appear to be upside-down to mathematicians (who expect positive numbers to grow upward), but in fact this is the way personal computers typically treat the screen.

Locating the Pixel Cursor: LOCATE

Before we show you how to draw on the bit-mapped display, you need to understand something called the pixel cursor, or PC. A pixel corresponds

Figure 6-4. Bit-Mapped Mode Coordinate System



to a dot location on the screen. The "pixel cursor" is similar to the flashing cursor you see in the Commodore 128 text mode that tells where the next character will be displayed. The pixel cursor indicates where the next dot will be placed, but there is no blinking dot to be seen. The PC is used differently in graphics. In BASIC statements where the optional coordinates are omitted, the pixel cursor is what is used as the default coordinates. For example, if we use the CIRCLE statement (we'll see it soon) without specifying the center coordinates for the circle, the pixel cursor is used as the default. How do we specify where the pixel cursor is located?

The LOCATE statement is used to move the pixel cursor to the desired coordinates on the screen. The syntax of LOCATE is:

```
LOCATE x-coordinate, y-coordinate
```

Here is an example of LOCATE that moves the PC to the center of the screen:

```
10 LOCATE 160,100
```

Note that you will see nothing when this statment is executed, but the next time you draw something and leave out a beginning coordinate, it will start at the PC location. This statement can be used to control where text is printed on the graphics screen, as we shall see when we learn how to use the CHAR statement. Now let's see how you actually draw on the graphics screen.

Drawing Lines and Points: DRAW

Now you know how to select the different graphic modes, set the color sources, and position the pixel cursor, let's learn how lines and points are easily drawn. The Commodore 128 BASIC 7.0 uses a format of line-drawing statement also found in the Apple II BASIC called the DRAW statement. The syntax of DRAW is as follows:

```
DRAW color source, a1, b1, TO a2, b2, . . .
```

In this description "color source" is a value of 1 to 3 representing the source on which to draw. For example, if color source equals one it would mean draw in the color of the foreground for the bit-mapped mode, a two would mean draw in the color of the foreground for the multicolor1 mode, a three in the multicolor2 mode. The values a1 and b1 represent the x-y

coordinates for the starting point of the line, and the values a2 and b2 are the x-y coordinates for the ending point for the line. Thus the statement:

```
100 DRAW 1,10,10 TO 50,50
```

will draw a line in the foreground color for the standard bit-mapped mode from the location 10,10 to the location 50,50 on the screen. The three periods in the syntax definition tell us that you can continue to specify coordinates to DRAW TO, and thereby create complex geometric figures. For example, the statements:

```
100 DRAW 1,100,100 TO 160,40 TO 220,100 TO 100,100: REM Triangle
110 DRAW 1,0,0 TO 0,199 TO 319,199 TO 319,0 TO 0,0: REM Border
```

would draw a triangle centered on the screen and then a borderline around the edges of the screen.

You can also use the DRAW statement to "plot" individual points by simply leaving out the TO and the ending coordinate specification. Think of this as a line with a length of one pixel. Thus the statements:

```
30  FOR X=1 to 319
40  Y=(X^2)/490: REM plot curve of Y=X**2
50  DRAW 1,X,Y: REM  note special draw with no TO
60  NEXT X
```

will produce a plot of the curving equation Y equals X squared. (Of course, you can use any equation you wish with the DRAW statement.)

Drawing Filled and Unfilled Rectangles: BOX

The BOX statement allows you to draw the outlines of rectangles of any width and height and any location on the bit-mapped screen. What is more, you can specify that the rectangle be filled with the current color source, so you get a solid filled box. To make the statement even more powerful, you can set the degree of rotation for the rectangle, causing it to spin about its center a precise number of degrees before it is displayed. Such a feature is useful for plotting in color coordinates, doing simulations, for turtle graphics and for games.

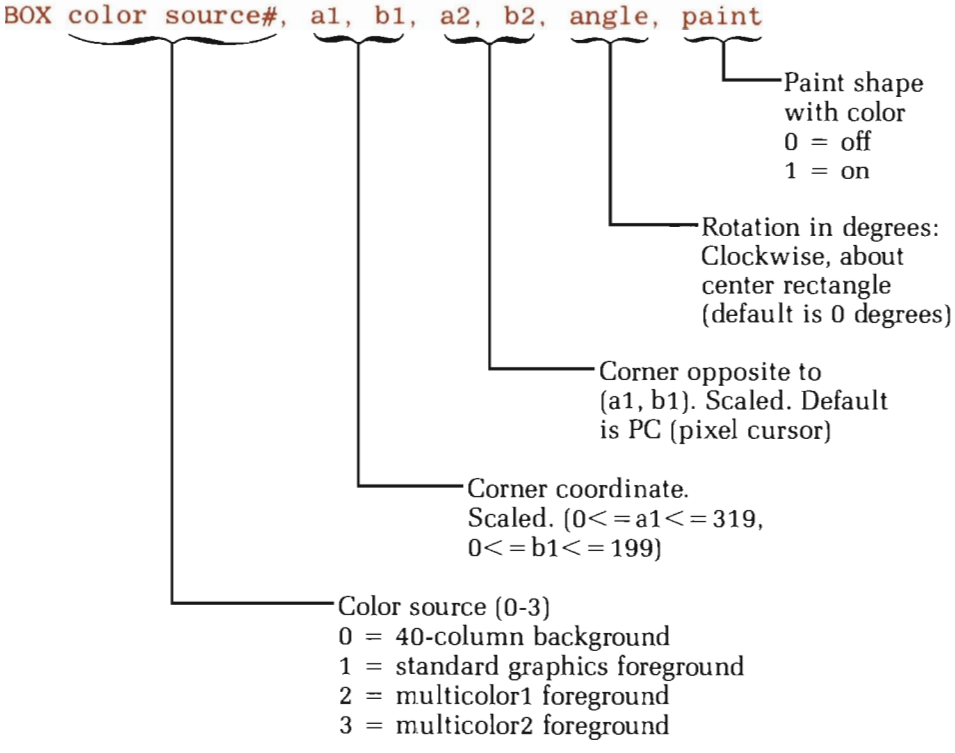
The syntax of the BOX statement is shown in Figure 6-5.

A typical BOX statement would look like this:

```
100 BOX 1,10,10,60,60
```

The first parameter, color source #, is used to specify what screen to use for the rectangle, and can be 0 to 3. Color source is normally the standard bit mapped screen (color source# = 1) and is the default. It may also be the multicolor1 mode (2), multicolor2 mode (3), or the 40-column text mode (0). The next parameters, a1 and b1, are the coordinates of the upper left corner of the rectangle. The parameters a2 and b2 are the coordinates of the lower right corner of the rectangle. In the above statement a rectangle is drawn from the coordinate 10,10 to the coordinate 60,60. The box is drawn on the standard bit mapped screen since color source number is

Figure 6-5. The BOX Statement Syntax



1. You can leave out a parameter by typing the comma that normally follows it. For example:

```
100 BOX ,10,10,16,60
```

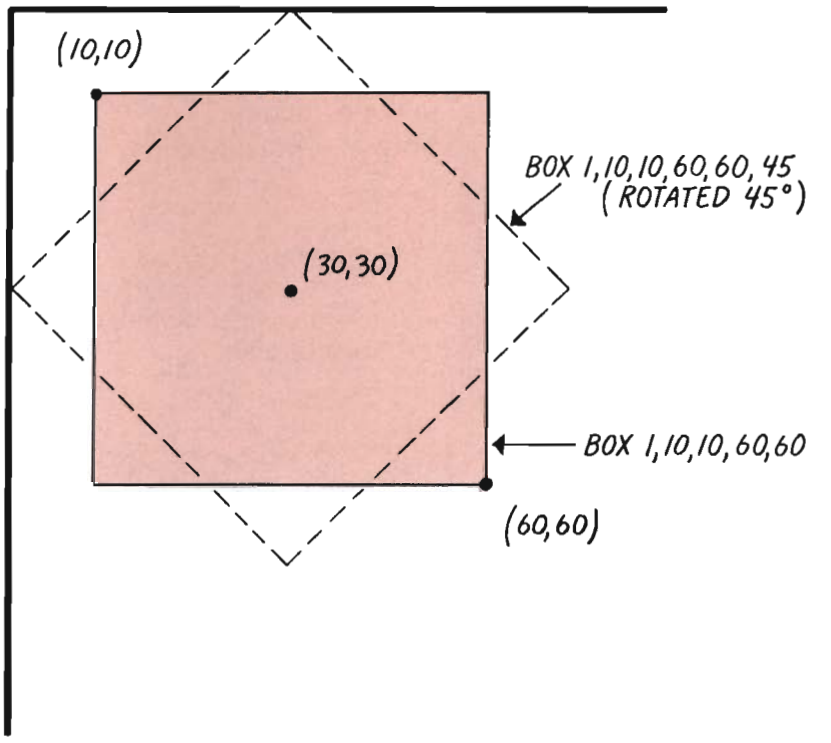
uses the default, or last value set for the color source #, which is not included in the statement above but is represented by a comma.

The parameter following the opposite corner coordinates is called angle, and controls the number of degrees that the rectangle is rotated before it is displayed. For example, the statement:

```
110 BOX ,10,10,60,60,45,1
```

will draw a box on the screen, centered about 30,30, and rotated 45 degrees as shown in Figure 6-6.

Figure 6-6. BOX Using the "Angle" Parameter of 45 Degrees



The final parameter in the BOX statement controls the color filling the rectangle. If the last parameter is a 1, then the box will be filled, while if the parameter is not included, or is 0, the box will not be filled and only its outline will appear. Painting is done in the same color as the current color source foregound color. Thus, if you want a box outlined in a different color you must first draw a filled box in the color you want, then change the color to the outline color, and finally draw a non-filled rectangle over the filled one.

Circles, Ovals, Arcs, Triangles and Polygons: CIRCLE

We've mentioned that you can draw circles on the screen before, but we didn't tell you that the CIRCLE statement is provided to draw all kinds of regular shapes on the screen, from circles to ovals, arcs and many-sided figures like octagons. The syntax for CIRCLE is shown in Figure 6-7.

The first parameter sets the color source number and the second two parameters control the center of the circle. If they are omitted the default is the pixel cursor. The second two parameters are the x-radius and the y-radius. Although it might seem strange to have two different radius parameters, two are included so that we can draw ellipses. The character of an ellipse is that it has two radius values. If you omit the y-radius it defaults to the value of the x-radius. However, this will not give a perfect circle because the X and Y axes are scaled differently.

In the following example, an ellipse is drawn centered on the coordinates 160,100.

```
10 CIRCLE ,160,100,65,10
```

To draw a perfect circle, the y-radius must be adjusted to compensate for the scaling. This statement will draw a circle:

```
20 CIRCLE ,160,100,65,50
```

Here we see that the y-radius (50) is made smaller than the x-radius (65) by a ratio of about .77.

The parameters after the radii are used to draw some portion of the circumference of the circle. The parameter sa is the starting angle for the circumference and the parameter ea is the ending angle for the circumference. Thus the statement:

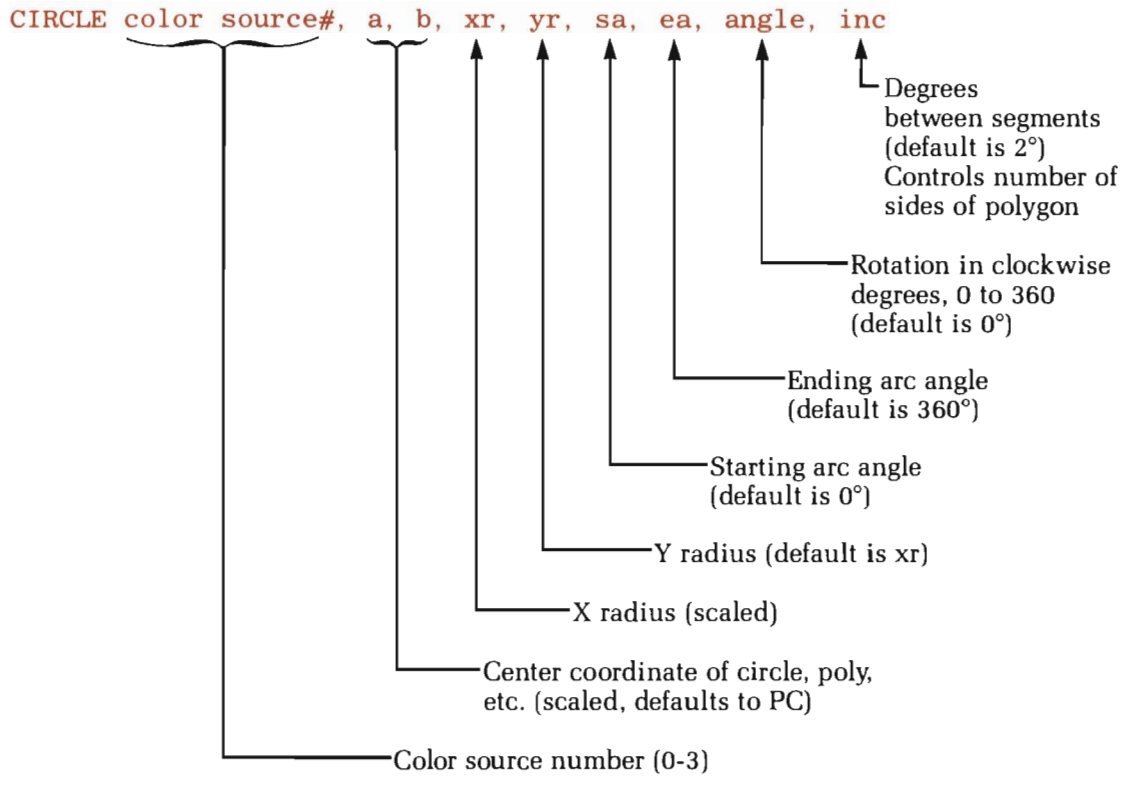
```
100 CIRCLE ,160,100,65,50,90,270
```

will draw an arc starting at the 90 degree mark (3 o'clock) and ending at the 270 degree mark (9 o'clock). This will look like a smile.

Figure 6-8 shows an example of the meaning of the parameters of the CIRCLE statement.

The second-to-last parameter in the CIRCLE statement is called angle. As in the BOX statement, it sets the rotation for the shape drawn with CIRCLE. The angle of rotation is specified in degrees clockwise and the rotation center is the center of the circle (a,b). The rotation parameter allows us to rotate an ellipse or an arc easily. The last parameter in the CIRCLE statement is the most interesting one. The parameter inc sets increment in degrees between "segments" that will be drawn when the parameter is used. This works because the circle is actually being drawn already in very small 2-degree segments. The segments connect along the

Figure 6-7. The CIRCLE Statement Syntax Draws Many Regular



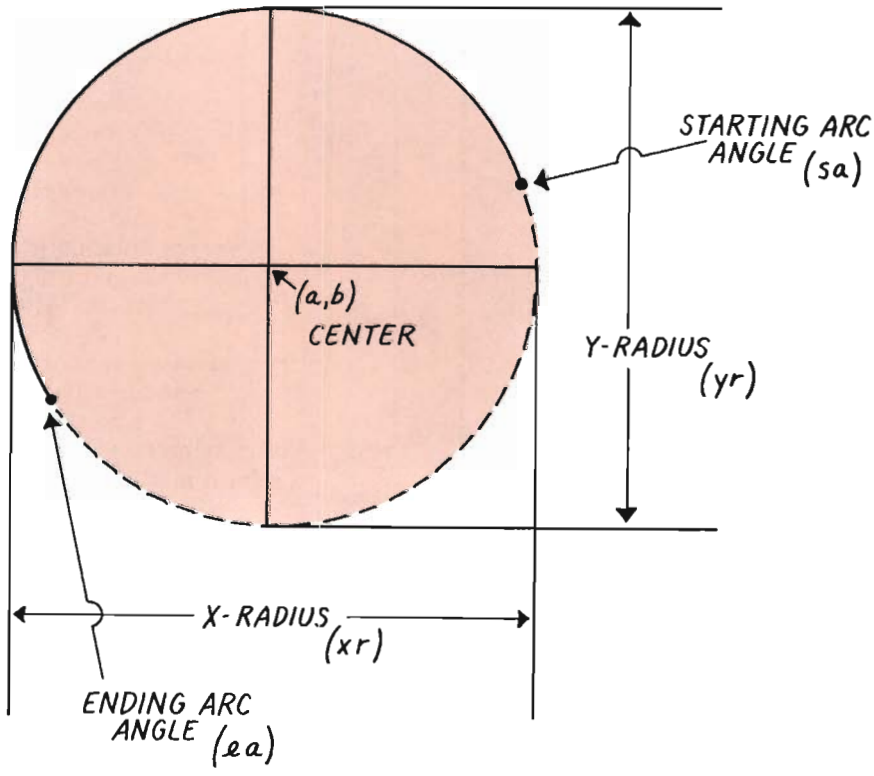
circumference of the circle, but since the segments are so small the circle looks like a circle. If we include the inc parameter and put an angle in it, such as 120, the CIRCLE statement will interpret this to mean the number of degrees between segments, and will figure out that there are three segments, hence a triangle. Here are some examples of using the inc parameter:

```
100 CIRCLE ,60,40,20,18,,,,45:  REM  draws an octagon (8 sides)
110 CIRCLE ,260,40,20,18,,,,90:    REM  draws a diamond (4 sides)
```

Filling and Painting: PAINT

You may have noticed that you could not specify a paint or not paint parameter in CIRCLE like you could in BOX. The statement PAINT allows you to fill any enclosed (completely outlined) shape with color. The color

Figure 6-8. Example of CIRCLE Parameters



filling the outline must be either the same color as the fill outline or any non-background color if you are in a multi-color mode where there is more then one foreground color. The syntax of the PAINT statement is:

```
PAINT  color source, a, b, mode
```

Here, color source is a number between 0 and 3 and represents the screen that you wish to paint on. The parameters a and b represent the starting coordinates for the filling, with the PC being the default if omitted. The mode parameter if 0 specifies an area defined by the color source, and if 1 specifies an area defined by any non-background source. If the location of a and b is already colored, the Commodore 128 will not paint the area enclosed by the boundry of your shape. You must pick a new starting point inside a shape not filled with a foreground color.

Here is an example of using PAINT. We draw a circle and then fill it with foreground color:

```
10 CIRCLE ,160,100,65,50:   REM draws a circle about 160,100
20 PAINT ,160,100:          REM fills circle with foreground color
```

Mixing Text with Graphics: CHAR

As we explained the different graphic modes for the C128, we noted that the two text modes (40-and 80-column) could only be mixed with the bit-mapped graphics modes by using a split screen. If you are in the bit-mapped mode and you type a character, it will be send to the text mode screen and you will not see it unless you have the screen split. But what if you want to add labels right next to figures drawn on the bit-mapped display screen, such as graphs, forms, and so on? Commodore 128 BASIC 7.0 provides the CHAR statement for drawing characters on the bit-mapped screen. The syntax of CHAR is:

```
CHAR color source, x, y, string, reverse flag
```

Here the color source is the screen mode you want to put the character on, x is the character column (0-39) and y is the character row (0-24). Note that CHAR does not start at a pixel coordinate, but rather a text row/column, which is much coarser. Therefore to (for instance) draw a title centered inside a circle, you will have to draw the title first, then draw the circle and adjust it pixel by pixel until it's right.

The string parameter is the text you wish displayed. The string will wrap around if it's too long to fit on a line. The reverse flag parameter

changes the character "field" to inverse if it's 1, and makes it "normal" if it's 0.

CHAR gets its text directly from the Commodore 128 ROM. In the graphics mode, CHAR does not offer all the treats that PRINT does in the text mode. You can't put control characters or graphics in the CHAR statement and see them appear on the graphics screen. But CHAR can be used in the text mode, and then will allow control and graphics character features to work.

Here is an example of drawing a circle and putting a label "Circle" inside it.

```
10 CIRCLE ,160,100,65,50:  REM draws a circle about 160,100
20 CHAR 1,17,12,"Circle":  REM puts a label in center
```

Finding Out About Pixels and Modes: RDOT and RCLR

If you use the Commodore 128 in its BASIC mode one of the things you will find yourself doing frequently is moving objects, and then trying to determine whether the object is on top of or touching another object or a particular backgound. Sometimes you need to know where the PC is and do not want to have your program try to keep track of it. The function for determining the position of the PC and the color source for that location is called RDOT, and its syntax is as follows:

```
RDOT (N)
```

Assuming the PC is set, if N is 0, the x position of the pixel cursor is returned, if N is 1, the y position of the pixel cursor is returned and if N is 2 the color source for the pixel is returned (i.e., a number representing the mode we are using).

The RCLR function is used to return the color assigned to the color source N, where N is between 0 and 6 as shown earlier in Table 6-3. The RCLR returns a number between 1 and 16 (listed in Table 6-2). For example, if you set the foreground color of the graphics mode to six, then when you used:

```
10 X=RCLR(2)
20 PRINT X
```

X will be equal to six. You use RCLR with RDOT to determine the exact color of a pixel for a particular color source. The RGR statement returns the current graphics mode, which is between 0 and 6.

Changing the Size of Images: SCALE

One command in the Commodore 128 vocabulary that may be useful allows you to alter the normal scale on the screen from its normal 320 x 200 to a second format of 1024 by 1024! Before you go crazy thinking you have access to every point on such a large screen, take heed: there are still only 320 x 200 pixels on the screen. But each one no longer represents a distance of just one. Now each horizontal dot is equivalent to 1024/320 or 3.2 units, and every vertical dot is equivalent to 5.12 dots.

The new, larger screen matrix created by SCALE is useful for adapting graphic programs which are already written for many computers. Dozens of high-end graphics terminals have screen limits of 1024 by 1024, and there are hundreds of professional programs written to these scales. With the C128 SCALE command it is possible to more easily adapt these program without having to recalculate all the coordinates.

Understand that if you have designed a program that draws objects just in the area defined by the coordinates 320 x 200, should you forget to adjust the coordinates and switch to the 1024 x 1024 mode, your program will do all its output to a small area in the upper left corner segment of the 1024 display.

This statement also has the effect of equalizing the coordinates for shapes drawn in all graphics modes. That is, the normal multicolor mode is 160 x 200, while standard graphics mode is 320 x 200. When SCALE is set to 1, these two modes both have the range 1024 x 1024 and respond in equal distances to the coordinates fed them.

Shape Saving and Drawing with Strings

Suppose you want to draw a detailed object on the screen, then move it around to different locations. Suppose your shape had been drawn using dozens of graphic statements, like DRAW, and BOX, and took a long time to draw. You could make the drawing statements a subroutine and call it with a new set of coordinates each time. But this would really be very slow. However, BASIC 7.0 has a feature that allows you to transfer a graphic shape from the screen to a string variable, and to transfer the string back to anywhere on the screen almost instantly.

Capturing the Image: SSHAPE

The syntax for the statment that captures a rectangular area of the bit-mapped screen into a BASIC string variable is:

```
SSHAPE string name, x1, y1, x2, y2
```

Here string name is the name of the BASIC string variable that will receive the shape data. The coordinates x1,y1 are the upper left corner of the rectangle surrounding the shape to be transferred and x2,y2 are the lower right corner of the rectagle. Thus to save the image of an object that is in the rectangle 10,10 to 33,31 into the string F$, you would execute the statement:

```
100 SSHAPE F$,10,10,33,31
```

Because BASIC string variables are limited to 255 characters, the size of the area that can be saved is limited. You can use these formulas to calculate the string size, given the two corners of the area to be saved are known.

For standard bit-map:

Length = INT((ABS(x1-x2) + 1)/8 + .99)*(ABS(y1-y2) + 1) + 4

For the muticolor mode the formula is:

Length = INT((ABS(x1-x2) + 1)/4 + .99)*(ABS(y1-y2) + 1) + 4

The shape that you are transferring is copied row by row into the string. The last four bytes of the string will contain the row and column lengths and are used by GSHAPE, as we shall see.

Drawing the Image: GSHAPE

The GSHAPE statement works in just the opposite way of the SSHAPE. It places the contents of the string variable onto the bit-mapped screen. The syntax of the statement is:

```
GSHAPE string, a, b, mode
```

In this statement the parameter "string" is the string variable containing the transferred screen data, and a,b are the coordinates of where we want the top left corner of the object to be drawn. If a,b is omitted the

current PC will be used as coordinate to draw from. Thus, assuming that our figure is stored in F$, here are some examples:

```
100 GSHAPE F$:           REM Draws the object at the current PC
110 GSHAPE F$,160,100:   REM Draws the object in F$ at 160,100
```

Graphics Modes

GSHAPE does much more then just rubber stamp a graphic image on the screen. The "mode" parameter controls how the pixels of the graphic shape mix with pixels already on the screen. There are five different values for mode and these are shown in Table 6-4.

Let's explore these modes in more detail.

Replace or Copy Mode

The AS IS mode completely overlays the shape on the backgound so that each bit in the background is replaced by the bits in the foreground. Therefore, regardless of what is drawn on the bit-mapped screen, when we draw with GSHAPE in the copy mode it will completely change every pixel that is under it.

The Inverted Mode

The INVERTED mode draws the shape just as in the copy AS IS mode, but it changes all the foreground colors to background and vice versa. This means that if you have a red character showing on a black screen, and you use INVERTED mode, the character will change to black on a red background.

Table 6-4. Graphics Modes for GSHAPE

Value	*Meaning*	
0	AS IS	Replace or copy mode
1	INVERTED	Invert the foreground and background colors
2	OR	Logical OR with background
3	AND	Logical AND with background
4	XOR	Logical XOR with background

The OR Mode

The OR mode "adds" the pixels of the shape image with the pixels of the background so that when either pixel (or both pixels) is in the foreground color, the result on the screen will also be in the foreground color. Only if both pixels are in the background color will the result on the screen be in the background color. This mode is referred to as "foreground wins" because it only takes one plane (image or bit-map) in the foreground color to get foregound color. This mode follows the logic shown in Table 6-5.

The AND Mode

The AND mode logically ANDs the shape and the screen data, effectively erasing the parts of the shapes that do not coincide. The rule is that only when both image and bit-mapped pixels are in the foreground will the result on the screen be in foreground. Any other combination gives the background color for result. The result is that only parts of the two shapes that are the same show up. Table 6-6 shows the logic for the AND mode.

Table 6-5. Logic for the OR Mode with GSHAPE

Shape	*Screen*	*Result on Screen*
foreground	foreground	foreground
foreground	background	foreground
background	foreground	foreground
background	background	background

Table 6-6. Logic for the AND Mode with GSHAPE

Shape	*Screen*	*Result on Screen*
foreground	foreground	foreground
foreground	background	background
background	foreground	background
background	background	background

The XOR mode

The XOR mode is very useful. It performs an exclusive OR of the shape pixels with the screen pixels so that only the coordinates that have different pixels are displayed! In other words, only pixels of the image and bit map that are different in color (one is in foreground and the other in background) will result in a foreground color pixel. Table 6-7 shows the logic for the XOR mode.

The XOR mode is used frequently when you want to draw an object over a background and you do not want to erase or change the background. The trick is to draw the shape with GSHAPE on the background in the XOR mode, then draw it again without changing the coordinates. The second time you draw the image, the original background will be completely restored. Then you move to a new location and repeat this process. Note that the other modes do not allow this "non-destructive" type of drawing.

The program below shows an example of SSHAPE and GSHAPE. We draw a circle on the screen, transfer it to a string variable with SSHAPE, clear the screen, print a line of text, then move the object across the screen with GSHAPE in the XOR mode. The text is not erased.

```
 90  GRAPHIC 2,1
100  CIRCLE 1,10,8,10,8:       REM draws circle at 10,8
110  SSHAPE A$,0,0,20,16:      REM gets circle into string
115  SCNCLR
120  PRINT CHAR 1,0,0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
130  FOR X=0 TO 295 STEP 2:    REM loops the x coordinate
140  GSHAPE A$,X,0,4:          REM draws the shape using XOR
150  FOR D=1 TO 50: NEXT D:    REM delays
160  GSHAPE A$,X,0,4:          REM erase the shape using XOR
170  NEXT X:                   REM till done
```

Table 6-7. Logic for the XOR Mode with GSHAPE

Shape	*Screen*	*Result on Screen*
foreground	foreground	background
foreground	background	foreground
background	foreground	foreground
background	background	background

Sprite Graphics

We just learned that moving graphic objects on the bit-mapped display requires close attention to how the object's pixels combine with those of the background. We must use the XOR mode to avoid changing or erasing the background pixels. If you spent more time writing programs that move and draw objects, you would quickly discover another problem: not enough speed. As the size of the object you are animating grows in pixels, it takes more time to draw it on the screen. This slows down the maximum speed of the object because we must wait longer for it to be drawn. Further, every single position of the object must be laboriously calculated by your program: this involves looping, incrementing a pair of position coordinates, testing when they are at the desired spot, and so on. Further, imagine trying to keep track of several graphic objects on the screen in a program at the same time. How would you know, for example, when one object struck another — or if it collided with the edge of the playfield? The truth is that this would require extremely complicated BASIC code, and that it would be so slow as to be ineffective for animation, simulations or today's arcade quality games.

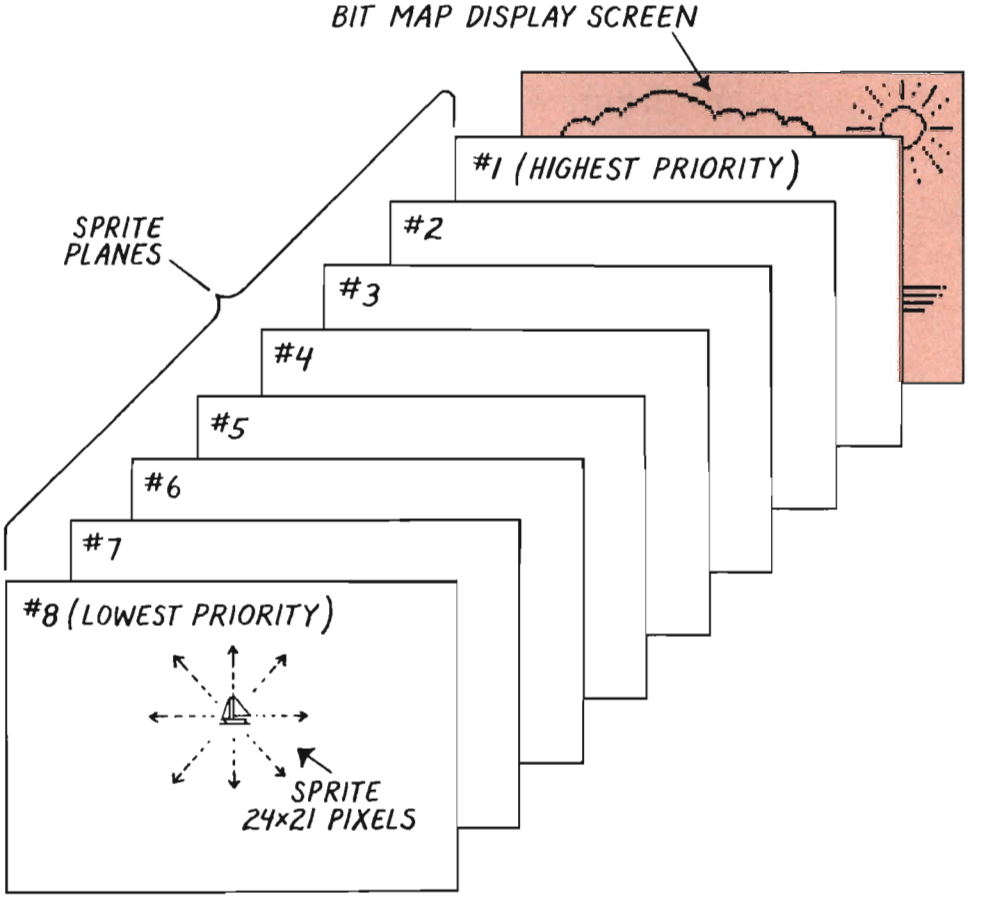
The Commodore 128 and C64 both provide a way to move and animate objects through a software mechanism called a "sprite" (much harder to control from C64 BASIC, however). Sprites are not soft drinks but rather software-"intelligent" graphic objects that you can program from BASIC 7.0 (and C64 BASIC 2.0 if you are willing to use POKEs and PEEKs). A sprite is a rectangular area 24 pixels wide by 21 pixels high. There are eight such sprites in the Commodore 128. You draw an image in the sprite with a special sprite editor built into BASIC (or you can use DATA statements and POKEs). There are two major points that make sprites extremely important and powerful programming tools:

1. Each sprite exists and moves on its own independent sprite screen "plane". Each sprite has its own color and shape. The sprite planes are totally independent of each other, and totally independent of the information in the bit-map screen (see Figure 6-9). When you request a sprite to move across the screen, it does so in a manner that does not erase or affect in any way the pixels of the standard bit-mapped background; nor does it affect the pixels in another sprite plane. This wonderful feature means your program does not have to concern itself with keeping track of what sprite overlapped with what sprite or what background. If you did, your program would be much more complex.
2. Sprites can be automatically moved by the VIC II chip to any position on the screen — you just give the sprite the final coordinates and it

will move smoothly from its original position to the resting position you desire. Or you can give the sprite a "heading" (an angle between 0 and 360 degrees) and a speed (between 0 and 15), and it will automatically start moving as specified — wrapping around when it hits the screen boundary, and moving forever until you command it to stop. This type of control of objects results in very powerful animation with very little demand on the programmer, and is the reason such good graphics software exists for the C64.

Sprites can also be combined with each other to create large and colorful graphic images. Connecting four sprites together gives an object

Figure 6-9. Sprites Exist on Independent Sprite Planes



that is 96 pixels wide by 84 pixels tall, which is quite large: 30% to 40% the size of the entire screen.

Each sprite can be given a display "priority" that makes the sprite move either behind or in front of your background images. This feature allows three dimensional effects in programs, where, for example, a truck can be drawn moving behind a row of trees and in front of a row of houses.

Your program can easily determine when sprites collide with each other or with the background. Such a collision will cause an "interrupt" to your BASIC program, and then you can branch to a special routine for processing the collision. This means it is easy to write a game where there are lots of moving players, or where there are sections of objects exploding. There is no need to constantly check the coordinates of the graphic objects as you would in non-sprite driven programs.

Sprites can also be expanded in the x or y directions to give the effect of rapid growth. When this happens, the pixels double in size. You can also define a sprite as a standard bit-mapped mode sprite or a multicolor mode sprite. Let's learn how sprites are created and used.

How Sprites are Created

There are three methods for creating sprites:

1. Drawing and defining the object with built-in statements. In this approach you draw your desired sprite object using the various graphics statements, such as DRAW, CIRCLE, BOX and PAINT. Then you use SSHAPE statement to get the picture into a string. Next, you transfer the string picture to the sprite with a SPRSAV statement. Then you enable and color the sprite with the SPRITE statement and move it around with the MOVSPR move sprite statement. You save your sprite to disk with the BSAVE filename command. (By the way, BSAVE lets you specify a range of RAM memory and have the binary values in it stored directly to a file on the disk. BLOAD gets the binary file and loads it exactly back into RAM.)
2. Using the SPRDEF Sprite Definition mode (sprite editor or designer). In this approach you use the built-in Commodore 128 sprite editor to draw your sprite right on the screen. You can control how the picture looks in a magnified format by moving a cursor and turning different parts of the grid on or off. When completed, the sprite is enabled and moved with the SPRSAV and MOVSPR statements.
3. Using the same method as the Commodore 64. In this approach, you had to draw your sprite on a 24 x 21 grid, convert the pattern to a

series of numbers and then POKE these numbers into the special sprite area of memory. We won't cover this method.

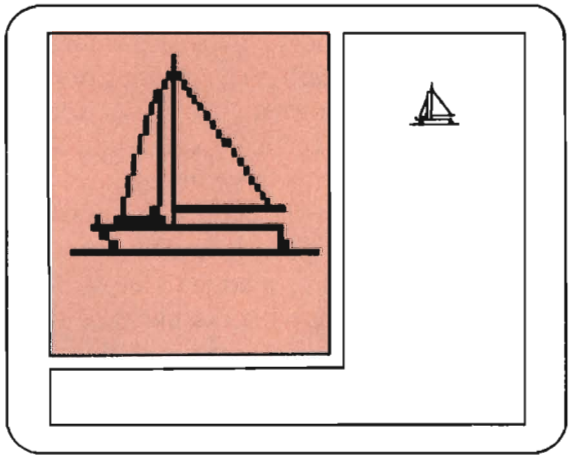
Both methods 1 and 2 are useful. Let's first describe how to use the sprite editor, because it takes such a small amount of program code and typing to get a sprite moving on the screen.

Sprite Designer Mode: SPRDEF

A sprite designer is built into the Commodore 128, which makes it easy to design sprite images. The sprite designer is modeled after sprite editors available for the C64. To enter the sprite designer mode from BASIC 7.0, type the command SPRDEF. The screen immediately clears and then displays a sprite grid in the upper left portion of the screen, as shown in Figure 6-10. At the bottom of the screen you will see the prompt SPRITE NUMBER ?.

When you enter this mode the normal execution of BASIC 7.0 is suspended. The keyboard enters a new mode where the keys are redefined to control the making of sprites. When you type in a sprite number between 1 and 8, the sprite grid (work area) is filled with the magnified sprite while the sprite itself is displayed in the upper right corner of the screen. The work area has the dimensions of 24 characters wide by 21 characters high so that each character position corresponds to one pixel in the sprite. While you are in the sprite designer mode, you use the cursor keys to move a special + cursor in the work area. As you fill in a character

Figure 6-10. Sprite Definition Screen



position within the work area, you can see the corresponding pixel in the displayed sprite turn on.

A sprite may be defined as a standard bit-mapped sprite or a multicolor bit mode sprite. You can select any of the eight foreground colors for the sprite using the keyboard (this is for convenience only — the final color for a sprite is set with the SPRITE statement). You can also double the size of the sprite horizontally or vertically or save the sprite by holding down Shift while pressing Return. Further details of editor use are discussed in the documentation accompanying the Commodore 128.

Turning On and Defining Sprites: SPRITE

Once you leave the sprite designer, any sprites you defined are stored in a sprite storage area in memory. However, the sprites have no color at this point, and are not "on", or visible on the screen. The SPRITE statement is used to turn the sprites on and off and to give them a foreground color. The syntax for the SPRITE statement is shown in Figure 6-11.

```
SPRITE number, on/off, fgnd, priority, x-exp, y-exp, mode
```

In this statement the parameter "number" is the sprite number 1 through 8 that you wish to control. The "on/off" parameter determines whether the sprite is on or off. A 1 means it's on; a 0 means it's off. The parameter "fgnd" is the foreground color you want to set the sprite to, and can be a value between 1 and 16.

Priority of Sprites

The "priority" parameter controls whether the sprite is behind (1) or in front of (0) images in the bit-mapped display. With respect to each other, a sprite's priority is set by its number — a lower number sprite will appear "in front of" a higher number sprite. Thus, sprite number 1 has the highest priority and sprite number 8 the lowest. The x-exp and y-exp parameters control the sprite's horizontal and vertical expansion. When either parameter is a 1, the sprite becomes twice as large in the corresponding direction.

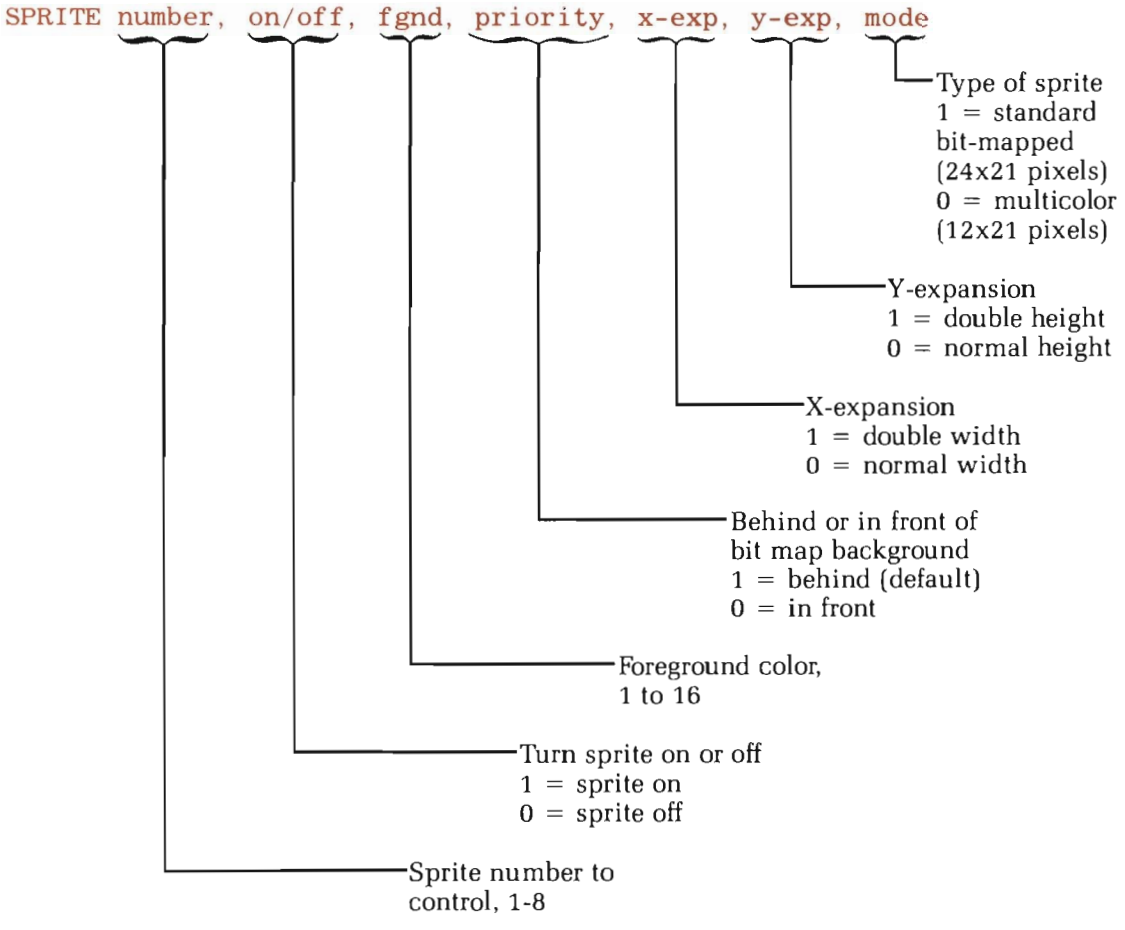
The mode parameter determines if the sprite will be a standard bit-mapped sprite or a multicolor sprite. A multicolor sprite occurs when this parameter is 1, and that sprite changes size to 12 dots wide by 21 dots high. The horizontal direction shrinks because the horizontal resolution of the multicolor mode is only 160 pixels. You can combine multicolor and standard bit-mapped sprites in any bit-mapped mode, but you can't display sprites in the text mode. The standard bit-mapped sprites

can have two colors: the background color set with the COLOR statement and the foreground color set with the SPRITE statement. Multicolor sprites can have two foreground colors and these are set with the SPRCOL statement.

Here are some examples of setting up sprites:

```
100 SPRITE 1,1,7:        REM  sprite 1, on, blue
110 SPRITE 2,1,5,1:      REM  sprite 2, on, purple, behind
120 SPRITE 8,0,8,,1,1    REM  sprite 8, off, yellow, in front, x-
                              exp, y-exp
130 SPRITE 8,1,7,,,,1    REM sprite 8, on, blue, frnt, no exp,
                             multicolor mode
```

Figure 6-11. The Syntax for the SPRITE Statement



The example in line 120 turns sprite number 8 off, changes its color to yellow, its priority defaults to in front, and the x and y sizes of the sprite are doubled. Note when a sprite parameter is not specified its default is normally 0.

Animating Sprites: MOVSPR

Now that we know how to define a sprite with the sprite designer and how to turn the sprite on and off, color it, and set its priority, let's see how to put it in motion. The MOVSPR statement is used to set a sprite's position, start it moving and stop it. There are three syntaxes for MOVSPR, as follows:

```
MOVSPR number, x, y:          REM absolute position
MOVSPR number, +/-x, +/-y:    REM relative position
MOVSPR number, x#y:           REM angle and speed
```

Here the parameter "number" is the sprite number you wish to move (1 to 8), while x and y are the coordinates of the new sprite location. When you put the values of x and y in the sprite statement, the sprite will move to the new location automatically. For example:

```
100 MOVSPR 1, 285, 178
```

will move sprite number 1 from its current position to the lower right corner of the screen.

If you use relative values for x and y, (i.e., you give them + or – signs, as shown in the second example) the sprite is moved relative to the last sprite position (not relative to the PC). For example:

```
110 MOVSPR 1, +50, -50
```

will move sprite number 1 from its last position to 50 pixels toward the right screen horizontally and 50 pixels towards the top of the screen vertically. Note this is not the same as the statement:

```
110 MOVSPR 1, 50, 50
```

which will move the sprite to the location 50, 50.

The third form of the MOVSPR statement allows you to set sprites into automatic motion and to stop them. In this "turtle mode" case we use the third syntax for MOVSPR, where x and y are separated by a #

sign. In this case the value of x specifies a clockwise angle in degrees for the direction the sprite will move between 0 and 360 degrees. The value for y represents a constant speed for the sprite, from 0 through 15. In this example:

```
100 MOVSPR 2,135#8:     REM sprite 2, 135 degrees, speed 8
```

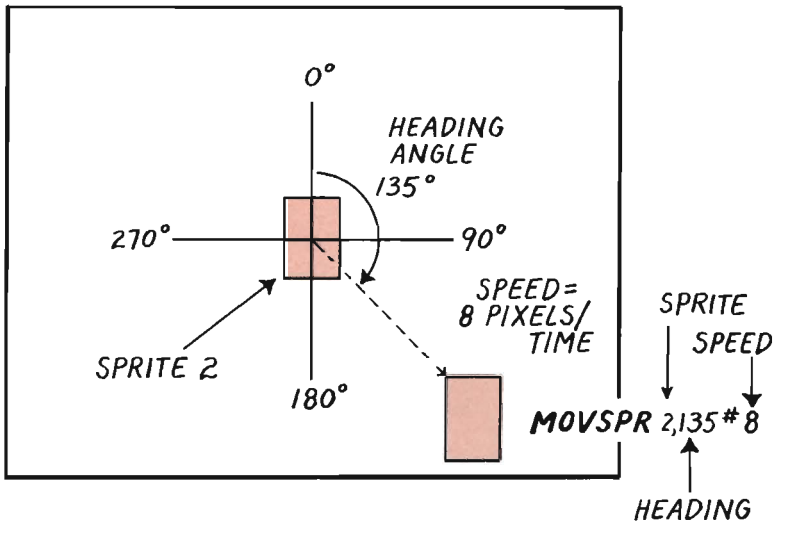
sprite number 2 will move on a 135 degree angle (down and to the right) at a relative speed of 8. This is shown in Figure 6-12.

What is wonderful about these sprites is that you can set one or all of them in motion and they will continue to move, wrapping around the screen and passing over each other, without any slowdown in the program. When a sprite is turned off with the SPRITE command, it remembers its last position. When it is turned on again with SPRITE, the object moves off in the last heading and speed, unless it is changed.

Handling Sprite Collisions: COLLISION and BUMP

Another very useful feature of Commodore BASIC and its sprites is that you can determine when sprites collide with each other or with the background. The COLLISION statement is used to allow your program to

Figure 6-12. Automatic Motion with MOVSPR



be signaled when sprites collide and the BUMP statement to determine which sprites collided.

Event Trapping: COLLISION

The COLLISION statement detects three different types of "events": collisions between sprites, collisions between sprites and the bit map images, and light pen activation. When one of these events occurs, BASIC will finish whatever statement it was executing, and then it will transfer control to the the first line in your collision-handling routine, as specified by the COLLISION statement. When your routine is complete, a RETURN statement must be executed. This causes the program to resume on the line following the one that was executing when the event occurred.

Thus your program is interrupted by a collision, and sent to a specific routine which controls what happens when there is a collision. This is called event "trapping". The event subroutine might reverse the direction of the sprite if it hits a wall on the screen edge, or it might make a sprite "blow up" and disappear if it hits another sprite, and so on. The COLLISION statement effectively "arms" the program to respond to such collisions.

The syntax of the COLLISION statement is:

```
COLLISION event, line number
```

Here the event parameter represents the type of event to be detected, as shown in Table 6-8.

The parameter line number is the line number you wish the program to branch to when the type interrupt occurs. When line number is not included in the statement, trapping is disabled. Understand that a collision occurs only when a non-background part of a sprite overlaps the non-background part of another sprite. Although the sprite (moving image) pixels are independent of the bit-mapped screen pixels, the VIC II chip can tell when the sprite image and bit-mapped screen contents are over-

Table 6-8. Meaning of Event Numbers in MOVSPR Statement

Value	*Event to Detect*
0	Sprite to sprite collision
1	Sprite to display data collision
2	Light pen activation

lapping. Thus a sprite-to-bit-mapped collision occurs whenever part of a sprite that is in the foreground color occupies the same location as a pixel in the bit-map that is in the foreground color. Sprites that are turned off will not cause collisions to occur.

Although you may have several types of event detection active at the same time, only one collision can be processed on the C128 at a time. Therefore the first thing that must be done by your event handling routine is to turn off further processing of events until it is finished. After you process the event and return to the main program, any event that occurred while you were in the previous process will be remembered and cause a new event interrupt. Hopefully your event handling routine will be brief so that any sprite collisions won't be missed.

Here is how a typical program that used COLLISION detection of sprites might look:

```
100  COLOR statement to set color
110  GRAPHIC statement to set mode

(BASIC statements to set up program, draw screen background, etc.)

500  COLLISION 1,1000:  REM start detecting sprite to sprite collisions
510  COLLISION 2,2000:  REM start detecting sprite to bkgnd collisions

520  (Loop here while waiting for sprites to collide with each
other or with the background)

1000  COLLISION 1: COLLISION 2:  REM  disable all sprite
collision detection
1005  (process sprite to sprite collision, reenable collision
detection)
1500  RETURN

2000  COLLISION 0: COLLISION 1: REM disable all sprite collision
detection
2005  (process sprite to background collision, reenable collision
detection)
2500  RETURN
```

Here you can see that after the program has set up its initial screens and other parameters, it "arms" the collision detection in lines 500 and 510. Next the program enters a loop where it can refresh the various things that the program is doing, moving any objects that are not sprites, playing music, reading the joystick, and so on. We call this the treadmill part of the program. Whenever a sprite hits another sprite or hits the background, the program will branch to the subroutine in lines 1000 or 2000 respec-

tively, process the collision and return. For example, a sprite colliding with an edge of a rectangle surrounding the screen can be "bounced" off it by performing a mathematical computation on the angle in the MOVSPR statement. Note that the light pen type of event will cause an interrupt if enabled.

What Sprites Collided: BUMP

Just knowing that a sprite collided with another sprite or with the background may not be enough. Suppose there is more than one sprite? In a sprite-to-sprite collision you need a way to tell what sprites collided. Or if there is a sprite-to-bit-mapped-image collision you need to know what sprite hit the image. The BUMP function is provided for this purpose.

After you get a collision event interrupt and have branched to the processing routine, the BUMP function is used to tell what sprites have collided with each other or the background. The syntax of BUMP is:

```
BUMP (event type)
```

Since bump is a function its result must be returned to a variable to be used in the program, such as R = BUMP(1). The parameter event type is used to tell BUMP what type of event you wish to know about. It corresponds exactly to the event types in the COLLISION statement, as listed in the previous table. BUMP returns a number between 0 and 255. The number is interpreted as an 8 bit binary value, with each bit position representing a sprite that has collided with another sprite or with the background. When that bit is on (1) there was a collision, when off (0) there was no collision for that sprite. When there are multiple collisions between sprites several bit positions will become active. In such a case you would use the RSPPOS function (described later). This function returns the current x-position, the y-position and the speed (1-15) if the sprite is moving.

Here is an example of BUMP:

```
1000 A=BUMP(0): B=BUMP(1)
```

This line reads the sprite to sprite collisions, puts the result in the variable A, reads the sprite to bit-map collisions, and puts the results in the variable B. The example shown below uses the logical AND operator to tell when sprite #2 has collided with the edge of the screen.

```
100  GOSUB 3000:  REM  draw screen boundary for sprite to hit
110  COLLISION 1,1000:REM enable sprite to sprite collision detect
```

```
120  MOVSPR 1,160,190,45#8:     REM start sprite1, move on angle
130  MOVSPR 2,160,190,135#8:   REM start sprite2, move on angle
200  GOTO 200

1000  REM  SPRITE COLLISION - DETERMINE WHAT ONE
1010  COLLISION 1:  REM DISABLE INTERRUPTS
1020  IF RBUMP(1) AND 2 THEN GOSUB 2000: REM SPRITE 2 HIT?
1030  COLLISION 1,1000: RETURN:  REM REARM COLLISION

2000  REM  GOT A SPRITE #2 COLLISION
2510  (Process sprite2 hit with screen edge)
2600  COLLISION 1,1000: RETURN:  REM REARM COLLISION
```

Where are Sprites: RSPPOS function

The RSPPOS function is used to determine the position and speed of a sprite. If you didn't have the RSPPOS function you would have to keep track of sprite position and speed with variables in your program, which is possible but complex. The syntax of RSPPOS is:

```
RSPPOS (sprite, data)
```

Here the sprite parameter is the sprite number you wish to get information about, while the data parameter determines what type of information is returned by RSPPOS, as shown in Table 6-9. A good use of RSPPOS is to determine which of many bit-mapped images you might have struck with a sprite. For example, in a game of Breakout you would make the ball a sprite and the many bricks bit-mapped images. Then each time there was a collision with the bit map you would use the RSPPOS function to return the x and y position of the sprite, and thus find out what brick was struck. Then you could erase the individual brick from the screen.

Table 6-9. Meaning of Data Parameter in RSPPOS Statement

Data	*What is Returned*
0	Current x-position of sprite
1	Current y-position of sprite
2	Current speed (0-15) of sprite

Special Notes about Sprites

You can make a sprite invisible by making its foreground color that same as your bit-map background color. The sprite will continue moving if it is in the automatic heading and speed mode but it will not be seen. There are many games where such a technique is useful. However, if collision detection is enabled, the sprite will still cause an interrupt with other sprites or the bit-mapped screen images. If the routine that processes sprites can be told not to bother with the invisible sprites, then this "clocking" technique will work.

Although there are only eight sprites, since you can use strings to hold images that can be transferred into sprites with SPRSAV, you can have many sprite definitions stored in strings and read them into sprites while the program is running. A game with eight race cars, for example, can easily be changed into a game with eight rocket ships.

The speed value in the MOVSPR statement (0 – 15) is used to control the actual number of dot positions to skip over in an equal time interval before drawing. Therefore, the movement is a series of instantaneous relocations. Thus, a sprite moving at a high enough speed or a sprite that has little foreground color in it (recall that it's only the foreground portion of sprites that interact) may miss colliding with a thin line or small shape, or another small sprite, by "jumping" over it.

Windows

Now that we understand bit-mapped graphics and sprites, let's take a look at one more modern feature of the Commodore 128; its windows.

Windows are text areas on the screen that can be independently controlled, either from BASIC or from the keyboard with the Esc key. You can change a window's dimensions (its width and height) and you can change its position. Once a window is set up all the text you send to the screen, everything you type (including input to INPUT or GET statements) and listings of your BASIC program appear within the window's boundaries and do not affect any other text on the screen. Statements for clearing the screen, moving the cursor, and so on, still work — but only in the area defined by the new window. The windows can be moved to any column or row. When its position is altered to a different row and column, the upper left corner of the window becomes the new home position (0,0).

What Are Windows Good For?

With windows you can organize the output of your program so that different areas of the screen are dedicated to specific functions. For example, in a Computer Aided Instruction type program, you might have an explanatory part of lessons restricted to one window, questions about the lesson output to a second window and the user input restricted to a third window. The nice part about this is that when you output or input to or from the different windows you do not have to worry about interfering with the other windows: everything is protected that is outside the current active window.

If you move the window from the old position, the information in it will not be changed unless you overlap it with a newly-positioned window. There is no automatic refresh of window contents in BASIC 7.0, so you will have to write your own routines to handle that.

The syntax for the window statement is:

```
WINDOW  top-left column, top left row, bottom-right column,
bottom-right row,
clear
```

Here the maximum and minimum values for the column and row depend on whether you are in the 40-column or 80-column mode. If the clear option is 1 the window is cleared after being set. For example, to set up a window that is 15 columns wide, 10 lines deep, starts at row 10 and column10, and clears the window, we would use the statement:

```
100 WINDOW 10, 10, 15, 20, 1
```

One advantage of using windows is that their contents can be manipulated with the Commodore [Esc] key. Commodore provides a powerful screen editor feature via the [Esc] key that can be exploited by your program. If you display any information on the screen in a window, you can allow the user of your program to delete lines, insert, scroll the contents of the window, change its size and position, and so on. This is not a word processing editor, but rather a screen editor; if you scroll up one line the text that moves outside the window is gone. It is also possible to control the screen from inside the program by sending the [Esc] key sequence in a PRINT statement to the screen. Some of the escape key functions are *mnemonic*; that is, the letter of the command represents the first letter of the function. For example, to insert a line in a window you type [Esc] I, to delete a line you type [Esc] d.

Get the Status of Current Window and Display: RWINDOW

Suppose the user with the [Esc] key functions resizes the window used to display program output information, and you want to send information to this window. Since its size has changed, the program would want to know the new parameters, so it could adjust its output accordingly (you would not want to output text in 40 columns if the window has been set to 20).

We can find out the size of a window with the RWINDOW function, whose syntax is:

```
RWINDOW(n)
```

where the parameter "n" represents the parameter you wish to know about. If n is set to 0 the number of lines is returned, while if n is set to 1 the number of rows is returned. If n is set to 2, the RWINDOW command returns either 40 or 80, depending on what current console device is in use. Suppose a window was 20 columns wide and you were printing text that was 40 columns wide. You could write your program so that it always first checked the width of the window with the RWINDOW(1) function. Then it used this number to control the number of words that were output to the window so that they always wrapped on at the end of a word before reaching 20 columns. To do this, the program's display output routines would have to check the length of each line and adjust it to break correctly.

You've come a long way in this chapter! Now that you've been introduced to some of the major aspects of graphics on the Commodore 128, let's move on and look at ways to generate sound effects and music.

7

Sound and Music

In this chapter you'll learn:

- **What kinds of sounds the Commodore 128 can produce**
- **The software and hardware available for generating music**
- **The principles behind Commodore 128 sound generation**
- **How to use the new BASIC 7.0 sound-producing commands**

The sound chip built into the Commodore 128 is probably the most sophisticated sound-producing device on any personal computer, regardless of price. Originally designed to generate the professional-quality sounds of video games, the sound chip is capable of truly amazing performance and versatility. It can reproduce the sound of almost any musical instrument, and an almost infinite variety of sound effects, including the human voice.

In this chapter we'll first introduce you to the general sound capabilities of the Commodore 128, and explore some of the existing software for music generation. Then we'll discuss in some detail how the Commodore 128 produces sound, and finally we'll show you how to achieve sound effects and music using BASIC 7.0.

What Can You Do with Sound on the Commodore 128?

The easiest way to take advantage of the Commodore 128's sound capability is to purchase commercial software. Mostly oriented toward music,

this software will typically let you turn your C128 into a sophisticated music synthesizer, rivaling professional machines costing thousands of dollars. Many musicians, composers, and rock groups are already using the Commodore 64 as a synthesizer, and the C128 has all the music power of the C64.

On the other hand, you may be interested in learning how to write your own sound-generation programs. If so, you'll find the Commodore 128, used in C128 mode, the ideal vehicle. The new BASIC 7.0 makes controlling the sound much easier than on previous Commodore computers, and the instant feedback provided by BASIC means that you can easily experiment with the various statements until you achieve just the effect you like.

Of course, you can also write programs in C64 mode — using BASIC 2.0 — that generate sound. However, you will be at a considerable disadvantage, since the Commodore 64 (and the C64 mode on the Commodore 128) lack the BASIC commands which make programming sound in the C128 mode so easy. Instead, you must resort to using BASIC's POKE instruction to insert numbers into certain memory locations (which are in turn connected to the SID chip). Having to know what numbers to POKE in, and what memory locations to POKE them into, makes programming sound effects considerably more difficult and inflexible in C64 mode than in C128 mode.

In the Chips with SID

The Commodore 128 has a special custom-made chip, called SID (for Sound Interface Device), which handles all its sound generation tasks. This chip can produce three separate channels, (called *voices*) of sound at the same time. Thus the C128 can easily play music with three-part harmony. The individual character of each of these three voices can be separately modified, so that one voice can sound like a trumpet, another like a guitar, and a third like a piano. This is achieved by modifying the *waveform* and the *envelope* of the sound. We'll describe what this means later in the chapter. By changing the envelope and waveform any of the three voices can also be made to produce a rich variety of sound effects, from machine guns and bombs falling, to the rustle of leaves on an autumn afternoon.

Once you know how to use BASIC to create sounds, you can add sound effects or music to any program you write. Most people think about games when they think about sound effects, and certainly the Commodore is a perfect environment for creating games with great graphics and astonishing sound effects. However, sound is also useful in programs besides

games, such as word-processor and spreadsheet programs. For instance, keystrokes can be signalled by a small clicking sound, so you're sure you've pressed the key, and incorrect input can be signalled by beeps. The sound-generating abilities of the Commodore 128 can even be used in electronics. For example, the C128 could generate a variety of waveforms, including white noise, to test audio and TV equipment.

What Equipment Do You Need to Use Sound?

Although the Commodore 128 has the SID sound generator chip built into it, the output of this chip needs to be amplified by an external device before it is audible, since there is no amplifier or speaker built into the C128.

Often, the TV set or monitor you're using as a display has an amplifier and speaker built in. The new Commodore 1902 color monitor, for example, has these features, as well as a volume control. By connecting the audio output from the C128 (either from the TV outlet or the Direct Video connection) to your display device, you'll be able to hear the sounds the C128 is making. (See the diagram in the chapter on peripherals for where these connections are.)

You can also run this audio output to your stereo amplifier (although the sound will not be in stereo). This is useful if you're using a monitor without audio capability. With the volume turned up and the right program, this can generate awe-inspiring effects: you (and your neighbors too, if you're not careful) will swear that an intergalactic battle cruiser is passing overhead. Even at low volume levels your stereo amplifier will produce better sound, and will do a better job of reproducing the very high and very low tones of which the Commodore 128 is capable.

Commercial Software and Hardware for Sound Generation

There is a wide variety of music software and add-on hardware devices, such as keyboards, already available for the Commodore 64. All of these products will work with the Commodore 128 in C64 mode, and some may also work in C128 mode. Manufacturers are also beginning to produce products specifically for the C128 mode.

In this section we'll describe, in general terms, what some of these products do. We won't describe specific products; rather we'll try to give you an idea of what sort of features are available and what to look for

when buying music-oriented programs and equipment for your Commodore 128. At the end of the chapter we include a list of companies producing music software and hardware.

Add-on Piano Keyboards

It's possible to use the regular keyboard keys on the Commodore 128 for playing and composing music. Some available software uses this approach. Sometimes only one row of keys is used, accessing a single octave; other products use all four rows of keys, assigning each row to a different octave.

However, for anyone used to playing the piano or a similar keyboard instrument, using typewriter type keys to play music is difficult. To solve this problem, a variety of manufacturers offer piano-style keyboards which can be added to the C128. Which one of these is right for you depends on how serious you are about music on the Commodore 128, and how deep your pocketbook is.

The simplest and least expensive (less than $50) of these keyboards fits right over the C128 keyboard. It's called an *overlay* keyboard. It looks like a small piano keyboard, but pressing each note causes one of the character keys on the keyboard to be depressed. Special software interprets these key presses and translates them into musical notes. This type of keyboard is not as easy to learn to play as a real piano or synthesizer keyboard, and you probably won't be able to play as fast on it. However, practice makes perfect, and there are virtuosos of even this humble approach to keying in music.

The next step up in sophistication are membrane-type keyboards which plug into the controller (joystick) ports of the C128. These are somewhat more convenient to use than the overlay keyboard, but still do not have the same action or feel as a synthesizer keyboard. Cost is between $50 and $100.

If you don't mind spending a little more — from $150 to $300 — you can buy a full-featured add-on keyboard that rivals those on commercial-grade synthesizers. A variety of firms makes these keyboards, which typically feature several octaves of notes.

Music Software

Music software available for the Commodore 128 serves a variety of functions. We'll talk about each of these functions in turn, although some software integrates some or all of the functions together in one product.

Other manufacturers offer specific products aimed at specific functions, but at lower prices than the integrated products.

As a Simple Musical Instrument

First, you can simply use the Commodore 128 as a musical instrument. Either using the regular Commodore 128 keyboard, or using an add-on keyboard, you play a melody, and the C128 generates the sounds of the notes and sends them out to be amplified by your TV set, monitor, or stereo amplifier. In this mode the C128 is operating as a simple music synthesizer. You can typically play using a variety of different voices and instruments.

Some packages offer a rhythm section which plays along with you, giving you a choice of rock, waltz, samba, and so forth. You can change octaves, use vibrato, change the tempo, and mold the sound of the C128 to imitate a wide variety of instruments, all while you're playing. Some products let you play several notes at once, or play notes and chords at the same time. However, no more than three notes can be played at once, since the Commodore 128 has only three voices.

As a Playback Device

Many software manufacturers have recorded albums of music which you can play on your Commodore 128, somewhat as if it were a record or tape player. The music generally comes on disks, and must be used with a master program to interpret the disks and turn them into music. Almost every variety of music has been recorded for the C128: rock, jazz, classical, country, and so on.

You can not only listen to these albums, you can also use them as take-off points for your own experimentation. You can rewrite them, using the composing programs to be described below. You can eliminate one voice in a composition and play that part yourself. You can change the voices, instruments, and tempo, to see what new effects you can create.

Many of these playback programs show you the note being played on the screen, as it's being played. Typically these programs display the note on a musical staff. Other programs provide a more complex visual display to accompany the music. This can range from simple shapes and colors to complex animation: in effect you're watching a complete computer-generated video.

As Composition Devices

Probably the most exciting kind of music software for the Commodore 128 lets you compose your own music. Typically, as you write notes, they are

stored in memory where they can be retrieved and modified. There are a variety of different ways to do this, depending on the particular software package. With some products you type the notes in at the keyboard; with others you use the joystick to position notes on a musical staff shown on the screen. You can use sharps and flats, and notes of different duration such as half, quarter, and sixteenth notes. You can write in three different voices, and assign the sound of different instruments to each voice. When you've finished any part of the music, you can play it to see how it sounds. If you don't like it, you can edit it and replay it until you do.

Many programs that let you play or compose music also let you print out the musical score on your printer. If you're a composer, this is invaluable. As soon as you finish your masterpiece, you can print it out, then mail off copies to your agent or the major studios so you can start making your fortune. A typical printout will have all the professional features of printed music: treble and bass clefs, sharps and flats, different length notes, key signatures, and so forth.

Teaching Music

There are several programs on the market which teach you music, or teach you specifically about the music-generation capabilities of the SID chip. These can provide a far more interactive and exciting environment for learning about music than traditional classroom methods.

We've given you some idea of the capabilities of commercial sound-generation software. But suppose you want to write your own sound-generating programs? The next sections explore how sound works on the Commodore 128, and how you can use BASIC 7.0 to take control of the sound generation.

Sound Ideas

If you want to write your own programs to generate sounds, you'll need to know something about sound in general, and about how the SID chip "thinks" about sound. That's the purpose of this section. In the section following this one, we'll show you what BASIC statements to use to achieve the effects we describe here. If you're already an old hand at programming sound on the Commodore 64, and you just want to learn about the new statements for generating sound in BASIC 7.0, then you can skip this section and go on to the next one, called Sound and BASIC 7.0.

Simple musical sounds have three major characteristics: frequency, volume, and duration. Let's look at these in turn.

Frequency

One of the most important aspects of any sound is its *frequency*. The frequency specifies how high or how low a pitch the sound has: a flute is higher-pitched than a cello; that is, its sound has a higher frequency. Frequency is measured in *cycles per second*, or cps. Human hearing extends from about 40 cps to somewhat under 20,000 cps, depending on the particular person. The C128 can't generate this wide a range of tones directly: the highest it can generate is somewhere around 4,000 cps. However, the harmonics produced by the SID chip extend much higher. (We'll define *harmonics* in a moment.)

When you program the C128 to generate a sound, you'll give the SID chip a number to tell it what frequency you want to generate. This number is not the frequency itself, but a number related to it, as we'll see in the section on BASIC.

All sound consists of waves in the air. These waves can be thought of as looking something like waves in the ocean, with troughs and crests repeating themselves over and over. In a high-pitched (high-frequency) sound the crests are closer together; in a low-pitched (low-frequency) sound they are further apart.

We mentioned harmonics. What are they? Briefly, almost any musical note consists not only of the fundamental frequency of the note, but of a variety of other frequencies as well. The other frequencies are multiples of the original frequency, and are called harmonics. The first harmonic is twice the frequency of the note itself. The second harmonic is three times the original frequency, and so forth. The harmonics are not as loud as the original frequency. Which specific harmonics are loud, and which soft, determines the characteristics of the note: its timbre and voice. It is these characteristics which the SID chip alters to generate the sounds of different instruments.

Volume

Next to the frequency, the *volume* (sometimes called "loudness" or "amplitude") of a sound is probably its most obvious feature. If you think of sound as a series of waves, then the volume is the height of the waves. You can control the volume of all the sound output of the Commodore 128 from your program, in much the same way that you can turn down the volume on a radio with a knob. You can also regulate the volume of sounds individually. We'll have more to say about this later.

Duration

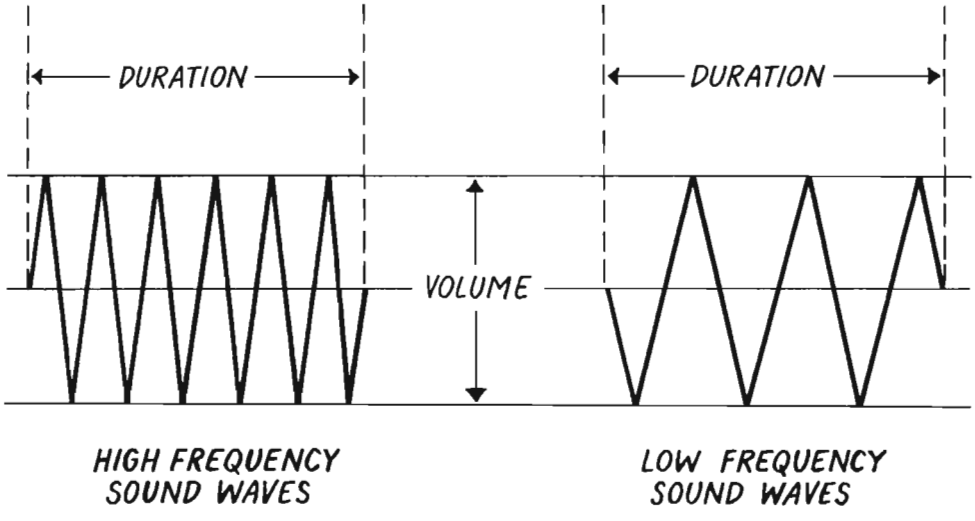
Any sound has a given *duration*, or length of time it lasts. The crack of a stick breaking, for instance, is shorter than the wail of a train whistle. When you use the Commodore 128 to generate sound, you'll give it a number to tell it how long a sound should last.

There are several aspects to specifying duration in music. First, there is tempo. The tempo specifies how long a *measure* is. A measure is a fundamental unit of music notation. There are a certain fixed number of notes in each measure. A note can be a *whole note*, a *half note*, which is half as long as a whole note, a *quarter note* (half as long as a half note), an *eighth note*, or a *sixteenth note*. The exact number of notes in a measure depends on the *time* of the music. For instance, if a musical piece is in 4/4 time, then there will be four quarter notes (or two half notes or one whole note) in a measure. If a piece is in 3/4 (waltz) time, then there will be three quarter notes in a measure.

So to establish how long a musical note will last, you need to specify both the tempo, and the kind of note it is: whether half, quarter, or whatever. We'll have more to say about musical notation when we discuss the PLAY statement in the next section.

Figure 7-1 shows the frequency, duration, and volume of some typical sound waves. If you connected a microphone to an oscilloscope (a device that makes pictures of electrical wave forms), and held the microphone

Figure 7-1. Frequency, Volume, and Duration of Sound Waves



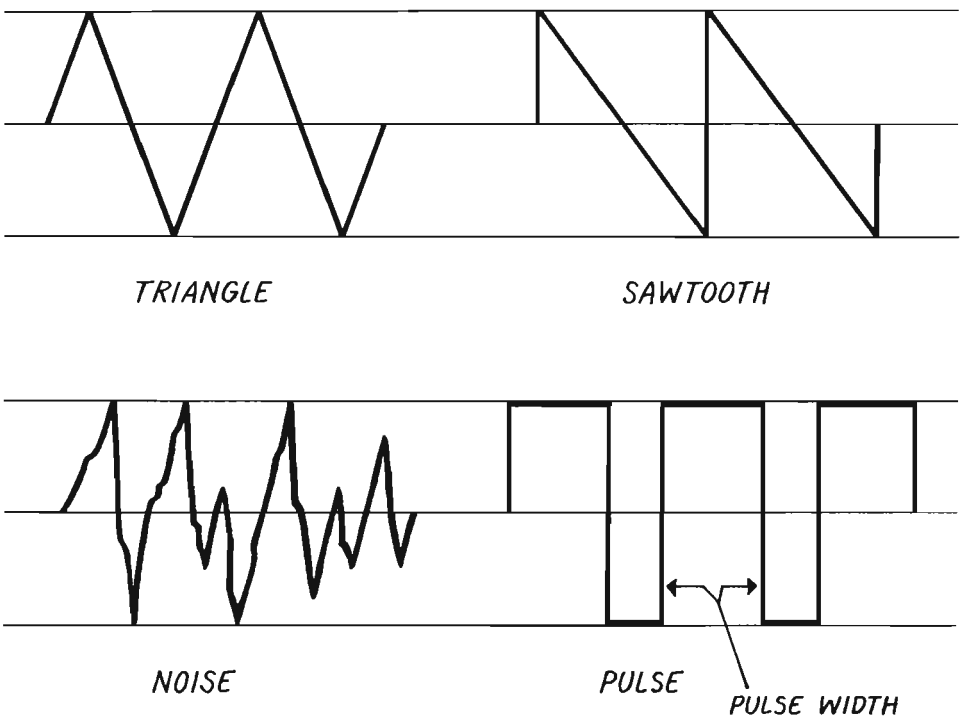
next to a source of sound, you would see waves shaped something like this. Usually, however, a note generated by a musical instrument is considerably more complicated than these simple shapes.

On many home computers what we have told you about frequency, volume, and duration is all you need to know about sound. These computers can make simple beeping sounds of different pitches, but that's all. They cannot perform the more sophisiticated sound operations needed to shape the sounds to accurately imitate different musical instruments and other effects.

Wave Forms

In Figure 7-1 we showed only one kind of waveform: the triangle. However, the SID chip can also generate several others: sawtooth, noise, and pulse. These four waveforms are shown in Figure 7-2.

Figure 7-2. The Four Different Wave Forms



Sawtooth and *triangle* are similar in that they produce simple melodic sounds. The triangle sounds are more mellow, like woodwinds; while the sawtooth is brassier. (These sounds are different because they have different harmonics.)

The *noise* waveform is used for an entirely different purpose. Rather than generating musical tones, it generates rumbling and hissing sounds. It's used for sound effects such as explosions, gunshots, and avalanches. It achieves these effects by using a very irregular waveform. This creates what's called "white" noise, a mixture of many different frequencies which sounds like static. Low frequencies produce a rumbling sound, while higher frequencies produce "shhh" and "ssss" sounds. By molding this white noise with the envelope (to be described next) or filters, you can create an amazing variety of effects.

The *pulse* wave generates tones which work best at reproducing certain kinds of instruments, such as pianos. This waveform requires you to specify the width of the pulses. Different pulse widths produce different effects.

The Envelope

When a note is played on a musical instrument — a violin, for example — it doesn't simply start suddenly, continue at a fixed volume for a period of time, and then stop suddenly, as Figure 7-1 implies. If you could look at a picture of the sound made by a real musical instrument on the oscilloscope, you'd see something different and more complicated happening. Imagine a note as having a lifetime of its own, and going through a series of phases in its life. A man is born, grows larger and stronger, loses some of his strength but hangs in there for many decades, and finally dies. A musical note goes through a similar process, but its phases are called attack, decay, sustain, and release.

When these four parts of the note are put together, the resulting waveform is called the *envelope* of the note. Let's look at the envelope's parts in more detail.

Attack

First, the sound of our violin note takes a certain length of time to build up to a peak volume. This length of time is called the *attack*, and can be thought of as the growth period of the note. The attack does not take a long time (it's measured in milliseconds, or thousandths of a second), but it does influence how the note will sound. Cymbals, for instance, have a shorter attack than a violin or flute.

Decay

Once the note has reached it's peak volume, it immediately begins to die away. You can think of this as a person beginning to lose maximum athletic ability soon after reaching full size — peak ability begins to decline almost as soon as it's been achieved.

Sustain

During its decay phase, a note will lose some of its strength, but not all. It will drop to a level somewhat lower than its peak. This level is called the *sustain* level. Note that sustain refers to a volume level, not to a length of time, as with attack and decay. The length of time that a note remains at the sustain level is determined by the duration of the note, as mentioned earlier.

Release

After a note has been sounded for a certain length of time (the duration), it comes to an end. However, it does not end instantly, but dies away more or less gradually. The length of time during which it dies away is called the *release* part of the note.

Putting It All Together

Figure 7-3 shows the envelope of a note with the attack, decay, sustain and release phases.

Notice that the envelope serves only to *shape* the existing sound waves. The sound waves and the envelope are different kinds of things. The sound waves have their own frequency and duration, which is not changed by the envelope. The envelope only specifies how the waves will grow and die away. Notice that in the diagrams of the sound waves we showed both the upper and lower parts of the waves; while in the diagram of the envelope only the upper parts of the soundwaves are shown. This is done only to simplify the diagram, the individual waves still oscillate both above and below the line, even though the oscillations below the line are not shown.

Not all instruments use all four phases of the envelope. Woodwinds (like clarinets and flutes) do, as do the strings (violins) and the brass instruments (trumpets and French horns). But percussion instruments may omit the sustain phase altogether. In the case of a piano (a percussion instrument), there is a decay phase, so that the envelope goes up sharply, slides down the decay slope, and then during release drops to zero; but with drums and cymbals the decay and release phases are merged together,

so that their envelope goes up sharply, and then returns immediately to zero. This is shown in Figure 7-4.

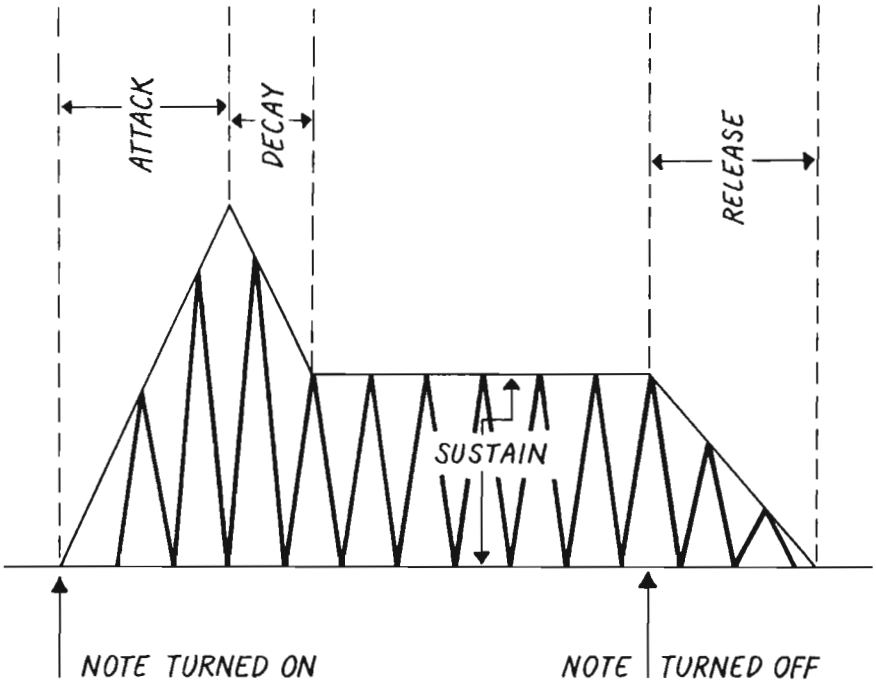
Actually, there is usually a variety of different envelope shapes which will generate the sound of a particular instrument. Often, for instance, you can eliminate either the decay or the release phase and achieve roughly the same effect.

In the next section of this chapter, on BASIC 7.0, you'll learn how to generate the sounds of particular instruments by designing "custom" envelopes; that is, selecting appropriate values for attack, decay, sustain, and release. Commodore has also built the envelopes for some instruments right into BASIC; for these instruments you don't need to design your own envelope.

Sweep

Another way that notes can be modified by the SID chip is by using sweep. Sweep lets you start a note at a certain frequency and then vary the

Figure 7-3. The Envelope of a Musical Note



frequency while the note is in progress. You need to tell SID whether you want the sweep to go up or down, what frequency you want to end on, and what frequency step value to use in sweeping (this determines how rapidly the sweep will take place).

Voices

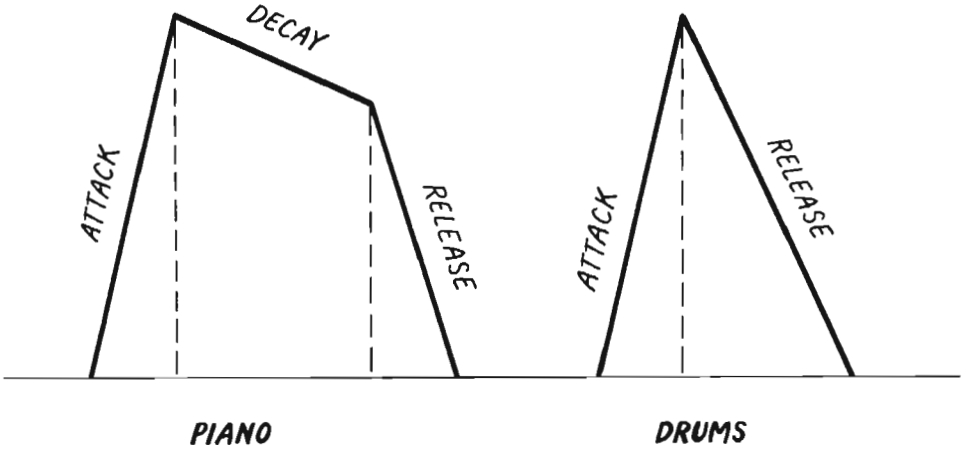
We've already mentioned that the SID chip is capable of producing three independent voices or channels of sound output. Each of these voices can be programmed independently, so the effect is almost like that of having three separate sound chips. Each voice can be given its own frequency, duration, and envelope. Notice, however, that the volume control on the SID chip operates on all three voices at once. You can specify the sustain volume for individual voices, but the overall volume affects them all the same way.

Filters

Besides the envelope, there are other more subtle ways to modify the sounds created by the SID chip. One way is through the use of *filters*.

A filter is an electronic circuit which blocks out certain frequencies of sound. The bass and treble tone controls of your stereo are filters. When you turn the treble control to its minimum setting, the higher-frequency parts of the sound are filtered out; and when you turn the bass knob to

Figure 7-4. Envelopes for Piano and Drums



minimum, the lower-frequency sounds are filtered out. You can use the filters in SID in very much the same way you use the tone controls on your stereo: to emphasize sounds of certain frequencies and diminish sounds of other frequencies.

High-and Low-pass Filters

There are three kinds of filters available in the SID: high-pass, low-pass, and band-pass. When you use the high-pass filter, you block out the low-frequency sounds, and pass the highs, just as you do when you turn down the bass on your stereo. This creates tinny, high-pitched sounds. The low-pass filter, on the other hand, blocks out the high frequencies, and creates deep, solid tones.

Cutoff Frequency

What frequencies, exactly, are we talking about when we discuss filtering out the low or the high frequencies? On your stereo you have no way of knowing exactly what frequencies are eliminated when you turn the knob. The SID chip, on the other hand, lets you specify exactly where you want the cutoff point to be. For instance, if you're using a high-pass filter and you specify a cutoff frequency of 1000 cps, then all the frequencies below 1000 cps will be filtered out or eliminated. Similarly, if you're using a low-pass filter with the same cutoff frequency, then all the sounds with a frequency higher than 1000 cps will be filtered out.

The electronic circuits that act as filters aren't perfect, so the cutoff frequencies are not exact. The frequencies on the "wrong" side of the cutoff will be transmitted to some extent, but the further a frequency is from the cutoff, the more it will be attenuated. Figure 7-5 shows the effect of filters on the sound levels at a range of frequencies.

Bandpass Filter

SID offers a third kind of filter called the *bandpass* filter. As its name implies, this circuit filters out both high and low frequencies, so that only those in the midrange can get through. In the case of this filter the term "cutoff" frequency is a misnomer, since the number used for this frequency now specifies the point at which the sound will *pass through* the filter, rather than being cut off.

Figure 7-5 shows the effects of these three kinds of filters on the amplitude of sound transmitted at different frequencies.

By combining the high-and low-pass filters you can create a filter which is the opposite of the band-pass filter: it filters out mid-range frequencies, rather than letting them pass through.

Another aspect of the filters that can be varied is the *resonance*. This determines how the frequencies of the sound near the cutoff frequency behave, and can be modified to achieve varying degrees of "sharpness" in the resulting sound.

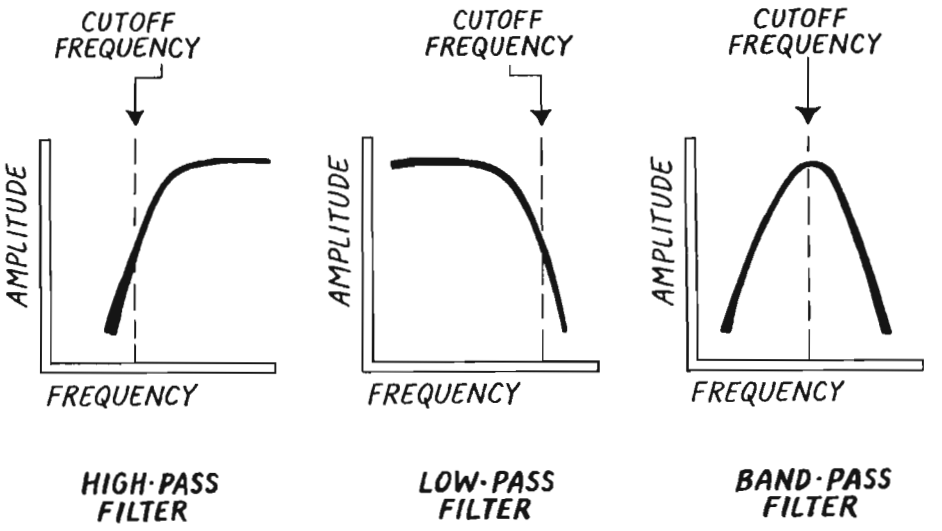
Something to note about the filters is that they operate on all three voices at once. Thus you can't use filters to emphasize the bass on one voice and the treble on another.

Ring Modulation and Synchronization

It is possible to combine two voices together to produce interesting musical effects. In *ring modulation*, the outputs of two voices are added together. This can produce the "ringing" effect heard in bells and gongs. In *synchronization* two voices are logically ANDed together. This can be used to produce a rising and falling sound, such as a mosquito might make, and other complex harmonic structures.

This review has only scratched the surface of sound generation on the Commodore 128. There are many other ways to shape and modify sound, but at this point let's turn to the specifics of sound generation with BASIC.

Figure 7-5. Filters



Sound and BASIC 7.0

In this section we'll describe the commands that BASIC 7.0 uses to control the sound and music capabilities of the SID chip. There are six of these statements: VOL, SOUND, ENVELOPE, PLAY, TEMPO, and FILTER. We'll describe each of these statements in turn.

The VOL Statement

As we mentioned above, the volume of all three voices can be controlled together by one BASIC statement. This statement is VOL, for volume. This is one of the simpler statements, since it has only one parameter. Here's a typical program line in BASIC, using this statement:

```
100  VOL 8
```

The possible settings for the volume range from 0 to 15, where 0 is off (no sound) and 15 is maximum.

Since you can adjust overall sound volume using the knob on your TV set or stereo, you may find yourself keeping the VOL at its highest level most of the time, and controlling individual notes with the sustain level. However VOL is useful in sound effects, and for turning the sound on and off. Notice however, that *you must use VOL before you can use the SOUND statement to actually generate sound.* You can't get any sound out of the SID chip until you've used VOL to select a non-zero volume level.

The SOUND Statement

The SOUND statement is the keystone of BASIC 7.0's sound effects capability. With it you can set the frequency, duration, sweep, and waveform of a sound. It's not usually used for playing music; the PLAY statement does that. SOUND is used for sound effects and other situations where you want very close control of the frequency and other characteristics of the sounds you're producing.

The Simple Version of SOUND

In its simplest form, SOUND can be used to set only the frequency and duration of a sound. You must also specify which voice is to be used, since every sound produced must be made by one of SID's three voices. For instance, the program line

```
100  SOUND 2, 4291, 60
```

will cause a tone with the frequency of middle C to be played by voice number 2 for one second. Lets look at the three parameters used in this statement.

The first parameter in SOUND (the 2 in this case) specifies the voice. The range of choices here is from 1 to 3.

The second parameter is the *frequency*. This is a number from 0 to 65535. Unfortunately, this number does not correspond to the actual frequency of the sound. However, it's related to it by the following simple formula:

Parameter value = Frequency / 0.06097

Thus while the range of values for the frequency parameter runs from 0 to 65535, the range of actual frequencies runs from 0 to about 4000 cps (although values below about 40 cps will sound more like a series of clicks than a tone).

Middle C on the musical scale has a frequency of 262 cps, so, dividing this number by 0.06097, we find the value of the parameter to generate this note is 4297. Table 7-1 shows the frequencies and parameter values

Table 7-1. Frequencies and Frequency-parameter Values for Musical Notes

Note	*Actual Frequency*	*Sound Frequency Parameters*
C	262	4297
C#	277	4543
D	294	4822
D#	311	5100
E	330	5412
F	349	5724
F#	370	6069
G	392	6429
G#	415	6807
A	440	7217
A#	466	7643
B	494	8102
C	523	8578

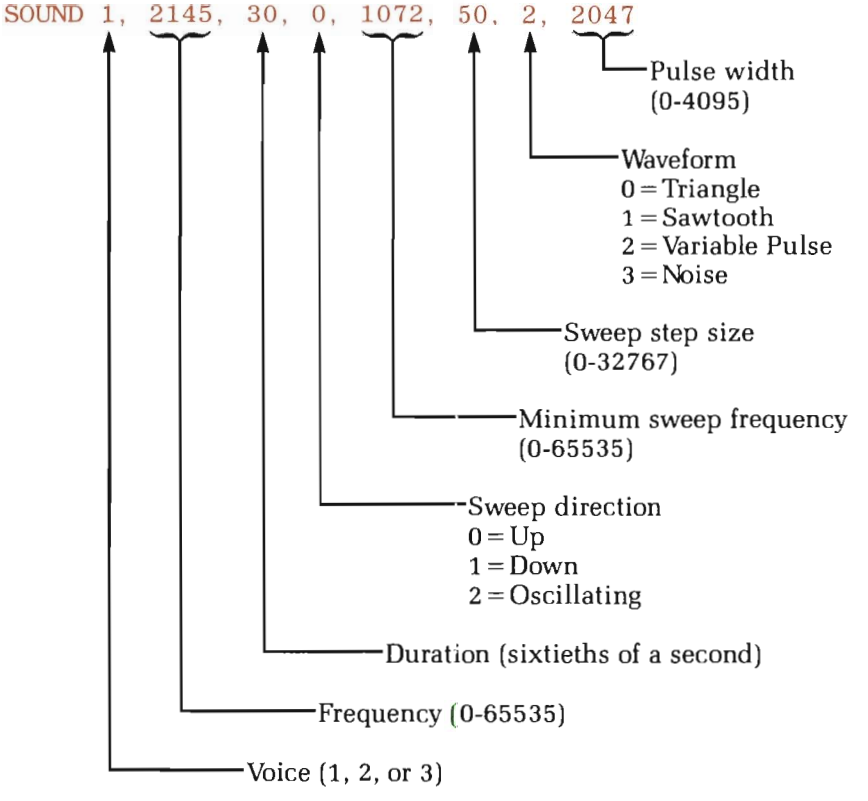
for the notes in the octave starting with middle C. The notes in the next higher octave can be found by doubling these values, and the notes in the octave below can be found by dividing them by 2.

The third parameter in the sound statement is the *duration*. This is the length of time the sound will last, expressed in units of 1/60 of a second. Thus if this parameter is set to 60, the sound will last for one second. A setting of 30 will give you 1/2 second, a setting of 120 will give you two seconds, and so on.

Setting the Sweep and Waveform

The SOUND statement can be used just as we've shown it above, with only three parameters. However, it can also be used to specify the *sweep* and the *waveform* of the sound. Figure 7-6 shows a SOUND statement with all the parameters identified.

Figure 7-6. The SOUND Statement



When the sweep parameters are used, the note will start at one frequency and move up or down to another frequency as it's being played. To cause this sweep to take place you must specify three parameters in addition to voice, frequency and duration. These new parameters are sweep direction, minimum sweep frequency, and sweep step size.

Sweep direction specifies whether you want the sound to sweep up in frequency while it's being played (a value of 0), to sweep down (a value of 1), or to oscillate between a high and low value (a value of 2).

The *minimum sweep frequency* specifies the lower of the two frequencies used for the sweep: the sound will either sweep down to this frequency from the value specified in the original frequency parameter (if you specified down as the direction), or it will start at the minimum frequency and sweep upward (if you specified up as the direction). If you specified the oscillating mode, the pitch of the sound will oscillate between the two values.

The *sweep step size* determines how rapidly the sweep will take place. Although this parameter can have a value as high as 32,767, it is often used with values in the hundreds or lower.

The following short program will cause the frequency to sweep down through one octave, to middle C, using a frequency step size of 100:

```
100 VOL 15
110 SOUND 2, 8578, 60, 1, 4291, 100, 0
```

For the sound of a police siren, try this program, which uses the oscillating sweep between the frequency parameters of 30000 and 40000, for 5 seconds:

```
100 VOL 15
110 SOUND 1, 40000, 300, 2, 30000, 1
```

You can also use the SOUND statement to select one of the four *waveforms* the SID chip is capable of generating: triangle, sawtooth, variable pulse, or white noise. If you select the variable pulse waveform, then you must also specify the *pulse width*. This parameter can run from 0 to 4095. When its value is half the maximum, or 2048, the wave shape is that of a square wave, and the sound produced has the "fullest" or most sonorous quality. Values close to this are often used to imitate the sound of a piano. Other values for the pulse width make a sharper, more buzzy sound.

This program will use the sawtooth waveform to generate the note A above middle C. Notice that since sweep is not used, the three parameters that pertain to sweep are all zero.

```
100 VOL 15
110 SOUND 1, 7216, 30, 0, 0, 0, 1, 0
```

The following program will make the sound something like an organ, using the variable pulse waveform and a square wave:

```
100 VOL 15
110 SOUND 1, 7216, 30, 0, 0, 0, 2, 2047
```

Here's a program that makes a short sound like a shot, using the white noise waveform. By combining a stream of these sounds with a FOR...NEXT loop, you could generate the sound of a machine gun.

```
100 VOL 15
110 SOUND 1, 1000, 10, 0, 0, 0, 3, 0
```

Note that the SOUND statement causes the sound to be produced as soon as it's executed. Once started, the sound will continue for the duration specified.

The ENVELOPE Statement

Now that we know how to generate sounds, we'll move on to music generation on the Commodore 128. The principal statement used to generate music is PLAY. However, to understand the PLAY statement, you'll first need to know how three other statements work: ENVELOPE, TEMPO, and FILTER. We'll cover these in order, then discuss PLAY.

ENVELOPE defines the characteristics of a particular envelope. After an envelope has been defined, you can then use PLAY to specify what particular notes are to be played using this envelope. By itself ENVELOPE does not generate sound; it only provides a definition for later use in the PLAY statement. (To understand the purpose of ENVELOPE, you must be familiar with the idea of envelopes and attack-decay-sustain-release phases of the envelope as discussed in the first part of this chapter.)

Predefined Instruments

The Commodore 128 actually has built into it the parameters for ten "standard" instruments, from calliope to xylophone. That is, the envelopes

for these instruments have been predefined and stored in the computer, so that you don't need to figure them out yourself. Table 7-2 shows these instruments; the attack, decay, sustain, and release (ADSR) settings; and the waveforms for each one.

The purpose of the ENVELOPE statement is to *change* these predefined envelopes to create new instruments with new sounds. This is done simply by executing the ENVELOPE statement containing the number of the envelope you wish to change, and the various parameters you want to substitute for the standard ones. Figure 7-7 shows the various parameters of the ENVELOPE statement.

The first parameter in this statement is the envelope number. If you want to change the standard flute, which is envelope number 4, for example, then you would use a 4 here.

The next four parameters of ENVELOPE specify the envelope itself. They are attack, decay, sustain, and release (these terms were defined earlier in this chapter). All these parameters can have values between 0 and 15. Note that while attack, decay, and release refer to times, sustain refers to the *volume* or amplitude of the note during its sustain phase. (The actual duration of the sustain phase is specified in the PLAY statement.)

How long attack, decay, or release will last when set to a particular value is shown in Table 7-3. (The abbreviation "ms" means milliseconds, or thousandths of a second.)

The final two parameters in ENVELOPE specify the waveform and (if the variable pulse waveform is selected) the pulse width. These two

Table 7-2. Predefined Instruments

Number	*Instrument*	*Attack*	*Decay*	*Sustain*	*Release*	*Waveform*	*Width*
0	Piano	0	9	0	0	2	1536
1	Accordian	12	0	12	0	1	
2	Calliope	0	0	25	0	0	
3	Drum	0	5	5	0	3	
4	Flute	9	4	4	0	0	
5	Guitar	0	9	2	1	1	
6	Harpsichord	0	9	0	0	2	512
7	Organ	0	9	9	0	2	2048
8	Trumpet	8	9	4	1	2	512
9	Xylophone	0	9	0	0	0	

parameters are the same as those used in the SOUND statement, with one addition: besides the four waveforms we could specify with SOUND, it is also possible to specify *ring modulation*.

In ring modulation, voice 3 is used as a sort of envelope to modulate voice 1. Try selecting a very low-frequency value for voice 3, and playing it at the same time as a normal note from voice 1. You should obtain a variety of "ringing" sounds, such as a bell makes. Experimentation will be helpful here.

As an example of using ENVELOPE, the following statement will redefine the trumpet envelope, lengthening the release phase so the sound dies away more slowly, and increasing the sustain volume from 4 to 6.

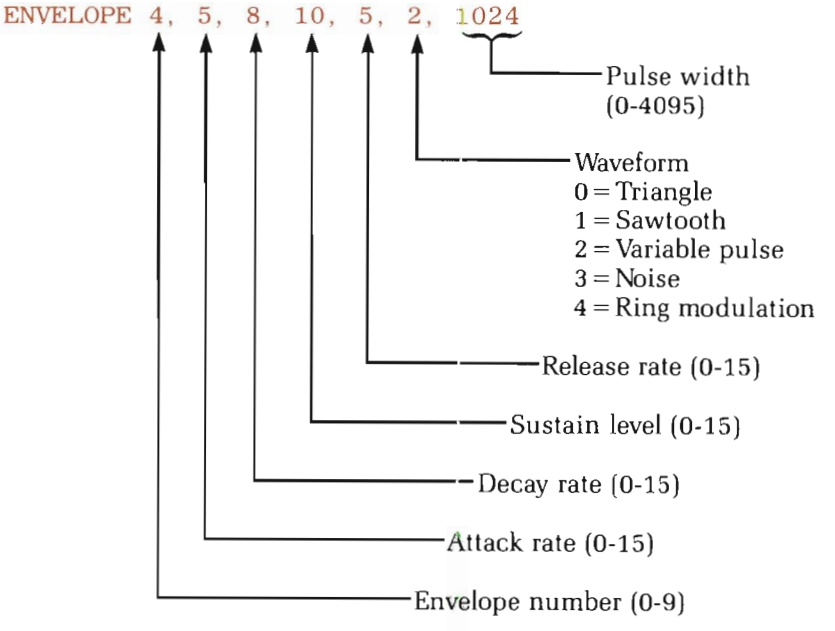
```
100 ENVELOPE 8, 8, 9, 6, 3, 2, 512
```

The TEMPO Statement

The TEMPO statement acts like the setting on a metronome. It specifies how long a measure will take, and thus the duration of whole notes, half notes, and so on. Here's TEMPO being used to specify a tempo of 40:

```
100 TEMPO 40
```

Figure 7-7. The ENVELOPE Statement



The parameter in the TEMPO statement can have any value between 0 and 255. At a setting of 0 the notes will sound continuously, one after the other. The default setting is 8. The higher the value of the TEMPO parameter, the shorter the notes will be. Thus a value of 6 will provide slow dance music, while a value of 200 will be useful for sequences of fast sounds in games.

Here's the formula relating the values used in TEMPO to the actual time in seconds that a measure takes:

duration (seconds) = 19.22 / tempo setting

Thus if you use the statement TEMPO 10, a measure will be about 2 seconds long; while if you specify TEMPO 80, then a measure will be about 1/4 of a second long.

The FILTER Statement

Earlier in this chapter we described the actions of various filters: high-pass, low-pass, and band-pass. The FILTER statement in BASIC 7.0 en-

Table 7-3. Attack, Decay, and Release Durations

Parameter Value	*Attack Duration*	*Decay and Release Durations*
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

ables us to turn on these filters in the SID chip, thus filtering the sound generated by our program before it reaches the outside world.

Here's a typical example of the FILTER statement:

```
100 FILTER 1000, 0, 0, 1, 10
```

The first parameter in the FILTER statement is the *cutoff frequency*, which can have a value from 0 to 2047. The next three parameters act as switches, and have a value of either 0 — meaning off, or 1 — meaning on. The first one is the switch for the low-pass filter, the second for the band-pass, and the third for the high-pass filter. Note that we can turn on several filters at the same time. The fifth and last parameter is the resonance, which can range in value from 0 to 15.

The statement above thus sets the cutoff frequency to 1000, turns off the low-and band-pass filters, turns on the high-pass filter, and sets the resonance at 10.

The PLAY Statement

The PLAY statement is an amazingly powerful statement. It not only plays sequences of musical notes, it also lets you set the envelope of the notes, play with multiple voices, and specify a filter, among other things.

Notes

The notes to be played by PLAY are specified by their letter values. These values are written in a string called the "play string." Thus the statement

```
100 PLAY "C D E C"
```

plays the first four notes of "Frere Jacques" in the key of C. The play string is everything in quotes following the PLAY. The spaces aren't really needed between the notes, but are good to include for clarity.

You can use sharps and flats as well. Here are the same four notes transposed into the key of D, with an F# note:

```
100 PLAY "D E #F D"
```

Notice that the sharp sign must *precede* the note, unlike the usual musical notation. A flat note is indicated by a dollar sign ($).

You can also play a rest, which is no sound for the same length of time ordinarily used by a particular note. The letter for this is R, and is used just like other notes.

Timing

The notes we've been playing so far are *whole notes*, which is the default note duration. However, you can specify any length note you wish. For example, the statement

```
100 PLAY "HA QB"
```

plays A as a half note and B as a quarter note. The list below shows how letters can be used to specify the different time values of the notes.

W = Whole note
H = Half note
Q = Quarter note
I = Eighth note
S = Sixteenth note
Period (.) = Dotted note (1½ times usual value)

Once you've used one of these letters to specify a time value for a note, all notes following will have the same time value until you change it by using a different letter.

As an example, the following PLAY statement will play the first two measures of "Mary Had a Little Lamb."

```
100 PLAY ".QE SD QC D E E HE"
```

Or, how about the first four measures of "Claire de Lune:"

```
100 PLAY "Q C C C D H E D Q C E D D H C"
```

The values given to individual notes are relative. That is, when you specify a quarter note, for example, you aren't specifying a particular period of time the note will take. To do this, you need to use the TEMPO statement. If you've used the TEMPO statement with a parameter of 20, for example, then a measure will be about one second long, and so a quarter note will be one quarter of this, or ¼ of a second. If your TEMPO parameter was 40, on the other hand, a measure will be ½ second long, so a quarter note will only take ⅛ of a second.

Here, for example, is a program which sets the volume level to the maximum, redefines the piano envelope, sets the tempo to 40, and plays

the notes of the scale as quarter notes, which will give them a duration of ⅛ of a second.

```
100  VOL 15
110  ENVELOPE 0, 2, 10, 10, 0, 2, 2047
120  TEMPO 40
130  PLAY "Q C D E F G A B"
```

Synthesizer Control

You can specify a variety of other characteristics of the sounds generated by a PLAY statement. Together these characteristics are referred to as synthesizer control. Using PLAY, you can specify what voice to use, what octave to play in, what envelope to use, the volume of the sound, and whether the filter is on or off. These attributes are all specified with single letters followed by a number. Because none of the letters is the same as the notes of the scale or the letters used for the notes' time values, BASIC doesn't get confused.

Table 7-4 shows the range of values possible for each of these characteristics.

Thus the control code V2 would turn on the second voice, O6 would specify the 6th octave, T7 would specify the seventh envelope, U4 would set the volume to 4, and X1 would turn the filter on. You can use some or all of these codes at any time in a play statement. However, if you use more than one, then you should put them in the order shown in the table; SID can process them faster that way.

The following program line selects voice 1 and envelope 5, sets the volume to 15, and plays all the notes of the scale as quarter notes:

```
100 PLAY "V1 T5 U15 Q C D E F G A B C"
```

Table 7-4. Single-letter Controls for the PLAY Statement

Control Character	*Characteristic*	*Range*	*Default Value*
V	Voice	1-3	1
O	Octave	0-6	4
T	Envelope	0-9	0
U	Volume	0-15	9
X	Filter	0 = off, 1 = on	0

Multiple Voices

To play multiple voices you start the first voice going by specifying what notes it is to play, then, using the V control code, you shift to the next voice and tell it what notes to start playing. Because BASIC is so fast, there is no discernable lag between when the first voice starts and when the next one starts. However, to keep the voices in synchronization, it's important to start longer notes before sequences of shorter notes. Thus if one voice will be playing two quarter notes at the same time a second voice will be playing one half note, you should start the half note first.

The basic unit of time in music is the measure, and this is a good interval to focus on when writing multiple-part music. Write the codes for one measure at a time, giving the codes for each voice in turn. For example, the line,

```
100 PLAY "V1 O4 W C   V2 Q O6 C O5 G E C"
```

starts the first voice off playing one whole note, C, then starts the second voice off playing four quarter notes, C, G, E, and C. The note played by the first voice is in the 4th octave. The first C played by the second voice is in the 6th octave, while the remaining notes are in the 5th octave.

With this background, you should have a good start on making music on your Commodore 128.

A

Addresses of Companies and Organizations

Chapter 1

Commodore Business Machines, 1200 Wilson Dr., West Chester, PA 19380

Digital Research, P.O. Box DRI, Monterey, CA 93942 (415) 649-3896

Chapter 2

Anchor Automation, 6913 Valjean Ave., Van Nuys, CA 91406 (818) 997-7758

Hayes Microcomputer Products, Inc., P.O. Box 105203, Atlanta, GA 30348 (404) 662-7100

Human Engineered Software, 390 Swift Ave. #14, South San Francisco, CA 94080 (415) 468-4111

MSD Systems, Inc., 10031 Monroe Drive, Suite 206, Dallas, TX 75229 (214) 357-4434

Chapter 3

CompuServe Information Service, 5000 Arlington Centre Blvd., Columbus, OH 43220 (614) 457-8600

COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403 (919) 275-9809

Contemporary Marketing, Inc., 1 Bala Avenue, Bala Cynwyd, PA 19004 (215) 667-3683

CW Communications/Peterborough, Inc., 80 Pine St., Peterborough, NH 03458 (603) 924-9471

Chapter 4

Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510 (616) 241-5510

Clockwork Computers, Inc., 4612 Holly Ridge Road, Rockville, MD 20853 (301) 924-5509

Commodore Business Machines, 1200 Wilson Dr., West Chester, PA 19380

Continental Software, 11223 South Hindry Ave., Los Angeles, CA 90045 (213) 410-3977

Davidson & Associates, Inc., 6069 Groveoak Place #12, Rancho Palos Verdes, CA 90274 (213) 373-9473

DesignWare, Inc., 185 Berry Street, San Francisco, CA 94107 (415) 546-1866

Handic Software, Inc., 520 Fellowship Road, Suite B-206, Mount Laurel, NJ 08054 (609) 866-1001

Infocom, 125 CambridgePark Dr., Cambridge, MA 02140 (617) 492-6000

Microprose Software, 120 Lakefront Dr., Haunt Valley, MD 21030 (301) 667-1151

Micro Technics Solutions Corp, P.O. Box 2940, New Haven, CT 06515 (203) 389-8383

Professional Software Inc., 51 Freemont Steet, Needham, MA 02194 (617) 444-5224

Sierra ON-LINE, Sierra On-Line Building, Coarsegold, CA 93614

Skyles Electric Works, 231E South Whisman Road, Mountain View, CA 94041 (415) 965-1735

subLOGIC Corporation, 713 Edgebrook Drive, Champaign, IL 61820 (217) 359-8482

Chapter 5

Adventure International, P.O. Box 3435, Longwood, FL 32750 (305) 862-6917

CompuServe Information Service, 5000 Arlington Centre Blvd., Columbus, OH 43220 (614) 457-8600

Computing!, 2519 Greenwich Street, San Francisco, CA 94123 (415) 567-1634

Control-C Software, 6441 Southwest Canyon Court, Portland, OR 97221 (503) 297-7153

CP/M Users Group (CP/MUG), 1651 Third Ave., New York, NY 10028 (212) 929-2326

Digital Research, P.O. Box DRI, Monterey, CA 93942 (415) 649-3896

Dow Jones & Co., Inc., Route 1 at Ridge Road, South Brunswick, NJ 08852 (609) 452-2000

First Osborne User Group (FOG), P.O. Box 3474, Daly City, CA 94015 (415) 755-4140

Hayes Microcomputer Products, Inc., P.O. Box 105203, Atlanta, GA 30348 (404) 662-7100

Kaypro, 533 Stevens Avenue, Solana Beach, CA 92075 (619) 481-4300

MicroPro International, 33 San Pablo Ave., San Rafael, CA 94903 (415) 499-1200

Microsoft Corporation, 10700 Northup Way, Box 97200, Bellevue, WA 98009 (206) 828-8080

Open Systems, 6477 City West Parkway, Eden Prairie, Minnesota 55344 (612) 829-0111

Osborne Computer Company, 42680 Christy, Fremont, CA 94538

Osborne/McGraw-Hill, 2600 Tenth Street, Berkeley, CA 94710 (415) 548-2805

PeachTree Software, 3445 Peachtree Road NE, Atlanta, GA 30326 (404) 239-3000

Howard W. Sams, 4300 West 62nd Street, Indianapolis, IN 46268 (317) 298-5400

Sorcim/IUS, 2310 Lundy Ave, San Jose, CA 95131 (408) 942-1727

Sybex, 2344 Sixth Street, Berkeley, CA 94710 (415) 848-8233

The Source, Reader's Digest Association, Inc., 200 Park Ave,, New York, New York 10166 (212) 953-0030

Chapter 7

DesignWare, 185 Berry Street, San Francisco, CA 94107 (415) 546-1866

Electronic Arts, 2755 Campus Drive, San Mateo, CA 94403 (415) 571-7171

Electronic Lab Industries, P.O Box 7167, Baltimore, MD 21218 (301) 366-8138

En-Tech Software, P.O. Box 881, Sun Valley, CA 91353 (818) 768-6646

Melodian, Inc., 115 Broadway, Suite 1202, New York, NY 10006 (212) 406-5163

Passport Designs, Inc., 625 Miramonte Street, Half Moon Bay, CA 94019 (415) 726-0280

QuickSilva, Inc., 14307 Benbrush, San Antonio, TX 78248 (512) 366-5514

Sequential Circuits, Inc., 3051 North First Street, San Jose, CA 95134 (408) 946-5240

Sight and Sound Software, Inc., 3200 South 166th Street, New Berlin, WI 53151 (414) 784-5850

Waveform Corporation, 418 N. Buchanan Circle, #12, Pacheco, CA 94553 (415) 825-1722

Index

☐ COMMODORE 64 PROGRAMMER'S REFERENCE GUIDE

A Top 10 best-seller since its introduction, this programmer's working tool and reference source is packed with professional tips and information on exploring your C-64. Includes a complete, detailed dictionary of all Commodore BASIC commands, statements, and functions. BASIC program samples then show you how each item works. Mix machine language with BASIC and use hi-res effectively with this easy-to-use guide. Commodore Computer.

ISBN 0-672-22056-3 . . . . . **$19.95**

☐ COMMODORE 128 PROGRAMMER'S REFERENCE GUIDE

Here's the key to unlocking the advanced features of the Commodore 128. This excellent reference guide shows you how to master BASIC programming, machine language programming, the graphics system, sound system, and much more. Also includes hardware details such as chip specifications, input/output parts and lines. Commodore Computer.

ISBN 0-672-22479-8 . . . . . **$22.95**

☐ CP/M® PRIMER (2nd Edition)

This tutorial companion to the *CP/M Bible* is highly acclaimed and widely used by novices and advanced programmers alike. Includes the details of CP/M terminology, operation, capabilities, internal structure, plus a convenient tear-out reference card with CP/M commands. This revised edition allows you to begin using new or old CP/M versions immediately in any application. Waite and Murtha.

ISBN 0-672-22170-5 . . . . . **$16.95**

☐ SOUL OF CP/M: HOW TO USE THE HIDDEN POWER OF YOUR CP/M SYSTEM

Recommended for those who have read the *CP/M Primer* or who are otherwise familiar with CP/M's outer layer utilities. This companion volume teaches you how to use and modify CP/M's internal features, including how to modify BIOS and use CP/M system calls in your own programs. Waite and Lafore.

ISBN 0-672-22030-X **$19.95**

☐ CP/M BIBLE: THE AUTHORITATIVE REFERENCE GUIDE TO CP/M

Already a classic, this highly detailed reference manual puts CP/M's commands and syntax at your fingertips. Instant one-stop access to all CP/M keywords, commands, utilities, and conventions are found in this easy-to-use format. If you use CP/M, you need this book. Waite and Angermeyer.

ISBN 0-672-22015-6 . . . . . **$19.95**

☐ COMPUTER GRAPHICS PRIMER

Graphics communicate quickly. Gain this essential skill, and apply it in your own programs. Program examples are presented in Applesoft BASIC, and can be translated to other BASIC language dialects. Save 1000 words. Use graphics and animation to enhance your programs. Mitchell Waite.

ISBN 0-672-21650-7 . . . . . **$15.95**

☐ ARTIFICIAL INTELLIGENCE PRIMER

This AI Primer discusses all of the major concepts of the AI expert systems field. The book is presented at an introductory level for those with little or no previous experience with the subject. The question and answer format, as well as the glossary, provide the beginner with a thorough understanding of AI. Louis Frenzel.

ISBN 0-672-22442-9 . . . . . **$17.95**

☐ PRINTER CONNECTION BIBLE

At last! A book that teaches non-technical people how to connect a computer to a printer. Covers major computer/printer combinations, and supplies detailed diagrams of required cables, dip-switching settings, etc. The book is graphically oriented with the diagrams illustrating numerous printer/computer combinations. House and Marble.

ISBN 0-672-22406-2 . . . . . **$16.95**

☐ COMMODORE 64/128 ASSEMBLY LANGUAGE PROGRAMMING

A complete introduction to assembler language programming for Commodore owners. Contains a wealth of assembly language routines to use for creating music and sound programs, sprite graphics, and designing your own character set. Written in English, not "computerese", an excellent book for beginner and intermediate level programmers. Mark Andrews.

ISBN 0-672-22444-5 . . . . . **$14.95**

☐ COMMODORE 64 GRAPHICS AND SOUNDS

Quickly learn to exploit the powerful graphic and sound capabilities of the C-64. Create your own spectacular routines utilizing graphics and sounds instantly. Loaded with sample programs, detailed illustrations, and thorough explanations covering bit-mapped graphics, three-voice music, sprites, sound effects and multiple graphics combinations. Save even more time on your way to having fun with graphics and sounds . . . buy the Combo Pack, and save your fingers. Timothy Orr Knight.
Book:
ISBN 0-672-22278-7 . . . . . **$8.95**
Combo Pack
ISBN 0-672-26186-3 . . . . . **$19.95**

☐ COMMODORE 64 TROUBLESHOOTING AND REPAIR GUIDE

Is your Commodore 64 on the fritz? It may be something you can repair yourself simply and inexpensively. Troubleshooting flowcharts allow you to diagnose and remedy the probable cause of failure, and a final chapter on advanced troubleshooting shows the more adventuresome how to perform more complicated repairs. Some knowledge of electronics required. Robert C. Brenner.
ISBN 0-672-22363-5 . . . . . **$18.95**

☐ EXPERIMENTS IN ARTIFICIAL INTELLIGENCE FOR SMALL COMPUTERS

Can a computer really think? Decide for yourself as you conduct interesting and exciting experiments in artificial intelligence. Duplicate such human functions as reasoning, creativity, problem solving, verbal communication, and game planning. Sample programs furnished. John Krutch.
ISBN 0-672-21785-6 . . . . . **$9.95**

☐ COMPUTER GRAPHICS USER'S GUIDE

This is your idea book for using computer-generated high-res imagery for fun and profit. Subjects include basic geometery, fundamental computing, turning your ideas into pictures, and ways to transfer these pictures from the computer to video tape or film. Contains beautiful, full-color photographs. Andrew S. Glassner.
ISBN 0-672-22064-4 . . . . . **$19.95**

☐ INTRODUCTION TO ELECTRONIC SPEECH SYNTHESIS

Why do computers talk funny? This book helps you understand how a human "voice" is electronically created, explains three digital synthesis technologies, and relates speech quality, data rate, and memory devices. Neil Sclater.
ISBN 0-672-21896-8 . . . . . **$9.95**

☐ ELECTRONICALLY HEARING: COMPUTER SPEECH RECOGNITION

The human ability to interpret and understand voice communication is not easily duplicated in computers. This book brings you up-to-date on the lastest developments in the field and covers the practical aspects of computer speech analysis and recognition. Necessary math and speech concepts are included where appropriate. John P. Cater.
ISBN 0-672-22173-X **$13.95**

☐ ELECTRONICALLY SPEAKING: COMPUTER SPEECH GENERATION

Interest in digitized speech is rapidly expanding. Learn the basics of generating synthetic speech with an Apple II, TRS-80, or other popular microcomputer. Also includes a history of synthetic speech research since the 1800's. John P. Cater.
ISBN 0-672-21947-6 . . . . . **$14.95**

☐ USING COMPUTER INFORMATION SERVICES

There's a world of information out there waiting for you to use. Learn to use your microcomputer to communicate with the national computer networks and their wide range of over the network services. Make your computer a powerful communications tool. Sturtz and Williams.
ISBN 0-672-21997-2 . . . . . **$12.95**

☐ THE LOCAL AREA NETWORK BOOK

Defines and discusses localized computer networks as a versatile means of communication. You'll learn how networks developed and what local networks can do; what's necessary in components, techniques, standards, and protocols; how some LAN products work and how real LANs operate; and how to plan a network from scratch. E. G. Brooner.
ISBN 0-672-22254-X **$7.95**

More Excellent Books in the Sams/Waite Primer Series

BASIC PROGRAMMING PRIMER (2nd Edition)
Gives you fundamental BASIC keywords, statements, and functions usable with the IBM® PC, Apple® II, or any other computer running a variation of Microsoft BASIC. Also covers advanced BASIC, new game program listings, more.
Waite and Pardee
No. 22014 ..... **$17.95**

C™ PRIMER PLUS
A clear, complete introduction to the C language that guides you in the fundamentals, covers use of Microcomputer C with assembly language, and contains many sample programs usable with any standard C compiler.
Waite, Prata, and Martin
No. 22090 ..... **$19.95**

COMPUTER GRAPHICS PRIMER
A classic Sams best seller that helps you learn all types of graphics programming, including animation. Many program examples written in Applesoft.
Mitchell Waite
No. 21650 ..... **$15.95**

CP/M BIBLE: THE AUTHORITATIVE REFERENCE GUIDE TO CP/M
Gives you instant, one-stop access to all CP/M keywords, commands, utilities, conventions, and more. A must for any computerist using any version of CP/M.
Waite and Angermeyer
No. 22015 ..... **$19.95**

CP/M PRIMER (2nd Edition)
Completely updated to give you the know-how to begin working with new or old CP/M versions immediately. Includes CP/M terminology, operation, capabilities, internal structure, and more.
Waite and Murtha
No. 22170 ..... **$16.95**

MICROCOMPUTER PRIMER (2nd Edition)
Shows you **basic** computer concepts, the electronics behind the logic, what happens inside the computer as a program runs, and a little about languages and operating systems.
Waite and Pardee
No. 21653 ..... **$14.50**

SOUL OF CP/M® : HOW TO USE THE HIDDEN POWER OF YOUR CP/M SYSTEM
Teaches you how to use and modify CP/M's internal features, use CP/M system calls, and more. You'll need to read *CP/M PRIMER*, or be otherwise familiar with CP/M's outer-layer utilities.
Waite and Lafore
No. 22030 ..... **$19.95**

SUPERCALC PRIMER
Makes it much easier for you to find your way out of a sticky calculation, hit the right keys to get a given answer, or otherwise get the most out of your SuperCalc spreadsheet program. Fully illustrated.
Waite, Burns, and Venit
No. 22087 ..... **$16.95**

PASCAL PRIMER
Guides you swiftly through Pascal program structure, procedures, variables, decision making statements, and numeric functions. Contains many useful examples and eight appendices.
Waite and Fox
No. 21793 ..... **$17.95**

TIMEX SINCLAIR BASIC PRIMER, WITH GRAPHICS
Easiest, most effective introduction ever to computers and programming. Graphics let you "see" your commands work as your own programs gradually develop into realities.
Waite and Chapnick
No. 22077 ..... **$9.95**

UNIX™ PRIMER PLUS
Presents the elements of UNIX clearly, simply, and accurately, for ready understanding by anyone in any field who needs to learn, use, or work with UNIX in some way. Fully illustrated.
Waite, Martin, and Prata
No. 22028 ..... **$19.95**

YOUR OWN COMPUTER (2nd Edition)
Shows how to choose a computer for your own uses and explores applications, buzzwords, programs, hardware, and peripherals for 30 different models.
Waite and Pardee
No. 21860 ..... **$8.95**

SAMS BOOK & SOFTWARE ORDER CARD

Catalog No.	Qty.	Price	Total	Catalog No.	Qty.	Price	Total

Subtotal
AR,CA,FL,IN,NC, NY,OH,TN,WV residents add local sales tax
Add Handling charge 2.00
Total Amount Enclosed

☐ Check ☐ Money Order
☐ MasterCard ☐ Visa

Account Number _____ Expires _____

Name (print) _____

Signature _____

Address _____

City _____ State _____ Zip _____

Offer good in USA only. Prices subject to change without notice. Full payment must accompany your order

WC004

To locate the Sams retailer or distributor nearest you, call 1-800-428-SAMS (residents in IN, AK, HI call 317-298-5566).

If your retailer or distributor doesn't stock the Sams publication you need, you can order directly from SAMS. Orders placed directly with SAMS are subject to a $2.00 additional handling charge per order.

PHONE ORDERS

You may order by phone by calling either number listed above to charge your order to your VISA or MASTERCARD.

MAIL ORDERS

Use the order form below or send your order on a plain piece of paper. Be sure to:

(1) Include your name, address, city, state, and zip.

(2) The titles of the books you need, the product numbers (see other side of this card) and the quanity of each one you'd like.

(3) Add the total cost for the books, add local sales tax. Add $2.00 for handling.

(4) Include your check or money order for the full amount due, or

(5) Charge your VISA or MASTERCARD. Charge orders must include the account number, card expiration date and your signature. Shipping charges will be added to credit card orders.

(6) Mail your order to:
Howard W. Sams & Co., Inc.
Dept DM
P.O. Box 7092
Indianapolis, Indiana 46206

In Canada, contact Lenbrook Electronics, Markham, Ontario L3E 1H2

All books available from Sams Distributors, Bookstores, and Computer Stores. Offer good in U.S. only. Note Distributor Computer Store and Dealer inquiries are welcome.

See other side for list of Sams/Waite Primers and their prices.

Place Stamp Here

HOWARD W. SAMS & CO., INC.
P.O. BOX 7092
INDIANAPOLIS, INDIANA 46206

SAMS

The Waite Group

The Official Book for the Commodore 128™ Personal Computer

Tap the power of Commodore's most exciting computer, the Commodore 128. With its three completely different operating modes—64™, 128, and CP/M®, it's really three computers in one.

In this, the only official guide to the Commodore 128, you'll find how you can use these different modes in various applications. You'll learn how to:

- Create exciting, detailed graphics—up to 640×200 point, 16 color screens
- Use the 64 mode to run thousands of existing Commodore 64 programs
- Program in three-voice sound
- Use the world's largest base of professional-level business software in the CP/M mode
- Achieve faster disk access in the 128 mode
- Choose among the new peripherals
- Use spreadsheets, word processors, data bases, and much more!

Even though the Commodore 128 is designed primarily as a business computer, this book will show you how the computer can be used at home as well. *The Official Book For The Commodore 128 Personal Computer* is easy to read and perfect for first-time users or seasoned Commodore 64 owners.

Mitchell Waite is president of The Waite Group, Inc., a California company specializing in book writing and documentation for the computer marketplace. He is an experienced programmer, fluent in a variety of computer languages and has studied nuclear engineering, built biofeedback machines, written poetry and raced motorcycles. Mr. Waite is the co-author of *UNIX™ Primer Plus, C Primer Plus, CP/M Primer, Soul of CP/M*, published by Sams.

Robert Lafore is editorial director of The Waite Group, Inc. Mr. Lafore has been in the computer industry since 1965, and is co-author of *Soul of CP/M*. He has worked as a petroleum engineer in South East Asia, as a newspaper columnist, and has operated his own software company. He holds degrees in mathematics and electrical engineering.

Jerry Volpe is a managing editor of The Waite Group, Inc., and was formerly a technical education course writer for The Heath Company. Mr. Volpe is experienced in digital electronics and cryptographic communications, and has been a Commodore computer user since 1982. He holds a degree in occupational education.

Howard W. Sams & Co., Inc.
A Subsidiary of Macmillan, Inc.
4300 West 62nd Street, Indianapolis, IN 46268 U.S.A.

0 81262 22456 4

$12.95/22456

ISBN: 0-672-22456-9